

Reducing Inter Iteration Dependency Delays in Multiprocessor Systems for Large Graphs

Michael Sheliga
Department of Technology
Eastern Kentucky University
Richmond, KY 40475

Nelson Luiz Passos
Department of Computer Science
Midwestern State University
Wichita Falls, TX 76308

Edwin Hsing-Mean Sha
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083

ABSTRACT

Applications such as image processing, fluid mechanics, and weather forecasting require high computer performance. Researchers and designers in those areas are looking for solutions to multi-dimensional problems through the use of parallel computers and/or specialized hardware. It is known that in highly parallel computers communication is often the limiting execution speed bottleneck. While the problem of calculating and minimizing communication costs due to loop data dependencies has been widely studied, such research has involved changing the way iterations are partitioned, not modifying the graph or loop data dependencies themselves. This paper uses algorithms that minimize loop data communication for multi-dimensional graphs by modifying the structure of the input graph and changing the distribution of the loop dependencies using multi-dimensional retiming. These algorithms are extended to large graphs and compared for effectiveness. Results are shown which illustrate the efficiency of the algorithms as well as the savings achieved for large graphs.

Keywords: Multi-Dimensional Data Flows Graphs, Retiming, Retiling, Loop Tiling and Communication Minimization.

1. INTRODUCTION

The use of parallel computer systems has reduced the execution time of parallel applications considerably. In addition, the speed of individual processors has also increased leading to a further reduction in system latency. Unfortunately, with the increases of

speed in today's processors, it has been difficult for parallel systems to keep up with the resulting interprocessor communication bandwidth. As such communication costs are becoming an increasingly important design consideration.

Communication minimization requires not only an efficient strategy for the partitioning of computational tasks, but also an optimized method of distributing delays in the graph (retiming) so as to reduce the amount of data that needs to be moved between iterations. Much of the initial research on retiming focuses on one-dimensional scheduling problems [7, 9, 12]. Multi-dimensional retiming research was originally focused around the improvement of parallelism inherent in multi-dimensional applications. Some studies focusing on uniform nested loop scheduling are similar to the solution of the multi-dimensional problem. These loop quantization [2]. These techniques do not change the structure of the iterations, and therefore may not achieve a fully parallel solution. Other techniques that search beyond loop boundaries include perfect pipelining [3] and Doacross loops [10]. Perfect pipelining has the disadvantage of unpredictability of the size of the repeating pattern and the gain of performance. Doacross presents an overlap of consecutive iterations that contributes to the improvement of the execution rate. These two methods also have their limitations and may not get the optimal result.

Later research has studied the scheduling of multi-dimensional applications. For example, the affine-by-statement technique [11], and the index shift method [18] are able to achieve a fully parallel execution of multi-dimensional tasks, utilizing algorithms based on linear programming techniques. However, these methods do not consider possible memory changes and, consequently, they may introduce new queues dependent on the problem size. This research culminated in the chained multi-dimensional retiming technique proposed in [19, 20] which is able to achieve a fully parallel solution in polynomial time while considering memory elements.

This work was partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, NSF IIS-0513669, and Microsoft, USA

Since then research in this field has been applied and expanded to a number of areas including clustered computing systems [1, 4, 13] and superscalar processors [6]. In [12] multi-dimensional retiming was applied to DSP applications, similar to the ones considered in this study, but once again focusing on parallelization.

While the above results use multi-dimensional retiming, they are geared towards full parallelism and scheduling, not communication minimization. Previous research on communication minimization, on the other hand, has concentrated on decreasing the communication by changing the partition, not by modifying the delays themselves.

Research on the best way to minimize communication for a given data flow graph, was introduced with Irigoien and Triolet's paper [16] which set the background assumptions for the problem (which they termed "supernode partitioning", but which most later papers call "tiling"). They presented a technique called hyperplane partitioning, which partitions the iteration space into regularly spaced hyperplanes. However, in their paper they do not present a method of determining the size or shape of the tiles so as to minimize the communication.

Ramanujam and Sadayappan proposed a solution that considered only lower triangular matrices [21]. As such, it did not always find the optimal solution. Later, Boulet and others proposed a method of generating an optimal tile for a given data flow graph [7]. These results relied on the fact that the *effective communication* for different size partitions is the same, as long as the partitions are scaled properly. For example, for two-dimensional graphs, their results showed that the communication grows as the square root of the number of nodes, as long as the tile shape remains constant. Hence, the communication per square root of area will be constant even as the number of nodes per partition increases. These results were made more precise by Calland and Risset [8], by noting that redundant communication between partitions may be eliminated. Hodzic and Shang further extended these results to include the calculation of the size of each partition so as to minimize the total running time [14]. More recently, tiling has been used to improve cache utilization [5, 15, 17]. However, these last results did not consider communication time between processors.

In this paper retiming is used to minimize the communication for a graph that represents data dependencies of uniform nested loops. A simple example of uniform loop dependencies can be found in dependencies between accesses to the same array, as in the following code segment.

```

for i = 1 to 1000
    for j 1 to 1000
        A[i,j] = A[i-1, j-1] + A[i-1, j-2].
    end
end

```

If different processors calculate each $A[i, j]$ this data will need to be transferred between processors. However, if the same processor calculates two or more iterations, the values of certain neighboring iterations will not have to be transferred to another processor. For example, if processor 1 calculates the values of $A[6,6]$, $A[5,5]$ and $A[5,4]$ in the above example, then no data from another processor is needed to calculate $A[6,6]$.

This paper uses existent techniques to determine both the shape, and size of the iterations to be executed on a particular processor for a given set of loop dependencies. Furthermore, this paper also minimizes the communication requirements imposed by uniform inter-array loop dependencies by changing the dependencies themselves. These dependencies are difficult to modify since changing one dependency often means changing other ones as well. Three techniques are used to reduce the communication requirements. First, the structure of the graph is modified where possible so as to combine dependencies and eliminate nodes. Second, the communication is reduced by using the method of multi-dimensional retiming. Third, the overall magnitude of the dependencies is reduced so as to lessen the communication. These algorithms were previously shown to be able to give minimal results for most real-world filters [22]. Therefore these algorithms are tested on larger, randomly generated graphs and examine how they perform.

Section 2 describes basics of retiming and the communication calculations. Section 3 introduces the flow of algorithms involved. Section 4 demonstrates the effectiveness of the algorithm for large input graphs while section 5 draws a conclusion from the results obtained and discusses ideas for future research.

2. PRELIMINARIES

Multi-dimensional Graphs, Retiming and Loop Partitions

A *multi-dimensional data flow graph* (MDDFG) $G = (V, E, d, t)$ is a node-weighted and edge-weighted directed graph, such as the one shown in Figure 1(A). Figure 1(A) presents a simple multi-dimensional data flow graph (DFG). The vectors next to the edges in the graph represent multi-dimensional delays that indicate what iteration data will be used in. For example, the vector (0,2), on edge $e_{X,C}$ represents the fact that data generated by node X during iteration (0,0), will be used by node C during iteration (0,2). Figure 1(B), shows the associated iteration space. In the iteration space delays are represented by the lines with arrows, while the group of iterations that are to be executed by a single processor is bounded by dashed lines. For example, the delay associated with edge $e_{X,C}$ is shown by the thick black arrow in the (0,2) direction.

A *multi-dimensional retiming* r is a function from V to Z^n that redistributes the nodes in the original dependence graph created by the replication of an MDDFG G . In summary, a multi-dimensional retiming "pushes" the same delay from all incoming edges of a node to all outgoing edges of the same node, or vice-versa. As an example consider Figures 1(E) and (G). Notice these graphs are exactly the same except that the two delays of value (3,2) on both incoming edges of node C in (E) have been pushed through node C to node C's only outgoing edge in (G). For further examples and details see [18].

Intuitively it is important to know that the vectors on each edge (delays) are responsible for the communication requirements of the graph. It may be shown that when calculating the effective communication for such a graph it is best to align the iterations to be executed on the same processor, or loop tile, with the worst (outermost or extreme) delay vector of the graph. Hence, in (B), the partition is aligned with the vectors (2,0), and (0,2). By doing so the effective total communication will be minimized. Further details and examples may be found in [7].

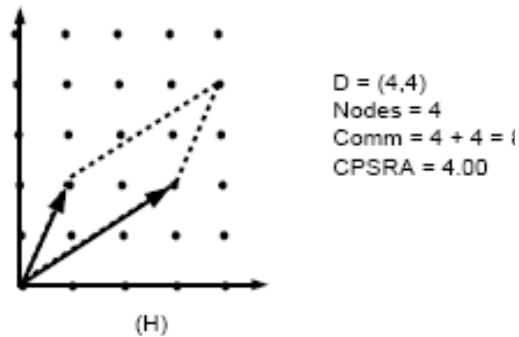
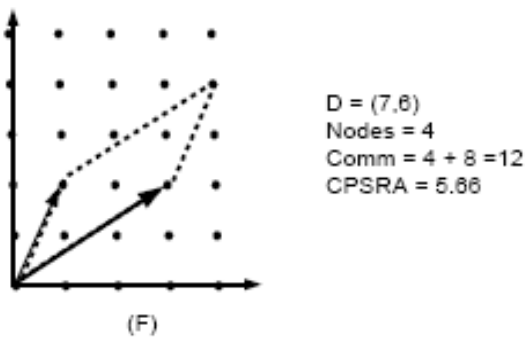
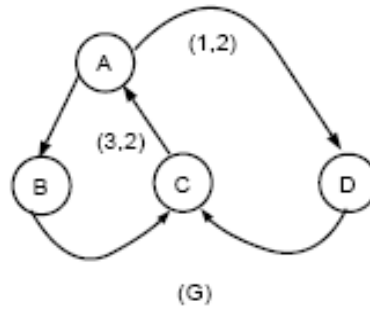
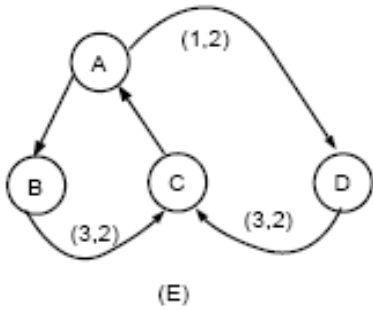
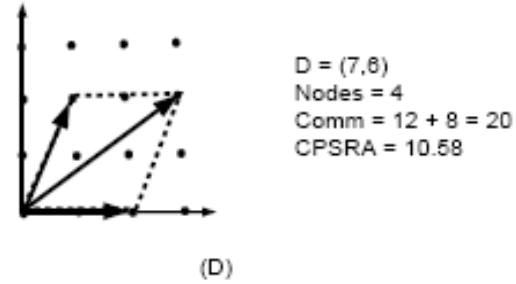
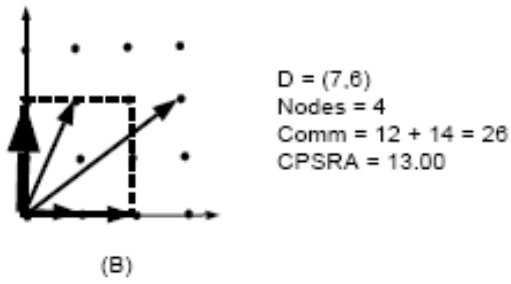
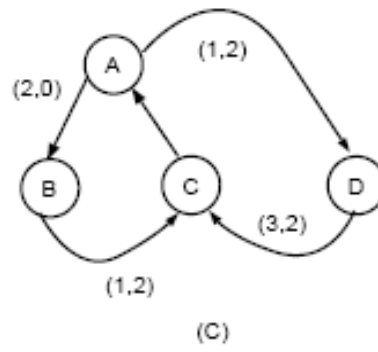
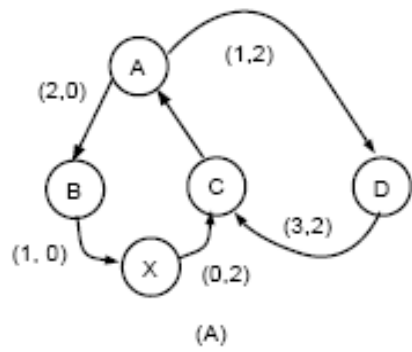


Figure 1: (A) Original data flow graph. (B) Iteration space for the original data flow graph with effective communication of 13.0. (C) The graph after graph simplification. (D) Iteration space for (C) with effective communication of 10.58. (E) The graph after angle modification. (F) Iteration space for (E) with effective communication of 5.66. (G) The graph after delay reduction. (H) Iteration space for (G) with effective communication 4.0.

Furthermore, for a given graph, the best possible partition is may easily be found. Hence, once the extreme delay vectors of a graph are known the CPSRA can easily be determined. The key result for the partition in (B) is that the effective total communication delay for the graph, or CPSRA, is equivalent to 13.0 communication units. Notice that the modified graph in (E) has a CPSRA of 5.66 as noted in (F) while the retimed graph of (G) has a CPSRA of 4.00 as noted in (G).

While the details of retiming or minimizing the CPSRA may not be clear to unfamiliar readers, there are several important results to remember. (1) Retiming pushes delays from all incoming edges of a node to all outgoing edges. This normally changes the angle of all the associated edges. (2) The total communication delay will be less if the outermost delays are “pulled” towards each other. (3) The total communication delay will be less if the magnitude of all delays is decreased. (4) The total communication delay (CPSRA) for a graph can be optimized once the best delays have been found and the outermost (extreme) delays have been established. Therefore retiming is used to “draw in” the outermost delay vectors. Note, however that reducing the angle of one delay may have the unintended consequence of increasing the angle of another delay. Finally note that a basic theory of retiming is that the total delay for any loop can not be changed. Therefore, (5) the loops with the most clockwise and counter-clockwise sum of delays give us the optimum that can be achieved.

Communication Minimization Algorithm

Figure 1 can also be used to introduce the concepts of the overall communication minimization algorithm. Figure 2 shows the overall flow of this algorithm. The overall communication minimization algorithm begins with graph simplification, followed by angle modification and then uses delay reduction. The main goal of the algorithm is to change the delay vectors of the graph so as to reduce the communication. This is largely done using multi-dimensional (MD) retiming to draw in the outermost delay vectors.

Since deciding on the exact retiming vectors is a difficult and computationally intensive problem, it is worthwhile to reduce the complexity of the graph using *graph simplification*. Figures 1(C) and (D) demonstrate the concept of graph simplification which is an important pre-processing step. During this process, certain nodes are eliminated from the graph. For example, in (C), node X is eliminated and its associated edges are combined to produce a new edge, $e_{B,C}$ with a delay equal to the sum of delays on edges, $e_{B,X}$ and $e_{X,C}$. Note that in the associated iteration space shown in (D), the outermost delay vectors as well as the associated partition have been modified, with the resulting effective communication reduced to 11.55. Just as importantly, the graph has been simplified by eliminating node X.

Once graph simplification is done, MD retiming is used to reduce the communication in two ways. First, it is possible to modify the extreme vectors of the graph using MD retiming. This is known as *angle modification*. This enables the shape of the partition to be changed, and the resulting communication to be reduced.

Figures (E) and (F) demonstrate the concept of *angle modification*. During this process, *multi-dimensional retiming* is

used to redistribute the delays of the graph so as to reduce the angle of the outermost (extreme) delay vectors. In (E) delay (2, 0) is “pushed” through node B, resulting in a new delay on edge $e_{B,C}$ of (3, 2). The angle of the resulting partition, as shown in (F) has been further reduced, and the effective communication reduced to 5.66. Note that graph simplification could also have been used to accomplish this task, with the additional benefit of simplifying the graph. When possible, graph simplification is the preferred technique, however there are many cases when it is not possible.

Angle modification is accomplished using two different algorithms which are compared for large graphs in the results. The first algorithm, *extreme vector modification*, changes only the worst delays in the graph by “borrowing” delays from neighboring edges. This is repeated with the next worst delay until no more delays can be improved. The second algorithm, *loop modification*, changes all delays in a loop at once. Loop modification takes a different approach than extreme vector modification. It considers the delay sum of entire loops instead of individual vectors. The intuition behind this approach is to retime the loops that have the worst angles first, since these loops will be the hardest to adjust.

Finally, after angle modification, it is possible to reduce the overall magnitude of the delay vectors in the graph, thereby reducing the overall communication. This is known as *delay reduction*. During delay reduction delays are pushed through nodes that have more incoming edges than outgoing edges, or more outgoing edges than incoming edges. However, this is only done when it does not change the extreme vectors of the graph.

Figures 1(G) and (H) demonstrate the concept of *delay reduction*. With this technique the angle of the partition is not changed. Instead, the sum of all delays in the graph is decreased. The sum of all delays in the graph in (E) is (7,6). Figure 1(G) shows a retimed version of 1(E). In (E), where a (3,2), delay is pushed through node C. By doing so the sum of all delays in the graph is reduced from (7,6), to (4,4). The new effective communication, as shown in (H), is 4.0, which is 31% of the communication in (B). Note that delay reduction is done after angle modification since delay reduction, if done properly, will not change the angle of the outermost vectors, but angle modification is likely to change the sum of the delays in the graph. Further details of this algorithm may be found in [22].

3. EXPERIMENTAL RESULTS

This section presents experimental results for several different sizes of large graphs. Figure 3 shows the results for graphs from with between 20 and 70 nodes. Several comments should be made to explain the figure. These graphs were randomly generated with both sparse and dense number of edges ranging from $N \log N$ to twice $N \log N$, where N is the number of nodes. The loop types of the graphs were also varied. For “rand” loops all edges were randomly generated. For large loops edges were added such that some loops comprised of all nodes were created. The “Orig Spread” column represents the original spread of the outermost delays in the original graph, in degrees.

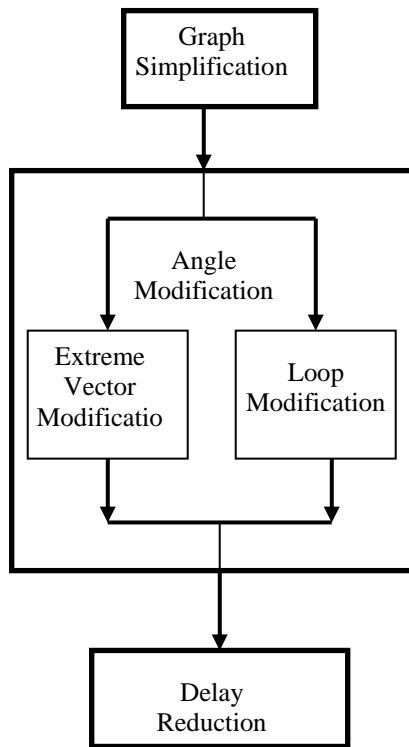


Figure 2: The Overall Algorithm Flow

The “Min Spread” represents the minimum spread between the angles of the outermost loops of the graph. It should be noted that this angle can not be changed and therefore represent the minimum angle attainable for the modified graph. It should also be noted that the numbers in these columns are highly dependant upon the algorithms used to add delays to the edges. In short adding edges and creating loops of the correct type is analogous to the inverse of the problem considered in this paper. In all the results for each row below represent the averages for ten graphs of the indicated types.

Results are next compared for the two main angle modification algorithms. For each algorithm the final spread, extra spread and final communication percentage is given. The “Final Spread” is the final angle between the outermost angles in degrees. The “Extra Spread” is the difference between the final spread and theoretical minimum spread. Rows with an extra spread of 0 indicate that the algorithm was able to retime the graph and achieve the minimum possible spread. This column is largely used as a benchmark of the success of the algorithms. Finally the “Final Comm Percent” column indicates the percentage of communication for the final graph compared to the communication for the original graph. As an example, for the first row a graph of 20 nodes with $20\log 20$ edges was created. All edges were created between two random nodes. The original spread of the graph’s vectors was 156 degrees while the minimum spread of the loops was 84 degrees. The extreme vector algorithm was able to achieve a modified spread of 84 degree and a final communication of 475 of the original. The loop modification algorithm was able to do the same. Cells with dashes in them indicate that the indicated algorithm was not able to be completed on the given graph on a standard

computer in less than one hour. Results for such graphs were not considered.

Analysis of these results shows that extreme vector modification is very efficient. It is able to retime the graph in most cases such that the theoretical minimum spread as determined by the loops is achieved. It is not able to reach the minimum spread in all cases, especially for “dense” graphs with large number of edges; however it is close. On the other hand, due to the time intensive nature of the algorithms involved its run time is more than one hour for as little as 60 nodes. On the other hand the loop modification algorithm is able to be completed for all but the largest graphs in a reasonable amount of time. However it is less successful in modifying the delays and results in spreads that are up to 22 degrees away from optimal. These results match the results found in [22] which showed the extreme vector modification was able to achieve better results since it did not get stuck in local minima.

4. CONCLUSION

Applications such as image processing, fluid mechanics, and weather forecasting, require high computer performance. Researchers and designers in those areas are looking for solutions to multi-dimensional problems through the use of parallel computers and or specialized hardware. It is known that in highly parallel computers, communication is often the limiting execution speed bottleneck. While the problem of calculating and minimizing communication costs due to loop data dependencies has been previously studied, such research has involved changing the way iterations are partitioned, not modifying the graph or loop data dependencies themselves. This paper presents algorithms that minimize loop data communication for multi-dimensional graphs by modifying the structure of the input graph and changing the distribution of the loop dependencies themselves using multi-dimensional retiming. The algorithms presented are applied to large graphs. These algorithms are able to find the optimal result in many cases. However, due to the time intensive nature of the algorithms they are limited to certain sizes of graphs. Results that illustrate the savings which are possible with these solutions are presented for several randomly generated input systems.

5. REFERENCES

- [1] J. H. Abawajy and S. Dandamudi, "Parallel Jobs Scheduling on Multi-cluster Computing Systems," in Proceedings of the International Conference on Cluster Computing, Hong Kong, December 2003, pp. 11-18.
- [2] A. Aiken and A. Nicolau, "Loop Quantization: An Analysis and Algorithm," Technical Report 87-821, Department of Computer Science, Cornell University, March 1987.
- [3] A. Aiken and A. Nicolau, "Perfect Pipelining: A New Loop Parallelization Technique," in Proceedings of the 1988 Symposium on Programming, Lecture Notes in Computer Science, no. 300, March, 1988, pp. 221-235. 25
- [4] C. Akturan and M. F. Jacome, "CALiBeR: A Software Pipelining Algorithm for Clustered VLIW Processors," in Proceedings of the International Conference on Computer-Aided Design, November 2001.
- [5] P. D'Alberto, A. Nicolau, A. Veidenbaum, R. Gupta, "Line Size Adaptivity Analysis of Parameterized Loop Nests for

Graph Characteristics					Extreme Vector Modification			Loop Modification		
Nodes	Edges	Loop Type	Orig Spread	Min Spread	Final Spread	Extra Spread	Final Comm Percent	Final Spread	Extra Spread	Final Comm Percent
20	NlogN	Rand	156	84	84	0	47	84	0	47
20	NlogN	Large	158	82	82	0	47	90	8	50
20	2NlogN	Rand	160	79	79	0	48	79	0	48
20	2NlogN	Large	161	84	84	0	46	95	11	50
40	NlogN	Rand	170	88	88	0	41	88	0	41
40	NlogN	Large	170	86	86	0	43	95	9	50
40	2NlogN	Rand	172	88	94	6	42	102	14	52
40	2NlogN	Large	171	87	87	0	42	99	12	48
60	NlogN	Rand	175	90	95	5	40	108	18	53
60	NlogN	Large	175	89	93	4	41	107	18	55
60	2NlogN	Rand	176	90	-	-	-	104	14	51
60	2NlogN	Large	176	89	-	-	-	106	17	55
70	NlogN	Rand	176	92	92	0	40	112	20	57
70	NlogN	Large	176	92	100	8	42	114	22	58
70	2NlogN	Rand	176	92	-	-	-	-	-	-
70	2NlogN	Large	176	92	-	-	-	-	-	-

Figure 3: Comparison of Algorithms for Large Filters

- Direct Mapped Data Cache” in IEEE Transactions on Computers, February 2005, pp. 185-197.
- [6] V. Andronache, R. P. Simpson, N. L. Passos, “An Efficient Implementation of Nested Loop Control Instructions for Super Scalar Processors,” in Proceedings of the Midwest Symposium on Circuits and Systems, Notre Dame, IN, August, 1998, pp. 82-85.
- [7] P. Boulet, A. Darté, T. Risset and Y. Robert, “(Pen)-ultimate Tiling,” in Proceedings of the Scalable High Performance Computing Conference, pp. 568-576, May 1994.
- [8] P.-Y. Calland and T. Risset, “Precise Tiling for Uniform Nested Loops,” in Proceedings of the 1995 IEEE International Conference on Application-Specific Array Processors (ASAP’95), pp. 330-337, 1995.
- [9] E. Cohen and Nimrod Megiddo, “Strongly Polynomial-Time and NC Algorithms for Detecting Cycles in Dynamic Graphs,” in Proceedings of the 21st ACM Annual Symposium on Theory of Computing, 1989, pp. 523-534.
- [10] R. Cytron, “Doacross: Beyond Vectorization for Multiprocessors,” in Proceedings of the International Conference on Parallel Processing, 1986, pp. 836-844.
- [11] A. Darté and Y. Robert, “Constructive Methods for Scheduling Uniform Loop Nests,” IEEE Transactions on Parallel and Distributed Systems, 1994, Vol. 5, no. 8, pp. 814-822.
- [12] F. Fernandez, and A. Sanchez, “Application of Multidimensional Retiming and Matroid Theory to DSP Algorithm Parallelization,” in Proceedings of the 25th EUROMICRO Conf., September 1999, V. 1, pp. 511 - 518.
- [13] L. He, S. A. Jarvis, D. P. Spooner and G. R. Nudd, “Dynamic Scheduling of Parallel Real-time Jobs by Modeling Spare Capabilities in Heterogeneous Clusters,” in Proceedings of the International Conference on Cluster Computing, Hong Kong, Dec. 2003, pp. 2-10.
- [14] E. Hodzic and W. Shang, “On Supernode Transformation with Minimized Total Running Time,” in IEEE Transactions on Parallel and Distributed Systems, May 1998 pp. 417-428.
- [15] Q. Huang, J. Xue and V. Xavier, “Code Tiling for Improving the Cache Performance of PDE Solvers,” in Proceedings of the 2003 International Conference on Parallel Processing (ICPP’03), October 2003, pp. 615.
- [16] F. Irigoien and R. Triolet, “Supernode Partitioning,” in Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages, Jan 1988, pp. 319-329.
- [17] Z. Li, “Optimal Skewed Tiling for Cache Locality Enhancement” in Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS’03), April 2003, pp. 37a.
- [18] L.-S. Liu, C.-W. Ho and J.-P. Sheu, “On the Parallelism of Nested For-Loops Using Index Shift Method,” in Proceedings of the 1990 International Conference on Parallel Processing, 1990, Vol. II, pp. 119-123.
- [19] N. L. Passos and E. H.-M. Sha “Full Parallelism in Uniform Nested Loops using Multi-Dimensional Retiming,” in Proceedings of 23rd International Conf. on Parallel Processing, August, 1994, vol. II, pp. 130-133.
- [20] N. L. Passos, E. H.-M. Sha, and S. C. Bass, “Partitioning and Retiming of Multi-Dimensional Systems,” in Proceedings of the IEEE International Conference on Circuits and Systems, May, 1994, Vol. 4, pp. 227-230.
- [21] J. Ramanujam and P. Sadayappan, “Tiling of Iteration Spaces for multicomputers,” in Proceedings of the International Conference on Parallel Processing, Vol. 2, pp. 179-186, August, 1990.
- [22] M. Sheliga, T. Yu, F. Chen and E. Sha, “Graoph Transformation for Communication Minimization Using Retiming,” in Proceedings of the IEEE International Symposium on Circuits and Systems, June, 1998, Vol. 6, pp. 207-21