Supercomputing Frontiers and Innovations

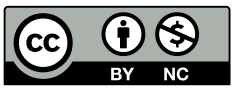
2018, Vol. 5, No. 1

Scope

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

Editorial Board

Contents



DOI: 10.14529/jsfi180101

AlgoWiki Project as an Extension of the Top500 Methodology

Alexander Antonov[1], *Jack Dongarra*[1,2], *Vladimir Voevodin*[1]

The AlgoWiki project is dedicated to describing the parallel structure and key features of various algorithms. The descriptions are intended to provide complete information about algorithms' properties, which are needed to adequately assess their implementation efficiency for any computing platform. This work sets out the key areas for further development of the project which were recently developed based on working with the AlgoWiki encyclopedia. We are suggesting an approach to extend the Top500 methodology, which is commonly used to compare various computing platforms.

Keywords: AlgoWiki, parallel structure, algorithm's properties, Top500 methodology, problems, methods, algorithms, implementations, computing platforms.

Introduction

The computing world is constantly changing, and there are numerous reasons for this. New problems appear regularly that require increasingly powerful computing platforms. New ideas appear that are reflected in the computer architecture and help to improve performance or have a positive impact on energy consumption and cost. Technological progress results in developing and utilizing new computing environments which, unlike classical computers, can be highly heterogeneous and distributed. Ultimately these changes result in the need to carefully review algorithm structure and properties in order to answer the main question: can a problem be solved within a certain predefined level of efficiency (e.g., within a reasonable time), and if so, how can this be done? If the answer is obviously positive, all other questions are irrelevant; otherwise a solution needs to be found. The main question is: does the algorithm have properties that match well with specific features of the computing system's infrastructure? Is it possible to find the minimum spanning tree of a graph with 2^{28} nodes on a vector computer? Is it possible to effectively solve large sparse linear equation systems in distributed computing environments? These questions cannot be answered without understanding the algorithm's properties.

Today, descriptions of various algorithms can be found in numerous books, systems, online resources and other sources [1–6]. Some focus on a mathematical formulation, others show a possible software implementation or study serial complexity. However, the main feature of modern computing platforms is a high degree of parallelism and special memory structure. This is what should be considered first of all if we wish to talk about efficiently implementing algorithms on computing systems at any level, from mobile devices to supercomputers.

1. About the AlgoWiki Project

The AlgoWiki project [7–9] is dedicated to describing the structure and key features of various algorithms. All descriptions follow the same structure (`http://algowiki-project.org/en/Description_of_algorithm_properties_and_structure`), which allows for easy comparison of various algorithms. The descriptions are intended to provide complete information about algorithm's properties, which is needed to ensure their efficient implementation on any computing platform. Each algorithm description in AlgoWiki is divided into two parts. The first part

[1] Lomonosov Moscow State University, Moscow, Russian Federation
[2] University of Tennessee, Knoxville, USA

of the description contains information that does not depend on the software implementation or computing platforms used. This is the theoretical potential of the algorithm determined by mathematics which we can rely on for implementations on any computing platform. The second part of the algorithm's description is oriented towards practical application, as it considers the interconnection between the algorithm's properties, specific parallel programming technologies and various classes of computing systems.

The project has been implemented using Wiki technologies, similar to the Wikipedia project: existing algorithm descriptions are available to everyone, and at the same time all experts can contribute their knowledge to AlgoWiki by adding descriptions of new algorithms or by making the information more exact for the existing ones.

2. The AlgoWiki Project and Top500 Methodology

The project currently presents a multitude of algorithms from various areas: linear algebra, graph algorithms, sorting algorithms, quantum system modeling algorithms, etc. The project is continually growing, covering more and more new areas including descriptions of new algorithms. Whenever a scientist works on an algorithm, he or she creates a new article in AlgoWiki, which contains a description of the algorithm's theoretical potential and particular features regarding its implementation on various computers.

In this regard, AlgoWiki offers a good basis for the natural extension of the Top500 methodology used to compare computing systems today. Linpack [10] was historically the first widely adopted approach which led to the creation of the list of the most powerful supercomputers (`http://top500.org/`). However, Linpack only reflects one aspect of computing platforms. That is why other tests were suggested later which became the basis for the Graph500 [11] and HPCG [12] benchmarks. All three ratings use the same technique: a basic algorithm is chosen and its software implementation is written and executed on each computing system in question, which results in a number that is used to judge the computer's properties. The number helps easily compare different computers to one another, including a compilation of the ratings described above.

The AlgoWiki project can be used to extend on this methodology. In fact, AlgoWiki offers a multitude of descriptions for very diverse algorithms, for which we have execution data on multiple computing platforms. The underlying algorithms for Linpack, Graph500 and HPCG, among others, are represented in AlgoWiki and correspond to three points out of the total multitude of algorithms in the project. Giving the computing community an opportunity to save the execution results for any algorithm, we can substantially extend the possibilities for comparing computing platforms. Using the AlgoWiki potential, we can move from three points (corresponding to Linpack, Graph500 and HPCG) to an analysis based on dozens, if not hundreds of various algorithms. We do not have to select every AlgoWiki algorithm for a detailed analysis and comparison; instead we can focus on the most interesting ones. If the corresponding algorithm is missing from AlgoWiki, it can be added, establishing the first step for formulating the respective new rating.

This extension of the Top500 methodology within AlgoWiki encyclopedia has several important implications. We can not only compare the results of various algorithms on different computers, but also analyze and understand the reasons behind these results: detailed descriptions of all algorithms are always at hand in AlgoWiki. It is also important that AlgoWiki can be used to store more than just the results obtained for record-setting computer configurations and

very large sets of input data: lesser values can be of substantial practical interest and are also available for analysis. In this respect, ratings like the Top500 following any AlgoWiki algorithm are just the tip of the iceberg representing the entire multitude of data stored in AlgoWiki for each specific algorithm.

3. Problems, Methods, Algorithms, Implementations, Computing Platforms

In practice, the possibilities enabled by AlgoWiki for analyzing the Algorithm–Computer combinations are much wider. The classification of algorithms in AlgoWiki is structured to support the clear identification of three levels: problem, method, algorithm (`http://algowiki-project.org/en/Algorithm_classification`). These three levels are marked with special icons in the classification, but in reality AlgoWiki has five levels:

Problem → Method → Algorithm → Implementation → Computing platform.

When data on an algorithm's execution on a particular computing system are submitted to AlgoWiki, information about the entire chain is stored, from Problem to Computing Platform. This gives extra freedom to perform comparisons and analyses. In particular, a researcher armed with the data structure in this manner can query the AlgoWiki database to make the following comparisons:

- computer performance achieved using different methods to solve the same problem;
- computer performance achieved using different methods to solve the same problem with fixed data size;
- the time to solve a problem of fixed size using different methods to address the problem;
- computer performance achieved using different implementations of the same method to address the same problem;
- the time to solve a problem of fixed size using different methods to address it in clusters containing, for example, up to 128 nodes;
- methods that demonstrate the maximum/minimum/predefined efficiency for a given class of computers; and many others.

In addition to analysis based on parameters like time, performance and efficiency, it is also possible to conduct a different kind of qualitative analysis within AlgoWiki, in particular to find:

- algorithms used to solve a given problem;
- problems a given algorithm is used to solve;
- algorithms that are used to solve a given problem with serial complexity below $O(n^2)$;
- all method-computer pairs used to solve a given problem, where the method used has serial complexity below $O(n^2)$ and parallel complexity below $O(n)$, while the implementation efficiency on a computer exceeds 40%.

For any elements of the chain indicated above: Problem, Method, Algorithm, Implementation and Computing Platform, the relevant values can be set as constants, while the others can be variable; this allows new ratings to be built or to find combinations that best suit required conditions. For example, the data in Tab. 1 show the performance of various computers (in MTEPS) when solving the "Strongly Connected Components Search" problem for two graph sizes with 2^{18} and 2^{20} nodes. Each line in the AlgoWiki table additionally contains a detailed description of the computer environment from which this data were obtained: a link to the implementation (executable file or source code), the number of compute nodes and cores, the

Table 1. Strongly Connected Components: performance of various implementations on different computers for two types of graphs (RMAT and SSCA–2) and number of nodes equals to 2^{18} and 2^{20}

Implementation	Computing Platform	MTEPS	GraphType	GraphSize
Ligra	Lomonosov-2 (x86)	830.0	RMAT	2^{20}
RCC for GPU	Lomonosov-2 (NVIDIA P100)	634.0	RMAT	2^{20}
GAP	Lomonosov-2 (x86)	547.0	RMAT	2^{20}
RCC for PU	Lomonosov-2 (x86)	418.0	RMAT	2^{20}
PBGL MPI	IBM BlueGene/P	232.9	RMAT	2^{20}
RCC for GPU	Lomonosov-2 (NVIDIA K40)	195.0	RMAT	2^{18}
RCC for PU	IBM Regatta	53.6	SSCA–2	2^{18}
PBGL MPI	IBM BlueGene/P	45.7	RMAT	2^{20}
RCC for PU	Lomonosov (x86)	41.0	RMAT	2^{20}
RCC for PU	IBM Regatta	36.9	RMAT	2^{18}
RCC for PU	Lomonosov (x86)	32.5	RMAT	2^{20}
PBGL MPI	IBM BlueGene/P	13.1	SSCA–2	2^{18}
RCC for PU	Lomonosov (x86)	10.1	SSCA–2	2^{20}
RCC for PU	Lomonosov (x86)	8.3	SSCA–2	2^{20}
PBGL MPI	Lomonosov NVIDIA 2090)	2.3	SSCA–2	2^{18}
PBGL MPI	IBM BlueGene/P	0.2	RMAT	2^{20}
PBGL MPI	IBM BlueGene/P	0.1	SSCA–2	2^{18}

Table 2. Source Shortest Paths: performance of different implementations of different algorithms on various computers for graphs with number of nodes equals to 2^{20} and 2^{21}

Method	Implementation	Computing Platform	MTEPS	GraphSize
Bellman–Ford	RCC for GPU	Lomonosov	1309.0	2^{20}
Bellman–Ford	Ligra	Lomonosov–2	1035.0	2^{21}
Delta–Stepping	PBGL MPI	Cluster “Angara”	809.5	2^{21}
Delta–Stepping	GAP	Lomonosov–2	616.0	2^{21}
Delta–Stepping	GAP	Lomonosov–2	512.0	2^{20}
Bellman–Ford	Ligra	Lomonosov–2	511.0	2^{20}
Bellman–Ford	RCC for GPU	Lomonosov	452.9	2^{20}
Bellman–Ford	RCC for CPU	Lomonosov	435.0	2^{21}
Bellman–Ford	RCC for CPU	Lomonosov–2	426.0	2^{21}
Bellman–Ford	RCC for CPU	Lomonosov–2	418.0	2^{20}
Bellman–Ford	Graph500 MPI	Lomonosov	350.0	2^{20}
Bellman–Ford	RCC for CPU	Lomonosov	204.1	2^{20}
Bellman–Ford	RCC for CPU	Lomonosov	183.5	2^{20}
Delta–Stepping	PBGL MPI	Lomonosov	174.0	2^{21}
Dijkstra’s	PBGL MPI	Cluster “Angara”	150.0	2^{20}
Delta–Stepping	PBGL MPI	Lomonosov	124.1	2^{21}
Bellman–Ford	Graph500 MPI	Lomonosov	120.0	2^{20}
Bellman–Ford	Graph500 MPI	Lomonosov	18.0	2^{20}
Bellman–Ford	Graph500 MPI	Lomonosov	11.8	2^{21}
Dijkstra’s	PBGL MPI	IBM BlueGene/P	8.9	2^{20}
Dijkstra’s	PBGL MPI	Lomonosov	5.3	2^{21}
Delta–Stepping	PBGL MPI	IBM BlueGene/P	3.8	2^{20}
Dijkstra’s	PBGL MPI	Cluster “Angara”	2.5	2^{21}
Delta–Stepping	PBGL MPI	IBM BlueGene/P	1.3	2^{20}
Dijkstra’s	PBGL MPI	IBM BlueGene/P	0.6	2^{20}

compiler used, compiler options, settings for generating input graphs and other relevant parameters. Details on the implementations presented in the table can be found here: Ligra [13], GAP [14], PBGL [15], “RCC for CPU/GPU” correspond to an AlgoWiki user’s own implementations for CPU/GPU. Table 2 compares different implementations of different algorithms for solving the “Single Source Shortest Paths” problem on different platforms, for graphs with 2^{20} and 2^{21} nodes.

It should be noted that in these examples, specific performance values are obtained by AlgoWiki users and are largely determined by their experience and diligence. These are not necessarily the top performance figures: they show results obtained by people in practice. There is only one requirement: when submitting data in the AlgoWiki database, users must provide

all of the information that would be needed to reproduce and verify the results, and possibly improve upon them in the future.

Conclusion

The first stage of the AlgoWiki project was geared toward achieving the following two goals: developing a technology for describing the parallel structure of algorithms, and widening the project database with descriptions of actual algorithms. These goals have been accomplished, which enables further project development in a number of new areas. In particular, one of those areas is to extend the Top500 methodology used to compare high-performance computing systems. Implementing this will require expanding the project's functionality, giving AlgoWiki users a chance to save data on algorithm execution parameters in the project database. As a result, the project will not only provide detailed algorithm descriptions, but also enable the review and comparison of algorithm execution results on any computing platform.

Acknowledgements

The results were obtained in Lomonosov Moscow State University with the financial support of the Russian Science Foundation (Agreement № 14–11–00190). The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Press, W., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C. Cambridge University Press, second edition (1992)

2. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., Van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition, SIAM (1994), `http://www.netlib.org/linalg/html_templates/Templates.html`, accessed: 2018-03-22

3. List of algorithms. `https://en.wikipedia.org/wiki/List_of_algorithms`, accessed: 2018-03-22

4. Enabling AI in every Application. `http://algorithmia.com/`, accessed: 2018-03-22

5. ALGLIB. `http://www.alglib.net/`, accessed: 2018-03-22

6. A Library of Parallel Algorithms. `http://www.cs.cmu.edu/~scandal/nesl/algorithms.html`, accessed: 2018-03-22

7. Voevodin, Vl., Antonov, A., Dongarra, J.: AlgoWiki: an Open Encyclopedia of Parallel Algorithmic Features. In: Supercomputing Frontiers and Innovations, vol. 2, no. 1, pp. 4–18 (2015), DOI: 10.14529/jsfi150101

8. Antonov, A., Voevodin, Vad., Voevodin, Vl., Teplov, A.: A Study of the Dynamic Characteristics of Software Implementation as an Essential Part for a Universal Description of Algorithm Properties. In: 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Proceedings, 17–19 February 2016, pp. 359–363. DOI: 10.1109/PDP.2016.24

9. Voevodin, Vl., Antonov, A., Dongarra, J.: Why is it hard to describe properties of algorithms? In: Procedia Computer Science, vol. 101, pp. 4–7 (2016), DOI: 10.1016/j.procs.2016.11.002

10. Dongarra, J.J., Bunch, J.R., Moler, G.B., Stewart, G.W.: LINPACK Users' Guide. Society for Industrial and Applied Mathematics, 1979–1993.

11. Murphy, R.C., Wheeler, K.B., Barrett, B.W., Ang, J.A.: Introducing the Graph 500. In: Cray User's Group (CUG), May 5, 2010, vol. 19, pp. 45–74 (2010)

12. Heroux, M., Dongarra, J.: Toward a New Metric for Ranking High Performance Computing Systems. In: UTK EECS Tech Report and Sandia National Labs Report SAND2013–4744, June 2013.

13. Shun, J., Blelloch, G.E.: Ligra: a lightweight graph processing framework for shared memory. In: ACM Sigplan Notices, vol. 48, no. 8, pp. 135–146. DOI: 10.1145/2517327.2442530

14. Beamer, S., Asanovi, K., Patterson, D.: The GAP Benchmark Suite. arXiv:1508.03619 [cs.DC] (2015)

15. Parallel Boost Graph Library. `http://www.boost.org/doc/libs/1_51_0/libs/graph_parallel/doc/html/index.html`, accessed: 2018-03-22

DOI: 10.14529/jsfi180102

Record-and-Replay Techniques for HPC Systems: A Survey

Dylan Chapp[1], *Kento Sato*[2], *Dong H. Ahn*[2], *Michela Taufer*[1]

Record-and-replay techniques provide the ability to record executions of nondeterministic applications and re-execute them identically. These techniques find use in the contexts of debugging, reproducibility, and fault-tolerance, especially in the presence of nondeterministic factors such as message races. Record-and-replay techniques are highly diverse in terms of the fidelity of replay they provide, the assumptions they make about the recorded application, the programming models they target, and the runtime overheads they impose. In the high performance computing (HPC) environment, all the above factors must be considered in concert, thus presenting additional implementation challenges. In this manuscript, we survey record-and-replay techniques in terms of the programming models they target and the workloads on which they were evaluated, providing a categorization of these techniques benefiting application developers and researchers targeting exascale challenges. This manuscript answers three questions through this survey: What are the gaps in the existing space of record-and-replay techniques? What is the roadmap to widespread use of record-and-replay on production-scale HPC workloads? And, what are the critical open problems that must be addressed to make record-and-replay viable at exascale?

Keywords: reproducibility, nondeterminism, fault-tolerance, exascale, message-passing, shared memory, proxy application, HPC benchmarks.

Introduction

Record-and-replay (R&R) techniques provide the ability to monitor and record changes in program state over one execution (i.e., the recorded execution) of an application and reproduce those changes–and thus, the behavior of the application–during a subsequent execution (i.e., the replayed execution). The convergence of extremely high levels of hardware concurrency, the effective overlap of computation and communication, and overall code complexity make R&R a vital tool for coping with non-determinism in HPC applications. Non-determinism in HPC applications is a growing problem [1, 15, 48] and it manifests at multiple levels. Non-determinism may manifest in low-level communication primitives (e.g., the inherent non-determinism of non-blocking matching functions in MPI); it may manifest in libraries (e.g., dynamic load-balancing libraries [29]); or it may manifest at the application level (e.g., Monte-Carlo simulations). The common motivation of all R&R techniques is to manage these forms of non-determinism. Specifically, R&R techniques play key roles in three different HPC contexts. First, these techniques are used for debugging parallel programs that exhibit non-deterministic bugs during a code execution when moving, for example, from a smaller to a larger scale or from one platform to a different one. Second, R&R is useful during testing parallel programs, where users may want to ensure numerical or scientific repetition of certain results despite, for example, not enforcing specific message interleavings. Third, R&R techniques find alternative use in the context of fault-tolerance where the execution of one or more processes may need to be rolled back and replayed to recover from a fault.

This survey presents a rigorous classification of R&R techniques based on the programming model the R&R technique targets (i.e., shared memory or distributed memory) and the workloads on which the techniques have been evaluated (i.e., theoretical, non-HPC, HPC benchmarks, and HPC applications). For each programming model, we refine our categorization based

[1]University of Delaware, Newark, United States
[2]Lawrence Livermore National Laboratory, Livermore, United States

on the overall approach the technique takes (i.e., data-replay, order-replay, or a hybrid of the two). This dimension is shown in Fig. 1 and 2 on the horizontal axis. The vertical axis roughly corresponds to time, as we show the evolution of R&R techniques through conceptual eras. For each technique, we indicate the category of workloads it is evaluated against. We consider four categories: first, an absence of empirical evaluation (i.e., for purely theoretical works); second, non-HPC workloads (e.g., network servers, as these are common evaluation workloads for shared memory techniques); third, HPC benchmarks (e.g., the NAS Parallel Benchmarks); and fourth, full HPC applications. In Fig. 1 and 2 we represent the paper that introduced each technique as a box. The evaluation workloads are indicated by symbols in the boxes. For techniques that are evaluated against multiple categories of workloads, multiple symbols are shown. Additionally, due to the impact of logical clock algorithms on R&R (both for shared and distributed memory techniques) and interest in leveraging specialized hardware (for shared memory techniques only), we color the boxes corresponding to use of these features.

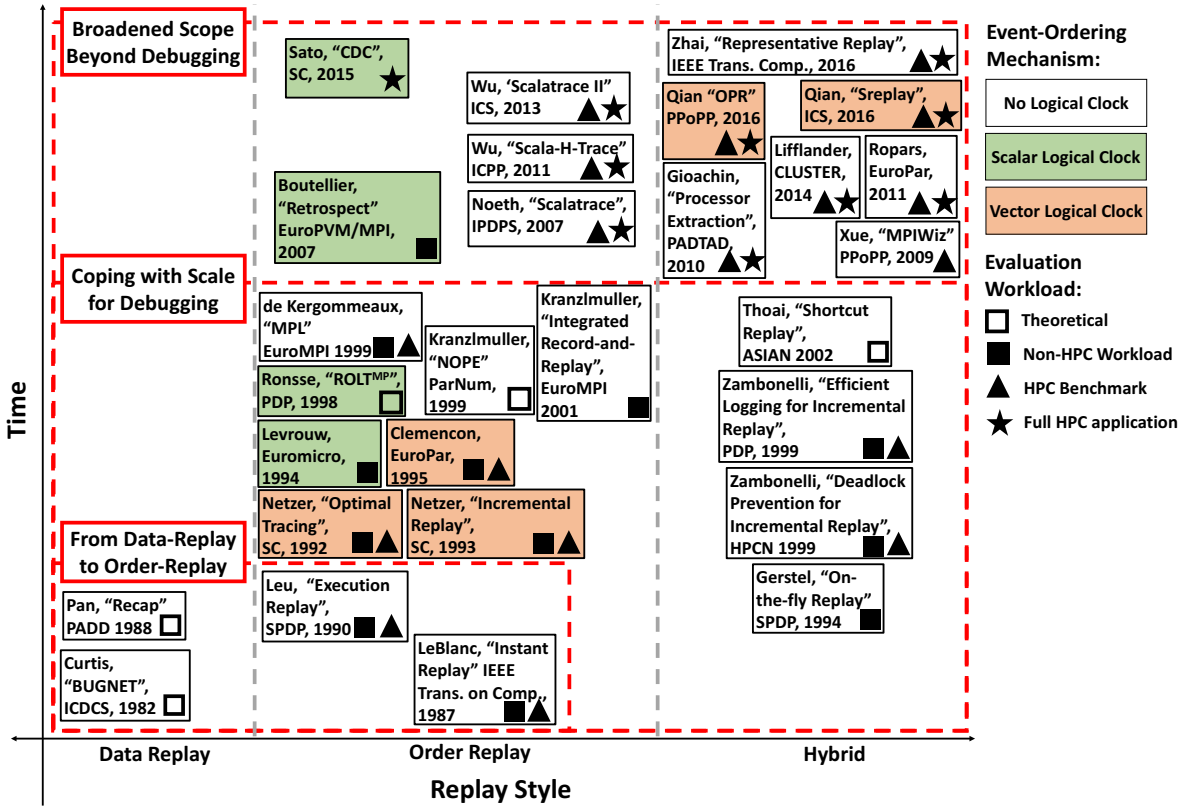


Figure 1. Record-and-replay techniques for distributed memory programming models

Our survey targets three communities: 1) researchers who are interested in filling the gaps in the existing space of R&R techniques; 2) developers and maintainers of HPC applications who use R&R primarily as a debugging aid; 3) the community of researchers exploring uses of R&R at exascale, including for fault-tolerance, resilience, and reproducibility. For each of these communities we identify a guiding question that this survey answers. Our survey supports researchers attempting to design R&R techniques that address so-far-unexplored regions of the technique space. They will benefit from the taxonomy that this survey develops by learning how to situate their work. The guiding question for this community is: Where are the gaps in the technique space? An HPC developer benefits from this survey because it informs them about which R&R techniques apply to their workloads. The guiding question for this community is:

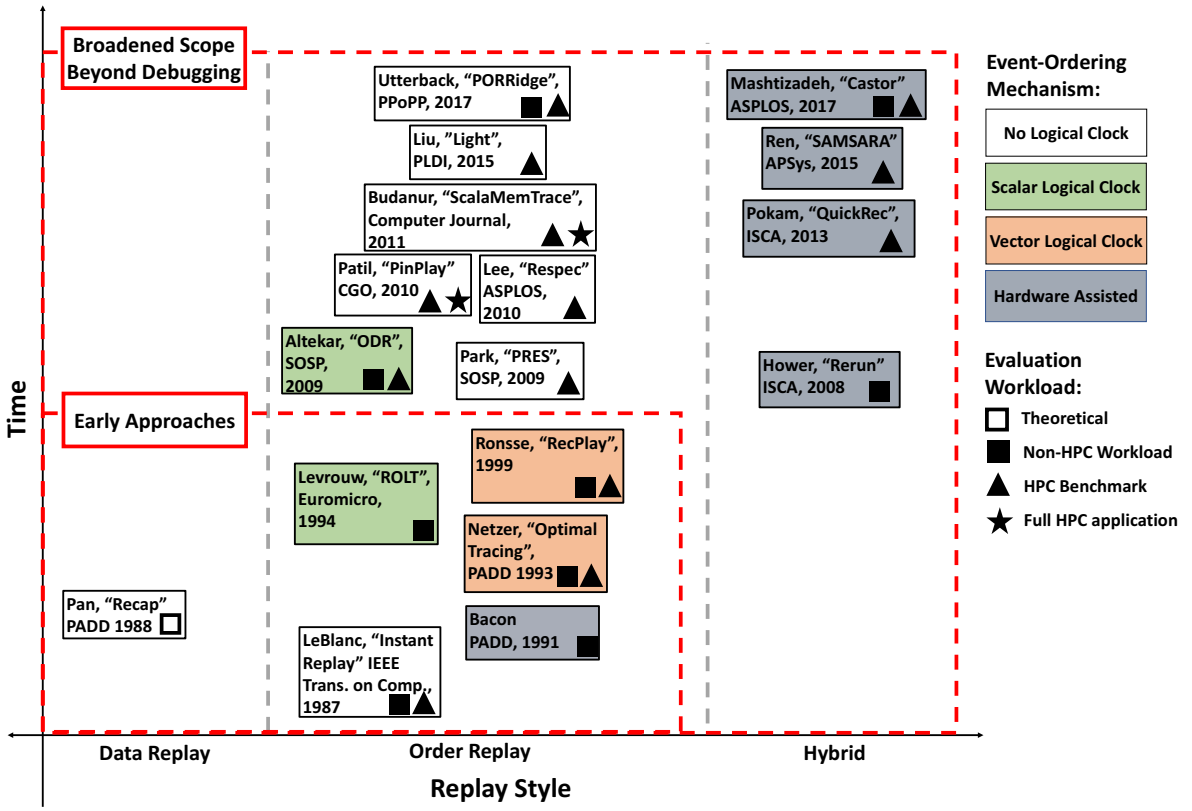


Figure 2. Record-and-replay techniques for shared memory programming models

What are the workloads to which R&R techniques apply? Finally, an exascale-centric researcher benefits from this survey by gaining a holistic view of how R&R techniques have overcome scaling challenges. The guiding question for this community is: What are the open problems standing between the state-of-the-art in R&R and suitability for future exascale systems?

The remainder of the article is structured in the following way. In Section 1, we provide terminology and definitions needed for evaluating R&R techniques. In Section 2, we discuss techniques targeting shared memory programming models. In Section 3, we discuss R&R techniques targeting distributed memory programming models, specifically focusing on techniques targeting message-passing applications using MPI due to its prevalence in HPC. Conclusion contains the lessons learned from our evaluation and concludes the article.

1. Concepts and Definitions

In this section, we introduce concepts and definitions that recur throughout the manuscript. Regardless of whether an R&R technique is being used for debugging, computational reproducibility, or fault-tolerance, the technique must observe and represent nondeterministic application behavior during the recording phase, and then recover that same behavior during replay. First, we introduce terms to describe the generic styles of R&R. Next, we define terms that are common across all R&R techniques. We conclude this section by introducing the concept of replay fidelity and relating it to the level of determinism present in the recorded application.

Data-replay techniques explicitly record the contents of communication. In the context of R&R techniques for distributed memory applications, specifically message-passing applications, a data-replay technique records the contents of messages. For techniques targeting shared-memory applications, a data-replay technique records the values read from or written to shared

memory locations. In contrast with data-replay, order-replay techniques only record the relative ordering of communication. In the context of distributed-memory applications, specifically message-passing applications, an order-replay technique records the interleaving of message receptions. For shared-memory applications, an order-replay technique records the order in which threads accessed shared memory locations. Order-replay techniques avoid the overheads associated to copying and storing large message contents, but must make more assumptions about the global ordering of communication (e.g., that messages are delivered through FIFO queues) than data-replay techniques. Hybrid-replay techniques combine data-replay and order-replay approaches in an attempt to balance the greater scalability of order-replay with the richness of debugging information and flexibility that data-replay provides (e.g., replaying only a subset of processes). Techniques that aim to avoid replaying many non-buggy processes in order to replay a single buggy one are referred to as subgroup-replay techniques.

Every R&R technique builds up a representation of an execution during the recording phase. This representation exists in memory during the recording and may be persisted to disk, either in chunks during recording or all at once at the end, and necessarily imposes some overhead in order to monitor events and update the representation. Consequently, we will refer to two kinds of overheads: memory overhead and execution time overhead. The memory overhead refers to the size of the in-memory representation, and the execution time overhead refers to the slowdown relative to an unmonitored execution. Additionally, we will refer to the on-disk representation as the trace of the execution. Each technique also defines a level of replay fidelity, by which we mean a contract-like description of which events in the replayed run are guaranteed to match those from the recorded run. For instance, a technique may guarantee that messages are received by each process in the same order during replay only if they are sent in the same order during replay. Another technique may guarantee that both types of events (i.e., receives and sends) interleave in the same order. In this case, the second technique is said to have higher replay fidelity than the first one.

Replay fidelity, specifically the ease with which the desired level of it can be achieved, is intimately related to the determinism class [7] of the application being recorded. HPC applications commonly consist of intervals of deterministic computation bounded by periods of communication or synchronization, which may incorporate some element of non-determinism. The determinism class of an application is largely determined by how this communication is implemented, specifically how it is overlapped by computation. Generally speaking, a high level of replay fidelity is more challenging to achieve for applications the determinism class of which is less strict.

2. Record-and-Replay for Shared Memory

In this section we discuss R&R techniques that target shared memory models. These techniques target replaying the behavior of an application on a single node of an HPC system. The presence of fat nodes and accelerators in HPC systems makes these techniques relevant for peta and exascale computing. First, we discuss early approaches to R&R for shared memory applications. Next, we cover techniques that employed logical clocks as a means of managing trace growth as systems scaled in size. Finally, we survey contemporary techniques which we classify as either debugging-oriented (i.e., emphasizing high-fidelity replay) or targeting reproducibility of execution characteristics in a more relaxed sense.

2.1. Early Approaches – Plurality of Directions

From the late 1980s through the 1990s, R&R for shared memory models explored tradeoffs between replay styles (i.e., data-replay vs. order-replay), but focused primarily on debugging, and thus, high-fidelity replay. In 1987, LeBlanc introduced Instant Replay [23], which adopted an order-replay approach of associating a version number to each shared-memory access. Instant Replay is a common predecessor not only of later order-replay techniques for shared memory models, but also of many techniques for distributed memory models as discussed in Section 3. The empirical evaluation of Instant Replay was conducted on a 128-cpu system and used a Gaussian Elimination workload, indicating early interest in facilitating debugging of HPC applications. In contrast, Pan's 1988 Recap [38] technique combined periodic checkpointing with a data-replay logging approach. Additionally, Recap requires the compiler to generate instrumentation code, rather than being implemented as a library. While achieving greater flexibility of replay (e.g., the ability to replay from a checkpoint as opposed to the start of the execution), Recap is limited by the daunting growth rate of its log. Beyond order and data-replay, hardware-assisted approaches also emerged. The first R&R technique designed to explicitly take advantage of hardware was introduced by Bacon and Goldstein in 1991 [3]. The latter technique logs traffic between the main memory and caches and establishes a total order on shared-memory accesses by recording instruction counter values.

In the early 1990s, Netzer introduced the shared memory version of his Optimal Replay technique [35], addressing the overhead associated with tracing every shared memory access via on-the-fly race detection using vector clocks [12]. Netzer's work on Optimal Replay, both the shared memory version discussed in this section and the distributed memory version discussed in the following section, are among the first in the R&R space to acknowledge the need for adaptive tracing, to explore the tradeoff between execution time overhead and the memory footprint of the trace [33], and to leverage logical clocks. In both the original and subsequent work on Optimal Replay, the technique was evaluated against the SPLASH-2 benchmark suite on up to 16 threads.

To contrast with the overheads imposed by using vector clocks in [35], Levrouw et al. introduced an order-replay technique [26] based on scalar logical clocks [22]. The authors provide theoretical justification and empirical evaluation supporting their claim that their technique can outperform its two major competitors at the time–Netzer's Optimal Replay and LeBlanc's Instant Replay–both in terms of recording overhead and memory footprint. However, the evaluation workloads do not resemble HPC workloads (e.g., sorting) and are run on a maximum of four threads.

This period of development culminated in Ronsse and de Bosschere's RecPlay [45] debugging framework. This work uses multiple kinds of logical clocks: scalar clocks to order synchronization events during recording, and snooped matrix clocks [4] to do race detection during replay and abort when data races prohibit correct replay. RecPlay was evaluated against the SPLASH-2 benchmark suite, demonstrating worst-case execution time overhead during recording of 25.9%. As will be seen throughout the rest of this manuscript, the idea of augmenting R&R through logical clocks persists to the present day.

2.2. Broadened Scope: Adding Reproducibility

Contemporary approaches to R&R of shared-memory applications address a broad spectrum of reproducibility challenges. In this section, besides the traditional debugging-centric techniques, we discuss techniques that target more relaxed notions of reproducibility. Across these techniques we observe two trends: increased diversity of programming models targeted, and increased interest in hardware assisted approaches.

2.2.1. Reproducibility-Centric Techniques

Due to the overheads associated with high-fidelity replay, recent techniques targeting multi-core systems have opted to relax the requirement of exactly replaying the recorded run. Instead, these techniques try to replay a run the output of which matches the recorded run's output, regardless of the internal state-transitions that produce the output. Altekar introduced Output-Deterministic Replay (ODR) [2], which records only a subset of the necessary information to guarantee reproducible outputs but, augments this subset during replay with a search procedure to iterate towards a thread schedule that reproduces the final state of the recorded run. While this strategy can be applied to debugging, it can also clearly aid in situations where reproducibility of numerical outputs is desired. ODR was evaluated against a suite of applications including three of the SPLASH-2 benchmarks and demonstrated average recording overhead of 1.6×.

Simultaneously, Park introduced a similar technique called Probabilistic Replay with Execution Sketching (PRES) [39]. PRES also relaxes the common replay fidelity requirement, but in a different way. Rather than defining an equivalent execution more loosely, PRES proposes to gradually approach exact replay over multiple replay runs. By building up an execution sketch during recording and then iteratively exploring the constrained set of possible executions that agree with the sketch, PRES is able to hone in on executions that reproduce a buggy run. The evaluation was conducted on a subset of the SPLASH-2 benchmarks on up to four threads, at which scale execution time overhead of 10.6% was observed for an N-Body workload.

Inspired by fault-tolerance, Lee introduced Respec [24] which also targets determinism at the output level. In contrast with most popular techniques used at this point in time, Respec runs its "replay" concurrently with the monitored execution, periodically checking for divergence between the two runs and maintaining a checkpoint of the executions' last agreed-upon state. Respec manages overhead by only logging a subset of the most common synchronization operations, and borrows ideas from fault-tolerance protocols to correct itself when the replaying execution drifts too far from the monitored execution. Respec was evaluated, on up to four threads, on the PARSEC and SPLASH-2 benchmark suites.

At the beginning of this decade, many codes are parallelized on HPC systems by using a shared-memory model on each node coupled with a distributed-memory model for inter-node communication (i.e., the MPI/X model). Budanur et al., introduced an R&R technique for studying memory inefficiencies in this scenario called ScalaMemTrace [6]. ScalaMemTrace leverages Extended Power Regular Section Descriptors (EPSRDs) to exploit repeated behavioral patterns across multiple levels of the memory hierarchy to realize a compressed representation of a memory trace. ScalaMemTrace was evaluated against HPC-centric workloads, namely the MPI/OpenMP implementation of the Sequoia AMG benchmark and two computational kernels: matrix-matrix multiplication and vector addition. In a weak-scaling study of the kernels, ScalaMemTrace's compression afforded near-constant trace size on up to 64 threads, in contrast

with the uncompressed version of the same trace which grew exponentially with the number of threads. In a strong-scaling study of the AMG benchmark, the compressed traces grew linearly with the number of threads, but nevertheless afforded a 50% reduction in size. The manuscript acknowledges that ScalaMemTrace's replay fidelity is limited, namely that the AMG benchmark was only replayed at 91% accuracy.

2.2.2. Debugging-Centric Techniques

While the first reproducibility-centric R&R techniques appeared and evolved, as described in the previous section, debugging-centric techniques still remain a high priority. The major challenge shared memory presents for debugging is the sheer number of events (i.e., shared memory accesses) that must be traced to ensure the high-fidelity replay required. Contemporary debugging-centric techniques address this challenge via two strategies: first, working primarily at the software level (e.g., by employing an algorithm to ensure that a necessary and sufficient trace is recorded for the desired level of replay fidelity); and second, by leveraging specialized hardware to reduce recording overhead (e.g., by using hardware clocks with tighter synchronization guarantees).

At the software level, we observe a single vendor-driven effort to address the challenges of R&R with a highly general technique. Intel proposed PinPlay [40], which is an R&R technique built atop their Pin dynamic instrumentation framework. PinPlay provides highly flexible R&R options (e.g., subgroup replay) and offers composability with other Pin-based tools (thus, pursuing generality). PinPlay's evaluation was conducted on realistic HPC workloads, including the Parallel Ocean Program, the MILC quantum chromodynamics code, and a weather prediction application. On these workloads, PinPlay observes between 36× and 147× execution time overhead during recording, which, despite being high, is indicative of the challenge of recording sufficient information for high-fidelity replay of scientific applications.

From the academic community, efforts include reformulation of replay as a satisfiability problem, and specialization to task-parallel runtimes.

Liu et al.'s Light [28] introduces a novel software-level technique leveraging Satisfiability Modulo Theory (SMT) solvers. Light formally characterizes the minimal trace data needed for the desired level of replay. Light focuses on logging flow dependence of shared memory accesses, and then during replay generates a thread schedule that respects the recorded flow dependencies by formulating it as a satisfiability problem. Light is evaluated on a diverse set of benchmarks ranging from scientific applications to web server and web crawling applications.

Though all techniques in this section have focused on recording threads' behavior, in the case of task-based parallel runtimes such as Cilk Plus, this is not an option since the runtime's scheduler may employ variable numbers of threads from run to run in a nondeterministic fashion. A so-called "Processor-oblivious" R&R is needed in this case. PORRidge [51] is the first such technique of this kind, which targets Cilk Plus programs where shared objects are protected by locks. Rather than maintaining per-thread records, PORRidge maintains per-lock records and constrains the Cilk scheduler to conform to these records' access orders by augmenting the scheduler's DAG-representation of the execution with extra edges that represent the happens-before relationships observed during recording. PORRidge is evaluated against a diverse set of benchmarks, including two from the PARSEC suite, and three nondeterministic graph algorithms from the Problem Based Benchmark Suite. The execution time overhead of the recording phase

ranges from essentially negligible to 3.39×, with a mean of 1.62× across all benchmarks. The graph benchmarks saw the highest overheads for both recording and replaying.

In parallel with novel software-level approaches to R&R, researchers have endeavored to exploit advances in hardware and propose future hardware augmentations. In an effort to reduce recording overhead while maintaining the level of replay fidelity needed for debugging Hower et al. introduced Rerun [17], which proposes to use relatively little dedicated memory per core compared with prior approaches, coupled with a scalar logical clock to establish a partial order on atomic episodes, i.e. periods during which one thread does not race with any other. Additional hardware-assisted techniques were proposed throughout the 2010s, most notably QuickRec [41] which proposes an extension of Intel's architecture to support efficient record and replay, and SAMSARA [44] which leverages hardware-assisted virtualization extensions. QuickRec was evaluated on the SPLASH-2 benchmarks on up to eight threads, whereas SAMSARA was evaluated on the PARSEC benchmarks on up to four threads.

More recently, the tool Castor [30] has been developed to provide recording with such low overhead that it may be left on by default, rather than only used during specific debugging runs. Castor uses a time stamping procedure that relies on synchronized hardware clocks. In contrast with other tools, Castor monitors replayed execution and dynamically evaluates whether the replay sufficiently matches the record. Castor maintains per-thread records of nondeterministic events which are aggregated to a master record periodically. Castor's primary contribution is maximization of log throughput during recording and replay by use of transactional memory and recording nondeterminism at multiple levels by combining an LLVM pass, library interpositioning, and thread-level logging. Castor is evaluated against the PARSEC benchmark suite. Though Castor demonstrates low recording overhead for most PARSEC benchmarks, the Radiosity benchmark incurs overheads up to 25% when running on 10 threads. The authors claim that this is due to Radiosity's sensitivity to the log aggregation's effect on caches and the fact that Radiosity periodically overwhelmed the aggregator.

3. Record-and-Replay for Distributed Memory

In this section we discuss R&R techniques that target distributed memory models. As the Message-Passing Interface (MPI) [32] became the *de facto* standard for distributed-memory HPC applications, R&R techniques increasingly targeted MPI applications. Therefore, we focus on these techniques in this section. First, we discuss early approaches to R&R techniques, moving from data-replay to order-replay in pursuit of scalability. Next, we identify an era of progress in order-replay due in large part to the integration of logical clock algorithms. Finally, we identify contemporary techniques which we classify into three main research directions: first, debugging-oriented techniques with an emphasis on high-fidelity replay; second, techniques oriented towards reproducibility of communication characteristics and performance analysis; and third, techniques that straddle the line between R&R and fault-tolerance.

3.1. Early Approaches – from Data-Replay to Order-Replay

The earliest era of R&R techniques for distributed memory applications took the data-replay approach for three reasons. First, early R&R tools were developed in the debugging community rather than the HPC community, so logging messages' content to provide rich debugging information at the expense of scalability was seen as an acceptable tradeoff. Second, message-passing

standards such as MPI were still maturing, which limited the development of order-replay techniques. Third, data-replay does not require that all processes be replayed, allowing users to focus on buggy processes. Representative data-replay techniques from this era include BUGNET [11] and Recap [38]. BUGNET logged all message contents and represents one of the first tools specifically designed to alleviate the burden of nondeterminism when debugging parallel code. Recap represented a step forward in two ways. First, and most prominently, Recap was one of the first tools to provide subgroup replay; recognizing that in the debugging context, only faulty processes need to be replayed. Second, Recap explicitly acknowledges that the rate of trace generation constrains the technique's applicability.

As the HPC community began to recognize the need for R&R tools, order-replay soon emerged as an attractive alternative to data-replay. LeBlanc's Instant Replay [23] was the earliest such technique, as discussed in Section 2.1. Instant Replay models all interprocess communication as accesses to shared objects and assigns version numbers to each such access during the recording phase. This approach circumvents the need to copy and store message buffers' contents, and so is attractive for the debugging scenario where an application may need to run for a long time before a bug manifests. The empirical evaluation of Instant Replay demonstrated the space savings that can be realized by an order-replay approach. Specifically, LeBlanc shows that for a Gaussian Elimination workload on 64 processes, Instant Replay's trace occupied over 300x less space than a data-replay trace wherein all messages' contents are logged.

Despite achieving significant reduction in trace size, Instant Replay presented challenges to practical implementation, which were identified and addressed by Leu et al. in their proposal of Execution Replay [25]. Leu observed that Instant Replay's versioning algorithm requires sending extra messages, which may interfere with debugging. Execution Replay constrains itself to send no additional messages. Additionally Leu took steps towards formalizing the notion of equivalent executions (i.e., What does it mean to say that a replay is "correct"?). Finally, Leu explicitly addresses replay of both blocking and non-blocking communication–a distinction absent in prior work. Throughout this initial period of development, we observe the emergence of the key driving problems that guide research into R&R to this day (i.e., the tradeoff between memory overhead and execution time overhead). Moreover, the evolution of systems architecture exacerbated the need to manage the above-mentioned tradeoff. This lead to the burst of research on R&R techniques throughout the 1990s and early 2000s discussed in the next section.

3.2. Coping with Scale for Debugging

Order-replay techniques were developed in response to the growing need in HPC for scalable R&R. Through the 1990's and early 2000's, two related design goals emerged. First, minimizing the memory overhead and trace size was recognized as necessary to apply R&R to long-running, memory-hungry HPC applications. Second, minimizing the execution time overhead was recognized as necessary, because even small changes in event timings could prevent bugs from manifesting during the recording phase. Logical clocks [22] (i.e., algorithms for establishing orders on events across multiple processes) became instrumental to tackling both problems. Both challenges are addressed in the following sections, discussing enhancments associated to logical clocks and hybridization of replay techniques.

3.2.1. Enhancing Order-Replay Techniques with Logical Clocks

Netzer's work on Optimal Replay [36] is archetypical of the research direction for R&R during the 1990's. The general idea was to detect potential message races at runtime using a vector-valued logical clock [12] and log only enough information to cause the races' outcomes during replay to match their outcomes during the recorded run. All other non-racing communication could be safely assumed to repeat without the guidance of the recorded trace. Empirical evaluation of Optimal Replay on a set of computational kernels (e.g., matrix multiplication) indicated that only 1-14% of messages required tracing, leading to significant reductions in tracing overhead. Beyond Optimal Replay, Netzer also introduced Incremental Replay [34], which combined checkpointing with the kind of message-logging common in R&R to allow replayed executions to "fast-forward" to the region of the execution where a bug actually occured. While the focus on debugging is clear, this work is indicative of the conceptual overlaps between the debugging and HPC fault tolerance communities – a trend that has persisted to the present.

Shortly following Netzer's work on Optimal Replay, MPI was standardized and embraced by the HPC community as the *de facto* mechanism for distributed memory scientific applications. Consequently, R&R techniques for distributed memory applications began to specifically target MPI. The earliest example of this is Clemencon's work [9] on the *Annai* programming environment's parallel debugging tool. Clemencon's technique builds on the vector-clock-based race detector of Optimal Replay, adding an additional scalar logical clock per process. This work is significant not only in its use of logical clocks, but also in that it addresses the nondeterminism inherent in non-blocking communication, including non-blocking probes. This work was evaluated on communication microbenchmarks, as well as more HPC-oriented benchmarks such as BiCGSTAB, on up to 64 processes.

Despite the power of vector clocks to determine when events must be recorded, the need to piggyback vector timestamps on all messages becomes increasingly prohibitive as system size increases. ROLT^{MP} [46] attempts to overcome this scalability barrier by using scalar logical clocks during recording to determine when events must be explicitly logged, coupled with additional checks during replay to compensate for the limited ability of scalar clocks to order events across processes [8].

As MPI saw increasingly widespread adoption in HPC, techniques for R&R emerged that leverage MPI's point-to-point communication semantics in order to reduce recording overhead for applications with limited nondeterminism. Kranzlmuller introduced such a technique in the Nondeterministic Program Evaluator (NOPE) [20]. This work was novel not only in that it forgoes logical clocks in favor of a simpler logging mechanism that leverages MPI's non-overtaking rule, but also in that NOPE supports replay of alternative executions (i.e., one execution is recorded but multiple possible executions, including the recorded one, may be replayed). NOPE's drawbacks are that it assumes the only sources of receiver-side nondeterminism are wildcard receives, and that the recorded application is send-deterministic [7].

In addition to leveraging the features of MPI itself, R&R tools were developed in response to the particular programming idioms HPC developers wrote into their codes. De Kergommeaux et al. developed MPL* [18] which targeted correct replay of MPI applications that use non-blocking test functions in polling loops. This programming idiom is commonly used to overlap communication and computation, but one result is that the number of tests becomes nondeterministic. MPL* compactly represents sequences of test calls such that applications that use the number of test calls (e.g., for control flow) can be correctly replayed. Kranzlmuller later went a step

beyond, opting to integrate R&R functionality into the MPI library itself [19]. The primary advantage of such a scheme is convenience for the user in a debugging context since there is no need to instrument code, link with additional libraries beyond MPI, or manage traces. Later work by Bouteiller et al. produced Retrospect [5], which integrates R&R directly into an MPI implementation's lower-level communication routines and can switch between data-replay and order-replay strategies based on user needs. Retrospect set a new standard for empirical evaluation of distributed memory R&R techniques as it was evaluated on the NAS Parallel Benchmarks on up to $1,024$ processes.

3.2.2. Hybrid Replay and Integration of Checkpointing

Due to the growing popularity of order-replay, data-replay approaches were not explored to the same degree. However, notable exceptions include Gertsel's On-the-Fly Replay [13] and several works of Zambonelli [55, 56]. Gerstel's work introduced a scheme where the entire network of processes is duplicated, and one-way channels between the processes in the "real" network propagate the nondeterministic actions of those processes to their partners in the other network. In a sense, this scheme pursues "replay-while-recording" debugging. In contrast, Zambonelli's work builds off of Netzer's work on Incremental Replay, first providing a solution to avoid deadlocks then optimizing its message logging algorithm. This work showed that by piggybacking limited additional information on all messages, its message-logging scheme reduced the number of messages that needed to be logged by 10%.

Along similar lines, Thoai's Shortcut Replay [50] proposes replaying from a combination of globally consistent and inconsistent checkpoints. The value proposition of Shortcut Replay is being able to skip segments of the execution during replay that are not immediately relevant to a debugging task. Shortcut Replay leverages the event graph model of MPI programs' executions to determine how to safely combine checkpoints to avoid inconsistency during replay.

3.3. Broadened Scope: Adding Reproducibility and Fault Tolerance

Contemporary research in R&R has advanced along two novel directions in addition to debugging: reproducibility and fault-tolerance. First, a class of R&R techniques targeting more relaxed definitions of reproducibility (e.g., for characterizing the communication behaviors of applications or studying performance portability) has emerged. These techniques do not provide the fine-grained, high-fidelity replay of debugging-oriented techniques, but instead prioritize scaling to larger executions. Second, cross-pollination between the debugging and fault-tolerance communities has led to a class of fault-tolerance protocols involving message-logging that leverage techniques from traditional R&R. First we discuss these new directions, and we conclude this section by discussing contemporary approaches to debugging-centric record-and-replay.

3.3.1. Reproducibility-Centric Techniques

Although debugging continues to be a major focus of R&R, recent interest from the HPC community in reproducibility (of e.g., performance, communication behavior, scientific outcomes of simulations) has motivated R&R techniques that target replay with a more relaxed fidelty requirement than in the debugging case.

A prominent example of this trend is the Scalatrace tools. The original Scalatrace [37] tool proposed trace compression techniques that provide nearly constant-sized traces for MPI

applications with sufficiently regular communication. Regular section descriptors (RSDs) and their recursive counterparts power-RSDs are used to compactly represent MPI communication events, which, combined with a suite of MPI-specific encoding techniques, significantly reduce trace sizes for applications with relatively little nondeterminism.

Scalatrace's immediate successor, Scala-H-Trace [53] provides probabilistic replay. Rather than exactly replicating fine-grained MPI events such as individual message receptions, Scala-H-Trace allows the user to specify a tuning parameter that controls the tradeoff between fidelity of replay and scalability of recording. By forgoing the lossless tracing approach of its predecessor, Scala-H-Trace was able to achieve near-constant trace file sizes when recording executions of the Parallel Ocean Program on up to $2,048$ processes. In contrast, the original Scalatrace was only able to record the same executions on up to $1,1024$ processes. Additionally, Scala-H-Trace demonstrated constant trace sizes for the NPB conjugate gradient workload on $2,048$ processes. An important caveat is that these results were obtained using the lowest replay fidelity setting of Scala-H-Trace, and the metric used for evaluating correctness of replay is overall execution time. Despite limited applicability to debugging, techniques providing probabilistic replay can be invaluable for scientific reproducibility and for ensuring that the communication behavior of applications has characteristics that agree with programmer intent.

Scala-H-Trace's successor, Scalatrace II [52], also embraces probabilistic replay. Proposed as a major redesign of the compression techniques developed in Scalatrace for applications with more irregular behavior, Scalatrace II is evaluated against a subset of NPB, the Sweep3D neutron transport kernel, and the Parallel Ocean Program on up to 400 processes. Like Scala-H-Trace, Scalatrace II performs lossy compression of MPI events and thus its correctness metric for replay is overall execution time. In all evaluation workloads, Scalatrace II achieves good agreement between replay time and originally observed execution time–experience an average execution time error of 5.7%. Like its predecessor, the probabilistic replay Scalatrace II provides is useful for achieving coarse-grained reproducibility of communication behavior.

Scalatrace was evaluated on stencils of various sizes, a subset of the NPB, and Raptor, a simulation framework supporting adaptive mesh refinement used in this evaluation to run a hydrodynamics simulation. These workloads were recorded on up to 512 processes and in most cases demonstrated nearly constant trace size and memory footprint during recording. However, the conjugate gradient and Fourier transform benchmarks from NPB yielded trace sizes that increased with the number of the processors. In the case of the conjugate gradient benchmark, the role of asynchronous point-to-point communication in the increased trace size is acknowledged.

Most recently, Zhai et al. [57] introduced "Representative Replay", which targets performance prediction. Representative replay permits users to record an execution of a parallel application on a limited platform (e.g., a single node of a cluster) and obtain predictions of the execution time of the same application on the full platform via a controlled replayed execution.

3.3.2. Fault-Tolerance-Centric Techniques

The last aspect of R&R's applicability is in fault-tolerance. As the exascale era looms, fault-tolerance occupies an increasingly important role in HPC. R&R techhniques and fault-tolerance protocols (especially their message-logging aspects) share many concerns, namely concise representation of nondeterministic application behaviors. Similarities can also be seen between debugging-oriented techniques that seek to replay only buggy processes, and fault-tolerance protocols that combine checkpointing with message-logging; seeking to avoid rolling back ex-

ecution any farther than necessary. In this section, we discuss fault-tolerance protocols that incorporate deterministic replay.

Partial message-logging protocols, such as those introduced by Guermouche et al. [16] and Meneses et al. [31] depend on clustering of processes into process groups that realize similar communication patterns in order to achieive efficient deterministic replay. Ropars et al. introduced a process clustering technique [47] that leverages the regular communication patterns of MPI collectives and uses a bisection-based graph partitioning algorithm to compute process clusters that facilitate partial message-logging protocols. For empirical evaluation, Ropars' clustering algorithm is coupled with the above-cited partial message-logging protocols and ran against set an HPC benchmarks including the NAS Parallel Benchmarks and LAMMPS. The key contribution of this work is that the process clustering algorithm takes the characteristics of the partial message-logging scheme it is coupled to into account, resulting in process clusters that are conducive to reducing the overall number of messages logged. Specifically, the technique requires logging less than 5.3% of all messages for all but one of their workloads.

Due to the daunting task of achieving fault-tolerance at exascale, reseachers seek to leverage application characteristics to achieve scalable replay of failed processes. Lifflander et al. developed an algebraic methodology for their fault-tolerance protocol [27] that can determine when events in the execution of a distributed-memory application may safely commute (i.e., occur in any order without altering the program state after their completion), thus a message-logging R&R scheme with lower overheads. Beyond this work's theoretical contributions, the fault-tolerance protocol was rigourously evaluated against HPC applications across a variety of scientific domains (e.g., molecular dynamics and hydrodynamics). The evaluation platform was an IBM BG/P system with $40,960$ nodes, and the workloads were ran on up to $131,072$ processes.

3.3.3. Debugging-Centric Techniques

The debugging-oriented research direction of R&R continues to prioritize reducing the trace size and memory footprint of the recording phase. Xue et al. proposed Subgroup Reproducible Replay (SRR) [54] to address not only the problem of trace size, but also provide the user with flexible replay options (i.e. replaying only those processes deemed relevant to the debugging task). SRR leverages the fact that most HPC applications are structured so that any one process does most of its communication with a limited subset of other processes (e.g., communicating only with its neighbors in a stencil pattern). With this in mind, SRR adopts a hybrid-replay strategy by recording only message interleavings within groups of processes (order-replay), while recording the contents of messages between groups (data-replay). While it is acknowledged that application developers' expertise may be needed to specify the appropriate process groups, the grouping can also be obtained by partitioning a weighted graph representing the application's communication channels. SRR was evaluated on a diverse benchmark suite containing a subset of the NAS Parallel Benchmarks, as well as a parallel graph benchmark and a benchmark specifically designed to stress SRR's ability to replay nondeterministic probes. The benchmarks were run on a maximum of 64 processes.

In a distinct approch to hybrid-replay, Gioachin et al. introduced "Processor Extraction" [14] where multiple R&R passes are used to iteratively progress towards replaying a desired bug. The first pass uses a lightweight order-replay approach. Subsequent passes use increasingly heavyweight data-replay approaches but on increasingly small subsets of processes, thereby

narrowing down the root cause of bugs and limiting the tracing overhead. This technique was evaluated on up to 1,024 processes. The evaluation workloads ranged from synthetic benchmarks to full-fledged scientific applications, specifically ChaNGa and NAMD.

Despite the flexibility of hybrid-replay schemes, pure order-replay remains attractive for HPC, especially when fine-grained high-fidelty replay is desired for debugging. To this end, Sato et al. developed Clock-Delta Compression (CDC) [49], a technique for compressing the record of nondeterministic events. The technique piggybacks scalar logical timestamps on each message much in the same way that earlier techniques such as ROLT^{MP} do, but instead of explicitly storing the timestamps, CDC stores a permutation that maps a reference order, which can be deterministically generated during replay, to the observed order from the recorded run, thereby realizing a reduction in trace size. Clock-Delta Compression was evaluated against an HPC proxy application "The Monte Carlo Benchmark" (MCB) [10] on 3,072 processes, for which CDC was able to show a reduction in trace size of two orders of magnitude while maintaining execution time overhead of 13.1-25.5%.

Finally, as MPI continues to evolve, so must the R&R techniques targeting it. MPI's one-sided communication routines in particular pose unique challenges to R&R. Quian et al. proposed two techniques for addressing this challenge–OPR [42] and its successor SReplay [43]. SReplay proposes a hybrid-replay scheme which permits replay of subgroups of processes. Like MPIWiz, SReplay does order-replay within process groups for which it employs a specialized vector clock algorithm. Unlike previous works that have associated vector clocks to individual memory regions, ranges of regions are associated to the same vector clock in SReplay thereby enhancing its scalability. SReplay was evaluated against standard benchmarks for R&R tools (e.g., the NAS Parallel Benchmarks), but also against applications exhibiting considerably greater nondeterminism (e.g., unbalanced tree search and parallel DeBruijn graph construction) on up to 1,024 processes.

Conclusion

Record-and-replay techniques were originally developed enabling cyclic debugging in the presence of nondeterminism, but since then their scope has expanded to include more relaxed forms of reproducibility, as well as integrating into fault tolerance protocols. This expansion of scope is driven by the three communities we identify as associated to R&R. In this section we revisit the guiding questions of each community listed in Section and provide answers.

For researchers whose goal is to develop new R&R techniques, the guiding question is: Where are the gaps in the technique space? We observe three gaps. First, there are no techniques that offload recording overhead to accelerators, which are becoming more prevalent on HPC systems. Second, debugging-oriented techniques have not been evaluated at the scale of full system runs on petascale platforms. Third, there has been minimal investigation of composability of shared memory and distributed memory techniques.

For HPC developers, the guiding question is: What are the workloads to which R&R techniques apply? We observe that the workloads R&R techniques are evaluated against have grown in complexity (as evidenced in Fig. 1 by the increase in full HPC application evaluation workloads with time), however the scale at which recording is feasible depends strongly on desired replay fidelity. For reproducibility studies, recording runs on over $5K$ processes has become feasible. For debugging, particularly of long-running applications, $5K$ processes is still a barrier.

Finally, for exascale-oriented researchers, the guiding question is: What are the open problems preventing state-of-the-art R&R from being deployed at exascale? For this community, we consider first the ways R&R overcame scaling challenges in the past, then propose problems to consider for the future. Driven by the need to minimize recording overhead, R&R techniques employed logical clock algorithms, leveraged compression to reduce memory overhead and trace size, and proposed methods that adapt to application characteristics. For R&R to remain viable at exascale we observe three potential untapped directions. First, elements of recording and replaying may be able to be offloaded to GPUs or other accelerators. Second, machine learning may be needed to more-precisely adapt recording strategies to application characteristics. Third, logical clock algorithms continue to advance (e.g., logical-physical clocks [21]), exposing the possibility of upgrading old techniques with new event-ordering strategies.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by LLNL under contract DE-AC52-07NA27344 (LLNL-JRNL-749010).

References

1. Ahn, D.H., Lee, G.L., Gopalakrishnan, G., Rakamarić, Z., Schulz, M., Laguna, I.: Overcoming extreme-scale reproducibility challenges through a unified, targeted, and multilevel toolset. In: Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering. pp. 41–44. SE-HPCCSE '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2532352.2532357

2. Altekar, G., Stoica, I.: ODR: Output-deterministic Replay for Multicore Debugging. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. pp. 193–206. SOSP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1629575.1629594

3. Bacon, D.F., Goldstein, S.C.: Hardware-assisted Replay of Multiprocessor Programs. In: Proceedings of the 1991 ACM/ONR Workshop on Parallel and Distributed Debugging. pp. 194–206. PADD '91, ACM, New York, NY, USA (1991), DOI: 10.1145/122759.122777

4. Bosschere, K.D., Ronsse, M.: Clock snooping and its application in on-the-fly data race detection. In: 1997 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '97), 18-20 December 1997, Taipei, Taiwan. pp. 324–330 (1997), DOI: 10.1109/ISPAN.1997.645115

5. Bouteiller, A., Bosilca, G., Dongarra, J.: Retrospect: Deterministic Replay of MPI Applications for Interactive Distributed Debugging. In: Proceedings of the 14th European PVM/MPI User's Group Meeting, Paris, France, September 30 – October 3, 2007. pp. 297–306. Springer, Berlin, Heidelberg (2007), DOI: 10.1007/978-3-540-75416-9_41

6. Budanur, S., Mueller, F., Gamblin, T.: Memory trace compression and replay for SPMD systems using extended PRSDs. SIGMETRICS Performance Evaluation Review 38(4), 30–36 (2011), DOI: 10.1145/1964218.1964224

7. Cappello, F., Guermouche, A., Snir, M.: On communication determinism in parallel HPC applications. In: Proceedings of the 19th International Conference on Computer Communications and Networks, IEEE ICCCN 2010, Zürich, Switzerland, August 2-5, 2010. pp. 1–8 (2010), DOI: 10.1109/ICCCN.2010.5560143

8. Charron-Bost, B.: Concerning the size of logical clocks in distributed systems. Information Processing Letters 39(1), 11–16 (1991), DOI: 10.1016/0020-0190(91)90055-M

9. Clémençon, C., Fritscher, J., Meehan, M.J., Rühl, R.: An implementation of race detection and deterministic replay with MPI, pp. 155–166. Springer, Berlin, Heidelberg (1995), DOI: 10.1007/BFb0020462

10. Cleveland, M.A., Brunner, T.A., Gentile, N.A., Keasler, J.A.: Obtaining identical results with double precision global accuracy on different numbers of processors in parallel particle Monte Carlo simulations. Journal of Computational Physics 251, 223–236 (2013), DOI: 10.1016/j.jcp.2013.05.041

11. Curtis, R., Wittie, L.D.: BUGNET: A debugging system for parallel programming environments. In: Proceedings of the 3rd International Conference on Distributed Computing Systems, Miami/Ft. Lauderdale, Florida, USA, October 18-22, 1982. pp. 394–400 (1982)

12. Fidge, C.J.: Partial orders for parallel debugging. In: Proceedings of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, University of Wisconsin, Madison, Wisconsin, USA, May 5-6, 1988. pp. 183–194 (1988), DOI: 10.1145/68210.69233

13. Gerstel, O.O., Zaks, S., Hurfin, M., Plouzeau, N., Raynal, M.: On-the-fly replay: a practical paradigm and its implementation for distributed debugging. In: Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing, SPDP 1994, Dallas, Texas, USA, October 26-29, 1994. pp. 266–272 (1994), DOI: 10.1109/SPDP.1994.346158

14. Gioachin, F., Zheng, G., Kalé, L.V.: Robust Non-intrusive Record-replay with Processor Extraction. In: Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging. pp. 9–19. PADTAD '10, ACM, New York, NY, USA (2010), DOI: 10.1145/1866210.1866211

15. Gopalakrishnan, G., Hovland, P.D., Iancu, C., Krishnamoorthy, S., Laguna, I., Lethin, R.A., Sen, K., Siegel, S.F., Solar-Lezama, A.: Report of the HPC correctness summit, January 25-26, 2017, Washington, DC. CoRR abs/1705.07478 (2017), `http://arxiv.org/abs/1705.07478`, accessed: 2017-12-22

16. Guermouche, A., Ropars, T., Brunet, E., Snir, M., Cappello, F.: Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic MPI Applications. In: Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011. pp. 989–1000 (2011), DOI: 10.1109/IPDPS.2011.95

17. Hower, D., Hill, M.D.: Rerun: Exploiting Episodes for Lightweight Memory Race Recording. In: 35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China. pp. 265–276 (2008), DOI: 10.1109/ISCA.2008.26

18. de Kergommeaux, J.C., Ronsse, M., De Bosschere, K.: MPL: Efficient Record/Replay of nondeterministic features of message passing libraries, pp. 141–148. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/3-540-48158-3_18

19. Kranzlmüller, D., Schaubschläger, C., Volkert, J.: An Integrated Record&Replay Mechanism for Nondeterministic Message Passing Programs, pp. 192–200. Springer, Berlin, Heidelberg (2001), DOI: 10.1007/3-540-45417-9_28

20. Kranzlmüller, D., Volkert, J.: NOPE: A Nondeterministic Program Evaluator, pp. 490–499. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/3-540-49164-3_47

21. Kulkarni, S.S., Demirbas, M., Madappa, D., Avva, B., Leone, M.: Logical physical clocks. In: Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings. pp. 17–32 (2014), DOI: 10.1007/978-3-319-14472-6_2

22. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978), DOI: 10.1145/359545.359563

23. LeBlanc, T.J., Mellor-Crummey, J.M.: Debugging Parallel Programs with Instant Replay. IEEE Transactions on Computers 36(4), 471–482 (1987), DOI: 10.1109/TC.1987.1676929

24. Lee, D., Wester, B., Veeraraghavan, K., Narayanasamy, S., Chen, P.M., Flinn, J.: Respec: Efficient Online Multiprocessor Replay via Speculation and External Determinism. In: Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems. pp. 77–90. ASPLOS XV, ACM, New York, NY, USA (2010), DOI: 10.1145/1736020.1736031

25. Leu, E., Schiper, A., Zramdini, A.W.: Execution Replay on Distributed Memory Architectures. In: Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing, SPDP 1990, Dallas, Texas, USA, December 9-13, 1990. pp. 106–112 (1990), DOI: 10.1109/SPDP.1990.143516

26. Levrouw, L., Audenaert, K.M.R., Campenhout, J.M.V.: A New Trace And Replay System For Shared Memory Programs Based On Lamport Clocks. In: Proceedings of the Second Euromicro Workshop on Parallel and Distributed Processing, PDP 1994, January 26-28, 1994, Malaga, Spain. pp. 471–478 (1994), DOI: 10.1109/EMPDP.1994.592529

27. Lifflander, J., Meneses, E., Menon, H., Miller, P., Krishnamoorthy, S., Kalé, L.V.: Scalable replay with partial-order dependencies for message-logging fault tolerance. In: 2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014. pp. 19–28 (2014), DOI: 10.1109/CLUSTER.2014.6968739

28. Liu, P., Zhang, X., Tripp, O., Zheng, Y.: Light: Replay via Tightly Bounded Recording. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 55–64. PLDI '15, ACM, New York, NY, USA (2015), DOI: 10.1145/2737924.2738001

29. Lusk, E.L., Pieper, S.C., Butler, R.M., Univ., M.T.S.: More scalability, less pain : A simple programming model and its implementation for extreme computing. SciDAC Rev. 17(1), 30–37 (2010)

30. Mashtizadeh, A.J., Garfinkel, T., Terei, D., Mazieres, D., Rosenblum, M.: Towards Practical Default-On Multi-Core Record/Replay. In: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 693–708. ASPLOS '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3037697.3037751

31. Meneses, E., Mendes, C.L., Kalé, L.V.: Team-Based Message Logging: Preliminary Results. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia. pp. 697–702 (2010), DOI: 10.1109/CCGRID.2010.110

32. MPI: A Message-Passing Interface Standard, Version 3.0, `http://mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf`, accessed: 2017-12-22

33. Netzer, R.H.B.: Trace size vs parallelism in trace-and-replay debugging of shared-memory programs, pp. 617–632. Springer, Berlin, Heidelberg (1993), DOI: 10.1007/3-540-57659-2_35

34. Netzer, R.H.B., Xu, J.: Adaptive Message Logging for Incremental Replay of Message-passing Programs. In: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing. pp. 840–849. Supercomputing '93, ACM, New York, NY, USA (1993), DOI: 10.1145/169627.169850

35. Netzer, R.H.B.: Optimal Tracing and Replay for Debugging Shared-Memory Parallel Programs. In: Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, California, USA, May 17-18, 1993. pp. 1–11 (1993), DOI: 10.1145/174266.174268

36. Netzer, R.H.B., Miller, B.P.: Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs. In: Proceedings Supercomputing '92, Minneapolis, MN, USA, November 16-20, 1992. pp. 502–511 (1992), DOI: 10.1109/SUPERC.1992.236654

37. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable Compression and Replay of Communication Traces in Massively Parallel Environments. In: Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium. pp. 1–11 (2007), DOI: 10.1109/IPDPS.2007.370261

38. Pan, D.Z., Linton, M.A.: Supporting Reverse Execution for Parallel Programs. In: Proceedings of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. pp. 124–129. PADD '88, ACM, New York, NY, USA (1988), DOI: 10.1145/68210.69227

39. Park, S., Zhou, Y., Xiong, W., Yin, Z., Kaushik, R., Lee, K.H., Lu, S.: PRES: Probabilistic Replay with Execution Sketching on Multiprocessors. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. pp. 177–192. SOSP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1629575.1629593

40. Patil, H., Pereira, C., Stallcup, M., Lueck, G., Cownie, J.: PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs. In: Proceedings of the

8th Annual IEEE/ACM International Symposium on Code Generation and Optimization. pp. 2–11. CGO '10, ACM, New York, NY, USA (2010), DOI: 10.1145/1772954.1772958

41. Pokam, G., Danne, K., Pereira, C., Kassa, R., Kranich, T., Hu, S., Gottschlich, J., Honarmand, N., Dautenhahn, N., King, S.T., Torrellas, J.: QuickRec: Prototyping an Intel Architecture Extension for Record and Replay of Multithreaded Programs. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. pp. 643–654. ISCA '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2485922.2485977

42. Qian, X., Sen, K., Hargrove, P., Iancu, C.: OPR: Deterministic Group Replay for One-sided Communication. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 47:1–47:2. PPoPP '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2851141.2851179

43. Qian, X., Sen, K., Hargrove, P., Iancu, C.: SReplay: Deterministic Sub-Group Replay for One-Sided Communication. In: Proceedings of the 2016 International Conference on Supercomputing. pp. 17:1–17:13. ICS '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2925426.2926264

44. Ren, S., Li, C., Tan, L., Xiao, Z.: Samsara: Efficient Deterministic Replay with Hardware Virtualization Extensions. In: Proceedings of the 6th Asia-Pacific Workshop on Systems. pp. 9:1–9:7. APSys '15, ACM, New York, NY, USA (2015), DOI: 10.1145/2797022.2797028

45. Ronsse, M., De Bosschere, K.: RecPlay: A Fully Integrated Practical Record/Replay System. ACM Transactions on Computer Systems 17(2), 133–152 (1999), DOI: 10.1145/312203.312214

46. Ronsse, M., Kranzlmüller, D.: Rolt$^{\mathrm{mp}}$-replay of Lamport timestamps for message passing systems. In: PDP. pp. 87–93 (1998), DOI: 10.1109/EMPDP.1998.647184

47. Ropars, T., Guermouche, A., Uçar, B., Meneses, E., Kalé, L.V., Cappello, F.: On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications. In: Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part I. pp. 567–578 (2011), DOI: 10.1007/978-3-642-23400-2_53

48. Sato, K., Ahn, D.H., Laguna, I., Lee, G.L., Schulz, M., Chambreau, C.M.: Noise injection techniques to expose subtle and unintended message races. In: Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 89–101. PPoPP '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3018743.3018767

49. Sato, K., Ahn, D.H., Laguna, I., Lee, G.L., Schulz, M.: Clock delta compression for scalable order-replay of non-deterministic parallel applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015. pp. 62:1–62:12 (2015), DOI: 10.1145/2807591.2807642

50. Thoai, N., Kranzlmüller, D., Volkert, J.: Shortcut Replay: A Replay Technique for Debugging Long-Running Parallel Programs, pp. 34–46. Springer, Berlin, Heidelberg (2002), DOI: 10.1007/3-540-36184-7_5

51. Utterback, R., Agrawal, K., Lee, I.A., Kulkarni, M.: Processor-oblivious record and replay. In: Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 145–161. PPoPP '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3018743.3018764

52. Wu, X., Mueller, F.: Elastic and Scalable Tracing and Accurate Replay of Non-deterministic Events. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. pp. 59–68. ICS '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2464996.2465001

53. Wu, X., Vijayakumar, K., Mueller, F., Ma, X., Roth, P.C.: Probabilistic Communication and I/O Tracing with Deterministic Replay at Scale. In: International Conference on Parallel Processing, ICPP 2011, Taipei, Taiwan, September 13-16, 2011. pp. 196–205 (2011), DOI: 10.1109/ICPP.2011.50

54. Xue, R., Liu, X., Wu, M., Guo, Z., Chen, W., Zheng, W., Zhang, Z., Voelker, G.: MPIWiz: Subgroup Reproducible Replay of MPI Applications. In: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 251–260. PPoPP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1504176.1504213

55. Zambonelli, F.: Deadlock prevention in incremental replay of message-passing programs. In: Sloot, P., Bubak, M., Hoekstra, A., Hertzberger, B. (eds.) High-Performance Computing and Networking: 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12-14, 1999 Proceedings. pp. 593–602. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/BFb0100620

56. Zambonelli, F., Netzer, R.H.B.: Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999. pp. 392–398 (1999), DOI: 10.1109/IPPS.1999.760506

57. Zhai, J., Chen, W., Zheng, W., Li, K.: Performance Prediction for Large-Scale Parallel Applications Using Representative Replay. IEEE Transactions on Computers 65(7), 2184–2198 (2016), DOI: 10.1109/TC.2015.2479630

DOI: 10.14529/jsfi180103

Survey of Storage Systems for High-Performance Computing

Jakob Lüttgau[1], *Michael Kuhn*[2], *Kira Duwe*[1], *Yevhen Alforov*[1], *Eugen Betke*[1], *Julian Kunkel*[1], *Thomas Ludwig*[1,2]

In current supercomputers, storage is typically provided by parallel distributed file systems for hot data and tape archives for cold data. These file systems are often compatible with local file systems due to their use of the POSIX interface and semantics, which eases development and debugging because applications can easily run both on workstations and supercomputers. There is a wide variety of file systems to choose from, each tuned for different use cases and implementing different optimizations. However, the overall application performance is often held back by I/O bottlenecks due to insufficient performance of file systems or I/O libraries for highly parallel workloads. Performance problems are dealt with using novel storage hardware technologies as well as alternative I/O semantics and interfaces. These approaches have to be integrated into the storage stack seamlessly to make them convenient to use. Upcoming storage systems abandon the traditional POSIX interface and semantics in favor of alternative concepts such as object and key-value storage; moreover, they heavily rely on technologies such as NVM and burst buffers to improve performance. Additional tiers of storage hardware will increase the importance of hierarchical storage management. Many of these changes will be disruptive and require application developers to rethink their approaches to data management and I/O. A thorough understanding of today's storage infrastructures, including their strengths and weaknesses, is crucially important for designing and implementing scalable storage systems suitable for demands of exascale computing.

Keywords: storage hierarchy, file system, storage system, I/O interface, data format.

Introduction

Supercomputers are valuable tools for scientific and industrial users [26]; they allow conducting experiments and generating insight in areas which are otherwise too expensive, too dangerous or impossible with other available technology. Large-scale modeling, simulation and analysis are used to optimize existing technologies, to peek into the future and to understand phenomena where direct means for imaging or observation are missing. Typical workloads in high-performance computing (HPC) include climate simulations, numerical weather prediction as well as computational fluid dynamics and finite element methods in physics, engineering and astrophysics [26]. In biology and chemistry, protein folding and molecular dynamics are especially compute-intensive. With the rise of precision medicine, HPC is also about to become more relevant on an individual level. To solve these tasks, many scientific applications are frequently reading and writing large amounts of data from and to the attached storage systems.

Unfortunately, processor, memory and network technologies advance on divergent trajectories. Clock frequencies did not increase notably for years, and even Moore's law is slowing down as the technology approaches economical and physical limits [45]. Yet, compute capabilities continue to rise dramatically due to massive use of parallelism and distributed computing [93]. Memory and storage technologies, however, have not benefited from comparable advancements so that only a fraction of the computed results can be stored permanently [50]. This mismatch is sometimes referred to as the memory wall, which forces users to decide which information is considered valuable enough for preservation [65]. Besides the memory/storage challenge, policy and practicality demand to limit the next generation of exascale systems to stay within a 20 MW

[1]Deutsches Klimarechenzentrum GmbH, Hamburg, Germany
[2]Universität Hamburg, Hamburg, Germany

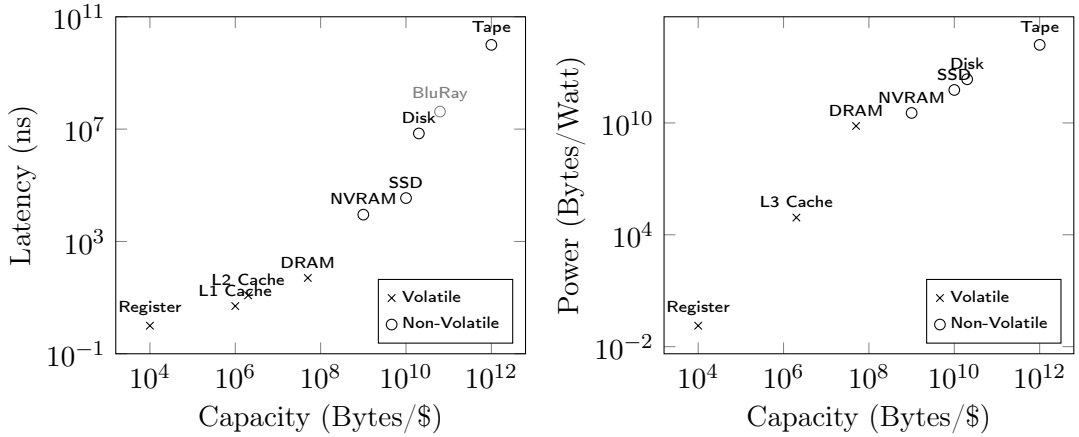


Figure 1. Comparison of capacity, latency and power characteristics for the most important technologies of the memory and storage hierarchies

power envelope. In 2016, the US Department of Energy (DoE) released estimates that data centers accounted for 1.8 % of the country's total electricity usage [85]. While adding additional storage hardware promises the realization of arbitrary aggregated throughput, seeking parity to compute capabilities with currently available technologies would result in data centers dominated by ever larger storage systems, which eventually would exceed power and budget constraints.

The remainder of this paper is structured as follows: Section 1 provides an overview of different memory and storage technologies which form the most basic building blocks for actual storage systems. In Section 2, the most relevant as well as upcoming high-performance storage systems are introduced and characterized for comparison. Storage system semantics and convenient interfaces are discussed in Section 3 due to their impact on user experience and system performance. Section 4 collects the most important future developments which may transform the storage landscape. Finally, the survey is concluded with a short summary.

1. Storage and Memory Technologies

Memory and storage technologies feature widely different performance characteristics, which usually requires finding a trade-off between latency, capacity and cost. Figure 1 illustrates the relationships between latency, capacity, cost and power requirements for the most important volatile and non-volatile memory technologies. Lower latency technologies tend to have lower capacity, lack persistence and are more costly. As a result, memory hierarchies are deployed in combination with caching techniques to provide fast access to hot data that is currently in use. Large amounts of cold data are stored on much more affordable storage media. For data centers, it is often feasible to provide a variety of storage tiers with deep hierarchies. The bulk of data is stored and archived on cheap high capacity but high latency storage media (for example, tape), while data with fast access requirements is staged onto lower latency storage tiers (for example, hard disk drives and solid-state drives). An overview of the different storage technologies including their performance metrics and costs are provided in Tab. 1.

Many research works during the last decade have been devoted to the possible employment of cloud computing technologies as a platform for running HPC applications [19, 105]. Most of these works investigated the performance of chosen application runs on such platforms, including Amazon Web Services, OpenStack etc. [28, 55]. The results showed that only applications that

Table 1. Comparison of memory and storage technologies and performance characteristics [32, 33, 43, 47, 62, 81, 82, 94]

Technology/ Form Factor	Latency	Throughput Read/Write	IOPS	Capacity Unit	Cost ~$/GB	Power ~W/GB	Endurance DWPD	Retention Time
Registers	< 1 ns	-	-	32/64 bits	-	-	∞	hours
L1 Cache	~ 5 ns	-	-	32+32 KB	-	-	∞	hours
L2 Cache	~ 10 ns	-	-	< 1024 KB	-	-	∞	hours
L3 Cache	~ 20 ns	-	-	8–12 MB	-	-	∞	hours
DRAM	~ 80 ns	17/17 GB/s	-	< 64 GiB	5.000	0.1500	∞	~ 5 ms
NVRAM	~ 5 µs	2.5/2.5 GB/s	4.6M	< 480 GB	1.200	0.0300	(~1000)	(~10 y)
SSD (NVMe)	~ 20 µs	8.0/5.0 GB/s	1.2M	< 32 TB	0.200	0.0007	~ 10-25	> 10 y
SSD	~ 100 µs	2.1/2.0 GB/s	0.8M	< 8 TB	0.100	0.0018	~ 10-25	> 10 y
HDD	~ 10 ms	250/240 MB/s	< 500	< 14 TB	0.030	0.0004	-	> 10 y
Tape	> 20 s	315/315 MB/s	-	< 15 TB	0.001	-	-	> 30 y

are not data-intensive are suitable for cloud deployments [27]. There are also projects aiming to develop unified systems that can be leveraged by both HPC and big data worlds [86].

1.1. Tape

For decades, tapes have been the most affordable technology for long-term storage, making it the most common cold storage technology. Unlike other storage technologies, tape is robust to many forms of physical mishandling. As the technology has been used for almost a century, the aging of tape is well understood, featuring reliable data retention times easily exceeding 30 years. Tape media are inexpensive with a price of $ 0.01 per GiB. For the total cost of ownership (TCO), tapes are usually attractive because no energy is required for inactive media. Tapes are expected to feature capacities of up to 200 TB per tape [83]; prototypes with this capacity have been showcased at conferences and in lab conditions by Fujitsu and IBM. With Linear Tape Open (LTO), an industry standard protects investments into tape libraries with a roadmap for the next 6–8 years, with guaranteed backwards compatibility for two LTO generations. About every two years, a new generation of LTO tapes is released to the market, which roughly doubles the capacity and also improves the read/write performance [88]. Typically, the most recent LTO cartridge will provide capacities slightly higher than available hard disk drives (HDDs). For sequential workloads, tape drives often outperform HDDs when comparing read/write throughput. To reduce spool times when reaching the end of the tape, tapes are usually using a linear serpentine layout of tracks. Technologies such as the Linear Tape File System (LTFS) and affordable EEPROMs attached to tapes ensure that tapes are portable, self-describing and easily indexable. This allows to easily migrate large amounts of data from one library to another. Redundant Array of Independent Tape (RAIT) will help tape to cope with simulation output that may exceed the capacity of a single tape and to improve read performance when multiple tapes hold the same data [36]. Special tapes with write-once-read-many (WORM) semantics are available for temper resistant archival as may be required by company policies or government regulation. Tape systems are commonly deployed in combination with a disk-based cache.

1.2. Hard Disk Drives

In terms of gigabytes shipped, HDDs are dominating the storage technology landscape. The market demand, driven by consumers and data centers, provides manufacturers with the necessary economics of scale to produce a high-tech product with delicate mechanics at competitive prices. Until today, the bulk of active data in data centers is provisioned using HDDs. In addition to their price, HDDs provide a very reliable way to store data [48].

HDDs store data on the magnetic coating of rotating platters, sometimes featuring up to eight platters in a 3.5-inch drive. An actuator arm allows positioning the read/write head on different locations of the disk. The polarity of the magnetization is used to encode for individual states of a bit. HDDs feature very high areal data densities while providing mediocre random read/write performance. Most modern HDDs use the so-called perpendicular recording to achieve higher data densities and to prevent data loss due to magnetic instabilities (superparamagnetic limit). Even higher data densities can be achieved using shingled magnetic recording (SMR), but the capacity reduces write performance because it may be necessary to rewrite overlapping magnetic shingles [31]. Helium-filled drives allow increasing the RPM of a drive and enable tighter packing of platters because the lower friction of helium reduces vibration [104]. Decreasing cost for other storage media such as NAND-based solid-state drives (SSDs) limits incentives to further tweak HDDs based storage systems.

1.3. Solid-State Drives

Solid-state drives are commonly used to improve small random access I/O performance in storage systems. Database systems and metadata subsystems of HPC storage systems employ SSDs. Most commercially available SSDs are based on NAND flash memory cells. More recently, there are also SSDs that feature 3D XPoint technologies, which offer improved I/O operations per second (IOPS) but disappoint in terms of throughput performance so far. Both forms of SSDs employ a variety of techniques to maximize lifetime and performance. This includes wear leveling to prevent a drive from failing early because of access patterns that would quickly degrade specific cells. To meet warranty guarantees, most SDDs use over-provisioning internally and remap faulty cells transparently.

Many metrics for SSDs' endurance rating are available; they are not standardized, however, and often rely on an opaque workload chosen by the specific vendor. Terabytes written (TBW) and drive writes per day (DWPD) have turned out to be among the most transparent ones because they allow users to compare against their own assumed workload. SSDs are most commonly available as 2.5-inch SAS/SATA drives. For highest performance, PCIe cards using NVM Express (NVMe) are also available, but usually at a higher cost.

New manufacturing techniques that allow arranging NAND in 3D promise larger capacity SDDs, likely resulting in density advantages over HDDs [67]. While this adds access latency due to another indirection, higher overall throughput for large block I/O is expected. Current architectures will not be able to fully exploit 3D NAND due to a lack of wider buses.

1.4. Non-Volatile Memory

Non-volatile random-access memory (NVRAM) aims to provide similar latency characteristics as DRAM or SRAM while retaining data when powered off. While there exist a variety of candidates

for NVRAM (for example, PCM, MRAM, FeRAM, ReRAM, Flash Memory), commercial products that would render DRAM or SRAM obsolete are not available as of 2018.

1.4.1. Flash Memory

Flash memory is the most widespread form of NVRAM, but the performance characteristics and especially memory wear justify research in alternative technologies. Two variants of flash memory cells (NAND and NOR) are available, and both exhibit memory wear which eventually leads to the failure of the memory cells. NOR-based flash memory can provide byte addressable random access to data, but experiences long erase and write times. NAND is far more common, as it features higher data densities and provides block-level access semantics. Flash memory can either store individual bits using single-level cells (SLC), or multiple bits using multi-level cells (MLC). Single-level cells can provide higher performance, better endurance and longer data retention times. MLC provides higher data densities and therefore achieves higher capacities at lower cost, but with reduced performance and less endurance.

1.4.2. Phase-Change Memory

As of 2016, NVRAM based on phase-change memory (PCM), called 3D XPoint, is marketed by Intel and Micron. Intel distributes the technology in the Optane product line of SSDs, which provides in the order of 1,000,000 IOPS and features better write endurance than NAND-based SSDs. Performance of read/write throughput is on par with NAND-based SSDs at this time.

1.5. Volatile Memory

Besides non-volatile memory technologies for permanent data storage, volatile technologies such as dynamic random-access memory (DRAM) and static random-access memory (SRAM) are commonly used to accelerate access to data that is currently in use or anticipated to be used soon. A computer's main memory is often used to speed up file access by providing page/disk caches. On the one hand, they can be used to speculatively read more blocks of a file than have been requested by an application, which is called read-ahead. On the other hand, they are used for slightly delayed write operations in order to avoid latency penalties when performing many small I/O requests. Some database systems load all data into volatile main memory to provide optimal query performance, while changes to the data are logged and applied asynchronously to persistent copies of the data.

1.5.1. SRAM

Modern CPUs provide low latency access to about 8–12 MB of SRAM-based memory arranged in multiple cache levels. In multi-core architectures, L1 caches are usually exclusive to a single compute core, while L2 and higher cache levels are usually shared between multiple cores. The cache coherence semantics can vary for different architectures. SRAM retains data up to a few hours, which makes it very power efficient when idle. When SRAM is accessed frequently, power consumption can be higher than DRAM.

Table 2. NVM characteristics [4]

Property	SRAM	DRAM	HDD	NAND	STT-RAM	ReRAM	PCM	FeRAM
Cell size (F^2)	120 - 200	60 - 100	N/A	4 - 6	6 - 50	4 - 10	4 - 12	6 - 40
Write Endurance	10^{16}	$> 10^{15}$	$> 10^{15}$	10^4 - 10^5	10^{12} - 10^{15}	10^8 - 10^{11}	10^8 - 10^9	10^{14} - 10^{15}
Read Latency	~ 0.2 - $2ns$	$10ns$	3 - $5ms$	15 - $35\mu s$	2 - $35ns$	$\sim 10ns$	20 - $60ns$	20 - $80ns$
Write Latency	~ 0.2 - $2ns$	$10ns$	3 - $5ms$	200 - $500\mu s$	3 - $50ns$	$\sim 50ns$	20 - $150ns$	50 - $75ns$
Leakage Power	High	Medium	(mech.)	Low	Low	Low	Low	Low
Energy (R/W)	Low	Medium	(mech.)	Low	Low/High	Low/High	Medium/High	Low/High
Maturity	Mature	Mature	Mature	Mature	Test chips	Test chips	Test chips	Manufactured

1.5.2. DRAM

Because SRAM is relatively expensive to produce due to its structural complexity as well as its density disadvantage in comparison to DRAM, DRAM is used for computer main memory instead of SRAM. The downside of DRAM is a relatively high power consumption because DRAM-based memory cells encode data as a charge in a capacitor that needs to be refreshed regularly to prevent data loss. In combination with an uninterrupted power supply to ensure operation long enough to drain data to a non-volatile memory technology, DRAM is sometimes treated just like a non-volatile class of high performance storage.

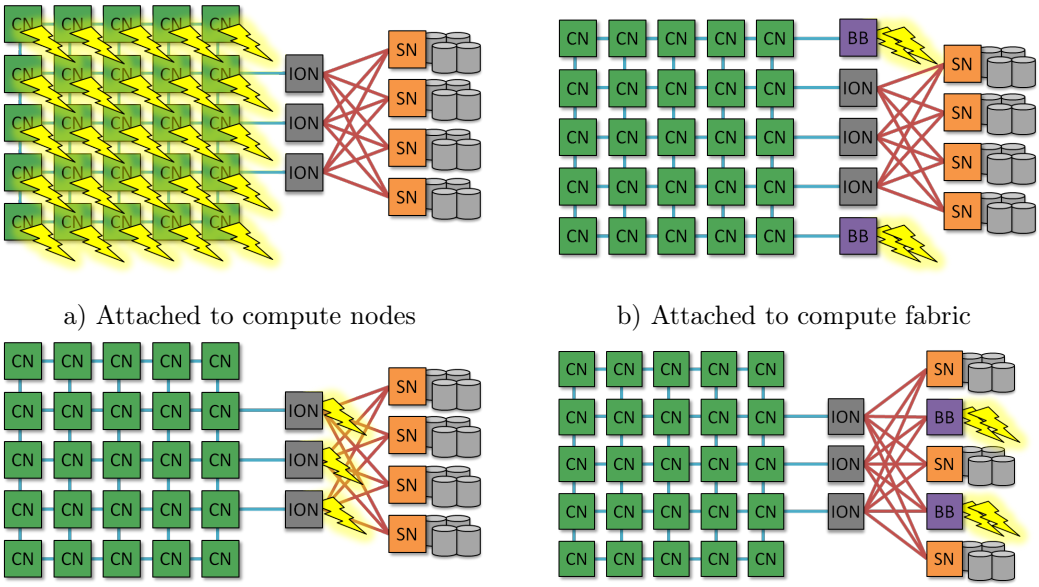
1.6. Accelerators

As stated in [4], the upcoming non-volatile memory will probably revolutionize the memory stack. The technologies will be used to add new memory layers (vertical integration) and to support existing technologies (horizontal integration). The most promising technologies are PCM (phase-change memory), MRAM (magneto-resistive RAM), FeRAM (ferroelectric RAM) and ReRAM (resistive RAM). As shown in Tab. 2, the read/write access to these technologies is getting closer to RAM, which offers corresponding opportunities. In fact, many accelerators use the faster technology as a kind of cache for inefficient I/O to the slower storage technology. There is no ultimate solution so far, so we will describe some of the most promising approaches.

Burst Buffers. Flash-based acceleration hardware can be integrated at different locations in the system [75]. Compute nodes can be equipped with local flash storage. Another possibility is to incorporate dedicated nodes, also referred to as burst buffers. Furthermore, buffers can be built into the storage system itself. These three variants are illustrated in Fig. 2.

A burst buffer acts as a fast write-behind cache that transparently migrates data from the burst buffer's fast storage to a traditional parallel file system. Typically, burst buffers rely on flash or NVM to support random I/O workloads that HDD-based file systems struggle with. For flash-based SSDs, many vendors offer high-performance storage solutions: DDN's Infinite Memory Engine (IME) [12], IBM's FlashSystem [37] and Cray's DataWarp accelerator [11] are popular examples. Using comprehensive strategies to utilize flash chips concurrently, these solutions are powerful and robust to guarantee availability and durability of data for many years.

Hybrid Flash Arrays. Even though burst buffers provide the possibilities to increase the system's performance dramatically, the hardware's potential can often not be fully exploited due to transfer limitations. As a result, they are used at their full speed only a fraction of the time while imposing a great part in the overall costs. Therefore, more efficient solutions are required.



a) Attached to compute nodes

b) Attached to compute fabric

c) Attached to I/O nodes

d) Attached to storage fabric

Figure 2. An overview of different burst buffer architectures [57] (CN = compute node, ION = I/O node, SN = storage node)

Seagate's Nytro Intelligent I/O Manager (NytroXD) is a hybrid flash array consisting of a small amount of flash combined with HDDs to work within Lustre and Spectrum Scale [75]. It acts as a block device directing small I/O accesses to flash while the large I/O calls are passed to the HDDs. With this approach temporal and spatial fragmentation can be reduced. Especially mixed I/O workloads and I/O of small block sizes profit from NytroXD.

Other Hybrid Caching Approaches. In order to find a trade-off between performance as well as acquisition and maintenance costs, various hybrid systems have been proposed consisting of NVM/NVRAM and SSDs. OTF-AST (on-the-fly automated storage tiering) consists of byte-accessible NVDIMM (non-volatile dual inline memory module) and SSDs and investigates the potential benefits of forwarding the problematic I/O accesses to the NVRAM, while the rest is passed directly to the main storage [70]. The results show that although the average response time of I/O accesses is decreased, the migration algorithm needs to be adapted as the migration overhead is considerably smaller between DIMM and SSDs than between SSDs and HDDs. A different approach is to determine automatically whether a given memory area can benefit from DRAM characteristics by using a profiling tool [18]. Furthermore, NOVA is a file system aiming to increase the performance and offering strong consistency guarantees at the same time through additional metadata structures held in DRAM, accelerating the lookup. Additionally, the data and metadata is stored in logs on NVM, providing atomicity and fast concurrent access in a random manner [102].

Memory Management. Further acceleration approaches are taken in the area of storage allocation in order to optimize the system's utilization dynamically and thereby increase the I/O throughput. DynIMS is a dynamic memory controller, developed to speed up HPC applications that use Spark [103]. With DynIMS, an improvement of a factor of five can be achieved in contrast

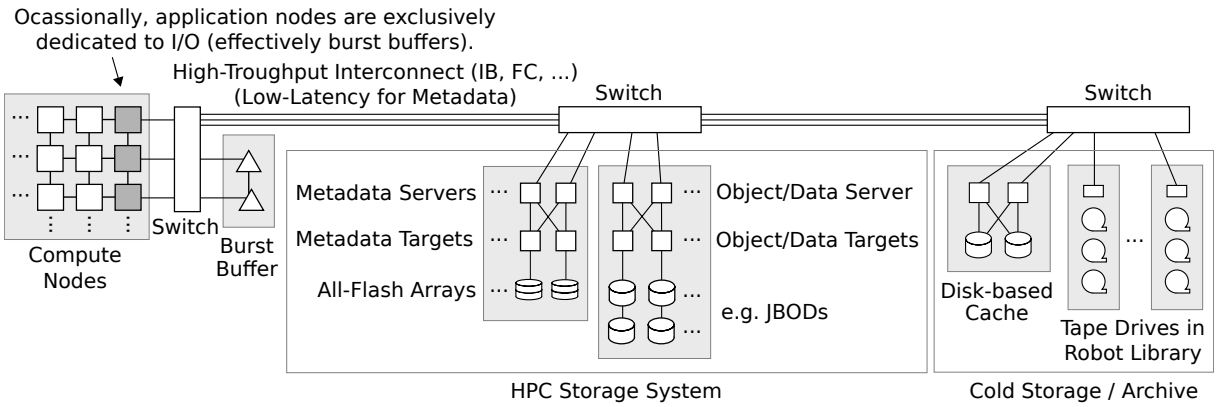


Figure 3. A data center with a typical HPC storage system and an archive for cold data; the network is only an approximation for better overview and may differ in a real deployment

to static allocation. Future efforts are targeted at adjusted cache management for file systems like Lustre and Spectrum Scale.

Metadata Accelerator. FSMAC is a metadata accelerator that exploits byte-addressable NVM technology [9, 99]. Metadata is quite small, typically smaller than a block of a file system, which makes access to metadata inefficient. FSMAC separates metadata and data, forwarding data to the storage and metadata to the byte-addressable NVM. For synchronous I/O, this approach achieved a speedup factor of 49.2, while a speedup of 7.22 was reached for asynchronous I/O.

In-Memory Storage. Network-attached in-memory storage promises to provide optimal performance that exceeds SSD-based solutions. A benefit is high performance predictability and low variance. The usage of DRAM for storing intermediate data is not new, and RAM drives have been used in operating systems for decades. However, offered RAM storage was used as temporary local storage and not durable and usually not accessible from remote nodes. Exporting RAM storage via parallel file systems was used mainly for performance evaluation but without durability guarantees. BurstMem provides a burst buffer with write-behind capabilities by extending Memcached [97]. Experiments show that the ingress performance grows up to 100 GB/s with 128 BurstMem servers.

The Kove XPD is a robust scale-out pooled memory solution that allows aggregating multiple InfiniBand links and devices into one big virtual address space that can be dynamically partitioned [49]. Internally, this remote memory is asynchronously backed up to SATA RAID embedded in the XPDs and, thus, together with an UPS can be considered to be non-volatile. The system offers various APIs to access this memory such as treating it as a block device. The XPDs can also be used to speed up MPI-IO accesses [51]: three Kove XPDs delivered a throughput of up to 60 GiB/s. When increasing the number of clients or servers, a throughput close to the available network bandwidth can be achieved already with 100 KiB random accesses.

2. Storage and File Systems

Providing reliable, efficient and easy to use storage and file systems is one of the main issues today in HPC because a wide variety of scientific applications produces and analyzes enormous volumes of data. File systems offer an interface to the underlying storage device and link an identifier such as the file name to the corresponding physical addresses of the storage hardware.

Thereby, a more comfortable and simplified use of storage devices is enabled. Traditionally, they realize the concept of hierarchical structuring through the use of directories and files. Besides the actual file content, metadata such as the file size and access times are managed. Over the years, several file systems have been proposed and established, offering a wide range of functionality. Especially in HPC systems, parallel distributed file systems are deployed, allowing to spread data across numerous storage devices and combining the particular features to increase the throughput as well as the system's capacity. However, due to the rapidly increasing data sizes, more sophisticated and specialized approaches are required for handling the enormous amount of information. At the same time, new and more powerful storage and network technologies are developed, posing challenges to exploit the respective capabilities. Besides the old file system concepts, other approaches have found their way into HPC systems. Figure 3 gives an overview of an archetypal HPC system with a number of different storage systems and technologies.

Hence, the need for high-throughput concurrent read and write capabilities of HPC applications led to development of parallel and distributed file systems. In this section we discuss the most popular and widely used systems and their characteristics. Among them are Lustre, Spectrum Scale, BeeGFS, OrangeFS, Ceph and GlusterFS.

2.1. Spectrum Scale

Spectrum Scale is a scalable high-performance data management solution developed by IBM for enterprises that need to process and store massive amounts of unstructured data [39]; it is based on the former General Parallel File System (GPFS) [38]. The parallel file system provides concurrent data access with the ability to perform data analysis and archiving at one place. Spectrum Scale unifies different storage tiers like SSDs, HDDs and tapes, as well as analytics into a single scale-out solution. This enables users to choose optimal storage for their files or object data and move them as quickly as possible with low costs. Spectrum Scale is fully POSIX-compliant, which allows it to support many traditional HPC applications.

The file system helps to avoid a performance bottleneck for metadata-intensive applications by configuring dedicated servers for metadata updates. Otherwise, data can be mixed together with metadata. Another bottleneck regarding single-server performance is also avoided in Spectrum Scale as all servers and clients can access and share the data without moving it. Thus, even a client can play the role of a server.

Even though Spectrum Scale is a very capable file system and has a great support by IBM with the integration of various useful tools, it is still quite expensive especially for non-profit clusters with research purposes.

2.2. Lustre

Lustre is a parallel distributed file system that is used on supercomputers. It is licensed under the GNU General Public License (GPLv2) and can be extended and improved. Because of its high performance, Lustre is used on more than half of the 100 fastest supercomputers in the world.

The file system's architecture distinguishes between clients and servers. Clients use RPC messages to communicate with the servers, which perform the actual I/O operations. While all clients are identical, the servers can have different roles: Object Storage Servers (OSS) manage the file system's data in the form of objects; clients can access byte ranges within the objects. Metadata Servers (MDS) manage the file system's metadata; after retrieving the metadata, clients

are able to independently contact the appropriate OSSs. Each server is connected to possibly multiple targets (OSTs/MDTs) that store the actual file data or metadata, respectively.

Lustre runs in kernel space, that is, most functionality has been implemented in the form of kernel modules, which has advantages and disadvantages. On the one hand, by using the kernel's virtual file system (VFS) Lustre can provide a POSIX-compliant file system that is compatible with existing applications. On the other hand, each file system operation requires a system call, which can be expensive when dealing with high-performance network and storage devices.

In line with its open approach to Lustre development, Intel has funded five Intel Parallel Computing Centers to integrate new features into Lustre. Among others, these centers are working on quality of service for I/O performance, file system compression, as well as better integration of TSM storage backends and big data workflows.

2.3. BeeGFS

The parallel and POSIX-compliant cluster file system BeeGFS was developed for I/O-intensive HPC applications [21]. Its architecture has a client-server design and consists of three key components: clients, metadata servers and storage servers. The scalability and flexibility of BeeGFS can be reached simply by increasing the number of servers and disks required for specific users. All their data is transparently distributed across multiple servers using striping (chunk by chunk of a given size). Besides data distribution, metadata is also striped over several metadata servers on a directory level, with each server storing a part of the complete file system tree. In this way, fast access to the data is provided. BeeGFS enables load balancing for metadata as well.

The client kernel module of the BeeGFS system is free and under the GPL license, the server is covered by the BeeGFS EULA. Hence, commercial support is optionally available. In the future, developers of BeeGFS aim to improve tools for monitoring and diagnostics, as well as extend the POSIX interface support.

2.4. User-Level File Systems

In contrast to kernel space file systems such as Lustre, user-level file systems do not require any kernel modules to run. This typically makes it easier to use such file systems in a supercomputer environment, where users typically do not have root privileges.

OrangeFS is a parallel distributed file system that runs completely in user space. It is open source and licensed under the GNU Lesser General Public License (LGPL). It provides excellent MPI-IO support through a native ADIO backend and provides a wide range of user interfaces, including several Direct Interface libraries, a FUSE file system and an optional kernel module [96]. Similar to other file systems, OrangeFS has dedicated servers for data and metadata storage. OrangeFS uses arbitrary local POSIX file systems for data and can use either Berkeley DB (BDB) or Lightning Memory-Mapped Database (LMDB) for metadata.

IBIO is a user-level InfiniBand-based file system, and is designed as an intermediate layer between compute nodes and parallel file system [80]. It aims to improve performance for checkpoint/restart use cases, and they discuss redundancy schemes to increase reliability. With SSDs and FDR Infiniband, they achieve on one server a throughput of 2 GB/s and 3 GB/s for write and read, respectively.

Table 3. Performance of different file systems on the IO-500 list (ordered by node count)

System	FS	Nodes	IOR in GiB/s				MDTest in kIOP/s					
			Easy		Hard		Easy	Hard	Easy	Hard	Easy	Hard
			Write	Read	Write	Read	Create		Stat		Delete	
Oakforest	IME	2048	740	430	600	260	28	2	54	62	36	1
Shaheen	Lustre	1000	330	220	1.4	81	13	14	120	130	15	11
Shaheen	DataWarp	300	970	900	16	39	51	11	49	39	49	19
Mistral	Lustre	100	160	160	1.5	7	18	18	150	160	8	9
Seislab	BeeGFS	24	19	22	0.9	2	100	5	430	57	170	14
Sonasad	S.Scale	10	34	32	0.2	2	57	22	340	530	48	85

GlusterFS is another one POSIX-compliant, free and open-source distributed file system [14]. Like other traditional storage solutions, it has a client-server model but does not need a dedicated metadata server. All data and metadata are stored on several devices (called volumes), which are dedicated to different servers. GlusterFS locates files algorithmically using an elastic hashing algorithm. This no-metadata server architecture ensures better performance, linear scalability and reliability. At the same time, GlusterFS is a network file system that provides file-based storage only; block and object interfaces must be built on top of it.

2.5. File System Comparison

In the IO-500[3], the performance behavior for data and metadata benchmarks of different file systems is listed. Similar to the TOP500 list for computing architectures, IO-500 aims to track performance growth over the years and analyze changes in the storage landscape. The IO-500 does not only provide key metrics of expected performance but serves as a repository for fostering and sharing best practices within the community. The benchmark methodology harnesses the MDTest and the IOR benchmarks. A collection of hard tests with preconfigured parameters are designed to show a worst-case scenario of unoptimized applications. For IOR, this means random I/O of 47,000 chunks, and for MDTest a single shared directory with files of 3,901 bytes. A set of easy tests are configurable and optimizable by the user and aim to show the potential of the storage system tested. For IOR, easy tests typically are sequential I/O of large chunks, and for MDTest empty files are used.

An excerpt from the Nov. 2017 list (rounded) is shown in Tab. 3. It shows that IME excels at IOR hard (random) performance, but the metadata performance is worse than that of Lustre. Data Warp on Shaheen improves the throughput of sequential I/O, but random I/O does not benefit much. The small configurations tested of BeeGFS and Spectrum Scale do not allow to make conclusions on the throughput. However, for metadata, BeeGFS and a recent version of Spectrum Scale shine compared to Lustre and IME.

2.6. HPSS

In the early 1990s, national laboratories of the Department of Energy and IBM recognized there was an enormous challenge ahead in order to manage the exponential growth of data [98]. They developed the High Performance Storage System (HPSS) to provide a scalable hierarchical storage

[3] `https://www.io500.org/`

system that meets the requirements to handle future hardware generations. The architecture's main focus lies in hierarchical storage management (HSM) and data archiving. It is a widespread solution in today's storage systems, mainly used for the management of tape archives. The total managed data volume equals 2.2 EB only for scientific data [34].

2.7. ECFS and MARS

Other storage systems focused on data archiving developed by the European Centre for Medium-Range Weather Forecasts (ECMWF) are ECMWF's File Storage System (ECFS) and Meteorological Archival and Retrieval System (MARS) [25]. An HPSS manages the tape archives for both systems as well as the disk cache for ECFS where the files are accessed using a unique path. MARS is an object store providing an interface similar to a database. By using queries in a custom language, a list of relevant fields can be set which are then joined into a package and stored in the system. The field database (FDB) stages and caches fields which are often accessed. ECFS contains relatively few files which are used concurrently and experiences mainly write calls. In MARS, however, the files are equally relevant and mostly read [87]. Thus, both systems provide powerful storage management for researchers interested in weather modeling. MARS allows HPC users to access huge amounts of meteorological data stored only in GRIB and BUFR formats, collected over the last 30 years.

2.8. Ceph

Ceph is a free and open-source platform that offers file-, block- and object-based data storing on a single distributed cluster [100]. The system implements distributed object-storage on a base of Reliable Autonomic Distributed Object Store (RADOS) system [101]. It is responsible for data migration, replication, failure detection, and failure recovery to the cluster. Integration of the near-POSIX-compliant CephFS file system allows many applications to utilize the benefits and capabilities of the scalable environment. Ceph makes use of intelligent Object Storage Devices (OSDs). These units provide file I/O (reads and writes) for all clients which interact with them. Data and metadata are decoupled because all the operations for metadata altering are performed by Metadata Servers (MDSs). Ceph dynamically distributes the metadata management and responsibility for the file system directory hierarchy among tens or even hundreds of those MDSs.

Ceph, however, still has some drawbacks. Among them is the limitation of only being able to deploy one CephFS per cluster and the current test phase of reliability on real-world use-cases. Some features and utilities are still in an experimental phase as well. For instance, usage of snapshots could cause client nodes or MDSs to terminate unexpectedly. In addition, Ceph is designed with HDDs as its basis and needs improvements in performance when disks are replaced with SSDs, and data access pattern is random [71].

3. Interfaces and Data Formats

Interfaces play an important role in using file and storage system, especially in the HPC context. On the one hand, interfaces should be convenient to use, so that developers can focus on the applications' functionality instead of the I/O interface. On the other hand, they should be able to deliver high performance by supporting parallel access. Moreover, being able to easily exchange data is of fundamental importance in research. Each interface typically supports one or

more data formats that can be accessed using it. Data formats also influence how fast data can be accessed and how exchangeable it is.

3.1. POSIX

The POSIX I/O interface's first formal specification dates back to 1988 when it was included in POSIX.1. Later, specifications for asynchronous and synchronous I/O were added in POSIX.1b from 1993 [40]. Even though it was designed primarily for local file systems, POSIX is widely used, even in parallel distributed file systems, and thus provides excellent portability.

Due to its focus on local file systems and portability, POSIX features very strict consistency and coherence requirements. For instance, `write` operations have to be visible to other clients immediately after the system call returns. These strict requirements pose a serious bottleneck in parallel distributed file systems as they require coordination and synchronization of all clients [58]. Moreover, I/O is intended to be atomic but not strictly required to be so.

Additionally, POSIX files are opaque byte streams and, therefore, applications are not able to inform the file system about data structures that might be used for more intelligent I/O and data placement decisions.

The effort for POSIX HPC I/O extensions aimed to address some of POSIX's limitations by introducing functionality for group open, non-contiguous read/write and optimizations for metadata performance [46, 95]. However, none of the extensions were integrated into any major file system, requiring applications to use the traditional interface.

3.2. MPI-IO

In contrast to the POSIX interface, MPI-IO has been designed from the ground up for parallel I/O. It was introduced in the MPI standard's version 2.0 in 1997 [66] and defines I/O operations in an analogous fashion to MPI's established message passing operations. MPI-IO represents an I/O middleware that abstracts from the actual underlying file system and thus offers portability for parallel applications. For instance, the ADIO layer of the popular MPI-IO implementation ROMIO includes support and optimizations for POSIX, NFS, OrangeFS and many other file systems [35]. MPI-IO's interface is element-oriented and supports MPI datatypes to access data within files. The actual I/O functions look and behave very similar to their POSIX counterparts [84].

MPI-IO's semantics differ drastically from POSIX's. Specifically, its consistency requirements are less strict than those defined by POSIX [10, 89]. Non-overlapping or non-concurrent write operations are handled correctly when using MPI-IO's default semantics. In contrast to POSIX's immediate visibility of changes for all participating clients, MPI-IO requires changes to be visible immediately only to the writing process itself. Other processes first have to synchronize their view of the file using `MPI_File_sync` and `MPI_Barrier`.

Moreover, MPI-IO offers an atomic mode for workloads that require stricter semantics. The mode can be enabled (and disabled) at runtime using `MPI_File_set_atomicity`. It allows concurrent and conflicting writes to be handled correctly and also makes changes visible to all processes within the same communicator without explicitly synchronizing them. However, the atomic mode can be difficult to support [53, 77].

3.3. HDF and NetCDF

As described above, many I/O interfaces only feature access based ones on bytes or elements. Additionally, they do not encode the file structure within the files themselves, requiring a priori knowledge to be able to access existing files. HDF and NetCDF are two popular interfaces for working with self-describing data. Self-describing data formats contain all necessary information to be able to access files even if their structure is not known beforehand. This allows effortless exchange of data between scientists.

The Hierarchical Data Format (HDF) consists of a set of file formats and libraries for accessing self-describing collections of data. It is used in many scientific applications [30]. HDF5 supports two major types of data structures: datasets and groups, which are similar to files and directories in traditional file systems. Datasets are used to store typed data, while groups are used to structure the namespace. Groups can contain datasets as well as groups, which leads to a hierarchical layout. Datasets are typed and can store multi-dimensional arrays of several different data types. Another similarity to file systems is how objects are accessed: they use POSIX-like paths such as `/path/to/dataset`. Moreover, datasets and groups can be associated with additional metadata using user-defined attributes. This can be used to store arbitrary information together with the actual data. HDF5 uses existing I/O interfaces such as POSIX and MPI-IO to perform the actual I/O. When using the MPI-IO backend, parallel I/O can be performed from multiple clients into a shared HDF5 file.

The Network Common Data Format (NetCDF) also consists of a set of libraries and self-describing data formats [68]. It is mainly used in scientific applications, especially in the fields of climatology, meteorology and oceanography [76]. NetCDF's current version 4 uses HDF5 underneath but reduces the number of supported features for more convenient use. As is the case with HDF5, NetCDF-4 supports parallel I/O.

Unfortunately, over time these libraries accumulate legacy. For example, in HDF5 this becomes apparent in optimizations designed for systems that differ considerably from today's parallel file systems. Because HDF5 in the past could not and today does not pass on logical information about the structure of the data to lower levels, there is no way to account for it. Eventually, well-intentioned heuristics distributed across different layers begin to impede each other.

3.4. ADIOS

The Adaptable IO System (ADIOS) has been designed to solve some of the problems that come with the approaches discussed above. The basic objective is to provide a library of tuned I/O routines to serve as a middleware on a wide range of hardware [56, 61]. Often, applications are highly optimized for specific system architectures to increase performance. As scientific code has a long lifetime, it is executed on several generations of supercomputers. Therefore, changes are required to fully exploit the respective system's capabilities. ADIOS offers the possibility to perform I/O using an abstract definition of the application's data structures in an XML file. This definition is then used to automatically generate code to perform the actual I/O, which allows decoupling the I/O portion of the remaining application code. Once the old I/O calls are replaced by the automatically generated code, there is no need for future recompiling as the implementation of the I/O behavior is no longer in the application. The actual I/O functionality is realized by so-called engines acting as different backends for several self-describing data formats such

as ADIOS's own bp (binary packed) format, HDF5, and NetCDF. Moreover, ADIOS supports advanced functionality such as on-the-fly data transformations [5].

3.5. SIONlib

SIONlib provides scalable access to task-local files [22]. This is achieved by internally mapping all accesses to a single or small number of physical files and aligning them to the file system's block size. By supporting the common `fread` and `fwrite` interfaces, SIONlib minimizes the amount of changes necessary to applications. SIONlib's approach is required to overcome shortcomings in current file systems. Due to metadata performance bottlenecks, file systems often cannot deal well with large numbers of files. Moreover, because of the strong consistency semantics described above, shared file performance is often dramatically degraded when performing unaligned I/O to a shared file. By intelligently managing the number of underlying physical files and transparently aligning the data, SIONlib can alleviate these problems.

3.6. Domain-Specific Approaches

In addition to the previously mentioned generic I/O libraries, there are a multitude of domain-specific I/O and data management libraries available. For instance, the Gridded Binary (GRIB) format is widely used in meteorology to store weather and climate data [24]. The Climate Data Interface (CDI) provides a convenient interface for climate-related data and supports multiple data formats, including GRIB and NetCDF [8]. PIO is an application-level I/O library for earth system models and supports multiple backends such as MPI-IO and various versions of NetCDF [13]. Lemon is a library for parallel I/O mainly used in high-energy physics [15]; it enables efficient I/O of both binary data and associated metadata in the SciDAC Lattice QCD Interchange Message Encapsulation format.

3.7. Conclusion and Comparison

All production-level file systems currently in use offer a POSIX I/O interface that treats file data as an opaque byte stream. As it is not possible to reconstruct the data format from this byte stream, self-describing data formats such as NetCDF and ADIOS are widely used to be able to exchange data with other researchers and annotate data with meaningful metadata.

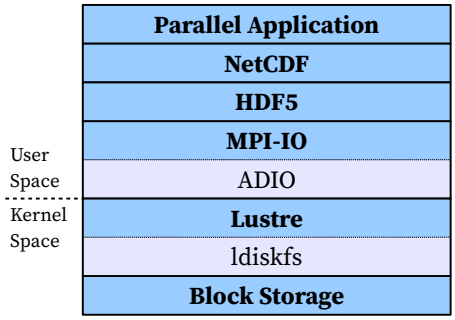


Figure 4. Exemplary HPC I/O stack

Figure 4 illustrates a typical HPC I/O stack. Applications only interface directly with NetCDF, which depends on HDF5 and so on. The coupling between the different layers is loose and mainly used for performance tuning. Structural information about the data is lost as it is handed down

through the layers: while an application might pass multi-dimensional matrices of numerical data to NetCDF, MPI-IO is only aware of a stream of elements, and Lustre's POSIX interface handles file data as raw bytes. A comparison of different I/O interfaces, including their ease of use, supported data formats, exchangeability of data and supported languages, is shown in Tab. 4.

Table 4. Comparison of I/O interfaces used in HPC
(Conv. = Convenience, Exch. = Exchangeability)

Interface	Conv.	Formats	Exch.	Languages
POSIX	Low	Raw	✗	C, C++, Fortran and more
MPI-IO	Low	Raw	✗	C, C++, Fortran, Java, Python and more
HDF	Medium	HDF4, HDF5	✓	C, C++, Fortran, Java and more
NetCDF	Medium	NetCDF, HDF5	✓	C, C++, Fortran, Java, R and more
ADIOS	High	bp, bp2, HDF5, NetCDF-4	✓	C, C++, Fortran
SIONlib	Low	Raw	✗	C, C++, Fortran
GRIB	Medium	GRIB	✓[4]	C, C++, Fortran, Python
CDI	Medium	GRIB, NetCDF and more	✓[5]	C, C++, Fortran
PIO	Medium	Raw, NetCDF	✓[5]	C, C++, Fortran
Lemon	Medium	SciDAC LIME	✗	C, C++

4. Future Developments

This section collects current efforts and anticipated developments for storage systems and technologies. We identify five main factors/areas. Each is summarized in a dedicated subsection, but they should not be seen as independent of each other: 1) future applications and system requirements, as well as market effects; 2) co-design efforts; 3) new and advanced technologies; 4) alternative storage architectures; 5) future interfaces.

4.1. Future Applications and Market Effects

Exascale applications, big data analytics and machine learning are already anticipated workloads. It seems reasonable to expect an increase in diversity and even less predictable access patterns than before. Exascale simulations require storage systems that will be able to serve tens of thousands of nodes [23]. Larger node counts are expected to introduce higher error rates, which results in the deployment of fault-tolerance mechanism and also incurs stress onto storage systems [17, 59]. Large-scale observation systems (for example, in astrophysics) as well as small-scale data loggers (for example, Internet of Things) will require storage systems that can consume and store data at unprecedented scale, ideally with in-transit data processing capabilities. In many cases, scientific workloads are lacking market relevance, and are thus not a priority for many vendors. Storage products offered by vendors are more likely to address the demands of cloud providers and business-driven big data analytics and machine learning. At the same time, the commoditization of technologies used in consumer electronics as well as cloud services influences on which technologies will be considered for the design of next-generation storage systems.

[4] Only individual records are self-describing.
[5] Depends on the chosen data format.

4.2. Co-Design

Incremental changes to existing solutions appear to be insufficient to address the challenges ahead, which is why co-design efforts increasingly include all stakeholders and technical layers. A major driver in HPC innovation is the Department of Energy, which focuses on two approaches to co-design: 1) application-centric and 2) architecture-centric co-design [1]. As exascale systems are approaching, and the storage problematic intensifies, many efforts (including ECP [16], FastForward [44], ADIOS [72], HDF VOL [29], ESiWACE [20], NEXTGenIO [69] and SAGE [78]) are working on the modernization of how applications and libraries down to the storage hardware handle I/O. Co-design can yield highly optimized solutions for special use cases but is not affordable for many smaller communities within HPC.

4.3. Technologies

Section 1 was focusing on technologies and products that are already and widely deployed in data centers. This section will summarize some upcoming changes to the existing technologies but also the expected impact of more speculative technologies that require more research before they find their way into data centers. An important trend is the addition of wider buses and asynchronous protocols for data movement in the form of NVMe and also the support for high-bandwidth memory (HBM) [74]. HBM requires architecture changes, which are not backwards compatible with older hardware, but will bring significant benefit to bandwidth-bound applications [73]. With 3D NAND, the capacity of SSDs will improve further, which might eventually replace HDDs in data centers [41]. NVRAM will likely have a considerable impact on the storage landscape, but most of the known candidates are not ready for mass production or remain too expensive to replace other technologies [4]. Many data transformations such as compression or encryption could be performed out-of-core, either on GPUs or in-transit [2, 42].

For long term-storage and cold data, tape seems likely to remain competitive [83]. One promising alternative, especially for WORM data, could be holographic storage which features high densities and slightly more favorable access semantics than tape [79]. DNA may be interesting as a data storage medium for being durable, while also featuring very high volumetric densities. Applications have already been explored, but technology is currently not feasible due to the state of DNA synthesis and sequencing techniques [3].

4.3.1. Open-Channel SSDs

A new class of SSDs which improves and optimizes the performance of traditional SSDs has been recently introduced as Open-Channel SSDs [52]. They allow splitting the internal capacity of a disk into a number of I/O channels for making the parallel data access faster and maximizing the overall read/write bandwidth. This is a software-defined hardware because the management of parallel units at Open-Channel SSDs is moved from the embedded processor within the device to the hosting file system. It is possible to reduce and predict latency by intelligently controlling I/O submissions. In addition, data placement and I/O scheduling are provided trough NVM management as a block device either at the application level or at the hosting file system.

4.4. Storage and File Systems

There is a wide variety of new and upcoming approaches for file and storage systems [7]. Their optimization and improvement is highly required due to the challenges regarding managing the vast amount of data from I/O-intensive applications. The HPC community aims to relax the strict POSIX semantics without losing the support for legacy applications. Leveraging cloud computing features and its advantages is a new promising trend today. In this section, we will provide an overview of some of the most important ones and highlight the impact they will have on applications and developers.

DAOS. The Fast Forward Storage and IO (FFSIO) project aims at providing an exascale storage system which is capable of dealing with the requirements of HPC applications, as well as big data type workloads. It aims at introducing a new I/O stack and supporting more complex basic data types like containers and key-arrays [60]. Its functionality ranges from a general I/O interface at the top over an I/O forwarding and an I/O dispatcher layer to the Distributed Application Object Store (DAOS) layer, which offers a persistent storage interface and translates the object model visible to the user to the demands of the underlying infrastructure. DAOS will require large quantities of NVRAM and NVMe devices and will therefore not be suitable for all environments. Specifically, the high prices for these relatively new technologies will limit its use both in data centers and especially research, at least in the near future.

PPFS. Post-Petascale File System (PPFS) based on object storage using OpenNVM and targets to converge both HPC and cloud computing [90, 91]. This system uses a key-value store for metadata storage and non-blocking distributed transactions to update multiple entries at the same time. In this way, the offered platform achieves high performance and avoids POSIX compliance.

SoMeta. Scalable Object-Centric Metadata Management (SoMeta) is intended for future object-centric storage systems, providing the corresponding metadata infrastructure [92]. A distributed hash table is used to organize metadata objects that contain the file system metadata. Additionally, developers have the possibility to annotate this metadata with user-specific tags such as additional information about the application.

EMPRESS. Extensible Metadata Provider for Extreme-Scale Scientific Simulations (EMPRESS) offers customizable metadata tagging in order to mark the interesting data of large-scale simulations before storing. This simplifies locating the relevant data in post-processing and can help avoid searches through the complete data [54].

Týr. Týr is a blob storage system with support for transactions; it provides blob storage functionality and high access parallelism [64]. Measurements show that many applications do not require most of the functionality provided by full-fledged file systems but instead only use a subset that can be provided by blob or object storage systems [63].

4.5. Interfaces and Data Formats

DAOS. In addition to introducing a novel user-space storage system, DAOS will also support new ways of performing I/O in a scalable way [6]. From an application developer's point of

view, DAOS will provide a rich I/O interface in the form of key-array objects with support for both structured and unstructured data. Additionally, established I/O interfaces such as a legacy POSIX interface and an HDF5 interface will be supported natively. Similar to databases, DAOS has support for transactions, that is, multiple operations can be batched in a single transaction, which becomes immutable, durable and consistent once committed as an epoch. On the one hand, this allows multiple processes to perform asynchronous write operations without having to worry about consistency problems. On the other hand, read operations will always have a consistent view because they are based on a committed (and thus immutable) epoch.

Conclusion

This survey takes a snapshot of the current storage landscape and accentuates the areas that require more research in the future. Current co-design efforts outline a plausible path to exascale systems, but in the long term, the widening gap between computing and storage system capabilities requires coordinated efforts on multiple fronts. Fundamental research and funding directed towards software and hardware storage technologies are required. On the hardware side, NVRAM storage will likely transform how we build storage systems. On the one hand, NVRAM can improve the capabilities to record large amounts of data at the pace required to be useful for later analysis tasks. On the other hand, NVRAM can dramatically simplify storage systems, which currently add complexity to every effort for relatively modest performance improvements.

Hardware improvements alone will not ensure high-performance storage systems to keep pace with the ever-increasing computational power. Applications and workloads, as well as data centers, differ, but as many hardware components cannot be operated economically for over five years, hardware-specific optimizations in applications are only feasible to a limited extent. Applications typically have much longer lifetimes and, thus, research in software to come up with convenient and portable interfaces is required. Approaches such as DAOS show that it is possible to offer advanced and novel I/O interfaces without breaking backwards compatibility with existing applications. By natively supporting established high-level interfaces such as HDF5, applications do not need to be ported if they are already making use of such an interface. Moreover, additional information made available by high-level interfaces can be used for optimizing I/O and data management decisions.

Currently, a large number of domain-specific solutions are in use due to differing requirements within each domain. Concentrating efforts on providing efficient and scalable solutions that are generic enough to be used in multiple or all domains would allow reducing the fragmentation we currently observe. This, however, is not a purely technical problem and would require broad agreement across many domains. A more realistic goal would be to provide a solid base that can be extended with relatively thin wrappers for each specific domain. For interfaces and data formats, this has already happened in part with multiple domains (including high-energy physics and climate science) basing their solutions on the established HDF5 format.

In addition, training activities for application developers but also programs to educate experts which will develop the next generation of storage systems and technologies are necessary. Data-driven sciences provide huge socio-economic benefits, but they are slowed down due to a lack of experts, convenient software and sufficiently powerful systems.

Acknowledgements

Parts of this publication are enabled by the following projects: BigStorage (Storage-based Convergence between HPC and Cloud to Handle Big Data), funded by the European Union under the Marie Skłodowska-Curie Actions (H2020-MSCA-ITN-2014-642963). ESiWACE, which received funding from the EU Horizon 2020 research and innovation programme under grant agreement no. 675191. This material reflects only the authors' view and the EU commission is not responsible for any use that may be made of the information it contains.

References

1. Ang, J.A.: High performance computing co-design strategies. In: Proceedings of the 2015 International Symposium on Memory Systems. pp. 51–52. MEMSYS '15, ACM, New York, NY, USA (2015), DOI: 10.1145/2818950.2818959

2. Bennett, J., Abbasi, H., Bremer, P., Grout, R.W., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., Pébay, P.P., Thompson, D.C., Yu, H., Zhang, F., Chen, J.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA, November 11 - 15, 2012 (2012), DOI: 10.1109/SC.2012.31

3. Bornholt, J., Lopez, R., Carmean, D.M., Ceze, L., Seelig, G., Strauss, K.: A DNA-Based Archival Storage System. SIGPLAN Notices 51(4), 637–649 (2016), DOI: 10.1145/2954679.2872397

4. Boukhobza, J., Rubini, S., Chen, R., Shao, Z.: Emerging nvm: A survey on architectural integration and research challenges. ACM Transactions on Design Automation of Electronic Systems (TODAES) 23(2), 14:1–14:32 (2017), DOI: 10.1145/3131848

5. BoyukaII, D.A., Lakshminarasimhan, S., Zou, X., Gong, Z., Jenkins, J., Schendel, E.R., Podhorszki, N., Liu, Q., Klasky, S., Samatova, N.F.: Transparent in situ data transformations in ADIOS. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014. pp. 256–266 (2014), DOI: 10.1109/CCGrid.2014.73

6. Breitenfeld, M.S., Fortner, N., Henderson, J., Soumagne, J., Chaarawi, M., Lombardi, J., Koziol, Q.: DAOS for extreme-scale systems in scientific applications. CoRR abs/1712.00423 (2017), `http://arxiv.org/abs/1712.00423`, accessed: 2018-03-01

7. Brinkmann, A., Mohror, K., Yu, W.: Challenges and opportunities of user-level file systems for HPC (dagstuhl seminar 17202). Dagstuhl Reports 7(5), 97–139 (2017), DOI: 10.4230/DagRep.7.5.97

8. CDI Developers: Climate Data Interface. `https://code.mpimet.mpg.de/projects/cdi` (2018), accessed: 2018-02-01

9. Chen, J., Wei, Q., Chen, C., Wu, L.: FSMAC: A file system metadata accelerator with non-volatile memory. In: IEEE 29th Symposium on Mass Storage Systems and Technologies, MSST 2013, May 6-10, 2013, Long Beach, CA, USA. pp. 1–11 (2013), DOI: 10.1109/MSST.2013.6558440

10. Corbett, P., Feitelson, D., Fineberg, S., Hsu, Y., Nitzberg, B., Prost, J.P., Snir, M., Traversat, B., Wong, P.: Overview of the MPI-IO Parallel I/O Interface. In: IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems. pp. 1–15 (1995), http://lovelace.nas.nasa.gov/MPI-IO/iopads95-paper.ps, accessed: 2018-03-01

11. Cray: CRAY XC40 DataWarp Applications I/O Accelerator. http://www.cray.com/sites/default/files/resources/CrayXC40-DataWarp.pdf, accessed: 2018-03-01

12. DDN: Worlds's most advanced application aware I/O acceleration solutions. http://www.ddn.com/products/infinite-memory-engine-ime14k, accessed: 2018-03-01

13. Dennis, J.M., Edwards, J., Loy, R.M., Jacob, R.L., Mirin, A.A., Craig, A.P., Vertenstein, M.: An application-level parallel I/O library for earth system models. The International Journal of High Performance Computing Applications (IJHPCA) 26(1), 43–53 (2012), DOI: 10.1177/1094342011428143

14. Depardon, B., Le Mahec, G., Séguin, C.: Analysis of Six Distributed File Systems. Research report (2013), https://hal.inria.fr/hal-00789086, accessed: 2018-02-01

15. Deuzeman, A., Reker, S., Urbach, C.: Lemon: An MPI parallel I/O library for data encapsulation using LIME. Computer Physics Communications 183(6), 1321–1335 (2012), DOI: 10.1016/j.cpc.2012.01.016

16. DOE, NISA: Exascale Computing Project (ECP). https://www.exascaleproject.org/ (2017), accessed: 2018-03-01

17. Dongarra, J.J., Tomov, S., Luszczek, P., Kurzak, J., Gates, M., Yamazaki, I., Anzt, H., Haidar, A., Abdelfattah, A.: With extreme computing, the rules have changed. Computing in Science and Engineering 19(3), 52–62 (2017), DOI: 10.1109/MCSE.2017.48

18. Dulloor, S., Roy, A., Zhao, Z., Sundaram, N., Satish, N., Sankaran, R., Jackson, J., Schwan, K.: Data tiering in heterogeneous memory systems. In: Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016. pp. 15:1–15:16 (2016), DOI: 10.1145/2901318.2901344

19. Duran-Limon, H.A., Flores-Contreras, J., Parlavantzas, N., Zhao, M., Meulenert-Peña, A.: Efficient execution of the WRF model and other HPC applications in the cloud. Earth Science Informatics 9(3), 365–382 (2016), DOI: 10.1007/s12145-016-0253-7

20. ESiWACE: Centre of Excellence in Simulation of Weather and Climate in Europe. https://www.esiwace.eu/, accessed: 2018-03-01

21. Fraunhofer ITWM, ThinkParQ: BeeGFS - The Leading Parallel Cluster File System. https://www.beegfs.io (2018), accessed: 2018-02-01

22. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel I/O to task-local files. In: Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, 2009, Portland, Oregon, USA (2009), DOI: 10.1145/1654059.1654077

23. Geist, A., Reed, D.A.: A survey of high-performance computing scaling challenges. The International Journal of High Performance Computing Applications (IJHPCA) 31(1), 104–113 (2017), DOI: 10.1177/1094342015597083

24. Gensel, J., Josselin, D., Vandenbroucke, D. (eds.): Bridging the Geographic Information Sciences - International AGILE'2012 Conference, Avignon, France, April 24-27, 2012. Lecture Notes in Geoinformation and Cartography, Springer (2012), DOI: 10.1007/978-3-642-29063-3

25. Grawinkel, M., Nagel, L., Mäsker, M., Padua, F., Brinkmann, A., Sorth, L.: Analysis of the ecmwf storage landscape. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies. pp. 15–27. FAST '15, USENIX Association, Berkeley, CA, USA (2015), http://dl.acm.org/citation.cfm?id=2750482.2750484, accessed: 2018-03-01

26. Guest, M.: Prace: The Scientific Case for HPC in Europe. Insight publishers, Bristol (2012)

27. Gupta, A., Kalé, L.V., Gioachin, F., March, V., Suen, C.H., Lee, B., Faraboschi, P., Kaufmann, R., Milojicic, D.S.: The who, what, why, and how of high performance computing in the cloud. In: IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013. vol. 1, pp. 306–314 (2013), DOI: 10.1109/CloudCom.2013.47

28. Gupta, A., Milojicic, D.: Evaluation of hpc applications on cloud. In: Proceedings of the 2011 Sixth Open Cirrus Summit. pp. 22–26. OCS '11, IEEE Computer Society, Washington, DC, USA (2011), DOI: 10.1109/OCS.2011.10

29. HDF Group: RFC: Virtual Object Layer (2014)

30. HDF5: Hierarchical data model. https://www.hdfgroup.org/hdf5/, accessed: 2018-03-01

31. He, W., Du, D.H.C.: Smart: An approach to shingled magnetic recording translation. In: 15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017. pp. 121–134 (2017), https://www.usenix.org/conference/fast17/technical-sessions/presentation/he, accessed: 2018-03-01

32. HGST: Ultrastar-Hs14-DS. https://www.hgst.com/sites/default/files/resources/Ultrastar-Hs14-DS.pdf, accessed: 2018-03-01

33. HGST: Ultrastar-SN200. https://www.hgst.com/sites/default/files/resources/Ultrastar-SN200-Series-datasheet.pdf, accessed: 2018-03-01

34. High Performance Storage System: Publicly disclosed HPSS deployments. http://www.hpss-collaboration.org/customersT.shtml (2018), accessed: 2018-02-01

35. Hubovsky, R., Kunz, F.: Dealing with Massive Data: from Parallel I/O to Grid I/O. Master's thesis, University of Vienna, Department of Data Engineering (2004)

36. Hughes, J., Fisher, D., Dehart, K., Wilbanks, B., Alt, J.: HPSS RAIT Architecture. White paper of the HPSS collaboration (2009), http://www.hpss-collaboration.org/documents/HPSS_RAIT_Architecture.pdf, accessed: 2018-03-01

37. IBM: Flash Storage. https://www.ibm.com/storage/flash, accessed: 2018-03-01

38. IBM: General Parallel File System. https://www.ibm.com/support/knowledgecenter/en/SSFKCN/gpfs_welcome.html (2016), accessed: 2018-02-01

39. IBM: Ibm spectrum scale: Overview. http://www.ibm.com/systems/storage/spectrum/scale/ (2016), accessed: 2018-03-01

40. IEEE, T., Group, T.O.: Standard for Information Technology – Portable Operating System Interface (POSIX) Base Specifications, Issue 7. IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013) pp. 1–3906 (2013), DOI: 10.1109/IEEESTD.2013.6506091

41. Intel: Intel® 3D NAND Technology Transforms the Economics of Storage. https://www.intel.com/content/www/us/en/solid-state-drives/3d-nand-technology-animation.html, accessed: 2018-03-01

42. Intel: Intel® QuickAssist Technology (Intel® QAT) Improves Data Center... https://www.intel.com/content/www/us/en/architecture-and-technology/intel-quick-assist-technology-overview.html, accessed: 2018-03-01

43. Intel: Optane SSD DC P4800X. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf, accessed: 2018-03-01

44. Intel, The HDF Group, EMC, Cray: Fast Forward Storage and I/O - Final Report. https://wiki.hpdd.intel.com/display/PUB/Fast+Forward+Storage+and+IO+Program+Documents?preview=/12127153/22872065/M8.5%20FF-Storage%20Final%20Report%20v3.pdf (2014), accessed: 2018-03-01

45. ITRS: International technology roadmap for semiconductors - 2.0. Tech. rep. (2015)

46. Kimpe, D., Ross, R.: Storage models: Past, present, and future. High Performance Parallel I/O pp. 335–345 (2014)

47. Kingston: KSM24LQ4/64HAI. https://www.kingston.com/dataSheets/KSM24LQ4_64HAI.pdf, accessed: 2018-03-01

48. Klein, A.: Backblaze Hard Drive Stats for 2017. https://www.backblaze.com/blog/hard-drive-stats-for-2017/ (2018), accessed: 2018-03-01

49. Kove Corporation: About xpress disk (xpd) (2015), http://www.hamburgnet.de/products/kove/Kove-XPD-L3-4-datasheet.pdf, accessed: 2018-03-01

50. Kuhn, M., Kunkel, J., Ludwig, T.: Data Compression for Climate Data. Supercomputing Frontiers and Innovations 3(1), 75–94 (2016), DOI: 10.14529/jsfi160105

51. Kunkel, J.M., Betke, E.: An MPI-IO in-memory driver for non-volatile pooled memory of the kove XPD. In: High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers. pp. 679–690 (2017), DOI: 10.1007/978-3-319-67630-2_48

52. Labs, C.: Open-Channel Solid State Drives. https://openchannelssd.readthedocs.io/en/latest, accessed: 2018-02-01

53. Latham, R., Ross, R.B., Thakur, R.: Implementing MPI-IO atomic mode and shared file pointers using MPI one-sided communication. The International Journal of High Performance Computing Applications (IJHPCA) 21(2), 132–143 (2007), DOI: 10.1177/1094342007077859

54. Lawson, M., Ulmer, C., Mukherjee, S., Templet, G., Lofstead, J.F., Levy, S., Widener, P.M., Kordenbrock, T.: Empress: extensible metadata provider for extreme-scale scientific simulations. In: Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, PDSW-DISCS@SC 2017, Denver, CO, USA, November 13, 2017. pp. 19–24 (2017), DOI: 10.1145/3149393.3149403

55. Ledyayev, R., Richter, H.: High performance computing in a cloud using OpenStack. Cloud Computing pp. 108–113 (2014), https://pdfs.semanticscholar.org/2a5d/9c7afcf6b70ad83bca0c4262b66ef654415a.pdf, accessed: 2018-03-01

56. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua, R., Lofstead, J.F., Oldfield, R., Parashar, M., Samatova, N.F., Schwan, K., Shoshani, A., Wolf, M., Wu, K., Yu, W.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. Concurrency and Computation: Practice and Experience 26(7), 1453–1473 (2014), DOI: 10.1002/cpe.3125

57. Lockwood, G.K.: Reviewing the state of the art of burst buffers. https://glennklockwood.blogspot.de/2017/03/reviewing-state-of-art-of-burst-buffers.html (2017), accessed: 2018-02-01

58. Lockwood, G.: What's So Bad About POSIX I/O? https://www.nextplatform.com/2017/09/11/whats-bad-posix-io/ (2017), accessed: 2018-02-01

59. Lockwood, G.K., Hazen, D., Koziol, Q., Canon, R., Antypas, K., Balewski, J., Balthaser, N., Bhimji, W., Botts, J., Broughton, J., et al.: Storage 2020: A vision for the future of hpc storage (2017), https://escholarship.org/uc/item/744479dp, accessed: 2018-03-01

60. Lofstead, J.F., Jimenez, I., Maltzahn, C., Koziol, Q., Bent, J., Barton, E.: DAOS and friends: a proposal for an exascale storage system. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016. pp. 585–596 (2016), DOI: 10.1109/SC.2016.49

61. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In: 6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, MA, USA, June 23, 2008. pp. 15–24 (2008), DOI: 10.1145/1383529.1383533

62. LTO: Linear Tape Open (LTO) Website. https://www.lto.org/technology/what-is-lto-technology/, accessed: 2018-03-01

63. Matri, P., Alforov, Y., Brandon, A., Kuhn, M., Carns, P.H., Ludwig, T.: Could blobs fuel storage-based convergence between HPC and big data? In: 2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017. pp. 81–86 (2017), DOI: 10.1109/CLUSTER.2017.63

64. Matri, P., Costan, A., Antoniu, G., Montes, J., Pérez, M.S.: Týr: blob storage meets built-in transactions. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016. pp. 573–584 (2016), DOI: 10.1109/SC.2016.48

65. McKee, S.A.: Reflections on the memory wall. In: Proceedings of the First Conference on Computing Frontiers, 2004, Ischia, Italy, April 14-16, 2004. p. 162 (2004), DOI: 10.1145/977091.977115

66. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Version 3.0. http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf (2012), accessed: 2014-11-01

67. Micheloni, R., Crippa, L., Zambelli, C., Olivo, P.: Architectural and integration options for 3d NAND flash memories. Computers 6(3), 27 (2017), DOI: 10.3390/computers6030027

68. NetCDF: Network common data format. https://www.unidata.ucar.edu/software/netcdf/, accessed: 2018-03-01

69. NEXTGenIO: Next Generation I/O for the exascale. http://www.nextgenio.eu/, accessed: 2018-03-01

70. Oe, K., Nanri, T., Okamura, K.: Feasibility study for building hybrid storage system consisting of non-volatile DIMM and SSD. In: Fourth International Symposium on Computing and Networking, CANDAR 2016, Hiroshima, Japan, November 22-25, 2016. pp. 454–457 (2016), DOI: 10.1109/CANDAR.2016.0085

71. Oh, M., Eom, J., Yoon, J., Yun, J.Y., Kim, S., Yeom, H.Y.: Performance optimization for all flash scale-out storage. In: 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016. pp. 316–325 (2016), DOI: 10.1109/CLUSTER.2016.11

72. ORNL: Adios. https://www.olcf.ornl.gov/center-projects/adios/ (2017), accessed: 2018-03-01

73. Peng, I.B., Gioiosa, R., Kestor, G., Laure, E., Markidis, S.: Exploring the Performance Benefit of Hybrid Memory System on HPC Environments. arXiv:1704.08273 [cs] pp. 683–692 (2017), DOI: 10.1109/IPDPSW.2017.115

74. Perarnau, S., Zounmevo, J.A., Gerofi, B., Iskra, K., Beckman, P.H.: Exploring data migration for future deep-memory many-core systems. In: 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, 2016. pp. 289–297 (2016), DOI: 10.1109/CLUSTER.2016.42

75. Petersen, T.K., Bent, J.: Hybrid flash arrays for HPC storage systems: An alternative to burst buffers. In: 2017 IEEE High Performance Extreme Computing Conference, HPEC 2017, Waltham, MA, USA, September 12-14, 2017. pp. 1–7 (2017), DOI: 10.1109/HPEC.2017.8091092

76. Rew, R., Davis, G.: Netcdf: an interface for scientific data access. IEEE Computer Graphics and Applications 10(4), 76–82 (1990), DOI: 10.1109/38.56302

77. Ross, R.B., Latham, R., Gropp, W., Thakur, R., Toonen, B.R.: Implementing MPI-IO atomic mode without file system support. In: 5th International Symposium on Cluster Computing and the Grid (CCGrid 2005), 9-12 May, 2005, Cardiff, UK. pp. 1135–1142 (2005), DOI: 10.1109/CCGRID.2005.1558687

78. SAGE: SAGE | Redefining Data Storage for Extreme Data and Exascale Computing. http://www.sagestorage.eu/, accessed: 2018-03-01

79. Sai Anuhya, D., Agrawal, S., Publication, I.: 3-d holographic data storage 4, 232–239 (2013)

80. Sato, K., Mohror, K., Moody, A., Gamblin, T., de Supinski, B.R., Maruyama, N., Matsuoka, S.: A user-level infiniband-based file system and checkpoint strategy for burst buffers. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014. pp. 21–30 (2014), DOI: 10.1109/CCGrid.2014.24

81. Seagate: Nytro 5910. https://www.seagate.com/files/www-content/datasheets/pdfs/nytro-5910-nvme-ssdDS1953-3-1801DE-de_DE.pdf, accessed: 2018-03-01

82. Seagate: Nytro3000. https://www.seagate.com/files/www-content/datasheets/pdfs/nytro-3000-sas-ssdDS1950-2-1711DE-de_DE.pdf, accessed: 2018-03-01

83. Sebastian, A.: IBM and Sony cram up to 330 terabytes into tiny tape cartridge. https://arstechnica.com/information-technology/2017/08/ibm-and-sony-cram-up-to-330tb-into-tiny-tape-cartridge/ (2017), accessed: 2018-03-01

84. Sehrish, S.: Improving Performance and Programmer Productivity for I/O-Intensive High Performance Computing Applications. PhD thesis, School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida (2010), http://purl.fcla.edu/fcla/etd/CFE0003236, accessed: 2018-03-01

85. Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J., Masanet, E., Horner, N., Azevedo, I., Lintner, W.: United States data center energy usage report. https://pubarchive.lbl.gov/islandora/object/ir%3A1005775/ (2016), accessed: 2018-03-01

86. Shi, W., Ju, D., Wang, D.: Saga: A cost efficient file system based on cloud storage service. In: Economics of Grids, Clouds, Systems, and Services - 8th International Workshop, GECON 2011, Paphos, Cyprus, December 5, 2011, Revised Selected Papers. pp. 173–184 (2011), DOI: 10.1007/978-3-642-28675-9_13

87. Smart, S.D., Quintino, T., Raoult, B.: A Scalable Object Store for Meteorological and Climate Data. In: Proceedings of the Platform for Advanced Scientific Computing Conference. pp. 13:1–13:8. PASC '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3093172.3093238

88. Spectralogic: LTO Roadmap. https://www.spectralogic.com/features/lto-7/ (2017), accessed: 2018-03-01

89. Sterling, T., Lusk, E., Gropp, W.: Beowulf Cluster Computing with Linux. MIT Press, Cambridge, MA, USA, 2 edn. (2003)

90. Takatsu, F., Hiraga, K., Tatebe, O.: Design of object storage using opennvm for high-performance distributed file system. Journal of Information Processing 24(5), 824–833 (2016), DOI: 10.2197/ipsjjip.24.824

91. Takatsu, F., Hiraga, K., Tatebe, O.: PPFS: A scale-out distributed file system for post-petascale systems. In: 18th IEEE International Conference on High Performance Computing and Communications; 14th IEEE International Conference on Smart City; 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016, Sydney, Australia, December 12-14, 2016. pp. 1477–1484 (2016), DOI: 10.1109/HPCC-SmartCity-DSS.2016.0210

92. Tang, H., Byna, S., Dong, B., Liu, J., Koziol, Q.: Someta: Scalable object-centric metadata management for high performance computing. In: 2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017. pp. 359–369 (2017), DOI: 10.1109/CLUSTER.2017.53

93. Top500: Top500 Supercomputer Sites. http://www.top500.org/ (2017), accessed: 2018-03-01

94. Toshiba: eSSD-PX05SHB. https://toshiba.semicon-storage.com/content/dam/toshiba-ss/asia-pacific/docs/product/storage/product-manual/eSSD-PX05SHB-product-overview.pdf, accessed: 2018-03-01

95. Vilayannur, M., Lang, S., Ross, R., Klundt, R., Ward, L.: Extending the POSIX I/O Interface: A Parallel File System Perspective. Tech. Rep. ANL/MCS-TM-302 (2008), http://www.mcs.anl.gov/uploads/cels/papers/TM-302-FINAL.pdf, accessed: 2018-03-01

96. Vilayannur, M., Ross, R.B., Carns, P.H., Thakur, R., Sivasubramaniam, A., Kandemir, M.T.: On the performance of the POSIX I/O interface to PVFS. In: 12th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2004), 11-13 February 2004, A Coruna, Spain. pp. 332–339 (2004), DOI: 10.1109/EMPDP.2004.1271463

97. Wang, T., Oral, S., Wang, Y., Settlemyer, B.W., Atchley, S., Yu, W.: Burstmem: A high-performance burst buffer system for scientific applications. In: 2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014. pp. 71–79 (2014), DOI: 10.1109/BigData.2014.7004215

98. Watson, R.W.: High performance storage system scalability: Architecture, implementation and experience. In: 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005), Information Retrieval from Very Large Storage Systems, CD-ROM, 11-14 April 2005, Monterey, CA, USA. pp. 145–159 (2005), DOI: 10.1109/MSST.2005.17

99. Wei, Q., Chen, J., Chen, C.: Accelerating file system metadata access with byte-addressable nonvolatile memory. ACM Transactions on Storage (TOS) 11(3), 12:1–12:28 (2015), DOI: 10.1145/2766453

100. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA. pp. 307–320 (2006), http://www.usenix.org/events/osdi06/tech/weil.html, accessed: 2018-03-01

101. Weil, S.A., Leung, A.W., Brandt, S.A., Maltzahn, C.: RADOS: a scalable, reliable storage service for petabyte-scale storage clusters. In: Proceedings of the 2nd International Petascale Data Storage Workshop (PDSW '07), November 11, 2007, Reno, Nevada, USA. pp. 35–44 (2007), DOI: 10.1145/1374596.1374606

102. Xu, J., Swanson, S.: NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In: 14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016. pp. 323–338 (2016), https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu, accessed: 2018-03-01

103. Xuan, P., Luo, F., Ge, R., Srimani, P.K.: Dynims: A dynamic memory controller for in-memory storage on HPC systems. CoRR abs/1609.09294 (2016), http://arxiv.org/abs/1609.09294, accessed: 2018-03-01

104. Yang, J., Tan, C.P.H., Ong, E.H.: Thermal analysis of helium-filled enterprise disk drive. Microsystem Technologies 16(10), 1699–1704 (2010), DOI: 10.1007/s00542-010-1121-x

105. Zhang, Z., Barbary, K., Nothaft, F.A., Sparks, E.R., Zahn, O., Franklin, M.J., Patterson, D.A., Perlmutter, S.: Scientific computing meets big data technology: An astronomy use case. In: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015. pp. 918–927 (2015), DOI: 10.1109/BigData.2015.7363840

DOI: 10.14529/jsfi180104

The High-Q Club: Experience with Extreme-scaling Application Codes

***Dirk Brömmel*[1], *Wolfgang Frings*[1], *Brian J. N. Wylie*[1], *Bernd Mohr*[1], *Paul Gibbon*[1], *Thomas Lippert*[1]**

Jülich Supercomputing Centre (JSC) started running (extreme) scaling workshops with its first IBM Blue Gene supercomputer, finally spanning three generations each seeing an increase in the number of cores and available threads. Over the years, this workshop series attracted numerous international code teams and resulted in many applications capable of running on all available cores of each system.

This article reviews some of the knowledge gained with running and tuning highly-scalable applications, focussing on *JUQUEEN*, the IBM Blue Gene/Q at JSC. The ability to execute successfully on all 458,752 cores with up to 1.8 million processes or threads may qualify codes for the High-Q Club, which serves as a showcase for diverse codes scaling to the entire 28 racks, effectively defining a collection of the highest scaling codes on *JUQUEEN*. The intention was to encourage other developers to invest in tuning and scaling their codes while identifying the necessary key aspects for that goal.

As this era closes, it is timely to compare the characteristics of the 32 High-Q Club member codes, considering their strong and/or weak scaling, exploitation of hardware threading, and whether/how intra-node multi-threading is employed combined with message-passing. We also identify the obstacles for scaling such as inefficient use of limited compute node memory and file I/O as key governing factors. Overall, the analysis provides guidance as to how applications may (need to) be designed in the future to exploit expected exascale computer systems.

Keywords: JUQUEEN, IBM Blue Gene/Q, extreme scaling, application codes, High-Q Club.

Introduction

Jülich Supercomputing Centre (JSC) has more than a decade of experience with the range of IBM Blue Gene systems and scaling HPC applications to use their considerable capabilities effectively. Applications that demonstrate scalability to exploit the entire computational resources of the *JUQUEEN* system qualify for recognition in the High-Q Club. With the decommissioning of *JUQUEEN* in spring 2018, it is timely to analyse the characteristics of these extremely scalable applications for valuable insight into how applications may (need to) look in the future to exploit forthcoming exascale computer systems. Furthermore, supporting development tools and libraries also need to have commensurate scalability to address current and future application needs on these massively-parallel systems.

We start by reviewing application scaling activities at JSC in Section 1, focussing on its leadership computer systems, and in Section 2 the *JUQUEEN* hardware environment, followed with an overview of the associated High-Q Club in Section 3. Member applications and their basic characteristics are summarised in Section 4, as adapted to the BG/Q software environment, while Section 5 is a detailed comparison of the demonstrated scalability of these codes on *JUQUEEN*. Specific tools which have proven effective for use with them at (very) large scale are then reviewed in Section 6. Finally we present our conclusions from this study regarding High-Q Club member application codes and readiness for exascale.

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

1. Background

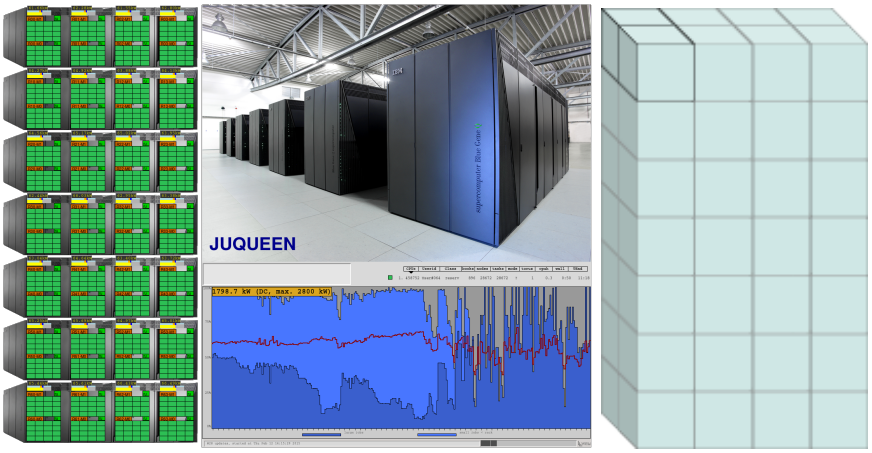
The first of the IBM Blue Gene series installed in 2005 by Jülich Supercomputing Centre was the *JUBL* Blue Gene/L, succeeded by the *JUGENE* Blue Gene/P [2] in 2007, and ultimately the *JUQUEEN* Blue Gene/Q [34] in 2012 which remained in operation to 2018. Key characteristics of these supercomputers are summarised in Tab. 1. Part of each of these systems was normally reserved for small/short application development executions, complemented by a variety of larger partitions of different sizes for longer jobs. While batch jobs could also be submitted to queues requiring the full system, they would only be run immediately following maintenance sessions or other predefined times to avoid undue interference with general usage. "Big Blue Gene Weeks" with seven or more days restricted exclusively to large-scale jobs were introduced first in 2015 and readily were exploited by numerous application teams, both for scaling tests and production. Before then, scaling-up applications could be a protracted process as a series of larger jobs was prepared, eventually executed, and adjusted. Dedicated scaling, and subsequently extreme scaling, workshops were held to facilitate this.

A "Blue Gene/L Scaling Workshop" [10] was held in 2006, became a "Blue Gene/P Porting, Tuning & Scaling Workshop" in 2008 [25], followed by dedicated "Extreme Scaling Workshops" in 2009 [26], 2010 [27] and 2011 [28]. These latter three workshops attracted 28 teams selected from around the world to investigate scalability on the most massively-parallel supercomputer at the time with its 294,912 cores. 26 of their codes were successfully executed at that scale, three became ACM Gordon Bell prize finalists, and one participant was awarded an ACM/IEEE-CS George Michael Memorial HPC fellowship.

"Extreme Scaling Workshops" for Blue Gene/Q continued in 2015 [4], 2016 [6] and 2017 [7] with a similar format. Based on their demonstrated and planned use of BG/Q, a total of 19

Table 1. Blue Gene series of computers installed at Jülich Supercomputing Centre

Name	*JUBL*	*JUGENE*	*JUQUEEN*
Installation year	2005	2007	2012
Series	BG/L	BG/P	BG/Q
Racks	8	72	28
Compute nodes	8,192	73,728	28,672
Processor	PowerPC 440	PowerPC 450	PowerPC A2
Frequency (MHz)	700	850	1,600
Compute cores	2	4	16
Hardware threads/core	1	1	4
Total cores	16,384	294,912	458,752
Total threads	16,384	294,912	1,835,008
Main memory (TiB)	4.1	144	448
Memory per core (MiB)	256	512	1,024
I/O Nodes	288	600	248
Network	3D torus	3D torus	5D torus
Peak performance (TFlop/s)	45.9	1,002.7	5,872.0
Footprint (m^2)	15	130	82
Power consumption (kW)	179	2,268	2,301



JUQUEEN

Partition	Cores
$8 = 4\times1\times2$	65,536
$16 = 4\times2\times2$	131,072
$24 = 4\times3\times2$	196,608
$32 = 4\times4\times2$	262,144
$40 = 4\times5\times2$	327,680
$48 = 4\times6\times2$	393,216
$14 = 1\times7\times2$	114,688
$28 = 2\times7\times2$	229,376
$42 = 3\times7\times2$	344,064
$56 = 4\times7\times2$	458,752

Figure 1. *JUQUEEN* Blue Gene/Q as presented by the *LLview* system monitor (28 racks with two midplanes each with 16 nodeboards of 32 processors), schematic of the 56 BG/Q rack midplanes arranged $4\times7\times2$ and list of partitions with corresponding number of cores available when scaling rows or columns of racks to the entire configuration

international teams were selected and hosted on-site by JSC for two or three days with dedicated access to *JUQUEEN* for up to 50 hours in each event. Many of the teams' codes had thematic overlap with JSC Simulation Laboratories (SimLabs), which provided assistance during the workshops along with Cross-Sectional Teams available to do performance analyses and suggest optimisation opportunities. The first of these workshops had seven applications all running successfully within 24 hours on all 28 racks (using 458,752 cores), however, in the following two editions some participants encountered difficulties which they were unable to resolve, and there were only eleven additional successes. File I/O bottlenecks were a frequent constraint for some codes, while large-scale in situ interactive visualization using 458,752 MPI processes running on 28 racks coupled via *JUSITU* to VisIt was successfully demonstrated as a possible alternative [16].

These workshops motivated Leibniz Supercomputing Centre (LRZ), JSC's partner with HLRS in the German national Gauss Centre for Supercomputing (GCS), to adopt a similar format for workshops to scale applications on the *SuperMUC* IBM iDataPlex system [18], and similar opportunities are expected to be offered on future HPC systems at JSC.

2. *JUQUEEN* Blue Gene/Q

JUQUEEN[2] is an IBM Blue Gene/Q system consisting of 28 racks [34] (Fig. 1), each with two midplanes comprising 512 compute nodes with 1.6 GHz PowerPC A2 processors and 16 GiB RAM, connected via a custom five-dimensional torus network. Compute node processors provide 16 cores to applications, each with a 256-bit SIMD/vector unit and capable of running four hardware threads, therefore *JUQUEEN* offers a total of 458,752 cores and can concurrently run 1,835,008 processes or threads. An additional 248 I/O nodes connect via Cisco network switches to the *JUST* GPFS filesystem.

IBM Blue Gene/Q systems [1] (like their predecessors) have demonstrated their merits for large-scale HPC regarding energy-efficiency, reliability and scalability to extremely large con-

[2] `http://www.fz-juelich.de/ias/jsc/juqueen`

figurations, making them highly-productive workhorses. *JUQUEEN* was ranked number five for energy efficiency as well as fifth for HPL performance in the November 2012 Green500 and Top500 lists of supercomputers, and coarse-grain energy usage is monitored during operation (though no control is available for applications to adjust their usage). System warnings and errors are also closely monitored, leading to pre-emptive hardware replacement at the next scheduled maintenance, such that their is rarely impact on production (and particularly full-system jobs). Application-level checkpointing is still recommended for long-running jobs.

By providing dedicated system partitions, applications receive isolated resources for computation and communication, the latter based on a proprietary 5-D torus network. With an extra 17th processor core for system services and lightweight compute-node kernel (CNK), applications are further isolated from "system noise" that can otherwise severely impact performance of collective operations. And with uniform memory access, process/thread placement and bindings to address NUMA issues are unnecessary. On the other hand, compute-node memory is limited (typically 16 GiB) and the relatively weak individual processor cores require the effective use of large numbers of them by applications, which is further exacerbated by the in-order processor architecture [20]. Multiple application threads for the four hardware threads per processor core are therefore generally considered to be the best bet for increased instruction throughput and latency hiding.

Figure 1 shows a topological schematic of the 28 BG/Q racks of *JUQUEEN* with its 56 midplanes arranged $4 \times 7 \times 2$. When performing scaling tests using rows or columns of racks, increments of 8 or 14 midplanes are available. (Square numbers of 1, 4, 9 and 16 racks can also be used, however, $25 = 5 \times 5$ racks cannot be configured, which severely limits scaling.) Note that physical adjacency of racks is not essential to exploit mesh or torus topologies, however, even-sized dimensions generally offer superior performance since odd-sized dimensions do not support torus connectivity. 16 racks (32 midplanes) therefore constitute the preferred configuration for compact torus communication, and this partition is available and commonly used in general operation of *JUQUEEN*. Partitions larger than this are typically only made available after maintenance periods and for specially scheduled sessions. In particular, full-system executions requiring all 28 racks occupy the rack that is otherwise reserved for small/short development jobs. Large jobs using 24 racks are therefore often a convenient compromise.

Bearing these considerations in mind, executions on partitions with 24 racks (48 midplanes, 86% of entire system) and perhaps even only 20/21 racks (40/42 midplanes, 71/75%) could be covered by the definition of "large-scale" with respect to *JUQUEEN*.

3. High-Q Club

The High-Q Club is a collection of the highest scaling codes on *JUQUEEN*, and membership requires codes to run successfully using all 28 racks. Codes also have to demonstrate that they profit from each additional rack of *JUQUEEN*, either with reduced time to solution when strong scaling a fixed problem size or a tolerable increase in runtime when weak scaling progressively larger problems. Furthermore, application configurations should be beyond toy examples, and use of all available hardware threads is encouraged which is often best achieved via mixed-mode programming. Each code is then individually evaluated based on its weak or strong scaling results with no strict limit on efficiency.

The full description of the High-Q Club codes and a summary of their scaling performance along with developer and contact information is maintained on-line [22]. Further detail is avail-

able in participants' reports from Extreme Scaling Workshops [4, 6, 7]. 32 codes are listed (with those from Extreme Scaling Workshops marked with an asterix*):

1D-NEGF *1D Non-Equilibrium Green's Functions framework for transport phenomena*
JSC SimLab Quantum Materials

*CIAO *multiphysics, multiscale NS solver for turbulent reacting flows in complex geometries*
RWTH Aachen University Institute for Combustion Technology [16]

*Code_Saturne *multiphysics simulation of the Navier-Stokes equations*
EDF & STFC Daresbury Laboratory[3]

*CoreNeuron *electrical activity of neuronal networks with morphologically-detailed neurons*
EPFL Blue Brain Project and Yale University [29]

dynQCD *lattice quantum chromodynamics with dynamical fermions*
JSC SimLab Nuclear and Particle Physics & Bergische Universität Wuppertal

*FE2TI *scale-bridging approach incorporating micro-mechanics in macroscopic simulations of multi-phase steels*
Universities of Cologne, Freiberg, Duisburg-Essen, Dresden and Erlangen-Nürnberg [23]

*FEMPAR *massively-parallel finite-element simulation of multi-physics PDE problems*
Universitat Politècnica de Catalunya CIMNE

Gysela *gyrokinetic semi-Lagrangian code for plasma turbulence simulations*
CEA-IRFM Cadarache

hp-fRG *hierarchically parallelised functional renormalisation group calculations*
JSC [31]

*ICON *solver for fully compressible non-hydrostatic motion at very high horizontal resolution*
Deutsches Klimarechenzentrum (DKRZ) & JSC SimLab Climate Science

IMD *classical molecular dynamics simulations*
Ruhr-Universität Bochum & JSC SimLab Molecular Systems

JURASSIC *infrared radiative transfer in the Earth's atmosphere*
JSC SimLab Climate Science

JuSPIC *fully relativistic particle-in-cell code for plasma physics and laser-plasma interaction*
JSC SimLab Plasma Physics

*KKRnano *Korringa-Kohn-Rostoker Green function code for quantum description of nano-materials in all-electron density-functional calculations*
Forschungszentrum Jülich IAS

LAMMPS-DCM *molecular dynamics simulation with dynamic cutoff method for arbitrarily large interfacial systems*
RWTH Aachen University AICES [33]

MP2C *massively-parallel multi-particle collision dynamics for soft matter physics and mesoscopic hydrodynamics*
JSC SimLab Molecular Systems [9]

*MPAS-A *multi-scale non-hydrostatic atmospheric model for global, convection-resolving climate simulations*
Karlsruhe Institute of Technology & National Center for Atmospheric Research [19]

[3] `http://www.code-saturne.org/`

$\mu\phi$ (muPhi) *algebraic multi-grid solver for simulation of water flow and solute transport in porous media*
Universität Heidelberg

Musubi *multi-component Lattice Boltzmann solver for flow simulations*
Universität Siegen [30]

NEST *large-scale simulations of biological neuronal networks*
Forschungszentrum Jülich INM-6, IAS-6 & JSC SimLab Neuroscience [21]

OpenTBL *direct numerical simulation of turbulent flows*
Universidad Politécnica de Madrid

*ParFlow+p4est *high resolution parallel simulation of variably saturated flow*
Forschungszentrum Jülich IBG-3, Colorado School of Mines, LLNL & Universität Bonn [8]

*pe *physics engine framework for simulations of rigid bodies with arbitrary shapes*
Universität Erlangen-Nürnberg[4]

PEPC *tree code for N-body simulations, beam-plasma interaction, vortex dynamics, gravitational interaction, molecular dynamics simulations*
JSC SimLab Plasma Physics

PMG+PFASST *space-time parallel solver for systems of ODEs with linear stiff terms*
LBNL, Universität Wuppertal, Università della Svizzera italiana & JSC

PP-Code *simulator for relativistic and non-relativistic astrophysical plasmas*
University of Copenhagen

*psOpen *direct numerical simulation of fine-scale turbulence*
Jülich-Aachen Research Alliance & TU Freiberg [17]

*Seven-League Hydro (SLH) *all Mach number fluid dynamics in astrophysics*
Heidelberg Institute for Theoretical Studies[5]

*SHOCK *structured high-order finite-difference computational kernel for numerical simulation of compressible flow*
RWTH Aachen University Shock Wave Laboratory [14]

TERRA-NEO *simulation of Earth mantle dynamics*
Universität Erlangen-Nürnberg, LMU & TUM

waLBerla *Lattice-Boltzmann method for simulation of fluid scenarios*
Universität Erlangen-Nürnberg

ZFS *multiphysics framework for compressible/incompressible flow, aero-acoustics & combustion*
RWTH Aachen Uni. Inst. of Aerodynamics & JSC SimLab Fluids and Solids Engineering

Half of the member codes involved institutions from the local region, ten were from the rest of Germany, and six from other European nations. International collaborations included four institutions in the USA.

4. Parallel Program & Execution Configuration Characteristics

Characteristics of these application codes (as contributed by the respective code teams submitting their data) are summarised in Tab. 2 and discussed in this section, with scaling performance compared in the following section. Five codes were accepted to the High-Q Club when

[4] https://www10.informatik.uni-erlangen.de/Research/Projects/pe/
[5] https://slh-code.org/

Table 2. High-Q Club member application code characteristics

Compiler and main programming languages (excluding external libraries), parallelisation including maximal process/thread concurrency (per compute node and overall) and strong and/or weak scaling type, and file I/O implementation. (Supported capabilities unused for scaling runs on *JUQUEEN* in parenthesis)

		Programming				Parallelisation							
Code	Accepted	Compiler / Languages				Tasking		Threading		Concurrency	Type		File I/O
1D-NEGF	2018/02	XL:	C			MPI	1	OpenMP	64	64: 1,835,008	S		N/A
*CIAO	2015/08	XL:			Ftn	MPI	16			16: 458,752	S		MPI-IO,HDF5
*Code_Saturne	2016/03	XL:	C		Ftn	MPI	16	OpenMP	4	64: 1,835,008	S		MPI-IO
*CoreNeuron	2015/02	XL:	C	C++		MPI	1	OpenMP	64	64: 1,835,008	S	W	MPI-IO
dynQCD	2013/06	XL:	C			SPI	1	pthreads	64	64: 1,835,008	S		*unspecified*
*FE2TI	2015/02	XL:	C	C++		MPI	16	OpenMP	4	64: 1,835,008	S	W	N/A
*FEMPAR	2014/12	XL:			F08	MPI	64	(OpenMP)		64: 1,756,001		W	N/A
Gysela	2013/06	XL:	C		F90	MPI		OMP+pthrd		64: 1,835,008		W	(HDF5)
hp-fRG	2016/10	XL:	C	C++		MPI	1	OpenMP	32	32: 917,504	S		N/A
*ICON	2015/02	XL:	C		Ftn	MPI	1	OpenMP	64	64: 1,835,008	S		(netCDF)
IMD	2014/10	XL:	C			MPI	64	(OpenMP)		64: 1,835,008		W	*unspecified*
JURASSIC	2014/05	XL:	C			MPI	32	OpenMP	2	64: 1,835,008		W	netCDF
JuSPIC	2013/10	GCC:			F90	MPI	4	OpenMP	16	64: 1,835,008	S		MPI-IO, POSIX
*KKRnano	2014/10	XL:			F03	MPI	4	OpenMP	16	64: 1,835,008	S		SIONlib
LAMMPS-DCM	2015/04	XL:		C++		MPI	4	OpenMP	16	64: 1,835,008	S	W	N/A
*MPAS-A	2017/03	XL:	C		Ftn	MPI	16	(OpenMP)		16: 458,752	S		SIONlib,PIO,pNetCDF
MP2C	2013/11	XL:			Ftn	MPI	32			32: 917,504		W	SIONlib
muPhi ($\mu\phi$)	2013/11	XL:		C++		MPI	32			32: 917,504		W	SIONlib
Musubi	2014/12	XL:			F03	MPI		OpenMP		32: 917,504	?	?	N/A
NEST	2013/11	XL:		C++		MPI	1	OpenMP	64	64: 1,835,008		W	(SIONlib)
OpenTBL	2014/05	XL:			F03	MPI		OpenMP		64: 1,835,008		W	pHDF5
*ParFlow+p4est	2017/03	XL:	C		Ftn	MPI	16			16: 458,752	S		MPI-IO
*pe	2017/03	GCC:		C++		MPI	64			64: 1,835,008		W	N/A
PEPC	2013/06	GCC:	C		F03	MPI	1	pthreads	61	61: 1,744,992		W	(SIONlib,MPI-IO,vtk)
PMG+PFASST	2013/08	XL:	C		F03	MPI	16	(pthreads)		16: 458,752	S		N/A
PP-Code	2014/08	XL:			F90	MPI		OpenMP		64: 1,835,008	S	W	*unspecified*
*psOpen	2015/02	XL:			F90	MPI	32	OpenMP	2	64: 1,835,008	S		pHDF5
*SHOCK	2015/02	XL:	C			MPI	64			64: 1,835,008	S	W	(cgns/HDF5)
*SLH	2016/02	XL:	C		F95	MPI	16	OpenMP	4	64: 1,835,008	S		MPI-IO
TERRA-NEO	2013/06	XL:		C++	Ftn	MPI		OpenMP		64: 1,835,008		W	*unspecified*
waLBerla	2013/06	XL:		C++		MPI		OpenMP		64: 1,835,008	?	?	N/A
ZFS	2015/03	Clang:		C++		MPI	16	OpenMP	2	32: 917,504	S		(pNetCDF)

it opened in June 2013, and another group of five codes added as a result of the Extreme Scaling Workshop in February 2015, otherwise membership submission requests and acceptance were more sporadic and generally diminishing towards 2018 when *JUQUEEN* is due to be decommissioned. Throughout this period no particular trends seem discernable in the characteristics of the accepted codes.

Member codes are typically able to run in a variety of configurations, which may trade-off different capabilities and indeed evolve as the code continues to be developed, however, this analysis provides a snapshot at the point of qualification for membership.

Programming Languages. IBM provide their XL suite of optimising compilers for C, C++, and Fortran which feature support for automatic SIMD vectorisation and QPX vector intrinsics, as well as transactional memory (TM) and speculative execution (SE), though relatively few codes have investigated or currently exploit these capabilities. GCC and LLVM/Clang compilers are also available, often providing support for newer language standards (e.g., C++11) and non-standard extensions which can ease porting, or in some cases delivering better performance: these have been exploited by several High-Q member codes (JuSPIC, pe, PEPC & ZFS).

Since Blue Gene/Q offers lower-level function calls for some hardware-specific features that are sometimes not available for all programming languages, a starting point is looking at the languages used. Figure 2 (left) shows a Venn set diagram of the main programming language(s) used, i.e. languages used by the parallel application itself and not its auxiliary libraries or pre/post-processing components. It indicates that all three major programming languages are roughly equally popular (without considering lines of code). Seven combine Fortran with C, three used C++ and C, one Fortran and C++, and the remainder exclusively used a single language. Notably, Python and similar languages relying on dynamic linking have not been used

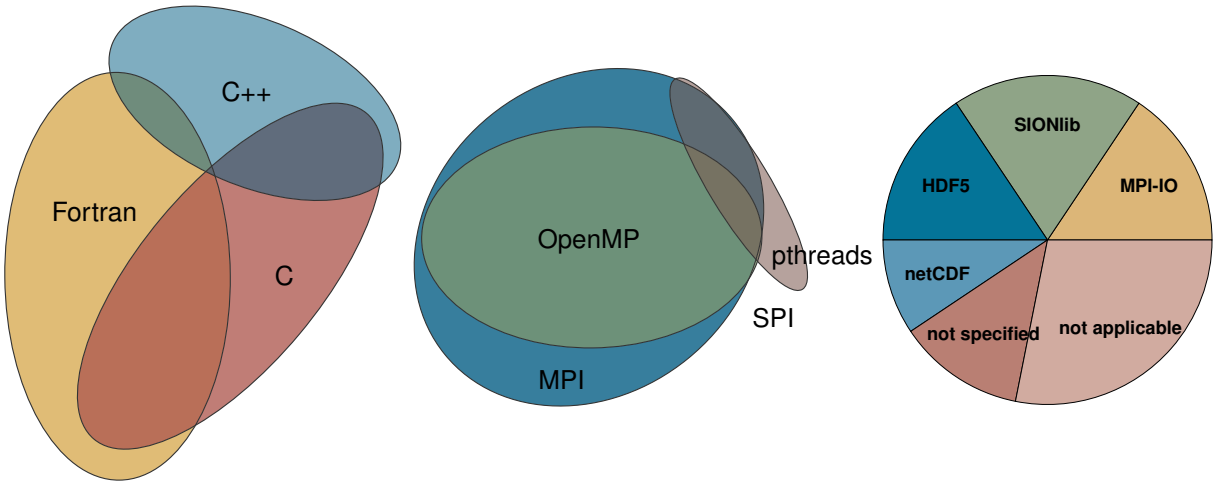


Figure 2. Left: Venn set diagram of main programming languages used by High-Q Club member codes. Middle: Venn set diagram of parallelisation modes of High-Q Club member codes run on *JUQUEEN*. Right: Pie-chart showing file I/O as available in High-Q Club member codes

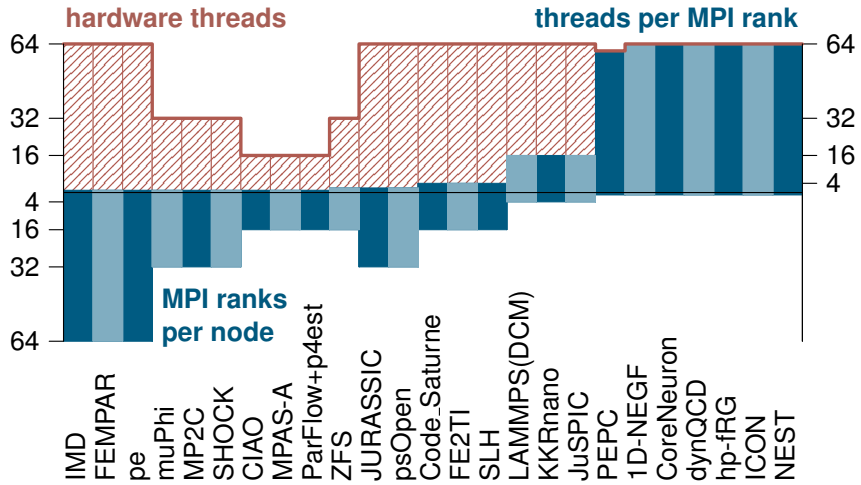


Figure 3. Bar chart of MPI ranks per node and threads per MPI rank, determining the number of hardware threads exploited, for executions of High-Q Club member codes on *JUQUEEN*

by any High-Q Club member code, even though tools like *Spindle*[6] were developed to address performance issues with loading shared libraries [12].

Optimised libraries are also provided by IBM for BG/Q, that are often both more convenient and performant than self-written versions. Only in a few special cases High-Q codes used their own libraries, e.g., psOpen for non-blocking 3D Fast Fourier Transforms [17]. Issues which initially inhibited scaling of ParFlow were addressed by employing the p4est library for mesh partitioning and an updated version of the HYPRE library of linear solvers [8].

Parallelisation Modes. The four hardware threads per core of the Blue Gene/Q chip in conjunction with the limited amount of memory recommend to make use of multi-threaded programming. It is therefore interesting to see whether this is indeed the preferred programming model and whether available memory is an issue. Figure 2 (middle) shows a Venn set diagram of the parallelisation modes used, revealing that mixed-mode programming does indeed dominate.

[6] https://computation.llnl.gov/project/spindle/

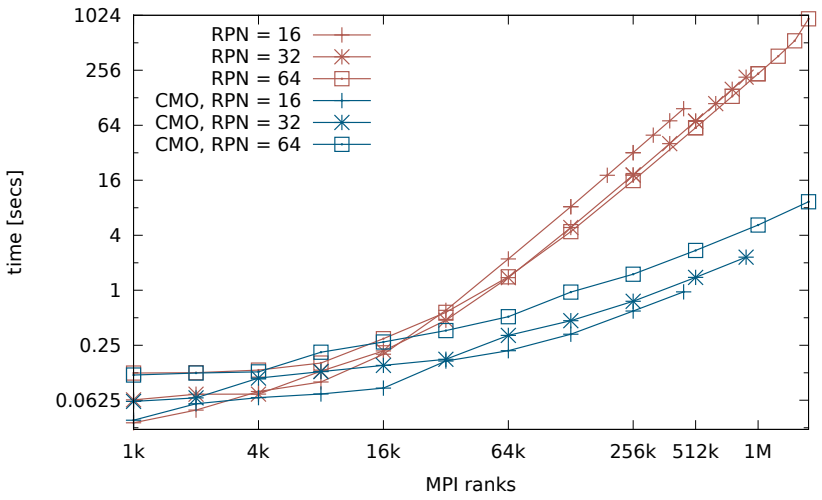


Figure 4. Scaling of wallclock execution time for `MPI_Comm_split` sub-communicator creation on *JUQUEEN* Blue Gene/Q: default and less memory-optimised alternative (CMO)

IBM provide an optimised implementation of MPI, which is almost ubiquitous for portable distributed-memory parallelisation, such that only one High-Q code (dynQCD) found it worthwhile to program an alternate version that directly used the underlying machine-specific SPI primitives. While a number of High-Q codes have demonstrated that they can scale when using only MPI ("MPI everywhere"), this requires adapting to very limited per-rank memory (i.e., less than 256 MiB/rank with 64 ranks per node) and switching to MPI communicator management routines that are optimised to reduce execution time rather than memory utilisation[7], further reducing memory available to the application itself. Figure 4 compares the execution time of both versions.

Ten codes exclusively used MPI for their scaling runs, both between and within compute nodes, accommodating to the restricted per-process memory and even trading higher memory requirements for faster MPI communicator management: this allowed FEMPAR to reduce sub-communicator creation time (`MPI_Comm_split`) from 15 minutes to under 10 seconds.

Convenient and portable multi-threading within compute nodes is supported via OpenMP, though some newer directives are not always optimised (e.g., for tasking). Only a few High-Q codes have pursued using POSIX threads (pthreads) for additional multi-threading control (dynQCD, Gysela, PEPC & PMG+PFASST). Almost all High-Q codes use MPI only from a single thread (`MPI_THREAD_FUNNELED`), with at least Gysela, ICON, and PEPC requiring `MPI_THREAD_MULTIPLE` which internally needs to use locks for synchronisation and precludes use of hardware support in the torus network for collective operations.

The majority of High-Q Club codes successfully employ OpenMP multi-threading to exploit compute node shared memory in conjunction with MPI. A memory fragmentation issue in a third-party library inhibited the use of OpenMP by FEMPAR, problems with nested parallel regions blocked MPAS-A, and an earlier investigation with the SHOCK code found this not to be beneficial. CoreNeuron has an ongoing effort investigating use of OpenMP-3 tasking and new MPI-3 capabilities (e.g. non-blocking collectives) are under consideration, so these are generally expected to become increasingly important.

[7] via setting the `PAMID_COLLECTIVES_MEMORY_OPTIMIZED` environment variable

File I/O Libraries. Figure 2 (right) shows a pie-chart breakdown of the I/O libraries used by High-Q Club codes, although in most cases writing output and in some cases reading input files was disabled for their large-scale executions, and synthesised or replicated data was used instead. Some of the (early) submissions for the High-Q Club unfortunately did not specify their file I/O usage. One quarter of the High-Q Club codes can use either (p)HDF5 or (p)NetCDF, despite their often disappointing performance, whereas one-sixth can use MPI file I/O directly. 20% of High-Q Club codes have migrated to using *SIONlib* for effective parallel I/O (see Section 6.1).

Compute Node Memory. For CoreNeuron available memory was the limiting factor for larger simulations, with the current limit being 155 million neurons using 15.9 GiB of RAM. The other neuroscience code NEST was similarly constrained by the amount of compute node memory available to store simulation data, however, ultimately managed to simulate 645 million neurons on *JUQUEEN* when using all available cores. MPAS-A required 1 GiB of memory on each process for its regular 3 km mesh simulation (over 65 million grid cells with 41 vertical levels), and could therefore only use a single hardware thread per core, limiting its effective performance. Using all four hardware threads of each processor code, FEMPAR was able to increase its efficiency and scalability to 1.75 million processes using $27\frac{1}{2}$ racks of *JUQUEEN* when employing an additional (fourth-)level of domain decomposition.

Concurrency. Figure 3 shows the relation between the number of MPI ranks and threads per node. On either side of this diagram are the two extremes of using all 64 hardware threads on each CPU by either 64 MPI ranks or 64 OpenMP/POSIX threads. Whereas multiples of two matching the available hardware were generally employed, PEPC delivered its best performance with the rather unusual number of 61 POSIX threads. Hatching shows the resulting number of hardware threads used by the codes, i.e. the concurrency. Clearly, codes benefit from using more hardware threads than physical cores and favour this configuration.

5. Comparison of Code Scalability

An overview of application code execution time scaling on *JUQUEEN* entails comparison of achievements in *strong* (fixed total problem size), and *weak* (fixed problem size per process or thread) scaling. This section reviews execution performance of submissions that qualified for membership. Various member codes continued to improve their performance and scalability (or that of additional functionality) beyond that of their qualifying submission.

A significant spread in execution results, and diverse scaling characteristics of the codes are visible in Fig. 5. A single BG/Q rack was chosen to provide a convenient baseline for scaling, however, a (half-rack) mid-plane would also have offered an isolated dedicated resource for this purpose. For strong scaling a minimum of three measurements spanning a factor of four in size up to the full configuration of 28 racks was mandated. Note that in many cases timings do not have a common baseline of one rack since datasets sometimes did not fit available memory, or no data was provided for 1,024 compute nodes: for strong scaling an execution with a minimum of seven racks (one quarter of *JUQUEEN*) is accepted for a baseline, and perfect scaling assumed from a single rack to the baseline. While full-system runs have a dedicated allocation of all 28 racks, other measurements were generally done when *JUQUEEN* is operational with a full and varying workload (which particularly impacts parallel I/O to the shared GPFS filesystem).

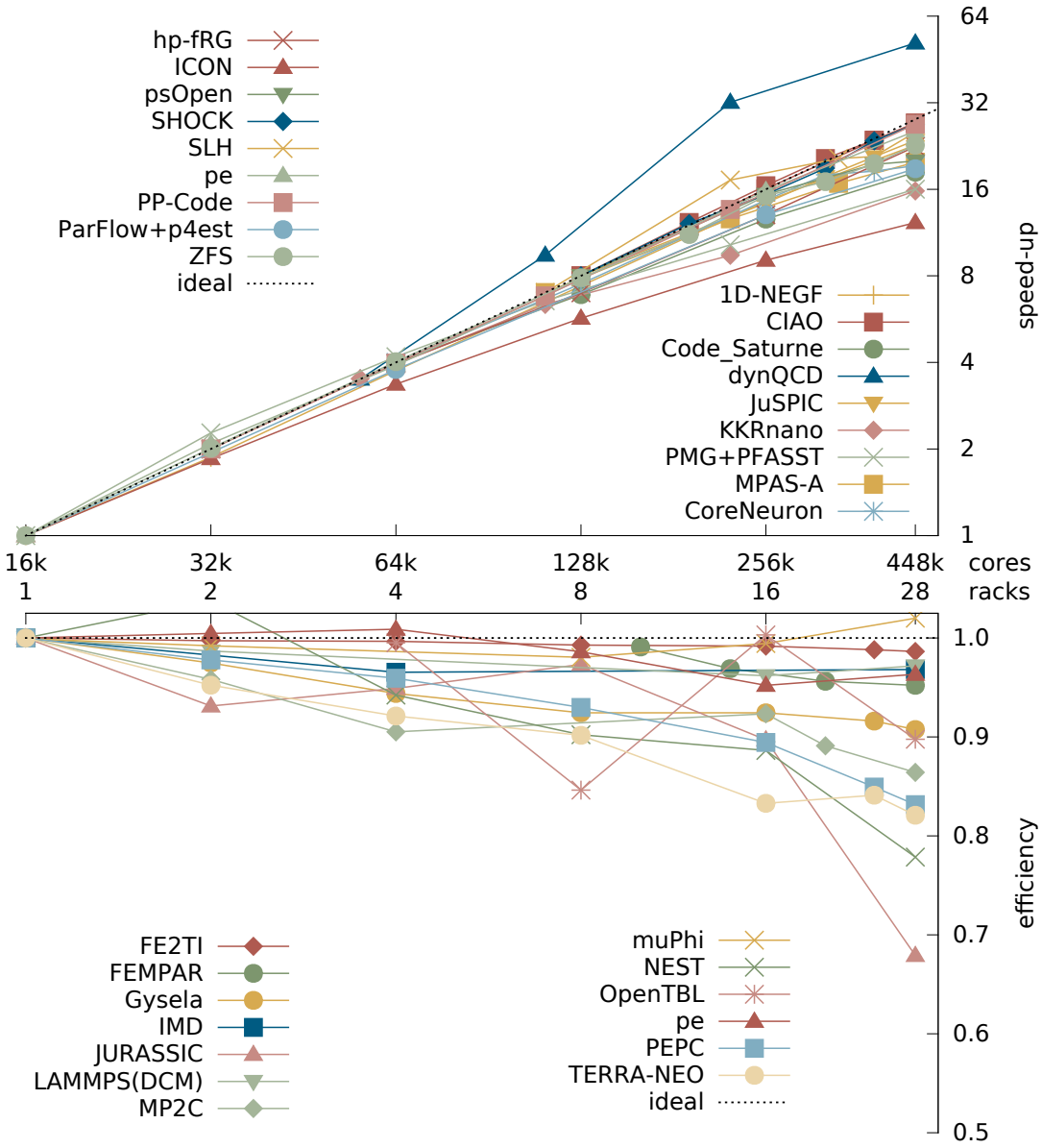


Figure 5. Strong and weak scaling of High-Q Club member application codes on *JUQUEEN*. Compared to a single rack, ideal strong scaling has a speed-up of 28× on 28 racks and weak scaling has 100% efficiency. Of the 18 codes showing strong scalability, ten maintained scaling efficiency above 80%, and only three were below 65% efficiency

Many High-Q Club member codes demonstrated very good strong-scaling speed-up close to 28×, with dynQCD standing out with superlinear speed-up of 52× due to its exceptional ability to exploit caches as problem size per thread decreases. ICON only achieved a modest 12× speed-up, and while this is less than 50% of the ideal, clear reduction in overall time to completion was shown.

Size of dataset was often critical for successful strong scaling to 28 racks, as diminishing per-rank computation can be overwhelmed by growing communication costs. Scaling of MPAS-A with a dataset of 65 million grid cells was only demonstrated to 24 racks (with worse performance for 28 racks), however, simulations using a 2 km global mesh with more than 147 million grid

cells scaled to the full 28 racks. Several codes switched to lower-precision (32-bit instead of 64-bit) datatypes to allow them to fit larger simulations in available memory.

Weak scaling is generally easier, as shown by the High-Q Club member codes maintaining over 80% efficiency from a single to 28 racks. JURASSIC only managed 68% efficiency, due to excessive I/O for the reduced-size test case, which was the lowest accepted for club membership, whereas muPhi was able to achieve 102% efficiency on 28 racks.

Various codes show erratic scaling performance, most likely due to topological effects. SHOCK is characterised by particularly poor configurations with an odd number of racks in one dimension (i.e. 4×3, 4×5 and 4×7). Similarly, OpenTBL shows marked efficiency drops for non-square numbers of racks (8 and 28).

Most optimisations employed by the codes are not specific to Blue Gene (or BG/Q) systems, but can also be exploited on other highly-parallel systems. High-Q Club codes have also run at scale on various Cray supercomputers, K computer, *MareNostrum-III*, *SuperMUC* and other x86-based computers, as well as on systems with GPGPUs [3].

6. Supporting Tools and Libraries

A variety of tools and libraries were invaluable during application tuning and scaling on *JUQUEEN*. Particularly during workshops, *LLview*[8] (shown in the left part of Fig. 1) facilitated monitoring the current system usage and additionally showing job energy consumption and file I/O performance. Custom mappings of MPI process ranks to *JUQUEEN* compute nodes generated by the *Rubik*[9] tool were investigated with psOpen and found to deliver some benefits, however, for the largest machine partitions these did not provide the expected reduction in communication times yet suffered from greatly increased application launch/initialisation time.

Efficient parallel file I/O libraries and performance analysis tools both delivered significant benefits for many applications, specifically when addressing extreme scalability.

6.1. Managing Parallel File I/O

A critical point attracting increasing attention is performance of file I/O, which is often a scalability constraint for codes which need to read and write huge datasets or open a large number of files. Large-scale executions of various High-Q member codes using the popular HDF5 and pNetCDF libraries needed to disable file I/O and synthesise initialisation data, e.g., CoreNeuron replicated a small dataset to fill memory to 15.9 GiB.

Initial full-scale runs of MPAS-A needed 20 minutes to load its initial condition data of 1.2 TiB using PIO/NetCDF and simulation output was disabled for large-scale tests to avoid similar writing inefficiency. This deficiency was subsequently addressed by adopting SIONlib to improve file I/O, allowing MPAS-A to load 1.4 TiB of finer resolution mesh data and output 4 TiB of model data to disk at three stages of the model run. Benefits were also observed on other systems, i.e. MPAS-A reported a 10x speed-up from PIO/NetCDF on 1,024 nodes or more on SuperMUC [7].

The *SIONlib*[10] library for parallel task-local file I/O, was specifically developed to address such file I/O scalability limitations [11, 13]. It has been used effectively by four High-Q codes

[8] http://www.fz-juelich.de/jsc/llview
[9] https://computation.llnl.gov/project/performance-analysis-through-visualization/software.php
[10] http://www.fz-juelich.de/jsc/sionlib/

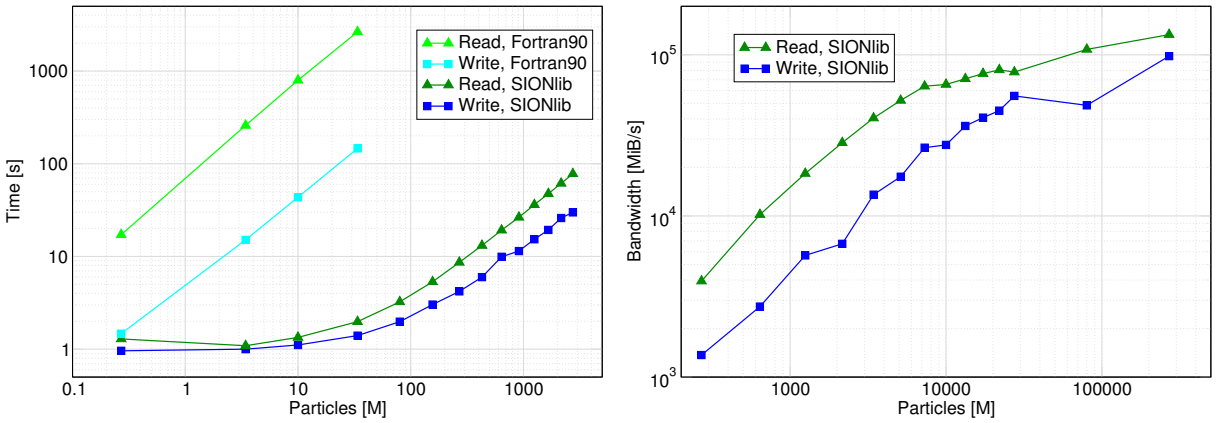


Figure 6. Left: MP2C file I/O time on one *JUQUEEN* midplane (8,192 MPI ranks) using traditional F90 I/O compared to improved I/O with *SIONlib*. Right: MP2C file I/O bandwidth for checkpointing executions with 1.8 million MPI ranks on 28 racks of *JUQUEEN* using *SIONlib* for reading and writing particle data

(KKRnano, MPAS-A, MP2C and muPhi) and several other applications are investigating migrating to adopt it (e.g. NEST [32]).

The scalability limitations of naïve parallel file management is demonstrated by I/O optimization results from the integration of *SIONlib* into MP2C [9]. MP2C couples multiple-particle collision dynamics with molecular dynamics to implement mesoscale simulation of hydrodynamic media. MP2C uses a large number of particles in its calculation, which have to be read from files in the initialisation phase and stored in files at the end of a simulation run in restart files. Furthermore, MP2C regularly does I/O to checkpoint files to be able to restart after system or program failure.

Originally, MP2C implemented I/O to a single file storing all particle data, which prevents bottlenecks which arise when using a large number of task-local files. However, data output was implemented with standard Fortran90 I/O calls for writing in two steps by first collecting the data on one task and then writing the data in a second step from that task to file. Data input is done individually by each MP2C MPI rank, reading the whole input file and selecting those particles that will be needed in the local domain. As both approaches limit the scalability of data input and output, I/O had to be improved for using it at large scale on *JUQUEEN*. Figure 6 (*Fortran90 write and read*) shows the results of initial benchmark runs on one BG/Q midplane with 8,192 MPI ranks using the traditional I/O approach. The measurements at the small scale of one midplane show the increasing time for I/O operations which hinders MP2C from using more than 50 million particles. However, the algorithm and its memory requirements allow orders of magnitude larger numbers of particles.

SIONlib supports efficient parallel file I/O for applications which use task-local I/O to individual files for each MPI rank. It was easily integrated into MP2C by replacing Fortran90 file open and close calls by corresponding *SIONlib* collective calls and write and read operations of local particle data with *SIONlib* I/O operations. After changing approximately 50 lines of code, we could reduce MP2C I/O time by several orders of magnitude. Furthermore, we could scale MP2C's I/O on one midplane to more than 4 billion particles, which is 125 times more than in the traditional approach (see *SIONlib write and read* in Fig. 6).

With integration of *SIONlib* into MP2C, it now runs with enabled I/O at the full scale of *JUQUEEN*. Figure 6 shows the I/O bandwidth for reading and writing different numbers of

particles. Data was stored on the GPFS scratch filesystem which provides a theoretical I/O bandwidth of 200 GiB/s. MP2C could achieve at the largest scale about 100 GiB/s for writing and more than 130 GiB/s for reading, which is 50–66 % of the peak bandwidth. For the largest number of particles (270 trillion), the restart file of MP2C was 14.4 TiB which could be written in 147 s (and read in 108 s).

6.2. Parallel Performance Analysis

During their development and specifically when preparing for large-scale runs on *JUQUEEN*, the execution performance of various High-Q Club applications was investigated with open-source tools. While most analyses are adequate at modest scales (e.g., a single BG/Q rack with 64k processes/threads), occassionally it is necessary to investigate performance issues that only manifest at larger scales.

The *Darshan*[11] tool was typically most convenient for low-overhead measurement and analysis of file I/O, distinguishing MPI file I/O from underlying POSIX file I/O per file and detailed breakdown of operation counts and I/O sizes. This was complemented by the *Scalasca*[12] toolset for scalable performance analysis of large-scale parallel applications [15] which is widely deployed on some of the largest HPC systems and clusters, supporting runtime summarization and event trace analyses of MPI, and OpenMP primarily focus on locating and quantifying communication and synchronization inefficiencies in C/C++/Fortran applications. Scalasca uses its own parallel trace tools with the community-developed *Score-P*[13] instrumentation and measurement infrastructure [24], itself based on OTF2 event trace and CUBE profile libraries. *SIONlib* is employed for efficient large-scale parallel file I/O when writing and reading OTF2 trace files (e.g., handling one container file per BG/Q IONode).

Scalasca measurements of applications running with 1.8 million threads on *JUQUEEN* have been done where 64 OpenMP threads are used for a single MPI process on each processor. In such an execution configuration the 16 GiB of processor memory is adequate to store profile and execution trace data collected during measurement for unification and collation during finalisation. This is not the case when 32 or 64 MPI processes split the available processor memory, along with memory required by the MPI library and application itself, restricting measurements to smaller configurations.

Parallel execution profiles are by default based on full compiler instrumentation of application user-level source routines, combined with OPARI2 instrumentation of OpenMP constructs and PMPI interposition on MPI library routines. Where these initial profile measurements manifest notable execution time dilation, filtering during measurement or selective instrumentation can be employed, directed by scoring which assesses event frequencies and associated measurement overheads. Iterative refinement of instrumentation and measurement of profiles is essential prior to collecting event traces where overheads are more significant, particularly with respect to in-memory buffering during measurement and final trace sizes.

Event traces are analysed by *Scalasca* trace tools with a parallel replay following measurement within the allocated partition, using the same configuration of MPI processes and OpenMP threads, to determine the fraction of OpenMP and MPI time due to waiting, the origins of delays, and critical execution path. Since traces are loaded entirely in memory for forward and

[11] `http://www.mcs.anl.gov/darshan`
[12] `http://www.scalasca.org/`
[13] `http://www.score-p.org/`

backward event replays, which require additional data-structures and pointers, the number of recorded events similarly governs the size of trace that can be analysed. (While analysis of smaller execution configurations may employ larger partitions with more memory, this option is not possible for event traces from the full system.)

Finally, the minimal set of metrics provided in *Score-P* profiles and *Scalasca* trace analysis reports are subsequently post-processed by a remapper which derives a large number of additional metrics and hierarchies. Since the CUBE remapper and GUI are serial processes requiring large amounts of RAM to process metrics in memory (and to minimise expensive paging to disk), generally these are best done on dedicated visualisation nodes, such as those of the JSC general-purpose Linux cluster *JURECA* with 1 TiB shared RAM.

During an Extreme Scaling Workshop, *Scalasca* helped identify a critical performance issue that manifest at large scale with a version of the NEST application when it was importing 1.9 TiB of neuron and synapse data with HDF5 configured to use collective MPI file I/O. To avoid IBM XL C++ compiler instrumentation overhead, manual annotation of the relevant code regions was used to augment the instrumentation of OpenMP and MPI. Measurement of an execution with 16 OpenMP threads for 28,672 MPI ranks (458,752 threads in total) revealed a large imbalance in MPI File I/O which was mirrored in the following OpenMP parallel region. Instead of the expected MPI collective file I/O, much less efficient individual file I/O was found which originated from a mismatch between the import module's data structure and the HDF5 file objects [5]. After suitably modifying the import data structure to match the HDF5 file object, the imbalance was eliminated, and performance greatly improved [32].

The cost of MPI collective file writing for final simulation output of the CIAO application on *JUQUEEN* with 458,752 MPI ranks was also identified by *Scalasca* as a key performance limitation and motivation for *JUSITU* coupling in situ visualization to VisIt [16] as previously incorporated in psOpen and ZFS.

Exponentially growing memory requirements of ParFlow were located to originate from version of the HYPRE library used by its by preconditioner using memory allocation profiling [8]. Examination of *Scalasca* execution traces was key to determining optimal load-balancing of MPI and OpenMP computations, and associated workload distribution and loop scheduling strategies, to allow hp-fRG to scale effectively to use all of the *JUQUEEN* compute nodes with 1.8 million threads [31].

Conclusions

As the highly productive operation of the *JUQUEEN* Blue Gene/Q by Jülich Supercomputing Centre draws to a close in spring 2018, the High-Q Club documents 32 codes from a wide range of HPC application fields that demonstrated effective extreme-scale execution using its entire 458,752 cores (and often 1.8 million threads). Standard programming languages and MPI combined with multi-threading was sufficient, and provided a straightforward migration path for application developers which has also delivered performance and scalability benefits on diverse HPC computer systems (including K computer, Cray supercomputer systems and other clusters). Similar ease-of-use and reliability of well-established homogeneous Blue Gene/Q systems probably cannot be expected to be representative of the current and future generations of heterogeneous HPC systems, however, we believe it is a worthwhile goal.

Each of the High-Q Club member codes is quite distinct, encountering and resolving a variety of often unique impediments in scaling to the full *JUQUEEN* configuration. Code teams

themselves ultimately determine whether and how to address the considerable challenges, with the High-Q Club promoting successes. Engagement of experts from JSC involved close long-term collaborations in some cases to very little in others. Extreme scaling workshops provided a brief opportunity for particularly intense interaction and experimentation, which benefits many code teams.

Extreme scaling on *JUQUEEN* generally required adapting to the limited compute node memory, either via employing alternate communicator management optimised for large numbers of MPI ranks or effective exploitation of OpenMP multi-threading in a mixed-mode configuration. Often file I/O is not done in a scalable fashion, requiring many codes to forfeit I/O (and use synthetic or replicated simulation data) for their large-scale runs. Application codes which are leaner and less restrictive in their memory and I/O usage can be desirable as they can exploit more affordable systems, however, each code has its own requirements and constraints. High-Q Club member codes are those which were able to adapt, but many important codes may not be so fortunate. Despite these limitations, notable extreme-scale simulation capabilities were demonstrated and led to subsequent Big Blue Gene Weeks with prioritised production executions.

The High-Q Club was entirely neutral as to how application codes achieve qualifying scalability. JSC training and consultancy introduce and consider a variety of technologies and techniques (including novel programming models and languages) from which application developers themselves decide which to pursue based on their individual cost-benefit determination. Our assessment of the High-Q Club member code characteristics shows that incremental changes were convenient for a wide variety of codes. The absence of disruptive approaches can perhaps be explained by the additional effort required and associated technology immaturity at this time.

While focussing on scalability to the entire 28 racks of the *JUQUEEN* BG/Q installed at JSC was a natural choice, it is also rather arbitrary. Loosening the High-Q Club qualification criteria to accept scaling to 85% (or perhaps even 70%) of the entire *JUQUEEN* system could also have been justifiable and still offer value in distinguishing extreme-scaling codes. Additional credit may also have been appropriate for executions that are representative of production configurations including associated file I/O, and possibly other desirable aspects (such as node-level optimisation via vectorisation and core over-subscription). Greater differentiation between the High-Q Club member codes is also desirable, particularly for better insight into readiness for capability-mode production on current leadership systems and expected future exascale systems. These aspects will be re-considered for the successor to the High-Q Club.

Acknowledgements

We would like to thank the numerous application code-teams who participated in Extreme Scaling Workshops at JSC and contributors to the High-Q Club for generously sharing their experience, identifying performance and scalability inhibitors and effective solutions. We also recognise the invaluable assistance provided by the JSC Simulation Laboratories, Cross-Sectional Teams, system administrators, and *JUQUEEN* support staff.

References

1. Allock, W., Bacon, C., Bailey, A., Bair, R., et. al.: Blue Gene/Q: *Sequoia* and *Mira*. In: Vetter, J. (ed.) Contemporary High Performance Computing: From Petascale toward Exascale. pp. 225–282. Chapman & Hall/CRC (2013)

2. Attig, N., Docter, J., Frings, W., Grotendorst, J., Gutheil, I., Janetzko, F., Mextorf, O., Mohr, B., Stephan, M., Wolkersdorfer, K., Wollschläger, L., Krieg, S., Lippert, T.: Blue Gene/P: *JUGENE*. In: Vetter, J. (ed.) Contemporary High Performance Computing: From Petascale toward Exascale. pp. 153–188. Chapman & Hall/CRC (2013)

3. Brömmel, D., Frings, W., Wylie, B.: MAXI – Multi-system Application Extreme-scaling Imperative. In: Joubert, G.R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.) Parallel Computing: On the Road to Exascale. vol. 27, pp. 765–766. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-765

4. Brömmel, D., Frings, W., Wylie, B.J.N.: JUQUEEN Extreme Scaling Workshop 2015. Tech. Rep. FZJ-JSC-IB-2015-01 (2015), `http://juser.fz-juelich.de/record/188191`, accessed: 2018-03-21

5. Brömmel, D., Frings, W., Wylie, B.J.N.: Extreme-scaling Applications En Route to Exascale. In: Proceedings of the Exascale Applications and Software Conference 2016. pp. 1:1–1:10. EASC '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2938615.2938616

6. Brömmel, D., Frings, W., Wylie, B.J.N.: JUQUEEN Extreme Scaling Workshop 2016. Tech. Rep. FZJ-JSC-IB-2016-01 (2016), `http://juser.fz-juelich.de/record/283461`, accessed: 2018-03-21

7. Brömmel, D., Frings, W., Wylie, B.J.N.: JUQUEEN Extreme Scaling Workshop 2017. Tech. Rep. FZJ-JSC-IB-2017-01 (2017), `http://juser.fz-juelich.de/record/828084`, accessed: 2018-03-21

8. Burstedde, C., Fonseca, J.A., Kollet, S.: Enhancing speed and scalability of the ParFlow simulation code. Computational Geosciences 22(1), 347–361 (2018), DOI: 10.1007/s10596-017-9696-2

9. Freche, J., Frings, W., Sutmann, G.: High-Throughput Parallel-I/O using *SIONlib* for Mesoscopic Particle Dynamics Simulations on Massively Parallel Computers. In: Chapman, B., Desprez, F., Joubert, G.R., Lichnewsky, A., Peters, F., Priol, T. (eds.) Parallel Computing: From Multicores and GPU's to Petascale. vol. 19, pp. 371–378. IOS Press (2010), DOI: 10.3233/978-1-60750-530-3-371

10. Frings, W., Mohr, B., Orth, B.: Report on the Jülich Blue Gene/L Scaling Workshop 2006. Tech. Rep. FZJ-ZAM-IB-2007-02, Jülich (2007), `http://juser.fz-juelich.de/record/55967`, accessed: 2018-03-21

11. Frings, W.: Efficient Task-Local I/O Operations of Massively Parallel Applications. Ph.D. thesis, RWTH Aachen University, Jülich (2016), `http://juser.fz-juelich.de/record/811621`, accessed: 2018-03-21

12. Frings, W., Ahn, D.H., LeGendre, M., Gamblin, T., de Supinski, B.R., Wolf, F.: Massively Parallel Loading. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. pp. 389–398. ICS '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2464996.2465020

13. Frings, W., Wolf, F., Petkov, V.: Scalable Massively Parallel I/O to Task-local Files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 17:1–17:11. SC '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1654059.1654077

14. Gageik, M., Klioutchnikov, I., Olivier, H.: Mesh study for a direct numerical simulation of the transonic flow at $\mathrm{Re}_c = 500,000$ around a NACA 0012 airfoil. Computers & Fluids 122, 153–164 (2015), DOI: 10.1016/j.compfluid.2015.08.030

15. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca Performance Toolset Architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (2010), DOI: 10.1002/cpe.1556

16. Göbbert, J.H., Bode, M., Wylie, B.J.N.: Extreme-Scale In Situ Visualization of Turbulent Flows on IBM Blue Gene/Q JUQUEEN. In: Taufer, M., Mohr, B., Kunkel, J.M. (eds.) High Performance Computing. pp. 45–55. Springer International Publishing, Cham (2016), DOI: 10.1007/978-3-319-46079-6_4

17. Göbbert, J., Gauding, M., Ansorge, C., Hentschel, B., Kuhlen, T., Pitsch, H.: Direct numerical simulation of fluid turbulence at extreme scale with psOpen. In: Gerhard, R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.) Parallel Computing: On the Road to Exascale. vol. 27, pp. 777–785. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-777

18. Hammer, N., Jamitzky, F., Satzger, H., et al.: Extreme scale-out SuperMUC phase 2 – lessons learned. In: Gerhard, R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.) Parallel Computing: On the Road to Exascale. vol. 27, pp. 827–836. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-827

19. Heinzeller, D., Duda, M.G., Kunstmann, H.: Towards convection-resolving, global atmospheric simulations with the Model for Prediction Across Scales (MPAS) v3.1: an extreme scaling experiment. Geoscientific Model Development 9(1), 77–110 (2016), DOI: 10.5194/gmd-9-77-2016

20. IBM Corporation: IBM System Blue Gene Solution Blue Gene/Q application development. `http://www.redbooks.ibm.com/`, accessed: 2018-03-21

21. Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., Diesmann, M., Kunkel, S.: Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. Frontiers in Neuroinformatics 12 (2018), DOI: 10.3389/fninf.2018.00002

22. Jülich Supercomputing Centre: The High-Q Club. `http://www.fz-juelich.de/ias/jsc/high-q-club`, accessed: 2018-03-21

23. Klawonn, A., Lanser, M., Rheinbach, O.: $\mathrm{FE}^2\mathrm{TI}$: Computational scale bridging for dual-phase steels. In: Joubert, G.R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.)

Parallel Computing: On the Road to Exascale. vol. 27, pp. 797–806. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-797

24. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Tools for High Performance Computing 2011. pp. 79–91. Springer, Berlin, Heidelberg (2012), DOI: 10.1007/978-3-642-31476-6_7

25. Mohr, B., Frings, W. (eds.): Jülich Blue Gene/P Porting, Tuning & Scaling Workshop 2008, Innovatives Supercomputing in Deutschland (InSiDE), vol. 6 (2008), http://inside.hlrs.de/_old/htm/Edition_02_08/article_28.html, accessed: 2018-03-21

26. Mohr, B., Frings, W.: Jülich Blue Gene/P Extreme Scaling Workshop 2009. Tech. Rep. FZJ-JSC-IB-2010-02, Jülich (2010), http://juser.fz-juelich.de/record/8924, accessed: 2018-03-21

27. Mohr, B., Frings, W.: Jülich Blue Gene/P Extreme Scaling Workshop 2010. Tech. Rep. FZJ-JSC-IB-2010-03, Jülich (2010), http://juser.fz-juelich.de/record/9600, accessed: 2018-03-21

28. Mohr, B., Frings, W.: Jülich Blue Gene/P Extreme Scaling Workshop 2011. Tech. Rep. FZJ-JSC-IB-2011-02, Jülich (2011), http://juser.fz-juelich.de/record/15866, accessed: 2018-03-21

29. Ovcharenko, A., Kumbhar, P., Hines, M., Cremonesi, F., Ewart, T., Yates, S., Schürmann, F., Delalondre, F.: Simulating morphologically detailed neuronal networks at extreme scale. In: Joubert, G.R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.) Parallel Computing: On the Road to Exascale. vol. 27, pp. 787–796. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-787

30. Qi, J., Jain, K., Klimach, H., Roller, S., Schürmann, F., Delalondre, F.: Performance evaluation of the LBM solver Musubi on various HPC architectures. In: Joubert, G.R., Leather, H., Parsons, M., Peters, F., Sawyer, M. (eds.) Parallel Computing: On the Road to Exascale. vol. 27, pp. 807–816. IOS Press (2016), DOI: 10.3233/978-1-61499-621-7-807

31. Rohe, D.: Hierarchical Parallelisation of Functional Renormalisation Group Calculations – hp-fRG. Computer Physics Communications 207, 160–169 (2015), DOI: 10.1016/j.cpc.2016.05.024

32. Schumann, T., Frings, W., Peyser, A., Schenck, W., Thust, K., Eppler, J.M.: Modeling the I/O behavior of the NEST simulator using a proxy. In: Conference Proceedings of the YIC GACM 2015 / ed.: Stefanie Elgeti; Jaan-Willem Simon. 3rd ECCOMAS Young Investigators Conference, Aachen (Germany), 20–23 July 2015, RWTH Aachen University (2015), http://juser.fz-juelich.de/record/202952, accessed: 2018-03-21

33. Springer, P., Ismail, A.E., Bientinesi, P.: A Scalable, Linear-Time Dynamic Cutoff Algorithm for Molecular Dynamics. In: Kunkel, J.M., Ludwig, T. (eds.) High Performance Computing. pp. 155–170. Springer International Publishing, Cham (2015), DOI: 10.1007/978-3-319-20119-1_12

34. Stephan, M., Doctor, J.: JUQUEEN: IBM Blue Gene/Q supercomputer system at the Jülich Supercomputing Centre. Journal of Large-Scale Research Facilities 1, 1–5 (2015), DOI: 10.17815/jlsrf-1-18

DOI: 10.14529/jsfi180105

Exploiting the Performance Benefits of Storage Class Memory for HPC and HPDA Workflows

Michèle Weiland[1], *Adrian Jackson*[1], *Nick Johnson*[1], *Mark Parsons*[1]

Byte-addressable storage class memory (SCM) is an upcoming technology that will transform the memory and storage hierarchy of HPC systems by dramatically reducing the latency gap between DRAM and persistent storage. In this paper, we discuss general SCM characteristics, including the different hardware configurations and data access mechanisms SCM is likely to provide. We outline the performance challenges I/O requirements place on traditional scientific workflows and present how data access through SCM can have a beneficial impact on the performance of such workflows, in particular those with large scale data dependencies. We describe the system software components that are required to enabled workflow and data aware resource allocation scheduling in order to optimise both system throughput and time to solution for individual applications; these include a data scheduler and data movers. We also present an illustration of the performance improvement potential of the technology, based on initial workflow performance benchmarks with I/O dependencies.

Keywords: NVRAM, 3D XPoint, SCM, workflows, resource scheduling.

Introduction

Today's supercomputers offer a computing environment that focuses on compute performance first and foremost. The advent of byte-addressable non-volatile memory however means that in the coming years supercomputers will have sufficient memory capacity per compute node to no longer be exclusively used (and useful) for high-performance scientific computation (HPC), but also for high-performance data analytics (HPDA). Rather than simply ingesting data that was generated on a different system, input data for simulations will be prepared directly on the supercomputer where the simulation will be executed, and simulation output will be analysed and post-processed there as well. We predict that the mix of applications running on supercomputers will become broader: in addition to the largely compute intensive HPC applications, there will be memory and I/O intensive HPDA applications as full scientific workflows are enabled on a single system. This break in the status quo motivates the contributions of this paper: a discussion of the performance benefits of non-volatile memory, in particular with a view to optimising the end-to-end performance of workflows with complex compute and data dependencies; and a description of the system software infrastructure that is necessary to support them.

1. Storage Class Memory

Byte-addressable, non-volatile memory (hereafter referred to as Storage Class Memory, or SCM) represents the latest advance in memory technologies. SCM promises to deliver both greater performance and endurance than existing storage technologies, as well as increased density in comparison to DRAM. Compute platforms with SCM will have access to non-volatile memory that is capable of storing several TBs of data per node. This very large memory capacity for servers, and long term high-performance persistent storage within the memory space of the servers, means that new techniques for performing I/O will emerge. SCM enables Direct

[1]EPCC, The University of Edinburgh, Edinburgh, United Kingdom

access (DAX) from applications to individual bytes of data; this is fundamentally different from the block-oriented way I/O is currently implemented [9].

Several manufacturers are working on delivering byte-addressable SCM to the market within the next few years. The development that is furthest advanced to date is the result of a collaboration by Intel and Micron, the 3D XPoint^{TM} NVDIMM [3]. As the name implies, this SCM sits in the DIMM slots next to the CPU and alongside DRAM, with access to the NVDIMM space managed via the processor's memory controller. Unlike DRAM however, data that is stored on the NVDIMM is persistent, which means that it can be used as a (potentially long-term) storage environment as well as memory. 3D XPoint^{TM} NVDIMMs can be used in two different modes [6], which have implications for how applications can exploit these memory spaces. Changing between the two modes requires a system reboot.

1.1. Data Access

SCM has the potential to enable synchronous byte-level I/O, moving away from the asynchronous block-based file I/O applications currently rely on. In current asynchronous I/O, applications pass data to the operating system (O/S) which then uses driver software to issue an I/O command, adding the I/O request into a queue on a hardware controller. The hardware controller will process that command when ready, notifying the O/S that the I/O operation has finished through an interrupt to the device driver.

SCM can be accessed simply by using a load or store instruction, as with any other memory operation an application undertakes. This may require an additional instruction to ensure the data is persistent (fully committed to the non-volatile memory), if persistence is required by an application, or such persistence guarantees may be provided by the hardware through fault tolerant power supplies protecting volatile memory within the system (such as asynchronous DRAM refresh). With SCM providing significantly lower latencies than external storage devices, the traditional I/O block access model, using interrupts, becomes inefficient because of the overhead of context switches between user and kernel mode (which can take thousands of CPU cycles). Furthermore, with SCM it becomes possible to implement remote access to data stored in the memory using RDMA technology over a suitable interconnect. Using high performance networks can enable access to data stored in SCM in remote nodes faster than accessing local high performance SSDs via traditional I/O interfaces and stacks inside a node. Therefore, it is possible to use SCM to greatly improve I/O performance within a server, increase the memory capacity of a server, or provide a remote data store with high performance access for a group of servers to share. Such storage hardware can also be scaled up by adding more SCM memory in a server, or adding more nodes to the remote data store, allowing the I/O performance of a system to scale as required.

However, if SCM is provisioned in the servers in a supercomputer, there must be software support for managing data within the SCM. This includes moving data as required for the jobs running on the system, and providing the functionality to let applications run on any server and still utilise the SCM for fast I/O and storage (i.e. applications should be able to access SCM in remote nodes if the system is configured with SCM only in a subset of all nodes).

As SCM is persistent, it also has the potential to be used to implement techniques for resiliency, providing backup for data from active applications, or providing long term storage for databases or data stores required by a range of applications. With support from the system software, servers could be enabled to handle power loss without experiencing data loss, effi-

ciently and transparently recovering from power failure. Applications could resume from their latest running state and maintaining data, with little performance overhead, especially compared to current techniques of writing data to external storage devices such as high performance filesystems.

1.2. 1-Level Memory

The first of the two modes that SCM can operate in is 1-level memory, or 1LM, which views main memory (DRAM) and NVRAM as two separate memory spaces, both accessible by applications (see Fig. 1). This mode is conceptually similar to the Flat Mode configuration of the high bandwidth, on-package, MCDRAM in current Intel Xeon Phi™ processors (code name Knights Landing or KNL) [10]. DRAM is managed via standard memory APIs, such as *malloc*, and represents the only visible memory space for the operating system. The NVRAM on the other hand is managed by persistent memory and filesystem APIs, such as *pmem i/o* [8] and *mmap*, and presents the non-volatile part of the system memory. Both allow access via direct CPU load and store instructions. In order to take advantage of SCM in 1LM mode, either the system software, or the applications have to be adapted to be able to manually use these two distinct address spaces.

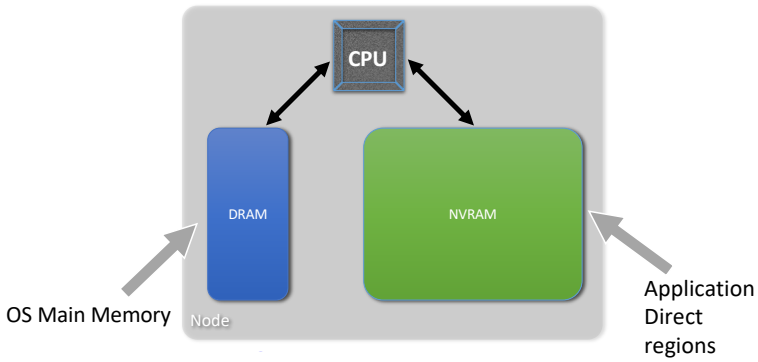


Figure 1. 1LM mode, where DRAM and NVRAM are two separate memory spaces

1.3. 2-Level Memory

2-level memory, or 2LM, configures DRAM as a cache in front of the NVRAM (see Fig. 2). Applications only see the memory space of the SCM; data that is being used is transparently stored in DRAM, and moved to SCM when no longer immediately required by the memory controller (as in standard CPU caches). This is very similar to the Cache Mode configuration of MCDRAM on KNL processors. This mode of operation does not require applications to be altered to exploit the capacity of SCM, and aims to give memory access performance at near to main memory speeds whilst providing the large memory space of SCM. Exactly how well the main memory cache performs depends on the specific memory requirements and access pattern of a given application. Furthermore, in this mode the persistence of the NVRAM contents cannot be guaranteed, due to the volatile nature of the DRAM cache that, at any given time, may hold updated versions of data stored in NVRAM. Therefore, the non-volatile characteristics of SCM are not exploited in this mode of operation. In 2LM mode, it is also possible to divide the SCM space into two partitions: memory (not persistent) and “app direct” (persistent).

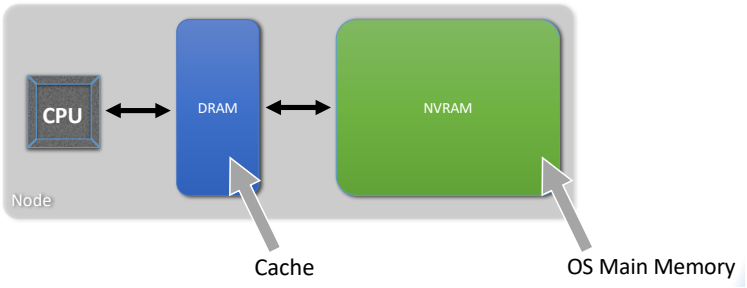


Figure 2. 2LM mode, where DRAM is a cache to the NVRAM memory space

```
#SBATCH --ntasks=64
#SBATCH --time=01:00:00

cd job1
mpirun -n 64 ./job1
cd ../job2
mpirun -n 32 ./job2
cd ../job3
mpirun -n 1 ./job3
```

Figure 3. Single SLURM script submission: there is no queuing time between jobs, but the maximum number of resources is used throughout

2. Workflows

Many scientific simulations are the result of a workflow, i.e. a series of applications that each perform a specific task within the simulation, rather than that of a single application [2]. A workflow may include steps such as data pre-processing and manipulation, computational simulation, output reduction and post-processing, or visualisation. The capacity and performance characteristics of SCM mean that steps of a workflow that would previously have been executed away from the main supercomputer (e.g. on a dedicated high-memory system) are more likely to be performed in situ. Moving the entire workflow onto a single system simplifies the simulation setup and removes the need to make simulation data available across multiple systems. Users currently set up such workflows in three different ways:

1. By putting all the jobs into a single script, requesting the maximum number of resources to be required by the steps of the workflow at any point in time (see Fig. 3).
2. By creating a chain of jobs, each requesting the correct amount of resources, for example by submitting the next job in the workflow through inserting a job submission command at the end of the preceding job (see Fig. 4).
3. By specifying dependency conditions using the job scheduler (see Fig. 5).

Only the final one of these three options implicitly supports the notion that there can be a dependency between jobs, however the dependency in this case is limited to being temporal.

Two different types of workflows can be envisaged: firstly, *monolithic* workflows are comprised of applications that each use the same amount of resources (i.e. the same number of compute nodes); and secondly, *composite* workflows that consist of applications with varying demands on the resources. The example in Fig. 3 is an instance of composite workflow.

```
#SBATCH --ntasks=64
#SBATCH --time=01:00:00

cd job1
mpirun -n 64 ./job1
cd ../job2
sbatch job2.sh
```

Figure 4. Basic job chaining in a SLURM script: the chained job will be put into the queue as if it was submitted manually

```
JOB1_ID=$(sbatch job1.sh)
echo $JOB1_ID
sbatch --dependency=afterok:$JOB1_ID job2.sh
```

Figure 5. Defining a dependency in SLURM: the scheduler uses the job ID to check that the preceding job has completed successfully, and the next job is then submitted into the queue

All of the three workflow setup approaches can have drawbacks: the first approach minimises the end-to-end runtime T_{all}, because there is a single queuing penalty T_{queue} at the start of the job which has to be added on to the time when the job is running T_{run}. However, unless all the workflow steps require the same number of compute nodes, i.e. unless the workflow is monolithic, this approach is wasteful in terms of resources, because the maximum number of compute nodes R_{max} is used for the entire duration of the job:

$$T_{all} = T_{queue} + \sum_{i=1}^{N} T_{(i,run)}, \tag{1}$$

$$R_{all} = R_{max} * T_{all}. \tag{2}$$

If there is a large discrepancy between the resources required by each workflow step (say if one of the steps is serial and another uses hundreds of nodes), not only does this approach quickly become very expensive, it also impacts the utilisation of the system, because although nodes are allocated to a job, they are not active all the time the job is executing but remain remain unavailable for other jobs.

The second and third approaches minimise resource utilisation in the composite workflow case, because for each step N, the correct amount of resources is requested. The time to completion for the workflow however will now have to include additional queuing time T_{queue} on top of compute time T_{run} for each of the N steps of the workflow. On a busy machine where queuing times are long, this approach can have a significant impact on the total time to solution:

$$R_{all} = \sum_{i=1}^{N} R_i * T_i, \tag{3}$$

$$T_{all} = \sum_{i=1}^{N} T_{(i,queue)} + T_{(i,run)}. \tag{4}$$

2.1. Workflows with Data Dependencies

On today's supercomputers, data dependencies with workflows are largely implemented by sharing data through files that are written to, and read from, a shared file system. Therefore, regardless of the approach that is taken for the execution of the workflow, T_{run} includes reading data from the file system at the start of a job, and writing results back at the end, in addition to performing the computation:

$$T_{run} = T_{I/O} + T_{compute}. \tag{5}$$

The aim is to minimise both the resource utilisation R_{all} and the time to solution T_{all} by providing true support for complex workflows within the job scheduler, and by reducing the I/O component $T_{I/O}$ of the runtime as much as possible through allowing workflows to share data without writing to a file system that is external to the compute nodes.

For supercomputers that execute a lot of data-intensive workflows, I/O performance presents a considerable performance bottleneck that the arrival of SCM will help alleviate. On a system with SCM, the data that is produced as part of a workflow can be kept on the compute nodes to be consumed in situ until the workflow concludes. In 1LM mode for instance, this could be achieved by simply using local files. In order to achieve this, the job scheduler and resource manager must understand and support the notions of both workflows and of data locality, so that individual steps of a workflow are placed on compute nodes that (ideally) already have a copy of the data.

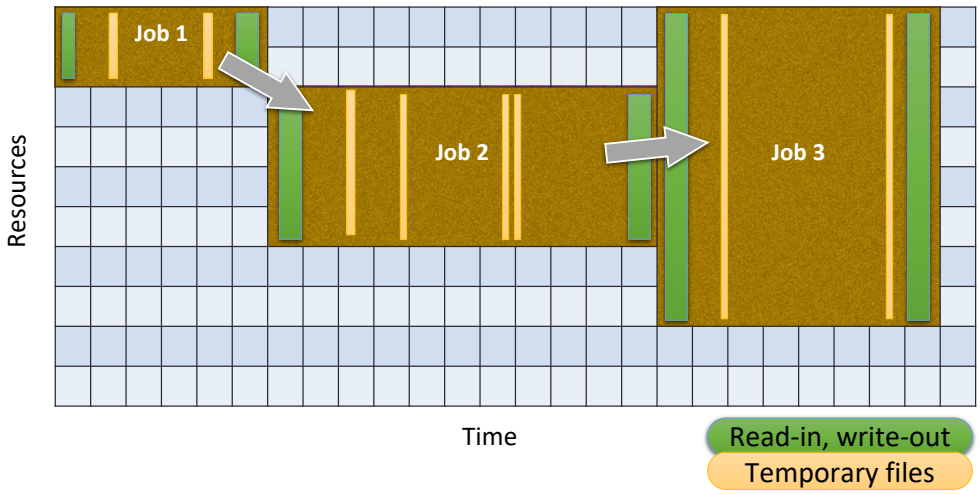


Figure 6. Example of a workflow where information is shared by writing out and reading in data. Data written by Job 1 is ingested by Job 2; and data written by Job 2 is in turn ingested by Job 3

Providing support for workflows without impacting the throughput of standard jobs is potentially complex if no restrictions are applied to the workflows. We therefore limit our support to workflows with the following properties:

1. At the time of submission of the workflow, the full workflow must be known, i.e. no extra steps can be added while the workflow is active.
2. A workflow must have data dependencies.

3. The temporal dependencies of the workflow steps must be fixed, i.e. each job must know which job(s) immediately precede(s) and follow(s) it, and this cannot be changed.
4. As part of the workflow description, the data that is shared between the different jobs is listed explicitly, and only this data forms part of the workflow.

2.2. Using SCM to Optimise Resource Usage and Time to Solution

SCM brings the opportunity for supercomputers to start offering more fundamental support for workflows with data dependencies. SCM can adopt the role of large (persistent) memory or of a fast storage device (or a combination of the two), which means that compute nodes with local SCM will be able to tackle a much wider workload than traditional HPC systems. The key benefit SCM offers is that it will allow applications to ingest and output data (in any form) with minimal involvement from the external file system. Prior to a job running, its associated input data can be pre-loaded onto the SCM of the compute nodes that will be allocated to the job. Similarly, a job can write its final output data locally to SCM; once the job has completed, and assuming the output data does not need to be read by another job, the data can be moved off the compute node and onto external storage. The benefit is that compute resources are used primarily for compute, and not for I/O, and time to solution and system throughput both improve. In theory, once data has been moved from the network attached storage to the compute node, it can remain there and be accessible until it is explicitly removed. In practice, there are a number of questions that arise, such as: who is responsible for moving data to and from a compute node; how long should data be kept on a compute node when it is not being used; what is the impact of moving data in the background on the performance of all jobs; and how does workflow support fit into a charging model for users. In order for workflow support to be transparent to the user (a key requirement for usability), the system software must address these questions.

3. Outline of Required System Software

The system software provides the functionality necessary to fulfil the requirements listed above. From the perspective of providing transparent support for workflows, with not only temporal but also data dependencies, a number of scenarios are supported:

- Applications can request to share data through SCM. This functionality is primarily for sharing data between different components of the same computational workflow, but it could also be used to share a common dataset between a group of users.
- A user can request for data to be loaded into SCM prior to a job starting, or for it to be moved off SCM after a job has completed. This is not dissimilar to current Burst Buffer technology and is not limited to workflows, but it supports the notion that data can be moved in the background, while nodes are performing computations.
- Data access is restricted to the owner of the job or workflow, or to users that are explicitly granted access. Encryption of data is enabled in order to make sure those access restrictions are maintained.
- A user can choose between 1LM and 2LM mode, if they are supported by the SCM hardware. Rebooting a node into a particular mode is achieved through the resource manager, i.e. the user can specify the job environment in the batch submission script.

- The resource manager can allocate nodes based on storage/memory capacity, as well as compute. If data can be temporarily stored on a compute node, the capacity for storage or memory that is available to other jobs will be reduced for the duration.

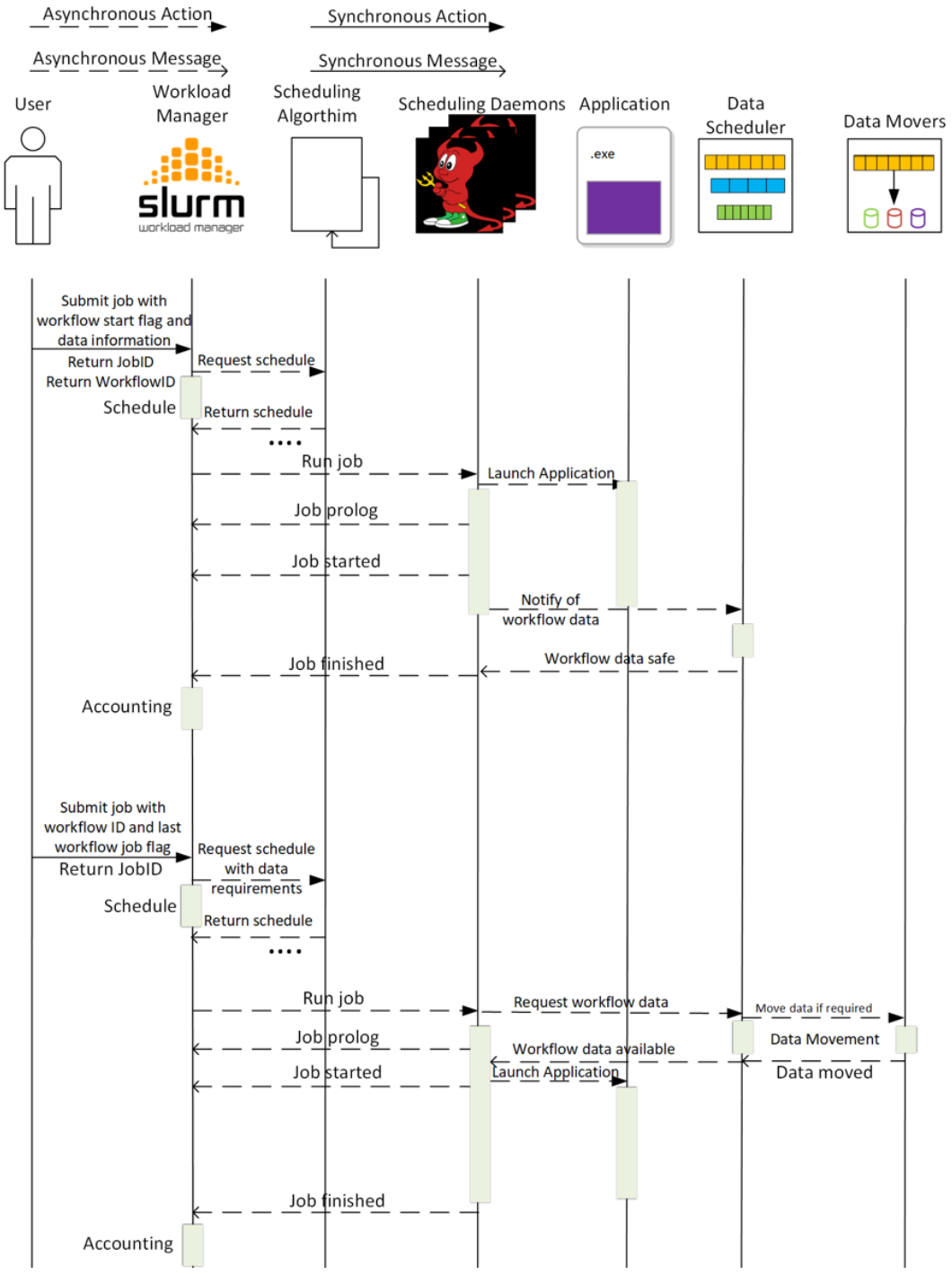


Figure 7. Sequence diagram describing how the workload manager, schedulers and data movers implement a workflow

Figure 7 shows the main components that enable a workflow with data dependencies on a system with SCM: the workload manager, the job and data schedulers, and the data movers.

3.1. Workload Manager and Job Scheduler

As shown earlier, workload managers (the example in Fig. 5 shows SLURM [5]) allow for users to define temporal dependencies between applications. However what is currently not commonly possible on supercomputers is for data dependencies to be defined in the same way. One reason for this is that HPC systems mostly do not have local storage, and it is therefore not possible to keep data close to where the computation takes place: if the data has to be read from or written to an external file system, there are no performance advantages to be gained from understanding the data dependencies.

The workload manager must also be able to query a node's mode of operation, and the amount of free storage (or memory, depending on how the SCM is used) that can be assigned to a job. This latter point is particularly important if data is left on a node for a period of time, e.g. because the next step in a workflow is not ready, and the node's compute capabilities can be used for other work.

The data dependency requirements of a job or workflow are described in the submission script in a similar way in which the compute resource needs would be outlined today. In addition to listing the walltime and number of processes or nodes, the I/O requirements (including the size of the data) are also described. With this additional information, and the workload manager's ability to query the system state, the job scheduler can (if there is sufficient capacity) assign the components of a workflow to be near the data that forms part of this workflow.

The job scheduler is aware of the SCM resources in the system at all times, and it allows users to specify SCM requirements and data movement requirements, to understand the configuration of the compute nodes and to allow users to specify configuration requirements of compute nodes for their jobs. The job scheduler communicates with all job scheduling and data scheduling components of the system software, to enable the system to run in as efficient a manner as possible and ensure user data is secure and safe. The job scheduler consists of multiple scheduling components, including components that can schedule jobs based on data location or energy policies.

3.2. Data Scheduler

The data scheduler operates under the instruction from other software components, be they system software (e.g. the workload manager or job scheduler), or directly through the request of an application. The main responsibility of the data scheduler is to orchestrate the migration of data. This can be between the different hardware levels within a single node, independently from an application, or between different nodes in the system, specifically to and from SCM on other nodes over the high performance interconnect.

The data scheduler also keeps track of data and what hardware that data is located in. On the compute nodes, the data scheduler component provides the local functionality to move data between storage levels (e.g. filesystem, SCM, DRAM) as instructed by the higher level component.

3.3. Data Movers

Data movers are simple software components that are used by the data scheduler to undertake specific data operations, such as copying data between different filesystems, from local to remote compute nodes, and between different data storage targets (for instance between an

object store and a filesystem). Separate data movers implement different operations, allowing targeted optimisations for each operation.

4. Assessing the Performance Optimisation Potential

To evaluate the potential for workflow optimisation that SCM, and an SCM aware job scheduler, can enable we undertook some benchmarking of the I/O costs that a workflow can experience. For these tests we create separate producer and consumer applications, that either generate or read a set of data files. The applications can generate or utilise different numbers or sizes of files to enable a range of different types of workflow interaction to be explored. These applications do not perform any other work, and thus only represent the I/O aspects of the workflow, but they do allow us understand the potential for performance optimisation from SCM functionality over a range of hardware and workflow configurations. We evaluate workflow I/O costs, writing data from the producer and reading data with the consumer, using a single compute node, with three different hardware configurations:

1. External Lustre filesystem;
2. Internal SSD storage device;
3. Memory mapped filesystem.

The external Lustre filesystem is equivalent to many current HPC system configurations. The internal SSD storage device represents compute node local storage, but without the potential read and write performance that true SCM hardware offers. The memory mapped filesystem represents performance that is closer to SCM technology. File-based I/O is generally subject to O/S level caching, keeping recent data in memory for re-use rather than requiring the data to be fetched from disk. Such caching can offer significant performance benefits for recently used file data, but is not representative of workflows where applications could be run on different nodes or at different times, and therefore would not be able to benefit from such I/O caching. Therefore, we ran our single producer-consumer benchmarks both with and without I/O caching to enable the evaluation of the benefit of such functionality and the impact of being unable to utilise it.

The data is collected on a single dual-socket node containing 2 Intel Xeon Platinum 8160F CPUs (24 cores @ 2.10 GHz each) with 192GB of DDR4 memory and a local 800GB Intel SSD DC S3710 Series SSD device. The node is connected to a 750GB Lustre (version 2.9) filesystem. Table 1 presents the average of 5 runs of each benchmark on the different hardware options we have previously outlined without I/O caching, using the following file configurations:

- 10 files, each of 1GB (10 x 1GB);
- 100 files, each of 100MB (100 x 100MB);
- 1,000 files, each of 10MB (1,000 x 10MB).

The results presented are using a single producer and a single consumer application on the node. It is evident from Tab. 1 that significant savings can be made to the workflow overheads associated with transferring data between workflow components. Simple writing to a local storage device (SSD) rather than the external filesystem reduces the I/O cost by up to five times. Moving from writing to a traditional storage device to writing data to memory brings even larger benefits, with the best performance around twelve times faster than writing to the local disk, and around fifty eight times faster than writing to the external filesystem. Whilst SCM is unlikely to achieve performance as good as memory mapped filesystem hosted on DRAM, these

Table 1. Performance of workflow benchmark *without* I/O caching (single producer-consumer), using three different file configurations. The performance is reported as time in seconds. Times in brackets are (write/read) times for the (producer/consumer)

Hardware	10 x 1GB	100 x 100MB	1000 x 10MB
Lustre	291.41 (137.77/153.64)	216.68 (105.34/111.34)	196.80 (102.42/94.38)
SSD	61.26 (34.39/26.87)	54.53 (29.05/25.48)	54.44 (28.09/26.35)
Memory	4.97 (3.02/1.95)	4.47 (2.99/1.48)	4.70 (3.16/1.54)

Table 2. Performance of workflow benchmark *with* I/O caching (single producer-consumer), using three different file configurations. The performance is reported as time in seconds. Times in brackets are the (write/read) times for the (producer/consumer)

Hardware	10 x 1GB	100 x 100MB	1000 x 10MB
Lustre	144.36 (138.94/5.38)	104.26 (102.09/2.18)	103.90 (102.24/1.64)
SSD	36.12 (34.37/2.04)	30.73 (29.11/1.62)	30.08 (28.28/1.80)
Memory	4.62 (3.00/1.62)	4.66 (3.04/1.62)	4.91 (3.25/1.66)

results indicate the performance differences between storage devices, such as fast SSDs, and true memory technologies.

Table 2 presents the average of 5 runs of the same benchmarks, but this time with I/O caching enabled, highlighting the performance optimisation potential enabled by a SCM and data aware job scheduler. It is evident from the results in the table that both the external filesystem and internal storage device benefit significantly from I/O caching enabled at the operating system level. The retention of data within the compute node can improve the performance even with fast I/O devices. We note that I/O caching does not significantly impact the memory mapped filesystem as that approach is already keeping data in memory rather than requiring I/O to access the underlying persistent storage device. As well as evaluating a single producer-consumer combination, we also evaluated the performance impact of workflow optimisation with SCM using multiple producer-consumers on a single node. We benchmarked using 36 producers and 36 consumers, running the same tests as before, albeit with smaller numbers of files to enable the benchmarks to finish in a reasonable time, and the data to be resident in memory.

Table 3 presents the results of the no-caching test, with the main number as the maximum runtime (maximum write time plus maximum read time) for the workflow, and the number in brackets as the minimum runtime (minimum write time plus minimum read time). It is evident from the table that the performance impact of multiple processes undertaking I/O at the same, or similar, times is significantly larger with the external filesystem than that with the local device or writing data to memory.

The performance differential between the Lustre and SSD benchmarks is larger for the multiple process tests ($\approx 5.7 - 6.4\times$) than for the single process tests ($\approx 3.6 - 4.6\times$). The Memory benchmark performance is in fact significantly faster than the single producer-consumer bench-

Table 3. Performance of workflow benchmark without I/O caching (36 producer-consumers). Performance is reported as maximum time in seconds (minimum time in brackets)

Hardware	1 x 1GB	10 x 100MB	100 x 10MB
Lustre	978.25 (790.15)	911.58 (591.07)	906.87 (829.00)
SSD	152.00 (142.99)	155.63 (138.14)	158.12 (150.78)
Memory	0.84 (0.77)	1.06 (0.79)	1.03 (0.66)

Table 4. Characteristics of the jobs used in the illustration of the optimisation potential

Job ID	Number of nodes	$T_{I/O}$ (s)	$T_{compute}$ (s)	T_{run} (s)	Part of Workflow?
1	4	10 + 30	50	90	y
2	1	10 + 0	180	190	n
3	15	10 + 0	30	40	n
4	1	20 + 10	400	430	n
5	7	10 + 10	70	90	n
6	15	10 + 10	20	40	n
7	4	30 + 120	180	330	y
8	8	0 + 60	250	310	n
9	4	10 + 30	10	50	y
10	2	20 + 30	150	200	n

marks, however we believe this is because I/O for the small data sizes used in these benchmarks can exploit cache memory more efficiently. Furthermore, the performance variability of node local I/O (SSD) is also significantly smaller ($\approx 5\% - 12\%$) than for the external filesystem ($\approx 9\% - 54\%$), demonstrating another potential benefit of SCM (reduced performance variability).

4.1. Illustration of Optimisation Potential

In this section, we give a simple example of how data aware workflow scheduling using SCM can improve both the performance of the workflow itself, and improve the resource usage on the system. We assume a system with 20 nodes and schedule 10 very short jobs onto these nodes. The characteristics of the jobs are described in Tab. 4; for illustration purposes, each job is broken down into three distinct phases (input, compute and output), and may or may not be part of a workflow. Jobs that are not part of a workflow are scheduled onto the system by incrementing ID in a round-robin fashion, if there is sufficient space to accommodate them. Jobs that are part of a workflow required the preceding components of the workflow to be completed before they can be executed; in our example, Jobs 1, 7 and 9 form a workflow with data dependencies. It is assumed that no data is stored locally by default.

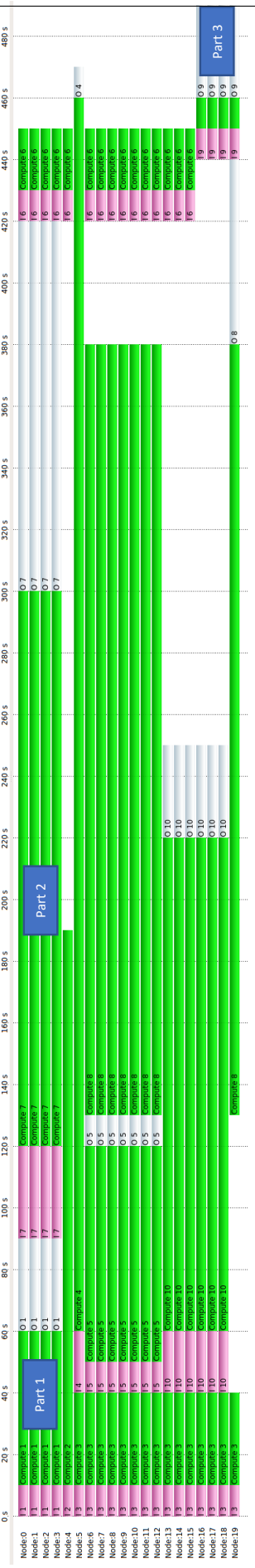


Figure 8. Timeline of example jobs being scheduled, without awareness of data dependencies within a workflow. Time to solution for all three parts (Jobs 1, 7 and 9) of the workflow: 490s

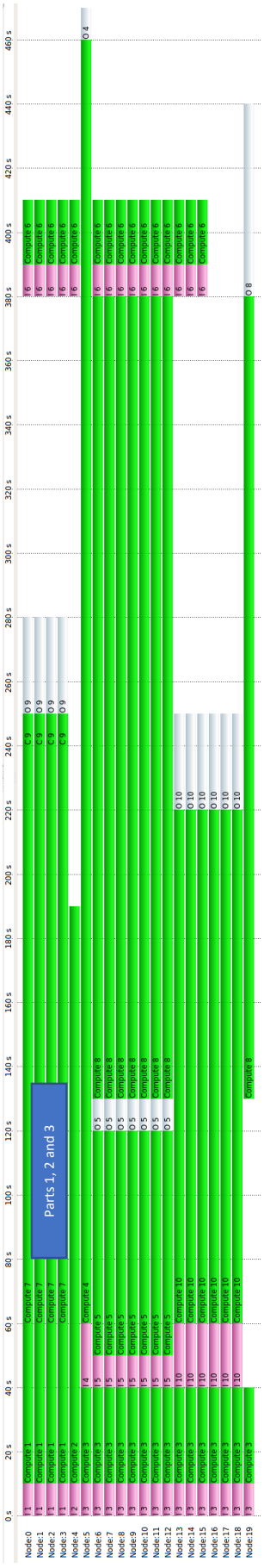


Figure 9. Timeline of example jobs being scheduled, with awareness of data dependencies within a workflow and avoiding writing to/reading from the external file system. Time to solution for all three parts (Jobs 1, 7 and 9) of the workflow: 280s

Figure 8 shows a timeline of the allocation of the example jobs using a standard scheduler that is not aware of data dependencies inside a workflow. As can be seen from the timeline, parts 1 and 2 of the workflow happen to be scheduled onto the same nodes by chance (i.e. node 0–3). However, the 3rd component of the workflow is scheduled on nodes 16–19, because another job has taken over the resources used by the first two parts of the workflow. Despite parts 1 and 2 of the workflow being scheduled on the same resources, Job 1 has to write its output to the external file system. This data represents the input for Job 7, which in turn has to reload the data from the external file system. The total individual runtimes for the workflow components T_{run} are 90s, 330s and 50s, a total of 470s. However, because Job 9 is held in the queue for a short amount of time (in this example 20s), the total time to solution T_{all} increases to 490s.

Figure 9 illustrates how the allocation changes if the scheduler understands data dependencies and the importance of data locality in workflows. Two separate optimisations occur: firstly, the 3 parts of the workflow are now scheduled to use the same nodes 0-3; secondly, and as a direct result of the first step, data can now be kept locally on the compute nodes. There is no need to access the external file system. I/O cost will be close to DRAM speed and thus vastly reduced even when compared to SSD. In our example here, we assume (for simplicity) that I/O cost tends to 0 when using SCM. The first job in the workflow still needs to read its input from external storage, and the final job needs to write back to external storage, but all other I/O can be local. This results in the following runtimes per job:

$$\begin{aligned} T_{run,Job1} &= 10s + 50s = 60s, \\ T_{run,Job7} &= 0s + 180s = 180s, \\ T_{run,Job9} &= 30s + 10s = 40s. \end{aligned} \tag{6}$$

As the jobs are scheduled consecutively onto the same nodes, there is no additional queuing time, and T_{all} is simply the sum of the individual runtimes, i.e. 280s.

As a further optimisation, which we do not consider in this example, it is possible to pre-load and post-move the input/output data of particularly data-intensive jobs in the background, prior to them starting execution, or after they have completed. This increases the resource allocation constraints that the scheduler has to work around, however, the potential gains in time to solution and resource utilisation are significant.

5. Related Work

Recent years have seen a lot of research emerge around the topics of I/O performance (notably with the arrival of new storage technologies), scheduling of large-scale systems and scientific workflows. Daley et al. [1] assess how Burst Buffers can alleviate the I/O bottleneck of some scientific workflows, and acknowledge the associated data management challenges. Also primarily focussed on Burst Buffers, Herbein et al. [4] discuss a technique for making scheduling policies I/O aware, taking into account the different bandwidths of the storage hierarchy in order to avoid I/O contention. Rodrigo et al. [7] address the idea of workflow-aware scheduling, having recognised that workloads on HPC system are a more commonly comprised of an interdependent series of jobs. They propose a workflow-aware extension to the the widely used SLURM scheduler, which goes beyond the simple temporal dependencies between jobs that are supported in most resource managers.

Conclusions

In this paper, we outline the opportunities for performance improvements that byte-addressable storage class memory, such as the upcoming 3D XPoint^{TM} NVDIMMs can bring in particular to data intensive applications. We also present the system software that needs to be put in place in order to support data aware workflow scheduling using persistent memory. Overall, the benefits are clear: if a workflow can be scheduled so that its time to solution is decreased, but without impinging on other jobs, the system resources are freed up sooner and the total workload throughput for the HPC system is improved.

Acknowledgements

The NEXTGenIO project has received funding from the European Union's H2020 Research and Innovation Programme under Grant Agreement no 671951. The project works on addressing the I/O challenge, a key bottleneck as the HPC community is moving towards the Exascale. NEXTGenIO is an internal collaboration between research organisations and industry to develop a prototype hardware platform that uses on-node SCM to bridge the latency gap between memory and storage. The project is also developing the necessary system software to enable the transparent use of SCM, with the aim to improve application performance, as well as system utilisation and workload throughput.

References

1. Daley, C., Ghoshal, D., Lockwood, G., Dosanjh, S., Ramakrishnan, L., Wright, N.: Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers. Future Generation Computer Systems (2017), DOI: 10.1016/j.future.2017.12.022

2. Deelman, E., Peterka, T., Altintas, I., Carothers, C.D., van Dam, K.K., Moreland, K., Parashar, M., Ramakrishnan, L., Taufer, M., Vetter, J.: The Future of Scientific Workflows. The International Journal of High Performance Computing Applications 32(1), 159–175 (2018), DOI: 10.1177/1094342017704893

3. Hady, F.T., Foong, A., Veal, B., Williams, D.: Platform Storage Performance with 3D XPoint Technology. Proceedings of the IEEE 105(9), 1822–1833 (Sept 2017), DOI: 10.1109/JPROC.2017.2731776

4. Herbein, S., Ahn, D.H., Lipari, D., Scogland, T.R., Stearman, M., Grondona, M., Garlick, J., Springmeyer, B., Taufer, M.: Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. pp. 69–80. HPDC '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2907294.2907316

5. Jette, M.A., Yoo, A.B., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In: In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003. pp. 44–60. Springer-Verlag (2002)

6. Joydeep, R., Varghese, G., Inder M., S., Jeffrey R., W.: Intel Patent on Multi-Level Memory Configuration for Non-Volatile Memory Technology. https://www.google.com/patents/US20150178204 (2013), accessed: 2018-04-11

7. Rodrigo, G.P., Elmroth, E., Östberg, P.O., Ramakrishnan, L.: Enabling Workflow-Aware Scheduling on HPC Systems. In: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing. pp. 3–14. HPDC '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3078597.3078604

8. Rudoff, A.: Persistent Memory Programming. http://pmem.io (2017), accessed: 2018-04-11

9. Rudoff, A.: Persistent Memory: The Value to HPC and the Challenges. In: Proceedings of the Workshop on Memory Centric Programming for HPC. pp. 7–10. MCHPC'17, ACM, New York, NY, USA (2017), DOI: 10.1145/3145617.3158213

10. Sunny, G.: Getting Ready for Intel© Xeon Phi™ Processor Product Family. https://software.intel.com/en-us/articles/getting-ready-for-KNL (2017), accessed: 2018-04-11

DOI: 10.14529/jsfi180106

A General Guide to Applying Machine Learning to Computer Architecture

***Daniel Nemirovsky*[1], *Tugberk Arkose*[1], *Nikola Markovic*[2], *Mario Nemirovsky*[1,3], *Osman Unsal*[1,4], *Adrian Cristal*[1,4], *Mateo Valero*[1,4]**

The resurgence of machine learning since the late 1990s has been enabled by significant advances in computing performance and the growth of big data. The ability of these algorithms to detect complex patterns in data which are extremely difficult to achieve manually, helps to produce effective predictive models. Whilst computer architects have been accelerating the performance of machine learning algorithms with GPUs and custom hardware, there have been few implementations leveraging these algorithms to improve the computer system performance. The work that has been conducted, however, has produced considerably promising results.

The purpose of this paper is to serve as a foundational base and guide to future computer architecture research seeking to make use of machine learning models for improving system efficiency. We describe a method that highlights when, why, and how to utilize machine learning models for improving system performance and provide a relevant example showcasing the effectiveness of applying machine learning in computer architecture. We describe a process of data generation every execution quantum and parameter engineering. This is followed by a survey of a set of popular machine learning models. We discuss their strengths and weaknesses and provide an evaluation of implementations for the purpose of creating a workload performance predictor for different core types in an x86 processor. The predictions can then be exploited by a scheduler for heterogeneous processors to improve the system throughput. The algorithms of focus are stochastic gradient descent based linear regression, decision trees, random forests, artificial neural networks, and k-nearest neighbors.

Keywords: machine learning, computer architecture, data science, parameter engineering, performance prediction, scheduling.

Introduction

Thanks to the increasing amounts of processing power and data generation over the last decade, there have been impressive machine learning applications in computer vision and natural language processing [11], gaming [16], and content recommendation systems [13] to name a few. The growth of data, use cases, and increasing popularity have triggered a rise of frameworks, which allow easier implementations of machine learning models which can run on commodity GPUs without developers having to build the models from scratch. A couple of popular frameworks include TensorFlow [1] and Caffe [8].

The rising popularity of machine learning and desire to perform larger and faster computations has encouraged the development of hardware accelerators [15] that can compete with GPUs while consuming much less energy especially for deep convolution networks (CNNs) [20]. Computer architects have focused so rigorously on specialized hardware for machine learning that as of yet, there has been limited research making use of machine learning algorithms to improve computer performance.

[1]Barcelona Supercomputing Center, Barcelona, Spain
[2]Microsoft, Belgrade, Serbia
[3]ICREA, Barcelona, Spain
[4]Polytechnic University of Catalonia, Barcelona, Spain

However, the few works that have done so in the areas of CPU scheduling [18, 19], cache replacement [10, 25], and branch prediction [9] have shown tremendous promise. These are but a few of the opportunities we foresee where machine learning could provide a significant advantage towards improving the efficiency of computer systems.

The goal of this work is to incentivize and provide a general guide to computer architects for applying machine learning to improve system performance. We describe a method that highlights when, why, and how to utilize machine learning models for improving system performance and provide a case study showcasing the effectiveness of applying machine learning to predict workload performance on an x86 core at the execution quantum granularity. The predictors can take input data gathered from different core types therefore acting as a cross core type predictor. The predictions can then be exploited by a scheduler for heterogeneous CMPs to improve system throughput. The machine learning algorithms within the scope of this work include stochastic gradient descent based linear regression, decision trees, random forests, artificial neural networks, and k-nearest neighbors.

The outline consists of firstly defining a problem (Section 1) which includes the overarching goals, constraints, and important attributes. This is followed by an exploration into how to understanding the data that can be generated, and whether a non linear prediction model is needed (Section 2). If so, then machine learning algorithms can be identified, trained, fine tuned, evaluated and integrated into a overarching solution (Section 3).[5] Prior to the conclusion, Section 4 explores related work and useful references for applying machine learning to computer architecture.

1. Clarifying a Computer Architecture Problem for Machine Learning

Conducting an exploratory analysis of a target system, workloads, and improvement goals is the first step in clarifying if and how machine learning can be utilized within the scope of the problem. As computer architects, we seek to improve the efficiency and performance of computer systems, therefore it is important to identify the components and metrics that characterize the system and improvement goals such as instructions per cycle (IPC), latencies (cycles or seconds), or energy consumed (Joules). Different target systems will generally have different constraints. For example, the specific metrics that define the improvement goals of a distributed datacenter (e.g., response time as a metric) may differ from those for improving a system on a chip (e.g., millions of instructions per second or MIPS) or graphical processing unit (e.g., floating point operations per second or FLOPS). Moreover, even when the metrics are the same (e.g., power requirements in Watts), the solutions may target components at different scales (e.g., circuit level - RTL design, microarchitecture - instruction window width, SoC level - cache/memory organization, and cluster level - interconnection layout and distribution of tasks). Identifying the target workloads (e.g., computational, memory, and/or I/O intensive) that will be executed is also useful for determining the expected behaviors of the components and whether system modifications are likely to translate into significant performance benefits.

Before deciding to apply machine learning, it is often useful to ask what additional knowledge would help improve the main components of interest in a computer system. In other words, which

[5]The full code complementing this work can be found at: https://github.com/dnemirov/ml_computer_architecture.

metrics that characterize the runtime behavior of a system and its components are valuable to know a priori. For example, if we are looking to improve the efficiency of a cache, it may be useful to know the access patterns and adapt the cache accordingly.

Predictors can provide additional knowledge about runtime behavior, but they should be complemented by mechanisms that transform the extra knowledge into system improvements. Viable predictor implementations should be assessed based on their accuracy, overheads, and feasibility of the mechanism that will exploit the prediction. It is also beneficial to analyze how different prediction accuracies can affect system improvements and overheads.

Though conventional branch predictors are typically not based on machine learning algorithms, the example is illuminative. It highlights how a prediction method relies not only on a predictor, but also a mechanism to exploit the predicted value and to handle inaccuracies. Branch prediction uses a predictor to estimate the outcomes of conditional branches. It takes an input (e.g., a branch instruction) and based on its prediction algorithm (e.g., 2-bit saturating counter [27]), produces an output (e.g., taken or not taken). Due to the latency constraints of how long it takes to make a prediction, branch predictors are generally implemented in hardware. The prediction is exploited by a mechanism in the microarchitecture which allows the processor to continue to execute instructions and roll back the execution state in case of a misprediction (at the cost of precious execution cycles). Depending on the constraints, the predictors and the mechanisms that exploit the predictions can be implemented in software and/or hardware.

Separately, CPU scheduling on a heterogeneous chip multiprocessor (CMP) can benefit from knowing a priori knowledge about how each software thread will perform on the different hardware cores. The metric in this case is not based on a binary classification as in the branch predictor, but instead can use the number of instructions per cycle (IPC) as a metric to gauge performance and system throughput. In this work, we will focus on understanding when and how to utilize machine learning algorithms to improve system performance. Specifically, the next sections present a case study of utilizing machine learning algorithms to improve system performance by focusing on predicting workload performance in IPC based on data collected every execution quantum (1ms).

2. Understanding Data

After specifying a problem by identifying the target system, workloads, and performance metrics, it is important to identify what available data is available and how it can be collected. The data that will be generated is dependent upon the simulation framework that will be used to conduct the execution experiments.

2.1. Simulation Framework

For this work, we utilize the Sniper [3] simulation platform. Sniper is a popular hardware-validated parallel x86-64 multicore simulator capable of executing multithreaded applications as well as running multiple programs concurrently. The simulator can be configured to run both homogeneous and heterogeneous multicore architectures and uses the interval core model to obtain performance results.

We have set up the Sniper simulation framework to simulate a commodity x86 Nehalem processor (specifics detailed in Tab. 1). To model how the system performs under a variety of

computational intensive target workloads, the simulation executes applications from two different benchmark suites, SPEC2006 [7] and SPLASH-2 [26].

The SPEC2006 benchmark suite is an industry-standardized, CPU-intensive benchmark suite, stressing a system's processor and memory subsystem. The SPLASH-2 benchmark suite is composed of a mix of different multithreaded kernels and applications focusing on high performance computing, graphics, and signal processing. The entirety of the benchmark suites (26 SPEC2006 and 13 SPLASH-2 workloads) are used with the exception of those which did not compile in our platform (*dealII, sphinx3, volrend*, and *wrf*).

Table 1. Simulated CPU configuration

Architecture	x86 Nehalem based
Frequency	2.66 GHz
Out of Order	4-wide issue width, 12-stage out-of-order, 128-entry ROB, and 48-entry LD/ST queue
L1 caches	Separate instruction and data 32KB write-through, 4-cycle latency, 8-way set associative, LRU replacement
L2 cache	Unified 256KB write-back, 8-cycle latency, 8-way set associative, LRU replacement
L3 cache	4MB, write-back, 30 cycle latency, 16-way set associative
Memory	Modeling all queues and delays, 120 cycle latency, controller bandwidth 7.6 GB/s

2.2. Data Generation

System simulators provide increased design flexibility compared to physical devices while offering detailed insights into runtime behaviors. An added benefit of using Sniper is the ability to output the statistics of different runtime behaviors (hereinafter referred to as *attributes*). Some of the statistics of interest include the number of micro operations (uops), branch prediction results, and cache and TLB accesses and misses. We have configured Sniper to periodically output a set of statistics that is generalized into nine ratio based attributes plus one IPC target value shown in Tab. 2. Generalizing the input attributes to the predictors enables predicting a workload's performance on a certain core type while possibly using input from executions on a different core type but with the same ISA. For example, to predict how a thread currently executing on a large core will perform on a small core, the attribute values collected during the previous execution quantum on the large core are generalized into ratios and provided as input to the predictor for small core which outputs an estimated IPC on the small core. Ratio based attributes are also useful for conducting a system analysis based on the predictions for synthetic workloads that have different ratio values for the attributes.

Each workload from both benchmark suites is executed on the simulated x86 processor, and the attributes are collected every execution quantum until the workload finishes. The amount of time needed to finish executing the workloads varies and as a result, the total data collected averaged to about 550 samples per workload and a total of 21,441 samples for all 39 workloads.

2.3. Data Division

Before conducting any further data analysis, it is important to separate the data into a *train* set which we can poke into and analyze and a separate *test* set that will be used to evaluate the final models. Exploring the complete data without separating it into a train and test set biases any analysis due to a priori knowledge about the test set. A common technique is to set aside between 70%-80% of the data for the train set and 20%-30% for the test set. However, as described above, the two benchmark suites not only contain a different amount of individual benchmarks (26 SPEC2006 vs 13 SPLASH-2), but the completion times vary between the benchmarks as well. This results in different quantities of data samples available for each benchmark. As an additional measure to guard against biasing the training for the test set, instead of combining all the samples from all workloads into on large data set and then separating it into train and test sets, we separate the workloads themselves into different data sets. Therefore, the train set consists of roughly 70% or 19 benchmarks from SPEC2006 and 70% or 10 benchmarks from SPLASH-2. That leaves 7 SPEC2006 and 3 SPLASH-2 workloads for the test set. There are numerous possible combinations of which benchmarks to select for the train and test sets. To account for this, we train and evaluate the machine learning model 1000 different times each using a different combinations of benchmarks for the train and test sets chosen at random. The evaluation results in Section 3.3 are based on the averages and standard deviation of the 1,000 different train and test error results.

Another method for accounting for benchmark idiosyncrasies could be using an equal number of samples from each of the workloads in the train set during the learning phase. However, this would affect how representative the train set will be of the amount of time the system is executing the target workloads. It is important to note that any transformation on the train set such as normalization is also performed on the test set.

2.4. Data Exploration

Exploring the data requires a mix of domain knowledge and utilizing several techniques with which to understand the distribution attributes and their relation to one another. Based on computer architecture domain knowledge, we can deduce that certain of the attributes may be highly correlated (e.g., IPC and L3 miss rate). Cleaning the data sets ensures that the amount of memory and computational overheads needed to work with the data sets is condensed. Removing noisy and/or redundant attributes can also be useful for reducing errors in our predictors later on.

A useful approach is to plot the Pearson correlations between the attributes in the train data set as is shown in Fig. 1. The darker red represents a higher positive correlation and the deeper blue represents a strong negative correlation. Confirming our intuition, there are several attributes that are highly correlated with one another including percentage of branch operations and the branch miss rate, as well as the miss rates of the caches. If any pair of attributes is highly correlated (say a threshold of > 0.9), it may be beneficial for the model efficiency to either remove one of the pairs or combine both attributes onto a single new attribute. A comparison can then be made using a model trained with all of the attributes compared to one trained with a reduced attribute set. In this work, the attributes do not seem to exhibit extremely high correlation (> 0.9) so we keep all the attributes for training.

Training and evaluating a simple linear regression model using IPC and each of the attributes independently can provide baseline error measurements and indicate whether it may be useful to apply non linear machine learning algorithms.

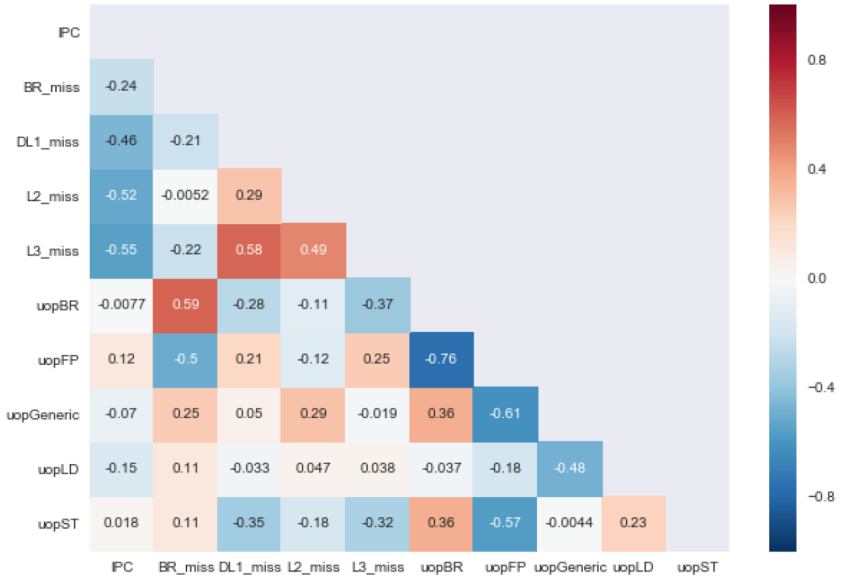


Figure 1. Pearson correlation heatmap of the attributes. The values of the attributes are based on ratio percentages (e.g., uopFP is the percentage of micro operations that are floating point)

Figure 2 shows the linear regression predictors based on the L3 miss rate attribute which has the closest correlation with the IPC. The plot visualizes how the prediction line is not able to capture the non-linear relationship of the training data and target value for even the most correlated attribute. This observation is highlighted in Tab. 2 which presents the root mean square error (RMSE) on the training set for each of the separate linear models trained using an individual attribute as the input x and the IPC as the target y. The resulting errors are considerable given that they are around 0.7 and that the average IPC range is between 0 and 4. This reveals that there is an opportunity for improvement using machine learning predictors.

Visualizing the distributions of the attributes in the train data set can provide additional insights into range of values for the attributes. The histograms of the IPC, L3 cache miss rate, branch misprediction rate, and percentage FP uops are plotted in Fig. 3. We can observe that (i) there are disproportionately more occurrences within a particular range of values for each of these attributes, (ii) the values of the attributes are in significantly different scales, and (iii) a majority of the distributions have long tails. Such varied scales and distributions make it harder for most machine learning algorithms to learn effectively. Therefore, we can utilize standardization techniques to transform the scale and distribution of these attributes and make detecting patterns and relationships easier for the machine learning algorithms. A method to do this is to subtract the mean value then divide by the variance, therefore transforming the values of the features to have a zero mean and unit variance ($\frac{X-\mu}{\sigma}$). This does not bound values to a specific range and is able to deal with outliers in a manner which other methods such as min-max scaling cannot.

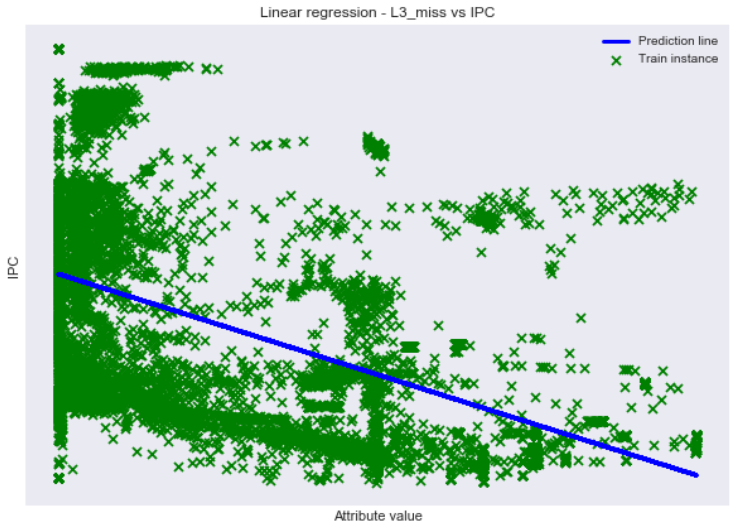


Figure 2. Scatter plot of L3 miss rate vs IPC from the training data set. Also plotted is the single-attribute (L3 miss rate) based linear regression prediction line in blue

Table 2. Runtime attributes expressed as ratio values collected each 1ms execution quantum. Also shows each attribute's correlation with target IPC and the RMSE of the predictions against the training data set using simple linear regression

Attribute	**Correlation with IPC**	**Linear reg RMSE**
% uopLD	-0.1538	0.7414
% uopST	0.0176	0.7502
% uopBR	-0.0077	0.7503
% uopFP	0.1157	0.7453
% uopGeneric	-0.0700	0.7485
BR miss %	-0.2399	0.7284
DL1 miss %	-0.4599	0.6663
L2 miss %	-0.5172	0.6422
L3 miss %	-0.5527	0.6253

3. A Case Study in Applying Machine Learning to Solve Computer Architecture Problems

Given that simple linear models leave much to be desired in terms of prediction error, it is a reasonable next step to see if machine learning based predictors can do better and by how much. This section demonstrates how to utilize machine learning algorithms with the data that has been generated, cleaned, and normalized to create predictors capable of estimating the

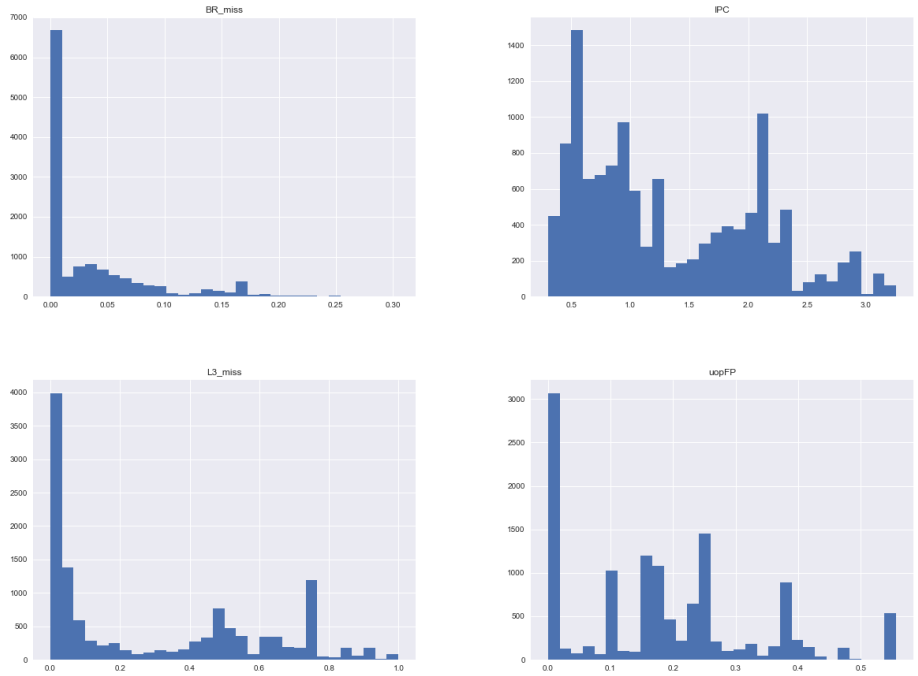


Figure 3. Histograms of the attributes branch misprediction rate, IPC, L3 miss rate, and percentage of FP uops. The y axis values are based on the quantity of instances from a total of over 13,000 samples which fall into the attribute range in the x axis

performance (measured in IPC) of a workload on an x86 core during an execution quantum. The goal of the predictor is to achieve low error and be able to predict using input data collected from executions on different core types. This enables cross core workload performance prediction which can be useful for a scheduler to improve system throughput. We analyze a set of popular machine learning algorithms, fine tune their learning and architectures, and lastly evaluate the final predictor errors.

3.1. Machine Learning Algorithms

In contrast to unsupervised learning which is useful for finding patterns in unstructured data, supervised training allows a machine learning model to learn to predict classes or values based on minimizing a loss function that quantifies the error between the predicted values and the target values. Since the data we have collected is labelled (i.e., the target IPC values are available in the data sets), we will focus on supervised machine learning methods. Moreover, since IPC is a continuous and not a categorical value, the machine learning models of interest are regressors, meaning they predict a continuous numerical value and not a class. Other areas in computer architecture may require the prediction of classes such as the case of branch prediction (i.e., a binary classification prediction of either branch taken or not taken).

The machine learning algorithms within the scope of this work are linear regression using stochastic gradient descent (SGD), decision trees, random forests, artificial neural networks (ANNs), and k-nearest neighbors (kNN). The computational cost and prediction ability of the

machine learning algorithms is regulated through *hyperparameters* which define the architecture of the specific algorithms. An overview of each of these algorithms and their hyperparameters is described below. All models are implemented using the Scikit-Learn Python framework [21] which offers a powerful toolbox of machine learning algorithms as well as preprocessing (e.g., normalization methods) and fine tuning methods (e.g., cross-validation and grid search methods). Implementations for production purposes requiring strict timing constraints could instead implement the algorithms directly in a lower level language (e.g., C) and rely on hardware acceleration to reduce training and prediction latency.

3.1.1. Linear Regression Using Stochastic Gradient dDescent (SGD)

Stochastic gradient descent (SGD) is useful machine learning alternative for finding a linear model without having to utilize the normal equation which does not scale well with large data sets since it requires inverting an input matrix which carries a complexity of $O(n^{2.3})$ to $O(n^3)$. This algorithm finds a linear function (e.g., $f(x) = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$) that uses SGD for training to learn the set of weights $w_{1...n}$ that should be multiplied to every input parameter $x_{1...n}$ and bias term w_0. It is straightforward to implement and generally provides low variance but high error (i.e., bias), especially when used to approximate non linear functions. This model will use SGD to approximate a linear equation using all nine of the attributes plus an intercept term.

During training, the model predicts an output for every sample from the train dataset and compares it to the target value using a *loss function*. The result of the loss function is what the algorithm will try to minimize at every step of the training. Though we calculate the loss for every sample, the weights can be updated after every sample chosen randomly from the training data set (SGD), after calculating the sum of the losses for a subset of the total train data set (mini batch), or after calculating the sum of the losses for the entire train data set (full batch). Here we utilize SGD to update the weights.

For regression, we use the mean squared error loss function $MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$, where m is the number of samples in the training batch, y_i is the target IPC value, and the predicted IPC value is $\hat{y}_i = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$. To update the weights, the partial derivative of the loss function with respect to the weight is multiplied with a learning rate hyperparameter α and then added to the old weight value. This is represented by the formula $w_i^{(new)} = w_i + \alpha * \frac{\partial MSE}{\partial w_i}$. The learning rate may be either static (e.g., $\alpha = 0.01$) or dynamically adjustable as is the case with momentum optimization [22]. The learning terminates when the algorithm converges to a minimum loss. This is always the case when using the MSE loss function for linear regression since it is a convex function. To prevent overfitting, especially for high dimensional training data sets, it is useful to add L1 ($\beta \sum w_j^2$) and/or L2 ($\beta \sum \mid w_j \mid$) regularization terms to the loss function to constrain the weights. Once trained, this linear model can be then used to make predictions by simply computing a weighed sum of the input parameters and a bias term. The principal hyperparameters of this model are the polynomial degree of the inputs, L1 and L2 regularization terms, loss function, and learning rate.

3.1.2. Decision Trees

Decision trees are able to predict a target value by inferring rules from the data features and creating a binary tree to express the model. A benefit of decision trees is that they do not require

the data to be normalized before training or predicting, thus reducing the amount of preparation time. The algorithm builds a binary tree node by node by focusing on a single attribute k and a threshold value for that attribute t_k at a time. The algorithm relies on splitting the training data set by using a loss function J which minimizes the MSE, $J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$. A node's MSE value is calculated by using the predicted value, $\hat{y}$ is based on averaging the target y value for all m instances belonging to that node.

Decision trees are simple to build and interpret. They also make it possible to rank the feature importances based on how close they are to the root of the tree (i.e., node depth). However, they are sensitive to rotations and small variations in the training data set which are aligned along non-orthogonal decision boundaries. Decision trees are non-parametric and also tend to overfit the training data set if left unconstrained during the construction of the binary tree. Reducing the degrees of freedom helps to reduce overfitting at the cost of increased error. A few interesting hyperparameters to help regularize the decision tree is setting its maximum depth and number of leaf nodes as well as the minimum number of instances a leaf or node must contain to split.

3.1.3. Random Forests

Random forests are an ensemble of shallow decision trees (i.e., estimators), each of which is trained on different random subset of the training data set and attributes. The technique for random sampling of the training data using replacement is known as *bagging* [2]. For this work, the random subsets are chosen using the *random patches* [14] technique which applies the bagging method to both training data and attributes. The output prediction of the random forest is the average of the predictions from all of the estimators. The increased diversity of the subsets and estimators results in larger individual bias (i.e., error) of each estimator but less variance overall than a single decision tree. This approach generally yields a better model than using a single decision tree except when features are highly correlated. It also tends to overfit if not adequately constrained. Random forests enable ranking feature importances by computing the average tree depth of a feature in all estimators. Their hyperparameters include many of those of the decision tree as well as the number of small decision trees estimators to use.

3.1.4. Artificial Neural Networks (ANNs)

ANN is a popular learning algorithm that is used to learn a non-linear function $f(x) = y$ through the use of training on an input set x and a target y. The relationships learned by the ANNs are often hard to identify and program manually, yet they can be lightweight and flexible to implement. They are capable of approximating complex non-linear functions and computing predictions quickly, but deep ANNs are also prone to overfitting the training data set.

An ANN consists of a set of input attributes (also known as input parameters) $x_1, x_2, ..., x_d$ of d dimensions. In the fully-feedforward ANN that is implemented, all of these inputs are connected to every unit in the first hidden layer of the ANN, the outputs of each layer then connect to all the units of the next layer and so on in unidirectional fashion. Each input x_i is assigned a numerical weight $w_{i,j}$ for its connection to unit j. The sum of all incoming connections to a unit multiplied by their corresponding weight is then performed ($z_j = \sum_{i=1}^{d} x_i * w_{i,j}$) before being passed into the unit's non-linear activate function $h(z_j)$. The activate function used in this work is the rectified linear (ReLU) function expressed as $h(z) = max(0, z)$. ReLU is fast to

compute and does not have a maximum saturation such as sigmoid or the hyperbolic tangent function which helps in reducing vanishing gradients during backpropagation (discussed below). The output of the activate function from the units from the l^{th} layer is fed to the input of the units of the $(l+1)^{th}$ layer. The final prediction is based on the outputs from all the units of the last hidden layer without passing through the activation function. The weights of the ANN are randomly initialized using Glorot initialization [5] from a uniform distribution between $+/-\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$.

To train an ANN, the backpropagation algorithm is utilized which defines a method to propagate the gradient of the loss function with respect to the ANN weights backwards from the final layer to the first. To update the weights, these partial derivatives, which represent the slope of the loss function with resect to the weights, are multiplied by the learning rate hyperparameter α and then added to the old weight value. This is represented by the formula $w_{i,j} = w_{i,j} + \alpha * \frac{\partial MSE}{\partial w_{i,j}}$. Similar to linear regression using SGD, we can utilize stochastic, mini batch, or full batch gradient descent to update the weights L1 and L2 regularization to reduce overfitting. Learning terminates upon convergence (when the partial derivatives have zero slope), after a given number of training epochs (an epoch is a full training pass over all batch iterations), or when the loss function of a validation set (discussed below) starts to steadily increase. The main ANN hyperparameters are the number of units, number of layers, activation function, loss function, batch size, regularization terms, and learning rate.

3.1.5. k-Nearest Neighbors (kNN)

The kNN algorithm predicts the IPC value for a new instance by firstly comparing its distance to all available data points and identifying the k-nearest data point neighbors. It then outputs the average of the IPC value of the k nearest data points as its prediction for the new instance. The distance formula used can vary (in this work the Euclidean one is used), but the dimensions of the inputs correspond to the number of attributes of the data points. The neighbors can be weighed either uniformly or by their distance to the new instance. The hyperparameter k acts to regularize the algorithm with a higher value generally reducing noise. Typically the kNN algorithm is one of the most straightforward machine learning methods to understand and implement. It is also advantageous because the algorithm is non-parameterized (i.e., does not make assumptions on the input data probability distribution) and easily adapts to changes in new data. The main drawbacks include prediction computation cost as well as its sensitivity to localized anomalies and biases. kNN is a lazy learning technique meaning that the computation is done at prediction time as opposed to training. To predict for a new instance, the algorithm must compute the distances of the new instance with all existing data points to find the nearest k neighbors.

3.1.6. Overheads

When deciding upon a model to implement to help solve a specific problem, it is critical to compare their overheads and see if they fall within the given problem's constraints. This is especially the case in computer architecture where even minimal latency and memory overheads may outweigh the benefits of a proposed solution.

Tab. 3 compares the computational and memory complexities for the different machine learning models. The training computational complexity is generally higher than when predicting

Table 3. Complexity overheads of machine learning models

Model	Training	Predicting	Memory	Notes
SGD linear regression	O(ndi)	O(d)	O(d)	Where n is #training samples, d is #input dimensions (attributes), and i is #iterations.
Decision tree	O($nd\log(d)$)	O($\log(d)$)	O($\log(d)$)	n and d same as SGD.
Random forest	O($tnd\log(d)$)	O($t\log(d)$)	O($t\log(d)$)	Where t is #of decision tree estimators. Both n and d are typically subsets.
ANN	O($nedu$))	O(du)	O(du)	Where e is #training epochs, and u is the total # units.
kNN	-	O($nd + nk$)	O(nd)	Where k is #neighbors.

because the algorithms tend to perform several sequential iterations over the learning data set in order to reduce the loss function. As a greedy algorithm, however, kNN does not require to be trained to compute a prediction hence it has no training computational complexity. Conversely, kNN requires large prediction computational and memory complexity since it needs to calculate the distance between the new instance with all previous data points.

The computations for training and prediction are floating point arithmetic operations and the memory complexity represent the amount of data that needs to be stored and loaded. For example, an ANN composed of 11 input parameters, two hidden layers of 6 units and one output unit consists of about $(11 + 1) * 6 + (6 + 1) * 6 + (6 + 1) * 1) = 121$ floating point (FP) weights (the +1 is due to the bias term) needed to be stored and loaded. The amount of FP computations needed to be performed at each layer l of the ANN consists of a set of FP multiplication and addition operations, $FPops_l = d_l u_l + (d_l - 1)u_l$, can be separated into multiplication and addition FP ops. In this case, d_l is the input dimension to the l^{th} layer, and u_l is the number of units in the l^{th} layer of the ANN. The computations needed for each ANN prediction is $(12 * 6 + 11 * 6) + (7 * 6 + 6 * 6) + (7 * 1 + 6 * 1) = 199$ FP ops. It is up to the architect to analyze whether these computations and memory footprints can be handled in an efficient manner so as to keep the overheads within the constraints.

3.2. Model Validation and Fine Tuning

In order to fine tune an algorithm's hyperparameters, it is useful to determine whether it suffers from high bias (i.e., prediction error) and/or high variance (i.e., overfitting) when predicting for the training data set and for the testing data set. A useful performance measure often utilized for evaluating regression problems is root mean squared error (RMSE). It provides a measure of the standard deviation of the prediction errors, and for normally distributed errors approximately 68% of the errors will fall within the RMSE value.

However, adjusting the hyperparameters based on how the algorithm predicts for the test data set will bias the training for the test data set. To make sure the testing data set is used for a final unbiased evaluation of the algorithms, the evaluation for fine tuning the models is made using a separate *validation* data set.

The validation data set is a random subset of the training data set that is kept aside (i.e., not used during the training phase) to evaluate the bias and variance of the algorithms. A more sophisticated and balanced validation method that is capable of using all of the training data for both training and evaluation is known as *k-fold cross-validation*. This method randomly splits the training data set into k subsets called *folds*. A model is then trained k times using a different evaluation fold each time and training using the remaining $k-1$ folds. For example, to train an ANN using 5-fold cross validation, the training data set will be divided into 5 folds (i.e., 5 data subsets each containing 20% of the total instances in the training data set). The ANN model will be trained 5 different times, each time using a different fold for evaluation and the other four folds for training. The final k evaluation errors can be averaged to produce a single value and additionally provide a standard deviation precision measurement.

To fine tune a model, k-fold cross-validation can be combined with a hyperparameter grid search technique. The *validation curves* illustrate how different hyperparameter values affect a model's training and validation errors. Deciding on a hyperparameter value using the validation curves is intuitive since the validation error curve will tend to decrease as the model becomes more powerful, but then increase at the point where the model complexity increases to the point of it overfitting. For example, based on the validation curve of the ANN in Fig. 4, the model chosen has 1 hidden layer of 100 hidden units. As is shown in the validation curve in Fig. 5, the kNN prediction error increases significantly after around $k = 5$. The hyperparameters of the other models were chosen using a similar grid search and validation curve analyses.

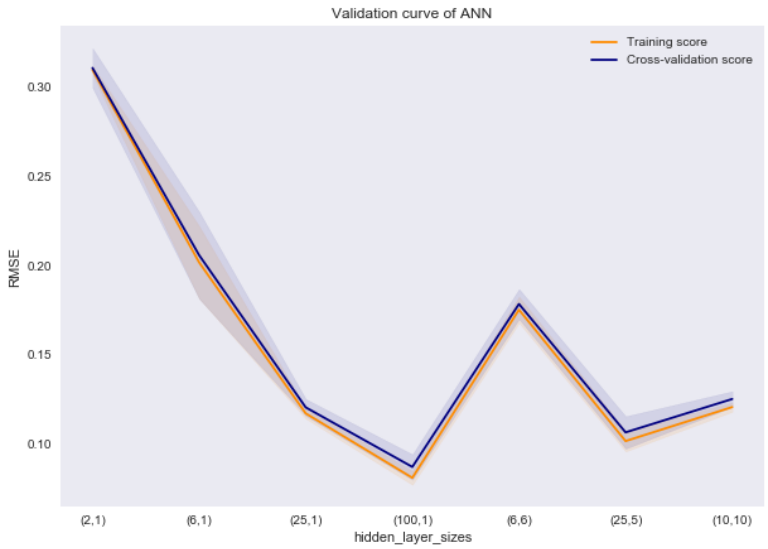


Figure 4. Validation curve of the ANN. The x axis ticks represent different models having (#hidden units, #hidden layers)

Once a set of hyperparameters for a model is chosen, it is useful to plot the *learning curves*, which visualize how the training and validation errors change as the model learns with more and

more of the training data. The learning curve for the SGD linear regression model is shown in Fig. 6. The shaded regions around the darker lines represent the standard deviations of the errors from the different cross-validation folds. Apparent from the figure is that as more train data is used during the training phase, the error decreases for the validation data set, but increases for the training data set. The standard deviations are also considerable due to the large variations between the instances and poor ability of the model to capture non linearities.

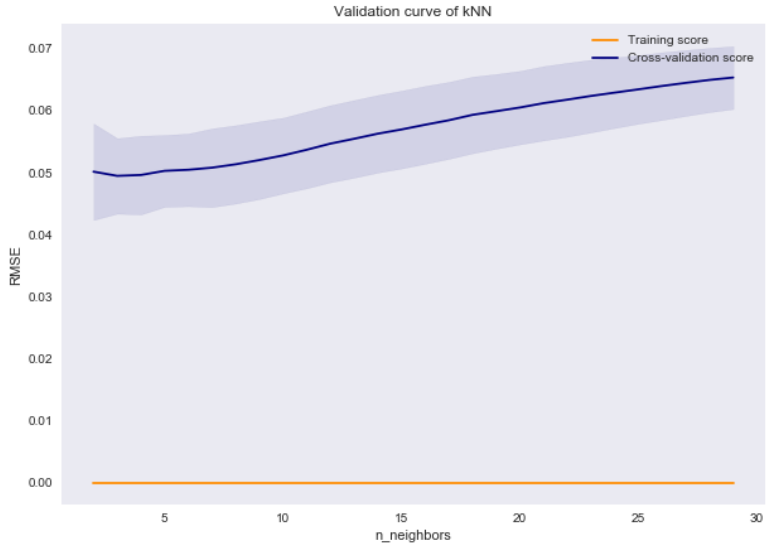


Figure 5. Validation curve of the kNN model. The x axis ticks represent values for k

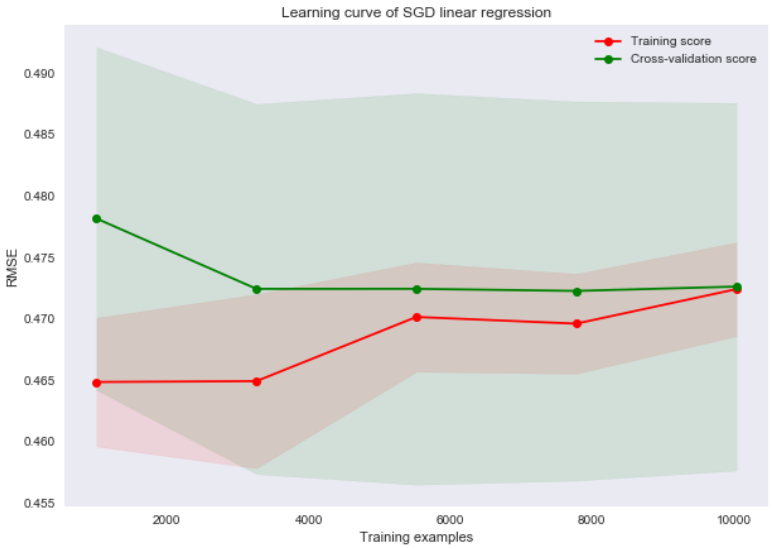


Figure 6. Learning curve of the SGD linear regression model. As the model is able to train using more data, the error decreases for the validation data set but increases for the training data set

Generally, the training error will increase as the number of instances for training increases, though it will tend to settle lower than the validation error. The bias depends upon how much

error (i.e., how high on the y axis) the training error settles at, and the variance depends upon the gap between the training and validation curves. Greater bias indicate more error and greater variance denotes that the model has probably overfit to the training data and will perform significantly worse on unseen data than on the train set. A solution to high bias is to increase the complexity of the model and the number and/or quality of attributes and data. To reduce variance, it is often useful to simplify the model by reducing complexity or adding regularization, remove input attributes, and increase the diversity and quantity of the data. If the right tails of the learning curves do not settle, then adding more training data could serve to reduce the bias.

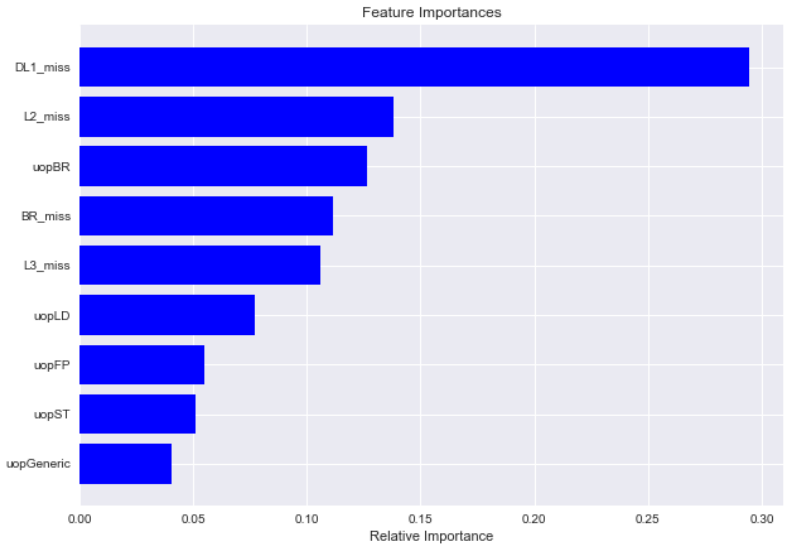


Figure 7. The feature importances of the random forest model

As mentioned previously, the decision tree and random forest algorithms are capable of ranking the importance of the input attributes. The feature importance of the trained random forest model is shown in Fig. 7. These relate closely to the correlations with the IPC that were shown in Tab. 2. If further reduction of attributes is desired (e.g., to reduce overfitting or computational or memory complexity), then feature importances will help to highlight the attributes which are most useful. For example, we could reduce the amount of attributes from 9 to 5 by keeping only those with importance value over 0.1. Then we could train a separate random forest model using these 5 attributes and compare the error and overhead results to decide which to implement.

Once a preferred set of input attributes and hyperparameters is identified using the combination of these techniques, a final version of the model is trained using the full training data set.

3.3. Final Model Results

At this point, the final machine learning models are fine tuned and trained with the whole training data set. They are ready to predict the IPC for execution quantum samples in the test data set. Tab. 4 describes the hyperparameters of the final models and compares their errors on the training, and test data sets. The results are based on the averages and standard deviation of the errors after 1,000 different runs, each time with a different set of benchmarks for the train

and test data sets. In general, the models exhibit high variance but low bias especially compared to the single attribute linear regression predictors from Section 2.4.

Table 4. Final machine learning model results. Final hyperparameters and root mean squared error (RMSE) for models with original attributes and ratio transformed attributes

Model	Final hyperparameters	Train error	Test error	Test Stdev
SGD linear regression using all 9 attributes	L1 regularization	0.3954	0.5248	0.1405
Decision tree	tree depth = 25, min leaf samples = 2, min split samples = 2	0.0255	0.6310	0.1909
Random forest	num estimators = 20, max features to evaluate = 3	0.0188	0.4981	0.1567
ANN	1 hidden layer, 100 hidden units, 400 epochs	0.0738	0.5839	0.2127
kNN	k=5, distance based neighbor weights	≈ 0	0.5516	0.1571

The significantly larger errors and standard deviation on the test set are curious and also indicative of high variance and could be the result of overfitting to the training set, but also that the diversity found in the training set unlike that contained in the test data set. Fig. 8 provides a 2-D visualization comparing how the SGD linear regressor and random forest models make their predictions as opposed to the simple single-feature linear regression model from Section 2.4. Apparent from the figure is that the machine learning based models, especially the random forest, tend to predict the training data exceptionally well but may also be a product of overfitting.

The k-nearest neighbors algorithm particularly suffers from high variance with nearly zero error on the training set and over 0.5 for the test. This is likely due to having many similar instances in the training data set. The training instances close to the testing instances also seemed to perform very differently which skewed the model's prediction. The decision tree model also exhibits significant variance between the train and test errors. This is the case even after fine tuning the model using cross validation and a hyperparameter grid search, again due to poor representation of the test data within the training data set.

For future models, it would be useful to gather more data from a wider set of benchmark suites and ensure that the training data is representative of the diversity of the benchmarks that will need to be predicted for during testing. Retraining the models as the system executes new data, the so-called online training, can also help to reduce the variance of the model.

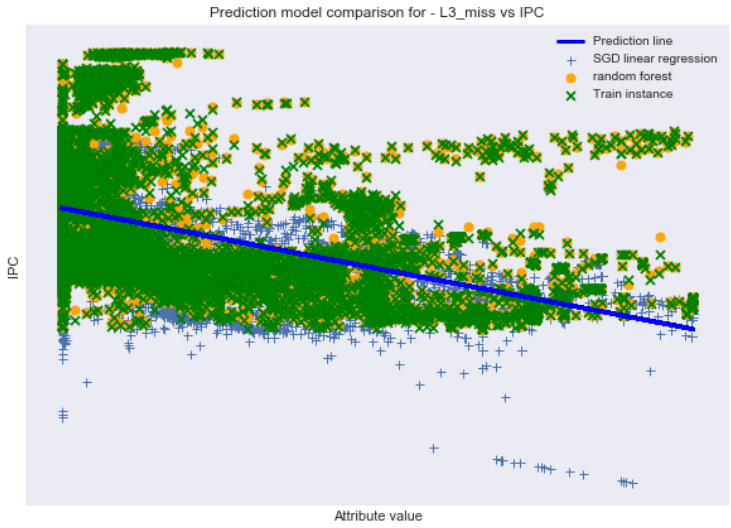


Figure 8. Scatter plot of L3 miss rate vs IPC from the training data set. Also plotted are the predictions based on the single-attribute linear regression, the multi-attribute SGD linear regression, and the random forest

The model that has the lowest prediction error on the test set is the random forest. SGD linear regression comes in at second, has the least variance between the train and test sets. However, it also suffers from an order of magnitude higher training error than any other models. In case of similar applications being frequently run on a system, the machine learning models would be able to predict significantly better than the SGD regressor, especially if making use of online learning. Though the random forest produces the least amount of test prediction error, any final implementation choice will depend upon a careful comparison of the target benchmarks, errors, and performance and space requirements.

3.4. Exploiting the Predictors

Once a final predictor has been chosen to be implemented, a mechanism must be identified which is able to exploit the extra system knowledge and translate it into system efficiency gains. The added knowledge gained thanks to the use of a predictor such as an ANN, is the IPC value for a benchmark on a hardware core for an execution quantum. A useful mechanism to exploit knowing how well workloads would perform on different core types would be a resource manager such as a CPU scheduler for heterogeneous systems. Given that several workloads may be running concurrently on a heterogeneous system composed of several cores of different types, the scheduler can utilize a specific IPC predictor per core type to predict how the workloads will perform on all other core types. The scheduler can then compare all the different possible workload to core mapping combinations and choose the one that results in the highest system IPC.

An implementation approach is to modify the scheduler code within the OS to collect the attributes and also run the predictions using the trained machine learning algorithms. This is the approach taken in [18, 19] and has been shown to produce around 30% performance improvements over state-of-the-art schedulers. Other examples of machine learning predictors being

exploited by mechanisms to improve system performance are in the area of branch prediction [9] and cache line reusability [10, 25]. Knowledge is powerful when exploited adeptly.

4. Related Work

The application of machine learning to the field of computer architecture is currently in its inceptive stages with the few exploratory studies showing impressive promise. Recently, there has been pioneering studies conducted on applying machine/deep learning to CPU scheduling. In the works [18, 19] artificial neural network performance predictors are used by the scheduler to improve the system throughput over a Linux based scheduler by over 30%. Other approaches to using machine/deep learning for scheduling has been to classify applications, as well as to identify process attributes and a program's execution history. This is the approach of [17] which used decision trees to characterize whole programs and customize CPU time slices to reduce application turn around time by decreasing the amount of context swaps. The work presented in [12] studies using structural similarity accuracy values and support vector machines and linear regression to predict thread performance on different core types at a high granularity level (1 second). In the study [6], CPU burst times of whole jobs for computational grids are estimated using a machine learning approach. An approach that utilized machine learning for selecting whether to execute a task on a CPU or GPU based on the size of the input data is done by Shulga et al. [24]. Fedorova et al. [4] proposes an algorithm that uses reinforcement learning to maximize normalized aggregate IPC. They demonstrate the need for balanced core assignment but do not provide an implementation.

For branch prediction, Jimenez et al. [9] proposed using a perceptron based predictor in order to improve CPU performance. Several studies have applied machine learning for the purpose of cache management. In the work [10, 25] the authors propose perceptron learning for reuse prediction and also present a prediction method for future reuse of cache blocks using different types of parameters. Predicting L2 cache behavior is done using machine learning for the purpose of adapting a process scheduler for reducing shared L2 contention in [23].

Conclusion

The revitalization of machine learning has led to a vast and diverse set of useful applications that affect daily lives. The ability of the algorithms to learn complex non-linear relationships between the attributes of the data and the target values has led to them being utilized as powerful prediction models. While there has been much interest recently in accelerating machine learning algorithms with custom hardware, there have been few applications of machine learning to improve system performance.

The goal of this paper has been to serve as a foundational base and guide to future computer architecture research seeking to make use of machine learning models for improving system efficiency. We have described a process to highlight when, why, and how to utilize machine learning models for improving system performance and provided a relevant example showcasing the ability of machine learning based cross core IPC predictors to help enable CPU schedulers to improve system throughput.

We have analyzed a set of popular machine learning models including stochastic gradient descent based linear regression, decision trees, random forests, artificial neural networks, and k-nearest neighbors. This was followed by a discussion of the algorithms' inner workings, com-

putational and memory complexities, and a process to fine tune and evaluate the models. After comparing the results of the predictors, the random forest narrowly produces the lowest root mean squared error in its testing predictions. Finally, we discussed how the predictor can be exploited by a mechanism such as a scheduler for heterogeneous systems in order to improve the overall system performance.

For future work, reinforcement learning may be a fruitful option to explore in using machine learning to improve scheduling. Predicting application performance and energy consumption, cache accesses, memory and I/O latencies, branch conditions, and interference effects between threads are just a few examples of useful knowledge that can help to improve system performance and energy efficiency if adequately exploited. In addition, testing the implementations on real systems is a pragmatic approach forward that helps to validate and continue pioneer applying machine learning to computer architecture.

Acknowledgements

This work has been supported by the European Research Council (ERC) Advanced Grant RoMoL (Grant Agreemnt 321253) and by the Spanish Ministry of Science and Innovation (contract TIN 2015-65316P).

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P.A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zhang, X.: Tensorflow: A system for large-scale machine learning. CoRR abs/1605.08695 (2016), http://arxiv.org/abs/1605.08695, accessed: 2018-03-01

2. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (Aug 1996), DOI: 10.1007/bf00058655

3. Carlson, T.E., Heirman, W., Eeckhout, L.: Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC. ACM Press (2011), DOI: 10.1145/2063384.2063454

4. Fedorova, A., Vengerov, D., Doucette, D.: Operating system scheduling on heterogeneous core systems. In: Proceedings of the Workshop on Operating System Support for Heterogeneous Multicore Architectures (2007), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.369.7891

5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 13–15 May 2010, Chia Laguna Resort,

Sardinia, Italy. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR (2010), http://proceedings.mlr.press/v9/glorot10a.html, accessed: 2018-03-01

6. Helmy, T., Al-Azani, S., Bin-Obaidellah, O.: A machine learning-based approach to estimate the CPU-burst time for processes in the computational grids. In: 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS). IEEE (Dec 2015), DOI: 10.1109/aims.2015.11

7. Henning, J.L.: SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News 34(4), 1–17 (Sep 2006), DOI: 10.1145/1186736.1186737

8. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22Nd ACM International Conference on Multimedia. pp. 675–678. MM '14, ACM, New York, NY, USA (2014), DOI: 10.1145/2647868.2654889

9. Jimenez, D.A., Lin, C.: Dynamic branch prediction with perceptrons. In: Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. pp. 197–206. IEEE Comput. Soc (2001), DOI: 10.1109/HPCA.2001.903263

10. Jiménez, D.A., Teran, E.: Multiperspective reuse prediction. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50. pp. 436–448. ACM Press (2017), DOI: 10.1145/3123939.3123942

11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (May 2015), DOI: 10.1038/nature14539

12. Li, C.V., Petrucci, V., Mosse, D.: Predicting thread profiles across core types via machine learning on heterogeneous multiprocessors. In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE (Nov 2016), DOI: 10.1109/sbesc.2016.017

13. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Computing 7(1), 76–80 (Jan 2003), DOI: 10.1109/mic.2003.1167344

14. Louppe, G., Geurts, P.: Ensembles on random patches. In: Machine Learning and Knowledge Discovery in Databases, pp. 346–361. Springer Berlin Heidelberg (2012), DOI: 10.1007/978-3-642-33460-3_28

15. Misra, J., Saha, I.: Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing 74(1-3), 239–255 (Dec 2010), DOI: 10.1016/j.neucom.2010.03.021

16. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013), http://arxiv.org/abs/1312.5602, accessed: 2018-03-01

17. Negi, A., Kumar, P.: Applying machine learning techniques to improve linux process scheduling. In: TENCON 2005 - 2005 IEEE Region 10 Conference. pp. 1–6. IEEE (Nov 2005), DOI: 10.1109/tencon.2005.300837

18. Nemirovsky, D., Arkose, T., Markovic, N., Nemirovsky, M., Unsal, O., Cristal, A.: A machine learning approach for performance prediction and scheduling on heterogeneous CPUs.

In: 2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 121–128. IEEE (Oct 2017), DOI: 10.1109/sbac-pad.2017.23

19. Nemirovsky, D., Arkose, T., Markovic, N., Nemirovsky, M., Unsal, O., Cristal, A., Valero, M.: A deep learning mapper (DLM) for scheduling on heterogeneous systems. In: Communications in Computer and Information Science, pp. 3–20. Springer International Publishing (Dec 2017), DOI: 10.1007/978-3-319-73353-1_1

20. Ovtcharov, K., Ruwase, O., Kim, J.Y., Fowers, J., Strauss, K., Chung, E.S.: Toward accelerating deep learning at scale using specialized hardware in the datacenter. In: 2015 IEEE Hot Chips 27 Symposium (HCS). IEEE (Aug 2015), DOI: 10.1109/hotchips.2015.7477459

21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. The Journal of Machine Learning Research 12, 2825–2830 (Nov 2011), http://dl.acm.org/citation.cfm?id=1953048.2078195, accessed: 2018-03-01

22. Polyak, B.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics 4(5), 1–17 (Jan 1964), DOI: 10.1016/0041-5553(64)90137-5

23. Rai, J.K., Negi, A., Wankar, R., Nayak, K.D.: A machine learning based meta-scheduler for multi-core processors. International Journal of Adaptive, Resilient and Autonomic Systems 1(4), 46–59 (Oct 2010), DOI: 10.4018/jaras.2010100104

24. Shulga, D.A., Kapustin, A.A., Kozlov, A.A., Kozyrev, A.A., Rovnyagin, M.M.: The scheduling based on machine learning for heterogeneous CPU/GPU systems. In: 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW). IEEE (Feb 2016), DOI: 10.1109/eiconrusnw.2016.7448189

25. Teran, E., Wang, Z., Jimenez, D.A.: Perceptron learning for reuse prediction. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE (Oct 2016), DOI: 10.1109/micro.2016.7783705

26. Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings 22nd Annual International Symposium on Computer Architecture. pp. 24–36. ACM (Jun 1995), DOI: 10.1109/isca.1995.524546

27. Yeh, T.Y., Patt, Y.N.: Two-level adaptive training branch prediction. In: Proceedings of the 24th annual international symposium on Microarchitecture - MICRO 24. ACM Press (1991), DOI: 10.1145/123465.123475