DOI: 10.14529/jsfi150102

Applications for Ultrascale Computing

***Lars Ailo Bongo*[1], *Raimondas Čiegis*[2], *Neki Frasheri*[3], *Jing Gong*[4], *Dragi Kimovski*[5], *Peter Kropf*[6], *Svetozar Margenov*[7], *Milan Mihajlović*[8], *Maya Neytcheva* [9], *Thomas Rauber*[10], *Gudula Rünger*[11], *Roman Trobec*[12], *Roel Wuyts*[13], *Roman Wyrzykowski*[14]**

Studies of real complex physical and engineering problems represented by multiscale and multiphysics computer simulations have an increasing demand for computing power. The demand is driven by the increasing scales and complexity of the scientific problems investigated or the time constraints. Ultrascale computing systems could offer the computing power required to solve these problems. Future ultrascale systems will be large-scale complex computing systems combining technologies from high performance computing, distributed systems, big data, and cloud computing. The challenge of developing and programming complex algorithms that can efficiently perform on such systems is twofold. Firstly, the complex computer simulations have to be either developed from scratch, or redesigned in order to yield high performance, while retaining correct functional behaviour. Secondly, ultrascale computing systems impose a number of non-functional cross-cutting concerns, such as fault tolerance or energy consumption, which can significantly impact the deployment of applications on large complex computing systems. This article discusses the state-of-the-art of programming for current and future large-scale computing systems with an emphasis on complex applications. We derive a number of requirements regarding programming and execution support by studying several computationally demanding applications that the authors are currently developing and discuss their potential and necessary upgrades for ultrascale execution on ultrascale facilities.

Keywords: sustainable ultrascale systems, impact factors on applications, multiscale and multiphysics applications, computational modelling.

Introduction

A glance over the historical development of computational science shows that software and hardware developments have always been driven by the need for a continual growth. For software this is a continuously increasing growth in complexity of algorithms, of data sizes and processing requirements; for hardware these have been and are the technological inventions providing increasing computing power and storage capabilities. Topics such as High Performance Computing (HPC), distributed systems, big data and cloud computing are well-established domains of software and hardware development reflecting this tendency. In the near future, the growth in algorithmic complexity, data volumes to be processed, and available computing power is expected

[1]University of Tromsø, Tromsø, Norway
[2]Vilnius Gediminas Technical University, Lithuania
[3]University of Tirana, Albania
[4]Royal University of Technology, Stockholm, Sweden
[5]University for Information Science and Technology "St. Paul the Apostle", Ohrid, Macedonia
[6]University of Neuchâtel, Switzerland
[7]Institute of Information and Communication Technologies, Sofia, Bulgaria
[8]The University of Manchester, UK
[9]Uppsala University, Sweden
[10]University Bayreuth, Germany
[11]Technical University Chemnitz, Germany
[12]Jožef Stefan Institute, Ljubljana, Slovenia
[13]imec, Leuven, Belgium and KU Leuven, Belgium
[14]Czestochowa University of Technology, Poland

to reach extreme scales. Successful handling of this growth and maintaining performance on such scales requires the existing and the emerging hardware and software aspects and concerns to be re-evaluated and adapted to the new paradigms. The generic term ultrascale computing captures all these efforts and challenges.

Ultrascale computing systems are expected to have the form of large-scale complex systems comprising parallel and distributed components as well as heterogeneous processors, e.g. enhanced by Graphical Processing Units (GPU), Field Programmable Gate Arrays (FPGA) or other types of accelerators. These compute facilities might be provided by cloud architectures that are made available to scientific computing communities in an effort to achieve scientific cloud computing [85]. Large-scale scientific simulations often have to deal with large volumes of data which may come from multiple sources and are diverse, complex, and have massive scale, requiring the big data techniques to be included. Due to the complexity of ultrascale systems, their efficient usage is a challenging task which exceeds the effort for programming and maintaining the HPC systems that are available today. Consequently, adequate support for developing software on different levels is needed to properly exploit the hardware potential offered by ultrascale systems.

The experience with current HPC systems has shown that some of the available application codes and also some of the well-developed algorithms are not suitable for the hardware, since their internal structure and behavior lacks a high degree of parallelism and flexibility [30, 89]. This situation is expected to be even more critical for ultrascale computing. An important starting point for investigating the potential of ultrascale computing is to identify algorithms, applications, and services amenable to ultrascale systems. In addition, the requirements that need to be fulfilled to port applications to ultrascale systems have to be identified. This will enable the development of new applications that will conform to these requirements and recommendations.

In this article, we discuss a large variety of aspects which might be crucial for application codes to be suitable for ultrascale computing systems and to exploit the compute power for achieving a sufficiently high performance and scalability of those applications on emerging ultrascale platforms. In this context, we highlight the issues that are important for migrating existing parallel applications to ultrascale platforms. Application areas amenable to ultrascale computing include earth sciences, astrophysics, chemistry such as molecular dynamics, material sciences, life sciences such as analysis of short-read sequencing data, health science, high energy physics such as QCD, fluid dynamics, coupled multiscale and multiphysics methods. In addition, diverse applications for analysing large and heterogeneous data sets in social, financial, and industrial contexts are candidate areas for ultrascale computing. To illustrate the significance of these issues, we first review the current state-of-the-art in HPC execution of a typical multi-scale simulation with separated spatio-temporal scales. We then proceed with a review of several real parallel applications that the authors of this article are working on and discuss the likely impact that the ultrascale execution paradigm will have on these applications.

Ultrascale computing offers a way to provide sufficient computing resources for a persistent increase in problem sizes and parameter sets needed to process increasingly larger computational tasks in a required amount of time. It is general consent that applications will have to be re-designed or re-programmed substantially in order to perform efficiently on the heterogeneous hardware and to exploit fully the available technology of ultrascale computing systems [26]. This might require new data structures, new algorithms, or even new mathematics. New programming models for flexible coding and performance adaptation as well as more abstract and

advanced programming interfaces and domain-specific languages might be the key components for delivering highly sustainable and scalable applications. However, each redesign of an application for ultrascale computing systems must take into consideration cross-cutting issues, which include resilience and fault tolerance mechanisms, handling of data I/O, especially for a growing amount of data on geographically distributed systems (big data), power management and energy efficiency as well as programmability and portability with respect to the underlying ultrascale hardware system.

This article discusses the challenges for achieving ultrascale performance from an application perspective. The rest of the article is structured as follows: In Section 1 we discuss hardware as well as software development and program execution issues related to ultrascale computing. Then, in Section 2 we consider a number of specific applications from different areas and discuss their potential and requirements for ultrascale computing. Section 2.2.7 concludes the article.

1. Hardware and software issues for programming ultrascale computing applications

Designing applications for ultrascale computing systems includes a multitude of different interacting challenges. Firstly, this involves programming of a functionally correct application software that provides correct simulation results. In this context, it is useful that the application is based on a developed mathematical formalism of the scientific problem. A numerical computation problem, for example, can benefit from algorithms that have proven asymptotic convergence and provide error estimates. Secondly, the ultrascale application software has to fulfill non-functional properties, including a large variety of criteria ensuring software efficiency.

There is a whole range of hardware and software issues and challenges associated with computing on ultrascale platforms. From user's requirements point of view we emphasize time constraints (closely related to the total execution time and the reservation of HPC resources), the energy consumption, resilience (measured as the average time between consecutive failures within the system), and the impact of speed, security and quality of service of the interconnection networks. From the user involvement perspective, the productivity, i.e., the effort required to develop an ultrascale application (either from scratch or adapting an existing application to a new computing platform) is relevant. This section discusses a number of hardware and software properties that are expected to be of utmost importance for ultrascale computing.

1.1. Hardware and infrastructure issues

The emergence of new hardware platforms aimed at achieving ultrascale performance involves new heterogeneous technologies, such as accelerators (GPUs, FPGAs, or many integration core (MIC) architectures), as well as techniques for reducing energy consumption or network enhancements.

1.1.1. Power consumption and energy efficiency

The reduction and control of the energy consumption per flop (floating-point operation) is essential for achieving sustainable ultrascale computing. Energy-efficient processors with features such as power gating or DVFS (dynamic voltage frequency scaling) are designed to enable a reduction of the energy consumption at hardware level. These energy-saving hardware features

are supported at different software levels. At the system software level, appropriate runtime systems can be developed to map computational parts of an application to hardware resources such that the overall energy consumption is reduced. The runtime systems are based on suitable load balancing and scheduling methods with the goal to minimize the resulting energy consumption. Access to energy-minimizing runtime systems at the application level opens an opportunity for developing energy-aware ultrascale applications.

As a basis for an energy-efficient mapping of computations to hardware resources, suitable energy models are required that capture the power and energy behavior of application programs [83]. These models have to take the computational characteristics of an application into consideration to provide good estimates for its power consumption on a target architecture. The availability of suitable energy models is an important requirement to address the energy behavior of ultrascale systems from the application perspective. Accurate energy models depend on the identification of the key influencing factors, which is still an open research question.

At the application level, it can be observed that different applications may lead to a different power consumptions, depending on the computational behavior of the application and the resulting usage of hardware resources [84]. For ultrascale systems, the memory access and communication patterns will definitely play an important role. The redesign and reimplementation of the algorithms/codes for ultrascale applications need to address the problem of extensive communication patterns. In addition, the use of specialised architectures such as FPGAs, which are known to have favourable flop/Joule ratios [46], can be considered for reimplementing some frequently used kernels from scientific codes, for example, parts of the linear algebra routines.

1.1.2. Sustainable data storage and data management

Ultrascale applications have a wide variety of data storage and management requirements. The execution time of traditional HPC applications is typically dominated by floating-point computations, i.e. they are computationally intensive. An emerging class of ultrascale applications are big data applications [50, 92, 95], for which the application performance is determined by the performance of data access operations. Such applications are essential in fields such as genomics, astronomy, high-energy physics, or data analysis in web-scale companies. Some complex applications also combine computationally intensive and data intensive parts. Examples from the life sciences domain include machine learning techniques that process, among other inputs, high-resolution images.

Computationally intensive applications are typically executed on hardware platforms with a centralized network-attached high capacity storage system. Such platforms may restrict the choice of resource allocation for the distributed multiscale applications to be described in Section 2. Applications transfer data from the storage system to the compute nodes, execute the computations, and write the results back to the storage system. The data transfer may be transparent to the applications, or exposed through a library for parallel I/O such as MPI-IO. Since the execution time is computation bound, the I/O bandwidth does not significantly influence the application performance. In contrast, data intensive applications require much higher I/O bandwidth. To achieve high aggregated I/O bandwidth, the storage can be distributed among the compute nodes such that the data to be processed is read directly from a local disk. The Hadoop Distributed File System [90] (an implementation of the Google File System design [35]) provides such a distributed storage system. Current distributed storage systems such as Spanner [22] provide data management services for data stored on geographically separated sites.

Data-intensive applications may be implemented using programming models such as MapReduce [25] or Spark [104], high-level languages such as Pig [76], SQL-like languages such as HiveQL [96], or libraries such as Mahout [78]. In addition, applications may require services such as fault-tolerance [25], random I/O [21], low-latency operations [59], iterative computations [104], incremental computations [39], transaction support [22], or secure data storage.

1.1.3. Self-configurability

FPGA accelerators, being reconfigurable, offer some desirable possibilities for introducing self-configurability in ultrascale platforms, allowing a flexible re-adjustment of the hardware features to match the requirements of a particular application. Compared to general-purpose computers of equivalent performance, FPGAs are characterized by a better performance to power consumption ratio and lower cost. The combination of a general-purpose processor and application-specific processors synthesized in a reconfigurable logic with a structure utilizing features of the executed algorithms allows an increase of the overall performance by orders of magnitude.

However, FPGA-based accelerators and reconfigurable computer systems (that use FPGAs as a processing unit) face some typical problems: (1) The process of coding applications requires a special program to perform computing tasks load-balanced between the general-purpose computer and the FPGAs. (2) FPGAs require designing application-specific processor soft-cores; (3) FPGAs are only effective for certain classes of problems and data patterns, for which the application-specific processor soft-cores have originally been developed. Problems related to designing heterogeneous computer systems with hardware accelerators, are discussed in [71].

1.1.4. Interconnection networks

The interconnection network (ICN) is a critical element of every high performance computing system. It strongly determines its overall performance as well as the development and the operating costs. Enabling future advances in ultrascale computing requires the development of efficient, flexible, and highly scalable ICNs. The main responsibility of an interconnection network is to provide fast and reliable data transfer with respect to point-to-point and collective communication. The requirements for communication performance of the network imply high and stable maximal bandwidth and low latency. In a contemporary parallel system, the ICN connects hundreds of thousands of computing nodes. As the amount of computing nodes increases, the communication traffic and the resulting latency can rise dramatically, resulting in a degradation of system's computational performance. In order to overcome the scalability limitations, the developers usually implement enhanced interconnection networks based on high radix switches and specially tuned up topologies, routing algorithms and flow control mechanisms. Furthermore, the operating system and the management of the communication between the processes are highly optimized to efficiently utilize the communicational resources. From user's point of view, the underlying structure and characteristics of the network are known and can be used for optimization of the parallel code. In a sense, it is of paramount importance for the resulting performance to take the structure of the ICN into account when developing a specific code for ultrascale systems.

Due to their direct influence on speed-up and scalability, and consequently the run-time and power consumption of applications, ICNs play an important role in cooperating and coordinating

ultrascale computing systems. Today the ICN's performance has the same relevance as the performance of the CPU because the execution time depends on both communication time and computation time. The efficiency of most realistic parallel applications is determined to a large extent by the architecture's ICN. Matching the application communication patterns to the architecture of the ICN can shorten the overall execution time and increase the number of processors that can be efficiently exploited, which both leads to a higher ultimate speedup. The throughput, performance, low latency and quality of service of the ICNs are crucial for achieving scalability and good performance when applications are run on federated HPC resources (see, for example the multi-scale model described in Section 2). The performance of the ICNs for realistic multiscale applications has been studied in detail in [23].

The performance of ICNs depends on many factors with the three most relevant ones being topology, routing, and flow-control algorithms. The routing and flow-control algorithms have advanced to a state where efficient techniques are already developed and used [24]. Many network topologies have already been present since the dawn of parallel computing and are still widely used. With contemporary standards like Infiniband, vendors and end-users do not have the possibility to alter the routing and flow-control. Recent initiatives in the Network Functions Virtualization (NFV) support software-based virtual implementations of networking devices [28] such as switches, routers, firewalls, traffic analyzers, load balancers, etc. NFV can be easily combined with the concept of Software Defined Networking (SDN), which improves the performance and manageability of network functions. The end users can apply SDN to define a number of different topologies, based on an anticipated usage.

A further step towards a performance increase is made possible by an improved ICN topology or by innovative technological approaches in optical networking, which could solve current ICN bottlenecks such as message latency and non-efficient collective communication. New approaches in Networks on Chips (NoC) with high-level node radices will be considered as an option for further improvement of the performance on the chip level. It is expected that ICNs will be able to adapt dynamically to the current application in some optimal way in the near future. It is important to analyze the applications requirements regarding the currently used ICN technologies in HPC parallel systems [97], focusing on ICNs used in the present top-level systems. Based on past and present technology trends it is also relevant to establish several proposals for future development of ICNs that are expected to fit better the needs of high-performance parallel computer applications [98] or to be specifically tailored to exascale applications.

Most of the applications from Section 2 are sensitive to the ICN performance because of underlying matrix operations and advanced data structure with complex, often non-local, data manipulation. For example, well-known parallel performance degradations that can be observed for computations with sparse matrices, see Section 2.2.6, could be overcome in part by developing data traffic models that try to optimize those for frequently used sparse matrix kernels. These computations are inherently problematic on parallel architectures due to their low computation to communication ratios.

1.1.5. Cloud computing

The cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized compute facilities, which are shared between users and can be used by a service mechanism. Thus, the cloud can play an important role for ultrascale computing. The term cloud computing refers both to the hardware and software providing the services and to

the application software provided as a service over the internet [11]. A network of parallel and distributed compute facilities belonging to different administrative domains has already been used in grid computing, which has been successful in scientific applications usually provided as task applications, scientific workflows, or MPI applications. However, grid computing has also some disadvantages, such as applications not fitting to those programming models or issues of being unable to get access to grid infrastructures. Cloud computing aims at solving these problems by, e.g., providing the entire computing stack from hardware to the application level and a pay by use basis.

The use of cloud computing for scientific application and high performance is being discussed and investigated recently. Programming models comprise task models, thread models, MapReduce models, scientific workflows, actor models or the parameter sweep model (PSM) [100]. Popular infrastructures include Amazon EC2, Google App Engine, Microsoft Azure, or Manjrasoft Aneka. The performance and cost are of specific interest. The Amazon EC2 infrastructure is evaluated for the scientific code FEFF [85] and for several other scientific workflows [53]. Open-source cloud computing solutions, including Eucalyptus, CloudStack and OpenNebula, have been studied with respect to their use and performance in geosciences [45]. A recent study on the application of cloud computing to scientific workflows discusses data intensive applications [13]. The active scientific discussion of this topic shows that there are very promising approaches and that cloud computing may play a very important role for scientific applications in ultrascale computing. The integration of cloud computing and big data is a necessary next step and first studies are given in [10].

1.2. Program Execution Issues

Program execution issues comprise properties such as scalability, resilience, security, integrity and privacy of data, which are connected with properties of the actual processing of applications.

1.2.1. Scalability

Throughout the development history of parallel systems, scalability was one of the pivotal features, and will remain so when moving to ultrascales. Two aspects of scalability have to be mentioned. The first one is the scalability of the hardware resources, ensuring that adding more such resources will allow the maintainance of the system's properties and operability. The second aspect is the software scalability, measured as the effect of increased computational and communication load as well as increased input and output, which can sustain a near-peak performance. A scalable system should allow an upscaling from few-variable small data sets on current parallel systems to many-variable large data sets on future ultrascale systems while maintaining high level system's efficiency.

Two principal types of hardware scalability can be distinguished: *scale-out scalability* means that more compute nodes are added to a system, and *scale-up* scalabilty means that more resources are added to a node, where the previously mentioned heterogeneity at micro-level can play a role [72]. Scalability raises difficult challenges as well in the case of substantial increase of the size of databases where strong consistency might be needed to be replaced by the weaker eventual consistency.

Software scalability captures the behavior of an application on a hardware system with substantial computing resources on which the application should exhibit satisfactory efficiency [74]. For HPC systems, software scalability is typically related to the execution time and the resulting speedup when running the application. We distinguish between *strong scaling* or *fixed size scalability*, capturing the scaling behavior of a problem of fixed size running on increasing resources, and *weak scaling*, capturing the scaling behavior when the problem size is increased in line with the number of computational resources, thus keeping the computational load per resource, e.g. the CPU core, roughly constant. However, weak scaling is much more relevant for ultrascale execution. Many applications, such as those presented in Section 2, would struggle to achieve very good strong scaling beyond hundreds of processors, but with a suitable load per processor may weakly scale way beyond that. Software scalability is especially important, since the resources of the ultrascale systems should be used efficiently without a significant re-writing of the application code when porting it to another hardware platform.

As very different heterogeneous system architectures are expected to become available in the near future, software adaptivity also plays an important role in the context of ultrascale systems. Ideally, the application software should be able to adapt automatically or with minimal changes to a new execution situation on a new architecture and a good scalability may ease this process. Scalability for ultrascale systems requires novel or improved approaches, such as task-based approaches or Network Functions Virtualization and Software Defined Networks as previously discussed in Section 1.1.4.

1.2.2. Resilience

Efficient utilisation of ultrascale computer architectures in scientific computing is restricted by possible deficiencies in the availability and reliability of the resources. To handle hardware failures, the software needs fault-tolerance features, such as checkpointing and rollback facilities. In particular, the possibility of unavailable resources requires that an application has frequent checkpoints for synchronisation and correctness verification. For example in the case of a multiscale model described in Section 2, the information exchange points between the models are the natural choice for checkpointing. Changing of the runtime system configuration caused by failing processing elements are usually difficult to handle by deterministic numerical algorithms and require a complete restart of the application. This can be alleviated to some extent by checkpointing. Efficient mapping of numerical algorithms to high-end architectures should allow for robust execution in the presence of hardware failures, either by the ability to take preemptive actions before a failure affects a running application, or, by creating a (hardware/software) fault-tolerant version of the algorithm, capable of recovering the solution within a timescale that is much shorter than that of re-running the entire application. An example is a fault-tolerant multigrid solver for exascale computation [47]. In addition, the computational error introduced by this process should be mathematically bounded, e.g. easy to measure and control. Some algorithms, such as iterative solution methods, can handle a certain amount of (non-repetitive) hardware failures and regain consistence without requiring any additional hardware features, see [19] for a more detailed discussion.

1.2.3. Security, integrity and privacy of data

Security, integrity and privacy of data are essential in the development of state-of-the-art ultrascale computing systems, in particular if these are cloud-based. The intrinsic heterogeneity of such systems induces flexible and ever-changing structures in the sense of geographical and logical distribution of the hardware resources, thus allowing scalability of the system to a very large size. Unfortunately, this architectural concept requires numerous remote data transfers. The problem is aggravated when we take into account that nodes distributed on different geographical points could be instructed to execute simultaneously several codes on a given data set, which is frequently locally stored.

The transfer and storage of data poses security threats, which includes risks involved in the transfer itself and security risks connected with the enormous scaling of the system. Each time a new node is connected to the system, the security risks could increase, as the new node could be infected by an ill-intended code or it could be a Trojan node. Moreover, frequent data transfers could be intercepted and the data could be stolen and used for unwanted purposes. The data security, especially in today's big data world, still remains an open problem. An interesting concept that is yet to be investigated is to consider whether computations could be performed on coded data.

1.3. Program Development Issues

Program development may constitute a significant effort, especially for ultrascale systems, and effective support for reducing program development time is important. In this section, we discuss the related issues of programmability, portability, and productivity in more detail.

1.3.1. Programmability

Programmability of highly parallel, heterogeneous computing systems is a major concern for potential applications. It expresses the ability to implement an application in such a way that ultrascale computer systems can be efficiently exploited to ensure its high-performance execution and good utilisation of computer resources. The term programmability includes the requirement for portability, since parts of or the entire application may have to be ported to newer and larger hardware platforms. Well-established and standardized programming models, such as MPI or OpenMP, as well as portable libraries or simulation tools are important to support portability. In the context of grid applications, such as a distributed multi-scale problem discussed in Section 2, various flexible coupling tools have been developed to couple existing (parallel) single-scale models. Such coupling tools should be flexible and generic as much as possible, thus minimising the development effort and maximising software reusability [14, 16].

However, porting codes are usually not enough to achieve high performance and a redesign or sometimes even a reimplementation of an application might be required. The complex task of redesigning an application has to be supported by a programming environment and a concise programming model for ultrascale applications. Such a programming model should provide an abstract view of the coarse (top-down) structure of an application. The specific way to support programmability is still to be investigated and proposed solutions may be application-specific. For example, a formalism based on complex automata was developed for the design of multiscale models, and a markup language, called MML has been designed to allow their formal description [15, 44]. The implementations based on the abstract view may include many well-

known subsolutions and the inclusion of standards, such as MPI, seems reasonable [82]. The requirement for the support of programmability will be investigated for specific applications in Section 2. Programmability is also strongly related to productivity in code design for sustainable ultrascale computing.

Task-based programming approaches in combination with suitable runtime systems can support the software adaptivity of applications, since the task-based approach allows a hardware-independent formulation of the application and the mapping of tasks to hardware resources can be performed by a runtime system so that the resources are efficiently used. The task-based programming decouples the specification of application's computations from the actual mapping to the computing resources. The runtime system can dynamically map tasks that are ready for execution to the computing resources, thus providing a dynamic load balancing that can adapt to the current execution situation of the hardware platform. This can help to enable an efficient use of the computing resources and a good overall scalability of the application, provided that enough tasks are available for execution at each point in time during the execution of the application. An example of a high-level runtime system that allows the allocation of federated HPC resources for distributed component applications, e.g. a loosely coupled multiscale model, is the Application Hosting Environment (AHE) [38]. Task-based approaches can be used with single-processor tasks [58], where each task is executed by a single execution unit, or multiprocessor tasks, where each task can be executed by multiple execution units in parallel [80, 81]. In the latter case, the actual number of execution units can be adapted to the execution situation at the time of task execution. Task-based approaches can also be used for balancing load between CPU and GPU by providing tasks in different versions for different platforms such as CPU or GPU and assigning a suitable version to the platforms with free resources [66].

The task-based programming paradigm is a promising approach for developing scalable solvers for ultrascale computers. However, it has to be taken into account that a considerable number of large real world applications are dealing with parallel algorithms for models based on partial differential equations (PDE). Parallelization of such algorithms is based on the paradigm of data parallelism. It is important to investigate whether the existing parallel algorithms can be redesigned by using the task-based templates (e.g. Monte-Carlo methods).

1.3.2. Portability

The portability of parallel codes and algorithms is an essential issue for any specialist involved in parallel computing and applications. There are two aspects of portability: portability of functionality and portability of efficiency. Clearly, the portability issues are mainly connected to selection, definition and continuous improvement of programming languages and standards such as MPI, OpenMP and hybrid programming MPI+OpenMP. These have served as programming models very efficiently for the last 20 years. Now the situation is changing and new parallel architectures such as manycores GPU require new ideas and tools, e.g. CUDA. Interesting approaches in this direction include several new languages that are based on a Partitioned Global Address Space (PGAS) concept which uses a global address space that is logically partitioned between the resources such that each resource has a portion of the address space attached to it to support the locality of memory reference. Languages, such as Cilk, that are based on the concept of tasks or task-based libraries also provide a useful abstraction that can support the portability to new hardware systems, see the discussion in Section 1.2.1.

Another way to provide portability is to use special libraries and templates or macros, well adapted to some specialized types of problems. This is often done for stencil driven algorithms. Examples of such libraries and toolkits are PETSc, LAPACK, ML, Hypre, CVODE [3, 4, 6–8]. These software packages are expected to be highly scalable and adaptable when used in ultrascale systems incorporating different architecture. Due to the portability of the libraries, their use can increase the portability of application codes. There is an interaction between the development of scientific libraries and the usage of new programming approaches for the development of these libraries in the sense that the usage of specific programming approaches influences the characteristics and the efficiency of the libraries. The third way to provide portability is to use simulation tools such as OpenFOAM, COMSOL, ANSYS [1, 2, 5], where the portability of solvers is guaranteed by the developers of the software packages.

1.3.3. Productivity

Productivity refers to the efficiency of implementing applications on specific architectures. The resulting application should be functional and reasonably efficient. Productivity for ultrascale systems involves the effort required to extend existing (parallel) applications to the new ultrascale systems such that the available resources are sufficiently well exploited. It is clear that a complete re-architecturing and re-implementation of the application with a new programming model should be avoided if possible. For large applications, this would be a huge effort even if large code blocks could be re-used. However, it would be unreasonable to expect that no change in the software will be required in order to use it efficiently on an ultrascale system. A preferred scenario involves applications that can be adapted with minor changes to the new platforms. Another aspect of productivity is the extensibility of the application code to include new features and functionalities without negative effect on its scalability and efficiency.

The reimplementation effort needs to be sustainable in the sense of making an application reasonably efficient on various ultrascale architectures. The use of novel task-based runtime systems which decouple the concerns of the computation specification and its mapping to computational resources can be an important step towards an increase in software development productivity for ultrascale systems. Productivity is inherently intertwined with other requirements discussed in previous subsections. In particular, a good portability as well as a good scalability of an application code leads to a higher productivity.

Related to productivity are radically new cross-platform software development and performance analysis tools aiming at increasing the capabilities of the codes to take an advantage of the phenomenal power of ultrascale computing platforms. Examples of highly scalable debuggers that target exa- and ultrascale systems are TotalView and Allinea, that provide troubleshooting for a wide variety of applications including serial, parallel, multi-threaded, multiprocess, and remote applications.

Performance analysis of parallel codes is already increasingly difficult at existing scales. Therefore, novel paradigms and techniques to measure, track, analyse and visualize performance data are necessary to be developed, in order to facilitate faster and more intuitive analysis of a wide range of gathered performance data, including execution time, memory system behavior, power consumption and resiliency to faults.

2. Some specific applications and implementation requirements

In this section, specific applications are investigated concerning their requirements for exascale execution, as identified in Section 1. After discussing a prototype multisclae application in Subsection 2.1, we present several specific applications in Subsection 2.2.

2.1. A prototype multiscale application

Processes and phenomena of interest in many scientific disciplines involve a complex mix of sub-systems that may operate on inherently different spatio-temporal scales. To model accurately the behaviour of such systems one needs to develop a scheme which would capture with sufficient detail the contributions of each sub-model, and couple them seamlessly into a global, computationally feasible system. Such models are commonly referred to as *multiscale models*. Multiscale modelling has numerous applications, including astrophysics (simulation of thermonuclear processes in stars and galaxies [34]), biology (studies of live organisms, spanning from genome to the entire population [88]), high energy physics (modelling of the fusion process and nuclear reactors [41]), engineering (simulations of structures, devices and chemical processes [69]), environmental science (climate modelling, weather prediction [57]), and material science (nano-composites [93]), to name a few.

The main components of a multiscale method are the single scale sub-models, the scale bridging techniques, and the deployment strategies. To emphasize the challenges related to performing computer simulations of a multiscale problem on a ultrascale computing facility we restrict our attention to a model application involving two single scale sub-models with well separated spatio-temporal scales. The single scale sub-models are assumed to be implemented as parallel legacy codes (e.g. using MPI or OpenMP). Putting the single scale sub-models together is the main algorithmic and software engineering challenge in multiscale modelling. This process is referred to as the *scale bridging*. Domain-specific techniques, such as sampling, projection, lifting, homogenisation (coarse graining), micro-macro coupling are the instances of scale bridging techniques. In terms of the coupling strategies, we can distinguish between *tight* coupling and *loose* coupling. Tight coupling is a feasible alternative when spatio-temporal scales of the sub-models are close, or partially overlap. In such cases, monolithic coupling may be an advantageous option [40]. Loose coupling strategies are effective for multiscale models with sub-models operating on well separated scales. This approach is more flexible in terms of the reuse of legacy codes for single scale sub-models and their deployment on distributed HPC resources.

At the methodology level there is a number of challenges that are awaiting answers, for example, finding generic theories and formalisms for model coupling, defining the minimal set of conservation laws for scale bridging, formulating mathematically rigorous theories for multiscale modelling, including the error analysis [43]. At the implementation level, scale bridging is usually handled by the software middleware, commonly referred to as *coupling framework* [38]. In terms of the coupling patterns, we can distinguish *acyclically coupled* simulations, in which the sub-models (codes) are run sequentially, with the output of one model serving as an input of the other (i.e. the sub-models are not mutually dependent during the execution), or *cyclically coupled*, where a mutual interdependency (a feedback loop) between the sub-models exist. In the latter case the sub-models can be run either sequentially or concurrently.

Concerning the mapping of computational tasks to a specific architecture, distributed computing strategies are of particular interest for achieving ultrascale performance. An efficient

mapping of sub-models depends on the software systems that handle the advanced reservation of federated computational resources, monitor data transfers, and provide easy to use GUI on a server. An example of such a system is the Application Hosting Environment (AHE) [105].

Next, we discuss how the issues covered in Section 1 affect the execution of a distributed multiscale application. In terms of the hardware issues, mentioned in Section 1.1, power consumption and energy efficiency can be addressed by deploying single scale applications on the architectures most suitable for the underlying algorithms (see the discussion in [14]). In this context, the use of hardware accelerators for certain sub-tasks within single scale sub-models can be beneficial, providing that the legacy codes support such extensions. Based on the existing experience, the impact of the interconnections speed on the performance of distributed multiscale applications is well documented (see [23]) and no drastic change of the behaviour is forseen. If cross-site applications with significant data traffic are executed, or the applications have requirements for interactive graphical rendering and steering, the existence of high bandwidth, low latency dedicated network interconnects between the HPC sites is one of the crucial factors for achieving high-performance execution.

In terms of program execution issues, scalability of a multiscale application depends on the choice of HPC architectures to execute single scale sub-models [14]. This would involve considerations, such as employing data locality when using large data sets (cf. Subsection 1.2.3) and software licenses. To achieve better scalability of single scale sub-models, some reprogramming and even redesign of algorithms may be necessary. This also applies to the attempts to improve the resilience of single scale models, as well as utilize energy saving options if such are provided.

In terms of programming development issues, such as programmability (Subsection 1.3), and productivity (Subsection 1.3.3), the strategy of using well-established legacy codes, coupled together with general-purpose coupling frameworks [16], [38] and deployed flexibly on distributed HPC resources via the virtualisation tools, such as AHE [105] is currently considered to be state-of-the-art. However, significant programming effort may be needed in some cases to bring the single scale models to scale well on the emerging ultrascale architectures.

2.2. Scientific Computing applications

In the remaining part of this section we present seven applied problems, that are based on the solution of discrete systems of PDEs and lead to important classes of supercomputing applications. During the last few decades computational mathematics and numerical simulations have been steadily drawing much attention, enabling the development of advanced technologies and contributing to better understanding of numerous natural phenomena that are not tractable via classical theoretical research or lab or field experiments. Performing numerical simulation of very complex physical, biological or social systems enables the society to address important issues such as identifying environmental problems, improving technological processes, developing biomedical applications, new materials, etc. In addition, numerical simulations are sometimes the only viable option in studying large systems, for example when the experiments are too expensive, time consuming or unethical to perform.

A significant class of mathematical models involves partial differential equations (PDEs), which are converted into discrete models using some suitable discretization methods such as Finite Differences (FDM), Finite Elements (FEM), Finite Volume (FVM), Isogeometric Analysis (IgA), referred to as *local methods* or as Boundary Integral Methods (BEM), meshless methods or spectral methods, referred to as *global methods*. Some combinations of local and global methods

are also in use. We note that, in general, the local discretization methods give raise to very large linear algebraic systems of equations with *sparse* matrices, while the matrices arising from global methods are much smaller, but *dense*. The type of discretization method is tightly related to certain data structures that in turn have a significant impact on the potential performance of those problems, when implemented on HPC and parallel computer platforms.

The methods of choice for solving large sparse systems of linear algebraic equations are the iterative solution methods, in particular, the preconditioned Krylov solvers, see [87].

2.2.1. Finite element based supercomputer applications

Problem 1 [FEM]: Nowadays, FEM are considered as one of the leading computational technologies for continuum (macroscopic) modelling in science and engineering. The advanced FEM-based simulations are often inter-disciplinary and involve multiple spatio-temporal scales, leading to practically unlimited requirements for supercomputing resources. The FEM supercomputing applications are inherently computationally intensive. At the same time, the most frequently used algorithms and their parallel implementations, are strongly coupled, e.g., via scalar products that entail global reduction operations, causing specific requirements with respect to the balance between computations and communications. In this context, we focus on topics related to single process problems (scalar or vector) which could be stationary (e.g. elliptic PDEs) or time dependent (e.g. parabolic PDEs).

More than 70% of the entire computing time of FEM-based engineering simulations is spent in solving linear algebra problems. This can be verified by using popular libraries, such as Trilinos [8] and HYPRE [3]. The included efficient fast parallel preconditioned conjugate gradients type solvers are often of (nearly) optimal complexity, see e.g. the available implementations of algebraic multigrid (AMG) methods. However, the robustness of these solvers for some classes of problems is still a challenging problem. As an example, we mention models of strongly heterogeneous media and/or strong anisotropy, where the well-established solution techniques face difficulties. The same applies to singular perturbations of the elliptic PDEs, such as the convection-diffusion problem, which is a major issue in fluid mechanics and transport phenomena.

The efficient implementation of FEM models requires mesh generation and partitioning of the graph representing the sparsity pattern of the matrices in the resulting linear systems. The available mesh generators construct a coarse unstructured mesh (for example using Netgen), which is then refined uniformly in parallel. This is not necessarily a computationally optimal scenario and adaptive refinement may be a better option, but requires frequent grid repartitioning to preserve the load balance. One potential solution to this problem can be the low cost mechanism for particle distribution, described in Subsection 2.2.7. The complete parallel generation of conforming unstructured meshes is still a challenge. The next related problem concerns the mesh partitioning. To illustrate it, we could refer to the commonly used software packages ParMETIS and SCOTCH. These packages are based on recursive partitioning strategies balancing the measure of the sub graphs and minimizing at the same time the measure of the interfaces. In this respect, the quality of the results could be considered as acceptable. The problem is that the number of the neighbors is not properly controlled which leads to serious problems mapping the graph of the algorithm onto the graph of the parallel architecture.

The last related comment concerns the more general problem of balancing local and global communications. For parallel distributed systems with hundreds of thousands of pro-

cessors/cores, the global communications related, e.g., to dot product, Fast Fourier Transform (FFT), etc. have become one of the fundamental bottlenecks, indicating that some of the user communities will have to change their way of thinking. As a consequence, most probably some of the FFT-based codes will have to be modified to AMG solvers as a way to avoid the transposition step reducing also avoid the logarithmic factor in the almost optimal order of computational complexity. Similar to other approaches, AMG may have its own problems when mapped to an architecture with a large number of processors – recent works aimed at reducing the operator complexity, improving the quality of interpolation, reducing the communication patterns at coarse levels, exploring fine-grained parallelism, while retaining numerical robustness are definitely of interest.

Challenges towards ultrascale FEM applications: The energy efficiency of FEM applications on ultrascale systems is one of the key challenges. New proper metrics need to be developed and tuned to get a complex assessment of the related simulators. The fault-tolerance issues need to be addressed in a proper way at the method, algorithm and software implementation level. Iterative solvers and the time-stepping algorithms have inherently self-correcting mechanisms which should be developed further and tuned in the context of ultrascale computing. The development of specific algorithm-based fault tolerance mechanisms will become increasingly significant with the scale of the computer system. For example, faults that occur in the parallel geometric multigrid solver are studied in various model scenarios in [47].

The general conclusion is that systematic algorithmic adaptations will be required if anticipated ultrascale hardware is to fully utilise its potential for FEM applications. Such algorithms aim to minimize synchronizations, memory usage, and memory transfers, while extra flops on locally cached data are almost "free", see e.g. [55]. In multilevel/multigrid solvers this could be achieved by more aggressive coarsening. An important complementary approach is the hybridization of algorithms (including hybrid deterministic-stochastic solvers), aimed at better fit to the hybrid hardware architecture.

Current experience: The experience of the IICT-BAS team includes FEM supercomputing simulations of bio-medical, environmental and engineering problems. High parallel scalability (both, strong and weak) is obtained on heterogeneous Linux platforms, including IBM Blue Gene/P, HPC CPU clusters, and more recently hybrid CPU/GPU/MIC clusters. The used programming methods and environments include C, MPI, OpenMP, CUDA. Among more recently released libraries for platforms with accelerators, we could mention PARALUTION. More information can be found in [74, 75, 79].

2.2.2. Parallel preconditioning of multi-physics problems

Problem 2 [BlockPrec]: Recent advances in computational modelling techniques and the increasing computing power allow us to tackle complex multi-physics and engineering problems that involve several unknown physical quantities described by a system of PDEs, such as thermal convection, fluid-structure interaction, magnetohydrodynamics. Grouping the discrete unknowns of the same type imposes in a natural way a block structure on the coefficient matrix. In this context, block preconditioners are commonly deployed to accelerate the convergence of Krylov solvers. The block structure of the coefficient matrix enables the use of existing preconditioners for the constituent single-physics sub-problems and available software libraries (such as AMG implemented in Hypre [3] or Trilinos [8]) to solve approximately the scalar subproblems. Implementing the solution of subproblems using available highly tuned and computationally

efficient software toolboxes reflects the programmability and productivity issues, highlighted in Section 1. Block preconditioners also favour local interprocessor communications, as opposed to long range communications that are prevalent in global multigrid solvers for multi-physics problems.

Current experience: We have developed the block preconditioning framework (BPF) within OOMPH-LIB (see http://oomph-lib.org/), an object-oriented multi-physics finite element library [29, 73, 91]. The BPF facilitates rapid development of new preconditioners, while hiding the low-level implementation details (including parallelisation). However, the overall parallel performance of any multi-physics solver crucially depends on scalar solvers (usually library codes produced by third parties), such as algebraic multigrid (AMG). Standard AMG codes, such as BoomerAMG [3] (developed at LLNL) perform robustly and show good scalability over hundreds of processors. Novel coarsening techniques (PIMS, HIMS, non-Galerkin) and techniques for enhanced (long-distance) interpolation are designed to reduce communication and enhance scalability, while retaining robustness, especially for complex diffusion and convection-diffusion problems.

Challenges towards ultrascale FEM applications: In contrast to the extensive studies of the numerical properties of these methods, there is comparatively little work on resilience and energy efficiency of AMG solvers (some recent results are found in [47]). AMG and the more general multilevel and domain decomposition frameworks are among the most numerically efficient techniques for solving large scale linear systems with sparse matrices, arising from discrete PDE models. The numerical efficiency, usually measured in the number of iterations, required to obtain convergent solution, relates to the underlying hierarchical structure of these methods, that allows a fast transfer of error information through the utilization of *coarse* problem representations. The implementational counterpart of the coarse data structures is the long range communication, and this is expected to be a serious bottleneck for ultrascale systems. Local communications are very attractive for massively parallel implementations, however in general, these do not guarantee sufficient numerical efficiency. Therefore, upgrading the existing and developing new optimal linear solvers that perform efficiently on ultrascale computers while satisfying resilience and ennergy efficiency requirements is very relevant.

2.2.3. Numerical solutions of multi-physics problems using Meshless methods

Problem 3 [Meshless]: A common feature of the local discretization methods, such as DFM, FEM, FVM, IgA, BEM, is that they rely on some discretization mesh and this fact sometimes poses additional difficulties as resolving complex geometries, adaptive refinements that involve local mesh refinements and derefinements, large stencils, handling moving meshes to resolve dynamical interfaces, etc., see Subsection 2.2.1 and [73]. The latter requirements affect the parallelization and implementation of these methods on HPC platforms, namely, the load balancing, the amount of local communications, etc.

A promising alternative to the mesh-based methods is the class of the so-called Local Meshless Methods (LMM), based on scattered discretization points. Of particular interest are the methods that result in algebraic systems of equations with better conditioned matrices [63]. LMM allow for easy implementation of local refinements and derefinements [61], basis augmentation, increasing approximation quality, treating special features in the problem, such as sharp discontinues or other intricate situations, which might occur in complex simulations. These potential advantages can be usually accomplished by an increased number of discretization nodes

to preserve the desired accuracy with additional work in the identification of nearest meshless nodes that influence the solution [99]. The validation of potential benefits of LMM on ultrascale architectures remains a significant research challenge [32].

Current experience: A parallel computational framework for solving multi-physics problems based on LMM has been developed, (see http://www-e6.ijs.si/ParallelAndDistributedSystems/) providing the possibility to model and perform numerical simulations of problems originating from molecular dynamics, graph algorithms (clique) and discrete simulations (ECG simulator based on Action Potential in cells), multiple transport equations (heat, bio-heat, solute, radiation, etc.), multi-phase fluid flow (free fluid, porous media), phase change dynamics on micro-macro levels, drift-diffusion equations (semiconductor simulations) [62], r-adaptive dynamic nodal distributions, all on non-uniform domains. The code is parallelized and can be efficiently executed on multi-core computers and distributed systems [60]. The communication issues have been studied and tested on an in-house computing cluster. The solvers are coupled with an evolutionary multiobjective optimization package for automatic optimization of parameters. The local meshless code could be extended to exascale range and could be used for performance benchmarking on novel ultrascale hardware platforms.

2.2.4. Earth Sciences Applications

Problem 4a [GEO1]: Earth sciences include a number of scientific disciplines such as geology, geophysics, ecology, hydrology, oceanography, climatology etc., that are relevant for the living conditions of human society, the extraction of raw materials and circulation of wastes. Earth System Science mixes together physics, geology, geophysics, engineering, chemistry, mathematics and computations to study interconnected systems operating on extreme time and spacial scales where small-scale heterogeneities affect large-scale phenomena (see also the discussion in Section 2.1). The scientific research accommodating these scales pushes and challenges the frontiers of numerical and computational methods [31, 49, 103].

The complexity of physical, chemical and biological processes, as well as the volume of data structures makes the HPC an imperative for the analysis, modeling and simulation of the underlying processes. For such problems, where experiments are impossible, the extreme-scale computer simulations can enable the solution of high resolution models and the analysis of very large data sets, including: regional climate changes (sea level rise, drought and flooding, and severe weather patterns). Climate models, developed through decades, have over one million lines of code. At the same time the architectural changes of the computer platforms need more sophisticated algorithms and computer techniques [12, 51, 52], in order to utilize fully the computing power provided by the new hardware. Oceanographic, atmospheric and climate simulations are typical examples of applications that require HPC to process a huge volume of data (for example, the analysis of remote sensing data in space-time domains to evaluate environmental changes and predict their future evolution) [9, 68, 101]. Moreover, such an analysis must be coupled with the simulation of related environmental processes and interfaced with human activities.

A particular instance of Earth science applications is geophysical modelling and inversion. A related work within the FP7 project HP-SEE on "Geophysical Modeling and Inversion" is performed in the Center for Research and Development in IT of the Polytechnic University of Tirana in collaboration with specialists from the Faculty of Geology and Mining and of the Academy of Sciences in Albania. It illustrates the scalability of geoscience applications in HPC systems and the need for ultra-scale computing in order to cope with high resolution models

required for regional and local scientific and engineering studies. The project targets inverse problems, related to gravity anomalies using approximation based on relaxation methods with runtimes in the range of $O(N^8)$ where N is the spatial resolution in one dimension. Geosections are represented by 3D matrices, the structure and density of which changes during iterations. Due to 2D to 3D mapping, the problem is, in general, very ill-conditioned.

Current experience: The implementation has been developed in C with MPI and OpenMP. Tests have been performed on the HPCG Cluster at the Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, and the SGE system of the NIIFI Supercomputing Center at University of Pécs, Hungary, with up to 1000 cores. It was possible to achieve in a reasonable runtime a resolution with $N = 80$ nodes corresponding to a spatial step of 50 meters. This may be sufficient when simulating gravity effects but the resolution should be much higher when studying other geophysical phenomena of interest, for instance, searching for 2D structures with thickness of the order of one meter in a complicated heterogeneous and partially anisotropic medium for applications in other areas, such as magnetism and electricity.

The implementation maintains a balance between energy consumption and network communications. To facilitate this, each process on the individual cores does some redundant computations, thus reducing the communication at the expense of the increased energy consumption.

Problem 4b [GEO2]: Another example of a geophysical application problem that is of significant importance and a potential impact for our society is the so-called glacial isostatic adjustment (GIA) that comprises the response of the solid Earth to redistribution of mass due to alternating glaciation and deglaciation periods. The processes that cause subsidence or uplift of the Earth surface are active today. To fully understand the interplay between the different processes, and, for example, be able to predict how coast lines will be affected and how glaciers and ice sheets will retreat, these have to be coupled also to recent global warming trend and melting of the current ice sheets and glaciers world wide.

Due to the extreme space and time scales involved (the simulations should be performed on the Earth globe for time periods of about 100000 years) the GIA processes can only be studied via computer simulations. A detailed model of the phenomena includes three-dimensional geometry, viscoelastic and inhomogeneous material behavior, self-gravitation effects, modelled via a coupled system of partial differential equations.

Current experience: Presently, at the Division of Scientific Computing, Uppsala University, a two-dimensional benchmark, often used by the geophysicists, has been studied from a point of view of accuracy as well as regarding numerical and computational efficiency. The problem is discretized using FEM and performance studies have been done on CPU and GPU platforms using OpenMP and MPI paradigms (cf. [27]). The long-term aim is to couple GIA modeling with other large scale models, such as Climate and Sea-level changes, Ice modeling etc.

Problem 4c [GEO3]: Massive paralleism can be achieved when simulating environmental systems, for instance studying surface water - groundwater interactions. The HydroGeoSphere (HGS) software package ([17]) is an advanced tool, allowing for modelling physics-based interactions and feedback mechanisms between the two compartments. HGS is a numerically demanding code implementing a 3D control-volume finite element hydrologic model with a fully integrated surface-subsurface water flow and solute, including thermal energy transport.

The model parameters employed in HGS need to be calibrated in order to adequately represent a given environmental system. So-called data assimilation systems provide an alternative to conventional model calibration systems: they allow sequential update of system states and

model parameters whenever new data becomes available, thus guaranteeing a continuous improvement of the predictions. The Ensemble Kalman Filter (EnKF) [33] allows to quantify the prediction uncertancies, thus providing an optimal data assimilation mechanism in conjuction with HGS and the environmental data. The prediction provided by an EnKF-based simulation is then represented by the statistical moments of the ensemble of realizations. Such modelling systems are ideal for parallel processing, due to the high number of required simulations, and the readjustment and the recalibration of the model parameters allowed by the EnKF data assimilation technologies.

Current experience: At the University of Neuchatel, a cloud-based environment (OpenStack, AWS S3 compliant object store) has been developed to provide *near-real-time* re-adjustment of system states and re-calibration of model parameters whenever new monitoring data becomes available [65, 67]. When simulating larger fine grained HGS models, parallelization is essential because tightly-coupled highly-nonlinear partial differential equations are solved. The target parallelization includes the assembly of the Jacobian matrix for the iterative linearization method and the iterative solution method, in this case the preconditioned BiCGSTAB method, [48]. Performance studies of linear solvers are currently undertaken on CPU and GPU using OpenMP and MPI paradigms.

2.2.5. OpenACC acceleration for Nek5000: spectral element CFD code

Problem 5 [CFD]: This application targets problems in Computational Fluid Dynamics (CFD), in particular, very large scale simulations of incompressible flow problems, efficient and robust solvers for the arising linear systems and achieving high performance efficiency on heterogeneous HPC platforms.

Current experience: At present, the numerical simulations are performed using Nek5000 – an open-source code for simulating incompressible flows and its discretization scheme is based on the spectral-element method [37, 70]. In this approach, the incompressible Navier-Stokes equations are discretized in space by using high-order, weighted residual techniques employing tensor-product polynomial bases.

The code is widely used in a broad range of applications and more than 200 users are using Nek5000 in the world. Within the EU project CRESTA (Collaborative Research into Exascale Systemware, Tools and Applications), PDC-HPC at KTH Royal Institute of Technology mainly focuses on software challenges using hybrid computer architectures with accelerators for ultrascale simulations in collaboration with KTH Mechanics, EPCC, Cray UK and Argonne National laboratory. We have ported the CFD code Nek5000 on massive parallel hybrid CPU/GPU systems and presented a case study of porting simplified version, NekBone, to a parallel GPU-accelerated system. We reached a parallel efficiency of 68.7% on 16,384 GPUs of the Titan XK7 supercomputer at the Oak Ridge National Laboratory. Currently the full Nek5000 code is ported and optimized to multi-GPU systems and can run on 1,024 GPUs. The application is written in mixed C/Fortran and requires a system with multi-GPUs.

As discussed in Section 1.3, a particular attention should be paid to portability and productivity of ultrascale applications on heterogeneous HPC architectures. Productivity will be decreased if codes are rewritten in a low-level language such as CUDA for GPU accelerator systems. With the OpenACC [77] compiler directives, to port Nek5000 to GPU systems only requires a few additional command lines of code [70].

Nek5000 employs a multigrid preconditioner that combines the Schwarz overlapping method with subdomain solvers based on the fast diagonalization method. A sophisticated multigrid preconditioner is expected to reduce the global solution time when the Nek5000 code is ported to multi-GPU systems. For further improvements we intend to investigate the efficient preconditioner discussed in Section 2.2.2.

2.2.6. The EULAG numerical model

Problem 6 [STENCIL]: The EULAG model [86] is an ideal tool for performing numerical experiments in a virtual laboratory using time-dependent 3D adaptive meshes and complex, time-dependent model geometries. The flexibility is due to the unique model design that combines the nonoscillatory forward-in-time (NFT) numerical algorithms and a robust elliptic solver with generalized coordinates. The code is written as a research tool with numerous options controlling the numerical accuracy, and allows for a wide range of numerical sensitivity tests. The computational core of EULAG consists of two main parts: MPDATA advective transport algorithm [94] and the Generalized Conjugate Residual (GCR) elliptic solver [18] with the Thomas preconditioner. A more sophisticated preconditioner could be used in this context, see Section 2.2.2.

When implementing the EULAG model, the most time-consuming parts correspond to stencil-based computations. The suitability of stencil-based computations for ultrascale systems is studied in [56] for the CFD simulations on clusters with GPU nodes and in [102] for mesoscale atmospheric modeling on the Tianhye-2 system with Intel Xeon Phi co-processors. In the multi-GPU implementation, the PCI Express bandwidth and synchronization overheads are among the main bottlenecks. An important observation for this rather complex code is that the operations performed at the subdomain boundaries are currently performed much slower than fast stencil operations in the subdomain interiors, which restricts the overall code performance. In line with this observations, the algorithm proposed for Tianhye-2 is simplified as much as possible, allowing to achieve weak scaling efficiency up to 6,144 nodes, with over 8% of the peak performance in double precision.

Current experience: The development of the model and the implementation are a collaborative effort between the Czestochowa University of Technology, Poznan Supercomputing and the Networking Center, Institute of Meteorology and Water Management in Warsaw. The code is ported to multi GPUs and multi Intel Xeon Phi platforms [86, 94]. The application is written in C++/Fortran using CUDA, OpenMP and MPI standards. It requires a cluster with nodes containing NVIDIA GPUs or Intel Xeon Phi co-processors. The application is optimized for Fermi and Kepler GPU architectures, as well as Intel MIC architecture. Memory requirements include about 20 GB of HDD (for input and output data), and about 16 GB of RAM per node and about 8 GB of inter co-processor memory.

The performance of the EULAG system within a single node of cluster is mostly limited by the low flop-per-byte ratio of computation - less than 1.7 for MPDATA, and even less than 0.2 for the GCR solver, while the minimum flop-per-byte ratio required e.q. by NVIDIA Tesla K40m to achieve the maximum performance is 5.2. The main constraint for providing scalability of the application across cluster nodes is the presence of global communications in the GCR elliptic solver. As already pointed out, this is the crucial bottleneck for all sparse matrix calculations – a low computation to fetch ratios, made even worse on multicore architectures where multiple cores have common fetch buses.

2.2.7. Particle–In-Cell method for particle distributions – Helsim

Problem 7 [SPACE]: Space weather refers to conditions on the Sun, in the interplanetary space and in the Earth space environment. These conditions can influence the performance and reliability of space-borne and ground-based technological systems, and can affect human life or health [54].

Astrophysicists researching space weather are interested in the behavior of plasma, a high energy and highly conductive gas, where the atoms have been broken into their nuclei and their freely moving electrons. To study the small-scale plasma behavior, necessary to understand its kinetic effects, the Vlasov equation, self consistently coupled with Maxwell's equations, needs to be solved [64]. This is commonly solved using the particle-in-cell (PIC) method [42]. In PIC, individual macro-particles in a Lagrangian frame, that mimic the behaviour of the distribution function, are tracked in a continuous phase space. Moments of the distribution function, such as charge densities for plasma physics simulations, are computed simultaneously on an Eulerian frame (fixed cells).

Current experience: The software package Helsim implements an explicit three-dimensional electro-magnetic PIC simulation. It was developed in the ExaScience Lab in Leuven, Belgium, with contributions from many partners in the project. A particular care of load balancing is taken in Helism, which allows the simulation of experimental configurations with highly non-uniform particle distributions. Moreover, Helsim includes interactive in-situ visualization capabilities.

Helsim uses the Shark Library [20] [15] to store all distributed data structures, including the particles and the various grids. Shark is a high-level library for programming distributed n-dimensional grids in a highly productive manner, aiming to improve programmability and productivity while remaining portable (see Section 1.3). Broadly speaking, Shark manages the bookkeeping and the distribution of grid data structures, and offers specific computation and communication operations to work with the grid data. It extensively uses C++ constructs, such as lambda-expressions, to make the work with distributed n-dimensional grids easy. The Shark runtime system manages parallelism on three levels, which are common in today's multicore cluster architectures: distributed memory parallelism using one-sided communication from MPI 2/3; shared memory parallelism using a thread scheduler such as OpenMP, direct Pthreads, Intel Threading Building Blocks (TBB), and others; and SIMD vector instructions using compiler auto-vectorization, assisted with pragmas. The work on Shark continues in the context of the FP7 project Exa2ct, where we are integrating with the GASPI/GPI (Global Address Space Programming Interface) [16].

Specific for Helsim, when compared to other PIC simulators, is that particles are evenly distributed over the cluster such that each core holds the same amount of particles, stored according to the cell they belong to. As particles move throughout space during the simulation a low-cost, lightweight mechanism is used to adjust the particle distributions. It was initially opted for this dedicated mechanism but it could be worthwhile to use (or at least get inspiration from) grid or mesh partitioning techniques as discussed in Section 2.2.1.

The 3D fields (electric field, magnetic field, etc.) are block-distributed over the cluster, completely decoupled from the particle data structures. When particle information is propagated to the fields (charge density and current interpolation), and vice versa (interpolating the electric and magnetic fields to particle positions) each core uses a local representation of the grid in

[15] Freely available on github: https://github.com/ExaScience/shark
[16] See http://www.gpi-site.com/gpi2/gaspi/

order to be able to work locally and overlap computation with communication (updating the actual distributed grid). When done, this local information is then propagated to (and merged with) the distributed grid.

The current version of Helsim uses a fairly simple CG solver but we are currently integrating the state-of-the-art pipelined CG solvers developed at the lab [36]. It may be worthwhile to explore whether preconditioning techniques might help even more, see Section 2.2.2.

Helsim also features in-situ visualisation that runs in parallel with the simulation, directly using the data from the simulation, using a custom distributed raycasting engine. Helsim was run on up to 32 thousands cores on the Curie T0 System in France, as well as on various smaller clusters. The primary goal is to increase *weak scaling* (see 1.2.1), which is important to be able to tackle ever larger simulations. Helsim was developed at the ExaScience Lab, a collaboration between Imec, Intel, and all five Flemish universities.

Conclusions

We have put forward a number of hardware, software and program execution issues which will, in our opinion, influence the development and upgrades of computing applications for future ultrascale architectures. In this context, a prototype multiscale application and seven specific simulation applications from science and engineering serve as a testbed for a discussion on their current state-of-the-art and their future development directions, based on current experience. Although each of the applications has specific demands for porting to ultrascale systems, a pattern favouring coupled applications of increasing complexity and size as candidates for ultrascale execution, is clearly emerging. This means that future computational power will be used for advanced applications simulating increasingly complex phemonena, while reusing the existing single scale/physics models. Thus, it seems important to invest an effort into support for programmability enabling the porting of legacy codes into a single application that couples well-established sub-models. A variety of general or domain-specific well-established coupling platforms already exist, and it is timely to provide the standardization, which would support productivity for the application programmer. Full and proper understanding the algorithmic or mathematical model behind the coupling is essential for the standardization process, which is currently restricted to single coupled simulation codes. Mathematical standardization of model coupling would help to create more accurate and computationally efficient ultrascale applications. In summary, there seems to be a high potential for future ultrascale applications from the simulation problem point-of-view. A challenging task is the integration of the cross-cutting concerns, fault tolerance and reduction of the energy consumption. An open research question is whether these issues should be included into the simulation algorithm or being solved separately.

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)" and is co-authored by members of the Working group 6 on Applications of this action.

References

1. Ansys simulation software, http://www.ansys.com/.

2. Comsol multiphysics, http://www.comsol.com/.

3. Hypre, http://acts.nersc.gov/hypre/.

4. Linear algebra package (LAPACK), http://www.netlib.org/lapack/.

5. OpenFOAM, http://www.openfoam.com/.

6. Portable, Extensible Toolkit for Scientific Computation (PETSC), http://www.mcs.anl.gov/petsc/.

7. SUite of Nonlinear and DIfferential/ALgebraic equation Solvers (SUNDIALS), http://acts.nersc.gov/sundials/.

8. Trilinos ML, http://trilinos.org/packages/ml/.

9. Modeling Atmospheric and Oceanic Flows: Insights from Laboratory Experiments and Numerical Simulations. American Geophysical Union, Wiley, 2014.

10. D. Agrawal, S. Das, and A. El Abbadi. Big Data and Cloud Computing: Current State and Future Opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533. ACM, 2011. DOI: 10.1145/1951365.1951432.

11. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, April 2010. DOI: 10.1145/1721654.1721672.

12. S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright. Report on Exascale Computing. Technical report, ASCAC, 2010.

13. G.B. Berriman, G. Juve, J Vöckler, E. Deelman, and M. Rynge. The Application of Cloud Computing to Scientific Workflows: A Study of Cost and Performance. *Phil. Trans. R. Soc. A*, 371(1983), January 2013. Funding Acknowledgements: NSF OCI-0910812, NSF OCI-0943725, NASA NCC5-626, and Amazon Educational Grant.

14. J. Borgdorff, M. Ben Belgacem, C. Bona-Casas, L. Fazendiero, D. Groen, O. Hoenen, A. Mizeranschi, J.L. Suter, D. Coster, P.V. Coveney, W. Dubitzky, A.G. Hoekstra, P. Strand, and B. Chopard. Performance of distributed multiscale simulations. *Philospohical Transactions of the Royal Society*, A372:20130407, 2014.

15. J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, and A.G. Hoekstra. Foundations of distributed multiscale computing: formalization, specification and analysis. *Journal of Parallel and Distributed Computing*, 73:465–483, 2013.

16. J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P.V. Coveney, and A.G. Hoekstra. Distributed multiscale coupling with MUSCLE2, the Multiscale Coupling Library and Environment. *Journal of Computational Science*, 5:719–731, 2014.

17. P. Brunner and C.T. Simmons. HydroGeoSphere: A Fully Integrated, Physically Based Hydrological Model. *Ground Water*, 50(2):170–176, 2012.

18. Eisenstat S. C., H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 20:345–357, 1983.

19. F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience: 2014 update. *Supercomputing Frontiers and Innovations*, 1:5–28, 2014.

20. I. Chakroun, T. Vander Aa, B. De Fraine, P. Costanza, T. Haber, R. Wuyts, and W. Demeuter. ExaShark: A Scalable Hybrid Array Kit for Exascale Simulation. In *Proceedings of 23rd High Performance Computing Symposium (HPC 2015)*, 2015.

21. F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D. A Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

22. J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J.J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, and et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

23. P.V. Coveney, G. Giupponi, S. Jha, S. Manos, J. MacLaren, S.M. Pickles, R.S. Saskena, T. Soddermann, J.L. Suter, M. Thyveetil, and S.J. Zasada. Large scale computational science on federated international grids: The role of switched optical networks. *Future Generation Computer Systems*, 26:99–110, 2010.

24. W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

25. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

26. J. Dongarra and et al. The International Exascale Software Project Roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, February 2011.

27. A. Dorostkar, D. Lukarski, B. Lund, M. Neytcheva, Y. Notay, and P. Schmidt. Parallel performance study of block-preconditioned iterative methods on multicore computer systems. In *Proceedings of the Europar 2014 conference*. Springer LNCS, 2014.

28. D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17:20–27, 2013.

29. H.C. Elman, M.D. Mihajlović, and D.J. Silvester. Fast iterative solvers for buoyancy driven flow problems. *Journal of Computational Physics*, 230(10):3900–3914, 2011.

30. V. Escuder, R. Duran, and R. Rico. Analysis of x86 ISA Condition Codes Influence on Superscalar Execution. In Aluru S., Parashar M., Badrinath R., and Prasanna V.K., editors, *High Performance Computing - HiPC 2007: 14th International Conference, Goa - India*, 2007.

31. A. Fichtner, D. Giardini, A. Jackson, T. Nissen-Meyer, D. Peter, J. Robertsson, P. Tackley, L. Dalguer, D. Roten, O. Schenk, and M. Grote. HPC Roadmap. White paper, Solid Earth Dynamics, 2013.

32. M.J. Flynn, O. Mencer, V. Milutinović, G. Rakocević, P. Stenstrom, R. Trobec, and M. Valero. Moving from Petaflops to Petadata. *Commun. ACM*, 56(5):39–42, 2013.

33. H.J. Hendricks Franssen and W. Kinzelbach. Ensemble Kalman filtering versus sequential self-calibration for inverse modelling of dynamic groundwater flow systems. *Journal of Hydrology*, 365(3-4):261–274, 2009.

34. B. Fryxell, K. Olson, P. Ricker, F.X. Timmes, M. Zingale, D.Q. Lamb, P. MacNeice, R. Rosner, J.W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal*, 131:273–334, 2000.

35. S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

36. P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.

37. J. Gong, S. Markidis, M. Schliephake, E. Laure, D.S. Henningson, P. Schlatter, A. Peplinski, A. Hart, J. Doleschal, D. Henty, and P. F. Fischer. Nek5000 with OpenACC. In S. Markidis and E. Laure, editors, *Solving Software Challenges for Exascale*, pages 57–68, Heidelberg Dordrecht London New York, 2015. Springer LNCS 8759.

38. D. Groen, S.J. Zasada, and P.V. Coveney. Survey of Multiscale and Multiphysics Applications and Communities. *Computing in Science and Engineering*, 16:34–43, 2014.

39. P.K. Gunda, L. Ravindranath, C.A. Thekkath, Y. Yu, and L. Zhuang. Nectar: Automatic Management of Data and Computation in Datacenters. In *OSDI*, volume 10, pages 1–8, 2010.

40. M. Heil, A.L. Hazel, and J. Boyle. Solvers for large-displacement fluid-structure interaction problems: Segregated vs. monolithic approaches. *Computational Mechanics*, 43:91–101, 2008.

41. D.J. Hill. Nuclear Energy for the Future. *Nature Materials*, 7:680–682, 2008.

42. R. Hockney and J. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, 1988.

43. A. Hoekstra, B. Chopard, and P. Coveney. Multiscale modelling and simulation: a position paper. *Philosophical Transactions of the Royal Society*, A372:20130377, 2014.

44. A.G. Hoekstra, E. Lorenz, J.-L. Falcone, and B. Chopard. Towards a complex automata formalism for multiscale modeling. *International Journal for Multiscale Computational Engineering*, 5:491–502, 2007.

45. Q. Huang, C. Yang, K. Liu, J. Xia, C. Xu, J. Li, Z. Gui, M. Sun, and Z. Li. Evaluating Open-source Cloud Computing Solutions for Geosciences. *Comput. Geosci.*, 59:41–52, September 2013. DOI: 10.1016/j.cageo.2013.05.001.

46. S. Huang, S. Xiao, and W. Feng. On the energy efficiency of graphics processing units for scientific computing. In *IPDPS 2009. IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2009.

47. M. Huber, B. Gmeiner, U. Ruede, and B. Wohlmuth. Resilience for Exascale Enabled Multigrid Methods. *arXiv*, 1501.07400v1, 2015.

48. H.-T. Hwang. *Development of a parallel computational framework to solve flow and transport in integrated surface-subsurface hydrologic systems*. PhD thesis, University of Waterloo, Canada, 2012.

49. L. Hwang, T. Jordan, L. Kellogg, J. Tromp, and R. Willemann. Advancing Solid Earth System Science Through High-Performance Computing. Technical report, Lawrence Livermore National laboratory, University of California, 2014.

50. H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J.M. Patel, R. Ramakrishnan, and C. Shahabi. Big Data and Its Technical Challenges. *Commun. ACM*, 57(7):86–94, July 2014.

51. H. Johansen, D. Bernholdt, B. Collins, M. Heroux, R. Jacob, P. Jones, L.C. McInnes, J.D. Moulton, T. Ndousse-Fetter, D. Post, and W. Tang. Extreme-Scale Scientific Application Software Productivity: Harnessing the Full Capability of Extreme-Scale Computing. White paper, ASCR, 2013.

52. H. Johansen, L.C. McInnes, D.E. Bernholdt, J. Carver, M. Heroux, R. Hornung, P. Jones, B. Lucas, and A. Siegel. Software Productivity for Extreme-Scale Science Workshop Report. White paper, Workshop on Software Productivity for Extreme-scale Science, January 13-14, 2014, Rockville, MD, 2014.

53. G. Juve, E. Deelman, B. Berriman, B. Berman, and P. Maechling. An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2. *J. Grid Comput.*, 10(1):5–21, March 2012. DOI: 10.1007/s10723-012-9207-6.

54. J. Kappenman. Geomagnetic Storms and Their Impacts on the U.S. Power Grid. Technical Report Meta-R-319, Metatech Corporation, January 2010.

55. D. Keyes. Keynote Presentation: Adapting Upstream Applications to Extreme Scale. In *EAGE Workshop on High Performance Computing for Upstream*, 2014.

56. A. Khajeh-Saeed and J. Blair Perot. Computational Fluid Dynamics Simulations Using Many Graphics Processors. *Computing in Science & Engineering*, 14(3):10–19, 2012.

57. R. Klein, S. Vater, E. Paeschke, and D. Ruprecht. Multiple scales methods in meteorology. In *Asymptotic Methods in Fluid Mechanics: Survey and Recent Advances, CISM Courses and Lectures*, volume 523, pages 127–196. Springer, 2010.

58. M. Korch and T. Rauber. A Comparison of Task Pools for Dynamic Load Balancing of Irregular Algorithms. *Concurrency and Computation: Practice and Experience*, 16:1–47, 2004.

59. M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *7th Biennial Conference on Innovative Data Systems Research (CIDR'15)*. CIDR, 2015.

60. G. Kosec, M. Depolli, A. Rashkovska, and R. Trobec. Super linear speedup in a local parallel meshless solution of thermo-fluid problems. *Computers and Structures*, 133:30–38, 2014.

61. G. Kosec and B. Šarler. H-adaptive local radial basis function collocation meshless method. *CMC: Computers, Materials, and Continua*, 26(3):227–253, 2011.

62. G. Kosec and R Trobec. Simulation of semiconductor devices with a local numerical approach. *Engineering Analysis with Boundary Elements*, 50:69–75, 2015.

63. G. Kosec and P. Zinterhof. Local strong form meshless method on multiple Graphics Processing Units. *CMES: Computer Modeling in Engineering and Sciences*, 91(5):377–396, 2013.

64. N.A. Krall and A.W. Trivelpiece. *Principles of plasma physics*. International Series in Pure and Applied Physics. McGraw-Hill, 1973.

65. P. Kropf, E. Schiller, P. Brunner, O. Schilling, D. Hunkeler, and A. Lapin. Wireless Mesh Networks and Cloud Computing for Real Time Environmental Simulations. In *Recent Advances in Information and Communication Technology - Proceedings of the 10th International Conference on Computing and Information Technology, IC2IT 2014, Angsana Laguna, Phuket, Thailand, 8-9 May, 2014*, number 265 in Advances in Intelligent Systems and Computing, pages 1–11. Springer, 2014.

66. J. L. Lang and G. Rünger. An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration. *Journal of Parallel and Distributed Computing*, 74(9):2884–2897, 2014.

67. A. Lapin, E. Schiller, P. Kropf, O. Schilling, P. Brunner, A. Jamaković-Kapić, T. Braun, and S. Maffioletti. Real-time envoronmental monitoring for cloud-based hydrogeological modelling with Hydrogeosphere. In *High Performance Computing and Communications conference HPCC*, Paris, 2014. IEEE.

68. M. Guest (lead author). The Scientific Case for High Performance Computing in Europe 2012-2020. Technical report, FP7 Project PRACE RI-261557, 2014.

69. A. Lucia. Multi-Scale Methods and Complex Processes: A Survey and Look Ahead. *Computers and Chemical Engineering*, 34:1467–1475, 2010.

70. S. Markidis, J. Gong, M. Schliephake, E. Laure, A. Hart, D. Henty, K. Heisey, and P. F. Fischer. OpenACC Acceleration of Nek5000, Spectral Element Code. *International Journal of High Performance Computing Applications (accepted)*, 2015.

71. A. Melnyk and V. Melnyk. *Personal Supercomputers: Architecture, Design, Application.* Lviv Polytechnic National University Publishing, 2013.

72. M.T. Fisher M.L. Abbott. *Scalability Rules: 50 Principles for Scaling Web Sites.* Addison-Westey, 2011.

73. R.L. Muddle, M.D. Mihajlović, and M. Heil. An efficient Preconditioner for Monolithically-Coupled Large-Displacement Fluid-Structure Interaction Problems with Pseudo-Solid Mesh Updates. *Journal of Computational Physics*, 231(21):7315–7334, 2012.

74. S. Margenov N. Kosturski and Y. Vutov. Balancing the Communications and Computations in Parallel FEM Simulations on Unstructured Grids. In K. Karczewski R. Wyrzykowski, J. Dongara and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, pages 211–220, Heidelberg Dordrecht London New York, 2012. Springer LNCS 7204.

75. S. Margenov N. Kosturski and Y. Vutov. Computer Simulation of RF Liver Ablation on an MRI Scan Data. In M.D. Todorov, editor, *Application of Mathematics in Technical and Natural Sciences*, pages 120–126, Melville, NY, USA, 2012. AIP Conf. Proc. 1487.

76. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

77. OpenACC. `http://www.openacc.org`.

78. S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in action.* Manning, 2011.

79. S. Margenov P. Arbenz and Y. Vutov. Parallel MIC(0) preconditioning of 3D elliptic problems discretized by Rannacher-Turek finite elements. *Computers and Mathematics with Applications*, 55(10):2197–2211, 2008.

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

81. 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.

82. T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems, Second edition.* Springer, 2013.

83. 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.

84. 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.

85. J. Rehr, F. Vila, J. Gardner, L. Svec, and M. Prange. Scientific Computing in the Cloud. *Computing in Science and Engineering*, pages 34–43, 2010.

86. K. Rojek, M. Ciznicki, B. Rosa, P. Kopta, M. Kulczewski, K. Kurowski, Z. Piotrowski, L. Szustak, D. Wojcik, and R. Wyrzykowski. Adaptation of fluid model EULAG to graphics processing unit architecture. *Concurrency and Computation: Practice and Experience*, 27(4):937–957, 2014. DOI: 10.1002/cpe.3417.

87. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

88. S. Schnell, R. Grima, and P.K. Maini. Multiscale Modeling in Biology. *American Scientist*, 95:134–142, 2007.

89. Y. Shimizu and M. Takashi. Effect of topology on parallel computing for optimizing large scale logistics through binary PSO. In Lockhart Bogle I.D. and Fairweather M., editors, *Computer Aided Chemical Engineering Volume 30, Pages 1-1435 (2012) - 22 European Symposium on Computer Aided Process Engineering*, 2012.

90. K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010.

91. C.A. Smethurst, D.J. Silvester, and M.D. Mihajlović. Unstructured finite element method for the solution of the Boussinesq problem in three dimensions. *International Journal for Numerical Methods in Fluids*, 73(9):791–812, 2013.

92. Science Staff. Special Collection: Dealing with Data. *Science*, 331(6018), 2011.

93. J. Suter, D. Groen, L. Kabalana, and P.V. Coveney. Distributed Multiscale Simulations of Clay-Polymer Nanocomposites. In *MRS Proceedings*, volume 1470. Cambridge University Press, 2012.

94. L. Szustak, K. Rojek, T. Olas, L. Kuczynski, K. Halbiniak, and P. Gepner. Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. *Scientific Programming*, page 642705, 2015.

95. S. Tansley and K.M. Tolle. *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research, 2009.

96. A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

97. Top500. *Top500 supercomputers site*. Available on: http://www.top500.org, (accessed August 2014).

98. R. Trobec. Two-dimensional Regular d-meshes. *Parallel Comput.*, 26(13-14):1945–1953, 2000.

99. R. Trobec. *Advances in the MLPG meshless methods*, chapter Experimental analysis of methods for moving least squares support determination, pages 307–358. Duluth: Tech Science Press, 2009.

100. C. Vecchiola, S. Pandey, and R. Buyya. High-Performance Cloud Computing: A View of Scientific Applications. In *Proc. of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN '09, pages 4–16. IEEE Computer Society, 2009. DOI: 10.1109/I-SPAN.2009.150.

101. W.M. Washington and C.L. Parkinson. *Introduction To Three-dimensional Climate Modeling*. University Science Books, California, 2005.

102. W. Xue, Ch. Yang, H. Fu, Y. Xu, J. Liao, L. Gan, Y. Lu, R. Ranjan, and L. Wang. Ultra-scalable CPU-MIC Acceleration of Mesoscale Atmospheric Modeling on Tianhe-2. *IEEE Transaction on Computers*, 8, 2014. DOI: 10.1109/TC.2014.2366754.

103. C. Yang, M. Goodchild, Q. Huang, D. Nebert, R. Raskin, Y. Xu, M. Bambacus, and D. Fay. Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4):305–329, July 2011.

104. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

105. S.J. Zasada and P.V. Coveney. Virtualizing access to scientific applications with the Application Hosting Environment. *Computer Physics Communications*, 180:2513–2525, 2009.

Received February 27, 2015.