28

A Knowledge Based Approach to Parallel Software Engineering

P. Milligan, P. P. Sage, P. J. P. McMullan and P. H. Corr
The Queen's University of Belfast
Department of Computer Science
Belfast BT7 1NN
N. Ireland
Tel: +44 1232 245133 Extn. 4645
Fax: +44 1232 331232
E-mail: p.milligan@qub.ac.uk

Abstract

Advances in technology have resulted in the development of many different multiprocessor systems. Unfortunately these have not been accompanied by advances in portable, user-friendly program development environments. This paper overviews the prototype of a development and migration environment for parallel software engineering which incorporates the application of knowledge based techniques to the core topics of loop restructuring, code generation and code evaluation.

Keywords

CASE environments and support tools; application of AI (expert system) techniques; autoparallelisation.

1 INTRODUCTION

In the past few years there has been a dramatic increase in the number of different multiprocessor systems in the marketplace. This development has provided the user with an increased potential processing power and an even wider range of parallel machine architectures. However, this increase in power and choice has not been accompanied by an increase in flexible, portable, user-friendly program development environments. If the

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

potential users of multiprocessor systems are to become actual users such environments must be provided.

It is fair to say that the majority of scientific and engineering users do not have the time or the desire to understand the intricacies of data dependence analysis, parallel program design and load balancing techniques for multiprocessor machines. For this potential user base a viable development environment must provide not only facilities for the development of new code but also an integrated toolset to ease the migration of their existing, mainly sequential, codes. The Fortport prototype (Milligan et al, 1992, Quill et al, 1995) described here is an attempt to provide such an integrated parallel software development and migration environment for Fortran programmers. Central to the ethos of the Fortport prototype is the belief that the user should have control over the extent of their involvement in the parallelisation process. It should be possible for novice users to be freed from the responsibility of detecting parallelisable sections of code and distributing the code over the available processors. Alternatively, experienced users should have the facility to interact with all phases of the parallelisation and distribution of the code. To enable a novice user to devolve responsibility entirely to the system implies that the system has sufficient expert knowledge to accomplish the task. This paper discusses how such expert knowledge has been provided within the Fortport prototype with particular emphasis on the migration of existing sequential code onto a multiprocessor architecture.

2 KNOWLEDGE ACQUISITION AND APPLICATION

One of the dominant problems associated with program development systems for parallel architectures has been the inability to completely automate the system. Several development environments exist, e.g. SUPERB (Zima et al, 1988) and PDE (Decker et al, 1993), but it is inevitable that they require some form of user interaction to assist with the process of code parallelisation. This interaction can take two forms, either the user annotates the program to indicate to a compiler that certain actions are required or the user interacts with the system during execution to choose transformations or data partitioning schemes.

The effect of the user interaction is to assist the development environment by providing user expertise. Hence it should be possible to develop expert systems to at best replace, or at worst supplement, this user interaction.

To investigate the viability of this approach two expert systems were proposed and developed for use within the FortPort prototype. One system would assist with the process of transformation selection and one with code generation and evaluation.

A common approach to the development of both expert systems was adopted. The approach was to hand code the required solutions and then analyse the decision making processes followed in the development of the code, i.e. a reverse engineering model was used. This model enabled the key steps in the two processes (parallelisation and generation/distribution) to be identified. In addition the key facts that trigger the various decision making steps could be identified. These key facts or characteristics again fall into two groups, namely loop characteristics and performance characteristics.

Subsequently the rules used in the expert systems were derived by combining the key steps identified in the reverse engineering phase with the relevant characteristics. In general the rules have the form:

```
define rule 'name'
   condition list
        =>   action.                                          (1)
```

The condition list contains one or more expressions based on the characteristics which must be satisfied for the designated action to take place. When all of the conditions in a list are met the rule is fired. Examples of complete rules are given in a later section of the paper.

Future extensions to the knowledge acquisition phase will add characteristics derived from the application domain, prior and historical knowledge, i.e. decisions made in the past that may be applicable again. In other words a long term goal of the system will be to provide a learning environment.

3 THE PROTOTYPE SYSTEM

3.1 Input Handling and Graph Construction

The input handler generates a representation of the user program in the form of a graph. The graph is formed from a hierarchy of nodes. A detailed description of the graph nodes has been prepared by (Sage et al, 1993).

One of the key reasons for choosing a graph based approach is the ease with which it may be modified and extended. In, for example, the parallelisation of a program targeted at a system employing a message passing model for inter-process communication, the communication primitives will have to be included explicitly in the program. This can be achieved by inserting additional nodes in the graph, known as ghost nodes.

3.2 Graph Transformation

The graph of a loop is traversed by a number of analysers which build up a picture of the loop. This picture forms part of the input to the parallelisation expert system. Basically, as loops represent a rich source of potential parallelism, the goal is to identify the best loop distribution possible. This requires the system to undertake traditional dependence analysis and reduction, followed by loop distribution. A variety of traditional and novel techniques are provided, e.g. statement reordering, loop interchange, loop skewing, variable copying and scalar and array expansion are provided as core or kernel activities.

To illustrate the principle of loop picturing consider the following trivial example:

```
        DO I = 1, N
            DO J = 1, N
s1:             A(I,J) = B(I,J)
s2:             C(I, J) = A(I+1, J)
            ENDDO
        ENDDO
```

Some of the facts of this loop are represented as follows:

```
(loop 1 I N)                /* outer loop, subscript I, upper bound N */
(loop 2 J N)                /* inner loop, subscript J, upper bound N */
(concurrent 2)              /* only the second loop (J-loop) is parallel */
(anti 2 1)                  /* anti dependence (due to A) from s2 to s1 */
(actual A by Row)           /* user specified                           */
(actual (B byRow)
(actual (C byRow)
(overall byRow 1)           /* characterisation analysis has identified the*/
(overall byRow 2)           /* overall data partition for each statement*/
```

3.3 Application of Knowledge Based Techniques

The rule-based approach used in this system receives as input a list of facts generated by the loop analysis. A set of rules have been built up as the result of studying code parallelisation. There are multiple goals, one for each transformation, and the system forward chains through the rule base until all goals are met or the system fails. However, this process can result in a number of valid transformations being selected.

The specific examples given below deal with loop interchange strategies:

```
(defrule loop-interchange-check-4
  (not (outer-loop-parallel)
  (concurrent $?front ?num $?rear)
  (test (<> ?num 1))
        =>
              (assert (apply Loop-Interchange 1 ?num))) and

(defrule loop-interchange-check-7
  (declare (salience 100)
  (not (clashing-data-distributions)
  (overall byRow 1)
  (loop 1 I ?)
  (loop 2 J ?)        =>        (assert (apply Loop-Interchange NOT REQUIRED)))
```

Both of these rules are concerned with determining whether or not a loop interchange strategy should be applied. Using the fact list derived from the trivial example in the preceding section both rules will fire giving rise to an apparent conflict. This is resolved by considering the weighting factor (salience) associated with each of the rules. For the rules described above ensuring that the partitioned iterations on each processor access local data is more important that ensuring parallel loops. This is denoted by giving the second rule a weighting of 100.

A set of rules has been developed for use with the core transformations that are implemented in the current version of the prototype. The results output from the transformation phase are passed to the generation and evaluation phase. Here a series of codes (based on the alternatives identified by the transformation selection phase) can be generated and evaluated.

3.4 Code Generation and Evaluation

The code generator accepts as input the modified graph produced by the parallelisation phase and generates lists of information representing the characteristics of this 'parallel' graph. Once again, information on the remaining data dependence, loop boundaries, variable accesses, hotspot analysis is gathered.

This information, together with some basic characteristics of the target architecture, is fed to the generator expert system (GES) which returns recommendations on initial code and data distribution.

Within the prototype the parallel architecture is regarded as a master/slave topology with an SPMD model. Inter-process communication is handled by PVM like communications strategy. Future changes will introduce the use of the MPI scheme. For demonstration purposes the current version of the prototype generates CSTools Fortran for execution on a Meiko M40.

As a program is executed profiling information is gathered. The initial distribution assumes that all slave processors will have the same execution profile. Clearly this will only be true for very simple programs and the information from the first execution will indicate which processors carry the major compute-intensive elements of the program.

Once identified, the compute-intensive element(s) can be subjected to closer scrutiny to attempt to identify the precise sections of code that are proving to be time consuming. For example problems can arise due to communication overload or external library calls. The profiler will find the subroutine(s) causing the delays and indicate the nature of the problem. This information can be used by the programmer to enable the code to be distributed in a different manner. Alternatively the information can be fed to the GES, i.e. a feedback loop is available.

The GES can handle the feedback information obtained by the profiler in several ways. Initially an attempt is made to eliminate the problem(s) in a task by recommending a different loop distribution. This will require the generator to produce an alternative partitioning of the arrays associated with the loop involved and hopefully will reduce communication times.

If this approach fails then an alternative may be to produce a different distribution of the program code across the slave processors. If this approach is adopted then the execution profiling and feedback runs again to analyse the new situation and report accordingly.

However it may be the case that having tried different loop distributions and different code partitioning strategies that no real gain in performance can be detected. If this situation arises then the GES will report this fact to the parallelisation expert system. If this step is taken then the parallelisation phase is reactivated with the goal of identifying an alternative set of transformations that will be applied to the original graph-based representation of the source program. In other words the complete development/migration cycle begins again.

4 CONCLUSION

The FortPort prototype, currently under beta-test, offers graph construction, knowledge driven loop restructuring and knowledge driven code generation. Transformations to remove or reduce data dependence are selected dynamically based on loop characteristics. Code and data is distributed across a multiprocessor architecture again on the basis of the analysis of loop and architecture characteristics.

The existing model for the creation of rules, i.e. analysis of hand coding, will be supplemented by a new approach. At the moment the transformations are expressed directly in program code. However a transformation is in effect a graph reordering function, i.e. the effect of applying a transformation to a graph-based representation of a program is simply to produce another graph. A kernel set of graph manipulations, so-called atomic operations, have been isolated. All existing transformations in the system can be expressed in terms of suitable combinations of these atomic operations. Future work will explore the effect of different combinations of the atomic operations on a program graph with the goal of identifying new transformations for parallelisation.

The major strength of the FortPort system is that it provides a complete development and migration environment. Novice users can receive expert help with the complex task of parallelising a program. Equally, experienced users can benefit from the recommendations produced by the expert systems. While the prototype will accept F77 the final version of the system will accept both F90 and HPF and will assist a user with the task of selecting appropriate parallelising statements.

5 REFERENCES

Decker, K.M., Dorvac, J.J. and Rehmann, R.M. (1993) A Knowledge-Based Scientific Parallel Programming Environment, Technical Report CSCS-TR-93-07.

Milligan, P., McConnell, R.K., Rea, S.A., Benson G. and Sage, P.P. (1992) Apparently Sequential Programming Environments for Parallel Computing, *Parallel Computing and Transputer Applications, IOS Press CIMNE*, Barcelona, 297-306.

Quill, J.C., McConnell, R.C. and Milligan, P. (1995) A Prototype Environment for Parallelization, *Lecture Notes in Computer Science*, **919**, 936-936.

Sage, P.P., Milligan, P., McConnell, R.K., Rea, S.A. and McCarney,M.T. (1993) Graph Management within the FortPort Migration Environment, *Microprocessing and Microprogramming*, **33**, 137-140.

Zima, H.P., Bast, H.J. and Gerndt, H.M. (1988) SUPERB - a tool for semi-automatic MIMD/ SIMD parallelisation, *Parallel Computing*, **6**, 1-18.

6 BIOGRAPHY

Peter Milligan is a senior lecturer in the Department of Computer Science. Currently his research programmes are devoted to the design and implementation of intelligent, semi-automated programming environments for the generation of parallel programs. In addition Dr Milligan works on the migration of mathematical codes to parallel architectures. Dr Milligan has supervised over 30 MSc and PhD students and has been involved in the organisation of six international conferences and three international workshops devoted to parallel and distributed computing. Patrick Corr is a lecturer in the Department of Computer Science. His research interests centre on the application of artificial intelligence techniques, particularly neural networks, to a range of problems in science and engineering. Paul Sage and Paul McMullan are postgraduate students currently completing PhD theses on aspects of parallel software development.