ularly for the distributed networks, are meant only as a rough guide. Our heuristics have been partially validated by independent static scheduling packages [11], but further work is needed to compare simluation results with actual multifrontal timings on exisiting architectures. Although we have attempted to model realistic situations, we have not explored all possible considerations; the field of distributed direct solvers for large sparse systems is still developing and contains various areas of active research, such as generating good elimination trees for parallel execution, finding efficient partitioning/clustering hueristics for the processor- assignment problem, and dealing with scattered input matrices. We hope that by careful performance modeling and experimental testing we can better understand these subtle issues.

7 Acknowledgments

The author wishes to thank Iain Duff and Patrick Amestoy for numerous discussions on the multifrontal method and their helpful comments and suggestions which have improved the presentation of this paper. This study was conducted during a visiting appointment to CERFACS in Toulouse, France.

References

1. P. R. Amestoy, *MUPS: Subroutine Library Specification*, CERFACS, June 1991.
2. J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Host, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, 1991.
3. I. S. Duff, "Parallel implementation of multifrontal schemes," *Parallel Computing*, North-Holland, Vol. 3, pp. 193-204, 1986.
4. I. S. Duff, R. G. Grimes, J. G. Lewis, "Sparse Matrix Problems," *ACM Trans. Math. Softw.*, Vol 14, pp. 1-14, 1989.
5. I. S. Duff, L. S. Johnson, "Node Orderings and Concurrency in Structurally-Symmetric Sparse Problems," *Parallel Supercomputing: Methods, Algorithms, and Applications*, John Wiley & Sons Ltd., 1989.
6. I. S. Duff, J. K. Reid, "The Multifrontal Solution of Unsymmetric Sets of Linear Equations," *SIAM J. Sci. Stat. Comput.*, Vol. 5, No. 3, pp 633-641.
7. M. T. Heath, E. Ng, B. W. Peyton, "Parallel Algorithms for Sparse Linear Systems," *SIAM Review*, Vol. 33, No. 3, pp 420-460, 1991.
8. R. Lucas, T. Blank, J. Tiemann, "A Parallel Solution Method for Large Sparse Systems of Equations," *IEEE Transactions on Computer-Aided Design,* Vol. CAD-6, No. 6, November 1987, pp. 981-991.
9. L. S. Ostrouchov, M. T. Heath, C. H. Romine, "Modeling Speedup in Parallel Sparse Matrix Factorization," Technical Report ORNL/TM-11786, Oak Ridge National Laboratory, Oak Ridge, TN, 1990.
10. R. Pozo, "Performance Modeling of Parallel Architectures for Scientific Computing", Ph. D. Thesis, University of Colorado, Boulder, 1991.
11. Yang and A. Gerasoulis, "PYRROS: Static scheduling and code generation for message passing multiprocessors," *Proc. of 6th ACM Inter. Conf. on Supercomputing*, Washington D.C., July, 1992.

This article was processed using the LaTeX macro package with LLNCS style

Table 6. Execution times (in seconds) due to the combined node and elimination tree parallelism.

Simulated Architecture	LA70x70	LA200x200	BCCSTK24	BCCSTK33	LNS3937	NNC1374
Single Cray-2	0.22	7.73	0.71	23.05	0.20	0.01
Suns/Ethernet	16.01	255.77	43.56	768.01	23.42	0.55
iPSC/2	1.95	20.24	4.23	61.31	2.71	0.20
SPARC-2/Ethernet	3.65	79.30	12.97	366.69	3.14	0.14
iPSC/860	0.83	10.32	1.94	30.89	1.16	0.04
RS/6000 LAN	0.33	4.89	0.86	14.01	0.48	0.02
SPARCS/HIPPI	0.50	4.12	0.97	10.77	0.69	0.04
Crays/HIPPI	0.07	1.01	0.18	2.83	0.10	0.00

distributed systems with thousands of processors.

One source of difficulty lies in the imbalance of elimination trees for sparse matrices with irregular patterns. By comparison, one typically sees better performance for similar algorithms on structured problems [8][9]. The coarse-grained parallelism offered by the elimination tree of irregular patterns is rather poor (less than 4 in our examples) and the significant communication costs on distributed architectures limit the amount of parallelism available to decompose single frontal factorization into finer grained tasks. It should also be noted that the multifrontal method is already highly efficient on a single processor, since it employs dense linear algebra in its computational kernel. This efficient computation section, however, also serves to make the communication bandwidth limitation even more visible.

Another difficulty arises from the need to explicitly deal with task mapping and scheduling issues. In shared-memory environments, such concerns can be handled quite effectively by simple task-queue scheduling techniques. In distributed memory environments with considerably higher communication costs and fragmented memory pools, such approaches are not as effective. In practice, a static mapping, predetermined scheduling, or some hybrid static-dynamic mapping is required before the numerical factorization begins. Such mapping problems have been shown to be NP-complete and can be quite expensive to calculate, although various hueristics exist for suboptimal solutions. Nevertheless, this computation is also part of solving the problem and must also be properly accounted for in the overall execution time. In our study, we have not addressed how the sparse matrix is generated or its initial distribution within the parallel machine. This is a critical issue but impossible to study in isolation, since it requires an understanding of how the matrix data is distributed by the driving application.

Finally, we should remind the reader that these conclusions are preliminary and the results from these simulations are not meant to be definitive. Some of the architecture representations are, at best, reasonable estimates; execution timings, partic-

Table 3. Optimal elimination tree speedup of example problems.

	LA70x70	LA200x200	BCCSTK24	BCCSTK33	LNS3937	NNC1374
Speedup	3.14	2.65	2.75	1.74	2.01	3.57

Table 4. Speedups due to the parallelization of individual nodes wihtin the elimination tree.

Simulated Architecture	LA70x70	LA200x200	BCCSTK24	BCCSTK33	LNS3937	NNC1374
Suns/Ethernet	1.04	1.63	1.20	2.68	1.03	1.00
iPSC/2	5.24	14.49	9.74	33.66	5.84	1.75
SPARC-2/Ethernet	1.00	1.00	1.00	1.00	1.00	1.00
iPSC/860	1.45	2.96	1.96	5.43	1.53	1.01
RS/6000 LAN	1.03	1.71	1.22	2.96	1.02	1.00
SPARCS/HIPPI	2.02	5.56	3.30	14.61	2.28	1.02
Crays/HIPPI	1.02	1.68	1.19	2.98	1.01	1.00

on various distributed memory architectures. Given an unbounded number of processors, some well-balanced systems, such as the iPSC/2, showed moderate speedups; most simulations of workstation clusters, however, rarely produced speedups greater than ten.

The results from these simulations suggest that multifrontal methods share similar characterizations with other parallel direct solvers –they are quite effective at achieving reasonable speedups on shared memory architectures and small clusters of workstations, but they may prove more challenging to adapt to massively parallel

Table 5. Combined speedup of various architectures

Simulated Architecture	LA70x70	LA200x200	BCCSTK24	BCCSTK33	LNS3937	NNC1374
Suns/Ethernet	3.45	7.56	4.09	7.50	2.11	2.60
iPSC/2	37.67	127.21	56.14	125.19	24.28	9.83
SPARC-2/Ethernet	1.51	2.44	1.38	1.57	1.57	1.00
iPSC/860	6.62	18.74	9.21	18.66	4.25	3.23
RS/6000 LAN	3.31	7.91	4.15	8.23	2.06	1.58
SPARCS/HIPPI	11.13	46.96	18.37	53.50	7.10	3.56
Crays/HIPPI	3.17	7.64	3.95	8.14	2.00	1.23

graph. This is a reasonable for assumption for LANs or worm-hole routine topologies, e.g. iPSC/2 and iPSC/860 [10].

Next, we consider the clustering and partioning algorithms to the map the computational tree onto the virtual architecture. For the tree-parallelism case, we utilize a mapping algorithm similar to **dominant sequence clustering** (DSC) algorithm of Yang & Gerasoulis[11].

The algorithm is based on stepwise refinement, and optimizes for ideal parallelism, making no concessions for load balancing or any other **limited resource** considerations. Because we are only interested in modeling speedup, we do not require the algorithm to produce a schedule, but simply the resulting time required to execute the corresponding computation graph. It is thus ideal for establishing upper bounds on the available parallelism exhibited by a given application problem.

5 Results

The results of these simulations are given in tables 3 through 6. The optimal speedup of the amalgamated elimination trees (Eq. 6) for our test problems is given in table 3. Here we have assumed ideal conditions; there are no communication costs. Even under such ideal conditions, these speedups are between 1 and 3 due the imbalance of the respective elimination trees. This suggests we need to utilize node parallelism as well for effective parallelization.

In table 4 we list the results on node-only parallelism. Here we traverse the nodes of the tree sequentially but perform large frontal factorizations within each node in parallel (eq. 13). This provides better speedups with systems such as the iPSC/2 and a SPARC-2/HIPPI network achieving speedups of 34 and 15, respectively.

Significant improvement can be achieved by combining both node-leve land tree-level parallelism. This is shown in table 5, with the corresponding execution times in table 6. We have also included a single 100 Mflop supercomputer as a reference point to compare against conventional vector-pipelined technologies.

As expected, the results in table 5 illustrate that the larger problems provide the best speedups. Well balanced systems such as the iPSC/2 in which the time to compute a floating point result is comparable to the time required for sending one, show particular good speedups. Unfortunately, the overall execution times are not as high, due to the conservative power of its processors. We can also see from this table that the communication bandwidth of Ethernet severly limits a large cluster of workstations to be utilized effectively. For example, utilizing various SPARC-2s connected by Ethernet (a popular configuration in scientific and research environments) provides speedups less than three, even for the large LA200x200 and BCCSTK24 examples.

6 Conclusions

We have derived analytical performance models for the numerical factorization phase of the mutlifrontal method and have applied these models to large examples from the Harwell-Boeing Sparse Matrix Suite to give an indication of the speedups obtainable

of these larger examples are indicative of the medium-scale computing problems many scientists and engineers encounter on a daily basis. The Harwell-Boeing Test Suite was chosen because it is a widely used reference for the design, validation, and benchmarking of sparse matrix solvers and allows for comparison between various machine architectures. A list of the various problems used in this study in shown in Table 2. Furthermore, we have included 5pt and 9pt Laplacian discretization examples which allow us to construct sparse matrix problems of arbitrary size. Finally, we have purposely included a small-scale problem, NCC1374, to illustrate how problem size limits effective parallelism.

Table 2. Example matrices from the Harwell-Boeing Test Suite.

#	Name	Size	nz	Description
1	9pt70x70	4900	43264	5-point of Laplacian operator (70x70 grid)
2	9pt200x200	40,000	198802	9-point of Laplacian operator (200x200 grid)
3	BCCSTK15	3948	177816	offshore platform
4	BCCSTK18	11948	149090	nuclear power station
5	BCCSTK24	3562	159910	Calgary Olympic Arena
6	BCCSTK33	8738	591904	automobile steering mechanism
7	LNS3937	3937	25407	Linearized Navier-Stokes equations
8	NNC1374	1374	8606	Nuclear reactor core modelling

4.3 Simulations

The simulation process consists of three basic parts: (1) given the original matrix, we perform reordering, amalgamation, and symbolic factorization to generate an amalgamated elimination tree representing data dependencies between the elimination varaibles, (2) using the analytical models developed in the previous sections, together with an (α, β, γ) architecture description, we annotate the nodes the elimination tree with the respective computation and communication times, and (3) perform clustering and partitioning algorithms (see below) to determine possible speedups and performance times.

The elimination trees for these problems were produced using the MUPSAD() subroutine from the MUPS[1] software package for shared-memory parallel multifrontal solvers. In all our test matrices we use the minimum degree ordering algorithm together with an amalgamation process (setting the amalgamation parameter to 4) which generally enhances performance by increasing node granularity. All results reported here are for double precision (8 bytes) floating point format.

With the amalgamted tree in place, we then annotate its nodes with the respective computation times (Eqs. 1, 2, 3) and the edges with communication times (Eq. 8). Because we model communication time as $\alpha + \beta n$ between any two nodes, we imply that the underlying virtual architecture behaves like a completely connected

4.1 Example Parallel Architectures

We characterize distributed computer architectures by the three classical parameters: γ, the average time to perform a logical or floating point operation, α, the communication startup cost and β, the communication bandwidth, or the per-byte cost of sending a large message. Having a simple parameterization of a parallel architecture (α, β, γ) allows us to study the effects of each component and experiment with various hypothetical situations. Rather than study random combinations of (α, β, γ), however, we have focused on combinations that typify existing architectures as well as architectures of the future. Besides conventional distributed memory parallel machines, such as the iPSC/860 and iPSC/2, we have included various networks of popular workstations including: a cluster of SUN-3's on standarded ethernet, a dedicated LAN of RS/6000s, various SPARC-2 on a fiber optic network, and multiple Cray supercomputers connected together by a 50 MB/sec HIPPI interface. The list of example architectures used in this study is shown in Table 1 and includes examples from a wide range of computing resources. Megaflop numbers are for double precision operations. Some of performance numbers, such as the iPSC/860 and iPSC/2, were taken from direct experimentation [10] while others were based on reasonable approximations. We approximated Ethernet speeds at 100 KBytes/sec over an average load. The performance numbers for the HIPPI channel bandwidth is, at best, a reasonable estimate.

Table 1. Parameterization of example parallel architectures.

Simulated Architecture	Computation (Mflops)	Comm. Startup ($\mu secs$)	Comm. Bandwidth (MB/sec)
Sun-3 on Ethernet	0.4	5,000	0.1
Intel iPSC/2	0.3	321.8	2.8
SPARC-2s on Ethernet	4.0	5,000	0.1
Intel iPSC/860	4.0	321.8	2.8
SPARC-2s on HIPPI	4.0	2,000	50.0
RS/6000 Dedicated LAN	20.0	5,000	1.0
Crays on Ethernet	100.0	5,000	0.1
Crays on HIPPI	100.0	5,000	50.0

4.2 Example Sparse Matrix Problems

To make our study relevant to practical situations, we have utilized matrices taken from real scientific and engineering applications. These test matrices are part of the the Harwell-Boeing Test Suite [4], and include a linearized Navier-Stokes problem, an example from nuclear reactor modelling, various structural engineering examples of power stations, offshore platforms, and an automobile steering mechanism. Some

$K_i \times K_i$ matrix must be broken up into p pieces and sent to respective processors. Thus,

$$F_i(p) = \underbrace{2(p-1)\left[\alpha + w\beta\frac{K_i^2}{p}\right]}_{\text{communication}} + \underbrace{\frac{F_i(1)}{p}}_{\text{computation}} \tag{12}$$

where $F_i(1)$ is simply F_i defined above. The factor of 2 appears in the first term because the subproblems are again brought back to the original processor before computation continues to the next father node. The $(p-1)$ factor occurs since the $(p-1)$ panels must be dealt out to subsequent processors. The optimal time required to perform the assembly and parallel factorization at each node i is given by

$$T_i = \min_{1 \le p \le K_i} \left(\underbrace{A_i}_{\text{assembly}} + \underbrace{2(p-1)M(\frac{K_i^2}{p})}_{\text{communication}} + \underbrace{\frac{F_i(1)}{p}}_{\text{computation}} \right). \tag{13}$$

Since we are still processing the elimination tree sequentially, the optimal time for the complete computation is simply this sum over all nodes in the elimination tree

$$T^*_{\text{node}} = \sum_{i=1}^{n} T_i. \tag{14}$$

3.3 Combining Node and Tree Parallelism

Clearly, the best solution combines both tree and node levels of parallelism. The analytical expression looks similar to Eq. 11, except that the F_i term has been replaced by $F_i(p)$ and the optimization space is over three dimensions,

$$T^*_{\text{combined}} \equiv$$

$$\min_{\substack{\mathbf{f}() \\ 1 \le i \le K_i}} \max_{\ell} \left(\sum_{i \in \text{path}_\ell} A_i + 2(p-1)M(\frac{K_i^2}{p}) + \frac{F_i(1)}{p} + (1 - \delta_{i,\mathbf{f}(S_i^{-1})})M((K_i - P_i)^2). \right)$$

Computing this combinatorical optimization is computationally unfeasible due to the large search space; in Section 4.3 we describe an hueristic solutions to this problem.

4 Computational Performance Models

This section presents experimental results obtained by applying the previous analytical models to various test problems from the Harwell-Boeing Sparse Matrix Suite. Using an concise parameterization of distributed computer architecture, we provide optimistic estimates on the performance of the parallelized multifrontal method on various computing platforms. The speedups presented here are for an unbounded number of processors, since were are interested in studying the communication and granularity effects of the algorithm under highly scalable architectures.

where α is the startup cost, β per-byte transmission cost and w is the number of bytes per word (typically four for single precision, eight for double precision). This linear model fits many distributed memory machines rather well for point-to-point communication [10]. The communication time required for node i is given by

$$C_i = M([K_i - P_i]^2) \tag{8}$$
$$= \alpha + w\beta * (K_i - P_i)^2 \tag{9}$$

The communication cost is incurred only if the father node is located on a different physical processor. Thus, if $\mathbf{f}(i)$ denotes the processor assigned to node i, the actual communication time is then given by

$$C_i^* = (1 - \delta_{\mathbf{f}(i),\mathbf{f}(S_i^{-1})})\ (\alpha + w\beta * (K_i - P_i)^2) \tag{10}$$

where δ is the Kronecker delta symbol ($\delta_{i,j} = 1$ if $i = j$ or 0 otherwise) and S_i^{-1} is the father node of i. The total time required to process the complete computational tree, therefore, depends on the processor mapping, $\mathbf{f}()$, and the optimal time is given by

$$T^*_{\text{tree}} = \min_{\mathbf{f}()} \left[\max_{\ell} \sum_{i \in \text{path}_\ell} (A_i + F_i) + (1 - \delta_{\mathbf{f}(i),\mathbf{f}(S_i^{-1})})(\alpha + w\beta * (K_i - P_i)^2) \right] \tag{11}$$

The speedup of the **tree parallelism** over the sequential implementation is given by $s_{\text{tree}} = T^*_{\text{seq}}/T^*_{\text{tree}}$

3.2 Node Parallelism

An alternative is to exploit node-level parallelism while processing the computational tree sequentially. This approach is more typical of shared-memory paradigms, since we effectively keep the same sequential code but parallelize the inner loops of the computational kernels.

This approach is still effective in the distributed memory environment because nodes near the root of the computation tree will dominate the execution time and will have to be decomposed in parallel to achieve significant speedups. Modelling the dense LU parallel factorization of a single node, however, can be rather complicated since the best parallel algorithms rely on the effective overlapping of communication with computation for their successful implementation .

The assembly process, however, is more difficult to parallelize since it involves combining results from child nodes. Here we assume that only the factorization phase ($O(n^3)$) is parallelized, i.e. the assembling of the contribution blocks ($O(n^2)$) occurs sequentially. We also make the simplifying assumption that most of the internal communication is involved in the original distribution of the frontal matrix over various processors.

Let p denote the number of processors. At each node i the factorization phase, F_i now becomes a function of p, the number of processors it is distributed over. The

The computation at each node consists of the assembly part A_i and the numerical elimination part, F_i (step 2 above.) Thus, the sequential time, T_i, spent at elimination tree node i is given by

$$T_i = A_i + F_i \tag{3}$$

where A_i denotes the assembly portion involving indirect additions during the processing of a contribution block. Assuming that logical comparisons, floating point multiplies, adds, and divides all take roughly the same time we have, as shown in [5],

$$F_i = \gamma \left[P_i(1 + 2K_j(K_j + 1)) - \frac{2}{3}(P_j + 1)(3K_j - P_j + 1) \right]. \tag{4}$$

The overall sequential complexity, then, is simply the sum of work on all nodes in the elimination tree:

$$T^*_{\text{seq}} = \sum_{i=1}^{n} (A_i + F_i) \tag{5}$$

3.1 Tree Parallelism

To exploit the natural parallelism of the computational tree, we run a sequential version frontal factorization code in each node but process various nodes in parallel. From the program development point of view, this approach is quite attractive since it involves little modification of the sequential code. The node subroutines essentially remain unchanged.

Under ideal conditions with zero communication costs and an unbounded number of processors, the time complexity is simply the longest execution path from leaf to node of the computational tree,

$$T^*{}_{\text{root}} = \max_{\ell} \sum_{i \in \text{path}_\ell} (A_i + F_i) \tag{6}$$

Unfortunately, since most of the computational bulk lies near the root nodes where parallelism is limited, speedups using only this parallelizing strategy are rather small. In Duff & Johnson[5], for example, this speedup varied from 1.9 to 4.0 for various 30x30 sample problems, even when an unbounded number of processors is allowed. In our test cases we found these speedups to remain unchanged for larger 200x200 examples. (See table 3.)

With realistic communication considerations these speedups drop even further. Typically, large clusters of nodes near the leaves get mapped to a single processor. This is necessary because for small nodes it essentially takes much longer to ship the contribution block between processors that to compute it the frontal factorization. Remember that the communication at node i consists of shipping a $(K_i - P_i) \times (K_i - P_i)$ contribution matrix to the father node. We assume the communication cost of moving n words of data between two processors is given by

$$M(n) = \alpha + w\beta * n \tag{7}$$

2.1 Multifrontal Method

Given a sparse matrix A, the reordering and amalgamation phases produce an **amalgamated elimination tree** which describes the data dependencies between the elimination steps (figure 1-b). That is, processing begins at the leaves of the tree and the elimination step at each node cannot begin until its child nodes have completed. Each node in this elimination tree factors a small dense matrix and ships a portion of this matrix up to its father node. The amalgamation phase combine various elimination variables into a single factorization step to increase the granularity of each node and enhance vectorization of the computational kernels. Each node i is annotated with two values: K_i the size of the frontal matrix, and P_i, the number of elimination variables.

Using the analytical models discussed in section 3, one can transform this elimination tree into a **computational graph** (figure 1-e) whose nodes represent computational requirements and edges denote the communication requirements between tasks. The structure of the elimination and computation tree are the same – the computation tree simply has these extra annotations. Coupled with characterizations of machine architectures (figure 1-f), such computational graphs then need to be **mapped** onto a given parallel architecture where nodes of the computational graph are assigned to individual processors.

3 Analytical Performance Models

The reordering and amalgamation phase (figure 1-b) produces an annotated tree where each node i is labeled with its number of pivots (P_i) and the frontal matrix size (K_i), where $P_i \leq K_i$. The size of the of contribution block is $(K_i - P_i)$ x $(K_i - P_i)$, and the size of the frontal matrix is K_i x K_i. As discussed in Duff & Johnson [5], the sequential multifrontal factorization code consists of the following basic steps. At each node in the elimination tree we perform assembly of contribution blocks from child nodes; for each pivot in frontal matrix, (1) determine the pivot (within the fully-summed variables 1,2, $\ldots, P_i$), (2) divide pivot column, and (3) update trailing submatrices in frontal matrix. Let n be the number of nodes in the amalgamated elimination tree, and let γ represent the average to time execute a logical or floating point operation. We can characterize the time required to complete the frontal factorization at node i as follows. The assembly portion consists of receiving the contribution block from the child nodes and adding them into the local frontal matrix. Thus the time required for this is given by

$$A_i = \gamma\left(\sum_{j \in S_i} (K_j - P_j)^2\right) \tag{1}$$

where S_i denotes the nodes which are children of node i. Similarly, the factorization time F_i of the frontal matrix is given by

$$F_i = \gamma \left(\underbrace{\sum_{j=1}^{P_i} (K_i - j + 1)}_{\text{scan for pivot}} + \underbrace{\sum_{j=1}^{P_i} (K_i - j)}_{\text{scaling}} + \underbrace{2 \sum_{j=1}^{P_i} (K_i - j)^2}_{\text{rank-1 updates}} \right). \tag{2}$$

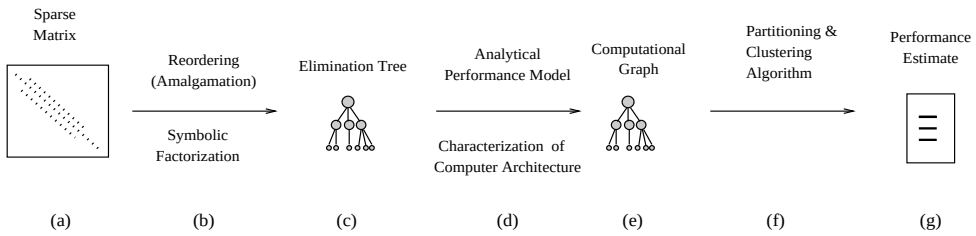

Fig. 1. Overview of the analysis and performance modeling phases for a mutlifrontal sparse matrix solver.

We present analytical performance models which characterize the communication and computation aspects of this algorithm. By utilizing such models in software simulations, we can study the performance characteristics of the multifrontal method on various architecture characterizations and provide upper bounds for the optimal speedups observed for various engineering and scientific problems taken from the Harwell-Boeing Test Suite[4]. This collection includes structural finite-element problems with irregular grids. Our architecture representations include conventional distributed memory parallel machines, such as the iPSC/860 and iPSC/2, and various distributed networks, such as a cluster of SUN-3's on standard Ethernet, a dedicated LAN of RS/6000s, various SPARC-2 on a dedicated high-speed network, and multiple supercomputers connected by a 50 MB/sec HIPPI interface. The intent is not to benchmark one architecture against another (after all, these are only approximations), but to understand how the communication and computation aspects of a particular architecture affects the obtainable parallelism in distributed sparse matrix methods.

2 Direct Sparse Matrix Factorizations

Although our study focuses on the multifrontal method, some of its characteristics are typical of other direct methods, such as the column-submatrix Cholesky factorization. In particular, the computational kernel of the multifrontal method is quite similar to the right-looking LU decomposition algorithm [2] for dense matrices. A full introduction to the multifrontal method is beyond the scope of this paper; here we briefly discuss issues relevant to the performance modeling and direct the reader to [6] for details of the numerical algorithm.

We focus on the numerical factorization phase of parallel multifrontal methods since it is the most computationally intensive section and generally dominates the overall execution time. By comparison, the ordering, and symbolic factorization phases of the solution are generally smaller and have limited benefits from parallelization [7]. Furthermore, in many practical cases multiple factorizations are performed on matrices with identical sparsity structure, thereby requiring the ordering and symbolic factorization phases be executed only once.

Performance Modeling of Sparse Matrix Methods for Distributed Memory Architectures.

Roldan Pozo

Department of Computer Science, University of Tennessee
Knoxville, TN 37919, USA
pozo@cs.utk.edu

Abstract. We present analytical performance models for the numerical factorization phase of the multifrontal method for sparse matrices. Using a concise characterization of parallel architectures, we provide upper-bound estimates for the speedups observed on actual test problems taken from scientific and engineering applications. Representative architectures include an iPSC/2, iPSC/860, various clusters of workstations, and supercomputers connected by HIPPI interfaces. Simulation results suggest that the effective parallelism of these problems is quite sensitive to the communication bandwidth of the underlying architecture and load imbalances in the computational graph due to irregular data patterns.

1 Introduction

The solution of large sparse linear systems constantly arises in many large scale scientific applications, including computational fluid dynamics, structural engineering, weather forecasting, computational physics, circuit simulations, and biomedicine. Because these problems require significant computational resources, there has been much interest to solve large sparse linear systems on parallel architectures. Recent attention has focused on distributed memory architectures that can scale with increasing problem sizes, with growing interest in small clusters of powerful workstations as a cost-effective means of synthesizing large computational engine.

Various efforts have been directed at solving large sparse systems using direct solvers on distributed memory architectures; a good survey appears in Heath, et. al. [7]. One technique, the **multifrontal** method has exhibited a good potential for parallelism on shared memory architectures [3] and is particularly well-suited for high performance computer architectures since it involves no indirect addressing in its innermost loops . The multifrontal method was originally motivated by problems in structural finite element applications but it is also applicable to a larger class of scientific and engineering domains.

In this paper we address how the communication and computational characteristics of a given computer architecture affect the performance of the numerical factorization phase of the multifrontal method. The computational graphs associated with such algorithms typically resemble an unbalanced tree with hundreds or thousands of independent paths, and the granularity of each task increasing towards the root. Thus an efficient implementation onto distributed memory architectures requires the **partitioning** (or parallelization) of large nodes near the tree root, and the **clustering** of small leaf nodes onto a single processor due to their fine granularity.