Parallel Computing Models

David McCaughan, *HPC Analyst*
SHARCNET, University of Guelph
dbm@sharcnet.ca

Parallel Computing Models

- Parallel Random Access Memory (PRAM)
  - conceptual model of multiple processors accessing memories
  - the idea of "clusters" broadens our notion of parallelism
- How is memory access controlled?
  - Exclusive Read, Exclusive Write (EREW)
    - only one processor can access a memory location at one time
  - Concurrent Read, Exclusive Write (CREW)
    - multiple processors can read a memory location, but only one at a time can write to it
  - Exclusive Read, Concurrent Write (ERCW)
    - multiple processors can write to a memory location but reads are exclusive
  - Concurrent Read, Concurrent Write (CRCW)
    - processors may freely read or write to memory locations without restriction

Flynn's Taxonomy (1966)

- Flynn characterized systems by
  - number of instruction streams
  - number of data streams

		Instructions	
		single	multiple
Data	single	**SISD** •1 processor •1 program •von Neumann	**MISD** •odd concept •vector processors (?)
	multiple	**SIMD** •1 CPU / many ALUs •each with memory •*synchronous*	**MIMD** •multiple CPUs •each with memory •*asynchronous*

SISD

- Traditional model of sequential computation
  - one processor
  - one memory space
- *Sequential algorithms* state a step-wise solution to a problem using simple instructions and only the following three classes of operation
  - sequence
  - iteration
  - branching
- Processor executes one instruction at a time
- ***Single Instruction Single Data (SISD)***

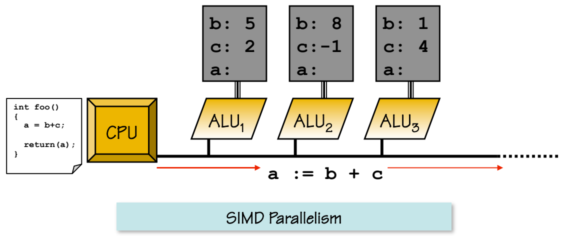
SHARCNET™

SIMD

- *Single Instruction Multiple Data* (SIMD)
  - one processor (conceptually)
  - multiple memories
    - or concurrent access to a single shared memory
  - all ALUs execute the exact same instruction
  - e.g.
    - Vector machines
    - AltiVec/Velocity Engine (Motorola, IBM, Apple)
    - MMX/SSE/SSE2 (Intel)

SHARCNET™

SIMD (cont.)



SHARCNET™

SIMD (cont.)

- Ideally suited to data parallel problems
  - solution requires we apply the same operation/algorithm to data elements that can be operated upon independently
    - expectation of high independence of operations (ideally perfectly parallel data)
  - e.g. vector arithmetic
    - compute <c> := <a> + <b>
    - on each processor, *i*
      - load elements a_i and b_i
      - execute $a_i + b_i$
      - store result c_i

SHARCNET™

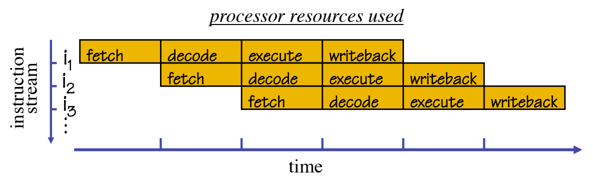
SIMD (cont.)

- Still amenable where communication patterns are highly regular and involve the same operations
  - e.g.
    - Euclidean inner product
    - summation
    - AND/OR
    - max/min
    - etc.

MISD

- *Multiple Instruction Single Data* (MISD)
  - unusual model to conceptualize
    - some argue there is no such thing
  - multiple operations applied to single stream of data
    - contrast with SIMD
  - e.g. pipelining
    - operations naturally divided into steps that make use of different resources
    - issues: branch prediction, instruction stalls, flushing a pipeline

MISD (cont.)

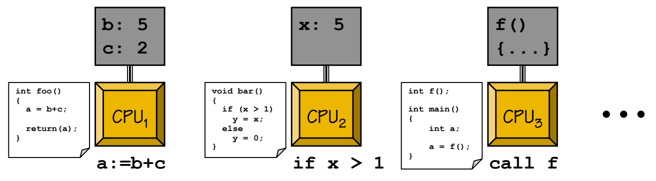


Pipelining as MISD model

MIMD

- *Multiple Instruction Multiple Data* (MIMD)
  - many processors executing different programs
- Memory organization is significant
  - Distributed memory
    - unique memory associated with each processor
    - issues: communication overhead
  - Shared memory
    - processors share a single physical memory
    - programs can share blocks of memory between them
    - issues: exclusive access, race conditions, synchronization, scalability

MIMD (Distributed Memory)



```
int foo()
{
  a = b+c;

  return(a);
}
```

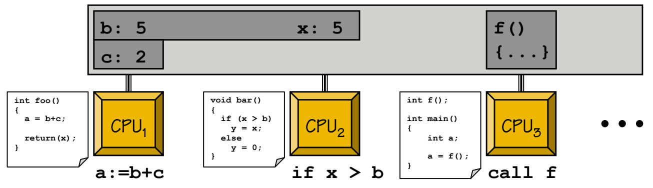
```
void bar()
{
  if (x > 1)
    y = x;
  else
    y = 0;
}
```

```
int f();

int main()
{
    int a;

    a = f();
}
```

Distributed Memory MIMD

SHARCNET™

MIMD (Shared Memory)



```
int foo()
{
  a = b+c;

  return(x);
}
```

```
void bar()
{
  if (x > b)
    y = x;
  else
    y = 0;
}
```

```
int f();

int main()
{
    int a;

    a = f();
}
```

Shared Memory MIMD

HPC Resources

SHARCNET™

MIMD (cont.)

- Most powerful and general model for parallel computing
  - it should be clear that SIMD models are a subset of MIMD
  - required to achieve task parallelism
- Unrestricted, unsynchronized parallelism can be difficult to design, test and deploy in practice
  - it is more common for the solution to a problem to be somewhat regular
    - natural points of synchronization even where the behaviour accross processes diverges
  - typical MIMD solutions involve partitioning the solution space and have only processes on partition boundaries communicate locally with one another (sub-problems are all nearly identical)

HPC Resources

SHARCNET™

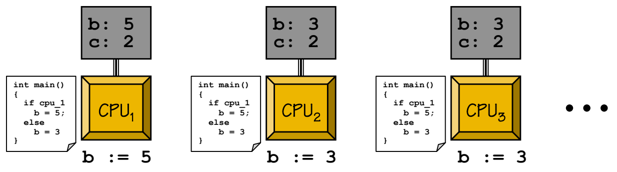
SPMD

- *Single Program Multiple Data* (SPMD)
  – special case MIMD
  – many processors executing the *same* program
    - conditional branches used where specific behaviour required on the processors
  – shared/distributed memory organization

HPC Resources

SHARCNET™

SPMD (cont.)



```
int main()
{
  if cpu_1
    b = 5;
  else
    b = 3
}
```

```
int main()
{
  if cpu_1
    b = 5;
  else
    b = 3
}
```

```
int main()
{
  if cpu_1
    b = 5;
  else
    b = 3
}
```

SPMD illustrating conditional branching to control divergent behaviour

HPC Resources

SHARCNET™

Communication

- Whatever the model, processors and memory need some way to communicate with one another
  - cache coherence
  - bus vs. crossbar switch
  - interconnection networks
    - multistage interconnection networks (MIN)
    - mesh, toroid, hypercube
  - routing solutions
    - pipelining
    - wormhole routing

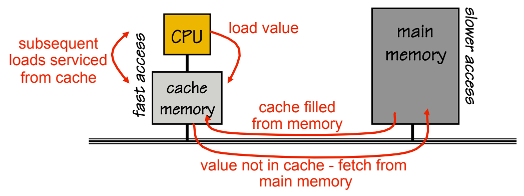
SHARCNET™

Cache Coherence

- Modern processors don't really see memory directly
  - view it through their cache (often multiple levels of it)
  - cache holds copies of values from main memory in very fast local storage
- Cache has such a dramatic effect on system performance it cannot be ignored (unit stride movement through memory, or close to it, is very common)

SHARCNET™

Cache Coherence



SHARCNET™

Cache Coherence (cont.)

- Cache coherence refers to keeping the contents of cache and memory consistent across multiple processors
  - ensuring cache coherence can impact performance
  - ignoring it can produce unusual behaviour
- Traditional strategies for cache coherence
  - write-through
    - variables written to cache are written to memory immediately
    - this works with a shared bus as other caches can see the memory update on the bus and update their contents
    - places pressure on bus bandwidth
  - writeback
    - cache only writes back contents when it needs the space
    - caches coordinate contents with one another via dedicated hardware

SHARCNET™

Cache Coherence (cont.)

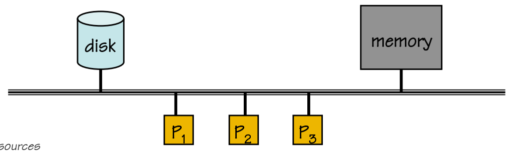
- As we consider more complex models cache coherence remains an issue
  - shared memory systems become increasingly complex
    - especially true for non-uniform memory models
  - in a message passing environment the issue can be ignored between nodes
    - communication between nodes is explicit
- Programmer typically does not have to deal with cache coherence issues directly
  - overall performance can be affected depending on the cache coherence policy implemented by the system vendor

HPC Resources

SHARCNET™

Bus

- Standard bus architecture
  - simply connect everything to the same bus
- Benefits
  - simple design
  - easier maintaining cache coherence (write-through or write-back)
- Issues
  - contention for the bus
  - performance degrades quickly



HPC Resources

SHARCNET™

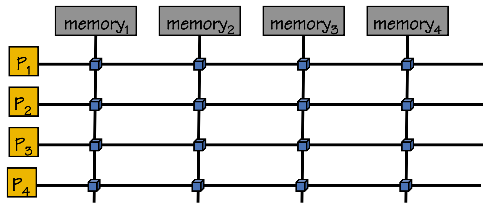
Crossbar Switch

- Each piece of hardware has a bus-like communication channel organized so that they interconnect in a grid
  - some path exists between each piece of hardware through the crossbar
  - attempts to mitigate bottleneck a single bus produces
- Switches at intersections route connections as necessary
  - must also resolve contention when more that one device wishes to access the same hardware

HPC Resources

SHARCNET™

Crossbar Switch (cont.)



A 4x4 Crossbar Switch

HPC Resources

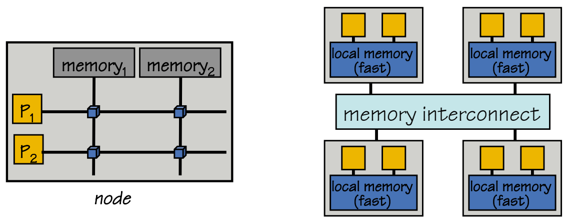
Crossbar Switch (cont.)

- Benefits
  - uniform memory access
  - efficient shared memory
  - programmer's view of system is simplified
- Issues
  - cache coherence (requires some means of broadcasting updates)
  - memory access patterns still significant
    - processors hammering the same block of memory will still saturate available bandwidth
  - does not scale well
    - increasing complexity (particularly the switches)
    - increasing cost

Non-uniform Memory/ Clusters

- Allow models to scale by abandoning uniform memory access
  - processors organized into nodes containing processors and local (fast) memory
  - nodes connected using an interconnect
  - local memory will be much faster than accessing memory on another node
    - shared memory systems may make global memory appear as a single memory space
    - in the extreme case is message passing in a cluster where there is no direct remote memory access

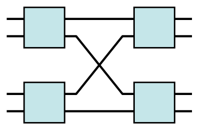
Non-uniform Memory / Clusters (cont.)



A Non-Uniform Architecture Constructed from 2-Processor Nodes

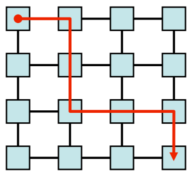
Multistage Interconnection Networks (MIN)

- A network of interconnected crossbars
  - scalability at the cost of additional routing steps to access memory
  - e.g.
    - 2x2 crossbar allows 2 nodes to communicate in one step
    - 4 2x2 crossbars allow 4 nodes to communicate in two steps
- Benefits
  - economical, relatively simple, scalability
- Issues
  - increased latency
  - increased opportunity for contention



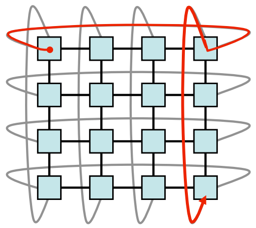
Mesh Topologies

- Nodes are interconnected in a two or three-dimensional grid
- Communication involves finding a path from one node to another through the grid
  - can involve large numbers of steps (highly non-uniform communication)
  - pipelining and intelligent routing to maximize performance



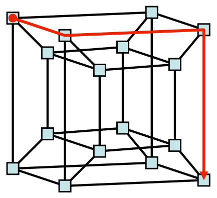
Toroid Topologies

- Mesh topology with "wrap-around" connections between nodes on the edge of the mesh
- Cuts number of hops in the worst case by 50%
- More complex to realize than simple meshes



Hypercube Topologies

- Minimize number of hops by organizing node adjacency on a hypercube
- Worst case number of hops for *n* nodes is $log_2 n$
  - e.g.
    - 256 nodes in 8-cube has max 8 hops (1024 connections)
    - in a 16 x16 2-D mesh it would have max 30 hops (225 connections)



Routing

- Pipelining
  - all data in a given communication can be routed using the same path
  - data packets sent one after another after the initial packet
  - path length (number of hops) only an issue for the first packet
    - remaining packets will arrive in sequence immediately after the first
    - significantly improves performance in multi-hop interconnects
- Wormhole routing (William Dally, MIT)
  - Allow data to route itself through an interconnection network
    - bits encoding destination sent at beginning of message
    - each router interprets the leading bits to determine outgoing path, strips these bits and sends rest of message unchanged
    - data sent by pipelining
    - the visual metaphor is of the head of the message moving through nodes like a worm with the data trailing behind