

Understanding the Behavior of Shared Memory Applications Using the SMiLE Monitoring Framework

Jie Tao, Wolfgang Karl, and Martin Schulz

Abstract—

Data locality is a key factor for the performance of parallel systems. In a Distributed Shared Memory (DSM) system, however, it is difficult for the users to maintain a high data locality as it is usually a priori unknown how the data is distributed among the nodes. In this article we introduce a monitoring framework that allows users to understand the memory behavior of parallel applications. The information offered by the monitor system enables data to be allocated or redistributed more rationally among memories. This reduces remote memory accesses and further improves parallel performance.

Keywords—SCI, monitoring, data locality, performance optimization.

I. MOTIVATION

The performance of a parallel system depends mainly on three facts: (i) whether the parallelism within a program is explored to a maximum. It is always an important task of parallel development to explore parallelism of a program to the extreme so that a multiprocessor or a cluster system has a high utilization, no matter how the parallelism is presented—e.g., indicated explicitly by the programmer [18], recognized by a parallel compiler [4], or distinguished automatically by tools [5]. (ii) whether the tasks are evenly allocated among processors. Load balancing is also one of the key factors that affect system performance. The motivation of load balancing is to reduce the average completion time of processes and improve the utilization of processors. In scenarios where some processors are overloaded while others are underloaded or even idle the speedup of a parallel execution will not be high due to this unbalanced load distribution. Many processor assignment and task allocation systems with load-balancing properties have been therefore implemented [15], [6], [7], [2]. (iii) whether the shared data used by all parallel threads is rationally distributed among memories of processors. If a thread accesses a remote memory very frequently the proportion of communication overhead to the execution time will be relatively high so that the parallel performance is worsened. This case is particularly distinct in DSM characterized systems with NUMA character, as they are already evident in the SMiLE¹ project at LRR-TUM.

Jie Tao is a staff member of Jilin University of China and is currently pursuing her ph.D at Munich University of Technology of Germany. Wolfgang Karl and Martin Schulz: LRR-TUM, Institut für Informatik, Technische Universität München, D-80290 München, Germany. E-mail: {tao, karlw, schulzm}@in.tum.de

¹Shared Memory in a LAN-like Environment

The SMiLE project [9] investigates high performance cluster computing. As the limited communication performance over standard LANs restricts performance of parallel applications, the high-speed low-latency Scalable Coherent Interface (SCI) [1], [8] is taken as the interconnection technology. SCI supports memory-oriented transactions over a ringlet-based network and enables a hardware-supported distributed shared memory (DSM) system on top of a cluster of PCs.

With hardware-based DSM and high performance communication characteristics, the SMiLE cluster is well suited for High Performance Computing (HPC). It is noticed, however, that some parallel applications don't run efficiently upon the SMiLE cluster due to the high overhead associated with too many remote memory accesses [17]. Although remote memory access via hardware-supported DSM delivers high communication performance, they are still an order of magnitude more expensive than local ones. To compensate, we are designing an adaptive run-time system which focuses on improving the locality of memory references made by parallel threads and aims at minimizing remote memory accesses. To know the location of data, however, is not a simple task, especially for DSM systems. In such a system users usually do not know on which nodes and pages the data is located within the global virtual memory. This is due to the fact that data is by-default distributed randomly (explicit allocation is of course possible) in the global memory abstraction. Hence, to understand memory behavior of parallel programs a monitoring system is needed. The SMiLE hardware monitor is designed specifically for this purpose. It provides the programmer or system software with detailed information about all memory transactions over the network and the behavior of user-defined events. This information allows the user or system developer to partition or redistribute data more precisely knowing the effect of lower inefficiency caused by remote memory accesses.

The remainder of this paper is structured as follows: first the SMiLE hardware monitor and its simulation system will be introduced, followed by the first results of tests using the simulation system. The possible measures contributing to the improvement of data locality are then described. In section 5 some related works are discussed. Finally, we conclude with a short summary of our work.

had to be implemented. This system includes not only a simulator of the hardware monitor but also a simulator of the global shared memory and cache memory which is used to produce the packets that would be transferred if the application ran truly on our SCI-Cluster. The packets are stored in a trace file, which is used to stimulate the hardware monitor simulation.

B. Simulation of SCI-VM and Cache Memory

SCI Virtual Memory (SCI-VM) [16], [8] is a concept for the implementation of a global virtual memory using hybrid hardware/software DSM. This concept can be applied in a fully transparent manner allowing the execution of shared memory applications on a global virtual memory without any changes. The memory is distributed at page granularity and remote pages are mapped from other nodes using SCI's HW-DSM mechanisms. The basic address entity for these mapping is the SCI physical address space comprising all physical memory within the cluster. From there, pages can be mapped into the PCI address spaces, and further into processes' virtual address spaces. There they are merged with the local pages forming a global virtual address space, the SCI Virtual Memory.

On top of the SCI-VM several shared memory programming models including Pthreads, Win32threads, and SPMD [8] have been or are currently being implemented. In these models the shared data of an application is partitioned into 4096 bytes pages and usually distributed transparently among all nodes in a round-robin fashion. Some programming models allow an optional direct placement of some data structures onto specific nodes while the remainder of the data is allocated transparently. As the purpose of this work is to understand the memory behavior of true shared memory programs this option is not used in our simulation system.

By default, remotely mapped memory segments or pages are not cached, as current mainstream SCI implementations do not feature a cache consistency protocol³. Enabling caching without precautions is therefore dangerous, as the caches of the cluster nodes will get into an inconsistent state. On the other hand, disabling caching causes severe performance problems, as remote reads are still one to two orders of magnitude slower compared to local reads from main memory, despite the low-latency hardware support of the SCI network adapter [17]. To compensate for this performance problem, the user is given the option to enable caching for remote SCI memory. The consistency problem is tolerated by applying a relaxed consistency model based on synchronization points and then enforced at these points by flushing the caches on the node requiring a consistent state of the memory system.

All of this can be simulated in an execution-driven fashion. However, as the caching behavior has severe impacts on the generated network traffic, a cache simulator has to be included

³The SCI standard includes an optional cache coherence protocol. This, however, cannot be applied to PCI-SCI adapters due to the inconsistent nature of the PCI bus.

in the overall simulation system. This cache simulator collects the request description from the memory simulator. It then checks its line tags and states and determines whether the requested line is present. In the case of a read hit, the LRU information is modified and a satisfied signal is returned. In the case of a read miss, the state is changed to "fetching the line from memory" and the whole line is fetched. In addition, the LRU information is checked, a line is replaced⁴, and a read miss signal is sent back. In the case of a write, the data will be written directly to memory (and also to the cache by a write hit) regardless of the line state. All caches on other nodes are instructed to invalidate their own copy of the line, if they have one.

In order to produce a trace file that includes all memory transactions and that can be used as the basis for the monitor simulator, Limes [11], a multiprocessor simulation tool for PC platform, is applied in our simulation system. Limes creates an illusion of N connected processors executing a parallel application, on a single-processor machine. We have modified the kernel and the memory simulation parts of Limes to enable the simulation of the SCI-VM system. Moreover, a packet construction mechanism adhering to the SCI standard has been added. Whenever a memory access request to shared data is created by the kernel of Limes, the cache simulator is called first. If the cache simulator returns a miss signal, the location of the requested memory cell is determined. Should the address be remote, a packet is added to the trace file. A packet comprises a source ID, a destination ID, a transaction type and a memory description (SCI-VM address, page number and page offset).

C. Simulation of the Hardware Monitor

The simulation of the hardware monitor includes a simulator, a driver and a user interface. The simulator handles every packet in the trace file provided by the memory simulator. The abstracted information from a packet is first compared with the data in the *associative counter array*. If there is a matching entry, the corresponding counter is incremented, otherwise a new tag is created. In case no more space is available within the counter array, the least recently used counter-tag pair is flushed to the ring buffer, and the write-pointer of the buffer is incremented. The information in the packet is then compared with the *page table* and the *event station* for the static mode to determine whether the packet corresponds to an user-defined event. The *static counter array* is then modified accordingly. The detailed operation is as follows: the page number and node number in the *page table* are checked first and an index is acquired if a matching entry in the *page table* exists. Based on this index, the *event station* is then searched to see if the carried information by the packet satisfies one of its entries. In this case, the *count*, the *enable*, and the *disable* fields in the

⁴The simulation system actually does not store the cache contents; only the cache tags are maintained.

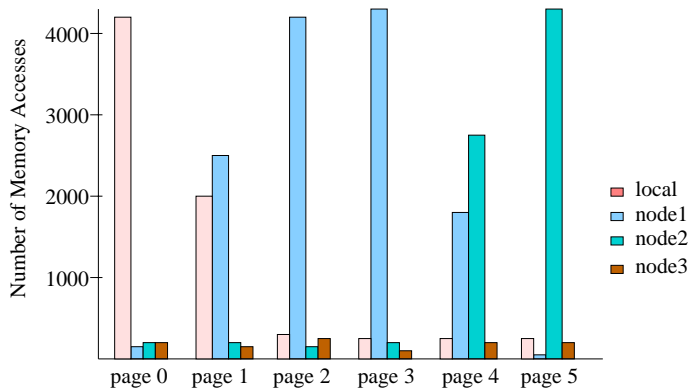


Fig. 3. Memory accesses cases of a few pages of the RADIX program

page \ node	0	1	2	3	4	5	6	7	8	9
local	64	0	0	16.7	7.9	1.9	1.4	19	35.5	100
1	12.2	0	0	74.2	1.1	94.8	0.5	21.7	0	0
2	12.3	0	0	3.6	1.6	1.5	0.9	27.9	64.5	0
3	11.5	0	0	5.5	89.4	1.8	97.2	31.5	0	0

Fig. 4. Percentage(%) of local and remote accesses(FFT)

static counter array are examined one after another. Depending on the result of the examination, a counter is incremented, a new counter is initiated or an old one is concluded. The value of a counter is diverted to the ring buffer if it overflows.

The driver of the monitor simulator aims at offering the programmer with a user library which can be used to initialize counters, to define events, to read data from counters or ring buffer, etc. The user should periodically remove the data from the ring buffer in order to avoid the loss of data in the case of a buffer overflow. The user interface provides several functions with which the statistics for memory references to a page or a special range of a page can be generated. The statistics (number of local and remote accesses) about the following items are offered by the interface:

1. a page in a node
2. a certain range of a memory
3. all pages in a node
4. the first N most frequently accessed pages
5. the first N most frequently accessed memory cells

Now let us examine some preliminary results.

III. MEMORY LOCALITY EXPERIMENTS USING THE SIMULATION SYSTEM

To understand the memory behavior of parallel programs in detail, we have tested a few applications from SPLASH2-Benchmark suite [19] using the SMiLE simulation system. The results can be seen in Fig.3,4,5,6.

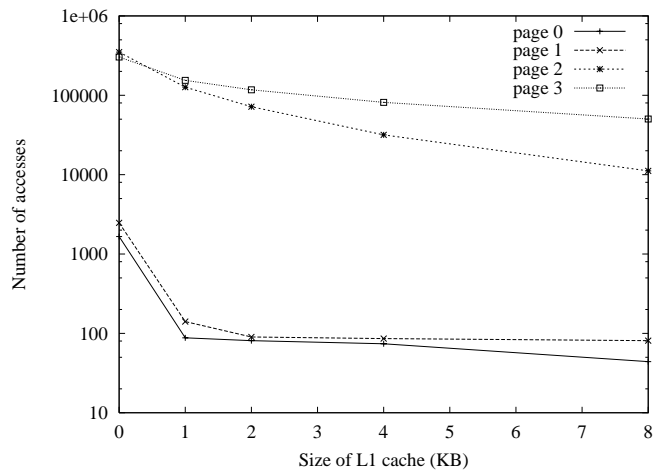


Fig. 5. Comparison of remote memory accesses with different cache sizes(RADIX)

Fig.3 gives the access cases of a few pages of the RADIX program. The results indicate that the access pattern changes from page to page. Some pages are accessed much more frequently by the local processor while others are accessed more often remotely. There are also pages which are frequently accessed by several processors. Hence, it is insufficient to reduce the analysis of an application to a single memory access pattern. In order to fully analyze and understand the behavior of an application, global and fine grain tool environments, like the SMiLE monitoring approach, are needed. Only this will enable high layers to implement a successful optimization strategy.

Fig.4 shows the memory behavior of every page in node 0 of the FFT Kernel. It is clear that for most pages remote accesses take a great proportion of the total memory accesses. For page 5, for example, the memory access from node 1 is 94.8% while the local proportion is only 1.9%. From the test results of the FFT Kernel we have noticed that among the total count of 159229 memory transactions the remote proportion was 114755 transactions, i.e. 72.1%. As the most time is spent on the communication the speedup of FFT on our SMiLE cluster is very low when applied without any optimization. Such information forms the basis for better distribution mechanisms.

Fig.5 and Fig.6 illustrate a general picture of memory behaviors of the LU and the RADIX application with and without caches. Both of them show that remote accesses are decreased with increased cache capacity. A significant reduction, however, is only visible when comparing cached and uncached memory. Remote accesses are therefore for some cases still a great percentage, even if a large cache is installed. For the LU program, for example, the remote access to shared data accounts for 44.5% of the total memory references made by the program even when the cache size is 512KB.

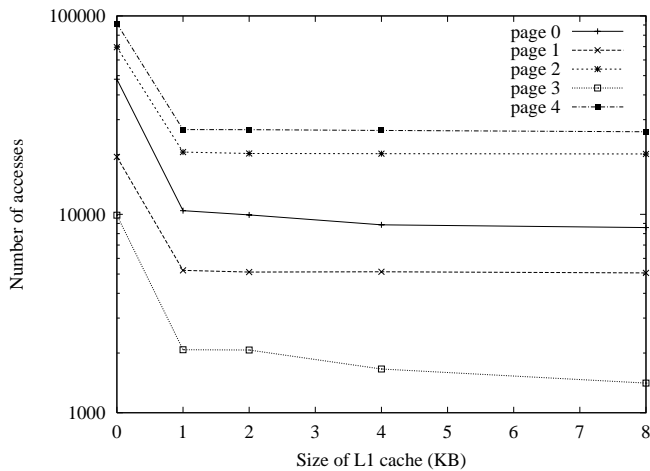


Fig. 6. Comparison of remote memory accesses with different cache sizes(LU)

IV. DESIGNING A TOOL ENVIRONMENT ON TOP OF THE SMILE MONITORING APPROACH

The experiments above clearly imply that the parallel performance would be better when data is more rationally distributed among processors. There are two principal ways to achieve this goal. The improvement of data locality can be made both by the programmer and by the system. In the first case the data locality is improved before an execution, i.e. the data is already placed in an appropriate processor at allocation time. The SCI-VM offers a programmer with the possibility of indicating the location of the data explicitly in the application. The second case can be implemented with the help of a transparent adaptive run-time system. This adaptive system collects and analyzes the information about the low-level memory behavior and redistributes data during the execution of a program. Data will be migrated transparently according to the decisions of the adaptive system. Although the decision can only be made after a certain analysis time the performance will be correspondingly improved for programs which have a long running time.

In the case when the programmer is responsible for the locality optimization a visualizer is needed to show the monitoring information to the programmer vividly. This kind of off-line tool offers the programmer with well understandable statistics of different memory access cases as well as an analysis of the application's memory behavior. It is expected that with help of visualizers, the programmer can make a correct judgment on the placement of the data.

The adaptive run-time system consists of a decision-making and a data- and threads-migration subsystem. The decision subsystem analyzes all memory transactions with the goal to recognize which part of the global memory is accessed very frequently and which part very rarely, and by which threads. According to this analysis a decision is made on which thread or page is supposed to be migrated if at all. The migration sub-

system will then automatically and transparently migrate data and threads at run-time with the help of appropriate mechanisms in the underlying layers. In this case the communication among processors will be minimized.

In some cases thread migration is necessary. From the test results described in chapter 3 we notice that the memory behavior of different pages can differ within a single application. There are pages that are accessed frequently by only one thread (local or remote) and there are pages that are frequently accessed by more threads. A page that is accessed frequently by only one remote thread should be migrated to the local memory of the thread. A page that is accessed frequently both by a local thread and a remote one, however, should not be migrated and it should be taken into consideration whether the thread should be moved to the node in which the page lies. In case a page is accessed often by more threads (local and remote) an additional tradeoff can be made between the option of holding it in a single node or replicated on every node.

The visualizer and the adaptive run-time system described above are currently being developed within the SMiLE project. A mechanism for mapping each memory location observed by the monitoring system to its corresponding program data structure identifier (procedure and variable names) is being implemented as well. This mechanism aims at giving the programmer a visualization of the program's memory behavior in familiar terms. Together with the monitoring data, the mapping mechanism helps the programmer optimize their programs so as to reduce the effect of remote memory accesses on performance. For the FFT Kernel, for example, the ratio of remote accesses to the total memory references would be decreased from 72.1% to 19.7%, if the data are distributed rationally among processors, e.g., the fifth page on node 0 is located on node 1 instead of node 0, the fourth page on node 1 is located on node 0 instead of node 1, etc.. Thus, the inefficiency caused by remote memory accesses will go down very greatly.

V. RELATED WORK

In a NUMA system, the performance depends greatly on the locality of the data distribution. It is important to know which pages are often accessed remotely so as to improve the locality. Surprisingly few hardware performance monitors have been developed. The hardware performance monitor in Computer System Laboratory in Stanford University [10], [14] was built for the DASH multiprocessor. It is controlled by software and provides low-level information on software reference characteristics and hardware resource utilization. The monitor hardware can trace and count network events. This trace information can be used by high-level protocols for a detailed analysis of the memory access patterns and overall application behaviors.

The SHRIMP hardware monitor [13] can be used to keep watch on all packets arriving at a specific node. In contrast to our hybrid hardware monitor, it has a DRAM in the moni-

tor card to record the complete access histograms and run-time measurements of a running application. Similar to the SMiLE approach these measurements can be used to modify mutable parts of the system hardware, to help with software development and application tuning, and to characterize and evaluate the behavior and performance of a prototype system.

Another trace instrument for SCI-based systems [12] has been designed at Trinity College Dublin. This instrument enables the non-intrusive monitoring of SCI interface traffic. Full traces can be transferred later into a relational database. Users are allowed to extend and adapt the predefined database queries to their specific needs. This lacks, however, the run-time ability and allows no on-line mechanisms, being intended primarily for offline analysis, debugging and validation.

VI. CONCLUSION

Efficient shared memory programming is one of the main motivations of the SMiLE project. In order to achieve this goal, both the low-level technology and the high-level mechanisms have to be taken into consideration. In this paper we discuss principally a monitoring strategy that aims at improving the efficiency of parallel applications through the reduction of remote memory accesses.

As memory references happen frequently, fine-grained monitoring with hardware support is required to understand memory behavior. The SMiLE hardware monitor as part of the event-driven hybrid monitoring approach provides the user with detailed information about low-level memory transactions. Based on the statistics of the observed memory transactions data or threads can be redistributed among processors with an effect of the reduction of remote memory accesses. It is expected that system performance will be thereby improved.

REFERENCES

- [1] IEEE Standard for the Scalable Coherent Interface(SCI). IEEE Std 1596-1992,1993, IEEE 345 East 47th Street, New York, NY 10017-2394, USA.
- [2] Anurag Acharya, Guy Edjlali, and Joel Saltz. The utility of exploiting idle workstations for parallel computation. In *Proceedings of the 1997 ACM SIGMETRICS conference on Measurement & modeling of computer systems*, pages 225–235, Seattle, WA USA, June 1997.
- [3] G. Acher, H. Hellwagner, W. Karl, and M. Leberrecht. A PCI-SCI bridge for building a PC-cluster with distributed shared memory. In *Proceedings of the Sixth International Workshop on SCI-based High-Performance Low-Cost Computing*, pages 1–8, Santa Clara, CA, September 1996.
- [4] Yosi Ben-Asher, Dror G. Feitelson, and Larry Rudolph. Parc,—,An extension of C for shared memory parallel processing. *Journal of Software practice & Experience*, 26(5):581–612, 1996.
- [5] William Blume, Rudolf Eigenmann, Jay Hoeflinger, David Padua, Paul Petersen Lawrence Rauchwerger, and Peng Tu. Automatic detection of parallelism—A grand challenge for High-Performance computing. *IEEE Parallel & Distributed Technology*, pages 37–46, 1994.
- [6] Mark E. Crovella, Mor Harchol-Balter, and Cristina D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In *Proceedings of the joint international conference on measurement and modeling of computer systems*, pages 268–269, Madison, WI USA, June 1998.
- [7] Mor Harchol-Balter and Allen B. Downey. Exploiting process lifetime distribution dor dynamic load balancing. In *Proceedings of the ACM SIGMETRICS conference on Measurement & modeling of computer systems*, pages 13–24, Philadelphia, PA USA, May 1996.
- [8] Hermann Hellwagner and Alexander Reinefeld, editors. *SCI: Scalable Coherent Interface: Architecture and Software for High-Performance Computer Clusters*, volume 1734 of Lecture Notes in Computer Science. Springer-Verlag, 1999.
- [9] Wolfgang Karl, Markus Leberrecht, and Martin Schulz. Supporting shared memory and message passing on clusters of PCs with a SMiLE. In *Proceedings of the third International Workshop, CANPC'99*, volume 1602 of Lecture Notes in Computer Science, Orlando, Florida, USA(together with HPCA-5), January 1999. Springer Verlag, Heidelberg.
- [10] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The dash prototype: Logic overhead and performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, January 1993.
- [11] D. Magdic. Limes: an execution-driven multiprocessor simulation tool for the i486+-based PCs. School of Electrical Engineering, Department of Computer Engineering, University of Belgrade, POB 816 11000 Belgrade, Serbia, Yugoslavia, 1997.
- [12] Michael Manzke and Brian Coghlan. Non-intrusive deep tracing of SCI interconnect traffic. In *Conference Proceedings of SCI Europe'99*, pages 53–58, Toulouse, France, September 1999.
- [13] M. Martonosi, D. W. Clark, and M. Mesarina. The SHRIMP performance monitor: Design and applications. In *Proc. SIGMETRICS Symposium on Parallel and Distributed Tools*, pages 61–69, Philadelphia, May 1996.
- [14] Margaret Martonosi, David Ofelt, and Mark Heinrich. Integrating performance monitoring and communication in parallel computers. In *Proceedings of the ACM SIGMETRICS conference on Measurement & modeling of computer systems*, pages 138–147, Philadelphia, PA USA, May 1996.
- [15] Oscar Plata and Francisco F. Rivera. Combining static and dynamic scheduling on distributed-memory multiprocessors. In *Proceedings of the 8th conference on ACM international conference on supercomputing*, pages 186–195, Manchester, England, July 1994.
- [16] M. Schulz. SCI-VM: A flexible base for transparent shared memory programming models on clusters of PCs. In *Proceedings of HIPS'99*, volume 1586 of Lecture Notes in Computer Science, pages 19–33, Berlin, April 1999. Springer Verlag.
- [17] Martin Schulz and Hermann Hellwagner. Global virtual memory based on SCI-DSM. In *Proceedings of SCI-Europe '98*, pages 59–67, Bordeaux, France, September 1998. Cheshire Henbury.
- [18] V. Sunderam, J. Dongarra, A. Geist, and R Manchek. The PVM concurrent computing system: Evolution, experiences, and trends. *Parallel Computing*, 20(4):531–547, April 1994.
- [19] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop 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.