Multifolding: An Efficient Mapping of Parallel Algorithms onto Array Processors

Walter Cunto and Jorge H. Gonçalves R.

IBM Scientific Center, P.O. Box 64778, Caracas 1061A, Venezuela.

walter@caracas1.vnet.ibm.com, goncalve@caracas1.vnet.ibm.com

Abstract

Multifolding is a new practical heuristic method for mapping parallel algorithms onto array processor architectures. It considers symmetries present in the regular mesh structure of array processors and design restrictions. The method starts from a straightforward implementation that maps processes onto an array of an unbounded number of processing units. From this, Multifolding attempts to configure more efficient implementations with the same execution time and fewer processing units. Finally, configurations are iteratively derived while the method trades off number of processing units *vs* execution time in order to meet design constraints. We develop some examples to show the practical use of Multifolding.

1 Introduction

Traditionally, parallel wavefront algorithms have been implemented on array processor architectures with a redundant number of processing units [1, 2, 3, 4, 5]. Therefore, methods that reduce this redundancy without substantially degrading the performance of the implementations are needed.

> **Problem Statement:** Given a wavefront algorithm defined as a set of parallel processes subject to a precedence relationship of regular shape, find a less redundant and time efficient mapping of the processes onto of an array processor system. Implementations with array processor systems are constrained by the maximum number of communication links and memory size of each processing unit, point-to-point and local interprocessor communication that conform a topology of regular shape (for example, an orthogonal grid), and a bounded array size.

The problem stated above is a special case of the scheduling problem [6]. Scheduling a set of identical processes subject to precedence relations onto a bounded set of homogeneous processors is known to be in the family of the NP-Complete problems [6]. This problem is more difficult to solve if processes are not identical and further restrictions must be considered. Since efficient computational methods for these type of problems are unlikely to exist, heuristic methods are useful for practical and specific purposes.

Multifolding is a new heuristic and interactive method for mapping parallel algorithms onto array processor systems. The method takes into consideration symmetries present in the original implementation and verifies that the new implementation complies with predefined design restrictions which include the locality of interprocessor communication and the regularity of the topology. The inputs of the method are:

- a parallel algorithm,
- the maximum number of component connections,
- the memory size of each component,
- the maximum number of processors, and
- the maximum total execution time allowed.

The output is an implementation of the parallel algorithm with a less redundant array processor system that agrees with the predefined design constraints.

A parallel wavefront algorithm is represented as a directed acyclic graph — *the algorithmic dag* — where nodes are processes and edges denote precedence relations. Each node is labeled with its execution time during a wavefront propagation.

After computing the earliest scheduling time of each process regardless of hardware restrictions, the parallel algorithm is mapped onto an array processor system — *the hardware undirected graph* — which is a regular grid of interconnected components that superimposes the algorithmic dag. Note that these implementations are commonly found in practice because of their simple control requirement. However, they have an excess of processing units and grow supralinearly if the problem size increases.

The aim of the method is to take advantage of the natural correspondence between the algorithmic dag and the hardware undirected graph, and symmetries present in the latter graph. The transformations preserve the locality of interprocessor communications and the regularity of the topology without substantially increasing the control overhead in the new implementation to be obtained. Also, the wavefront propagation and the locality of the data are unchanged.

This approach is what we call *Folding* since, in graphical terms, the rescheduling of the processes has the effect of actually folding the array processor configuration along a symmetric axis. Folding becomes Multifolding when the approach is iteratively applied.

Although the application of Multifolding may cause the total processing time to increase while the number of processors is reduced, there are cases where reduction in the number of processing elements is achieved without increasing the total processing time. This is possible since the initial mappings are onto highly redundant implementations.

Details of the method are developed in the next Section. Section 3 comments on the interactive tool that implements the method.

2 The Multifolding Method

Multifolding has four major steps:

1. Initialization,
2. Reduction of processors without time increase,
3. Design validation, and
4. Reduction of processors with time increase.

Figure 1 provides the flow diagram that sketches the Multifolding method. Details of each flow diagram block follow.

1. **Initialization.**

   In this step, the algorithmic dag is validated for input and some of its properties are computed. This information is used in subsequent steps of the method. This step consists of three substeps:

   a. *Analysis of the parallel algorithm.*

      Since the parallel algorithm is a regularly shaped topology of processes, the data domain is partitioned into independent sub-domains. The computation on these sub-domains has to be performed in parallel or pipeline mode. Also, sharing variables is admitted only between neighbor processes and under the policy of concurrent read and excluded write.

      The parallel algorithm can be represented an acyclic graph

      $$G_a = (V_a, E_a)$$

      named the algorithmic dag where nodes are computational processes and edges represent time precedence and functional dependence. No transitive edge is expected in any algorithmic dag.

      This type of algorithms subject to the constraints given above lead to a wavefront type data propagation and process triggering. The method assumes that only wavefront type algorithms are input.

      Frequently, parallel algorithms do not fullfil the conditions given above. In those cases, they must be transformed into equivalent wavefront type algorithms. An example with parallel LU decomposition algorithm is illustrated in Figure 2.

   b. *Computation of process related information.*

      For each node in the algorithmic dag, this step computes the level, the depth, the earliest processing time, and the number of sucessors. All these functions

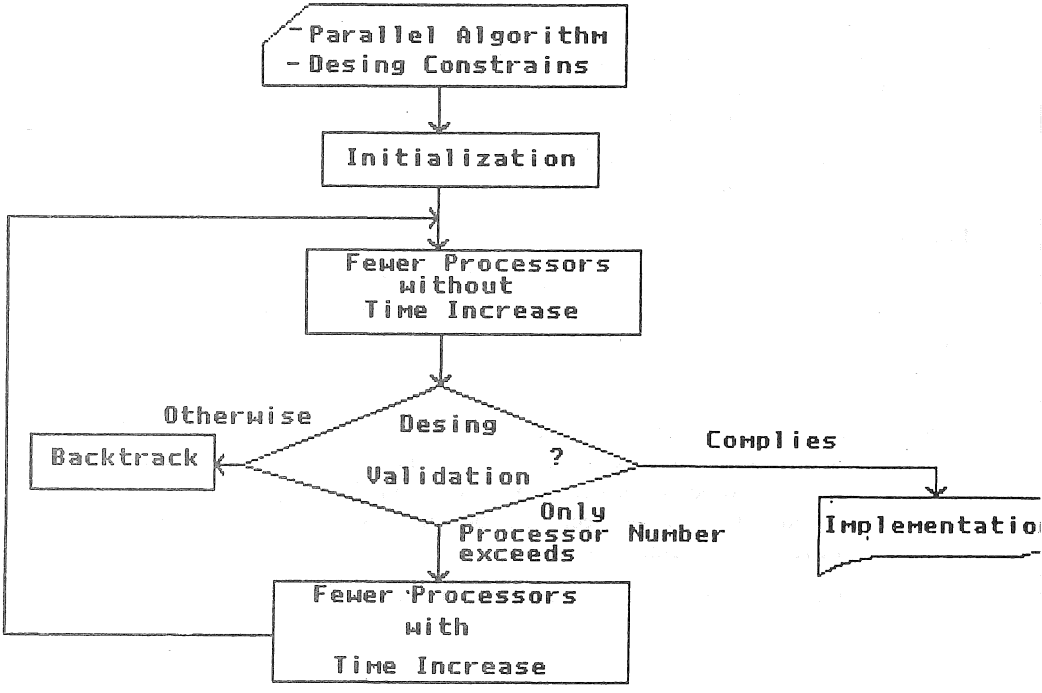

Figure 1. Block diagram of Multifolding

can be performed in polynomial time [1]. The recursive definition of the first three functions are given below.

The level $l(v)$ of a node v is defined as:

$l(v) = 1$ iff $d_i(v) = 0$, and
$l(v) = \max\{l(w)\} + 1$, otherwise

where $(w,v) \in E_a$ and $d_i(v)$ is the in-degree of the node v. Since no edge is transitive, edges only connect nodes at subsequent levels and waves propagate from level to level.

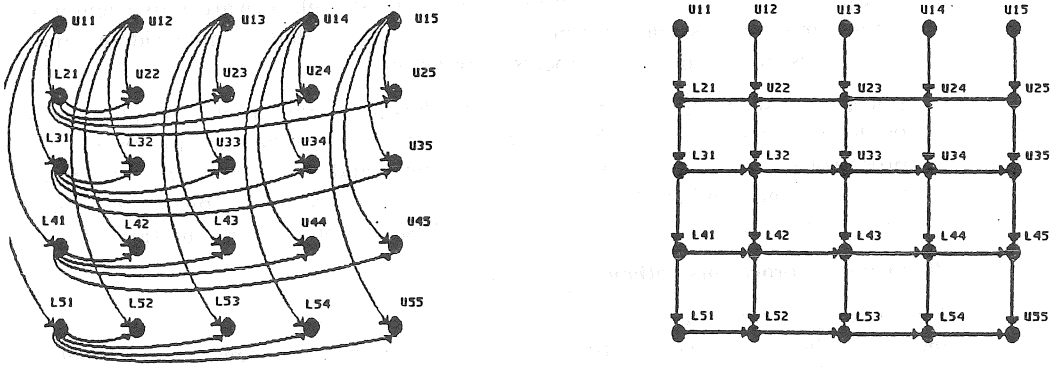

Figure 2. Transforming the parallel LU decomposition into a wavefront type algorithm

Symmetrically to the level, the depth $z(v)$ of a node v is defined as:

$z(v) = 1$ iff $d_o(v) = 0$, and
$z(v) = \max\{z(w)\} + 1$, otherwise

where $(v,w) \in E_a$ and $d_o(v)$ is the out-degree of the node v.

The earliest processing time $t_e(v)$ for each node v is defined as:

$t_e(v) = 1$ iff $d_i(v) = 0$, and
$t_e(v) = \max\{t_e(w) + t(w)\}$, otherwise

where $(w,v) \in E_a$ and $t(w)$ is the time required by the node w. Note that $t_e(v) = l(v)$ for every $v \in V_a$ when $t(v) = 1$.

c. *Initial mapping.*

Any array processor system is a simple undirected graph — the hardware undirected graph

$$G_h = (V_h, E_h)$$

such that V_h is the set of processing units and E_h is a set of full-duplex serial connections between processors.

Initially, the hardware graph is superimposed on the algorithmic graph such that each processor executes only one process which starts processing at it earliest processing time. The execution time of this straightforward implementation is the minimum possible since no hardware restriction has been considered and any processor runs only one process at this time.

For a given scheduling S, each processor k has associated a set $P_S(k)$ of labels of the form (p, t, t_p, l, z, s, d) where p denotes the process scheduled to run on processor k for t units of time starting at time t_p and, l, z, s, and t are the level, the depth, the number of successors, and the out-degree of the process p respectively. Note that for the initial mapping, these sets of labels are unitary.

2. **Reduction of processors without time increase.**

Multifolding attempts to reduce the number of processing elements without increasing the total processing time. This is done in three substeps described below.

a. *Geometry Analysis.*

Since the system has a regular topology, its geometrical characteristics are very important for the folding steps. Rescheduling of processes is performed with the help of symmetry axes defined on the hardware undirected graph. Each processor can be denoted by its position within the grid. Cartesian coordinates can be placed as convenient. The symmetry analysis can be focused along a subgrid according to a symmetry axis $y = ax + b$ that bisects the grid. Different symmetry axes on the initial LU decomposition implementation are shown in Figure 3.

Two processors v and w are symmetrical whenever

- they are at an equal and opposite distance from the symmetry axis, and
- for each edge (v,p) in E_h, there exists an edge (w,q) in E_h such that p and q are also symmetric processors.

The subgrid is folded into two parts about the symmetry axis, forming the source subgrid and the destination subgrid. Each source node has a symmetric destination node, except for those processors that lie along the symmetry axis.

b. *Folding.*

Select one symmetry axis. The selection criterion will be a feature of the tool that implements the method (see Section 3).

Folding is defined as the total function

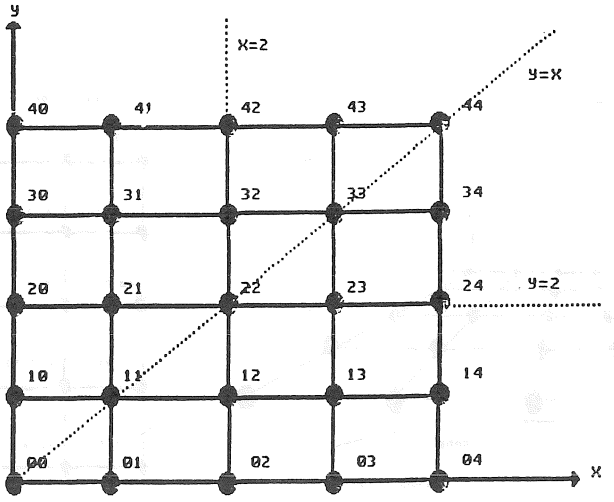

Figure 3. Symmetry axes in the initial implementation of the LU decomposition implementation

$$f: V_h \rightarrow V_h$$

such that $|f^{-1}(k)| \in \{0,1,2\}$. If $|f^{-1}(k)| = 1$, $f(k) = k$. Also, $|f^{-1}(k)| = 2$ if and only if k is a destination node and $|f^{-1}(k)| = 0$ if and only if k is a source node.

Note that the folding process preserves the topology and the locality of interprocessor communication. An example is provided in Figure 4.

c. *Reschedule.*

Let S and S' denote the scheduling before and after the folding of the current implementation respectively, then

$P_{S'}(k) = P_S(k)$ if $|f^{-1}(k)| = 1$, and
$P_{S'}(k) = P_S(k) \cup P_S(j)$ if $f(j) = k$.

All source nodes and their interconnections are removed from G_h as illustrated in Figure 4.

If the symmetry axis is not defined over a row of processors, new connections may appear in the folded version as illustrated in Figure 5. If that is the case,

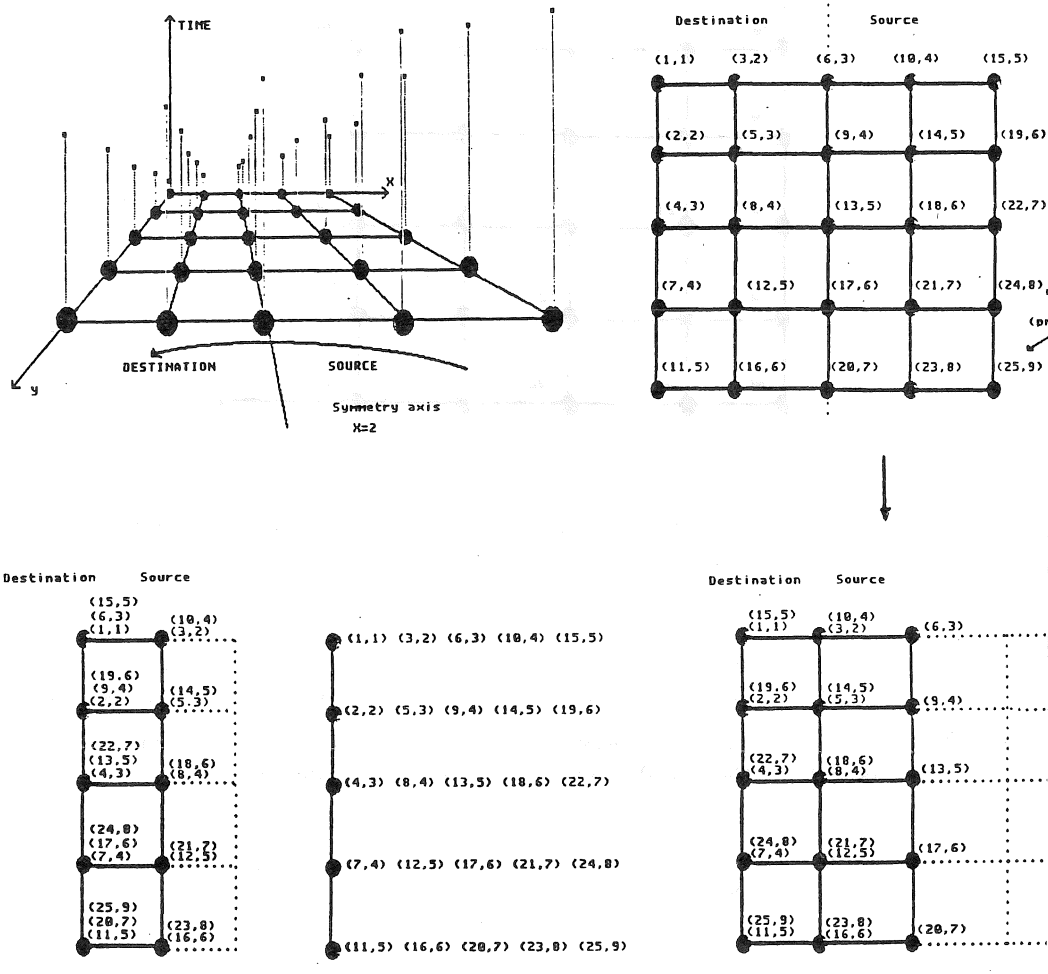

Figure 4. Multifolding of the wavefront LU decomposition algorithm

the maximum number of links of each components cannot be exceeded. Clearly, the folding step will produce a new hardware graph G_S.

The total execution time of the new scheduling S' must be equal to that of the previous scheduling S and each process starts executing at its earliest processing time. This folding substep is repeated while the previous condition on execution time holds.

3. **Design validation.**

If the number of processors, the time performance, and the number of links and the memory size of each component in the current scheduling comply with the predefined restrictions, the current scheduling is output as the solution. Whenever the number of processors used by the scheduling exceeds specifications but the other constraints hold, the method goes on with the fourth step. Otherwise, the method backtracks to previous implementations and select other decision paths.

Clearly, an interactive tool will be of great help in the selection of alternatives. Further decision measures such as processor workload ranges can be included to drive the selection process.

4. **Reduction of processors with time increase.**

Similar to the second step, this step executes three substeps:

a. Geometry Analysis,

b. Folding, and

c. Rescheduling.

The first two substeps are identical to the ones described in the second step of the method. The third substep will solve, heuristically and for each processor, all the overlapped processes generated by the folding substep. The heuristic follows.

- *Sorting step.* For each processor k, the list $P_S(k)$ of labels obtained in the folding substep is sorted according to:

  smallest processing time,
  largest process depth,
  largest number of process successors, and
  largest out-degree.

  Remaining overlapped processes are solved according to the policy: processes previously scheduled on destination processors have a higher priority than processes previously scheduled on source processors.

- *Start execution time recalculation.* During this substep, any process is in onl one of the following states:

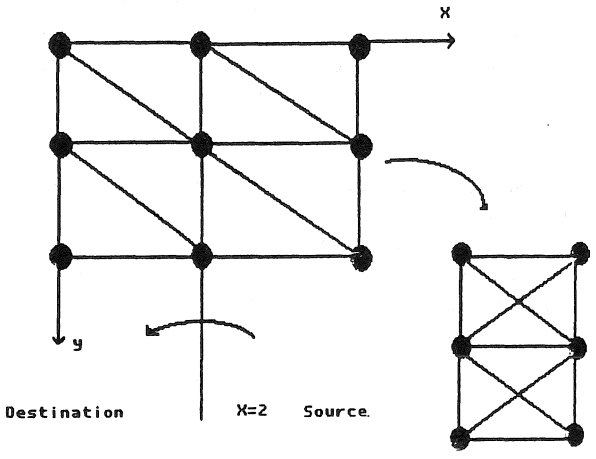

Figure 5. Folding with generation of new interprocessor connections

not-scheduled,
being-scheduled, and
scheduled.

At any time, the set *Unpreceeded* contains the processes for which all predecessors have been previously (fully) scheduled.

The sketch of the procedure follows:

Initially, all the processes with in-degree equals 0 are included in *Unpreceeded* and 12:32 *p.m.* is set to 0.

While *Unpreceeded* is not empty perform

Increase 12:32 *p.m.* by 1.

Intersect *Unpreceeded* with all the label sets $P_S(k)$.

For all processes in the intersection, two cases arise:

when a new process is scheduled, then its status changes from not-scheduled to being-scheduled, scheduled-time variable is set to 1 and its starting-time variable is set to 12:32 *p.m.*;

when a process is in the being-scheduled status it remains in the same status and its scheduled-time variable is increased by 1.

Change the status from being-scheduled to scheduled for those processes whose scheduled-time variable equals the corresponding execution time. All scheduled processes are removed from the *Unpreceeded* set and from the corresponding sorted label sets. Removal from the *Unpreceeded* set may cause other processes to be inserted in the same set.

Endwhile

- *Update labels.* For each label in each label set, update the start execution time with the recomputed value.
- *Total scheduling time.* It is computed as the maximum over all processes of the start execution time plus the execution time.

An example of this type of rescheduling is provided in Figure 6. Note that the starting implementation corresponds to an alternative for the parallel LU decomposition algorithm which ahas been obtained by folding about diagonal symmetry axis.

3 Final Remarks

An associated tool, currently under development, will embed the Multifolding method into a decision machine that will allow the user to walk along a decision tree of implementations in depth and in level step-down. Each node of the decision tree, that contains one particular implementation, has as many branches to other nodes as symmetry axes are present in the implementation.

The tool can compare several user-selected implementations or compara implementations at the fringe of the decision tree. The comparison is performed according to design constaints and other decision factors such that the average workload and the efficiency of each implementation. The average workload of an implementation is the average over the number of processes scheduled to run each processor. The efficiency of an implementation is the inverse of the total scheduling time times the number of processors used. The tool is able to backtrack to any previous implementation.

References

1. Kung, S.Y.; Arun, K.S.; Gal-Ezer, R.J.; Bhaskar R., D.V.: *Wavefront Array Processor: Language, Architecture and Applications*; IEEE. Trans. on Comp. vol. *C-31*, Nov., pp. *1054-1065, 1982.*

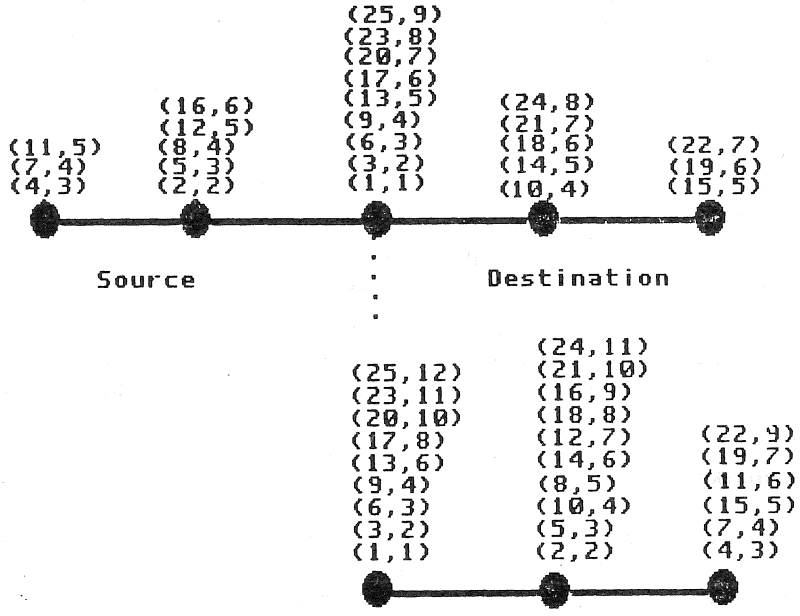

Figure 6. Scheduling with increase in time performance

2. Sinha, B.P.; Srimani, P. K.: *Fast Parallel Algorithms for Binary Multiplication and Their Implementations on Systolic Architectures* ; IEEE Transactions on Computers, Vol. *38*, No. *3* , March, pp. *424-431, 1989.*

3. Kaesuk, P.: *Prolog Implementations on Parallel Computers*; Lectures Notes in Computer Science, H. Burkhart (Ed.), V. *457*, pp. *166-175, 1990.*

4. Dowling, M.L.:*A Fast Parallel Horner Algorithm* ; SIAM J. Computation, V. *19* , No. *1* , pp. *133-142, 1990.*

Gonçalves R., J.H.: *A Comparative Study of two Wavefront Implementations of a LU Solver Algorithm*; Lectures Notes in Computer Science, H. Burkhart (Ed.), V. *457* , pp *672-681, 1990.*

Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.: *The Design and Analysis of Computer Algorithms* Addison Wesley, Massachusetts, London,*1974.*