Practical Parallel Programming Intensive

SciNet HPC Consortium

9-13 May 2011



SciNet HPC Consortium ()

Practical Parallel Programming Intensive

9-13 May 2011 1 / 140

Welcome to the intensive parallel programming course!





Part I

The Course



Main goal of the course

... to enable young researchers already experienced in scientific computing to leave with the knowledge necessary to begin writing the parallel codes needed for their research.

The course will be a mix of lectures and immediate feedback on practical assignments, designed to ensure that students leave with significant experience in both OpenMP and MPI, two of the standards for parallel computing today.

So there'll ne a lot of typing and programming to help build skills with OpenMP and MPI.

We will use C and Fortran. We'll assume that you already know one of them, but not both.



Schedule

Mon May 9	AM	Intro to Parallel Computing, SciNet resources
-	РМ	OpenMP I +hands on
Tue May 10	AM	OpenMP II +hands on
	РМ	MPI I +hands on
Wed May 11	AM	MPI II +hands on
	РМ	Explicit PDEs: Hydrodynamics +hands on
Thu May 12	AM	Particle Methods: N-body +hands on
	РМ	GPU Programming <i>+hands on</i>
Fri May 13	AM	Performance tools & Best practices



Strongly recommended books

(not provided by us)

- B. Chapman, G. Jost and R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming (MIT Press, Cambridge 2008)
- W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, second edition (MIT Press, Cambridge 1999).



Part II

Introduction to Parallel Programming



Why Parallel Programming?



Faster

There's a limit to how fast 1 computer can compute.

So use more computers!



SciNet HPC Consortium ()

9-13 May 2011 8 / 140

Why Parallel Programming?



Faster

There's a limit to how fast 1 computer can compute.

Ø Bigger

There's a limit to how much memory, disk, etc, can be put on 1 computer.

So use more computers!



SciNet HPC Consortium ()

9–13 May 2011 8 / 140

Why Parallel Programming?



Faster

There's a limit to how fast 1 computer can compute.

Ø Bigger

There's a limit to how much memory, disk, etc, can be put on 1 computer.

6 More

Want to do the same thing that was done on 1 computer, but *thousands of times*.

So use more computers!



Why is it necessary?

- Modern experiments and observations yield vastly more data to be processsed than in the past.
- As more computing resources become available (SciNet), the bar for cutting edge simulations is raised.
- Science that before could not even be done becomes reachable.



Why is it necessary?

- Modern experiments and observations yield vastly more data to be processsed than in the past.
- As more computing resources become available (SciNet), the bar for cutting edge simulations is raised.
- Science that before could not even be done becomes reachable.

However:



Why is it necessary?

- Modern experiments and observations yield vastly more data to be processsed than in the past.
- As more computing resources become available (SciNet), the bar for cutting edge simulations is raised.
- Science that before could not even be done becomes reachable.

However:

- Advances in clock speeds, bigger and faster memory and disks have been lagging as compared to e.g. 10 years ago. *Can no longer "just wait a year" and get a better computer.*
- So more computing resources here means: more cores running concurrently.
- Even most laptops now have 2 or more cpus.
- So parallel computing is necessary.



Wait, what about Moore's Law?



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

pute • calcul

Wait, what about Moore's Law?





Wait, what about Moore's Law?



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?

throughput
$$=$$
 H $=$ $rac{{\sf N}}{{\sf T}}$

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



Scaling — Throughput

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



• 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 "embarrasingly 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 "embarrasingly parallel".



Scaling – Speedup

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

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

• For embarrasingly parallel applications: Linear speed up.



- 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"













Speed-up (without parallel overhead):

$$\mathsf{S} = \frac{\mathsf{N}\mathsf{T}_1 + \mathsf{T}_{\mathsf{s}}}{\frac{\mathsf{N}\mathsf{T}_1}{\mathsf{P}} + \mathsf{T}_{\mathsf{s}}}$$

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

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



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,

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



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,

$$S = rac{1}{f + (1 - f)/P} \stackrel{P o \infty}{\longrightarrow} rac{1}{f}$$



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,

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



Scaling efficiency

Speed-up compared to ideal factor **P**:

Efficiency = $\frac{S}{P}$

This will invariably fall off for larger P except for embarrasing parallel problems.

$$\text{Efficiency} \sim \frac{1}{\text{fP}} \stackrel{\text{P} \rightarrow \infty}{\longrightarrow} 0$$

You cannot get 100% efficiency in any non-trivial problem.

All you can aim for here is to make the efficiency as least low as possible. Sometimes, that can mean running on less processors, but more problems at the same time.



Timing example

- Say 100s in integration cost
- 5s in reduction
- Neglect communication cost
- What happens as we vary number of processors P?



Time = (100s)/P + 5

Throughput example

$$H(P) = \frac{N}{Time(P)}$$

• Say we are doing **k** at the same time, on **P** processors total.

$$H_k(P) = rac{kN}{Time(P/k)}$$



SciNet HPC Consortium ()

Practical Parallel Programming Intensive

9–13 May 2011 22 / 140
Big 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.



Less ideal case of Amdahl's law

We assumed reduction is constant. But it will in fact increase with P, from sum of results of all processors

$T_s\approx PT_1$

Serial fraction now a function of **P**:

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

Amdahl:

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

COMPUTE • CAICU

Less ideal case of Amdahl's law

We assumed reduction is constant. But it will in fact increase with P, from sum of results of all processors

$T_s\approx PT_1$

Serial fraction now a function of **P**:

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

Amdahl:

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

Example: N = 100, $T_1 = 1s$...

COMPUTE • CAICUL

Less ideal case of Amdahl's law

We assumed reduction is constant. But it will in fact increase with P, from sum of results of all processors

$T_s \approx PT_1$ 5.5 5 4.5 Serial fraction now a function of **P**: 4 Time (s) Speed-up 3.5 $f(P) = \frac{P}{N}$ 3 2.5 2 Amdahl: 1.5 1 10 15 20 25 30 35 40 45 50 5 $S(P) = \frac{I}{f(P) + [1 - f(P)]/P}$ Example: $N = 100, T_1 = 1s...$ ipute • caicUl

CANADA

24 / 140

Scale up!

The larger **N**, the smaller the serial fraction:

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





25 / 140

9-13 May 2011

Scale up!

The larger \mathbf{N} , the smaller the serial fraction:

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



Weak scaling: Increase problem size while increasing P

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

Good weak scaling means this time approaches a constant for large \mathbf{P} .



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 \mathbf{P} .

Gustafson's Law

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

SciNet HPC Consortium ()









'Serial' fraction now different function of **P**:

$$f(\mathsf{P}) = \frac{2\log\mathsf{P}}{\mathsf{N}}$$

Amdahl:

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



'Serial' fraction now different function of \mathbf{P} :

$$f(\mathsf{P}) = \frac{2\log\mathsf{P}}{\mathsf{N}}$$

Amdahl:

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

Example: N = 100, $T_1 = 1s...$







Weak scaling

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

Should approach constant for large **P**. Let's see...

Weak scaling

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

Should approach constant for large **P**. Let's see... Not quite!



9-13 May 2011

Weak scaling

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

Should approach constant for large **P**. Let's see... Not quite! But much better than before.



Practical Parallel Programming Intensive

9-13 May 2011

Weak scaling

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

Should approach constant for large **P**. Let's see... Not quite! But much better than before.



9–13 May 2011

28 / 140

Weak scaling

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

Should approach constant for large **P**. Let's see... Not quite! But much better than before.

Gustafson?

It turns out that Gustafson's law assumes that the serial cost does not change with \mathbf{P} .

Here that grows logarithmically with

 ${\bf P},$ and this is reflected in the weak scaling.



Time (s)

Weak scaling

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

Should approach constant for large **P**. Let's see... Not quite! But much better than before.

Gustafson?

It turns out that Gustafson's law assumes that the serial cost does not change with \mathbf{P} .



Really not that bad.

Here that grows logarithmically with *and other algorithms can do better*. **P**, and this is reflected in the weak scaling.

9–13 May 2011

28 / 140

Big Lesson #2

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



29 / 140

9-13 May 2011

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.





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

- 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.

Example (PDE Domain decomposition)





Big Lesson #3

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



33 / 140

9-13 May 2011

Parallel Computers



Top500.org:

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

Info on architecture, etc.

Rmax and Rpeak values are in TFlops. For more details about other fields, check the TOP500 description.

Power data in KW for entire system

TOP500 List - November 2010 (1-100)

Home Lists November 2010

Rank	Site	Computer/Year Vendor	Cores	Rmax	Rpeak	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.0
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	6950.6
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	2580.0
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	1398.6
5	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.0
6	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.0



next

34 / 140

Supercomputer architectures

- Clusters, or, distributed memory machines
 In essence a bunch of desktops linked together by a network
 ("interconnect"). Easy and cheap.
- Multi-core machines, or, shared memory machines These can see the same memory. Limited number of cores, typically, and much more \$\$\$.
- Vector machines.

These were the early supercomputers, and could do the same operation on a large number of numbers at the same time. Very \$\$\$\$\$, especially at scale.

These days, most chips have some low-level, small size vectorization, but you rarely need to worry about it (compiler should do this).

Most supercomputers are a hybrid combo of these different architectures.



Distributed Memory: Clusters

Simplest type of parallel computer to build

- Take existing powerful standalone computers
- And network them



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



Cluster Communication Cost

	Latency	Bandwidth
GigE	10 <i>µ</i> s	1 Gb/s
	(10,000 ns)	(60 ns/double)
Infiniband	2 µ s	2-10 Gb/s
	(2,000 ns)	(10 ns /double)

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



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

Eile										
Eile										ijdursi@gpc-f102ni
	Edit View	Term	ninal	Tabs	Help					
top -	17:27:34	up 2	da	vs. 1	:40.	1 us	er	, loa	d aver	rage: 1.81, 0.56, 0.20
Tasks	: 142 tot	al.	3	runnin	a. 13	9 sle	ep	ing.	0 sto	opped. 0 zombie
Cpu(s): 95.9%u	s. 3	.0%	sv. 0	.0%ni	. 0.1	0%	id. 0	. 0%wa.	. 0.1%hi. 1.0%si. 0.0%st
Mem:	16411872	k tot	al.	2778	368k	used.	1	363350	4k fre	ee. 256k buffers
Swap:	θ	k tot	al,		θk	used,			0k fre	ee, 2265652k cached
Π										
PID	USER	PR	NI	VIRT	RES	SHR	s	%CPU	WEM	TIME+ COMMAND
18121	ljdursi	25	Θ	89536	1076	840	R	779.0	0.0	0:29.01 diffusion-omp
17193	root	15	Θ	35300	2580	60	S	15.0	0.0	0:01.57 pbs mom
17192	root	15	Θ	35300	3216	696	R	6.0	0.0	0:00.48 pbs mom
1	root	15	0	10344	740	612	S	0.0	0.0	0:01.45 init
2	root	RT	-5	0	0	Θ	s	0.0	0.0	0:00.00 migration/0
3	root	34	19	Θ	0	Θ	s	0.0	0.0	0:00.00 ksoftirgd/0
4	root	RT	- 5	0	0	Θ	S	0.0	0.0	0:00.00 watchdog/0
5	root	RT	- 5	0	0	Θ	s	0.0	0.0	0:00.01 migration/1
6	root	34	19	0	0	Θ	S	0.0	0.0	0:00.01 ksoftirgd/1
7	root	RT	- 5	Θ	0	Θ	S	0.0	0.0	0:00.00 watchdog/1
8	root	RT	- 5	Θ	0	Θ	s	0.0	0.0	0:00.00 migration/2
9	root	34	19	0	0	Θ	s	0.0	0.0	0:00.00 ksoftirgd/2
10	root	RT	-5	Θ	0	Θ	s	0.0	0.0	0:00.00 watchdog/2
11	root	RT	- 5	Θ	0	Θ	s	0.0	0.0	0:00.00 migration/3
							-			lidural@are \$102ac
										ijdursi@gpc-r102nt
Ele I	dit View	Term	inal	Tabs	Help					
top -	17:33:58	up 2	day	/s, 1:	47,	1 use	er,	load	d aver	age: 0.80, 0.31, 0.17
fasks:	150 tota	11,	9 1	unning	, 14	1 slee	pi	ng,	0 sto	opped, 0 zombie
Cpu(s)	:100.0%us	, Θ	. 0%5	y, 0.	0%n1	. 0.6	1%t	d, 0.	θ%wa,	0.0%hi, 0.0%si, 0.0%st
	164118728									
fem:	X012X0721	tota	al,	28011	72k 1	used,	13	610706	k fre	e, 256k buffers
Mem: Swap:	01	tota	al, al,	28011	72k u 0k u	used, used,	13	610700	0k fre 0k fre	e, 256k buffers e, 2268568k cached
Mem: Swap:]	01	tota tota	al, al,	28011	0k (used, used,	13	610700	0k fre 0k fre	e, 256k buffers e, 2268568k cached
Mem: Swap: PID	USER	tota tota	al, al, NI	28011 VIRT	72k (0k (used, used, SHR	13	610700 ()k fre)k fre	e, 256k buffers e, 2268568k cached TINE+ CONNAND
Mem: Swap: PID 18393	0) USER ljdursi	R tota PR 25	al, al, NI	28011 VIRT 187m	72k (0k (RES 5504	used, used, SHR 3484	13 S R	610706	0k fre 0k fre 0kEM	ee, 256k buffers ee, 2268568k cached TINE+ CONMAND 0:05.45 diffusion-mpi
Mem: 5wap: 910 18393 18395	USER ljdursi ljdursi	PR 25 25	al, al, NH 0 0	28011 VIRT 187m 187m	72k 0 0k 0 RES 5504 5512	used, used, SHR 3484 3492	S R R	610706 6 100.2 100.2	0k fre 0k fre 0.0 0.0 0.0	e, 256k buffers e, 2268568k cached TIME+ CONVAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi
Mem: Swap: PID 18393 18395 18397	USER ljdursi ljdursi ljdursi	PR 25 25 25 25	Al, Al, NI 0 0 0	28011 VIRT 187m 187m 187m	72k 0 0k 0 5504 5512 5508	sed, sed, 3484 3492 3488	S R R R	CPU (100.2 100.2 100.2	0k fre 0k fre 0.0 0.0 0.0 0.0	e, 256k buffers e, 2268568k cached THNEC COMMAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi
Mem: Swap: PID 18393 18395 18397 18392	USER ljdursi ljdursi ljdursi ljdursi ljdursi	e tota tota 25 25 25 25 25	al, al, NI 0 0 0 0	28011 187m 187m 187m 187m	72k 0 0k 0 5504 5512 5508 5580	3484 3488 3488 3488 3556	R R R R	CPU 0 100.2 100.2 100.2 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0	ee, 256k buffers ee, 2268568k cached THME COMMAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi
Mem: Swap: 18393 18395 18397 18392 18394	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi	PR 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m	72k 0 0k 0 5504 5512 5508 5580 5580 5580	3484 3482 3488 3488 3556 3488	R R R R R R R	610700 CPU 6 100.2 100.2 100.2 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0	e, 256k buffers e, 2268568k cached TINE+ CONMAND 0:05.45 diffusion-mp1 0:05.46 diffusion-mp1 0:05.40 diffusion-mp1 0:05.40 diffusion-mp1
Mem: Swap: PID 18393 18395 18395 18397 18392 18394 18396	0) USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi	PR 25 25 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m	72k 0 0k 0 5504 5512 5508 5580 5580 5580 5584 5512	sed, sed, 3484 3492 3488 3556 3488 3556 3488 3492	S R R R R R R R R R	610700 000.2 100.2 100.2 99.9 99.9 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0	e. 256k buffers e. 2268568k cached TIME COMMAND 0:05.45 dlffusion-mpi 0:05.46 dlffusion-mpi 0:05.46 dlffusion-mpi 0:05.45 dlffusion-mpi 0:05.45 dlffusion-mpi
Mem : Swap : 18393 18395 18395 18397 18392 18394 18396 18398	0H USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi	e tota tota 25 25 25 25 25 25 25 25 25 25 25	Al, al, 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 18	72k 0 0k 0 5504 5512 5508 5508 5580 5580 5504 5512 5500	SHR 3484 3492 3488 3556 3488 3492 3488 3492 3480	R R R R R R R R R R R R R R R R R R R	CPU 0 00.2 100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	e, 256k buffers e, 2568568k cached THEL COUVAID 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi
Mem: Swap: 18393 18395 18395 18395 18392 18394 18396 18398 18399	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi	Example 25 25 25 25 25 25 25 25 25 25 25 25 25 2	al, al, 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 18	72k 0 0k 0 5504 5512 5508 5580 5580 5580 5512 5500 5512	sed, ised, 3484 3492 3488 3556 3488 3492 3480 3492 3480 3492	R R R R R R R R R R R R R R R R R R R	8610706 84CPU 8 106.2 106.2 106.2 99.9 99.9 99.9 99.9 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	ie, 256k buffers ie, 2268568k cached THNE COWAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.45 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi
Mem: Swap: PID 18393 18395 18395 18394 18396 18398 18398 18399 1	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi root	25 25 25 25 25 25 25 25 25 25 25 25 25 2	al, al, 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 18	72k 0 0k 0 5504 5512 5508 5580 5580 5580 5512 5500 5512 740	SHR 3484 3492 3488 3556 3488 3492 3480 3492 3480 3492 612	R R R R R R R R R R R R R R R R R R R	*CPU * *CPU * 100.2 100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	e, 256k buffers e, 2268566k cached TIVE CONVAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi
Mem: Swap: PHD 18393 18395 18395 18394 18396 18398 18398 18399 1 2	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi root	e tot: 25 25 25 25 25 25 25 25 25 25	Al, Al, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 18	72k 0 0k 0 5504 5512 5508 5504 5504 5504 5512 5500 5512 5500 5512 740 0	SHR 3484 3492 3488 3556 3488 3492 3480 3492 3480 3492 612 0	S R R R R R R R R R R R S S	*CPU * *CPU * 100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 99.9	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	ie, 256k buffers ie, 2568568k cached THNE COWAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.49 diffusion-mpi 0:05.45 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi
Mem: Swap: 18393 18395 18395 18395 18394 18396 18398 18399 1 1 2 3	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi root root root	PR 25 25 25 25 25 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 10344 0 0	72k 0 0k 0 5504 5512 5508 5504 5512 5508 5504 5512 5500 5512 740 0 0	ISEd, ISEd, 3484 3492 3488 3556 3488 3492 3480 3492 612 0 0	S R R R R R R R R R R R R S S	8610700 800210 100.2 100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 99.9 0.0 0.0	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	e, 256k buffers e, 226856k cached TIVE COMMAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.90 migration/0 0:00.00 kottira/0
Mem: Swap: PID 18393 18395 18395 18396 18398 18398 18399 1 2 3 4	USER ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi ljdursi root root root	PR 25 25 25 25 25 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 0344 0 0 0 0	72k 0 0k 0 5504 5512 5508 5508 5504 5512 5500 5512 5500 5512 740 0 0 0 0	ISEd, ISEd, 3484 3492 3488 3556 3488 3492 3480 3492 612 0 0 0 0	S R R R R R R R R R R R R R S S S	610700 80020 8 100.2 100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 99.9 0.0 0.0	0k fre 0k fre 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.	e, 256k buffers e, 226568k cached 11kt CC02AND 0:55.45 diffusion-mpi 0:55.46 diffusion-mpi 0:55.46 diffusion-mpi 0:55.45 diffusion-mpi 0:55.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.45 diffusion-mpi 0:56.56 diffusion-mpi 0:50.69 ksoftirqq/0
Mem: 5wap: 18393 18395 18395 18394 18396 18398 18399 1 18399 1 2 3 4 5	0) USER Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi root root root root root	e tota tota 25 25 25 25 25 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 0 0 0 0 0 0 0	72k 0 0k 0 5504 5512 5508 5504 5512 5508 5504 5512 5500 5512 740 0 0 0 0 0	SHR 3484 3492 3488 3556 3488 3492 3492 3492 3492 612 0 0 0 0 0 0	S R R R R R R R R R R R R R S S S S S	CPU (100.2 100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 99.9 0.0 0.0	0k fre 0k fre 0.0	e, 256k buffers e, 2568568k cached TIVE COMMAND 0:05.43 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:05.43 diffusion-mpi 0:06.98 diffusion-mpi 0:08.98 diffusion-mpi
Mem: 5wap: 18393 18395 18395 18394 18394 18396 18398 18399 1 2 3 4 5 6	01 USER Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Ujdursi Toot root root root root root	e tota tota 25 25 25 25 25 25 25 25 25 25 25 25 25	al, al, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	28011 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 187m 0 0 0 0 0 0 0 0 0 0 0 0 0	72k 0 0k 0 5504 5512 5508 5504 5512 5508 5504 5512 5500 5512 740 0 0 0 0 0 0 0	SHR 3484 3492 3488 3556 3488 3492 3480 3492 612 0 0 0 0 0 0 0 0 0 0	S R R R R R R R R R R R R R R S S S S S	CPU (100.2 100.2 99.9 99.9 99.9 99.9 99.9 99.9 0.0 0.0	0k free 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	e, 256k buffers e, 2268568k cached 114EF COMPAND 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.46 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.45 diffusion-mpi 0:05.46 diffusion-mpi 0:05.45 diffusion-mpi 0:05.05 diffusion/0 0:06.01 migration/1



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 \$\$\$
- Very simple to program!!





Shared Memory Communication Cost

	Latency	Bandwidth
GigE	10 <i>µ</i> s	1 Gb/s
	(10,000 ns)	(60 ns/double)
Infiniband	$2 \ \mu s$	2-10 Gb/s
	(2,000 ns)	(10 ns /double)
NUMA	0.1 μ s	10-20 Gb/s
(shared memory)	(100 ns)	(4 ns /double)

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



Hybrid Architectures

- Multicore machines linked together with an interconnect
- Many cores have modest vector capabilities.
- Machines with GPU: GPU is multi-core, but the amount of shared memory is limited.

We will focus on the aspects that affect the programmer:

- Shared memory: OpenMP
- Distributed memory: MPI
- Graphics computing: CUDA, OpenCL





Big Lesson #4

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

