Supercomputing Frontiers and Innovations

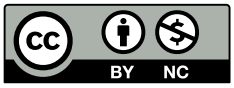
2015, Vol. 2, No. 2

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents



Special Issue on "Sustainable Ultrascale Computing Systems"

The ever-increasing presence of digital data and computers requires pushing for new levels of scalability and sustainability of computing systems in order to address the huge data and processing requirements of the near future. Systems need are expected to be two to three orders of magnitude larger than todays systems, including systems with unprecedented amounts of heterogeneous hardware, lines of source code, numbers of users, and volumes of data, sustainability is critical to ensure the feasibility of those systems.

Achieving that scale is a major problem today, given the existing limitations in scale, software, power, etc. Exascale systems, massive parallel systems that will execute 1 ExaFLOP/second, were announced by 2020, but the different programs in USA and Japan are now delaying that goal to well advanced the 20s. However, even if Exascale systems are developed in a near future, they may not be valid for all kind of problems. Due to those needs, currently there is an emerging cross-domain interaction between high-performance in clouds and the adoption of distributed programming paradigms, in scientific applications.

Ultrascale computing systems (UCS), can be a solution. Envisioned as large-scale complex systems joining parallel and distributed computing systems, which can be located at multiple sites and cooperate to provide the required resources and performance to the users, they could extend individual systems to provide the resources needed. However, the cooperation between HPC and distributed system communities still poses many challenges towards building the Ultrascale systems of the future, especially in unifying the services to deploy sustainable applications portable to HPC systems, multi-clouds, data centers, and big data.

The scope of this special issue is to present important aspects towards making more sustainable Ultrascale systems following the topics addressed in the COST Action IC 1305 Network for Sustainable Ultrascale Computing (NESUS), which main goal is to establish an open European research network targeting realistic and sustainable solutions for ultrascale computing. Six papers have been selected from submissions received, which were reviewed by qualified anonymous referees according to the practices of this Journal.

In the paper Exascale machines require new programming paradigms and runtimes, the authors study the need of providing new programming paradigms that can allow the applications programmers to express the complexity and models of Ultrascale applications. They also study the need of creating new runtimes able to cope with scale and heterogeneity foreseen in future systems. In Acceleration of MPI mechanisms for sustainable HPC applications, several optimizations for the MPI runtime are proposed to enhance it with aspects like hierarchical programming, malleability, energy savings, dynamic data management, etc.

The paper Resilience within Ultrascale Computing System: Challenges and Opportunities from Nesus Project, presents challenges and opportunities for ultrascale systems based on current activities of the European COST Action Nesus (IC1305), which has a working group devoted to resiliency aspects.

The last three papers of the special issue are related to energy efficiency issues, including two surveys and a position paper from the NESUS project. In the paper Energy Measurement Tools for Ultrascale Computing: A Survey, the authors discuss the problem of measuring energy in a very large scale system and make a survey of existing techniques that can be used to cope with this problem. The paper Energy-efficient Algorithms for Ultrascale Systems is a survey of the current research efforts and results related to energy efficiency in the diverse areas of

software, discussing open problems and questions concerning energy-related techniques with an emphasis on the application algorithmic side. Finally, in the paper, Energy Efficiency for Ultrascale Systems: Challenges and Trends from Nesus presents challenges and trends associated with energy efficiency for ultrascale systems based on current activities of the working group on Energy Efficiency in the European COST Action Nesus IC1305. The analysis contains major areas that are related to studies of energy efficiency in ultrascale systems: heterogeneous and low power hardware architectures, power monitoring at large scale, modeling and simulation of ultrascale systems, energy-aware scheduling and resource management, and energy-efficient application design.

We would like to thank all authors who submitted papers, including those whose papers were not selected for this Special Issue. A special note of thanks goes to all of the reviewers for their cooperation. Without them, this issue would not be possible.

Guest editor:

Jesus Carretero

DOI: 10.14529/jsfi150201

Exascale Machines Require New Programming Paradigms and Runtimes

***Georges Da Costa*[1], *Thomas Fahringer*[2], *Juan-Antonio Rico-Gallego*[3], *Ivan Grasso*[2], *Atanas Hristov*[4], *Helen D. Karatza*[5], *Alexey Lastovetsky*[6], *Fabrizio Marozzo*[8], *Dana Petcu*[7], *Georgios L. Stavrinides*[5], *Domenico Talia*[8], *Paolo Trunfio*[8], *Hrachya Astsatryan*[9]**

Extreme scale parallel computing systems will have tens of thousands of optionally accelerator-equipped nodes with hundreds of cores each, as well as deep memory hierarchies and complex interconnect topologies. Such exascale systems will provide hardware parallelism at multiple levels and will be energy constrained. Their extreme scale and the rapidly deteriorating reliability of their hardware components means that exascale systems will exhibit low mean-time-between-failure values. Furthermore, existing programming models already require heroic programming and optimization efforts to achieve high efficiency on current supercomputers. Invariably, these efforts are platform-specific and non-portable. In this article, we explore the shortcomings of existing programming models and runtimes for large-scale computing systems. We propose and discuss important features of programming paradigms and runtimes to deal with exascale computing systems with a special focus on data-intensive applications and resilience. Finally, we discuss code sustainability issues and propose several software metrics that are of paramount importance for code development for ultrascale computing systems.

Keywords: programming models, ultrascale, runtimes, extreme scale.

Introduction

Ultrascale systems are envisioned as large-scale complex systems joining parallel and distributed computing systems that will be two to three orders of magnitude larger than today's systems reaching millions to billions elements. New programming models and runtimes will be necessary to use efficiently these new infrastructures. To achieve results on this front, the European Union funded the *COST Action Nesus IC1305* [72]. Its goal is to establish an open European research network targeting sustainable solutions for ultrascale computing, aiming at cross fertilization among HPC, large-scale distributed systems and big data management. The network contributes to gluing together disparate researchers working across different areas and provides them with a meeting ground to exchange ideas, identify synergies and pursue common activities in research topics, such as sustainable software solutions (applications and system software stack), data management, energy efficiency and resilience.

One key element on the ultrascale front is the necessity of new sustainable programming and execution models in the context of rapid underlying computing architecture changing. There is a need to explore synergies among emerging programming models and runtimes from HPC, distributed systems and big data management communities. To improve the programmability of

[1]University of Toulouse, Toulouse, France
[2]University of Innsbruck, Innsbruck, Austria
[3]University of Extremadura, Badajoz, Spain
[4]UIST, Orhid, Republic of Macedonia
[5]Aristotle University of Thessaloniki, Thessaloniki, Greece
[6]University College Dublin, Dublin, Ireland
[7]West University of Timisoara, Timisoara, Romania
[8]University of Calabria, Rende CS, Italy
[9]National Academy of Sciences of Armenia, Yerevan, Armenia

future systems, the main changing factor will be the substantially higher levels of concurrency, asynchrony, failures and heterogeneous architectures.

At different levels, teams of scientists are tackling this challenge. The goal of the European Cost Action Nesus is to establish an open European research network. One of the key elements this action will target is the proposal and implementation of novel programming paradigms and runtimes in order to make the ultrascale a reality. On the other hand, the main objective of the *European Exascale Software Initiative* [73] is to provide recommendations on strategic European actions with a particular focus on software key issues improvement, cross cutting issues advances and gap analysis. The common goal of these two different actions, is to provide a coherent agenda for research, but also establish a code and regulation at a European level, in order to reach ultrascale computing.

At the international level, the goal of the *International Exascale Software Project* [74] is also to provide an international common vision for cooperation in order to reach ultrascale: "The guiding purpose of the IESP is to empower ultra-high resolution and data-intensive science and engineering research through the year 2020, by developing a plan for: (a) a common, high-quality computational environment for petascale/exascale systems and (b) catalyzing, coordinating and sustaining the effort of the international open source software community to create that environment as quickly as possible".

This article explores programming models and runtimes required to facilitate the task of scaling and extracting performance on continuously evolving platforms, while providing resilience and fault-tolerant mechanisms to tackle the increasing probability of failures throughout the whole software stack. However, currently no programming solution exists that satisfies all these requirements. Therefore, new programming models and languages are required towards this direction. The whole point of view on application will have to change. As we will show, the current wall between runtime and application models leads to most of these problem. Programmers will need new tools but also new way to assess their programs. As we will see, data will be a key concept around which failure-tolerant high number of micro-threads will be generated using high-level information by adaptive runtime.

This article is structured as follows: the next section describes the requirements from the programmability point of view for extra large-scale systems such as ultrascale systems. The second section describes how data shift the paradigm of processor-centric management toward a data-centric one in next generation systems. The third section describes how resilience will be of critical importance, since faults and reconfiguration will be a recurring element in such a large-scale infrastructure. The fourth section presents the elements required to reach sustainable software development, whereas the final section concludes this article.

1. Improved programmability for extra large-scale systems

Supercomputers have become an essential tool in numerous research areas. Enabling future advances in science requires the development of efficient parallel applications, which are able to meet the computational demands. The modern high-performance computing systems (HPCS) are composed of hundreds of thousand computational nodes. Due to the rapidly increasing scale of those systems, programmers cannot have a complete view of the system. The programmability strongly determines the overall performance of a high performance computing system. It is a substrate over which processors, memory and I/O devices are exchanging data and instructions. It should have high scalability, which will support the development of the next generation exas-

cale supercomputers. Programmers also need to have an abstraction that allows them to manage hundreds of millions to billions of concurrent threads. Abstraction allows organizing programs into comprehensible fragments, which is very important for clarity, maintenance and scalability of the system. It also allows increasing of programmability by defining new languages on top of the existing language and, by defining completely new parallel programming languages. This makes abstraction an important part of most parallel paradigms and runtimes. Formerly, computer architectures were designed primarily for improved performance or for energy efficiency. In future exascale architectures, one of the top challenges will be enabling a programmable environment for the next generation architectures. In reality, programmability is a metric, which is really difficult to define and measure. The next generation architectures should minimize the chances of parallel computational errors while relieving the programmer from managing low-level tasks.

In order to explore this situation more precisely, one aim of this research is to investigate the limitations of current programming models along-with evaluations of promises of hybrid programming model to solve these scaling-related difficulties.

1.1. Limitations of the current programming models

Reaching exascale in terms of computing nodes requires the transition from current control of thousands of threads to billions of threads as well as the adaptation of the performance models to cope with an increased level of failures. One simple model that is used to program at any scale is a utopian idea as proved in the last twenty years of 'standardized' parallel computing. Unfortunately the exascale has become the reason for improving programming systems, as existing programming models implementations, more than a reason for a change in the programming models. This approach was classified by Gropp and Snir in [2] as evolutionary. According to their view, the five most important characteristics of the programming models that are affected by the exascale transition are: the thread scheduling, the communications, the synchronization, the data distribution and the control views.

1.1.1. Limitations of message passing programming model

The current vision on exascale system is at the moment to exploit distributed memory parallelism, and therefore the message passing model is likely to be used at least partially. Moreover the most popular system implementation of the model, MPI, has been shown to run with millions of cores for particular problems. MPI is based upon standard sequential programming languages, augmented with low-level message passing constructs, forcing users to deal with all aspects of parallelization, including the distribution of data and work to cores, communication and synchronization. MPI primarily favors static data distribution and is consequently not well suited to deal with dynamic load balancing.

However, it has been shown that the all-to-all communication algorithms used in the message passing models are not scalable (most commonly used implementations often assume a fully connected network and have dense communication patterns) while all-to-some, one-sided or sparse communication patterns are more reliable.

Furthermore, parallel I/O is a limiting factor in the MPI systems, showing that the current MPI-IO model should be reconsidered. In particular the limitations are related to the collective access to the I/O request and the data partitioning.

1.1.2. Limitations of shared-memory programming models

The exascale system is expected to handle hundreds of cores in the one CPU or GPU. Using shared-memory systems is a feasible alternative to message passing in the case of medium size parallel systems in order to reduce the programming overhead as is moving the parallelization burden from the programmer to the compiler.

The most popular shared-memory system, OpenMP, is following a parallelism control model that does not allow the control of data distribution and uses non-scalable synchronization mechanism like locks or atomic sections. Moreover, the global view of data leads easily to non-efficient programming as encouraging synchronization joins of all threads' remote data accesses similar to the local ones.

The emerging Partitioned Global Address Space model (PGAS) is trying to overcome the scalability problems of the global shared-memory model [3]. The PGAS model is likely to have benefit where non-global communication patterns can be implemented with minimal synchronization and overlap of computation and communication. Moreover, the scalability of I/O mechanisms in PGAS depends only on the scalability of the underlying I/O infrastructure and is not limited by the model. However, the scalability is limited to thousands of cores (with the exception of X10 which is implementing an asynchronous PGAS model). The load balancing is still an open issue for the systems that implement the model. Furthermore, it is not possible yet to sub-structure threads into subgroups.

1.1.3. Limitations of heterogeneous programming

Clusters of heterogeneous nodes composed of multi-core CPUs and GPUs are increasingly being used for High Performance Computing due to the benefit in peak performance and energy efficiency. In order to fully harvest the computational capabilities of such architectures, application developers often employ a combination of different parallel programming paradigms (e.g. OpenCL, CUDA, MPI and OpenMP). However, heterogeneous computing also poses the new challenge of how to handle the diversity of execution environment and programming models. The Open Computing Language [60] introduces an open standard for general-purpose parallel programming of heterogeneous systems. An OpenCL program may target any OpenCL-compliant device and today many vendors provide an implementation of the OpenCL standard. An OpenCL program comprises a host program and a set of kernels intended to run on a compute device. It also includes a language for kernel programming, and an API for transferring data between host and device memory and for executing kernels.

Single node hardware design is shifting to a heterogeneous nature. At the same time many of today's largest HPC systems are clusters that combine heterogeneous compute device architectures. Although OpenCL has been designed to work with multiple devices, it only considers local devices available on a single machine. However, the host-device semantics can be potentially applied to remote, distributed devices accessible on different compute nodes. Porting single-node multi-device applications to clusters that combine heterogeneous compute device architectures is not straightforward and in addition it requires the use of a communication layer for data exchange between nodes. Writing programs for such platforms is error prone and tedious. Therefore, new abstractions, programming models and tools are required to deal with these problems.

1.2. Exascale promise of the hybrid programming model

Using a message passing model for the inter-node parallelism and a shared-memory programming model for intra-node parallelism is nowadays seen as a promising path to reach the exascale. The hybrid model is referred as MPI+X, where X represents the programming model that supports threads. The most common X is OpenMP, while there are options for X, like OpenACC.

However, restrictions on the MPI+X model are still in place, for example how MPI can be used in a multi-threaded process. In particular, threads cannot be individually identified as the source or target of MPI messages, or an MPI barrier synchronize the execution of the processes but does not guarantee their synchronization in terms of memory views. The proposal to use MPI Endpoints in all-to-all communications from [4] is a step forward in order to facilitate high performance communication between multi-threaded processes.

Furthermore, combining different programming styles like message passing with shared memory programming lends itself to information hiding between different layers that may be important for optimization. Different runtime systems involved with these programming models lack a global view that can have a severe impact on the overall performance.

However, the biggest problem of MPI+X is the competition for the resources like bandwidth (accessing memory via the inter-node interconnect) [2]. Furthermore, an important obstacle is the memory-footprint and efficient memory usage, as the available memory per core or node is not expected to scale linearly with the number of cores and nodes, and the MPI+X functionality must cope with the expected decrease of space per core or node.

Multitasking is a mean to increase the ability to deal with fluctuations in execution of the threads due to the fault handling or power management strategies. PGAS+multitasking is providing a programming model analogous with MPI+X.

In exascale system storage and communication hierarchies will be deeper than the current ones. Therefore it is expected that the hierarchical programming models should replace the current two level ones [1].

The one-side communication model enables programming in a shared-memory-like programming style. In MPI it is based on the concept of a communication window to which the MPI processes in a communicator statically attach contiguous segments of their local memory for exposure to other processes; the access to the window is granted by synchronization operations. The model separates the communication operations and synchronization for data consistency, allowing the programmer to delay and schedule the actual communications. However the model is criticized for being difficult to be used efficiently.

1.2.1. Innovative programming for heterogeneous computing systems

In recent years, heterogeneous systems have received a great amount of attention from the research community. Although several projects have been recently proposed to facilitate the programming of clusters with heterogeneous nodes [54–59, 68, 69], none of them combines support for high performance inter-node data transfer, support for a wide number of different devices and a simplified programming model.

Kim et al. [56] proposed the *SnuCL* framework that extends the original OpenCL semantics to heterogeneous cluster environments. SnuCL relies on the OpenCL language with few extensions to directly support collective patterns of MPI. Indeed, in SnuCL the programmer is

responsible to take care of the efficient data transfers between nodes. In that sense, end users of the SnuCL platform need to have an understanding of MPI collective calls semantics in order to be able to write scalable programs.

Also other works have investigated the problem of extending the OpenCL semantics to access a cluster of nodes. The Many GPUs Package (*MGP*) [69] is a library and runtime system that using the MOSIX VCL layer enables unmodified OpenCL applications to be executed on clusters. *Hybrid OpenCL* [68] is based on the FOXC OpenCL runtime and extends it with a network layer that allows the access to devices in a distributed system. The *clOpenCL* [59] platform comprises a wrapper library and a set of user-level daemons. Every call to an OpenCL primitive is intercepted by the wrapper which redirects its execution to a specific daemon at a cluster node or to the local runtime. *dOpenCL* [55] extends the OpenCL standard, such that arbitrary compute devices installed on any node of a distributed system can be used together within a single application. *Distributed OpenCL* [54] is a framework that allows the distribution of computing processes to many resources connected via network using JSON RPC as a communication layer. *OpenCL Remote* [58] is a framework which extends both OpenCL's platform model and memory model with a network client-server paradigm. *Virtual OpenCL* [57], based on the OpenCL programming model, exposes physical GPUs as decoupled virtual resources that can be transparently managed independent of the application execution.

An innovative approach to program clusters of nodes composed of multi-core CPUs and GPUs has been introduced through *libWater* [71], a library-based extension of the OpenCL programming paradigm that simplifies the development of applications for distributed heterogeneous architectures.

libWater aims to improve both productivity and implementation efficiency when parallelizing an application targeting a heterogeneous platform by achieving two design goals: (*i*) transparent abstraction of the underlying distributed architecture, such that devices belonging to a remote node are accessible like a local device; (*ii*) access to performance-related details since it supports the OpenCL kernel logic. The *libWater* programming model extends the OpenCL standard by replacing the host code with a simplified interface. *libWater* also comes with a novel device query language (DQL) for OpenCL device management and discovery. A lightweight distributed runtime environment has been developed which dispatches the work between remote devices, based on asynchronous execution of both communications and OpenCL commands. *libWater* runtime also collects and arranges dependencies between commands in the form of a powerful representation called *command DAG*. The *command DAG* can be effectively exploited to improve the scalability. For this purpose a collective communication pattern recognition analysis and optimization has been introduced that matches multiple single point-to-point data transfers and dynamically replaces them with a more efficient collective operation (e.g. *scatter*, *gather* and *broadcast*) supported by MPI.

Besides OpenCL-based approaches, also CUDA solutions have been proposed to simplify distributed systems programming. *CUDASA* [66] is an extension of the CUDA programming language which extends parallelism to multi-GPU systems and GPU-cluster environments. *rCUDA* [61] is a distributed implementation of the CUDA API that enables shared remote GPGPU in HPC clusters. *cudaMPI* [62] is a message passing library for distributed-memory GPU clusters that extends the MPI interface to work with data stored on the GPU using

the CUDA programming interface. All of these approaches are limited to devices that support CUDA, i.e. NVidia GPU accelerators, and therefore they cannot be used to address heterogeneous systems which combines CPUs and accelerators from different vendors.

Other projects have investigated how to simplify the OpenCL programming interface. Sun et. al [65], proposed a task queuing extension for OpenCL that provides a high-level API based on the concepts of work pools and work units. *Intel CLU* [75], *OCL-MLA* [53] and *SimpleOpencl* [70] are lightweight API designed to help programmers to rapidly prototype heterogeneous programs.

A more sophisticated approach was proposed in [67]. OmpSs relies on compiler technologies to generate host and kernel code from a sequential program annotated with pragmas. The runtime of OmpSs internally uses a DAG with the scope of scheduling. However, the DAG is not dynamically optimized as done by *libWater*.

2. Data-intensive programming and runtimes

The data intensity of scientific and engineering applications forces the expansion of exascale system. It puts a focus on architectures, programming models, runtime systems improvement on data intensive computing. A major challenge is to utilize the available technologies and large-scale computing resources effectively to tackle the scientific and societal challenges. This section describes the runtime requirements and scalable programming models for data-intensive applications, as well as new data access, communication, and processing operations for data-intensive applications.

2.1. Next generation MPI

MPI is the most widely used standard [10] in the current petascale systems, supporting among others the message passing model. It has proven high performance portability and scalability [9, 17], as well as stability over the last 20 years. MPI provides a (nearly) fixed number of statically scheduled processes with a local view of the data distributed across the system. Nevertheless, ultrascale systems are not going to be built scaling incrementally from current systems, which probably will have a high impact on all levels of the software stack [2, 18]. The international community agrees that changes need to be done in current software at all levels. Future parallel and distributed applications push to explore alternative scalable and reliable programming models [13].

Current HPC systems need to scale up by three orders of magnitude to meet exascale. While a sharp rise in the number of nodes of this magnitude is not expected, the critical growth will come from the intra-node capacities. Fat nodes with a large number of lightweight heterogeneous processing elements, including accelerators, will be common in the ultrascale platforms, mixing parallel and distributed systems. In addition, memory per core ratio is expected to decrease, while the number of NUMA nodes will grow to alleviate the problem of memory coherence between hundreds of cores [16]. On the software side, weak scaling of applications running on such platforms will demand more computation resources to manage huge volumes of data. Nowadays MPI applications, most of which are built using the *bulk-synchronous* synchronization model [11] as a sequence of communication and computation over the interchanged data stages, will continue to be important, but multi-physics and adaptive meshing applications, with multiple components implemented using different programming models, and with dynamic starting and finalization of

such components, will become common in ultrascale. Apart from the most regular applications, this synchronization model is already a strangle point.

In this scenario, programming models need to face multiple challenges to efficiently exploit resources with a high level of programmability [14]: scalability and parallelism increase, energy efficient resource management and data movement, and I/O and resilience in applications among others.

MPI has successfully faced the scalability challenge at petascale with the so-called hybrid model, represented as MPI+X, meaning MPI to communicate between nodes and a shared memory programming model (e.g. OpenMP for shared memory and OpenACC for accelerators) inside a node. This scheme provides with load balancing and reduces the memory footprint and data copies in shared memory, and it is likely to continue in the future. Notwithstanding, increase in node scale, heterogeneity and complexity of integration of processing elements will demand improved techniques for balancing the computational load between potentially large number of processes running kernels composing the application [24].

Increasing imbalances in large-scale applications, aggravated by hardware power management, localized failures or system noise, require synchronization-avoiding algorithms, adaptable to dynamic changes in the hardware and the applications. An example is the collective algorithms based on a non-deterministic point-to-point communication pattern, and able to capture and deal with relevant network properties related to heterogeneity [23]. The MPI specification provides support to mitigate load imbalance issues through the one-sided communication model, non-blocking collectives, or the scalable neighbor collectives for communication along the virtual user defined topology. In the meanwhile, specification and implementation scalability issues have been detected [17]. They need to be either avoided, as the use of all to all inherently non-scalable collectives, or improved, as initialization or communicator creation, in exascale applications.

To support the hybrid programming model, MPI defines levels of thread-safety. Lower levels are suitable for bulk-synchronous applications, while higher levels require synchronization mechanisms, which lead current MPI libraries to a significant performance degradation. *Communication endpoints* [19] mechanism is a proposal to extend the MPI standard for reducing contention inside a single process by allowing to attach threads to different endpoints for sending and receiving messages.

Big data volumes and the power consumed in moving data across the system makes data management one of the main concerns in future systems. In the distributed view, the common methodology of reading data from a centralized filesystem, spreading it over the processing elements and writing the results is energy and performance inefficient, and failure prone. Data will be distributed across the system, and the placement of MPI processes in a virtual topology needs to adapt to the data layout to improve the performance, which will require better mapping algorithms. MPI addresses these challenges in shared memory by a programming model based on shared data windows accessible by processes in the same node, hence avoiding horizontal data movement inside the node. However, lack of data locality awareness leads to vertical movement of data across the memory hierarchy, which degrades performance. For instance, the communication buffers received by MPI processes and the access by the local OpenMP threads for computing on them will need smarter scheduling policies. Data-centric approaches are needed for describing data in the system and apply the computation where such data resides [12, 15].

Another critical challenge for MPI to support exascale systems is the resilience, a cross-cutting issue affecting the whole software stack. Current checkpointing/restart methods are

insufficient for future systems under a failure ratio of a few hours and in the presence of silent errors, and traditional triple modular redundancy (TMR) is not affordable in an energy efficient manner. New techniques of resilient computing have been proposed and developed, also in the MPI context [5, 6]. One proposal for increasing resilience to node failures is to implement *malleable* applications, able to adapt their execution to the available resources in the presence of hardware errors, and avoiding the restart of the application [7].

Alternatives to MPI come from the Partitioned Global Address languages (PGAs) and High Productivity Computing Systems (HPCS) programming languages. PGAs programming models provide a global view of data with explicit communication as CAF [38] (Co-Array Fortran), or implicit communication as UPC [40] (Unified Parallel C). However, static scheduling and poor performance issues make them currently far from replacing the well established and successful MPI+X hybrid model. Moreover, OpenMP presents problems with nested parallelism and vertical locality, so the possibility of MPI+PGAs has been, and continues to be evaluated [8, 22] to provide a programming environment better suited to the future platforms. HPCS languages, such as a Chapel [63] and X10 [64], provide a global view of data and control. For instance, Chapel provides programming constructions at different levels of abstraction. It includes features for computation-centric parallelism based on tasks, as well as data-centric programming capabilities. For instance, the *locale* construction describes the compute nodes in the target architecture and allows to reasoning about locality and affinity, and to manage global views of distributed arrays.

2.2. Runtime requirements for data-intensive applications

Developing data-intensive applications over exascale platforms requires the availability of effective runtime systems. This subsection discusses which functional and non-functional requirements should be fulfilled by future runtime systems to support users in designing and executing complex data-intensive applications over large-scale parallel and distributed platforms in an effective and scalable way.

The functional requirements can be grouped into four areas: data management, tool management, design management, and execution management [39].

Data management. Data to be processed can be stored in different formats, such as relational databases, NoSQL databases, binary files, plain files, or semi-structured documents. The runtime system should provide mechanisms to store and access such data independently from their specific format. In addition, metadata formalisms should be provided to describe the relevant information associated with data (e.g., location, format, availability, available views), in order to enable their effective access, manipulation and processing.

Tool management. Data processing tools include programs and libraries for data selection, transformation, visualization, mining and evaluation. The runtime system should provide mechanisms to access and use such tools independently from their specific implementation. Also in this case metadata should be provided to describe the most important features of such tools (e.g., their function, location, usage).

Design management. From a design perspective, three main classes of data-intensive applications can be identified: single-task applications, in which a single sequential or parallel process task is performed on a given data set; parameter sweeping applications, in which data are analyzed using multiple instances of a data processing tool with different parameters; workflow-based applications, in which data-intensive applications are specified as possibly complex workflows.

A runtime system should provide an environment to effectively design all the above-mentioned classes of applications.

Execution management. The system should provide a parallel/distributed execution environment to support the efficient execution of data-intensive applications designed by the users. Since applications range from single tasks to complex workflows, the runtime system should cope with such a variety of applications. In particular, the execution environment should provide the following functionalities, which are related to the different phases of application execution: accessing the data to be processed; allocating the needed compute resources; running the application based on the user specifications, which may be expressed as a workflow; allowing users to monitor an applications execution.

The non-functional requirements can be defined at three levels: user, architecture, and infrastructure.

From a user point of view, non-functional requirements to be satisfied include:

- *Data protection.* The system should protect data from both unauthorized access and intentional/incidental losses.
- *Usability.* The system should be easy to use by users, without the need of undertaking any specialized training.

From an architecture perspective, the following principles should inspire system design:

- *Openness and extensibility.* The architecture should be open to the integration of new data processing tools and libraries; moreover, existing tools and libraries should be open for extension and modifications.
- *Independence from infrastructure.* The architecture should be designed to be as independent as possible from the underlying infrastructure; in other terms, the system services should be able to exploit the basic functionalities provided by different infrastructures.

Finally, from an infrastructure perspective, non-functional requirements include:

- *Heterogeneous/Distributed data support.* The infrastructure should be able to cope with very large and high dimensional data sets, stored in different formats in a single site, or geographically distributed across many sites.
- *Availability.* The infrastructure should be in a functioning condition even in the presence of failures that affect a subset of the hardware/software resources. Thus, effective mechanisms (e.g., redundancy) should be implemented to ensure dependable access to sensitive resources such as user data.
- *Scalability.* The infrastructure should be able to handle a growing workload (deriving from larger data to process or heavier algorithms to execute) in an efficient and effective way, by dynamically allocating the needed resources (processors, storage, network). Moreover, as soon as the workload decreases, the infrastructure should release the unneeded resources.
- *Efficiency.* The infrastructure should minimize resource consumption for a given task to execute. In the case of parallel/distributed tasks, efficient allocation of processing nodes should be guaranteed. Additionally, the infrastructure should be highly utilized so to provide efficient services.

Even though several research systems fulfilling most of these requirements have been developed, such as Pegasus [25], Taverna [26], Kepler [27], ClowdFlows [28], E-Science Central [29], and COMPSs [30], they are designed to work on conventional HPC platforms, such as clusters, Grids, and - in some cases - Clouds. Therefore, it is necessary to study novel architectures,

environments and mechanisms to fulfill the requirements discussed above, so as to effectively support design and execution of data-intensive applications in future exascale systems.

2.3. Scalable programming models for data-intensive applications

Data-intensive applications often involve a large number of data processing tools that must be executed in a coordinated way to analyze huge amount of data. This section discusses the need for scalable programming models to support the effective design and execution of data-intensive applications on a massive number of processors.

Implementing efficient data-intensive applications is not trivial and requires skills of parallel and distributed programming. For instance, it is necessary to express the task dependencies and their parallelism, to use mechanisms of synchronization and load balancing, and to properly manage the memory and the communication among tasks. Moreover, the computing infrastructures are heterogeneous and require different libraries and tools to interact with them. To cope with all these problems, different *scalable programming models* have been proposed for writing data-intensive applications [31].

Scalable programming models may be categorized based on their level of abstraction (i.e., high-level and low-level scalable models) and based on how they allow programmers to create applications (i.e., visual or code-based formalisms).

Using *high-level scalable models*, the programmers define only the high-level logic of applications while hiding the low-level details that are not fundamental for application design, including infrastructure-dependent execution details. The programmer is helped in application definition and the application performance depends on the compiler that analyzes the application code and optimizes its execution on the underlying infrastructure. Instead, *low-level scalable models* allow the programmers to interact directly with computing and storage elements of the underlying infrastructure and thus to define the applications parallelism directly. Defining an application requires more skills and the application performance strongly depends on the quality of the code written by the programmer.

Data-intensive applications can be designed through *visual programming formalism*, which is a convenient design approach for high-level users, e.g. domain-expert analysts having a limited understanding of programming. In addition, a graphical representation of workflows intrinsically captures parallelism at the task level, without the need to make parallelism explicit through control structures [32]. *Code-based formalism* allows users to program complex applications more rapidly, in a more concise way, and with higher flexibility [33]. The code-base applications can be defined in different ways: i) with a language or an extension of language that allows to express parallel applications; ii) with some annotations in the application code that permits the compiler to understand which instructions will be executed in parallel; and iii) using a library in the application code that adds parallelism to application.

Given the variety of data-intensive applications (from single-task to workflow-based) and types of users (from end users to skilled programmers) that can be envisioned in future exascale systems, there will be a need for scalable programming models with different levels of abstractions (high-level and low-level) and different design formalisms (visual and code-based), according to the classification outlined above. Thus, the programming models should adapt to user needs by ensuring a good trade-off between ease in defining applications and efficiency of executing them on exascale architectures composed by a massive number of processors.

2.4. New data access, communication, and processing operations for data-intensive applications

This subsection discusses the need for new operations supporting data access, data exchange and data processing to enable scalable data-intensive applications on a large number of processing elements.

Data-intensive applications are software programs that have a significant need to process large volumes of data [21]. Such applications devote most of their processing time to run I/O operations and to exchange and move data among the processing elements of a parallel computing infrastructure. Parallel processing of data-intensive applications typically involves accessing, pre-processing, partitioning, aggregating, querying, and visualizing data which can be processed independently. These operations are executed using application programs running in parallel on a scalable computing platform that can be a large Cloud system or a massively parallel machine composed of many thousand processors. In particular, the main challenges for programming data-intensive applications on exascale computing systems come from the potential scalability and resilience of mechanisms and operations made available to developers for accessing and managing data. Indeed, processing very large data volumes requires operations and new algorithms able to scale in loading, storing, and processing massive amounts of data that generally must be partitioned in very small data grains on which analysis is done by thousands to millions of simple parallel operations.

Evolutionary models have been recently proposed that extend or adapt traditional parallel programming models like MPI, OpenMP, MapReduce (e.g., Pig Latin) to limit the communication overhead (in the case of message-passing models) or to limit the synchronization control (in the case of shared-models languages) [2]. On the other hand, new models, languages and APIs based on a revolutionary approach, such as X10, ECL, GA, SHMEM, UPC, and Chapel have been developed. In this case, novel parallel paradigms are devised to address the requirements of massive parallelism.

Languages such as X10, UPC, GA and Chapel are based on a partitioned global address space (PGAS) memory model that can be suited to implement data-intensive exascale applications because it uses private data structures and limits the amount of shared data among parallel threads. Together with different approaches (e.g., Pig Latin and ECL) those models must be further investigated and adapted for providing data-centered scalable programming models useful to support the efficient implementation of exascale data analysis applications composed of up to millions of computing units that process small data elements and exchange them with a very limited set of processing elements. A scalable programming model based on basic operations for data intensive/data-driven applications must include operations for parallel data access, data-driven local communication, data processing on limited groups of cores, near-data synchronization, in-memory querying, group-level data aggregation, and locality-based data selection and classification.

Supporting efficient data-intensive applications on exascale systems will require an accurate modeling of basic operations and of the programming languages/APIs that will include them. At the same time, a significant programming effort of developers will be needed to implement complex algorithms and data-driven applications such that used, for example, in big data analysis and distributed data mining. Programmers must be able to design and implement scalable algorithms by using the operations sketched above. To reach this goal, a coordinated effort between the operation/language designers and the application developers would be very fruitful.

3. Resilience

As exascale systems grow in computational power and scale, failure rates inevitably increase. Therefore, one of the major challenges in these systems is to effectively and efficiently maintain the system reliability. This requires to handle failures efficiently, so that the system can continue to operate with satisfactory performance. The timing constraints of the workload, as well as the heterogeneity of the system resources, constitute another critical issue that must be addressed by the scheduling strategy that is employed in such systems. Therefore, the next generation code will need to be *resistant to failures*. Advanced modeling and simulation techniques are the basic means of investigating fault tolerance in exascale systems, before performing the costly prototyping actions required for resilient code generation.

3.1. Modeling and simulation of failures in large-scale systems

Exascale computing provides a large-scale, heterogeneous distributed computing environment for the processing of demanding jobs. Resilience is one of the most important aspects of exascale systems. Due to the complexity of such systems, their performance is usually examined by *simulation* rather than by analytical techniques. Analytical modeling of complex systems is difficult and often requires several simplifying assumptions. Such assumptions might have an unpredictable impact on the results. For this reason, there have been many research efforts in developing tractable simulation models of large-scale systems.

In [34], simulation models are used to investigate performance issues in distributed systems where the processors are subject to failures. In this research, the author considers that failures are a Poisson process with a rate that reflects the failure probability of processors. Processor repair time has been considered as an exponentially distributed random variable with a mean value that reflects the average time required for the distributed processors to recover. The failure/ repair model of this paper can be used in combination with other models in the case of large-scale distributed processors.

3.2. Checkpointing in exascale systems

Application resilience is an important issue that must be addressed in order to realize the benefits of future systems. If a failure occurs, recovery can be handled by *checkpoint-restart (CPR)*, that is, by terminating the job and restarting it from its last stored checkpoint. There are views that this approach is not expected to scale efficiently to exascale, so different mechanisms are explored in the literature. Gamell et al. in [35] have implemented Fenix, a framework for enabling recovery from failures for MPI-based parallel applications in an online manner (i.e. without disrupting the job). This framework relies on *application-driven, diskless, implicitly-coordinated checkpointing*. Selective checkpoints are created at specific points within the application, guaranteeing global consistency without requiring a coordination protocol.

Zhao et al. in [36] investigate the suitability of a checkpointing mechanism for exascale computers, across both parallel and distributed filesystems. It is shown that a checkpointing mechanism on parallel filesystems is not suitable for exascale systems. However, the simulation results reveal that a *distributed filesystem* with local persistent storage could enable efficient checkpointing at exascale.

In [37], the authors define a model for future systems that faces the problem of *latent errors*, i.e. errors that go undetected for some time. They use their proposed model to derive

optimal checkpoint intervals for systems with latent errors. The importance of a multi-version checkpointing system is explored. They conclude that a multi-version model outperforms a single checkpointing scheme in all cases, while for exascale scenarios, the multi-version model increases efficiency significantly.

Many applications in large-scale systems have an inherent need for fault tolerance and high-quality results within *strict timing constraints*. The scheduling algorithm employed in such cases must guarantee that every job will meet its deadline, while providing at the same time high-quality (i.e. precise) results. In [41], the authors study the scheduling of parallel jobs in a distributed real-time system with possible software faults. They model the system with a queuing network model and evaluate the performance of the scheduling algorithms with simulation techniques. For each scheduling policy they provide an alternative version which allows *imprecise computations*. They propose a performance metric which takes into account not only the number of jobs guaranteed, but also the precision of the results of each guaranteed job. Their simulation results reveal that the alternative versions of the algorithms outperform their respective counterparts. The authors employ the technique of imprecise computations, combined with checkpointing, in order to enhance fault tolerance in real-time systems. They consider monotone jobs that consist of a mandatory part, followed by an optional part. In order for a job to produce an acceptable result, it is required that at least the mandatory part of the job must be completed. The precision of the results is further increased, if the optional part is allowed to be executed longer. The aim is to guarantee that all jobs will complete at least their mandatory part before their deadline.

The authors employ *application-directed checkpointing*. When a software failure occurs during the execution of a job that has completed its mandatory part, there is no need to rollback and re-execute the job. In this case, the system accepts as its result the one produced by its mandatory part, assuming that a checkpoint takes place when a job completes its mandatory part. According to the research findings in [41], in large-scale systems where many software failures can occur, scheduling algorithms based on the technique of imprecise computations could be effectively employed for the fault-tolerant execution of parallel real-time jobs.

3.3. Alternative programming models for fault tolerance in exascale systems

However, programming models that enable more appropriate recovery strategies than CPR are required in exascale systems. Towards this direction, Heroux in [42] presents the following four programming models for developing new algorithms:

- ***Skeptical Programming (SkP):*** SkP requires that algorithm developers should expect that silent data corruption is possible, so that they can develop validation tests.
- ***Relaxed Bulk-synchronous Programming (RBSP):*** RBSP is possible with the introduction of MPI 3.0.
- ***Local Failure, Local Recovery (LFLR):*** LFLR provides programmers with the ability to recover locally and continue application execution when a process is lost. This model requires more support from the underlying system layers.
- ***Selective Reliability Programming (SRP):*** SRP provides the programmer with the ability to selectively declare the reliability of specific data and compute regions.

The User Level Failure Mitigation (ULFM) interface has been proposed to provide fault-tolerant semantics in MPI. In [43], the authors present their experiences on using ULFM in a case study to exploit the advantages and difficulties of this interface to program fault-tolerant

MPI applications. They found that ULFM is suitable for specific types of applications, but it provides few benefits for general MPI applications.

The issue of fault-tolerant MPI is also considered in [44]. Due to the fact that the system kills all the remaining processes and restarts the application from the last saved checkpoint when an MPI process is lost, it is expected that this approach will not work for future extreme scale systems. The authors address this scaling issue through the LFLR programming model. In order to achieve this model, they design and implement a software framework using a prototype MPI with ULFM (MPI-ULFM).

4. Code sustainability and other metrics

Software designers for supercomputers face new challenges. Their code must be efficient whatever the underlying platform is while not wasting computing time for crossing abstraction layers. Several tools presented in the previous sections provide designers and programmers with tools to abstract from the underlying hardware while achieving the maximum performance.

The clear goal to achieve is to increase the raw performance of supercomputers, but it is not anymore the simple *the faster, the better*. Two reasons show that taking care of raw performance is no more sufficient:

- Life of code is way longer than hardware life;
- Other metrics (energy, plasticity, scalability) become more and more important.

4.1. Life cycle of codes

HPC world is comprised of a few widely used codes that serve as base library and a majority of ad-hoc codes often mainly designed and programmed by non-computer scientists.

As an example for the first category, in the scientific computing domain, which aims at constructing mathematical models and numerical solution techniques for solving problems arising in science and engineering, Scalapack [51] is a largely used library of high-performance linear algebra routines for parallel distributed memory machines. It is a base of large-scale scientific codes and can run on nearly every classical supercomputers. It encompasses BLAS (Basic Linear Algebra Subprograms) and pBLAS (parallel BLAS) libraries. This package is comprised of old C and Fortran codes and was first released in 1979 from NASA [52] for BLAS and 1996 for Scalapack [51]. In the last version (version 2.0.2, May 2012) large parts of code are still dating from the first version of 1996, being raw computing code in Fortran or higher level code in C. This version also contains code from nearly every year from 1996 to 2012. This code was able to evolve up to present days due to the community of users behind it. Most other less used libraries or software did not have this chance. But even this library has several sustainability problems such as new hardware architectures: Several supercomputer projects are planning to use GPU [50] or ARM [49] processors instead of classical standard x86 ones.

Concerning the second category the difficulties are even higher, as a large number of codes has been tested and evaluated only on a handful of supercomputers. Their scalability is unknown on different networks or memory topologies for example. In this case, these codes are not sustainable as they require a major rewrite to run on new architecture.

Hence new programming paradigms such as skeletons [47], or YML [48] are needed to reach sustainable codes that run efficiently on the latest generation of supercomputers. Concerning

exascale computing the situation is even more dire as the exact detail of these architectures is still cloudy.

4.2. New metrics

Further away from sustainability of the code itself, other metrics are important for designers and programmers. Power consumption of supercomputers is reaching thresholds that prevent them from continuing to grow like before [46]. The main three metrics that programmers have to confront are:

- *Raw power consumption*: Depending on the particular instructions, library, memory and network access patterns, application will consume different power consumption at particular time and different overall energy for the same work;
- *Scalability*: The capability of scaling up is key as future exascale systems will be composed of hundreds of thousands of cores;
- *Plasticity*: It is the capability of the software to adapt to the underlying hardware architecture (ARM/x86/GPU, network topology, memory hierarchy,. . .) but also to reconfigure itself by changing the number of allocated resources or migrating between architectures at runtime.

At the moment most tools to provide insight on the code to programmers are aiming toward raw computing performance or memory and network usage. Only a few tools exist to provide feedback to programmers on such needed metrics. Valgreen [45] offers to give insight on the power consumption of codes for example. But at the moment manual evaluation is needed in order to evaluate these metrics for any code.

Conclusion

In this article we explored and discussed programming models and runtimes required for scalable high performance computing systems that comprise a very large number of processors and threads. Currently no programming solutions exist that satisfy all the main requirements of such systems. Therefore, new programming models are required that support data locality, minimize data exchange and synchronization, while providing resilience and fault-tolerant mechanisms in order to tackle the increasing probability of failures in such large and complex systems.

New programming models and languages will be a key component of exascale systems. Their design and implementation is one of the pillars of the future exascale strategy that is based on the development of massively parallel hardware, small-grain parallel algorithms and scalable programming tools. All those components must be developed to make that strategy effective and useful. Furthermore, in order to reach actual sustainability, code must reinvent itself and be more independent of the underlying hardware.

One main element will be to create new communication channels between runtime software and development environment. Indeed the latter have all relevant high-level information concerning application structure and adaptation capabilities but they are usually lost when the time to actually run the application comes.

As exascale systems grow in computational power and scale, their resilience becomes increasingly important. Due to the complexity of such systems, fault tolerance must be achieved by employing more effective approaches than the traditional checkpointing scheme. Even though

many alternative approaches have been proposed in the literature, further research is required towards this direction.

Finally, a way to provide higher abstraction from design time to execution time that will be investigated is to extend MPI standards to support this abstraction and to provide higher scalability support.

The work presented in this article has been partially supported by EU under the COST program Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)".

References

1. Ganesh Bikshandi, Jia Guo, Daniel Hoeflinger, Gheorghe Almasi, Basilio B. Fraguela, Mara J. Garzarn, David Padua, and Christoph von Praun, Programming for parallelism and locality with hierarchically tiled arrays. In Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '06). ACM, 48-57, 2006. DOI: 10.1145/1122971.1122981.
2. William Gropp, Marc Snir. Programming for exascale computers. Computing in Science and Engineering, 15(6):27–35, 2013. DOI: 10.1109/mcse.2013.96.
3. John Jenkins, James Dinan, Pavan Balaji, Nagiza F. Samatova, and Rajeev Thakur. Enabling fast, noncontiguous GPU data movement in hybrid MPI+GPU environments. In IEEE International Conference on Cluster Computing (CLUSTER), pages 468–476, 2012. DOI: 10.1109/cluster.2012.72.
4. Jesper Larsson Träff, Antoine Rougier, and Sascha Hunold. Implementing a classic: Zero-copy all-to-all communication with MPI datatypes. In 28th ACM International Conference on Supercomputing (ICS), pages 135–144, 2014. DOI: 10.1145/2597652.2597662.
5. G. Bosilca, A. Bouteiller, and F. Cappello. MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In ACM/IEEE Supercomputing Conference, page 29. IEEE, 2002. DOI: 10.1109/sc.2002.10048.
6. G. E. Fagg, A. Bukovsky, and J. J. Dongarra. HARNESS and fault tolerant MPI. Parallel Computing, 27(11):1479–1495, October 2001. DOI: 10.1016/s0167-8191(01)00100-4.
7. C. George and S. S. Vadhiyar. ADFT: An adaptive framework for fault tolerance on large scale systems using application malleability. Procedia Computer Science, 9:166–175, 2012. DOI: 10.1016/j.procs.2012.04.018.
8. J. Dinan, P. Balaji, E. Lusk, P. Sadayappan, and R. Thakur. Hybrid parallel programming with MPI and unified parallel C. in Proceedings of the 7th ACM international conference on Computing frontiers, CF '10, 2010. DOI: 10.1145/1787275.1787323.
9. P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur, and J. L. Träff, MPI on millions of cores. Parallel Processing Letters, vol. 21, no. 1, pp. 45–60, 2011. DOI: 10.1142/s0129626411000060.
10. MPI Forum: MPI: A Message-Passing Interface Standard. Version 3.0 (September 4 2012).

11. Jerry Eriksson, Radoslaw Januszewski, Olli-Pekka Lehto, Carlo Cavazzoni, Torsten Wilde and Jeanette Wilde. System Software and Application Development Environments. PRACE Second Implementation Phase Project, D11.2, 2014.

12. D. Unat, J. Shalf, T. Hoefler, T. Schulthess, A. Dubey et. al. Programming Abstractions for Data Locality. White Paper, PADAL Workshop, 28-29 April, 2014, Lugano Switzerland.

13. S. Amarasinghe, M. Hall, R. Lethin, K. Pingali, D. Quinlan, V. Sarkar, J. Shalf, R. Lucas, and K. Yelick. ASCR programming challenges for exascale computing. Report of the 2011 workshop on exascale programming challenges, University of Southern California, Information Sciences Institute, July 2011.

14. R. Lucas et. al. Top Ten Exascale Research Challenges. DOE ASCAC Subcommittee Report, February 2014.

15. C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice & Experience, 23(2):187–198, February 2011. DOI: 10.1002/cpe.1631.

16. Ang, J. A., Barrett, R. F., Benner, R. E., Burke, D., Chan, C., Cook, J., Donofrio, D., Hammond, S. D., Hemmert, K. S., Kelly, S. M., Le, H., Leung, V. J., Resnick, D. R., Rodrigues, A. F., Shalf, J., Stark, D., Unat, D. and Wright, N. J. Abstract Machine Models and Proxy Architectures for Exascale Computing. Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing. Co-HPC '14, New Orleans, Louisiana. IEEE Press 978-1-4799-7564-8, pages: 25-32. 2014. DOI: 10.1109/co-hpc.2014.4.

17. R. Thakur, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, J. Larsson Träff. MPI at Exascale. Department of Energy SciDAC workshop, Jul, 2010.

18. J. Dongarra et al. The International Exascale Software Project roadmap. International Journal of High Performance Computing Applications, 25:3–60, 2011.

19. J. Dinan, R. Grant, P. Balaji, D. Goodell, D. Miller, M. Snir, R. Thakur. Enabling communication concurrency through flexible MPI endpoints. International Journal of High Performance Computing Applications, volume 28, pages 390-405. 2014. DOI: 10.1177/1094342014548772.

20. J. Carretero, J. Garcia-Blas, D. Singh, F. Isaila, T. Fahringer, R. Prodan, G. Bosilca, A. Lastovetsky, C. Symeonidou, H. Perez-Sanchez, J. Cecilia. Optimizations to Enhance Sustainability of MPI Applications. Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, Kyoto, Japan, 2014. DOI: 10.1145/2642769.2642797.

21. I. Gorton, P. Greenfield, A. Szalay, R. Williams. Data-intensive computing in the 21st century. IEEE Computer, 41 (4), 30-32. DOI: 10.1109/mc.2008.122.

22. European Commission EPiGRAM project (grant agreement no 610598). http://www.epigram-project.eu.

23. K. Dichev, F. Reid, A. Lastovetsky. Efficient and Reliable Network Tomography in Heterogeneous Networks Using BitTorrent Broadcasts and Clustering Algorithms. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC12). Salt Lake City, Utah, USA. 2012. IEEE Computer Society Press, ISBN: 978-1-4673-0804-5, pp. 36:1–36:11. DOI: 10.1109/sc.2012.52.

24. Z. Zhong, V. Rychkov, A. Lastovetsky. Data Partitioning on Multicore and Multi-GPU Platforms Using Functional Performance Models. IEEE Transactions on Computers, PrePrints. DOI: 10.1109/TC.2014.2375202.

25. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 13(3):219–237, 2005. DOI: 10.1155/2005/128026.

26. K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Research, 41(W1):W557–W561, July 2013. DOI: 10.1093/nar/gkt328.

27. B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18(10):1039–1065, 2006. DOI: 10.1002/cpe.994.

28. J. Kranjc, V. Podpecan, and N. Lavrac. ClowdFlows: A Cloud Based Scientific Workflow Platform. In P. Flach, T. Bie, and N. Cristianini, editors, Machine Learning and Knowledge Discovery in Databases, LNCS 7524: 816–819. Springer, Heidelberg, Germany, 2012. DOI: 10.1007/978-3-642-33486-3_54.

29. H. Hiden, S.Woodman, P.Watson, and J. Cala. Developing cloud applications using the e-Science Central platform. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 371(1983), January 2013. DOI: 10.1098/rsta.2012.0085.

30. F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia. ServiceSs: An Interoperable Programming Framework for the Cloud. Journal of Grid Computing, vol. 12, n. 1, pp. 67-91, 2014. DOI: 10.1007/s10723-013-9272-5.

31. J. Diaz, C. Munoz-Caro and A. Nino. A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era, Parallel and Distributed Systems, IEEE Transactions on , vol.23, no.8, pp.1369-1386, Aug. 2012. DOI: 10.1109/tpds.2011.308.

32. K. Maheshwari and J. Montagnat. Scientific workflow development using both visual and script-based representation. In Proceedings of the 2010 6th World Congress on Services, SERVICES '10, pages 328–335, Washington, DC, USA, 2010. DOI: 10.1109/services.2010.14.

33. F. Marozzo, D. Talia, P. Trunfio, "JS4Cloud: Script-based Workflow Programming for Scalable Data Analysis on Cloud Platforms". Concurrency and Computation: Practice and Experience, Wiley InterScience, 2015. DOI: 10.1002/cpe.3563.

34. H. D. Karatza. Performance analysis of a distributed system under time-varying workload and processor failures. Proceedings of the 1st Balkan Conference on Informatics (BCI'03), Nov. 2003, Thessaloniki, Greece , pp. 502–516.

35. M. Gamell, D. S. Katz, H. Kolla, J. Chen, S. Klasky, and M. Parashar. Exploring automatic, online failure recovery for scientific applications at extreme scales. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14), Nov. 2014, New Orleans, USA, pp. 895–906. DOI: 10.1109/sc.2014.78.

36. D. Zhao, D. Zhang, K. Wang, and I. Raicu. Exploring reliability of exascale systems through simulations. Proceedings of the High Performance Computing Symposium (HPC'13), Apr. 2013, San Diego, USA, pp. 1–9.

37. G. Lu, Z. Zheng, and A. A. Chien. When is multi-version checkpointing needed? Proceedings of the 3rd Workshop on Fault-tolerance for HPC at Extreme Scale (FTXS'13), Jun. 2013, New York, USA, pp. 49–56. DOI: 10.1145/2465813.2465821.

38. Numrich, Robert W., and John Reid. Co-Array Fortran for parallel programming. ACM Sigplan Fortran Forum. Vol. 17. No. 2. ACM, 1998. DOI: 10.1145/289918.289920.

39. F. Marozzo, D. Talia, P. Trunfio. Cloud Services for Distributed Knowledge Discovery. In: Encyclopedia of Cloud Computing, S. Murugesan, I. Bojanova (Editors), Wiley-IEEE, 2016.

40. El-Ghazawi, Tarek, and Lauren Smith. UPC: unified parallel C. Proceedings of the 2006 ACM/IEEE conference on Supercomputing. ACM, 2006. DOI: 10.1145/1188455.1188483.

41. G. L. Stavrinides and H. D. Karatza. Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations. Simulation: Transactions of the Society for Modeling and Simulation International, vol. 85, no. 8, 2009, pp. 525–536. DOI: 10.1177/0037549709340729.

42. M. A. Heroux. Toward resilient algorithms and applications. arXiv:1402.3809, March 2014.

43. I. Laguna, D. F. Richards, T. Gamblin, M. Schulz, and B. R. de Supinski. Evaluating user-level fault tolerance for MPI applications. Proceedings of the 21st European MPI Users' Group Meeting (EuroMPI/ASIA'14), Sep. 2014, Kyoto, Japan, pp. 57–62. DOI: 10.1145/2642769.2642775.

44. K. Teranishi and M. A. Heroux. Toward local failure local recovery resilience model using MPI-ULFM. Proceedings of the 21st European MPI Users' Group Meeting (EuroMPI/ASIA'14), Sep. 2014, Kyoto, Japan, pp. 51–56. DOI: 10.1145/2642769.2642774.

45. Leandro Fontoura Cupertino, Georges Da Costa, Jean-Marc Pierson. Towards a generic power estimator. In : Computer Science - Research and Development, Springer Berlin / Heidelberg, Special issue : Ena-HPC 2014, July 2014. DOI: 10.1007/s00450-014-0264-x.

46. Shalf, John, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. High Performance Computing for Computational Science–VECPAR 2010. Springer Berlin Heidelberg, 2011. 1-25. DOI: 10.1007/978-3-642-19328-6_1.

47. Cole, Murray. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel computing 30 (3), Elsevier 2004: pages 389-406. DOI: 10.1016/j.parco.2003.12.002.

48. Petiton, S., Sato, M., Emad, N., Calvin, C., Tsuji, M., and Dandouna, M. Multi level programming Paradigm for Extreme Computing. In SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo, 2014. EDP Sciences. DOI: 10.1051/snamc/201404305.

49. Ramirez, A. (2011). European scalable and power efficient HPC platform based on low-power embedded technology. On-Line. Access date: March/2012. URL: `http://www.eesi-project.eu/media/BarcelonaConference/Day2/13-Mont-Blanc_Overview.pdf`.

50. Nukada, Akira, Kento Sato, and Satoshi Matsuoka. Scalable multi-gpu 3-d fft for tsubame 2.0 supercomputer. Proceedings of the International Conference on High Performance

Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012. DOI: 10.1109/sc.2012.100.

51. Choi, Jaeyoung, James Demmel, Inderjiit Dhillon, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, Ken Stanley, David Walker, and R. Clinton Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance. In Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science, pp. 95–106. Springer Berlin Heidelberg, 1996. DOI: 10.1007/3-540-60902-4_12.
52. Lawson, Chuck L., Richard J. Hanson, David R. Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. ACM Transactions on Mathematical Software (TOMS) 5, no. 3 (1979): 308–323. DOI: 10.1145/355841.355847.
53. OCL-MLA, `http://tuxfan.github.com/ocl-mla/`.
54. Bariş Eskikaya and D Turgay Altilar, "Distributed OpenCL Distributing OpenCL Platform on Network Scale", IJCA 2012, pp. 26–30.
55. Philipp Kegel, Michel Steuwer and Sergei Gorlatch, "dOpenCL: Towards a Uniform Programming Approach for Distributed Heterogeneous Multi-/Many-Core Systems" IPDPS Workshops 2012. DOI: 10.1109/ipdpsw.2012.16.
56. Jungwon Kim, Sangmin Seo, Jun Lee, Jeongho Nah, Gangwon Jo and Jaejin Lee, "SnuCL: an OpenCL framework for heterogeneous CPU/GPU clusters", ICS 2012. DOI: 10.1145/2304576.2304623.
57. Shucai Xiao and Wu-chun Feng, "Generalizing the Utility of GPUs in Large-Scale Heterogeneous Computing Systems", IPDPS Workshops, 2012. DOI: 10.1109/ipdpsw.2012.325.
58. Ridvan Özaydin and D. Turgay Altilar, "OpenCL Remote: Extending OpenCL Platform Model to Network Scale", HPCC-ICESS 2012. DOI: 10.1109/hpcc.2012.117.
59. Alves Albano, Rufino Jose, Pina Antonio and Santos Luis Paulo, "Enabling task-level scheduling on heterogeneous platforms", Workshop GPGPU, 2012.
60. Khronos OpenCL Working Group, "The OpenCL 1.2 specification", 2012, `http://www.khronos.org/opencl`.
61. José Duato, Antonio J. Peña, Federico Silla, Rafael Mayo and Enrique S. Quintana-Ortí, "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters", HPCS 2010. DOI: 10.1109/hpcs.2010.5547126.
62. Orion S. Lawlor, "Message passing for GPGPU clusters: CudaMPI", CLUSTER 2009. DOI: 10.1109/clustr.2009.5289129.
63. Chamberlain, Bradford L., David Callahan, and Hans P. Zima. Parallel programmability and the chapel language. International Journal of High Performance Computing Applications 21.3 (2007): 291-312. DOI: 10.1177/1094342007078442.
64. Charles, Philippe, et al. X10: an object-oriented approach to non-uniform cluster computing. ACM Sigplan Notices 40.10 (2005): 519-538. DOI: 10.1145/1103845.1094852.
65. Sun Enqiang, Schaa Dana, Bagley Richard, Rubin Norman and Kaeli David, "Enabling task-level scheduling on heterogeneous platforms", WORKSHOP GPGPU 2012. DOI: 10.1145/2159430.2159440.
66. Magnus Strengert, Christoph Müller, Carsten Dachsbacher and Thomas Ertl, "CUDASA: Compute Unified Device and Systems Architecture", EGPGV 2008.

67. Bueno Javier, Planas Judit, Duran Alejandro, Badia Rosa M., Martorell Xavier, Ayguade Eduard and Labarta Jesus, "Productive Programming of GPU Clusters with OmpSs", IPDPS 2012. DOI: 10.1109/ipdps.2012.58.

68. Ryo Aoki, Shuichi Oikawa, Takashi Nakamura and Satoshi Miki, "Hybrid OpenCL: Enhancing OpenCL for Distributed Processing", ISPA 2011. DOI: 10.1109/ispa.2011.28.

69. A. Barak, T. Ben-Nun, E. Levy and A. Shiloh, "A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices", Workshop PPAC 2010. DOI: 10.1109/clusterwksp.2010.5613086.

70. Simple-opencl `http://code.google.com/p/simple-opencl/`.

71. Ivan Grasso, Simone Pellegrini, Biagio Cosenza, Thomas Fahringer. "libwater: Heterogeneous Distributed Computing Made Easy", ACM International Conference on Supercomputing, Eugene, USA, 2013. DOI: 10.1145/2464996.2465008.

72. Nesus European Cost Action IC1305 `http://www.nesus.eu/`.

73. NCF, Peter Michielse, and Patrick Aerts NCF. "European Exascale Software Initiative".

74. Dongarra, Jack. "The international exascale software project roadmap". International Journal of High Performance Computing Applications (2011): 1094342010391989. APA.

75. Computing Language Utility, Intel Corporation, `http://software.intel.com/`.

Received February 27, 2015.

DOI: 10.14529/jsfi150202

Acceleration of MPI Mechanisms for Sustainable HPC Applications

***Jesus Carretero*[1], *Javier Garcia-Blas*[1], *David E. Singh*[1], *Florin Isaila*[1], *Alexey Lastovetsky*[2], *Thomas Fahringer*[3], *Radu Prodan*[3], *Peter Zangerl*[3], *Christi Symeonidou*[4], *George Bosilca*[5], *Afshin Fassihi*[6], *Horacio Pérez-Sánchez*[6]**

Ultrascale computing systems are meant to reach a growth of two or three orders of magnitude of today computing systems. However, to achieve the performances required, we will need to design and implement more sustainable solutions for ultra-scale computing systems, understanding sustainability in a holistic manner to address challenges as economy-of-scale, agile elastic scalability, heterogeneity, programmability, fault resilience, energy efficiency, and scalable storage. Some of those solutions could be provided into MPI, but other should be devised as higher level concepts, less generalists, but adapted to applicative domains, possibly as programming patterns or or libraries. In this paper, we show some proposals to extend MPI trying to cover major domains that are relevant towards sustainability: MPI programming optimizations and programming models, resilience, data management, and their usage from applications.

Keywords: MPI, MPI sustainability, programming models, resilience, data management, MPI applications.

Introduction

The interest of governments, industry, and researchers in very large scale computing systems has significantly increased in recent years, and steady growth of computing infrastructures is expected to continue in data centers and supercomputers due to the ever-increasing data and processing requirements of various domain applications, which are constantly pushing the computational limits of current computing resources. However, it seems that we have reached a point where system growth can no longer be addressed in an incremental way, due to the huge challenges lying ahead. In particular scalability, energy barrier, data management, programmability, and reliability all pose serious threats to tomorrow's cyberinfrastructure.

The idea of an Ultrascale Computing Systems (UCS), envisioned as a large-scale complex system joining parallel and distributed computing systems that cooperate to provide solutions to the users might be one solution to these growing problems at scale. As all the above models rely on distributed memory systems, the Message-Passing Interface (MPI) remains a promising paradigm to develop and deploy parallel applications, and it is already proven at larger scale — with machines running 100K+ processes. However, can we be sure that MPI will be sustainable in Ultrascale systems? If we understand sustainability as the probability that today's MPI functionality will be useful, available, and improved in the future, the answer is "yes". MPI behaves as a portability layer between the application developer and the hardware resources, hiding most architectural details from application developers. The independence from the computing platform has allowed new versions of MPI to include features that, when carefully combined

[1]University Carlos III of Madrid, Madrid, Spain
[2]University College Dublin, Dublin, Ireland
[3]University of Innsbruck, Innsbruck, Austria
[4]ICS, FORTH, Heraclion, Greece
[5]University of Tennessee, Knoxville, USA
[6]Universidad Católica San Antonio de Murcia (UCAM), Murcia, Spain

with other libraries and integrated into dynamic high-level programming paradigms, permit the development of adaptable applications and novel programming paradigms, molding themselves to the scale of the underlying execution platform.

However, we will need to design and implement more sustainable solutions for Ultrascale computing systems, understanding sustainability in a holistic manner to address challenges like economy-of-scale, agile elastic scalability, heterogeneity, programmability, fault resilience, energy efficiency, and scalable storage. Some of those solutions could be integrated and provided by MPI, but others should be devised as higher level concepts, less general, but adapted to applicative domains, possibly as programming patterns or libraries. In this paper, we layout some proposals to extend MPI to cover major relevant domains in a move towards sustainability, including: MPI programming optimizations and programming models, resilience, data management, and their usage for applications.

The remainder of this paper is organized as follows. Section 1 covers communication optimizations, while Section 2 addresses the area of resilience. Section 3 talks about storage and I/O techniques, Section 4 deals with energy constraints, and Section 5 presents some application and algorithm optimizations. The final section concludes the paper.

1. Enhancing MPI runtime and programming models

As the scale and complexity of systems increases, it is becoming more important to provide MPI users with optimizations and programming models to hide this complexity, while providing a mechanism to expose part of this information for application developers seeking knowledge of low-level functions. One possible way to achieve automatic application optimization is to provide a layered API and allow a compiler tool to convert between MPI and this layered API, as necessary. Another potential approach would involve more efficiently integrating new programming models (e.g., OpenMP or PGAS) for cooperatively sharing not only a common high-level goal—such as a view of the application's time-to-completion—but all resources of the targeted platform. In this section, we focus on some optimizations shown to enhance MPI's scalability and performance. These optimizations provide minimum APIs to transparently enhance portability and sustainability of application software, thus minimizing the adaptation effort.

1.1. Distributed Region-based memory Allocation and Synchronization

Even though the existing distributed global address memory models, such as PGAS, support global pointers, their potential efficiency is hindered by the expensive and unnecessary messages generated by global memory accesses. In order to transfer their data among nodes, they must either marshal and un-marshal their data during the communication, or be represented in a non-intuitive manner.

DRASync [23] is a region-based allocator that implements a global address space abstraction for MPI programs with pointer-based data structures. Regions are a collection of contiguous memory spaces used for storing data. They offer great locality since similar data can be placed together and can be easily transmitted in bulk. DRASync offers an API for creating, deleting, and transferring such regions. It enables MPI processes to operate on a region's data by acquiring the containing region and releasing it at the end of computation for other processes to acquire. Each region is combined with ownership semantics, allowing the process that created it, or one that acquired it, to have exclusive write permissions to its data. DRASync, however, does not

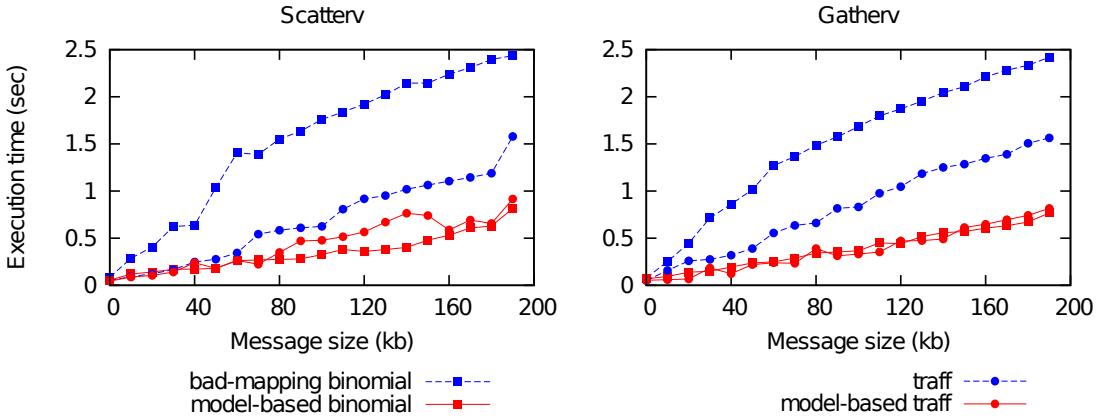


Figure 1. Scatterv and Gatherv operations on geographically distributed clusters from Grid5000

restrain other MPI processes, that are not owners, from acquiring read-only copies of the region. Thus, acquire/release operations are akin to reader-writer locks and enable DRASync to provide an intuitive synchronization tool that simplifies the design of MPI applications.

DRASync has been evaluated over the Myrmics [17] allocator using two application-level benchmarks, the Barnes-Hut N-body simulation and the Delaunay triangulation with variant datasets. The encouraging outcome highlighted the fact that DRASync produces comparable performance results while providing a more intuitive synchronization abstraction for programmers.

1.2. Optimization of MPI collectives

Algorithms for MPI collective communication operations typically translate the collective communication pattern as a combination of point-to-point operations in an overlay topology, mostly a tree-like structure. The traditional targets for such an algorithmic deployment are homogeneous platforms with identical processors and communication layers. When applied to heterogeneous platforms, these implementations may be far from optimal, mainly due to the uneven communication capabilities of the different links in the underlying network. In [10], we proposed to use heterogeneous communication performance models and their prediction to find more efficient, almost optimal, communication trees for collective algorithms on heterogeneous networks. The models take into account the heterogeneous capabilities of the underlying network of computers when constructing communication trees. Model predictions are used during the dynamic construction of communication trees either by changing the mapping of the application processes or changing the tree structure altogether. Experiments on Grid5000 using 39 nodes geographically distributed over 5 clusters stretched over 2 sites, demonstrate that the proposed model-based algorithms clearly outperform their non-model-based counterparts on heterogeneous networks (see fig. 1).

1.3. MPI communication with adaptive compression

Adaptive-CoMPI [11] is an MPI extension which performs the adaptive message compression of MPI-based applications to reduce communication volume, and thus time, and enhances application performance. It is implemented as a library connected through the Abstract Device

Interface of MPICH so that it can be used with any MPI-based application in a transparent manner, as the user does not need to modify the source code. Adaptive-CoMPI addresses all types of communications, and includes different compression techniques (LZO [26] , RLE [16], HUFFMAN [8], RICE [5] and FPC [19]) that can be used transparently to users (by means of MPI hints).

The architecture of MPICH consists of 3 layers: Application Programmer Interface (API), Abstract Device Interface (ADI) and Channel Interface (CI). The ADI layer is a portable layer, while the Channel layer is not. Therefore, we have modified the ADI layer in order to include the Runtime Compression strategy, independently of the channel and protocol used. Therefore, applying Runtime Compression strategy on point-to-point routines, not only compresses these communications, but also the collective ones. The same is true for blocking and non-blocking communication.

Algorithm 1 Blocking message send

```
len-buffer = contig-size  count
if(len > 2048)
      algorithm-compress ← Read-Hint-User()
       len-compress ← Compression-Message(buff, len, algorithm-compress, buff-compress)
       Send-Message-contiguous(buff-compress, len-compress, src-rank, dest-rank)
end if
```

Algorithm 2 Blocking message reception

```
Check if data is compressed
if(check-request == 1)
      flag-head = Study-Head-of-Buffer(request.buf)
      if(flag-head == yes-compression)
      Decompress the buffer
            buf ← Decompression-Message(request.buf)
      else
            Copy(request.buf → buf)
      end if
end if
```

For all messages sent, a header is included to inform the receiver process if it has to decompress the message or not, and which algorithm must be used.

Adaptive-CoMPI includes two possible compression strategies. The first one, called *Runtime Adaptive Strategy* (RAS) analyzes the performance of the communication network and the efficiency of different compression algorithms before the application execution. Based on this information, during the application execution, it decides if it is worth it to compress a message or not, and if so, it chooses the most appropriate compression algorithm. This feature allows the Runtime Adaptive Strategy to offer adaptive compression capabilities without any previous knowledge of the application characteristics. RAS decision process consists of the following steps:

1. Selecting the best compression algorithm at the beginning of the application and everytime the data type changes.
2. Finding the minimum size of the message from which the performance improves.

3. Sending the message (compressed or not).
4. For subsequent messages with the same datatype, the process compares the message size with $length - yes - compression$. If the size of the message is higher, then the message is sent compressed, and uncompressed otherwise.
5. Once the message is sent, and if the decision is to compress, the process checks if the compression-ratio is less than one. In the number of mistakes is higher than a certain threshold within a time interval, the compression is disabled (see fig. 2a).
6. In the other case, if the decision was uncompressed, the process updates the number of messages that have been sent uncompressed. When the number is higher than a reevaluation threshold, then the evaluation is restarted (see fig. 2b).

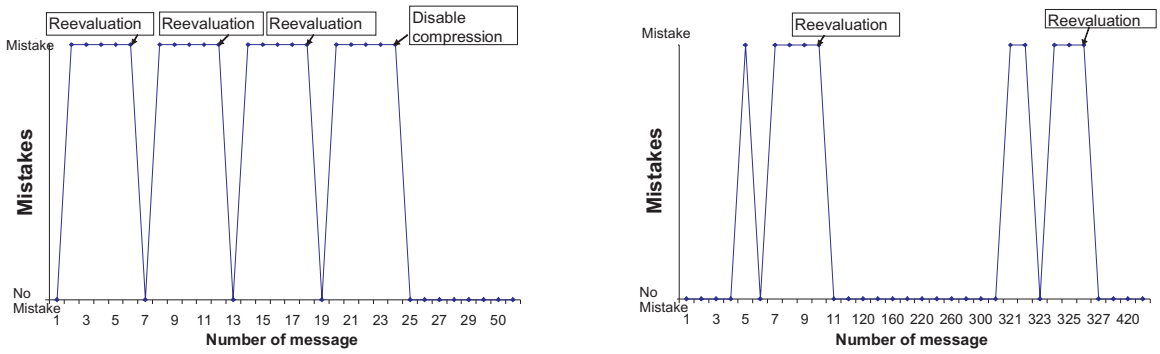


Figure 2. Adaptive-CoMPI with RAS strategy. Learning from errors.

One of the main characteristics of RAS is that it can adapt itself to the applications behavior at runtime. This strategy learns from previous messages which is the compression algorithm to be used and the size from which it obtains a benefit by compressing data.

The second approach, called *Guided Strategy*, provides an application-tailored solution based on the prior application analysis using the application profiling. With this approach along the first execution of the MPI application all the messages are stored in a log file. Upon completion, the best compression algorithm is determined off-line for each message and it is registered in a *decision rules* file. When the application is executed again with the same input parameters and in the same environment, Adaptive-CoMPI extracts the information from the *decision rules* file and applies the most appropriate compression technique for each message.

Adaptive-CoMPI has been evaluated using real applications (BIPS3D, PSRG, and STEM), as well as using the NAS benchmarks. Fig. 3 shows the speedup achieved by BIPS3D when Adaptive-CoMPI techniques are applied. The benchmark has been executed running up to 256 processes in a cluster with dual-core nodes. As may be seen, using Adaptive-CoMPI provides always a performance increase, to a maximum of 1.8 speedup with the same resources.

Fig. 4 shows the speedup achieved for each strategy compared to the execution of the application without compression. Note that the Guided Strategy finds the best compression technique (including no compressing) for each message, providing the optimal compression rate for each independent message. We can observe that the Runtime Adaptive Strategy obtains a performance similar to the guided one which means that, globally speaking, it is able to efficiently compress the messages with no previous knowledge of the application.

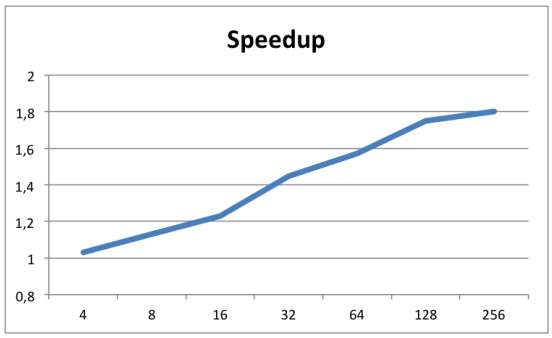


Figure 3. Speedup of Adaptive-CoMPI for the BIPS3D application

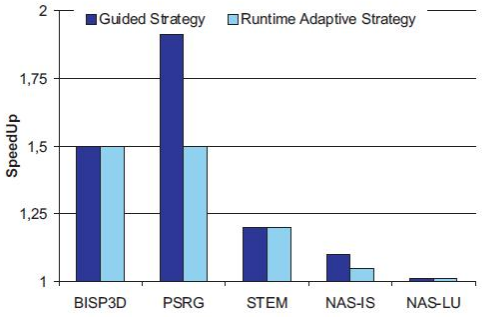


Figure 4. Comparing RAS and Guided-Strategy

2. Resilience

Should the number of components in supercomputers continue to increase, the mean time between failures (MTBF) is expected to decrease to a handful of hours, preventing any capacity application from succesfully delivering its scientific outcome. As a result, deploying fault tolerant strategies within HPC software stack will not only become critically important, but it will have a direct and lasting impact: a massive improvement in application runtime and efficient resource usage compared to currently deployed techniques used to alleviate the consequence of failures (that is, resubmission of failed jobs, and simplistic periodic checkpointing to disk). However, system level checkpoint/restart is unable, in its current state, to cope with very adversarial future failure patterns. This presents a clear need to improve checkpointing strategies by simultaneously addressing several issues: 1) optimizing the checkpoint procedure by minimizing the storage requirements and by diverging from centralized I/O strategies; and 2) allowing independent restart of failed processes without rollback of all processes [4]. One should understand that adopting such a solution allows the application to complete under a reasonable time interval, but in exchange requires significant investment in reliable hardware (more memory or NVRAM, increased reliable storage and so on). Thus, the relief is only temporary, as the increase in the number of components in the checkpoint restart chain will, by definition, have an impact on the MTBF. Moreover, the total ownership cost of the application will increase, as all these hardware additions will increase the energy requirements for large platforms, a requirement extremely difficult to satisfy at the Exascale level.

Thus, the first potential solution is to simultaneously address two of the major drawbacks of the system-level coordinated checkpoint, by decreasing not only the checkpoint size by also it's frequency. Such solutions have been thriving recently, proposing different interface to address this problem. As an example, in Fault Tolerant Messaging Interface (FMI) [22] employs a survivable communication runtime coupled with a fast, in-memory C/R and dynamic host allocation to enable low-latency recovery. The application developer highlights the critical data for the correct execution of the application, as well as windows of opportunity for a correct checkpoint, allowing the FMI runtime to decide the frequency and the amount of data to be checkpointed. On a somehow similar approach, Fault Tolerance Interface (FTI) [1] proposes to address these challenges by proposing a low-overhead high-frequency multi-level checkpoint approach, in which a highly-reliable topology-aware Reed-Salomon encoding is integrated deep inside the checkpoint scheme. A more data centric approach, named Containment Domain (CD) [7] proposes a programming construct that enable applications to express resilience needs and to interact with

the system to tune and specialize error detection, state preservation and restoration. They behave as weak transactional primitives and can be nested to take advantage of the machine and application hierarchies allowing hierarchical state preservation, restoration and recovery.

Faithful to their coordinated checkpoint/restart roots, these approaches inherit from a former programming period, where synchronous SPMD and BSD application were ruling the parallel application world. They are based on synchronous concepts, forcing a strict coordination not only during the checkpoint, but also during the restart (in addition to requiring a complete restart and a full data recovery). They provide little flexibility to the application to implement specialized fault management approaches, or to take advantage of algorithmic properties in order to code with the faults. Moreover, they do not provide support in the programming paradigm for fault detection without a drastic restart, nor to any kind of support from the message passing substrate.

However, over the last years, algorithm based fault tolerant techniques have proven to be capable to forgo checkpointing completely by employing a tailored, scalable protective strategy to maintain sufficient algorithm-specific redundancy to restore lost data pieces due to failures without a global need for restart. Moreover, a large number of application can cope with a lesser support for fault management from the runtime. Domain decomposition, naturally fault tolerant applications, and master-worker, in which the partial loss of the dataset is not a catastrophic event that commands interrupting progress toward the solution, are just a few examples of such resilient applications. All of these recovery patterns hit one of the historic roadblocks that have hindered the deployment of fault tolerant software: the lack of proper support from the popular communication libraries, MPI and PGAS, which thereby limits recovery options to full-job restart upon failure.

Resiliency should refer not only to the ability of the MPI application to be restarted after a failure, but also to the ability to survive failures and to recover to a consistent state from which the execution can be resumed. In recent developments, the MPI Forum has proposed an extension of the MPI standard that permits restoring the capability of MPI to communicate after failures strike [2]. One of the most strenuous challenges is to ensure that no MPI operation stalls from the consequences of failures, as fault tolerance is impossible if the application cannot regain full control of the execution. In the proposed standard, an error is returned when a failure prevents a communication from completing. However, it indicates only the local status of the operation, and does not permit assuming if, nor how, the associated failure has impacted MPI operations at other ranks. This design choice avoids expensive consensus synchronizations from obtruding into MPI routines, but leaves open the danger of some processes proceeding unaware of the failure. This novel proposal is a low level layer, basically the most basic portability layer, that can be exposed at the communication infrastructure level, to allow for flexible and portable higher level concepts, but adapted to specific applicative domains. Thus, these additions propose to put the resolution of such situations under the control of the application programmer, by providing supplementary interfaces that reconstruct a consistent global view of the application state (typical case for applications with collective communications). Aside from applications, these new interfaces can be used by high-level abstractions, such as FMI, FTI, CS, transactional fault tolerance, uncoordinated checkpoint-restart, and programming languages, to implement their own needs and to provide seamless support for advanced fault tolerance models that are thereby portable between MPI implementations.

3. Data and Input/Output

Data storage and management is a major concern for Ultrascale systems, as the increased scale of the systems and the data demand from the applications lead to major I/O overheads that are actually hampering the performance of the applications themselves. MPI has proposed asynchronous I/O operations to allow overlapping I/O and computation, but this feature does not reduce the latency of the system, which is inherent in the length of the I/O path. To this end, there is a major trend towards increasing data locality to avoid data movements: the data-centric paradigm.

In this sense, AHPIOS (Ad-Hoc Parallel I/O system for MPI applications) [14] proposes a scalable parallel I/O system completely implemented in MPI. AHPIOS allows MPI applications to dynamically manage and scale distributed partitions in a convenient way. The configuration of both MPI-IO and the storage management system is unified and allows for a tight integration of the optimizations of all layers. AHPIOS partitions are elastic as they conveniently scale up and down with the number of resources. AHPIOS proposes two collective I/O strategies, which leverage a two-tiered cooperative cache in order to exploit the spatial locality of data-intensive parallel applications. The file access latency is hidden from the applications through an asynchronous data staging strategy. The two-tiered cooperative cache scales with both the number of processors and storage resources. The first cooperative cache tier runs along with the application processes and hence scales with the number of application processes. The second cooperative cache tier runs at the I/O servers and, therefore, scales with the number of global storage devices. Finally, AHPIOS takes advantage of view-based I/O [3] is a file-system independent I/O optimization based on file views. View-based I/O avoids the necessity of transferring large lists of offset-length pairs at file access time as the present implementation of two-phase I/O.

Given an MPI application accessing files through the MPI-IO interface and a set of distributed storage resources, AHPIOS constructs a distributed partition on demand, which can be accessed transparently and efficiently. Files stored in one AHPIOS partition are transparently striped over storage resources, each partition being managed by a set of storage servers running together as an independent MPI application. Access to an AHPIOS partition is performed through an MPI-IO interface, allowing it to scale up and down on demand during run-time.

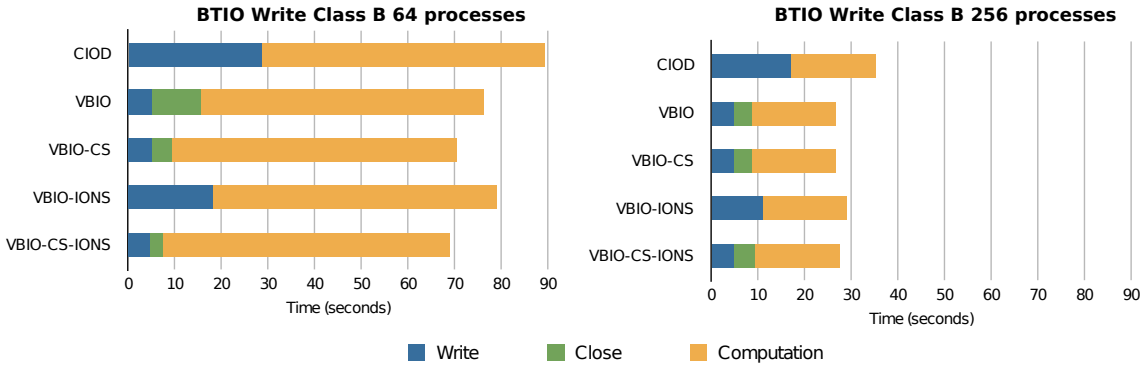


Figure 5. AHPIOS. BTIO class C measurements. ROMIO two-phase I/O over PVFS2, Lustre, AHPIOS and the two AHPIOS-based solutions: server-directed I/O and client-directed I/O

The performance and scalability of AHPIOS for an MPI application that writes and reads in parallel, disjoint, contiguous regions of a file, stored over an AHPIOS system for different numbers of AHPIOS servers, has been demonstrated on both commodity clusters and BlueGene/P

supercomputers. We have evaluated the performance of file writes of the BTIO benchmark for four different setups: two-phase I/O over the IBM solution (CIOD), AHPIOS without cache and view-based I/O as collectives (VBIO), AHPIOS and view-based I/O with client-side caching (VBIO-CS), and AHPIOS with view-based I/O with both client-side and I/O node-side caching (VBIO-CS-IONS). Fig. 5 shows the total time breakdown into compute time, file write time, and close time, for BTIO class B and C. The close time is relevant because all data is flushed to the file system when the file is closed. We notice that in all solutions the compute time is roughly the same. VBIO reduces the file write time without any asynchronous transfers. VBIO-CS reduces both the write time and close time, as data is asynchronously written from compute node to I/O node. For VBIO-CS-IONS, the network and I/O activity are almost entirely overlapped with computation. We conclude that the performance of file writes gradually improves with the increasing degree of asynchrony in the system.

4. Energy

Energy has become a major concern for the sustainability of future computer architectures. Providing MPI applications with malleable and energy-aware capabilities allows executing them more efficiently and with less energy requirements, as shown in this section.

Intel SandyBridge chips include the *Running Average Power Limit* (RAPL) interface that provides an energy estimation based on hardware monitoring. Other manufacturers like AMD, IBM and NVIDIA include similar interfaces in their products. We access to this information (via PAPI [24]) to evaluate the application power consumption. Fig. 6 shows the aggregated processor power of Jacobi and Conjugate Gradient running on a compute node consisting of two 6-core Intel Xeon E5-2620 processors. We can observe that the energy consumption is different for each application given that they have different compute and memory intensity levels. There is a sharp increase of power until 12 processes because the available resources (cores) in the system. From 12 to 24 processes the power (and performance) still slightly increase leveraging the processor hyper-threading capabilities.

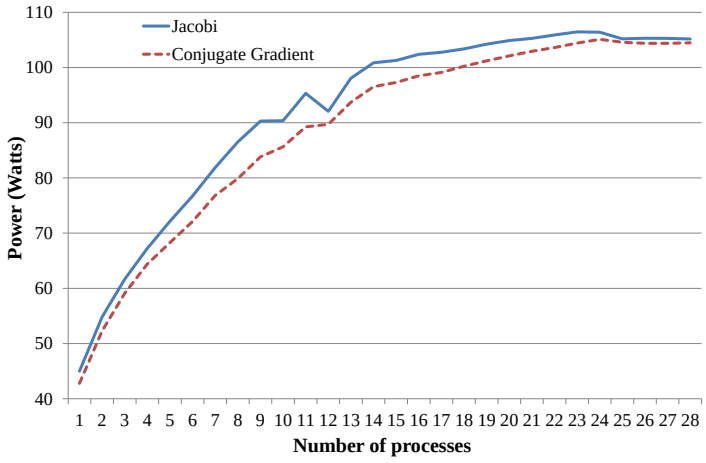
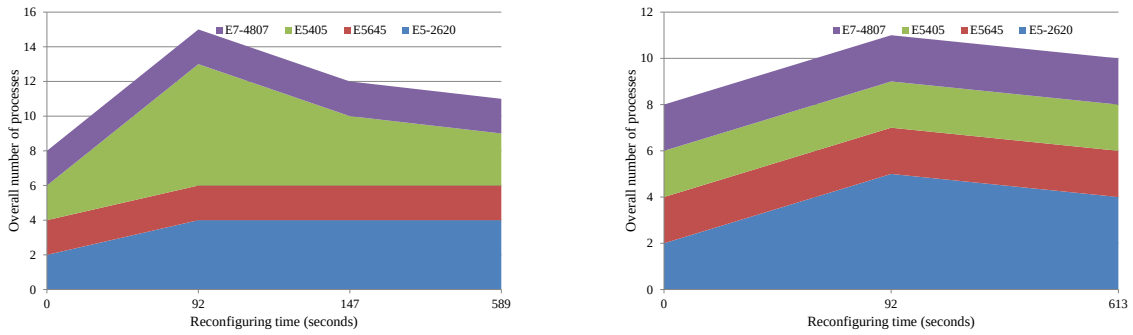


Figure 6. Aggregated CPU energy for Conjugate Gradient and Jacobi executed on a node with two Intel Xeon E5-2620 processors

FLEX-MPI [18] is an MPI extension which provides performance-aware dynamic reconfiguration capabilities to MPI applications. In addition, FLEX-MPI considers two different constraints: cost and energy. The cost constraint consists of reaching a given level of application performance (in FLOPs) at the smallest operational cost (measured in $ per CPU time). In the case of the energy constraint, we aim to reach the performance level with the smallest aggregated energy cost (in Joules) among all the processors involved in the application execution. Note that there are important differences between both constraints. For instance, the operational cost is usually constant for a given processor class, but the energy cost is strongly related to the processor load (as fig. 6 shows).

FlexMPI addresses heterogeneous architectures where each class of nodes has different energy, cost and performance specifications. Finding a solution to the aforementioned problems is usually non trivial given the existing trade-offs between performance and costs (both economic and energetic). For reaching these objectives, we employ a computational prediction model that takes into account both the application and platform characteristics. This model uses hardware counters to characterize the processor power for the considered program under different load scenarios. Later, during the program execution FLEX-MPI uses the PAPI library to survey hardware events (like energy and FLOPS) of each MPI process, and of the MPI interface, to collect the performance of the MPI communications. Based on the collected data, it decides to spawn or remove processes in order to achieve the user-defined performance objectives. In case of a spawn operation, Flex-MPI decides which compute nodes are the most appropriate to run the new created processes. In addition, when the number of processes changes, Flex-MPI also includes functionalities for performing the data redistribution, thereby guaranteeing appropriate load balance among the processes.



a) Efficiency without constraints

b) Efficiency with energy constraint

Figure 7. Number of processes and type of processors scheduled by Flex-MPI for a performance improvement objective of 30% and Jacobi benchmark with 20K input matrix

The results are encouraging, as was demonstrated by executing Jacobi method with two different matrix sizes (10K and 20K rows and columns) and performance improvement objective of 30%. We consider two scenarios for FlexMPI: the first one tries to reach the performance objective without any constraint. The second scenario combines the performance objective with the energy constraint. In our experiments we used a heterogeneous platform with four classes of Intel Xeon nodes: E52620, E5645, E74807 and E5405. The average GFLOP/Watt ratio per core values are respectively 0.13, 0.16, 0.24, and 0.32. Note that the last node is the most

energy-efficient one. Unfortunately, some compute nodes are not Sandy Bridge and it is not possible to measure the energy by hardware counters. For them, we indirectly obtained the energy under different loads by means of an empirical model based on Intel Xeon E5410 [21] and the extrapolation of the values obtained for E5-2620 node.

In our experiments, all the executions with Flex-MPI reached the performance improvement objectives. The reference version (without Flex-MPI) runs two application processes in each compute node type (summarizing 8 processes). For the 10K and 20K matrices the reference version produced an accumulated energy of 30.3 and 120.3 KJoules, respectively. For the efficiency objective (without constraints), the energy values are respectively 31.1 and 124.8 KJoules. For the efficiency objective with energy constraint the energy values are respectively 28.5 and 109.3 KJoules. As fig. 7 shows with an energy efficiency goal, Flex-MPI schedules more dynamic processes on the nodes which have a better performance/energy ratio. In this method, the overall operational energy cost is minimized keeping the performance objectives. This result demonstrates that the combined use of low-level monitoring and malleability at runtime would be a good option for achieving energy efficiency in MPI systems.

5. Applications and algorithms optimizations

As a library, MPI lacks knowledge about the expected behavior of the whole application, the so called"global-view programming model," which prevents certain optimizations that would be possible otherwise. In this section, we show some optimizations that are effective at the global level, and are thus proposed for the application level.

5.1. Hierarchical SUMMA

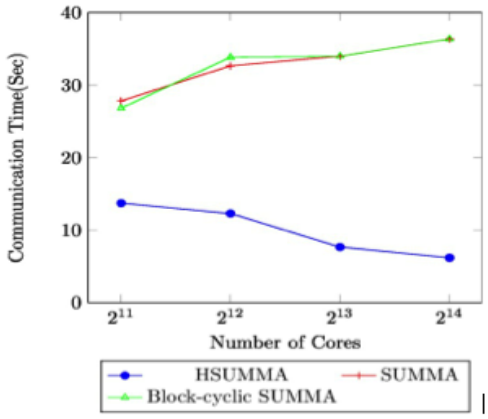


Figure 8. Communication time of SUMMA, block-cyclic SUMMA and HSUMMA on BG/P. p =16K, n = 65,536.

MPI collectives are very important building blocks for many scientific applications. In particular, MPI broadcast is used in many parallel linear algebra kernels such as matrix multiplication, LU factorization, and so on. The state-of-the-art broadcast algorithms used in the most popular MPI implementations were designed in mid 1990s with relatively small parallel systems in mind. Since then, the number of cores in high-end HPC systems has increased by three orders of magnitude and is going to further increase as the systems approach Ultrascale. While some

platform-specific algorithms were proposed later on, they do not address the issue of scale, as they try to optimize the traditional general-purpose algorithms for different specific network architectures and topologies. The first attempt to address the issue of scale is made in [13], where the authors challenge the traditional "flat" design of collective communication patterns in the context of SUMMA, the state-of-the-art parallel matrix multiplication algorithm. They transform SUMMA by introducing a two-level virtual hierarchy into the two-dimensional arrangement of processors. They theoretically prove that the transformed Hierarchical SUMMA (HSUMMA) can significantly outperform SUMMA on large-scale platforms. Their experiments on 16K cores have demonstrated almost a 6x improvement in communication cost, which translated into more than 2-fold speedup of HSUMMA over SUMMA (see fig. 8). Moreover, the optimization technique developed is not architecture or topology specific. While the authors aim to minimize the total communication cost of this application rather than the cost of the individual broadcasts, it has become evident that, despite being developed in the context of a particular application, the resulting technique is not application-bound, ensuring sustainability.

5.2. Application-level optimization of MPI applications with Compiler Support

Programming in MPI requires the programmer to write code from the point-of-view of a single processor/thread, an approach known as fragmented programming. One limiting factor for optimizing MPI is the fact that it is a pure library approach and thus only effective during the execution of the application. A lot of effort has been put into improving the performance of individual functions offered by MPI implementations in order to speed up the execution of MPI applications. These optimizations cannot be performed at the application level because the structure of the underlying program cannot be analyzed or changed by the MPI library in any way. On the other hand, normal compilers have no knowledge about the semantics of MPI function calls either, and thus have to treat them like black boxes—just like all library calls. A compiler which is aware of the semantics of MPI function semantics could (at least to some extent) analyze the behavior of a program along with its communication pattern, in order to optimize both.

We intend to optimize MPI applications by integrating MPI support in the Insieme compiler project [15]. The Insieme compiler framework enables the analysis of a given parallel application and applies source-to-source transformations to improve the overall performance. The output code of the compiler is intended to run within the Insieme runtime system, which provides basic communication primitives optimized for performance. The combination of a compiler and runtime system enables us to transform the program at compile time and also pass information about the program structure to the runtime system for further optimizations during program execution.

Optimizing message passing programs using specialized compilers has already been done long before MPI even existed. Moving communication calls within the code and replacing blocking with non-blocking communication can improve the communication/computation overlap and thus reduce the program execution time. Our approach should go one step further than previous MPI-aware compilers by analyzing high level patterns to find further optimization potential. An example illustrating such a pattern is depicted in the code of Algorithm 3.

An MPI-aware compiler could change the second call from MPI_Bcast to MPI_Ibcast, and thus send B asynchronously while the application is processing the data transmitted during

Algorithm 3 Example pattern for MPI_Bcast

```
MPI_Bcast(A, count, MPI_INT, 0, MPI_COMM_WORLD);
for (int i = 0; i < count; i++) {
        // process A
}
MPI_Bcast(B, count, MPI_INT, 0, MPI_COMM_WORLD);
       // process B similarly
```

the first broadcast, A. Additionally, our compiler can detect that, under some constraints, it would be beneficial to combine both broadcast operations into a single operation to reduce the communication overhead for small messages. Similarly, for larger messages it might decide to break down the message transfers into smaller chunks which will then be processed individually, creating a pipelined broadcast at the application level, as shown below. Transformations like these require program analysis in a compiler and simply cannot be done with a pure library approach.

Algorithm 4 Example pattern for optimized MPI_Bcast

```
for (offset = 0; offset < count; offset += tile_size) {
        MPI_Bcast(&A[offset], tile_size, MPI_INT, 0, MPI_COMM_WORLD);
        for (i = offset; i < offset + tile_size; i++) {
                // process tile of A
        }
}
//process remainder of A and do the same for B
```

5.3. Hybrid MPI-OpenMP Implementations

A hybrid programming solution might be implemented using OpenMP and MPI. Such approaches become more important on modern multi-core parallel systems, decreasing unnecessary communications between processes running on the same node, as well as, decreasing the memory consumption, and improving the load balance. With this implementation, both levels of parallelism, distributed and shared-memory, can be exploited. On one hand, the block-level parallelism is matched by the parallelism between nodes in the cluster (the data is distributed by using MPI). We mention below some of the most representative and efficient Hybrid MPI-OpenMP Implementations.

Molecular Dynamics using DL POLY: DL POLY, a large scale Molecular Dynamics (MD) application programmed using MPI was modified to add a layer of shared memory threading [6], and the code was tested on two multi-core clusters. At smaller core numbers on both systems the pure MPI code outperformed the hybrid message passing and shared memory code. The slower performance of the hybrid code at low core numbers was due to the extra overheads from the shared memory implementation, and the lack of any significant benefit from a reduced communication profile. For more cores on both systems, the hybrid code delivered better performance. In general the hybrid code spent less time carrying out communication than the pure MPI code, performing better at point to point communication at all core counts, and collec-

tive communication at higher core counts. This reduced communication was the main driver for performance improvements in the hybrid code. At low core counts the added overheads from OpenMP parallelization reduced the hybrid code performance, but the effects of these overheads decreased as the number of cores increased. The choice of system interconnect had an effect on the performance of the hybrid code when compared to the pure MPI code. Using a fast Infiniband interconnect the pure MPI code outperformed the hybrid up to a larger number of cores than when using a slower 10 GigE interconnect.

Molecular Dynamics using LAMMPS: Pal et al. [20] developed a computational scheme for MD simulations that exploited thread-parallelism as well as message passing techniques and implemented it on a cluster of 6 dual-quad-core blade servers (SMP nodes), connected using Infiniband and were the challenges and issues of such schemes were discussed in detail. They showed that such a coupled scheme could work nearly twice as fast as a pure message-passing based implementation for certain system sizes, owing to the additional overheads in the latter being circumvented by the former scheme. When using unthreaded MPI processes with this algorithm, the speed-up obtained saturated quickly on the cluster, well before the total number of available cores were utilized. A set of hybrid schemes were compared and were found to be competitive. The authors state that certain code-optimizations and computational loads may favor one particular scheme over the other and hence it is unwise to treat a particular scheme as the best processorthread configuration. However, they found that using unthreaded MPI processes was likely to be inefficient as compared to threaded processes. LAMMPS, which does not spawn threads for parallelization, was found to achieve a speed-up that was significantly inferior to that obtained by their hybrid algorithm. However, the algorithm used for the parallelization was not optimal, and its performance can be enhanced further. There is room for further improvements in the serial algorithm as well.

Adaptive Integral Method: Wei et al [25] presented a hybrid MPI/OpenMP parallelization technique for improving the scalability of classical adaptive integral method (AIM) accelerated classical iterative method of moments (MOM) solvers on multi-core clusters. The schemes they used were based on nested decompositions; a nested column-row decomposition was used for the classical MOM computations and a nested 1-D slab decomposition of the 3-D auxiliary regular grid was used for the AIM acceleration. The scalability of the resulting methods matrix fill time, memory requirement, and matrix solve time were examined theoretically and contrasted to that of a pure MPI parallelization. It was shown that when pure MPI parallelization was used on multi-core clusters, the scalability of both classical and AIM accelerated MOM were limited by two factors: (i) the memory needed for storing replicated geometry/basis function data, and (ii) the communications during the iterative matrix solution. The hybrid MPI/OpenMP parallelization was shown to be useful for both limitations because it did not replicate non-parallelized data structures among different cores of a processor making the memory requirement independent of the number of active cores and because it used fewer messages to communicate larger chunks of data among processors and reduced the impact of latency. For classical MOM, for which the matrix solution can be latency or bandwidth limited, hybrid MPI/OpenMP parallelization always alleviated both of the limiting factors effectively. For AIM accelerated MOM, for which the matrix solution can be grid or latency limited, hybrid MPI/OpenMP parallelization always alleviated the memory limitation but could alleviate the communication limitation only when the matrix solution was latency limited. They concluded that as the performance improvements are a function of the number of active cores in a processor, hybrid parallelization methods are

expected to become more important as the general trend of increasing number of cores in multi- and many-core processors continues.

Drug Discovery: Guerrero et al [12] developed a hybrid optimized version for Virtual Screening calculations in Drug Discovery, that reached up to a 229x speed-up factor, versus its sequential counterpart. On their implementation, threads cooperated in parallel to perform the calculations within each node in a vectorized fashion. Once the data had been distributed using MPI, the calculation of the energy was performed on each node with OpenMP, using its own memory and executing as many threads as the number of cores per node. Moreover, the communication and computation could be overlapped by asynchronous send/receive instructions. Next, MPI was used to move molecule related data between nodes, instead of sending all the information to each core. The communication was reduced by a *ratio of number of cores* per node, with respect to the MPI implementation. This hybrid distributed memory system exhibited good scalability with the number of processors, which is explained by the low number of communications required by the simulations in their hybrid MPI-OpenMP implementation. These hybrid solutions are adequate when the Virtual Screening kernels are computationally intensive and massively parallel in nature, and thus they are well suited to be accelerated on parallel architectures. A natural evolution can also be made with many-core systems located on each node.

Topological Analysis: Cui et al [9] discussed a hybrid MPI/OpenMP approach to implement a parallel processing of topological operations. They showed that implementing an OpenMP application is simpler and quicker than implementing an MPI application. In order to obtain speedup curves for the parallel scheme, they conducted within and overlap operations on a PC cluster. In the first experiment, China County-level Point Data and China County-level Polygon Data were used. In the second experiment, soil type map and land use map in Heilongjiang and Jilin province were used. Experimental performance results demonstrated that a mixed mode code, with the MPI parallelization occurring across the nodes and OpenMP parallelization within the nodes, was more efficient on a cluster as the mode matches the architecture more closely than a pure MPI model.

Conclusions

The MPI design and its different implementations have proven to be a critical piece of the roadmap to faster and more scalable parallel applications. Based on it's past successes, MPI will probably remain a major paradigm for programming distributed memory systems. However, in order to maintain a consistent degree of performance and portability, the revolutionary changes we witness at the hardware level must be mirrored at the software level. Thus, the MPI standard must be in a continuous state of re-examination and re-factoring, to better bridge high-level software constructs with the low-level hardware capability. As software researchers, we need to highlight and explore innovative and even potentially disruptive concepts and match them to alternative, faster, and more scalable algorithms.

In this paper, we have called attention to some MPI-level optimizations that are amenable to providing sustainable support to parallel applications. Hybrid programming models allow developers to use MPI as the upper level distribution mechanism, thus reducing the volume of communication and the memory needed. Adaptive compression allows developers to reduce MPI communications and storage overhead, while AHPIOS is aimed at increasing data locality and reducing I/O latency. Most of the proposals made are transparent to applications or can be made transparent through compiler support. Many more optimizations are possible for applications

that rely on MPI to evolve better programming models, resilience, data management, and energy efficiency mechanisms to reduce overhead, while creating evolving applications. Some of these mechanisms, like RMA, non-blocking, and neighborhood collectives, are introduced in the new MPI 3.0 standard, but the road to Ultrascale is still unpaved.

The work presented in this paper was partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.

References

1. Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. Fti: High performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 32:1–32:32, New York, NY, USA, 2011. ACM. DOI: 10.1145/2063384.2063427.

2. W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J.J. Dongarra. An evaluation of user-level failure mitigation support in mpi. 95:1–14, May 2013. DOI: 10.1007/s00607-013-0331-3.

3. Javier Garcia Blas, Florin Isaila, David E. Singh, and Jesus Carretero. View-based collective I/O for MPI-IO. In *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID '08.*, pages 409–416, May 2008.

4. G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra. Assessing the impact of abft and checkpoint composite strategies. In *16th Workshop on Advances in Parallel and Distributed Computational Models, IPDPS 2014*, Phoenix, AZ, May 2014. DOI: 10.1109/ipdpsw.2014.79.

5. M. Burtscher and Paruj R. fpc: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):1831, 2009. DOI: 10.1109/tc.2008.131.

6. Martin J Chorley and David W Walker. Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. *Journal of Computational Science*, 1(3):168–174, August 2010. DOI: 10.1016/j.jocs.2010.05.001.

7. Jinsuk Chung, Ikhwan Lee, Michael Sullivan, Jee Ho Ryoo, Dong Wan Kim, Doe Hyun Yoon, Larry Kaplan, and Mattan Erez. Containment domains: A scalable, efficient, and flexible resilience scheme for exascale systems. In *the Proceedings of SC12*, November 2012. DOI: 10.1109/sc.2012.36.

8. S. Coco, DArrigo V., and Giunta D. A rice-based lossless data compression system for space. In *2000 IEEE Nordic Signal Processing Symposium*, pages 133–142, May 2000.

9. Shulin Cui and Shuqing Zhang. Parallel processing of topological operations by using a hybrid MPI/OpenMP approach. In *Natural Computation (ICNC), 2013 Ninth International Conference on*, pages 1738–1742, 2013. DOI: 10.1109/icnc.2013.6818263.

10. Kiril Dichev, Vladimir Rychkov, and Alexey Lastovetsky. Two algorithms of irregular scatter/gather operations for heterogeneous platforms. In *Recent Advances in the Message Passing Interface*, pages 289–293. Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-15646-5_31.

11. Rosa Filgueira, Jesus Carretero, David E. Singh, Alejandro Calderon, and Alberto Nuez. Dynamic-COMPI: Dynamic optimization techniques for mpi parallel applications. *The Journal of Supercomputing*, 59(1):361–391, April 2012. DOI: 10.1007/s11227-010-0440-0.

12. Ginés D Guerrero, Horacio Pérez-Sánchez, José M Cecilia, and José M García. Parallelization of virtual screening in drug discovery on massively parallel architectures. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 588–595. IEEE, 2012. DOI: 10.1109/pdp.2012.26.

13. Khalid Hasanov, Jean-Noel Quintin, and Alexey Lastovetsky. Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms. *The Journal of Supercomputing*, pages 1–24, 2014. DOI: 10.1007/s11227-014-1133-x.

14. Florin Isaila, Francisco Javier Garcia Blas, Jesús Carretero, Wei-Keng Liao, and Alok Choudhary. A Scalable Message Passing Interface Implementation of an Ad-Hoc Parallel I/O System. *Int. J. High Perform. Comput. Appl.*, 24(2):164–184, May 2010. DOI: 10.1177/1094342009347890.

15. H. Jordan, P. Thoman, J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch. A multi-objective auto-tuning framework for parallel codes. In *Proc. of the Intl. Conference for High Performance Computing, Networking, Storage and Analysis (SC 2012)*. IEEE Computer Society Press, 2012. DOI: 10.1109/sc.2012.7.

16. Donald E. Knuth. Dynamic huffman coding. *Journal of Algorithms*, 6(2):163180, 1985. DOI: 10.1016/0196-6774(85)90036-7.

17. S. Lyberis, P. Pratikakis, DS. Nikolopoulos, M. Schulz, T. Gamblin, and BR. de Supinski. The myrmics memory allocator: hierarchical,message-passing allocation for global address spaces. In *Proceedings of the International Symposium on Memory Management*. 2012. DOI: 10.1145/2258996.2259001.

18. Gonzalo Martin, Maria-Cristina Marinescu, David E. Singh, and Jesus Carretero. FLEX-MPI: an MPI extension for supporting dynamic load balancing on heterogeneous non-dedicated systems. In *International European Conference on Parallel and Distributed Computing, EuroPar*, 2013. DOI: 10.1007/978-3-642-40047-6_16.

19. M.F.X.J. Oberhumer. *LZO real-time data compression library*. 2012.

20. Anirban Pal, Abhishek Agarwala, Soumyendu Raha, and Baidurya Bhattacharya. Performance metrics in a hybrid MPI–OpenMP based molecular dynamics simulation with short-range interactions. *Journal of Parallel and Distributed Computing*, 74(3):2203–2214, March 2014. DOI: 10.1016/j.jpdc.2013.12.008.

21. M. Pedram and Inkwon Hwang. Power and performance modeling in a virtualized server system. In *2010 39th International Conference on Parallel Processing Workshops (ICPPW),*, pages 520–526, Sept 2010. DOI: 10.1109/ICPPW.2010.76.

22. K. Sato, A. Moody, K. Mohror, T. Gamblin, B.R. de Supinski, N. Maruyama, and S. Matsuoka. Fmi: Fault tolerant messaging interface for fast and transparent recovery. In *Parallel*

and Distributed Processing Symposium, 2014 IEEE 28th International, pages 1225–1234, May 2014. DOI: 10.1109/IPDPS.2014.126.

23. C. Symeonidou, P. Pratikakis, A. Bilas, and DS. Nikolopoulos. Drasync: Distributed region-based memory allocation and synchronization. In *Proceedings of the 20th European MPI Users Group Meeting. EuroMPI 13*, page 4954. ACM, 2013. DOI: 10.1145/2488551.2488558.
24. V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *2012 41st International Conference on Parallel Processing Workshops (ICPPW)*, pages 262–268, Sept 2012. DOI: 10.1109/ICPPW.2012.39.
25. Fangzhou Wei and Ali E Yilmaz. Parallel Computing. *Parallel Computing*, 37(6-7):279–301, July 2011.
26. Robert Zigon. Run length encoding. *Dr. Dobb's Journal*, 14(2):126128, 1989.

Received February 11, 2015.

DOI: 10.14529/jsfi150203

Resilience within Ultrascale Computing System: Challenges and Opportunities from Nesus Project

Pascal Bouvry[1], *Rudolf Mayer*[2], *Jakub Muszyński*[1], *Dana Petcu*[3], *Andreas Rauber*[4], *Gianluca Tempesti*[5], *Tuan Trinh*[6], *Sébastien Varrette*[1]

Although resilience is already an established field in system science and many methodologies and approaches are available to deal with it, the unprecedented scales of computing, of the massive data to be managed, new network technologies, and drastically new forms of massive scale applications bring new challenges that need to be addressed. This paper reviews the challenges and approaches of resilience in ultrascale computing systems from multiple perspectives involving and addressing the resilience aspects of hardware-software co-design for ultrascale systems, resilience against (security) attacks, new approaches and methodologies to resilience in ultrascale systems, applications and case studies.

Keywords: high performance computing, fault tolerance, algorithm-based fault tolerance, extreme data, evolutionary algorithm, ultrascale computing system.

Introduction

Ultrascale computing is a new computing paradigm that comes naturally from the necessity of computing systems that should be able to handle massive data in possibly very large scale distributed systems, enabling new forms of applications that can serve a very large amount of users and in a timely manner that we have never experienced before.

Ultrascale Computing Systems (UCSs) are envisioned as large-scale complex systems joining parallel and distributed computing systems that will be two to three orders of magnitude larger than today's systems (considering the number of Central Processing Unit (CPU) cores). It is very challenging to find sustainable solutions for UCSs due to their scale and a wide range of possible applications and involved technologies. For example, we need to deal with cross fertilization among HPC, large-scale distributed systems, and big data management. One of the challenges regarding sustainable UCSs is resilience. Traditionally, it has been an important aspect in the area of critical infrastructure protection (e.g. traditional electrical grid and smart grids). Furthermore, it has also become popular in the area of information and communication technology (ICT), ICT systems, computing and large-scale distributed systems. In essence, resilience is an ability of a system to efficiently deliver and maintain (in a timely manner) a correct service despite failures and changes. It is important to emphasize this term in comparison with a closely related "fault tolerance". The latter indicates only a well-defined behaviour of a system once an error occurs. For example, a system is resilient to an effect on an error (in one of its components) if it continues correct operation and service delivery (possibly degraded in some way). Whereas, it is fault tolerant to the error when it is able to detect and notify about the existence of the problem with possible recovery to the correct state.

[1] University of Luxembourg, Esch-sur-Alzette, Luxembourg
[2] Secure Business Austria, Vienna, Austria
[3] West University of Timişoara, Timişoara, Romania
[4] Vienna University of Technology, Vienna, Austria
[5] University of York, York, UK
[6] Budapest University of Technology and Economics, Budapest, Hungary

The existing practices of dependable design deal reasonably well with achieving and predicting dependability in systems that are relatively closed and unchanging. Yet, the tendency to make all kinds of large-scale systems more interconnected, open, and able to change without new intervention by designers, makes existing techniques inadequate to deliver the same levels of dependability. For instance, evolution of the system itself and its uses impairs dependability: new components "create" system design faults or vulnerabilities by feature interaction or by triggering pre-existing bugs in existing components; likewise, new patterns of use arise, new interconnections open the system to attack by new potential adversaries, and so on.

Many new services and applications will be able to get advantage of ultrascale platforms such as big data analytics, life science genomics and HPC sequencing, high energy physics (such as QCD), scalable robust multiscale and multi-physics methods and diverse applications for analysing large and heterogeneous data sets related to social, financial, and industrial contexts. These applications have a need for Ultrascale Computing Systems (UCSs) due to scientific goals to simulate larger problems within a reasonable time period. However, it is generally agreed that applications will require substantial rewriting in order to scale and benefit from UCSs.

In this paper, we aim at providing an overview of Ultrascale Computing Systems (UCSs) and highlighting open problems. This includes:

- Exploring and reviewing the state-of-the-art approaches of continuous execution in the presence of failures in UCSs.
- Techniques to deal with hardware and system software failures or intentional changes within the complex system environment.
- Resilient, reactive schedulers that can survive errors at the node and/or the cluster-level, cluster-level monitoring and assessment of failures with pro-active actions to remedy failures before they actually occur (like migrating processes [13, 58], virtual machines [49], etc.), and malleable applications that can adapt their resource usage at run-time.

In particular, we approach the problem from different angles including fundamental issues such as reproducibility, repeatability and resiliency against security attacks; application-specific challenges such as hardware and software issues with big data, cloud-based cyber physical systems.The paper then also discusses new opportunities in providing and supporting resilience in ultrascale systems.

This paper is organized as follows: the section 1 reviews the basic notions of faults, Fault Tolerance and robustness. Then, several key issues need to be tackled to ensure a robust execution on top of an UCS system. The section 2 focuses on recent trends as regards the resilience of large scale computing systems, focusing on hardware failures (see §2.1) and on Algorithm-Based Fault Tolerance (ABFT) techniques where the fault tolerance scheme is tailored to the algorithm *i.e.* the application run. We will see that at this level, Evolutionary Algorithms (EAs) present all the characteristics to handle natively faulty executions, even at the scale foreseen in UCSs systems. Then, the section 3 will review the challenges linked to the notion of repeatibility and reproducibility in UCSs. The final section concludes the paper and provides some future directions and perspectives opened by this study.

1. Faults, Fault Tolerance and Robustness

Due to their inherent scale, UCSs are naturally prone to errors and failures which are no longer rare events [11, 12, 50, 52]. There are many sources of *faults* in distributed computing and they are inevitable due to the defects introduced into the system at the stages of its design,

construction or through its exploitation (e.g. software bugs, hardware faults, problems with data transfer) [4, 11, 12, 52]. A fault may occur by a deviation of a system from the required operation leading to an *error* (for instance a software bug becomes apparent after a subroutine call). This transition is called a fault activation, i.e. a *dormant* fault (not producing any errors) becomes *active*. An error is *detected* if its presence is indicated by a message or a signal, whereas not detected, present errors are called *latent*. Errors in the system may cause a (service) *failure* and depending on its type, successive faults and errors may be introduced (*error/failure propagation*). The distinction between faults, errors and failures is important because these terms create boundaries allowing analysis and coping with different threats. In essence, faults are the cause of errors (reflected in the state) which without proper handling may lead to failures (wrong and unexpected outcome). Following these definitions, *fault tolerance* is an ability of a system to behave in a well-defined manner once an error occurs.

1.1. Fault models for distributed computing

There are five specific fault models relevant in distributed computing: *omission*, *duplication*, *timing*, *crash*, and *byzantine failures* [53].

Omission and *duplication* failures are linked with problems in communication. Send-omission corresponds to a situation, when a message is not sent; receive-omission — when a message is not received. Duplication failures occur in the opposite situation — a message is sent or received more than once.

Timing failures occur when time constraints concerning the service execution or data delivery are not met. This type is not limited to delays only, since too early delivery of a service may also be undesirable.

The *crash* failure occurs in four variants, each additionally associated with its persistence. Transient crash failures correspond to the service restart: amnesia-crash (the system is restored to a predefined initial state, independent on the previous inputs), partial-amnesia-crash (a part of the system stays in the state before the crash, where the rest is reset to the initial conditions), and pause-crash (the system is restored to the state it had before the crash). Halt-crash is a permanent failure encountered when the system or the service is not restarted and remains unresponsive.

The last model — *byzantine* failure (also called *arbitrary*) — covers any (very often unexpected and inconsistent) responses of a service or a system at arbitrary times. In this case, failures may emerge periodically with varying results, scope, effects, etc. This is the most general and serious type of failure [53].

1.2. Dependable computing

Faults, errors and failures are *threats* to system's *dependability*. A system is described as dependable, when it is able to fulfil a contract for the delivery of its services avoiding frequent downtimes caused by failures.

Identification of threats does not automatically guarantee dependable computing. For this purpose, four main groups of appropriate methods have been defined [4]: *fault prevention*, *fault tolerance*, *fault removal*, and *fault forecasting*. As visible in fig. 1, all of them can be analysed from two points of view — either as means of avoidance/acceptance of faults or as approaches to support/assess dependability. Fault tolerance techniques aim to reduce (or even eliminate) the

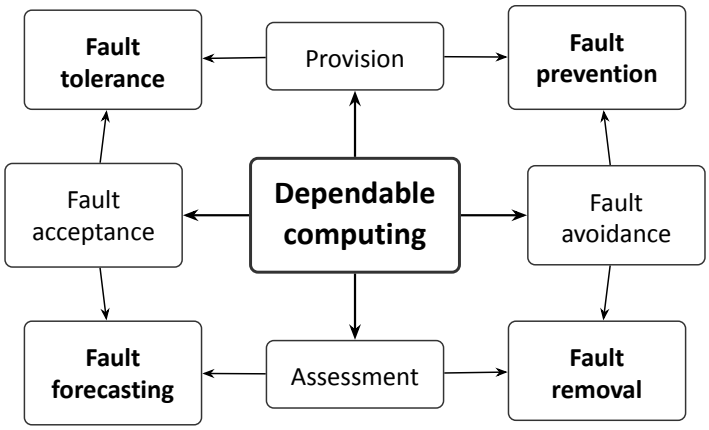


Figure 1. Means for dependable computing

amount of service failures in the presence of faults. The main goal of *fault prevention* methods is to minimize the number of faults occurred or introduced through usage and enforcement of various policies (concerning usage, access, development etc.) The next group — *fault removal* techniques — is concentrated around testing and verification (including formal methods). Finally, *fault forecasting* consists of means to estimate occurrences and consequences of faults (at a given time and later).

1.3. Fault tolerance

Fault tolerance techniques may be divided into two main, complementary categories [4]: *error detection*, and *recovery*. Error detection may be performed during normal service operation or while it is suspended. The first approach in this category — *concurrent detection* — is based on various tests carried out by components (software and/or hardware) involved in the particular activity or by elements specially designated for this function. For example, a component may calculate and verify checksums for the data which is processed by it. On the other hand, a firewall is a good illustration of a designated piece of hardware (or software) oriented on detection of intrusions and other malicious activities. *Preemptive detection* is associated with the maintenance and diagnostics of a system or a service. The focus in this approach is laid on identification of latent faults and dormant errors. It may be carried out at a system startup, at a service bootstrap, or during special maintenance sessions.

After an error of a fault is detected, recovery methods are applied. Depending on the problem type, *error or fault handling* techniques are used. The first group is focused on elimination of errors from the system state, while the second are designed to prevent activation of faults. In [4], the specific methods are separated from each other, where in practice this boundary is fuzzy and depends on the specific service and system types.

Generally, error handling is solved through:

1. *Rollback* [22] — the system is restored to the last known, error-free state. The approach here depends on a method used to track the changes of the state. A well known technique is *checkpointing* — the state of a system is saved periodically (e.g. the snapshot of a process is stored on a disk) as a potential recovery point in the future. Obviously, this solution is not straightforward in the case of distributed systems and there are many factors to consider. In such environment, checkpointing can be coordinated or not — with differences in reliability and the cost of synchronisation of the distributed components (for details see: [18, 31, 53]).

Rollback can be also implemented through the *message logging*. In this case, the communication between the components is tracked rather than their state. In case of an error, the system is restored by *replaying* the historical messages, allowing it to reach global consistency [53]. Sometimes both techniques are treated as one, as usually they complement each other.

2. *Rollforward* — the current, erroneous system state is discarded and replaced with a one newly created and initialised.
3. *Compensation* — solutions based on components' *redundancy* and *replication*, sometimes referred to as *fault masking*. In the first case, additional components (usually hardware) are kept in reserve [31]. If failures or errors occur, they are used to compensate the losses. For example, a connection to the Internet of a cloud platform should be based on solutions from at least two different Internet Service Providers (ISPs).

   Replication is based on the dispersion of multiple copies of the service components. A schema with replicas used only for the purpose of fault tolerance is called a *passive (primary-backup) replication* [31]. On the other hand, an *active replication* is when the replicas participate in providing the service, leading to increased performance and applicability of load balancing techniques [31]. Coherence is the major challenge here, and various approaches are used to support it. For instance, read-write protocols are crucial in active replication, as all replicas have to have the same state. Another worth to note example is clearly visible in volunteer-based platforms. An appropriate selection policy of the correct service response is needed when replicas return different answers, i.e. a method to reach quorum consensus is required [31].

These techniques are not exclusive and can be used together. If the system can not be restored to a correct state thanks to the compensation, rollback may be attempted. If this fails, then rollforward may be used.

The above mentioned methods may be referred to as *general-purpose* techniques. These solutions are relatively generic, which aid their implementation for almost any distributed computation. It is also possible to delegate responsibility for fault tolerance to the service (or application) itself, allowing tailoring the solution for specific needs — therefore forming an *application-specific* approach. A perfect example in this context is ABFT, originally applied to distributed matrix operations [14], where original matrices are extended with checksums before being scattered among the processing resources. This allows detection, location and correction of certain miscalculations, creating a *disk-less checkpointing* method. Similarly, in certain cases it is possible to continue the computation or the service operation despite the occurring errors. For instance, unavailable resource resulting from a crash-stop failure can be excluded from further use. In this work, the idea will be further analysed and extended to the context of byzantine errors and the nature-inspired distributed algorithms.

Fault handling techniques are applied after the system is restored to an error-free state (using the methods described above). As the aim now is to prevent future activation of detected faults, four subgroups according to the intention of the operation may be created. These are [4]: *diagnosis* (the error(s) are identified and their source(s) are located), *isolation* (faulty components are logically or physically separated and excluded from the service), *reconfiguration* (the service/platform is reconfigured to substitute or bypass the faulty elements), and *reinitialization* (the configuration of the system is adapted to the new conditions).

1.4. Robustness

When a given system is resilient to a given type of fault, one generally claims that this system is *robust*. Yet defining rigorously robustness is not an easy task and many contributions come with their own interpretation of what robustness is. Actually, there exists a systematic framework that permits to define a robust system unambiguously. In fact, this should be probably applied to any system or approach claiming to propose a fault-tolerance mechanism. This framework, formalized in [57], answers the following three questions:

1. What behavior of the system makes it robust?
2. What uncertainties is the system robust against?
3. Quantitatively, exactly how robust is the system?

The first question is generally linked to the technique or the algorithm applied. The second — explicitly lists the type of faults or disturbing elements targeted by the system. Answering it is critical to delimit the application range of the designed system and avoid counter examples selected in a context not addressed by the robust mechanism. The third question is probably the most difficult to answer, and at the same time the most vital to characterize the limits of the system. Indeed, there is nearly always a threshold on the error/fault rate above which the proposed infrastructure fails to remain robust and breaks (in some sense).

2. Resiliency within UCSs: From Hardware to ABFT Trends

2.1. Lessons Learned from Big Data Hardware

One of the direct consequences of the treatment of big data is, clearly, the requirement for extremely high processing power. And whereas research in the big data domain does not traditionally include research in processor and computer architecture, there is a clear correlation between the advances in the two domains. While it is obviously difficult to predict future developments in processing architectures with high accuracy, we have identified two major trends that are likely to affect big data processing: the development of *many-core devices* and *hardware/software codesign*.

The many-core approach represents a step-change in the number of processing units available either in single devices or in tightly-coupled arrays. Exploiting techniques and solutions derived from the Network-on-Chip (NoC) [32] and Graphical Processing Units (GPU) areas, many-core systems are likely to have a considerable impact on application development, pushing towards distributed memory and data-flow computational models. At the same time, the standard assumption of "more tasks than processors" will be loosened (or indeed inverted), reducing to some extent the complexity of processes such as task mapping and load balancing.

Hardware/software co-design implies that applications will move towards a co-synthesis of hardware and software: the compilation process will change to generate at the same time the code to be executed by a processor and one or more hardware co-processing units to accelerate computation. Intel and ARM have already announced alliances with Altera and Xilinx[7], respectively, to offer tight coupling between their processors and reconfigurable logic, while Microsoft recently introduced the reconfigurable Catapult system to accelerate its Bing servers [47].

These trends, coupled with the evolution of VLSI fabrication processes (the sensitivity of a device to faults increases as feature size decreases), introduce new challenges to the application

[7] `http://www.eejournal.com/archives/articles/20140624-intel`.

of fault tolerance in the hardware domain. In addition to increasing the probability of fabrication defects (not directly relevant to this article), the heterogeneous nature of these systems and their extreme density represent major challenges to reliability. Indeed, the notion of *hardware fault* itself is being affected and extended to include a wider variety of effects, such as variability and power/heat dissipation.

This section does not in any way claim to represent an exhaustive survey of this very complex area, nor even a thorough discussion of the topic, but rather wants to provide a brief "snapshot" of a few interesting approaches to achieve fault tolerance in hardware, starting with a brief outline of some key concepts and fundamental techniques.

2.2. Fault tolerance in digital hardware

One of the traditional classification methods subdivides online faults in hardware systems (i.e. faults that occur during the lifetime of a circuit, rather than at fabrication) into two categories: *permanent* and *transient* (a third category, *intermittent* faults, is outside the scope of this discussion).

Permanent faults are normally introduced by irreversible physical damage to a circuit (for example, short circuits). Rather common in fabrication, they are rare in the lifetime of a circuit, but become increasingly less so as circuits age. Once a permanent fault appears, it will continue to affect the operation of the circuit forever.

Transient faults have limited duration and will disappear with time. By far the most common example of transient faults is *Single-Event Upsets* (SEU), where radiation causes a change of state within a memory element in a circuit.

This distinction is highly relevant in the context of fault tolerance, defined as the ability of a system to operate correctly in the presence of faults. Generally, the design of a fault tolerant hardware system involves four successive steps [1]:

1. *Fault detection*: can the system detect the presence of a fault?
2. *Fault diagnosis or localization*: can the system identify (as precisely as needed) the exact nature and location of a fault?
3. *Fault limitation or containment*: can the impact of the fault on the operation of the system be circumscribed so that no irreversible damage (to the circuit or to the data) results?
4. *Fault repair*: can the functionality of the system be recovered?

While there is usually no significant difference between transient and permanent faults in the first three steps, the same does not apply to the last step: transient faults can allow the full recovery of the circuit functionality, whereas *graceful degradation* (e.g., [15]) is normally the objective in the case of permanent faults in the system.

2.3. Fundamental techniques

Any implementation of fault tolerance (or indeed of fault detection) in hardware implies, directly or indirectly, the use of *redundancy*. Specific applications of redundancy, however, vary significantly depending on the features of the hardware system. In general, three "families" of redundant techniques can be identified. Once again, the examples presented in this section are not meant to be exhaustive, but simply to illustrate the different ways in which redundancy can be applied in the context of fault tolerance in hardware systems.

2.3.1. Data or information redundancy

This type of techniques relies on the use of non-minimal coding to represent the data in a system. By far the most common implementation of data redundancy implies the use of *error detecting codes* (EDC), when the objective is fault detection, and of *error correcting codes* (ECC), when the objective is fault tolerance [1].

It is worth highlighting that, even though these techniques rely on information redundancy, they also imply considerable hardware overhead, not only due to the requirement for additional storage (due to the non-minimal encoding), but also because the computation of the additional redundant bits implies the presence of (sometimes significant) additional logic.

2.3.2. Hardware redundancy

Hardware redundancy techniques exploit additional resources more directly to achieve fault detection or tolerance. In the general case, the best-known hardware redundancy approaches exploit duplication (Double Modular Redundancy, or DMR) for fault detection or triplication (Triple Modular Redundancy, or TMR) for fault tolerance.

TMR in particular is a widely used technique for safety critical systems: three identical systems operate on identical data, and a 2-out-of-3 voter is used to detect faults in one system and recover the correct result from the others. In its most common implementations (for example, in space missions), TMR is usually applied to complete systems, but the technique can operate at all levels of granularity (for example, it would be possible, if terribly inefficient, to design TMR systems for single logic gates).

2.3.3. Time redundancy

This type of approaches relies, generally speaking, on the repetition of computation and the comparison of the results between the different runs. In the simplest case, the same computation is repeated twice to generate identical results, allowing the detection of SEU. More sophisticated (but less generally applicable) approaches introduce differences in two executions (e.g. by inverting input data or shifting operands) in order to be able to detect permanent faults as well.

It is worth noting that time redundancy techniques are rarely used when fault tolerance is sought (being essentially limited to detection) but in theory can be extended to allow it in case of transient faults.

2.4. Fault tolerant design

In the introduction to this section, we highlighted how the heterogeneity and density of a type of devices that are likely to become relevant in big data treatment complicates considerably the task of achieving fault tolerant behaviour in hardware.

In particular, the heterogeneity introduced by the presence of programmable logic and the complexity of many-core devices implies that the notion of a single approach to fault tolerance applicable to every component of a system will have to be replaced by *ad-hoc* techniques. What follows is a short list of the main components of a complete system, followed by a brief analysis of their fault tolerance requirements and a few examples of approaches developed to achieve this goal, in order to illustrate some of the issues and difficulties that will have to be met.

2.4.1. Memories

Memory elements are probably the hardware components that require the highest degree of fault tolerance: their extremely regular structure implies that transistor density in memories is substantially greater than in any other device (the largest memory device commercial available in 2015 reaches a transistor count of almost 140 billion, compared for example to 4.3 billion of the largest processor). This level of density has resulted in the introduction of fault tolerant features even in commonly available commercial memories.

Reliability in memories takes essentially two forms: to protect against SEUs, the use or redundant ECC bits associated with each memory word is common and well-advertised [39], while marginally less known is the use of spare memory locations to replace permanently damaged ones. The latter technique, used extensively at fabrication for laser-based permanent reconfiguration, has also been applied in an on-line self-repair setting [16].

2.4.2. Programmable logic

Programmable logic devices (generally referred to as Field Programmable Gate Arrays) are regular circuits that can reach extremely high transistor counts. In 2015, the largest commercial FPGA device (the Virtex-Ultrascale XCVU440 by Xilinx) contains more than 20 billion transistors.

The regularity of FPGAs has sparked a significant amount of research into self-testing and self-repairing programmable devices since the late 1990s [2, 34, 37], but to the best of our knowledge this research has yet to impact consumer products (even considering potential fabrication-time improvement measures similar to those described in the previous section for memories), with the exception of radiation-hardening for space applications.

In reality, the relationship between programmable logic and fault tolerance would merit a more complete analysis, since the interplay between the fabric of the FPGA itself and the circuit that is implemented within the fabric can lead to complex interactions. Such an analysis is however beyond the scope of this article. Interestingly in this context, even though its FPGA fabric itself does not appear to contain explicit features for fault-tolerance, Xilinx supports a design tool to allow a degree of fault-tolerance in implemented designs through its Isolation Design Flow, specifically aimed at fault containment.

2.4.3. Single processing cores

The main driving force in the development of high-performance processors has been, until recently at least, sheer computational speed. In the last few years, power consumption has become an additional strong design consideration, particularly since the pace of improvements in performance has started to slow. Since fault tolerance, with its redundancy requirements, has negative implications both for performance and for power consumption, relatively little research into fault tolerant cores has reached the consumer market.

The situation is somewhat different outside of the high-performance end of the spectrum, where examples of processors specifically designed for fault tolerance exist (for example, the NGMP processor developed on behalf of the European Space Agency [3] or the Cortex-R series by ARM), demonstrating at least the feasibility of such implementations.

More recently, the RAZOR approach [23] represents a fault tolerance technique aimed specifically at detecting (and possibly correcting) timing errors within processor pipelines using a particular kind of time redundancy approach that exploits delays in the clock distribution lines.

2.4.4. On-chip networks

Networks are a crucial element of any system where processors have to share information, and therefore represent a fundamental aspect not only of many-core devices, but also of any multiprocessor system. Often rivalling in size and complexity with the processing units themselves, networks and their routers have traditionally been a fertile ground for research on fault tolerance.

Indeed, even when limiting the scope of the investigation to on-chip networks, numerous books and surveys exist that classify, describe, and analyse the most significant approaches to fault tolerance (for example, [7, 45, 48]). Very broadly, most of the fundamental redundancy techniques have been applied, in one form or another, to the problem of implementing fault-tolerant on-chip networks, ranging from data redundancy (e.g. parity or ECC encoding of transmitted packets), through hardware redundancy (e.g. additional routing logic), to time redundancy (e.g. repeated data transmission).

2.4.5. Many-Core arrays

An accurate analysis of fault tolerance in many-core devices is of course hampered by the lack of commercially-available devices (the Intel MIC Architecture, based on Xeon Phi co-processors, is a step in this direction but at the moment is limited to a maximum of 61 cores and relies on conventional programming models within a coarse-grained architecture). Using transistor density as a rough indicator of the fault sensitivity of a device (keeping in mind that issue related to heat dissipation can be included in the definition), it is no surprise that fault tolerance is generally considered as one of the key enabling technologies for this type of device: once again, the regular structure of many-core architecture is likely to have a significant impact on transistor count. Today, for example, transistor count in GPUs (the commercial devices that, arguably, bear the closest resemblance to many-core systems, both for the number of cores and for the programming model) is roughly twice that of Intel processors using similar fabrication processes, and even in the case of Intel this type of density was achieved only in multi-core (and hence regular) devices.

The lack of generally accessible many-core platforms implies that most of the existing approaches to fault-tolerance in this kind of systems remain at a somewhat higher abstraction layer and typically rely on mechanisms of task remapping through OS routines or dedicated middleware [9, 10, 33, 59]. Specific hardware-level approaches, on the other hand, have been applied to GPUs (e.g., [55]) and could have an impact on many-core systems. Indeed, one of the few prototype many-core platforms with a degree of accessibility (ClearSpeed's CSX700 processor) boasts a number of dedicated hardware error-correction mechanisms, hinting at least to the importance of fault tolerance in this type of devices, whereas no information is available on fault-tolerance mechanisms in the Intel MIC architecture.

2.5. Toward Inherent Software Resilience: ABFT nature of EAs

Evolutionary Algorithms (EAs) are a class of solving techniques based on the Darwinian theory of evolution [19] which involves the search of a *population* of solutions.

A set of recent studies [20, 26, 30, 35, 42, 43] illustrate what seems to be a natural resilience of EAs against a model of destructive failures (crash failures). With a properly designed execution, the system experiences a *graceful degradation* [35]. This means, that up to some threshold and despite the failures, the results are still delivered. However, it either requires more time for the execution or the returned values are further from the optimum being searched.

3. Repeatibility and Reproducibility Challenges in UCSs

Repeatibility and reproducibility are important aspects of sound scientific research in all disciplines – yet more difficult to achieve than might be expected [5, 54]. Repeatability of the experiments denotes the ability to repeat the same experiment and achieve the same result, by the original investigator [56]. On the other hand, reproducibility enables the verification of the validity of the conclusions and claims drawn from scientific experiments by other researchers, independent from the original investigator.

Repeatibility is essential for all evidence-based sciences, as counterpart to formal proofs used in theoretical sciences or discourse used widely in e.g. the humanities. It is a key requirement for all sciences and studies relying on computational processes. The challenge of achieving repeatability, however, increases drastically with the complexity of the underlying computational processes, making the characteristics of the processes less intuitive to grasp and interpret and verify. It thus becomes an enormous challenge in the area of ultrascale computing given the enormous complexity of the massive amount of computing steps involved, and the numerous dependencies of an algorithm performing on a stack of software and hardware of considerable complexity.

While many disciplines have, over sometimes long-time periods, established a set of good practices for repeating and verifying their experiments (e.g. by using experiment logbooks in disciplines such as chemistry or physics, where the researchers record their experiments), computational science lags behind, and many investigations are hard to repeat or reproduce [17]. This can be attributed to the lower maturity of computer science methods and practices in general, the fast-moving pace of changing technology that is utilised to perform the experiments, or the multitude of different software components needed to interact to perform the experiments. Small variations in, for example, the version of a specific software can have a great impact on the final result which might deviate significantly from the expected outcome, as has promoinently been shown e.g. for the analysis of CT scans in the medical domain [27]. More severely, the source of such changes might not even be in the software that is specifically used for a certain task, but somewhere further down the stack of software the application depends on, including for example the operating system, system libraries or the very specific hardware environment being used. Recognizing these needs, steps are being taken to assist research in ensuring their results are more easily reproducible [24, 25]. The significant overhead in providing enough documentation to allow an exact reproduction of the experiment setup further adds to these difficulties.

Technical solutions to increase reproducibility in eScience research cover several branches. One type of solutions is aimed at recreating the technical environments where experiments are executed in. Simple approaches towards this goal include virtualising the complete environment the experiment is conducted in, e.g. by making a clone which can subsequently be redistributed and executed in Virtual Machines. Such approaches only partially allow reproducibility, as the cloned system is potentially containing many more applications than are actually needed, and no identification of which components are actually required is provided. Thus, a more favourable

approach is to recreate only the needed parts of the system. Code, Data, and Environment (CDE) [28] is such an approach, as it detects the required components during the runtime of a process. CDE works on Linux operating system environments and requires the user to prepend his commands to scripts or binaries by the cde command. CDE will then intercept system calls and gather all the files and binaries that were used in the execution. A packaged created thereof can then be transferred to a new environment.

CDE has a few shortcomings especially in distributed system set-ups. External systems may be utilised, e.g. by calling Web Services, taking over part of the computational tasks. CDE does not aim at detecting these calls. This challenge is exacerbated in more complex distributed set-ups such as may be encountered in ultra-scale computational environments.

Not only external, but also calls to local service applications are an issue. These are normally running in the background and started before the program execution, and thus not all the sources that are necessary to run them are detected. It is more problematic, though, that there is no explicit detection of such a service being a background or remote service. Thus, the fact that the capturing of the environment is incomplete remains unnoticed to users who are not familiar with all the details of the implementation. The Process Migration Framework [8] (PMF) is a solution similar to CDE, but takes specifically the distributed aspect into account.

Another approach to enable better repeatibility and reproducibility is in using standardised methods and techniques to author experiments, such as the use of *workflows*. Workflows allow a precise definition of the involved steps, the required environment, and the data flow between components. The modelling of workflows can be seen as an abstraction layer, as they describe the computational ecosystem of the software used during a process. Additionally, they provide an execution environment that integrates the required components to perform a process and execute all defined subtasks. Different scientific workflow management systems exist that allow scientists to combine services and infrastructure for their research. The most prominent examples of such systems are Taverna [41] and Kepler [36]. Vistrails [51] also adds versioning and provenance of the creation of the workflow itself. The Pegasus workflow engine [21] specifically aims at scalability, and allows executing workflows in cluster, grid and cloud infrastructures.

Building on top of workflows, the concept of workflow-centric Research Objects [6] (ROs) tries to describe research workflows in the wider eco-system they are embedded in. ROs are a means to aggregate or bundle resources used in a scientific investigation, such as a workflow, provenance from results of its execution, and other digital resources such as publications or data-sets. In addition, annotations are used to describe these objects further. A digital library exists for Workflows and Research Objects to be shared, such as the platform *my experiment*.[8]

Workflows facilitate many aspects of reproducibility. However, unless experiments are designed from the beginning to be implemented as a workflow, there is a significant overhead to migrate an existing solution to a workflow. Furthermore, workflows are normally limited in features they support, most prominently in the type of programming languages. Thus, not all experiments can be easily implemented in such a manner.

A model to describe a scientific process or experiment is presented in [38]. It allows the researcher to describe their experiments in a manner similar to the approach of Research Objects; however, this model is independent of a specific workflow engine, and provides a more refined set of concepts to specifiy the software and hardware setup utilised.

[8] http://www.myexperiment.org/

Another important aspect of reproducibility is the verification of the results obtained. The verification and validation of experiments aims to prove whether the replicated or repeated experiment has the same characteristics and performs in the same way as the original experiment – even if the original implementation is faulty. Simple approaches just comparing a final experiment outcome, e.g. a performance measure of a machine learning experiment, doesn't provide sufficient evidence on this task, especially in settings where probabilisitic learning is utilised, and also a close approximation of the original result would be accepted. Furthermore, a comparison on final outcomes provides no means to trace where potential deviations originated in the experiment. It is therefore required to analyze the characteristics of an experimental process wrt. to its significant properties, its determinism, and levels where significant states of a process can be compared. On further needs to define a set of measurements to be taken during an experiment, and identify approapriate metrics to compare values obtained from different experiment executions, beyond simple similarity [44]. A framework to formalise such a verification has been introduced as the VFramework [40], which is specifically tailored to process verification. It describes what conditions must be met and what actions need to be taken in order to compare the executions of two processes.

In the context of ultrascale computing, the distributed nature of large experiments poses challenges for repeating the results, as there are many potential sources for errors, and an increased demand for documentation. Also, approaches such as virtualisation or recreation of computing environments hit the boundaries of feasibility especially in larger distributed settings based on grid our cloud infrastructure, where the number of nodes to be stored becomes difficult to manage. Another challenge in ultrascale computing is in the nature of computing hardware utilised, which is often highly specialised towards certain tasks, and much more difficult to be captured and recreated in other settings.

Last, but not least, ultrascale computing is usually also tightly linked with massive volumes of data that need to be kept available and identifiable in sometimes highly dynamic environments. Proper data management and preservation have been prominently called for [29]. A key aspect in ultra-scale computing in this context are means to persistently identify the precise versions and subsets of data having been used in an experiment. The Working Group on Dynamic Data Citation of the Research Data Alliance[9] has been developing recommendations how to achieve such a machine-actionable citation mechanism for dynamic data that is currently being evaluated in a number of pilots [46].

Conclusion

In this paper, we first proposed an overview of resilient computing in Ultrascale Computing Systems, i.e., cross-layered techniques dealing with hardware and software failures or attacks, but also the necessary services including security and repeatability. We also described how new application needs such as big data and cyber-physical systems challenge existing computing paradigms and solutions.

New opportunities have been highlighted but they certainly require further investigations and in particular large-scale experiments and validations. What emerges is the need for the apparition of additional disruptive paradigms and solutions at all levels: from hardware, languages, compilers, operating systems, middleware, services, and application-level solutions. Offering a

[9] https://rd-alliance.org/groups/data-citation-wg.html

global view on the reliability/resilience issues will allow to define the right level of information exchange between all layers and components in order to have global (cross-layer/component) solution. Additional objectives such as performance, energy-efficiency and cost also need to be taken into account. We intend in the context of the COST NESUS project to have several focus groups aimed at defining more precisely the problems and approaches for such challenges.

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

1. M. Abd-El-Barr. *Design and Analysis of Reliable and Fault-tolerant Computer Systems.* Imperial College Press, 2007. DOI: 10.1142/9781860948909.
2. M. Abramovici, C. Strond, C. Hamilton, S. Wijesuriya, and V. Verma. Using roving stars for on-line testing and diagnosis of fpgas in fault-tolerant applications. In *Test Conference, 1999. Proceedings. International*, pages 973–982, 1999. DOI: 10.1109/TEST.1999.805830.
3. J. Andersson, J. Gaisler, and R. Weigand. Next Generation MultiPurpose Microprocessor. In *DASIA 2010 Data Systems In Aerospace*, volume 682 of *ESA Special Publication*, page 7, August 2010.
4. A. Avizienis, J.-C. Laprie, B. Randell, and C.E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004. DOI: 10.1109/tdsc.2004.2.
5. C. Glenn Begley and Lee M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, March 2012. DOI: 10.1038/483531a.
6. Khalid Belhajjame, Oscar Corcho, Daniel Garijo, Jun Zhao, Paolo Missier, David Newman, RaÃžl Palma, Sean Bechhofer, Esteban GarcÃŋa Cuesta, JosÃľ Manuel GÃşmez-PÃľrez, Stian Soiland-Reyes, Lourdes Verdes-Montenegro, David De Roure, and Carole Goble. Workflow-centric research objects: First class citizens in scholarly discourse. In *Proceedings of Workshop on the Semantic Publishing, (SePublica 2012) 9th Extended Semantic Web Conference*, May 28 2012.
7. Luca Benini and Giovanni De Michelli. *Networks on chips : technology and tools.* The Morgan Kaufmann series in systems on silicon. Elsevier Morgan Kaufmann Publishers, Amsterdam, Boston, Paris, 2006.
8. Johannes Binder, Stephan Strodl, and Andreas Rauber. Process migration framework – virtualising and documenting business processes. In *Proceedings of the 18th IEEE International EDOC Conference Workshops and Demonstrations (EDOCW 2014)*, pages 398–401, Ulm, Germany, September 2014. DOI: 10.1109/edocw.2014.66.
9. Cristiana Bolchini, Matteo Carminati, and Antonio Miele. Self-adaptive fault tolerance in multi-/many-core systems. *Journal of Electronic Testing*, 29(2):159–175, 2013. DOI: 10.1007/s10836-013-5367-y.

10. C. Braun and H. Wunderlich. Algorithm-based fault tolerance for many-core architectures. In *Test Symposium (ETS), 2010 15th IEEE European*, pages 253–253, May 2010. DOI: 10.1109/ETSYM.2010.5512738.

11. Franck Cappello. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *IJHPCA*, 23(3):212–226, 2009. DOI: 10.1177/1094342009106189.

12. Franck Cappello, Al Geist, Bill Gropp, Laxmikant V. Kalé, Bill Kramer, and Marc Snir. Toward exascale resilience. *IJHPCA*, 23(4):374–388, 2009. DOI: 10.1177/1094342009347767.

13. Sayantan Chakravorty, Celso L. Mendes, and Laxmikant V. KalÃl'. Proactive Fault Tolerance in MPI Applications via task Migration. In *In International Conference on High Performance Computing*, 2006. DOI: 10.1007/11945918_47.

14. Zizhong Chen, Graham E. Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra. Fault tolerant high performance computing by a coding approach. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 213–223, New York, NY, USA, 2005. ACM. DOI: 10.1145/1065944.1065973.

15. V. Cherkassky. A measure of graceful degradation in parallel-computer systems. *Reliability, IEEE Transactions on*, 38(1):76–81, Apr 1989. DOI: 10.1109/24.24577.

16. M. Choi, N.J. Park, K.M. George, B. Jin, N. Park, Y.B. Kim, and F. Lombardi. Fault tolerant memory design for hw/sw co-reliability in massively parallel computing systems. In *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 341–348, April 2003. DOI: 10.1109/NCA.2003.1201173.

17. Christian Collberg, Todd Proebsting, Gina Moraila, Akash Shankaran, Zuoming Shi, and Alex M Warren. Measuring reproducibility in computer systems research. Technical report, University of Arizona, 2014.

18. George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

19. C. Darwin. *The Origin of Species*. John Murray, 1859.

20. Francisco Fernandez De Vega. A Fault Tolerant Optimization Algorithm based on Evolutionary Computation. In *Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX'06)*, pages 335–342, Washington, DC, USA, 2006. IEEE Computer Society. DOI: 10.1109/depcos-relcomex.2006.2.

21. Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 2014. DOI: 10.1016/j.future.2014.10.008.

22. E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, September 2002. DOI: 10.1145/568522.568525.

23. D. Ernst, S. Das, Seokwoo Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10–20, Nov 2004. DOI: 10.1109/MM.2004.85.

24. James Taylor eivind Hovig Geir Kjetil Sandve, Anton Nekrutenko. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10), 10 2013. DOI: doi:10.1371/journal.pcbi.1003285.

25. Ian Gent. The recomputation manifesto, April 12 2013.

26. Daniel Lombrana González, Francisco Fernández de Vega, and Henri Casanova. Characterizing fault tolerance in genetic programming. In *Proc. of the 2009 workshop on Bio-inspired algorithms for distributed systems (BADS'09)*, pages 1–10, New York, NY, USA, 2009. ACM. DOI: 10.1145/1555284.1555286.

27. Ed H. B. M. Gronenschild, Petra Habets, Heidi I. L. Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS ONE*, 7(6), 06 2012. DOI: 10.1371/journal.pone.0038234.

28. Philip J. Guo. CDE: Run any Linux application on-demand without installation. In *Proceedings of the 25th international conference on Large Installation System Administration (LISA'11)*, pages 2–2, Berkeley, CA, USA, 2011.

29. Mark Guttenbrunner and Andreas Rauber. A measurement framework for evaluating emulators for digital preservation. *ACM Transactions on Information Systems (TOIS)*, 30(2), 3 2012. DOI: 10.1145/2180868.2180876.

30. J. Ignacio Hidalgo, Juan Lanchares, Francisco Fernández de Vega, and Daniel Lombrana. Is the island model fault tolerant? In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2737–2744, London, United Kingdom, July 7–11 2007. ACM. DOI: 10.1145/1274000.1274085.

31. Kai Hwang, Jack Dongarra, and Geoffrey C. Fox. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

32. A. Ivanov and G. De Micheli. Guest editors' introduction: The network-on-chip paradigm in practice and research. *Design Test of Computers, IEEE*, 22(5):399–403, Sept 2005. DOI: 10.1109/MDT.2005.111.

33. C.M. Jeffery and R.J.O. Figueiredo. Towards byzantine fault tolerance in many-core computing platforms. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 256–259, Dec 2007. DOI: 10.1109/PRDC.2007.40.

34. Fernanda Lima Kastensmidt, Luigi Carro, and Ricardo Reis. *Fault-Tolerance Techniques for SRAM-Based FPGAs (Frontiers in Electronic Testing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

35. J.L.J. Laredo, P. Bouvry, D.L. GonzÃąlez, F. FernÃąndez de Vega, M.G. Arenas, J.J. Merelo, and C.M. Fernandes. Designing robust volunteer-based evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(3):221–244, 2014. DOI: 10.1007/s10710-014-9213-5.

36. Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006. DOI: 10.1002/cpe.994.

37. D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, 88(4):516–543, April 2000. DOI: 10.1109/5.842998.

38. Rudolf Mayer, Tomasz Miksa, and Andreas Rauber. Ontologies for describing the context of scientific experiment processes. In *Proceedings of the 10th International Conference on e-Science*, Guarujá, SP, Brazil, October 20–24 2014. DOI: 10.1109/eScience.2014.47.

39. P. Mazumder. Design of a fault-tolerant dram with new on-chip ecc. In Israel Koren, editor, *Defect and Fault Tolerance in VLSI Systems*, pages 85–92. Springer US, 1989. DOI: 10.1007/978-1-4615-6799-8_8.

40. Tomasz Miksa, Stefan Proell, Rudolf Mayer, Stephan Strodl, Ricardo Vieira, José Barateiro, and Andreas Rauber. Framework for verification of preserved and redeployed processes. In *Proceedings of the 10th International Conference on Preservation of Digital Objects (iPres 2013)*, Lisbon, Portugal, September 2–6 2013.

41. Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Aleksandra Nenadic, Ian Dunlop, Alan Williams, Thomas Oinn, and Carole Goble. Taverna, reloaded. In M. Gertz, T. Hey, and B. Ludaescher, editors, *SSDBM 2010*, Heidelberg, Germany, June 2010. DOI: 10.1007/978-3-642-13818-8_33.

42. Elizabeth Montero and María-Cristina Riff. On-the-fly calibrating strategies for evolutionary algorithms. *Information Sciences*, 181(3):552–566, 2011.

43. Alicia Morales-Reyes, Evangelos F. Stefatos, Ahmet T. Erdogan, and Tughrul Arslan. Towards Fault-Tolerant Systems based on Adaptive Cellular Genetic Algorithms. In *Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems (AHS'08)*, pages 398–405, Noordwijk, The Netherlands, June 22-25 2008. IEEE Computer Society.

44. Nature. Data's shameful neglect. *Nature*, 461(7261), 9 2009. DOI: 10.1038/461145a.

45. Dongkook Park, C. Nicopoulos, Jongman Kim, N. Vijaykrishnan, and C.R. Das. Exploring fault-tolerant network-on-chip architectures. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 93–104, June 2006. DOI: 10.1109/DSN.2006.35.

46. Stefan Pröll and Andreas Rauber. Scalable Data Citation in Dynamic, Large Databases: Model and Reference Implementation. In *IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*, Santa Clara, CA, USA, October 2013. IEEE. DOI: 10.1109/bigdata.2013.6691588.

47. A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P.Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 13–24, June 2014. DOI: 10.1109/ISCA.2014.6853195.

48. Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Comput. Surv.*, 46(1):8:1–8:38, July 2013. DOI: 10.1145/2522968.2522976.

49. Stephen L. Scott, Christian Engelmann, Geoffroy R. Vallée, Thomas Naughton, Anand Tikotekar, George Ostrouchov, Chokchai Leangsuksun, Nichamon Naksinehaboon, Raja Nassar, Mihaela Paun, Frank Mueller, Chao Wang, Arun B. Nagarajan, and Jyothish Varma. A Tunable Holistic Resiliency Approach for High-performance Computing Systems. *SIGPLAN Not.*, 44(4):305–306, February 2009. DOI: 10.1145/1594835.1504227.

50. Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems (3rd Ed.): Design and Evaluation*. A. K. Peters, Ltd., Natick, MA, USA, 1998.

51. C.T. Silva, J. Freire, and S.P. Callahan. Provenance for visualizations: Reproducibility and beyond. *Computing in Science Engineering*, 9(5):82–89, October 2007. DOI: 10.1109/MCSE.2007.106.

52. Marc Snir, Robert W. Wisniewski, Jacob A. Abraham, Sarita V. Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A. Chien, Paul Coteus, Nathan DeBardeleben, Pedro C. Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. Addressing failures in exascale computing. *IJHPCA*, 28(2):129–173, 2014.

53. Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition, 2006.

54. The Economist. Trouble at the lab, October 19 2013.

55. S. Tselonis, V. Dimitsas, and D. Gizopoulos. The functional and performance tolerance of gpus to permanent faults in registers. In *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, pages 236–239, July 2013. DOI: 10.1109/IOLTS.2013.6604089.

56. Jan Vitek and Tomas Kalibera. R3: Repeatability, reproducibility and rigor. *SIGPLAN Not.*, 47(4a):30–36, March 2012. DOI: 10.1145/2442776.2442781.

57. Aida Vosoughi, Kashif Bilal, Samee Ullah Khan, Nasro Min-Allah, Juan Li, Nasir Ghani, Pascal Bouvry, and Sajjad Madani. A multidimensional robust greedy algorithm for resource path finding in large-scale distributed networks. In *Proceedings of the 8th International Conference on Frontiers of Information Technology*, FIT '10, pages 16:1–16:6, New York, NY, USA, 2010. ACM. DOI: http://doi.acm.org/10.1145/1943628.1943644.

58. Chao Wang, F. Mueller, C. Engelmann, and S.L. Scott. Proactive process-level live migration in HPC environments. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12, Nov 2008. DOI: 10.1109/SC.2008.5222634.

59. Keun Soo Yim and R.K. Iyer. A codesigned fault tolerance system for heterogeneous many-core processors. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 2053–2056, May 2011. DOI: 10.1109/IPDPS.2011.375.

Received March 8, 2015.

DOI: 10.14529/jsfi150204

Energy Measurement Tools for Ultrascale Computing: A Survey

Francisco Almeida[1], *Javier Arteaga*[1], *Vicente Blanco*[1], *Alberto Cabrera*[2]

With energy efficiency one of the main challenges on the way towards ultrascale systems, there is a great need for access to high-quality energy consumption data. Such data would enable researchers and designers to pinpoint energy inefficiencies at all levels of the computing stack, from whole nodes down to critical regions of code. However, measurement capabilities are often missing, and significantly differ between platforms where they exist. A standard is yet to be established. To that end, this paper attempts an extensive survey of energy measurement tools currently available at both the hardware and software level, comparing their features with respect to energy monitoring.

Keywords: energy measurement, power measurement, data acquisition tools, infrastructure management, ultrascale computing.

Introduction

Energy sustainability is a significant concern for high-performance computing, with cost of operation due to power draw being one of the main limiting factors for the design of new systems. This need to improve the energy efficiency of computation is compounded by the growth to ultrascale infrastructure.

To enable energy optimization across the whole stack, it is desirable to gain insight into the consumption of existing systems at all levels possible. Ideally, system designers and operators, as well as application developers, should be able to easily access precise power data ranging from whole systems to individual components inside a computation node. It should also be easily attributable to the code being executed, again ranging from entire processes to parts of specific threads.

However, the power monitoring capabilities of current high-performance computing and ultrascale systems are often more limited. In many cases, only aggregate and approximate data are available. A low level of precision and temporal resolution can be sufficient for administration and maintenance purposes, but many interesting applications, such as application energy efficiency analysis or energy-aware dynamic scheduling, require finer-grained measurements.

The issue of acquisition portability also remains open. Existing built-in component power sensors have to be read using as many different interfaces as vendors are involved. Current data center management standards (such as the Intelligent Platform Management Interface (IPMI) [21] or the Data Center Manageability Interface (DCMI) [20]) cannot leverage most sensors, providing instead low-resolution data from supported motherboards.

For these reasons, we believe there is a justification for standardization of energy data acquisition techniques. Such a unification attempt would benefit from an understanding of the current state of the art. To that end, we will survey a wide range of measurement hardware and software tools, providing a classification and a review of all capabilities identified as relevant to the analysis of energy consumption.

The remainder of this work is structured as follows: section 1 defines the scope of our survey, referencing related work. Section 2 reports available power and energy measurement devices, grouping them by physical location on the compute node. Section 3 presents a taxonomy and

[1]Universidad de La Laguna, Santa Cruz de Tenerife, Spain
[2]Instituto Tecnológico y de Energías Renovables, Santa Cruz de Tenerife, Spain

a review of software tools and libraries involved in the measurement process. The final section concludes the paper and outlines future work.

1. Related work

This survey covers hardware and software systems reporting direct energy and power measurements, with particular focus on current high-resolution approaches, where consumption data for each component is updated at a high frequency. As such, energy consumption estimation techniques, analytical modelling tools and power-aware management systems fall outside our scope.

Previous surveys have mapped the landscape of energy measurement tools. A broad picture is presented in [6], which defines a taxonomy encompassing methods based on measurement, estimation and analytical modelling. However, coverage of hardware based methods is not comprehensive. The state of the art has also significantly changed since its publication.

In [33], both energy modeling and energy measurement techniques are covered. Here, the main focus is not on data acquisition, and many existing hardware and software tools are not included.

A more measurement-oriented survey of methods is found in [19]. However, at sub-node granularity, only two custom instrumentation systems are presented. It also does not cover later developments in integrated power sensors.

An in-depth exploration on some hardware methods can be found on [16], providing detailed insights into the quality of a few chosen power meters and sensors. It covers a subset of the devices considered in this survey.

2. Measurement hardware

Energy consumption data is acquired through hardware sensors. There are a number of implementations, which are typically grouped in three categories by physical location (fig. 1):

- Integrated, where specific hardware components (such as a GPU or CPU) already contain measurement circuitry,
- intra-node, instrumentation devices placed inside the node that can perform probing at component or power lane level, and
- external power meters, measuring total load outside the node's power supply.

Each of these approaches presents their own tradeoff between precision of measurement, temporal resolution, cost of deployment and intrusiveness. The interfaces involved in data acquisition are also different between them. A comparison of interfaces and sampling frequencies among some of the current systems is provided in tab. 1.

2.1. Integrated sensors

2.1.1. CPUs

CPUs typically allow access to a number of performance counters, such as instruction or cache miss counts, that are useful in profiling applications. In modern units, it is also possible to query energy consumption estimates directly provided from the processor.

Intel processors from Sandy Bridge onwards implement the Running Average Power Limit (RAPL) interface [24], which provides running counters of total energy consumed per package

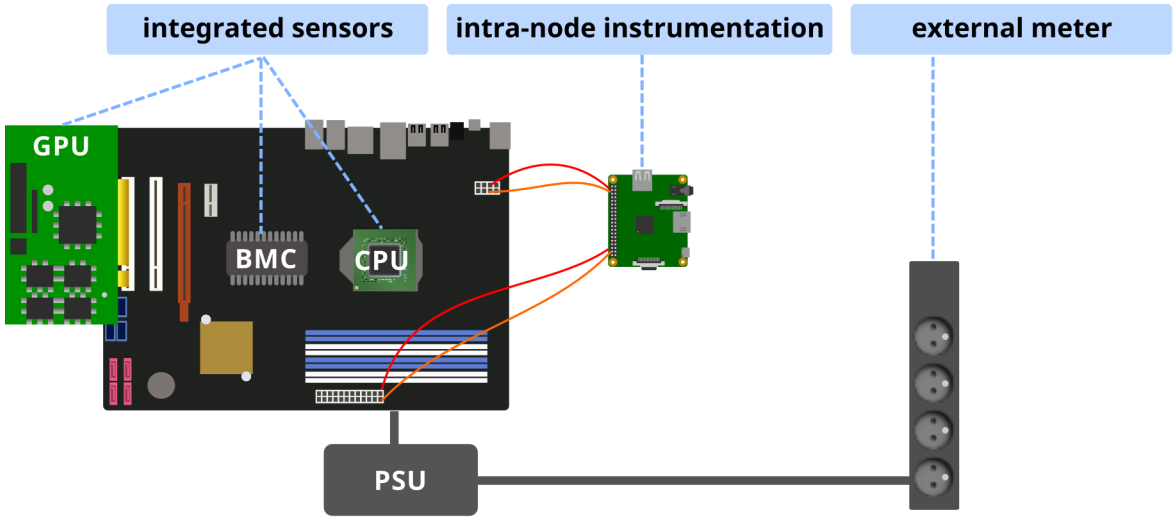


Figure 1. Diagram of hardware measurement tools

(and also, on some models, a total for the DRAM). This interface provides updates roughly every millisecond. However, these counters may overflow: it is up to the interface user to take this fact into consideration. This means constantly polling the registers, in turn increasing the load on a CPU core and adds overhead to the measurements.

AMD processors starting from the Bulldozer (family 15h) microarchitecture also export an estimation of average power over a certain interval through the Application Power Management (APM) [1] capability. Unfortunately, the actual implementation for the Bulldozer family has been shown [16] to provide an inaccurate estimate, particularly during processor sleep states.

2.1.2. GPUs and accelerator cards

Hybrid systems, containing some sort of accelerator units such as GPUs, have been on the rise in recent years [18]. It is thus also desirable to have these components provide energy consumption data from the hardware level.

Many current GPUs provide support for power limiting, which implies the capability to either measure or perform a meaningful estimate of power usage. Nvidia Tesla and Quadro GPUs (from the Fermi GF11x family onwards) additionally make this data available to the user as instant power draw values with nominal accuracy up to ±5% through the Nvidia Management Library [34] C API. Update frequency is not documented, and many potential complications, such as significant sensor lag or sampling interval variability, have been experimentally identified [9].

Another accelerator, the Intel Xeon Phi, exposes a greater number of power sensors: connector inputs, voltage regulator ouputs, and readings for both instant and averaged power draw of the entire card through its System Management Controller chip [23]. Temporal resolution and precision are of 50ms and 1W, respectively. Two methods are provided to query this data [22]: in-band, which involves the Symmetric Comunications Interface and both accelerator and host software support, and out-of-band (without waking up the coprocessor card), via the standard Intelligent Platform Management Bus protocol over the System Management Bus. Multiple software interfaces are provided for both of these methods.

2.1.3. Mainboard

The Advanced Configuration and Power Interface (ACPI) open specification [47] defines power management and device configuration interfaces between an operating system and the BIOS or UEFI. Some power-related information is accessible through ACPI, such as supported processor power states and their expected consumption. As for measuring actual power usage, however, the usefulness of ACPI is limited to rough system-wide estimations in battery-powered systems.

A more advanced form of power usage monitoring can be performed on some motherboards supporting the Intelligent Platform Management Interface (IPMI) [21] (and thus equipped with a Baseboard Management Controller (BMC) monitoring chip). There exist a number of vendor-specific extensions (as implemented by the Dell Remote Access Controller, HP Integrated Lights-Out or Intel Node Manager technologies) which specifically relate to power usage. A more recent standard, the Data Center Manageability Interface (DCMI) [20], builds on top of IPMI 2.0 and introduces power monitoring sensor requirements. However, both standards are devised for administration rather than research purposes, and offer power sampling rates on the order of seconds in the best case.

Other vendors provide entirely proprietary measurement interfaces. This is the case of the IBM Blue Gene/Q, where every node board is fitted with a FPGA which polls voltage and current of all different power domains every 560ms [55]. This information can then be retrieved through IBM's Environmental Monitoring (EMON) API.

2.2. Intra-node instrumentation

For the purposes of dynamic application power profiling, the integrated sensors typically available often do not provide the necessary level of measurement accuracy and subsystem coverage. In these cases, researchers typically employ more sophisticated, and often custom-designed, hardware tooling. For example, the Linux Energy Attribution and Accounting Platform [37] instruments a system's main board to provide power readings using a data acquisition board, which are then exposed to the user via the Linux `/proc` filesystem.

PowerPack [14] was one of the first frameworks aimed towards high-fidelity power-performance profiling. It is comprised of a collection of hardware sensors, meters and data acquisition devices and a software stack providing device drivers and user acquisition interfaces.

The PowerMon line of devices [4], developed by the Renaissance Computing Institute, is inserted between a system's power supply and motherboard, monitoring voltage and current on DC rails to components. Its latest iteration (PowerMon2) can measure up to 8 channels, reaching measurement frequencies up to 1 kSa/s (samples per second) per channel (3 kSa/s aggregate) with low voltage and current measurement error.

A later development in node instrumentation is the PowerInsight [28] from Sandia National Laboratories. Based on the BeagleBone single-board computer plus a custom carrier-board and software layer, PowerInsight allows for instrumentation of up to 15 rails using both special cabling and PCI riser devices fitted with Hall effect sensors. Measurement frequencies are claimed to reach 1 kSa/s counting user-space overhead, down from the 4 kSa/s supported by hardware. Voltage and current accuracies are reported to be higher than those of PowerMon2 [28].

The High Definition Energy Efficiency Monitoring (HDEEM) project [17] implements a similar approach to PowerInsight, claiming a high quality of results due to work on noise filtering

and sensor calibration, potential for high temporal resolution thanks to the speed afforded by the PCIe bus and greater interoperability through integration with the IPMI specification. Hardware cost is also said to be reduced [17], as their hardware implementation builds on top of an already present Baseboard Management Controller chip instead of completely relying on a custom board.

Other devices that may be used for instrumentation include the ARM Energy Probe [3], used with the ARM DS-5 toolchain for energy optimization of software on ARM boards; National Instruments data acquisition equipment [32], which is also available in PCI/PCIe form factors suitable for node instrumentation; and many current measurement integrated circuits based on current-shunt or Hall effect.

2.3. External meters

Measurement from outside the node's power supply is a fairly straightforward method, with little intrusiveness and lower cost than node instrumentation. However, it produces the least useful results: power draw values cannot be attributed to specific components or processes within the node, and the time granularity is typically coarse.

Dedicated power meters available can be inserted between a system and wall outlet. Examples of consumer-grade products are the Kill A Watt [35] and Watt's Up [50] devices, with a time granularity of 1 Sa/s. For the purposes of application power analysis, more high-end devices such as ZES ZIMMER [57] or Yokogawa [54] products can provide far more precise data.

Many power distribution units (PDUs) [10, 40] and power supply units (PSUs) used in data centers also include monitoring capabilities through a variety of interfaces (SNMP, IPMI, Modbus...).

Finally, external custom designs have also been developed for previous work such as PowerScope [13], which used a digital multimeter with a trigger input connected to profiling software, or the Energy Endoscope [43], where an application-specific integrated circuit (ASIC) dedicated to real-time energy monitoring was built.

2.4. Assessment

It is always desirable that power sensors be integrated in hardware components. Some current CPUs and GPUs include reasonably useful measurement capabilities, but there is a room for improvement in accuracy and latency. Other important targets, such as ARM-based processors, as well as subsystems such as disks, memory or network cards, usually offer none of these features. In all cases, energy data is only available at the component level, with no reliable mechanism to attribute energy consumption to a particular software task within a multicore system.

Hardware vendors should aim to provide accurate energy and power data with as fine spatial and temporal granularity as possible. This would enable efficiency gains backed by precise accounting of the power consumption of any specific subsystem down to the process level. Implementation of out-of-band channels would also help minimize the intrusiveness of readings.

In the long term, built-in sensors should phase out any custom-tailored intranode instrumentation on datacenters and HPC deployments. A hardware interface standard for these sensors is also necessary, covering both in-band and out-of-band collection of high-resolution performance data from all components and all external meters present in a computing system. Design insight can be drawn from both vendor-specific and custom implementations seen in this section.

Table 1. Comparison of hardware power measurement systems

Hardware	Vendor/group	Acquisition interfaces	Temporal resolution
Integrated sensors			
Intel CPUs (Sandy Bridge+)	Intel	RAPL	1 kSa/s [17]
AMD CPUs (Bulldozer+)	AMD	APM	100 Sa/s [17]
Tesla, Quadro GPUs	Nvidia	NVML	>60 Sa/s [9, 25]
Xeon Phi	Intel	Custom, IPMI+SMBus	20 Sa/s [23]
Node instrumentation			
PowerPack	Virginia Tech	NI DAQs, Watt's Up [14]	Unknown
PowerMon2	RENCI	Serial port	1 kSa/s (per channel) [5]
PowerInsight	Sandia National Labs, Penguin Computing	SPI bus	>1 kSa/s [28]
HDEEM	TU Dresden, Bull	PCIe and IPMI	1 kSa/s planned [17]
External meters			
Watt's Up Pro	Watt's Up Meters	USB, Ethernet (.Net)	1 Sa/s [49]
Schleifenbauer PDU	Schleifenbauer	Network-based (SNMP, custom...)	1 Sa/s [39]
ZES LMG450	ZES ZIMMER	Serial, Parallel	20 Sa/s [56]

3. Software tools

Many different software tools can be used for power and energy analysis, from low-level software interfaces to full-fledged analysis frameworks, including profilers, tracing and visualization systems, and estimation and modeling tools.

In this section, we will cover those related to the acquisition of power data from hardware sensors. In particular, a comparison of software acquisition interfaces by supported hardware is provided in tab. 2.

3.1. Power-aware low-level profiling interfaces

The Performance API (PAPI) [8], from the University of Tennessee, is well-established as a library interface for hardware performance counters. In recent years, it has included support for many energy measurement sources, such as RAPL, NVML, Xeon Phi or IBM EMON [29, 51, 52]. This allows for easy extraction of power data for projects and researchers which are already users of PAPI. However, PAPI remains focused on in-band measurement, and data from external meters must be collected with some other tool.

Some profiling frameworks focused on CPU performance counters support CPU-based energy metrics. For example, the likwid [46] profiler, as well as the `perf_events` [26] Linux kernel subsystem (a tool designed for profiling CPU performance counters) both directly implement measurement through the RAPL interface.

3.2. Power-aware profiling frameworks

Larger profiling frameworks, with tracing and visualization systems, are usually built on top of PAPI and other libraries, using them as data providers for larger analysis and visualization frameworks. Because of this, many of them are capable of working with in-band power consumption measurement to some extent. These include Paraver [36], Vampir [31], HPCView [30], the Tuning and Analysis Utilities (TAU) [42], Open—Speedshop [41]. Scalasca [15] or Periscope [7].

Some attempts exist to offer interoperability between these systems. In particular, the Score-P [27] measurement infrastructure is compatible with Vampir [31] (replacing the older Vampir-Trace open source library), Scalasca, Periscope and TAU tools. Additional support for energy metrics is currently being worked on within the follow-up Score-E [48] project.

Other profiling solutions also rely on specific hardware systems. The Multiple Metrics Modeling Infrastructure (MuMMI) [53] is one such case, building upon PAPI, PowerPack and the Prophesy [45] performance modeling and prediction framework.

3.3. Power-specific software interfaces

PowerAPI [38] is a recent attempt from Sandia National Laboratories to standardize access to power measurement data and power control at all levels of a given HPC facility, down to hardware components. It is comprised of a specification defining a model of a computation system, user roles and the reference API. A prototype implementation is provided that already implements native support for some energy data sources including RAPL, Cray products supporting power management, the WattsUp meter and PowerInsight.

The Energy Measurement Library (EML) [12] is an open source C library developed at Universidad de La Laguna providing a simple interface for acquisition of hardware energy consumption data through code instrumentation. The API is designed around the concept of inserting asynchronous instrumentation calls around relevant sections of code. Within these sections, the library polls the underlying interfaces, gathering energy data and managing all needed threads and memory [11]. Hardware support is provided for RAPL, NVML, Xeon Phi and Schleifenbauer PDUs.

pmlib [2] is a software package developed at Universitat Jaume I to research the power-performance of parallel scientific code. The framework uses a client/server model for out-of-band power tracing of a target node instrumented with power meter devices. Supported meters include APC PDU units, Watt's Up Pro devices, a data acquisition system from National Instruments, and custom transducer-based designs.

SchedMon [44], from the Signal Processing Group at INESC-ID, reports hardware power consumption by coupling a Linux kernel module for access to hardware counters, a library providing the measurements to userspace, and a reference commandline user interface. It currently obtains power data from the RAPL interface.

Table 2. Comparison of low-level software power measurement interfaces

Name	Author	License	Power measurement support
Power-aware profilers and interfaces			
PAPI	University of Tennessee	BSD	Intel RAPL, Intel Xeon Phi, Nvidia NVML, IBM EMON
likwid	Jan Treibig et al. [46]	GPLv3	Intel RAPL
`perf_events`	Community	GPLv2	Intel RAPL
Power-specific software interfaces			
PowerAPI (prototype)	Sandia National Laboratories	BSD	Intel RAPL, Cray XTPM, PowerInsight, Watt's Up
EML	Universidad de La Laguna	GPLv2	Intel RAPL, Nvidia NVML, Intel Xeon Phi, Schleifenbauer PDU
pmlib	Universitat Jaume I	Unknown	Intel RAPL, APC PDU, Watt's Up, National Instrument DAQs, custom
SchedMon	INESC-ID	MIT	Intel RAPL

3.4. Assessment

Both HPC tools and research code benefit from a set of standard capabilities for energy data acquisition. Thus, they should be built upon low-level software abstraction layers which give a simple interface to performance data sources, including energy. In the absence of hardware standards to leverage, it would fall upon this layer to encapsulate the complexity of dealing with different sensor types. Currently, PAPI is in a good position to also fill this role for energy and power, although it is hindered by a lack of out-of-band measurement support.

Further abstraction layers should define standard data trace formats and utilities for easy interoperability between tools. This format must take into account the scalability concerns involved in dealing with high-resolution data in large-scale deployment. In the HPC analysis tooling front, some important standardization work is already underway in the form of projects such as Score-P. These standards should continue to mature and be adopted by more HPC frameworks.

Lastly, many high-level tools could be extended through these libraries to consider available energy data. Datacenter operators and researchers alike should have analysis and visualization tools with the ability to "zoom in" on the energy consumption incurred for a given program, user, or even piece of code, both offline and online, looking at any level of hardware granularity: from a whole system, to a specific core within one processor. Many other practical applications where this data could drive energy efficiency gains can be imagined, such as energy-aware scheduling and load balancing, or compiler-assisted energy optimization of program code.

Conclusions

This survey has attempted to outline the current state of direct energy and power measurement techniques. In doing so, we aim to help future researchers choose an appropriate solution

for their analysis needs, as well as help guide future developments in hardware and software tools.

As we have shown, there is a great amount of diversity in energy and power measurement approaches, with many coexisting implementations and interfaces. Most of the current power measurement solutions were originally designed for administration and management purposes, typically measuring at the whole node level, and with a temporal resolution of 1 Sa/s. This accuracy can be unsuitable for fine-grained application consumption analysis. However, recent designs from both HPC vendors and research groups have achieved significantly higher temporal resolution (up to the order of 1 kSa/s) and better spatial granularity (with separate per-component measurement channels).

A number of challenges still stand in the way of ubiquitous high-fidelity consumption monitoring. The community depends on hardware vendors to embed accurate, fine-grained sensors into hardware components, relieving the need to design and install custom-tailored instrumentation hardware into their systems.

Additionally, to ensure the portability of instrumentation software, a greater degree of standardization of capabilities and interfaces is desirable. This would entail cooperation between vendors to ensure hardware interoperability of energy data acquisition techniques, much like what has already been done in the data center administration front with the IPMI and DCMI standards.

Another important consideration for future developments is scalability of both the instrumentation and acquisition processes. Instrumentation scalability would benefit from integration of reliable power measurement sensors exposing standard interfaces within hardware components: custom instrumentation systems can incur significant development and installation difficulty. Acquisition scalability will depend on the development of advanced collection and data processing systems, given that handling of fine-grained measurements at large scale will likely prove not to be an easy task in itself.

This work was supported by the Spanish Ministry of Education and Science through the TIN2011-24598 project, Spanish CAPAP-H4 network, and NESUS IC1315 COST Action.

References

1. Advanced Micro Devices. AMD BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors, January 2013.
2. Pedro Alonso, Rosa M Badia, Jesus Labarta, Maria Barreda, Manuel F Dolz, Rafael Mayo, Enrique S Quintana-Orti, and Ruymán Reyes. Tools for power-energy modelling and analysis of parallel scientific applications. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 420–429. IEEE, 2012. DOI: 10.1109/icpp.2012.57.
3. ARM Limited. ARM Energy Probe. `http://ds.arm.com/ds-5/optimize/arm-energy-probe/`.
4. D. Bedard, Min Yeol Lim, R. Fowler, and A. Porterfield. Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In *IEEE South-*

eastCon 2010 (SoutheastCon), Proceedings of the, pages 479–484, March 2010. DOI: 10.1109/secon.2010.5453824.

5. Daniel Bedard, R Fowler, M Linn, and Allan Porterfield. Powermon 2: Fine-grained, integrated power measurement. *Renaissance Computing Institute, Tech. Rep. TR-09-04*, 2009.
6. Shajulin Benedict. Energy-aware performance analysis methodologies for HPC architectures—an exploratory study. *Journal of Network and Computer Applications*, 35(6):1709 – 1719, 2012.
7. Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. Periscope: An online-based distributed performance analysis tool. In *Tools for High Performance Computing 2009*, pages 1–16. Springer, 2010. DOI: 10.1007/978-3-642-11261-4_1.
8. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, August 2000.
9. Martin Burtscher, Ivan Zecena, and Ziliang Zong. Measuring GPU power with the K20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, GPGPU-7, pages 28:28–28:36, New York, NY, USA, 2014. ACM. DOI: 10.1145/2576779.2576783.
10. APC by Schneider Elecric. Metered-by-outlet rack pdus, March 2013.
11. Alberto Cabrera, Francisco Almeida, Javier Arteaga, and Vicente Blanco. Measuring energy consumption using EML (Energy Measurement Library). *Computer Science-Research and Development*, pages 1–9, 2014.
12. Alberto Cabrera, Francisco Almeida, and Vicente Blanco. Eml, an energy measurement library. *31st International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2013: Student Poster Abstracts*, 2013.
13. Jason Flinn and Mahadev Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA*, pages 2–10. IEEE Computer Society, 1999. DOI: 10.1109/mcsa.1999.749272.
14. Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, May 2010. DOI: 10.1109/TPDS.2009.76.
15. Markus Geimer, Felix Wolf, Brian JN Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010. DOI: 10.1002/cpe.1556.
16. D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W.E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 194–204, April 2013. DOI: 10.1109/ISPASS.2013.6557170.
17. Daniel Hackenberg, Thomas Ilsche, Joseph Schuchart, Robert chöne, Wolfgang E. Nagel, Marc Simon, and iannis Georgiou. Hdeem: High definition energy efficiency monitoring. In *Proceedings of the 2Nd International Workshop on Energy Efficient upercomputing*, E2SC '14, pages 1–10, Piscataway, NJ, USA, 2014. IEEE Press.

18. Nicole Hemsoth. Are Supercomputing's Elite Turning Backs on Accelerators? http://www.hpcwire.com/2014/06/26/accelerators-hold/, June 2014.

19. Chung-Hsing Hsu and S.W. Poole. Power measurement for high performance computing: State of the art. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–6, July 2011. DOI: 10.1109/IGCC.2011.6008596.

20. Intel Corporation. Data Center Manageability Interface Specification, August 2011.

21. Intel Corporation. Intelligent Platform Management Interface Spec, October 2013.

22. Intel Corporation. Determining the Idle Power of an Intel® Xeon Phi™ Coprocessor. https://software.intel.com/en-us/articles/determining-the-idle-power-of-an-intel-xeon-phi-coprocessor, June 2014.

23. Intel Corporation. Intel® Xeon Phi™ Coprocessor Datasheet. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-datasheet.html, April 2014.

24. Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Number 253669-053US. January 2015.

25. Kiran Kasichayanula, Dan Terpstra, Piotr Luszczek, Stan Tomov, Shirley Moore, and Gregory D Peterson. Power aware computing on gpus. In *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*, pages 64–73. IEEE, 2012. DOI: 10.1109/saahpc.2012.26.

26. kernel.org. Perf Wiki. Rlhttps://perf.wiki.kernel.org/index.php?title=Main_Page&oldid=3491, 2014.

27. Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Tools for High Performance Computing 2011*, pages 79–91. Springer, 2012. DOI: 10.1007/978-3-642-31476-6_7.

28. James H Laros, Phil Pokorny, and David DeBonis. Powerinsight-a commodity power measurement capability. In *Green Computing Conference (IGCC), 2013 International*, pages 1–6. IEEE, 2013. DOI: 10.1109/igcc.2013.6604485.

29. Heike McCraw, James Ralph, Anthony Danalis, and Jack Dongarra. Power monitoring with papi for extreme scale architectures and dataflow-based programming models. 2014. DOI: 10.1109/cluster.2014.6968672.

30. John Mellor-Crummey, Robert J Fowler, Gabriel Marin, and Nathan Tallent. Hpcview: A tool for top-down analysis of node performance. *The Journal of Supercomputing*, 23(1):81–104, 2002. DOI: 10.1023/a:1015789220266.

31. Wolfgang E Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. Vampir: Visualization and analysis of mpi resources. 1996.

32. National Instruments Corporation. What Is Data Acquisition? http://www.ni.com/data-acquisition/what-is/.

33. Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review*, 47(3):42–49, 2013. DOI: 10.1145/2553070.2553077.

34. NVIDIA Corporation. NVML API Reference Guide, March 2014.

35. P3 International. Kill A Watt product page. http://www.p3international.com/products/p4400.html.

36. Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31. mar, 1995.

37. Sebastian Ryffel. Lea^{2}p: The linux energy attribution and accounting platform. Master's thesis, Swiss Federal Institute of Technology, 2009.

38. Sandia National Laboratories. High performance computing power application programming interface (api) specification. http://powerapi.sandia.gov/, 2014.

39. Schleifenbauer. Schleifenbauer operation manual. http://schleifenbauer.eu/dynamisch/bibliotheek/16_0_NL_usermanual_v2.1.pdf.

40. Alain Schuermans. Schleifenbauer products bv, March 2012.

41. Martin Schulz, Jim Galarowicz, Don Maghrak, William Hachfeld, David Montoya, and Scott Cranford. Open— speedshop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008. DOI: 10.1155/2008/713705.

42. Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006. DOI: 10.1177/1094342006064482.

43. Thanos Stathopoulos, Dustin McIntire, and William J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *IPSN*, pages 383–394. IEEE Computer Society, 2008. DOI: 10.1109/ipsn.2008.36.

44. Luís Taniça, Aleksandar Ilic, Pedro Tomás, and Leonel Sousa. Schedmon: A performance and energy monitoring tool for modern multi-cores. In *Euro-Par 2014: Parallel Processing Workshops*, pages 230–241. Springer, 2014. DOI: 10.1007/978-3-319-14313-2_20.

45. Valerie E Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling n, Rick Stevens, Mark Hereld, and Ivan R Judson. Prophesy: An infrastructure for analyzing and modeling the performance parallel and distributed applications. In *High-Performance Distributed Computing, 2000. Proceedings. The nth International Symposium on*, pages 302–303. IEEE, 2000. DOI: 10.1109/hpdc.2000.868668.

46. J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.

47. Unified EFI, Inc. Advanced Configuration and Power Interface Specification, July 2014.

48. Virtual Institute High Productivity Supercomputing. Score-E. http://www.vi-hps.org/projects/score-e/, 2013.

49. Watt's Up Meters. Operators Manual. https://www.wattsupmeters.com/secure/downloads/manual_rev_9_corded0812.pdf.

50. Watt's Up Meters. Watt's Up product page. https://www.wattsupmeters.com/.

51. V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with papi. In *International Workshop on Power-Aware Systems and Architectures*, Pittsburgh, PA, September 2012. DOI: 10.1109/icppw.2012.39.

52. Vincent M Weaver, Daniel Terpstra, Heike McCraw, Matt Johnson, Kiran Kasichayanula, James Ralph, John Nelson, Philip Mucci, Tushar Mohan, and Shirley Moore. Papi 5: Measuring power, energy, and the cloud. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 124–125. IEEE, 2013. DOI: 10.1109/ispass.2013.6557155.

53. Xingfu Wu, Charles Lively, Valerie Taylor, Hung-Ching Chang, Chun-Yi Su, Kirk Cameron, Shirley Moore, Daniel Terpstra, and Vince Weaver. Mummi: multiple metrics modeling infrastructure. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on*, pages 289–295. IEEE, 2013. DOI: 10.1109/snpd.2013.73.

54. Yokogawa Electric Corporation. Power Analyzers. `http://tmi.yokogawa.com/products/digital-power-analyzers/`.

55. K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, Chenjie Yu, and S. Coghlan. Evaluating power-monitoring capabilities on ibm blue gene/p and blue gene/q. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 36–44, September 2012. DOI: 10.1109/CLUSTER.2012.62.

56. ZES ZIMMER. Brochure LMG450. `http://www.zes.com/en/content/download/286/2473/file/lmg450_prospekt_1002_e.pdf`.

57. ZES ZIMMER Electronic Systems GmbH. Products: Precision Power Analyzer. `http://www.zes.com/en/Products/Precision-Power-Analyzer`.

Received March 1, 2015.

DOI: 10.14529/jsfi150205

Energy-efficient Algorithms for Ultrascale Systems

Jesus Carretero[1]***, Salvatore Distefano***[2]***, Dana Petcu***[3]***, Daniel Pop***[3]***,***
Thomas Rauber[4]***, Gudula Rünger***[5]***, David E. Singh***[1]

The chances to reach Exascale or Ultrascale Computing are strongly connected with the problem of the energy consumption for processing applications. For physical and economical reasons, the energy consumption has to be reduced significantly to make Ultrascale Computing possible. The research efforts towards energy-saving mechanisms of the hardware have already made energy-aware hardware systems available. However, to achieve a strong energy reduction, hardware mechanisms must be complemented with new energy-efficient software that can exploit them so that the foreseen energy savings actually result. In the software area, there also exist a multitude of research approaches towards energy saving, often concentrating either on the system software level or the application organization level, reflecting the expertise of the corresponding research group. The challenge of reducing the energy consumption dramatically to make Ultrascale Computing possible is so ambitious that a concerted action combining research efforts through all the software levels seems reasonable. In this article, we discuss the current research efforts and results related to energy efficiency in the diverse areas of software. We conclude with open problems and questions concerning energy-related techniques with an emphasis on the application or algorithmic side.

Keywords: energy-awareness, energy-efficient algorithms, ultrascale computing.

Introduction

The performance of high-end HPC systems has been increased roughly by a factor of 1000 in each of the last two decades. With the world's most powerful systems already well past the Petaflop/s level in 2014, a projection of this trend leads to the prediction that by 2022, Exascale computing will be possible. However, progress towards this goal is threatened by energy issues because, based on the current technology, systems with Exascale performance would use excessive amounts of energy (e.g. Tianhe-2, a 33 PFlops system, needs about 18 MW). Moreover, due to physical constraints, the performance of processing elements can no longer be assumed to follow Moore's Law. Accordingly, because of physical constraints and environmental issues, power and energy consumption are considered to be one of the largest challenges for Exascale systems. The US DOE Exascale Initiative has set a target of 20 MW for the power consumption of an Exascale system. To achieve 1 ExaFLOP using 20 MW, the average energy cost per flop must be limited to 20 picojoules (20 pJs/FLOP), including all costs for memory accesses and communication [108]. However, the supercomputers on the current Top500 list need between 300 and 8000 pJs/flop[6].

Consequently, reducing the energy consumption for computing has become an increasingly important research topic in recent years, with the research community following two main research directions: The first direction is concerned with power-aware and thermal-aware hardware design, including low-power techniques on all levels, i.e. the circuit and logic level, the processor, the memory and the interconnects. The second research direction is based on the development

[1]University Carlos III of Madrid, Madrid, Spain
[2]Politecnico di Milano, Milano, Italy
[3]West University of Timisoara, Timisoara, Romania
[4]University Bayreut, Bayreut, Germany
[5]Technical University Chemnitz, Chemnitz, Germany
[6]`www.top500.org`

of power-aware software for the entire software stack, including operating systems, compilers, applications and algorithms. This second direction is the topic of this survey article, in which we summarize important contributions towards energy reduction that can be provided by the system software or the programming model and discuss how these contributions can be used for the construction of energy-efficient algorithms and applications. An important step towards a systematic development of energy-efficient algorithms is the energy-oriented investigation of benchmark programs. As an example, the energy characteristics of benchmark programs such as SPEC CPU and PARSEC are investigated and algorithmic techniques for energy saving are considered. The emphasis of our investigation is on large-scale complex computing systems, which will be referred to as Ultrascale or Exascale systems in the following.

The rest of the article is structured as follows: Section 1 gives a brief overview of the hardware mechanisms that can be used to reduce energy consumption. Section 2 deals with system support for energy efficiency and presents some energy metrics as well as novel energy measurement and power management techniques. Section 3 studies how the programming model and the software development process can support the construction of energy-efficient algorithms and applications. Section 4 considers the energy consumption of algorithms and discusses algorithmic techniques to enhance energy awareness at the programming level. The final section concludes the article with a discussion of important research directions that are crucial for reaching energy efficiency in algorithms.

1. Hardware mechanisms for energy saving

Nowadays, computers include different power management techniques which support the reduction of energy consumption. Examples are dynamic voltage frequency scaling (DVFS), clock gating, and power gating. Moreover, the usage of special instructions and specialized coprocessors can also help to reduce energy consumption.

DVFS [4] can reduce the clock frequency and voltage level of different components of the compute node (processors, DRAM memories, etc.) at the expense of some performance degradation. Currently, DVFS is broadly supported by low-power and high performance processors provided by different manufacturers under different names (e.g. *SpeedStep* in Intel processors and *PowerNow* or *Cool 'n' Quiet* in AMD processors). There are three factors that need to be considered when DVFS is applied: (a) the dynamic power, which has a quadratic relationship with frequency-voltage scaling; (b) the static power, which increases exponentially with the voltage; and (c) the performance, which has a linear relationship with the frequency. Because of its negative performance impact, DVFS may only be effective for non CPU-bounded applications, see Section 4.1 for more details.

Clock Gating [97] reduces the power consumption by disabling the clock in those parts of the circuit that are idle or, like in the case of flip-flops, maintain a steady state that does not need to be refreshed. The power used to drive the clock signal can represent more than a half of the overall power consumption. Therefore, clock gating can potentially achieve a significant energy reduction. This technique can be controlled both at hardware and software level. Hardware-level approaches typically provide a finer granularity, allowing also to disable components inside a functional block. Software-level approaches are usually applied at entire functional blocks, but they allow more elaborated energy-saving policies.

Power gating [96] is a more aggressive approach in which a functional block is disconnected from the power supply, powering off all its components. Nowadays, existing processors contain

clock gating logic managed by a power reduction policy for almost every functional block. For some components clock gating is used in combination with power gating features. Given that the entire functional unit is disconnected, power gating achieves a better power reduction than clock gating. However, given that the functional unit state is erased, it is necessary to provide mechanisms for saving and restoring the states of the functional units, which increases the complexity and complicates resource utilization when applying power gating to active components that need to preserve their state.

The use of special instructions can also help to reduce the energy consumption for compute-intensive applications. Examples are the SIMD vector instructions provided by the AVX (advanced vector extensions) instructions for the x86 architecture or the AES (advanced encryption standard) instructions to support encryption and decryption. Those instructions lead to an effective use of the corresponding transistors, thus reducing the energy consumption per operation [71].

Similarly, the use of specialized coprocessors or accelerators, such as GPU (Graphics Processing Unit), MIC (Many Integrated Cores) or FPGA (Field Programmable Gate Array), can also lead to a smaller energy consumption compared to general purpose CPUs. As an example, the NVIDIA "Fermi" generation of GPUs requires about 200 picojoules of energy to execute one instruction, which is 10x less than for the most efficient x86 CPU.

2. System support for energy efficiency

In order to obtain the benefits offered by an Ultrascale or Exascale system, it will be increasingly important to provide system services for an effective management of the system resources on behalf of the applications. Those services can be offered to the applications through the programming environment or through specialized libraries, but they should be as transparent to the user as possible to support application porting and sustainability. As energy is a cross-layer issue, several aspects of the system software and the operating system should be involved in energy efficiency resource management, but it is also paramount to provide metrics and facilities to monitor and express energy at the processor and system level.

2.1. Resource management

Currently, power requirements are driving the co-design of HPC systems, which in turn sets the course for a radical change in how to express the need for increasingly scarce resources, as well as how to manage them. Knowing that Ultrascale and Exascale systems will inevitably rely on a high-level heterogeneity of resources and new HPC usage challenges (such as providing performance hand-in-hand with energy efficiency), they need to become more and more self-aware with respect to performance, energy and resilience [36]. New usages, like many-task computing paradigms, will force the system to host, schedule, and load balance millions of heterogeneous tasks. Existing research provides analytical studies quantifying and comparing expected performance of new solutions proposed.

Another approach is to use layered solutions, such as the use of algorithm-specific checkpointing combined with system-level checkpointing [19], or to use imperfect fault predictors [10]. Following this trend, decentralized approaches for a multi-objective, energy-aware resource management will be a likely replacement for centralized approaches when these do not scale up. Gossip-based [65] and hierarchical approaches [124] are examples that have been proposed for

load balancing. However, the scale to which they have been evaluated and the complexity of their balancing requirements is far from what is expected for Exascale.

2.2. Energy metrics

In order to properly evaluate a specific system property, it is necessary to define corresponding metrics. With regard to energy, the main basic metric is usually the unit of work or amount of heat transferred, measured in *Joule* (J), while the power, i.e. the amount of transferred energy in time, is measured in $Watt$ (W).

In the computing system context, several initiatives related to energy measurement and management have been started, mostly grouped under the umbrella of Green IT. Some of them focus on distributed systems, aiming at identifying specific metrics for assessing energy efficiency in these systems. A good example is GreenGrid, which is "an association of IT professionals seeking to dramatically raise the energy efficiency of datacenters through a series of short-term and long-term proposals" [104]. They propose to use two main metrics for evaluating energy efficiency in datacenters: Power Usage Effectiveness (PUE), and Datacenter Infrastructure Efficiency (DCiE) [11, 16]. PUE is defined as follows:

$$PUE = \frac{TotalFacilityEnergy}{ITEquipmentEnergy}$$

while $DCiE$ is specified as its reciprocal:

$$DCiE = \frac{1}{PUE} = \frac{ITEquipmentEnergy}{TotalFacilityEnergy} \times 100\% \quad .$$

The energy for the total facility is the overall amount of energy consumed by the whole data center, including IT systems and facilities. The IT systems energy is the energy consumed by just the IT equipment such as processing, storage, and network components for data management and processing. The facilities include all other subsystems, such as UPS and power management systems, cooling systems, lighting systems, etc.

Other interesting initiatives in the direction towards widely used metrics and, possibly, standards, are Energy Star [110] and SPECpower [64]. Energy Star specifies specific rules, provides a rating for energy efficiency, called the Energy Star score, and is based on SPECpower. SPECpower is mainly a benchmark for evaluating the energy efficiency of server-class compute equipments. Several Performance-per-Power metrics have been proposed which report the ratio between a given performance metric (such as response time, throughput, utilization, delay, bandwidth, etc.) and the energy consumed for obtaining such a performance. An example is the metric transactions per second per Watt (TPS/Watt), using the throughput as performance metrics.

For the particular characteristics of Exascale platforms, specific energy efficiency metrics are not yet specified and a metric that is able to take performance, scalability, as well as energy efficiency into account still needs to be introduced.

2.3. Energy measurement techniques

A major challenge for energy measurement and monitoring is their use on heterogeneous platforms through a standard access monitoring interface. Standardized monitoring interfaces for energy and resource utilization are necessary to support local and global control decisions and

should be able to handle the diversity of hardware devices, such as GPUs, embedded CPUs, and nonvolatile low-power memory and storage. An example for a standardized access to performance counters is the PAPI interface, which currently can be used on a large number of platforms including the Intel Core i7 architecture, NVIDIA GPUs, the Intel Xeon Phi and IBM Blue Gene/Q systems [78].

For CPU power monitoring, one approach consists in finding the relationship between the power consumption and the utilization level. The utilization level is computed from different workloads that stress different components of the system (CPU, memory, I/O, etc.). In the literature [31, 81] it has been shown that the power consumption and the utilization level are related linearly, regardless of the type of workload and the configuration of the processor, e.g. in terms of operational frequency or the number of active cores.

As an alternative, the CPU performance can be indirectly modeled by means of hardware counters that capture different hardware events, such as the number of cache accesses or the number of instructions issued [98]. Performance monitoring counters do not require program modifications or an intrusion into the hardware structure and they can accurately reflect the activity levels of the processor or the memory subsystem. An example of this modeling technique is given in [66], where the event-based power prediction is enhanced by using the correlation of the power consumption with the change in core die temperature and the ambient temperature. Recent Intel CPU architectures include the Running Average Power Limit (RAPL) energy sensors to measure the power consumption of different components, including the CPU and the memory controller. The use of these counters is an efficient and low overhead alternative to measure the power of a system using specialized power meters [45]. Energy modeling approaches and a comparison with measured energy values are discussed in [88].

2.4. Power management techniques

The Advanced Configuration and Power Interface (ACPI) [26] is an open standard for device power management co-developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. It specifies different global and device energy states, which range from fully operational to completely powered off, and provides an interface to manage and monitor the power of the infrastructure components. ACPI can be accessed by the user with the aid of user-defined policies, such as specifying an application power level, or by the operating system, which applies power policies based on the platform load, such as switching the components to a low power state after a time of inactivity.

There are also advanced tools that provide support for a real-time power management of the infrastructure components, including servers, storage, network, and cooling equipment. Examples are the Intel Datacenter Manager [28], the IBM Systems Director Active Energy Manager [27], and the HP Power Advisor [50]. They provide a single cross-platform view, can be used at multiple hierarchy levels, and support different energy policies, such as power capping, power saving and generation, and the analysis of power history data logs. In addition, most of these tools are fully integrated in the infrastructure management software, allowing it to perform energy-aware tasks, such as workload scheduling.

Several approaches address the improvement of the system energy efficiency. An example is given in [44], where DVFS is used to control the CPU power based on different policies which are applied considering the number of executed instructions, the memory traffic, and the consumer power of the processor. Memscale [33] applies dynamic frequency scaling to the complete out-

of-chip memory subsystem (memory controller, memory channel, and DRAM device), as well as dynamic voltage scaling to the memory controller. It includes a control algorithm that minimizes the overall system energy based on performance counter monitoring. This work was extended [32] to multiple memory devices and controllers. [62] presents an energy model for the execution of a parallel conjugate gradient method split between the CPU and the GPU. The approach considers the CPU, GPU, and RAM energy consumption and uses the information to perform an energy-aware workload distribution minimizing the execution time. A more global approach is followed in [24], where a runtime optimization technique is presented for improving energy efficiency in processors, disks, and networks.

The effectiveness of DVFS is restricted by the range of the minimum and the maximum voltages at which the transistors can operate. Moreover, DVFS is difficult to apply when workloads of different characteristics are executed. To overcome these problems, the idea of complementing DVFS with power gating has been proposed. [75] introduces PGCapping, a system that integrates power gating with DVFS for chip multiprocessors. [1] presents a gating-aware scheduler and a power gating scheme for GPGPU execution units that achieve significant energy saving in simulations.

When considering large computing infrastructures, the power proportionality arises, besides the energy efficiency, as a crucial concept. Power-proportionality means that the system's energy usage is proportional to its workload. In this way, the machine would consume no power in the idle state and would gradually increase the power consumption as the workload increases. An Exascale architecture should be both energy efficient and power proportional. However, existing systems are far from fulfilling this requirement. Consequently, it is necessary to develop new hardware and software tools that help to achieve it [38]. Examples for such tools are described in [105] and [6]. The first one shows a power-proportional distributed storage system for data centers that powers down servers according to the load level and considering the performance degradation, availability and data consistency. The second one presents a distributed filesystem based on the Hadoop DFS. It provides power proportionality minimizing the number of active nodes, including power-proportional capabilities for failures such as minimizing the number of nodes that need to be restored when there is a failure of the filesystem. [47] describes a solution to provide energy proportionality for networks by dynamically adapting the energy consumption of a network through traffic patterns analysis and by finding minimum power network subsets. A survey of techniques that aim to improve the energy efficiency of computing and network resources is given in [80], covering techniques that operate both on parallel and distributed system levels.

2.5. Monitoring and Benchmarking

With specific regard to Exascale platforms, there are three main challenges for energy efficiency metrics and monitoring: (1) scalability, (2) standard access monitoring methods, and (3) its application on heterogeneous platforms [53]. Monitoring everything produces extremely large trace files making their analysis prohibitive. Alternatives are statistical models [83], time series approaches [67], and data filtering with a distributed analysis that produces small trace files with a small runtime overhead [60, 84].

At node level, it is crucial to find the relationship between the power consumption and the utilization level computed, which seems to be linear [31, 81]. As discussed above, one possibility is to use hardware counters to model the CPU performance [98] and Intel RAPL to measure the

CPU and memory controller power consumption [45, 66]. At the whole compute infrastructure level, power proportionality arises as a crucial concept [39, 70]. Even if the current hardware components are not power-proportional, we can see in the literature examples of system wide [47, 105] and system specific models to achieve power-proportionality. In any case, standardized monitoring interfaces for energy and resource utilization are needed to handle the diversity of hardware and support local and global control decisions based on well-known and accepted metrics, see Section 2.3.

The energy metrics collected at node and system level must be provided to the operating system and the system software to optimize important energy-consuming operations in extreme-scale systems. One of these operations is data movement, as it is recognized that today data movement and storage uses more power than computation in many HPC usages. As an example, [37] indicates explicitly that managing data movement may be an energy-efficiency technique.

Coupled to monitoring frameworks, benchmarking provides useful and complete tools for the proper evaluation of distributed systems. Many stable benchmarking suites are available for HPC systems, such as the NAS Parallel Benchmarks (NPB) [13] and LINPACK [35], which for example is used for the performance evaluation and comparison of the Top500 list entries, see www.top500.org. There are also some interesting attempts towards standards in benchmarking. The most authoritative ones are the Standard Performance Evaluation Corp (SPEC) [101] and TPC [109]. The Standard Performance Evaluation Corp (SPEC) has developed solutions that can be adopted in distributed and cloud environments, such as SPECvirt, SPEC SOA, and SPECweb. With specific regard to energy, SPEC define the SPECpower_ssj2008 benchmark [64], considering performance and energy efficiency altogether. TPC is a non-profit corporation defining transaction processing and database benchmarks through verifiable TPC performance data to the industry. The TPC benchmarks can be considered as application-level benchmarks in distributed environments and they are a basis for the evaluation of the actual performance offered by standard transactional software on the top of (physical or virtual) machines.

3. Programming models and software development

An important aspect for the development of energy-aware applications is the use of suitable programming models. This is the main topic of this section, along with a coverage of energy-aware scheduling algorithms and software development approaches.

3.1. Hierarchical programming models

Applications for Exascale computing are expected to incorporate multiple programming models. For example, a single application might incorporate components that are based on MPI and other components that are based on other paradigms. The particular combination of programming models may differ over time (different execution phases of the application) or space (e.g. some of the nodes run MPI, and others run shared-memory libraries). It is widely believed that to cope with these models, Exascale systems will require support for hierarchical programming models, which include more than two levels of today's models (such as MPI + OpenMP) [42]. The particular combination of programming models may differ over time (e.g. different execution phases of the application) or space (e.g. some of the nodes run MPI, and others run shared-memory libraries). It is widely believed that to cope with these models, Exascale systems will require support for hierarchical programming models, which may include

more than two levels of today's models (such as MPI + OpenMP) [42]. In Exascale systems, hierarchies with a higher number of levels and a larger degree of parallelism will coexist with more heterogeneous hardware, making load balancing and communication reduction a critical task. Those features can be addressed through functional portability and performance portability. Even through functional portability can be achieved due to standardized environments such as MPI or OpenCL, performance portability, however, is often a crucial issue, as the required abstractions are still not present in the current HPC code generation tools. Performance portability for future systems might require a durable abstraction expressed in programming models that do not exist for HPC code generation so far [58].

Examples for existing hierarchical programming models are the TwoL [85] and the Tlib [86] approaches, which are both defined on top of MPI and allow a flexible and hierarchical grouping of processes into groups each of which can execute multi-processor tasks (M-tasks). The M-tasks are the basic execution units and each M-task can be executed by an arbitrary number of processing cores. In the TwoL approach, the M-tasks can be combined using a coordination language, which allows the specification of input-output and control dependences between M-tasks. M-tasks without a dependence between them can be executed in parallel on disjoint groups of processors. The runtime system can select a suitable number of processing cores for each M-task and can decide which of the M-tasks are executed in parallel. If the internal M-task communication is based on collective MPI operations, it is often advantageous to execute M-tasks in parallel as this reduces the communication overhead. This approach can also be used to enable an energy-efficient execution of M-task programs [87], since the runtime system can perform the mapping of M-tasks to cores based on an energy minimization instead of a performance maximization goal. It is also possible to provide different implementations for M-tasks, such as a standard MPI implementation, a GPU implementation and a specialized implementation for MIC processors, and select the most energy-efficient implementation at runtime, depending on the hardware resources available. To support such an energy-efficient mapping, it is important that the runtime system has access to suitable monitoring facilities (see Section 2.5) or can use suitable energy metrics (see Section 2.2).

The M-task model can also be used to support performance portability, since the same M-task program can be executed on different hardware platforms and the runtime system is responsible for the appropriate mapping to the hardware resources. For different hardware platforms, the runtime system can select different mappings and different M-tasks could be executed in parallel, if this results in a faster or more energy-efficient execution.

3.2. Many task approaches

The ever-increasing performance of supercomputer systems is enabling the emergence of new problem-solving methods that require an efficient execution of many concurrent and interacting tasks, usually integrating data analysis and visualization, to maximise the productivity on Exascale systems [37]. Hence, Exascale systems will need new problem-solving approaches beyond hierarchical models.

One of the most promising candidate approaches is the many-task programming model, with the workflow model currently being the most widely used many task-like technique. An example of these tools is Swift/T, a description language and runtime system that supports the dynamic creation and execution of workflows with varying granularity on high-component-count platforms. The Swift/T system [117] provides an asynchronous dynamic load balancer (ADLB),

which dynamically distributes the tasks among the nodes [119]. The problem is that communication and synchronization for shared global resources (as files) could degrade performance in case of the absence of data locality. Current research has shown that emerging high-speed networks outperform physical disk solutions, which reduces the relevance of disk locality [7]. Thus, most solutions provided for ultrascale will be based on the intensive usage of RAM and NVRAM memory near the processors. However, existing software engineering methods and models do not provide a mechanism to express energy aspects in applications and they still rely on system services that are not energy-aware.

3.3. Energy-aware scheduling algorithms

In order to cope with energy saving while considering the particularities of Exascale systems, i.e. various levels of heterogeneity, fault tolerance, strong energy consumption constraints, it is mandatory to move towards an energy-aware resource management [22], including scheduling algorithms that are able to handle various levels of heterogeneity and the diversity of available resources [73].

Power-aware scheduling algorithms for homogeneous systems are already available for more than one decade [46, 51, 72]. Popular approaches commonly use DVFS to reduce the power consumption of processing elements during idle times and during slack times of non-critical jobs [115]. Other approaches even power off the entire computing node with only a small impact on the resulting makespan [76].

In many HPC usage scenarios, data movements consume more power than computations do, so that reducing data movement can be considered an energy-efficiency technique [37]. Therefore, energy-aware scheduling algorithms should guide the system to schedule computation jobs to the nodes containing the required data, thus avoiding costly data movement and considering the trade-offs between data locality and load balance. While traditional task clustering algorithms reduce the makespan by zeroing edges of high communication costs, a Power Aware Task Clustering (PATC) algorithm has recently been proposed [115] that guides the edge zeroing process with the objective of reducing the power consumption. The initial experiments were performed on homogeneous small clusters (100 PEs), where promising results have been obtained, specifically yielding up to 39% energy saving, which is more than double compared to 16% obtained on EADUS and TEBUS algorithms [122] that do not use DVFS. Energy-aware algorithms have also been developed and tested against heterogeneous clusters. The EETCS (Efficient-Energy based Task Clustering Scheduling) algorithm [69] significantly reduces the power consumption by shrinking the communication energy consumption when allocating parallel tasks to heterogeneous computing nodes. Another example is RADS (Resource-Aware Scheduling Algorithm with Duplication) [79], which saves up to 15% resource power consumption compared to similar algorithms.

Current scheduling and load balancing mechanisms are using meta-heuristics to solve the multi-criteria optimization problem taking into account the overload of the system and the incoming task requirements. Traditional multi-objective optimization algorithms, including population based metaheuristics aiming to estimate Pareto optimal sets, require an adaptation in order to be effective in the case of ultrascale dynamic optimization. In [22] a two-stage approach is proposed: First, a list of preliminary schedules resulting from a static multi-criteria optimization method is computed at design time. Then the schedules are adapted, using low cost operations, according to the particular requirements of the running applications and the

characteristics of the available resources. However, the approach has not been tested in the context of large scale dynamic scheduling. Another aspect to be considered is the exploration of the relationship between tasks and computing resources and the proper usage of data location [14]. Existing scheduling techniques for Exascale rely on various combinatorial optimization algorithms. For example, in [103] a new approach is proposed for simultaneously reducing the energy consumption while maximizing system performance. The method consists in computing the Pareto front of optimal solutions to the bi-objective problem of minimizing energy and makespan for a bag of tasks allocated to a set of heterogeneous compute resources.

The ultrascale dimension, the heterogeneous architecture of current parallel systems, and the need to re-schedule due to system faults have not been taken into consideration yet, especially not together with energy awareness. The task scheduler needs to support locality-awareness and be capable of supporting function shipping and data shipping as interchangeable alternatives. For this purpose, all data movement operations need to be abstracted as asynchronous tasks whose completion can trigger additional computation tasks and data movements. Moreover, the current slow meta-heuristic based mechanism should be redesigned to ensure a real-time reaction especially in the case of re-scheduling. A set of strategies, such as minimal energy consumption with deadline matching in scheduling mechanism assuming no faults, or energy aware rescheduling in the case of faults without time limits, should be defined as working conditions for the resource management system.

3.4. Energy-aware software development process

In a complex and highly distributed context, energy awareness should be applied at any level, both hardware and software, and within them. It needs to be addressed at different layers and services adopting a holistic approach. With regard to software, energy efficiency and optimization could be implemented and enforced at several levels: (a) at low level, through specific scheduling algorithms; (b) at code level, by optimizing programs and compilers and also by adopting specific, e.g. hierarchical, programming models and design patterns; and (c) at higher levels, in the software development process. In the latter case, the goal is to design the overall software architecture taking into account energy aspects and metrics, thus also considering a possible deployment in an Ultrascale infrastructure for the overall software. This approach comes from software performance engineering [99, 100], which is a systematic, quantitative technique to construct software systems that meet performance objectives. It includes performance requirements and goals into a software development process, a technique also known as performance-driven development [68, 74, 77]. As in the test-driven development [15], the performance-driven development is an iterative process composed of development and performance evaluation phases at each cycle.

The idea of an energy-aware software development process, which aims at enabling and taking into account energy efficiency and other important deployment properties and requirements at the early stages of the software lifecycle, is not new in literature but quite unexplored, especially in large scale parallel and distributed contexts. The first attempt in such a direction is green software engineering [21, 61] and development [2, 95]. All of those approaches mainly suggest adopting a green, sustainable software development process taking into account energy properties, but so far just provide some suggestions and guidelines for this purpose, mainly at lower levels, e.g. code, programming models, or design patterns. A slightly more concrete solution is discussed in [106] where a reference model for sustainable software development,

called GreenRM, is defined according to the ISO/IEC 14001 environmental requirements. But also in this case a model mainly containing only some guidelines is defined. Therefore, addressing energy, green and sustainability issues in the software development process is still an open problem.

4. Energy-efficient algorithms

As stated in the introduction, a huge reduction in the average energy cost per flop is required for Exascale systems [108]. There have been large efforts on the hardware side which aim at a reduction of the energy consumption, including new memory systems and new processor technologies with power management, see Section 1. However, while these techniques can help to significantly reduce the energy consumption of unloaded systems, their contribution to the energy consumption of loaded systems is quite limited. Most of the efforts for reducing the energy consumption of loaded systems are directed towards an efficient control of the power management techniques according to the system load, but the contribution of these techniques may not be sufficient to reach the 20 MW target for Exascale systems.

A major problem in current approaches is that the algorithms or the applications being executed have no direct interaction with the hardware system to express or control energy needs. Such an interaction is needed to bring energy-awareness to the application level and to support a goal-directed use of algorithmic changes or transformations of the application code. In this section, we give an overview of the most important aspects for the energy awareness of algorithms, including the energy characteristics of algorithms, the effect of algorithmic changes and transformations on the resulting energy consumption, as well as adaptivity approaches used to cope with the increasing heterogeneity of HPC systems resulting from the integration of accelerators such as GPU, MIC or FPGAs. Finally, we show some specific examples for energy-efficient algorithms from different areas.

4.1. Energy characteristics of algorithms

Hardware mechanisms introduced during the last years to reduce the overall energy consumption of processors (see Section 1) will also play an important role for future Ultrascale systems. Thus, it is important to study the influence of these techniques on algorithms and applications. In particular, it has to be investigated whether these techniques can be employed to reduce the energy consumption of algorithms and which specific characteristics of algorithms have an effect on the resulting energy consumption. If the influencing factors are known and can be captured quantitatively, this information can be used to tune applications towards a smaller energy consumption by applying suitable algorithmic transformation techniques.

The energy consumption E of an algorithm can be described by the power consumption P of the execution resources employed and by integrating P over the execution time of the algorithm: $E = \int_{t=t_0}^{t_{max}} P(t)dt$. Typically, the power consumption varies during the execution time of the application, depending on the specific execution situation of the application and the resulting usage of the different execution resources. The variations of the power consumption during the execution time can be measured in detail with specialized power meters and power acquisition systems [90] (see Section 2.2), but hardware counters can be used as well (e.g. Intel RAPL interface). However, the specific interaction of computation and power consumption is

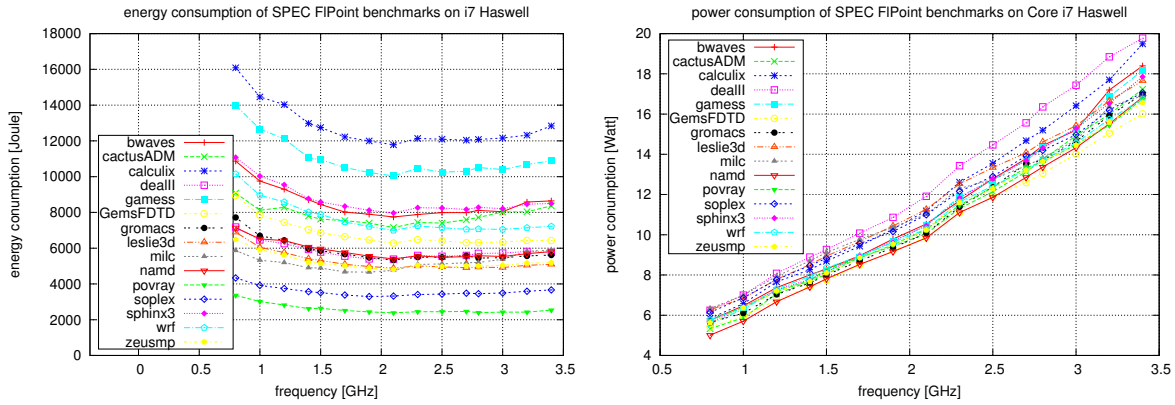


Figure 1. SPEC CPU2006 floating-point benchmarks on an Intel Core i7 Haswell processor: energy consumption (left), and power consumption (right) for varying frequencies [90]

complex and it is challenging to predict which algorithmic properties lead to which amount of power consumption at a specific point in the execution time.

The power consumption of processors comprises a dynamic and a static power consumption part [56]. The dynamic power consumption P_{dyn} is related to the switching activity of the processor during execution and it can be expected that it is smaller during processor idle periods. The static power consumption P_{stat} captures the leakage power, which becomes more important for processors with smaller transistor size, and it is present even if there is no switching activity of the transistors. It has been stated that in 2014 25%–40% of the total power consumption in server chips was caused by leakage power [48]. For DVFS processors, the dynamic power consumption increases significantly with the operational frequency f, and often, a dependence $P_{dyn}(f) = \gamma \cdot f^{\alpha}$ with $2.5 \leq \alpha \leq 3$ is assumed, where γ is a suitable parameter. The dependence of the static power consumption P_{stat} on f is typically quite small and is often neglected and assumed to be constant [56].

The average power consumption of algorithms increases with the operational frequency. Fig. 1 shows the dependence of the energy and the power consumption on the frequency for the SPEC CPU2006 floating-point benchmarks, which consist of real (sequential) programs from different application areas, (see [48] and [90] for more details). It can be observed that for most of the programs a frequency between 2.0 and 2.5 GHz leads to the smallest energy consumption. It can also be observed that different SPEC programs lead to different amounts of power consumption, which shows that there is a dependence of the power consumption on the features of the application. This effect is even larger for parallel applications, as those included in the PARSEC benchmarks that contain parallel programs from different application areas, see [17]. Fig. 2 shows the average energy and power consumption of the PARSEC benchmarks for different frequencies. As shown, the variation of the power consumption is much larger than for the SPEC benchmarks. Fig. 2 also shows that the difference between the largest and the smallest average power consumption for the different applications is more than 100% (see [89] for details). It can be concluded that parallel execution adds significant variations to the power consumptions observed.

The observation that the power consumption may be quite different for different algorithms and applications leads to the question which algorithmic properties have an influence on the resulting power consumption. For parallel applications, the speedup obtained plays a role and

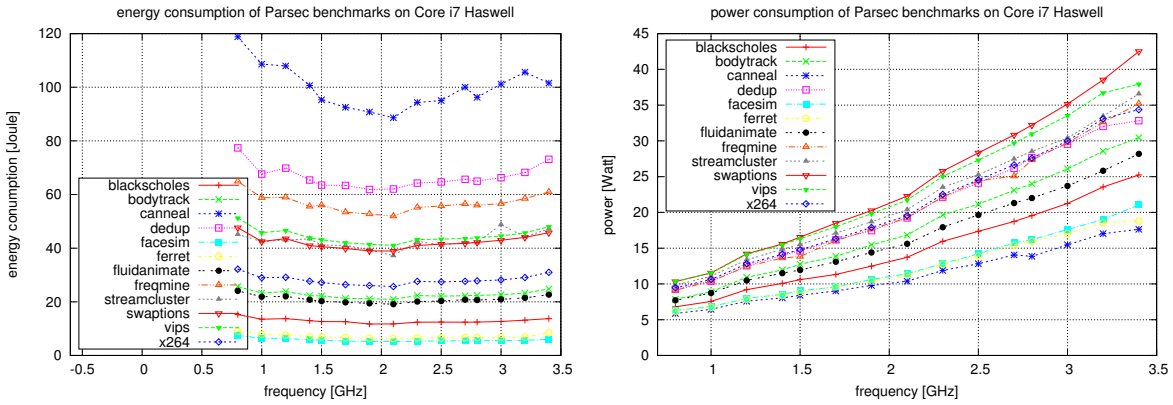


Figure 2. PARSEC benchmarks executed with eight threads on an Intel Core i7 Haswell processor: energy consumption (left), and power consumption (right) for varying frequencies [89]

it can be observed that applications with a larger speedup tend to have a larger power consumption than applications with a smaller speedup [90]. This can be explained by the fact that applications with a smaller speedup typically include more idle times during which some parts of the processing cores can be powered down, thus reducing the average power consumption. However, there are other influences that will be discussed in more detail in the next subsection.

4.2. Algorithmic techniques towards energy awareness

There are some efforts to explore the energy effects of specific programming techniques for selected algorithms, mainly from the area of linear algebra [5], with the goal of advancing towards an energy optimization of algorithms. Seminal articles in the literature demonstrate that a huge number of technical applications can be decomposed into up to 7 or 13 "Dwarfs" [9], which are a small set of common kernels with a tremendous impact on a huge number of computing-intensive applications and libraries. Thus, it seems advisable to concentrate on those kernels.

Systematic approaches that investigate the energy effects of algorithmic changes and transformations are very rare. Some recent results show that standard techniques used for performance optimization, such as tiling, have only a minor effect on the energy consumption [41], since loading and storing data to the on-chip caches constitute the largest contribution to the dynamic energy consumption. Therefore, alternative techniques, such as register tiling [91], seem to be more promising for the energy optimization of algorithms than standard techniques used for performance optimization. Currently, it is not feasible to think of a single solution for the energy optimization of algorithms, as the energy behavior of the algorithms is closely related to specific architectures.

Several approaches model the energy consumption of application programs on CPUs or GPUs [23]. These models usually distinguish between the dynamic and the static power consumption, but they do not take algorithmic properties of the application into consideration. There are also some approaches that model the energy consumption of individual algorithms by considering the operations performed [59], however these approaches are difficult to transfer to other algorithms and they require a significant effort for the analysis at the algorithmic level. Another attempt in finding a relation between properties of the algorithms and the resulting energy consumption and execution time is described in [25], but the results are only presented

at the level of micro-benchmarks. So far, there is no broad investigation that determines which algorithmic properties have which effect on the energy consumption for a specific architecture. Thus, there is a need to develop algorithm-specific energy models and mechanisms to express the energy behavior of the algorithms on the underlying system. A survey of power and efficiency issues for numerical linear algebra methods [102] identifies several major techniques for energy savings, e.g. profiling, trading off performance, static and dynamic saving, and concludes that the current techniques are application-specific and difficult to generalize. The impact of different CPU workloads on power consumption and energy efficiency is studied in [111], showing that different workloads can lead to significant differences in energy efficiency.

In addition, the architecture of different HPC and Exascale systems is expected to be quite heterogeneous and rapidly developing [37], as they might include specialized niche market devices, such as GPUs, MIC and FPGA accelerators. This perspective constitutes a major challenge for the system software, comprising the operating system, runtime system, I/O system, and interfaces to the external environment, since the system software is responsible for an effective use of the hardware resources. However, algorithmic properties of an application also play an increasingly important role and it is required that the programmer uses the right programming techniques for the specific architecture of a given HPC system. This places a large burden on the programmer to tune her or his applications towards a better performance. Since this is often quite time-consuming, autotuning approaches [114] and efforts towards Self-Adapting Numerical Software (SANS) [34] have been proposed. Those aspects will be considered in more detail in the next subsection.

4.3. Autotuning approaches towards energy efficiency

Autotuning software is able to optimize its own execution parameters with respect to a specific objective function, which was usually the execution time, but might as well be the energy consumption. The methods for autotuning are diverse, including model-based parameter optimization, or an optimization based on candidate sets generated by the autotuning software. Autotuning based on a set of equivalent candidate implementations for an algorithm considers different candidate implementations using different programming techniques for the formulations of the algorithm, which, for example, may differ in their loop structure by applying loop transformations such as loop fusion, loop interchange, loop tiling, or loop unrolling. Moreover, different parameters for the loop transformation, such as block sizes for tiling or unrolling factors, can be used. The idea of the autotuning approaches is to automatically select one of the candidate implementations for a specific HPC architecture to reach a given optimization goal, such as minimal execution time or minimal energy consumption. The selection can be made both offline or online.

Offline autotuning performs the autotuning procedure at software installation time. In this scenario, the installation of the autotuning software or library can take a significant amount of time due to an extensive evaluation of the different candidate implementations using runtime tests or energy measurements. However, at runtime, the best implementation variant selected during the installation is directly used, with little or no overhead. Offline autotuning can be applied if there is no significant dependence of the runtime of the implementation variants on characteristics of the specific input. A number of offline autotuning libraries aiming at performance optimization already exist for decades: ATLAS [116] and PHiPAC [18] for dense matrix computations; OSKI [113] and SPARSITY [52] for sparse matrix computations; or FFTW [40]

for fast Fourier transformations. Offline frameworks, such as PERI [118], SPIRAL [82] and Green [12], allow the programmer to setup an application to be autotuned for a given micro-architecture. If supported by a model-based approach [121], the installation time overhead can be reduced. Model-based approaches use an analytical model of the execution platform and the algorithm to be executed, and select a set of implementation variants and parameter values which are then tested at installation time, which may reduce the number of variants to be tested significantly.

Besides the overall execution time of a specific algorithm, additional optimization goals, such as energy consumption or computing costs, need to be considered by auto-tuners. Therefore, more sophisticated methods capable of exploiting and identifying the trade-offs among these goals are required, like those presented in [43] where the authors present and discuss results of applying a multi-objective search-based auto-tuner to optimize for three conflicting criteria: execution time, energy consumption, and resource usage. Offline autotuning approaches for energy usage vs. performance degradation in scientific applications are discussed in [107], where the authors conduct several experiments in which the tuning is performed with respect to software level performance-related tunables, such as cache tiling factors and loop un-rolling factors, as well as for the processor clock frequency. [63] presents an energy-oriented autotuning for the ATLAS library.

If the execution time of the implementation variants depends on characteristics of the specific input, offline autotuning has to be replaced by *online* autotuning, where applications are able to monitor and automatically tune themselves to optimize a particular objective (execution time, energy consumption, etc.), as in the case shown for ordinary differential equations in [55]. Online autotuning can especially be used successfully for time-stepping methods. In this case, the time steps can be performed with different implementation variants and parameter values until the best implementation variant is found. Then this implementation variant is used for the remaining time steps, as shown in [62]. A model-based pre-selection phase can be used to reduce the number of implementation variants that need to be tested at runtime. For ordinary differential equations, this approach has been applied successfully [55], and it has been shown that the autotuning overhead at runtime is not too large. An automated online performance tuning approach for general applications is provided by the Active Harmony automated runtime system [29], which allows runtime switching of algorithms and tuning of libraries and application parameters to improve the resulting performance on a given hardware platform. The system uses a server which uses a Nelder-Mead method to search through a potentially large parameter space. The server sends a parameter selection to a client, which then measures the resulting performance and sends the corresponding information back to the server. This procedure is repeated until a good parameter selection has been found.

Another example for online autotuning is PowerDial [49], which converts static configuration parameters that already exist in a program into dynamic knobs that can be tuned at runtime, with the goal of trading QoS guarantees for meeting performance and power usage goals. The system uses an online learning stage to construct a linear model of the choice configuration space which can be subsequently tuned using a linear control system. In the SiblingRivalry [8] model, requests are processed by dividing the available cores in half, and processing two identical requests in parallel on each half. Half of the cores are devoted to a known program configuration, while the other half of the cores are used for an experimental program configuration chosen using a self-adapting evolutionary algorithm. The faster configuration (either the known or

the experimental one) is always kept and the other one is terminated. The authors show that over time, this model allows programs to adapt to changing dynamic environments and often outperform the original algorithm that uses the entire system.

As mentioned before, most of existing autotuning models consider the execution time as main objective function. However, the resulting energy consumption can also be directly used as an optimization goal of an autotuning approach. This can be based on energy measurements using hardware counters as they are, for example, provided by the Intel RAPL interface (see Section 2.5) or on a model for the energy consumption of the algorithm (see [62] for more information).

4.4. Examples of energy-efficient algorithms

Examples of energy-efficient algorithms can be found in the graph theory area. In [20], the authors propose a new algorithm which solves the min cut/max flow problem on a graph. It is based on augmenting paths and building two search trees, one from the source and the other from the sink, which are reused to avoid rebuilding them from scratch. Experimental comparisons show that the algorithm is faster and minimizes the energy usage for functions in vision. Another example is [94], in which a large-scale energy-efficient graph traversal is proposed. More recently, the initiative "EDGAR: Energy-efficient Data and Graph Algorithms Research" of the Berkeley Labs has been started to design new parallel algorithms to reduce communication costs of data and graph analysis algorithms in Exascale, aiming at a reduction of the execution time and the energy consumption. An important observation in this context is that the power required to transmit data in a network also depends on the length of the wire in traditional cooper networks, i.e., data exchanges between neighboring nodes in a network require less energy than exchanges between non-neighboring nodes. The energy consumption of different MPI collective communication operations has been investigated in [112], showing that the size of the execution platform plays an important role. A quantitative analysis of the energy costs of data movements between different levels of a memory hierarchy (main memory, L3, L2 and L1 cache) has been reported in [57]. The analysis is based on a set of micro-benchmarks that continuously access data stored in a given level of the memory hierarchy and measure the resulting energy consumption. An experimental evaluation captures several benchmarks, including the NAS parallel benchmarks suite and applications from the Exascale Co-Design centers. The results show that, in current systems, scientific applications spend between 18% and 40% of their total dynamic energy in moving data and between 19% and 36% in stalled cycles. The energy consumption of different data access patterns in PGAS (Partitioned Global Address Space) models has been investigated in [54].

Sorting algorithms are among the most important fundamental algorithms in computer science and many applications depend on efficient sorting techniques. Energy efficiency also plays an important role here and using an energy-efficient sorting could help in reducing the overall energy consumption significantly. The energy consumption of different basic sorting algorithms such as odd-even sort, shellsort, or quicksort has been investigated in [123], showing that quicksort leads to the smallest energy consumption and that the choice of a suitable recursion depth for quicksort may have a large influence on the energy consumption. An external sort benchmark JouleSort for evaluating the energy efficiency of a wide range of computer systems from clusters to handhelds is described in [92]. The energy consumption of vector and matrix operations as well as sorting and graph algorithms is investigated in [93], showing that the energy

consumption depends on the memory parallelism that the algorithms exhibit for a given data layout.

Other examples of energy-efficient algorithms can be found in thread scheduling [30], financial applications [3], and big data applications [120]. All these research efforts use memorization as a techniques to avoid repeating computation by caching previous results, thus achieving a better energy efficiency in application execution.

5. Discussion

The summarizing state-of-the-art analysis of energy-aware programming has shown that there already exists a multitude of research directions and results in many areas of computing. From this current research situation, we can derive a number of open problems to be solved for a successful energy-aware programming. As energy is a cross-layer issue, we argue that a holistic energy-aware approach is needed, which requires the development of interacting interfaces between the different software and hardware layers. Such an approach will allow researchers to investigate different directions of the ETP4HPC agenda. Three of these directions are addressed below: new energy-aware algorithms for Exascale, software engineering for extreme parallelism and energy-aware systems support for managing extreme scale systems.

New energy-aware algorithms for Exascale: Advancing the state-of-the-art at an algorithmic level needs to include energy-awareness into the algorithm/application level. One way of achieving this is the introduction of interacting interfaces between the different hardware and software layers, combined with algorithm-specific mathematical energy models. We argue that this will enable a dynamic adjustment of the computation and communication characteristics of algorithms/applications with the goal to achieve a perceivable reduction of the overall energy consumption. Such a new layered approach with its interacting interfaces will also allow a direct interaction between the control of the power management and the algorithm or application being executed. With the aid of annotations, applications may provide a parameterized energy model which can be exploited to articulate a policy for managing trade-offs on different system architectures. A general goal is that future energy-aware algorithms should not only be evaluated based on FLOPs but also based on energy cost of operations.

Software engineering for extreme parallelism: To hide the complexity of the development process of algorithms and applications for Exascale systems, we propose to develop a high level language environment supporting an energy-aware software development. This language environment should be intuitive and easy to handle for application programmers from diverse application areas. This can be achieved by using a human-like language or a descriptive or graphic annotation approach. For an increase of the acceptance and usability, it is important that such a language environment allows a seamless integration of different programming models, accompanied by support for a hierarchical development of all necessary Exascale system coordination, control and monitoring functions in a reasonably human-understandable way. It necessarily should provide energy consumption indicators which system designers and developers can rely on during software development so that they can achieve a reduction of the energy footprint of the resulting program code. Considering the heterogeneity of Exascale systems, a high-level software development process is needed in order to allow a seamless integration of multiple energy-aware programming models beyond the state-of-the-art. We propose a research agenda in this field targeted towards abstract hierarchical programming models and optimized many-task programming models. The first direction will allow the annotation of power and en-

ergy consumption information by defining energy patterns and constraints in the hierarchical programming model. Based on this abstract model, one can build a general hierarchical optimization technique for collective communication algorithms, such as MPI operations, which will not be platform specific but will address the scale of the HPC platform. The second direction should evolve existing programming models to enable locality-based optimizations through the intensive usage of RAM and NVRAM memory near the processors, thus avoiding data movements, along with an energy aware scheduling that will guide the system to schedule computation jobs in the nodes containing the required data taking also into account the trade-offs between data locality and load balance.

Energy-aware system support for managing extreme scale systems: Expressing the cross-layered nature of energy can be achieved by providing system mechanisms that support energy efficiency in extreme scale systems. The first research topic is the design of metrics and tools for exporting energy features, at node and system level, to the applications through (approximate) energy monitoring and management services. These services will be provided to the upper levels of the hierarchy to allow optimizations in runtime resources, libraries and applications. The second topic should investigate the elaboration of energy-efficient data access and communication models relaying on a better exploitation of data locality and layout, and supporting the development of cross-layer locality-aware I/O software. Equally promising and complementary to the previous topics, researchers should look into energy profiling at component and application level in order to dynamically redirect the workload to those components that can yield the maximum amount of throughput. Ultimately, it should be possible to predict the energy consumption of particular code segments. This information can be used to enable a dynamic provisioning of resources, to provide the ability to manage new important resources, such as power and data motion, through an energy aware scheduler and dispatcher, and an energy-aware load balancer that is conscious of the system energy, node energy, and data-locality needs. Last but not least, we need to elaborate novel energy-aware models, APIs and tools to automatically map applications onto heterogeneous architectures trying to optimize performance over energy ratio.

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305,"Network for Sustainable Ultrascale Computing (NESUS)".

References

1. M. Abdel-Majeed, D. Wong, and M. Annavaram. Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs. In *Proc. of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 111–122, New York, NY, USA, 2013. ACM. DOI: 10.1145/2540708.2540719.

2. S. Afzal, M.F. Saleem, F. Jan, and M. Ahmad. A Review on Green Software Development in a Cloud Environment Regarding Software Development Life Cycle (SDLC) Perspective. *International Journal of Computer Trends and Technology (IJCTT)*, 4(9), 2013.

3. G. Agosta, M. Bessi, E. Capra, and C. Francalanci. Dynamic memorization for energy efficiency in financial applications. In *Proc of the 2011 Int. Green Computing Conference*

and Workshops (IGCC), pages 1–8, July 2011.

4. S. Albers. Energy-efficient Algorithms. *Commun. ACM*, 53(5):86–96, May 2010. DOI: 10.1145/1735223.1735245.

5. J.I. Aliaga1, H. Anzt, M. Castillo, J. C. Fernandez, G. Leon, J. Perez, and E.S. Quintana-Orti. Unveiling the performance-energy trade-off in iterative linear system solvers for multithreaded processors. *Concurrency and Computation: Practice and Experience*, 26(17), 2014.

6. H Amur, J Cipar, V Gupta, and GR Ganger. Robust and Flexible Power-Proportional Storage. *ACM Symposium on Cloud Computing (SOCC)*, 2010.

7. G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-locality in Datacenter Computing Considered Irrelevant. In *Proc. of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

8. J. Ansel, M. Pacula, Y. Wong, C. Chan, M. Olszewski, U. O'Reilly, and S. Amarasinghe. SiblingRivalry: online autotuning through local competition. In *Proc. of the 2012 Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 91–100. ACM, 2012.

9. K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

10. G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni. Checkpointing Strategies with Prediction Windows. In *Proc. of the 19th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 1–10. IEEE, Dec 2013.

11. V. Avelar, D. Azevedo, and A. French. PUE^{TM}: A Comprehensive examination of the metric, http://www.thegreengrid.org/~/media/WhitePapers/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf?lang=en , 2012.

12. W. Baek and T. Chilimbi. Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation. In *Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM. DOI: 10.1145/1806596.1806620.

13. D.H. Bailey, E. Barszcz, L. Dagum, and H.D. Simon. NAS parallel benchmark results. *Parallel Distributed Technology: Systems Applications, IEEE*, 1(1):43–51, 1993.

14. O. Beaumont and L. Marchal. What Makes Affinity-Based Schedulers So Efficient ?, https://hal.inria.fr/hal-00875487. October 2013.

15. Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

16. C. Belady, A. Rawson, J.Pfleuger, and T. Cader. Green Grid Data Center Power Efficiency Metrics: PUE and DCIE, 2008.

17. C. Bienia, S. Kumar, J.P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the 17th Int. Conf. on Parallel Architectures and Compilation Techniques*, October 2008.

18. J. Bilmes, K. Asanovic, C. Chin, and J. Demmel. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology. In *Proc. of the 11th Int. Conf. on Supercomputing*, ICS '97, pages 340–347, New York, NY, USA, 1997. ACM. DOI: 10.1145/263580.263662.

19. W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. Extending the scope of the Checkpoint-on-Failure protocol for forward recovery in standard MPI. *Concurrency and computation: Practice and experience*, 25(17):2381–2393, 2013.

20. Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

21. C. Calero and M. Piattini, editors. *Green in Software Engineering*. Springer, 2015. ISBN 978-3-319-08580-7.

22. H. Casanova, Y. Robert, and U. Schwiegelshohn. Algorithms and Scheduling Techniques for Exascale Systems (Dagstuhl Seminar 13381). *Dagstuhl Reports*, 3(9):106–129, 2014. DOI: 10.4230/DagRep.3.9.106.

23. H. Chen and W. Shi. Power Measurement and Profiling. In I. Ahmad and S. Ranka, editors, *Handbook of Energy-Aware and Green Computing*, pages 649–674. CRC Press, 2012.

24. G.L.T. Chetsa, L. Lefevre, J. Pierson, P. Stolf, and G. Da Costa. Beyond CPU Frequency Scaling for a Fine-grained Energy Control of HPC Systems. In *Proc. of the 24th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 132–138, Oct 2012. DOI: 10.1109/SBAC-PAD.2012.32.

25. J. Choi, M. Dukhan, X. Liu, and R.W. Vuduc. Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks. In *Proc. of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 447–457, 2014. DOI: 10.1109/IPDPS.2014.54.

26. ACPI Promoters Corporation. Advanced configuration and power interface specification. Technical report, ACPI Promoters Corporation, 11 2013.

27. IBM Corporation. IBM Systems Director Active Energy Manager. http://lwww.ibm.com /systems/director/aem. Accessed January 16, 2015.

28. Intel Corporation. Intel Datacenter Manager Energy Director. http://www.intel.com/content /www/us/en/software/intel-energy-director-product-detail.html. Accessed January 16, 2015.

29. C. Ţăpuş, I-H. Chung, and J. Hollingsworth. Active Harmony: Towards Automated Performance Tuning. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC '02, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

30. H. Cui, J. Wu, C. Tsai, and J. Yang. Stable Deterministic Multithreading Through Schedule Memorization. In *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–13, Berkeley, CA, USA, 2010. USENIX Association.

31. W. Dargie. A Stochastic Model for Estimating the Power Consumption of a Processor. *IEEE Transactions on Computers*, PP(99):1–1, 2014. DOI: 10.1109/TC.2014.2315629.

32. Q. Deng, D. Meisner, A. Bhattacharjee, T.F. Wenisch, and R. Bianchini. MultiScale: Memory System DVFS with Multiple Memory Controllers. In *Proc. of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 297–302, New York, NY, USA, 2012. ACM. DOI: 10.1145/2333660.2333727.

33. Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, and R. Bianchini. MemScale: Active Low-power Modes for Main Memory. In *Proc. of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM. DOI: 10.1145/1950365.1950392.

34. J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar. Self-adapting Numerical Software (SANS) Effort. *IBM J. Res. Dev.*, 50(2/3):223–238, March 2006. DOI: 10.1147/rd.502.0223.

35. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

36. G.C. Durelli, M. Pogliani, A. Miele, C. Plessl, H. Riebler, M.D. Santambrogio, G. Vaz, and C. Bolchini. Runtime Resource Management in Heterogeneous System Architectures: The SAVE Approach. In *Proc. of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 142–149. IEEE, Aug 2014.

37. J. Dongarra et al. The International Exascale Software Project Roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, February 2011. DOI: 10.1177/1094342010391989.

38. P. Kogge et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.

39. EU. *European technological Platform for High Performance Computing, Vision White paper*, 2012.

40. M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

41. E. Garcia, J. Arteaga, R. Pavel, and G. Gao. Optimizing the LU Factorization for Energy Efficiency on a Many-Core Architecture. In *Proc. of the 26th Int. Workshop on Languages and Compilers for Parallel Computing (LCPC 2013)*, pages 237–251. Springer LNCS 8664, 2013.

42. W. Gropp and M. Snir. Programming for Exascale Computers. *Computing in Science and Engineering*, 15(6):27–35, 2013. DOI: 10.1109/MCSE.2013.96.

43. P. Gschwandtner, J. Durillo, and T. Fahringer. Multi-Objective Auto-Tuning with Insieme: Optimization and Trade-Off Analysis for Time, Energy and Resource Usage. In Fernando Silva, Inês Dutra, and Vítor Santos Costa, editors, *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 87–98. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-09873-9_8.

44. Shin gyu K., Chanho C., Hyeonsang E., H.Y. Yeom, and Huichung B. Energy-Centric DVFS Controling Method for Multi-core Platforms. In *2012 SC Companion: High Per-*

formance Computing, Networking, Storage and Analysis (SCC), pages 685–690, Nov 2012. DOI: 10.1109/SC.Companion.2012.94.

45. M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring Energy Consumption for Short Code Paths Using RAPL. *SIGMETRICS Perform. Eval. Rev.*, 40(3):13–17, January 2012. DOI: 10.1145/2425248.2425252.

46. Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira, Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM. DOI: 10.1145/1065944.1065969.

47. B Heller, S Seetharaman, P Mahadevan, Y Yiakoumis, P Sharma, S Banerjee, and N McKeown. ElasticTree: Saving energy in data center networks. *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 17–17, 2010.

48. J.L. Hennessy and D.A. Patterson. *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

49. H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic Knobs for Responsive Power-aware Computing. *SIGPLAN Not.*, 46(3):199–212, March 2011. DOI: 10.1145/1961296.1950390.

50. HP. HP Power Advisor A tool for estimating power requirements of HP enterprise solutions. Technical report, Hewlett-Packard Development Company, 01 2013.

51. C.-H. Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Proc. of the ACM/IEEE Conference on Supercomputing*, pages 1–1, Nov 2005. DOI: 10.1109/SC.2005.3.

52. E. Im and K. Yelick. Optimizing Sparse Matrix Computations for Register Reuse in SPARSITY. In V.. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, editors, *Computational Science — ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*, pages 127–136. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-45545-0_22.

53. K. Iskra, K. Yoshii, R. Gupta, and P. Beckman. Power Management for Exascale, 2012.

54. S. Jana, J. Schuchart, and B. Chapman. Analysis of Energy and Performance of PGAS-based Data Access Patterns. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, pages 15:1–15:10, New York, NY, USA, 2014. ACM. DOI: 10.1145/2676870.2676882.

55. N. Kalinnik, M. Korch, and T. Rauber. Online auto-tuning for the time-step-based parallel solution of ODEs on shared-memory systems. *Journal of Parallel and Distributed Computing*, 74(8):2722–2744, 2014.

56. S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 2008.

57. G. Kestor, R. Gioiosa, D. Kerbyson, and Hoisie A. Quantifying the energy cost of data movement in scientific applications. In *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC*, pages 56–65, 2013. DOI: 10.1109/IISWC.2013.6704670.

58. P. Kogge and J. Shalf. Exascale Computing Trends: Adjusting to the New Normal for Computer Architecture. *Computing in Science and Engineering*, 15(6):16–26, November 2013. DOI: 10.1109/MCSE.2013.95.

59. V.A. Korthikanti and G. Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In *SPAA '10: Proc. of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 157–165, New York, NY, USA, 2010. ACM. DOI: 10.1145/1810479.1810510.

60. I. Koutsopoulos and M. Halkidi. Measurement aggregation and routing techniques for energy-efficient estimation in wireless sensor networks. In *Proc. of the 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pages 1–10, May 2010.

61. P. Lago, R. Kazman, N. Meyer, M. Morisio, H.A. Müller, and F. Paulisch. Exploring Initial Challenges for Green Software Engineering: Summary of the First GREENS Workshop, at ICSE 2012. *SIGSOFT Softw. Eng. Notes*, 38(1):31–33, January 2013. DOI: 10.1145/2413038.2413062.

62. J. Lang and G. Rünger. An Execution Time and Energy Model for an Energy-aware Execution of a Conjugate Gradient Method with CPU/GPU Collaboration. *J. Parallel Distrib. Comput.*, 74(9):2884–2897, September 2014. DOI: 10.1016/j.jpdc.2014.06.001.

63. J. Lang, G. Rünger, and P. Stöcker. Towards energy-efficient linear algebra with an ATLAS library tuned for energy consumption. In *The 2015 International Conference on High Performance Computing and Simulation (HPCS 2015)*, 2015.

64. K.-D. Lange, M.G. Tricker, J.A. Arnold, H. Block, and S. Sharma. SPECpower_Ssj2008: Driving Server Energy Efficiency. In *Proc. of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 253–254, New York, NY, USA, 2012. ACM. DOI: 10.1145/2188286.2188329.

65. E. Levy, A. Barak, A. Shiloh, M. Lieber, C. Weinhold, and H. Härtig. Overhead of a Decentralized Gossip Algorithm on the Performance of HPC Applications. In *Proc. of the 4th Int. Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '14, pages 10:1–10:7, New York, NY, USA, 2014. ACM. DOI: 10.1145/2612262.2612271.

66. A. Lewis, S. Ghosh, and N.-F. Tzeng. Run-time Energy Consumption Estimation Based on Workload in Server Systems. In *Proc. of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

67. G. Li and Y. Wang. Automatic ARIMA modeling-based data aggregation scheme in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2013(1):1–13, 2013.

68. J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in Clouds. In *CLOUD '09: Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22, Washington, DC, USA, 2009. IEEE Computer Society. DOI: 10.1109/CLOUD.2009.5071528.

69. W. Liu, H. Li, Wei Du, and F. Shi. Energy-Aware Task Clustering Scheduling Algorithm for Heterogeneous Clusters. In *Proc. of the 2011 IEEE/ACM Int. Conf. on Green Computing and Communications*, GREENCOM '11, pages 34–37, Washington, DC, USA, 2011. IEEE Computer Society. DOI: 10.1109/GreenCom.2011.14.

70. P. Llopis, J.G. Blas, F. Isaila, and J. Carretero. Survey of Energy-Efficient and Power-Proportional Storage Systems. *The Computer Journal*, 2013. DOI: 10.1093/comjnl/bxt058.

71. M. Lorenz, P. Marwedel, T. Dräger, G. Fettweis, and R. Leupers. Compiler based exploration of DSP energy savings by SIMD operations. In *Proc. of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair*, pages 838–841, 2004. DOI: 10.1145/1015090.1015314.

72. Jiong Luo, Li-Shiuan Peh, and Niraj Jha. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Proc. of the Conf. on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 11150–, Washington, DC, USA, 2003. IEEE Computer Society.

73. T. M. Lynar, R. D. Herbert, S. Chivers, and W. J. Chivers. Resource allocation to conserve energy in distributed computing. *Int. J. Grid Util. Comput.*, 2(1):1–10, 2011.

74. H. Ma. *QoS-driven composition analysis for component-based system development.* PhD thesis, Computer Science Department, Richardson, TX, USA, 2007. Adviser-Yen, I-Ling.

75. K. Ma and X. Wang. PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 13–22, New York, NY, USA, 2012. ACM. DOI: 10.1145/2370816.2370821.

76. O. Mämmelä, M. Majanen, R. Basmadjian, H. De Meer, A. Giesler, and W. Homberg. Energy-aware job scheduler for high-performance computing. *Computer Science - Research and Development*, 27(4):265–275, 2012. DOI: 10.1007/s00450-011-0189-6.

77. E. Mancini, U. Villano, N. Mazzocca, M. Rak, and R. Torella. Performance-Driven Development of a Web Services Application using MetaPL/HeSSE. In *PDP '05: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 12–19, Washington, DC, USA, 2005. IEEE Computer Society. DOI: 10.1109/EMPDP.2005.31.

78. H. McCraw, J. Ralph, A. Danalis, and J. Dongarra. Power Monitoring with PAPI for Extreme Scale Architectures and Dataflow-based Programming Models. In *Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA 2014), IEEE Cluster 2014*, pages 385–391, Sept 2014.

79. J. Mei, K. Li, and K. Li. A Resource-aware Scheduling Algorithm with Reduced Task Duplication on Heterogeneous Computing Systems. *J. Supercomput.*, 68(3):1347–1377, June 2014. DOI: 10.1007/s11227-014-1090-4.

80. A. Orgerie, M. Dias de Assuncao, and L. Lefevre. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014. DOI: 10.1145/2532637.

81. M. Pedram and Inkwon Hwang. Power and Performance Modeling in a Virtualized Server System. In *Proc. of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, pages 520–526. IEEE, Sept 2010. DOI: 10.1109/ICPPW.2010.76.

82. M. Püschel, J.M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo. SPIRAL: Code Generation for DSP Transforms. *Proceedings of the IEEE*, 93(2):211–215, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

83. R. Rajagopalan and P.K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 8(4):48–63, 2006. DOI: 10.1109/COMST.2006.283821.

84. V. Rapp and K. Graffi. Continuous Gossip-Based Aggregation through Dynamic Information Aging. In *Proc. of the 22nd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7, July 2013.

85. T. Rauber and G. Rünger. A Transformation Approach to Derive Efficient Parallel Implementations. *IEEE Transactions on Software Engineering*, 26(4):315–339, 2000.

86. T. Rauber and G. Rünger. Tlib - A Library to Support Programming with Hierarchical Multi-Processor Tasks. *Journal of Parallel and Distributed Computing*, 65(3):347–360, 2005.

87. T. Rauber and G. Rünger. Towards an Energy Model for Modular Parallel Scientific Applications. In *IEEE International Conference on Green Computing and Communications (GreenCom 2012)*, pages 523–532. IEEE, 2012. DOI: 10.1109/GreenCom.2012.79.

88. T. Rauber and G. Rünger. Modeling and Analyzing the Energy Consumption of Fork-Join-based Task Parallel Programs. *Concurrency and Computation: Practice and Experience*, 27(1):211–236, 2015. DOI: 10.1002/cpe.3219.

89. T. Rauber, G. Rünger, and M. Schwind. Energy Measurement and Prediction for Multithreaded Programs. In *Proc. of the High Performance Computing Symposium*, HPC '14, pages 20:1–20:9, San Diego, CA, USA, 2014. Society for Computer Simulation International.

90. T. Rauber, G. Rünger, M. Schwind, H. Xu, and S. Melzner. Energy Measurement, Modeling, and Prediction for Processors with Frequency Scaling. *The Journal of Supercomputing*, 70(3):1451–1476, 2014. DOI: 10.1007/s11227-014-1236-4.

91. L. Renganarayana, U. Bondhugula, S. Derisavi, A. E. Eichenberger, and K. O'Brien. Compact multi-dimensional kernel extraction for register tiling. In *Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis*, page 45. ACM, 2009.

92. S. Rivoire, M. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 365–376. ACM, 2007. DOI: 10.1145/1247480.1247522.

93. S. Roy, A. Rudra, and A. Verma. Energy Aware Algorithmic Engineering. In *Proceedings of the 22nd Int. Symp. on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, MASCOTS '14, pages 321–330. IEEE Computer Society, 2014. DOI: 10.1109/MASCOTS.2014.47.

94. N. Satish, C. Kim, J. Chhugani, and P. Dubey. Large-scale Energy-efficient Graph Traversal: A Path to Efficient Data-intensive Supercomputing. In *Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 14:1–14:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

95. S.S. Shenoy and R. Eeratta. Green software development model: An approach towards sustainable software development. In *2011 Annual IEEE India Conference (INDICON)*, pages 1–6, Dec 2011. DOI: 10.1109/INDCON.2011.6139638.

96. Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai. Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-cell VLSI Designs. *ACM Trans. Des. Autom. Electron. Syst.*, 15(4):28:1–28:37, October 2010. DOI: 10.1145/1835420.1835421.

97. J. Shinde and S.S. Salankar. Clock gating A power optimizing technique for VLSI circuits. In *Proc. of the 2011 Annual IEEE India Conference (INDICON)*, pages 1–4, Dec 2011. DOI: 10.1109/INDCON.2011.6139440.

98. K. Singh, M. Bhadauria, and S. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009. DOI: 10.1145/1577129.1577137.

99. C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

100. C.U. Smith and L.G. Williams. *Performance Solutions: a Practical Guide to Creating Responsive, Scalable Software*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

101. SPEC Consortium. Standard Performance Evaluation Corp (SPEC), www.spec.org, 2015.

102. L. Tan, S. Kothapalli, L. Chen, O. Hussaini, R. Bissiri, and Z. Chen. A survey of power and energy efficient techniques for high performance numerical linear algebra operations. *Parallel Computing*, 40:559–573, 2014.

103. K.M. Tarplee, R. Friese, A.A. Maciejewski, and H.J. Siegel. Efficient and scalable computation of the energy and makespan Pareto front for heterogeneous computing systems. In *Proc. of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 401–408, Sept 2013.

104. The Green Grid Consortium. The Green Grid Website: http://www.thegreengrid.org/, 2014.

105. E. Thereska, A. Donnelly, and D. Narayanan. Sierra: practical power-proportionality for data center storage. In *EuroSys '11: Proc. of the 6th Conference on Computer systems*. ACM Request Permissions, April 2011.

106. M. Thiry, L. Frez, and A. Zoucas. GreenRM: Reference Model for Sustainable Software Development. In *Proc. of the 26th International Conference on Software Engineering and Knowledge Engineering*, pages 39–42, 2014.

107. A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely. Auto-tuning for Energy Usage in Scientific Applications. In *Proc. of the 2011 Int. Conf. on Parallel Processing - Volume 2*, Euro-Par'11, pages 178–187, Berlin, Heidelberg, 2012. Springer-Verlag. DOI: 10.1007/978-3-642-29740-3_21.

108. M.E. Tolentino and K.W. Cameron. The Optimist, the Pessimist, and the Global Race to Exascale in 20 Megawatts. *IEEE Computer*, 26(4):95–97, 2012.

109. TPC Consortium. Transaction Processing Performance Council (TPC), www.tpc.org, 2015.

110. U.S. Environmental Protection Agency. The ENERGYSTAR Website: http://www.energystar.gov/, 2015.

111. J. v. Kistowski, H. Block, J. Beckett, K. Lange, J. Arnold, and S. Kounev. Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive

Workloads. In *Proc. of the 6th ACM/SPEC Int. Conf. on Performance Engineering*, ICPE '15, pages 223–234, New York, NY, USA, 2015. ACM. DOI: 10.1145/2668930.2688057.

112. A. Venkatesh, K. Kandalla, and D. Panda. Evaluation of Energy Characteristics of MPI Communication Primitives with RAPL. In *Proc. of the International Workshop on High Performance Power-Aware Computing at IPDPS*. IEEE, 2013.

113. R. Vuduc, J. Demmel, and K. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. In *Proc. SciDAC, J. Physics: Conf. Ser.*, volume 16, pages 521–530, 2005. DOI: 10.1088/1742-6596/16/1/071.

114. R.W. Vuduc. Autotuning. In *Encyclopedia of Parallel Computing*, pages 102–105. 2011.

115. Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kołodziej, Rajiv Ranjan, Cheng zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661 – 1670, 2013.

116. R.C. Whaley and J.J. Dongarra. Automatically Tuned Linear Algebra Software. In *Proc. of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

117. M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, and I. Foster. Swift: A Language for Distributed Parallel Scripting. *Parallel Computing*, 37(9):633–652, September 2011. DOI: 10.1016/j.parco.2011.05.005.

118. S. Williams, K. Datta, J. Carter, L. Oliker, J. Shalf, K. Yelick, and D Bailey. PERI - auto-tuning memory-intensive kernels for multicore. *Journal of Physics Conference Series*, 125(1), July 2008.

119. J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, and I.T. Foster. Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In *Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 95–102, May 2013. DOI: 10.1109/CCGrid.2013.99.

120. D. Yamada, Sonobe T., H. Tezuka, and M. Inaba. Grid Spider: A framework for Data-Intensive research with Data Process Memorization Cache. In *Proc. of the 4th Int. Confererence on Resource Intensive Applications and Services. INTENSIVE 2012*, pages 5–8, 2012.

121. K. Yotov, X. Li, G. Ren, M.J. Garzarán, D. Padua, K. Pingali, and P. Stodghill. Is search really necessary to generate high-performance BLAS? *Proceedings of the IEEE*, 93(2):358–386, 2005.

122. Ziliang Z., A. Manzanares, B. Stinar, and Xiao Q. Energy-Aware Duplication Strategies for Scheduling Precedence-Constrained Parallel Tasks on Clusters. In *Proc. of the 2006 IEEE Int. Conf. on Cluster Computing*, pages 1–8, Sept 2006. DOI: 10.1109/CLUSTR.2006.311860.

123. I. Zecena, Ziliang Zong, Rong Ge, Tongdan Jin, Zizhong Chen, and Meikang Qiu. Energy consumption analysis of parallel sorting algorithms running on multicore systems. In *Proc. of the 2012 International Green Computing Conference (IGCC)*, pages 1–6. IEEE, June 2012. DOI: 10.1109/IGCC.2012.6322290.

124. S. Zheng, P. Zhang, and Zhang Q. A Routing Protocol Based on Energy Aware in Ad Hoc Networks. *Information Technology Journal*, 9(4):797–803, 2010. DOI: 10.3923/itj.2010.797.803.

Received February 27, 2015.

DOI: 10.14529/jsfi150206

Energy Efficiency for Ultrascale Systems: Challenges and Trends from Nesus Project

Michel Bagein[1], *Jorge Barbosa*[2], *Vicente Blanco*[3], *Ivona Brandic*[4], *Samuel Cremer*[1], *Sébastien Frémal*[1], *Helen D. Karatza*[5], *Laurent Lefevre*[6], *Toni Mastelic*[4], *Ariel Oleksiak*[7], *Anne-Cécile Orgerie*[8], *Georgios L. Stavrinides*[5], *Sébastien Varrette*[9]

Energy consumption is one of the main limiting factors for designing and deploying ultrascale systems. Therefore, this paper presents challenges and trends associated with energy efficiency for ultrascale systems based on current activities of the working group on "Energy Efficiency" in the European COST Action Nesus IC1305. The analysis contains major areas that are related to studies of energy efficiency in ultrascale systems: heterogeneous and low power hardware architectures, power monitoring at large scale, modeling and simulation of ultrascale systems, energy-aware scheduling and resource management, and energy-efficient application design.

Keywords: energy and power measurement, data acquisition tools, energy modeling, scheduling, applications, heterogeneous infrastructures, ultrascale computing.

Introduction

Energy consumption is one of the main limiting factors for designing and deploying Ultrascale systems. While energy monitoring and reporting combined with energy efficient design of applications and frameworks is currently explored and can be reachable at small scale, dealing with such concepts at ultra large scale is an open issue. Extracted from activities in working group on "Energy Efficiency" in the European COST Action Nesus IC1305[10], this article will present current activities, challenges and trends associated with energy efficiency for ultra scale systems.

Reaching levels of efficiency that are sufficient for ultrascale systems requires advances in several relevant areas as illustrated in fig. 1. First of all the reduction of power usage need to reach ultra scales is not possible without disruptive innovations in hardware. In particular, the use of new low power Systems on Chips (SoCs) and exploiting heterogeneity on various levels are promising trends. However, changes in hardware alone are not enough without proper assignment of applications to hardware and without optimising applications to take full advantage of hardware architectures. For instance, the use of hardware accelerators while improving efficiency usually requires specific implementation. Another important aspect needed to improve energy efficiency is accurate and real time power monitoring which can be a challenge in ultra scale systems per se. Monitoring and knowledge about hardware architecture and application characteristics must be applied by scheduling and resource-management techniques that are

[1]Université de Mons, Mons, Belgium
[2]Universidade do Porto, Porto, Portugal
[3]Universidad de La Laguna, Santa Cruz de Tenerife, Spain
[4]Vienna University of Technology, Vienna, Austria
[5]Aristotle University of Thessaloniki, Thessaloniki, Greece
[6]Inria Avalon, LIP Lab., Ecole Normale Supérieure of Lyon, Lyon, France
[7]Poznan Supercomputing and Networking Center, Poznan University of Technology, Poznan, Poland
[8]CNRS, IRISA, Rennes, France
[9]University of Luxembourg, Esch-sur-Alzette, Luxembourg
[10]Nesus European COST Acion I1305 : http://www.nesus.eu/

moving from pure performance goals to energy consumption and thermal issues. For all these areas modeling and simulation techniques are needed to analyse energy efficiency of hardware, applications and whole computing systems at ultrascale level. On top of these areas the general challenge is to integrate advances from all of them and to take a holistic approach to the energy-efficiency analysis and management of ultrascale systems. For example this approach should study software-hardware co-design or dependencies between IT systems components and infrastructure (including thermal management, cooling, and appropriate metrics for assessment of energy-efficiency). Other emerging areas of research include multi data centre management, taking into consideration energy availability and price, and environmental issues.

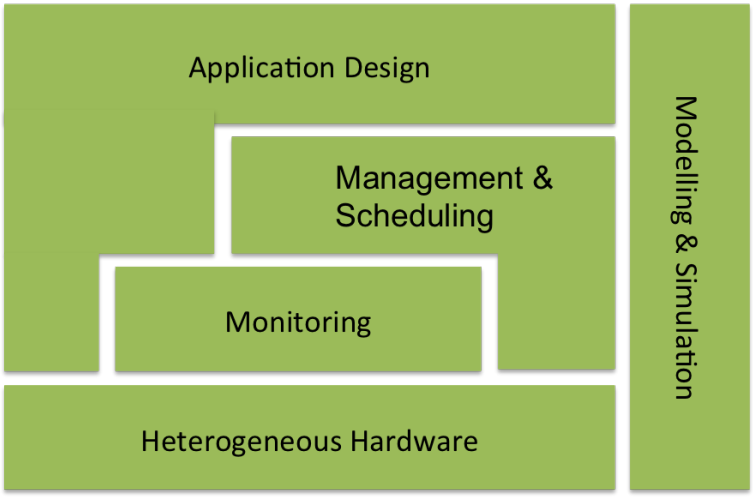


Figure 1. Major areas that affect the analysis of energy efficiency in ultrascale systems

This paper concentrates on activities related to energy efficiency of ultrascale systems being conducted within the Nesus project. Other surveys that deal with energy-efficiency in certain areas of data centres include energy efficiency in cloud computing [60], large scale distributed systems [63], and data centres [24].

Section 1 describes recent innovations at infrastructure level. Section 2 presents power and energy monitoring devices and frameworks for distributed systems and associated challenges on ultrascale. Section 3 addresses energy modeling and simulation while section 4 focuses on resource management and scheduling. Section 5 explores challenges in designing energy efficient large scale applications. The final section concludes this article.

1. Heterogeneous infrastructures : a key for energy efficiency at ultrascale level

Up to now, most HPC systems are built on general purpose multi-core processors that use the x86 and Power instruction sets (both to ensure backward productivity and enhance programmers productivity). They are mainly provided by three vendors: Intel (around 85.8% of the systems listed in the latest Top500 list[11]) and AMD (5.6%) for x86-64 bits CPUs and IBM (7.8%) for the RISC Power Architecture used by IBM POWER microprocessors. While initially designed to target the workstation and laptop market, these processors admittedly offer very good single-thread performance (for instance 16 double-precision Floating-Point Operations

[11] Top500 List of November 2014 – `http://top500.org`

per seconds (FLOPs) per cycle for Intel Nehalem), yet at the price of a relative low energy efficiency. For instance, tab. 1 details the Thermal Design Power (TDP) of the top four processors technologies present in the latest Top500 list.

Table 1. TDP of the main processors technologies present in the Top500 List (Nov. 2014)

Processor Technology	Top500 Count	Model Example	max. TDP	
Intel Sandybridge	231 (46.2%)	Xeon E5-2680 8C 2.7GHz	130W	16.25W/core
IBM Power BQC	25 (5%)	Power BQC 16C 1.6GHz	65W	4.1W/core
AMD x86_64	12 (2.4%)	Opteron 6200 16C "Interlagos"	115W	7.2W/core

In parallel, the main challenge opened to the HPC community remains the building an Exascale HPC system by 2020 while staying within a power budget of around 20 MW. As current measures within a typical blade server estimate that 32.5% of its supplied power are distributed to the processor, some simple arithmetic permit to estimate the average consumption per core in such an EFlops system: around 6.4 MW would be dedicated to the computing elements, and we can quantify their number by dividing the target computing capacity (1 EFlops) by the one of the current computing cores (16 GFlops) thus leading to approximately 62.5×10^6 cores within an Exascale system. Consequently, such a platform requires a maximal power consumption of **0.1W per core**. In order to achieve this ambitious goal, alternative low power processor architectures are required. In this context, two main directions are currently explored: (1) relying on accelerators and co-processors (either General-Purpose Graphics Processing Unit (GPGPU) such as the Nvidia Tesla cards, or Many Integrated Coress (MICs) *e.g.* Intel Xeon Phi) or (2) using the low-power processors (ARM, Intel Atom etc.) primarily designed for the mobile and embedded devices market. In parallel, the Cloud Computing (CC) paradigm has emerged as a promising approach to consolidate in a cost-effective way existing computing platforms so that many companies and researchers are engaged in efforts of scaling system software to meet the requirements of diversifying on-line cloud applications and services. Therefore future Ultrascale Computing Systems (UCSs) are envisioned as hybrid systems composed of heterogeneous resources and platforms ranging from "traditional" High Performance Computing (HPC) systems, CC infrastructures and ultra low-power computing systems. Another reason for this tendency toward an heterogeneous design is that there is no single approach which is optimal for all computing needs. Heterogeneous computing has become a necessity as it embodies the use of multiple approaches to computational processing (CPUs, GPUs, FPGAs, etc.) to achieve superior throughput for each big data workload. Of course, assuming the applications run on top of the platform are able to adapt to such heterogeneity, major power savings can be performed. In the next paragraphs, we will detail the reason behind these hardware and virtualization trends justifying there integration within an energy-efficient UCS platform.

Low-power processors. This growing market is nowadays considered as a credible basis to build HPC components. For instance, the aim of the Mont-Blanc project [3], launched October 1st 2011, is to design supercomputers from ARM processors, using 15 to 30 times less energy than conventional HPC platforms. The first part of the project (for the time period 2011-2013) lead to a proof-of-concept 120 MFlops/W cluster named `Tibidabo` based on NVidia Tegra2 Server-on-Chip (SoC) (128 nodes featuring ARM Cortex A9 processors having 2 cores at 1 GHz frequency). It is worth mentioning that the `Viridis` ARM cluster of the University of

Luxembourg (UL) HPC platform[12], released at approximately the same moment, outperforms `Tibidabo` since it has achieved a measured performance of 572 MFlops/W [49]. In all cases, the Mont-Blanc project in currently in its second phase (until 2016) to continue on these efforts using a total budget of 11,4M€. Similarly, the EuroCloud [2] project was focused on building ARM-based Server-on-Chip, integrating 3D DRAM to provide a very dense low-power server. The target was reaching a 10 times improvement in cost and energy-efficiency compared to state-of-the-art servers. Generally, the trend of utilizing large number of low-power processors to replace high-end CPUs is becoming more and more popular. Indeed, many different studies [46, 64, 66] prove that such embedded processors provide significant power savings when compared to regular hardware architecture.

More recently in [49], a comparative study has been performed as regards the performance and energy efficiency of cutting-edge high-density HPC platform enclosures featuring either very high-performing processors (such as Intel Core i7 or E7) yet having low power-efficiency, or the reverse i.e. energy efficient processors (such as Intel Atom, AMD Fusion or ARM Cortex A9) yet with limited computing capacity. The performed analysis confirms that when running time-critical applications, it is still better to choose performance-efficient CPUs, such as Intel Xeon E7 or Intel Core i7, as executions on these processors were considerably shorter compared to low-power devices. On the other hand, their power draw was very high. The competition between power-efficient devices is fierce and there is no single winner in the field of computational performance. The results are dependent on the executed benchmark. However, when considering the Performance per Watt (PpW) metric the ARM Cortex A9 always achieves the best results, sometimes even better than Intel when power-greedy processors are considered. Moreover, out of the three mentioned low-power CPUs, it executes applications in the shortest period of time and its total energy consumption is the least, in some cases up to 12 times lower than the energy usage of the rest of the CPUs.

Accelerators and co-processors. If the idea of benefiting from hardware heterogeneity (between Intel, AMD or ARM processors) is hopefully justified with the above-mentioned study, it becomes even more prominent due to the advent of General-Purpose Graphics Processing Unit (GPGPU) accelerators. Graphics Processing Units (GPUs) offer a greater performance per watts than conventional CPUs – for instance the Nvidia Tesla M2090 cards present in the HPC platform of the UL feature 512 cores for a TDP of 225W thus leading to 0.44W/core. Also, several application are inherently ready to take benefit from the optimized vector instructions featured by these decives. For instance, real-world Molecular Dynamics and Bio-informatics applications such as AMBER, mpiBLAST or MrBAYES can obtain great speedup when running on GPU-enabled systems. Data and graphics presented in selected NVIDIA benchmarks[13] show respectively a 2.9X and 7.4X acceleration for NAMD and AMBER applications compared to single CPU node execution when using one additional NVIDIA K20X GPU accelerator. Moreover, GPU accelerators were proven of relevance (together with ARM-based architectures) over a series of 5 Map-Reduce benchmarking applications in [34]. Since GPGPU systems are also quite energy-efficient, it definitively makes sense to try whenever possible to rely on such platforms, with the caveat that the programming cost is far from negligible, and end users are generally

[12] `http://hpc.uni.lu`

[13] 2013 NVIDIA Computational Chemistry & Biology benchmarks – `http://www.nvidia.com/docs/IO/122634/computational-chemistry-benchmarks.pdf`

reluctant to spend the necessary time to adapt their workflow to use accelerators (whether GPU or co-processor based).

Finally, UCS systems could also benefit from recent advances in the domain of Field-Programmable Gate Arrays (FPGAs) since such systems have been gaining momentum throughout genomics and life sciences. Programmable "on the fly", FPGAs are a way of achieving hardware-based, application-specific performance without the time and cost of developing specific applications. FPGAs work well on many bioinformatics applications, for example those that do searching and alignment which are highly parallelisable[14].

Virtualization and Cloud Computing (CC). At an intermediate level (between software and hardware), virtualization is emerging as the prominent approach to mutualize the energy consumed by a single server running multiple Virtual Machines (VMs) instances. This approach, commonly designated as Cloud Computing (CC) [17, 87] is increasingly advertised as THE solution to most IT problems. In this paradigm, shared IT resources are dynamically allocated to customer tasks and environments. It allows users to run applications or even complete systems on demand by deploying them on the Cloud that acts like a gigantic computing facility. In an HPC context, it is thus clear that the integration of the CC paradigm should be studied since there is a strong wish, at least from commercial entities (e.g. Google, Apple, Microsoft or Amazon), to serve HPC needs through Infrastructure-as-a-Service (IaaS) platforms to eventually replace in-house HPC platforms. However, little understanding has been obtained about the potential overhead in energy consumption and the throughput reduction for virtualized servers and/or computing resources, nor if it simply suits an environment as high-demanding as a HPC platform.

Our previous studies [44, 88] demonstrate that the overhead induced by the Cloud hypervisors cannot be neglected for a pure HPC workload – namely the High Performance Linpack (HPL) benchmark. Results show the fast degradation in the computing efficiency when the number for computing nodes is artificially increased through virtualization. Nevertheless, it is true that the above mentioned studies focus on a pure HPC workload (i.e. heavy computing and communication intensive) whereas we witness in general a large variety of job types in our clusters. For instance, sequential, mono-process or bag-of-tasks applications executed on a cluster in a embarrassingly parallel way will not be penalized as much by running in a VM instance.

As a conclusion, the heterogeneity in computing resources is a necessity for UCSs systems to ensure both a flexible adaptation to HPC workloads and significant power-savings. Yet taking advantage of these heterogeneous resources assumes the possibility of measuring with a reasonably good accuracy the performance and the energy-efficiency of the system. The next section details this aspect.

2. Power monitoring and profiling of ultrascale context

To enable energy optimization across the whole stack of an ultrascale high performance computing system, it is desirable to gain insight into the consumption of existing systems at all possible levels. Ideally, system designers and operators, as well as application developers, should be able to easily access precise power data ranging from whole systems to individual

[14] http://www.scientific-computing.com/news/news_story.php?news_id=2245

components inside a computation node. It should also be easily attributable to the code being executed, again ranging from entire processes to parts of specific threads.

Currently, only some of this data are usually available, which is provided through as many different interfaces as measurement devices and vendors are involved. There are grounds for a standardization of energy data acquisition. Several software and hardware tools are being used to analyze energy consumption in computing systems and data centers. Different levels of measurement are provided, with each tool offering its own tradeoff between precision and intrusiveness.

We can classify the set of existing tools according to the position within the target system where they are integrated.

2.1. External devices

These are energy measurement systems that have been used to measure energy consumption and efficiency outside the experimental nodes. Measurements can be performed without interfering with an experiment, but they may be infeasible for experiments that demand high precision measures. Examples of this kind of devices are dedicated power-meters such as the Kill-A-Watt [65] and Watt's Up Pro [91]; power distribution units (PDUs) with metering capabilities; PowerPack [43], which performs out-of-band measurements from various sources; vendor-specific external systems such as IBM Power Executive; PowerScope [40], which uses a digital multimeter controlled using customized system calls; and Energy Endoscope [77], that offers detailed real time measurements.

2.2. Intranode devices

The intranode group is composed of highly customized hardware instrumentation tools, such as the PowerMon line of devices [22], placed between a node's power supply and mainboard; or the PowerInsight device [57], designed for component-level instrumentation of commodity hardware; the ARM Energy Probe [15], integrated with the ARM development toolchain; and the Linux Energy Attribution and Accounting Platform (LEA^{2}P) [73].

2.3. Hardware sensors

Many recent components offer a number of built-in sensors able to directly report consumption data at runtime. These may be exposed as performance counters or through a vendor-provided API. For example, the Intel Running Average Power Limit (RAPL) interface reports per-package estimates of total energy consumed on Intel Sandy Bridge CPUs and later; the Nvidia Management Library (NVML) interface can query instant power draw values from recent Nvidia Tesla GPUs; some motherboards report power draw value through extensions to the Intelligent Platform Management Interface (IPMI)

2.4. Software interfaces

The Performance API (PAPI) [28] recently added a number of components which can access a system's integrated energy and power consumption. Being a mature library, it is a compelling choice for hardware counter data acquisition. However, we feel that there is a place for a higher-level abstraction with narrower scope and support for devices other than hardware counters

(as in external devices), which may build upon any hardware counter interfaces (in fact, work integrating PAPI as a low-level provider to our library is underway).

PowerAPI, from Sandia National Laboratories [74], is a recent attempt to standardize access to power measurement data and power control. It is comprised of an API specification and a reference implementation which already implements tightly coupled support for some energy data sources. The platform and user role models defined in this specification, however, are aimed towards HPC rather than cloud systems.

The Energy Measurement Library (EML) [29, 30] is a software library created to simplify analysis of energy consumption in heterogeneous systems. It provides a very simple interface for energy data acquisition and automatic run-time detection of available vendor interfaces and supported devices. These features abstract platform-specific details away from instrumentation code, greatly speeding up the measurement and experimentation process.

Other software interfaces which provide directly measured energy or power related information are: the `perf_events` [53] subsystem of the Linux kernel, which exports a variety of hardware counters to Linux userspace applications, the power measurement library `pmlib` [18], which implements a client/server architecture for out-of-band data collection from instrumented code; LIKWID, a performance-oriented library that accesses performance counters in x86 architectures [84].

Finally, a number of software interfaces provide similar energy information which is not directly measured, but estimated from runtime metrics and a certain analytical consumption model instead. These include the PowerAPI from Spirals research group [27] or the Energy Consumption Library [71].

2.5. Deploy to ultrascale level

Both physical wattmeters (either external, intranode or hardware sensors) and software-based interfaces present advantages and drawbacks which are amplified at the ultrascale level, while both solutions are desirable to monitor and to save energy.

Indeed, physical external wattmeters are the only solution to obtain a global view of the energy consumption of an ultrascale system: including the air conditionning system and the power units, which are non-negligible energy consumers. Such a global view is useful for the system administrator, to size the emergency power supply systems for instance, or to have an accurate trace in time of the electricity bill. It is also necessary for the system's task scheduler in order to avoid hot spots and balance the load energy-efficiently.

From the users' and applications' point of view, a much more detailed view is required, since their goal and scale are different. Indeed, users need a higher measurement frequency (which may go below the second) and a more focused view: at a node, core or even thread level. Software-based tools are more suitable for such a fine- grained view in spite of their intrusive behavior.

The challenge at ultrascale level consists in being able to combine both views and make them available through a usable API. For instance, energy monitoring information for a 150 nodes platform equipped with external wattmeters providing one measurement per second corresponds to approximately 70 GB of data annually [35]. At ultrascale level, the amount of energy monitoring data becomes rapidly unmanageable, even with scalable monitoring tools such as Ganglia [42].

In this context, an energy monitoring system may consist in an hybrid solution offering fine-grained views for short time periods based on software interfaces, and aggregated metrics

based on physical wattmeters for the higher views over longer time ranges. This system could rely on round-robin databases for scalability purposes. It could even be tunable by the user in order to avoid collecting, storing and processing useless energy monitoring data.

3. Energy modeling and simulation in ultrascale systems

As ultrascale systems gain momentum, their power provisioning has become a key concern. A significant amount of the energy consumed by the system's components is transformed into heat, which may harm the reliability and the overall performance of the system. The higher the energy consumption of an ultrascale system, the higher its operating and cooling cost is. Therefore, energy efficiency has become a critical aspect of ultrascale systems that attracts significant attention from the research community.

The ever increasing size and scale of such systems, inevitably leads to higher energy consumption. This has forced scientists to re-examine the full spectrum of scheduling and resource allocation algorithms. Accurate measurement at extreme scale is not an easy task. On the contrary, modeling and simulation techniques can be used to evaluate the performance of such systems, regardless of their scale. They provide tools for easy and safe experimentation, in order to investigate ways to reduce the energy consumption of such systems, assuring at the same time satisfactory system performance and quality of service. For example, modeling and simulation approaches can be utilized in case of energy efficient real-time scheduling in ultrascale systems, where applications must meet their deadline and therefore a multi-criteria scheduling policy is required to be employed.

A simulation model is preferred over analytical techniques, due to the fact that complex systems would require much simplification in order to be studied analytically. With simulation, we can evaluate the system for various workloads with different characteristics and for different system configurations, by replacing, adding and removing system components easily. By controlling the simulation parameters of the performed experiments, useful conclusions can be drawn about the impact of correlating factors [78].

3.1. Simulating multi-criteria scheduling techniques

In [83], two metrics are used which describe the computational power and energy efficiency of the system. Based on these metrics, the authors propose scheduling policies for hard real-time tasks that are executed on a heterogeneous cluster with power-aware *dynamic voltage and frequency scaling (DVFS)* processors. Clusters are often part of ultrascale systems, used as an underlying infrastructure in computational grids and clouds. Therefore, the findings of this research are also useful for ultrascale systems. In this study, the authors propose multi-criteria scheduling policies, in order to reduce the energy consumption of a power-aware heterogeneous cluster, meeting at the same time the task deadlines. The cluster consists of DVFS processors that can adjust their clock frequency, based on the performance requirements of the system. Furthermore, since hardware failures often occur in large-scale systems, the impact of replacing high-performance processors with high-efficiency processors is studied. The main reason that simulation experiments were performed instead of tests in a real system, is that it is practically impossible to isolate the energy consumption of the processors in a real system, as other factors may affect its energy consumption.

The service capacity of a system is determined by the number and service rate of its processors. In a simulation experiment, this information is required in order to adjust the arrival rate of the tasks, so that the system is balanced. The ratio between the arrival rate and the overall system service rate represents the load of the system. While setting the parameters for a small system is relatively an easy task, particular attention is needed in case of complex large-scale systems, so that the system stability is guaranteed. By increasing or decreasing the number of processors in the system model and by adjusting the arrival rate of the tasks, the performance of the system can be examined at different scales via simulation.

There have been several other research papers that examine by simulation energy efficient scheduling techniques in large-scale systems. For example, in [90], the authors propose a power-aware scheduling algorithm for clusters where virtual machines (VMs) are dynamically provided for executing tasks. Their algorithm is implemented in a simulator and in an experimental two-node multi-core cluster. In [81], the authors study the energy savings that can be gained from the application of DVFS and *dynamic power management (DPM)*, in a real-time 2-level heterogeneous grid system. DPM puts the idle components of the system into low-power sleep states whenever this is possible. Simulation results reveal that under certain conditions, these techniques can work together and achieve significant energy savings.

3.2. Modeling complex workloads

A large percentage of the workload submitted to large-scale systems is *bag-of-tasks (BoT)* applications. Each BoT is a collection of independent tasks that do not need to communicate with each other and they can run on any processor and in any order. BoT scheduling is extensively studied in the literature. In [82], Terzopoulos and Karatza view BoT scheduling from an energy efficiency perspective. They apply a DVFS mechanism to a heterogeneous cluster environment where BoTs are submitted dynamically. They also consider high-priority tasks in the workload. As tasks are distributed to the most appropriate computing nodes, faster processors could be congested with tasks while slower ones could remain idle for longer periods. Furthermore, when scheduling BoT applications, some tasks may finish sooner than other sibling tasks when they are executed by fast processors. This can result in large synchronization delays. In this case, by exploiting DVFS, these tasks could be executed at lower speeds and consume minimum amounts of energy. Simulation experiments show that by applying the proposed DVFS mechanism when BoTs are executed, energy savings can be achieved without affecting the execution of high-priority tasks.

In [54] users submit and run their tasks on the provider's resources with Service Level Agreements (SLAs). In SLAs, consumers and providers agree upon resource usage, pricing and quality of service commitments. Tasks are considered to be BoT applications with deadline constraints. The proposed power-aware scheduling algorithms are tested through simulation. In [26], the authors propose a Power Aware Job Scheduler (PAJS) for high performance computing clusters, where energy efficiency is achieved and response time is minimized by scaling the supply voltage. The proposed scheduler targets the optimization of both, power and performance. The key novelty in this research is the utilization of a *dynamic threshold-voltage scaling (DTVS)* technique for the reduction of cumulative power utilized by each node in the cluster. Furthermore, independent tasks within a job are scheduled to the most suitable computing nodes. Simulation results show the effectiveness of DTVS.

3.3. Energy efficient cloud computing

Cloud computing offers many benefits to users. However, large-scale virtualized data centers require large amounts of electrical power, resulting in high operating costs. Due to the variability of the workload, the VM placement in the servers should be optimized continuously in an online manner. In [23] Beloglazov and Buyya conduct competitive analysis and prove competitive ratios of optimal online deterministic algorithms for the single VM migration and dynamic VM consolidation problems. Additionally, based on an analysis of historical data, they propose adaptive heuristics for dynamic consolidation of VMs. With simulation experiments using real-world workload traces the authors show the effectiveness of the proposed algorithms, as energy savings are gained and a high level of adherence to the SLAs is achieved.

3.4. Modeling and simulation of thermal processes

In a view of higher densities and scale of computing systems one of big challenges is to combine the simulation of power distribution with thermodynamic ones that gives the overview of the overall system energy-efficiency at the desired level of details.

In order to characterize the thermal distribution, there are several approaches that are differentiated by their accuracy and required model size. Fig. 2 gives the general overview [51].

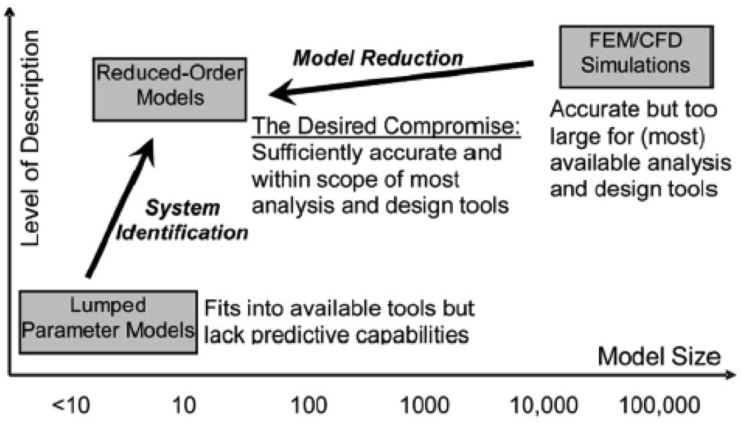


Figure 2. Existing approaches complexity [51]

For now, Computation Fluid Dynamics (CFD) simulations are considered as the most accurate approach. However, they require lots of effort, while preparing model and even more time to obtain rewarding results, which makes it expensive to use in terms of big systems simulations. As an alternative, Potential Flow Model (PFM) has been proposed [45], that benefits from the reduction of the model. Another approach follows proper orthogonal decomposition methodology [51] and corresponds to Reduced Order Models entity in fig. 2.

As the complexity of computing systems is increasing together with the growing importance of their energy efficiency, the processes of their evaluation has become difficult and complex to perform not only in real environments but also by the means of simulation tools. Thus, effort is being put to find a tradeoff between the simulation accuracy and required time complexity. As a results models based on the law of energy conservation and the basic heat transfer equation has been applied as the starting point of the work done by [62], [80], [76], where the heat recirculation idea was introduced and expanded into the concept of heat distribution matrix. Thermodynamics

models have also been introduced, together with the models for power consumption of cooling infrastructure for the whole data center in [70].

3.5. Energy-aware modeling and simulation tools

The growing importance of efficient computing systems and emergence of new computing paradigms caused gaining importance of modeling and simulation of various computing architectures and corresponding algorithms. In the recent years several simulation tools have been developed in order to address these issues. Among them, one should mention CloudSim [31], GreenCloud [55], BigSim [95] and DCworms [56]. CloudSim enables modeling and simulation of Cloud computing data centers and focuses on evaluating different approaches for provisioning host resources to virtual machines. It provides basic means to model power consumption and network traffic. GreenCloud pays more attention to networks aspects of cloud environments with a fine-grained modeling of the energy consumed by the elements of the data center, such as servers, switches, and links. However, a main focus is devoted to the packet-level simulation of communications in the data center networks. On the other hand, BigSim is a tool allowing simulation and performance prediction of machines with a very large number of processors. It provides the ability to evaluate the performance of specific applications on very large computer clusters. Finally, DCworms enables simulation of computing infrastructures to estimate their performance, energy consumption, and energy-efficiency metrics for diverse workloads and management policies. Compared to other tools, DCworms allows simulating a wide scope of physical and logical architectural patterns. In particular, it enables simulations of complex distributed architectures containing models of the whole data centers, containers, racks, nodes, etc. with a detailed energy and thermal modeling of each component. Moreover, DCworms provides means for modeling application performance both in HPC and cloud environments. These tools benefit from the models that are widely present in the literature. A direction of such studies span from network systems [89] through storage systems [13] up to servers systems [72]. Additionally, researchers considered also virtualized environments [68]. In [20] authors propose models that present data center power usage in a comprehensive way.

3.6. Summary

As a conclusion, energy efficiency and performance prediction of ultrascale systems is a difficult task that can greatly benefit from modeling and simulation techniques. However, full system simulation, as opposed to simulating individual system components, is still a difficult task. As stated in [61], full system simulation at an abstraction level that includes a sufficient level of detail is infeasible without resorting to parallel simulation. The main obstacles are the simulation execution time and the memory footprint. Therefore, by running parallel simulation models on large-scale systems, the performance of complex ultrascale systems can be predicted and enhanced. Although several approaches and tools have been developed, the emerging heterogeneous and low power hardware architectures as well as attempts to build ultrascale systems bring new challenges for simulations models. In particular, appropriate modeling of applications execution on heterogeneous computing nodes is needed. Simulations also should reflect energy consumption of all additional infrastructure needed by ultrascale systems. One of such essential element is cooling, which requires accurate (but sufficiently fast) simulation of thermal processes

in a computing center. The big challenge is also efficient simulation of large-scale systems, which may require simplified models and statistical/machine learning methods.

4. Resource management and scheduling in extra large scale systems

Research and development in large-scale systems over the last years had been mostly driven by performance, whereas rises in energy consumption were generally ignored. The result was a steadily growing in the performance, driven by more efficient system design and increasing density of the components according to Moore's law [37]. As the power wall was reached there was the need to increase performance by introducing parallel computing elements. Extra large scale systems will be characterized by the heterogeneity in hardware resources where a single node may be composed by a set of very different computing elements, such as a multicore CPU, manycore CPUs and GPUs, FPGAs, among others. The resource management for a system with such diversity of components needs to allow resource sharing at a finer level of granularity so that the performance per Watt of an active node is maximized. Future schedulers have to consider power limitations and energy usage optimization when making scheduling decisions [19], as power consumption is actually one of the main factors to achieve sustainability of extra large scale systems.

The concept of virtualization [86] has been successfully introduced in modern data centers, to achieve the necessary isolation among applications, and to increase resources usage rate. Virtualization was first introduced with the IBM mainframe systems in the 1960s, to refer to a virtual machine (VM) [10]. Although virtualization is a well developed technology it introduces additional overhead, that for scientific workload on an extra scale system may represent a significant loss of energy.

Resource sharing with the purpose of reducing the energy costs of a data center has been studied first for web loads [32], where workloads are of similar type. For scientific applications, the expected workloads are heterogeneous with different requirements in terms of computing power, storage, I/O and type of computing elements, thus imposing additional requirements on resource management and scheduling systems.

4.1. Workload characteristics

Scientific loads result from a variety of applications being the most common the following ones: a) bag-of-tasks (BoT), where each job is composed by a set of independent tasks that can run in any resource [82]; b) workflow applications, which represent many relevant real world problems [52], such as the *Montage* and *Epigenomics* workflows, among others. The first, created by NASA/IPAC, to stitch together multiple input images to create custom mosaics of the sky and, the second, is a workflow used for genome sequence processing; c) specific frameworks for data mining, such as the MapReduce model [33], and d) MPI jobs used in many processing intensive simulation problems. The diversity of workloads imposes challenges on resource management systems in order to deal simultaneously with such variety of applications.

4.2. Virtualization

A scheduler in a virtualized system has the purpose of deploying resources in a way to fulfill customer requests. More recently scheduling in virtualized systems has another goal, namely to deploy resources in order to minimize energy consumption.

Virtualization technology provides an additional infrastructure layer on top of which multiple VMs can be deployed. Virtualization technology can improve resource utilization, but it also consumes resources and thus creates an energy consumption overhead [59] - mostly through a hypervisor. As reported in [50], a hypervisor based on full virtualization, i.e., KVM , creates much higher overhead (11.6%) than one based on paravirtualization (0,47%), such as Xen, as opposed to using physical machines. Additionally, too big VM images are sources of additional losses, e.g. too large memory allocation and storage size.

While scheduling Cloud resources there are several causes of energy inefficiency [60]. For example rescheduling VMs every couple of minutes would perhaps give optimal deployment at the moment, however re-scheduling itself would probably consume more energy than it saves. Heavy VM migrations can lead to a performance overhead, as well as the energy overhead. While a performance loss can be avoided by using live migrations of VMs [58], the resulting energy overhead is often overlooked. When migrating a VM from one node to another, both nodes must be powered on until the migration is complete [69]. This includes both time and energy overheads for the migration, which is only rarely considered in the literature in the context of job placement.

Adaptation of more lightweight architectures for hypervisors, such as those based on micro-kernel [16], have still not take its full swing, and are mostly used in embedded systems, rather than Cloud Computing. However, most recently a technology based on Linux containers (e.g., Docker [85]) has shown some worthwhile benefits due to its lightweight design, fast deployment and low resource consumption footprint [39]. Furthermore, its performance is almost identical to bare-metal deployment as the applications running inside a container directly utilize hardware resources. On the one hand, due to its still relative immaturity the container technology offers a limited set of features, where adding more features could easily eliminate its benefits based on the lightweight design. On the other hand, hypervisor technology started with a heavyweight design, and due to optimization efforts it significantly increased its efficiency without losing its basic features.

4.3. Resource management and scheduling

Resource managers and schedulers for large systems are, in general, monolithic [38]. They are implemented as a single, centralized scheduling algorithm that controls the execution of all jobs. Examples are the widely used systems like PBS [67], Maui [48] and Moab [11] that are characterized by isolating the jobs in a static allocation of resources to each job. The centralized control becomes a scalability bottleneck for extra large systems and do not guarantee the simultaneous execution of different types of loads. To alleviate this control restraint, a two level control approach is applied that implements a high-level control scheduler, which offers sets of resources to other frameworks. An example is Mesos [36] that provides so-called resource offers to frameworks such as Hadoop/Yarn [41] and MPI jobs, which accept or reject these offers for scheduling their own jobs. The disadvantage of the two-level approach is that it leads to suboptimal resource usage as it isolates applications by assigning a specific set of resources to

each framework. Such approach represents a pessimistic sharing policy since it locks resources that can potentially be idle as they are taken by a frame- work that commonly exhibits uneven workload, which eventually results in a waste of energy. To improve the resource sharing among several frameworks and to overcome the centralized control constraint, Schwarzkopf et al. [38] proposed a flexible and scalable scheduler based on a shared-state approach. Full access to all nodes is granted to each framework that compete in a free-for-all manner, and uses an optimistic concurrency to resolve conflicts when updating the system state. In this approach all framework schedulers run in parallel, while having the constraint to observe the system state when committing the schedule decisions. If the scheduler fails a commit operation, it has to obtain a new schedule for its jobs.

Workflow applications represent a relevant class of scientific workloads and workflow scheduling has been addressed primarily for single workflow scheduling, i.e., a schedule is generated for a workflow and a specific number of processors, used exclusively throughout the workflow execution. When several workflows are submitted, they are considered as independent applications that are executed on independent subsets of processors. However, because of task precedence, not all processors are fully used when executing a workflow, thus leading to low efficiency. The efficient usage of any computing system depends on how well the workload is mapped to the processing units. One way to improve system efficiency is to consider concurrent workflows, i.e., sharing processors among applications, so that throughout its execution, the workflow can use any processor available in the system. Although the processors are not used exclusively by one workflow, only one task runs on a processor at any one time. Resource sharing among workflow applications with the aim of improving resource usage rate and dealing with the dynamic nature of a large system, where jobs are submitted at any time, have been studied in [14, 47, 93]. This approach may be used to specify a workflow scheduling framework to compete for resources in an extra large scale system.

The scheduling of BoT applications, also called, divisible load applications, have been extensively studied in a static and a dynamic approach. In [25] it was proposed a framework for running concurrent BoT applications in heterogeneous systems, where jobs are submitted and schedule dynamically without prior knowledge of the workload. Similarly with workflow concurrent execution, the aim is to reduce the maximum ratio between the processing time an application takes to complete concurrently with the time it would require to complete if executed alone.

From the previous discussion, we can see that there are frameworks available to manage the concurrent execution of the most typical scientific workloads, being a relevant middleware to be consider in order to obtain efficient resource management for extra large systems.

4.4. Summary

The diversity of workloads will require that a diversity of frameworks need to be considered in the same system, which interoperability may not exist or may not be desirable, to keep the system manageable. Additionally, with the large number of nodes that will constitute an extra large system, only a distributed concurrent deployment of sets of resources by framework schedulers, such as one proposed in [38], will have conditions to achieve scalability. However, this approach may not generate optimal energy efficient job scheduling as it only tackles an architectural design of the scheduler. Therefore, applying machine learning techniques to predict

workloads, such as [92] or the one introduced for Cloud environments in [12], may be developed to obtain long-term stability and improved energy eficient resource management.

5. Towards an Energy Efficient Design of Ultrascale Applications

To satisfy requirements and constraints in terms of computing and energy consumption, strong potential in energy savings can be achieved by the consolidation on physical servers of numerous virtualized services.

Moreover, Green 500 [5], the world contest of the most efficient HPC systems, shows that recent years trends offer real potential of energy saving and performances thanks to CPUs+GPUs heterogeneous systems. Nevertheless, both technologies are not yet fully compatible: the standardization of contexts of virtual machines does not accommodate the specific hardware of graphics accelerators and hybridization performance of these two technologies are still less than expected.

However, we think that the most efficient and easily deployed computing nodes can bring to ultrascale systems some energy solutions. We mainly focus our work on two points. Firstly, by kick up bottlenecks of interoperability between virtualized contexts and hardware accelerators and secondly, in boosting a widely used component that can easily optimize a number of current applications.

In the first section, we present the trends of the most efficient hardware architectures in a scientific context. In the second section we present two software mechanisms to reduce latencies of data transmissions between virtual machines and so provide unleashed access to graphics accelerators from virtualized applications.

Beyond scientific applications, where GPUs are widely appreciated, we will show that usage of accelerators can also bring strong potential performance and energy cutback. With this point of view, the third section focuses on a parallelized implementation of a DBMS engine, a fundamental computing component of many applications, from small embedded devices to Big Data applications.

The last section is finally interested in new perspectives for the use of SoC (system on Chip - SoC) and their potential in both computing performance and power consumption.

5.1. Green 500 and Hybrid Architectures

From several years, Green 500 top list, devoted for the most efficient supercomputing systems, shows that a trend is mainly driven by heterogeneous architectures, combining multi-core CPU and GPU (see tab. 2). The system performance must be at least as high as the 500th of the Top List of the world fastest computer [6].

Computing characteristics of current Green500 top machines indicate that most of them are build upon Intel IvyBridge CPU and Nvidia or AMD GPU. A new Japanese competitor, PEZY-SC [9] stands up at the second position with a proprietary accelerator embedding 1024 cores. It claims peak a performance of 50GFlops/W in single precision.

Globally, systems in Green500 show a predominance of Intel CPU (429), AMD (29) and IBM (38) high end processor. For accelerator, Nvidia GPU is also predominant (50), before Intel Xeon Phi (20), followed by AMD GPU (4). The couple Intel CPU and Nvidia GPU is

Table 2. Top 10 of Green500, November 2014

Gflops/ Watt	Year	Site	Manufacturer	Processor	Accelerator / Co-Processor
5,27	2014	GSI Helmholtz Center, Germany	AMD, ASUS, FIAS, GSI	Xeon E5-2690v2 10C 3GHz	AMD FirePro S9150
4,95	2014	High Energy Accelerator Research Organization / KEK, Japan	PEZY Computing / Exascaler Inc.	Xeon E5-2660v2 10C 2.2GHz	PEZY-SC
4,45	2013	GSIC Center, Tokyo Institute of Technology, Japan	NEC	Xeon E5-2620v2 6C 2.1GHz	Nvidia K20x
3,97	2014	Cray Inc., United States	Cray Inc.	Xeon E5-2660v2 10C 2.2GHz	Nvidia K40m
3,63	2013	Cambridge University, United Kingdom	Dell	Xeon E5-2630v2 6C 2.6GHz	Nvidia K20
3,54	2013	Financial Institution, United States	IBM	Xeon E5-2680v2 10C 2.8GHz	Nvidia K20x
3,52	2013	Center for Computational Sciences, University of Tsukuba	Cray Inc.	Xeon E5-2680v2 10C 2.8GHz	Nvidia K20x
3,46	2014	SURFsara	Bull SA	Xeon E5-2450v2 8C 2.5GHz	Nvidia K40m
3,19	2012	Swiss National Supercomputing Centre (CSCS)	Cray Inc.	Xeon E5-2670 8C 2.6GHz	Nvidia K20x
3,13	2013	ROMEO HPC Center - Champagne-Ardenne	Bull SA	Xeon E5-2650v2 8C 2.6GHz	Nvidia K20x

largely dominant in top 10. GPU is now well established as an efficient accelerator, either on computing performance or energy consumption.

Green 500 and Top 500 are mainly relevant in computation-intensive domain but these performance criteria could not be sufficiently suitable for many real world applications. [79] focused on an alternative efficiency benchmarking for large-scale graph algorithms class, more relevant to data-intensive problems.

The problem here is to minimize energy resources dedicated to large graph exploration. The authors of this work propose two reference parallel kernels (replicated-csr and replicated-csc) and different scales of free graph problems (from 220 up to 242 numbers of vertices). Benchmark uses the elapsed times for both kernels, but the rankings for Graph 500 [4] are determined by problem size and the throughput in numbers of edges traversed per second, TEPS (Traversed Edges Per Second).

Analysis of Graph 500 list [7] shows that best efficiency, 445 GTEPJ (109 Traversed Nodes Per Joule) was reached on a small scale graph problems (220 vertices) solved on hybrid machine, with Intel CPU and NVIDIA K20 GPU. The second competitor score, 230 GTEPJ, was reached by an Android Smartphone. On large scale problem (238 and 241 vertices), only two competitors provide machine power consumption in order to estimate energy efficiency. Those are BlueGene/Q (Power BQC 16C, 1.6GHz) with only 5.40 and 3.71 METPJ (106 Traversed Nodes Per Joule).

5.2. Sharing Hardware Accelerators Between Virtual Machines

Accelerators like GPU or MIC devices can dramatically improve computation performance. Therefore, it is interesting to exploit them in virtualized contexts, either with "on the shelf" libraries (e.g. cuFFT, cuSparse) or through custom code (OpenCL, CUDA). However, these accelerators are considered as specific hardware and do not match legacy requirements to enable

virtualized drivers. Nvidia Grid [8] announces a network GPU grid allowing several users to share GPU resources, but not in the framework of usual virtualization.

Nevertheless, there exist some tools that can be used to access and share GPUs in virtual environments. GVirtuS [21] is one of them. It acts as a bridge between the unique privileged virtual machine (VM) and unprivileged VM. To use hardware accelerator from one virtualized context, an application has to transfer operation requests and their data to the privileged VM, which has direct hardware access to accelerators. When the device terminates its job, results have to be returned back to unprivileged VM. Data transfer between VM is performed thanks to network links (TCP/UDP stack) which are implemented as ring buffers. This mechanism allows an overlapping of write and read operation; however, it leads to four data copies. This weakness can break down the performance in terms of time, energy and memory usage, although each copy of data is positioned in the unique and physical system memory.

In order to overcome it and boost performance, we developed two original techniques to optimize data transfers between a privileged and an unprivileged VMs, hosted on the same physical machine using Xen virtualization solution.

The first transfer mechanism transfers data already located in memory. Instead of copying data between virtual machines, we grant the privileged VM to access data pages of an unprivileged VM. In order to achieve this, we developed a kernel module, GNTADDR, which retrieves page identifiers of target memory zones. Identifiers are then transferred to the privileged VM which uses them to map pages and then access data. With a 4096 byte page size and 8 bytes per identifier, the transferred data volume is reduced by 512. The accelerator device managed by the privileged VM has therefore a direct access to data produced in an unprivileged VM without any expensive duplication. Shared pages are always owned by the unprivileged VM and still exist after the transfer. However, this mechanism cannot avoid the four duplications inherent of a ring buffer pages when identifiers need to be transferred. That is why we developed a second scheme.

The second scheme improves transfer of "short time lived" data. In order to avoid any duplication, we designed a ring buffer instantiated inside a shared memory block, named GNTRING. Shared memory spaces are managed (allocation, mapping, sharing) and owned by the ring buffer, but data can be placed and pulled out by processes either in privileged or unprivileged VM. Here again, the VM has a direct access to data hosted in the ring buffer without any duplication. An asynchronous mode is also implemented in order to be more flexible. This mode saves some internal signal interactions and is therefore more efficient.

With these two original mechanisms, data transfers between VM machines and hardware devices can be performed with a reduced delay. Performances of these tools are depicted at fig. 3. We compared our tools with the native Xen TCP socket transfer mechanism and the ring buffer implementation of XenSocket [94].

GNTRING is used to transfer page identifiers retrieved by GNTADDR. The resulting tool is named RINGADDR. These tools were evaluated with raw ping-pong data transfers, sending datasets of different sizes and waiting for the acknowledgment signal. For each size, we ran hundred transfers and we computed averages. Ring buffers have a size of 1 MB (256 pages). These tests were conducted on a hexacore CPU (AMD Phenom II X6 1090T 3.2 GHz) with 8 GB of RAM. We used Xen v4.3 hypervisor. The virtual machine has 2 GB of RAM and two CPU cores. Ubuntu v3.2 is the operating system for all domains.

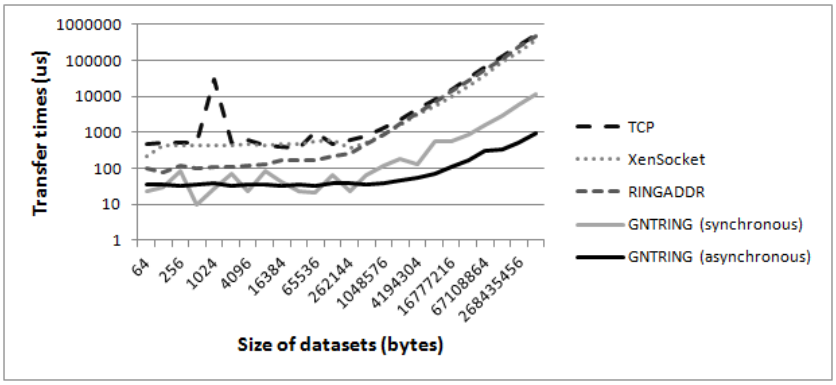


Figure 3. Performance comparison of Xen transfer mechanisms

We consider TCP transfer mechanism as our reference as it is the standard communication mechanism between machines. Firstly, we observe that XenSocket is more efficient than TCP, thanks to its internal ring buffer which overlaps read and write operations. Secondly, we see that RINGADDR is more efficient than XenSocket for data size smaller than 500 MB, with speedups between 2 and 5. Above 500 MB, performances are similar. Finally, we notice that both modes of GNTRING yield the best performances. Compared to XenSocket, speedups are between 5 and 30 in synchronous mode and between 6 and 390 in asynchronous mode. Peak measured bandwidth is about 500GB/s, and latency becomes quite transparent for data exchange between virtualized domains.

For future works, these tools will be extended to inter VM communications to perform more efficient data transfers without requiring hardware, e.g. MPI jobs or web services. Also, those tools will be integrated inside GVirtuS in order to improve hardware accelerator handling.

5.3. Low Power SoC

New embedded platforms, where CPU and GPU are shipped together on the same die (SoC) and share the same memory space, offer two main new perspectives. The first one is a capability to only transfer data owner grant between CPU and GPU, avoiding costly data duplication. The second one is to perform the same amount of workload on a low-power platform, expecting comparable performances with much less energy: new TegraK1 SoC's platform, which have equivalent computation capability (4 ARM cores + 192 GPU cores), but needs only 11W compared to test-platform (50W for CPU + 188W for GPU). In embedded systems (smartphones, tablets), the low power processors market is mainly dominated by ARM SoC. ARM architecture is RISC, so executable code is little bigger (20%) but core hardware architecture is less complex than x86 counterparts. Globally ARM cores need less energy than x86 core equivalents. ARM Cortex CPUs, and MALI GPU companion, are distributed as system on chip (SoC) design in order to be implemented by external founders but they are free to mix different kind of CPU and GPU cores on chips.

SoC technologies enable CPU and GPU but also memory and network blocks to be closely coupled on one chip. This technology could throw away two main hardware bottlenecks:

- Memory space becomes shared between CPUs and GPU. This improvement can eliminate time consuming data duplication between processor and accelerator.

- Hardware interfaces (PCI/PCIe) are replaced by direct links (routed lines), more efficient and less power consuming than external chipset (north or south bridge).

Anyway, this technology imposes some limitation, essentially about the lack of flexibility:

- Memory space is not expandable: current accelerators embed no more than 12 GB. This drawback could impose rewrite or redesign algorithms in a distributed approach.
- Lack of I/O: HDD, networking, etc.

Nvidia has developed in 2013 such technology on its TegraK1 processor which joins together four 32 bit CPU cores or two 64bit CPU cores with 192 Kepler GPU cores.GPU architecture in Tegra K1 is virtually identical to the Kepler GPU architecture used in high-end systems (Tesla K20), but also includes a number of optimizations for mobile system usage to preserve power and deliver industry-leading mobile GPU performance. While the highest-end Kepler GPUs in desktop, workstation, and supercomputers include up to 2880 single-precision floating point CUDA cores and consume around 200 W, the Kepler GPU in Tegra K1 consists of 192 CUDA, cores and consumes a couple of watts. Its peak performance is announced around 300 GFLOPS at 6 W total power draw [1] or 50 GFLOPS/W. This is over 10 times more power-efficient than the world most power-efficient oil-cooled supercomputer. This type of architecture has more cores than many entry-level to mainstream desktop GPUs of just few years ago.

At the end of 2014, Nvidia announces the next generation, including four 64 bit, four 32 bit CPU cores and 256 Maxwell GPU cores for its new TegraX1. Performance grows up to 1000/500 GFlops on FP16/FP32 with 2 times less energy.

5.4. Discussion

According to [55], we estimate that processing servers represent around 70% of total data-center energy consumption, while connection links and switches account for 30%. It means that efforts to reduce energy consumption must focus on computing servers and the same trend should be followed for data-center servers. Cooling is also a major energy consumer, but is not taken into account here. Its impact can be considered as a ratio penalty, linearly correlated with power of computing and networking components.

Green 500 and Green Graph 500 are mainly relevant in the domain of scientific computing, but what about most of data centers? One fact is that GPU accelerators can be now considered as valuable accelerators but they still need a larger adoption, especially in industrial and business applications. One of the most common usages of data centers is hosting enterprise information (ERP, CRM, mail or web servers, cloud, etc.) and commercial applications (e-business, finance and telecoms). Virtualization and services consolidation in these data centers become a common trend to reduce energy consumption by focusing workloads on fewer physical servers often save 40 to 80% [75]. In this matter, we point the lack of a solution which combines advantages of virtualized architectures and hardware accelerators.

GPUs benefit from much more computation power at the same order of energy consumption than classical CPUs. Except in some application fields like video games, graphical editing or HPC, GPUs are currently under exploited. There is indeed a considerable amount of scientific publications about exploitation of GPUs, either in computing power and in energy efficiency, for scientific simulations. Works are however relatively limited to other fields of applications, like those of data centers.

In the context of exascale applications, and specially in BigData or IoT elastics applications, virtualization flexibility is a major advantage for dynamically redeploying resources according

to user needs. Bringing this flexibility level to GPU accelerator could combine the advantages of power and energy efficiency to that kind of applications.

Conclusions

This article reports the various activities explored in the working group "Energy Efficiency" from the Nesus European COST IC1305 action. The identified trends and challenges studied by the group concentrate on the use of efficient hardware, especially exploiting heterogenity and low power chips, effective and lightweight monitoring for large scale systems, enabling analysis of future ultrascale systems by modeling and simulation, including detailed models of workloads as well as thermal and cooling aspects, specific resource management approaches, and proper design of applications with power and efficiency in mind.

Based on this analysis some specific observations were made that that led to identification of challenges to be studied further by the energy-efficiency group within the Nesus action.

First of all, the heterogeneity in computing resources is a necessity for ultrascale systems to ensure both a flexible adaptation to HPC workloads and significant power-savings. Yet taking advantage of these heterogeneous resources assumes the possibility of measuring with a reasonably good accuracy the performance and the energy-efficiency of the system. Another challenge apart from the energy-efficiency itself is the ease of programming, which can be a blocker. Hence, an important point to explore is a trade-off between energy-efficiency gains and programming effort (including a selection of a hardware platform, application design and optimization).

The challenge related to monitoring at ultrascale level, to make it manageable, is to be able to combine both fine-grained measurements for short time periods based on software interfaces, and aggregated metrics based on physical wattmeters for the higher views over longer time ranges.

As ultrascale systems are future goal rather than commonly available, energy efficiency and performance prediction of ultrascale systems is a difficult task that can greatly benefit from modeling and simulation techniques. However, full system simulation, as opposed to simulating individual system components, is still a difficult task. The main obstacles are the simulation execution time and the memory footprint. The possible solutions to this problem to be studied include running parallel simulation models on large-scale systems as well as simplified models and statistical/machine learning methods. Another challenges needed to tackle to obtain accurate full system simulation are appropriate modeling of applications execution on heterogeneous and low power hardware architectures, and simulation of cooling, which requires accurate (but sufficiently fast) simulation of thermal processes in a computing center.

Similarly to modeling and simulation the scheduling and resource management techniques will be constrained in their accuracy and scalability by a complexity and size of ultrascale systems. Thus, they will have to take advantage of distributed concurrent allocation of sets of resources and the use of machine learning techniques to enable improvements in energy efficiency. The important challenge will be to deal with a variety of workloads and adaptation of scheduling and resource management methods to their specifics.

Finally, the crucial aspect in obtaining energy-efficiency will be appropriate application design. To achieve it applications will have to take advantage of new heterogeneous and low power architectures. The use of these architectures will increase needs for interdisciplinary optimizations such as software-hardware co-design or exploiting detailed models of application by schedulers managing heterogeneous resources.

This work was supported by the Spanish Ministry of Education and Science through the TIN2011-24598 project, Spanish CAPAP-H4 network, by a grant from Polish National Science Center under award number 2013/08/A/ST6/00296, and NESUS IC1305 COST Action.

References

1. Embedded tegra & jetson tk1 blog. `https://plus.google.com/114318922342198493952/posts`.
2. Energy-conscious 3D Server-on-Chip for Green Cloud. `http://www.eurocloudserver.com/`.
3. European Mont-Blanc Project. `http://www.montblanc-project.eu/`.
4. Graph 500. `http://www.graph500.org`.
5. The green500 list - november 2014. `http://www.green500.org/news/green500-list-november-2014`.
6. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. `http://www.netlib.org/benchmark/hpl/`.
7. November 2014 — graph 500. `http://www.graph500.org/results_nov_2014`.
8. Nvidia grid - gprahics accelerated virtual desktops and applications. `http://www.nvidia.co.uk/object/grid-vdi-desktop-virtualisation-uk.html`.
9. Pezy-sc many core processor(2014). `pezy.co.jp/en/products/pezy-sc.html`.
10. R. J. Adair. *A virtual machine system for the 360/40*. International Business Machines Corporation, Cambridge Scientific Center, 1966.
11. Adaptive Computing. Moab workload manager administrator's guide, version 8.0.0. `http://docs.adaptivecomputing.com`, September 2014.
12. Samuel A. Ajila and Akindele A. Bankole. Cloud client prediction models using machine learning techniques. In *37th Annual IEEE Computer Software and Applications Conference, COMPSAC 2013, Kyoto, Japan, July 22-26, 2013*, pages 134–142, 2013. DOI: 10.1109/COMPSAC.2013.21.
13. Miriam Allalouf, Yuriy Arbitman, Michael Factor, Ronen I. Kat, Kalman Meth, and Dalit Naor. Storage modeling for power estimation. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 3:1–3:10, New York, NY, USA, 2009. ACM.
14. Hamid Arabnejad and J.G. Barbosa. Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 633—-639. IEEE, 2012. DOI: 10.1109/ispa.2012.94.
15. ARM Limited. ARM Energy Probe. `http://ds.arm.com/ds-5/optimize/arm-energy-probe/`.

16. F. Armand and M. Gien. A practical look at micro-kernels and virtual machine monitors. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–7, 2009. DOI: 10.1109/CCNC.2009.4784874.

17. Michael Armbrust and al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

18. Sergio Barrachina, Maria Barreda, Sandra Catalán, Manuel F. Dolz, Germán Fabregat, Rafael Mayo, and Enrique S. Quintana-Ortí. An integrated framework for power-performance analysis of parallel scientific workloads. *Energy 2013 : the third international conference on smart grids, green communications and it energy-aware technologies*, pages 114–119, mar 2013.

19. Luiz A. Barroso, Jimmy Clidaras, and Urs Holzle. *The Datacenter as a Computer*. Morgan and Claypool Publishers, 2nd edition edition, 2013. DOI: 10.2200/s00516ed2v01y201306cac024.

20. Robert Basmadjian, Nasir Ali, Florian Niedermeier, Hermann de Meer, and Giovanni Giuliani. A methodology to predict the power consumption of servers in data centres. In *Proceedings of the 2Nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11, pages 1–10, New York, NY, USA, 2011. ACM.

21. Michela Becchi, Kittisak Sajjapongse, Ian Graves, Adam Procter, Vignesh Ravi, and Srimat Chakradhar. A virtual memory based runtime to support multi-tenancy in clusters with gpus. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2012. ACM. DOI: 10.1145/2287076.2287090.

22. D. Bedard, Min Yeol Lim, R. Fowler, and A. Porterfield. Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pages 479–484, March 2010. DOI: 10.1109/SECON.2010.5453824.

23. Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012. DOI: 10.1002/cpe.1867.

24. Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *CoRR*, abs/1007.0066, 2010.

25. A. Benoit, L. Marchal, J.F. Pineau, Y. Robert, and F. Vivien. Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. *IEEE Transactions on Computers*, 59(2):202–217, 2010. DOI: 10.1109/tc.2009.117.

26. Kashif Bilal, Ahmad Fayyaz, Samee U Khan, and Saeeda Usman. Power-aware resource allocation in computer clusters using dynamic threshold voltage scaling and dynamic voltage scaling: comparison and analysis. *Cluster Computing*, pages 1–24, 2015.

27. Aurelien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, 2013(92), 2013.

28. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, August 2000.

29. A. Cabrera, F. Almeida, J. Arteaga, and V. Blanco. Energy Measurement Library (EML). `https://github.com/HPC-ULL/eml/`.

30. Alberto Cabrera, Francisco Almeida, and Vicente Blanco. Eml, an energy measurement library. In *31st International Symposium on Computer Performance, Modeling, Measurements and Evaluation*, 2013. Student Poster Abstracts.

31. Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, January 2011. DOI: 10.1002/spe.995.

32. J. Chase and R. Doyle. Balance of power: Energy management for server clusters. In *Workshop on Hot Topics in Operating Systems*, 2001. DOI: 10.1109/HOTOS.2001.990081.

33. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications ACM*, 51(1):107–113, 2008. DOI: 10.1145/1327452.1327492.

34. V. Delplace, P. Manneback, F. Pinel, S. Varrette, and P. Bouvry. Comparing the Performance and Power Usage of GPU and ARM Clusters for Map-Reduce. In *Proc. of the 3rd Intl. Conf. on Cloud and Green Computing (CGC'13)*, pages 199–200. IEEE Computer Society, Oct. 2013. DOI: 10.1109/cgc.2013.38.

35. Marcos Dias De Assuncao, Jean-Patrick Gelas, Laurent Lefèvre, and Anne-Cécile Orgerie. The Green Grid5000: Instrumenting a Grid with Energy Sensors. In Springer, editor, *INGRID'2010 : 5th International Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid*, pages 25–42, Poznan, Poland, May 2010. Springer.

36. B. Hindman et al. Mesos: A platform for fine-grained resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, Boston, MA, March 2011. USENIX Association.

37. G. E. Moore et al. Cramming more components onto integrated circuits. In *IEEE 86*, volume 1, pages 82–85, 1988. DOI: 10.1109/jproc.1998.658762.

38. M. Schwarzkopf et al. Omega: flexible, scalable schedulers for large compute clusters. In *EuroSys*, 2013. DOI: 10.1145/2465351.2465386.

39. Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. Technical report, IBM Research, 2014.

40. Jason Flinn and Mahadev Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA*, pages 2–10. IEEE Computer Society, 1999. DOI: 10.1109/mcsa.1999.749272.

41. A.S. Foundation. Apache hadoop nextgen mapreduce (yarn). `http://hadoop.apache.org/`, 2015.

42. Ganglia. Ganglia Monitoring System webpage.

43. Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, 2010. DOI: 10.1109/tpds.2009.76.

44. M. Guzek, S. Varrette, V. Plugaru, J. E. Sanchez, and P. Bouvry. A Holistic Model of the Performance and the Energy-Efficiency of Hypervisors in an HPC Environment. In *Proc. of the Intl. Conf. on Energy Efficiency in Large Scale Distributed Systems (EE-LSDS'13)*, volume 8046 of *LNCS*, pages 133–152, Vienna, Austria, Apr 2013. Springer Verlag.

45. C. M. Healey, Sheffer Z. R. VanGilder, J. W., and X. S. Zhang. Potential-flow modeling for data center applications. In *Proceedings of the ASME 2011 Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems*, volume 2, pages 527–534. ASME, 2011. DOI: 10.1115/IPACK2011-52136.

46. K.R. Hoffman and P. Hedge. ARM Cortex-A8 vs. Intel Atom: Architectural and Benchmark Comparisons. Technical report, University of Texas at Dallas, 2009.

47. C. Hsu, K. Huang, and F Wang. Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems*, 27(6):860–870, 2011. DOI: 10.1016/j.future.2010.10.015.

48. D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In D.G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *LNCS*, pages 87–102, 2001. DOI: 10.1007/3-540-45540-x_6.

49. M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry. Performance Evaluation and Energy Efficiency of High-Density HPC Platforms Based on Intel, AMD and ARM Processors. In *Proc. of the Intl. Conf. on Energy Efficiency in Large Scale Distributed Systems (EE-LSDS'13)*, volume 8046 of *LNCS*, pages 182–200, Vienna, Austria, Apr 2013. Springer Verlag. DOI: 10.1007/978-3-642-40517-4_16.

50. Yichao Jin, Yonggang Wen, and Qinghua Chen. Energy efficiency and server virtualization in data centers: An empirical investigation. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 133–138. IEEE, 2012. DOI: 10.1109/infcomw.2012.6193474.

51. Y. Joshi. Reduced order thermal models of multi-scale microsystems. In *Proceedings of the 14th International Heat Transfer Conference*, volume 8, pages 519–536. ASME, 2010. DOI: 10.1115/IHTC14-23373.

52. Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29:682–692, 2013. DOI: 10.1016/j.future.2012.08.015.

53. kernel.org. Perf Wiki. `https://perf.wiki.kernel.org/index.php?title=Main_Page&oldid=3491`, 2014.

54. Kyong Hoon Kim, Wan Yeon Lee, KIM Jong, and Rajkumar Buyya. Sla-based scheduling of bag-of-tasks applications on power-aware cluster systems. *IEICE TRANSACTIONS on Information and Systems*, 93(12):3194–3201, 2010. DOI: 10.1587/transinf.e93.d.3194.

55. D. Kliazovich, P. Bouvry, Y. Audzevich, and S.U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5, Dec 2010. DOI: 10.1109/GLOCOM.2010.5683561.

56. Krzysztof Kurowski, Ariel Oleksiak, Wojciech Piatek, Tomasz Piontek, Andrzej W. Przybyszewski, and Jan Weglarz. Dcworms - a tool for simulation of energy efficiency in dis-

tributed computing infrastructures. *Simulation Modelling Practice and Theory*, 39:135–151, 2013. DOI: 10.1016/j.simpat.2013.08.007.

57. James H. Laros, David DeBonis, and Phi Pokorny. *PowerInsight - A Commodity Power Measurement Capability.* April 2013. DOI: 10.1109/igcc.2013.6604485.

58. Haikun Liu, Hai Jin, Cheng-Zhong Xu, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. *Cluster computing*, 16(2):249–264, 2013. DOI: 10.1007/s10586-011-0194-3.

59. Toni Mastelic and Ivona Brandic. Timecap: Methodology for comparing it infrastructures based on time and capacity metrics. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 131–138. IEEE, 2013. DOI: 10.1109/cloud.2013.130.

60. Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Comput. Surv.*, 47(2):33:1–33:36, December 2014. DOI: 10.1145/2656204.

61. Cyriel Minkenberg, Wolfgang Denzel, German Rodriguez, and Robert Birke. End-to-end modeling and simulation of high-performance computing systems. In *Use Cases of Discrete Event Simulation*, pages 201–240. Springer, 2012. DOI: 10.1007/978-3-642-28777-0_11.

62. Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

63. Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014. DOI: 10.1145/2532637.

64. Z. Ou, B. Pang, Y. Deng, J.K. Nurminen, A. Ylä-Jääski, and P. Hui. Energy- and Cost-Efficiency Analysis of ARM-Based Clusters. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012*, pages 115–123, 2012. DOI: 10.1109/ccgrid.2012.84.

65. P3 International. Kill A Watt product page. `http://www.p3international.com/products/p4400.html`.

66. E. L. Padoin, D.A.G. de Oliveira, P. Velho, and P.O.A. Navaux. Time-to-Solution and Energy-to-Solution: A Comparison between ARM and Xeon. In *Third Workshop on Applications for Multi-Core Architectures (WAMCA)*, pages 48–53, 2012. DOI: 10.1109/wamca.2012.10.

67. PBS Works. Pbs professional 12.2, user's guide. `http://www.pbsworks.com`, September 2014.

68. M. Pedram and Inkwon Hwang. Power and performance modeling in a virtualized server system. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 520–526, Sept 2010. DOI: 10.1109/ICPPW.2010.76.

69. Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 225–233. ACM, 2010. DOI: 10.1145/1791314.1791350.

70. W. Piatek, A. Oleksiak, and G. Da Costa. Energy and thermal models for simulation of workload and resource management in computing systems. *Simulation Modelling Practice and Theory*, 2015. DOI: 10.1016/j.simpat.2015.04.008.

71. Jean-Marc Pierson, Georges Da Costa, and Lars Dittmann, editors. *Energy Efficiency in Large Scale Distributed Systems - COST IC0804 European Conference, EE-LSDS 2013, Vienna, Austria, April 22-24, 2013, Revised Selected Papers*, volume 8046 of *Lecture Notes in Computer Science*. Springer, 2013.

72. Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

73. Sebastian Ryffel. Lea^{2}p: The linux energy attribution and accounting platform. Master's thesis, Swiss Federal Institute of Technology, 2009.

74. Sandia National Laboratories. High performance computing power application programming interface (api) specification. `http://powerapi.sandia.gov/`, 2014.

75. Bernd Schäppi, Thomas Bogner, and Hellmut Teschner. Efficacité énergétique des technologies et infrastructures dans les datacentres et salles serveurs. Wien, Austria, 2011. Prime Energy IT.

76. Jayantha Siriwardana, Saman K. Halgamuge, Thomas Scherer, and Wolfgang Schott. Minimizing the thermal impact of computing equipment upgrades in data centers. *Energy and Buildings*, 50(0):81 – 92, 2012.

77. Thanos Stathopoulos, Dustin McIntire, and William J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *IPSN*, pages 383–394. IEEE Computer Society, 2008. DOI: 10.1109/IPSN.2008.36.

78. Georgios L Stavrinides and Helen D Karatza. The impact of resource heterogeneity on the timeliness of hard real-time complex jobs. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, page 65. ACM, 2014. DOI: 10.1145/2674396.2674469.

79. T. Suzumura, K. Ueno, H. Sato, K. Fujisawa, and S. Matsuoka. Performance characteristics of graph500 on large-scale distributed environment. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 149–158, Nov 2011. DOI: 10.1109/IISWC.2011.6114175.

80. Q. Tang, S.K.S. Gupta, D. Stanzione, and P. Cayton. Thermal-aware task scheduling to minimize energy usage of blade server based datacenters. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 195–202, Sept 2006. DOI: 10.1109/DASC.2006.47.

81. George Terzopoulos and Helen Karatza. Performance evaluation and energy consumption of a real-time heterogeneous grid system using dvs and dpm. *Simulation Modelling Practice and Theory*, 36:33–43, 2013. DOI: 10.1016/j.simpat.2013.04.006.

82. George Terzopoulos and Helen Karatza. Bag-of-task scheduling on power-aware clusters using a dvfs-based mechanism. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 833–840. IEEE, 2014. DOI: 10.1109/ipdpsw.2014.95.

83. George Terzopoulos and Helen Karatza. Energy-efficient real-time heterogeneous cluster scheduling with node replacement due to failures. *The Journal of Supercomputing*, 68(2):867–889, 2014. DOI: 10.1007/s11227-013-1070-0.

84. J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010. DOI: 10.1109/icppw.2010.38.

85. James Turnbull. *The Docker book*. Number v1.3.1. October 2014.

86. G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. L. Scott. System-level virtualization for high performance computing. In *16th Euromicro Conference on Distributed and Network-Based Processing*, pages 636–643, 2008. DOI: 10.1109/pdp.2008.85.

87. Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008. DOI: 10.1145/1496091.1496100.

88. S. Varrette, M. Guzek, V. Plugaru, X. Besseron, and P. Bouvry. HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors. In *Proc. of the 25th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2013)*, pages 89–96, Porto de Galinhas, Brazil, Oct. 2013. IEEE Computer Society. DOI: 10.1109/sbac-pad.2013.18.

89. A. Vishwanath Member, K. Hinton, R.W.A. Ayre, and R.S. Tucker. Modeling energy consumption in high-capacity routers and switches. *Selected Areas in Communications, IEEE Journal on*, 32(8):1524–1532, Aug 2014. DOI: 10.1109/JSAC.2014.2335312.

90. Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009. DOI: 10.1109/clustr.2009.5289182.

91. Watt's Up Meters. Watt's Up product page. `https://www.wattsupmeters.com/`.

92. Richard Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003. DOI: 10.1145/773056.773064.

93. Z. Yu and W. Shi. A planner-guided scheduling strategy for multiple workflow applications. In *Proceedings of the International Conference on Parallel Processing-Workshops (ICPP-W'08)*, pages 1–8. IEEE, 2008. DOI: 10.1109/icpp-w.2008.10.

94. Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi, and John Linwood Griffin. Xensocket: A high-throughput interdomain transport for virtual machines. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

95. Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kalé. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *In18th Intl.Paralleland Distr.Proc. Symp. IPDPS*, page 78, 2004.

Received March 3, 2015.