Trusted Performance Analysis on Systems With a Shared Memory

Jiaqi Zhao, Changlong Xue, Jie Tao, Rajiv Ranjan, *Student Member, IEEE*, Joanna Kołodziej, *Member, IEEE*, Lizhe Wang, *Senior Member, IEEE*, and Dan Chen

***Abstract*—With the increasing complexity of both data structures and computer architectures, the performance of applications needs fine tuning in order to achieve the expected runtime execution time. Performance tuning is traditionally based on the analysis of performance data. The analysis results may not be accurate, depending on the quality of the data and the applied analysis approaches. Therefore, application developers may ask: Can we trust the analysis results? This paper introduces our research work in performance optimization of the memory system, with a focus on the cache locality of a shared memory and the memory locality of a distributed shared memory. The quality of the data analysis is guaranteed by using both real performance data acquired at the runtime while the application is running and well-established data analysis algorithms in the field of bioinformatics and data mining. We verified the quality of the proposed approaches by optimizing a set of benchmark applications. The experimental results show a significant performance gain.**

***Index Terms*—Code optimization, data analysis, data locality, distributed shared memory, performance tuning.**

I. Introduction

MODERN computer systems are tending to exascale. As the computational capacity is getting larger, the system architecture is getting more complicated with not only heterogeneous devices but also a hierarchical organization. An example is Nvidia's Echelon exascale computer [22] that envisions processing nodes with a configurable memory hierarchy within a chip and interconnects with different latency and throughput characteristics on four levels: node, module, cabinet, and system. Another example is the Gordon supercomputer [36] at the San Diego Supercomputer Center, which is specifically designed for data-intensive computing. Gordon has a five-level memory hierarchy with a node-local shared memory, a distributed shared memory within a supernode, a distributed memory between supernodes, the Flash memory, and disk arrays.

At the same time, scientific applications are processing larger data sets that were not seen ever before [13], [14], [20], [41]–[43], [46]. The Large Hadron Collider (LHC) [10] in the European Organization for Nuclear Research (CERN), for instance, produces 15 PB of scientific data in a year. These data are processed by LHC five major experiments for different research purposes. The challenge for application developers lies not only in the volume of data but also in the data structures that are becoming more complicated. Today, it is even harder for programmers to produce codes that run efficiently on a target architecture. Performance tuning can be observed as an essential task for application developers or users.

The performance of an application can be influenced by different components of a system [25]. However, the memory performance has been a key issue for the overall performance. This issue becomes more important for today's applications that are processing very large data, the so-called big data. In these data-intensive applications, the time for accessing the data may dominate the entire execution time. Considering the hierarchical memory organization of modern computers, it is possible to significantly improve the applications' performance when the data are reasonably distributed in the memory system, with a result that the processor can acquire the data from the closer memory location.

In this work, we focus on the memory performance of shared-memory systems with a goal of achieving data locality. In a system with a physically shared memory, like the multicore machines, data locality means cache locality, where the research focus is to bring into the caches only the working data that will be used or reused, thus decreasing the cache miss rate. For systems with a distributed shared memory, where the memories are distributed across the computing nodes but logically organized as a shared memory, data locality within the main memory is more important than the cache issues, because accessing a remote memory can take up to orders of magnitude longer than a local memory access. For example, the Gordon supercomputer has an access latency of 100 cycles with local memories, but

Manuscript received July 16, 2014; revised September 11, 2014; accepted October 8, 2014. The work of J. Zhao was supported by the Jilin Province Science and Technology Development Supporting Program 20140101206JC. The work of J. Tao was supported by the German Research Foundation (DFG) through the Priority Program 1648 "Software for Exascale Computing" (SPPEXA). The work of R. Ranjan was supported by the Department of Industry, Australia, under Grant AISRF08140 titled "Innovative Solutions for Deployment of BIgData and Disaster Management Applications on Clouds." *(Corresponding author: Lizhe Wang.)*

J. Zhao is with the School of Basic Science, Changchun University of Technology, Changchun 130012, China.

C. Xue is with the Jilin Provincial High Class Highway Construction Bureau, Changchun 130033, China.

J. Tao is with the Steinbuch Center for Computing, Karlsruhe Institute of Technology, 76021 Karlsruhe, Germany.

R. Ranjan is with the ICT Centre, Commonwealth Scientific and Industrial Research Organisation, Canberra, A.C.T. 2601, Australia.

J. Kołodziej is with the Institute of Computer Science, Cracow University of Technology, 31-155 Cracow, Poland, and also with the Intelligent Information Systems Group, AGH University of Science and Technology, 30-059 Cracow, Poland.

L. Wang is with the Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100094, China, and also with the School of Computer Science, China University of Geosciences, Beijing 100083, China (e-mail: lizhe.wang@gmail.com).

D. Chen is with the School of Computer Science, Wuhan University, Wuhan 430072, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2014.2365234

10 000 cycles with a remote memory. In this case, the applications will not run efficiently or even cannot get speedup on such nonuniform memory access systems if a processing node has to access remote memories frequently. Hence, applications must be optimized with respect to such data locality.

Data locality optimization can be done either automatically at the compiling time/runtime or manually by the programmers directly in the source code. The former frees the programmers from the optimization tasks but often cannot achieve a high performance gain. For example, the compiler-based optimization usually applies heuristic analysis to predict the runtime behavior and optimizes the code based on this kind of inaccurate analysis results. Hence, programmers often choose the latter, a more trusted and straightforward approach, to optimize their codes.

The basis for any kind of performance optimization is the performance data showing the memory access and execution behavior of the applications. The performance data can be acquired in different ways, such as simulation and modeling, code instrumentation, code analysis, or special hardware. The simulation approach can provide the performance data in acute detail; however, the quality of the data depends on the accuracy of the simulation. Code analysis is easy to implement, but the performance data are statically created without involving any runtime information. Code instrumentation collects the performance data by inserting annotations in the source code to explicitly trace interesting events or actions. The hardware approach requires specific devices for monitoring the runtime behavior. Nevertheless, this approach is capable of providing precise performance data and specific runtime information. For the quality of the performance data, we use a specific hardware in this work to inspect the communications between the computing nodes for data locality optimization on distributed-shared-memory systems. For the cache access behavior, we apply code instrumentation to acquire the performance data.

Depending on the applications and their working loads, the performance data can be huge. It is quite hard or even impossible for programmers to understand the runtime behavior of their applications by studying the data lines in a trace file. To make the performance data easier to understand, various visualization tools have been developed to demonstrate the application runtime actions in graphical views. An example is our cache visualizer yet another cache-visualizer for optimization [32], which aims to show the users the access patterns for optimizing the spatial locality of caches.

The visualization tools work well with applications that process a small data set. However, they will not be efficient for large data sets, because a graphical view with a large amount of information cannot highlight the abnormal behavior that needs to be studied and corrected. This problem possibly could be mitigated by reducing the volume of the performance data using approaches like lossy tracing [27], lower sampling rate [9], or tracing of only global events [7]. However, for an exact location of the problem, and, more importantly, for detecting the reasons and, potentially, the solution of the problems, a full trace is needed.

Therefore, we apply well-established analysis algorithms to process the performance data. For the cache optimization, we adopt an algorithm in bioinformatics to analyze memory access traces produced by a code instrumentor, with a result of finding access patterns that guide the users to optimize their applications in terms of the cache locality. By using the same algorithms for detecting repeated patterns from a deoxyribonucleic acid (DNA), a ribonucleic acid (RNA), or protein sequences in bioinformatics, we deliver accurate analysis results to the users.

For distributed-shared-memory systems, we apply conventional data mining algorithms, specifically the classification and regression trees [6] to process the performance data. This brings several advantages. First, the classification algorithm is self-learning, which guarantees the required accuracy of the analysis result and, thereby, the efficiency of the runtime dynamic performance tuning. Second, the same stream of performance data often contains several interesting patterns. For example, a memory access trace holds the access stride, access hot spots, chained references, and so on. By combining different rules of data clustering, all these patterns can be detected. This avoids the necessity of implementing a single algorithm for each pattern. Third, data mining algorithms can both detect the problem and predict the trend. This allows the runtime adaptation system to take actions before the problem occurs. Finally, a complete system optimization is associated with different optimization targets that interact with each other. For example, optimization on speedup can potentially enlarge the power consumption. Data mining algorithms allow a combined analysis of different data streams, thus being capable of delivering appropriate solutions for a balancing tradeoff between different performance metrics. Therefore, we use the data mining approach to analyze the internode communications acquired by a special monitor device.

In summary, this work makes the following contributions to the stand of research on performance analysis and optimization:

1) using mature data analysis algorithms in bioinformatics and data mining to process performance data for efficiency and accuracy;
2) acquiring the performance data at the runtime using hardware for guaranteeing the quality of the data;
3) optimizing a set of benchmark applications based on the analysis results;
4) validating the optimization as well as the analysis results with experiments and discussing the achievement.

The remainder of this paper is organized as follows. Section II introduces the related work in performance analysis and optimization. Section III describes the proposed data analysis approach for detecting both memory access pattern and communication bottlenecks. Section IV shows the experimental results with discussions. This paper concludes in Section V with a brief summary and several future directions.

II. Related Work

Performance analysis and optimization have been hot topics in the field of high-performance computing. With the increasing complexity of applications and computer architectures, applications initially do not run efficiently on the target machines and require performance tuning. Performance tuning is a recurring process, where the application is first executed on the target

architecture, performance data are collected at the runtime and then analyzed, applications are optimized based on the analysis result, and the application is executed again for further tuning.

The prerequisite for any performance optimization is the runtime performance data. The most straightforward way to acquire the runtime performance data is to use the performance counters integrated on the chips of modern processors. These counters are registers specifically designed for monitoring runtime events such as cache misses, context switches, translation lookaside buffer misses, and so on. To access the performance counters, a counter interface is required. Over the last few years, different low-level libraries or high-level interfaces have been implemented for programmers to access the performance counters in the source codes. *Perf* [17] is a low-level profiling tool delivered by the Linux kernel. It provides a simple command-line interface for users to start profiling a running application. *Perf* supports a number of measurable events, including software events like context switches and page faults, as well as hardware events such as the number of CPU cycles and cache misses. *Perf* aggregates the occurrences of the user-specified events and provides the profiling results as perf reports at the end of application's execution. The Performance Application Programming Interface (PAPI) [7] is a well-known and widely applied programming interface for accessing the hardware counters within an application's source code. PAPI provides both simple high-level interface for the acquisition of simple measurements and fully programmable low-level interface to the underlying counter hardware. The low-level PAPI interface deals with hardware events in groups, while the high-level interface simply provides the ability to start, stop, and read specific events. Aside from *perf* and PAPI, there are other well-known programming interfaces for hardware counters. OProfile [31] and *Perfmon* [39] are two system-wide profilers for Linux systems.

The performance data delivered by the counter interface are still at a low level even in the text format. To show the runtime application behavior at a high level, different visualization tools or analysis frameworks have been implemented in the research area [12], [15], [18], [28], [34]. Nevertheless, the widely used tools have been the well-known Tuning and Analysis Utilities (TAU), Vampir and VTune.

The TAU [38] is a profiling and tracing toolkit for performance analysis. It consists of a visualization component providing graphical displays of performance analysis results. This allows the user to identify the source of performance hot spots in the programs. The tool has been adopted by different researchers to characterize the I/O performance [37], to measure the GPU performance [26], and to study the applications' runtime behavior [19]. The Centre for Information Services and High Performance Computing at the University of Dresden developed Vampir [8] for performance visualization. It was originally developed to show the communication bottlenecks of the Message Passing Interface (MPI) applications but was extended to depict the cache performance. It can show the number of cache misses in a time-line view as well as in the source code, allowing the detection of cache critical code regions. The Intel VTune Performance Analyzer [21] is regarded as a useful tool for performance analysis. It provides several views, like "Sampling" and "Call Graph," to help programmers identify bottlenecks. For memory performance, it shows the absolute number of cache misses in a code region, allowing the user to detect functions and even code lines that introduce excessive cache misses.

While the visualization approach can depict the runtime bottlenecks and abnormal behavior for small applications, it is not adequate for showing the runtime behavior of applications with a large data set. In this case, we propose to use bioinformatics and data mining techniques to analyze the performance data. Several researchers also applied this approach for other purposes. Olaru *et al.* [30] investigated the application of classification and regression techniques in power system engineering. Athanasopoulou *et al.* [1] used classification to predict control monitoring rules in order to optimize the efficiency of electric power generation processes. In further fields, Letourneau *et al.* [24] investigated the use of decision trees, instance-based learning, and naive Bayes classifiers to optimize aircraft component replacements. Kusiak and Song [23] employed linear regression, neural networks, and decision trees to optimize combustion efficiency of coal-fired boilers.

The concrete task in this work is to use the conventional algorithms to process the memory performance data on shared-memory systems. The goal is to detect the cases of data access inefficiency for a further optimization with respect to data locality. As the memory performance is critical for the overall performance, researchers have been finding ways to improve the performance of the memory system. An example is the research work [16] conducted in the Unified Parallel C Group that is developing software utilities for shared-memory programming, as well as tools for analysis and tuning of shared-memory access performance on distributed systems. For the performance tuning, this research group developed a tool to monitor the life cycles of shared variables and to track the accesses to them. The collected performance data are analyzed using a specific mechanism for finding a problem. Our work is similar to this work, but we use more accurate detailed performance data delivered by a hardware device, which enables the detection of a correct location for the data sets. The efficiency of this approach is proven by our experimental results that show a significant performance gain with the locality optimization.

III. Analyzing the Runtime Memory Access Behavior

A memory hierarchy on a single-core processor is basically composed of a main memory and several levels of caches. On a multicore machine, the main memory is shared by the processing cores. On a distributed system, however, the main memory is distributed across the processing nodes. Therefore, the traditional way to program such architectures is to use the MPI. For enabling the simple shared-memory programming models, the memories on a distributed system may also be organized to form a virtually shared memory and a memory hierarchy with cache, local memory, and remote memory. The access time of a memory location with this hierarchy gets longer as the distance of the location to the processor gets larger. This means that the access to the caches is much faster than the accesses in the main memory, and the latency difference between a local memory

access and a remote memory access is even larger. Therefore, both data locality in the cache and memory locality are critical issues for the memory performance on a distributed-shared-memory system.

A. Detecting Memory Access Patterns for Cache Optimization

Due to the limited size of a cache memory, it is not possible to load the entire working set into the cache for fast accesses. Therefore, any kind of cache optimization has a final goal, i.e., to enable the reuse of the data loaded in the cache. This can be done by organizing the data with respect to the affinity between the data sets. In this paper, we mainly address the repeated address sequences. This information helps users order the data with spatial locality in the same cache block so that all data in the block are accessed by the processor.

1) Performance Data: The performance data for our cache optimization are delivered by a code instrumentor called Doctor, which was developed as a part of Augmint [29], a multiprocessor simulation toolkit for Intel x86 architectures. Doctor is used to augment the assembly codes with instrumentation instructions that generate memory access events. For every memory reference, Doctor inserts code to pass its address and the identification (ID) of the processor that performs the access. For this work, we slightly modified Doctor to generate a trace file that stores all memory accesses performed by an application at the runtime.

2) Using Teiresias to Detect Repeated Access Sequences: A repeated access sequence contains several memory addresses that are repeatedly accessed in the same order. For example, from this sample access address trace (100 200 300 400 $\cdots$ 100 200 301 400 $\cdots$ 100 200 302 400 $\cdots$ 100 200 303 400$\cdots$), we can observe two repeated sequences: 100, 200 and 100, 200, –, 400, where "–" is a position holder representing any memory address.

It can be seen that this access sequence is similar to a DNA, RNA, or protein sequence in bioinformatics. Therefore, we adopt the Teiresias [35] algorithm, which is often applied for pattern reorganization in bioinformatics. Teiresias works with a long sequence. It first finds small patterns in the sequence and then constructs them into larger ones. For small patterns, users have to specify a maximal length of the pattern (L) and the number of letters (N) in the pattern. The position holders are not counted as letters. For example, both of the patterns "ABC" and "$A{-}B{-}C$" have three letters, so that $N = 3$.

For generating small patterns with user-specified L and N, Teiresias first builds patterns of length one and then extends them to patterns of length L starting and ending with a letter. For each detected pattern, the Prefixes and Suffixes are computed. A Prefix is a prefix of a pattern that needs to end with a letter and contains at least two letters. A Suffix is a suffix of a pattern with a letter at the starting position. For instance, the possible Prefixes of the two aforementioned patterns are AB and $A{-}B$, while their Suffixes are BC and $B{-}C$. Based on the Prefixes and Suffixes, Teiresias then performs a convolution phase to those pairs of small patterns, where the Prefix of one pattern and the Suffix of the other are the same. Such pairs are combined to form larger patterns. For example, from the pattern $DF{-}A{-}T$ and $A{-}TSE$, Teiresias generates a pattern $DF{-}A{-}TSE$. We use the Teiresias algorithm to process the memory trace generated by the code instrumentor. To shorten the processing time for large performance data, we introduce a parameter K to specify the minimal occurrence of an address sequence, which reduces the number of small patterns and, thereby, the time for generating larger patterns in the convolution phase.

The algorithm was verified with a number of shared-memory applications from the SPLASH-II benchmark suite [44]. A common feature observed for all applications is that the number of patterns detected by the algorithm decreases with enlarging the length of the pattern L or the minimal occurrence K. However, for large groups, our algorithm is still capable of finding the repeated patterns. For example, with the RADIX application, Teiresias reported that more than 20 address groups of length 15 repeated at least 10 000 times. If the program is optimized by allocating the related data in the same cache block, cache misses will be significantly reduced.

B. Using Decision Trees to Predict Data Location

As mentioned, a distributed shared memory is comprised of a set of main memories that are distributed across the computing nodes. How to organize the working set of an application in these memories is a critical issue for the memory performance and, furthermore, the overall performance. The goal of our performance analysis in this case is to find the best location for the entire working set toward a minimal communication between the processing nodes. In our context, the decision tree mechanism in data mining is particularly useful; when analyzing internode communication, we try to find the access character of a certain memory address or access region and the optimal processor node for it. Here, an optimal processor is the one that has performed the most accesses on the data item. It is clear that the data item shall be placed on this processor for less remote accesses. Hence, we reduce the problem of internode communication into the problem of predicting an optimal node with a set of given attributes. At this point, we use the decision tree as a predictive model to conclude the target node of a data item.

1) Performance Data: This task requires knowledge about the runtime data transfers among computing nodes of a distributed system. This information cannot be delivered by performance counters that cover only the events within a single processing node. For inspecting the internode communications, we specifically designed and implemented a hardware monitor with the following three components:

1) a B-Link interface to the network for extracting information from the packets transferred on the network;
2) a counter module for temporally storing the acquired information;
3) a peripheral component interconnect interface that allows the users to deliver the monitoring data to the user space.

These components interact in the following way. The B-Link interface snoops the B-Link over which all incoming and outgoing packets to and from the network interface are transferred. It extracts information from the communication packets. The

information contains transaction commands (read, write, and lock), source and destination IDs, memory addresses (page number and offset), and packet descriptions (incoming, outgoing, response, and request). The acquired data are handed from the B-Link interface on to the counter module. As the primary part of the hardware monitor, the counter module is responsible for the recording of the monitoring events. It is organized with an event filter, a static counter array, and an associative counter array. The event filter is available for users to define events of interest. Such an event is actually a memory transaction performed on a special memory region specified by a page number and the top and bottom addresses. The two counter arrays are registers used to store the monitoring data.

In order to accommodate both analysis of specific events and performance overview, the hardware monitor offers two working modes: a static and a dynamic mode. In the former case, the integrated filter in the counter module allows the monitor to only trigger user-specified events. With the dynamic mode, the complete internode traffic is captured and recorded. For this work, we use dynamic monitoring to trace all packets delivered on the network, thus acquiring a full histogram of the communications between the processing nodes on the system. Each record in the histogram contains four attributes: source, destination, access type, and access address. Here, source and destination specify the sender and receiver of a packet, i.e., a remote memory reference on a distributed-shared-memory architecture.

2) Location Detection With Data Clustering: Data mining currently is an important approach to find information in large sets of data. The idea of data mining is to apply statistical methods to a given data set in order to discover potentially new and useful knowledge.

The data mining process describes a commonly accepted approach to identify certain patterns in given data sets. First, the data to be analyzed need to be identified and made accessible. This is done by our hardware monitor. Second, the acquired data are transformed so that they can be used as an input for a certain data mining algorithm. Here, we rely on a mining tool to perform the data transformation. Finally, the data are mined, and the result is used to compute predictions.

Depending on the data and the pattern to identify, several different data mining algorithms exist. While clustering helps to identify several data tuples with similar properties, association rules find combinations of attributes frequently occurring together in the whole data set. Classification and regression are methods to predict one attribute value based on a set of input attributes. Our goal in this work is to predict the location of a data set based on the communication records with the information about source and destination. This is the task of the classification and regression rules. Hence, we use these rules to analyze the monitoring data for finding the access character of a memory address and the optimal node for it.

One approach to do classification is to learn a decision tree [11], [45]. Such a tree consists of one root node and several subsequent nodes, which represent decision rules. The leaf nodes finally represent the class of a tuple satisfying all preceding conditions. Fig. 1 depicts a simple example. An unknown attribute of a data tuple is then predicted by traversing through the tree until reaching a leaf node containing a classification. For our case, the leaves are the processors contained in the system, and the goal is to search the decision tree to achieve a leaf for the given address. Fig. 1 demonstrates a case for predicting the optimal location for the memory address 100 that is accessed by program 2. According to the decision tree, this block shall be allocated to processor 3 because, for all addresses bigger than 100, the best destination is this processor.

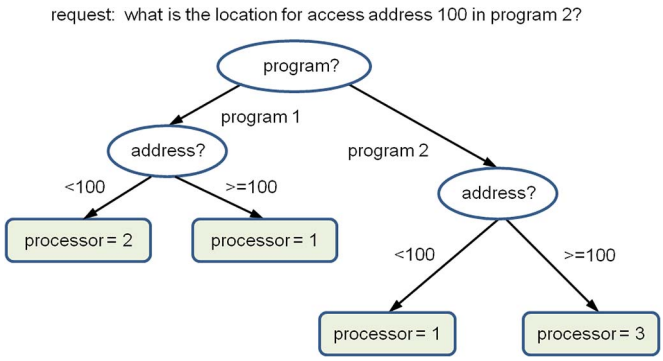


Fig. 1. Example of building a decision tree for memory locations.

Compared to the simple structure of decision trees and the easy way to classify data tuples, it is more complex to automatically learn decision trees from a database of sample records. The usual technique is a top-down induction of such trees, which starts at the root node and applies the same algorithm to all subsequent children. At every node, it has to decide which attribute to select and at which threshold value the attribute should be split. This is done in a greedy manner using entropy or information gain measures [4], [33]. If only records from the same class remain in one node or no further condition offers any improvement, the majority class is decided to be the final classification of the corresponding data tuples.

The monitoring data provide the access address for each single communication. This allows us to acquire the best position for individual data addresses. However, such fine granularity is generally not necessary for data locality optimization. Therefore, we transferred the data into a form in which the access frequency per processor is associated with each memory block the size of a virtual page. With this transformation, each data tuple consists of the desired memory block and one column for each processor in the system.

Fig. 2 depicts the whole procedure of analyzing the monitoring data. In the first step, the monitoring data are transformed. This is followed by the step of data clustering. The results are then delivered to the programmers with three different granularities: data set level, page level, and single address. The figure demonstrates a process of postprocessing the monitoring data. The approach can also be applied for runtime analysis of the monitoring data. However, the decision tree is built based on partial performance data and may not accurately predict the optimal nodes for applications with irregular access patterns.

IV. Validation and Results

We have applied the proposed approach to study the data affinities in standard benchmark applications and found interesting access sequences for cache optimization. However, since

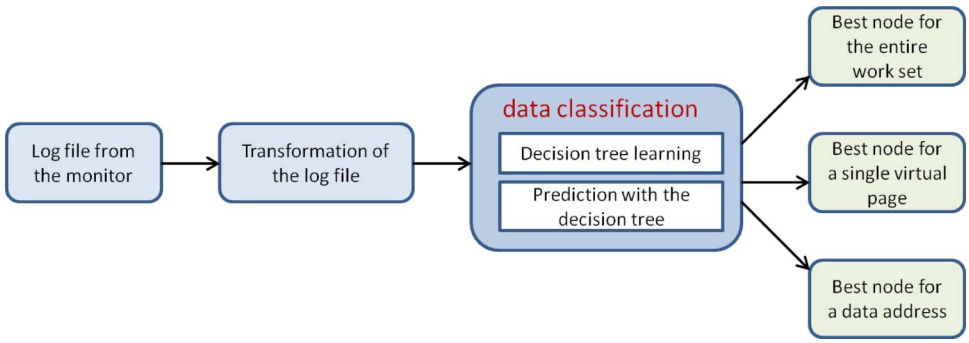


Fig. 2. Work flow of analyzing the monitoring data.

the memory locality problem on a distributed system is more challenging, we only performed optimizations in terms of the memory locality rather than the cache locality. This section shows our experimental results.

A. Experimental Applications

The applications for the experiment are chosen from the SPLASH-II benchmark suite [44]. The benchmark consists of several programs parallelized with m4 macros. The lower upper (LU) program factors a dense matrix into the product of a lower triangular matrix and an upper triangular one. The primary data structure in LU is the matrix being decomposed. For this experiment, we use a matrix of size 128×128. The fast Fourier transform (FFT) is a complex 1-D version of the "six-step" FFT algorithm described in [2]. The data set consists of n complex data points to be transformed and another n complex data points referred to as the roots of unity. For this experiment, FFT is computed using 2^{14} data points. RADIX implements an integer radix sort based on the method described in [3]. We perform this sort on 65 536 elements. WATER is an N-body molecular dynamics application that evaluates forces and potentials in a system of water molecules in the liquid state. For this experiment, a data size of 216 molecules is specified. The OCEAN program uses a restricted red–black Gauss–Seidel multigrid solver [5] to simulate the role of eddy and boundary currents on large-scale ocean movements. The simulation is performed for many time steps until the eddies and mean ocean flow attain a mutual balance. We use a grid of 130×130 to model the ocean basin. BARNES implements the Barnes–Hut method to simulate the interaction of a system of bodies (N-body problem). We perform this simulation using an N-body size of 1024.

B. Memory Locality Optimization

As mentioned and shown in Fig. 2, data mining provides us three results with different granularities: single memory address, individual virtual page, and complete data set. The first result enables very fine-grained optimization that allocates each data item to the corresponding processor node. However, such a granularity introduces high overhead, particularly for the case of runtime optimization. Therefore, we performed page-level optimization using single pages as an allocation unit and the coarse-grained optimization of allocating the whole data set on a single node, rather than the optimization with single memory addresses.

The optimization was performed by using annotations to explicitly specify the best location in the source code. In the case of page-level optimization, each virtual page is specifically allocated on its dominating node, while with the coarse-grained optimization, a single node is specified for the entire working load. A dominating node is the computing node that performs the most accesses on the virtual page.

The baseline for both optimization versions is the conventional data placement, i.e., all data are placed on the host node on which the job is submitted. We also addressed the first-touch scheme [40], which is usually applied to evaluate memory optimization on systems with a distributed shared memory. The first-touch scheme allocates data on the node that first accesses it. This scheme tends to behave better than other data distribution policies, because the node that first accesses a page is usually the node that mostly accesses it.

C. Optimization Results With a Strict Access Policy

The first experiment studies the direct impact of the memory locality optimization. For this, we ran all applications with different code versions, including the first-touch scheme, node-level optimization (opt-coarse), page-level optimization (opt-fine), and the conventional host-node data placement. We also executed the applications using different numbers of processing nodes in order to study the scalability of our locality optimization approach. According to the requirement of some applications, the number of processors is specifically chosen as a power of two. For each test, the execution time of all applications was measured, and the speedup was calculated by dividing the execution time of an optimized version via the time needed for running the code with the conventional data placement policy.

For the following tests, we use the most strict communication policy, which allows only one node to access a remote memory at the same time. During this delay, no other nodes can perform remote accesses.

Fig. 3 shows the experimental results of the coarse-grained optimization using two processing nodes. For each application, the figure demonstrates two results, with one showing the speedup of the first-touch version against the transparent version in absolute execution time and the other for the optimized version with a coarse granularity.

Observing the second bars with each application in the figure, it can be seen that all applications achieve performance improvement with the optimized version in contrast to the

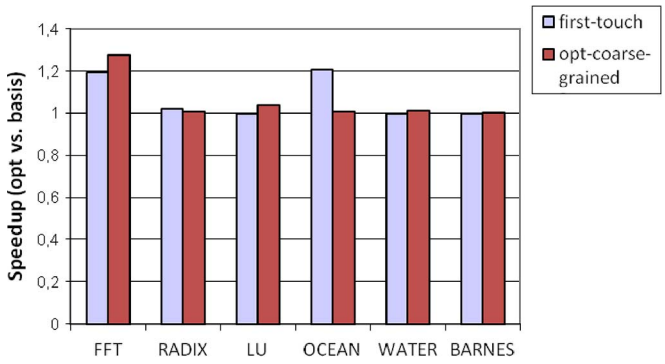


Fig. 3. Improvement of coarse-grained optimization on two nodes.

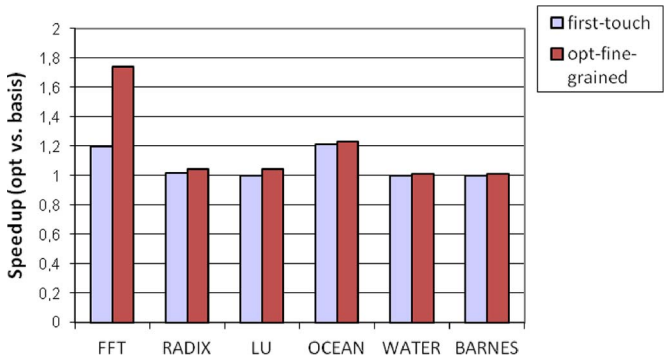


Fig. 4. Improvement of fine-grained optimization on two nodes.

default data allocation policy. For individual applications, FFT depicts the best performance gain with a speedup of 1.3 to the original code version. Other applications, however, show only a slight speedup. Nevertheless, most of the applications depict an improvement with the coarse-grained optimization in contrast to the first-touch scheme. This is demonstrated in the figure by the fact that the right bar for each application is higher than the left one, except the application RADIX and OCEAN.

Studying the results with fine-grained optimizations, we found different behaviors with some applications. Observing Fig. 4 that shows the speedup of both first-touch scheme and fine-grained optimization against the default data allocation policy, it can be seen that OCEAN achieves a speedup of 1.2 with our optimization and the optimized version performs now better than the first-touch scheme. RADIX also shows a better performance with this optimization in contrast to the first-touch scheme. For FFT, the speedup of our optimization is as high as 1.74.

To observe the applications' behavior on a larger system, we ran all applications using 32 processing nodes. Fig. 5 shows the experimental results, where each application has three bars corresponding to the speedup with the first-touch scheme, coarse-grained optimization, and fine-grained optimization. It can be clearly seen that the fine-grained optimization performs better than the other two cases for most of the applications. With the application RADIX, for example, a speedup of two has been achieved. Similarly, OCEAN also shows a high performance gain. This indicates that, on larger systems, coarse-grained optimization may not help significantly reduce the remote memory accesses, and in this case, fine-grained tuning at the page level rather than the whole working set is necessary.

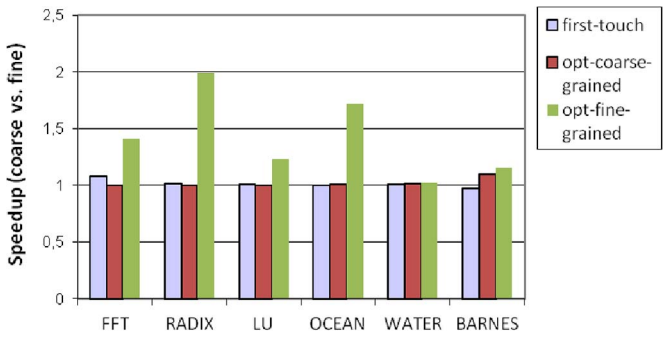


Fig. 5. Speedup of optimizations on 32 nodes.

Combining all three figures, it can be seen that, for all applications, except FFT, the first-touch scheme has no speedup to the basic policy, meaning that this scheme results in the same execution time as the default version. This also indicates that the first-touch scheme brings the same runtime data layout as the host-node scheme that allocates all data sets on the same node, namely, the host node. The reason may lie in the data initialization, which is usually done by the host and the first-touch scheme that puts all shared data on the host node that first accesses the data. Similarly, coarse-grained optimization allocates the entire data on a single node and therefore cannot introduce large performance achievement. This is because the global dominating node for the whole data set is discovered based on the total remote accesses each node performed at the runtime. Most pages are shared by many processors. The page-level optimization improves performance because of the existence of a dominating node for each individual page. However, we note that, in most cases, this dominating node varies from page to page. Therefore, node-level optimization could only improve the data locality in smaller systems. For example, on a two-node machine, there are only two candidates for a page; therefore, placing all data on the global dominating node can achieve speedup. However, on larger systems, the best position of a virtual page can be any of the processors. In this case, only a locality optimization with individual data sets can help improve the data locality.

The good performance with the page-level optimization is directly attributed to the reduction of remote accesses. For a deeper insight into this issue, we measured the number of remote accesses for both page-level optimization and transparent data placement. We then computed the reduction rate as the percentage of reduced remote accesses achieved by the optimized version to the total remote accesses introduced by the transparent version. Fig. 6 depicts the results with all applications.

Observing the curves in the figure, it can be seen that, for most applications, including LU, WATER, OCEAN, and BARNES, the reduction rate is higher on smaller systems than on larger ones. This is not surprising because, on smaller systems, a data page is requested by few processor nodes. However, on large systems, the references to a single page are distributed across a set of nodes that possibly all require the page frequently. This means that the optimization of exclusively putting the data on the dominating node can only remove the remote accesses from this node but not that of the others. Nevertheless, if an application has a lower shared degree, a

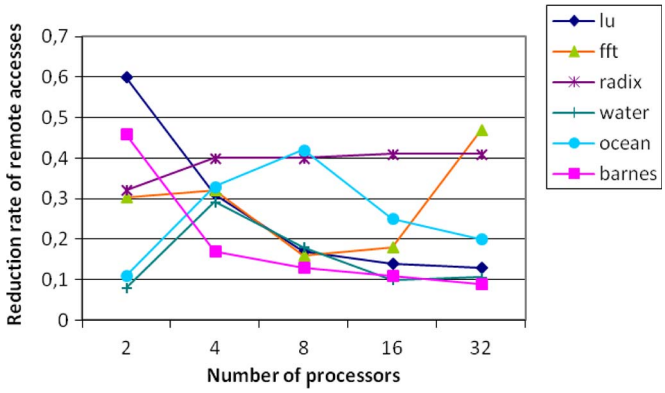


Fig. 6. Reduction rate of remote accesses.

data page would be potentially only dominantly accessed by a single processor. In this case, the reduction rate of remote accesses can still be high on larger systems. FFT and RADIX are such examples. With the FFT application, we achieved a reduction of as much as 47% in remote memory accesses on 32-processor systems, while RADIX shows a constant reduction rate of 40% from 4 to 32 processors. Therefore, we observed a high performance gain with these two applications.

In summary, the applications demonstrate different behaviors with our locality optimization. This distinction lies on the access pattern of each individual program. Using the proposed approach, we found that, for the LU application, nearly half of the total data pages are accessed equally by many processors, while the other half have several dominating nodes. Only 6% of the pages are accessed mainly by a single processor. We have placed each page to its best position suggested by our data classifier. However, for most pages, other nodes also require them. Hence, still many remote accesses exist after the location tuning. Similarly for WATER, although 36% of the data pages are dominantly accessed by a single node, the number of accesses by this node is only 2% of the total remote accesses. The others are shared by all or almost all processors, and these processors equally perform accesses to an individual page. For BARNES, the whole shared data are accessed by all nodes. Even though some nodes do not frequently request a page, there exists no dominating node for any data page. This means that at least two processors equally access the same page. Therefore, it is difficult to optimize this code. With FFT, 96% of the data pages have a dominating node, and 33% of them are exclusively accessed by this node. RADIX is even better with more than a half of the data pages accessed by a single node. For OCEAN, nearly 70% of the pages are exclusively used by one node, and 2/3 of the rest have a clear dominating processor. Therefore, the optimization with these three applications leads to a considerable performance gain.

D. Optimization With Concurrent Internode Communication

The results described previously are achieved with the assumption that only one remote access can be performed at the same time on the network. Many interconnection technologies allow concurrent communication between several processor pairs. In order to measure the maximal potential of performance gain with the memory optimization, we executed all

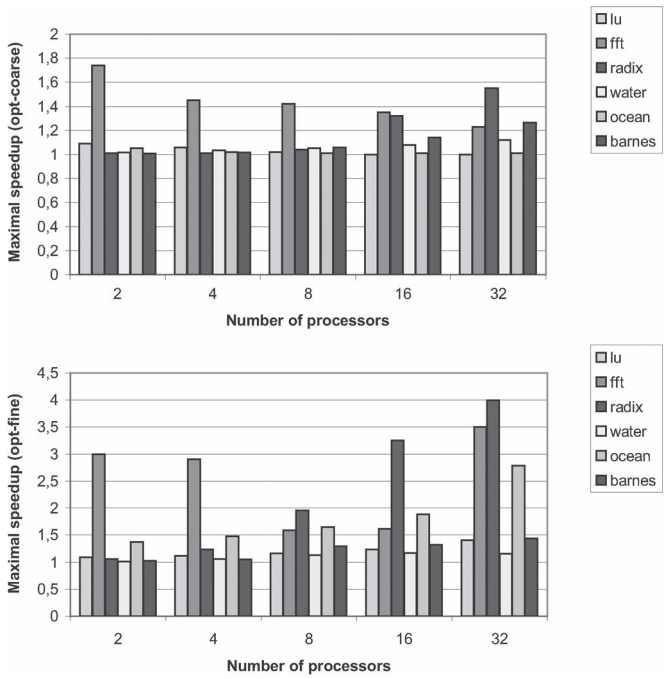


Fig. 7. Maximal improvement in execution time with both optimizations [(upper) coarse-grained optimization; (lower) page-level optimization].

applications with the condition that remote accesses from each processor can be handled immediately without waiting for a free connection.

Fig. 7 depicts the speedup of both node-level (upper figure) and page-level (lower figure) optimizations versus the host-node allocation policy. In comparison with the speedup shown in the previous section, the overall behavior with individual applications does not change, but a higher speedup can be clearly seen, as well as with the node-level optimization. The best case is the RADIX program on 32 processors, where a speedup of factor four has been acquired.

E. Improvement in Scalability

The last experiment aims at studying the scalability, an important performance metric for evaluating parallel systems. A system is scalable when the speedup of a parallel execution to the sequential execution increases linearly with the number of processors. We computed the speedup with all applications and different system scales, by dividing the time of the sequential execution with the time of a parallel run.

Fig. 8 shows the experimental results. The top diagram presents the speedup curves with the transparent version, i.e., using the host-node-based data allocation. These curves give us a first image that only one (the program BARNES) out of all six speedup lines goes linearly, indicating a good scalability. The curves with RADIX and WATER are lineal by eight processors, indicating a scalability on smaller systems. However, on large systems, they do not perform well. It can be seen that, with both codes, the parallel execution using 32 processors is longer than that with 16 nodes. LU and FFT show poor parallel performances, where speedup values of 3.20 and 4.65 were achieved on a 32-node system individually. The speedup line of OCEAN increases continuously but slightly. With this program, only a speedup of ten is achieved on 32-node systems.

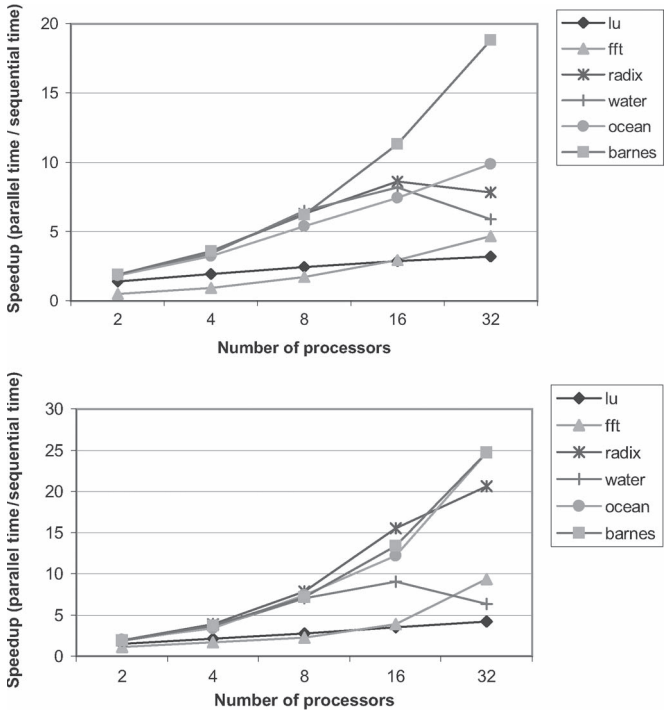


Fig. 8. Scalability comparison between transparent and optimized versions [(upper) parallel speedup with unoptimized version; (lower) parallel speedup with page-level optimization].

Switching to the other diagram in Fig. 8, we first see that RADIX and OCEAN are now scalable due to the page-level memory locality tuning. The speedup on 32-node systems, for instance, is more than 20 for both applications. WATER, however, still shows a slowdown with 32 processors in contrast to 16 processors, but this slowdown is much smaller than the unoptimized case. With LU, we did not get a significant improvement. For FFT, the optimization is quite efficient, particularly in the case of 32 processors where the speedup after the optimization is 100% more than that acquired with the unoptimized version.

In summary, the experimental results show that our data locality optimization approach can improve both single execution and scalability of parallel performance. Depending on the access pattern of applications and their individual performance problems, the improvement varies between applications.

V. Conclusion

Computing architectures and data structures are getting more complicated. This makes it hard for programmers to develop efficient applications. Performance tuning is nearly a necessary task for application developers.

We have proposed a code optimization approach of applying conventional algorithms to analyze the runtime performance data and further optimizing the applications manually based on the analysis results. Since both performance data and analysis mechanisms have high accuracies, we achieved a significant performance gain with the tested applications.

In the next step of this research work, we will study performance co-optimization, which involves several performance metrics combined into a single optimization process. For this, the data mining algorithms will help find the best tradeoff between these metrics. In addition, we plan to use real applications to validate the optimization results.

References

[1] C. Athanasopoulou, V. Chatziathanasiou, M. Komninou, and Z. Petkani, "Applying knowledge engineering and data mining for optimization of control monitoring of power plants," in *Proc. 6th IASTED Int. Conf. EuroPES*, 2006, pp. 190–195.

[2] D. H. Bailey, "FFTs in external or hierarchical memory," *J. Supercomput.*, vol. 4, no. 1, pp. 23–35, Mar. 1990.

[3] G. E. Blelloch *et al.*, "A comparison of sorting algorithms for the connection machine CM-2," in *Proc. 8th Annu. ACM Symp. Parallel Algorithms Architectures*, Jul. 1991, pp. 3–16.

[4] C. Borgelt, "A decision tree plug-in for data engine," in *Proc. 6th EUFIT*, Aachen, Germany, 1998, vol. 2, pp. 1299–1303, Verlag Mainz.

[5] A. Brandt, "Multi-level adaptive solutions to boundary-value problems," *Math. Comput.*, vol. 31, no. 138, pp. 333–390, Apr. 1977.

[6] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. London, U.K.: Chapman & Hall, 1993.

[7] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "Portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000.

[8] H. Brunst *et al.*, "Comprehensive Performance Tracking With Vampir 7," in *Tools for High Performance Computing*. New York, NY, USA: Springer-Verlag, 2009, pp. 17–29.

[9] B. R. Buck and J. K. Hollingsworth, "Data centric cache measurement on the Intel Itanium 2 processor," in *Proc. SuperComput.*, Nov. 2004, p. 58.

[10] CERN, LHC—The Large Hadron Collider, Meyrin, Switzerland, 2013. [Online]. Available: http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/

[11] A. Chaddad, P. O. Zinn, and R. R. Colen, "Brain tumor identification using Gaussian mixture model features and decision trees classifier," in *Proc. Annu. Conf. Inf. Sci. Syst.*, Mar. 2014, pp. 1–4.

[12] D. Chen, D. Li, M. Xiong, H. Bao, and X. Li, "GPGPU-aided ensemble empirical mode decomposition for EEG analysis during anaesthesia," *IEEE Trans. Inf. Technol. BioMed.*, vol. 14, no. 6, pp. 1417–1427, Nov. 2010.

[13] D. Chen *et al.*, "Natural disaster monitoring with wireless sensor networks: A case study of data-intensive applications upon low-cost scalable systems," *MONET*, vol. 18, no. 5, pp. 651–663, Oct. 2013.

[14] D. Chen, L. Wang, G. Ouyang, and X. Li, "Massively parallel neural signal processing on a many-core platform," *Comput. Sci. Eng.*, vol. 13, no. 6, pp. 42–51, Nov./Dec. 2011.

[15] D. Chen *et al.*, "Massively parallel modelling & simulation of large crowd with GPGPU," *J. Supercomput.*, vol. 63, no. 3, pp. 675–690, Mar. 2013.

[16] G. Cong, H. Wen, H. Murata, and Y. Negishi, "Tool-assisted optimization of shared-memory accesses in UPC applications," in *Proc. 14th Int. Conf. High Perform. Comput. Commun.*, Liverpool, UK, Jun. 2012, pp. 104–111.

[17] A. C. de Melo, "The new linux "perf" tools," in *Proc. 17 Int. Linux Syst. Technol. Conf.*, Sep. 2010, pp. 1–42.

[18] P. Guo, L. Wang, and P. Chen, "A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1112–1123, May 2014.

[19] J. R. Hammond, S. Krishnamoorthy, S. Shende, N. A. Romero, and A. D. Malony, "Performance characterization of global address space applications: A case study with NWChem," *Concurrency Comput., Practice Experience*, vol. 24, no. 2, pp. 135–154, Feb. 2012.

[20] F. Huang, D. Liu, X. Li, L. Wang, and W. Xu, "Preliminary study of a cluster-based open-source parallel GIS based on the GRASS GIS," *Int. J. Digital Earth*, vol. 4, no. 5, pp. 402–420, 2011.

[21] Intel, Intel VTune amplifier XE 2013: Performance and thread profiler, Mountain View, CA, USA, 2013. [Online]. Available: http://software.intel.com/en-us/intel-vtune-amplifier-xe

[22] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.

[23] A. Kusiak and Z. Song, "Combustion efficiency optimization and virtual testing: A data-mining approach," *IEEE Trans. Ind. Informat.*, vol. 2, no. 3, pp. 167–184, Aug. 2006.

[24] S. Letourneau, F. Famili, and S. Matwin, "Data mining to predict aircraft component replacement," *IEEE Intell. Syst. Appl.*, vol. 14, no. 6, pp. 59–66, Nov./Dec. 1999.

[25] Y. Ma *et al.*, "Distributed data structure templates for data-intensive remote sensing applications," *Concurrency Comput., Practice Experience*, vol. 25, no. 12, pp. 1784–1797, Aug. 2013.

[26] A. D. Malony *et al.*, "Parallel performance measurement of heterogeneous parallel systems with GPUs," in *Proc. Int. Conf. Parallel Process.*, Sep. 2011, pp. 176–185.
[27] J. Marathe, F. Mueller, and B. de Supinski, "A hybrid hardware/software approach to efficiently determine cache coherence bottlenecks," in *Proc. Int. Conf. Supercomput.*, Jun. 2005, pp. 21–30.
[28] S. Merchant and G. Prabhakar, "Tool for performance tuning and regression analyses of HPC systems and applications," in *Proc. 19th Int. Conf. High Perform. Comput.*, Pune, India, Dec. 2012, pp. 1–6.
[29] A.-T. Nguyen, M. Michael, A. Sharma, and J. Torrellas, "The Augmint multiprocessor simulation toolkit for Intel x86 architectures," in *Proc. IEEE Int. Conf. Comput. Des., VLSI Comput. Processors*, Oct. 1996, pp. 486–490.
[30] C. Olaru, P. Geurts, and L. Wehenkel, "Data mining tools and applications in power system engineering," in *Proc. 13th PSCC*, 1999, pp. 324–330.
[31] OProfile, A System Profiler for Linux, 2013. [Online]. Available: http://oprofile.sourceforge.net/
[32] B. Quaing, J. Tao, and W. Karl, "YACO: A user conducted visualization tool for supporting cache optimization," in *Proc. 1st Int. Conf., HPCC*, vol. 3726, *Lecture Notes in Computer Science*, Sorrento, Italy, Sep. 2005, pp. 694–703.
[33] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993.
[34] R. Ranjan, R. Buyya, and A. Harwood, "A case for cooperative and incentive based coupling of distributed clusters," in *Proc. 7th IEEE Int. Conf. Cluster*, Boston, MA, USA, Sep. 2005, pp. 1–11.
[35] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm," *Bioinformatics*, vol. 14, no. 1, pp. 55–67, Jan. 1998.
[36] SDSC, Gordon at the San Diego Supercomputing Center, La Jolla, CA, USA, 2013. [Online]. Available: http://www.sdsc.edu/us/resources/gordon/
[37] S. Shende, A. Malony, W. Spear, and K. Schuchardt, "Characterizing I/O performance using the TAU performance system," in *Proc. ICPP Parco Conf. Exascale Mini-Symp.*, 2011, pp. 1–10.
[38] S. Shende and A. D. Malony, "The TAU parallel performance system," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, May 2006.
[39] Sourceforge, Perfmon2—Improving Performance Monitoring on Linux, 2013. [Online]. Available: http://perfmon2.sourceforge.net/
[40] H. Takashi, O. Hiroshi, I. Takayoshi, and D. Henry, "Automatic data distribution method using first touch control for distributed shared memory multiprocessors," in *Proc. Languages Compilers Parallel Comput. Int. Workshop*, vol. 2624, *Lecture Notes in Computer Science*, 2001, pp. 147–161.
[41] L. Wang, D. Chen, Z. Deng, and F. Huang, "Virtual workflow system for distributed collaborative scientific applications on grids," *Comput. Electr. Eng.*, vol. 37, no. 3, pp. 300–310, May 2011.
[42] L. Wang, D. Chen, Y. Hu, Y. Ma, and J. Wang, "Towards enabling cyberinfrastructure as a service in clouds," *Comput. Electr. Eng.*, vol. 39, no. 1, pp. 3–14, Jan. 2013.
[43] L. Wang, D. Chen, and F. Huang, "Large scale distributed visualization on computational grids: A review," *Comput. Electr. Eng.*, vol. 37, no. 4, pp. 403–416, Jul. 2011.
[44] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Architecture*, Jun. 1995, pp. 24–36.
[45] J. Wyffels *et al.*, "Using a decision tree for real-time distributed indoor localization in healthcare environments," in *Proc. Int. Conf. Develop. Appl. Syst.*, May 2014, pp. 103–109.
[46] W. Zhang *et al.*, "Towards building a multi-datacenter infrastructure for massive remote sensing image processing," *Concurrency Comput., Practice Experience*, vol. 25, no. 12, pp. 1798–1812, Aug. 2013.

Jiaqi Zhao received the Master's degree from Northeast Normal University of China, Changchun, China, in 2006.

She is currently a Lecturer with Changchun University of Technology, Changchun. Her major research areas are vision cognitive algorithms and cloud computing. She has participated in the development of several research projects, and her research results have been published in several international conferences and journals.

Changlong Xue, photograph and biography not available at the time of publication.

Jie Tao received the Ph.D. degree from Munich University of Technology, Munich, Germany.

She has been a Lecturer and a Research Associate with Munich University of Technology. She is currently with Karlsruhe Institute of Technology, Karlsruhe, Germany. She is a Principal Investigator of several research projects. She has authored or coauthored 130 articles. Her earlier research focus was mainly on parallel programming models and performance tools. In the last years, she worked intensively on grid, cloud and data-intensive computing, and the virtualization technologies.

Dr. Tao has served as a Cochair or a Program Committee Member of international conferences/workshops and a Guest Editor of several journals.

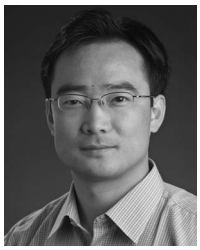
Rajiv Ranjan (S'06) received the Bachelor's degree in computer engineering from North Gujarat University, Gujarat, India, in 2002 and the Ph.D. degree in computer science and software engineering from The University of Melbourne, Melbourne, Vic., Australia, in 2009.

He is currently a Senior Research Scientist, a Julius Fellow, and a Project Leader with the Computational Informatics, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Canberra, ACT, Australia, where he is working on projects related to cloud and service computing. Previously, he was a Senior Research Associate (Lecturer Level B) with the School of Computer Science and Engineering, University of New South Wales, Sydney, N.S.W., Australia. He has authored or coauthored 84 (37 journal papers, 31 conference papers, nine book chapters, and seven books) scientific publications; approximately 70% of his journal papers and 60% of his conference papers have been A*/A ranked publication, according to the Australian Research Council's Excellence in Research for Australia (ERA). His papers have appeared at selective highly reputed venues, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (ERA A*), the IEEE TRANSACTIONS ON COMPUTERS (ERA A*), the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the Journal of Computer and System Sciences (ERA A*), the World Wide Web Conference (Core/ERA A*), the IEEE Intelligent Systems, Future Generation Computer Systems (ERA A), the IEEE Communications Surveys and Tutorials (#1 Computer Science Journal 2011), the IEEE Systems Journal, the Journal of Software Practice and Experience (ERA A), Springer Journal of Computing (ERA A), and the Journal of Concurrency and Computation: Practice and Experience (ERA A). He is broadly interested in the emerging areas of cloud, grid, and service computing. The main goal of his current research is to advance the fundamental understanding and state of the art of provisioning and delivery of application services in large, heterogeneous, uncertain, and evolving distributed systems (cloud, grids, data center, and web services).

Dr. Ranjan was a recipient of the following recognitions for his excellent research: 1) Best Paper Award from the IEEE HPCC 2013 Conference; 2) Outstanding Journal Paper on New Communications Topics Award (2009) from the IEEE Communications Society; 3) Goldstar Award (for excellence in research and having a near miss at ARC Discovery Project Grant) from the University of New South Wales; and 4) Julius Career Award (for excellence in research) from CSIRO. Recently (April 2014), he has been shortlisted as one of the finalists for the 2014 ICT Young Professional of the Year Award (administered by the Australian Government).

Joanna Kołodziej (M'11) received the MSD degree in theoretical mathematics and the Ph.D. in theoretical computer science from Jagiellonian University, Cracow, Poland.

Since September 1, 2012, she has been an Associate Professor with the Institute of Computer Science, Cracow University of Technology, Cracow. Previously, she was the Department of Mathematics and Computer Science, University of Bielsko-Biala, Bielsko-Biala, Poland. She is also a Research Collaborator of the Intelligent Information Systems Group with AGH University of Science and Technology, Cracow. The current topics of her research include grid and cloud computing, energy effectiveness and secure awareness in large-scale distributed systems, data intensive computing, and text mining.



Lizhe Wang (M'09–SM'12) received the B.E. and M.E. degrees from Tsinghua University, Beijing, China, and the D.Eng. degree from the University of Karlsruhe, Karlsruhe, Germany.

He is currently a Professor with the Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences (CAS), Beijing, China, and a Chutian Chair Professor with the School of Computer Science, China University of Geosciences (CUG), Beijing. He leads the high geo-performance computing group at CAS and the High Performance Computing Lab at CUG. His main research interests include high-performance computing, e-Science, and spatial data processing.

Prof. Wang is a Fellow of The Institution of Engineering and Technology and the British Computer Society.

Dan Chen received the B.Sc. degree in physics from Wuhan University, Wuhan, China, the M.Eng. degree from Huazhong University of Science and Technology, Wuhan, and the M.Eng. and Ph.D. degrees from Nanyang Technological University, Singapore.

He is currently a Professor with the School of Computer, Wuhan University. His research interests include computer modeling and simulation, high-performance computing, and neuroinformatics.