Data Locality in High Performance Computing

Subjects: Computer Science, Artificial Intelligence

Contributor: Sardar Usman , Rashid Mehmood , Iyad Katib , Aiiad Albeshri

Big data has revolutionized science and technology leading to the transformation of our societies. High-performance computing (HPC) provides the necessary computational power for big data analysis using artificial intelligence and methods. Data locality is a broad term that encapsulates different aspects including bringing computations to data, minimizing data movement by efficient exploitation of cache hierarchies, reducing intra- and inter-node communications, locality-aware process and thread mapping, and in situ and transit data analysis.

High-performance computing (HPC) big data High-Performance Data Analytics (HPDS) data locality

1. Introduction

High-performance computing (HPC) has been fundamental in developing many transformative applications [1][2][3][4][5]. These HPC applications require careful design of fundamental algorithms, e.g., the seven dwarfs [6][7][8][9][10][11]. Data locality has been coined as a key aspect among others and is regarded as one of the main challenges for exascale computing endorsed by many technical reports published in recent years [12][13][14][15]. The organization of memory into banks and NUMA regions, read-only memory, and multiple cache hierarchies make efficient data structure and optimization, a complex task. As we are heading towards exascale systems where the cost of data movement will be a dominant factor in terms of performance and energy efficiency. The complexities of managing data with different levels of memory hierarchies are further complicated by inter- and intra-node-level communication. Parallelism and communication are further constrained by heterogeneous core architecture. With the increase in platform heterogeneity, portability is a core issue that demands standardization of widely used approaches to automate performance tuning across different architectures [16].

The concurrency and input data size are on a rise and there is a need for efficient exploitation of heterogeneous architectures with an increasing number of cores to bridge a performance gap between application and target architecture. There is a need for optimization of data layout, data movement between processes/threads and data access patterns. Locality issues exist at different levels, from how application data is laid out to the increasing number of processing cores, complex memory hierarchies, inter-node communication, interconnects, and storage units [17]. The following section discusses the research related to data locality in the high-performance computing domain.

2. Application Perspectives

Data locality from an application's point of view demands the examining of a range of modeling methodologies, which needs exploration of a huge application space. The adaption of current HPC application code into exascale systems demands efficient utilization of heterogeneous resources, requiring innovations in application layout strategies, communication optimization, synchronization, resource availability, and data movement and access strategies.

3. Programming Languages, Compiler, and Libraries

The increasing level of hardware parallelism and deep hierarchies (core dies, chips, to node level) have posed a challenging task for efficient parallelism and data locality exploitation at all levels of hierarchy, which demands data locality support at different levels of the software ecosystem, i.e., programming language, compiler, libraries, etc. Majo et al. [18] proposed a parallel programming library for compose-able and portable data locality optimizations for NUMA systems. This library is based on Intel Threading Building Blocks (TBB) and allows the capturing of a programmer's insights for mapping tasks to available resources. Lezos et al. [19] presented a compiler-directed data locality optimization in MATLAB by developing a software tool, MemAssist, for efficient exploitation of targeted architecture and memory hierarchy. Regan-Kelley et al. [20] proposed a language and compiler support for optimizing parallelism, locality, and re-computation in image processing pipelines.

Current HPC models lack efficient exploitation of data locality due to the lack of language support for data locality policies e.g., array layouts, parallel scheduling, etc., an overexposure of target architecture, and a lack of support for user-defined policy abstractions. Chapel is a prominent parallel programming language developed by the joint efforts of Cray Inc., academia, and industry with a prime focus on the separation of parallelism and locality, and multi-resolution design with enhanced productivity features [21]. X10 [22] is based on a PGAS model and designed specifically for parallel computing that supports structured and unstructured parallelism, user-defined primitive struct types, and globally distributed arrays with co-location of execution and data. Huang et al. [23] addresses the data locality issues with explicit programming control of locality in the context of OpenMP and how it can be accomplished.

4. Cache Optimization Techniques

Locality optimizations by making efficient use of the cache, have been an active area of research for years. Locality can be categorized as temporal locality (reuse data once it has been brought into the memory) and spatial locality (use of every data element brought into the memory). The efficient exploitation of registers involves compiler, assembly-language, and programming-level optimizations. Cache line length, cache size, and cache replacement policy are some of the factors considered for the effective and efficient optimization of caches [24]. Gupta et al. [25] proposed a spatial locality-aware cache partitioning scheme by measuring spatial and temporal locality dynamically for optimal workload block size and capacity for effective cache sharing. Gonzalez et al. [26] proposed a dual data-cache organization for managing spatial and temporal locality by implementing a lazy cache policy that uses a

locality prediction table to make necessary predictions based on recently used instructions. [27][28] related to on-chip caching for efficient cache utilization, while [29][30] based their approach on data-access frequency.

The following section explains the basic data access optimization techniques engineered to improve cache efficiency.

4.1. Data Access Optimization

Data access optimizations are generally code transformations whose prime motivation is to increase temporal locality by reordering iterations in a nested loop. These optimization techniques are also used to expose parallelism and help in vectorizing loop iterations. Compilers used heuristics to decide the effectiveness of applying these transformations. If nested loops are not perfectly nested then loop skewing, unrolling, and peeling are used. Detailed information about these techniques can be found in [31][32].

Loop Skewing: When the carried dependencies prevent parallelizing, one can skew the loop-nest by modifying the aligning of iteration space coordinates and relabeling the statement instances in new coordinates [33].

Loop Peeling: Unfolding a few iterations of the loop to eliminate loop-carried dependencies is known as loop peeling.

Loop Interchange: When the order of a loop is not important then a loop-interchange transformation reverses the order of two adjacent loops in the loop-nest making sure all dependencies are preserved. Loop interchange is used to improve locality, enhance parallelism, register reuse, and vectorization [34].

Loop Fusion/Jamming: This technique involves the fusion of two adjacent loops having the same iteration space traversal into a single loop resulting in increased instruction level parallelism and data locality. The opposite of loop jamming is loop distribution, which divides a single loop into multiple loops with no carried dependencies. [34].

Loop Tiling: This loop transformation technique improves data locality by increasing the reuse of data in the cache by increasing the depth of the loop nest [35]. Selection of the best loop tile shape and size is a fundamental problem. Most of the current multicore processors have a shared last-level cache (LLC) and its space allocation depends on the co-execution of applications, which may cause interference in the shared cache [36]. Bao et al. [36] proposed a static compiler-based defensive tiling to choose the best tiling size for optimal performance. Some of the works related to loop tiling include [37][38] for the reduction of capacity misses, [39][40][41] for reducing communication, and [42][43][44] for auto/dynamic tuning of loop tiling.

4.2. Data Layout Optimizations

Data access optimization techniques may not be an optimal choice for data locality for computations with conflict misses, while data layout optimizations improve spatial locality by arranging data structure and variables in memory. Some of the most commonly used techniques for data locality optimization are inter- and intra-array

padding, array merging, array transpose, etc. [34]. Parallelism and efficient exploitation of data locality have been considered separate objectives in the literature. Kennedy et al. [45] explored this trade-off between data locality and parallelization by proposing a memory model to determine the reuse of cache lines. The model uses loop optimization algorithms for efficient exploitation of data locality and in-memory optimizations.

4.3. Cache Bypassing

The depth of cache hierarchies and cache size have been increasing to meet the high processing demands, along with mounting more cores on the chip. The performance of an application with little data reuse is severely affected by cache use. Researchers over the years engineered different techniques to effectively bypass the cache to improve the performance of an application. The potential benefits of cache bypassing are obvious but bring many challenges including implementation overhead, memory and performance overhead, etc. The different cache techniques, their potential benefits, challenges, and taxonomy is explained in detail by Sparsh Mittal [46].

5. Locality-Aware Scheduling and Load-Balancing

Applications need to exploit parallelism at multiple scales at the fine granularity and across a variety of irregular program and data structures and program inputs. Parallel algorithms demand extra space to enable the temporal decoupling necessary for achieving parallelism, as compared to sequential algorithms, which attempt to minimize space usage. As applications are becoming more and more data-intensive and task execution involves the processing of a huge volume of data, optimal load balancing, and locality-aware scheduling are critical issues to be resolved at the highest priority for exascale systems. Locality optimization (horizontal—between processing elements and vertical—between levels of hierarchy)) has a direct impact on energy consumption for exascale systems. Scheduling millions of tasks per second within latency constraints is one of the major challenges and current centralized scheduling systems (Falkon [47], SLURM [48], SGE [49], and Condor [50]) are deprived of handling fast scheduling. This issue is addressed by Ousterhout et al. [51] by presenting a stateless distributed scheduler called sparrow, which supports data-aware scheduling to help the collocating of task and input data.

Load balancing can be achieved by work stealing but the random migration of tasks results in poor data locality. Load balancing is a challenging task in a fully distributed environment as the scheduler has information about its own state. Load balancing techniques have been extensively studied over the years and can broadly be classified into static and dynamic techniques. These techniques achieve optimal load balancing in a centralized or distributed manner. Although work stealing is a widely used efficient load balancing technique e.g., OpenMP [52], Cilk [53], X10 [22], random work stealing results in poor scalability on large-scale systems [54]. Falt et al. [55] proposed a locality-aware task scheduling for parallel data stream systems.

Muddukrishna et al. [56] proposed a locality-aware task scheduling and runtime system-assisted data distribution algorithm for OpenMP tasks on NUMA systems and multicore processors. Ding et al. [57] proposed a cache hierarchy-aware loop–iterations–to–core mapping strategy by exploiting data reuse and minimizing data dependencies, which results in improved data locality. Lifflander et al. [58] proposed locality-aware optimization at

different phases of fork/join programs with optimal load balance based on Cilk and also provides programmatic support for work-stealing schedules which helps in user guidance on data locality.

Xue et al. [59] proposed a hybrid locality-aware dynamic load balancing and locality-aware loop distribution strategy to multiprocessors and enhanced performance is reported compared to other static/dynamic scheduling techniques. Isard et al. [60] proposed a multipurpose execution engine called Dryad for coarse-grain data-parallel applications for efficient fault resilience, data transportation, and data-aware task scheduling. Maglalang et al. [61] proposed a locality-aware dynamic task graph scheduler with optimal locality and load balance and minimum overhead.

Yoo et al. [62] proposed a locality-aware task scheduling for unstructured parallelism by developing a locality analysis framework. The offline scheduler takes workload profuse information as input and makes scheduling decisions that are optimized with underlying cache hierarchies. Paidel et al. [63] focused on the selection of tasks that are most favorable to migrate across nodes in a distributed environment which is further supported by application-level data locality. Choi et al. [64] proposed locality-aware resource management and workflow scheduling by balancing resource utilization and achieving data locality based on network bandwidth in an HPC cloud environment.

Work scheduling and stealing are the two most commonly used scheduling paradigms for scheduling multithreaded computations to workers in typical task-based parallel systems. Guo Yi [65] in his PhD work proposed a locality-aware work-stealing framework for the efficient exploitation of data locality (affinity) and an adaptive work-stealing scheduling algorithm.

Hindman et al. [66] proposed Mesos, a thin resource-sharing layer with the prime objective being to engineer a scalable and efficient system for sharing resources between heterogeneous frameworks by presenting an abstraction called a resource offer. The resources offered to the framework are decided by Mesos based on organizational policy, while which policies to accept, and tasks to run on them are decided by the framework. Mesos uses delay scheduling and data locality is achieved by taking turns reading data stored on each node. Isard et al. [67] proposed a fair scheduling of concurrent jobs with fine-grain resource sharing for distributed computing clusters called Quincy, which achieved better fairness and improved data locality.

6. Bulk Synchronous Processing (BSP)

The BSP model, which was presented by Valiant [68] and modified by McColl, is a bridging model to design parallel algorithms and mainly consists of processing components, equipped with local memory, a network for communication, and synchronization between components. Communication is facilitated by one-sided put-and-get calls rather than two-way send-and-receive operations. A barrier ensures that all one-sided communication is completed. The oversubscription of processing elements and problem decomposition are exploited by the BSP model for automatically distributed memory management. Logical processes are randomly assigned to processors for optimal load balancing and communication [69]. Google used BSP for graph analytics with Pregel [70] and map-

reduce, while some open-source projects (Apache Hama [71] and Giraph [72]) also extended the use of BSP by employing high-performance parallel programming models on top of Hadoop. There are different programming languages and interfaces based on the BSP model including BSPLib [73], BSPonMPI [74], Bulk Synchronous Parallel ML (BSML), and Multicore-BSP [75][76].

7. Out-of-Core Computing

Out-of-core algorithms (e.g., solvers for large systems of linear equations, as in nuclear physics) are used when the data to be processed are too large to fit in memory and data need to be fetched from storage devices e.g., hard and tape drives or memory attached via a network. As these auxiliary storage devices are slow and acceptable performance is achieved by data reuse in memory and how data are laid out in these storage devices for I/O on larger blocks [77].

The use of virtual memory, which enables programmers to access much larger than available memory without a need to know the actual location of data in the memory hierarchy. Different techniques proposed over the years to efficiently exploit data movement across different memory hierarchies i.e., caching, swapping and demand paging, etc. There are applications where virtual memory systems do not meet the programmer's expectations and do not provide enough virtual memory space. Out-of-core algorithms are used when primary and virtual memory is not enough to hold application data [78]. The principle of locality plays an important role in the performance of an application. Locality in terms of an out-of-core algorithm means that data must be laid out in a storage device to allow blocks of data to exchange between the memory of storage devices and also the reuse of data in memory. In a traditional disk-based approach, the processor remains idle until the data is loaded into memory and the next read is not initiated until the computation is finished. The author of [79] addresses this issue by proposing a two-process approach where disk I/O and computation are performed concurrently. One approach to overcome the limitations of speed at which data can be accessed from storage devices is the use of shared and distributed memories across the cluster, which demands the use of high-speed interconnects (InfiniBand) for the dataset to be loaded before the start of the algorithm in large aggregated distributed/shared memory. The use of high-speed interconnects improves performance but at the expense of a tangible cost of initial setup, maintenance, and energy consumption over time. The trend of the use of non-volatile memory NVM, e.g., low-power flash-based Solid-State Drives (SSDs) to speed up the I/O has increased over the years. These SSDs are used along with traditional storage devices on I/O nodes in a cluster environment, which results in enhanced performance with faster data access/load from these SSDs to I/O nodes [80].

Jung et al. [80] addresses the issues and potential benefits of co-locating the non-volatile memory and compute nodes, by presenting a compute local NVM architecture. They identify the drawbacks of modern file systems and proposed a novel Unified File System (UFS). In addition, there are numerous efforts in the literature focused on the usage of SSDs that include the use of SSDs as caches [81], FlashTier [82], and Mercury [83]. Salue et al. [84] presented an out-of-core task-based middleware for data-intensive computing and [77][85][86] are related to out-of-core algorithms for solving dense and linear equation systems.

8. Parallelism Mapping

The increase in system concurrency introduced a massive challenge for applications and system software to deal with large-scale parallelism. The widely deployed message passing model MPI may not be a suitable choice to deal with the demands of extreme scale parallelism for exascale systems. Finding a single anomalous process among millions of running processes and threads is not an easy task [87]. The issues related to parallelism are diverse and are very much program-dependent. Parallelism mapping is very much platform/hardware-dependent and optimal mapping decisions depend on many factors like scalability (how much potential parallelism should be exploited), the number of processors in use, scheduling policies, the relative cost of communication and computation, etc. There have been various efforts to address this issue. In multi-cluster environments, the bandwidth among nodes inside a single cluster is normally much higher than the bandwidth between two clusters; similarly, within node cores sharing, the cache can communicate much faster. Entities exchanging or sharing lots of data could be placed on hardware processing units physically close to each other. By doing so, the communication costs are reduced, thus decreasing the application's overall execution time and, as a consequence, its energy consumption [17]. Along with communication, application performance is also dependent on load imbalance, communication patterns, and memory usage.

Mapping threads/processes to cores is dependent on many factors like operating system, implementation (different implementations of MPI e.g., OpenMPI, MPICH, IntelMPI), and runtime system, i.e., Message Passing Interface (MPI), Partitioned Global Address Space (PGAS). The work related to the efficient mapping of processes to reduce the communication cost is based on finding the communication topology (communication pattern/graph of the application e.g., number and size of messages between processes, etc.) and network topology graph (e.g., the latency and bandwidth between different processor cores, inter- and intra-node communication cost, etc.) and then the appropriate selection of optimal cores where the processes should be mapped. One can achieve the task of the optimal mapping of processes to cores by running an application with monitoring tools to understand the communication pattern. Trace libraries can provide communication details, e.g., MPI Trace [88]. There is a lack of standardization for thread/process mapping at start-up but this can be implemented at the MPI-execution level. Unfortunately, current parallel programming paradigms seem unable to address the data locality issue to improve parallel-application scalability. There is a need for some evolutionary and revolutionary changes in parallel programing models to address these problems.

There is a considerable amount of work in the literature related to process placement for MPI applications based on correlating the communication and network topology by algorithmic means using graph theory and implementation based on a scheduler, compiler, or exploiting the MPI runtime environment.

8.1. Message Passing Interface Support for Process Mapping

The HPC application community has begun experimenting with the manual placement of individual processes in a parallel job, commonly referred to as "process placement" or "process affinity". MPI implementation provides different mapping patterns like by-node (a.k.a., scatter, cyclic) and by-slot (a.k.a., bunch, pack, block). The different

MPI implementations also provide numerous mpirun command line options and bind with different runtime configuration parameters to enhance the process mapping and binding. Assigning more than one process to a single processor is considered oversubscribing in most HPC environments and is generally discouraged as MPI/HPC applications are CPU-intensive; sharing multiple processes on a single processor causes starvation and performance degradation [89].

Given a parallel application, it is essential to efficiently map the MPI processes to the processors on the nodes. The communication pattern needs to be understood by running the application with profiling tools. Trace libraries can provide the communication details, e.g., MPI Trace, and process mapping can be done manually. Communication-assisted communication analysis can also be performed to map MPI processes to processors. The ultimate goal is to reduce communication by mapping processes with frequent communication on a single node. There could be a number of taxonomies to classify the mapping methodologies, like target architecture-based, optimization criteria-based, workload-based (Static or dynamic), etc. Static mapping methodologies are best suited for static workload scenarios where a predefined set of applications with known computation and communication behavior and a static platform are considered. As optimization is performed at design-time, the methodologies can use more thorough system information to make decisions for both homogeneous and heterogeneous architectures [90].

8.2. Algorithmic Approaches for Process Mapping

The speed of communication among cores in a multicore processor chip (intra-chip) varies with core selection since some cores in a processor chip share certain levels of cache and others do not. Consequently, intra-chip inter-process communication can be faster if the processes are running in cores with shared caches. The situation gets even worse when among cores on distinct processor chips in a cluster. Rodrigues et al. [91] used a graph mapping technique for the mapping process to cores by considering intra-chip, intra-node, and inter-node communication costs to improve the performance of applications with a stable communication pattern. The approach was tested by comparing the execution times of a real-world weather forecast model using default mapping and the proposed solution and obtained an improvement of up to 9.16%.

Rashti et al. [92] merged the node physical topology with network architecture and used graph embedding tools with an MPI library to override the trivial implementation of the topology functions and effectively reorder the initial process mapping. Hestness et al. [93] presented a detailed analysis of memory system behavior and effects for applications mapped to both CPU and GPU cores. Understanding the memory system behavior is very important as multiple cores are integrated on the same die that shares numerous resources.

The communication topology of an application and its underlying architecture affect the performance of point-to-point communication and MPI provides different primitives to gather such information such as MPI Cart create and MPI Graph create. An application communication graph can be created by calculating the communication cost between processes using message count and message volume. HU Chen et al. [94] proposed a profile-guided approach to finding the optimized mapping automatically to minimize the cost of point-to-point communications for arbitrary message-passing applications called MPIPP (MPI process placement toolset). This tool acquires the

communication profile of the MPI application and the network topology of the target clusters. They also proposed an algorithm for optimized mapping and enhanced performance is reported by comparing their solution with existing graph portioning algorithms.

Zhang et al. [95] proposed an approach for optimized process placement to handle collective communication by transforming them into a series of point-to-point communication operations. They decomposed a collective communication into point-to-point and then generated the communication pattern of the whole application. They used a graph-partitioning algorithm for optimized process mapping. Pilla et al. [96] proposed a topology-aware load balancing algorithm for multicore systems by modeling distances and communication among hardware components in terms of latency and bandwidth by exploiting the properties of the current parallel systems, i.e., network interconnection, multi-levels of cache, etc. The introduction of multicore processors introduces numerous challenges including competition between the various physical cores for shared resources. Having more cores in a single node causes multiple requests for the network interface, resulting in performance degradation [97]. This demands the distribution of parallel processes in available computing nodes such that requests arriving at each network interface be decreased. The queuing time of messages at interface queues will be decreased as well, resulting in enhanced performance. Zarrinchain et al. [97] addressed this issue by proposing a solution for mapping parallel processes to multicore clusters to reduce network interface contention by determining the length of the messages among processes and an appropriate value for the threshold (number of processes in each compute node) using the number of adjacent processes of each process and the number of available free cores in the computing nodes.

Guillaume et al. [98] proposed an efficient process mapping of an MPI application to better take advantage of a multicore environment without any modification of MPI implementation and improved performance is reported solely on the basis of relevant process placement. They extract an embedding of the application's graph from the target machine's graph and used scotch software to solve NP graph problems. Scotch applies graph theory, with a divide-and-conquer approach, to scientific computing problems such as graph and mesh partitioning, static mapping, and process ordering. The information is then used to create a mapping between MPI process ranks and each node's core numbers. Finally, an application-specific command line is generated.

Other work related to mapping includes architecture-specific mapping [99][100][101] for Blue Gene systems, [92][102][103] targeting multicore networks, [104] targets hybrid MPI/OpenMP mapping, [105] proposes a mapping library, and [106][107] advance programming standards that support virtual topology mapping.

8.3. Machine Learning-Based Parallelism Mapping

Programming with target architecture in mind and mapping parallelism to processors/cores to avoid/minimize communication are two alternative ways of optimizing application performance. Selecting the correct mapping scheme has a significant impact on performance and these mapping schemes are very much architecture-dependent. So, there is always a need for an automatic and portable solution for assigning tasks to a target architecture to achieve scalable parallelism.

Castro et al. [108] proposed a machine learning-based approach to do efficient thread mapping in transactional memory (TM) applications. Software TM libraries usually implement different mechanisms to detect and solve conflicts. As a consequence, it becomes much more complex to determine a suitable thread-mapping strategy for an application since it can behave differently according to conflict detection and resolution mechanisms. Grewe et al. [109] proposed a portable partitioning scheme for OpenCL programs on heterogeneous CPU and GPU architectures by extracting code features statically and using ML algorithms for predicting the best task partitioning. Tournavitis et al. [110] proposed profile-driven parallel detection and ML-based mapping to overcome the limitations of static analysis and traditional parallelizing compilers by using profiling data to extract control and data dependencies and then used an ML-based trained predictor to select the best scheduling policy offered by OpenMP i.e., CYCLIC, GUIDED, STATIC, and DYNAMIC.

Wang et al. [111] proposed a compiler-based automatic and portable approach for selecting the best thread scheduling policy based on an ML model learned offline, to map parallel programs to multicores. ML-based predictors use profiling information to characterize the code, data, and runtime features of a given program. The feature extractor needs several profiling runs for a program to extract these features. They used an Artificial Neural Network ANN and Support Vector Machine SVM to build two different learning models to predict program scalability and classify scheduling policies. The models were trained using a set of training data that consisted of pre-parallelized programs with selected features and desired mapping decisions. Long et al. [112] used an ML-based approach for cost-aware parallel workload allocation by using static program features. More specifically they used ML to determine the thread number allocated for a parallel java loop on the run time and don’t tackle portability. Adaptive multi-versioning for OpenMP parallelization via machine learning integrated in the compiler, to achieve parallelism. Pinel et al. [113] proposed an ML-based automatic parallelization of a scheduling heuristic, by defining a generic parallel pattern-matching engine that learns the algorithm to parallelize.

Emani et al. [114] proposed a predictive modeling approach that dynamically considers a number of thread selection policies and chooses the one it believes will perform best at every parallel loop and called this approach a mixture of experts. Their approach predicts the optimal number of threads for a program and the run-time environment. Emani et al. [115] proposed an efficient parallel mapping based on online change detection by combining an offline model with online adaption to find the optimal number of threads for an OpenMP program. Luk et al. [116] proposed a fully automatic mapping technique to map computations to processing elements on heterogeneous multiprocessors. They measured the parallelization speedups on matrix multiplication with a heterogeneous machine and implemented it in an experimental system called Qilin and report a performance close to manual mapping with an adaptability feature for different problem sizes and hardware configurations.

Dominguez et al. [117] extended their previous work by using Servet to map applications on multicore systems and analyze the performance of different parallel programming models i.e., message-passing, shared memory, and Portioned Global Address Space (PGAS). The featured extracted by Servet can be used to optimize the performance by choosing a more appropriate mapping policy without source code modification.

9. In Situ Data Analysis

The increasing data size, limited storage and bandwidth, efficient use of compute resources, difficulties in examining output data, and storing and retrieving output data for post data analysis are considered impractical for an exascale environment. The term data movement for in situ data analysis is mostly used to describe data movement back and forth to persistent storage and retrieving data for post data analysis. In situ analysis translates to saving in execution times, power, and storage cost, and avoiding either completely, or to a very large extent, massive data movement of simulations output to persistent storage.

In situ data analysis is performed on the data in the transition phase before they are written back into the parallel file system. Tiwari et al. [118] exploit the compute power in SSDs for in situ data analysis and called this approach Active Flash. This approach provides energy-efficient data analysis as computation near storage devices reduces the data movement cost; in addition, SSDs are equipped with low-power, multicore ARM-based controllers. In situ analysis has become one of the core aspects of data interpretation in large-scale scientific simulations. However, for data that already reside in backend storage systems, efficient data analysis is still a core issue.

Zheng et al. [119] proposed an in situ middleware system to facilitate the underlying scheduling tasks e.g., cycle stealing. They created an agile run-time and called it GoldRush, fine-grained scheduling to steal idle resources by ensuring minimal interruption to the simulations and in situ data analysis. The system makes use of the idle wasted resources of compute nodes to be efficiently used for in situ analysis. The experiment results showed enhanced performance, low data-movement cost, efficient resource utilization, and minimum interference with simulation. Sewell et al. [120] proposed a framework that uses both in situ and co-scheduling approaches for large-scale output data. They compare the performance by analyzing different setups to perform data analysis, i.e., in situ, co-scheduling, and a combination of both.

9.1. In Situ Compression

Scientific data are mostly regarded as effectively incompressible due to their inherently random nature and decompression also imposes extra overhead. Sriram et al. [121] addresses this problem of compression by exploiting temporal patterns in scientific data to compress data with minimal overhead on simulation runtime.

Zou et al. [122] worked on data compression for the removal of redundant data reduction, by using general compression techniques and proposed a use-specific method that allows users to remove redundant or non-critical data by using simple data queries. This method allows users to optimize output data and explicitly identify the data that need to be retained. General-purpose lossy compression techniques do not provide this level of flexibility.

9.2. Use of Indexing for In Situ Data Analysis

As the computation power increases at a brisk speed compared to storing data to disks and reading data back from these disks, J Kim et al. [123] used indexing and in situ processing to address these challenges. Indexing is a powerful and effective way of addressing data access issues, while the implementation of an indexing technology is embedded in DBMS, which lacks the ability to manage most scientific datasets. They used in situ data processing to create indexing in parallel, thus reducing the resources utilized, to store data back and forth from

disks. The usage of indexes improved the data access time and in situ data processing reduced the index creation time.

According to Sriram et al. [124], current state-of-the-art indexes require computation and memory-intensive processing, thus making indexing impractical for in situ processing. They propose DIRAQ, a parallel in situ, query-driven visualization, and analysis during simulation time that transforms the simulation output to a query-accessible form. This technique has a minimum overhead on simulation runtime and speeds up query-response time.

Yu Su et al. [125] proposed in situ Bitmap generation and performing data analysis based on these Bitmaps. Bitmap indices can be used as a summary structure for offline analysis tasks. Their work basically focused on in situ analysis of selected bitmaps, thus reducing the amount of simulation output data to be stored on the disk and reducing the memory requirements for in situ analysis. HPC systems with in situ data analysis analyze temporary datasets as they are generated. Permanent datasets are stored in the backend persistent storage systems; their efficient analysis is still a challenging task.

9.3. In Situ Visualization

As scientific simulations produce a huge volume of raw data, saving a vast amount of raw data for offline analysis is a complex task and not a suitable method for current petascale and future exascale systems. Karimabadi et al. [126] addresses this I/O issue through in situ visualization strategies. The main idea is to extract important features from raw data parallel with simulation and thus reducing the amount of raw data stored on the disk. Their work focused on the overhead associated with computation needs for in situ visualizations in parallel with the simulation run.

Yu et al. [127] investigated in situ visualization for turbulent-combustion simulations and explored in situ data processing and visualization strategies in an extremely parallel environment. Their results showed that in situ visualization enhanced performance and can be used for accelerating high-performance supercomputing and scientific discovery. Zou et al. [128] proposed an online data query system (FlexQuery) using inline performance monitoring and minimized data movement with low latency data-query execution. They demonstrated the dynamic deployment of queries by the proposed query system for low-latency remote data visualization.

Woodring et al. [129] addresses the issue of reducing memory footprints by sharing data between visualization libraries and simulations by using a zero-copy data structure. They optimized the traditional way of coupling different mesh-based codes for in situ data analysis where data needs to be explicitly copied from one implementation to another with the necessary translation. This results in redundant data, which ultimately increases memory footprints. They proposed an alternative way of sharing data between simulations through optimized dynamic on-demand data translation, with reduced memory footprints and memory per core. Nouanesengsy et al. [130] proposed a generalized analysis framework for automatically evaluating the relative importance of data in order to reduce data products, thus ensuring enough resources to process reduce datasets. The proposed framework prioritizes large-scale data by using user-defined prioritization measurements.

9.4. In Situ Feature Selection

Landge et al. [131] proposed in situ feature extraction techniques that allow state-of-the-art feature-based analysis to be performed in situ with minimal overhead on simulation runtime.

Zhang et al. [132] proposed a framework for in situ feature-based object tracking on distributed scientific datasets with decentralized online clustering DOC and a cluster tracking algorithm. They run in parallel with the simulation process and obtain data from on-chip-shared memory directly from the simulation. Their results showed that the proposed framework can be efficiently utilized for in situ data analysis of large-scale simulations.

References

1. Alotaibi, H.; Alsolami, F.; Abozinadah, E.; Mehmood, R. TAWSEEM: A Deep-Learning-Based Tool for Estimating the Number of Unknown Contributors in DNA Profiling. Electronics 2022, 11, 548.
2. Althumairi, A. 'Governmental Communication' launches the visual identity of the 'We are All Responsible' initiative to confront 'COVID 19'. Int. J. Environ. Res. Public Health 2021, 18, 282.
3. Muhammed, T.; Mehmood, R.; Albeshri, A.; Alsolami, F. HPC-Smart Infrastructures: A Review and Outlook on Performance Analysis Methods and Tools; Springer: Cham, Switzerland, 2020; pp. 427–451.
4. Aqib, M.; Mehmood, R.; Alzahrani, A.; Katib, I.; Albeshri, A.; Altowaijri, S.M. Smarter Traffic Prediction Using Big Data, In-Memory Computing, Deep Learning and GPUs. Sensors 2019, 19, 2206.
5. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. UbeHealth: A Personalized Ubiquitous Cloud and Edge-Enabled Networked Healthcare System for Smart Cities. IEEE Access 2018, 6, 32258–32285.
6. AlAhmadi, S.; Muhammed, T.; Mehmood, R.; Albeshri, A. Performance Characteristics for Sparse Matrix-Vector Multi-Plication on GPUs; Springer: Cham, Switzerland, 2020; pp. 409–426.
7. Mohammed, T.; Albeshri, A.; Katib, I.; Mehmood, R. DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems. J. Supercomput. 2020, 77, 6313–6355.
8. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. SURAA: A Novel Method and Tool for Loadbalanced and Coalesced SpMV Computations on GPUs. Appl. Sci. 2019, 9, 947.
9. Alahmadi, S.; Mohammed, T.; Albeshri, A.; Katib, I.; Mehmood, R. Performance Analysis of Sparse Matrix-Vector Multiplication (SpMV) on Graphics Processing Units (GPUs). Electronics 2020, 9, 1675.

10. Alyahya, H.; Mehmood, R.; Katib, I. Parallel Iterative Solution of Large Sparse Linear Equation Systems on the Intel MIC Architecture; Springer: Cham, Switzerland, 2019; pp. 377–407.
11. Mehmood, R.; Crowcroft, J. Parallel Iterative Solution Method for Large Sparse Linear Equation Systems. Technical Report Number UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, Cambridge, UK, 2005. 2005. Available online: https://www.cl.cam.ac.uk/research/srg/netos/papers/MC05.pdf (accessed on 26 February 2016).
12. Nicole Casal Moore. Towards a Breakthrough in Software for Advanced Computing. Available online: https://cse.engin.umich.edu/stories/a-breakthrough-for-large-scale-computing (accessed on 24 August 2022).
13. Guest, M. The Scientific Case for High Performance Computing in Europe 2012–2020. Tech. Rep. 2012.
14. Matsuoka, S.; Sato, H.; Tatebe, O.; Koibuchi, M.; Fujiwara, I.; Suzuki, S.; Kakuta, M.; Ishida, T.; Akiyama, Y.; Suzumura, T.; et al. Extreme Big Data (EBD): Next Generation Big Data Infrastructure Technologies Towards Yottabyte/Year. Supercomput. Front. Innov. 2014, 1, 89–107.
15. ETP4HPC, A. EuropEan Technology platform for High Performance Computing. In ETp4hpc ETP4HPC; Barcelona, Spain, 2013; Available online: https://www.etp4hpc.eu/pujades/files/ETP4HPC_book_singlePage.pdf (accessed on 1 November 2022).
16. Unat, D.; Dubey, A.; Hoefler, T.; Shalf, J.; Abraham, M.; Bianco, M.; Chamberlain, B.L.; Cledat, R.; Edwards, H.C.; Finkel, H.; et al. Trends in Data Locality Abstractions for HPC Systems. IEEE Trans. Parallel Distrib. Syst. 2017, 28, 3007–3020.
17. Hoefler, T.; Jeannot, E.; Mercier, G. An Overview of Topology Mapping Algorithms and Techniques in High-Performance Computing; Wiley-IEEE Press: Hoboken, NJ, USA, 2014; pp. 73–94.
18. Majo, Z.; Gross, T.R. A library for portable and composable data locality optimizations for NUMA systems. In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming—PPoPP 2015, San Francisco, CA, USA, 7–11 February 2015; volume 50; pp. 227–238.
19. Lezos, C.; Latifis, I.; Dimitroulakos, G.; Masselos, K. Compiler-Directed Data Locality Optimization in MATLAB. In Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems—SCOPES ’16, New York, NY, USA, 23–25 May 2016; pp. 6–9.
20. Ragan-Kelley, J.; Barnes, C.; Adams, A.; Paris, S.; Durand, F.; Amarasinghe, S. Halide. ACM SIGPLAN Not. 2013, 48, 519–530.
21. Chamberlain, B. Parallel Processing Languages: Cray’s Chapel Programming. Available online: https://www.cray.com/blog/chapel-productive-parallel-programming/ (accessed on 17 September 2022).

22. Charles, P.; Grothoff, C.; Saraswat, V.; Donawa, C.; Kielstra, A.; Ebcioglu, K.; von Praun, C.; Sarkar, V. X10. In Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications—OOPSLA ’05, New York, NY, USA, 16–20 October 2005; Volume 40, pp. 519–538.
23. Huang, L.; Jin, H.; Yi, L.; Chapman, B. Enabling locality-aware computations in OpenMP. Sci. Program. 2010, 18, 169–181.
24. Zhang, H.; Chen, G.; Ooi, B.C.; Tan, K.-L.; Zhang, M. In-Memory Big Data Management and Processing: A Survey. IEEE Trans. Knowl. Data Eng. 2015, 27, 1920–1948.
25. Gupta, S.; Zhou, H. Spatial Locality-Aware Cache Partitioning for Effective Cache Sharing. In Proceedings of the 2015 44th International Conference on Parallel Processing, Beijing, China, 1–4 September 2015; pp. 150–159.
26. González, A.; Aliagas, C.; Valero, M. A data cache with multiple caching strategies tuned to different types of locality. In Proceedings of the 9th International Conference on Supercomputing —ICS ’95, New York, NY, USA, 3–7 July 1995.
27. Seshadri, V.; Mutlu, O.; Kozuch, M.A.; Mowry, T.C. The evicted-address filter. In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques—PACT ’12, Minneapolis, MN, USA, 19–23 September 2012; p. 355.
28. Rivers, J.; Davidson, E. Reducing conflicts in direct-mapped caches with a temporality-based design. In Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing, Ithaca, NY, USA, 12 August 2002; Volume 1, pp. 154–163.
29. Johnson, T.L.; Hwu, W.-M.W. Run-time adaptive cache hierarchy management via reference analysis. In Proceedings of the 24th Annual International Symposium on Computer Architecture—ISCA ’97, Boulder, CO, USA, 2–4 June 1997; Volume 25, pp. 315–326.
30. Jiang, X.; Madan, N.; Zhao, L.; Upton, M.; Iyer, R.; Makineni, S.; Newell, D.; Solihin, Y.; Balasubramonian, R. CHOP: Adaptive filter-based DRAM caching for CMP server platforms. In Proceedings of the HPCA—16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore, India, 9–14 January 2010; pp. 1–12.
31. Muchnick, S.S. Advanced Compiler Design and Implementation; Morgan Kaufmann Publishers: Burlington, MA, USA, 1997.
32. Allen, R.; Kennedy, K. Optimizing Compilers for Modern Architectures: A Dependence-Based Approach; Morgan Kaufmann Pub-lishers: Burlington, MA, USA, 2001.
33. Wolfe, M. Loops skewing: The wavefront method revisited. Int. J. Parallel Program. 1986, 15, 279–293.

34. Kowarschik, M.; Weiß, C. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms; Springer: Berlin/Heidelberg, Germany, 2003; pp. 213–232.
35. Xue, J.; Ling, J. Loop Tiling for Parallelism; Kluwer Academic: New York, NY, USA, 2000.
36. Bao, B.; Ding, C. Defensive loop tiling for shared cache. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Shenzhen, China, 23–27 February 2013; pp. 1–11.
37. Wolf, M.E.; Lam, M.S. A data locality optimizing algorithm. In Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation—PLDI ’91, Toronto, Canada, 26–28 June 1991.
38. Irigoin, F.; Triolet, R. Supernode partitioning. In Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages—POPL ’88, Boston, MA, USA, 15–21 January 1988; pp. 319–329.
39. Zhou, X.; Giacalone, J.-P.; Garzarán, M.J.; Kuhn, R.H.; Ni, Y.; Padua, D. Hierarchical overlapped tiling. In Proceedings of the Tenth International Symposium on Code Generation and Optimization —CHO ’12, Montreal, Canada, 27 February–3 March 2012; pp. 207–218.
40. Liu, L.; Chen, L.; Wu, C.; Feng, X.-B. Global Tiling for Communication Minimal Parallelization on Distributed Memory Systems. In Euro-Par 2008—Parallel Processing; Springer: Berlin/Heidelberg, Germany, 2008; pp. 382–391.
41. Hogstedt, K.; Carter, L.; Ferrante, J. On the parallel execution time of tiled loops. IEEE Trans. Parallel Distrib. Syst. 2003, 14, 307–321.
42. Yi, Q. Automated programmable control and parameterization of compiler optimizations. In Proceedings of the International Symposium on Code Generation and Optimization (CGO 2011), Chamonix, France, 2–6 April 2011; pp. 97–106.
43. Hall, M.; Chame, J.; Chen, C.; Shin, J.; Rudy, G.; Khan, M.M. Loop Transformation Recipes for Code Generation and Auto-Tuning; Springer: Berlin/Heidelberg, Germany, 2010; pp. 50–64.
44. Tavarageri, S.; Pouchet, L.-N.; Ramanujam, J.; Rountev, A.; Sadayappan, P. Dynamic selection of tile sizes. In Proceedings of the 2011 18th International Conference on High Performance Computing, Bengaluru, India, 18–21 December 2011; pp. 1–10.
45. Kennedy, K.; McKinley, K.S. Optimizing for parallelism and data locality. In Proceedings of the 25th Anniversary International Conference on Supercomputing Anniversary Volume, New York, NY, USA, 2–6 June 2014; pp. 151–162.
46. Mittal, S. A Survey Of Cache Bypassing Techniques. J. Low Power Electron. Appl. 2016, 6, 5.
47. Raicu, I.; Zhao, Y.; Dumitrescu, C.; Foster, I.; Wilde, M. Falkon. In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing—SC ’07, New York, NY, USA, 10–16 November

2007; p. 43.

48. Yoo, A.B.; Jette, M.A.; Grondona, M. SLURM: Simple Linux Utility for Resource Management; Springer: Berlin/Heidelberg, Germany, 2003; pp. 44–60.
49. Gentzsch, W. Sun Grid Engine: Towards creating a compute power grid. In Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, QLD, Australia, 15–18 May 2002.
50. Thain, D.; Tannenbaum, T.; Livny, M. Distributed computing in practice: The Condor experience: Research Articles. Concurr. Comput. Pract. Exp. 2005, 17, 323–356.
51. Ousterhout, K.; Wendell, P.; Zaharia, M.; Stoica, I. Sparrow. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles—SOSP ’13, New York, NY, USA, 3–6 November 2013.
52. Olivier, S.; Porterfield, A.K.; Wheeler, K.B.; Spiegel, M.; Prins, J.F. OpenMP task scheduling strategies for multicore NUMA systems. Int. J. High Perform. Comput. Appl. 2012, 26, 110–124.
53. Frigo, M.; Leiserson, C.E.; Randall, K.H. The implementation of the Cilk-5 multithreaded language. ACM SIGPLAN Not. 1998, 33, 212–223.
54. Wang, K.; Zhou, X.; Li, T.; Zhao, D.; Lang, M.; Raicu, I. Optimizing load balancing and data-locality with data-aware scheduling. In Proceedings of the 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 27–30 October 2014; pp. 119–128.
55. Falt, Z.; Kruliš, M.; Bednárek, D.; Yaghob, J.; Zavoral, F. Locality Aware Task Scheduling in Parallel Data Stream Processing; Springer: Cham, Switzerland, 2015; pp. 331–342.
56. Muddukrishna, A.; Jonsson, P.A.; Brorsson, M. Locality-Aware Task Scheduling and Data Distribution for OpenMP Programs on NUMA Systems and Manycore Processors. Sci. Program. 2015, 2015, 1–16.
57. Ding, W.; Zhang, Y.; Kandemir, M.; Srinivas, J.; Yedlapalli, P. Locality-aware mapping and scheduling for multicores. In Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Shenzhen, China, 23–27 February 2013; pp. 1–12.
58. Lifflander, J.; Krishnamoorthy, S.; Kale, L.V. Optimizing Data Locality for Fork/Join Programs Using Constrained Work Stealing. In Proceedings of the SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 16–21 November 2014; pp. 857–868.
59. Xue, L.; Kandemir, M.; Chen, G.; Li, F.; Ozturk, O.; Ramanarayanan, R.; Vaidyanathan, B. Locality-Aware Distributed Loop Scheduling for Chip Multiprocessors. In Proceedings of the 20th International Conference on VLSI Design Held Jointly with 6th International Conference on Embedded Systems (VLSID’07), Bangalore, India, 6–10 January 2007; pp. 251–258.

60. Isard, M.; Budiu, M.; Yu, Y.; Birrell, A.; Fetterly, D. Dryad. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007—EuroSys ’07, New York, NY, USA, 21–23 March 2007; Volume 41, pp. 59–72.
61. Maglalang, J.; Krishnamoorthy, S.; Agrawal, K. Locality-Aware Dynamic Task Graph Scheduling. In Proceedings of the 2017 46th International Conference on Parallel Processing (ICPP), Bristol, UK, 14–17 August 2017; pp. 70–80.
62. Yoo, R.M.; Hughes, C.J.; Kim, C.; Chen, Y.-K.; Kozyrakis, C. Locality-Aware Task Management for Unstructured Par-allelism: A Quantitative Limit Study. In Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures, New York, NY, USA, 23–25 July 2013.
63. Paudel, J.; Tardieu, O.; Amaral, J.N. On the Merits of Distributed Work-Stealing on Selective Locality-Aware Tasks. In Proceedings of the 2013 42nd International Conference on Parallel Processing, Lyon, France, 1–4 October 2013; pp. 100–109.
64. Choi, J.; Adufu, T.; Kim, Y. Data-Locality Aware Scientific Workflow Scheduling Methods in HPC Cloud Environments. Int. J. Parallel Program. 2016, 45, 1128–1141.
65. Guo, Y. A Scalable Locality-Aware Adaptive Work-StealingScheduler for Multi-Core Task Parallelism. Ph.D. Thesis, Rice University, Houston, TX, USA, 2011.
66. Hindman, B.; Konwinski, A.; Zaharia, M.; Ghodsi, A.; Joseph, A.D.; Katz, R.; Shenker, S.; Stoica, I. Mesos: A platform for fine-grained resource sharing in the data center. In Proceedings of the 8th USENIX conference on Networked systems design and implementation. USENIX Association, Boston, MA, USA, March 30–April 1 2011; pp. 295–308.
67. Isard, M.; Prabhakaran, V.; Currey, J.; Wieder, U.; Talwar, K.; Goldberg, A. Quincy. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles—SOSP ’09, Big Sky, MT, USA, 11–14 October 2009.
68. Valiant, L.G. A bridging model for parallel computation. Commun. ACM 1990, 33, 103–111.
69. Cheatham, T.; Fahmyy, A.; Stefanescu, D.C.; Valiant, L.G. Bulk Synchronous Parallel Computing-A Paradigm for transportable Software. Harv. Comput. Sci. Group Tech. Rep. 1995.
70. Malewicz, G.; Austern, M.H.; Bik, A.J.; Dehnert, J.C.; Horn, I.; Leiser, N.; Czajkowski, G. Pregel. In Proceedings of the 2010 International Conference on Management of Data—SIGMOD ’10, New York, NY, USA, 6–10 June 2010; pp. 135–146.
71. Apache Hama Big Data and High-Performance Computing. Available online: https://hama.apache.org/ (accessed on 22 January 2018).
72. Giraph-Welcome To Apache Giraph. Available online: https://giraph.apache.org/ (accessed on 20 October 2022).

73. Hill, J.M.; McColl, B.; Stefanescu, D.C.; Goudreau, M.W.; Lang, K.; Rao, S.B.; Suel, T.; Tsantilas, T.; Bisseling, R.H. BSPlib: The BSP programming library. Parallel Comput. 1998, 24, 1947–1980.
74. BSPonMPI. Available online: https://bsponmpi.sourceforge.net/ (accessed on 20 January 2022).
75. Yzelman, A.N.; Bisseling, R.H.; Roose, D.; Meerbergen, K. MulticoreBSP for C: A High-Performance Library for Shared-Memory Parallel Programming. Int. J. Parallel Program. 2013, 42, 619–642.
76. Yzelman, A.; Bisseling, R.H. An object-oriented bulk synchronous parallel library for multicore programming. Concurr. Comput. Pr. Exp. 2011, 24, 533–553.
77. Abello, J.M.; Vitter, J.S. External memory algorithms: DIMACS Workshop External Memory and Visualization, May 20–22, 1998; American Mathematical Society: Providence, RI, USA, 1999.
78. Kwiatkowska, M.; Mehmood, R. Out-of-Core Solution of Large Linear Systems of Equations Arising from Stochastic Modelling; Springer: Berlin/Heidelberg, Germany, 2002; pp. 135–151.
79. Mehmood, R. Disk-Based Techniques for Efficient Solution of Large Markov Chains. PhD Thesis, School of Computer Science, University of Birmingham,, Birmingham, UK, 2004.
80. Jung, M.; Wilson, E.H.; Choi, W.; Shalf, J.; Aktulga, H.M.; Yang, C.; Saule, E.; Catalyurek, U.V.; Kandemir, M. Exploring the future of out-of-core computing with compute-local non-volatile memory. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on—SC ’13, Denver, CO, USA, 17–22 November 2013, 17–22 November 2013; pp. 1–11.
81. Koller, R.; Marmol, L.; Rangaswami, R.; Sundararaman, S.; Talagala, N.; Zhao, M. Write policies for host-side flash caches. In Proceedings of the 11th USENIX Conference on File and Storage Technologies. USENIX Association, San Jose, CA, USA, 12–15 February 2013; pp. 45–58.
82. Saxena, M.; Swift, M.M.; Zhang, Y. FlashTier. In Proceedings of the 7th ACM European Conference on Computer Systems—EuroSys ’12, New York, NY, USA, 10–13 April 2012; p. 267.
83. Byan, S.; Lentini, J.; Madan, A.; Pabon, L.; Condict, M.; Kimmel, J.; Kleiman, S.; Small, C.; Storer, M. Mercury: Host-side flash caching for the data center. In Proceedings of the 012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), Monterey, CA, USA, 19–20 April 2012; pp. 1–12.
84. Saule, E.; Aktulga, H.M.; Yang, C.; Ng, E.G.; Çatalyürek, Ü.V. An Out-of-Core Task-based Middleware for Da-ta-Intensive Scientific Computing. In Handbook on Data Centers; Springer: New York, NY, USA, 2015; pp. 647–667.
85. Rothberg, E.; Schreiber, R. Efficient Methods for Out-of-Core Sparse Cholesky Factorization. SIAM J. Sci. Comput. 1999, 21, 129–144.

86. Mandhapati, P.; Khaitan, S. High Performance Computing Using out-of-core Sparse Direct Solvers. World Acad. Sci. Eng. Technol. 2009, 3, 377–383.
87. Geist, A.; Lucas, R. Whitepaper on the Major Computer Science Challenges at Exascale. 2009. Available online: https://exascale.org/mediawiki/images/8/87/ExascaleSWChallenges-Geist_Lucas.pdf (accessed on 1 November 2022).
88. Das, B.V.D.; Kathiresan, N.; Ravindran, R. Process Mapping Parallel Computing. US8161127B2, 28 November 2011.
89. Hursey, J.; Squyres, J.M.; Dontje, T. Locality-Aware Parallel Process Mapping for Multi-core HPC Systems. In Proceedings of the 2011 IEEE International Conference on Cluster Computing, Austin, TX USA, 26–30 September 2011; pp. 527–531.
90. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems. In Proceedings of the 50th Annual Design Automation Conference on—DAC ’13, New York, NY, USA, 29 May 2013–7 June 2013; p. 1.
91. Rodrigues, E.R.; Madruga, F.L.; Navaux, P.O.A.; Panetta, J. Multi-core aware process mapping and its impact on communication overhead of parallel applications. In Proceedings of the 2009 IEEE Symposium on Computers and Communications, Sousse, Tunisia, 5–8 July 2009; pp. 811–817.
92. Rashti, M.J.; Green, J.; Balaji, P.; Afsahi, A.; Gropp, W. Multi-core and Network Aware MPI Topology Functions; Springer: Berlin/Heidelberg, Germany, 2011; pp. 50–60.
93. Hestness, J.; Keckler, S.W.; Wood, D.A. A comparative analysis of microarchitecture effects on CPU and GPU memory system behavior. In Proceedings of the 2014 IEEE International Symposium on Workload Characterization (IISWC), Raleigh, NC, USA, 26–28 October 2014; pp. 150–160.
94. Chen, H.; Chen, W.; Huang, J.; Robert, B.; Kuhn, H. MPIPP. In Proceedings of the 20th annual international conference on Supercomputing—ICS ’06, Cairns, QLD, Australia, 28 June–1 July 2006.
95. Zhang, J.; Zhai, J.; Chen, W.; Zheng, W. Process Mapping for MPI Collective Communications; Springer: Berlin/Heidelberg, Germany, 2009; pp. 81–92.
96. Pilla, L.L.; Ribeiro, C.P.; Coucheney, P.; Broquedis, F.; Gaujal, B.; Navaux, P.O.; Méhaut, J.-F. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. Futur. Gener. Comput. Syst. 2014, 30, 191–201.
97. Zarrinchian, G.; Soryani, M.; Analoui, M. A New Process Placement Algorithm in Multi-Core Clusters Aimed to Reducing Network Interface Contention; Springer: Berlin/Heidelberg, Germany, 2012; pp. 1041–1050.

98. Mercier, G.; Clet-Ortega, J. Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments; Springer: Berlin/Heidelberg, Germany, 2009; pp. 104–115.
99. Balaji, P.; Gupta, R.; Vishnu, A.; Beckman, P. Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems. Comput. Sci. Res. Dev. 2011, 26, 247–256.
100. Smith, B.E.; Bode, B. Performance Effects of Node Mappings on the IBM BlueGene/L Machine; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1005–1013.
101. Yu, H.; Chung, I.-H.; Moreira, J. Topology Mapping for Blue Gene/L Supercomputer. In Proceedings of the ACM/IEEE SC 2006 Conference (SC'06), Cairns, QLD, Australia, 28 June–1 July 2006; p. 52.
102. Ito, S.; Goto, K.; Ono, K. Automatically optimized core mapping to subdomains of domain decomposition method on multicore parallel environments. Comput. Fluids 2013, 80, 88–93.
103. Traff, J. Implementing the MPI Process Topology Mechanism. In Proceedings of the ACM/IEEE SC 2002 Conference (SC'02), Baltimore, MD, USA, 16–22 November 2002; p. 28.
104. Dümmler, J.; Rauber, T.; Rünger, G. Mapping Algorithms for Multiprocessor Tasks on Multi-Core Clusters. In Proceedings of the 2008 37th International Conference on Parallel Processing, Washington, DC, USA, 9–11 September 2008; pp. 141–148.
105. Hoefler, T.; Snir, M. Generic topology mapping strategies for large-scale parallel architectures. In Proceedings of the International Conference on Supercomputing—ICS '11, Tucson, AZ, USA, 31 May–4 June 2011; pp. 75–84.
106. Kale, L.V.; Krishnan, S. CHARM++: A Portable Concurrent Object Oriented System Based on C++; Technical Report; University of Illinois at Urbana-Champaign: Champaign, IL, USA, 1993.
107. El-Ghazawi, T. UPC: Distributed Shared Memory Programming; Wiley: Hoboken, NJ, USA, 2005.
108. Castro, M.; Goes, L.F.W.; Ribeiro, C.P.; Cole, M.; Cintra, M.; Mehaut, J.-F. A machine learning-based approach for thread mapping on transactional memory applications. In Proceedings of the 2011 18th International Conference on High Performance Computing, New York, NY, USA, 12–18 November 2011; pp. 1–10.
109. Grewe, D.; O'Boyle, M.F.P. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL; Springer: Berlin/Heidelberg, Germany, 2011; pp. 286–305.
110. Tournavitis, G.; Wang, Z.; Franke, B.; O'Boyle, M.F.P. Towards a holistic approach to auto-parallelization. ACM SIGPLAN Not. 2009, 44, 177–187.
111. Wang, Z.; O'Boyle, M.F. Mapping parallelism to multi-cores. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming—PpoPP '09, Raleigh, NC, USA, 14–18 February 2008; Volume 44, p. 75.

112. Long, S.; Fursin, G.; Franke, B. A Cost-Aware Parallel Workload Allocation Approach Based on Machine Learning Techniques; Springer: Berlin/Heidelberg, Germany, 2007; pp. 506–515.
113. Pinel, F.; Bouvry, P.; Dorronsoro, B.; Khan, S.U. Savant: Automatic parallelization of a scheduling heuristic with machine learning. Nat. Biol. 2013, 52–57.
114. Emani, M.K.; O'Boyle, M. Celebrating diversity: A mixture of experts approach for runtime mapping in dynamic environments. In Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation—PLDI 2015, Portland, OR, USA, 13–17 June 2015.
115. Emani, M.K.; O'Boyle, M. Change Detection Based Parallelism Mapping: Exploiting Offline Models and Online Adaptation; Springer International Publishing: Cham, Switzerland, 2015; pp. 208–223.
116. Luk, C.-K.; Hong, S.; Kim, H. Qilin. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture—Micro-42, New York, NY, USA, 12–16 December 2009; pp. 45–55.
117. González-Domínguez, J.; Taboada, G.L.; Fraguela, B.B.; Martín, M.J.; Touriño, J. Automatic mapping of parallel applications on multicore architectures using the Servet benchmark suite. Comput. Electr. Eng. 2012, 38, 258–269.
118. Tiwari, D.; Vazhkudai, S.S.; Kim, Y.; Ma, X.; Boboila, S.; Desnoyers, P.J. Reducing Data Movement Costs using Ener-gy-Efficient, Active Computation on SSD. In Proceedings of the 2012 Workshop on Power-Aware Computing and Systems, Hollywood, CA, USA, 7 October 2012.
119. Zheng, F.; Yu, H.; Hantas, C.; Wolf, M.; Eisenhauer, G.; Schwan, K.; Abbasi, H.; Klasky, S. GoldRush. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on—SC '13, Denver, CO, USA, 17–22 November 2013.
120. Sewell, C.; Heitmann, K.; Finkel, H.; Zagaris, G.; Parete-Koon, S.T.; Fasel, P.K.; Pope, A.; Frontiere, N.; Lo, L.-T.; Messer, B.; et al. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis—SC '15, Atlanta, GA, USA, 9 November 2015; p. 50.
121. Lakshminarasimhan, S.; Shah, N.; Ethier, S.; Klasky, S.; Latham, R.; Ross, R.; Samatova, N.F. Compressing the Incompressible with ISABELA: In-Situ Reduction of Spatio-temporal Data. Springer: Berlin/Heidelberg, Germany, 2011; pp. 366–379.
122. Zou, H.; Zheng, F.; Wolf, M.; Eisenhauer, G.; Schwan, K.; Abbasi, H.; Liu, Q.; Podhorszki, N.; Klasky, S.; Wolf, M. Quality-Aware Data Management for Large Scale Scientific Applications. In Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, 24–29 June 2012; pp. 816–820.

123. Kim, J.; Abbasi, H.; Chacon, L.; Docan, C.; Klasky, S.; Liu, Q.; Podhorszki, N.; Shoshani, A.; Wu, K. Parallel in situ indexing for data-intensive computing. In Proceedings of the 2011 IEEE Symposium on Large Data Analysis and Visualization, Providence, RI, USA, 23–24 October 2011; pp. 65–72.

124. Lakshminarasimhan, S.; Boyuka, D.A.; Pendse, S.V.; Zou, X.; Jenkins, J.; Vishwanath, V.; Papka, M.E.; Samatova, N.F. Scalable in situ scientific data encoding for analytical query processing. In Proceedings of the 22nd international symposium on High-performance parallel and distributed computing, New York, NY, USA, 17–21 June 2013; pp. 1–12.

125. Su, Y.; Wang, Y.; Agrawal, G. In-Situ Bitmaps Generation and Efficient Data Analysis based on Bitmaps. In 24th International Symposium on High-Performance Parallel and Distributed Computing—HPDC ’15; ACM: New York, NY, USA, 2015; pp. 61–72.

126. Karimabadi, H.; Loring, B.; O’Leary, P.; Majumdar, A.; Tatineni, M.; Geveci, B. In-situ visualization for global hybrid simulations. In Proceedings of the Conference on Extreme Science and Engineering Discovery Environment Gateway to Discovery—XSEDE ’13, Atlanta, GA, USA, 13–18 July 2013; p. 1.

127. Yu, H.; Wang, C.; Grout, R.W.; Chen, J.H.; Ma, K.-L. In Situ Visualization for Large-Scale Combustion Simulations. IEEE Comput. Graph. Appl. 2010, 30, 45–57.

128. Zou, H.; Schwan, K.; Slawinska, M.; Wolf, M.; Eisenhauer, G.; Zheng, F.; Dayal, J.; Logan, J.; Liu, Q.; Klasky, S.; et al. FlexQuery: An online query system for interactive remote visual data exploration at large scale. In Proceedings of the 2013 IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, USA, 23–27 September 2013; pp. 1–8.

129. Woodring, J.; Ahrens, J.; Tautges, T.J.; Peterka, T.; Vishwanath, V.; Geveci, B. On-demand unstructured mesh translation for reducing memory pressure during in situ analysis. In Proceedings of the 8th International Workshop on Ultrascale Visualization—UltraVis ’13, Denver, CO, USA, 17–22 November 2013.

130. Nouanesengsy, B.; Woodring, J.; Patchett, J.; Myers, K.; Ahrens, J. ADR visualization: A generalized framework for ranking large-scale scientific data using Analysis-Driven Refinement. In Proceedings of the 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), Paris, France, 9–10 November 2014; pp. 43–50.

131. Landge, A.G.; Pascucci, V.; Gyulassy, A.; Bennett, J.C.; Kolla, H.; Chen, J.; Bremer, P.-T. In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees. In Proceedings of the SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, USA, 16–21 November 2014; pp. 1020–1031.

132. Zhang, F.; Lasluisa, S.; Jin, T.; Rodero, I.; Bui, H.; Parashar, M. In-situ Feature-Based Objects Tracking for Large-Scale Scientific Simulations. In Proceedings of the 2012 SC Companion: High

Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, 24–29 June 2012; pp. 736–740.