The Persistent Challenge of Data locality in Post-Exascale Era

Didem Unat, *Koç University, Istanbul, Turkey*
Anshu Dubey, *Argonne National Laboratory, USA*
Emmanuel Jeannot, *Inria, LaBRI, U. Bordeaux, France*
John Shalf, *Lawrence Berkeley National Laboratory, USA*

Abstract—
The era of exascale computing, exemplified by systems like Frontier, achieving exaflop-level performance marks a milestone. However, the quest for sheer compute power leads to strong imbalance in system design. Hence, scaling advancements in memory, network bandwidth, and storage are also necessary and poses challenges, with a crucial need to address data locality issues. This paper underscores the fundamental importance of data locality as a key abstraction for optimizing application performance. Despite notable software solutions, the growing complexity of parallelism and memory hierarchy demands performance portable data locality solutions across diverse computing platforms. The manuscript revisits data locality aspects, covering hardware considerations, application perspectives, software stack abstractions and tool support. It concludes with insights into data locality challenges and opportunities, emphasizing the ongoing significance of collaborative research for progress in this critical issue.

Introduction

The High-Performance Computing (HPC) community has achieved a milestone in the exascale era, exemplified by Frontier[1] and Aurora[2] which are capable of reaching exaflop-level performance. In addition to technological evolution in computational speeds, this level of performance also required scaling up memory and network bandwidth, storage and memory capacity, and latency by a factor of 1,000 compared to their petascale counterparts However, further improvements in memory and network efficiency are constrained by the energy required to transport data to the computational cores on a chip and within a system, resulting in a noticeable bottleneck at every level of the memory and communication hierarchy. As a consequence, the ability of the optimizing software stack to position data at the right place at the right time, defined as *data locality*, becomes a critical abstraction rather than a mere tuning option that it used to be in the past.

While there have been notable advancements in software support for data locality since Wolfe's 1989 paper 'More Iteration Space Tiling,' it is clear that the challenge continues. An imbalance has always existed between growth in concurrency and the cost of data movement, the latter being much slower to improve. Although exponential growth in concurrency has slowed down, the issue of data movement costs has worsened [3], [11] instead of improving. The efficiency of data movement in copper wires has remained stagnant over the past decade because of fundamental physics. When the size of a transistor is reduced, its intrinsic capacitance decreases, leading to improved efficiency. However, the resistance of copper wire at room temperature is already near its optimal state. When data traverses the chip for tasks like cache-coherence or inter-processor communication, it exacts a toll in terms of power consumption, and also compromises computational performance due to latency and bandwidth constraints.

The simultaneous escalation of core heterogeneity and memory diversity has created an intricate web of complexities which require data movement solutions at intra-node, inter-node, and global system communication levels. We need higher-level abstractions and algorithmic innovations to assist application developers in tackling this growing complexity and reduce the effort required to adapt codes to different computing platforms. Performance portability has now become

XXXX-XXX © 2025 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000
[1]https://www.olcf.ornl.gov/frontier/
[2]https://www.alcf.anl.gov/aurora

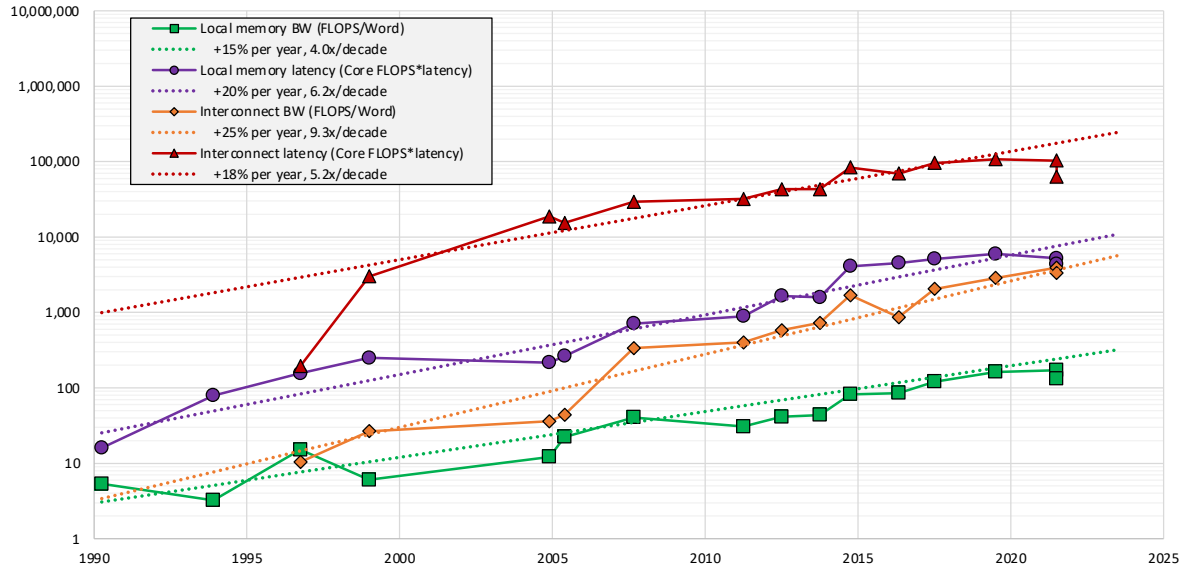

FIGURE 1. System balance evolution between memory, network and compute (The data collection and the Figure were provided by John McCalpin, Texas Advanced Computing Center).

synonymous with being able to compute on increasingly heterogeneous computing platforms.

We revisit and remind the community about this important problem as described in our community survey paper from more than a decade ago [12]. This manuscript covers various aspects of the data locality challenges that needed to be addressed then, and have since become more urgent and complex. New challenges are brought forth by two orthogonal axes of rapidly expanding diversity in platforms and scientific workflows. These challenges range from hardware considerations and application perspectives to abstractions within the software stack.

HPC community needs to foster collaborative interactions between domain scientists, computer scientists, and applied mathematicians to make progress, because, we believe that earlier model of siloed developments and innovations are unlikely to meet the growing challenges in this area. The next sections in the paper enumerate and describe these challenges in more detail individually. We begin with the lowest layers of the software stack, then address challenges at the application level before focusing on the intermediate layers (e.g., system software). The final section examines algorithms, which present challenges that span across all layers. The concluding section provides outlooks pertaining to the data locality challenges and opportunities.

Hardware Trends and Emerging Solutions

Over the preceding decades, when transistors got smaller, their operating voltage could also be reduced, which led to huge energy efficiency gains as power consumption is approximately proportional to V^2F where V is the voltage and the clock frequency is F. The phenomenal gains in energy reduction coupled with reduced gate capacitance, led to exponential increases in clock frequency. However, that stalled when devices reached a supply voltage of close to 0.5V since below that voltage it becomes difficult to turn a transistor completely on or completely off. This end of Dennard scaling forced chip manufacturers to turn from exponentially increasing clock frequencies to exponentially increasing data-parallelism. But it also led to a situation where transistors continued to improve in energy efficiency as they further shrank, but the energy efficiency of the wires (the interconnect) that connected those transistors together into a circuit ceased to improve because the resistance of those connections increased if the wire was reduced in size, or at best did not improve where on-chip wires were kept at the same diameter.

Another significant trend is related to the increasing disparity between computational power and memory performance. McCaplin highlights the fact that memory performance has been significantly trailing behind computational power, with a ratio of more than

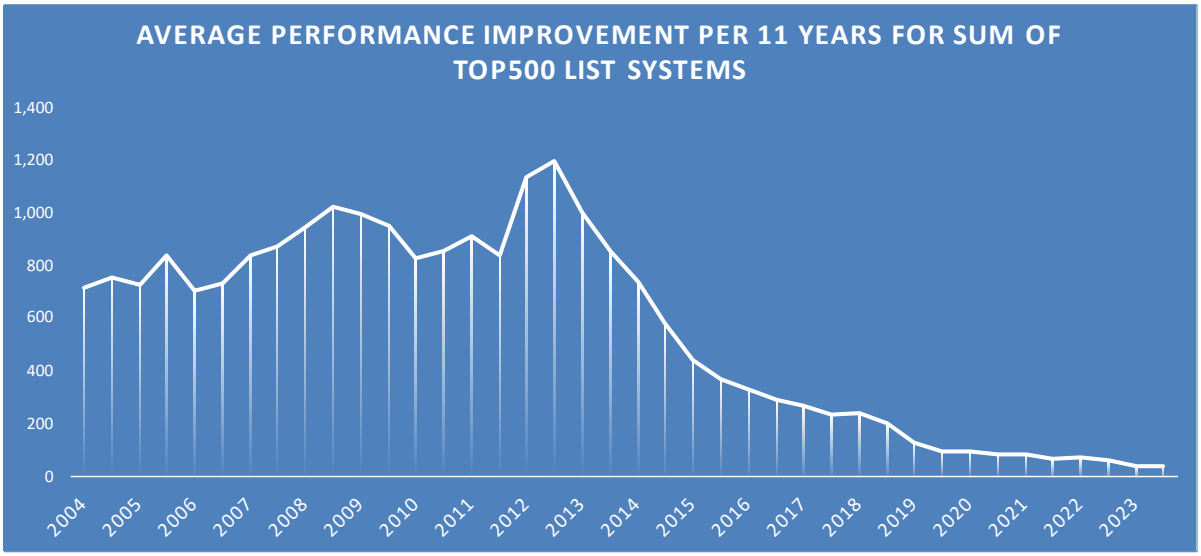

FIGURE 2. This figure shows the 11 year average rate of growth in the performance of the sum of all machines on the Top500 list from 2003 until present.

100x per decade for memory latency and 4.5x per decade for memory bandwidth [8]. More recent data, as illustrated in Figure 1, shows that the trends are continuing: memory latency is decreasing at a slower rate than the growth of processing power (Flops) (20% per year on average). As non-uniform memory access has become the standard approach for designing and configuring multicore/multiprocessor systems, and with the emergence of heterogeneous memory systems featuring technologies like High Bandwidth Memory (HBM), Dynamic Random Access Memory (DRAM) and Non-volatile memories simultaneously, the cost of misplacing data objects (buffers) has become highly detrimental and problematic at the node level. In large-scale supercomputers, as noted by the authors in [7], they have observed a similar trend: a decrease in the ratio of total memory bandwidth to the HPL performance of top-tier supercomputers. This ratio dropped from 1.18 for BlueGene L in 2009 to 0.13 for Frontera in 2019. Consequently, the need for locality-aware tools and methodologies to comprehend the various unexplored trade-offs has become critical for optimizing performance in modern HPC systems.

As the performance of individual processing cores has ceased improving, there are moves to exploit specialization and even accelerators that exploit new dimensions of parallelism beyond data-parallelism. The industry has embraced chiplets, where different types of cores are integrated into a single processor chip, as a means to co-integrate diverse heterogeneous accelerators – mostly driven by the AI/ML market. Some of these emerging architectures employ a classical dataflow execution model, which creates new opportunities and challenges of exposing and preserving data locality.

Device-centric communication as mechanisms for multi-accelerator execution has become the trend to reduce the involvement of the CPU in the critical path as a way to improve data locality [5]. Historically, Network Interface Cards (NICs) were typically connected directly to the CPU or, in more recent designs, to a PCIe switch linked to both the CPU and GPUs. Given that the data primarily resides in GPU's memory, they are now being placed on the GPU-device to reduce latency. For example, AMD's Infinity Fabric is a high-performance interconnect technology designed to facilitate communication and data transfer between various components within AMD processors and GPUs. Similarly, NVLink of Nvidia facilitates high bandwidth and low latency direct access between Nvidia GPUs and addresses the bandwidth limitations of PCIe, which has been observed to be a transfer bottleneck in GPU-accelerated applications.

Hence, in summary, the challenges of the end of Dennard scaling and of the cost of data movement remain, with the latter becoming worse over the past decade. The emerging challenge is the slowing of Moore's Law that has led to a dramatic slowdown in the rate of performance improvement for HPC systems. As shown in Figure 2 there was a consistent

500x improvement every decade in performance in the early days, but that rate has slowed down to less than 10x every decade. The burgeoning AI/ML market has responded to the Moore's Law slowdown with a plethora of heterogeneous accelerators and memories as a means of continuing performance growth through architecture specialization and extensive use of lower/mixed precision. These trends are bound to penetrate into the HPC ecosystems also.

Application Software Trends

Traditional scientific applications often see a positive feedback loop, where an increasing level of understanding leads to the development of more complex models. These models, in turn, demand the creation of more complex software. Furthermore, end-to-end workflows are evolving to incorporate a diverse range of models. These workflows can encompass a spectrum of computational techniques, including partial differential equation solvers, algebraic solvers, AI/ML-based models, and other data analytics components, all integrated into a single process. These trends increase the diversity of data and memory access patterns, usually with significantly adverse effect on data locality of the application.

It is not obvious at all that the challenge of diminishing locality has even begun to be appreciated by the applications community. In the current generation of platforms most of the computational power resides in GPUs, motivating applications to move their flops to the GPUs. That merely changes the focus from one type of data locality (maximizing on-node computation and cache reuse) to node level data locality with massive fine-grained parallelism. In the worst case, applications maintain two algorithmic variants for their software if there is significant differences between CPU and GPU versions and they intend to use both.

However, with the landscape shifting towards specialized hardware as mentioned in the previous section, the next generation of HPC machines are likely to be very different. Also, routine inclusion of AI/ML models at production level in simulations is still in its infancy. So true reckoning with the impact of complex workflow on data locality has not occurred for a vast majority of applications. If the trend towards specializations accelerates, and in many ways that seems the only way to gain greater computational capacities, mere band-aid solutions are unlikely to yield needed outcomes. From the scientific workflow's standpoint this is a three-way challenge — understanding the computational behavior of the application to identify the potential for data locality, understanding the map of data locality to specific hardware resources, and the ability to express the map to program models effectively. For application developers this will imply greater emphasis on flexible and composable software design, and for abstractions and programming models this will imply providing ways to expose data locality of the application to optimizers and computation mappers. Neither of these objectives can be achieved without changing the development model from siloed one to collaborative one as mentioned earlier.

Heterogeneous Memory and Storage

In many systems, we are facing heterogeneous memory with fast but small memory (such as High Bandwidth Memory) or large but slower memory (such as Non-Volatile Random Access Memory). Managing data placement and buffer allocation across these levels is crucial for performance.

Several software libraries, such as *libnuma* for managing Non-Uniform Memory Access (NUMA) nodes and *memkind* for high-bandwidth memory allocation, expose memory system functionality and provide specialized behavior. However, these tools operate independently and lack a unified approach to managing heterogeneous memory across tiers. For instance, libnuma places memory pages in specific NUMA nodes, and memkind allocates memory in faster memory tiers. Yet, neither considers node-specific performance or capacity constraints for data placement. Additionally, OpenMP-level tools manage buffer allocation in different memory kinds but fail to address key challenges, such as dynamic data migration between tiers and transparent interception of memory allocation without modifying application code. What is missing is not just additional tools but a composable, automated framework that integrates existing libraries to seamlessly handle memory placement, migration, and allocation across heterogeneous memory systems.

Another key challenge in launch-time optimization is multi-layered mapping, which involves assigning resources across various system hierarchies—cores, sockets, nodes, and racks—while accounting for communication patterns and data locality. Effective mapping must balance intra-node locality to optimize memory access and inter-node locality to minimize communication overhead. However, current tools lack the ability to automate this process across all levels.

In addition, application phase management is essential for performance. Applications often progress through distinct phases, each with unique resource and data requirements (e.g., data loading, computation, aggregation). Ideally, resource mappings should adjust

dynamically to match these changing patterns.

While tools like *hwloc* handle node-level topology, we lack portable tools for managing inter-node and storage hierarchies, which complicates data-locality-aware resource selection. For example, automatic I/O node selection on systems like Theta can lead to suboptimal data access if storage locality is not considered. Addressing these gaps requires dynamic, phase-aware mapping frameworks that optimize both data movement and resource usage throughout an application's lifecycle.

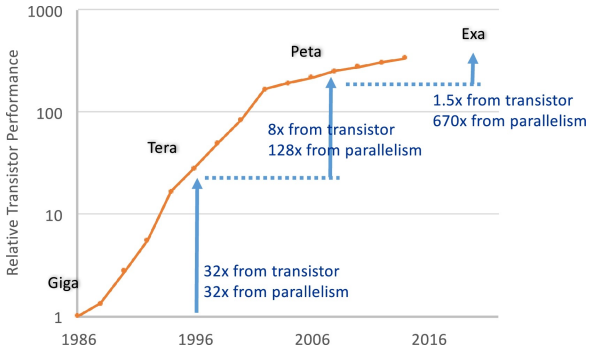

FIGURE 3. More performance is derived from parallelism than from transistor performance. Figure was provided by Shekhar Borkar, Qualcomm Inc.).

Emerging Programming and Execution Environments

When distributed memory parallelism was the dominant model, scaling optimizations could often be addressed separately from arithmetic and computational optimizations. These scaling strategies tended to generalize well across platforms with minimal code adjustments, while most of the platform-specific tuning focused on node-local optimizations to improve cache locality. As mentioned in the previous section this paradigm has continued to hold in the current generation of platforms, but with significant increase in node-level complexity. The dominant approach to implementing high-performance computing (HPC) applications is to use hybrid models that can be generalized as MPI + X (e.g. OpenMP, CUDA, SYCL, HIP, or C++ based abstractions) where MPI handles offnode parallelism, while X handles node local parallelism. While Partitioned Global Address Space (PGAS) models have seen limited adoption, Nvidia's NVSHMEM has been gaining popularity. All of these approaches have permitted the applications to remain viable with relatively modest modifications largely focused on restructuring of hot-spots in the solvers.

However, with hardware within nodes getting more complex as shown in Figure 3 these solutions are unlikely to remain sufficient to address the variety of computational patterns that may emerge. The HPC community needs a fundamental shift in programming and execution environments to support features critical for optimizing data locality within nodes. These features should include:

- Explicit expression of locality. The working set of data should be mappable to specific memory hierarchies and specialized devices (e.g., GPU memory, HBM, or shared caches) without requiring extensive manual data restructuring.
- Separation of algorithmic logic from memory-specific optimizations. The ability to express invariant components of software independently of data layout allows flexibility, akin to how C++ abstractions like templates facilitate this separation.
- Efficient specification or inference of computation and data movement. To avoid costly memory transfers, the system must efficiently map computations and their associated data to the appropriate devices, minimizing latency and optimizing resource utilization.

While some of these features are available in existing tools and frameworks (e.g., CUDA or OpenMP for explicit memory placement, polyhedral methods for dependency analysis), they remain incomplete. Additionally, Python's widespread use in deep learning has made it a contender in the HPC space even though it lacks a performant ecosystem for anything other than deep learning so far. New Programming langagues (e.g. Chapel, Julia), and enhancements to existing languages provide better solution, however, while transitioning to a new programming language might be optimal in the long term, it is impractical for the short term. To address sustainability concerns and verification challenges, an incremental adoption strategy for existing codes is essential. In the next section we discuss how redesigning programming systems can help in mitigating some of these challenges.

Programming Systems

The traditional languages of choice in HPC applications have been C/C++/Fortran, with Python rapidly gaining because of deep learning as mentioned earlier. All of these languages lack mechanisms to make data dependencies and locality constraints explicit in the source code. Instead, memory is often treated as a

linear, sequential space, leaving the burden on compilers or runtimes to infer efficient layouts—frequently with suboptimal results. What is missing are high-level constructs that expose data locality and memory hierarchies directly within the language semantics. Cache-friendly data layouts and tile-based decompositions, for example, are crucial for modern architectures but are difficult to express and manage in languages designed for sequential execution. Polyhedral methods have proven effective for optimizing loop nests and analyzing data dependencies in regular workloads but struggle with dynamic or irregular data structures, making them insufficient as a universal solution.

A more comprehensive solution would involve integrating data locality annotations, hierarchical memory abstractions, and predictable data movement patterns directly into programming models. Effective abstractions for data locality in any programming model must balance low overhead with high-level semantic information. This includes providing data dependency details critical for compiler optimizations and runtime efficiency. Innovative approaches like using the visibility algorithm [1] from computer graphics for dependency analysis offer promise. Addressing side effects is equally important, where declarative solutions and modern languages provide notable advantages over traditional languages, which often rely on conservative assumptions about external function calls. Abstractions should accommodate both automatic and explicit data movement, with varying levels of control tailored to specific use cases.

Tiling based programming models point to a way forward by providing a structured way to express and manage memory through tiled computation patterns. While these show promise in balancing portability and performance, they cannot do so without being deeply integrated into the application's computational structure, which may necessitate significant code refactoring.

Many computational science and engineering applications follow a bulk-synchronous model, which simplifies dependency handling for regular applications. However, hardware heterogeneity has made lockstep execution increasingly impractical. To address this, the growing synchronization costs associated with extreme parallelism must be mitigated. This calls for a reevaluation of Bulk Synchronous Programming (BSP) models in favor of task-based programming. A well-designed API that supports hierarchical decomposition and dependencies can allow applications to account for runtime locality without requiring a shift to functional programming. Current task based models are too general, therefore have too much overhead and nearly impossible to integrate with existing applications.

Beyond static analysis, profiling, auto-tuning, and user feedback play essential roles in optimizing performance. While user-provided hints will remain relevant, automated, measurement-driven tools are better suited for scaling to large codes. The hierarchical nature of modern hardware is well-established, but its exposure to users varies. Domain-specific solutions, tailored to specific hardware hierarchy levels, may offer the best outcomes. The success of new programming systems hinges on effective application adoption and collaboration between introspection and application guidance for optimization through performance models and tools described in the next section.

Performance Modeling and Tools

Performance models are essential for guiding the software stack (e.g., compilers and runtime systems) or aiding programmers by providing a comprehensive application model integrated with a machine model. Current modeling approaches encompass static, dynamic, empirical, and learning-based cost models, each complementing the other. In addition, learning-based models maintain databases of application configurations and tuning parameters, enabling performance predictions for future runs. AI-assisted tools are rapidly emerging to help users interpret and analyze profiling data. A key advancement for these tools is extending their capabilities to support data locality optimizations and offer actionable suggestions for code improvements.

The primary challenge in performance modeling lies in creating models that are both concise and precise while ensuring their practical utility and feasibility for real-world applications. Several Roofline-based modeling tools have been developed to extract the arithmetic intensity of applications, gaining widespread adoption among leading vendors. Some of these tools also offer guidance to users on improving performance and data locality by analyzing profiled data. Although numerous tools are available for measuring, monitoring, and analyzing performance—some specifically focusing on data movement—understanding the data generated by tools is almost an expertise of its own which most users lack. Moreover, very few tools [6] provide information about cross-node communication for newly emerged technologies such as NCCL, RCCL, and NVSHMEM. Lastly, tools for power monitoring are limited, either because power consumption is considered less critical to end-users compared to performance, or because vendors provide only restricted means for its measurement.

In short, significant opportunities for improvement include: i) enhancing the presentation of data to be more intuitive for programmers, facilitating reasoning about data locality optimizations; ii) addressing the lack of support for correlating power consumption with performance; and iii) incorporating advanced visualization and feedback mechanisms to assist end-users in performance debugging through AI or non-AI based solutions.

Algorithms

The focus of performance portability across different hardware architectures has largely been on data structures and access patterns without changing the algorithm. In many instances this is insufficient – some algorithms simply do not map well on certain architectures. For example algorithms with random or irregular access patterns are not well suited for GPUs. At times algorithms acquire irregularity in an attempt to reduce computation, an approach that is helpful for better performance on the CPUs. In those instances, rethinking the control structure of a computation can significantly enhance its data locality on target devices. For example, in the context of SuperLU, by redesigning the algorithm with 3D communication-avoiding techniques, a 24x speedup was achieved [10], demonstrating strong scaling to 4,096 GPUs.

In contrast, there are applications where irregularity is unavoidable in known algorithms. In the absence of any new algorithm without irregularity emerging, the applications may resort to profiling and user intuition with fine tuning to minimize the cost incurred due to irregularity. Given that fine tuning is usually platform specific the application may end up with several versions of the implentation. A concommitant challenge is maintaining these algorithmic variants for different devices. C++ based abstractions solved this challenge of multiple implementation variants for data structures and patterns, but they cannot handle algorithmic variants. CG-Kit [9] is a new tool that confronts some of these challenges. It generates code control structure from a recipe while user defined arithmetic sections act as building blocks for the code. For this, or other similar approaches to be effective cost-benefit analysis is necessary, which in turn needs performance modes described in section .

In another development, mixed precision has proven highly effective in AI workloads and offers significant potential for HPC applications. The primary advantage lies in reducing data movement rather than lowering compute costs [2] as lower-precision data requires less bandwidth and storage, leading to faster communication and reduced energy consumption. However, leveraging mixed precision in HPC requires tools to identify which parts of the code can tolerate lower precision and which require higher precision. Moreover, the concept of precision is evolving, and adopting mixed precision is a promising strategy for incorporating emerging AI accelerators in HPC. Numerous studies demonstrate that HPC can benefit from mixed precision by revising algorithms to accommodate precision loss. Achieving this, however, necessitates close collaboration between domain experts and computer scientists. Lastly, to seamlessly integrate mixed precision into applications, new algorithms and support from programming systems, as well as verification methods, are essential.

Outlook

Over the last decade, although the pace of exponential growth in concurrency has moderated, the relative cost of data movement has become worse. The machine imbalance between compute speed and memory bandwidth has been increasing at a rate of 15–30% per year [4]. Thus, data movement optimizations both in the hardware and software are more important than ever. Next, we highlight some of the challenges and opportunities for data locality management research (both hardware and software) in the near future.

- HPC is no longer solely about performance; data locality solutions now also need to consider programmer productivity and code portability.
- With the rising costs of data movement and increasing energy consumption in large-scale systems, energy-efficient data management strategies will become critical. Research should focus on optimizing algorithms and hardware designs to minimize the power footprint of computing. This requires a shift in mind set where performance per watt becomes a major metric.
- HPC systems are not only used for simulation but also for the training and inference of AI models. Large-scale models trigger new challenges as they require managing both training data and models efficiently across nodes and within a node.
- Even though we have entered the exascale era, it is unlikely that we will achieve 100 exaflops by the end of this decade using the same path that led to the first exaflop system. Thus, any performance improvements will likely come from algorithmic changes, software optimizations, and co-design.

- Codesign integrates hardware and software development to optimize performance by abstracting hardware diversity through libraries, allowing applications to indirectly leverage hardware features. Adopting this approach more broadly can help address data locality challenges effectively.
- It is no longer sufficient to focus on programming languages alone. We need to work within an ecosystem to provide automation for data locality management.
- Generative AI can be helpful in writing data-locality-aware code, providing hints on directives, helping in performance debugging by analyzing and profiling the code.
- In order to mitigate the increasing synchronization costs associated with achieving extremely high levels of parallelism, it becomes essential to reassess conventional programming models and adopt modern alternatives, such as task-based programming models.
- Hiding communication latency is becoming progressively challenging. It is necessary to develop new algorithms aimed at reducing and hiding communication, sometimes even at the expense of redundant computation.
- Existing libraries that offer specialized memory management lack a unified approach for handling heterogeneous memory; a composable, automated framework integrating these tools is needed to streamline memory placement, migration, and allocation.
- As systems grow larger, the cost of data movement in fault-tolerant mechanisms (e.g. checkpoint-restart or data replication) becomes significant. Research into resilient algorithms that reduce unnecessary data movement while maintaining fault tolerance is crucial.
- The notion of precision is undergoing a transformation. Employing mixed precision represents a promising approach to mitigate data movement. To fully integrate mixed precision in applications, there is a requirement for the development of new algorithms and support from programming systems.
- Traversing the chip for cache-coherence or other inter-processor communication is costly either in power (if the on-chip data paths are over-designs) or in computational performance due to latency and bandwidth bottlenecks. Thus locality management even within the chip has never been more important.
- Advanced packaging has enabled tighter integration of accelerators and CPUs, such as the Accelerated Processing Unit (APU) where the GPUs and CPUs are co-integrated in the same package. As chiplets technologies become more pervasive, we anticipate heterogeneous integration of other features such as co-packaged optics to offer massive bandwidth between components of future systems.
- New emerging technologies (Compute Express Link, Processing in Memory, and Nonvolatile Memory) raise new challenges in terms of performance and data access. Integrating these technologies can enhance data locality by enabling dynamic memory pooling, efficient access, and reduced data movement.

Conclusion

In summary, this manuscript aims to unite researchers around the crucial concept that prioritizing data locality is fundamental for organizing computation. The evident tight connection between data locality and parallelism highlights the predominant focus of existing programming abstractions on compute-related concepts. It is now imperative to recognize data locality as the cornerstone of computation, especially as hopes for architectural improvements wane. Potential performance enhancements are anticipated through changes in algorithms, software optimizations, and comprehensive system redesigns.

Within the HPC community, there is active exploration of innovative approaches to describe computation and parallelism while minimizing data movement. The nearing maturity of some projects in this field underscores the urgency for research collaborations during this critical phase. Achieving these objectives requires co-design, community organization, thoughtful consideration of our impact on the software stack, and collaborative efforts to develop interoperable solutions for data locality.

ACKNOWLEDGMENTS

Authors would like to thank the 6^{th} PADAL (Programming and Abstractions for Data Locality Workshop) participants for engaging in lively discussions at the workshop, which greatly contributed to the compilation of this manuscript. Dr. Unat has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 949587). This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

REFERENCES

1. M. Bauer, E. Slaughter, S. Treichler, W. Lee, M. Garland, and A. Aiken. Visibility algorithms for dynamic dependence analysis and distributed coherence. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, page 218–231, New York, NY, USA, 2023. Association for Computing Machinery.
2. Q. Cao, S. Abdulah, H. Ltaief, M. G. Genton, D. Keyes, and G. Bosilca. Reducing data motion and energy consumption of geospatial modeling applications using automated precision conversion. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 330–342, Los Alamitos, CA, USA, nov 2023. IEEE Computer Society.
3. V. Cavé, R. Clédat, P. Griffin, A. More, B. Seshasayee, S. Borkar, S. Chatterjee, D. Dunning, and J. Fryman. Traleika glacier: A hardware-software co-designed approach to exascale computing. *Parallel Comput.*, 64(C):33–49, May 2017.
4. J. Dongarra, M. Gates, P. Luszczek, and S. Tomov. Translational process: Mathematical software perspective. *Journal of Computational Science*, 52:101216, 2021. Case Studies in Translational Computer Science.
5. I. Ismayilov, J. Baydamirli, D. Sağbili, M. Wahib, and D. Unat. Multi-gpu communication schemes for iterative solvers: When cpus are not in charge. In *Proceedings of the 37th International Conference on Supercomputing*, ICS '23, page 192–202, New York, NY, USA, 2023. ACM.
6. M. K. T. Issa, M. A. Sasongko, I. Turimbetov, J. Baydamirli, D. Sağbili, and D. Unat. Snoopie: A multi-gpu communication profiler and visualizer. In *Proceedings of the 38th ACM International Conference on Supercomputing*, ICS '24, page 525–536, New York, NY, USA, 2024. ACM.
7. A. Khan, H. Sim, S. S. Vazhkudai, A. R. Butt, and Y. Kim. An analysis of system balance and architectural trends based on top500 supercomputers. In *The International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2021, page 11–22, New York, NY, USA, 2021. Association for Computing Machinery.
8. J. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. HPCWire https://www.hpcwire.com/2016/11/07/mccalpin-traces-hpc-system-balance-trends, 2016.
9. J. Rudi, Y. Lee, A. H. Chadha, M. Wahib, K. Weide, J. P. O'Neal, and A. Dubey. CG-Kit: Code generation toolkit for performant and maintainable variants of source code applied to Flash-X hydrodynamics simulations. *Future Generation Computer Systems*, 163:107511, 2025.
10. P. Sao, X. S. Li, and R. Vuduc. A communication-avoiding 3d algorithm for sparse lu factorization on heterogeneous systems. *Journal of Parallel and Distributed Computing*, 131:218–234, 2019.
11. J. Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190061, 2020.
12. D. Unat, A. Dubey, T. Hoefler, J. Shalf, M. Abraham, M. Bianco, B. L. Chamberlain, R. Cledat, H. C. Edwards, H. Finkel, K. Fuerlinger, F. Hannig, E. Jeannot, A. Kamil, J. Keasler, P. H. J. Kelly, V. Leung, H. Ltaief, N. Maruyama, C. J. Newburn, and M. Pericás. Trends in data locality abstractions for hpc systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):3007–3020, 2017.

Didem Unat has been a faculty member at Koç University since 2014. She is known for her work on programming models, performance tools, and system software for data locality. She received her PhD degree from the University of California, San Diego in Computer Science. She was named the "Emerging Woman Leader in Technical Computing" by ACM SigHPC in 2021.

Anshu Dubey is a Senior Computational Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory and a Senior Scientist at the University of Chicago. Her research interest include sustainability of scientific software, high-performance computing technologies, performance portability and software architecture. She received her Ph.D. in computer science from Old Dominion University.

Emmanuel Jeannot is a senior research scientist at Inria. He earned his PhD degree in computer science from the Ecole Normale Supérieure de Lyon (France) in 1999. Since 2009, Jeannot has been at INRIA Bordeaux Sud-Ouest, where he leads the TADaaM team, and at the LaBRI laboratory of the University of Bordeaux. His primary research interests include runtime systems and topology-aware algorithms.

John Shalf John Shalf is Department Head for Computer Science Lawrence Berkeley National Laboratory, and recently was deputy director of Hardware Technology for the DOE Exascale Computing Project. Shalf is a coauthor of over 80 publications in the field of parallel computing software and HPC technology.