Parallel Gaussian Elimination

1. LU and Cholesky Factorization
   - factoring a square matrix
   - tiled Cholesky factorization
2. Blocked LU Factorization
   - deriving blocked formulations of LU
   - right and left looking LU factorizations
   - tiled algorithm for LU factorization
3. The PLASMA Software Library
   - Parallel Linear Algebra Software for Multicore Architectures
   - running an example

MCS 572 Lecture 21
Introduction to Supercomputing
Jan Verschelde, 3 March 2014

Parallel Gaussian Elimination

1. LU and Cholesky Factorization
   - factoring a square matrix
   - tiled Cholesky factorization
2. Blocked LU Factorization
   - deriving blocked formulations of LU
   - right and left looking LU factorizations
   - tiled algorithm for LU factorization
3. The PLASMA Software Library
   - Parallel Linear Algebra Software for Multicore Architectures
   - running an example

solving $A\mathbf{x} = \mathbf{b}$ with the LU factorization

To solve an n-dimensional linear system $A\mathbf{x} = \mathbf{b}$
we factor A as a product of two triangular matrices, $A = LU$:

- L is lower triangular, $L = [\ell_{i,j}]$, $\ell_{i,j} = 0$ if $j > i$ and $\ell_{i,i} = 1$.
- U is upper triangular $U = [u_{i,j}]$, $u_{i,j} = 0$ if $i > j$.

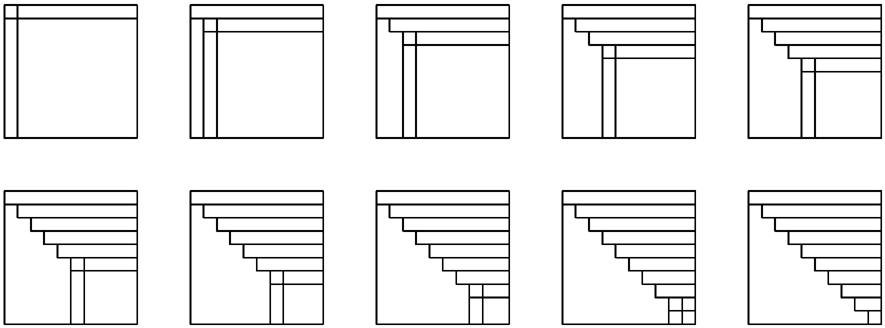
Solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving $L(U\mathbf{x}) = \mathbf{b}$:

1. Forward substitution: $L\mathbf{y} = \mathbf{b}$.
2. Backward substitution: $U\mathbf{x} = \mathbf{y}$.

Factoring A costs $O(n^3)$, solving triangular systems costs $O(n^2)$.

For numerical stability, we apply partial pivoting and compute $PA = LU$, where P is a permutation matrix.

LU factorization of the matrix A



for column $j = 1, 2, \ldots, n-1$ in A do

1. find the largest element $a_{i,j}$ in column j (for $i \geq j$);
2. if $i \neq j$, then swap rows i and j;
3. for $i = j+1, \ldots n$, for $k = j+1, \ldots, n$ do $a_{i,k} := a_{i,k} - \left(\frac{a_{i,j}}{a_{j,j}} \right) a_{j,k}$.

Cholesky factorization

If A is symmetric, $A^T = A$, and positive semidefinite: $\forall \mathbf{x} : \mathbf{x}^T A \mathbf{x} \geq 0$, then we better compute a Cholesky factorization: $A = LL^T$, where L is a lower triangular matrix.

Because A is positive semidefinite, no pivoting is needed, and we need about half as many operations as LU.

```
for $j = 1, 2, \ldots, n$ do
    for $k = 1, 2, \ldots, j-1$ do
        $a_{j,j} := a_{j,j} - a_{j,k}^2$;
    $a_{j,j} := \sqrt{a_{j,j}}$;
    for $i = j+1, \ldots, n$ do
        for $k = 1, 2, \ldots, j$ do
            $a_{i,j} := a_{i,j} - a_{i,k} a_{j,k}$
        $a_{i,j} := a_{i,j} / a_{j,j}$
```

Parallel Gaussian Elimination

1. LU and Cholesky Factorization
   - factoring a square matrix
   - tiled Cholesky factorization

2. Blocked LU Factorization
   - deriving blocked formulations of LU
   - right and left looking LU factorizations
   - tiled algorithm for LU factorization

3. The PLASMA Software Library
   - Parallel Linear Algebra Software for Multicore Architectures
   - running an example

tiled matrices

Let A be a symmetric, positive definite n-by-n matrix.

For tile size b, let $n = p \times b$ and consider

$$A = \begin{bmatrix} A_{1,1} & A_{2,1} & \cdots & A_{p,1} \\ A_{2,1} & A_{2,2} & \cdots & A_{p,2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p,1} & A_{p,2} & \cdots & A_{p,p} \end{bmatrix},$$

where $A_{i,j}$ is an b-by-b matrix.

A crude classification of memory hierarchies distinguishes between registers (small), cache (medium), and main memory (large).

To reduce data movements, we want to keep data in registers and cache as much as possible.

tiled Cholesky factorization

```
for $k = 1, 2, \ldots, p$ do
    DPOTF2($A_{k,k}, L_{k,k}$);                    —— $L_{k,k}$ := Cholesky($A_{k,k}$)
    for $i = k+1, \ldots, p$ do
        DTRSM($L_{k,k}, A_{i,k}, L_{i,k}$);            —— $L_{i,k} := A_{i,k} L_{k,k}^{-T}$
    end for;
    for $i = k+1, \ldots, p$ do
        for $j = k+1, \ldots, p$ do
            DGSMM($L_{i,k}, L_{j,k}, A_{i,j}$);        —— $A_{i,j} := A_{i,j} - L_{i,k} L_{j,k}^T$
        end for;
end for.
```

Parallel Gaussian Elimination

1. LU and Cholesky Factorization
   - factoring a square matrix
   - tiled Cholesky factorization
2. Blocked LU Factorization
   - deriving blocked formulations of LU
   - right and left looking LU factorizations
   - tiled algorithm for LU factorization
3. The PLASMA Software Library
   - Parallel Linear Algebra Software for Multicore Architectures
   - running an example

blocked LU factorization

The optimal size of the blocks is machine dependent.

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} = \begin{bmatrix} L_{1,1} & & \\ L_{2,1} & L_{2,2} & \\ L_{3,1} & L_{3,2} & L_{3,3} \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ & U_{2,2} & U_{2,3} \\ & & U_{3,3} \end{bmatrix}$$

Expanding the right hand side and equating to the matrix at the left gives formulations for the LU factorization.

$$\begin{array}{lll} A_{1,1} = L_{1,1}U_{1,1} & A_{1,2} = L_{1,1}U_{1,2} & A_{1,3} = L_{1,1}U_{1,3} \\ A_{2,1} = L_{2,1}U_{1,1} & A_{2,2} = L_{2,1}U_{1,2} + L_{2,2}U_{2,2} & A_{2,3} = L_{2,1}U_{1,3} + L_{2,2}U_{2,3} \\ A_{3,1} = L_{3,1}U_{1,1} & A_{3,2} = L_{3,1}U_{1,2} + L_{3,2}U_{2,2} & A_{3,3} = L_{3,1}U_{1,3} + L_{3,2}U_{2,3} \\ & & \qquad\qquad + L_{3,3}U_{3,3} \end{array}$$

right looking LU

We store the $L_{i,j}$'s and $U_{i,j}$'s in the original matrix:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} = \begin{bmatrix} L_{1,1} & & \\ L_{2,1} & I & \\ L_{3,1} & & I \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ & B_{2,2} & B_{2,3} \\ & B_{3,2} & B_{3,3} \end{bmatrix}$$

The matrices $B_{i,j}$'s are obtained after a first block LU step.
To find $L_{2,2}$, $L_{3,2}$, and $U_{2,2}$ we use

$$\left\{ \begin{array}{l} A_{2,2} = L_{2,1}U_{1,2} + L_{2,2}U_{2,2} \\ A_{3,2} = L_{3,1}U_{1,2} + L_{3,2}U_{2,2} \end{array} \right. \quad \text{and} \quad \left\{ \begin{array}{l} A_{2,2} = L_{2,1}U_{1,2} + B_{2,2} \\ A_{3,2} = L_{3,1}U_{1,2} + B_{3,2} \end{array} \right.$$

Eliminating $A_{2,2} - L_{2,1}U_{1,2}$ and $A_{3,2} - L_{3,1}U_{1,2}$ gives

$$\left\{ \begin{array}{l} B_{2,2} = L_{2,2}U_{2,2} \\ B_{3,2} = L_{3,2}U_{2,2} \end{array} \right.$$

Via LU on $B_{2,2}$ we obtain $L_{2,2}$ and $U_{2,2}$. Then: $L_{3,2} := B_{3,2}U_{2,2}^{-1}$.

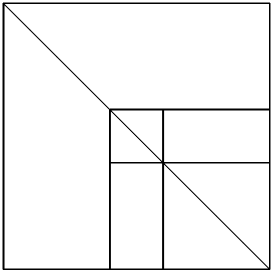
Parallel Gaussian Elimination

right and left looking

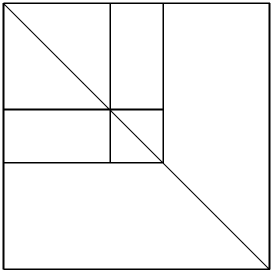
The formulas we derived are similar to the scalar case and are called *right looking*.

But we may organize the LU factorization differently:

right looking



left looking



What is good looking? Left is best for data access.

left looking formulas

Going from $P_1 A$ to $P_2 P_1 A$:

$$\begin{bmatrix} L_{1,1} & & \\ L_{2,1} & I & \\ L_{3,1} & & I \end{bmatrix} \begin{bmatrix} U_{1,1} & A_{1,2} & A_{1,3} \\ & A_{2,2} & A_{2,3} \\ & A_{3,2} & A_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} L_{1,1} & & \\ L_{2,1} & L_{2,2} & \\ L_{3,1} & L_{3,2} & I \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & A_{1,3} \\ & U_{2,2} & A_{2,3} \\ & & A_{3,3} \end{bmatrix}$$

We keep the original $A_{i,j}$'s and postpone updating to the right.

1. We get $U_{1,2}$ via $A_{1,2} = L_{1,1} U_{1,2}$ and compute $U_{1,2} = L_{1,1}^{-1} A_{1,2}$.
2. To compute $L_{2,2}$ and $L_{3,2}$ do $\begin{bmatrix} B_{2,2} \\ B_{3,2} \end{bmatrix} = \begin{bmatrix} A_{2,2} \\ A_{3,2} \end{bmatrix} - \begin{bmatrix} L_{2,1} \\ L_{3,1} \end{bmatrix} U_{1,2}$.

   and factor $P_2 \begin{bmatrix} B_{2,2} \\ B_{3,2} \end{bmatrix} = \begin{bmatrix} L_{2,2} \\ L_{3,2} \end{bmatrix} U_{2,2}$ as before.

   Replace $\begin{bmatrix} A_{2,3} \\ A_{3,3} \end{bmatrix} := P_2 \begin{bmatrix} A_{2,3} \\ A_{3,3} \end{bmatrix}$ and $\begin{bmatrix} L_{2,1} \\ L_{3,1} \end{bmatrix} := P_2 \begin{bmatrix} L_{2,1} \\ L_{3,1} \end{bmatrix}$.

Parallel Gaussian Elimination

tiled LU factorization

```
for $k = 1, 2, \ldots, p$ do
   DGETF($A_{k,k}, L_{k,k}, U_{k,k}, P_{k,k}$);   $-- L_{k,k}, U_{k,k}, P_{k,k} := \mathrm{LU}(A_{k,k})$
   for $j = k+1, \ldots, p$ do
      DGESSM($A_{k,j}, L_{k,k}, P_{k,k}, U_{k,j}$);   $-- U_{k,j} := L_{k,k}^{-1} P_{k,k} A_{k,j}$
   end for;
   for $i = k+1, \ldots, p$ do
      DTSTRF($U_{k,k}, A_{i,k}, P_{i,k}$);   $-- U_{k,k}, L_{i,k}, P_{i,k} := LU\left(\begin{bmatrix} U_{k,k} \\ A_{i,k} \end{bmatrix}\right)$
      for $j = k+1, \ldots, p$ do
         DSSSM($U_{k,j}, A_{i,j}, L_{i,k}, P_{i,k}$);   $-- \begin{bmatrix} U_{k,j} \\ A_{i,j} \end{bmatrix} := L_{i,k}^{-1} P_{i,k} \begin{bmatrix} U_{k,j} \\ A_{i,j} \end{bmatrix}$
      end for;
end for.
```

Parallel Gaussian Elimination

the PLASMA software library

Parallel Linear Algebra Software for Multicore Architectures.

- Software using FORTRAN and C.
- Design for efficiency on homogeneous multicore processors and multi-socket systems of multicore processors.
- Built using a small set of sequential routines as building blocks, referred to as *core BLAS*.
- Free to download from `http://icl.cs.utk.edu/plasma`.

Capabilities and limitations:

- Can solve dense linear systems and least squares problems.
- Unlike LAPACK, PLASMA currently does not solve eigenvalue or singular value problems and provide no support for band matrices.

Basic Linear Algebra Subprograms: BLAS

1. Level-1 BLAS: vector-vector operations, $O(n)$ cost.
   inner products, norms, $\mathbf{x} \pm \mathbf{y}$, $\alpha\mathbf{x} + \mathbf{y}$.

2. Level-2 BLAS: matrix-vector operations, $O(mn)$ cost.
   - $\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$
   - $A = A + \alpha\mathbf{x}\mathbf{y}^T$, rank one update
   - $\mathbf{x} = T^{-1}\mathbf{b}$, for T a triangular matrix

3. Level-3 BLAS: matrix-matrix operations, $O(kmn)$ cost.
   - $C = \alpha AB + \beta C$
   - $C = \alpha AA^T + \beta C$, rank k update of symmetric matrix
   - $B = \alpha TB$, for T a triangular matrix
   - $B = \alpha T^{-1}B$, solve linear system with many right hand sides

graph driven asynchronous execution

We view a blocked algorithm as a Directed Acyclic Graph (DAG):

- nodes are computational tasks performed in kernel subroutines;
- edges represent the dependencies among the tasks.

Given a DAG, tasks are scheduled asynchronously and independently, considering the dependencies imposed by the edges in the DAG.

A critical path in the DAG connects those nodes that have the highest number of outgoing edges.

The scheduling policy assigns higher priority to those tasks that lie on the critical path.

Parallel Gaussian Elimination

1. LU and Cholesky Factorization
   - factoring a square matrix
   - tiled Cholesky factorization
2. Blocked LU Factorization
   - deriving blocked formulations of LU
   - right and left looking LU factorizations
   - tiled algorithm for LU factorization
3. The PLASMA Software Library
   - Parallel Linear Algebra Software for Multicore Architectures
   - running an example

compiling example_cposv

The directory examples in
/usr/local/plasma-installer_2.6.0/build/plasma_2.6.0
contains example_cposv, an example for a Cholesky factorization of a symmetric positive definite matrix.
Running make as defined in the examples directory:

```
[root@kepler examples]# make example_cposv
gcc -O2 -DADD_ -I../include -I../quark \
-I/usr/local/plasma-installer_2.6.0/install/include -c \
example_cposv.c -o example_cposv.o
gfortran  example_cposv.o -o example_cposv -L../lib -lplasma \
-lcoreblasqw -lcoreblas -lplasma -L../quark -lquark \
-L/usr/local/plasma-installer_2.6.0/install/lib -lcblas \
-L/usr/local/plasma-installer_2.6.0/install/lib -llapacke \
-L/usr/local/plasma-installer_2.6.0/install/lib -ltmg -llapack \
-L/usr/local/plasma-installer_2.6.0/install/lib -lrefblas \
-lpthread -lm
[root@kepler examples]#
```

running `example_cposv`

Cholesky factorization with the dimension equal to 10,000.

```
[root@kepler examples]# time ./example_dposv
-- PLASMA is initialized to run on 2 cores.
============
Checking the Residual of the solution
-- ||Ax-B||_oo/((||A||_oo||x||_oo+||B||_oo).N.eps) = 2.710205e-03
-- The solution is CORRECT !
-- Run of DPOSV example successful !

real    1m22.271s
user    2m42.534s
sys     0m1.064s
[root@kepler examples]#
```

checking the speedups

The wall clock times for Cholesky on dimension 10,000:

#cores	real time	speedup
1	2m 40.913s	
2	1m 22.271s	1.96
4	42.621s	3.78
8	23.480s	6.85
16	12.647s	12.72

suggested reading

- E. Agullo, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, J. Langou, H. Ltaief, P. Luszczek, and A. YarKhan. **PLASMA Users' Guide. Parallel Linear Algebra Software for Multicore Architectures. Version 2.0**.
- A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. **A class of parallel tiled linear algebra algorithms for multicore architectures.** *Parallel Computing* 35: 38-53, 2009.

Summary + Exercises

In Wilkinson and Allen, parallel linear system solving is in §11.3.2.

Exercises:

1. Write your own parallel shared memory version of the Cholesky factorization, using OpenMP, Pthreads, or the Intel TBB.
2. Derive right looking LU factorization formulas with pivoting, i.e.: introducing permutation matrices P. Develop first the formulas for a 3-by-3 block matrix and then generalize the formulas into an algorithm for any p-by-p block matrix.
3. Take an appropriate example from the PLASMA installation to test the speedup of the multicore LU factorization routines.