The Utilisation of Heuristic Techniques in Parallel Code Generation

P. Milligan, B. McCollum, P.J.P. McMullan and P.H. Corr
School of Computer Science
Queen's University, Belfast
Belfast BT7 1NN

P.Milligan@qub.ac.uk
Tel: +44 28 90 335469
Fax: +44 28 90 683890

1. Background

The goal of this paper is to demonstrate the utilisation of artificial intelligence (AI) technology within the field of auto-parallelisation. The paper will demonstrate the relevance and value of using AI approaches by reviewing three parallelisation environments, The Mathematician's Devil, FortPort and KATT. These environments are used as a framework for the discussion on the use of AI in the chosen application area. The paper shows the development and increasing sophistication in the deployment of AI techniques, as a means of resolving problems within the chosen application area. The application area is chosen as it has significant importance in its own right and is one where the utilisation of AI techniques can be shown to have valuable impact.

2. Introduction

Although architectures capable of parallel computation have been available for some time interest in this area of computing remains undiminished. From the introduction of vector and array processors, through interest in low cost clusters, to the recent introduction of computational grids the area continues to present considerable problems for researchers.

While it can be argued that the study of early parallel systems met with considerable success in terms of the software systems developed for use with these machines the same cannot be said for multi-processor systems. The work of Zima et al on vector processing systems yielded high quality compilers. However, it is difficult to find comparable systems for use with multi-processor machines. Heggarty [1] identifies a number of problems associated with both academic and commercial software systems designed for use with multi-processors. In addition, the frequently reported paradigm shift associated with the introduction of the computational grid model is associated with a dearth of middleware for such a system.

The software problems associated with parallel systems arise from the expectations of the main user groups. The principal applications of these machines lie in the area of scientific computation where the main programming language is Fortran. While Fortran has evolved with time from the first ANSI standard for Fortran 77, through Vienna Fortran, Fortran D, High Performance Fortran, to the current Fortran 95, the problems remain largely unchanged.

On the whole, users of multi-processor systems prefer the software to map their programs onto the chosen target architecture. Attempts to encourage users to use special libraries of parallelised code, or to use special parallel programming languages such as 3L Fortran, Occam, etc., have met with little success. Consequently, research has focussed on the development of advanced compilation systems to automatically parallelise code and on development environments to support this automation process.

3. Heuristic approaches

In an attempt to provide the users with valuable autoparallelisation environments attention turned to a number of alternative approaches. One such approach centred on the use of pattern or template

matching (CSCS [2]). In such systems, attempts are made to identify the functionality of sections of sequential code and to replace these with highly parallelised alternatives. Systems using this approach appear to have found a use in certain specialised areas but have not been successful in a general sense. A more widely adopted approach was the use of performance estimation, e.g. Zima and Faringher [3]. In this approach, the execution performance of a program is estimated with the goal of identifying the most computationally intensive sections. These sections of code are then parallelised to improve the overall performance of the program. An equally promising approach arises from the application of artificial intelligence (AI) techniques. For example, genetic algorithms simulated annealing and formal mathematical notations [4, 5] have been brought to bear. However, to date few of these approaches have been used within an overall parallelisation environment.

The paper reviews three programming environments, The Mathematicians Devil [6], FortPort [7] and KATT [8], with the aim of demonstrating how the use of AI techniques has enabled new insights to be made into the field of autoparallelisation and a novel approach, based on the use of hyper-heuristics, is proposed.

4. The Mathematician's Devil

The first use of AI techniques within the autoparallelisation work at Queen's University Belfast (QUB), was the employment of an intelligent knowledge based system (IKBS) as part of the Mathematician's Devil. Within this context, the IKBS was defined to be composed of two main components: a knowledge base and an inference engine. The goal of the Mathematician's Devil was the provision of a set of tools to assist mathematicians with the specification and implementation of linear algebra software on multiprocessor architectures.

Programs developed within the Mathematicians Devil were expressed using a high-level Pascal-like notation called SIMPL. A significant feature of SIMPL was that it provided access to libraries of highly parallelised routines based on the BLAS. The system converted programs expressed in SIMPL into a tuple sequence (in effect a graph), where each tuple reflected the type of the source and destination data and the nature of the operation/function being used. A graph transformation phase then performed a data dependence scan to identify and remove inter-statement dependencies. During this scan, each tuple was expanded to contain unique label and dependence list information. In essence this stage of the process replaced tuples (as required) with sequences of simpler tuples, involving the use of temporary variables, thereby eliminating the dependencies. This expanded graph formed the input to the core of the system, the subroutine selection phase.

The selection of appropriate library routines was performed by the IKBS, which in turn produced a process-to-processor map containing a specific allocation of library subroutines to machine processors. To enable this mapping to be produced correctly the knowledge base component of the IKBS was developed to contain three pieces of information:

- the numerical operations to be distributed (provided by the graph derived from the original user program)
- the numerical algorithms which are available to implement each operation (provided by a system maintained library definition)
- the topology of the target machine (provided by a user supplied abstract topology definition).

The inference engine employed an iterative algorithm, with each iteration producing subroutine selections for those tuples whose pre-conditions were satisfied. The engine was constructed from two parts:

- an *extractor* which separated out those tuples whose pre-condition lists are satisfied, and labelled these with a unique level number, and
- an *allocator* which selected suitable library subroutines for each tuple within the extracted level and employed an adapted curve fitting algorithm to produce a balanced distribution of these tuples (processes) over the target architecture (processors).

The balanced distribution was achieved by giving the slowest operation highest priority (allocating it a relatively large portion of the available processor topology). Faster operations were restricted to smaller portions of the topology. The prototype system was successfully implemented, tested, and functioned very well within its intended environment. However, it has limited applicability as it was restricted to a specific application area and used a very specialised programming language. To capitalise on the advantages promised by this first use of AI a new system was proposed that would use a more general programming language (Fortran) and would not be restricted to a specific application area.

5. FortPort

5.1 General

The new system, called FortPort, consisted of a suite of transformation and code-restructuring tools. In operation, its basic functionality resembled that of the Mathematician's Devil in that:

- an input handler accepted a sequential source code and converted this to an intermediate form; a hierarchical graph which described the syntactical structure of the code,
- this graph formed the input to a transformation stage, which restructured the graph in an attempt to remove or reduce potential obstacles to the production of a parallel solution while ensuring that the semantic meaning of the original code remained consistent throughout all transformations,
- generation and evaluation stages were then used to convert the transformed graph to a parallel form (known as the potentially parallel graph) and to create a set of parallel processes.

At this point two AI techniques were incorporated into the system, one based on hot spot analysis, and the other involving a sequence of expert system guidance tools.

5.2 Utilisation of Hotspot Analysis

Experience has shown that many Fortran programs spend most of their execution time within small sections of computationally expensive code. If these code fragments can be identified then this information can be used to inform the auto-parallelisation process. In other words, this information can be used to form another heuristic that is used to improve the compilation process.

Hotspot analysis was added to the system by developing a new tool that manipulated the graph created by the source analyser. This tool had three levels of operation: subroutine level, loop level and library routine/function level. The first level allowed a user to identify the subroutine, or subroutines, that required the most execution time and then, using the other two levels of analysis, to find the most computationally intensive loop or library calls. Within each of these analysis levels, the same basic approach was adopted. A version of the original program, enhanced by the addition of profiling statements, was executed on one processor of the target multi-processor architecture creating a file of timing diagnostics. Subsequently, this file was analysed and the relevant data extracted.

This new information enabled the transformation tool to focus attention on the computationally intensive components of the user program rather than wasting time trying to parallelise the entire code.

5.3 Introduction of Expert Systems

A number of knowledge sources are available within the environment, e.g. performance estimation data generated by hotspot analysis, the occurrence of obstacles such as data dependence, limiting factors derived from the nature of the target architecture, etc.. This information can be gathered to form a knowledge pool that can be mined to provide expert guidance to the parallelisation process. Such information can be generated automatically, e.g.

- the information supplied by hotspot analysis
- the results of executing codes on parallel systems

or can be provided by system developers, e.g.

- the descriptions of target architectures, how many processors are there, how much local memory do they have, how are they interconnected, what the bandwidth of the system is, etc.

This information can be exploited by the use of expert systems. The first source used within FortPort was the data generated by the hotspot analyser. This data was used by a partitioning tool (an expert system) to compile a list of possible distributions for the sequential code. Although there are a limited number of distribution solutions, it is possible that a large list of potential transformation solutions can be produced and subsequently require testing to determine the most suitable, i.e. the transformation that provides the best performance results when executed on the target parallel platform.

This optimum solution is found by providing an improvement cycle within the system. Each possible transformation is applied to the user code and after execution the results are stored in a database. A "best-so-far" process is employed to keep a record of the most effective transformation during the improvement cycle.

While the iterative methods applied within the improvement cycle provide a method for finding the optimal transformation solution a major disadvantage can be identified. Given a sequential code with a large number of possible transformations, the problem lies in finding the most effective transformation in a time frame acceptable to the user of the system. The improvement cycle may require a large number of iterations in order to ensure that the best performance has been found. The expert system provides guidance for the choice of optimisation parameter values. However, it cannot dismiss possible solutions in the belief that the performance results will be poor. All potentially performance-improving techniques must be applied. Testing one solution itself may take some time, therefore obtaining parallel performance statistics for many configurations can be considered unfeasible.

A process was required to reduce the potential volume of testing, while retaining the power to find the optimum solution for the problem. One method was to use the 'experience' gained from a historical record of past parallelisation cases. This required a database of optimum transformation solutions for a wide range of codes previously parallelised (by hand or using iterative methods). Each problem, and the associated most effective solution, are stored in the database along with the performance results obtained. It then may be possible to find a matching problem within the database, and use the previously recorded transformation. The stored performance results indicate the potential parallel execution performance that could be obtained by applying this transformation to the sequential code.

The database effectively becomes a knowledge base to supply satisfactory transformations quickly without the need to test for all possibilities. The pattern matching capabilities of modern expert system shells make the creation of such a knowledge base within the system relatively straightforward.

This method is effective for problem cases to which an associated solution exists within the knowledge base. However, there are further issues to address regarding the operation and effectiveness of this method. A knowledge base with a limited amount of cases may not contain any codes similar to the code under consideration and make this method unproductive. Furthermore, due to the complexity of structural programming languages, codes that are similar may exist in the knowledge base, but may not be similar enough to guarantee the effectiveness of the associated solution when applied to the sequential code.

If a degree of 'fuzziness' could be introduced to the technique then matches could be made based on relevance to the original problem. If the current case matches a known case in the knowledge base almost exactly, the relevance is high. As the degree of similarity between cases decreases, the measure of relevance decreases accordingly.

6. KATT (Knowledge Assisted Transformation Tool)

The expert systems used in the output handler (Code generator) within KATT has achieved a degree of success but to date has taken a rather narrow view of the field of knowledge engineering. By themselves, expert systems offer only a limited representation of the full knowledge available to the

system. A broader knowledge utilisation strategy is proposed in which the top-down model exemplified by expert systems is combined with the bottom-up model associated with neural networks. Neural networks are useful for fast identification of implicit knowledge by automatically analysing cases of historical data. It is believed that the exploitation of an appropriate neural paradigm will significantly enhance the effectiveness of the development environment in providing the expertise necessary to effectively distribute code. The meta-heuristic approach provides an environment within which the strengths of the expert system and neural network can be used in a complementary manner to provide problem solving with the likelihood of a high degree of success. To date, no such combination of approaches has been applied to the problem of data distribution.

Neural networks have a unique set of characteristics. They can learn from experience, generalise from examples and abstract essential information from noisy data. 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 an output, has ensured that they have found application in many diverse disciplines.

The features stated above, together with the fact that neural networks have been successfully applied within the KATT environment for the identification of sequences of program transformations or the reduction or elimination of data dependencies, make their use attractive in this research.

Neural networks have achieved notable success in other areas where heuristic solutions are sought. Rather than the neural network actually trying to solve the combinatorial optimisation problem of data distribution, the output of the neural network will be classified into a number of groups based on the characteristics of the original program. This grouping will represent inherent features in the code, which it is proposed will indicate which data partitioning is more appropriate. This approach of applying pattern matching techniques to the problem of data distribution to date has shown promising results.

The use of neural technology will complement the existing expert system within the KATT environment in advising on appropriate partitioning strategies through exploitation of relevant knowledge implicit in the code itself. It is therefore critical that the essential information pertinent to data partitioning is captured as input to the network.

The neural network tool is responsible for the choice of a data partitioning strategy. The knowledge required to decide on this is code specific knowledge, i.e. facts about code, and programmer specific knowledge (parallelisation knowledge). Given an input of essential information about a code and an output of the most appropriate data partitioning to apply to that code, the learned mapping held by the network effectively encapsulates knowledge which may be brought to bear on the process.
The expert system is responsible for mapping the chosen partition onto the target architecture. The knowledge required to decide on this is code specific knowledge (loop bounds), problem specific knowledge (speed-up required) and architecture specific knowledge (architecture characteristics).

Given the suggested partitioning, architecture specific knowledge is required before the system can determine an actual distribution on the target architecture. This is best represented explicitly within an expert system but currently held implicitly, hard coded for a particular target, within the current KATT system. The expert system together with the neural model advise on a suitable distribution strategy. Combining these approaches within a coherent framework has improved the ability of KATT to offer strategic intelligent guidance to the user through broader and deeper access to the knowledge pool.

7.0 KATT+: A Hyper-Heuristic Model

Heuristics (derived from the Greek for discovery) are criteria, methods, or principles for deciding among several alternative courses of action. Computerised techniques which attempt to incorporate such “rules of thumb” are termed heuristic techniques. The hyper-heuristics approach differs from meta-heuristic approaches, which refer to heuristics which control simpler heuristics for a narrow range of problems, rather than of choosing between a range of heuristic approaches to solve a wide

range of problems. This is a term coined to describe the idea of using a number of different heuristics together, so that the actual heuristic applied may differ at each decision point. In essence, hyper-heuristics are heuristics to choose heuristics. The hyper-heuristics provides a method for managing these simple heuristics, which are poor at producing effective solutions when considered simply, to yield a good solution overall. An example of the use of this concept is given in [9], in which a genetic algorithm (GA) is applied to open-shop scheduling problems, to evolve the choice of heuristic to apply whenever a task is to be added to the schedule under construction.

The hyper-heuristic approach represents a novel and exciting area of research and was been developed and successfully applied to scheduling problems by the Automated Scheduling and Planning Group at the University of Nottingham. The KATT environment, as presented, provides an overview of the development of an automated/semi-automated parallelisation environment and details the approaches at Queen's University to incorporate several heuristic based techniques into a common hyper-heuristic framework.

8. Conclusions

It has been accepted that the production of a fully automated parallelisation environment that will deliver fully optimised code, for a range of target architectures, within an acceptable time frame must remain an unattainable dream. The work reported here has demonstrated that the utilisation of AI technologies can enable parallelisation environments to be developed that will produce acceptable solutions within reasonable time frames by offering expert guidance to system users. The deployment of a range of AI techniques, expert systems and neural networks, and their combination via the use of a meta-heuristic model has enabled significant advances to be made.

The planned introduction of further AI techniques (case based reasoning and genetic algorithms), and the use of the novel hyper-heuristic model will enable further advances to be achieved.

9. Limited Bibliography

1. J. Heggarty, "Parallel R-Matrix Computation", PhD Thesis, Queen's University Belfast, 1998
2. R. Rehmann, "Automatic Generation of Programs for a Scientific Parallel Programming Environment", Swiss Scientific Computing Centre, ETH Zurich, pp 1-9, CSCS-TR-94-02, 1994
3. H. Zima, "Automatic Vectorisation and Parallelisation for Supercomputers", Software for Parallel Computers, pp107-120, Chapman and Hall, 1992
4. 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
5. N. Mansour, and G. Fox, "Allocating data to distributed memory multiprocessors by genetic algorithms", Concurrency: Practice and Experience, Vol. 6(6), pp485-504, September 1994
6. T.J.G. Benson, P. Milligan, R. McConnell and A. Rea, "A Knowledge Based Approach to the Development of Parallel Programs", IEEE Computer Society Press, ISBN 1066-6192-92, pp 457-463, 1992
7. R. McConnell, P. Sage, P. Milligan, A. Rea and P.J.P. McMullan, "Hot Spot Analysis within the FortPort Migration Tool for Parallel Platforms", Microprocessing and Microprogramming 37, pp141-144, 1993
8. P.H. Corr, P. Milligan and V. Purnell, "A Neural network Based Tool for Semi-automatic Code Transformation, Springer Lecture Notes in Computer Science 1981, pp 142-153, 2001
9. H-L Fang, P.M. Ross and D. Corne, "A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems", Proceedings of ECAI 94, pp 590-594, 1994