

Performance Evaluation of SPEC OMP Benchmarks on the InfiniBand-Based Cluster System*

Shinyoung Kim Seo Hee Inho Park Seon Wook Kim
Department of Electronics and Computer Engineering
Korea University, Seoul, Korea
<http://compiler.korea.ac.kr>

Abstract

Recently many advanced hardware-based interconnection schemes to connect physically distributed processing nodes have been developed to overcome performance limitation of networks, and one of them is the InfiniBand Architecture (IBA) which supports shared-memory programming semantics by means of remote direct-memory access (RDMA) and atomic operations. In this paper, we evaluate performance of SPEC OMP benchmarks on the InfiniBand-based Distributed Virtual Shared-Memory (DVSM) system in detail. Our research helps the system and application developers to estimate the applicability of the OpenMP applications on a commodity cluster of workstations to use the state-of-the-art interconnection networks for the next decades.

1. Introduction

In order to provide easy programming environment on the distributed-memory system, many distributed virtual shared-memory (DVSM) systems have been introduced [1, 8, 15]. The DVSM systems allow processes to access physically distributed memory spaces through one software-enabled virtual shared memory space, and therefore a programmer is able to use the shared-memory parallel programming APIs on the system, such as OpenMP [13] and PThread. However, the performance of applications on the DVSM system, especially when executing parallel applications, heavily depends on the network performance and network programming semantics because data coherence mechanism across processes on the software-enabled virtual shared memory space is supported only by software over the network.

In the previously designed DVSM systems [1, 8, 15] a process should communicate directly with other processes in order to exchange data through process interruption as shown in Figure 1 (a). Obviously this approach results in large execution overhead due to signal handling for process interruption and communication latency, and it degrades the total system performance significantly. Differently from the previously used schemes, on the InfiniBand-based DVSM system a process does not need to interrupt other processes in order to access data on remote nodes because the InfiniBand architecture provides hardware legacy software protocol tasks to support Remote Direct Memory Access (RDMA) and atomic operations (Figure 1 (b)).

The InfiniBand Trade Association [4] run by Intel, IBM, Dell Computer, and so on, has proposed the InfiniBand Architecture (IBA) in 1999, which consists of processing nodes, I/O nodes, and System Area Network (SAN) to connect nodes. Figure 2 gives an overview of the IBA. The distinguished features in the IBA are using a channel based interconnection switched fabric, and supporting Remote Direct Memory Access (RDMA) and atomic operations (i.e. compare and swap, fetch and add). The switched fabric in the IBA guarantees high stability and wider bandwidth. The IBA can be used in constructing from a small size cluster which has one processor and a few I/O device, to a large-scale parallel supercomputer consisting of hundreds of processors and thousands of I/O devices. The detailed description about the InfiniBand architecture can be found in [4].

In this paper, we evaluate performance of SPEC OMP benchmarks on our designed InfiniBand-based Distributed Virtual Shared-Memory (DVSM) system by comparing with the IPoIB-based traditional system on the IBA, especially TreadMarks [1]. Also we discuss code optimization issues. More specifically, the serializing small parallel code sections, which is the most effective scheme on the hardware SMP, is not applicable on the DVSM system and even makes the performance worse due to increment of data movement between serial and parallel sections. We argue that this paper contribute to help the

* This work was supported by the Electronics and Telecommunications Research Institute (ETRI), Taejeon, Korea.

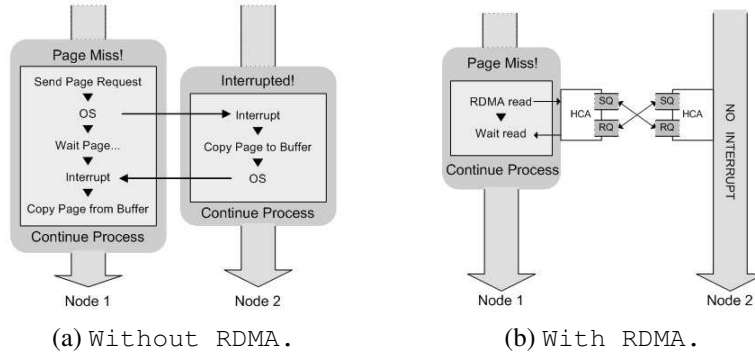


Figure 1. The difference between two schemes to access data on remote nodes.

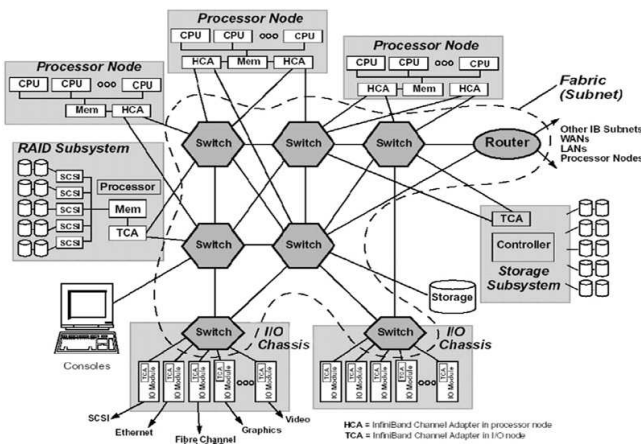


Figure 2. InfiniBand System Area Network [4].

system and application developers to estimate the applicability of OpenMP applications on a commodity cluster of workstations to use the state-of-the-art interconnection networks for the next decades.

The paper is organized as follows. Section 2 briefly presents our IBA-based DVSM implementation [14]. In Section 3 we characterize the performance of SPEC OMP benchmarks in detail, and in Section 4 the conclusion is made.

2. InfiniBand-Based Distributed Virtual Shared-Memory Systems

The data coherence mechanism across processes needs to maintain at least the following three information: modified data and their references per process, data access history (i.e. read and write) per coherence granularity, and access ordering (i.e. time). Our implementation is based on

the lazy release consistency (LRC) model, which allows the protocol to aggregate the transmission of shared memory writes until a later synchronization point in order to minimize data communication between processes [1]. The synchronization between processes provides access ordering information, and *twin* and *diff* mechanisms distinguish between original and modified data, i.e. we know which references are modified [1]. In the TreadMarks system, a process to request data should interrupt a data providing process for communication, and therefore processes naturally know access history at communication time. But because the RDMA feature removes process interruption for data communication, we need a new mechanism to track access history on the InfiniBand-based DVSM system.

In order to record read and write operations per page (our system's granularity), we use an interval table whose entry has two bits to mark read and write operations and whose number of entries is equal to the number of pages in globally allocated virtual shared-memory. When a page segmentation violation occurs, the related bit is set. The interval table is cleared after all necessary diff operations are performed to other processes, since the event history is not needed any more (at this time dependences are completely resolved).

Each synchronization includes two internal barriers. At the first barrier each process broadcasts the interval table to all the participant processes, and at the second barrier the diff operations are applied to other processes if necessary by considering all the other processes' interval tables. It is well known that the overhead associated with the page segmentation violation is very large, and it is one of the most important factors to determine the overall performance. In order to reduce the occurrence of segmentation violations, we assign an ownership to each page. Basically the page owner is not memory-protected, and it is write memory-protected only after other processes perform read and write operations. The page owner information is maintained inside the interval table by extending one bit to mark a page owner-

ship. The details are shown in [14].

There are a few research activities about programming environment on top of the InfiniBand. The research in [6] describes the implementation of the OS-layered DVSM on the IBA. This approach needs to modify kernels in order to integrate the InfiniBand primitives with OS kernels. Our approach is the application-layered InfiniBand-based DVSM, and therefore we do not need any modification inside kernels. And while our framework uses a lazy release consistency, the DVSM in [6] uses a sequential consistency. Also the InfiniBand-based MPI implementation has been done extensively [9].

3. Performance Evaluation

3.1. Experiment Methods

For performance evaluation of SPEC OMP3.51 suite [3] on our IBA-based DVSM system, we used two versions of four benchmarks (*swim*, *mgrid*, *wupwise*, and *applu*). One version is to use officially delivered benchmarks from SPEC (called as unoptimized versions), and another is to parallelize only important loops (their execution time is more than 95% of the total execution in serial codes) in order to achieve the maximum performance by eliminating parallel code overhead (called as optimized versions) [2]. The parallel code overhead means code transformation overhead from OpenMP codes into thread-based forms by the OpenMP preprocessor and thread management overhead. We scaled down the number of iterations in SPEC OMP for short execution (it does not change any program characteristics) and we used *ref* data set.

We implemented the DVSM system on the InfiniBand to use Verbs API in order to use full features of the InfiniBand semantics. We used eight processing nodes connected with the 4X Mellanox InfiniBand, and each node includes a Intel 2.0GHz Xeon processor, 512MB memory, and 133MHz 4X PCI-X 128MB memory HCA board. Also for comparing with the performance on the hardware SMP machine, we measured the SPEC applications on the bus-based SMP to use four 2.8GHz Xeon processors and 512MB memory.

For execution of the OpenMP benchmarks on the DVSM system, we used the Polaris parallelizing compiler infrastructure [5]. The Polaris compiler translates the OpenMP directive codes into thread-based programs like a normal OpenMP preprocessor [12, 16], and links with runtime libraries to manage processes on the DVSM system.

There are the following six measurements for each benchmark: Execution of unoptimized versions on the hardware-based SMP machines (*UNOPT_SMP*), TreadMarks (Version 1.0.3.3) execution of unoptimized versions to use IPoIB protocol stack on the InfiniBand

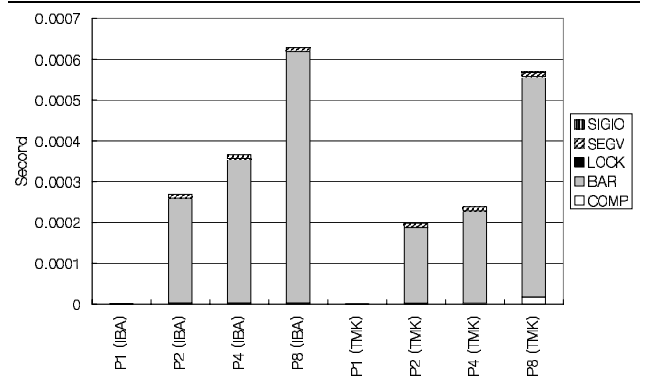


Figure 3. Process fork/join overhead of the OpenMP parallel directive on our system (IBA) and TreadMarks (TMK).

(*UNOPT_TMK*), and execution of unoptimized and optimized versions on our InfiniBand-based DVSM system (*UOPT_IBA* and *OPT_IBA*).

The execution time is categorized into the following performance metrics: *COMP* (execution time of the application itself), *BAR1* (overhead at the first barrier on the InfiniBand implementation or the total barrier time on the TreadMarks system), *BAR2* (overhead at the second barrier on the InfiniBand implementation), *BAR* (sum of *BAR1* and *BAR2*), *LOCK* (lock related overhead), *SEGV* (overhead related to page segmentation violation due to page misses), and *SIGIO* (overhead due to the process interruption from a service request process).

3.2. Performance

3.2.1. Overhead of OpenMP Primitives In order to measure overhead of OpenMP primitives such as *parallel*, *atomic*, and *shared*, we designed a microbenchmark. Figure 3 shows the process fork/join overhead, i.e. the overhead related to *parallel* OpenMP directive on our system and TreadMarks on one to eight processing nodes. The overhead in both measurements proportionally increases with the number of processes, and its value on our system is higher than TreadMarks. The reason of the performance difference is that as mentioned in the previous section, our system maintains an interval table to track access history, and at the synchronization point a process needs to broadcast the interval table to all the other processes for page diff actions. The table size per process is about 6.4KB for 100MB virtually allocated shared-memory space.

Figure 4 shows overhead of OpenMP *atomic* primitive on our system and TreadMarks. The lock overhead on the InfiniBand-based DVSM system is constant because the InfiniBand architecture provides hardware-implemented

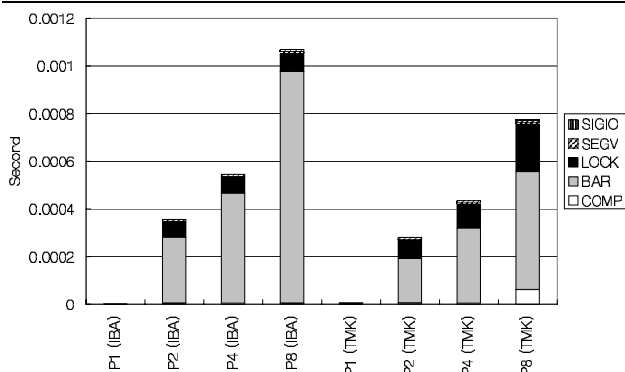


Figure 4. Overhead of OpenMP atomic primitive on our system (IBA) and TreadMarks (TMK).

atomic semantics. On the TreadMarks system, the overhead increases proportionally with the number of processes.

Figure 5 shows overhead of page movement between serial and parallel sections, i.e. related to `shared` data on our system and TreadMarks. In order to measure overhead only related to `shared` data movement we evaluated the overhead occurred by page segmentation violation. The overhead of data movement from parallel to serial and inside parallel sections on our system is much lower than that on TreadMarks, since our system uses RDMA features provided by the InfiniBand architecture. But from serial to parallel sections the overhead on our system is larger than on the TreadMarks system. Our system uses the home-based LRC protocol. In other words, a page owner is write-memory protected after child processes read the data from a master process, i.e. the page owner [7]. It incurs the segmentation violation overhead on a page owner. On the TreadMarks system multiple owners are allowed (homeless LRC protocol), and therefore there are less segmentation violations on a page owner process.

3.2.2. Overall Performance of OMP Benchmarks Figure 6 shows the speedup of the OpenMP benchmarks on 2, 4, and 8 processors with respect to one processor execution. In `applu` and `swim` there is no speedup in the DVSM executions, and in `mgrid` there is speedup at the execution of unoptimized version on our DVSM system, but less than the SMP execution. In `wupwise`, the speedup on both DVSM systems is higher than the SMP execution. The result in the figure is different from that in [11]. The reason of the performance difference is that our processor’s performance is much higher than those in [11], so the communication overhead dominates the overall execution time.

Also the figure shows that the most effective optimization scheme on hardware SMP machines, i.e. serialization

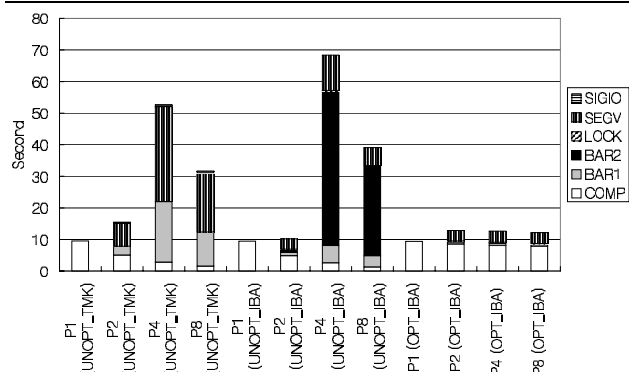


Figure 7. Overhead analysis in APPLU.

of small parallel code sections in order to reduce the parallelization overhead, is not effective on the DVSM system in all benchmarks except for `applu` (in `applu` we do not use the serialization and it will be discussed shortly). We found that the parallel code overhead is not important, and the most important consideration for performance optimization is data (page) locality. More specifically, the serialization of parallel sections incurs more data movement between serial and parallel sections, i.e. between host and child processes. It incurs more page segmentation violations when child processes start parallel region execution to use data updated in serialized parallel sections. It makes programmer optimize codes very difficult, and traditional optimization techniques used on the hardware SMP may be ineffective on the DVSM system.

3.2.3. APPLU Figure 7 shows that the unoptimized version of APPLU has enough parallelism, since `COMP` metrics are well scaled down proportionally with the number of processes. But in this code version, the overhead is very large because the most important loop `ssor_do_2_2` includes the barrier synchronization per iteration. Since we use the LRC protocol, the large number of barriers incurs huge diff operations and eventually degrades the overall performance significantly. The barrier overhead on our system is larger than on the TreadMarks system, and it was discussed in Figure 5 (a). When parallelizing only important loops [2] except for `ssor_do_2_2`, the overhead is significantly degraded. But there is no performance gain, because we parallelized only 18% of the serial code execution.

3.2.4. MGRID Figure 8 shows execution overhead in MGRID. There is a little speedup in unoptimized versions, but the performance is significantly degraded when only parallelizing important loops due to excessive page segmentation violations in serialized `zero30_do_1`, `zran3_do_1`, and `zero3_do_1` code sections. On the hardware SMP machines, we have serialized small parallel sections in order to remove parallelization overhead. But

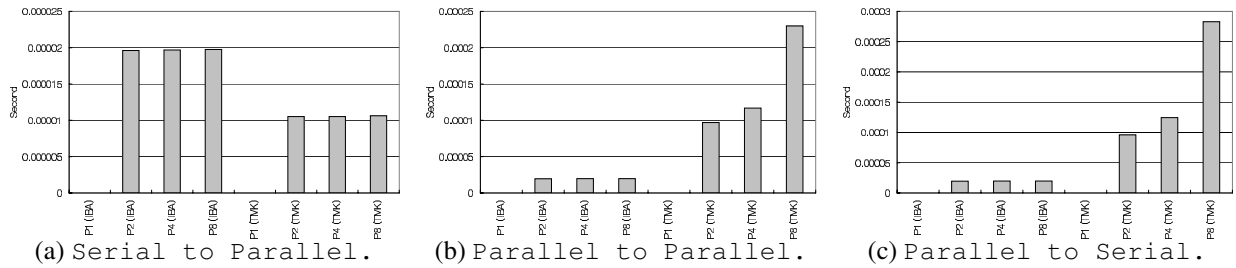


Figure 5. Overhead of data movement between serial and parallel sections on our system (IBA) and TreadMarks (TMK).

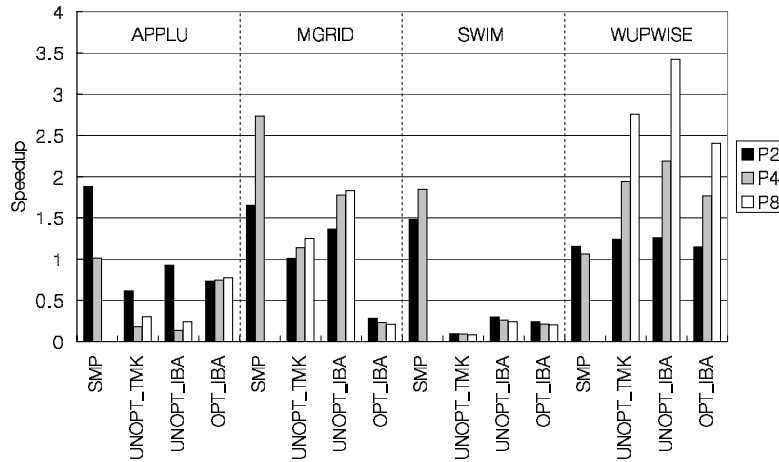


Figure 6. Speedup of the OpenMP applications.

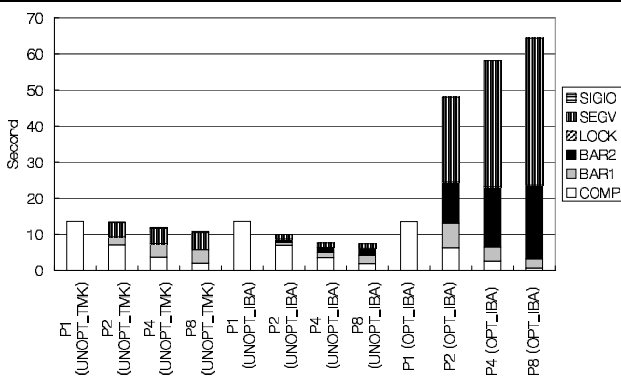


Figure 8. Overhead analysis in MGRID.

most overheads on the DVSM systems result from page locality, and the serialization of parallel regions largely degrades the performance. One of the solutions to alleviate this problem is to execute serial code sections also on child processes [10]

3.2.5. SWIM Figure 9 shows execution overhead in SWIM. Most loops are parallelized in an unoptimized version, but many page faults at the serial sections just after `calc1_do_1` and `calc2_do_1` results in significantly large overhead. The optimized code makes this problem worse.

3.2.6. WUPWISE Figure 10 shows overhead metrics of WUPWISE execution. There is very a few small parallel regions in this benchmark, and there is no noticeable difference in unoptimized and optimized versions. Most of SEGV overhead is generated at the serial section after the `zcopy_do_1` parallel region.

4. Conclusion

In this paper, we evaluated performance of SPEC OMP benchmarks on the next generation network technology based DVSM system, i.e. the InfiniBand Architecture. From experiment we showed that the next-generation of network technologies greatly improve the performance over the tra-

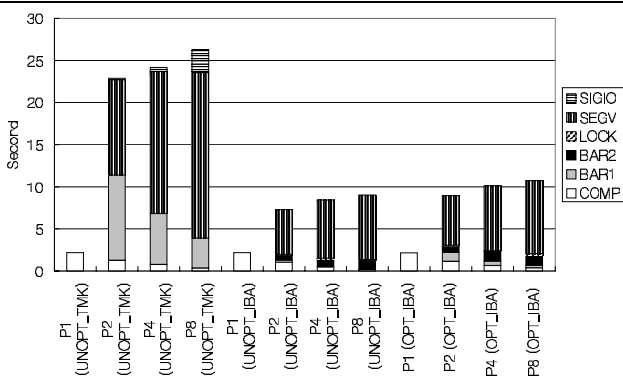


Figure 9. Overhead analysis in SWIM.

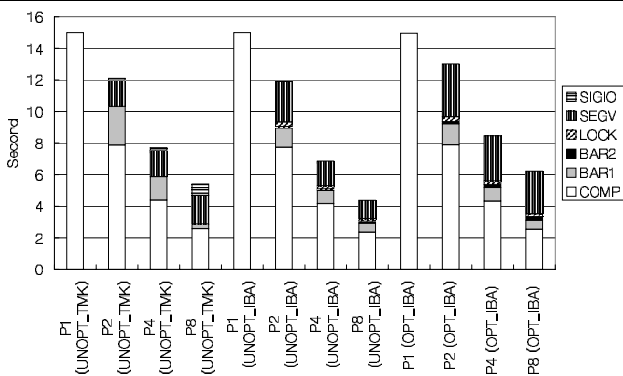


Figure 10. Overhead analysis in WUPWISE.

ditional implementation of DVSM systems. The performance gain results from using the remote DMA features, which does not involve any process interruption for data communication. Also we showed that the serialization of parallel code sections to reduce parallel code overhead results in severe overhead on the DVSM systems due to more data movement between serial and parallel sections. Even if the new network architecture helps to reduce the communication overhead, the application performance is very sensitive to the data locality. Since the standard OpenMP specification does not provide a mechanism to control data distribution, we need more advanced techniques to predict page movement and prefetch pages correctly inside the DVSM for better data locality and hiding latency.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, 1996.

[2] V. Aslot. Performance characterization of the SPEC OMP benchmarks. Master's thesis, Electrical and Computer Engineering, Purdue University, May 2002.

[3] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady. SPECComp: A new benchmark suite for measuring parallel computer performance. *Lecture Notes in Computer Science (WOMPEI2001)*, 2104:1–10, 2001.

[4] I. T. Association. *InfiniBand Architecture Specification, Release 1.0*. October 2000.

[5] A. Basumallik, S. J. Min, and R. Eigenmann. Towards OpenMP execution on software distributed shared memory systems. *Lecture Notes in Computer Science (WOMPEI'2002)*, 2327:457–468, 2002.

[6] T. Birk, L. Liss, and A. Schuster. Efficient exploitation of kernel access to InfiniBand: a software DSM example. In *Hot Interconnects*, Stanford, CA, August 2003.

[7] A. L. Cox, E. de Lara, C. Y. C. Hu, and W. Zwaenepoel. A performance comparison of homeless and home-based lazy release consistency protocols for software shared memory. In *Proc. of the 5th IEEE Symp. on High-Performance Computer Architecture (HPCA-5)*, 1999.

[8] L. Kontothanassis, G. Hunt, R. Stets, N. Hardavellas, M. Cierniak, S. Parthasarthy, W. Meira, S. Dwarkadas, and M. Scott. VM-based shared memory on low-latency, remote-memory access networks. In *The 24th International Symposium on Computer Architecture (ISCA-24)*, June 1997.

[9] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and implementation of MPICH2 over infiniband with RDMA support. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[10] H. Lu, A. L. Cox, and W. Zwaenepoel. Contention elimination by replication of sequential sections in distributed shared memory programs. *ACM SIGPLAN Notices*, 36(7):53–61, 2001.

[11] S. J. Min, A. Basumallik, and R. Eigenmann. Supporting realistic OpenMP applications on a commodity cluster of workstations. *Lecture Notes in Computer Science (WOMPAT2003)*, 2716:170–179, 2003.

[12] S. J. Min, S. W. Kim, M. Voss, S. I. Lee, and R. Eigenmann. Portable compilers for OpenMP. *Lecture Notes in Computer Science*, 2104:11–19, 2001.

[13] OpenMP Forum, <http://www.openmp.org/>. *OpenMP: A Proposed Industry Standard API for Shared Memory Programming*, October 1997.

[14] I. Park, S. W. Kim, and K. Park. Characterization of OpenMP applications on the InfiniBand-based distributed virtual shared memory system. In *Accepted at the International Conference on High Performance Computing*, 2004.

[15] R. Samanta, A. Bilas, L. Iftode, and J. P. Singh. Home-based SVM protocols for SMP clusters: Design, simulations, implementation and performance. In *The Fourth IEEE Symposium on High-Performance Computer Architecture (HPCA-4)*, Jan. 1998.

[16] M. Sato, S. Satoh, K. Kusano, and Y. Tanaka. Design of OpenMP compiler for a SMP cluster. In *The 1st European Workshop on OpenMP (EWOMP'99)*, pages 32–39, September 1999.