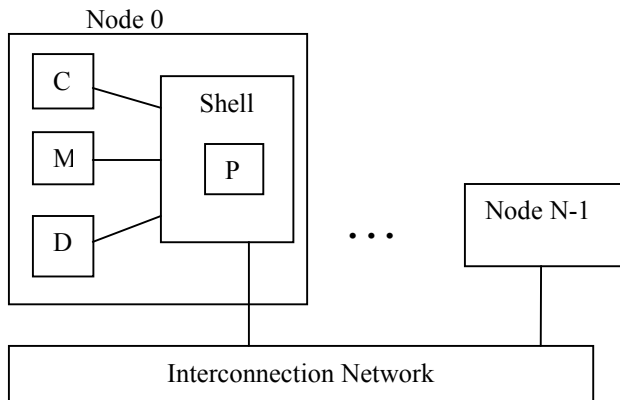


Parallel Computer Architectures and Performance Models

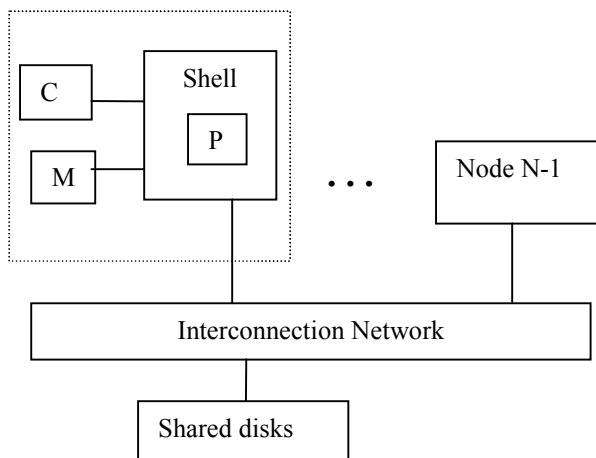
A parallel machine model is an abstraction which characterizes the fundamental operating properties that impact performance of parallel computers. Such a model suppresses the implementation details of the machine, yet describes its operation with sufficient detail to yield meaningful measures of performance. Such a model should not be confused with an *architectural structure*, which depicts how system components are connected and which resources are shared. For example, Flynn's architectural classifications (Hwang, fig. 1.3) are usually depicted by structures in block diagram form that show the interconnection of the processors, the information paths, and the shared resources. However, to determine performance of a specific machine on a specific computation, we need a model for how the computation will proceed and what assumptions may be made about conflicts with shared resources and time delays caused by overhead.

Another example of architectural structure is given by the three basic structures to which all parallel computers seem to be converging (see ref 1):

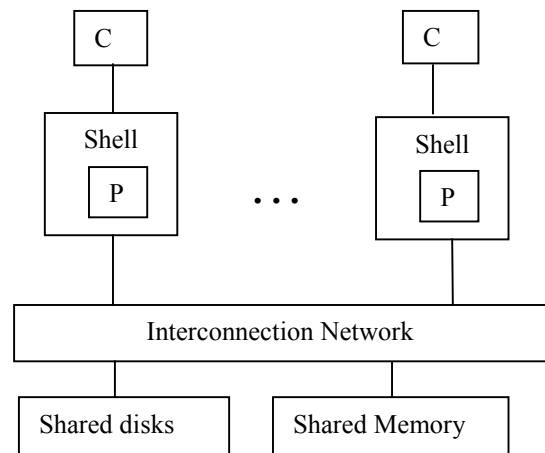


C = Cache
 M = R. A. Memory
 D = Disk
 P = Processor
 Shell = Specialized circuitry that interfaces the Processor to the rest of the node.

(a) Shared-nothing architecture

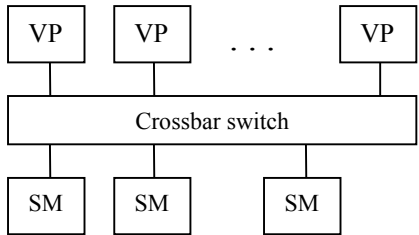


(b) Shared-disk architecture

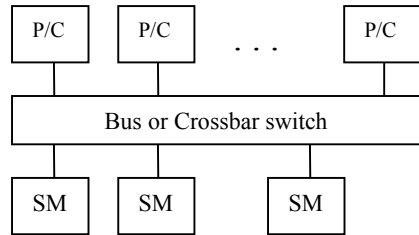


(c) Shared-memory architecture

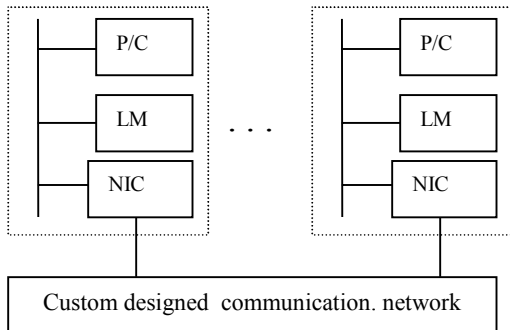
In reference 1, Hwang gives another architectural classification scheme, pointing out that large scale computer systems are generally implemented as one of 6 practical physical machine organizations: Flynn's SIMD model, and 5 MIMD variations shown below.



PVP - Parallel Vector Processor

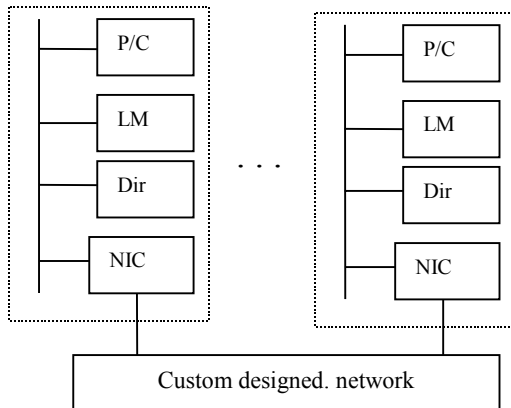


SMP - Symmetric Multiprocessor

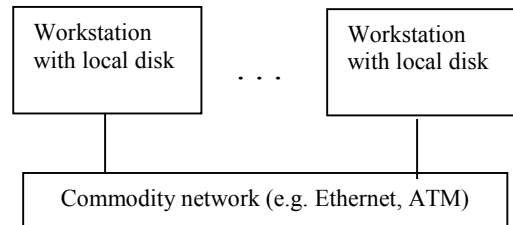


MPP - Massively Parallel Processor

SM - shared memory module
 LM - local memory
 NIC - network interface circuitry
 VP - vector processor
 P/C - scalar processor and cache
 Dir - address directory/translation



DSM - Distributed shared memory machine



COW - Cluster of workstations

In this course, there will be little discussion of the COW class system, the most loosely coupled system of the 6. Most of the issues with this class are software/operating system issues associated with synchronizing the various active processes and communications. This class system works best on problems where the workload can be divided into large chunks that can be executed in parallel with very low communication demands.

The tightly MPP class is for big problem scientific computing, and is characterized by using commercial microprocessors at each node with physically distributed memory across nodes, a high bandwidth communication network with nearly zero latency, and being scalable to hundreds and even thousands of processing nodes. The only difference between the MPP class and the SIMD class is asynchronous versus synchronous instruction execution. Examples are the Intel Paragon

and TFLOP. We will study the SIMD/MPP class and their interconnect networks near the end of the course.

The difference between the MPP and the DSM class is basically treating the distributed memory as an unshared versus a shared resource. Thus the address space of the processors is local in the MPP versus global in the DSM machines. Although the memory in the DSM is distributed like the MPP class, the hardware and system software create the illusion of a single address space to the application software. Examples are the Stanford DASH and the Cray T3D.

The PVP class machine contains a small number of powerful vector processors capable of at least 1 Gflop/sec, connected through a custom crossbar to high speed pipelined memory modules that can deliver data at a hellacious rate. Such machines do not generally use caches, but rather use a large number of *vector* registers and an instruction buffer. After covering the general topic of pipelining, we will discuss the PVP class machines in more detail. Examples are the various Cray vector machines and the Japanese machines such as the NEC SX-4.

Performance Models

A parallel performance model can be characterized by a number of so-called "Semantic" attributes, and a number of performance attributes. The semantic attributes are as follows:

Homogeneity - how alike the processors act when performing a parallel computation. In a MIMD class machine, for example, each processor can be executing different instructions, whereas on a SIMD machine each processor executes identical instructions at the same time.

Synchrony - how tightly synchronized the processors are. Real MIMD computers are asynchronous, with each processor operating on it's own cycle. If a processor has to wait for other processors, then extra "*synchronization operations*" must occur. This is one type of *overhead*.

Interaction mechanism - how the processors interact to affect one another's computation. This may be through shared variables in memory, or through messages or data passed between processors.

Memory address space - the set of memory addresses accessible to each processor. A MIMD machine interacting through message passing is often called a *multicomputer*, and each processor has it's own address space such as shown in figure (a) above. Figure (c) above is the class of MIMD machine called a *multiprocessor*, where all processors share a single memory space.

Memory model - how the access time varies over the address space, and how the machine handles conflicts in shared memory access. If each processor has equal access time to all addresses it can reference, then this is called the UMA model (uniform memory access). If the access time varies depending on location, this is called the NUMA model (non-uniform memory access). Exclusive read-exclusive write (EREW) means a given location or possible memory module can be read or written by only one processor at a time. CREW means a location or module can be simultaneously read by multiple processors, but exclusive write still holds. CRCW requires concurrent writes to be resolved by some specified conflict policy. In this regard, *consistency* describes how rigorously an algorithm's sequential ordering of memory references is adhered to.

We will use various performance attributes throughout the course. Some are as follows:

Machine size (n)- number of processors in the machine.

Workload (W)- the total machine operations to be performed in a computation, measured in MFLOP, instructions, or some such fine-grained computational unit.

Sequential execution time (T_1) - time to execute the workload sequentially. This time is typically expressed in cycles in this class, since unlike last semester, we are generally not interested in clock rate effect on performance, but rather the effect of all other factors.

Critical path time (T_∞) – The time to execute the workload with an infinite number of processors available and no overhead. This idealized time is determined only by the algorithm itself.

Parallel execution time (T_n) - time for n processors on a real machine to execute the workload.

This time includes all forms of overhead associated with parallel execution

Speed or Throughput (W/T_n) - the execution rate on an n processor system, measured in FLOPs/unit-time or instructions/unit-time.

Speedup ($S_n = T_1/T_n$) - how much faster in an actual machine, n processors compared to 1 will perform the workload. The ratio T_1/T_∞ is called the *asymptotic speedup*.

Efficiency ($E_n = S_n/n$) - fraction of the theoretical maximum speedup achieved by n processors

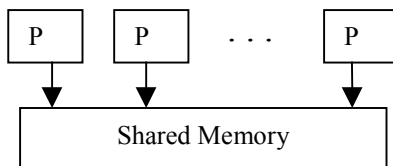
Degree of Parallelism (DOP) - for a given piece of the workload, the number of processors that can be kept busy sharing that piece of computation equally. Neglecting overhead, we assume that if k processors work together on any workload, the workload gets done k times as fast as a sequential execution.

Average parallelism ($P_n = (W\Delta)/T_n$) - average number of processors kept busy during a computation. The computing rate Δ is used to get consistent units. E.g. If workload is given as instructions, then we need to use the CPI if T_n is going to be computed in cycles.

Scalability - The attributes of a computer system which allow it to be gracefully and linearly scaled up or down in size, to handle smaller or larger workloads, or to obtain proportional decreases or increase in speed on a given application. The applications run on a scalable machine may not scale well. Good scalability requires the algorithm *and* the machine to have the right properties.

PRAM model (Parallel Random Access Machine):

This model assumes a shared memory, multiprocessor machine as shown:



1. The machine size n can be arbitrarily large
2. The machine is synchronous at the instruction level. That is, each processor is executing its own series of instructions, and the entire machine operates at a basic time step (cycle). Within each cycle, each processor executes exactly one operation or does nothing, i.e. it is *idle*. An instruction can be any random access machine instruction, such as: fetch some operands from memory, perform an ALU operation on the data, and store the result back in memory.
3. All processors implicitly synchronize on each cycle and the synchronization overhead is assumed to be zero. Communication is done through reading and writing of shared variables.
4. Memory access can be specified to be UMA, NUMA, EREW, CREW, or CRCW with a defined conflict policy.

The PRAM model can apply to SIMD class machines if all processors execute identical instructions on the same cycle, or to MIMD class machines if the processors are executing different instructions. Load imbalance is the only form of overhead in the PRAM model.

BSP (Bulk Synchronous Parallel) Model:

This model generalizes the PRAM so that each processor is at a node with its own memory, and the processors are interconnected through a communication network. The synchronous execution cycle now consists of fixed length super steps, in which each processor executes a series of instructions using data only in its local memory for w clocks, as well as a communication step that takes R

clocks. Synchrony is realized through what is called *barrier synchronization*, The barrier forces the processors to wait until all memory and communication operations by all processors are completed in the current super step before beginning the next super step. The BSP model accounts for possible overhead from communication and synchronization in addition to load imbalance caused by idle processors.

PSM (Phase Parallel Model):

This more recent model is similar to, but more general than the BSP model. The computation proceeds as a series of variable length steps, called phases, and a new phase cannot begin until all processors have completed the current phase. The computation continuously cycles through three phase types:

1. Parallelism phase, in which the overhead work of process management is performed.
2. Computation phase, in which one or more processors execute a number of local computations.
3. Interaction phase, in which communications, synchronization operations, or aggregation operations (e.g. reduction) are performed.

In contrast to the previous 2 models, which assume a uniform computation cycle, the workload and speed can vary in the PSM from phase to phase. The compute time during any given computation phase is determined by the maximum workload of any processor on that step. Similarly, the communication time or synchronization time may vary from step to step causing a non-uniform time for the interaction phases. Thus, this is the most realistic model of parallel performance, and I might add my favorite. The best way to understand all these models is through examples of specific computations on specific architectures.

As an example, we give the semantic attributes of Hwang's above 6 multiprocessor classes when applying any of the above performance models.

Attributes	SIMD	PVP/SMP	DSM	MPP/COW
Homogeniety	SIMD	MIMD	MIMD	MIMD
Synchrony	Instruction level	Asynchronous	Asynchronous	Asynchronous
Interaction	Message passing	Shared variable	Shared variable	Message passing
Address space	Multiple	Single	Single	Multiple
Memory access	UMA	UMA	NUMA	NUMA
Memory model	N/A	CREW with sequential consistancy	CREW or CRCW with weak consistancy	N/A
Example machines	Hypercube, Connection machine	IBM R50, Cray T-90	SGI Origin 2000, Stanford DASH	Cray T3E Berkely NOW

Reference

Hwang, K. and Xu, Z., "Scalable Parallel Computing", 1998, McGraw-Hill, ISBN 0-07-031798-4.