Controlling Distributed Shared Memory Consistency from High Level Programming Languages

Yvon JÉGOU

IRISA / INRIA
Campus de Beaulieu,
35042, RENNES CEDEX, FRANCE,
Yvon.Jegou@irisa.fr

Abstract One of the keys for the success of parallel processing is the availability of high-level programming languages for on-the-shelf parallel architectures. Using explicit message passing models allows efficient executions. However, direct programming on these execution models does not give all benefits of high-level programming in terms of software productivity or portability. HPF avoids the need for explicit message passing but still suffers from low performance when the data accesses cannot be predicted with enough precision at compile-time. OpenMP is defined on a shared memory model. The use of a distributed shared memory (DSM) has been shown to facilitate high-level programming languages in terms of productivity and debugging. But the cost of managing the consistency of the distributed memories limits the performance. In this paper, we show that it is possible to control the consistency constraints on a DSM from compile-time analysis of the programs and so, to increase the efficiency of this execution model.

1 Introduction

The classical compilation scheme for High Performance Fortran (HPF) is based on the application of the owner-compute rule which, after distributing the ownership of array elements to the processors, distributes the charge of executing each instruction to the processor owning the variable modified by this instruction. It is possible to map HPF distributed arrays on a distributed shared memory and to avoid the burden of explicitly updating the local memories through message passing. OpenMP was primarily defined for shared memory multi-processors. But it is possible to transform OpenMP programs for an SPMD computation model if the accesses to the non-private data can be managed, for instance, through a software DSM.

However many authors have reported poor performance on sequentially consistent distributed shared memories for HPF and for OpenMP mainly because the high latencies in inter-processor communications penalize the management of sequential consistency protocols. The situation is even worse in the presence of false sharing.

The independence of the iterations of a parallelized loop guarantees that the modifications to the data during one iteration need not be visible for the execution of the other iterations of the same loop. The propagation of the modifications can be delayed until the next synchronization point. Many projects such as TreadMarks [1] or Munin

[3] have shown that the use of a relaxed consistency software DSM along with some consistency control on synchronizations can avoid the cost of maintaining the memory consistency during the execution of a parallel loop.

Mome (**Mo**dify-**me**rge) is a software relaxed consistency, multiple writers DSM. Mome mainly targets the execution of programs from the High Performance Computing domain which exhibit loop-level parallelism. This DSM allows to generate consistency requests on sections of the shared space and so to take advantage of the analysis capacity of modern compilers.

Next section compares implicit and explicit consistency management and their relations with HPF and OpenMP compilation models. Section 3 presents the Mome DSM consistency model and Sect. 4 gives some details on the possible controls of the DSM behavior from the programs. Some experimental results are presented in Sect. 5.

2 Implicit versus Explicit Consistency Management

Using an implicit consistency management, the modifications to the shared memories become automatically visible after synchronization operations such as locks or barriers. Race-free parallel programs should produce the same results as with a sequential consistency management when run with an implicit consistency management. TreadMarks [1] follows this model and implements a *release consistency* model with a *lazy invalidate* protocol. Using an explicit consistency management scheme, the modifications from other processors are integrated in a memory section only upon explicit request from the application. Entry consistency [2] implements a form of explicit consistency management through the association of shared objects with synchronization variables. The Mome DSM implements another form of explicit management where a processor can specify the memory section which must be updated without making reference to synchronization variables.

The choice between these schemes depends mainly on the compilation technique in use. The execution flow is controlled in HPF by the owner-compute rule. This rule states that each data element must always be updated by the same processor. As long as this rule can be applied, a distributed data element is always up-to-date on the owning processor. For the execution of a parallel loop, it is necessary to check for the consistency of the non-owned accessed elements only. The explicit consistency management scheme seems to be well adapted to the HPF model. The HPF compilation strategy for Mome is identical to the compilation strategy for an explicit message passing model. The main difference is that the message passing calls in the generated code are replaced by consistency requests to the DSM on the same memory sections. In many HPF implementations, the generated code communicates with the run-time system only through object descriptors. The adaption of these implementations to the Mome DSM is then limited to replacing the calls to the message passing layer by calls to the DSM interface inside the run-time system of the language. No modification of the source program is necessary.

In the OpenMP computation model, the parallelism is specified through program directives. The compilation scheme is not based on the owner-compute rule. Each data element can potentially be modified by any processor. It is still possible to use an ex-

plicit consistency scheme if a static data access analysis can be performed at compile-time. An implicit consistency management scheme seems to be the only choice in many cases for the OpenMP programming model.

But real applications do not always fit exactly in one of these models. A strict application of the owner-compute rule on HPF programs does not always produce efficient results, for instance in the presence of indirect array accesses or with complex loop bodies. Moreover, poor data access analysis can lead to update requests on the whole shared data using the explicit consistency scheme. Although the OpenMP computation model is not based on data access analysis, the performance of the resulting codes can often be optimized if the data access patterns are considered in the loop distributions.

3 Mome DSM Consistency Model

Basically, Mome implements a simple relaxed consistency model for the DSM page management. A processor must send a consistency request to the DSM specifying a shared memory address range each time the view of this address range must integrate modifications from other processors. Mome maintains a global clock and most of the consistency requests make reference to this clock. For each memory page, the DSM keeps track of the date of the oldest modification which has not yet been propagated. If this modification date precedes the reference date of a consistency request, all known modifications are merged (exclusive or $\bigoplus$ of the modified pages) and a new version of the page is created. In the other case, the current version of the page is validated and no modification needs to be integrated. The date associated to a consistency request can be the current date, the date of the last synchronization barrier, the last release date of an acquired mutex lock, or the date of any other event detected by the application.

All the consistency requests are implemented in Mome using a two-step sequence. The first step, at the time of the call, is local to the requesting processor: the consistency constraints (mainly the date) are recorded in the local page descriptor and, if necessary, the memory section is unmapped from the application. The second step is applied when the application tries to read or write inside the memory section and generates a page fault. This second step involves a transaction with a global page manager which can confirm the local copy of the page if the current version has been validated. If modifications to the memory section need to be made visible, the processor receives the new version of the page after all known modifications have been merged.

This two-step procedure limits the updates to the pages which are really used by the application and implements a form of *lazy updates*. Using a combination of consistency requests and prefetch requests, the DSM interface allows to update the local memory before the application generates a page fault.

4 Consistency Management Optimizations

In order to fully exploit the analysis capacity of modern compilers, the Mome interface offers some level of control on the consistency management strategy through prefetching, through on-the-fly consistency checks and through page manager redistribution.

It is possible to combine the consistency requests with non blocking read or write prefetch requests on the shared data. This possibility especially targets the case where the compile-time data access analysis can predict with enough precision which parallel variables are to be accessed in the near future. Data prefetching reduces the page fault latency.

The current implementation of Mome uses a directory-based page management. All processors share a global page manager for each page. The Mome DSM allows for the migration of a page manager, for instance, on the processors making the most frequent requests. This possibility reduces the latency of the consistency requests when enough affinity exists between the data distribution and the distribution of the control. This is the case when the owner-compute rule is applied on HPF programs: the managers can be distributed according to the data distribution.

The basic consistency management strategy of Mome uses a two-step procedure where all access rights are removed from the application during the first step and a transaction with the global manager is generated on the resulting page faults. Although the cost of the consistency management can be reduced using data prefetches, at least two system calls and one page fault need to be generated for each page even if no update of the current copy is needed. This situation comes from the fact that, using Mome, all consistency requests must be generated by the consumers. At the opposite of TreadMarks, Mome does not automatically propagate the write notices. The *on-the-fly consistency checks* we are currently integrating in the Mome DSM partly fulfills this weakness. In on-the-fly mode, which can be selected at the page level, the global page manager broadcasts some information to all processor holding a copy of a page as soon as some other processor is allowed to modify the page. Using a simple test based on the local state of the page and on the visible date of the manager, it is possible to know if pending modifications are present on the page. This on-the-fly consistency management avoids unnecessary system calls when the program exhibits enough data locality on the processors. This situation often happens on OpenMP codes because the affinity between distributed data and distributed iteration space are not exploited by the compilers as it is the case in HPF using the owner-compute rule. This system is close to the write-notice propagation of TreadMarks, although in Mome the propagation of this information is not tied to synchronization requests.

5 Experiments

5.1 Simulated Code

This section shows the usefulness of some Mome features on the `tomcatv` parallel application. The program was transformed manually because no HPF compilation system is currently available for the Mome DSM. The modifications were limited to the mapping of the arrays on the DSM shared space, to the distribution of the iteration spaces on the processors and to the insertion of consistency requests before the loops.

During an external iteration of `tomcatv`, each data element of a two-dimensional grid is updated from the values of its neighbors in the previous iteration. Using HPF, all arrays of `tomcatv` are column-wise block distributed. Each processor reads one column updated by each of its two neighbors during each iteration.

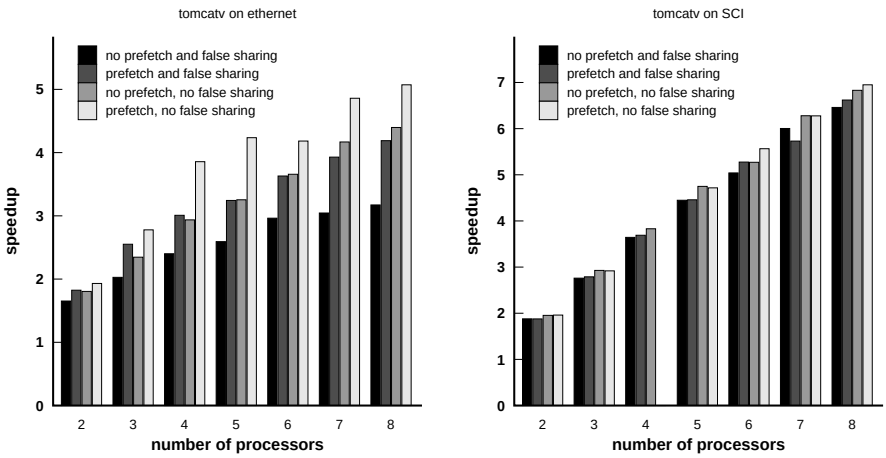

Figure1. Data prefetch on `tomcatv`

`tomcatv` was run on a 1023×1023 problem for the false sharing case and on a 1024×1024 problem when false sharing is avoided. The experimentation platform is a group of PCs connected through an SCI ring as well as a 100Mbits Ethernet network. Each node is a dual Pentium II processor running SMP Linux 2.2.7.

5.2 Data Prefetch

The experiments of Fig. 1 consider the application of the HPF owner-compute rule (consistency requests only on the pages containing shared data) combined with read prefetches. The prefetch operations are asynchronous and the computation is started before the requested data is present. Prefetching the left column of a block in `tomcatv` does not speedup the computation. But the prefetch request loads the right column of the block asynchronously during the block computation. With the SCI communication layer, the latency of page-fault resolution is low enough and the improvement is small. This is not the case for the Ethernet layer, as shown in Fig. 1. False sharing increases the latency of page-fault resolution because the modifications to the page must be gathered and merged. Prefetching in the presence of false sharing improves the speedup, mainly when the latency of the communications is high. The decision to insert prefetch requests is straightforward using the HPF compilation model. In fact, it is possible to systematically insert a read prefetch request everywhere the message passing version inserts a message receive. In general, prefetching pages used in the near future optimizes the execution. But prefetching unused pages can overload the system and delay the treatment of useful requests. Prefetching should be avoided in case of weak data analysis. In [6], M. Karlsson and P. Stenström propose a technique where prefetching is decided after analyzing the recent requests from the application. Such a pure run-time technique does not depend on the analysis ability of the compilers.

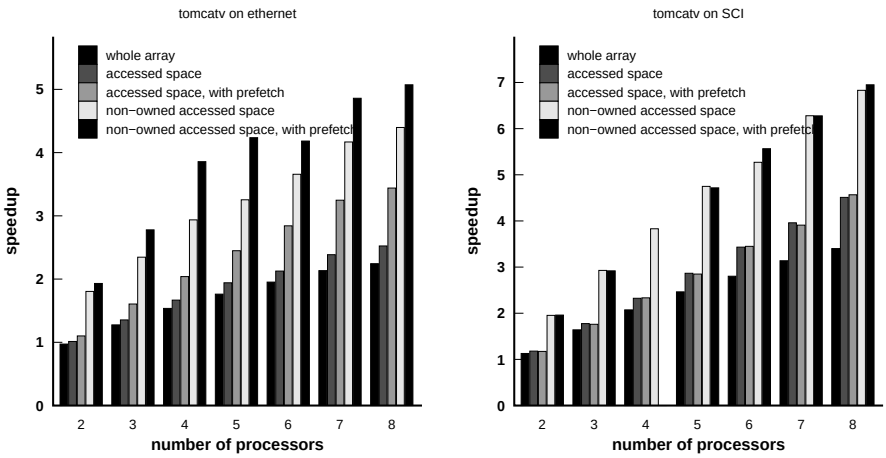

Figure2. Data synchronization on `tomcatv`

5.3 Consistency Management Strategy

In the experiments of Fig. 1, the consistency requests were generated for non-owned accessed data only. Such optimizations are not possible on OpenMP without a global analysis of the loops in the program. Figure 2 compares the execution speedups when the whole shared space, or the accessed space, or only the non-owned accessed space are synchronized. Using the current implementation of Mome, a page transaction costs a local thread synchronization and can generate a system call if the page must be unmapped. The on-the-fly optimization should avoid the page transactions on unused pages. During the loop execution, each accessed page produces a page-fault and generates communication with its page manager. In the experiments of Fig. 2, only the non-owned columns necessitate processor communication. Without prefetching, the performance for the case where only the accessed parts are synchronized is close to the performance of the case where the whole shared space is synchronized. In fact, the extra cost is due to the thread synchronizations only because the pages which are never accessed are not mapped in the application. Prefetching increases the efficiency of the execution on Ethernet. The performance difference between the accessed case and the non-owned case comes from the extra pages faults generated for the owned part of the array. This difference shows the benefits of a strict application of the owner-compute rule. This difference also shows the speedup expected from the integration of the on-the-fly consistency check: this check should avoid all transactions on the pages which are not modified by other processors.

5.4 Manager Distribution

Mome uses a directory-based management of the shared pages. When the owner-compute rule is applied, managing a page on the processor in charge of the data contained in this

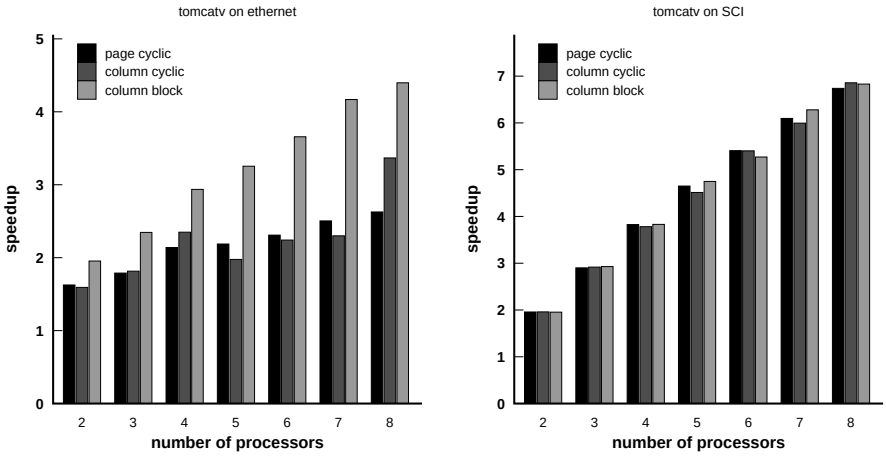

Figure3. Managers distribution on `tomcatv`

page should optimize the communications. OpenMP does not propose any directive for loop distribution control. However, for many parallel loops, it is possible to control the distribution of the iteration space through static scheduling. In this case, it is also possible to insert a call to the DSM interface and to redistribute the page managers according to this scheduling. The technique proposed by F. Mueller in [8] could also be applied at run-time for the case where the affinity between the iteration space and the data cannot be analyzed at compile-time. Figure 3 shows the performance of program executions using different manager distributions on Ethernet and on SCI. The page cyclic distribution is the initial page distribution of the Mome DSM. The latency using SCI is low enough and the performance does not depend on the distribution. But this is not the case for Ethernet. The latency of inter-processor communication directly affects the efficiency of page-fault resolution.

6 Related Work

Several papers describe the integration of run-time and compile-time approaches for the execution of parallelized codes on a software DSM. Dwarkadas et al. [5] augment the DSM interface with entry points which can exploit the message passing capabilities of the underlying hardware. Using these entry points the compiler can request for direct data transfers and reduce the overhead of run-time consistency maintenance. In our system, we have restricted the use of the information received from the compiler to consistency management. But all data transfers go through the DSM page migrations.

In [4], Chandra and Larus also consider explicit management of the shared data between the producers (writers) and the consumers (readers). Using the owner-compute rule, the owners explicitly update the local memories of the readers. This optimization

make the non-owned blocks available before the parallel loop executes, so that no access fault occurs during the loop. At the opposite of Chandra and Larus's proposition which deactivate the DSM consistency management in order to optimize, we consider a completely relaxed consistency model and the compiler inserts consistency requests only on the relevant memory sections.

H. Lu and Al. in [7] describe an implementation of OpenMP using the SUIF compiler on networks of workstation upon the TreadMarks DSM. Their implementations considers all parallel and synchronization features of OpenMP and is not limited to the exploitation of loop parallelism as in our experimentations.

7 Conclusion and Future Work

In this paper, we showed that it is possible to optimize the comportment of applications using a DSM from information extracted at compile-time. An HPF run-time system is currently being implemented upon the Mome DSM and should show the effectiveness of our approach for the execution of parallel codes on networks of workstations. The integration of on-the-fly consistency checks is also in progress and will reduce the cost of making consistent large amounts of shared data, as it is the case with OpenMP when no data access analysis is performed.

References

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

[2] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The midway distributed shared memory system. In *Proc. of the 38th IEEE Int'l Computer Conf. (COMPCON Spring'93)*, pages 528–537, February 1993.

[3] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Techniques for reducing consistency-related communication in distributed shared memory systems. *ACM Transactions on Computer Systems*, 13(3):205–243, August 1995.

[4] Satish Chandra and Larus James R. Optimizing communication in hpf programs on fine-grain distributed shared memory. In *6th ACM SIGPLAN Symposiun on Principles and Practice of Parallel Programming*, June 1977. Las Vegas, June 18-21.

[5] Sandhya Dwarkadas, Honghui Lu, Alan L. Cox, Ramakrishnan Rajomony, and Willy Zwaenepoel. Combining compile-time and run-time support for efficient software distributed shared memory. In *Proceedings of IEEE, Special Issue on Distributed Shared Memory*, volume 87, No 3, pages 467–475, March 1999.

[6] M. Karlsson and P. Stenström. Effectiveness of dynamic prefetching in multiple-writer distributed virtual shared memory system. *Journal of Parallel and Distributed Computing*, 43(2):79–93, July 1997.

[7] H. Lu, Y. C. Hu, and W. Zwaenepoel. Openmp on networks of workstations. In *Proceedings of Supercomputing'98*, 1998.

[8] F. Mueller. Adaptative dsm-runtime behavior via speculative data distribution. In J. Jose and al., editors, *Parallel and Distributed Processing – Workshop on Run-Time Systems for Parallel Programming*, volume 1586 of *LNCS*, pages 553–567. Springer, April 1999.