EEE4120F

High Performance Embedded Systems

Lecture 12: Distributed and Shared Memory Architectures

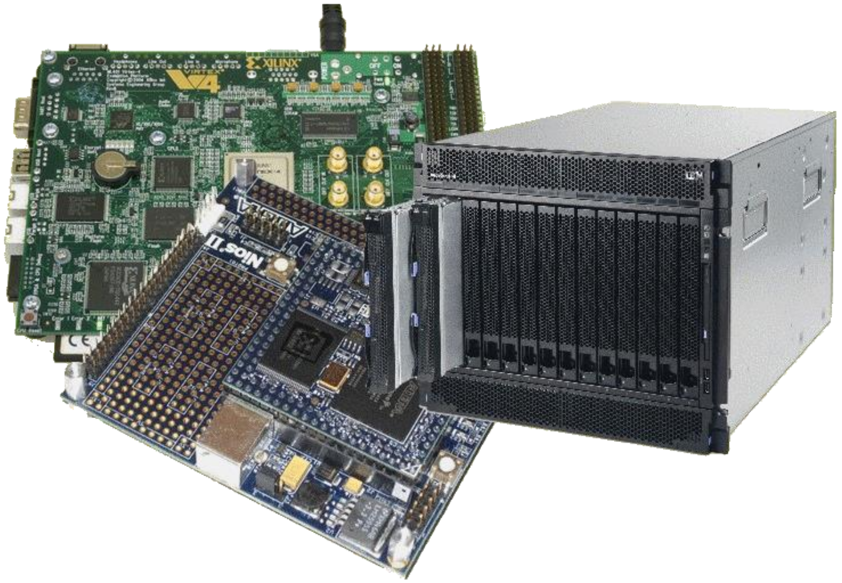
(This provides a fairly high-level discussion of these issues which are explored in more depth next term)

Presented by

Simon Winberg

Lecture Overview

- Distributed memory infrastructure
- Shared memory infrastructure
- Hybrid memory infrastructure
- Warming up for MMU, DMA and memory considerations to get into next term



(this version does not include Verilog scenarios, those moved to later lecture)

Distributed vs. Shared Memory

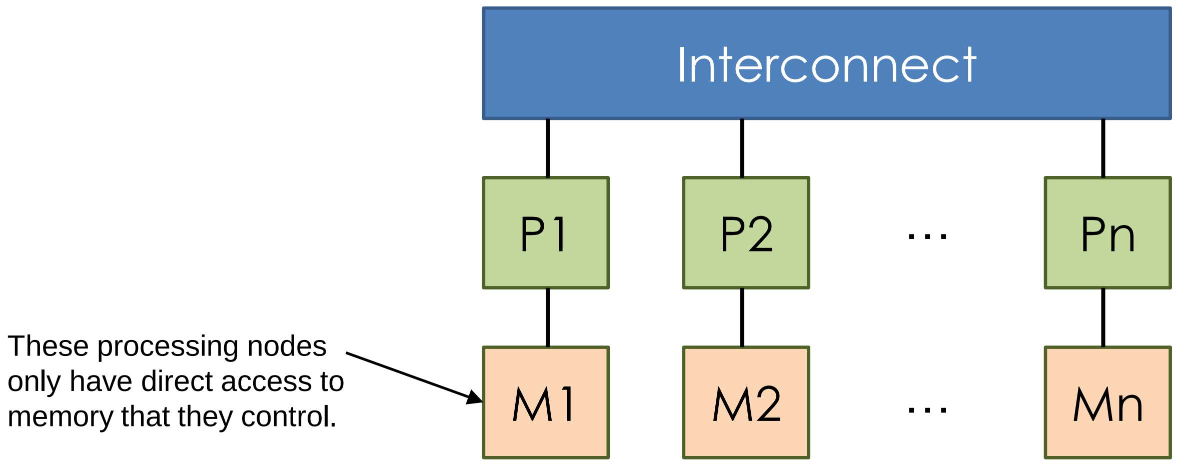
- We discussed communication for shared memory in Lecture 11… and that such communication is likely invisible.
- Message passing, often used in distributed memory, is often more visible communication.
- But what are the hardware approaches to implement shared and distributed memory?
- The aim of this lecture is to touch on those aspects briefly so that you have an understanding of how it can be done.

Distributed memory infrastructure

EEE4120F

Distributed Memory – the model

- The distributed model of memory is based around interconnected processing nodes, each having have their own memory. It involves: interconnects, the processors, and the processors' memory, as illustrated:


Interconnect
P1
P2
…
Pn
M1
M2
…
Mn
These processing nodes only have direct access to memory that they control.

Distributed Memory – advantages

- Main advantages of distributed memory:
  - Memory is scalable, although limited by number of connected processors (i.e. can add processors with memory to the network to get more memory)
- Can build very large systems, possibly many 1000s of processing nodes!
- Each processor has rapid access to *its own memory* without interference or cache coherency problems cause by other processors
- Cost effective. Easier to build (just use off-the-shelf / easily purchased parts)

Distributed Memory – disadvantages

- To read or write information from/to another processor’s memory a message must be sent over the network to processor concerned.
- Need to keep track somehow of where the memory is, i.e. who has it. (although things like memory brokers can be used instead to offload this task)
- Programmer is responsible for many of the details of the memory access and possibly communication – more places that mistakes can creep in
- May be difficult to distribute a complex data structure, and parallelize how maintenance and synchronization of such a data structure is done (e.g., think of possibly many processor working collaborating on an image, there may be stages where multiple processors are contending for access simultaneously).

Distributed Memory – Programming Models

- You might already know about the most popular one:
  - Message Passing Interface (MPI)
- Others:
  - Parallel Virtual Machine (PVM): https://en.wikipedia.org/wiki/Parallel_Virtual_Machine
  - LibDSM: (quite different to MPI): https://github.com/Micrified/libdsm
  - OpenSHMEM (some similarities to MPI, at least OpenMPI and can translate between easily): http://openshmem.org/site/About

Distributed Memory – is it No Remote or Remote?

- Basically indicate if an external network, e.g. needing switches, are involved…
- NORMA = NO Remote Memory Access
  - Nodes connected via network adaptors to external switches
- RMA = Remote Memory Access
  - Processing nodes connected via internal special interconnect hardware (e.g. SMP)
  - Often allows one-sided memory transfers (i.e. a get, or a put at a time)

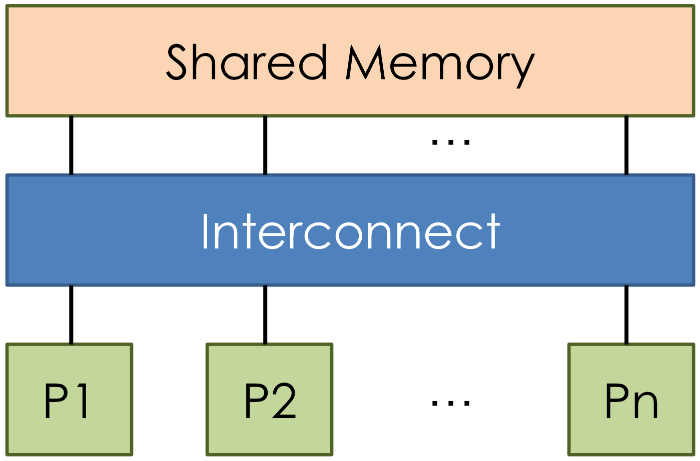
Shared memory infrastructure

EEE4120F

Shared Memory – the model

- To be more precise, this is more accurately described as "shared memory address space accessible by all processors"

The physical memory hardware modules storing the data may still be distributed somehow, but the different processing nodes use the same address to access a particular data item.



Shared Memory – complications for cache and local memory

- The processors may have their own local memory (e.g. caches) to hold copies of some global memory to boost performance.
- Maintaining the consistency of such copies (caches) is usually done by specialized hardware

Shared Memory – Programming Models

- You already know about the most popular ones:
  - pThreads (in Linux)
  - OpenMP
  - (not necessarily MPI… although it might end up using shared memory, possibility not knowing it is shared memory, or the library might be design to leverage shared memory for nodes that can access it)

Shared Memory – Advantages

- Global address space is more user-friendly for the programmer.
- Allows potential to use data structures, possibly even reusing the code without needing changes, and for the programmer to do so efficiently and with little modification
- Overall, Typically easier to program, less worry about explicit messaging between processors; provides:
  - Implicit communication in response to use of shared data (i.e. happens behind-the scenes, out of sight of programmer)
  - May, highly likely, will still need explicit synchronization and/or use of semaphores (i.e. to avoid multiple processors writing to the same location at once)
- Data sharing (communication) between tasks is very fast (not needing to be packed into messages send between processors)
- (Many of these points are mentioned in the OpenMP lecture)

Shared Memory – Disadvantages

- Needs specialized and likely expensive hardware for efficient (and scalable) memory access and cache coherence
- Tends not to be so scalable, may be limited to what is provided by the computer system (e.g. limited to say 10s to 100s of nodes; although in the case of GPUs many 100s available … but GPU kernels are usually much smaller and more focused than full application programs)

Shared Memory – Access Time Classifications

- UMA (Uniform Memory Access)
  - Equal access times to memory from each processor (most SMP provide this)
  - Almost always cache-coherent
  - Interconnects tends to be either*:
    - Bus: easier, and cheaper, but less scalable access time. (i.e. multiple processors share common memory bus to access global memory).
    - Crossbar: more difficult to implement, and more expensive (e.g. on FPGA takes more logic, essentially an exponential growth problem) but the very scalable speed.

* This is one of the important trade-off decision in designing these systems… A bus is highly scalable for adding systems, but not scalable in speed: more systems means more traffic and likelihood of slower transfers. Whereas the crossbar is the option: a specific crossbar design is fixed to connect only the chosen number of nodes, but provide good speed between any of the nodes wanting to connect.

Shared Memory – Access Time Classifications

- NUMA (Non-Uniform Memory Access)
  - Usually this is just a linking of multiple UMA nodes with a switching network (much like the Hybrid model which is next)
  - Nodes that share UMA can share memory more efficiently… but there may still be issues of synchronizing with all the other nodes / maintaining chace coherence on other machines
  - The cc-NUMA: a NUMA system enhanced to maintain fast cache coherence with other nodes.

* This is one of the important trade-off decision in designing these systems… A bus is highly scalable for adding systems, but not scalable in speed: more systems means more traffic and likelihood of slower transfers. Whereas the crossbar is the option: a specific crossbar design is fixed to connect only the chosen number of nodes, but provide good speed between any of the nodes wanting to connect.

Hybrid Memory

- Hybrid Memory is the logical extension of combining the best aspects of the distributed and shared memory architectures
- This brings us back to the ideal coupling of the MPI and the OpenMP households… essentially uses:
  - Message passing between nodes + Multi-threading within nodes

Other (commercial) examples of using Hybrid memory:
IBM BlueGene/P; Cray XT4

Hybrid Memory

Two typical forms of hybrid memory…

- Clusters of shared memory nodes, where:
  - Number of nodes >> number processors within node
  - Often use small, cheap SMP nodes (rack-mounted)
  - (These are sometimes called clumps within a cluster)
  - Commonly uses a standard network (e.g., Gigabit Ethernet)
- Constellation systems
  - Number of processors inside node > number of nodes
  - Comprises larger, more costly UMA or cc-NUMA nodes
  - Commonly uses special high-speed interconnects

MMU, DMA and other specialized memory hardware facilities …

- This lecture has intentionally not gone into issues of datapaths connecting with memory, FPGA memory solutions, or Digital Accelerators and memory … it is just a mild warm-up to get you thinking about these memory models and ways to share memory between collaborating processors (not necessary just CPUs)
- Topics that will be covered next term include:
  - Memory Management Units (essential component of implementing an effective memory architecture for a computer or digital accelerator)
  - Direct Memory Access (DMA) and their flavors
  - Other considerations for memory

Concluding high-level system design considerations

- This lecture concludes high-level design concepts for parallel systems
- You might now be wondering what is next in this course…
  let’s get directly into that!

Next week plans …

- The YODA mission begins!
- FPGAs, Verilog and Vivado
- Nexys4
- Thoughts of using iVerilog and where and how that can replace using Vivado

Disclaimers and copyright/licensing details

Image sources:

Pixabay

commons.wikimedia.org

Images from flickr