Memory Hierarchies and Thread Performance

• Last Time
  » Parallelism = Performance
  » Threads and Nodes as Basis for Large-scale Parallelism

• Today
  » Caches and Performance
  » Threads and Caches
  » Parallel Machine Structure

• Reminders/Announcements
  » Homework #2 went out Tuesday (Due April 21)
  » Homework #1 already returned (see Sagnik)

The Processor-Memory Gap

CSE 160 Chien, Spring 2005
Caches

- Caches are Critical to “span the gap”
  - Exploit Locality; Keep the needed data in a small fast memory
  - Avoid Accessing the Slow Memory
- If Successful
  - Processor can execute faster, memory requests are OFTEN satisfied quickly
  - Memory system can be much lower bandwidth, many fewer requests actually go to memory

A Typical Memory Hierarchy

- small expensive $/bit
  - on-chip caches
  - off-chip cache
  - main memory
- large cheap $/bit
Other Attributes of A Memory Hierarchy

- Low latency | High Bandwidth
  - Cache
  - L2 Cache
  - L3 Cache
  - on-chip caches
  - off-chip cache

- High Latency | Low Bandwidth
  - Memory
  - main memory

Finding Cache Data

- For fast access, must be able to find data quickly
- Lookup: discard a few address bits, access the fast memory
- Tags indicate the data that's really there
- Why does this work?
  - Memory location + tag -> complete address
Cache Access

- Part of Memory Address applied to cache
- Tags checked against remaining address bits
- If match -> Hit!, use data
- No match -> Miss, retrieve data from memory
- This works pretty well... but there are some complications...

Data Tags

Memory address lines = Hit?

Multiword Cache Entries

- Cache entries are several contiguous words of memory
- Shared tag, involves only contiguous addresses
  - reduces tag overhead
  - exploits spatial locality
- A few address lines select the word from the line on a "hit".... Which ones?
Accessing a Sample Cache

- **64 KB cache, direct-mapped, 32-byte cache block size**

  \[
  \frac{64\text{ KB}}{32\text{ bytes}} = \frac{2\text{ K cache blocks/sets}}{256}\text{ sets/bytes/word offset}
  \]

- **32 KB cache, 2-way set-associative, 16-byte block size**

  \[
  \frac{32\text{ KB}}{16\text{ bytes}} = \frac{1\text{ K cache sets}}{18\text{ sets/bytes/word offset}}
  \]
Example - HP/Compaq/DEC Alpha 21164 Caches

- ICache and DCache -- 8 KB, DM, 32-byte lines
- L2 cache -- 96 KB, 3-way SA, 32-byte lines
- L3 cache -- 1 MB, DM, 32-byte lines

Example

- 2.4 Ghz Dual Xeon Server
  » Dual Processors
  » L1 Cache – 8KB, 2-3 clocks, ~10GB/s
  » L2 Cache – 512KB, 6GB/s
  » Memory System – 100’s of clocks, 2GB/s shared

- Caches must be effective for this system to work well.
- What is the ratio, just on bandwidth alone? Block sizes…
- What type of working set is possible for an application? (how much locality)

This is comparable to the FWGrid Opteron Nodes!
Caches and Single Thread Performance

- So, why does this matter?
  - Single thread Performance is tied to cache performance
  - High Locality, high cache hit rate is essential
  - Low hit rate => low thread performance
  - Single thread performance is the building block for high parallel performance
- For good parallel performance, single threads must make good use of cache hierarchies

Multithreaded, Hyperthreading, or Simultaneous MT Processors

- Typically share the same memory hierarchy
  - Same caches, same "ports" to access the cache
  - No increased capacity, no increased bandwidth
  - Try to get higher throughput out of the processor
- Collective use of the Memory Hierarchy must have high locality
  - Capacity: Threads should have small "working sets", Overlapping working sets even better
  - Bandwidth: Not too many misses, or Miss bursts at different times
Multi-Processors: Cache Coherence Problem

- Multiple Processors Sharing a single Memory System
  - Separate cache hierarchy for each processor
- Copies of a datum may exist in multiple Caches
  - Writes take a finite amount of time to propagate
  - Caches might contain a value and respond while another processor is writing

Coherence Thru Write-back Caches

- First write captures an exclusive copy of the cache block
- Successive writes can be buffered in the cache; reducing the required bus bandwidth; key for scalability
- Cache must store exclusive and modified state, complicates the cache
- Eventually, the dirty, modified block must be written back into the main memory (or supplied to another cache)
- Basic scheme used for many machines...
Basic Write-back Cache Protocol

- Invalid, Shared, Dirty (exclusive); A/B → Get A, do B
- Dirty state is obtained through an ownership bus transaction, exiting this state requires a flush to write the data back
- BusRd gets a read copy, BusRdX gets a write copy
- Work out all of the transactions. Just about the simplest.

Illinois Protocol (MESI)

- Extend 3-state protocol with VE (valid, exclusive)
- Idea: Optimize BusRd and BusRdX into one transaction, keep track of private read copies.
- Natural extension of protocol is other ways.
- Used in SGI Challenge arrays and MOST multiprocessors…
Illinois Protocol (FSM)

- Other transitions identical to 3-state protocol
- "remembers" whether have the only copy

Complications -- Life ain’t so simple...

- Write buffers
  - Delay write visibility, how does this affect memory consistency model?
  - Aggregation, order can be changed as well. Flush instructions if necessary
- Split transaction busses
  - Operations are not atomic, may be many outstanding at the same time (again out of order)
  - No problem if the operations are not related (as above)
- Cache designs require simultaneous access
  - tag replication or dual porting
  - multi-level caches can alleviate this problem
- Reality: cache controllers compete with processors as the “really hard things to get right” in systems design
Caches and Multiple Processors

- So, why does this matter?
  - Coordination and data sharing amongst threads can be expensive
  - Cross-processor – hundreds of cycles; All the way down and all the way back up
  - *Ideally: Threads operate on independent data.*
    - Allow execution without interference in a single processor
    - Allow execution with high efficiency on separate processors

---

Parallel Machine Taxonomy
Flynn’s Taxonomy

- Flynn (1966) Classified machines by data and control streams

<table>
<thead>
<tr>
<th>Single Instruction</th>
<th>Single Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data</td>
<td>Multiple Data</td>
</tr>
<tr>
<td>(SISD)</td>
<td>SIMD</td>
</tr>
<tr>
<td></td>
<td>Multiple Instruction</td>
</tr>
<tr>
<td></td>
<td>Multiple Data</td>
</tr>
<tr>
<td></td>
<td>(MIMD)</td>
</tr>
</tbody>
</table>

SISD

- SISD
  - Model of serial von Neumann machine
  - Logically, single control processor
SIMD

- SIMD
  - All processors execute the same program in *lockstep*
  - Data that each processor sees may be different
  - Single control processor
  - Individual processors can be turned on/off at each cycle (“masking”)
  - CM-2, MasPar are some examples

CM2 Architecture

- CM2 had 8K-64K 1 bit custom processors
- Data Vault provides peripheral mass storage
- Programs had normal sequential control flow but all operations happened in parallel so CM HW supported data-parallel programming model
MIMD

- All processors execute their own set of instructions
- Processors operate on separate data streams
- No centralized clock implied
- MIMD machines may be shared memory or message-passing
- SP-2, T3E, Origin 2K, Tera MTA, Clusters, Cray’s, etc.
- => Most Machines built from traditional processors have this structure

Models for Communication

- Parallel program = program composed of tasks (processes) which communicate to accomplish an overall computational goal
- Two prevalent models for communication:
  » Message passing (MP)
  » Shared memory (SM)
Message Passing Communication

- Processes in message passing program communicate by passing messages

- Basic message passing primitives
  - Send(parameter list)
  - Receive(parameter list)
  - Parameters depend on the software and can be complex

Flavors of message passing

- Synchronous used for routines that return when the message transfer has been completed
  - Synchronous send waits until the complete message can be accepted by the receiving process before sending the message (send suspends until receive)
  - Synchronous receive will wait until the message it is expecting arrives (receive suspends until message sent)
  - Also called blocking

request to send
acknowledgement
message
Asynchronous (Non-blocking) Message Passing

- **Nonblocking sends** return whether or not the message has been received
  - If receiving processor not ready, message may be stored in message buffer
  - Message buffer used to hold messages being sent by A prior to being accepted by receive in B

- MPI:
  - Routines that use a message buffer and return after their local actions complete are **blocking** (even though message transfer may not be complete)
  - Routines that return immediately are **non-blocking**

**Architectural support for Message Passing**

- Interconnection network should provide connectivity, low latency, high bandwidth

- Many interconnection networks developed over last 2 decades
  - Small-size: non-blocking (Xbar)
  - Medium-size: non-blocking (Xbar)
  - Large-scale: Mesh, torus, multi-stage networks
Shared Memory Communication

- Processes in shared memory program communicate by accessing shared variables and data structures.
- Basic shared memory primitives:
  » Read to a shared variable (or object)
  » Write to a shared variable (or object)

Interconnection Network

Basic Shared Memory Multiprocessor Architecture

Accessing Shared Objects

- Conflicts may arise if multiple processes want to write to a shared variable at the same time.

Process A, B:
read x
compute x + 1
write x

- Programmer, language, and/or architecture must provide means of resolving conflicts.
Architectural Support for Shared Memory

- 4 basic types of interconnection media:
  » Bus (not really used any more)
  » Crossbar switch
  » Multistage network
  » Interconnection network with distributed shared memory

Limited Scalable Media

- Crossbar
  » Crossbar switch connects m processors and n memories with distinct paths between each processor/memory pair
  » Crossbar provides uniform access to shared memory (UMA)
  » $O(mn)$ switches required for m processors and n memories
  » Crossbar scalable in terms of performance but not in terms of cost, used for basic switching mechanism in SP2
Multistage Networks

- Multistage networks provide more scalable performance than bus but less costly to scale than crossbar
- Typically max{log n, log m} stages connect n processors and m shared memories
- "Omega" networks (butterfly, shuffle-exchange) commonly used for multistage network
- Multistage network used for CM-5 (fat-tree connects processor/memory pairs), BBN Butterfly (butterfly), IBM RP3 (omega)

Omega Networks

- Butterfly multistage
  » Used for BBN Butterfly, TC2000

- Shuffle multistage
  » Used for RP3, SP2 high performance switch
Fat-tree Interconnect

- Bandwidth is increased towards the root
- Used for data network for CM-5 (MIMD MPP)
  » 4 leaf nodes, internal nodes have 2 or 4 children
- To route from leaf A to leaf B, pick random switch C in the least common ancestor fat node of A and B, take unique tree route from A to C and from C to B

Binary fat-tree in which all internal nodes have two children

Distributed Shared Memory

- Memory is physically distributed but programmed as shared memory
  » Programmers find shared memory paradigm desirable
  » Shared memory distributed among processors, accesses may be sent as messages
  » Access to local memory and global shared memory creates NUMA (non-uniform memory access architectures)
  » BBN butterfly is NUMA shared memory multiprocessor
Using both Shared Memory and Message Passing together

- Clusters of SMPs may be effectively programmed using both SM and MP
  - Shared Memory used within a multiple processor machine/node
  - Message Passing used between nodes
- FWGrid has this structure

Summary

- Single Thread Performance is the building block for parallel performance
  - Threads should have high data locality, and share caches efficiently for MT, HT, SMT to work well
- Threads run on Multiprocessors must be decoupled to achieve good parallelism
  - High data locality, but moderate sharing of data
- Parallel Machine Taxonomy
  - SISD (traditional), SIMD, MIMD
  - Shared and Distributed Memory