# Making Your Research Go Faster: Advanced HPCC CI-Days December 16, 2014

https://wiki.hpcc.msu.edu/x/JwJiAQ

Dirk Colbry

colbrydi@msu.edu

Director, High Performance Computing Center

Institute for Cyber-Enabled Research





# Agenda

- Overview
- Advanced System Description
- Powertools
- Doing more faster
  - Pleasantly Parallel, Shared Memory, Shared Network,
     Accelerators, Standard Libraries
- Tricks and tips





# Assumptions

- You have logged in and used the HPCC or similar system
- You are familiar the the Linux command line
- You have some programming / scripting experience
- You are here to learn how to leverage HPCC resources better





# How this workshop works

- I think you work best from doing. So we will do a lot of hands on examples.
- When you get tired of listening to me talk, skip ahead to an exercise and give it a try.
- Exercises are denoted by the following icon in your notes:





# Red and Green Flags

- Use the provided sticky notes to communicate without raised hands:
  - -NO Sticky = I am working
  - -Green = I am done and ready to move on
  - -Red = I am stuck and need more time and/or I could use some help





# Submission Scripts

- Design Goals
  - One script does everything
  - Easy to read
  - Easily given to others
  - Easily moved to different directories





# Agenda

- Overview
- Advanced System Description
- Powertools
- Doing more faster
  - Pleasantly Parallel, Shared Memory, Shared Network,
     Accelerators, Standard Libraries
- Tricks and tips



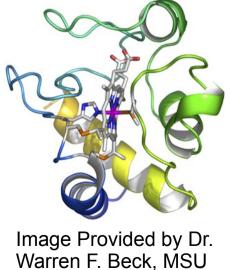


## What problems are we solving?

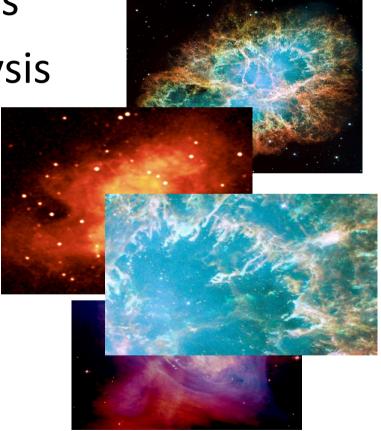
Simulations

Data Analysis

Search







Images from, "Understanding the H<sub>2</sub> Emission from the Crab Nebula", C.T. Richardson, J.A. Baldwin, G.J. Ferland, E.D. Loh, Charles A. Huehn, A.C. Fabian, P.Salomé

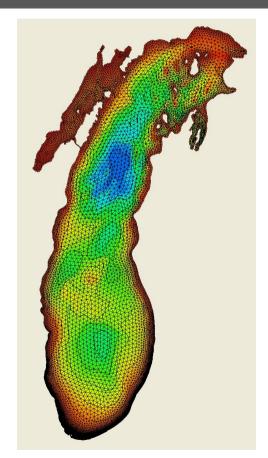
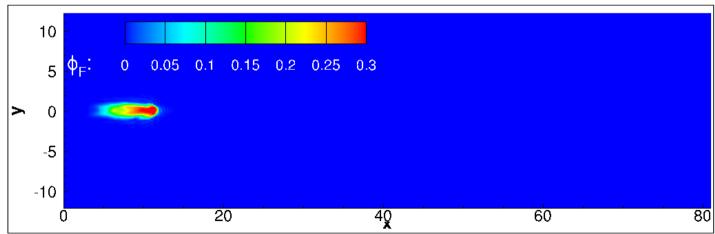


Image Provided by Dr. Mantha Phanikumar. MSU



## Simulations

- Typically System of PDE (Partial Differential equations)
  - Fluid dynamics
  - Finite element analysis
  - Molecular dynamics
  - Weather
  - Etc.
- Mathematically equivalent to inverse of a matrix

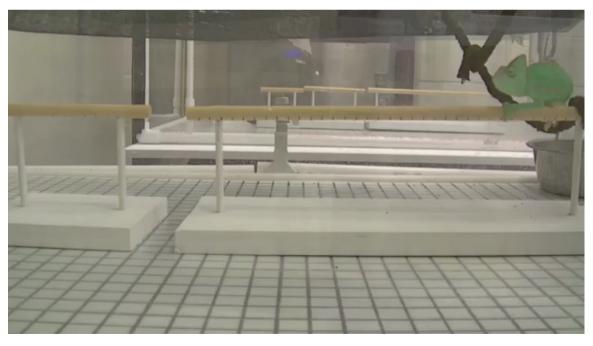


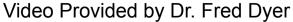




# Data Analysis

- Computer vision tasks
- Some Bioinformatics
- Astrophysics
- Etc.



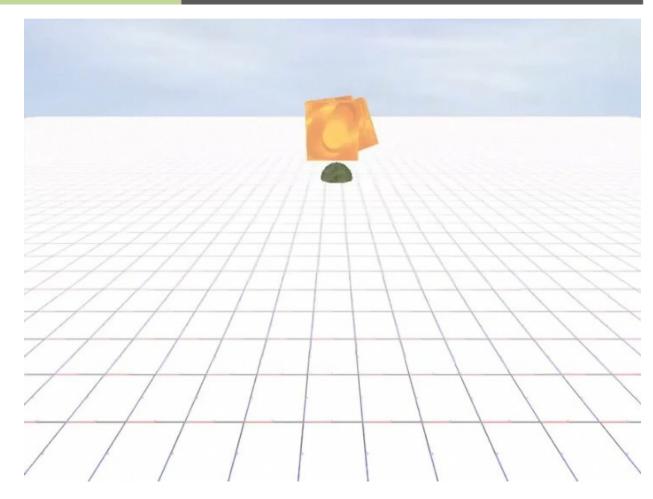


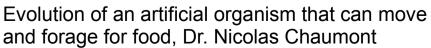




## Search

- Genome sequencing
- Analytics
- Optimization
- Etc.









## **HPC Systems**

- Large Memory Nodes (up to 6TB!`)
- GPU Accelerated cluster (K20, M1060)
- PHI Accelerated cluster (5110p)
- Over 600 nodes, 7000 computing cores
- Access to high throughput condor cluster
- 363TB high speed parallel scratch file space
- 50GB replicated file spaces
- Access to large open-source software stack and specialized bioinformatics VMs







## Free Access to software

- Compiled open-source software stack
  - Close to 2000 titles!
- Optimized Math/Communications libraries
- Some commercial software available
  - E.g. Ansys, MATLAB (+many toolboxes), Stata,Gauss, SAS







Full list: http://wiki.hpcc.msu.edu

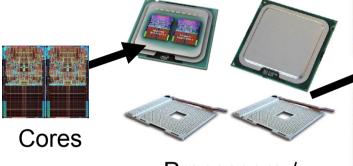
# General Purpose Clusters







Commodity Cluster



Processors / Sockets





Chassis





Rack



# **Buy-In Opportunities**

- We will maintain your computers for you
- Researchers get exclusive use of their nodes within 4 hours of submitting a job
- Buy-in jobs will automatically overflow into the general resources.





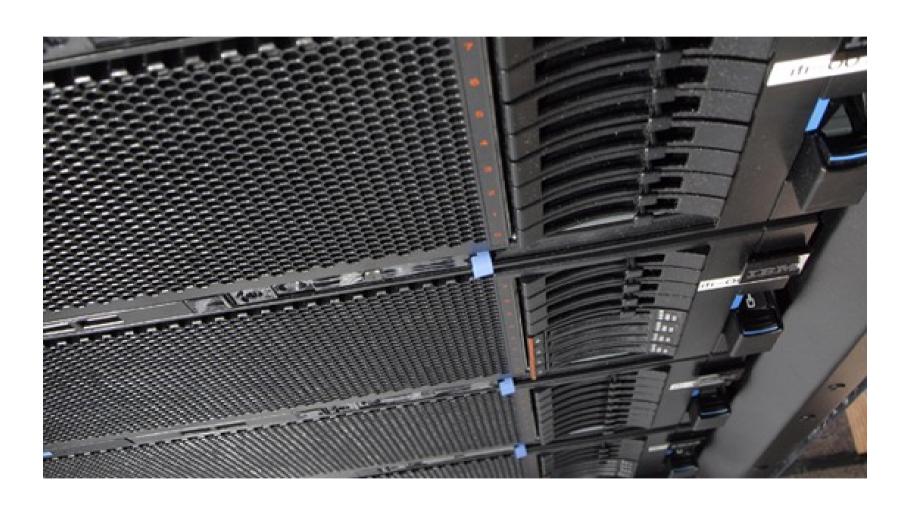
#### Current Buy-In options (2014)

- 20 cores, 64 Gb, \$3,806\*
- 20 cores, 256 Gb, \$5339\*
- 20 cores, 128 Gb, 2 Nvidia K20, \$7899\*
- 20 cores, 128 Gb, 2 Intel 5115P, \$9043\*
- 48 cores, 1 Tb, \$29,979
- 48 cores, 1.5 Tb, \$34,989
- 48 cores, 3 Tb, \$60,995
- 96 cores, 6 Tb, \$142,772
- Replicated storage: \$175/TB per year





## Large Shared Memory Systems (Fat Nodes)



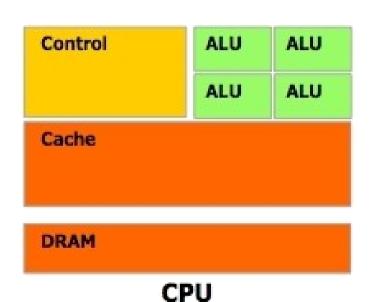


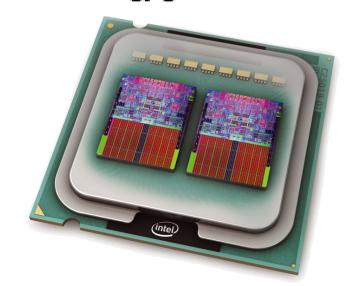


## Shared Memory Communication

- Fast!
- Cores on a system share the same memory
- OpenMP
- Fat nodes
  - -96 cores
  - 6TB of memory









# Accelerated Systems





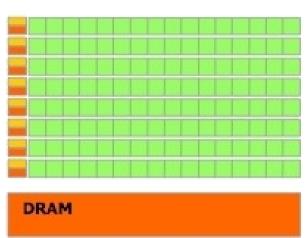


## **GPU**





- Cards used to render graphics on a computer
- Hundreds of cores
- Not very smart cores
- But, if you can make your research look like graphics rendering you may be able to run really fast!







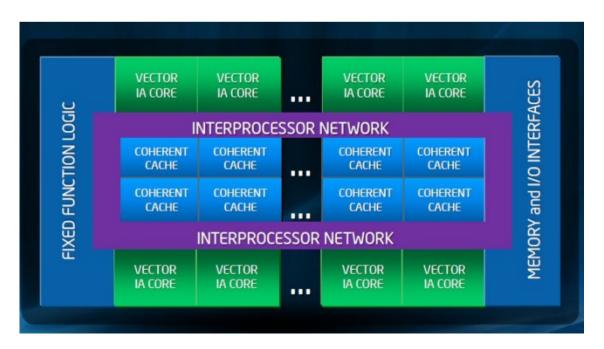
## Intel Xeon Phi

Cross between CPU and GPU

About 61 Pentium III cores

Less cores/slower than GPU

- Easier to use than GP







## High Throughput HTCondor Cluster







## MSU HTCondor Cluster

- Runs like a screen saver and Scavenges CPU cycles:
  - Approximately 400+ nodes
  - Approximately 7000 cores
  - Windows 7







# Agenda

- Overview
- Advanced System Description
- Powertools
- Doing more faster
  - Pleasantly Parallel, Shared Memory Parallelization,
     Shared Network, Accelerators, Standard Libraries
- Tricks and tips





# Evaluation Gateway

- Same as regular gateway just a different URL:
  - eval.hpcc.msu.edu
- Also slightly different splash screen.
- Great "back door" if there is a problem on the gateway VM.





## **Evaluation Nodes**

- Intended for Testing and Evaluation of different hardware.
- Just special development nodes.
- No corresponding hardware in the cluster.

Name	Cores	Memory	Accelerators	Notes
eval-amd09	8	8GB	_	
eval-k10	4	8GB	1 Nvidia k10	Nvidia Graphics Node
eval-atom13	8	24GB	_	
eval-k40	20	128GB	1 Nvidia K40	Nvidia Graphics Node









- Step 1: Log into eval.hpcc.msu.edu
- Step 2: ssh into a eval node (evaluation node)
  - > ssh eval-k40
- Step 3: do something

```
Last login: Wed Dec 10 09:40:47 2014 from staff-dhcp-163.staff
             ////_/_/_/_/_/_//_
        Welcome to Michigan State's High Performance Computing Center
                 ** Unauthorized access is prohibited **
   We recommend using dev-intel14 (or nodes with low usage).
   For GPU development please use green nodes.
   For MIC development please use underlined nodes
        Development Nodes (usage)
                                             Filesystem Information
    eval-amd09 (low)
                       eval-atom13 (n/a)
                                               ${HOME} at 85% usage
                       eval-k40 (low)
     eval-k10 (low)
                                              (used ~864G of 1.0T)
                Cluster Load (utilization)
        short jobs (< 4 hrs) ( 51%)
                                         general jobs (93%)
                                    accelerated nodes ( 19%)
        nodes with > 256 GB ( 83%)
```



## What are Powertools

- Powertools are scripts and programs to make interfacing with the HPCC simpler
- The tools are written mostly by HPCC staff and users.
- Think of most of these as "Beta" software.









- Load the powertools module file:
  - >module load powertools
- To list the currently available tools type "powertools" after loading the powertools module
  - >powertools





## Common Powertools

- Any developer node shortcut
  - > dev
- Developer node shortcuts (intel07, gfx08, intel09, gfx10, gfx11, intel14)
- Two commands in one:
  - Automatically ssh directly to the developer node
  - Then automatically cd to the current directory from the previous node





## More Common Powertools

- powertools list powertools and common commands not standard on linux systems
- **sj** show jobs in the queue for the current user
- starttime show estimated start times for a job
- mailme E-mail yourself a file
- clusterstate show a summary of the current state of the nodes in the cluster





## **Even More Powertools**

- getexample provides a copy of examples for various tasks written by iCER staff
- quota list your home directory disk usage
- priority\_status Shows the status of an individuals buy-in nodes.
- poweruser Set up your account to load powertools by default





#### How to turn on powertools as default?



Edit your .bashrc

> nano ~/.bashrc

 add the following line: module load powertools You can also just use the "poweruser" powertool

 Modifying your bashrc is required if you want to use the developer node shortcuts and hop between different nodes without constantly typing "module load powertools





# Agenda

- Overview
- Advanced System Description
- Powertools
- Doing more faster
  - Pleasantly Parallel, Shared Memory Parallelization,
     Shared Network, Accelerators, Standard Libraries
- Tricks and tips





## What is the Bottleneck

- Not enough Memory
  - Solution: use a bigger node (6tb 96 cores)
- Slow File I/O
  - Solution: use scratch
  - Solution: use a ram disk
- Too many calculations
  - Solution: run your code in parallel





### Steps to parallel code

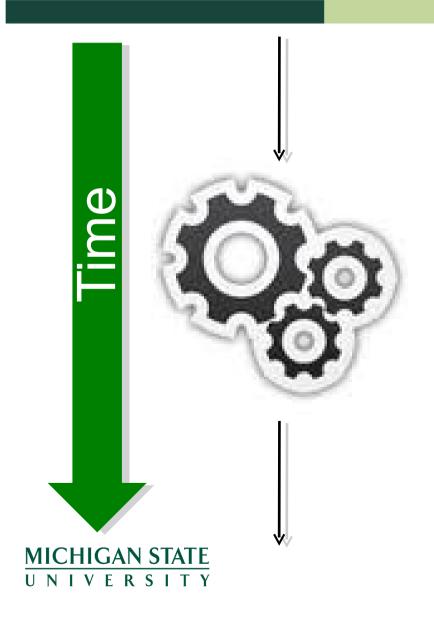
Note: Every application is different

- 1. Analyze your code
  - Profilers (gprof, vtune, map, perfreport, tau)
  - Debuggers / memory trackers (gdb, ddt, totalview)
- 1. Optimize calculations
  - Trade memory for time (i.e., never do the same calculation twice)
- 1. Find ways to parallelize
  - Look for loops
  - Find iterations independent from each other
  - Determine how much information needs to be transferred





### Single Thread Jobs

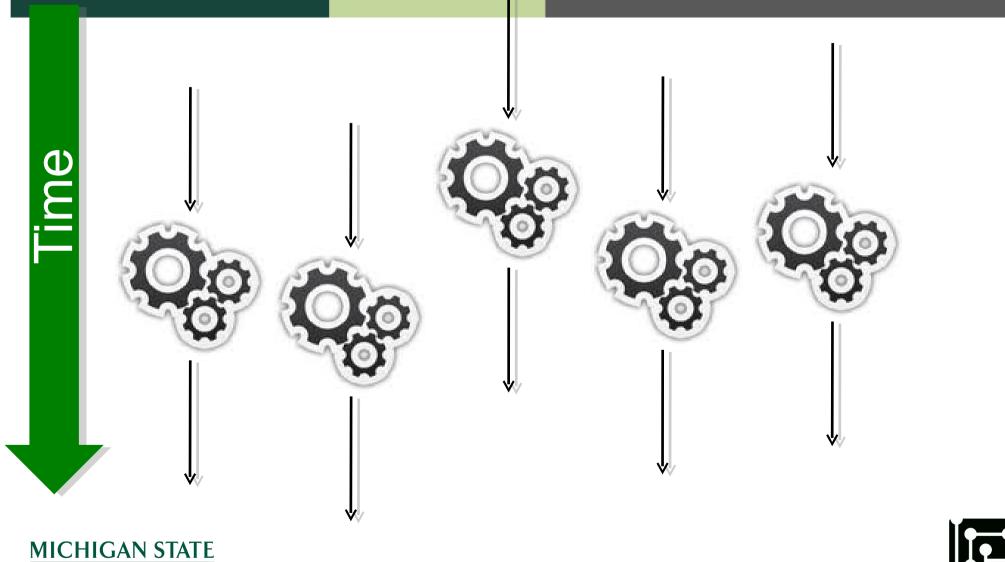


One CPU can only run one thing at a time. (sort of)



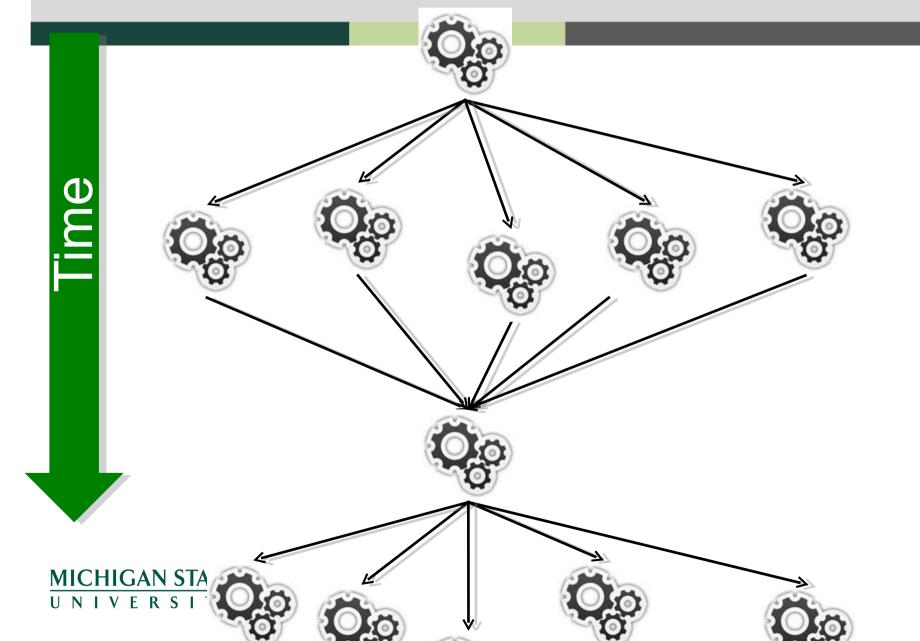
## Pleasantly Parallel

UNIVERSITY



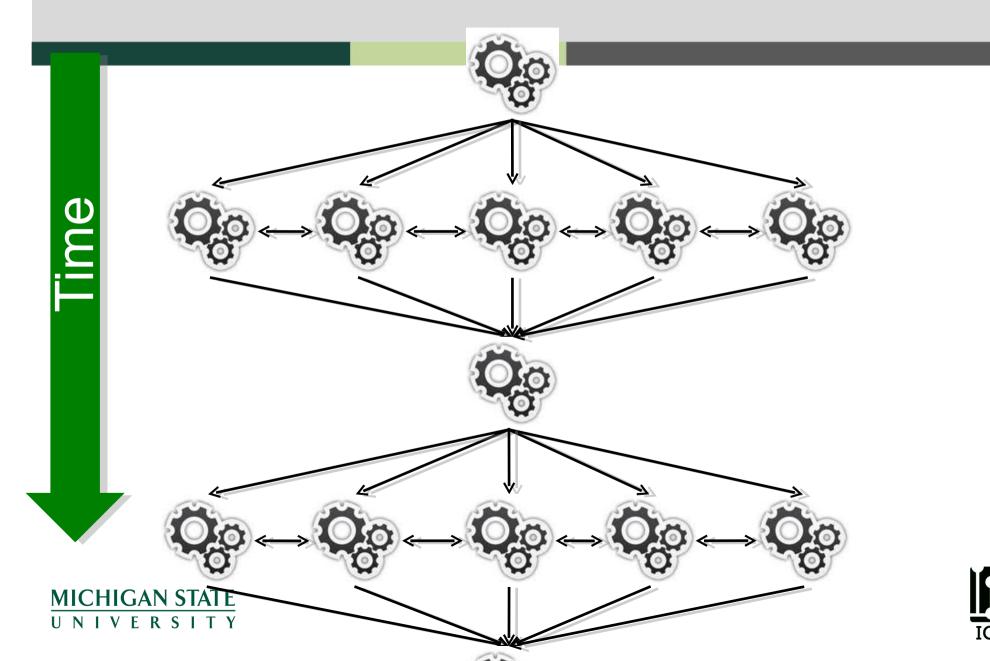


# Loosely Coupled





# Tightly Coupled



#### Communication

- Shared Memory
- Shared Network
- Distributed Network
- Dedicated Accelerators
- Hybrid Systems







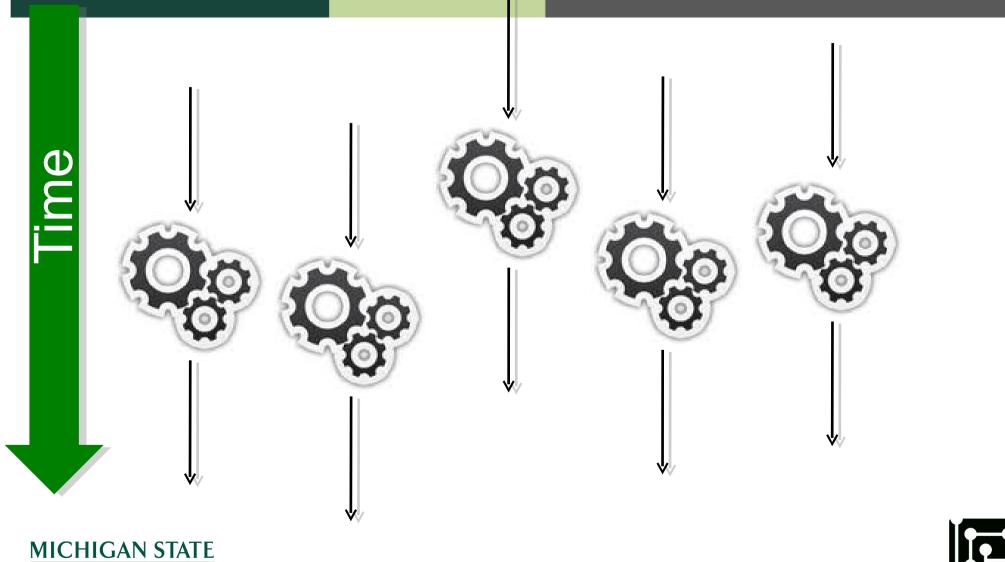
# Pleasantly Parallel





## Pleasantly Parallel

UNIVERSITY





### How fast can we go?

- T How long does each operation take?
- N How many operations do you need to run?
- CPUs Number of Cores job will run on.
- Single CPU time estimate:
  - -TxN
- Best possible Pleasantly parallel time:
  - (TxN)\*overhead/CPUs

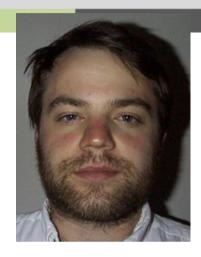


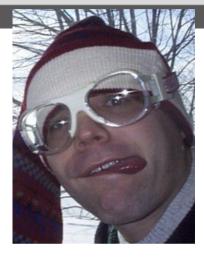


## Who are you? -- Biometrics













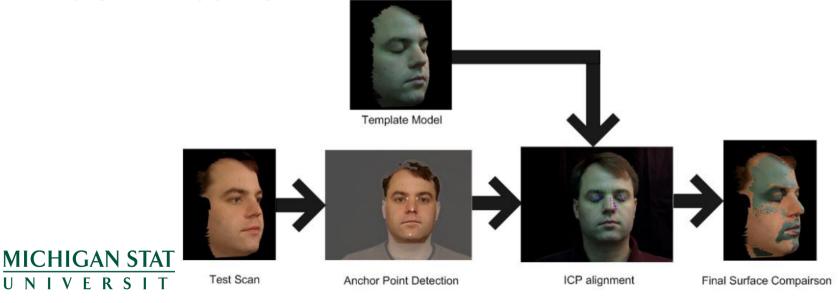






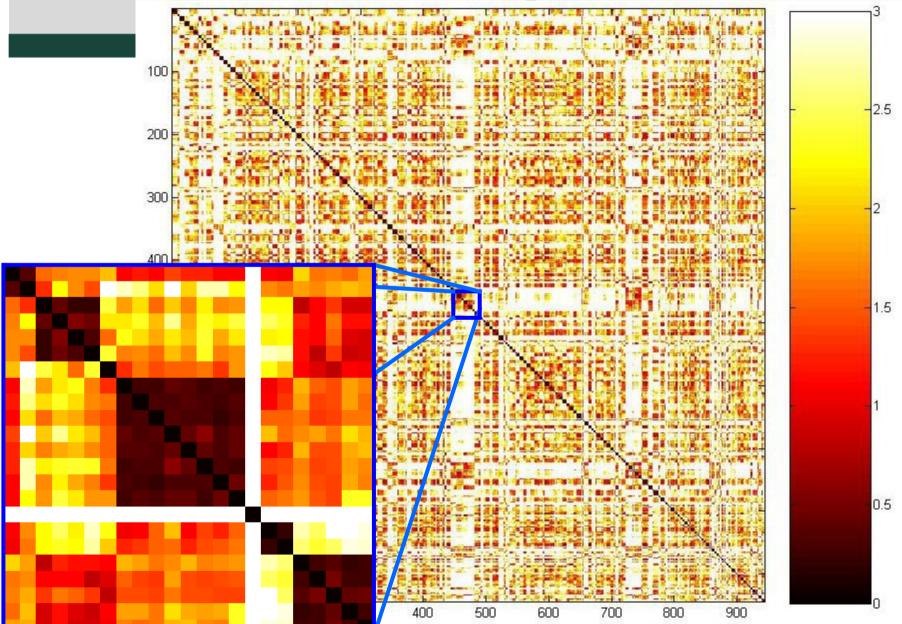
#### Pairwise-All Problem

- Database of faces
- Compare everything to everything else
- Calculate a Matching score to use for identification





#### 943 x 943 Similarity Matrix





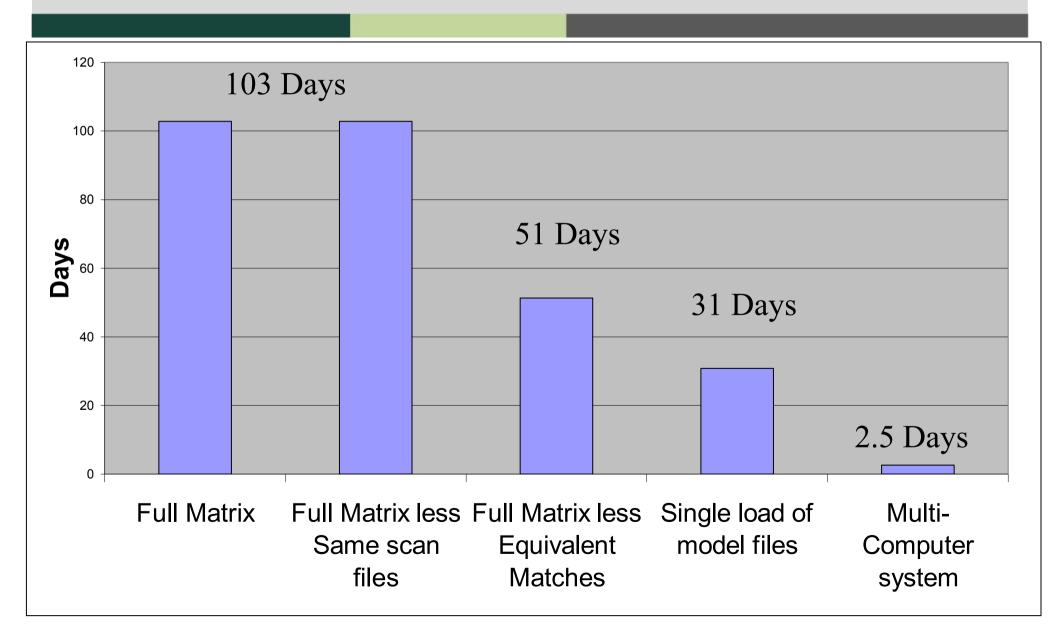
#### **Estimated Calculation Times**

- Preprocessing
  - 943 \* 12 (seconds) 189 Minutes
- Matching
  - 943 \* 943 \* 5 (seconds) 103 Days
- Scans matched to themselves always result in 0 mm
  - (943 \* 943 943) \* 5 (seconds) 103 Days
- The Proposed Alignment Algorithm is symmetric.
- We also load models once per row instead of every time





#### Calculation Time for Full Similarity Matrix



#### How do we go even bigger?

- 5000 scans.
  - 1.5 years on a single processor computer
  - 13 days on our ad-hoc cluster.
  - 1.5 days a commodity cluster at MSU





## Steps to Pleasantly Parallel

- Figure out command line
- Estimate single job time:
  - Should be > 5 minutes
  - Should be < 1 week</p>
  - Best if < 4 hours</p>
- Make a submissions script
- Submit Job





## Pleasantly Parallel Example

• Folder full of input files:

1.in	5.in	9.in	13.in	17.in
2.in	6.in	10.in	14.in	18.in
3.in	7.in	11.in	15.in	19.in
4.in	8.in	12.in	16.in	

Want folder full of output files:

1.out	5.out	9.out	13.out	17.out
2.out	6.out	10.out	14.out	18.out
3.out	7.out	11.out	15.out	19.out
4.out	8.out	12.out	16.out	

Command Syntax:



./myprogram inputfile > outputfile



### PBS Job Arrays

- One submission script copied many times
- Uses the PBS –t option
  - Ranges: 1-10
  - Lists: 2,4,100,3
  - Combination: 1-10,20,50,100
- Distinguish between jobs by using the PBS\_ARRAYID environment variable







## Simple Job Array

```
#!/bin/bash -login
#PBS -1 walltime=00:05:00,mem=2gb
#PBS -1 nodes=1:ppn=1,feature=gbe
#PBS -t 1-19
cd ${PBS O WORKDIR}
mkdir ${PBS ARRAYID}
Cd ${PBS ARRAYID}
../myprogram ../${PBS ARRAYID}.in > ${PBS ARRAYID}.out
qstat -f ${PBS JOBID}
```





- Get the bleder\_farm example:
  - > getexample
  - >getexample blender farm
  - >cd ./blender farm
- Look at the qusb file, using "less" command
  - >less blender\_farm.qsub
- Submit the job
  - >qsub blender farm.qsub





#### HPCC Job array limitations

- Can not have more than 520 cores running at once
- Can not submit more than 1000 jobs at once
- Each job can not run longer than one week

Lots of ways to work around these limitations





### Job array numbers

- All numbers in a job array have the same base number
  - -7478210
- Each PBS\_ARRAYID is show in square brackets
  - -7478210[1]
  - -7478210[2]
- Delete all jobs using one command





#### Unrolling Loops

- Your program has independent loops
  - Each iteration of the loop does not depend on the other iterations
  - Loop can be executed in any order
  - 5 Minutes < Iteration Time < 1 week</li>
  - Output of each iteration must be easy to save and recombine for next step of workflow
- Rewrite your program to accept an iteration number as an input
  - ./myprogram IterationNumber

UNIVERSITY

Rewrite your program to save output and use an additional program for post processing





## Simple Unrolled Loop

```
#!/bin/bash -login
#PBS -1 walltime=00:05:00
#PBS -l nodes=1:ppn=1,feature=gbe
#PBS -t 1-100
cd ${PBS O WORKID}
./myprogram ${PBS ARRAYID}
qstat -f ${PBS JOBID}
```

#### Task Queue

- A list of tasks (also called treatments, inputs,
   ...) that distinguish what needs to be done.
- Each pleasantly parallel process (worker) checks the list and picks work not yet completed.
- The trick is to not have two workers do the same task.





#### List of Commands



Commands.txt

```
./myprogram -a 100 -z 3023
./myprogram dosomething different
./myprogram
./myprogram -s 100
./myprogram -s 200
./myprogram -s 300
./myprogram -w 400
./myotherporgram
./mythirdprogram
```









```
#!/bin/bash -login
#PBS -1 walltime=00:05:00
#PBS -1 nodes=1:ppn=1,feature=gbe
#PBS -t 1-100
cd ${PBS O WORKID}
cmd=`tail -n ${PBS ARRAYID} commands.txt | head -n 1`
echo ${cmd}
${cmd}
qstat -f ${PBS_JOBID}
```

### Files as Semaphores (FAS)

- Use a list of input files as your task list
- Use a list of output files (or flag files) as your in-progress/complete list
- Rely on the file system to ensure that no two jobs are selected at the same time (not a great assumption but it works)

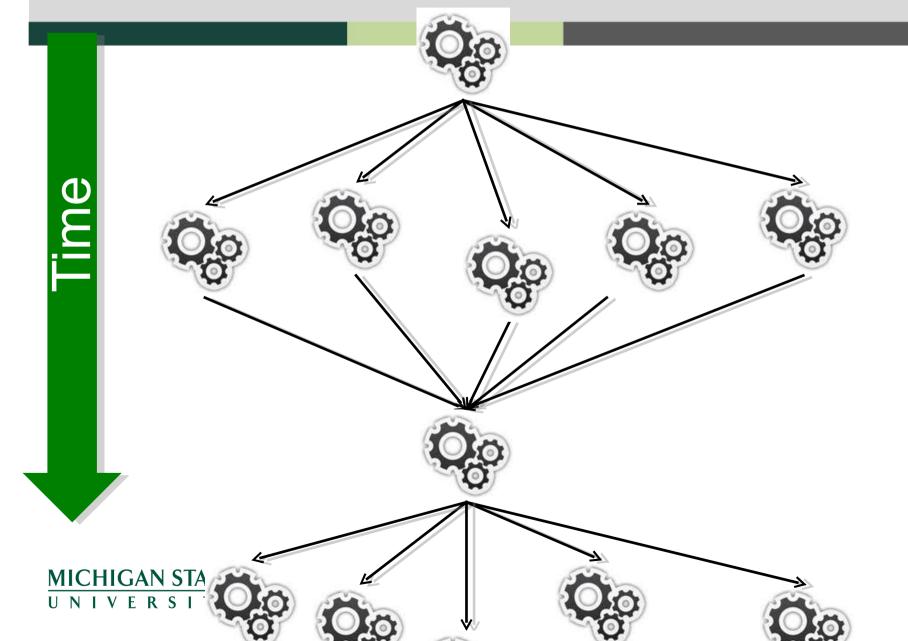




#### Simple FAS

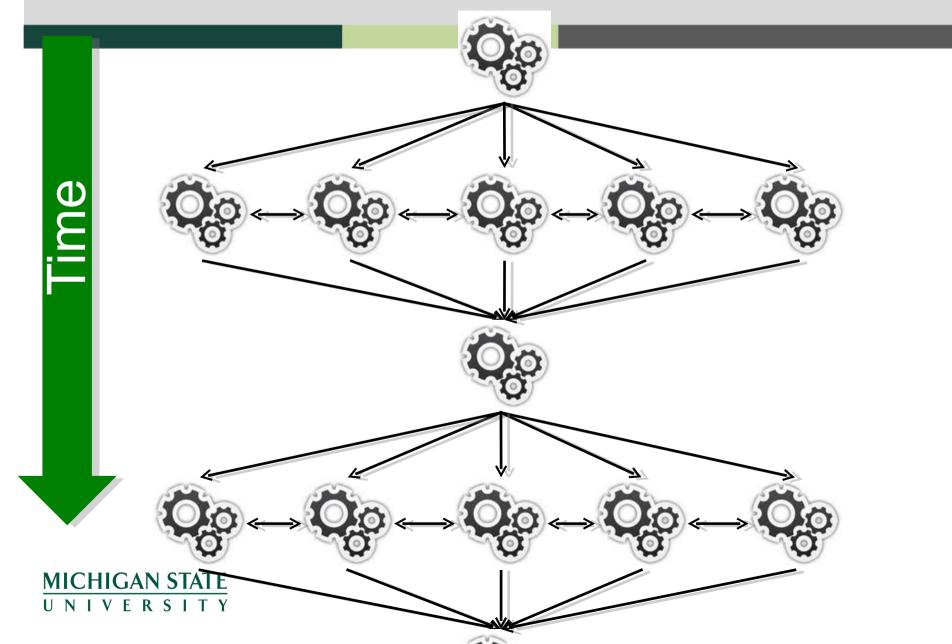
```
#!/bin/bash -login
#PBS -1 walltime=00:05:00
#PBS -1 nodes=1:ppn=1,feature=gbe
#PBS -t 1-100
cd ${PBS O WORKID}
sleep $(( ${RANDOM} % 100 ))
for file in *.in; do
  output="./${file%.*}.out"
  if [ ! -f ${output} ]; then
    touch ${output}
    ./myprogram ${file} > ${output}
    qsub -t 0 -N ${PBS_JOBNAME} ${0}
    exit 0
  fi
done
```

# Loosely Coupled





# Tightly Coupled





## Shared Memory Parallelization





## Shared Memory

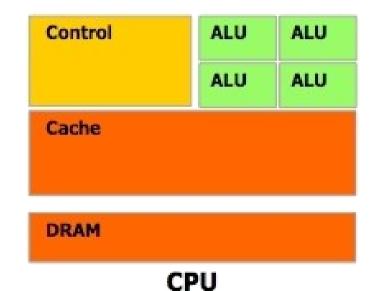
- Different threads (cores, processes)
   communicate though pointers to the same
   memory location
- Problems can occur if different threads write the same memory at the same time
- Flags (also called locks and/or semaphores)
  are used to allow only one thread to access
  memory at the same time

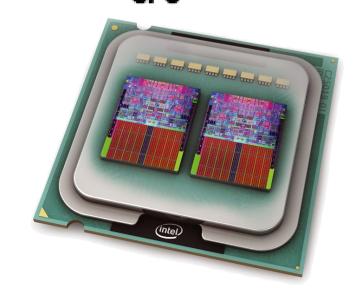




#### Shared Memory Communication

- Cores on a processor share the same memory
- OpenMP
- Fat nodes
  - -96 cores
  - 6TB of memory







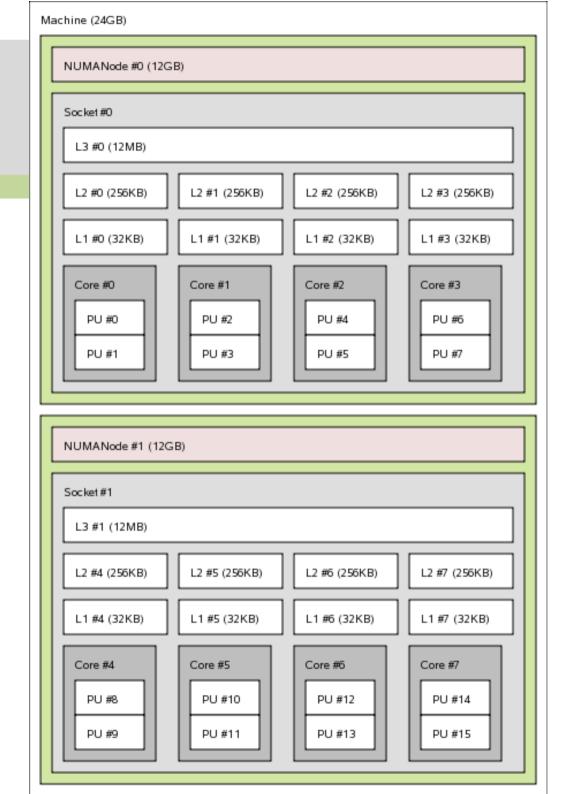
MICHIGAN STATE

#### Intel10

- 2 sockets
- 8 cores
- 24 GB memory

Figure generated using hwloc





#### Intel14 – basic node

2 sockets, 20 cores, 64 GB memory

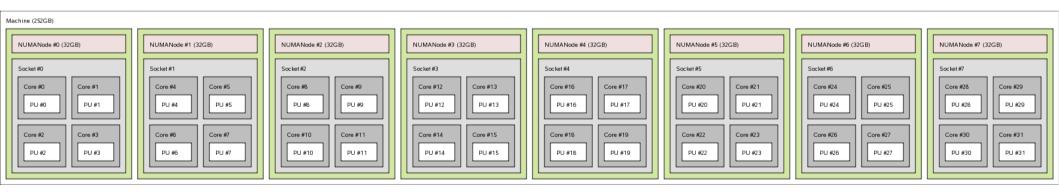






### Large Memory Example

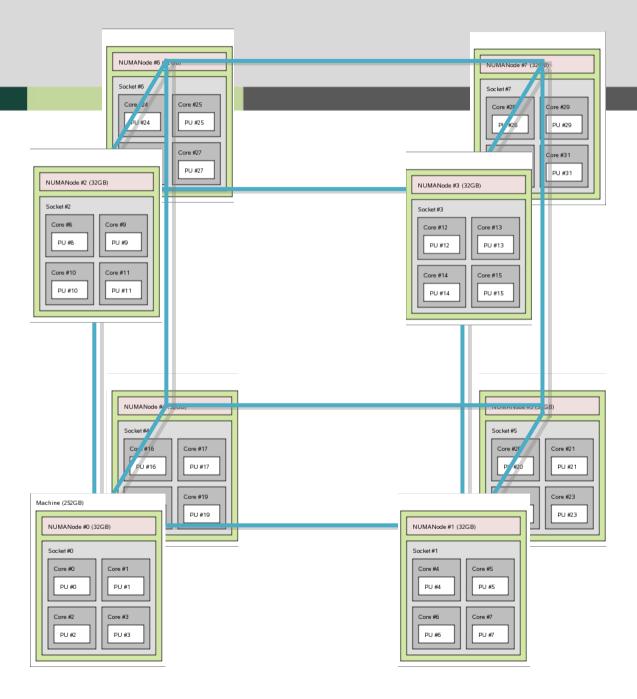
- eval-amd09
- 8 sockets, 32 cores, 256 GB RAM







#### **NUMA**







#### Intel14-xl

- 8 sockets
- 96 cores
- 6TB of RAM

Figure generated using hwloc







#### Shared memory submission scripts

- Typically one node with multiple processors per node (ppn)
  - #PBS -I nodes=1:ppn=8
- Different programs use different methods to tell them how many processors to use
  - Command line arguments
  - Environment variables





# Example: shared memory Script

- Bowtie uses shared memory parallelization
- Get the bowtie example
  - >getexample bowtie



```
>cd ./bowtie
```

- Look at the submission script
  - >less ./bowtie.qsub
- Run the job

```
>qsub bowtie.qsub
```





#### OpenMP

- Common Shared Memory parallelizaiton
- Single program runs in many threads
- Really easy to pick loops that are parallel and split them into multi threads
- Minor modifications to code that can be written not to affect single





# OpenMP is easy

```
#include <omp.h>
 #pragma omp parallel for
 for (i=0;i<100;++i) {
   A(i) = A(i) + B
```

#### Compile OpenMP Jobs

- Use compiler option fopenmp.
  - -fopenmp
- Example:

gcc –fopenmp mycode.cc –o mycode





# simpleOMP.qsub example

```
#!/bin/bash -login
#PBS -1 walltime=00:01:00
#PBS -l nodes=1:ppn=5, feature=qbe
cd ${PBS O WORKDIR}
export OMP NUM THREADS=${PBS NUM PPN}
./simpleOMP
qstat -f ${PBS JOBID}
```

### Try another getexample

getexample helloOpenMP getexample OpenMP\_profiling





# Shared Network Parallelization





#### MPI on HPCC

- Two Flavors of MPI
- Switching flavors and compiling
- Running in a script
- Running on the developer nodes





# MPI program (1 of 4)

```
/* Needed for printf'ing */
#include <stdio.h>
#include <stdlib.h>
/* Get the MPI header file */
#include <mpi.h>
/* Max number of nodes to test */
#define max nodes 264
/* Largest hostname string hostnames */
#define str length 50
```

# MPI program (2 of 4)

```
int main(int argc, char **argv)
   /* Declare variables */
   int proc, rank, size, namelen;
   int ids[max nodes];
   char hostname[str length][max nodes];
   char p name[str length];
  MPI Status status;
  MPI Init(&argc, &argv);
  MPI Comm rank(MPI COMM WORLD, &rank);
  MPI Comm size (MPI COMM WORLD, &size);
  MPI Get processor name (p name, &namelen);
```

# MPI program (3 of 4)

```
if (rank==0) {
   printf("Hello From: %s I am the receiving processor
%d of %d\n",p name, rank+1, size);
   for (proc=1;proc<size;proc++) {</pre>
      MPI Recv(&hostname[0][proc], \\
               str length,MPI INT,proc, \\
               1,MPI COMM WORLD, &status);
      MPI Recv(&ids[proc], \\
               str length,MPI INT,proc, \\
               2,MPI COMM WORLD, &status);
      printf("Hello From: %-20s I am processor %d of
%d\n",&hostname[0][proc], ids[proc]+1, size);
```

# MPI program (4 of 4)

```
else { // NOT Rank 0
    srand(rank);
    int t = rand() %10+1;
    sleep(t);
    MPI Send(&p name, str length, \\
              MPI INT, 0, 1, MPI COMM WORLD);
    MPI Send(&rank,str length, \\
              MPI INT, 0, 2, MPI COMM WORLD);
 MPI Finalize();
 return(0);
```

#### Two Flavors of MPI

- mvapich vs openmpi (default)
- Historically mvapich was much faster that openmpi
- The newest version of openmpi is just as fast as mvapich
- I feel that openmpi is much easier to use, but either will work on HPCC





### Switching Flavors



- Use the "module" command to switch between the two versions of mpi
- Openmpi module is loaded by default
- To switch to myapich you first need to unload openmpi:
  - > module unload OpenMPI
- Then you need to load mvapich:
  - > module load MVAPICH

NIVERSITY

- You can do both commands in one step by using swap:
  - > module swap OpenMPI MVAPICH



#### MPI Submission Scripts

#### openmpi

```
#!/bin/bash -login
#PBS -l nodes=10:ppn=1
cd ${PBS_O_WORKDIR}
mpirun program_name>
```

#### mvapich

```
#!/bin/bash -login
#PBS -l nodes=10:ppn=1
cd ${PBS_O_WORKDIR}
module swap OpenMPI MVAPICH
mpiexec program_name>
```





# Trying out an example



- 1. Log on to one of the developer nodes
- 2. Load the powertools module:
  - > module load powertools
- Run the getexample program. This will create a folder called helloMPI:
  - > getexample helloMPI
- Change to the helloMPI directory and read the readme files
- 2. Or just type the following on the command line:





# Testing MPI jobs on dev node

- Use mpirun instead of mpiexec
- Need a hostfile
  - > echo \$HOSTNAME >> ./hostfile
  - > echo \$HOSTNAME >> ./hostfile
  - > echo \$HOSTNAME >> ./hostfile
  - > echo \$HOSTNAME >> ./hostfile
- MPIRUN example:
  - > mpirun -np 4 -hostfile ./hostfile helloMPI







#### Running on the Command Line

- The scheduler automatically knows how many and where to run MPI processes.
- However, on the command line, you need to specify the nodes and processors.
- openmpi and mvapich are a little different.





#### Command Line Differences

- Openmpi
  - mpirun
  - Default assumes one process on the current host.
  - You do not even need the **mpirun** command to run the default.
  - Optionally you can use the –n and –hostfile options to change the default

- mvapich
  - mpirun
  - Requires both the –np and –machinefile flag to run.



#### Command line

mvapich

```
mpirun -np 4 -machinefile machinefile program_name>
```

openmpi

```
mpirun -n 4 -hostfile machinefile cprogram_name>
```

 NOTE: I did a check and either MPI implementation will work with either notation.





#### Which MPI command do you use?

	Command Line	Job Script
openmpi	mpirun	mpirun
mvapich	mpirun	mpiexec





#### **Accelerator Cards**



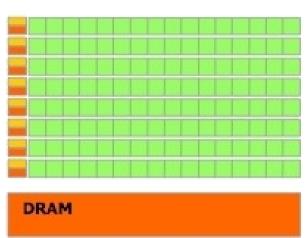


#### **GPU**





- Cards used to render graphics on a computer
- Hundreds of cores
- Not very smart cores
- But, if you can make your research look like graphics rendering you may be able to run really fast!







#### Running on the GPU

- Program Starts on the CPU
  - Copy data to GPU (slow-ish)
  - Run kernel threads on GPU (very fast)
  - Copy results back to CPU (slow-ish)

 There are a lot of clever ways to fully utilize both the GPU and CPU.





#### Pros and Cons

#### Benefits

- Lots of processing cores.
- Works with the CPU as a co-processor
- Very fast local memory bandwidth
- Large online community of developers

#### Drawbacks

- Can be difficult to program.
- Memory Transfers
   between GPU and
   CPU are costly (time).
- Cores typically run the same code.
- Errors are not detected (on older cards)
- Double precision calculations are slow (On older cards)



# CUDA program (1 of 5)

```
#include "cuda.h"
#include <iostream>
using namespace std;
void printGrid(float an array[16][16]) {
  for (int i = 0; i < 16; i++) {
      for (int j = 0; j < 16; j++) {
           cout << an array[i][j];</pre>
      cout << endl;</pre>
```



### CUDA program (2 of 5)





# CUDA program (3 of 5)

```
int main()
  float our array[16][16];
  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
      our array[i][j] = 0;
```





# CUDA program (4 of 5)





# CUDA program (5 of 5)

```
//STEP 3: SET UP
dim3 blockSize(8,8,1);
dim3 gridSize(2,2,1);
//STEP 4: RUN
theKernel<<<gridSize, blockSize>>>(our array d);
//STEP 5: TRANSFER
printGrid(our array);
cudaMemcpy(our_array, our_array_d, size, \\
          cudaMemcpyDeviceToHost);
cout << "----" << endl;
printGrid(our array);
```





#### Compile CUDA Jobs

- Just like MPI, to compile an cuda program you need to use the cuda compiler wrappers:
  - nvcc simple.cu -o simple\_cuda





### Try a cuda getexample

getexample cuda getexample cuda\_clock getexample cuda\_hybrid getexample NAMD\_CUDA\_example





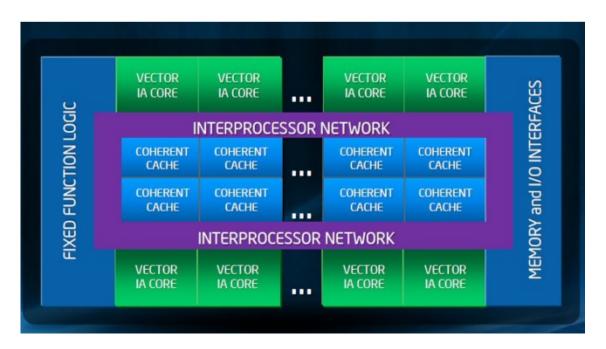
#### Intel Xeon Phi

Cross between CPU and GPU

About 61 Pentium III cores

Less cores/slower than GPU

- Easier to use than GP





- MPI
- OPenMP



# Try a Phi Card example

getexample MIC\_examples getexample MKL\_mic





#### Standard Libraries





#### Standard Libraries

- When possible take advantage of parallel libraries
  - Easy to use
  - Saves time
  - Takes care of the parallel coding for you
  - Tested and vetted by the community





#### Math Kernel Library

- getexample MKL\_benchmark
- getexample MKL\_c\_eigenvalues
- getexample MKL\_Example
- getexample MKL\_mic
- getexample MKL\_parallel





#### Other Libraries

- fftw
- BLAS
- ACML
- BLAS (Basic Linar Algibra)
- Lapak
- trilinos
- petsc
- Magma
- Cudatools
- Mumps





#### Which approach is the best?

- Depends on what you are doing?
- Depends on how much communication you need.
- Depends on what hardware you have.
- Depends on how much time you have.

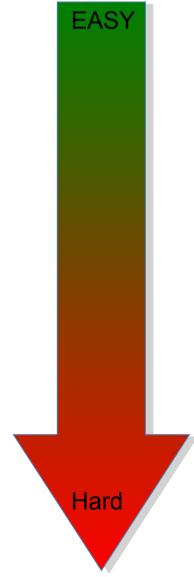




#### My Recommendations

- Pleasantly Parallel
- Standard Libraries
- OpenMP
- OpenACC
- OpenMP on Phi
- MPI
- MPI on Phi?







# Agenda

- Overview
- Advanced System Description
- Powertools
- Doing more faster
  - Pleasantly Parallel, Shared Memory, Shared Network,
     Accelerators, Standard Libraries
- Tricks and tips





# Tips and Tricks Going beyond system Limits





# Going beyond System Limits

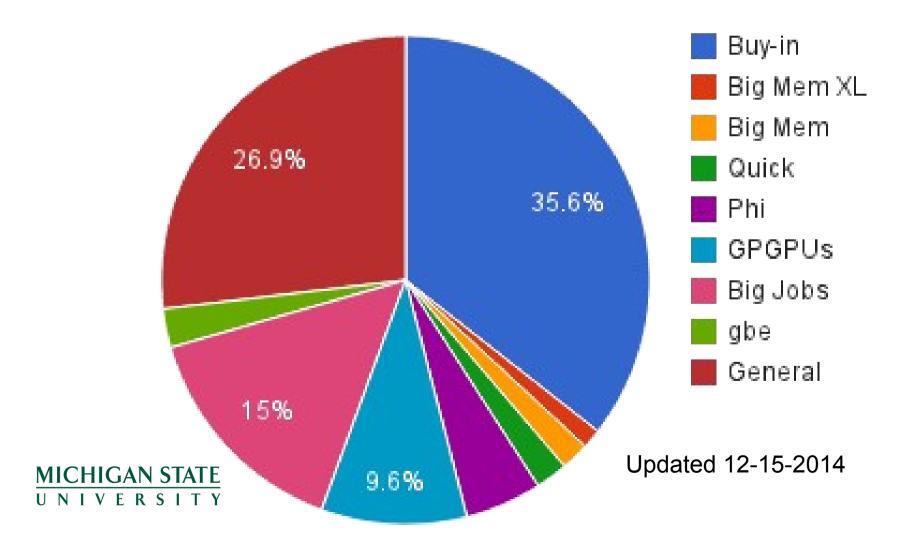
- Going beyond system Limits
  - More than 520 jobs
  - Jobs longer than 1 week
  - Taking advantage of more nodes





# Structure of the system

#### Core Reservations





#### Finding more Nodes

- Owners are guaranteed access to their buy-in node within 4 hours. If they are not using the node, others can use it:
  - #PBS –I walltime=04:00:00
- Some of the nodes do not have Infiniband.
  If you are not using scratch and do not
  need between node communication you
  can access these nodes:
  - #PBS feature=gbe





# Checkpoint / Restart

- What?
  - Save the state of your program
  - Restart your program from the saved point
- Hows
  - Design into your program
  - BLCR (Berkley Lab Checkpoint Restart)
  - Condor Checkpoint Restart
  - Others
- Mhh5
  - Robust jobs
    - As HPC scales ... hardware failures are guaranteed
  - Longer jobs
  - Better science





# Getting Help

- Documentation and User Manual wiki.hpcc.msu.edu
- Contact HPCC and iCER Staff for:
  - Reporting System Problems
  - HPC Program writing/debugging Consultation
  - Help with HPC grant writing
  - System Requests
  - Other General Questions
- Primary form of contact http://contact.icer.msu.edu/
- HPCC Request tracking system rt.hpcc.msu.edu
- HPCC Phone (517) 353-9309
- HPCC Office 1400 PBS
- Open Office Hours 1pm Monday (BPS 1440)



