HPC Challenge Awards

Class 2 – Productivity

Cleve Moler
The MathWorks, Inc.
SC|06, November 14, 2006

Parallel Computing with MATLAB

Distributed Computing Toolbox

Version 3.0 (2006b) is available now.

Version ?.? (2007a) is in development.

Parallel Model

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
- labSendReceive

MathWorks HPC Lab

Manufactured by Rackable Systems
16 dual socket, dual core Opteron 285s
2.6 GHz
4 Gbytes/node
Gigabit Ethernet
Linux
AMD ACML math library
MPICH2

numlabs = 1 … 64

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;
disp(perf)
```

```
P>> numlabs

      16

P>> stream

  1:     2.4539
  2:     2.4784
  3:     2.4260
  4:     2.4303
  5:     2.4236
  6:     2.4758
  7:     2.4194
  8:     2.4306
  9:     2.4353
 10:     2.4486
 11:     2.4409
 12:     2.4608
 13:     2.4682
 14:     2.4516
 15:     2.4315
 16:     2.4341
```

```
P>> numlabs

      64

P>> stream

  1:     1.5969
  2:     1.6194
  3:     1.6019
       .
       .
       .
 12:     1.7027
       .
       .
       .
 43:     1.5944
       .
       .
       .
 62:     1.6796
 63:     1.6250
 64:     1.6019
```

```
A\b
```

Matrix Computations

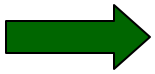
- M
  - Productivity
  - Sparse matrices
  - Improving performance
- ScaLAPACK
  - Performance
  - Expertise

Redistribute on the fly

- M
  - 1d unblocked noncyclic
  - Rows or columns
  - Efficient for common array operations
- ScaLAPACK
  - 2d block cyclic
  - Scales to many processors

Distributed colon operation

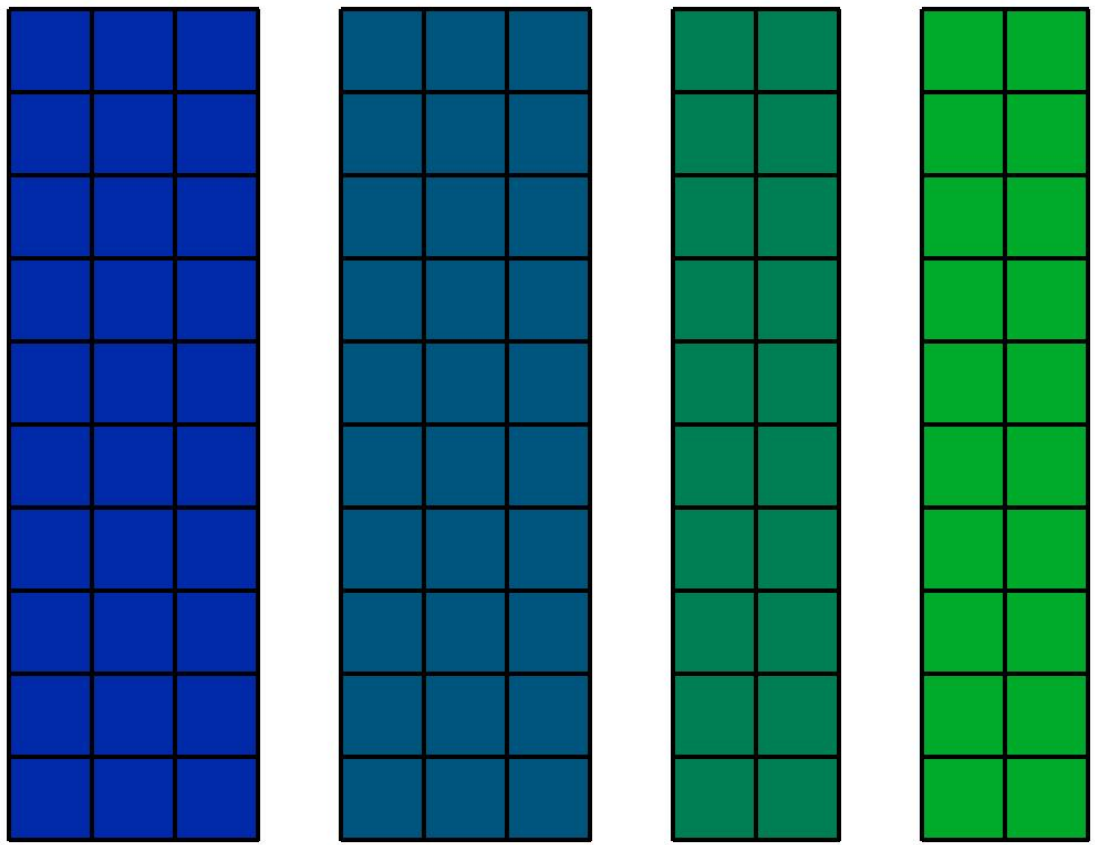
```
dcolon(1,1,10)

[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)
```



```
function B = mat2scala(A)

%
%   A = MATLAB dcolon column distribution
%
%   11  12 | 13 14 | 15 16 | 17 | 18 | 19
%   21  22 | 23 24 | 25 26 | 27 | 28 | 29
%   31  32 | 33 34 | 35 36 | 37 | 38 | 39
%   41  42 | 43 44 | 45 46 | 47 | 48 | 49
%   51  52 | 53 54 | 55 56 | 57 | 58 | 59
%   61  62 | 63 64 | 65 66 | 67 | 68 | 69
%   71  72 | 73 74 | 75 76 | 77 | 78 | 79
%   81  82 | 83 84 | 85 86 | 87 | 88 | 89
%   91  92 | 93 94 | 95 96 | 97 | 98 | 99
%
%   B = ScaLAPACK distribution
%
%   11  12 . 17  18 | 13  14 . 19 | 15  16
%   21  22 . 27  28 | 23  24 . 29 | 25  26
%   ....................................
%   51  52 . 57  58 | 53  54 . 59 | 55  56
%   61  62 . 67  68 | 63  64 . 69 | 65  66
%   ....................................
%   91  92 . 97  98 | 93  94 . 99 | 95  96
%   ------------------------------------
%   31  32 . 37  38 | 33  34 . 39 | 35  36
%   41  42 . 47  48 | 43  44 . 49 | 45  46
%   ....................................
%   71  72 . 77  78 | 73  74 . 79 | 75  76
%   81  82 . 87  88 | 83  84 . 89 | 85  86
```

Scaled speedup

If

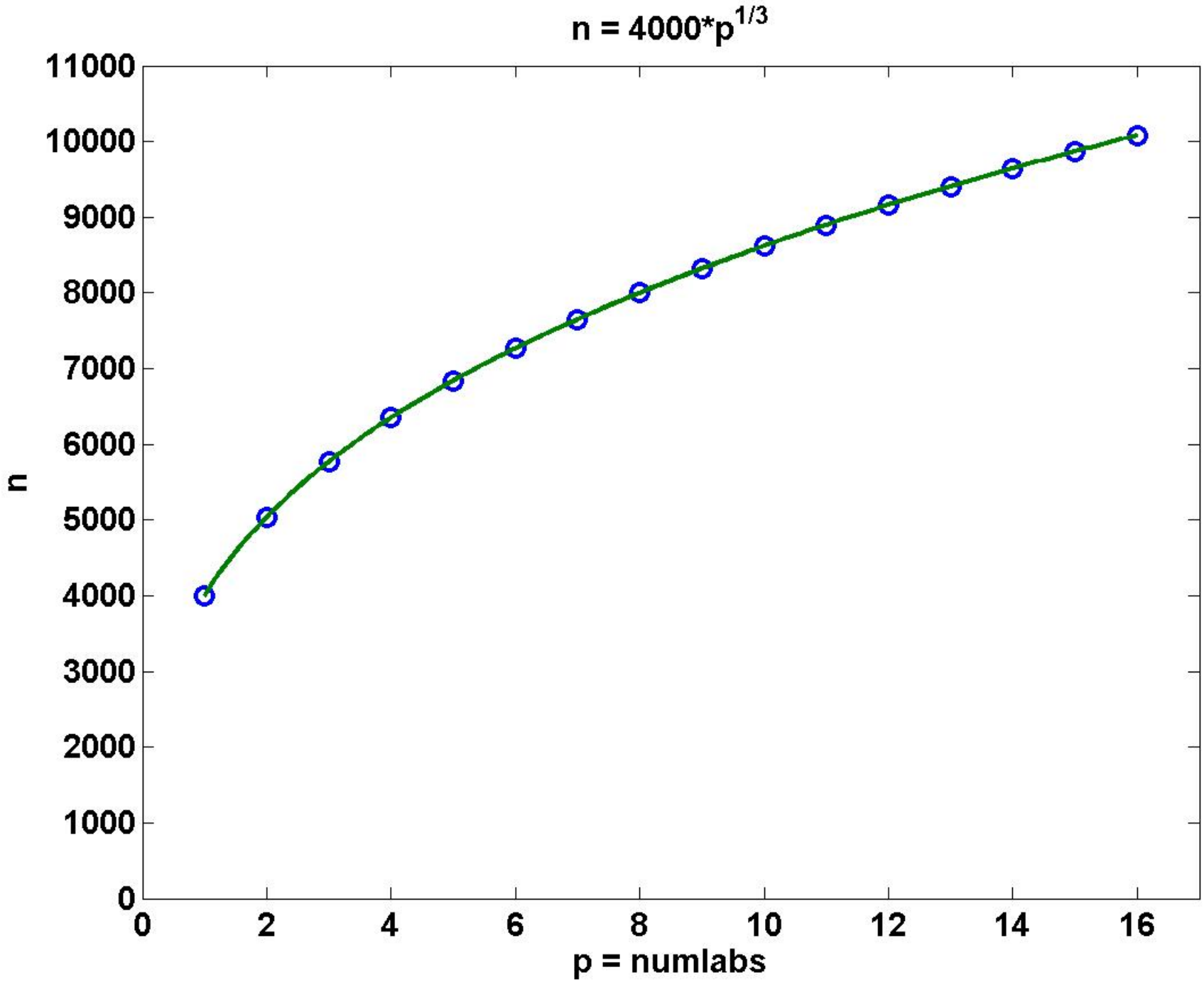
p = numlabs

$n \sim p^{1/3}$

then, with perfect speedup

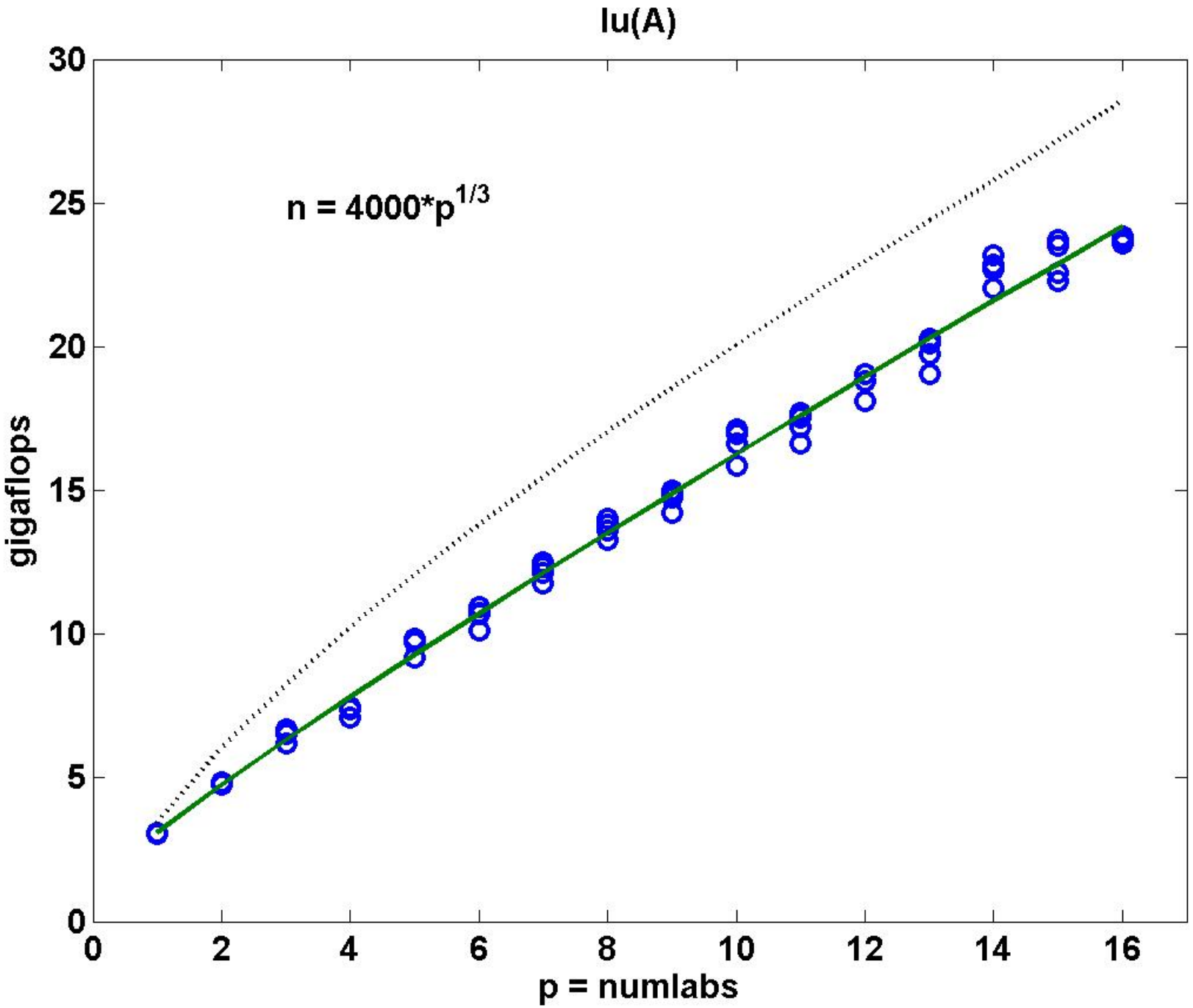
time ~ constant

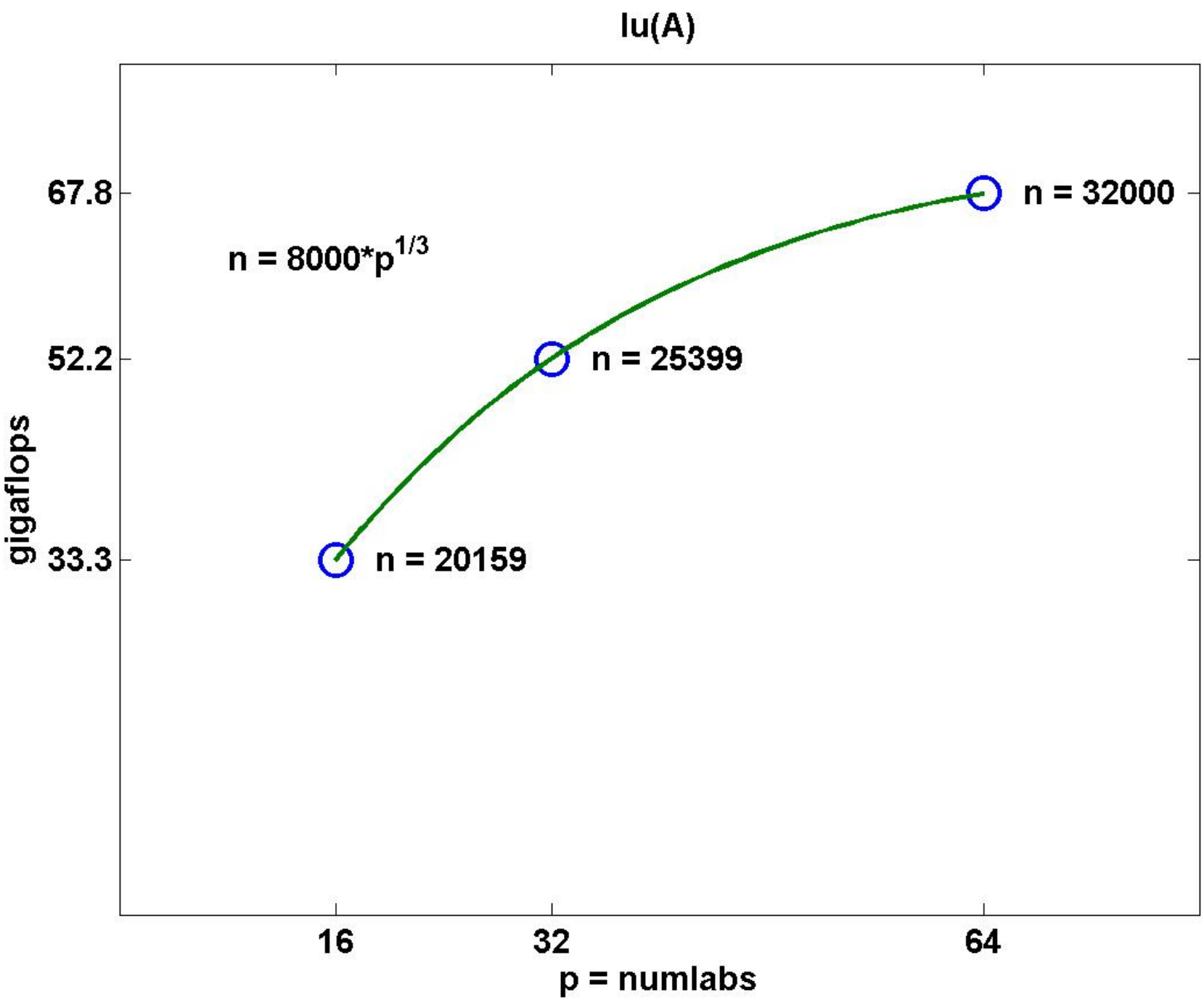
megaflops ~ p


n = 4000*p^1/3
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
n
0
2
4
6
8
10
12
14
16
p = numlabs

Main program

```
% HPC solve linear equations

   n = ceil(4000*numlabs^(1/3));
   A = randn(n,n,darray);
   b = ones(n,1);
   tic
      x = A\b;
   t = toc;
   gf = (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,gf,max(r));
   end
```


lu(A)
n = 4000*p^1/3
gigaflops
p = numlabs
0
5
10
15
20
25
30
0
2
4
6
8
10
12
14
16

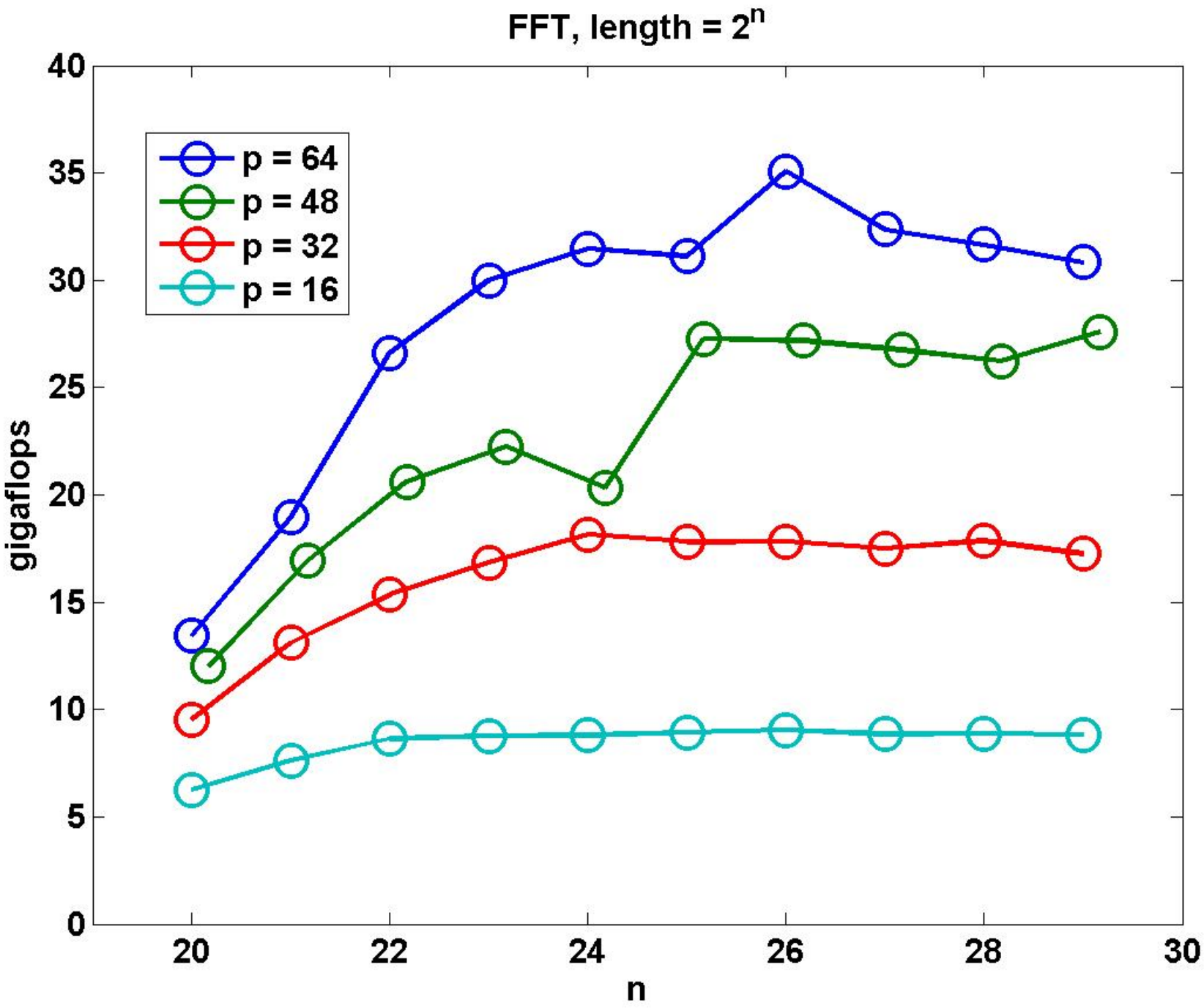

lu(A)
67.8
52.2
33.3
gigaflops
n = 8000*p^1/3
n = 32000
n = 25399
n = 20159
16
32
64
p = numlabs

FFT

FFT main program

```
function xfft(n)

% Create a complex row vector, distributed over the labs.
x = rand(1,n,darray()) + rand(1,n,darray())*i;

% Time the forward FFT
tic
y = fft1(x);
t = toc;

% Performance in gigaflops
perf = 5*n*log2(n)/t/1.e9;

% Error is scaled by roundoff level
z = ifft1(y);
err = norm(x-z,inf)/(log2(n)*eps);

fprintf('%8d %8d %8.3f %8.3f\n',numlabs,n,perf,err)
```

```
function x = fft1(x)
% FFT1  Distributed one-dimensional finite Fourier transform.
%    FFT1(x) where x is a distributed row vector
%            whose length is a multiple of numlabs.

n = length(x);
m = n/numlabs;
d = distribdim(x);

% Reshape into two-dimensional array with numlab rows
x = reshape(x,numlabs,m);

% Transpose
x = redistribute(x.',d);

% Local one-dimensional FFT
xloc = fft(local(x));

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

% Transpose again
x = redistribute(x.',d);

% More local one-dimensional FFTs
xloc = fft(local(x));

% Assemble final result
x = darray(xloc,d);
x = redistribute(x.',d);
x = reshape(x,1,n);
```


FFT, length = 2^n
gigaflops
n
p = 64
p = 48
p = 32
p = 16
0
5
10
15
20
25
30
35
40
20
22
24
26
28
30