Optimizing Data-Parallel Programs for Clusters of SMPs[1]

Siegfried Benkner, Maria Lucka, Viera Sipkova

Institute for Software Science
University of Vienna
Austria

Abstract.

High Performance Fortran (HPF) is a high-level data-parallel programming language which has been designed mainly for distributed-memory parallel computers. We present extensions of HPF for clusters of shared-memory multiprocessors and outline a hybrid parallelization strategy, efficiently combining distributed-memory with shared-memory parallelism.

1 Introduction

Clusters of (symmetric) shared-memory multiprocessors (SMPs) have become the most promising parallel computing platforms for scientific computing. Examples of such systems include multiprocessor clusters from SUN, SGI, IBM, a variety of multi-processor PC clusters, supercomputers like the NEC SX-5 or the future Japanese Earth Simulator and the ASCI White machine. SMP clusters consist of a set of multi-processor compute nodes connected via a high-speed interconnection network. While processors within a node have direct access to a shared memory, accessing data located on other nodes has to be realized by means of message-passing. As a consequence, the complexity of user applications development is significantly increased, forcing programmers to deal with shared-memory programming issues such as multi-threading and synchronization, as well as with distributed-memory issues such as data distribution and message-passing communication.

There are mainly two trends in parallel programming, depending on how the address space of parallel systems is organized. On the one hand, the standard application programming interface (API) for message-passing, MPI[13], is widely used, mainly for distributed-memory systems. On the other hand, a standard API for shared-memory parallel programming, OpenMP[15], has recently become available. While OpenMP is restricted to shared-memory architectures only, MPI programs can also be executed on shared-memory machines and clusters. However, MPI programs that are executed on clusters of SMPs usually do not directly utilize the shared-memory available within nodes and thus may miss a number of optimization opportunities.

A promising approach for parallel programming attempts to combine MPI and OpenMP in a single application [6][9][10]. Such a strategy attempts to fully exploit the potential of SMP clusters by relying on data distribution and explicit message-passing between the nodes of a cluster, and on shared-memory and multi-threading within the nodes. While such an approach allows optimizing parallel programs by taking the hybrid architecture

[1] This work was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund and by NEC Europe Ltd. as part of the ADVICE project in cooperation with the C&C Research Laboratories, St. Augustin, Germany

of SMP clusters into account, applications written in such a way tend to become extremely complex.

In contrast to MPI and OpenMP, High Performance Fortran (HPF) [11] is a high-level parallel programming language which can be employed on both distributed-memory and shared-memory machines. Although HPF programs can be compiled for clusters of SMPs, the language does not provide features for exploiting the hierarchical structure of clusters. As a consequence, current HPF compilers usually ignore the shared-memory aspect of SMP clusters and treat such machines as distributed-memory systems.

In order to overcome these shortcomings of HPF and its compilers, we propose extensions of the HPF mapping mechanisms such that the hierarchical structure of SMP clusters can be taken into account. Based on these features, an HPF compiler can then adopt a hybrid parallelization strategy whereby distributed-memory parallelism based on message-passing, e.g. MPI, is exploited across the nodes of a cluster, while shared-memory parallelism is exploited within SMP nodes by relying on multi-threading, e.g. OpenMP.

The rest of this paper is organized as follows. Section 2 proposes language features for optimizing existing HPF programs for clusters of SMPs by providing an explicit specification of the hierarchical structure of clusters. Section 3 sketches a hierarchical HPF compilation and execution model for clusters of SMPs. Experimental performance results are presented in Section 4. A discussion of related work and concluding remarks are provided in Sections 5 and 6, respectively.

2 Exploiting the Hierarchical Structure of SMP Clusters

HPF provides the concept of abstract processor arrangements for establishing an abstraction of the parallel target architecture in the form of one or more rectilinear processor arrays. Processor arrays are utilized within data distribution directives to describe a mapping of array elements to abstract processors. Array elements mapped to an abstract processor are *owned* by that processor. *Ownership* of data is the central concept for the execution of data parallel programs. Based on the ownership of data, the distribution of computations to abstract processors and the necessary communication and synchronization are derived automatically.

Consider now an SMP cluster consisting of `NN` nodes, each equipped with `NPN` processors. Currently, if an HPF program is targeted to an SMP cluster, abstract processors are either associated with the `NN` nodes of the cluster or with the `NN*NPN` processors. In the first case, data arrays are distributed only across the `NN` nodes of the cluster and therefore only parallelism of degree `NN` can be exploited. In the second case, where abstract HPF processors are associated with the processors of a cluster, potential parallelism of degree `NN*NPN` can be exploited. However, by viewing an SMP cluster as a distributed-memory machine consisting of `NN*NPN` processors, the shared-memory available within nodes is usually not exploited, since data distribution and communication are performed within nodes as well.

2.1 Processor Mappings

The concept of *processor mappings* is introduced for describing the hierarchical structure of SMP clusters. A processor mapping specifies a mapping of an abstract processor array to an abstract node array. The `NODES` directive is introduced for declaring one or more abstract node arrays. Processor mappings are specified by using a subset of the HPF distribution mechanisms. For homogeneous clusters, the usual HPF `block` distribution format may be used. Heterogeneous clusters can be supported by means of the `gen_block` distribution format of the Approved Extensions of HPF. In order to support abstract node arrays whose sizes are determined upon start of a program, the new intrinsic function `number_of_nodes()` is provided, which returns the actual number of nodes used to execute a program. Examples of processor mappings are shown in Figure 1.

Processor mappings provide a simple means for optimizing HPF applications for SMP clusters. Using processor mappings, the hierarchical structure of SMP clusters may be specified, without the need to change existing HPF directives. Based on processor mappings, an HPF compiler can adopt a cluster-specific parallelization strategy, which exploits distributed-memory parallelism across the nodes of a cluster, and shared-memory parallelism within nodes. A full specification of HPF extensions for SMP clusters can be found in [3].

```
!hpf$ processors P(8)
!hpf$ nodes N(4)
!hpf$ distribute P(block) onto N

      real A(NA)
!hpf$ distribute A(block) onto P
```
(a)

```
!hpf$ processors R(4,8)
!hpf$ nodes M(4)
!hpf$ distribute R(*,block) onto M

      real B(NB)
!hpf$ distribute B(cyclic,block) onto R
```
(b)

Figure 1: Examples of processor mappings. In Figure (a) the processor mapping specifies a 4x2 SMP cluster, while Figure (b) specifies a cluster of 4 nodes each with 8 processors arranged in a 4x2 configuration.

2.2 Exploiting DM and SM Parallelism

If a dimension of an abstract processor array is distributed by `block` or `gen_block`, contiguous blocks of processors are mapped to the nodes in the corresponding dimension of the specified abstract node array. As a consequence of such a processor mapping, both distributed-memory parallelism and shared-memory parallelism may be exploited for all data array dimensions that are mapped to that processor array dimension. On the other hand, if in a processor mapping a dimension of an abstract processor array is distributed by means of "*", all abstract processors in that dimension are mapped to the same node of an abstract node array, and thus only shared-memory parallelism may be exploited across array dimensions which have been mapped to that processor array dimension.

In Figure 1 (a), both distributed- and shared-memory parallelism may be exploited for array `A`. In Figure 1 (b), only shared-memory parallelism may be exploited across the first dimension of array `B`, while both shared-memory and distributed-memory parallelism may be exploited across the second dimension.

2.3 Inter- and Intra-Node Data Mappings

An HPF data mapping directive, e.g. `distribute A(block) onto P`, determines for each processor of the abstract processor array `P` those parts of the data array `A` that are owned by it. If, in addition, a processor mapping for `P` with respect to a node array `N` is specified, an *inter-node data mapping* and an *intra-node data mapping* may be automatically derived from the data mapping and the processor mapping. Inter-node data mappings control distributed-memory parallelization across the nodes of a cluster while intra-node mappings control shared-memory parallelization within nodes.

An *inter-node data mapping* determines for each node of the node array those parts of array `A` that are owned it. The implicit assumption is that those portions of an array owned by a node are allocated in an unpartitioned way in the shared memory of this node, while data is distributed across the local memory of nodes, according to the derived inter-node mapping.

Intra-node data mappings specify a mapping of the data allocated on a node of a cluster with respect to the processors within a node. Intra-node data mappings are utilized by the compiler to control the exploitation of thread parallelism within SMP nodes.

3 Hierarchical Compilation and Execution Model

In this section we sketch a hierarchical compilation and execution model for clusters of SMPs which is adopted for the parallelization of HPF programs that utilize the proposed language extensions. This model has been realized within the VFC compiler [1], a source-to-source translation systems which transforms HPF programs into explicitly parallel Fortran programs. As opposed to the usual HPF compilation [4], where a single-threaded SPMD node program is generated, a multi-threaded node program is generated under the hierarchical execution model. An extended HPF program is compiled by VFC for clusters of SMPs as follows.

First, VFC analyzes data mappings and processor mappings in order to derive for each array dimension an inter-node mapping and an intra-node data mapping. During this analysis, each array dimension is classified as DM, SM, or DM/SM, depending on the type of parallelism that may be exploited across this dimension, or as SEQ, if the array dimension is replicated. This information is propagated to parallel loops which are classified accordingly.

Second, in the distributed-memory parallelization phase, VFC translates the original HPF program into an SPMD message-passing node program by distributing data and work to the nodes of the cluster and inserting MPI message-passing calls in order to communicate non-local data between nodes. This parallelization phase is controlled by the inter-node data mappings.

Third, the intermediate SPMD program is parallelized for shared-memory according to the intra-node data mapping derived by the compiler. Those loops which have been classified as DM/SM or SM are transformed in such a way that additional thread parallelism within the nodes is exploited by inserting corresponding OpenMP directives. Work distribution of loops and array assignments is derived from the intra-node data mapping of the accessed arrays and realized by corresponding OpenMP work-sharing constructs and/or appropriate loop transformations (e.g. strip-mining). Data consistency of shared data objects is enforced by inserting appropriate OpenMP synchronization primitives [2]. This shared-memory parallelization phase is driven by the intra-node data mappings.

Finally, the parallel MPI/OpenMP code generated by the VFC compiler is compiled with an OpenMP Fortran compiler and linked with the MPI library.

HPF programs compiled for clusters of SMPs as outlined above, are executed according to a *hierarchical execution model*. Within this model an HPF program is executed in parallel by a set of MPI processes, each of which executes on a separate node of an SMP cluster within its own local address space. Process parallelism, data partitioning, and message-passing communication based on MPI is utilized across nodes. Each node process generates a set of threads which emulate the abstract processors mapped to a node and which execute concurrently in the shared address space of a node. The data mapped to one node is allocated in a non-partitioned way in the shared memory, regardless of the intra-node mapping. Parallel execution of threads within nodes is organized on the basis of the derived intra-node data mapping, which controls the distribution of computations among the threads. Consistency of shared data objects is guaranteed by automatically generated synchronization primitives [2].

4. Experimental Results

In this section we report on early experiments with the proposed language extensions and the hierarchical compilation and execution model on a Beowulf cluster consisting of 8 nodes, each equipped with two Pentium II (400MHz) processors, connected by Fast Ethernet. For the performance experiments, we used an HPF kernel from a numerical pricing module developed in the context of the AURORA Financial Management System [8]. This kernel realizes a backward induction algorithm on a Hull and White interest rate tree. In the HPF code the interest rate tree is represented by 2-dimensional arrays and several index vectors which capture the structure of the tree. All these arrays have been distributed by usual HPF distribution mechanisms. The main computations are performed in an inner `INDEPENDENT` loop with indirect data accesses, which operates on a single level of the Hull and White tree. Due to the peculiarities of the algorithm, communication is required for each level of the tree, introducing a significant communication overhead.

The performance experiments were performed for a tree with 3650 levels. We compared the original HPF kernel to an extended kernel, where an additional processor mapping was used for specifying the hierarchical structure of the PC cluster. The original HPF kernel was compiled by VFC to Fortran/MPI, while the extended kernel (HPF-Cluster) was compiled to Fortran/MPI/OpenMP and executed according to the hierarchical execution model combining MPI process parallelism with OpenMP thread parallelism. Both kernels were measured on up 8 nodes, utilizing always both processors of a node.

As Figure 2 shows, the HPF-Cluster version significantly outperforms the original HPF kernel. The major reason for this performance difference seems to be the communication overhead induced by MPI communication[2] within the nodes of the cluster.

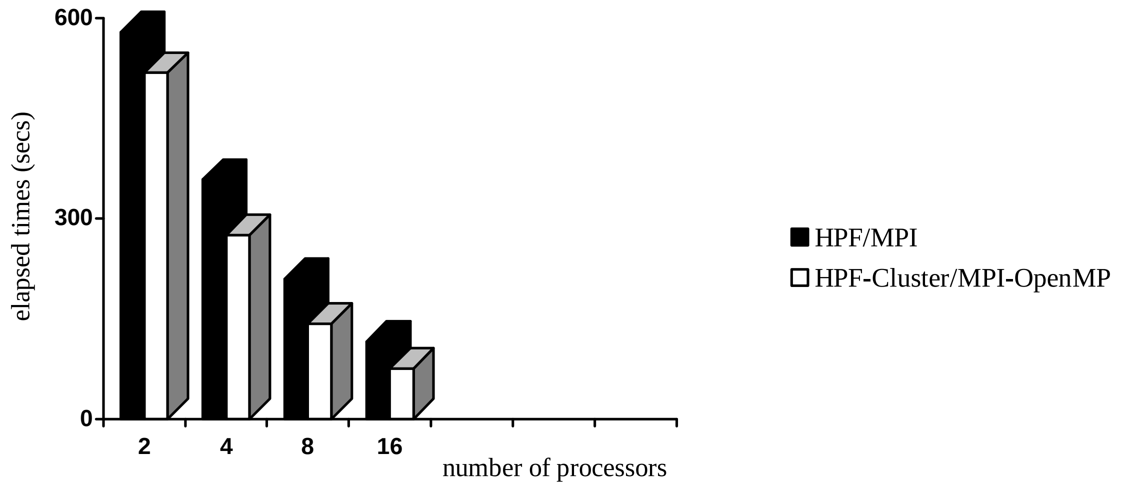

Figure 2. Experimental results for a financial optimization kernel on a Beowulf cluster with dual-processor nodes.

5 Related Work

Several researchers have worked on high-level parallel programming support for SMP clusters.

MPISpro Power Fortran from SGI offers OpenMP with HPF-like data placement directives [14]. Compaq has added a set of new directives to its Fortran for Tru64 UNIX that extend the OpenMP Fortran API by features for controlling the placement of data in memory and the placement of computations that operate on that data [5]. Chapman, Mehrotra and Zima [7] propose a set of OpenMP extensions, similar to HPF mapping directives, for locality control. Portland Group, Inc. proposes a high-level programming model [12] that extends OpenMP with additional data mapping directives, library routines and environment variables for controlling data locality with respect to the nodes of SMP clusters.

[2] The MPI version used in this evaluation did not offer optimized communication primitivies via shared memory for intra-node communication.

All these approaches introduce data mapping features into OpenMP in order to control locality, but still utilize the explicit work distribution via OpenMP directives. Our approach is based on HPF and relies on an implicit work distribution which is derived automatically from the data mapping.

6 Conclusions

Processor mappings provide a simple means for optimizing existing HPF applications for SMP clusters. Using processor mappings the hierarchical structure of SMP clusters may be explicitly specified, without the need to change existing HPF directives. Based on processor mappings, an HPF compiler can adopt a cluster-specific parallelization strategy in order to efficiently exploit distributed-memory parallelism across the nodes of a cluster, and shared-memory parallelism within nodes. Such a parallelization strategy has been realized within the VFC compiler based on a hierarchical compilation and execution model that combines MPI processes parallelism OpenMP thread parallelism.

Experimental results indicate that a hybrid parallelization strategy combining distributed and shared-memory parallelism has the potential to outperform a strategy based on pure message-passing. A more detailed evaluation of our approach and the investigation of its relevance for a larger range of scientific applications will be the subject of future work.

Bibliography

[1] S. Benkner. VFC: The Vienna Fortran Compiler. *Scientific Programming*, 7(1):67--81, 1999.

[2] S. Benkner and T. Brandes. Exploiting Data Locality on Scalable Shared Memory Machines with Data Parallel Programs. *Proceedings Euro-Par 2000 Parallel Processing*, Munich, Springer Verlag, 2000.

[3] S. Benkner and T. Brandes. HPF Extensions for Clusters of SMPs.Technical Report, ADVANCE Deliverable of the NEC/Uni Vienna Co-operation ADVANCE, University of Vienna, March 2001.

[4] S. Benkner and H. Zima. Compiling High Performance Fortran for Distributed Memory Architectures. *Parallel Computing* 25 (13-14), pp. 1785-1825, Elsevier Science B.V., 1999.

[5] J. Bircsak, P. Craig, R. Crowell, Z. Cvetanovic, J. Harris, C. Nelson, and C. Offner. Extending OpenMP for NUMA Machines. *Proceedings of SC 2000*, Dallas, November 2000.

[6] F. Cappello and D. Etieble. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. *Proceedings of SC 2000: High Performance Networking and Computing Conference*, Dallas, Nov. 2000.

[7] B. Chapman, P. Mehrotra, H. Zima. Enhancing OpenMP with Features for Locality Control. *Proceedings ECWMF Workshop Towards Teracomputing - The Use of Parallel Processors in Meteorology*, 1998.

[8] E. Dockner, H. Moritsch, G. Pflug, and A. Swietanowski. AURORA financial management system: From Model Design to Implementation. *Technical report AURORA* TR1998-08, University of Vienna, June 1998.

[9] O. Haan. Matrix Transpose with Hybrid OpenMP / MPI Parallelization. Technical Report, Presentation given at SCICOMP 2000, http://www.spscicomp.org/2000/userpres.html#haan, 2000.

[10] D. S. Henty. Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling. *Proceedings of SC 2000*, Dallas, November 2000.

[11] High Performance Fortran Forum. High Performance Fortran Language Specification. Version 2.0, Department of Computer Science, Rice University, January 1997.

[12] M. Leair, J. Merlin, S. Nakamoto, V. Schuster, and M. Wolfe. Distributed OMP - A Programming Model for SMP Clusters. *International Workshop on Compilers for Parallel Computers*, Aussois, France, 2000.

[13] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Vers. 1.1, June 1995.

[14] Silicon Graphics Inc. MIPSpro Power Fortran 77 Programmer's Guide: OpenMP Multiprocessing Directives. Technical Report Document 007-2361-007, 1999.

[15] The OpenMP Forum. OpenMP Fortran Application Program Interface. Version 1.1, November 1999. http://www.openmp.org.