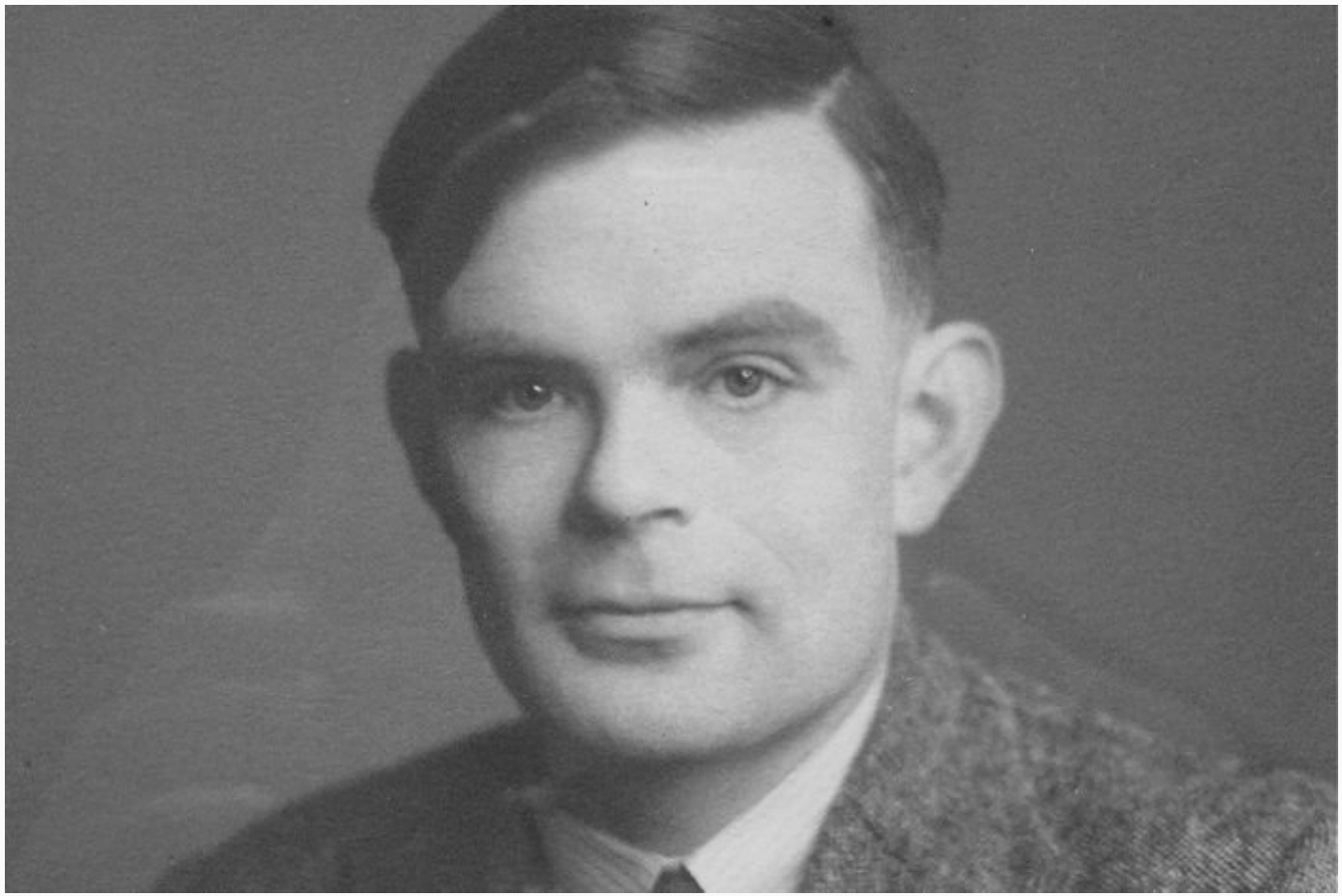
Introduction to Parallel Computing

NeSI Computational Science Team
support@nesi.org.nz

Outline

Story of computing
The Beginning

Alan Turing



Story of computing

Turing's Machine

- During World War II, Alan Turing - a British mathematician started to work with Britain's code-breaking centre and deciphered Germany's U-boat Enigma, saving battle of the Atlantic!
- He conceived the principles of modern computers and brought about his famous "Turing's Machine" in 1936.
- "Turing's Machine" is a hypothetical device that manipulates symbols on a strip of tape according to a table of rules which could simulate the logic of almost any computer algorithm.

Story of computing

Z1

- On the other side of the spectrum Konrad Zuse - a German civil engineer in Berlin - started dreaming of a machine that could do mechanical calculations.
- He started working in his parents' apartment and built his first electro-mechanical computer – Z1 in the same year 1936.
- It was a floating point binary mechanical calculator with limited programmability, reading instructions from a perforated 35 mm film.

Story of computing

Konrad Zuse

So we have a rare situation of two people coming from two different backgrounds and laying foundations for computer science.

Story of computing

First computers

- Invention of transistors in 1947 by John Bardeen, Walter Brattain & Wiliam Shockley replaced vacuum tubes used in early computers.
- Computers started to reduce in size and increase in speed.

Story of computing
Need for speed

Seymour Cray

- There was great appetite for speed, which was fuelled by aspirations of an American named Seymour Cray.
- He designed the first supercomputer - Cray-1 - in 1976. He is called the "father of supercomputing".

Story of computing

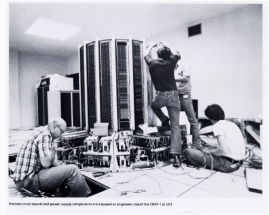
Cray-1

First supercomputers were monolithic in structure and architecture.

Printed circuit boards and power supply components are exposed as engineers install the CRAY-1 at UCS

Cray-1

First supercomputers were monolithic in structure and architecture.



This slide is intentionally left blank.

Hegelian dialectics

Thesis & anti-thesis

- History of anything can be explained using Hegelian dichotomy.
- German philosopher Georg Friedrich Hegel explained history as a dialectic between thesis, anti-thesis and resulting synthesis.
- This means that there could emerge a theory first and it could be confronted by an opposing theory. The dialectic between these opposing theories will find a compromise or consensus by assimilating the main aspects of both, in the due course of time.

Hegelian dialectics

Thesis: one

Two rules

- If we look carefully at the history of computing, we could see these Hegelian cycles of thesis and anti-thesis resulting in a synthesis.
- Computing, or even supercomputing started with one machine getting bigger and gaining speed. That was the thesis.
- Two rules emerged to support this model:
  - Moore's Law
  - Amdahl's Law

Hegelian dialectics

Moore's law

- Proposed by Gordon E. Moore (Co-founder of Intel) in 1965.
- In simple terms the law states that processor speeds, or overall processing power for computers will double every two years.
- More specifically it stated that the number of transistors on an affordable CPU would double every two years.
- Over roughly 50 years from 1961, the number of transistors doubled approximately every 18 months!

Hegelian dialectics

Moore's law in airline industry!

In 1978, a commercial flight between New York and Paris cost around $900 and took seven hours. If the principles of Moore's Law had been applied to the airline industry the way they have to the semiconductor industry since 1978, that flight would now cost about a penny and take less than one second.

Hegelian dialectics

Gordon E. Moore

Hegelian dialectics

Amdahl's law

- Proposed by Gene Amdahl in his 1967 technical paper titled "Validity of the single processor approach to achieving large scale computing capabilities".
- The famous Amdahl's formula below is not in this paper, but only a small literal description in the 4th paragraph of the 4 page paper!
- "The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program."

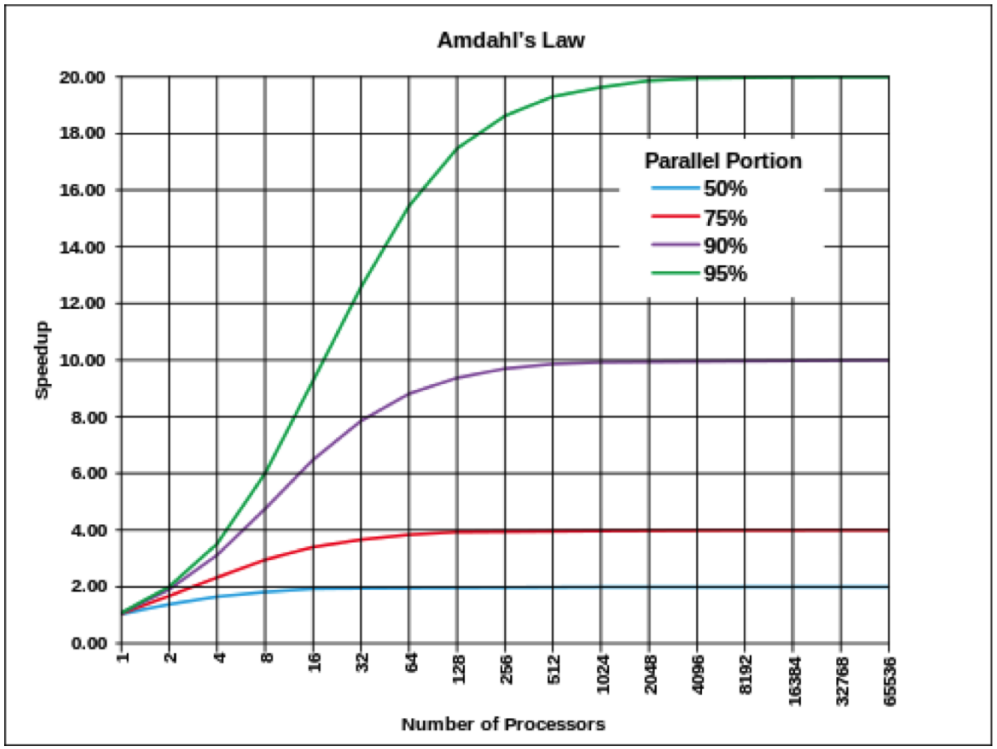
Hegelian dialectics

Amdahl's equation

$$S_{\mathrm{N}} = \frac{1}{(1-P) + \frac{P}{N}}$$

- Where:
  - P is the proportion of a program that can be made parallel
  - $(1-P)$ is the proportion that cannot be parallelized
  - N is the number of processors
- Original idea of Amdahl's law was to show the limitations of parallel computing!

Hegelian dialectics



- Not even considering the overheads of parallelization
- If parallization cannot be done evenly, results will be much worse!

Hegelian dialectics

Gene Amdahl

Hegelian dialectics

Anti-thesis: many

Rules that go wrong!

- Amdahl's law actually revealed the possibility of parallel computing:
  - There is actual increase in speed, if the algorithm is parallelizable.
  - There is practically no other way to increase speed in many cases, even if this is not the most efficient way to do it.
- Collapse of Moore's law is in the vicinity:
  - "In about 10 years or so, we will see the collapse of Moore's Law" says Physicist Michio Kaku, a professor of theoretical physics at City University of New York (2013).
  - Because of the heat and leakage associated with silicon based transistors.

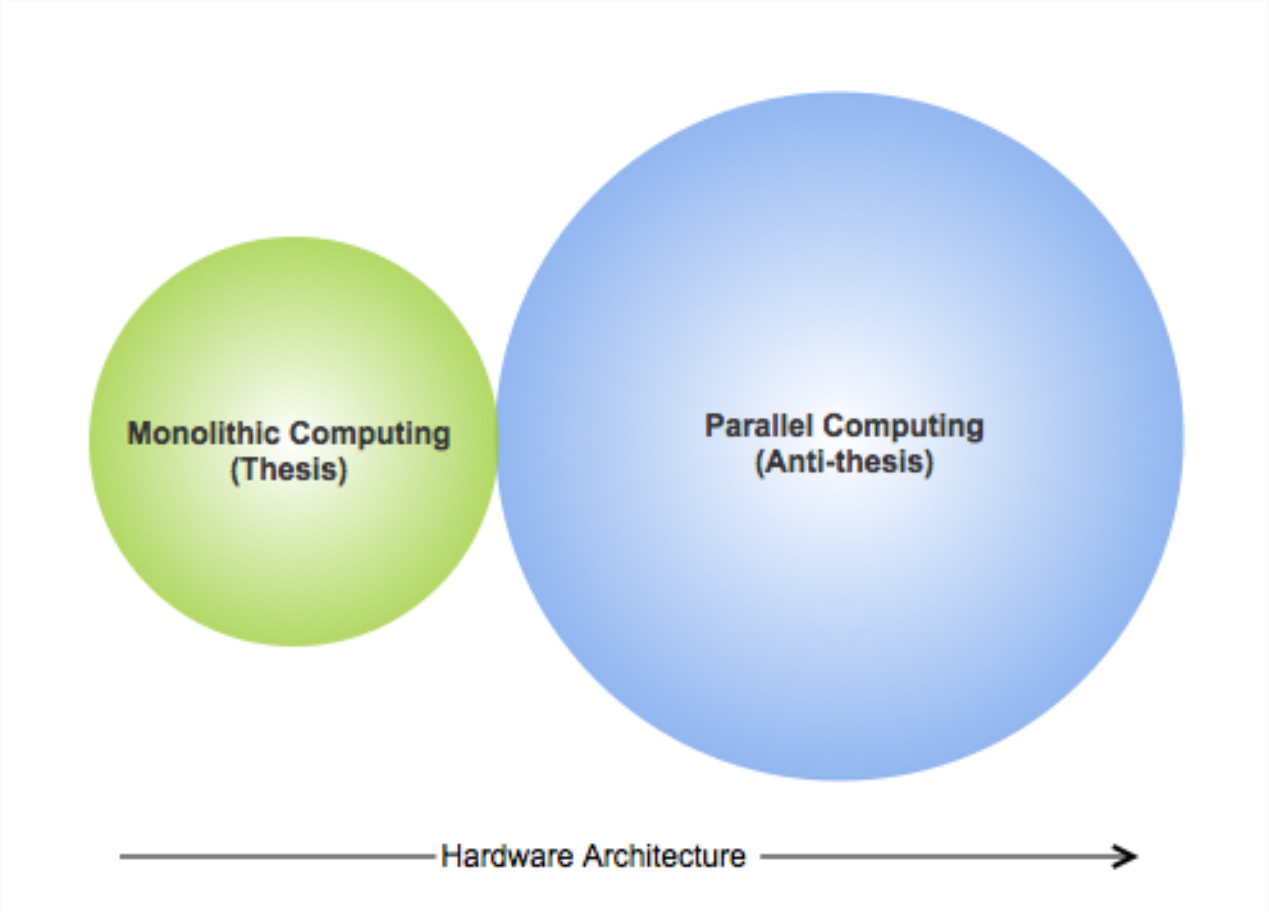
Rules that go wrong!

- Amdahl's law actually revealed the possibility of parallel computing:
  - There is actual increase in speed, if the algorithm is parallelizable.
  - There is practically no other way to increase speed in many cases, even if this is not the most efficient way to do it.
- Collapse of Moore's law is in the vicinity:
  - "In about 10 years or so, we will see the collapse of Moore's Law" says Physicist Michio Kaku, a professor of theoretical physics at City University of New York (2013).
  - Because of the heat and leakage associated with silicon based transistors.

This slide is intentionally left blank.

Parallel computing

The anti-thesis



Parallel computing

Background

Parallel computing emerged as an anti-thesis. Factors that have accelerated

- The arrival of cheap commodity computing hardware
- The theoretical possibility of achieving the same or even greater speeds of serial HPC computing by harnessing the distributed computing capabilities of commodity hardware
- Recognition that the nature itself is parallel, however complex it may appear. For example:
  - Parallel genome sequencing algorithms
  - Parallel Monte Carlo algorithms

Parallel computing

Definition

- Parallel computing is the art of breaking up a big chunk of serial computation into smaller atomic units which can be done in parallel.
- "It is the simultaneous use of multiple compute resources to solve a computational problem" [1].

[1] Ref: Lawrence Livermore National Laboratory, US

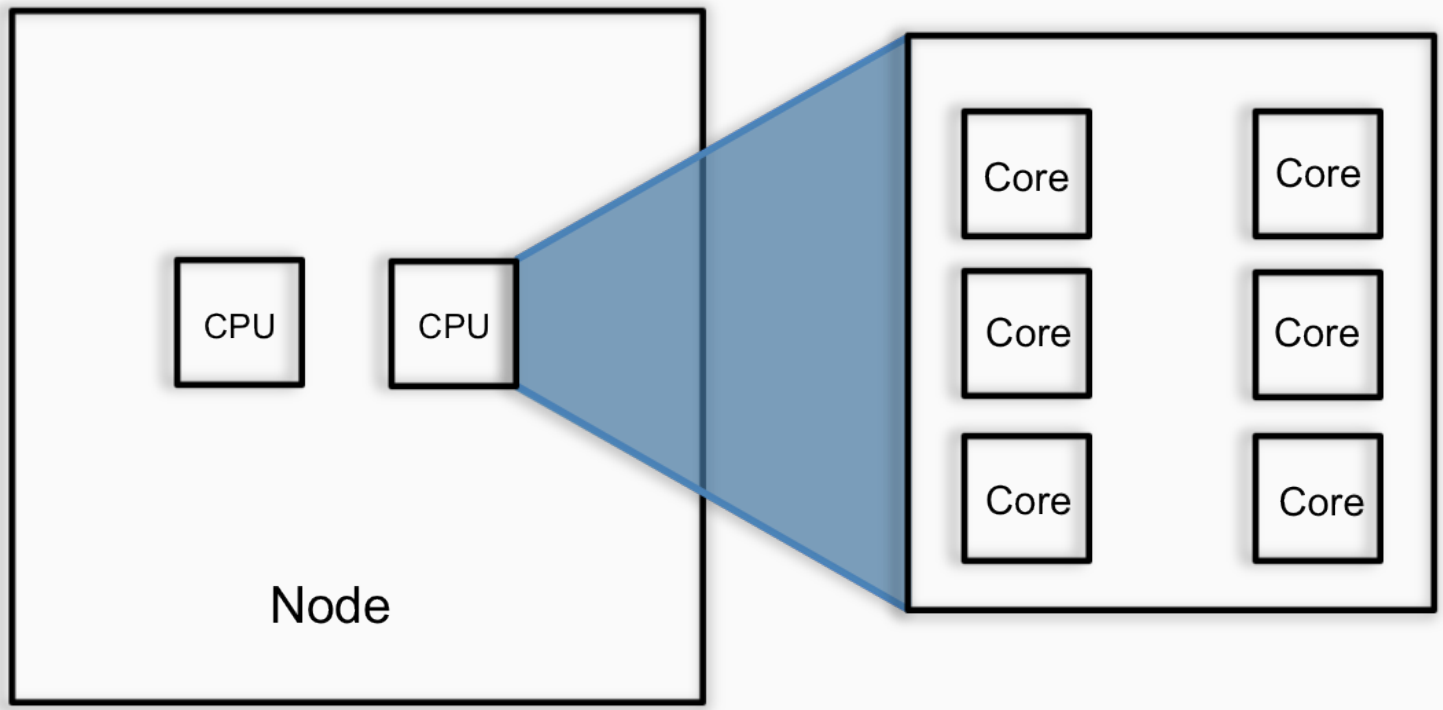
Parallel computing

Key concepts

- A cluster is a network of computers, sometimes called nodes or hosts.
- Each computer has several processors.
- Each processor has several cores.
- A core does the computing.
- If an application uses more than one core, it runs faster on a cluster.

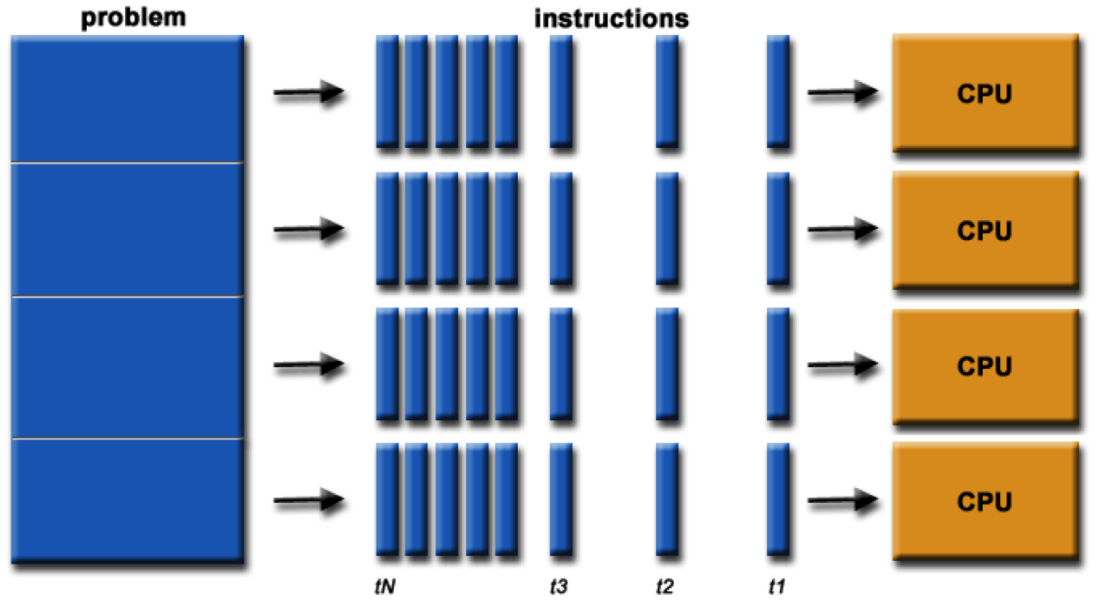
Parallel computing

A node overview



Parallel computing
Principle

- Breaking down of a problem into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors



Evolution of parallel computing

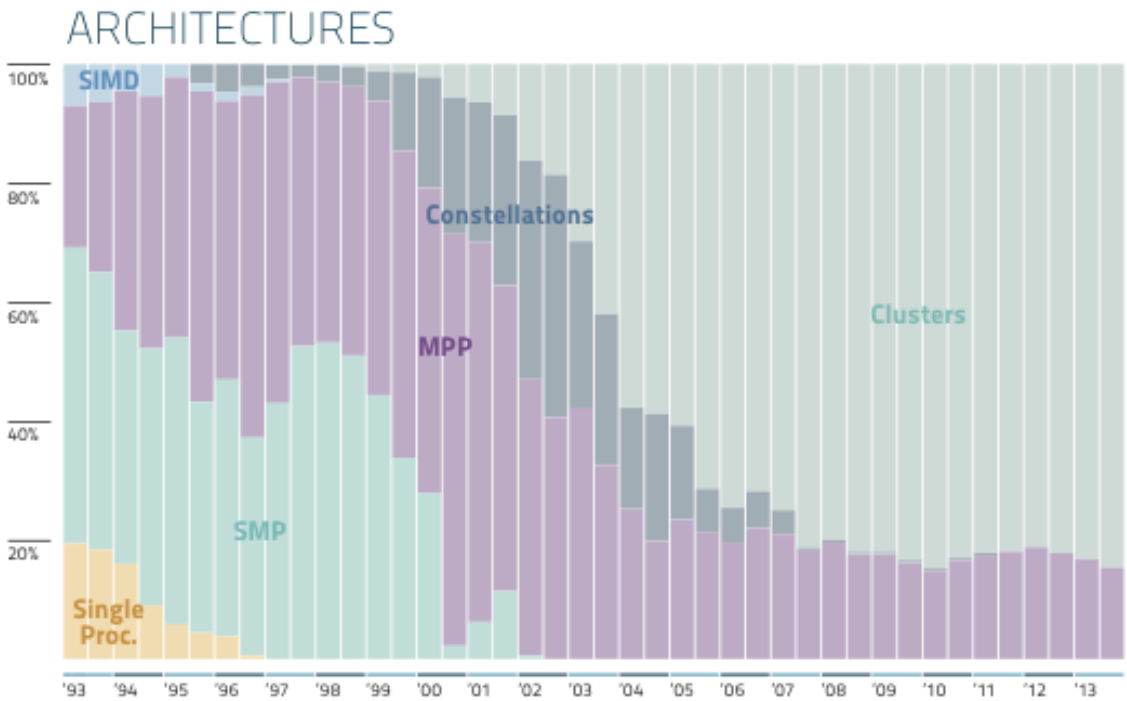
Grid & SOA

- Grid Computing
  - It is the distributed computing model which orchestrates distributed computing resources of different academic and research institutions to work like a single, unified computing system.
- Service Oriented Approach (SOA)
  - SOA is a programming paradigm based on the idea of composing applications by invoking network-available services to accomplish some tasks, mainly adopted by enterprise circles.
  - This was adopted mainly by business and enterprise community.

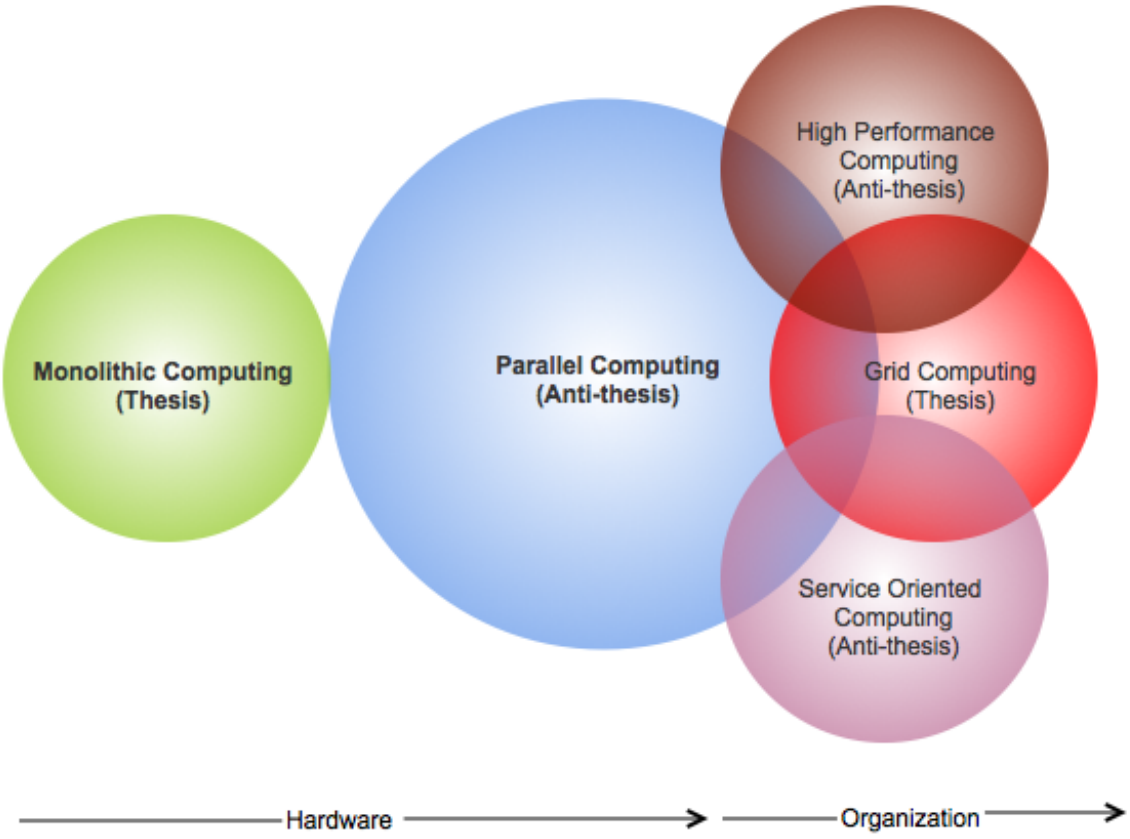
Evolution of parallel computing

High Performance Computing

- TOP500 list of 2002 reported that 20 percent of HPC installations as "clusters"- parallel computing emerged as a serious platform for HPC.
- TOP500 list of 2013 reports that "cluster" share is about 80 percent.



Hegelian cycles of parallel computing



Hegelian synthesis

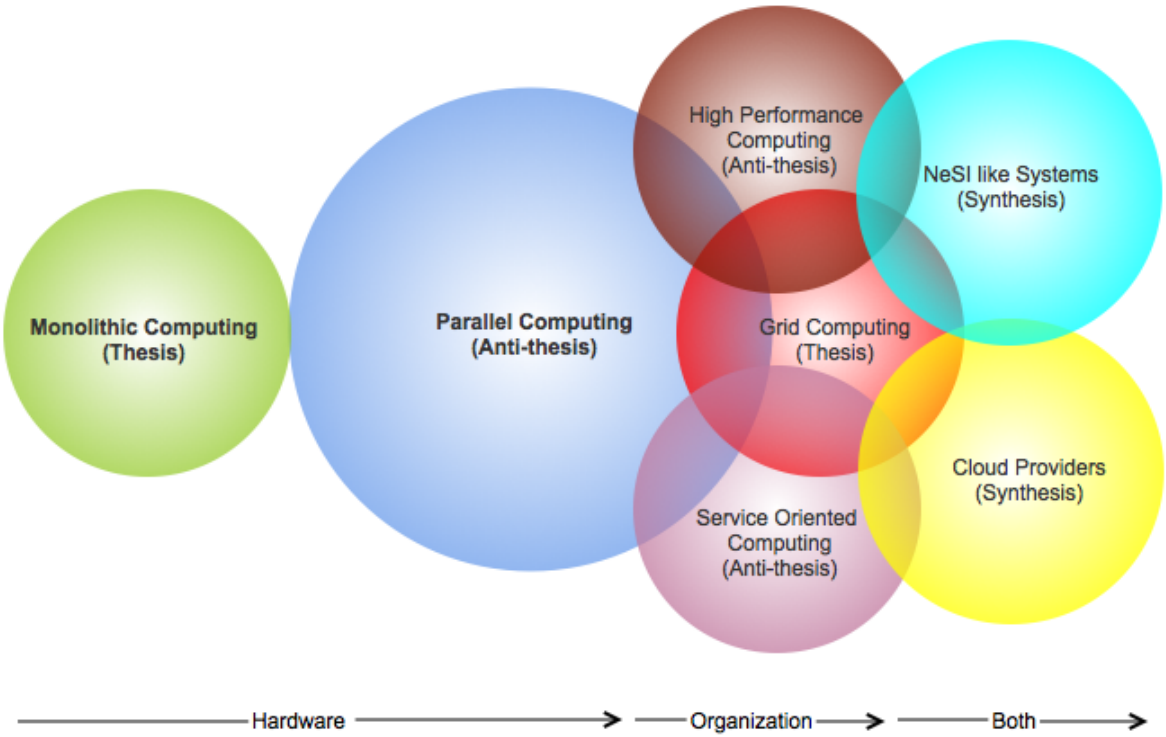
NeSI like systems

- Mixing grid computing with high performance computing
- Aiming to cater to research and academic community

Cloud providers

- Merging serviced oriented computing with grid concepts
- Providing services to enterprises and small businesses
  - It uses virtualization techniques to orchestrate storage, memory and network resources of geographically distributed data centers as a unified unit.
  - It is made available to customers on-demand with apparent elasticity.

Synthesis


High Performance Computing (Anti-thesis)
NeSI like Systems (Synthesis)
Monolithic Computing (Thesis)
Parallel Computing (Anti-thesis)
Grid Computing (Thesis)
Cloud Providers (Synthesis)
Service Oriented Computing (Anti-thesis)
Hardware
Organization
Both

NeSI Systems

Facilites

NeSI HPC resources cater to research and academic community of New Zealand and are spread around three major facilities:

- BlueFern Supercomputing Center at the University of Canterbury, Christchurch
- NIWA Supercomputing Center, Wellington
- CeR Supercomputing Center at the University of Auckland.

There is a single **grid** interface called **Grisu** to access both BlueFern and CeR (NeSI Pan) clusters.

NeSI Systems

Available HPC architectures

NeSI provides several kind of HPC architectures and solutions to cater for various needs.

- Bluegene P
- Power6 and Power7
- Intel Westmere & SandyBridge
- Kepler and Fermi GPU servers
- Intel Xeon Phi Co-Processor

NeSI Systems

Available bioinformatics applications

Many general purpose scientific applications are already available in NeSI systems.

- **Velvet:** Sequence assembler for very short reads
- **Bowtie:** Aligns short reads to a reference genome
- **BLAST:** Searches for regions of similarity between biological sequences
- **Hmmer:** Protein sequence search and alignment with Hidden Markov Models
- **Muscle:** Multiple sequence aligner
- **PhyML:** Maximum likelihood phylogenies
- **MrBayes:** Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models

NeSI Systems

Performance analysis

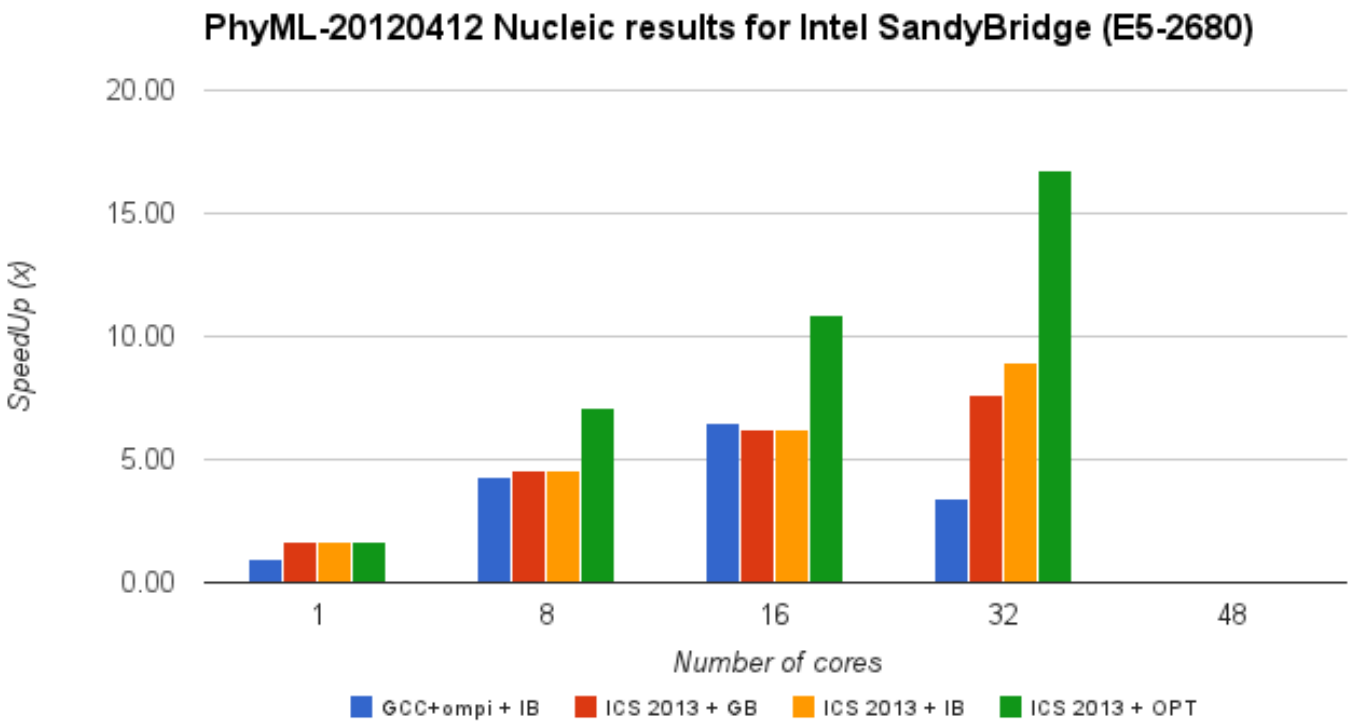
PhyML Case Study

PhyML is a software that estimates maximum likelihood phylogenies from alignments of nucleotide or amino acid sequences. The main strength of PhyML lies in the large number of substitution models coupled to various options to search the space of phylogenetic tree topologies, going from very fast and efficient methods to slower but generally more accurate approaches.
The right compilers and optimization options for an specific architecture can increase the performance quite a lot!

NeSI Systems

Performance analysis

PhyML Case Study : Speed Up

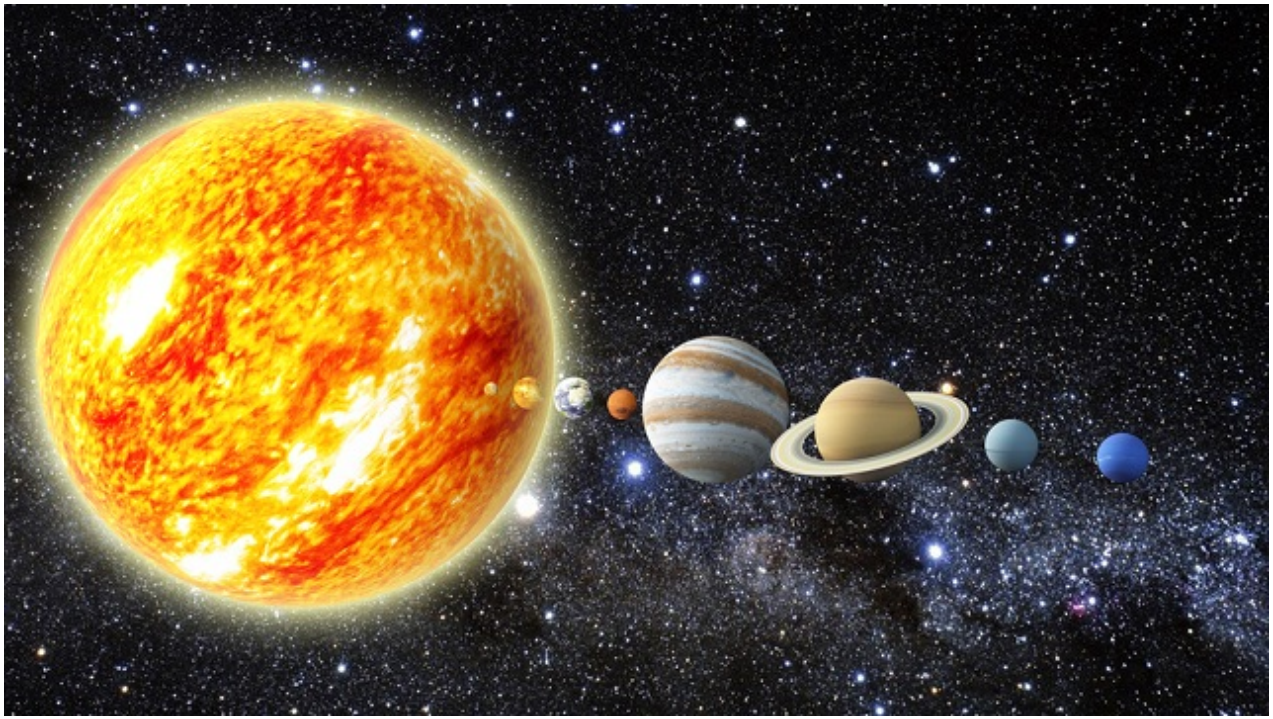


Parallel programming

Rationale

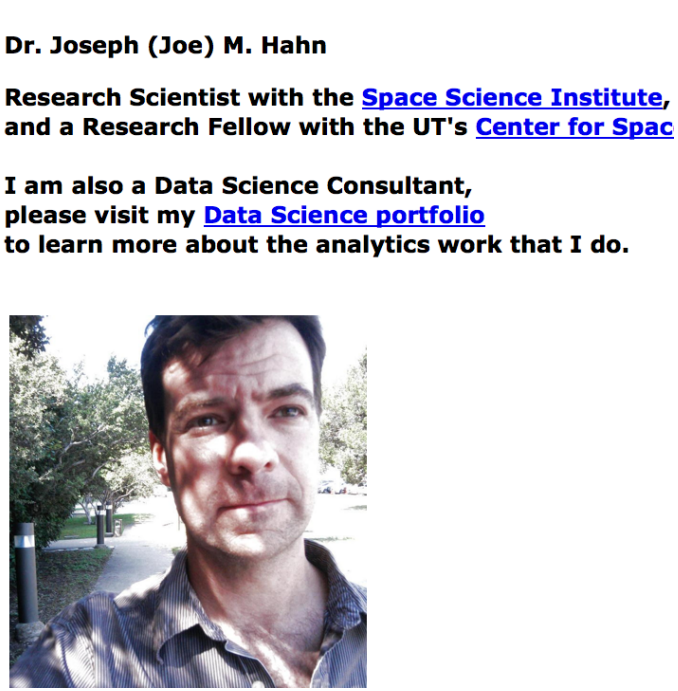
- Is there a rationale for writing your own parallel codes when we already got general purpose scientific applications?
- General purpose scientific applications are good for research.
  - 80 percent of NeSI cluster usage is by those applications at the moment.
  - They are easy to customise and use.
- However as the research complexity increases and its scope is refined general purpose scientific applications may not cater to all the needs of researchers.

Parallel problems
Case Study: N Body Problem



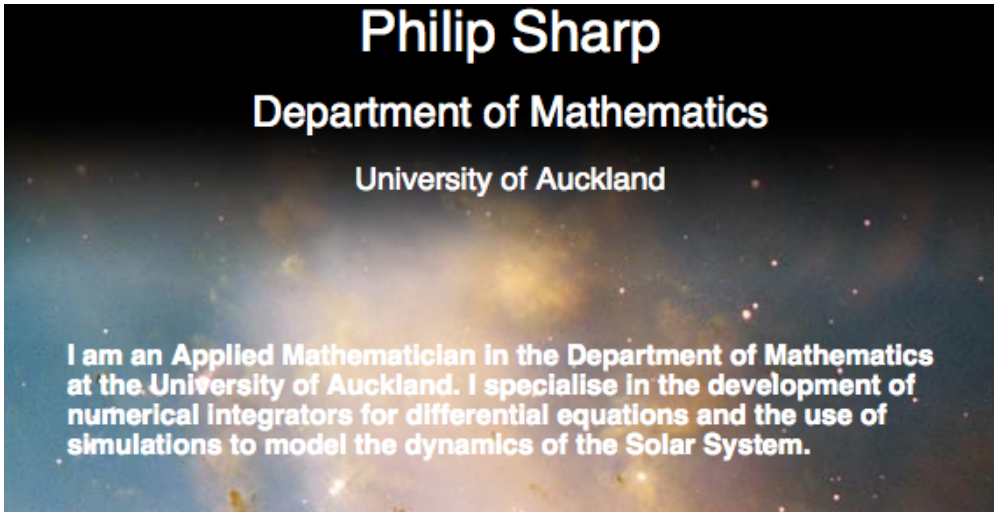
Gravitational n-body problem where:
Force is a function of mass and acceleration ($f = m * a$)

Parallel programming
Case Study: N Body Problem



- Mercury: N body simulator developed by Dr John Chamber at NASA Ames Research Center, California
- Dr Joseph M. Hahn of Space Science Institue, Colorado had to come up with a varient Mercury to take care of additional drag forces that drive planet migration!

Parallel programming
Case Study: N Body Problem



- Philip Sharp at University of Auckland Started implementing Mercury n-body integration on GPUs.
- Then he found out that he has been experimenting with more robust versions of those techniques.
- He is now rebuilding the n-body integrator from scratch.

Parallel programming
Case Study: N Body Problem

We need to look for hotspots and bottlenecks to parallelize an algorithm.

Hotspots

- Hotspots are the areas where we have massive iterative works, which can be parallelized into optimally smaller units that are independent of each other.
- Those units of work are called tasks and such algorithms can be called "embarrassingly parallel" in nature.

Parallel programming
Case Study: N Body Problem

Bottlenecks

- Bottlenecks are the places where computations becomes inter-dependent. Usually it is where there will be a need to synchronise the data before we proceed to do another set of parallel tasks.
- If an algorithm has got too many inter-dependent tasks, it will be called a fine-grained algorithm and may not able to parallelize efficiently.

Direct n-body problem

Governing equations

$$\mathbf{f}_{ij} = G\frac{m_i m_j}{||\mathbf{r}_{ij}||^2} \cdot \frac{\mathbf{r}_{ij}}{||\mathbf{r}_{ij}||}$$

$$\mathbf{F}_i = \sum_{1 \leq j \leq N} \mathbf{f}_{ij} \quad , \; j \neq i$$

$$\mathbf{F}_i = m_i \mathbf{a}_i$$

$$\mathbf{p}_i = \iint \mathbf{a}_i \, \mathrm{d}t$$

Where:

- Force (F) is function of mass and acceleration and
- Position (p) is a function of velocity/acceleration and time.

Direct n-body problem

Pseudocode

```
1: Input initial positons and velocities of particles
2: while each simulation step do
3:     for each particle do
4:         Compute total force on the particle
5:     end for
6:     for each particle do
7:         Compute velocity and position of the particle
8:     end for
9:     Output new velocity and position of particles
10: end while
11: Output final velocity and position of particles
```

Direct n-body problem

Parallel decomposition

There are two major decomposition techniques:

- Domain/data decomposition
  - It forms the foundation for most parallel algorithms
  - Here we assign different subset of data to different processors and do the same or similar computation over it.
- Functional/task decomposition
  - It is an alternate way where we assign different functions to different processors
  - When algorithms does not have any obvious data structure that can be decomposed.
  - It could the case of a single task at the root of a tree creates new tasks for each subtree, based on the mode of computation rather than the structure of the data.

Direct n-body problem

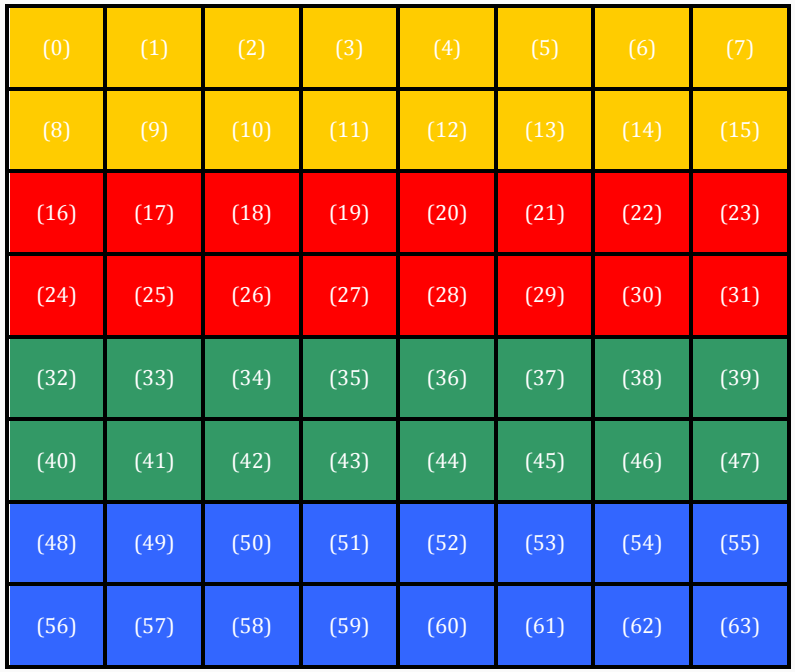
Parallel decomposition

- We can visualize bodies in direct n-body problem as members of an array or a matrix (2 dimensional array).
- This data can be divided into subsets: it is a case of domain/data parallelism.

Direct n-body problem

Data decomposition as an array

(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
(16)	(17)	(18)	(19)	(20)	(21)	(22)	(23)
(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)
(32)	(33)	(34)	(35)	(36)	(37)	(38)	(39)
(40)	(41)	(42)	(43)	(44)	(45)	(46)	(47)
(48)	(49)	(50)	(51)	(52)	(53)	(54)	(55)
(56)	(57)	(58)	(59)	(60)	(61)	(62)	(63)



Direct n-body problem

Data decomposition as a matrix

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(2, 7)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)	(3, 7)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	(4, 7)
(5, 0)	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)	(5, 7)
(6, 0)	(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)	(6, 7)
(7, 0)	(7, 1)	(7, 2)	(7, 3)	(7, 4)	(7, 5)	(7, 6)	(7, 7)

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(2, 7)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)	(3, 7)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	(4, 7)
(5, 0)	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)	(5, 7)
(6, 0)	(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)	(6, 7)
(7, 0)	(7, 1)	(7, 2)	(7, 3)	(7, 4)	(7, 5)	(7, 6)	(7, 7)

Direct n-body problem

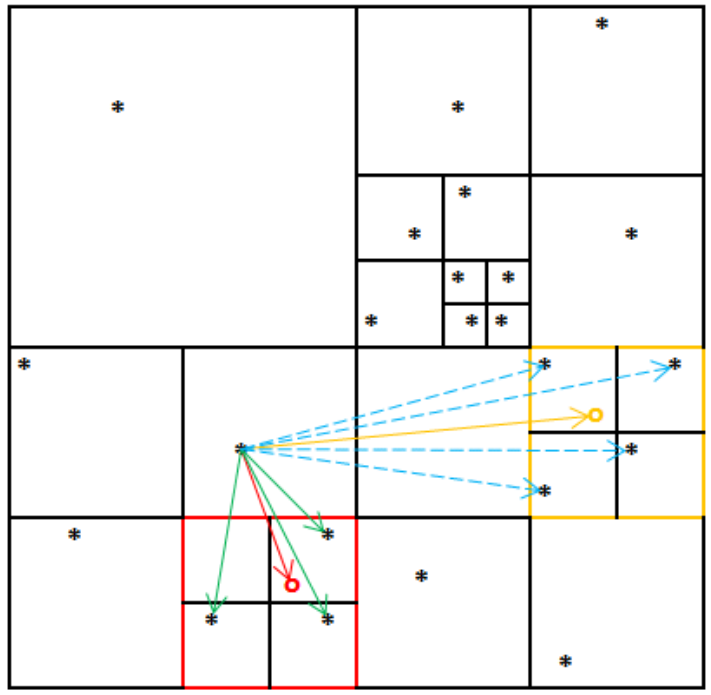
Parallel decomposition

- The algorithm is reduced to parallel matrix-vector operations (Note: Discussion on OpenMP will use an example of matrix multiplication)
- However all nodes need to have access to all the data in this model, which could put constraint on memory or latency.
- There are other n-body algorithms to mitigate this issue, where bodies are geographically grouped together to consider it as a large single body (Example: Barnes–Hut method).
- All data matrix problems cannot be divided as we did in direct n-body method.

N-body problem

Barnes–Hut method

Parallel decomposition



It reduces the complexity of problem from $O(n^2)$to $O(nlogn)$

Parallel decomposition

Heat diffusion problem

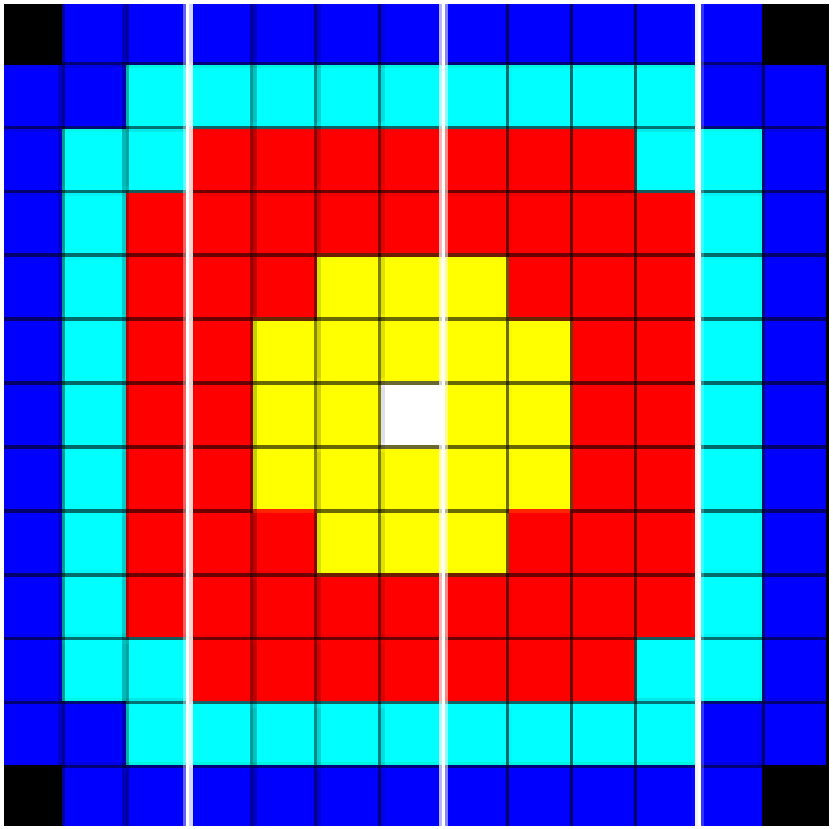


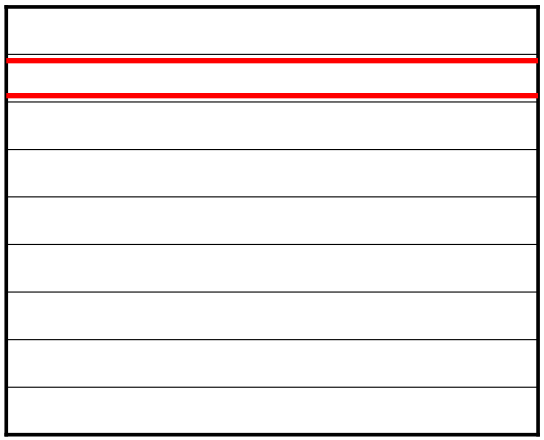
Figure : Heat diffusion on a 2D plate

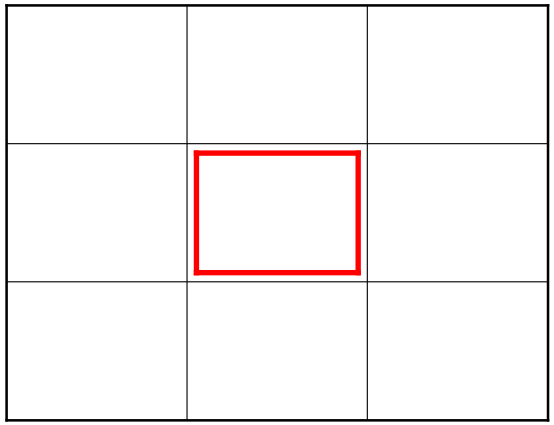
For updating the cells, we need all the neighbours of all the cells.

Parallel decomposition

Heat diffusion problem

The economy of decomposition



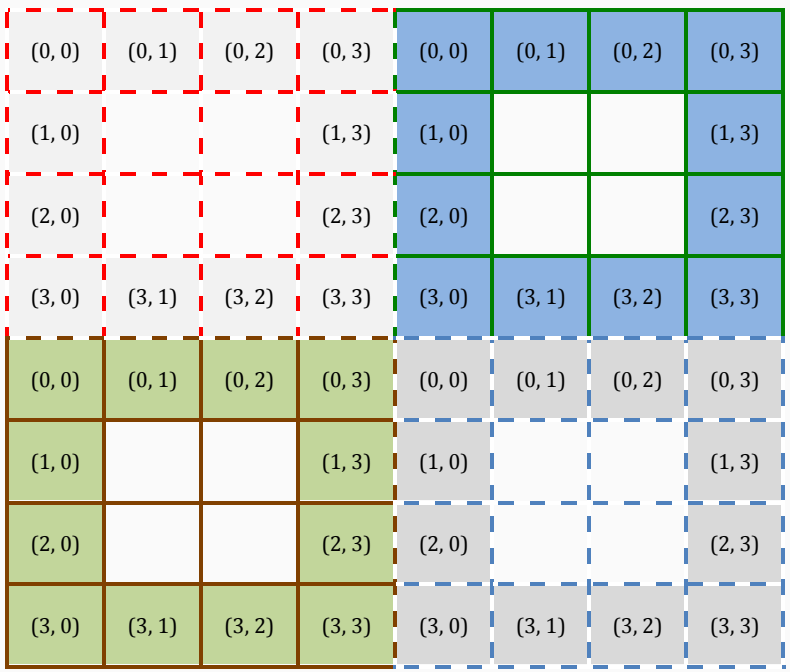


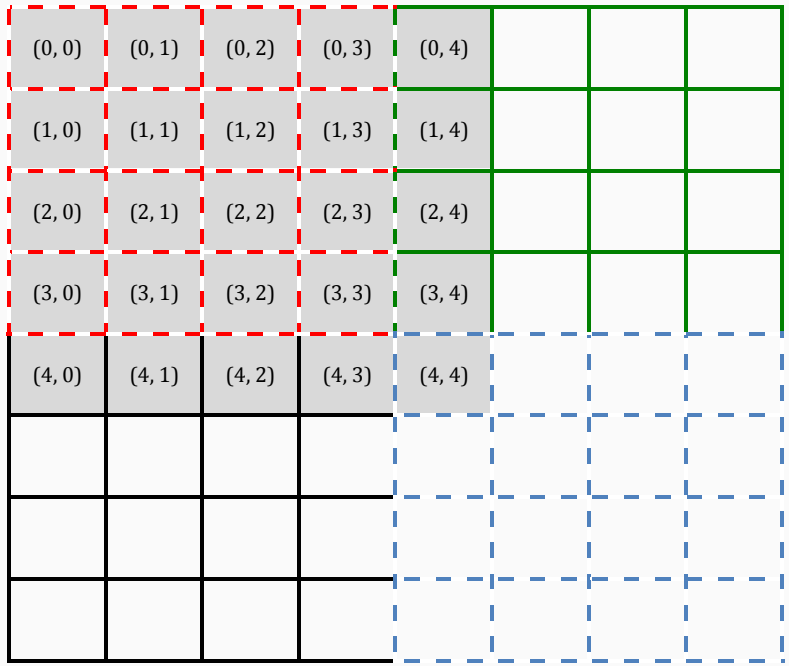
Partitioning as near-square rectangular blocks will give an advantage of $4 * (\sqrt{p}$ - $1)N/\sqrt{p}$ times over partitioning as rows (where p is perimeter and N is number of partitions) in terms of the amount of data to be synchronized.

Parallel decomposition

Heat diffusion problem

Ghost cells





Data visualization after sharing ghost cells among the processors.

Parallel decomposition

Bioinformatics problem

Case study: Bioinformatics

Parallel decomposition
Bioinformatics problem

Case study: Bioinformatics

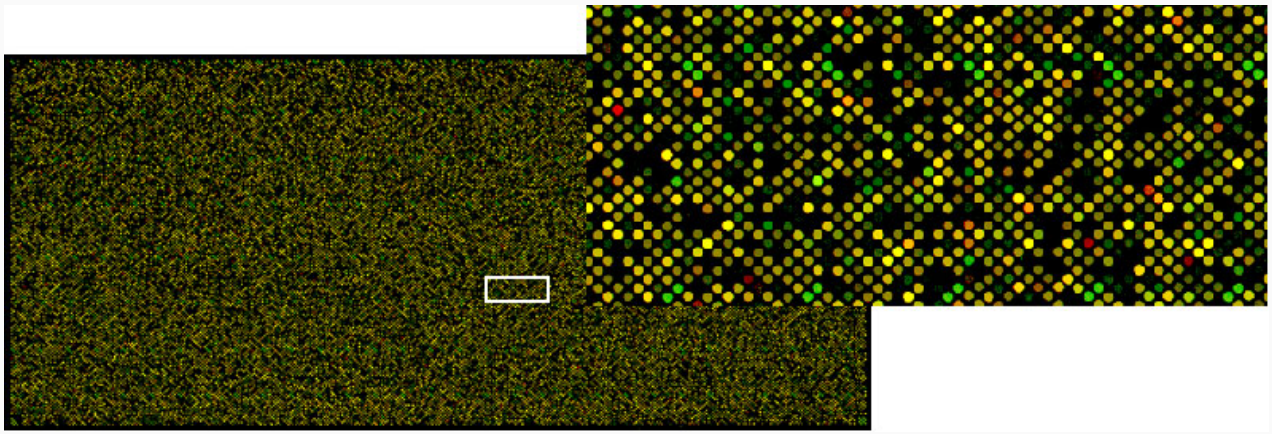
This slide is intentionally left blank.

Parallel decomposition

Bioinformatics problem

Biological data can be big, eg:

- Gene expression data: DNA microarrays arrays now provide tens of thousands of values per sample

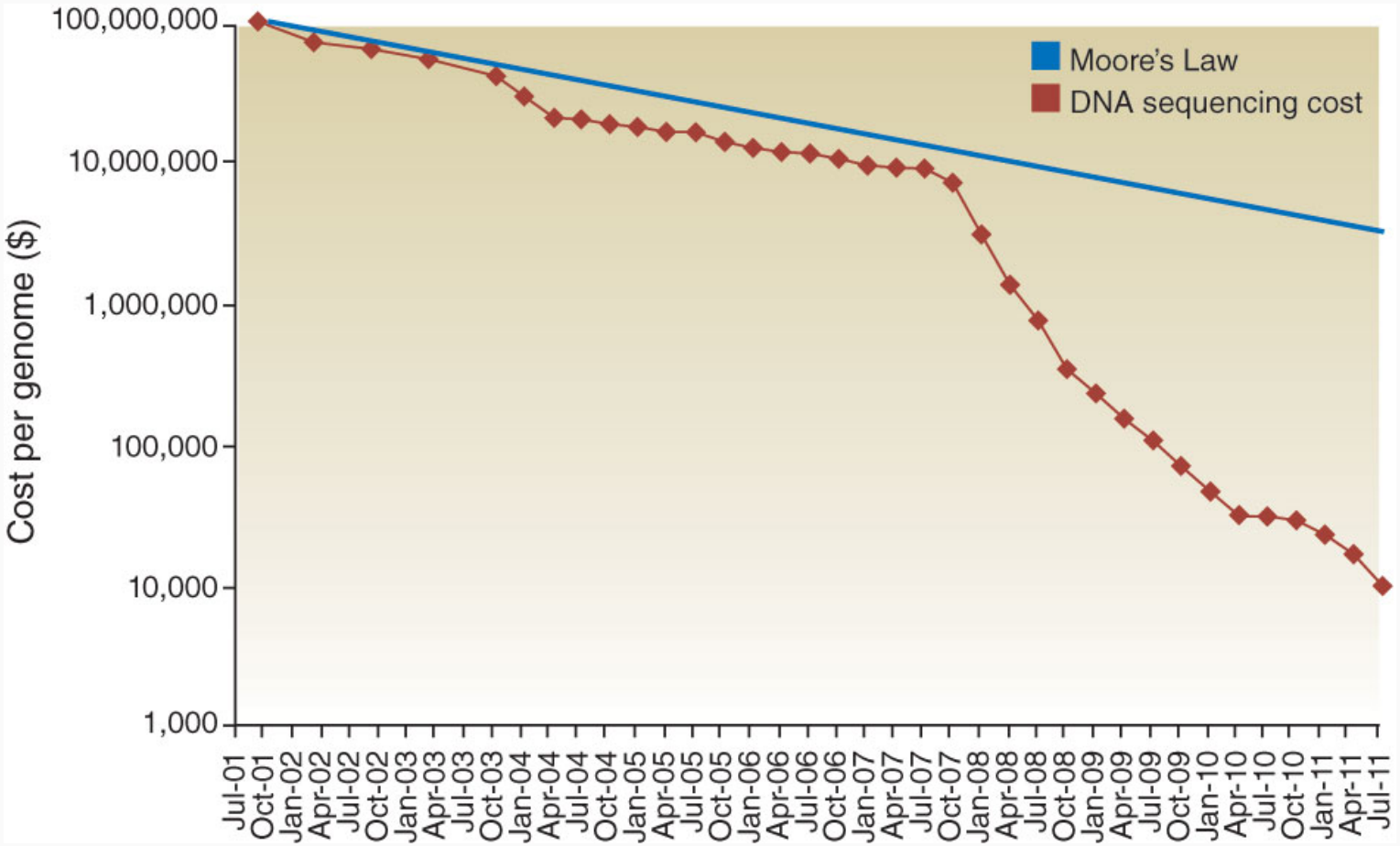


- Protein X-ray Crystalography
- DNA Sequence

Parallel decomposition

Bioinformatics problem

DNA Sequencing technology is outpacing Moore's Law



Parallel decomposition

Bioinformatics problem

Sequence Assembly

- Raw sequence data is assembled into fewer, longer sequences by aligning overlaps. eg:
  - Each sequence read: 10s to 1000s of bases.
  - Longest human chromosome: 249 million bases.
- It's like a million piece jigsaw puzzle with duplicates: memory hungry and not fully parallelisable.

Parallel decomposition

Bioinformatics problem

Homology Search / Alignment

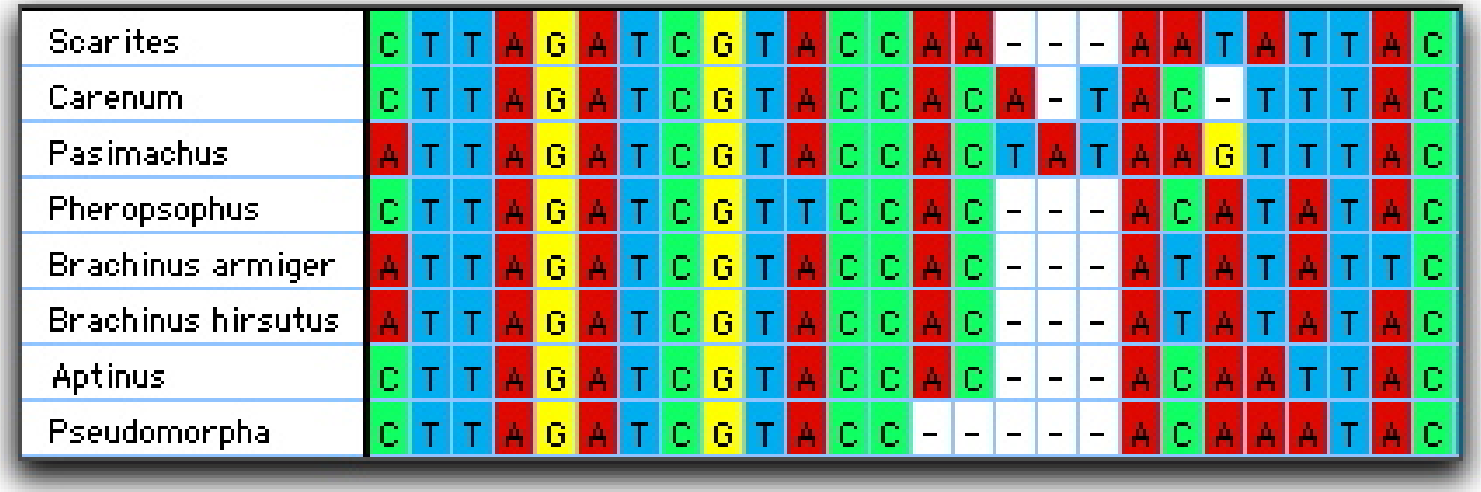
- Q: What (parts of) known sequences are (parts of) your new sequence similar (i.e.: related) to?
- Another overlap alignment problem, but with fuzzier matching.
- Your new sequences vs known sequences, eg: the 37 million proteins in the "refseq" database.
- Can be parallelised by dividing up the queries or the reference sequences between the CPUs.
- e.g: BLAST is multi-threaded, dividing up the database between the threads.

Parallel decomposition

Bioinformatics problem

Multiple Sequence Alignment

Aligning all the related sequences is also computationaly intensive even though the amount of data is generally not so large by this step.



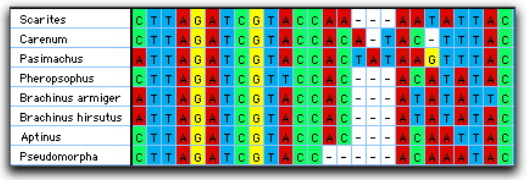
Parallel decomposition

Bioinformatics problem

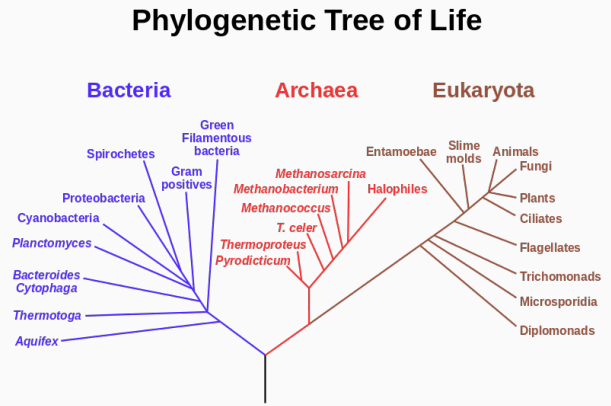
Pylogenetic Reconstruction

Derive a tree of descent from multiple aligned sequences

Input



Output



Parallel decomposition

Bioinformatics problem

Parallelise tree evaluation by:

- Tree Branch - to some degree.
- Sequence Position - more generally.

Parallelise tree optimisation by:

- Parameter Space - eg: the rate of mutation may take on different values.
- Tree Space - the number of possible tree topologies increases quadratically.

Parallel programming

Memory Architecture

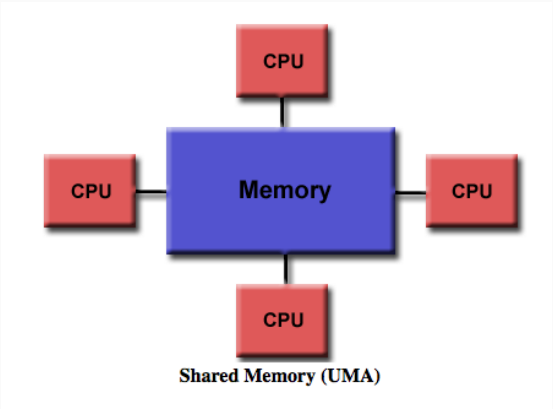
Architectural differences between memory architectures have implications on how we program.

- Shared-memory systems uses a single address space allowing processors to communicate through variables stored in a shared address space.
- In distributed-memory system each processor has its own memory module and connected over a high speed network for communication.

Parallel programming

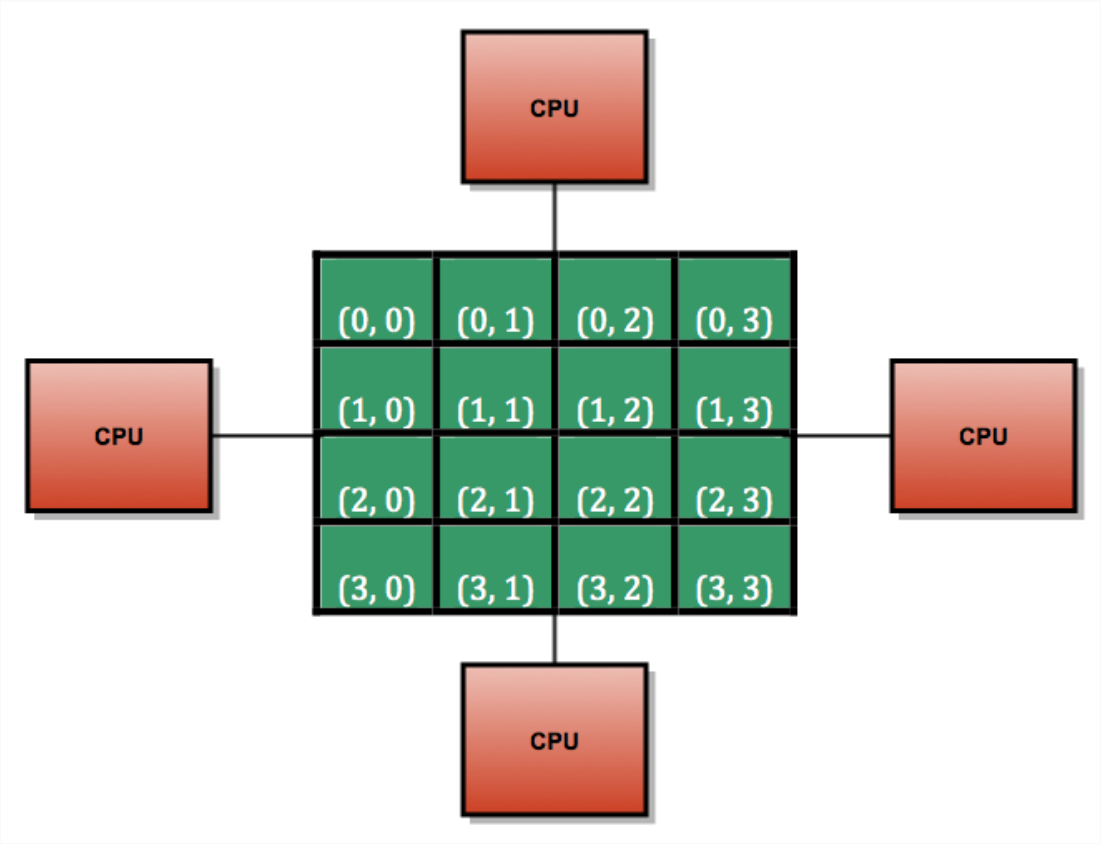
Shared memory

- Symmetric multiprocessing (SMP) : two or more identical processors are connected to a single shared main memory.
- This shared memory may be simultaneously accessed by single program using multiple threads.
- The most popular parallel programming paradigm is OpenMP.



Parallel programming
Shared memory

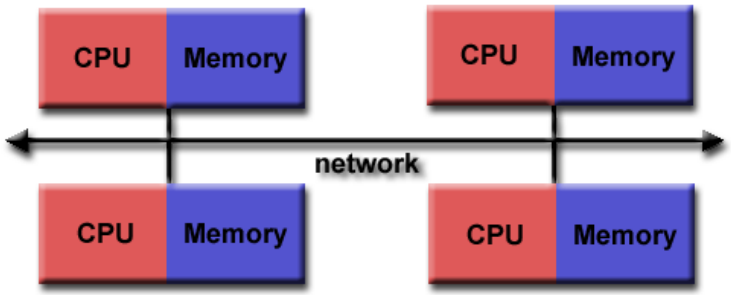
Shared memory: address system



Parallel programming

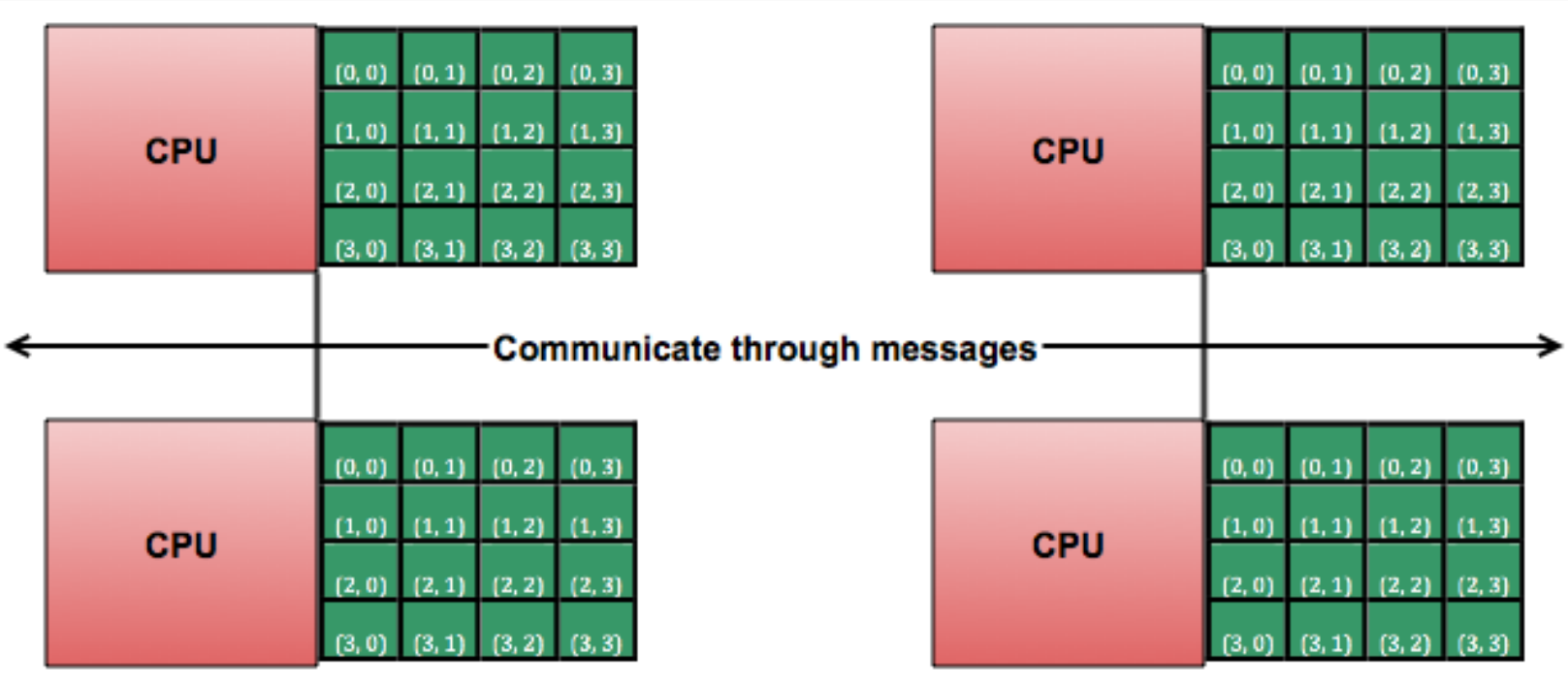
Distributed memory

- Multiple-processor computer system in which each process has its own private memory.
- Computational tasks can only operate on local data.
- If remote data is required, the computational task must communicate with one or more remote processors.
- The most popular distribute memory programming paradigm is MPI.



Parallel programming

Distributed memory: address system



Parallel programming

OpenMP vs MPI

- OpenMP
  - Easy to parallelize existing codes without much coding effort.
  - Look for iterative operations
  - Use OpenMP directives (pragma) to parallelize it. What it does is to create threads that will run parallel on different cores of the same node.
- MPI
  - If you want to scale your application beyond the maximum number of cores available on a node.
  - MPI is only a standard for message passing libraries based on the consensus of the MPI Forum,
  - There are different implementations of it.
  - Popular implementations are: OpenMPI, MPICH, Intel MPI.

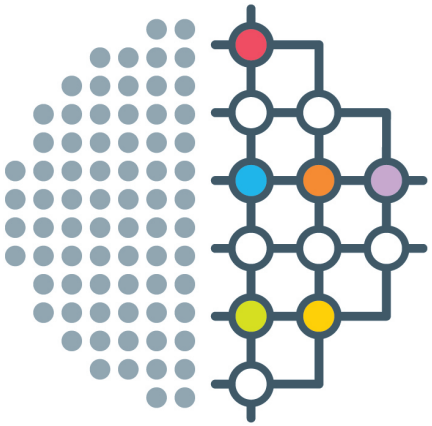
Parallel programming

Hybrid programming model

- Mix and match OpenMP and MPI
- Use OpenMP directives within a single node and MPI communications over the network within the same program.

Acknowledgments

- Slides developed by:
  - Jaison Mulerikkal, PhD
  - Peter Maxwell
- For:



NeSI
New Zealand eScience Infrastructure

THE UNIVERSITY OF AUCKLAND NEW ZEALAND
UC UNIVERSITY OF CANTERBURY
NIWA Taihoro Nukurangi
ministry of science + innovation
Manaaki Whenua Landcare Research
UNIVERSITY of OTAGO NEW ZEALAND

Acknowledgments

Reference

- Blaise Barney, Introduction to Parallel Computing, LLNL