CS-251: Parallel and Distributed Computing

Lecture 04 – Hardware Architectures

Shared Memory

Dr. Zeeshan Rafi

PhD MIS, MPhil IT, BS Software Engineering
Former Software Engineer, Database Administrator
System Analyst, Big Data Analyst
Member Turkish Scientific & Technological Research Council

Department of Computing and Information Technology

Istanbul University, TR

KHAS University, TR

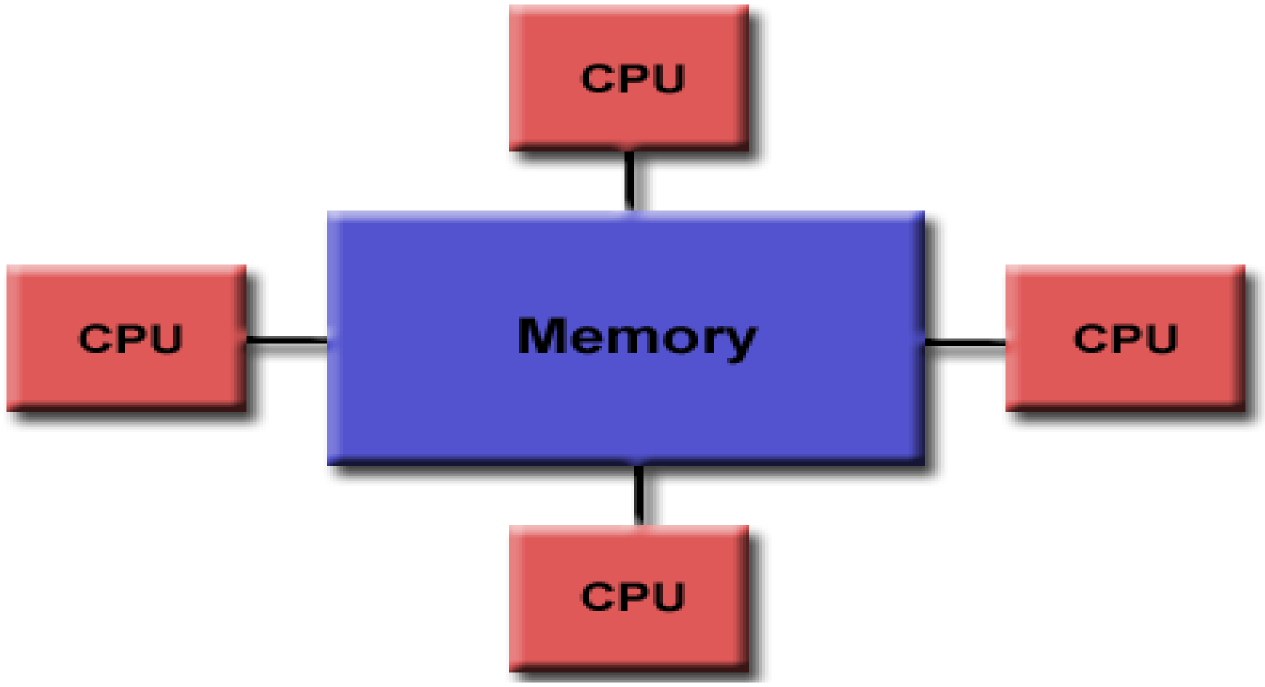
University of Gujrat, PK

GCF University, Pk

Shared Memory

Shared Memory

- All processors access all memory as global address space
  - Multiple processors can operate independently but share the same memory resources
  - Changes in a memory location effected by one processor are visible to all other processors



Shared Memory (UMA)

Shared Memory

- Shared memory machines are classified

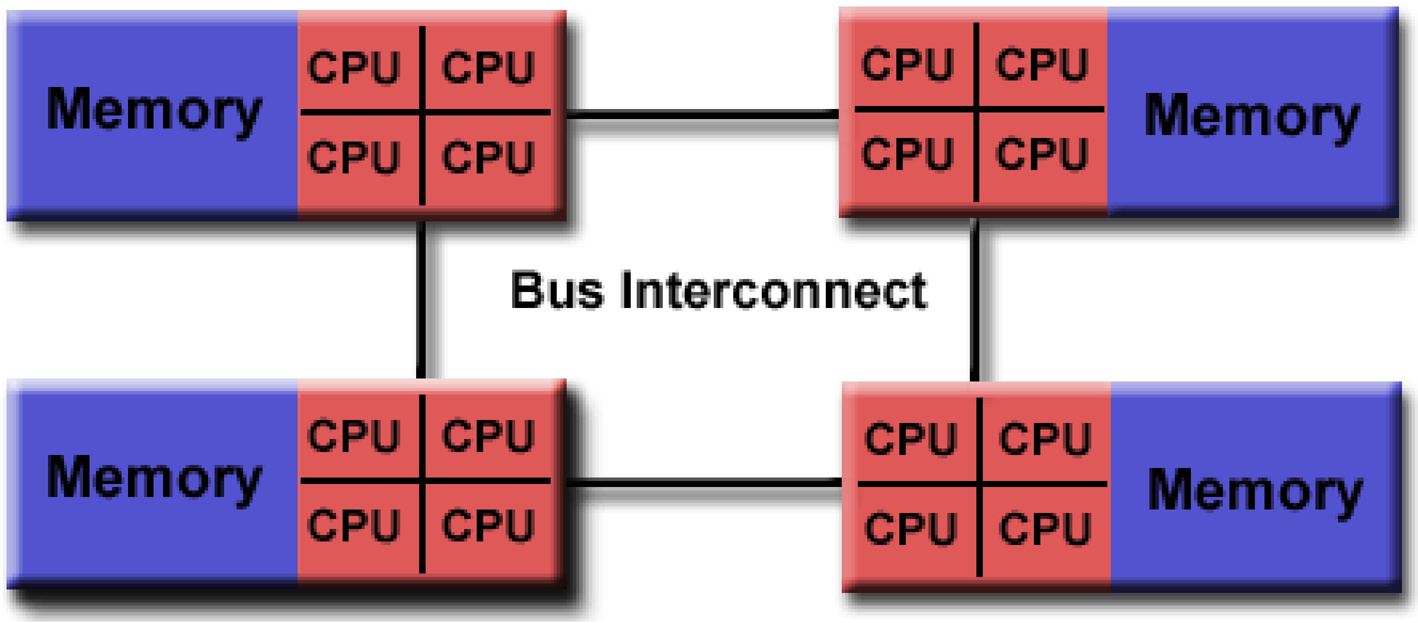
  as *UMA* and *NUMA*, based upon memory access times

  - Uniform Memory Access (UMA)
    - Most commonly represented today by Symmetric Multiprocessor (SMP) machines
    - Identical processors
    - Equal access and access times to memory
    - Sometimes called CC-UMA - Cache Coherent UMA
      - Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level

Shared Memory

– Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA



Shared Memory (NUMA)

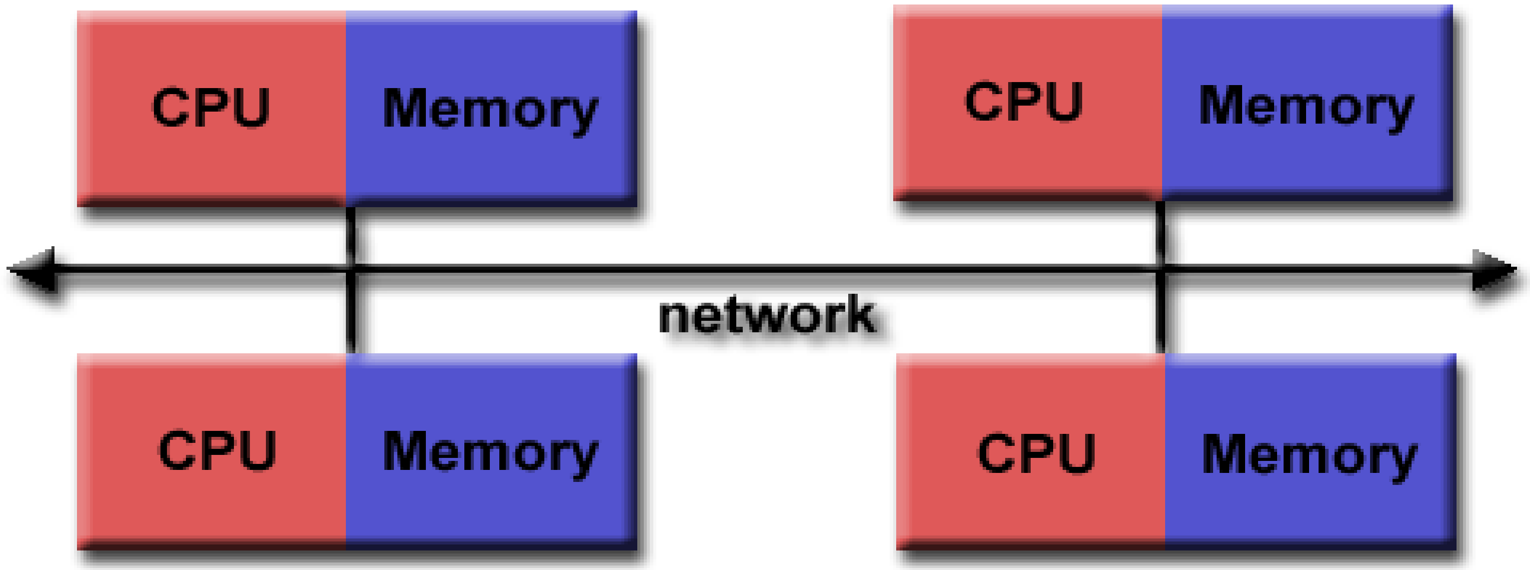
Shared Memory

- Advantages
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

- Disadvantages
  - Primary disadvantage is the lack of scalability between memory and CPUs
    - Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management
  - Programmer responsibility for synchronization constructs that ensure "correct" access of global memory

Distributed Memory

- Processors have their own local memory
  - Changes to processor’s local memory have no effect on the memory of other processors
  - When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated
  - Synchronization between tasks is likewise the programmer's responsibility
  - The network "fabric" used for data transfer varies widely, though it



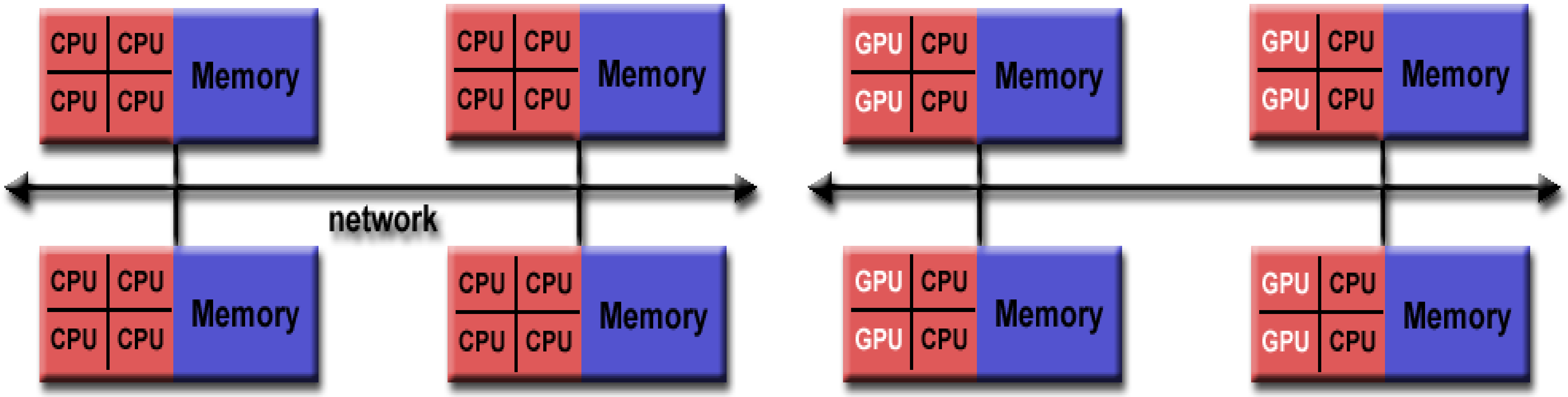
Distributed Memory

- Advantages
  - Memory is scalable with the number of processors.
    - Increase the number of processors and the size of memory increases proportionately
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking

- Disadvantages
  - The programmer is responsible for many of the details associated with data communication between processors.
  - Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures
  - The shared memory component can be a shared memory machine or graphics processing units
  - The distributed memory component is the networking of multiple shared memory or GPU machines



Hybrid Distributed-Shared Memory

- Advantages and Disadvantages
  - Whatever is common to both shared and distributed memory architectures
  - Increased scalability is an important advantage
  - Increased programmer complexity is an important disadvantage

Parallel Programming Models

Parallel Programming Model

- Programming model provides an abstract view of computing system
  - Abstraction above hardware and memory architectures
  - Value of a programming model is usually judged on its generality
    - how well a range of different problems can be expressed and
    - how well they execute on a range of different architectures
  - The implementation of a programming model can take several forms such as libraries invoked from traditional sequential languages, language extensions, or complete new execution models

Parallel Programming Model

- Parallel programming models in common use:
  - Shared Memory (without threads)
  - Threads
  - Distributed Memory / Message Passing
  - Data Parallel
  - Hybrid
  - Single Program Multiple Data (SPMD)
  - Multiple Program Multiple Data (MPMD)

- These models are NOT specific to a particular type of machine or memory architecture
  - Any of these models can be implemented on any underlying hardware

Parallel Programming Model

- SHARED memory model on a DISTRIBUTED memory machine
  - Machine memory was physically distributed across networked machines, but appeared to the user as a single shared memory (global address space).
  - This approach is referred to as virtual shared memory

- DISTRIBUTED memory model on a SHARED memory machine
  - The SGI Origin 2000 employed the CC-NUMA type of shared memory architecture, where every task has direct access to global address space spread across all machines.
  - However, the ability to send and receive messages using MPI, as is commonly done over a network of distributed memory machines, was implemented and commonly used

Shared Memory Model - Without Threads

- Tasks share a common address space
  - Efficient means of passing data between programs
  - Various mechanisms such as locks / semaphores may be used to control access to the shared memory
  - Programmer's point of view
    - The notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks
    - Program development can often be simplified

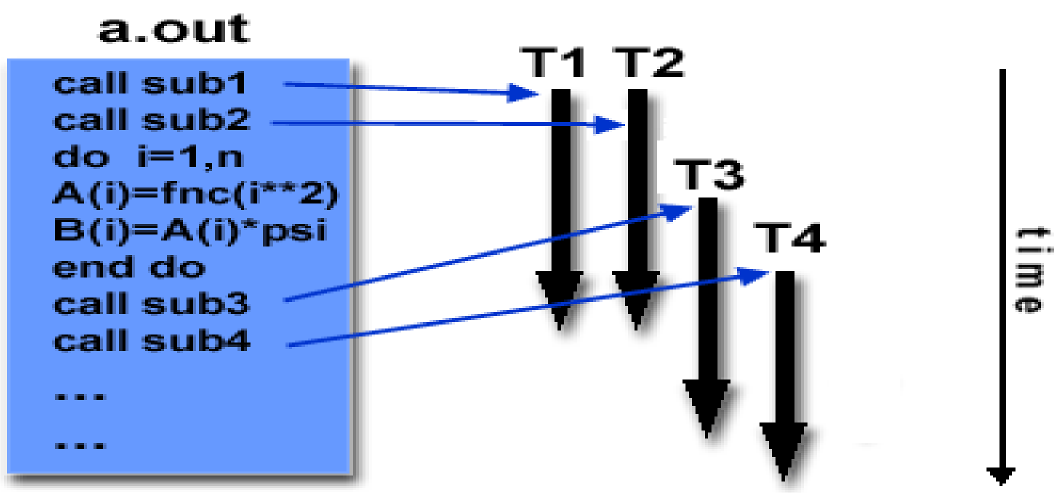
- Disadvantage in terms of performance
  - It becomes more difficult to understand and manage data locality:
    - Keeping data local to the processor that works on it conserves memory accesses, cache refreshes and bus traffic that occurs when multiple processors use the same data

Shared Memory Model - Without Threads

- Implementations
  - Native compilers or hardware translate user program variables into actual memory addresses, which are global
    - On stand-alone shared memory machines, this is straightforward.
    - On distributed shared memory machines, memory is physically distributed across a network of machines, but made global through specialized hardware and software

Threads Model

- Type of shared memory programming model
  - A single "heavy weight" process can have multiple "light weight", concurrent execution paths
  - Main program a.out is scheduled by native OS
  - a.out loads and acquires all of the necessary system and user resources to run. This is the "heavy weight" process
  - a.out performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently



Threads Model

- Each thread has local data, but also, shares the entire resources of a.out

- This saves the overhead associated with replicating a program's resources for each thread ("light weight"). Each thread also benefits from a global memory view because it shares the memory space of a.out

- Threads communicate with each other through global memory (updating address locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time

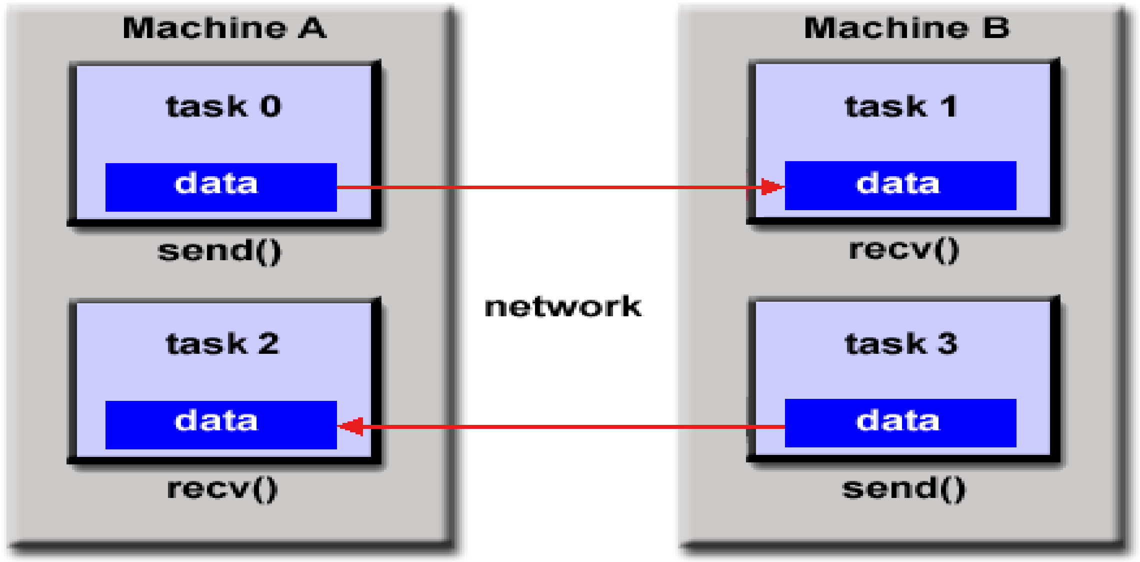
- Threads can come and go, but a.out remains present to provide the necessary shared resources until the application has completed

Threads Model

- Implementations
  - *POSIX Threads*
    - Library based; requires parallel coding
    - C Language only
    - Commonly referred to as Pthreads.
    - Most hardware vendors now offer Pthreads in addition to their proprietary threads implementations.
    - Very explicit parallelism; requires significant programmer attention to detail.
  - *OpenMP*
    - Compiler directive based; can use serial code
    - Portable / multi-platform, including Unix and Windows platforms
    - Available in C/C++ and Fortran implementations
    - Can be very easy and simple to use

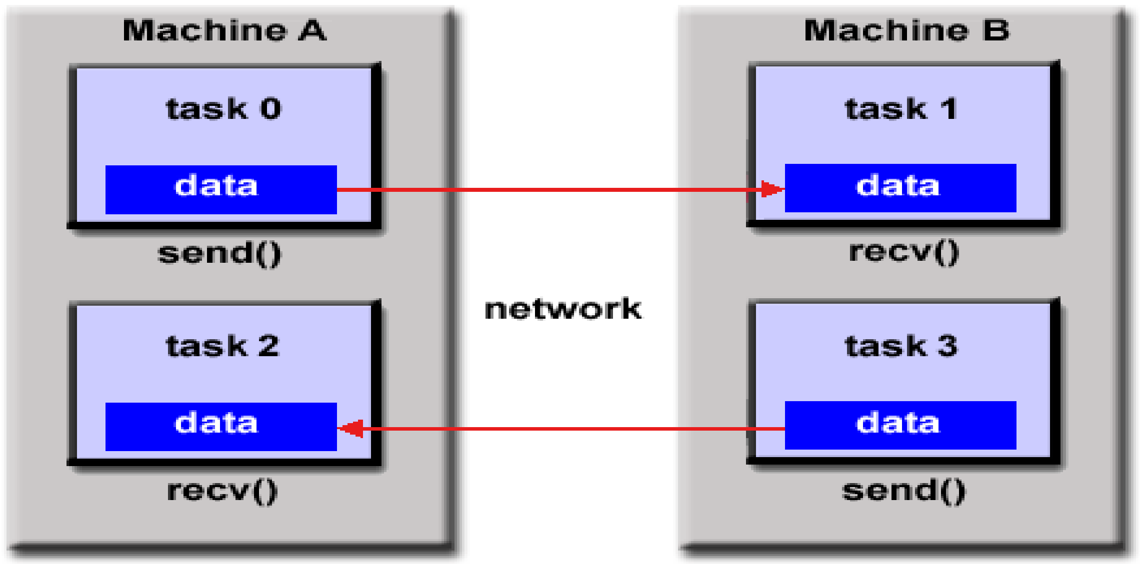
Distributed Memory / Message Passing Model

- A set of tasks that use their own local memory during computation
  - Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
  - Tasks exchange data through communications by sending and receiving messages.
  - Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.



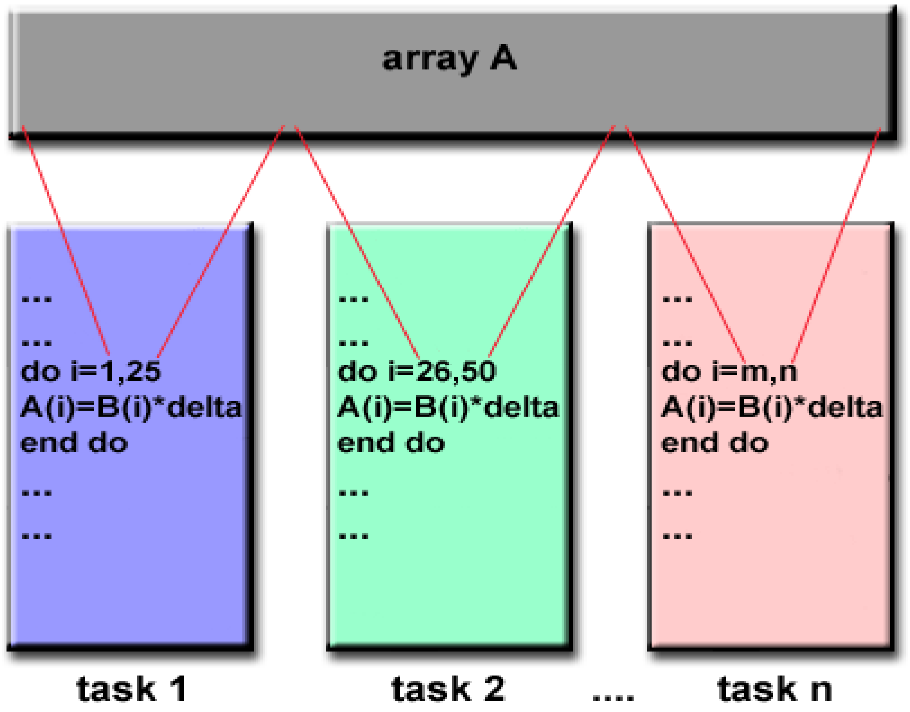
Distributed Memory / Message Passing Model

- Implementations
  - From a programming perspective, message passing implementations usually comprise a library of subroutines
  - Calls to these subroutines are imbedded in source code
  - MPI specifications are available on the web at http://www.mpi-forum.org/docs/
  - MPI implementations exist for virtually all popular parallel computing platforms



Data Parallel Model

- Address space is treated globally
- A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure
  - On shared memory architectures, all tasks may have access to the data structure through global memory
  - On distributed memory architectures the data structure is split up and resides as "chunks" in the local memory of each task

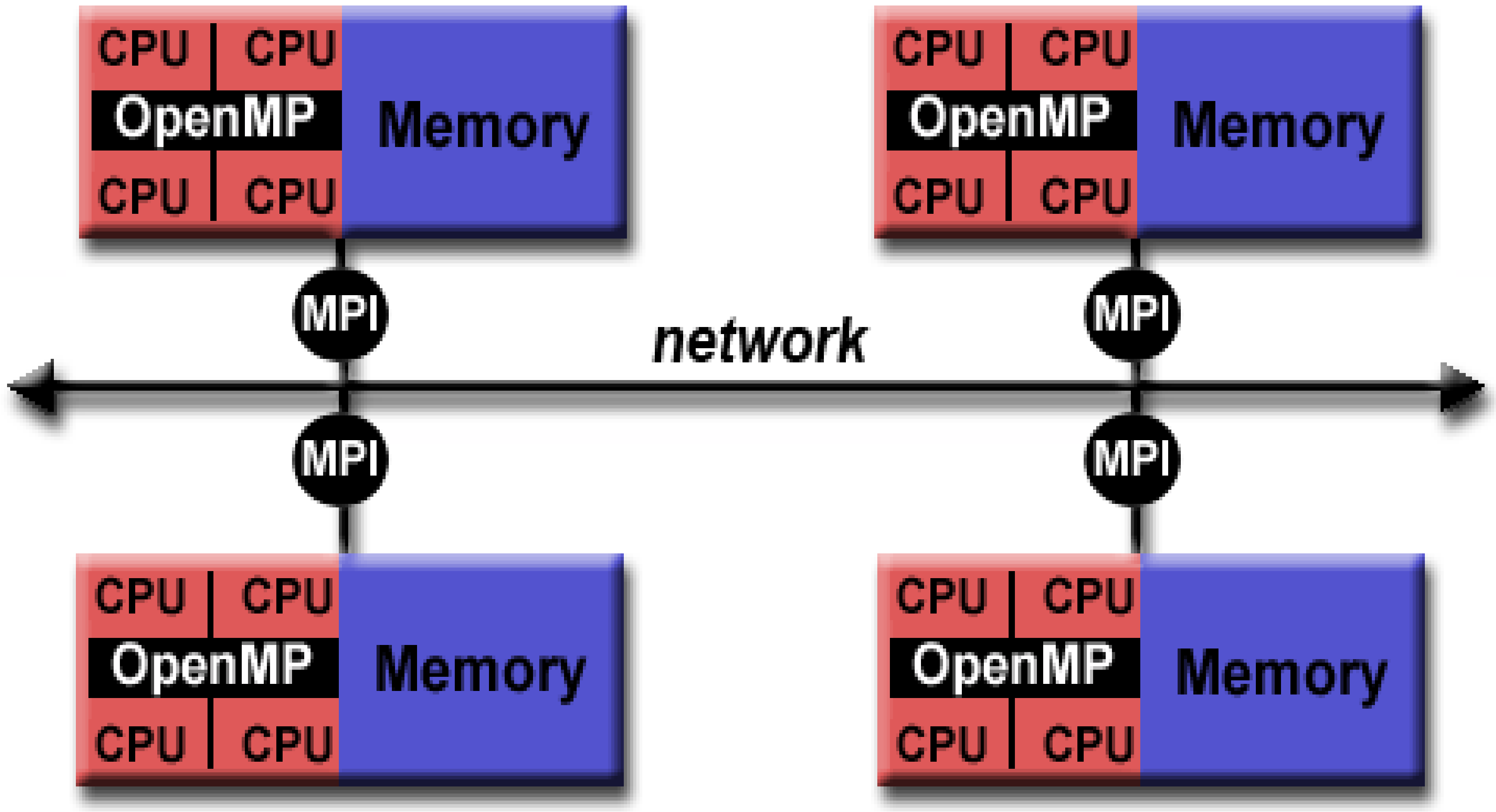


Data Parallel Model

- Implementations
  - Unified Parallel C (UPC): an extension to the C programming language for SPMD parallel programming. Compiler dependent. More information: http://upc.lbl.gov/
  - Global Arrays: provides a shared memory style programming environment in the context of distributed array data structures. Public domain library with C and Fortran77 bindings. More information: http://www.emsl.pnl.gov/docs/global/
  - X10: a PGAS based parallel programming language being developed by IBM at the Thomas J. Watson Research Center. More information: http://x10-lang.org/

Hybrid Model

- A hybrid model combines more than one of the previously described programming models

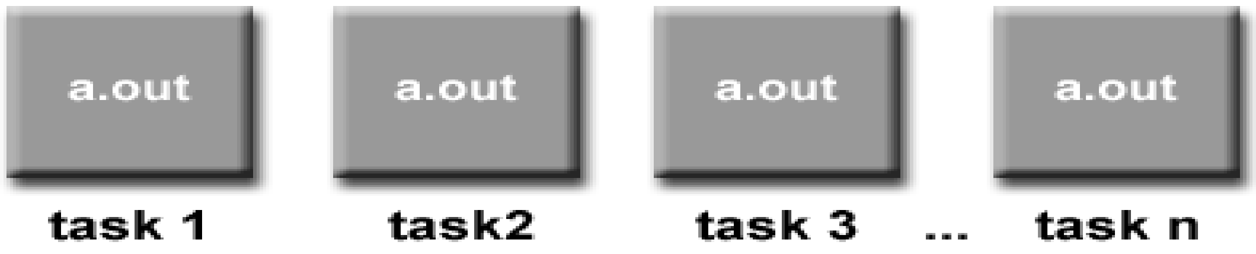


Single Program Multiple Data

- High level programming model that can be built upon any combination of the previously mentioned parallel programming models

- SINGLE PROGRAM: All tasks execute their copy of the same program simultaneously.
  - This program can be threads, message passing, data parallel or hybrid.

- MULTIPLE DATA: All tasks may use different data



Multiple Program Multiple Data

- High level programming model that can be built upon any combination of the previously mentioned parallel programming models

- MULTIPLE PROGRAM: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid.

- MULTIPLE DATA: All tasks may use different data