CSE524 Parallel Computation

Lawrence Snyder
www.cs.washington.edu/CSEp524

3 April 2007

Administrivia

- Grading: 20% HW, 70% Proj, 10% Talk
- Burn your book!
- Consumer software + CMPs
  - MozoDojo, a graphic mosaic construction tool

**MultiThreading:** To make the best use possible of multi-core and multi-processor machines, MozoDojo is heavily multithreaded. All computations are dispatched on available processors. The difference is huge between a single G4 and a CoreDuo processor.

- There a HW assigned at end today

[Sign up for the email list]
Review

Value?

Rule for down flow?

Review Solution

Send parent value to left child; send parent + left child value to right child
Plan for today

- Parallel Hardware
  - Shared: Multicore, SMP
  - Distributed: Cluster, HPC Fire breather
- Models of Computation
  - RAM
  - PRAM
  - CTA
  - Reflection
- Communication modes
  - Shared, Message Passing, One Sided

Flynn’s Taxonomy

- Michael Flynn had an early way to classify machines, two forms of which are still used:

<table>
<thead>
<tr>
<th>Flynn’s Taxonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
</tr>
<tr>
<td>Multiple</td>
</tr>
<tr>
<td>Data Stream</td>
</tr>
</tbody>
</table>

  - SIMD -- single instruction, multiple data
  - MIMD -- multiple instruction, multiple data

  Our interest is exclusively with MIMD
Digression on SIMD

- Applying one instruction to multiple values...
  - Was important when memory was expensive
  - Powerful for tight, “inner loop” crunching
  - Performs in rigid lock-step w/apply/not protocol
  - SIMD implements most parallelism inefficiently

**Dilemma**: How to get SIMD crunching power with the flexibility needed to control program logic efficiently? **Cell** is perhaps an answer.

The Problem w/Parallel Architectures

- The problem with parallel machines is
  - They are different from sequential machines
  - They are different from each other
- Both problems complicate programming

- Our solution: Adopt a machine model that abstracts performance critical features

**But first, Let's look at some specific machines**
MESI Protocol

- Standard Protocol for cache-coherent shared memory
  - Mechanism for multiple caches to give single memory image
  - We will not study it
  - 4 states can be amazingly rich

Thanks: Slater & Tibrewala of CMU

MESI, Intuitively

- Upon loading, a line is marked E, subsequent reads are OK; write marks M
- Seeing another load, mark as S
- A write to an S, sends I to all, marks as M
- Another’s read to an M line, writes it back, marks it S
- Read/write to an I misses
- Related scheme: MOESI (used by AMD)
AMD Dual Core Operton
Comparing Core Duo/Dual Core

Front Side Bus
Memory Bus Controller
L2 Cache
Intel
L1-I L1-D L1-I L1-D
Processor P0 Processor P1

Mem Ctrlr
Cross-Bar Interconnect
System Request Interface
L2 Cache
AMD
L1-I L1-D
Processor P0
L2 Cache
AMD
L1-I L1-D
Processor P1

Comparing Core Duo/Dual Core

Front Side Bus
Memory Bus Controller
L2 Cache
Intel
L1-I L1-D L1-I L1-D
Processor P0 Processor P1

Mem Ctrlr
Cross-Bar Interconnect
System Request Interface
L2 Cache
AMD
L1-I L1-D
Processor P0
L2 Cache
AMD
L1-I L1-D
Processor P1

Mem Ctrlr
Cross-Bar Interconnect
System Request Interface
L2 Cache
AMD
L1-I L1-D
Processor P0
L2 Cache
AMD
L1-I L1-D
Processor P1
SMP on a Bus

- The bus is a point that serializes references
- A serializing point is a shared mem enabler
Sun Fire E25K

Cross-Bar Switch

- A crossbar is a network connecting each processor to every other processor
- Used in CMU’s 1971 C.MMP, 16 proc PDP-11s
- Crossbars grow as $n^2$ making them impractical for large $n$
Sun Fire E25K

- X-bar gives low latency for snoops allowing for shared memory
- 18 x 18 X-bar is basically the limit
- Raising the number of processors per node will, on average, increase congestion
- How could we make a larger machine?

Co-Processor Architectures

- A powerful parallel design is to add 1 or more subordinate processors to std design
  - Floating point instructions once implemented this way
  - Graphics Processing Units - deep pipelining
  - Cell Processor - multiple SIMD units
  - Attached FPGA chip(s) - compile to a circuit

- These architectures will be discussed later
Clustering systems using InfiniBand Hardware

- Interconnecting with InfiniBand
- Switch-based technology
  - Host channel adapters (HCA)
  - Peripheral computer interconnect (PC)

Thanks: IBM’s Clustering systems using InfiniBand Hardware

Clustering systems using InfiniBand Hardware

- Cheap to build using commodity technologies
- Effective when interconnect is “switched”
- Easy to extend, usually in increments of 1
- Processors often have disks “nearby”
- No shared memory
- Latencies are usually large
- Programming uses message passing
Supercomputer

- **BlueGene/L**

BlueGene/L Specs

- A 64x32x32 torus = 65K 2-core processors
- Cut-through routing gives a worst-case latency of 6.4 µs
- Processor nodes are dual PPC-440 with “double hummer” FPUs
- Collective network performs global reduce for the “usual” functions
- #1 on November’s Top 500 at 280 TF
Summarizing Architectures

- Two main classes
  - Complete connection: CMPs, SMPs, X-bar
    - Preserve single memory image
    - Complete connection limits scaling to …
    - Available to everyone
  - Sparse connection: Clusters, Supercomputers, Networked computers used for parallelism (Grid)
    - Separate memory images
    - Can grow “arbitrarily” large
    - Available to everyone with air conditioning
  - Differences are significant; world views diverge

The Parallel Programming Problem

- Some computations can be platform specific
- Most should be platform independent
- Parallel Software Development Problem: How do we neutralize the machine differences given that
  - Some knowledge of execution behavior is needed to write programs that perform
  - Programs must port across platforms effortlessly, meaning, by at most recompilation
Options for Solving the PPP

- Leave the problem to the compiler …
  - Very low level parallelism (ILP) is already being exploited
  - Sequential languages cause us to introduce unintentional sequentiality
  - Parallel solutions often require a paradigm shift
  - Compiler writers’ track record over past 3 decades not promising … recall HPF
  - Bottom Line: Compilers will get more helpful, but they probably won’t solve the PPP
Options for Solving the PPP

- Adopt a very abstract language that can target to any platform ...

- No one wants to learn a new language, no matter how cool
- How does a programmer know how efficient or effective his/her code is? Interpreted code?
- What are the “right” abstractions and statement forms for such a language?
  - Emphasize programmer convenience?
  - Emphasize compiler translation effectiveness?
Options for Solving the PPP

- Agree on a set of parallel primitives (spawn process, lock location, etc.) and create libraries that work w/ sequential code …

- Libraries are a mature technology
- To work with multiple languages, limit base language assumptions … L.C.D. facilities
- Libraries use a stylized interface (fcn call) limiting parallelism-specific abstractions possible
- Achieving consistent semantics is difficult
Options for Solving the PPP

- Create an abstract machine model that accurately describes common capabilities and let the language facilities catch up …

- Not a full solution until languages are available
- The solution works in sequential world (RAM)
- Requires discovering (and predicting) what the common capabilities are
- Solution needs to be (continually) validated against actual experience
Options for Solving the PPP

- Leave the problem to the compiler … ●
- Adopt a very abstract language that can target to any platform … ●
- Agree on a set of parallel primitives (spawn process, lock location, etc.) and create libraries that work w/ sequential code … ●
- Create an abstract machine model that accurately describes common capabilities and let the language facilities catch up … ●

---

Break

---
Reason by Analogy: RAM Model

- The Random Access Machine is our friend
  - Control, ALU, (Unlimited) Memory, [Input, Output]
  - Fetch/execute cycle runs 1 inst. pointed at by PC
  - Memory references are “unit time” independent of location
    - Gives RAM it’s name in preference to von Neumann
    - “Unit time” is not literally true, but caches fake it
  - Executes “3-address” instructions

It’s so intuitive, it seems like there’s no other way to compute!

How To Use the RAM

- When reasoning about performance …
  - Worry about how many instructions executed because execution time proportional to cycles
  - Treat memory references (operand fetch) as a negligible part of the instruction execution
  - Estimate time and space needs based on increasing problem size, $O(n)$
    - Linear search vs Binary search
  - Crucial to effectively using C, etc.
Generalization of RAM: PRAM

- Parallel Random Access Machine (PRAM)
  - Unlimited number of processors
  - Processors are standard RAM machines, executing synchronously
  - Memory reference is “unit time”
  - Outcome of collisions at memory specified
    - EREW, CREW, CRCW …
  - Model fails bc synchronous execution w/ unit cost memory reference does not scale

CTA Model

- Candidate Type Architecture: A model with $P$ standard processors, $d$ degree, $\lambda$ latency

- Node == processor + memory + NIC

  Key Property: Local memory ref is 1, global memory is $\lambda$
What CTA Doesn’t Describe

- CTA has no global memory … but memory could be globally *addressed*
- Mechanism for referencing memory not specified: shared, message passing, 1-side
- Interconnection network not specified
- $\lambda$ is not specified beyond $\lambda >> 1$ -- cannot be because every machine is different
- Controller, combining network “optional”

Typical Values for $\lambda$

- Lambda can be estimated for any machine (given numbers include no contention or congestion)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>System Configuration</th>
<th>$\lambda$ Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>AMD</td>
<td>100</td>
</tr>
<tr>
<td>SMP</td>
<td>Sun Fire E25K</td>
<td>400-660</td>
</tr>
<tr>
<td>Cluster</td>
<td>Itanium + Myrinet</td>
<td>4100-5100</td>
</tr>
<tr>
<td>Super</td>
<td>BlueGene/L</td>
<td>5000</td>
</tr>
</tbody>
</table>

As with merchandizing: It’s location, location, location!
PRAM Mispredicts Preferred Alg

- Consider finding maximum of $n$ numbers
- Best algorithm
  - CRCW PRAM: Valiant’s algorithm $O(\log \log n)$
  - CTA Model: Tournament algorithm $O(\log n)$
- Observed performance real implementation
  - PRAM: $O(\log n \log \log n)$
  - CTA: $O(\log n)$
- We do not want a model that directs us to an impractical solution

Apply CTA to Count 3s

- How does CTA guide us for Count 3s pgm
  - Array segments will be allocated to local mem
  - Each processor should count 3s in its segment
  - Global total should be formed using reduction
  - Performance is
    - Full parallelism for local processing
    - $\lambda \log n$ for combining (and broadcast)
    - Base of log should be large, i.e high degree nodes
- Same solution as before, but by different rt
Communication Mechanisms

- Shared addressing
  - One consistent memory image; primitives are load and store
  - Must protect locations from races
  - Widely considered most convenient, though it is often tough to get a program to perform
  - CTA implies that best practice is to keep as much of the problem private; use sharing only to communicate

  A common pitfall: Logic is too fine grain

Communication Mechanisms

- Message Passing
  - No global memory image; primitives are send() and recv()
  - Required for most large machines
  - User writes in sequential language with message passing library:
    - Message Passing Interface (MPI)
    - Parallel Virtual Machine (PVM)
  - CTA implies that best practice is to build and use own abstractions

  Lack of abstractions makes message passing brutal
Communication Mechanisms

- One Sided Communication
  - One global address space; primitives are get() and put()
  - Consistency is the programmer’s responsibility
  - Elevating mem copy to a comm mechanism
  - Programmer writes in sequential language with library calls -- not widely available unfortunately
  - CTA implies that best practice is to build and use own abstractions

One-sided is lighter weight than message passing

Assignment for next week

- Read Chapter 3
- Homework Problem: Analyze the complexity of the Odd/Even Interchange Sort: Given array A[n], exchange o/e pairs if not ordered, then exchange e/o pairs if not ordered, then repeat until sorted
- Analyze in CTA model (i.e. for \( P, \lambda, d \)), and charge the o/e-e/o pair \( c \) time if operands are local; ignore all other local computation