Potential and Achievable Parallelism in Unsymmetric-Pattern Multifrontal LU Factorization *

Steven M. Hadfield
Department of Mathematical Sciences
United States Air Force Academy
Colorado Springs, Colorado, USA
phone: (719) 472-4470

Timothy A. Davis
Computer and Information Sciences Department
University of Florida
Gainesville, Florida, USA
phone: (904) 392-1481
email: davis@cis.ufl.edu

Technical Report TR-94-027,
Computer and Information Sciences Department,
University of Florida

April 19, 1994

*This project is supported the National Science Foundation (ASC-9111263, DMS-9223088)

Abstract

The unsymmetric-pattern multifrontal method of Davis and Duff [2] generalizes earlier multifrontal approaches to LU factorization by removing the assumption of a symmetric-pattern of nonzeros in the sparse matrix. As a result, the underlying computational structure becomes a directed acyclic graph (DAG) instead of a tree. This research explores the potential parallelism available in the unsymmetric-pattern multifrontal method using both unbounded and bounded parallelism models based on this DAG. The bounded parallelism model is extended to reflect the performance characteristics of the nCUBE 2 distributed memory multiprocessor to investigate the achievable parallelism. Finally, a factorization-only version of the method is implemented on the nCUBE 2 and its achieved parallelism evaluated.

List of Symbols

r_j - Lower case r with subscripted lowercase j
c_j - Lower case c with subscripted lowercase j
p_j - Lower case p with subscripted lowercase j
a_j - Lower case a with subscripted lowercase j
s_j - Lower case s with subscripted lowercase j
l - Lower case l in italics
d - Lower case d in italics
μ - Lowercase greek letter Mu

Large sparse systems of linear equations frequently arise in many application areas such as circuit design, power systems, oil reservoir modeling, chemical engineering, structural engineering, and computational fluid dynamics [4]. Direct solution methods, as opposed to iterative approximations, are often required for these systems. When the coefficient matrix is symmetric and positive-definite, Cholesky factorization can be efficiently and effectively employed [10]. In the more general case, LU factorization is necessary [4].

While sparsity in the coefficient matrix allows a significant reduction of required computations, it also provides opportunities for the exploitation of parallelism. However, much of the investigation of parallelism has focused on Cholesky factorization [11]. The multifrontal method originally proposed by Duff and Reid [6, 7] has application to both Cholesky and LU factorization [12] and significant parallel potential [3].

With the multifrontal approach, the sparse matrix is decomposed into a set of partially overlapping dense submatrices and a structure is defined for the computations by the relationships between these dense submatrices. While parallelism is available between independent submatrices, there is also parallelism within these dense submatrices and means of exploiting this parallelism have been extensively investigated [8].

Duff and Johnsson [5] explored the available parallelism in the multifrontal method applied to symmetric-pattern matrices using analytical models based on unbounded parallelism. In this work, we explore the both the available and achievable parallelism of a new multifrontal method for the LU factorization of unsymmetric-pattern matrices developed by Davis and Duff [2].

First some of the key concepts of the unsymmetric-pattern multifrontal method are outlined. Then the available parallelism is determined using unbounded parallelism models similar to those of Duff and Johnsson [5]. These models are refined into bounded parallelism models to determine how much parallelism is available on various finite processor sets. Achievable parallelism is then explored using a simulation based on the performance characteristics of the nCUBE 2 distributed memory multiprocessor. Finally, the unsymmetric-pattern multifrontal method is implemented on the nCUBE 2 and its performance assessed.

1 Unsymmetric-Pattern Multifrontal Method

Multifrontal methods for the factorization of sparse matrices [2, 6, 7, 12] decompose the sparse matrix into a set of overlapping dense submatrices called *frontal matrices*. Each frontal matrix contains one or more pivots and is partially factorized according to these pivots. The updated entries in the unfactorized portion of a frontal matrix (called the *contribution block*) are passed and assembled (added) into subsequent frontal matrices. Each contribution block entry must be passed and assembled into one and only one subsequent frontal matrix. The *assembly directed acyclic graph (DAG)* represents frontal matrices as nodes and the passing of contributions blocks as directed edges. With previous multifrontal methods, the assumption of a symmetric-pattern matrix caused the assembly DAG to be a tree (or a forest). The tree structure results since the contribution block of a frontal matrix can be completely absorbed within a single subsequent frontal matrix. When the assumption of symmetry in the pattern is removed, contribution blocks need to be fragmented with the various pieces passed to different frontal matrices.

The assembly DAG also serves as a task graph with the nodes representing the factorization of individual frontal matrices and the edges as inter-task data dependencies. Large grain parallelism is available among independent nodes in the assembly DAG. Furthermore, the factorization of

the dense frontal matrices provides a second level of parallelism because of the independence and regularity in the factorization for each pivot. Earlier efforts have shown that exploitation of both levels of parallelism is necessary for the best performance [3, 5].

2 Unbounded Parallelism Models

The unbounded parallelism models (UPMs) explore the amount of theoretical parallelism available using analytical techniques and assume an unbounded number of available processors. The underlying model of computation is a multiple instruction, multiple data (MIMD) parallel random access machine (PRAM). The memory model is assumed to allow concurrent reads but only exclusive writes (CREW). The models are based on the assembly DAG. Nodes (representing frontal matrices) are weighted according to their predicted parallel factorization time. Three models are presented. Model 1 assumes no parallelism within frontal matrices with only a single processor assigned to each frontal matrix. Model 2 uses a medium grain parallelism within each frontal matrix with a processor assigned to each column of the frontal matrix. Model 3 allows a fine grain parallelism with a processor assigned to each entry of a frontal matrix. The assembly of contribution blocks is represented by edge weights that estimate the time required for assembly. Potential parallelism in the three models is determined by taking the ratio of all node and edge weights over the node and edge weights for the heaviest weighted path through the assembly DAG.

The node weights for a frontal matrix j under these three models are shown in equations (1), (2), and (3). In these models frontal matrix j has r_j rows, c_j columns, p_j pivots, a_j incoming entries to assemble, and s_j frontal matrices from which contributions are received. The four terms in each node weight equation stand for assembly of incoming contributions, numerical checking of each pivot, computation of the multipliers (column of the L factor), and update of the active submatrix, respectively.

$$\text{Model 1: } a_j + \Sigma_{i=1}^{p_j}(r_j - i + 1) + \Sigma_{i=1}^{p_j}(r_j - i) + 2\Sigma_{i=1}^{p_j}(r_j - i)(c_j - i) \tag{1}$$

$$\text{Model 2: } a_j/c_j + \Sigma_{i=1}^{p_j}(r_j - i + 1) + \Sigma_{i=1}^{p_j}(r_j - i) + 2\Sigma_{i=1}^{p_j}(r_j - i) \tag{2}$$

$$\text{Model 3: } \lceil \log_2 s_j \rceil + \Sigma_{i=1}^{p_j} \lceil \log_2(r_j - i + 1) \rceil + p_j + 2p_j \tag{3}$$

In the case of a dense matrix, Model 1 corresponds to using a single processor with $O(n^3)$ execution time, where n is the order of the matrix. Model 2 would use $O(n)$ processors and require $O(n^2)$ time. Model 3 would use $O(n^2)$ processors and require $O(n \log n)$ time.

The four test matrices analyzed are GRE_1107, GEMAT11, SHERMAN5, and RDIST1. They are of orders 1107, 4929, 3312, and 4134, respectively with the number of nonzero entries being 5664, 33108, 20793, and 94408. SHERMAN5 is the only matrix with a near-symmetric pattern. With Model 1, the speedups ranged from only 1.2 to 4.3 indicating little available parallelism within the assembly DAGs themselves. The Model 2 & 3 results are seen as the maximum achieved speedups in Figures 1 and 2, respectively. These results reveal significant potential benefits when parallelism is exploited both within and across frontal matrices.

3 Bounded Parallelism Models

The bounded parallelism models use simulation to determine how much speedup can be achieved on a fixed number of available processors. Only Models 2 and 3 have corresponding bounded parallelism models. Node and edge weights are assigned as they were in the UPM models. Processors sets are defined as powers of 2 from 2^0 to 2^{16} (1 to 65,536). Frontal matrix tasks (corresponding to the nodes) are placed in a ready queue when all of their predecessors have completed execution. Tasks are scheduled from this ready queue based on a heaviest node first priority scheme. Allocation of processors to tasks is nominally one per frontal matrix column in Model 2 and one per frontal matrix entry in Model 3. If a sufficient number of processors are not available, the smaller available set is allocated, unless waiting for the next task to complete and free its processors will result in a sooner completion time. The task's parallel execution time is adjusted per the number of processors allocated.

The speedup results of the bounded parallelism model corresponding to Model 2 and Model 3 are shown in Figures 1 and 2, respectively. These results indicate that the theoretically available parallelism of the unbounded parallelism models can be achieved on reasonably sized processor sets. With the medium grain parallelism of Model 2, almost all of the potential parallelism is achieved with only 512 processors (2^9). The fine grain parallelism of Model 3 was achieved with 65,536 processors (2^{16}).

4 Distributed Memory Performance Model

The distributed memory performance model extends the medium grain bounded parallelism model (Model 2) to reflect the performance characteristics of the nCUBE 2 multiprocessor. Within this model, individual frontal matrices are broken up by columns and distributed to the processors of an assigned subcube in a scattered fashion. The factorization process has the processor that owns the current pivot column compute the multipliers (column of L) and broadcast them to the other processors. All of the processors in the frontal matrix's subcube then update their active columns using these multipliers. This is commonly refered to as a *fan-out* algorithm [8]. Node weights correspond to a predicted parallel execution time. This predicted time is based on an analytical model of the fan-out algorithm with specific parameters set according to results from an implementation and evaluation of this algorithm on the nCUBE 2. Equation (4) describes the time (in μsecs) required for the multipliers' broadcast with l the length of the message in bytes and d the dimension of the subcube.

$$broadcast\_time = 195.22 + 0.122l + 50.02d + 0.45ld \tag{4}$$

Assembly of incoming contributions is assumed to be evenly distributed across the subcube processors. A distinct message is assumed for each edge. The edge weights are based on the size of the contribution block passed on the edge. Point-to-point messages are assumed for passing contributions with the time required for a particular message (in μsecs) determined by Equation (5).

$$direct\_msg\_time = 170.32 + 4.54d + 0.573l \tag{5}$$

The scheduling of tasks is done based on a critical path priority scheme. The *critical path priority* for a particular node (task) is the heaviest weighted path from that node to an exit node (a node

with no out-going edges). Processor allocations are done as subcubes with subcube size based on the size of the frontal matrix. Specific subcube assignments are not tracked and fragmentation of the hypercube is neglected.

The results of this distributed memory performance model are shown in Figure 3. These results indicate that about 25 percent of the theoretically available parallelism should be achieved on the nCUBE 2. This is consistent with similar results for Cholesky factorization [13].

5 Implementation Results

Within the implementation, factorization of the frontal matrices is done using a pipelined, fan-out algorithm similar to that used in the distributed memory model [9]. With pipelining, however, the processor owning the next pivot column will update only that column and compute and send the next pivot's multipliers before doing the rest of the updates for the current pivot. This allows the communication to be overlapped with computation. Experiments revealed that about 35 percent of the communication is effectively overlapped by using pipelining. The Basic Linear Algebra Subroutines - Level 1 (BLAS-1) are used for the bulk of the computations required for factorization. These routines are about four times faster than the C-language code upon which the distributed memory model is based. While the inclusion of pipelining improved parallelism, use of the BLAS-1 routines reduced the ratio of computation to communication and the combined effect was to reduce the parallelism achieved by frontal matrix factorization. (Of course overall execution times were significantly improved).

Task scheduling for the implementation was done using critical path priorities. Processor allocation was done using a proportional scheme that favored inter-frontal matrix parallelism (between nodes) over intra-frontal matrix parallelism (within nodes). Specific subcubes were assigned to maximize overlapping of communicating frontal matrix tasks. Subcube management was done with a variant of the binary buddy system. Frontal matrix columns were assigned to processors in either a scattered or blocked format depending upon which produced the best overall execution time.

The speedups achieved by this implementation are shown in Figure 4. Typically, they are less than that predicted by the distributed memory model but this is mainly attributed to the effects of using the faster BLAS-1 routines.

6 Conclusion

The primary conclusion of this research is that the unsymmetric-pattern multifrontal method has significant potential for parallelism. The unbounded parallelism models quantified this potential parallelism and the bounded parallelism models indicated that the parallelism was fully achievable on reasonably sized processor sets. The distributed memory model revealed that about 25% of the available parallelism is achievable on contemporary architectures. Finally, the results of these models were validated by an actual implementation of the method on the nCUBE 2.

Importantly, the assembly DAGS used in this effort were produced using a sequential formulation of the unsymmetric-pattern multifrontal method [1]. Parallelism was not a specific objective in their derivation. Alternative formulations of these assembly DAGs are possible and could reveal significant additional parallelism. Such alternatives are currently under investigation.

References

[1] T. A. Davis. Users' guide for the unsymmetric-pattern multifrontal package (UMFPACK). Technical Report TR-93-020, Computer and Information Sciences Department, University of Florida, Gainesville, FL, June 1993. To obtain UMFPACK, send email to `netlib@ornl.gov` with the message: `send umfpack.shar from misc`.

[2] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for parallel sparse LU factorization. Technical Report TR-93-018, Computer and Info. Sci. Dept., University of Florida, Gainesville, FL, 1993.

[3] I. S. Duff. Parallel implementation of multifrontal schemes. *Parallel Computing*, 3:193–204, 1986.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications, New York, NY, 1989.

[5] I. S. Duff and L. S. Johnsson. Node orderings and concurrency in structurally-symmetric sparse problems. In Graham F. Carey, editor, *Parallel Supercomputing: Methods, Algorithms, and Applications*, pages 177–189. John Wiley and Sons Ltd., New York, NY, 1989.

[6] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. on Math. Software*, 9:302–325, 1983.

[7] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric set of linear equations. *SIAM J. of Sci. and Stat. Computing*, 5(3):633–641, 1984.

[8] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh. Parallel algorithms for dense linear algebra computations. In R. J. Plemmons, editor, *Parallel Algorithms for Matrix Computations*, pages 1–82. SIAM, Philadelphia, PA, 1990.

[9] G. A. Geist and M. Heath. Matrix factorization on a hypercube. In M. Heath, editor, *Hypercube Multiprocessors 1986*, pages 161–180. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[10] A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive-Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[11] Michael T. Heath, Esmond Ng, and Barry W. Peyton. Parallel algorithms for sparse linear systems. In R. J. Plemmons, editor, *Parallel Algorithms for Matrix Computations*, pages 83–124. SIAM, Philadelphia, PA, 1990.

[12] J. W. H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Review*, 34(1):82–109, 1992.

[13] L. S. Ostrouchov, M. T. Heath, and C. H. Romine. Modeling speedup in parallel sparse matrix factorization. Technical Report ORNL/TM-11786, Oak Ridge National Laboratory, Oak Ridge, TN, 1990.

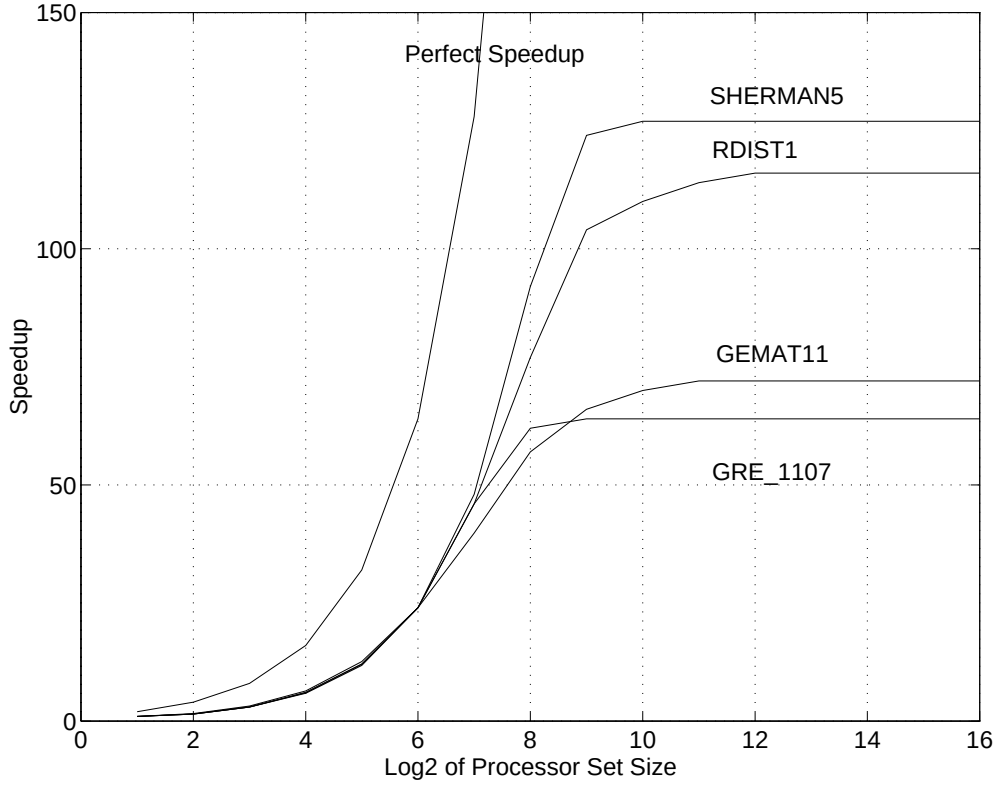

Figure 1: Model 2 Results (Medium Grain Parallelism)

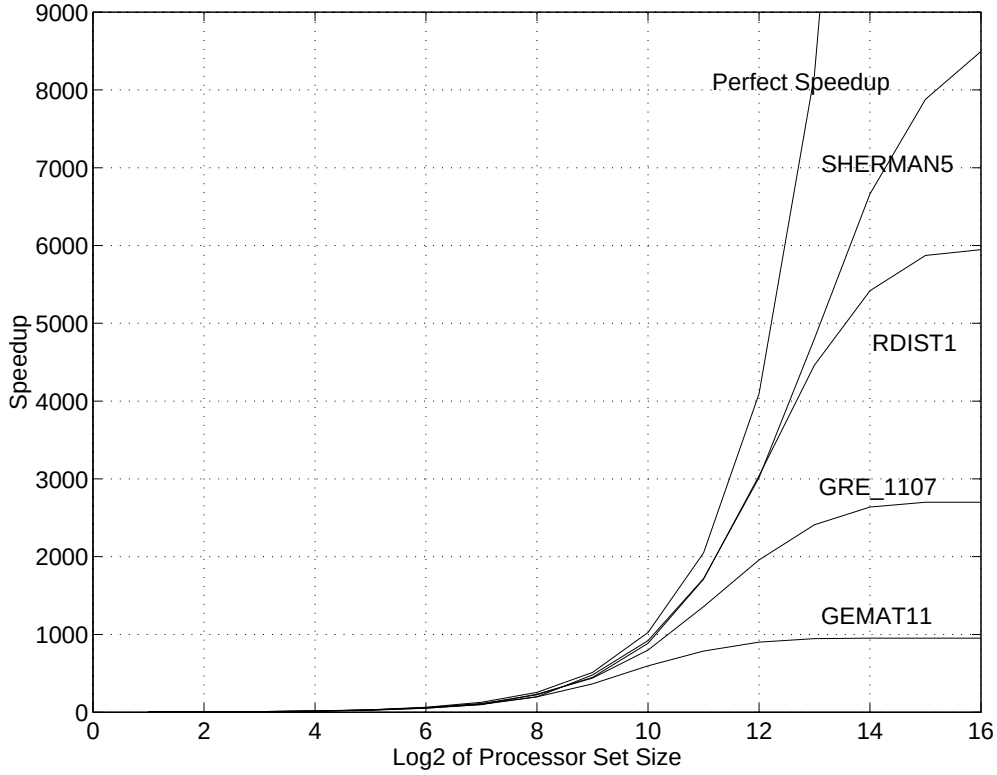

Figure 2: Model 3 Results (Fine Grain Parallelism)

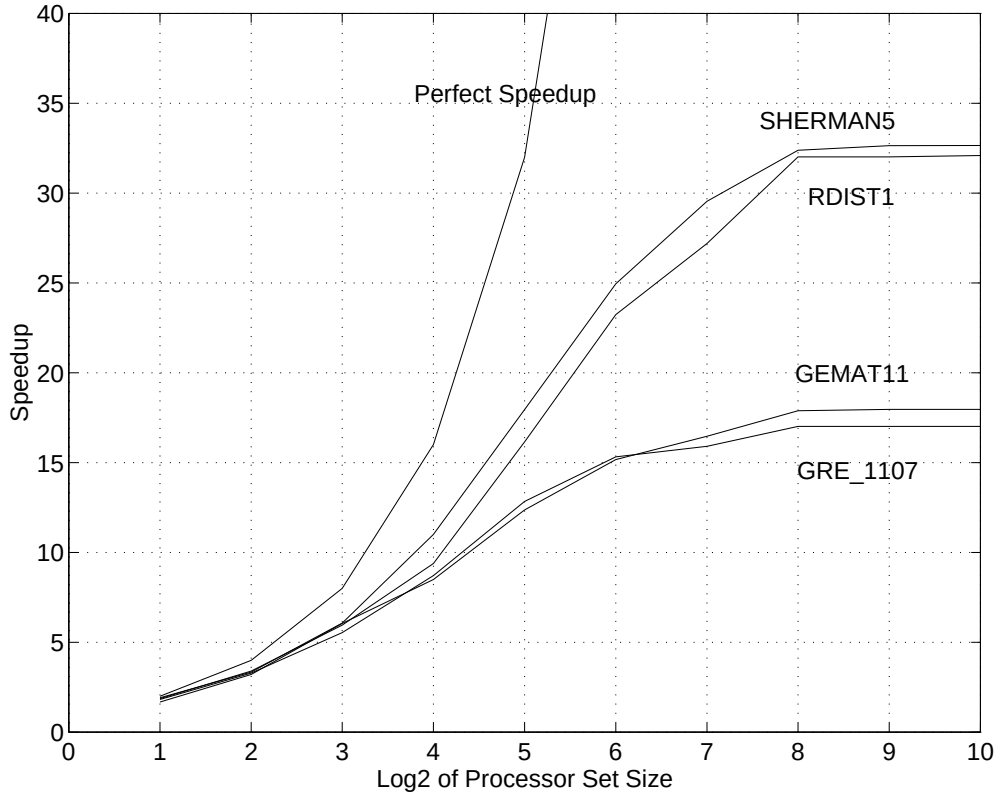

Figure 3: Distributed Memory Model Results

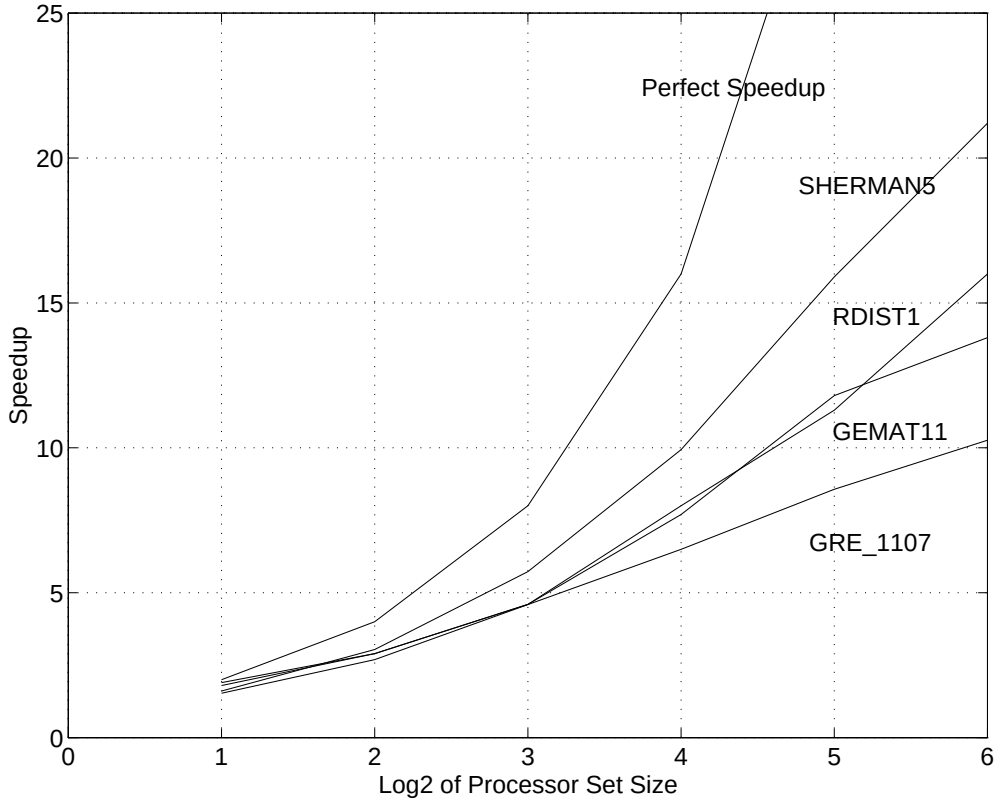

Figure 4: Implementation Results