

Lecture 11: March 3

Lecturer: Jim Demmel

Scribes: Humberto Gonzalez

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

This is the second lecture about dense linear algebra. The slides of these two lectures are in:
<http://www.cs.berkeley.edu/~demmel/cs267-sp08>

There are also some class projects at the end of the web page.

11.1 Gaussian Elimination

11.1.1 Sequential algorithm

The objective of this algorithm is to solve the a linear system of equations of the form $Ax = b$. The basic algorithm is:

```
for i = 1:n-1
  for j = i+1:n
    tmp = A(j,i)
    for k = i:n
      A(j,k) = A(j,k) - ( tmp / A(i,i) ) * A(i,k)
```

which can be simplified to 3 lines of Matlab:

```
for i = 1:n-1
  A(i+1:n,i) = A(i+1:n,i) / A(i,i)
  A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i) * A(i,i+1:n)
```

Once the matrix has been factorized as $A = L \cdot U$ (cost of $\frac{2}{3}n^3$ flops) the linear system can be solved in two steps (n^2 flops each): first solve $Ly = b$ and then solve $Ux = y$.

If the matrix has elements with values close to zero in the diagonal then the previous algorithm can lead to numerical instability. Consider the following example:

$$\begin{bmatrix} 10^{-n} & 1 \\ 1 & a \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^n & 1 \end{bmatrix} \begin{bmatrix} 10^{-n} & 1 \\ 0 & a - 10^n \end{bmatrix} \quad (11.1)$$

if n is a big number compared to the order of magnitude of a , then $a - 10^n \approx -10^n$ losing the information of one of the elements in the matrix. To avoid this kind of problem we have to add permute the rows (*pivot*) in order to get a bigger diagonal value (chosen among the ones in the same column). Thus, the algorithm becomes:

```

for i = 1:n-1
    Find k = arg maxj∈[i,n] |A(j, i)|
    if A(k, i) = 0
        Exit with a warning, the matrix is singular
    elseif k != i
        Swap rows i and k
    endif
    A(i+1:n, i) = A(i+1:n, i) / A(i, i)
    A(i+1:n, i+1:n) = A(i+1:n, i+1:n) - A(i+1:n, i) * A(i, i+1:n)

```

and the new factorization of the matrix is $A = P \cdot L \cdot U$ where P is the permutation matrix.

Another problem that has to be taken into account is that all the computations in our 3 line Matlab code involve *BLAS1* and *BLAS2* functions (only vector operations), but we know that *BLAS3* (matrix-matrix multiply) is asymptotically optimal as the size of the problem increases. The solution for this problem is to use blocking, so instead of eliminating one column at the time we will eliminate a block of columns, with the condition of choosing a block size that fits in cache.

11.1.2 Parallel algorithm

Assuming the communications don't use any time, this algorithm can be computed in $O(3n)$ steps with n^2 processors. If we add pivoting then we get $O(n \log n)$ because we have to find the maximum value of each column.

Just like when we analyzed the parallel matrix-matrix multiply algorithm, the best layout for this type of problem is the 2D row and column block cyclic layout, since it keeps most of the processors busy during a complete run and under computations for most sub-matrices.

The details about the algorithm can be obtained directly from the slides of the lecture.

To measure the performance of this algorithm (and all the other *ScaLAPACK* algorithms) the following model is used:

$$\text{Total time} = \text{time(FLOPS)} + \text{time(Communications)} + \text{time(messages)}$$

where each of these terms is a function of the number of processors and the size of the problem.

11.1.3 Multi-core algorithm

The biggest problem with the algorithm just described is that a lot of time is spent waiting for the nodes to finish the communications among them to start computing the next part. This can be improved by implementing a smarter scheduling that keeps every processor busy most of the time.

The new scheduling approach is based on a tree of dependencies, where each task can be performed if and only if its parent tasks were already performed. In this way we don't need to synchronize the processor, we just have to keep track of the next available task to be performed according to the tree of dependencies. Now the most important part of the algorithm is the implementation of the scheduler. This approach is called "dynamic scheduling".

11.2 Other linear algebra problems

The following problems can be solved using *LAPACK* (and some of them are implemented in parallel with *ScaLAPACK*):

- “One-sided” problems: Gaussian elimination ($A = LU$), Cholesky ($A = LL^T$), QR.
- “Two-sided” problems: Eigenvalue decomposition, Singular value decomposition.

“One-sided” problems are asymptotically 100% *BLAS3*, so they are considered efficient. “Two-sided” problems can’t be completely expressed as *BLAS3* problems, so they are much more complicated to parallelize optimally.

11.3 Future tasks

- Automated tuning for all (or at least most) *ScaLAPACK* algorithms.
- Take into account the hardware of each multi-core architecture.
- Are there better layouts than 2D row and column cyclic blocks?.
- Better schedulers to dynamically assign jobs from the tree of dependencies.
- Use *GPU*’s to speed up computations.
- Develop mixed precision algorithms, where most of the calculations are performed in single-precision and later refined in double-precision.
- Develop new communication avoidance algorithms, where most of the computations are done locally and achieving a final result recursively.