

# The Role of Language in the Unified Model for Compiling Systolic Computations for Multicomputers

Brian J. d'Auriol<sup>1</sup>

*Department of Computer Science  
The University of Manitoba  
Winnipeg, Manitoba, Canada, R3T 2N2*

Donald Kent Bierley

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

## Abstract

The language requirements of the unified model for compiling systolic algorithms on multicomputers is discussed. Furthermore, an analysis of existing work to determine their suitability for incorporation into the unified model is conducted. On the basis of this analysis, particular components of existing work are identified as potential components for inclusion or modification into the unified model. An overview of the unified model is provided for convenience of the reader.

*Keywords:* Language Models, Systolic Computing, Parallelizing Compilers.

## 1 Introduction

A new model for compiling systolic computations for multicomputers was recently proposed in 1995 by d'Auriol and Bhavsar [1, 2, 3, 4] as a model suitable for a compiler to analyze a given systolic algorithm as input, consider various high level language program representations of the input algorithm together with efficiency issues and generate the source code corresponding to the best choice implementation. Central to an implementation of the unified model is the role that systolic language plays in its various components. For example, some specification language will need to be adopted for representation of the input algorithm. Furthermore, as part of the internal processes of the tasks within the unified model, a systolic array representation is also necessary. Lastly, code generation (using language template libraries) is necessary in two contexts: (a) to be able to estimate execution time behavior, and (b) to be able to generate the code so that a native compiler can be used to compile the resulting implementation into executable code.

We note that much work has been reported on soft-systolic computing systems (see for example [5, 6, 7, 8, 9, 10, 11]). Much of the work cited herein has been reviewed in context of the unified model [4], however, what is very

interesting to note in the context of this paper is that many of these systems incorporate a language or code model specific to systolic processing.

The scope of this paper is to firstly examine the language processing requirements of the unified model and secondly, to discuss elements from existing and past work which may be incorporated into a language model suitable for use in the unified model. A three-criteria approach is used to establish the utility of existing methods and components. The criteria consists of the following in the order given.

1. Can existing methods be incorporated without modification?
2. Are existing methods suitable for modification so that they can be incorporated easily?
3. What elements from existing methods can be incorporated into a specific language model which can then be used in the unified model?

This paper will consider some of the existing work as indicated above in the context of the above three criteria. In this paper, we will show that the language processing requirements of the unified model are more complex than that which exist in existing systems, and consequently, a selection of existing methods together with new proposed methods will need to be made.

A brief overview of the unified model is repeated in the next section for convenience of the reader. Section 3 discusses the language requirements of the unified model while Section 4 discusses the various elements of existing work as relating to the unified model. Concluding remarks are given in Section 5.

## 2 Overview of Unified Model

The unified model,  $\mathcal{M}$ , is defined as the quintuplet  $\mathcal{M} = (\mathcal{A}, \mathbf{Z}, \Phi, \mathbf{M}, \mathbf{\Pi})$ , where  $\mathcal{A}$  is the given systolic algorithm requiring implementation,  $\mathbf{Z}$  is a set of partitioning strategies,  $\Phi$  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.

---

<sup>1</sup>This work was done while the author was at Dept. of Comp. Sci., Wright State University, Dayton, Ohio, USA, 45435.

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,  $\Pi_1$  represents the process of transforming the given systolic computation into an intermediate form.  $\Pi_2$  is the process of determining the partitioning strategy as well as the details of the implementational objects.  $\Pi_3$  represents the process of partitioning the systolic array intermediate representation from  $\Pi_1$ , representing it in a source code intermediate form and evaluating the latter in terms of its expected execution time.  $\Pi_4$  represents the code generation process. Finally,  $\Pi_5$  represents the final stage of compiling the generated code into object code.

Formally, this set of procedures is summarized as follows. There exists a systolic array so that for all partitioning strategies defined by  $\mathbf{Z}$  there exists an associated implementational object such that

$$\Pi_1(\mathcal{A}) \mapsto s \quad (1)$$

$$\Pi_2(\zeta, s) \mapsto \Phi_\zeta \quad (2)$$

Given Eq. (1) and Eq. (2), the evaluation of the multiple candidate implementations and subsequent generation and compilation of the final high level language program representation is given by:

$$\Pi_3(\Phi, \mathbf{Z}, s, \mathbf{M}) \mapsto \mathcal{I}^r \quad (3)$$

$$\Pi_4(\mathcal{I}^r) \mapsto \mathcal{I} \quad (4)$$

$$\Pi_5(\mathcal{I}) \mapsto \mathcal{I}^c \quad (5)$$

where,  $s = (\hat{G}, I, F)$  (i.e. a systolic array is composed of a graph of vertices and directed edges, a data sequencing of inputs distributed through time and space and a set of functions mapped to each vertex in the graph),  $\zeta \in \mathbf{Z}$ ,  $\mathcal{I}^r$  is the selected implementation,  $\mathcal{I}$  is the final high level language program representation of the implementation and  $\mathcal{I}^c$  is the compiled version of  $\mathcal{I}$ .

An example of a  $4 \times 4$  array structure is given in Figure 1(a) where a circle represents a distinct sub-computation and a directed edge represents the dependence between the sub-computations. Since several candidate partitions  $\zeta$  are input to  $\Pi_2$ , the execution of  $\Pi_2$  leads to the corresponding implementational objects denoted by  $\Phi_\zeta$ .

An implementational object is the three-tuple  $\Phi = (\Xi, \mathcal{P}, \Upsilon)$ , where  $\Xi$  is a set of *prototype codes* together with a set  $\mathcal{P}$  of *implementation parameters* and a *performance model*  $\Upsilon$ . The prototype codes are pre-defined source code templates which essentially are rules for generating source code in a language for which a native compiler is already available. The set of implementation parameters describes the *exact* characteristics of the implementation and is used in the associated performance model. The performance model evaluates the expected execution time for the given systolic algorithm. The performance model is parameterized so that it can provide a good estimate of the execution time while maintaining general applicability.

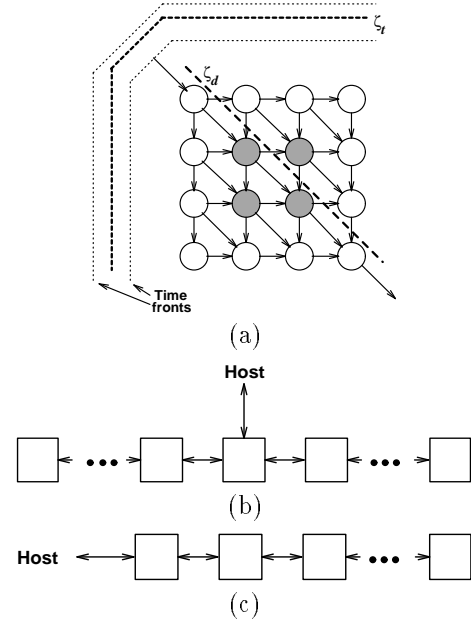


Figure 1: Systolic array representation, partitioning and resultant processor networks: (a) an example of  $s$  and two partitions,  $\zeta_d$  and  $\zeta_t$ , (b) processor network due to  $\zeta_d$ , and (c) processor network due to  $\zeta_t$ .

The application of  $\Pi_2$  represents a two-step process. In the first step, candidate partitions are applied to the systolic array generating a set of processor topologies required for each of the candidate partitions. For example, in Figure 1(a), two representative partitions corresponding to the Diagonal Partition,  $\zeta_d$ , and the Temporal Partition,  $\zeta_t$ , are illustrated by heavy dashed lines. From our previous investigations [3, 4], the processor topology associated with Diagonal Partitioning is shown in Figure 1(b), while that associated with Temporal Partitioning is shown in Figure 1(c). We also note that a master-slave farming execution model applies to the implementation executed on the topology shown in Figure 1(c) whereas, a static allocation method applies to the implementation executed on the topology shown in Figure 1(b). We lastly note that many other partitions are possible. In the second step, the three components of the implementational object are defined.

### 3 Language Requirements

The language requirements of the unified model are subdivided into the following categories.

1. **Specification Language:** Algorithm specification requirements can be represented in a sequential language which allows for the specification of recurrence relations, in a parallel language which allows for the spatial-temporal distribution of the input algorithm, or as a data dependency graph with associated functions mapped to each point in the graph. A specification language must be able to represent the systolic

algorithm in sufficient detail so as to apply Eq. (1). As we will see in the next section, most of the existing work has been in this area.

2. **Systolic Array Language:** All requirements of an algorithm as specified by the Specification Language must be represented in a systolic representation. The components of this representation are given by  $(\hat{G}, I, F)$  as derived by Eq. (1). As before, much work has been reported in this area.
3. **Template Specification:** This form is distinct in that it represents the structure of the implementation of the final source code version. That is, it describes the machine and native compiler environments necessary so that a program representation can be constructed. It incorporates the following five sub-categories.
  - (a) Configuration: The allocation of processes to processors and related allocation of communication channels, together with appropriate start-up initialization routines.
  - (b) Routers: The processes which are fully responsible for handling all communication to and from a computation process across a network of processors.
  - (c) Computation: The processes which perform the actual computation.
  - (d) Communication: The auxiliary code representation which provides details on the communication specific structure (e.g. channel datatyping).
  - (e) Input/Output Control: The code representation which is responsible for controlling the runtime processing requirements.

The construction of groups of templates, as in, for example, a template library, is part of Eq. (2).

4. **Implementation Specification:** This form is derived from the combination of the Template Specification and the Systolic Specification. That is to say, the details of the systolic algorithm are combined with the details of the final source code implementation program structure so as to generate a final source code implementation specific to the given input algorithm. The derivation of this specification is done as part of Eq. (3).
5. **Source Code Language:** The representation of the generated source code program in a standard computer language for which a native compiler already exists (e.g. FORTRAN, C or occam). This form is expected to be similar to that of the Implementation Specification. This form is generated by Eq. (4).

## 4 Previous Work Relating to the Language Requirements

A graphical interface system has been used by the Poker/Hearts system [11] to allow the user to specify the systolic algorithm code in a sequential language. This, in effect, is similar to the Systolic Array Language specification in the unified model although, in the unified model, the algorithm is specified in a non-systolic form which is then automatically translated to the systolic form.

The functionality of the interface allows the specification for the design of the systolic array. For example, the programmer uses a drawing editor to place communication links between the processors. Although this also refers to the systolic array design, in terms of the unified model, this is similar to some of the requirements in the Template Specification, particularly the configuration section.

Another system which employed a graphical user interface in the design of a systolic array is the SDEF system [6]. Both Hearts/Poker and SDEF provide for the generation of source code which can then be executed (or compiled by a native compiler).

There are numerous computer languages developed or used to also represent the systolic arrays. These include C-stolic [7], occam [12], FP [13], and W2 [14]. All of these examples are used only in the specification of the algorithm or, when such is specified as a systolic array, in a systolic array specification.

The Generalized Systolic Computation (GSC) [8] model extends the ideas of the previously discussed systems in that the GSC model incorporates a Specification Language combined with the Systolic Array Language specification. It also has elements of the Implementation Specification. However, due to the combination of the various specifications as well as other limiting factors, the GSC model may not be suitable with respect to the processing requirements of the unified model.

The Systolizing Compilation Scheme (SCS) proposed by Lengauer, Barnett and Hudson [10] is a two-step compilation scheme for systolic algorithms. This scheme consists of systolic array design followed by the generation of code for a target language (occam and W2). The input for this scheme is essentially a Specification Language. An intermediate form termed 'repeaters' has been introduced in their work. A 'repeater' is a triplet which expresses characteristics of a systolic array in a machine independent form. As such, they may be considered as a variation of the Systolic Array Language specification.

The preceding discussion of the categories of existing work with respect to the unified model is summarized in Figure 2. Here, the three categories of 'Algorithm Specification', 'Intermediate Form' and 'Code Generation' indicate that the categorized systems provide primarily algorithmic specification, provide primarily some form of intermediate specification and provide for the generation of code respectively. We note that only the system by

Lengauer et al. provides for all of the functions which the unified model requires. Further, we note the interesting properties of the graphical user interface provided by the Hearts/Poker system and the correspondence to some of the requirements of the unified model.

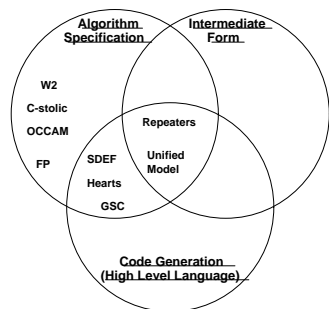


Figure 2: Classification of Existing Work

In terms of the unified model’s language requirements, much of the work surveyed in this section relates to the algorithm and systolic specification. Virtually no work has been observed in this context in terms of template specification while the Implementation Specification has also received little attention. Several of the systems (SDEF, GSC and SCS) consider the generation of source code.

We note that the SCS system more closely matches the requirements of the unified model than any of the other systems reviewed.

## 5 Conclusion

We have proposed a five category breakdown on the language requirements inherent in the unified model for compiling systolic computations for multicomputers. These categories essentially represent the various phases that an input algorithm to a compiler based on this model would be represented in during the processing associated with such a compiler. Further, we have reviewed past work in the area of implementing systolic computations and have noted that much work has been reported on the more introductory aspects of the unified model’s requirements. Some work has been reported on the remaining requirements. However, no evidence has been observed on work relating to the unification of language roles across the requirements of the unified model — although some unification is noted across *some* of the requirements of the unified model.

## References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] G.M. Megson, *An Introduction to Systolic Algorithm Design*. New York, USA: Oxford University Press, 1992.
- [6] B.R. Engstrom and P.R. Cappello, “The SDEF Programming System,” *Journal of Parallel and Distributed Computing*, Vol. 7, pp. 201–231, 1989.
- [7] D.Lavenier, F.Raimbault and P.Frison, “I/O and Computation Overlap on SIMD Systolic Arrays,” *Journal of VLSI Signal Processing*, Vol. 9, pp. 153–165, April 1995.
- [8] M.D. Rice and S.B. Seidman, “A Multiprocessor Testbed for Generalized Systolic Computation,” *Proceedings of Transputing ’91*, 1991, P. W. et al., (ed.), IOS Press, Amsterdam, pp. 281–295, 1991.
- [9] P.M. Samwell, “Experience with Occam for Simulating Systolic and Wavefront Arrays,” *Software Engineering Journal*, Vol. 1, pp. 196–204, Sept. 1986.
- [10] C. Lengauer, M. Barnett and D.G. Hudson, “Towards Systolizing Compilation,” *Distributed Computing*, Vol. 5, pp. 7–24, . 1991.
- [11] L. Snyder, “A Dialect of the Poker Programming Environment Specialized for Systolic Computation,” *Proc. International Workshop on Systolic Arrays*, University of Oxford, July 1986, July 1986.
- [12] Inmos Limited, Prentice-Hall Ltd., Hertfordshire, UK, *Occam 2 Reference Manual*, Inmos Document No. 72 occ 45 01 ed., 1988.
- [13] Y-C Lin, “Translating from FP to Occam for Systolic Algorithms,” *Microprocessors and Microsystems*, Vol. 15, pp. 435–444, Oct. 1991.
- [14] P. Tseng, *A Systolic Array Parallelizing Compiler*. 101 Philip Drive, Assinippi Park, Norwell, Mass. 02061, USA: Kluwer Academic Publishers, 1990.