Tools for Development of Programs for a Cluster of Shared Memory Multiprocessors

B. Chapman, J. Merlin, D. Pritchard
Dept. of Electronics & Computer Science
University of Southampton
Southampton, United Kingdom

F. Bodin, Y. Mevel
IRISA
University of Rennes
Rennes, France

T. Sørevik
Institute for Informatics
University of Bergen
Bergen, Norway

L. Hill
Simulog SA
Sophia Antipolis
France

Abstract *Applications are increasingly being executed on computational systems that have hierarchical parallelism. There are several programming paradigms which may be used to adapt a program for execution in such an environment. In this paper, we outline some of the challenges in porting codes to such systems, and describe a programming environment that we are creating to support the migration of sequential and MPI code to a cluster of shared memory parallel systems, where the target program may include MPI, OpenMP or both.*

Keywords: parallel programming, distributed shared memory, parallelization, program transformations, program development environment

1 Introduction

Most major hardware vendors market cost-effective parallel systems in which a modest number of processors share memory. Such shared memory parallel workstations (SMPs) are being increasingly deployed, not only as stand-alone computers, but also in workstation clusters. They are also used to populate the nodes of distributed memory machines, e.g. IBM's SP-2, Quadrics QM-1. The SGI Origin 2000 and HP SP2xxx systems provide cache coherency across their SMP nodes and have very low latency. Users are encouraged to program the entire system as if it were a large SMP. Within the US ASCII project, large-scale platforms are being constructed with a similar hierarchy of parallelism. At the other end of the scale, we expect to see a proliferation of workstations and PCs with shared memory, and clusters of such machines being used to provide low cost high performance computing.

Current programming models have not been developed with hierarchies of parallelism in mind. Moreover, there are vast differences in the computational power, interconnection technology and system software of the systems mentioned above. It is thus impossible to recommend a single programming approach for all of them. The relative cost of remote data accesses, for example, may significantly affect the overall program development strategy. Some of the issues involved at the highest end are discussed by Keyes et al. in [11].

We are creating a programming environment to facilitate application development for the range of systems described above. We are studying alternative programming models for SMP clusters in order to better understand the issues involved. When completed, the **POST environment** will support the creation of programs using MPI and OpenMP, either alone or in combination, to target such systems. The environment will be highly interactive and will cooperate with the user to develop code rather than provide a high degree of automation. POST will help the user analyze an existing program before adapting it; yet it will also apply Case-Based Reasoning techniques to derive a successful adaptation strategy from a database of known strategies.

This paper is organised as follows. In the next section we discuss potential programming models for clusters of SMPs. We then describe experiments and early work using such models, before introducing the features of our environment in Section 4. Related work is outlined and our efforts are then summarized.

2 Programming SMP Clusters

For the purposes of this paper we consider each of the systems described above to be a cluster of SMPs, where each SMP is a *node* in the cluster. Program performance on such systems will be affected by all of the system features mentioned in the previous section.

Available parallel programming models include MPI, for explicit message-passing programming, HPF for higher-level distributed memory programming, and OpenMP for multi-threaded, shared memory programming.

MPI is the most flexible and general of these, and it *can* be used to program such a system. However, it does not directly support the exploitation of shared memory, and processes executing within a common shared memory environment must communicate explicitly. Simple experiments on a Quadrics QM-1 with SMP nodes showed that current MPI libraries exchange data faster between processors on different nodes than on the same node [14]. HPF has benefits in terms of software maintenance, but is less general and also does not provide features for shared memory programming.

OpenMP has been designed explicitly for shared memory parallel programming and realises this at an appropriately high level. Two features make it particularly interesting for SMP clusters: firstly, it permits nested parallelism, and secondly, it facilitates the specification of coarse grain parallel regions. However, OpenMP can only be deployed on a cluster of SMPs if there is software support for cache coherency across the cluster.

Where this is not the case, MPI may be used either alone or in conjunction with OpenMP. The latter option permits direct exploitation of the architectural features of the system. It is also possible to interface HPF with OpenMP, and we are investigating the viability of this high-level alternative [7].

3 Some Experiments

To date, most of our experience in using OpenMP has been on clusters of medium-sized systems and on the Origin 2000. Although the latter differs substantially from the lower-cost SMP workstation clusters, it has permitted us to experiment with both MPI and OpenMP, and with SGI native constructs for scheduling parallel loops, and allocating data and threads.

In one such experiment, the 3400-line NAS-BT benchmark was ported to OpenMP and evaluated on a 128-node Origin 2000 [2]. At each time step it solves a large system of algebraic equations iteratively using the SSOR method[1]. The righthand side is updated in six separate, independent routines: it is easy to parallelize with a `PARALLEL SECTIONS` directive, needing a minor code modification to enable the use of an OpenMP reduction operation. But this strategy does not provide work for more than six processors. Since each section contains a large amount of fine grained parallelism, OpenMP's nested parallelism could remedy this: unfortunately, it was not implemented in our compiler. Worse still, data accesses differed in the individual sections, and the load balance was thus not good, even on exactly six threads. Timings are in [2].

The loops performing grid point updates could be fully parallelized. Since iterations performed identical work, it was expected that an even distribution of iterations to threads would

[1] A new version is under development at NASA Ames which is better suited to OpenMP parallelization.

lead to a well-balanced computation. This was not the case: two of the threads consistently outperformed the others. The speedier threads were located on the same node as the data! Several experiments, by ourselves and others, confirm the importance of coordinating the placement of work and data on the Origin 2000. The remedies which are available on this system are *not* features of OpenMP. On SMP clusters with longer latencies this will be even more important.

MPI and OpenMP have been successfully used together in several notable experiments, including one award-winning computation [4], and are becoming an accepted combination, especially for running large programs on SMP clusters, e.g. [8]. We discuss two examples.

Colleagues at Bergen have parallelized a 2D seismic model for the Origin 2000 using both MPI and OpenMP [9]. In this case, the problem required parallelization of a number of shots, which were independent except for a global reduction terminating the computation. This was easy under MPI, but the maximum number of shots was 30. Since each shot contained much internal parallelism, OpenMP was used to parallelize the process code. As a result, the work for each shot scaled up to about 20 processors. With applications of this nature, it is relatively easy to create an outer layer of MPI code and then use OpenMP to exploit fine grain parallelism.

MICOM is an Atlantic ocean model code with roughly 20000 lines of code and about 600 loop nests. A message passing version of this program has been further adapted to run on clusters of SMPs [18]. SC-MICOM is an MPI program which starts up a number of OpenMP threads on each executing node. It realizes a hierarchical data distribution strategy matching the cluster architecture. MPI communication is required to exchange boundary values at the global level.

4 The POST Development Environment

The POST program development environment is an interactive system which provides Fortran program analysis and transformations to support parallelisation with MPI library calls and/or OpenMP directives.

At the heart of the environment is FORESYS [19], a commercial Fortran source code analysis and restructuring system. It is an interactive system which is used for reverse engineering and upgrading of existing software, and for validation and quality assurance of new code. It performs an in-depth interprocedural program analysis and can display a variety of information, such as the call graph, data flow and data dependence information. It links graphical representations with source text displays wherever appropriate—for example, clicking on a call graph node displays the source code of the corresponding program unit —and enables rapid navigation of the source code. Some of its advanced display functionality was originally realised in the ANALYST prototype [6] and incorporated into FORESYS in the ESPRIT project FITS [5].

Our implementation work is realized by TSF [3], a FORESYS extension that provides access to its abstract syntax tree and the results of restructuring analysis via a custom scripting language. It can extract and insert information into the FORESYS program database, and cause it to derive additional data.

The POST environment provides several levels of support for code adaptation, which we now describe. All features described here are realized via FORESYS and TSF.

4.1 Basic Level of Support

At the lowest level, the environment helps a user gain an overview of the source program, which may already have parallel constructs. Support is provided for inserting MPI or OpenMP parallel constructs, in the form of templates. Syntactic checking ensures that environment variables have been inserted (cf. the

```
SCRIPT insert_CALL()
 . . . . . . .
IF (NOTUSED("mpi_ierr")) THEN // exists?
 DECLARE ("include 'mpif.h'") // no: add
 DECLARE ("integer mpi_ierr") // declare
ENDIF
IF (NOTUSED("mpi_rank")) THEN
 DECLARE ("integer mpi_rank")
ENDIF
 . . . . . . .
ENDSCRIPT
```

Figure 1: TSF script example.

TSF script in Figure 1), and that matching constructs, such as the `END` of a parallel region, have been properly defined. Other functionality includes the following:

Analysis: One can **browse** and **edit** the source code using graphical and text displays. Various kinds of global and local information on data structures, data flow and aliasing are available, and may be presented within a graph or by highlighting and colour-coding in the source text display. Data dependence graphs may be used to select loops for OpenMP parallelization.

Data **locality analysis** will show the relative usage of array dimensions, including those with indirect accesses.

We intend to provide support for analyzing a program's I/O structure, since it may need to be reorganized as part of the parallelization process, as well as analysis aimed at reducing synchronisation points, e.g. by inserting OpenMP `NOWAIT` clauses where possible.

Program structure display: Displays are being developed to show MPI processes and their interactions, as well as the scope of OpenMP parallel regions. The latter is needed because work sharing constructs may be placed inside procedures called from within parallel regions.

Transformations: A set of program transformations is provided, including loop distribution, loop fusion, loop interchange and array expansion, which may be applied by interactively by the user.

An algorithm is under construction to automatically parallelize selected loops under OpenMP. It will use interprocedural analysis to privatize loop variables.

4.2 Higher Levels of Support

The POST environment provides features which go beyond the capabilities mentioned above.

4.2.1 Guidance in Applying Transformations

We provide improved support for the task of creating OpenMP code by developing a feature to guide the user in this task. This includes a profile option to help detect the most time-consuming regions of a program, and also to determine whether there are performance problems associated with memory usage. Profiling results show what fraction of the data accessed by a loop was in cache.

We will suggest code improvements to the user in order to enable better automatic parallelization, or will propose pre-defined sequences of transformations to improve upon initial parallelization. In order to do so, we are studying the effect of a variety of transformations, and their combinations on selected target architectures [20].

4.2.2 User-Defined Transformations

The TSF scripting language is provided as part of the POST environment. Since it enables users to create new transformations, the system is extensible. An application developer may perform a variety of program modifications, including highly repetitive tasks, via TSF scripts.

We briefly illustrate their use by showing how they enabled a fast MPI parallelization

```
   DO 50 K = 1, KK
    KNM = K + KK*(NM-1)
    DO 50 J = 1, JJ
     DO 60 L = 1, ISU(J) !iterate over sea
      DO 60 I = IFU(J,L),ILU(J,L) !sea coords
       PU(I,J,K+1) = PU(I,J,K)+DPU(I,J,KNM)
60   CONTINUE
     DO 70 L = 1, ISV(J)
      DO 70 I = IFV(J,L), ILV(J,L)
       PV(I,J,K+1) = PV(I,J,K)+DPV(I,J,KNM)
70   CONTINUE
50 CONTINUE
```

Figure 2: MICOM: example loop

of the sequential MICOM code (see Section 3). Figure 2 shows a typical loop structure. Loops J, L, and I scan only those coordinates that correspond to the Atlantic ocean, and skip land areas. Loop bounds are read from pre-defined data structures. We distribute arrays by block in the first two dimensions, and may parallelize the loop nest in both J and I dimensions. Transforming the code requires adapting array declarations, ensuring that the local size includes overlap regions, modifying assignments and modifying loop bounds, where some redundant computations are performed to reduce communication. MPI communication calls are inserted.

A small set of TSF scripts was written to perform these changes. This effort took just a few days and saved weeks of manual adaptation. Results for our example loop are shown in Figure 3.

Additional benefits of using scripts are that it can reduce the likelihood of programmer error and that it enables the same strategy to be applied to an upgraded sequential version with minimal effort.

4.2.3 Case-Based Reasoning

Another component of the environment will provide an alternative approach to program transformation. It aims to capitalize on, and facilitate the reuse of, the experience of program developers.

```
   do 50 K = 1,KK
    KNM = K + KK*(NM-1)
    do 50 J = 1-margin,jj+margin
     do 60 L = 1,ISU(J)
      do 60 I = max(1-margin,IFU(J,L)),
  &                 min(ii+margin,ILU(J,L))
       PU(I,J,K+1) = PU(I,J,K)+DPU(I,J,KNM)
60   continue
     do 70 L = 1,ISV(J)
      do 70 I = max(1-margin,IFV(J,L)),
  &                 min(ii+margin,ILV(J,L))
       PV(I,J,K+1) = PV(I,J,K)+DPV(I,J,KNM)
70   continue
50 continue
```

Figure 3: MICOM loop; result of TSF scripts

The expert knowledge associated with parallelization is not well structured and does not lend itself to conventional rule-based systems. In contrast, the Case-Based Reasoning approach [12] aims to solve a given problem by adapting the solution of a similar one already encountered. Its application in our context requires us to *identify* code fragments for investigation: these can be obtained by profiling. In order to *retrieve* similar stored cases, a notion of similarity must be defined and used to compare a code fragment with stored examples. Its definition may be context-dependent. Finding a suitable definition is hard: a preliminary study [16] reports good results with abstractions based upon syntax, data dependence, control flow, etc. The solution adopted in the retrieved code may be automatically *reused*, if it consists of predefined automatic transformations, or it may need to be performed by the user. Finally, it should be possible to *retain* the new solution.

In our environment, a case is a pair C = (*Problem, Solution*) where *Problem* is a transformation problem and *Solution* is a sequence of transformations which represent its solution. In practice, C is defined as a set of fields. The *initial properties* define the context of a case, including the target architecture, the name of the transformation, and the application domain. The *final properties* specify properties of the code after applying the solution. The *original code fragment* is the state before the proposed solution is implemented. The *code pattern* describes code fragments on which the solution can be applied. It is used to identify cases that are similar to the original code fragment. A *list of program transformations* is given, which is to be applied to the original code fragment. Finally, we require a *documentation* of the solution, which informs the user aware of the usage of, and any restrictions to, the proposed solution. This is especially important in our environment, which largely gives control to the user.

Many applications are possible. This approach may be suitable, for example, if a given loop performs a parallel array transpose inefficiently. The facility would adapt a good transpose algorithm for the given case. It may also enable the parallelization of loops that are too hard for the automatic system, e.g. if they require handling arithmetic series.

5 Related Work

In addition to vendor-supplied parallel program development toolsets, there are a range of products from individual vendors to support the creation of MPI or OpenMP code.

These include the Kuck and Associates product line [13] for shared memory parallelisation. Their *Visual KAP for OpenMP* operates as a companion preprocessor for the Digital Visual Fortran compiler. Although it has many user-controlled options, it essentially functions as a batch tool on the source files. The *FORGE* parallelization tools [1] from Applied Parallel Research enable source code editing and offer control and data flow information, in combination with an HPF compiler. *VAST* [17] from Pacific Sierra Research helps a user migrate from Fortran 77 to HPF code by automatic code cleanup and translation. We are not aware of tools which explicitly support the development of programs that combine the OpenMP and MPI paradigms.

Interactive display of analysis results and the provision of user transformations was pioneered in the *ParaScope Editor*, for improving shared memory programs. *CAPTools* [10] interacts with the user on the basis of a program data base, which the user may query and modify, in order to improve the automatic transformation of a sequential program to an MPI one. Finally, *SUIF Explorer* [15] supports parallelization by identifying the most important loops via profiling. It takes a different approach from POST to help the user perform this task, for example, applying program slicing to present information on dependence-causing variables.

6 Conclusions and Future Work

In this paper we have described a portable and extensible programming environment for the development of parallel Fortran applications under MPI and OpenMP. We provide help for a range of different programming approaches, and at different levels of sophistication. This task is challenging, since many constructs must be supported. The capabilities of the underlying system offer the opportunity to explore a variety of approaches to supporting parallelisation, including interactive guidance for the process of program transformation.

We have begun to work with applications developers at several industrial sites in Europe in order to improve the functionality and features of our environment, and in order to obtain material for the case-based reasoning system.

Our programming model is far from ideal. In particular, the use of a combined MPI-OpenMP approach requires the application developer to deal with two distinct and demanding programming models. A number of researchers are working on extending the OpenMP approach so that it may be better suited to hierarchical parallel systems. We hope to contribute to that search. Other research focuses on the task of providing cache-coherency across a range of different systems.

Acknowledgments

This work is part-funded by the ESPRIT project 29920, POST (Programming with the OpenMP STandard), whose name we have borrowed.

We thank Jerry Yan at NASA Ames for discussions on a number of related topics, including the NAS benchmarks, and David Lancaster at Southampton University for discussions and for providing benchmark results. We also thank Wolfgang Nagel and colleagues at the TU Dresden, who extended their VAMPIR performance analysis tool to enable closer investigation of the behaviour of some of the OpenMP codes.

References

[1] Applied Parallel Research: *APR's FORGE 90 parallelization tools for High Performance Fortran.* APR, June 1993

[2] R. Blikberg and T. Sørevik. *Early experiences with OpenMP on the Origin 2000.* Proc. European Cray MPP meeting, Munich, Sept 1998

[3] F. Bodin, Y. Mevel, and R. Quiniou. *A user level program transformation tool,* Proc. Int. Conference on Supercomputing, 1998

[4] S.W. Bova, C.P. Breshears, C. Cuicchi, Z. Demirilek and H.A. Gabb. *Dual Level Parallel Analysis of Harbor Wave Response Using MPI and OpenMP.* Proc. Supercomputing '98, Orlando, 1998

[5] B. Chapman, F. Bodin, L. Hill, J. Merlin, G. Viland and F. Wollenweber. *FITS—A Light-Weight Integrated Programming Environment.* To appear in Proc. Europar '99, Toulouse, 1999

[6] B. Chapman and M. Egg. *ANALYST: Tool Support for the Migration of Fortran Applications to Parallel Systems,* Proc. PDPTA' 97, Las Vegas, June 30–July 3, 1997

[7] B. Chapman and P. Mehrotra. *OpenMP and HPF: Integrating Two Paradigms.* Proc. Europar '98, Southampton, 1998

[8] T. Faulkner. *Performance Implications of Process and Memory Placement using a Multi-Level Parallel Programming Model on the Cray Origin 2000.* Available at URL `www.nas.nasa.gov/~faulkner`

[9] L. A. Frøyland, F. Manne and Norunn Skjei. *2D seismic modelling on the Cray Origin 2000.* Internal report 1998-02-13 for Norsk Hydro. In Norwegian

[10] C.S. Ierotheou, S.P. Johnson, M. Cross and P.F. Leggett. *Computer aided parallelisation tools (CAPTools)—Conceptual overview and performance on the parallelisation of structured mesh codes,* Parallel Computing, 22(2), 163–195, 1996

[11] D. Keyes, D. Kaushik and B. Smith. *Prospects for CFD on Petafloops Systems.* ICASE Report No. 97-37, NASA Langley Research Center, 1997

[12] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993

[13] Kuck and Associates. *KAP/Pro Toolset for OpenMP.* See `www.kai.com/kpts/`

[14] David Lancaster. *Results of GENESIS benchmark experiments.* Available at URL `http://gather.ecs.soton.ac.uk/`

[15] S.-W. Liao, A. Diwan, R.B. Bosh Jr., A. Ghuloum, M. S. Lam. *SUIF Explorer: An interactive and interprocedural parallelizer.* Technical Report, Computer Systems Lab, Stanford University, 1998

[16] Y. Mevel *Environnement pour le portage de code oriente performance sur machines paralleles et monoprocesseurs.* Ph.D. Thesis, University of Rennes, France, 1999

[17] C. Rodden and B. Brode. *VAST/Parallel: automatic parallelisation for SMP systems.* Pacific Sierra Research Corp., 1998.

[18] A. Sawdey. SC-MICOM. Software and documentation available from `ftp://ftp-mount.ee.umn.edu/pub/ocean/`

[19] Simulog SA, *FORESYS, FORtran Engineering SYStem, Reference Manual V1.5*, Simulog, Guyancourt, France, 1996

[20] S.K. Singhai and K.S. McKinley. *A Parametrized Loop Fusion Algorithm for Improving Parallelism and Cache Locality.* Computer Journal, 40(6), 340–355, 1997