High Performance Computing (HPC) Introduction

Ontario Summer School on
High Performance Computing

Scott Northrup
SciNet HPC Consortium
Compute Canada

June 9th, 2014

Outline

Acknowledgments

Contributing Material

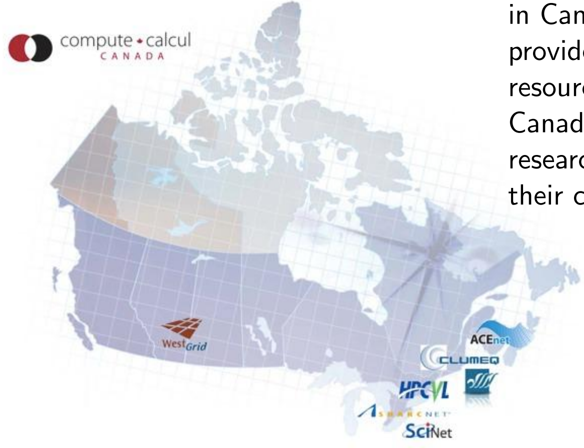
- SciNet Parallel Scientific Computing Course
  - L. J. Dursi & R. V. Zon, SciNet
- Parallel Computing Models - D. McCaughan, SHARCNET
- High Performance Computing - D. McCaughan, SHARCNET
- HPC Architecture Overview - H. Merez, SHARCNET
- Intro to HPC - T. Whitehead, SHARCNET

Outline

SciNet is ...

. . . a consortium for High-Performance Computing consisting of researchers at U. of T. and its associated hospitals.

One of 7 consortia in Canada that provide HPC resources to Canadian academic researchers and their collaborators.



SciNet
compute • calcul
CANADA

SciNet is ...

. . . home to the 1st and 2nd most powerful research supercomputers in Canada (and a few more)

SciNet is ...

. . . where to take courses on computational topics, e.g.

- Intro to SciNet
- Linux Shell
- Scientific Programming (Modern FORTRAN / C++)
- GPGPU with CUDA
- Intro to Research Computing with Python
- Scientific Computing Course (for credit for physics/astro grads)
- Ontario HPC Summerschool

https://support.scinet.utoronto.ca/education/

SciNet people

...technical analysts who can work directly with you.

- Bertrand Brelier
- Jonathan Dursi
- Scott Northrup
- Erik Spence
- Ramses van Zon
- Daniel Gruner (CTO-software)

+ the people that make sure everything runs smoothly.

- Jaime Pinto
- Joseph Chen
- Jason Chong
- Ching-Hsing Yu
- Neil Knecht
- Leslie Groer
- Chris Loken (CTO)

+ Technical director: Prof. Richard Peltier

+ Business manager: Teresa Henriques

+ Project coordinator: Jillian Dempsey

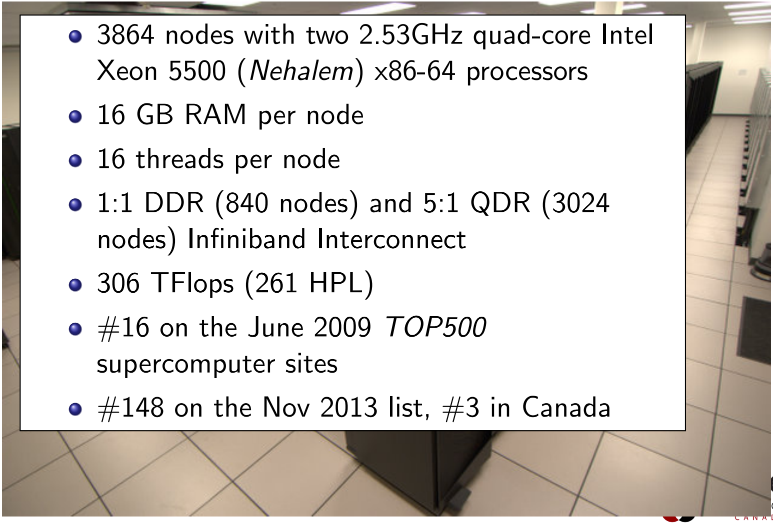
How to get an account

Any qualified researcher at a Canadian university can get a SciNet account through the following process:

1. Register for a Compute Canada Database (CCDB) account
2. Non-faculty need a sponsor (supervisor's CCRI number), who has to have a SciNet account already.
3. Login to CCDB and apply for a SciNet account (click *Apply* beside SciNet on the *Consortium Accounts* page)
4. Agree to the Acceptable Usage Policy (e.g., don't share account, respect others, we can monitor your jobs)

Resources at SciNet

General Purpose Cluster (GPC)



- 3864 nodes with two 2.53GHz quad-core Intel Xeon 5500 (*Nehalem*) x86-64 processors
- 16 GB RAM per node
- 16 threads per node
- 1:1 DDR (840 nodes) and 5:1 QDR (3024 nodes) Infiniband Interconnect
- 306 TFlops (261 HPL)
- #16 on the June 2009 *TOP500* supercomputer sites
- #148 on the Nov 2013 list, #3 in Canada

Other Compute Resources at SciNet

Tightly Coupled System (TCS)

Power 7 Linux Cluster (P7)

GPU Devel Nodes (ARC/Gravity)

Blue Gene/Q (BGQ)

Storage Resources at SciNet

Disk space

- 1.4 PB of storage in 1790 drives
- Two controllers each delivering 4-5 GB/s (r/w)
- *Shared* file system GPFS on all systems

Storage space

- HPSS: 5TB Tape-backed storage

Outline

Scientific High Performance Computing

What is it?

HPC is essentially leveraging larger and/or multiple computers to solve computations in parallel.

Scientific High Performance Computing

What is it?

HPC is essentially leveraging larger and/or multiple computers to solve computations in parallel.

What does it involve?

- hardware - pipelining, instruction sets, multi-processors, inter-connects
- algorithms - concurrency, efficiency, communications
- software - parallel approaches, compilers, optimization, libraries

Scientific High Performance Computing

What is it?

HPC is essentially leveraging larger and/or multiple computers to solve computations in parallel.

What does it involve?

- hardware - pipelining, instruction sets, multi-processors, inter-connects
- algorithms - concurrency, efficiency, communications
- software - parallel approaches, compilers, optimization, libraries

When do I need HPC?

- My problem takes to long ⟶ more/faster computation
- My problem is to big ⟶ more memory
- My data is to big ⟶ more storage

Scientific High Performance Computing

Why is it necessary?

- Modern experiments and observations yield vastly more data to be processed than in the past.
- As more computing resources become available, the bar for cutting edge simulations is raised.
- Science that could not have been done before becomes tractable.

Scientific High Performance Computing

Why is it necessary?

- Modern experiments and observations yield vastly more data to be processed than in the past.
- As more computing resources become available, the bar for cutting edge simulations is raised.
- Science that could not have been done before becomes tractable.

However

- Advances in clock speeds, bigger and faster memory and storage have been lagging as compared to e.g. 10 years ago. *Can no longer "just wait a year" and get a better computer.*
- So modern HPC means more hardware, not faster hardware.
- Thus parallel programming/computing is required.

Analogy

HR Dilemma

- Problem: job needs to get done faster

Analogy

HR Dilemma

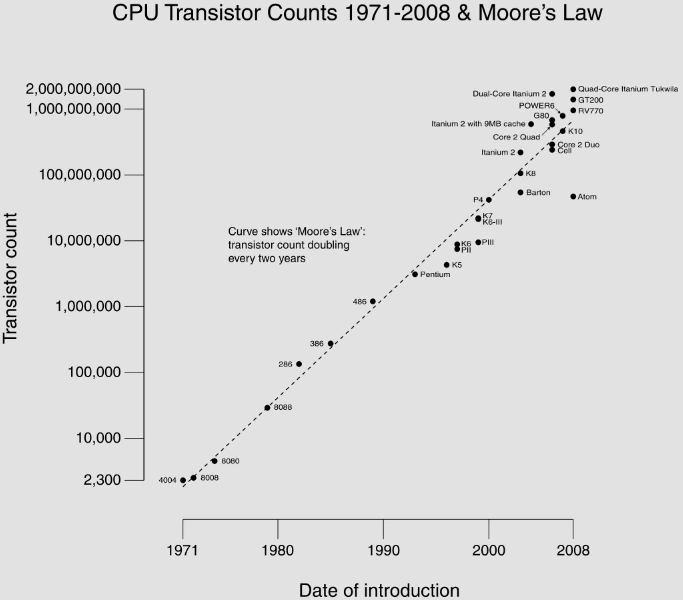
- Problem: job needs to get done faster
  - can't hire substantially faster people
  - can hire more people

Analogy

HR Dilemma

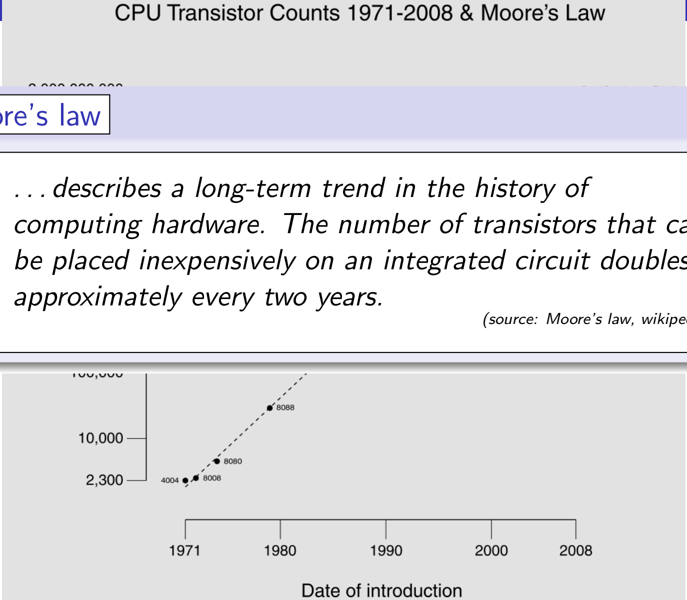
- Problem: job needs to get done faster
  - can't hire substantially faster people
  - can hire more people
- Solution:
  - split work up between people (divide and conquer)
  - requires rethinking the work flow process
  - requires administration overhead
  - eventually administration larger than actual work

Wait, what about Moore's Law?



(source: Transistor Count and Moore's Law - 2008.svg, by Wgsimon, wikipedia)

Wait, what about Moore's Law?



Moore's law

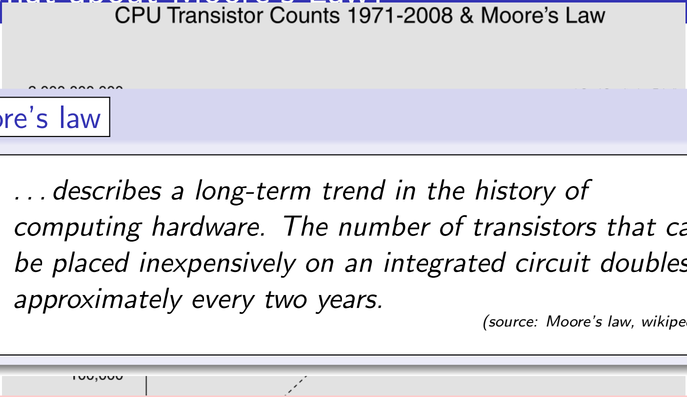
. . . describes a long-term trend in the history of computing hardware. The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years.

(source: Moore's law, wikipedia)

(source: Transistor Count and Moore's Law - 2008.svg, by Wgsimon, wikipedia)

SciNet compute • calcul CANADA

Wait, what about Moore's Law?



Moore's law

...describes a long-term trend in the history of computing hardware. The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years.

(source: Moore's law, wikipedia)

But...

- *Moores Law didn't promise us clock speed.*
- *More transistors but getting hard to push clockspeed up. Power density is limiting factor.*
- *So more cores at fixed clock speed.*

Outline

Parallel Computing

Thinking Parallel

The general idea is if one processor is good, many processors will be better

- Parallel programming is not generally trivial
- Tools for automated parallelism are either highly specialized or absent
- serial algorithms/mathematics don't always work well in parallel without modification

Parallel Computing

Thinking Parallel

The general idea is if one processor is good, many processors will be better

- Parallel programming is not generally trivial
- Tools for automated parallelism are either highly specialized or absent
- serial algorithms/mathematics don't always work well in parallel without modification

Parallel Programming

- its Necessary (serial performance has peaked)
- its Everywhere (cellphones, tablets, laptops, etc)
- its still increaseing (Sequoia 1.5 M cores, Tianhe-2 3.12M cores)

Parallel Computing

NVIDIA Serial vs Parallel Computing

https://www.youtube.com/watch?v=XcolCeWIcss

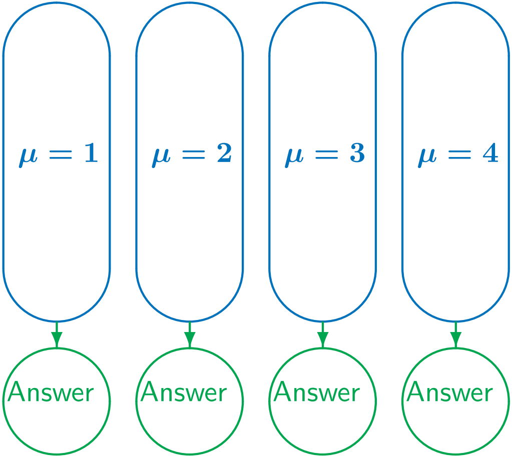
Concurrency

- Must have something to do for all these cores.
- Find parts of the program that can done independently, and therefore concurrently.
- There must be many such parts.
- There order of execution should not matter either.
- Data dependencies limit concurrency.

(source: http://flickr.com/photos/splorp)

Parameter study: best case scenario

- Aim is to get results from a model as a parameter varies.
- Can run the serial program on each processor at the same time.
- Get "more" done.

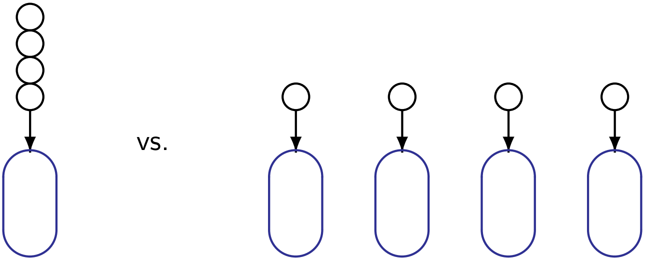


Throughput

- How many tasks can you do per time unit?

$$\text{throughput} = H = \frac{N}{T}$$

- Maximizing H means that you can do as much as possible.
- Independent tasks: using P processors increases H by a factor P



$T = NT_1$
$H = 1/T_1$

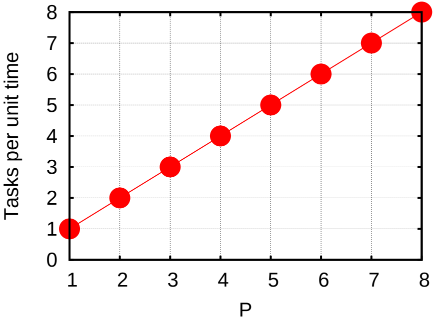
$T = NT_1/P$
$H = P/T_1$

Scaling — Throughput

- How a problem's throughput scales as processor number increases ("strong scaling").
- In this case, linear scaling:

$$H \propto P$$

- This is Perfect scaling.

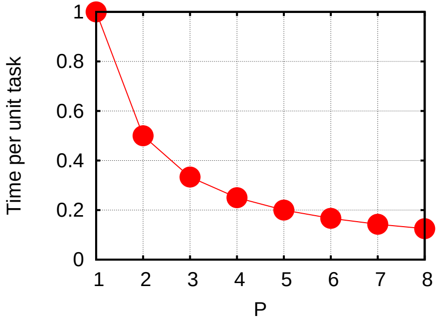


Scaling – Time

- How a problem's timing scales as processor number increases.
- Measured by the time to do one unit. In this case, inverse linear scaling:

$$T \propto 1/P$$

- Again this is the ideal case, or "embarrassingly parallel".

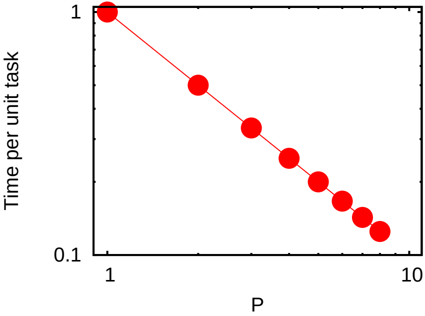


Scaling – Time

- How a problem's timing scales as processor number increases.
- Measured by the time to do one unit. In this case, inverse linear scaling:

$$T \propto 1/P$$

- Again this is the ideal case, or "embarrassingly parallel".

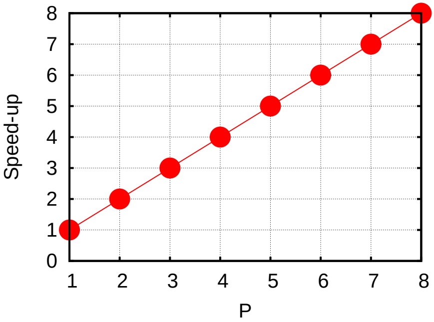


Scaling – Speedup

- How much faster the problem is solved as processor number increases.
- Measured by the serial time divided by the parallel time

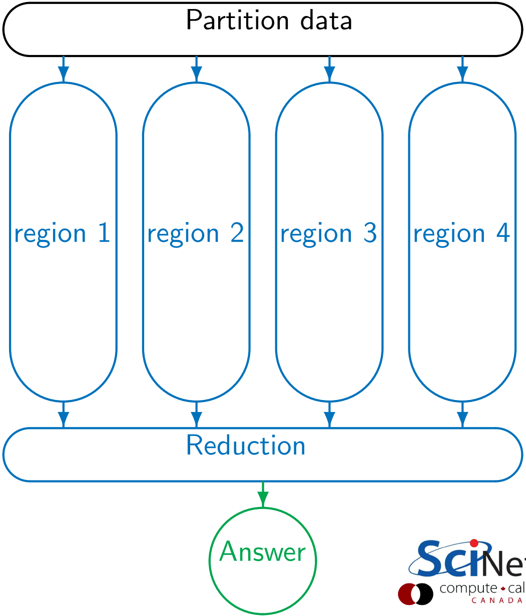
$$S = \frac{T_{serial}}{T(P)} \propto P$$

- For embarrassingly parallel applications: Linear speed up.



Non-ideal cases

- Say we want to integrate some tabulated experimental data.
- Integration can be split up, so different regions are summed by each processor.
- Non-ideal:
  - First need to get data to processor
  - And at the end bring together all the sums: "reduction"



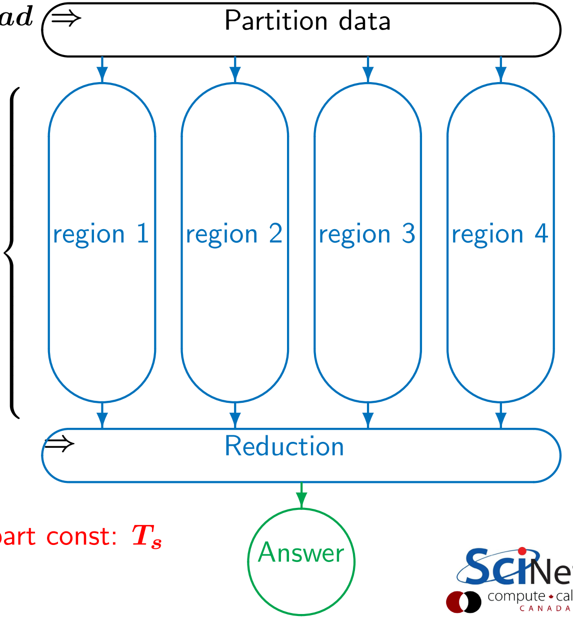
Non-ideal cases

$\boldsymbol{Parallel\ overhead}$ ⇒

$\boldsymbol{Parallel\ region} \Rightarrow$

Perfectly Parallel
(for large $\boldsymbol{N}$)

$\boldsymbol{Serial\ portion}$ ⇒



Suppose non-parallel part const: $\boldsymbol{T_s}$

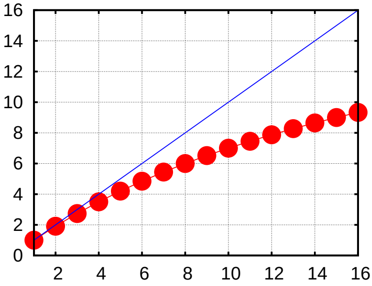
Amdahl's law

Speed-up (without parallel overhead):

$$S = \frac{NT_1 + T_s}{\frac{NT_1}{P} + T_s}$$

or, calling $f = T_s/(T_s + NT_1)$ the serial fraction,

$$S = \frac{1}{f + (1-f)/P}$$



(for $f = 5\%$)

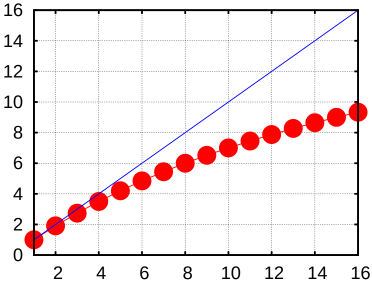
Amdahl's law

Speed-up (without parallel overhead):

$$S = \frac{NT_1 + T_s}{\frac{NT_1}{P} + T_s}$$

or, calling $f = T_s/(T_s + NT_1)$ the serial fraction,

$$S = \frac{1}{f + (1-f)/P} \quad \xrightarrow{P\to\infty} \quad \frac{1}{f}$$



Serial part dominates asymptotically.

Speed-up limited, no matter size of P.

And this is the overly optimistic case!

(for $f = 5\%$)

SciNet compute • calcul CANADA

HPC Lesson #1

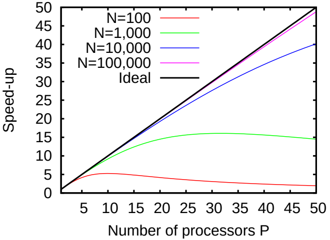
Always keep throughput in mind: if you have several runs, running more of them at the same time on less processors per run is often advantageous.

Trying to beat Amdahl's law #1

Scale up!

The larger N, the smaller the serial fraction:

$$f(P) = \frac{P}{N}$$

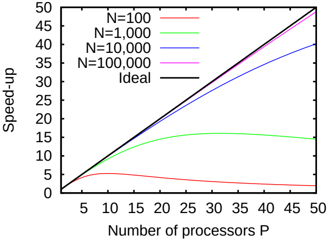


Trying to beat Amdahl's law #1

Scale up!

The larger N, the smaller the serial fraction:

$$f(P) = \frac{P}{N}$$



Weak scaling: Increase problem size while increasing P

$$Time_{weak}(P) = Time(N = n \times P, P)$$

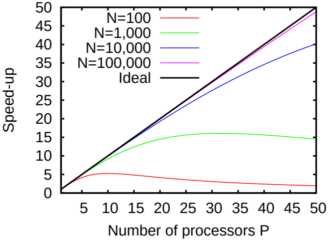
Good weak scaling means this time approaches a constant for large P.

Trying to beat Amdahl's law #1

Scale up!

The larger N, the smaller the serial fraction:

$$f(P) = \frac{P}{N}$$



Weak scaling: Increase problem size while increasing P

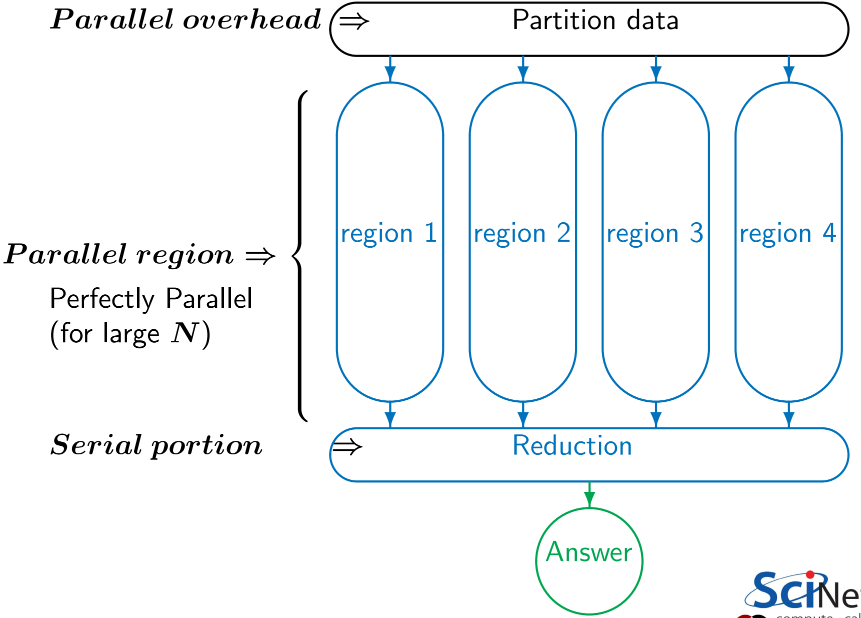
$$Time_{weak}(P) = Time(N = n \times P, P)$$

Good weak scaling means this time approaches a constant for large P.

Gustafson's Law

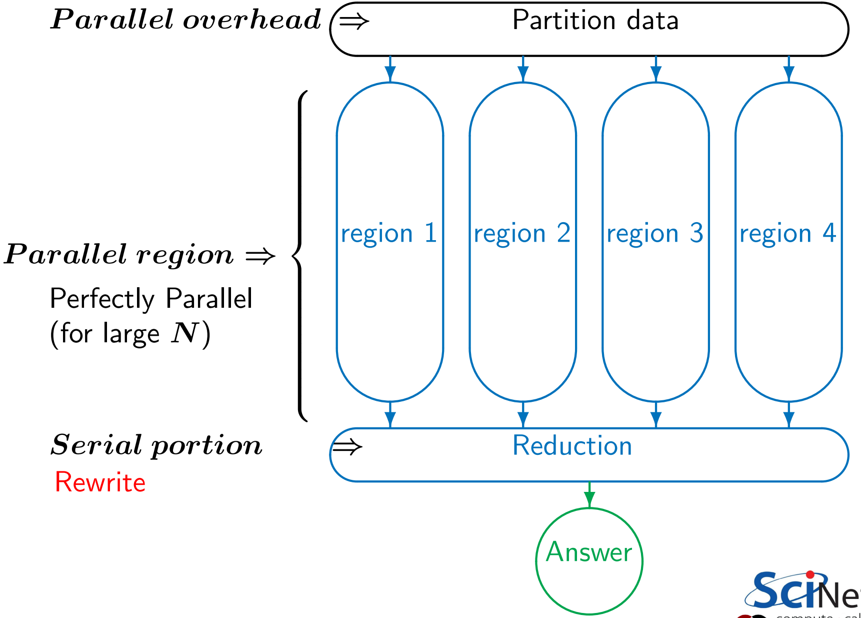
Any large enough problem can be efficiently parallelized (Efficiency$\longrightarrow$1).

Trying to beat Amdahl's law #2



SciNet compute • calcul CANADA

Trying to beat Amdahl's law #2



Trying to beat Amdahl's law #2

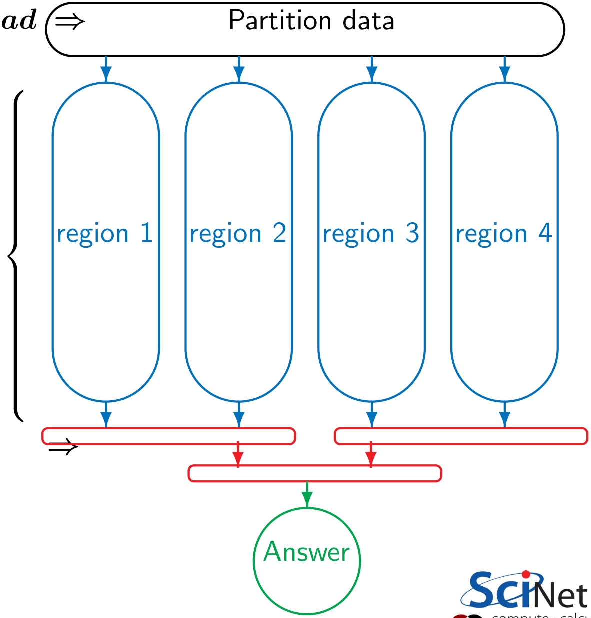
Parallel overhead ⇒

Parallel region ⇒

Perfectly Parallel
(for large N)

Serial portion ⇒

Rewrite



Trying to beat Amdahl's law #2

Parallel overhead ⇒

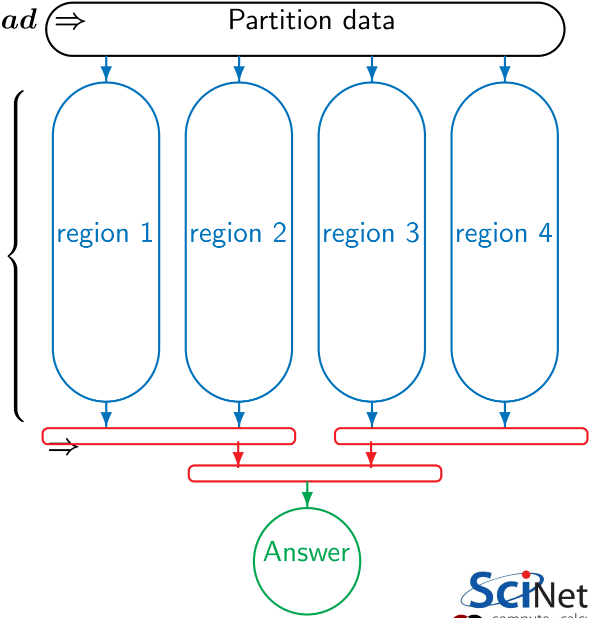
Parallel region ⇒

Perfectly Parallel
(for large N)

Serial portion ⇒

Rewrite

$\propto \log_2(P)$

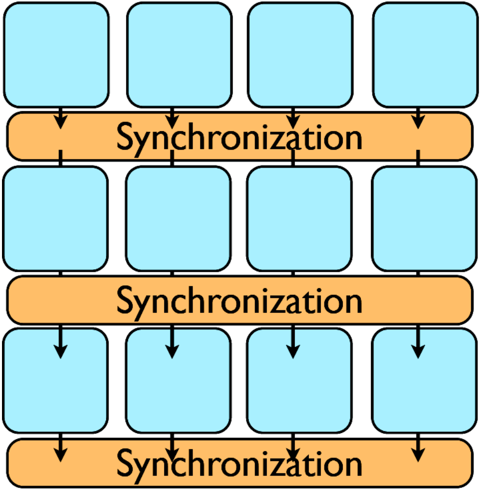


HPC Lesson #2

Optimal Serial Algorithm for your problem may not be the P $\longrightarrow$1 limit of your optimal parallel algorithm.

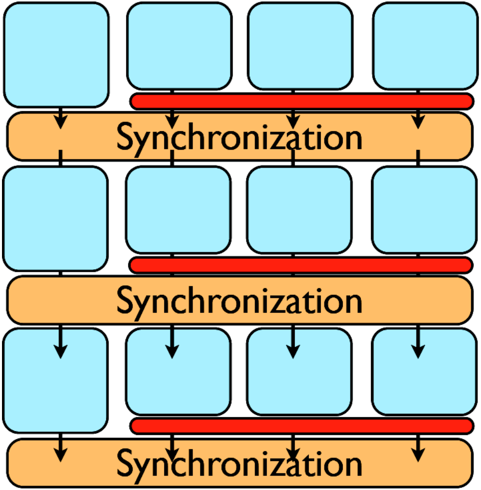
Synchronization

- Most problems are not purely concurrent.
- Some level of synchronization or exchange of information is needed between tasks.
- While synchronizing, nothing else happens: increases Amdahl's f.
- And synchronizations are themselves costly.

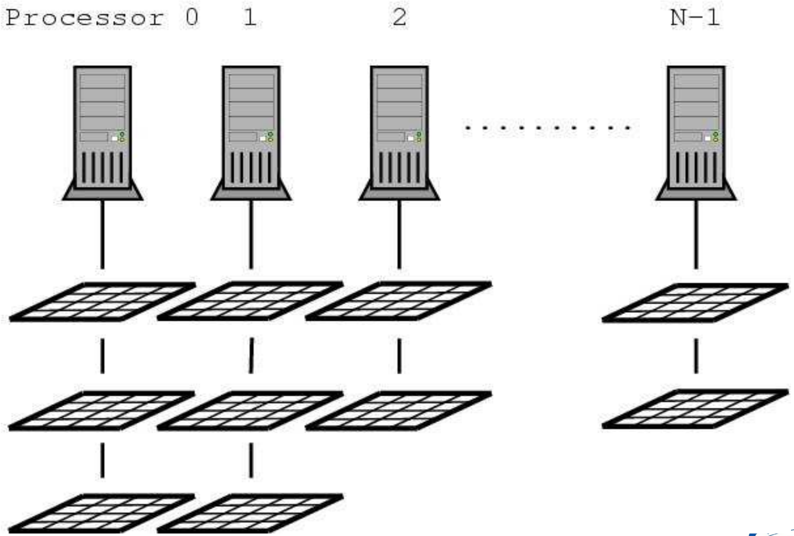


Load Balancing

- The division of calculations among the processors may not be equal.
- Some processors would already be done, while others are still going.
- Effectively using less than P processors: This reduces the efficiency.
- Aim for load balanced algorithms.



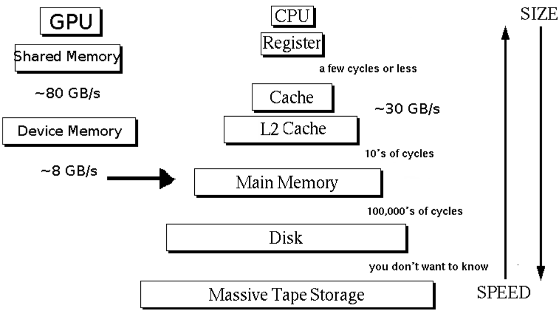
Load Balancing



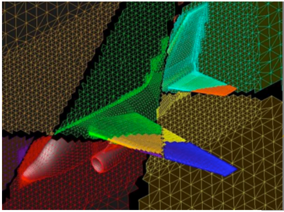
Locality

- So far we neglected communication costs.
- But communication costs are more expensive than computation!
- To minimize communication to computation ratio:
  * Keep the data where it is needed.
  * Make sure as little data as possible is to be communicated.
  * Make shared data as local to the right processors as possible.
- Local data means less need for syncs, or smaller-scale syncs.
- Local syncs can alleviate load balancing issues.

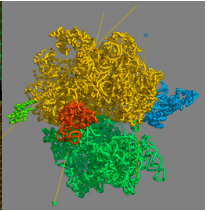
Locality


GPU
Shared Memory
~80 GB/s
Device Memory
~8 GB/s
CPU
Register
a few cycles or less
Cache
~30 GB/s
L2 Cache
10's of cycles
Main Memory
100,000's of cycles
Disk
you don't want to know
Massive Tape Storage
SIZE
SPEED

Domain Decomposition



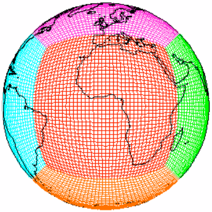
http://adg.stanford.edu/aa241/design/compaero.html



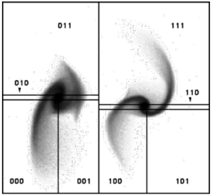
http://www.uea.ac.uk/cmp/research/cmpbio/Protein+Dynamics,+Structure+and+Function

Domain Decomposition

- A very common approach to parallelizing on distributed memory computers
- Maintain Locality; need local data mostly, this means only surface data needs to be sent between processes.



http://sivo.gsfc.nasa.gov/cubedsphere_comp.html

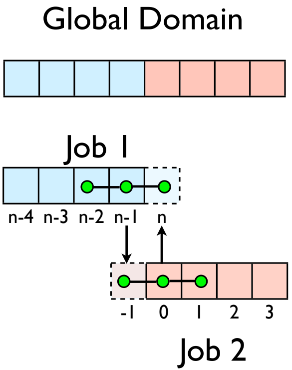


http://www.cita.utoronto.ca/~dubinski/treecode/node8.html

Domain Decomposition

Guardcells

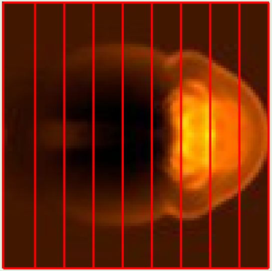
- Works for parallel decomposition!
- Job 1 needs info on Job 2s 0th zone, Job 2 needs info on Job 1s last zone
- Pad array with ‘guardcells’ and fill them with the info from the appropriate node by message passing or shared memory



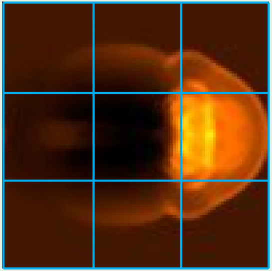
Locality

Example (PDE Domain decomposition)

wrong



right



HPC Lesson #3

Parallel algorithm design is about finding as much concurrency as possible, and arranging it in a way that maximizes locality.

Outline

HPC Systems

Top500.org:

List of the worlds 500 largest supercomputers. Updated every 6 months,

Info on architecture, etc.

Top500 List - November 2013

R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

R_{peak} values are for the normal CPU clock rate. For the effeciency of the systems you should take the Turbu CPU clock rate into account.

previous | 1 | 2 | 3 | 4 | 5 | next

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115984	6271.0	7788.9	2325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462462	5168.1	8520.1	4510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458752	5008.9	5872.0	2301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393216	4293.3	5033.2	1972
10	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147456	2897.0	3185.1	3423

HPC Systems

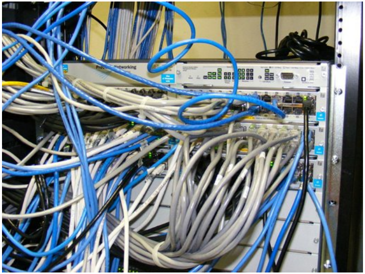
Architectures

- Clusters, or, distributed memory machines
  - A bunch of servers linked together by a network ("interconnect").
  - commodity x86 with gigE, Cray XK, IBM BGQ
- Symmetric Multiprocessor (SMP) machines, or, shared memory machines
  - These can all see the same memory, typically a limited number of cores.
  - IBM Pseries, Cray SMT, SGI Altix/UV
- Vector machines.
  - No longer dominant in HPC anymore.
  - Cray, NEC
- Accelerator (GPU, Cell, MIC, FPGA)
  - Heterogeneous use of standard CPU's with a specialized accelerator.
  - NVIDIA, AMD, Intel, Xilinx

Distributed Memory: Clusters

Simplest type of parallel computer to build

- Take existing powerful standalone computers
- And network them



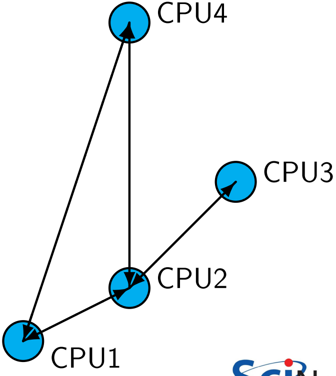
(source: http://flickr.com/photos/eurleif)

Distributed Memory: Clusters

Each node is independent!

Parallel code consists of programs running on separate computers, communicating with each other.

Could be entirely different programs.

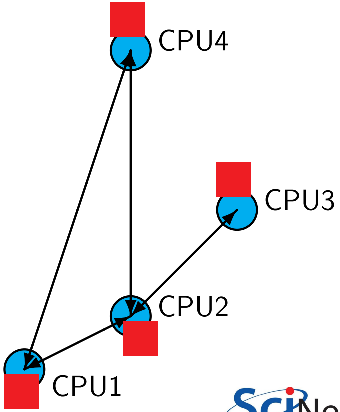


Distributed Memory: Clusters

Each node is independent!
Parallel code consists of programs running on separate computers, communicating with each other.
Could be entirely different programs.

Each node has own memory!
Whenever it needs data from another region, requests it from that CPU.

Usual model: "message passing"



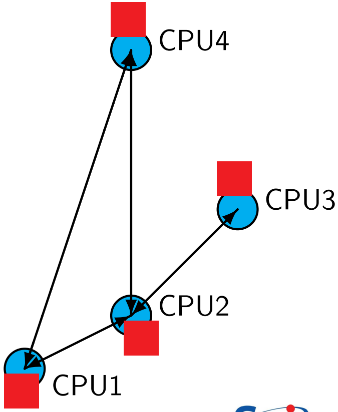
Clusters+Message Passing

Hardware:

Easy to build
(Harder to build well)
Can build larger and larger clusters relatively easily

Software:

Every communication has to be hand-coded: hard to program



Task (function, control) Parallelism

Work to be done is decomposed across processors

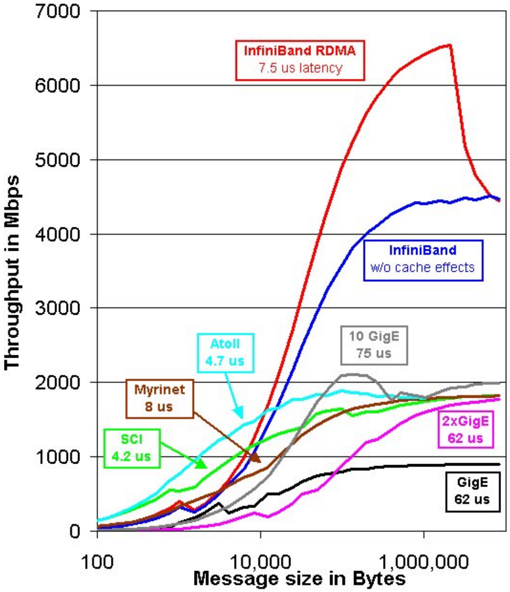
- e.g. divide and conquer
- each processor responsible for some part of the algorithm
- communication mechanism is significant
- must be possible for different processors to be performing different tasks

Cluster Communication Cost

	Latency	Bandwidth
GigE	10 μs (10,000 ns)	1 Gb/s (60 ns/double)
Infiniband	2 μs (2,000 ns)	2-10 Gb/s (10 ns /double)

Processor speed: O(GFLOP) $\sim$ few ns or less.

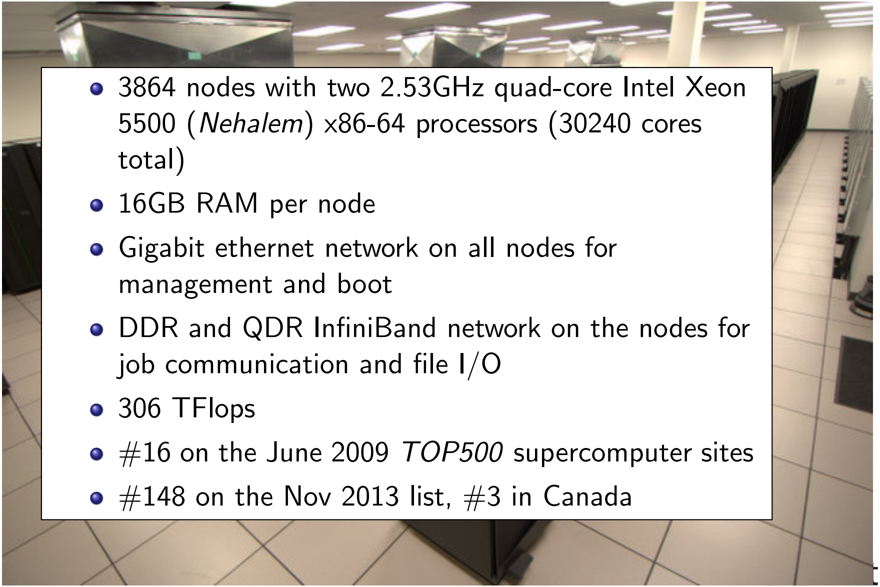
Cluster Communication Cost



SciNet General Purpose Cluster (GPC)

compute • calcul
CANADA

SciNet General Purpose Cluster (GPC)



- 3864 nodes with two 2.53GHz quad-core Intel Xeon 5500 (*Nehalem*) x86-64 processors (30240 cores total)
- 16GB RAM per node
- Gigabit ethernet network on all nodes for management and boot
- DDR and QDR InfiniBand network on the nodes for job communication and file I/O
- 306 TFlops
- #16 on the June 2009 *TOP500* supercomputer sites
- #148 on the Nov 2013 list, #3 in Canada

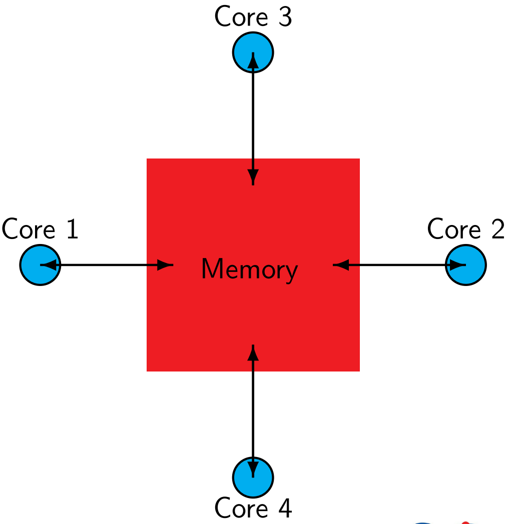
Shared Memory

One large bank of memory, different computing cores acting on it. All 'see' same data.

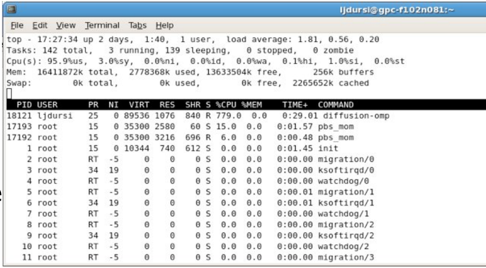
Any coordination done through memory

Could use message passing, but no need.

Each code is assigned a thread of execution of a single program that acts on the data.

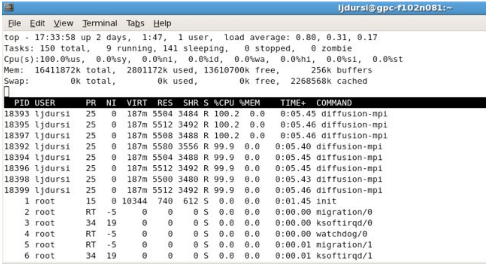


Threads versus Processes



Threads:

Threads of execution within one process, with access to the same memory etc.



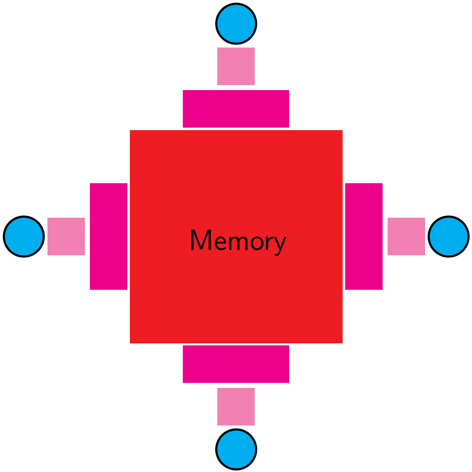
Processes:

Independent tasks with their own memory and resources

Shared Memory: NUMA

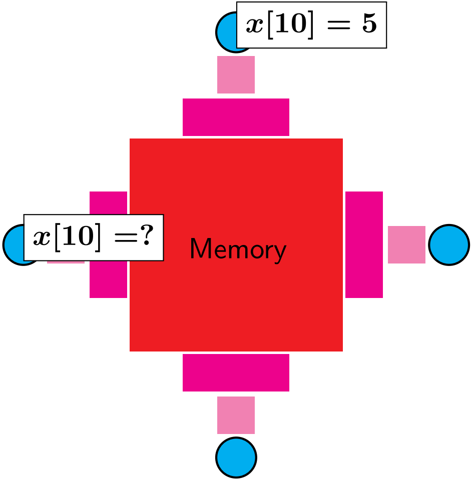
Non-Uniform Memory Access

- Each core typically has some memory of its own.
- Cores have cache too.
- Keeping this memory coherent is extremely challenging.



Coherency

- The different levels of memory imply multiple copies of some regions
- Multiple cores mean can update unpredictably
- Very expensive hardware
- Hard to scale up to lots of processors.
- Very simple to program!!



Data (Loop) Parallelism

Data is distributed across processors

- easier to program, compiler optimization
- code otherwise looks fairly sequential
- benefits from minimal communication overhead
- scale limitations

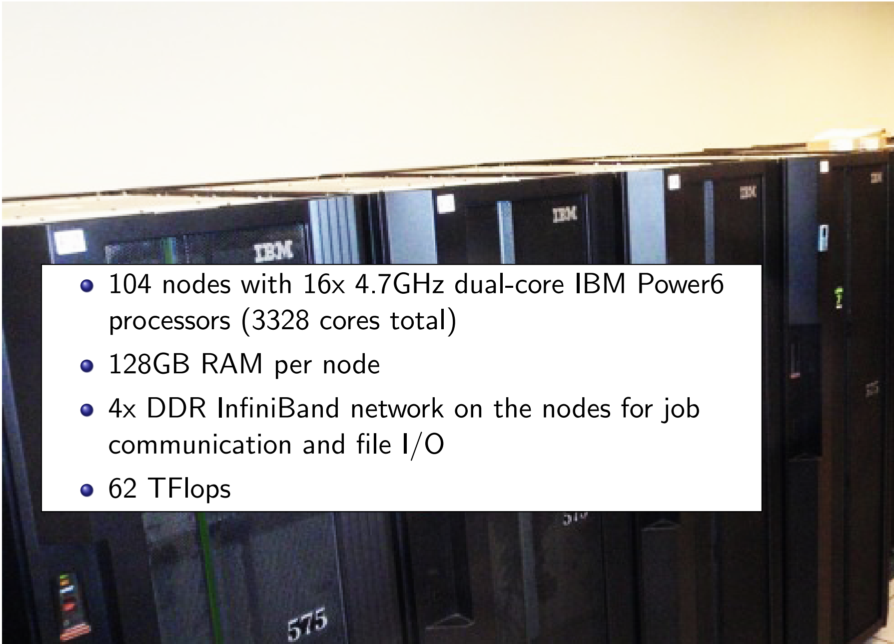
Shared Memory Communication Cost

	Latency	Bandwidth
GigE	10 μs (10,000 ns)	1 Gb/s (60 ns/double)
Infiniband	2 μs (2,000 ns)	2-10 Gb/s (10 ns /double)
NUMA (shared memory)	0.1 μs (100 ns)	10-20 Gb/s (4 ns /double)

Processor speed: O(GFLOP) $\sim$ few ns or less.

SciNet Tightly Coupled System (TCS)

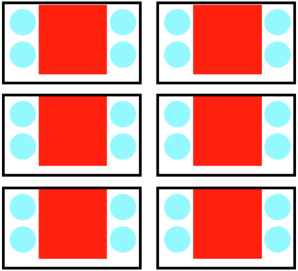
SciNet Tightly Coupled System (TCS)



- 104 nodes with 16x 4.7GHz dual-core IBM Power6 processors (3328 cores total)
- 128GB RAM per node
- 4x DDR InfiniBand network on the nodes for job communication and file I/O
- 62 TFlops

Hybrid Architectures

- Use shared and distributed memory together (i.e. OpenMP with MPI).
- Need to exploit multi-level parallelism.
- Homogeneous
  - Identical multicore machines linked together with an interconnect.
  - Many cores have modest vector capabilities.
  - Thread on-node, MPI for off-node.
- Heterogeneous
  - Same as above, but with an accelerator as well.
  - GPU, Xeon Phi, FPGA.



SciNet compute • calcul CANADA

Heterogeneous Computing

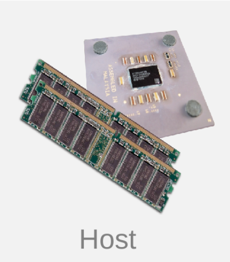
What is it?

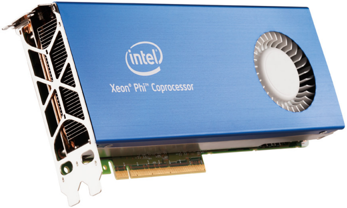
- Use different compute device(s) concurrently in the same computation.
- Commonly using a CPU with an accelator: GPU, Xeon Phi, FPGA, ...
- Example: Leverage CPUs for general computing components and use GPU's for data parallel / FLOP intensive components.
- Pros: Faster and cheaper ($/FLOP/Watt) computation
- Cons: More complicated to program

Heterogeneous Computing

Terminology

- GPGPU : General Purpose Graphics Processing Unit
- HOST : CPU and its memory
- DEVICE : Accelerator (GPU/Phi) and its memory





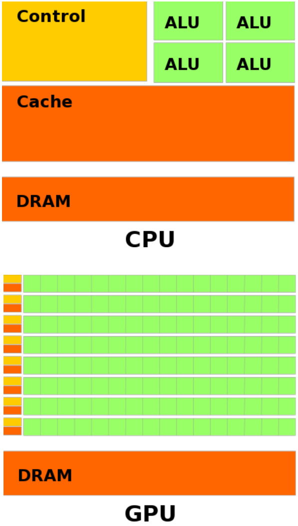
GPU vs. CPUs

CPU

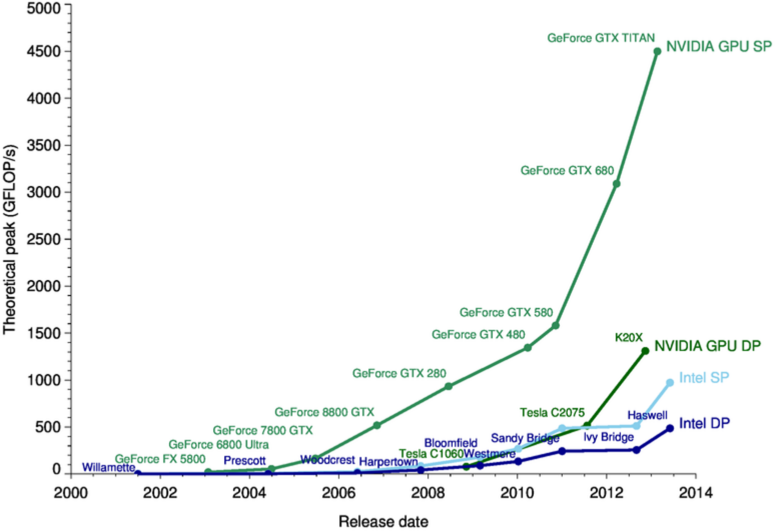
- general purpose
- task parallelism (diverse tasks)
- maximize serial performance
- large cache
- multi-threaded (4-16)
- some SIMD (SSE, AVX)

GPU

- data parallelism (single task)
- maximize throughput
- small cache
- super-threaded (500-2000+)
- almost all SIMD



GPGPU



http://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html

Xeon Phi

What is it?

- Intel x86 based Accelerator/Co-processor
- Many Integrated Cores (MIC) Architecture
- Large number of low-powered, but low cost (computational overhead, power, size, monetary cost) processors (pentiums).

Xeon Phi

What is it?

- Intel x86 based Accelerator/Co-processor
- Many Integrated Cores (MIC) Architecture
- Large number of low-powered, but low cost (computational overhead, power, size, monetary cost) processors (pentiums).

Xeon Phi 5110P (Knights Corner)

- 60 cores @ 1.053GHz
- 8 GB memory
- 4-way SMT (240 threads)
- PCIe Gen2 bus connectivity
- Runs linux onboard

NOTE: Next Gen Knights Landing: 72-core native processor

Heterogeneous Speedup

What kind of speedup can I expect?

- $\sim$1 TFLOPs per GPU vs. $\sim$100 GFLOPs multi-core CPU
- 0x - 50x reported

Heterogeneous Speedup

What kind of speedup can I expect?

- $\sim$1 TFLOPs per GPU vs. $\sim$100 GFLOPs multi-core CPU
- 0x - 50x reported

Speedup depends on

- problem structure
  - need many identical independent calculations
  - preferably sequential memory access
- single vs. double precision (K20 3.52 TF SP vs 1.17 TF DP)
- data locality
- level of intimacy with hardware
- programming time investment

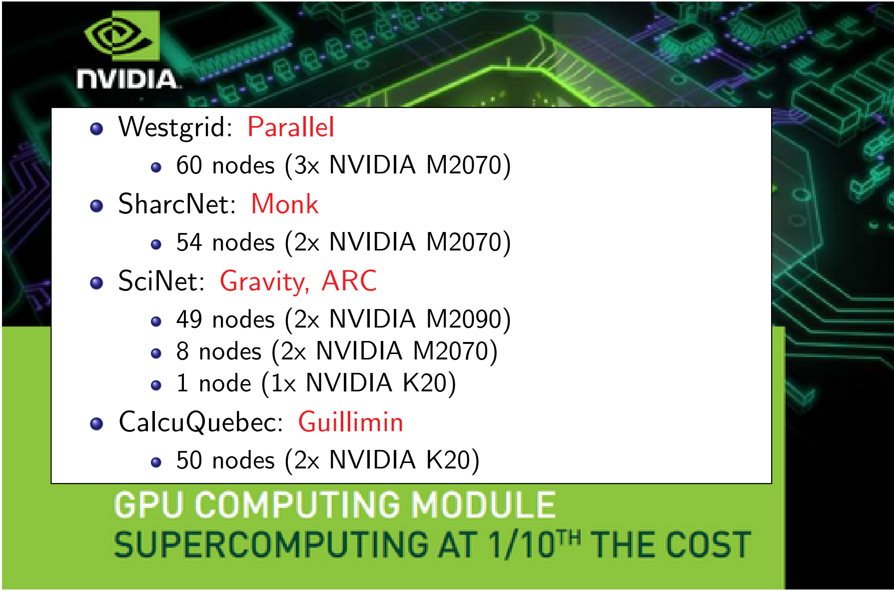
Accelerator Programming

Languages

- GPGPU Only
  - OpenGL, DirectX (Graphics only)
  - CUDA (NVIDIA proprietary)
- OpenCL (1.0, 1.1, 2.0)
- OpenACC
- OpenMP 4.0

Compute Canada GPU Resources

- Westgrid: Parallel
  - 60 nodes (3x NVIDIA M2070)
- SharcNet: Monk
  - 54 nodes (2x NVIDIA M2070)
- SciNet: Gravity, ARC
  - 49 nodes (2x NVIDIA M2090)
  - 8 nodes (2x NVIDIA M2070)
  - 1 node (1x NVIDIA K20)
- CalcuQuebec: Guillimin
  - 50 nodes (2x NVIDIA K20)



Compute Canada Xeon Phi Resources

SciNet - ArcX

- 1 node (1 x 8-core Sandybridge Xeon, 32GB)
- 1 x Intel Xeon Phi 3120A (57 1.1 GHz cores and 6GB)
- `qsub -l nodes=1:ppn=8,walltime=2:00:00 -q arcX -I`
- module load intel/14.0.1 intelmpi/4.1.2.040

Calcu Quebec - Guillimin

- 50 nodes (2 x 8-core Intel Sandy Bridge Xeon, 64GB)
- 2 x Intel Xeon Phi 5110P (60 1.053GHz cores and 8GB)

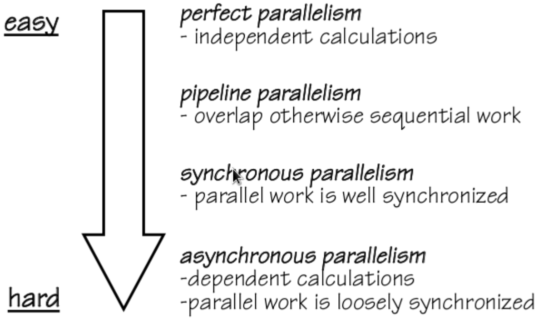
HPC Lesson #4

The best approach to parallelizing your problem will depend on both details of your problem and of the hardware available.

Outline

Program Structure

Structure of the problem dictates the ease with which we can implement parallel solutions easy



Parallel Granularity

Granularity

A measure of the amount of processing performed before communication between processes is required.

Parallelism

- Fine Grained
  - constant communication necessary
  - best suited to shared memory environments
- Coarse Grained
  - significant computation performed before communication is necessary
  - ideally suited to message-passing environments
- Perfect
  - no communication necessary

HPC Programming Models

Languages

- serial
  - C, C++, Fortran
- threaded (shared memory)
  - OpenMP, pthreads
- message passing (distributed memory)
  - MPI, PGAS (UPC, Coarray Fortran)
- accelerator (GPU, Cell, MIC, FPGA)
  - CUDA, OpenCL, OpenACC

HPC Systems

HPC Software Stack

- Typically GNU/Linux
- non-interactive batch processing using a queuing system scheduler
- software packages and versions usually available as “modules”
- Parallel filesystem (GPFS,Lustre)

Outline

SciNet compute • calcul CANADA

Serial Jobs

- SciNet is primarily a parallel computing resource. (Parallel here means OpenMP and MPI, not many serial jobs.)

Serial Jobs

- SciNet is primarily a parallel computing resource. (Parallel here means OpenMP and MPI, not many serial jobs.)
- You should never submit purely serial jobs to the GPC queue. The scheduling queue gives you a full 8-core node. Per-node scheduling of serial jobs would mean wasting 7 cpus.

Serial Jobs

- SciNet is primarily a parallel computing resource. (Parallel here means OpenMP and MPI, not many serial jobs.)
- You should never submit purely serial jobs to the GPC queue. The scheduling queue gives you a full 8-core node. Per-node scheduling of serial jobs would mean wasting 7 cpus.
- Nonetheless, if you can make efficient use of the resources using serial runs and get good science done, that's good too.

Serial Jobs

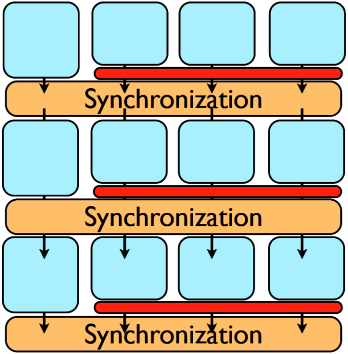
- SciNet is primarily a parallel computing resource. (Parallel here means OpenMP and MPI, not many serial jobs.)
- You should never submit purely serial jobs to the GPC queue. The scheduling queue gives you a full 8-core node. Per-node scheduling of serial jobs would mean wasting 7 cpus.
- Nonetheless, if you can make efficient use of the resources using serial runs and get good science done, that's good too.
- Users need to utilize whole nodes by running at least 8 serial runs at once.

Easy case: serial runs of equal duration

```
#PBS -l nodes=1:ppn=8,walltime=1:00:00
cd $PBS_O_WORKDIR
(cd rundir1; ./dorun1) &
(cd rundir2; ./dorun2) &
(cd rundir3; ./dorun3) &
(cd rundir4; ./dorun4) &
(cd rundir5; ./dorun5) &
(cd rundir6; ./dorun6) &
(cd rundir7; ./dorun7) &
(cd rundir8; ./dorun8) &
wait # or all runs get killed immediately
```

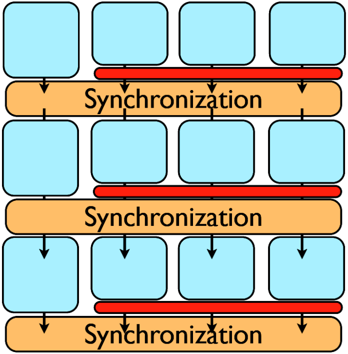
Hard case: serial runs of unequal duration

Different runs may not take the same time: **load imbalance**.



Hard case: serial runs of unequal duration

Different runs may not take the same time: **load imbalance**.



- Want to keep all 8 cores on a node busy.
- Or even 16 virtual cores on a node (HyperThreading).
- ⇒ GNU Parallel can do this

GNU Parallel

- GNU parallel is a a tool to run multiple (serial) jobs in parallel. *As parallel is used within a GPC job, we'll call these* **subjobs**.
- It allows you to keep the processors on each 8-core node busy, if you provide enough subjobs.
- GNU Parallel can use multiple nodes as well.

On the GPC cluster:

- GNU parallel is accessible on the GPC in the module gnu-parallel, which you can load in your .bashrc.

```
$ module load gnu-parallel/20121022
```

- There are currently (Nov 2012) three gnu-parallel modules on the GPC. Although for compatibility gnu-parallel/2010 is the default, we recommend using gnu-parallel/20121022.

GNU Parallel Example

SETUP

- A serial c++ code 'mycode.cc' needs to be compiled.

GNU Parallel Example

SETUP

- A serial c++ code 'mycode.cc' needs to be compiled.
- It needs to be run 32 times with different parameters, 1 through 32.

GNU Parallel Example

SETUP

- A serial c++ code 'mycode.cc' needs to be compiled.
- It needs to be run 32 times with different parameters, 1 through 32.
- The parameters are given as a command line argument.

GNU Parallel Example

SETUP

- A serial c++ code 'mycode.cc' needs to be compiled.
- It needs to be run 32 times with different parameters, 1 through 32.
- The parameters are given as a command line argument.
- 8 subjobs of this code fit into the GPC compute nodes's memory.

GNU Parallel Example

SETUP

- A serial c++ code 'mycode.cc' needs to be compiled.
- It needs to be run 32 times with different parameters, 1 through 32.
- The parameters are given as a command line argument.
- 8 subjobs of this code fit into the GPC compute nodes's memory.
- Each serial run on average takes $\sim$ 2 hour.

GNU Parallel Example

$

GNU Parallel Example

```
$ cd $SCRATCH/example
$
```

GNU Parallel Example

```
$ cd $SCRATCH/example
$ module load intel
$
```

GNU Parallel Example

```
$ cd $SCRATCH/example
$ module load intel
$ icpc -O3 -xhost mycode.cc -o myapp
$
```

GNU Parallel Example

```
$ cd $SCRATCH/example
$ module load intel
$ icpc -O3 -xhost mycode.cc -o myapp
$ cat > subjob.lst
  mkdir run01; cd run01; ../myapp 1 > out
  mkdir run02; cd run02; ../myapp 2 > out
  ...
  mkdir run32; cd run32; ../myapp 32 > out
$
```

GNU Parallel Example

```
$ cd $SCRATCH/example
$ module load intel
$ icpc -O3 -xhost mycode.cc -o myapp
$ cat > subjob.lst
  mkdir run01; cd run01; ../myapp 1 > out
  mkdir run02; cd run02; ../myapp 2 > out
  ...
  mkdir run32; cd run32; ../myapp 32 > out
$ cat > GPJob
     #PBS -l nodes=1:ppn=8,walltime=12:00:00
     cd $SCRATCH/example
     module load intel gnu-parallel/20121022
     parallel --jobs 8 < subjob.lst
$
```

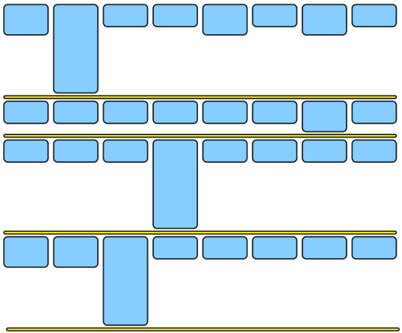
GNU Parallel Example

```
$ cd $SCRATCH/example
$ module load intel
$ icpc -O3 -xhost mycode.cc -o myapp
$ cat > subjob.lst
  mkdir run01; cd run01; ../myapp 1 > out
  mkdir run02; cd run02; ../myapp 2 > out
  ...
  mkdir run32; cd run32; ../myapp 32 > out
$ cat > GPJob
     #PBS -l nodes=1:ppn=8,walltime=12:00:00
     cd $SCRATCH/example
     module load intel gnu-parallel/20121022
     parallel --jobs 8 < subjob.lst
$ qsub GPJob
     2961985.gpc-sched
$
```

GNU Parallel Example

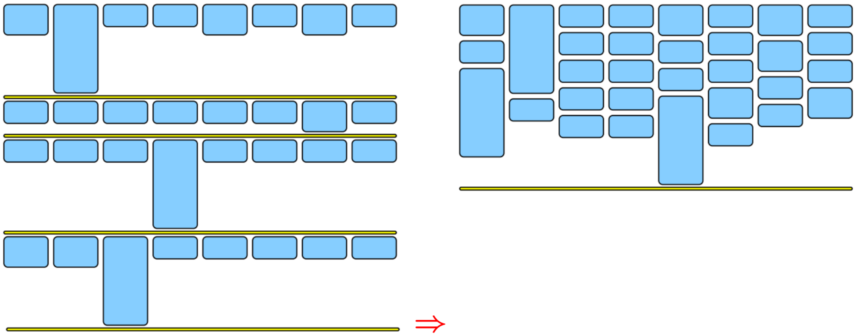
```
$ cd $SCRATCH/example
$ module load intel
$ icpc -O3 -xhost mycode.cc -o myapp
$ cat > subjob.lst
  mkdir run01; cd run01; ../myapp 1 > out
  mkdir run02; cd run02; ../myapp 2 > out
  ...
  mkdir run32; cd run32; ../myapp 32 > out
$ cat > GPJob
    #PBS -l nodes=1:ppn=8,walltime=12:00:00
    cd $SCRATCH/example
    module load intel gnu-parallel/20121022
    parallel --jobs 8 < subjob.lst
$ qsub GPJob
    2961985.gpc-sched
$ ls
    GPJob   GPJob.e2961985   GPJob.o2961985   subjob.lst
    myapp   run01            run02            run03
    ...
```

GNU Parallel Example



17 hours
42% utilization

GNU Parallel Example



17 hours
42% utilization

10 hours
72% utilization

GNU Parallel Details

What else can it do?

- Recover from crashes (joblog/resume options)
- Span multiple nodes

Using GNU Parallel

- wiki.scinethpc.ca/wiki/index.php/User_Serial
- wiki.scinethpc.ca/wiki/images/7/7b/Tech-talk-gnu-parallel.pdf
- www.gnu.org/software/parallel
- www.youtube.com/playlist?list=PL284C9FF2488BC6D1
- O. Tange, GNU Parallel – The Command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.