Part II: Architecture

Goal: Understand the main properties of parallel computers

What’s The Deal With Hardware?

- Facts Concerning Hardware
  - Parallel computers differ dramatically from each other -- there is no standard architecture
  - No single programming target!
  - Parallelism introduces costs not present in vN machines -- communication; influence of external events
  - Many parallel architectures have failed
  - Details of parallel computer are of no greater concern to programmers than details of vN

The “no single target” is key problem to solve
Our Plan

- Though computers of little concern, we ground our thinking in reality
- Introduce instances of basic designs
  - Multicore
  - Symmetric Multiprocessors (SMPs)
  - Large scale parallel machines
  - Clusters
  - Blue Gene/L
  - Attached processors: Cell, GPGPUs, FPGA

Multi-core Chips

- Multi-core means more than one processor per chip
- Consequence of Moore’s Law
- IBM’s PowerPC 2002, AMD Dual Core Opteron 2005, Intel CoreDuo 2006
- A small amount of multi-threading included
- Main advantage: More ops per tick
- Main disadvantages: Programming, BW
Moore’s Law In Action

- 2 32-bit Pentiums
- Private 32K L1s
- Shared 2M-4M L2
- MESI cc-protocol
- Shared bus control and memory bus

Intel CoreDuo

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

- 2 64-bit Opterons
- 64K private L1s
- 1 MB private L2s
- MOESI cc-protocol
- Direct connect shared memory
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 Ctrl | HT

Cross-Bar Interconnect

System Request Interface

L2 Cache
AMD | AMD

L1-I | L1-D | L1-I | L1-D

Processor P0 | Processor P1

Cross-Bar Interconnect

System Request Interface

L2 Cache
AMD | AMD | AMD | AMD

L1-I | L1-D | L1-I | L1-D

Processor P0 | Processor P1

Mem Ctrl

Front Side Bus
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

Clusters

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

Thanks: IBM’s Clustering systems using InfiniBand Hardware
Clusters

- 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

Networks
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 possible parallelism-specific abstractions
- 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
Summary of Options for 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 … ●

Today’s Plan

- Goal from last time: Adopt a parallel machine model for use in thinking about algorithms and programming
- First step: Review how we use von Neumann
- Second step: Introduce the CTA
- Third step: Discuss how it relates to machines of last time
- Communication methods and CTA
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 the effective use of 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

---

PRAM is too abstract, but not irrelevant

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”

More On the CTA

- Consider what the diagram means…

More On the CTA

☐ Consider what the diagram means…

---

More On the CTA

☐ Consider what the diagram means…
More On the CTA

- Consider what the diagram doesn’t mean…

- After ACKing that CTA doesn’t model buses, accept that it’s a good first approx.

Typical Values for $\lambda$

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Model</th>
<th>$\lambda$</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>

$\text{Lg } \lambda \text{ range => cannot be ignored}$

As with merchandizing: It’s location, location, location!
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

Programming Implications

- How does CTA influence programming …
- Discuss
  - Expression evaluation: Same/Different?
  - Relationship among processors?
  - Data structures?
  - Organization of work?
  - …
Compare CTA and PRAM Models

- Consider algorithm to find max of \{a_1...a_n\}
- Valiant’s algorithm is best PRAM algorithm

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1:a2</td>
<td>a1:a3</td>
<td>a2:a3</td>
<td>a4:a5</td>
<td>a4:a6</td>
<td>a5:a6</td>
</tr>
<tr>
<td>set smaller to 0 in ans; promote winners to next phase</td>
<td>set smaller to 0 in ans; promote winners to next phase</td>
<td>set smaller to 0 in ans; promote winners to next phase</td>
<td>set smaller to 0 in ans; promote winners to next phase</td>
<td>set smaller to 0 in ans; promote winners to next phase</td>
<td>...</td>
</tr>
</tbody>
</table>

- \(P/3\) groups require \(3^*P/3 = P\) processors
- Requires \(\log \log n\) stages of const time

- Valiant’s totally clever PRAM algorithm: Exploits 1 tick mem ref

**PRAM Mispredicts Preferred Alg**

- For task of 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
    - Base of log should be large, i.e high degree nodes
- Same solution as before, but by different rt

Another algorithm

- How does CTA guide us to programming Odd/Even Transposition Sort?
  - Given array of items $A[n]$
  - Continue, until sorted, to
    - compare odd/even pairs (e.g. 1:2) and exchanging if out of order
    - compare even/odd pairs (e.g. 2:3) and exchanging if out of order
- In Knuth, it’s a for-loop inside a while-loop
- How should it be programmed for CTA?
Assignment for Friday

- Read 0.5 Chapter 3 (middle p. 73)
- 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, repeating until sorted
- Program it and analyze in CTA model (i.e. for $P, \lambda, d$), charging $c$ time if exchange operands local; ignore all other local computation

Plan For Coming Weeks

- Classroom discussion/programming in ||
- Read Chapter 11 to page 311
- Direct threads programming: PThreads or Java threads
  - First program, due Monday is PThreads program running on CMP or SMP to solve count 3s -- mostly logistics
  - More involved program a week later
- Then on to message passing