Parallel Programming & Cluster Computing

Overview of Parallelism

Henry Neeman, University of Oklahoma
Paul Gray, University of Northern Iowa

SC08 Education Program's Workshop on Parallel & Cluster computing
Oklahoma Supercomputing Symposium, Monday October 6 2008

Outline

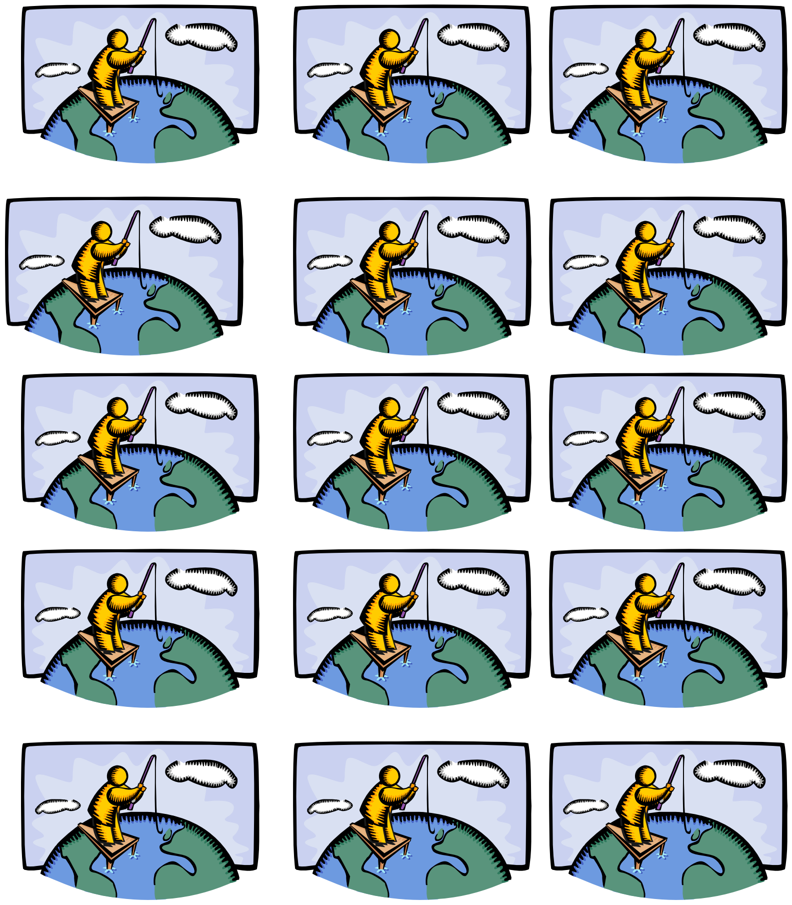
- Parallelism
- The Jigsaw Puzzle Analogy for Parallelism
- The Desert Islands Analogy for Distributed Parallelism
- Parallelism Issues

Parallelism

Parallelism

Parallelism means doing multiple things at the same time: you can get more work done in the same time.

Less fish …



More fish!

What Is Parallelism?

- ***<u>Parallelism</u>*** is the use of multiple processors to solve a problem, and in particular the use of multiple processors working concurrently on different parts of a problem.
- The different parts could be different tasks, or the same tasks on different pieces of the problem's data.

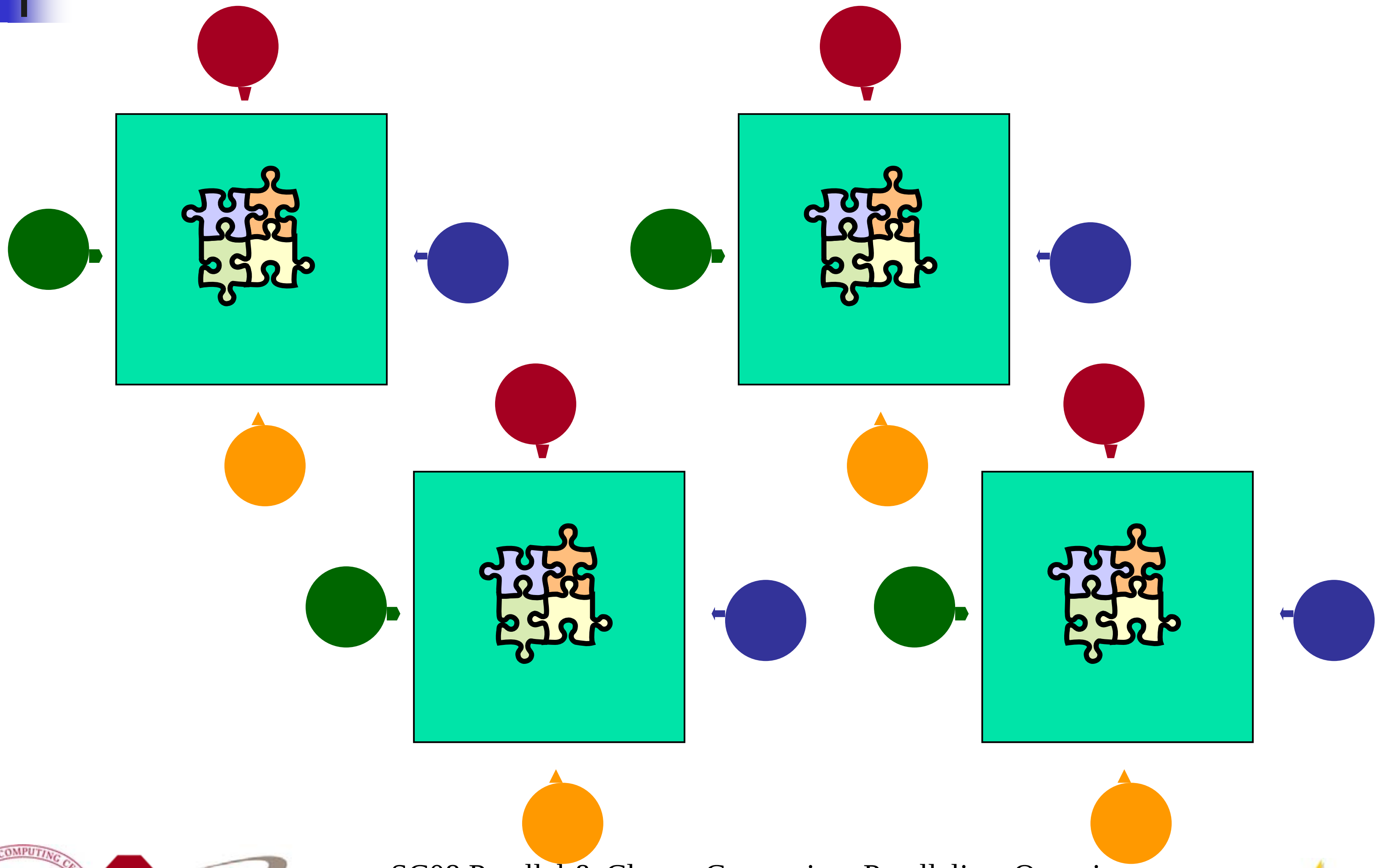
Why Parallelism Is Good

- **The Trees**: We like parallelism because, as the number of processing units working on a problem grows, we can solve **the same problem in less time**.
- **The Forest**: We like parallelism because, as the number of processing units working on a problem grows, we can solve **bigger problems**.

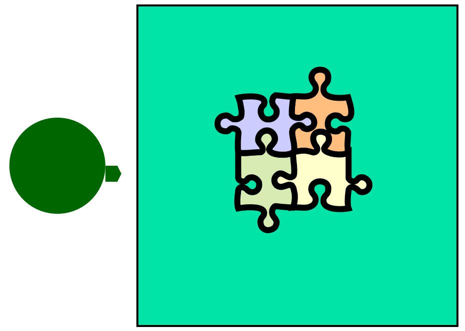
Kinds of Parallelism

- Shared Memory Multithreading
- Distributed Memory Multiprocessing
- Hybrid Shared/Distributed Parallelism

The Jigsaw Puzzle Analogy



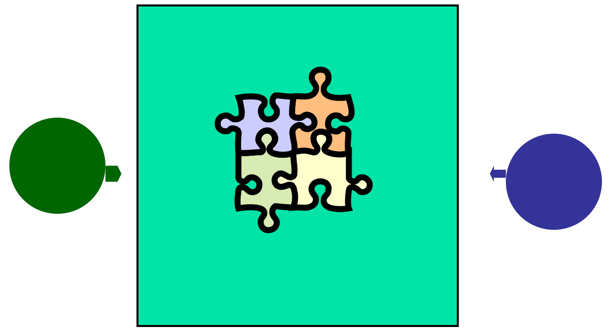
Serial Computing



Suppose you want to do a jigsaw puzzle that has, say, a thousand pieces.

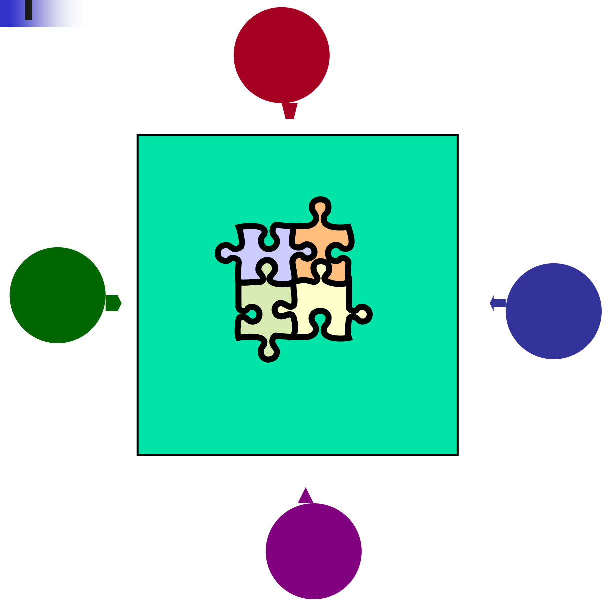
We can imagine that it'll take you a certain amount of time. Let's say that you can put the puzzle together in an hour.

Shared Memory Parallelism



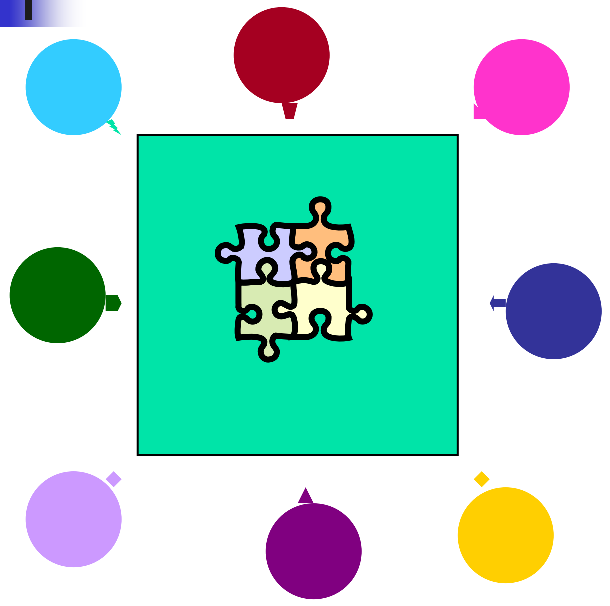
If Paul sits across the table from you, then he can work on his half of the puzzle and you can work on yours. Once in a while, you'll both reach into the pile of pieces at the same time (you'll ***contend*** for the same resource), which will cause a little bit of slowdown. And from time to time you'll have to work together (***communicate***) at the interface between his half and yours. The speedup will be nearly 2-to-1: y'all might take 35 minutes instead of 30.

The More the Merrier?



Now let’s put Charlie and Scott on the other two sides of the table. Each of you can work on a part of the puzzle, but there’ll be a lot more contention for the shared resource (the pile of puzzle pieces) and a lot more communication at the interfaces. So y’all will get noticeably less than a 4-to-1 speedup, but you’ll still have an improvement, maybe something like 3-to-1: the four of you can get it done in 20 minutes instead of an hour.

Diminishing Returns



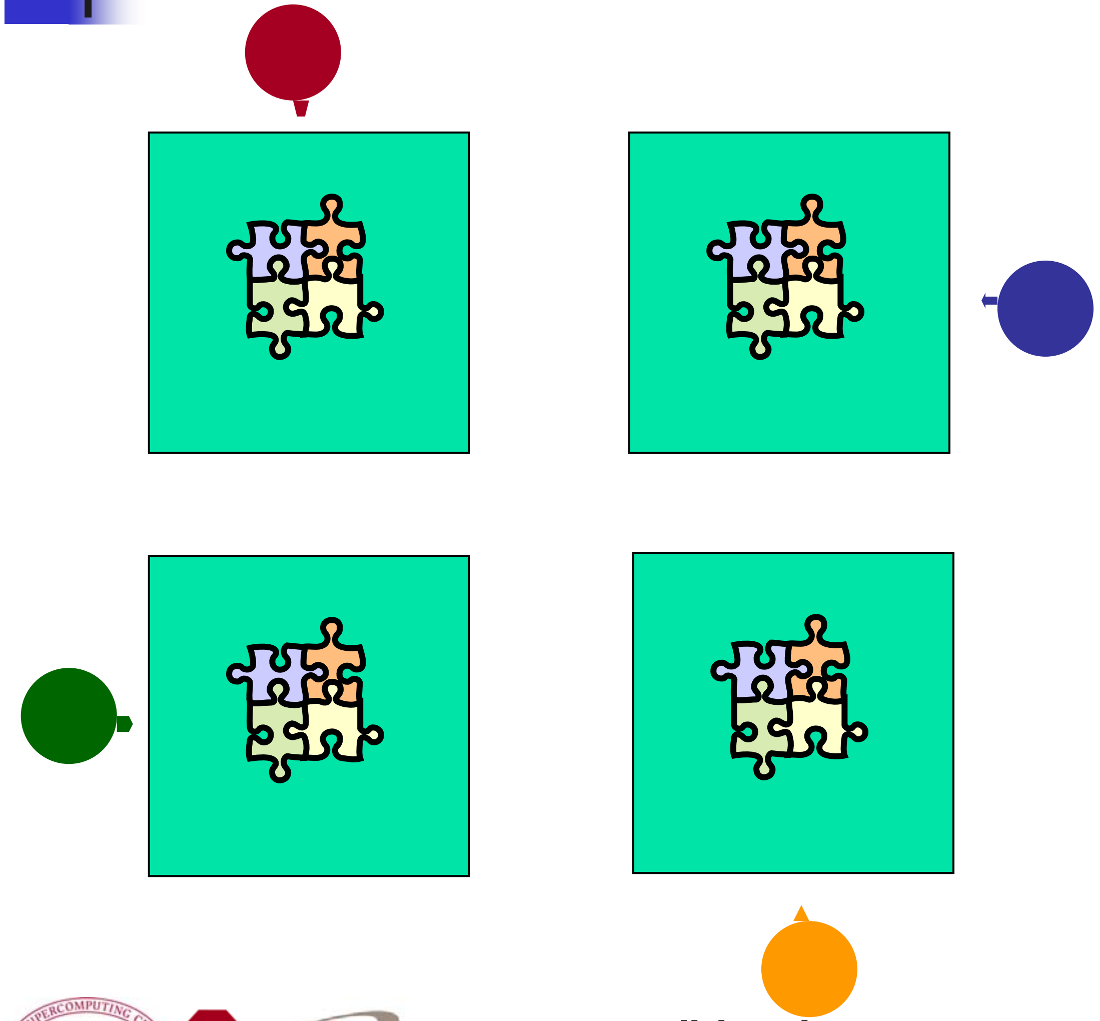
If we now put Rebecca and Jen and Alisa and Darlene on the corners of the table, there’s going to be a whole lot of contention for the shared resource, and a lot of communication at the many interfaces. So the speedup y’all get will be much less than we’d like; you’ll be lucky to get 5-to-1.

So we can see that adding more and more workers onto a shared resource is eventually going to have a diminishing return.

Distributed Parallelism

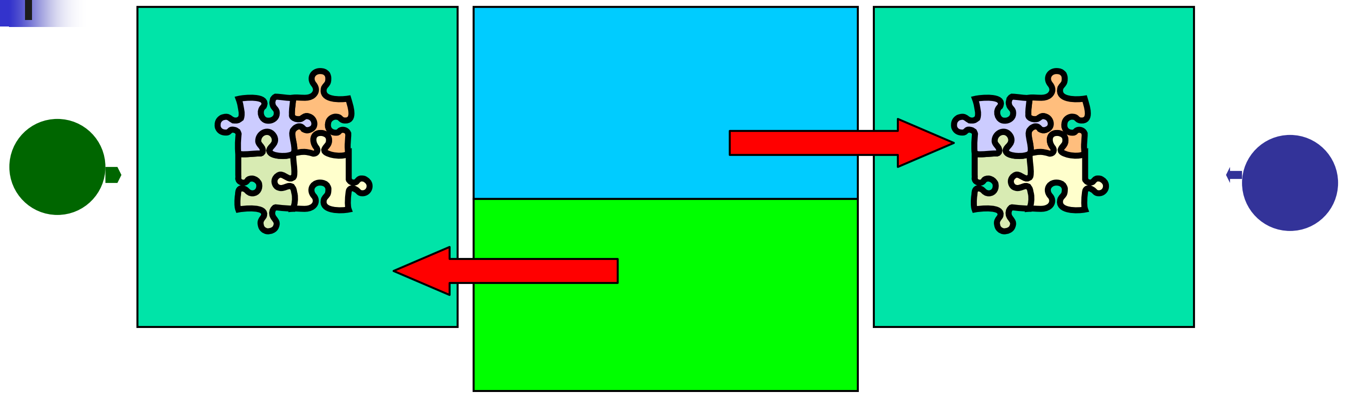
Now let’s try something a little different. Let’s set up two tables, and let’s put you at one of them and Paul at the other. Let’s put half of the puzzle pieces on your table and the other half of the pieces on Paul’s. Now y’all can work completely independently, without any contention for a shared resource. **BUT**, the cost of communicating is **MUCH** higher (you have to scootch your tables together), and you need the ability to split up (***decompose***) the puzzle pieces reasonably evenly, which may be tricky to do for some puzzles.

More Distributed Processors



It's a lot easier to add more processors in distributed parallelism. But, you always have to be aware of the need to decompose the problem and to communicate between the processors. Also, as you add more processors, it may be harder to ***load balance*** the amount of work that each processor gets.

Load Balancing



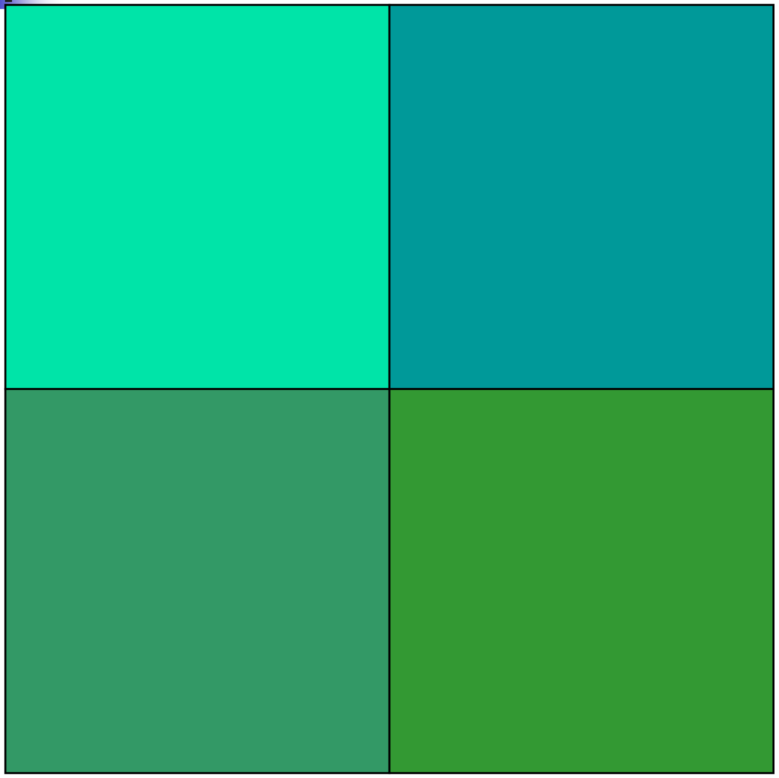
Load balancing means giving everyone roughly the same amount of work to do.

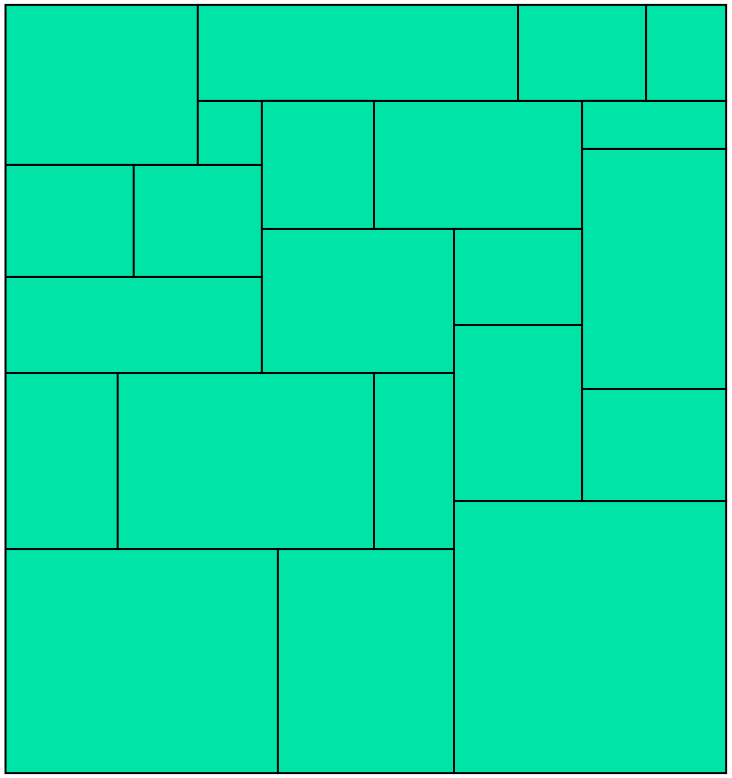
For example, if the jigsaw puzzle is half grass and half sky, then you can do the grass and Julie can do the sky, and then y'all only have to communicate at the horizon – and the amount of work that each of you does on your own is roughly equal. So you'll get pretty good speedup.

Load Balancing Is Good

- When every processor gets the same amount of work, the job is ***load balanced***.
- We like load balancing, because it means that our speedup can potentially be linear: if we run on N_p processors, it takes $1/N_p$ as much time as on one processor.
- For some codes, figuring out how to balance the load is trivial (e.g., breaking a big unchanging array into sub-arrays).
- For others, load balancing is very tricky (e.g., a dynamically evolving collection of arbitrarily many blocks of arbitrary size).

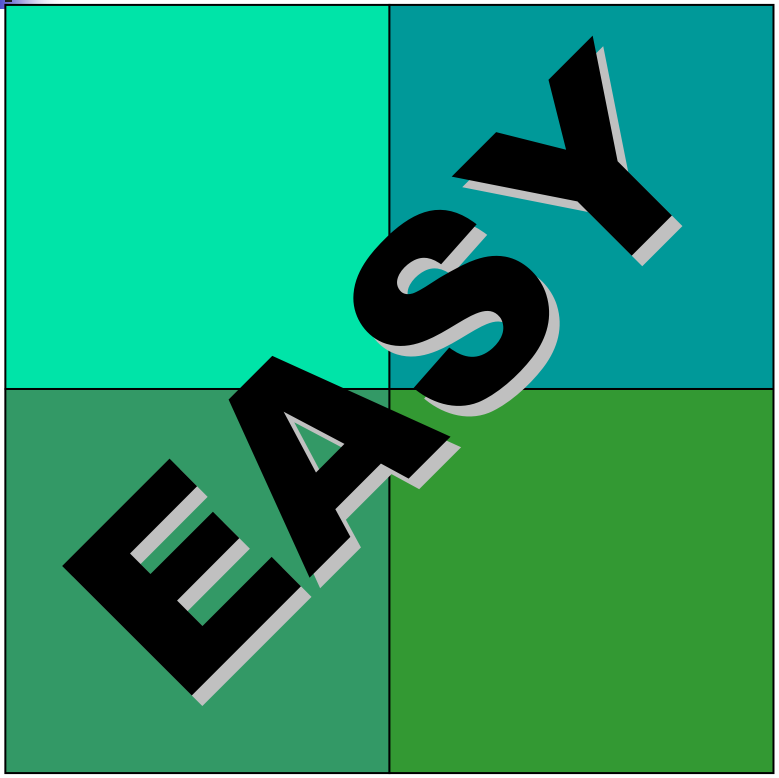
Load Balancing

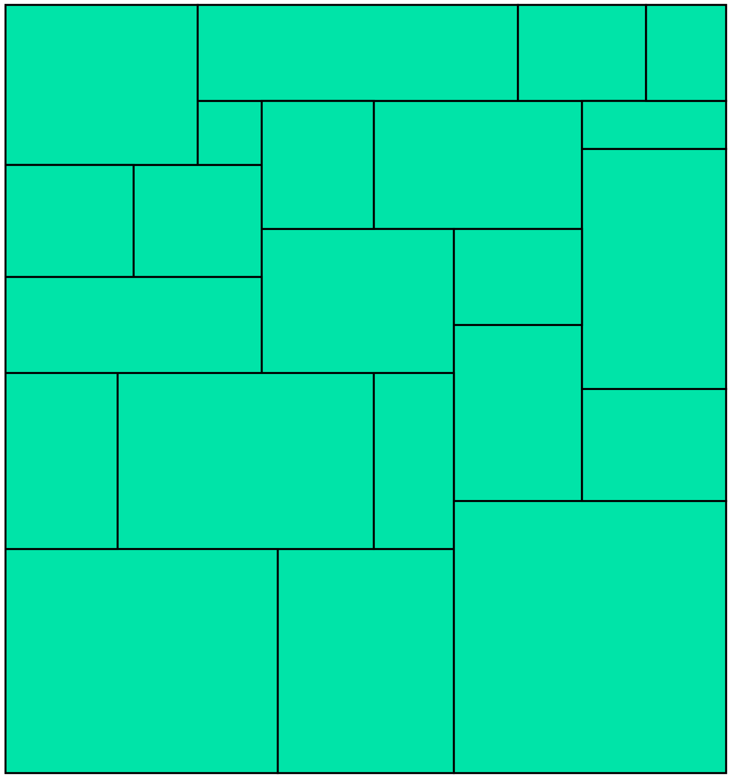




Load balancing can be easy, if the problem splits up into chunks of roughly equal size, with one chunk per processor. Or load balancing can be very hard.

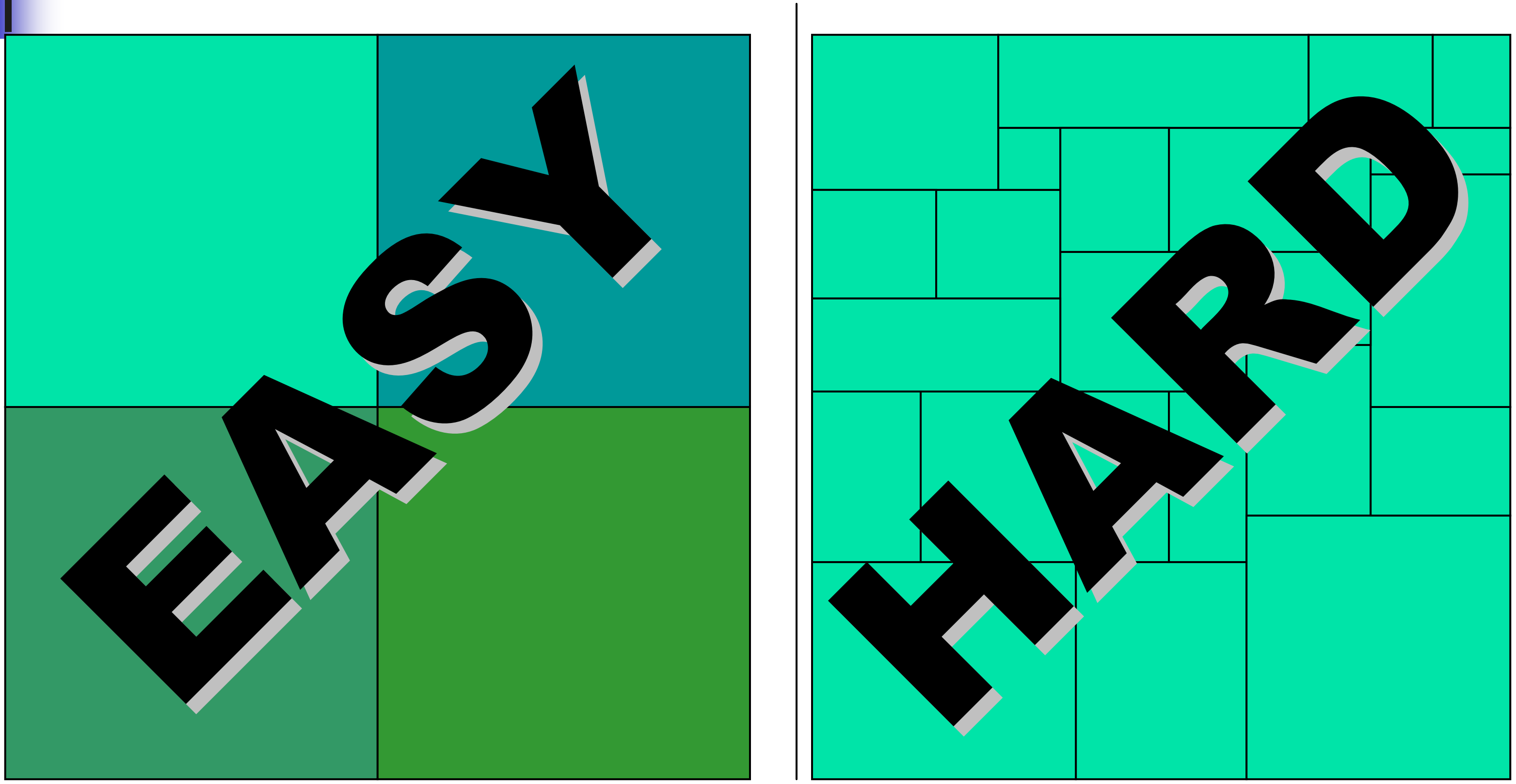
Load Balancing





Load balancing can be easy, if the problem splits up into chunks of roughly equal size, with one chunk per processor. Or load balancing can be very hard.

Load Balancing



Load balancing can be easy, if the problem splits up into chunks of roughly equal size, with one chunk per processor. Or load balancing can be very hard.

Distributed Multiprocessing: The Desert Islands Analogy

An Island Hut

- Imagine you’re on an island in a little hut.
- Inside the hut is a desk.
- On the desk is:
- a **phone**;
- a **pencil**;
- a **calculator**;
- a piece of paper with **numbers**;
- a piece of paper with **instructions**.

Instructions

- The **instructions** are split into two kinds:
- **Arithmetic/Logical**: e.g.,
  - Add the 27th number to the 239th number
  - Compare the 96th number to the 118th number to see whether they are equal
- **Communication**: e.g.,
  - dial 555-0127 and leave a voicemail containing the 962nd number
  - call your voicemail box and collect a voicemail from 555-0063 and put that number in the 715th slot

Is There Anybody Out There?

- If you're in a hut on an island, you **aren't specifically aware** of anyone else.
- Especially, you don't know whether anyone else is working on the same problem as you are, and you don't know who's at the other end of the phone line.
- All you know is what to do with the voicemails you get, and what phone numbers to send voicemails to.

Someone Might Be Out There

- Now suppose that Paul is on another island somewhere, in the same kind of hut, with the same kind of equipment.
- Suppose that he has the same list of instructions as you, but a different set of numbers (both data and phone numbers).
- Like you, he doesn’t know whether there’s anyone else working on his problem.

Even More People Out There

- Now suppose that Charlie and Scott are also in huts on islands.
- Suppose that each of the four has the exact same list of instructions, but different lists of numbers.
- And suppose that the phone numbers that people call are each others'. That is, your instructions have you call Paul, Charlie and Scott, Paul's has him call Charlie, Scott and you, and so on.
- Then you might all be working together on the same problem.

All Data Are Private

- Notice that you can't see Paul's or Charlie's or Scott's numbers, nor can they see yours or each other's.
- Thus, everyone's numbers are **private**: there's no way for anyone to share numbers, **except by leaving them in voicemails**.

Long Distance Calls: 2 Costs

- When you make a long distance phone call, you typically have to pay two costs:
- **Connection charge**: the **fixed** cost of connecting your phone to someone else's, even if you're only connected for a second
- **Per-minute charge**: the cost per minute of talking, once you're connected
- If the connection charge is large, then you want to make as few calls as possible.

Like Desert Islands

- Distributed parallelism is very much like the Desert Islands analogy:
- Processors are **independent** of each other.
- All data are **private**.
- Processes communicate by **passing messages** (like voicemails).
- The cost of passing a message is split into the ***latency*** (connection time) and the ***bandwidth*** (time per byte).

Latency vs Bandwidth on `topdawg`

- We recently tested the Infiniband interconnect on OU's large Linux cluster.
- **Latency** – the time for the first bit to show up at the destination – is about 3 microseconds;
- **Bandwidth** – the speed of the subsequent bits – is about 5 Gigabits per second.
- Thus, on topdawg's Infiniband:
- the 1st bit of a message shows up in 3 microsec;
- the 2nd bit shows up in 0.2 nanosec.
- So latency is **15,000 times worse** than bandwidth!

Latency vs Bandwidth on `topdawg`

We recently tested the Infiniband interconnect on OU’s large Linux cluster.

Latency – the time for the first bit to show up at the destination – is about 3 microseconds;

Bandwidth – the speed of the subsequent bits – is about 5 Gigabits per second.

Latency is **15,000 times worse** than bandwidth!

That’s like having a long distance service that charges

- $150 to make a call;
- 1¢ per minute – after the **first 10 days** of the call.

Parallelism Issues

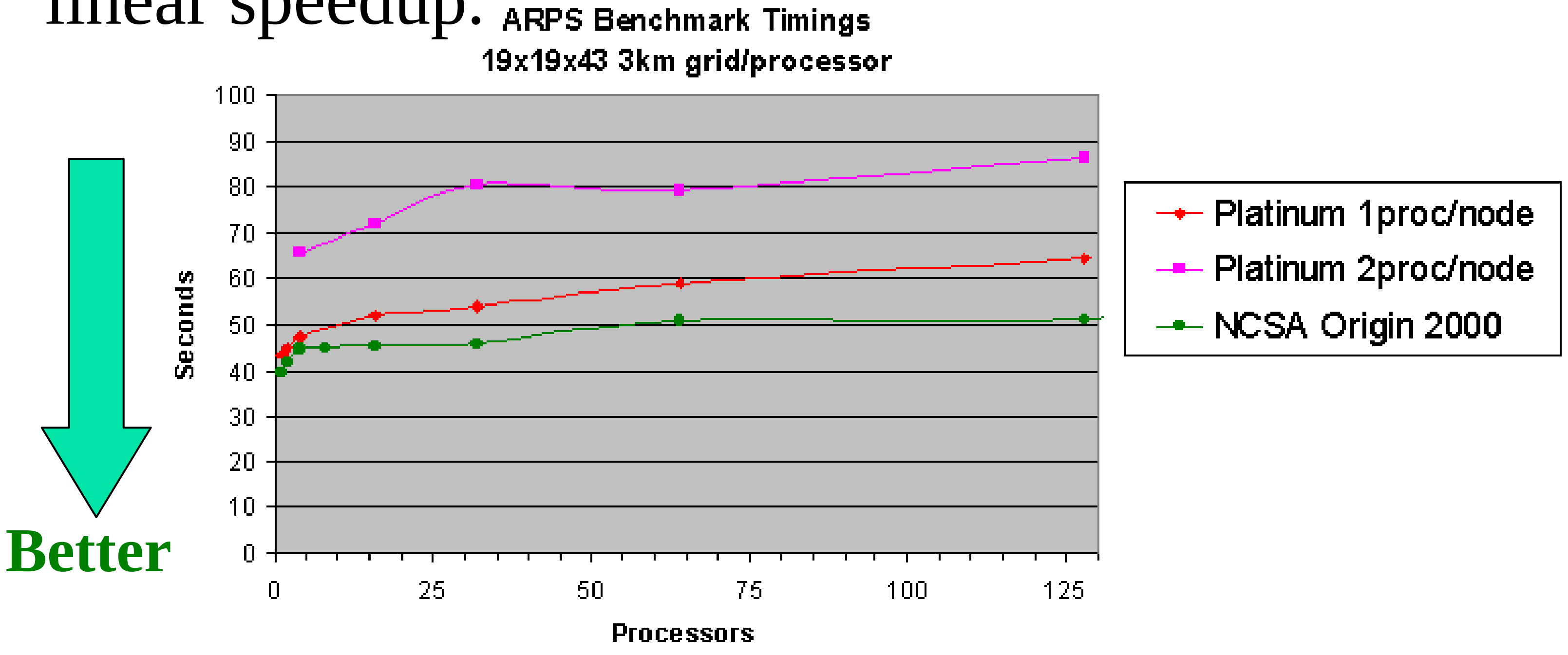
Speedup

The goal in parallelism is ***linear speedup***: getting the speed of the job to increase by a factor equal to the number of processors.

Very few programs actually exhibit linear speedup, but some come close.

Scalability

Scalable means "performs just as well regardless of how big the problem is." A scalable code has near linear speedup.



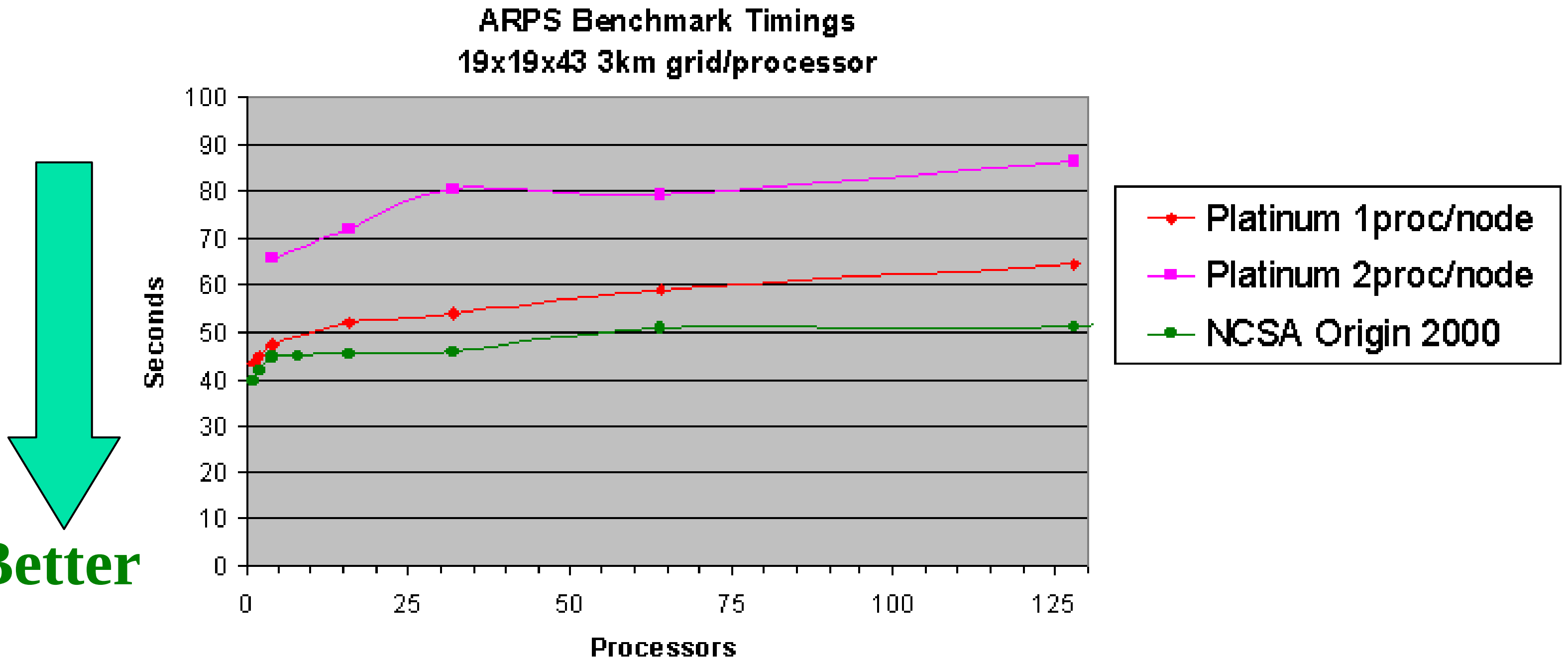
Platinum = NCSA 1024 processor PIII/1GHZ Linux Cluster
Note: NCSA Origin timings are scaled from 19x19x53 domains.

Strong vs Weak Scalability

- ***Strong Scalability***: If you double the number of processors, but you keep the problem size constant, then the problem takes half as long to complete (i.e., the speed doubles).
- ***Weak Scalability***: If you double the number of processors, and double the problem size, then the problem takes the same amount of time to complete (i.e., the speed doubles).

Scalability

This benchmark shows **weak** scalability.



Platinum = NCSA 1024 processor PIII/1GHZ Linux Cluster
Note: NCSA Origin timings are scaled from 19x19x53 domains.

Granularity

Granularity is the size of the subproblem that each process works on, and in particular the size that it works on between communicating or synchronizing with the others.

Some codes are ***coarse grain*** (a few very big parallel parts) and some are ***fine grain*** (many little parallel parts).

Usually, coarse grain codes are more scalable than fine grain codes, because less time is spent managing the parallelism, so more is spent getting the work done.

Parallel Overhead

Parallelism isn’t free. Behind the scenes, the compiler and the hardware have to do a lot of ***overhead*** work to make parallelism happen.

The overhead typically includes:

- **Managing** the multiple processes
- **Communication** among processes
- ***Synchronization*** (described later)

To Learn More

http://www.oscer.ou.edu/

Thanks for your attention!

Questions?