Design of Parallel Algorithms

Gregory G. Howes
Department of Physics and Astronomy
University of Iowa

PHYS 5905: Numerical Simulation of Plasmas
Department of Physics and Astronomy
University of Iowa
Spring 2019

THE UNIVERSITY OF IOWA

Thank you

This presentation borrows heavily from information freely available on the web by Ian Foster and Blaise Barney (see references)

Outline

- Basics of Parallel Algorithm Design
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping

- Final Thoughts

- References

Design of Parallel Algorithms

The University of Iowa

- Ensure that you understand fully the problem and/or the serial code that you wish to make parallel

- Identify the program hotspots
  - These are places where most of the computational work is being done
  - Making these sections parallel will lead to the most improvement
  - Profiling can help to determine the hotspots (more on this Wednesday)

- Identify bottlenecks in the program
  - Some sections of the code are disproportionately slow
  - It is often possible to restructure a code to minimize the bottlenecks

- Sometimes it is possible to identify a different computational algorithm that has much better scaling properties

P C A M

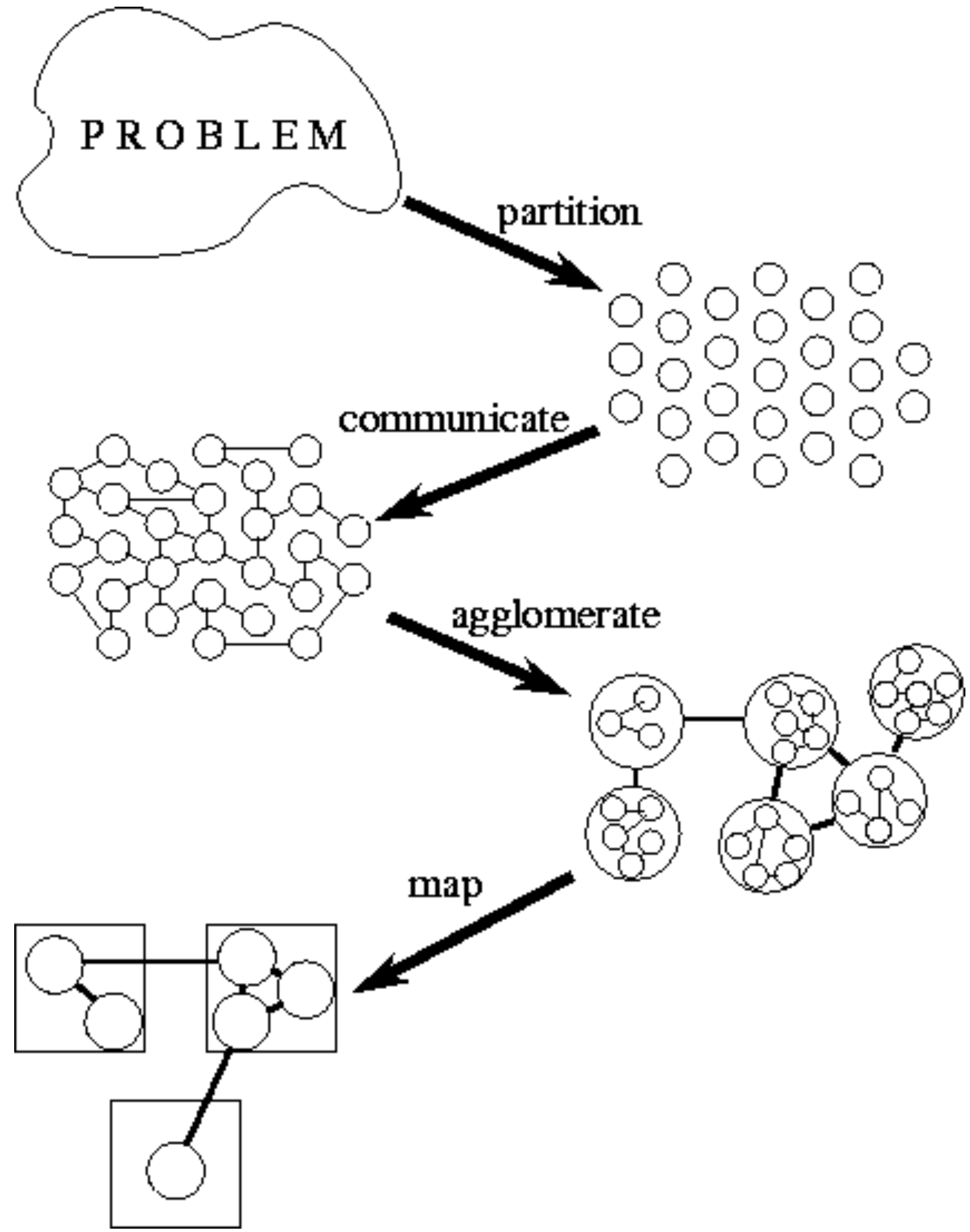
Methodological Approach to Parallel Algorithm Design:

1) Partitioning

2) Communication

3) Agglomeration

4) Mapping

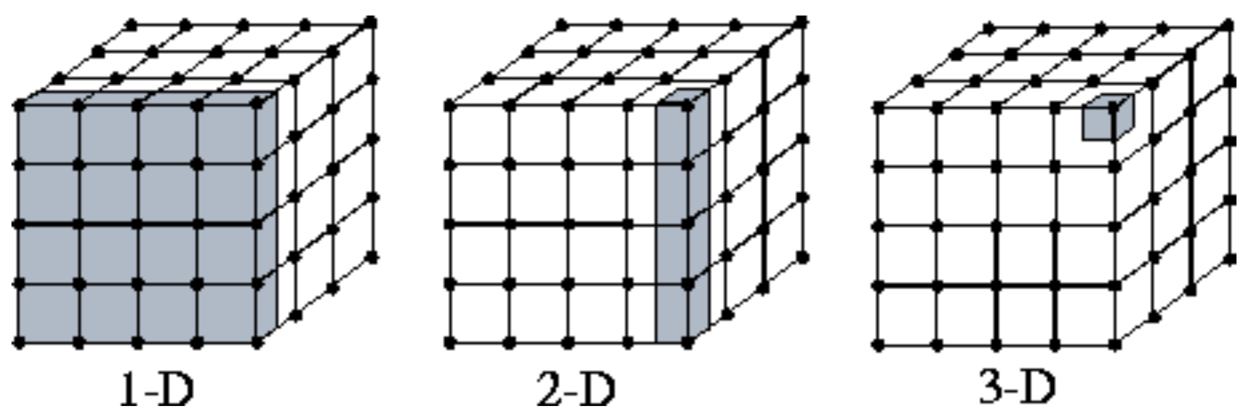


Partitioning

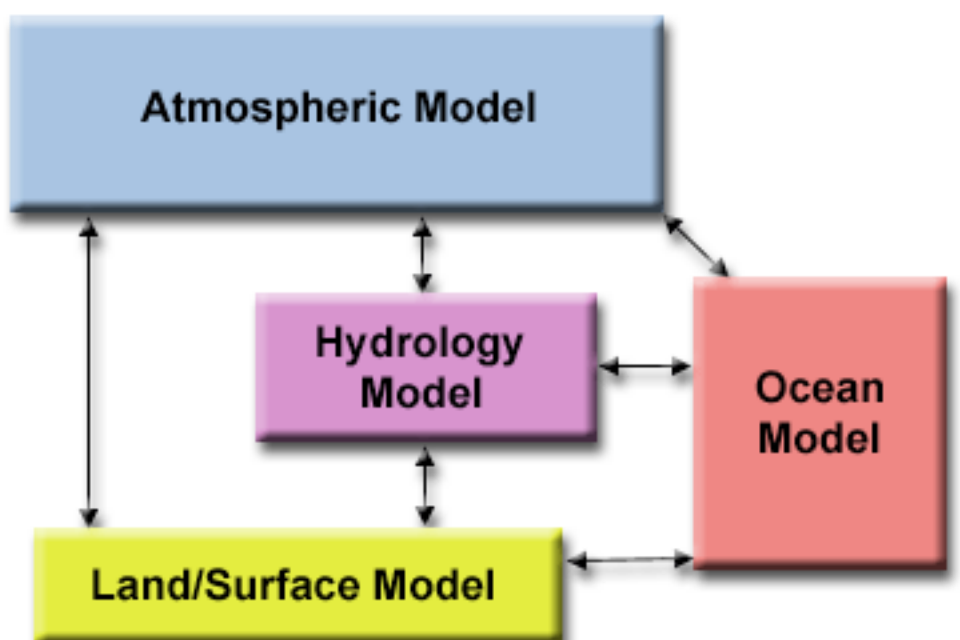
- Split both the computation to be performed and the data into a large number of small tasks (fine-grained)

Two primary ways of decomposing the problem:

- Domain Decomposition



- Functional Decomposition



P C A M

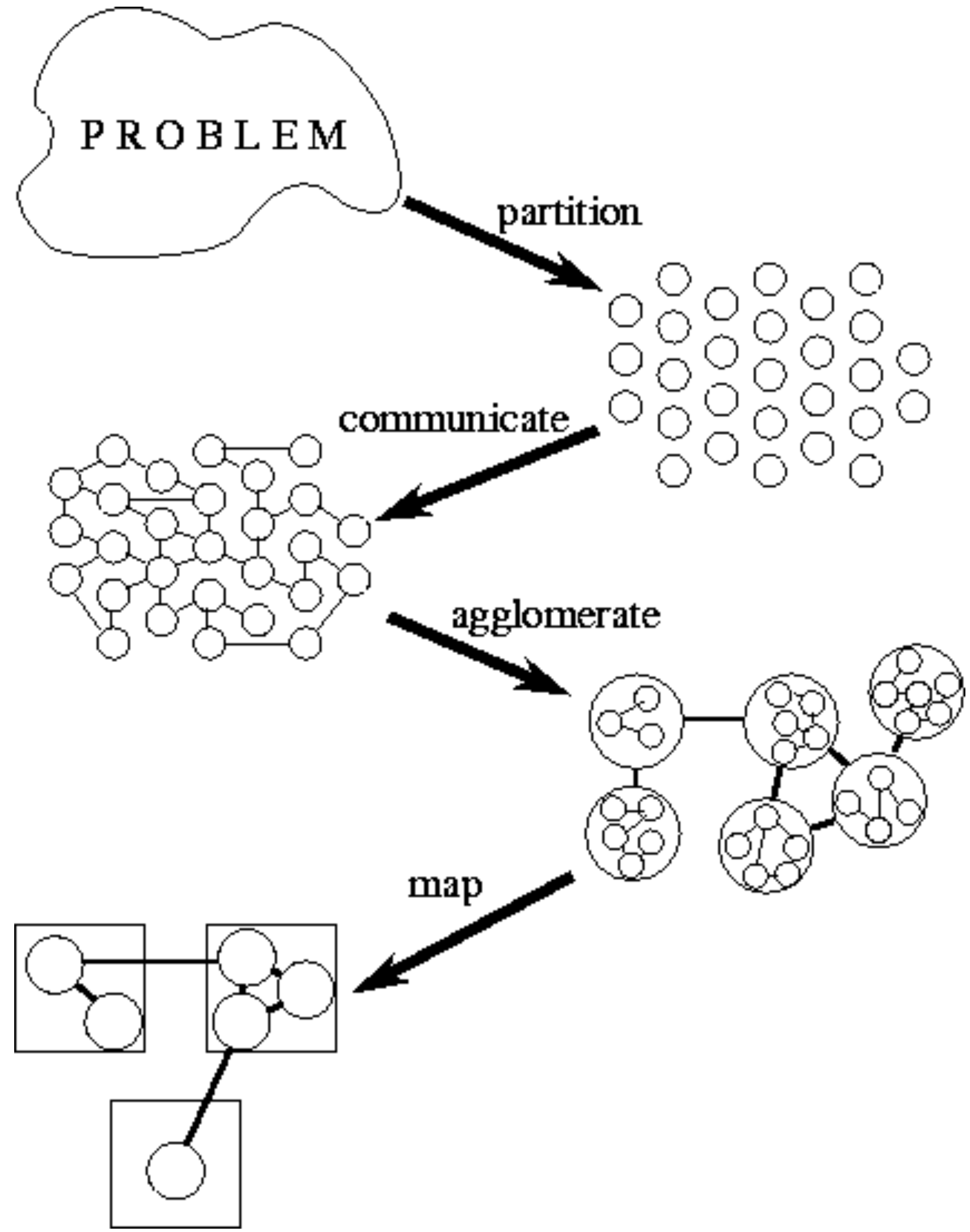
Methodological Approach to Parallel Algorithm Design:

1) Partitioning

2) Communication

3) Agglomeration

4) Mapping



Communication

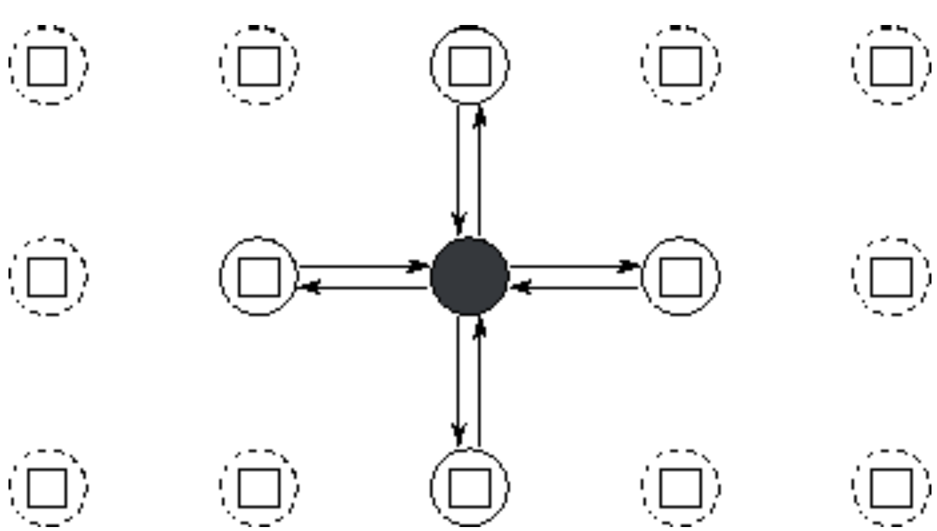
- Identify the necessary communication between the fine-grained tasks to perform the necessary computation

- For functional decomposition, this task is often relatively straightforward

- For domain decomposition, this can a challenging task. We'll consider some examples:

Finite Difference Relaxation:

$$f_{i,j}^{t+1} = \frac{4f_{i,j}^t + f_{i-1,j}^t + f_{i+1,j}^t + f_{i,j-1}^t + f_{i,j+1}^t}{8}$$

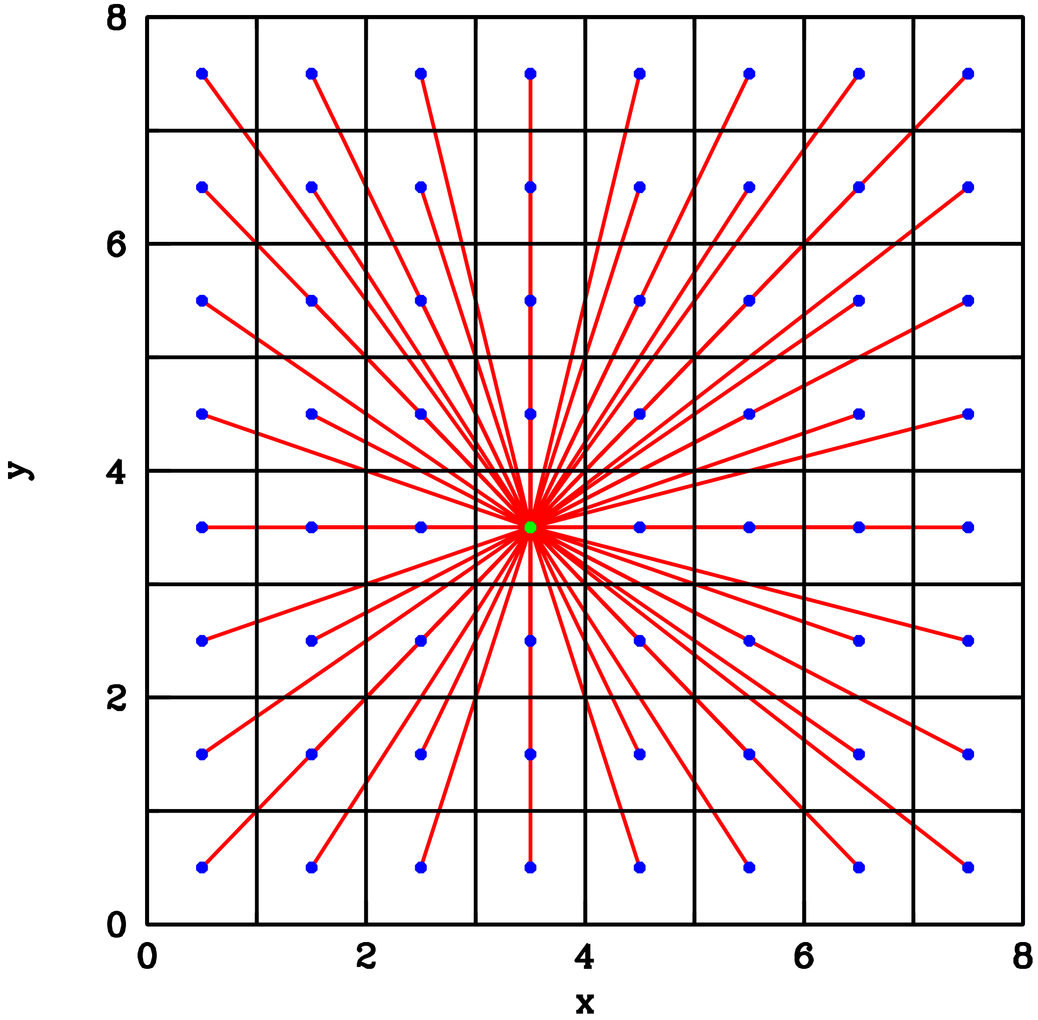


- This is a local communication, involving only neighboring tasks

Communication

Gravitational N-Body Problems:

- This is a global communication, requiring information from all tasks



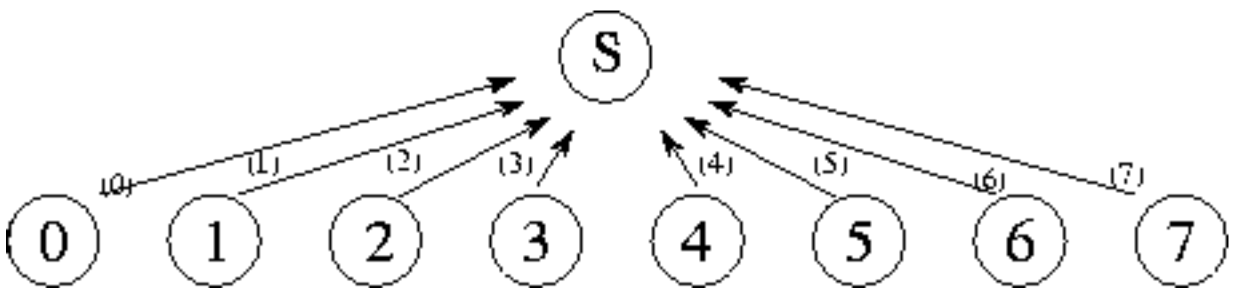
When communication is necessary, it is important to employ a scheme that executes the communications between different tasks concurrently.

Schemes for Global Communication

Consider the problem of summing the values on N=8 different processors

- This is an example of a parallel process generically called reduction.

Method 1: Summing by a Manager task, S



- Requires N=8 communications

- If all processors require the sum, it will require 2N=16 communications

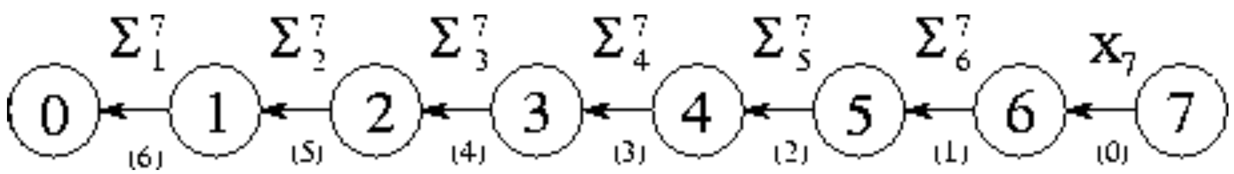
This is a poor parallel algorithm!

- Two properties of this method hinder parallel execution:
  - The algorithm is centralized, the manager participates in all interactions
  - The algorithm is sequential, without communications occurring concurrently

Schemes for Global Communication

<u>Method II</u>: Line or Ring Communications

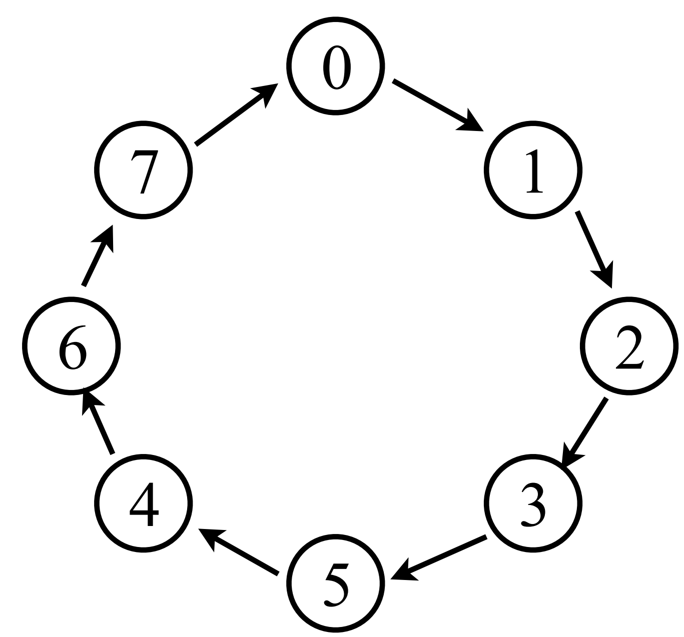
• By decentralizing, one can achieve some savings



• Requires N-1=7 communications, but it is still sequential

• If all processors require the sum, we can achieve this result with the same number of concurrent communications

- By arranging the communcations in a ring, we can distribute the sum at all processors in N-1=7 communication steps.

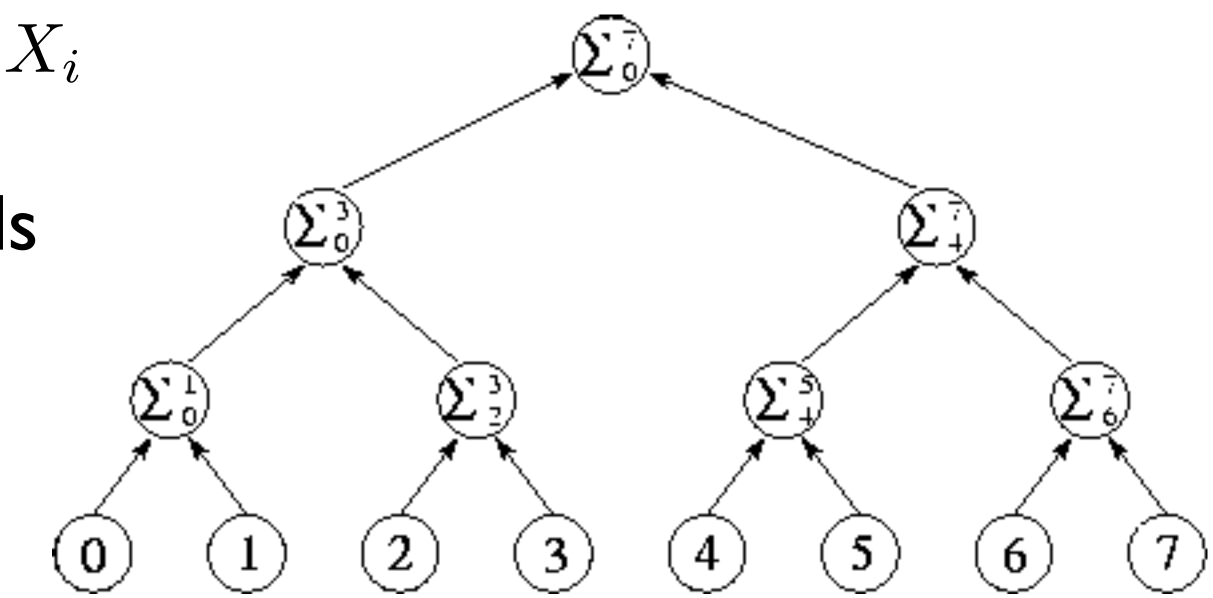


Schemes for Global Communication

Method III: Tree Communications

- But we can do better by using a divide and conquer approach to the problem
  - Split problem into two of equivalent size, to be performed concurrently

$$\sum_{i=0}^{N-1} X_i = \sum_{i=0}^{N/2-1} X_i + \sum_{i=N/2}^{N-1} X_i$$



- Recursive application of this principle leads to a tree approach

- Requires $\log_2 N=3$ communication steps

- Distribution of the sum to all processors can be accomplished with the same $\log_2 N=3$ communication steps.

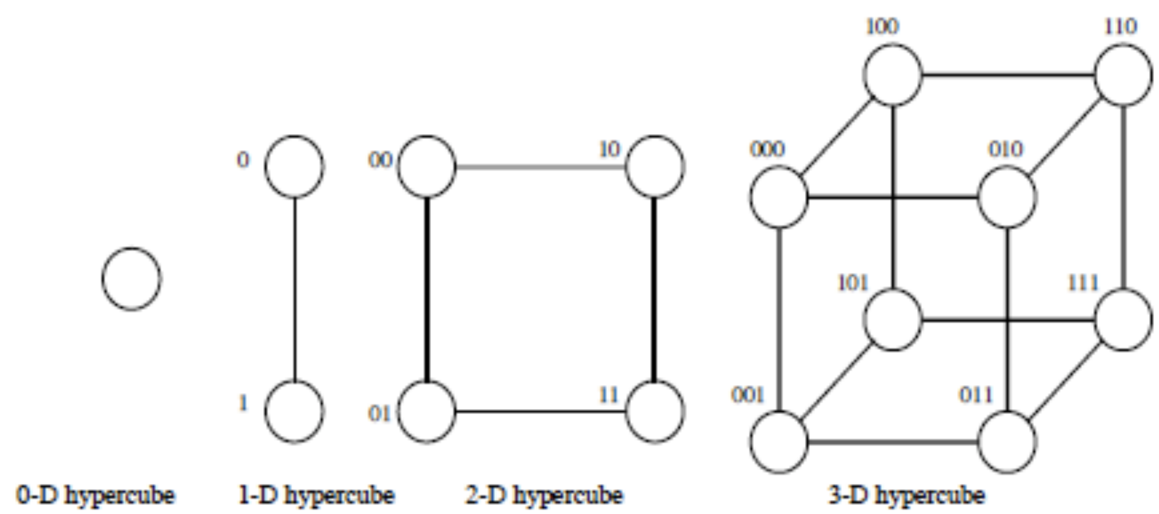
This is called a hypercube communication scheme

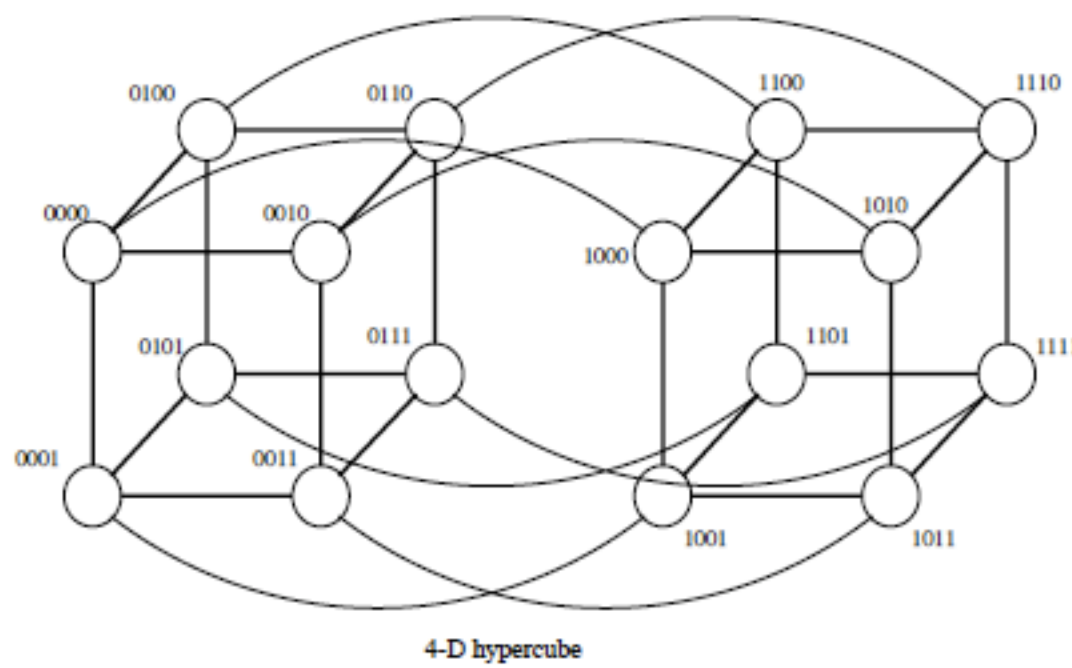
Hypercube Communication

In Hypercube Communications,

-All tasks communicate with one other tasks at each step,

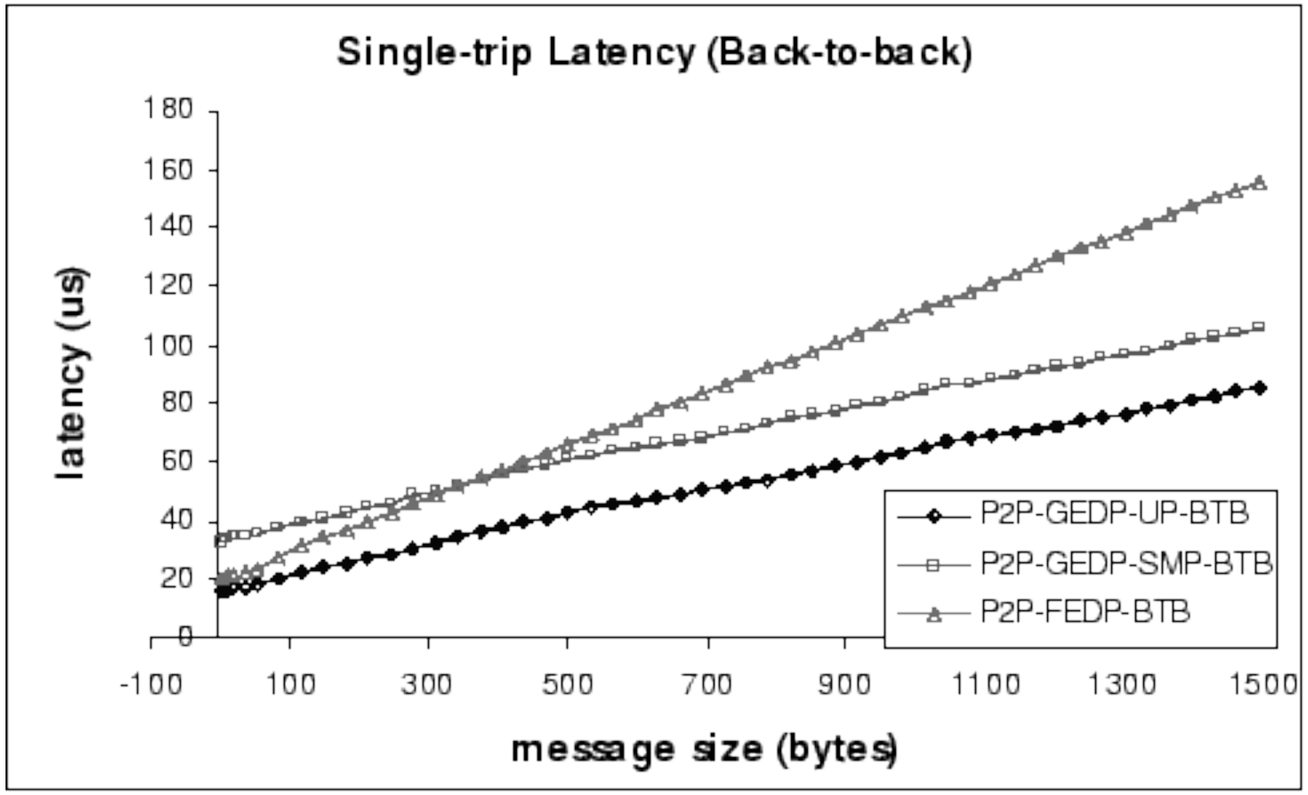
-At each step, the task passes along all of the information it has gathered up to that point





Communication: Latency vs. Bandwidth

Cost of Communications (Overhead):



- Latency: The time it takes to send a minimal message (1 bit) from A to B

- Bandwidth: The amount of data that can be communicated per unit of time

Factors to consider:

- Sending many small messages will cause latency to dominate the communications overhead
  - It is better to package many small messages into one large message

- The less information that needs to be transmitted, the less time the communications will require.

- It is often best to have all necessary communication occur at the same time

Synchronous vs. Asynchronous Communication

Consider a communication involving a message sent from task A to task B

Synchronous Communication:

• Task A sends the message, and must wait until task B receives message to move on

• Also known as blocking communication

Asynchronous Communication:

• After task A has sent the message, it can move on to do other work. When task B receives the message doesn't matter to task A.

• Also known as non-blocking communication

• Requires care to insure that different tasks don't get wildly out of step, possibly leading to race conditions or deadlocks.

P C A M

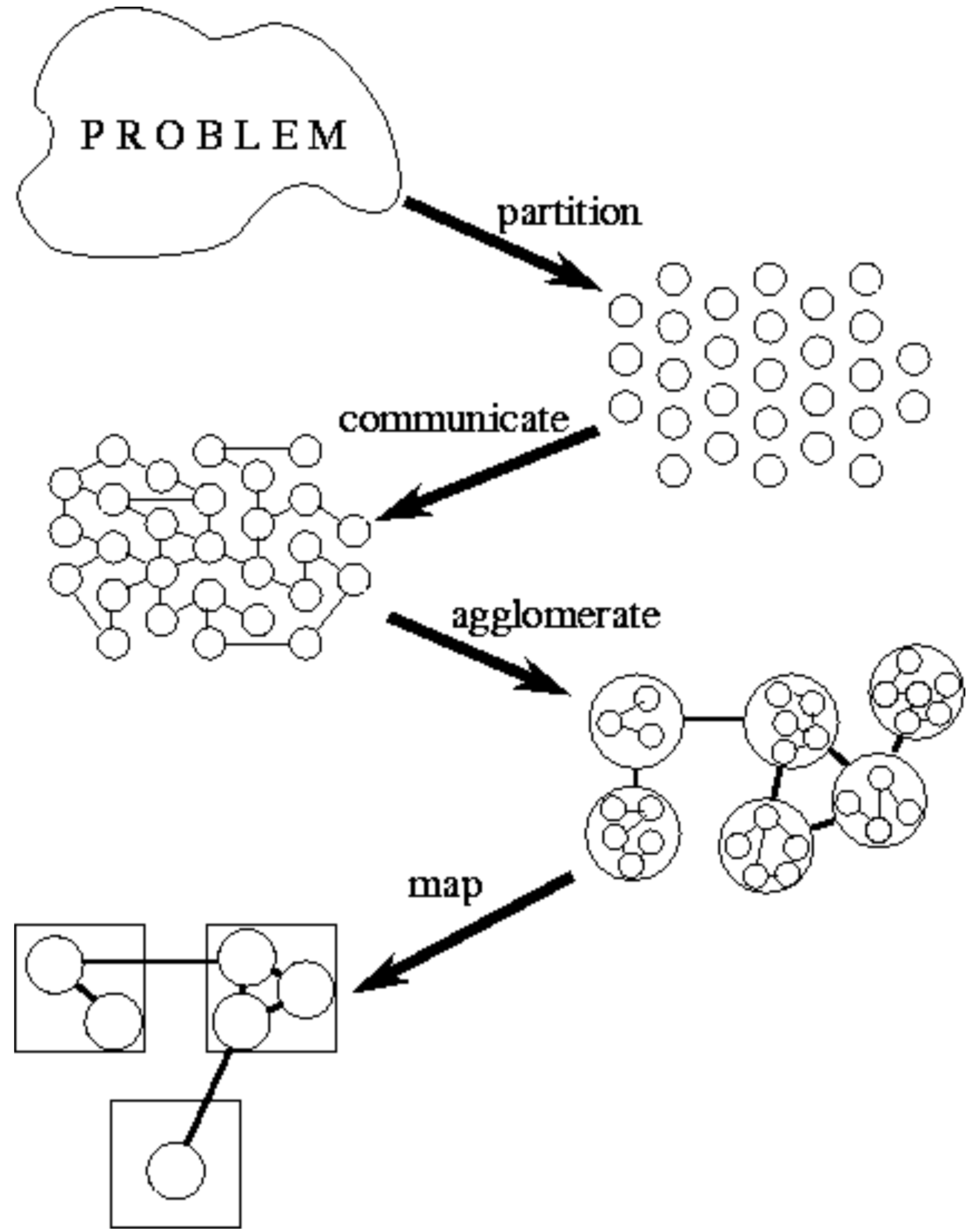
Methodological Approach to Parallel Algorithm Design:

1) Partitioning

2) Communication

3) Agglomeration

4) Mapping



Agglomeration

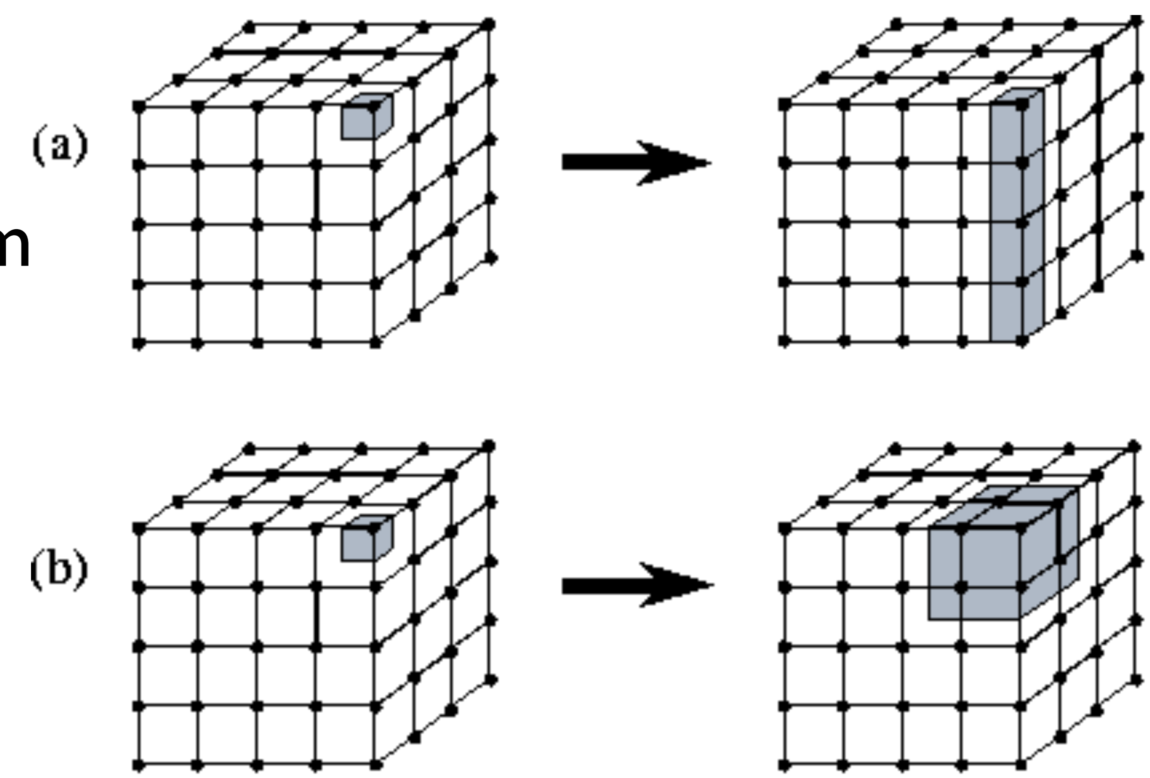
• Fine-grained partitioning of a problem is generally not an efficient parallel design
  - Requires too much communication of data to be efficient

• Agglomeration is required to achieve data locality and good performance

<u>Agglomeration</u>:

• Combine the many fine-grained tasks from partitioning into fewer coarse-grained tasks of larger size

• This task must take into account the details of the problem in order to achieve an algorithm with good scaling properties and good efficiency



Granularity

Granularity is the ratio of local computation to communication.

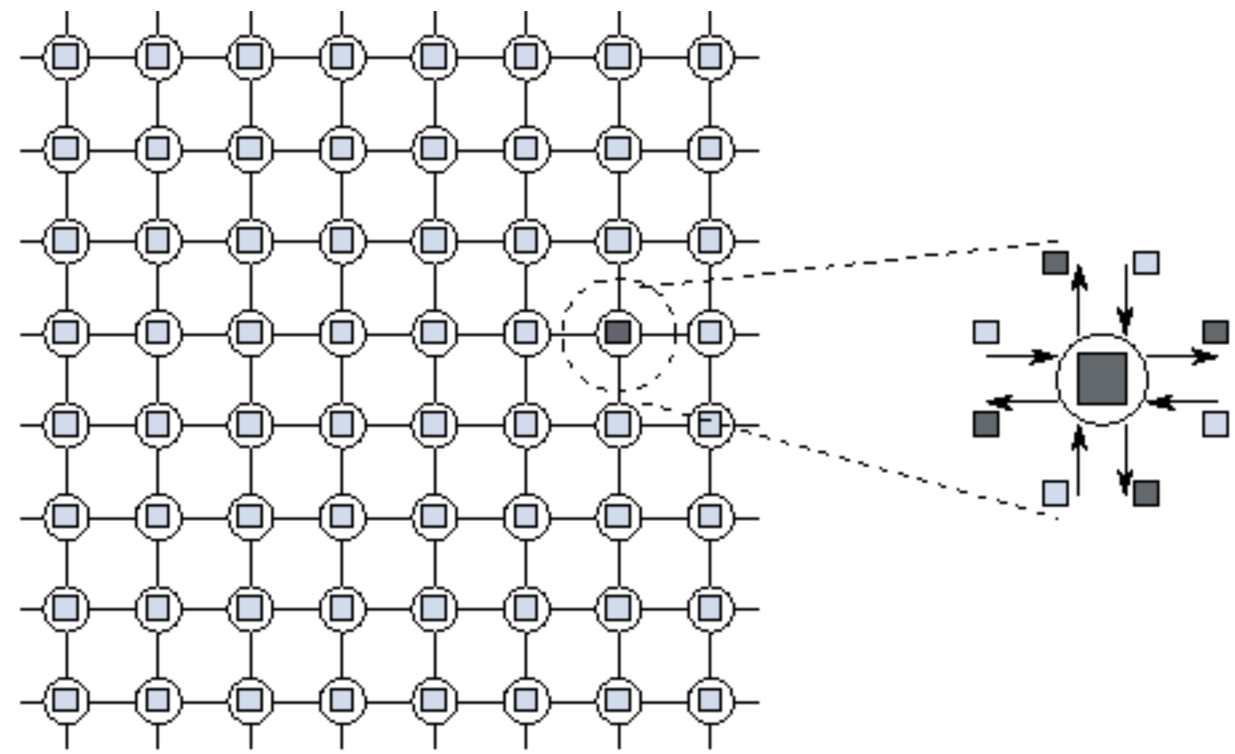
- Agglomeration is used to increase the granularity, improving performance since communication is slow compared to computation.

- By combining many finely grained tasks, we reduce both:
  (i) number of communications
  (ii) size of communications

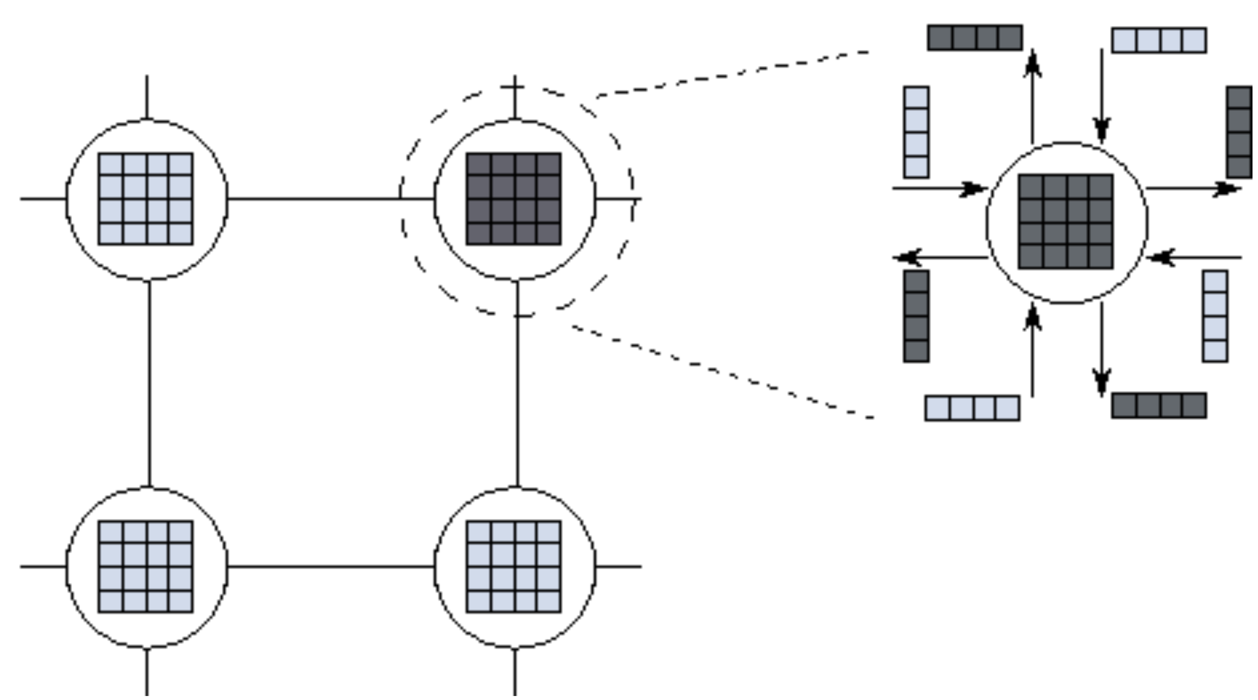
- In (a), updating 16 points requires
  (i) 16x4=64 communications
  (ii) passing 64 "bits"

- In (b), updating 16 points requires
  (i) 4 communications
  (ii) passing 16 "bits"

(a)



(b)



Surface-to-Volume in Domain Decomposition

For domain decomposition in problems with local data dependency, (ex. finite difference):

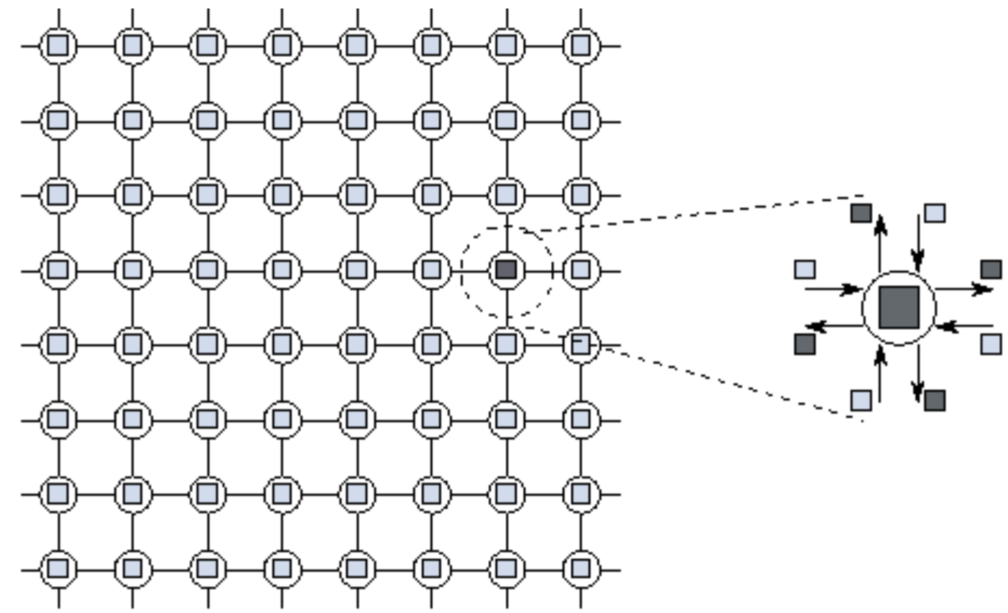
- Communication is proportional to subdomain surface area
- Computation is proportional to volume of the subdomain

<u>For this 2-D problem</u>:

(a) Surface $S = 4d$ & Area $A = d^2$

Thus, $$\frac{S}{A} = \frac{4}{d}$$

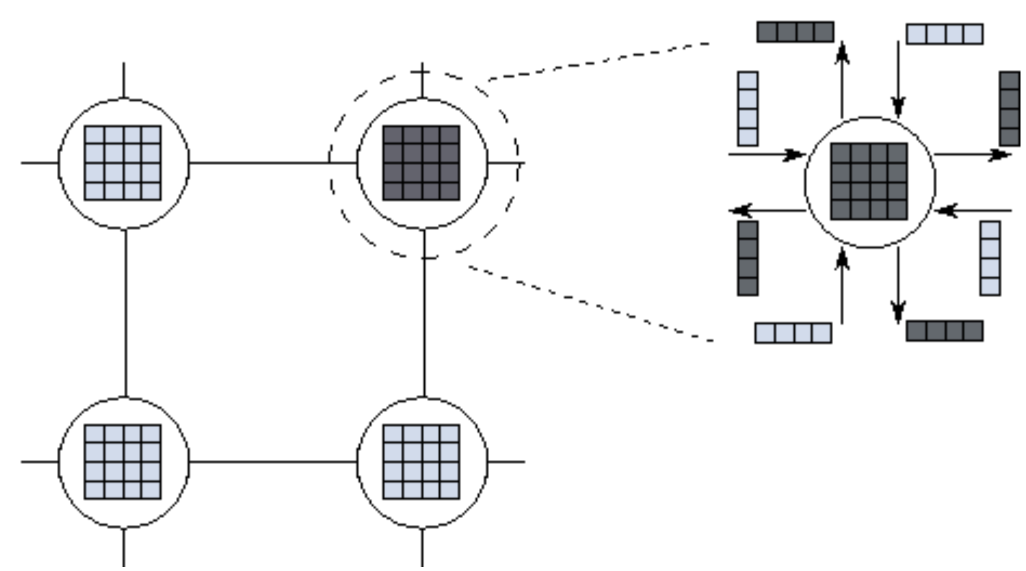
(a)



(b) Surface $S = 16d$ & Area $A = 16d^2$

Thus, $$\frac{S}{A} = \frac{1}{d}$$

(b)



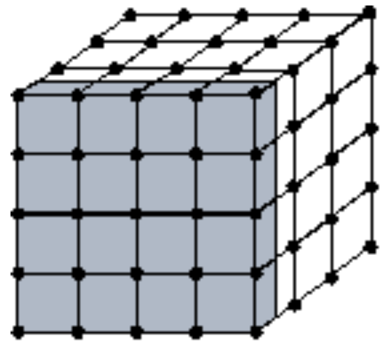
Decrease of surface-to-volume ratio is equivalent to increased granularity

Other Factors in Agglomeration

Maintaining flexibility:

- It is possible to make choices in designing a parallel algorithm that limit flexibility

- For example, if 3-D data is decomposed in only 1-D, it will limit the scalability of the application



We'll see this later in the weak scaling example of HYDRO

Replication of Data and/or Computation:

- Sometimes significant savings in communication can be made by replicating either data or computation

- Although from a serial point of view this seems inefficient and wasteful, because communication is much slower than computation, it can often lead to significant improvements in performance.

P C A M

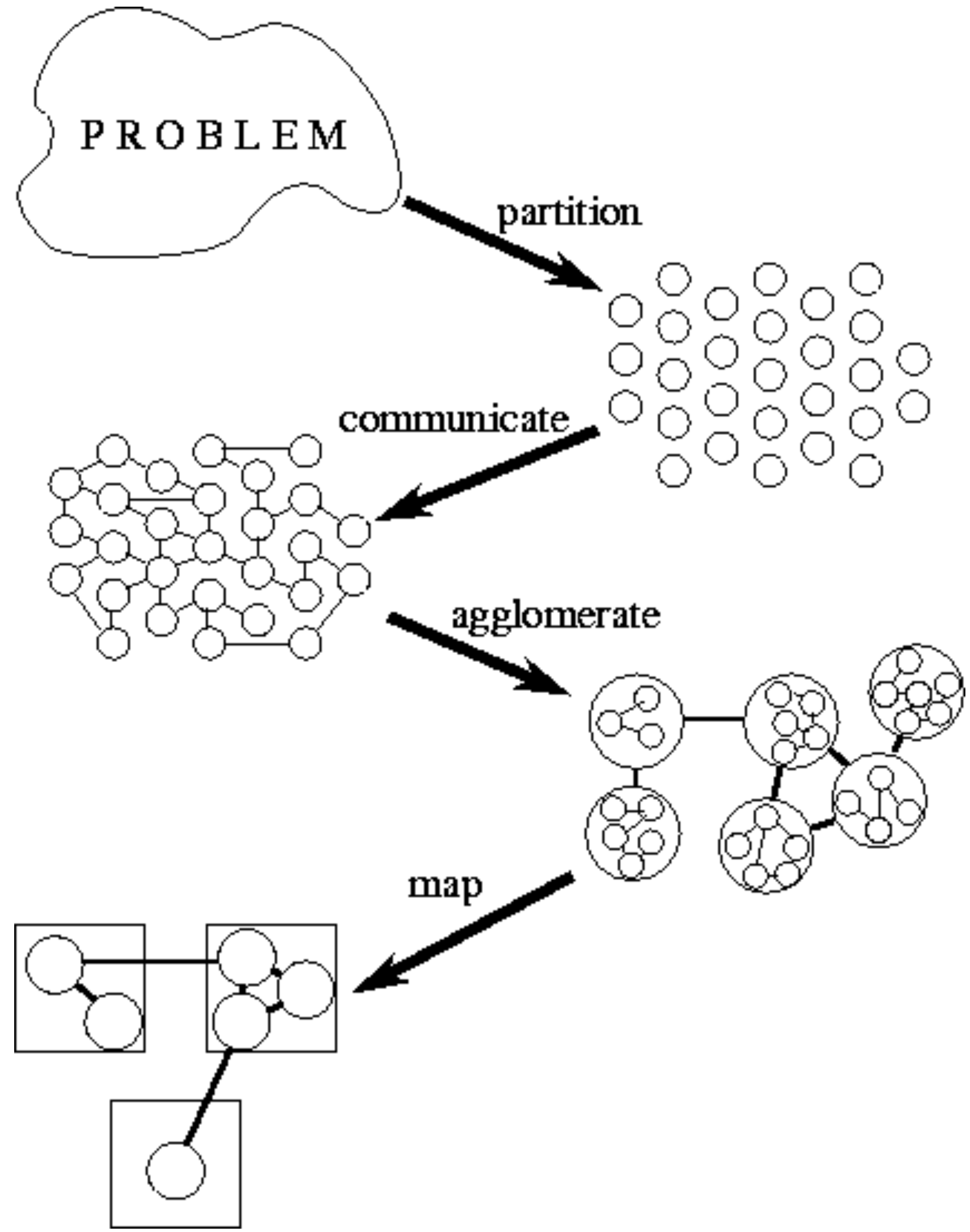
Methodological Approach to Parallel Algorithm Design:

1) Partitioning

2) Communication

3) Agglomeration

4) Mapping



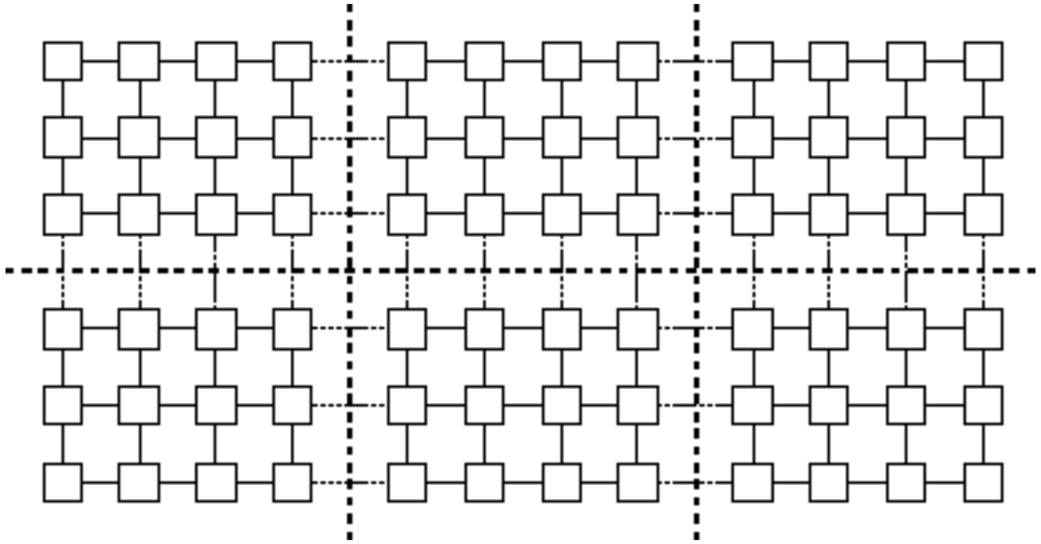
Mapping

The University of Iowa

Mapping Coarse-grained Tasks to Processors:

- Goal: To minimize total execution time

- Guidelines:
  - Tasks that can execute concurrently map to different processors
  - Tasks that communicate frequently map to the same processor

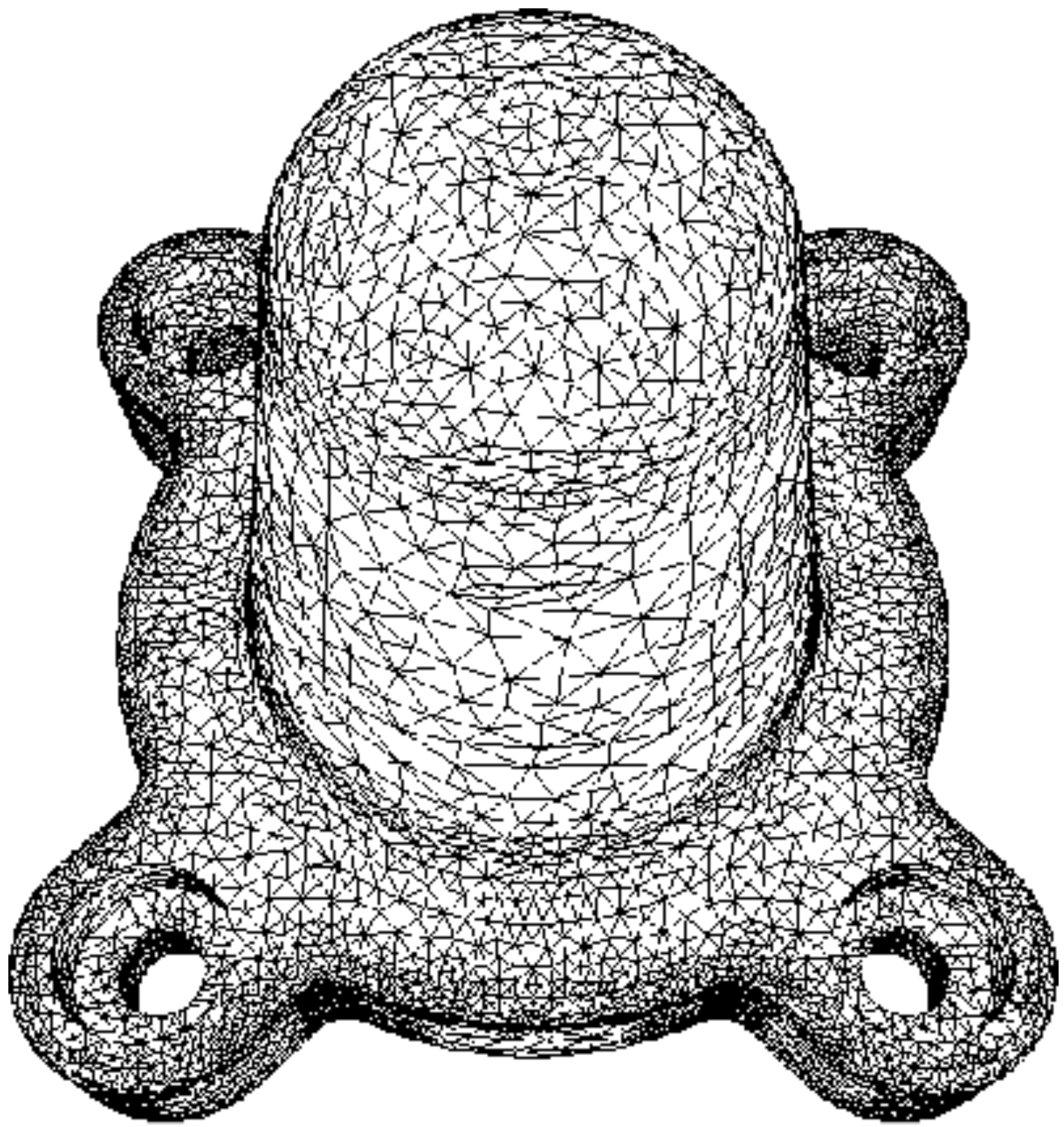


- For many domain decomposition approaches, the agglomeration stage decreases the number of coarse-grained tasks to exactly the number of processors, and the job is done

- In general, however, one wants to map tasks to achieve good load balancing

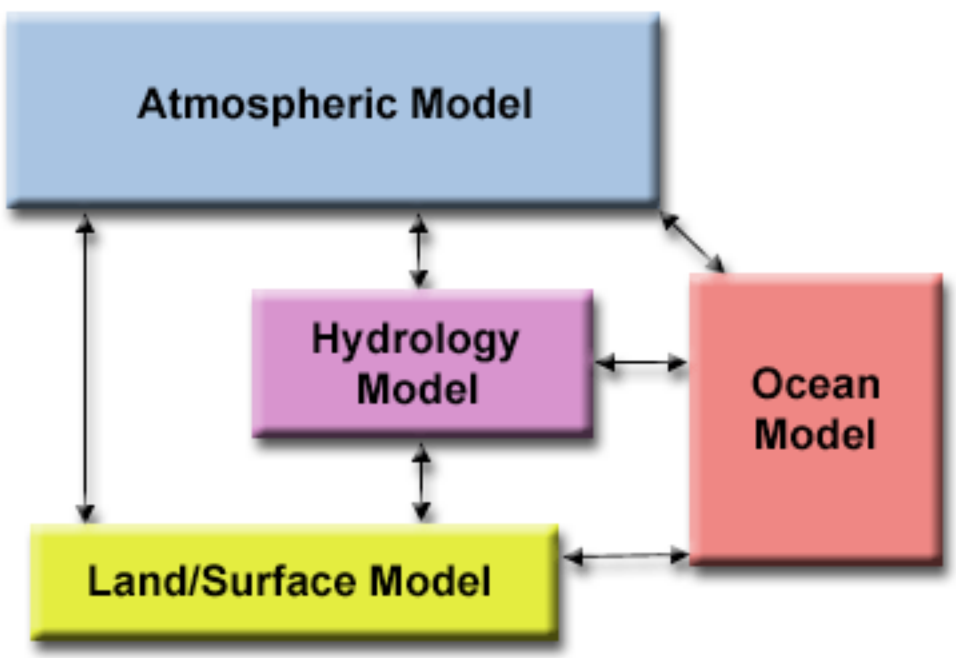
Load Balancing

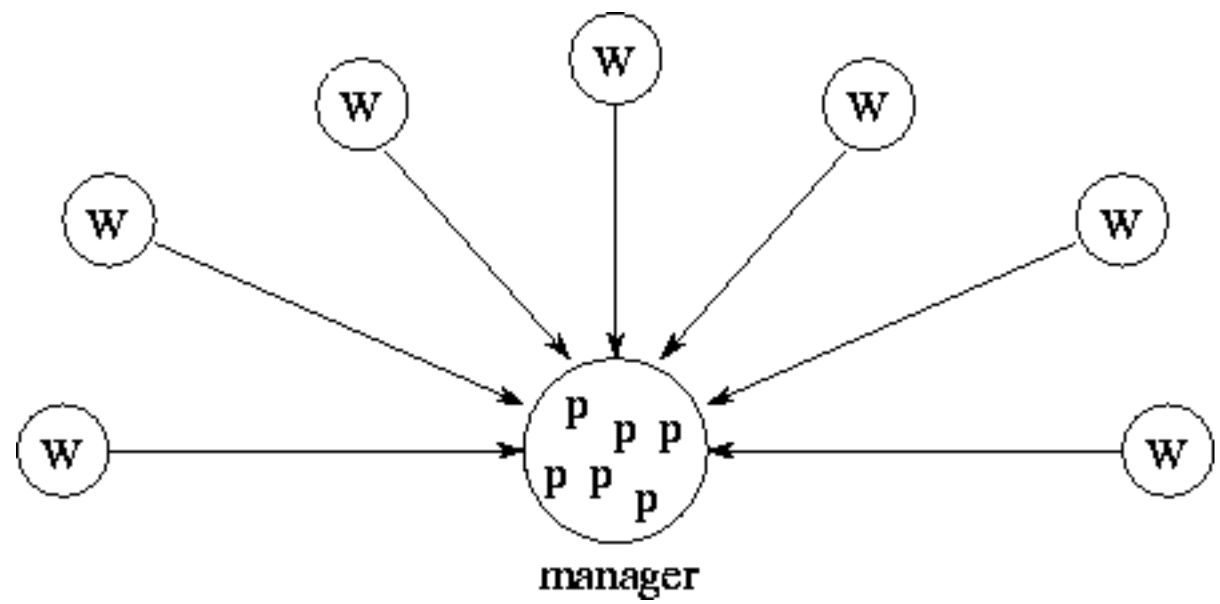
- Good parallel scaling and efficiency requires that all processors have an equal amount of work
- Otherwise, some processors will sit around idle, while others are completing their work, leading to a less efficient computation
- Complicated Load Balancing algorithms often must be employed.



Load Balancing

- For problems involving functional decomposition or a master/slave design, load balancing can be a very significant challange





Parting Thoughts

- Part of the challenge of parallel computing is that the most efficient parallelization strategy for each problem generally requires a unique solution.

- It is generally worthwhile spending significant time considering alternative algorithms to find an optimal one, rather than just implementing the first thing that comes to mind

- But, consider the time required to code a given parallel implementation
  - You can use a less efficient method if the implementation is much easier.
  - You can always improve the parallelization scheme later. Just focus on making the code parallel first.

TIME is the ultimate factor is choosing a parallelization strategy---Your Time!

References

The University of Iowa

Introductory Information on Parallel Computing

- Designing and Building Parallel Programs, Ian Foster
  http://www.mcs.anl.gov/~itf/dbpp/
  -Somewhat dated (1995), but an excellent online textbook with detailed discussion about many aspects of HPC. This presentation borrowed heavily from this reference
- Introduction to Parallel Computing, Blaise Barney
  https://computing.llnl.gov/tutorials/parallel_comp/
  -Up to date introduction to parallel computing with excellent links to further information
- MPICH2: Message Passage Inteface (MPI) Implementation
  http://www.mcs.anl.gov/research/projects/mpich2/
  -The most widely used Message Passage Interface (MPI) Implementation
- OpenMP
  http://openmp.org/wp/
  -Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran
- Numerical Recipes
  http://www.nr.com/
  -Incredibly useful reference for a wide range of numerical methods, though not focused on parallel algorithms.
- The Top 500 Computers in the World
  http://www.top500.org/
  -Updated semi-annually list of the Top 500 Supercomputers

References

Introductory Information on Parallel Computing

- Message Passing Interface (MPI), Blaise Barney
  https://computing.llnl.gov/tutorials/mpi/
  -Excellent tutorial on the use of MPI, with both Fortran and C example code

- OpenMP, Blaise Barney
  https://computing.llnl.gov/tutorials/openMP/
  -Excellent tutorial on the use of OpenMP, with both Fortran and C example code

- High Performance Computing Training Materials, Lawrence Livermore National Lab
  https://computing.llnl.gov/?set=training&page=index
  -An excellent online set of webpages with detailed tutorials on many aspects of high performance computing.