Utilising Knowledge Systems to Enable Enhanced Data Distribution in a Parallelisation Environment

B. MCCOLLUM, P. MILLIGAN AND P. H. CORR
School of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN
N. Ireland
Tel: +44 1232 274626
Fax: +44 1232 683890

Abstract:-In spite of considerable research it is widely accepted that the goal of fully automating the process of migrating legacy codes to parallel architectures must remain an unreachable target. Where there have been successes these have been restricted to discrete domains with clearly defined boundaries, e.g. the PDE system [1]. The most favourable general approaches are those which use AI techniques, usually expert systems. This paper introduces a new knowledge model that contains a significantly greater proportion of information than any other approach to date. It is believed that using this novel approach enhances the production of parallelised code. The system is illustrated using a code selected from the LINPACK library, TRED2.

Keywords:- Knowledge Models, Data Distribution, Software Re-Engineering, Semi-Automatic Parallelisation, Legacy Systems IMACS/IEEE CSCC'99 Proceedings, Pages:3301-3307

1 Re-engineering Sequential Codes for Multiprocessor Platforms

It is perhaps pejorative, but nonetheless accurate, to characterise the majority of typical users of multiprocessor systems as unskilled in the arts of parallelisation. Typically, they will be confirmed in the use of essentially sequential languages such as FORTRAN and have neither the time nor the desire to understand the intricacies of parallel development techniques or the peculiarities of target architectures in their endless search for enhanced performance. For such a user there is a compelling need for the development of environments which minimise user involvement with such complications. In short, if novice users are to realise the potential of multiprocessor systems then much of the expert knowledge required to develop parallel code on multiprocessor architectures must be provided within the development environment.

The ideal solution is a fully automatic approach in which a user can simply input a sequential program to the system, be it migrated or newly developed, and receive as output an efficient parallel equivalent. Automatic parallelisation scores high on expression, as programmers are able to use conventional languages, but is problematic in that it requires inherently complex issues such as data dependence analysis, parallel program design, data distribution and load balancing issues to be addressed. The approach also scores low on portability as solutions may require considerable modification when run on different architectures.

Existing parallelisation systems adopt a range of techniques in an effort to minimise or eliminate the complexity inherent in the fully automated approach. Almost invariably the burden of providing the necessary guidance and expertise lacking in the system falls back on the user. Indeed, existing systems may be classified by the extent to which user interaction is required in the process of code parallelisation. At one end of the spectrum is the purely language based approach in which the user is entirely responsible for determining how parallelism is to be achieved by annotating the code with appropriate compiler directives. The other end of the spectrum represents the goal of a fully automatic parallelisation environment, independent of application domain and requiring no user guidance. Between these extremes lie a number of environments which permit the user to interact with the

system during execution, with varying degrees of guidance, to choose appropriate program transformations or data partitioning schemes.

This paper focuses on one of the core problems in constructing a parallelisation system, that of dealing with data distribution. For a parallelisation tool to be effective at parallelising real user applications it must provide automatic data partitioning algorithms as an essential part of the environment[1]. Hence, the criteria against which systems should be judged is the extent to which they provide domain independent code handling, and that they require no user interaction for the generation of effective and efficient data distributions. In the following section we review a number of existing systems and evaluate them against these criteria.

2. Related Work

In all areas except the automatic approach, parallelisation is guided by data parallelism and based upon a user-provided specification of data distribution. High level programming languages extensions to FORTRAN, such as Vienna FORTRAN [2, 3] and HPF [4]require the user to specify the distribution of program data as directives within the source code. Although it may sometimes be relatively straightforward for a programmer to select an appropriate set of mappings, in many situations the process is difficult and requires a tedious analysis of the code and its data references. While this language based approach is independent of domain it is entirely reliant on user interaction and thus fails the criteria as stated in the previous section.

FORGE90 [1], a windows based environment, implements data partitioning of a program by inserting directives into a parallelisation database that it maintains. The user sets up a decomposition in a window and applies this to a set of arrays selected in a second window. The user must also specify which loops in the code are to be distributed. In a similar manner CAPTools [5] contains an algorithm for partitioning and distributing application code data in an organised manner across a target processor topology. However, the initial partition must be defined by a user who must choose the relevant array and associated array index. It is clear that both environments require user interaction at early and/or intermediate stages of the parallelisation process thereby failing the second criteria.

The final category to be considered encompasses those tools which deal with domain specific applications, e.g., CSCS [6]. Typically such tools provide efficient and realistic solutions. However, as they deal with only limited domains it is clear that they fail the criteria as they are unsuitable for general codes.

It is clear that the elimination of user interaction either in the form of program annotators or as reactive selectors to semi-automated tools is the essential goal. In an effort to remove user interaction more recent approaches, e.g., Benson [7] and Zima [5], have focused on the use of expert system models to facilitate knowledge driven parallelisation. However, this approach, while demonstrating some success, has not entirely eliminated the necessity for user interaction. If expert knowledge of the parallelisation process is to be made available within a parallelisation environment it is crucial that the structure of and relationships between the various sources of that expert knowledge is fully understood. Without such an understanding effective exploitation of the available knowledge is inevitably limited.

The KATT project provides a solution to this problem by offering users of parallel systems a set of tools incorporating AI technology to produce efficient parallel code. Central to this approach is the development of a novel knowledge model which provides a consistent framework for the capture, analysis and utilisation of the full range of relevant information. To date we have produced a range of tools, FortPort [8, 9, 10], which parallelise codes using a number of expert systems to assist with code transformation, data distribution, code generation and execution analysis and feedback. However, there are limitations to the type of information that can be modelled and applied using expert systems. It is our contention that the use of neural networks will complement expert systems in this respect and provide access to a much broader range of information [11,12]. The following sections of the paper review relevant knowledge sources, introduce the novel knowledge model and demonstrate the viability of this approach by applying our techniques to a real code, TRED2, previously used by O'Boyle [14].

3. Knowledge Sources

There are a number of knowledge sources which must be utilised to assist with the parallelisation process.

These sources are; the expertise that exists among users; the hierarchical nature of that expertise; the variety of architectural paradigms; the variety of problem domains; the structure of the program code itself and performance profiles gathered during code execution.

There are obvious relationships between the areas mentioned above, e.g., performance profiles are intimately related to the particular architecture used while users may have varying degrees of expertise in some or all of the other areas. Any abstract model of the available knowledge must be driven with specific information and associated inter-source relationships extracted from the following sources, namely, architecture specific, code specific, problem specific and programmer specific.

3.1. Architecture Specific Knowledge:

Knowledge is available about the architecture of the target machine. This knowledge will include, number of processors, details of the memory architecture and the time taken for communication of data between different processors (latency, bandwidth, and the cost of routing between non-adjacent processors). Knowledge is also available on the actual configuration on which the program will run, e.g., the type of communication protocol provided.

The architecture currently being used consists a cluster of five Pentium processors with a UNIX operating system connected via a 10Mb bandwidth LAN switch in a star topology. Communication templates have been developed using C with embedded MPI statements to provide the necessary inter-processor communication.

3.2. Code Specific Knowledge:

This can be divided into source specific knowledge and execution specific knowledge:

Source Specific Knowledge: Considerable knowledge is available from the input sequential program, e.g., the number and extent of the computationally intensive parts of the program can be identified, the size of data sets operated upon can be determined and the spread of the computational load, loop bounds, access patterns to arrays, presence and type of data dependencies, etc. can all be determined from an analysis of the source code. A selection of this knowledge is currently held as facts within the expert systems in KATT for the purpose of program transformation, to eliminate data dependencies and suggest data distributions. An example of the knowledge available and how it is utilised is presented in the case study in section 6.

Execution Specific Knowledge: When a sequential code or its parallel equivalent is executed on a target architecture it is possible to gather a profile of the code's performance and produce an execution analysis. Such an analysis, indicating, for example, processor utilisation, communication timings, etc., represents knowledge which can have a role in determining whether or not a chosen parallelisation strategy has been successful and can be used to inform subsequent parallelisation recommendations.

3.3. Problem Specific Knowledge:

Work is ongoing in analysing a range of numerical problems and techniques in order to determine how they may best be implemented on currently available architectures. Much of this work involves hand crafting a solution for optimal performance. Whereas this work is important, the main objective of the KATT project is to provide automated tools to assist in the optimisation process. If expert knowledge related to these areas can be gathered and stored then it is available to offer guidance during execution of the system.

If a particular part of a program or calculation can be recognised then this knowledge can be used to indicate the most efficient parallel implementation on the designated architecture. For example, the purpose of the TRED2 program is to reduce a symmetric matrix to a symmetric tridiagonal matrix using and accumulating orthogonal similarity transformations. Once this program has been parallelised capturing knowledge on how the transformation was achieved may be brought to bear on further codes with the same or similar aims suggesting that they too should be parallelised in a similar fashion. This knowledge may be gained either explicitly or gathered implicitly during the knowledge extraction process.

3.4. Programmer Specific Knowledge:

This can be divided into sequential code knowledge and parallelisation knowledge:-

Sequential Code Knowledge: It is not unreasonable to assume that a newly developed code represents a clear

expression of the algorithm used to solve a particular problem. In particular, an experienced programmer will have a range of experiential programming knowledge which will be reflected in the code under investigation. However, as a consequence of the inevitable maintenance activity associated with legacy codes the structure of the code is degraded making it difficult to extract the knowledge necessary to suggest the original intention. TRED2 is a very structured code will no evidence of loops or sub routines having been restructured. As opposed to many examples of legacy codes studied, this particular piece of code provides an ideal starting point for the parallelisation process.

Parallelisation Knowledge*:* This relates to how to transform sequential code for execution on a particular parallel machine. It specifies what parts of the program are to be parallelised, how processes interact, in what order they are to be carried out and how to resolve conflicts if they arise. As before, the programmer will have a range of experiential programming knowledge varying from beginner to expert. TRED2 may be parallelised by firstly applying a series of transformations to enhance the explicit parallelism within the program. This parallelised version subsequently undergoes data partitioning [13]. The particular techniques used and the order they are applied are one example of how to go about parallelising the program and are dependent on who is working on the code and their experience with parallelisation.

4. Structure and Relationships between Knowledge Sources

The available knowledge can be regarded as a plethora of partially overlapping and partially conflicting information. To reduce this confusion it is necessary to organise the information in a logical manner.

At first sight it appears that the information can be arranged as a spectrum running from the ‘low level’ architecture specific information to the more ‘high level’ problem specific information. However, subsequent attempts to determine precise boundaries between information categories within the spectrum fail. If the knowledge available is grouped in a slightly different manner, the inherent overlap between the categories, which is an essential facet of the real world and has to be retained, can be recognised and presented in a logical manner as a ‘knowledge pool’. This knowledge pool is shown conceptually in Figure 1 together with an indication of the extent to which it is exploited by existing systems.

The ‘best’ information for parallelisation will occur where all four knowledge components intersect, the so called *knowledge enriched zone*. Hence, the ultimate goal of the KATT project is to take up a position within this zone. To date, as shown in the diagram, none of the existing systems lie within this zone.

Within the current KATT environment the exploitation of the full range of available knowledge is limited as a result of using only expert system techniques. Expert systems do not provide any mechanism for pattern matching, an essential component of any knowledge driven data partitioning algorithm [14]. Neural networks provide this necessary pattern matching ability.

The ability of neural networks to generalise and extract patterns from a corpus of data, allied with their capacity to learn an appropriate mapping between an input and output, has ensured that they have found application in many diverse disciplines. Neural networks have a unique set of characteristics. They can, learn from experience, generalise from examples, and abstract essential information from noisy data. These features also make their use attractive in this context. Given an input of essential information about a code and an output of the most appropriate parallelisation operations to apply to that code, the learned mapping held by the network effectively encapsulates knowledge which may be brought to bear on the process. To use neural techniques successfully, initially in extracting and then utilising knowledge from code, requires firstly, a characterisation scheme capable of representing the information about a code necessary for a decision on the most appropriate parallelisation operations to apply and secondly, access to a large corpus of codes, and preferably the associated operations, from which to learn. The eventual performance of the network in suggesting the most appropriate parallelisation operations is heavily dependent on the quality and accuracy of the original training examples and the characterisation scheme used.

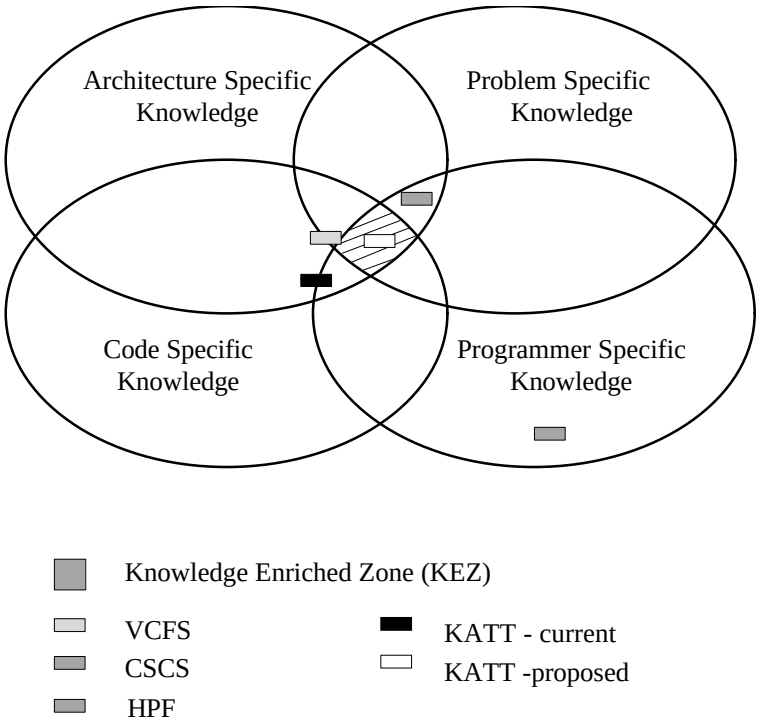

Figure 1. Overlapping knowledge sources and their exploitation by parallelisation systems.

5. Advantage over the Current System

Within this project the strength of the expert system lies in its ability to scrutinise the code at a higher structural level and perform certain rule based actions. The characteristics of the input code cause certain rules within the expert system to be fired. These rules reflect the structure of the users' code at a high level. It is this ability to scrutinise code at a high level which is the strength of the expert system.

The main advantage of the neural network is its ability to generalise from code, recognising and extracting detail. This information is required to facilitate pattern matching, a fundamental activity in support of code distribution strategies. The neural network provides faster pattern matching and greater detail than the expert system. Combining these approaches within a coherent framework improves the ability of KATT to offer strategic intelligent guidance to the user through broader and deeper access to the knowledge pool. Such a system maximises knowledge utilisation and effectively positions KATT within the knowledge enriched zone.

6. Illustrative Case Study:- Knowledge Extraction for Data Partitioning

To illustrate the viability of this approach and demonstrate the resultant knowledge utilisation a case study is introduced. The case study is focused on a typical, real, non trivial problem taken from the EISPACK library and was used by O'Boyle [14] to demonstrate a data partitioning algorithm.

Sequential analysis of the code identifies five major computationally intensive areas as indicated on the listing. As parallelisation of these areas has the greatest potential to maximise performance attention will focus, in the first instance, on these regions. For the purpose of illustrating the process of knowledge extraction from code and its subsequent use in determining the most appropriate data partitioning, the code section containing computationally intensive areas 1 and 2 will be examined in detail. In partitioning the data it is assumed that minimising communication is more important than load balance; this is based on the supposition that non-local accesses cost more than computation [13]. This is simplistic but will suffice for purposes of illustration.

Consider the following code fragment containing the computationally intensive loops labelled 1 and 2 respectively:

Tread_(a)

```
      DO 45 J = 1, L
         F = D(J)
         Z(J,1) = F
         G = E(J) + Z(J, J)* F                    - - (1)
         JP1 = J + 1
         IF (L .LT. JP1) GO TO 44
         DO 43 K = JP1 , L
               G = G + Z(K, J)* D(K)
               E(K) = E(K)  + Z(K, J) * F         - - (2)
43       CONTINUE
44       E(J) = G
45    CONTINUE
```

The following knowledge relevant to data partitioning can be extracted from the code by inspection. Within KATT the information is elicited automatically.

1)	Code section identification	Tread_(a)	
2)	Number of loops in code section	2	
3)	Number of statements before 1st loop	0	5
4)	Loop Identification	L_1	L_2
	a) Number of statements in loop	9	2
	b) Type of loop transformation	none	none
	c) Loop variable component	row	row
	d) Computational access pattern	colm	row
5)	Number of dependencies	1	1
	a) Data dependence type	flow	Input
	b) Within single statement	no	no
	c) Loop independent	yes	yes
	d) Loop carried forward (i.e. Direction vector)	no	no
	e) Component dependency in		2nd
	f) Dependence Distance	-	-
	g) Which loop carries dependence	2	-

The knowledge represented above is derived directly from the code and as such represents *source specific knowledge* as detailed in section 3.2. In this instance, source specific knowledge alone is sufficient to determine an appropriate data partitioning.

1) Code section Tread_(a)

2) Loop Identification L_1 L_2

3) Block partitioning yes yes
   Both loops can be partitioned by block with the insertion of communication statements at the computational boundaries

4) Row partitioning no yes
   Loop_1 increases the J component of the array Z(J, 1) but access is also required of Z(J, J) during the same iteration of the loop.

5) Column partitioning yes no
   Loop_2 increases the K component of the array Z(K, J) i.e. the array access pattern is by row.

In the current KATT environment an expert system is used to synthesise the source specific knowledge and produce a recommended data partition. This section of KATT is being revised to include a neural network component thereby providing more comprehensive coverage of this knowledge source [9, 12].

Using the knowledge of the dependencies within the loop structures of Tread_(a) block or column data partitioning will both minimise communication, thereby maximising program speed-up during the parallelisation process, and ensure that the integrity of the outcome of the identified sections of code is maintained. Given the suggested partitioning, architecture specific knowledge (best represented explicitly within an expert system but currently held implicitly, hard coded for a particular target, within the current KATT system) is required before the system can determine an actual distribution on the target platform.

Having analysed the code and drawn on the appropriate knowledge sources to determine a suitable data distribution KATT is now in a position to execute the equivalent parallel code on the target platform. While executing, performance profiling information is gathered and used to determine if an acceptable improvement in performance over sequential execution has been achieved. If performance proves unacceptable this profiling information is fed back into the parallelisation environment and used, again with the assistance of an expert system, to inform a revised data partition and/or distribution recommendation. Iterative feedback continues in this way until either an acceptable performance improvement is achieved or the system determines that an acceptable improvement is unattainable.

7. Conclusion

In this paper a novel knowledge model visualised as a knowledge pool is presented. The model provides a basis for improving the parallelisation process by enabling a greater volume of information relevant to the re-engineering process to be extracted from a legacy code and utilised. Exploiting this knowledge requires a combination of expert system and neural network techniques, each bringing their own strengths to bear and combining to take full advantage of the knowledge available. By exploiting the strengths of both expert system and neural network approaches within a common framework the overall quality and scope of the parallelisation process are greatly enhanced. A case study, based on a real code, is used to illustrate the techniques used to facilitate data distribution.

References:

1. R.Rehmann, Automatic Generation of Programs for a Scientific Parallel Programming Environment, Technical Report CSCS-TR-94-02, Swiss Scientific Computing Centre, May 1994.

2 P. F. Leggett, A. T. J. Marsh, S. P. Johnston and M. Cross, Integrating User Knowledge with Information from Parallelisation Tools to Facilitate the Automatic Generation of Efficient Parallel Fortran Code., Parallel Computing, vol. 22, pp259 - 288, 1996.

3. J. Hulman, S. Andel, B. Chapman and H. P. Zima, Intelligent Parallelization within the Vienna Fortran Compilation system, Proc. Forth Workshop on Compilers for Parallel Computers, 1993, pp 455-467

4. B. Chapman, T. Fahringer and H. Zima, Automatic support for data distribution on distributed memory multiprocessor systems, in U. Banerjee et al. Eds. Proceedings of the 6th Workshop in Language and Compilers for Parallel Computing. Lecture Notes in Computer Science, vol 768. New York: Springer-Verlag pp184-199, 1993

5. High Performance Fortran Forum, High Performance Fortran Language Specification, Version 1.0, Rice University, May 1993.

6. S Johnston, et. al., The Design and Evaluation of "CAPTools" - A Computer Aided Parallelisation Tool-kit, Paper No. 98/IM/39, CMS Press, 1998

7. K. Decker, J Dvorak and R Rehmann, A knowledge-based scientific parallel programming environment, Swiss Scientific Computing Centre, CSCS-TR-93-07, 1993.

8. T.J.D.Benson, The Mathematician's Devil, PhD thesis, QUB 1993

9. P.Milligan, P. P. Sage, P. J. P. McMullan and P. H. Corr. A Knowledge Based Approach to Parallel Software Engineering. In, Software Engineering for Parallel and Distributed Systems, Chapman and Hall, ISBN 0-412-75640-0, pp 297 - 302, 1996.

10. P. J. P. McMullan, P. Milligan, P. P. Sage and P. H. Corr. A Knowledge Based Approach to the Parallelisation, Generation and Evaluation of Code for Execution on Parallel Architectures. IEEE Computer Society Press, ISBN 0-8186-7703-1, pp 58 - 63, 1997

11. P. J. P. McMullan, P. Milligan and P. H. Corr. Knowledge Assisted Code Generation and Analysis. Lecture Notes in Computer Science 1225, Springer Verlag, ISBN 3-540-62898-3, pp 1030-1031, 1997

12. V. Purnell, P. H. Corr and P.Milligan, "Neural Networks for Code Transformation", Lecture Notes in Computer Science, 1225, Springer Verlag, pp 1028-1029

13. V. Purnell, P. H. Corr and P.Milligan, " A Novel Approach to Loop Characterisation", IEEE Computer Society Press, pp 272-277, 1997

14. M. O'Boyle, A Data Partitioning Algorithm for Distributed Memory Compilation, Department of Computer Science, University of Manchester, Technical Report UMCS-93-7-1

15. P. J. P. McMullan, The Intelligent Generation and Analysis of Code for Parallel Platforms, PhD thesis, QUB 1996.