Introduction to Cloud Computing

Parallel Processing III

15-319, spring 2010
10th Lecture, Feb 11th

Majd F. Sakr

Spring 2010 ©

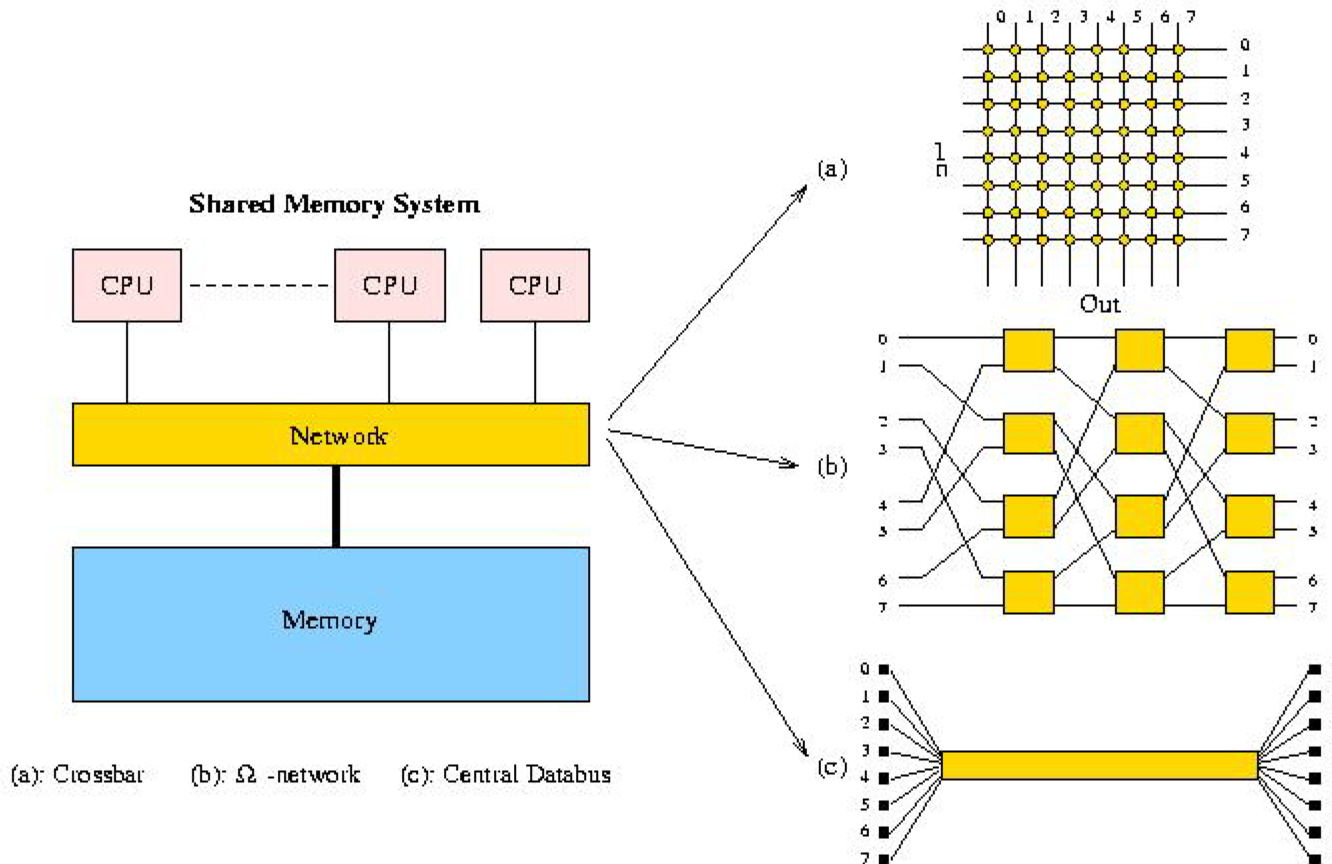
Review

- Architectures
- Interconnect

http://www.phys.uu.nl/~steen/web03/sm-mimd.html

Review

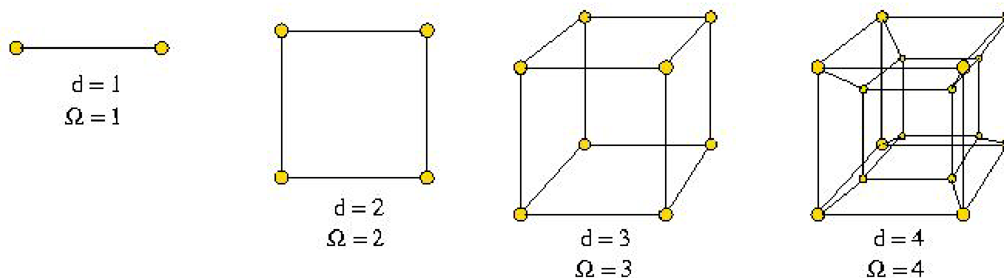
- Shared Memory MIMD



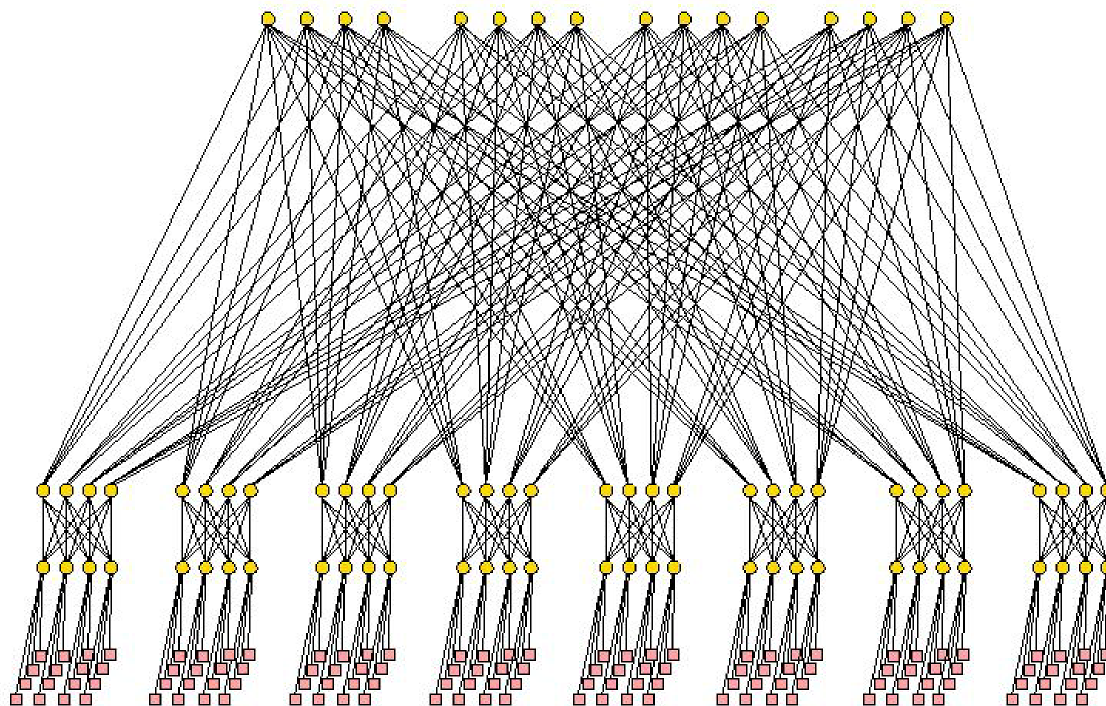
http://www.phys.uu.nl/~steen/web03/sm-mimd.html

Review

- **Distributed Memory MIMD**



(a) Hypercubes, dimension 1-4.

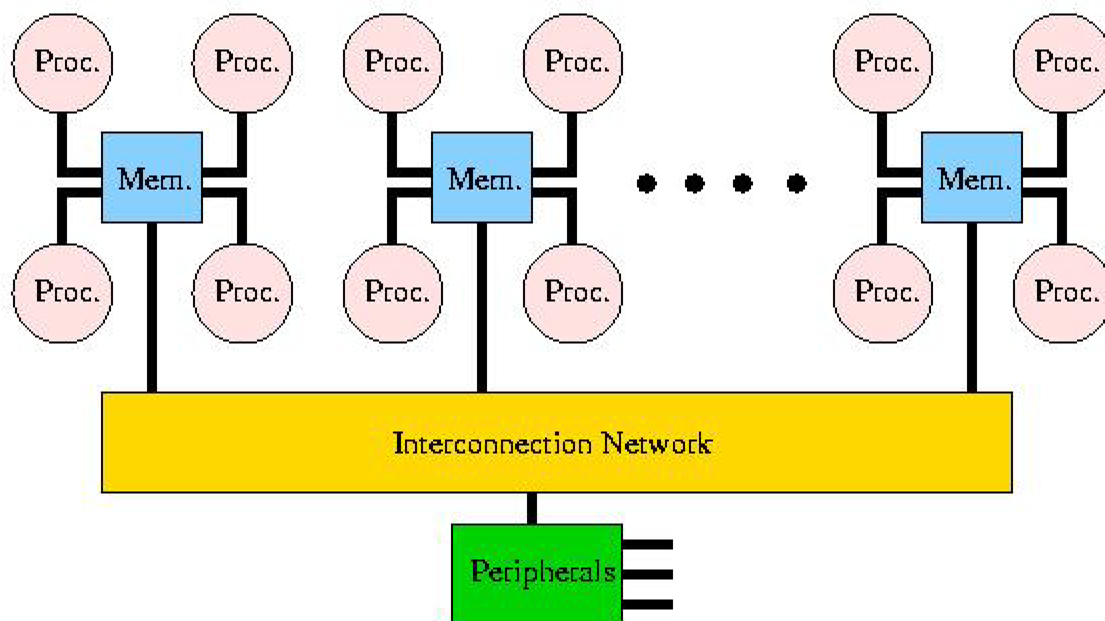


(b) A 128-way fat tree.

http://www.phys.uu.nl/~steen/web03/dm-mimd.html

Review

- **Hybrids**
  - Cache-coherent NUMA



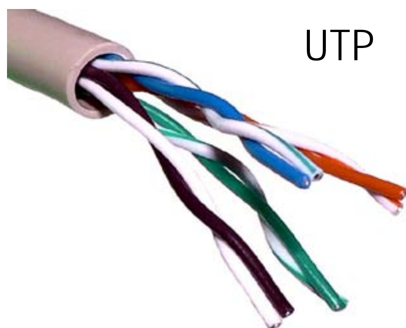
http://www.phys.uu.nl/~steen/web03/ccNUMA.html

High BW & Speed Networks

- Server and cluster backbones typically need fast interconnects
- Gigabit Ethernet
  - 10 Gigabit
  - 100 Gigabit
- Myrinet
- Infiniband

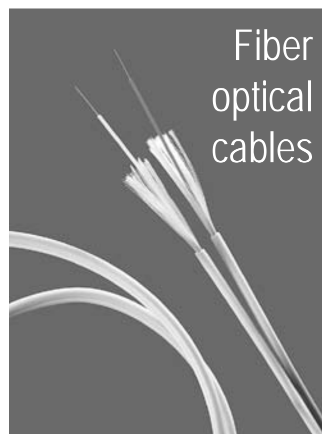
Gigabit Ethernet

- Known as "IEEE Standard 802.3z"
- Offers 1 Gbps raw bandwidth
- Speed: (10 x speed of fast Ethernet)
  (100 x speed of regular Ethernet)
- 1 Gig Ethernet uses UTP cables
- 10 Gig Ethernet and 100 Gig Ethernet are emerging technologies, typically require fiber optical cables



UTP

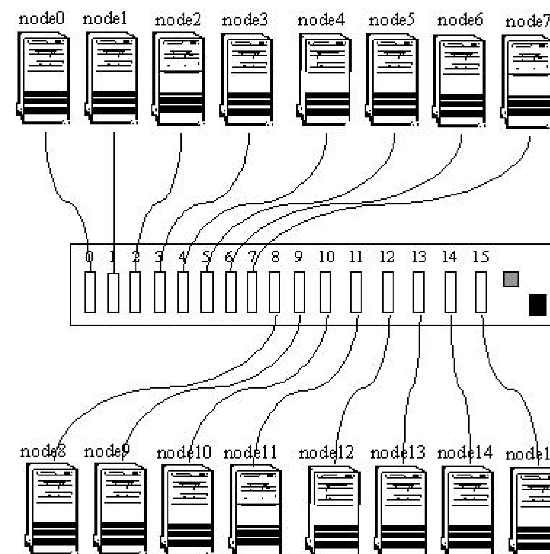
commons.wikimedia.org/wiki/File:UTP_cable.jpg



http://www.directindustry.com/prod/lapp-group/fiber-optic-cable-17287-404578.html

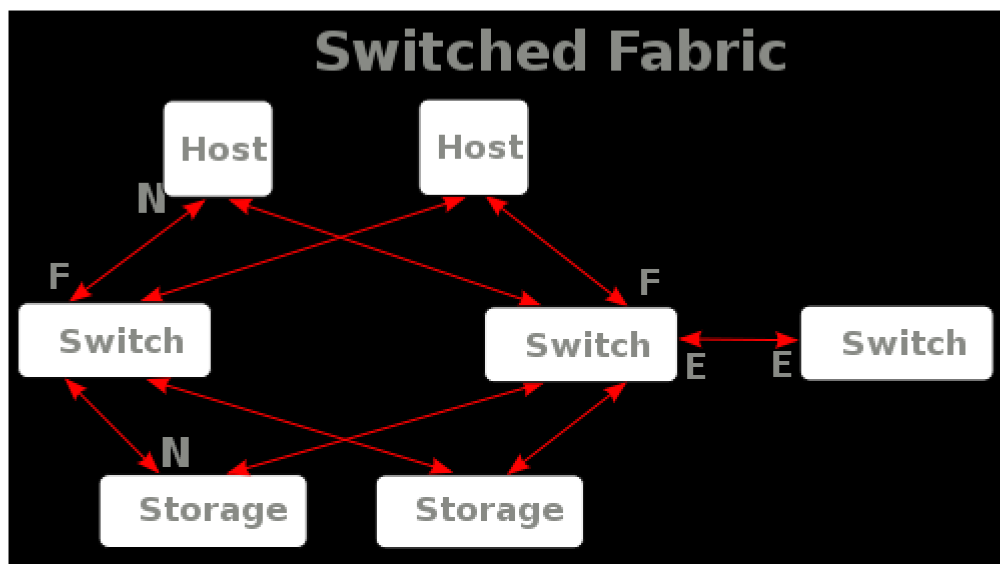
Myrinet

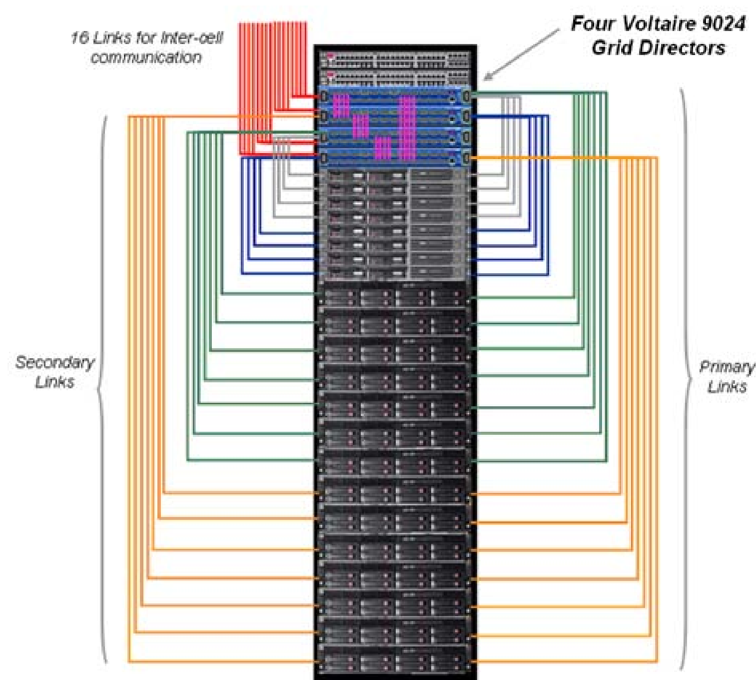
- High-speed Local Area Network Interconnect
- Typically requires two fiber optic cables per node (upstream and downstream)
- Offers low-latency networking with low protocol overhead @ 1.9 Gbps (messages in usec range)
- Next Generation (Myri-10G) is 10 Gbps.



Infiniband

- High-bandwidth interconnect primarily for processors to high performance I/O devices
- InfiniBand offers point-to-point bidirectional serial links which forms a switched fabric
- Upto 120 Gbps theoretical bandwidth (message in usec range)





Lecture Outline

- Parallel Computing Design Considerations
- Limits and Costs of Parallel Computing
- Parallel Computing Performance Analysis
- Examples of Problems Solved By Parallel Computing

How to Parallelize

- Automatic vs. Manual Parallelization

- Design Considerations

1. Can the Problem be parallelized?
2. Program's hotspots & bottlenecks?
3. Partitioning
4. Communications
5. Synchronization
6. Data Dependencies
7. Load Balancing
8. Granularity
9. Input/Output

Automatic VS Manual Parallelism

- **Mostly, developing parallel programs has been manual. This is complex, time consuming, and error-prone process.**

- **Parallelizing compiler or pre-processor is used to parallelize serial code. This complier usually works in two different ways:**
  - **Fully Automatic:**

    The compiler analyzes the source code and specifies parts that could be parallelized.
  - **Programmer Directed:**

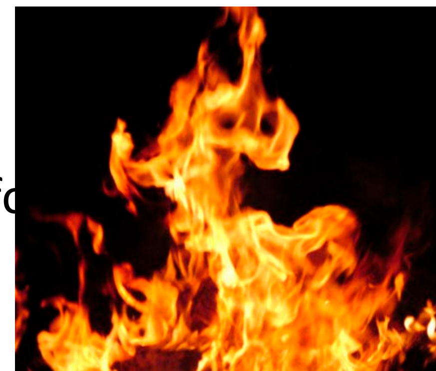
    The programmer uses compiler flags to explicitly tell the compiler how to parallelize the code.

(1) Can the Problem be Parallelized?

- **Parallelism Inhibitors:**
  - ✓ Control vs. data dependencies
  - ✓ Examples:
    - ➢ Parallelizable Problem: Multiply each element of the array by 2
    - ➢ Non-parallelizable Problem: Fibonacci sequence
  - ✓ Handling Data Dependencies
- **Parallelism Slow-down:**
  - ✓ Communications bottleneck

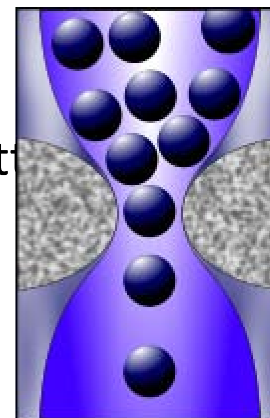
(2) Hotspots & Bottlenecks

- **Hotspots:**
  - What are they? Account for most of CPU usage
  - How to define them in the program? Profiling & Perfo
    Analysis
  - Parallelism focus should be on these spots



Source: http://scavenging.wordpress.com/2009/05/

- **Bottlenecks:**
  - What are they? slow areas
  - Can we redesign the algorithm to reduce /eliminate bott



Source: http://zubinmehta.files.wordpress.com/

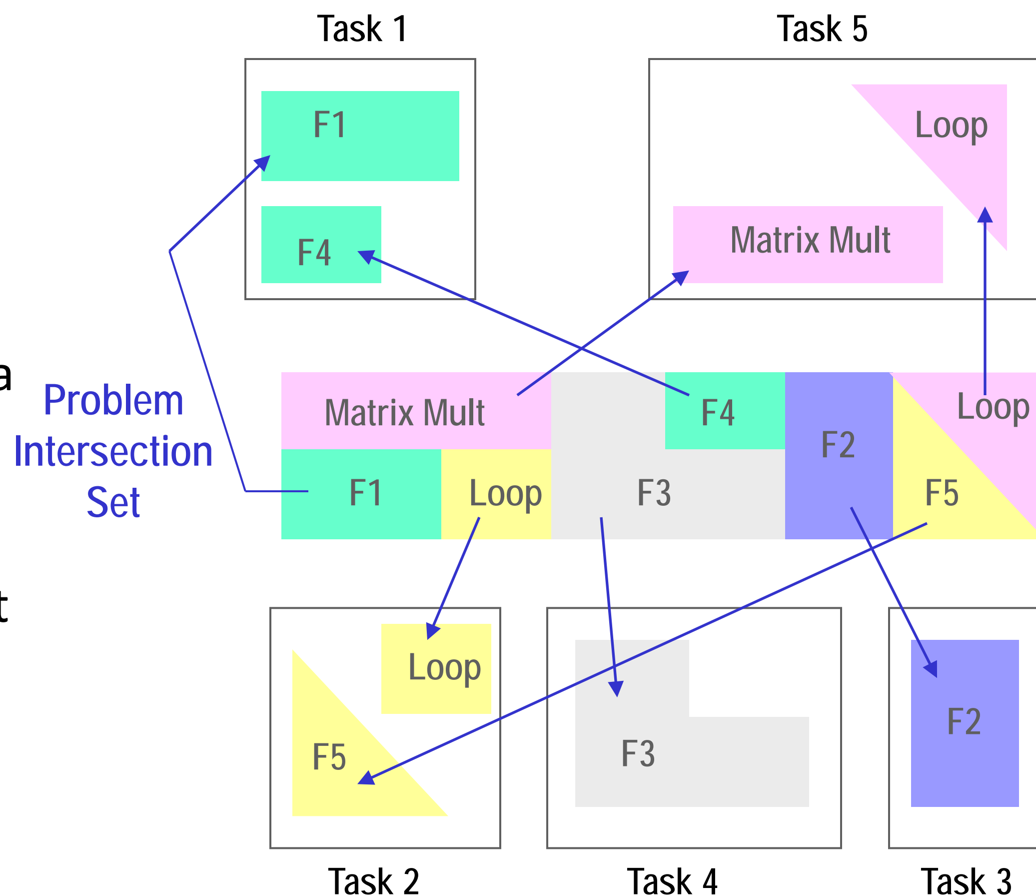
(3) Partitioning/Decomposition

- **Dividing the problem into chunks/parts of work that can be distributed to multiple tasks.**
- **Best Partitioning happens where there is minimum I/O & communication**
- **Ways to Partition?**
  - Domain Decomposition
  - Functional Decomposition

http://tinypic.com/view.php?pic=a29ah0&s=3

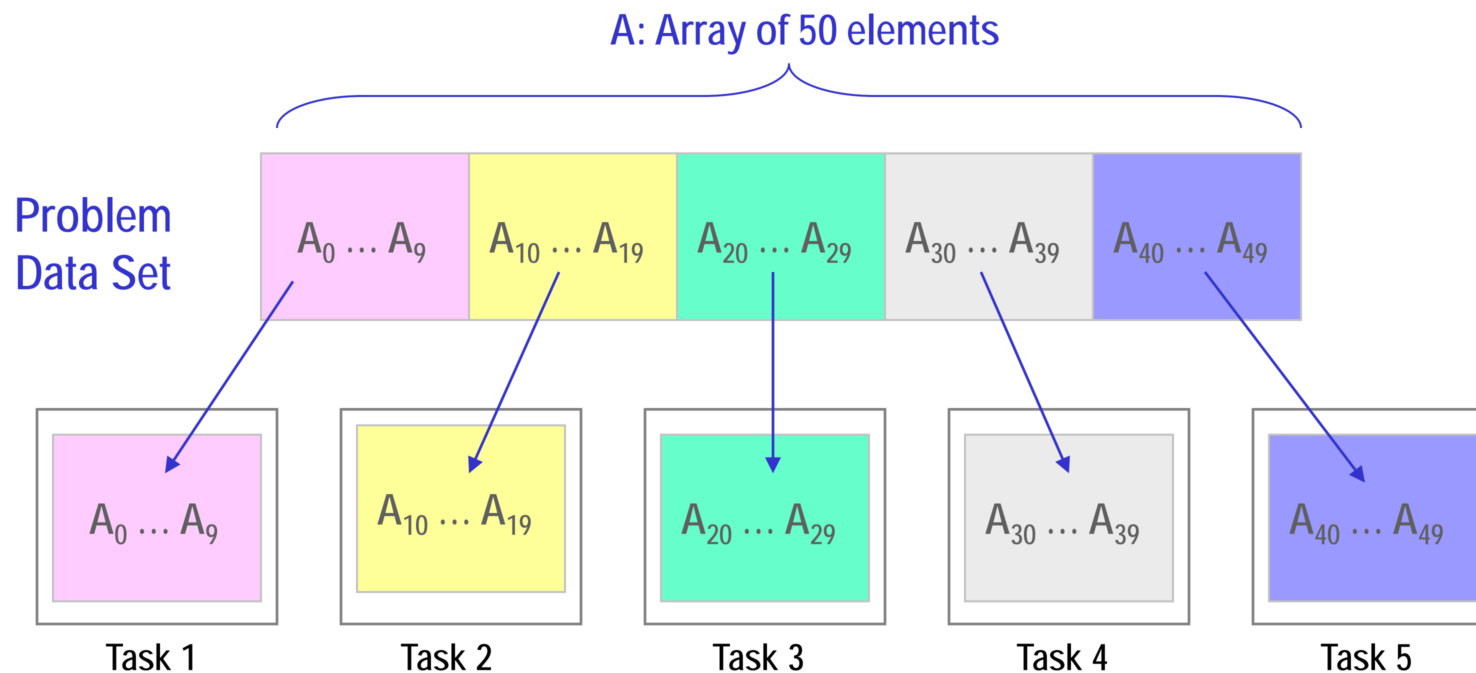
(3) Partitioning/Decomposition

- **Functional Decomposition**
  - The focus is on the computation to be performed, not on the data given to the problem.
  - The problem is partitioned best on the work that must be done. Each task computes a part of the overall work.



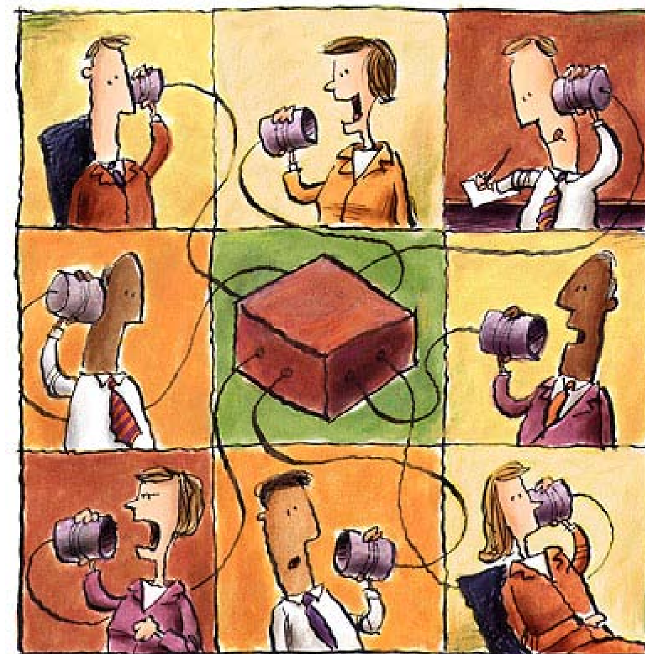
(3) Partitioning/Decomposition

- **Domain Decomposition**
  - The data given to the problem is divided into chunks.
  - Each chunk is given to a task that performs the operation on it. Tasks run in parallel.



(4) Communication

- Do we need inter-task communication?
- Inter-task Communication Considerations:
  1. Costs
  2. Visibility
  3. Synchronous vs. Asynchronous
  4. Scope
  5. Efficiency



(4) Communication

- **When is task communication needed?**
  - **Embarrassingly parallel problems**

    Problem is simple that it can be partitioned into tasks with no need for the tasks to share any data.
    - Loosely coupled
  - **Complex problems**

    Problem is complex, it can be partitioned into tasks, but tasks need to share data with each other to complete the computations.
    - Tightly coupled

Spring 2010 ©

(4) Communication: Considerations

- **Inter-task Communication Considerations:**

  1. **Costs:**

- **Time Cost:**
  - **Time waiting to synchronize**
  - **Bandwidth saturation**
  - **Bandwidth VS Latency**
- **Resources Costs:**
  - Machine cycles and resources
- **Both (Time & Resources Cost):**
  - Overhead & Complexity

(4) Communication: Considerations

1) Costs

http://theragblog.blogspot.com/2009/04/

- **Time Cost**
  - **Time waiting to synchronize**

    Communications synchronization between tasks. Tasks spend some of their time waiting for others instead of working.
  - **Bandwidth saturation**

    Competing communication traffic fill the network bandwidth.
  - **Bandwidth VS Latency**
    - Latency is the time it takes to send a minimum size message.
    - Bandwidth is the amount of data that can be transferred per time unit.
    - Small messages experience both delays. It's better to package multiple small messages in one big message.

(4) Communication: Considerations

1) Costs

- **Resources Cost**
  - **Machine cycles and resources**

    Machine cycles and resources used to package and transmit data. They are supposed to be used for computation.

http://www.mewan.net/curriculum/pshe/

- **Time and Resources Cost**
  - **Overhead & Complexity**

    Inter-task communication involve overhead in terms of time spent and resource consumed.

(4) Communication: Considerations

2) Visibility

- Task communication is more visible in some models (ex: Message Passing Model), and is under the programmer's control.
- In other models (ex: Data Parallel Model), communication is often done transparently with no control of the programmer.

Source: http://www.tqnyc.org/2009/00767/Weather%20vocabulary.html

(4) Communication: Considerations

3) Synchronous vs. asynchronous communications

Synchronous communications

- Require some type of "handshaking" between tasks that are sharing data. This could be done implicitly or explicitly.
- Blocking communications:
  Some work must be held until the communications are done.

Asynchronous communications

- Allow tasks to communicate data independently from the work they are doing.
- Non-blocking communications.
- Advantage: Interleaving computation with communication.

(4) Communication: Considerations

4) Scope

- **Communication Scope:**

  Which tasks needs to communicate with each other.

  - **Point-to-point :** two tasks are communicating, one acts as the sender/producer of data, the other acts as the receiver/consumer.
  - **Collective:** Data is communicated between more than two tasks. They are members in a common group, or collective.

http://en.wikipedia.org/wiki/File:4x_rifle_scope.jpg

Spring 2010 ©

(4) Communication: Considerations

5) Efficiency

- Efficiency varies based on the applications requirements and the programmer's decisions. Programmer must decide:
  - Which factors should have more impact on task communication.
  - Which implementation to use for the chosen model.
  - What type of communication needs to be used.
  - Network(s) type and number.

(5) Synchronization (1/4)

- **Process Synchronization:**

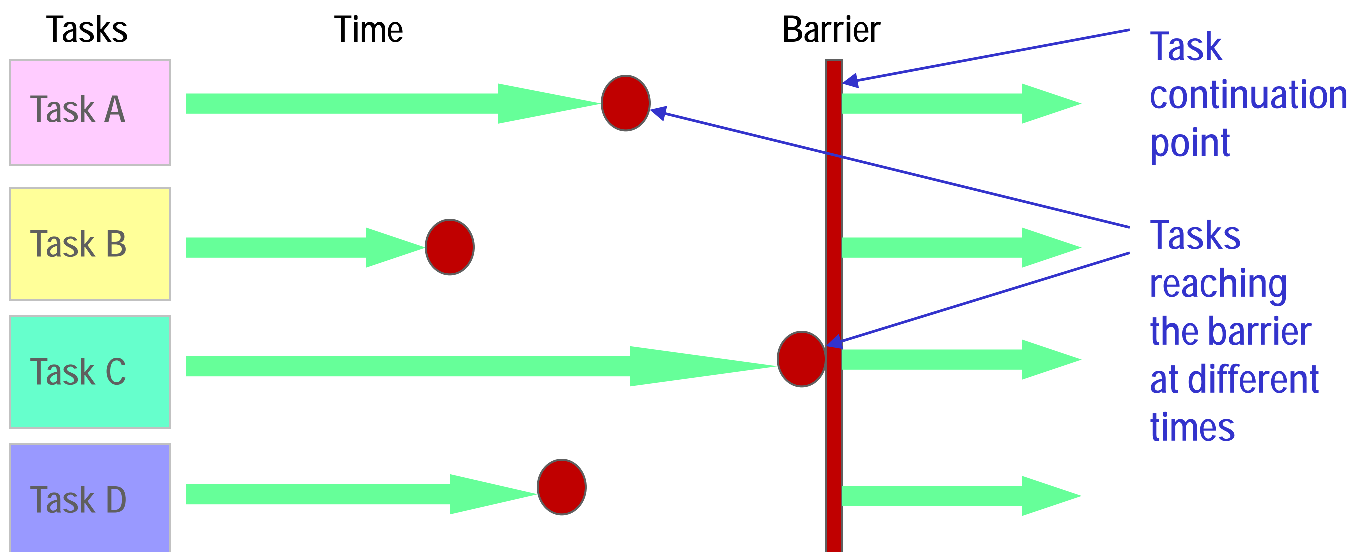
  Reaching an agreement between simultaneous processes/tasks regarding a sequence of in-order steps to complete an action

  - **Barriers**
  - **Locks/ Semaphores**
  - **Synchronous Communication Operations**

(5) Synchronization (2/4)

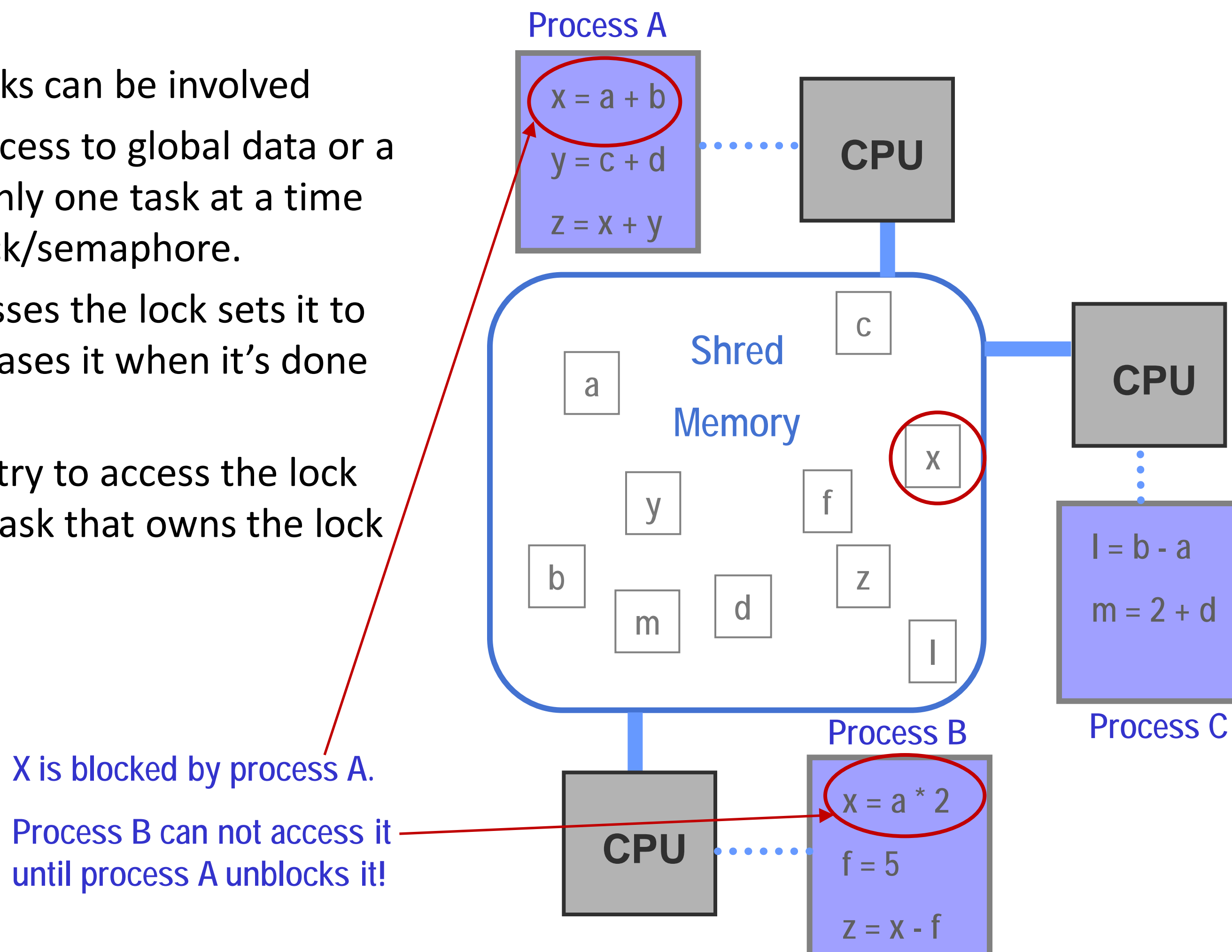
- **Barriers**
  - A point at which a task must stop, and can not proceed until all tasks are synchronized.
  - Mostly, all tasks are involved.
  - Each task keeps performing its work until reaching a barrier point. Then, it stops and keeps waiting for the last task to reach the barrier.
  - When last task reaches

    the barrier, all tasks are synchronized.
  - From this point, tasks continue their work.



(5) Synchronization (3/4)

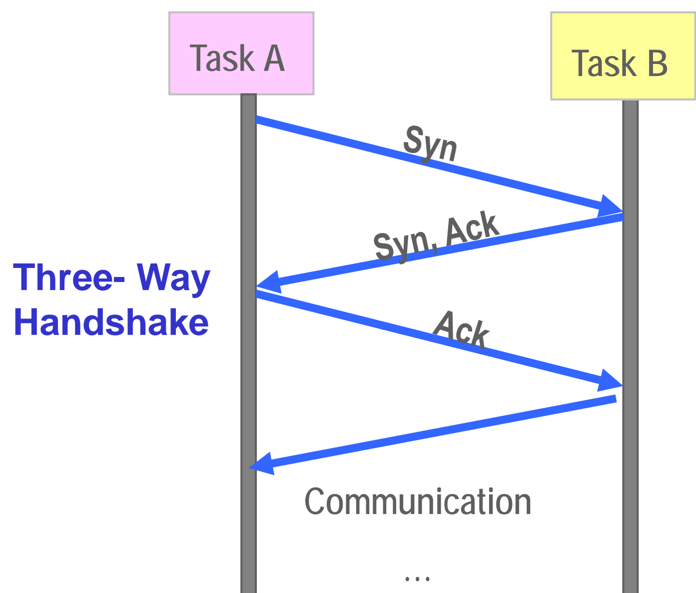
- **Locks/ Semaphores**
  - Any number of tasks can be involved
  - Used to protect access to global data or a section of code. Only one task at a time may access the lock/semaphore.
  - The first task accesses the lock sets it to be locked and releases it when it's done with it.
  - When other tasks try to access the lock they fail until the task that owns the lock releases it.

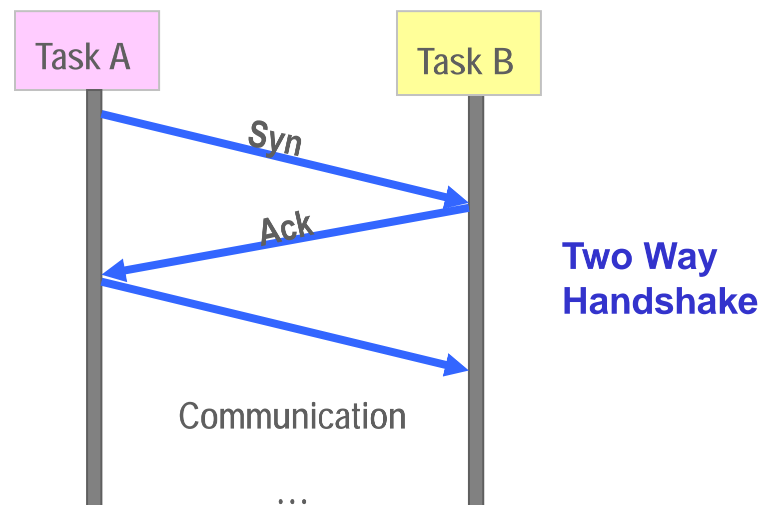


(5) Synchronization (4/4)

- **Synchronous communication operations**
  - Involves only tasks executing a communication operation
  - Coordination is required between the task that is performing an operation and the other tasks performing the communication
  - Require some type of "handshaking" between tasks that are sharing data. This could be done implicitly or explicitly.
  - Blocking communications:

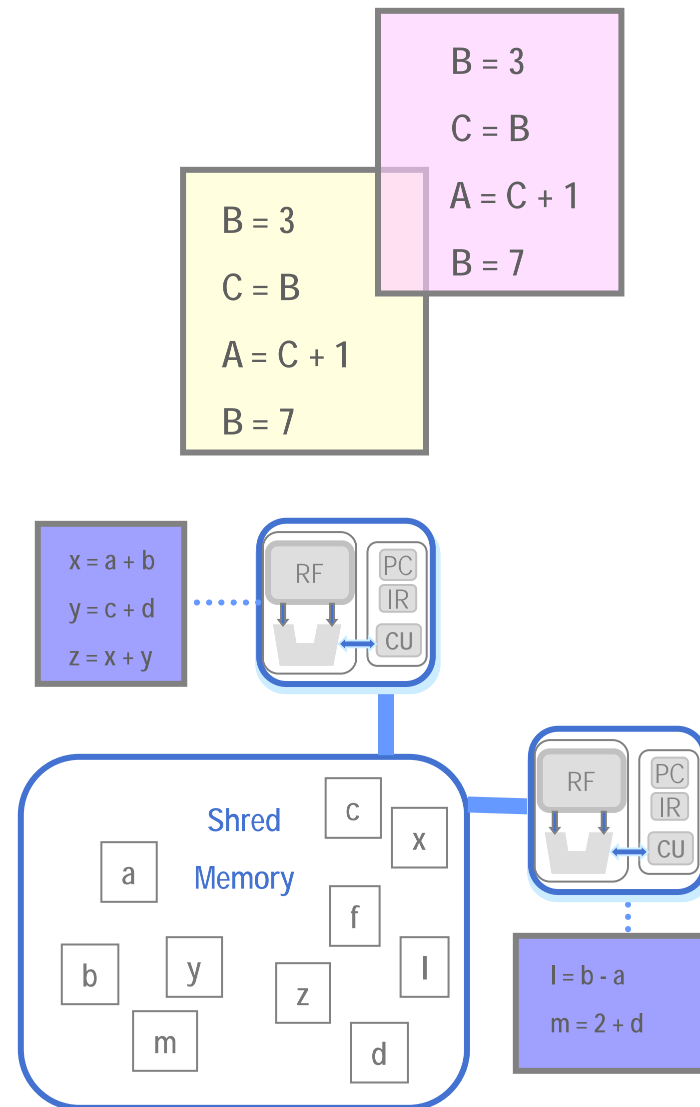
    Some work must be held until the communications are done.





(6) Data Dependencies

- The order of program statement execution effects the results of the program.
- Multiple use of data stored in the same location by multiple tasks.
- Dependencies make one of the primary inhibitors to parallelism.



(6) Data Dependencies

- **Handling Data Dependencies:**
  - **Distributed memory architectures:**

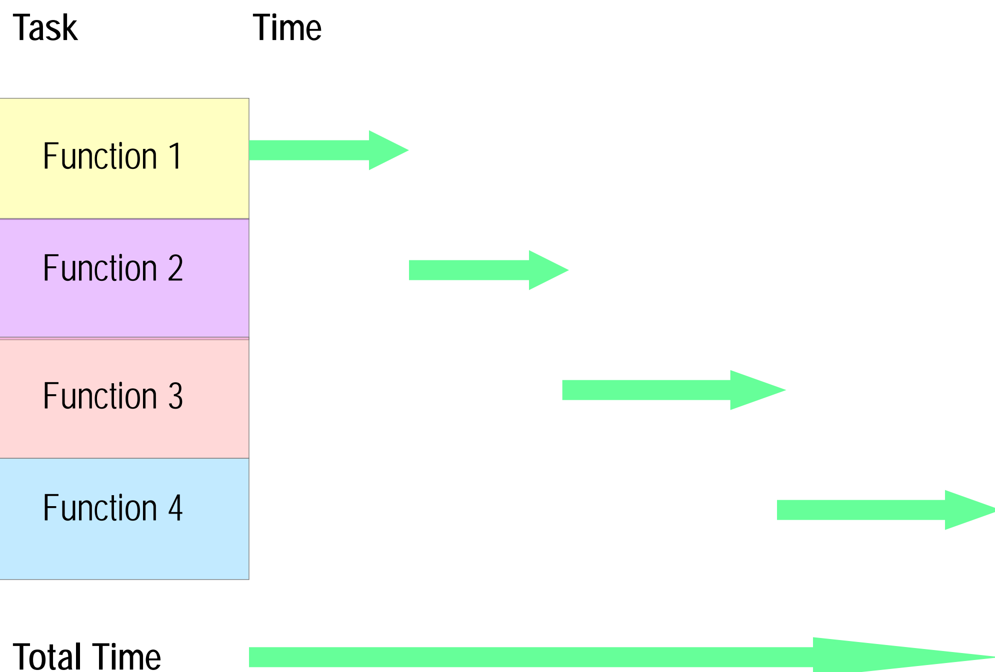
    Required data can be transferred at synchronization points.

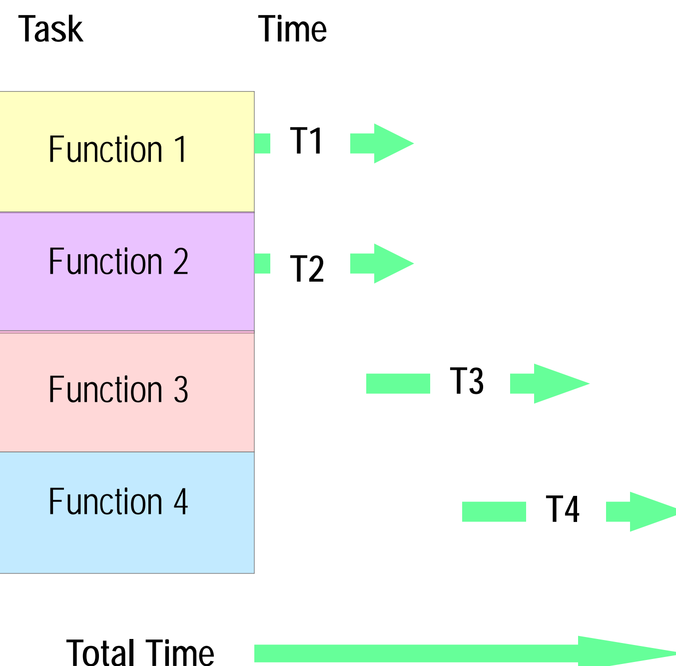
  - **Shared memory architectures:**

    Operations of reading from/writing to the memory can be synchronized among tasks.

(7) Load Balancing

- How to distribute work among all tasks so they are all kept busy all of the time?
- When barrier synchronization is used, the slowest task determines the performance.





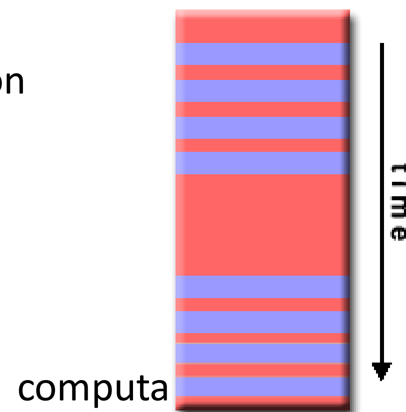
(7) Load Balancing

- **Ways to achieve Load Balancing**
  - **Equally partitioning the work each task receives.**
    - For operations that perform similar tasks to all data elements (ex: array, matrix, loop, …).
  - **Dynamic work assignment:**
    - Using a scheduler/task-pool.
    - Developing an algorithm that detects and handles imbalances.

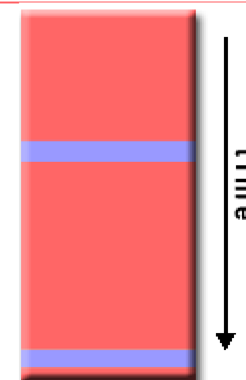
(8) Granularity

- **Computation : Communication ratio**

- **Fine-grain Parallelism:**
  - Small amounts of computation events are done between communication events.
  - Make load balancing easier.
  - High communication overhead.
  - Small opportunity to enhance performance.
  - If its too fine, communication overhead takes much longer than computa



- **Coarse-grain Parallelism:**
  - Large amounts of computational events are done between communication events.
  - Large opportunity to enhance performance.
  - Harder to do load balancing efficiently .



- **Which is better?**

Lecture Outline

- Parallel Computing Design Considerations
- Limits and Costs of Parallel Computing
- Parallel Computing Performance Analysis
- Examples of Problems Solved By Parallel Computing

Limits and costs of Parallel Computing (1/5)

- Complexity
- Scalability
- Portability
- Resource Requirements

Limits and costs of Parallel Computing (2/5)

- **Complexity**
  - Parallel programs are much more complex than corresponding serial ones because they have several tasks running at the same time and data flowing between them.
  - Costs of complexity are measured in all aspects of software development: Design, Coding, Debugging, Tuning, Maintenance.

Source: http://www.jamb.ca

  - When designing parallel programs, a programmer should stick to good software development practices to keep the program complexity to minimum.

Limits and costs of Parallel Computing (3/5)

Source: http://www.gensight.com/

- **Scalability**
  - What makes parallel programs scale?
  - Adding more processors?
  - Other factors
    - Software factors
      - Most of algorithms have limits to scalability. They reach a point where adding more resources causes performance to decrease.
      - Supporting subsystems and libraries can limit scalability.
    - Hardware factors
      - Memory-CPU bus bandwidth on an SMP machine.
      - Communications network bandwidth.
      - Amount of memory on machine(s).
      - Processor clock speed.

Limits and costs of Parallel Computing (4/5)

- **Portability**
  - Due to standardization, portability issues in parallel programs are not very serious as they used to be.
  - However…
    - All of the usual portability issues in serial programs apply to parallel ones.
    - Even in with standardized APIs, implementation differences that required code modifications exist.
    - Operating systems causes many portability issues.
    - Hardware architectures are highly variable which affect portability.

Source: http://www.stockphotopro.com/

Limits and costs of Parallel Computing (5/5)

- **Resource Requirements**
  - Parallel computing is used to decrease execution time, but to reach this, more CPU time is required.
    - EX: a parallel code that takes1 hour using 8 CPUs, would take 8 hours of CPU time when done serially.
  - Memory required for parallel code is much more than corresponding serial code.
    - Data replication.
    - Overheads caused by using support libraries and subsystems.
    - Communication overhead
  - Short running parallel programs, performance can be decreased.
    - Overhead caused by setting up parallel environment, create tasks, communication, execution time, …

http://www.verseone.com/image/servers.jpg

Lecture Outline

- Parallel Computing Design Considerations
- Limits and Costs of Parallel Computing
- Parallel Computing Performance Analysis
- Examples of Problems Solved By Parallel Computing

Spring 2010 ©

Parallel Computing Performance Analysis

Many ways are used to measure the performance of parallel computing programs, such as:

- **Amdahl's Law: helps decide whether a program merits penalization.**
- **Gustafson's Law: a way to evaluate the performance of a parallel program.**
- **Karp-Flatt Metric: helps deciding whether the principle barrier to the program speedup is the amount of inherently sequential code or parallel overhead.**
- **The Isoefficieny Metric: a way to evaluate the scalability of a parallel algorithm executing on a parallel computer.**

Amdahl's Law

- **Used to find the maximum expected improvement in an entire application when only one part of it is improved.**

- **An application could be bounded by one of the following main factors:**
  - Computation Time
  - Memory Access Time
  - Disk Access Time
  - Network Access Time

Amdahl's Law: Example

Consider an application that spends 70% of its time on computation, 10% accessing memory, 10% accessing disk, and 10% accessing network.

Type of work	Memory Access	Computation	Disk Access	Network Access
Time	10%	70%	10%	10%

- **What is the bounding factor to this application?**
- **What is the expected improvement percentage in its performance if:**
  - **The memory access speed is doubled?**
  - **The computational speed is doubled?**

Amdahl's Law: Example

Consider an application that spends 70% of its time on computation, 10% accessing memory, 10% accessing disk, and 10% accessing network.

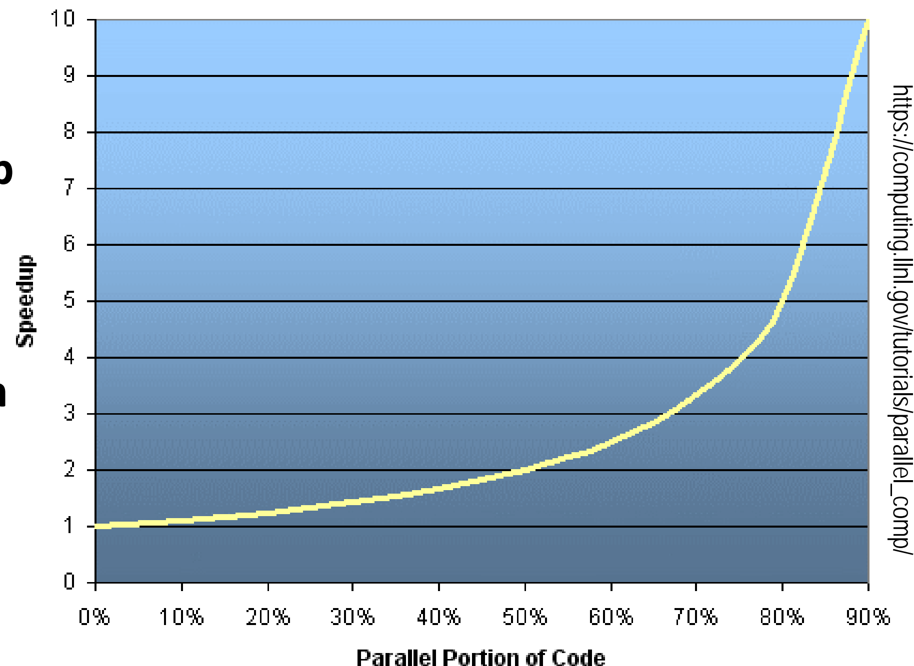
Type of work	Memory Access	Computation	Disk Access	Network Access
Time	10%	70%	10%	10%

- **What is the bounding factor to this application? Computation**
- **What is the expected improvement percentage in its performance if:**
  - **The memory access speed is doubled? 5%**
  - **The computational speed is doubled? 35%**

Spring 2010 ©

Using Amdahl's Law in Analyzing Performance of Parallel Computing

- Amdahl's Law is used to predict the maximum improvement in the speedup when using multiple processors in parallel.
- Speedup: how much a parallel program is faster than the corresponding serial one.



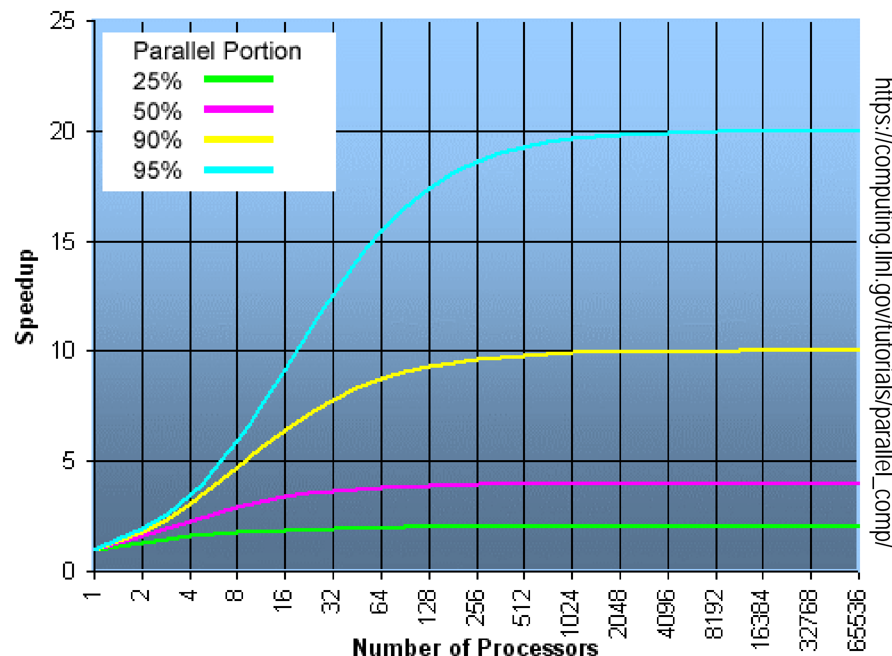
$$Speedup = \frac{1}{1-P}$$

- If P =0 (none of the code is parallelized) speedup = 1 (no speedup).
- If P = 1 (code is parallelized), speedup = ∞ (theoretically).
- If 1/2 of the code is parallelized, speedup = 2 (meaning the code will run twice as fast).

Using Amdahl's Law in Analyzing Performance of Parallel Computing

- When using multiple processors in parallel, program's speedup is limited by the time required by the sequential part of the program.

$$Speedup = \frac{1}{\frac{P}{N} + S}$$



- P = parallel fraction

  N = number of processors

  S = serial fraction

- Scalability Limitation:

  Problems in which parallel fraction increases as the problem size increases are more scalable than those with fixed parallel fraction.

Using Amdahl's in Parallel Computing: Example

Consider the an application that has 4 different functions: F1: taking 5% of the running time, F2: 10%, F3: 80%, and F4: 5%.

- **Parallelizing which part of the application would mostly improve the performance?**

- **Assume that parts: F1, F3, and 4 can all be parallelized, but F2 must be done serially. What is the best performance speed up that could be reached by parallelizing those parts?**

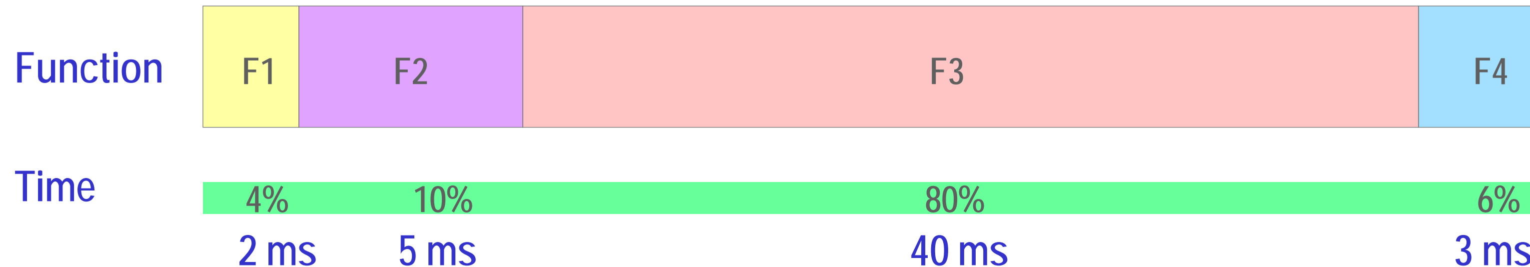
Using Amdahl's in Parallel Computing: Example

Consider the an application that has 4 different functions: F1: taking 5% of the running time, F2: 10%, F3: 80%, and F4: 5%.

- **Parallelizing which part of the application would mostly improve the performance?**
F3

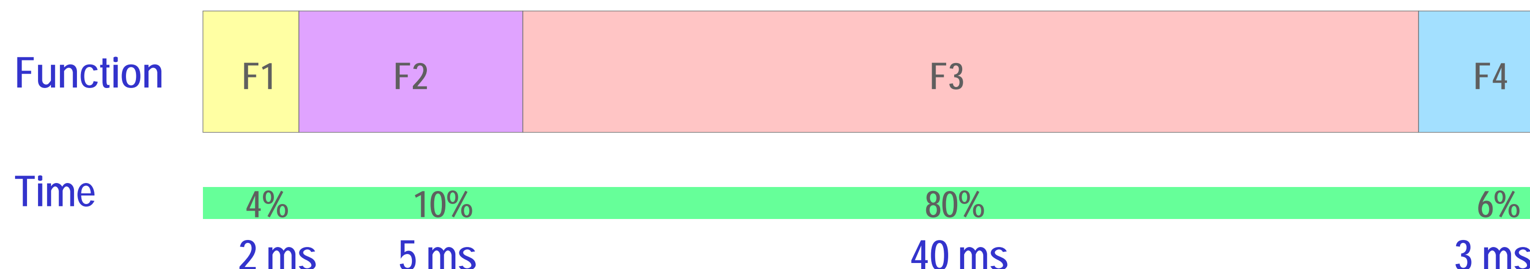
- **Assume that parts: F1, F3, and 4 can all be parallelized, but F2 must be done serially. What is the best performance speed up that could be reached by parallelizing those part? 10 times faster**

Using Amdahl's in Parallel Computing: Example



- Suppose that running the whole application requires 50 ms. From the last question, what is the best running time that we can reach?
- Can the application run any faster than this? Why?

Using Amdahl's in Parallel Computing: Example



- **Suppose that running the whole application requires 50 ms. From the last question, what is the best running time that we can reach?**

  Max speedup is 10, best running time is 5 ms

- **Can the application run any faster than this? Why?**

  No, because no matter how the parallel part is fast (fastest is t=0), we can not decrease the time required to run the serial part (5 seconds).

Lecture Outline

- Parallel Computing Design Considerations
- Limits and Costs of Parallel Computing
- Parallel Computing Performance Analysis
- **Examples of Problems Solved By Parallel Computing**

Parallelization Examples

- **Array Processing**
  - Apply an operation or function on each element of the array.
- **PI Calculation**
  - Discussed previously in the Demo!
- **Can you think of more examples?**

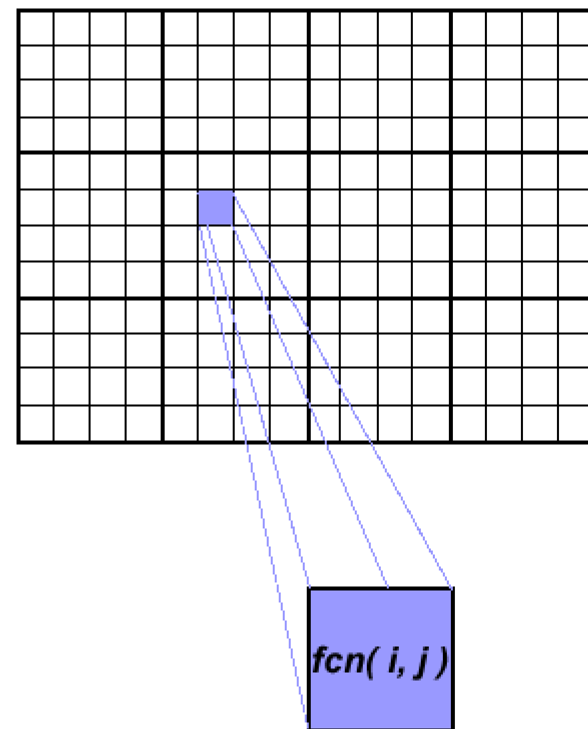
Parallelization Examples

- **Array Processing**
  - Apply the same operation or function on each element of the array.
  - Independent, no communication is needed.
  - Embarrassingly parallel!
  - Divide the array into smaller chunks and distribute them over processors so the operations can be done in parallel.

Example: Array Processing

- **Serial Code:**

```
do j = 1,n
    do i = 1,n
        a(i,j) = fcn(i,j)
    end do
end do
```

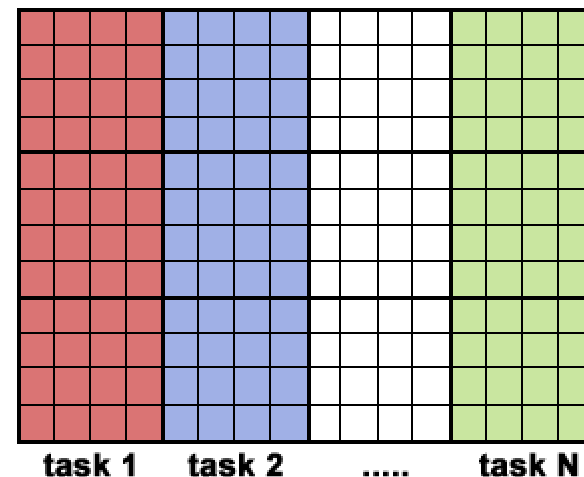


But, this is computationally intensive!

Example: Array Processing

- Parallel Code:

```
find out if I am MASTER or WORKER

if I am MASTER
        initialize the array
        send each WORKER info on part of array it owns
        send each WORKER its portion of initial array
        receive from each WORKER results

else if I am WORKER
        receive from MASTER info on part of array I own
         receive from MASTER my portion of initial array

        // calculate my portion of array
        do j = my first column,my last column
                do i = 1,n a(i,j) = fcn(i,j)
                 end do
        end do
        send MASTER results
endif
```

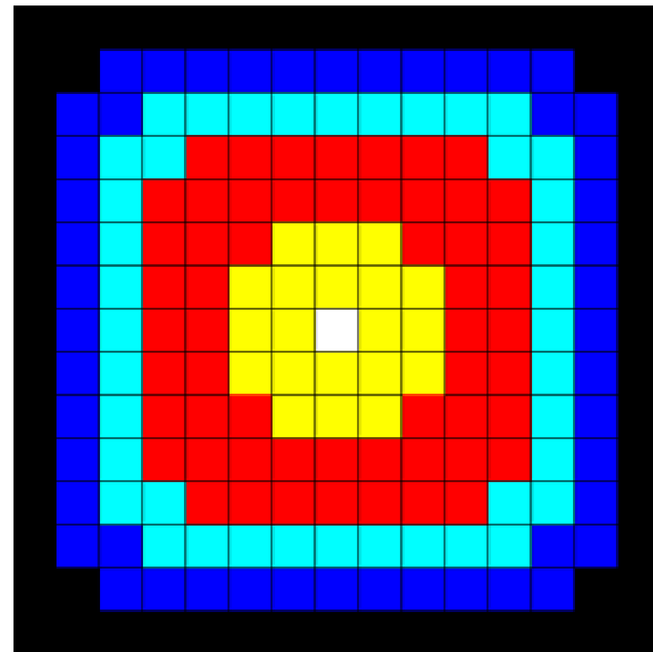


Array is divided into chunks, each processor own a chunk, and execute the portion of the loop corresponding to it.

Parallelization Examples: Simple Heat Equation

- The Heat Equation describes the change in temperature in a given region over time, given the initial temperature distribution and the boundary conditions.

- To solve the equation on a 2D region, a finite differencing scheme is used. 2D array is used to represent the temperature distribution. So, the initial array is used to calculate the array representing the change in the distribution.



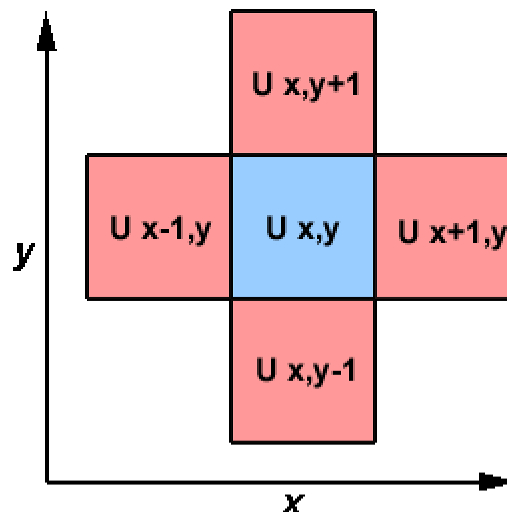
- The initial temperature is zero on the boundaries and high in the middle. Boundary temperature is held at zero.

- Calculating an element depends on neighbor element values. So, communication is required!

Parallelization Examples: Simple Heat Equation

- **To calculate one element $U_{x,y}$:**

$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$

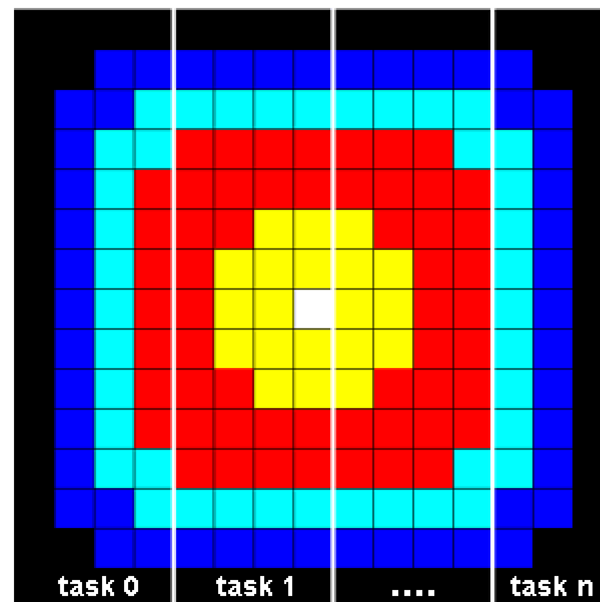


- **Serial Code:**

```
do iy = 2, ny - 1

     do ix = 2, nx - 1

             u2(ix, iy) = u1(ix, iy)
                           + cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy))
                           + cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))
     end do

end do
```

Parallelization Examples: Simple Heat Equation

- **Parallel Way:**
  - The array in divided into chunks, each is done through a task.
  - Data dependencies are determined because we have 2 types of elements:
    - interior elements belonging to a task are independent of other tasks.
    - border elements are dependent on the neighbor elements, so communication is required.



  - Master process sends initial info to worker processes, checks whether convergence is reached, and collects results.
  - Worker process calculates solutions and communicate as neighbor processes when it's required.

Parallelization Examples: Simple Heat Equation

- **Parallel Code:**

```
find out if I am MASTER or WORKER

if I am MASTER
      initialize array
      send each WORKER starting info and subarray

      do until all WORKERS converge
         gather from all WORKERS convergence data
         broadcast to all WORKERS convergence signal
      end do

      receive results from each WORKER

else if I am WORKER
      receive from MASTER starting info and subarray

       do until solution converged
         update time
         send neighbors my border info
         receive from neighbors their border info
         update my portion of solution array
         determine if my solution has converged
             send MASTER convergence data
          receive from MASTER convergence signal
      end do

      send MASTER results
endif
```

References

- http://en.wikipedia.org/wiki/Parallel_programming_model
- http://www.buyya.com/cluster/v2chap1.pdf
- https://computing.llnl.gov/tutorials/parallel_comp/
- http://www.acm.org/crossroads/xrds8-3/programming.html
- http://researchcomp.stanford.edu/hpc/archives/HPCparallel.pdf
- http://www.mcs.anl.gov/~itf/dbpp/text/node9.html
- http://en.wikipedia.org/wiki/Parallel_computing
- http://www.azalisaudi.com/para/Para-Week2-TypesOfPara.pdf