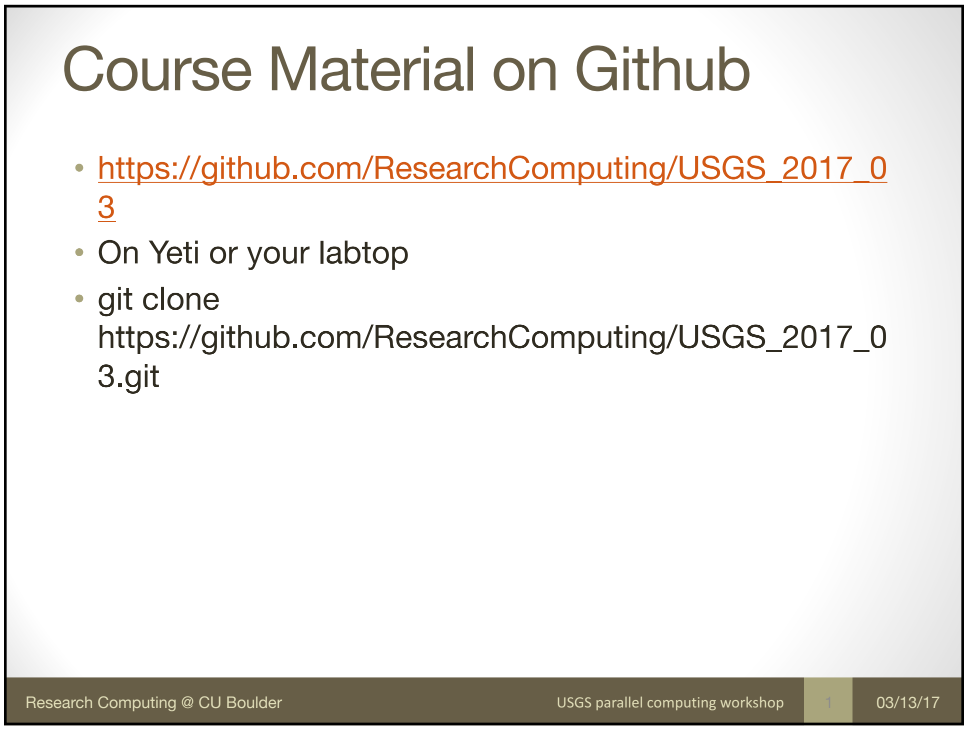
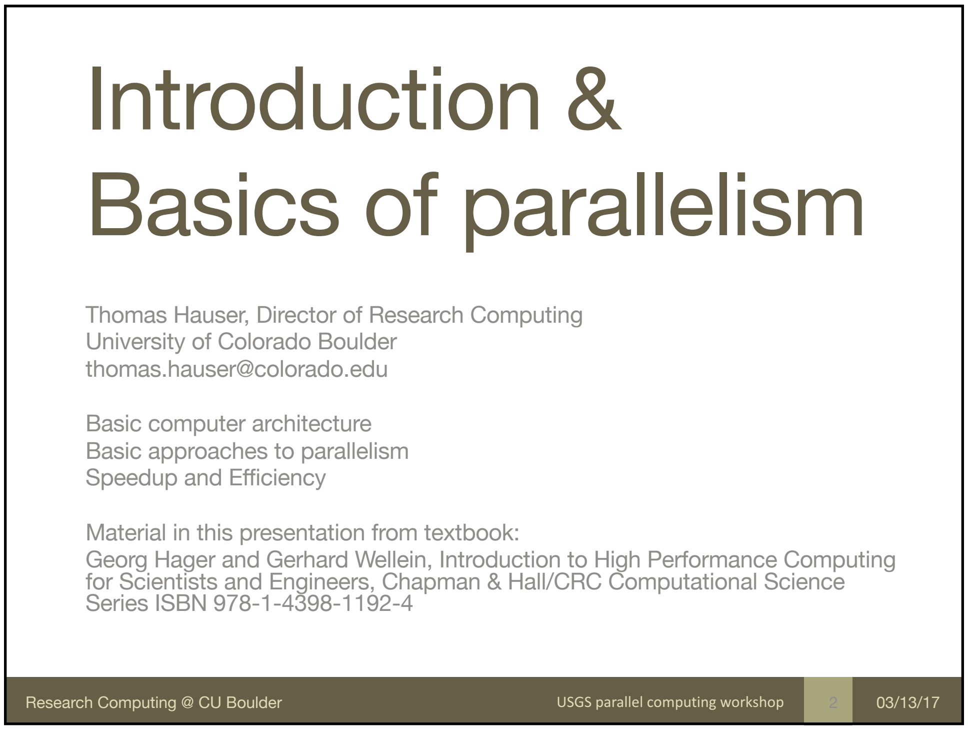


Course Material on Github

- https://github.com/ResearchComputing/USGS_2017_03
- On Yeti or your labtop
- git clone https://github.com/ResearchComputing/USGS_2017_03.git

Research Computing @ CU Boulder USGS parallel computing workshop 1 03/13/17



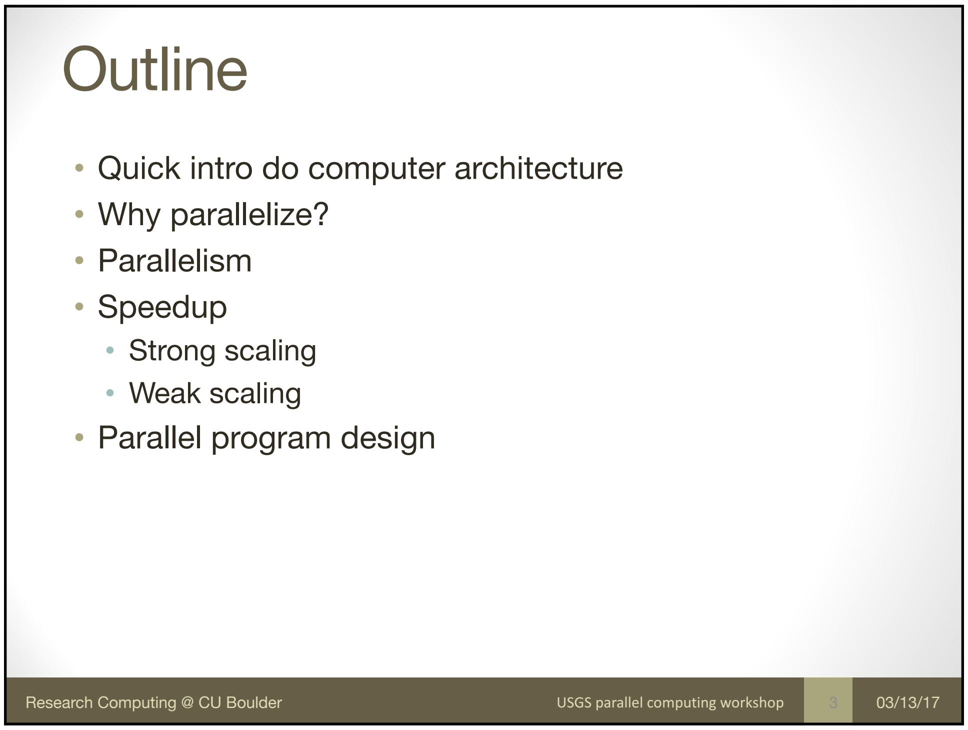
Introduction & Basics of parallelism

Thomas Hauser, Director of Research Computing
University of Colorado Boulder
thomas.hauser@colorado.edu

Basic computer architecture
Basic approaches to parallelism
Speedup and Efficiency

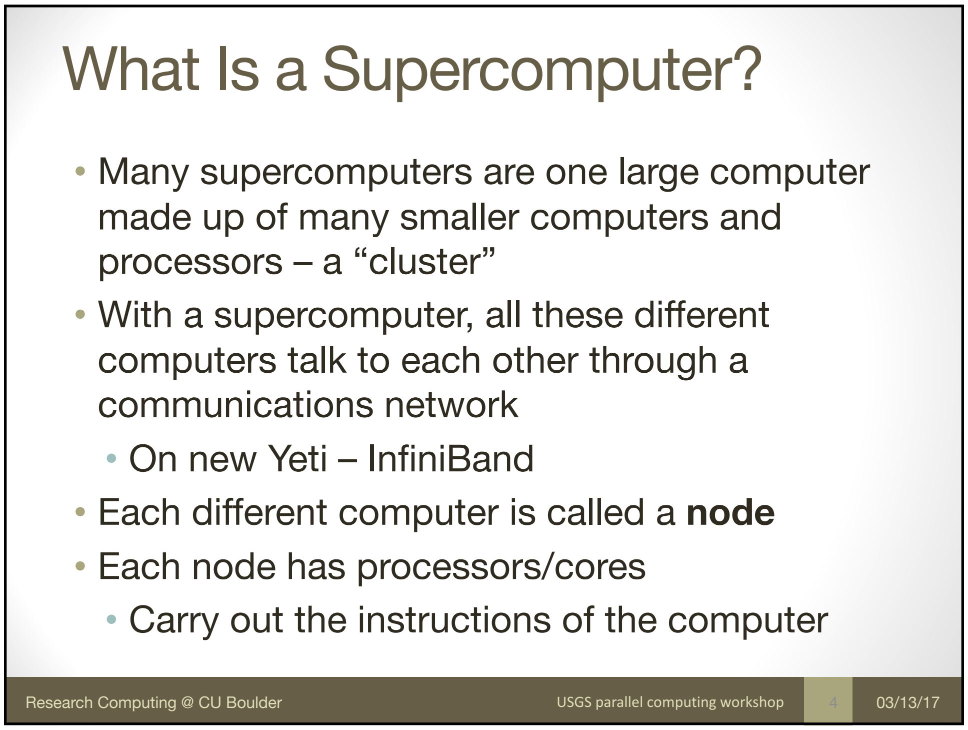
Material in this presentation from textbook:
Georg Hager and Gerhard Wellein, Introduction to High Performance Computing for Scientists and Engineers, Chapman & Hall/CRC Computational Science Series ISBN 978-1-4398-1192-4

Research Computing @ CU Boulder USGS parallel computing workshop 2 03/13/17



Outline

- Quick intro do computer architecture
- Why parallelize?
- Parallelism
- Speedup
  - Strong scaling
  - Weak scaling
- Parallel program design



What Is a Supercomputer?

- Many supercomputers are one large computer made up of many smaller computers and processors – a “cluster”
- With a supercomputer, all these different computers talk to each other through a communications network
  - On new Yeti – InfiniBand
- Each different computer is called a **node**
- Each node has processors/cores
  - Carry out the instructions of the computer

Why Use a Supercomputer?

- Supercomputers give you the opportunity to solve problems that are too complex for the desktop
  - Might take hours, days, weeks, months, years
  - If you use a supercomputer, might only take minutes, hours, days, or weeks
- Useful for problems that require large amounts of memory

Computers and Cars - Analogy

≈

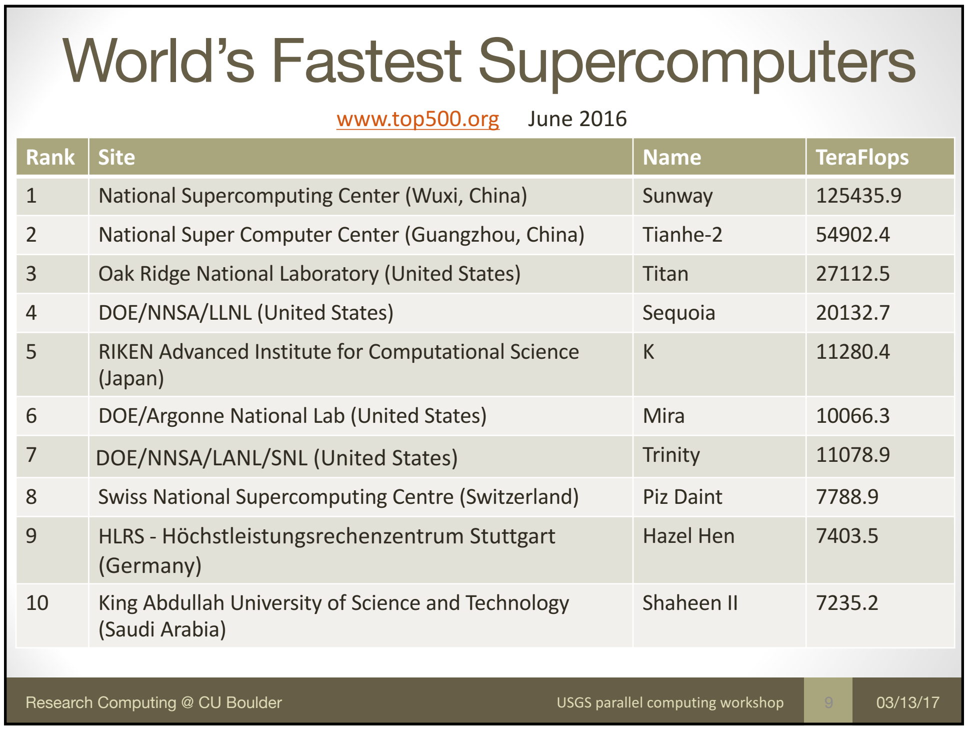
Computers and Cars - Analogy

≈

Image from cray.com

Why Use a Supercomputer?

- Supercomputers give you the opportunity to solve problems that are too complex for the desktop
  - Might take hours, days, weeks, months, years
  - If you use a supercomputer, might only take minutes, hours, days, or weeks
- Useful for problems that require large amounts of memory

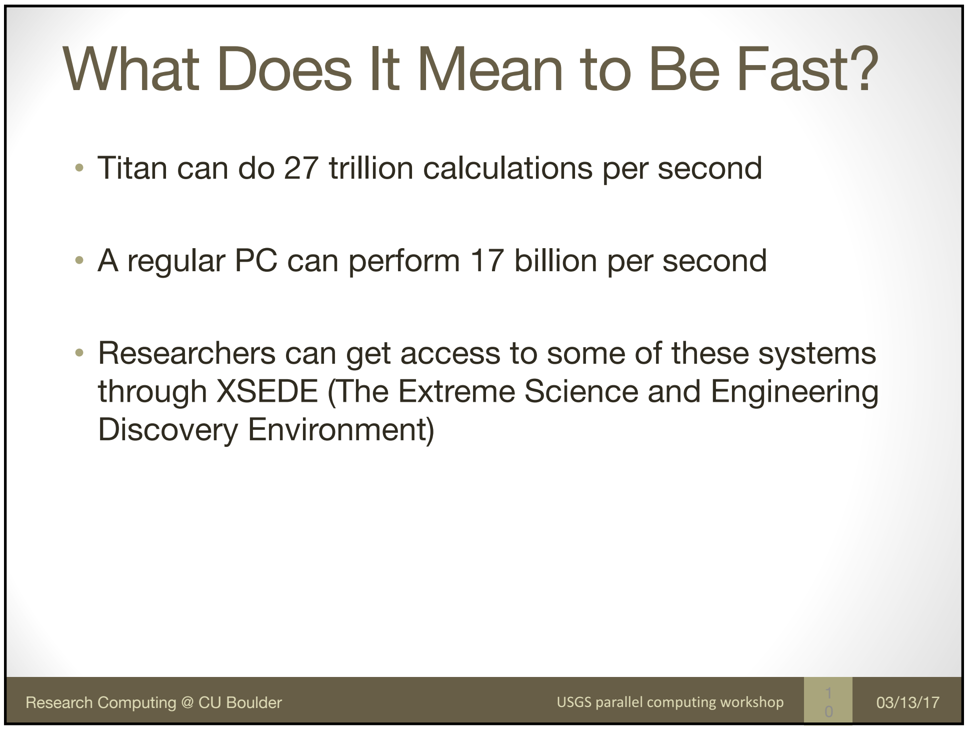


World's Fastest Supercomputers

www.top500.org June 2016

Rank	Site	Name	TeraFlops
1	National Supercomputing Center (Wuxi, China)	Sunway	125435.9
2	National Super Computer Center (Guangzhou, China)	Tianhe-2	54902.4
3	Oak Ridge National Laboratory (United States)	Titan	27112.5
4	DOE/NNSA/LLNL (United States)	Sequoia	20132.7
5	RIKEN Advanced Institute for Computational Science (Japan)	K	11280.4
6	DOE/Argonne National Lab (United States)	Mira	10066.3
7	DOE/NNSA/LANL/SNL (United States)	Trinity	11078.9
8	Swiss National Supercomputing Centre (Switzerland)	Piz Daint	7788.9
9	HLRS - Höchstleistungsrechenzentrum Stuttgart (Germany)	Hazel Hen	7403.5
10	King Abdullah University of Science and Technology (Saudi Arabia)	Shaheen II	7235.2

Research Computing @ CU Boulder USGS parallel computing workshop 9 03/13/17



What Does It Mean to Be Fast?

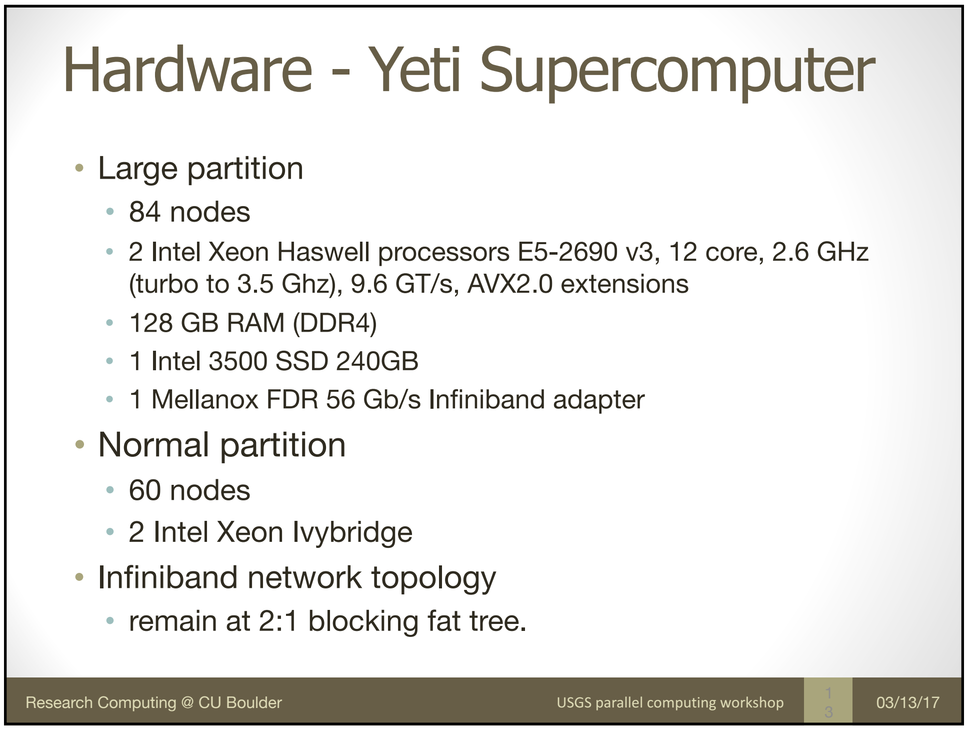
- Titan can do 27 trillion calculations per second
- A regular PC can perform 17 billion per second
- Researchers can get access to some of these systems through XSEDE (The Extreme Science and Engineering Discovery Environment)

Research Computing @ CU Boulder USGS parallel computing workshop 10 03/13/17

Supercomputer Details

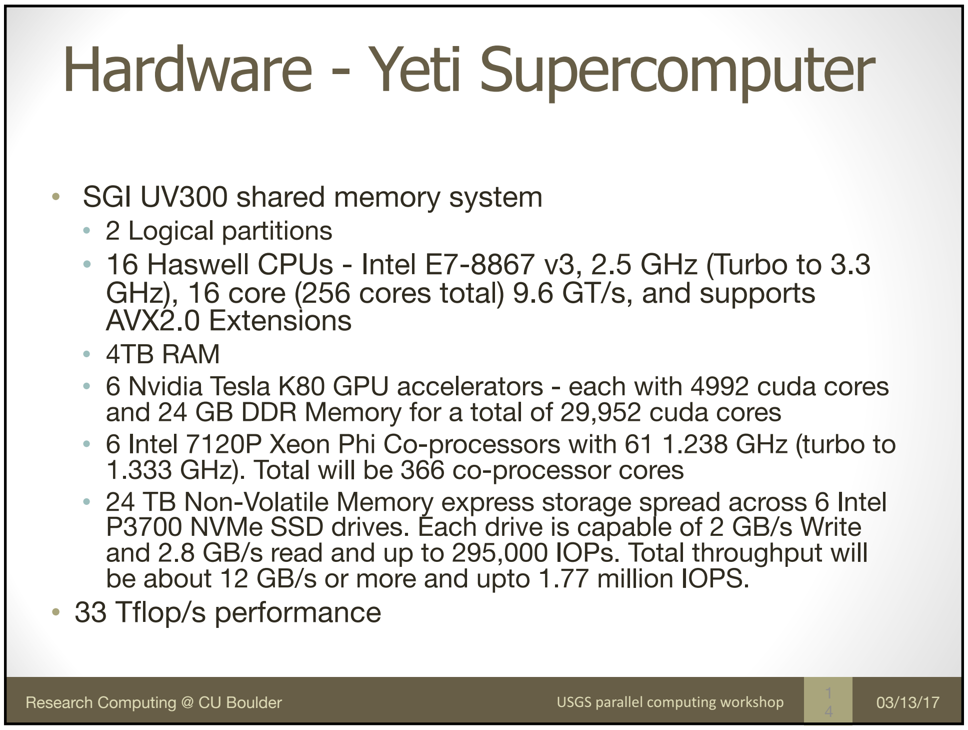
Hardware - Yeti Supercomputer

- 3728 CPU Cores
- 40 Gb/s Infiniband Node Interconnect
- 2.75 GB/s Lustre Throughput
- 6.2 GB/s CXFS Throughput
- 488 TB Configured Storage (1.01 PB Raw)
- 4 Partitions
  - Normal (Distributed Memory)
    - Large – minimum 240 cores
    - Long – 30 days (300 cores total)
  - UV (Shared Memory + GPUs)
    - 3 nodes
      - Shared memory
      - GPUs
      - Intel Phis
- 6 Racks
- Red Hat Enterprise Linux/Scientific Linux 6.7



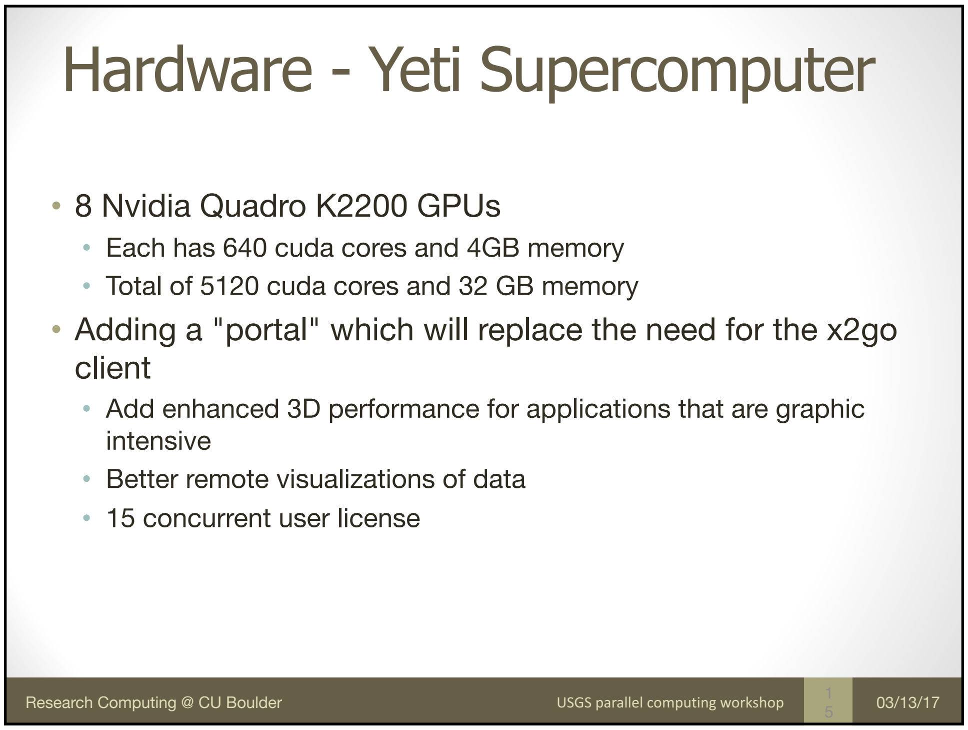
Hardware - Yeti Supercomputer

- Large partition
  - 84 nodes
  - 2 Intel Xeon Haswell processors E5-2690 v3, 12 core, 2.6 GHz (turbo to 3.5 Ghz), 9.6 GT/s, AVX2.0 extensions
  - 128 GB RAM (DDR4)
  - 1 Intel 3500 SSD 240GB
  - 1 Mellanox FDR 56 Gb/s Infiniband adapter
- Normal partition
  - 60 nodes
  - 2 Intel Xeon Ivybridge
- Infiniband network topology
  - remain at 2:1 blocking fat tree.



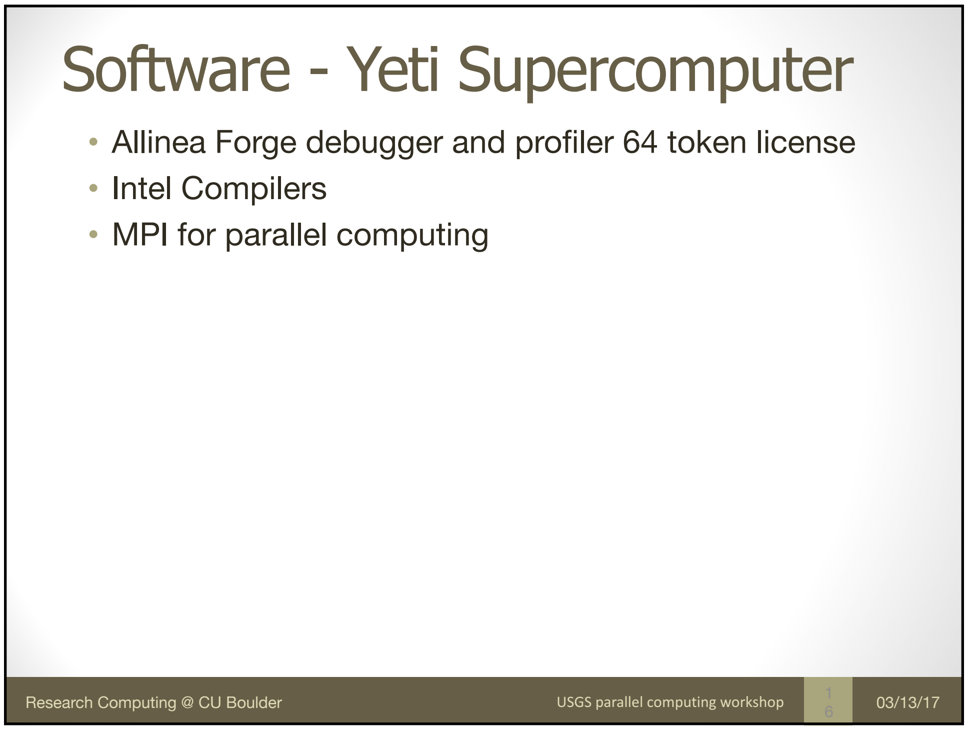
Hardware - Yeti Supercomputer

- SGI UV300 shared memory system
  - 2 Logical partitions
  - 16 Haswell CPUs - Intel E7-8867 v3, 2.5 GHz (Turbo to 3.3 GHz), 16 core (256 cores total) 9.6 GT/s, and supports AVX2.0 Extensions
  - 4TB RAM
  - 6 Nvidia Tesla K80 GPU accelerators - each with 4992 cuda cores and 24 GB DDR Memory for a total of 29,952 cuda cores
  - 6 Intel 7120P Xeon Phi Co-processors with 61 1.238 GHz (turbo to 1.333 GHz). Total will be 366 co-processor cores
  - 24 TB Non-Volatile Memory express storage spread across 6 Intel P3700 NVMe SSD drives. Each drive is capable of 2 GB/s Write and 2.8 GB/s read and up to 295,000 IOPs. Total throughput will be about 12 GB/s or more and upto 1.77 million IOPS.
- 33 Tflop/s performance



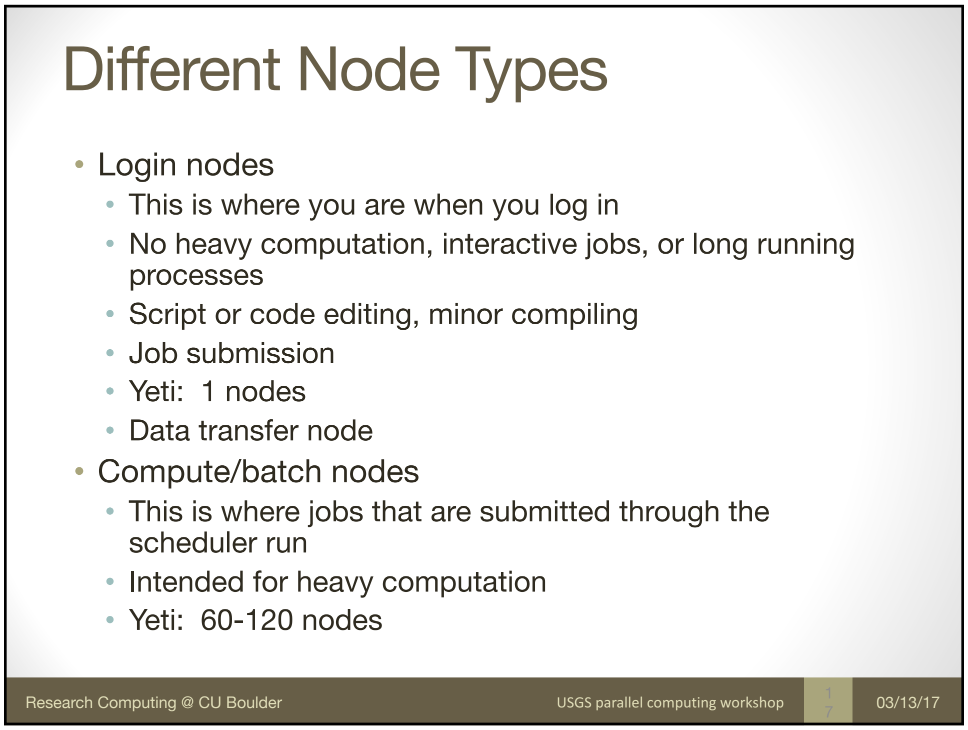
Hardware - Yeti Supercomputer

- 8 Nvidia Quadro K2200 GPUs
  - Each has 640 cuda cores and 4GB memory
  - Total of 5120 cuda cores and 32 GB memory
- Adding a "portal" which will replace the need for the x2go client
  - Add enhanced 3D performance for applications that are graphic intensive
  - Better remote visualizations of data
  - 15 concurrent user license



Software - Yeti Supercomputer

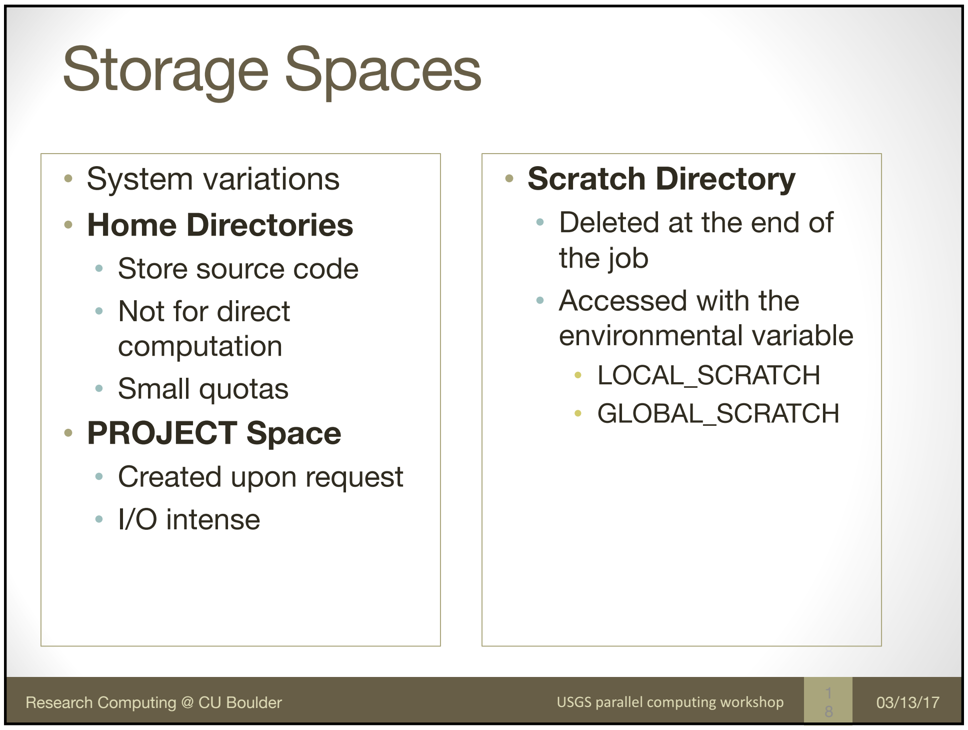
- Allinea Forge debugger and profiler 64 token license
- Intel Compilers
- MPI for parallel computing



Different Node Types

- Login nodes
  - 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
  - Yeti: 1 nodes
  - Data transfer node
- Compute/batch nodes
  - This is where jobs that are submitted through the scheduler run
  - Intended for heavy computation
  - Yeti: 60-120 nodes

Research Computing @ CU Boulder USGS parallel computing workshop 17 03/13/17

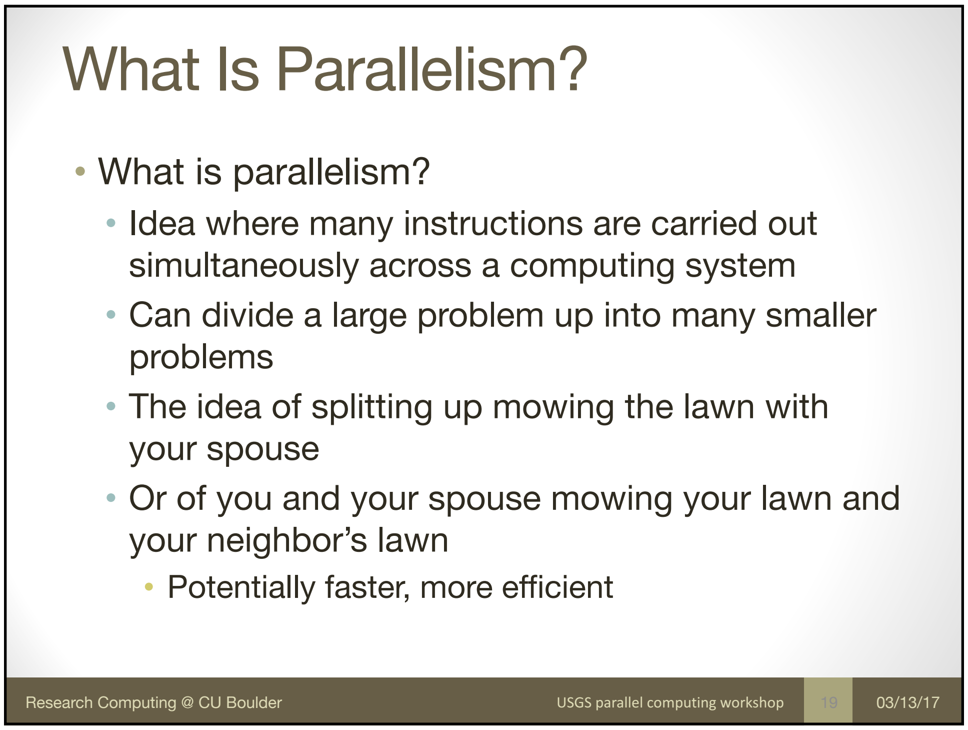


Storage Spaces

- System variations
- **Home Directories**
  - Store source code
  - Not for direct computation
  - Small quotas
- **PROJECT Space**
  - Created upon request
  - I/O intense

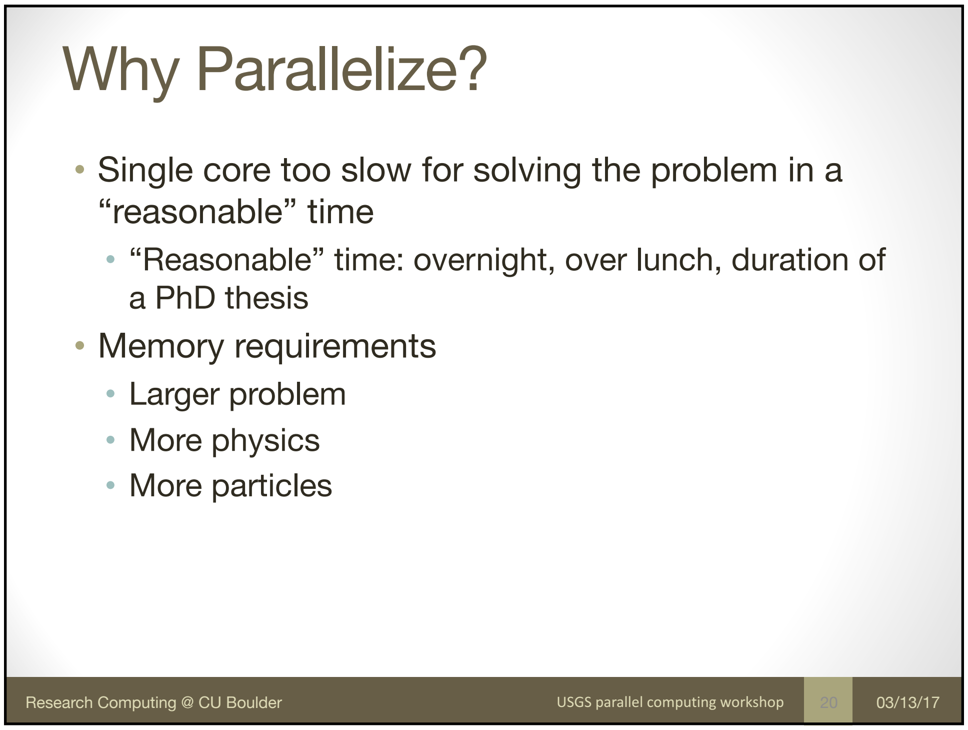
- **Scratch Directory**
  - Deleted at the end of the job
  - Accessed with the environmental variable
    - LOCAL_SCRATCH
    - GLOBAL_SCRATCH

Research Computing @ CU Boulder USGS parallel computing workshop 18 03/13/17



What Is Parallelism?

- What is parallelism?
  - Idea where many instructions are carried out simultaneously across a computing system
  - Can divide a large problem up into many smaller problems
  - The idea of splitting up mowing the lawn with your spouse
  - Or of you and your spouse mowing your lawn and your neighbor's lawn
    - Potentially faster, more efficient



Why Parallelize?

- Single core too slow for solving the problem in a "reasonable" time
  - "Reasonable" time: overnight, over lunch, duration of a PhD thesis
- Memory requirements
  - Larger problem
  - More physics
  - More particles

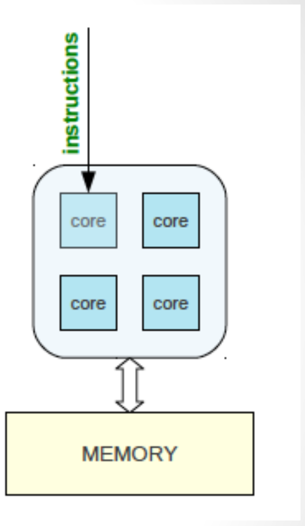
Serial Processing – Thought Experiment

- Jigsaw puzzle analogy**
- Have a 1000 piece jigsaw puzzle
  - You can do it yourself, maybe it will take 1 hour to do
  - Serial processing
- Maybe you have three friends sitting nearby willing to help, but you won't let them
  - Wasted resources

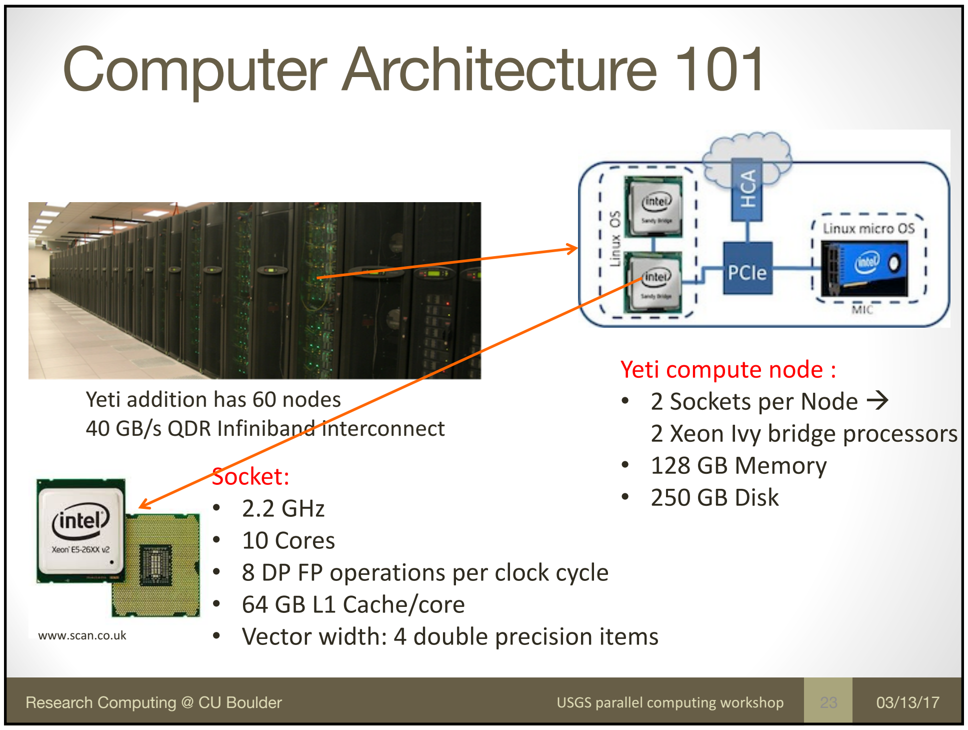
**from Henry Neeman, OSCER, "Supercomputing in Plain English"

Serial Processing

- Instructions are executed on one core
- The other cores sit idle
- If a task is running, Task 2 waits for Task 1 to complete, etc.
- Wasting resources
- Want to instead parallelize and use all cores

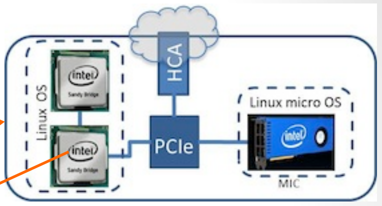


Source: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf



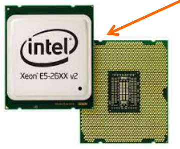
Computer Architecture 101

Yeti addition has 60 nodes
40 GB/s QDR Infiniband interconnect



Yeti compute node :

- 2 Sockets per Node →
  2 Xeon Ivy bridge processors
- 128 GB Memory
- 250 GB Disk

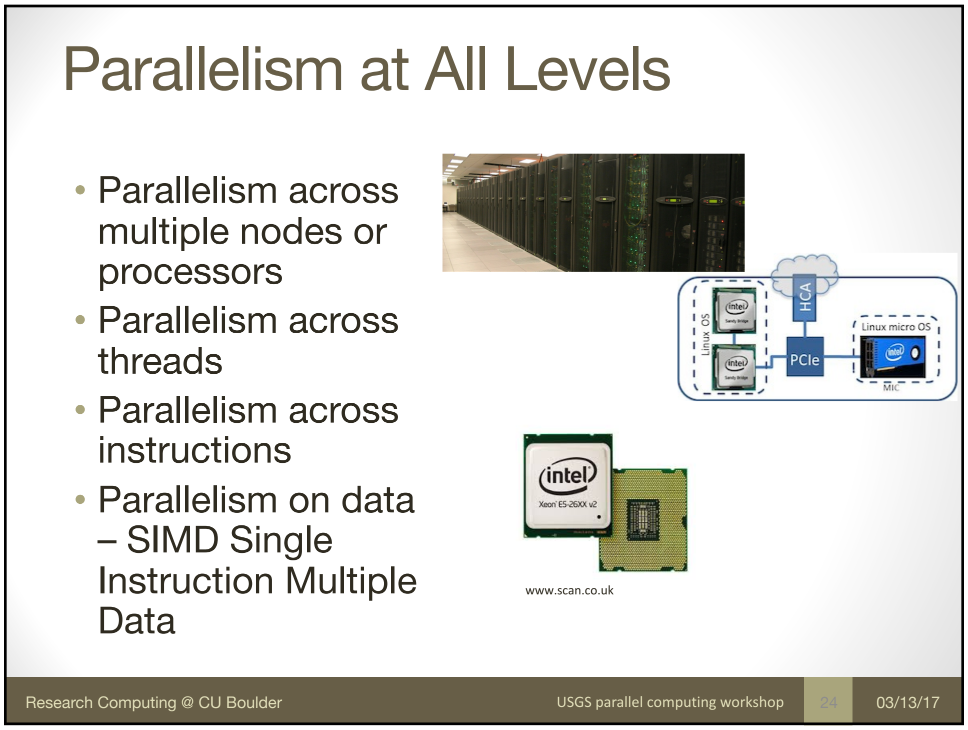


www.scan.co.uk

Socket:

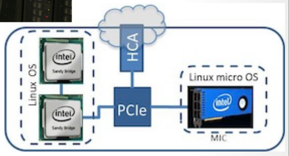
- 2.2 GHz
- 10 Cores
- 8 DP FP operations per clock cycle
- 64 GB L1 Cache/core
- Vector width: 4 double precision items

Research Computing @ CU Boulder USGS parallel computing workshop 23 03/13/17



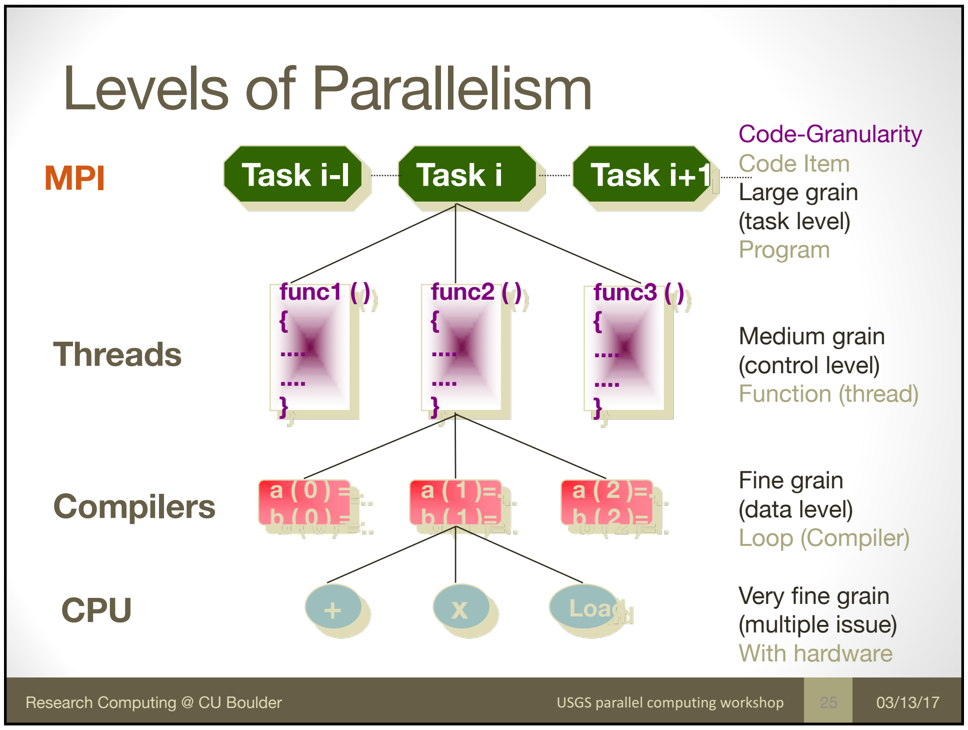
Parallelism at All Levels

- Parallelism across multiple nodes or processors
- Parallelism across threads
- Parallelism across instructions
- Parallelism on data – SIMD Single Instruction Multiple Data

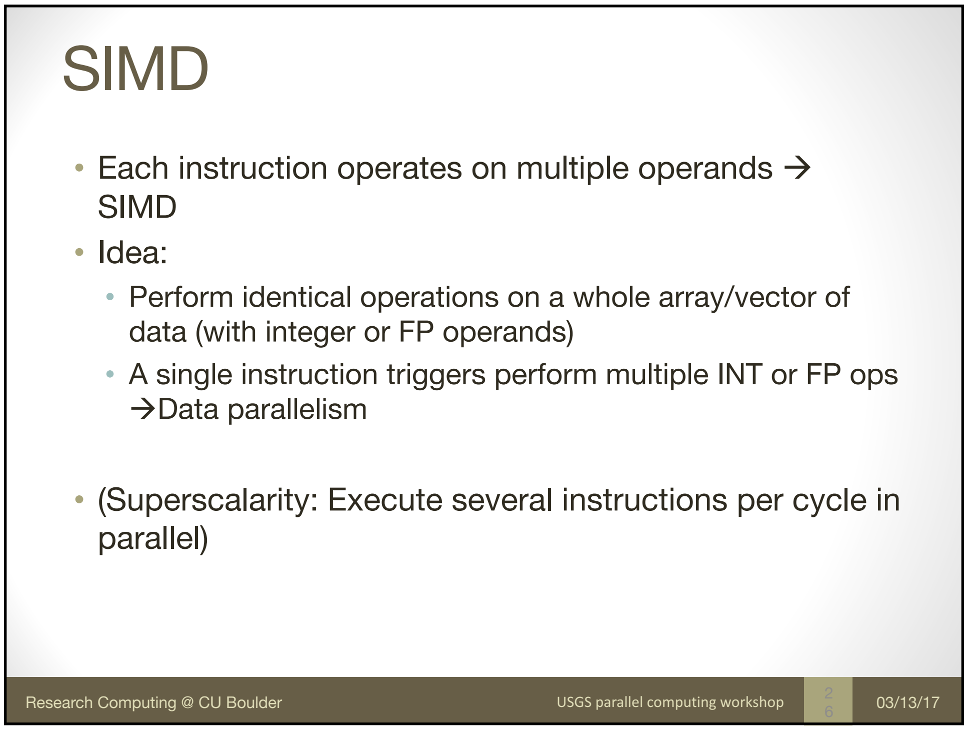


www.scan.co.uk

Research Computing @ CU Boulder USGS parallel computing workshop 24 03/13/17

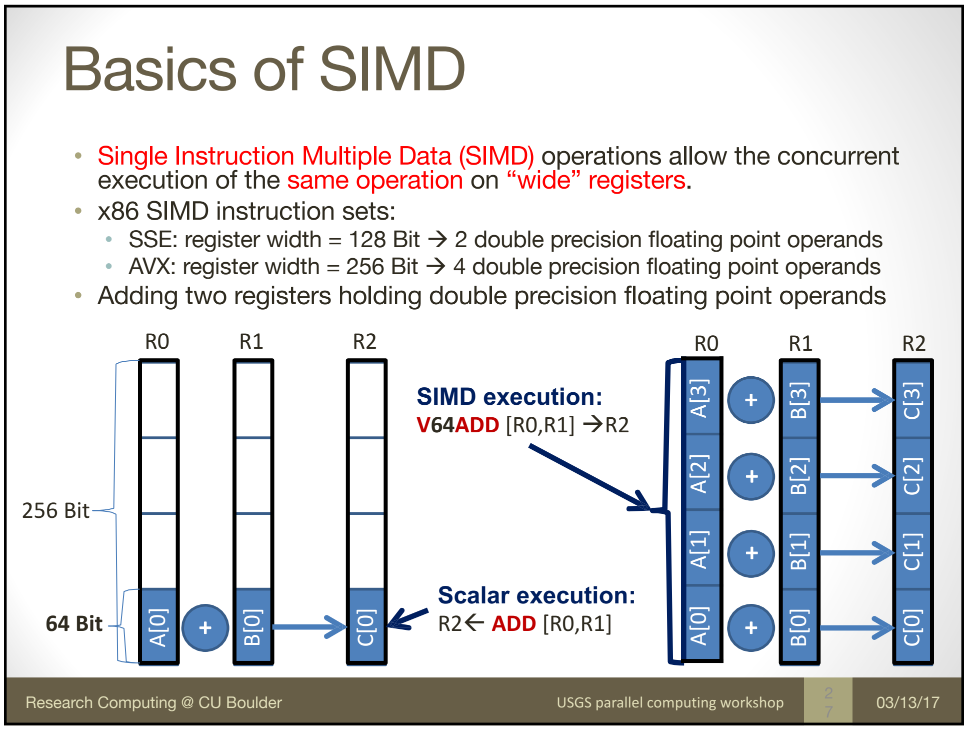


Levels of Parallelism



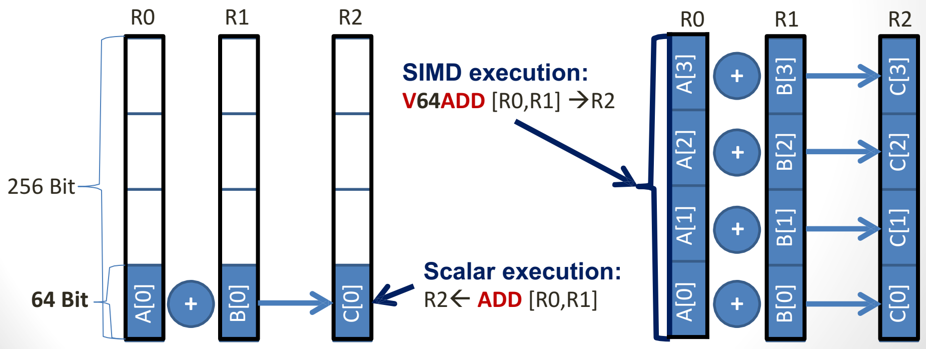
SIMD

- Each instruction operates on multiple operands → SIMD
- Idea:
  - Perform identical operations on a whole array/vector of data (with integer or FP operands)
  - A single instruction triggers perform multiple INT or FP ops →Data parallelism
- (Superscalarity: Execute several instructions per cycle in parallel)

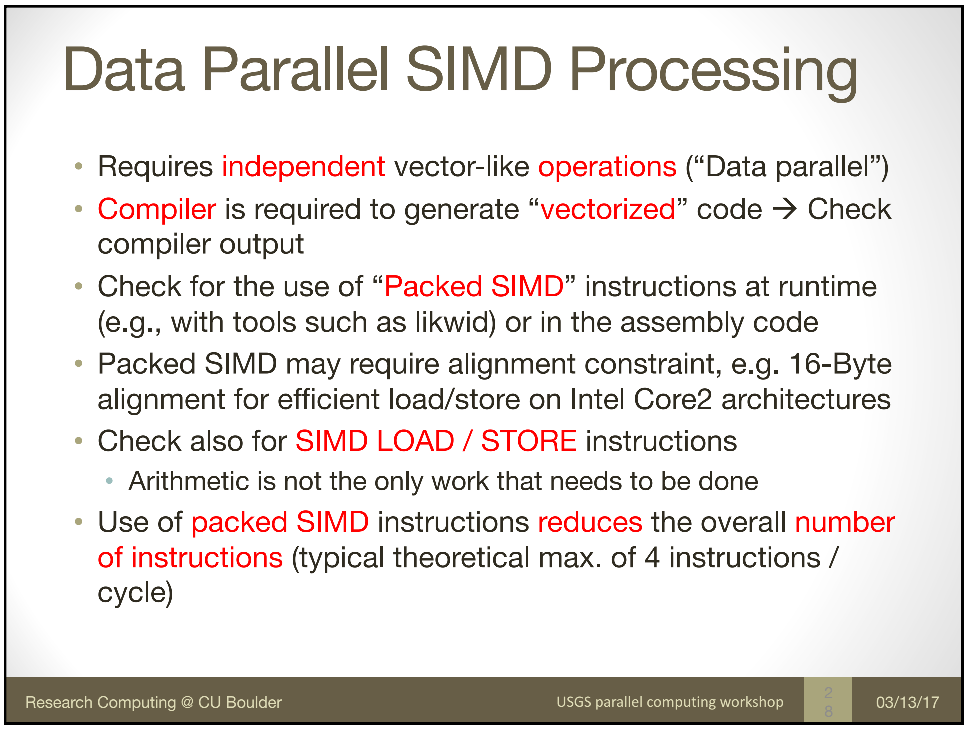


Basics of SIMD

- Single Instruction Multiple Data (SIMD) operations allow the concurrent execution of the same operation on “wide” registers.
- x86 SIMD instruction sets:
  - SSE: register width = 128 Bit → 2 double precision floating point operands
  - AVX: register width = 256 Bit → 4 double precision floating point operands
- Adding two registers holding double precision floating point operands



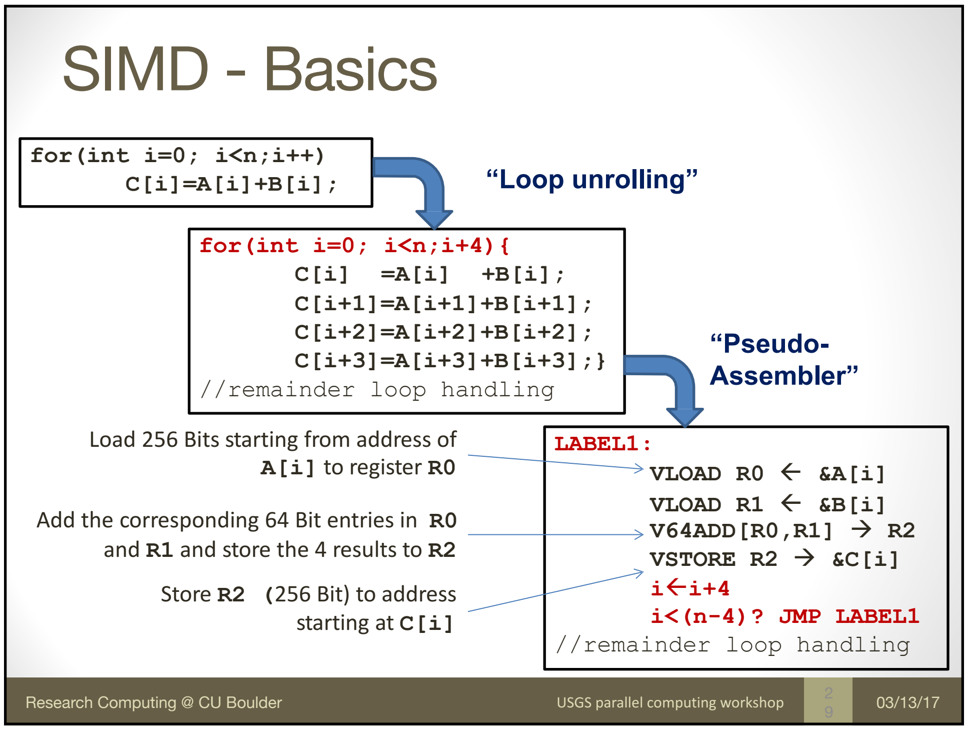
Research Computing @ CU Boulder USGS parallel computing workshop 27 03/13/17



Data Parallel SIMD Processing

- Requires independent vector-like operations (“Data parallel”)
- Compiler is required to generate “vectorized” code → Check compiler output
- Check for the use of “Packed SIMD” instructions at runtime (e.g., with tools such as likwid) or in the assembly code
- Packed SIMD may require alignment constraint, e.g. 16-Byte alignment for efficient load/store on Intel Core2 architectures
- Check also for SIMD LOAD / STORE instructions
  - Arithmetic is not the only work that needs to be done
- Use of packed SIMD instructions reduces the overall number of instructions (typical theoretical max. of 4 instructions / cycle)

Research Computing @ CU Boulder USGS parallel computing workshop 28 03/13/17



SIMD - Basics

```
for(int i=0; i<n;i++)
        C[i]=A[i]+B[i];
```

“Loop unrolling”

```
for(int i=0; i<n;i+4){
        C[i]  =A[i]  +B[i];
        C[i+1]=A[i+1]+B[i+1];
        C[i+2]=A[i+2]+B[i+2];
        C[i+3]=A[i+3]+B[i+3];}
//remainder loop handling
```

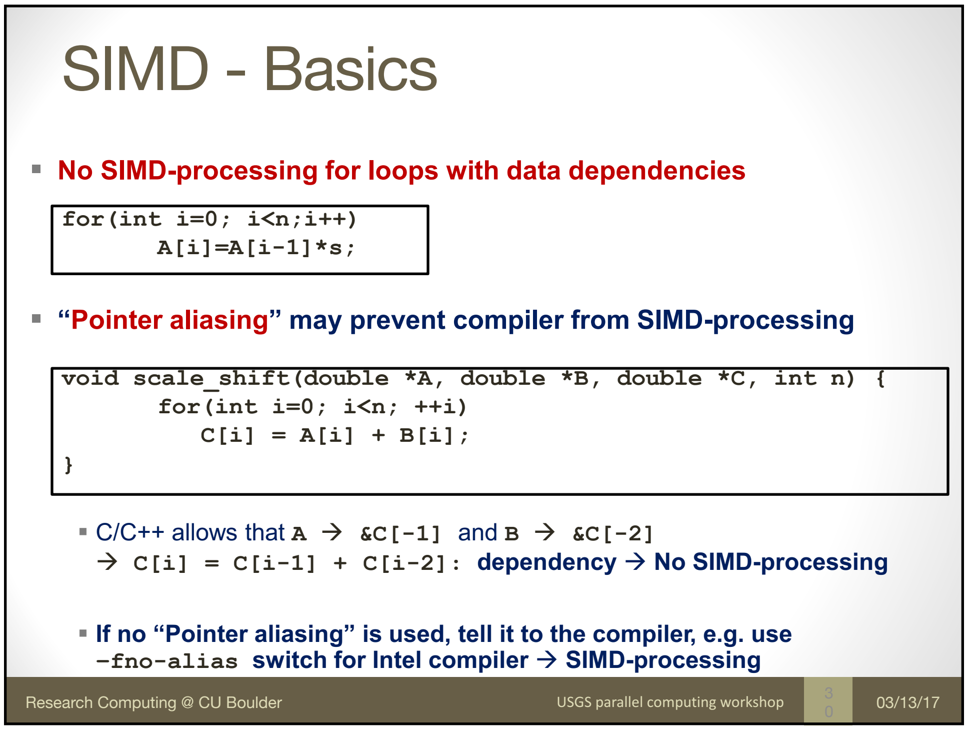
“Pseudo-Assembler”

```
LABEL1:
        VLOAD R0 ← &A[i]
        VLOAD R1 ← &B[i]
        V64ADD[R0,R1] → R2
        VSTORE R2 → &C[i]
        i←i+4
        i<(n-4)? JMP LABEL1
//remainder loop handling
```

Load 256 Bits starting from address of **A[i]** to register **R0**

Add the corresponding 64 Bit entries in **R0** and **R1** and store the 4 results to **R2**

Store **R2** (256 Bit) to address starting at **C[i]**



SIMD - Basics

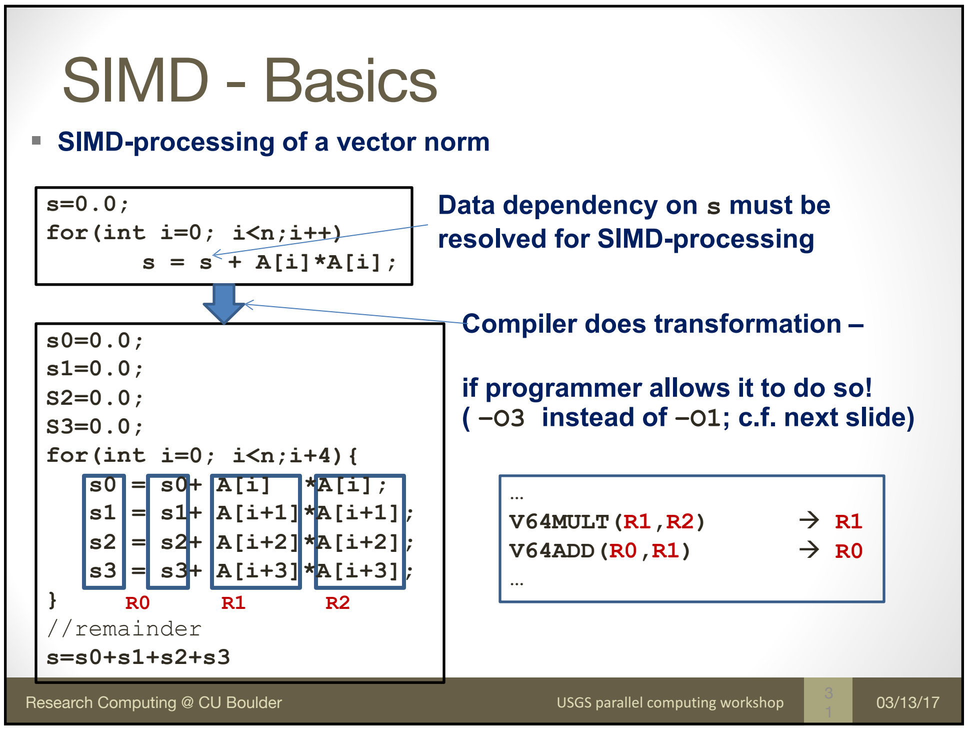
- **No SIMD-processing for loops with data dependencies**

```
for(int i=0; i<n;i++)
        A[i]=A[i-1]*s;
```

- **“Pointer aliasing” may prevent compiler from SIMD-processing**

```
void scale_shift(double *A, double *B, double *C, int n) {
        for(int i=0; i<n; ++i)
            C[i] = A[i] + B[i];
}
```

  - C/C++ allows that **A** → **&C[-1]** and **B** → **&C[-2]** → **C[i] = C[i-1] + C[i-2]**: **dependency → No SIMD-processing**
  - **If no “Pointer aliasing” is used, tell it to the compiler, e.g. use -fno-alias switch for Intel compiler → SIMD-processing**



SIMD - Basics

- **SIMD-processing of a vector norm**

```
s=0.0;
for(int i=0; i<n;i++)
        s = s + A[i]*A[i];
```

Data dependency on s must be resolved for SIMD-processing

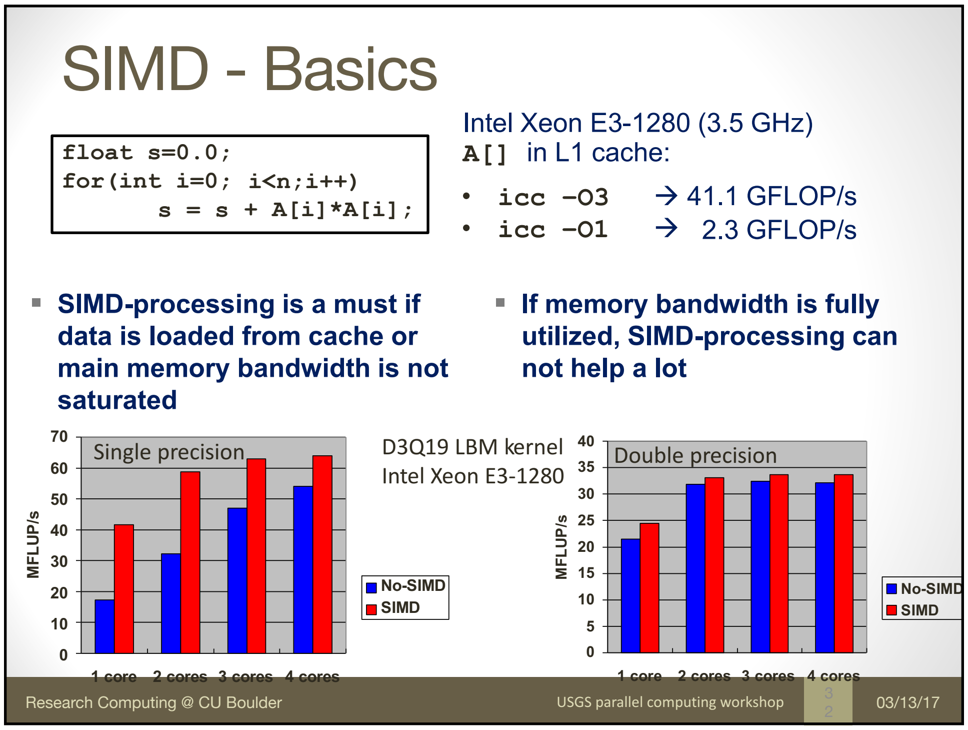
Compiler does transformation –

if programmer allows it to do so! (-O3 instead of -O1; c.f. next slide)

```
s0=0.0;
s1=0.0;
S2=0.0;
S3=0.0;
for(int i=0; i<n;i+4){
   s0 = s0+ A[i]  *A[i];
   s1 = s1+ A[i+1]*A[i+1];
   s2 = s2+ A[i+2]*A[i+2];
   s3 = s3+ A[i+3]*A[i+3];
}      R0      R1       R2
//remainder
s=s0+s1+s2+s3
```

```
…
V64MULT(R1,R2)        → R1
V64ADD(R0,R1)         → R0
…
```

Research Computing @ CU Boulder   USGS parallel computing workshop   31   03/13/17



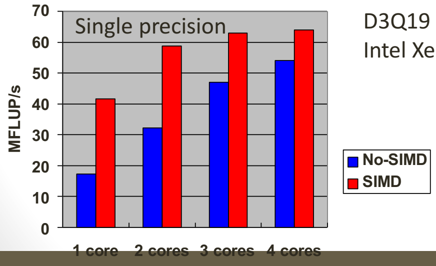
SIMD - Basics

```
float s=0.0;
for(int i=0; i<n;i++)
        s = s + A[i]*A[i];
```

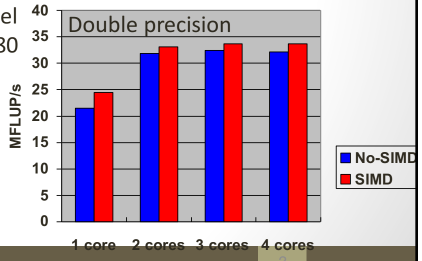
Intel Xeon E3-1280 (3.5 GHz)
`A[]` in L1 cache:

- `icc -O3` → 41.1 GFLOP/s
- `icc -O1` → 2.3 GFLOP/s

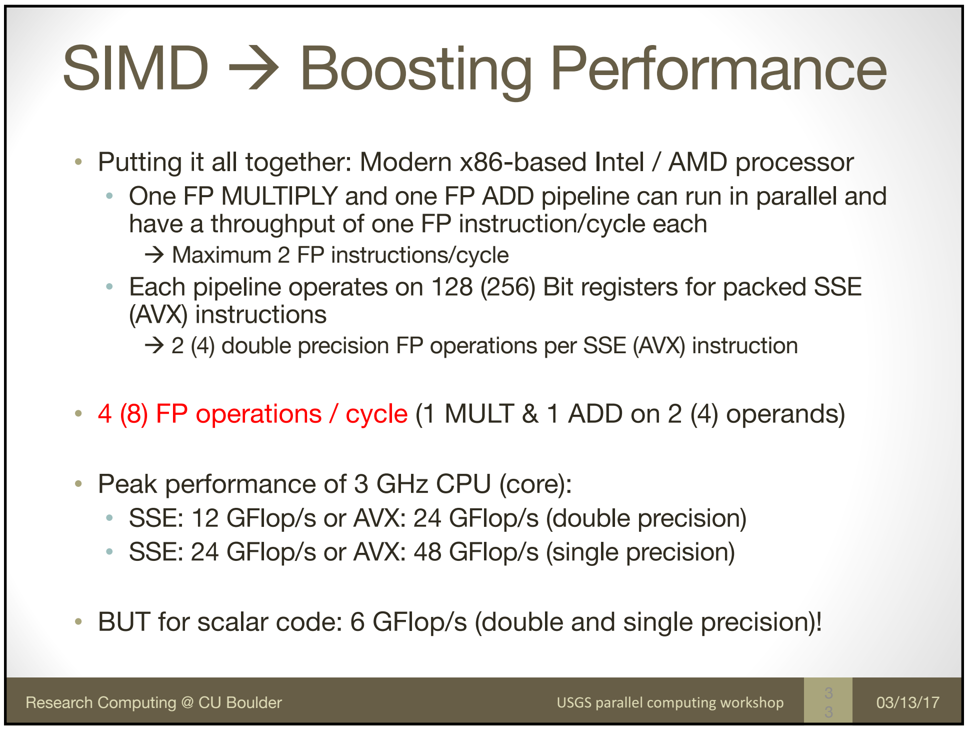
- **SIMD-processing is a must if data is loaded from cache or main memory bandwidth is not saturated**
- **If memory bandwidth is fully utilized, SIMD-processing can not help a lot**



D3Q19 LBM kernel
Intel Xeon E3-1280



Research Computing @ CU Boulder   USGS parallel computing workshop   32   03/13/17



SIMD → Boosting Performance

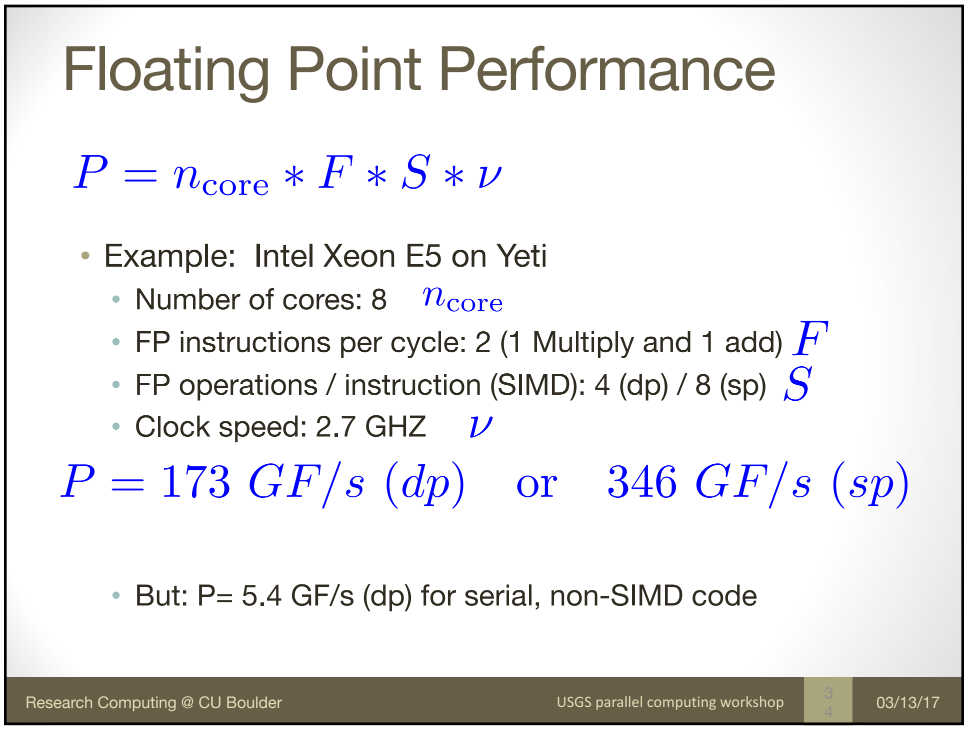
- Putting it all together: Modern x86-based Intel / AMD processor
  - One FP MULTIPLY and one FP ADD pipeline can run in parallel and have a throughput of one FP instruction/cycle each
    → Maximum 2 FP instructions/cycle
  - Each pipeline operates on 128 (256) Bit registers for packed SSE (AVX) instructions
    → 2 (4) double precision FP operations per SSE (AVX) instruction

- 4 (8) FP operations / cycle (1 MULT & 1 ADD on 2 (4) operands)

- Peak performance of 3 GHz CPU (core):
  - SSE: 12 GFlop/s or AVX: 24 GFlop/s (double precision)
  - SSE: 24 GFlop/s or AVX: 48 GFlop/s (single precision)

- BUT for scalar code: 6 GFlop/s (double and single precision)!

Research Computing @ CU Boulder USGS parallel computing workshop 33 03/13/17



Floating Point Performance

$$P = n_{\text{core}} * F * S * \nu$$

- Example: Intel Xeon E5 on Yeti
  - Number of cores: 8 n_{core}
  - FP instructions per cycle: 2 (1 Multiply and 1 add) F
  - FP operations / instruction (SIMD): 4 (dp) / 8 (sp) S
  - Clock speed: 2.7 GHZ ν

$$P = 173\ GF/s\ (dp) \quad \text{or} \quad 346\ GF/s\ (sp)$$

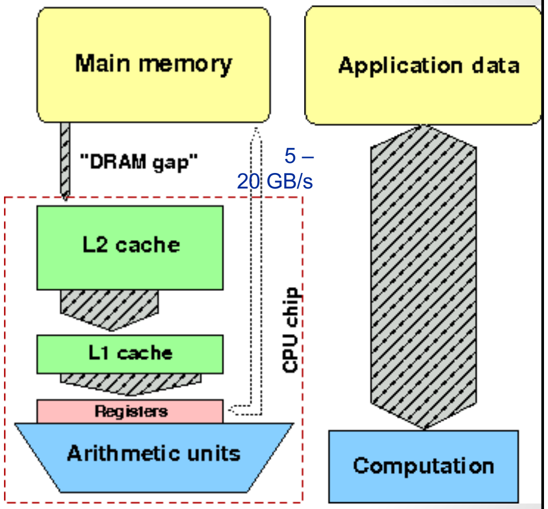
- But: P= 5.4 GF/s (dp) for serial, non-SIMD code

Research Computing @ CU Boulder USGS parallel computing workshop 34 03/13/17

Memory hierarchy

1. CPU/Arithmetic unit issues a load request to transfer a data item to a register
2. Cache logic automatically checks all cache levels if data item is already in cache
3. If data item is in cache "cache hit" it is loaded to register.
4. If data item is in no cache level ("cache miss") data item is loaded from main memory and a copy is held in cache

If cache is already full another cache line must be invalidated or "evicted" in 4



Shared Memory Parallel Processing – Thought Experiment

- Jigsaw puzzle analogy**
  - Let's say you decide to let one of your friends, Stacey, join you
  - Stacey and you sit at a table and each work on half the puzzle
    - In theory you reduce the puzzle time completion by half
    - However, other time overhead
      - Reaching for the same puzzle pieces
        - Resource contention
      - Communicating about puzzle interfaces
    - Might take 35 minutes instead of 30

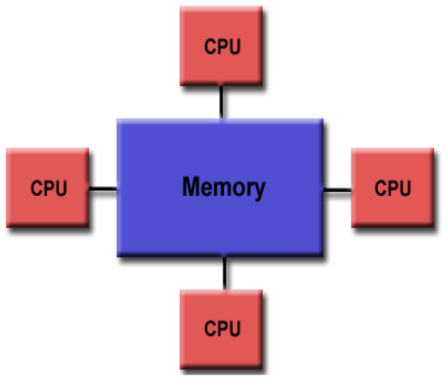
**from Henry Neeman, OSCER, "Supercomputing in Plain English"

Shared Memory Parallel Processing – Thought Experiment

- Jigsaw puzzle analogy**
  - Now you let your other two friends, Fred and Jim, join in
    - Now conceivably could finish in ¼ the time (15 minutes)
  - But there's even more contention for resources
  - More communication
  - Slows down the process even more (maybe takes 23 minutes to complete instead)
  - Too many people slows down the process too much to make it worthwhile
    - Eventually have a "diminishing return"

**from Henry Neeman, OSCER, "Supercomputing in Plain English"

Shared-memory Model

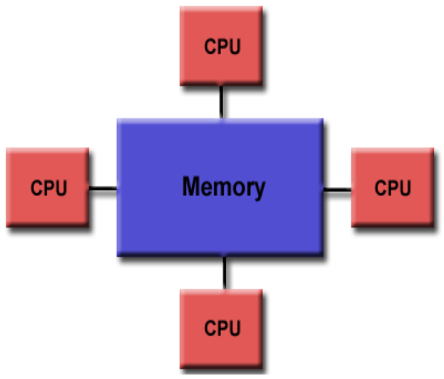


The concept is that all processors can access all memory available

Multiple processors can perform tasks on their own but share the same memory

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Shared-memory Model

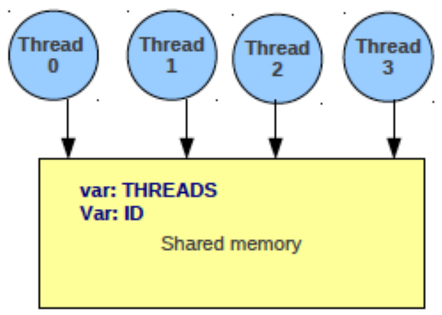


Advantage: data sharing is fast and uniform

Disadvantage: adding more processors can cause performance issues when accessing the same shared memory resource

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Shared-memory Model



A thread is a block of code with one entry and one exit that is abstract and is mapped onto a physical core. Multiple threads can be mapped onto one core.

Threads communicate by depositing contents in shared memory area

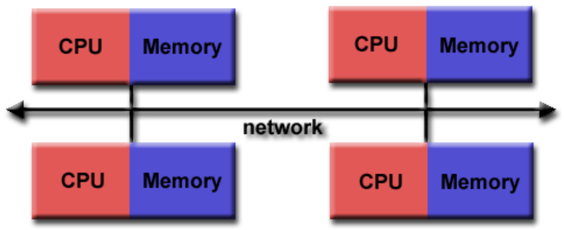
Source: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

Distributed Memory Parallel Processing – Thought Experiment

- Jigsaw puzzle analogy**
  - Now we have two tables with one person at each table doing the puzzle
  - We split the puzzle equally between tables
  - Each person works completely independently
  - But to communicate costs more
    - How do you work out connecting the puzzle?
  - Can you really divide up the puzzle evenly?

**from Henry Neeman, OSCER, “Supercomputing in Plain English”

Distributed-memory Model

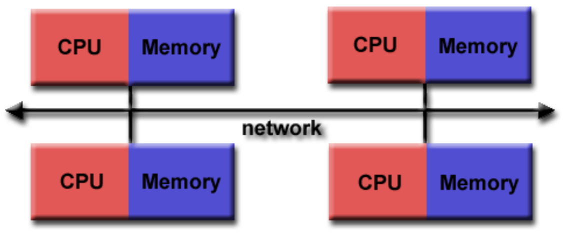


Distributed memory requires a communication network to connect memory

Processors have own memory and don’t map globally

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Distributed-memory Model

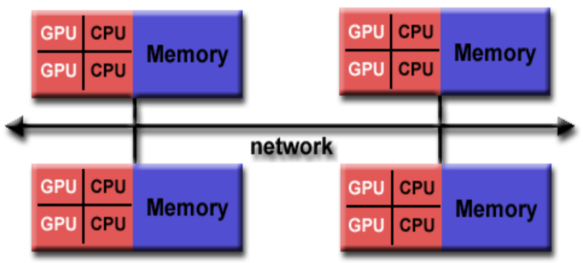


Programmers explicitly define how processors access other processor's memory

Advantage: scalable memory
Disadvantage: need to know parallel programming!

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

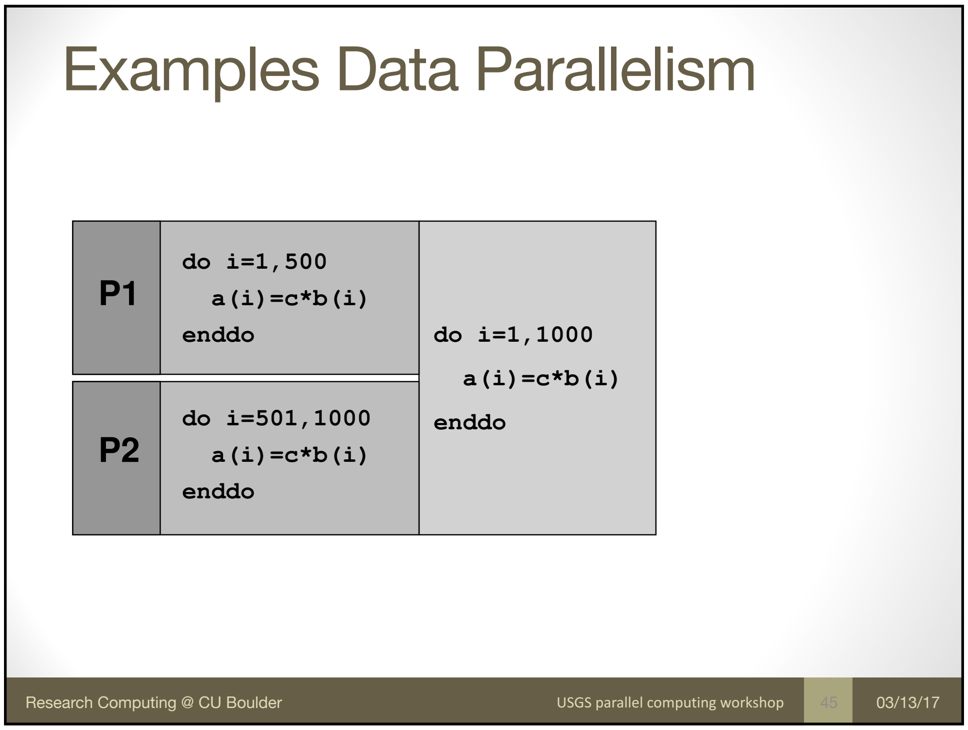
Distributed-Shared Memory



Most large and fast computers now

Shared memory machines connected to other shared memory machines

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared



Examples Data Parallelism

P1

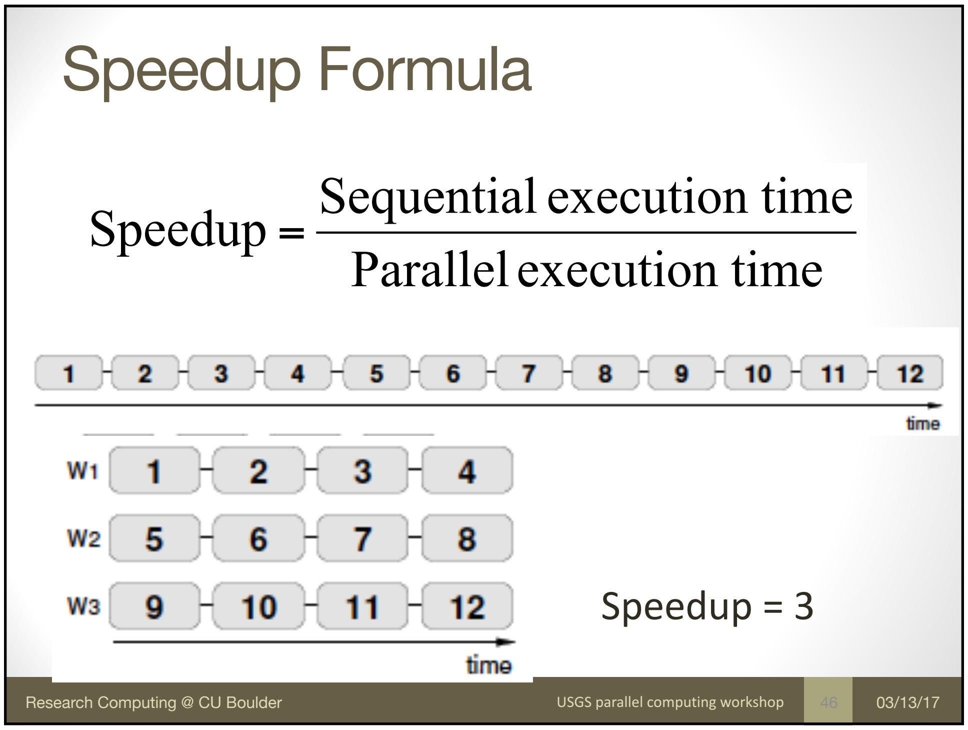
```
do i=1,500
  a(i)=c*b(i)
enddo
```

P2

```
do i=501,1000
  a(i)=c*b(i)
enddo
```

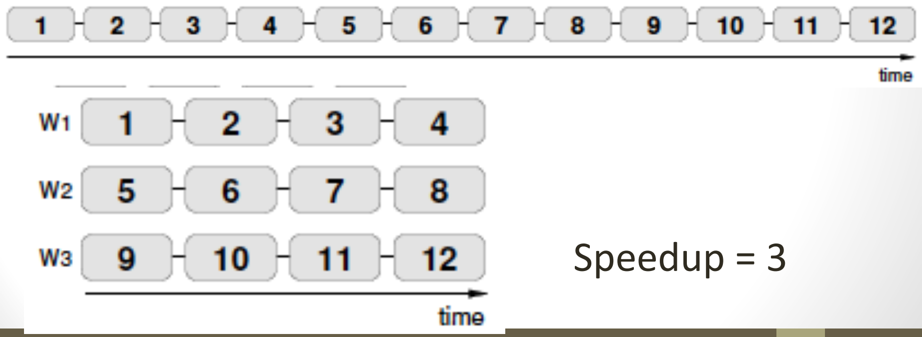
```
do i=1,1000
  a(i)=c*b(i)
enddo
```

Research Computing @ CU Boulder USGS parallel computing workshop 45 03/13/17



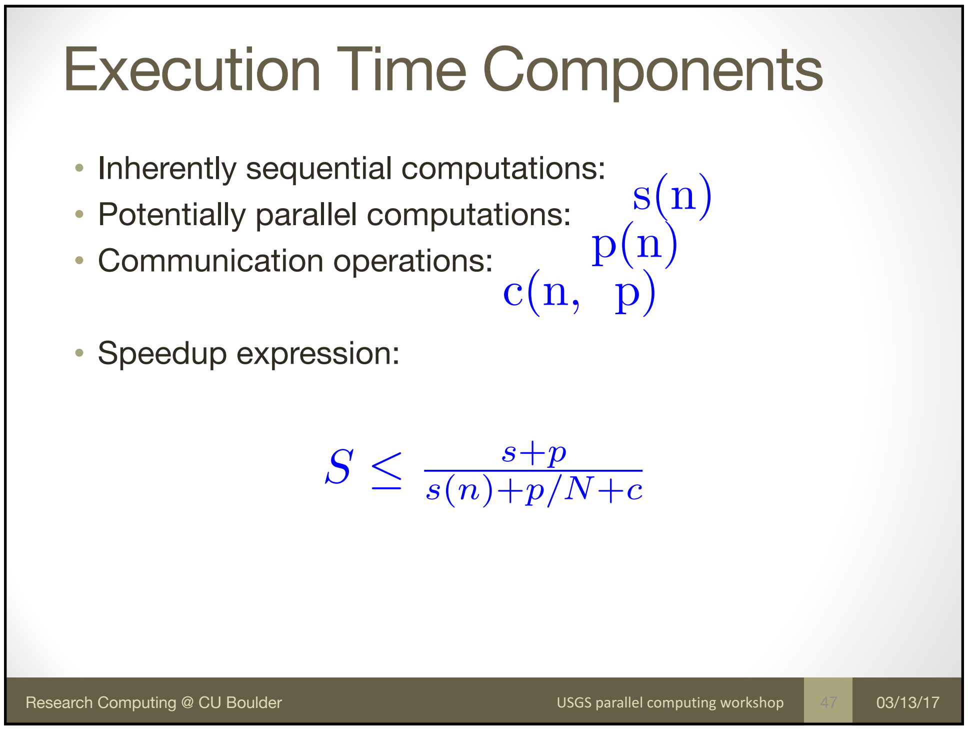
Speedup Formula

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$



Speedup = 3

Research Computing @ CU Boulder USGS parallel computing workshop 46 03/13/17

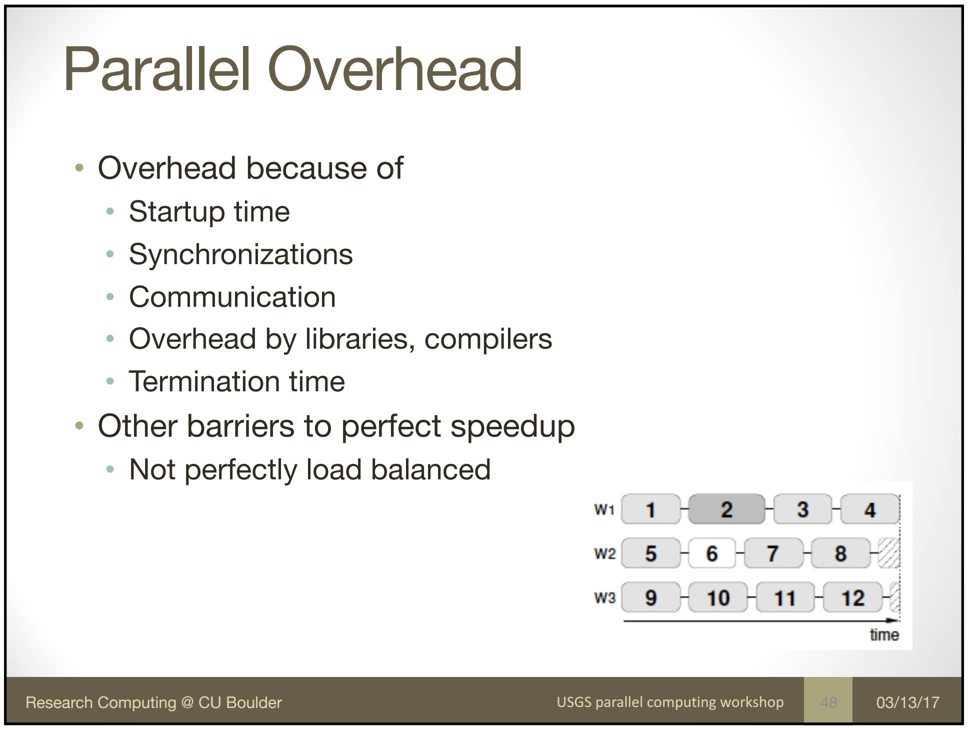


Execution Time Components

- Inherently sequential computations: $s(n)$
- Potentially parallel computations: $p(n)$
- Communication operations: $c(n, p)$

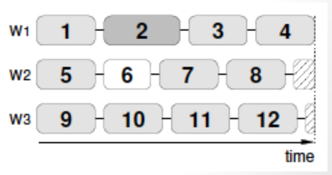
- Speedup expression:

$$S \leq \frac{s+p}{s(n)+p/N+c}$$



Parallel Overhead

- Overhead because of
  - Startup time
  - Synchronizations
  - Communication
  - Overhead by libraries, compilers
  - Termination time
- Other barriers to perfect speedup
  - Not perfectly load balanced

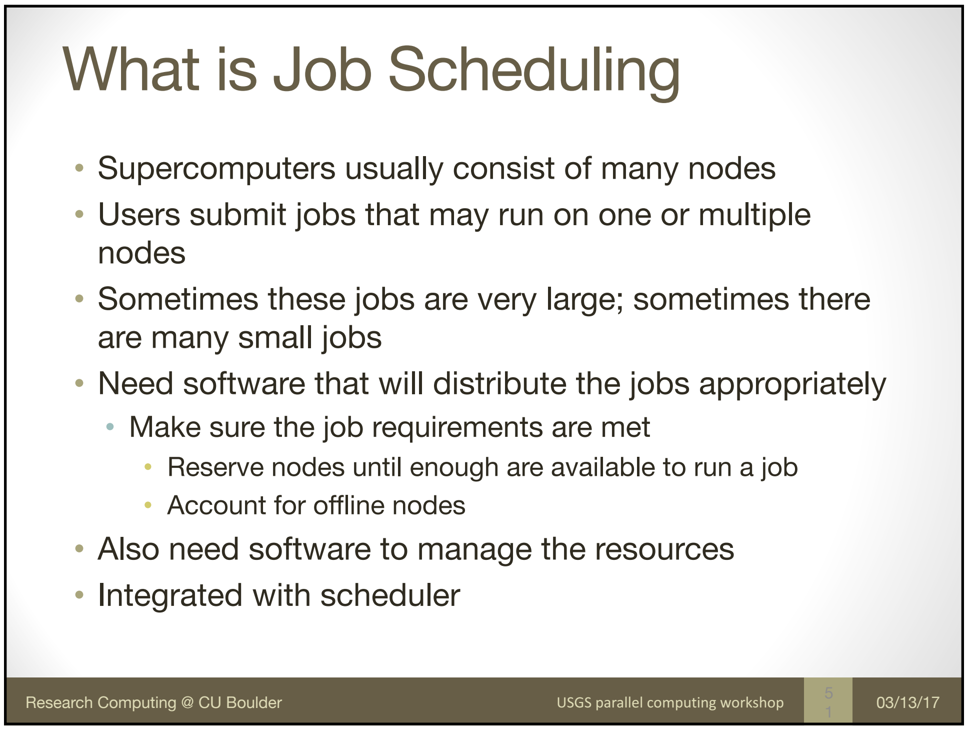


Efficiency

$$\text{Efficiency} = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}$$

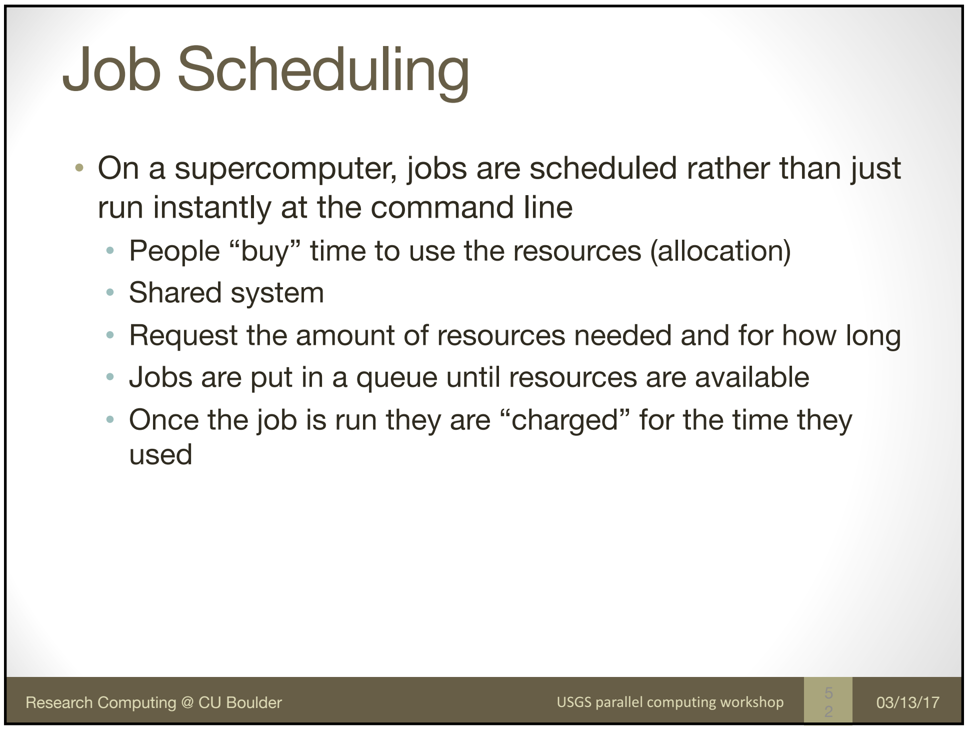
$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

Jobs



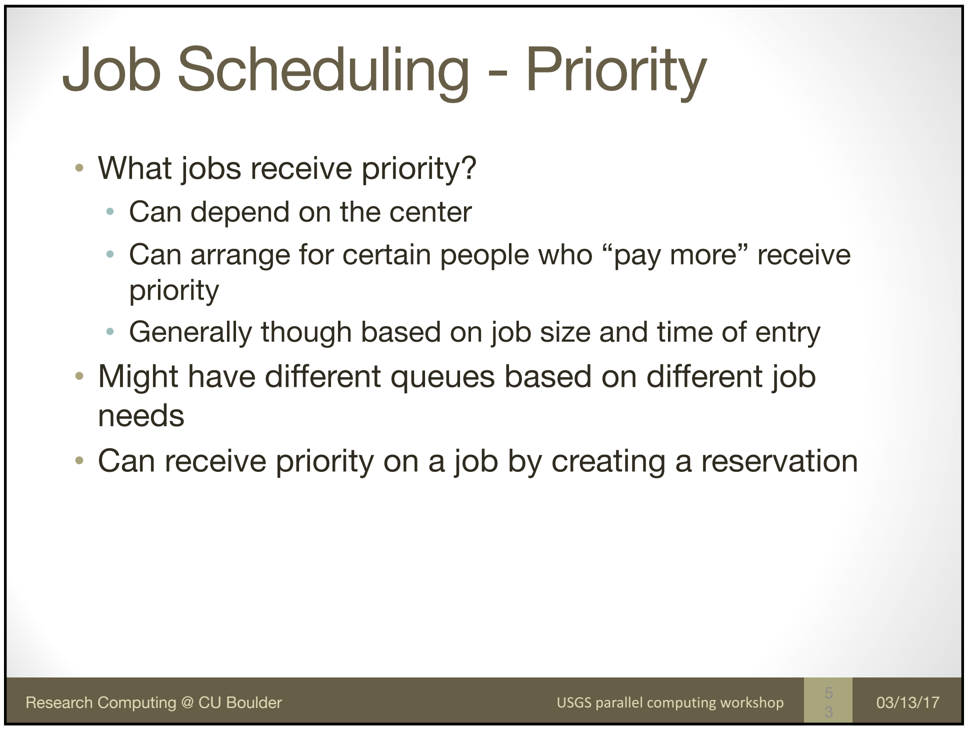
What is Job Scheduling

- Supercomputers usually consist of many nodes
- Users submit jobs that may run on one or multiple nodes
- Sometimes these jobs are very large; sometimes there are many small jobs
- Need software that will distribute the jobs appropriately
  - Make sure the job requirements are met
    - Reserve nodes until enough are available to run a job
    - Account for offline nodes
- Also need software to manage the resources
- Integrated with scheduler



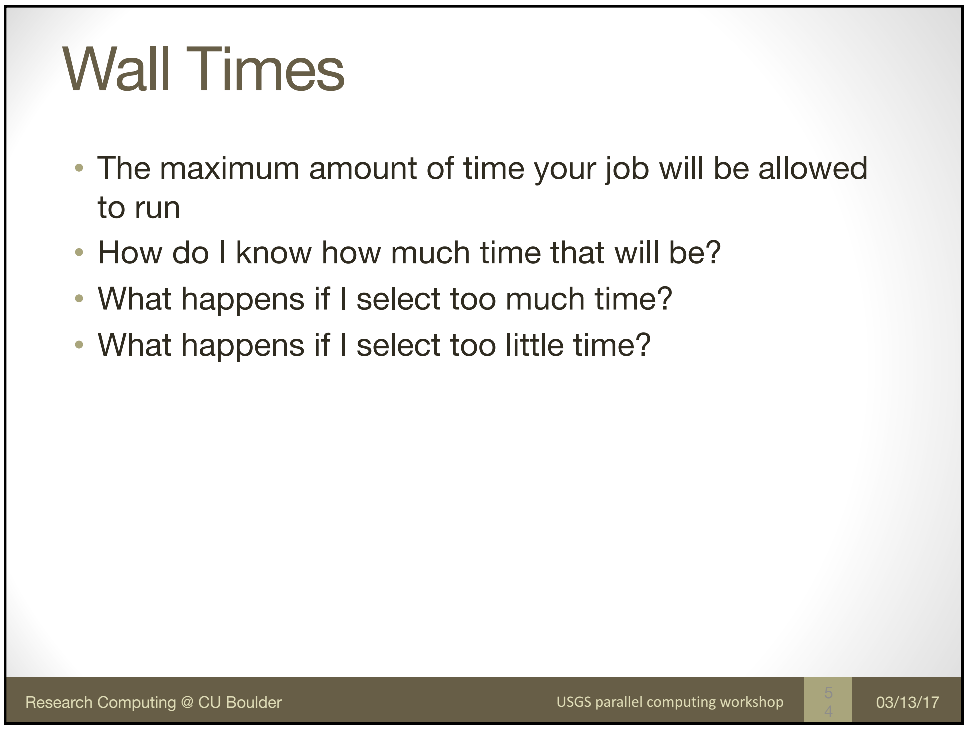
Job Scheduling

- On a supercomputer, jobs are scheduled rather than just run instantly at the command line
  - People "buy" time to use the resources (allocation)
  - Shared system
  - Request the amount of resources needed and for how long
  - Jobs are put in a queue until resources are available
  - Once the job is run they are "charged" for the time they used



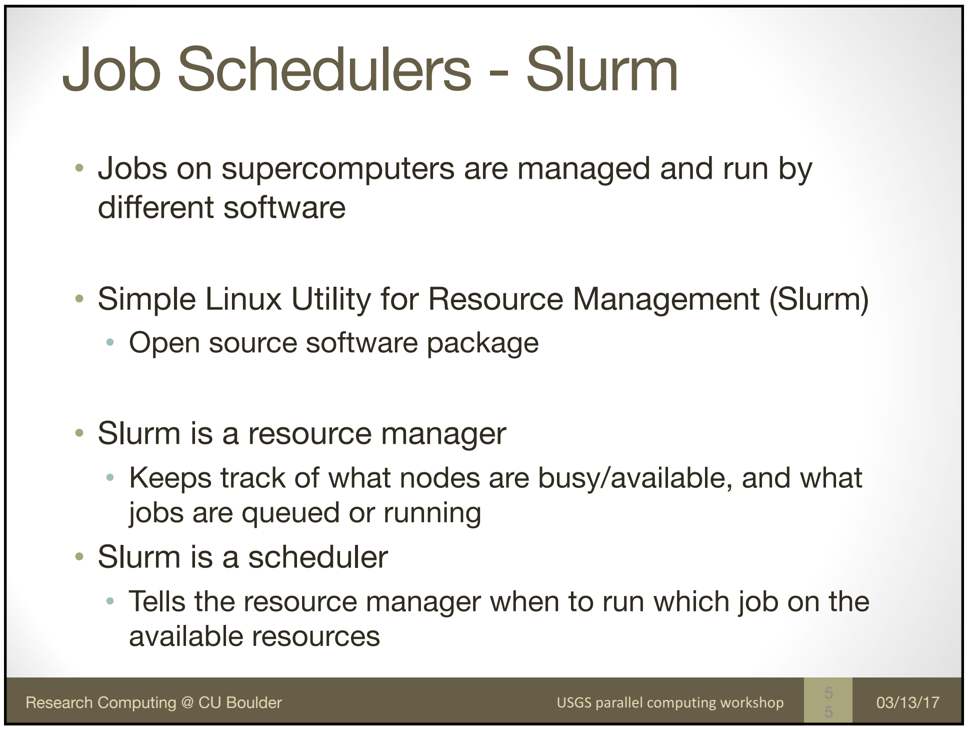
Job Scheduling - Priority

- What jobs receive priority?
  - Can depend on the center
  - Can arrange for certain people who “pay more” receive priority
  - Generally though based on job size and time of entry
- Might have different queues based on different job needs
- Can receive priority on a job by creating a reservation



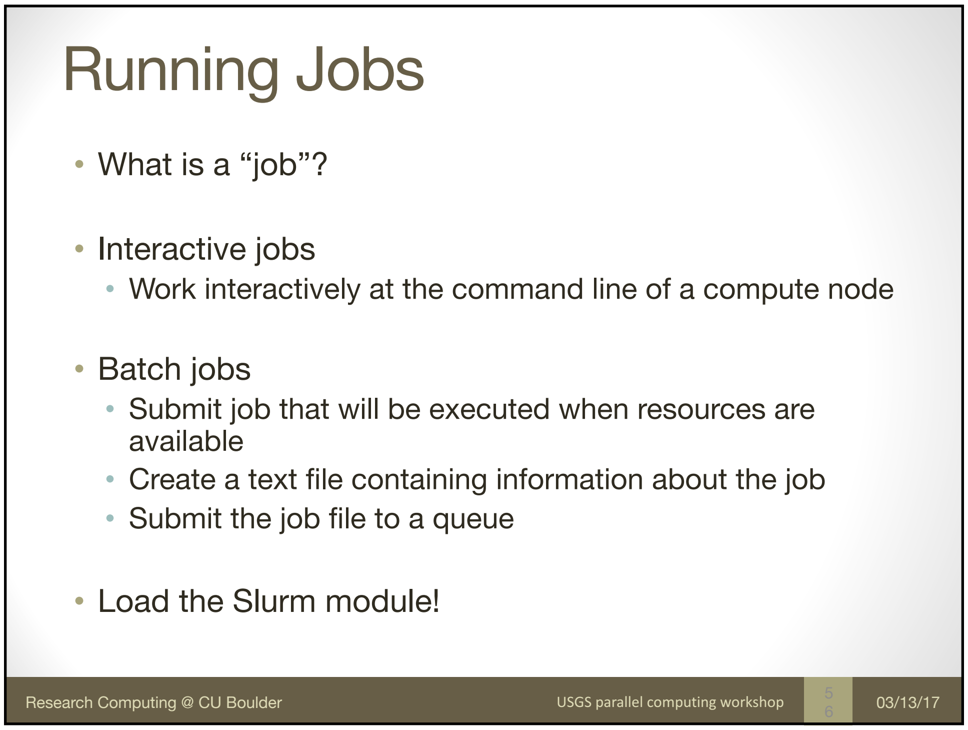
Wall Times

- The maximum amount of time your job will be allowed to run
- How do I know how much time that will be?
- What happens if I select too much time?
- What happens if I select too little time?



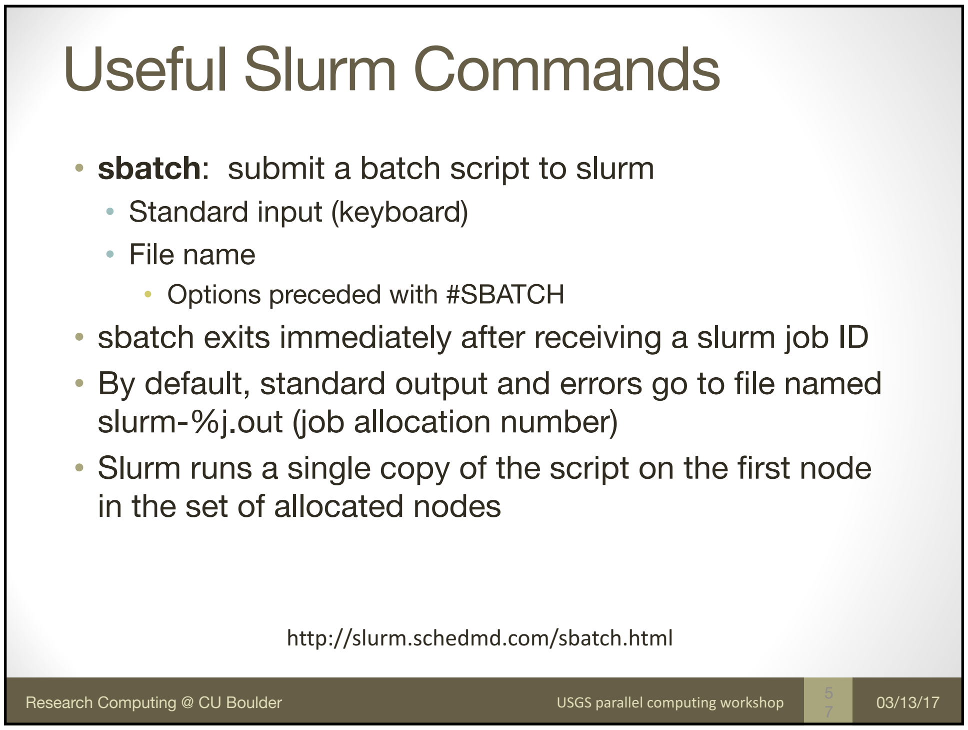
Job Schedulers - Slurm

- Jobs on supercomputers are managed and run by different software
- Simple Linux Utility for Resource Management (Slurm)
  - Open source software package
- Slurm is a resource manager
  - Keeps track of what nodes are busy/available, and what jobs are queued or running
- Slurm is a scheduler
  - Tells the resource manager when to run which job on the available resources



Running Jobs

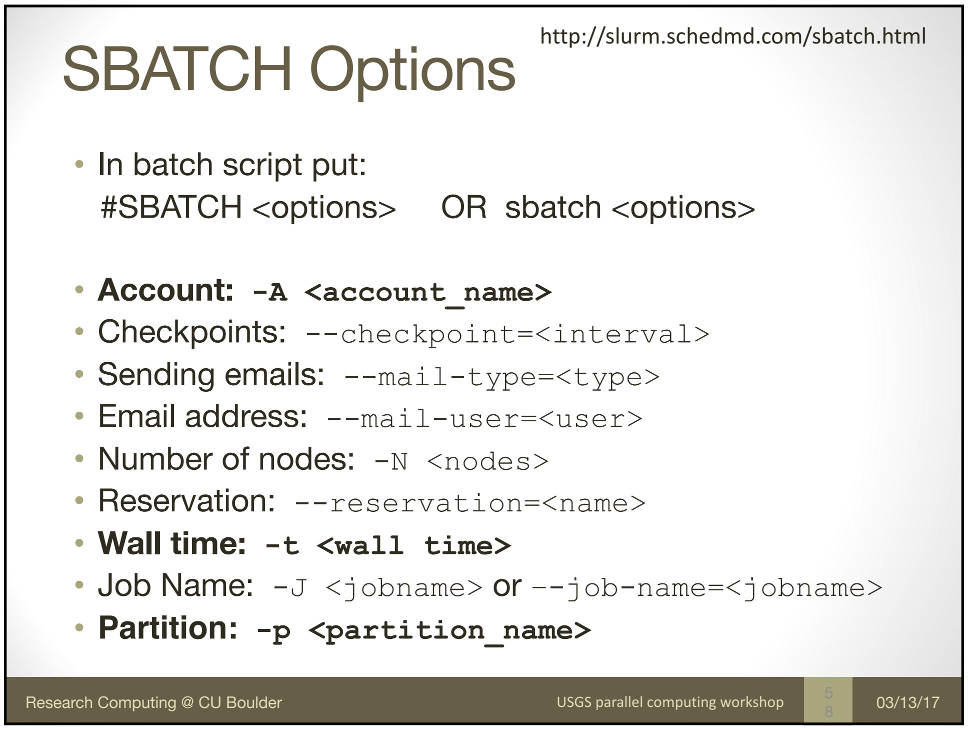
- What is a “job”?
- Interactive jobs
  - Work interactively at the command line of a compute node
- Batch jobs
  - Submit job that will be executed when resources are available
  - Create a text file containing information about the job
  - Submit the job file to a queue
- Load the Slurm module!



Useful Slurm Commands

- **sbatch**: submit a batch script to slurm
  - Standard input (keyboard)
  - File name
    - Options preceded with #SBATCH
- sbatch exits immediately after receiving a slurm job ID
- By default, standard output and errors go to file named slurm-%j.out (job allocation number)
- Slurm runs a single copy of the script on the first node in the set of allocated nodes

http://slurm.schedmd.com/sbatch.html

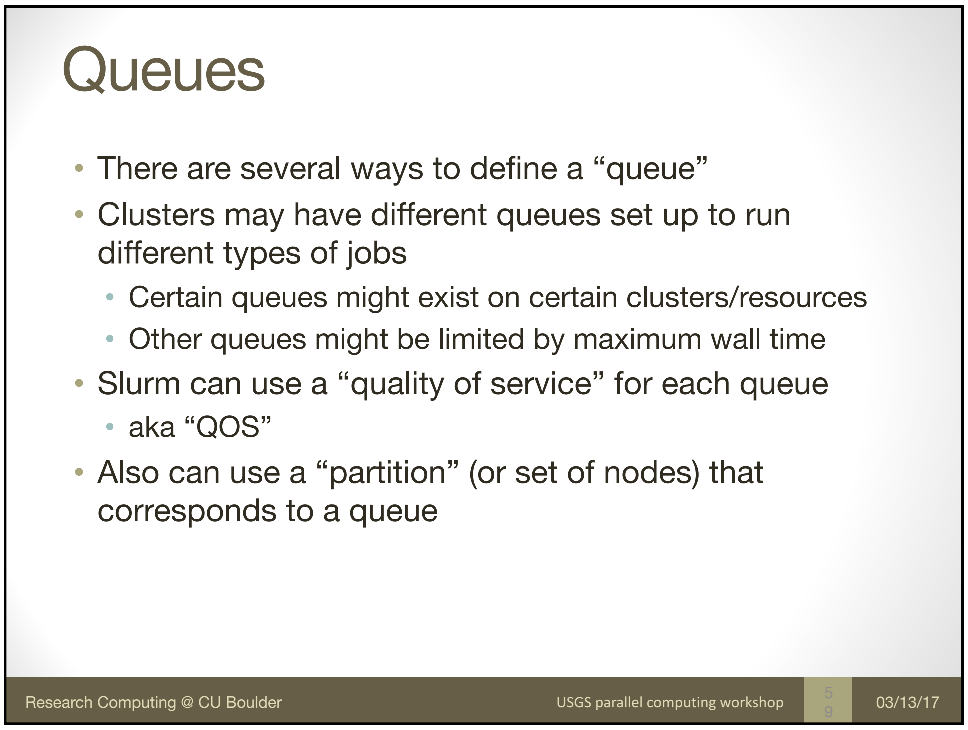


SBATCH Options

http://slurm.schedmd.com/sbatch.html

- In batch script put:
  #SBATCH <options> OR sbatch <options>

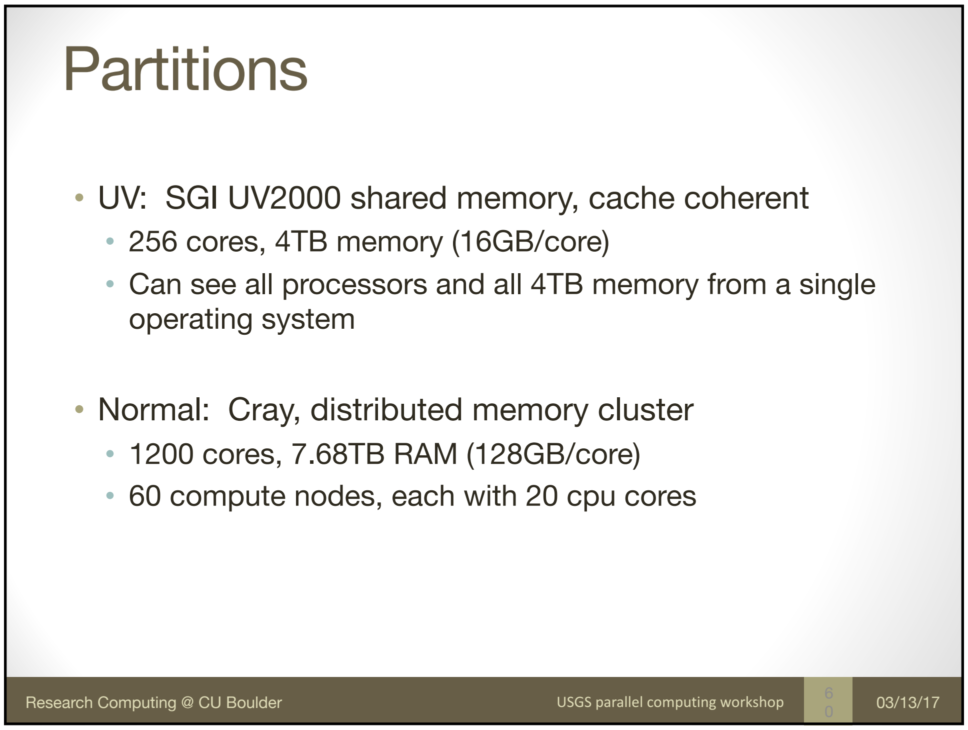
- **Account: `-A <account_name>`**
- Checkpoints: `--checkpoint=<interval>`
- Sending emails: `--mail-type=<type>`
- Email address: `--mail-user=<user>`
- Number of nodes: `-N <nodes>`
- Reservation: `--reservation=<name>`
- **Wall time: `-t <wall time>`**
- Job Name: `-J <jobname>` or `--job-name=<jobname>`
- **Partition: `-p <partition_name>`**



Queues

- There are several ways to define a “queue”
- Clusters may have different queues set up to run different types of jobs
  - Certain queues might exist on certain clusters/resources
  - Other queues might be limited by maximum wall time
- Slurm can use a “quality of service” for each queue
  - aka “QOS”
- Also can use a “partition” (or set of nodes) that corresponds to a queue

Research Computing @ CU Boulder USGS parallel computing workshop 59 03/13/17



Partitions

- UV: SGI UV2000 shared memory, cache coherent
  - 256 cores, 4TB memory (16GB/core)
  - Can see all processors and all 4TB memory from a single operating system
- Normal: Cray, distributed memory cluster
  - 1200 cores, 7.68TB RAM (128GB/core)
  - 60 compute nodes, each with 20 cpu cores

Research Computing @ CU Boulder USGS parallel computing workshop 60 03/13/17

Software

- Common software is available to everyone on the systems
- To find out what software is available, you can type `module avail`
- To set up your environment to use a software package, type `module load <package>/<version>`
- Can install your own software
  - But you are responsible for support
  - We can assist

Login to Yeti

- Step1. Log in to Yeti.

```
Laptop ~$  ssh
name@yeti.cr.usgs.gov
```

- Step 2. Clone the git repository

```
yeti-login01 ~$ git clone  \
  https://github.com/ResearchComp
  uting/USGS_2017_03
```

Login to Yeti

- Step1. Log in to Yeti.

```
Laptop ~$  ssh
name@yeti.cr.usgs.gov
```

- Step 2. Clone the git repository

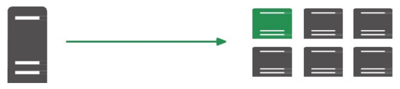
```
yeti-login01 ~$ git clone  \
  https://github.com/ResearchComp
  uting/USGS_2017_03
```

3a. Start an interactive compute job on UV partition

```
 yeti-login01 ~$ salloc -A training               \
              -p UV                               \
              -t 01:00:00  -n 1                   \
              --cpus-per-task=4                   \
              --reservation=training_UV.   \
              --gres=gpu:tesla:[1-6]
```

3b. Start a interactive compute job on normal partition

```
 yeti-login01 ~$ salloc -A training               \
              -p  normal                          \
              -t  01:00:00  -n  1                 \
              --cpus-per-task=4                   \
              --reservation=training_normal
```



4. Load the Intel parallel studio module

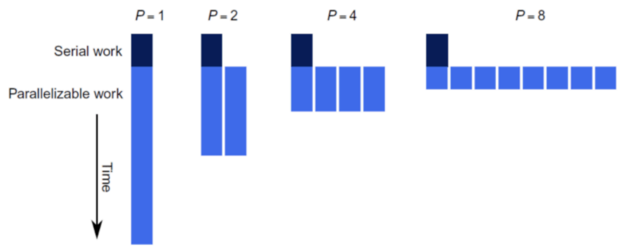
```
compute80  ~$  module load intel/psxe-2015
```

This contains the C, C++ and Fortran compilers as well as the Intel MPI library.

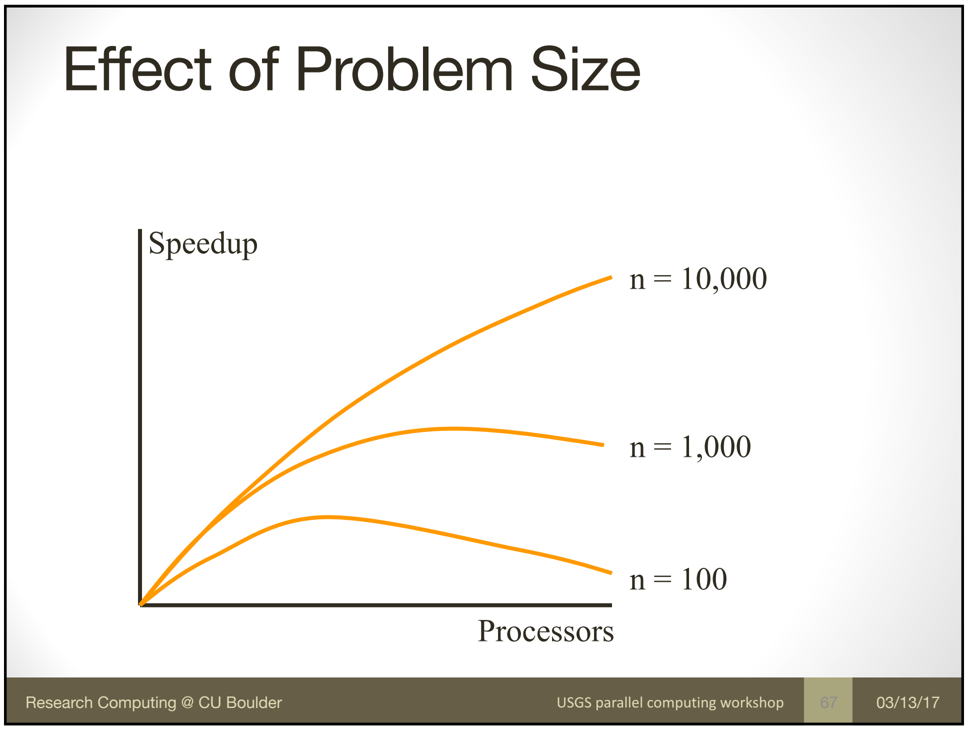
Language	Compiler
C	icc
C++	icpc
Fortran	ifort

Strong Scaling

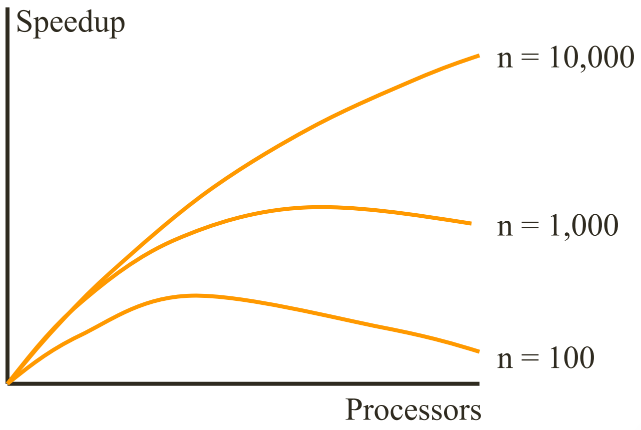
- Keep problem size the same
- Increase the number of processors

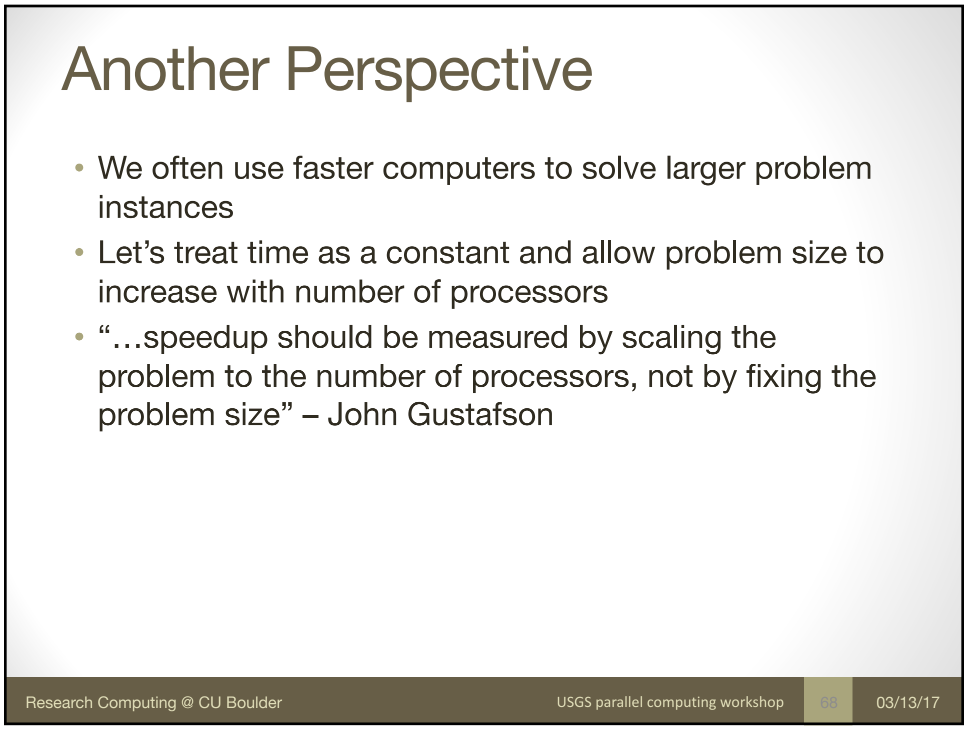


http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2



Effect of Problem Size

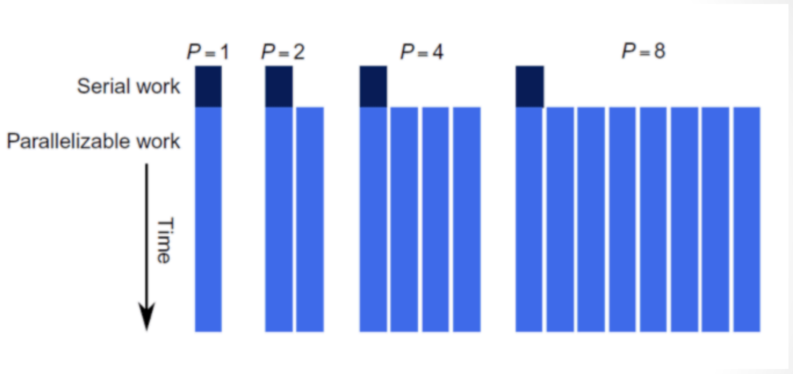




Another Perspective

- We often use faster computers to solve larger problem instances
- Let’s treat time as a constant and allow problem size to increase with number of processors
- “…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size” – John Gustafson

Weak Scaling



http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

Summary

- Access to main memory is most of the times your bottleneck
- Speedup
- Strong Scaling
- Weak Scaling