Multiprocessors
Chapter 8

CS448

1

The Greed for Speed

- Two general approaches to making computers faster
- Faster uniprocessor
  - All the techniques we've been looking at so far, plus others…
  - Nice since existing programs still work without changing them, except may need to be re-compiled with optimizations
  - But diminishing returns with higher cost, as with Amdahl's law
- Parallel processor
  - Typically a collection general purpose uniprocessors today
  - Large variation in memory access
  - Required for high-end computer systems, e.g. supercomputing, DOE ASCI program

2

Parallel Processing

- Advantages
  - Performance gains possible
  - Can be relatively inexpensive today with commodity processors
- Disadvantages
  - Software must now be changed radically to take advantage of the parallel machine
  - Hardware challenges
  - New types of overhead and organizational problems await the parallel machine

3

Types of Parallelism

- We've already seen some sorts of parallelism…
  - lookahead and pipelining
  - data and control parallelism
  - vectorization
  - concurrency
  - partitioning
  - interleaving physical subsystems (e.g. memory)
  - multiplicity and replication (e.g. multiple functional units)
  - time and space sharing
  - multitasking and multiprogramming
  - multithreading
  - distributed computing
- We'll focus primarily on multiprocessing here

4

Multiprocessing

- A few issues that stand out from uniprocessing
  - Communication
    - Interprocessor communication now comes into play
    - Can treat similarly to I/O
    - Issues of latency and bandwidth
  - Resource allocation
    - Allocated by programmer, compiler, hardware?

5

Communication among Multiple Processors

- Software perspective
  - Shared memory
    - E.g. one processor writes to memory location X, second processor reads from memory location X
    - Gets complicated with local vs. remote memory
    - Sharing and access model
    - Issues of speed, contention
  - Explicitly send messages to specific processors via send and receive
    - Similar to how computers operate on a network
    - Usually seen as message passing
- Hardware perspective
  - Software and hardware models should not conflict for efficiency
    - E.g. software treats “broadcast to all” as a cheap operation, when processor hardware does not support broadcast efficiently

6

Flynn's Taxonomy

- Proposed in 1972
- SISD – Single Instruction Single Data
  - Current uniprocessor
- SIMD – Single Instruction Multiple Data
  - Same instruction operated on in parallel by multiple processors using different data streams
- MISD – Multiple Instruction Single Data
  - Many instructions operated on in parallel by multiple instructions using the same data stream
  - No computers use this today
- MIMD – Multiple Instruction Multiple Data
  - Most flexible, each processor fetches and operates on its own data independently

7

SIMD vs. MIMD

- Rise and Fall and Rise of the SIMD machine
  - Was the earliest of the parallel computing models
    - Iliac IV, Connection Machine, Thinking Machines
  - Good at working with arrays in for-loops
  - Not so good at case statements, each execution unit must perform a different operation on its data
  - Revived somewhat today with MMX, SSE
- MIMD Machine more flexible
  - Can function as single-user machine for one application or multiprogrammed machines running many tasks simultaneously
  - Can build on cost/performance advantages of COTS microprocessors
  - Nearly all multiprocessors today built using commercial uniprocessors

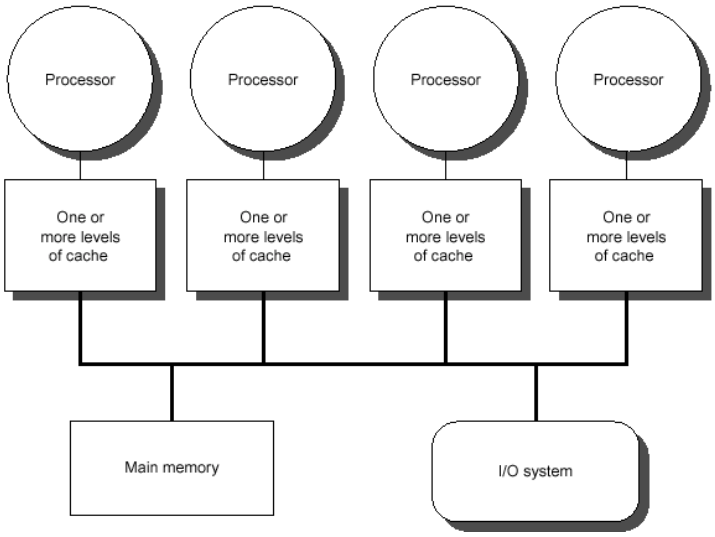
8

MIMD Architecture

- Two general classes of MIMD machines
  - Centralized Shared-Memory (CSM) Architectures
    - Typically used with a small number of processors
    - Connected to a single centralized memory somehow, typically via a bus
    - Sometimes called Uniform Memory Access (UMA) machines
    - Scalability issues with larger number of processors
  - Distributed Shared Memory (DSM) Architecture
    - Individual nodes contain memory, interconnected by some type of network
    - Easy to scale up memory, good if most accesses are to local memory
    - Latency and bandwidth issues between processors becomes key
    - Sometimes called Non-Uniform Memory Access (NUMA) machines
- Hybrid machines incorporating features of both are also possible

9

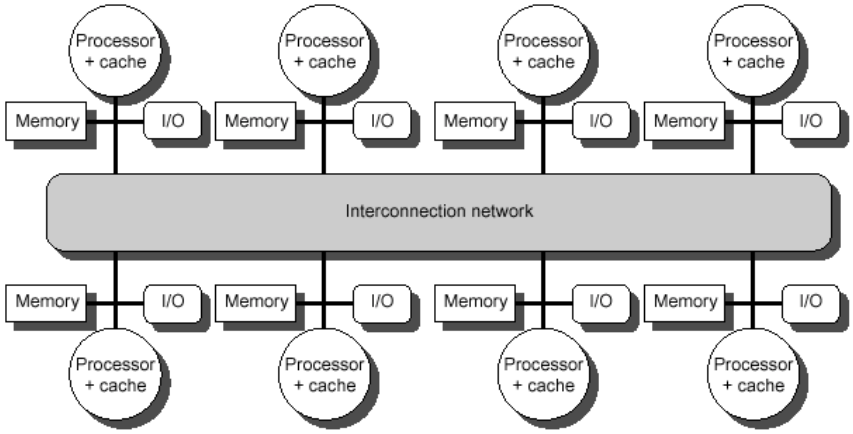
Centralized Shared Memory



Note: cache coherency problem

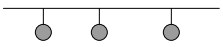
10

Distributed Shared Memory

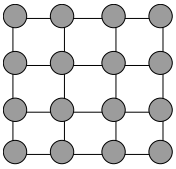


11

Example Interconnection Networks

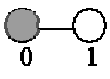


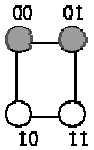
Bus

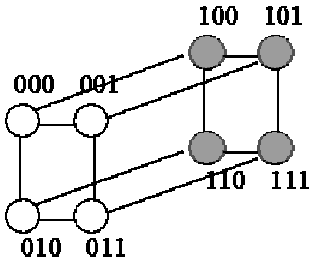


2D Mesh

Hypercube







12

Models for Memory, Communications

- Shared Memory
  - Does not mean there is a single centralized memory
- Address Space
  - May consist of multiple private address spaces logically disjoint in addition to shared memory
  - Essentially separate computers; sometimes called a multicomputer machine
  - For machines with multiple address spaces, communication of data performed by explicitly passing data between processors
    - Called *Message Passing Machines*

13

Message Passing Machines

- Data transmitted through interconnect similar to sending over a LAN
- For processor A to access or operate on data in processor B
  - A sends message to request data or operation to B
  - Message considered a *Remote Procedure Call* (RPC)
  - B performs operation or access on behalf of A and returns the result with a reply message
- Synchronous when A waits for reply before continuing
- Asynchronous when A continues operating while waiting for reply from B
- Program libraries exist to make RPC and message passing easier, e.g. MPI

14

Comparison of Communication Mechanisms

- Shared Memory Communication
  - Compatibility with well-understood mechanisms
  - Ease of programming, similar to uniprocessor
  - Low overhead for communicating small items
    - Memory mapping in hardware, not through OS
  - Can use hardware-controlled caching to reduce frequency of remote communication
- Message Passing Communication
  - Hardware can be simplified in some cases (we'll see coherent caching problems in a minute)
  - Communication is explicit, forcing programmers to pay attention and optimize (like delayed branch)
    - Could be a disadvantage as well!

15

Should Match SW to HW

- Message Passing model on shared memory architecture
  - Not too difficult, could "send" data by copying from one portion of the address space to another
- Shared Memory on Message Passing architecture
  - More difficult, without hw support for shared memory, the OS will need to handle things
  - High overhead for sending small loads and stores
- In either case, the resulting system will be slower than if the natural mapping from SW to HW is used

16

Communication Performance

- Communication Bandwidth
  - Data rate we can transmit data
  - Determined by communication hardware, mechanism
  - Slowest node for a data path can determine the communication bandwidth
- Communication Latency
  - Propagation time
  - Latency = Sender overhead + Time of Flight + Transmission time + Receiver overhead
  - Crucial metric to performance!
- Latency Hiding
  - Methods to hide latency by overlapping operations
  - But puts additional burden on software system and the programmer in many cases

17

Sample Remote Access Times

Large latency of remote access can significantly impact performance

Must take into account in designing algorithms!

Machine	Communication mechanism	Interconnection network	Processor count	Typical remote memory access time
SPARCCenter	Shared memory	Bus	≤ 20	1 µs
SGI Challenge	Shared memory	Bus	≤ 36	1 µs
Cray T3D	Shared memory	3D torus	32–2048	1 µs
Convex Exemplar	Shared memory	Crossbar + ring	8–64	2 µs
KSR-1	Shared memory	Hierarchical ring	32–256	2–6 µs
CM-5	Message passing	Fat tree	32–1024	10 µs
Intel Paragon	Message passing	2D mesh	32–2048	10–30 µs
IBM SP-2	Message passing	Multistage switch	2–512	30–100 µs

Load time for shared memory, Reply time for Message Passing

18

Performance Example

- Unfortunately, stringing together N processors with performance P does not give us N*P as the new performance
  - Factors coming into play
    - Amount of parallelism
    - Conventional factors (TLB miss, cache miss, etc.)
    - Shared memory overhead
    - Message passing overhead
- Can modify uniprocessor performance model:
  - CPUTime = IC * CPI * Parallel_Overhead * CycleTime

19

Communications Cost Example

- Multiprocessor with:
  - 2000ns to handle remote memory reference
  - All other references hit in local cache
  - Cycle time is 10ns
  - Base CPI is 1.0
  - How much faster if there is no communication vs. 0.5% of instructions involve remote communications?
- New effective CPI:
  - CPI(new) = Base_CPI + RemoteRequestRate * RemoteRequestCost
  - RemoteRequestCost = 2000ns / 10ns = 200 cycles
  - CPI(new) = 1.0 + (0.05)(200) = 2.0
- All local machine is twice as fast as the new machine
  - Means we'd like to limit communications as much as possible (e.g. cache)
  - Of course this doesn't include the work done by other processors in parallelizing an application!

20

Sample Machines

- Central Shared Memory
  - Sequent Symmetry S-81
    - Bus interconnect, thirty 386 CPU's with separate FPU
  - IBM ES/9000
    - Crossbar interconnect, 6 ES/9000 CPU's
  - BBN TC- 2000
    - Butterfly switch interconnect, 512 Motorola 68000 CPU's, hybrid NUMA architecture with preferred memory module
- Distributed Shared Memory
  - Intel Paragon
    - 2D mesh, 50 Mhz i860 CPU's, 128Mb per node, up to 2048 nodes
  - nCube
    - Hypercube, custom CISC CPU, 64Mb per node, up to 8196 nodes

21

Application Domain

- Multiprocessing performance is closely related to the application
  - Much more care must be taken in construction a *parallel algorithm* to take advantage of the hardware than for a uniprocessor machine
    - With uniprocessor, could rely on compiler techniques, hardware such as pipelining, etc. to help us out
    - Not so much of this help available for parallel machines, more of a burden on the programmer
  - Performance can vary significantly from one application to another, e.g.
    - Matrix multiplication – lots of places for parallelism
    - Computing a checksum – less room for parallelism, lots of dependencies on previous calculations

22

Example Problems

- Fast Fourier Transform
  - Convert signal from time to frequency domain
- LU Kernel
  - Solve linear algebra computations
- Barnes
- Ocean

23

Barnes

- Galaxy evolution, N-bodies with gravitational forces acting on them
- To reduce computational time required
  - Gravity drops off as square of the distance
  - Takes advantage of this property by treating “far away” bodies as a single point of combined mass at the centroid of the bodies, reducing N items to a single item
  - Each node represents an octree, or eight children representing eight cubes in space
  - Tree created to represent density of objects in space
- Challenges for parallelism
  - Each processor given some subtree to work on
  - Distribution of bodies is non-uniform and changes over time
  - So we must re-partition work among the processes to maintain balance
  - Requires communicating small amounts of data, implying shared-memory architecture may be most efficient

24

Ocean

- Ocean simulation, influence of eddy and currents on large-scale flow in the ocean
- To reduce computational time required
  - Ocean is broken up into grids, more grids gives more resolution and increases accuracy but requires more processing
  - Processing a grid cell requires data from neighboring cells
- Challenges for parallelism
  - Each processor given a grid cell to work on
  - Processors must communicate with their neighbors in a synchronized fashion before proceeding to the next step
  - Implies a DSM machine laid out in a mesh format would match nicely to this problem

25

Computation vs. Communication

- A key factor in the performance of parallel programs is the ratio of computation to communication
  - High implies lots of computation for each datum communicated (good since communication is expensive)
- Analysis of computation vs. communication varies on the problem and algorithm
  - Results shown on next slide for the four sample apps
  - We won't show how we arose at these figures (work like this left for the Algorithms class)
  - P = number of processors
  - N = data set size

26

Computation vs. Communication

Application	Scaling of computation	Scaling of communication	Scaling of computation-to-communication
FFT	$\frac{n\log n}{p}$	$\frac{n}{p}$	$\log n$
LU	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Barnes	$\frac{n\log n}{p}$	Approximately $\frac{\sqrt{n}(\log n)}{\sqrt{p}}$	Approximately $\frac{\sqrt{n}}{\sqrt{p}}$
Ocean	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$

Scaling on a per-processor basis

Computation: As P increases, computation goes down

Communication: As P increases, comm. goes down but less slowly than computation

Ratio: As P increases, computation-to-comm ratio goes down, which is bad. If data size the same, more inefficiencies in comm. Equation tells us how to balance N with P to maintain any desired amount of work spent in computation or comm

27

OS Workload

- There is also overhead with the OS
  - Just as we have with a uniprocessor
- Example on eight-processor system running "make"
  - Distribution of execution time:

Mode	% instructions executed	% execution time
Idle	69%	64%
User	27%	27%
Sync	1%	2%
Kernel	3%	7%

  - Most time actually spent idle waiting on the disk!
- Bottom line: Application behavior a key factor on performance with a parallel machine, thought must be given into the algorithm and performance-related issues

28