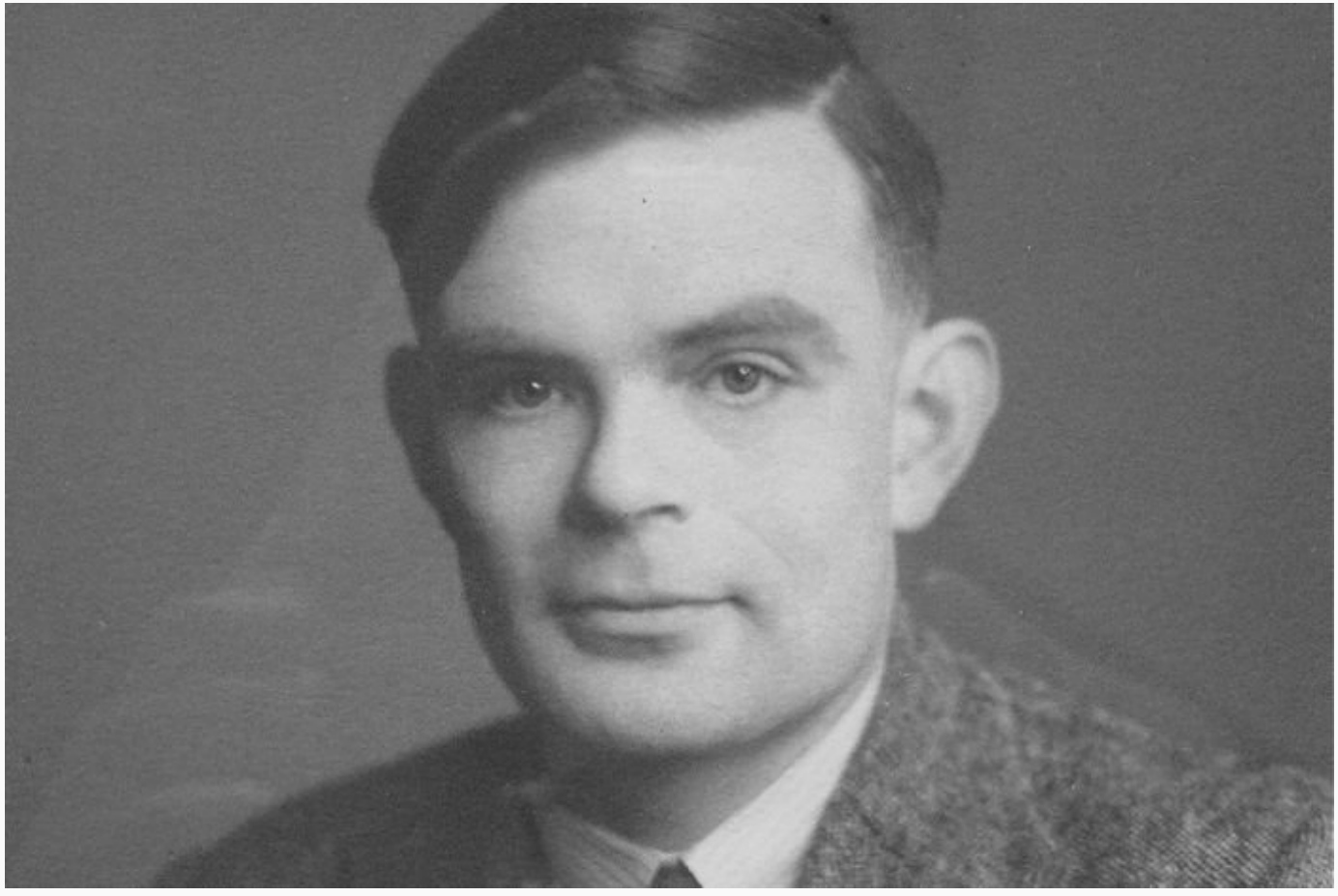
Introduction to Parallel Computing

NeSI Computational Science Team
support@nesi.org.nz

Outline

Story of computing
The Beginning

Alan Turing



Story of computing

Turing 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, winning the battle of Atlantic!
- He conceived the principles of modern computers and introduced his famous "Turing Machine" in 1936.
- "Turing Machine" is a hypothetical device
  - It manipulates symbols on a strip of tape based on a table of rules.
  - It 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, the 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 situation of two people coming from two different backgrounds and laying foundations of computer science.

Story of computing
Need for speed

Seymour Cray

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

Story of computing

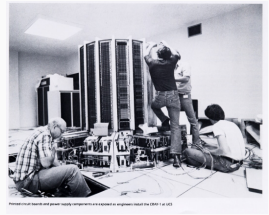
Cray-1

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.



Cray-2 had only 1 gigaflops peak performance with a cost as high as $32,000 per megaflops, whereas your PC will have around 40 gigaflops peak performance at $0.04 cost per megaflop ("Flops" stands for floating point operations per second).

Hegelian dialectics

Thesis: one

Two rules

- If we look carefully at the history of computing, we could see the Hegelian cycles of thesis and anti-thesis resulting in a synthesis.
- Computing, or even supercomputing started with monolithic machines getting bigger and faster. 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 processing speed of 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 speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program."
- The famous Amdahl's equation below is not in this paper, but only a small literal description in the 4th paragraph of the 4 page paper!

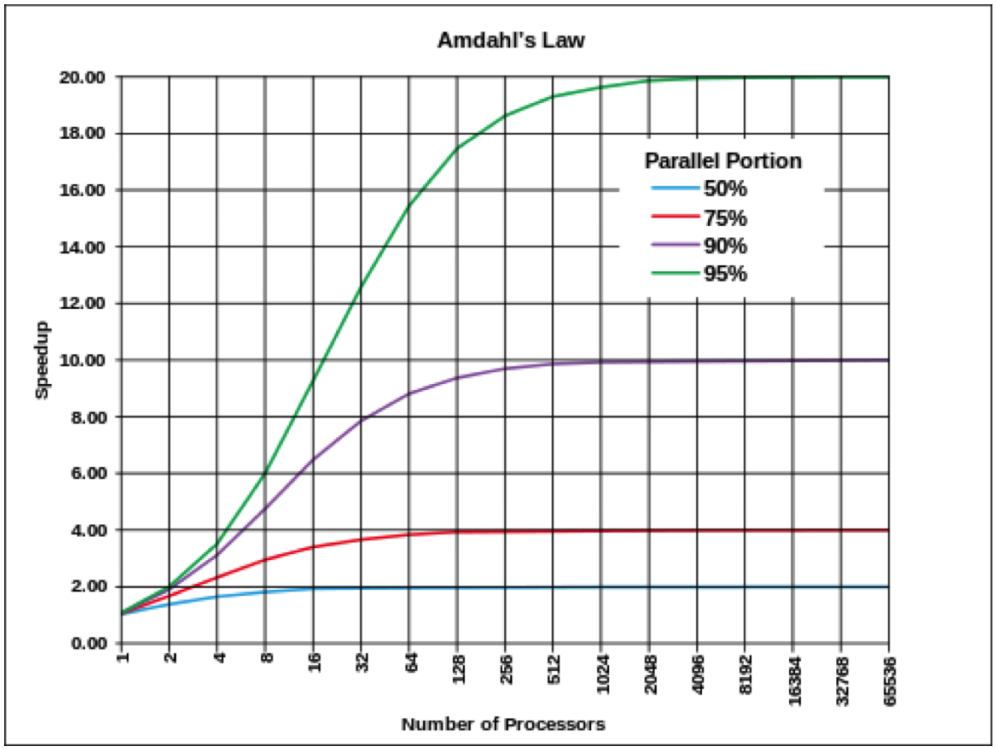
Hegelian dialectics

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.
  - S is speedup.
- The idea of Amdahl's law was to show the limitations of parallel computing!

Hegelian dialectics



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

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 parallelisable.
  - There is practically no other way to increase speed in many cases, even if theoretically this may 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!

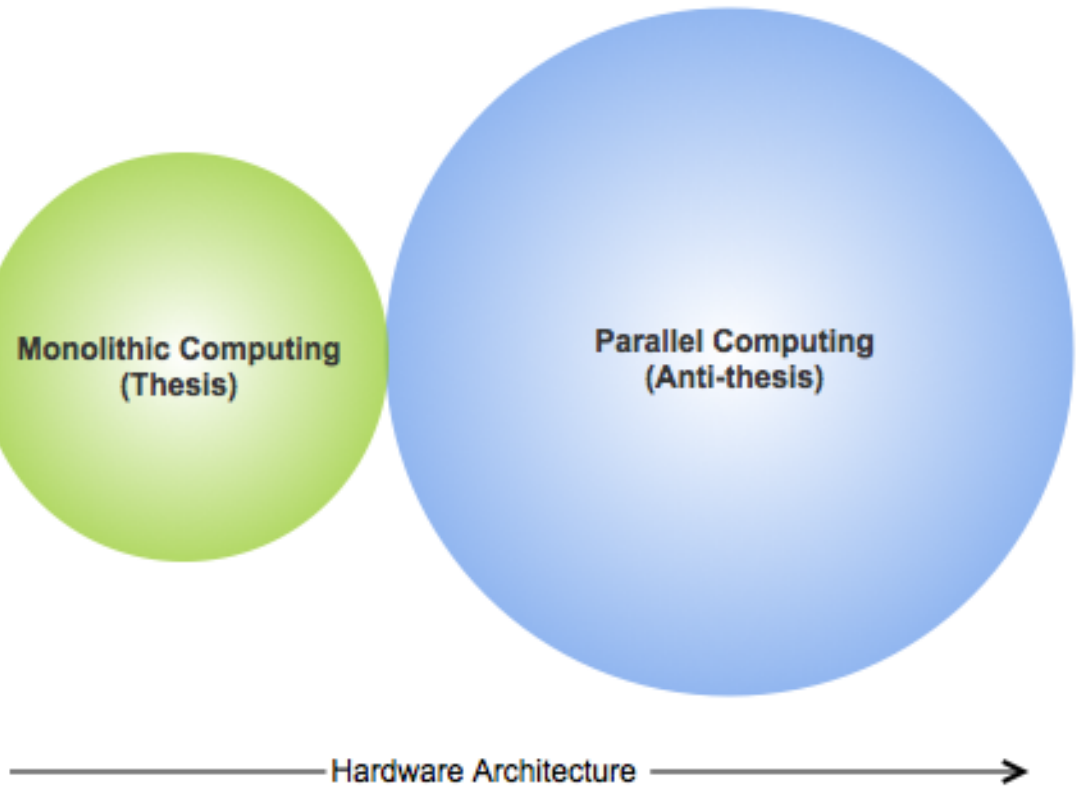
- Amdahl's law actually revealed the possibility of parallel computing:
  - There is actual increase in speed, if the algorithm is parallelisable.
  - There is practically no other way to increase speed in many cases, even if theoretically this may not the most efficient way to do it.
- Collapse of Moore's 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.

Hegelian dialectics

Gene Amdahl

Evolution of computing - 1



Parallel computing

Background

Parallel computing emerged as an anti-thesis.

- It is the art of breaking up a big chunk of serial computation into smaller atomic units which can be done in parallel.
  Factors that helped:
  - The arrival of cheap commodity computing hardware.
  - A theoretical possibility of achieving comparable speeds to serial HPC computing using parallel computing.
  - Recognition that nature itself is parallel, however complex it may appear.

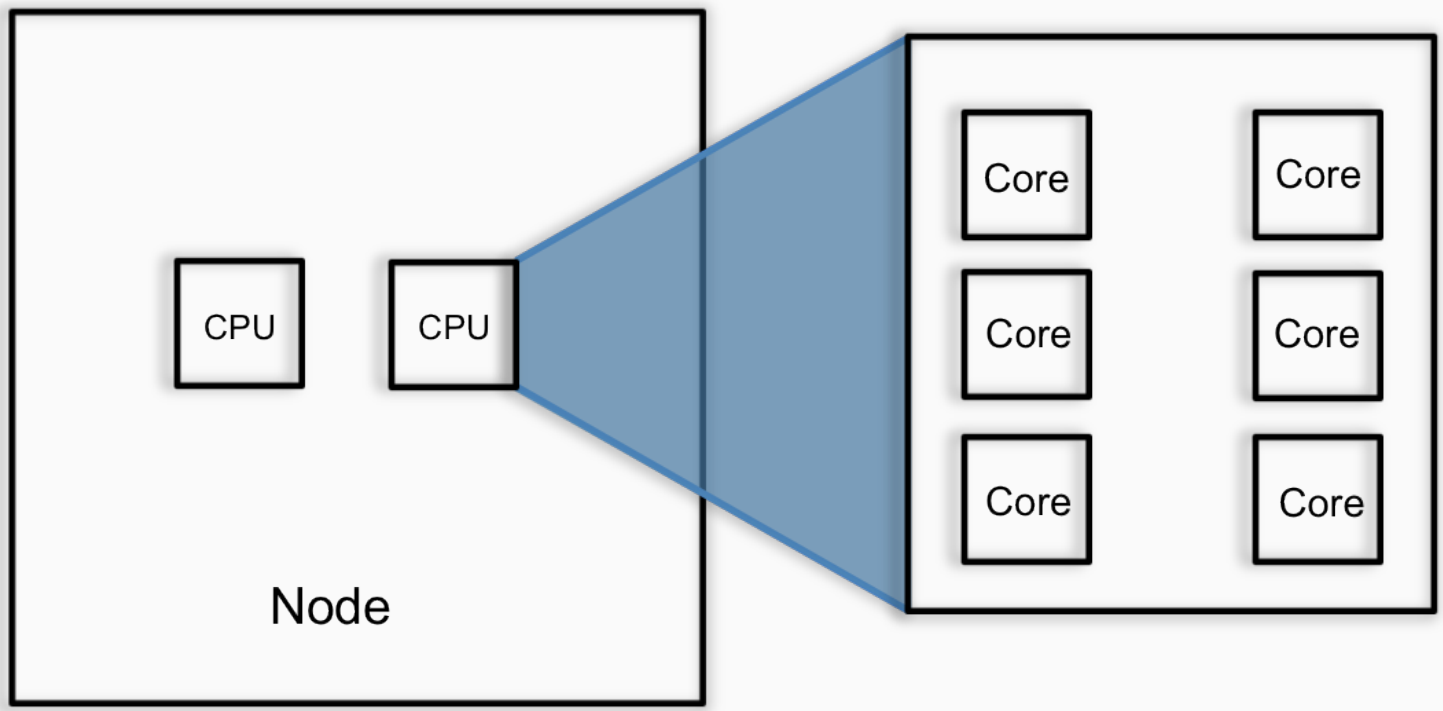
Parallel computing

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 can use more than one core, it runs faster on a cluster.

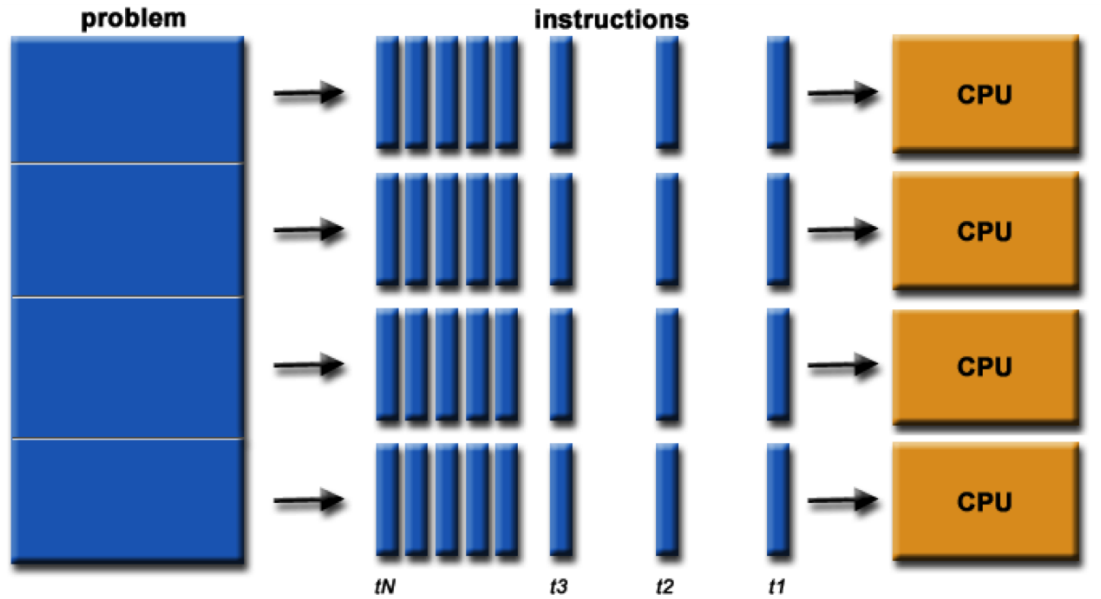
Parallel computing

Node overview



Parallel computing
Principles

- Breaking down of a problem into discrete parts that can be solved concurrently.
- Each partial solution consists of a series of instructions.
- Set of instructions are executed simultaneously on different processors.



Evolution of parallel computing

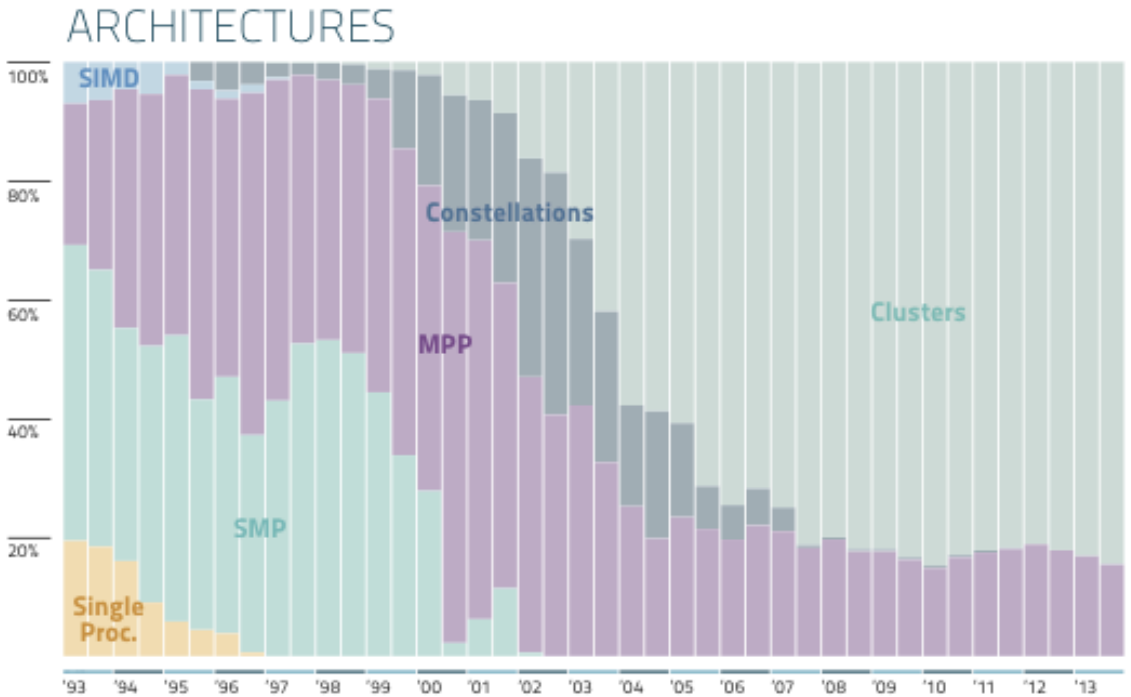
Grid & SOA

- Grid Computing
  - A distributed computing model that orchestrates computing resources of several academic and research institutions to work like a unified computing system.
- Service Oriented Approach (SOA)
  - In SOA applications are composed by invoking network available services to accomplish some tasks.
  - This paradigm is adopted mainly by business and enterprise community.

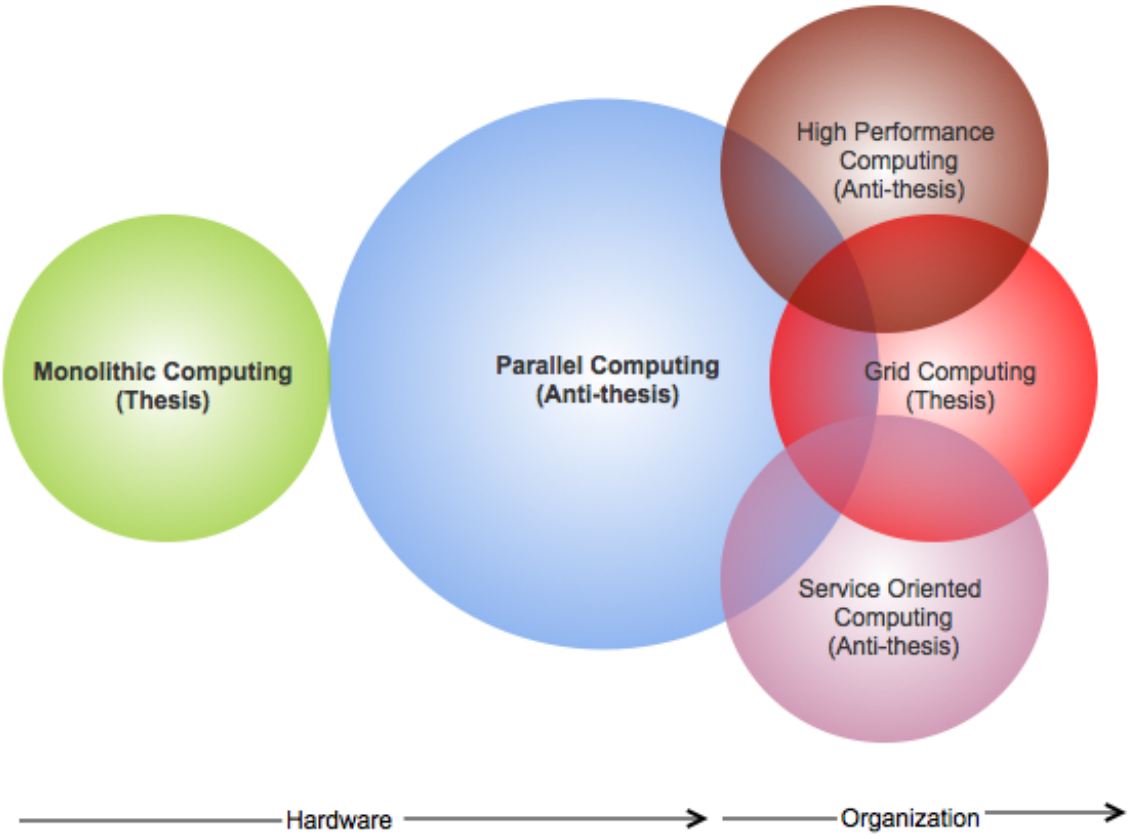
Evolution of parallel computing

High performance computing

- In 2002, TOP500 list reported that 20 percent of HPC installations as "clusters". This marked the emergence of parallel computing as a serious platform for HPC.
- By 2013, 80 % of TOP500 supercomputers were "clusters".



Evolution of computing - 2



Hegelian synthesis

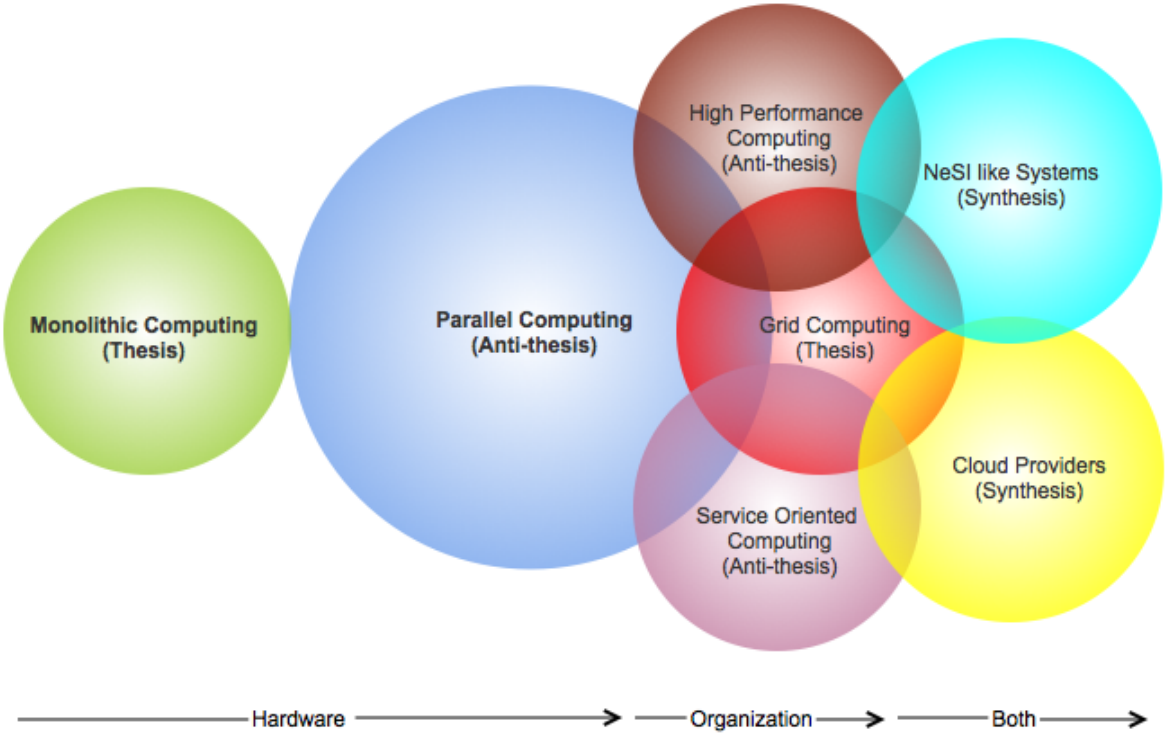
NeSI-like systems

- 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.
  - Orchestrate storage, memory and network resources of geographically distributed data centers as a unified unit.
  - The resources are made available to customers on demand with apparent elasticity.

Evolution of computing - 3



NeSI Systems

Facilites

NeSI HPC resources cater to New Zealand's research and academic community and are spread between three major facilities:

- BlueFern Supercomputer at the University of Canterbury, Christchurch.
- NIWA HPC Facility, Wellington.
- Centre for e-Research at the University of Auckland.

There is a single **grid** interface called **Grisu** that can be used 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

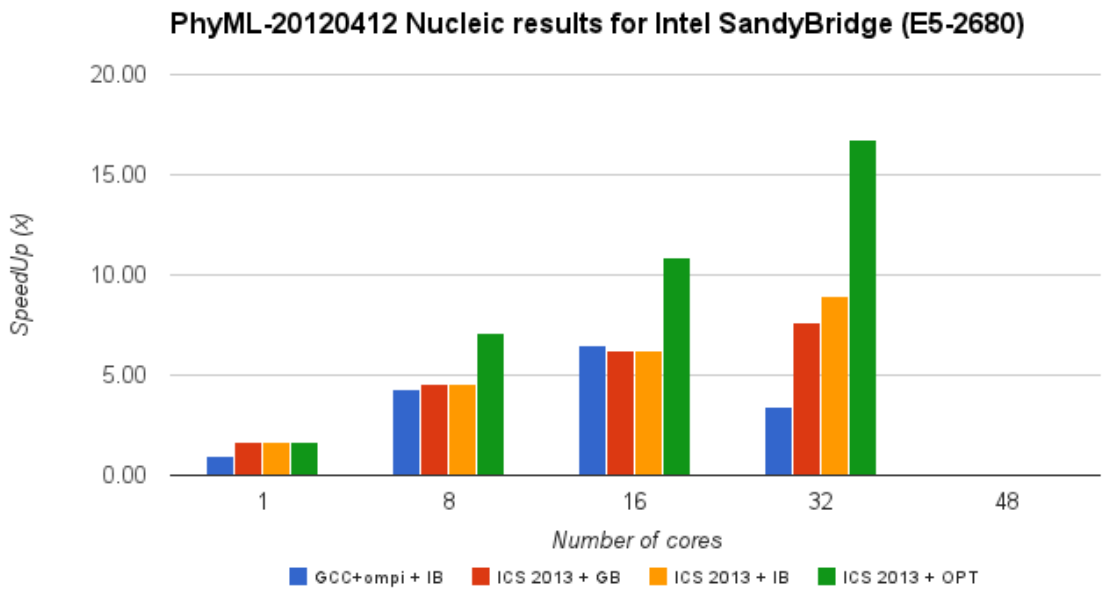
Many general purpose scientific applications are already available in NeSI systems.

- **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: on Pan cluster

- **PhyML** is a software that estimates maximum likelihood phylogenies.
- The right compilers and optimization options for a specific architecture can increase the performance quite a lot!



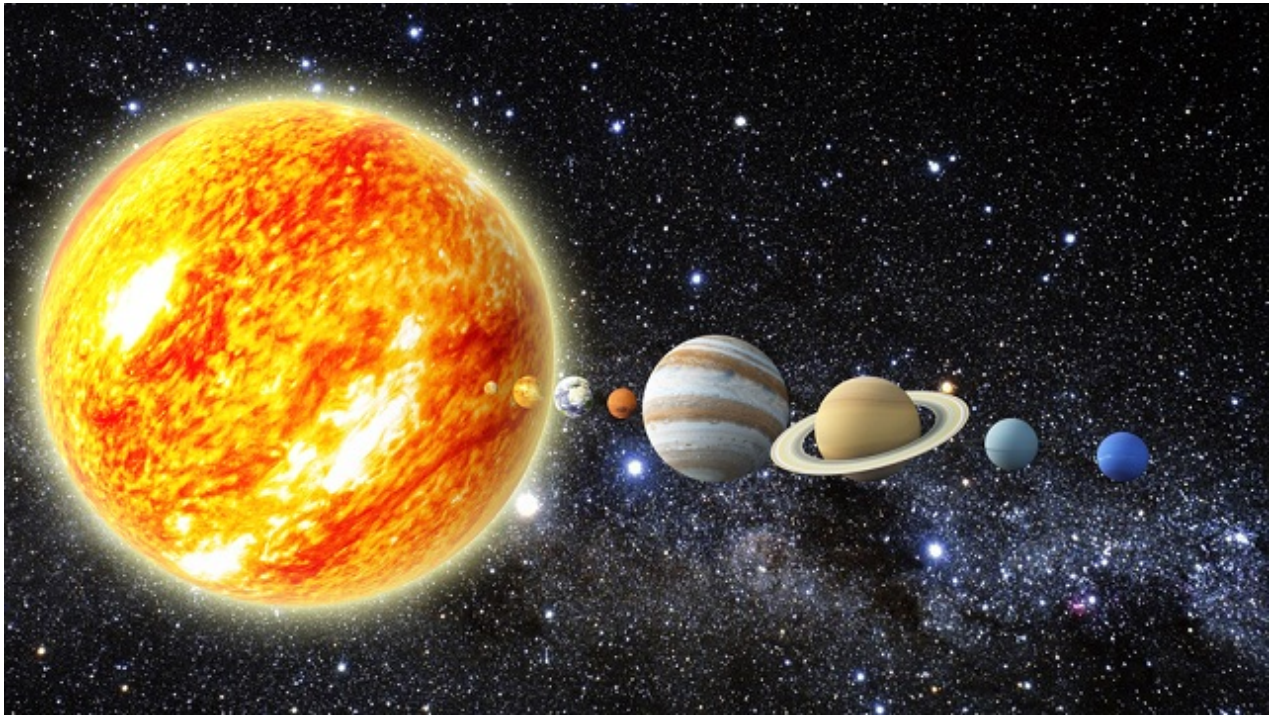
Parallel programming

Rationale

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

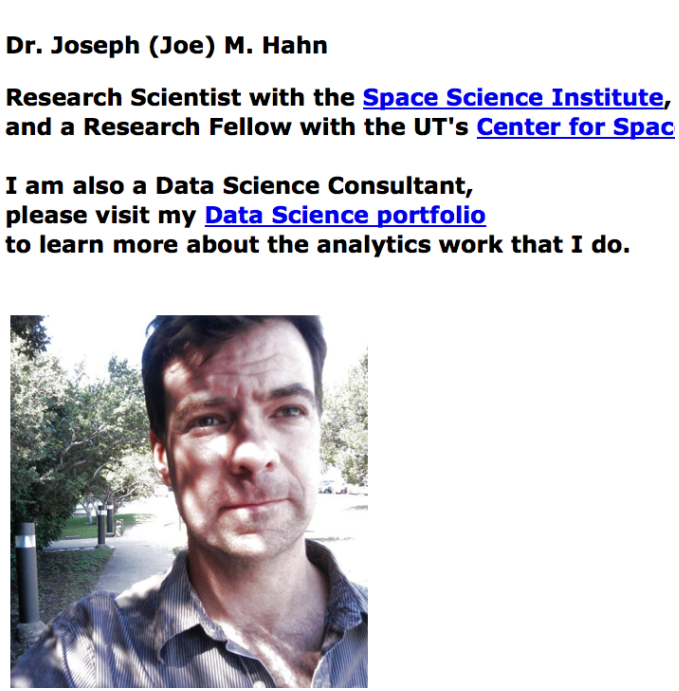
Parallel problems

Case Study: N Body Problem



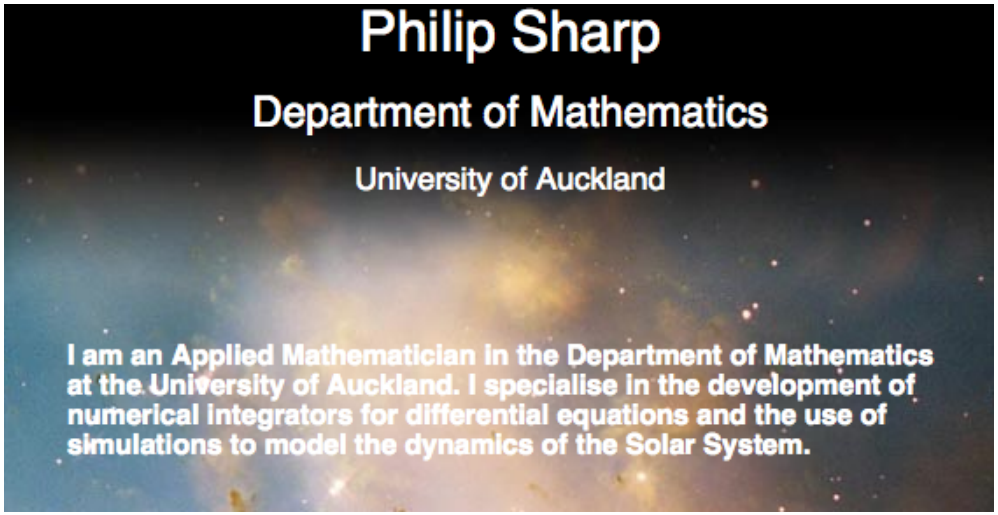
Gravitational n-body problem (Newton's laws of force)
Force is a function of mass and acceleration ($f = m \times a$)

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 Institute, Colorado had to come up with a variant 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 parallelise an algorithm.

Hotspots

- Hotspots are the areas where we have massive iterative works, which can be parallelized into smaller units that are independent of each other.
- Those units of work are called tasks and such algorithms are often described as “embarrassingly parallel”.

Parallel programming
Case Study: N Body Problem

Bottlenecks

- Bottlenecks are the places where computations become inter-dependent. Usually there is a need to synchronise the data before we begin 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 parallelise efficiently.

Direct n-body problem

Governing equations

$$\mathbf{f}_{ij} = G\frac{m_i m_j}{||\mathbf{r}_{ij}||^2} \cdot \frac{\mathbf{r}_{ij}}{||\mathbf{r}_{ij}||} \tag{1}$$

$$\mathbf{F}_i = \sum_{1 \leq j \leq N} \mathbf{f}_{ij} \quad , \; j \neq i \tag{2}$$

$$\mathbf{F}_i = m_i \mathbf{a}_i \tag{3}$$

$$\mathbf{p}_i = \iint \mathbf{a}_i \, \mathrm{d}t \tag{4}$$

Where:

- Force (F) is a function of mass and acceleration (2).
- Position (p) is a function of velocity/acceleration and time (4).

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
  - 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 them.
- Functional/task decomposition
  - An alternative strategy where we assign different functions to different processors.
  - When algorithms have no obvious data structure that can be decomposed.
  - It could be 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

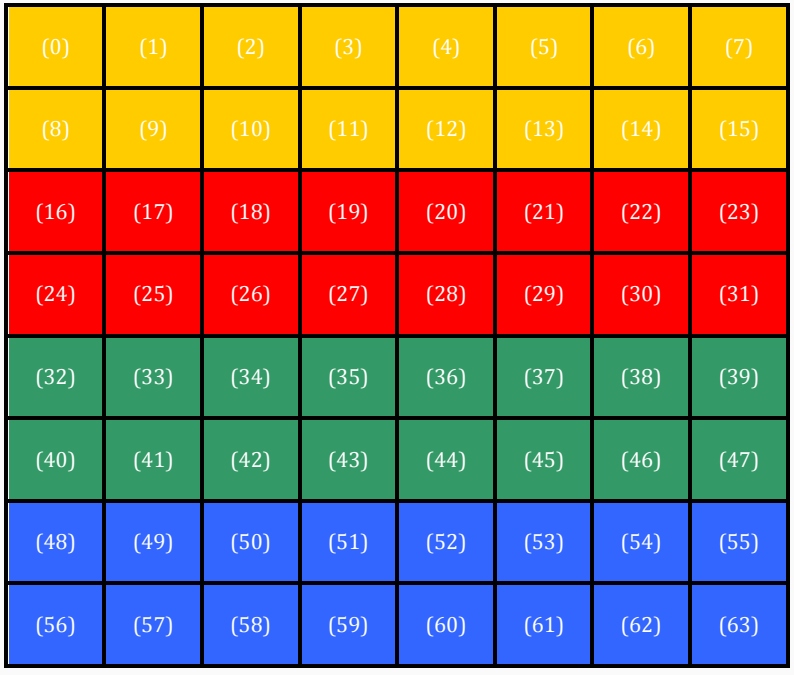
Parallel decomposition

- We can visualize bodies in direct n-body problem as elements of an 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)

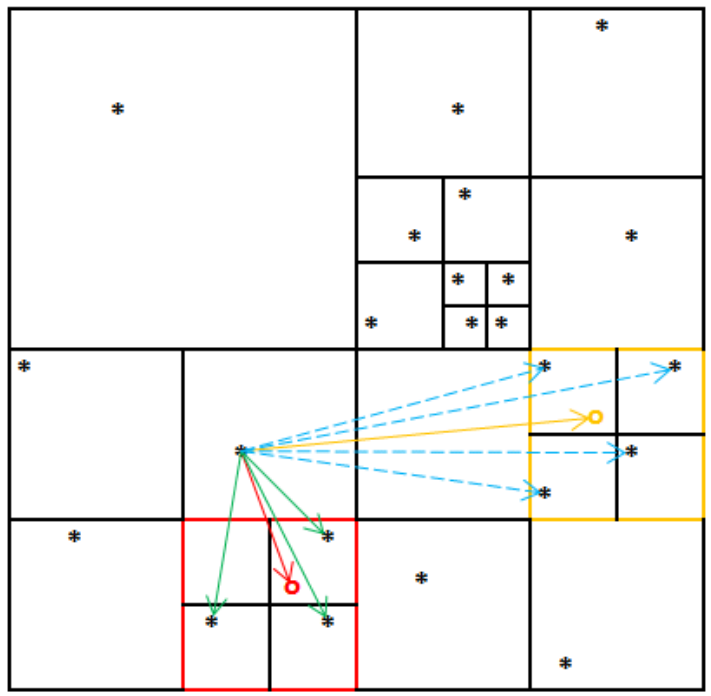
Direct n-body problem

Parallel decomposition

- The algorithm is reduced to parallel matrix-vector operations.
- However, all nodes need to have access to all the data in this model, which could put constraints on memory.
- There are other n-body algorithms to mitigate this issue, where bodies are geographically grouped together and considered as a large single body (Example: Barnes–Hut method).

N-body problem

Barnes–Hut method



Technically, it reduces the complexity of problem from $O(n^2)$to $O(n \times logn)$

Parallel decomposition

Heat diffusion problem

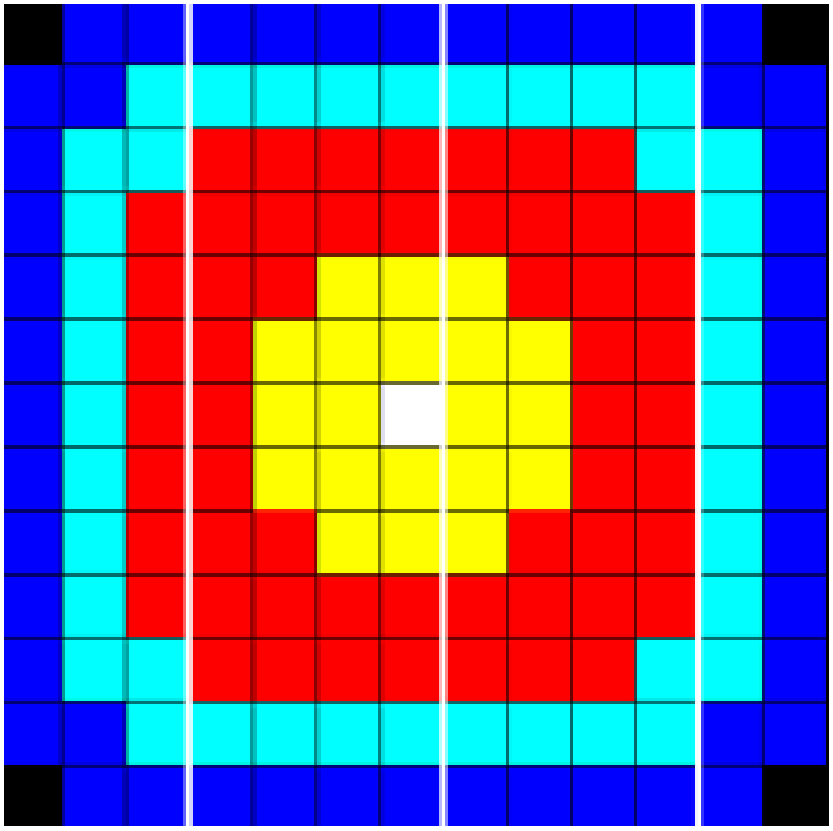


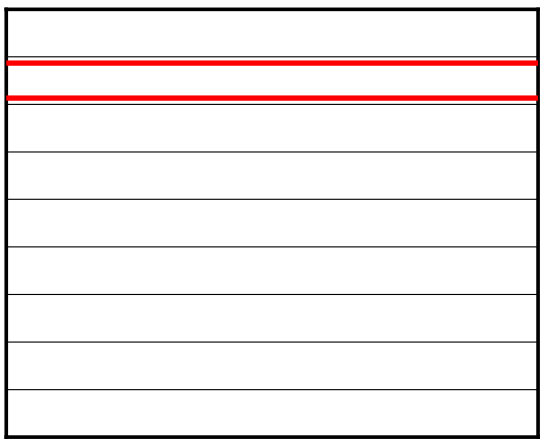
Figure : Heat diffusion on a 2D plate

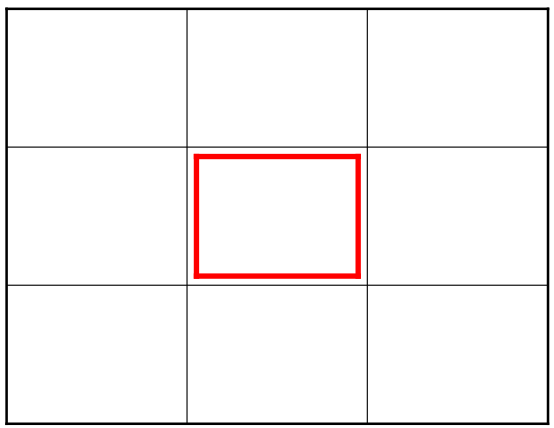
Crux: To update each cell, we need information about all its neighbouring cells.

Parallel decomposition

Heat diffusion problem

The economy of decomposition



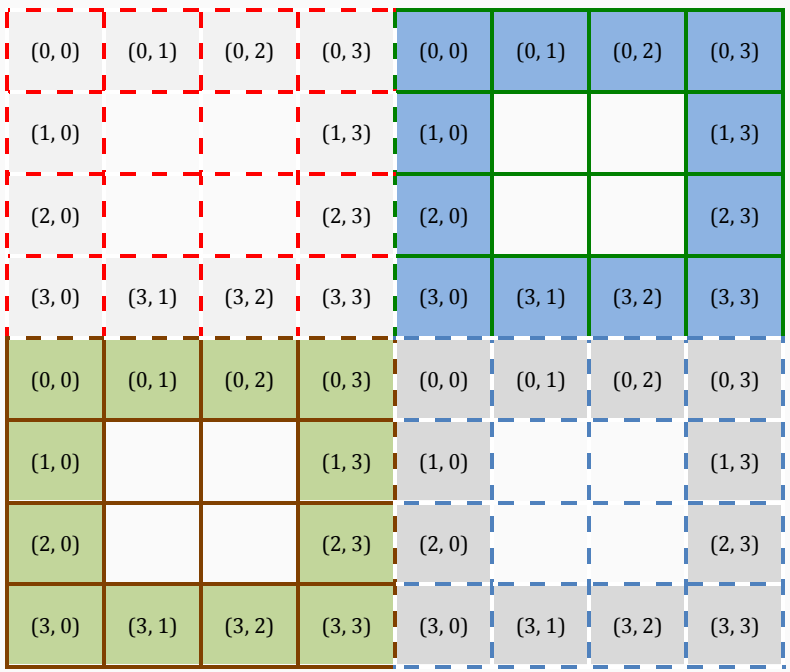


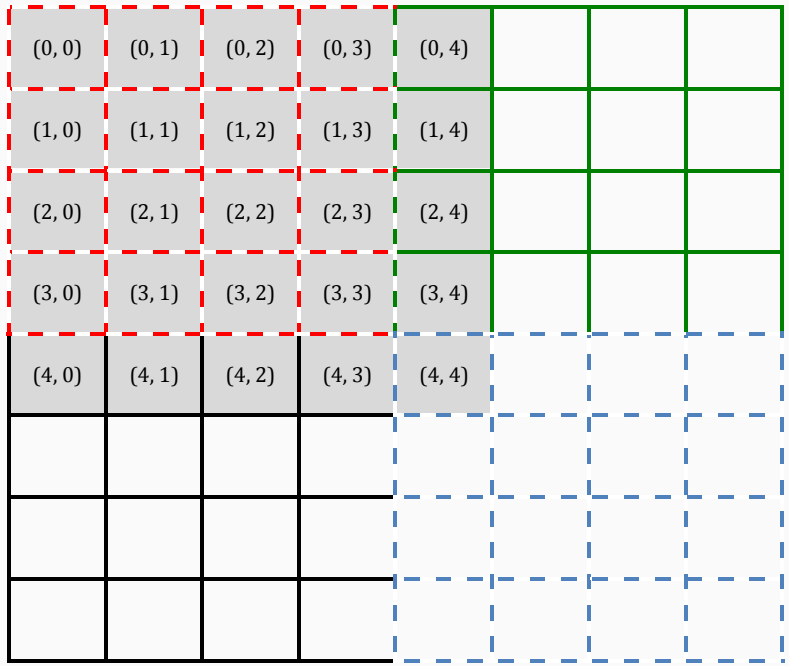
Partitioning as near-square rectangular blocks will give an advantage of $4 \times (\sqrt{p}$ - $1)N/\sqrt{p}$ times over partitioning as rows (where p is total perimeter and N is number of partitions) in terms of the amount of data to be synchronized.

Parallel decomposition

Heat diffusion problem

Ghost cells





Data visualization after sharing ghost cells among the processors.

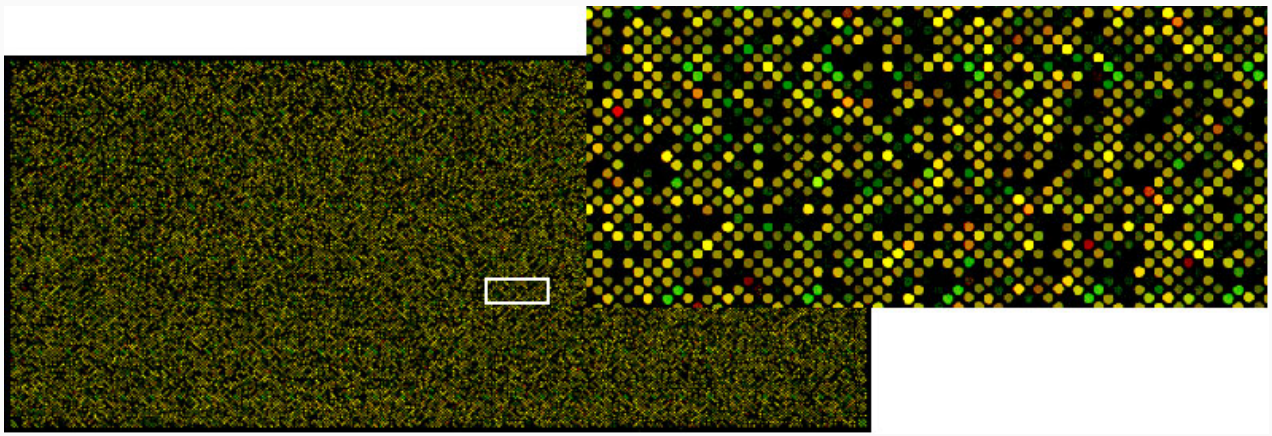
Parallel decomposition

Bioinformatics

Biological data can be big

A few examples:

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

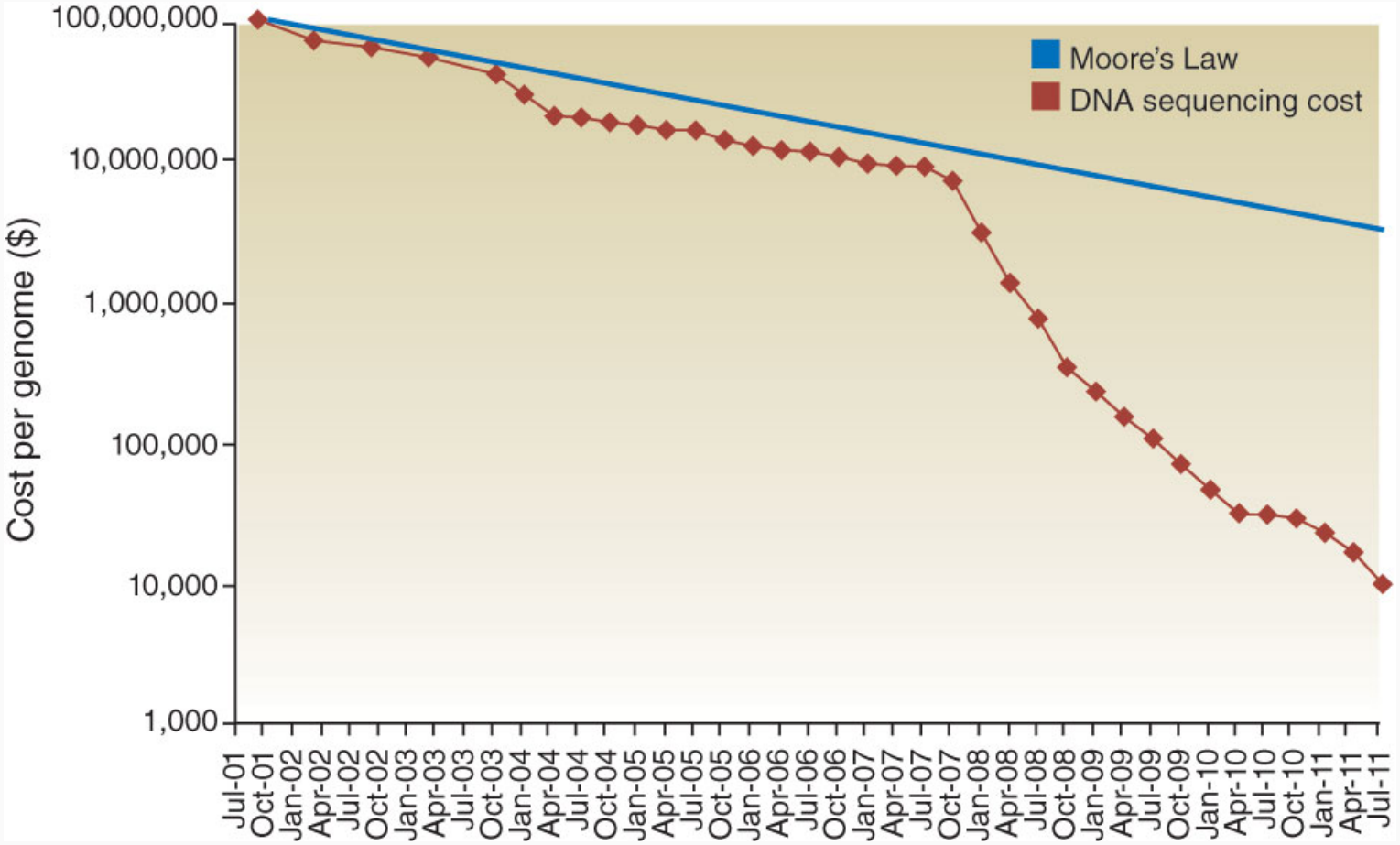


- Protein X-ray Crystallography.
- DNA Sequencing.

Parallel decomposition

Bioinformatics

DNA sequencing technology is outpacing Moore's Law

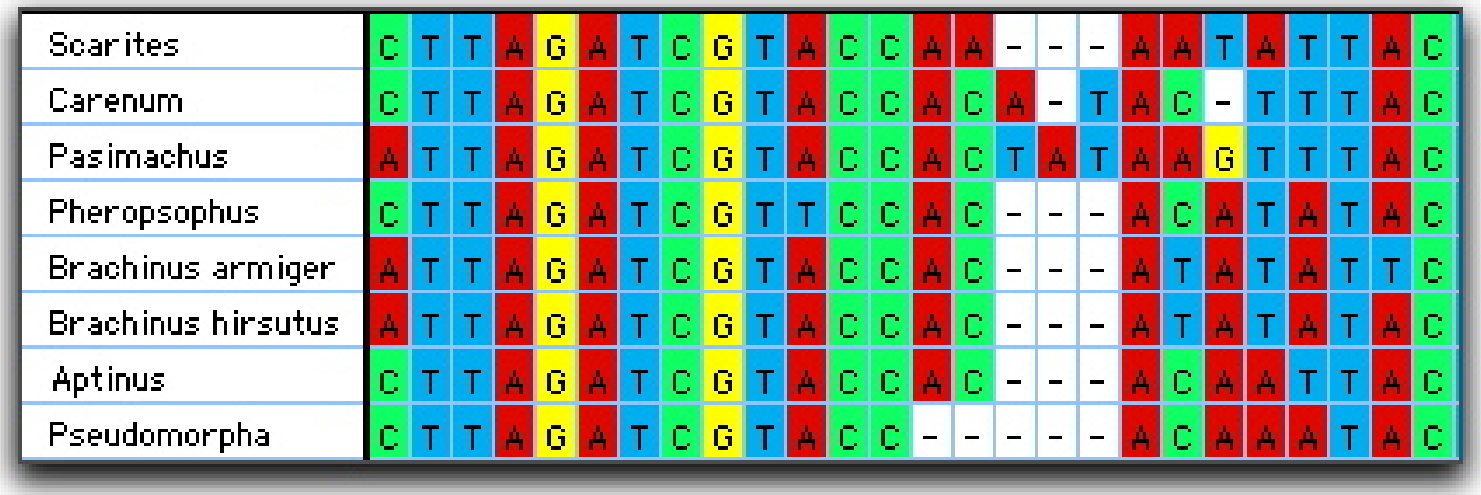


Parallel decomposition

Bioinformatics example

Multiple Sequence Alignment

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



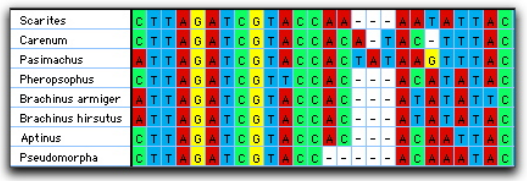
Parallel decomposition

Bioinformatics problem

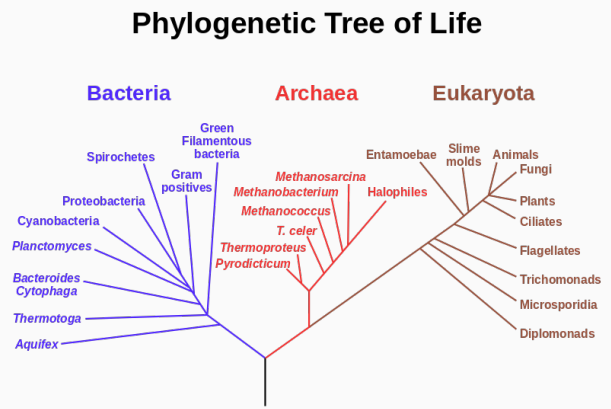
Pylogenetic Reconstruction

Derive a tree of descent from multiple aligned sequences.

Input



Output



Parallel problems - Bioinformatics

Evaluating the likelihood of a particular tree

Given a tree T, alignment A and substitution model S, the core of a likelihood calculation is:

for each parent node p in T (in a postorder traversal) **do**
  for each alignment column, a **do**
    $L_{pa} \leftarrow [1, 1, 1, 1]$
  end for
  for each child node, c, of p **do**
    for each alignment column, a **do**
      $L_{pa} \leftarrow L_{pa} \times (S_{pc} \times L_{ca})$
    end for
  end for
end for

Parallel decomposition

Bioinformatics problem

Parallelise tree evaluation by:

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

Parallelise tree optimisation by:

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

Parallelise tree evaluation by:

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

Parallelise tree optimisation by:

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

This slide is intentionally left blank.

Parallel programming

Memory Architecture

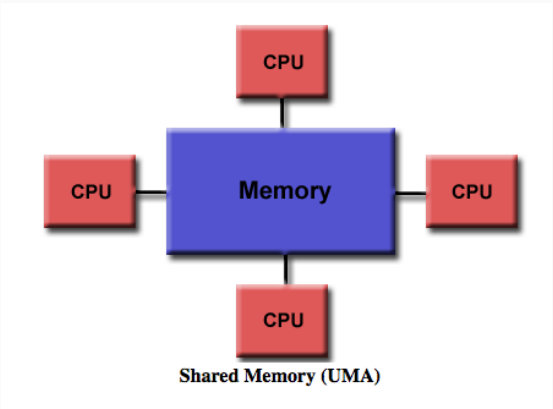
Architectural differences between memory architectures have implications for how we program.

- Shared-memory systems use a single address space, allowing processors to communicate through variables stored in a shared address space.
- In distributed-memory systems each processor has its own memory module and over a high speed network communications take place.

Parallel programming

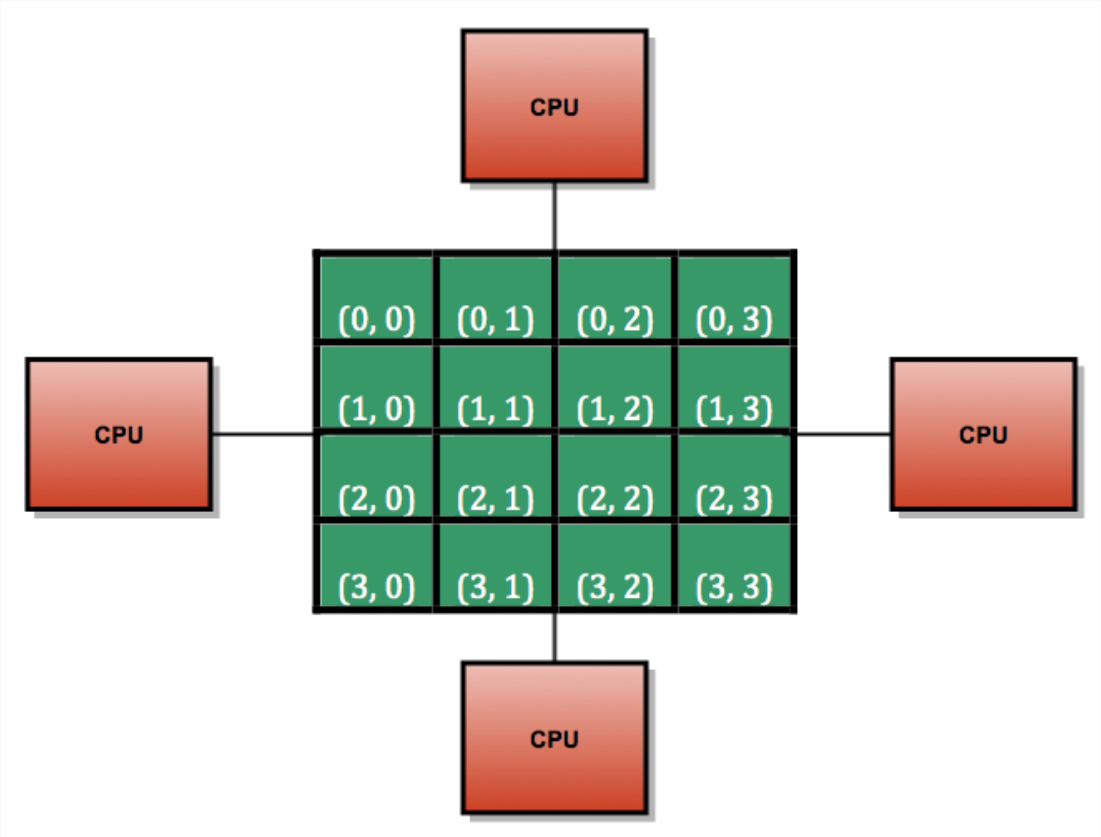
Shared memory

- Symmetric multiprocessing (SMP) : two or more identical processors share the same memory.
- This shared memory may be simultaneously accessed by multiple threads of same program.
- The most popular shared memory parallel programming paradigm is OpenMP.



Parallel programming
Shared memory

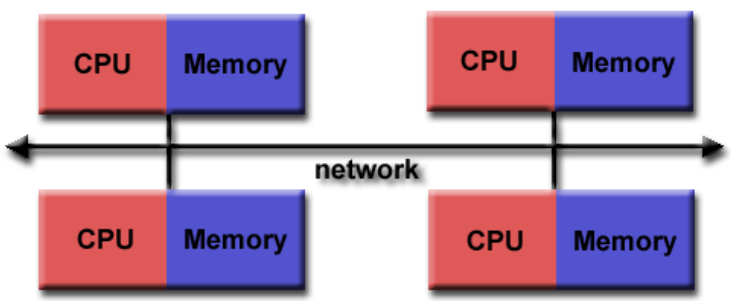
Shared memory: address system



Parallel programming

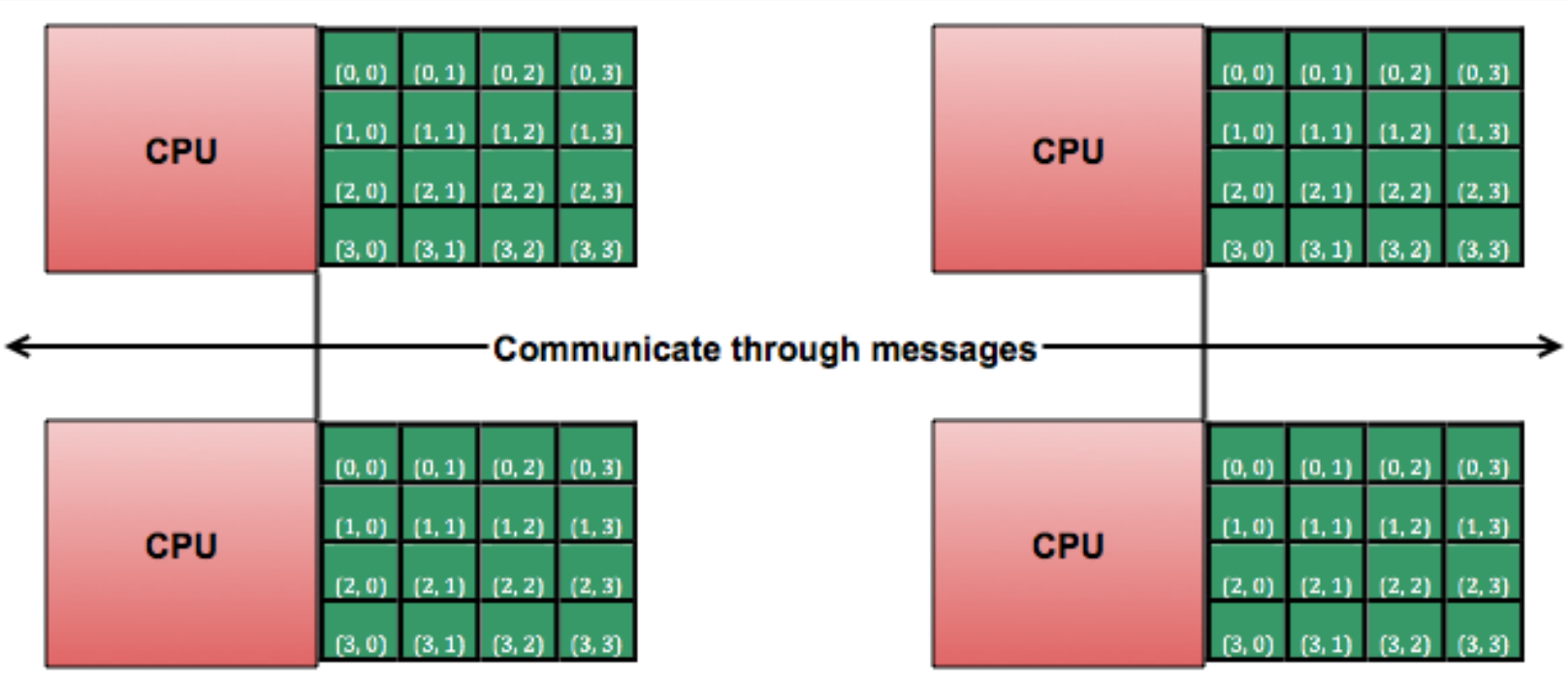
Distributed memory

- 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 distributed memory programming paradigm is MPI.



Parallel programming

Distributed memory: address system



Parallel programming

OpenMP vs MPI

- OpenMP
  - Easy to parallelise existing codes without much coding effort.
  - Look for iterative operations.
  - Use OpenMP directives (pragma) to parallelise it. These create threads that will run in 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 include: OpenMPI, MPICH, Intel MPI.

Parallel programming

Hybrid programming model

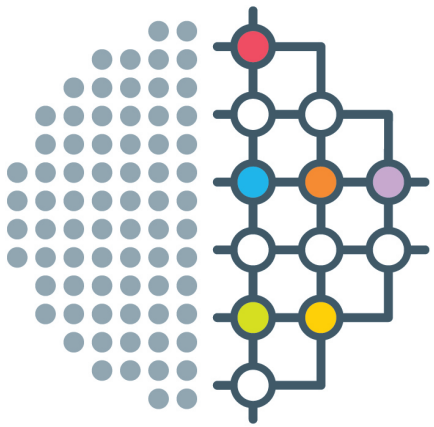
- Mix and match OpenMP and MPI:
  - OpenMP manages the workload on each (SMP) node.
  - MPI facilitates communication among multiple nodes over the network.
- The best of both worlds:
  - OpenMP ensures efficient utilization of shared memory within a (SMP) node.
  - MPI facilitates efficient inter-node operations and sending of complex data structures.

Parallel programming

Thank You
Questions?

Acknowledgments

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



NeSI
New Zealand eScience Infrastructure

THE UNIVERSITY OF AUCKLAND NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

UC UNIVERSITY OF CANTERBURY

NIWA Taihoro Nukurangi

ministry of science + innovation

Manaaki Whenua Landcare Research

UNIVERSITY of OTAGO Te Whare Wānanga o Otāgo NEW ZEALAND

Acknowledgments

Reference

- Blaise Barney, Introduction to Parallel Computing, LLNL.
- Dr John Rugis, NeSI.