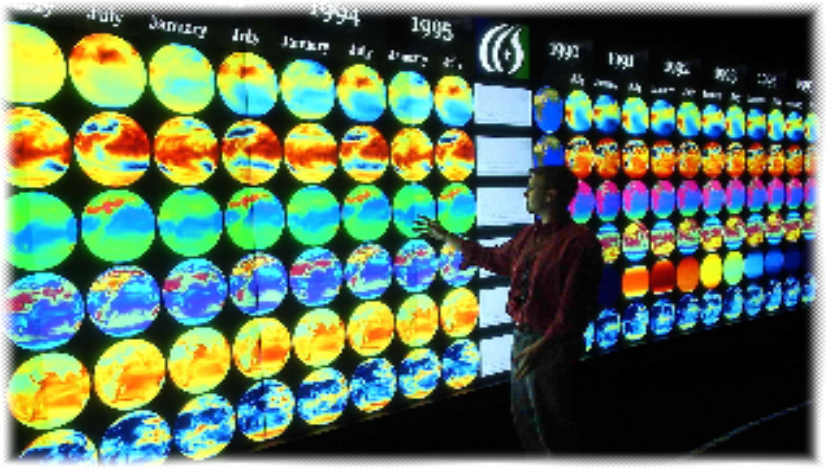
Computational Science for Undergraduate Research Experiences (CSURE-REU)



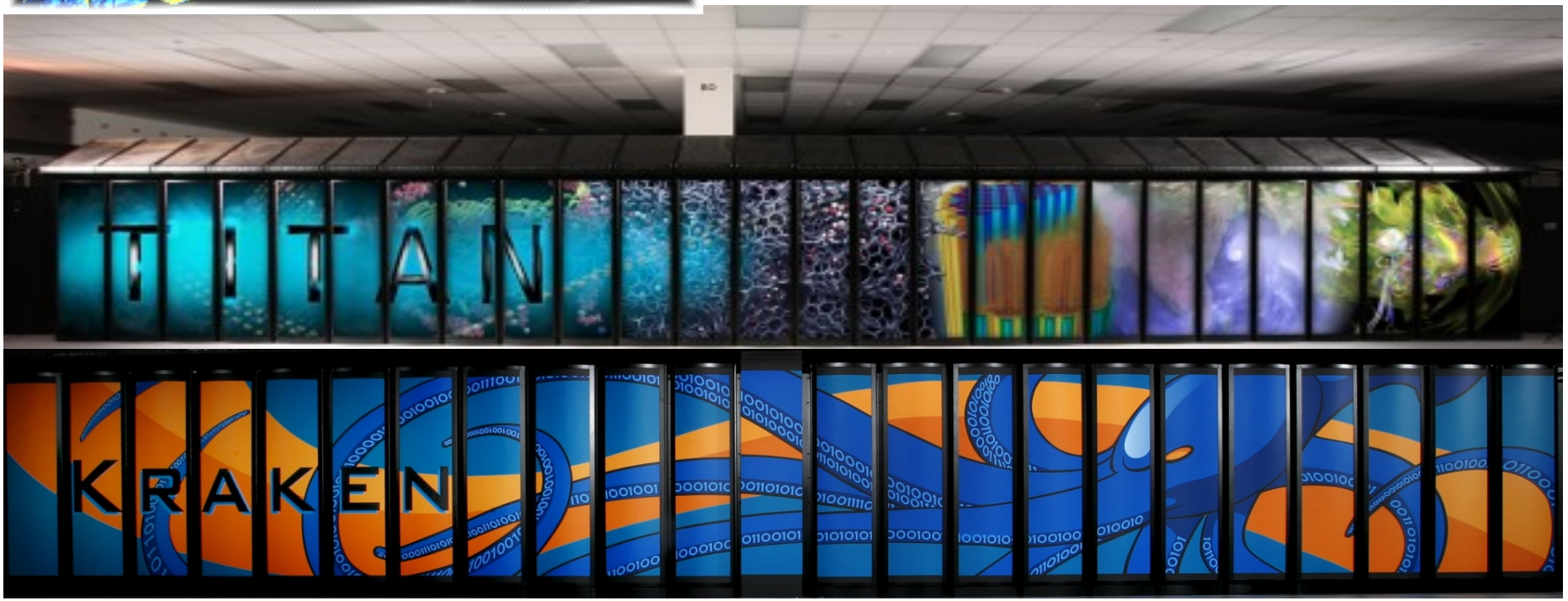
Kwai Wong

Joint Institute for Computational Sciences

kwong@utk.edu

www.jics.utk.edu/staff/kwong

June 3, 2013



Agenda

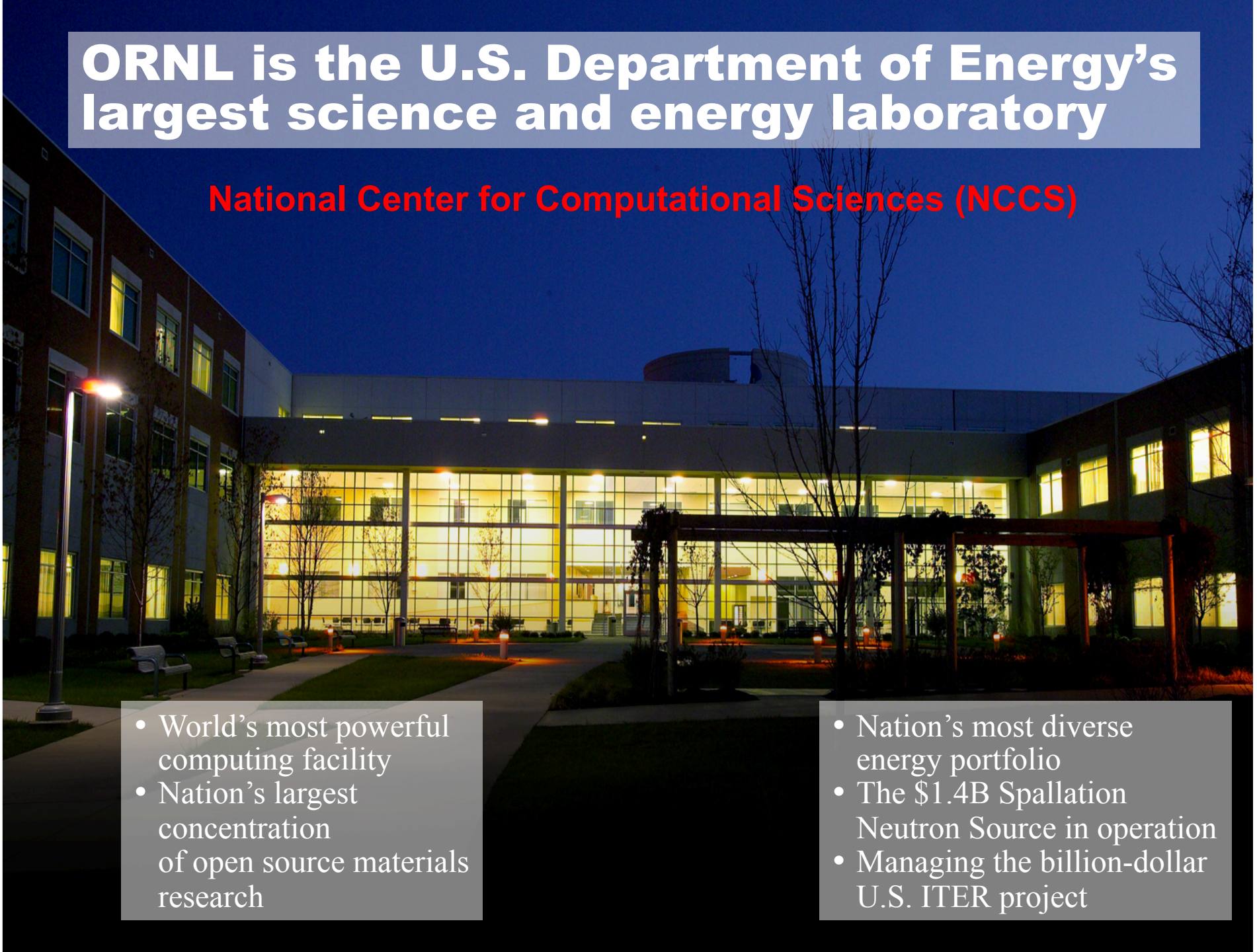
- Welcome to Knoxville , introduction to JICS personnel
- Logistics : I-9 forms, questionnaire, Apartment, questions
- Shuttle leaves at 8:30 at Quarry Trail Apt and pick up at 5:30pm
- One time Grocery run this Monday at 6;00pm. Or every Saturday
- Program starts at 9:00 am and ends at ~5:00 pm with 90 minutes lunch break.
- Round table introduction
- Project assignments , teams
- First week is basic training, June 3 – 7
- Monday morning- round up, campus walk, afternoon- Linus OS, vi
- Tuesday : compiling, C, Fortran, C++
- Wednesday : programing and scripting languages
- Thursday : ORNL , badging, tours, NICS overview
- Friday : Project overview

Menu

Joint Institute for Computational Sciences

- JICS is a collaboration between UT and ORNL since 1991
- Joint Faculty, Research, Education, Outreach
- Kraken (65M), RDAV (10M), Keeneland (12M), Beacon (1M)
- Staffed with 40+ FTEs, multiple projects NSF, DOE, DOE, ..
- Total JICS funding > $100M

ORNL is the U.S. Department of Energy's largest science and energy laboratory



- World's most powerful computing facility
- Nation's largest concentration of open source materials research

- Nation's most diverse energy portfolio
- The $1.4B Spallation Neutron Source in operation
- Managing the billion-dollar U.S. ITER project

Titan: World's Most Powerful Computer

- Cray XK7 – 27.12 PetaFLOPS (T. Peak), 17.59 PFLOPS (HPL)
- 18688 AMD Opteron processor– 16 cores, 32 GB memory - 2.26 GHz
- CPU 2,26 x 4 x18688 = 2.392 PF,
- 18688 K20X Nvidia GPU- 6GB memory, 14 active SM, DP 1.31 TFLOPS
- (CPU) 2.26 x 4 x 18688 = 2.392 ; (GPU) PF 1.31 x 18688 x 14 = 24.27 PF
- 64.8% ; 710 TB RAM ~ 10 times faster than jaguar; 9 Megawatt,

Menu

Kraken: 1st Academic PetaFLOPS Computer (3rd 2009)

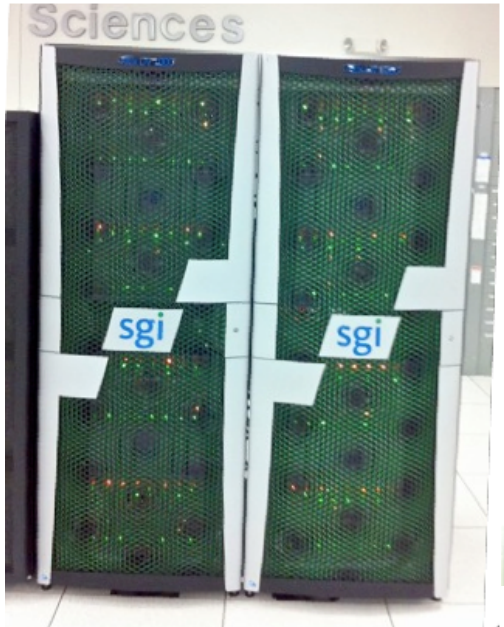
- Cray XT5 – 1.17 PetaFLOPS (Peak)
- 100 cabinets in 4 rows
- 9408 compute nodes (112896 cores)
- Each node has 12 cores - 2.6 GHz AMD (Istanbul) Processor
- 16 GB RAM per node
- 147TB of compute memory
- Scratch disk space, with 2.4PB of usable space
- www.nics.utk.edu

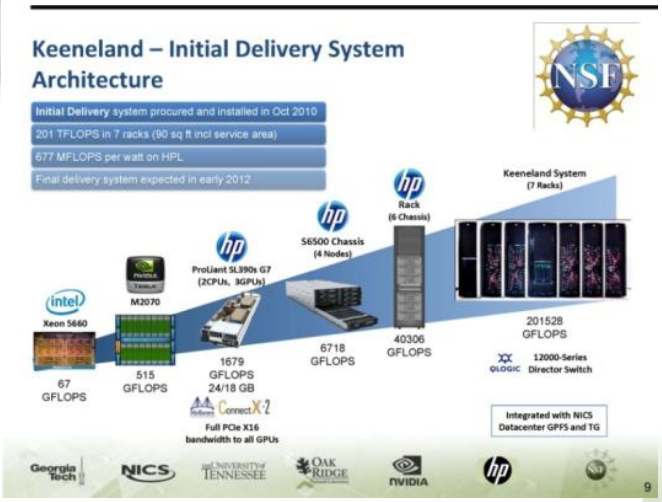
Nautilus & Keeneland

RDAV

- SGI Ultraviolet – 10 TFLOPS (Peak)
- 128 nodes x 8 cores (1024 cores) + 16 GPU
- 4.0 GB per core ; 4 TB Global Addressable RAM SMP
- 1 PB parallel file space; addressable from kraken
- Data Analysis; Pre & Post processing

- HP SL250 – 264 node 615 TFLOPS (Peak), # 74
- 1 node : 2 Sandy Bridge,16 cores + 3 M2090 GPU
- 32 GB per node + 6 GB / GPU
- 4224 cores + 792 GPU
- 1 PB parallel file space; lustre
- Interconnect : infinite band 4 x QDR
- Multi-GPU processing

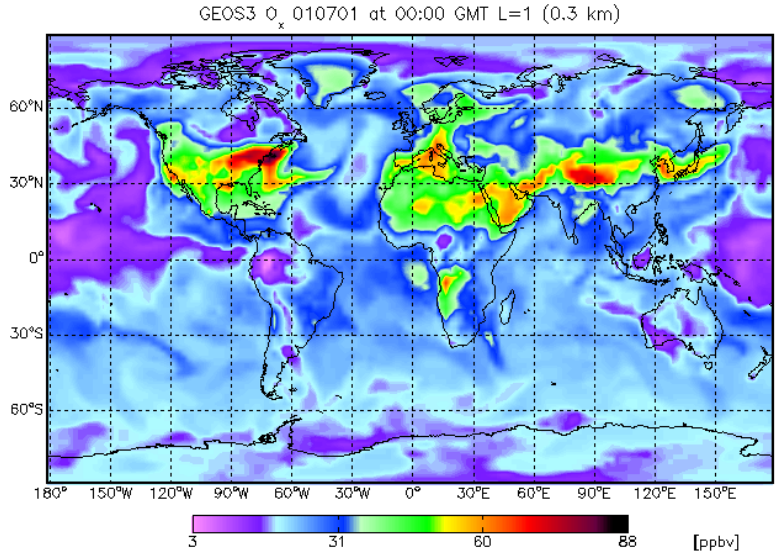


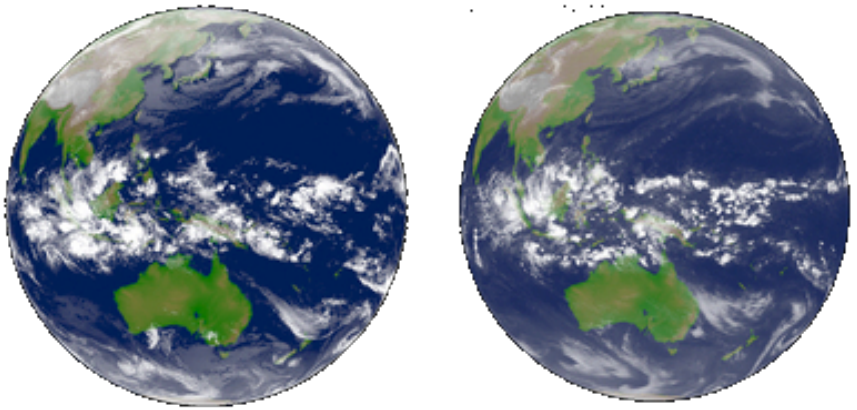


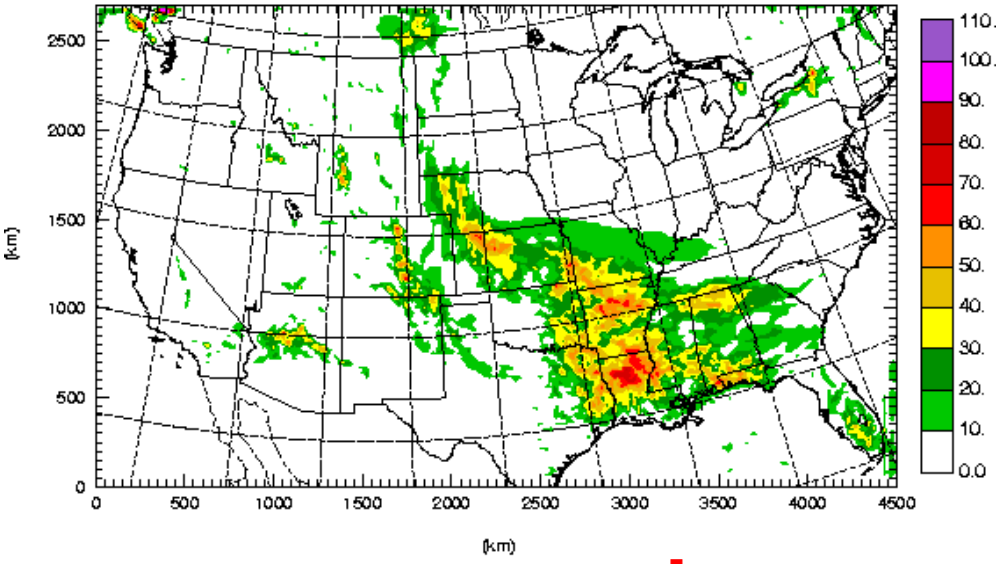
WORLD RECORD!
"Beacon" at NICS

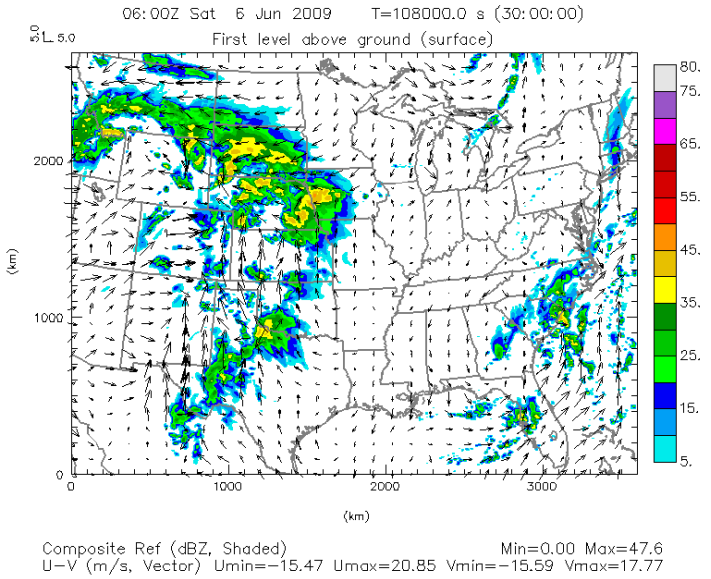
Intel® Xeon® + Intel Xeon Phi™
Cluster
First to Deliver
2.499 GigaFLOPS / Watt
71.4% efficiency
#1 on current Green500

Climate Simulations and Weather (Storms) forecast







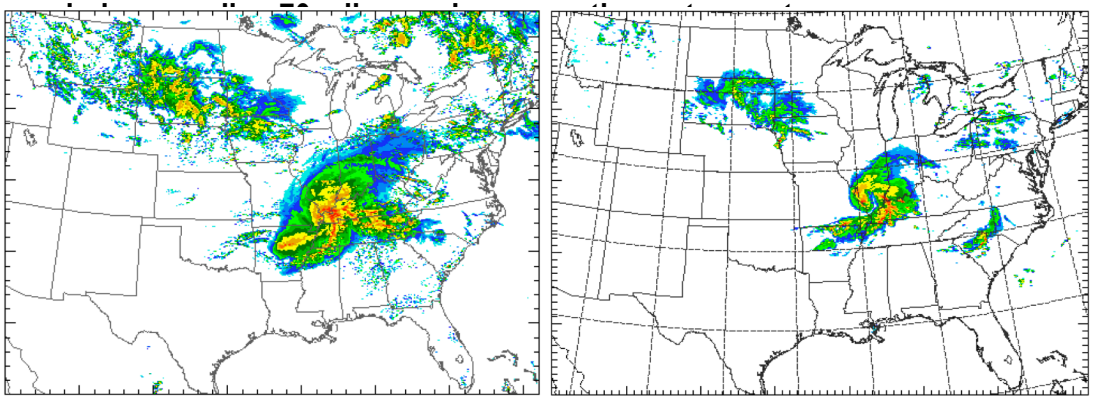


www.caps.ou.edu

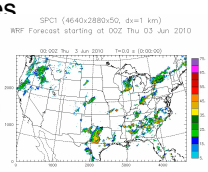
Center for Analysis and Prediction of Storms (CAPS), U. Oklahoma

Predicting Continental US Scale Weather at up to 1 km Grid Spacing in Realtime with Full-Scale Radar Data Assimilation

- **26 x 4km and one 1km forecasts were performed on the XT4 Athena using all 18000 processor cores in dedicated mode 5 nights each week during April – June 2010**
  - **the 30-hour long forecasts typically take 5 hours to complete during the overnight hours.**
  - **The same machine was also used to run the two ARPS 4 km forecasts as part of the ensemble.**
  - **Running realtime forecasts at such a high resolution for a continental-scale domain was a first in this line of research, while direct assimilation of data from over 120 operational weather radars at such a high resolution had never been done before.**
- **The figure shows an example forecast at 1 km grid spacing for a case where widespread wind, hail and flooding damages occurred over southern Missouri and southern Illinois**
  - **Nearly 30 tornadoes were reported in the same region. The severe weather was caused by an intense mesoscale convective vortex that contained a large bow echo.**
  - **Figure shows a comparison between the 18-hour forecast of the mesoscale vortex on the 1 km grid, as compared to the radar observation. The model reproduced the mesoscale vortex very well and predicted a large area of intense surface**



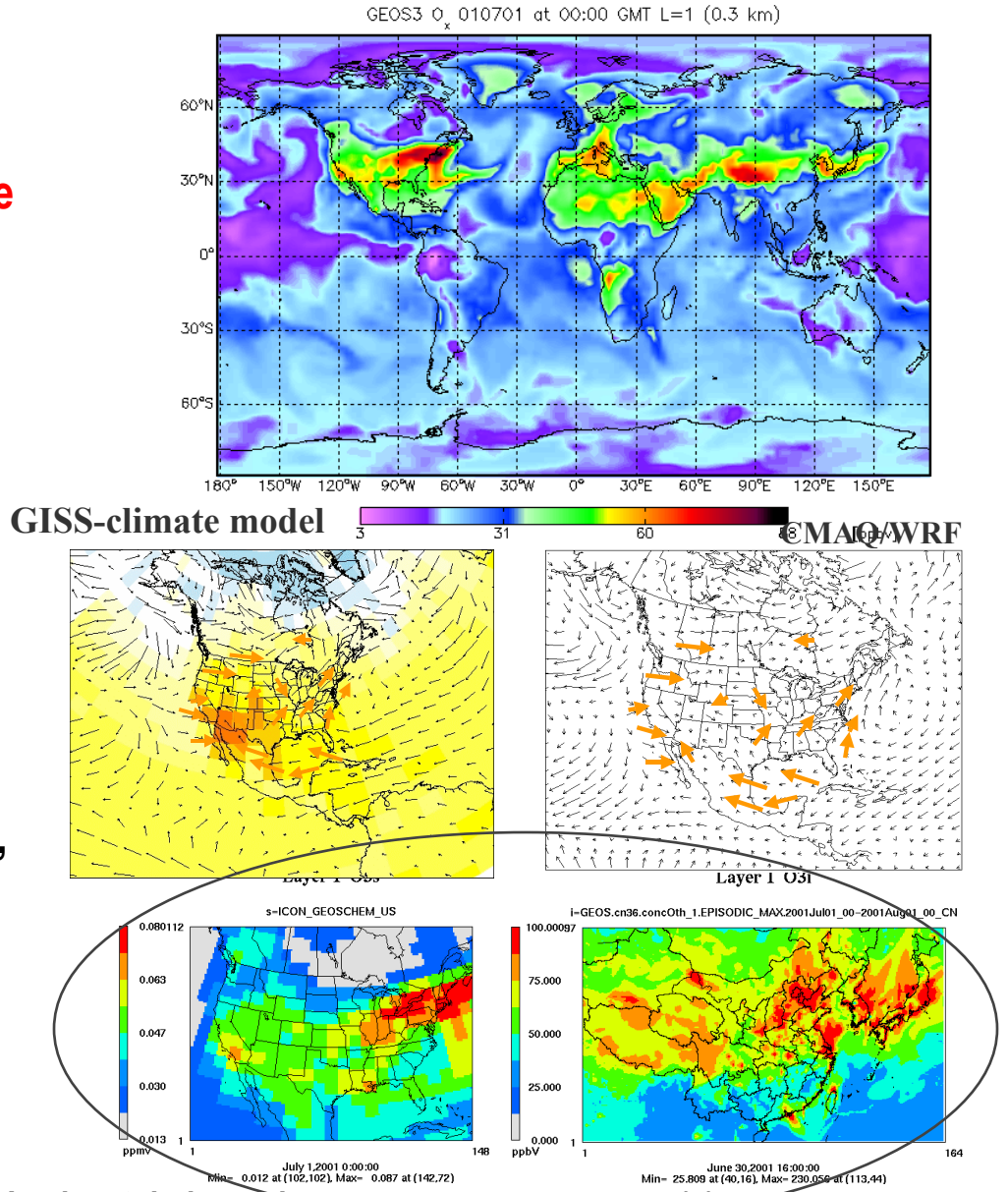
Radar reflectivity field produced by the CAPS 1-km forecast on NICS/ UTK Cray XT5, using 9,600 cores (left), as compared to radar observation of the same quantity (right). The forecast length is 18 hours and the fields are valid1t 18UTC, 8 May 2009.



Multi-dimensional Climate and Air Quality Study,

Joshua Fu

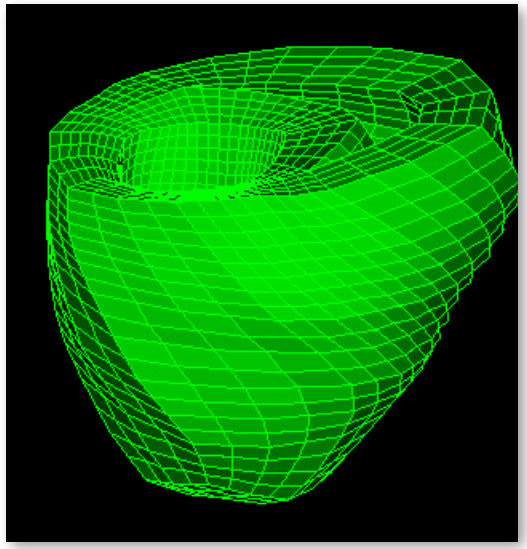
- Predict U.S. air quality in 2050 for future air quality planning
- Evaluate the effect of U.S. climate changes in 2050
- Pollutants source and receptor study for United Nations
- Downscaling applications coupling climate and air quality model, CCSM to WRF, CMAQ
- Challenges : petacale computing, 2 TB data per yearly simulation per scenario, workflow managements, and data postprocessing
- NICS helping validate models

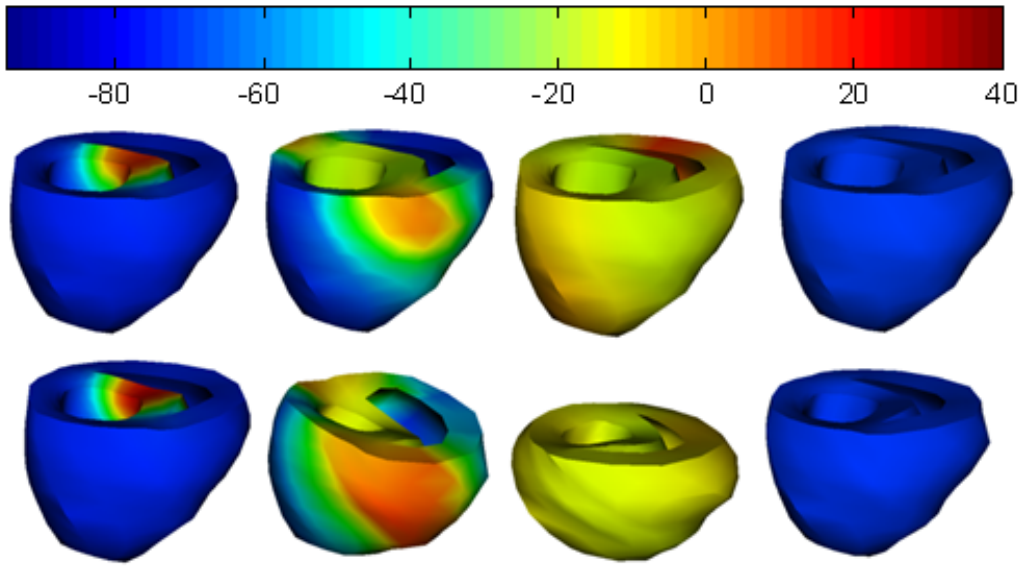


Mathematical Modeling of Heart Rhythm Disorders,

Xiaopeng Zhao, MABE, UTK

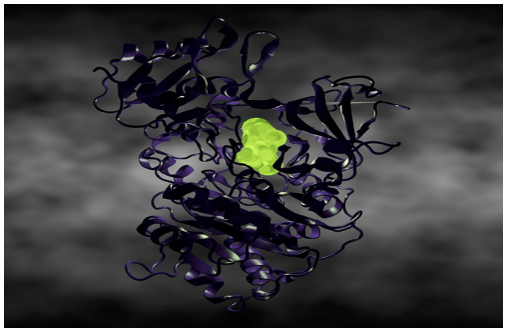
- A complete understanding of heart rhythm disorders requires a system-levels investigation on the interaction between electrical, chemical, and mechanical activities on biological scales ranging from ion channels to single cells to multi-cellular tissue and organ.
- The goal is to develop a viable computing framework to model the cardiac electrical wave propagation of the human heart. The work will integrate models from multiple physics fields including electrophysiology, electro-mechanics, and mechanical deformations.



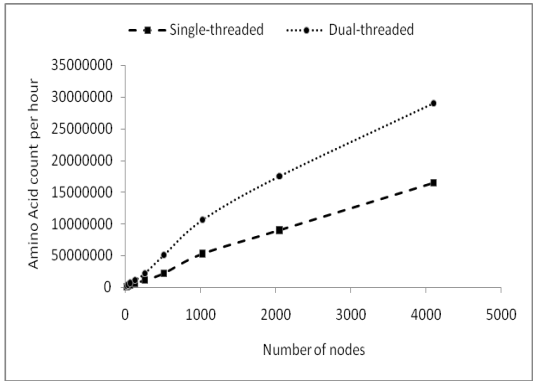


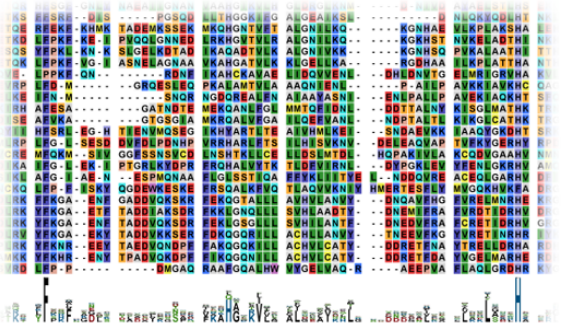
Highly Scalable Parallel HMMER and BLAST

C. Halloy, B. Rekepalli, and I. Jouline*, UTK



- **HMMER – Protein Domain Identification tool**
  - **MPI-HMMER limited performance**
  - **HMMER compares sequences to a database of hidden Markov models to identify known domains within the sequences**
- **New HSP-HMMER code - Excellent performance**
  - **Currently ~10000x faster than MPI-HMMER for 1K processes**
  - **Scales up to 98,000 cores very well**
- **HSP-HMMER reduces time to identify the Pfam functional domains in 6.5 millions proteins of the "nr" (non redundant) database from 2 months on clusters down to less than 10 minutes! using 98000 processing cores..**
  - **B Rekepalli, C Halloy, IB Jouline. "HSP-HMMER: a Tool for Protein Domain Identification on a Large Scale," ACM SAC 2009, 766-770.**
- ***This is critical, considering that the protein database continues doubling in size every 6 months!***
- ***HSParallel BLAST now scales to 50,000 cores on Kraken. Tests are still under way.***



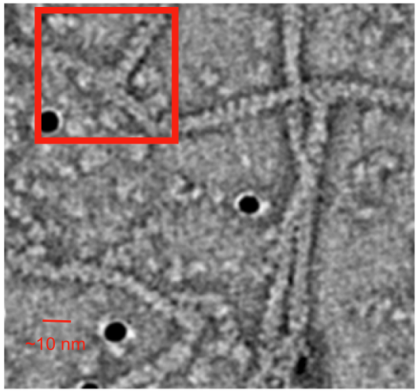


A part of an alignment for the Globin family from Pfam

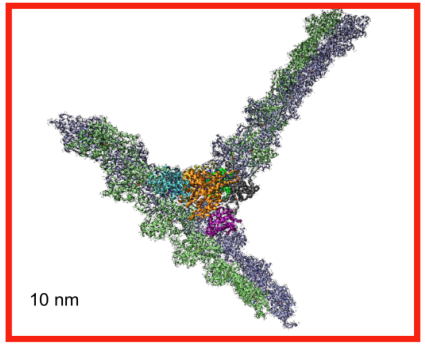
Multiscale Simulation of Biological Assemblies
Greg Voth, U. Utah

The actin-Arp2/3 branch junction

- Confers shape and structure to most types of cells
- Among the largest biomolecular MD simulations performed to date (NAMD, 4,000 cores).

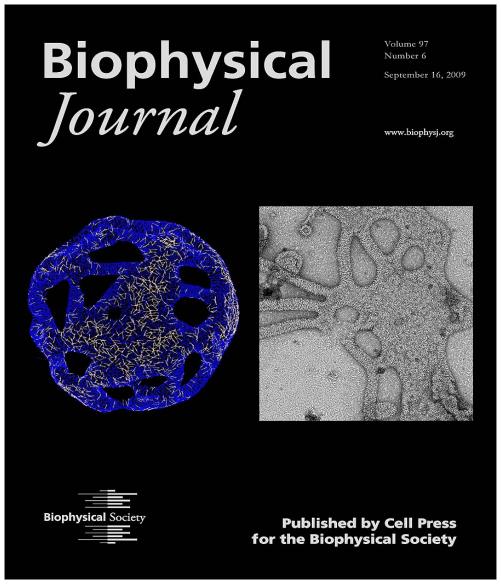


Cryo-EM image of actin cytoskeleton[1]



Atomic model of the actin-Arp2/3 branch junction

[1] I. Rouiller et al. 2008. *J. Cell Biol.* 180:887-895.

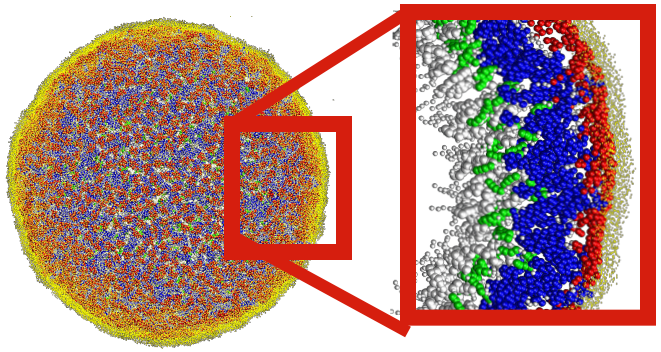


Blue figure is simulation. Grey figure is experimental collaborator results. Blue figure matches grey figure -- theory meets experiment in biology.

Multiscale simulations of membrane remodeling. The *first* direct comparison of mesoscale simulation with electron microscopy imaging. (0.75 million CG sites, equivalent to 10^{11} atoms, using TANTALUS over 2,000 cores).

The *first* CG-MD simulations of the entire immature HIV-1 virion

- 0.75 million CG sites, equivalent to 10^8 atoms, using TANTALUS over 2,000 cores



Science enabled by NICS

- Petascale supercomputing resources allow for long timescale simulation.
- Thus able to provide meaningful feedback to experimental research in structural biology
- "Kraken is fundamentally changing how we think about molecular simulation: things we used to dream about doing are now possible"

Atomistic Simulations of Future Nanoelectronics Transistors
Mathieu Luisier, Purdue

Objectives:

- accelerate nanoscale transistor innovation with petascale simulation
- help experimentalists design low power nanodevices

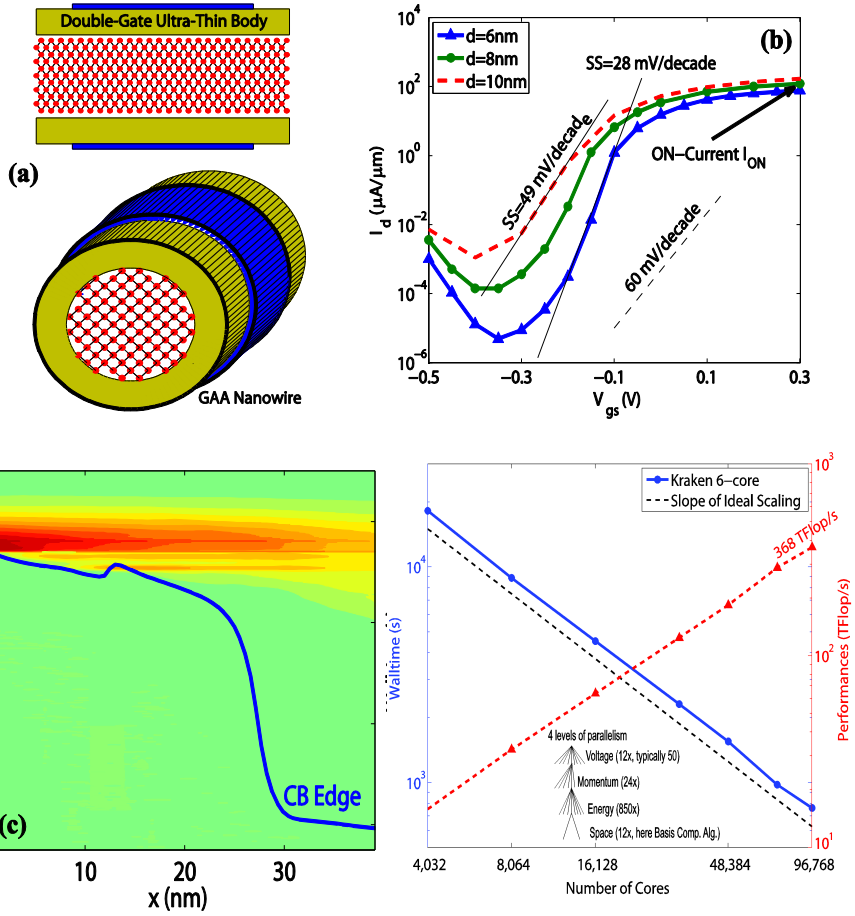
Approach: OMEN - a massively parallel, atomistic, full-band quantum transport simulator for 1-D, 2-D, and 3-D nanodevices based on the Non-equilibrium Green's Function Formalism

Results:

- Reproduced InAs HEMT experimental data in an hour with 96,768 cores rather than weeks on cluster; 368 Tflops/s
  - Will be part of a paper on electron-phonon scattering in nanowire TFETs and opens a door towards larger device structures
- Si nanowire with 4nm diameter simulated on 3,000 nodes with 8GB of memory per core

Impacts:

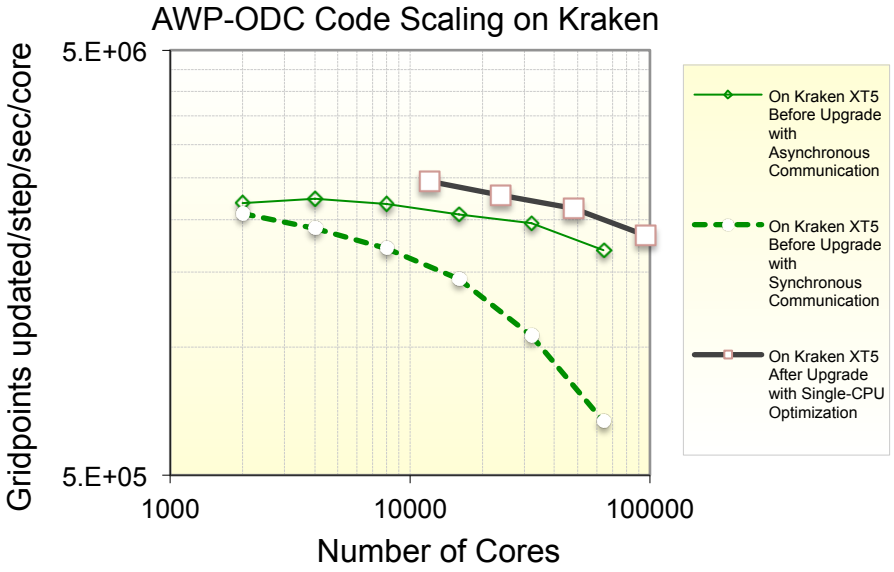
- first demonstration of electron-phonon scattering in a 3-D, atomistic, and full-band basis on Kraken.
  - proved that electron-phonon scattering plays a more important role when the diameter of the nanowire increases, as expected
- Full machine runs are the key to simulating large device structures and to reducing the computational time down to the minute scale instead of months on a single core.

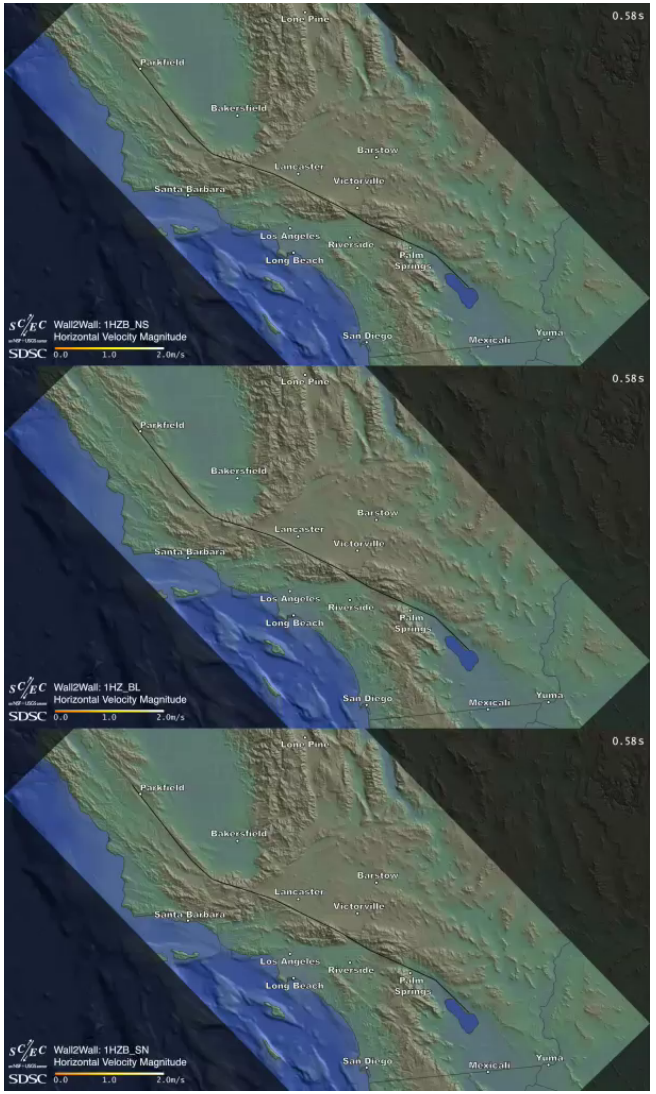


Overview of OMEN capabilities. (a) Examples of nanoelectronics devices that OMEN can handle (double-gate ultra-thin-body and gate-all-around nanowire FET). (b) Transfer characteristics Id-Vgs of different types of TFETs. (c) Spectral current of a GAA NW FET with electron-phonon scattering. (d) OMEN scaling performances and sustained performance on Kraken up to 96,768 cores.

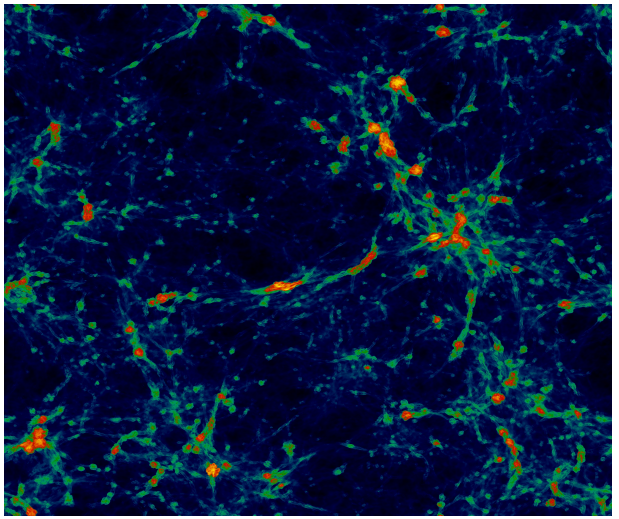
Simulating the Big One on Kraken
T. Jordan, Southern California Earthquake Center

- Biggest Earthquake Simulation on San Andreas Fault, the Big One
- Simulated in a 32 billion grid point subset of the SCEC Community Velocity Model (CVM) V4 with a minimum shear-wave velocity of 500 m/s up to a maximum frequency of 1 Hz.
- 96,000 processor cores used for production runs on Kraken, 2.6 hrs WCT, 53 sustained TeraFlop/s



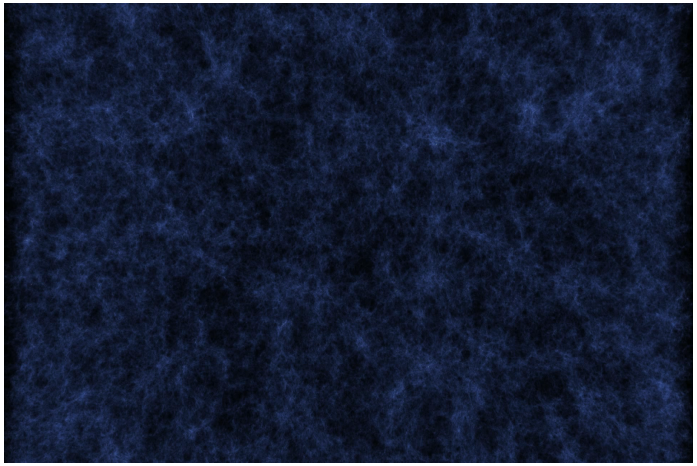


Simulating the formation of early galaxies
Robert Harkness, UC San Diego



ENZO - Hybrid MPI/OpenMP code

- Current model is $6{,}400^3$ = 268 billion cells and dark matter particles!
  - **Definitely the World's Largest!**
  - **Star formation and feedback (energy & momentum)**
  - Running on 93,750 cores, 125 TB of Kraken
    - "A Blue Waters scale" simulation
  - Largest hydrodynamic cosmology simulation ever done
  - First to simulate large enough volume of the universe to resolve galaxies across a sufficiently wide range of masses and luminosities
  - Last checkpoint at redshift 15.5; need to get to 6
    - Requires about 10 more 24-hour 94,000 core runs
- *"The most productive platform in NSF portfolio for ENZO simulations, bar none," Harkness*



\

Computational Science for Undergraduate Research Experiences (CSURE)

Basics of LINUX OS

Kwai Wong, JICS, UTK

kwong@utk.edu

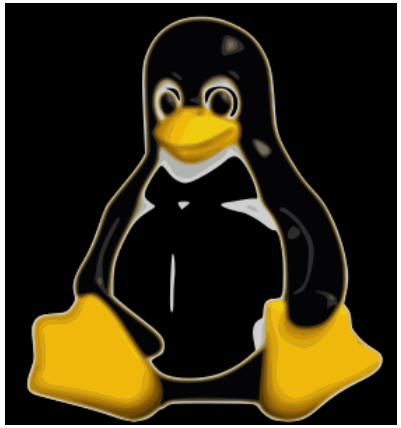
June 3, 2013

What do we use and do

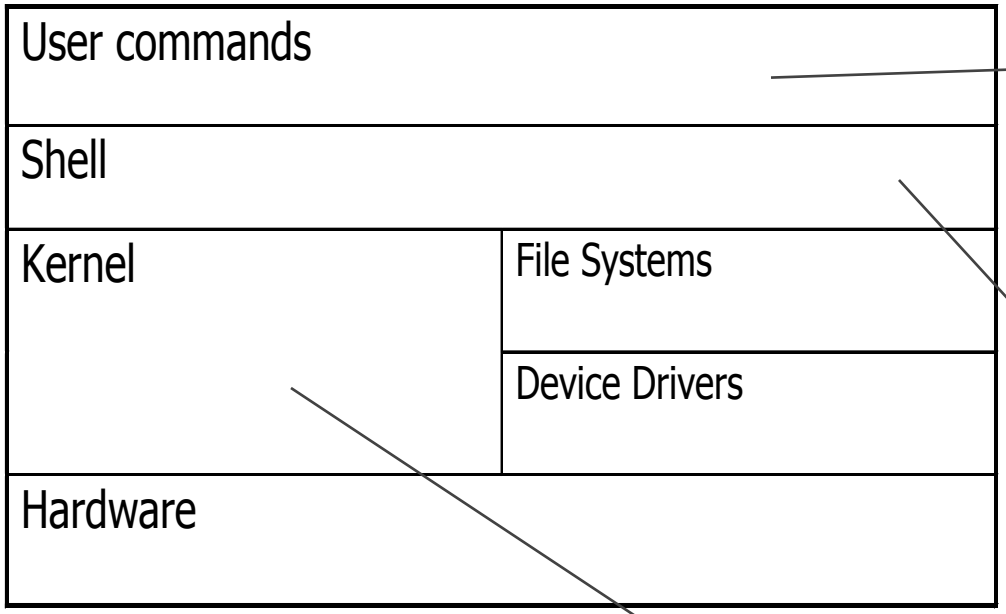
- Linux operating system in general
  - FILE, PATH, FILE MODE
- General overview Linux OS and terminal commands
  - file, program, executable program
  - cd , ls, mkdir, cp, mv , rm, xedit, gedit, env, path
- Tools and simple programming skills
- Compilers – gcc, g++
  - “gcc –o pexe ./prime.c” ; “ ./pexe “
- Projects, exercises, challenges, games !!
- Summer projects ??

LINUX – timeline

- In 70's, UNIX OS by Bell Lab, for main frame computer
- C is developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix OS
- In 80's, Microsoft's DOS, Apple MAC
- GNU project started by Richard Stallman in 1984 : free software, C compiler in 1991
- Linux Torvalds, a college sophomore, wrote the first Linux kernel in Sept. 1991 based on Minix developed by Anfrew Tanenbaum. …. UNIX on PC ~~ LINUX
- www.linux .org, www.gnu.org

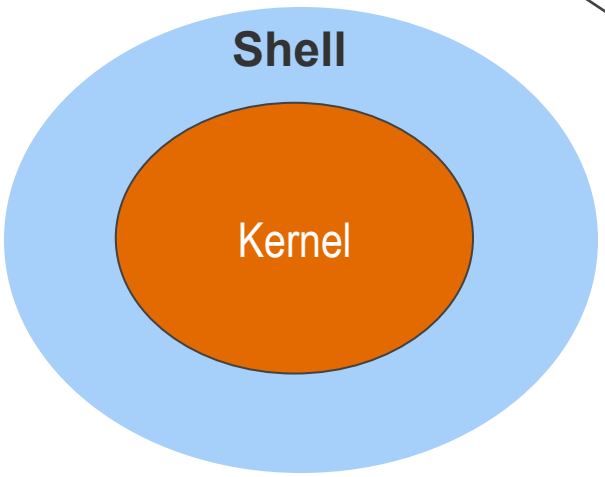


The Linux System – OS



User commands includes executable programs and scripts

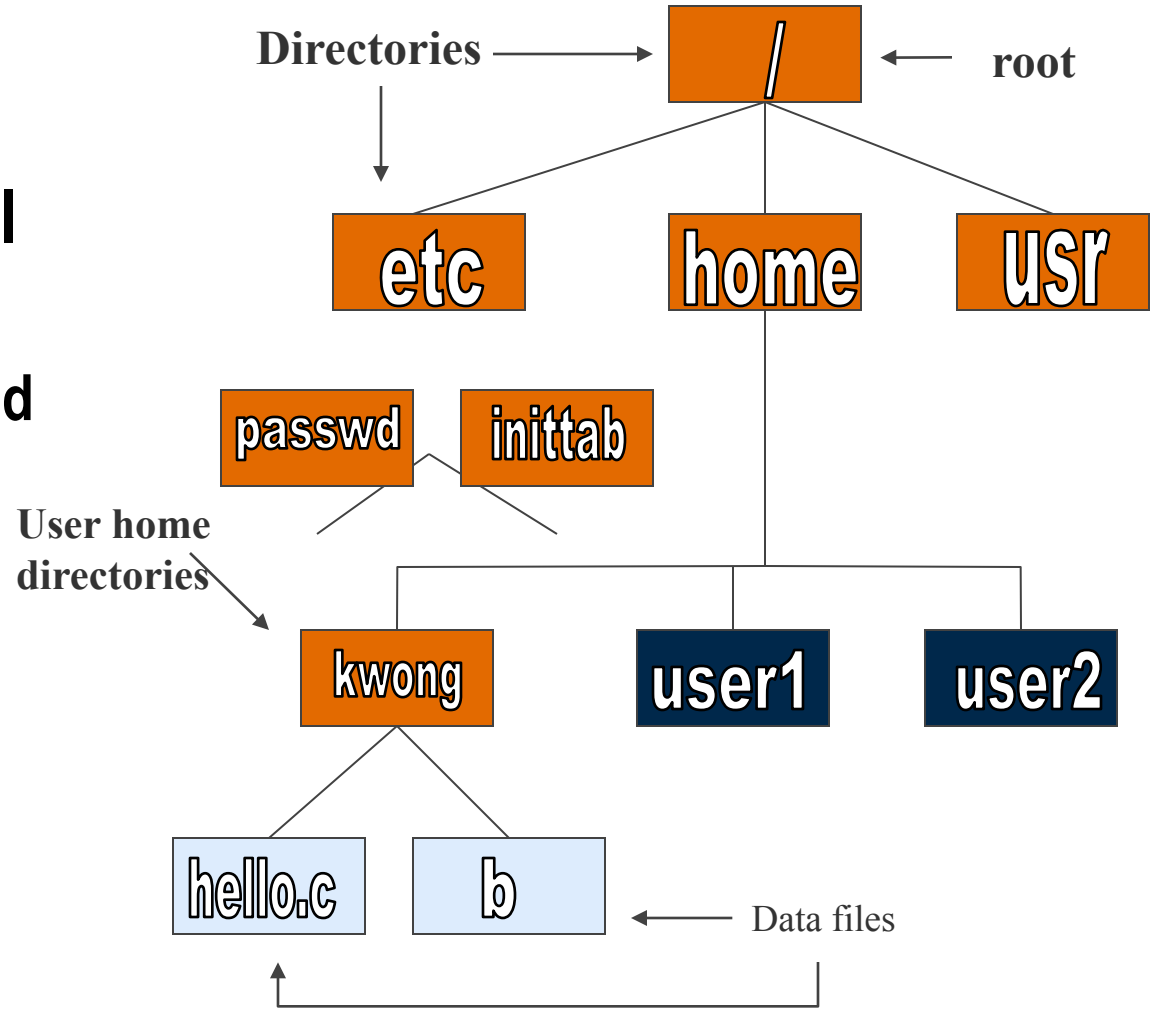
The shell interprets user commands. It is responsible for finding the commands and starting their execution. Several different shells are available. Bash is popular,



The kernel manages the hardware resources for the rest of the system.

Linux File System

- Linux files are stored in a single rooted, hierarchical file system
  - Data files are stored in directories (folders)
  - Directories may be nested as deep as needed



LINUX OS – FILE , FILE, more FILES

- The Linux kernel is written in C
- EVERYTHING is considered as a FILE in Linux
- FILE ~~ program : allow read, write, execute

  Create a file --------- simple text

  Compile a file ------- object file, machine file

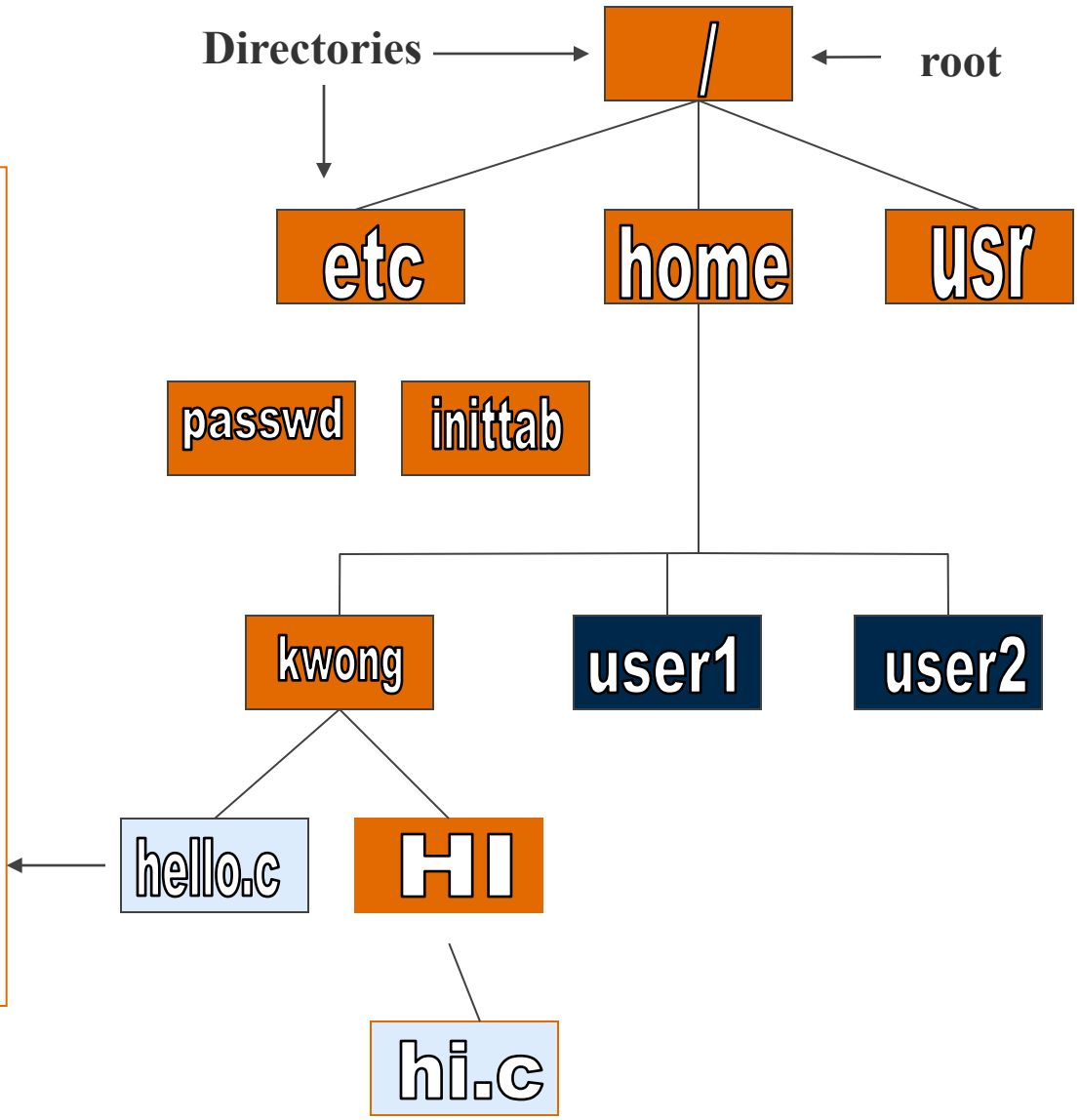
  Link a file ---------- executable file, run on the machine

```
/* Simple Helloworld C Example : hello.c */
#include <stdio.h>

int main ()
{
    printf( "Helloworld \n");

return 0;
}
```

To compile : > gcc –o hexe ./hello.c
To run in a computer : > ./hexe

Directory and Path : Absolute and relative

```
$> cd
$>pwd
/home/kwong
$> ls
hello.c  HI
$>cd HI
$> ls
hi.c
$>pwd
/home/kwong/HI
$> cd ..
$> pwd
/home/kwong
```



Path

- **Absolute**
  - use pwd to find the address of the file
  - E.g. home/kwong/CLASS/”file” in the CLASS directory”
- **Relative**
  - use ./ to tell the computer the program is in the current directory
  - E.g use the ./”selected file in the current directory”

File Permissions : read (r), write (w), execute(x)

- **The long version of a file listing (`ls -l`) will display the file permissions:**

```
-rwxrwxr-x   1 kwong  kwong      5224 Dec 30 03:22 hexe
-rw-rw-r--   1 kwong  kwong       221 Dec 30 03:59 hello.c
-rw-rw-r--   1 kwong  kwong      1514 Dec 30 03:59 hello.s
drwxrwxr-x   7 kwong  kwong      1024 Dec 31 14:52 HI
```

Permissions Owner Group

-rwxrwxrwx (777)

- Other permissions
- Group permissions
- Owner permissions
- Directory flag (d=directory; l=link)

READ ~ 4 ; WRITE ~ 2 ; EXECUTE ~ 1 === total ~ 7

Changing File Permissions

- Use the **chmod** command to change file permissions
  - The permissions are encoded as an octal number

```
chmod 755 file  # Owner=rwx Group=r-x Other=r-x
chmod 500 file2 # Owner=r-x Group=--- Other=---

chmod 644 file3 # Owner=rw- Group=r-- Other=r--
chmod +x  file  # Add execute permission to file for all
chmod o-r file  # Remove read permission for others
chmod a+w file  # Add write permission for everyone
```

READ ~ 4 ; WRITE ~ 2 ; EXECUTE ~ 1 === total ~ 7

Installation

- Burn Ubuntu 12.4 from the Linux website
- Put in CD and press F12 while the machine is turned on
- A boot menu should pop up and press boot up from CD-Drive
- Configure Settings, set username and password, time zone ,and internet connection
- Restart machine and access your account, make sure to take out the CD when you reboot

Terminal

- For interactive access to your computer, use command to do the work.
- Way to view all of your personal files and hidden files such as source code
- Also used to write programs in languages such as C++, C, Fortran, and Python

Commands

- ls – lists files
- top – shows what programs the computer is running
- cd – changes directory
- cd .. – goes back one directory
- cp “filename” “newfilename” – copies files
- mv “filename” “newfilename” – moves the file
- ls –l – shows what can be read, written, and executable
- pwd – tells your absolute path of the file you are in
- mkdir – makes a directory
- sudo apt-get – gets something you need (update or install, Ubuntu only)
- env

VI editor

- Used to create a file
- 2 modes: Insert and View
- Press ESC to be in View mode
- Press letter “i” to be in insert mode
- To save your work press ESC and “:wq”
- To quit without saving press ESC and “:q!”
- Webpage for help : google “vi editor summary pdf”
- http://www.cs.colostate.edu/helpdocs/vi.html or others

Starting vi – the vi material are copied from the webpage

Opening an existing file

vi *filename*

Creating a new file

vi *filename*

In your workshop directory, create a new file called ***mytext***

```
vi mytext
```

Vi Modes of Operation

- **Command Mode**
  - **Allows the entry of commands to manipulate text**
  - **Default mode when vi starts**
  - **Use Escape key to move into command mode**
- **Insert Mode and**
  - **Puts anything you type into the current file**
  - **To get into insert mode, commands are *a* (append) and *i* (insert)**

*1. Use the **i command** to move into insert mode (**Press i key**).*
2. Attempt to type in the title of your favorite song.
*3. Use the **Esc key** to move to command mode.*

Quit and Save Changes in vi

:q Quit the editor

:q! Quit without saving changes to the file

1. *Use the **Esc key** to make sure you are in command mode.*
2. *Use the **:q** command to try to quit vi*
3. *Use the **:q! command** to force quit without saving (**Enter :q!)**.*

:wq Write/save changes and quite

:w Write/Save changes, but don't quit

1. *Type **vi mysong** to re-edit your song file.*
2. *Use the **i command** to move into insert mode (**Press i key**).*
3. *Retype the title of your favorite song.*
4. *Use the **Esc key** to move to command mode.*
5. *Use the **:w** command to write/save your edits to file*
6. *Use the **i command** to enter **insert** mode (**Enter i)**.*
7. *Type **Title:** somewhere on the line with the song title.*
8. *Use the **Esc key** to move to command mode.*
9. *Use the **:wq** command to save and quit vi .*

Vi Editor

- **How to type commands in command mode**

  [count] command [where]

  count : Its a number

  where : Specifies how many lines or how much of the document the command affects. It can also be any command that moves the cursor.

Moving the cursor in vi

h key **move cursor to the *left* one position**

l key **move cursor right *one* position**

j key **move cursor *down* one line**

k key **move cursor *up* one line**

1. *Type **vi mysong** to re-edit your song file.*
2. *Use the **l command** several times to move cursor to the far right*
3. *Use the **a command** to move into **append** mode (**Press a key**).*
4. *Use the **Enter key** to start a new line of text.*
5. *Type: **Artist:** and then the name of the artist*
6. *Use the **Esc key** to move to command mode .*
7. *Practice moving cursor up, down, left, and right with **h,l,j,k** keys.*

Simple vi editing commands

r **replace one character under the cursor**

x **delete 1 character under the cursor.**

2x **delete 2 characters (3x, etc.)**

u **undo the last change to the file**

1. *Use the **Esc key** to make sure you are still in **command mode**.*
2. *Reposition your cursor and use the **a, l, r** and **x** **commands** to repair any typos in your title and artist, and change the title to **ALL CAPS***
3. *Use the **:w** command to save your changes.*

Cutting text in Vi

d^

Deletes from current cursor position to the beginning of the line

d$

Deletes from current cursor position to the end of the line

Dw

Deletes from current cursor position to the end of the word

dd

Deletes one line from current cursor position. Specify count to delete many lines.

Cutting & Yanking Text in Vi

dd	**Delete (cut) 1 line from current cursor position**
2dd	**Delete (cut) 2 lines (*3dd* to cut 2 lines, etc.)**
p	**paste lines below current line**

1. *Move cursor to top line and type **dd** to cut the title line*
2. *Use the **p** command to paste the title line below the artist line*
3. *Use the **p** command to paste it again.*

Cutting & Yanking Text in Vi

yy **yank (copy) a single line**

2yy **yank (copy) 2 lines (*3yy* to copy 3 lines, etc.)**

P **paste lines before current line**

1. *Move cursor to first of the 2 title lines and type **2yy** to yank/copy 2 lines*
2. *Move cursor to the first line, then use the **capital P** command to paste the two yanked links above the artist*

Vi Editor

To go to a specific line in the file

:linenumber

1. *Go to the 3rd line by typing* ***:3***
2. *Go to the 1st line by typing* ***:1***
3. *Go to the last line by typing* ***G***

Vi string/search

/[pattern] search forward for the pattern

?[pattern] search backward for the pattern

n search for the next instance of a string

1. *Search forward for the next line containing the string **Title** by typing **/Title***
2. *Search forward for the next instance of **Title** by typing **n***
3. *Search backward for the most recent instance of Title by typing **?Title***
4. *Search backward for the next most recent instance of Title by typing **n***

More commands

yl

yank a single character. Specify count to yank more characters

yw

yank a single word. Specify count to yank more words

d^

Deletes from current cursor position to the beginning of the line

d$

Deletes from current cursor position to the end of the line

Dw

Deletes from current cursor position to the end of the word

Practice Editing with vi

Take 5 minutes to practice what you've learned by entering as many of the lyrics to the song as you can.

Use yank and paste to repeat chorus lines.

Use :w to write changes every 30 seconds.

Have one title line at line 1.

Have one artist line at line 2.

Save file and exit vi when finished or time expires.

CSURE-REU

Compiling, Linking, Performance

Kwai Wong, JICS at UTK

kwong@utk.edu

June. 3, 2013

TOP 500 – www.top500.org

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT YH Cluster, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
5	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00

75% of peak ;7.8 GFLOPS/CORE;

Numbers : Lots of Them:

- Core : computing unit : processor
- Dual core machine (Intel or AMD CPU) : a CPU with 2 cores, each core is a 2.4 GHz computing unit with 2GB of RAM (memory in the processor not disk space)
- Binary bits (b) : "0" or "1" , 1 Byte (B) = 8 bits
- Binary number : 11111111= $(2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^{0)} = (2^8 - 1) = 255$!!
- 32 bits machine or operating system => largest integer (all positive) = $(2^{32} - 1)$ = (4,294,967,296 -1) or range of integer = $-(2^{31})$ to $(2^{31} - 1)$
- 64 bits machine or operating system => range of integer = $-(2^{63})$ to $(2^{63} - 1)$
- Kilo (K) = 10^3 (or 2^{10}) ; Mega (M) = 10^6 (or 2^{20}); Giga (G) = 10^9 (or 2^{30}); Tera (T billion) = 10^{12} (or 2^{40}) ; Peta (P) = 10^{15} (or 2^{50})
- FLoating Point Operation (+, -, / , *) : (10.1 + 0.1) * 1.0 / 2.0 = 5.1 => 3 FLOP
- FLOPS = FLOP per second :: 1 PetaFLOPS (kraken) = 10^{15} FLOP in one second
- FLOPS in a core = (clock rate) x (floating point operation in one clock cycle)
- Peak Rate = (FLOPS in one compute unit, core) x (no. of core)

LINUX OS – FILE , FILE, more FILES

- The Linux kernel is written in C
- EVERYTHING is considered as a FILE in Linux
- FILE ~~ program : allow read, write, execute

  Create a file --------- simple text

  Compile a file ------- object file, machine file

  Link a file ---------- executable file, run on the machine

```
/* Simple Helloworld C Example : hello.c */
#include <stdio.h>

int main ()
{
    printf( "Helloworld \n");

return 0;
}
```

To compile : > gcc –o hexe ./hello.c
To run in a computer : > ./hexe

More about compiling : --

To compile : > gcc hello.c –o hexe
To run in a computer : > ./hexe

To compile : > gcc –c hello.c ====➔ hello.o
To link > gcc hello.o –o ./hexe ===➔ executable
To run > ./hexe

➢**gcc –v hello.c –o hexe**

……
COLLECT_GCC_OPTIONS='-v' '-mtune=generic'
/usr/lib/gcc/x86_64-linux-gnu/4.4.1/collect2 --build-id --eh-frame-hdr
-m elf_x86_64 --hash-style=both -dynamic-linker /lib64/ld-linux-x86-64.so.2 -z relro
/usr/lib/gcc/x86_64-linux-gnu/4.4.1/../../../../lib/crt1.o
/usr/lib/gcc/x86_64-linux-gnu/4.4.1/../../../../lib/crti.o
/usr/lib/gcc/x86_64-linux-gnu/4.4.1/crtbegin.o
-L/usr/lib/gcc/x86_64-linux-gnu/4.4.1 -L/usr/lib/gcc/x86_64-linux-gnu/4.4.1
-L/usr/lib/gcc/x86_64-linux-gnu/4.4.1/../../../../lib -L/lib/../lib
-L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/4.4.1/../../..
-L/usr/lib/x86_64-linux-gnu /tmp/ccm2W1PN.o
-lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed
-lgcc_s --no-as-needed /usr/lib/gcc/x86_64-linux-gnu/4.4.1/crtend.o
/usr/lib/gcc/x86_64-linux-gnu/4.4.1/../../../../lib/crtn.o

More hello1.c and myprint.c

```
/* More Helloworld C Example : hello1.c */
int myprint();

#include <stdio.h>
int main ()
{
    myprint( );
return 0;
}
```

```
/* myprint C function : myprint.c */
#include <stdio.h>

int mprint()
{
    printf( "Helloworld \n");
return 0;
}
```

To compile : > gcc –c hello1.c myprint.c ====➔ hello1.o, myprint.o
To link > gcc *.o –o ./hexe1 ===➔ combine all object files to an executable
To run > ./hexe1

Makefile :

```
hello1:
	gcc –O3 -c hello1.c
	gcc –O3 -c myprint.c
	gcc *.o -o hexe1

clean:
	rm *.o
```

```
CC = gcc
LINKER = gcc

hello1: *.o
	$(CC) –O3 -c hello1.c
	$(CC) –O3 -c myprint.c
	$(LINKER ) *.o -o hexe1

clean:
	rm *.o
```

To compile
> make
gcc –O3 -c hello1.c
gcc –O3 -c myprint.c
gcc *.o -o hexe1

>make
Nothing to do

To clean all object files:
>make clean

Timing Code – clock() , FLOPS

```
/* Simple timing C Example : tc.c */
#include <stdio.h>
#include <sys/types.h>
#include <time.h>

int main ()
{
  int i;
  double a=1.0, b=1.0, c=0.0, ttime, tflops;
  clock_t start, end;
  start = clock();
  for(i=0; i<1000000000;i++)  c=a+b;
  end = clock();
  ttime = (double) (end-start) / CLOCKS_PER_SEC;
  tflops = 1.0/ ttime ;
  printf( "  CPU time = %f ; GFLOPS = %f \n", ttime, tflops);
return 0;
}
```

To compile : > gcc –o tcexe ./tc.c
To run in a computer : > ./tcexe

matmul.c

```
#include <sys/types.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>

#define DIM 500

int main() {

   clock_t start, end;
   double a=1.0, b=1.0 , c=1.0, d=0.0, ttime, tflops;
   static double A[DIM][DIM], B[DIM][DIM], C[DIM][DIM];
   int i,j,k;

   for (i = 0; i < DIM; ++i) {
      for (j = 0; j < DIM; ++j) {
         A[i][j] = 1.0;
         B[i][j] = 1.0;
         C[i][j] = 0.0;
      }
    }
```

matmul.c

```
start = clock() ;
for (i = 0; i < DIM; i++) {
      for (j = 0; j < DIM; j++) {
           for (k = 0; k < DIM; k++) {
               C[i][j] = C[i][j] + A[i][k]*B[k][j];
           }
      }
   }
   end =  clock() ;
   printf("  C[0][0] = %f, C[last][last] = %f \n", C[0][0],C[DIM-1]
[DIM-1] );
   ttime = (double ) ( end - start)/CLOCKS_PER_SEC  ;
   tflops = 2.0*DIM*DIM*DIM/ttime/1000000000;
   printf("  CPU time = %f , GFLOPS = %f \n", ttime , tflops );

return 0;
}
```

To compile : > gcc –O3 matmul.c –o mmexe
To run in a computer : > ./mmexe

mm.f

```
PROGRAM dgemmtest
      IMPLICIT NONE

      integer i, j, nn, n, m, k, LD
      double precision A(500,500), B(500,500), C(500,500)
      double precision alpha, beta, rtime1, rtime2, rtime, rflops
      real etime          ! Declare the type of etime()
      real elapsed(2)     ! For receiving user and system time
      real startt, total  ! For receiving total time

       alpha = 1.0d0
       beta = 1.0d0
       nn = 500
       LD = 500

C      m = rows of A, k = cols of A (= rows of B), n = cols of B

      m = nn
      k = nn
      n = nn
```

mm.f

```
C     Generate matrices A, B, & C:
      do 30 i=1, m
        do 30 j=1, n
        A(i,j) = 1.0d0
        B(i,j) = 1.0d0
        C(i,j) = 1.0d0
30     continue

      startt = etime(elapsed)
      call dgemm('n','n',m,n,k,alpha,A,LD,B,LD,beta,C,m)
      total = etime(elapsed) - startt

      rtime=total*1.0d0
      rflops = 2.0d0*nn*nn*nn/rtime/1000000.0
      write(*,*) ' ***** MFLOPS = ', rflops
      write(*,*) ' ****  My time = ', total, C(1,1)

      end
```

> gfortran –O3 mm.f –o mmexe –L/home/kwong/LAPACK/lib -lblas
To run in a computer : > ./mmexe

CSURE-REU

FORTRAN F90 Overview

Kwai Wong, JICS at UTK

kwong@utk.edu

June. 3, 2013

F90 Features

- Major extension of F77
- All of FORTRAN 77
- Syntax improvements including free-form source
- Array operations
- New intrinsic procedures (arrays, bit manipulation)
- New and improved control constructs
- Dynamic storage allocation
- User-defined data types
- Pointers
- Procedure interfaces

Syntax Improvements

- Statements may appear anywhere
- Columns 1-6 are no longer reserved
- Line continuation -- “&”
- Trailing comments may be used -- “!”
- Multiple statements allowed in one line
- Underline symbol “_” is permitted
- 31 characters for length of variables
- Example :

```
TMP_VALUE_OF_X = X ; X = Y ; Y &
= TMP_VALUE_OF_X ! swap X and Y
```

Language Elements

- **Attributes are extra properties of variables in Type specifications**

```
INTEGER,  PARAMETER :: n=1000
REAL,  DIMENSION(n,n) :: a , b
```

- **Data Types :**

```
INTEGER, DIMENSION(10) : : m, n
REAL X(-10 : 20), Y(1:50)
CHARACTER :: CH
LOGICAL :: TF
```

- **Do loops :**

```
DO I = 0, 10
M(I) = I*I + 1
END DO
```

Array Features

F77

```
REAL A(50,50), B(50,50), C(50,50)
DO I =1, 50
    DO J = 1,50
    C(I,J) = A(I, J) +  B (I, J)
    END DO
END DO
```

F90

```
REAL, DIMENSION(50,50) :: A, B, C
C = A + B
----------------------------------------
REAL, DIMENSION(5,20) :: X, Y
REAL, DIMENSION(-2:2, 1:20) :: Z
!elementwise multiplication
Z = 4.0*Y * X
```

Array Allocations

Allocatable Array

(*creation and destruction are user-controlled*)

```
PROGRAM simulate

IMPLICIT NONE

INTEGER :: n

INTEGER, DIMENSION(:, :),  &
     ALLOCATABLE :: a

PRINT *, n

……

ALLOCATE( a (n, 2*n) )

……

DEALLOCATE ( a )

END
```

Automatic Arrays

(created on entry and destroyed on exit from procedure)

```
PROGRAM auto_array

INTEGER :: n,m

READ *, n,m

CALL sim(n,m)

END

SUBROUTINE sim(n,m)

REAL :: a(n,m), b(m)

…….

RETURN

END
```

Assumed-Shape Array

- **Assume shape of actual argument to which it is associated**

```
subroutine asshape( f, isign, indx)
!assumed-shape array
real, dimension( : ) :: f
integer isign, indx
…….
end subroutine asshape
```

- Assumed-shape arrays require an explicit interface (compiler verifies matching arguments)

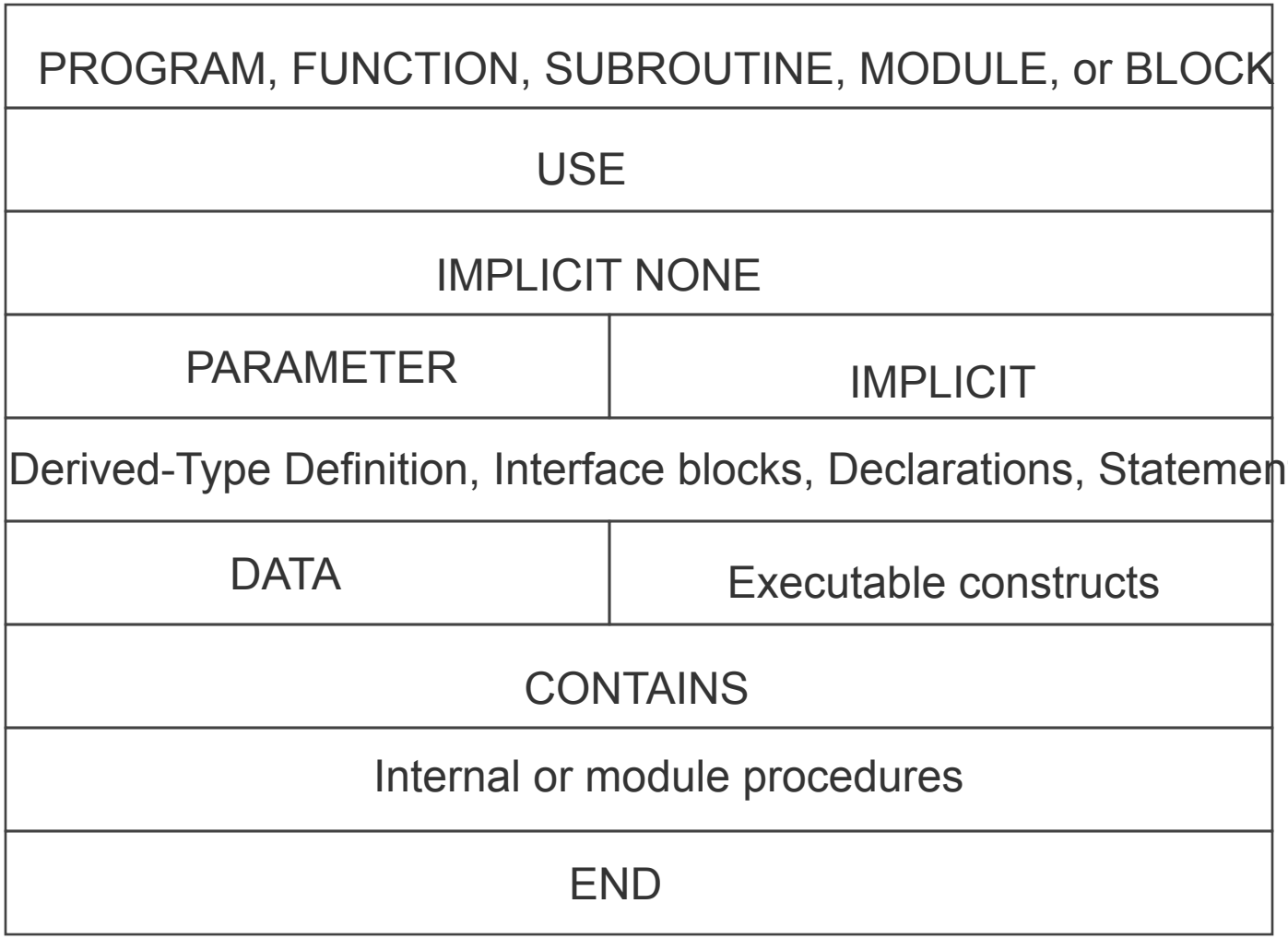
```
program main
integer, parameter :: isign=0,
    indx=10,  nx=2**indx
real, dimension(nx) :: f
interface
      subroutine asshape(a, j,k)
      real, dimension( : ) :: a
      integer :: j, k
end interface
call asshape(f, isign, indx)
end
```

Interface Blocks

- Interface blocks provide the compiler with all the information necessary to make consistency checks and ensure that enough information is communicated to the procedure at run-time.
- An interface declaration gives the characteristics (attributes) of both the dummy arguments (e.g. name, kind, type, and rank) and the procedure (e.g. name, class, and type).

```
Interface
        subroutine of function header
        declarations of dummy arguments
        end subroutine or function
end interface
```

Statement Ordering



Module and Use Statements

- Modules contain declarations, functions, and type definitions that can be conveniently accessed and used by executable program units
- Modules create interface blocks automatically. It is sometimes advantageous to package INTERFACE definitions into a module.

```
module test
……. declarations ….
  contains
          subroutine abc(x,y)
          ……
          end subroutine abc
end module test
```

- *Use* statement makes modules "available", like F77 COMMON and *INCLUDE*

Example : Module

```
module swapping
    contains
    subroutine swap (x, y)
    real, intent(inout) : : x, y
    tmp = x;  x= y  ; y = tmp
    end subroutine swap
end module swapping
program trymodule
    use swaping
    real : : a = 1.0 , b=2.0
    call swap(a, b)
end program trymodule
```

```
module Pye
!  save makes pi a global constant
!  acts like common
    real, save :: pi = 3.1415926
end module Pye

program Area
    use Pye
    implicit none
    real : : r
    read * , r
    print* , “ Area = “, pi *r *r
end program Area
```

Derived Types

- User defined type from intrinsic and previously defined types
- Various components can be unified by a derived type

```
type private_complex
        real :: real, imaginary
end type private_complex

type (private_complex) :: a, b, c
a%real = 1.0
b%imaginary = 2.0
c%real = a%real*b%real - a%imaginary*b%imaginary
c%imaginary = a%real*b%imaginary + a%imaginary*b%real
```

Encapsulation in Modules

- Grouping of data and operations into a single well-defined unit

```
module private_complex_module
type private_complex  ! define type
      real :: real, imaginary
end type private_complex
contains
    type (private_complex) function pc_mult(a,b)  ! function def.
    type (private_complex), intent (in) : : a, b
    pc_mult%real = a%real*b%real - a%imaginary*b%imaginary
    pc_mult%imaginary = a%real*b%imaginary +  a%imaginary*b%real
    end function pc_mult
end module private_complex_module
```

Encapsulation (Cont'd)

- A main program to multiply two private_complex numbers

```
program main
use private_complex_module  !  bring in the module
type (private_complex) : : a, b, c
                a%real = 1.0
                a%imaginary = -1.0
                b%real = -1.0
                b%imaginary = 2.0
                c = pc_mult(a, b)
                print *,  'c= ', c%real, c%imaginary
                stop
  end program main
```

FORTRAN 90 Pointers

- Pointer variables do not hold data, they point to scalar or array variables which themselves may contain data
- Target : the space to which a pointer variable points
- Pointer and target declarations :

```
real, pointer :: ptor
real, dimension(:,:), pointer :: ptoa
real, target : : x, y  ! may associate with ptor
real dimension(5,3), target ::a, b ! may associate with ptoa
x = 3.1416
ptor => y   ! pointer assignment (aliasing)
ptor = x  !  “normal” assignment ,  y = x
nullify(ptr)  ! disassociate pointer from y
ptoa => a(3:5:2, ::2)
```

Intrinsic Functions (F90)

- Array construction functions
  - SPREAD, PACK, RESHAPE,…
- Vector and matrix multiplication
  - DOT_PRODUCT, MATMUL
- Reduction functions
  - SUM, PRODUCT, COUNT, MAXCAL, ANY, ALL...
- Geometric location functions
  - MAXLOC, MINLOC
- Array manipulation functions
  - CSHIFT, EOSHIFT, TRANSPOSE…...

Examples (F90)

```
REAL :: a(100), b(4,100)
scalar = SUM(a)                ! sum of all elements
a = PRODUCT( b, DIM=1)  ! product of elements in first dim
scalar = COUNT ( a = = 0)   ! gives number of zero elements
scalar = MAXVAL ( a , MASK = a .LT. 0)
```

```
LOGICAL a(n)
REAL, DIMENSION(n) :: b, c
IF ( ALL(a)  )      ……          ! global AND
IF( ALL(b = = c)  ……          ! true if all elements equal
IF ( ANY(a)  )  ……            ! global OR
IF ( ANY( b < 0.0 ) ) …..     ! true if any elements < 0.0
```

Acknowledgements

- NSF, DOE, Cray, NCCS, NICS staff, students
- many helpful hands and collaborators
- Pictures are obtained from the web.