Performance Analysis of Parallel Right-Looking Sparse LU Factorization on Two Dimensional Grids of Processors

Laura Grigori[1] and Xiaoye S. Li[2]

[1] INRIA Rennes
Campus Universitaire de Beaulieu, 35042 Rennes, France.
Laura.Grigori@irisa.fr
[2] Lawrence Berkeley National Laboratory, MS 50F-1650
One Cyclotron Road, Berkeley, CA 94720, USA.
xsli@lbl.gov

Abstract. We investigate performance characteristics for the LU factorization of large matrices with various sparsity patterns. We consider supernodal *right-looking* parallel factorization on a two dimensional grid of processors, making use of static pivoting. We develop a performance model and we validate it using the implementation in SuperLU_DIST, the real matrices and the IBM Power3 machine at NERSC. We use this model to obtain performance bounds on parallel computers, to perform scalability analysis and to identify performance bottlenecks. We also discuss the role of load balance and data distribution in this approach.

1 Introduction

A valuable tool in designing a parallel algorithm is to analyze its performance characteristics for various classes of applications and machine configurations. Very often, good performance models reveal communication inefficiency and memory access contention that limit the overall performance. Modeling these aspects in detail can give insights into the performance bottlenecks and help improve the algorithm. The goal of this paper is to analyze performance characteristics and scalability for the LU factorization of large matrices with various sparsity patterns.

For dense matrices, the factorization algorithms have been shown to exhibit good scalability, where the efficiency can be approximately maintained as the number of processors increases when the memory requirements per processor are held constant [2]. For sparse matrices, however, the efficiency is much harder to predict since the sparsity patterns vary with different applications. Several results exist in the literature [1, 4, 6], which were obtained for particular classes of matrices arising from the discretization of a physical domain. They show that factorization is not always scalable with respect to memory use. For sparse matrices resulting from two-dimensional domains, the best parallel algorithm lead to an increase of the memory at a rate of $O(P \log P)$ with increasing P [4]. It

is worth mentioning that for matrices resulting from three-dimensional domains, the best algorithm is scalable with respect to memory size.

In this work, we develop a performance model for a sparse factorization algorithm that is suitable for analyzing performance with arbitrary input matrix. We use a classical model to describe an ideal machine architecture in terms of processor speed, network latency and bandwidth. Using this machine model and several input characteristics (order of the matrix, number of nonzeros, etc), we analyze a supernodal *right-looking* parallel factorization on two dimensional grids of processors, making use of static pivoting. This analysis allows us to obtain performance upper bounds on parallel computers, to perform scalability analysis, to identify performance bottlenecks and to discuss the role of load balance and data distribution. More importantly, our performance model reveals the relationship between parallel runtime and matrix sparsity, where the sparsity is measured with respect to the underlying hardware's characteristics. Given any combination of application and architecture, we can obtain this sparsity measure. Then our model can quantitatively predict not only the performance on this machine, but also what hardware parameters to improve are most critical to improve the performance for this type of applications. We validate our analytical model using the actual factorization algorithm implemented in the SuperLU_DIST [5] solver, the real-world matrices and the IBM Power3 machine at NERSC. We also show that the runtime predicted by our model is more accurate than that predicted by simply examining the workload on the critical path, because our model takes into account both task dependency and communication overhead.

The rest of the paper is organized as follows: Section 2 introduces a performance analysis model for the right-looking factorization, with its scalability analysis. The experimental results validating the performance model are presented in Section 3 and Section 4 draws the conclusions.

2 Parallel right-looking sparse LU factorization on two dimensional grids of processors

Consider factorizing a sparse unsymmetric $n \times n$ matrix A into the product of a unit lower triangular matrix L and an upper triangular matrix U. We discuss a parallel execution of this factorization on a two dimensional grid of processors. The matrix is partitioned into $N \times N$ blocks of submatrices using unsymmetric supernodes (columns of L with the same nonzero structure). These blocks of submatrices are further distributed among a two dimensional grid $P_r \times P_c$ of P processors ($P_r \times P_c \leq P$) using a block cyclic distribution. With this distribution, a block at position (I, J) of the matrix ($0 \leq I, J < N$) will be mapped on the process at position $(I \, mod \, P_r, J \, mod \, P_c)$ of the grid. $U(K, J)$ ($L(K, J)$) denotes a submatrix of U (L) at row block index K and column block index J.

The algorithm below describes a right-looking factorization and Figure 1 illustrates the respective execution on a rectangular grid of processors. This algorithm loops over the N supernodes. In the K-th iteration, the first $K - 1$ block columns of L and block rows of U are already computed. At this iteration,

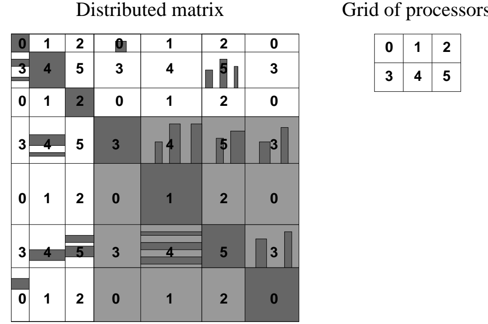

Fig. 1. Illustration of parallel right-looking factorization

first the column of processors owning block column K of L factors this block column $L(K:N,K)$; second, the row of processors owning block row K of U performs the triangular solve to compute $U(K,K+1:N)$; and third, all the processors update the trailing matrix using $L(K+1:N,K)$ and $U(K,K+1:N)$. This third step requires most of the work and also exhibits most of the parallelism in the right-looking approach.

for $K := 1$ to N **do**
  Factorize block column $L(K:N,K)$
  Perform triangular solves: $U(K,K+1:N) := L(K,K)^{-1} \times A(K,K+1:N)$
  for $J := K+1$ to N with $U(K,J) \neq 0$ **do**
    for $I := K+1$ to N with $L(I,K) \neq 0$ **do**
      Update trailing submatrix:
      $A(I,J) := A(I,J) - L(I,K) \times U(K,J)$
    end for
  end for
end for

The performance model we develop for the sparse LU factorization is close to the performance model developed for the dense factorization algorithms in ScaLAPACK [2]. Processors have local memory and are connected by a network that provides each processor direct links with any of its 4 direct neighbors (mesh-like).

To simplify analysis and to make the model easier to understand, we make the following assumptions:

- We use one parameter to describe the processor flop rate, denoted γ, and we ignore communication collisions. We estimate the time for sending a message of m items between two processors as $\alpha + m\beta$, where α denotes the latency and β the inverse of the bandwidth.
- We approximate the cost of a broadcast to p processors by $\log p$ [2]. Furthermore, the LU factorization uses a pipelined execution to overlap some of the communication with computation, and in this case the cost of a broadcast is estimated by 2 [2].

– We assume that the computation of each supernode lies on the critical path of execution, that is the length of the critical path is N. We also assume that the load and the data is evenly distributed among processors. Later in Section 3, we will provide the experimental data verifying these assumptions.

Runtime estimation. We use the following notations to estimate the runtime to factorize an $n \times n$ matrix. We use c_k to denote the number of off-diagonal elements in each column of block column K of L, r_k to denote the number of off-diagonal elements in each row of block row K of U, $nnz(L)$ to denote the number of nonzeros in the off-diagonal blocks of L, $nnz(U)$ to denote the number of nonzeros in the off-diagonal blocks of U. $M = 2\sum_{k=1}^{n} c_k r_k$ is the total number of flops in the trailing matrix update, counting both multiplications and additions. $F = nnz(L) + M$ is the total number of flops in the factorization.

With the above notations, the sequential runtime can be estimated as $T_s = nnz(L)\gamma + M\gamma = F\gamma$.

We assume each processor in the column processors owning block column K gets $s \cdot c_k/P_r$ elements, where $s \cdot c_k$ is the number of nonzeros in the block column K and P_r is the number of processors in the column. Block row K of U is distributed in a similar manner.The parallel runtime using a square grid of processors can be expressed as:

$$T(N, \sqrt{P} \times \sqrt{P}) \approx \frac{F}{P}\gamma + (2N + \frac{1}{2}N \log P)\alpha + \frac{(2nnz(L) + \frac{1}{2}nnz(U) \log P)}{\sqrt{P}}\beta$$

The first term represents the parallelization of the computation. The second term represents the number of broadcasting messages. The third term represents the volume of communication overhead.

Scalability analysis. We now examine the scalability using a square grid of processors of size P, where the efficiency of the right-looking algorithm is given by the following formula:

$$\epsilon(N, \sqrt{P} \times \sqrt{P}) = \frac{T_s(N)}{PT(N, \sqrt{P} \times \sqrt{P})} \tag{1}$$

$$\approx \left[1 + \frac{NP \log P}{F}\frac{\alpha}{\gamma} + \frac{(2nnz(L) + nnz(U) \log P)\sqrt{P}}{F}\frac{\beta}{\gamma}\right]^{-1} \tag{2}$$

One interesting question is which of the three terms dominates efficiency (depending on the size and the sparsity of the matrix). The preliminary remark is that, for very dense problems (F large), the first term significantly affects the parallel efficiency.

For the other cases, we can compare the last two terms to determine which one is dominant. That is, if we ignore the factors 2 and $\log P$ in the third term,

we need to compare $\sqrt{P}\frac{\alpha}{\beta}$ with $\frac{nnz(L+U)}{N}$. Assuming that the network's latency-bandwidth product is given ($\frac{\alpha}{\beta}$), we can determine if the ratio of the latency to the flop rate (α/γ term) or the ratio of the inverse of the bandwidth to the flop rate (β/γ term) dominates efficiency. Overall, the following observations hold:

Case 1 For sparser problems ($\sqrt{P}\frac{\alpha}{\beta} > \frac{nnz(L+U)}{N}$), the α/γ term dominates efficiency.

Case 2 For denser problems ($\sqrt{P}\frac{\alpha}{\beta} < \frac{nnz(L+U)}{N}$), the β/γ term dominates efficiency.

Case 3 For problems for which $\frac{nnz(L+U)}{N}$ is close to $\sqrt{P}\frac{\alpha}{\beta}$, the β/γ term can be dominant on smaller number of processors, and with increasing number of processors the α/γ term can become dominant.

Note that even for Case 2, the algorithm behaviour varies during the factorization: at the beginning of the factorization, where the matrix is generally sparser and the messages are shorter, α/γ term dominates the efficiency, while at the end of the factorization where the matrix becomes denser, β/γ term dominates the efficiency.

Let us now consider matrices in Case 2. For these matrices, in order to maintain a constant efficiency, $\frac{F}{nnz(L+U)}$ must grow proportionally with $\sqrt{P}$. On the other hand, for a scalable algorithm in terms of memory requirements, the memory requirement $nnz(L+U)$ should not grow faster than P. Thus, when we allow $nnz(L+U) \propto P$, the condition $F \propto nnz(L+U)^{3/2}$ must be satisfied, and the efficiency can be approximately maintained constant. (In reality, even for these matrices, α/γ term as well as $\log P$ factor will still contribute to efficiency degradation.) We note that the matrices with N unknowns arising from discretization of Laplacian operator on three-dimensional finite element grids fall into this category. Using nested dissection, the number of fill-ins in such matrix is on the order of $O(N^{4/3})$ while the amount of work is on the order of $O(N^2)$. Maintaining a fixed efficiency requires that the number of processors P grows proportionally with $N^{4/3}$, the size of the factored matrix. In essence, the efficiency for these problems can be approximately maintained if the memory requirement per processor is constant. Note that α/γ term grows with $N^{1/3}$, which also contributes to efficiency loss.

3 Experimental results

In this section, we compare the analytical results against experimental results obtained on a IBM Power3 machine at NERSC, with real-world matrices. The test matrices and their characteristics are presented in Table 1. They are ordered according to the last column, $nnz(L+U)/N$, which we use as a measure of the sparsity of the matrix.

The first goal of our experiments is to analyze the different assumptions we have made during the development of the performance model. Consider again the efficiency Equation (2) in Section 2. For the first term, we assume that the

Matrix	Order	N	$nnz(A)$	$nnz(L+U)$ $\times 10^6$	$Flops(F)$ $\times 10^9$	$nnz(L+U)/N$ $\times 10^3$
af23560	23560	10440	484256	11.8	5.41	1.13
rma10	46835	6427	2374001	9.36	1.61	1.45
ecl32	51993	25827	380415	41.4	60.45	1.60
bbmat	38744	11212	1771722	35.0	25.24	3.12
inv-extr1	30412	6987	1793881	28.1	27.26	4.02
ex11	16614	2033	1096948	11.5	5.60	5.65
fidapm11	22294	3873	623554	26.5	26.80	6.84
mixingtank	29957	4609	1995041	44.7	79.57	9.69

Table 1. Benchmark matrices.

load is evenly distributed among processors, while for the third term we assume that the data is evenly distributed among processors. Note that we also assume that the computation of each supernode lies on the critical path of execution. Our experiments show that a two dimensional distribution of the data on a two dimensional grid of processors leads to a balanced distribution of the data. They also show that for almost all the matrices, the critical path assumption is realistic [3].

However, the load balance assumption is not always valid. To assess load balance, we consider the load F to be the number of floating point operations to factorize the matrix. We then compute the load lying on the critical path F_{CP} by adding at each iteration the load of the most loaded processor in this iteration. More precisely, consider f_{pi} being the load of processor p at iteration i (number of flops performed by this processor at iteration i). Then $F_{CP} = \sum_{i=1}^{N} \max_{p=1}^{P} f_{pi}$. The load balance factor is computed as $LB = \frac{F_{CP} P}{F}$. In other words, LB is the load of heaviest processors lying on the critical path divided by the average load per processor. The closer this factor approaches 1, the better is the load balance. Contrary to the "usual" way, when the load balance factor is computed as the average load divided by the maximum load among all the processors, our computation of load balance is more precise. Note that we can also use $\frac{F}{F_{CP}}$ to compute a crude upper bound on the parallel speedup, which takes into account the workload on the critical path but ignores communication cost and task dependency. The results are presented in Table 2, in the rows denoted by LB. We observe that the workload distribution is good for large matrices on a small number of processors. But it can degrade quickly for some matrices such as rma10, for which load balance can degrade by a factor of 2 when increasing the numbers of processors by a factor of 2. Consequently, efficiency will suffer a significant degradation.

		P=1	P = 4	P = 16	P = 32	P = 64	P = 128
af23560	time	9.95	3.95	2.36	2.30	3.17	3.38
	LB	1.0	1.28	2.05	2.92	4.21	6.67
rma10	time	3.41	2.12	1.90	1.99	3.03	3.17
	LB	1.0	1.66	2.75	5.62	9.13	16.07
ecl32	time	104.27	29.34	9.49	7.25	7.31	7.22
	LB	1.0	1.09	1.28	1.52	1.80	2.37
bbmat	time	67.37	19.50	7.61	5.64	6.05	6.50
	LB	1.0	1.21	1.75	2.35	3.17	4.88
inv-extr1	time	73.13	19.08	6.46	4.72	4.95	5.29
	LB	1.0	1.14	1.42	1.87	2.45	3.51
ex11	time	9.49	3.27	1.54	1.33	1.65	2.07
	LB	1.0	1.27	1.90	2.58	3.53	5.32
fidapm11	time	51.99	14.21	4.84	3.57	3.47	3.81
	LB	1.0	1.15	1.46	1.83	2.31	3.13
mixingtank	time	119.45	33.87	9.52	6.47	5.53	5.27
	LB	1.0	1.08	1.25	1.43	1.63	2.04

Table 2. Runtimes (in seconds) and load distribution (LB) for right-looking factorization on two dimensional grids of processors

The second goal of the experiments is to show how the experimental results support our analytical performance model developed in Section 2. For this, we use the analytical performance model to predict the speedup that each matrix should attain with an increasing number of processors. Then we compare the predicted speedup against the speedup obtained by SuperLU_DIST. The plots in Figure 2 display these results, where the predicted speedup for each matrix M is denoted by Mp, and the actually obtained (measured) speedup is denoted by Mm. We also display in these plots the upper bound obtained from the workload on the critical path, given by $\frac{F}{F_{CP}}$, and we denote it as MLB.

As the plots show, the analytical performance model predicts well the performance on a small number of processors (up to 30-40 processors), while the predicted speedup starts to deviate above the measured speedup with an increase in the number of processors. This is because on a smaller number of processors our model assumptions are rather realistic, but on a larger number of processors the assumptions are deviating from reality. That is why we see the degraded scalability. More detailed data were reported in [3].

The upper bound based only on the workload on the critical path can be loose, since it ignores communication and task dependency. However, it often corroborates the general trend of the speedup predicted by our analytical model.

For several matrices, such as bbmat, inv-extr1, fidapm11 and mixingtank, this upper bound is very close to the prediction given by our analytical model (see Figure 2), implying that for those matrices, load imbalance is a more severe problem than communication, and improving load balance alone can greatly improve the overall performance.

The third goal of the experiments is to study the actual efficiency case by case for all the test matrices. One approach to do this is to observe how the factorization runtime degrades as the number of processors increases for different matrices. For this we report in Table 2 the runtime in seconds of SuperLU_DIST factorization. These results illustrate that good speedups can be obtained on a small number of processors and show how efficiency degrades on a larger number of processors. As one would expect, the efficiency degrades faster for problems of smaller size (number of flops smaller), and slower for larger problems.

We now examine how the actual model parameters (sparsity, α, β and γ) affect the performance. On the IBM Power3, the measured latency is 8.0 microseconds and the bandwidth ($1/\beta$) for our medium size of messages is 494 MB/s [7]. The latency-bandwidth product is $\alpha/\beta = 4 \times 10^3$. Table 1 shows that the algorithm's efficiency for some matrices is clearly dominated by the α/γ term, such as af23560, rma10, ecl32 (Case 1 matrices). For the other matrices, mixingtank, fidapm11, ex11, inv-extr1, the efficiency is significantly affected by the β/γ term (Case 2 matrices).

Matrices af23560 and ex11 have an approximately equal number of flops, and almost similar runtimes on one processor. But efficiency degrades faster for af23560 than for ex11. This is because the efficiency of af23560 is mostly affected by the α/γ term (Case 1), while the efficiency of ex11 is mainly affected by the β/γ term (Case 2), and thus its performance degrades slower than for af23560. For denser matrices which fall into Case 2, such as mixingtank, the algorithm achieves much better efficiency even on large number of processors. Therefore, the algorithm is more sensitive to latency than bandwidth.

4 Conclusions

We developed a performance model for a sparse right-looking LU factorization algorithm and validated this model using the SuperLU_DIST solver, real-world matrices and the IBM Power3 machine at NERSC.

Using this model, we first analyzed the efficiency of this algorithm with increasing number of processors and problem size. We concluded that for matrices satisfying a certain relation (namely $F \propto nnz(L+U)^{3/2}$) between their problem size and their memory requirements, the algorithm is scalable with respect to memory use. This relation is satisfied by matrices arising from the 3D model problems. For these matrices the efficiency can be roughly maintained constant when the number of processors increases and the memory requirement per processor is constant. But for matrices arising from the 2D model problems, the algorithm is not scalable with respect to memory use, the same as sparse Cholesky factorization [4].

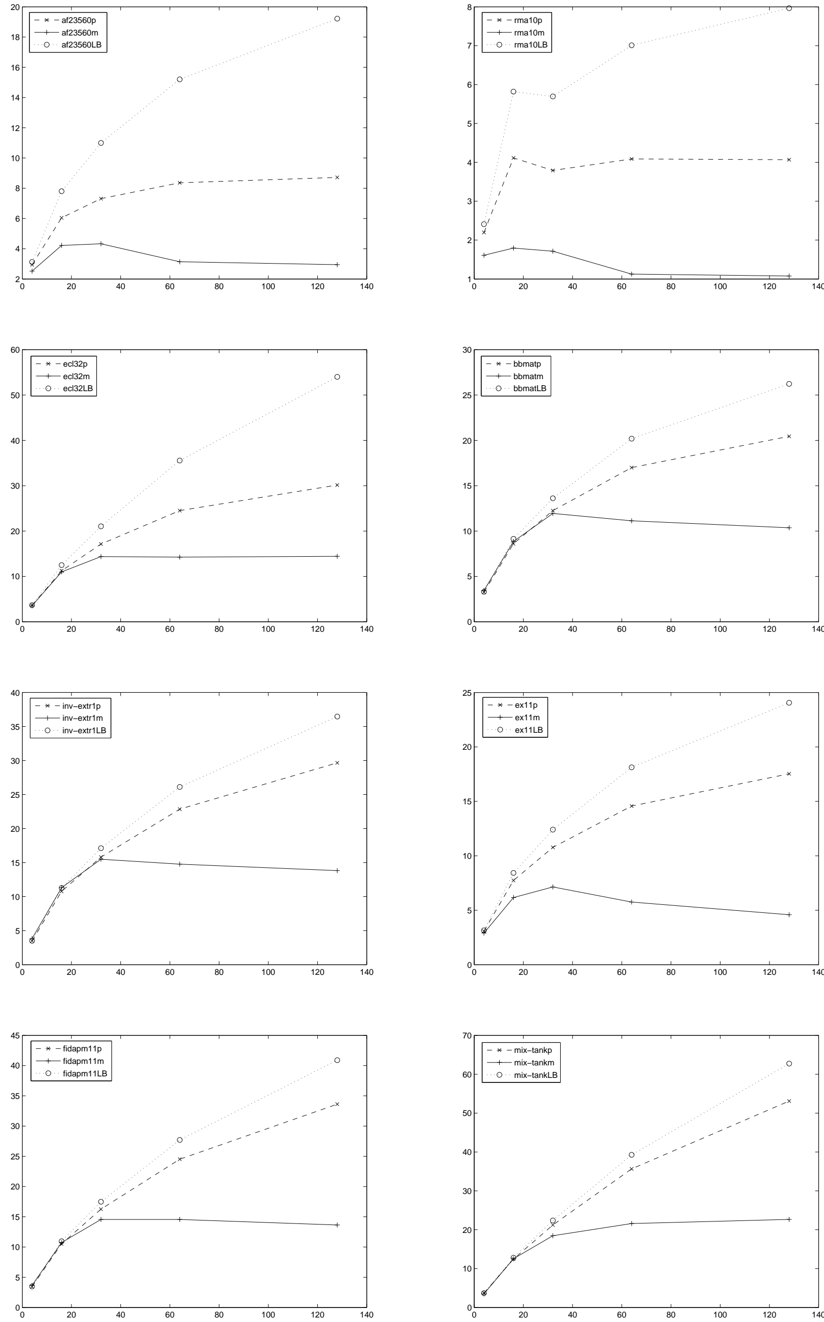

Fig. 2. Speedups predicted by our performance model (labeled "p") and by the load balance constraint (labeled "LB"), versus the measured speedups (labeled "m").

Secondly, we analyzed the efficiency of this algorithm for fixed problem size and increasing number of processors. We observed that good speedups can be obtained on smaller number of processors. On larger number of processors, the efficiency degrades faster for sparser problems which are more sensitive to the latency of the network. A two dimensional distribution of the data on a two dimensional grid of processors leads to a balanced distribution of the data. It also leads to a balanced distribution of the load on smaller number of processors. But the load balance is usually poor on larger number of processors. We believe that load imbalance and insufficient amount of work (F) relative to communication overhead are the main sources of worse efficiency on large number of processors.

One practical use of our theoretical efficiency bound is as follows. For certain application domain, the matrices usually exhibit a similar sparsity pattern. We can measure the sparsity with respect to the underlying machine parameters, i.e., floating-point speed, the network latency and bandwidth. Depending on whether they belong to Case 1 or Case 2, we can determine the most critical hardware parameters which need to be improved in order to enhance the performance for this class of application. In addition, given several choices of machines, we can predict which hardware combination is best for this application.

References

1. Cleve Ashcraft. The fan-both family of column-based distributed Cholesky factorization algorithms. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 159–191. Springer Verlag, 1994.
2. Jack K. Dongarra, Robert A. van de Geijn, and David W. Walker. Scalability Issues Affecting the Design of a Dense Linear Algebra Library. *Journal of Parallel and Distributed Computing*, 22(3):523–537, 1994.
3. Laura Grigori and Xiaoye S. Li. Performance analysis of parallel supernodal sparse lu factorization. Technical Report LBNL-54497, Lawrence Berkeley National Laboratory, Feburary 2004.
4. Anshul Gupta, George Karypis, and Vipin Kumar. Highly Scalable Parallel Algorithms for Sparse Matrix Factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.
5. Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.
6. Robert Schreiber. Scalability of sparse direct solvers. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 191–211. Springer Verlag, 1994.
7. Adrian Wong. Private communication. Lawrence Berkeley National Laboratory, 2002.