HPC Challenge Awards

Class 2 – Productivity

Cleve Moler
The MathWorks, Inc.
SC|05, November 15, 2005

Parallel Computing with MATLAB

Major project at the MathWorks
Beta release sometime in 2006

Distributed memory multiprocessors
Single Program, Distributed Data
All processors run the same MATLAB program

"Think Matrices, Not Messages"

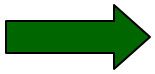
A MATLAB process is a "lab"

numlabs
labindex
parfor
dcolon
darray
gop
labSend
labReceive

Distributed colon operation

dcolon(1,1,10)

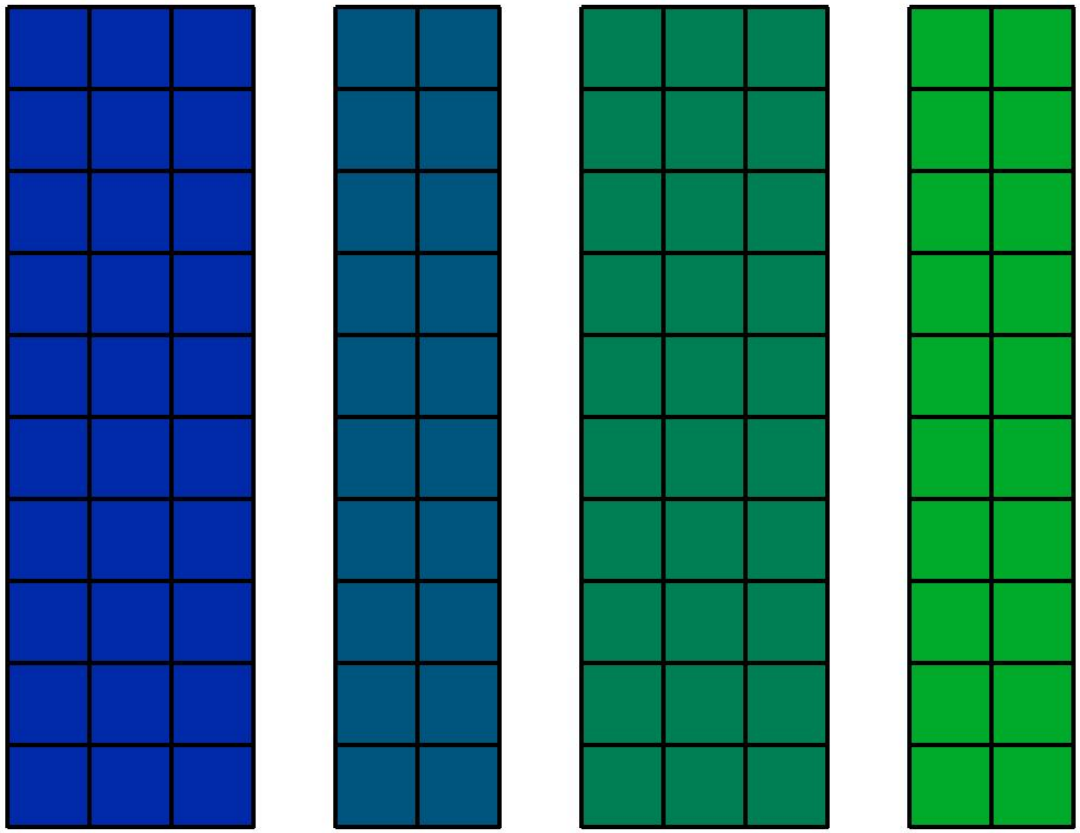
[1 2 3 4 5 6 7 8 9 10]



[1 2 3] [4 5] [6 7 8] [9 10]

Distributed arrays

```
A = darray(Aloc,[10,10],2)
```



Benchmarks

Cray XD1
5 dual processor Opterons
2.4 GHz
2 Gbytes/processor
Linux
MPICH2
AMD BLAS

numlabs = 10

Stream

```
% HPC_STREAM
% Each lab works independently.

m = 2000000;
b = randn(m,1);
c = randn(m,1);
alpha = randn;

% Repeat the test n times
n = 10;
t = zeros(n,1);
for k = 1:n
   tic
   a = b + alpha*c;
   t(k) = toc;
end

% Performance uses the best time
perf = 24*m/min(t)/1.e9
```

```
P>> HPC_stream

  1: perf =
  1:     1.9883
  2: perf =
  2:     1.9687
  3: perf =
  3:     1.9568
  4: perf =
  4:     1.8735
  5: perf =
  5:     2.0109
  6: perf =
  6:     1.9617
  7: perf =
  7:     1.9581
  8: perf =
  8:     1.9566
  9: perf =
  9:     1.9503
 10: perf =
 10:     1.9651
```

http://www.cs.virginia.edu/stream/ref.html

```
% gcc -O3 stream.c -o stream
% stream

Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:        1829.8259       0.0176       0.0175       0.0178
Scale:       1809.6447       0.0178       0.0177       0.0180
Add:         1973.6740       0.0245       0.0243       0.0246
Triad:       1916.1005       0.0252       0.0251       0.0253
```

Matrix Computations

- ScaLAPACK
  - Performance
  - Expertise
- M
  - Productivity
  - Sparse matrices
  - Improving performance

```
% HPC solve linear equations

for n = 200:200:15000
   A = drandn(n,n);
   b = ones(n,1);
   tic
      x = A\b;
   t = toc;
   mu = (2/3*n^3 + 2*n^2)/t/1.e9;
   r(1) = norm(A*x-b,inf)/(eps*norm(A,1)*n)
   r(2) = norm(A*x-b,inf)/(eps*norm(A,1)*norm(x,1))
   r(3) = norm(A*x-b,inf)/(eps*norm(A,inf)*norm(x,inf)*n)
   if labindex == 1;
      fprintf('%6d  %10.4f  %10.4f  %14.4e\n',n,t,mu,max(r));
   end
end
```

```
function x = mldivide(A,b)

% Factor the matrix

[L,U,p] = lu(A);
L = local(L);
U = local(U);
[n,n] = size(A);

% Pivot the right hand side

x = b(p,:);

% Solve L*y = b using successive local solves.

for p = 1:numlabs
   if labindex == p
      [s,t] = dcolonspan(1,1,n,p);
      i = (t+1):n;
      j = 1:(t-s+1);
      k = s:t;
      x(k,:) = L(k,j)\x(k,:);
      x(i,:) = x(i,:) - L(i,j)*x(k,:);
      if p < numlabs
         labSend(x,p+1)
      end
   elseif labindex == p+1
      x = labReceive(p);
   end
end
```

```
% Solve U*x = y using successive local solves.

for p = numlabs:-1:1
   if labindex == p
      [s,t] = dcolonspan(1,1,n,p);
      i = 1:(s-1);
      j = 1:(t-s+1);
      k = s:t;
      x(k,:) = U(k,j)\x(k,:);
      x(i,:) = x(i,:) - U(i,j)*x(k,:);
      if p > 1
         labSend(x,p-1)
      end
   elseif labindex == p-1
      x = labReceive(p);
   end
end

% Broadcast the result

x = labBroadcast(1,x);
```

```
function [L,A,piv] = lu(A)
%LU     Distributed LU factorization
%   [L,U,p] = lu(A) produces a distributed unit lower trapezoidal matrix L,
%   the same size as A, a distributed square upper triangular matrix U,
%   and a replicated permutation vector p, so that L*U = A(p,:)

[m,n] = size(A);
nlabs = numlabs;
piv = (1:m)';

% Swap blocks of columns to improve load balance.

A = swapblocks(A);
Aloc = local(A);
nloc = size(Aloc,2);

% Number of columns in "offdiagonal" subblocks.

w = floor(n/nlabs^2);

% Loop over the subblocks and the processors

s = 0;  % Number of rows already processed by ALL processors.
t = 0;  % Number of columns already processed by this processor.
```

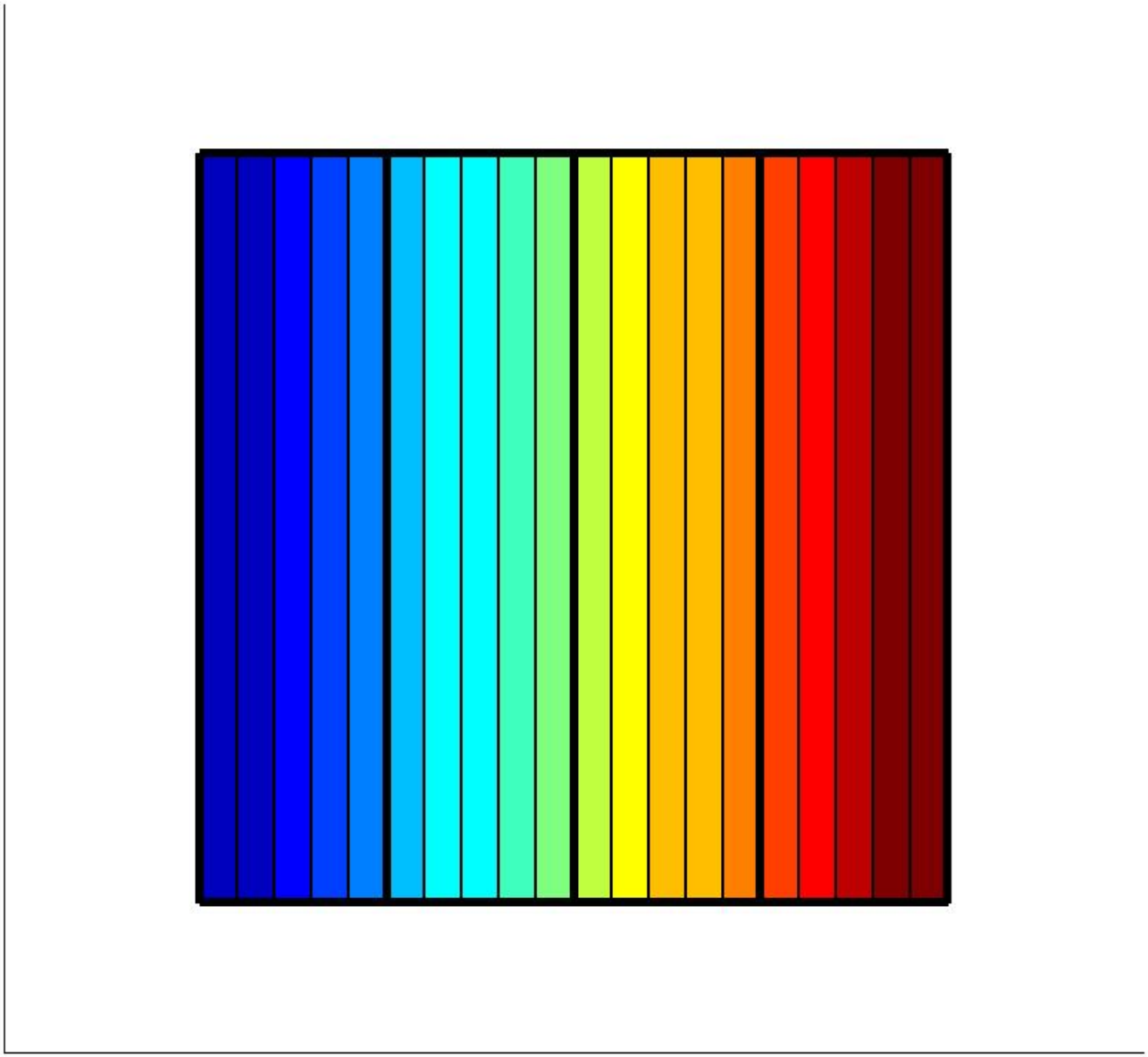
```
for p = 1:nlabs
   for r = 1:nlabs
      if p ~= r
         npr = w;
      else
         nlocr = ceil(r*n/nlabs)-ceil((r-1)*n/nlabs);
         npr = nlocr - (nlabs-1)*w;
      end

      if r == labindex

         % Compute the LU factorization of the (p,r)-th subblock
         j = (t+1):(t+npr);
         i = (s+1):m;
         [Aloc(i,j) pkr] = lu(Aloc(i,j),'econ');
         pkr = pkr+s;
         Lkr = tril(Aloc(i,j));
         Lkr(1:(m-s+1):(m-s)*npr) = 1;
         t = t + npr;
         j = [1:(t-npr) (t+1):nloc];

      else

         % Set aside storage for receive pivots and multipliers.
         pkr = zeros(m-s,1);
         Lkr = zeros(m-s,npr);
         j = 1:nloc;

      end
```

```
        % Broadcast pivots and multipliers.

        labBroadcast(r,pkr);
        labBroadcast(r,Lkr);

        % Apply pivots to all other subblocks.

        i = (s+1):m;
        piv(i) = piv(pkr);
        Aloc(i,j) = Aloc(pkr,j);

        % Apply multipliers to following subblocks.

        ik = 1:npr;
        iu = ik+s;
        ia = (s+npr+1):m;
        il = ia-s;
        j = (t+1):nloc;
        Aloc(iu,j) = Lkr(ik,ik)\Aloc(iu,j);
        Aloc(ia,j) = Aloc(ia,j) - Lkr(il,ik)*Aloc(iu,j);
        s = s + npr;

    end
end
A = darray(Aloc,2,[m n]);
```

```
% Undo the column swaps.

A = swapblocks(A);

% Break into lower and upper triangular matrices.

L = tril(A,-1) + deye(size(A));
if m > n
   Aloc = local(A);
   Aloc = Aloc(1:n,:);
   A = darray(Aloc,2,[n n]);
end
A = triu(A);
```

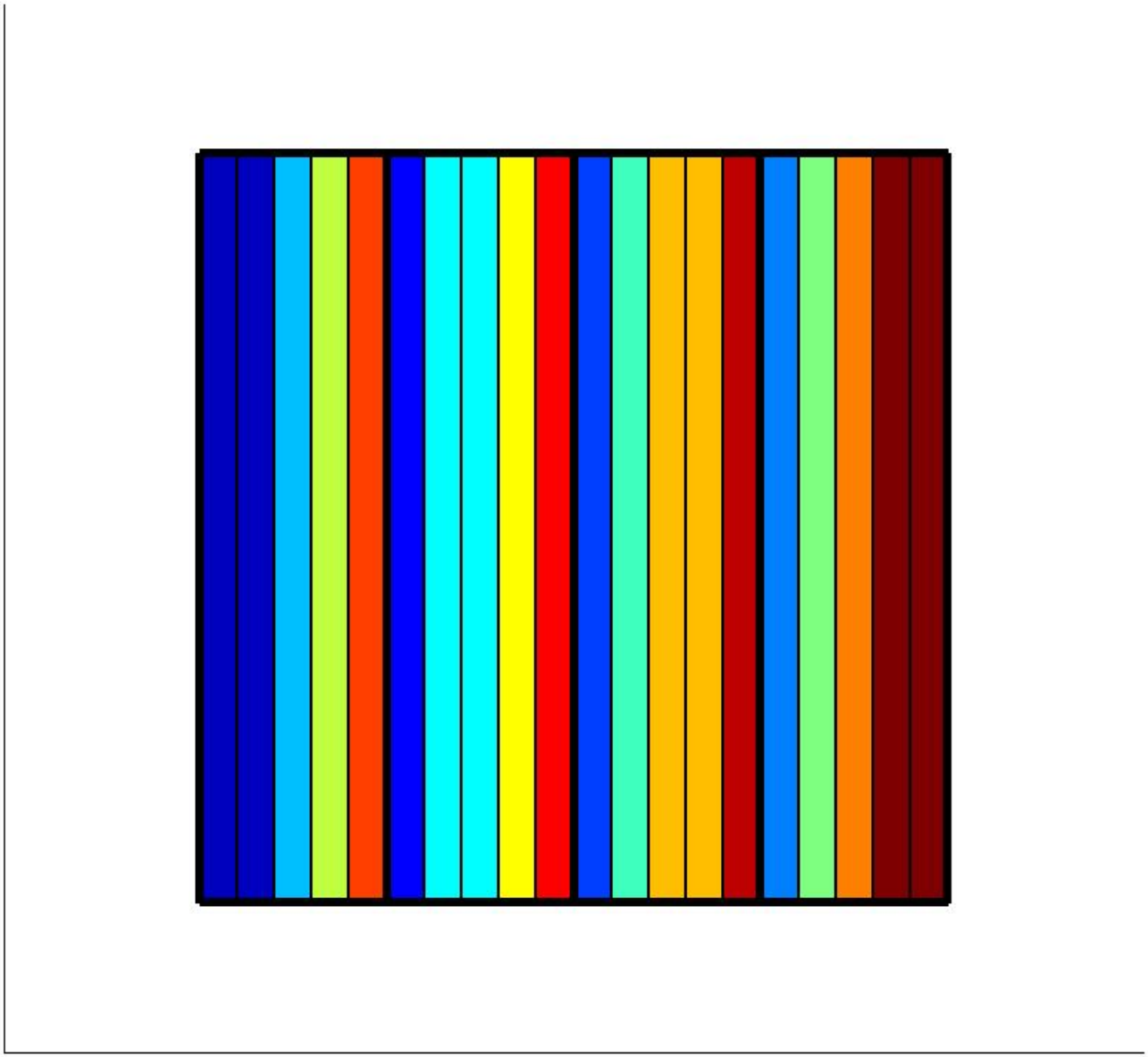
```
function A = swapblocks(A)
% SWAPBLOCKS  Redistribute columns of distributed array.
%    A = swapblocks(A)
%    Improves load balancing for lu and qr factorizations.
%    Swap blocks of columns of width floor(n/numlabs^2).

[m,n] = size(A);
nlabs = numlabs;
w = floor(n/nlabs^2);
Aloc = local(A);
nloc = size(Aloc,2);
for p = 1:nlabs
   if p == labindex
      for r = p+1:nlabs
         j = nloc - (nlabs+1-r)*w + (1:w);
         T = Aloc(:,j);
         labSend(T,r)
         Aloc(:,j) = labReceive(r);
      end
   elseif labindex > p
      j = (p-1)*w + (1:w);
      T = Aloc(:,j);
      Aloc(:,j) = labReceive(p);
      labSend(T,p);
   end
end
A = darray(Aloc,2,[m n]);
```

Block column distribution

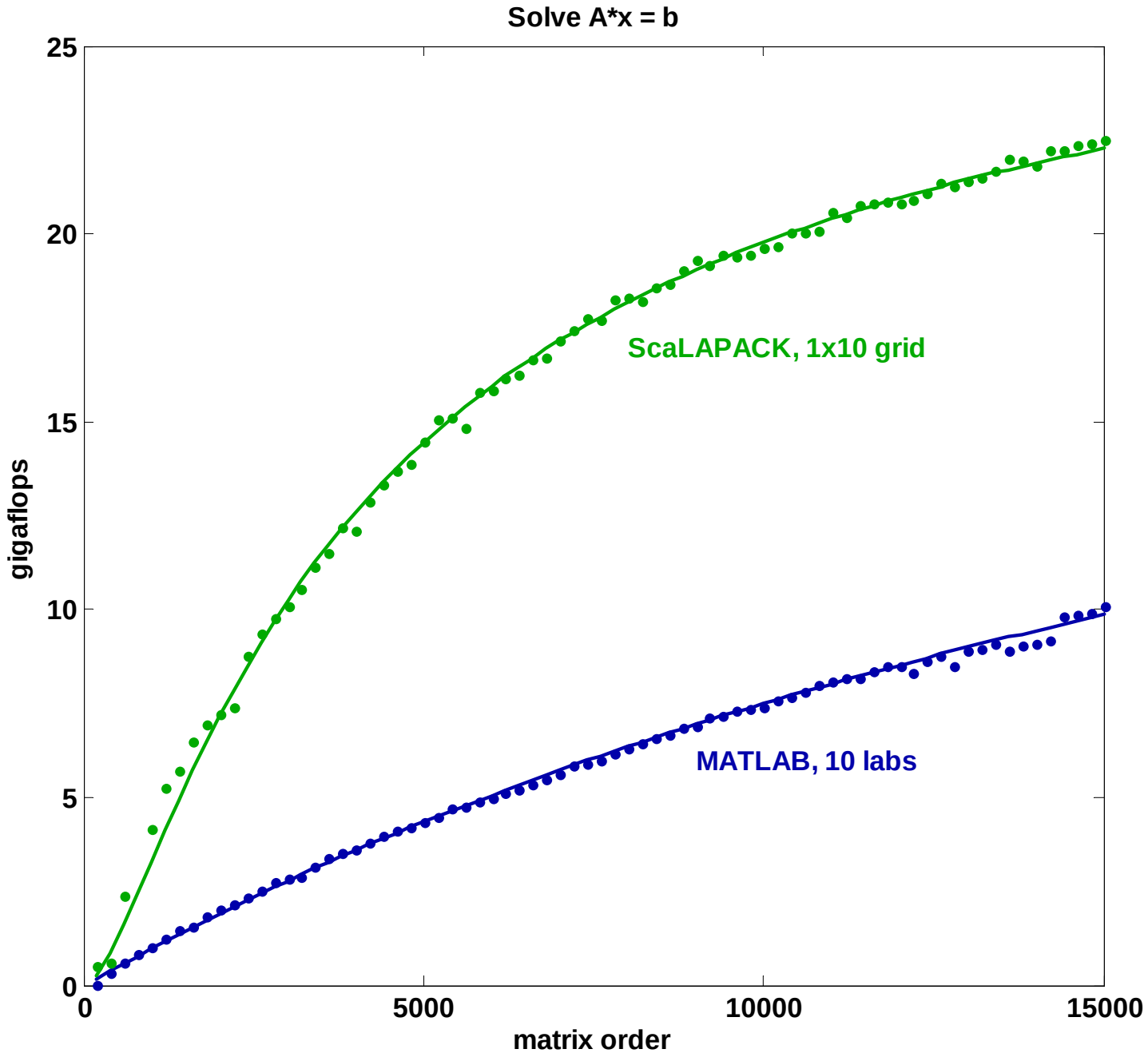


n = 20, numlabs = 4

Swap subblocks distribution

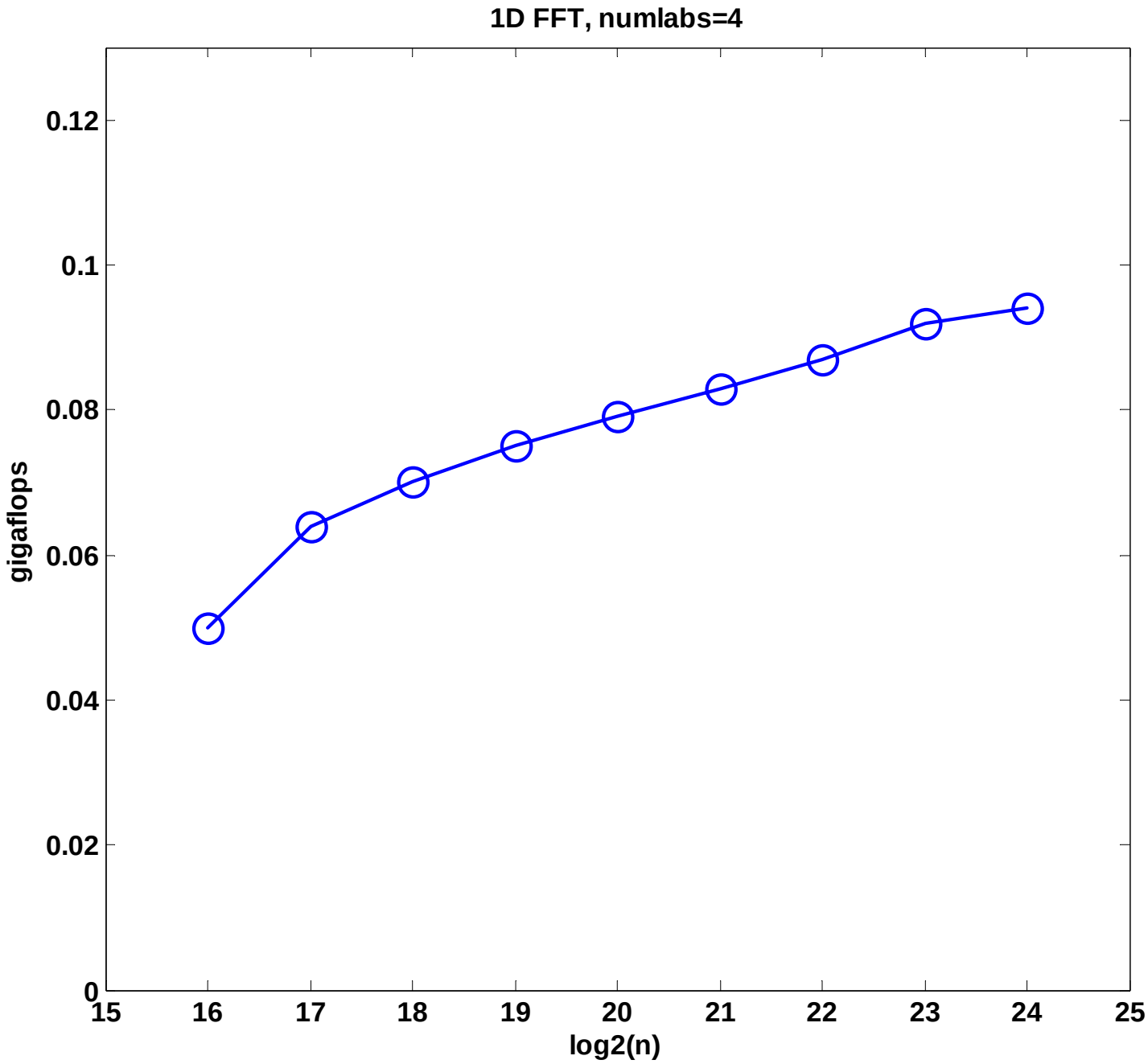


n = 20, numlabs = 4


Solve A*x = b
25
20
15
10
5
0
gigaflops
0
5000
10000
15000
matrix order
ScaLAPACK, 1x10 grid
MATLAB, 10 labs

FFT

```
function x = fft(x)
% FFT  Distributed one-dimensional finite Fourier transform.
%      fft(x) where length(x) is a multiple of numlabs.

% Reshape to matrix with numlabs rows
s = size(x);
n = prod(s);
m = n/numlabs;
x = reshape(x,numlabs,m);

% Unconjugated matrix transpose
x = x.';

% Local 1-D FFTs
xloc = fft(local(x));

% Local twiddle factors
omega = exp(2*pi*i*(labindex-1)/n);
xloc = (omega.^(0:m-1)') .* xloc;
x = darray(xloc,distributor(x),size(x));

% Unconjugated matrix transpose
x = x.';

% Local 1-D FFTs
xloc = fft(local(x));

% Return distributed row or column vector
x = darray(xloc,distributor(x),size(x));
x = x.';
x = reshape(x,s);
```


1D FFT, numlabs=4
gigaflops
log2(n)
0
0.02
0.04
0.06
0.08
0.1
0.12
15
16
17
18
19
20
21
22
23
24
25

RandomAccess

```
function T = randomaccess(T,nu)
% HPC_RandomAccess
% T = randomaccess(T,nu)
% runs the random access algorithm for nu steps.

mask = n-1;
ran = uint64(1);
for k = 1:nu
   if bitget(ran,64)
      ran = bitxor(bitshift(ran,1),poly);
   else
      ran = bitshift(ran,1);
   end
   idx = double(bitand(mask,ran))+1;
   T(idx) = bitxor(T(idx),ran);
end
```

HPCC Executable Source Lines

Stream	*16*
A\b	*148*
FFT	*24*
RandomAccess	*23*