Parallelism and Puzzles

Cleve Moler

SIAM Annual Meeting

Denver, July 8, 2009

Part I

My Experiences with Parallel Computing

Conclusion

More Powerful Computers

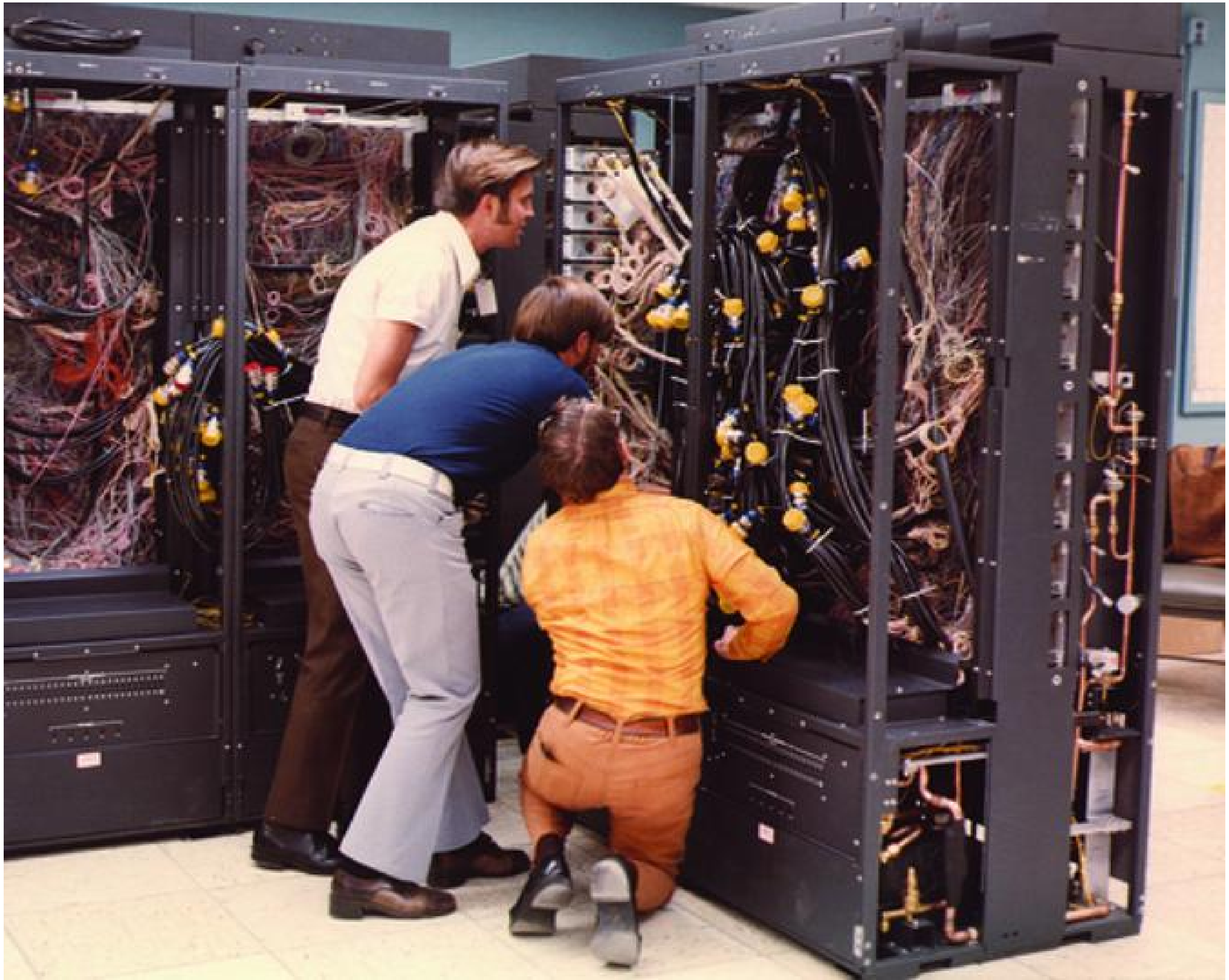
=>

Coarser Granularity

=>

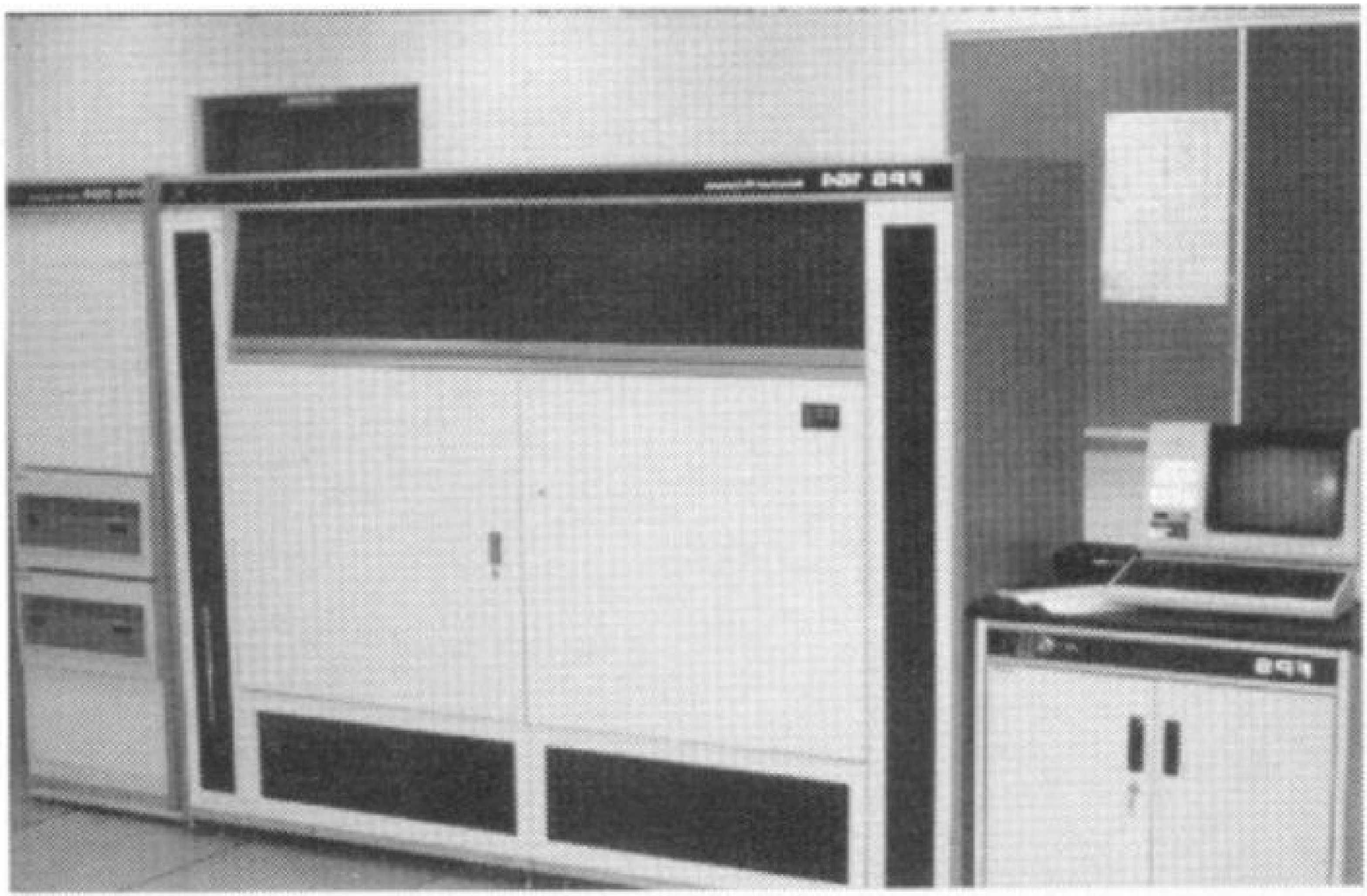
Embarrassingly Parallel

CDC 7600
Early 1970's



Cray 1
Late 1970's

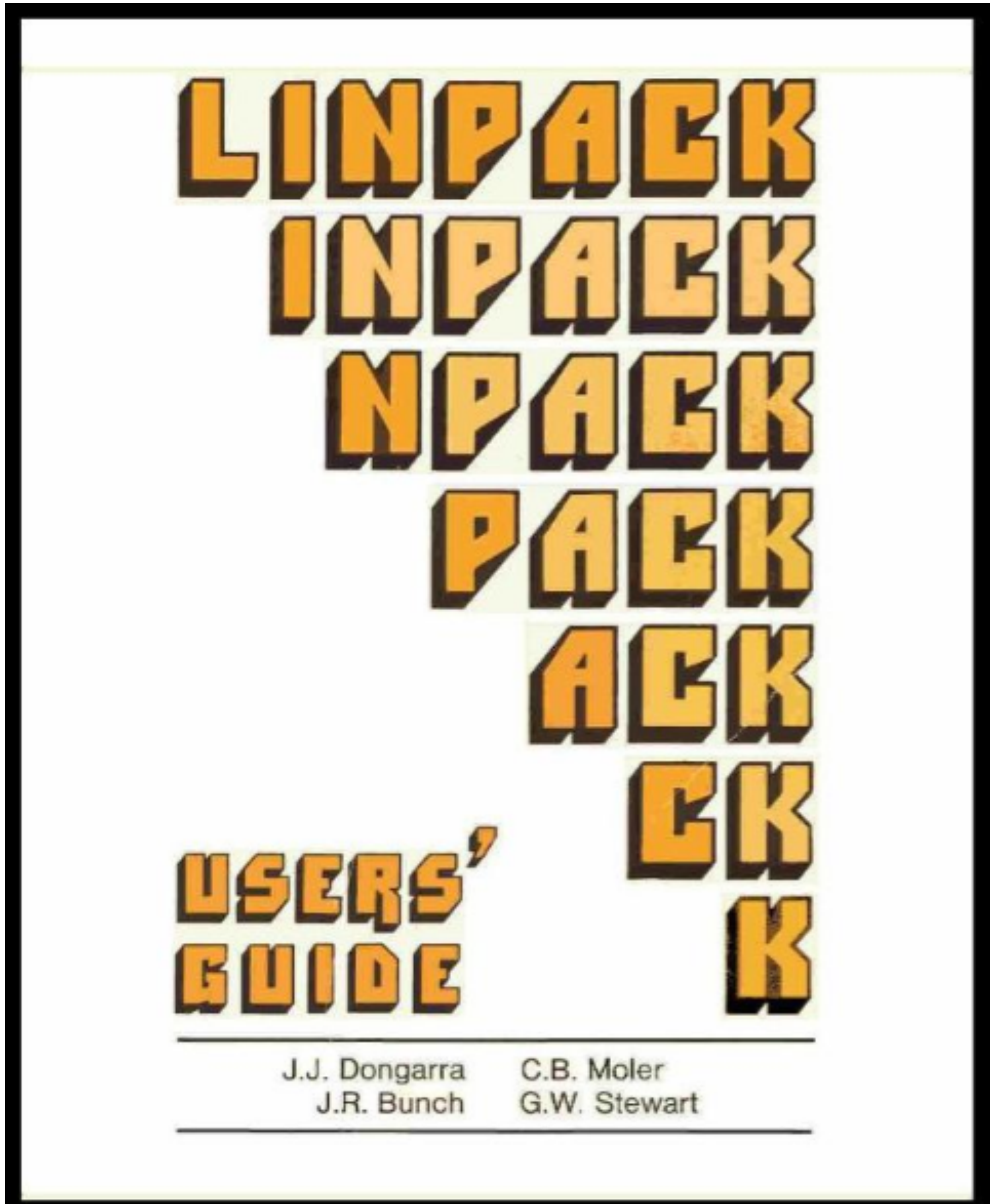
FPS 164
Early 1980's



1970’s – 1980’s

“Parallel”
≡
General purpose ops
and floating point ops
done simultaneously.

1979



LINPACK
INPACK
NPACK
PACK
ACK
CK
K
USERS' GUIDE
J.J. Dongarra
J.R. Bunch
C.B. Moler
G.W. Stewart

LINPACK Benchmark

UNIT = 10**6 TIME/(1/3 100**3 + 100**2)

Facility	TIME N=100 secs.	UNIT micro- secs.	Computer	Type	Compiler
NCAR	.049	0.14	CRAY-1	S	CFT, Assembly BLAS
LASL	.148	0.43	CDC 7600	S	FTN, Assembly BLAS
NCAR	.192	0.56	CRAY-1	S	CFT
LASL	.210	0.61	CDC 7600	S	FTN
Argonne	.297	0.86	IBM 370/195	D	H
NCAR	.359	1.05	CDC 7600	S	Local
Argonne	.388	1.33	IBM 3033	D	H
NASA Langley	.489	1.42	CDC Cyber 175	S	FTN
U. Ill. Urbana	.506	1.47	CDC Cyber 175	S	Ext. 4.6
LLL	.554	1.61	CDC 7600	S	CHAT, No optimize
SLAC	.579	1.69	IBM 370/168	D	H Ext., Fast mult.
Michigan	.631	1.84	Amdahl 470/V6	D	H
Toronto	.890	2.59	IBM 370/165	D	H Ext., Fast mult.
Northwestern	1.44	4.20	CDC 6600	S	FTN
Texas	1.93*	5.63	CDC 6600	S	RUN
China Lake	1.95*	5.69	Univac 1110	S	V
Yale	2.59	7.53	DEC KL-20	S	F20
Bell Labs	3.46	10.1	Honeywell 6080	S	Y
Wisconsin	3.49	10.1	Univac 1110	S	V
Iowa State	3.54	10.2	Itel AS/5 mod3	D	H
U. Ill. Chicago	4.10	11.9	IBM 370/158	D	G1
Purdue	5.69	16.6	CDC 6500	S	FUN
U. C. San Diego	13.1	38.2	Burroughs 6700	S	H
Yale	17.1*	49.9	DEC KA-10	S	F40

* TIME(100) = (100/75)**3 SGEFA(75) + (100/75)**2 SGESL(75)

$\frac{2}{3} n^3 + 2n^2$ ops / time

UNIT = 10**6 TIME/(1/3 100**3 -

Facility		TIME N=100 secs.	UNIT micro- secs.	Computer
NCAR	14.0	.049	0.14	CRAY-1
LASL	4.64	.148	0.43	CDC 7600
NCAR	3.54	.192	0.56	CRAY-1
LASL	3.27	.210	0.61	CDC 7600
Argonne	2.31	.297	0.86	IBM 370/
NCAR	1.91	.359	1.05	CDC 7600
Argonne	1.77	.388	1.33	IBM 3033
NASA Langley	1.40	.489	1.42	CDC Cyber
U. Ill. Urbana	1.36	.506	1.47	CDC Cyber
LLL	1.24	.554	1.61	CDC 7600
SLAC	1.19	.579	1.69	IBM 370/
Michigan	1.09	.631	1.84	Amdahl 4
Toronto	.772	.890	2.59	IBM 370/
Northwestern	.477	1.44	4.20	CDC 6600
Texas	.356	1.93*	5.63	CDC 6600

U. Ill. Urbana	.506	1.47	CDC Cybe
LLL	.554	1.61	CDC 7600
SLAC	.579	1.69	IBM 370/
Michigan	.631	1.84	Amdahl 4
Toronto	.690	2.59	IBM 370/
Northwestern	1.44	4.20	CDC 6600
Texas	1.93*	5.63	CDC 6600
China Lake	1.95*	5.69	Univac 1
Yale	2.59	7.53	DEC KL-2
Bell Labs	3.46	10.1	Honeywel
Wisconsin	3.49	10.1	Univac 1
Iowa State	3.54	10.2	Itel AS/
U. Ill. Chicago	4.10	11.9	IBM 370/
Purdue	5.69	16.6	CDC 6500
U. C. San Diego	13.1	38.2	Burrough
Yale	17.1*	49.9	DEC KA-1

* TIME(100) = (100/75)**3 SGEFA(75) +

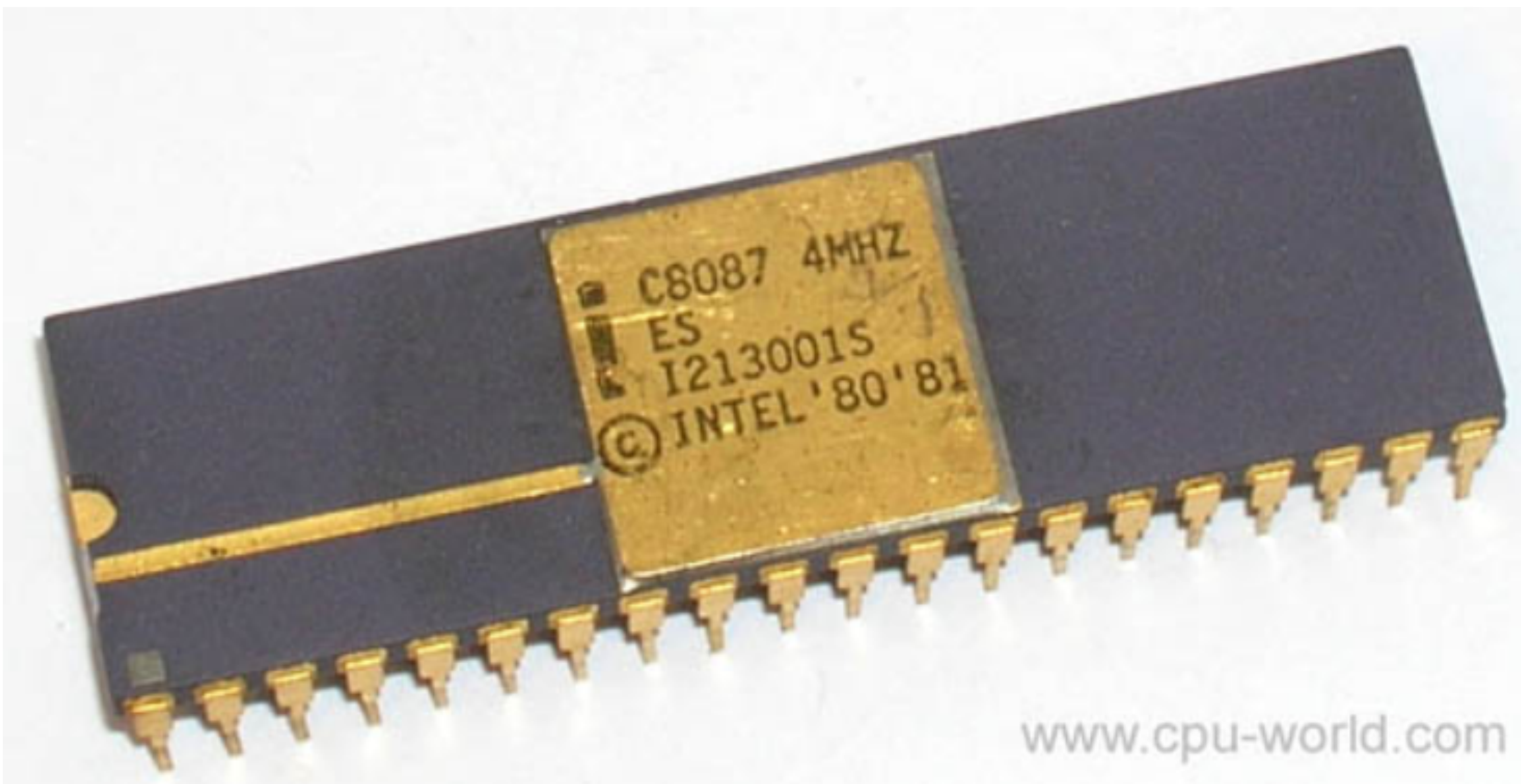
TOP500®

SUPERCOMPUTER SITES

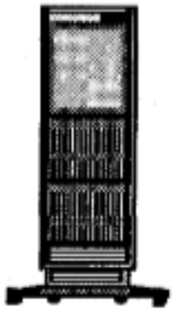
http://www.top500.org/

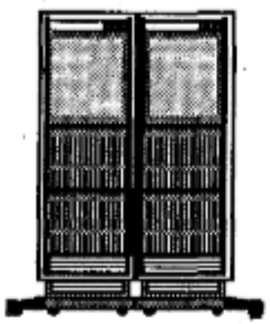
Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2008 IBM	129600	1105.00	1456.70	2483.47
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT5 QC 2.3 GHz / 2008 Cray Inc.	150152	1059.00	1381.40	6950.60
3	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70	2268.00
4	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0/2.66 GHz / 2008 SGI	51200	487.01	608.83	2090.00
5	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
	National Institute for Computational	Kraken XT5 - Cray				

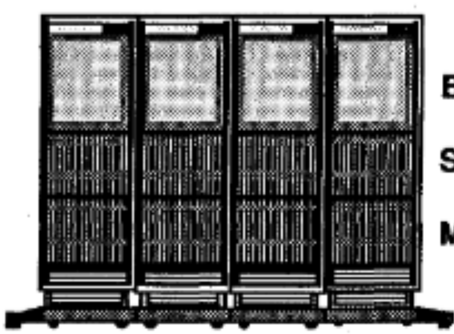
1980
Intel 8087
~50 kFLOPs



The iPSC System Family







	Name	Nodes	Memory	MFLOPS	Price
Base System	iPSC/d5	32	16 MBytes	2	$171.5K
Memory System	iPSC/d4-MX	16	72 MBytes	1	$184.4K
Numeric System	iPSC/d4-VX	16	24 MBytes	106	$296.1K
Base System	iPSC/d6	64	32 MBytes	4	$293.5K
Memory System	iPSC/d5-MX	32	144 MBytes	2	$311.3K
Numeric System	iPSC/d5-VX	32	48 MBytes	212	$516.7K
Base System	iPSC/d7	128	64 MBytes	8	$524.6K
Symbolic System	iPSC/d6-MX	64	288 MBytes	4	$558.2K
Memory System	iPSC/d6-VX	64	96 MBytes	424	$947.5K

DISTRIBUTED GAUSSIAN ELIMINATION

n = order of matrix
p = number of processors
id = individual processor index
m = number of columns in id-th processor
$\quad = \lceil n/p \rceil$ or $\lfloor n/p \rfloor$ if $id <$ or $\geq$ (n mod p)
A = distributed matrix, stored in n by m array on each processor

```
l = 1
for k = 1 to n do

   if id = (k-1) mod p then
      find pivot in l-th column of A
      e = - (portion of l-th column of A ) / pivot
      send e to all other processors
      l = l + 1
   else
      wait to recv e
   endif

   for j = l to m do
      s = a_{k,j}
      for i = k+1 to n do
         a_{i,j} = a_{i,j} + s·e_i
      end i loop
   end j loop

end k loop
```

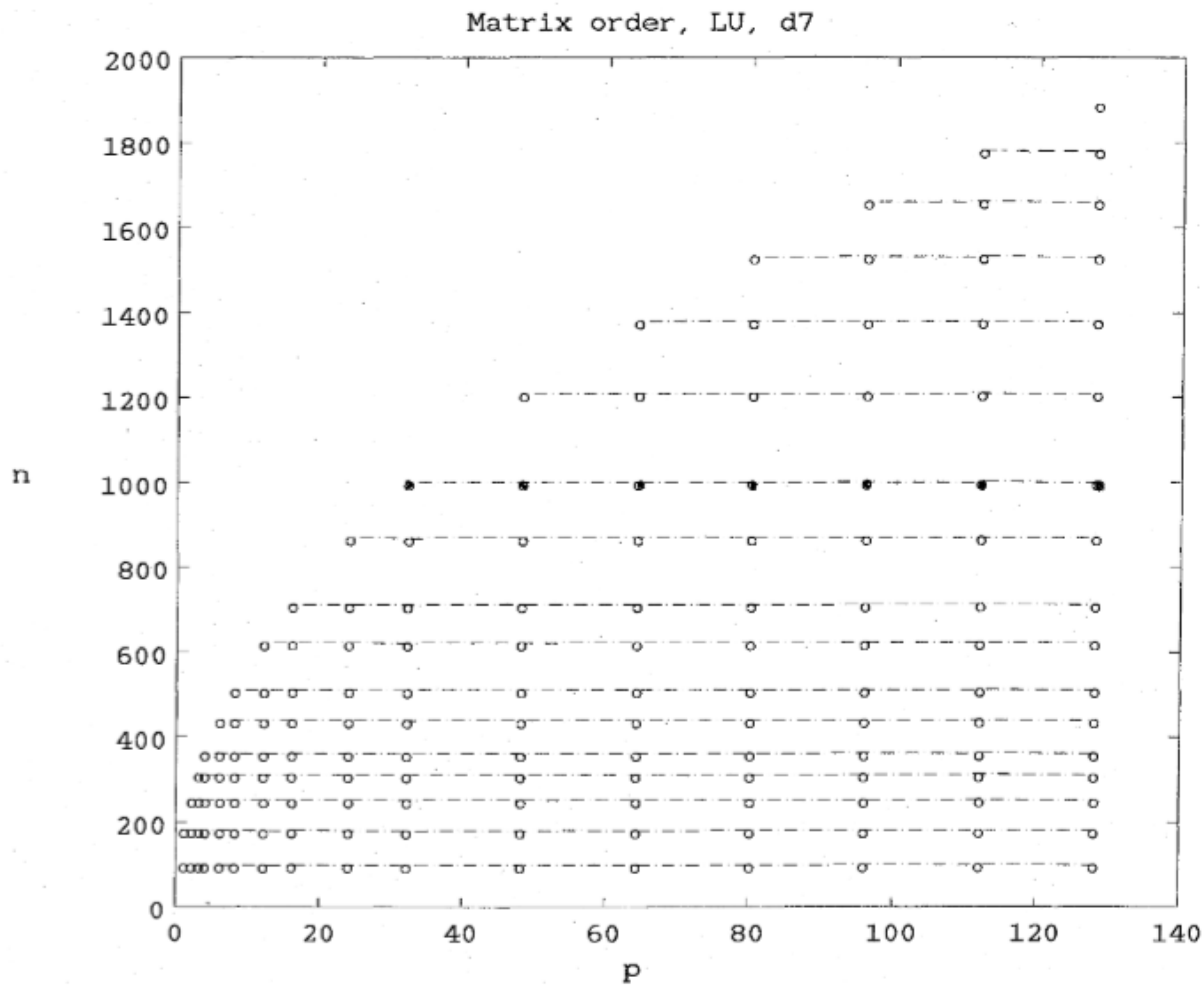
Memory Considerations

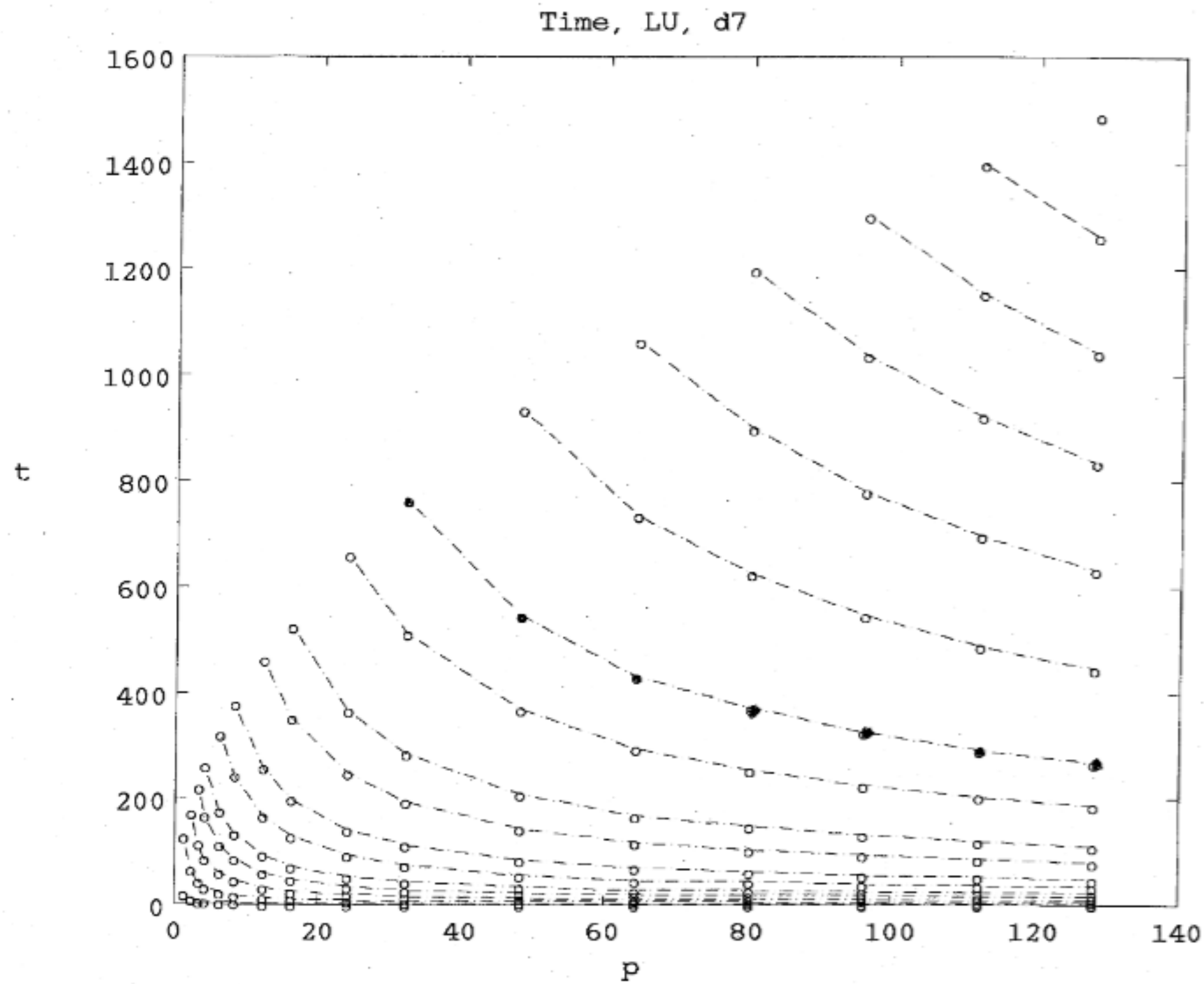
M = number of 64-bit words available per processor

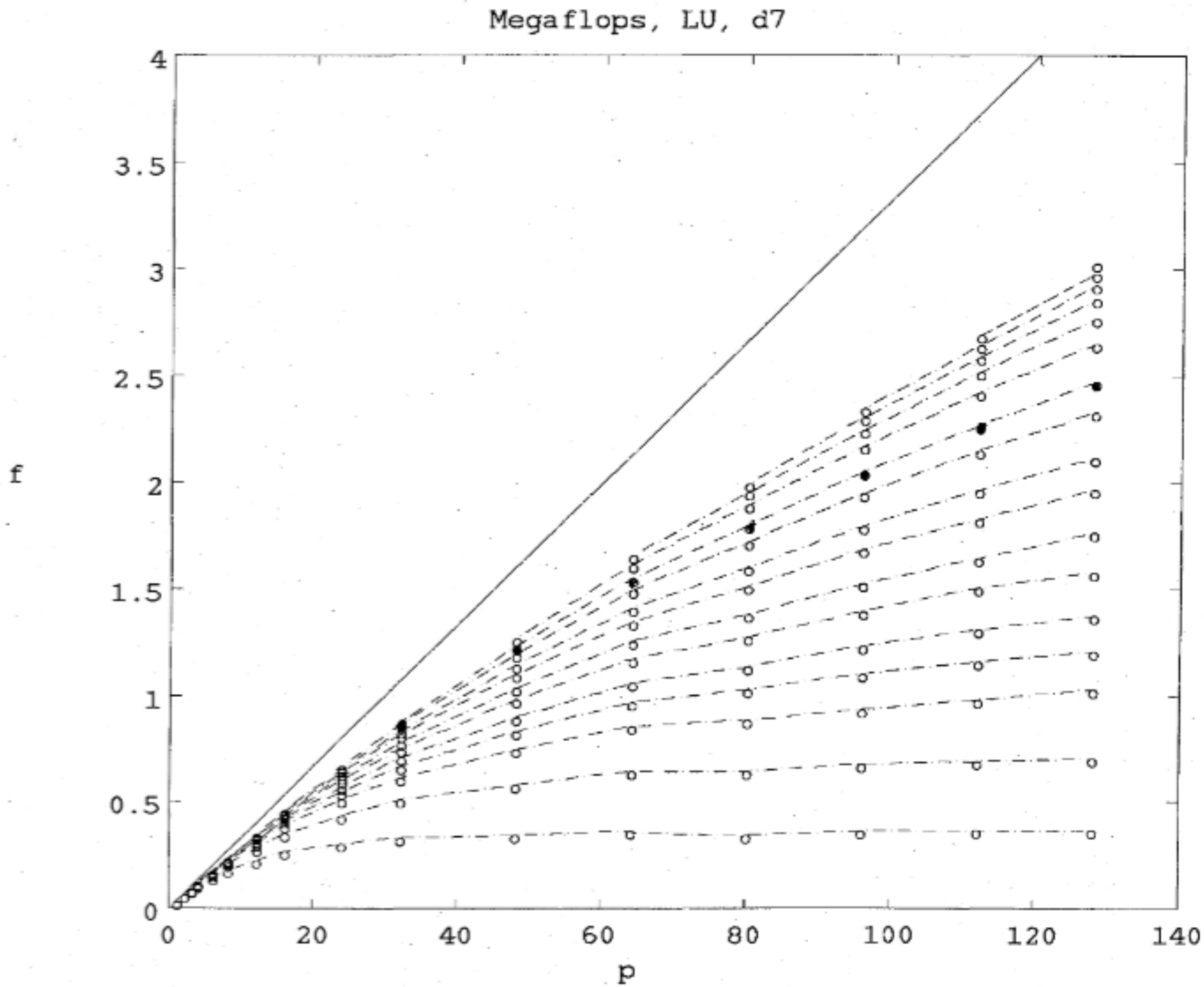
iPSC	M(thousands)
Standard	36
Vector	106
Memory	512

$$n \lceil n/p \rceil + 3n \leq M$$

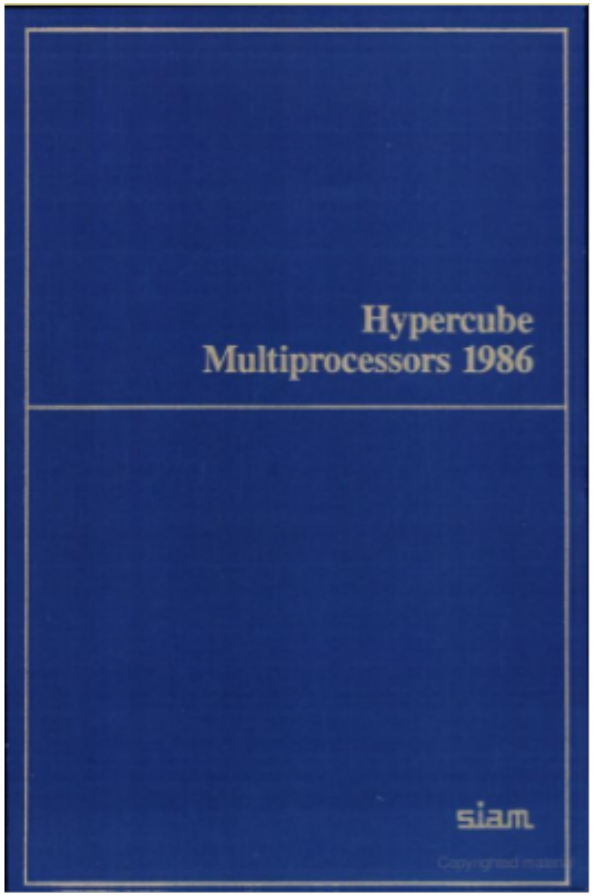
$$n_{\max} = \sqrt{pM + (2p)^2} - (2p)$$


Matrix order, LU, d7
n
p
2000
1800
1600
1400
1200
1000
800
600
400
200
0
20
40
60
80
100
120
140


Time, LU, d7
t
p
1600
1400
1200
1000
800
600
400
200
0
0
20
40
60
80
100
120
140


Megaflops, LU, d7
f
p
0
0.5
1
1.5
2
2.5
3
3.5
4
0
20
40
60
80
100
120
140

1986



Mike Heath, editor,
“Proceedings of the First Conference on Hypercube Multiprocessors Knoxville, Tennessee, 1985."

“Embarrassingly Parallel”

One important way in which LINPACK and EISPACK will be used on such machines is in applications with many tasks involving matrices small enough to be stored on a single processor. The conventional subroutines can be used on the individual processors with no modification. We call such applications “embarrassingly parallel” to emphasize the fact that, while there is a high degree of parallelism and it is possible to make efficient use of many processors, the granularity is large enough that no cooperation between the processors is required within the matrix computations.

“Weak Speedup”

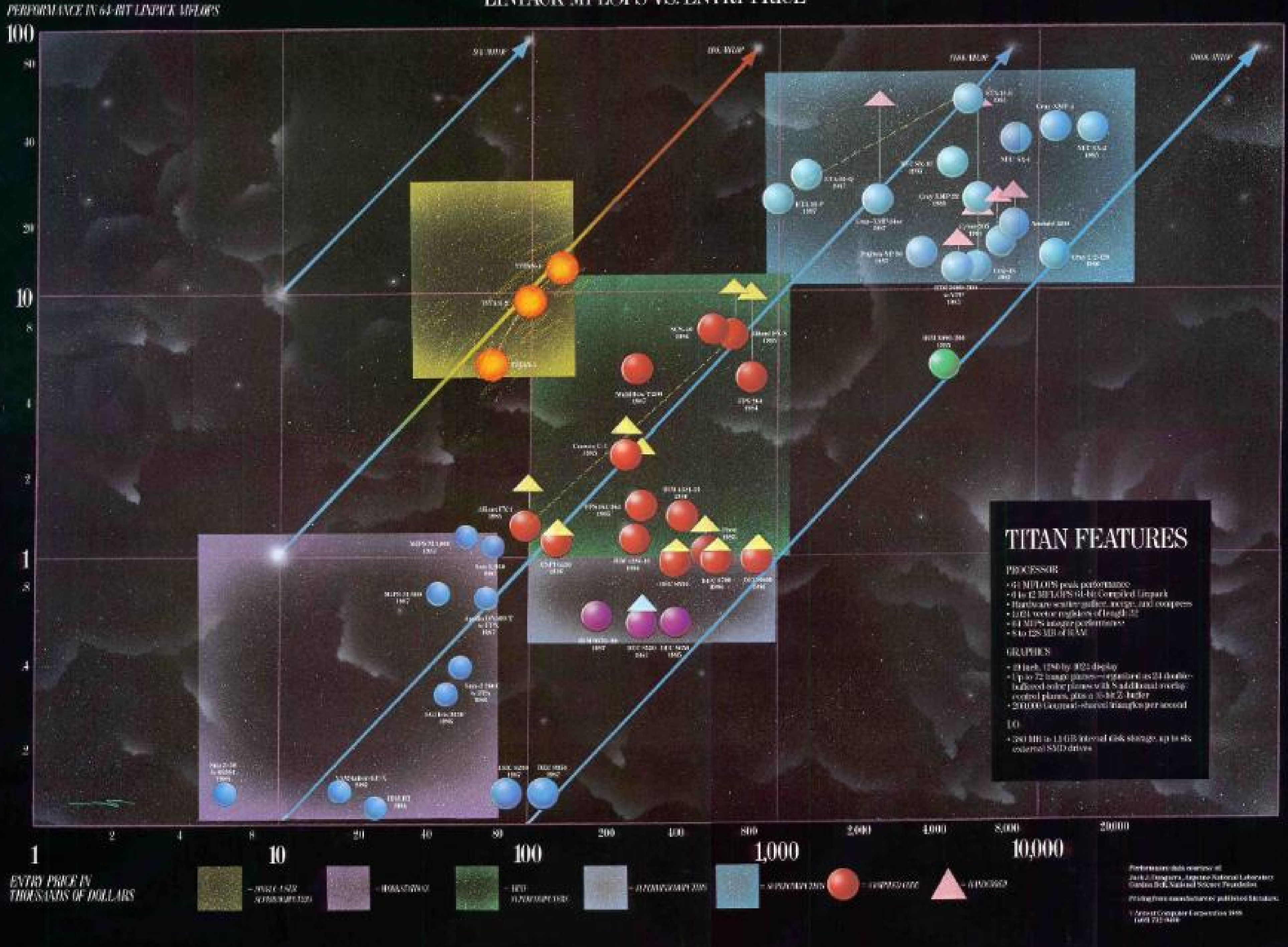
To fully utilize the system, we must consider problems involving many matrices …, or matrices of larger order. … The performance is strongly dependent on the two parameters, n and p. For a given p, there is a maximum value of n determined by the amount of memory available. …

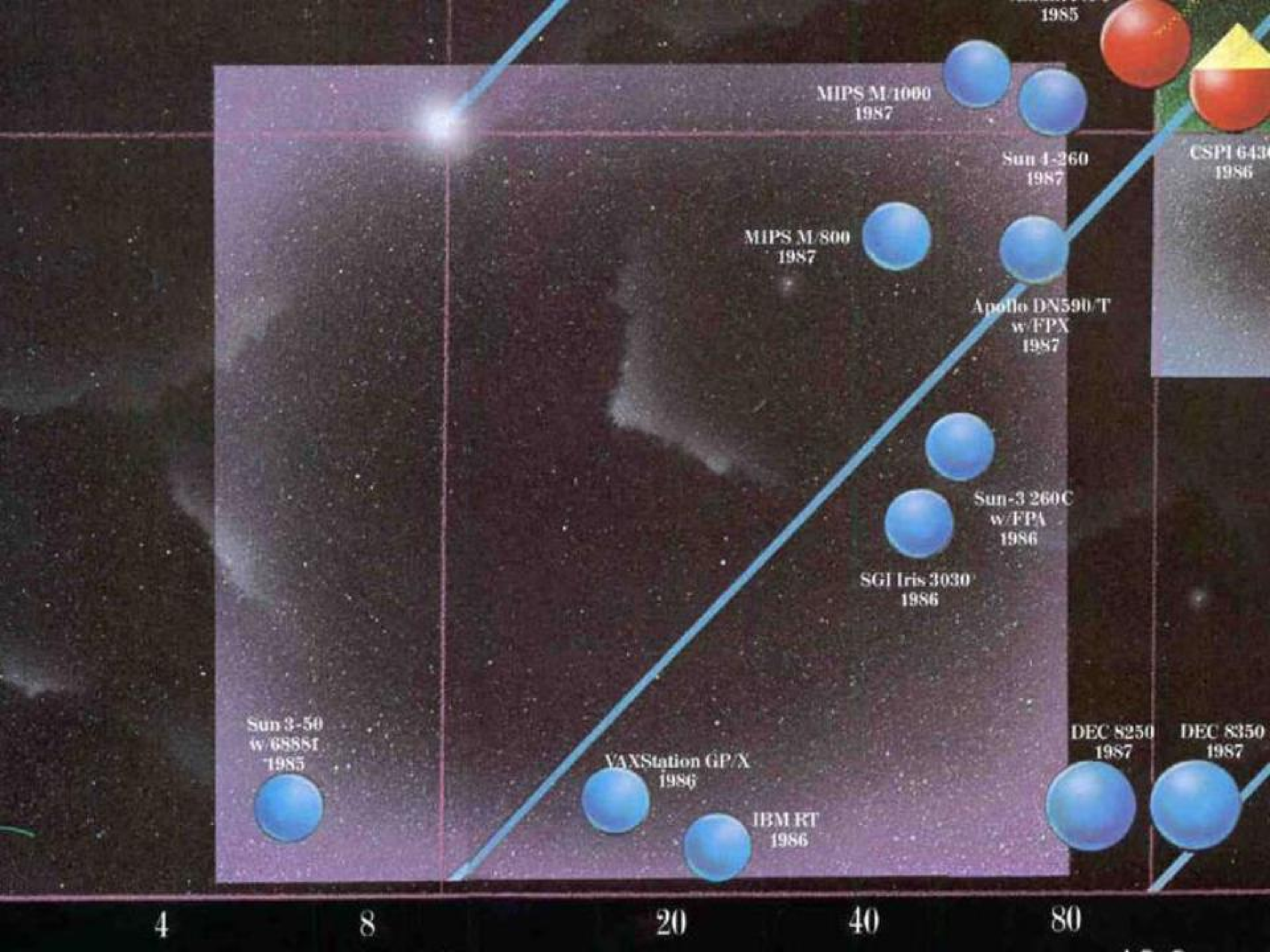
$$\overline{n}_{\text{max}} \approx \sqrt{pM}$$

“Megaflops per Gallon”

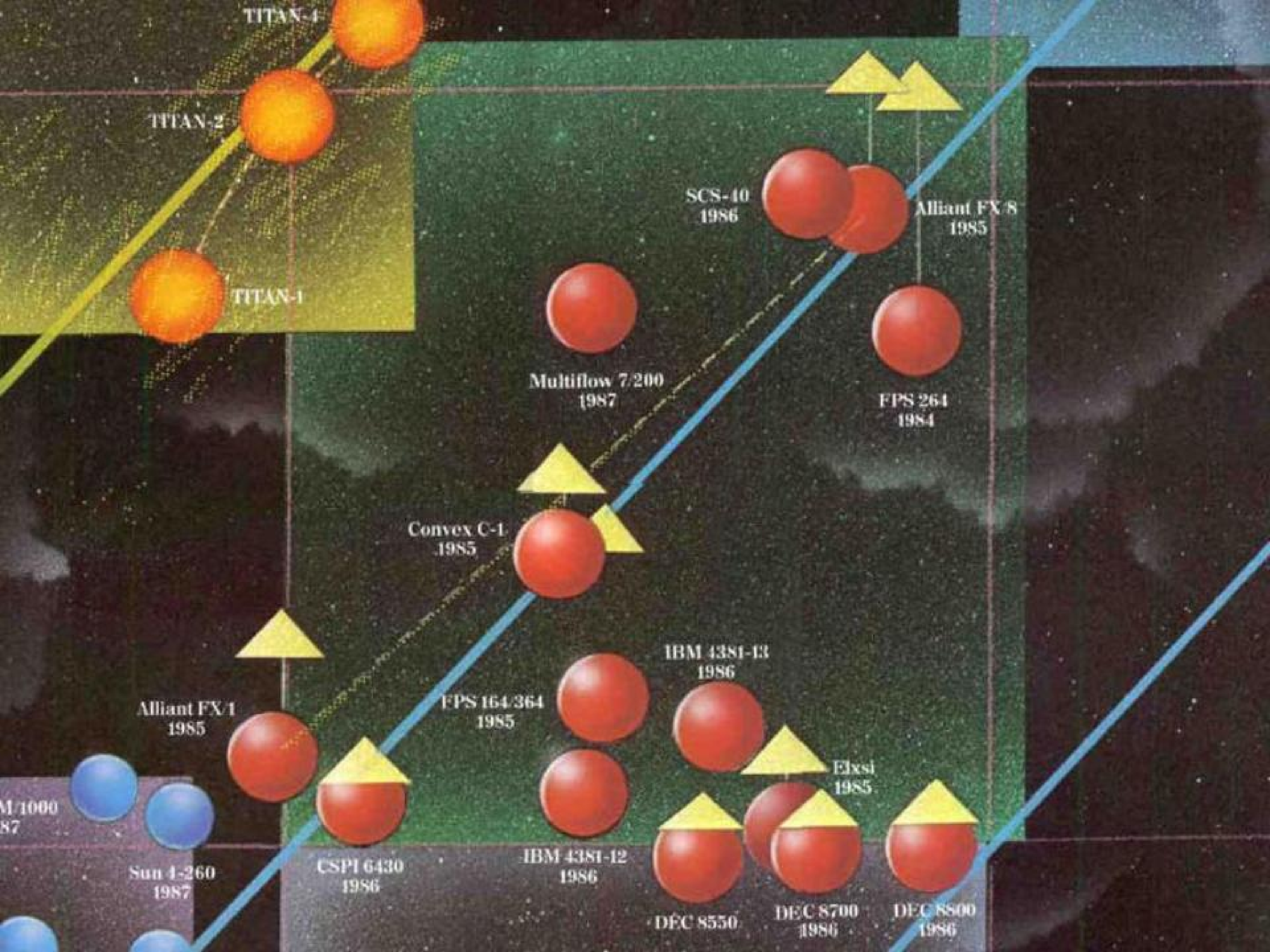
THE UNIVERSE OF SUPERCOMPUTING

LINPACK MFLOPS VS. ENTRY PRICE



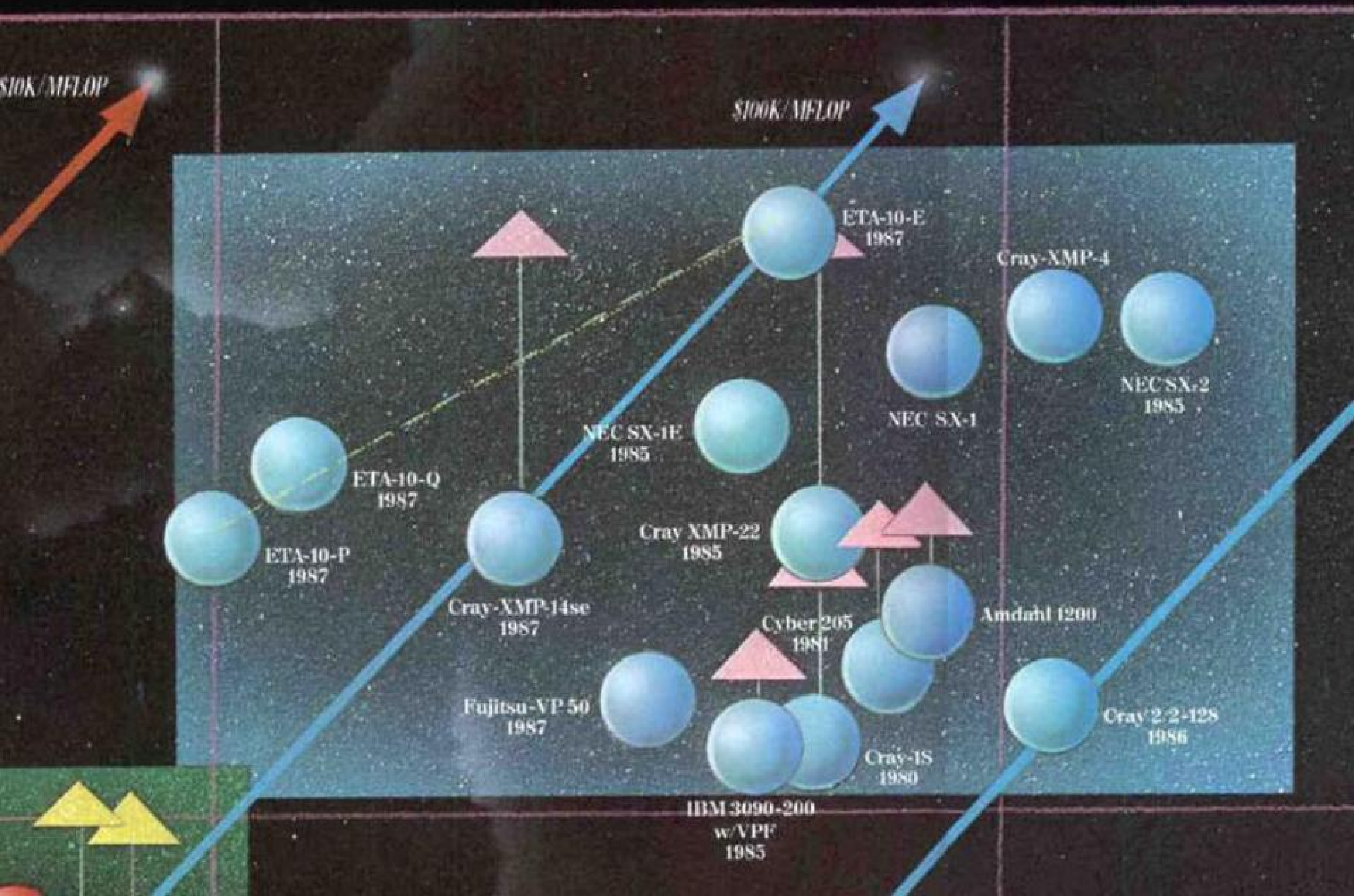


1985
MIPS M/1000
1987
CSPI 643
1986
Sun 4-260
1987
MIPS M/800
1987
Apollo DN590/T
w/FPX
1987
Sun-3 260C
w/FPA
1986
SGI Iris 3030
1986
Sun 3-50
w/68881
1985
VAXStation GP/X
1986
IBM RT
1986
DEC 8250
1987
DEC 8350
1987
4
8
20
40
80



TITAN-4
TITAN-2
TITAN-1
SCS-40
1986
Alliant FX/8
1985
Multiflow 7/200
1987
FPS 264
1984
Convex C-1
1985
Alliant FX/1
1985
FPS 164/364
1985
IBM 4381-13
1986
Elxsi
1985
M/1000
87
Sun 4-260
1987
CSPI 6430
1986
IBM 4381-12
1986
DEC 8550
DEC 8700
1986
DEC 8800
1986

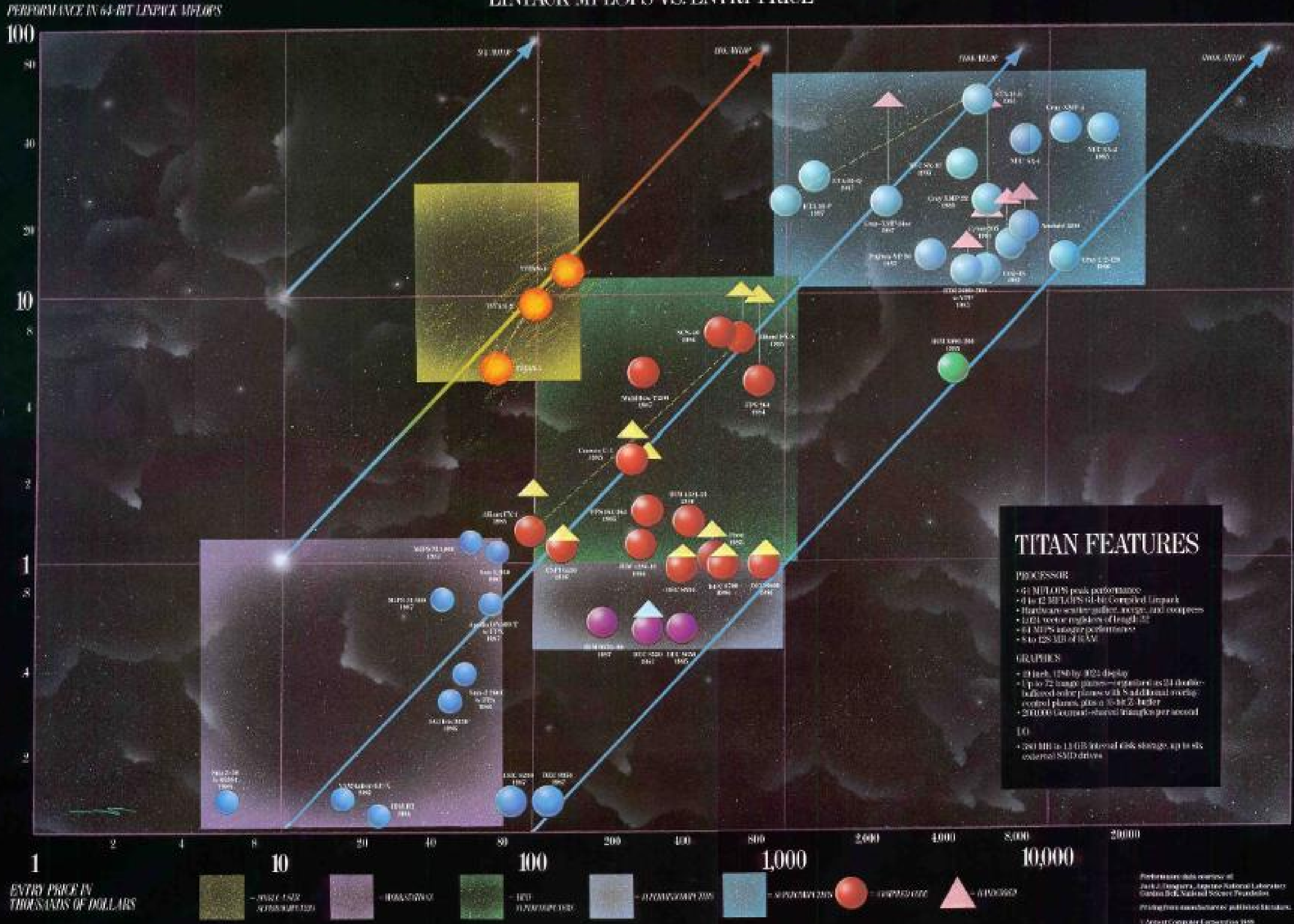
TRY PRICE



$10K/MFLOP
$100K/MFLOP
ETA-10-E
1987
Cray-XMP-4
NEC SX-2
1985
NEC SX-1
NEC SX-1E
1985
ETA-10-Q
1987
ETA-10-P
1987
Cray XMP-22
1985
Cray-XMP-14se
1987
Cyber 205
1981
Amdahl 1200
Fujitsu-VP 50
1987
Cray 2/2-128
1986
Cray-1S
1980
IBM 3090-200
w/VPF
1985

THE UNIVERSE OF SUPERCOMPUTING

LINPACK MFLOPS VS. ENTRY PRICE



TITAN FEATURES

PROCESSOR

- 64 MFLOPS peak performance
- 6 to 12 MFLOPS 64-bit Compiled Linpack
- Hardware scatter-gather, merge, and compress
- 1,024 vector registers of length 32
- 64 MIPS integer performance
- 8 to 128 MB of RAM

GRAPHICS

- 19 inch, 1280 by 1024 display
- Up to 72 image planes—organized as 24 double-buffered color planes with 8 additional overlay control planes, plus a 16-bit Z-buffer
- 200,000 Gouraud-shaded triangles per second

I/O

- 380 MB to 1.3 GB internal disk storage, up to six external SMD drives

Performance data courtesy of:
Jack J. Dongarra, Argonne National Laboratory
Gordon Bell, National Science Foundation

Pricing from manufacturers' published literature.

1990 - 2005

I am hardly involved in parallel computing, except for …

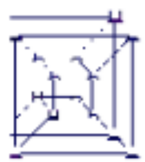
Why there isn't a parallel MATLAB

Our experience has made us skeptical

by Cleve Moler

There actually have been a few experimental versions of MATLAB for parallel computers. None of them has been effective enough to justify development beyond the experimental prototype. But we have learned enough from these experiences to make us skeptical about the viability of a fully functional MATLAB running on today's parallel machines. There are three basic difficulties:

- Memory model
- Granularity
- Business situation



A 16-node hypercube parallel computer. Each node can send messages directly to its nearest neighbors and indirectly to all other nodes.

Memory model

The most important attribute of a parallel computer is its memory model. Large-scale, massively parallel computers have potentially thousands of processors and *distributed memory*, that is, each processor has its own memory. Smaller scale machines, including some high-end workstations, have only a few processors and *shared memory*.

A good example of a distributed memory parallel computer is one of the first commercially available parallel computers, the *Intel iPSC*, where we tried to make our first parallel MATLAB almost ten years ago. It had up to 128 nodes—each a separate single board computer with an Intel microprocessor and maybe half a megabyte of memory. In principle, each node could execute a different program, but we usually ran the same program on all of them. Each node could send messages directly to its nearest neighbors and indirectly to all the other nodes. The whole machine was controlled by a front-end host, which initiated tasks, collected results, and handled all I/O.

We ran MATLAB on the host and gave names with capital letters to the functions in the parallel math library. So `INV(A)` or `FFT(X)` would start with a matrix in the host memory, split it into equally sized submatrices, send each of the submatrices to a node, invoke the parallel routine, and then collect the results back on the host. It took far longer to distribute the data than it did to do the computation. Any matrix that would fit into memory on the host was too small to make effective use of the parallel computer itself.

The situation hasn't changed very much in ten years. MATLAB is a lot bigger, and parallel computers are a lot faster. But distributed memory is still a fundamental difficulty. One of MATLAB's most attractive features is its memory model. There are no declarations or allocations—it is all handled automatically. The key question is: *Where are the matrices stored?* It is still true today that any matrix that fits into the host memory should probably stay there.

Granularity

A little over five years ago, we had a parallel MATLAB on a shared memory multiprocessor, the Ardent Titan, but we didn't tell the world about it. The most effective use of this machine, as well as today's multiprocessor workstations, is already done automatically by the operating system. MATLAB should run on only one processor, while other tasks, like the X-Windows server, use the other processors. In typical use, MATLAB spends only a small portion of its time in routines that can be parallelized, like the ones in the math library. It spends much more time in places like the parser, the interpreter, and the graphics routines, where any parallelism is difficult to find.

There are some special situations where parallel computation within MATLAB would be effective. For example, suppose I want to find what fraction of a large number of matrices have eigenvalues in the left half plane. The obvious place to parallelize this is on the outer loop. It's not necessary to use more than one processor to generate a single matrix or to compute its eigenvalues. The only place the processors would need to cooperate is in merging their final counts. However, to get MATLAB to handle this kind of parallelism would require fundamental changes to its architecture.

Business situation

It doesn't make good business sense for us to undertake fundamental changes in MATLAB's architecture. There are not enough potential customers with parallel machines. Most of the MATLAB community would rather see us devote our efforts to improving our conventional, uniprocessor software. So, we will continue to track developments in parallel computing, but we don't expect to get seriously involved again in the near future. ■

Cleve Moler is chairman and co-founder of The MathWorks. His e-mail address is moler@mathworks.com

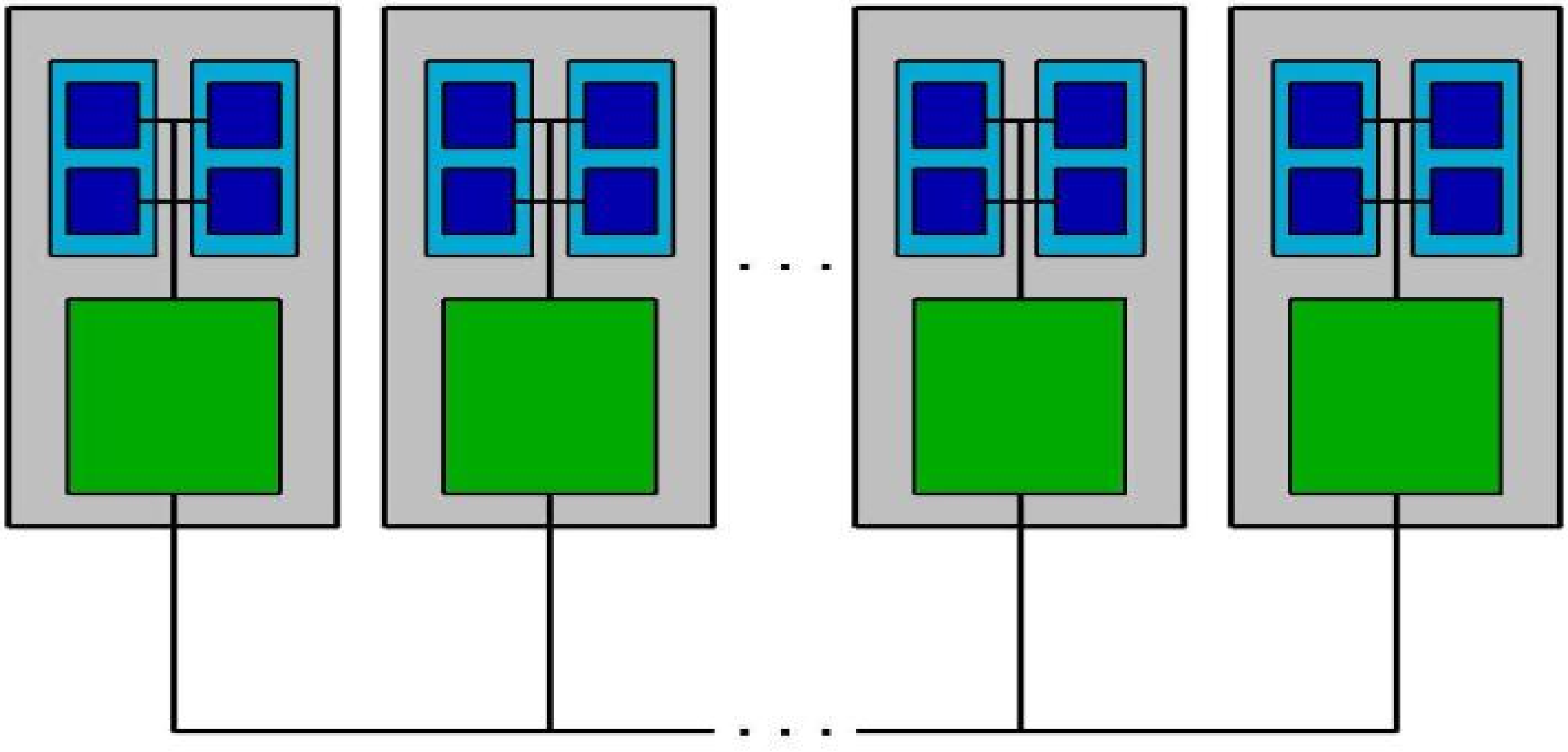
Cleve's Corner, 1995
Why there isn't a parallel MATLAB

- Memory model
- Granularity
- Business situation

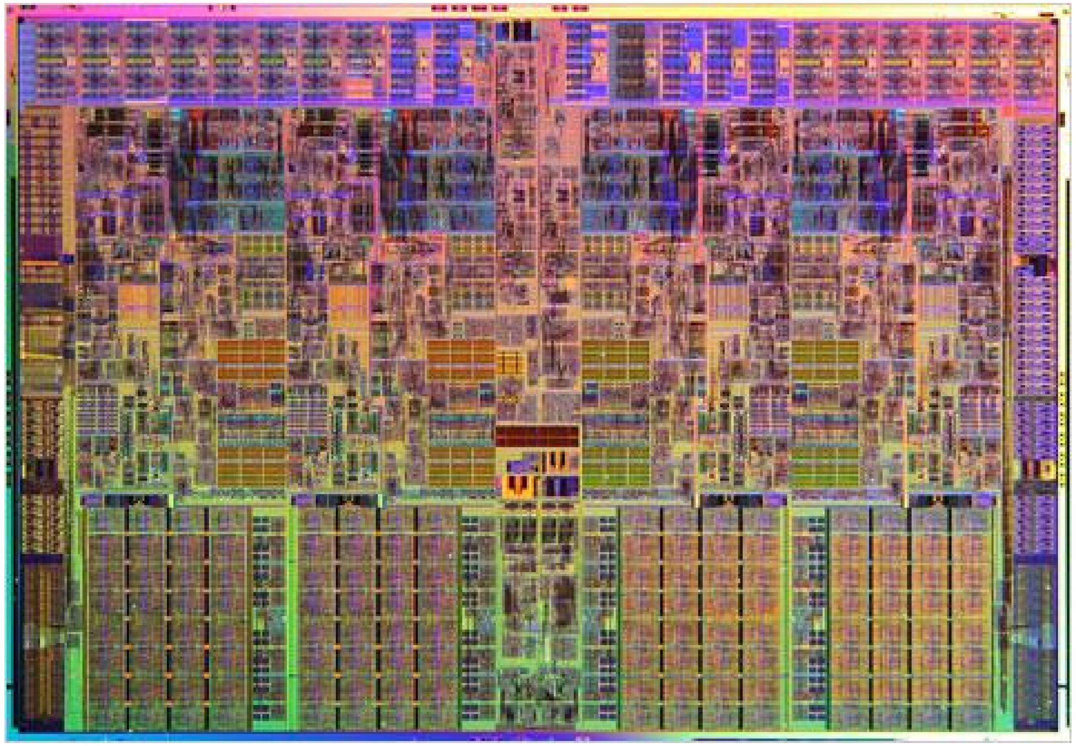
2005

Ron Choy's Web page at MIT lists
27 Parallel MATLABs.
None of them are from The MathWorks

Parallel Computing Today



Quad Core Microprocessor



Multicore Parallelism

- Fine grained
- Multithreaded
- Shared memory
- Automatic
- Dangerous
- Not scalable
- Memory bandwidth
- ISMOP (Its' a Small Matter of Programming)

ORNL Jaguar
180,828 cores

Multicomputer Parallelism

- Coarse grained
- Message passing
- Distributed memory
- Explicit

Parallel MATLAB

- Introduced in 2005
- *NOT* for top 500, but everybody else
- Now at Version 4.2
- Parallel Computing Toolbox
- Distributed Computing Server

Parallel Computing Toolbox

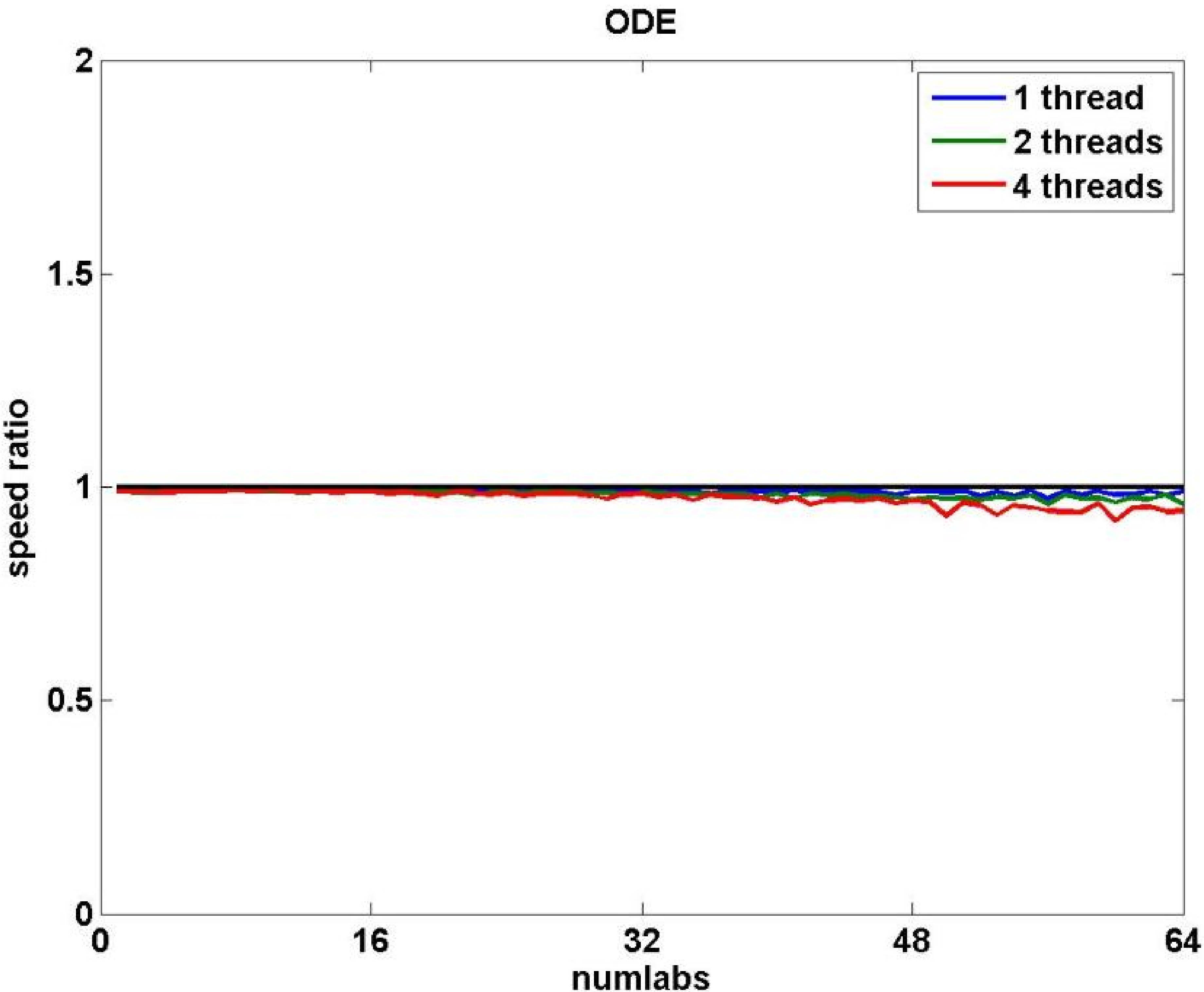
- ≤ 8 local “labs”
- **`parfor`**
- **`spmd`**
- distributed arrays

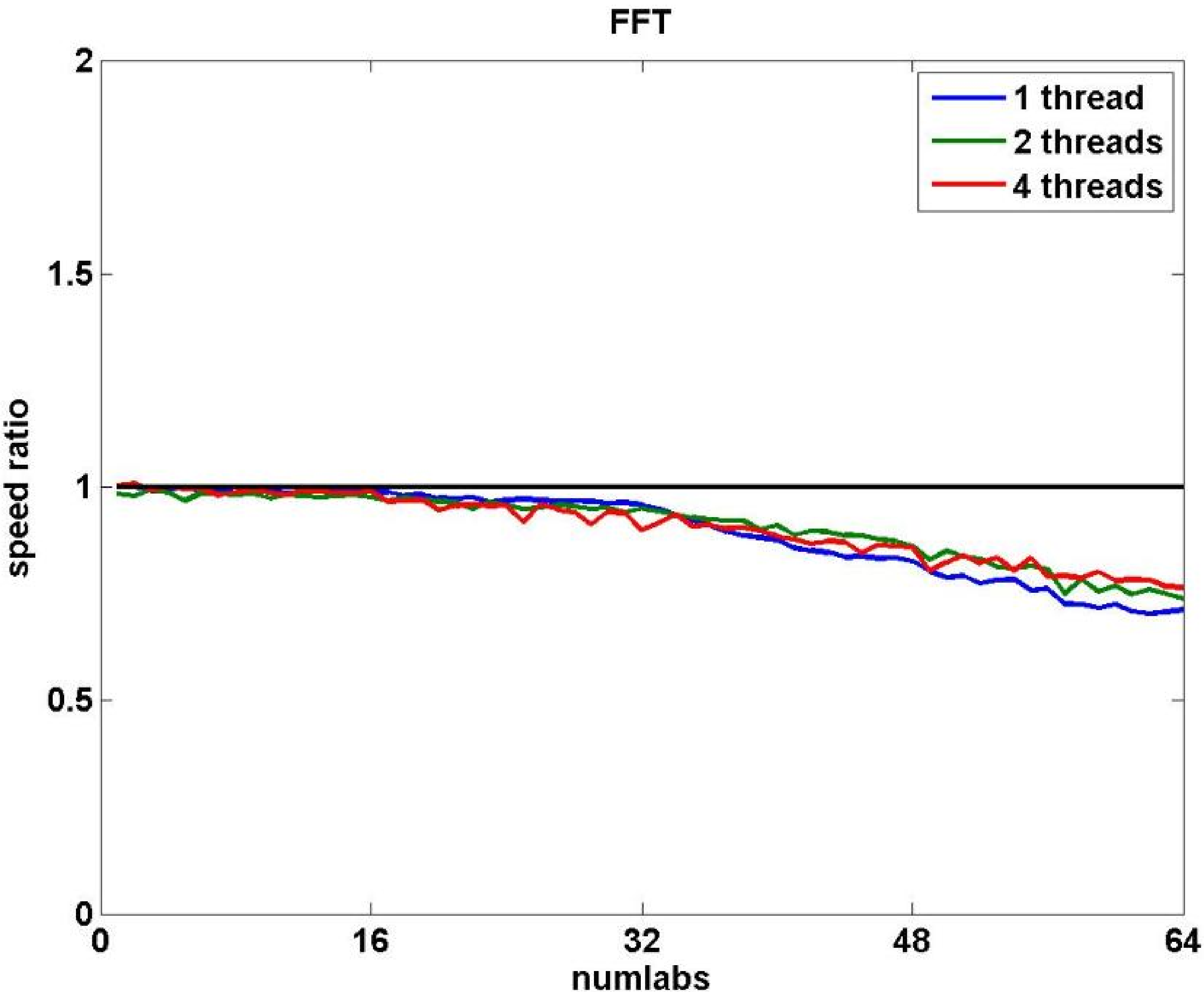
Distributed Computing Server

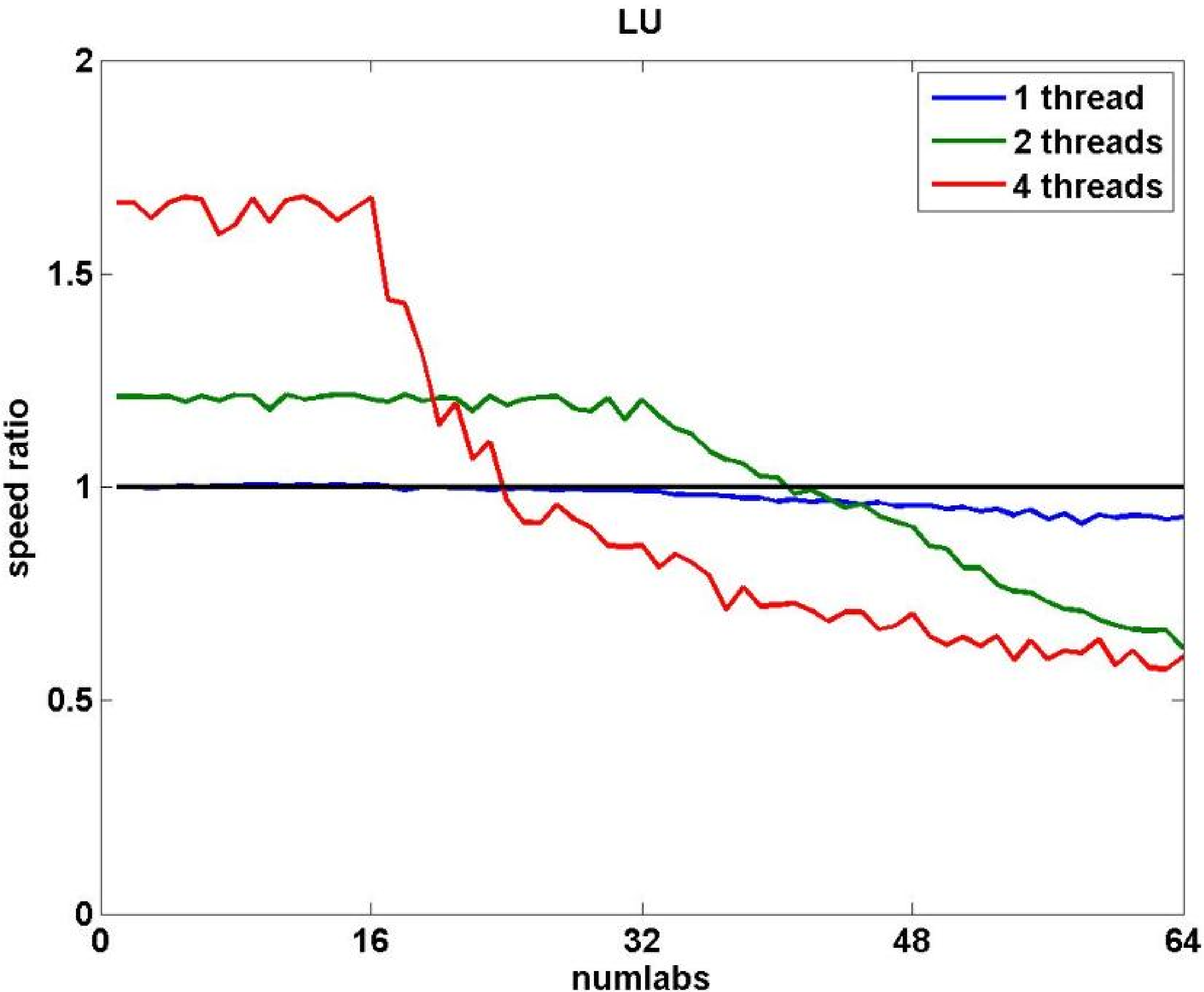
- > 8 “labs”
- Interface to job managers

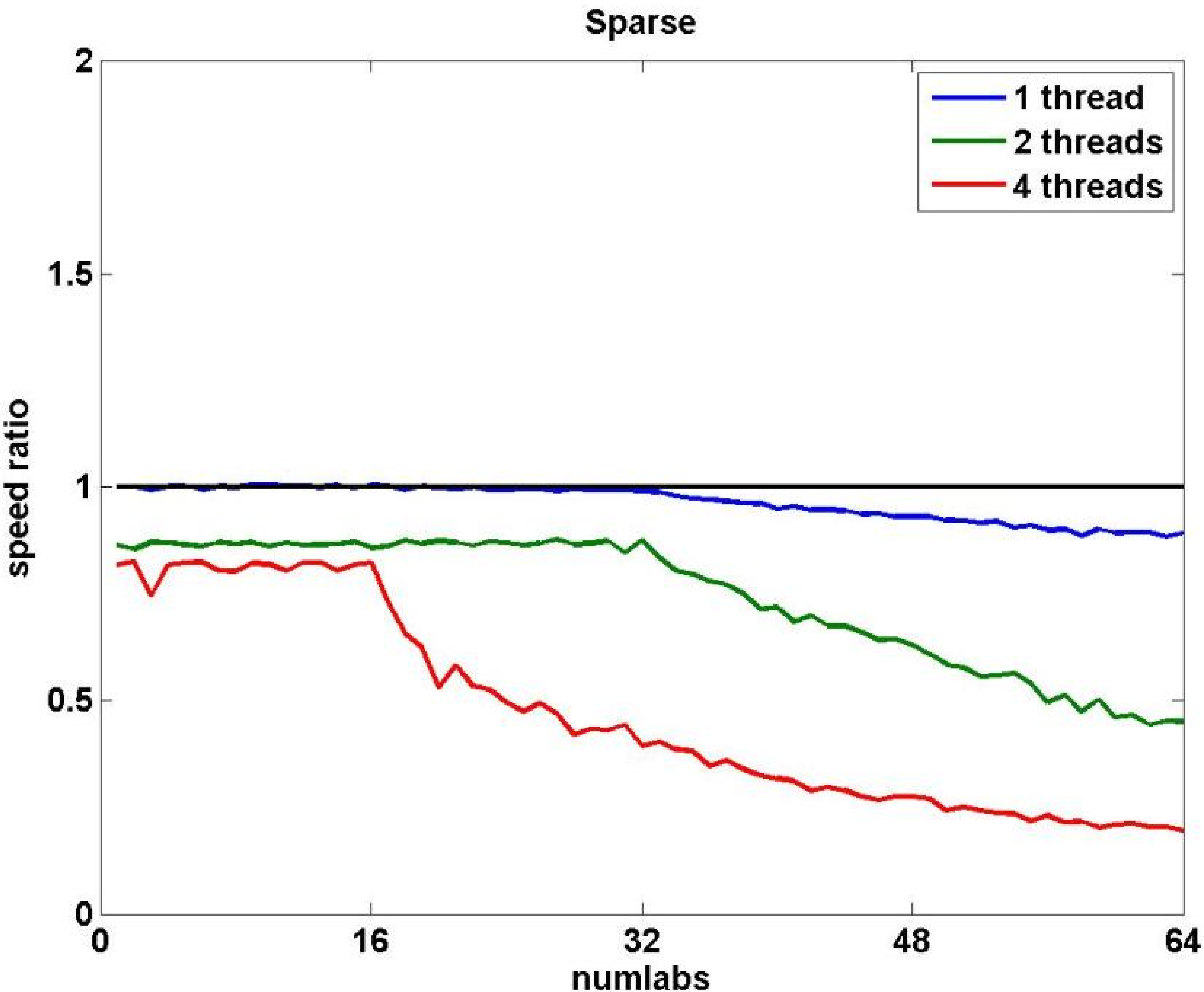
Embarrassingly Parallel Multithreaded Benchmarks

- MATLAB 7.4 (R2007a)
- 16 dual-processor, dual-core Opterons
- $1 \leq$ labs ≤ 64
- $1 \leq$ threads ≤ 4
- **`ode`**
- **`fft`**
- **`LU`**
- **`sparse`**


ODE
1 thread
2 threads
4 threads
speed ratio
numlabs
0
0.5
1
1.5
2
0
16
32
48
64


FFT
1 thread
2 threads
4 threads
speed ratio
numlabs
0
0.5
1
1.5
2
0
16
32
48
64

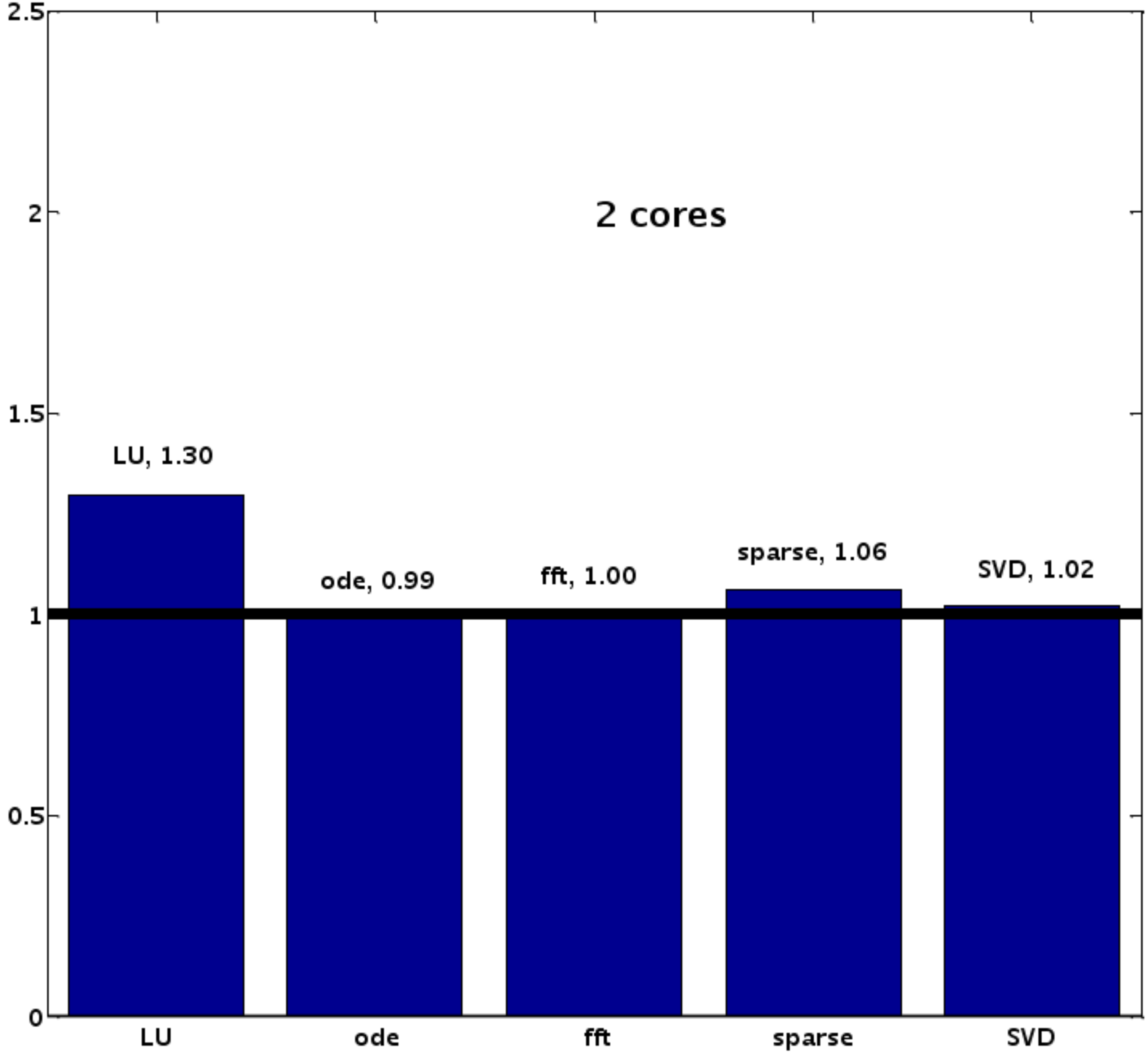

LU
1 thread
2 threads
4 threads
speed ratio
numlabs
0
0.5
1
1.5
2
0
16
32
48
64

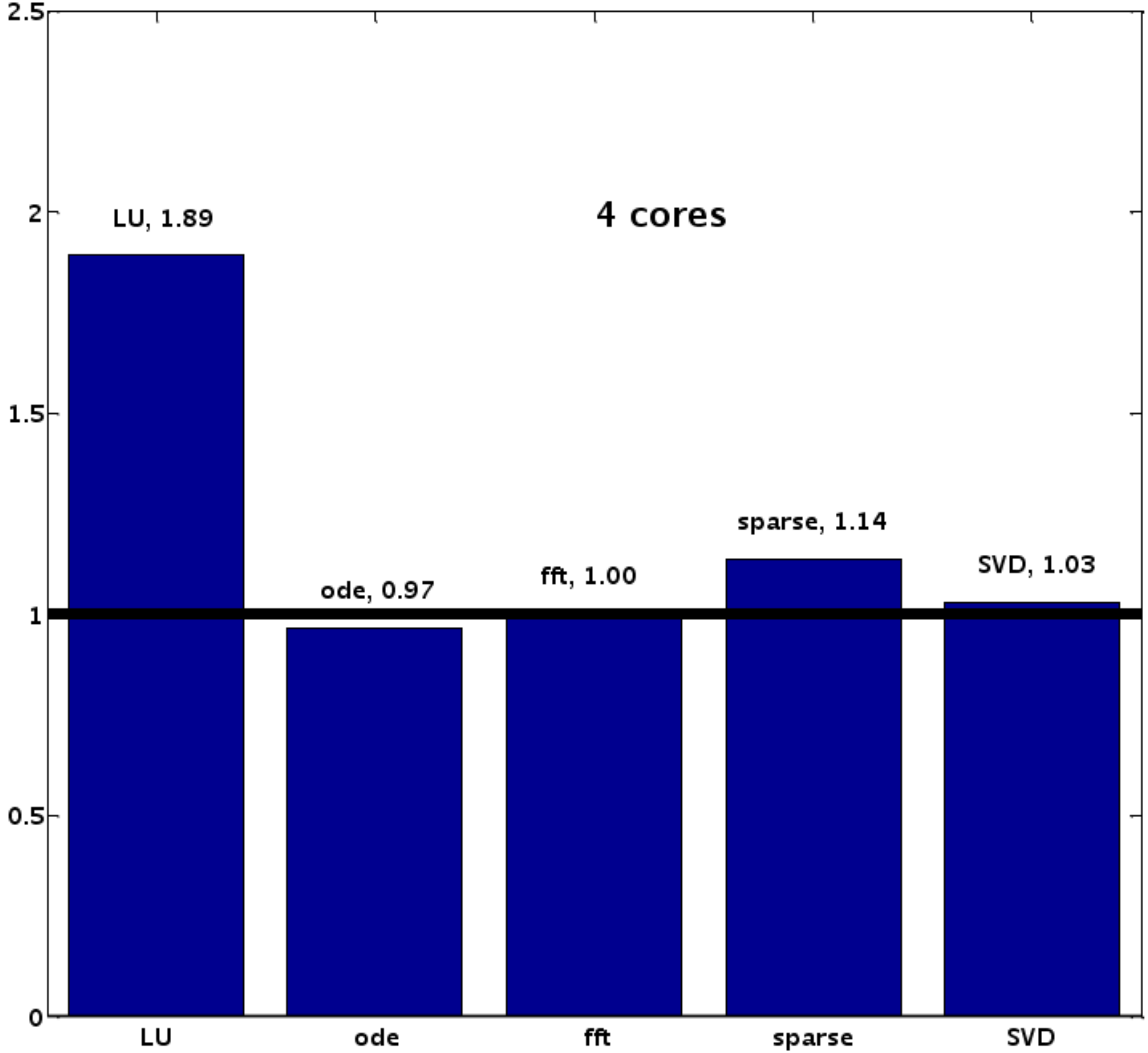


Sparse
1 thread
2 threads
4 threads
speed ratio
numlabs
0
0.5
1
1.5
2
0
16
32
48
64

Multithreaded benchmarks

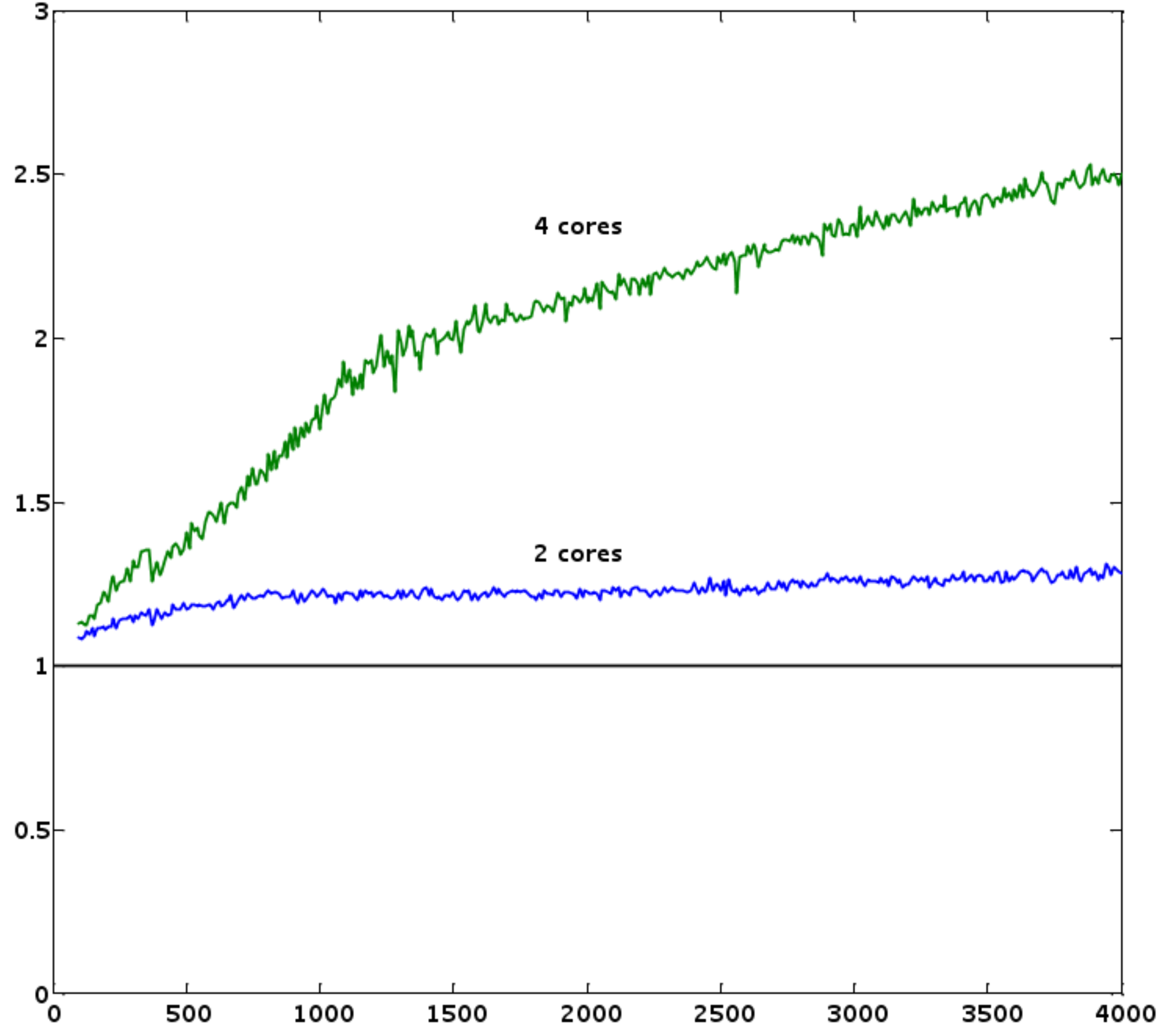
- MATLAB 7.9 (R2009b)
- HP D5100 home computer
- Intel Core2 QUAD CPU, 2.83 GHz
- threads = [1, 2, 4]
- **`LU(1000)`**
- **`fft(2^20)`**
- **`ode,`** van der Pol, $0 \le t \le 400$
- **`sparse \, delsq(numgrid('L',300))`**
- **`SVD(1000)`**


2 cores
LU, 1.30
ode, 0.99
fft, 1.00
sparse, 1.06
SVD, 1.02
2.5
2
1.5
1
0.5
0
LU
ode
fft
sparse
SVD

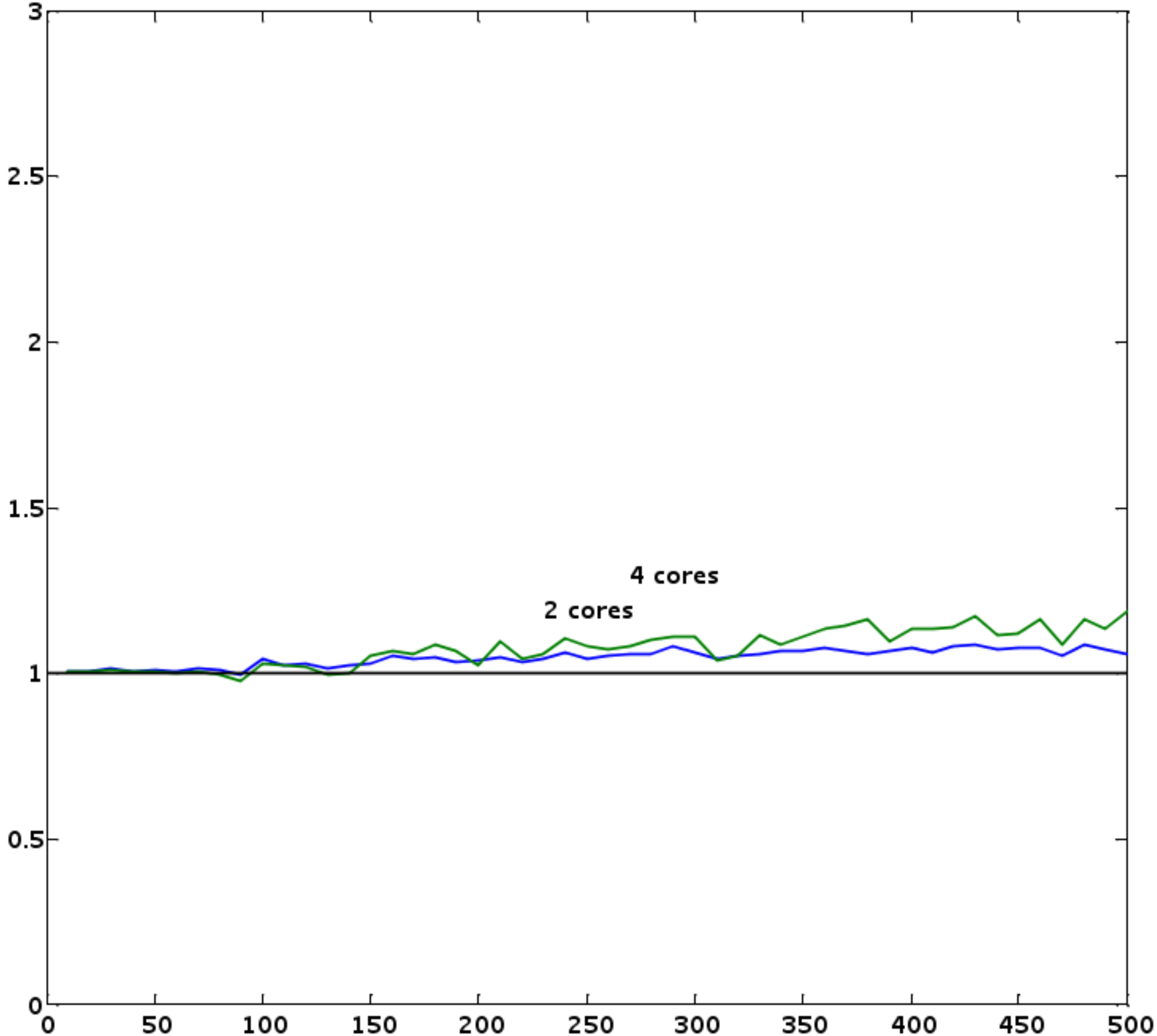

4 cores
LU, 1.89
ode, 0.97
fft, 1.00
sparse, 1.14
SVD, 1.03
2.5
2
1.5
1
0.5
0
LU
ode
fft
sparse
SVD

Matrix benchmarks, vary size

- **`LU(n)`**
- **`sparse \, delsq(numgrid('L',g))`**



4 cores
2 cores
3
2.5
2
1.5
1
0.5
0
0
500
1000
1500
2000
2500
3000
3500
4000



3
2.5
2
1.5
1
0.5
0
0
50
100
150
200
250
300
350
400
450
500
4 cores
2 cores

FLOPs Don’t Count Anymore

- Memory Touches
- Power Consumption
- Parallelism

What Can Be Parallelized?

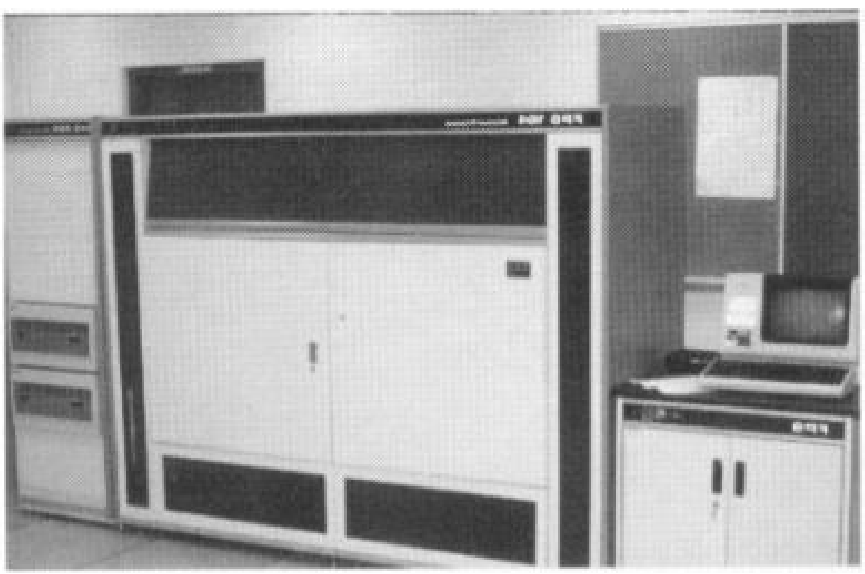
- Programming is the easy part
- Discovering parallelism is hard
- No algorithmic theory

Embarrassingly Parallel

- “Fully Parallel”
- Monte Carlo
- Parameter sweeps
- Most prevalent

GPUs and FPGAs

- Today's FPS 164



→

Effective Parallelism

- Twice as much output
- Two sets of parameters
- *NOT* twice as fast
- Multithreading is a bad idea

Conclusion

More Powerful Computers

=>

Coarser Granularity

=>

Embarrassingly Parallel

Part II

Experiments with MATLAB

http://www.mathworks.com/moler/exm

Experiments with MATLAB

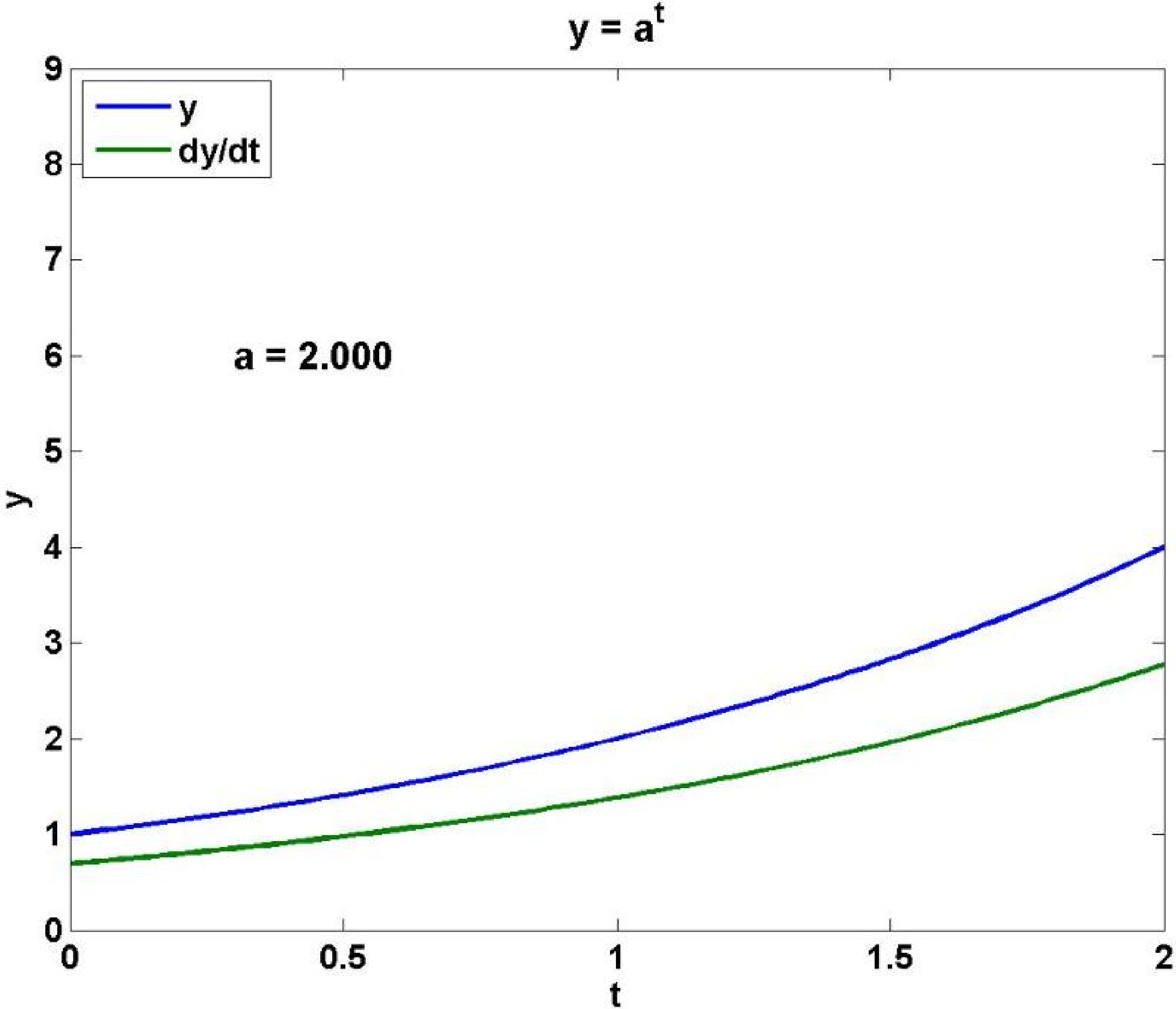
Preface
Iteration
Fibonacci Numbers
Calendars and Clocks
T Puzzle
Matrices
Fractal Fern
Magic Squares
TicTacToe Magic
Game of Life
Mandelbrot Set
Linear Equations
Google PageRank
Ordinary Differential Equations
Exponential Function
Predators and Prey
Shallow Water Equations
Orbits
Sudoku

Homework:

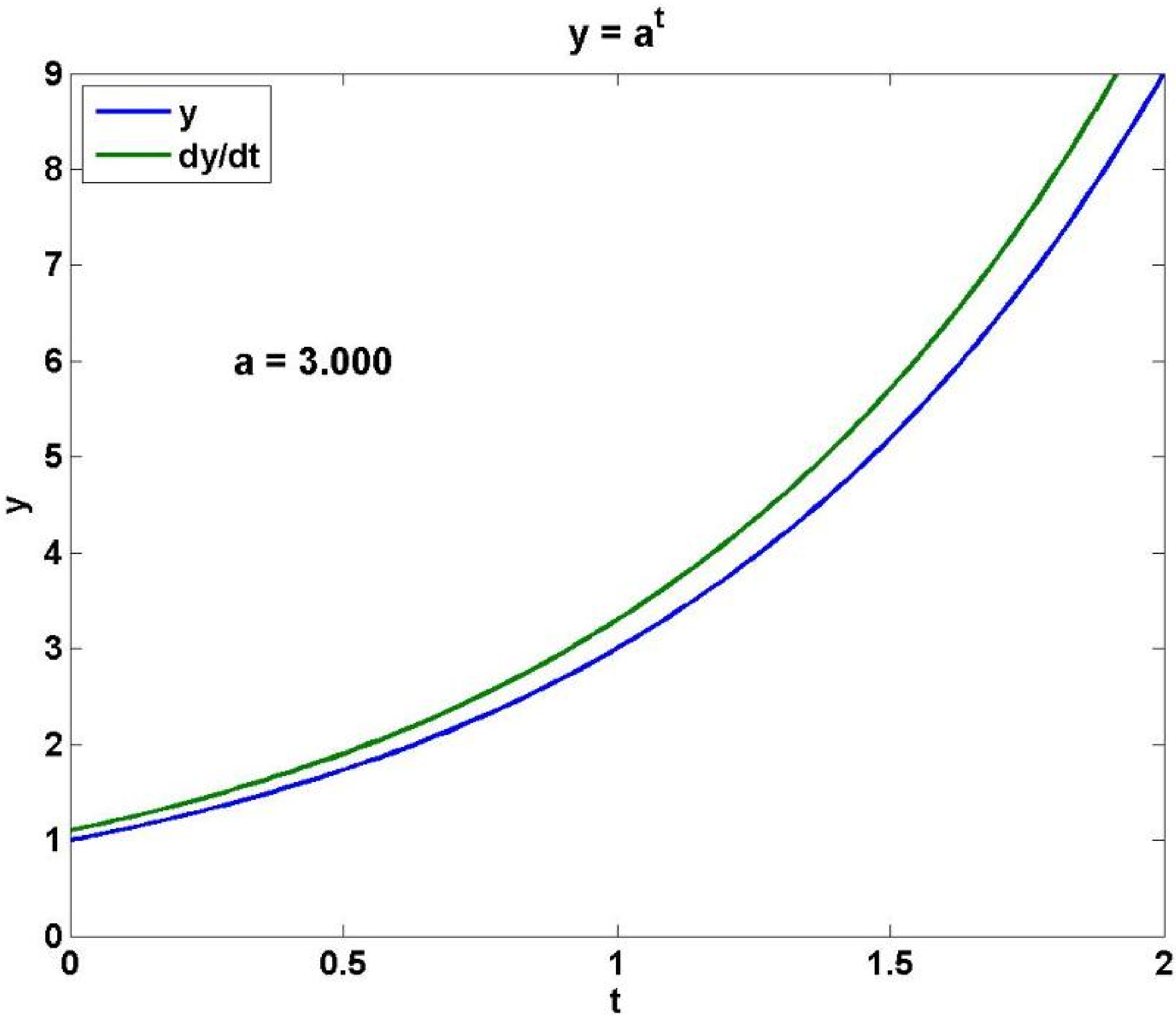
Friday the 13th is unlucky, but is it unlikely?

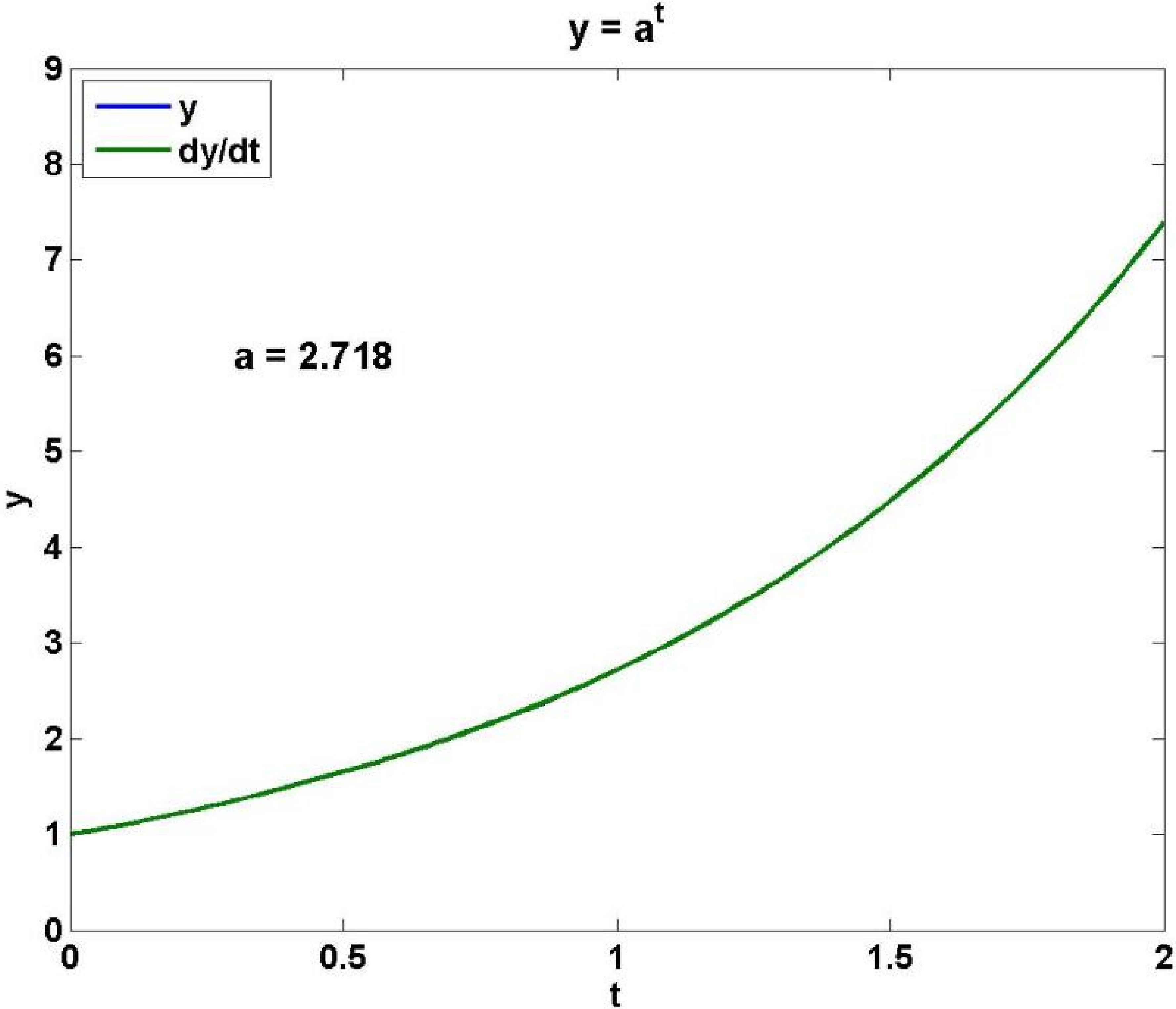
What is the probability that
the 13th of any month is on a Friday?

See *Experiments with MATLAB/Calendars.*



y = a^t
y
dy/dt
a = 2.000
9
8
7
6
5
4
3
2
1
0
y
0
0.5
1
1.5
2
t

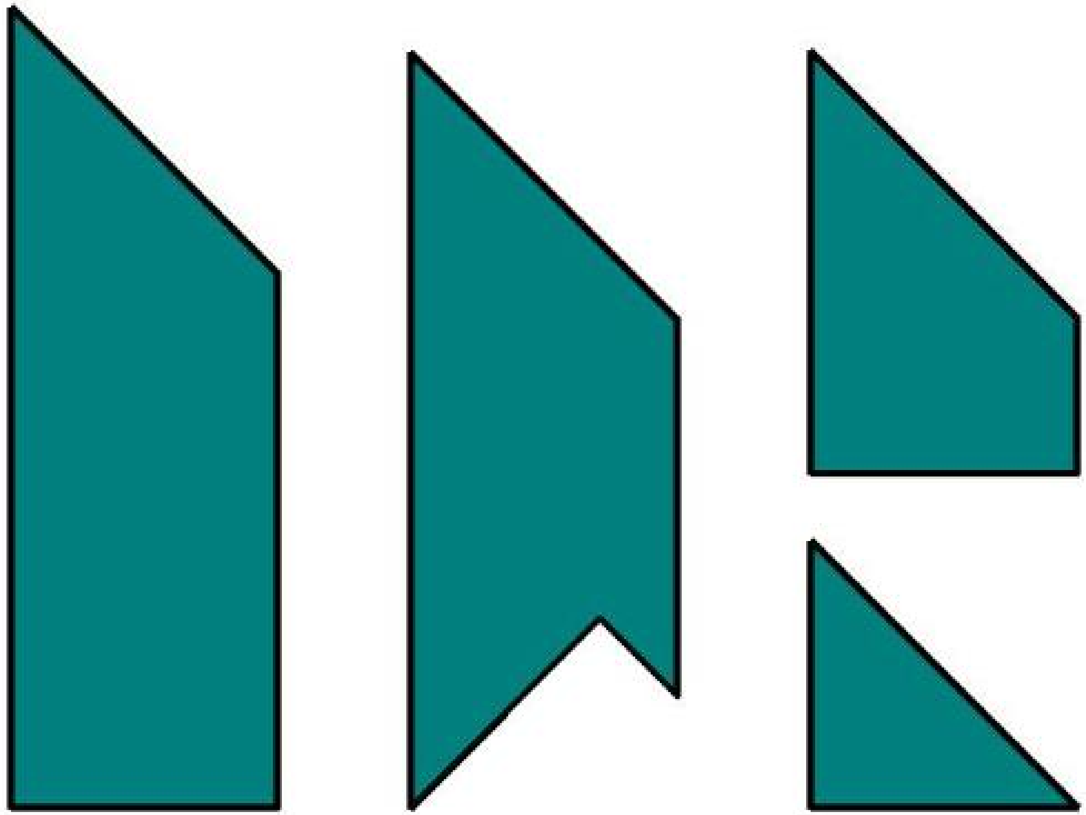

y = a^t
y
dy/dt
a = 3.000
9
8
7
6
5
4
3
2
1
0
0
0.5
1
1.5
2
t
y

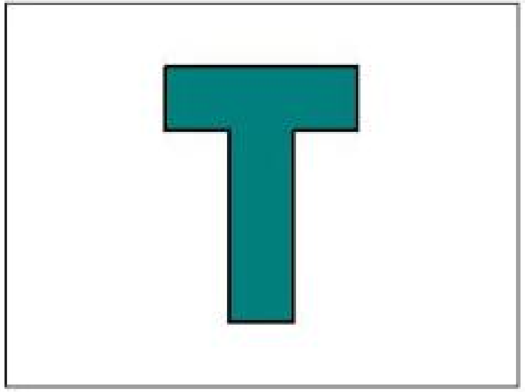


y = a^t
y
dy/dt
a = 2.718
9
8
7
6
5
4
3
2
1
0
0
0.5
1
1.5
2
t
y

Approximate Derivative

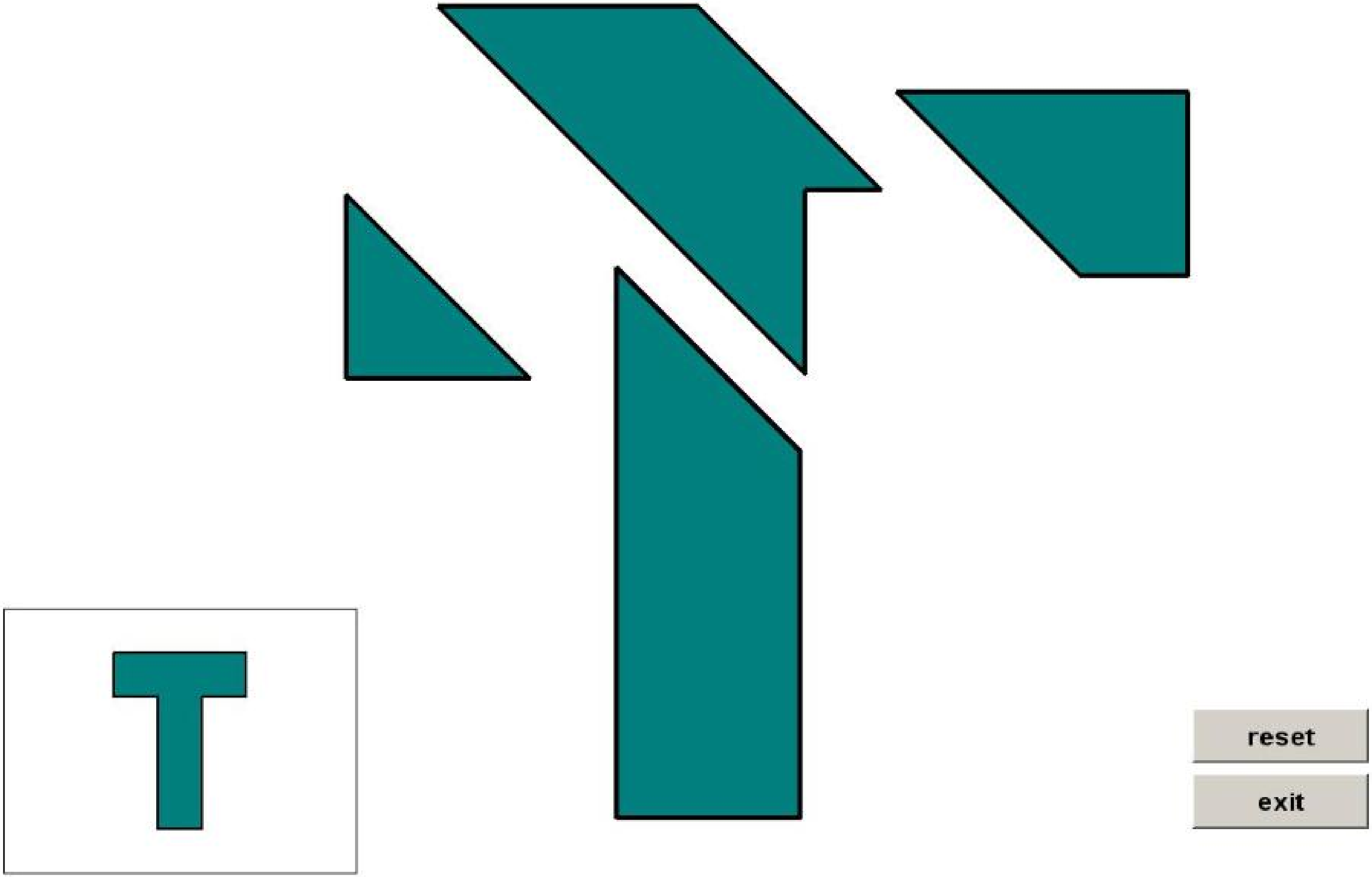
```
yp = (a.^(t+h) - a.^t)/h
```

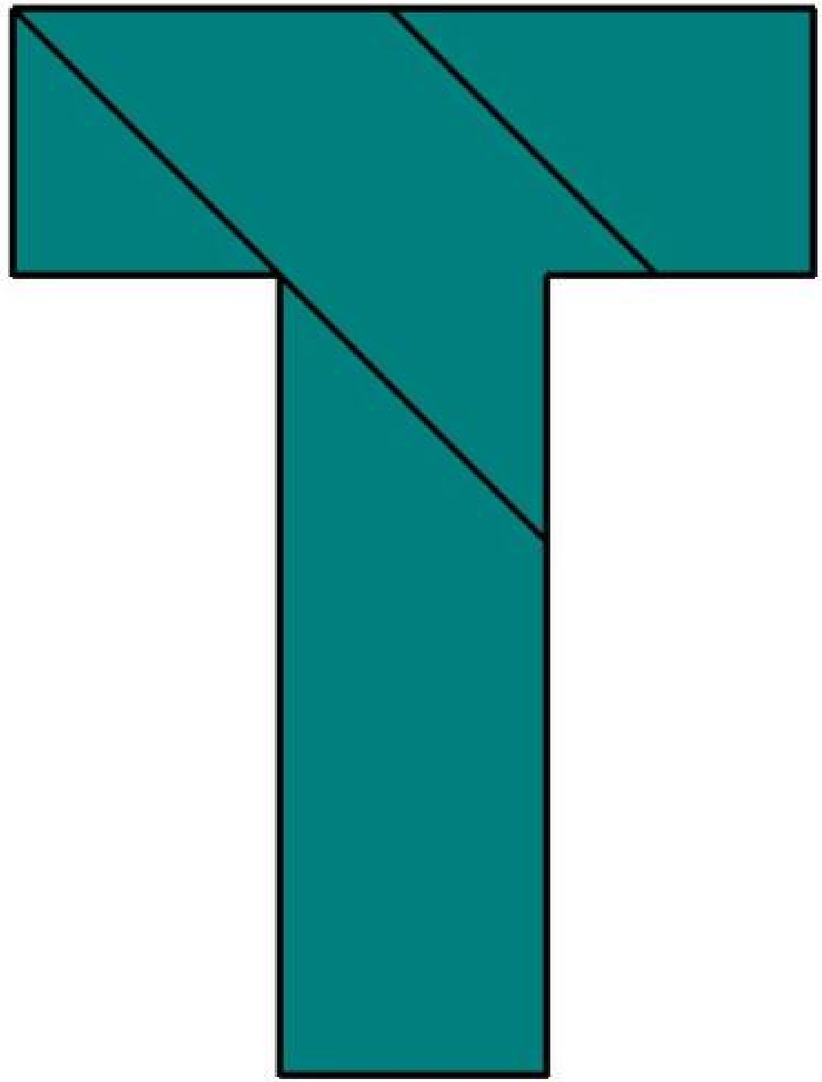


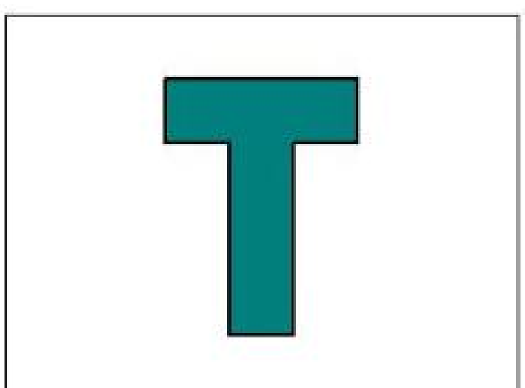


reset

exit


reset
exit





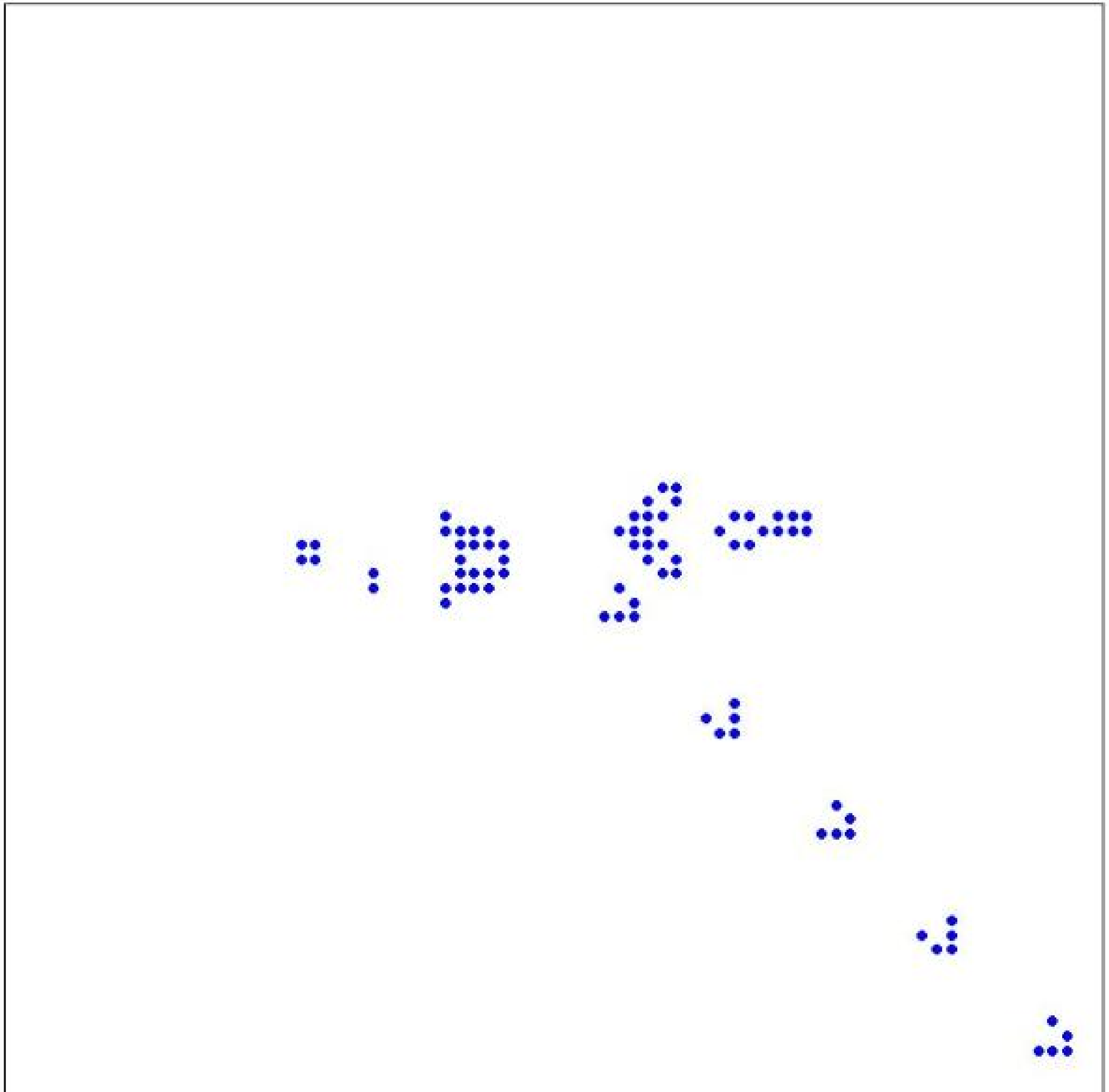
reset

exit

Rotation

```
z = exp(i*theta)*(z - mu) + mu
```

Gosper glider gun

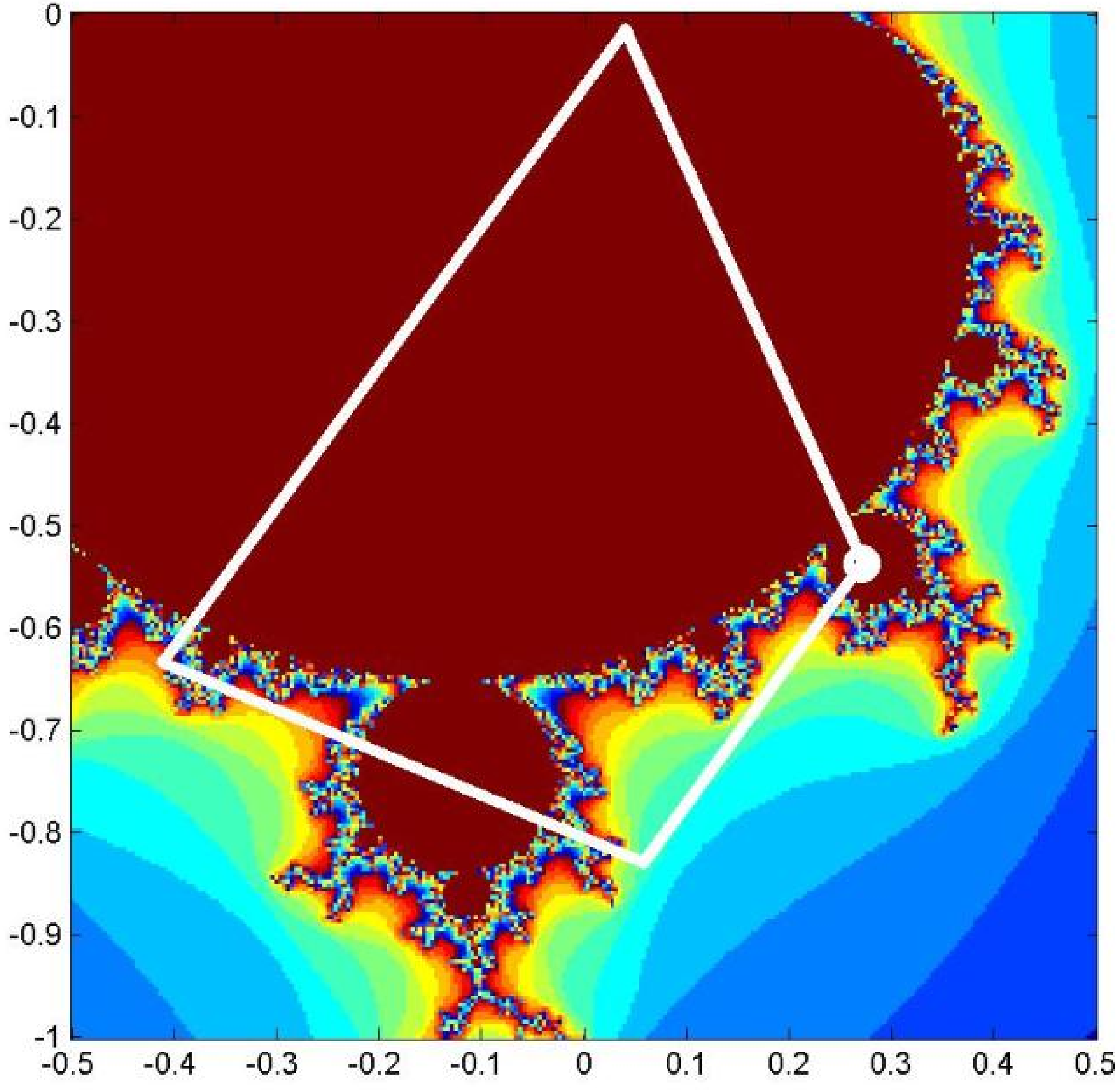


t=143, pop= 80

step | slow | fast | redo | next | random | quit

Life

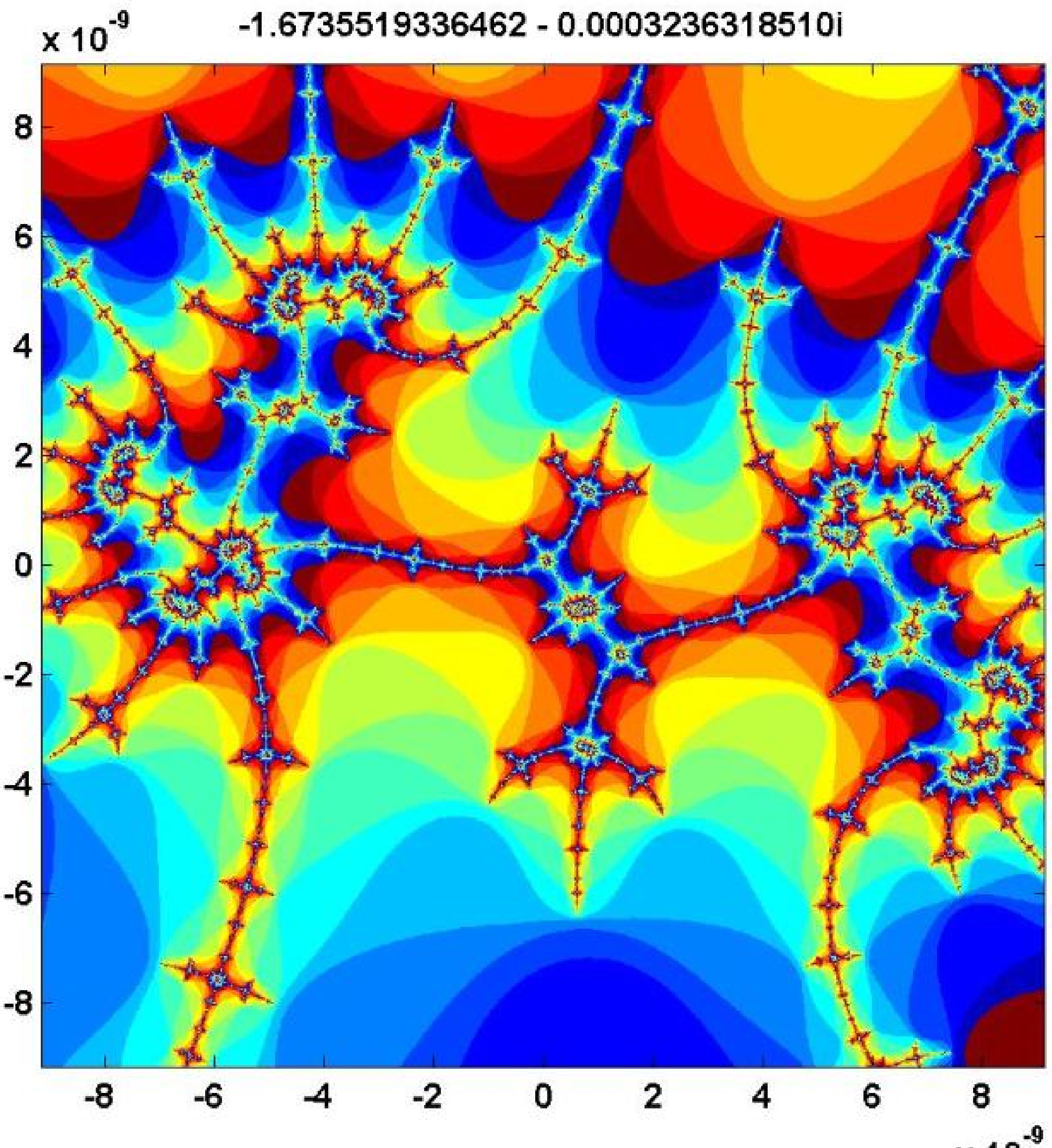
```
Y = X(:,p) + X(:,q) + X(p,:) + X(q,:) + ...
X(p,p) + X(q,q) + X(p,q) + X(q,p);
X = (X & (Y == 2)) | (Y == 3);
```

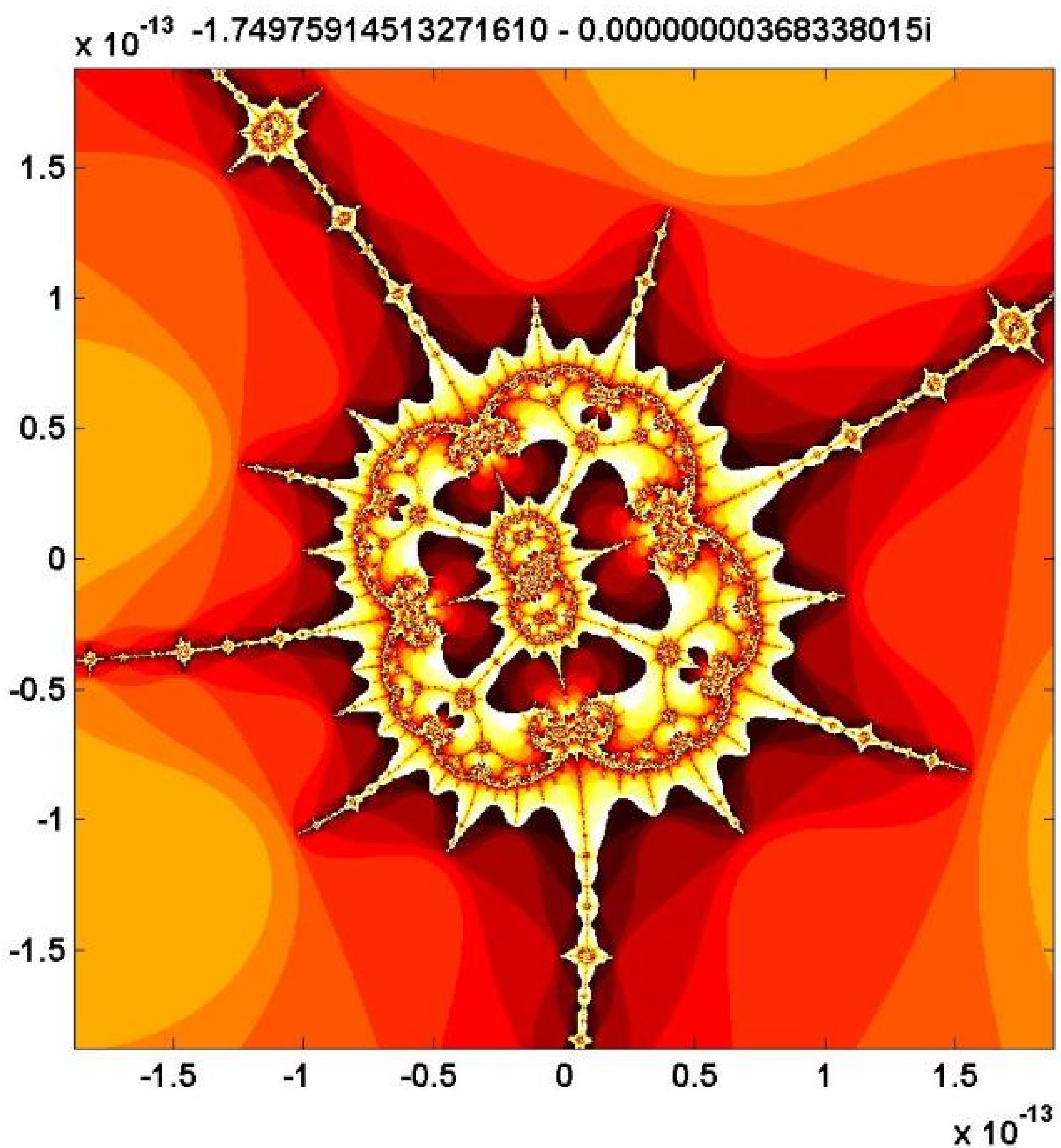


0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-0.5 -0.4 -0.3 -0.2 -0.1 0 0.1 0.2 0.3 0.4 0.5

Mandelbrot

```
z = z.*z + z0;
kz(abs(z) < 2) = d;
```


-1.6735519336462 - 0.0003236318510i
x 10^-9
8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8
x 10^-9



x 10^-13
-1.7497591451327161 0 - 0.00000000368338015i
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
x 10^-13