

SYSTOLIC AND WAVEFRONT ARRAY ALGORITHMS ON DISTRIBUTED MEMORY, MULTIPROCESSOR COMPUTERS

Brian J. d'Auriol and Virendra C. Bhavsar [†]
Parallel/Distributed Processing Group
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3, Canada
Email: k3mi@unb.ca, bhavsar@unb.ca

Abstract

The issues of transforming systolic and wavefront algorithms to specific machine-dependent implementations are examined. A brief review of current research in this area indicates that there remain many open questions. A general purpose schema for such implementations which incorporates consistency, efficient use of resources, automatability and wide applicability is proposed. Such a schema is aimed at incorporating it into a compiler for these algorithms. Further, this schema is evaluated by considering a specific systolic algorithm (Weighted Levenshtein Distance) and its implementations. Comments are made about the suitability of the implementations and are accompanied by specific results from actual implementations. An example of actual implementations is given. This example demonstrates some of the potential hazards inherent in any implementation of systolic or wavefront arrays.

Introduction

Systolic Computations are usually specified in terms of some algorithmic representation. Affine recurrence relations and directed acyclic graphs (DAGs) are also often used. An important observation for such specifications is that often they are single instruction, multiple data (SIMD) algorithms. Although implementation of these algorithms on multiple instruction, multiple data (MIMD) computers is not straightforward, efficient implementations can be derived. Even in some general sense, programming systolic arrays can be very difficult, as summed up by Hwang in Section 2.4¹.

Wavefront arrays are similar to systolic arrays. However, an important difference is that while communication in systolic computations occurs synchronously, it occurs asynchronously in wavefront arrays. Such asynchronous communication is more amenable for MIMD implementations. As this difference has not had any tangible effects on our results to date and furthermore, since Hwang in Section 10.5¹ describes loop transformation schemes that can construct such wavefront computations out of systolic computations, we will hereafter refer to systolic computations as also including wavefront computations .

In this paper we examine the issues of transforming systolic algorithms to implementations on distributed memory, multiprocessor computers. In Section 2, we discuss the general issues involved in such transformations and how current research has approached solutions. Section 3 discusses various issues relating to both the algorithmic and implementational aspects of the transformation. Specific performance models for several proposed implementations are given. In Section 4, we propose a strategy to transform a given algorithm to an efficient implementation. Such a strategy is aimed at incorporating it into a compiler. Section 5 examines a specific systolic algorithm, namely the Weighted Levenshtein Distance Computation, and its corresponding implementations. Conclusions are given in the final Section.

[†]Member, Super*Can

Previous Work

Systolic arrays, introduced by Kung² in the late 1970's, have been widely applied to VLSI implementations of numerous algorithms. Its parallel processing capabilities combined with a simple and scalable architecture have generated considerable interest. Indeed, many systolic algorithms now exist, including algorithms for matrix-matrix multiplication, L-U decomposition, string pattern matching and digital signal processing.

As parallelization tools and computers became available, interest in software programmed systolic arrays developed. An example of a software implementation of is given by Samwell in 1986³ where systolic algorithms for the matrix-vector and digital filter computations are represented by Occam processes. In this case, the implementation described directly represents the corresponding systolic computation.

Software implementation of systolic arrays continues to be of interest, as exemplified in the 1990 work by Megson⁴ for model reduction and by our own work in 1991 for Reconfigurable Learning Machines using the Weighted Levenshtein Distance computation⁵. Interest has seemingly shifted somewhat from the 'how it can be done' to the more dogmatic 'does it do it any better' philosophy. In the 1990 work by Megson, for example, several implementations are derived and their corresponding performances are evaluated. In our previous work in 1991 and 1992^{5, 6, 7}, we have examined various implementations for the Weighted Levenshtein Distance computation and have evaluated their corresponding performances.

Work on related aspects of systolic arrays is also noted. For example, the effects of message-based communication are described in the 1991 work by Ramanujam and Sadayappan⁸, while generation of target code for parallel, distributed-memory machines is described in the 1991 work by Li and Chen⁹.

However, work on a unified schema for the software implementation of systolic algorithms in general is of recent interest. The question is now 'how best to do it'. Several groups are now working in this area. For example, Tseng¹⁰ describes work on a systolic array parallelizing compiler. Gupta and Banerjee¹¹ describe an approach to the problem of automatic data partitioning. Their aim is to provide a compiler that will identify constraints on the data distribution and generate an executable. Rice and Seidman¹² describe their efforts to unify elements such as language, verification, computational model and implementation issues in a wavefront context. This recent emphasis on automating the implementation of systolic algorithms addresses a current and significant problem with such implementations, namely, the inability to easily exploit the potential parallelism inherent in these algorithms. This issue remains outstanding.

Our current research efforts can best be described as seeking a unified and general purpose schema for automatic generation of efficient code for systolic/wavefront arrays on MIMD machines addressing the solution of this question. Our previous efforts^{5, 6, 7} have concentrated on multiple implementations of a specific wavefront algorithm in context of our general schema. Issues such as language expressibility (using Occam), decomposition, granularity and network topology have been investigated.

Algorithms to Implementations

This section considers the issues involved in transforming a given systolic algorithm into one or more possible implementations. Two primary concerns which have been identified are: the correctness of the transformation and the determination of the optimum implementation. The correctness criteria is based on the complete analysis of all data and control dependencies existing in the given algorithm. Such analyses are now common for compilers for vector computers. The determination of which of the possible implementations is optimum is based upon the aforementioned analysis as well as the evaluation of machine dependent characteristics. Both of these issues are discussed in detail below.

ALGORITHMIC ISSUES

We define a systolic/wavefront algorithm as any algorithm which has the following properties:

- **Recurrence:** The algorithm is described by a replicated or recurrence set of instructions arranged in a fixed lattice structure. Such a structure is often termed a *systolic array* and each set of instructions is termed a *cell*.
- **Regular and Rhythmic Communication:** The input data and/or any other intermediate data are passed through cells in a regular and rhythmic fashion.

This definition is also consistent with common usage.

Additional properties to those in the definition can usually be found. These include (but may not be limited to) the following:

- **Communication Patterns:** Data is passed from cell to cell according to specific (usually linear) functions. This flow of information fully describes the inter-dependence of cells and in particular, represents both data and control inter-dependencies.
- **Cell Functions:** More than one set of instructions may be defined over specific regions in the systolic array. Thus groups of cells may perform different functions within the same systolic array. Moreover, the systolic array itself may allow for more than one type of computation to occur. In this case, each cell has a predefined range of functions which are appropriately executed.
- **Topological Differences:** Groups of cells or regions of the systolic array may require different communication and computation topologies. Thus, there is some crossover between this and the previous category. An example of this is the borders of any systolic array.

We will assume that a given systolic algorithm is specified in some consistent manner and furthermore, we can analyze and extract properties which fully characterize the algorithm. We loosely define this collection of properties as the *Property Set, P_A* , corresponding to the given algorithm, A .

IMPLEMENTATION ISSUES

We now consider possible implementations of a given systolic algorithm. We define an implementation to be a program which generates the identical result as the corresponding algorithm. It is easily established that multiple implementations for an algorithm are possible. Consider a machine with a number of processors which are interconnected by some complex (and possibly dynamic) interconnection network. The algorithm can usually be decomposed so as to execute on a group of processors with varying interconnections. The different interconnections impact on the flow of data, and thus impact on the structure of the implementation. Moreover, it is usually the case that a particular implementation can be scaled so as to execute on one or more processors.

Although any given algorithm may be transformed into one or more implementations, there may be some types of implementations to which the algorithm cannot be transformed (see Figure 1). The first systolic algorithm can be transformed to the first two types of implementations while the second algorithm can be transformed to one of the previously used implementations and a new one. In general, the i^{th} algorithm can be transformed into one or several of a set of m types of implementations.

The transformation from algorithm to implementation must preserve the property set of the algorithm. Thus the issue of implementation correctness is not a concern. However, the determination of an optimal implementation is not as straightforward. An optimum implementation depends not only on the property set of the algorithm but also on machine dependent characteristics.

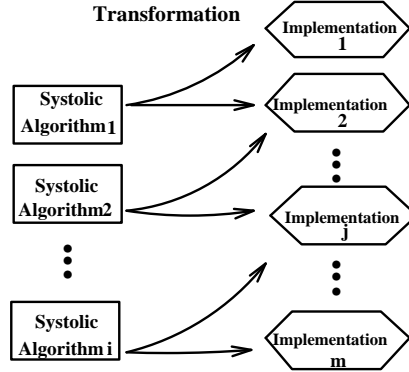


Fig. 1: Possible Transformations of Systolic Algorithms to Types of Implementations

Machine characteristics can be grouped as a set. Such factors as: computational speed, communication speed, number of processors, interconnection of processors, and granularity, impact on the efficiency of a particular implementation. It is expected then, that tradeoff considerations exist. For instance, it may be desirable to minimize the number of processors utilized.

Each implementation is predefined and has associated with it a cost function. Parameters taken from both the property and machine sets are used to evaluate a specific implementation. Based on this evaluation, a suitable implementation is chosen and the corresponding code is generated.

We have, at present, defined several specific implementations which are general enough to represent systolic algorithms; others are under development. These implementations are aimed at various levels of granularity and in particular, exploit parallelism within a single task which consists of one execution of a systolic algorithm as well as between multiple tasks which represent multiple executions of the same systolic algorithm for different sets of input data. The latter may also consist of the same structure, but not necessarily the same function. These implementations are discussed below.

We consider the following implementation possibilities:

- **Systolic:** Directly represent the structure of the systolic algorithm; each cell is mapped to a single software process and all communication occurs as explicitly given in the algorithm. This implementation attempts to allow parallelism at both levels.

In this case, the expected total execution time for a desired task is composed of the computation, T_{cp} ; the communication, T_{cm} ; the inter-processor communication, T_{cb} and the process switching component, T_{ps} . An additional factor, T_o , is added to take care of any additional time, especially that which is due to certain anomalies noted in our earlier work. The total execution time is given as: $T_{tt} = T_{cp} + T_{cm} + T_{cb} + T_{ps} + T_o$ where T_{cp} , T_{cm} and T_{cb} depend upon both algorithmic and machine properties while T_{ps} depends upon machine properties only.

- **Ring:** Groups of cells are collapsed into single and distinct processes which are then distributed across a pipeline of processors. The communication pattern reduces to that of a standard linear pipeline. Although this implementation is aimed at multiple tasks, an alternative method can address the issue of parallelism within a task.

The expected total time can be expressed as: $T_{tt} = T_{cp} + T_{cm}$, where T_{tt} , T_{cp} and T_{cm} are defined as before. Both of the components also depend upon algorithmic and machine properties.

- **Farming:** Some portion of the task is farmed out to a pool of processors. Two types of farming, each aimed at the two levels of parallelism, can be defined. In the first type, each instance of the

computation (task) is farmed out. thus, inherent parallelism within a single computation (task) is ignored. The expected total time in this case can be expressed in much the same way as that for Pipeline, $T_{tt} = T_{cp} + T_{cm}$. However, the communication component is modeled in a different way. As with Pipeline, both components depend upon algorithmic and machine properties. In the second type, however, groups of cells are collapsed into single and distinct processes (similar to pipeline) which are then farmed out. Inherent parallelism within a single computation is used.

- Hybrid: Combinations of the above implementations allowing for inter- and intra-parallelism can be defined: Farming/Pipeline, Farming/Systolic, and Farming/Farming (i.e. combining the two definitions of Farming above)

Proposed Strategy

Based upon our investigations, we propose a general purpose strategy for implementing systolic algorithms on distributed memory, multiprocessor computers. Such a strategy incorporates consistency, efficient use of resources, automatability and wide applicability (see Figure 2).

A systolic algorithm can be specified in one of a number of predefined forms (e.g. recurrence relation or directed acyclic graph). This specification of the algorithm forms the input to the compiler. The first stage of the compiler is to analyze the algorithmic specification and extract all properties of the algorithm. The second stage examines the property and machine sets (the latter of which can be predefined) and extracts relevant parameters. The next stage evaluates each of the possible implementations by evaluating the associated cost functions. Once a particular implementation has been chosen, the last stage generates the required code. In particular, code generation represents the algorithm in some high-level language which must then be compiled for execution purposes.

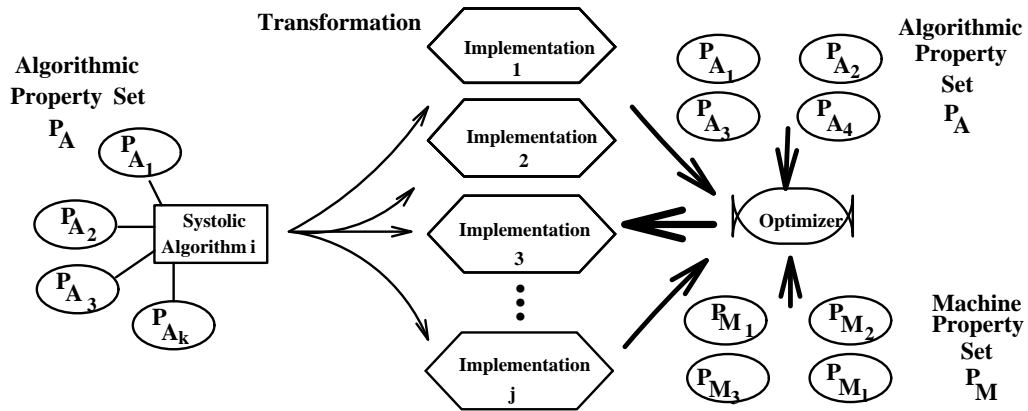


Fig. 2: General Compiler Scheme

Case Study

We now turn our attention to considering the transformation and subsequent implementation of the Weighted Levenshtein Distance (WLD) Algorithm. This computation exhibits many of the properties of systolic arrays that we wish to investigate. Moreover, the WLD computation is useful in various application areas, including error correction and pattern recognition. All of our implementations at present are targeted for Inmos T800/805 based multi-transputer systems.

THE WEIGHTED LEVENSHTTEIN DISTANCE COMPUTATION

The Weighted Levenshtein Distance (WLD) ⁵ computation is a scalar measurement of the work involved in transforming one character string, A , into another, B . We assume that the strings

are composed of single, distinct symbols, without need to attach significance to what these strings represent. Each symbol has associated with it a cost factor representing both its insertion and deletion cost.

The WLD computation can be represented by one of two related algorithms: an array based sequential algorithm or a systolic algorithm exhibiting wavefront features. Its systolic nature can be represented as a two-dimensional, hexagonal array. In addition, two distinct levels of parallelism can be defined: a fine grain parallelism due to the inherent structure of the systolic algorithm and a medium grain parallelism due to simultaneous distance computations (if required).

The algorithm is specified as follows. We denote the length of strings A and B as N_A and N_B respectively; the calculation matrix, M has dimensions $(N_B + 1) \times (N_A + 1)$. For convenience, we consider the first element of both strings A and B to be a_1 and b_1 respectively, whereas the first element in the matrix is $m_{0,0}$. The sequential algorithm is based on these definitions and is given in Figure 3. The functions $ins(x)$ and $del(x)$ represent the insertion and deletion cost of the symbol x ; $sub(x, y)$ represents the substitution cost of replacing x with y . The substitution cost is mentioned for generality and is not used in our work. A small example is provided in Figure 4.

1. $m_{0,0} \leftarrow 0$ (by definition)
2. Compute top row (insertion cost)
 $m_{0,j} \leftarrow m_{0,j-1} + ins(b_j); 1 \leq j \leq N_B$
3. Compute left column (deletion cost)
 $m_{i,0} \leftarrow m_{i-1,0} + del(a_i); 1 \leq i \leq N_A$
4. Compute all further elements
 if $a_i = b_j; 1 \leq i \leq N_A; 1 \leq j \leq N_B$
 $m_{i,j} = m_{i-1,j-1}$
 else
 $m_{i,j} = \min[m_{i-1,j} + del(a_i),$
 $m_{i,j-1} + ins(b_j),$
 $m_{i-1,j-1} + sub(a_i, b_j)]$
5. Return m_{N_B, N_A} .

	λ	a	b	b	c
λ	0	.1	.3	.5	.8
a	.1	0	.2	.4	.7
b	.3	.2	0	.2	.5
a	.4	.3	.1	.3	.6

Fig. 4: WLD Computation with insertion/deletion costs as follows: a=.1, b=.2, and c=.3

Fig. 3: WLD Sequential Algorithm

ANALYSIS AND IMPLEMENTATIONS

Based on the given algorithm, we offer a brief analysis of possible implementations in terms of algorithmic and machine properties.

The most direct transformation from WLD is the Systolic implementation. Each systolic cell is implemented as a distinct process and the communication patterns between the cells are preserved. When the number of processes is much greater than the number of processors available, groups of cell processes must be allocated to each of the processors. Load balancing and network topology factors influence the allocation method. Processor characteristics such as processor switching time, differences in latency between communication within or external to a processor and external communication overloading due to multiple processes multiplexed over a single link also have a further impact on the implementation. Multiplexing, when done, introduces additional factors, for example, asynchronous communication may need to be explicitly represented. Our results have indicated that communication is excessively costly, especially when combined with the multiplexing issue.

The Ring implementation (referred to as pipeline in our earlier work) seeks to provide a higher degree of granularity than that defined in the Systolic implementation. The interpretation is that the WLD computation consists of the combination of subsets of smaller WLD-like computations. Each subset can be implemented as a single process; all computations within a subset can be real-

ized by the application of the appropriate sections of the sequential algorithm. Thus the i^{th} process depends on the output of the $(i-1)^{\text{th}}$ process. Processes are mapped onto a ring of processors on a one-to-one basis; the computation is initiated on the first processor; the result is available on the last. Since it is a ring, the master processor has access to both the initiation and result. An alternative is to define appropriate subsets for allocation onto a mesh topology. Our results have indicated that the ring implementation provides best speedup for multiple computations. Generated code is also compact. It is indicative that modifying the implementation's structure so as to allow parallelism within a single computation (task) would also lead to increased computational speed.

Farming (medium grain parallelism) ignores the parallelism within a single WLD computation and instead, distributes multiple computations. Thus each computation process is mapped to a processor on a one-to-one basis (many-to-one is also possible with some small modifications). The topology of the farming network dramatically impacts upon the number of effective processors. Our results for this implementation have been very positive, providing good speedup.

Farming (fine grain parallelism) uses the parallelism within a single computation and is similar to Pipeline in concept and Farming (medium grain) in structure. However, due to the *min* function used in the algorithm, a non-deterministic influence is introduced. This inhibits the potential of this implementation.

Various Hybrid implementations are also possible. Essentially, various combinations of the above factors can be fine-tuned so as to provide a 'better' implementation.

RELATED PROBLEMS

As previously described, there are two main procedures to consider: the transformation and the optimization. Errors may occur whenever incomplete information is present and in particular, where the property set of the algorithm is not preserved. This section describes an example of such a problem.

The problem was often observed when two computations (tasks) were initiated when employing multiplexing routers using the Systolic Implementation. Essentially, the router would sometimes attempt to forward a message from the *second* task before all possible messages from the *first* task had completed. Since waiting processes are blocked on a read, the router would itself become blocked on a send. In this case, no further communication can occur and the program hangs up.

The cause of this problem is that a message corresponding to the second computation was multiplexed before all messages corresponding to the first computation were completed. This clearly violates the asynchronous property of the algorithmic property set. An explicit enforcement is necessary to correct this violation. When such an enforcement (implemented as a round-robin multiplexor) was included into the implementation, this problem disappeared.

Conclusion

Since its proposal in 1978, systolic computations have become very popular for many applications. Such algorithms, initially aimed at VLSI, have recently (past decade) been considered as a viable alternative for software implementations. However, efficient implementations of these algorithms on general-purpose MIMD type computers remains an open question. Indeed, it is the subject for vigorous research efforts by several groups, ourselves included.

We have proposed a general purpose schema for the efficient implementation of systolic and wavefront algorithms on MIMD machines. Objectives of this schema include: consistency, efficient use of resources, automatability and wide applicability. These features are to be incorporated into a compiler. As part of earlier work, we have defined a number of possible implementations and

have extensively researched several of these. We have now reviewed these implementations in light of our proposed schema.

Acknowledgements

The first author acknowledges the support from NSERC in funding for PhD work at the University of New Brunswick. This research is also partially supported by the NSERC grant, OGP0089, held by the second author.

References

- [1] K. Hwang, *Advanced Computer Architecture, Parallelism, Scalability, Programmability*. McGraw-Hill, Inc., 1993.
- [2] H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI)," *Sparse Matrix Proceedings*, I. Duff and G.W. Stewart (Eds.), SIAM, pp. 256-282, 1979.
- [3] P.M. Samwell, "Experience with Occam for Simulating Systolic and Wavefront Arrays," *Software Engineering Journal*, Vol. 1, pp. 196-204, Sept. 1986.
- [4] G.M. Megson, "Transputer Implementation of Systolic Arrays for Model Reduction," *IEE Proceedings*, Vol. 137, pp. 343-352, Sept. 1990.
- [5] B.J. d'Auriol, V.C. Bhavsar, L. Goldfarb, "Systolic Array Implementations for Reconfigurable Learning Machines on Transputers," *Proc. of Supercomputing Symposium '91*, Fredericton, N.B., Canada, June, 3-5, 1991, V.C. Bhavsar and U.G. Gujar, (eds.), University of New Brunswick Press, Fredericton, N.B., pp. 105-119, June 1991.
- [6] B.J. d'Auriol, V.C. Bhavsar, L. Goldfarb, "Multi-Transputer Implementations of the Metric Approach to Pattern Recognition Using Weighted Levenshtein Distance," *Applications of Transputer 3: Volume II*, August 1991, T.S. Durrani et al., (eds.), IOS Press, Amsterdam, pp. 388-393, August 1991.
- [7] B.J. d'Auriol and V.C. Bhavsar, "Evaluation of the Multi-Transputer Implementations of the Weighted Levenshtein Distance Computation," *Proc. of International Conference on Parallel Computing and Transputer Applications '92 (PACTA '92)*, Barcelona, Spain, Sept., 21-25 Sept. Published as 'Parallel Computing and Transputer Applications', Part II, 1992, M. Valero et al., (eds.), IOS Press, Amsterdam, pp. 879-888, Sept., Sept. 1992.
- [8] J. Ramanujam and P. Sadayappan, "Compile-Time Techniques for Data Distribution in Distributed Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, pp. 472-482, Oct. 1991.
- [9] J. Li and M. Chen, "Compiling Communication-Efficient Programs for Massively Parallel Machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, pp. 361-376, July 1991.
- [10] P. Tseng, *A Systolic Array Parallelizing Compiler*. 101 Philip Drive, Assinippi Park, Norwell, Mass. 02061, USA: Kluwer Academic Publishers, 1990.
- [11] M. Gupta and P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, pp. 179-193, March 1992.
- [12] M.D. Rice and S.B. Seidman, "A Methodology for Generalized Wavefront Computation," *Second International Workshop on Array Structures, ATABLE-92*, Montreal, Quebec, Canada, June-July, 29-1, 1992, June 1992.