Data Distribution, Analysis and Evaluation of code - An Expert System Approach

P. J. P. MCMULLAN, P. MILLIGAN and P. H. CORR
School of Computer Science
The Queen's University of Belfast
Belfast BT7 1NN
NORTHERN IRELAND

Abstract: - Considerable research has been carried out into the problem of porting sequential legacy codes to a parallel equivalent for execution on multiprocessor computer systems. The potential improvement in execution performance has made this a worthwhile task, although there has been limited success in the search for a solution. The systems which have made the most advances in this area are those which utilise existing knowledge relating to the problem. This paper focuses on a system which employs a suite of knowledge based expert systems to provide intelligent parallelization of legacy codes. Each stage of the transformation process is illustrated with examples and justification of the expert system decision making processes. An illustrative case study based on a mathematical Fortran code is used to demonstrate the success of the system.

Key-Words: - Automatic, Intelligent, Parallelisation, Distribution, Expert-Systems, Knowledge-Based, Multiprocessor
IMACS/IEEE CSCC'99 Proceedings, Pages:5241-5246

1 Introduction

There has been considerable research over the last number of years into systems aimed at transforming existing legacy programs to an equivalent form for execution on multiprocessor systems with the intention of obtaining an improvement in execution performance [1,2,3]. This has led to the realisation that the level of expertise needed from such a system is similar to that which a human expert applies when parallelizing sequential programs by hand.

The responsibility of providing expertise still lies with the user in existing parallelization systems. Data distribution is one of the major obstacles limiting pure automation within the parallelization process. A system which automatically provides effective data partitioning algorithms can be considered suitable in replacing the human expert [4,5,6,7]. The Fortport project [8,9,10] provides a solution to this problem by presenting a suite of tools to apply AI technology in the parallelization of sequential codes. This approach is based on an underlying knowledge model to influence the transformation process.

2 The Fortport Solution

The Fortport system consists of a number of inter-related components, as shown in Fig. 1. The Input handler accepts Fortran 77 source code and converts it into an intermediate form; a hierarchical syntax graph. This provides an effective structure for transformation of the program code. The Transformation stage uses transformation tools guided by an expert system to restructure the syntax graph and remove data dependencies. The transformed graph, known as the potentially parallel graph, is used as input to the Generation stage.

The Generation stage consists of a suite of analysis and distribution tools again guided by an expert system which produces a parallel equivalent of the input sequential program. The generated program is executed on the target architecture. The results obtained are analysed by the evaluation tools to determine if additional refinement or a new data distribution strategy is required. Feedback is then provided to the generation stage. The feedback cycle continues to redefine the parallelization strategy until the optimum parallel performance has been achieved.

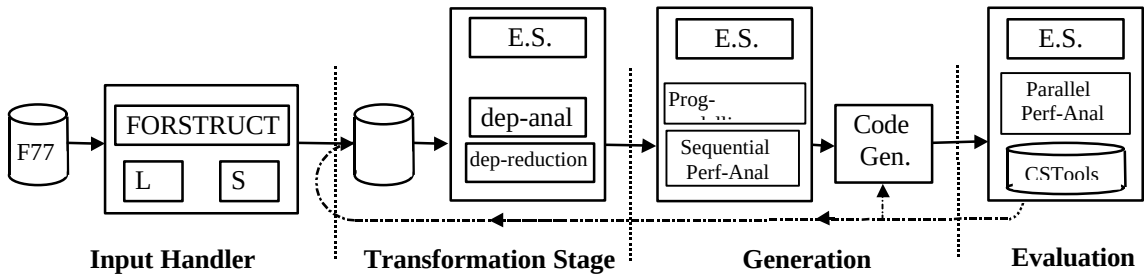

Fig. 1 The Fortport System

3 Knowledge Sources

The knowledge base used by the expert system is built upon a number of knowledge sources. These sources include *Architecture specific knowledge*, *Source code characteristics*, *Expert System parallelization knowledge*, *Expert User parallelization knowledge*, *Sequential and Parallel performance statistics* and *Parallelization strategy records.*

The human expert uses each knowledge source to influence the steps involved in parallelizing a sequential program. To emulate this process effectively, the knowledge must be captured and stored effectively. A mechanism is also required which enables the information to be used to form decisions in a similar manner to the reasoning processes of the human expert.

4 Knowledge-based Expert Systems

Two approaches to utilizing knowledge are with the use of Neural Networks and Expert Systems. Both forms of artificial intelligence allow patterns or rules to influence the structure or outcome of decisions. The KATT system uses Neural Network technology to isolate characteristics to influence data distribution in the parallelization of legacy codes.

Fortport uses the CLIPS Expert System to examine code at a structural level. The Expert System is beneficial for the problem at hand, allowing the use of knowledge sources such as expert system parallelization strategies. The novice user can rely on expert system decisions, whereas the expert user can influence the decision making process.

Throughout generation and evaluation, a knowledge base is maintained to record and utilize relevant information obtained from the pool of knowledge sources, such as an appropriate parallelization strategy. Information within this knowledge base forms the parameters which influence the entire transformation process. The expert system tools utilize the knowledge base to drive the generation and evaluation stages. The knowledge base is also used as a record of successful parallelization strategies.

5 Generation and Evaluation

The generation and evaluation stages comprise five main tools, as shown in Figure 2; *Program Modeler*, *Sequential Performance Analyser*, *Data Distribution Tool*, *Code Generation* and *Parallel Performance Analyser*. Each of the tools consist of several components which handle analysis, transformation and decision making processes.

The Program Modeler is used to perform a source code analysis of the transformed program in an attempt to model the behaviour of the program when executed. This information is used to highlight areas of the code for further analysis. The modeler also builds up a database of information, the characteristics of certain sections of code, in order to help identify suitable parallelization strategies. The Program Modeler includes a Program Model Analyser and a Profile Guidance Evaluator. The Profile Guidance Evaluator uses estimation results from the Program Model Analyser to decide on the best profiling technique to be used in subsequent sequential performance analysis.

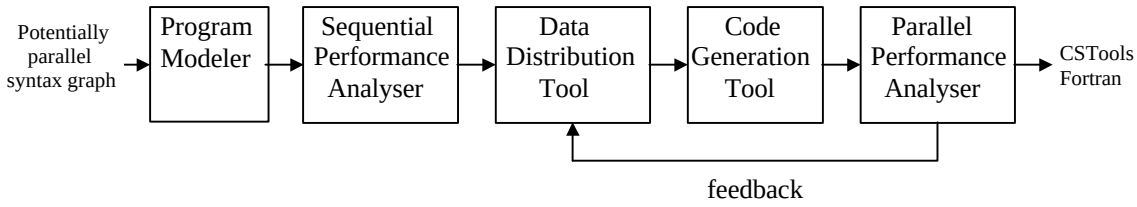

Fig. 2 The flow of control between modules

The Sequential Performance Analyser identifies sections of the sequential program for which parallelization will be most beneficial. These areas are identified from the results of profiled sequential execution, which highlights computationally intensive areas of the sequential program. Parallelization tools are then concentrated on these hotspots. The Sequential Performance Analyser contains a Code Profiler, which produces a sequential version of the program which, when executed, produces real-time performance information. This information is then used by the Sequential Performance Evaluator to decide which areas are conentrated upon in the distribution phase.

The Data Distribution Tool is the main parallelization tool within the system. It converts from sequential to parallel, generating processes and distributing work among the processors. Source code analysis and profiled execution analysis influence the parallelization decisions taken. Characterization and Distribution Analysis is performed on the sequential code using the information obtained earlier. A Strategy Evaluation mechanism then builds a list of parallelization strategies, from which the best apparent strategy is chosen. Alternative strategies can be chosen from the list based on further parallel performance analysis.

The Code Generation Tool creates the parallel source code for the parallelized equivalent of the original sequential program. The Parallel Performance Analyser uses the execution of this parallel version to identify performance improvements which can be made and isolate the most effective parallelization strategy. A Parallel Code Profiler provides real-time parallel execution performance results. The Parallel Performance Evaluator determines how successful each parallelization strategy has been in achieving a performance improvement, both in terms of the sequential program and previous parallelization attempts. An improvement cycle is implemented to pass control back to the Data Distribution Tool.

The user can choose to interact with the system at any stage, and can influence choices and decisions based on human expertise. The novice user may choose not to interact with the system, in which case decisions are left to the expert system.

5.1 Profile Guidance Evaluator

The performance estimation results of program model analysis are used to decide which sections of code require performance analysis and how much profile information is required and obtainable. Under ideal circumstances all parts of a sequential code can be profiled. However, factors which may restrict this include sequential execution time, profile degradation and information excess. There are three main steps in the operation of the profile guidance evaluator: initial evaluation (is profiling possible), hotspot evaluation (Potential hotspots) and profile level evaluation (amount of information possible and required).

Hotspot evaluation requires estimating the potentially significant (computationally intensive) areas of code. The evaluator decides whether the area may be a hotspot and the level of profiling required, based on the estimated performance results. The user can decide to preclude areas which, based on their expert-user evaluation may not signify as a hotspot. The level of profiling for each hotspot can also be changed. An example of one of the rules used by the expert system to choose profiling level follows:

```
(defrule mem-1
  (profile-usage  (level all)
  (amount ?needed)
  (accuracy 100))
```

```
(mem-allows  (amount ?available)
             (accuracy 100))
(test (> ?needed ?available))
=>
(assert (advice profile-level all)))
```

The memory available and memory required for the desired level of profiling are compared. If the memory resources are sufficient, profiling at this level is possible. A new expert system rule indicating this is created.

5.2 Sequential Profile Evaluator

Whereas performance estimation is a guide to locating potential hotspots, the Sequential Profile Evaluator bases decisions on actual performance statistics. Compute-intensive sections of code are identified and distribution techniques are applied in an attempt to improve overall program performance. However, some sections of code may still be considered unsuitable for parallelization, due to the existence of data dependencies between statements or input and output statements. Although in some cases data dependencies can be removed, input/output is treated as a serious obstacle. The evaluator is designed to identify this and remove such sections from the parallelization target area:

```
(defrule check-input-output
  (loop    (loop ?loopnum)
  (input-output true))
  ?old <- (fact hotspot loop ?loopnum)
  =>
  (printout "Loop cannot be
       parallelized due to I/O")
  (retract ?old))
```

The evaluator can advise to terminate the parallelization process for cases where no hotspots exist or sequential execution time is too small to warrant parallelization. The expert user can over-ride such decisions.

5.3 Data Distribution Strategy Evaluator

The Data Distribution Strategy Evaluator will make decisions on how best to parallelize the hotspots identified. Decisions made are based on the structure of the code under parallelization and feedback, if any, resulting from performance analysis from previously attempted data distribution strategies within the parallelization cycle. The distribution strategy is chosen from a set of possibilities in "best first" order. Distributions possible are dictated by characteristics of the hotspot code; for example, the amount and type of data dependencies which exist between statements in the body of a loop. When a particular distribution is chosen, further evaluation will decide on a more detailed strategy based on the code structure.

The factors which distribution influence are types of data partitioning available, processors required and processor communication, message-passing optimization and load-balancing. For example, the choice of optimal number of processors for a parallelization strategy is dictated by the loop bounds of a loop-based hotspot:

```
defrule loop-bound-check-1
   (loop ?loop ?iterator ?bound)
   (test (>= ?bound 16))
   (test (= (mod ?bound 16) 0))
   =>
   (assert (incr-popularity-16)))
```

The History Knowledge Base can provide information on previously successful distribution strategies which have employed on programs with similar characteristics.

5.4 Parallel Profile Evaluator

The Parallel Profile Evaluator evaluates the results of parallel performance analysis to indicate the level of success for the current parallelization strategy. Decisions are fed back to the data distribution stage, forming the improvement cycle. The improvement cycle is concerned with finding the optimum parallel performance, using a Situation and Result comparison for each phase of the cycle. The Situation defines strategy information such as data distribution, processors used, load balances and communication flags. The Result for one situation comprises hotspot execution time, computation time, communication send/receive time and communication idle time.

Success evaluation can establish whether the current strategy is the most effective. Feedback to the data distribution stage involves choosing possible improvements and which may be most effective. Improvements which have already been applied are disregarded:

```
(defrule decide-15
   (okay-to-rebalance)
   (attempted-advice rebalance)
    =>
   (printout t “Rebalance
    already attempted” crlf))
```

5.5 The History Knowledge Base

The History Knowledge Base is a structured collection of program code characteristics and associated parallelization strategies built from many previous parallelization attempts. Whether effective or not, these strategies can be used to compare against similar programs fed into the system in order to further influence decisions during the parallelization process. During each stage of evaluation, program characteristics, parallelization strategies attempted and associated results are included in the Knowledge Base.

6 Illustrative Case Study – A Hessenberg-Schur Algorithm

To demonstrate the viability of the Expert System approach in automatic parallelization of legacy codes, a case study is introduced. The case study is a non-trivial Hessenberg-Schur Algorithm which is implemented as part of a mathematical package called the Sylvester system. The program is called “SYLG” and is used by The Department of Applied Mathematics and Theoretical Physics at The Queen’s University of Belfast [11].

The original “SYLG” code is input to the pre-parallelization transformation tools. The code is then ready for analysis and tranformation to parallel form. The program modeller builds up a list of estimated performance characteristics. These indicate which areas of code are potential hotspots. Four loop blocks are identified, and the percentage of the total estimated execution time for each is output. One loop is identified as compute intensive:

```
Loop 1 label 130 takes 91% of
full execution time
ASSERTION: Loop 1 is compute-
intensive
Taking this advice
```

The expert system determines that this loop block will require profiling in order to determine precisely whether it is compute intensive. Sequential profiling takes place and the expert system analysis proves that the loop is indeed a hotspot:

```
Loop 1 label 130 takes 18.27 s
ASSERTION: Loop 1 is compute-
intensive
Taking this advice
```

Parallelization can be concentrated upon this section of the “SYLG” code. The expert system records the full structure and performance characteristics of the loop block and the statements within the loop body. These can then be used during the parallelization cycle to influence data distribution strategies. The hotspot loop is a complex loop block containing three inner loops, one of which has a further level of nesting. Although there are no dependencies within the loop block the expert system identifies an induction, limiting the number of distribution strategies possible. The expert system determines that the outer loop is the only one not affected, so distribution is concentrated on this loop. Parallel profiling within the improvement cycle provides information with which the expert system determines the most successful data distribution. This will be the distribution which provides the best performance speedup between the sequential and parallel versions of the code. The improvement cycle determines the best distribution, which splits the loop over 4 processors using a COLUMN partitioning scheme. A speedup is obtained:

```
The current best performance has
been achieved as a result
of the previous advice. This
strategy is now stored
The best speedup is now 1.24
```

The strategy details are stored as this may ultimately form the final parallelization solution. However, communication optimization strategies are also applied in an attempt to further improve the resultant speedup. The expert system determines that staggering communication messages between the slaves results in a further improvement in speedup:

```
The current best performance
has been achieved as a result
of the previous advice. This
strategy is now stored
The best speedup is now 1.25
```

No further potential improvements are possible, and the improvement cycle terminates with a speedup of 1.25. This speedup is significant for intensive use of the package, i.e. when large sets of data are processed, normally requiring a lengthy processing period.

The final step stage in the process is to store this successful strategy in the knowledge base, along with the characteristics of the code to which this strategy was applied. The parallel version of the code is output and marks the final stage of the parallelization process.

7 Conclusion

The Fortport system set out to emulate human expertise for the problem of automatic parallelization of Fortran code. The steps involved in this process are separated and each are controlled by a main decision making expert system. Parallelization attempts produce both performance results and information for possible further performance improvement. The system also learns techniques and strategies for programs with common identifiable characteristics.

The system has been tested for many cases of common program constructs and algorithms, with satisfactory performance results. The system has also been tested using an existing scientific problem; the mathematical Sylvester algorithm. This case study has been demonstrated to produce a performance improvement. Expert user intervention was not required in attaining the speedup. Therefore the expert system approach within the Fortport system is effective in solving the problem of intelligent automatic parallelization.

References:

[1] J. Hulman, S. Andel, B. Chapman and H. P. Zima, Intelligent Parallelization within the Vienna Fortran Compilation system, *Proceedings of the Fourth Workshop on Compilers for Parallel Computers*, 1993, pp 455-467

[2] 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

[3] High Performance Fortran Forum, *High Performance Fortran Language Specification, Version 1.0*, Rice University, Houston, May 1993.

[4] 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.

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

[6] K Decker, J Dvorak and R Rehmann, *A knowledge-based scientific parallel programming environment*, Swiss Scientific Computing Centre, 1993.

[7] S. Andel, B. M. Chapman and H. P. Zima, An Expert Advisor for Parallel Programming Environments and its Realization within the Framework of the Vienna Fortran Compilation System, *4th Workshop on Compilers for Parallel Computers*, Delft, 1993.

[8] 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.

[9] 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

[10] P. J. P. McMullan, *The Intelligent Generation and Analysis of Code for Parallel Platforms*, PhD thesis, QUB, 1996.

[11] D. M. Tiernan, *Collocation Studies in Fracture Mechanics and Quantum Mechanics*, Ph.D. Thesis, QUB, 1996.