III. Architecture - Logical and physical organization

March 6th, 2009

Content

- Logical organization
  - Flynn taxonomy
  - SIMD and MIMD
  - Communication
    - shared data space
    - exchanging messages
- Physical organization
  - Historical context
  - Shared memory versus distributed memory

Why? How?

- There are dozens of different parallel architectures, among them:
  - networks of workstations (NoW),
  - clusters of off-the shelf PCs,
  - massively parallel supercomputers,
  - tightly coupled symmetric multiprocessors,
  - multiprocessor workstations

  Etc
- Some architectures provide better performance than others.
  - The notion of performance of a parallel system is quite theoretical since it also depends on the kind of application which is used on it.
- **Logical organization** refers to a programmer's view of the platform
- **Physical organization** refers to the actual hardware organization of the platform.

Logical organization

- Parallel computers can be divided into two main categories:
  1. **Control-flow** parallel computers (focus exclusively on this ones!!!)
     - based on the same principles as the sequential or von Neumann computer,
     - multiple instructions can be executed at any given time.
  2. **Data-flow** parallel computers,
     - sometimes referred to as “non-von Neumann,”
     - completely different: no pointer to active instruction(s) or a locus of control.
     - control is distributed: availability of operands triggers the activation of instructs.
- Critical components of parallel comp.from a programmer's perspective:
  1. ways of expressing parallel tasks - referred to as the **control structure**
  2. mechanisms for specifying interact.betw.tasks-**communication model**
- Parallel tasks can be specified at various levels of granularity:
  1. One extreme, each progr.in a set of progs is viewed as 1 parallel task.
  2. Other extreme, individual instrs within a progr.are viewed as par. tasks.
  3. Between these extremes: a range of models for specifying the control structure of progrs& the corresponding architectural support for them.

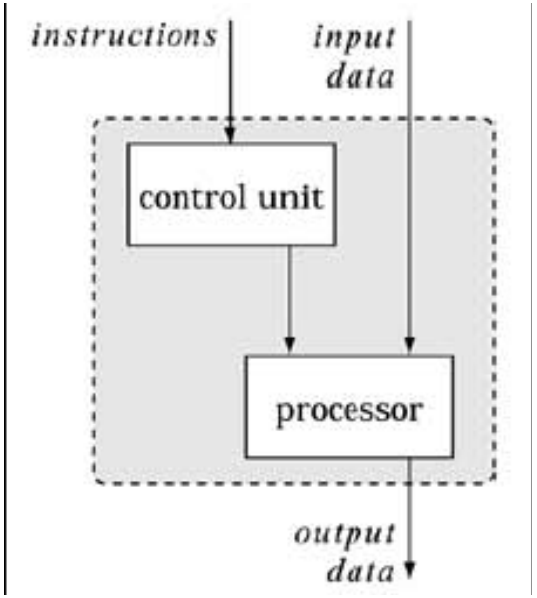
Flynn's taxonomy

Control Structure of Parallel Platforms

- M. J. Flynn (1972) introduced a system for the categorisation of the system architectures of computers
  - It is still valid today and cited in every book about parallel computing.
  - It is far the most common way to characterize parallel syst. archs.
- Categorizes all computers according to the
  - no. of instruction streams and data streams they have,
  - where a stream is a sequence of instructions or data on which a computer operates.
- 4 classes of computers based on the no. of
  - instruction streams (single or multiple) and
  - data streams (single or multiple)
  - ⇒ abbreviations SISD, SIMD, MISD, and MIMD (pronounced “sis-dee,” “simdee,” and so forth)
- This classification is based upon the relationship between the instructions and the manipulated data.
  - ⇒ it is quite general and does not reflect all the aspects of parallel syst.

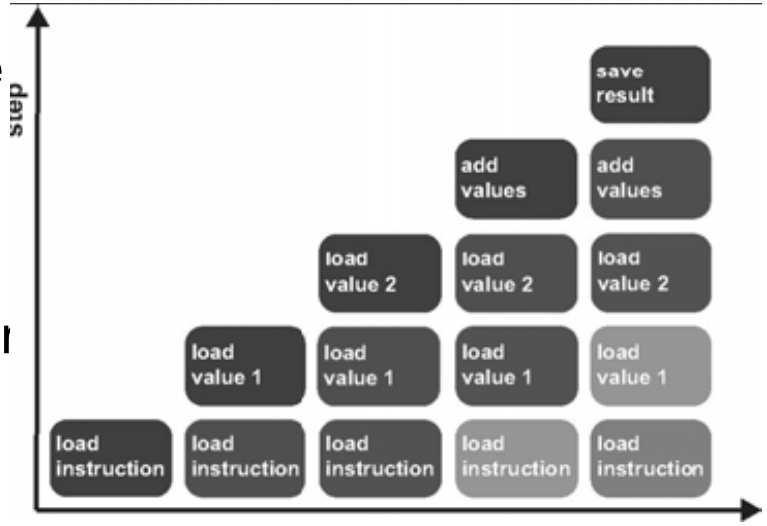
Single Instruction, Single Data (SISD)

- One stream of instructions processes a single stream of data
- This is the common von Neumann model used in virtually all single processor computers.
- The most simple type of computer performs one instruction per cycle
  - such as reading from memory, addition of two values
  - only one set of data or operand
- Such a system is called a ***scalar computer***.
- Example: Adding two values in 5 steps => adding *n* values in 5*n* steps



SISD and pipelines

- In reality each of the steps is actually composed of several sub-steps, increasing the no. cycles required for one summation even more.
- The solution to this inefficient use of processing power is ***pipelining:***
  - If there is one functional unit available for each of the five steps required, the addition still requires five cycles.
  - The advantage is that with all functional units being busy at the same time, one result is produced every cycle.
- For the summation of *n* pairs of numbers, only (*n*−1)+5 cycles are then required.
- Fig. shows the summation in a pipeline.



Summation of *n* values

SISD and superscalar

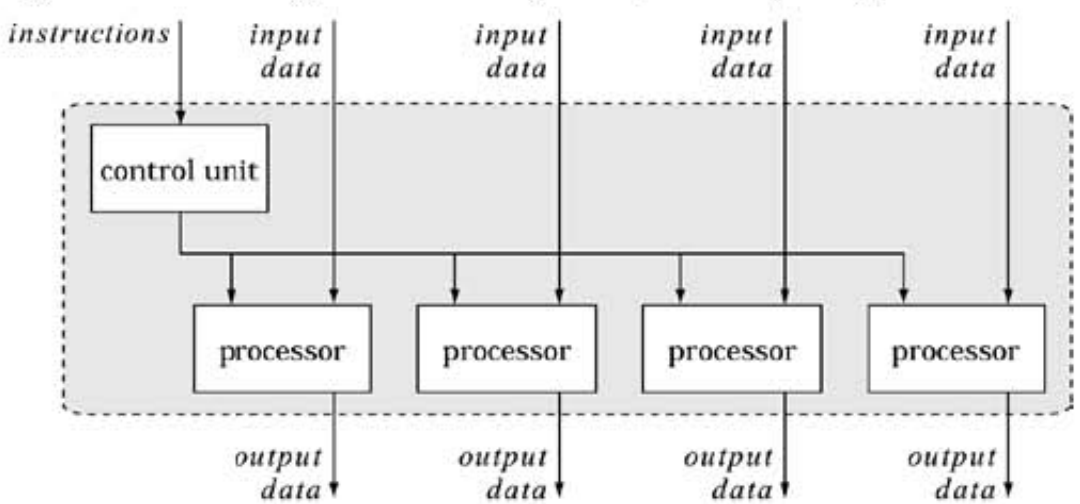
- As the execution of instructions usually takes more than five steps, pipelines are made longer in real processors.
- Long pipelines generate a new problem:
  - If there is a branching event (*if* -statement),
    - the pipeline has to be emptied and filled again
    - there is a no. cycles eq. to the pipeline length until results are again delivered.
  - Compilers and CPUs also try to minimize this problem by “guessing” the outcome (branch prediction).
- The power of a proc. can be increased by combining several pipelines.
  - This is then called a ***superscalar*** processor.
  - Fixed-point and logical calculations (performed in *ALU* - Arithmetic/ Logical Unit) are usually separated from floating-point math (done by *FPU* – Floating Point Unit).
  - The FPU is commonly subdivided in a unit for + and one for x.
  - These units may be present several times, & some procs have additional functional units for / & the computation of square roots.
  - To actually gain a benefit from having several pipelines, these have to be used at the same time (“pipeline of pipes”)

Multiple Instruction, Single Data (MISD)

- No well known systs fit this designation.
- Such a computer is neither theoretically nor practically possible
- It is mentioned for the sake of completeness.
- Some authors are viewing MISD as generalized pipelines in which each stage performs a relatively complex operation
  - as opposed to ordinary pipelines found in modern processors where eacl stage does a very simple instruction-level operation.
  - a single data stream enters the machine consisting of *p* processors
  - various transformations are performed on each data item before it is passed on to the next processor(s).
  - Successive data items can go through different transformations
    - data-dependent conditional statements in the instruction streams (control-driven)
    - special control tags carried along with the data (data-driven).
  - The MISD organization can thus be viewed as a flexible or high-level pipeline with multiple paths and programmable stages.
    - The key difference between the above pipeline and a MISD architecture is that the floating-point pipeline stages are not programmable.

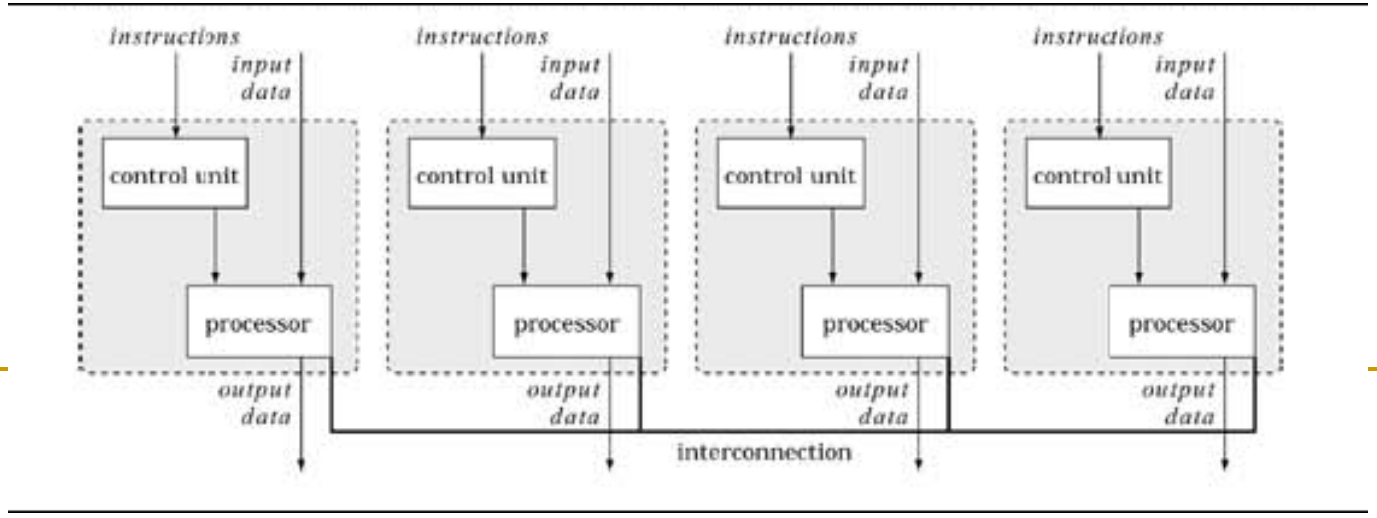
Single Instruction, Multiple Data (SIMD)

- A single instruction stream is concurrently broadcast to multiple processors, each with its own data stream.
- Commercial systems: Thinking Machines, MasPar, CPP DAP Gamma II, Quadrics Apemille
- Typically deployed in specialized applications, such as digital signal processing, that are suited to fine grained parallelism and require little inter process communication.
- Vector processors, which operate on vector data in a pipelined fashion, can also be categorized as SIMD.
- Exploiting this parallelism is usually done by the compiler.



Multiple Instruction, Multiple Data (MIMD)

- Each PE has its own stream of instructions operating on its own data.
- Is the most general of the architectures: each of the other cases can be mapped onto the MIMD architecture.
- The vast majority of modern parallel systems fit into this category.
- On a MIMD computer each of the parallel processing units executes operations independently of each other, subject to synchronization through appropriate message passing at specified time intervals.
- Both parallel data distribution as well as the message passing and synchronization are under user control.
- Ex: Intel Gamma, Delta Touchstone, Cray C-90, IBM SP2 (1990s).



SIMD case: Vector computer

- A computer that performs one instruction on several data sets is called a ***vector*** computer.
- Vector computers work just like the pipelined scalar computer
- The difference is that instead of processing single values, vectors of data are processed in one cycle.
- The number of values in a vector is limited by the CPU design.
  - A vector processor than can simultaneously work with 64 vector elements can also generate 64 results per cycle
- To actually use the theoretically possible performance of a vector computer, the calculations themselves need to be vectorized.
- Example:
  - Consider the following code segment that adds 2 vectors:

    for (i = 0; i < 1000; i++) c[i] = a[i] + b[i];
  - Various iterations of the loop are independent of each other; i.e., c[0] = a[0] + b[0]; c[1] = a[1] + b[1];, etc., can all be executed independently of each other.
  - If there is a mechanism for executing the same instruction, in this case add on all the processors with appropriate data, we could execute this loop much faster.

SIMD and MIMD

Vectorisation is not dead

- In practice:
  - Vector computers used to be very common in the field of HPC, as they allowed very high performance even at lower CPU clock speeds.
  - They have begun to slowly disappear.
    - Vector processors are very complex and thus expensive, and perform poorly with non-vectorisable problems.
    - Today's scalar processors are much cheaper and achieve higher CPU clock speeds.
- With the Pentium III, Intel introduced *SSE* (Streaming SIMD Extensions), which is a set of vector instructions.
  - In certain applications, such as video encoding, the use of these vector instructions can offer quite impressive performance increases.
  - More vector instructions were added with SSE2 (Pentium 4) and SSE3 (Pentium 4 Prescott).

SIMD and the control unit

- A single control unit dispatches instructions to each processing unit.
- The same instruction is executed synchronously by all processing units.
- Examples of systems:
  - Old: Illiac IV, MPP, DAP, CM-2, and MasPar MP-1 Thinking Machines CM-2 or the NCUBE Inc. computers of the 1980s
  - Modern: MMX units in Intel processors and DSP chips such as the Sharc, Intel Pentium processor with SSE
  - Each processor performs the same arithmetic operation (or stays idle) during each computer clock, as controlled by a central control unit.
- High-level languages are used, and computation and communication among processors are synchronized implicitly at every clock period.
- These architectural enhancements rely on the highly structured (regular) nature of the underlying computations, for example in image processing and graphics, to deliver improved performance.

SIMD and array processors

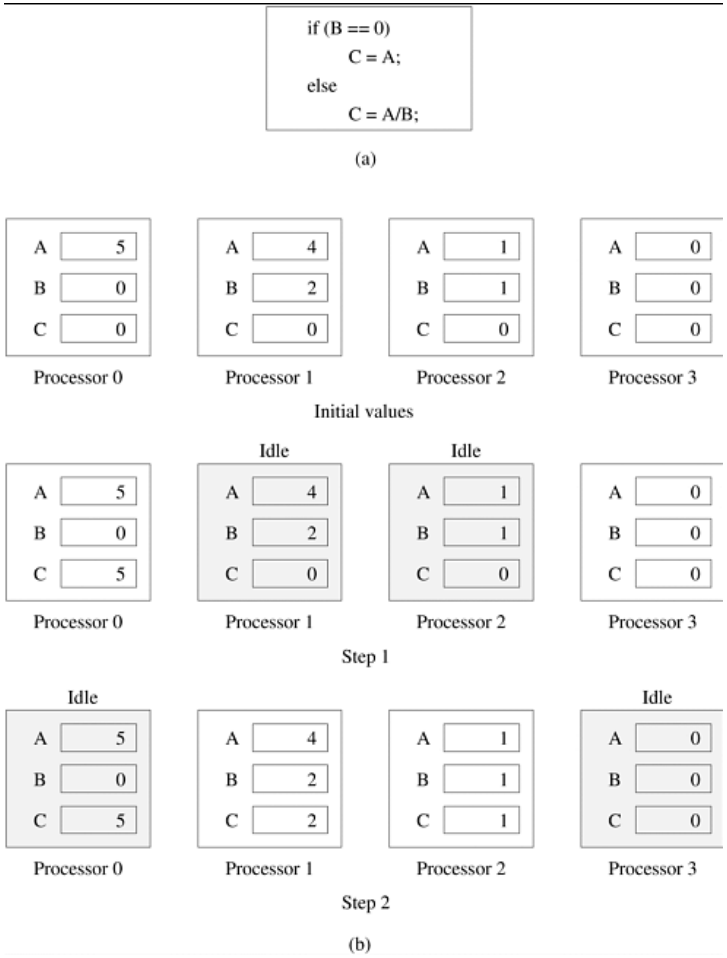
- If SIMD's processors are directed by instructions issued from a central control unit, are characterized as **array processors**.
  - A relatively large number of relatively weak processors, each associated with a relatively small memory.
  - Processors are combined into a matrix-like topology, hence the popular name of this category - processor arrays.
  - Program compilation&array processing manag.takes place in control unit.
  - Each processor performs operations on separate data streams;
  - All processors may perform the same operation, or some of them may skip a given operation or a sequence of operations.
- Vendors were ICL, MasPar and Thinking Machines.
- Currently, SIMDs are no longer produced for the mainstream of par.com
- Advantage: the processors work synchronously, which enables relatively easy program tracing and debugging.
- Disadvantages:
  - Relatively difficult to use them for unstructured problems

SIMD is a data parallel architecture

- The key characteristic of the programming model is that operations can be performed in parallel on each element of a large regular data structure, such as an array or matrix.
- The program is logically a single thread of control, carrying out a sequence of either sequential or parallel steps.

- While the SIMD concept works well for structured computations on parallel data structures such as arrays, often it is necessary to selectively turn off operations on certain data items.
  - Most SIMD programming paradigms allow for an "activity mask".
    - This is a binary mask associated with each data item and operation that specifies whether it should participate in the operation or not.
    - Conditional execution can be detrimental to the performance of SIMD processors and therefore must be used with care.

Conditional statements in SIMD



- The conditional statement in Fig. is executed in two steps:

1. In the first step, all processors that have B equal to zero execute the instruction C =A. All other processors are idle.
2. In the second step, the 'else' part of the instruction (C =A/B) is executed. The processors that were active in the first step now become idle.

- Not highly efficient!

SIMD design choices: Synchronous vs. asynchronous

- Each processor can execute or ignore the instruction being broadcast based on its local state or data-dependent conditions.
  - This leads to some inefficiency in executing conditional computations.
- For example: an “if-then-else” statement is executed by first enabling the processors for which the condition is satisfied and then flipping the “enable” bit before getting into the “else” part.
  - On the average, half of the processors will be idle for each branch.
- Even worse for “case” statements involving multiway branches.
- A possible cure is to use the **asynchronous** version of SIMD,
  - known as **SPMD** (spim-dee or **single-program, multiple data**):
  - each processor runs its own copy of the common program.
  - the advantage is that in an “if-then-else” computation, each processor will only spend time on the relevant branch.
  - the disadvantages include the need for occasional synchronization and the higher complexity of each processor, which must now have a program memory and instruction fetch/decode logic.

SIMD design choices: Custom vs. commodity-chip SIMD

- commodity (off-the-shelf) components:
  - components tend to be inexpensive because of mass production.
  - such general-purpose components will likely contain elements that may not be needed for a particular design.
  - These extra components may complicate the design, manufacture, and testing of the SIMD machine and may introduce speed penalties as well.
- custom components
  - including ASICs = application-specific ICs, multichip modules, or WSI = wafer-scale integrated circuits
  - generally offer better performance
  - lead to much higher cost in view of their development costs being borne by a relatively small number of parallel machine users

Multi-core

- Up to this point, we only considered systems that process just one instruction per cycle.
  - This applies to all computers containing only one processing core
- With multi-core CPUs, single-CPU systems can have more than one processing core, making them MIMD systems.
- Combining several processing cores or processors (no matter if scalar or vector processors) yields a computer that can process several instructions and data sets per cycle.

Between SIMD and MIMD

- SIMD inefficiency => a natural evolution of multiprocessor systems towards the more flexible MIMD model
  - especially the merged programming model in which there is a single program on each node.
    - This merged programming model is a hybrid between the data parallel model and the message passing model
- Successfully exemplified by Connection Machine CM-5
- In this SPMD (single program multiple data) model,
  - data parallel programs can enable or disable the message passing mode typical for MIMD,
  - ⇒ thus one can take advantage of the best features of both models
- Other examples of such platforms: Sun Ultra Servers, multiprocessor PCs, workstation clusters, the IBM SP.

SIMD pro and contra

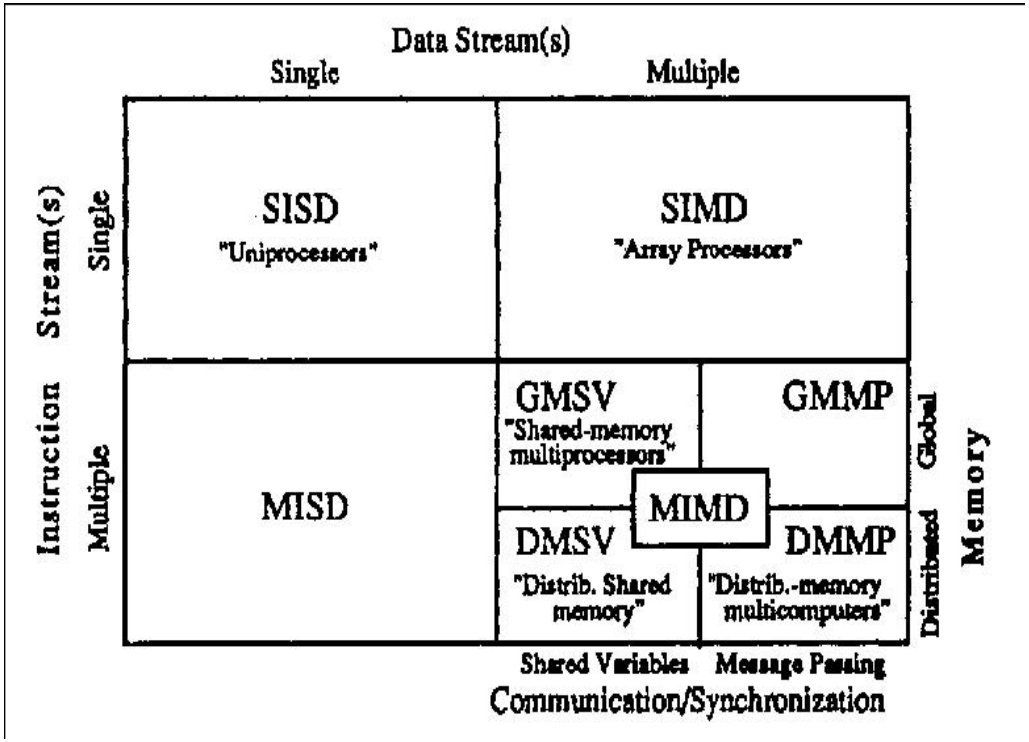
- SIMD computers require less hardware than MIMD computers
  - because they have only one global control unit.
  - because only one copy of the program needs to be stored.
- MIMD computers store the program and operating system at each processor.
- The relative unpopularity of SIMD processors as general purpose compute engines can be attributed to:
  - their specialized hardware architectures,
  - economic factors,
  - design constraints,
  - product life-cycle, and
  - application characteristics
  - extensive design effort resulting in longer product development times
  - the irregular nature of many applications

MIMD pro and contra

- higher flexibility of the MIMD architecture compared with SIMD
- ability to take advantage of commodity microprocessors
- ⇒ avoiding lengthy development cycles
- ⇒ getting a free ride on the speed improvement curve for such microprocessors.
- Most effective for medium- to coarse-grain parallel applications,
  - computation is divided into relatively large subcomputations or *tasks* whose executions are assigned to the various processors.
- Advantages of MIMD machines include:
  - flexibility in exploiting various forms of parallelism,
  - relative ease of partitioning into smaller independent parallel processors in a multiuser environment
  - less difficult expansion (scalability).
- Disadvantages include:
  - considerable interprocessor communication overhead and
  - more difficult programming.

MIMD subcategories | Flynn–Johnson classification

- 1988: **E. E. Johnson** proposed a further classification of MIMD based on :
  1. their memory structure: global or distributed
  2. mechanism used for communic./ synchronization: shared variables or message passing.
- GMSV: **shared-memory multiprocessors**
- DMMP: **distributed-memory multicomputers**
- DMSV: is sometimes called **distributed shared memory**
  - combine the implementation ease of distributed memory with the programming ease of the shared-variable scheme
- GMMP is not widely used



Shared memory (SM-MIMD)

- All processors are connected to a common mem (*RAM*-Random Access Mem)
- *Usually* all processors are identical and have equal memory access
  - This is called **symmetric multiprocessing (SMP)**.
- The connection between procs and memory is of predominant importance.
- For example: a shared memory system with a **bus** connection.
  - advantage of a bus is its expandability.
  - disadvantage is that all processors have to share the bandwidth provided by the bus
- To circumvent the problem of limited memory bandwidth, direct connections from each CPU to each memory module are desired.
  - This can be achieved by using a **crossbar switch**.
  - The problem is their high complexity when many connections need to be made.
  - This problem can be weakened by using multi-stage crossbar switches, which in turn leads to longer communication times.
  - ⇒ No. CPUs&mem modules than can be connected by crossbar switches is limited.
- Advantage of shared mem.systs: all processors make use of the whole mem
  - ⇒ This makes them easy to program and efficient to use.
- The limiting factor to their performance is the number of processors and memory modules that can be connected to each other.
  - ⇒ Shared memory-systems usually consist of rather few processors.

Distributed memory (DM-MIMD)

- Each processor has its own local memory.
- The processors are connected to each other.
- The demands imposed on the communication network are lower than in the case of a SM-MIMD
  - the communication between processors may be slower than the communication between processor and memory.
- Distributed memory systems can be hugely expanded
  - Several thousand processors are not uncommon, this is called **massively parallel processing (MPP)**.
- To actually use the theoretical performance, much more programming effort than with shared memory systems is required.
  - The problem has to be subdivided into parts that require little communication.
  - The processors can only access their own memory.
  - Should they require data from the memory of another processor, then these data have to be copied.
  - Due to the relatively slow communications network between the processors, this should be avoided as much as possible.

ccNUMA

- Shared memory systems suffer from a limited system size,
- Distributed memory systems suffer from the arduous communication between the memories of the processors.
- A compromise is the **ccNUMA (cache coherent non-uniform memory access)** architecture.
  - consists of several SMP systems.
  - these are connected to each other by means of a fast communications network, often crossbar switches.
  - Access to the whole, distributed or non-unified memory is possible via a common cache.
  - A ccNUMA system is as easy to use as a true shared memory system, at the same time it is much easier to expand.
  - To achieve optimal performance, it has to be made sure that local memory is used, and not the memory of the other modules, which is only accessible via the slow communications network.
  - The modular structure is another big advantage of this architecture.
    - Most ccNUMA system consist of modules that can be plugged together to get systems of various sizes.

Design issues: MPP—massively vs. moderately parallel processor

- Massive parallelism is generally taken to include 1000 or more procs
- Is it more cost-effective to build a parallel processor out of a relatively small no.of powerful procs or a massive no.of very simple procs
  - the “herd of elephants” or the “army of ants” approach?
- A general answer cannot be given to this question, as the best choice is both application- and technology-dependent.
- In the 1980s:
  - several massively parallel computers were built and marketed.
- In the 1990s:
  - a general shift from massive to moderate parallelism (tens to hundreds of processors),
- The notion of massive parallelism has not been abandoned,
  - particularly at the highest level of performance required for Grand Challenge problems (see Top 500!)

Design issues: Tightly vs. loosely coupled MIMD

- Which is a better approach to HPC?
  1. using specially designed multiprocessors/ multicomputers
  2. a collection of ordinary workstations that are interconnected by commodity networks and whose interactions are coordinated by special system software and distributed file systems
     - referred to as *network of workstations* (NOW) or **cluster computing**, has been gaining popularity in recent years.
- An intermediate approach is to link tightly coupled clusters of processors via commodity networks:
  - Clusters of clusters = Grids
  - This is essentially a hierarchical approach that works best when there is a great deal of data access locality.

Design issues: Explicit message passing vs. virtual shared memory

- Which scheme is better?
  1. forcing the users to explicitly specify all messages that must be sent between processors
  2. allow them to program in an abstract higher-level model, with the required messages automatically generated by the system software
- This question is essentially very similar to the one asked in the early days of high-level languages and virtual memory:
  - At some point in the past, programming in assembly languages and doing explicit transfers between secondary and primary memories could lead to higher efficiency
  - Nowadays, software is so complex and compilers and OS so advanced that it no longer makes sense to hand-optimize the programs, except in limited time-critical instances.
- However, we are not yet at that point in parallel processing, and hiding the explicit communication structure of a parallel machine from the programmer has nontrivial consequences for performance.

Communication Model of Parallel Platforms

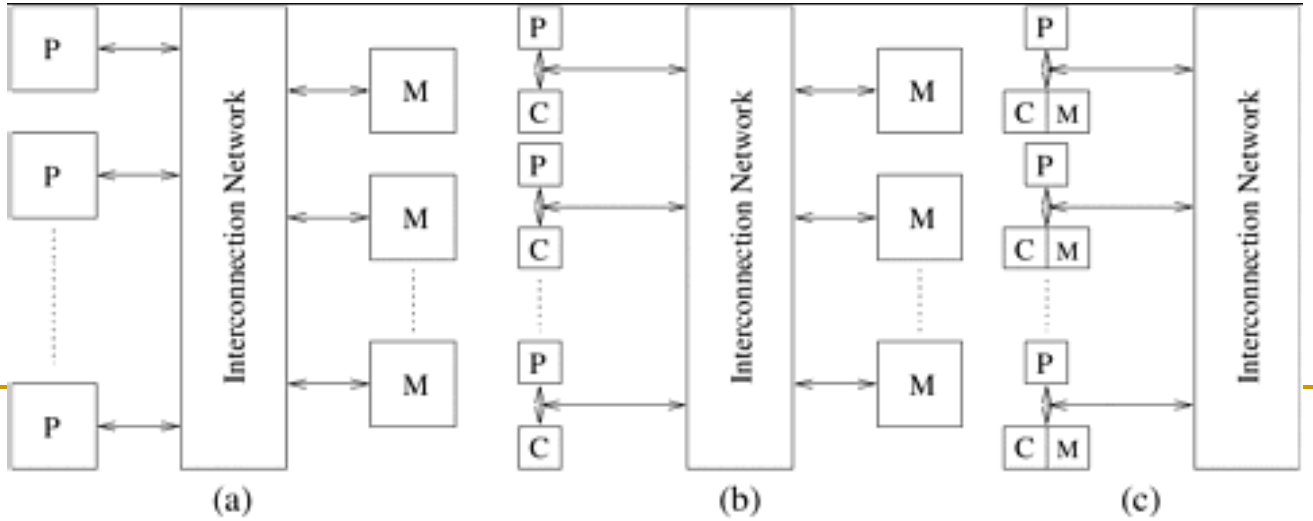
1. shared data space
2. exchanging messages

Shared-Address-Space Platform

- Supports a common data space that is accessible to all processors.
- Processors interact by modifying data objs stored in shared-address-space.
- Shared-address-space platforms supporting SPMD programming are also referred to as **multiprocessors**.
- Memory in shared-address-space platforms can be local (exclusive to a processor) or global (common to all processors).

1. If the time taken by a processor to access any memory word in the system (global or local) is identical, the platform is classified as a **uniform memory access** (UMA) multicomputer (see (a) and (b))

- If the time taken to access certain memory words is longer than others, the platform is called a **non-uniform memory access** (NUMA) multicomputer.
- Machines such as the SGI Origin 2000 and Sun Ultra HPC servers belong to the class of NUMA multiprocessors.



Issues in using Shared-Address

- The presence of a global memory space makes programming such platforms much easier.
- All read-only interactions are invisible to the programmer, as they are coded no differently than in a serial program.
  - This greatly eases the burden of writing parallel programs.
- Read/write interactions are harder to program than the read-only, as these operations require mutual exclusion for concurrent accesses.
  - Shared-address-space programming paradigms such as threads (POSIX, NT) and directives (OpenMP) therefore support synchronization using locks and related mechanisms.
- The presence of caches on processors also raises the issue of multiple copies of a single memory word being manipulated by two or more processors at the same time.
  - Supporting a shared-address-space involves two major tasks:
    1. need an address translation mechanism to locates a memory word
    2. ensure that concurrent operations on multiple copies of the same memory word have well-defined semantics - **cache coherence mechanism**.

Issues in using Shared-Address

- shared-address-space machines
  - only support an address translation mechanism
  - leave the task of ensuring coherence to the programmer.
    - The native programming model for such platforms consists of primitives such as get and put.
    - These primitives allow a processor to get (and put) variables stored at a remote processor.
    - If one of the copies of this variable is changed, the other copies are not automatically updated or invalidated.
- Since 2005, x86- compatible CPUs designed for desktop computers are available with two “cores”(makes them dual-processor systs).
  - This cheap extra computing power has to be used efficiently, which requires parallel programming.
  - Parallel programming methods that work on multi-core PCs also work on larger shared memory systems,
  - A program designed for a cluster or other type of distributed memory system will also perform well on multi-core PC.

Shared-address-space vs. shared-memory computers

- The term shared-memory computer is historically used for architectures in which the memory is physically shared among various processors,
  - Each processor has equal access to any memory segment.
  - This is *identical to the UMA model*
- This is in contrast to a distributed-memory computer:
  - Different segments of the memory are physically associated with different processing elements.
- Either of these **physical models**, shared or distributed memory, can present the **logical view** of a disjoint or shared-address-space platform.
  - A distributed-memory shared-address-space computer is identical to a NUMA machine.

Message-Passing Platforms

- Each processing node with its own exclusive address space.
- Each of these processing nodes can either be single processors or a shared-address-space multiprocessor
  - a trend that is fast gaining momentum in modern message-passing parallel computers.
- Instances of such a view come naturally from clustered workstations
- On such platforms, interactions between processes running on different nodes must be accomplished using messages (=> message passing).
  - This exchange of messages is used to transfer data, work, and to synchronize actions among the processes.
- Message-passing paradigms support execution of a different program on each of the nodes.
- Examples of parallel platforms that support the message-passing paradigm include the IBM SP, SGI Origin 2000, and workstation clusters.
- It is easy to emulate a message-passing architecture containing on a shared-address-space computer with the same number of nodes.
- Emulating a shared-address-space architecture on a message-passing computer is costly,
  - accessing another node's memory requires sending and receiving messages.

Message passing as programming paradigm

- Interactions are accomplished by sending and receiving messages => the basic operations are send and receive.
- Since the send and receive operations must specify target addresses, there must be a mechanism to assign a unique identification or ID to each of the multiple processes executing a parallel program.
  - This ID is typically made available to the program using a function such as whoami, which returns to a calling process its ID.
- There is one other function that is typically needed to complete the basic set of message-passing operations – numprocs, which specifies the no. of processes participating in the ensemble.
- With these four basic operations, it is possible to write any message-passing program.
- Different message-passing APIs, such as the

1. Message Passing Interface (MPI) and
2. Parallel Virtual Machine (PVM),

support these basic operations and a variety of higher level functionality under different function names.

Physical organization

Supercomputer

- A **supercomputer** is the fastest computer of its time
- Today's supercomputer is tomorrow's desktop or laptop computer.
- One of the first supercomputers of historical significance was the Cray-1.
  - It was used quite successfully in many applications involving large-scale simulation in the early 1980s.
  - The Cray-1 was not a parallel computer, however, but it employed a powerful (at the time) vector processor with many vector registers attached to the main memory.
- **Today, all supercomputers are parallel computers** (see Top500).
  - Some are based on specialized processors and networks
  - But the majority are based on commodity hardware and open source operating system and applications software.

Historical context: intrinsec & explicit parallelism

- Parallelism initially invaded computers at the processor level under several aspects:
  - The first one took place during the era of scalar processors,
    - in the development of coprocessors taking in charge some specific tasks of the working unit (mathematical ops, communications, graphics,...) and relieving the Central Processing Unit (CPU). Another aspect has resided in the processor itself.
  - The development of
    1. Complementary Metal-Oxide-Semiconductor (CMOS) technology since 1963
    2. the Very-Large-Scale Integration (VLSI) since the 1980s

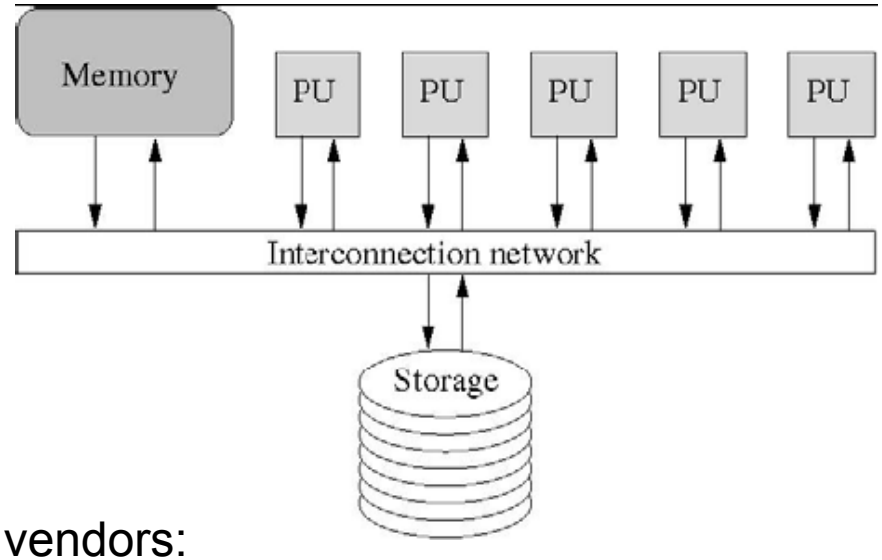
  ⇒ the inclusion of more and more complex components in the processors such as pipelines and multiple computation units.
- As the CMOS technology is nearer and nearer its physical limits

⇒ intrinsic parallelization of the processors has logically been followed by the emergence of multiple-core processors.

[More comments in the textbook!]

Shared vs. distributed memory

Parallel machine with shared memory



Examples of “old” vendors:

- most famous examples are the Cray series such as Cray- 1 and 2, and Cray X-MP and Y-MP, Cray X1E;
- others vendors: Convex or Alliant, Univ. of Illinois (Illiac IV), BBN, Sequent, SGI.

Resurge resurgence of shared memory computers in the form of:

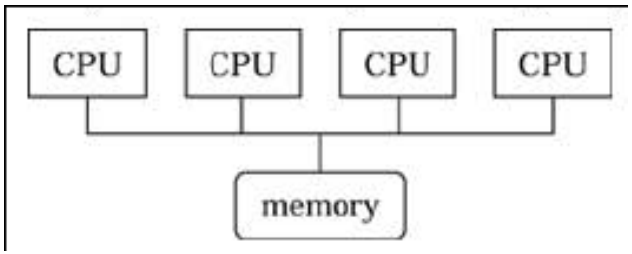
- multiprocessor desktops, which usually contain two or four processors;
- midrange systems with four or eight processors, primarily used as servers;
- and high-performance nodes for the top of the line parallel computers, which usually contain 16, 32, or even 64 processors (switching technology is applied).

Advantages & disadvantages

- Advantages:
  - neither require data distributions over the processors nor data messages between them.
  - the communications between PUs required for the control of the application are implicitly performed via the shared memory and can thus be very fast.
  - the memory connection is facilitated by fast bus technology or a variety of switch types (i.e., omega, butterfly etc)
  - easy to program
    - with loop parallelization being the simplest and the most efficient means of achieving parallelism
- Disadvantages:
  - Cache memory resulted in relatively complicated data management/cache coherency issues.
  - the fact that it always was and still is almost impossible
    - to scale shared memory architectures to more than 32 processors
    - to simultaneously avoid saturating the bandwidth between the processors and the global memory.

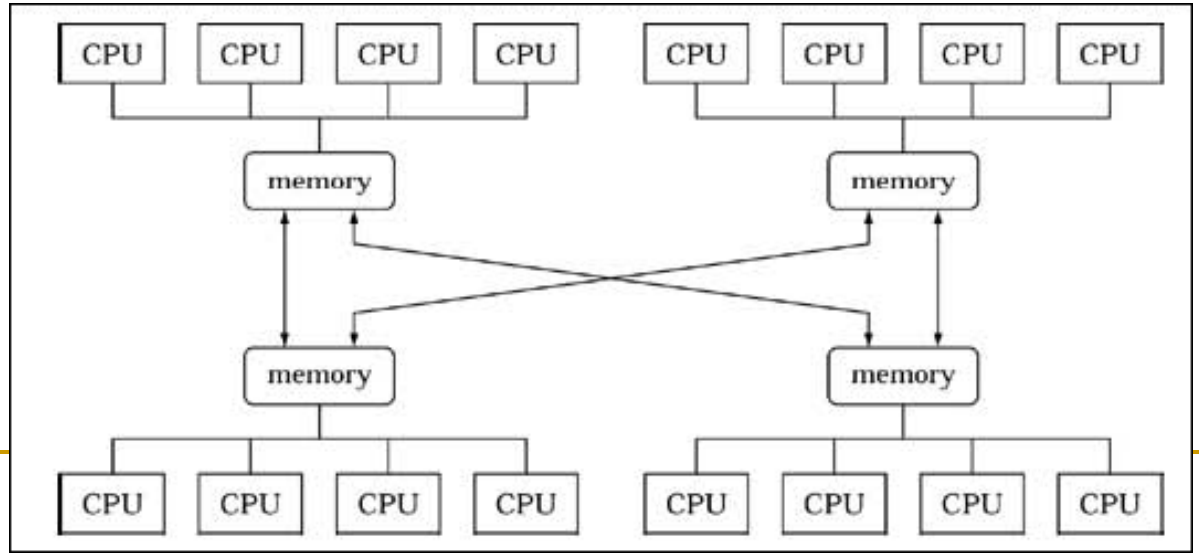
Special class 1: SMPs – symmetric multiprocessors

- all processors share a connection to a common memory and access all memory locations *at equal speeds*.
- SMP systems are arguably the easiest parallel systems to program because programmers do not need to distribute data structures among processors.
- because increasing the number of processors increases contention for the memory, the processor/memory bandwidth is typically a limiting factor.
- SMP systems do not scale well and are limited to small numbers of processors.



Special class 2: NUMA - non-uniform memory access

- the memory is shared,
  - it is uniformly addressable from all processors,
  - BUT some blocks of memory may be physically more closely associated with some processors than others.
  - ⇒ This reduces the memory bandwidth bottleneck& allows systs with more procs;
  - ⇒ The access time from a processor to a memory location can be significantly different depending on how "close" the memory location is to the processor
- To mitigate the effects of non-uniform access, each processor has a cache, along with a protocol to keep cache entries coherent
  - ⇒ another name for these architectures is **cache-coherent non-uniform memory access systems** (ccNUMA)



Drawbacks of memory centralization & solutions

- it implies a very high memory bandwidth, potentially with concurrency, in order to avoid bottlenecks
  - ⇒ the interconnection network between the memory and the PUs as well as the memory controller speed are often the limiting factors of the no. of PUs included in that kind of machine.
- Solutions?
  1. Processors can access memory through a special processor-to-memory **network that must have very low latency**
     - quite a difficult design challenge for more than a few processors
  2. ***memory latency-hiding*** techniques must be employed
     - An example of such methods is the use of multithreading in the processors so that they continue with useful processing functions while they wait for pending memory access requests to be serviced.
  3. [optional] processor-to-processor network may be used for coordination and synchronization purposes.

Examples of processor-to-memory and processor-to-processor networks

- Assume:
  - number *p* of processors,
  - number *m* of memory modules

1. Crossbar switch; O(*pm*) complexity, and thus quite costly for highly parallel systems
2. Single or multiple buses (the latter with complete or partial connectivity)
3. Multistage interconnection network (MIN); cheaper than Example 1, more bandwidth than Example 2

Cache

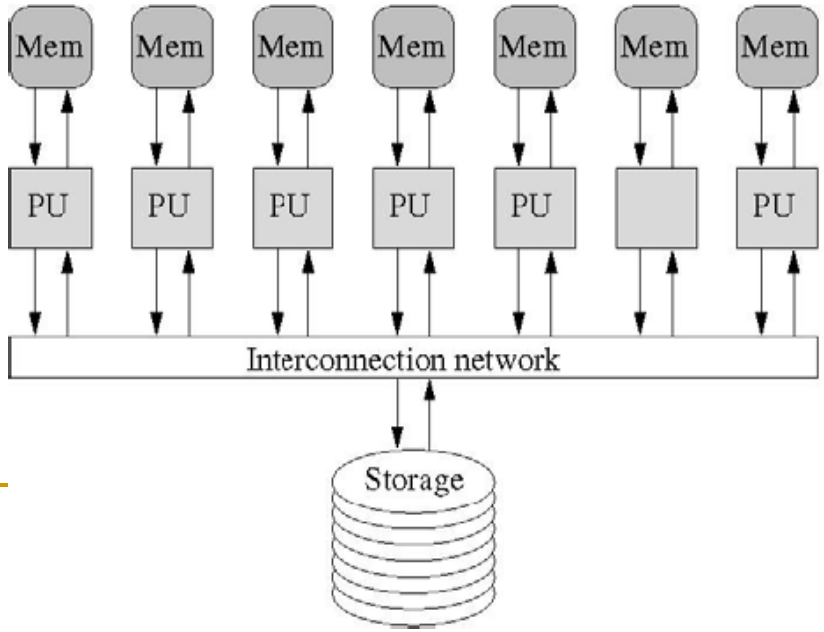
- Motivation: reduce the amount of data that must pass through the processor-to-memory interconnection network
- Use a private cache memory of reasonable size within each processor.
- The reason that using cache memories reduces the traffic through the network is the same here as for conventional processors:
  - locality of data access,
  - repeated access to the same data, and
  - the greater efficiency of block, as opposed to word-at-a-time, data transfers.
- Multiple caches gives rise to the ***cache coherence*** problem:
  - Copies of data in the main memory & in various caches may become inconsistent.
  - Approach:
    1. Do not cache shared data at all or allow only a single cache copy.
       - If the volume of shared data is small and access to it infrequent, these policies work quite well.
    2. Do not cache “writeable” shared data or allow only a single cache copy.
       - Read-only shared data can be placed in multiple caches with no complication.

Use a cache coherence protocol

- introduce a nontrivial consistency enforcement overhead, depending on the coherence protocol used, but removes the above restrictions.
- Examples:
  - *snoopy caches* for bus-based systems
    - each cache monitors all data transfers on the bus to see if the validity of the data it is holding will be affected
  - *directory-based schemes*
    - where writeable shared data are “owned” by a single processor or cache at any given time, with a directory used to determine physical data locations

Parallel machines with distributed memory

- the PUs are still linked together by an interconnection network
- each PU has its own memory with an exclusive access.
- some resources which are still shared like, for example, the mass storage or the I/O operations
- Vendors: Intel, NCube, Inmos, Convex, Cray, IBM.
- “Old” Representatives: Cray T3D/T3E, Intel iPSC/2 and Paragon, Connection Machines (CM-1 to CM-5), MasPar (MP1& MP2).
- Current: IBM Blue Gene or the Cray Red Storm.



Advantages & disadvantages

- Advantages:
  - can be scaled to a large number of processors.
    - Even some of the early distributed memory machines successfully employed up to 1024 processors.
  - larger flexibility in terms of design and configuration has made them more financially viable than their predecessors
- Disadvantages:
  - more difficult to write and debug than shared memory p.c
  - the necessity of a data distribution over the processors' own memory and
  - the use of messages passing between processors to exchange data or information to control the application
  - the performances of the interconnection network between the PUs is also a critical point

Interconnection networks

- Each processor is usually connected to the network through multiple links or *channels*
- *direct networks:* the processor channels are directly connected to their counterparts in other processors according to some interconnection pattern or *topology.*
- Example:
  1. **Bus**: a congestion-bound, but cheap, and scalable with respect to cost network.
  2. **Dynamic networks**: for example, a *crossbar* which is hard to scale and has very many switches, is expensive, and typically used for only limited number of processors.
  3. **Static networks**: for example, Ω-networks, arrays, rings, meshes, tori, hypercubes.
  4. **Combinations**: for example, a bus connecting clusters which are connected by a static network.

Classification: MPP and clusters

- MPP (massively parallel processors)
  - the processors and the network infrastructure are tightly coupled and specialized for use in a parallel computer.
  - extremely scalable, in some cases supporting the use of many thousands of processors in a single system.
- Clusters:
  - Of tens order

Distrib mem. P.C. as NUMA & hybrid systems

- distributed-memory MIMD machines are sometimes described as nonuniform memory access (NUMA) architectures.
  - *all-cache* or *cache-only* memory architecture (**COMA**) for such machines.
- Hybrid:
  - clusters of nodes with separate address spaces in which each node contains several processors that share memory.
  - made from clusters of SMPs connected by a fast network are currently the dominant trend in high-performance computing.
    - For example, in late 2003, four of the five fastest computers in the world were hybrid systems.