Introduction to Scientific Computing on Boston University's IBM p-series Machines

Doug Sondak
sondak@bu.edu

Boston University
Scientific Computing
and Visualization

BOSTON UNIVERSITY

SCV

Outline

- Introduction
- Hardware
- Account Information
- Batch systems
- Compilers
- Parallel Processing
- Profiling
- Libraries
- Debuggers
- Other Software

Introduction

- What is different about *scientific* computing?
  - Large resource requirements
    - CPU time
    - Memory
  - Some excellent science can be performed on a PC, but we won't deal with that here.
  - Some non-scientific computing requires large resources, and the following material *will* be applicable.

IBM p690

- 1.3 GHz Power4 processors
  - 2 processors per chip
- shared memory
- 3 machines, *kite, frisbee, and pogo,* with
  - 32 processors each
  - 32 GB memory each
- 1 machine, *domino*, with
  - 16 processors
  - 16 GB memory

IBM p655

- 1.1 GHz Power4 processors
  - 2 processors per chip
- shared memory
- 6 machines, *twister*, *scrabble*, *marbles*, *crayon*, *litebrite*, and *hotwheels*, each with
  - 8 processors
  - 16 GB memory
- ***twister*** is login machine for p690/p655

IBM p655 (cont'd)

- Three additional machines
  - *jacks, playdoh, slinky*
  - 8 1.7 GHz processors
  - 8 GB memory
  - priority given to CISM group
    - if you don't know what CISM is, you're not in it!
  - charged at a higher rate proportional to higher clock speed

IBM p655 Data Caches

- L1
  - 32 KB per processor (64 KB per chip)
- L2
  - 1.41 MB
  - shared by both procs. on a chip
  - unified (data, instructions, page table entries)
- L3
  - 128 MB
  - shared by group of 8 processors
  - off-chip

Account Information and Policies

- to apply for an account

  http://scv.bu.edu/

  - click on accounts → applications
  - click on apply for a new project

- general information
  - click on Information for New SCF Users

Account Information and Policies (cont'd)

- account balance information
  - go to https://acct.bu.edu/SCF/UsersData/
  - click on your username
  - click on "Project Data"
- home disk space is limited
  - for larger requirements, apply for /project directory
  - http://scv.bu.edu/accounts/applications.html
    - click on request a Project Disk Space allocation
- interactive runs limited to 10 CPU-min.
  - use batch system for longer jobs

Archive System

- "archive" is a facility for long- or short-term storage of large files.
- storage

  archive *filename*
- retrieval

  archive –retrieve *filename*
- works from twister, skate, or cootie
- see man page for more info.

Batch System - LSF

- bqueues command
  - lists queues
  - shows current jobs

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
donotuse	1	Open:Active	-	-	-	-	0	0	0	0
p4-test	1	Open:Active	4	4	-	-	0	0	0	0
p4-short	1	Open:Active	2	1	-	-	0	0	0	0
p4-long	1	Open:Active	16	5	-	8	35	20	15	0
p4-verylong	1	Open:Active	2	1	-	-	2	0	2	0
p4-mp4	1	Open:Active	1	-	-	-	0	0	0	0
p4-mp8	1	Open:Active	2	1	-	1	1	0	1	0
p4-cism-mp8	1	Open:Active	3	-	-	1	1	0	1	0
p4-mp16	1	Open:Active	4	2	-	2	4	0	4	0
p4-ibmsur-mp16	1	Open:Active	1	1	-	-	1	0	1	0
p4-mp32	1	Open:Active	1	1	-	-	1	0	1	0

LSF (cont'd)

- **donotuse** and **p4-test** queues are for administrative purposes
- **p4-short**, **p4-long**, and **p4-verylong** queues are for serial (single processor) jobs
- "mp" queues are for parallel processing
  - suffix indicates maximum number of processors
    - **p4-mp8** queue is for parallel jobs with up to 8 processors
- **p4-cism-mp8** and **p4-ibmsur-mp16** are only available to members of certain projects
  - If you're not sure if you can use them, you probably can't!

LSF (3)

- a few important fields in "bqueues" output:
  - MAX – maximum number of jobs allowed to run at a time
  - JL/U – maximum number of jobs allowed to run at a time per user
  - NJOBS – number of jobs in queue
  - PEND – number of jobs pending, i.e., waiting to run
  - RUN – number of jobs running
- choose queue with appropriate number of processors

LSF (4)

- queue time limits

queue	CPU limit (hrs.)	wall-clock limit (hrs.)
p4-short	2	2.5
p4-long	32	40
p4-verylong	64	80
p4-mp4	16	5
p4-mp8	32	5
p4-cism-mp8	32	5
p4-mp16	64	5
p4-mp32	128	5
p4-ibmsur-mp16	128	9

LSF (5)

- bsub command
  - simplest way to run:

    bsub –q *qname myprog*
  - suggested way to run:
    - create a short script
    - submit the script to the batch queue
    - this allows you to set environment variables, etc.

LSF (6)

- sample script

```
#!/bin/tcsh
    setenv OMP_NUM_THREADS 4
    mycode < myin > myout
```

- make sure script has execute permission
  - chmod 755 myscript

LSF (7)

- bjobs command
  - with no flags, gives status of all your current batch jobs
  - -q *quename* to specify a particular queue
  - -u all to show all users' jobs

twister:~ % bjobs -u all -q p4-mp16

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST
257949	onejob	RUN	p4-mp16	twister	frisbee
257955	quehog	RUN	p4-mp16	twister	frisbee
257956	quehog	RUN	p4-mp16	twister	kite
257957	quehog	PEND	p4-mp16	twister	
257958	quehog	PEND	p4-mp16	twister	
257959	quehog	PEND	p4-mp16	twister	

LSF (8)

- additional details on queues

  http://scv.bu.edu/SCV/scf-techsumm.html

Compilers

- IBM AIX native compilers, e.g., xlc, xlf95
- GNU (gcc, g++, g77)

AIX Compilers

- different compiler *names* (really scripts) perform some tasks which are handled by compiler *flags* on many other systems
  - parallel compiler names differ for SMP, message-passing, and combined parallelization methods
  - do *not* link with MPI library (-lmpi)
    - taken care of automatically by specific compiler name (see next slide)

AIX Compilers (cont'd)

	Serial	MPI	OpenMP	Mixed
Fortran 77	xlf	mpxlf	xlf_r	mpxlf_r
Fortran 90	xlf90	mpxlf90	xlf90_r	mpxlf90_r
Fortran 95	xlf95	mpxlf95	xlf95_r	mpxlf95_r
C	cc	mpcc	cc_r	mpcc_r
	xlc	mpxlc	xlc_r	mpxlc_r
C++	xlC	mpCC	xlC_r	mpCC_r

AIX Compilers (3)

- xlc default flags
  - -qalias=ansi
    - optimizer assumes that pointers can only point to an object of the same type (potentially better optimization)
  - -qlanglvl=ansi
    - ansi c

AIX Compilers (4)

- xlc default flags (cont'd)
  - -qro
    - string literals (*e.g.*, char *p = "mystring";) placed in "read-only" memory (text segment); cannot be modified
  - -qroconst
    - constants placed in read-only memory

AIX Compilers – (5)

- cc default flags
  - -qalias = extended
    - optimizer assumes that pointers may point to object whose address is taken, regardless of type (potentially weaker optimization)
  - -qlanglvl=extended
    - extended (not ansi) c
    - "compatibility with the RT compiler and classic language levels"

AIX Compilers – (6)

- cc default flags (cont'd)
  - -qnoro
    - string literals (e.g., char *p = "mystring";) can be modified
    - may use more memory than –qro
  - -qnoroconst
    - constants not placed in read-only memory

AIX Compilers – (7)

- 64-bit
  - -q64
  - use if you need more than 2GB
  - **has nothing to do with accuracy**, simply increases address space

AIX Compilers – (8)

- optimization levels
  - -O basic optimization
  - -O2 same as -O
  - -O3 more aggressive optimization
  - -O4 even more aggressive optimization; optimize for current architecture; IPA
  - -O5 aggressive IPA

AIX Compilers – (9)

- If using O3 or below, can (should!) optimize for local hardware (done automatically for -O4 and -O5):

-qarch=auto	optimize for resident architecture
-qtune=auto	optimize for resident processor
-qcache=auto	optimize for resident cache

AIX Compilers – (10)

- If you're using IPA and you get warnings about partition sizes, try

  -qipa=partition=large

- 32-bit default data segment limit 256MB
  - data segment contains static, common, and allocatable variables and arrays
  - can increase limit to a maximum of 2GB with 32-bit compilation

    -bmaxdata:0x80000000

  - bmaxdata not needed with -q64

AIX Compilers – (11)

- -O5 does *not* include function inlining
- function inlining flags:

-Q	compiler decides what functions to inline
-Q+func1:func2	only inline specified functions
-Q -Q-func1:func2	let compiler decide, but do not inline specified functions

AIX Compilers – (12)

- array bounds checking
  - -C or -qcheck
  - slows code down a lot
- floating point exceptions
  - -qflttrap=ov:und:zero:inv:en -qsigtrap -g
  - overflow, underflow, divide-by-zero, invalid operation
  - :en is required to "enable" the traps
  - -qsigtrap results in a trace for exception
  - -g lets trace report line number of exception

AIX Compilers – (13)

- compiler documentation:

  http://twister.bu.edu/

Parallel Processing

- MPI
- OpenMP

AIX MPI

- different conventions than you may be used to from other systems
- compile using compiler name with mp prefix, e.g., mpcc
  - this runs a script
  - automatically links to MPI libraries
  - do *not* use -lmpi

AIX MPI (cont'd)

- Do not use mpirun!
- mycode –procs 4
  - number of procs. specified using –procs, not –np
- -labelio yes
  - labels output to std. out with process no.
  - also can set environment variable MP_LABELIO to yes

AIX OpenMP

- use _r suffix on compiler name
  - e.g., xlc_r
- use –qsmp=omp flag
  - tells compiler to interpret OpenMP directives
- automatic parallelization

  -qsmp
  - sometimes works ok; give it a try

AIX OpenMP (cont'd)

- automatic parallelization (cont'd)

  -qreport=smplist

  - produces listing file
  - mycode.lst
  - includes information on parallelization of loops

- per-thread stack limit
  - default 4 MB
  - can be increased with environment variable

    setenv XLSMPOPTS $XLSMPOPTS\:stack=*size*

    where *size* is the size in bytes

AIX OpenMP (3)

- must declare OpenMP functions

  integer OMP_GET_NUM_THREADS

- running is the same as on other systems, e.g.,

  setenv OMP_NUM_THREADS 4

  mycode < myin > myout

Profiling

- profile tells you how much time is spent in each routine
- use gprof
- compile with -pg
- file gmon.out will be created when you run
- gprof >& myprof
  - note that gprof output goes to std err (&)
- for multiple procs. (MPI), copy or link gmon.out.*n* to gmon.out, then run gprof

gprof Call Graph

ngranularity: Each sample hit covers 4 bytes. Time: 435.04 seconds

time in routines called from specified routine

total time for run

index	%time	self	descendents	called/total called+self	parents name children	index
		0.00	340.50	1/1	.__start [2]	
[1]	78.3	0.00	340.50	1	.main [1]	
		2.12	319.50	10/10	.contrl [3]	
		0.04	7.30	10/10	.force [34]	
		0.00	5.27	1/1	.initia [40]	
		0.56	3.43	1/1	.plot3da [49]	
		0.00	1.27	1/1	.data [73]	

time in specified routine

2.12 sec. spent in contrl

319.50 sec. spent in routines called from contrl

gprof Flat Profile

ngranularity: Each sample hit covers 4 bytes. Time: 435.04 seconds

89.17 sec. spent in *conduct*

conduct was called 10 times

8917 m-sec. spent in each call to *conduct*

10918 m-sec. spent in each call to *conduct*, including routines called by *conduct*

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
20.5	89.17	89.17	10	8917.00	10918.00	.conduct [5]
7.6	122.34	33.17	323	102.69	102.69	.getxyz [8]
7.5	154.77	32.43				.__mcount [9]
7.2	186.16	31.39	189880	0.17	0.17	.btri [10]
7.2	217.33	31.17				.kickpipes [12]
5.1	239.58	22.25	309895200	0.00	0.00	.rmnmod [16]
2.3	249.67	10.09	269	37.51	37.51	.getq [24]

⋮

xprofiler

- graphical interface to gprof
- compile with -g -pg -Ox
  - Ox represents whatever level of optimization you're using (e.g., O5)
- run code
  - produces gmon.out file
- type xprofiler command

AIX Scientific Libraries

- linear algebra
- matrix operations
- eigensystem analysis
- Fourier analysis
- sorting and searching
- interpolation
- numerical quadrature
- random number generation

AIX Scientific Libraries (cont'd)

- ESSLSMP
  - for use with "SMP processors" (that's us!)
  - some serial, some parallel
    - parallel versions use multiple threads
    - thread safe; serial versions may be called within multithreaded regions (or on a single thread)
  - link with -lesslsmp

AIX Scientific Libraries (3)

- PESSLSMP
  - parallel message-passing version of library (e.g., MPI)
- link flags:

  -lpesslsmp -lesslsmp -lblacssmp

AIX Scientific Libraries (3)

- documentation – go to

  http://twister.bu.edu

  and click on

  Engineering and Scientific Subroutine Library (ESSL) V4.2 Guide and Reference

  or

  Parallel ESSL V3.2 Guide and Reference

AIX Fast Math

- MASS library
  - **M**athematical **A**cceleration **S**ub**S**ystem
- faster versions of some intrinsic Fortran functions
  - sqrt, rsqrt, exp, log, sin, cos, tan, atan, atan2, sinh, cosh, tanh, dnint, x**y
- work with Fortran or C
- differ from standard functions in last bit (at most)

AIX Fast Math (cont'd)

- simply link to mass library:

  Fortran: -lmass

  C: -lmass -lm

- sample approx. speedups

exp	2.4
log	1.6
sin	2.2
complex atan	4.7

AIX Fast Math (3)

- vector routines
  - require minor code changes
  - not portable
  - large potential speedup
- link with –lmassv
- subroutine calls
  - use prefix on function name
    - vs for 4-byte reals (single precision)
    - v for 8-byte reals (double precision)

AIX Fast Math (4)

- example: single-precision exponential

  call vsexp(y,x,n)

  - x is the input vector of length n
  - y is the output vector of length n

- sample speedups

	4-byte	8-byte
exp	9.7	6.7
log	12.3	10.4
sin	10.0	9.8
complex atan	16.7	16.5

AIX Fast Math (5)

- For details, see MASS documentation

  http://twister.bu.edu/

  - click on

    XL C/C++ Programming Guide v8.0

    or

    XL Fortran Optimization and Programming Guide v10.1

    and go to chapter on high performance libraries

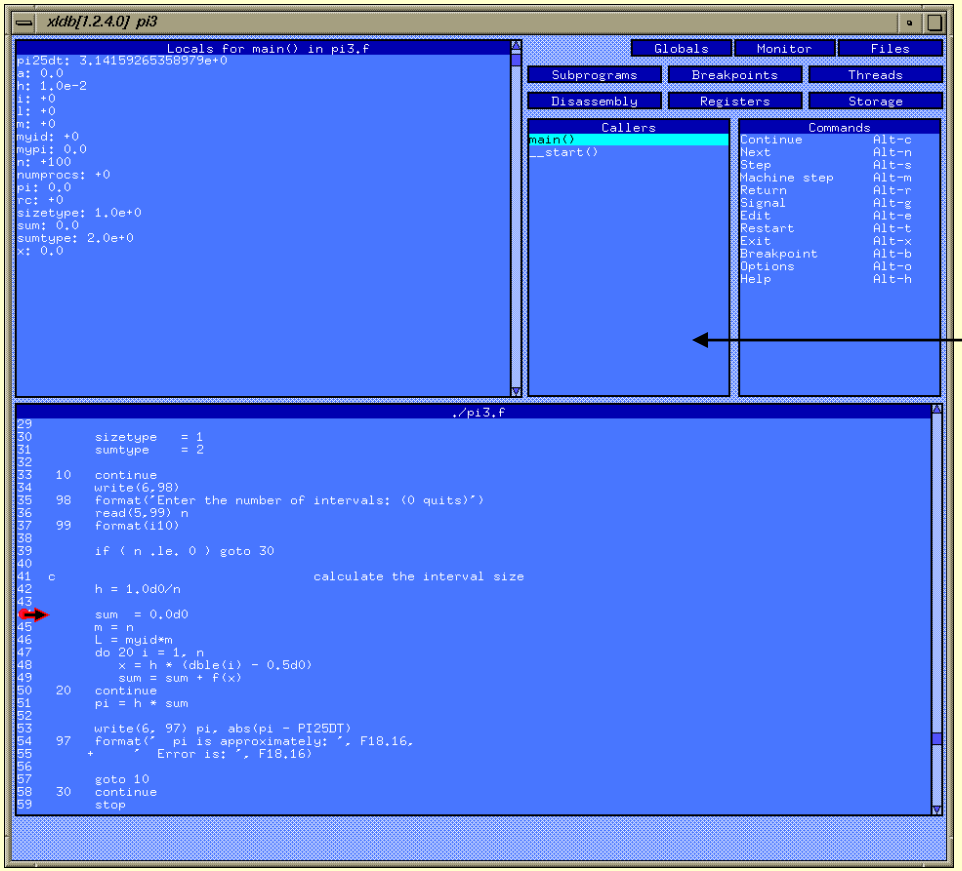
AIX Debuggers

- dbx - standard command-line unix debugger
- pdbx - parallel version of dbx
- xldb - debugger with graphical interface

xldb

- compile with –g, no optimization
- xldb mycode
  - window pops up with source, etc.
- group of blue bars at the top right
  - click on bar to open window
  - to minimize window, click on bar at top to get menu, click on "minimize"
- to set breakpoint, click on source line
- to navigate, see "commands" window

xldb (cont'd)



output →

source listing →

breakpoint →

these bars minimize/maximize windows ←

commands ←

calling routines ←

pdbx

- Command-line parallel debugger
  - parallel version of dbx
- Compile with –g, no optimization
- To start pdbx, give pdbx command followed by normal run command

```
pdbx  pi3  –procs 2
```

pdbx (cont'd)

- if source is not in working directory, can specify location:

```
pdbx  pi3   –procs 2    -I ../../sourcedir
```

pdbx (3)

```
pdbx Version 3, Release 2 -- Feb 23 2003 15:55:50
0:Core file "
0:" is not a valid core file (ignored)
1:Core file "
1:" is not a valid core file (ignored)
1:reading symbolic information ...
0:reading symbolic information ...
0:[1] stopped in pi3 at line 20 ($t1)
0:   20   program pi3
1:[1] stopped in pi3 at line 20 ($t1)
1:   20   program pi3
0031-504  Partition loaded ...

pdbx(all)
```

Results from each process are labeled with the process number

always get these irrelevant messages about core files

automatically stops at 1st executable line in code

pdbx prompt

pdbx (4)

```
pdbx(all) list
  0:   21
  0:   22     include 'mpif.h'
  0:   23
  0:   24     double precision  PI25DT
  0:   25     parameter       (PI25DT = 3.14159265358979323846264338327950288419d0)
  0:   26
  0:   27     double precision  mypi, pi, h, sum, x, f, a
  0:   28     integer n, myid, numprocs, i, rc
  0:   29     ! function to integrate
  0:   30     f(a) = 4.d0 / (1.d0 + a*a)
  1:   21
  1:   22     include 'mpif.h'
  1:   23
  1:   24     double precision  PI25DT
  1:   25     parameter       (PI25DT = 3.14159265358979323846264338327950288419d0)
  1:   26
  1:   27     double precision  mypi, pi, h, sum, x, f, a
  1:   28     integer n, myid, numprocs, i, rc
  1:   29     ! function to integrate
  1:   30     f(a) = 4.d0 / (1.d0 + a*a)
```

lists next 10 lines on each processor

pdbx (5)

- List specified range of lines using comma as delimiter

```
pdbx(all) list 28,30
   0:   28      integer n, myid, numprocs, i, rc
   0:   29      ! function to integrate
   0:   30      f(a) = 4.d0 / (1.d0 + a*a)
   1:   28      integer n, myid, numprocs, i, rc
   1:   29      ! function to integrate
   1:   30      f(a) = 4.d0 / (1.d0 + a*a)
```

pdbx (6)

- specify process with on *procno* prefix
  - for list, next, etc.

```
pdbx(all) on 0 list 28,30
   0:   28      integer n, myid, numprocs, i, rc
   0:   29      ! function to integrate
   0:   30      f(a) = 4.d0 / (1.d0 + a*a)
```

pdbx (6)

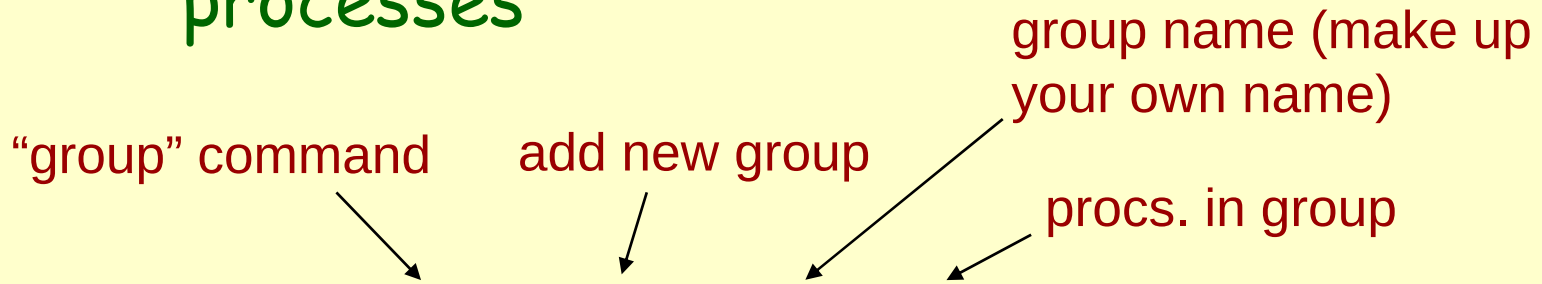
- on *procno* can also be used alone
  - subsequent commands only apply to specified process
  - current process shown in prompt

pdbx(all) on 2

pdbx(2)

pdbx (7)

- processes can be grouped
  - commands can be applied to subset of processes




```
pdbx(all) group   add   g03   0,3
0029-2040 2 tasks were added to group "g03".
```

pdbx (8)

- "on" command can be used with group name

  pdbx(all) on g03

  pdbx(g03)

- note change in prompt
- to change back to "all":

  pdbx(g03) on all

pdbx (9)

- breakpoints
  - stop at 30
  - stop in *subprogram*
- status lists all current breakpoints

```
pdbx(all) status
all:[0] stop in muiwl1
all:[1] stop at "../oldtempsource/muiwl1.F":
```

  - all means that it pertains to all processes

pdbx (10)

- to delete breakpoints

```
pdbx(all) status
all:[0] stop in muowl2
all:[1] stop at "../source_v14_kbreakup/muowl2.F":632
all:[2] stop at "../source_v14_kbreakup/muowl2.F":697

pdbx(all) delete 0
pdbx(all) delete 1

pdbx(all) status
all:[2] stop at "../source_v14_kbreakup/muowl2.F":697
```

pdbx (11)

- breakpoints can be qualified using logical expressions
  - logical expressions have C syntax, even when using Fortran

```
pdbx(all) stop at 271 if( (i = = 50) && (j = = 10) && (k = = 5) )
all:[1] stop at "../oldtempsource/muiwl1.F":271 if( (I = = 50) &&
    (j = = 10) && (k = = 5) )
```

  - must use ( ) for multiple conditions
  - may be slow

pdbx (12)

- next marches to next line in source (executes current line)
  - will step over function/subroutine calls
- step is the same as next except that it will step *into* function/subroutine calls
- both next and step can take numerical argument to specify number of lines to execute

  next 10

pdbx (13)

- print prints value of specified variable

```
pdbx(all) print k
0:4
1:7
```

value

process number

- print array values with either ( ) or [ ]

```
pdbx(all) print rvlu[5]
0:0.996023297
1:0.985406339
```

pdbx (14)

- print range of array values:

```
pdbx(3) p fval(12..16)
   3:(12) = 0.0530325808
   3:(13) = 0.0146476058
   3:(14) = 0.0307097323
   3:(15) = 0.0095740892
   3:(16) = 0.00736919558
```

pdbx (15)

- to get information on a variable's declaration

```
pdbx(3) whatis stotmxloc
    3: real*4  stotmxloc(305,41)
```

Other Scientific Software

- Matlab
- Mathematica
- Maple

Matlab

- language for scientific computing
- very powerful and intuitive
- can be used to solve small or medium sized problems
  - major number crunching can get slow
- excellent plot package
- we have an old version on our AIX machines
  - The Mathworks no longer supports AIX
  - latest version available on linux cluster

Other Scientific Software – Matlab (cont'd)

- tutorial:

  http://scv.bu.edu/Tutorials/MATLAB/

Other Scientific Software - Mathematica

- similar to Matlab
- performs symbolic equation manipulation

  http://scv.bu.edu/Graphics/mathematica.html

Other Scientific Software - Maple

- performs symbolic equation manipulation as well as other mathematical functions
- available on AIX systems and linux cluster
  - suggest using cluster since it's faster
- type "xmaple" at prompt
  - look at help => new users for good tutorials

Human Help

- scientific computing, parallelization, optimization

  Doug Sondak sondak@bu.edu

  Kadin Tseng kadin@bu.edu

- administrative or system issues

  bugs@twister.bu.edu