Computing on Mio Libraries and GPUs

Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu
Director - CSM High Performance Computing
Director - Golden Energy Computing Organization

http://inside.mines.edu/mio/tutorial/

Libraries

- Why you should be interested
- What is available
- Issues
- Some results
- We have examples

Why?

- Many common problems have been solved
- Writing a good parallel routine is difficult
- “It’s not about the computer”

What?

•Intel-MKL

- BLAS
- BLACS
- **LAPACK**
- **ScaLAPACK**
- PBLAS
- Sparse Solver
- Vector Math Library (VML)
- Vector Statistical Library (VSL)
- Conventional DFTs and Cluster DFTs
- Partial Differential Equations support
- Optimization Solvers.

•Fourier Transforms

- p3dfft
- fftw

•NAG Library (Not installed)

- Optimization, both Local and Global
- Linear, quadratic, integer and nonlinear programming and least squares problems
- Ordinary and partial differential equations, and mesh generation
- Solution of dense, banded and sparse linear equations and eigenvalue problems
- Solution of linear and nonlinear least squares problems
- Curve and surface fitting and interpolation
- Special functions
- Numerical integration and integral equations
- Roots of nonlinear equations (including polynomials)
- Option Pricing Formulae
- Wavelet Transforms
- Statistical facilities
- Random number generation
- Simple calculations on statistical data
- Correlation and regression analysis
- Multivariate methods
- Analysis of variance and contingency table analysis
- Time series analysis
- Nonparametric statistics

•PETSC

- Solution of linear and non-linear partial differential equations
- Focus on problems discretized using semi or fully implicit methods

•Cuda

- CULA Linear Algebra Library from EMPhotonics

Issues

- Can be difficult to link
- Can be difficult to set up, especially parallel
- Portability

ScaLAPACK

- Scalable Linear Algebra PACKage
- Developing team from University of Tennessee, University of California Berkeley, ORNL, Rice U.,UCLA, UIUC etc.
- Support in Commercial Packages
  - NAG Parallel Library
  - IBM PESSL
  - CRAY Scientific Library
  - IMSL
  - MKL - INTEL

ScaLAPACK contains:

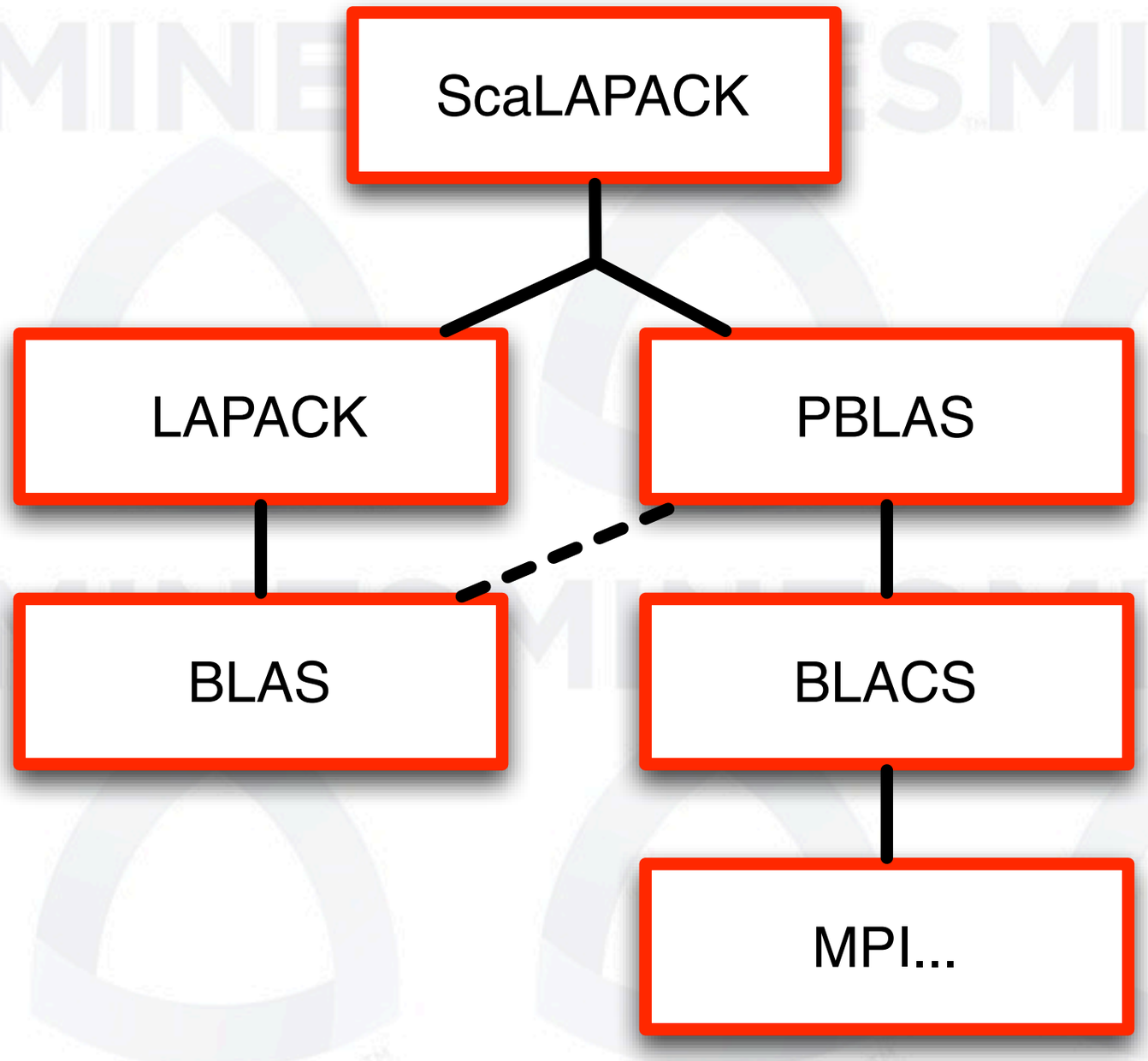
- LAPACK
  - Routines are single-PE Linear Algebra solvers, utilities, etc.
- BLAS
  - Single-PE processor work-horse routines (vector, vector-matrix & matrix-matrix)

These are useful in there own right

ScaLAPACK contains:

- PBLAS
  - Parallel counterparts of BLAS. Relies on BLACS for blocking and moving data.
- BLACS
  - Constructs 1-D & 2-D processor grids for matrix decomposition and nearest neighbor communication patterns. Uses message passing libraries (MPI) for data movement.

Dependencies & Parallelism of Component Packages



ScaLAPACK uses distributed data

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

Global Matrix

Global Matrix (9x9)
Block Size =2x2
Cyclic on 2x3 PE Grid

11	12	17	18		
21	22	27	28		
51	52	57	58		
61	62	67	68		
91	92	97	98		

PE (0,0)

13	14	19			
23	24	29			
53	54	59			
63	64	69			
93	94	99			

PE (0,1)

15	16				
25	26				
55	56				
65	66				
95	96				

PE (0,2)

31	32	37	38		
41	42	47	48		
71	72	77	78		
81	82	87	88		

PE (1,0)

33	34	39			
43	44	49			
73	74	79			
83	84	89			

PE (1,1)

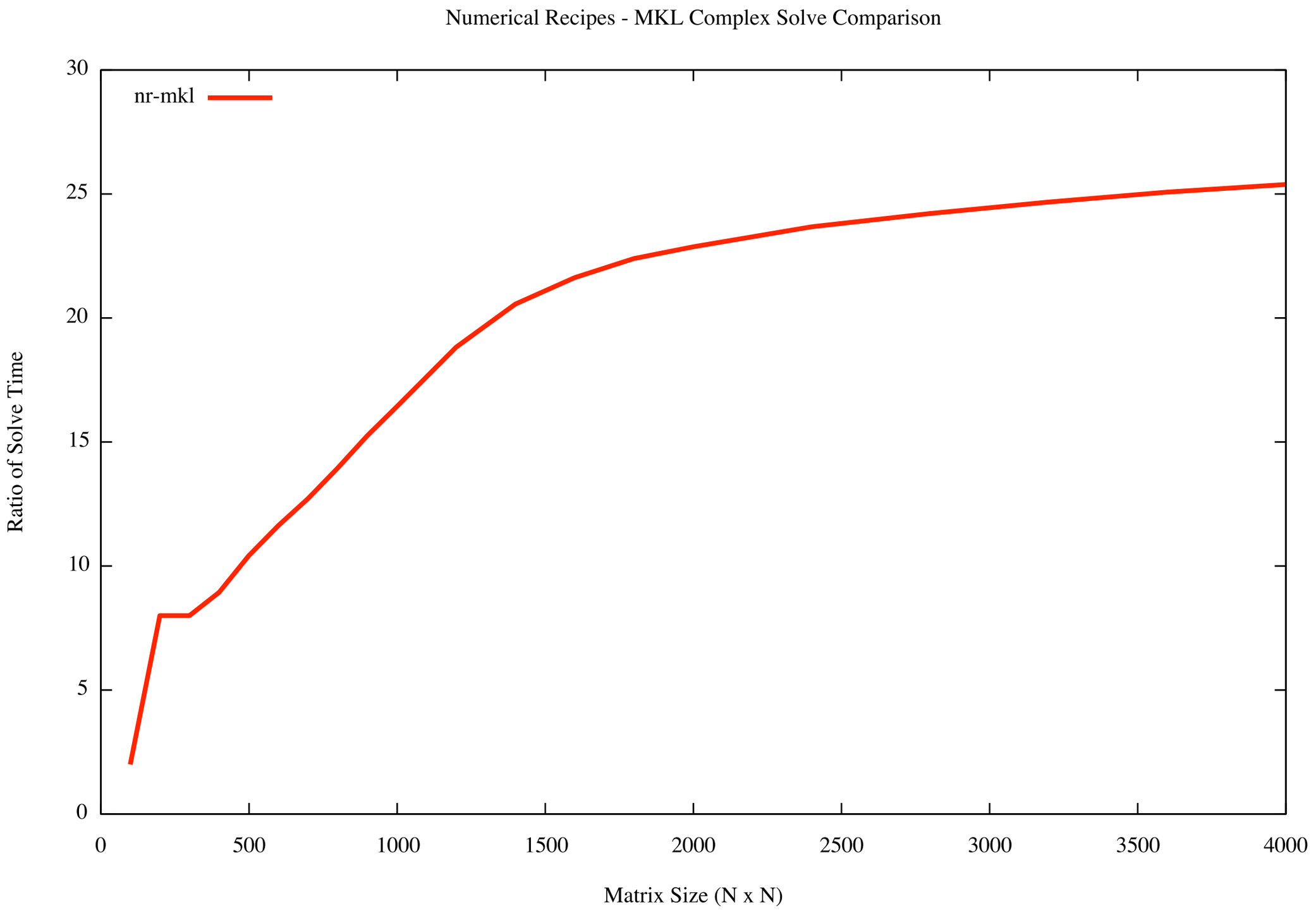
35	36				
45	46				
75	76				
85	86				

PE (1,2)

If you use ScaLAPACK or LAPACK or RA/MIO

Use the Intel MKL version

Numerical Recipes / MKL



MKL Notes

- Manual is huge - 3646 pages
- ~~Two~~ 3 hardest parts
  - Finding the correct routine
  - Getting the arguments correct
  - Getting it to link correctly

Some Examples

- ScaLAPACK
  - http://geco.mines.edu/software/mkl/index.shtml
  - Includes complete general parallel dense solve
- LAPACK
  - http://geco.mines.edu/tesla/culaexample/index.shtml
  - Includes complete complex linear solve
  - Also talks about GPU version
- Starting From Scratch
  - http://geco.mines.edu/software/mkl/casestudy.html
- FFT
  - http://geco.mines.edu/software/mkl/fftw/index.html
  - Don't run these on the head node

Compile and Link for LAPACK/MKL

```
ifort -O3  fourd.f90 \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_intel_lp64.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_core.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_sequential.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_core.a \
-lpthread \
-o fourd.mkl
```

Compile and Link for ScaLAPACK/MKL

```
mpif90 -O1   linsolve.f90 \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_scalapack_lp64.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_intel_lp64.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_blacs_openmpi_lp64.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_sequential.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_core.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_sequential.a \
/opt/intel/Compiler/11.1/069/mkl/lib/em64t/libmkl_core.a \
-lpthread -i_dynamic \
-o lin_f
```

-mkl option helps

Can often replace the list of libraries with the command line option -mkl. See the example links given above for make files

```
icc   test.c   -I/opt/intel/Compiler/11.1/069/mkl/include/fftw -mkl -o
test_c
```

```
ifort test.f90 -I/opt/intel/Compiler/11.1/069/mkl/include/fftw -mkl -o
test_f
```

```
icc test2.c -o test2_c -I/opt/intel/Compiler/11.1/069/mkl/include/fftw -
L/opt/intel/Compiler/11.1/069/mkl/lib/em64t -lfftw2xc_intel -mkl
```

```
ifort test2.f -o test2_f -I/opt/intel/Compiler/11.1/069/mkl/include/fftw
-L/opt/intel/Compiler/11.1/069/mkl/lib/em64t -lfftw2xf_intel -mkl
```

PETSc

http://www.mcs.anl.gov/petsc/petsc-as/

PETSc, **pronounced PET-see** (the S is silent), is a **suite of data structures and routines for the scalable (parallel) solution** of scientific applications modeled by partial differential equations. **Includes sparse iterative solvers.**

PETSc includes a large suite of **parallel linear, nonlinear equation solvers and ODE integrators** that are easily used in application codes written in C, C++, Fortran and Python

Applications of PETSc (243)

- Nano-simulations (46)
- Biology/Medical(37)
  - Cardiology
  - Imaging and Surgery
- Fusion (13)
- Geosciences (29)
- Environmental/Subsurface Flow (30)
- Computational Fluid Dynamics (53)
- Wave propagation and the Helmholtz equation (12)
- Optimization (11)
- Fast Algorithms (2)
- Other Application Areas (79)

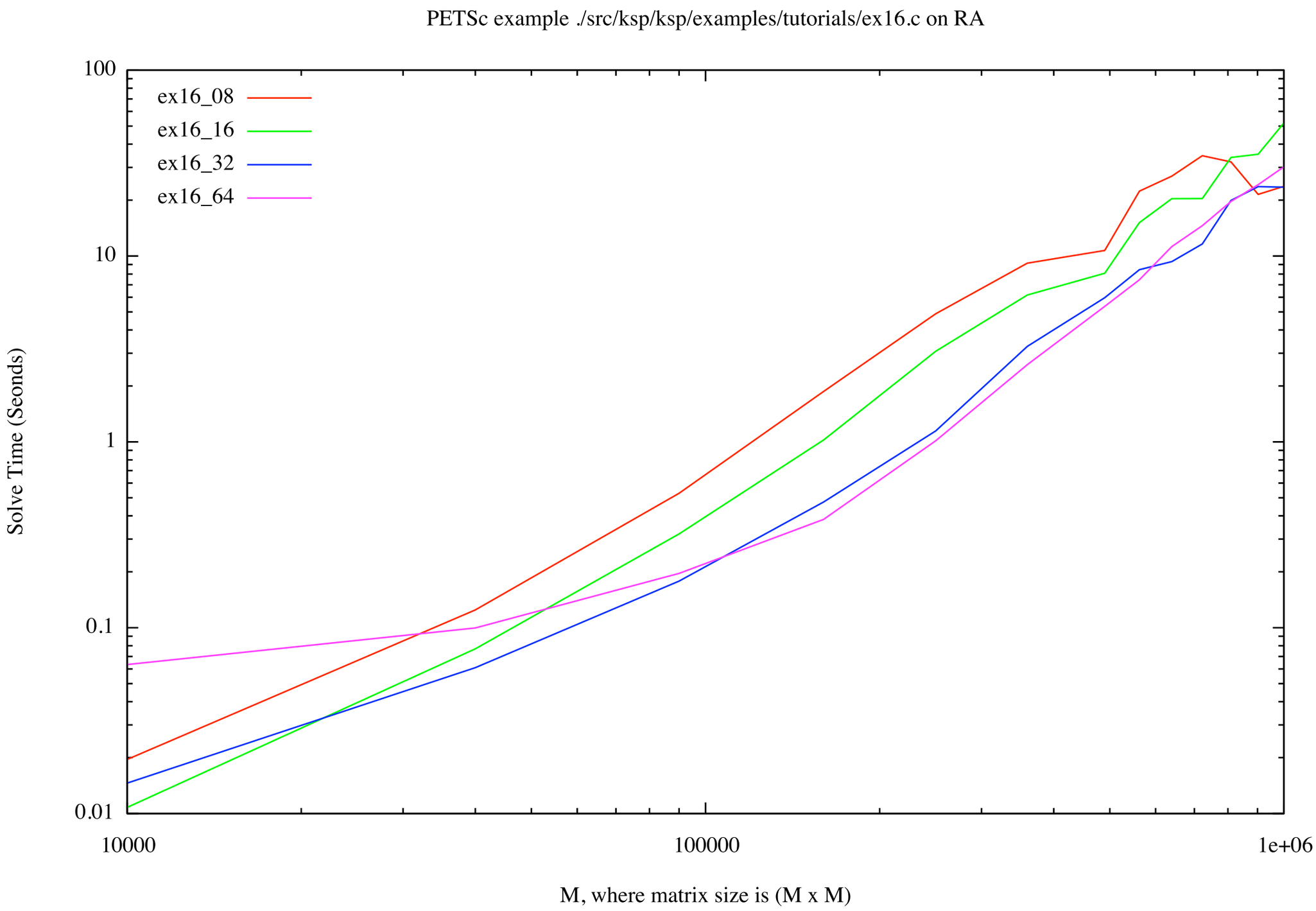
PETSc build on RA

August 2, 2010

```
mkdir petsc

cd petsc

wget http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.1-p4.tar.gz

tar -xzf *gz

cd petsc-3.1-p4

./config/configure.py \
      PETSC_ARCH=linux-intel  \
      --with-blas-lapack-dir=/opt/intel/Compiler/11.1/069/mkl \
      --with-mpi-dir=/opt/ra5_openmpi_intel/1.4.1

make \
     PETSC_DIR=/lustre/home/tkaiser/work/petsc/petsc-3.1-p4 \
     PETSC_ARCH=linux-intel \
     all
```

Builds fine. Test may fail because it tries to run an MPI job on the front end

Sparse Linear Solve on RA



Summary

- Don’t reinvent the wheel
- Look at available libraries
  - LAPACK/ScaLAPACK/MKL
  - PETSc
  - Domain Specific
- On RA and Mio use MKL for LAPACK/ ScaLAPACK

Links

- MKL LAPACK ScaLAPACK
  - http://geco.mines.edu/software/mkl/index.shtml
- LAPACK & CUDA Cula
  - http://geco.mines.edu/tesla/culaexample/index.shtml
- FFT
  - http://geco.mines.edu/software/mkl/fftw/index.html
- I am starting from scratch...
  - http://geco.mines.edu/software/mkl/casestudy.html
- PETSc
  - http://www.mcs.anl.gov/petsc/petsc-as/

Very complete example of MKL/LAPACK solver usage.
Used to compare versions of libraries

GPU Programming Platforms and Running

Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu

http://geco.mines.edu/tesla

What is it?

- GPU (graphic procession unit) programming is using a GPU to solve computational problems
  - People originally did actually just that
  - GPUs are evolving so that they are more useful for general processing
- The GPU is a coprocessor that is used for particular types of computation
- NVIDIA is big in this market
  - Tesla
  - Video cards

Programming for NVIDIA

- Massively thread based
- CUDA language
  - C superset
  - Adds methods to run threads on GPU and transfer data to/from GPU
  - Code that runs on GPU is called a Kernel
- Portland Group compiler has OpenMP like directives
- Libraries (Lapack subset)

Threads

- Can have many processing cores that share memory (think hundreds)
- Unlike OpenMP you have more threads than processing cores (think thousands)
- Easy to get to run
- Hard to get to run well
  - Fast memory is small
  - Need to keep the processors busy

Threads

- Threads are grouped
  - Grid of some dimension (gridDim.x gridDim.y)
  - Each element of the grid is a block
    - Blocks have dimension (blockDim.x blockDim.y blockDim.z)
    - A thread thus has a unique five tuple
      - blockIdx.x, blockIdx.y, threadIdx.x, threadIdx.y, threadIdx.z
- Threads only operate on data passed to the GPU

Where to run?

- Cuda is supported on systems with newer Nvidia graphics cards
- Examples:
  - Current generation of Apple Mac laptops
  - Mac Pro with NVIDIA GeForce GT 120 or aftermarket GTX 285
- Can run in emulation mode on machines that don't have a NVIDIA card

Mio node n48

- Purchased using TechFee money
  - Available to students
  - Professors need to ask their students
- Front end
  - Dell 1950 node
  - 24 Gbytes of Ram
- PCI 8x connection to Tesla S1070

N48 - Tesla S1070

# of Tesla GPUs	4
# of Streaming Processor Cores	960 (240 per processor)
Frequency of processor cores	1.296 to 1.44 GHz
Single Precision floating point performance (peak)	3.73 to 4.14 TFlops
Double Precision floating point performance (peak)	311 to 345 GFlops
Floating Point Precision	IEEE 754 single & double
Total Dedicated Memory	16 GB
Memory Interface	512-bit
Memory Bandwidth	408 GB/sec
Max Power Consumption	800 W
System Interface	PCIe x16 or x8
Software Development Tools	C-based CUDA Toolkit

http://www.nvidia.com/object/product_tesla_s1070_us.html

deviceQuery - example

```
[tkaiser@cuda1 release]$ ./deviceQuery
CUDA Device Query (Runtime API) version (CUDART static linking)
There are 4 devices supporting CUDA

Device 0: "Tesla C1060"
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         3
  Total amount of global memory:                 4294705152 bytes
  Number of multiprocessors:                     30
  Number of cores:                               240
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 16384
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.44 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     No
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host threads can
      use this device simultaneously)
```

Mio node n53

- Purchased using grant money
- HP machine designed specifically for GPUs
  - SL390
  - 12 Intel cores
  - 48 Gbytes of Ram
  - Three M2070 cards
  - Room for 3 more nodes = 12 more

deviceQuery - example

```
There are 3 devices supporting CUDA

Device 0: "Tesla M2070"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5636554752 bytes
  Multiprocessors x Cores/MP = Cores:            14 (MP) x 32 (Cores/MP) = 448 (Cores)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Clock rate:                                    1.15 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     No
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host threads can use this
device simultaneously)
  Concurrent kernel execution:                   Yes
  Device has ECC support enabled:                Yes
  Device is using TCC driver mode:               No
```

Running on Mio GPU nodes

http://inside.mines.edu/mio/page6.html

```
cd $DATA
mkdir guide
cd guide
wget -o wget.out http://inside.mines.edu/mio/source/cuda.tgz
tar -xzf cuda.tgz
cat pgi
vi pgi    sub 11.5 for 11.3
source pgi
cp ~/.bashrc ~/.bashrc-old-pg
cat pgi >> ~/.bashrc
make
qsub cudarun
```

nvcc “cuda” Compiler

```
[tkaiser@cuda1 release]$ nvcc --help

Usage  : nvcc [options] <inputfile>

Options for specifying the compilation phase
============================================
More exactly, this option specifies up to which stage the input files must be compiled,
according to the following compilation trajectories for different input file types:
        .c/.cc/.cpp/.cxx : preprocess, compile, link
        .i/.ii           : compile, link
        .cu              : preprocess, cuda frontend, ptxassemble,
                           merge with host C code, compile, link
        .gpu             : nvopencc compile into cubin
        .ptx             : ptxassemble into cubin.
...
...
...


--device-emulation                                  (-deviceemu)
        Generate code for the GPGPU emulation library.
```

“hello world” example

- Cuda Kernel routines can not print unless compiled in emulation mode
- Reads in dimension of the grid (gx,gy)
- Reads in dimension of block within grid(bx,by,bz)
- Number of threads=gx*gy*bx*by*bz
- Allocates a local and device array nthreads*6
- Calls kernel

Kernel Uses:

- gridDim.x gridDim.y
  - Size of grid
- blockIdx.x blockIdx.y
  - Which block for thread
- blockDim.x blockDim.y blockDim.z
  - Size of each block
- threadIdx.x threadIdx.y threadIdx.z
  - Thread within block

Calculates a “natural” thread id and stuffs everything in an array

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
void checkCUDAError(const char *msg);
 __global__ void Kernel(int *dat);
main() {
    int *dat_local, *dat_remote;
    int gx,gy;
    int bx,by,bz;
    int size;
    int numthreads,j;

    scanf("%d %d",&gx,&gy);
    scanf("%d %d %d",&bx,&by,&bz);

    dim3 dimGrid(gx,gy);
    dim3 dimBlock(bx,by,bz);

    numthreads=gx*gy*bx*by*bz;

    size=6*sizeof(int)*numthreads;
    cudaMalloc((void**) &dat_remote, size);
       checkCUDAError("cudaMalloc");
    dat_local=(int*)malloc(size);

    Kernel<<<dimGrid,dimBlock>>>(dat_remote);

    checkCUDAError("Kernel");
...
...
```

```
 __global__ void Kernel(int *dat) {
/* get my block within a grid */
    int myblock=blockIdx.x+blockIdx.y*gridDim.x;
/* how big is each block within a grid */
    int blocksize=blockDim.x*blockDim.y*blockDim.z;
/* get thread within a block */
    int subthread=threadIdx.z*(blockDim.x*blockDim.y)+threadIdx.y*blockDim.x+threadIdx.x;
/* find my thread */
    int thread=myblock*blocksize+subthread;
#if __DEVICE_EMULATION__
    printf("gridDim=(%3d %3d) blockIdx=(%3d %3d)    blockDim=(%3d %3d %3d)  threadIdx=(%3d
%3d %3d)  %6d\n",
      gridDim.x,gridDim.y,
      blockIdx.x,blockIdx.y,
      blockDim.x,blockDim.y,blockDim.z,
      threadIdx.x,threadIdx.y,threadIdx.z,thread);
#endif
/* starting index into array */
    int index=thread*6;
    dat[index]=thread;
    dat[index+1]=blockIdx.x;
    dat[index+2]=blockIdx.y;
    dat[index+3]=threadIdx.x;
    dat[index+4]=threadIdx.y;
    dat[index+5]=threadIdx.z;
}
```

```
...
...
    cudaMemcpy(dat_local, dat_remote, size,cudaMemcpyDeviceToHost);
    for(int i=0;i<numthreads;i++) {
        j=i*6;
        printf("%6d      %3d %3d      %3d %3d %3d\n",
        dat_local[j],
        dat_local[j+1],dat_local[j+2],
        dat_local[j+3],dat_local[j+4],dat_local[j+5]);
    }

}
```

```
void checkCUDAError(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if( cudaSuccess != err)
    {
        fprintf(stderr, "Cuda error: %s: %s.\n", msg,
                                    cudaGetErrorString( err) );
        exit(EXIT_FAILURE);
    }
}
```

```
void checkCUDAError(const char *msg)
{
    cudaError_t err = cudaGetLastError();
    if( cudaSuccess != err)
    {
        fprintf(stderr, "Cuda error: %s: %s.\n", msg,
                                  cudaGetErrorString( err) );
        exit(EXIT_FAILURE);
    }
}
```

```
[peloton:~/wave3d] tkaiser% nvcc testinput.cu  -o testinput
[peloton:~/wave3d] tkaiser% ./testinput
2 3
2 3 4
      0          0    0          0    0    0
      1          0    0          1    0    0
      2          0    0          0    1    0
      3          0    0          1    1    0
      4          0    0          0    2    0
      5          0    0          1    2    0
      6          0    0          0    0    1
      7          0    0          1    0    1
      8          0    0          0    1    1
...
...
    134          1    2          0    1    2
    135          1    2          1    1    2
    136          1    2          0    2    2
    137          1    2          1    2    2
    138          1    2          0    0    3
    139          1    2          1    0    3
    140          1    2          0    1    3
    141          1    2          1    1    3
    142          1    2          0    2    3
    143          1    2          1    2    3
[peloton:~/wave3d] tkaiser%
```

Size of grid

Size of each block within each element of the grid

CULA Linear Algebra Library Installed

- CULA is a GPU-accelerated linear algebra library that utilizes the NVIDIA CUDA parallel computing architecture to dramatically improve the computation speed of sophisticated mathematics.
- http://www.culatools.com/
- We have the free version which has a subset of the whole package

Local Example

- http://geco.mines.edu/tesla/culaexample/index.shtml
- "Scattering by a Dielectric Cylinder of Arbitrary Cross Section Shape." (Radar Cross Section)
- Does a linear solve of a complex matrix using the routine cula_cgesv
- We compare this to the MKL routine cgesv

CULA 13X faster