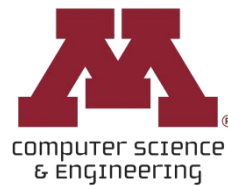


CSCI 5451: Introduction to Parallel Computing

Lecture 5: Tasks to Processors



UNIVERSITY OF MINNESOTA
Driven to Discover®

Announcement (09/17)

❑ NVCC Questions

- Who has run the test code on the cuda cluster?
- If so, have you encountered any errors?
- If you have, what errors have you encountered & when did you enroll in the course?

❑ HW1

- Released next week
- Will cover OpenMP

❑ Project Deadlines

- To discuss in detail next week, as well
- Start thinking about groups



Lecture Overview

How do we efficiently map from our task dependency graphs onto real processors?



Lecture Overview

❑ Communication

- Message Passing
 - ✓ Routing Techniques
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

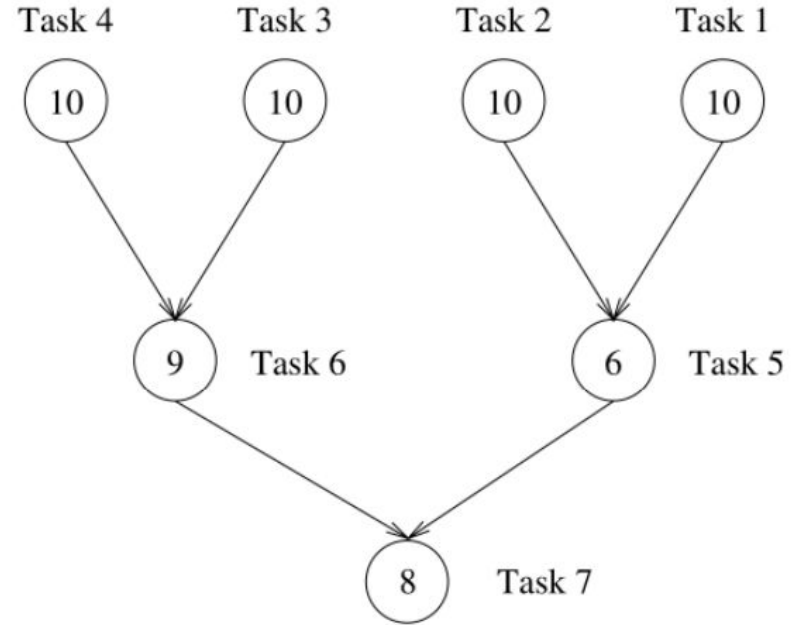
❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



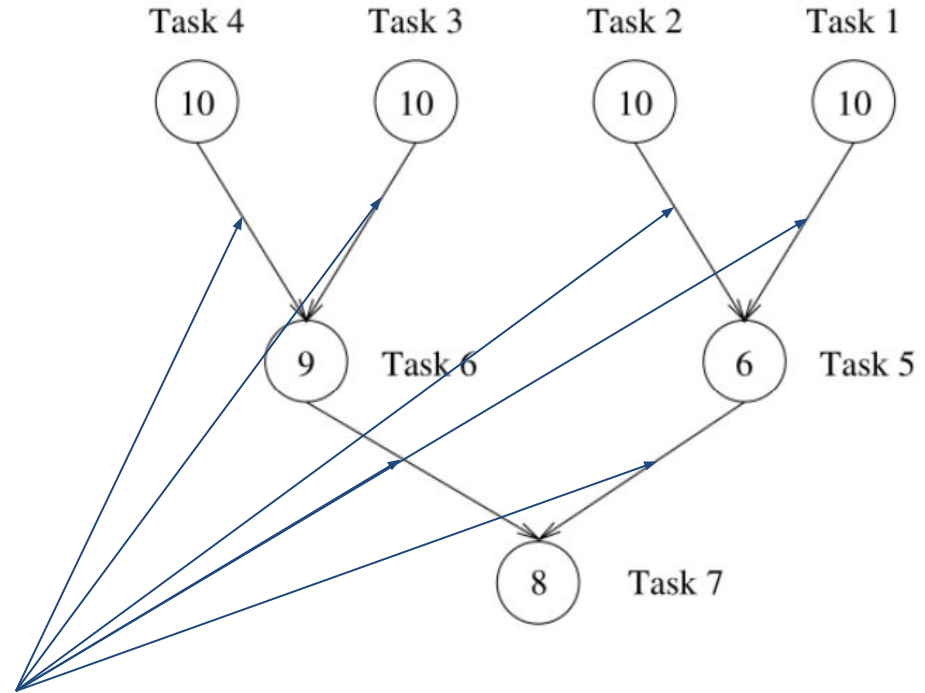
Real Life Task Execution

- ❑ In our last lecture we assumed that task overheads could be ignored for sake of simplicity
- ❑ This is not a reasonable assumption in practical settings
- ❑ We must further understand how tasks will execute in a real world environment



Real Life Task Execution

- ❑ In our last lecture we assumed that task overheads could be ignored for sake of simplicity
- ❑ This is not a reasonable assumption in practical settings
- ❑ We must further understand how tasks will execute in a real world environment



Interactions matter



Lecture Overview

❑ Communication

- **Message Passing**

- ✓ Routing Techniques
- ✓ Minimizing Communication Costs
- ✓ Routing Mechanisms

❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



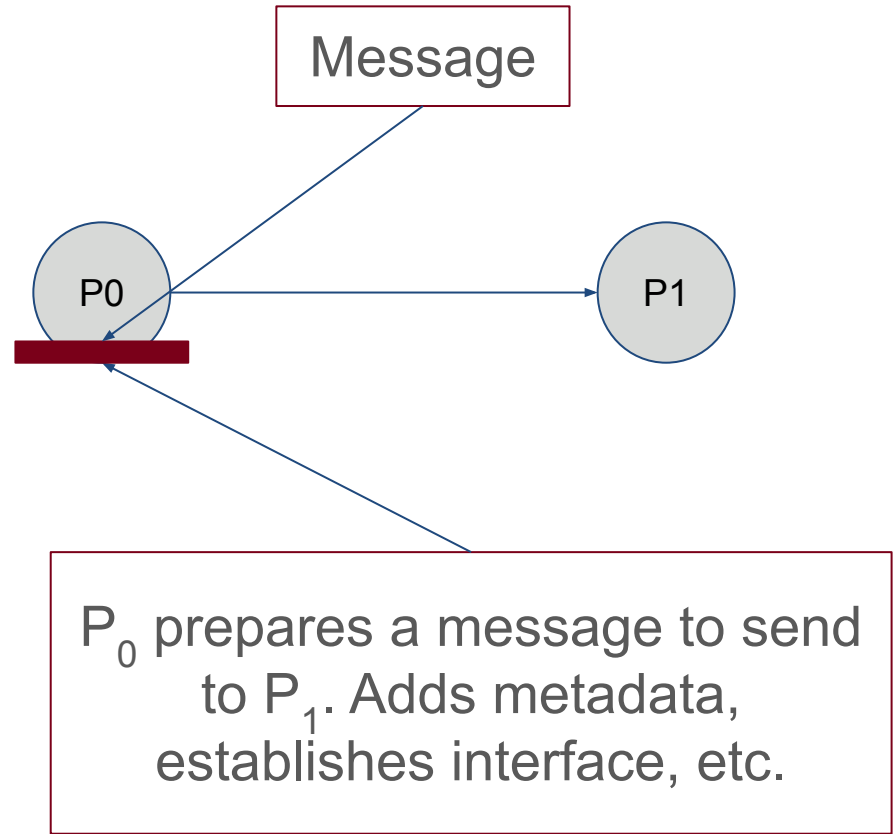
Communication Basics

- ❑ Startup Time (t_s): The time required to prepare a message to be sent along to the next processing element
- ❑ Per-hop time (t_h): The time it takes for the header of a message to go from one node to the next.
- ❑ Per-word transfer time (t_w): Time it takes to put a single word on the link. If the bandwidth of a given channel link is r words per second, then $t_w = 1/r$



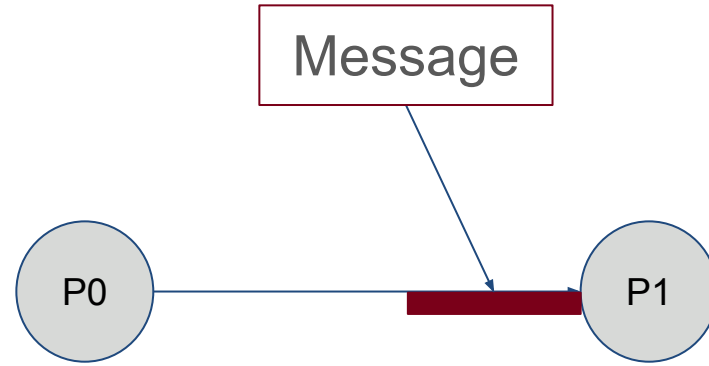
Communication Basics

- ❑ Startup Time (t_s): The time required to prepare a message to be sent along to the next processing element
- ❑ Per-hop time (t_h): The time it takes for the header of a message to go from one node to the next.
- ❑ Per-word transfer time (t_w): Time it takes to put a single word on the link. If the bandwidth of a given channel link is r words per second, then $t_w = 1/r$



Communication Basics

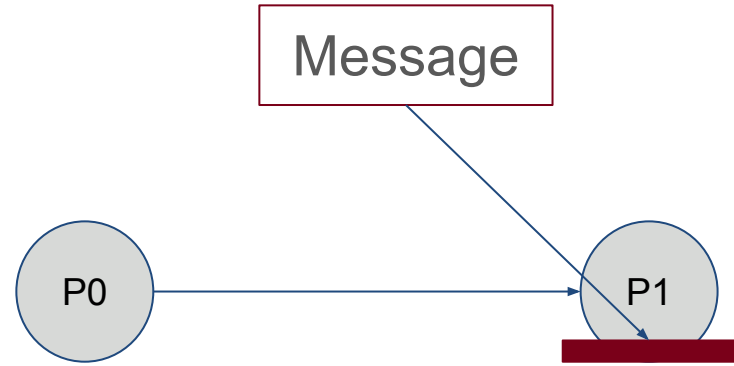
- ❑ Startup Time (t_s): The time required to prepare a message to be sent along to the next processing element
- ❑ Per-hop time (t_h): The time it takes for the header of a message to go from one node to the next.
- ❑ Per-word transfer time (t_w): Time it takes to put a single word on the link. If the bandwidth of a given channel link is r words per second, then $t_w = 1/r$



P_0 sends the message to P_1 . It takes t_h seconds for the first bit to reach P_1 .

Communication Basics

- ❑ Startup Time (t_s): The time required to prepare a message to be sent along to the next processing element
- ❑ Per-hop time (t_h): The time it takes for the header of a message to go from one node to the next.
- ❑ Per-word transfer time (t_w): Time it takes to put a single word on the link. If the bandwidth of a given channel link is r words per second, then $t_w = 1/r$



It takes t_w seconds for the remainder of the message to get to P_1 .

Lecture Overview

❑ Communication

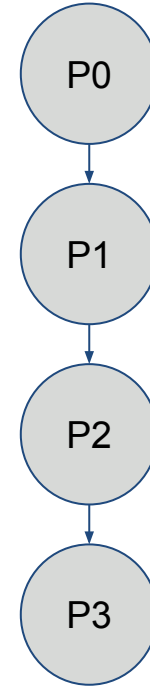
- **Message Passing**
 - ✓ **Routing Techniques**
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



How can we pass messages between nodes when there is no direct connection (or the direct connection is occupied)?

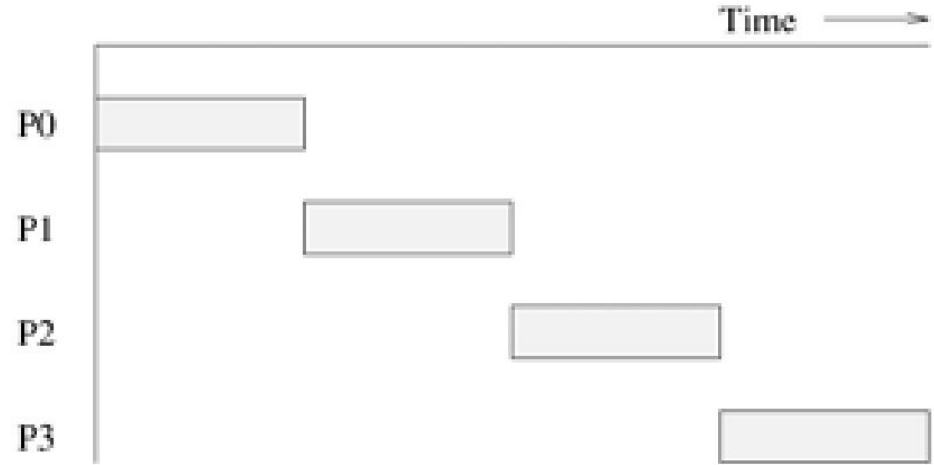


Want to pass a message from P_0 to P_3



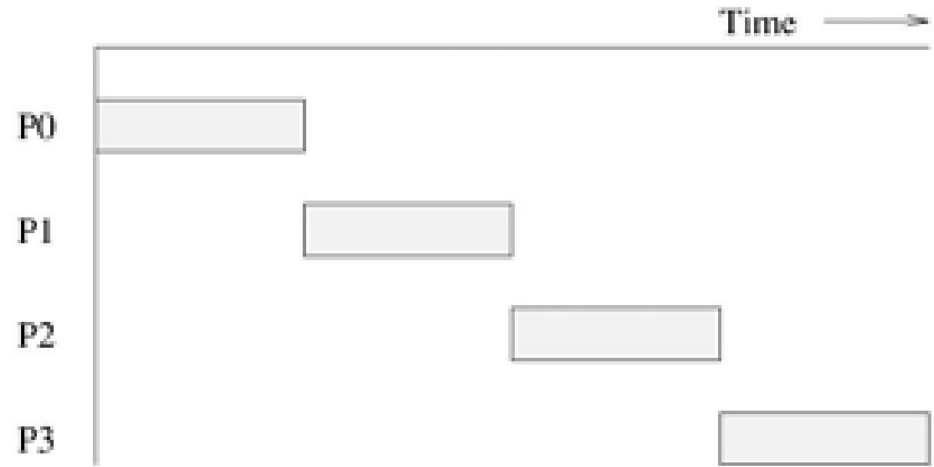
Store-and-Forward Routing

- ❑ Each node waits to pass along a message until it has received and stored the entire thing
- ❑ Suppose we have a message of size m and there are k links we wish to traverse. How long in terms of t_s , t_h , t_w will it take to transmit this message?



Store-and-Forward Routing

- ❑ Each node waits to pass along a message until it has received and stored the entire thing
- ❑ Suppose we have a message of size m and there are k links we wish to traverse. How long in terms of t_s , t_h , t_w will it take to transmit this message?

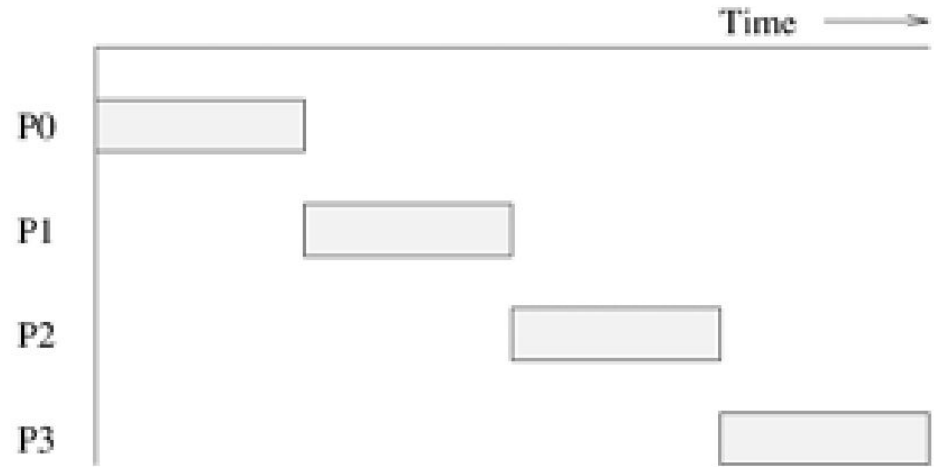


$$t_{comm} = t_s + (mt_w + t_h)k$$



Store-and-Forward Routing

- ❑ Each node waits to pass along a message until it has received and stored the entire thing
- ❑ Suppose we have a message of size m and there are k links we wish to traverse. How long in terms of t_s, t_h, t_w will it take to transmit this message?



$$t_{comm} = t_s + mkt_w$$

Usually t_h is very small & can be ignored



What is a potential issue with Store-and-Forward Routing?



What is a potential issue with Store-and-Forward Routing?

We have to wait for entire messages to be received before we can pass them along. Poor, especially in cases where we expect larger messages.



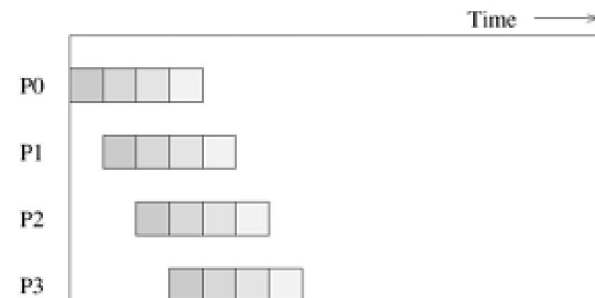
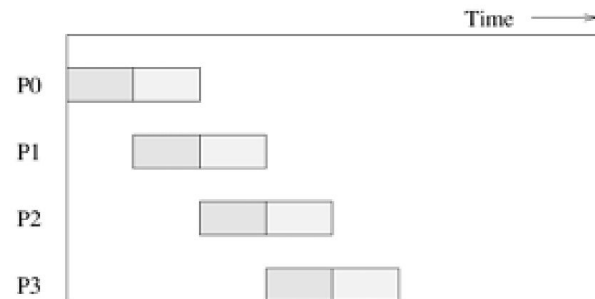
Packing Routing

Method

- Split the message into many smaller pieces
- Re-assemble at destination

Advantages

- Can take different paths
- Less overhead from packet loss
- Better error correction



Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?



Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?

$$t_{\text{comm}} = t_s + t_{w1}m + t_hk + t_{w2}(r + s) + \left(\frac{m}{r} - 1\right)t_{w2}(r + s)$$



Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?

$$t_{\text{comm}} = \boxed{t_s} + \boxed{t_{w1}m} + t_h k + t_{w2}(r + s) + \left(\frac{m}{r} - 1\right) t_{w2}(r + s)$$

Time to prepare message

Time to packetize message

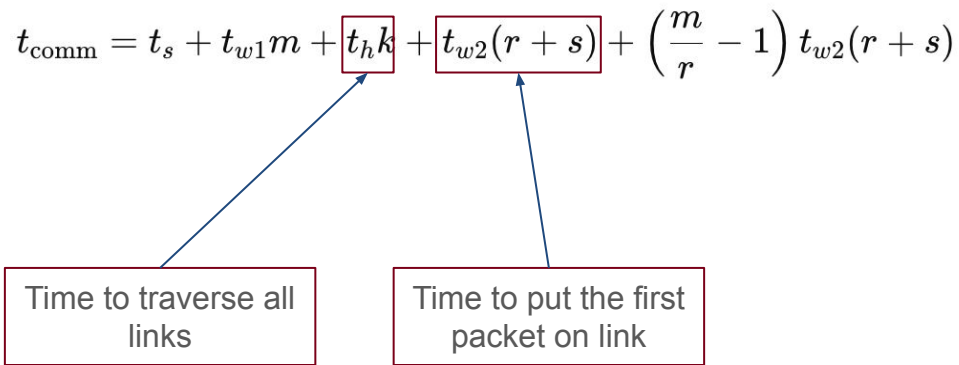


Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?

$$t_{\text{comm}} = t_s + t_{w1}m + \boxed{t_h k} + \boxed{t_{w2}(r + s)} + \left(\frac{m}{r} - 1\right) t_{w2}(r + s)$$


Time to traverse all links

Time to put the first packet on link



Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?

$$t_{\text{comm}} = t_s + t_{w1}m + t_hk + t_{w2}(r + s) + \left(\frac{m}{r} - 1\right)t_{w2}(r + s)$$

Time for all remaining packets to be put on link



Packing Routing

□ Assume

- Packets take the same route (constant # of hops)
- Packetizing the message at the first node takes mt_{w1}
- We traverse k hops
- Per-word transfer times are t_{w2}
- Each packet has message size of r and additional information s (for a total size of $r + s$)

□ How long in terms of $t_s, t_h, t_{w1}, t_{w2}, r, s, k$ will it take to transmit this message?

$$\begin{aligned} t_{\text{comm}} &= t_s + t_{w1}m + t_hk + t_{w2}(r + s) + \left(\frac{m}{r} - 1\right) t_{w2}(r + s) \\ &= t_s + t_{w1}m + t_hk + t_{w2}m + t_{w2}\frac{s}{r}m \\ &= t_s + t_hk + t_wm, \end{aligned}$$

where

$$t_w = t_{w1} + t_{w2} \left(1 + \frac{s}{r}\right).$$



Packing Routing

Store + Forward

$$t_{comm} = t_s + t_h k + m t_w k$$

Packet

$$t_{comm} = t_s + t_h k + t_{w1} m + t_{w2} m(1 + s/r)$$

When will Packet routing be faster
than Store & Forward Routing?



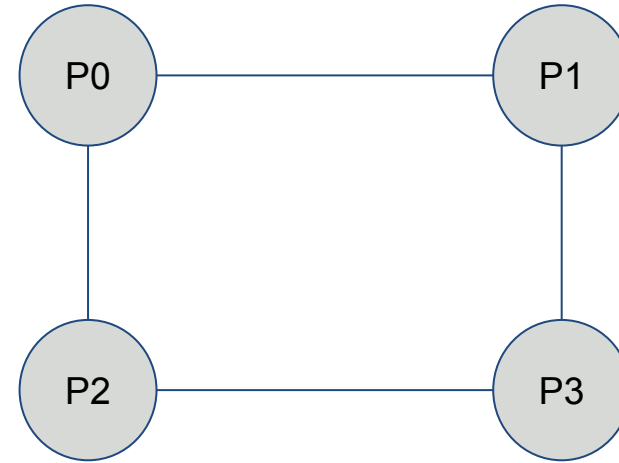
Cut Through Routing

❑ Method

- Allocate full path between nodes beforehand
- Eliminate time spent at each node, just continuously pass bits onward
- Message is broken into Flow Control Digits (Flits)

❑ Advantages

- Extra packet overhead eliminated
- In-sequence delivery



$$t_{comm} = t_s + kt_h + t_w m$$

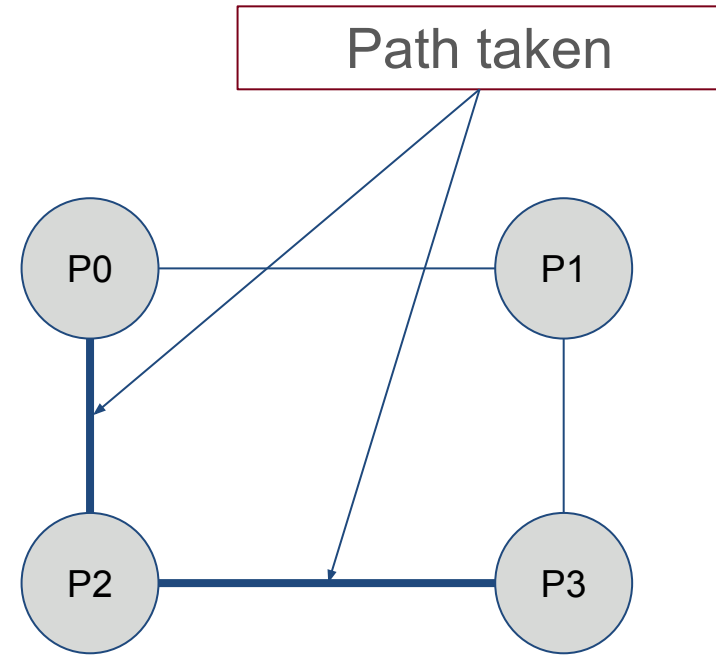
Cut Through Routing

❑ Method

- Allocate full path between nodes beforehand
- Eliminate time spent at each node, just continuously pass bits onward
- Message is broken into Flow Control Digits (Flits)

❑ Advantages

- Extra packet overhead eliminated
- In-sequence delivery



P0 wants to send a message to P3

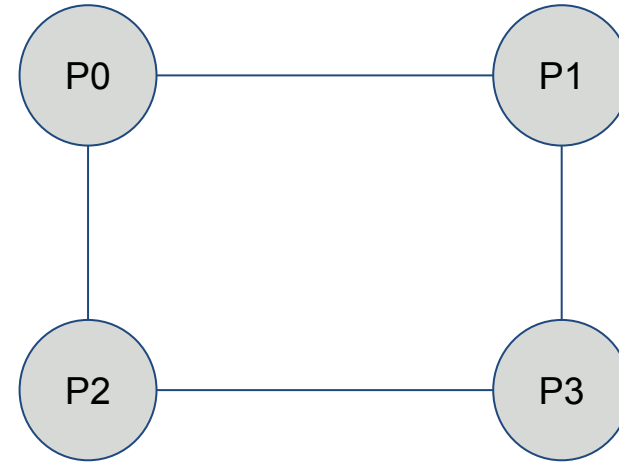
Cut Through Routing

❑ Method

- Allocate full path between nodes beforehand
- Eliminate time spent at each node, just continuously pass bits onward
- Message is broken into Flow Control Digits (Flits)

❑ Advantages

- Extra packet overhead eliminated
- In-sequence delivery



What if P0 wants to send a message to P3 while P1 wants to send a message to P2?

Cut Through Routing

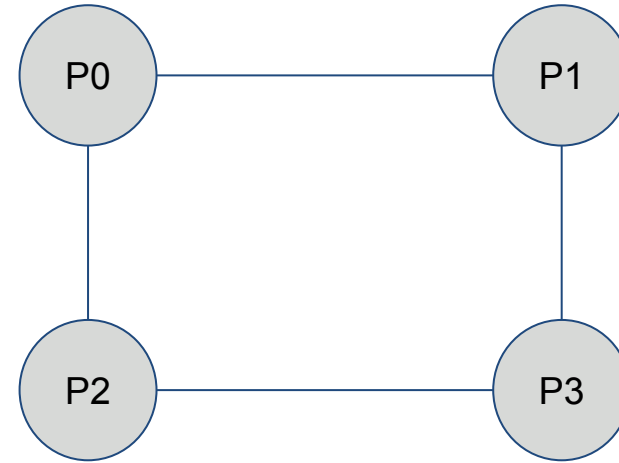
❑ Method

- Allocate full path between nodes beforehand
- Eliminate time spent at each node, just continuously pass bits onward
- Message is broken into Flow Control Digits (Flits)

❑ Advantages

- Extra packet overhead eliminated
- In-sequence delivery

Deadlock. Multiple paths overlap.



What if P0 wants to send a message to P3 while P1 wants to send a message to P2?



Lecture Overview

❑ Communication

- **Message Passing**

- ✓ Routing Techniques
- ✓ **Minimizing Communication Costs**
- ✓ Routing Mechanisms

❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



In general, what can we do to minimize communication costs?



Minimizing Communication Costs

- ❑ Communicate in bulk
- ❑ Minimize volume of data
- ❑ Minimize distance of data transfer

$$t_{comm} = t_s + kt_h + t_w m$$

We consider the Cut-Through routing equation above, but the thinking in the following slides holds for Packet & Store-Forward Routing as well



Minimizing Communication Costs

- ☐ Communicate in bulk
- ☐ Minimize volume of data
- ☐ Minimize distance of data transfer

$$t_{comm} = t_s + kt_h + t_w m$$

Group messages together. Each separate message incurs another startup cost.



Minimizing Communication Costs

- ☐ Communicate in bulk
- ☐ Minimize volume of data
- ☐ Minimize distance of data transfer

$$t_{comm} = t_s + kt_h + t_w m$$

Minimizing message size lowers the above term.



Minimizing Communication Costs

- ❑ Communicate in bulk
- ❑ Minimize volume of data
- ❑ Minimize distance of data transfer

$$t_{comm} = t_s + kt_h + t_w m$$

Use more intelligent routing & better process-processor mapping (see later slides in this lecture). Results in fewer hops.



Lecture Overview

❑ Communication

- **Message Passing**

- ✓ Routing Techniques
- ✓ Minimizing Communication Costs
- ✓ **Routing Mechanisms**

❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



What does smarter routing look like?



Routing Mechanism (Classification)

□ Minimal vs. Non-Minimal

- Minimal: Always select the shortest path to communicate between source & destination
- Non-minimal: Longer routes can be taken to avoid congestion

□ Deterministic vs. dynamic

- Deterministic: Routes are known between each process in advance
- Dynamic: Routes are determined *on-the-fly* during program execution



Routing Mechanism (Examples)

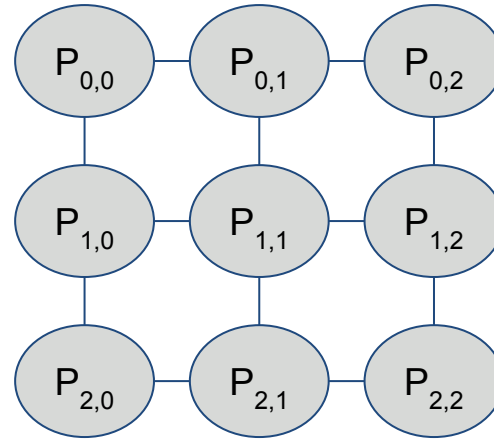
$$P_{0,0} \rightarrow P_{2,2}$$

□ XY-Routing

- Processor topology is a 2-d grid
- First pass message to appropriate y location
- Then to appropriate x location

□ E-cube routing

- Processor topology is hypercube
- Processors are labelled in binary
- Pass along each bit-wise dimension, from least significant to most significant



Routing Mechanism (Examples)

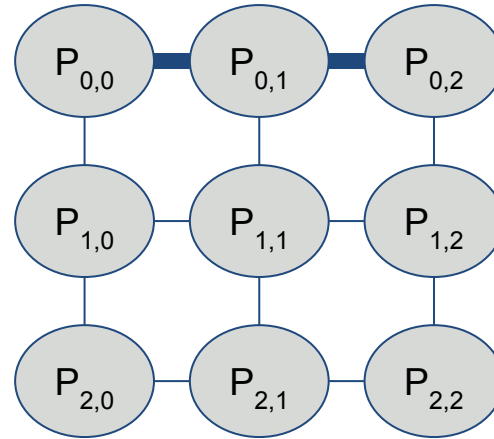
$$P_{0,0} \rightarrow P_{2,2}$$

□ XY-Routing

- Processor topology is a 2-d grid
- First pass message to appropriate y location
- Then to appropriate x location

□ E-cube routing

- Processor topology is hypercube
- Processors are labelled in binary
- Pass along each bit-wise dimension, from least significant to most significant



Routing Mechanism (Examples)

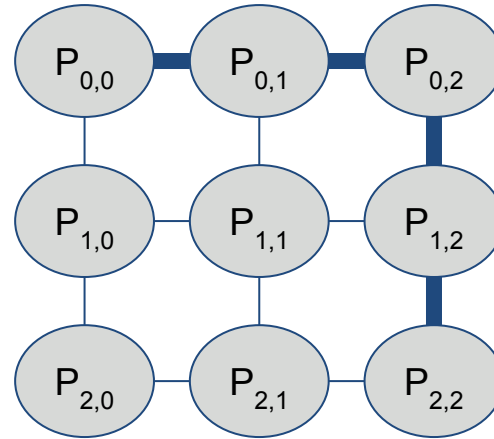
$$P_{0,0} \rightarrow P_{2,2}$$

□ XY-Routing

- Processor topology is a 2-d grid
- First pass message to appropriate y location
- Then to appropriate x location

□ E-cube routing

- Processor topology is hypercube
- Processors are labelled in binary
- Pass along each bit-wise dimension, from least significant to most significant



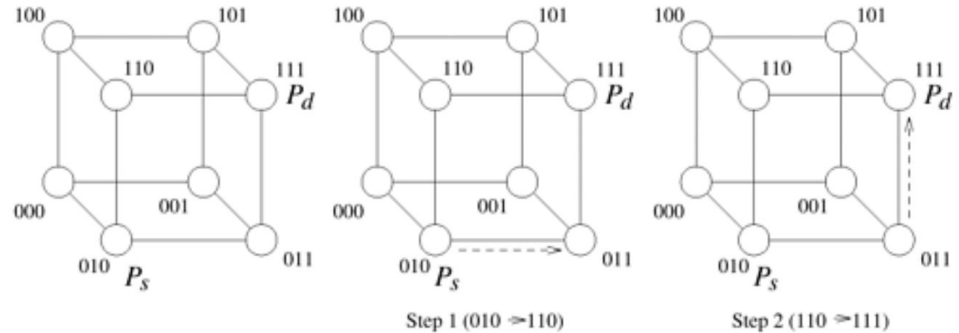
Routing Mechanism (Examples)

□ XY-Routing

- Processor topology is a 2-d grid
- First pass message to appropriate y location
- Then to appropriate x location

□ E-cube routing

- Processor topology is hypercube
- Processors are labelled in binary
- Pass along each bit-wise dimension, from least significant to most significant



Lecture Overview

❑ Communication

- Message Passing
 - ✓ Routing Techniques
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

❑ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- Processes to processors (2.7)



How do we go from tasks to physical processors?



Tasks vs. Processes vs. Processors

□ In general, creating a well running parallel program requires performing the following in order:

1. Define the tasks
2. Define task interactions + TDG
3. Determine which processes will work on which tasks
4. Make sure that each process is properly mapped onto hardware processors

We discussed this in the previous lecture



Tasks vs. Processes vs. Processors

□ In general, creating a well running parallel program requires performing the following in order:

1. Define the tasks
2. Define task interactions + TDG
3. Determine which processes will work on which tasks
4. Make sure that each process is properly mapped onto hardware processors

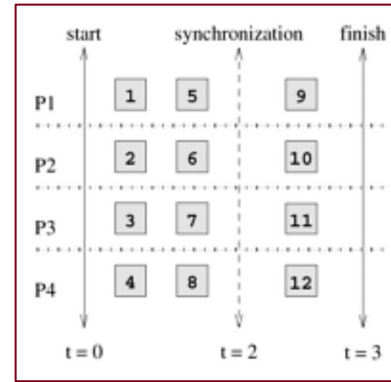
Processes are what we will eventually map onto hardware. Each process can be assigned many tasks. Processes are still computational abstractions, they are not yet bound to physical hardware.



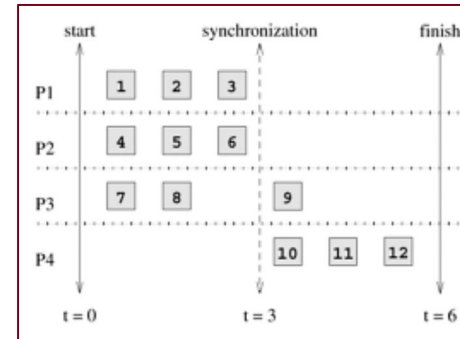
Tasks vs. Processes vs. Processors

❑ In general, creating a well running parallel program requires performing the following in order:

1. Define the tasks
2. Define task interactions + TDG
3. Determine which processes will work on which tasks
4. Make sure that each process is properly mapped onto hardware processors



Good
mapping of
12 tasks to
4
processes



Bad
mapping of
12 tasks to
4
processes



Lecture Overview

❑ Communication

- Message Passing
 - ✓ Routing Techniques
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

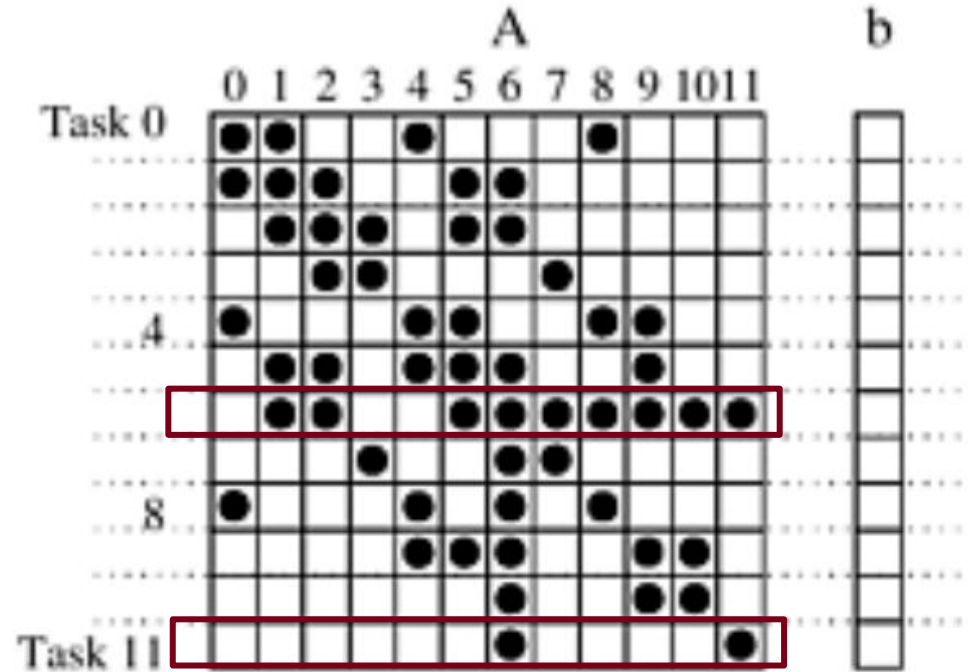
❑ Mapping

- **Tasks to processes**
 - ✓ **Load Balancing**
 - ✓ Interaction Overhead
- Processes to processors (2.7)



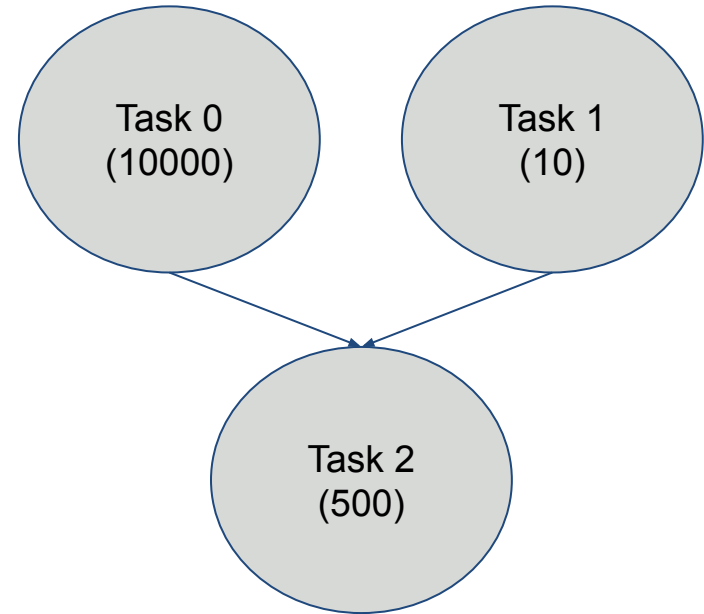
Load Balancing Motivation

- ❑ Consider sparse matrix multiplication at right
- ❑ Specifically, tasks 6 & 11
- ❑ Task 6 has 17 units of work assigned to it
- ❑ Task 11 has 3 units of work
- ❑ If we can run 12 processes concurrently, then we will not be taking full advantage of our hardware



Load Balancing Motivation

- ❑ We want to assign tasks to processes so that no process is idle for very long
- ❑ If we have wildly uneven task work, then we should think about how to recreate our TDG with more concurrency in mind



Static Vs. Dynamic Mapping

❑ Static Mapping:

- We determine exactly what tasks are given to each process before program execution.
- Useful with static task generation & known task sizes

❑ Dynamic Mapping:

- Tasks are distributed to each process during program execution
- Useful with non-uniform tasks, dynamic task generation



Static Vs. Dynamic Mapping

❑ Static Mapping:

- We determine exactly what tasks are given to each process before program execution.
- Useful with static task generation & known task sizes

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

❑ Dynamic Mapping:

- Tasks are distributed to each process during program execution
- Useful with non-uniform tasks, dynamic task generation

$$\text{Task 1: } C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$

$$\text{Task 2: } C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$

$$\text{Task 3: } C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$

$$\text{Task 4: } C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$



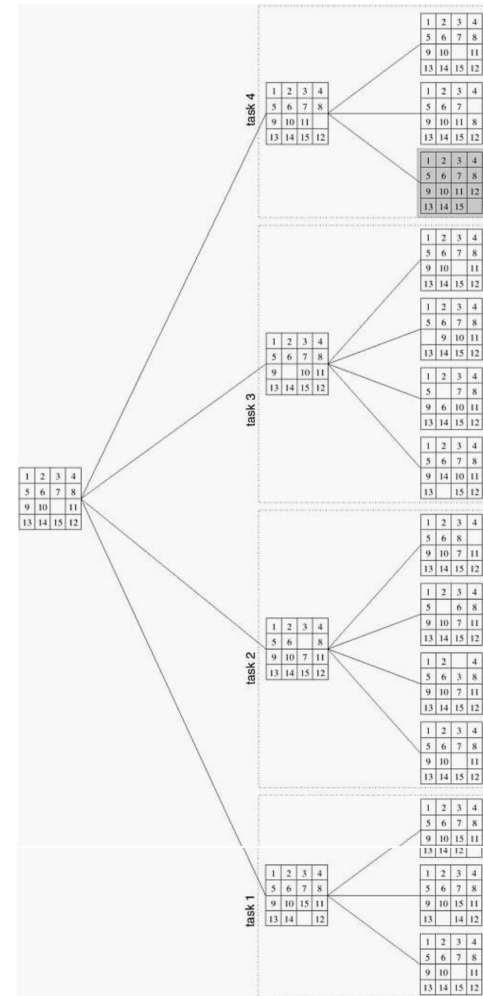
Static Vs. Dynamic Mapping

❑ Static Mapping:

- We determine exactly what tasks are given to each process before program execution.
- Useful with static task generation & known task sizes

❑ Dynamic Mapping:

- Tasks are distributed to each process during program execution
- Useful with non-uniform tasks, dynamic task generation



What are some examples of intelligent Static Mapping Schemes?



Block Distributions

row-wise distribution

P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7

column-wise distribution

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
-------	-------	-------	-------	-------	-------	-------	-------



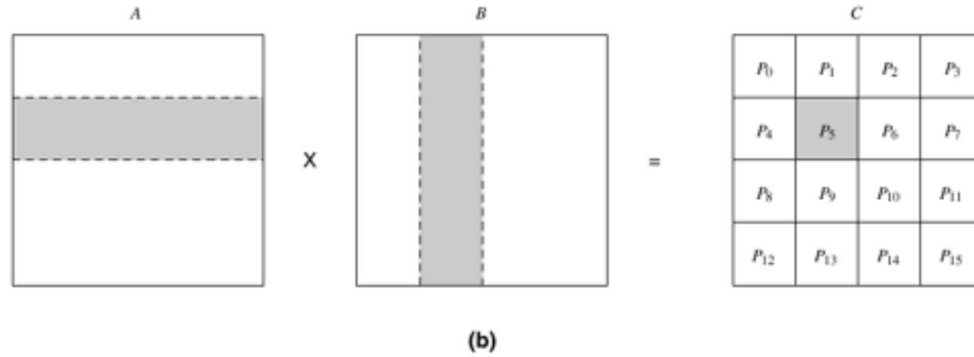
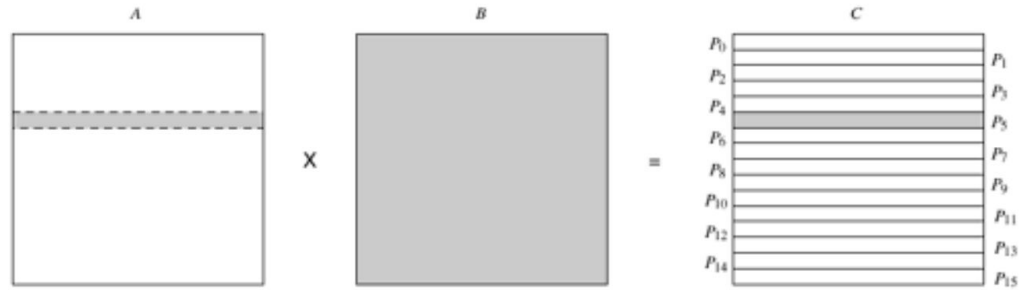
Block Distributions

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}



Block Distributions



When might block distributions not work as well?



When might block distributions not work as well?

Algorithms with unequal work assigned to each block



Block-Cyclic

LU Factorization

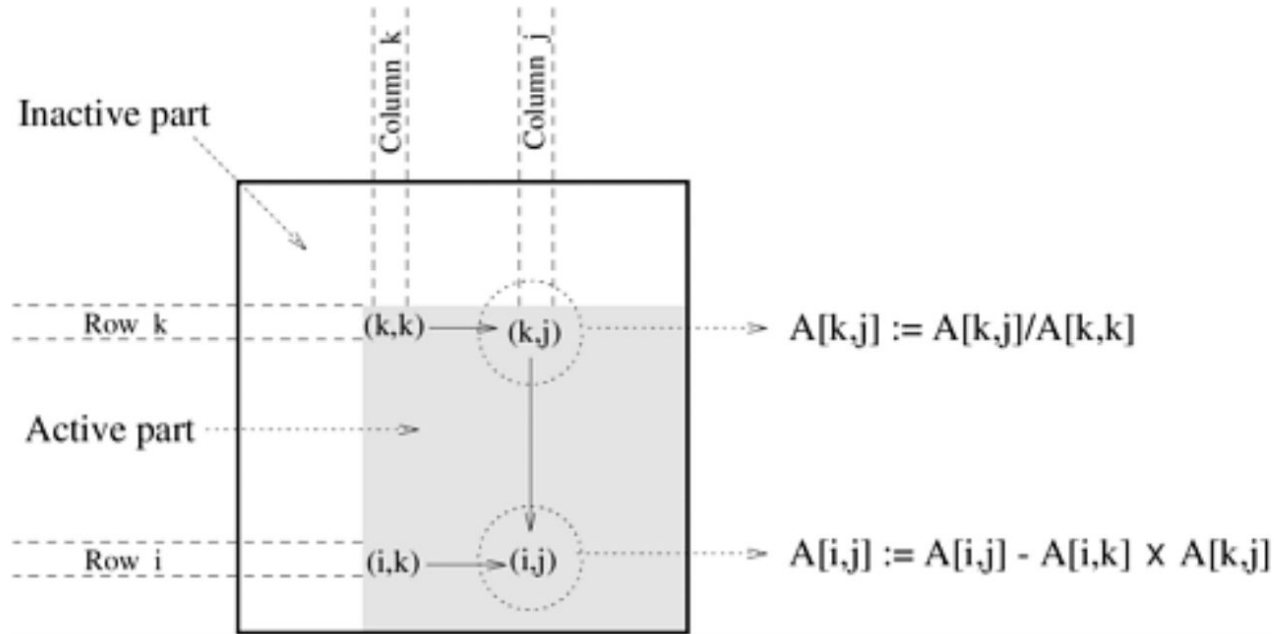
$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1,1} & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \cdot \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ 0 & U_{2,2} & U_{2,3} \\ 0 & 0 & U_{3,3} \end{pmatrix}$$

$$\begin{array}{l|l|l} 1: A_{1,1} \rightarrow L_{1,1}U_{1,1} & 6: A_{2,2} = A_{2,2} - L_{2,1}U_{1,2} & 11: L_{3,2} = A_{3,2}U_{2,2}^{-1} \\ 2: L_{2,1} = A_{2,1}U_{1,1}^{-1} & 7: A_{3,2} = A_{3,2} - L_{3,1}U_{1,2} & 12: U_{2,3} = L_{2,2}^{-1}A_{2,3} \\ 3: L_{3,1} = A_{3,1}U_{1,1}^{-1} & 8: A_{2,3} = A_{2,3} - L_{2,1}U_{1,3} & 13: A_{3,3} = A_{3,3} - L_{3,2}U_{2,3} \\ 4: U_{1,2} = L_{1,1}^{-1}A_{1,2} & 9: A_{3,3} = A_{3,3} - L_{3,1}U_{1,3} & 14: A_{3,3} \rightarrow L_{3,3}U_{3,3} \\ 5: U_{1,3} = L_{1,1}^{-1}A_{1,3} & 10: A_{2,2} \rightarrow L_{2,2}U_{2,2} & \end{array}$$



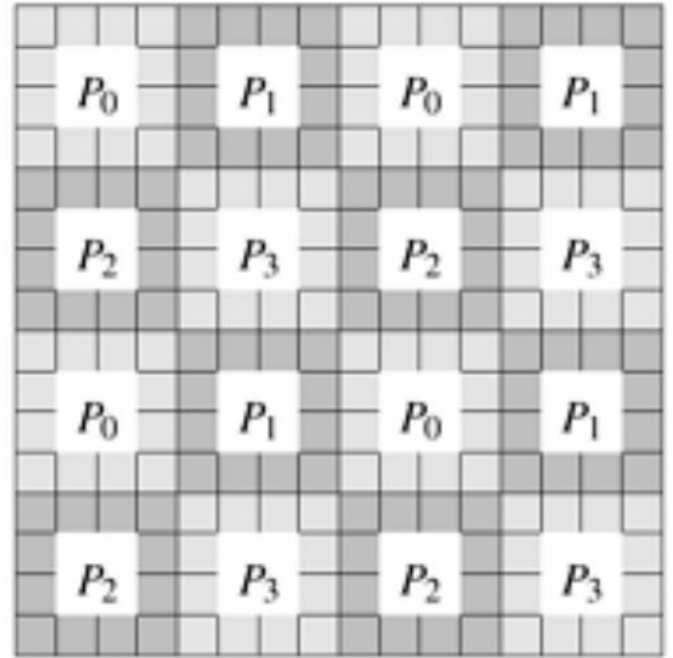
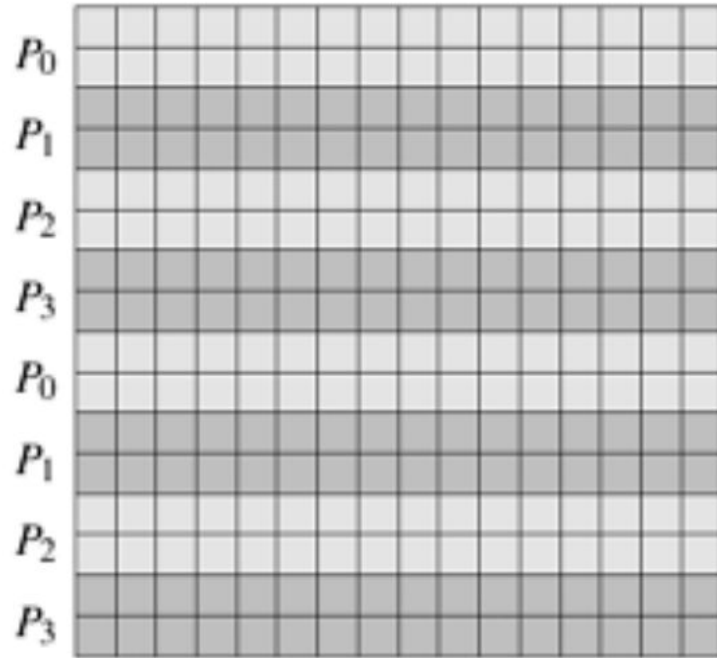
Block-Cyclic

LU Factorization



Block-Cyclic

Better Partitioning



Even more irregular computations (e.g. sparse matrix-vector multiplication) still have issues...

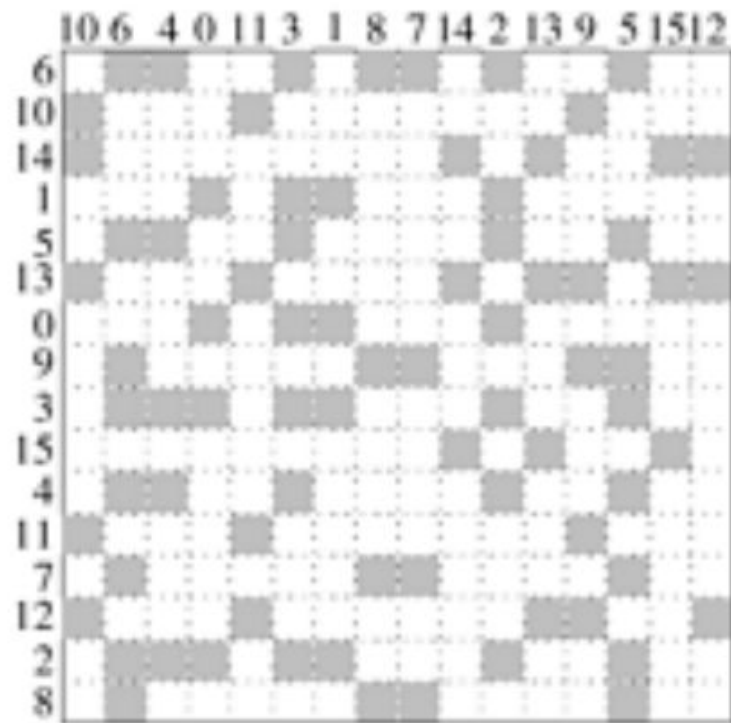


Even more irregular computations (e.g. sparse matrix-vector multiplication) still have issues...

Resolve by randomizing task assignments
to different processes.



Randomized



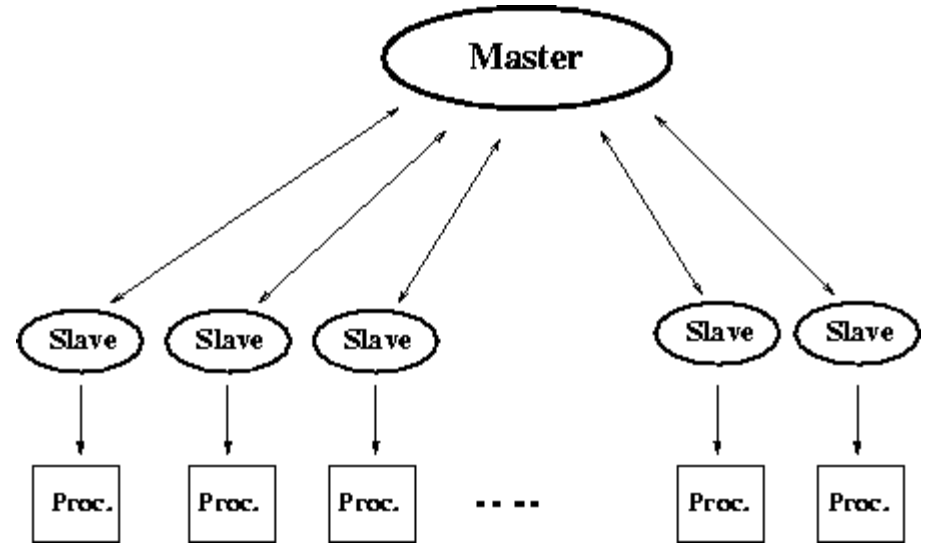
P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

How about Dynamic Mapping Schemes?



Controller-Work Model

- ❑ One process is the *controller* and distributes tasks to a larger number of *worker* processes
- ❑ Useful for non-uniform tasks, in cases where tasks have minimal interactions & when tasks are generated during program execution



Lecture Overview

❑ Communication

- Message Passing
 - ✓ Routing Techniques
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

❑ Mapping

- **Tasks to processes**
 - ✓ Load Balancing
 - ✓ **Interaction Overhead**
- Processes to processors (2.7)



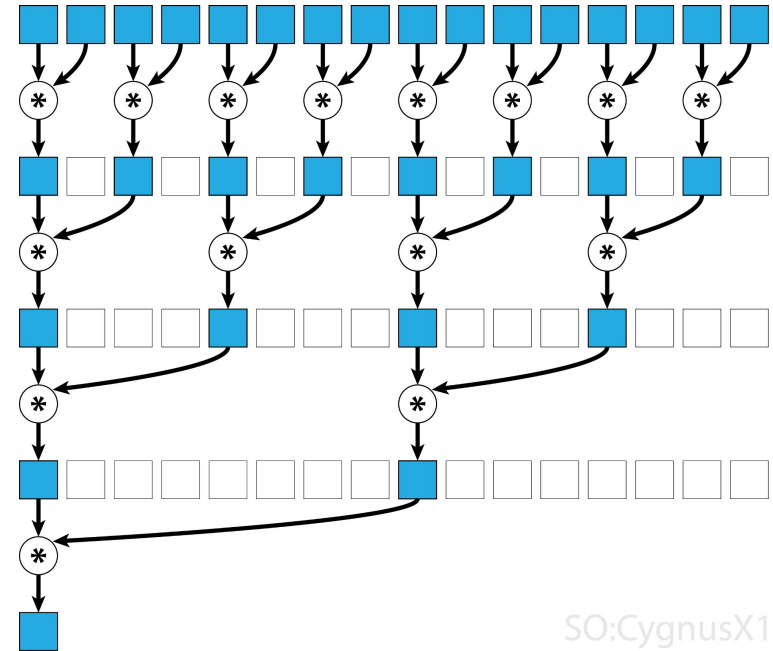
Task Interaction Graph

- ❑ In addition to directed Task Decomposition Graphs, we can construct Task Interaction Graphs
- ❑ These show which processes must communicate with one another during execution
- ❑ Examples
 - Reduction
 - Sparse Matrix-Vector Multiplication



Task Interaction Graph

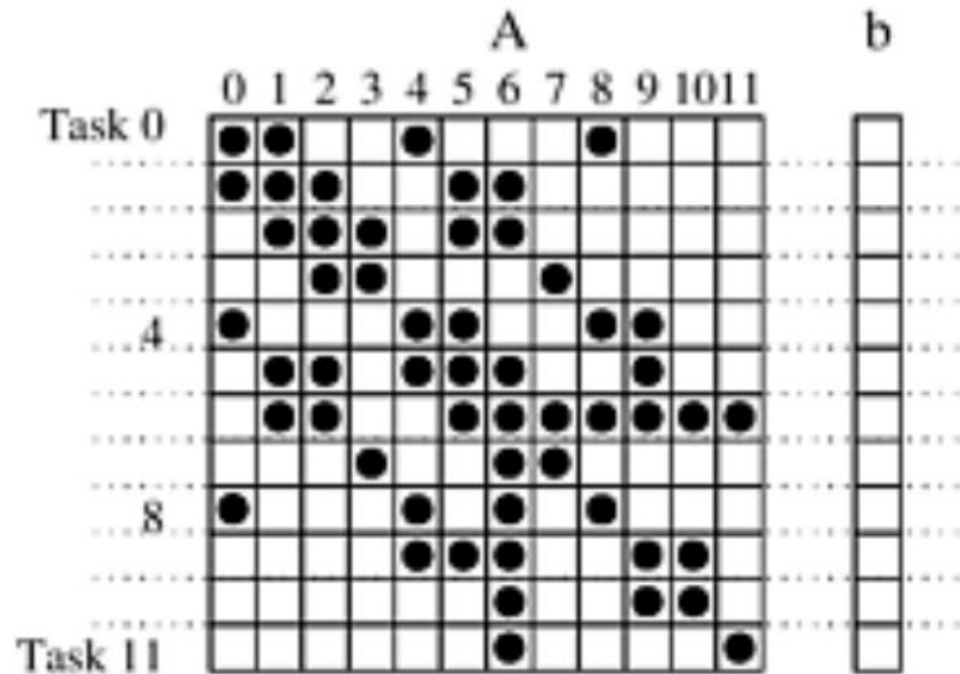
- ❑ In addition to directed Task Decomposition Graphs, we can construct Task Interaction Graphs
- ❑ These show which processes must communicate with one another during execution
- ❑ Examples
 - Reduction
 - Sparse Matrix-Vector Multiplication



Task Interaction Graph

- ❑ In addition to directed Task Decomposition Graphs, we can construct Task Interaction Graphs
- ❑ These show which processes must communicate with one another during execution
- ❑ Examples
 - Reduction
 - Sparse Matrix-Vector Multiplication

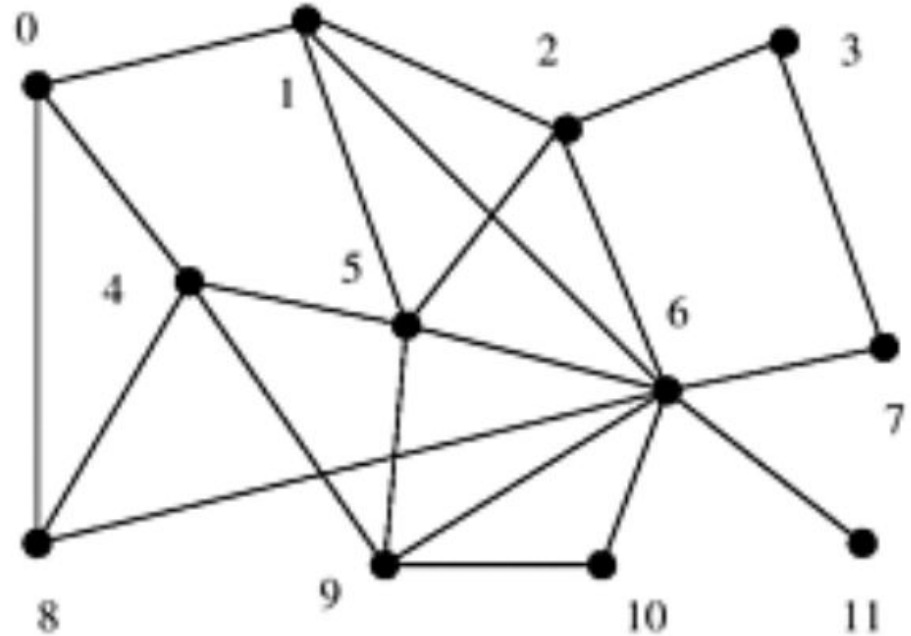
Split matrix A and vector b among 12 tasks



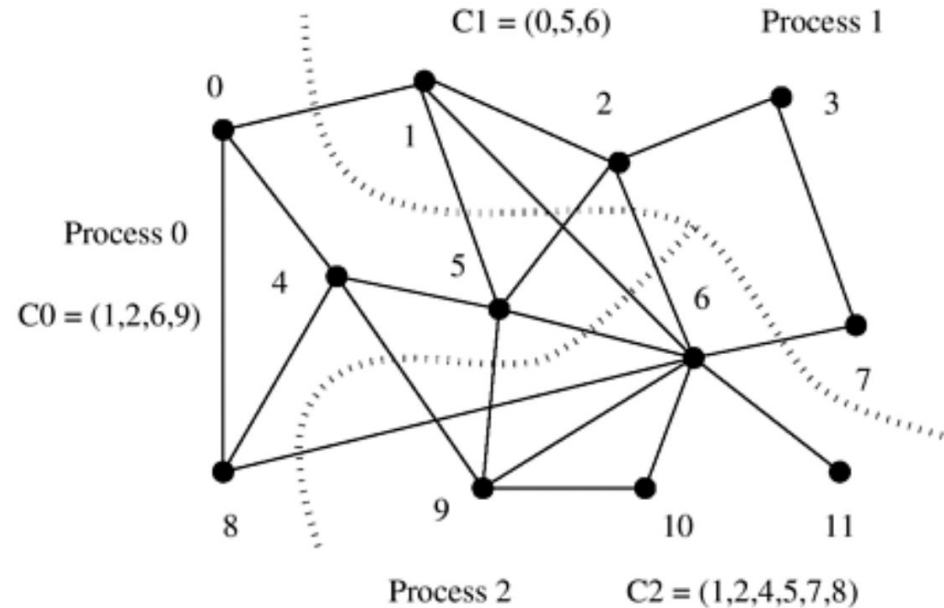
Task Interaction Graph

- ❑ In addition to directed Task Decomposition Graphs, we can construct Task Interaction Graphs
- ❑ These show which processes must communicate with one another during execution
- ❑ Examples
 - Reduction
 - Sparse Matrix-Vector Multiplication

Split matrix A and vector b among 12 tasks



We want to assign
tasks to processes
such that tasks
between processes
communicate very
little



Lecture Ended Here. Remaining slides to be covered in the next lecture.



General Strategies for Minimizing Interaction

- ❑ Min Data Exchange
- ❑ Min Frequency of Interactions (cut graphs properly)
- ❑ Min Contention/Hot Spot
- ❑ Overlap comms + Interactions
- ❑ Replicate Data + Computations
- ❑ Use Optimized Collective Interactions (MPI)
- ❑ Overlap interactions with other interactions



Lecture Overview

□ Communication

- Message Passing
 - ✓ Routing Techniques
 - ✓ Minimizing Communication Costs
 - ✓ Routing Mechanisms

□ Mapping

- Tasks to processes
 - ✓ Load Balancing
 - ✓ Interaction Overhead
- **Processes to processors (2.7)**



Tasks vs. Processes vs. Processors

❑ In general, creating a well running parallel program requires performing the following in order:

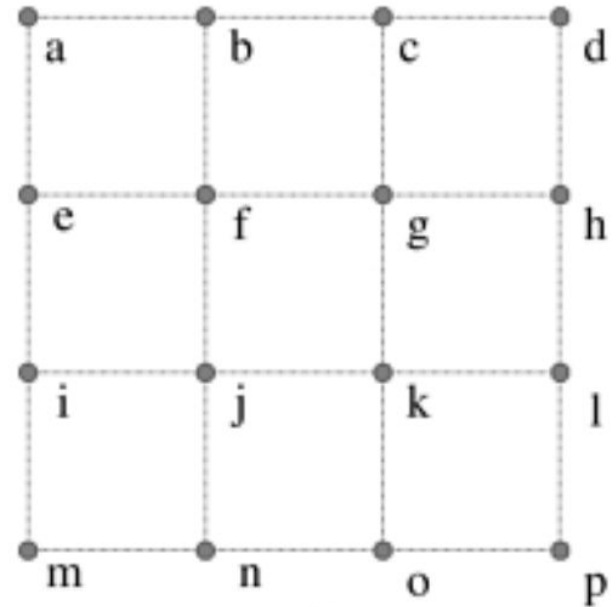
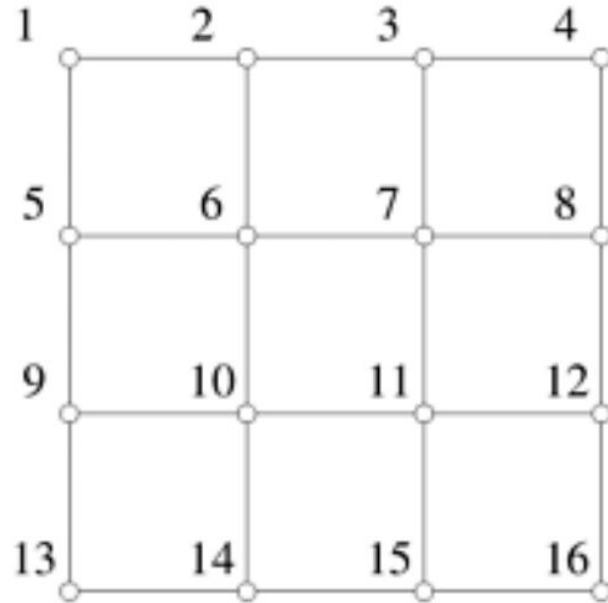
1. Define the tasks
2. Define task interactions + TDG
3. Determine which processes will work on which tasks
4. Make sure that each process is properly mapped onto hardware

We want to ensure that the way we organize our processes is how they are mapped onto the hardware.



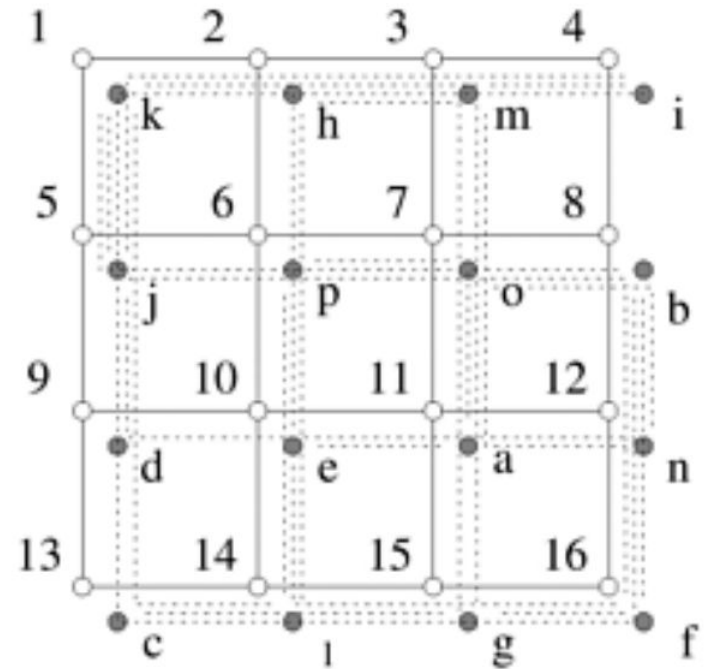
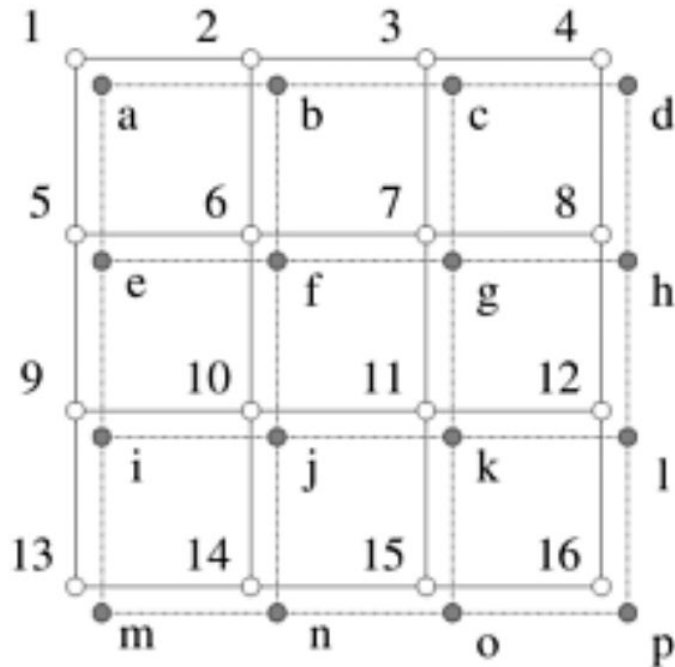
Process - Processor Mappings

Good Mapping



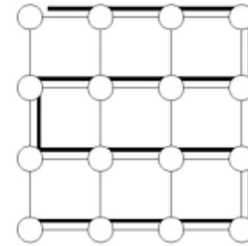
Process - Processor Mappings

Bad Mapping

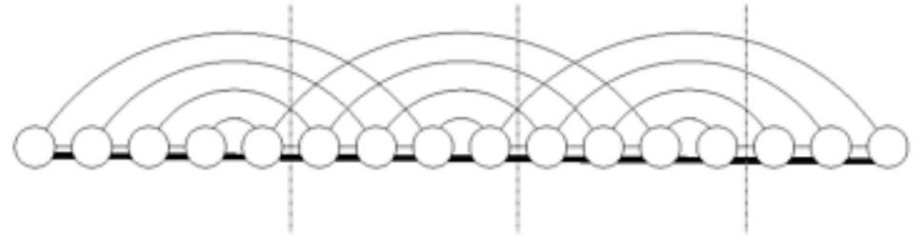


Mapping Different Process-Processor Graphs

- ❑ Sometimes the process interactions we design for our initial algorithm will not match those of the actual hardware
- ❑ When there are fewer links in the hardware than our process setup
 - Congestion (Many processes trying to communicate over the same link)
 - Dilation (Processes may have to communicate over longer distances)



(a) Mapping a linear array into a 2D mesh (congestion 1).



(b) Inverting the mapping – mapping a 2D mesh into a linear array (congestion 5)