Using Traditional Data Analysis Algorithms to Detect Access Patterns for Big Data Processing

Jiaqi Zhao[1], Jie Tao[2], Lizhe Wang[3], Rajiv Ranjan[4], and Joanna Kołodziej[5]
[1]School of Basic Science, Changchun University of Technology, P.R. China
[2]Steinbuch Center for Computing, Karlsruhe Institute of Technology, Germany
[3]Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, China
[4] ICT Centre, Commonwealth Scientific and Industrial Research Organisation, Australia
[5] Institute of Computer Science, Cracow University of Technology, Poland
scorpiozhao@yahoo.com.cn, jie.tao@kit.edu, lizhe.wang@gmail.com, rranjans@gmail.com, jokoldziej@pk.edu.pl

***Abstract*—The data sets produced in our daily life is getting larger and larger. How to manage and analyze such big data is currently a grand challenge for scientists in various research fields. MapReduce is regarded as an appropriate programming model for processing such big data. However, the users or developers still need to efficiently program appropriate data processing actions related to their analytics requirements. In other words analytics actions in MapReduce is not portable across different big data types. In this paper we propose to adopt traditional data clustering algorithms to automatically analyze large data sets. We applied this approach to process performance data on distributed shared memory machines for detecting the application access patterns. The advantage is that application developers need not write codes to understand the runtime access behavior of their applications. We optimized several benchmark applications based on the analysis results and the experiments show a considerable improvement in terms of execution time and speedup.**

Keywords: Data Analysis, Memory Performance, Data Locality, Distributed Shared Memory, Code Optimization

I. Introduction

The amount of data created in our daily life is growing exponentially. Scientific instruments, the Web, and the simulation facilities are the major sources for this big data. The Large Hadron Collider [10] in the European Organization for Nuclear Research (CERN), for example, produces 15 Petabytes of scientific data every year. The Large Synoptic Survey Telescope (LSST) [11] generates 20 Terabytes per night, which aggregates to a total data volume of 60 Petabytes in a year. Massive data sets are not only being generated by scientific applications but also by Web operators including Google, Facebook, Twitter, and eBay. Today, data-intensive computing is gaining significant momentum and it is expected that data-based science will eventually bypass the conventional computation-based simulation science as the main driving force of HPC in the future [17].

As applications need to process larger and larger data sets, the performance data or statistics, such as cache miss, memory usage, processor throughput, and the like, which are collected at the runtime for understanding the applications execution behavior, are also growing up. It is a fact that the analysis of performance data is necessary for optimizing the placement of applications on the underlying computing platform like physical servers and virtual machines. We have been working with performance data in the last several years and developed different tools for acquiring and visualizing the performance data on both physical architectures [25] and the virtualized machines [34]. The visualization with graphical views makes it easy for programmers to understand and analyse the performance data. However, such tools do not work well for visualising the performance data related to processing massive or big data sets. The main reason behind this is the fact that a single view cannot highlight the abnormal points over large time series. This problem could be possibly eliminated by reducing the amount of data using either lossy tracing [23], 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, a full trace is required. Therefore, we need more advanced solutions for processing the large performance data.

Currently, MapReduce [12] is widely used for processing large data sets. With a Map and a Reduce function, MapReduce provides simple semantics for users to program data analysis tasks. However, it can be burden for a programmer to write specialized code for understanding and analysing the runtime performance data of his application.

Therefore, we follow a traditional approach by applying data clustering algorithms, concretely the classification and regression trees [6], to process the performance data. This solution is based on several considerations: 1) The data clustering algorithms are capable of self-learning. This guarantees the required accuracy of the analysis result and thereby the efficiency of the runtime performance tuning; 2) The same stream of performance data often contains several interesting patterns. For example, a memory access trace holds the access stride, access hotspots, 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; 3) Data clustering algorithms not only detect the problem but also can predict the trend. This allows the runtime adaptation system to take actions before the problem occurs; and 4) A complete system optimization is associated with different optimization targets that interact with each other. For example, optimization on the speedup can potentially enlarge the power consumption. The generic data analysis approach allows a combined analysis of different data streams and hence is potentially capable of delivering appropriate solutions for a balancing trade-off across all performance metrics.

For this concrete work we apply the proposed approach to conduct data locality optimization on a distributed shared memory system. Today shared memory models are increasingly used to develop parallel programs due to its simplicity. Traditionally, shared memory programs run on systems with a physically shared memory. Such systems, however, are limited to system scales. Therefore, architecture vendors are producing machines with a distributed shared memory. The memory hierarchy in such machines is usually multi-levels with different access time at each level. An example is the Gordon Supercomputer [14] at the San Diego Supercomputer Center, which is specifically designed for data-intensive computing. Gorden 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. A problem with such machines is the different latency between a local memory access and a remote access. This latency distinction can be several factors, as on some commercial machines like SGI Origin and Altix, and even one hundred folds such as on Gorden.

The main novel contributions of our paper include: (i) optimization of data allocation of parallel application on a shared memory cluster via intelligent analysis of runtime performance data. We collected the performance data using a monitor simulator; (ii) application of the data mining techniques, i.e., the classification and regression trees, to the performance data; (iii) detection of the bottlenecks with large remote memory accesses; and (iv) conducting extensive experimental evaluations, which show that our approach changes 40% of the total memory accesses from remote to local hence achieving an improvement in execution time with a factor of two.

The remainder of the paper is organized as follows. Section II gives a brief overview of the related work. This is followed by introducing how the performance data are analyzed with traditional data analysis algorithms in Section III. In Section IV the experimental results with data locality optimization are demonstrated. The paper concludes in Section V with a short summary and future directions.

II. Related Work

Performance optimization is an important research area and has been intensively investigated. The basis for any performance optimization is to first understand the applications runtime behavior. To help application developers in this task, different visualization tools or analysis frameworks have been implemented in the last years.

The Tuning and Analysis Utilities (TAU) [28] 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 hotspots in the programs. The tool has been adopted by researchers to characterize the I/O performance [27], measure the GPU performance [22] as well as study the application behavior [15].

The Center for Information Services and High Performance Computing at the University of Dresdnen developed Vampir [8] for performance visualization. It was originally developed to show the communication bottlenecks of MPI applications, but 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 [18] 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 miss in a code region allowing the user to detect functions and even code lines that introduce excessive cache misses.

In addition to the visualization tools described above, analysis frameworks [32] and models [3] were also developed to support programmers with the task of analyzing performance data. Scalasca [32], [16] is a trace-analysis infrastructure supporting the performance optimization of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks, in particular those concerning communication and synchronization, and offers guidance in exploring their causes. Dinu, Pop and Cristea [13] developed a model for representing the patterns in the parallel time series describing the distributed system parameters and states. Based on the model, an application architecture was implemented for systems that adopt advanced machine learning techniques for detecting and learning patterns. The application was implemented as an add-on to the well-known MonALISA monitoring framework for distributed systems.

We go beyond of the traditional approach of using visualization to show the performance data by using classification and regression techniques that are usually used for data mining. Actually, these algorithms have been applied in other research fields.

Olaru et al. [24] investigated the application of classification and regression techniques in power system engi-

neering. 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, Létourneau et al. [20] investigated the use of decision trees, instance based learning and naïve Bayes classifiers to optimize aircraft component replacements. Kusiak and Song [19] employed linear regression, neural networks and decision trees to optimize combustion efficiency of coal-fired boilers.

Due to the fact that performance data are becoming larger and the data clustering algorithms can analyze large data with low overhead, we propose the approach of using these algorithms to process the performance data with a case study on distributed shared memory systems. The idea is to apply statistical methods to a given data set in order to discover potentially new and useful knowledge.

III. Detecting the Access Pattern

The prerequisite for any performance optimization is the runtime performance data. Modern processors are equipped with a performance monitoring unit that contains several registers for tracing the runtime events such as cache miss, context switches, TLB miss and so on. Most of the performance analysis tools, including those introduced in the previous section, rely on the performance counters to collect the runtime data. However, for this work the hardware counters cannot fully help us because they are not able to trace the communications between computing nodes in a distributed system.

We specifically designed a hardware monitor for this purpose. The hardware device is composed of three components: a B-Link interface to the network for extracting information from the packets transferred on the network; a counter array for temporally storing the acquired information; and a PCI interface that allows the users to deliver the monitoring data to the user space. The monitor can be configured into a static and a dynamic working mode. In the former case, an integrated filter in the counter array allows the monitor to only trigger user specified events. With the dynamic mode, the complete inter-node traffic is captured and recorded. For this work, we use the dynamic monitoring to inspect all packets delivered on the network, thus to acquire a complete histogram of the communications between the processor 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.

The next step is to apply data clustering approach to find useful information in large sets of data, concretely the access pattern for this work. The data clustering process describes a common approach to identify certain patterns in given data sets. The acquired data are first transformed to a specific format required by the applied algorithm and then mined to create results for computing predictions.

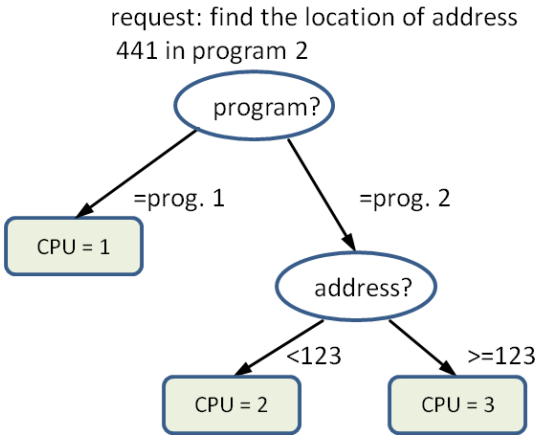

Figure 1. A sample decision tree for predicting a processor node

Depending on the input data and the expected patterns to identify, several different algorithms exist. While the 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.

With respect to performance data, classification algorithms are especially useful. When analyzing inter-node communications, we try to find the access characteristics 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 inter-node communication into the problem of predicting an optimal node with a set of given attributes.

One approach to do classification is to learn a decision tree. Such a tree consists of one root node and several subsequent nodes, which represent the decision rules. The leaf nodes finally represent the class of a tuple satisfying all the conditions denoted by parent nodes. Figure 1 depicts a simple example. An unknown attribute of a data tuple is 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. Figure 1 demonstrates a search process for the location of a data block containing the memory address 441 that is accessed by the program prog.2. According to the decision tree, this block shall be allocated to CPU 3 because for all addresses bigger than 123 the best destination is this processor.

Since the decision trees have a simple structure and forms

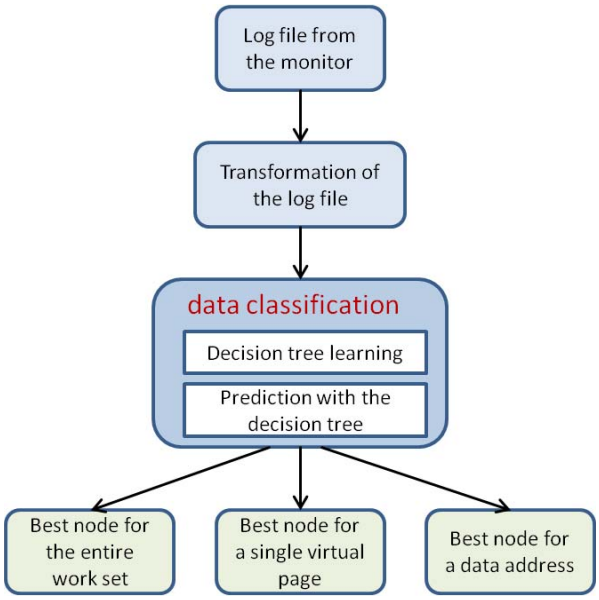

Figure 2. The analysis procedure for processing performance data

an easy way for classifying data tuples, it is more complex to automatically learn the 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 be decided which attribute has to be selected and at which threshold value the attribute should be split. This is done in a greedy manner using entropy or information gain measures [26], [5]. If only the records from the same class remain in one node or no further conditions offer any improvement, the majority class is decided to be the final classification of the corresponding data tuples.

Actually, the monitoring data provides the access address for each single communication. This allows us to acquire the best position of individual data addresses. However, such fine granularity is generally not necessary for data locality optimization. Therefore, we transferred the data into a format in which the access frequency per processor is associated with each memory block in 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.

Figure 2 depicts the whole procedure of analyzing the monitoring data. In the first step, the monitoring data is transformed. This is followed by data clustering. The results are then delivered to the programmers with three different granularities, i.e., data set level, page level and single address.

IV. Optimization and Experimental Results

In order to examine the feasibility of the discovered knowledge, we performed data locality optimization on several standard big data applications.

A. Experimental Setup

In order to conduct repeatable experiments under controlled settings we decided to develop a monitor simulator instead of using the real hardware monitor. We integrated the monitor simulator into a multiprocessor simulator [30] that focuses on modeling the memory system of Non-Uniform Memory Access (NUMA) architectures and the execution of shared memory applications parallelized with m4 macros used by, e.g., the SPLASH-II benchmark suite [33]. This simulator allows us to specify various memory configurations, including the cache associated parameters and the access latency for different memory locations. More specifically, it is possible to explicitly specify the location of the complete working set or an individual virtual page. We use this feature to realize both the coarse-grained and fine-grained memory locality optimization. In addition, the simulator delivers as output the execution time of a program and the number of accesses at each memory location. This allows us to study the impact of optimizations in the code in terms of the memory performance.

The applications for the experiments are chosen from the SPLASH-II benchmark. The benchmark consists of several programs. The 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. FFT is a complex, one-dimensional 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 simulated using 2^{14} data points. RADIX implements an integer radix sort based on the method described in [4]. We performed this sort on 65,536 elements. WATER is a N-body molecular dynamics application that evaluates the forces and potentials in a system of water molecules in the liquid state. For this experiment a data size of 216 molecules was specified. The OCEAN program uses a restricted Red-Black Gauss-Seidel Multigrid solver 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 performed this simulation using a N-body size of 1,024.

B. Locality Optimization

As mentioned and shown in Figure 2, data clustering provides us three results with different granularities, i.e., single memory address, individual virtual page and the complete data set. The first result enables extremely fine-grained optimization, i.e. to allocate each data item to the corresponding processor node. However, such a granularity introduces high overhead, especially for the case of run-time optimization. Therefore, we performed the fine-grained 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.

The optimization was enabled by using the annotations, provided by the simulation platform, 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 transparent 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 [29], which is usually applied to evaluate the memory optimization on systems with a distributed shared memory [21], [31]. First-touch 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 with Exclusive Data Access

Our first experiment aims at studying the direct impact of the optimizations. For this, we simulated all applications with first-touch, node-level optimization (opt-coarse), page-level optimization (opt-fine), and the conventional host-node data placement. We also executed the application using different numbers of processors in order to examine 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 time needed for running the code with the conventional data placement policy via the execution time of an optimized version, i.e., $S = T_{original}/T_{optimized}$.

The L1 cache is specified as a two-way associative cache with a size of 16 KB, while the L2 cache is four-way associative with a size of 512 KB. The local memory access latency is specified as 100 CPU cycles and remote access latency 1,000 cycles, based on the configuration of modern commodity processors. For the following tests, we use the most strict communication policy, which allows only one node to access one remote memory at the same time. During this delay no other nodes can perform remote accesses.

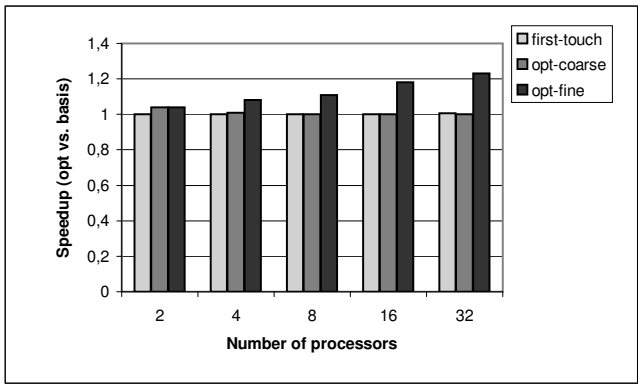

Figure 3. Improvement in execution time with the LU application

Figure 3 to 7 depict the experimental results with different applications. The x-axis of these figures shows the number of processors we used for the experiments, while the y-axis depicts the speedup of the optimized code version against the transparent version in absolute execution time. The figures show the data with the two optimization schemes and the allocation policy first-touch.

Observing the result with LU, as illustrated in Figure 3, it can be seen that the first-touch scheme has no speedup to the basic policy, where the y-axis shows a value of 1 in speedup for all tests meaning that this scheme results in the same execution time as the default version. This also indicates that first-touch brings the same runtime data layout as the host-node scheme that allocates all data sets on the same node, i.e., the host node. The reason may lie on the data initialization, which is usually done by the host and the first-touch scheme hence puts all shared data on the host node that first accesses the data. The figure also depicts that the node-level optimization has a similar behavior, where only on two-processor systems a slight speedup in execution time is observed. Page-level optimization, on the other hand, introduces a speedup with the LU application and the speedup arises with the number of processors. For example, a speedup of factor 1.04 is achieved with two-processor systems, while on 32 processors a factor of 1.23 is obtained. This renders that the page-level optimization achieves a good scalability with LU.

FFT shows a different behavior. As illustrated in Figure 4, all three data allocation policies perform better than the conventional data placement. Nevertheless, both first-touch scheme and coarse-grained optimization present a poor scalability because the improvement goes down as the number of processors increases. The same behavior can also be seen with the fine-grained page-level optimization on small systems. However, using page-level optimization the performance achievement arises and goes up starting with 16-node

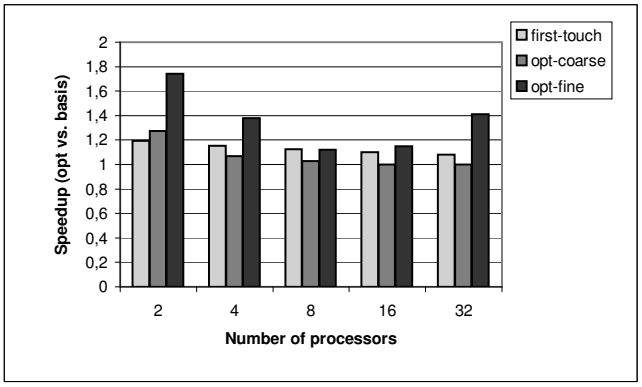

Figure 4. Improvement in execution time with the FFT application

systems. The speedup on a 32-node system, for example, is still lower than that on the two-node system but higher than the case with four-node systems. Further observation of the trend on larger systems would be interesting. Unfortunately, the simulation platform is limited to 32 processors.

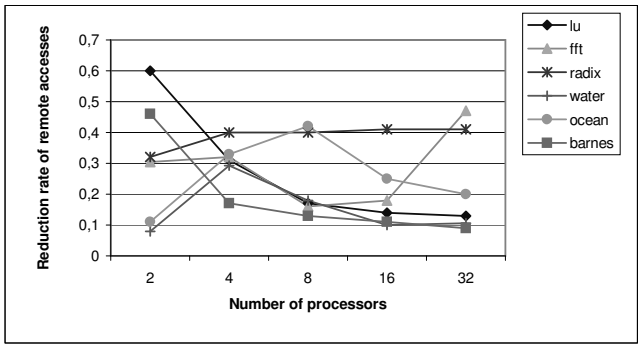

Figure 5. Reduction rate of remote accesses on different system scales

The improved runtime behavior with page-level optimization directly contributes to the better data locality. For a deeper insight into this issue we have measured the number of remote accesses for both page-level optimization and the transparent data placement. We then computed the reduction rate which is defined as the percentage of reduced remote accesses achieved by the optimized version to the total remote accesses introduced by the transparent version. Figure 5 presents the results with all tested applications running on systems with 2 to 32 processing nodes.

Observing the curves in Figure 5, it can be seen that for most applications the reduction rate is higher on smaller systems than on larger ones, for example, LU, WATER, OCEAN, and BARNES. 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 which 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 of the others. Nevertheless, if an application has a lower shared degree, a 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 high as 47% in remote memory accesses on 32-processor systems, while RADIX shows a constant reduction rate of 40% from 4 to 32 processors.

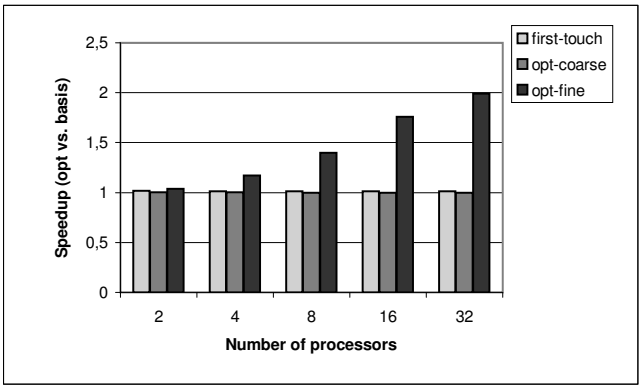

Figure 6. Improvement in execution time with the RADIX application

The high reduction rate of RADIX directly results in the performance gain in terms of execution time. As demonstrated in Figure 6, page-level locality optimization introduces an exciting speedup in execution time and this improvement increases drastically with the number of processors. On a 32-node system, for instance, we achieved a speedup of as high as a factor of two. The scalability achieved with RADIX has to be contributed by the reduction in remote memory accesses. As shown in the previous figure, the reduction rate with RADIX is similar with systems of different scales, which also means a similar reduction in the remote access penalty. In this case, the speedup on larger systems is higher because the execution time is smaller.

The application WATER, however, does not present good results with the optimization. As shown in Figure 7, even the page-level locality tuning is not effective. This is caused by its specific access pattern which will be explained later.

In summary, the applications demonstrate different behavior with our locality optimization. This distinction lies directly 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 is accessed equally by many processors, while the other half has several dominating nodes. Only 6% of the pages is 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

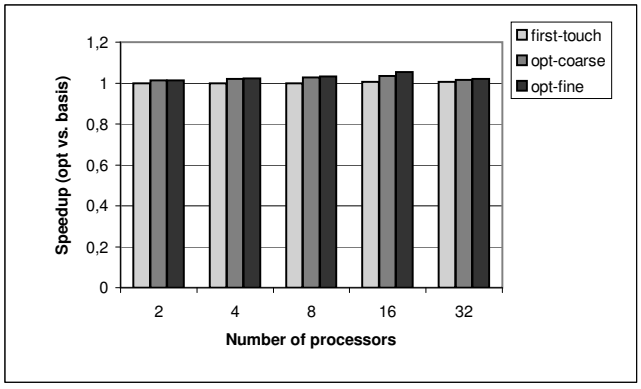

Figure 7. Improvement in execution time with the WATER application

remote accesses exist after the location tuning. Similarly for WATER, although 36% of the data pages is 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 perform equally accesses to an individual page. For BARNES the whole shared data is 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, nevertheless, 96% of the data pages has 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 has a clear dominating processor. Therefore, the optimization with these three applications leads to a considerable performance gain.

As observed in the figures the coarse-grained optimization generally does not work well. The reason is as following. The global dominating node for the whole data set is discovered based on the total remote accesses each node performed at the runtime. Actually, most pages are shared by many processors. The page-level optimization improves the 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 improve the data locality in smaller systems. For example, on a two-node machine, there are only two candidates for a page; hence, placing all data on the global dominating node can achieve speedup. However, on larger systems the best position for a virtual page can be any of the processors; therefore, only a fine tuning with the ability of placing each page on a specific node is able to improve the memory access behavior.

V. Conclusions

This paper describes our research work of applying generic data analysis techniques for memory locality optimization on systems with a distributed memory. The goal of this work is to validate the feasibility of the data classification algorithms in the task of detecting performance bottlenecks and proposing an efficient solution. This is actually the first step towards our final target of establishing self-optimizing parallel systems with respect to different interacting performance metrics. For this research goal we deploy data clustering to evaluate the runtime performance data because this technique provides the required properties such as analysis accuracy, overhead, and association of multiple optimization targets. The proposed approach has shown its ability in finding access patterns and bottlenecks. The data analysis results have guided us to achieve significant improvement in both execution time and scalability of parallel programs.

With this initial experience, we will further address other performance metrics, like the cache performance, processor utility, energy consumption, etc. A more exciting future work is to optimize the system with all these metrics taken into account.

Acknowledgements

Dr. Lizhe Wang's work is supported by the National Natural Science Foundation of China (61361120098).

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 *Proceedings of the 6th IASTED International Conference on European Power and Energy Systems (EuroPES)*, 2006.

[2] D. H. Bailey. FFTs in External or Hierarchical Memory. *Journal of Supercomputing*, 4(1):23–35, March 1990.

[3] N. Bessis, S. Sotiriadis V., Cristea, and F. Pop. Modelling Requirements for Enabling Meta-scheduling in Inter-Clouds and Inter-Enterprises. In *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, pages 149–156, 2011.

[4] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. Zagha. A Comparison of Sorting Algorithms for the Connection Machine CM-2. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 3–16, July 1991.

[5] Christian Borgelt. A Decision Tree Plug-In for DataEngine. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, volume 2, pages 1299–1303, Aachen, Germany, 1998. Verlag Mainz.

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

[7] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. Portable Programming Interface for Performance Evaluation on Modern Processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, 2000.

[8] H. Brunst, D. Hackenberg, G. Juckeland, and H. Rohling. Comprehensive Performance Tracking with Vampir 7. In M.S. Mller, M.M. Resch, A. Schulz, and W.E. Nagel, editors, *Tools for High Performance Computing*, pages 17–29. Springer, 2009.

[9] B. R. Buck and J. K. Hollingsworth. Data Centric Cache Measurement on the Intel Itanium 2 Processor. In *Proceedings of SuperComputing*, November 2004.

[10] CERN. LHC – The Large Hadron Collider. Web Page. http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/.

[11] The LSST Corporation. Large Synoptic Survey Telescope. Web Page. http://www.lsst.org/lsst/.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[13] C. M. Dinu, F. Pop, and V. Cristea. Pattern Detection Model for Monitoring Distributed Systems. In *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 268–275, 2011.

[14] Gordon at san diego supercomputing center. http://www.sdsc.edu/us/resources/gordon/.

[15] 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 and Computation: Practice and Experience*, 24(2):135–154, 2012.

[16] M. Hermanns, S. Krishnamoorthy, and F. Wolf. A scalable infrastructure for the performance analysis of passive target synchronization. *Parallel Computing*, 39(3):132–145, March 2013.

[17] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[18] Intel. Intel VTune Amplifier XE 2013: Performance and Thread Profiler. http://software.intel.com/en-us/intel-vtune-amplifier-xe.

[19] A. Kusiak and Z. Song. Combustion Efficiency Optimization and Virtual Testing: A Data-Mining Approach. *IEEE Transactions on Industrial Informatics*, 2(3):167–184, August 2006.

[20] Sylvain Letourneau, Fazel Famili, and Stan Matwin. Data Mining to Predict Aircraft Component Replacement. *IEEE Intelligent Systems*, 14(6):59–66, 1999.

[21] H. Löf, M. Nordén, and S. Holmgren. Improving Geographical Locality of Data for Shared Memory Implementations of PDE Solvers. In *Computational Science - ICCS 2004*, volume 3037 of *Lecture Notes in Computer Science*, pages 9–16, 2004.

[22] A. D. Malony, S. Biersdorff, S. Shende, H. Jagode, S. Tomov, G. Juckeland, R. Dietrich, D. Poole, and C. Lamb. Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs. In *Proceedings of International Conference on Parallel Processing*, pages 176–185, September 2011.

[23] J. Marathe, F. Mueller, and B. de Supinski. "A Hybrid Hardware/Software Approach to Efficiently Determine Cache Coherence Bottlenecks. In *Proceedings of the International Conference on Supercomputing*, pages 21–30, June 2005.

[24] C. Olaru, P. Geurts, and L. Wehenkel. Data mining tools and applications in power system engineering. In *Proceedings of the Power Systems Computation Conference (PSCC)*, 1999.

[25] B. Quaing, J. Tao, and W. Karl. YACO: A User Conducted Visualization Tool for Supporting Cache Optimization. In *High Performance Computing and Communcations: First International Conference, HPCC 2005. Proceedings*, volume 3726 of Lecture Notes in Computer Science, pages 694–703, Sorrento, Italy, September 2005.

[26] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1993.

[27] S. Shende, A. Malony, W. Spear, and K. Schuchardt. Characterizing I/O Performance Using the TAU Performance System. In *Proceedings of the ICPP Parco 2011 conference Exascale Mini-symposium*.

[28] S. Shende and A. D. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[29] H. Takashi, O. Hiroshi, I. Takayoshi, and D. Henry. Automatic Data Distribution Method Using First Touch Control for Distributed Shared Memory Multiprocessors. In *Languages and compilers for parallel computing. International workshop*, volume 2624 of *Lecture Notes in Computer Science*, pages 147–161, 2001.

[30] J. Tao, M. Schulz, and W. Karl. A Simulation Tool for Evaluating Shared Memory Systems. In *Proceedings of the 36th Annual Simulation Symposium*, pages 335–342, Orlando, Florida, April 2003.

[31] M. M. Tikir and J. K. Hollingsworth. Using Hardware Counters to Automatically Improve Memory Performance. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, 2004.

[32] F. Wolf. Scalasca. In *Encyclopedia of Parallel Computing*, pages 1775–1785. Springer, October 2011.

[33] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[34] J. Zhao, J. Tao, L. Wang, and A. Wirooks. A Toolchain For Profiling Virtual Machines. In *Proceedings of the 27th European Conference on Modelling and Simulation (ECMS 2013)*, pages 497–503, Aalesund, Norway, May 2013.