# Supercomputing Frontiers and Innovations

**2014, Vol. 1, No. 2.**

## Scope

- *Parallel and distributed computing technologies.* Parallel programming languages and libraries. Methods, algorithms and tools for analysis, debugging and optimization of supercomputing applications. Co-design concepts.
- *Next-generation supercomputer architectures.* Advanced supercomputer architectures. Reconfigurable and hybrid supercomputing systems. Accelerator based supercomputing systems. Modeling and development of new high-performance computing hardware.
- *Supercomputing Applications.* Solving of computationally intensive problems using supercomputers in computational mathematics, computational physics, chemistry, hydro-gas-dynamics and heat transfer, nonlinear transient problems in mechanics, bioinformatics, engineering, nanotechnology, climate, ecology, cryptography, and other prospective areas.
- *Visualization.* Parallel methods and algorithms of visualizing of computational experiments.
- *Management, administration and monitoring of supercomputing systems.* Methods, algorithms and tools of management, technical support and monitoring of highly parallel supercomputing systems.
- *Parallel System Software and Tools.* Tools, performance evaluation, development tools, run-time systems, libraries.
- *Parallel database systems.* Development of parallel database management systems for multiprocessor (multicore) computing systems. Methods and algorithms of parallel database processing. High performance data mining.
- *Supercomputing Education.* Practice and experience of supercomputing education. Innovative approaches on teaching of HPC and parallel computing technologies.

## Editorial Board

# Contents

**DOI: 10.14529/jsfi140201**

# Scalability prediction for fundamental performance factors

***Claudia Rosas*[1], *Judit Giménez*[12], *Jesús Labarta*[12]**

Inferring the expected performance for parallel applications is getting harder than ever; applications need to be modeled for restricted or nonexistent systems and performance analysts are required to identify and extrapolate their behavior using only the available resources. Prediction models can be based on detailed knowledge of the application algorithms or on blindly trying to extrapolate measurements from existing architectures and codes. This paper describes the work done to define an intermediate methodology where the combination of (a) the essential knowledge about fundamental factors in parallel codes, and (b) detailed analysis of the application behavior at low core counts on current platforms, guides the modeling efforts to estimate behavior at very large core counts. Our methodology integrates the use of several components like instrumentation package, visualization tools, simulators, analytical models and very high level information from the application running on systems in production to build a performance model.

*Keywords: parallel efficiency, curve-fitting, exascale computing, analysis and prediction.*

## Introduction

Within the race toward exascale computing, to infer the scaling capacity of current parallel codes has become essential [1]. If we are able to identify primary factors that define the efficiency, we can use them to predict the scalability of the code. Systems and codes are getting more complex and their performance analysis may result costly in time. At the same time, validation on current non-existent machines is not an option, and in consequence, scientists must take advantage of already known techniques and tools to overcome these restrictions. Tools are helpful to identify, in a short time, the primary factors of real parallel codes executed in the available machines. Then, collected information can be used to outline potential restrictions on future computational systems [2].

Different philosophies have been followed to perform prediction studies in the past. The approaches range from a vision based on first principles on one side to blind fitting of metrics and extrapolation on the other. An effort to investigate the performance of MPI applications at large core counts uses parallel discrete event simulations to run the application in a controlled environment [3]. Most of them demand from specific models or abstractions of the parallel code (and the system) [4], [5] or massive fittings of time-based metrics to predict performance of specific functions [6]. However, there is low information about the insights of the real underlying cause of the inefficiencies, and the knowledge about the influence of different architectural characteristics can be useful to improve the code.

This work further develops the modeling and extrapolation tasks initially presented in [7], broadening the scope and scale. We consider that the components that represent essential features of the program and their evolution may be related to primary models of parallelism such as Amdahl's law. This proposal starts by capturing detailed data from traces of very few runs of the parallel code at low core counts in machines that are in production, i.e. without additional tunings for exclusivity. From the traces, significant performance components such as load balance and transfer can be measured, fitted and extrapolated at large core counts.

Our method relies on the detailed preprocessing of available traces to determine appropriate sections with minimal noise perturbation. Two arguments sustain the use of traces. First, having

[1]Barcelona Supercomputing Center (BSC), Barcelona, Spain
[2]Technical University of Catalonia (UPC), Barcelona, Spain

few points to fit, the blind use of functions with many parameters would lead to undetermined systems with many possible solutions in the explored core count range. These solutions may have huge differences in performance when extrapolating for large core counts. Second, it is our belief that low core count runs provide enough information on the fundamental behavior of parallel code, and can easily complement existent time profiles reports of the different routines.

We provide an automatic framework to produce the models and reports, automatizing the error-prone task of collection and processing the measurements to increase the availability of the analyst for observation, interpretation and let him/her focus on the factors or significant interest. The analyst can perform additional measurements, or what-if predictions, using analysis and simulation tools (Paraver, Dimemas, Clustering, etc.) to find out outliers and for better understanding of the fundamental factors. Collected data is used to model the expected performance of an application in that specific machine for larger core counts.

The use of fundamental factors is highly informative for developers and can guide optimization efforts in the most productive direction. For example, for an application whose main problem is load balance it is highly non-productive to spend time and energy re-factoring it using non-blocking MPI calls, even if the standard time profile indicates that a large fraction of the time is consumed by MPI. Compared to the first principles approach, our method does not require prior knowledge from the analyst on the code nor a specific modeling effort for each new one.

This work combines a set of steps that can be performed semi-automatically to reduce the total time for data extraction and processing. The main stages are:

- **Identify Structure:** starting with traces from an instrumented execution of the parallel application at different core counts identify relevant sections to analyze and extrapolate;
- **Phase Performance Analysis:** compute fundamental efficiency factors (load balance, serialization, transfer) for each identified region;
- **Scalability Prediction:** extrapolate the fundamental efficiency factors computed at low core counts to infer their evolution at large core counts.

The main contributions of this work are:

- A methodology to extrapolate the efficiency computed at low core counts to large core counts in the same platform;
- A validation of the methodology on 4 cases corresponding to different applications, platforms and strong or weak scaling runs;
- An extension of the methodology to predict the impact at large core counts of architectural or system improvements (in particular OS and network noise elimination).

The rest of this paper is organized as follows. Section 1 outlines our approach and describes the underlying prediction model in detail. Section 2 presents the experiments performed to evaluate the effectiveness of our method. Related work is discussed in Section 3, and final remarks and further steps are in Section 4.

## 1. Description of the methodology

### 1.1. Identify Structure

Our method begins with a trace for a number of MPI ranks, obtained from executing an instrumented parallel code or by simulating the execution on a target machine. Then, we visualize the trace to generate clean cuts of the representative regions or phases from the temporal structure of the execution. A main phase suggests regions of computation and/or communication that may show different behaviors or that are independent between them, e.g. separating long computational regions from communication intensive phases. In this step, we can measure duration of the computation, or number of MPI calls to highlight additional details in the interval that may suggest additional phases (other regions that may become interesting to analyze). To reduce potential noise introduced by the machine during the execution, the output of this step is one or more cuts of the cleanest regions of the trace for each core count. A region may contain one or more iterations, and it represents a part with potential performance bottlenecks or that may have significant impact to the execution.

## 1.2. Phase Performance Analysis

To report a quantitative summary of the performance of identified phases, all the trace cuts will be processed using our automatic framework. The model decomposes the parallel efficiency metric as a product of factors with normalized values between 0 (very bad) and 1 (perfect) [7]. The factors correspond to fundamental behavioral aspects of parallel codes and are load balance, serialization and transfer.

- **Parallel Efficiency** reflects the performance obtained from executing in parallel the code. It is expressed as the product of the three fundamental factors: Load Balance, Serialization and Transfer. Efficiency of the parallelization is presented in expression 1.

$$\eta_{\|} = LB * Ser * Trf \tag{1}$$

- **Load Balance efficiency** reflects the potential efficiency loss caused by imbalance in the total computation time by each process. It is measured as the ratio between the average computing time ($\sum_i t_i/P$) and the maximum computation time ($max(t_i)$) from all the processes $i = \{0, ..., P\}$, as shown in expression 2.

$$LB = \frac{\sum_i t_i}{P * max(t_i)} \tag{2}$$

- **Serialization efficiency** reflects the inefficiency caused by dependencies in the code. It is measured by simulating an instantaneous communications (ideal) scenario using Dimemas and collecting the maximum efficiency achieved by a single process ($ideal(eff_i)$ in expression 3).

$$Ser = Max(ideal(eff_i)) \tag{3}$$

- **Transfer efficiency** reflects the performance loss caused by actual data transfer. It is caused by the MPI overhead plus the interconnection network noise and can be measured in expression 4, as the maximum efficiency achieved by a single process in the real execution ($Comm_{eff}$) and the inefficiency from serialization ($Ser$).

$$Trf = \frac{Comm_{eff}}{Ser} \tag{4}$$

In this step, we detail the performance of the application based on observations from collected measurements. The model provides a general view of the inefficiencies in the code and the relative importance of the performance factors. As a first advantage of our proposal, this type of model provides useful information to suggest the appropriate strategy to improve the performance in the application. From a broader point of view, the use of fundamental factors facilitates a unified modeling approach for strong and weak scaling scenarios, thus providing a fast and approximate approach to infer the evolution of the parallel code. Expressing performance in terms of parallel efficiency brings flexibility when selecting the number of iterations for each cut. In a steady iterative parallel code, a few iterations will typically be sufficient to represent its behavior. Reporting efficiency instead of absolute time makes the metric essentially insensitive to the number of iterations, eliminating the need to ensure that exactly the same number of iterations is analyzed for all core counts. The analysis tools (Paraver) provide mechanisms (for example analyzing histograms of the duration of computation, the cycles counter, among others) to identify regions of the traces that may be significantly perturbed by noise. Sufficiently clean cuts of the traces can thus be obtained at different core counts without the constraint of requiring them to be of exactly the same number of iterations. The advantage of this approach is that the analysis can be focused on regions with less perturbations.

## 1.3. Scalability Prediction

To infer the behavior of phases of an application in a specific platform at large core counts, our extrapolation approach fits the components of the multiplicative model presented in expression 1. The model is fed with data collected from traces obtained from several, not many runs using low core counts in the same target platform. We argue that the results of the analysis phase using a small number of executions for low core counts, combined with the fundamental underlying system behavior can help us identify the specific scalability model for each component of the multiplicative model. By independently extrapolating the individual components of this model we can observe their evolution; how relevant are they to the overall performance of the parallel code; potential variations of significant factor as core counts increase, and infer a potential performance for a core count that has not being executed before.

The original prediction strategy proposed in [7] observed that an extrapolation based on model factors led to more accurate predictions that extrapolate the overall efficiency. Nevertheless, these observations were based on projections for limited increases in the core counts. We aim at studying the impacts for much larger core counts. In an intent to use linear models, results where unacceptable because they get negative with large values of cores.

The basic default model is Amdahl's law formulation. This is a general and approximate model that represents a first approach to describe the effect of non-parallel regions, where inefficiencies are caused by an activity that can not be executed concurrently with other activities. This may be caused by computations being serial, but can also constitute an abstract model of other serialization behavior such as contentions in the network resources. The formulation of such an Amdahl based approach in terms of the efficiency model is presented in expression 5.

$$Amdahl_{fit} = \frac{amdahl_0}{f_{amdahl} + (1 - f_{amdahl}) * P} \tag{5}$$

Other possible pattern of concurrency corresponds to pipelined computation. Expression 6 models a behavior of alternating segments of totally parallel computation with perfectly pipelined segments.

$$Pipe_{fit} = \frac{pipe_0 * P}{(1 - f_{pipe}) + f_{pipe} * (2 * P - 1)} \tag{6}$$

Moreover, additional features of the program may suggest a different fitting for each factor, e.g. a constant behavior when there are no changes in efficiency when scaling may indicate that a factor can be treated as a constant value at very large scale.

## 2. Experimental Evaluation

For the experimentation, we use three applications from the CORAL suite: HACC, Nekbone, and AMG2013; and the CFD application AVBP. The Hardware Accelerated Cosmology Code (HACC) [8] parallel benchmark is a flexible framework that uses N-body techniques to simulate the evolution of the Universe from its early times to today and to advance our understanding of dark energy and dark matter. The Nekbone is a proxy-app from the Nek5000 software [9], which executes computationally intense linear solvers. Nekbone has been created to be easily adapted to different platforms, communication structures, and scalability studies. AMG2013 [10] is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. AMG2013 is a highly synchronous code and parallelism is achieved by domain decomposition. Parallel efficiency is largely determined by the size of data chunks in the decomposition, and the speed of communications and computations on the machine. AVBP [11] solves the three-dimensional compressible Navier-Stokes on unstructured and hybrid grids. AVBP includes integrated parallel domain partition and data reordering tools. The scaling: weak or strong, and the number of processes used in the runs of each application are summarized in tab. 1.

CORAL applications have been executed in MareNostrum III, a machine based on Intel Xeon E5 processors, iDataPlex Compute Racks and Infiniband network. Traces from AVBP have been obtained in Juropa, a supercomputer based on Intel Xeon X5570 processors, Sun constellation systems and Infiniband network. The machines operated in normal production using fully populated nodes, i.e. there is non dedicated network and some potential noise from OS may be introduced in the runs.

### 2.1. Identify Structure

To identify the structure of the application we visualized the traces for each execution with various ranks. A manual process that leads to obtain clean cuts of interesting phases within the execution. In general, applications present an iterative structure, and a clean cut is a region with low or none perturbations from the system. The identification process is based on the observation of metrics of the execution to limit the phases. Metrics as the duration of computational burst, cycles per microseconds, or the behavior of MPI calls (colored areas in the images shown below, where each color represents a type of MPI call) result very useful in this task.

In a trace of one iteration of HACC we can observe a large computationally intensive phase of $\approx 250s$. This phase, presented at the left side of fig. 1, shows low communications and some

**Table 1.** Applications used in for the experiments

| Strong Scaling | Ranks | Weak Scaling | Ranks |
|---|---|---|---|
| HACC | 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 | AMG2013 | 32, 64, 96, 128, 192, 256, 384 |
| Nekbone | 2, 4, 8, 16, 32, 64, 128, 256, 512 | AVBP | 16, 32, 64, 96, 128,192, 256, 520, 768, 1024,1040, 1280, 1536 |

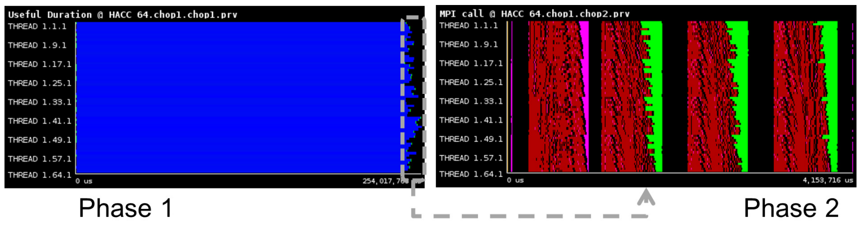
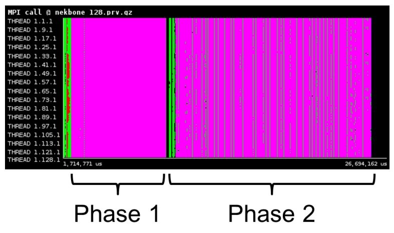
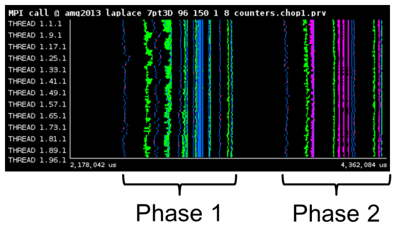

Useful Duration @ HACC 64.chop1.chop1.prv
MPI call @ HACC 64.chop1.chop2.prv
Phase 1
Phase 2

**Figure 1.** Phases in one iteration of HACC code




MPI call @ amg2013 laplace 7pt3D 96 150 1 8 counters.chop1.prv
Phase 1
Phase 2

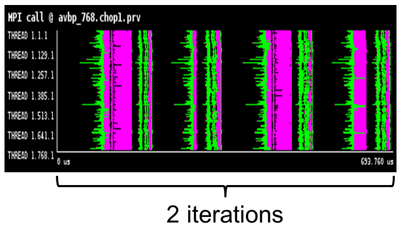


**Figure 2.** Phases in Nekbone **Figure 3.** Phases in AMG2013 **Figure 4.** Iteration in AVBP

imbalances at the end. The iteration has also a small communication phase ($\approx 4s$) shown at the right side of the figure. Inside this phase, we could identify sub phases of finer grain, yet for the scope of our analysis the main two phases are enough.

In Nekbone we identify two phases with an iterative structure, presented in fig. 2. To avoid the impact of potential noise of the system, we choose iterations with less variations between the bursts. At the same time, the number of iterations chosen may variate when increasing the number of core counts. A decision that does not affect the effectiveness of our method as it measures the parallel efficiency of the total region.

For AMG2013, we focus our analyses in the regions with greater presence of MPI calls, differentiating one phase dominated by point-to-point communications from a second phase that presents more collectives calls. These phases, shown in fig. 3, are consecutive within one iteration of the application.

Finally, we took two outer iterations of AVBP to test the sensitivity of our method using a coarser grain. In the cut, there are regions of compute with some imbalances (dark areas in fig. 4), which are followed by point-to-point communications and a subsequent synchronization step before starting new computations.

## 2.2. Phase Performance Analysis

We apply the automatic framework for basic analysis to each phase identified to extract the main performance metrics and factors of the performance model. Below we summarize the outputs of the efficiency associated to the fundamental factors for all the applications. Values range from 0 to 1 being those closer to 1 which present greater efficiency.

For phases 1 and 2 of HACC, we can observe the evolution of the performance factors when changing the number of ranks. In this parallel code, the evolution of all three factors in the computational phase (fig. 5) describes a highly efficient region with almost no imbalance or contention. In the analysis of the communication phase (fig. 6), Transfer ($Trf$) followed by Serialization ($Ser$) are dominant factors in the overall performance loss. After 256 ranks, Transfer reports an efficiency of 0.6 and Serialization reports around 0.8, thus suggesting the presence of some contention or delay in the communications between processes. The Load Balance in fig. 6 seems to remain steady for all number of ranks. To better understand the fundamental behavior of this metric, we compute the load balance across processes in terms of total number of instructions they execute rather than the time they take. The result is perfect load balancing across all the core counts. In addition, after analyzing the trace, processors seem to perform computations in stages inside phase 2 (similar to a pipeline decomposition), in consequence, we decide to model the serialization factor by expression 6.

The analysis from the two phases of Nekbone reports regions with an efficiency of above 0.8 (fig. 7 and fig. 8). Load balance shows a slight drop after 32 processes that later seems to stabilize. An analysis of the cycles per second metric in the traces confirms the existence of a small level of preemption, thus suggesting that the reduction observed in this factor derives from noise in the system.

The analysis from the phases of AMG2013, shown fig. 9 and fig. 10, suggest a performance loss mainly dominated by load imbalances. Both phases present this behavior and by checking in the trace files, we verified that this is due to preemptions introduced by noise. In addition, for larger core counts transfer efficiency in phase 1 shows greater degradations. After analyzing the traces, the inefficiencies are caused by contention in point-to-point communications.

The AVBP executions reported values with an initial steady behavior up to 384 processes (left side in fig. 11). When increasing the number of cores abrupt variations appear. The trace files reported that these variations are caused by unexpected noise in the run, further discussion of this subject is in section 2.4.

## 2.3. Scalability prediction

### *2.3.1. Model Fitting*

From the collected measurements and performance observed from the analysis of the traces, we now want to generate a model to infer the expected performance at large core counts. Contention can be one of the main causes of performance loss, and Amdahl's Law reflects the presence of this type of restrictions when increasing core counts. In consequence, we propose to fit all the fundamental factors using the expression 5 by default. Metrics were fitted using the least square regression model taking as a reference current measurements from executions with low core counts. The fit considers the dependent variable to be inside the range $[0.0, 1.0]$. In the results shown below, we use expression 5 to infer the expected parallel efficiency from all the fundamentals factors for Nekbone, AVBP, AMG2013, and for Transfer and Load Balance in HACC. For almost all the applications, Amdahl's function generates a curve that follows the closest gradient to the measured points (presented from fig. 12 to fig. 17). In addition, it provides a reasonable fitting with small influence of the number of points used.

### *2.3.2. Validation of Results*

In this section, we compare the efficiencies predicted for a machine from runs using small core counts with the values collected from real traces using at least runs 3 times larger than the number of processes used for prediction. In the available machines, we have obtained traces up to 1536 processes (AVBP in Juropa) and 4096 processes (HACC in MareNostrum III). Predicted efficiencies and their associated relative error (inside parentheses) are summarized in the tables below. Prediction results are from the phase of the application that shows the bigger relative

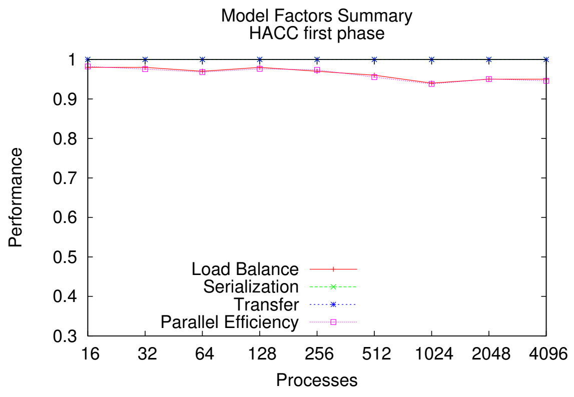


**Figure 5.** Model factors phase 1 HACC

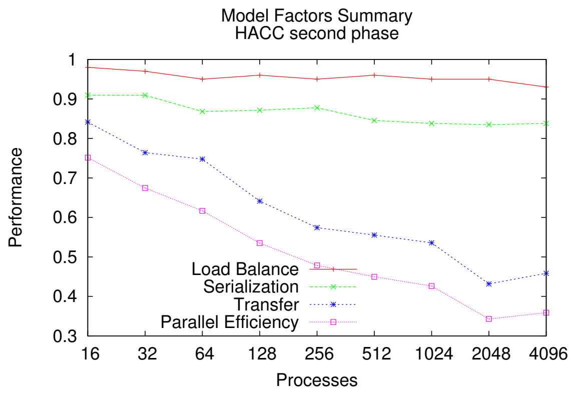


**Figure 6.** Model factors phase 2 HACC

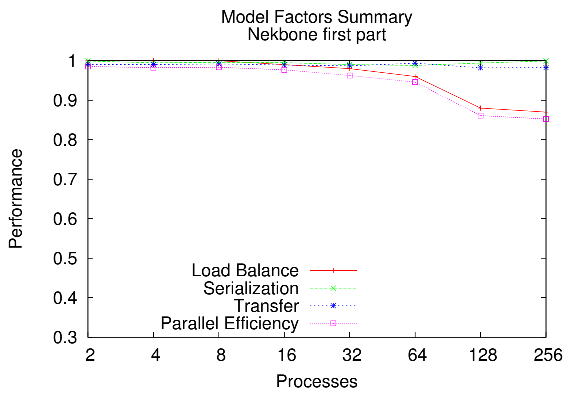


**Figure 7.** Model factors first part Nekbone

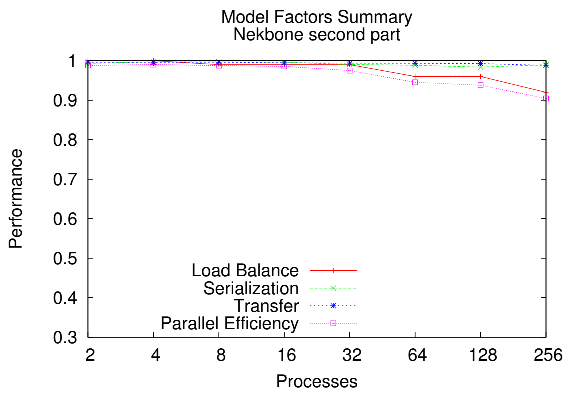


**Figure 8.** Model factors second part Nekbone

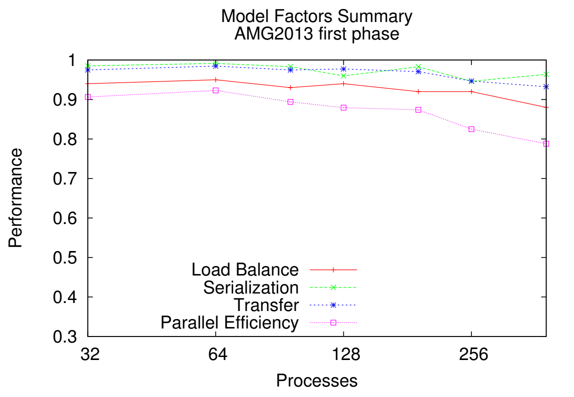


**Figure 9.** Model factors phase 1 AMG2013

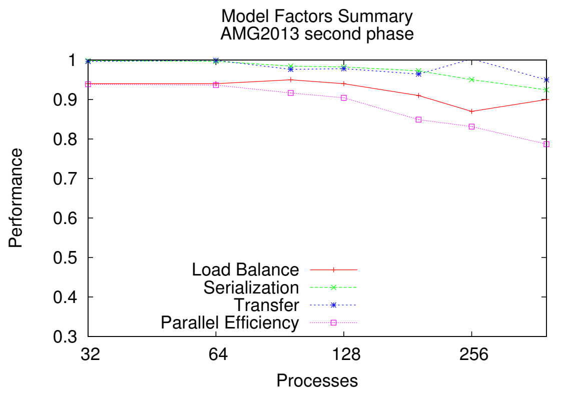


**Figure 10.** Model factors phase 2 AMG2013

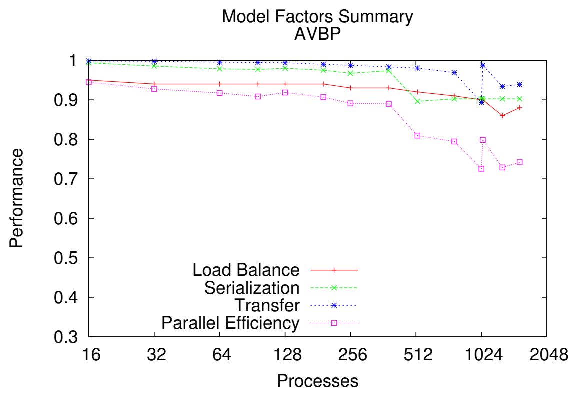


**Figure 11.** Model factors from two iterations of AVBP

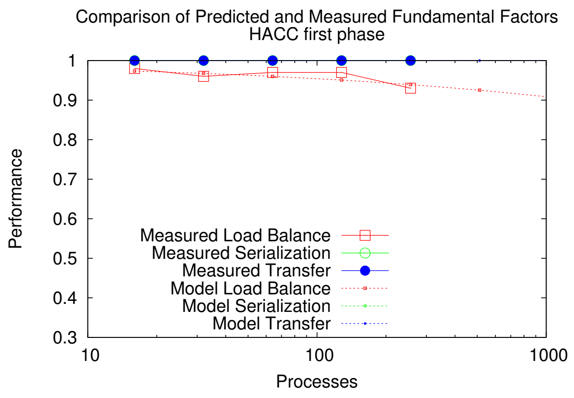


**Figure 12.** Fitting phase 1 HACC

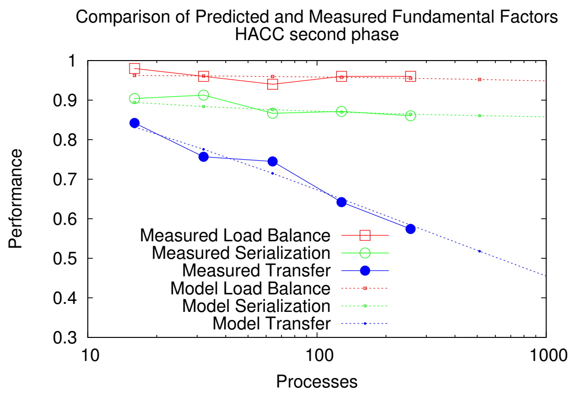


**Figure 13.** Fitting phase 2 HACC

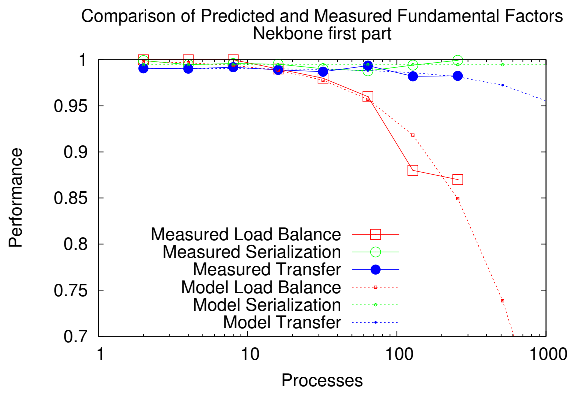


**Figure 14.** Fitting first part Nekbone

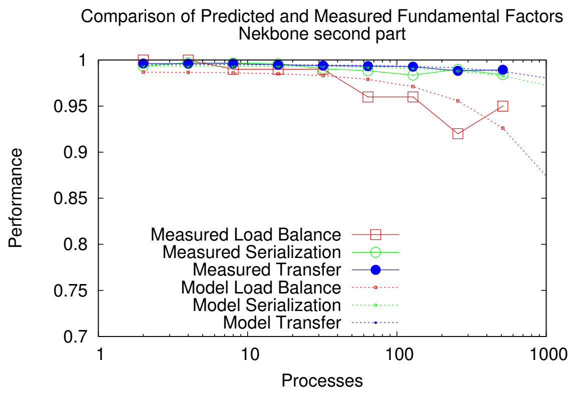


**Figure 15.** Fitting second part Nekbone

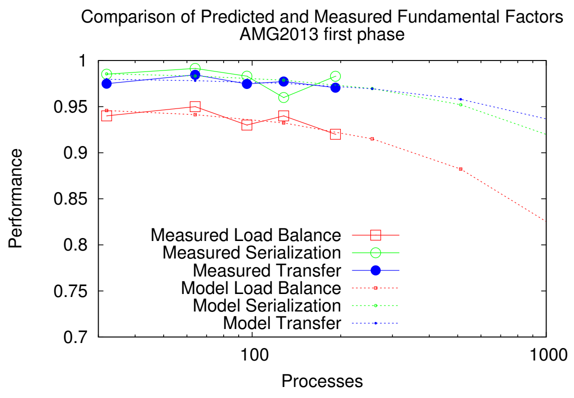


**Figure 16.** Fitting phase 1 AMG2013

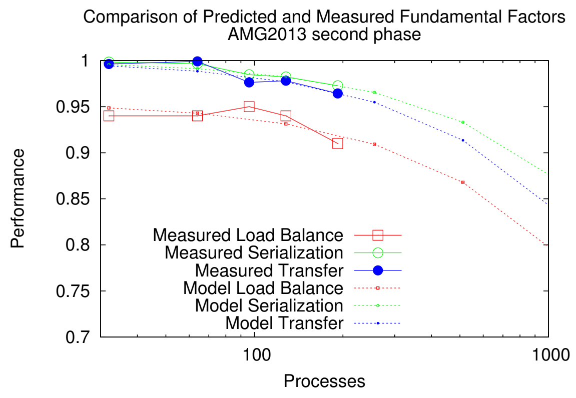


**Figure 17.** Fitting phase 2 AMG2013

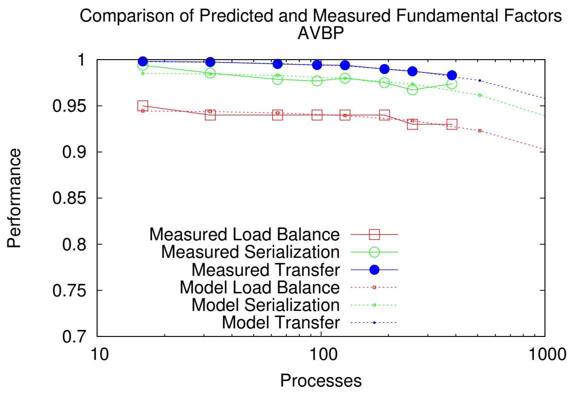


**Figure 18.** Fitting from AVBP

error. The error for each fundamental factor shows how optimistic (+) or pessimistic (-) is our prediction.

For HACC, phase 2 with up to 256 processes were used to project the efficiency for 512, 1024, 2048 and 4096. Results shown in tab. 2, show a low relative error in the predictions of load balance and serialization. Even for transfer, which reports variable errors, we are able to predict the expected efficiency for a number of processes 20 times larger loosing around a 25% of accuracy. With Nekbone, we use traces up to 128 processes to project the efficiency of 256 and 512. Efficiencies reported in tab. 3 show a relative error of less than a 0.1%. We used traces up to 128 processes to project the efficiency for AMG2013. Results for predicted efficiencies for 256 and 384 processes are reported in tab. 4. These results show a relative error of less than 10%. For AVBP, we used executions with up to 384 processes to project the expected efficiency for 520, 768, 1024, and 1536 presented in tab. 5, and the differences between the expected and the real measurement are not greater than 18%.

### *2.3.3. Projection for large core counts*

From collected efficiencies of each fundamental factor our framework extrapolates the expected total parallel efficiency for up to $10^6$ cores. In weak scaling, phase 1 of HACC shows in fig. 19 a constant behavior of Serialization and Transfer, and a low degradation of Load Bal-

**Table 2.** Predicted efficiency and relative error for HACC (phase 2), extrapolated from runs using 16 to 256 cores

| Ranks | Load Balance | Serialization | Transfer | Parallel Efficiency |
|---|---|---|---|---|
| 512 | 0.952(−0.82)% | 0.860(+1.81)% | 0.517(−6.78)% | 0.424(−5.61)% |
| 1024 | 0.948(−0.17)% | 0.857(+2.34)% | 0.452(−15.47)% | 0.368(−13.64)% |
| 2048 | 0.943(−0.68)% | 0.855(+2.45)% | 0.391(−9.39)% | 0.315(−7.80)% |
| 4096 | 0.937(+0.82)% | 0.853(+1.83)% | 0.333(−27.19)% | 0.267(−25.25)% |

**Table 3.** Predicted efficiency and relative error for Nekbone (phase 2), extrapolated from runs using 2 to 128 cores

| Ranks | Load Balance | Serialization | Transfer | Parallel Efficiency |
|---|---|---|---|---|
| 256 | 0.972(−0.003)% | 0.985(+0.002)% | 0.992(−0.001)% | 0.950(−0.002)% |
| 512 | 0.950(−0.007)% | 0.977(+0.004)% | 0.989(+0.001)% | 0.919(−0.001)% |

**Table 4.** Predicted efficiency and relative error for AMG2013 (phase 2), extrapolated from runs using 32 to 192 cores

| Ranks | Load Balance | Serialization | Transfer | Parallel Efficiency |
|---|---|---|---|---|
| 256 | 0.921(−0.16)% | 0.937(+0.95)% | 0.941(+0.55)% | 0.813(+1.45)% |
| 384 | 0.908(−3.08)% | 0.908(+6.05)% | 0.914(+1.94)% | 0.754(+4.42)% |

**Table 5.** Predicted efficiency and relative error for AVBP, extrapolated from runs using 16 to 256 cores

| Ranks | Load Balance | Serialization | Transfer | Parallel Efficiency |
|---|---|---|---|---|
| 520 | 0.907(−1.42)% | 0.925(+3.23)% | 0.973(−0.70)% | 0.816(+0.93)% |
| 768 | 0.886(−2.54)% | 0.893(−1.00)% | 0.961(−0.82)% | 0.761(−4.15)% |
| 1024 | 0.867(−3.66)% | 0.862(−4.40)% | 0.948(+6.21)% | 0.709(−2.16)% |
| 1536 | 0.830(−5.68)% | 0.807(−10.54)% | 0.925(−1.45)% | 0.620(−16.46)% |

ance, thus resulting in a region that will scale relatively well at very large core counts. For phase 2 (fig. 20), at 2k cores the parallel efficiency is below 0.4, thus suggesting that the problem will be the communication contention, while a pipelined serialization structure may represent a secondary but less relevant cause of inefficiency.

Phase 1 of Nekbone shows a total parallel efficiency equal to 0.4 at 2k cores (fig. 21). An efficiency loss mainly dominated by load imbalances. Although this suggest uneven distribution of the work, the histogram of instructions reveals that there is no computational imbalance. The imbalance appears because of IPC differences between process. This behavior would need deeper analysis. At the same point, parallel efficiency for phase 2 is also dominated by load imbalances (fig. 22), however, this phase shows a reasonable efficiency (0.7) for 2k processes and reaches the efficiency of 0.4 at a larger number of cores than phase 1 (around 8k).

In strong scaling, AMG2013 shows similar expected behaviors for both phases (fig. 23 and fig. 24). Phase 1 is mainly dominated by load imbalances, and shows a parallel efficiency of 0.5 at 2k cores. On the contrary, in phase 2 due to the coupling between all the fundamental factors, a lower efficiency (0.4) for the same number of cores is reported.

Finally, the extrapolation of parallel efficiency for AVBP shows a performance loss tightly coupled to all 3 factors. At 2k cores the total efficiency predicted is 0.7, and it is worth mentioning that the problem size provided by the developers of this application was expected to scale well up to 1k cores. In addition, for AVBP, and unlike the rest of the applications, there is a shift between load balance and serialization as the influential factor in loss of efficiency after 10k (fig. 25). From this observation, the expected dominant factor for a given number of cores may not result the same when scaling the application for larger core counts. It suggests using different optimization techniques depending on the scale at which the application will be executed.

## 2.4. Additional enhancements: Noise reduction

Predictions until now were based on runs of an application in current available machines. The noise introduced by the system or by the MPI engine is extrapolated and we can thus infer how the application will behave when scaling the platform. Some interesting questions emerge: Which is the impact that some characteristics of the system (i.e. noise) may have in the final prediction?. What would be the behavior if those aspects are improved or worsened?. What is the influence of noise of the machine, of interconnection network noise, of the bandwidth of the system, of disturbances introduced by the progression engine of MPI, among others. In this section, we demonstrate how our approach and infrastructure can be used to address above questions. We will demonstrate it analyzing the impact of noise in the Juropa platform used for the AVBP runs.

### *2.4.1. Impact of noise in communication*

Recall that our model uses a simulation with no latencies (ideal) to collect the values of Serialization and the resulting values are used to calculate Transfer. By using an ideal scenario potential disturbances within MPI in the original trace are cleaned, and inefficiencies are not assigned to Serialization but to Transfer. This factor can be read as the cost of data movement in the machine where the traces have been obtained, but it also captures other inefficiencies, such as: network noise, issues with the MPI progression engine of preemptions inside MPI calls. This effect was detected on AVBP where there is a significant difference between the behavior

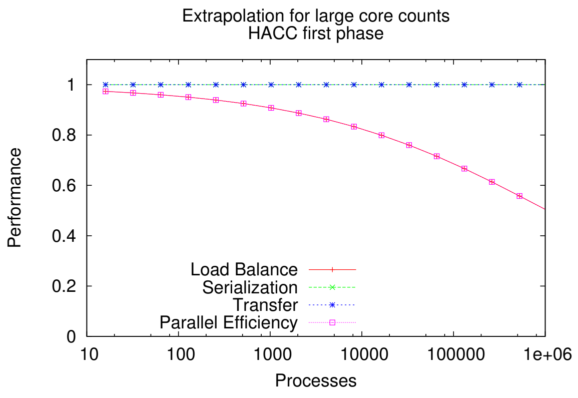


**Figure 19.** Extrapolation phase 1 HACC

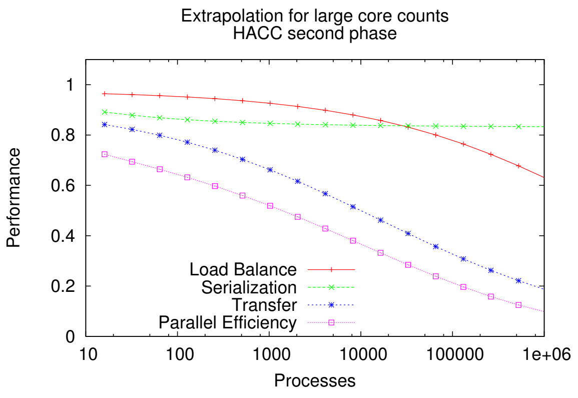


**Figure 20.** Extrapolation phase 2 HACC

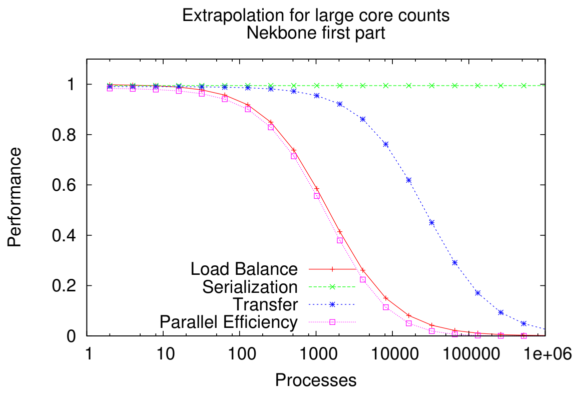


**Figure 21.** Extrapolation first part Nekbone

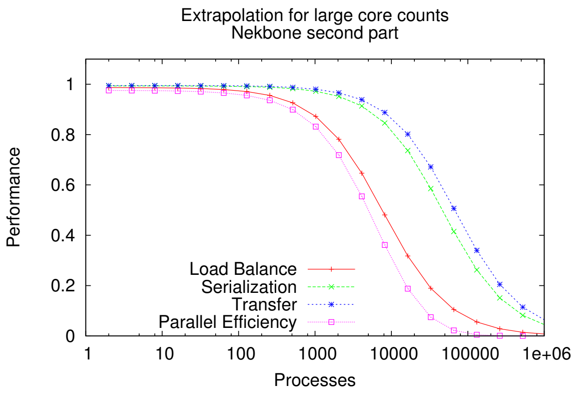


**Figure 22.** Extrapolation second part Nekbone

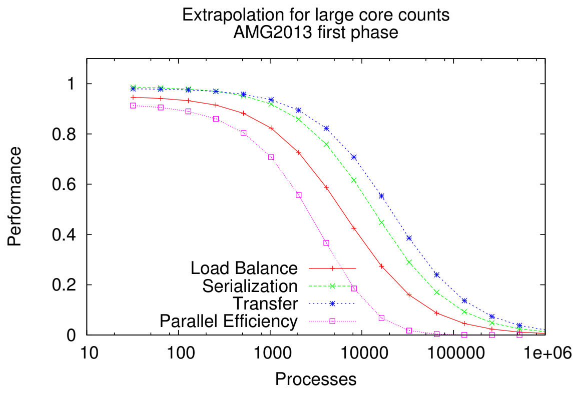


**Figure 23.** Extrapolation phase 1 AMG2013

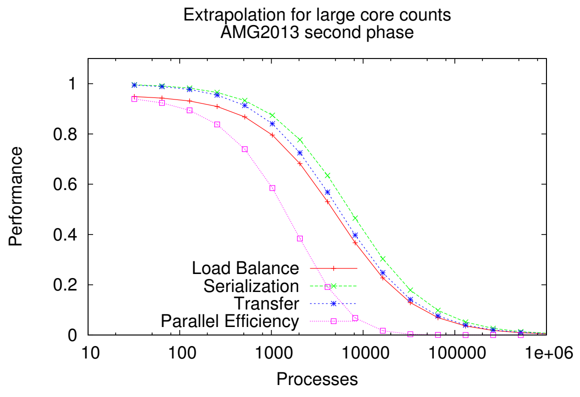


**Figure 24.** Extrapolation phase 2 AMG2013

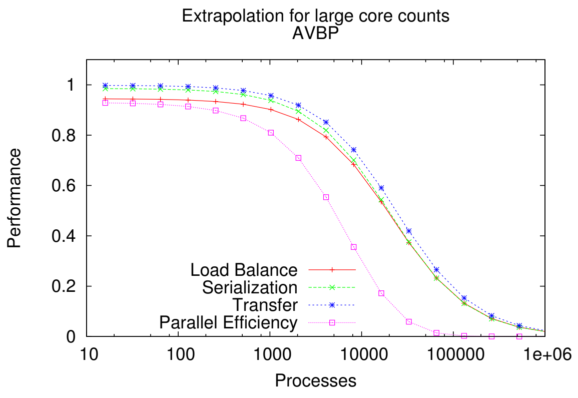


**Figure 25.** Extrapolation for large core counts for AVBP

of the 1024 and 1040 runs. Looking at the traces we detected problems on the MPI progression engine that can be seen on fig. 26 and affects the exit time of some collectives (Allreduce in this case).

Our apporach to eliminate the effect of such perturbations within MPI in the final results starts by generating a Dimemas trace from the original Paraver traces. The Dimemas traces capture the computation demands from the computation records in the original paraver trace. A Dimemas simulation with nominal values for the target machine rebuilds the precise time behavior on a platform without noise in the communication. The methodology presented in this paper can be applied to the paraver traces resulting from the simulation to extrapolate the efficiency factors of the application if the network/MPI noise is eliminated. The result of projecting the factors from traces of AVBP without disturbances in communication is shown in fig. 27. We can see how the transfer factor has now better scalability. Even if the network noise had an important impact in the transfer efficiency, the overall applications is still dominated by load balance and serialization and thus global performance does not significantly improves eliminating such noise.

### *2.4.2. Impact of noise in computation*

Noise may also affect the computational phases. To eliminate noise from these phases, the original trace is translated to Dimemas format using the available cycles counter to determine the duration of the computation burst. As hardware counters are virtualized they do not count while the process is preempted. By knowing the frequency of the processor and the number of actual cycles, a very precise computation of the non perturbed duration of computation burst can be obtained.

Repeating the process described in the previous section with the new conversion mechanism we obtain the extrapolation results reported in fig. 28. Now the load balance prediction scales better, thus suggesting that part of the identified unbalance was not originated by the application but by the system noise. By eliminating the noise from the machine and from communications, the serialization is now the dominant factor in efficiency loss.

From this additional study, we conclude that when predicting scalability of parallel applications we must be aware that current machines and interfaces are introducing variations (noise) in the executions. The design of mechanisms to include these noise factors into the prediction model may result a necessity.

## 3. Related Work

The approach based on first principles of [4] requires a deep knowledge of the application algorithms and parallelization structure to build analytical models directly from such knowledge. As it may result costly in terms of the deep application knowledge required, is computationally inexpensive and informative, having proved useful to actually identify problems in machines that did not get the predicted performance.

The work presented by [12], uses analytical application models to derive the performance of NAS BT benchmark in future systems. The approach builds a precise model of computation, memory usage, and communication to evaluate the potential paths to scale an application. The analysis provides insights of potential bottlenecks but does not formalizes an strategy to predict the expected performance.

A multiscale simulation based methodology is introduced by [13]. This work models the cluster level of parallelism by means of Dimemas, a simulator of a distributed memory target machine, parametrized by network and overall sequential performance parameters. The application characteristics are captured in a trace of a real run with the desired number of processes on an existing machine. The sequential performance extrapolations fed into Dimemas as part of the multiscale approach are computed using instruction level simulators, thus a costly process.

In an effort to investigate the performance of Message Passing Interface applications at large core counts, parallel discrete event simulations have been used to run the application in a controlled environment [3] and observe its behavior. The work of [14] and [15], analyzed the impact of communications when scaling to a larger number of processors, and the expected degradation in the network bandwidth, respectively. These works have been focused on the communication interface and in the hardware rather than in the basic behavior of the applications using them.

The work of [6] proposes to blindly fit metrics (essentially time) measured for the main routines of a program when running at low core counts on an existing platform. A large number of fitting functions is tried and the one reporting the best fit is selected as model of that routine. The approach is useful to identify trends and point to routines that will become bottlenecks at larger core counts in the same platform. The method, although it is simple and does not require excessive calculations, offers limited insight about influential factors in the performance loss.

A methodology to extrapolate the computational behavior of large-scale HPC applications has been presented in [16]. Their method extrapolates application traces as a relevant technique to understand how an application scales on a particular system, and can be useful to detect the impact of incremental or major changes in the hardware being used to run the application.

Several efforts to evaluate scalability of parallel applications has been made in [17]. They present a performance model for an specific phase of the AMG application, exposing existent bottlenecks and predicting the expected scalability in future machines based on their analytical model for computation and communication.

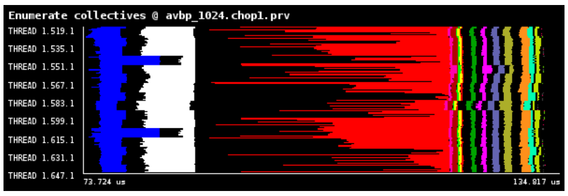


**Figure 26.** Delay in collectives for real trace of AVBP

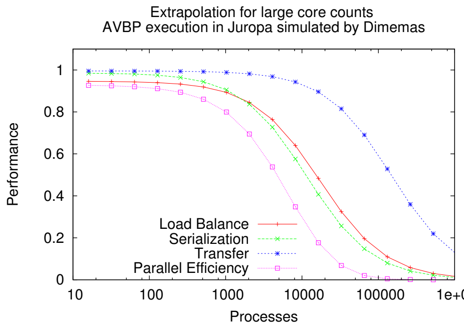


**Figure 27.** Extrapolation of parallel efficiency obtained from simulations of AVBP in Juropa (time)

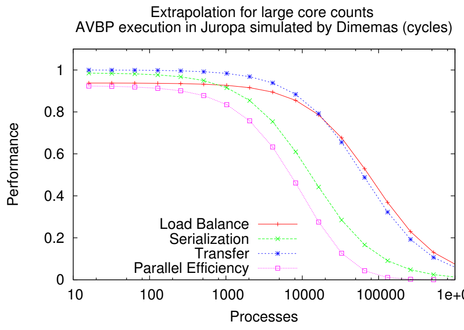


**Figure 28.** Extrapolation of parallel efficiency obtained from simulations of AVBP in Juropa (cycles)

## 4. Conclusions

In this paper, we described a methodology to collect primary components of current parallel codes and infer their expected behavior when scaled to larger core counts. To extrapolate the expected parallel efficiency, the approach extracted basic knowledge from traces obtained from runs using a low number of processes. Traces used in this work were obtained in two different machines that are currently in production (MareNostrum III and Juropa), and the process of analysis and estimation was performed by means of available performance tools and a semi-automatic framework. Our framework collected from the traces three fundamental components of parallel efficiency: load balance, serialization and transfer. Then, a first general model based on Amdahl's law was used to infer the evolution of each factor for large scale executions. We evaluated the method using 3 applications from the CORAL suite (HACC, Nekbone and AMG2013) and a CFD application AVBP. Predictions of expected efficiencies based on executions at low core counts showed a low relative error. Scalability projections showed interesting behaviors for strong and weak scaling, such as applications mainly dominated by load imbalances, efficiency loss caused by coupling of factors, among others. In general, from obtained results our method provides an inexpensive and useful tool to quickly infer the expected scalability of parallel codes.

As further steps, the method can be easily refined by including additional extrapolation models to fit different behaviors for new parallel codes. Similarly, to complement the current model with the potential effect of noise introduced by the machine or inherent noise of running a parallel code remains as a future work. The methodology can also be enriched by knowledge obtained from simulations of the parallel code on different architectures, thus providing additional insights on how the code may evolve in different platforms.

*This work has been partially supported by the Intel-BSC Exascale Lab and the DEEP Project.*

## References

1. A. Geist and R. Lucas, "Major Computer Science Challenges At Exascale," *Int. J. of High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 427–436, 2009.

2. J. Åström, A. Carter, J. Hetherington, K. Ioakimidis, E. Lindahl, G. Mozdzynski, R. Nash, P. Schlatter, A. Signell, and J. Westerholm, "Preparing Scientific Application Software for Exascale Computing," in *Applied Parallel and Scientific Computing*, vol. 7782, pp. 27–42, Springer Berlin Heidelberg, 2013.

3. Christian Engelmann, "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale," *Future Generation Computer Systems*, vol. 30, no. 0, pp. 59–65, 2014.

4. K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho, "Using Performance Modeling to Design Large-Scale Systems," *Computer*, vol. 42, pp. 42–49, Nov 2009.

5. P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, "Exascale Workload Characterization and Architecture Implications," in *Proc. of the High Perf. Computing Symposium*, pp. 5:1–5:8, 2013.

6. A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes," in *Proc. Intl. Conf. for High Perf. Computing, Networking, Storage and Analysis*, SC '13, pp. 45:1–45:12, 2013.

7. M. Casas, R. M. Badia, and J. Labarta, "Automatic Analysis of Speedup of MPI Applications," in *Proc. 22nd Intl. Conf. on Supercomputing*, pp. 349–358, 2008.

8. S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukić, "The Universe at Extreme Scale: Multi-petaflop Sky Simulation on the BG/Q," in *Proc. Intl. Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '12, pp. 4:1–4:11, 2012.

9. Center for Exascale Simulation of Advanced Reactors, "Proxy-Apps for Thermal Hydraulics." https://cesar.mcs.anl.gov/content/software/thermal_hydraulics.

10. Van E. Henson and Ulrike M. Yang, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Appl. Numer. Math.*, vol. 41, no. 1, pp. 155–177, 2002.

11. N. Gourdain, L. Gicquel, M. Montagnac, O. Vermorel, M. Gazaix, G. Staffelbach, M. Garcia, J.-F. Boussuge, and T. Poinsot, "High performance parallel computing of flows in complex geometries: I. Methods," *Comput. Sci. Disc*, vol. 2, no. 1, p. 015003, 2009.

12. R. van der Wijngaart, S. Sridharan, and V. Lee, "Extending the BT NAS Parallel Benchmark to exascale computing," in *Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, SC '12, pp. 1–9, 2012.

13. J. Gonzalez, M. Casas, J. Gimenez, M. Moreto, A. Ramirez, J. Labarta, and M. Valero, "Simulating Whole Supercomputer Applications," *IEEE Micro*, vol. 31, no. 3, pp. 32–45, 2011.

14. X. Wu and F. Mueller, "ScalaExtrap: Trace-based Communication Extrapolation for Spmd Programs," in *Proc. 16th ACM Symposium on Principles and Practice of Parallel Prog.*, pp. 113–122, 2011.

15. S. Dosanjh, R. Barrett, D. Doerfler, S. Hammond, K. Hemmert, M. Heroux, P. Lin, K. Pedretti, A. Rodrigues, T. Trucano, and J. Luitjens, "Exascale design space exploration and co-design," *Future Generation Computer Systems*, vol. 30, no. 0, pp. 46–58, 2014.

16. L. Carrington, M. A. Laurenzano, and A. Tiwari, "Inferring Large-Scale Computation Behavior via Trace Extrapolation," in *Proc. IEEE 27th Intl Symp. on Par. & Dist. Proc. Workshops and PhD Forum*, pp. 1667–1674, 2013.

17. H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp, "Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms," in *Proc. Intl. Conf. on Supercomputing*, pp. 172–181, 2011.

DOI: 10.14529/jsfi140202

# Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications

*Hayk Shoukourian*[1,2,*] , *Torsten Wilde*[1], *Axel Auweter*[1], *Arndt Bode*[1,2]

Keeping energy costs in budget and operating within available capacities of power distribution and cooling systems is becoming an important requirement for High Performance Computing (HPC) data centers. It is even more important when considering the estimated power requirements for Exascale computing. Power and energy capping are two of emerging techniques aimed towards controlling and efficient budgeting of power and energy consumption within the data center. Implementation of both techniques requires a knowledge of, potentially unknown, power and energy consumption data of the given parallel HPC applications for different numbers of compute servers (nodes).

This paper introduces an Adaptive Energy and Power Consumption Prediction (AEPCP) model capable of predicting the power and energy consumption of parallel HPC applications for different number of compute nodes. The suggested model is application specific and describes the behavior of power and energy with respect to the number of utilized compute nodes, taking as an input the available history power/energy data of an application. It provides a generic solution that can be used for each application but it produces an application specific result. The *AEPCP* model allows for ahead of time power and energy consumption prediction and adapts with each additional execution of the application improving the associated prediction accuracy. The model does not require any application code instrumentation and does not introduce any application performance degradation. Thus it is a *high level* application energy and power consumption prediction model. The validity and the applicability of the suggested *AEPCP* model is shown in this paper through the empirical results achieved using two application-benchmarks on the SuperMUC HPC system (the 10th fastest supercomputer in the world, according to Top500 November 2013 rankings) deployed at Leibniz Supercomputing Centre.

*Keywords: adaptive prediction, energy consumption, power consumption, energy capping, power capping, AEPCP model, energy measurement, node scaling, EtS prediction, HPC.*

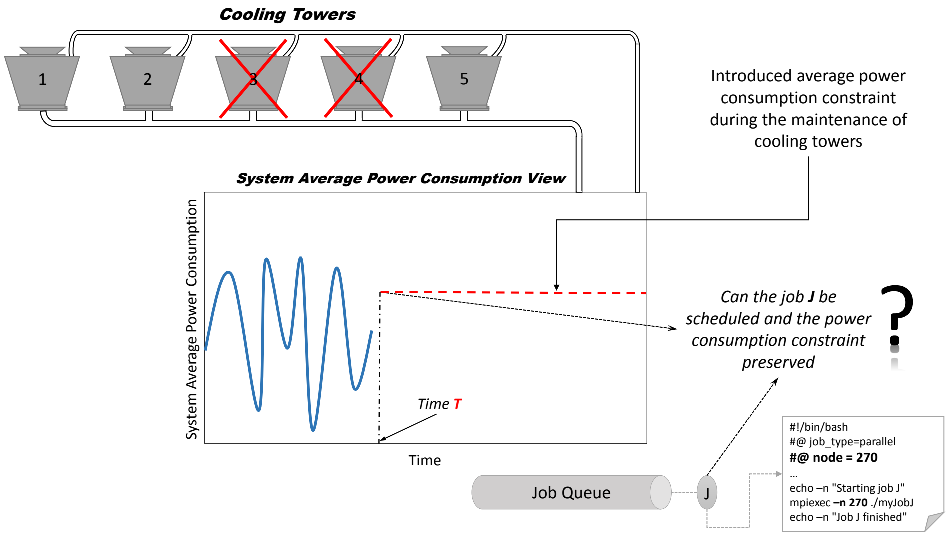
## Introduction

With the ever increasing growth of applications requiring a scalable, reliable, and low cost access to high-end computing, many modern data centers have grown larger and denser making power consumption a dominating factor for the Total Cost of Ownership (TCO) of supercomputing sites [18, 19]. This increase in power consumption not only converts into high operating costs, but also to high carbon footprint which affects the environmental sustainability, as well as straining the capacity limits of current data center's power delivery and cooling infrastructures. All these make a well-defined and efficient power management process a necessity for achieving a sustainable and cost-effective High Performance Computing (HPC) data center. Power and energy capping are two of the emerging techniques for controlling power and energy consumption in a data center [7].

*Power capping* limits the amount of power a system can consume when executing various applications, thus aiming to keep the system usage within a given power limit and prevent possible power overloads. Power capping covers a wide range of use cases: from limited power

[1]Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities
Boltzmannstraße 1, 85748 Garching bei München, Germany
[2]Technische Universität München (TUM), Fakultät für Informatik I10
Boltzmannstraße 3, 85748 Garching bei München, Germany
[*]Corresponding author. Email address: hayk.shoukourian@{lrz.de; in.tum.de}
Tel.: +49 89 35831 8772; Fax: +49 89 35831 8572

deliveries and/or limited cooling capacities; through the handling of power exceptions (e.g. unexpected peaks in system utilization); to power budgeting and mitigation of 'power-hungry' or malicious applications capable of generating dangerous power surges. Two interesting possible scenarios for power capping in a HPC data center are: avoiding runtime power peak, which can be addressed by new CPU features, such as setting a hardware power bound [27]; and temporary power constraints due to infrastructure maintenance (as illustrated in Figure 1). Figure 1 shows the average power consumption behavior (blue solid line) of a given HPC system cooled with the use of the data center's cooling towers depicted on the top of the image.


Cooling Towers
1
2
3
4
5
Introduced average power consumption constraint during the maintenance of cooling towers
System Average Power Consumption View
System Average Power Consumption
Can the job J be scheduled and the power consumption constraint preserved
?
Time T
Time
Job Queue
J

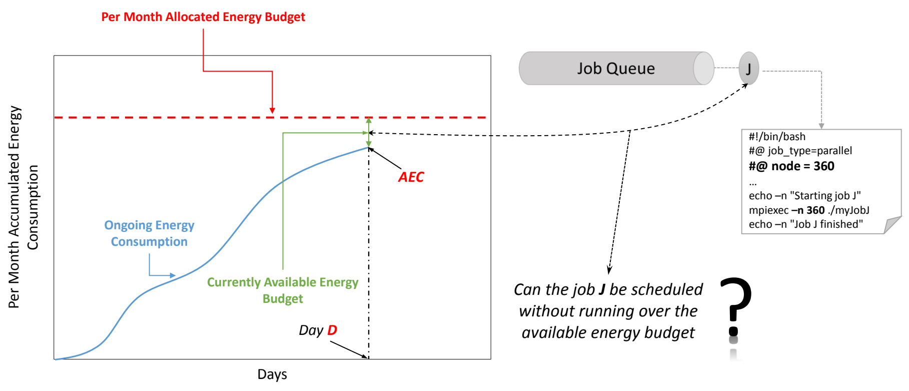
```
#!/bin/bash
#@ job_type=parallel
#@ node = 270
…
echo –n "Starting job J"
mpiexec –n 270 ./myJobJ
echo –n "Job J finished"
```

**Figure 1.** Power Capping use case scenario

Assume that at time $T$ two of the data center's cooling towers are in maintenance, introducing a temporary average power consumption constraint for the system. Now, assume that there is a queued job (application) $J$ with a utilization requirement of 270 compute nodes/servers, which needs to be scheduled for execution. In order to determine whether the execution of job $J$ is possible within the introduced average power consumption constraint, the information on the potential power consumption of job $J$ with 270 compute nodes is required. Without this information the scheduling of job $J$ could overload the available cooling capacity.

While power capping is useful, the majority of current techniques (e.g. [8, 13]) that implement power capping involve dynamic voltage frequency scaling [15], that will, in most cases, increase the runtime of the application [15], thus increasing the integral of power consumption over time (energy). *Energy capping* is another management technique that limits the amount of energy a system can consume when executing applications for a given time period. In other words, energy capping limits the integral amount of power consumption over time and, in contrast to power capping, it does not limit the amount of power the system can consume at a given point in time. From a data center perspective, energy capping is currently a more important approach, since energy consumption equals costs. The knowledge on application potential energy consumption for a given number of compute nodes will allow for a power-cost optimization by shifting low priority applications with higher energy/power consumption rates to off-peak hours, when the cost of electrical power is cheaper. This knowledge will also allow for energy-driven charging policies as an alternative to currently existing CPU-hour based charging policies.

A typical use case scenario of energy capping is illustrated in Figure 2. The dashed red line in Figure 2 shows the introduced per month allocated energy budget that a system can consume on a monthly basis (this can be for the whole system or on a per user/customer basis), whereas the blue solid line shows the ongoing energy consumption.


Per Month Allocated Energy Budget
Per Month Accumulated Energy Consumption
Ongoing Energy Consumption
Currently Available Energy Budget
AEC
Day D
Days
Job Queue
J
Can the job J be scheduled without running over the available energy budget
?

```
#!/bin/bash
#@ job_type=parallel
#@ node = 360
...
echo –n "Starting job J"
mpiexec –n 360 ./myJobJ
echo –n "Job J finished"
```

**Figure 2.** Energy Capping use case scenario

Assume that on day $D$ the system has already an Accumulated Energy Consumption ($AEC$, Figure 2) of a given amount. Assume further, that there is a pending job $J$, with requested 360 compute nodes. In order to understand whether the job $J$ can still be scheduled for execution within the available energy budget, the resource management system has to have the information on the potential energy consumption of the job $J$ with 360 compute nodes.

Though power and energy capping for these use case scenarios (as described for Figure 1, Figure 2) solve different problems, they both require the same knowledge of, potentially unknown, power and energy consumption profiles of applications to be executed. Without the access to this knowledge, the implementation of these techniques will be incomplete. This paper proposes an *Adaptive Energy and Power Consumption Prediction* (AEPCP) model capable of predicting the Energy-to-Solution (EtS) [4, 22] and the Average Power Consumption (APC) [37] metrics for any parallel HPC applications with respect to the given number of compute nodes. The $AEPCP$ model requires unique identifiers for each application and takes the *available* application historical power/energy data as an input. It is worth noting that this data is, in the most cases, already available in the current data center energy/power monitoring and resource management tools. The application can behave differently with different input data sets or if some system settings are changed (e.g. system dynamic voltage and frequency scaling governor configurations). Therefor, each substantial change needs to be treated as a different application and requires a new unique identifier. The model is validated for *strong scaling* applications (i.e. applications with fixed input problem size) as well as for *weak scaling* applications (i.e. applications with adjusted input problem size).

The remainder of this paper is structured as follows. Section 1 gives some background information on application scalability. Section 2 provides a survey on related works. Section 3 illustrates the prediction process and introduces the $AEPCP$ model. Section 4 describes the application-benchmarks as well as the compute system which were used to validate the suggested model. Section 5 presents the EtS results for application strong and weak scaling scenarios. Section 6 shows the APC prediction results, and discusses the benefits of $AEPCP$ based APC prediction as compared to the usage of vendor provided maximum power boundaries of system compute nodes. Section 7 looks at the future $AEPCP$ model enhancement directions, and finally Section 8 concludes the paper.

## 1. Background

The scalability of a parallel HPC application shows the relation between application execution time and the number of application utilized compute resources, e.g. nodes. Scaling is referred to as **strong** when an application input problem size (i.e. the amount of required computation) stays constant independently from the number of compute nodes which are utilized to solve that problem. This implies that an application demonstrating a strong scaling will have a smaller execution time, i.e. will solve the computation faster, as the number of compute nodes increase.

Scaling is referred to as **weak** when the input problem size of the application is fixed for each utilized compute node. This indicates that the execution time of an application under weak scaling will show a constant behavior since the input problem size increases accordingly with the number of utilized compute nodes. Figure 3 shows the execution-time, i.e. Time-to-Solution (TtS), behavior for strong and weak scaling scenarios.

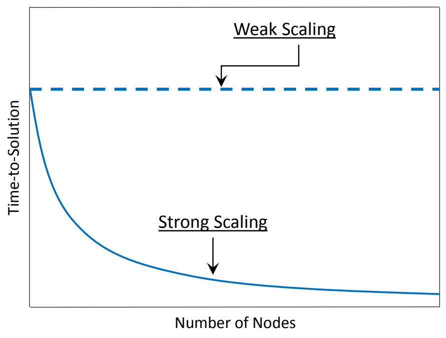


**Figure 3.** Theoretical TtS curves for strong and weak scaling scenarios

The limits of theoretically possible speedups achieved by parallel HPC applications in the case of strong and weak scaling and the outline of the theoretical boundaries of APC and EtS metrics under compute node scaling are presented in Subsection 1.1 and Subsection 1.2. The following denotations and definitions are used throughout these subsections:

- $t_s(n)$ - processing time of the application serial part using $n$ nodes;
- $t_p(n)$ - processing time of the application parallel part using $n$ nodes;
- $T(1) = t_s(1) + t_p(1)$ - processing time of the application sequential and parallel parts using 1 node;
- $T(n) = t_s(1) + t_p(n)$ - processing time of the application sequential and parallel parts using $n$ nodes;
- $p = \frac{t_p(1)}{t_s(1)+t_p(1)}$ - the **non-scaled** fraction of the application **parallel** part [29], i.e. the parallel portion of computation on a **sequential** system ($0 \leq p \leq 1$). Thus the **non-scaled** fraction of the application **sequential** part will be $(1 - p)$;
- $p^* = \frac{t_p(n)}{t_s(1)+t_p(n)}$ - the **scaled** fraction of the application **parallel** part [29], i.e. the parallel portion of computation on a **parallel** system ($0 \leq p^* \leq 1$). Thus the **scaled** fraction of the application **sequential** part will be $(1 - p^*)$.

## 1.1. Strong Scaling - Amdahl's Law

Strong scaling was first described analytically by Gene Amdahl in 1967 [1]. According to Amdahl's law, the possible speedup that a parallel application can achieve using $n$ ($n \geq 1$) compute nodes is:

$$Speedup(n) = \frac{T(1)}{T(n)} = \frac{1}{(1-p) + \frac{p}{n}} \tag{1}$$

The total $T(n)$ processing time of sequential and parallel parts using $n$ compute nodes, according to Amdahl's law (Equation 1), can be derived as:

$$T(n) = T(1) \cdot [(1-p) + \frac{p}{n}] \tag{2}$$

A study by Woo and Lee [37], considering Amdahl's law, proposes an analytical model for calculating the average power consumption $P(n)$ of a given application when executed on $n$ compute nodes.

$$P(n) = \frac{1 + (n-1) \cdot k \cdot (1-p)}{(1-p) + \frac{p}{n}} \tag{3}$$

where $k$ is the fraction of power that is consumed by the compute node in idle state ($0 \leq k \leq 1$). This further means that when an application demonstrates *ideal scalability*[1], then $P(n) = n$, as illustrated in Figure 4 (dashed yellow line). While when an application demonstrates *no scalability*[2], $P(n) = 1 + (n-1) \cdot k$ (solid yellow line in Figure 4).

Having Equation 2 and Equation 3, the EtS $E(n)$ of a given application can be derived as follows:

$$E(n) = T(n) \cdot P(n) = T(1) \cdot [1 + (n-1) \cdot k \cdot (1-p)] = \mathrm{O}(n) \tag{4}$$

which means that in the case of an application demonstrating *ideal scalability*, the EtS behavior for that application will be constant with respect to the number $n$ of utilized compute nodes. Whereas, in the case of an application with *no scalability*, the corresponding EtS behavior will be linear. The dashed and solid red lines in Figure 5 illustrate these scenarios. This further means, that the realistic EtS behavior of applications must be in between these constant and linear boundary lines.

## 1.2. Weak Scaling - Gustafson's Law

The speedup of applications demonstrating a week scaling was first described analytically by John L. Gustafson [11] as:

$$Speedup(n) = \frac{T(1)}{T(n)} = 1 + (n-1) \cdot p^{*} \tag{5}$$

Following the same observation proposed in [37], we can state that it takes $t_s(1)$ to execute the sequential portion of the computation and it takes $t_p(n)$ to execute the parallel portion of

[1]In other words the strictly serial $(1-p)$ fraction of the application is 0.
[2]In other words the computation fraction $p$ that can be parallelized is 0.

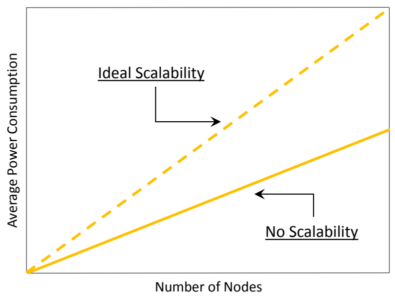


**Figure 4.** Theoretical APC curves for ideal and no scalability cases for **strong and weak** scaling scenario

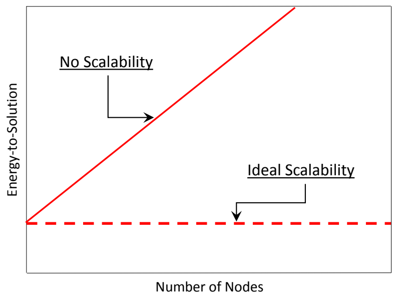


**Figure 5.** Theoretical EtS curves for ideal and no scalability cases for **strong** scaling scenario

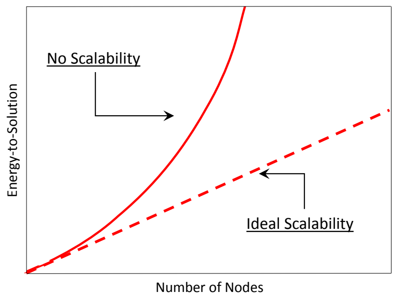


**Figure 6.** Theoretical EtS curves for ideal and no scalability cases for **weak** scaling scenario

the computation. Assuming that the fraction of power that is consumed by the compute node in idle state is $k$ $(0 \le k \le 1)$, the average power consumption $P(n)$ with respect to the number of utilized compute nodes can be written as:

$$\begin{aligned} P(n) = & \frac{t_s(1) \cdot [1 + (n-1) \cdot k] + t_p(n) \cdot n}{t_s(1) + t_p(n)} = (1 - p^*) \cdot [1 + (n-1) \cdot k] + p^* \cdot n = \\ & 1 + p^* \cdot (n-1) + (1 - p^*) \cdot (n-1) \cdot k = \mathrm{O}\big(n\big) \end{aligned} \tag{6}$$

This further means that in the case of an application that shows *ideal scalability*[3], the average power consumption $P(n)$ is: $P(n) = (n-1) + 1 = n$ (Figure 4). Since the execution time in the case of ideal scalability remains constant as the input problem sizes increases in parallel with the number of compute nodes, we can further state that the EtS behavior of the application $E(n)$ with respect to the given $n$ number of compute nodes is of a linear order: $E(n) = P(n) \cdot TtS(n) = n \cdot \mathrm{O}\big(1\big) = \mathrm{O}\big(n\big)$. The dashed red line in Figure 6 depicts this scenario.

In the case of an application that shows *no scalability*[4], the average power consumption $P(n)$ (from Equation 6) is: $P(n) = 1 + (n-1) \cdot k$ (Figure 4). Since the execution time of an application in the case of no scalability increases linearly with the input problem size and the number of compute nodes, the EtS $E(n)$, in the case of no scalability, will show a quadratic behavior with respect to the number of compute nodes $n$: $E(n) = P(n) \cdot TtS(n) = [1 + (n-1) \cdot k] \cdot \mathrm{O}\big(n\big) = \mathrm{O}\big(n^2\big)$ (solid red line in Figure 6).

As can be deducted from the above discussion, the average power consumption of an application, for both strong and weak scaling applications, is the highest when it demonstrates ideal scalability. Therefor, an artificial hardware power cap [27] might keep an application from providing the highest performance and could increase the overall TtS, and subsequently EtS as well.

Although, it was possible to derive the analytical EtS $E(n)$ and APC $P(n)$ boundary curves for strong and weak scaling applications with respect to the given $n$ number of compute nodes, the knowledge of an application's non-scaled $p$ (in case of strong scaling) or scaled $p^*$ (in case of weak scaling) fractions (which are application specific information) is necessary in order to estimate the energy/power consumption for a given $n$ number of compute nodes. The obtainment

[3]In other words the scaled fraction of the application sequential part $1 - p^*$ is 0.
[4]In other words the scaled fraction of the application parallel part $p^*$ is 0.

of this application specific information is not trivial, and might be even impossible, in real-world scenarios where myriad of different HPC applications are run in a HPC data center.

## 2. Related Work

An approach aimed towards performance prediction is described by Ipek et al. in [16]. The authors introduce a similar, adaptive model for predicting the TtS of parallel applications with respect to the input problem size of the application but with a fixed number of compute nodes. Even though, it could be possible to derive the energy consumption of an application using the corresponding knowledge of TtS and vendor provided maximum thermal design power [14] of a system compute node, this approach will not be applicable for our use case of energy and power capping, since it does not provide a knowledge on TtS behavior with respect to different numbers of compute nodes.

A study directed towards cross platform energy usage estimation of individual applications is found in [6]. The authors suggest a model capable of predicting the energy consumption of a given application during the application's execution phase. This model is not applicable for implementing energy/power capping techniques since it does not provide information on energy/power consumption of a given application in advance, which is required by the system resource manager for scheduling applications and preserving the predefined system energy/power consumption constraints.

Another set of approaches focused on predicting the energy consumption of applications using analytic models is found in [12] and in [5]. These approaches focus on predicting the power consumption of a given application with respect to a given CPU frequency. They both require knowledge of either the application (e.g. scaling properties) and/or the platform characteristics for different CPU frequencies. Both models are not yet extended/validated for multi-node compute systems and are analytic predictive models, which usually do not completely capture the interactions between underlying architecture and running software, and often require additional manual tuning [16].

A technique aimed towards controlling power consumption is found in [13]. It proposes a model, called "Pack & Cap", that adaptively manages the number of cores and CPU frequency depending on the given application characteristics, in order to meet the user-defined power constraints. "Pack & Cap" model is not applicable for the HPC domain, because, *first*, "Pack & Cap" model was validated on a single, quad core server node, and, as authors mention, the suggested technique is not yet extended/validated for large scale computing systems. *Second*, it needs a large volume of application performance data to conduct power/energy capping, which could not be available in real world scenarios. *Third*, it does not predict the power/energy consumption of applications. *In the end*, the model is targeted specifically for virtual machines, and might not therefor be easily adapted for HPC systems.

Another set of works focused on application energy/power consumption prediction, given application in-depth characteristics, is found in [24] and in [32]. [24] presents an energy consumption prediction model requiring application tracing (information on floating point operation count, memory operation count, etc.) and information on the energy profile of the target compute system (e.g. average energy cost per fundamental operation), obtained through the use of several special benchmarks. Although the suggested model could be used for a cross platform application energy consumption prediction, if the required energy profile data (e.g. achievable memory bandwidths for each level of the memory hierarchy) of the target system is available, their method involves application code instrumentation and attempts to split the application into "basic blocks" [24]. This would require a lot of effort when dealing with several hundred different applications, which is typically the case for modern HPC data centers. [32] suggests a quasi-analytical model, which combines the application analytic description (achieved through extensive application analysis) with the compute platform parameters (such as per-core power consumption of a computation unit, and power consumption during inter-processor communication) obtained through experimental benchmarks. While useful, the validation of a model was shown using a single benchmark and the suggested method requires a thorough analysis of the given application, which could be impractical in real-world scenarios, when several applications with different characteristics are queued for execution.

In summary, none of the aforecited models predicts the application energy/power consumption with respect to the number of compute nodes, and thus none of them can be applied for implementing power and energy capping techniques for our use case on large scale computing systems.

## 3. Framework

This section introduces the Adaptive Energy and Power Consumption Prediction ($AEPCP$) process, the $AEPCP$ model, and the monitoring tool which was used to obtain the application profile data.

### 3.1. The AEPCP Process

The prediction process of the approach suggested in this paper is outlined in Figure 7. The $AEPCP$ process has two inputs: the application identifier, which is used to uniquely identify an application, and the number of system compute resources (e.g. CPU, compute nodes, accelerators, etc.), which are planned to be utilized by a given application. The application identifier is used to query the application relevant history information from the system monitoring tool (step (1), Figure 7).

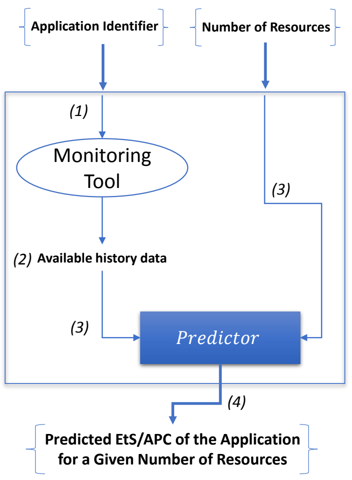

Application Identifier
Number of Resources
(1)
Monitoring Tool
(3)
(2) Available history data
(3)
Predictor
(4)
Predicted EtS/APC of the Application for a Given Number of Resources

**Figure 7.** Overview of the $AEPCP$ process

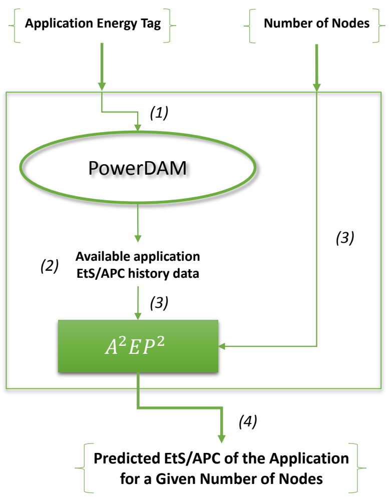

Application Energy Tag
Number of Nodes
(1)
PowerDAM
(3)
(2) Available application EtS/APC history data
(3)
$A^2EP^2$
(4)
Predicted EtS/APC of the Application for a Given Number of Nodes

**Figure 8.** Overview of the $AEPCP$ model

This application-relevant history profile data (step 2), together with the number of compute resources, is passed to the predictor (step 3) for corresponding EtS/APC prediction. Using this data, the predictor then reports the predicted EtS/APC value for the application with respect to the given node number (step 4).

## 3.2. The AEPCP Model

Figure 8 presents the overview of the *AEPCP* model based on the prediction process described above. The *AEPCP* model takes as input: *(i)* the application *energy tag* as an application unique identifier, which is supported by the IBM LoadLeveler [17] resource management system and is specified by the user on a unique-per-application basis; and *(ii)* the number of compute nodes as compute resource number (a compute node is the smallest compute unit available to an application on the SuperMUC [21] supercomputer which was used to validate the *AEPCP* model and is briefly described in Subsection 4.2).

The Adaptive Application Energy and Power Predictor ($A^2EP^2$) is used by the *AEPCP* model to estimate the application EtS/APC consumption for any given number of compute nodes. $A^2EP^2$ requires the application historical EtS/APC data. Figure 9 illustrates the workflow of $A^2EP^2$. As can be seen, if the application has already been executed for a given number of compute nodes (i.e. the EtS/APC consumption for that given number of compute nodes is known), then $A^2EP^2$ reports the averaged[5] value of all the available application history EtS/APC consumption data for that given number of compute nodes (step $Y1$, Figure 9).

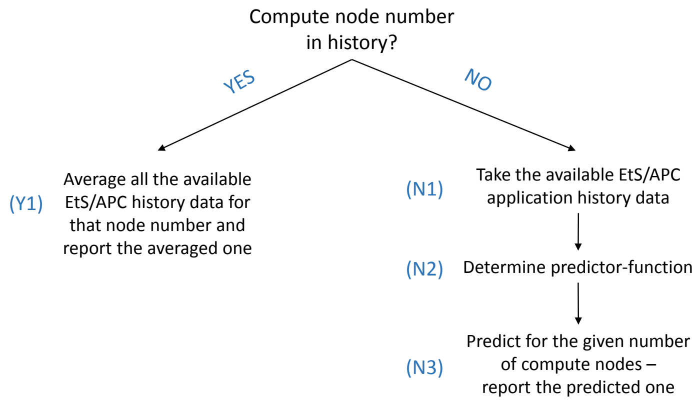

Compute node number in history?
YES
NO
(Y1)
Average all the available EtS/APC history data for that node number and report the averaged one
(N1)
Take the available EtS/APC application history data
(N2)
Determine predictor-function
(N3)
Predict for the given number of compute nodes – report the predicted one

**Figure 9.** $A^2EP^2$ workflow

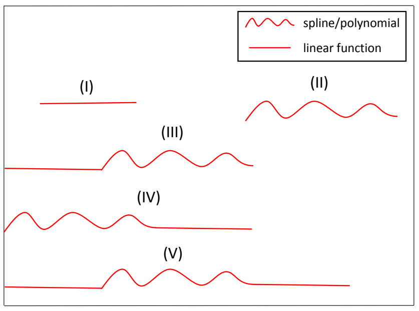


**Figure 10.** $A^2EP^2$ predictor estimation scenarios

If the history data of application EtS/APC consumption for a given number of compute nodes is not available, then $A^2EP^2$ queries the existing history data (step $N1$, Figure 9). This data, in our case, is obtained via a monitoring software toolset called PowerDAM [30, 31] (steps 1 and 2, Figure 8), which is an energy measuring and evaluating tool aimed at collecting and correlating data from different aspects of the data center. Once the application EtS/APC consumption history data is obtained, $A^2EP^2$ tries to determine a predictor-function (step $N2$, Figure 9) which will have an allowed, user-specified, percentage Root Mean Square Error (%RMSE). %RMSE is calculated from RMSE [23] as follows:

$$\%RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^{measured} - x_i^{predicted})^2} \cdot \frac{100 \cdot n}{\sum_{i=1}^{n} x_i^{measured}} \tag{7}$$

where

[5]This can be modified to the maximum or the minimum depending on the use case.

- $n$ is the number of available real measurements
- $x_i^{measured}$ is the $i^{th}$ measured real value
- $x_i^{predicted}$ is the $i^{th}$ predicted value

Several estimation techniques (e.g. ordinary least squares, spline interpolation, etc.) accompanied with energy/power consumption specific constraints (e.g. strict positivity) are used by $A^2EP^2$ for predictor-function determination. Knowing that EtS/APC of both strong and weak scaling applications is of the order of $\mathrm{O}(n)$ or $\mathrm{O}(n^2)$ (Section 1), $A^2EP^2$ analyzes the available history data and tries to find data points (from the obtained application history EtS/APC data) which would have a linear dependency. Depending on the found data points, $A^2EP^2$ divides the available history data set into linear and non-linear segments. $A^2EP^2$ distinguishes five different segmentations, as illustrated in Figure 10: *linear* (case I) is used for tracking the boundary curves described in Section 1; *non-linear* (case II) is used to track the transitional scaling phases between ideal scalability and no-scalability; *linear combined with non-linear* (case III) is used when to track the mixture of boundary and transitional scaling behavior; *non-linear combined with linear* (case IV) is used to track the mixture of transitional and boundary scaling behavior; and *linear combined with non-linear combined with linear* (case V) is used to track the mixture of boundary-transitional-boundary scaling behavior. For each linear segment, $A^2EP^2$ uses ordinary least squares to find a linear predictor-function which will have an allowed %RMSE with the available data set in that linear segment. For the non-linear segment, $A^2EP^2$ uses spline/polynomial interpolations (including also $1^{st}$ order splines/polynomials) in order to find a predictor-function which will have an allowed %RMSE rate with the history EtS/APC data points which are in that non-linear segment. Although, one could argue that there is no need for estimating higher than $2^{nd}$ order splines/polynomials because of known theoretical boundary, our experiments show that in the case of very limited application history EtS/APC consumption data, the higher order splines/polynomials are helpful and could result in a better prediction accuracy for a specific range of compute nodes.

Once the predictor-function is obtained from $A^2EP^2$, it is then used to estimate the EtS/APC values of the application for a given number of compute nodes (steps (4) and (5)). As can be observed, $A^2EP^2$ implementation is generic and produces individual results for each unique application. It adapts with each additionally available EtS/APC profile data-point for improving the accuracy of the determined (application-specific) predictor-function.

In summary, the described $AEPCP$ model: *(i)* is application neutral - does not need any knowledge on application type (e.g. communication, computation, or memory intensive), scaling properties, etc.; *(ii)* does not require any application code instrumentation; *(iii)* does not introduce any application performance degradation; *(iv)* allows for ahead of time EtS/APC prediction of a given application for a given number of compute nodes (does not require any partial/phase executions); and *(v)* automatically captures the complexity of the underlying hardware platform by taking the input data directly from the system [16], i.e. does not require any manual tuning of application properties or architectural peculiarities of the target platform.

## 4. Benchmarks and Compute System

### 4.1. Benchmarks

This subsection describes the two application-benchmarks which were used to validate the proposed model.

**Hydro** [20] is an application-benchmark extracted from the real world astrophysical code RAMSES [35]. Hydro is a computational fluid dynamics 2D code, which uses the finite volume method, with a second order Godunov scheme [9] and a Riemann solver [26] at each interface on a 2D mesh, for solving the compressible Euler equations of hydrodynamics.

**EPOCH** is a plasma physics simulation code developed at the University of Warwick as part of the Extendable PIC Open Collaboration Project [2]. EPOCH is based upon the particle push and field update algorithms developed by Hartmut Ruhl [28]. It uses the MPI-parallelized explicit $2^{nd}$ order relativistic particle-in-cell method, including a dynamic MPI load balancing option.

In contrast to many kernel and synthetic benchmarks, which are used to measure and test certain characteristics (e.g. processor power, communication rate, etc.) of the target platform, Hydro, as well as EPOCH (being application-benchmarks) provide a better measure of a real world performance. Hydro is part of the PRACE (Partnership for Advanced Computing in Europe) [25] prototype evaluation benchmark suit and EPOCH is an open-source real world application used by a large plasma physics community.

### 4.2. Compute System

SuperMUC (Figure 25), with a peak performance of 3 PetaFLOPS ($= 3 \times 10^{15}$ Floating Point Operations per Second), is the $10^{th}$ fastest supercomputer in the world (according to Top500 [36] November 2013 rankings) and is a GCS (Gauss Center for Supercomputing) infrastructure system made available to PRACE users. SuperMUC has 155.656 processor cores in 9421 compute nodes and uses IBM LoadLeveler [17] as a resource management system. It's active components (e.g. processors, memory) are directly cooled with an inlet water temperature of up to $40^\circ$ Celsius [21], allowing for chiller free cooling.

Four re-executions of the EPOCH benchmark on SuperMUC using the **same** set of compute nodes for the node numbers 20, 90, 180, and 256 showed that the measurement error per node number does not exceed 1.2%. Therefor, the quality of a single measurement (independently from the number of utilized compute nodes) is relatively high and there is no strong need for a re-execution of any benchmark.

## 5. Predicting Energy-to-Solution

This section presents the EtS prediction results for Hydro and Epoch using the *AEPCP* model. The history data points used throughout the paper were chosen on a random basis, since: *(i)* the data center has no control on the resource configurations requested by the users; and *(ii)* to explicitly show that model is independent from any specific history data.

### 5.1. EtS of Hydro Under Strong Scaling

Figure 11 shows the execution time of Hydro under strong scaling, which adheres to the theoretical discussion presented in Section 1 (Figure 3). Assume that there are three EtS data points in the monitoring history for Hydro (when executed under strong scaling) namely for compute node numbers: 130 with EtS of 7.6kWh; 135 with EtS of 7.9 kWh; and 220 with EtS of 7.6 kWh. Assume further, that there is an application in a job queue, which has an energy tag of strong scaling Hydro, and has a request of 320 compute nodes. The question to answer here is: *is it possible to predict the energy consumption of Hydro, when executed on* 320 *compute nodes, with* ***only*** *the knowledge of EtS consumption for compute nodes* 130, 135, *and* 220*?* Figure 12 shows that the use of $AEPCP$ model leads to a positive answer. The x-axis in Figure 12 represents the compute node number and y-axis represents the corresponding EtS in kWh. The red circle points correspond to the available EtS values. The red solid line shows the predictor-function curve, which was determined by $A^2EP^2$. A spline with smoothing degree of 1 having an %RMSE of 1% (with the EtS values of node numbers 130, 135, and 220) was estimated by $AEPCP$ model as a predictor function(red solid line, Figure 12). This estimated predictor-function estimates a 7.4 kWh energy consumption for compute node number 320. The green 'x' point in Figure 12 corresponds to the measured, EtS value (7.5 kWh) of Hydro when executed on 320 compute nodes. As can be seen, the prediction error rate[6] for 320 compute nodes is 1.3 %.

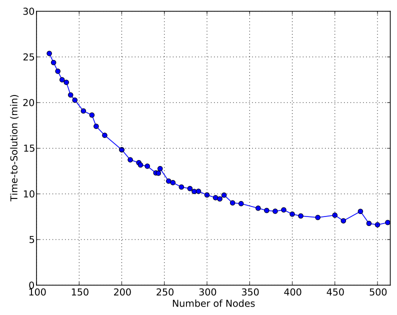
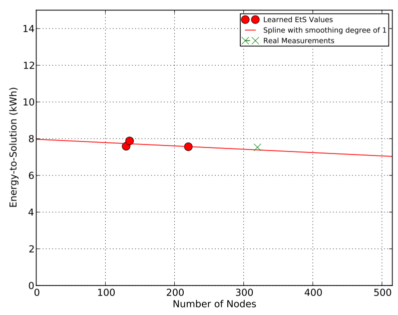
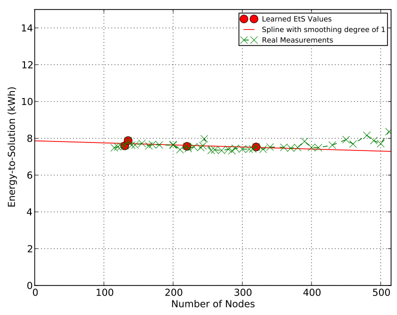


**Figure 11.** Measured TtS of Hydro under strong scaling



*Note: available data points are for node numbers:* 130, 135, *and* 220

**Figure 12.** EtS prediction curve and the measured EtS for node number 320



*Note: available data points are for node numbers:* 130, 135, 220, *and* 320

**Figure 13.** EtS prediction curve of Hydro under strong scaling

Figure 13 illustrates the case, when in addition to the Hydro EtS consumption data of compute node numbers 130, 135, 220, the EtS consumption value for already executed 320 compute node number is available to the $A^2EP^2$. In this case, a spline with smoothing degree of 1 (but with a different angle) having an %RMSE of 1% (the corresponding EtS value of 320 compute nodes was added to the original set of EtS data points for 130, 135, and 220 compute node numbers) was determined by the $A^2EP^2$ as a predictor-function. The red solid line in Figure 13 illustrates the curve of the predictor-function. The green '-x-x-' curve in Figure 13[7] corresponds to the measured (and **not available** to $A^2EP^2$) Hydro EtS values for different compute node numbers. As can be seen, the determined predictor-function (the red solid line in Figure 13) shows a relatively small deviation error rate from the measured data (the green '-x-x-' curve in Figure 13). Table 1 summarizes the detailed EtS prediction results for a random set of compute node numbers.

Figure 15 illustrates the real measurements of Hydro, again under strong scaling (Figure 14), but with a smaller input problem size. As usual, the green '-x-x-' points correspond to the real measured EtS data for different compute node numbers, whereas the red line corresponds to the determined predictor-function by $A^2EP^2$ using the available EtS values for node numbers: 1, 2, 4, 8, 16, 60, and 165 (red circles in Figure 15). As can be seen, a spline with a smoothing degree 2 (having an %RMSE of 1% with the available EtS values of node numbers: 1, 2, 4, 8,

[6] Calculated as: (| predicted value − measured value | /measured value) ∗ 100.
[7]Subsequently presented figures adhere to the same denotations.

| Number of Nodes | Measured EtS Value (kWh) | Predicted EtS Value (kWh) | Prediction Error[6] (%) |
|---|---|---|---|
| 115 | 7.5 | 7.7 | 2.7 |
| 200 | 7.7 | 7.6 | 1.3 |
| 285 | 7.5 | 7.3 | 2.7 |
| 300 | 7.4 | 7.5 | 1.4 |
| 340 | 7.5 | 7.5 | 0 |
| 400 | 7.5 | 7.4 | 1.3 |
| 460 | 7.7 | 7.3 | 5.1 |
| 500 | 7.7 | 7.3 | 5.2 |

**Table 1.** EtS prediction results for Hydro (strong scaling)

16, 60, and 165) was determined as a predictor-function by $A^2EP^2$. Although this determined quadratic behavior contradicts the estimated theoretical linear boundary (Equation 4, Figure 5), it provides an approximation with relative small error rate when compared with the measured data. On the other hand this estimated quadratic predictor starts to deviate from the real measurement data when the application approaches the saturation point, by transitioning to a non-scaling behavior, and thus according to Equation 4, shows a linear behavior of energy consumption with respect to the number of utilized compute nodes.

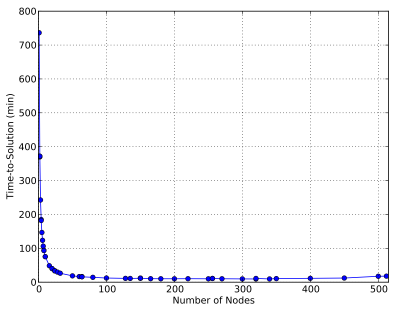
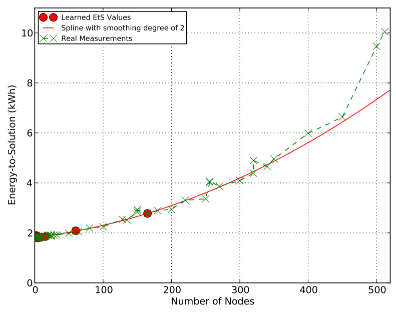
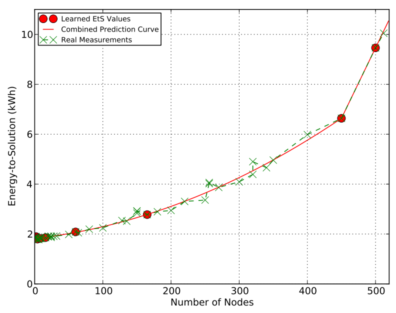


**Figure 14.** Measured TtS of Hydro under strong scaling *(smaller input problem size)*



*Note: available data points are for nodes:* 1, 2, 4, 8, 16, 60, *and* 165

**Figure 15.** EtS prediction curve of Hydro under strong scaling *(smaller input problem size)*



*Note: additionally available data points are for nodes:* 450 *and* 500

**Figure 16.** Revisited EtS prediction curve of Hydro under strong scaling *(smaller input problem size)*

One could argue, that there is no reason for executing an application (and thus conducting a prediction) on a higher number of nodes than the node number on which the saturation point for a given application was observed, since no performance increase for that application will be recorded. While true, $A^2EP^2$ tries to capture this behavior when sufficient data is available. Figure 16 illustrates this option, when EtS values for node numbers 450 and 500 were additionally available to $A^2EP^2$ for capturing this transitional behavior. As can be seen, the transitional-boundary behavior is tracked at the node number 450, and the quadratic function (illustrating the transitional[8] behavior) is now combined with the linear function illustrating the boundary behavior (case IV, Figure 10).

## 5.2. EtS of Hydro Under Weak Scaling

Figure 17 illustrates the expected (Section 1) execution behavior of Hydro under weak scaling[9]. Two Hydro EtS values were available for conducting the prediction (for nodes: 6 with EtS of 0.54 kWh; and for 32 with EtS of 2.84 kWh).

[8]In other words from ideal scaling to non-scaling.

[9]For weak scaling, also in Subsection 5.4, we assume that the problem sizes were uniformly adjusted for each compute node number.

As can be observed, a linear predictor-function was estimated by $A^2EP^2$ (Figure 18) showing a relatively small error rate for up to 512 compute nodes. Table 2 recaps the prediction results for a random set of compute node numbers.

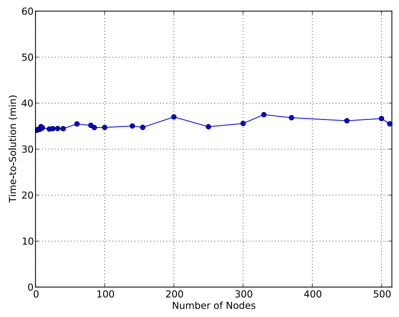
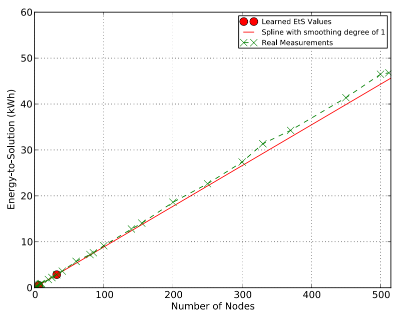


**Figure 17.** Measured TtS of Hydro under weak scaling



*Note: available data points are for nodes: 6, and 32*

**Figure 18.** EtS prediction curve of Hydro under weak scaling

| Number of Nodes | Measured EtS Value (kWh) | Predicted EtS Value (kWh) | Prediction Error[6] (%) |
|---|---|---|---|
| 10 | 0.9 | 0.9 | 0 |
| 25 | 2.2 | 2.2 | 0 |
| 40 | 3.6 | 3.6 | 0 |
| 80 | 7.2 | 7.1 | 1.4 |
| 100 | 9.1 | 8.9 | 2.2 |
| 200 | 18.6 | 17.7 | 4.8 |
| 370 | 34.3 | 32.8 | 4.4 |
| 450 | 41.4 | 39.9 | 3.6 |
| 512 | 46.8 | 45.4 | 3 |

**Table 2.** EtS prediction results for Hydro under weak scaling

## 5.3. EtS of EPOCH Under Strong Scaling

Four EtS values (for node numbers: 64 with EtS of 8.6 kWh; 75 with EtS of 8.8 kWh; 90 with EtS of 8.7 kWh, and 128 with EtS of 8.7 kWh) were available for conducting the EtS prediction. Figure 19 shows the measured TtS of EPOCH under strong scaling.

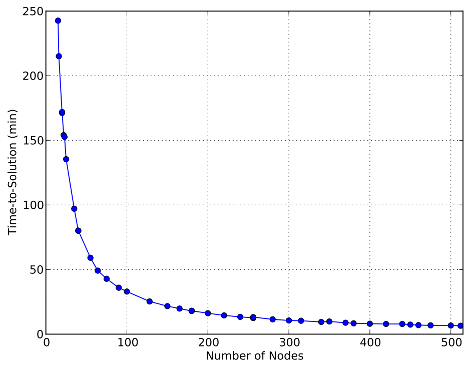
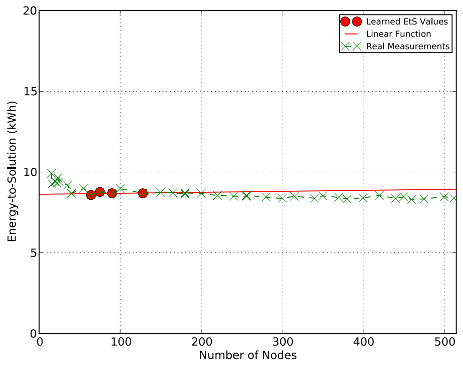


**Figure 19.** Measured TtS of EPOCH under strong scaling



*Note: available data points are for nodes:* 64, 75, 90 *and* 128

**Figure 20.** EtS prediction curve of EPOCH under strong scaling

A linear predictor-function, with an %RMSE of 0.8 with the available data points (red circle points, Figure 20) was determined by $A^2EP^2$. Figure 20 shows the curve of the determined linear predictor-function (red solid line). The green '-x-x-' curve corresponds to the real measured EtS data of EPOCH under strong scaling.

## 5.4. EtS of EPOCH Under Weak Scaling

Three EtS values (for node numbers: 16 with EtS of 1.1 kWh; 40 with EtS of 2.9 kWh; and 64 with EtS of 4.5 kWh) were available for conducting EtS prediction.

Figure 21 shows the measured TtS behavior for EPOCH under weak scaling. As can be seen it adheres to the theory. A linear function, having an %RMSE of 2.2% with the available data, was constructed by $A^E P^2$. Figure 22 shows the curve of the constructed predictor-function.

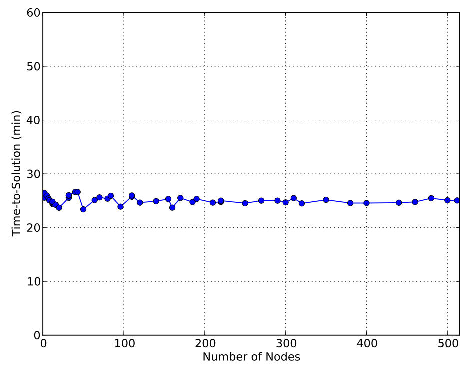
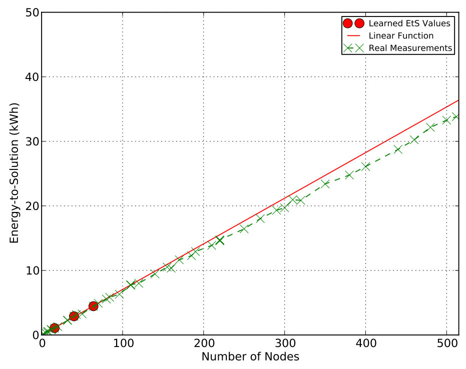


**Figure 21.** Measured TtS of EPOCH under weak scaling

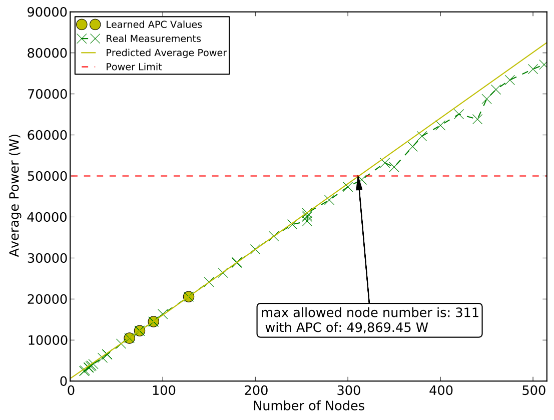


*Note: available data points are for nodes:* 16, 40, *and* 64

**Figure 22.** EtS prediction curve of EPOCH under weak scaling

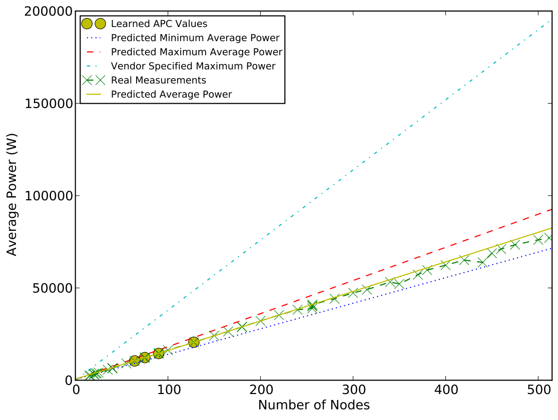
## 6. Predicting Average Power Consumption

Figure 23 shows the Average Power Consumption (APC) prediction results using the available APC values of four node numbers. As can be seen, the estimated linear predictor-function shows a relatively small error rate for up to 512 compute nodes. Another observation that can be inferred from Figure 23 is that the $AEPCP$ model can suggest the maximum compute node number that can be utilized by the application while preserving the introduced power consumption constraint. Figure 23 illustrates this option in the case of a 50,000 W power consumption limit. As can be seen, the maximum allowed compute node for running Epoch on SuperMUC, in the case of 50,000 W constraint, is 311 with predicted APC of 49,869.45 W.



*Note: available data points are for nodes:* 64, 75, 90 *and* 128

**Figure 23.** APC prediction curve for EPOCH under strong scaling



*Note: available data points are for nodes:* 64, 75, 90 *and* 128

**Figure 24.** Max and Min APC values for EPOCH under strong scaling

Our observations on SuperMUC supercomputer (Figure 25) show that the average power draw of the individual compute nodes differ when running the same application. This could be due to manufacturing tolerances and variations (e.g. processors [27], memory, power supplies, voltage regulators, etc.). Figure 26 shows the average power draws of different compute nodes of one of the SuperMUC's Islands (which consists of 516 compute nodes) when running a single MPrime[10] [10] benchmark. As can be seen, despite the hardware homogeneity across the SuperMUC's Island, there is a maximum of 41 W difference (nodes *i05r05a19-ib* with 188 W and *i05r03c28-ib* with 229 W) in average power draw of compute nodes.

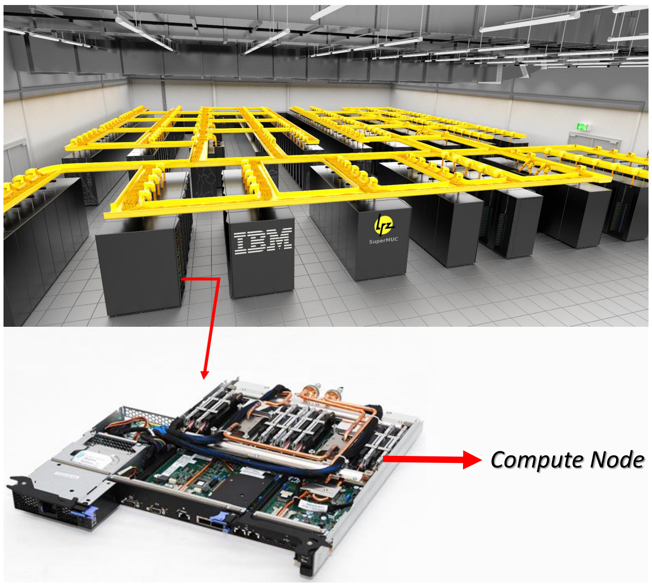
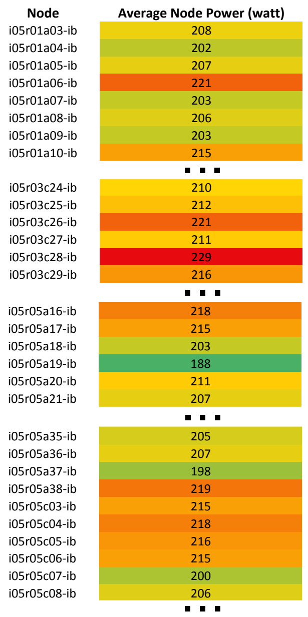


**Figure 25.** The SuperMUC Supercomputer



| Node | Average Node Power (watt) |
|---|---|
| i05r01a03-ib | 208 |
| i05r01a04-ib | 202 |
| i05r01a05-ib | 207 |
| i05r01a06-ib | 221 |
| i05r01a07-ib | 203 |
| i05r01a08-ib | 206 |
| i05r01a09-ib | 203 |
| i05r01a10-ib | 215 |
| | ▪ ▪ ▪ |
| i05r03c24-ib | 210 |
| i05r03c25-ib | 212 |
| i05r03c26-ib | 221 |
| i05r03c27-ib | 211 |
| i05r03c28-ib | 229 |
| i05r03c29-ib | 216 |
| | ▪ ▪ ▪ |
| i05r05a16-ib | 218 |
| i05r05a17-ib | 215 |
| i05r05a18-ib | 203 |
| i05r05a19-ib | 188 |
| i05r05a20-ib | 211 |
| i05r05a21-ib | 207 |
| | ▪ ▪ ▪ |
| i05r05a35-ib | 205 |
| i05r05a36-ib | 207 |
| i05r05a37-ib | 198 |
| i05r05a38-ib | 219 |
| i05r05c03-ib | 215 |
| i05r05c04-ib | 218 |
| i05r05c05-ib | 216 |
| i05r05c06-ib | 215 |
| i05r05c07-ib | 200 |
| i05r05c08-ib | 206 |
| | ▪ ▪ ▪ |

**Figure 26.** Power draw of compute nodes of the SuperMUC Island

The same behavior of compute node average power draw deviation under the same application execution was observed on CoolMUC [3] (shown in Figure 27). CoolMUC is a direct warm water cooled AMD processor based Linux cluster built by MEGWARE [33] and equipped with 178 compute nodes ($2 \times 8$-core AMD CPU). It is connected to a SorTech [34] adsorption chiller allowing the exploration of further possibilities of waste heat reuse of the system. CoolMUC has closed racks, and therefore does not require room air conditioning (Figure 27). All heat is removed solely via the chiller-less water cooling loop of the LRZ computer center infrastructure.

Figure 28 shows the power draws of different compute nodes of the CoolMUC Linux cluster when running the same single MPrime benchmark. As can be seen, despite the hardware homogeneity across the cluster, a maximum of 21 W difference in average power draw of compute nodes was observed (nodes *lxa11* with 240 W and *lxa46* with 261 W) during the MPrime benchmark.

If a *system compute node power classification* (Figure 26, Figure 28) is available, then the *AEPCP* model also predicts an application's possible maximum and minimum APC values for the scheduler application-assigned *"best"* and *"worst"* (in terms of power consumption) compute nodes. Using the APC history profile data of a given job $J$, *AEPCP* normalizes these values to the usage of the best compute node using Equation 8, and to the usage of the worst compute node, using Equation 9.

[10]Mprime is an application-benchmark that searches for Mersenne prime numbers, i.e. prime numbers of form $2^p - 1$, using Fast Fourier Transform algorithm. It introduces an intense workload to processor and memory, and because of that reason is usually used for system stability testing.

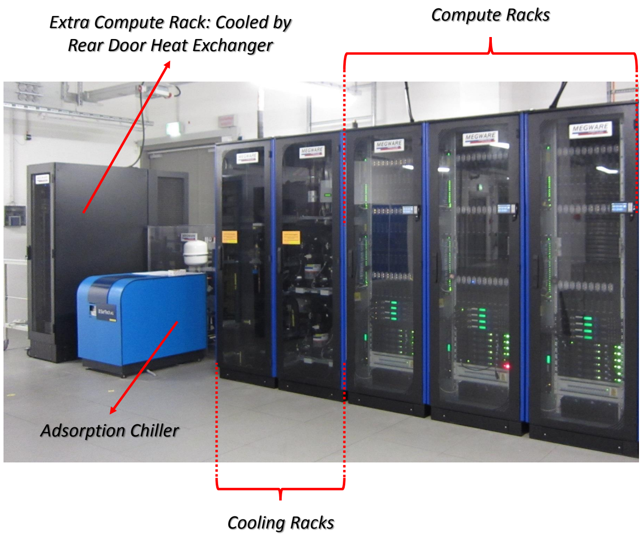
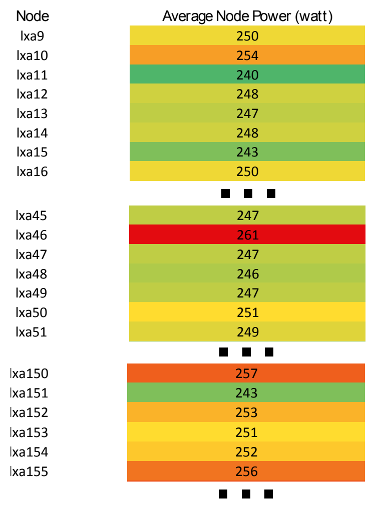


**Figure 27.** The CoolMUC Linux Cluster



| Node | Average Node Power (watt) |
|---|---|
| lxa9 | 250 |
| lxa10 | 254 |
| lxa11 | 240 |
| lxa12 | 248 |
| lxa13 | 247 |
| lxa14 | 248 |
| lxa15 | 243 |
| lxa16 | 250 |
| | ■ ■ ■ |
| lxa45 | 247 |
| lxa46 | 261 |
| lxa47 | 247 |
| lxa48 | 246 |
| lxa49 | 247 |
| lxa50 | 251 |
| lxa51 | 249 |
| | ■ ■ ■ |
| lxa150 | 257 |
| lxa151 | 243 |
| lxa152 | 253 |
| lxa153 | 251 |
| lxa154 | 252 |
| lxa155 | 256 |
| | ■ ■ ■ |

**Figure 28.** Power draw of compute nodes of the CoolMUC Linux Cluster

$$\parallel APC(J)^i \parallel_{min} = APC(J)^i - \sum_{\text{u utilized node of J}} (P_u - P_{min}) \tag{8}$$

$$\parallel APC(J)^i \parallel_{max} = APC(J)^i + \sum_{\text{u utilized node of J}} (P_{max} - P_u) \tag{9}$$

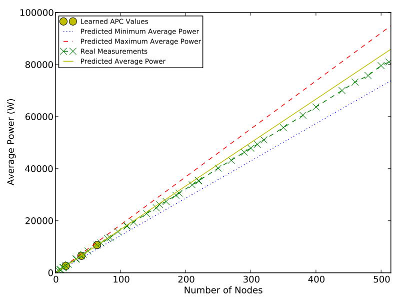
where

- $APC(J)^i$ - is the average mean power draw of job $J$ using $i$ compute nodes
- $P_u$ - is the average power draw of compute node $u$ obtained from the system compute node power classification (Figure 26)
- $P_{min}$ - is the average power draw of the most efficient (in terms of power) system compute node
- $P_{max}$ - is the average power draw of the least efficient (in terms of power) system compute node

Figure 24 illustrates this option of $AEPCP$ for EPOCH under strong scaling. The dashed '--' red line illustrates the predicted maximum APC behavior, the bottom dotted '...' blue line the predicted minimum APC behavior, the green '-x-x-' line the real measurement data (obtained from executions on the compute nodes of SuperMUC's Island Figure 26), and finally the yellow straight line depicts the $AEPCP$'s APC predictions.

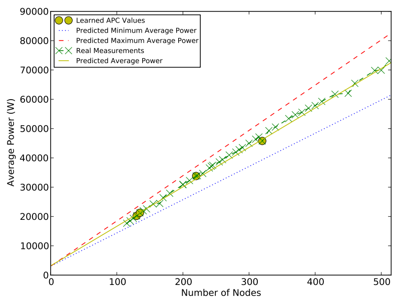
Several things can be observed from Figure 24. First, that the real measurements do not deviate much from the predicted APC values and stay in between predicted maximum and minimum APC values. Second, one could argue that the maximum APC value of an application for a given number of compute nodes can be derived by multiplying the given compute node number $n$ by $P_{\text{one node}}$ (maximum APC value of one node obtained from system-vendor provided compute node peak power consumption specification). The cyan '-.-' line in Figure 24 illustrates this $n \cdot P_{\text{one node}}$ approximation. While correct, this approximation gives a very rough boundary. For example, the usage of system-vendor provided approximation will lead to $118,108.47$ W power consumption estimation for 311 compute nodes on SuperMUC. Whereas the $AEPCP$ predicted maximum power consumption for the same 311 compute nodes, when running EPOCH under strong scaling, is 55, 993.13 W. As can be seen, the vendor specification based approximation is roughly two times larger as compared to the one estimated by the *AEPCP* model.

Figure 29, Figure 30, and Figure 31 illustrate the APC prediction results for EPOCH weak scaling, Hydro strong scaling and Hydro weak scaling correspondingly. As can be seen, all the three predictor-function curves show very small deviation rates from the measured values.



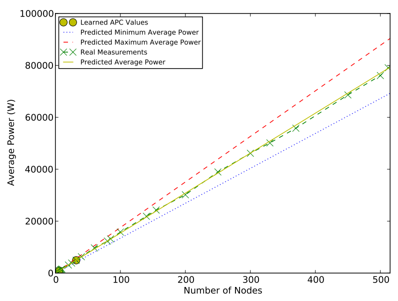
*Note: available data points are for nodes:* 16, 40 *and* 64

**Figure 29.** Max and Min APC values for EPOCH under weak scaling



*Note: available data points are for nodes:* 130, 135, 220, *and* 320

**Figure 30.** Max and Min APC values for Hydro under strong scaling



*Note: available data points are for nodes:* 6, *and* 32

**Figure 31.** Max and Min APC values for Hydro under weak scaling

## 7. Future Work

As was seen in Section 6, the power draw of the same application on different sets of compute nodes can differ despite hardware homogeneity across the HPC system. Thus, the possibility of compute resource set specific prediction, i.e. the support for exact declaration of compute resources for which the EtS/APC of the given application should be predicted, will produce more accurate results. It is worth noting that some of the EtS/APC measurements might not be completely accurate (e.g. due to possible noisy power sensor readings from which EtS/APC are calculated), and at the same time are not completely false. The specification of measurement "quality" as a weight in the set of available measurements, will allow for a better accuracy in prediction.

In addition to these two points, it is planned to develop an interface between the resource management system(s) and the *AEPCP* model. This interface will allow to dynamically track the possible violations of predefined energy and power consumption constraints depending on *(i)* the current workload information (obtained from the resource management system) and *(ii)* the predicted EtS/APC values for that workload (obtained from the *AEPCP* model).

This work will be included in a toolset at LRZ in order to support energy efficient supercomputing covering and optimizing the full set of influencing parameters: building and cooling infrastructure, supercomputer hardware, application and algorithms, systems software and tools.

## 8. Conclusion

The following contributions have been made in this paper:

- demonstration of the concept applicability for application power/energy consumption prediction for unknown number of compute nodes from previously observed data;
- explanation of how the application power/energy boundary curves can be defined from the known theoretical works and how this information can be applied in practice;

- exploration of the potential of the presented *Adaptive Energy and Power Consumption Prediction (AEPCP)* model for HPC data center power and energy capping use-cases;
- discussion on how the differences in HPC system compute node power can be used for power prediction;
- provision of a process and a generic implementation that provides application-specific power/energy consumption prediction results without need of the *AEPCP* model-implementation changes;
- since the *AEPCP* model is part of the PowerDAM toolset, this prediction can be done automatically for each application (queued or running) on the HPC system without any application specific adjustments.

The presented AEPCP model is a very interesting solution for HPC data centers, since it requires no application specific knowledge or information. The achieved accuracy is sufficient for the presented two most important use cases. By validating the model, we are just starting to scratch the surface for future possibilities. We are particularly looking forward to apply the model for system/user/data center energy budgeting and system peak power prediction. The suggested model can be an ideal building block for a real-world implementation of energy-aware resource management systems. It can also be used to help users/customers to actively take control over their power/energy budget and can help data centers to move to energy-driven charging policies alternatively to currently existing CPU-hour based charging policies.

*The work presented here has been carried out within the PRACE Second Implementation Phase project PRACE-2IP in the Work Package "Prototyping" which has received funding from the European Commission's Seventh Framework Program (FP7/2007-2013) under grant agreement no. RI-283493 and within the SIMOPEK project which has received funding from the German Federal Ministry of Education and Research (BMBF) under grand agreement no. 01IH13007A. The work was achieved using the PRACE Research Infrastructure resources at BAdW-LRZ with support of the State of Bavaria, Germany.*

*The EPOCH code used in this research was developed under UK Engineering and Physics Sciences Research Council grants EP/G054940/1, EP/G055165/1 and EP/G056803/1.*

*The authors would like to thank Jeanette Wilde for her valuable comments, Dr. Alexander Block for his kind support in performing experiments, and Dr. Anupam Karmakar for helpful discussions on EPOCH usage.*

## References

1. Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

2. T. Arber and et al. EPOCH: Extendable PIC Open Collaboration. http://ccpforge.cse.rl.ac.uk/gf/project/epoch/, 2014.

3. Axel Auweter and Herbert Huber. Direct Warm Water Cooled Linux Cluster Munich: CoolMUC. http://inside.hlrs.de/htm/Edition_01_12/article_26.html, 2012.

4. Arndt Bode. Energy to solution: A new mission for parallel computing. In Felix Wolf, Bernd Mohr, and Dieter Mey, editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of

*Lecture Notes in Computer Science*, pages 1–2. Springer Berlin Heidelberg, 2013.

5. Luigi Brochard, Raj Panda, and Sid Vemuganti. Optimizing performance and energy of hpc application on POWER7. *Computer Science - Research and Development*, 25(3-4):135–140, 2010.

6. G.L. Tsafack Chetsa, L. Lefèvre, J.M. Pierson, P. Stolf, and G. Da Costa. Exploiting performance counters to predict and improve energy performance of HPC systems. *Future Generation Computer Systems*, 2013.

7. Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

8. Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. *SIGMETRICS Perform. Eval. Rev.*, 37(1):157–168, June 2009.

9. Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959.

10. Great Internet Mersenne Prime Search. http://www.mersenne.org/freesoft/, 2014.

11. John L Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

12. Georg Hager, Jan Treibig, Johannes Habich, and Gerhard Wellein. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience*, 2014.

13. Can Hankendi and Ayse K Coskun. Adaptive power and resource management techniques for multi-threaded workloads. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, pages 2302–2305. IEEE Computer Society, 2013.

14. John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2012.

15. Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, pages 1–, Washington, DC, USA, 2005. IEEE Computer Society.

16. Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. An approach to performance prediction for parallel applications. In *Euro-Par 2005 Parallel Processing*, pages 196–205. Springer, 2005.

17. Subramanian Kannan, Mark Roberts, Peter Mayes, Dave Brelsford, and Joseph F Skovira. Workload Management with LoadLeveler. http://www.redbooks.ibm.com/abstracts/sg246038.html, 2001.

18. Jonathan G Koomey. Estimating total power consumption by servers in the US and the world, 2007.

19. Jonathan G Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3), 2008.

20. Pierre-François Lavallée, Guillaume Colin de Verdière, Philippe Wautelet, Dimitri Lecas, and Jean-Michel Dupays. Porting and optimizing hydro to new platforms and programming paradigms-lessons learnt. http://www.prace-ri.eu/IMG/pdf/porting_and_optimizing_hydro_to_new_platforms.pdf, December 2012.

21. Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities. http://www.lrz.de/, 2014.

22. Timo Minartz, JulianM. Kunkel, and Thomas Ludwig. Simulation of power consumption of energy efficient cluster hardware. *Computer Science - Research and Development*, 25(3-4):165–175, 2010.

23. Kevin P Murphy. *Machine learning: a probabilistic perspective.* The MIT Press, Cambridge, MA, 2012.

24. Catherine Mills Olschanowsky, Tajana Rosing, Allan Snavely, Laura Carrington, Mustafa M Tikir, and Michael Laurenzano. Fine-grained energy consumption characterization and modeling. In *High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2010 DoD*, pages 487–497. IEEE, 2010.

25. Partnership For Advance Computing In Europe. http://www.prace-ri.eu/, 2014.

26. Philip L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

27. Barry Rountree, Dong H Ahn, Bronis R de Supinski, David K Lowenthal, and Martin Schulz. Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 947–953. IEEE, 2012.

28. Hartmut Ruhl. Classical Particle Simulations with the PSC code. https://www.physik.uni-muenchen.de/lehre/vorlesungen/wise_09_10/tvi_mas_compphys/vorlesung/Lecturescript.pdf.

29. Yuan Shi. Reevaluating Amdahl's law and Gustafson's law. *Computer Sciences Department, Temple University (MS: 38-24)*, 1996.

30. Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Monitoring power data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers. http://www.sciencedirect.com/science/article/pii/S1364815213002934, 2013.

31. Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Petra Piochacz. Towards a unified energy efficiency evaluation toolset: an approach and its implementation at Leibniz Supercomputing Centre (LRZ). http://dx.doi.org/10.3929/ethz-a-007337628, 2013.

32. Shuaiwen Leon Song, Kevin Barker, and Darren Kerbyson. Unified performance and power modeling of scientific workloads. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, E2SC '13, pages 4:1–4:8, New York, NY, USA, 2013. ACM.

33. MEGWARE Computer Vertrieb und Service GmbH. http://www.megware.com/en/default.aspx, 2014.

34. SorTech AG. http://www.sortech.de/en/, 2014.

35. Romain Teyssier. The RAMSES Code. http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_sstechnique.php?id_ast=904, 2013.

36. Top500. http://top500.org/, 2013.

37. Dong Hyuk Woo and Hsien-Hsin S Lee. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *IEEE computer*, 41(12):24–31, 2008.

DOI: 10.14529/jsfi140203

# SLOWER: A performance model for Exascale computing

*Thomas Sterling*[1], *Daniel Kogler*[1], *Matthew Anderson*[1], *Maciej Brodowicz*[1]

A performance framework is introduced to facilitate the development and optimization of extreme-scale abstract execution models and the future systems derived from them. SLOWER defines a six-dimensional design trade-off space based on sources of performance degradation that are invariant across system classes. Exemplar previous generation execution models (e.g., vector) are examined in terms of the SLOWER parameters to illustrate their alternative responses to changing enabling technologies. New technology trends leading to nano-scale and the end of Moore's Law demand future innovations to address these same performance factors. An experimental execution model, ParalleX, is described to postulate one possible advanced abstraction upon which to base next generation hardware and software systems. A detailed examination is presented of how this class of dynamic adaptive execution model addresses SLOWER for advances in efficiency and scalability. To represent the SLOWER trade-off space, a queue model has been developed and is described. A set of simulation experiments spanning ranges of key parameters is presented to expose some initial properties of the SLOWER framework.

## 1. Introduction

As technology advances, the means to effectively benefit from its improved properties changes, sometimes significantly, in computer architecture, programming models, supporting system software, and other aspects. Together these reflect a crosscutting execution model, which must evolve to respond to opportunities and challenges of the emerging enabling technologies. But the choice space is often large at least in detail as execution model alternatives and system designs are explored. Guidance is required to govern derivation of new execution models and the design of underlying systems incorporating their innovative concepts. Currently, the challenge of realizing extreme-scale operation is one of efficiency and scalability within the constraints of energy and reliability.

The history of high performance computing is punctuated with dramatic paradigm shifts resulting in a succession of innovative execution models as the device technologies have progressed. But they share in common a set of key factors to which each has responded. Together these factors constitute a performance framework, within which past and future execution models can be analyzed and evaluated as well as compared. Such a performance framework can serve as the needed guidance of future system development by characterizing the trade-offs in structure, semantics, control, and enabling mechanisms. This paper describes one such performance model, SLOWER, that is serving in this capacity for research into extreme-scale execution models, runtime system software, and programming interfaces.

The breakdown of Dennard scaling [8] and the nearing end of Moore's Law in conjunction with the dominant limitation of power consumption are posing the greatest challenge to continued performance growth in two decades. Radical departures from previous conventional massively parallel processors (MPPs) and commodity clusters now incorporating multicore sockets and graphics processing unit (GPU) accelerators are requiring innovations in system design, operation, and usage. To optimize these new class of systems requires a performance framework to guide design evolution in a meaningful and quantifiable trade-off space.

The experimental SLOWER performance framework establishes a six-dimensional space of consideration; each letter of the acronym reflecting one of the dimensions. Starvation reflects

[1] Center for Research in Extreme Scale Technologies, School of Informatics and Computing, Indiana University, Bloomington, IN 47408, USA

an insufficiency of concurrent work to keep all the critical resources engaged. Latency quantifies the response time distance (usually in a measure of time, e.g., cycles) for remote access and service requests. Overhead distinguishes the work needed to manage parallel resources and task scheduling but that does not actually contribute to the useful computational work itself. Delay due to waiting time incurred due to resource access conflicts captures the last of the key dependencies of performance degradation. But two remaining factors contribute indirectly to effective system usage. Energy and its rate of consumption (power) is a key parameter in the raw sequential performance potentially affecting both logic voltage and clock rate. The final parameter, reliability, directly impacts availability of the system and therefore the percentage of time the system can be employed for useful work. Together these can expose the dominant contributors to the effects related to performance degradation.

Unfortunately, at this time, there is no adequate formalization of the SLOWER framework; no unified set of equations that fully captures the subtle interrelationships among the contributing factors. To make the situation worse, these parameters are not purely orthogonal; therefore a perturbation of one may directly alter another. This makes the framework less than ideal for direct analysis and application to distinguish among possible future systems and programming models. Recent work on exploring the SLOWER trade-off space is presented and discussed in this paper. SLOWER is described in detail with quantifiable units of measure. A table is provided that categorizes the history of multiple execution models over the last forty years in terms of their response to the SLOWER factors and their respective technology base. An example of a possible future class of execution models is represented by the experimental ParalleX model. ParalleX is described in some detail and then how it addresses the SLOWER factors through the assumption of dynamic adaptive techniques is discussed to show that alternative paradigms may be possible to enable future extreme-scale computing. To move this exploration from the qualitative to quantitative, a set of experiments are being conducted through the use of queuing simulation for parameter sweeps to expose the resulting trade-off space. Early experimental results are presented that exhibit partial sensitivities among the dominant parameters and inform consideration of design alternatives. The paper closes with summary conclusions and immediate future work.

## 2. Related Work

The most important historical performance model is Amdahl's law [2] which provides a simple expression for the maximum expected speedup of an algorithm using multiple processing units while disregarding issues of latency, overhead, and contention. Among performance models which do consider the impact of latency and overhead, the LogP [6] and related family of LogGP [1] and LogGPS [10] models have long formed the core of parallel computation modeling. LogP, summarized as **L**atency, **o**verhead, **g**ap required between two send/receive operations, and **P**, the number of processors, enables performance modeling with distributed memory and communication between processing units. Subsequent extensions of LogP incorporated longer messages (LogGP) and synchronization costs (LogGPS). These models have been extremely successful in guiding algorithm development and performance using conventional programming model approaches such as the Message Passing Interface (MPI). Before LogP, the Parallel Random Access Machine, or PRAM, model [9] also functioned as a performance model to guide parallel algorithm development by examining performance for multiple processors having constant time access to a shared memory. While a simple model, the flat cost of PRAM did not closely model real behavior for the multiple layers of communication systems in modern architectures. Vector Random Access Machine [3] (V-RAM), a vector performance model incorporates instructions for vector operations in the context of sequential random-access memory.

The LogP family of performance models have been used to analyze performance of many MPI based applications. However, the rise of multi-core architectures has resulted in new performance models that recognize multi-threading. The Multi-core LogP (MLogP) model [5] extends LogP while redefining the $P$ parameter in terms of the amount of computing power available rather than just the number of processors in order to better model performance of many-tasking approaches. The Log$_{\text{N}}$P [4] extends LogP to incorporate the impact of memory and middleware with the capability to model point-to-point communication in clusters of Symmetric Multi-Processor nodes (SMPs). SLOWER, in contrast, provides a non-LogP based framework for multi-core architectures with a focus on medium and fine-grained parallelism while also incorporating the Exascale relevant issues of energy and resilience.

## 3. SLOWER

A performance model can serve as a means of evaluating the effectiveness of execution models, distinguishing among distinct execution models, and devising improved execution models with respect to available enabling technologies. A useful performance model is capable of describing sensitivities of performance metrics to contributing factors. The SLOWER performance model is introduced to provide a framework for evaluation and analysis of execution models. It guides their development and the implementation of advanced systems the structure and operation of which they govern. SLOWER is derived from a set of key parameters that contribute to the degradation of performance. The units of each parameter translate to metrics of time. Throughout the history of high performance computing these factors have been addressed directly or implicitly in the context of prevailing underlying technologies through the design of architecture, methods of parallel programming, and the mediation and control of system software. These foundational parameters have been invariants across the many generations of supercomputers and are a framework with which to track the transitions in computing systems and methods.

A formulation gives overall average performance in terms of these parameters is given in Eqn 1:

$$P = e(L, O, W) \times s(S) \times \mu(E) \times a(R) \tag{1}$$

where $P$ is average performance, $e$ is efficiency ($0 < e < 1$), $L$ is latency, $O$ is overhead, $W$ are delays resulting from contention for shared objects, $s$ is scalability or concurrency, $S$ is starvation, $\mu$ is the normalized per execution unit per cycle performance, $E$ is the power factor (energy) that determines clock rate, $a$ is the availability of resources ($0 < a < 1$), $R$ is the reliability that determines down-time events.

Equation 1 identifies the critical SLOWER parameters and shows their interrelationships and contributions to the overall performance. Both the efficiency and availability are fractional. The availability measure takes into account those operational issues that can detract from a system's ability to perform real work. One of these, $R$, is the Mean Time Between Failures (MTBF) of the system. This combined with scheduled downtime and maintenance time (time to recovery) determines the fraction of the total time a system can actually deliver user computation. This parameter can be extended to incorporate scheduling conflicts with other workload in the job stream.

There is an assumed peak single instruction stream rate of operation, $\mu$, that combines the register-to-register operation rate with the average load/store memory access time, which is admittedly application and cache hierarchy dependent. Here, however, is exposed the relationship between the clock rate and the power consumption, $E$. Within a restricted range, clock rate and therefore $\mu$ is proportional to the rate of energy consumption (power).

Scalability, $s$, is the amount of concurrency or number of execution units (e.g., cores) that are allocated to a computation at the same time. Starvation, $S$, is the inverse reflecting the absence of work to be performed through concurrency. Starvation is either due to an insufficiency of pending or available work to be performed or an imbalance of the work distribution across computing resources (some have too much while others too little).

The efficiency, $e$, of usage of the system is the ratio of the sustained rate of operation execution to the peak rate. As shown in this relationship a number of factors contribute to the efficiency. Latency, $L$, is the time to make a remote access or service request (not including the overhead of doing so) of an unloaded system. Latency can be measured in clock cycles. The delay, $W$ (waiting for delayed access) incurred for access contention to shared logical or physical resources takes into consideration the effects due to a loaded system. Such access conflicts can occur for networks, memory banks, or synchronization objects (e.g., barriers). Overhead, $O$, is the additional work required to manage parallel resources and task scheduling beyond the actual useful work of the computation itself. Overhead consumes resources (and cycles) that could otherwise be employed for useful work thus reducing the relative efficiency of system operation. From these parameters is derived the SLOWER performance model framework.

While the performance relationship is valid for any particular operating point on average, it is imperfect in deriving a deeper understanding through sensitivity analysis because the dominant parameters are not orthogonal. One trivial case is that parameters measured in cycles may vary depending on the energy consumption rate, $E$, that causes the clock rate to change and therefore the cycle time. More significant is that the efficiency, $e$, and the scalability, $S$, are interrelated. Usually over part of the range of system scale, as $S$ increases, generally $e$ decreases. More subtly, overhead not only wastes cycles, it imposes a lower bound on the effective granularity that can be employed. For fixed size work (strong scaled), this puts an upper bound on parallelism and therefore an asymptotic limit on delivered performance. In spite of these shortcomings, SLOWER allows developers of parallel execution models and system (hardware and software) architectures to reason formally about design choices, sensitivities, and projected capabilities.

SLOWER does not directly reflect one other important quality metric, productivity. This parameter is poorly understood and not formally defined. Nonetheless, notionally it not only embodies the issues associated with SLOWER, it also combines the other life cycle issues such as programmability, generality, performance portability, and their trade-offs with delivered performance. Productivity is outside the scope of this work but crucial to establishing long term strategies and direction of future extreme-scale computing systems and methods.

## 4. Experimental Setup

To demonstrate the viability of the SLOWER execution model, a simulation tool called "JaamSim" [11], developed by Ausenco, was used to simulate abstract machines with a variety of parameters. The parameters considered included network latency, instruction mix, overhead

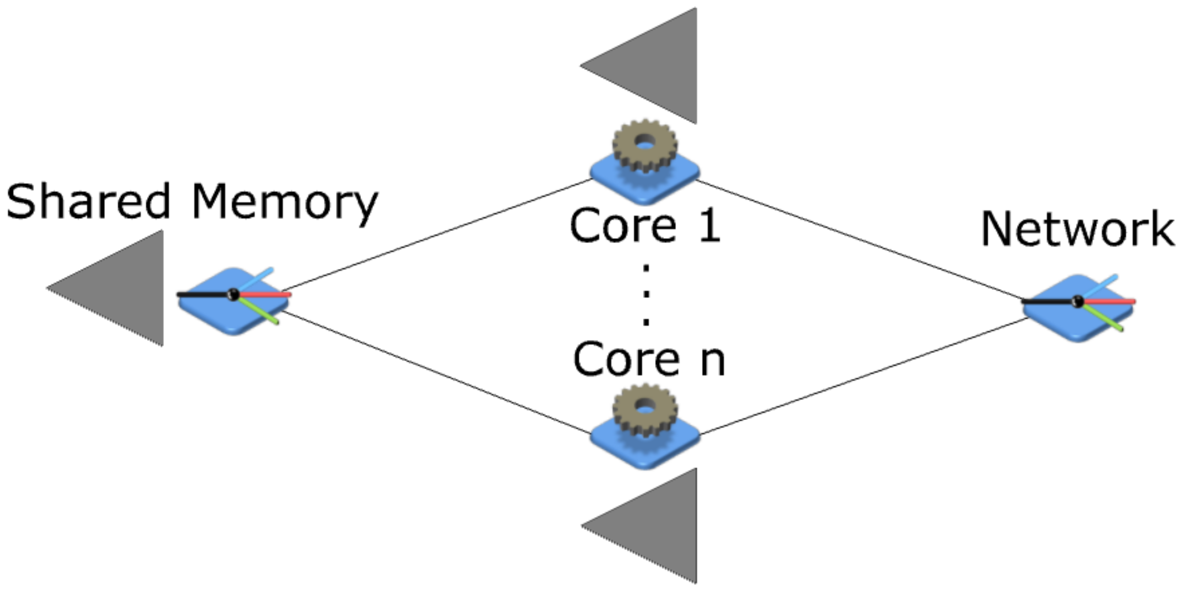


**Figure 1.** This JaamSim model builder diagram expresses the general flow of work in simulations. There are three types of components present: cores, the network, and memory. Each gear-component represents a core, and the cores are labeled from 1 to $n$ for some $n$ declared in the simulation parameters. The components with a three-way branch symbol (i.e. the network and memory components) indicate that any task arriving at that component is sent back to its place of origin after the branch component's processing is completed. Each component with a triangle next to it indicates that the component makes use of a queue data structure for handling work (in this case, lightweight tasks). Only one task may be worked on per core at a time. Likewise, only one memory request can be serviced by the shared memory at a time. Additionally, if a core sends a task to the memory unit, that core halts processing of any further tasks in its queue until the task returns from memory.

for context switching and network interaction, available parallelism per core, and number of cores per node. Here, available parallelism per core is represented by the number of lightweight tasks ready to run waiting on that core. Effectively, starvation, latency, overhead, and some effects of contention are modeled in these simulations, while energy and reliability are left out.

Each simulation assumed a single shared memory unit across all cores, and unlimited network bandwidth. Additionally, the lightweight tasks are assumed to never complete. In other words, available parallelism is held constant throughout each simulation and context switching overhead is only encountered when tasks switch due to a network operation.

For all simulations, the performance gain of programs using non-blocking protocols for network operations over programs using blocking protocols is measured. To compare the relative performance, each set of simulations was run twice. The first time using the parameters as described above with a non-blocking protocol, and the second time using identical parameters (except for overhead) while using a blocking protocol. The simulations of programs using a blocking protocol were all given an overhead of zero for context switching on network operations, as the program waits for the network operation to complete rather than changing tasks. An example simulation diagram can be seen in Figure 1.

In the context of this work, programs following blocking protocols temporarily cease performing useful work whenever a network operation is encountered. Work is resumed upon completion of the network operation. Note that only the core on which the network operation is invoked stalls during the network operation. No context-switching occurs for these programs except upon task completion (which these simulations assume does not happen).

Non-blocking programs also temporarily cease performing useful work upon encountering a network operation, but do not wait until the network operation completes before resuming work. Rather, once the network operation has been invoked, the program performs a context-switch if there are other tasks in the program able to work. If no tasks are ready, then the program switches to the task that will be ready the soonest and then stalls until that task is ready to proceed with useful computations. The program will always perform a context-switch upon encountering a network operation, even if the context-switch overhead is greater than the network latency. This is because the program is assumed to not have knowledge of the time required to perform a context-switch nor of the network latency.

Performances of different simulation setups were compared via the total number of operations each setup completed in a given constant amount of time. This constant time was set to be the time required for $2^{17}$ register operations (reg-ops) to complete in the simulation. All durations in the simulation are normalized based on the time required for a register operation to complete. The choice for using $2^{17}$ reg-ops was made because this number proved sufficiently large to hide any significant noise in the performance results of simulations with network latencies of 8192 reg-ops (the largest latency simulated).

In all simulation models, the access time for local memory (excluding effects of contention) was set to 200 reg-ops. However, not every memory operation access the shared local memory. The simulations also take into account the effects of cache performance. The cache performance of all simulations is defined such that the programs run with a 90% L1-cache hit rate and a 90% L2-cache hit rate, and the hardware has L1-cache access times of 1 reg-op and L2-cache access times of 10 reg-ops. Additionally, the instruction mixes for all simulation models always include exactly 68% register instructions (excluding network and context-switching overheads). The remaining instructions are divided between memory accesses and network operations.

Four sets of simulations were run. The first set examined the effects of varying the overheads on the overall performance of the system. For these simulations, the model applied a constant overhead for each network access that represented the cost of context switching and network communication. The overheads applied ranged from 1 to 8192 reg-ops. The number of cores used in the simulation ranged from 1 to 32. All other parameters were kept constant. The network latency was set to 8192 reg-ops; each core was provided 64 tasks; and the instruction mix contained 8% network operations and 24% memory operations.

The second set of simulations explored the effects of having differing numbers of lightweight tasks available per core for various numbers of cores per node. The purpose of these simulations was to visualize the relative effects of available parallelism versus memory contention. For each of these simulations, the network latency was held constant at the equivalent of 8192 reg-ops; context-switching overhead was set to 16 reg-ops; and the instruction mix contained 8% network operations and 24% memory operations.

The third set of simulations examined the effects of changing the amount of contention experienced by an application (albeit limited to memory contention) with respect to various network latencies. This was accomplished by varying the number of cores present in the system. For these simulations, the number of cores was varied from 1 to 32. Network latency was ranged between 64 and 8192 reg-ops. All other parameters were kept constant. Overhead was set to 16 reg-ops; each core was given 64 lightweight tasks; and the instruction mix contained 8% network operations and 24% memory operations.

The final set of simulations examined in more detail the interrelationship of latency and starvation given a variety of instruction mixes. Because one form of starvation is the lack of parallelism (the other form being poor load-balancing), starvation's effects are examined by varying the number of available lightweight tasks per core. The network latency ranged from only 64 reg-ops to 8192 reg-ops, the number of available tasks ranged from 1 task per core to 128 tasks per core, and the instruction mixes contained anywhere from 0% to 16% network operations. Context switching overheads were kept constant at 16 reg-ops.

## 5. Experimental Results

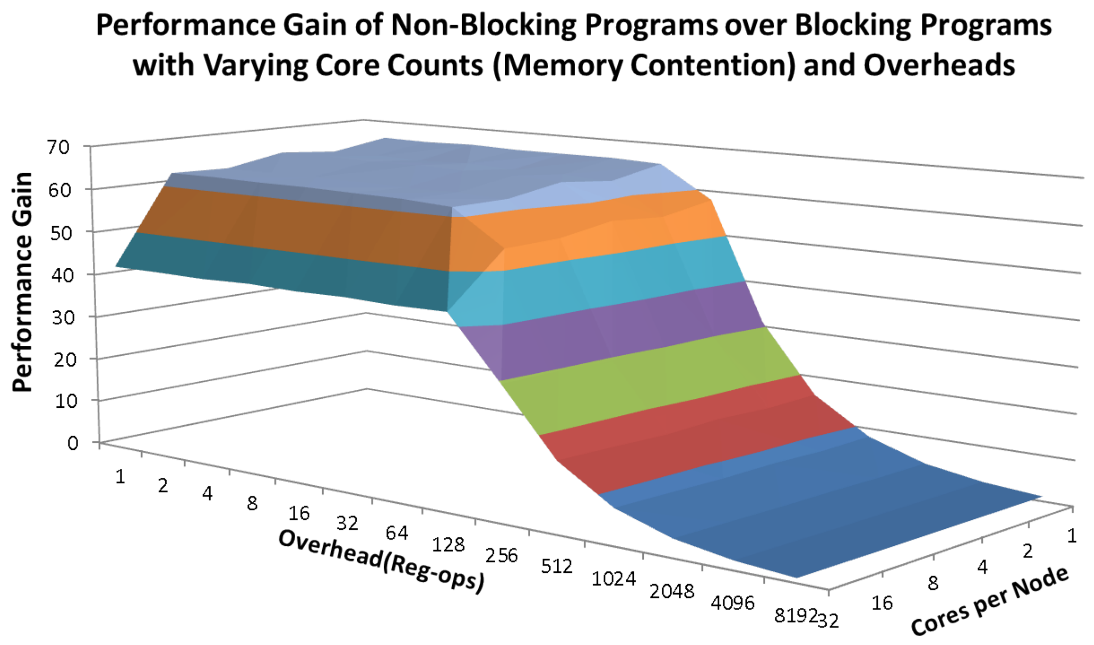
The first set of simulations looked into the effects of overhead on the performance of an abstract machine running a program with a particular instruction mix. The results of this set of simulations can be seen in Figure 2. The graph presented shows the performance gain of a program using a non-blocking protocol when performing network operations as compared to a program that blocks on network operations.

As the graph indicates, there is a clear plateau in the performance gain of non-blocking programs over blocking programs. This plateau occurs when two conditions are met. The first condition is with regards to the effect of the overhead of context-switching and the number of available tasks waiting on each core. When the number of tasks multiplied by the overhead of a single context switch is less than the network latency, then we can achieve little further performance gain by continuing to decrease the overhead. This is because at this point, further decreasing the overhead increases the probability that all tasks on that core will end up waiting on network operations. The second condition relates to memory contention, which increases as the number of cores sharing the memory increases.

In order to further explore the plateau phenomenon, the second set of simulations was run. The difference in this set of simulations is that the tasks per core is varied and the overhead is held constant, whereas the previous set held a constant availability of tasks but with different levels of overhead. What is gathered from these experiments is that both decreasing the overhead and increasing the available parallelism lead to an exponential increase in the performance gain, up until the point where the product of the two equals the network latency. At this point, a plateau is reached. What is interesting is that in the former case the product is decreasing yet in the latter case the product is increasing.

This may seem contradictory to the previous results at first. After all, the previously mentioned effect reached its plateau when the latency was greater than the product of overhead and number of tasks. Closer examination reveals that these are in fact different scenarios. In Figure 2, the limiting factor in the performance gain was the lack of available parallelism, i.e. starvation. Decreasing overhead would not negate the effects of starvation. In Figure 3, the limiting factor is this time the overhead of context-switching. Adding more tasks to the system will not decrease the limiting effects of the overhead.

The results of the third set of simulations can be seen in Figure 4. The same plateau as in Figure 2 is visible here as well, though here the effect can be seen as limited to the higher latencies. This is because for lower latencies it is no longer possible for all tasks of a single core to be waiting on the network simultaneously. However, this does not explain the sudden decrease in performance gain that occurs as the number of cores is increased. The expected trend



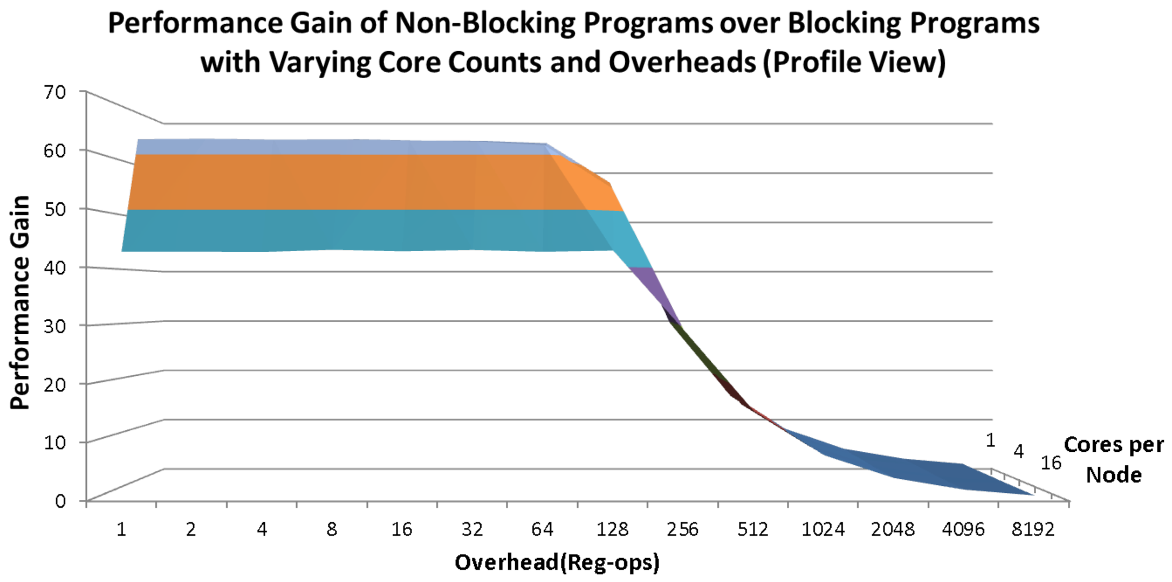


**Figure 2.** The performance gain of programs that do not block on network operations over those that do. This plot assumes a network latency equivalent to 8192 register operations. The number of parallel tasks available on each core is 64. The instruction mix consists of 68% register operations, 24% memory operations and 8% network operations. Overhead is measured with respect to the number of register operations that could complete in the same elapsed time. This figure demonstrates the deterioration in performance gain as both context-switch overheads increase and the number of cores increases. The latter decrease is due to increased contention for memory resources experienced by the non-blocking programs. The plateau in performance gain begins to occur where the product of the overhead and the number of tasks available equals the network latency ( $128 \times 64 = 8192$ ). At this point the available parallelism begins to be exhausted.

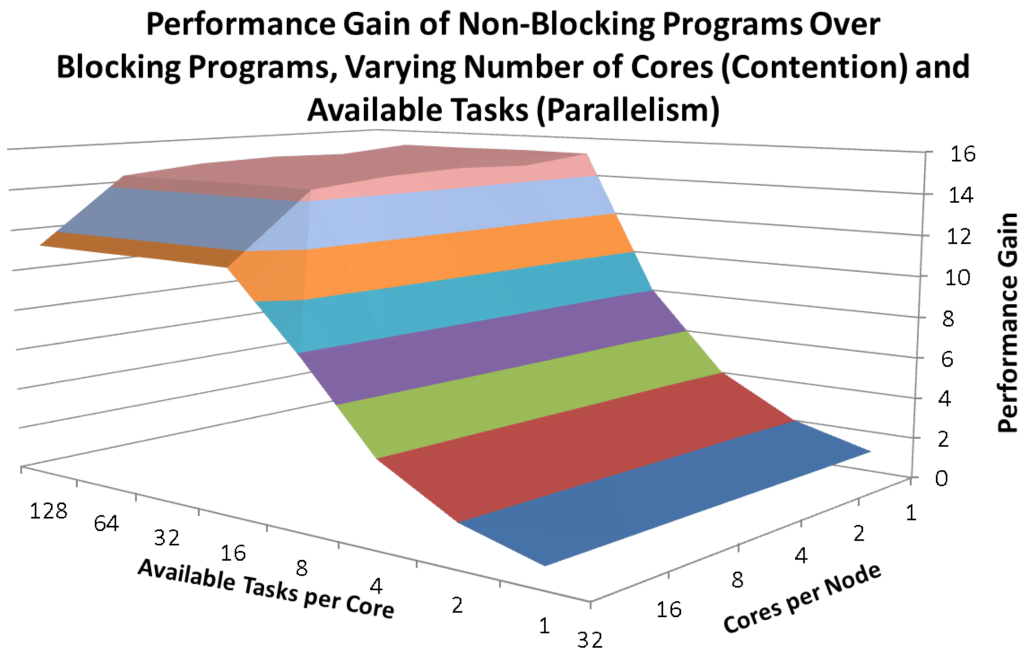


**Figure 3.** The performance gain of several simulations using non-blocking computations as compared to identical simulations that block computations upon accessing the network. All simulations were given an instruction mix consisting of 68% register operations, 8% network operations, and 24% memory operations. The network latency is equivalent to 8192 register operations. The overhead for context-switching in non-blocking simulations is equivalent to 16 register operations. Whereas Figure 2 plateaued when the available parallelism was insufficient, this plot plateaus when the overhead (with respect to the latency) limits how much parallelism can be taken advantage of.

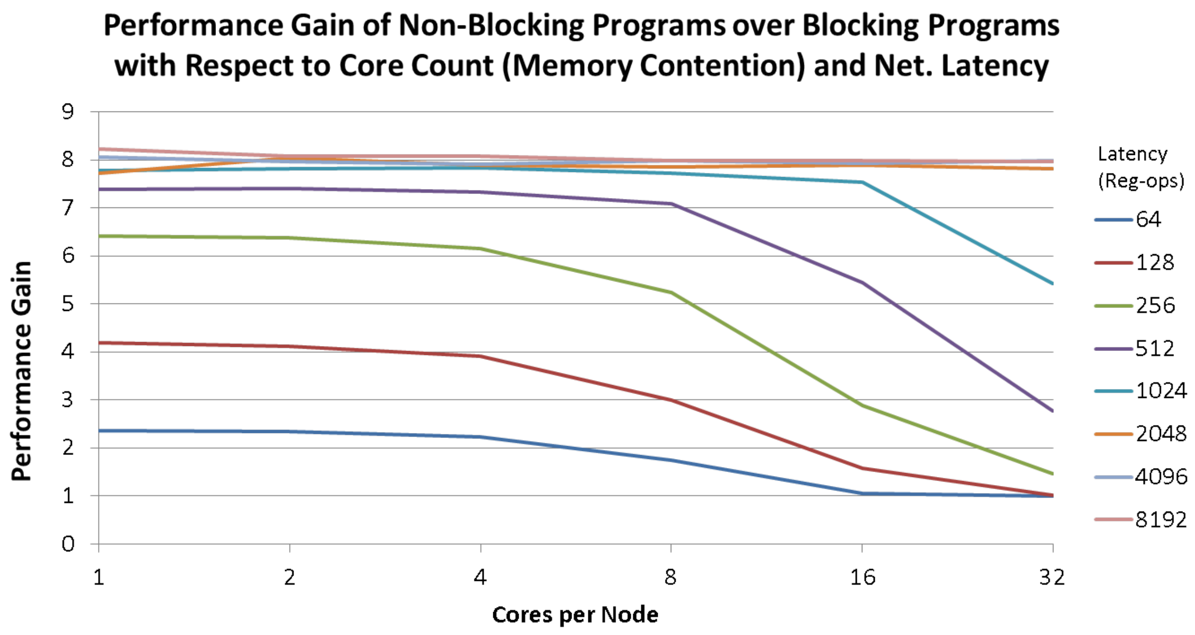


**Figure 4.** The performance gain of several simulations using non-blocking computations as compared to identical simulations that block computations upon accessing the network. All simulations were given an instruction mix consisting of 68% register operations, 8% network operations, and 24% memory operations. The overhead for context-switching in non-blocking simulations is equivalent to 16 register operations. Contention is also represented here; the likelihood for contention increases as the number of cores increases. For higher latencies,the probability of contention is lower, and its effects are less noticeable. Each core in the non-blocking simulations was assumed to have 8 tasks available for work. The number of tasks available for work (8) provides the upper bound for performance gain. The upper bound is slightly exceeded for higher latency cases on one core per node; the contributing factors to this are under investigation. This graph illustrates the trend that as network latency increases, the ability to overlap communication and computation also increases for non-blocking programs. The highest latencies maintain a relatively constant performance gain, consistent with Figure 2.

is that more cores present would cause a steady decrease in performance gain due to increased contention. Indeed, for most of the latency curves this behavior is observed. For the highest latencies, however, this effect is suppressed by the lack of available work (i.e. more tasks will spend more time on the network). This helps suppress the effects of contention because this decreases the probability that multiple tasks will be attempting to access the shared memory resource simultaneously. The sudden drop-off from the plateaus occurs at the point that the detrimental effects of contention become greater than the suppression of the performance gain due to starvation.

A portion of the results of the final set of simulations can be seen in Figure 5. The plot provided demonstrates trends with a constant latency of 1024 reg-ops. The results of simulations with different latencies all took on a similar appearance; higher latencies would merely shift the plot to the right while scaling up the performance gain.

This plot of this specific latency reveals several trends in the performance gain for non-blocking computational models. The first trend is the plateau in performance gain upon reaching a certain level of parallelism. This is the same effect as seen in Figure 3. Specifically, this plateau

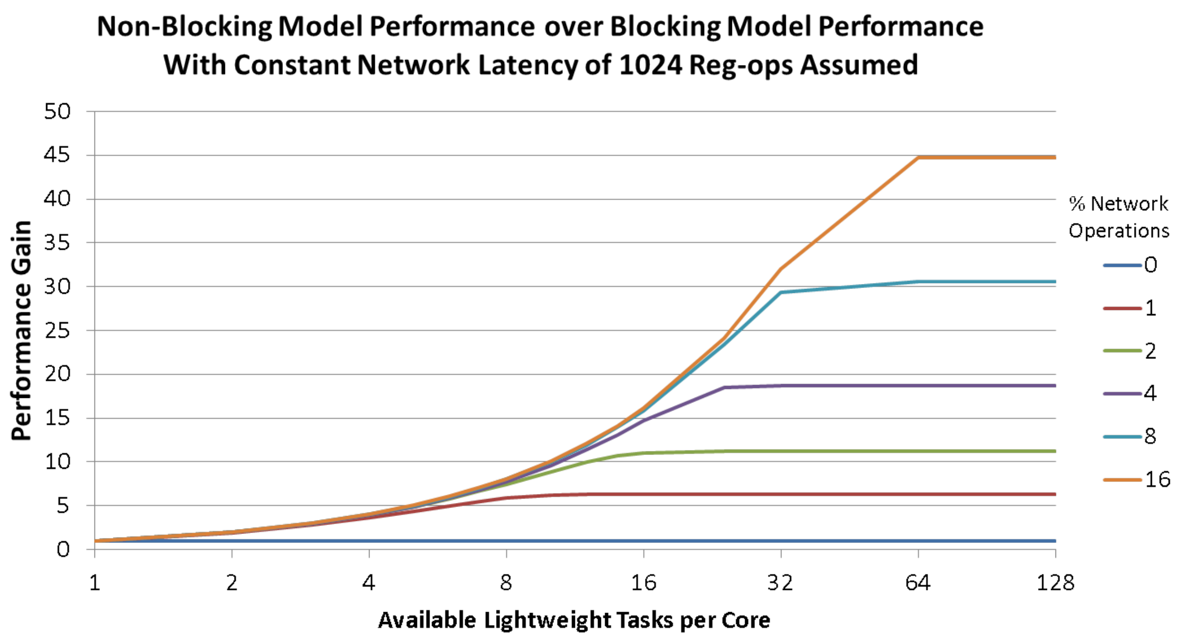


**Figure 5.** This figure represents the performance gain of a non-blocking program over a blocking program. Each graph plots the effects of changing the number of available tasks per core for various instruction mixes (each mix contains 68% register operations; the percentage of network operations is listed and the remaining percentage is memory operations). The network latency is equivalent to 1024 register operations representing a very fast network. The context switching overhead for non-blocking simulations is consistently 16 register operations. Only single core data were used for this plot. Each curve approaches an asymptote in performance gain near the point where the product of the number of available tasks and the context-switching overhead equals the network latency. This asymptote is consistent with Figure 3.

occurs when the product of the context-switch overhead and the number of available tasks is greater than the network latency. Even if all operations were network operations, at this point no further gain could be achieved by adding more available tasks to the system.

However, as indicated by the curves in Figure 5, there is an additional factor at play in determining where exactly this plateau begins. This factor is the instruction mix. For instruction mixes with fewer network operations, the gain curve appears to flatten earlier. This is because the probability that network operations will occur is low and thus the system is less likely to benefit from having more tasks ready to be run. In fact, it is the instruction mix that determines how rapidly the curves taper off to a plateau in all of Figures 2, 3, 4, and 5. Additionally, the instruction mix also determines the number of cores per node at which the contention for shared memory becomes noticeably detrimental. In general, however, it is the combination of the effects of starvation, latency, and overhead that determine the value of the plateau in performance gain.

## 6. How Do Major Execution Models Address SLOW

Table 1 examines several established execution models in terms of the SLOWER performance model but without energy and reliability (SLOW). Each entry of the table describes specific paradigm each execution model employs to addresses the corresponding source of performance degradation.

**Table 1.** A comparison of different execution models in terms of SLOW

| | **Starvation** | **Latency** | **Overhead** | **Contention** |
|---|---|---|---|---|
| Sequential issue | Restricted to single instruction stream | Memory hidden by cache hierarchy | Avoidance; no parallelism to control | Only one request at a time |
| Vector | Fine grain scalar operations making up vector | Hide latency through pipelining | Amortize control across vector elements | Pipelined data flow movement |
| SIMD[†] | Data parallelism | Accesses of separate ALUs[*] to local data | Control amortized through broadcast | Overlap out of phase memory bank access |
| CSP[‡] | Process level parallelism | Coarse computation phases offset the effects of communication | Minimized by static scheduling | One request at a time for local data |

[†] Single Instruction, Multiple Data.

[‡] Communicating Sequential Processes.

[*] Arithmetic Logic Units.

As enabling technologies evolve, different approaches to their use must be developed to exploit the new opportunities they afford and to address the challenges they impose. The history of high performance computing is punctuated with a series of such paradigm shifts reflected by a succession of execution models, each responsive to the changes in technology properties as characterized by the factors of the SLOWER performance model. In the current decade of the post-VLSI (Very Large Scale Integration) era the technology demands system architectures comprising multiple cores per socket as the principal means of enhancing scalability either as a homogeneous array of cores or in compound throughput structures for specific favorable data flows. In addition to the need for medium grain parallelism, eventually at the billion-way level for the Exascale performance regime, the new architectures will require deep memory hierarchies, high and varying communication latencies, limitations on energy consumption, asynchrony of operation, and dynamic adaptive methods of resource management and task scheduling to exploit runtime information for improved efficiency. Other aspects of future execution models need to work effectively with lower average memory capacity per core and efficient lightweight communication access patterns.

A class of emerging experimental execution models has been evolving to address these technology challenges through the synthesis of a number of complementary semantic constructs. The ParalleX execution model [7], briefly discussed below, is representative of these models although it is, by no means, the only example.

ParalleX is an experimental many-tasking execution model that supports message-driven computation and uses dynamic adaptive methods to manage asynchrony and enable scalable operation on large systems. It achieves that by exploring synergistic effects arising from interactions of its primary semantic components, which include:

- *Locality* - an encapsulation of resources in a synchronous domain that guarantees bounded access and service request time as well as compound atomic sequences of operations;
- *Global Name Space* - provides the semantic means of accessing any first class objects in a physically distributed application;
- *ParalleX Processes* - an abstraction of distributed context hierarchy integrating data objects, actions, task data, and mapping data;
- *Compute Complex* - a generalization of thread concept;
- *Local Control Objects* - provide synchronization, management of parallelism, and migration of continuations;
- *Parcels* - implement message-driven computing that combines data transfer and event-based information using a variant of active messages to permit the management of distributed flow control in a context of asynchrony.

## 7. How ParalleX Addresses SLOW

The development of the ParalleX execution model is driven to a significant degree by the factors incorporated in the SLOWER performance model. ParalleX is intended, as were previous models before it, to mitigate the causes of performance degradation to improve both efficiency and scalability. In addition, ParalleX is being extended to respond to energy and reliability concerns. These latter two factors are not discussed in this paper to any extent and are only included for sake of completeness. Here each is considered in terms of the ParalleX strategy employed to mitigate it.

### 7.1. Starvation

Starvation is perhaps the principal limiting condition for extreme-scale computing. It is the insufficiency of concurrent available application work to fully utilize the physical processing resources. It is made more complicated by the distribution of the pending work across the separate system resources. Even with sufficient total concurrency, starvation can occur due to poor local allocation of work to resources. ParalleX addresses the challenges contributing to starvation by increasing the amount of parallelism available and facilitating work distribution. To expose and exploit more parallelism than conventional practices in a single unified model ParalleX incorporates coarse, medium, and fine grain parallelism semantics.

Coarse-grained parallelism is represented as ParalleX processes, which can span multiple localities (nodes), share nodes with other processes, and migrate if necessary. Processes provide the hierarchical framework for work and state dynamic distribution across system resources. Processes can have descendant processes in a dynamic tree of parent-child relationships, by which they provide coarse parallelism.

ParalleX organizes the actual work it performs as medium grain compute complexes, which often are performed as threads. The use of medium grained threads can expose much more parallelism than pure coarse grain models and permits context switching for non-blocking of physical resources. Local Control Objects provide declarative constraint based synchronization and scheduling to increase parallelism by permitting sophisticated control relationships and support dynamic adaptivity.

Fine grain parallelism is given in terms of static data flow control semantics within the context of a compute complex. This is a generalization of the myriad ad hoc approaches currently used for Instruction Level Parallelism and permits translation to efficient forms of ILP for individual core architectures. Near fine grain parallelism is also exposed for usually remote compound atomic sequences of operations to exploit Remote Direct Memory Access (RDMA) techniques and smart networks.

Parallelism is sometimes simplistically described as either data parallelism or control parallelism. ParalleX treats these as aspects of a more general continuum in which a combination of these two can lead to runtime parallelism discovery based on meta-data for such structures as time varying irregular graphs. As discussed shortly, the reduction of overheads permitted by ParalleX allows finer granularity to be effectively exploited increasing the total amount of parallelism available, even for fixed size (strong scaled) applications. By providing powerful but focused synchronization methods (LCO), ParalleX largely eliminates over constraining global barriers, enabling overlap of successive phases of an evolving computation and exposing this additional form of parallelism. Together these principles embodied within the ParalleX model provide generality and flexibility both in size and form to maximize scalability.

## 7.2. Latency

Latency is the time for remote access and service requests resulting from distance of data movement. The primary effect of latency to performance is the blocking of the use of physical resources waiting for responses to remote data access requests and services. ParalleX addresses latency throught both introduction of mechanisms to reduce its magnitude and hiding its effects. ParalleX compute complexes are dynamic and can be exchanged in their use of physical resources. This context switching permits a thread that is waiting for a long latency request to be replaced by a pending task ready to do work, overlapping computation with communication. Multithreading of work trades parallelism for latency hiding. Parcels move work to the data, not always requiring data to be gathered to a fixed location of control. This allows migration of continuations (control state) to places where data access will be most intense, significantly reducing the latencies of the data accesses through the reduction of number and size of global messages. The use of local control objects provides mechanisms for event-driven control simultaneously addressing latency of action at a distance and managing the uncertainty of asynchrony of long latency operation.

## 7.3. Overhead

Overhead is the extra work performed to manage resources and support task scheduling. It is wasteful of time and energy compared to sequential execution. ParalleX within the constraints of hardware capability reduces the overheads and their effects found with conventional practices. It does this through the definition of focused control conditions, LCO, that support event driven computation control to minimize wasted work and avoid overly constraining semantics. Chief among these is the global barrier, which is almost all but eliminated through ParalleX. LCOs, benefitting from prior art, express rich semantics which has a powerful effect on parallel control state with minimum amount of overhead work. ParalleX encourages lightweight user threads in lieu of heavy weight OS threads (e.g., Pthreads) that are optimized to the needs of a specific application runtime to reduce the overhead of context switching time. To further avoid overheads, for very lightweight remote operations, ParalleX allows these to be specified and performed without incurring the overheads of compute complex instantiation.

### 7.4. Waiting Due to Contention

When more than one action requires access to a logical or physical resource at the same time, contention for the resource causes delays to all of the activities not given immediate access. ParalleX supports dynamic adaptive means by which, when viable, the runtime can allocate a different available resource with similar capabilities. Examples include allocation of memory banks, rerouting of network traffic where multiple paths exist, distribution of synchronization elements, multiple physical threads for task execution. The event-driven distributed flow control eliminates polling and reduces the number of sources of synchronization delays.

## 8. Conclusions

A performance framework has been introduced to facilitate development and optimization of extreme-scale abstract execution models and the future systems derived from them. SLOWER defines a six-dimensional design trade-off space based on sources of performance degradation that are derived from the physics of the underlying hardware systems and the control software. They are invariant across system classes in that they apply broadly and are shown here to invoke different system responses to changing enabling technologies. Looking forward, the new technology trends, which are leading to nano-scale and the end of Moore's Law, demand future innovations to address these same performance factors. An experimental execution model, ParalleX, is described to postulate one possible advanced abstraction upon which to base next generation hardware and software systems. A detailed examination is presented of how this class of dynamic adaptive execution models addresses each of the performance factors of SLOWER, opening a new path to highly efficient and scalable system design. There is not yet a unified formal analytical relation of the SLOWER performance framework, although it is an objective of this work to lead to such a theory. An alternative approach is to represent the trade-off space through a queue model and employ simulation methods spanning ranges of key parameters to expose the properties of the SLOWER framework. While this research path is still in progress, early results shown here demonstrate aspects of the parameterized design regime consistent with real world experience. These promising results also suggest a means to empirically derive the sensitivities of the interrelated factors. The immediate future work is to more extensively explore the SLOWER trade-offs through enhanced versions of the queuing simulation and use this to derive and validate an analytical formalism.

## 9. Acknowledgements

This work was supported in part by the Department of Energy under Award Numbers DE-SC0008809 and DE-NA0002377, and the Department of Defense High Performance Computing Modernization Program under Contract Number GS04T09DBC0017.

## References

1. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model — one step closer towards a realistic model for parallel computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 95–105, New York, NY, USA, 1995. ACM. ISBN 0-89791-717-0. DOI: 10.1145/215399.215427.

2. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. DOI: 10.1145/1465482.1465560.

3. G. E. Blelloch. *Vector Models for Data-parallel Computing.* MIT Press, Cambridge, MA, USA, 1990. ISBN 0-262-02313-X.

4. K. W. Cameron, R. Ge, and X.-H. Sun. logNP and log3P: Accurate analytical models of point-to-point communication in distributed systems. *IEEE Trans. Comput.*, 56(3):314–327, Mar. 2007. ISSN 0018-9340. DOI: 10.1109/TC.2007.38.

5. C.-K. Chui. The LogP and MLogP models for parallel image processing with multi-core microprocessor. In *Proceedings of the 2010 Symposium on Information and Communication Technology*, SoICT '10, pages 23–27, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0105-3. DOI: 10.1145/1852611.1852616.

6. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM. ISBN 0-89791-589-5. DOI: 10.1145/155332.155333.

7. C. Dekate, M. Anderson, M. Brodowicz, H. Kaiser, B. Adelstein-Lelbach, and T. Sterling. Improving the scalability of parallel N-body applications with an event-driven constraint-based execution model. *International Journal of High Performance Computing Applications*, 26(3):319–332, 2012. DOI: 10.1177/1094342012440585. URL http://hpc.sagepub.com/content/26/3/319.abstract.

8. R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, SC-9(5):256–268, 1974.

9. D. Eppstein and Z. Galil. Parallel algorithmic techniques for combinational computation. *Annual Review of Computer Science*, 3(1):233–283, 1988. DOI: 10.1146/annurev.cs.03.060188.001313. URL http://dx.doi.org/10.1146/annurev.cs.03.060188.001313.

10. F. Ino, N. Fujimoto, and K. Hagihara. LogGPS: A parallel computational model for synchronization analysis. *SIGPLAN Not.*, 36(7):133–142, June 2001. ISSN 0362-1340. DOI: 10.1145/568014.379592.

11. R. Pasupathy, S. Kim, A. Tolk, R. Hill, and M. Kuhl. Open-source simulation software "JAAMSIM".

DOI: 10.14529/jsfi140204

# Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations

***Torsten Hoefler***[1] ***and Dmitry Moor***[1]

## 1. Introduction

Collective operations are among the most important communication operations in shared- and distributed-memory parallel applications. In this paper, we analyze the tradeoffs between energy, memory, and runtime of different algorithms that implement such operations. We show that existing algorithms have varying behavior and that no known algorithm is optimal in all three regards. We also demonstrate examples where of three different algorithms solving the same problem, each algorithm is best in a different metric. We conclude by posing the challenge to explore the resulting tradeoffs in a more structured manner.

The performance of collective operations often directly affects the performance of parallel applications significantly. Thus, many researchers designed fast algorithms and optimized implementations for various collective communication operations. The newest version of the Message Passing Interface (MPI) standard [30], the de-facto standard for distributed-memory parallel programming, offers a set of commonly-used collective communications. These operations cover most use-cases discovered in the last two centuries and we thus use them as a representative sample for our analyses. In general, collective patterns reflect key characteristics of parallel algorithms at large numbers of processing elements, for example, parallel reductions are used to implement parallel summation and alltoall is a key part of many parallel sorting algorithms and linear transformations.

Recent hardware developments in large-scale computing increase the relative importance of other features besides pure execution time: energy and memory consumption may soon be key characteristics. Minimizing *energy consumption* is especially important in the context of large-scale systems or small battery-powered devices. *Memory consumption* is important in systems that offer hardware to support the execution of collective operations. Here, we assume state-of-the-art *offloaded* execution models (e.g., [23]) where communication schedules are downloaded into the network device that operates with severely limited resources. The increasing availability of such offload architectures motivates us to model the memory consumption of offloaded collective communications.

In this work, we provide an overview and a classification of state-of-the-art algorithms for various collective operations. Our report is not meant to cover all possible algorithms for implementing collective operations of which there are far too many to fit in the space limitations of this short article. Instead, our classification and analysis shall establish a discussion basis for the fundamental tradeoffs between runtime, energy, and memory consumption. For each algorithm, we derive analytic models for all three key metrics. Our theoretical study shows, for example, that reducing the number of messages sent may reduce the performance but, at the same time, decrease energy and memory consumption. Furthermore, our analysis of existing algorithms allows us to point out gaps and define future research topics. In general, we argue for a more general design mechanism that considers the multi-objective optimization problem for time, energy, and memory.

[1]ETH Zürich, Zürich, Switzerland

## 2. Architectural Energy, Runtime, and Memory Models

Good architectural models strike the balance between minimizing the number of parameters and modeling the architecture's main effects accurately. A small number of parameters facilitates reasoning about algorithms and algorithm design and simplifies the optimization problems in the context of real applications. However, models need to capture the main parameters that determine the performance of the implementation on the target architecture. Several such models for the performance of communication algorithms have been designed. The most prominent ones belong to the LogP family while many other models can either be expressed as subsets of LogP (e.g., alpha-beta) or have a similar character but increase the complexity of the parameters, (e.g., PlogP [25]). For the purpose of this paper, we use LogGP [1] as a model for the execution time because we believe that it expresses the most relevant architecture parameters while still allowing elegant formulations of optimization problems.
We now proceed to discuss several communication technologies and mechanisms in the context of collective algorithms and the LogGP model.

**Message Passing** Message Passing is the basis of the design of LogGP. Here, **L** denotes the maximum communication latency between two endpoints. The parameter **o** represents the constant CPU overhead for sending or receiving a single message, e.g., the call to the message passing library. The parameter **g** is the equivalent overhead for sending or receiving a message caused by the network interface. The maximum of o and g limits the small-message injection rate, an important parameter of current interconnection networks. The model also implies that only $L/g$ messages can be in flight between two processes at any time. The parameter **G** models the cost per injected Byte at the network interface, this is the reciprocal bandwidth. Finally, the number of processes is represented by **P**.

**Noncoherent Shared Memory** Noncoherent shared memory systems as used in remote direct memory access (RDMA) communications or for the data transfer between CPUs and GPUs are similar to message passing systems. The typical programming interface to such systems are put and get operations that store into or load from remote memory. The main difference to message passing is that the receiver is not explicitly involved and thus $o$ is not charged at the destination. However, all other parameters remain. For the purpose of this article, we ignore this discrepancy with the traditional LogGP model.

**Coherent Shared Memory** Coherent memory systems are slightly more complex. Coherence between multiple caches is often guaranteed by a cache coherence protocol operating on blocks of memory (e.g., cache lines). The protocol ensures that each block always holds exactly one value in the whole system. Such protocols often allow for multiple readers (i.e., multiple identical copies of the block) but each write access requires exclusive ownership. Since all communication is implicitly performed during standard load/store accesses, performance characteristics are more complex and LogGP is only an approximate model for such transfers in the general case. Yet, if the amount of sharing is low (i.e., data is transferred from each writer to a single reader), then LogGP can model the performance characteristics accurately. Ramos and Hoefler [35] provide a detailed explanation of the intricacies of modeling for cache-coherent systems and related work.

**Network Offload Architectures** Some newer network architectures such as Portals IV [7] or CORE-Direct [14] allow to offload collective operations to the network device. This enables faster execution (messages do not need to travel to the CPU) and isolation (computations on the CPU and collective communications do not interfere and can progress independently). This reduces the impact of small delays on the CPU, often called system noise [20, 47] and allows asynchronous execution of nonblocking collective operations [18]. Communications are performed using messages and can thus be modeled using the LogGP model. Offload devices have limited resources to store communication schedules and we model the memory consumption of each algorithm in such devices.

**Runtime Models** We will use LogGP to model the approximate runtime of the algorithms on all target systems. Furthermore, in order to keep the models interpretable, we set $o > g$ and assume that the LogGP CPU overhead $o$ is also charged in offloading devices so that we never need to charge $g$ ($o$ for offloading devices is most likely much smaller than $o$ on a general-purpose CPU). We also assume that the cost to transmit a message of size $s$ is $T_{\mathrm{msg}} = L + 2o + sG$. We report the maximum finishing time that any process needs.

**Energy Models** Energy consumption can generally be split into two components: dynamic and static energy [28, 29]. The static energy is the leakage energy during the operation of an electronic device, regardless of the device's activity. Dynamic energy represents the energy that is consumed by activities such as computation, sending and receiving messages, or memory accesses. For the purpose of our analysis, we assume that computation and local memory operations (e.g., shuffling data) are free. These assumptions are similar to the LogGP model which also only considers network transactions. To model the energy for communication, we assume that each message consumes a fixed energy $\mathbf{e}$. This represents the setup cost to send a zero-byte message and is similar to $o$ and $g$ in the LogP model, we do not separate CPU and network costs because energy consumption is additive and can thus be captured by a single parameter. Furthermore, we denote the energy required to transport each byte from the source's memory to the destination's memory as $\mathbf{E}$, similar to LogGP's $G$ parameter. This model assumes a fully connected network such that the energy consumption does not depend on the location of the source and destination. Thus, ignoring local computations, the total energy consumption of a collective operation is $L = T \cdot P + D$ where $T$ is the runtime (e.g., modeled by LogGP), $P$ is the leakage power, and $D$ is the dynamic energy model. In our analysis, we derive dynamic energy models for the overall operation (the sum of all dynamic energies consumed at each process).

**Memory Models** Similarly, we derive a simple model for capturing memory overheads for offloading devices. To offload a collective operation to a network device, one copies some state (e.g., a set of triggers [7] or a set of management queue entries [14]) that models the execution schedule to the device. The device then generates messages based on arriving messages from other processes and the local state without CPU involvement. Here, we assume that each sent message has to be represented explicitly as a descriptor in the offloaded operation. We assume that these descriptors have the constant size $\mathbf{d}$. This descriptor size does not depend on the size of the actual message to be sent or received. We report the maximum memory needed by any process.

## 3. Implementation Strategies for Collective Operations

Instead of describing algorithms for specific collectives, we discuss common algorithms to implement collective operations. For each of these algorithms, we develop runtime, energy, and memory overhead models. We then proceed to briefly describe each of MPI's collective operations and discuss how the algorithms can be used to implement it. This method reflects the state-of-the-art in which collective libraries often implement a set of algorithm skeletons and match them to particular collective implementations [12].

### 3.1. Existing Collective Algorithms

Each collective algorithm exploits a particular *virtual topology*, i.e., a directed graph representing message propagation between processes. We distinguish between three classes of collective algorithms: (1) trees in various shapes and forms, (2) distribution algorithms, and (3) specialized algorithms.

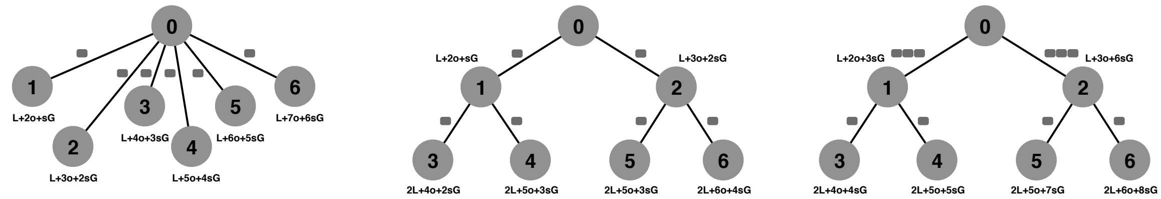
Trees can be used to implement any collective communication. In these algorithms, processes are arranged in a tree shape and messages are flowing from parents to children or vice versa, depending on the collective operation. Some collectives require personalized data (e.g., scatter/gather) such that the messages grow or shrink as they are sent along the tree while other operations either replicate or reduce the data (e.g., reduce, broadcast) leading to constant-size messages. Trees are often used for communicating small messages because in most cases, leave processes only receive messages and are thus not able to use their own send bandwidth. Simple pipelines (i.e., degenerated regular trees) that minimize the number of leaves often provide excellent and simple solutions for very large message sizes. We will also discuss double-tree algorithms that improve the latency over such simple pipelines.

While trees can be used to implement any collective, they may incur a higher cost if they need to be combined. For example, unrooted collectives where all processes receive the result (e.g., allreduce) require communication up and down a tree. These communications can be efficiently implemented using distribution patterns that can also be seen as intertwined trees rooted at each process. A third class of specialized algorithms takes advantage of either specific hardware properties such as topology or multicast semantics or specific semantics of the collective problem.

We now proceed to describe existing tree algorithms followed by distribution patterns. We conclude this subsection by referencing several specialized algorithms. A simple lower bound for the runtime of all algorithms is $\Omega(o \log P) + sG$ because data needs to reach all processes and data must be sent at least once. Similarly, a lower bound to the energy consumption is $(P-1)(e+sE)$ and a lower bound for the memory consumption is $d$ because each process must receive the data once. We will provide exact and simplified models for each algorithm; the simplified models use mixed asymptotic notation for $s \to \infty$ and $P \to \infty$ to facilitate highest intuition.

#### *3.1.1. Flat Tree Algorithms*

We start with the simplest algorithm for collective operations—a flat tree (FT) [25] in which a single processor sends messages to all destinations directly. Figure 1a provides an example of such a tree for a non-personalized or personalized operation. The gray squares at communication edges denote the communicated data of size $s$. The annotations in this and the following figures denote the finishing times of the processes in the example. In all figures, we assume that data is sent to the children of a process in the order drawn, beginning with the leftmost. Though simplicity of



a) Flat tree (both). b) Binary tree (non-personal). c) Binary tree (personal).

**Figure 1.** Flat and binary trees ($k = 2$) with seven processes ($P = 7$) in personal and non-personal configurations.

the algorithm is a clear advantage, its sequential communication limits performance. The time to perform such an operation (personalized or not) is $T_{\text{FT}} = L+oP+sG(P-1) = (o+sG)P-\mathcal{O}(s)$ in the LogGP model. The dynamic energy consumption of such a communication can be estimated as $D_{\text{FT}} = (P-1)(e+sE) = P(e+sE) - \mathcal{O}(s)$. The maximally needed storage at the root of the tree is $M_{\text{FT}} = d(P-1)$.

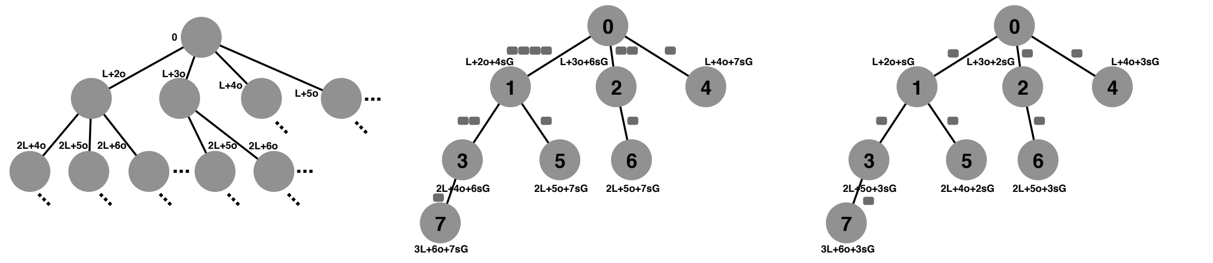
### *3.1.2. Regular Trees*

A widely used topology for rooted collective operations is based on regular trees. In such trees, processes perform communications concurrently and thus achieve better performance than flat trees. Trees are called regular when each inner process has the same number of child nodes. We call trees with $k$ such children per process $k$-ary trees; in this sense, flat trees can be seen as regular trees with $k = (P-1)$.

To illustrate the concept, Figures 1b and 1c show non-personalized and personalized communications along a binary tree, respectively. General $k$-ary trees (KT) require $\log_k(P)$ total parallel communication steps. In particular, the time of a $k$-ary tree algorithm for a non-personalized operation is $T_{\text{KT}} = (L + k(o + sG) + o)\lfloor\log_k P\rfloor = (L + ko + ksG)\log_k P - \mathcal{O}(s) + \log_k P \cdot \mathcal{O}(1)$. The dynamic energy model for the same algorithm is $D_{\text{KT}} = (P-1)(e+sE) = P(e+sE)-\mathcal{O}(s)$. The storage requirements for $k$-ary trees are $M_{\text{KT}} = kd$ because each process sends to at most $k$ children.

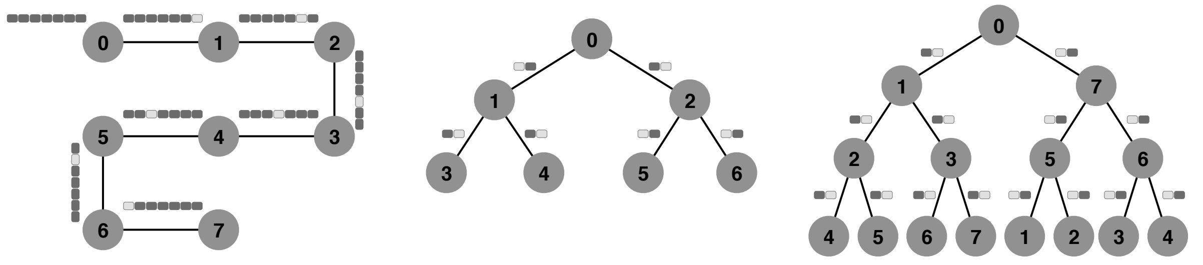
For personalized communications on full trees (which we mark with a tilde above the virtual topology type, e.g., $\widetilde{\text{KT}}$), the communication time can be modeled with $T_{\widetilde{\text{KT}}} = \lfloor\log_k P\rfloor(L+o(k+1))+sG\sum_{i=0}^{\lfloor\log_k P\rfloor}(\lfloor\log_k P\rfloor-i)k^i = (L+ko)\log_k P+sGP\cdot\mathcal{O}(1)+\mathcal{O}(\log P-s)$. Here, one can simply count the packet along the rightmost path assuming that messages are sent to each left child first. The dynamic energy consumption is $D_{\widetilde{\text{KT}}} = e(P-1)+sE\cdot k^{\lfloor\log_k P\rfloor}\sum_{i=0}^{\lfloor\log_k P\rfloor-1}(\lfloor\log_k P\rfloor - i)\frac{1}{k^i} \approx P(e+sE\log_k P) + \mathcal{O}(sP)$ (for large $k$) and the memory consumption is $M_{\widetilde{\text{KT}}} = kd$ as in the non-personalized case.

Pjesivac-Grbovic et al. [33] use splitted binary trees (SB) to accelerate non-personalized communications. They use a normal binary tree but instead of distributing the whole message along each tree edge, the message is divided into two parts. The first part is sent to the nodes of the left subtree of the root, while the second part is distributed among nodes of the right subtree of the root. Once a node received the data and sent it on to its children, it also sends it to its own counterpart in the other subtree. The approximate time of the splitted binary tree algorithm is a combination of the normal binary tree non-personalized algorithm with $\frac{s}{2}$ data and a full



a) Optimal tree (non-personal). b) Binomial tree (personal). c) Binomial tree (non-personal).

**Figure 2.** Optimal Fibonacci trees and binomial trees with eight processes ($P = 8$) in personal and non-personal configurations.



a) Linear pipeline ($P = 8$). b) Tree pipeline ($P = 7$). c) Double tree ($P = 8$).

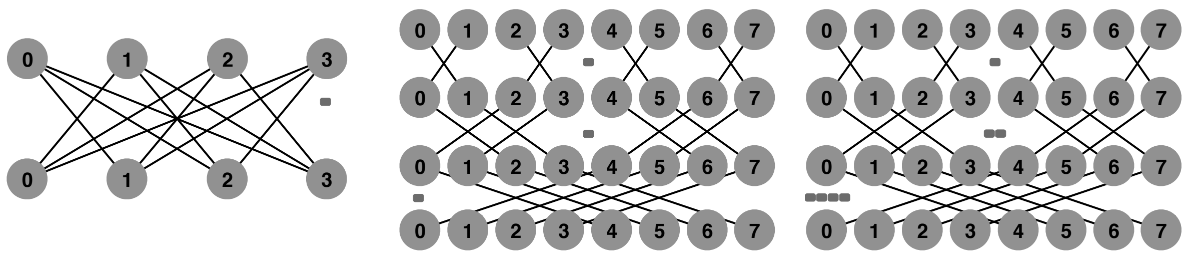
**Figure 3.** Non-personalized pipelined trees and double trees with seven or eight processes.

exchange: $T_{\mathrm{SB}} = (L+2(o+\frac{s}{2}G)+o)\lfloor\log_2 P\rfloor+2o+L+\frac{s}{2}G = \log_2 P(L+3o+sG)+s\log_2 P\cdot\mathcal{O}(1)$. The estimated dynamic energy for this algorithm is $D_{\mathrm{SB}} = 2(e+\frac{s}{2}E)(P-1) = P(2e+sE)-\mathcal{O}(s)$ while the memory model is $M_{\mathrm{SB}} = 3d$.

### *3.1.3. Irregular Trees*

While simplicity of regular tree algorithms is a strong advantage and they are asymptotically optimal for small messages, they are generally not strictly optimal. For example, Karp et al. [24] demonstrate that Fibonacci trees are optimal for single-item broadcasts and thus non-personalized tree communication in the LogP model. Figure 2a shows the optimal tree construction, each node is labeled with its arrival time and the best broadcast tree for $P$ processes is constructed from the $P$ nodes with the smallest labels. Karp et al. also state that, if $f_n$ and $f_{n+1}$ are the consecutive members of the generalized Fibonacci sequence s.t. $f_n < P-1 < f_{n+1}$, the lower bound for broadcasting $s$ items is $n+1+L+(s-1)-\lfloor\sum_t \frac{f_t}{P-1}\rfloor$ [24] (assuming $g = 1$, $o = 0$, $G = 0$). For personalized tree communication, Alexandrov et al. [1] as well as Iannello [22] show that in the LogGP model the usage of irregular trees for virtual topologies allows to achieve better performance. Both algorithms are hard to derive and have not been used in practice to the best of the authors knowledge.

A much simpler class of irregular trees that improves over regular trees are $k$-nomial trees. Here, we discuss the most-used binomial tree (BT) ($k = 2$) as example and we assume that $P$ is a power of two. The runtime of non-personalized binomial trees is $T_{\mathrm{BT}} = (L+2o+sG)\log_2 P$, their dynamic energy consumption is $D_{\mathrm{BT}} = (P-1)(e+sE) = P(e+sE)-\mathcal{O}(s)$, and their memory use is $M_{\mathrm{BT}} = d\log_2 P$ at the root process. The runtime of personalized binomial trees is $T_{\widetilde{\mathrm{BT}}} = (2o+L)\log_2 P+sG(P-1) = (2o+L)\log_2 P+sGP-\mathcal{O}(s)$, their dynamic energy consumption is $D_{\widetilde{\mathrm{BT}}} = e(P-1)+sE\frac{P}{2}\log_2 P = Pe+sE\frac{P}{2}\log_2 P-\mathcal{O}(1)$, and their memory



a) Direct sends (both). b) Butterfly (non-personal). c) Butterfly (personal).

**Figure 4.** Different distribution algorithms for unrooted collectives. Only one data packet is shown at each stage for readability.

consumption is $M_{\widetilde{\mathrm{BT}}} = d \log_2 P$. Figures 2b and 2c show examples for personalized and non-personalized binomial trees.

Binomial tree algorithms are commonly used for small messages; for larger messages, more complex algorithms provide better results (see, for example, various algorithms proposed by Van de Geijn et al. [6, 41, 44]). We will now discuss pipelined trees that have a similar goal to improve bandwidth.

### *3.1.4. Pipelined Tree Algorithms*

Pipeline algorithms are based on the idea to divide a large message into multiple small pieces and to distribute these pieces among processors in a pipeline fashion [33, 38]. Here, different virtual topologies can be utilized for transmitting the data. Linear pipelines as illustrated in Figure 3a are simplest while tree pipelines as illustrated in Figure 3b allow to reduce latencies. As before, our models assume that data is sent down the left pipe first and then alternating. We also assume in this case that the send and receive overheads ($o$) can be charged simultaneously (e.g., in a multicore environment). Pipelines are often used as building blocks for more complex algorithms [40]. For example, in a non-personalized setting, the runtime of a pipelined binary tree (PBT) algorithm can be estimated as $T_{\mathrm{PBT}} = 2(o + \frac{s}{N}G)N + L + o + (o + 2(o + \frac{s}{N}G) + L)(\lfloor \log_2 P \rfloor - 1) = \log_2 P(L + 3o + \frac{sG}{N}) + \mathcal{O}(N + s)$, where $N$ is the number of pieces into which the message is divided and it typically depends on $s$. The corresponding dynamic energy model is $D_{\mathrm{PBT}} = (P-1)(e + \frac{s}{N}E)N = P(Ne + sE) - \mathcal{O}(s)$ and the storage requirement is $M_{\mathrm{PBT}} = 2dN$. For personalized communications, the runtime is $T_{\widetilde{\mathrm{PBT}}} = o + (o + (2^{\lfloor \log_2 P \rfloor} - 1)\frac{s}{N}G)2N + L + (o + L)(\lfloor \log_2 P \rfloor - 1) + 2(o(\lfloor \log_2 P \rfloor - 1) + \frac{s}{N}G(2(2^{\lfloor \log_2 P \rfloor - 1} - 1) - (\lfloor \log_2 P \rfloor - 1))) = (L + 3o)\log_2 P + sG(P\mathcal{O}(1) - \frac{2}{N}\log_2 P) + N\mathcal{O}(1)$ , the dynamic energy consumption is $D_{\widetilde{\mathrm{PBT}}} = N(2e(2^{\lfloor \log_2 P \rfloor} - 1) + \frac{sE}{N}2^{\lfloor \log_2 P \rfloor}(1 + 2(\lfloor \log_2 P \rfloor - 1) + (2^{1 - \lfloor \log_2 P \rfloor} - 1))) = NP\mathcal{O}(1)e + sEP\mathcal{O}(1)(\log_2 P + \frac{1}{P\mathcal{O}(1)} - \mathcal{O}(1))$, and the memory overhead is $M_{\widetilde{\mathrm{PBT}}} = 2Nd$.

### *3.1.5. Double Trees*

While pipelined trees improve the overall bandwidth utilization, they are still not optimal. The reason for this is that the leaves in the tree never transmit messages and thus do not contribute their bandwidths. To use the leaves' bandwidth, one can employ two trees with different structure (leave nodes) such that each node sends eventually. Sanders and Träff [39, 40] demonstrate such a two-tree virtual topology that achieves full bandwidth, extending and simplifying an earlier algorithm [45]. The authors utilize two trees so that the interior nodes of the first tree correspond to the leaf nodes of the second tree and vice versa (see Figure 3c). They also describe

a scheduling algorithm to define from which parent node the data should be received at the current step and to which child node the data should be forwarded. The approach only applies to non-personal communications. The runtime of this double tree (DT) algorithm is $T_{\mathrm{DT}} = (L + 3o + sG) + T_{\mathrm{PBT}}\left(\frac{s}{2}\right)$. Its dynamic energy consumption is $D_{\mathrm{DT}} = 2(e + \frac{s}{2}E) + 2D_{\mathrm{PBT}}\left(\frac{s}{2}\right)$ and the memory consumption for this approach is $M_{\mathrm{DT}} = 2dN$.
This algorithm concludes our treatment of successively more complex algorithms for rooted collective communications. We now proceed to discuss distribution patterns such as direct send, dissemination, and butterfly algorithms for unrooted collective communications.

#### *3.1.6. Direct Sends*

In unrooted collectives, typically all processes receive some data from every other process, either personalized or reduced. This can be achieved by a direct send (DS) topology among all processes. This is similar to a flat tree rooted at each process. The runtime for the personalized as well as the non-personalized variant is $T_{\mathrm{DS}} = L + (P-1)(o + sG) = P(o + sG) - \mathcal{O}(s)$, the energy consumption is $D_{\mathrm{DS}} = P(P-1)(e + sE) = P^2(e + sE) - \mathcal{O}(Ps)$, and the memory consumption at each process is $M_{\mathrm{DS}} = (P-1)d$. Figure 4a illustrates the DS scheme.

#### *3.1.7. Dissemination and Butterfly Algorithms*

The well-known Butterfly (BF) graph [8] implements a binary scheme to quickly exchange data among all processes which can be applied if $P$ is a power of two. The dissemination approach [15] generalizes this scheme to arbitrary numbers of processes. Here, we limit ourselves to the simpler case where $P$ is a power of two. In the Butterfly pattern, data is communicated between processes with exponentially growing distances, i.e., in the $k$-th step, nodes at distance $2^k$ from each other exchange data. Thus, $\log_2 P$ steps are required to complete the communication.
The non-personalized version of butterfly executes in time $T_{\mathrm{BF}} = (2o + sG + L)\log_2 P$, with a dynamic energy consumption of $D_{\mathrm{BF}} = (e + sE)P\log_2 P$, and with a memory consumption of $M_{\mathrm{BF}} = d\log_2 P$.
The well-known recursive doubling algorithm [44] as well as the Bruck algorithm [9] implement a personalized variant of the Butterfly pattern. If we ignore local data shuffles, then the runtime of this personalized algorithm is $T_{\widetilde{\mathrm{BF}}} = (2o+L)\log_2 P + Gs(P-1) = (2o+L)\log_2 P + sGP - \mathcal{O}(s)$. Its energy consumption can be modeled as $D_{\widetilde{\mathrm{BF}}} = eP\log_2 P + sE(P-1)P = P(e\log_2 P + sEP) - \mathcal{O}(sP)$ and its memory requirement is $M_{\widetilde{\mathrm{BF}}} = d\log_2 P$. Each model increases with a multiplicative constant if the number of processes is not equal to a power of two [44]. Figures 4b and 4c illustrate the Butterfly pattern with eight processes in non-personalized and personalized configurations, respectively.

#### *3.1.8. More Specific Algorithms*

Several researchers developed algorithms that are tuned to particular properties of the machine. For example, several algorithms that specialize to the network topology exist. Some others utilize special hardware features. We provide some examples here but this list is not meant to be complete.

**Hardware-specific algorithms** Ali et al. [2] provide algorithms for collective communications on the Cell B.E. chip, Panda et al. demonstrate a series of algorithms tuned to InfiniBand networks and RDMA systems [27, 42], and Almasi et al. [3] show optimization techniques for the BlueGene/L Torus network.

**Topology-aware algorithms** There is a class of algorithms that take the network topology and congestion into account. For example, Sack and Gropp [36, 37] introduce a congestion-aware model for network communication. In the same articles they propose a recursive-doubling distance-halving algorithms for the allgather and reduce scatter collectives for Clos and Torus networks. Payne et al. [32] describe several algorithms on how to implement some reduction operations on a 2-dimensional mesh and Barnett et al. [5] develop a broadcasting algorithm for the mesh topology. Watts and Van de Geijn [48] show a pipelined broadcast for mesh architectures and Chan et al. [10] show how to utilize all available links in Torus networks.

**Using Unreliable Multicast Hardware** Other algorithms base on special hardware features such as multicast [11]. Multicast packets can be lost and in order to guarantee reliable transmission, recovery algorithms are necessary. One such recovery protocol is presented by Hoefler et al. [19]. Their protocol combines InfiniBand (or Ethernet) unreliable multicast with reliable point-to-point messages to achieve a with high probability constant-time ($\mathcal{O}(1)$ complexity) broadcast operation. Using these special hardware features allows us to circumvent the logarithmic lower bound.

## 3.2. Implementing Collective Operations

We now briefly discuss how the modeled algorithms can be combined to implement collective operations. We follow our previous categorization into rooted collectives implemented by personalized or non-personalized trees and unrooted collectives implemented by personalized or non-personalized distribution algorithms.

### *3.2.1. Rooted Collectives*

Table 1 shows an overview of the tradeoffs in various personalized and non-personalized tree algorithms. We use the previously introduced subscripts as abbreviation: FT for flat trees, KT for $k$-ary regular trees, BT for binomial trees, PBT for pipelined binary trees, and DT for double trees. Abbreviations with a tilde on top, e.g., $\widetilde{\text{FT}}$, denote personalized versions of the algorithms.

| | FT, $\widetilde{\text{FT}}$ | KT | $\widetilde{\text{KT}}$ | BT | $\widetilde{\text{BT}}$ | PBT | $\widetilde{\text{PBT}}$ | DT |
|---|---|---|---|---|---|---|---|---|
| **T** | $P(o+sG)$ | $(L+ko+ksG)\log_k P$ | $(L+ko)\log_k P+sGP$ | $(L+2o+sG)\lg P$ | $(L+2o)\lg P+sGP$ | $(L+3o+\frac{sG}{N})\lg P$ | $sG(P-\frac{2}{N}\lg P)+(L+3o)\lg P$ | $(L+3o+sG)+T_{PBT}\left(\frac{s}{2}\right)$ |
| **D** | $P(e+sE)$ | $P(e+sE)$ | $P(e+sE\log_k P)$ | $P(e+sE)$ | $P(e+\frac{sE}{2}\lg P)$ | $P(Ne+sE)$ | $P(Ne+sE\lg P)$ | $2(e+\frac{s}{2}E)+2D_{PBT}\left(\frac{s}{2}\right)$ |
| **M** | $Pd$ | $kd$ | $kd$ | $d\lg P$ | $d\lg P$ | $2dN$ | $2dN$ | $2dN$ |

**Table 1.** Overview of tree algorithms for rooted collectives (minor terms are dropped, lg stands for $\log_2$).

**Broadcast/Reduce** Broadcast and reduce are structurally similar but very different in their semantics. In a broadcast, a single message of size $s$ is distributed (copied) from a designated root process to all other $P-1$ processes. In a reduction, each process contributes a message of size $s$. The associative (and often commutative) operator $\oplus$ combines all $P$ messages into a single result of size $s$ at a designated root process: $r = m_1 \oplus m_2 \oplus m_3 \oplus \cdots \oplus m_P$.
Both collectives can be implemented with non-personalized tree algorithms. Binomial and binary trees are commonly used for implementations of small-message broadcast and reduction [43, 44]. Large-message operations can be implemented with double trees. Our models in Table 1 show that, for non-personalized communications, double-trees are the best contenders in terms of runtime (for all $s$ and $P$). However, they require more dynamic energy and memory due to the pipelining of messages. The exact number of additional messages sent depends on the number of pipeline segments $N$, which in turn is chosen based on the LogGP parameters and $s$. If the memory is constrained, then pipelining would be limited, possibly leading to suboptimal performance. All non-pipelined algorithms are work-optimal and thus consume the minimal energy. Regular $k$-ary trees have only constant memory overhead and are thus best for execution in very limited offload settings.

**Scatter/Gather** In a scatter, a designated process (root) sends personalized messages, each of size $s$, to $P-1$ other processes. In a gather, the root process receives different messages, each of size $s$, from $P-1$ processes and stores them locally. Both collectives can be implemented using personalized tree algorithms. For example, Binomial trees have been used to perform both, scatter and gather [4].
Our models in Table 1 show that, for personalized communications with small $P$, flat trees are best. Other regular and irregular trees reduce the latency to a logarithmic term and thus benefit large $P$ but they are not work-optimal and send multiple messages multiple times and thus harm large $s$. For large $s$ and small $P$ one can use linear pipelines to utilize the bandwidth of all processes as discussed before. Alexandrov et al. [1] formulate the condition for an optimal gather tree in LogGP but to the best of the authors' knowledge, no practical algorithm is known that achieves this bound. In terms of energy, we remark that all tree algorithms increase dynamic energy consumption significantly in comparison to a flat tree. Memory consumption is similar to the non-personalized algorithms where the pipelining versions may dominate and $k$-ary regular trees are minimal for small $k$.

#### *3.2.2. Unrooted Collectives*

Table 2 shows an overview of various distribution algorithms and trees that can be used for unrooted collectives. We use the previously defined abbreviations for distribution algorithms: DS for direct send and BF for Butterfly. We compare these to implementations with two combined trees, such as a $k$-ary tree to reduce data towards a root followed by a second $k$-ary tree to broadcast data to all processes, which we denote as 2xKT. We only combine trees of similar nature and show some select examples even though combinations of any two trees can be used in practice.

**Allreduce/Barrier** Allreduce is similar to reduce in that all processes contribute a message of size $s$ and $r = m_1 \oplus m_2 \oplus m_3 \oplus \cdots \oplus m_P$ is computed. However, as opposed to reduce, the final $r$ will be distributed to all processes. The Barrier collective guarantees that no process completes

| | DS, $\widetilde{\text{DS}}$ | BF | $\widetilde{\text{BF}}$ | 2xKT | 2x$\widetilde{\text{KT}}$ | 2xPBT | 2x$\widetilde{\text{PBT}}$ |
|---|---|---|---|---|---|---|---|
| **T** | $P(o+sG)$ | $(L+2o+sG)\log_2 P$ | $\log_2 P(2o+L)$ $+sGP$ | $2(L+ko+$ $ksG)\log_k P$ | $2(L+ko)\log_k P$ $+2sGP$ | $2(L+3o$ $+\frac{sG}{N})\lg P$ | $2sG(P-\frac{2}{N}\lg P)+$ $2(L+3o)\lg P$ |
| **D** | $P^2(e+sE)$ | $P\log_2 P(e+sE)$ | $eP\log_2 P+$ $P^2 sE$ | $2P(e+sE)$ | $2P(e+sE\log_k P)$ | $2P(Ne+sE)$ | $2P(Ne+$ $sE\lg P)$ |
| **M** | $Pd$ | $d\log_2 P$ | $d\log_2 P$ | $2kd$ | $2kd$ | $4dN$ | $4dN$ |

**Table 2.** Overview of algorithms for unrooted collectives (minor terms are dropped, $\alpha = L + o + sG$).

the operation before all processes called it. It is similar to allreduce with a zero-sized message and is commonly implemented using the same algorithms. Both collectives can be implemented using two trees, a reduction to a root followed by a broadcast to all processes as in [21]. However, a more time-efficient implementation would be non-personalized distribution such as the Butterfly pattern [31, 34, 49].

The models in Table 2 suggest that, for non-personalized communication, Butterfly patterns are fastest for all $s$ and $P$. However, their dynamic energy consumption is asymptotically higher than the combination of two trees. Combining two pipelined trees can improve tree performance for large messages. Butterfly consumes logarithmically growing memory at each node, two $k$-ary trees could reduce this memory consumption to a constant.

**Allgather/Alltoall** Allgather is similar to a gather but the result is distributed to all processes. A simple but slow implementation would be a gather followed by a broadcast. In alltoall, each process has $P$ messages of size $s$. Each of these messages is sent to another target process, so that each process sends and receives $P-1$ messages (and an implicit message to itself). Direct send or Bruck's algorithm (using a personalized Butterfly communication) can be used to implement such collective operations. In addition, these operations can be implemented using personalized trees that gather the result to a single node and broadcast it to all nodes.

The models in Table 2 suggest that, for personalized communication, Butterfly patterns are fastest for all small $s$ and large $P$ but quickly become inefficient with growing $s$. Direct sends are most efficient for large $s$ and small $P$. Tree patterns are always more expensive in terms of runtime and energy consumption than distribution patterns. However, tree patterns can provide a constant memory consumption while other patterns have linear or logarithmic memory requirements in $P$.

### *3.2.3. Other Collectives*

**Scans/Reduce Scatter** In prefix scan operations, each process specifies a message of size $s$ and received the partial sum of all messages specified by processes with a lower id than itself. I.e., the process with id $k$ receives $r_k = m_1 \oplus m_2 \oplus m_3 \oplus \cdots \oplus m_k$ (assuming $k > 3$). A reduce scatter performs a reduction of a message of size $Ps$ specified at each process. Then, messages of size $s$ are scattered to each $P$ process. Both steps are performed together so that algorithms can optimize them as a single step. Reduce scatter can be implemented by a simple reduce followed by a scatter and scans can be implemented by rooting a different reduction tree at each process. However, merging the trees can lead to substantial performance improvements for reduce scatter [22] as well as scans.

**Neighborhood Collectives** MPI-3 introduces neighborhood collective operations [16] where the programmer can specify any communication pattern and in this way build his own collective communication operation. For example, one can express all non-reduction collective operations as neighborhood collectives. However, the expressiveness of this operation comes at the cost of optimizability. Thus, there are no generic optimization algorithms for these operations yet.
For the purpose of the analyses in this paper, we ignore irregular/vector collective operations.

## 4. Discussion and Open Problems

We now conclude our theoretical analyses with a brief summary of the lessons learned followed by an outlook to important open problems and future research directions in the area of optimizing collective communications.

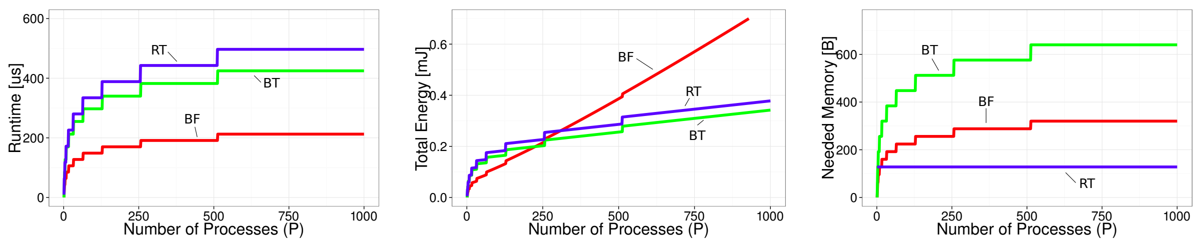
### 4.1. Approaching the Optimal

Some systems combine existing algorithms using an auto-tuning approach for algorithm selection [46]. Pjesivac-Grbovic et al. [33] for example utilize decision trees to select the best algorithm at runtime while Faraj and Yuan [13] use collective building blocks to tune them to a particular network topology. Yet, all these approaches are not strictly optimal. Selecting different algorithms and parameters for them automatically may yield significant speedups over any single algorithm. However, the problem of attaining the best bounds in terms of latency and bandwidth in the full spectrum of possible datasizes $s$ and process numbers $P$ remains open for many personalized communication algorithms.

**Problem 1: Runtime-optimal collective algorithms** We identified four essential classes of algorithms that need to be developed to attack this problem: trees with personalized and non-personalized data and dissemination mechanisms with personalized and non-personalized data. While several discrete algorithms exist for both, we expect that a general latency- and bandwidth-optimal solution will significantly improve upon the state-of-the-art.

### 4.2. Energy, Memory, and Runtime Tradeoffs

In our analysis, we identified several problems where algorithms with a smaller runtime consume more energy than algorithms with a larger runtime and vice-versa. In addition, we found that the best algorithms are generally not space optimal. This means that offloading devices with strictly limited resources may not be able to use the best known algorithms. To illustrate the tradeoff, we plot our models for a set of parameters chosen to represent an InfiniBand network architecture. These parameters are approximate and vary across installations, however, they provide insight into the tradeoffs between energy consumption and runtime.
As LogGP parameters, we use previously reported values measured for InfiniBand using MPI: $L = 6$ $\mu$s, $o = 4.7$ $\mu$s, $G = 0.73$ ns/B [17]. Kim et al. [26] model the memory read and write power consumption per MTU packet (2048 B) per switch as 8.1 pJ. We use this data to approximate the NIC power consumption assuming that each Byte in a packet is read and written once and a single packet is needed to send a 0-Byte messages. Thus, we assume $e = 16.5$ pJ, $E = 8.1$



a) Runtime. b) Energy. c) Memory.

**Figure 5.** Example for the tradeoff between runtime, energy, and memory for non-personalized distribution (e.g., allreduce) of 8 Bytes for 2-ary regular trees (RT), binomial trees (BT), and Butterfly (BF).

nJ/B, and a static NIC chip power of $P = 0.5$ W for our model. For the memory overhead, we assume that each descriptor stores a pointer, an offset, a trigger counter, and a target address. We assume that each of these fields is represented by a 64-Bit number, thus $d = 32$ B.
Figure 5 shows one particular example for a non-personal distribution communication that could be used to implement allreduce. We compare only three different options: two 2-ary trees, two binary trees, and Butterfly to instantiate the intuition from Table 2 with real-world parameters. The runtime model shows that the Butterfly algorithm is by far the best option followed by the binomial tree and the binary tree. However, in the energy model, Butterfly is far worse than both, binomial and binary trees for large numbers of processes. In fact, its dynamic energy consumption is always higher than the trees but for small process counts, the performance advantage reduces the static energy consumption in comparison to the trees. The memory model shows that the regular binary tree has the lowest, even constant memory consumption per process followed by Butterfly and binary tree. We observe that depending on the target metric, each of the three algorithms can perform best: Butterfly has the best performance, binomial trees use the least energy, and binary trees require the least memory in the network interface.

**Problem 2: Energy-optimal collective algorithms** Finding the energy-optimal algorithm for a given set of parameters (the dynamic energy consumption with $e$ and $E$ and the static power consumption $P$) for each collective operation remains an open and challenging topic as it requires to optimize time to minimize static energy in combination with the dynamic energy consumption. The optimal algorithm in terms of dynamic energy is often the simple linear algorithm that would result in excessive static energy consumption. The exact tradeoff between these algorithms is determined by the energy and runtime models as well as the energy and runtime parameters.

**Problem 3: Pareto-optimum for energy and runtime** If both previous problems are attained, one could phrase the Pareto-optimal region for the energy consumption versus the runtime. This allows to optimize the runtime in energy-constrained systems as well as the energy consumption in real-time systems. In power-constrained settings, one could also limit the dynamic energy consumption to stay within certain limits.

**Problem 4: Optimal neighborhood collective operations** The problem of optimizing neighborhood collectives is not well understood. Since they can represent any arbitrary collective operation, an optimal solution (in terms of energy consumption or runtime) would also yield optimal solutions for all MPI collectives.

### 4.3. Tradeoffs for Offload Architectures

Collective offload architectures often offer limited space on the device. The optimization problem (in terms of power and energy) can now be formulated under the restriction of limited space on the device. Our models show that each algorithm can be implemented with constant space per device. However, we also show that the necessary algorithms are slower than the best known algorithms. Interestingly, the slowdown of the constant-space algorithms seems to be limited to a factor of two compared to the best known practical algorithm. The difference may be higher when compared to close-to-optimal solutions such as Fibonacci trees and optimal personalized schedules.

We also found that many best known algorithms utilize pipelining, a technique where the memory consumption grows with the size of the sent data. Designers of offload architectures may consider to support pipelining of $N$ messages with a constant-size operation. In addition, one could allow to offload simple programs to the network card that generate sends on the fly without pre-programming everything at initialization time.

**Problem 5: Optimal memory-constrained collectives** The problem to determine the runtime- or energy-optimal schedule under the constraint of space on the offloading device may be important to support future collective offload architectures.

## 5. Conclusions

This study provides an overview of existing collective algorithms and implementations. We describe the most common algorithms for implementing collective operations in practice. However, our list is not meant to be exhaustive. We classify these algorithms into three groups: tree-shaped algorithms, distribution algorithms, and optimized schedules. The first two groups base on virtual topologies which can be used in a personalized and non-personalized setting. The last group includes optimized and specialized messaging schedules for particular cases.

We derive runtime, energy, and memory consumption models for each algorithm and compare the algorithms within each group. Our models and comparisons provide fundamental insights into the nature of these algorithms and various tradeoffs involved. For example, we show that runtime-optimal algorithms always exhibit non-optimal dynamic energy consumption. In the case of non-personalized distribution, the energy consumption of the fastest algorithm is asymptotically higher than the consumption of an algorithm that is only a slower by a constant. We also show that optimal algorithms always require more memory in offload devices than other algorithms that are only slower by a constant. This provides interesting optimization problems to find the best tradeoffs between runtime, energy, and memory consumption in offload devices.

In our theoretical study, we identified several research problems and open questions. We believe that it is most important to understand the tradeoff between energy and runtime and possibly memory consumption in offload devices. It is also interesting to design offloading protocols and devices that require minimal storage in the network architecture. In addition, a generic framework to design close-to-optimal schedules for predefined as well as neighborhood collective operations would be a valuable contribution to the state of the art.

## References

1. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model—One Step Closer Towards a Realistic Model for Parallel Computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 95–105, New York, NY, USA, 1995. ACM. ISBN 0-89791-717-0. DOI: `10.1145/215399.215427`.
2. Q. Ali, S. P. Midkiff, and V. S. Pai. Efficient High Performance Collective Communication for the Cell Blade. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 193–203, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-498-0. DOI: `10.1145/1542275.1542306`.
3. G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of MPI Collective Communication on BlueGene/L Systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, pages 253–262, New York, NY, USA, 2005. ACM. ISBN 1-59593-167-8. DOI: `10.1145/1088149.1088183`.
4. M. Banikazemi and D. K. Panda. Efficient scatter communication in wormhole k-ary n-cubes with multidestination message passing, 1996.
5. M. Barnett, D. G. Payne, and R. V. D. Geijn. Optimal broadcasting in mesh-connected architectures. Technical report, 1991.
6. M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. Geijn, and J. Watts. Interprocessor Collective Communication Library (InterCom). In *Proceedings of the Scalable High Performance Computing Conference*, pages 357–364. IEEE Computer Society Press, 1994.
7. Barrett, B.W. and Brightwell, R. and Hemmert, S. and Pedretti, K. and Wheeler K. and Underwood, K.D. and Reisen, R. and Maccabe, A.B., and Hudson, T. *The Portals 4.0 network programming interface.* Sandia National Laboratories, November 2012. Technical Report SAND2012-10087.
8. E. D. Brooks, III. The butterfly barrier. *Int. J. Parallel Program.*, 15(4):295–307, Oct. 1986. ISSN 0885-7458. DOI: `10.1007/BF01407877`.
9. J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8:1143 – 1156, 1997.
10. E. Chan, R. van de Geijn, W. Gropp, and R. Thakur. Collective communication on architectures that support simultaneous communication over multiple links. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '06, pages 2–11, New York, NY, USA, 2006. ACM. ISBN 1-59593-189-9. DOI: `10.1145/1122971.1122975`.
11. H. A. Chen, Y. O. Carrasco, and A. W. Apon. MPI Collective Operations over IP Multicast. In *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, IPDPS '00, pages 51–60, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67442-X. URL `http://dl.acm.org/citation.cfm?id=645612.662816`.
12. G. Fagg, G. Bosilca, J. Pjeivac-Grbovic, T. Angskun, and J. Dongarra. Tuned: An Open MPI Collective Communications Component. In *Distributed and Parallel Systems*, pages 65–72. Springer US, 2007. ISBN 978-0-387-69857-1. DOI: `10.1007/978-0-387-69858-8_7`.
13. A. Faraj and X. Yuan. Automatic generation and tuning of mpi collective communication routines. In *Proceedings of the 19th Annual International Conference on Supercomputing*,

ICS '05, pages 393–402, New York, NY, USA, 2005. ACM. ISBN 1-59593-167-8. DOI: 10.1145/1088149.1088202.

14. R. Graham, S. Poole, P. Shamis, G. Bloch, N. Bloch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer. ConnectX-2 InfiniBand management queues: First investigation of the new support for network offloaded collective operations. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 53–62, 2010.
15. D. Hensgen, R. Finkel, and U. Manber. Two algorithms for barrier synchronization. *Int. J. Parallel Program.*, 17(1):1–17, Feb. 1988. ISSN 0885-7458. DOI: 10.1007/BF01379320.
16. T. Hoefler and T. Schneider. Optimization Principles for Collective Neighborhood Communications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 98:1–98:10. IEEE Computer Society Press, Nov. 2012. ISBN 978-1-4673-0804-5.
17. T. Hoefler, A. Lichei, and W. Rehm. Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In *Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium, PMEO'07 Workshop*. IEEE Computer Society, Mar. 2007. ISBN 1-4244-0909-8.
18. T. Hoefler, A. Lumsdaine, and W. Rehm. Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. In *Proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.
19. T. Hoefler, C. Siebert, and W. Rehm. A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast. In *Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium (CAC'07 Workshop)*, page 232, Mar. 2007. ISBN 1-4244-0909-8.
20. T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.
21. L. P. Huse. Collective communication on dedicated clusters of workstations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 469–476. Springer-Verlag, 1999.
22. G. Iannello. Efficient Algorithms for the Reduce-Scatter Operation in LogGP. *IEEE Trans. Parallel Distrib. Syst.*, 8(9):970–982, Sept. 1997. ISSN 1045-9219. DOI: 10.1109/71.615442.
23. K. Kandalla, H. Subramoni, J. Vienne, S. Raikar, K. Tomko, S. Sur, and D. Panda. Designing Non-blocking Broadcast with Collective Offload on InfiniBand Clusters: A Case Study with HPL. In *High Performance Interconnects, 2011 IEEE 19th Annual Symposium on*, pages 27–34, Aug 2011. DOI: 10.1109/HOTI.2011.14.
24. R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser. Optimal Broadcast and Summation in the LogP Model. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '93, pages 142–153, New York, NY, USA, 1993. ACM. ISBN 0-89791-599-2. DOI: 10.1145/165231.165250.
25. T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Program-*

*ming*, PPoPP '99, pages 131–140, New York, NY, USA, 1999. ACM. ISBN 1-58113-100-3. DOI: 10.1145/301104.301116.

26. E. J. Kim, G. Link, K. H. Yum, N. Vijaykrishnan, M. Kandemir, M. Irwin, and C. Das. A holistic approach to designing energy-efficient cluster interconnects. *Computers, IEEE Transactions on*, 54(6):660–671, Jun 2005. ISSN 0018-9340. DOI: 10.1109/TC.2005.86.
27. S. Kini, J. Liu, J. Wu, P. Wyckoff, and D. Panda. Fast and scalable barrier using rdma and multicast mechanisms for infiniband-based clusters. In J. Dongarra, D. Laforenza, and S. Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2840 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20149-6. DOI: 10.1007/978-3-540-39924-7_51.
28. V. A. Korthikanti and G. Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In F. M. auf der Heide and C. A. Phillips, editors, *SPAA*, pages 157–165. ACM, 2010. ISBN 978-1-4503-0079-7.
29. V. A. Korthikanti and G. Agha. Energy-performance trade-off analysis of parallel algorithms for shared memory architectures. In *Sustainable Computing: Informatics and Systems, In Press*, 2011.
30. MPI Forum. *MPI: A Message-Passing Interface standard. Version 3.0*, 2012.
31. P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, Feb. 2009. ISSN 0743-7315. DOI: 10.1016/j.jpdc.2008.09.002.
32. B. L. Payne, M. Barnett, R. Littlefield, D. G. Payne, and R. V. D. Geijn. Global combine on mesh architectures with wormhole routing. In *Proc. of 7 th Int. Parallel Proc. Symp*, 1993.
33. J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of mpi collective operations. In *19th IEEE International Parallel and Distributed Processing Symposium, 2005. Proceedings.*, 2005.
34. R. Rabenseifner. Optimization of collective reduction operations. In *Proceedings of the International Conference on Computational Science*, ICCS 2004, pages 1–9, 2004.
35. S. Ramos and T. Hoefler. Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 97–108. ACM, Jun. 2013. ISBN 978-1-4503-1910-2.
36. P. Sack. Scalable collective message-passing algorithms. In *PhD Thesis.* University of Illinois at Urbana-Champain, 2011.
37. P. Sack and W. Gropp. Faster topology-aware collective algorithms through non-minimal communication. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 45–54, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1160-1. DOI: 10.1145/2145816.2145823.
38. P. Sanders and J. F. Sibeyn. A bandwidth latency tradeoff for broadcast and reduction. *Inf. Process. Lett.*, 86(1):33–38, Apr. 2003. ISSN 0020-0190. DOI: 10.1016/S0020-0190(02)00473-8.
39. P. Sanders, J. Speck, and J. Traff. Full bandwidth broadcast, reduction and scan with only two trees. In *Proceedings of the 14th European conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2007.

40. P. Sanders, J. Speck, and J. L. Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.*, 35(12):581–594, Dec. 2009. ISSN 0167-8191. DOI: `10.1016/j.parco.2009.09.001`.
41. M. Shroff and R. A. V. D. Geijn. CollMark: MPI Collective Communication Benchmark. Technical report, 2000.
42. S. Sur, U. K. R. Bondhugula, A. Mamidala, H. W. Jin, and D. K. Panda. High performance rdma based all-to-all broadcast for infiniband clusters. In *Proceedings of the 12th International Conference on High Performance Computing*, HiPC'05, pages 148–157, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-30936-5, 978-3-540-30936-9. DOI: `10.1007/11602569_19`.
43. R. Thakur and W. Gropp. Improving the performance of collective operations in MPICH. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2840 in LNCS, Springer Verlag (2003) 257267 10th European PVM/MPI Users Group Meeting*, pages 257–267. Springer Verlag, 2003.
44. R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of Collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.
45. J. L. Träff and A. Ripke. Optimal broadcast for fully connected networks. In *Proceedings of the First International Conference on High Performance Computing and Communications*, HPCC'05, pages 45–56, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29031-1, 978-3-540-29031-5. DOI: `10.1007/11557654_8`.
46. S. S. Vadhiyar, G. E. Fagg, and J. Dongarra. Automatically tuned collective communications. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7803-9802-5. URL `http://dl.acm.org/citation.cfm?id=370049.370055`.
47. A. Wagner, D. Buntinas, D. Panda, and R. Brightwell. Application-bypass reduction for large-scale clusters. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 404–411, Dec 2003. DOI: `10.1109/CLUSTR.2003.1253340`.
48. J. Watts and R. Van De Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(02):281–292, 1995.
49. W. Yu, D. K. Panda, and D. Buntinas. Scalable, High-performance NIC-based All-to-all Broadcast over Myrinet/GM. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 125–134, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8694-9. URL `http://dl.acm.org/citation.cfm?id=1111682.1111714`.

DOI: 10.14529/jsfi140205

# Data Compression for the Exascale Computing Era – Survey

*Seung Woo Son*[1], *Zhengzhang Chen*[1], *William Hendrix*[1], *Ankit Agrawal*[1], *Wei-keng Liao*[1], *and Alok Choudhary*[1]

While periodic checkpointing has been an important mechanism for tolerating faults in high-performance computing (HPC) systems, it is cost-prohibitive as the HPC system approaches exascale. Applying compression techniques is one common way to mitigate such burdens by reducing the data size, but they are often found to be less effective for scientific datasets. Traditional lossless compression techniques that look for repeated patterns are ineffective for scientific data in which high-precision data is used and hence common patterns are rare to find. In this paper, we present a comparison of several lossless and lossy data compression algorithms and discuss their methodology under the exascale environment. As data volume increases, we discover an increasing trend of new domain-driven algorithms that exploit the inherent characteristics exhibited in many scientific dataset, such as relatively small changes in data values from one simulation iteration to the next or among neighboring data. In particular, significant data reduction has been observed in lossy compression. This paper also discusses how the errors introduced by lossy compressions are controlled and the tradeoffs with the compression ratio.

*Keywords: Fault tolerance, checkpoint/restart, lossless/lossy compression, error bound, data clustering.*

## 1. Introduction

The future extreme scale computing systems [13, 35] are facing several challenges in architecture, energy constraints, memory scaling, limited I/O, and scalability of software stacks. As the scientific simulations running on such systems continue to scale, the possibility of hardware/software component failures will become too significant to ignore. Therefore, a greater need is emerging for effective resiliency mechanisms that consider the constrains of (relatively) limited storage capability and energy cost of data movement (within deep memory hierarchy and off-node data transfer). The traditional approach to storing checkpoint data in raw formats will soon be cost-prohibitive. On the other hand, storing the states of simulations for restart purposes will remain necessary. Thus, in order to scale resiliency via the checkpoint/restart mechanism, multiple dimensions of the problem need to be considered to satisfy the constraints posed by such an extreme-scale system.

A popular solution to reduce checkpoint/restart costs is to apply data compression. However, because scientific datasets are mostly floating-point numbers (in single or double precision), a naive use of compression algorithms can merely bring a limited improvement in terms of the amount of data reduced while bearing a high compression overhead to perform compression. To tackle such problem of hard-to-compress scientific data, recently-proposed compression algorithms start to explore data characteristics exhibited in specific domains and develop ad-hoc strategies that perform well on those data. The motivation comes from the fact that most of the simulations calculates values on points (nodes, particles, etc.) in a discretized space to solve its own mathematical models and continues along the temporal dimension until a certain stop condition is met. One can potentially exploit the patterns exhibited along both spatial and temporal dimensions in improving effectiveness of existing compression algorithms because the variance of data in the neighborhood tends to be small in many cases. Existing compression algorithms seldom consider such data patterns.

[1]Department of EECS, Northwestern University, Evanston, USA

Lossy compressions [11, 15] often produce better compression ratios than the lossless counterpart, but the error rates are not easy to bound, and, in large scale simulations, unbounded error could introduce significant deviation from the actual values, leading to impact the outcome of the simulation. In other words, lossy compressions could make compressed data useless. Lossy compression on checkpointing implies several challenges for large-scale simulations, e.g., guaranteeing point-wise error bounds defined by the user, reducing a sufficiently large amount of storage space, performing compression in-situ (to reduce data movement), and taking advantages of data reduction by potentially being able to use locally-available non-volatile storage devices. In this paper, we survey a set of compression techniques and discuss whether they can achieve the above goals. We also describe ideas of making use of machine learning techniques to discover temporal change patterns, designing a data representation mechanism to capture the relative changes, and solutions to meet the user request on tolerable error bound.

We anticipate that checkpointing will continue to be a crucial component of resiliency and lossy compressions will become an acceptable method to reduce the data size at the runtime. The critical point is whether the data can be significantly reduced given a controlled error bound. Such an error tolerance will be specified by the user as an input parameter. For example, a user can indicate that maximum tolerable error per point of 0.1%, and lossy compression algorithms must guarantee that bound. Overall, applying data compression at runtime for checkpointing will become appealing to large-scale scientific simulations on future high-performance computing systems, as it can reduce storage space as well as save the energy resulted from data movement.

The remainder of this paper is organized as follows. The discussion of lossless compression algorithms for traditional checkpointing mechanisms is described in Section 2. Section 3 presents several studies to apply lossy compression algorithms on scientific applications. Finally, we conclude the paper in Section 4.

## 2. Lossless Compression

Compression has two advantages. At first, it can reduce the space required to store the data. Secondly, it can improve I/O bandwidth (disk or network communication), as the time spent on data checkpointing is reduced. However, compression comes with the CPU and memory overhead to compress and decompress the data. Thus, the effectiveness of applying data compression is determined by the achievable compression ratio of the selected compression algorithm and the time to compress the data. The future exascale systems are expected to exhibit a trend that the data movement among memory hierarchies and off-node data transfer will become relatively expensive in terms of time and energy, compared to the ever-increasing compute power. If the selected compression algorithm can produce a reasonable compression ratio, the benefit of applying data compression shall become more significant for the scientific applications running on those systems.

When it comes to compression, scientists often face the dilemma of choosing lossless compression and lossy compression. The former preserves data fidelity but can be slow and produce a poor compression ratio, whereas the latter introduces a cumbersome validation process on the approximated data. In this section, we first compare several existing lossless compression methods and their effectiveness when applied on the scientific datasets.

## 2.1. Integration within Checkpointing Framework

Welton *et al.* [36] have integrated and evaluated several lossless compression algorithms within the context of IOFSL, an I/O middleware. They evaluate their framework on using commonly-used compression algorithms on synthetic and real datasets. The experimental results show about 1.6x of compression ratio for the air/sea flex data from NCAR data archive. The framework could potentially serve as a baseline I/O utility for HPC systems; however, the algorithms they selected performed rather poorly on scientific data. Ibtesham *et al.* [17] also did a viability study that compares several lossless compression algorithms with the Berkeley Lab Checkpoint/Restart (BLCR) [14] framework. They essentially observed up to 2x compression ratio for several scientific datasets.

Islam *et al.* [18] present mcrEngine, a software framework that first merges compatible data in terms of data type (double, float, other) between checkpoint files and compresses the output of the merged data with an existing compression algorithm. The authors tested mcrEngine on five different codes that exhibited this type of compatibility between checkpoint files and compressed the checkpoint data by a factor of about 1.2. While this checkpoint-level similarity may not be applicable to every simulation code, the technique of combining this data before compression clearly improves compressibility.

Several high-level I/O libraries provide a compression capability in a transparent manner. HDF5 requires users to use chunking to create a compressed dataset. Currently HDF5 provides two predefined compression filters (zlib and szip). All other compression filters must be registered with HDF5 before they can be used [34]. ADIOS [27] also provides a similar compression capability as a data transformation mechanism in file-based I/O. ADIOS provides three common compression algorithms (zlib, gzip, and szip) as well as ISOBAR [30], which could potentially yield higher compression ratio on scientific datasets.

There are recent studies about exploiting compute power in SSDs to perform in-situ operations including compression. Active Flash by Boboila *et al.* [7] is such an example where the compression is performed within SSDs in the context of data analytic pipelines. Since their main objective is to determine feasibility of performing in-situ on SSDs, they also simply use existing lossless compression algorithm, LZO, on common HPC datasets: atmospheric and geosciences (NetCDF format) and bioinformatics (text format). Active Flash achieved about 1.96x of compression ratio for both datasets.

## 2.2. Increasing Compressibility through Transformations

Bicer *et al.* [6] propose a novel compression algorithm for climate data, CC (climate compression), that takes advantage of the spatial and temporal locality inherent in climate data to accelerate the storage and retrieval of data. Their methodology uses an exclusive or (`xor`) of adjacent or consecutive data to reduce the entropy in the data, which is analogous to taking a difference or ratio in the sense that similar data values will neutralize to form easily compressed datasets.

Schendel *et al.* propose the ISOBAR [29, 30] framework where it first divides the data into compressible and incompressible segments before applying lossless compression to reduce data size. PRIMACY [31] and ALACRITY [19] applied similar data transformation methods on byte columns in order to identify compressible byte segments. We note that ISOBAR used FPC [8], a fast and effective lossless compression algorithm designed for double-precision floating point

**Table 1.** Comparison of lossless compression schemes.

| Scheme | Transformation Applied | Algorithm | Compression Ratio |
|---|---|---|---|
| FPC [8] | not used | it first predicts values sequentially using two predictors (FCM and DFCM), and subsequently selects the closer predicted value to the actual. Lastly, it XORs the selected predicted value with the actual value, and leading-zero compresses the result. | 1.02x∼1.96x |
| ISOBAR [30] | divide byte-columns into compressible and incompressibles | apply zlib, bzlib2, (fpzip, FPC) on all compressible (after discarding noisy byte-columns). zlib is the main compression algorithm; others are for comparison purposes | 1.12x∼1.48x |
| PRIMACY [31] | frequency based permutation of ID values | apply zlib on transformed data | 1.13x∼2.16x |
| ALACRITY [19] | split floating-point values into sign, exponent, and significands | unique-value encoding of the most significant bytes (assuming high-order bytes (sign and exponents) are easy to compress); low-order bytes are compressed using ISOBAR | 1.19x∼1.58x |
| CC [6] | XOR on $\Delta$ of neighboring data point in the same iteration | apply zero-filled run length encoding | up to 2.13x |
| IOFSL [36] | not used | integration of LZO, bzip2, zlib within the I/O forwarding layer | ∼1.9x |
| Binary Masking [5] | bit masking (XOR) | apply zlib on bit masked data in order to partially decreases the entropy level | 1.11x∼1.33x |
| MCRENGINE [18] | variable merging in the same group | apply parallel gzip on the merged variables across processes | up to 1.18x |

numbers. FPC first predicts values sequentially using predictors, and subsequently selects the closer predicted value to the actual. It then XORs the selected predicted value with the actual value such that more leading zeroes in the predicted values, which helps improve compression ratios.

Bautista-Gomez and Capello [5] propose an algorithm related to ISOBAR in that both are lossless compression algorithms that seek to identify low-entropy segments of floating-point data and compress them separately. Bautista-Gomez and Capello, however, transform the data by applying a bit masking (`xor`) to the original data in order to reduce its entropy before compression. They present results for a number of scientific datasets (GTC dataset in single-precision and climate dataset in double-precision), achieving a maximum of around 40% data reduction.

## 2.3. Comparison

Table 1 summarizes the comparison of lossless compression schemes surveyed in this paper. Since lossless algorithms do not incur any loss of information in the uncompressed data, we mainly compare the compression ratio achieved by each method. The compression ratio $R$ for data D of size $|D|$ reduced to size $|D^{'}|$ is denoted as: $R = \frac{|D|}{|D'|}$. We note that scientific simulations use predominantly double-precision floating-point variables. Therefore, the compression ratio presented in Table 1 is for those only, though the algorithm can be applied to floating point numbers of different precision or other types of data. The main takeaway from Table 1 is that the data reduction by lossless compression is limited; the maximum achievable compression ratio is just above 2x, which is also possible after additional data transformation is applied. As described in the second column of Table 1, most compression algorithms apply some sort of data transformations in order to increase the effectiveness of data compression.

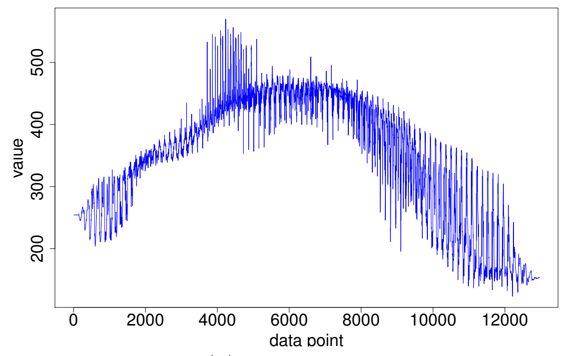
## 3. Lossy Compression

The intuition behind using lossy compression algorithms stems from the following three observations. First, scientific simulation data, like climate CMIP5 *rlus* data, are considered as one type of high entropy data [12, 28]. Such data often exhibits randomness without any distinct repetitive patterns in one single timestamp or iteration (see Figure 1 (a) or (b)). Thus, traditional *lossless* compression approaches, as described in Section 2, cannot achieve appreciable data reduction. Second, in many scientific applications, relative changes in data values are often not very significant either spatially (among neighboring data points) or temporally (from one simulation iteration to the next). As an example for this temporal similarity, more than 80% of climate rlus data remains unchanged or only change with a percentage less than 0.1% (see Figure 1 (c)). Third, unlike observational data, many scientific simulation codes can tolerate some error-bounded loss in their simulation data accuracy. Thus, *lossy* compression methods can offer some attractive features for data reduction.

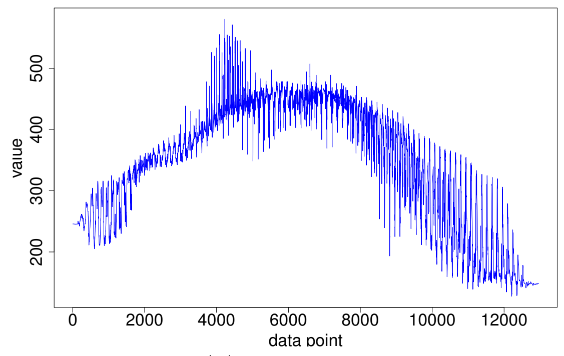
The effectiveness of lossy compression however heavily depends on the domain knowledge to select the right compression algorithms, and it is very difficult to get compression beyond a small factor with desired accuracy, not to speak of guaranteeing that the compression error will be smaller than a certain error rate, preferably specified by a user. Figure 1(c) shows the change in data values between two iterations (checkpoints) instead of individual iteration because representing data in change ratio could minimize coding space during compression. For example, a checkpoint with 100 million data points where there are potentially 100 million changes. Two data points where one changes from 10 to 11 and the other from 100 to 110 have the identical relative changes, which can be represented as the same 10 percent change. Therefore, data points with the same change percentage can be indexed by one number. The idea of considering the data changes along temporal domain transforms the data where individual checkpoint seldom exhibits repeated patterns into a space where common patterns in change percentages are easier to find. To ensure the quality of reduced data, one challenge of this approach is to select a set of change values that can represent a large number of neighbors within a small radius, a tolerable error rate specified by the user. This will potentially address the challenge of lossy compression in maintaining the quality of compressed data, which will be discussed in later sections. Furthermore, simulation parameters are often calibrated using data that are themselves subject to measurement error or other inaccuracies. Therefore, very small deviations ($< 1\%$) in restart fidelity are unlikely to hurt normal scientific simulations as long as such deviations are bounded.

### 3.1. Transformation Schemes

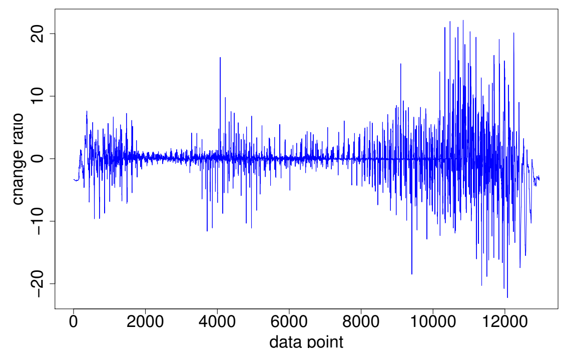
There are handful of prior works studied about applying lossy compression on scientific datasets. Lakshminarasimhan *et al.* [22] described ISABELA, a lossy compression technique based on applying B-splines to sorted data. By transforming the data, ISABELA was able to achieve high compression on data that was previously regarded as “incompressible,” while losing minimal data fidelity ($\geq$0.99 correlation with original data). Lakshminarasimhan *et al.* did not



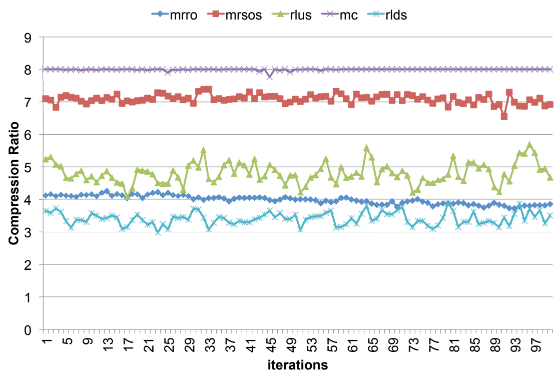
(a) iteration 1



(b) iteration 2



(c) the change between (a) and (b)



(d) compression ratio

**Figure 1.** Data representation of *rlus*, a variable from the CMIP5 climate simulation dataset: (a) original data distribution at iteration 1. (b) original data distribution at iteration 2. (c) the changing percentage of data values between two iterations, and (d) compression ratio ($\frac{\text{original size}}{\text{compressed size}}$) for the clustering-based algorithm on the CMIP5 simulation datasets [33]. Out of the dozens variables available in CMIP5 [1], we randomly chose five, namely *mrsos*, *mrro*, *mc*, *rlds*, and *rlus*. The resolution for these data is 2.5° by 2°. *mc* is a monthly simulation data, while other four are daily data. We note that all approximated data point is within the user-specified error bound, which is 0.1%. The approximation precision used is 8 bits (or 1 byte). The parallel K-means clustering algorithm [3, 20, 25] is used on the temporal change ratios $\Delta$ to get $2^B - 1$ clusters, where $B$ is the number of bits.

consider applying ISABELA to checkpoint data because they assume checkpoint data do not permit approximation.

As an another transformation approach, we have also studied a data transformation idea similar to video compression algorithms especially MPEG [24], which stores the differences between successive frames using temporal compression. Similar to MPEG's forward predictive coding where current frames are with reference to a past frame, we code the current checkpoint data based on the previous checkpoint. In order to this, the relative change (or change ratio) is calculated as $\Delta = \frac{D_c - D_p}{D_p}$, where $D_c$ and $D_p$ is the data point in the current and previous iteration, respectively. A potential problem with this approach is that $D_p$ cannot be zero. If $D_p$ is zero, $D_c$ cannot be compressible. This transformation technique is applied to our clustering-based algorithm.

## 3.2. Approximation Algorithms

Several recent studies [4, 16, 23] have evaluated Samplify's APplication AXceleration (APAX) Encoder along with other lossy compression algorithms on scientific datasets, mostly climate dataset. The APAX algorithm encodes sequential blocks of input data elements with user-selected block size between 64 and 16,384. The signal monitor tracks the input dataset's center frequency. The attenuator multiplies each input sample by a floating-point value that, in fixed-rate mode, varies from block to block under the control of an adaptive control loop that converges to the user's target compression ratio. The redundancy remover generates derivatives of the attenuated data series and determines which of the derivatives encodes using the fewest bits. The bit packer encodes groups of 4 successive samples using a joint exponent encoder (JEE). JEE exploits the fact that block floating-point exponents are highly correlated, which is commonly observed in many lossless compress techniques [8, 19, 29, 30, 31].

*fpzip* quantizes the significant bits while leaving the values in their floating-point format. *fpzip* fixes the quantization level to be a power of two, thereby effectively truncating (zeroing) a certain number of significant bits. Designed such, *fpzip* can be lossless if the quantization level is the same as the original data representation. ISABELA, on the other hand, applies the B-splines curve fitting (interpolation) to the sorted data. A B-splines curve is a sequence of piecewise lower order parametric curves joined together via knots. Specifically, they used the cubic B-splines for faster interpolation time and producing smooth curves [22].

The clustering-based approach, on the other hand, uses machine learning techniques once the change ratios for all data points are calculated. Once the change ratios of all data points have been calculated, using machine learning techniques, we first calculate the distributions of changes and then approximate them into an indexed space to achieve the goals of maximal data reduction. In our preliminary implementation, we use histogram for learning distribution of the change ratios obtained. Histogram is a popular method to learn the data distribution. For example, data shown in Figure 1(c) can be easily converted into a histogram. Histogram estimates the probability distribution by partitioning the data into discrete intervals or bins. The number of data points falling in a bin indicates the the frequency of the observations in that interval. Like other lossy compression algorithms, the clustering-based algorithms controls the precision of the approximation using $B$ bits. $B$ bits are used to store the index of a transformed data point. Since $B$ bits can represent $2^B$ different values and if the number of different change ratios in $\Delta$, $|\Delta D_i|$, is larger than $2^B$, then some of $\Delta$ must be grouped together and approximated by a representative ratio in the same group. We compute such approximation to fit all representative change ratios into a index table of size $2^B$.

While one can use rather simpler methods like equal-width or log-scale binning than the clustering-based binning, they may not perform well for cases where the distribution is highly irregular. One such example would be when there are several densely packed bins and those dense bins are spread unevenly. Neither the equal-width nor the log scheme is known to be capturing such an irregularity well. Data clustering is a technique to partition data into groups with a similarity (in terms of distance) while maximizing the difference among groups. Several prior studies [10, 21, 37] also have been used clustering techniques in compression, mostly for multimedia data (images, sounds, and videos). The idea behind those techniques based on the following characteristics on multimedia data. First, the data objects are highly similar from one time frame to another, that is, small temporal changes. Second, a certain amount of information loss is acceptable in their applications. Lastly, as described in [32], a substantial data reduction is highly desired. In many senses, many scientific simulation runs exhibit very similar behaviors as the multimedia data. This is the reason why the cluster analysis technique achieves better binning results for irregularly distributed datasets.

## 3.3. Error Bounding Methods

The main challenge when applying lossy compression is assessing its effect on simulation accuracy. Lossy compression is commonly used for multimedia data (photographs and videos) where errors need only to meet human perceptual requirements. Key-frames in videos periodically reset the error to zero. The scientific simulation codes solve time-dependent differential equations where errors may accumulate over time. The changes in the solution due to the use of compression may grow more rapidly if compression errors at one time step are strongly correlated with errors at the next time step. The changes due to the use of compression are thus likely to be functions of both the compression ratio and the error characteristics of the compression algorithm.

However, the fact that lossy compression may change simulation results is not necessarily a show-stopper. In many cases, a slight change in one simulation parameter also can change the simulation output. For example, scientists may choose a mesh resolution based on a trade-off between more accurate answers and the computational cost of the simulation. Computational scientists usually make these choices based on the desired accuracy of various physical quantities, not by comparing differences in per grid point basis like the mean square error between two simulations. For example, Laney *et al.* [23] have proposed to use the same integral physical quantities to assess the impact of compression.

ISABELA and *fpzip* bound errors using the accuracy metric that measures factors between the original and approximated values. For example, ISABELA uses Pearson correlation of $\geq$0.99, which implies that 99% of approximated data is within the error bound. Since this metric measures the strength and direction of the linear relationship between two, the error bound is *relative*, thus no absolute error guarantee outside the defined range. *fpzip* also allows the similar relative error bound because of its use of the non-uniform quantization [23].

In contrast to traditional lossy compression algorithms, the clustering-based scheme is designed to process data under the condition that the compressed data is guaranteed with a user-specified *absolute* error bound or a user tolerance error rate $E$. The value of $E$ is usually determined based on the application domain knowledge. For each point in $\Delta$, if $abs(\Delta) < E$, the clustering-based algorithm uses 0 as its approximation value because it already meets the user tolerance error threshold. Otherwise, it uses the clustering algorithm to learn the distribution of $\Delta$ and partitions the data in $\Delta$ based on their similarity in order to meet the user tolerance error-bound $E$. The compression ratio shown in Figure 1(d) was obtained while the error rate and the approximation precision are fixed at 0.1% and 8 bits, respectively. In terms of the mean error rate, all variables show less than 0.025% of error rates, guaranteeing the user-specified error rate. Similar to the clustering-based algorithm, APAX also allows the absolute error to be bounded within each APAX block. Quantization is the only source of loss in APAX, which can be tunable by the user [23].

## 3.4. Tradeoffs Approximation Precision and Error Rate

One obvious advantage of using lossy compression techniques is that users can tradeoff between the approximation precision and compression ratio. In order to demonstrate this, we varied the number of precision bits to see how the clustering-based lossy compression algorithm performs in terms of compression ratio and error rate. In this set of experiments, we fix the user-defined tolerable error rate at 0.1%. As one can expect, increasing the approximation bits improved the compression ratio significantly. For instance, the compression ratio increases dramatically when the number of bits changed from 8 to 9 bits, 40% to 80% while the mean error rate is increased only by 0.02%. Furthermore, if the approximation precision is 10 bits, then all data points became compressible, resulting in compression ratio of 8x with the mean error rate less than 0.05%. Other approximation schemes like log-scale or equal-with binning achieved similar results on *rlds* and other variables.

Lossy compression algorithms can perform differently depending on the amount of information loss. For example, when we vary the user tolerable error rate from 0.1% (default error bound) to 0.5%, the average compression ratio by the clustering-based scheme increased from 2.1x up to 4.7x. The mean error rate is also increased from 0.02% to 0.12% as the user tolerable error rate is increased 5 times. We however note that they are still much smaller than the user tolerable error rate. For example, we could maintain the mean error rate of 0.1% even with the user tolerable error rate of 0.4%.

## 3.5. Comparison

Table 2 gives a comparison of lossy compression algorithms described so far. Each scheme uses different transformation methods in order to increase compressibility, and then apply different approximation algorithms on those transformed data. As compared with Table 1, one can clearly see that lossy compression algorithms achieve much higher compression ratios; up to 8x depending on how compressible data is. Another important observation from Table 2 is that data transformation helps increase the compression ratio significantly. *fpzip* achieved only 1.29x of compression ratio because it uses only prediction mechanism based on data traversal without applying any transformation on the actual data. ISABELA and the clustering-based scheme achieved the highest compression ratio because both schemes first transform original data into a compression-friendly format.

We note that each data point (assuming 64-bit) in lossy compression algorithm is divided into two categories: compressible and incompressible. All the compressible data is represented as an approximation bit number (e.g., 8 bits) whereas the incompressible data is stored as the original bit number (i.e., 64-bit). Since the compressible portion is essentially represented as integer streams, we can further increase the compression ratio by applying one of existing lossless compression techniques like *zlib* [2], *bzlip2*, or *LZO* to the index data. As discussed in [22], indices, which are integer values, are easy to compress with standard lossless compressions, resulting in about 75%–90% of additional compression ratio.

While not extensively discussed in this paper, there are studies where data is encoded not at system-level but at application-level to tolerate faults. A recent study by Chen [9] describes an alternative technique, called Algorithm-Based Fault Tolerance (ABFT), that eschews traditional checkpointing techniques to incorporate error recovery into algorithmic design. Chen demonstrates essentially overhead-free recovery mechanisms for the Jacobi method and conjugate gradient descent, algorithmic error recovery mechanisms are by necessity specific to the code being run. Moreover, they are potentially vulnerable to compound or cascading failures, which periodic checkpointing would help to alleviate, even in cases where such techniques are applicable.

**Table 2.** Comparison of lossy compression schemes.

| Scheme | Transformation Applied | Approximation Algorithm | Compression Ratio | Error bound |
|---|---|---|---|---|
| ISABELA [22] | sorting | apply B-spline on sorted data | up to 5x | ≥0.99 of correlation |
| fpzip [26] | not used | traverse data in a coherent order and then uses the corresponding n-dimensional (where n is the dimensionality of the data) Lorenzo predictor to predict the subsequent values. It next maps the predicted values and actual values to their integer representations, and encodes the XORd' residual between these values. | 1.29x | relative |
| APAX [30] | not used | encodes sequential blocks of input data elements with user-selected block size between 64 and 16,384 | 1.33x∼4x | absolute |
| Clustering-based | change ratios between consecutive iterations | approximate on change ratios; full checkpoint initially and when the error rate is close to user-specified bound | 2.98x∼8x | <0.1% absolute |

## 4. Conclusion

This paper argues that while the traditional checkpointing continues to be a crucial mechanism to tolerate system failures in many scientific applications, it is also becoming challenging in the exascale era mainly because of limited I/O scalability and associated energy cost. This paper describes several efforts to use lossless compression on checkpoint data to relieve such overheads and shows that they are limited because of inherent randomness in scientific datasets. This paper then describes several lossy compression algorithms that radically change how checkpoint data is stored with tunable error bounding mechanisms. Hence, we predict the lossy compression to be a promising way to reduce checkpoint overheads without compromising the quality of dataset that scientific simulation operates on.

For lossy compressions to be actually deployed in the exscale computing era, several future challenges remain. The first challenge would be reducing the memory requirements to perform in-situ compression. This is especially challenging because of two facts: (1) per-core memory is expected to continue to decrease in the exascle systems, and (2) transformation techniques to improve the effectiveness of compressions typically require extra memory, making memory a scarce resource. Second, given the amount of increasing data volumes and number of CPU cores, the compression should be performed in parallel. A couple of current parallel I/O libraries support data compression, but the compression is performed individually, in order words, compression is locally optimized without knowing what others do. Lastly, compression algorithms must take advantages of several emerging techniques to be used in future exascale systems such as SSD, PCRAM, etc., as they will significantly impact future memory hierarchy.

## Acknowledgement

This work is supported in part by the following grants: NSF awards CCF-1029166 and OCI-1144061, and IIS-1343639; DOE awards DE-SC0005309, DESC0005340, and DESC0007456; AFOSR award FA9550-12-1-0458. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## References

1. `http://cmip-pcmdi.llnl.gov/cmip5/`.
2. `http://www.zlib.net/`.
3. Ankit Agrawal, Mostofa Patwary, William Hendrix, Wei-keng Liao, and Alok Choudhary. *High Performance Big Data Clustering*, pages 192–211. IOS Press, 2013.
4. Allison H. Baker, Haiying Xu, John M. Dennis, Michael N. Levy, Doug Nychka, Sheri A. Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A Methodology for Evaluating the Impact of Data Compression on Climate Simulation Data. In *Proceedings of the ACM International Symposium on High-Performance Parallel and Distributed Computing*, 2014. DOI: `10.1145/2600212.2600217`.
5. Leonardo Arturo Bautista-Gomez and Franck Cappello. Improving Floating Point Compression through Binary Masks. In *Proceedings of the IEEE International Conference on Big Data*, pages 326–331, 2013. DOI: `10.1109/BigData.2013.6691591`.
6. Tekin Bicer, Jian Yin, David Chiu, Gagan Agrawal, and Karen Schuchardt. Integrating Online Compression to Accelerate Large-Scale Data Analytics Applications. In *Proceedings of the IEEE 27th International Symposium on Parallel Distributed Processing*, pages 1205–1216, May 2013. DOI: `10.1109/IPDPS.2013.81`.
7. Simona Boboila, Youngjae Kim, Sudharshan S. Vazhkudai, Peter Desnoyers, and Galen M. Shipman. Active Flash: Out-of-core Data Analytics on Flash Storage. In *Proceedings of the 28th Symposium on Mass Storage Systems and Technologies*, pages 1–12, 2012. DOI: `10.1109/MSST.2012.6232366`.
8. Martin Burtscher and Paruj Ratanaworabhan. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Computers*, 58(1):18–31, 2009. DOI: `10.1109/TC.2008.131`.
9. Zizhong Chen. Algorithm-based Recovery for Iterative Methods Without Checkpointing. In *Proceedings of the International Symposium on High Performance Distributed Computing*, pages 73–84, 2011. DOI: `10.1145/1996130.1996142`.
10. C.-H. Chou, M.-C. Su, and E. Lai. A New Cluster Validity Measure and Its Application to Image Compression. *Pattern Anal. Appl.*, 7(2):205–220, July 2004. DOI: `10.1007/s10044-004-0218-1`.
11. Jin J. Chou and Les A. Piegl. Data Reduction Using Cubic Rational B-splines. *IEEE Computer Graphics and Applications*, 12(3):60–68, May 1992. DOI: `10.1109/38.135914`.
12. Thomas M. Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

13. David Donofrio, Leonid Oliker, John Shalf, Michael F. Wehner, Chris Rowen, Jens Krueger, Shoaib Kamil, and Marghoob Mohiyuddin. Energy-Efficient Computing for Extreme-Scale Science. *IEEE Computer*, 42(11):62–71, 2009. DOI: `10.1109/MC.2009.353`.

14. Jason Duell. The Design and Implementation of Berkeley Labs linux Checkpoint/Restart. Technical report, Lawrence Berkeley National Laboratory, 2003.

15. Michael Frazier. *An Introduction to Wavelets through Linear Algebra*. Undergraduate texts in mathematics. Springer, 1999.

16. Nathanael Hübbe, Al Wegener, JulianMartin Kunkel, Yi Ling, and Thomas Ludwig. Evaluating Lossy Compression on Climate Data. In JulianMartin Kunkel, Thomas Ludwig, and HansWerner Meuer, editors, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, volume 7905 of *Lecture Notes in Computer Science*, pages 343–356, 2013.

17. Dewan Ibtesham, Dorian C Arnold, Patrick G Bridges, Kurt B Ferreira, and Ron Brightwell. On the Viability of Compression for Reducing the Overheads of Checkpoint/Restart-Based Fault Tolerance. In *Proceedings of the International Conference on Parallel Processing*, pages 148–157, 2012. DOI: `10.1109/ICPP.2012.45`.

18. Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. de Supinski, and Rudolf Eigenmann. McrEngine: A Scalable Checkpointing System Using Data-aware Aggregation and Compression. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 17:1–17:11, 2012.

19. John Jenkins, Isha Arkatkar, Sriram Lakshminarasimhan, David A. Boyuka II, Eric R. Schendel, Neil Shah, Stéphane Ethier, Choong-Seock Chang, Jackie Chen, Hemanth Kolla, Scott Klasky, Robert B. Ross, and Nagiza F. Samatova. ALACRITY: Analytics-Driven Lossless Data Compression for Rapid In-Situ Indexing, Storing, and Querying. *Transactions on Large-Scale Data- and Knowledge-Centered Systems X - Special Issue on Database- and Expert-Systems Applications*, 10:95–114, 2013. DOI: `10.1007/978-3-642-41221-9_4`.

20. Ruoming Jin, Anjan Goswami, and Gagan Agrawal. Fast and Exact Out-of-core and Distributed K-means Clustering. *Knowl. Inf. Syst.*, 10(1):17–40, 2006. DOI: `10.1007/s10115-005-0210-0`.

21. Nicolaos B. Karayiannis and Pin-I Pai. Fuzzy Vector Quantization Algorithms and Their Application In image Compression. *IEEE Transactions on Image Processing*, 4:1193–1201, 1995. DOI: `10.1109/83.413164`.

22. Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data. In *Proceedings of the 17th International Conference on Parallel Processing*, pages 366–379, 2011. DOI: `10.1007/978-3-642-23400-2_34`.

23. Daniel Laney, Steven Langer, Christopher Weber, Peter Lindstrom, and Al Wegener. Assessing the Effects of Data Compression in Simulations Using Physically Motivated Metrics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 76:1–76:12, 2013. DOI: `10.1145/2503210.2503283`.

24. Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Commun. ACM*, 34(4):46–58, April 1991. DOI: `10.1145/103085.103090`.

25. Wei-keng Liao. Parallel K-Means Data Clustering. `http://www.eecs.northwestern.edu/~wkliao/Kmeans/`, 2005.

26. Peter Lindstrom and Martin Isenburg. Fast and Efficient Compression of Floating-Point Data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1245–1250, Sept 2006. DOI: `10.1109/TVCG.2006.143`.

27. Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible IO and Integration for Scientific Codes Through the Adaptable IO System (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, pages 15–24, 2008. DOI: `10.1145/1383529.1383533`.

28. Khalid Sayood. *Introduction to Data Compression (2nd Ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

29. Eric R. Schendel, Ye Jin, Neil Shah, Jackie Chen, Choong-Seock Chang, Seung-Hoe Ku, Stphane Ethier, Scott Klasky, Robert Latham, Robert B. Ross, and Nagiza F. Samatova. ISOBAR Preconditioner for Effective and High-throughput Lossless Data Compression. In *Proceedings of the 28th IEEE International Conference on Data Engineering*, 2012. DOI: `10.1109/ICDE.2012.114`.

30. Eric R. Schendel, Saurabh V. Pendse, John Jenkins, David A. Boyuka, II, Zhenhuan Gong, Sriram Lakshminarasimhan, Qing Liu, Hemanth Kolla, Jackie Chen, Scott Klasky, Robert Ross, and Nagiza F. Samatova. ISOBAR Hybrid Compression-I/O Interleaving for Large-scale Parallel I/O Optimization. In *Proceedings of the International ACM Symposium on High Performance Parallel and Distributed Computing*, June 2012. DOI: `10.1145/2287076.2287086`.

31. Neil Shah, Eric R. Schendel, Sriram Lakshminarasimhan, Saurabh V. Pendse, Terry Rogers, and Nagiza F. Samatova. Improving I/O Throughput with PRIMACY: Preconditioning ID-Mapper for Compressing Incompressibility. In *Proceedings of the IEEE International Conference on Cluster Computing*, September 2012. DOI: `10.1109/CLUSTER.2012.16`.

32. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, May 2005.

33. Karl E. Taylor, Ronald J. Stouffer, and Gerald A. Meehl. An Overview of CMIP5 and the Experiment Design. *Bull. Amer. Meteor. Soc.*, 93(4):485–498, October 2012.

34. The HDF Group. `http://www.hdfgroup.org/HDF5/`.

35. Josep Torrellas. Architectures for Extreme-Scale Computing. *Computer*, 42(11):28–35, November 2009. DOI: `10.1109/MC.2009.341`.

36. Benjamin Welton, Dries Kimpe, Jason Cope, Christina M. Patrick, Kamil Iskra, and Robert Ross. Improving I/O Forwarding Throughput with Data Compression. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 438–445, 2013. DOI: `10.1109/CLUSTER.2011.80`.

37. Yueting Zhuang, Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Adaptive Key Frame Extraction using Unsupervised Clustering. In *Proceedings of the International Conference on Image Processing*, pages 866–870, 1998. DOI: `10.1109/ICIP.1998.723655`.

DOI: 10.14529/jsfi140206

# Extreme Big Data (EBD): Next Generation Big Data Infrastructure Technologies Towards Yottabyte/Year

***Satoshi Matsuoka*** [1]***, Hitoshi Sato*** [1]***, Osamu Tatebe*** [2]***, Fuyumasa Takatsu*** [2]***, Mohamed Amin Jabri*** [2]***, Michihiro Koibuchi*** [3]***, Ikki Fujiwara*** [3]***, Shuji Suzuki*** [1]***, Masanori Kakuta*** [1]***, Takashi Ishida*** [1]***, Yutaka Akiyama*** [1]***, Toyotaro Suzumura*** [4]***, Koji Ueno*** [1]***, Hiroki Kanezashi*** [1]***, and Takemasa Miyoshi*** [5]

Our claim is that so-called "Big Data" will evolve into a new era with proliferation of data from multiple sources such as massive numbers of sensors whose resolution is increasing exponentially, high-resolution simulations generating huge data results, as well as evolution of social infrastructures that allow for "opening up of data silos", i.e., data sources being abundant across the world instead of being confined within an institution, much as how scientific data are being handled in the modern era as a common asset openly accessible within and across disciplines. Such a situation would create the need for not only petabytes to zetabytes of capacity and beyond, but also for extreme scale computing power. Our new project, sponsored under the Japanese JST-CREST program is called "Extreme Big Data", and aims to achieve the *convergence of extreme supercomputing and big data* in order to cope with such explosion of data. The project consists of six teams, three of which deals with defining future EBD convergent SW/HW architecture and system, and the other three the EBD co-design applications that represent different facets of big data, in metagenomics, social simulation, and climate simulation with real-time data assimilation. Although the project is still early in its lifetime, started in Oct. 2013, we have already achieved several notable results, including becoming world #1 on the Green Graph 500, a benchmark to measure the power efficiency of graph processing that appear in typical big data scenarios.

*Keywords: Big Data, Supercomputing, Extreme Computing and Big Data Convergence, Data Intensive Computing, Non-Volatile Memory.*

## 1. Introduction

"Big Data" has become a hot topic in mainstream IT; although large scale databases as well as data intensive computing has existed from the past, amassing petabytes to even zetabytes of data is becoming a reality, as overall data generation rate and its traffic, both on the Internet as well as dedicated instrumentation networks, are exploding at exponential ratio as Moore's law continues to speed up processing, enlarge memory, and increase sensor resolution, etc.

However, to most common people "Big Data" is almost synonymous to "mining people's private data (such as purchase history, credit card spending, location tracking, etc.) for marketing purposes". Such a definition is not only extremely narrow, limiting the potential application impact to the global society in areas such as medicine, energy, climate, manufacturing, etc., but also imposes unnecessary underestimation of the infrastructure necessary for future Big Data, causing unnecessary divide between traditional IDCs whose present-day "Big Data Infrastructure" often being a group of re-purposed web-server connected by at best 10GbE or even 1GbE endpoint speeds; this is in contrast with interconnects of supercomputers today offering a few orders of magnitude faster interconnects for massive data exchange for sophisticated processing of data.

[1] Tokyo Institute of Technology, Tokyo, Japan
[2] University of Tsukuba, Tsukuba, Japan
[3] National Institute of Informatics, Tokyo, Japan
[4] IBM Research / University College Dublin, Dublin, Ireland
[5] RIKEN Advanced Institute for Computational Science, Chuo-ku Kobe, Japan

As such, many instances of big data are not so “big” in terms of capacity nor processing complexity, the latter often being simple mining to detect simple statistical trends, and/or associativity with a small number of classes of silo’d datasets within an organization. This is why simple data processing abstractions on simple hardware, such as Hadoop[1] running on commodity servers, is widely employed. However, the future of big data is not expected to be the case. There are various predictions on “breaking down of silos” where organizations will open up their data for public consumption, either for free or for a fee, along with immense increase in varieties of data sources driven by technologies such as IoT[12]. There, meaningful information would be extracted from unstructured and seemingly uncorrelated data spanning exabytes to zetabytes, utilizing higher-order $O(n \times m)$ algorithms on irregular structures such as graphs, as well as conducting data assimilations with massive simulations in petaflops to even exaflops. This is already happening in data-intensive science, in areas such as particle physics[2], cosmology[8], life science[13], where sharing of large capacity research data as “open data” has become domain practice; there is strong likelihood that this will proliferate to the common Internet, just as the Web, which was originally envisioned to share scientific hypertext, took over the world as the mainstream information sharing IT infrastructure.

We refer to such evolved state of big data as “Extreme Big Data”, or EBD for short, as a counterpart to extreme computing. An IT infrastructure supporting EBD will involve massive requirements of compute, capacity and bandwidth of resources throughout the system, as well as co-existence of efficiency and real-time resource provisioning, as well as flexible programming environment and adaptability of compute to data locations to minimize the overall data movement Moreover, they have to be extremely power and space efficient, as those factors are the principle parameters nowadays that limit the overall capacity of a given IT system.

Given such requirements, our claim is that, neither existing supercomputers, nor traditional IDC Clouds, are appropriate for the task; rather, we believe that the convergence of the two are necessary, based on the upcoming technology innovations as well as our own R&D to actually achieve such effective convergence. These include extensive and hierarchical use of new generations of non-volatile memory (NVM) as well as processor-in-memory technologies to achieve high capacity in memory and processing with very low power; high-bandwidth many-core processors that can make use of such memory composed in a deep and hierarchical fashion; low latency access of elements of such memory hierarchy, especially NVMs, from all parts of the machine via a scalable, high-bandwidth, hi-bisection network; management of memory objects dispersed and resident throughout the system across application boundaries as *EBD Objects* in the hierarchy, as well as their automated performance tuning and high resiliency; various low-level big-data workload algorithms such as graph algorithms and sorting of various types of keys; various libraries, APIs, languages, as well as other programming abstractions for ease-of-use by the programmer, hiding the complexity of such a large and deep system; finally, resource management to accommodate complex workflows of both batch and real-time processing, being able to schedule tasks balancing the processing requirements versus minimizing data movement.

With such comprehensive overhaul of the entire system stack, coupled with advances in both computing and storage, we expect that we could amplify the EBD processing power of existing Cloud datacenters by several orders of magnitude. Our latest project titled “Extreme Big Data”, sponsored by the Japan Science and Technology Agency (JST), under the research area program “Advanced Core Technologies for Big Data Integration” of the Strategic Basic Research Program (CREST), embarked on a 5-year research to develop such EBD technologies during the period of Oct. 1st, 2013 to Sep. 30th, 2018. The project involves six internal teams, three being in EBD systems area and the other three in EBD applications, in order to pursue co-design of the EBD system stack. Figure 1 describes the overall scheme of the convergence of the HPC and the big data architecture, and the necessity of the co-design with EBD grand challenge applications.

In the project we plan on developing individual technological elements of EBD as mentioned above, such as designing the future HPC – Big Data convergent architecture, including memory hierarchy and network designs, as well as various algorithms and programing frameworks, along with acceleration of system-wide data and resource management. The co-design applications include life sciences, social simulations, and data assimilation in climate modeling, each with different system-level as well as API requirements for EBD. We will also pursue attaining top-level performance in big data benchmarks, such as the Graph 500[5], which measures the absolute performance of graph processing, as well as the Green Graph 500[6], which ranks the power efficiency of machines when processing the graphs under the Graph 500 rule. In fact, on the November 2013 edition of the Green Graph 500, we achieved No.1 and No.4 in the world, the latter result attained while offloading most of the graph data structures onto a Flash NVM. Sorting is another important benchmark, where we aim to challenge Petabyte sorting speeds on modern many-core processors such as GPUs and Xeon Phis. Finally, we plan on integrating results from other groups working on large-scale big data hardware/software stack. Some of the candidate systems include the Berkeley Spark[9] and ORNL Adios[4]. Figure 2 describes the overall scheme, which instantiates some of the details from Figure 1

The intermediate results of the EBD project will be implemented on our next generation TSUBAME3.0 supercomputer, the successor to the highly successful TSUBAME2.0[24], to be commissioned in the first half of 2016. In the current design of TSUBAME3.0 is being done as the phase 2 of the overall evolution of EBD architecture, and in that sense, TSUBAME3.0 could be the first EBD convergent architecture in production.

The rest of the paper provides an overview of each element of our EBD project, focusing on the candidate architecture, followed by synopsis of the system and application groups and their status quo as of June, 2014, 9 months into the 5-year project.

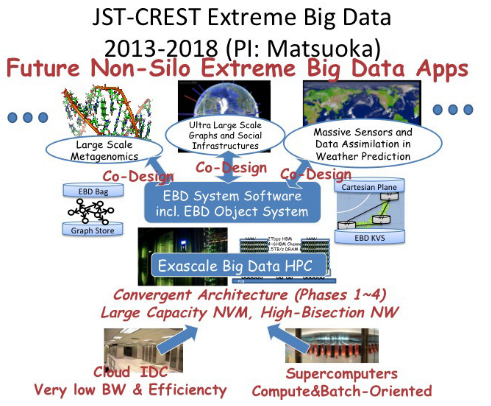

JST-CREST Extreme Big Data
2013-2018 (PI: Matsuoka)
Future Non-Silo Extreme Big Data Apps
Large Scale Metagenomics
Ultra Large Scale Graphs and Social Infrastructures
Massive Sensors and Data Assimilation in Weather Prediction
Co-Design
Co-Design
Co-Design
EBD Bag
Graph Store
EBD System Software incl. EBD Object System
Cartesian Plane
EBD KVS
Exascale Big Data HPC
Convergent Architecture (Phases 1~4)
Large Capacity NVM, High-Bisection NW
Cloud IDC
Very low BW & Efficiencty
Supercomputers
Compute&Batch-Oriented

Figure 1. Overview of EBD project

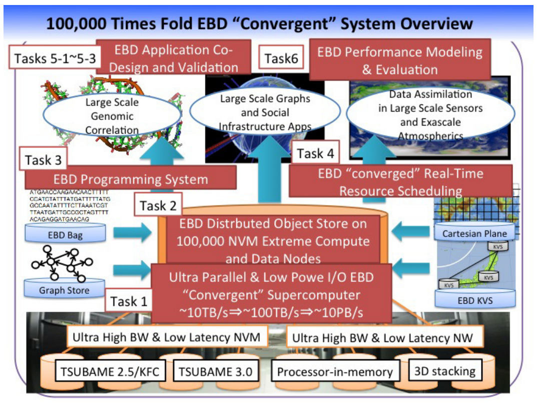

100,000 Times Fold EBD "Convergent" System Overview
Tasks 5-1~5-3
EBD Application Co-Design and Validation
Task6
EBD Performance Modeling & Evaluation
Large Scale Genomic Correlation
Large Scale Graphs and Social Infrastructure Apps
Data Assimilation in Large Scale Sensors and Exascale Atmospherics
Task 3
EBD Programming System
Task 4
EBD "converged" Real-Time Resource Scheduling
Task 2
EBD Distrbuted Object Store on 100,000 NVM Extreme Compute and Data Nodes
EBD Bag
Graph Store
Cartesian Plane
EBD KVS
Task 1
Ultra Parallel & Low Powe I/O EBD "Convergent" Supercomputer ~10TB/s⇒~100TB/s⇒~10PB/s
Ultra High BW & Low Latency NVM
Ultra High BW & Low Latency NW
TSUBAME 2.5/KFC
TSUBAME 3.0
Processor-in-memory
3D stacking

Figure 2. EBD software stacks

## 2. EBD Architecture

We will first discuss the envisioned EBD architecture. We will construct a series of EBD architecture prototypes, either by adding additional resources to existing machines such as the TSUBAME-KFC, or by building one entirely new. For both types of machines, the intent is to assess the behavior of EBD applications of the future; for this purpose, we will identify the various EBD benchmarks, as well as the EBD co-design applications, in order to model the performance requirements of several classes of EBD applications running on the prototype hardware.

At the same time, we will work with various vendors we have relationships with in our TSUBAME project. Of particular importance is the design of the overall memory hierarchy in relationship to the massively parallel, many-core processors such as the GPU or Xeon Phi that are representative of the current generation of hardware. Non-volatile memory will be used judiciously in order to attain large memory and storage capacity while preserving performance as much as possible. The four phases of the prototypes are being planned as follows: the first phase will involve standard mSATA SSD devices; second generation will involve much faster M.2 flash devices and large number of I/O channels directly utilizing fast I/O busses such as PCI-e; the third generation will incorporate next generation NVM directly connected to memory or some type of coherent bus such as HyperTransport or NVIDIA NV-Link; the third generation will involve placing flash memory or preferably newer generations of NVM on the memory bus of the CPU along with the DRAM, greatly enhancing the bandwidth and lowering the latency; finally, the future, forth generation hardware will integrate NVM, DRAM, and the lightweight data-centric processing core on the same 3-D die-stacking configuration, possibly integrated with heavyweight scalar-centric core that provides for performance in low-threaded workloads (Figure 3). For the last architecture, the traditional I/O bandwidth will equal the memory bandwidth on 3-D.

For each generation, we will be building a prototype, and/or creating a simulation environment to investigate their performance implications.

- Phase1 (2010- TSUBAME2.0-2.5 generation — Flash SSD local burst buffer): Each node will embody one or more server-grade SSD in place of HDDs, resulting in capacity of 100GB-1TB range. Although such configuration is becoming more commonplace, in 2010 the durability of SSDs were substantially questioned under heavy generalized HPC workload for TSUBAME2.0, which for the first time for a petascale supercomputer embodied 60GB $\times$ 2 sever-grade SLC SSDs per each node. Fortunately, such a concern has been disproven, for the nearly 3,000 SSDs in TSUBAME2.0 has seen very small number of failures, despite many of the localized and thus heavy I/O workloads being delegated to the SSDs. The aggregate capacity of SSDs in TSUBAME2.0 was 190 Terabytes, with approximately 400-500GB/s of I/O Bandwidth.
- Phase2 (2015- TSUBAME3.0 generation — large capacity flash with small embedded SSD modules): Each node will be designed from ground-up to embody multiple small high-bandwidth SSD modules such as the M.2 module with PCI-e connection. This will allow for pseudo-direct DMA access of the flash modules, and allows for global management of all the SSDs aggregated as a single namespace entity. Aggregate capacity can be as high as 10 Petabytes, with 10 Terabyte/s I/O bandwidth, which is several times faster compared to all the Top500 machines, whose I/O bandwidth does not exceed 2 Terabytes/s.

- Phase3: (2017-2020 Non Volatile Memory Modules): By placing the non-volatile memory modules onto the memory bus, in the form of DIMM or other modules, will allow for great increase in bandwidth and capacity while lowering the latency considerably, with DIMM modules affording bandwidths of 25 Gigabyte/s or greater. However, since the system will have to be composed of both DRAM and NVM DIMMs due to durability as well as performance issues, effective algorithms to make use of each memory region according to the access characteristics of memory objects must be devised. I/O bandwidth can become as high as Petabyte/s range depending on the NVM technology.
- Phase4: (2020-): (3-D integration of NVM and DRAM and PIM) The ultimate evolution of the architecture is 3-D stacking of NVM and DRAM on a low power processor, which is in effect a PIM (Processor-In-Memory) in modern incarnation3. Optionally, large cores can be integrated using 2.5D interposer technology for low latency, low-threaded processing. Such an architecture will totally unify NVM with DRAM, with tremendous bandwidth while being low power with 3-D stacking. I/O bandwidth, which is essentially on par with the memory bandwidth, can reach 10s to 100s of Petabytes/s, achieving a Yottabyte/year data processing capability.

As such, the evolution of the I/O bandwidth can be high as 4-5 orders of magnitude in 10 years, from hundreds of Gigabytes/s for TSUBAME2.0 in 2010, to 10s of Petabytes/s for the 2020- fourth generation EBD machine. The issue then, is what are the system software, programming paradigms, basic EBD algorithms, and applications that could best utilize such immense and rapid increase in performance.

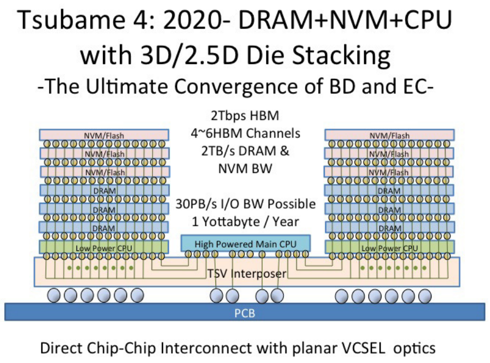


**Figure 3.** EBD architecture

## 3. EBD System Software

As described in previous section, EBD machines are based on various novel technologies derived from extreme-scale supercomputing, such as many-core parallel processing, ultra high bandwidth/low latency networks, non-volatile memory techniques, and high performance database techniques, etc.. Therefore, we have to develop completely redesigned new software stacks that support EBD architecture, since existing conventional system software can not support such new hardware device technologies.

### 3.1. Many-core I/O

Many-core processors such as GPU can provide extremely fast computational power and high memory bandwidth; however, the capacity of memory on GPU devices are limited in size to accommodate EBD applications. In order to mitigate the GPU capacity problem, we have designed a novel I/O prototype machine that consist of multiple GPU accelerators and multiple mini SATA (mSATA) SSD devices. In particular, aggregation of multiple mSATA SSDs and strategy for utilization the GPU and NVM devices are essential to perform high bandwidth, high IOPS as well as large capacity and low energy consumption. Our preliminary results based on basic I/O bandwidth to the prototype with 16 of mSATA SSD devices by fio I/O benchmarks exhibit that the sequential read bandwidth achieves 7.69GB/s (92.4% of theoretical peak), and the write bandwidth achieves 3.8GB/s (90.2% of theoretical peak). The results also exhibit that using multiple mSATA SSDs performs 3.20 to 7.60 times faster than a common PCI-e attached flash memory device, and 11.1 to 17.8 times faster than conventional SSDs attached on a local compute node of TSUBAME 2.5. We also measure the performance of a matrix-vector multiplication benchmark that overlaps file I/O from/to NVMs, memory copy between host and GPUs, and computation on GPUs. The results using 280MB to 140GB of input matrices, which exceed the GPU memory capacity, exhibit that our prototype can achieve 3.06GB/s from 8 mSATA SSDs to a GPU device by using the RAID0 configuration with appropriate stripe size. We also found that using pinned memory and setting small chunk size for overlapping significantly affect data transfer performance.

### 3.2. Programming Framework

EBD architecture with deep memory hierarchy is encapsulated as EBD objects. In order to operate EBD architecture from applications, we provide a programming framework for the EBD objects based on the system-level programming model and the user-level programming model. The system-level programming model provides specific interfaces and concrete implementations for EBD objects to collect hardware information and to control data layout, while the user-level programming model provides several features to control EBD objects from applications.

As an example of ongoing work on the EBD programming framework, we have developed a MapReduce-style programming framework, called HAMAR (Highly Accelerated MapReduce) for massively data parallel processing on EBD machines with deep hierarchical memory [26]. Our framework automatically handles memory overflows from GPUs by dynamically dividing data into multiple chunks and overlaps CPU-GPU data transfer and computation on GPUs as much as possible. Our experimental results for PageRank graph applications on TSUBAME2.5 using 1024 nodes (12288 CPU cores, 3072 GPUs) exhibit that our GPU-based implementation performs 2.10x faster than running on CPU when the size of graph exceeds the capacity of GPU's device memory.

### 3.3. Basic Algorithms

Based on novel features of of massively parallel many-core processors and nonvolatile memory devices, we develop basic algorithms, such as indexing, multi dimensional array matching, sorting, unstructured graph search, spatial clustering, etc., to support EBD applications.

As an instance of implementation to overcome deepening memory hierarchy in extreme-scale computing systems, we have developed a fast graph processing implementation with a novel graph offloading technique using NVMs for Hybrid BFS (Breadth-First Search) algorithm that is widely used in the Graph500 benchmark [21]. Hybrid BFS uses a mixture of two approaches, a conventional top-down approach and a bottom-up approach by changing search directions using parameters. Based on the DRAM-based NUMA-optimized Hybrid BFS implementation called NETAL (NETwork Analysis Library), which achieves 10.5 GTEPS on the Graph500 list (November 2012), we carefully offload infrequent accessed graph data to secondary semi-external memory devices such as NVMs and directly read the data on the devices on demand. The results using Graph500 problems shows that our approach with highly localized data access can achieve competitive performance with the conventional DRAM only approach, although we aggressively extend memory footprints onto NVMs. Using the implementation with some improvements, we have also achieved 4.35MTEPS/Watt on a Scale 30 problem and ranked the 4th position in the big data category in the Green Graph500 (November 2013), which is the 1st position using a single commodity server in the big data category.

Large-scale distributed sorting is another instance for EBD basic algorithms. Splitter-based parallel sorting algorithms are known to be highly efficient for distributed sorting due to their low communication complexity. Although using GPU accelerators could help to reduce the computational cost in general, their effectiveness in distributed sorting algorithms on large-scale heterogeneous GPU-based systems have not been well studied. We accelerate the existing HykSort algorithm by offloading costly computation phases to GPU devices. Preliminary investigations show that the local sort phase is dramatically accelerated by GPUs, while the merge phase achieves negligible performance improvement. We then evaluate the performances of our implementation with only local sort acceleration on the TSUBAME2.5 supercomputer that comprises over 4000 NVIDIA K20x GPUs. Performance evaluation of weak scaling shows that we achieve 389 times speedup with 0.25TB/s throughput when sorting 4TB 64bit integer on 1024 nodes compared to running on 1 node; on the other hand, for CPU vs. GPU comparison, our implementation achieves only 1.40 times speedup using 1024 nodes.

## 4. Distributed Object Store for EBD Applications

Extreme big data (EBD) applications may involve parallel access from handreds of thousands of processes, which requires not only scalable performance of bandwidth but also I/O operations per seconds (IOPS) to the number of processes. Important issue for scalable I/O bandwidth is to maximize access locality to read and write data, and to accomodate parallel access of metadata such as location of the data, from hundreds of thousands of clients. To improve the read access locality, data location aware process scheduling is essential, which assigns processes depending on the input data location. Regarding the write access, it is necessary to find a space to maximize the locality that may depend on successive processes that read the data as an input.

This approach is adopted by Google file system [18] and MapReduce [15]. The Google file system stores data blocks at local storage on compute nodes. The data blocks are replicated to improve reliability and availability. Jobs of the MapReduce are allocated on compute nodes considering the data block locations. Open source Apache Hadoop [1] implements the design of Google file system and MapReduce, and plenty of research follows [14, 39, 40]. The Gfarm file system [35] and the Pwrake workflow system [33] cover general workflow execution using this locality aware approach. The Gfarm file system also stores data at local storage on compute nodes. The Pwrake workflow system executes a workflow written in Rakefile in parallel assigning processes on a compute node where the input data is stored. The Pwrake, moreover, allocates processes to minimize data transfer size among compute nodes during a whole workflow execution by multi-constraint graph partitioning [34], which can reduce data transfer size of intermediate data generated during the workflow execution.

EBD applications do not really require posix file system interface. It requires rather key-value interface but having additional features. The first additional feature is specifying a collection of key-value pairs. We call this collection *a bag*. The second feature is a range query of keys to form a bag. How to form a bag is still under discussion, but the range query is considered to be a basic operation to select key-value pairs. The third feature is to specify parallel operations for one or two bags. When one bag is specified, the operations will be applied to each key in the specified bag in parallel. When two bags are specified, they will be applied to all combinations of two keys in each bag in parallel, which supports $O(mn)$ complexity computation when bags include $m$ and $n$ keys, respectively. The forth feature is a parallel query to analyze the output data.

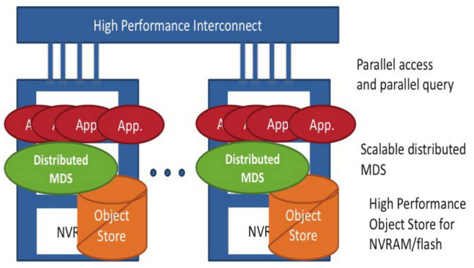

High Performance Interconnect
App
App.
Distributed MDS
Object Store
NVR
Parallel access and parallel query
Scalable distributed MDS
High Performance Object Store for NVRAM/flash

**Figure 4.** EBD distributed object store

We are designing an EBD distributed object store that has the above features, which consists of distributed metadata server and local object stores (Figure 4). Regarding a local object store, we assume flash storage device and non-volatile storage class memory in which the I/O performance is improved by parallel data access. Hard disk drives have been a major storage device, which performs well for sequential contiguous access, but not for parallel random accesses. On the other hand, in case of flash storage and non-volatile storage class memory, the I/O performance is improved by accessing in parallel. We design a nonblocking local object store that does not require a lock for concurrent accesses using the OpenNVM API [3]. The OpenNVM API provides a sparse address space and an atomic write operation. Traditional file system design manages blocks of file data using multi-level indirect reference. Instead, our design is based on a large-size region using the sparse address space, which avoids the overhead of indirect references. A preliminary evaluation shows scalable IOPS performance when the number of thread increases. When using 16 threads, it achieves 190 Kops/sec, while IOPS of existing directFS and XFS is not improved and stays at 61 and 16 Kops/sec, respectively.

Regarding a range query of keys, we are currently designing in-memory lock-free concurrent B+Tree. Supported operations are search, insert and delete. Dynamic rebalancing of the tree, i.e. tree nodes merge and split, is also supported. Nodes to be split or merged are frozen until replaced by new nodes. Search is not delayed by rebalancing. Search could access old (frozen) and new nodes. New Nodes may be modified for insert or delete only after replacement took place. Insert and delete operations help finalize the replacement of encountered frozen nodes. Currently the design is under the validation.

## 5. EBD Interconnection Networks

### 5.1. Objective

Extreme big data (EBD) processing would generate different access patterns from traditional stencil and uniform communications in parallel scientific applications and lightweight communications in datacenters. It sometimes becomes access to remote flash devices, while traditional HPC applications rely on remote direct memory access. In this section we will design interconnection networks for such EBD processing. Figure 5 shows our quantitative goals of EBD interconnection networks. Our main challenges are (1) communication to remote flash in low latency, i.e. less than $10\mu$s for 4kB transfer, and (2) optimization of non-regular access that may be determined by each execution. In this context we will attempt to make a new network topology and its custom deadlock-free routing, and fine-grain direct communication mechanism to storage. It is free from TCP/IP basis and we will support native communication, e.g. IBverb on InfiniBand.

### 5.2. Existing Datacenter and Supercomputer Networks

Existing interconnection networks for massively parallel computing can be classified into datacenter and supercomputer networks. Typical datacenter networks (DCNs) are built with a hierarchical structure with so-called top-of-rack (ToR) switches, aggregation switches, cluster routers, and border routers [10]. Since Ethernet has a good economy especially for 1Gbps and 10Gbps links and switches, it commonly used in DCNs. Fat tree or tree topologies are commonly used in DCNs, because they fit with its layered structure and user partitioning. By contrast, supercomputer networks use high-bandwidth links, such as 40Gbps, with custom routing on regular tightly-coupled topologies, such as $k$-ary $n$-cubes. Since InfiniBand is frequently used in supercomputer networks, it can support arbitrary topologies with their custom routings. Supercomputer networks are well optimized to regular communication patterns, such as stencil or uniform access by making the best use of regular structure. Our EBD interconnection networks should be different from both DCNs and supercomputer networks, because non-regular unpredictable access patterns are essential for EBD processing (see Figure 5).

### 5.3. Preliminary Design of EBD Interconnection Networks

#### *5.3.1. Network Topology*

Our preliminary design focuses on network topology. We attempt to make a metatopology in order to minimize jitter of network latency and average network latency for EBD direct storage communication that includes little regularity. For a given switch degree, we already propose a design framework of network topology[17]. The proposed topology introduces random connections to achieve low latency, but does so in a way that accounts for the physical layout of the topology so as to lead to further cable length and latency reductions[17]. Figure 6 is an example of the proposed topology and its layout. Graph analysis results[17] showed that the proposed topology has good properties in terms of latency, cable length, and throughput, when compared to traditional low-degree torus and moderate-degree hypercube topologies, to high-degree fully-connected Dragonfly topologies[22], to the HyperX[11] topology, and to recently proposed fully random topologies[23].

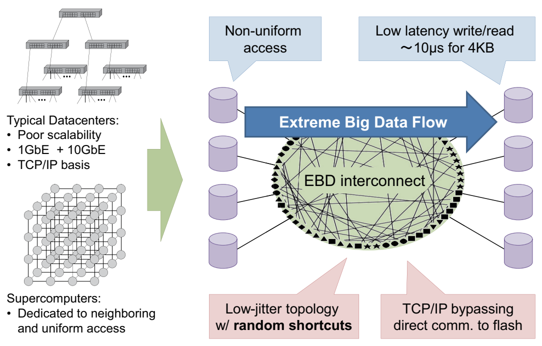

Typical Datacenters:
• Poor scalability
• 1GbE + 10GbE
• TCP/IP basis
Supercomputers:
• Dedicated to neighboring and uniform access
Non-uniform access
Low latency write/read ~10μs for 4KB
Extreme Big Data Flow
EBD interconnect
Low-jitter topology w/ random shortcuts
TCP/IP bypassing direct comm. to flash

**Figure 5.** An overview of EBD interconnection networks

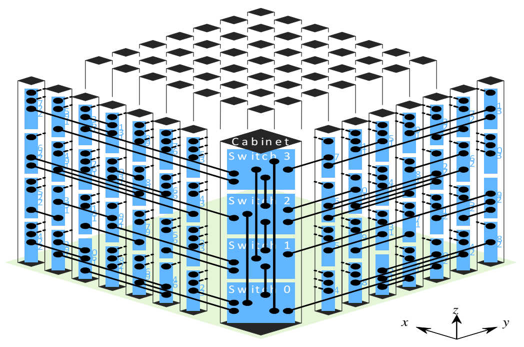


**Figure 6.** An example of the proposed network topology

### *5.3.2. Low-jitter Routing*

Our key challenge for the network topology that has a little regularity, such as one in the previous subsection, is to minimize the jitter of end-to-end latency for arbitrary pair of compute nodes. Historically a minimal or shortest-path deadlock-free routing has been used for interconnection networks. By contrast, in EBD interconnection networks, to minimize the variation of end-to-end network latency, our design allows to have non-minimal paths. We are also planning to use a quality-of-service (QoS) mechanism for this purpose.

A potential scalability issue is the computational cost of path computation for topology-agnostic deadlock-free routing, which is more complex than when routing on structured topologies (see the survey in [16]). However, this computation needs to be performed only when initially deploying the system and after a change in system configuration (e.g. due to link/switch failure). Consequently, concerns about path computation should arise only at extremely large scale[23].

### *5.3.3. Preliminary Evaluation*

We use a cycle-accurate network simulator written in C++ [23]. A header flit transfer requires at least 60 ns delay for a switch, including routing, virtual-channel allocation, switch allocation, and flit transfer from an channel to an output channel through a crossbar. Link delay is assumed to be 5 ns/m on an optical cable and the length of each cable is computed based on the cabinet layout. We use deadlock-free low-jitter routing for the proposed topology. We use three synthetic traffic patterns that determine source-and-destination pairs, namely *random uniform*.

Figure 7 shows latency (averaged over all messages) vs. accepted traffic (the load that is injected into the network in Gbps/host) for all three traffic patterns, for networks with 256 switches located in 64 cabinets. Our EBD interconnection networks, named as “Skywalk” in the figures, achieves latency lower than that of Random for the same degree. Only Dragonfly, resp. HyperX, has latency lower than Skywalk but with a much higher degrees of 19, resp. 17, when compared to Skywalk’s degree of at most 11 in these results[17].

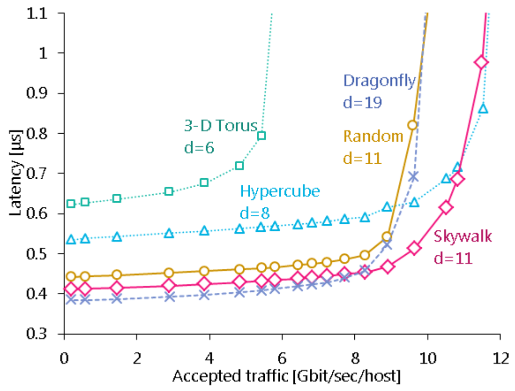


**Figure 7.** Average latency vs. throughput (results are taken from our previous work [17]). "Skywalk" is the design for our EBD interconnection networks.

## 6. Proxy EBD Applications

### 6.1. Metagenome Analysis

Today, the production of biological and medical data has been rapidly increased year by year. Particularly, the increase of genomic data is outstanding because of the improvement of DNA sequencers, which are generally called next-generation sequencers. The latest sequencer, Illumina's HiSeq 2500, can produce several hundred billions of base pairs (bp) of sequence data on a single run of the machine. The throughput of the system is approximately 10,000 times higher than that of previous sequencers. However, current most sequencers only produce information in short fragments of DNA sequences called reads. Thus, it is necessary to perform various analyses based on the DNA reads on a computer for obtaining biologically useful information. The current genome analyses require not only computation power but also huge storage and I/O performance because the analyses have to read huge amount of queries and a large reference sequence database and write the results whose size is often equal to or more than that of queries.

#### *6.1.1. Emerging requirements on metagenome analysis*

Metagenome analysis is the study of the genomes of uncultured microbes obtained directly from microbial communities in their natural habitats. The analysis is useful for not only understanding symbiotic systems but also watching environment pollutions [36]. However, the analysis has to deal with multiple organisms simultaneously and thus the size of the query sequences become extremely big data. It is often hundreds times larger than that of general genome analysis. In addition, metagenome analysis requires comparisons of sequence data obtained from a sequencer with sequence data of remote homologues in databases. Therefore, sensitive sequence homology searches and huge reference sequence database, which contains genome sequences of all known organisms, are required in metagenome analysis. As a result, comparison of EBD (queries) versus EBD (reference databases) is required, and there are serious problems both in the requirement of computation power and I/O performance.

For dealing the requirement of computation power on metagenome analysis, we have already developed several novel sequence search algorithms and tools for metagenome analysis. GHOSTM is a GPU-accelerated homology search tool and achieved calculation speeds that were 130 times faster than BLAST [28]. GHOSTX employs suffix array-based fast algorithm and achieved approximately 131-165 times acceleration over BLAST at similar levels of sensitivity [29]. However, by using the fast search algorithm, the problem of I/O performance had become more serious. Figure 8 shows the speedup of the GHOSTM-based metagenome analysis on TSUBAME2.0 of Tokyo Institute of Technology. The speedup drastically deceases at 2,520 GPUs because of insufficient load balancing and I/O performance of the system [27]. The result indicated the more sophisticated system especially for I/O handing is required for large-scale metagenome analysis.

Now, we are developing a novel metagenomic data analysis pipeline on TSUBAME2.5 and K computer at the RIKEN Advanced Institute for Computational Science. To improve the load balancing and I/O processes, we avoided using a simple batch job system, and developed "mpidp" that is a versatile job distribution framework based on Master-Worker model. The system, GHOST-MP, has already showed clearly better scaling than the previous system on TSUBAME2.5 (Figure 9)

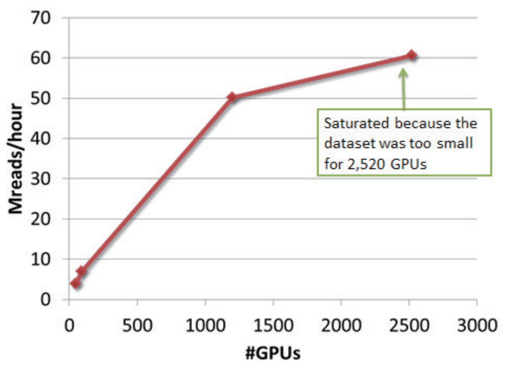


**Figure 8.** Speedup of GHOSTM-based pipeline on TSUBAME2.0

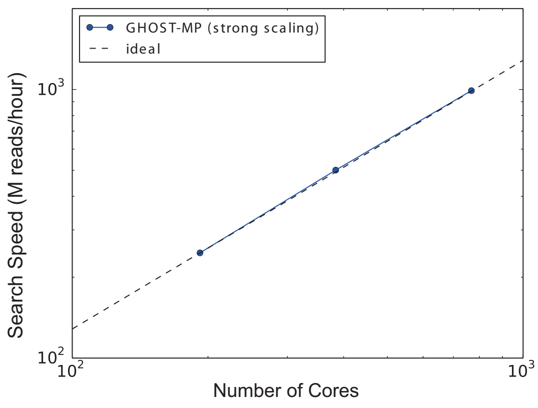


**Figure 9.** Speedup of GHOST-MP based pipeline on TSUBAME2.5

### *6.1.2. Metagenome analysis of human oral microbiome*

We are now analyzing huge amount of human oral microbiome metagenome data released by human microbiome project (HMP) [19] on K computer. The data consists of 418 samples from various parts of human oral cavity, and includes approximately 26 billion reads, which is more than 5Tb (base pairs) (Table 1). For every query read sequences, we have performed sensitive sequence homology searches to well-annotated reference sequence database (KEGG genes) and revealed evolutionally related proteins. Several biological analyses of the results, which include analysis of active genes on metabolic pathways (Figure 10), have been finished but the most of analyses are now ongoing.

### *6.1.3. Novel APIs for EBD processing*

Through the large-scale metagenome analysis on K computer and TSUBAME2.5, the bottlenecks of the pipeline have been gradually emerged. In addition, the "mpidp" is a simple tool and is not encapsulated. To use the current "mpidp", the users have to write additional codes and instruct the disk accesses. Therefore, we are now designing novel APIs for processing EBD such as metagenome data. Currently, we focus on metagenome analysis. However, the other applications such as protein-protein interaction prediction [25] and *in silico* screening of

**Table 1.** The detail of human oral metagenome data used in the analysis

| Subsite | # samples | # M reads | # base pairs (Gb) |
|---|---|---|---|
| Saliva | 5 | 278 | 56 |
| Keratinized gingiva | 6 | 361 | 73 |
| Buccal mucosa | 121 | 7478 | 1485 |
| Hard plate | 1 | 54 | 11 |
| Palatine tonsils | 6 | 373 | 74 |
| Subgingival plaque | 8 | 517 | 104 |
| Supragingival plaque | 128 | 7965 | 1595 |
| Throat | 7 | 393 | 79 |
| Tongue dorsum | 136 | 8708 | 1743 |
| Total | 418 | 26131 | 5220 |

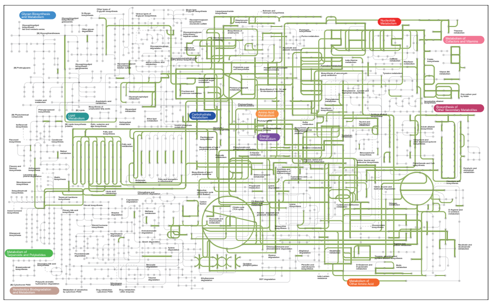


**Figure 10.** Active pathways in whole supragigival plaque samples

drug compounds also have to read and write huge amount of data, and importantly require a cross matching between EBD (query) versus EBD (reference). We will also perform requirement analysis for these applications in order to make the APIs be more universal.

## 6.2. Large-Scale Graph Analytics and Billion-Scale Social Simulation

This section introduces some of the example applications handling extremely big data with supercomputers such as large-scale network analysis, X10-based large-scale graph analytics library, Graph500 benchmark, and billion-scale social simulation.

### *6.2.1. ScaleGraph: X10-Based Large-Scale Graph Analytic Library*

Recently, large-scale graph analytics has become a very popular topic owing to the emergence of gigantic graphs whose number of vertices and edges is in millions, billions or even trillions. Many graph analytics libraries and frameworks have been proposed with various computational models and programming languages to deal with such graphs. X10 programming language is a PGAS language that aims at both software performance and programmer's productivity. We introduce ScaleGraph library [7] developed using X10 programming to illustrate the use of X10 for large-scale graph analytics. ScaleGraph library provides XPregel framework that is inspired by Google's Pregel computation model, serving as a building block for implementing graph kernels. We also optimized X10 runtime in some parts such as collective communication, String data representation, data structure, and file IO, to achieve further performance. As of this writing, the library supports varous parallel and distributed graph kernels such as PageRank, spectral clustering, degree distribution, betweenness centrality, HyperANF, strongly-connected component, maximum flow, single source shortest path, and breadth first search.

We evaluated the performance and scalability of all graph kernels available in ScaleGraph libraries on top of the TSUBAME 2.5 supercomputer with up to 128 nodes. The result shows that most kernels could solve billion-scale artifical data following social network structures and also achieve moderate scalability. We also performed the evaluation using the Twitter network as 2012/10 whose graph is composed of 496 millions of users and 28.5 billions of following relationships. The library successfully handles such a gigantic graph on 128 of machine nodes. To the best of our knowledge, ScaleGraph is the first X10-based library to address performance, scalability and productivity issues in dealing with large-scale graph analytics.

### *6.2.2. Graph500 Benchmark*

As an alternative to Linpack, Graph500 [5] was recently developed. We conducted a thorough study of the algorithms of the reference implementations and their performance in an earlier paper [32]. Based on that work, we implemented a scalable and high-performance implementation of an optimized Graph500 benchmark for large distributed environments [37][38]. In contrast to the computation-intensive benchmark used by TOP500, Graph500 is a data-intensive benchmark. It does breadth-first searches in undirected large graphs generated by a scalable data generator based on a Kronecker graph. There are six problem classes: toy, mini, small, medium, large, and huge. Each problem solves a different size graph defined by a Scale parameter, which is the base 2 logarithm of the number of vertices. For example, the level Scale 26 for toy means 226 and corresponds to 1010 bytes occupying 17 GB of memory. The six Scale values are 26, 29, 32, 36, 39, and 42 for the six classes. The largest problem, huge (Scale 42), needs to handle around 1.1 PB of memory. As of this writing, Scale 40 is the largest that has been solved by a top-ranked supercomputer. Our work [37] proposed an optimized method based on 2D partitioning and other methods such as communication compression and vertex sorting. Our optimized implementation can handle BFS (Breadth First Search) of a large graph with Scale 35 with 462.25 GE/s while using 1366 nodes and 16,392 CPU cores. This competition is greatly challenging since new scalable algorithms have been proposed rapidly. We have been continuously enhancing the scalable algorithm and implementation on various supercomputers.

### *6.2.3. Billion-Scale Social Simulation*

We introduce billion-scale social simulation [30][31] in this section. Towards the contribution to the human society, global economy, ecology, the analysis of human brain characteristics and our daily life, the research in multi-agent simulation is entering into the era of simulating billion-scale agents. Although prior arts tackle distributed agent simulation platform to achieve this goal, it is not sufficient to simulation billion-scale agent behaviors. Based on this observation, we report the first effort for building such an infrastructure platform that handles billion-scale agent simulation platform. In our previous work, we introduce X10-based agent simulation platform for such a purpose and presents its application to traffic simulation. We were able to handle only at maximum 10 millions of agents, but the performance was not scalable due to various reasons such as work load imbalance, global synchronization. Our work in [31] present the work of purely implementing the whole simulation stack including both the simulation runtime and the application layer such as traffic simulation by the use of the state-of-the-art PGAS language. By implementing the system in such a manner and evaluating the system in highly distributed systems, it is observed that the system can be close to handle billion-scale agents in near real-time. The first experimental result is that the performance scalability is greatly achieved by simulating 1 millions of agents on 1536 CPU cores and 256 nodes in the TSUBAME 2.5 supercomputer. By compiling fully X10-based agent simulation system into C++ and MPI, it only takes 77 seconds for 600 simulation steps which is nearly 10 times faster than real-time. Moreover, by using the entire whole country-wide network of India as the agents underlying infrastructure, we successfully simulated 1 billion agents with 400 nodes. This is the first attempt to deal with such a gigantic number of agents and we believe that this infrastructure would be the basis of large-scale agent simulation in various fields.

## 6.3. Big Data Assimilation in Weather Forecast Applications

Contemporary weather forecasting relies on numerical simulations, and the accuracy of numerical weather prediction (NWP) is very sensitive to the accuracy of initial conditions or estimates of the current state of the atmosphere. Data assimilation provides the optimal combination of numerical simulations and observational data based on statistical mathematics and finds accurate initial conditions for more accurate forecasts.

The EBD issue arises in NWP due to the rapid increase of data volume in numerical simulations and observations. With leading-edge high-performance computing and sensing technologies, we may design an extremely demanding NWP system by a factor of 100 or more, that is what we call the “Big Data Assimilation (BDA)” revolution. We foresee that the future of NWP would be directed to the BDA, that is considered to be an important EBD application.

The rapid development of high-performance computing technologies enables higher-resolution simulations, which in turn enable an effective use of dense and frequent observations that have not been used well previously. In fact, dense observations from satellites are thinned if the numerical simulations cannot represent the small-scale phenomena captured by satellites. Also, new sensing technologies enable new types of sensors that produce data at a rate greater by two orders of magnitude than the current sensors. For example, the phased array weather radars (PAWR) implemented at Osaka and Kobe provide three-dimensional scanning of raindrops every 30 seconds at a 100-m resolution with 100 elevation angles, these numbers compared with about 15 elevation angles in 5 minutes or so in conventional Doppler radars. The next-generation geostationary satellite “Himawari-8” of the Japan Meteorological Agency (JMA) is capable of scanning a few limited areas in 30 seconds, compared with the current rapid scan in 5 minutes.

The data produced by the new types of sensors are extremely rapid and useful for capturing smaller-scale phenomena. Severe weather events may be caused by a single cumulonimbus cloud at a few km scale with lifetime of the order of 10 minutes. For such short phenomena, the linear tangent assumption made in most data assimilation methods would be valid only in the order of less than a minute. This motivates us to design a super-rapid NWP system using the every-30-second data from the new types of sensors. At this moment, only the leading-edge supercomputers can run simulations at a 100-m or higher resolution to make use of the dense and frequent data. About 2 Peta floating point operations are required to run 100 30-second simulations at a 100-m resolution. 100 simulations provide an approximate representation of the forecast uncertainties in the local ensemble transform Kalman filter [20], an advanced data assimilation algorithm. We may spend only 10 seconds for the 100 simulations for timely forecasts, requiring 200 TFLOPS effective performance; this is possible using a quarter of 10-Peta-FLOPS "K computer" if we assume relatively optimistic 10 % efficiency.

The 100 simulations produce 200 GB data, that are transferred to the data assimilation module in a few seconds. Data assimilation merges the 200 GB forecast data with observations in the past 30 seconds, and produces another 200 GB data for the next 30-second forecasts. The tandem forecast-assimilation procedure are repeated every 30 seconds as new data keep coming so frequently.

To build the revolutionary rapid-update NWP system with high reliability, a co-design of software and hardware is essential. The rapid data transfer between forecast and data assimilation modules requires sufficient hardware capacity with specialized software design. Also, we expect that massively parallel high-performance EBD hardware would fail at a certain rate, while the data will keep coming every 30 seconds. In case of hardware failure, we may be forced to skip several steps of data assimilation. We could take advantage of four-dimensional LETKF [20], in which the past data are treated simultaneously in a single data assimilation step. Namely, we could skip several steps during the hardware failure, but still keep all data used effectively.

The BDA system with 30-second update cycles is two orders of magnitude more rapid than the currently used NWP systems. The operational global NWP is updated typically every 6 hours for synoptic weather at  2000 km scale. At the highest frequency for mesoscale weather at  20 to 200 km scale, forecasts are updated every hour in regional NWP systems including the Rapid Refresh (RAP) at the National Centers for Environmental Prediction (NCEP) and the Local Forecasting Model (LFM) at JMA. The 100-times more rapid processing of BDA is extremely demanding, but it is worth exploring such a rapid system to prevent and mitigate disasters caused by local severe weather such as heavy rainstorm and tornadoes.

## 7. Conclusions

We have introduced and gave an overview of our latest EBD (Extreme Big Data) project, which aims to achieve future convergence of big data and extreme supercomputing. By utilizing existing as well as future technologies gearing towards exascale, such as massively parallel processors, non-volatile memory, 3-D memory stacking, as well as high-bandwidth optics network, a new EBD convergent architecture can be constructed with which exabytes of data can be effectively processed. On top, a new system software stack that maximizes the underlying hardware to allow for massive data to be processed fast, including components such as bottom-level communication layer, distributed object management, basic algorithmic and programming frameworks, resource management, etc., is being developed, with considerations for exploiting other big data software framework. In order to assure that the EBD application requirements are being properly reflected, the system group is engaged in a co-design activity with three EBD application groups, each encompassing different type of application area as well as requirements. The results of the research will be deployed not only onto prototype hardware, but also production machines such as TSUBAME3.0, which is slated to be deployed in the first half of 2016 as one of the first "EBD" convergent production machine. As mentioned the project has only begun, still in its first year, but we have already achieved some notable results such as being world #1 on the Green Graph 500 list.

*This research was supported by JST, CREST.*

## References

1. *Apache Hadoop.* http://hadoop.apache.org.
2. *Worldwide LHC Computing Grid.* http://wlcg-public.web.cern.ch/.
3. *OpenNVM.* https://opennvm.gitbug.io.
4. *The Adaptable IO System (ADIOS).* https://www.olcf.ornl.gov/center-projects/adios/.
5. *Graph500.* http://www.graph500.org/.
6. *Green Graph500.* http://green.graph500.org.
7. *ScaleGraph Library.* http://www.scalegraph.org/.
8. *Sloan Digital Sky Survey.* http://www.sdss.org/.
9. *Spark.* http://spark.apache.org.
10. Dennis Abts and Bob Felderman. A guided tour of data-center networking. *Commun. ACM*, 55(6):44–51, 2012.
11. Jung Ho Ahn, Nathan Binkert, Al Davis, Moray McLaren, and Robert S. Schreiber. HyperX: topology, routing, and packaging of efficient large-scale networks. In *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, pages 1–11, 2009.
12. Kevin Ashton. That 'internet of things' thing. *RFID Journal*, 2009.
13. Nathan Blow. Metagenomics: exploring unseen communities. *Nature*, 453 (7195):687–90, 2008. ISSN 1476-4687. URL `http://www.biomedsearch.com/nih/Metagenomics-exploring-unseen-communities/18509446.html`.
14. Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 2736–2743, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4108-2. DOI: `10.1109/CIT.2010.458`.
15. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.
16. Jose Flich, Tor Skeie, Andres Mejia, Olav Lysne, Pedro Lopez, Antonio Robles, Jose Duato, Michihiro Koibuchi, Tomas Rokicki, and Jose Carlos Sancho. A Survey and Evaluation of Topology Agnostic Deterministic Routing Algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):405–425, 2012.
17. Ikki Fujiwara, Michihiro Koibuchi, Hiroki Matsutani, and Henri Casanova. Skywalk: a Topology for HPC Networks with Low-delay Switches. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, May 2014.
18. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP

'03, pages 29–43, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. DOI: `10.1145/945445.945450`.

19. The Human and Microbiome Project. A framework for human microbiome research. *Nature*, 486(7402):215–21, June 2012. ISSN 1476-4687. DOI: `10.1038/nature11209`.
20. B. R. HUNT, E. KALNAY, E. J. KOSTELICH, E. OTT, D. J. PATIL, T. SAUER, I. SZUNYOGH, J. A. YORKE, and A. V. ZIMIN. Four-dimensional ensemble kalman filtering. *Tellus A*, 56(4):273–277, 2004. ISSN 1600-0870. DOI: `10.1111/j.1600-0870.2004.00066.x`.
21. Ryo Mizote Yuichiro Yasui Katsuki Fujisawa Keita Iwabuchi, Hitoshi Sato and Satoshi Matsuoka. Hybrid bfs approach using semi-external memory. In *International Workshop on High Performance Data Intensive Computing (HPDIC2014)*, May 2014.
22. John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 77–88, 2008.
23. Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. A Case for Random Shortcut Topologies for HPC Interconnects. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 177–188, 2012.
24. Satoshi Matsuoka, Takayuki Aoki, Toshio Endo, Hitoshi Sato, Shin'ichiro Takizawa, Akihiko Nomura, and Kento Sato. TSUBAME2.0. In *Contemporary High Performance Computing*, Chapman & Hall/CRC Computational Science, pages 525–555. Chapman and Hall/CRC, April 2013. ISBN 978-1-4665-6834-1. DOI: `doi:10.1201/b14677-23`.
25. Yuri Matsuzaki, Nobuyuki Uchikoga, Masahito Ohue, Takehiro Shimoda, Toshiyuki Sato, Takashi Ishida, and Yutaka Akiyama. MEGADOCK 3.0: a high-performance protein-protein interaction prediction software using hybrid parallel computing for petascale supercomputing environments. *Source code for biology and medicine*, 8(1):18, January 2013. ISSN 1751-0473. DOI: `10.1186/1751-0473-8-18`.
26. Koichi Shirahata, Hitoshi Sato, Toyotaro Suzumura, and Satoshi Matsuoka. A scalable implementation of a mapreduce-based graph processing algorithm for large-scale heterogeneous supercomputers. *Cluster Computing and the Grid, IEEE International Symposium on*, 0: 277–284, 2013. DOI: `http://doi.ieeecomputersociety.org/10.1109/CCGrid.2013.85`.
27. Shuji Suzuki, Takashi Ishida, and Yutaka Akiyama. An ultra-fast computing pipeline for metagenome analysis with next-generation dna sequencers. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1549–1550, Nov 2012. DOI: `10.1109/SC.Companion.2012.320`.
28. Shuji Suzuki, Takashi Ishida, Ken Kurokawa, and Yutaka Akiyama. GHOSTM: a GPU-accelerated homology search tool for metagenomics. *PLoS One*, 7(5):e36060, January 2012. ISSN 1932-6203. DOI: `10.1371/journal.pone.0036060`.
29. Shuji Suzuki, Masanori Kakuta, Takashi Ishida, and Yutaka Akiyama. GHOSTX: An improved sequence homology search algorithm using a query suffix array and a database suffix array. submitted.
30. Toyotaro Suzumura and Hiroki Kanezashi. Highly scalable x10-based agent simulation platform and its application to large-scale traffic simulation. In *Proceedings of the 2012*

*IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '12, pages 243–250, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4846-3. DOI: `10.1109/DS-RT.2012.44`.

31. Toyotaro Suzumura and Hiroki Kanezashi. A holistic architecture for super real-time multiagent simulation platform. In *Winter Simulation Conference 2013*, 2013.

32. Toyotaro Suzumura, Koji Ueno, Hitoshi Sato, Katsuki Fujisawa, and Satoshi Matsuoka. Performance characteristics of graph500 on large-scale distributed environment. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, IISWC '11, pages 149–158, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-2063-5. DOI: `10.1109/IISWC.2011.6114175`.

33. Masahiro Tanaka and Osamu Tatebe. Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 356–359, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8. DOI: `10.1145/1851476.1851529`.

34. Masahiro Tanaka and Osamu Tatebe. Workflow scheduling to minimize data movement using multi-constraint graph partitioning. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:65–72, 2012. DOI: `http://doi.ieeecomputersociety.org/10.1109/CCGrid.2012.134`.

35. Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm grid file system. *New Generation Computing*, 28(3):257–275, 2010. ISSN 0288-3635. DOI: `10.1007/s00354-009-0089-5`.

36. Susannah G Tringe, Tao Zhang, Xuguo Liu, Yiting Yu, Wah Heng Lee, Jennifer Yap, Fei Yao, Sim Tiow Suan, Seah Keng Ing, Matthew Haynes, Forest Rohwer, Chia Lin Wei, Patrick Tan, James Bristow, Edward M Rubin, and Yijun Ruan. The airborne metagenome in an indoor urban environment. *PloS one*, 3(4):e1862, January 2008. ISSN 1932-6203. DOI: `10.1371/journal.pone.0001862`.

37. Koji Ueno and Toyotaro Suzumura. Highly scalable graph search for the graph500 benchmark. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 149–160, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0805-2. DOI: `10.1145/2287076.2287104`.

38. Koji Ueno and Toyotaro Suzumura. Parallel distributed breadth first search on gpu. In *High Performance Computing (HiPC), 2013 20th International Conference on*, pages 314–323, Dec 2013. DOI: `10.1109/HiPC.2013.6799136`.

39. Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

40. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 265–278, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-577-2. DOI: `10.1145/1755913.1755940`.

DOI: 10.14529/jsfi140207

# Scalable parallel performance measurement and analysis tools – state-of-the-art and future challenges

*B. Mohr*[1]

Current large-scale HPC systems consist of complex configurations with a huge number of potentially heterogeneous components. As the systems get larger, their behavior becomes more and more dynamic and unpredictable because of hard- and software re-configurations due to fault recovery and power usage optimizations. Deep software hierarchies of large, complex system software and middleware components are required to operate such systems. Therefore, porting, adapting and tuning applications to today's complex systems is a complicated and time-consuming task. Sophisticated integrated performance measurement, analysis, and optimization capabilities are required to efficiently utilize such systems. This article will summarize the state-of-the-art of scalable and portable parallel performance tools and the challenges these tools are facing on future extreme-scale and big data systems.

*Keywords: parallel programming, performance tools, extreme scale computing.*

## Introduction

Current large-scale HPC systems consist of complex configurations with a huge number of components. Each node has multiple multi-core sockets and often one or more additional accelerator units in the form of many-core nodes (e.g., Intel Xeon Phi), GPGPUs or FPGAs, resulting in a heterogeneous system architecture. Caches, memory and storage are attached to the components on various levels and are shared between components in varying degrees. Typically, there is a (huge) imbalance between the computational power of the components and the amount of memory available to these components. Thus, deep software hierarchies of large, complex system software and middleware components are required to efficiently use such systems.

Therefore, porting, adapting and tuning applications to today's complex systems is a complicated and time-consuming task. Sophisticated integrated performance measurement, analysis, and optimization capabilities are required to efficiently utilize such systems. However, designing and implementing efficient and useful performance tools is extremely difficult because of the same reasons. While the system and middleware software layers are designed to be transparent, they are typically not transparent with respect to performance. This *performance intransparency* will result in escalation of unforeseen problems to higher layers, including the application. This is not really a new problem, but certain properties of current large-scale and future extreme-scale systems significantly increase its severity and significance:

- At extreme-scale scale, there always will be failing components in the system, which has a large impact on performance. A real-world application will probably never run on the exact same configuration twice.
- Load balancing issues limit the success even on moderately parallel systems, and the challenge of data locality will become another severe issue which has to be addressed by appropriate mechanisms and tools.
- Dynamic power management, e.g., at hardware level inside a CPU or accelerator, will result in performance variability between cores, nodes, and across different runs. The alternative to run at lower but system-wide consistent speed without dynamic power adjustments may not be an option in the future.

[1]Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Jülich, Germany

- The challenge of predicting application performance at extreme scale will make it difficult to detect a performance problem if it is escalated undetected to the application level.
- The ever growing higher integration of components into a single chip and the use of more and more hardware accelerators makes it more difficult to monitor application performance and move performance data out of the system unless special hardware support will be integrated into future systems.

Altogether this will require an integrated, collaborative, and holistic approach to handle performance issues and correctly detect and analyze performance problems. In the following, we will summarize the state-of-the-art of scalable and portable parallel performance tools. Next, we discuss the various methods and approaches regarding tool portability, scalability, and integration. The article closes with an outlook on the challenges tools will face on future extreme-scale and big data systems.

## 1. Overview of state-of-the-art parallel performance tools

Research on parallel performance tools has a long history. First tools appeared at the same time as the first parallel computer systems back in the 1980s and early 90s [1]. Meanwhile, many performance instrumentation, measurement, analysis and visualization tools exist, and summarizing and listing all major methods, approaches and tools is impossible in a short journal paper. Therefore, in the following section we concentrate on the major tool sets that are (i) *portable*, i.e., they can be used on more than one of today's dominating architectures: Cray, IBM BlueGene, Fujitsu K computer, and Linux/UNIX clusters, (ii) *scalable*, i.e., it was demonstrated that they can be successfully used for an application executing on a couple of thousand nodes, (iii) *versatile*, i.e., they allow the performance analysis of all levels of today's HPC systems: message passing between nodes, multi-threading and multi-tasking inside nodes, and offloading to accelerators, and (iv) *supported*, i.e., there are well-established groups or organizations behind them which maintain and further develop them.

The structure of the following subsections is modelled after and re-uses excerpts from the VI-HPS Tools Guide [2], which also includes descriptions of further tools.

### 1.1. TAU

TAU [3, 4] is a comprehensive profiling and tracing toolkit that supports performance evaluation of programs written in C++, C, UPC, Fortran, Python, and Java. It is a robust, flexible, portable, and integrated framework and toolset for performance instrumentation, measurement, analysis, and visualization of large-scale parallel computer systems and applications. TAU supports both direct measurement as well as sampling modes of instrumentation and interfaces with external packages such as PAPI [25] or tools like Score-P, Scalasca, and Vampir (all described in the following sections). TAU is available under a BSD-style license.

#### *Typical Workflow*

TAU allows the user to instrument the program in a variety of ways including rewriting the binary using *tau_rewrite* or runtime pre-loading of shared objects using *tau_exec*. Source-level instrumentation typically involves substituting a compiler in the build process with a TAU compiler wrapper. This wrapper uses a given TAU configuration to link in the TAU library. At runtime, a user may specify different TAU environment variables to control the measurement options chosen for the performance experiment. This allows the user to generate call-path profiles, specify hardware performance counters, turn on event-based sampling, generate traces, or specify memory instrumentation options.

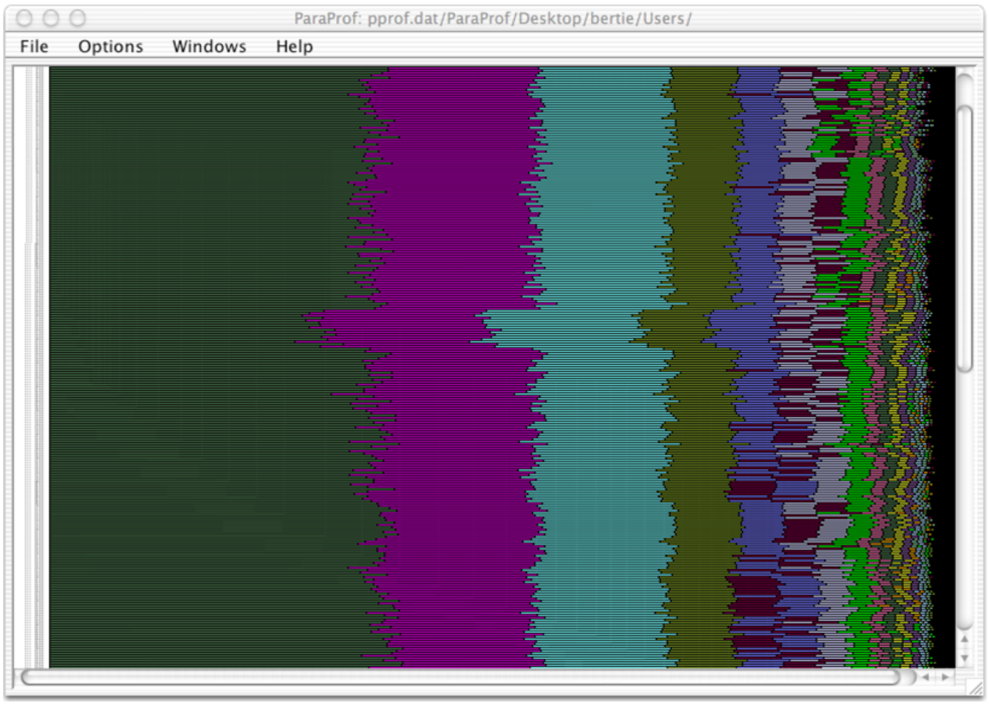


**Figure 1.** TAU main profile window. Each line shows a flattened histogram of a selected performance metric for a specific thread or task. The different colors represent the different program regions (e.g., functions, loops, parallel sections, etc.). The display allows to get a quick overview via comparing the differences/commonalities of the different threads/tasks. Clicking on a line label brings up more detailed information about the selected thread or task. Clicking on a colored region brings up more detailed information about the selected program region. The menu bar allows to choose the performance metric shown or to launch further, more detailed displays like a call-graph or communication matrix display. Results from larger performance experiments can be more easily analyzed with the 3D displays of TAU, see fig. 2.

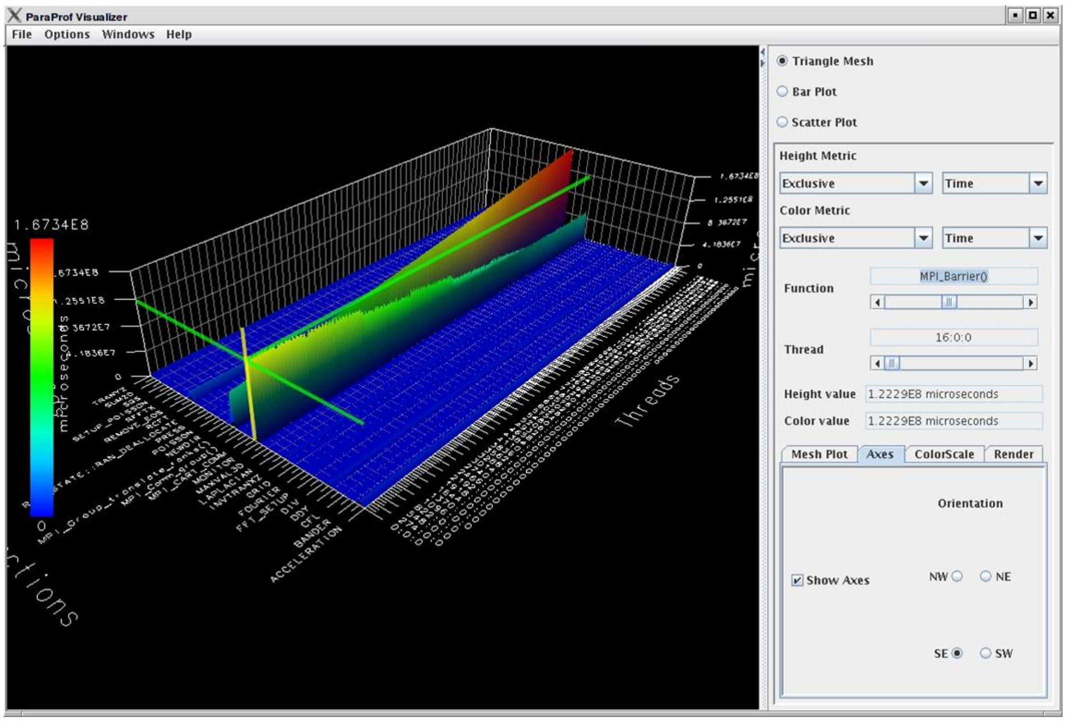
Performance-analysis results may be stored in TAUdb, a database for cross-experiment analysis and advanced performance data mining operations using TAU's PerfExplorer tool [5]. It may be visualized using ParaProf, TAU's profile browser that can show the extent of performance variation and compare executions, see fig. 1.

*Supported platforms*

IBM Blue Gene/P/Q, NVIDIA and AMD GPUs and Intel MIC systems, Cray XE/XK/XC30, SGI Altix, Fujitsu K Computer (FX10), NEC SX-9, Solaris & Linux clusters (x86/x86_64, MIPS, ARM), Windows, Apple Mac OS X.

*Supported Runtime Layers*

MPI, OpenMP (using GOMP, OMPT, and Opari instrumentation), Pthread, MPC Threads, Java Threads, Windows Threads, CUDA, OpenCL, OpenACC.



- [✓ Triangle Mesh] Bar Plot / Scatter Plot
- Height Metric: Exclusive, Time
- Color Metric: Exclusive, Time
- Function: MPI_Barrier()
- Thread: 16:0:0
- Height value: 1.2229E8 microseconds
- Color value: 1.2229E8 microseconds
- Show Axes ✓
- Orientation: NW / NE / [✓ SE] / SW

**Figure 2.** TAU 3D profile view. It displays the distribution of two selected performance metrics (encoded via height and color) over the axes program regions and threads/tasks.

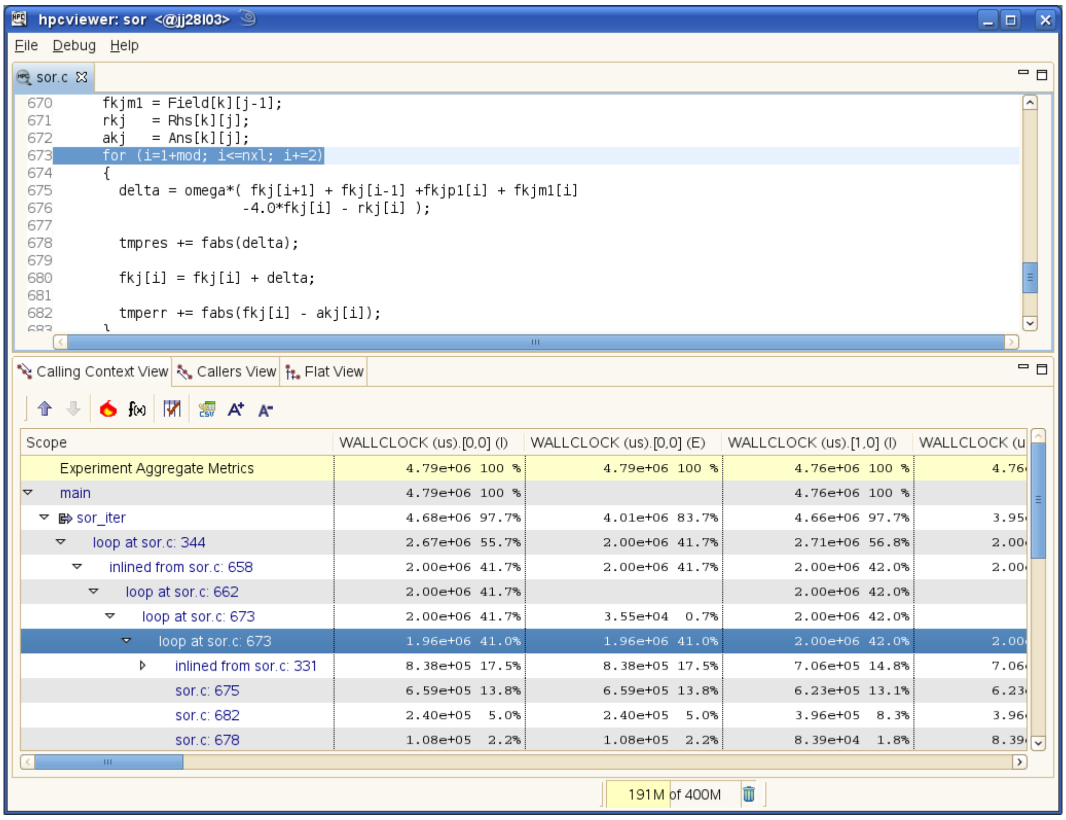
## 1.2. HPCToolkit

HPCToolkit [6, 7] is an integrated suite of tools for measurement and analysis of program performance on computers ranging from multi-core desktop systems to the largest supercomputers. HPCToolkit provides accurate measurements of a program's work, resource consumption, and inefficiency, correlates these metrics with the program's source code, works with multilingual, fully optimized binaries, and has very low measurement overhead. HPCToolkit's measurements provide support for analyzing a program's execution cost, inefficiency, and scaling characteristics both within and across nodes of a parallel system.

### *Typical Workflow*

HPCToolkit works by sampling an execution of a multi-threaded and/or multiprocess program using hardware performance counters, unwinding thread call stacks, and attributing the metric value associated with a sample event in a thread to the calling context of the thread/process in which the event occurred. Sampling has sometimes advantages over instrumentation for measuring program performance: it requires no modification of source code and it avoids potential blind spots (such as code available in only binary form). Sampling using performance counters enables fine-grained measurement and attribution of detailed costs including metrics such as operation counts, pipeline stalls, cache misses, and inter-cache communication in multi-core and multi-socket configurations. HPCToolkit also supports computing derived metrics such as cycles per instruction, waste, and relative efficiency to provide insight into a program's shortcomings. HPCToolkit is available under a BSD-style license.

HPCToolkit assembles performance measurements into a call-path profile that associates the costs of each function call with its full calling context. A unique capability of HPCToolkit is its nearly flawless ability to unwind a thread's call stack (which is often difficult and error-prone with highly optimized code). In addition, HPCToolkit uses binary analysis to attribute program performance metrics to full dynamic calling contexts augmented with information about call sites, source lines, loops and inlined code. Measurements can be analyzed in a variety of ways, see fig. 3: top-down in a calling context tree, which associates costs with the full calling context in which they are incurred; bottom-up in a view that apportions costs associated with a function to each of the contexts in which the function is called; and in a flat view that aggregates all costs associated with a function independent of calling context.



```
670      fkjm1 = Field[k][j-1];
671      rkj   = Rhs[k][j];
672      akj   = Ans[k][j];
673      for (i=1+mod; i<=nxl; i+=2)
674      {
675        delta = omega*( fkj[i+1] + fkj[i-1] +fkjp1[i] + fkjm1[i]
676                        -4.0*fkj[i] - rkj[i] );
677
678        tmpres += fabs(delta);
679
680        fkj[i] = fkj[i] + delta;
681
682        tmperr += fabs(fkj[i] - akj[i]);
```

| Scope | WALLCLOCK (us).[0,0] (I) | WALLCLOCK (us).[0,0] (E) | WALLCLOCK (us).[1,0] (I) | WALLCLOCK (u |
|---|---|---|---|---|
| Experiment Aggregate Metrics | 4.79e+06 100 % | 4.79e+06 100 % | 4.76e+06 100 % | 4.76 |
| main | 4.79e+06 100 % | | 4.76e+06 100 % | |
| sor_iter | 4.68e+06 97.7% | 4.01e+06 83.7% | 4.66e+06 97.7% | 3.95 |
| loop at sor.c: 344 | 2.67e+06 55.7% | 2.00e+06 41.7% | 2.71e+06 56.8% | 2.00 |
| inlined from sor.c: 658 | 2.00e+06 41.7% | 2.00e+06 41.7% | 2.00e+06 42.0% | 2.00 |
| loop at sor.c: 662 | 2.00e+06 41.7% | | 2.00e+06 42.0% | |
| loop at sor.c: 673 | 2.00e+06 41.7% | 3.55e+04 0.7% | 2.00e+06 42.0% | |
| loop at sor.c: 673 | 1.96e+06 41.0% | 1.96e+06 41.0% | 2.00e+06 42.0% | 2.00 |
| inlined from sor.c: 331 | 8.38e+05 17.5% | 8.38e+05 17.5% | 7.06e+05 14.8% | 7.06 |
| sor.c: 675 | 6.59e+05 13.8% | 6.59e+05 13.8% | 6.23e+05 13.1% | 6.23 |
| sor.c: 682 | 2.40e+05 5.0% | 2.40e+05 5.0% | 3.96e+05 8.3% | 3.96 |
| sor.c: 678 | 1.08e+05 2.2% | 1.08e+05 2.2% | 8.39e+04 1.8% | 8.39 |

**Figure 3.** HPCToolkit main window (hpcviewer) showing a top-down calling context analysis. Selecting a specific program region in the call tree at the bottom displays the corresponding source code in the upper part.

By working at the machine-code level, HPCToolkit accurately measures and attributes costs in executions of multilingual programs, even if they are linked with libraries available only in binary form. HPCToolkit supports performance analysis of fully optimized code; it even measures and attributes performance metrics to shared libraries that are dynamically loaded at runtime.

HPCToolkit also helps pinpointing scaling losses in parallel codes, both within multi-core nodes and across the nodes in a parallel system. Using differential analysis of call path profiles collected on different numbers of threads or processes enables one to quantify scalability losses and pinpoint their causes to individual lines of code executed in particular calling contexts.

*Supported platforms*

IBM Blue Gene/P/Q, Cray XE/XK/XC30, Linux clusters (x86/x86_64).

*Supported Runtime Layers*

HPCToolkit allows measuring and analyzing codes which span multiple processes (within and across nodes) and/or use multi-threading (e.g., OpenMP, Pthreads, etc.). As the tool collects data on the binary level, it is programming-model agnostic; the "translation" from binary objects (e.g., a runtime system function call) to programming model user-level constructs (e.g., OpenMP critical region directive) has to be done by the user.

## 1.3. Extrae/Paraver

Paraver [8–10] is a performance analyzer based on event traces with a great flexibility to explore the collected data, enabling detailed analysis of metrics variability and distribution with the objective of understanding the applications' behavior. Paraver has only two types of views, but a lot of flexibility to define and correlate them. Timelines provide the evolution with time (see fig. 4) and tables (histograms, profiles) a measurement of the metrics distribution.

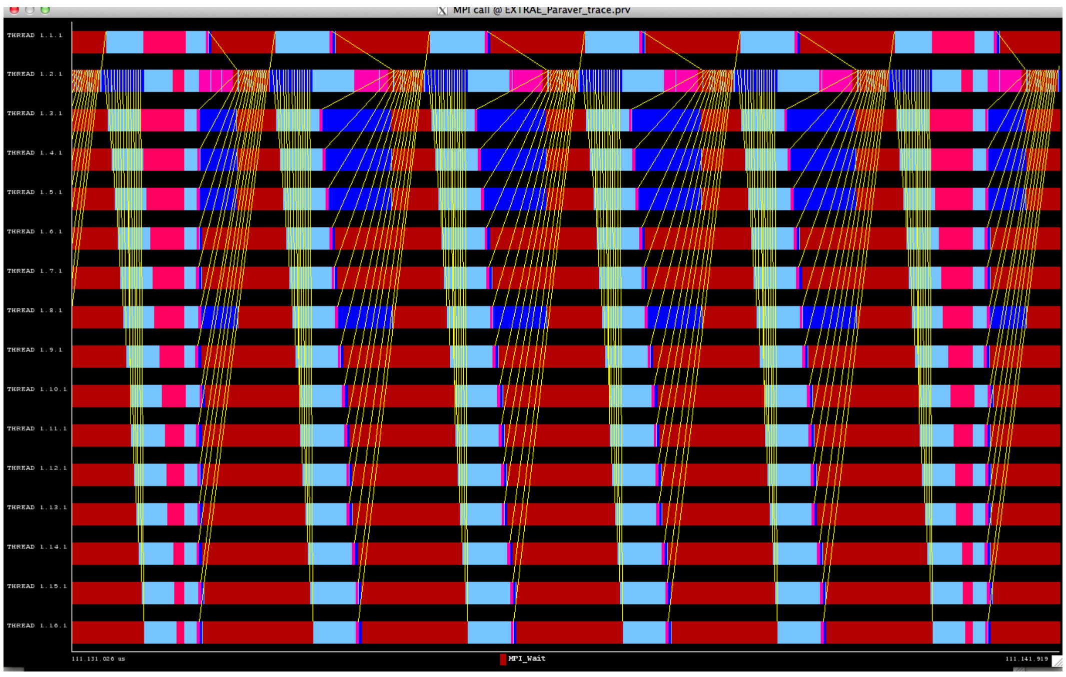


**Figure 4.** Paraver timeline window

To facilitate extracting insight from detailed performance data, during the last years new modules have been added implementing advanced performance analytics techniques. Clustering, tracking and folding allow the performance analyst to identify the program structure, study its evolution and look at the internal structure of the computation phases.

*Typical Workflow*

First, event traces are collected with the parallel program instrumentation and measurement package Extrae in a Paraver-specific format (PRV). It uses different mechanisms to insert measurement probes that vary from static interception of the runtime calls linking with the Extrae library to dynamic instrumentation using Dyninst [11]. The most frequent scenario is to use LD_PRELOAD to intercept runtime system calls of production binaries at loading time. The information collected by Extrae includes entry and exit to the programming model runtime, hardware counters (via PAPI), call stack references, user functions, periodic samples and user events. Extrae and Paraver are available under a LPGL license.

Once the necessary trace(s) are collected, the analyst drives the full process generating and validating his/her hypothesis about the application behavior within Paraver. The initial steps would be similar for all the analyses and are provided as a basic methodology tutorial, the results of these steps would allow to decide the path to follow. The tool provides an extensive list of pre-conceived configuration files. Paraver offers a very flexible way to combine multiple views, so as to generate new representations of the data and more complex derived metrics. Once a desired view is obtained, it can be stored in a configuration file to apply it again to the same trace or to a different one.

*Platform support*

Linux clusters (x86/x86_64, ARM, Power), IBM Blue Gene/P/Q, Fujitsu FX10, SGI Altix, Cray XT, Intel Xeon Phi, GPUs.

*Supported Runtime Layers*

MPI, OpenMP, OmpSs, Pthread, CUDA, OpenCL.

## 1.4. Vampir

The Vampir [12–14] event trace visualizer allows to study a program's runtime behavior at a fine level of detail. This includes the display of detailed performance event recordings over time in timelines and aggregated profiles. Interactive navigation and zooming are the key features of the tool, which help to quickly identify inefficient or faulty parts of a program.

*Typical Workflow*

Before using Vampir, an application program needs to be instrumented and executed with the community-developed parallel program instrumentation and measurement package Score-P (see section 1.6). Running the instrumented program produces a bundle of trace files in OTF2-format [16]. Earlier versions of Vampir were relying on an internal instrumentation and measurement package called VampirTrace which produced traces in OTF-Format [15]. Both VampirTrace and Score-P are available under a BSD-style license.

After a trace file has been loaded by Vampir, the Trace View window opens with a default set of charts as depicted in fig. 5. Vampir accepts traces in OTF, OTF2, and EPILOG [17] (a format used by earlier versions of the Scalasca tool, see next section). The Vampir visualizer is a commercial package.

*Platform support*

IBM Blue Gene/P/Q, AIX (x86_64, POWER6), Cray XE/XK/XC30, SGI UV/ICE/Altix, Linux clusters (x86/x86_64), Windows, Apple Mac OS X.

*Supported Runtime Layers*

MPI, OpenSHMEM, OpenMP, Pthread, CUDA, OpenCL, OpenACC.

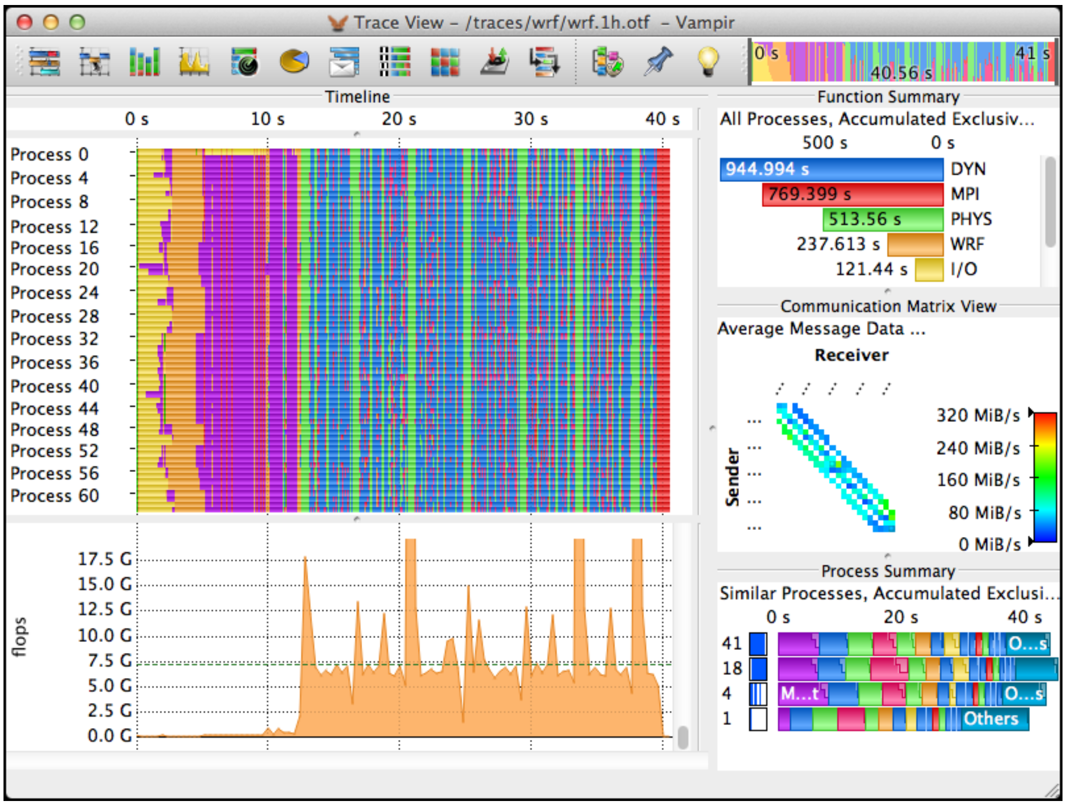
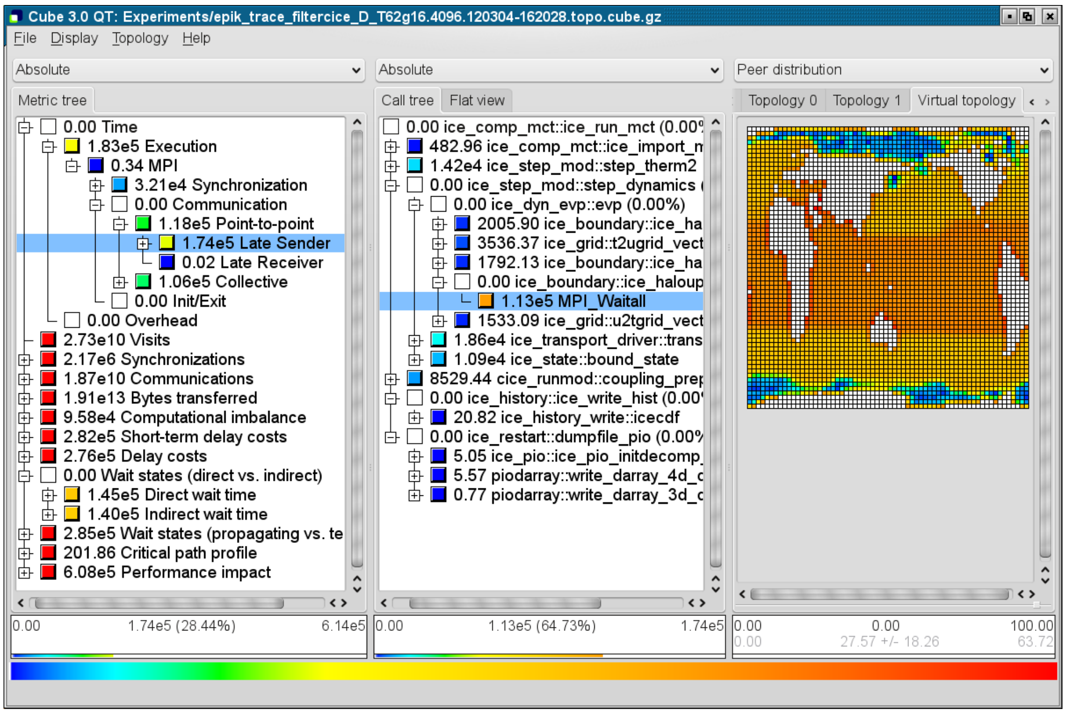


**Figure 5.** Vampir event trace browser. The charts shown can be divided into timeline charts and statistical charts. Timeline charts (left) show detailed event based information for arbitrary time intervals while statistical charts (right) reveal accumulated measures which were computed from the corresponding event data of the selected time interval in the timeline charts. Informational charts provide additional or explanatory information regarding timeline- and statistical charts. On top of the charts, two toolbars are available. The Charts Toolbar (top left) allows to add further charts to study I/O, inter-process communication and synchronization, and hardware performance metrics of the depicted program run. An overview of the phases of the entire program run is given in the Zoom Toolbar (top right), which can also be used to zoom and pan to the program phases of interest.

## 1.5. Scalasca

Scalasca [18, 19] supports the performance optimization of parallel programs by measuring and analysing their runtime behavior. The tool has been specifically designed for use on large-scale systems including IBM Blue Gene and Cray XE, but is also well suited for small- and medium-scale HPC platforms. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes. Scalasca is available under a BSD 3-Clause license.

The user of Scalasca can choose between two different analysis modes: (i) performance overview on the call-path level via profiling and (ii) the analysis of wait-state formation via event tracing. Wait states often occur in the wake of load imbalance and are serious obstacles to achieving satisfactory performance. The latest versions also includes a scalable critical path analysis [20] and root-cause analysis [21]. Performance-analysis results are presented to the user in an interactive explorer called Cube (fig. 6) that allows the investigation of the performance behavior on different levels of granularity along the dimensions metric, call path, and process.


Cube 3.0 QT: Experiments/epik_trace_filtercice_D_T62g16.4096.120304-162028.topo.cube.gz

File Display Topology Help

Absolute | Absolute | Peer distribution

Metric tree | Call tree, Flat view | Topology 0, Topology 1, Virtual topology

```
Metric tree
0.00 Time
  1.83e5 Execution
    0.34 MPI
      3.21e4 Synchronization
      0.00 Communication
        1.18e5 Point-to-point
          1.74e5 Late Sender
          0.02 Late Receiver
        1.06e5 Collective
      0.00 Init/Exit
  0.00 Overhead
2.73e10 Visits
2.17e6 Synchronizations
1.87e10 Communications
1.91e13 Bytes transferred
9.58e4 Computational imbalance
2.82e5 Short-term delay costs
2.76e5 Delay costs
0.00 Wait states (direct vs. indirect)
  1.45e5 Direct wait time
  1.40e5 Indirect wait time
2.85e5 Wait states (propagating vs. te
201.86 Critical path profile
6.08e5 Performance impact

0.00        1.74e5 (28.44%)        6.14e5
```

```
Call tree
0.00 ice_comp_mct::ice_run_mct (0.00%
482.96 ice_comp_mct::ice_import_m
1.42e4 ice_step_mod::step_therm2
0.00 ice_step_mod::step_dynamics (
  0.00 ice_dyn_evp::evp (0.00%)
    2005.90 ice_boundary::ice_ha
    3536.37 ice_grid::t2ugrid_vect
    1792.13 ice_boundary::ice_ha
    0.00 ice_boundary::ice_haloup
      1.13e5 MPI_Waitall
    1533.09 ice_grid::u2tgrid_vect
  1.86e4 ice_transport_driver::trans
  1.09e4 ice_state::bound_state
8529.44 cice_runmod::coupling_prep
0.00 ice_history::ice_write_hist (0.00
  20.82 ice_history_write::icecdf
0.00 ice_restart::dumpfile_pio (0.00%
  5.05 ice_pio::ice_pio_initdecomp_
  5.57 piodarray::write_darray_4d_c
  0.77 piodarray::write_darray_3d_c

0.00        1.13e5 (64.73%)        1.74e5
```

```
0.00        0.00            100.00
0.00    27.57 +/- 18.26      63.72
```

**Figure 6.** Scalasca's result explorer Cube. It allows interactive exploration of performance behavior along the dimensions performance metric (left), call tree (middle), and process topology (right). Selecting a metric displays the distribution of the corresponding value over the call tree. Selecting a call path again shows the distribution of the associated value over the machine or application topology. Expanding and collapsing metric or call tree nodes allows to investigate the performance at varying levels of detail.

### *Typical Workflow*

Before any Scalasca analysis can be carried out, the target application needs to be instrumented. For this task, Scalasca leverages now the community-driven instrumentation and measurement infrastructure Score-P (see section 1.6). Earlier versions of Scalasca (version 1) used an internal instrumentation and measurement package called EPIK which provided the same functionality. First, a call-path profile is collected and analyzed. After an optimized measurement configuration has been prepared based on initial profiles, a targeted event trace in EPILOG format (Scalasca version 1) or in OTF2 format (Scalasca version 2 or later) can be generated, and subsequently analyzed by Scalasca's automatic event trace analyzer after measurement is complete. This scalable analysis searches for inefficiency patterns and wait states, collects statistics about the detected instances, and identifies the critical path of the application. The analysis result can the be examined using the interactive performance report explorer Cube.

### *Platform support*

IBM Blue Gene/P/Q, Cray XT/XE/XK/XC, SGI Altix (incl. ICE + UV), Fujitsu FX-10 & K Computer, Tianhe-1A, IBM AIX clusters, Solaris & Linux clusters (x86/x86_64, ARM)

### *Supported Runtime Layers*

MPI, OpenSHMEM, OpenMP, OmpSs, HMPP, Pthread, CUDA.

## 1.6. Score-P

The Score-P [22, 23] instrumentation and measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis. It supports a wide range of HPC platforms and programming models. Score-P provides core measurement services for a range of specialized analysis tools, such as Vampir, Scalasca, TAU, and Periscope [24]. Further insights can be gained by employing additional tools on Score-P measurements. Score-P is available under a BSD 3-Clause license.

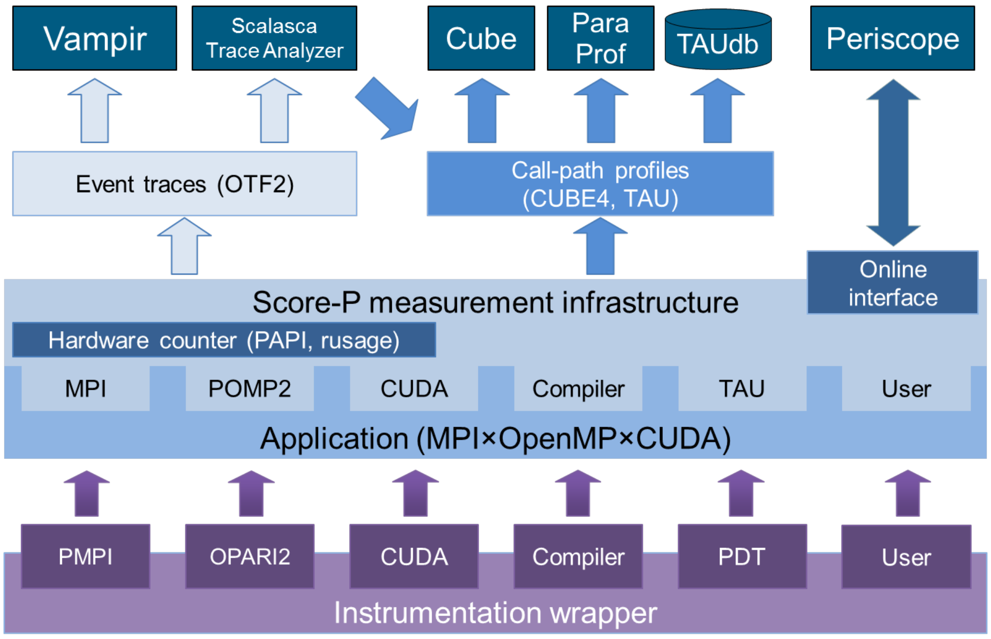


**Figure 7.** Score-P architecture

### *Typical Workflow*

The overall Score-P architecture is shown in fig. 7. To create measurements, the target program must be instrumented. Score-P offers various instrumentation options. User functions can be instrumented automatically with compiler support or source-to-source instrumentation (PDT) or manually. MPI functions are monitored via library interposition (PMPI). OpenMP constructs are instrumented with the source-to-source instrumenter OPARI2.

For measurement, the instrumented program can be configured to record an event trace in OTF2 format or produce a call-path profile in CUBE4 format. Optionally, PAPI hardware counters [25] can be recorded. Filtering techniques allow precise control over the amount of data to be collected. Call-path profiles can be examined in TAU (see section 1.1) or the Cube profile browser (see section 1.5). Event traces can be examined in Vampir (see section 1.4) or used for automatic bottleneck analysis with Scalasca (see section 1.5). Alternatively, on-line analysis with Periscope is possible.

### *Platform support*

IBM Blue Gene/P/Q, Cray XT/XE/XK, SGI Altix, K Computer, Fujitsu FX10, NEC SX-9, Solaris & Linux clusters (x86/x86_64, ARM)

*Supported Runtime Layers*

MPI, OpenSHMEM, OpenMP, OmpSs, HMPP, Pthread, CUDA.

## 2. Challenges for today's parallel performance tools

In this section, we discuss the basic challenges parallel performance tools face on today's large-scale systems: portability, scalability, and integration between tool components.

### 2.1. Portability

Although there was some consolidation in regard to operating systems for HPC clusters and high-end systems in the past decade – most of them are running some variant of Linux or at least Linux-compatible micro-kernels now – providing performance tools for these systems did not became much easier due to the openness and the lack of standardization of the Linux operating system. Basically no Linux cluster is exactly alike another one and each system provides a different set of compilers (e.g., GNU, Intel, PGI, IBM XL, Clang, SUN, Pathscale and others) and a different set of MPI libraries (e.g., MPICH, OpenMPI, Intel, LAM, HP, Scali, BullXMPI, SUN, IBM POE, Platform and more) in different versions. Depending on the compiler version, a different version of the OpenMP standard needs to be handled and supported by the tools; the same is true for the version of the MPI standard supported by the MPI library.

### 2.2. Scalability

The very large number of nodes and cores available to applications in future extreme-scale systems will be of course a severe challenge to tools, but the size of today's large-scale systems poses already problems now. In the November 2013 list of the TOP500 supercomputers, only four systems have less than 4,096 processor cores and the average is at almost 41,500 cores, which is an increase of 10,000 in just one year. Even the median system size is already over 17,400 cores. So tools must be able to handle at least tens of thousands cores and threads; ideally of course much more. Top 5 systems have multiple millions of cores. Various methods and approaches are employed by current performance tools to be able to handle the measurement and analysis of applications executing on large numbers of cores:

**Scalable data collection and reduction:** All tools listed in section 1 collect performance data in parallel either per process or even per thread and then use MPI to combine and merge the data either at the end of the execution or in a separate phase after the program measurement. Various I/O virtualization mechanisms (e.g., SIONlib [26]) are used to efficiently store process- and thread-local data in files. Scalasca's EPIK measurement system and Score-P no longer merge local traces into a global trace but use parallel I/O to access the separate traces concurrently and use MPI at the end of the program measurement to perform an efficient unification of measurement object identifiers and timestamp synchronization and correction. Another approach is used by Paraver: Automatic clustering, tracking and folding of trace events allows the performance analyst to identify the program structure, study its evolution and then only look at instances of the most important computation phases one at a time.

**Scalable parallel data analysis:** It is clear that the huge amount of performance data collected with a large-scale measurement can only be analyzed efficiently if the performance analysis components are parallelized themself. The Vampir toolset employs a distributed client/server model where the trace data is read and analyzed by a multi-node, multi-threaded server running on the target HPC system connected to a simple visualization client running on the workstation or laptop of the performance analyst. This way, the huge traces never have to be copied off the HPC system. The interactivity of the analysis process can be controlled by using more or less processes and threads for the parallel server process. The Scalasca trace analyzer also features a parallel and highly efficient performance bottleneck pattern search [18] and delay [21] and critical path [20] analysis. The algorithm is designed in a way that it uses the same number of MPI ranks and threads as the application under investigation, so the measurement and the subsequent trace analysis can be run in the same batch job. Time-consuming analysis modules from the Paraver analyzer and visualizer also have been parallelized with OmpSs.

**Scalable visualizations:** However, with an efficient parallel performance data collection and analysis, the bottleneck in the performance analysis process was just shifted to the result presentation phase. TAU uses 3D-displays to effectively display large amounts of data (see fig. 2) and Scalasca's report explorer Cube utilizes 2D and 3D displays to show performance data mapped on system hardware or application topologies (see fig. 6). Cube also uses hierarchical tree displays to allow browsing of performance data on various levels of detail. More research and new methods (e.g., visual data analytics) are needed here.

## 2.3. Integration

It is clear that only an integrated tool (environment) which can handle more than one level of parallelization – including the communication and synchronization aspects on the inter-node level, the multi-threading and multi-tasking issues at the intra-node level, and the data transfers, scheduling, and kernel invocations for attached accelerators – is a prerequisite for an efficient performance analysis process. This not only allows to investigate all performance relevant aspects of a program execution within one environment but also the influences and dependencies between the different parallelization levels. All tools listed in section 1 fall into this category. A big obstacle here is the current "zoo" of programming models available for multi-threading and off-loading; it is very hard for tools to support all of them and equally well.

However, it is just as important that performance measurement tools are integrated in some way with performance prediction and modeling tools. This way performance models can be automatically generated and/or calibrated with performance measurements and then can be much more successfully used to predict the application performance for other systems or configurations. A good example here is the Dimemas simulator which reconstructs the time behavior of a parallel application using a Paraver event trace that captures the time resource demands (CPU and network) of a parallel application as input. The target machine is modelled by a reduced set of key factors influencing the performance that model linear components like the point-to-point transfer time as well as non-linear factors like resources contention or synchronization. Using this simple model, Dimemas allows to simulate parametric studies in a very short time frame. Dimemas can generate a modified Paraver trace file as part of its output, enabling the user to conveniently examine the simulated run and understand the behavior. Another approach is used in the DFG SPPEXA Catwalk project, where a series of profile measurements collected with Score-P, representing a scaling or parameter study, is input to a special model-generation component which automatically approximates the scaling behavior for selected metrics for all kernels of the analyzed program [30]. The generated performance model can then be used to easily locate program parts with a bad scaling behavior, e.g., program parts that will become bottlenecks on future systems with either higher core counts or less memory capacity.

Finally, even integrated and powerful tool environments cannot fulfill all requirements. Different tools have different strength and weaknesses and provide different views on the same performance behavior. Ideally, profile and trace data can easily be imported and exported by the various tools via open interfaces to make it easy to compare the results between tools and to move from one tool to another if necessary. In the first decade of the current millennium, the international tools community worked hard to achieve this goal, see fig. 8 which actually only shows the result for a selected portion of the tool landscape.

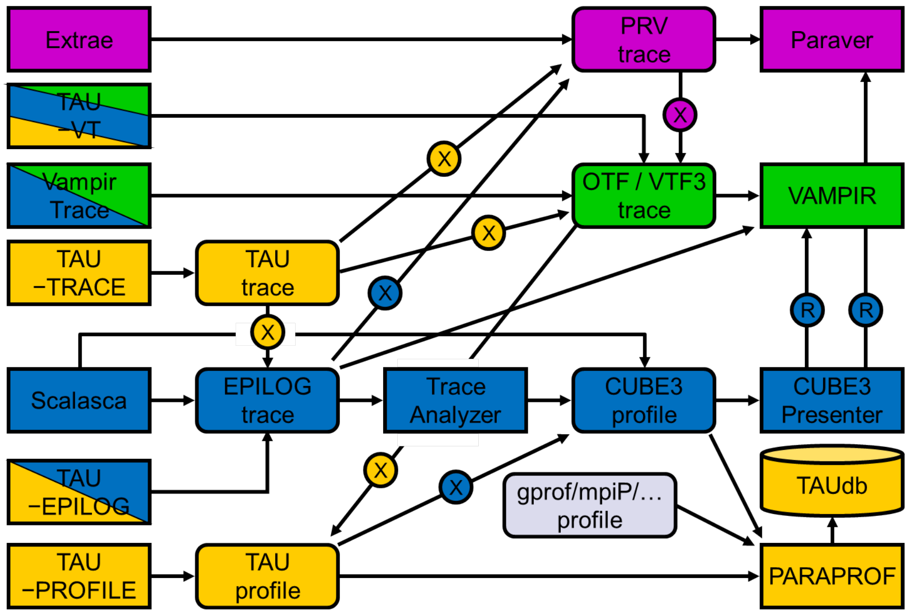


**Figure 8.** Tool integration efforts. The diagram shows the basic performance analysis workflows of the different tools around the year 2011. Rectangles represent tool components, boxes with round corners profile or trace data files. Lines labeled with an "X" in a circle indicate a conversion tool between the connected file formats. Lines labeled with an "R" in a circle describe Cube's ability to remotely control Vampir and Paraver [27].

.

Although kind of impressive, it quickly became clear that this complex situation is more confusing than helping for the typical tool user. This lead in the end to the community-developed performance instrumentation and measurement package Score-P which replaced the internal packages from Vampir, TAU, and Scalasca (see section 1.6), simplifying the resulting performance analysis workflow considerably. The use of common file formats like CUBE4 and OTF2 shared between a wide a range of tools allowed the definition of a well-defined and sophisticated performance analysis workflow in the EU-Russia research project HOPSA (see fig. 9). Beyond the tools described in this article, it also includes the commercial memory and threading analysis tool ThreadSpotter [28] and the system monitoring framework LAPTA [29]. For more details, especially more information about the implemented tool interactions, see [27].

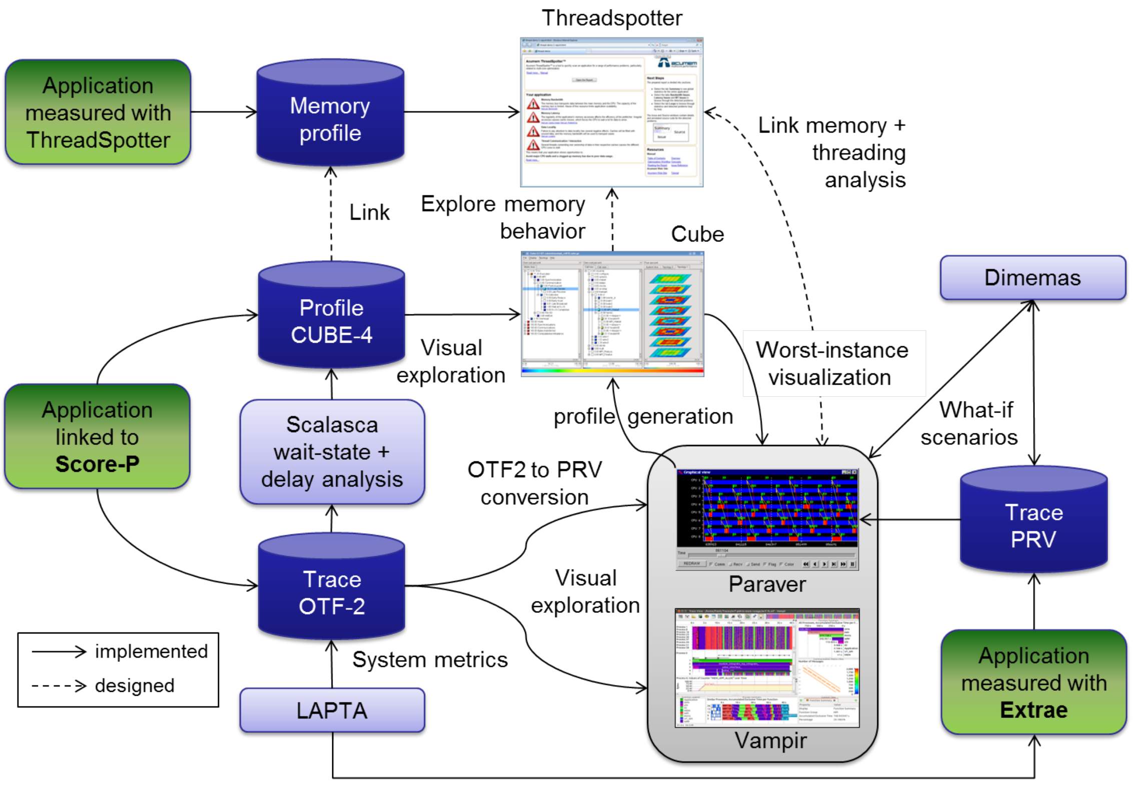


**Figure 9.** HOPSA workflow

## 3. Challenges for parallel performance tools for future extreme-scale and big data systems

A large number of approaches for performance analysis exist that have successfully applied at small and medium scale. The large amount of performance data may seem to impede the use at extreme scale. However, this is not the case as long as features like memory size and I/O capabilities scale with compute power. An instrumented application is nothing but an application with modified demands on the system executing it. This makes current approaches for performance analysis still feasible in the future as long as all involved software components are parallel and scalable. In addition to increased scalability, techniques like automatic analysis, advanced filtering, on-line monitoring, clustering, and analysis as well as data mining will be of increased importance. A combination of various techniques will have to be applied. The following considerations are key for a successful approach to performance analysis at extreme scale:

- Failover or operation with failed components in general should be performance neutral.
- An extreme scale system has to be capable to monitor the performance of components, not just the functionality.
- Hardware and software components need to provide sufficient performance details for the analysis if a performance problem unexpectedly escalates to higher levels.
- Metrics beyond FLOPs need to be developed to identify and quantify bottlenecks, to measure the sustained performance and the gap to the attainable peak performance.
- Programming models should be designed with performance analysis in mind. Part of that could be a (standardized) hidden control mechanism in the runtime system that will be

able to dynamically control – in time and space – the generation of performance data if requested.
- Performance analysis in the presence of noise requires inclusion of appropriate statistical descriptions.
- Performance analysis needs to incorporate techniques from the areas of signal processing, data mining, and visual data analytics.

***Acknowledgements.*** *The author would like to use this opportunity to thank the HPC performance tools community in general and the Scalasca, Vampir, TAU, and Paraver teams in particular for many fruitful discussions, insights, and sharing their experiences and tools.*

## References

1. R. Klar and N. Luttenberger: VLSI-based Monitoring of the Inter-Process-Communication of Multi-Microcomputer Systems with Shared Memory. Proceedings EUROMICRO '86, Microprocessing and Microprogramming, vol. 18, no. 15, 195–204, Venice, Italy, 1986.
2. Virtual Institute - High Productivity Supercomputing (VI-HPS): VI-HPS Tools Guide. Available at `http://www.vi-hps.org/tools/`.
3. S. Shende and A. D. Malony: The TAU Parallel Performance System. Intl. Journal of High Performance Computing Applications, 20(2):287–331, 2006. SAGE Publications. DOI: 10.1177/1094342006064482
4. TAU homepage. University of Oregon. `http://tau.uoregon.edu`
5. K. A. Huck and A. D. Malony: 2005. PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing. Proceedings ACM/IEEE conference on Supercomputing (SC '05). IEEE, Washington, DC, USA, 2005.
6. L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent: HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience, 22(6):685–701, 2010.
7. HPCToolkit homepage. Rice University. `http://hpctoolkit.org`
8. J. Labarta, S. Girona, V. Pillet, T. Cortes, L. Gregoris, DiP: A parallel program development environment. Proceedings 2nd Intl. Euro-Par Conference, Lyon, France, Springer, 1996.
9. H. Servat Gelabert, G. Llort Sanchez, J. Gimenez, and J. Labarta: Detailed performance analysis using coarse grain sampling. Proceedings Euro-Par 2009 - Parallel Processing Workshops, Delft, The Netherlands, August 2009, 185–198. Springer, 2010.
10. Paraver homepage. Barcelona Supercomputing Center. `http://www.bsc.es/paraver`
11. Dyninst homepage. University of Wisconsin - Madison. `http://www.dyninst.org/`
12. M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, W. E. Nagel: Developing Scalable Applications with Vampir, VampirServer and VampirTrace. Proceedings of ParCo 2007, Jülich, Germany, 637–644, IOS Press, 2007.
13. A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, W. E. Nagel: The Vampir Performance Analysis Tool-Set. Proceedings Parallel Tools Workshop 2008, 139–155, 2008.
14. Vampir homepage. Technical University Dresden. `http://www.vampir.eu`

15. A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. E. Nagel: Introducing the Open Trace Format (OTF), Proceedings Computational Science - ICCS 2006: 6th Intl. Conference, Reading, UK, Springer, 526–533, 2006.

16. D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, F. Wolf: Open Trace Format 2 - The Next Generation of Scalable Trace Formats and Support Libraries. Proceedings of ParCo 2011, Ghent, Belgium, 481–490, IOS Press, 2012.

17. F. Wolf, B. Mohr: EPILOG Binary Trace-Data Format. Technical Report FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich, 2004.

18. M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, B. Mohr: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 22(6):702–719, 2010.

19. Scalasca homepage. Jülich Supercomputing Centre and German Research School for Simulation Sciences. `http://www.scalasca.org`

20. D. Böhme, B. R. de Supinski, M. Geimer, M. Schulz, F. Wolf: Scalable Critical-Path Based Performance Analysis. Proceedings IEEE Intl. Parallel & Distributed Processing Symposium (IPDPS), Shanghai, China, 1330–1340, IEEE, 2012.

21. D. Böhme, M. Geimer, F. Wolf, L. Arnold: Identifying the root causes of wait states in large-scale parallel applications. Proceedings Intl. Conference on Parallel Processing (ICPP), San Diego, CA, USA, 90–100, IEEE, 2010.

22. D. an Mey, S. Biersdorff, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A.D. Malony, W.E. Nagel, Y. Oleynik, C. Rössel, P. Saviankou, D. Schmidl, S.S. Shende, M. Wagner, B. Wesarg, F. Wolf: Score-P: A Unified Performance Measurement System for Petascale Applications. In: Competence in High Performance Computing 2010 (CiHPC), 85–97. Springer, 2012.

23. Score-P homepage. Score-P Consortium. `http://www.score-p.org`

24. M. Gerndt and M. Ott: Automatic Performance Analysis with Periscope. Concurrency and Computation: Practice and Experience, 22(6):736–748, 2010.

25. PAPI homepage. University of Tennessee - Knoxville. `http://icl.cs.utk.edu/papi/`

26. W. Frings, F. Wolf, V. Petko:. Scalable massively parallel I/O to task-local files. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09). ACM, New York, NY, USA, Article 17, 2009.

27. B. Mohr, V. Voevodin, J. Giménez, E. Hagersten, A. Knüpfer, D. A. Nikitenko, M. Nilsson, H. Servat, A. Shah, F. Winkler, F. Wolf, and I. Zhukov: The HOPSA Workflow and Tools. Proceedings 6th Intl. Parallel Tools Workshop, Stuttgart, September 2012.

28. E. Berg, E. Hagersten: StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis. Proceedings IEEE Intl. Symposium on Performance Analysis of Systems and Software (ISPASS-2004), Austin, Texas, USA, 2004.

29. A.V. Adinets, P.A. Bryzgalov, Vad.V. Voevodin, S.A. Zhumatiy, D.A. Nikitenko: About one approach to monitoring, analysis and visualization of jobs on cluster system (In Russian). Numerical Methods and Programming, vol. 12, 90–93, 2011.

30. A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. Proceedings ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA, 1–12, ACM, 2013.