Compilation Issues for High Performance Computers: A Comparative Overview of a General Model and the Unified Model

Brian J. d'Auriol
Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada, R3T 2N2

Baqui Billah
Department of Computer Science and Engineering,
Wright State University,
Dayton, Ohio,
USA, 45435

Abstract

This paper presents a comparison of two models suitable for use in a compiler for high performance computers. The first model, termed 'the general model' is based on well-known data dependency techniques and a summary based on Banerjee's presentation is given. The second model, termed the 'Unified Model' is a recently proposed model which approaches the compilation issue from a different viewpoint. A summary of the unified model is given. Issues involved in compiling for high performance computers are discussed. These issues form the criteria for the comparison of the models.

Keywords: High Performance Computers, Compilation Issues, Models for Parallelizing Compilers.

1 Introduction

Sufficiently complex compilers are required in order to exploit the full potential of a parallel high performance computer. Typically, these types of compilers transform sequential code into equivalent parallel code to be executed efficiently on the target machine. Over the past three decades many techniques have been proposed; some of which have matured. Excellent discussions of many of these techniques can be found in such sources as [1, 2, 3, 4, 5]. However, much research work remains in this area. A new model suitable for compiling systolic computations for multicomputers has recently been proposed [6, 7, 8, 9]. This model considers the problem of compiling for high performance computers from a somewhat novel, although related viewpoint.

In this paper, we make a comparative study between a general model based on loop dependency analysis and transformations with the newly proposed model for compiling systolic computations for multicomputers. The general model is based on discussions given by Banerjee in [1, 2] while the newly proposed model, termed the 'Unified Model', is given by d'Auriol and Bhavsar in [6, 8, 9]. The general model has at its core, the data dependency analysis techniques which, in effect, allows a compiler that incorporates this model to determine the various dependences which inhibit parallelization and to perform a suitable restructuring to reduce or eliminate such inhibitations. A compiler which incorporates this model is expected to efficiently parallelize the given input algorithm. The unified model has a similar goal, namely, to efficiently parallelize a given input algorithm, however, it does so by incorporating the systolic processing model as the mechanism to determine parallelism inherent in the computation.

The purpose of this paper is to briefly present an overview of both models. Issues and criteria which will be used in a comparative evaluation of both of these models will be discussed. Similarities and differences of both models will be discussed. Included in our discussions is an illustrative example of the applicability of each model.

This paper contains the following sections. Section 2 discusses issues and criteria by which the comparison will be conducted. The following two sections, Section 3 and Section 4, will present overviews as well as a brief example of the application of the general and unified model respectively. Comparative discussion is given in Section 5 and conclusions are drawn in Section 6

2 Issues in High Performance Compilation

A high performance compiler should incorporate both machine independent (e.g. automatic restructuring) as well as machine dependent optimization techniques. The following issues are related to the integration of these two types of optimization areas. We note that not all issues may impact upon a particular compiling environment.

1. **Conversion of Sequential Code:** Most programs for high performance computers are written in some version of Fortran or, more recently, C. Of particular interest in sequential code conversion is the emphasis placed on the parallelization of sequential loop (iteration) structures and the corresponding statements

inside such structures.

Conversion of sequential code is necessary since (a) many legacy systems exist and (b) programmers continue to favor implementations in such languages (often for reasons of convenience). Enhancements to sequential languages (e.g. parallel C) imply increased ability to detect not only parallelism but also potential conflicts.

2. **Diversity of Architecture:** A high performance compiler should be able to generate code that would be execution time efficient on a target machine. Due to the diversity of the various candidate parallel machine architectures, code generation poses challenges in constructing compilers. Flow analysis and program optimization are two examples which are influenced by the architecture of the target machine.

3. **Portability and Availability:** The compiler should be scalable — intelligent enough to distribute a program to the number of processors available. Generality and machine independence in optimization increases portability. On the other hand, optimization of code can only be maximized if it is done with respect to the underlying architecture. One way to resolve this dichotomy is to run a native back-end optimizer that generates highly efficient code on that particular target machine.

4. **Loop Scheduling:** Efficient scheduling of parallel loops among the available processors is required. The choice of a particular scheduling technique depends both on the program and the target architecture. Each iteration of a parallel loop may be considered as a work unit. It is desirable that computational work load or iteration be distributed as uniformly as possible among the available processors across the period of execution. Often, the issue of scheduling is machine dependent.

5. **Execution Model:** The model of how a program would be executed on multiple processors is an issue closely related to the scheduling problem. This issue is much dependent on the target architecture and the operating system running on it. For example, the master-slave model has been discussed in [3]. For certain programs, parallel execution technique using divide and conquer strategy may be used.

6. **Data Layout / Partition:** In the above discussions on execution models, it was tacitly assumed that data is automatically distributed among the processes. In parallel computing, each process is responsible for processing a certain amount of data (shared or non-shared) associated with it. The method used to distribute the data so that efficiency is maintained is an issue. This issue is dependent on both the nature of the target machine and that of the program to be executed in parallel.

   Some data distribution methods include the following: (a) striping or checker boarding distribution, (b) cyclic or non-cyclic distribution, (c) template assignment (used in FORTRAN 90), (d) data replication (several copies of a single variable), (e) automatic data layout, and (f) dynamic alignment as determined by data flow analysis.

7. **Synchronization:** In a loop, if there is no data dependence across the iterations and each iteration can be executed independently, no synchronization is required. A restructuring compiler often transforms a sequential loop into one that can be executed in parallel with some kind of dependences. These dependences may be within different array elements (different iteration) of the same loop, or may be across loop nests. Inherent in this analysis and subsequent transformation of loop nests is the danger of introducing inefficiencies.

8. **Exceptions and Debugging:** Exception handling and/or providing run-time assistance to execution debugging is often a machine dependent issue.

9. **Performance and Efficiency Evaluation:** It is frequently important to be able to measure or estimate the potential increase in speed due to a parallelized implementation.

3 Overview of A Generalized Model

The basis of the model which would be used in a compiler can be found in [1, Chapter 5](Banerjee). We assume for this paper that the full model can be described by the algorithm given by Banerjee. We recognize that additional details would need to be considered prior to an actual implementation.

We formulate the generalized model as the tuple $\mathcal{G} = (\mathcal{L}, \mathcal{D}, \mathcal{T})$ where $\mathcal{L}$ is the loop nest under analysis, $\mathcal{D}$ is the set of conditional operations to determine the existence of dependences between the statements in $\mathcal{L}$, and $\mathcal{T}$ is the set of processes which transform $\mathcal{L}$ (with possible dependences as determined by $\mathcal{D}$)into a semantically equivalent loop nest $\overline{\mathcal{L}}$ which may be parallelized or vectorized to a greater extent than the original loop nest.

In the following discussion (the notation is taken from[1]), m refers to the number of loops in the loop nest, A and B are $m \times n$ matrices, a_0 and b_0 are integer n-vectors, I is an m-vector representing the loop index variables and the equations $(IA+a_0)$ and $(IB+b_0)$ denote the memory index of two variables (arrays for example) which participate in possible data dependence relations within the context of the loop nest $\mathcal{L}$.

An algorithm presented by Banerjee [1][Algorithm 5.1] is summarized herein as part of an example of the application of this model.

Step	Description
1	Obtain or compute values for m,n,A,B,a_0 and b_0. This may be referred to as the initialization step.
2	Compute a $2m \times 2m$ unimodular matrix U and a $2m \times n$ echelon matrix S such that: $$U \cdot \begin{pmatrix} A \\ -B \end{pmatrix} = S.$$ This equation represents the process of solving the general dependence equation of the two participating variables (normally arrays).
3	Consider an alternate computation, $tS = b_0 - a_0$, for t (a $2m$-vector). An integer solution to this equation indicates that the equation in Step 2 also has an integer solution. Note that this system is easy to solve since S is an echelon matrix.
4	Apply the Generalized GCD Test to Step 3. This test shows that no integer solution is possible for the system in Step 3 — algorithm terminates.
5	Compute both the known and undetermined components of t.
6	Apply dependence constraints (i.e. constraints upon the dependency set by virtue of the loop's upper and lower bounds) by solving a system of $4m$ inequalities in the number of undetermined components of t variables.
7	If the system of inequalities in Step 6 has no integer solution, then there is no dependency within the constraints of the loop nest — algorithm terminates.
8	Compute direction and distance vector, and determine type of dependency (e.g. true or anti).
9	Consider loop transformations (e.g. loop permutation, loop distribution, loop reversal, loop skewing, etc.) based on preserving the semantics of the original loop nest as well as on optimizing concerns (e.g. the parallelization or vectorization of some of the loop nest).
10	Terminate Algorithm.

We have successfully applied the presented algorithm to the following problem[10]. Given a particular FORTRAN program (which incorporates data dependency inhibitations for proper vectorization), identify such dependences so that a simple restructuring would allow proper vectorization on the IBM VF 180.

4 Overview of The Unified Model

The unified model, $\mathcal{M}$, is defined as the quintuplet $\mathcal{M} = (\mathcal{A}, \mathbf{Z}, \mathbf{\Phi}, \mathbf{M}, \mathbf{\Pi})$, where $\mathcal{A}$ is the given systolic algorithm requiring implementation, $\mathbf{Z}$ is a set of partitioning strategies, $\mathbf{\Phi}$ is the set of implementational objects corresponding to $\mathbf{Z}$, $\mathbf{M}$ is the set of allowable multicomputers and $\mathbf{\Pi}$ is the set of tasks which embody the application of the various stages proposed in the unified model.

An application of the unified model consists of the execution of the tasks represented by $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5\}$. Here, Π_1 represents the process of transforming the given systolic computation into an intermediate form. Π_2 is the process of determining the partitioning strategy as well as the details of the implementational objects. Π_3 represents the process of partitioning the systolic array intermediate representation from Π_1, representing it in a source code intermediate form and evaluating the latter in terms of its expected execution time. Π_4 represents the code generation process. Finally, Π_5 represents the final stage of compiling the generated code into object code.

We now illustrate how the unified model could be applied to a typical systolic algorithm. Our discussion perhaps emphasizes an overly simplistic application of the unified model for presentation purposes. Prior to applying the model, we assume that a set of implementational objects including performance models has been constructed by carrying out task Π_2, that a particular input specification language for the given algorithm exists, and that a particular multicomputer has been appropriately modeled. The input algorithm is first transformed to a systolic array by the application of Π_1. Π_3 is next used to determine the expected most efficient implementation. One method of implementing Π_3 could be as follows. A partitioning strategy is selected (i.e. choose one of the implementational objects) and the number of partitions is set to the number of available processors less one. A processor topology graph is constructed for this particular partition. Based on the processor graph, the prototype codes associated with the partitioning strategy are combined with the systolic algorithm and an analysis is performed. The results of the analyses are used during the execution of the performance model — which returns the expected execution time for that implementation. Each of the remaining implementational objects are considered in like manner; the chosen implementation is that which has the minimum expected execution time. Finally Π_4 and Π_5 complete the application.

5 Unified and General Model Comparisons

This section presents a comparison of the two models by firstly discussing general comparisons and secondly, by considering the criteria discussed in Section 2.

5.1 General Comments

The unified model deals with the detailed implementational issues when considering the efficient execution of a given algorithm for a given machine. As such, this model reflects a comprehensive methodology to analyze the given input algorithm, to determine the most efficient implementation, and to generate code for that implementation. The unified model divorces the machine

dependent issues from independent issues by considering all dependent issues as part of a second layer while leaving all of the independent issues as part of the first layer (exactly the distinction of the implementational object and its use in the unified model). The generalized model contains more extensive processing requirements and the sources reviewed in this paper do not really consider implementation issues.

In general,the generalized model does not specify a particular class of machine architectures and as such, could be portable to varied high performance computers. However, specific implementational issues such as the mapping of parallelized loop segments to processors may require additional support (i.e. such support is not inherent in the generalized model). We note that comments in[3] indicate that loop restructuring methods (that which was referred to in Step 9 of the general model's application algorithm) are dependent on the target computer architecture; as well, specific low-level compilation issues are influenced by the restructuring methods (e.g. register pressure).

The unified model relies heavily on the properties of systolic computations while the generalized model focuses on automatic discovery of parallelism in general, that is, over a large domain of parallel computing algorithms. We note that many common problems requiring computer solutions have equivalent systolic algorithms (e.g. matrix computations, database operations and sorting. However, the application of the general model is, by its nature, more widely applicable to algorithms including those which have non-systolic solutions.

The unified model requires considerable manual effort in the modeling of implementational objects as well as the modeling of multicomputers (i.e. the application of Π_2 is a manual process). While the processing requirements of the general model are somewhat computationally severe, most if not all can indeed be automated.

Lastly, the general model is a mature model and has been employed in many research and commercial compilers (refer to[1, 3, 5] for discussion of particular compiler projects) whereas the unified model is newly proposed and as not been yet employed in a compiler. Key issues relating to such implementation of the unified model, however, have been investigated[9].

5.2 Criteria Evaluation

1. **Conversion of Sequential Code:** Neither model specifies a specific format for input programs. However, the examples given in the sources for the general model strongly indicate that standard loop nests written in standard sequential languages are acceptable; this has been the case with respect to both research and commercial compilers which incorporate aspects of the general model. The unified model places restriction upon the input allowed (i.e. must be a set of uniform recurrence relations together with inputs and upper and lower bounds on the relations)[6], however, no specific specification language is adopted. It is expected that such a specification language would be close to a standard sequential language.

2. **Diversity of Architecture:** As as been previously discussed (Section 5.1), the sources reviewed do not describe architectural details for implementations of the general model. It is expected then that each application of the general model in a compiler would be specifically tailored for a particular architecture. The unified model, on the other hand, internally deals with architectural details of target machines. It does this by modeling specific target machines and using the modeled parameters (e.g. communication latency, computation time) in a performance model which, based on other parameters, compute the expected execution time of a specific implementation of the input algorithm for the particular target machine. The architectural modeling required by the unified model results in the requirement for manual intervention. However, once the machine has been modeled, an application of the unified model is expected to be automated. Only the t800 transputer has been modeled to date[11, 9].

3. **Portability and Availability:** On the theoretical level, the general model supports better portability than does the unified model since the general model (as reviewed herein) is a machine independent model. On the practical level, however, as has been previously mentioned in Section 5.1, many of the specific restructuring techniques impact upon portability and optimization issues. The unified model recognizes this relationship between low and high level analysis and restructuring by adopting a two-layer model where issues pertaining to each level are contained in respective layers. Feedback mechanisms have also been incorporated, however, the utility of such mechanisms has not been reported on.

4. **Loop Scheduling:** Due to the constraints on the input algorithm to the unified model, the unified model deals with restricted forms of loops. In particular, the unified model considers the parallelization of sequences of operations as determined from a systolic graph representation of the input algorithm (which is derived from the data dependency graph). Consequently, the unified model considers efficient loop scheduling for restricted types of loops. The general model, however, expends much effort first in loop analysis and then loop restructuring. Consequently, the general model also considers efficient loop scheduling. More cost is associated with the procedures in the general model, however, it is also more widely applicable.

5. **Execution Model:** Several execution models are

incorporated into the unified model, in particular, the farming (master-slave) model [7, 9] The loop restructuring portion of the general model can be influenced by a particular execution model.

6. **Data Layout / Partition:** Neither model specifies the layout or partitioning of the data. Often, this is dependent on loop scheduling. In the unified model, the data layout is obtained at the time of considering sequences of computations in parallel.

7. **Synchronization:** Since the general model does not specify implementation issues, synchronization in the general model is left to a specific compiler implementation. It is expected that optimized compilers would reduce synchronization issues. In the unified model, synchronization issues are significantly involved in the granularity analysis portion of the system. That is, since fine-grain parallelism is specified by the systolic representation, reduction in synchronization as a way to increase efficiency and execution time is an important component in the unified model. However, it is an *optional* component with respect to actually compiling an input algorithm; as such, few details have been reported.

8. **Exceptions and Debugging:** Neither model specifies conditions for exceptions and debugging.

9. **Performance and Efficiency Evaluation:** Again, since the general model does not specify implementation issues, no performance or efficiency evaluation is possible. Specific compiler implementations may be able to support performance evaluation. In the unified model, performance evaluation (in terms of estimated execution time) is an integral part of the model and is used to determine which of a number of candidate implementations is ultimately to be executed.

6 Conclusion

We have conducted a comparative overview of two models for compiling algorithms for high performance computers. The general model is based on the many techniques associated with dependency analysis and loop restructuring. We have based our review on the presentation by Banerjee. The unified model is a newly proposed model which considers the issues of compilation from the viewpoint of systolic processing. It is a relatively non-mature, however, preliminary results seem to be encouraging. In terms of the comparison conducted in this paper, the unified model is indeed suited for use in a compiler for high performance computers. However, the applicability of the general model remains more general purpose.

References

[1] U. Banerjee, *Loop Transformations for Restructuring Compilers — The Foundations.* 101 Philip Drive, Assinippi Park, Norwell, MA, USA, 02061: Kluwer Academic Publishers, 1993.

[2] U. Banerjee, *Loop Parallelization.* 101 Philip Drive, Assinippi Park, Norwell, MA, USA, 02061: Kluwer Academic Publishers, 1994.

[3] M. Wolfe, *High Performance Compilers for Parallel Computing.* 390 Bridge Parkway, Redwood City, CA, 94065: Addison-Wesley Publishing Company, 1996.

[4] D.E. Hudak and SG. Abraham, *Compiling Parallel Loops for High Performance Computers — Partitioning, Data Assignment and Remapping.* 101 Philip Drive, Assinippi Park, Norwell, MA, USA, 02061: Kluwer Academic Publishers, 1993.

[5] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers.* New York, New York: ACM Press, 1991.

[6] B.J. d'Auriol and V.C. Bhavsar, "Multicomputer Implementations of Systolic Computations: A Unified Approach," *Proc. of the 10th Annual International Conference on High Performance Computers (HPCS'96)*, Ottawa, Ontario, Canada, June, 5-7, 1996, June 1996.

[7] B.J. d'Auriol and V.C. Bhavsar, "Generic Program Representation and Evaluation of Systolic Computations on Multicomputers," *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Vol. III*, Sunnyvale, California, USA, August, 9-11, 1996, H. Arabnia, (ed.), pp. 1213–1224, August 1996.

[8] B.J. d'Auriol and V.C. Bhavsar, "A Unified Approach for Implementing Systolic Computations on Distributed Memory Multicomputers," Tech. Rep. WSU-CS-95-02, Department of Computer Science and Engineering, Wright State University, Dayton, Ohio 45435, USA, December 1995.

[9] B.J. d'Auriol, *A Unified Model for Compiling Systolic Computations for Distributed Memory Multicomputers.* PhD thesis, Faculty of Computer Science, University of New Brunswick, Fredericton, N.B. Canada, June 1995.

[10] B.J. d'Auriol, "Application of Dependency Analysis to a Vectorization Problem on an IBM3090/VF 180." Working Notes, 1993.

[11] B.J. d'Auriol and V.C. Bhavsar, "COMMAN — A Communication Analyzer for Occam 2," *Transputer Communications*, Vol. 3, April 1996. in press.