PETSc and Economics: Leveraging the "Portable Extensible Toolkit for Scientific Computation"

Jeremy Bejarano

University of Chicago
jbejarano@uchicago.edu

July 2013

Theme of the Week

We need more computational power!

The technologies available in high performace computing (HPC) remain largely untapped in the field of economics. HPC will allow us to solve bigger, more realistic models, and will help move the profession into the 21st century.

Points of Talk

- High Performance Computing
- What is PETSc?
- PETSc: At a Glance
- Uses in Economics

High Performance Computing

Recap

Why HPC?

- Computationally Intense Jobs (Long Run Times)
- Memory Contraints (Not feasible on single machine)
- I/O Constaints

Recap

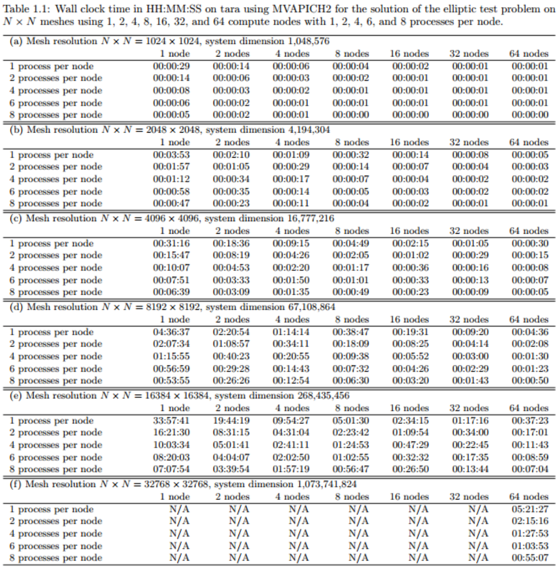


Table 1.1: Wall clock time in HH:MM:SS on tara using MVAPICH2 for the solution of the elliptic test problem on $N \times N$ meshes using 1, 2, 4, 8, 16, 32, and 64 compute nodes with 1, 2, 4, 6, and 8 processes per node.

(a) Mesh resolution $N \times N = 1024 \times 1024$, system dimension 1,048,576

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:00:29	00:00:14	00:00:06	00:00:04	00:00:02	00:00:01	00:00:01
2 processes per node	00:00:14	00:00:06	00:00:03	00:00:02	00:00:01	00:00:01	00:00:01
4 processes per node	00:00:08	00:00:03	00:00:02	00:00:01	00:00:01	00:00:01	00:00:01
6 processes per node	00:00:06	00:00:02	00:00:01	00:00:01	00:00:01	00:00:01	00:00:01
8 processes per node	00:00:05	00:00:02	00:00:01	00:00:00	00:00:00	00:00:00	00:00:00

(b) Mesh resolution $N \times N = 2048 \times 2048$, system dimension 4,194,304

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:03:53	00:02:10	00:01:09	00:00:32	00:00:14	00:00:08	00:00:05
2 processes per node	00:01:57	00:01:05	00:00:29	00:00:14	00:00:07	00:00:04	00:00:03
4 processes per node	00:01:12	00:00:34	00:00:17	00:00:07	00:00:04	00:00:02	00:00:02
6 processes per node	00:00:58	00:00:35	00:00:14	00:00:05	00:00:03	00:00:02	00:00:02
8 processes per node	00:00:47	00:00:23	00:00:11	00:00:04	00:00:02	00:00:01	00:00:01

(c) Mesh resolution $N \times N = 4096 \times 4096$, system dimension 16,777,216

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	00:31:16	00:18:36	00:09:15	00:04:49	00:02:15	00:01:05	00:00:30
2 processes per node	00:15:47	00:08:19	00:04:26	00:02:05	00:01:02	00:00:29	00:00:15
4 processes per node	00:10:07	00:04:53	00:02:20	00:01:17	00:00:36	00:00:16	00:00:08
6 processes per node	00:07:51	00:03:33	00:01:50	00:01:01	00:00:33	00:00:13	00:00:07
8 processes per node	00:06:39	00:03:09	00:01:35	00:00:49	00:00:23	00:00:09	00:00:05

(d) Mesh resolution $N \times N = 8192 \times 8192$, system dimension 67,108,864

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	04:36:37	02:20:54	01:14:14	00:38:47	00:19:31	00:09:20	00:04:36
2 processes per node	02:07:34	01:08:57	00:34:11	00:18:09	00:08:25	00:04:14	00:02:08
4 processes per node	01:15:55	00:40:23	00:20:55	00:09:38	00:05:52	00:03:00	00:01:30
6 processes per node	00:56:59	00:29:28	00:14:43	00:07:32	00:04:26	00:02:29	00:01:23
8 processes per node	00:53:55	00:26:26	00:12:54	00:06:30	00:03:20	00:01:43	00:00:50

(e) Mesh resolution $N \times N = 16384 \times 16384$, system dimension 268,435,456

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	33:57:41	19:44:19	09:54:27	05:01:30	02:34:15	01:17:16	00:37:23
2 processes per node	16:21:30	08:31:15	04:31:04	02:23:42	01:09:54	00:34:00	00:17:01
4 processes per node	10:03:34	05:01:41	02:41:11	01:24:53	00:47:29	00:22:45	00:11:43
6 processes per node	08:20:03	04:04:07	02:02:50	01:02:55	00:32:32	00:17:35	00:08:59
8 processes per node	07:07:54	03:39:54	01:57:19	00:56:47	00:26:50	00:13:44	00:07:04

(f) Mesh resolution $N \times N = 32768 \times 32768$, system dimension 1,073,741,824

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
1 process per node	N/A	N/A	N/A	N/A	N/A	N/A	05:21:27
2 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	02:15:16
4 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	01:27:53
6 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	01:03:53
8 processes per node	N/A	N/A	N/A	N/A	N/A	N/A	00:55:07

Recap

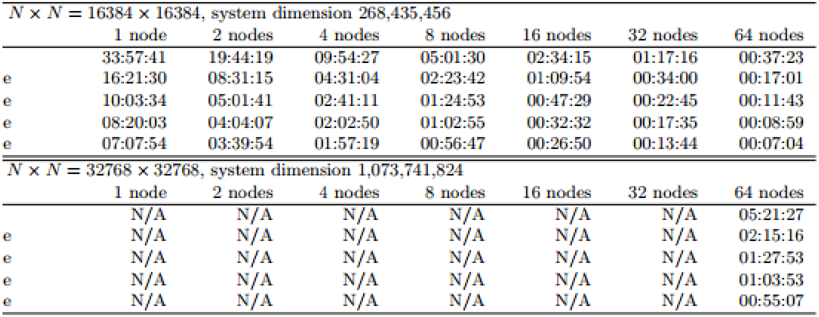


Figure : Lack of memory makes the problem infeasible.

$N \times N = 16384 \times 16384$, system dimension 268,435,456

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
	33:57:41	19:44:19	09:54:27	05:01:30	02:34:15	01:17:16	00:37:23
e	16:21:30	08:31:15	04:31:04	02:23:42	01:09:54	00:34:00	00:17:01
e	10:03:34	05:01:41	02:41:11	01:24:53	00:47:29	00:22:45	00:11:43
e	08:20:03	04:04:07	02:02:50	01:02:55	00:32:32	00:17:35	00:08:59
e	07:07:54	03:39:54	01:57:19	00:56:47	00:26:50	00:13:44	00:07:04

$N \times N = 32768 \times 32768$, system dimension 1,073,741,824

	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
	N/A	N/A	N/A	N/A	N/A	N/A	05:21:27
e	N/A	N/A	N/A	N/A	N/A	N/A	02:15:16
e	N/A	N/A	N/A	N/A	N/A	N/A	01:27:53
e	N/A	N/A	N/A	N/A	N/A	N/A	01:03:53
e	N/A	N/A	N/A	N/A	N/A	N/A	00:55:07

Challenges to Learning

What are the roadblocks to wide-spread adoption?

- Requires advanced computer and progamming knowledge
  - New programming language
  - Unix-like environment
  - Shell scripting
- Requires new algorithms
  - Can't avoid distributed memory environment
- New Software
- Access to appropriate hardware
  - Computer Cluster
  - GPUs
  - Hadoop Configured Cluster

Challenges to Learning

Any Solutions?

- Existing Code
- Parallel Libraries (Abstracting away parallelism)
- Amazon EC2, XSEDE

Parallel Libraries

Good parallel libraries will ease the process.

- The advantages of MPI over older message passing libraries are portability.
- “High-performance libraries often provide performance portability and hide the complexity of architecture-specific optimization details. Library reuse has been found to improve productivity and reduce bugs.”

Parallel Libraries built with MPI

In the distributed memory environment, what libraries are available?

ScaLAPACK	Solves dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems.
NAG Parallel	Linear Algebra, FFTs, Numerical Integration, Optimization
HDF5	Data model for storing and managing data
Others	PBGL, Parallel Boost Graph Library; PPM, Parallel Particle Mesh; ADLB, Asynchronous Dynamic Load Balancing

What is PETSc?

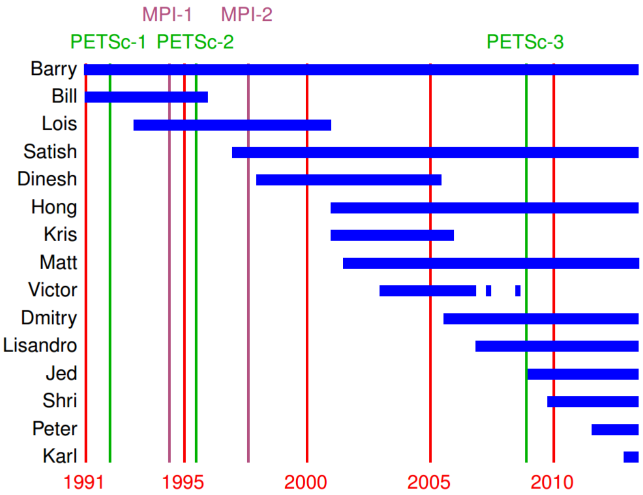
What is PETSc?

From the User's Manual,
"The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers."

Portable	Written using MPI
Extensible	Scalable and has many add-on packages such as TAO, SLEPc
Toolkit	Exposes methods at different levels of abstraction
Scientific Computation	Linear algebra, tools for PDEs, linear and nonlinear solvers

What is PETSc?

Out of the several different libraries for MPI mentioned, PETSc is one of the oldest, actively developed. The current version of PETSc is 3.4; released May 13, 2013.



What does PETSc do?

Purpose

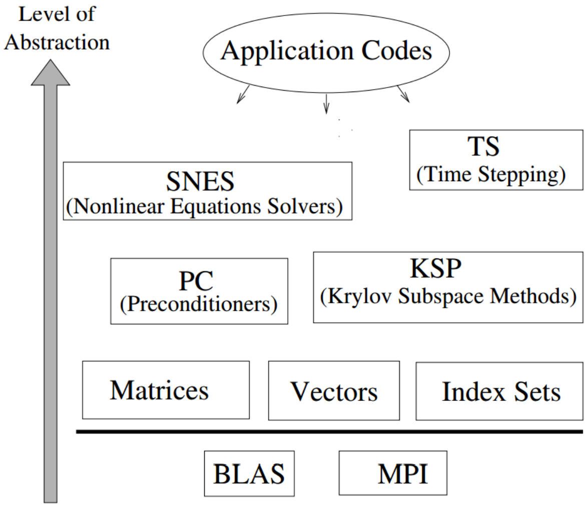
- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element
- Structured and Unstructered

What does PETSc do?

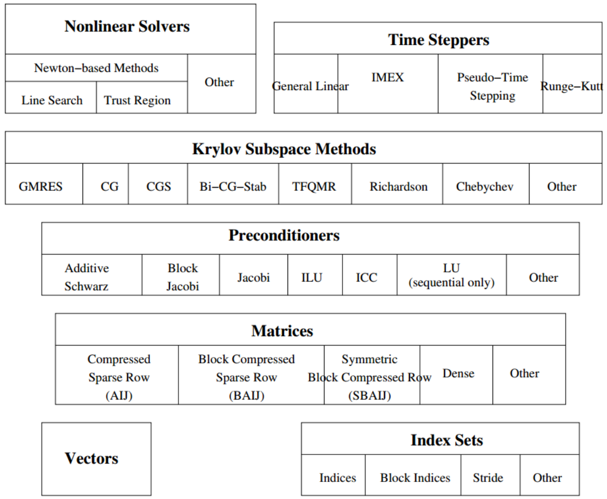
Contents

- Linear system solvers (sparse/dense, iterative/direct)
- Nonlinear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output
- Interface with many other programs (Matlab, Mathematica, Python, etc.)
- Makes use of other parallel paradigms (OpenMP, GPUs)

Organization oF PETSc



PETSc Numerical Components



Example of Scalability

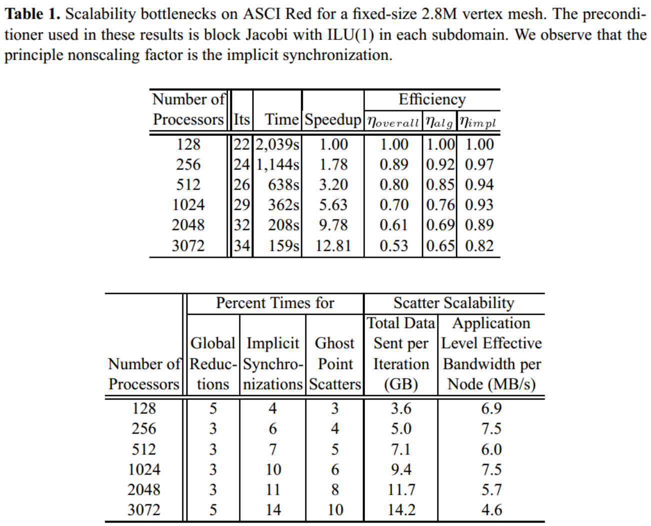


Table 1. Scalability bottlenecks on ASCI Red for a fixed-size 2.8M vertex mesh. The preconditioner used in these results is block Jacobi with ILU(1) in each subdomain. We observe that the principle nonscaling factor is the implicit synchronization.

Number of Processors	Its	Time	Speedup	Efficiency $\eta_{overall}$	η_{alg}	η_{impl}
128	22	2,039s	1.00	1.00	1.00	1.00
256	24	1,144s	1.78	0.89	0.92	0.97
512	26	638s	3.20	0.80	0.85	0.94
1024	29	362s	5.63	0.70	0.76	0.93
2048	32	208s	9.78	0.61	0.69	0.89
3072	34	159s	12.81	0.53	0.65	0.82

	Percent Times for			Scatter Scalability	
Number of Processors	Global Reductions	Implicit Synchronizations	Ghost Point Scatters	Total Data Sent per Iteration (GB)	Application Level Effective Bandwidth per Node (MB/s)
128	5	4	3	3.6	6.9
256	3	6	4	5.0	7.5
512	3	7	5	7.1	6.0
1024	3	10	6	9.4	7.5
2048	3	11	8	11.7	5.7
3072	5	14	10	14.2	4.6

PETSc: At a Glance

PETSc: At a Glance

PETSc Coding Features

- Easy creation of Vector and Matrix Objects
- Options for automatic distribution (parallel layout)
- Preprogrammed operations available

Object Creation: Vectors

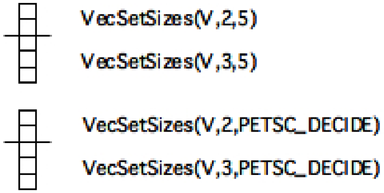
```
/* C */
Vec V;
ierr = VecCreate(MPI_COMM_SELF,&V); CHKERRQ(ierr);
ierr = VecDestroy(V); CHKERRQ(ierr);

! Fortran
Vec V
call VecCreate(MPI_COMM_SELF,V)
CHKERRQ(ierr)
call VecDestroy(V)
CHKERRQ(ierr);
```

Parallel Layout

Local size *m* and global size *M*

```
    VecSetSizes(Vec v, int m, int M);
```

Let PETSc specify global size.

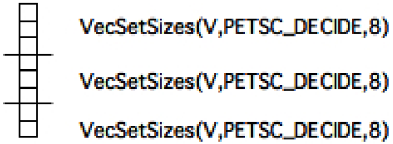


Parallel Layout

Local size *m* and global size *M*

```
    VecSetSizes(Vec v, int m, int M);
```

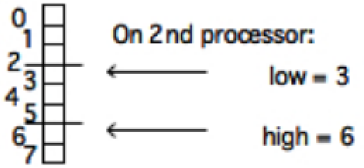
Let PETSc specify local size.


VecSetSizes(V,PETSC_DECIDE,8)
VecSetSizes(V,PETSC_DECIDE,8)
VecSetSizes(V,PETSC_DECIDE,8)

Parallel Layout

Query vector layout:

```
VecGetOwnershipRange(Vec x,PetscInt *low,PetscInt *high)
```



Preprogrammed Operations

```
VecAXPY(Vec y,PetscScalar a,Vec x);   /* y <- y + a x */
VecAYPX(Vec y,PetscScalar a,Vec x);   /* y <- a y + x */
VecScale(Vec x, PetscScalar a);
VecDot(Vec x, Vec y, PetscScalar *r); /* several variants */
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);
VecNorm(Vec x,NormType type, double *r);
VecSum(Vec x, PetscScalar *r);
VecCopy(Vec x, Vec y);
VecSwap(Vec x, Vec y);
VecPointwiseMult(Vec w,Vec x,Vec y);
VecPointwiseDivide(Vec w,Vec x,Vec y);
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);
VecMax(Vec x, int *idx, double *r);
VecMin(Vec x, int *idx, double *r);
VecAbs(Vec x);
VecReciprocal(Vec x);
VecShift(Vec x,PetscScalar s);
```

petsc4py

Create matrix and solver.

```
A = PETSc.Mat()
A.create(PETSc.COMM_WORLD)
A.setSizes([m*n, m*n])
A.setType('mpiaij')

ksp = PETSc.KSP()
ksp.create(PETSc.COMM_WORLD)
ksp.setOperators(A)
ksp.setFromOptions()
ksp.solve(b, x)
```

petsc4py example

Example from TACC

```
import petsc4py, sys
petsc4py.init(sys.argv)

from petsc4py import PETSc

# grid size and spacing
m, n  = 32, 32
hx = 1.0/(m-1)
hy = 1.0/(n-1)

# create sparse matrix
A = PETSc.Mat()
A.create(PETSc.COMM_WORLD)
A.setSizes([m*n, m*n])
A.setType('aij') # sparse

# precompute values for setting
# diagonal and non-diagonal entries
diagv = 2.0/hx**2 + 2.0/hy**2
offdx = -1.0/hx**2
offdy = -1.0/hy**2
```

```
# loop over owned block of rows on this
# processor and insert entry values
Istart, Iend = A.getOwnershipRange()
for I in xrange(Istart, Iend) :
    A[I,I] = diagv
    i = I//n    # map row number to
    j = I - i*n # grid coordinates
    if i> 0  : J = I-n; A[I,J] = offdx
    if i< m-1: J = I+n; A[I,J] = offdx
    if j> 0  : J = I-1; A[I,J] = offdy
    if j< n-1: J = I+1; A[I,J] = offdy

# communicate off-processor values
# and setup internal data structures
# for performing parallel operations
A.assemblyBegin()
A.assemblyEnd()
```

petsc4py example

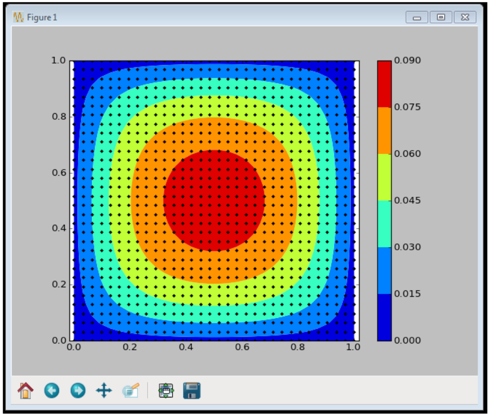
Example from TACC

```
# create linear solver
ksp = PETSc.KSP()
ksp.create(PETSc.COMM_WORLD)
# use conjugate gradients
ksp.setType('cg')
# and incomplete Cholesky
ksp.getPC().setType('icc')
# obtain sol & rhs vectors
x, b = A.getVecs()
x.set(0)
b.set(1)
# and next solve
ksp.setOperators(A)
ksp.setFromOptions()
ksp.solve(b, x)
```

```
try:
   from matplotlib import pylab
except ImportError:
   raise SystemExit("matplotlib
not available")
from numpy import mgrid
X, Y =  mgrid[0:1:1j*m,0:1:1j*n]
Z = x[...].reshape(m,n)
pylab.figure()
pylab.contourf(X,Y,Z)
pylab.plot(X.ravel(),Y.ravel(),'.k')
pylab.axis('equal')
pylab.colorbar()
pylab.show()
```

petsc4py example

Example from TACC



Uses in Economics

Continuous-time Life Cycle

A simple case of a life-cycle model is given as follows:

$$\max_{c} \int_0^T e^{-1\rho t} u(c) \, \mathrm{d}t$$

such that

$$\dot{A} = f(a) + w(t) - c(t)$$

$$A(0) = A(T) = 0$$

where u(c) is a concave utility function over consumption c, $w(t)$ is the wage rate at time t, $A(t)$ is assets at time t, and $f(A)$ is the return on invested assets. The boundary conditions reflect the assumption that assets are initially and terminally zero.

Continuous-time Life Cycle

The solution to this problem is derived from conditions on the Hamiltonian (for further details, see Kamien and Schwatrz, 2012), $H = u(c)\lambda(f(A) + w(t) - c)$. The costate equation is $\dot{\lambda} = \rho\lambda - \lambda f'(A)$. Furthermore, the maximum principle implies the condition $0 = u'(c) - \lambda$, and thus $c = C(\lambda)$. And this results in the system that characterizing the solution:

$$\dot{A} = f(A) + w - C(\lambda)$$
$$\dot{\lambda} = \lambda(\rho - f'(A))$$

with the same boundary conditions

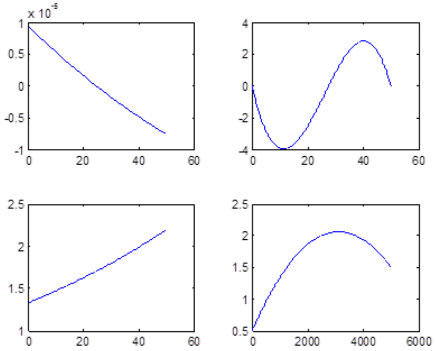
$$A(0) = A(T) = 0.$$

Continuous-time Life Cycle

discount rate	$\rho = 0.05$
time	$t \in [0, 50]$
return of assets	$f(A) = rA(t)$
interest rate	$r = 0.06$
wage function	$w(t) = 0.5 + \frac{5t}{T} - 4\left(\frac{t}{T}\right)^2$
utility function	$u(c) = ln(c)$

Continuous-time Life Cycle

Figure : Solution values plotted v. time: top-Left, RHS; top-right, assets; bottom-left, consumption; bottom-right, wages

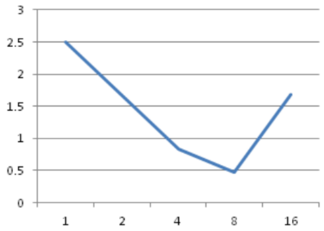


Speedup of Numerical Solution (Finite Differences)

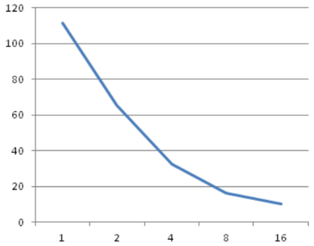
		PETSc, with p the number of processors				
Grid Points	Matlab	p=1	p=2	p=4	p=8	p=16
1×10^6	14.812566	0.809095	0.533468	0.288382	0.470327	0.971024
2×10^6	29.894963	1.681147	1.072255	0.562313	0.370278	1.330877
3×10^6	45.445266	2.502443	1.659407	0.838488	0.467811	1.688088
5×10^6	75.646605	4.451207	2.825604	1.427547	0.782719	0.703109
1×10^7	146.889423	9.041923	5.981552	2.940846	1.57849	1.046028
1×10^8	1496.928579	111.766608	65.19671	32.287323	16.023304	10.048579
1×10^9	***	***	*	*	*	92.058017

Table : Execution Times. Note: * = timing not yet available, *** = timing not possible—not enough memory! The matrix is a sparse $N \times N$.

Speedup of Numerical Solution (Finite Differences)



(A) $N = 3 \times 10^6$.



(B) $N = 1 \times 10^8$.

FIGURE 2. Plot of execution times (in seconds, on the y-axis) v. number of processors on the x-axis.

Overlapping Generations

See blackboard...

Conclusion

This is a work in progress. But it's important to note the techologies out there.

Credits

Some of the information and material for this slideshow have been adapted from various sources:

- Texas Advanced Computing Center
- Argonne National Laboratory Documentation
- PETSc User Manual
- Others...