A Preliminary Study of an Application Code Optimization when It's Submitted to be Executed on top of a DSM System

Mário Donato Marino, Geraldo Lino de Campos and Líria Matsumoto Sato

.

Department of Computer Engineering (PCS)
Polytechnic School, University of São Paulo
São Paulo, Brazil
e-mail:{mario, geraldo, liria}@regulus.pcs.usp.br

Abstract

Distributed Shared Memory (DSM) is an abstraction of shared address space over a network of computers, which is a distributed system. This model allows programs, that have been written for shared memory, to be easily ported, without the programmer worrying about movimentation, linearization and packaging of data, ports and send-receive primitives. In DSM systems to reduce computation time changes the relation between computation time and the pair time to obtain data and time spent on synchronizations. Nautilus is a DSM system, that uses release consistency and some techniques like multiple concurrent writers, multithreading and bypassing TCP/IP stack. This is a preliminary study to investigate what's the influence of the application code optimization over the speedup of Nautilus DSM system, which also ins in development. In this paper, we compare the speedup difference between an application program with and without a -O2 optimization of gcc compiler. For this comparison, we experiment this optimization with two different programs Matmul and EP (from NAS) and submit them on Nautilus.

area: distributed systems

1 Introduction

Paralellism with the agregation of several processors or nodes, has adopted two tendencies: physical shared memory and distributed memory.

In distributed memory model, the programmer needs to worry with movimentation, sending and receiving messages, packaging and linearization[1]. Because of the saturation and contention of the memory bus that interconnects the processors, the distributed model is more scaleable than the physical shared memory. On the other hand, the shared model is easier to create and to port programs. Distributed Shared Memory is a software abstraction[1] of shared memory over a computer network, as we can see in figure 1.

To operate as a DSM system, the system must [1]: *(i)* replicate or migrate data; *(ii)* permit portability of the parallel programs that have been written for shared memory; *(iii)* maintain the consistency of the data among nodes of the network.

The **main factor** that determinates the speedup of a DSM system is the **number of messages** that are transmitted through the net. In the last 5 years, the international community has been working intensively to reduce the number of messages. Some people investigate the data locality and data distribution. Some people attack the synchronization with distributed barrier and lock algorithms. Other people attack the consistency with the adoption of *weak consistency* models.

Basically, in DSM systems there are three types of messages that are transmitted through the net:

- data messages: if the data coherence unit is a page, there will be messages that contain pages;
- consistency messages: messages that contain *diffs* [2, 3](*diffs* are a codification of the modifications suffered by a page during a critical section) ;
- synchronization messages: parallel programs need synchronization to coordinate their actions. Barriers and locks are used to synchronizate distributed processes over the network. Thus, there will be messages of synchronization that implement barriers and locks.

As synchronization is a consequency of parallel programs behaviour, some memory consistency models have proposed to maintain the consistency only at these points (of synchronization), thus reducing dramatically the number of consistency messages. These memory models are weak consistency model and its variations. For instance, *release consistency*[2], *lazy release consistency*[3], *entry consistency[4]* and *scope consistency*[5].

As a consequence of this reduction, it's possible to achieve better speedups, similar to a physically shared memory machine [3].

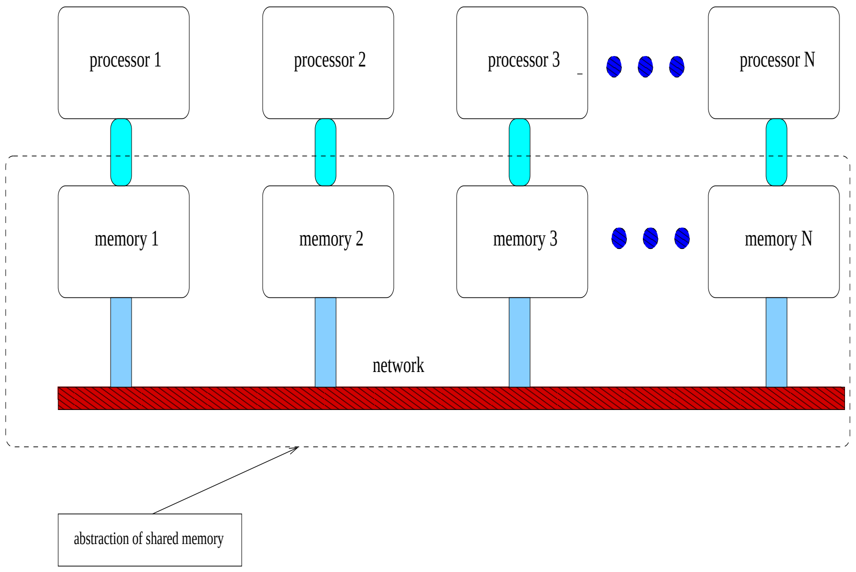

figure 1: abstraction of DSM [2, 3]

Concluding, DSM systems try to maximize the reduction of any type of messages that are transmitted through the net.

The DSM speedup will be good if the relation between computation time and the pair time to obtain data and time spent on synchronization operations is as high as possible.

If we optimize the application level code in sequential programming, in most cases a good computation time reduction is obtained. In DSM systems to reduce computation time changes the relation between computation time and the pair time to obtain data and time spent on synchronizations. But, what does happen if we optimize the code of an application and submit it on a DSM system ? This is a preliminary study to investigate what's the influence of the application code optimization over the speedup of the DSM system.

To investigate this purpose, we will use a free Unix and a free compiler: Linux (2.X) and gcc compiler. The optimization level which was chosen is **-O2**. The DSM system chosen is Nautilus and the version of this software evaluated is under development. Thus, the main objective of this article is to execute two applications with and without **-O2** level and compare the speedup difference. The applications chosen for this evaluation are Matmul and EP (from NAS benchmark).

The remaining of this paper is organized as follows: Section 2 introduces some *Nautilus* features and why Nautilus was chosen. Section 3 describes the **-O2** optimization code generated from gcc compiler. Section 4 describes the programs that are being submitted to *Nautilus* with its input parameters. Section 5 presents evaluation results on a network of PCs. This paper is summarized in Section 6.

2 Nautilus

Before describing Nautilus, it's useful to mention that some other DSM systems such as Munin[2], TreadMarks[3], CVM[7] and JIAJIA[6], in order to improve the performance, applied some techniques:

- multiple concurrent writers and *diffs* propagation [2, 3];
- lazy *diff* creation [3];
- page faults handling overheads eliminated via annotated load and store operations [7];
- *multithreading* [8] to minimize the context switch;
- use *sockets* and UDP protocol to minimize TCP overheads;
- bypass the TCP/IP stack [9];

The main features of Nautilus DSM system are:

- update or invalidate consistency messages, so it replicates the shared data;
- *release consistency* memory model mechanism ;
- multiple concurrent writer and *diffs*;

- tcp or udp protocols;
- multithreading (*linuxthreads*);
- bypassing of the tcp/ip system (not yet implemented);
- network of PCs (low cost);
- free Unix (Linux 2.X);
- primitives compatible with TreadMarks and JIAJIA; so it is too easy to modify programs from this DSM systems to be executed over *Nautilus.*

We will now justify why we use Nautilus DSM systems in our experiment.

As Quarks, TreadMarks, CVM and JIAJIA do, Nautilus can use the UDP protocol to minimize the overhead of TCP/IP system. We are working to add techniques (not yet implemented) from Gamma Project[9] to bypass the stacks of TCP and UDP protocols. We believe that with this technique we will greatly reduce the overhead of the system.

In 1995, for instance, Quarks used *pthreads* to minimize the switch context overhead of its routines. But in 1998, its development team chose not to use threads any more because they don't improve the performance a lot. In *Nautilus* system we use *linuxthreads*, because with the operating systems and machines evolution to *multithread* systems, threads generate less overhead than processes.

As TreadMarks and Quarks, *Nautilus* can be executed on a network of PCs, that has been much used on several projects[10]; so, with a free operational system and a low cost network of PCs, it's possible to have an environment for the development of parallel applications over a distributed memory machine model.

The *Nautilus* primitives are totally compatible with TreadMarks and JIAJIA.Thus, it's easy to port programs from other DSM systems or from physically shared memory machines using m4 macros. A good example of this compatibility is that the programs Matmul and EP (NAS) were modified from JIAJIA.

3 -O2 optimization of gcc compiler

-O2 optimization involves optimizations that do not involve a space-speed tradeoff are performed [10].

For more details, use the command man gcc on a unix machine that has gcc installed[10].

4 Experimental Evaluation

In this study, *Nautilus* is operating with:

- *update* ***protocol*** to maintain the consistency;
- TCP protocol.

We will evaluate two application programs, Matmul and EP (NAS benchmark) in order to evaluate the behavior of *the speedup,* with and without -O2 optimization of gcc.

4.1 Evaluation Programs

4.1.1 Matmul

Matmul is a simple matrix multiplication program with inner product algorithm. All arrays are allocated in shared memory. To achieve a good data locality, the multiplier is transposed before multiplication. This program requires no synchronization in the multiplication process, so only three barriers are used at the beginning and the end of the program[8].

The matrix **size** of the matrix used in this experiment is **1024x1024**.

4.1.2 EP (from NAS)

The Embarrassingly Parallel (EP) program from the NAS benchmark suite generates pairs of Gaussian random deviates with a scheme that is well suited for parallel computation and tabulates the number of pairs successively. The only communication and synchronization in this program is summing up a ten-integer list in a critical section at the end of program [8].

The parameter used in EP program is M=2^{24}.

4.2 Environment

The evaluation programs of this study are executed on top of *Nautilus* on the following network of PCs:

- nodes: K6 - 233 MHz (AMD), 64 MB of memory and 2 GB IDE disk;
- interconnection: a hub and fast ethernet cards (100 Mbits/s).

In order to measure the speedups, the network above was fully isolated from any other external network.

The operating system used was Linux Red Hat 5.0.

The compiler used in this experiment is gcc-2.7.2.3, with **-O2** option.

The applications were executed and the speedups measured using *Nautilus* running on up to **7 nodes**.

5 Analysis of the Results

5.1 Matmul

5.1.1 Matmul results

We have measured the execution time of Matmul 1024x1024 matrix sizes, increasing the number of nodes. The resulting times are shown in table1.

Also, we calculate some parameters:
$speedupnop(i) = \frac{tnop(1)}{tnop(i)}$,
$speedupop(i) = \frac{top(1)}{top(i)}$,
$tr = \frac{tnop-top}{tnop} \times 100$ *%, tr is the time reduction,*
$sr = \frac{speedupnop-speedupop}{speedupnop} \times 100$ *%, sr is the speedup reduction*
as shown in table 1, where

- (i) means the number of nodes *where Nautilus* is being executed;
- tnop(i)are the times (in seconds) with i nodes and without **-O2** optimization ;
- top(i)are the times (in seconds) with i nodes and **-O2** optimization;
- speedupnop(i) is the speedup with i nodes and without **-O2** optimization;
- speedupop(i) is the speedup with i nodes and **-O2** optimization;

(i)	tnop(i) (s)	top(i) (s)	speedupnop(i)	speedupop(i)	tr(%)	sr(%)
1	187.90	130.20	1.00	1.00	30.00	0.00
2	109.30	80.50	1.72	1.62	26.30	5.81
3	91.90	73.30	2.04	1.77	20.20	13.23
4	80.30	61.90	2.34	2.10	22.20	10.25
5	75.10	69.80	2.50	1.86	7.00	25.60
6	79.60	75.60	2.36	1.72	5.00	27.12
7	92.20	91.30	2.04	1.42	1.00	30.40

table 1: metrics obtained from *Nautilus* executing Matmul without and with optimization (size = 1024)

From table 1, we obtain figure 2, where the speedups of Matmul without and with -O2 optimization are ploted.

By observing figure 2, we have obtained a good speedup for up to 5 nodes, in both cases (with and without **-O2** optimization). The saturation of the speedup curve occurs because the PCs are interconnected by a hub, so when we increase the number of nodes, more collisions occur because more pages are transmitted through the net, saturating the network and slowing down the performance. The speedup obtained wasn't linear because, as we said, many pages are transmitted and because of the shared media (hub) several collisions occur slowing down the speedup.

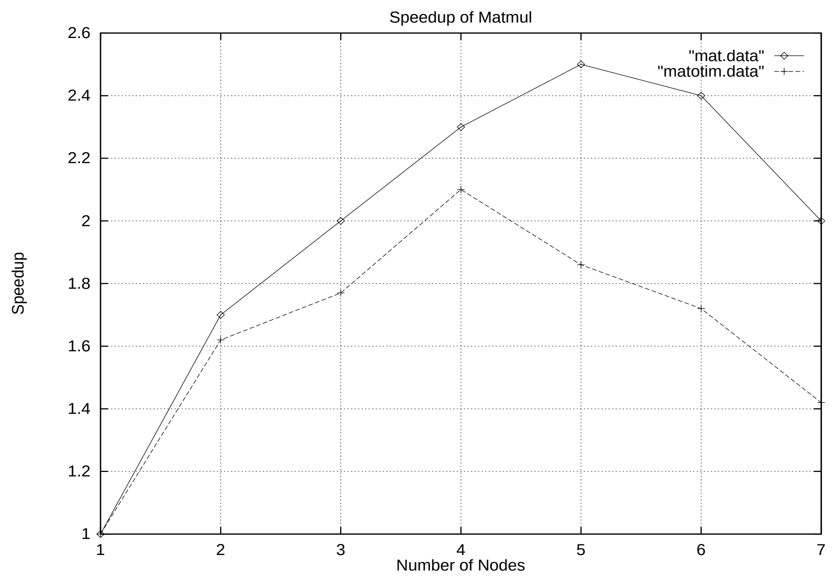

figure 2: speedup of Matmul without (mat.data) and with (matotim.data) -O2 optimization (matrix size = 1024)

5.1.2 Comparison between speedups of Matmul program with and without -O2 optimization

From table 1, the time reduction quotient is decreasing as we increase the number of nodes because of the saturation of the network, since more messages containing pages are transmitted to more nodes. Besides, by observing figure 2 and table 1 (speedup reduction column), for all nodes, the speedup of the optimized program is worser than the non optimized, as we mentioned previously, because as the application runs faster, the time to send synchronization and consistency messages becomes more decisive on the relation between computation time, synchronization and consistency.

5.2 EP (from NAS benchmark)

5.2.1 EP results

We have measured the execution time of EP with M=2^{24} increasing the number of nodes. The resulting times are shown in table 2.

As the same way as item 5.1.1, we calculate the same proposed metrics.

(i)	tnop(i) (s)	top(i) (s)	speedupnop(i)	speedupop(i)	tr(%)	sr(%)
1	124.30	90.40	1.00	1.00	27.27	0.00
2	62.40	46.80	1.99	1.93	25.00	3.01
3	43.40	32.80	2.86	2.75	24.42	3.85
4	33.90	26.60	3.67	3.40	21.53	7.36
5	30.00	22.80	4.14	3.96	24.00	4.35
6	26.90	22.70	4.62	3.98	15.98	13.85
7	25.20	22.80	4.93	3.96	9.52	19.68

table 2: metrics obtained from *Nautilus* executing EP without and with **-O2** optimization (M=2^{24})

As the same way as item 5.1.1 does, we can calculate time reduction and speedup reduction, as showed in table 2.

As we can see on table 2, the reduction of time remains constant until 5 nodes because of the small number of synchronization messages and the high locality that this program presents. For more than 5 nodes, the contention and saturation of the network because of the synchronization equalize the times, producing low time reduction.

The speedup reduction from table 2 shows that until 5 nodes the speedups are almost the same (low speedup reduction) and with more than 5 nodes the speedup reduction increases, as we justified in last paragraph.

By the same way as item 5.1.1, we can obtain from table 2 the speedup curve (figure 3) for EP.

By observing figure 3, we reach almost a linear speedup for up to 3 or 4 nodes and a good speedup for up to 7 nodes in both cases (with and without **-O2** optimization). Because the PCs are interconnected by a hub, when we increase the number of nodes, more collisions occur at the synchronization point, slowing down the performance. The EP program, as we said, has almost no communication, so the speedup is very good (almost linear) as we increase the number of processors.

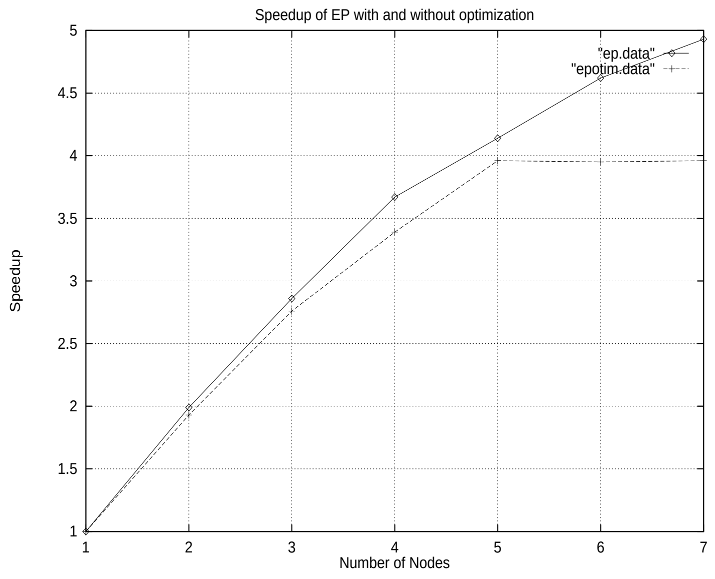

figure 3: speedup of EP without (ep.data) and with (epotim.data) **-O2** optimization ($M=2^{24}$)

5.2.2 Comparison between Matmul and EP

Concluding, the behavior of Matmul and EP are different: EP has more locality and few synchronization and data transmitted through the net, so the optimization is better in this case, giving small speedup reduction.

Matmul has more synchronization and more messages containing data (pages), thus the optimization does not produce good effects.

6 Conclusion

As we can see, applying some code optimization, using a network of workstations with shared media and a DSM which adopts release consistency model, gives worser speedups. We show that with two different applications, Matmul and EP, which generate a little synchronization, the speedup decreases depending on the locality and synchronization of these programs. Concluding, we could see that the speedup decreases with the application code optimization for the applications evaluated.

We expect that if we test with programs that have more synchronization, the optimization code does not improve the speedup, there isn't time reduction in this case. In the case studied in this paper, EP has less synchronization and data (pages) that are transmitted, so the optimization changes the speedup a little. Matmul has more synchronization and messages containing data.

We think if we apply a technique as from [9], as synchronization and time spent with consistency messages decrease, there will be small decrease of speedups we observed.

Verifying if there is speedup behavior difference, we intend to do as a future work:

- changing the hub by a switch;
- trying another programs, from SPLASH-II suite for example;

- applying a technique as from [9], improving the network performance;
- finish this version which is in development to evaluate again the influence of the same benchmarks;
- trying other optimization levels of gcc.

References

[1] Stum M. , Zhou S. , *Algorithms Implementing Distributed Shared Memory*, University of Toronto, IEEE Computer v.23 , n.5 , pp.54-64 , May 1990.

[2] Carter, J. B. , *Efficient Distributed Shared Memory Based on Multiprotocol Release Consistency*, PHD Thesis, Rice University, Houston, Texas, September, 1993.

[3] Keleher P. , *Lazy Release Consistency for Distributed Shared Memory*, PHD Thesis, University of Rochester, Texas, Houston, January 1995.

[4] Bershad B. N. , Zekauskas M. J. , SawDon W. A. , *The Midway Distributed Shared Memory System* , COMPCOM 1993.

[5] Iftode L., Singh J.P., Li K., *Scope Consistency: A Bridge between Release Consistency and Entry Consistency*, In The 8th Annual ACM Symposium on Parallel Algorithms and Architectures, 1996.

[6] Hu W., Shi W., Tang Z., *JIAJIA: An SVM System Based on a new Cache Coherence Protocol,* technical report no. 980001, Center of High Performance Computing , Institute of Computing Technology, Chinese Academy of Sciences, January, 1998.

[7] Keleher P., *The Relative Importance of Concurrent Writers and Weak Consistency Models*, in Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS-16), pp. 91-98, May 1996.

[8] Swanson M., Stoller L., Carter J., *Making Distributed Shared Memory Simple, Yet Efficient*, Computer Systems Laboratory, University of Utah, technical report , 1998.

[9] Chiola, G., *Gamma Project: Genoa Active Message Machine*, http:/www.disi.unige.it/project/gamma.

[10] http:/www.gnu.org¸ GNU Gcc manual