Best Practices in Parallel Computing

Peter Ruprecht
peter.ruprecht@colorado.edu

www.rc.colorado.edu

Slides: https://github.com/ResearchComputing/Parallelization_Workshop

Outline

- Hardware hierarchy: supercomputer to core
- Components of a compute node
- Interconnect
- Storage and filesystems
- Coprocessors
- Data parallelism (vectorization)
- Thread parallelism
- Multi-process parallelism
- High-Throughput Computing
- Operating system
- Getting logged in

Compute Hardware Architecture

Processing: Supercomputer – Node – Socket/CPU – Core

Memory: Distributed – RAM – L3 – L2 – L1

Interconnect network

Storage (disk)

Coprocessors

Non-cluster supercomputers (IBM Blue Gene, SGI UV)

Cloud?

Parallelism at all levels of the computing system is the name of the game today … SIMD, OpenMP, MPI

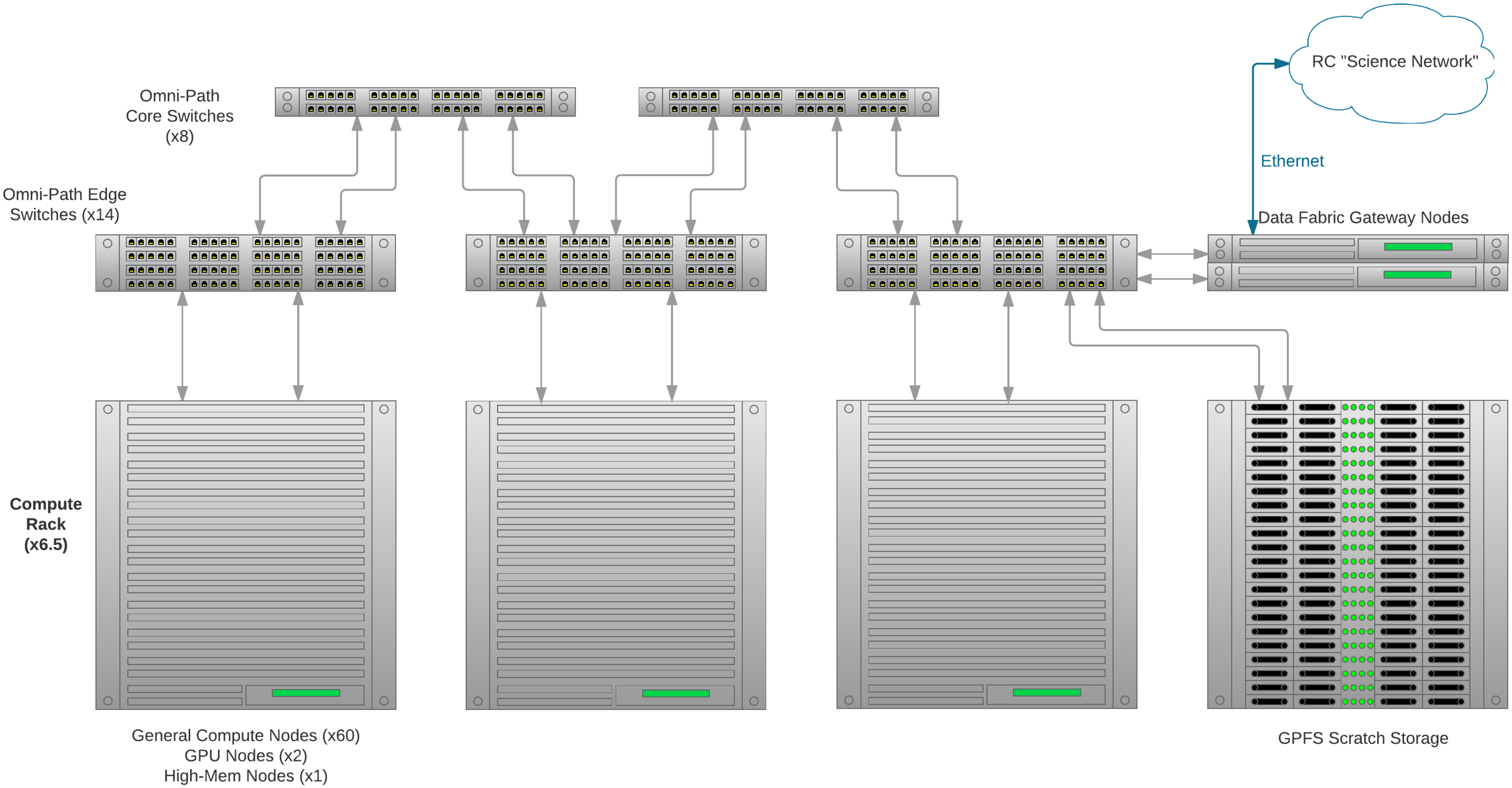
Clusters

Stampede (TACC)

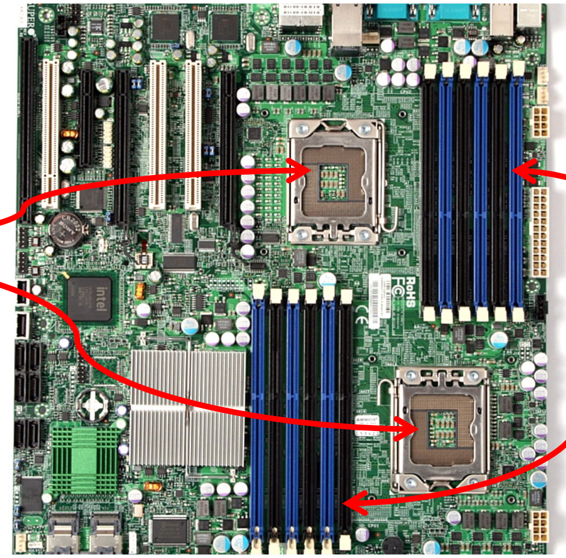
6400 nodes
- 2 sockets per node, 8 cores per socket

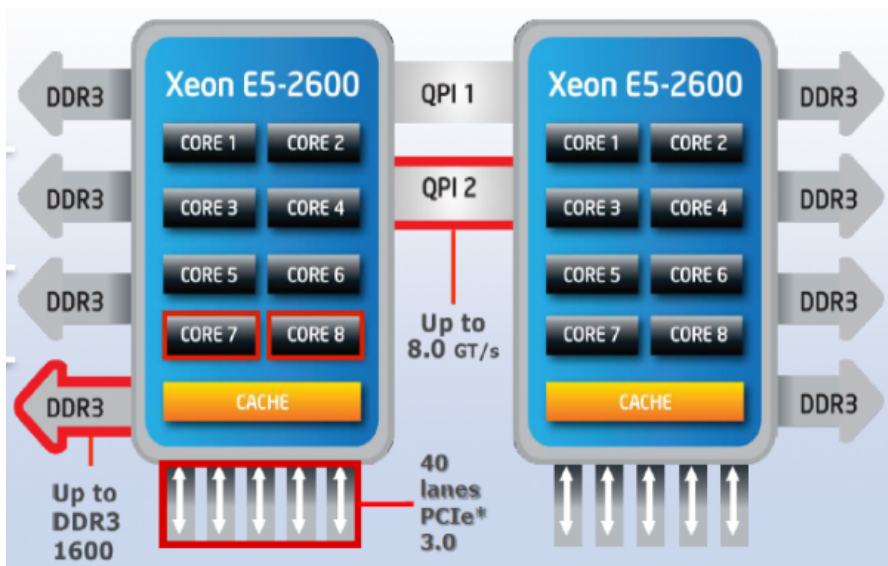
56 Gbps Infiniband interconnect
14 PB parallel storage system

SUMMIT SCHEMATIC



Nodes and Processors





Images: Supermicro, Intel

Host Processor Types

- x86_64 (Intel, AMD)
  - Most likely to encounter
  - General purpose
- Xeon Phi (Intel)
- POWER (IBM)
  - Blue Gene and future systems
- ARM
  - RISC, means fewer transistors
  - Low power

All processors operate at a certain frequency (several GHz) and most can perform several instructions per cycle.

Computer Bus Layout

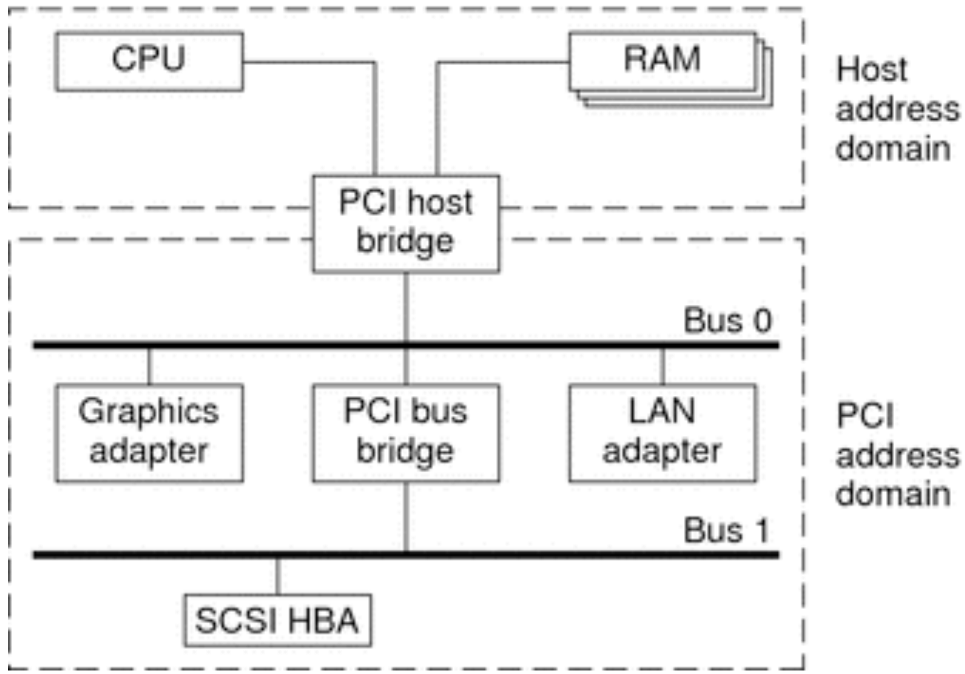


Image: Oracle

Memory

- Holds data that is being calculated on, as well as the running program itself
- Closest memory to CPU is actually on the processor die
  - Cache: Level 1 and 2 are dedicated to a single core
  - Level 3 is larger and shared between all cores in a socket
- Next closest is RAM
- Can also access memory on other nodes, via RDMA (remote direct memory access)

Shared memory is local to one node and several process threads can share the same data addresses.

Distributed memory can be on multiple nodes and each process normally has its own copy or part of the data.

Interconnect

- How to access distributed memory across nodes?
- Need a fast, *low-latency* network.
- Current interconnect technologies
  - InfiniBand (40, 56, 100 Gbps)
  - Ethernet (10, 40, 100 Gbps)
  - Aries/Gemini (Cray)
  - OmniPath (100 Gbps)

- RDMA normally requires special interconnect-aware libraries; frequently these are included with MPI

Interconnect is what makes a bunch of nodes into a supercomputer!

Storage

Different types of “disk” for different needs

- Local disk in the node, often SSD
- Shared scratch
  - Often accessed over cluster interconnect
  - Parallel filesystems, eg Lustre or GPFS
  - Traditionally tuned for high bandwidth, not high IOPS
  - May have a “burst buffer” layer in front of it
  - Short-term storage only!!
- Longer-term or archive
  - Often uses Hierarchical Storage Management

CPU pipeline

Optimizing for Data Access

- Page Fault, file on IDE disk: 1,000,000,000 cycles
- Page Fault, file in buffer cache: 10,000 cycles
- Page Fault, file on ram disk: 5,000 cycles
- Page Fault, zero page: 3,000 cycles
- Main memory access: about 200 cycles
- L3 cache hit: about 52 cycles
- L1 cache hit: 2 cycles

The Core i7 can issue 4 instructions per cycle. So a penalty of 2 cycles for L1 memory access means a missed opportunity for 7 instructions.

Co-Processors

Specialized, usually massively multi-core, separate from host processor (CPU).

- GPU (Graphics Processing Unit, “video card”)
  - Thousands of cores
  - Great for vectorizable or embarrassingly-parallel apps
  - Programming frameworks include CUDA, OpenACC, OpenCL
  - Many applications (eg, Matlab) support CUDA directly
- Xeon Phi (being phased out)
  - Dozens of cores
  - Existing x86 code can be directly recompiled to run on Phi
  - Function offload

Different Node Types

- Login nodes
    - Four virtual machines
    - This is where you are when you log in
    - No heavy computation, interactive jobs, or long running processes
    - Script or code editing, minor compiling
    - Job submission
- Compile nodes
    - Where you compile code
- Compute/batch nodes
    - This is where jobs that are submitted through the scheduler run
    - Intended for heavy computation

Data Storage Spaces

- **Home Directory**
  - /home/$USER
  - Not for direct computation
  - Small quota (2 GB)
  - Backed up
- **Project Directory**
  - /projects/$USER
  - Mid level quota (250 GB)
  - Large file storage
  - Backed up

- **Scratch Directory**
  - /scratch/summit/$USER
  - 10 TB quota
    - Can ask for more if needed
  - Files purged after 90 days

Interlude – logging in

Username/password on printed strips

Username is `user00XY`

Login node is `tutorial-login.rc.colorado.edu`

```
ssh user00XY@tutorial-login.rc.colorado.edu
```

```
git clone
https://github.com/ResearchComputing/Parallel
ization_Workshop
```

Parallelism

- Data parallelism (vectorization, "Single Instruction on Multiple Data" or SIMD)
- Thread parallelism
- Multi-processing
- High-throughput computing

Vectorization

Data register:

data 1	data 2	data 3	data 4

Consider this loop:

```
for (i=1 ; i<=N ; i++)
    b[i] = 3 + a[i];
```

(inspired by Intel Autovectorization Guide)

Without vectorization:

3 + a[1]	not used	not used	not used

With vectorization:

3 + a[1]	3 + a[2]	3 + a[3]	3 + a[4]

Multi-threading

- Process: A program that is executing on a computer (uses memory that is separate from other processes)
- Thread: A sequence of instructions within a process

- A process can have many threads executing at once
- Each thread can run on a separate CPU core

Multi-threading, continued

- Multi-threaded processes usually use *shared memory*
- Thus they can't span across more than one node

- OpenMP is an example of a threaded programming framework
- It is enabled via compiler directives, library functions, and environment variables

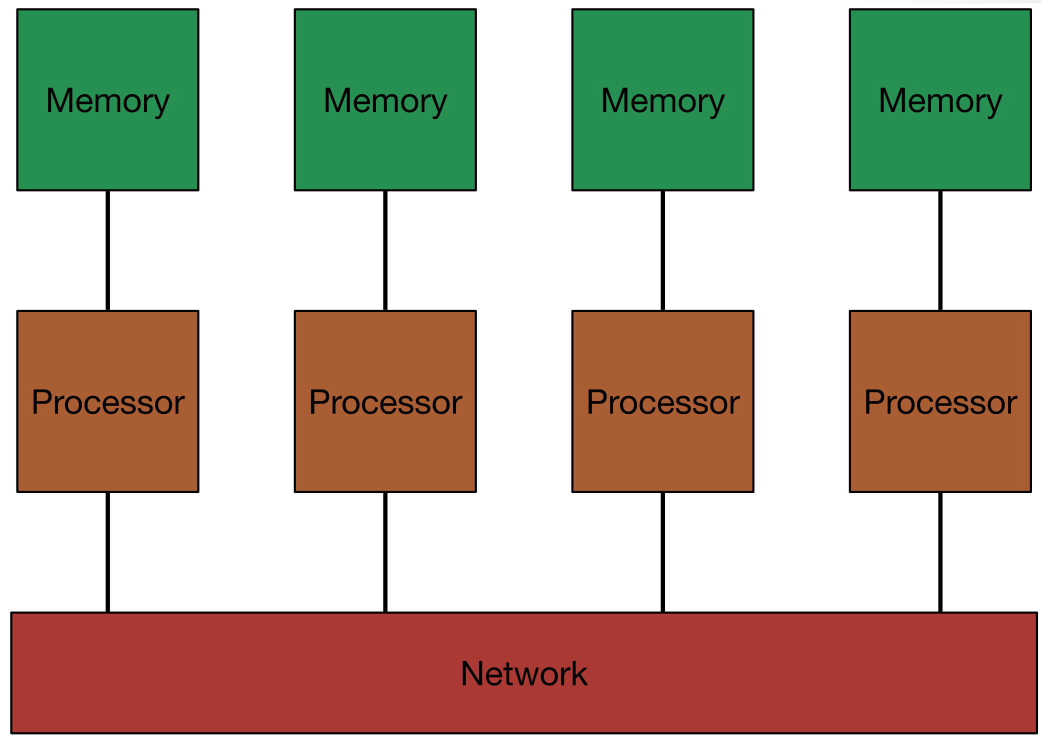
Multi-processing

- One program can start multiple processes that communicate with each other
- Each process has its own memory: *distributed memory*
- A multi-process application can run on one or more nodes

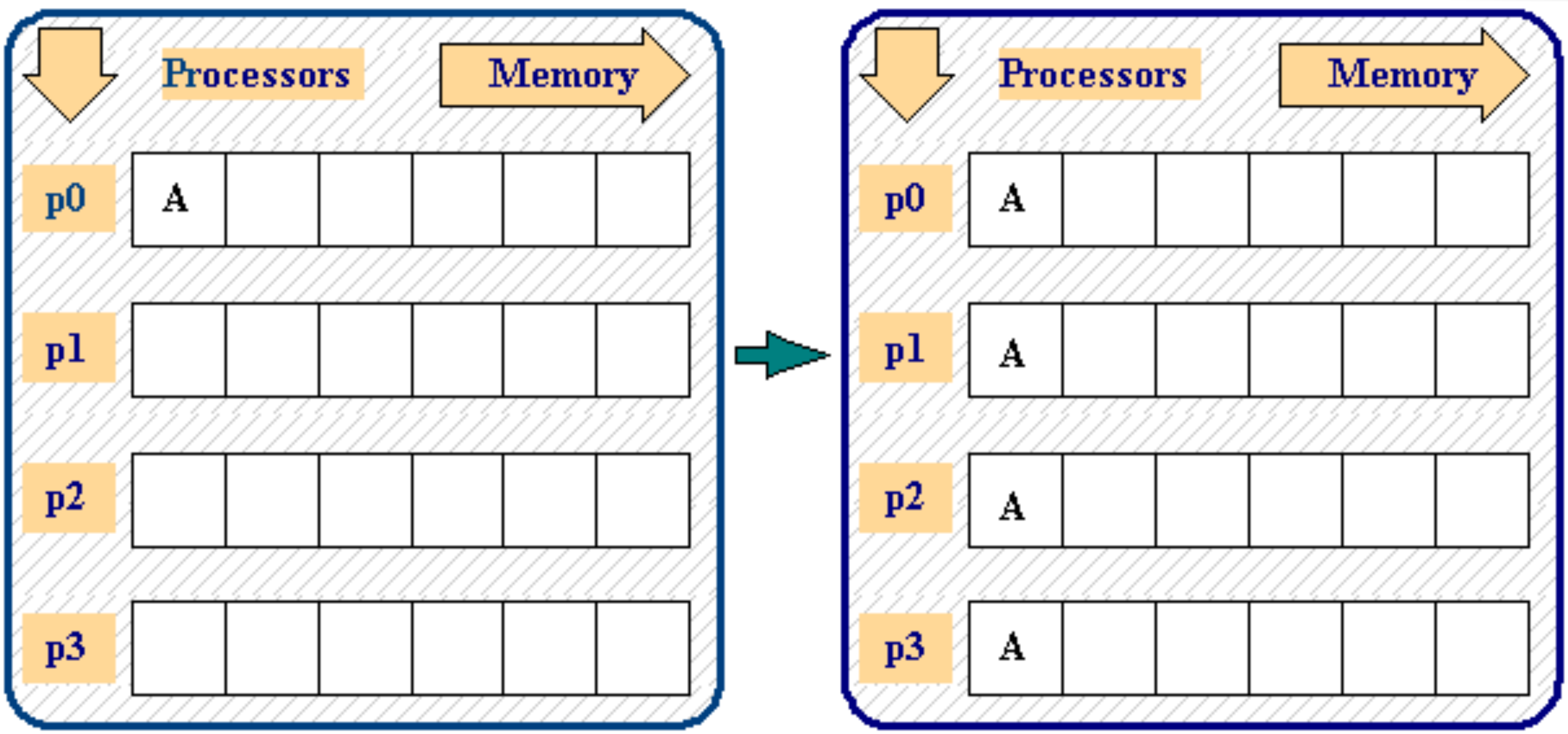
- MPI (Message Passing Interface) is an example of a threaded programming framework
- It is enabled via compiler directives, library functions, and MPI commands

Distributed Memory Computer

- Processors have different content in memory
- Data exchange by message passing



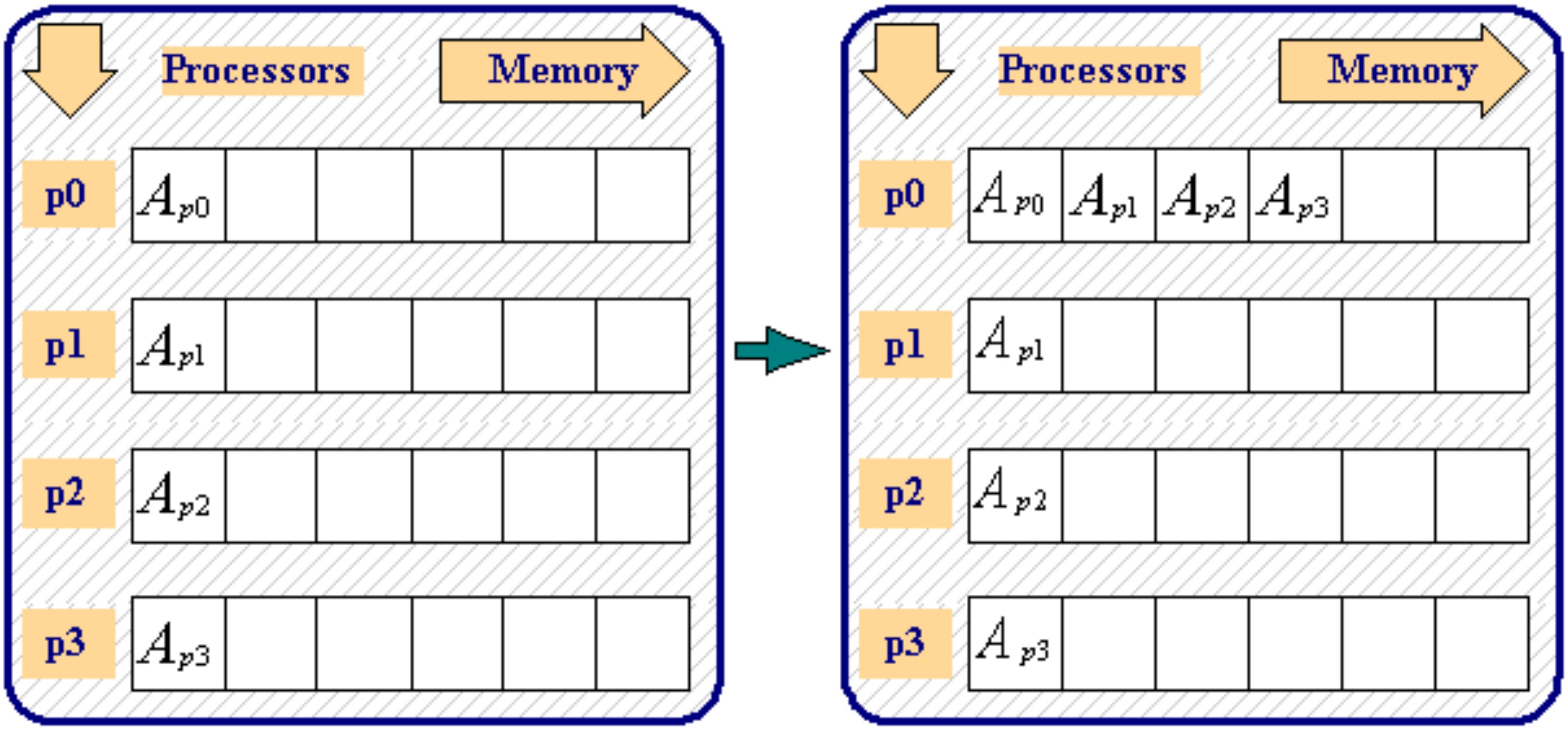
Broadcast




```
send_count = 1;
  root = 0;
  MPI_Bcast ( &a, send_count, MPI_INT, root, comm )
```

Figure from MPI-tutor: http://www.citutor.org/index.php

Gather



send_count = 1;
recv_count = 1;
recv_rank = 0;
MPI_Gather (&a, send_count, MPI_REAL, &a, recv_count, MPI_REAL, recv_rank, MPI_COMM_WORLD);

Figure from MPI-tutor: http://www.citutor.org/index.php

I/O Optimization

- Total Job Time = Computational Time
  + Communication Time + I/O time
- Thus, optimizing and parallelizing I/O can be important
- Parallel access requires special programming constructs or libraries, such as MPI-IO
  - MPI-IO consists of MPI functions for data reading and writing
  - Each MPI process can read or write a portion of a shared file
- Structured data formats such as HDF5 can help a lot

More on I/O Optimization

- Put no more than 10K files in a single directory
  - Use a structure of subdirectories for holding >10K files
- Avoid metadata intensive operations like `ls -l`
- When programming:
  - If a file only needs to be read, open it read-only
  - Do not open and close files too frequently
  - Assign a subset of cores for handing I/O
  - Give MPI-IO “hints” about how your I/O operations should be tuned

High-throughput Computing

- High-performance computing (HPC): a single computer provides substantial power to a single job

- HTC: many smaller computations run across many computers (or nodes)
- Often these jobs are independent of each other
- Possibly over a relatively long period of time

- Open Science Grid ; job arrays ; “load balancing” ; …

Operating Systems in HPC

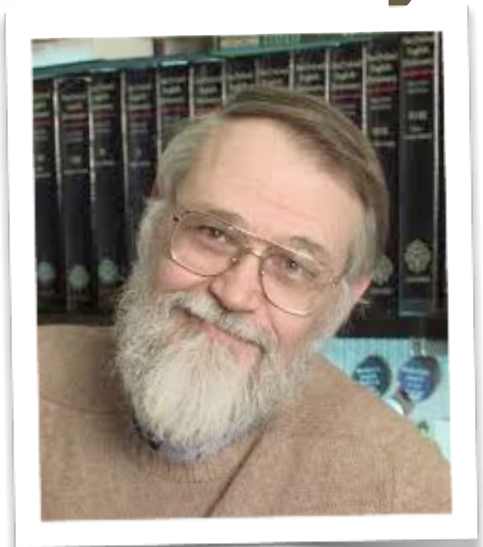
Operating system: software that manages computer hardware and the processes running on it.

- MS Windows – hasn't really gotten a foothold
- Mac OSX – even less of a factor
- Unix / Linux – some variation will be on virtually any HPC system you use
  - May be a stripped-down version optimized for the particular hardware, or
  - May be a full distribution

What is Linux?

- Part of the Unix family of operating systems.
- Started in early ‘90s by Linus Torvalds.
- Technically refers only to the kernel; software from the GNU project and elsewhere is layered on top to form a complete OS. Most is open source.
- Several full distributions are available – from enterprise-grade, like RHEL or SUSE, to more consumer-focused, like Ubuntu.
- Runs on everything from embedded systems to supercomputers.

History of Linux



Brian Kernighan
1970
"space travel" to Unix

Dennis Ritchie
1971
C

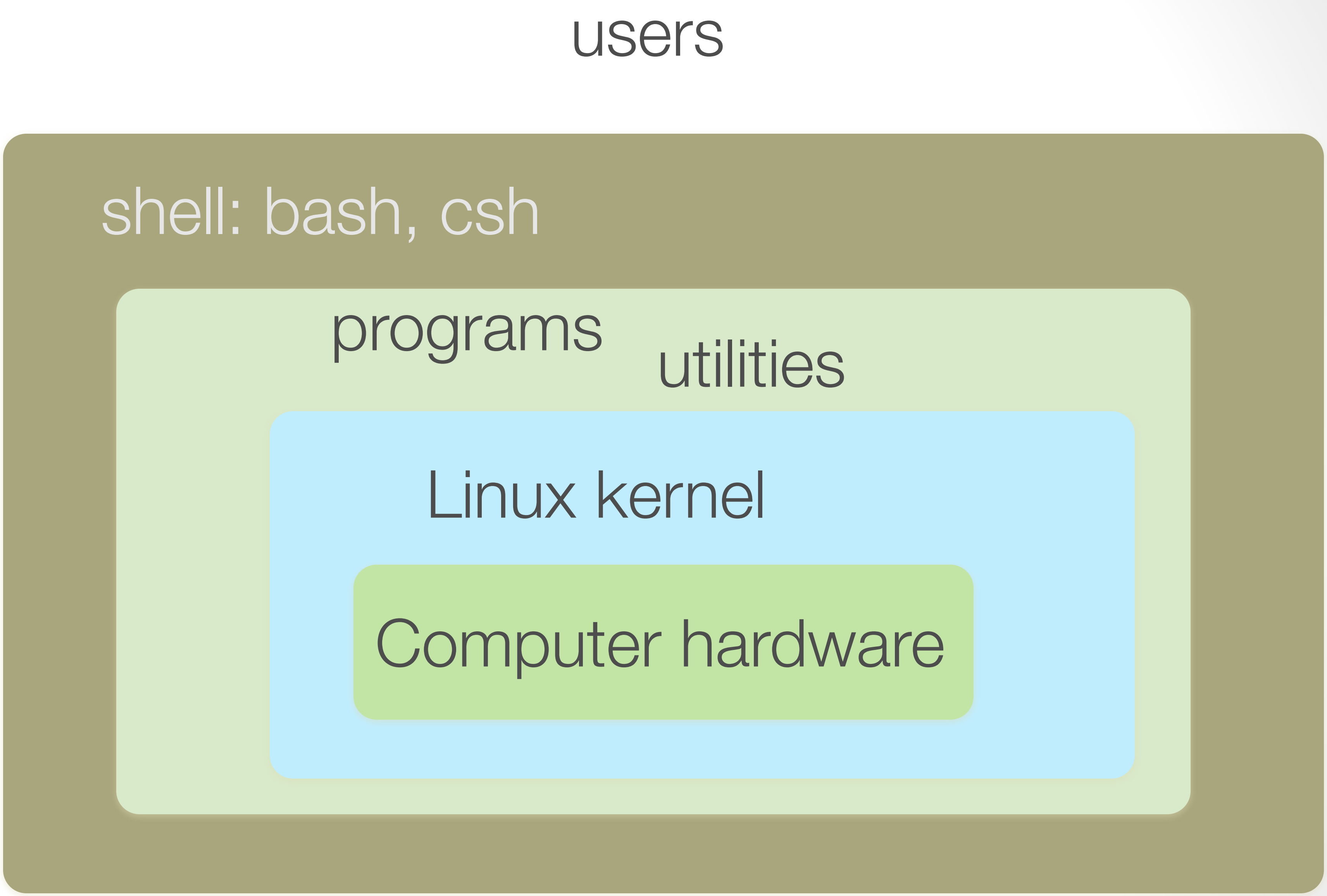
Richard Stallman
1983
Gnu Not Unix

Linus Torvalds
1991
Linux kernel for personal computers

Why use Linux?

- Linux command-line syntax may seem overwhelming to the new user, but:
- It's the default operating system on virtually all HPC systems
- It's extremely flexible
- It tries not to get in your way
- It's fast and powerful
- Open-source scientific applications are developed predominantly for Linux

- You can get started with a few basic commands and build from there


users
shell: bash, csh
programs
utilities
Linux kernel
Computer hardware

References

1. Eijkhout, Victor. *Introduction to High-Performance Scientific Computing*, 2015. https://bitbucket.org/VictorEijkhout/hpc-book-and-course/src
2. Hager, G, and G Wellein. *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.
3. Levesque, John, and Gene Wagenbreth. *High Performance Computing*. CRC Press, 2010.
4. Neeman, Henry. *Supercomputing in Plain English*, High Performance Computing Workshop Series, http://www.oscer.ou.edu/education.php

Thank you!

- Email rc-help@colorado.edu
- Twitter: CUBoulderRC

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

- Slides:
  https://github.com/ResearchComputing/Parallelization_Workshop