Deciding how to Parallelise a Problem

Introduction

We are going to look at two separate, but closely related aspects of how to parallelise a problem:

- Parallel Decomposition
  - How to split the problem into portions that can be solved in parallel
- Parallel Communication Architecture
  - How to communicate between nodes to ensure that they have the data required to solve the problem

Parallel Decomposition

Introduction

- There are a vast number of different ways in which you can potentially split problems in order solve them in parallel
- Even for a given problem there is usually not a single way in which it might be split
  - Not even necessarily a single best way to split a problem
  - May depend on computer resources available:
    - Number of cores available
    - Amount of memory available of each node
    - Relative speed of processors vs communications
- We will therefore only be looking at a limited set of common ways to split a problem

Data Decomposition

Split the data between the processes:

- Split the output data

- Split the input data

- Split both the input and output data

Data Decomposition: Split the output data

- In many problems, given complete knowledge of the input data, different portions of the output data can be calculated independently
- Example of where it is appropriate
  - Matrix multiplication – Each element in the answer matrix can be calculated independently of the others
  - Communication is required to distribute the input data, but not between the nodes

Do Worksheet 5 Exercise 1

Data Decomposition
Split the input data

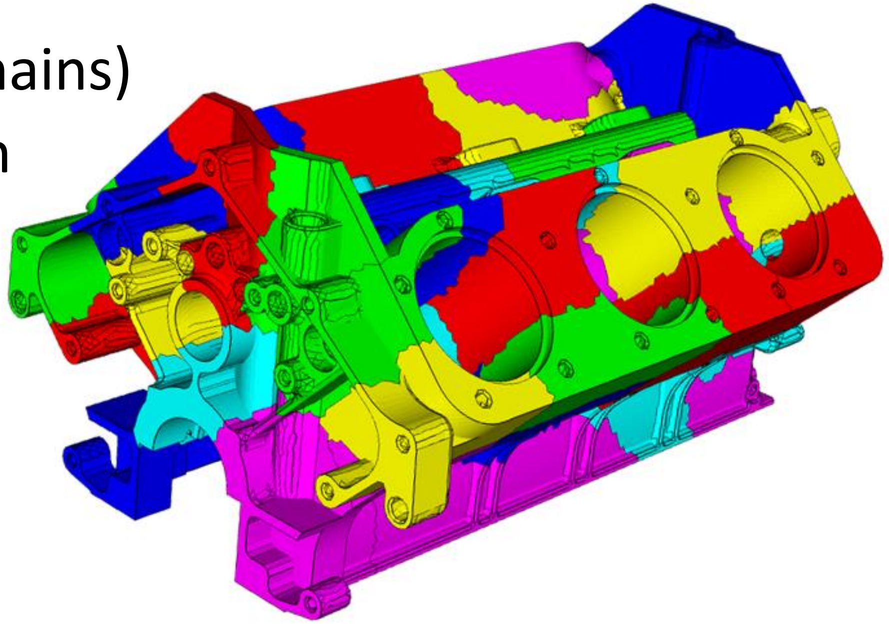
- Different portions of the input data assigned to different processes
- Contributing to a commonly held solution
  - Either globally known (e.g. a shared memory system) or known to a single master node

Data Decomposition
Split both the input and output data

- Different portions of the input and output data in a problem may be closely related
  - E.g. If simulating a physical system there may be input data on about a region or time, with some other properties of the system being calculated for those positions or times
- Very common approach in both distributed and shared memory systems
  - In distributed memory systems this will often require communication of information at the boundary of the data
  - In distributed memory system it restricts the need for blocks to memory associated with the edge of the data regions

Data Decomposition
Domain Decomposition

- Domain decomposition is a very commonly used example where both input and output data is split
- Used in the simulation of physical systems
  - The system is divided into a set of regions (domains)
  - Each process responsible for a different domain
  - Communication of data at the edge of domains



The next lecture will look at a domain decomposition example in detail

Data Decomposition
Load Balancing

- To ensure maximum parallel efficiency you don’t want some of the processes waiting idle while other processes work
- Need to spread the computational load – try to give each process the same amount of work to do
  - Typically responsible for the same number of degrees of freedom (e.g. the same number of nodes or elements)
  - If the simulation resolution is spatially constant this may be equivalent to evening splitting the sizes of the regions
- Need to simultaneously try to keep communication to a minimum
  - Reduce the surface area of the regions

Domain Decomposition
Balance between computation and communication

- In strongly coupled problems if you try to decrease the computational time by increasing the number of processes the computational efficiency will decrease
  - Proportion of time spent communicating will increase
  - Slightly different to Amdahl's law as it applies even if the entire problem is parallelised
- If we assume a constant resolution then, for a 3D system, computational time will be roughly proportional to the volume of a domain and communication to the surface area of the domain (volume of the domain raised to the power 2/3)
  - If V is the total volume of the system and N is the number of cores (k_1 and k_2 are constants associated with computational and communication speed)
  - Parallel efficiency (time taken if serial time scaled perfectly with number of cores divided by the actual time taken):

$$E_{3D} \approx \frac{k_1 \frac{V}{N}}{k_1 \frac{V}{N} + k_2 \left(\frac{V}{N}\right)^{\frac{2}{3}}} = \frac{1}{1 + \frac{k_2}{k_1}\left(\frac{V}{N}\right)^{-\frac{1}{3}}}$$

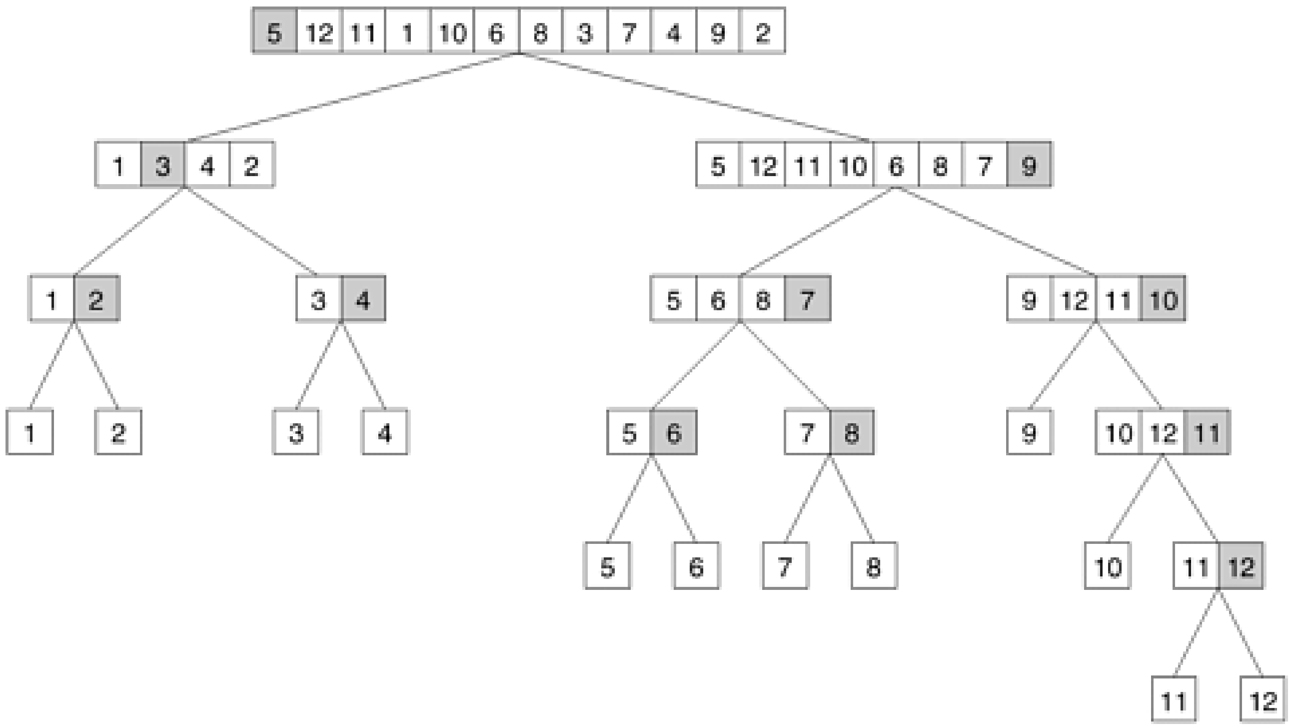
$$E_{2D} \approx \frac{1}{1 + \frac{k_2}{k_1}\left(\frac{V}{N}\right)^{-\frac{1}{2}}}$$

Recursive Decomposition

- Progressively split a problem into smaller portions and then combine the solutions
  - Works best for problems where the computational cost of the solution increases faster than linearly with the size of the problem and where the cost of splitting and combining solutions is comparatively cheap
  - So called “divide and conquer” algorithms
  - Often used in serial algorithms, but can also be used as the basis for parallel decomposition

Recursive Decomposition - Quicksort

- Classic example of recursive decomposition is the quicksort algorithm
  - Brute force sorting of *n* items is O(n^2)
  - …,but you can split lists in O(*n*) time and combine them in O(*1*) time



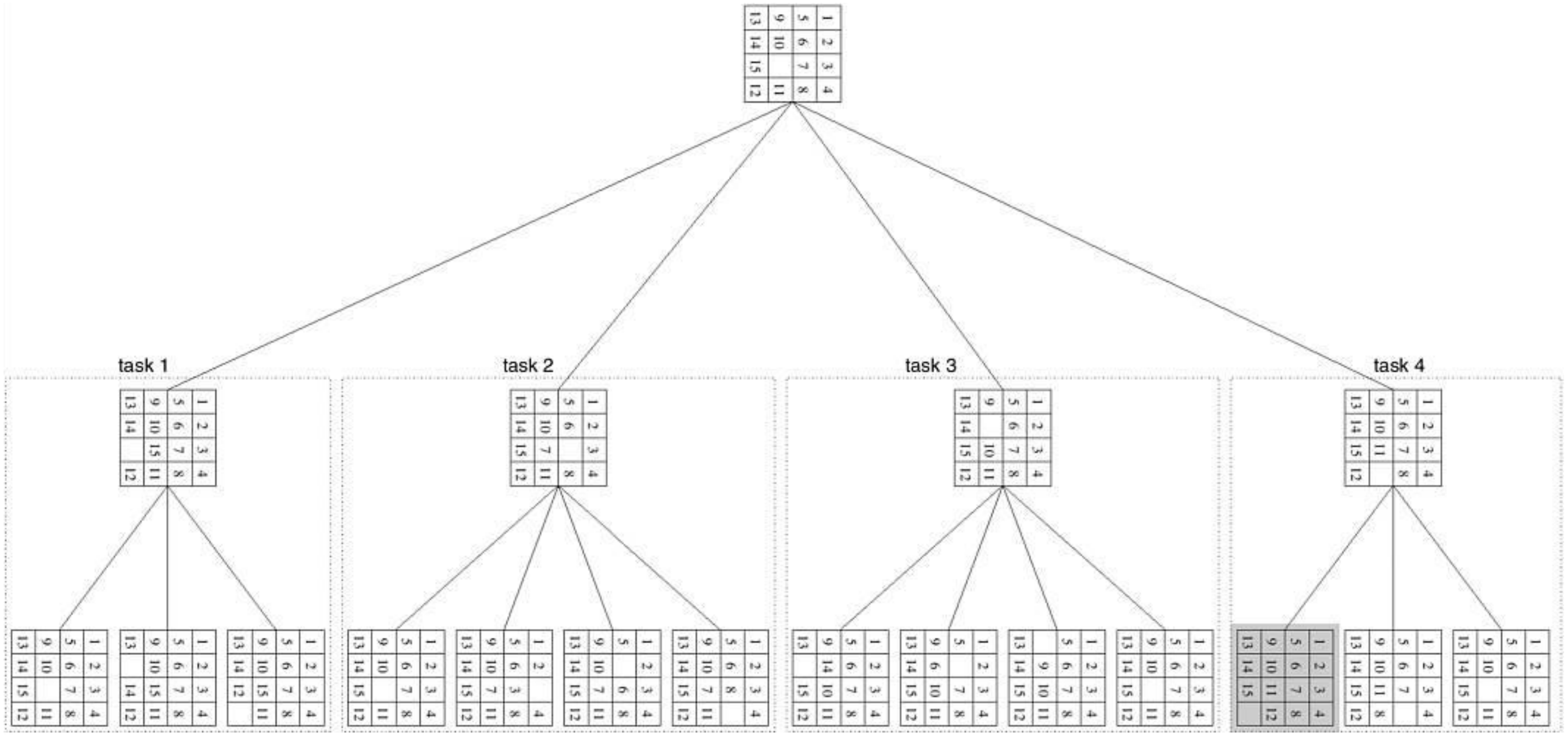
Do Worksheet 5 Exercise 2

Exploratory Decomposition

- Used for searching for solutions in multi-step problems
- Has some similarity to data decomposition in that the initial search space is split between processes
  - Note that this might not a true split of data, but could be a split of a parameter space
- The next stage is that a new set of data/parameters are produced from the previous stage and these are then split
  - Can be split onto new processes if available, otherwise the current processes become responsible for more states
  - Could allow processes to become available again if their search hits a dead-end

Exploratory Decomposition Example

- Finding solution to 15 puzzle problem



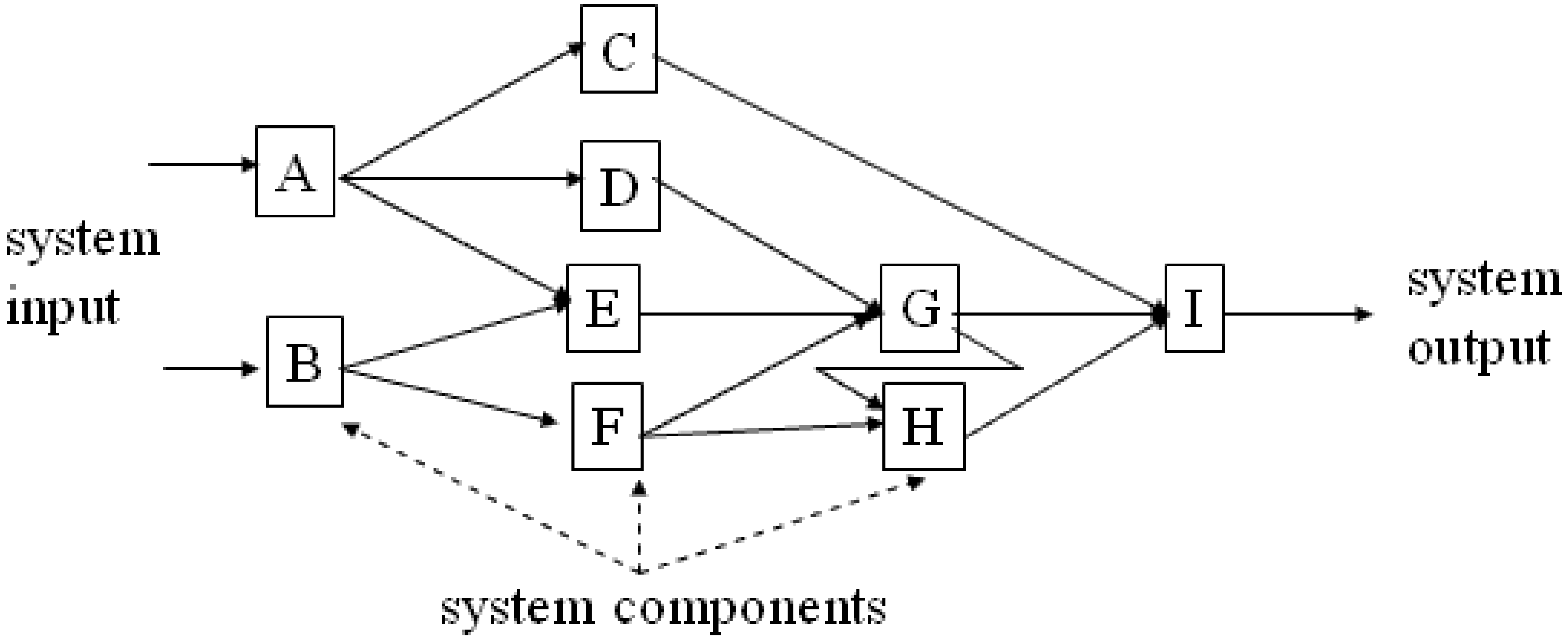
This is the Workshop Exercise

Speculative Decomposition

- Can be used in problems where subsequent tasks depend on the outcome of earlier tasks
  - E.g. two different tasks to complete depending on whether the solution to an earlier task is true or false
- In speculative decomposition you carry out all the subsequent tasks without waiting for the result from the earlier task
- Particularly advantageous if the earlier tasks take a comparable amount of time to the subsequent tasks

Speculative Decomposition Example

- Simulation of system with multiple interacting components



Parallel Communication Architectures

Different Types of Architecture

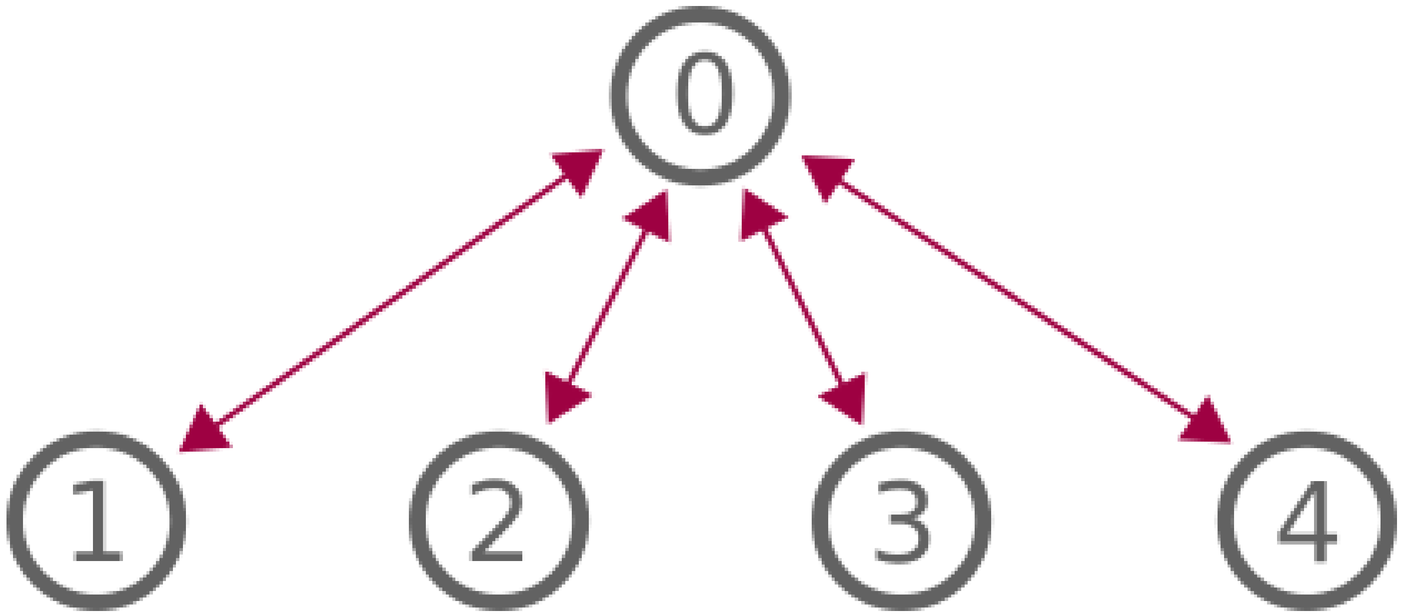
- As MPI simply provides the communication tools, it does not dictate the parallel architecture that will be used

Two main types of architecture

- Master/Slave
  - A master node controls the other nodes
  - …people are looking to change the name, but no agreement on replacement
- Peer to Peer
  - All nodes are equivalent to one anther

Master/Slave

- What is a Master-Slave architecture?
  - All communications go through a master node that controls a set of nodes that do tasks for it



Master/Slave

- Advantages:
  - Often relatively straightforward to implement
  - A single process has access to all the data
- Disadvantages:
  - Scalability issues as communications into the master can become a bottleneck

Master/Slave

When might you consider using a master/slave architecture:

- Trivially parallel problems
  - Problems with sub-tasks that can be carried out completely independently
  - Master farms out the problems to all the slave nodes and waits for the answers to come back

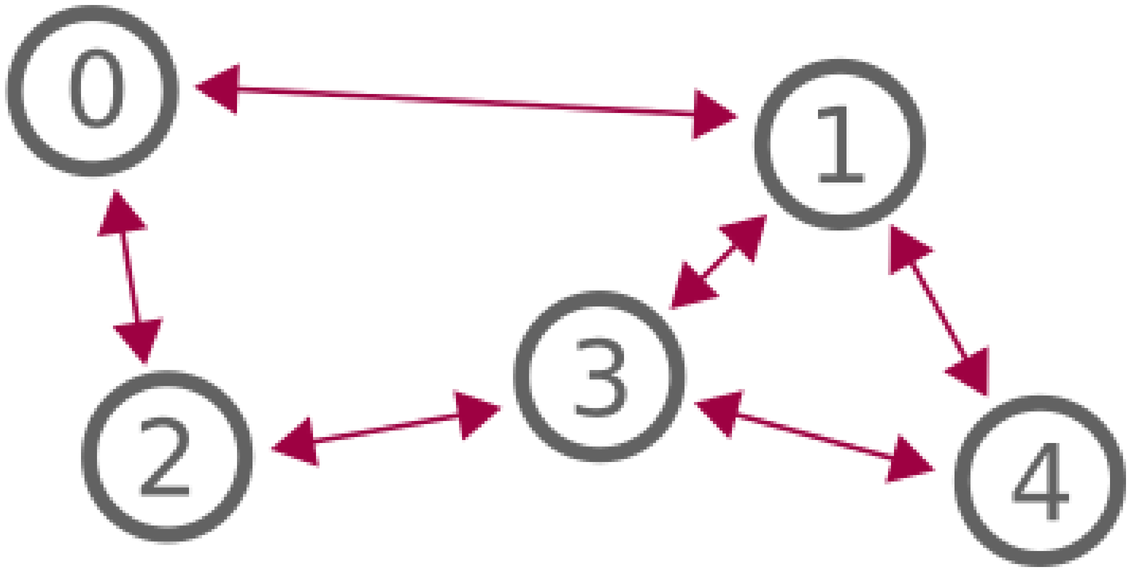
- Problems where data from all the nodes need to be collated
  - Information note just exchanged between nodes but needs to be combined
  - Might form part of an otherwise peer-to-peer architecture – see hybrid architectures

Master/Slave

- Types of communications to be used –
  - A Master/Slave architecture will usually rely heavily on collective communications
    - This will spread some of the communication load away from the master node
  - Scatter – Gather or Scatter – Reduce the usual communication methods
  - Can use non-blocking point to point if you wish to respond to individual slave nodes as they complete
    - Send new data to a node without waiting for all the nodes to complete

Peer to Peer

- What is a Peer to Peer architecture?
  - Every process has equal precedent and communicates directly with the other process (or, more usually, a subset of them)



Peer to Peer

- Advantages
  - Very good for problems that are strongly coupled, especially if each node needs to only communicate with a subset of neighbours
  - Good scalability as the amount of communication into a particular node will only be a weak function of the network size if the number of neighbours is system size independent
    - Often the case in domain decomposition problems

- Disadvantages
  - Often harder to code as no node is in charge of the system
  - Typically, no node will know the entire solution
    - Need to post-process results from different nodes

Peer to Peer

- Types of communications to be used –
  - Will mostly be reliant on non-blocking point to point communications - MPI_Isend and MPI_Irecv
  - Most appropriate choice if nodes only communicate with a subset of the other nodes and these sets of communications are fully interlinked
  - May sometimes need to communicate data between all the nodes (e.g. obtaining a single timestep when using a dynamic timestep)
    - MPI_Allgather or MPI_Allreduce most appropriate

Hybrid Architectures

- It is not required that a program stick religiously to single communication architecture
- An example of where a hybrid architecture is appropriate might be domain decomposition with dynamic load balancing
  - The main calculation loop is most efficiently done using peer-to-peer communications as each process only needs to communicate with the processes responsible for the neighbouring domains
  - For dynamic load balancing each processes needs to calculate their own computational and communication load, but this information needs to be collated at a single process in order for a new domain distribution to be calculated and distributed – a master slave type communication arrangement