A Meta-Heuristic Approach to Parallel Code Generation

B. MCCOLLUM, P.H.CORR AND P. MILLIGAN
School of Computer Science,
The Queen's University of Belfast
Belfast BT7 1NN
N. IRELAND
(b.mccollum, p.corr)@qub.ac.uk

Abstract The efficient generation of parallel code for multi processor environments, is a large and complicated issue. Attempts to address this problem have always resulted in significant input from users. Because of constraints on user knowledge and time, the automation of the process is a promising and practically important research area. In recent years heuristic approaches have been used to capture available knowledge and make it available for the parallelisation process. Here, the introduction of the novel approach of application of neural networks techniques is combined with an expert system technique to enhance the availability of knowledge to aid in the automatic generation of parallel code

Key Words Meta-heuristics, Neural Networks, Automatic Parallelisation, Data Distribution

1 Introduction

One of the principal limiting factor to the development of a fully automatic parallelisation environment is the automatic distribution of data to the available processors. Various aspects of this problem have proven to be intractable [1]. Recently heuristic techniques have re-invigorated research in this area. Systems such as expert systems, simulated annealing and genetic algorithms are currently being applied to the problem of data partitioning [2][3]. These approaches could be deemed to be incomplete for the following reasons:

i) they make generalising assumptions or tend to be biased towards particular problems [3].
ii) most rely on intensive analysis of the possible distribution solutions which can be time consuming for anything but quite small programs. A lot of time can be spent tuning the parallel code to become as efficient as possible [3].
iii) they are machine specific and are not portable across a number of architectures [4].
iv) they rely on limited pattern matching thus not exploiting to the full the fact that similar structures exist within different sets of sequential code. Although some may use expert systems or knowledge based systems to advise on distribution strategies, this type of knowledge is not easily represented in such systems [5].
v) In many cases, the search space is pruned to make it sufficiently small to make the search for the solution fast [5].

However, while many of these heuristic approaches have been useful but limited, it is our contention that a combination of such approaches founded in a consistent framework can offer a significant improvement.

Neural networks offer a method of extracting knowledge implicit in the code itself. This provides an alternative low-level, signal-based, view of the parallelisation problem in contrast to the high-level, symbolic view offered by an expert system. Combining both paradigms within a coherent knowledge hierarchy should improve the strategic intelligent guidance offered to users through access to a broader and deeper knowledge model. This knowledge model has been reported elsewhere [6].

This paper reports on this meta-heuristic approach for the purpose of data partitioning within a suite of existing knowledge based tools for automatic parallelisation, the KATT environment [2].

The architecture being used consists a cluster of five Pentium processor workstations (COW) 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. A SPMD programming model is used.

2 Existing Environments

In all areas, except an automatic approach, parallelisation is guided by data parallelism and based upon a user-provided specification of data distribution. Alternatively, the automatic approach invariably casts the problem as one of optimising processor usage while minimising communication. As an optimisation problem a range of techniques such as genetic algorithms,

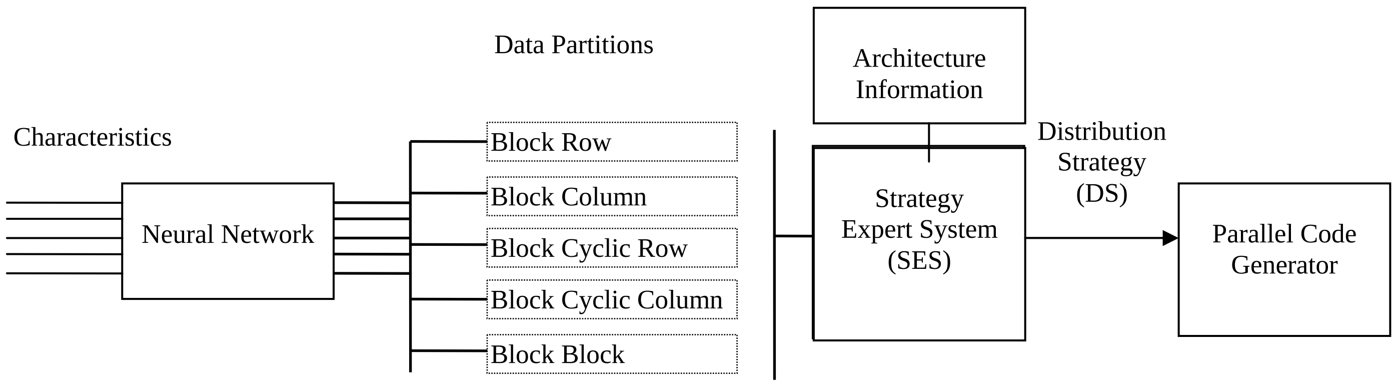


Figure 1 - Overview of system

simulated annealing and formal mathematical notation have been brought to bare [4,5,7,8]. As yet, few of these approaches have been used within an overall parallelisation environment.

Research reported here is grounded within the framework of the on-going Knowledge Assisted Transformation Tools (KATT) project (2). KATT provides an integrated software development and migration environment for sequential programmers. Central to the ethos of KATT is the belief that the user should have control over the extent of their involvement in the parallelisation process. For example, 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 code. To enable a novice user to devolve responsibility entirely to the system implies that the system must have sufficient expert knowledge to accomplish the task. With the current KATT environment an expert system chooses either a column, row or block distribution depending on the assignment statements within the section of code under scrutiny. Cyclic and Block-Cyclic partitioning are not presently covered due to the complication of load balancing issues. Captured knowledge is currently held in two forms; explicitly as the facts and rules used by the existing expert system; and implicitly as hard-coded information about the target architecture for example, or inferred from within the code itself. The current expert system is limited due to its poor pattern matching capabilities and inability to generalise from the specific rules within its knowledge base.

The approach taken here enhances the existing expert system approach by the introduction of a second heuristic technique - neural networks.

3 Meta-Heuristic Approach

An overview of the hybrid system is shown in figure 1. Here the neural network is responsible for suggesting an appropriate data partition given a set of characteristics derived from the code under consideration. The strategy expert system takes the suggested partition information together with the architectural information to determine a suitable data distribution strategy. This in turn is used to generate the actual parallel code for execution on the target architecture.

Distributions on the chosen architecture have been coded using MPI. These are, column block, column block cyclic, row block, row cyclic block and block block. Using Fortran with embedded MPI statements, code fragments have been distributed on the cluster of workstations (COWs) ensuring that the results before and after distribution are the same. The intention is to ascertain, through the experimentation process, how individual and particular combinations of characteristics are related to the choice of a particular data partition

4 Data Partitioning Strategy

As data dependencies are a serious impediment to parallelism [9], it is essential to remove and preferable eliminate them during the parallelism process. Data dependency reduction can be achieved either by program transformations [9] or internalisation [10]. program transformations allow the exploitation of low level parallelism and increase memory locality. These are cumbersome to do by hand but may have an important influence on overall performance. This method for data dependency reduction is currently used within the overall KATT project [2].
Another effective way to reduce data dependencies is to internalise the data dependencies within each code partition so that all values required by computation local to a processor are available in its local domain. If this is not possible, the compiler must insert the

appropriate communication statements to access non-local data. Due to the need to distribute data operated on by loops which have not necessarily gone through one of the established transformations [9], the internalisation method is used and is the overriding factor when choosing how code should be partitioned and distributed.

5 The Neural Network Approach

Neural networks have achieved notable success in other areas where heuristic solutions are sought.[13]. In this work we intend to take advantage of their ability to generalise and extract patterns from a corpus of codes, one of the failings identified in alternative techniques. Two distinct approaches are currently being using:

- *an iterative data partitioning selection technique* which uses a multilayer perceptron model to recommend a particular partitioning, selected from a restricted set, to apply to an input loop structure. Training the neural network requires a representative selection of loops, each of which must be characterised and analysed to determine the appropriate transformation. This process has been carried out matching loop characteristics to the data partitioning which gives maximum speed up in loop execution.
- *a clustering technique* which uses a previously constructed feature map to determine an appropriate data partitioning for the code. The clusters formed are investigated to determine which data partitioning is applicable. The feature map may then be labelled such that a partitioning may be associated with an input.

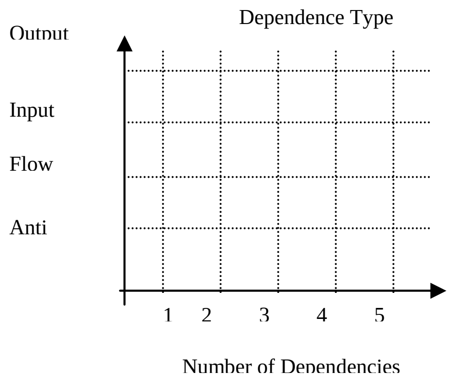
The key issues are identifying sources of knowledge, deciding on the level of complexity to be dealt with, establishing a suitable characterisation scheme and acquiring training examples from a significant corpus of codes.

6 Sources of Knowledge

A set of characteristics has been gleaned from a corpus of codes and stored in a code characteristic database. These characteristics are those which are considered important in the choice of an appropriate data partitioning scheme. Characteristics, particular those which inhibit parallelism (number and type of data dependencies), together with the appropriate data distribution, determined by experimentation, are processed for input to the neural network. Codes have been analysed from the following sources: a selection of loops from Banerjee's loops taken from "Loop Transformations for Restructuring Compilers" [11] and Dongarra's parallel loops [12].

7 Definition and coverage of problem space

To ensure that the completed neural network tool can deal appropriately with unseen code of arbitrary complexity it is essential that the training cases reflect the complexities found in real codes. We intend to model these complexities by defining a problem space occupied by exam13s of real code. We define a number of dimensions which delineate this problem space, namely; number of dependencies, type of dependencies, degree of nesting, computational shape presence of conditional statements, subroutine calls and symbolic bounds (figure 1).



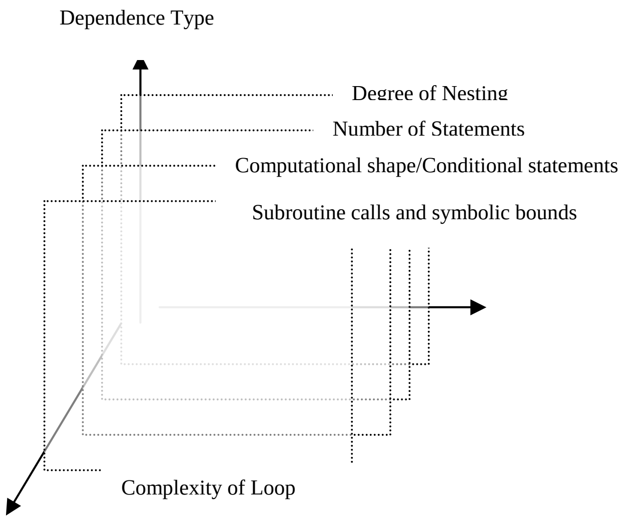

Dependence Type
Degree of Nesting
Number of Statements
Computational shape/Conditional statements
Subroutine calls and symbolic bounds
Complexity of Loop

Figure 1. Diagrammatic representation of problem space.

Codes within this space may be labelled with a complexity level.

Level 1

This is the simplest level and includes that body of codes where it is clear parallelism is not appropriate, as communication will always dominate computation. Nevertheless, a number of examples are needed for completeness. The characteristics of this level are the following:

i. Loops are perfectly nested i.e. only inner loop contains statements
ii. There are a maximum of two statements in the loop body
iii. Loops are normalised i.e. stride =1
iv. All loop bounds are known
v. Arrays within loops may have a maximum of two dimensions.

Level 2

This level is defined by the introduction of data dependencies and an increased number of statements within loop bounds.

Level 3

This level sees the introduction of higher dimensions, statements between loops, conditional statements and varying shapes of computation space.

Level 4

At this level those factors that typically prevent data dependence analysis and data distribution are introduced. These include; linear expression being too complex, symbolic terms in subscript expressions and non-constant direction vectors.

Each piece of code studied may be mapped to a number of points in the defined problem space. To ensure significant coverage four hundred and fifty cases have been identified from the loop corpus and labelled with the appropriate level. These cases cover a number of discrete points within the problem space. However, once trained the neural network represents a continuous model of the problem space. As a result, any unseen input code, provided that it lies within the defined problem space, may be mapped to an appropriate data partition.

8 Training of Network

Example codes have been identified at levels 1, 2 and 3 within the loop corpus. These have been characterised and, by a process of experimentation, the partitioning strategy and most efficient data distribution identified for the target architecture.

8.1 Code Characterisation and Coding

The characterisation scheme used is shown in detail in Appendix A. These are considered to be the principal features exhibited by a loop which influence the choice of data distribution. A binary coding scheme has been developed to code these characteristics for input to the neural network. Categorical fields containing two or more options i.e. data dependencies are encoded as a one-of-n encoding. In some cases it is appropriate to preserve some sort of relationship between neighbouring categories, e.g., whether a particular dependency is uniform, loop carried forward etc. This information is captured by a fuzzy one-of-n encoding scheme.

8.2 Generation of the Data Set

This process was carried out using the following algorithm:

```
REPEAT
    Scrutinise loop
    Sequential code time profiling
    Sequential code results analysis
    Characteristics manually extracted for use
     in production of parallel code
    REPEAT
        Chose Partitioning of Data
        Production of Parallel Code
        Parallel code results analysis
        Parallel code time profiling
        Store time
        Store DPP it stored if time is less than
         previous stored time
    UNTIL   All partitions have been used
    Associate partition with code characteristics
UNTIL  All loops analysed.
```

i.

This knowledge is provided by the expert system, which together with the neural model, draws from and exploits all available knowledge in advising on a suitable data distribution. Actual parallel code is then produced by adding a communications harness using MPI.

To prove the neural component we have, at this stage, fixed the architecture. All timings are taken on a four-processor workstation cluster.

Once this process has been completed each code has been characterised and an appropriate partition identified. This data set is then used to train a multilayer perceptron capable of taking the characterisation of an unseen code and producing the appropriate data partition.

9 An Illustrative Example

To illustrate the process, consider the following code fragment:

```
DO 30, I = 1, rows
      DO 40, J = 1, cols
            A (I, J) = B (I +3, J)
            B (I, J) = A (I, J)
            M = I*J
```

```
          B (I, J) = B(I, J) *A(I, J)
     CONTINUE
CONTINUE
```

The following relevant characteristics for partitioning are:

- 2 arrays
- Input dependence
- 2 Loop carried forward Anti dependence
- 2 Loop independent flow dependence
- Dependencies carried by outer loop

These characteristics are coded for input to the network which indicates that partitioning should be by column. The following table provides relative timings for both sequential and parallel partitioning for an array size of 800 by 800 (variables rows and cols in the code fragment above).

Sequential Time	31.394252
Column = 1	20.670667
Column = 10	20.283152
Column = 25	24.287525
Column = 50	30.393759
Column = 100	45.762204
Column = 200	56.749110

The expert system drawing on these facts together with the partitioning suggested by the neural network indicates a particular distribution i.e. a column based distribution cyclically on four processors with a block size of 10.

10 Further Work

The system outlined above employs a multilayer perceptron trained using a supervised learning approach where both the input (the characterisations) and the output (the associated partition) must be known. This requirement represents a limitation in the system in that, in order to gather the training data, the target architecture must be fixed. The trained network then is applicable only to this fixed architecture. Work is ongoing on an architecture independent approach using an unsupervised mapping network - the Kohonen self-organising network. The Kohonen network will cause the input characteristics to cluster. These clusters may be labelled with a suitable data partition. This information will supplement the architectural details characterised within the expert system.

11 Conclusion

A meta-heuristic approach to data distribution has been developed and implemented. The approach involves the definition and modelling of the problem space, characterisation of the knowledge available and exploitation by appropriate paradigms, namely neural networks and expert systems. Results to date have proven the technique but are limited to a fixed architecture due to the nature of the neural model used. Work is progressing on an alternative approach using Kohonen networks which will result in an architectural independent, automatic data distribution technique.

References:

[1] Ayguade, E., Garcia, J., Kremer, U., "Tools and techniques for automatic data layout", Parallel Computing 24 (1998) 557-578

[2] 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

[3] Mansour, N., Fox, G., "Allocating data to distributed memory multiprocessors by genetic algorithms", Concurrency: Practice and Experience, Vol. 6(6), 485-504(September 1994)

[4] Shenoy, U., Spikant, Y., Bhatkar, V., Kohli S., "Automatic Data Partitioning by Hierarchical Genetic Search", Parallel Algorithms and Applications, Vol. 14, pp 119-147.

[5] Chrisochoides, N., Mansour, N., Fox, G., "A Comparison of optimisation heuristics for the data mapping problem", Concurrency: Practice and Experience, Vol. 9(5), 319-343 (May 1997)

[6] McCollum B.G.C., Milligan P. and Corr P.H., “The Structure and Exploitation of Available Knowledge for Enhanced Data Distribution in a Parallel Environment” ,Software and Hardware Engineering for the 21st Century, Ed. N. E. Mastorakis, World Scientific and Engineering Society Press, 1999, pp139-145, ISBN 960-8052-06-8

[7] K. Kennedy and U.Kremer,” Automatic Data Alignment and Distribution for Loosely Synchronous Problems in an Interactive Programming Environment” Technical Report COMP TR91-155, Rice University, April 1991.

[8] K. Knobe, J. Lukas and G. Steele, “Data Optimisation: Allocation of arrays to reduce communication on SIMD Machines”, Journal

of Parallel and Distributed Computing 8, 102-118 (1990)

[9] Z. Shen, Z. Li and P. C. Yew, An Empirical Study of Fortran Programs for Parallelising Compilers", Technical Report 983, Centre for Supercomputing research and Development.

[10] A. Dierstein, R. Hayer and T. Rauber, "Automatic Data Distribution and Parallelization" Paralel Programming 1995

[11] U.Banerjee "Loop Transformations for Restructuring Compilers", Macmillan College Publishing Company, 1992

[12] J. Dongara, "Atest Suite for Parallelising Compilers: Description and Example Results", Parallel Computing, 17, pp 1247-1255, 1991.

[13] S Haykin "Neural Networks A Comprehensive Foundation" Macmillan College Publishing Company, Inc. 1994.

[14] Kohonen, " The Self-Organising Map", Procedures of the IEEE, vol.78, pp 1464-1480, 1990

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

Appendix A

The Characterisation scheme used.

N^o Characteristic

1) Code section identification
2) Number of loops in code section
3) Number of statements before first Loop
4) Loop Identification
   a) Loop Lower Bound
   b) Loop Upper Bound
   c) Stride of loop
   d) Level of Nesting
   e) Degree of Nesting
   f) Number of statements In Loop Body
   g) Type of Loop Transformation underwent
   h) Sequential timing of loop
5) Number of Flow Dependence
6) Number of Anti Dependence
7) Number of Output Dependence
8) Number of Input Dependence
9) Number of Dependencies
   a) Data Dependence Type
   b) Within Single Statement
   c) Across Statements
   d) Loop Independent
   e) Loop carried forward i.e. Direction vector
   f) Loop carried backwards i.e. Direction vector
   g) Dependence Distance
   h) If Dependence is Uniform
   i) Which loop carries dependence
10) Sequential timing of code section