Performance Evaluation of View-Oriented Parallel Programming on Cluster of Computers

Haifeng Shang[1], Jiaqi Zhang[1], Wenguang Chen[1], Weimin Zheng[1], and Zhiyi Huang[2]

[1] Institute of High Performance Computing, Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2] Department of Information Science, University of Otago, Dunedin, New Zealand
smickers@gmail.com, zation99@gmail.com, cwg@mail.tsinghua.edu.cn, zwm-dcs@tsinghua.edu.cn, hzy@cs.otago.ac.nz

Abstract. View-Oriented Parallel Programming(VOPP) is a novel programming style based on Distributed Shared Memory, which is friendly and easy for programmers to use. In this paper we compare VOPP with two other systems for parallel programming on clusters: LAM/MPI, a message passing system, and TreadMarks, a software distributed shared memory system. We present results for ten applications implemented and optimized using all the three systems. Experimental results demonstrate that VOPP is almost as efficient as Message Passing Interface when running on up to 32 processors, which means there is significant performance improvement compared with TreadMarks. The factors contributing to the performance of VOPP are discussed and analyzed. VOPP is still slower than MPI when the number of processes is large because of extra messages for separate synchronization and lack of bulk transfer mechanisms.

1 Introduction

Software distributed shared memory systems provide a shared memory abstraction on top of the native message passing facilities. It leaves the chore of message passing to the underlying DSM systems so that it is easier to program. However, DSM systems tend to generate more communication and therefore be less efficient than message passing systems [14].

View-Oriented Parallel Programming(VOPP) [3] is a programming style based on distributed shared memory. Traditional DSM systems like TreadMarks [1] are far from efficient compared with Message Passing Interface [7]. The reason is that programs written in MPI can be finely tuned by reducing unnecessary message passing. As we know, message passing is a significant cost for parallel applications, which is also true for DSM programs. Programmers cannot help reduce messages with traditional DSM systems as consistency maintenance of the underlying systems deals with the whole shared memory space.

We propose a novel VOPP programming style for DSM applications which optimizes DSM performance by introducing a new consistency maintenance protocol

R. Perrott et al. (Eds.): HPCC 2007, LNCS 4782, pp. 120–131, 2007.
© Springer-Verlag Berlin Heidelberg 2007

VOUPID [4] and allows programmers to participate in performance tuning by wise optimization of data allocation. VOPP programs perform much more efficiently than TreadMarks. The performance gain is two-fold. First, the consistency maintenance for views reduces unnecessary messages and unnecessary data. The consistency maintenance protocol VOUPID solves the diff accumulation problem which limits the performance of TreadMarks. Second, VOPP enables programmers to tune the performance by wisely partitioning views.

Our ultimate goal is to make VOPP performs as efficiently as MPI. So far, the performance of VOPP is comparable with MPI and they are almost as efficient as each other while running on less than 32 processors. In this paper, we are going to compare VOPP with TreadMarks and MPI in terms of performance and investigate the factors contributing to the performance of VOPP.

Contributions of this paper include:

1) We use 10 applications for performance evaluation. The result presented in this paper is more convincible than the previous research which only used 4 applications.[6]

2) We perform a detailed performance analysis between VOPP, MPI and TreadMarks and reveals that VOPP could achieve comparable performance with MPI when the number of processes is up to 32. The performance of VOPP is not as good as MPI when the number of processes is larger than 32. Then main reasons are extra messages for separate synchronization and lack of bulk transfer mechanisms.

The rest of the paper is organized as following. In section 2 we briefly describe the VOPP programming style. In section 3 we evaluate the performance of VOPP by comparing it with TreadMarks and MPI. The reasons of the results are also discussed and analyzed. Finally, we conclude our work and summarize the usability and programmability of VOPP.

2 View-Oriented Parallel Programming

In the View-Oriented Parallel Programming(VOPP) style, programmers should divide shared data into views according to the memory access pattern of the parallel algorithm. A view [3] consists of data objects that require consistency maintenance as a whole body. Views are indicated through primitives such as acquire_view and release_view. Acquire_view means acquiring exclusive access to a view, while release_view means finishing the access. In addition, acquire_Rview and release_Rview are provided for read-only accesses. The read-only access primitives can be called in a nested style while other access primitives cannot. By using these primitives, programmers are now focusing on the access of shared data rather than synchronization and mutual exclusion.

Views are defined implicitly by programmers in their mind. Therefore, it is convenient for programmers to use the shared data and optimize the programs by wisely partitioning views. Views cannot overlap each other and they are unchangeable once they're defined. A view can only be accessed when the primitives are used.

Programming in VOPP style, programmers can enjoy the convenience of accessing shared data with almost ordinary read and write operations. Programmers don't have to determine what to communicate as the message passing is left to the underlying system. Furthermore, by dividing shared data into views, VOPP allows programmers to participate in performance optimization.

Much work has been done in the previous papers [3,5,4] to introduce optimizations in VOPP compared with traditional distributed shared memory systems. In the following section, we will describe our recent work on VOPP.

3 Performance Comparison

In this section, we present our experimental results of 10 applications. All applications in our experiment are coded in VOPP, TreadMarks and MPI respectively. Our DSM system, optimized VODCA[15], is used to run the VOPP programs, Traditional DSM system, TreadMarks, is used to run the TreadMarks programs and LAM/MPI V7.1 is used to run the MPI programs.

The applications used in our experiment include Gauss, Integer Sort(IS), Successive Over-Relaxation(SOR), Neural network(NN), Water, Traveling Salesman Problem(TSP), Barnes-Hut, BT, CG and MG, which are mostly chosen from SPLASH-2 benchmark suite [12] and NAS Parallel Benchmarks(NPB) [13].

Our experiments were carried out on a cluster with Infiniband interconnections, running Linux 2.6. Each node has two 1.6GHz processors and 4 Gbytes memory. The page size of the virtual memory is 32 KB.

3.1 Improvement on VODCA

As we know, barriers incur many messages, especially when the number of processes increases. Furthermore, every process often waits for the slowest process as there is synchronization when barrier is called. It significantly slows down the parallel program if barriers are frequently called.

The source files of VODCA V1.0.1 can be downloaded from the web site http://vodca.otago.ac.nz/. Barriers in this VODCA totally rely on the work of the Vdc_barrier_manager, which is running on proc 0. When a barrier function is called, every process sends a barrier requirement to proc 0 and then waits for an end-of-barrier reply from Proc 0. We therefore could conclude that the barrier manager has too much burden that it is the bottleneck of the whole barrier process.

We implement the barrier with the binomial-tree model[11] whose complexity is just O(logN) instead of the O(N) of the original linear model. As a result the barrier time on 64 processes in the cluster we mentioned above is 623.8 microseconds, which is 17% faster than the original implementation. When the number processes increases, it is expected that our new barrier implementation could outperform the original VODCA implementation more significantly due to the complexity difference.

3.2 Performance Overview

Gauss implements the gauss elimination algorithm in parallel. The matrix size of Gauss is 8000*8000 and the number of iterations is 1024 in our tests. The TreadMarks Gauss program has the false sharing effect. The VOPP version has significantly improved the performance by removing the false sharing effect with local buffers. The speedup of Gauss is shown in Figure 1.

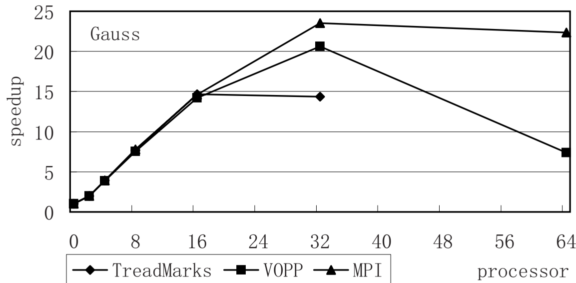

Fig. 1. Speedup of Gauss

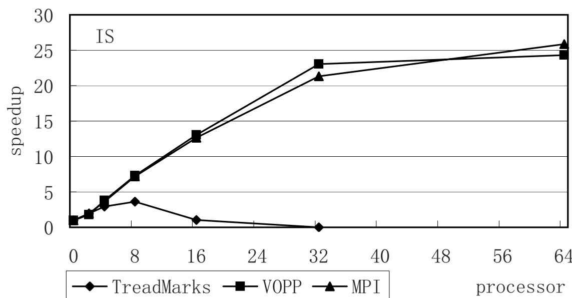

Fig. 2. Speedup of IS

IS ranks an unsorted sequence of N keys using bucket sort and the problem size in our tests is ($2^{27}*2^{17}$,40). The speedup of IS is shown in Figure 2.

SOR uses a simple iterative relaxation algorithm with a two-dimensional grid as input. Every element is updated to a function of its neighbors' values in each iteration. We use local buffers for those infrequently-shared data in our VOPP programs. In contrast, we use shared memory (a set of views) for those frequently-shared data such as the border elements. In our tests SOR processes a matrix with size of 8192*1024 in 100 iterations. The speedup of SOR is shown in Figure 3.

NN trains a back-propagation neural network in parallel using a training data set. The VOPP version of NN uses local buffers for infrequently-shared data and acquire_Rview for read-only data. The acquire_Rview for read-only data is

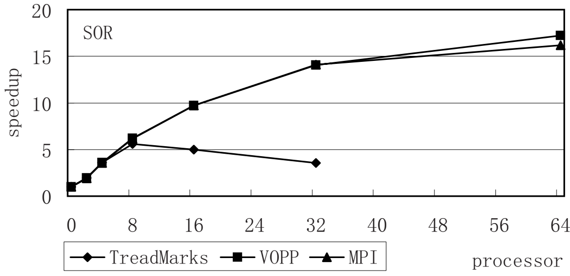

Fig. 3. Speedup of SOR

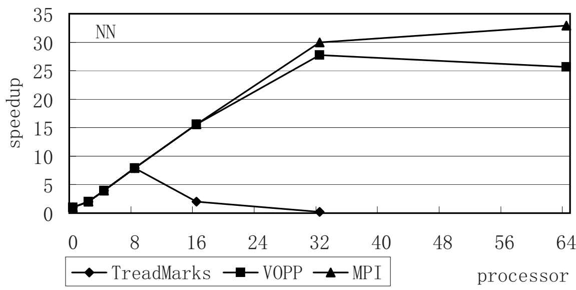

Fig. 4. Speedup of NN

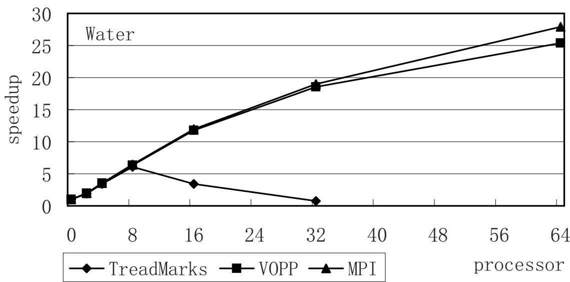

Fig. 5. Speedup of Water

very important for the VOPP program. The size of the neural network in NN is 9*40*1 and the number of epochs taken for the training is 1024. The speedup of NN is shown in Figure 4.

Water from the SPLASH-2 benchmark suite is a molecular dynamics simulation program. The bulk of the inter-processor communication happens during the force computation phase. Each processor computes and updates the intermolecular force between each of its molecules and each of n/2 molecules following

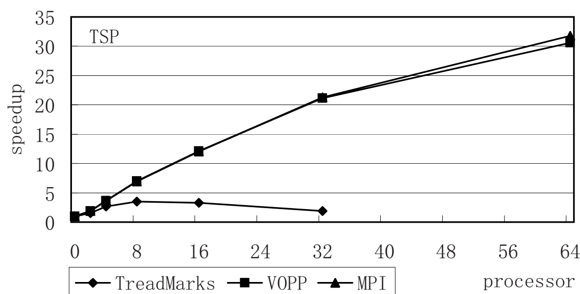

Fig. 6. Speedup of TSP

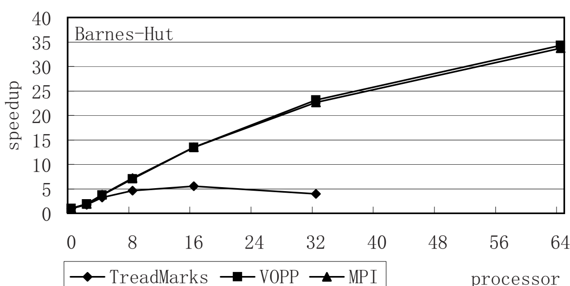

Fig. 7. Speedup of Barnes-Hut

it in the array in wrap-around fashion. The data set we used is 1728 molecules and 5 steps. The speedup of Water is shown in Figure 5.

TSP solves the traveling salesman problem using a branch and bound algorithm. We solve a 19-city problem with a recursive-solve threshold of 12. The speedup of TSP is shown in Figure 6.

Barnes-Hut from the SPLASH-2 benchmark suite is an N-body simulation using the hierarchical Barnes-Hut Method. We run Barnes-Hut with 32768 bodies in our experiment. The speedup of Barnes-Hut is shown in Figure 7.

BT creates a binary tree with a depth of 9. It keeps all those unexpanded nodes by using a task queue. The speedup of BT is shown in Figure 8.

CG from NAS Parallel Benchmarks implements conjugate gradient algorithm. The data set we used in the experiment is 15 niter and 11 nonzer and 14000 nn, which is defined as LARGE problem size. The speedup of CG is shown in Figure 9.

MG, which comes from NAS Parallel Benchmarks, solves a poisson problem on a 128 by 128 by 128 grid, using 20 multigrid iterations. The speedup of MG is shown in Figure 10. Speedup results for these applications are shown in Figure1 - Figure10. These figures show the relative speedup of the 10 applications written in different styles running in the same hardware environment while the number of processes increases. Programs of TreadMarks slow down significantly when

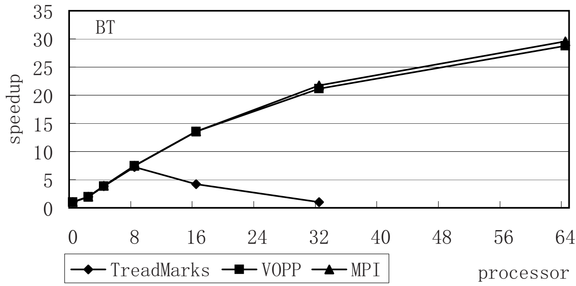

Fig. 8. Speedup of BT

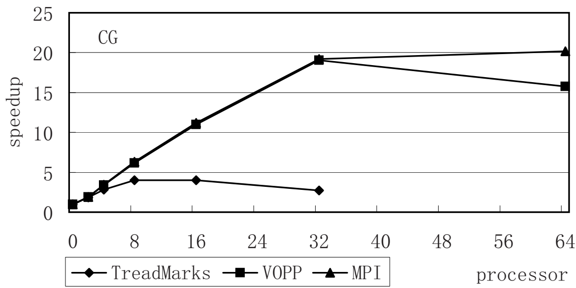

Fig. 9. Speedup of CG

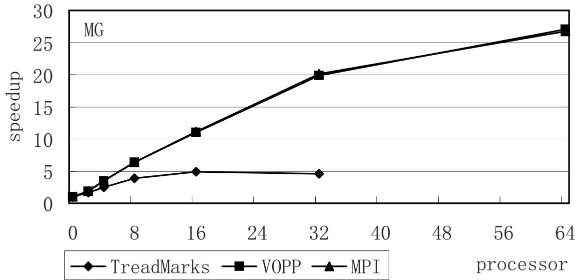

Fig. 10. Speedup of MG

running on more than 8 processors. Comparing the speedup of TreadMarks, VOPP and MPI on 32 processors, we can find that the performance of VOPP is much better than TreadMarks and it is nearly the same efficient as MPI. However, when running on more than 32 processors, some of VOPP programs including Gauss, NN and CG slow down and there are obvious performance gaps between VOPP and MPI. In contrast, SOR and Barnes-Hut of VOPP perform

better. For the rest five programs, there are very small performance gaps between VOPP and MPI.

3.3 Factors Contributing to the Performance of VOPP

Most of the differences in speedup and communication requirement between TreadMarks and MPI are a result of TreadMarks' lack of bulk transfer, extra messages for separate synchronization(barriers), false sharing and diff accumulation [9,10,14]. Contrastively, there is no false sharing in VOPP programs, as the views access are not nested and VOPP has succeeded in solving the problem of diff accumulation by introducing the new consistency maintenance protocol VOUPID [4,8].

Table 1. 32-Processor Total Messages for TreadMarks, VOPP and MPI

program	TreadMarks	VOPP	MPI
Gauss	2138734	1849612	247969
IS	239082	247682	39680
SOR	57850	49380	12400
NN	717741	696551	37355
Water	72800	65200	3760
TSP	74269	52276	5736
Barnes-Hut	604004	579006	1040
BT	52700	48200	7800
CG	84468	74849	12480
MG	125782	117601	21920

Table 1 and Table 2 provide figures for the number of messages and the amount of data exchanged when the programs are running on 32 processors. For TreadMarks and VOPP programs, we count the total number of messages and the total amount of data transferred. While for the MPI programs, we count the number of user-level messages and the amount of the user data sent in each run.

Table 2. 32-Processor Data Transferred(KB) for TreadMarks, VOPP and MPI

program	TreadMarks	VOPP	MPI
Gauss	23853491	7994304	7936991
IS	4012643	1318835	1300234
SOR	179728	52890	50840
NN	1603279	500457	282970
Water	91476	36942	36758
TSP	13925	171	168
Barnes-Hut	296320	208843	202560
BT	2383	816	761
CG	2884	1153	982
MG	12460	3752	3740

In Table 1, the two DSM systems, TreadMarks and VOPP require comparable amount of messages in ten different applications. They do not support bulk transfer either. In TreadMarks and VOPP systems, messages are incurred when updating shared data which is based on page fault. The messages occur in MPI programs are not comparable. In table 2, we can easily find the reason VOPP programs perform more efficiently than those of TreadMarks. By reducing false sharing and solving diff accumulation problem [3], VOPP programs send significantly less data when running on parallel systems. There are few unnecessary data transferred in VOPP programs, compared with those of MPI.

Although performance of VOPP has significantly improved and it is as efficient as MPI when running on less than 32 processors, we can find some programs of VOPP slow down when running on more processors. The main factors contributing to the performance slowdown are overheads of barriers and lack of bulk transfer.

As we point out in 3.1, barriers incur large amount of messages, especially when the number of processors increases. However, if we compare the VOPP programs with the MPI programs, we find there are much more barriers in the VOPP programs [7]. Gauss calls 15998 barriers; IS calls 120 barriers; SOR calls 200 barriers; NN calls 2410 barriers; Water calls 72 barriers; TSP calls 3 barriers; Barnes-Hut calls 19 barriers; BT calls 2 barriers; CG calls 1186 barriers and MG calls 484 barriers. In contrast, there are almost no barriers in MPI programs. Accordingly the performance gaps for Gauss, NN and CG are larger, while for other programs they are much smaller.

The reason why there have to be more barriers in VOPP programs has been explained in [5]. Barriers have to be used to make sure the sequential consistency. In contrast, in the MPI code, there is no need to use a barrier for synchronization, since the receive primitive is synchronized with the send primitive and is always finished after the send primitive. To verify the overhead of barriers is a significant contributor, we run our VOPP programs on two versions of VODCA and compare them with the MPI version programs. As the largest performance gap occurs in Gauss program, we choose Gauss for our verification. Firstly, we run Gauss on our optimized VODCA system with improved barrier in our previous tests, relatively, we now run Gauss again on original VODCA without improved barrier. Secondly, we intentionally make the number of barriers in VOPP Gauss the same as that in MPI version. We run the new Gauss on LAM/MPI, optimized VODCA with improved barrier and original VODCA without improved barrier.

In Figure 11, VOPP_ori represents VOPP program running on original VODCA without improved barrier. We find the performance gap is obviously larger between VOPP_ori and MPI when running on 64 processors. Compared with the performance gap between optimized VODCA and MPI, we can find that there is obvious performance gain by optimization of barrier.

In Figure 12, the performance gaps of the modified Gauss become very small. The MPI curve represents the MPI Gauss with unnecessary barriers. The VOPP curve represents the Gauss program with less barriers running on optimized VODCA with improved barrier, of which the execution result is wrong. The VOPP_ori represents the modified Gauss running on original VODCA without

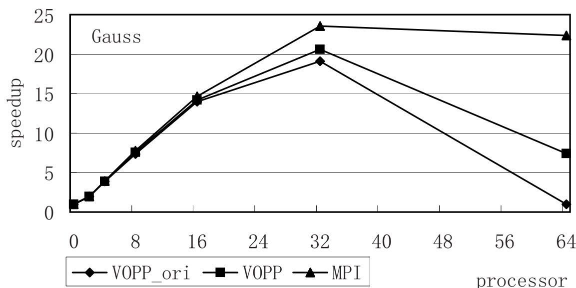

Fig. 11. Speedup of Gauss with/without improved barrier

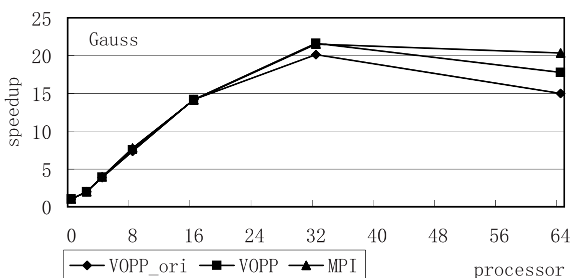

Fig. 12. Speedup of Gauss with same number of barriers

improved barrier. The numbers of barriers in each of them are nearly the same. For 64 processors, the speedup gap between VOPP and MPI is reduced from 15 to 3 and the gap between VOPP and VOPP_ori is also reduced, from 7 to about 3.

From the two figures above, we can conclude that the performance slowdown of VOPP programs is a result of the overheads from barriers. Although our optimized barrier performs more efficiently, the overheads of barriers are still the source contributing to the performance slowdown when the number of processors increases. We are considering replace barriers with some light weight synchronization primitives.

Like TreadMarks, VOPP does not support bulk transfer. The update of views is based on pages, which means for large amount of data to be updated, there will be extra messages to handle access misses. In contrast, MPI is able to aggregate large amounts of data in a single message. To simulate the effect of bulk transfer, we define the VOPP page size as a multiple of the hardware page size. By increasing the VOPP page size, a view is likely to be updated with less page faults, which means there will be less messages for handling the access miss.

As shown in Table 3, by increasing the VOPP page size, the total number of messages has been reduced greatly. And the speedup gap of Gauss between

Table 3. 64-processor messages, data transferred and speedup of Gauss

Gauss	VOPP_32KB	VOPP_64KB	MPI
messages	3921256	2248807	531362
data transferred	17149791	17144611	17026841
speedup	7.376	10.218	22.343

VOPP and MPI becomes smaller. The extra messages incurred by lack of bulk transfer is another source contributing to the performance of VOPP.

In summary, by reducing false sharing and solving the problem of diff accumulation, VOPP performs much more efficiently than TreadMarks. There are significantly less data to be transferred. However, the extra messages incurred by separate synchronization and lack of bulk transfer limit the performance of VOPP when the number of processors increases.

4 Conclusions

This paper evaluates a novel DSM programming style VOPP for cluster computers. Ten applications of all kinds are converted and optimized under three different parallel systems, VOPP, TreadMarks and LAM/MPI. Our experimental results demonstrate that, on a large variety of programs, there is significant performance advantage of VOPP against TreadMarks and VOPP is now comparable with MPI. The performance of VOPP is nearly as efficient as MPI on less than 32 processors. We also analyze the factors contributing to the performance gap between VOPP and MPI. VOPP is still slower than MPI when the number of processes is large because of extra messages for separate synchronization and lack of bulk transfer mechanisms.

As a software distributed shared memory system, VOPP is convenient for programmers to use and it is easy to achieve correctness and efficiency. It only requires programmers to insert primitives when a view is accessed. Programmers do not have to determine what to communicate but focus on the implementation of the parallel algorithm. Besides, programmers are allowed to participate the performance optimization by wisely partitioning views. For programs with complicated communication patterns, especially for those with complicated or irregular array accesses or with data structures accessed through pointers, VOPP shows better programmability than MPI. Our experience indicates that it is convenient to port traditional DSM programs to VOPP and it is easier to implement parallel algorithm using VOPP than MPI. Furthermore, the performance of VOPP is as efficient as MPI for 32 processors and we therefore have another choice for parallel programming on cluster of computers.

References

1. Keleher, P., Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: TreadMarks: Distributed shared memory on standrd workstations and operating systems. In: Proceedings of the 1994 Winter Usenix Conference, pp. 115–131 (1994)

2. Amza, C., Cox, A.L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W., Zwaenepoel, W.: TreadMarks: Shared memory computing on networks of workstations. IEEE Computer 29, 18–28 (1996)
3. Huang, Z., Purvis, M., Werstein, P.: View-Oriented Parallel Programming and View-based Consistenc. In: Proc. of the Fifth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 505–518 (2004)
4. Huang, Z., Purvis, M., Werstein, P.: View-Oriented Update Protocol with Integrated Diff for View-based Consistency. In: Proc. of the IEEE/ACM Symposium on Cluster Computing and Grid (2005)
5. Huang, Z., Purvis, M., Werstein, P.: Performance Evaluation of View-Oriented Parallel Programming. In: Proc. of the 2005 International Conference on Parallel Processing, IEEE Computer Society, Los Alamitos (2005)
6. Gropp, W., Lusk, E., Skjellum, A.: A high performance, portable implementation of the MPI message passing interface standar in Parallel Computing, 789–828 (1996)
7. Werstein, P., Pethick, M., Huang, Z.: A Performance Comparison of DSM, PVM and MP. In: Proc. of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 476–482 (2003)
8. Keleher, P.: Lazy Release Consistency for distributed shared memor. In: Ph.D. Thesis (Rice Univ) (1995)
9. Klaiber, A.C., Levy, H.M.: A comparison of message passing and shared memory architectures for data parallel language. In: Proc. of the 2th Annual International Symposium on Computer Architecture, pp. 94–106 (1994)
10. Lu, H., Dwarkadas, S., Cox, A.: Zwaenepoel: Quantifying the performance differences between PVM and TreadMark. Journal of Parallel and Distributed Computing, 65–78 (1997)
11. Hensgen, F.R., Manber, U.: Two algorithms for barrier synchronizatio. International Journal of Parallel Programming, 1–17 (1988)
12. Singh, J.P., Webber, W.-D., Gupta, A.: SPLASH: Stanford parallel applications for shared memor. In Technical Report, Department of Computer Science, Stanford University (1991)
13. Bailey, D., Barton, J., Lasinski, T., Simon, H.: The NAS parallel benchmark. In Technical Report 103863, NASA (1993)
14. Lu, H., Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: Message Passing versus distributed shared memory on networks of workstation. In: Proc. of SuperComputing 95 (1995)
15. Huang, Z., Chen, W., Purvis, M., Zheng, W.: VODCA: View-Oriented, Distributed, Cluster-based Approach to parallel computin. In: Proc. of the IEEE/ACM Symposium on Cluster Computing and Grid (2006)