Yale University
Department of Computer Science

Node Orderings and Concurrency in Structurally-Symmetric Sparse Problems

I.S. Duff and S.L. Johnsson

YALEU/DCS/TR-724
July 1989

Node orderings and concurrency in structurally-symmetric sparse problems

I. S. Duff and S. L. Johnsson*

ABSTRACT

In the solution of structurally-symmetric sparse linear systems by direct methods it is possible to exploit concurrency not only within the elimination operations at a single pivot step but also over the eliminations using different pivots. The pivot ordering that minimizes the number of arithmetic operations does not, in general, minimize the time for the concurrent solution of a system of equations. We investigate minimum degree and nested dissection orderings, as well as a few other ordering schemes, with respect to potential solution time and total arithmetic complexity for a few benchmark problems and an idealized parallel computer. The benchmark problems include 2-dimensional grid problems and some other standard sparse matrix test problems.

---

*Departments of Computer Science and Electrical Engineering, Yale University, New Haven, CT 06520, USA.

The original work was presented at the first International Conference on Vector and Parallel Computing held at Loen in Norway in June 1986.

To appear in *Parallel Supercomputing: Methods, Algorithms and Applications*. Edited by Graham Carey. John Wiley & Sons Ltd., Chichester and New York.

Keywords

Gaussian elimination, multifrontal methods, elimination trees, sparse orderings, parallelism, multiprocessing, minimum degree, nested dissection, sparse matrices.

Computer Science and Systems Division,
Harwell Laboratory,
Oxon OX11 0RA.

July 1989.

CONTENTS

1	Introduction	1
2	Measures of achievable parallelism	2
3	Orderings	5
4	Results	6
5	Conclusions	9

1 Introduction

The solution of structurally-symmetric sparse linear equations by direct methods offers an additional potential for concurrency over dense matrix problems. In this paper we estimate the available concurrency due to sparsity, and compare it with the concurrency that is available in the elimination within each pivot step. Johnsson (1985b) refers to these two forms of concurrency as concurrency in the elimination of multiple vertices (CEMV), and concurrency in the elimination of single vertices (CESV), although we will use alternative terms defined in Section 2 in this paper. In this paper we assume unbounded parallelism and, under this assumption, consider the potential speed-up both due to CEMV and CESV for some benchmark sparse matrix problems.

We represent the factorization by an elimination tree in which each node represents the elimination operations corresponding to a single pivot step and edges of the tree represent dependencies between different steps. This tree can be viewed as a computational graph for the solution process in the sense that operations at a node of the graph require information from the sons of the node. The tree thus defines a partial ordering; work corresponding to all leaf nodes can proceed immediately and concurrently and work corresponding to any other node can commence once the work corresponding to its sons is complete. An elimination tree thus defines a dependency graph in computer science terms. Duff (1986) discusses the use of the elimination tree in implementing a parallel elimination scheme. Here we use the tree to study various measures of parallelism and compare some of our measures with those obtained on real systems. An elimination tree can be uniquely constructed for any given ordering. However, any ordering of the nodes of the tree which respects the dependencies is valid and involves the same number of floating-point operations.

The computations at each node include the assembly of small full matrices from the sons together with information on the pivot row and column corresponding to the node. The pivoting operations are then performed on the assembled small full matrix, called a frontal matrix, and the non-pivot rows and columns are sent to the father node. The rows and columns of the small full matrices must be identified by integer index vectors and indirect addressing operations are required in the assembly phase.

Although we have defined the nodes of the elimination tree in terms of a single pivot, in practice we amalgamate nodes, so that a single tree node may correspond to several steps of Gaussian elimination. If the rows and columns in the son are a subset of those in the father then there is no loss in sparsity by amalgamating father and son pivots into the same tree node and performing both sets of eliminations at the single node. This node amalgamation can assist both vectorization and parallelism within the node, and we assume that it is done. Indeed Duff and Reid (1983) suggest performing additional node amalgamations at the cost of increasing slightly the overall work and storage, but we do not perform such amalgamations here. Node amalgamation also facilitates numerical pivoting since pivots can be chosen from anywhere within the rows and columns corresponding to the intended pivots. In the following, we do not assume that the matrix is diagonally dominant or symmetric and positive definite, and we include operation counts for pivoting operations.

We describe our model for calculating parallelism in Section 2 and the various orderings with which we experiment in Section 3. The results of our numerical experiments are given in Section 4. Finally, we present some concluding remarks in Section 5.

2 Measures of achievable parallelism

In assessing the potential speed-up from different schemes for parallelizing sparse matrix factorization, we consider only arithmetic and logic operations. Memory conflicts, bus or network delays, and the scheduling of a limited number of processing units are ignored. The reference measure we use for determining the available speed-up is simply the total number of operations for the sequential factorization, counting additions, multiplications, divisions, and comparisons equally. We require comparisons for the selection of pivots within the frontal matrix since we do not assume positive definiteness or diagonal dominance. We also include a count for the assembly of the node. For each node ordering we use the elimination tree to define measures of parallelism. If a quantity is defined at the nodes of the tree then we can sum this quantity over all nodes in the path from a leaf node to the root. The maximum over all leaf nodes of such sums occurs for the so-called "longest path". For appropriate quantities, this gives a measure of the sequential work required.

To assist in describing our different measures of parallelism, we sketch the Gaussian elimination procedure in Figure 2.1, where we have numbered the three basic loops in the elimination, noting that the overall number of executions of loop (2) is precisely the order of the matrix (the total number of pivots).

```
For all nodes in an order respecting dependencies                (1)
    For each Gaussian elimination step at node                   (2)
        Select pivot
        Compute multipliers
        For all non-pivot rows in frontal matrix                 (3)
            Update entries in rows
```

Figure 2.1. Sketch of tree-based Gaussian elimination.

The first measure of parallelism is obtained only from the sparsity of the problem (corresponding to loop (1) in Figure 2.1). Here we assume that the operations at any one node must be done sequentially. The operations at each node comprise arithmetic operations, comparisons for pivoting, and assembly operations. The total sequential count is obtained by summing these over all nodes of the tree. The measure of parallelism is obtained by dividing the total sequential count by the maximum sum of number of operations along any path from a leaf node to the root of the tree. Since this parallelism is at the outermost level of the triply nested Gaussian elimination loop, we call this measure "outer" in our numerical experiments in Section 4.

The next measure of parallelism is obtained from allowing full parallelism within the operations at a node. A parallel algorithm is used for pivot selection. This requires a number of comparisons that is logarithmic in the size of the frontal matrix. The calculation of the multipliers can be performed concurrently as can the rank-1 update in loops (3). Parallelizing the assembly process is considerably

simplified if the programming language includes a concurrent write instruction with addition, which uses hardware synchronization for accumulation. The Ultracomputer (Schwartz 1980), the RP3 (Pfister *et al.* 1985), the Connection Machine (Hillis 1985) and the Fluent (Ranade *et al.* 1988) are examples of architectures that offer such support. We assume this model and parallelize the assembly accordingly. We call this measure "2 inner".

In architectures with a significant overhead for parallelization of operations at such a fine level of granularity, an aggregation of operations may be desirable. For this reason, we also compute a measure of parallelism for sequential pivot selection, followed by calculating the multiplier and updating each row in serial mode but performing the updates of different rows in parallel. The assembly is also performed sequentially within a row, but concurrently within a column. We call this measure of parallelism for a node "1 inner" in Section 4.

Assuming that sufficiently many processors are available, different nodes of the elimination tree can be eliminated concurrently, as long as the dependency rules are not violated, in combination with concurrent elimination within individual nodes. We compute the parallelism that can be achieved by combining concurrent elimination of tree nodes, "outer ", with "1 inner" concurrency at the nodes, and call this measure "outer + 1 inner". We also determine the maximum parallelism attainable by concurrent elimination of tree nodes and maximally exploiting concurrency in each node elimination. We call this measure "outer + 2 inner".

In the formulae on the following page, we give the precise definitions of how the different complexity measures are computed and the corresponding measures of parallelism. The total number of pivots is n, that is the entire sparse matrix is of size $n \times n$. The number of nodes in the elimination tree is N_T, and the number of pivots in node j is P_j. The size of the frontal matrix at node j is $K_j \times K_j$. The path from the leaf node l to the root is identified by $path_l$. The number of additions for assembly of son i at node j is denoted $AA_j(i)$, and the number of sons of node j is S_j.

In the complexity expressions in these formulae, the term within the square brackets defines the number of operations required for one node of the elimination tree under the different schemes for exploiting parallelism. The first term in the complexity expression for a node represents the number of additions in sequence that are necessary for the assembly, the second term the number of comparisons in sequence for the pivot selection, the third term the number of arithmetic operations for the calculation of the multipliers, and the last term the number of sequential operations for the rank-1 update.

Note that in determining the complexity measures C_O, C_{O1}, and C_{O2} the maximum is sought for different expressions, and the selected paths may be different. The paths with the widest frontal matrices are likely to benefit the most from parallelizing one or both " inner" loops.

The complexity estimates for the different modes of parallelism are

(i) Sequential:

$$C_s = \sum_{j=1}^{N_T} \left[\sum_{i=1}^{S_j} AA_j(i) + \sum_{i=1}^{P_j} (K_j - i + 1) + \sum_{i=1}^{P_j} (K_j - i) + 2 \sum_{i=1}^{P_j} (K_j - i)^2 \right]$$

$$C_s = \sum_{j=1}^{N_T} \left[\sum_{i=1}^{S_j} AA_j(i) + P_j(1 + 2K_j(K_j+1) - \frac{2}{3}(P_j+1)(3K_j - P_j + 1)) \right]$$

(ii) Outer:

$$C_O = \max_l \sum_{j \in path_l} \left[\sum_{i=1}^{S_j} AA_j(i) + \sum_{i=1}^{P_j} (K_j - i + 1) + \sum_{i=1}^{P_j} (K_j - i) + 2 \sum_{i=1}^{P_j} (K_j - i)^2 \right]$$

$$C_O = \max_l \sum_{j \in path_l} \left[\sum_{i=1}^{S_j} AA_j(i) + P_j(1 + 2K_j(K_j+1) - \frac{2}{3}(P_j+1)(3K_j - P_j + 1)) \right]$$

(iii) 1 inner:

$$C_1 = \sum_{j=1}^{N_T} \left[\sum_{i=1}^{S_j} \sqrt{AA_j(i)} + \sum_{i=1}^{P_j} (K_j - i + 1) + P_j + 2 \sum_{i=1}^{P_j} (K_j - i) \right]$$

$$C_1 = \sum_{j=1}^{N_T} \left[\sum_{i=1}^{S_j} \sqrt{AA_j(i)} + \frac{1}{2} P_j(6K_j - 3P_j + 1) \right]$$

(iv) 2 inner:

$$C_2 = \sum_{j=1}^{N_T} \left[\lceil \log_2 S_j \rceil + \sum_{i=1}^{P_j} \lceil \log_2 (K_j - i + 1) \rceil + 3P_j \right]$$

(v) Outer + 1 inner:

$$C_{O1} = \max_l \sum_{j \in path_l} \left[\sum_{i=1}^{S_j} \sqrt{AA_j(i)} + \sum_{i=1}^{P_j} (K_j - i + 1) + P_j + 2 \sum_{i=1}^{P_j} (K_j - i) \right]$$

$$C_{O1} = \max_l \sum_{j \in path_l} \left[\sum_{i=1}^{S_j} \sqrt{AA_j(i)} + \frac{1}{2} P_j(6K_j - 3P_j + 1) \right]$$

(vi) Outer + 2 inner:

$$C_{O2} = \max_l \sum_{j \in path_l} \left[\lceil \log_2 S_j \rceil + \sum_{i=1}^{P_j} \lceil \log_2 (K_j - i + 1) \rceil + 3P_j \right]$$

When deciding how to measure speed-ups in the parallel implementation of an algorithm, two base measures are used. One is the performance of the same algorithm on one processor, and the other is the time for the best known sequential algorithm for performing the same computation. The latter measure is what is important in practice. The former provides insight into how a given ordering performs when the number of processors is increased. We use this measure in most of our tables, but the latter measure is used occasionally for reference.

3 Orderings

Ordering strategies that attempt to minimize storage or the total amount of arithmetic performed during elimination have been investigated extensively. For general graphs the two most common ordering strategies are minimum degree and nested dissection. Nested dissection is of optimal order with respect to fill-in for graphs with good separators, such as planar graphs. However, the minimum degree ordering yields competitive orderings for many graphs and is often better than nested dissection at reducing fill-in and arithmetic (for example, George and Liu 1981, Duff, Erisman, and Reid 1986). An ordering that is optimal with respect to storage or arithmetic is not necessarily optimal with respect to parallel complexity. With unbounded parallelism and a symmetric positive definite problem, an ordering that minimizes the maximum number of pivots along any path in the tree is optimal with respect to solution time. If each node represents a single pivot step, then an ordering that minimizes the tree height (that is the number of nodes on the longest path from a leaf node to the root) is optimal with respect to solution time, for unbounded parallelism.

A good example of the fact that an ordering that is optimal with respect to fill-in is far from optimal for any reasonable model of parallel computation is the solution of tridiagonal systems. For such systems orderings without fill-in exist, but they allow little parallelism. Cyclic reduction (or, equivalently for this case, nested dissection) yields a balanced tree with only $\log_2 n$ levels and much potential for parallelism for the solution of a system of order n. However, this ordering causes fill-in and about doubles the number of floating-point operations. The cyclic reduction algorithm eliminates the maximum number of variables at each stage. However, such algorithms are in general not optimal with respect to solution time even if the parallelism is unbounded (Johnsson 1985a). Furthermore, as will be demonstrated in Section 4, an ordering with a lower maximum number of pivot steps along any path from a leaf node to the root of the elimination tree than another ordering does not necessarily have the lowest maximum number of arithmetic operations along any path in the two elimination trees (and certainly not the total amount of arithmetic, as is the case for cyclic reduction).

In addition to the minimum degree and nested dissection orderings, we also consider orderings that are specifically designed to enhance the parallelism in the tree. Since the height of the tree strongly affects the amount of sequential computation required, we try an ordering that attempts to minimize the tree height. For this ordering we represent the matrix by a supervariable graph where each vertex can represent one or more variables (see, for example, Duff and Reid 1983). Pivots corresponding to variables in a supervariable will be eliminated at the same node of the elimination tree with amalgamated nodes. The ordering procedure starts by setting the depth of all vertices in the supervariable graph to 1, selecting one of these vertices (using minimum degree as a tie-breaker), and assigning a depth of two to all vertices to which this vertex is connected. We continue choosing

vertices of depth 1 until all are exhausted and then search for vertices of depth 2 and so on. In each case, uneliminated vertices adjacent to vertices being eliminated are assigned a depth equal to the maximum of their current depth and one more than the vertex being eliminated. As in the case of the minimum degree ordering, the heuristic just described is local and will not necessarily minimize the tree height over all orderings, although, as we see from our results in Section 4, it usually does a very effective job.

The use of the elimination tree height as a measure of parallelism is rather crude, since the assumption is that all nodes are equal. A simple extension is to account for amalgamation and weight a node by its number of pivots. The ordering proceeds by setting the depth of a vertex to the maximum of its current depth and the depth of the vertex being eliminated plus the number of variables in the supervariable at that vertex. We study the effect of this modification in Section 4.

On the regular grid problems from a five-point discretization of the Laplacian operator, using one step of elimination according to a red-black ordering followed by, for instance, a minimum degree or a nested dissection ordering may yield a better result than either ordering alone, because we will have maximized the number of leaf nodes in the elimination tree. We also include this one-step greedy ordering for the grid problems. Note that the first step of the minimum depth ordering may give a red-black ordering depending on the order selected by the minimum degree tie-breaker.

4 Results

The (partial) orderings of the variable eliminations in the sample problems have been obtained using general ordering routines. The minimum degree ordering routines are from the Harwell MA37 package (Duff and Reid 1984) and the nested dissection routines from Sparspak (George and Ng 1984). Our minimum height heuristic was obtained from a minor modification to the minimum degree code of MA37. The sample problems are chosen to represent both regularly structured (five-point discretization of the Laplacian on a two-dimensional grid) and less structured examples from the Harwell-Boeing test collection (Duff, Grimes, and Lewis 1989).

	30×30	10×100	LUNDA	ERIS1176	BCSSTK24
Outer	2.4	2.1	1.9	1.3	2.7
Outer + 1 inner	46	18	25	40	310
1 inner	12	6	12	20	81
2 inner	65	25	80	88	2093
Outer + 2 inner	421	104	173	468	12375

Table 4.1. Speed-ups due to various levels of parallelism on our ideal machine. Minimum degree ordering used throughout.

For $N \times N$ and $N \times N \times N$ grid problems the number of arithmetic operations at the root of the elimination tree is of the same order as the total number of operations. It follows that the parallelism that can be achieved from concurrent elimination of different pivots is $O(1)$. Thus exploiting sparsity

alone for parallelism may yield very limited speed-ups. But the speed-up from exploiting concurrency in one inner loop is $O(N)$ for a two dimensional problem and $O(N^2)$ in three dimensions. If concurrency is obtained over both inner loops the speed-ups are $O(N^2/\log_2 N)$ and $O(N^4/\log_2 N)$ in two and three dimensions, respectively. For a problem of order n that can be represented as a banded matrix with bandwidth $m \ll n$ the speed-up is $O(\frac{n/m}{\log_2(n/m)})$ from exploiting concurrency in the outer loop, while one inner loop yields a speed-up of $O(m)$ and both inner loops a speed-up of $O(m^2/\log_2 m)$. The results in Table 4.1 confirm that for the test cases only a small speed-up can be obtained from sparsity alone. When combined with sparsity within the nodes a more encouraging speed-up is obtained, as expected. The speed-up for one inner loop is always greater than the relative speed-up from parallelizing also the second loop, and sometimes significantly so. The concurrency in sparse elimination is substantial even in problems of small to moderate size, if all three loop levels are parallelized. Although the full "outer + 2 inner" figure may be difficult to obtain on an actual machine, the target can be viewed as an attractive goal for machines designed to work well at that fine a granularity.

Ordering		MD	ND	GMD1	GMD2	RB-MD	RB-ND
Height		24	13	11	11	20	12
Total ops	($\times 10^3$)	415	514	894	1046	472	457
Counts on longest path							
Number of pivots		124	80	118	112	119	76
Number operations	($\times 10^3$)	172	129	415	604	194	123
Row ops	($\times 10^3$)	9	6	14	16	9	6
Elim ops		985	643	1008	977	951	613

Table 4.2. Statistics for 30×30 grid problem.

In Table 4.2, the orderings are designated by MD (minimum degree), ND (nested dissection), GMD1 and GMD2 for the minimum height algorithms using tree height and weighted tree height respectively, and RB-MD and RB-ND for red-black followed by minimum degree and nested dissection, respectively. The good performance of minimum degree as a sequential ordering for reducing the operation count is illustrated in Table 4.2, where it gives the lowest total operation count among the orderings used. The operations count includes arithmetic operations for assembly, pivoting, and elimination. The number of operations in sequence for parallel assembly, parallel pivoting, and parallel elimination in the case of parallelization by "Outer + 2 inner" is labelled "Elim ops" in the table. The nested dissection ordering has a fairly good performance as a sequential ordering and its power as an ordering scheme for parallel elimination is seen in the last three rows of Table 4.2. The combination with a red-black ordering gives some improvement in the parallel orderings, although RB-MD is worse as a sequential ordering than minimum degree by itself. The two minimum height orderings reduce the height well, but even GMD2 is not as good as nested dissection at reducing the number of pivots on the longest path. Indeed GMD2 shows no real advance over GMD1 although it yields slightly fewer operations at the finest level of granularity. For this regular grid problem neither of them are as successful as nested dissection (or RB-ND) for exploiting parallelism.

Ordering	MD	ND	GMD1	GMD2	RB-MD	RB-ND
Outer	2.4	4.0	2.1	1.7	2.4	3.4
Outer + 1 inner	46	80	66	67	50	75
1 inner	12	13	20	23	13	13
2 inner	65	79	138	161	74	71
Outer + 2 inner	421	799	887	1070	496	746
Relative to best ordering						
Outer + 1 inner	46	69	29	26	46	69

Table 4.3. Speed-ups for 30×30 grid problem.

Most of the speed-ups in Table 4.3 are relative to the sequential operations for the same ordering. The speed-up relative to the best sequential ordering is included for the "outer + 1 inner" ordering in the last row of the table, where we show the speed-ups relative to the total number of operations for the minimum degree ordering, the best sequential ordering. The better parallelization properties of nested dissection mean that it may be the best algorithm on a parallel machine, even if the number of sequential operations is higher than for the minimum degree ordering. This is evident from the last row of Table 4.3. It is clear that nested dissection (or RB-ND) is the best parallel ordering in spite of its poorer sequential performance.

Ordering		MD	ND	GMD1	GMD2
Height		50	66	12	17
Total ops	($\times 10^3$)	618	873	645	636
Counts on longest path					
Number of pivots		175	111	103	82
Number operations	($\times 10^3$)	478	362	356	349
Row ops	($\times 10^3$)	16	12	11	10
Elim ops		1319	947	841	692

Table 4.4. Statistics for ERIS1176 matrix from test collection.

For the less regular problem from circuit analysis, the results in Tables 4.4 and 4.5 show a different performance to that for the regular grid problem. The minimum degree ordering is still the best sequential ordering, but nested dissection is now much poorer as a sequential ordering although it still beats minimum degree at exploiting sparsity. However, the minimum height algorithms are more effective here at reducing tree height and the number of pivots on the longest path, and GMD2 is the best ordering for the parallel operation counts, in spite of producing a tree with greater height than GMD1.

Ordering	MD	ND	GMD1	GMD2
Outer	1.3	2.4	1.8	1.8
Outer + 1 inner	40	75	61	65
1 inner	20	21	20	20
2 inner	88	118	92	91
Outer + 2 inner	468	922	767	912
Relative to best ordering				
Outer + 1 inner	40	52	56	62

Table 4.5. Speed-ups for ERIS1176 problem.

5 Conclusions

In a multiprocessor architecture with few processing elements relative to the number of variables in the problem an "optimum" ordering is likely to be "close" to an ordering that is desirable for a sequential machine, even if communication complexity is included. For an architecture in which the number of processing elements is of an order comparable to the number of variables, the parallelism with respect to concurrent elimination of different pivots is bounded only for the first few levels of the elimination tree. With the number of processing elements falling in the intermediate range there is a choice of exploiting various combinations. These findings are in broad agreement with the work of Duff, Gould, Lescrenier, and Reid (1987) who examined orderings on symmetric and positive-definite finite-element problems.

Our results in Section 4 confirm that the nested dissection ordering is very effective for exploiting parallelism in regular grid problems, but our new orderings based on minimizing the tree height show great promise for less regular problems.

In conclusion, there is much scope for exploiting concurrency in sparse elimination for any ordering, although it is important to utilize the parallelism both within the elimination tree nodes as well as across the nodes. If fine granularity working, at the level of a couple of arithmetic operations, is feasible, extremely high parallelism is available.

References

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct Methods for Sparse Matrices.* Oxford University Press, London.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1987). Sparse matrix test problems. *ACM Trans. Math. Softw.* 15, 1-14.

Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw.* 9, 302-325.

Duff, I. S. and Reid, J. K. (1984). The multifrontal solution of unsymmetric sets of linear systems. *SIAM J. Sci. Stat. Comput.* 5, 633-641.

Duff, I. S., Gould, N. I. M., Lescrenier, M., and Reid, J. K. (1987). The multifrontal method in a parallel environment. Report CSS 211, Computer Science and Systems Division, Harwell Laboratory.

George, A. and Liu, J. W. H. (1981). *Computer solution of large sparse positive-definite systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

George, A. and Ng, E. (1984). SPARSPAK : Waterloo sparse matrix package user's guide for SPARSPAK-B. CS-84-37, Department of Computer Science, University of Waterloo, Ontario, Canada.

Hillis, W. D. (1985). *The Connection Machine.* MIT Press, Cambridge, Massachusetts.

Johnsson, S. L. (1985a). Cyclic reduction on a binary tree. *Computer Physics Communications* 37, 195-203.

Johnsson, S. L. (1985b). Fast banded systems solvers for ensemble architectures. Report YALEU/DCS/RR-379, Department of Computer Science, Yale University, Connecticut.

Pfister, G. F., Brantley, W. C., George, D. A., Harvey, S. L., Kleinfelder, W. J., McAuliffe, K. P., Melton, E. A., Norton, V. A., and Weiss, J. (1985). The IBM Research Parallel Processor Prototype (RP3): Introduction and architecture. In *Proceedings of the 1985 International Conference on Parallel Processing, IEEE Computer Society, 764-771.*

Ranade, A. G., Bhatt, S. N., and Johnsson, S. L. (1988). The Fluent abstract machine. In *Advanced Research in VLSI. Proceedings of the fifth MIT VLSI Conference.* MIT Press, Cambridge, Massachusetts, 71-93.

Schwartz, J. T. (1980). Ultracomputers. *ACM Trans. Program. Lang. Syst.* 2, 484-521.