CS4961 Parallel Programming

Lecture 3: Introduction to Parallel Architectures

Mary Hall
August 30, 2011

---

Homework 1: Parallel Programming Basics

Turn in electronically on the CADE machines using the handin program: "handin cs4961 hw1 <probfile>

Problem 1: (#1.3 in textbook): Try to write pseudo-code for the tree-structured global sum illustrated in Figure 1.1. Assume the number of cores is a power of two (1, 2, 4, 8, …). Hints: Use a variable \( \text{divisor} \) to determine whether a core should send its sum or receive and add. The \( \text{divisor} \) should start with the value 2 and be doubled after each iteration. Also use a variable \( \text{core\_difference} \) to determine which core should be partnered with the current core. It should start with the value 1 and also be doubled after each iteration. For example, in the first iteration 0 \( \text{divisor} = 0 \) and 1 \( \text{divisor} = 1 \), so 0 receives and adds, while 1 sends. Also, in the first iteration 0 + \( \text{core\_difference} = 1 \) and 1 - \( \text{core\_difference} = 0 \), so 0 and 1 are paired in the first iteration.

---

Administrative UPDATE

- Nikhil office hours:
  - Monday, 2-3 PM, MEB 3115 Desk #12
  - Lab hours on Tuesday afternoons during programming assignments
- We’ll spend some class time going over the homework
- Next homework will be due next Thursday, Sept. 8
  - I’ll post the assignment by Thursday’s class, maybe earlier

---

Homework 1: Parallel Programming Basics

Problem 2: I recently had to tabulate results from a written survey that had four categories of respondents: (I) students; (II) academic professionals; (III) industry professionals; and, (IV) other. The number of respondents in each category was very different; for example, there were far more students than other categories. The respondents selected to which category they belonged and then answered 32 questions with five possible responses: (i) strongly agree; (ii) agree; (iii) neutral; (iv) disagree; and, (v) strongly disagree. My family members and I tabulated the results “in parallel” (assume there were four of us).

(a) Identify how data parallelism can be used to tabulate the results of the survey. Keep in mind that each individual survey is on a separate sheet of paper that only one “processor” can examine at a time. Identify scenarios that might lead to load imbalance with a purely data parallel scheme.

(b) Identify how task parallelism and combined task and data parallelism can be used to tabulate the results of the survey to improve upon the load imbalance you have identified.
Today’s Lecture

• Flynn’s Taxonomy
• Some types of parallel architectures
  - Shared memory
  - Distributed memory
• These platforms are things you will probably use
  - CADE Lab1 machines (Intel Nehalem i7)
  - Sun Ultrasparc T2 (water, next assignment)
  - Nvidia GTX260 GPUs in Lab1 machines
• And for fun, Jaguar, the fastest computer in the US
• Sources for this lecture:
  - Textbook
  - Jim Demmel, UC Berkeley
  - Notes on various architectures

Reading this week: Chapter 2.1-2.3 in textbook

Chapter 2: Parallel Hardware and Parallel Software

2.1 Some background
• The von Neumann architecture
• Processes, multitasking, and threads

2.2 Modifications to the von Neumann Model
• The basics of caching
• Cache Mappings
• Caches and programs: an example
• Virtual memory
• Instruction-level parallelism
• Hardware multithreading

2.3 Parallel Hardware
• SIMD systems
• MIMD systems
• Interconnection networks
• Cache coherence
• Shared-memory versus distributed-memory

An Abstract Parallel Architecture

• How is parallelism managed?
• Where is the memory physically located?
• Is it connected directly to processors?
• What is the connectivity of the network?

Why are we looking at a bunch of architectures

• There is no canonical parallel computer – a diversity of parallel architectures
  - Hence, there is no canonical parallel programming language
• Architecture has an enormous impact on performance
  - And we wouldn’t write parallel code if we didn’t care about performance
• Many parallel architectures fail to succeed commercially
  - Can’t always tell what is going to be around in N years

Challenge is to write parallel code that abstracts away architectural features, focuses on their commonality, and is therefore easily ported from one platform to another.
Retrospective (Jim Demmel)

- Historically, each parallel machine was unique, along with its programming model and programming language.
- It was necessary to throw away software and start over with each new kind of machine.
- Now we distinguish the programming model from the underlying machine, so we can write portably correct codes that run on many machines.
- Parallel algorithm design challenge is to make this process easy.
  - Still somewhat of an open research problem

The von Neumann Architecture

Conceptually, a von Neumann architecture executes one instruction at a time

Locality and Parallelism

- Large memories are slow, fast memories are small
- Program should do most work on local data
Uniprocessor and Parallel Architectures

Achieve performance by addressing the von Neumann bottleneck

- Reduce memory latency
  - Access data from "nearby" storage: registers, caches, scratchpad memory
  - We’ll look at this in detail in a few weeks
- Hide or Tolerate memory latency
  - Multithreading and, when necessary, context switches while memory is being serviced
  - Prefetching, predication, speculation
- Uniprocessors that execute multiple instructions in parallel
  - Pipelining
  - Multiple issue
  - SIMD multimedia extensions

How Does a Parallel Architecture Improve on this Further?

- Computation and data partitioning focus a single processor on a subset of data that can fit in nearby storage
- Can achieve performance gains with simpler processors
  - Even if individual processor performance is reduced, throughput can be increased
- Complements instruction-level parallelism techniques
  - Multiple threads operate on distinct data
  - Exploit ILP within a thread

Flynn’s Taxonomy

<table>
<thead>
<tr>
<th>SISD</th>
<th>(SIMD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single instruction stream</td>
<td>Single instruction stream</td>
</tr>
<tr>
<td>Single data stream</td>
<td>Multiple data stream</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MISD</th>
<th>(MIMD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple instruction stream</td>
<td>Multiple instruction stream</td>
</tr>
<tr>
<td>Single data stream</td>
<td>Multiple data stream</td>
</tr>
</tbody>
</table>

Classical von Neumann

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Instruction, Multiple Data (MIMD)</td>
<td>Multiple threads of control, processors periodically synch</td>
<td>parallel loop: [ \text{for all } i=0; i&lt;n; i++ ]</td>
</tr>
<tr>
<td>Single Program, Multiple Data (SPMD)</td>
<td>Multiple threads of control, but each processor executes same code</td>
<td>processor-specific code: if ( ($myid == 0) ) { }</td>
</tr>
</tbody>
</table>
Two main classes of parallel architecture organizations

- **Shared memory multiprocessor architectures**
  - A collection of autonomous processors connected to a memory system.
  - Supports a global address space where each processor can access each memory location.
- **Distributed memory architectures**
  - A collection of autonomous systems connected by an interconnect.
  - Each system has its own distinct address space, and processors must explicitly communicate to share data.
  - Clusters of PCs connected by commodity interconnect is the most common example.

### Programming Shared Memory Architectures

A shared-memory program is a collection of threads of control.
- Threads are created at program start or possibly dynamically.
- Each thread has **private variables**, e.g., local stack variables.
- Also a set of **shared variables**, e.g., static variables, shared common blocks, or global heap.
- Threads communicate **implicitly** by writing and reading **shared variables**.
- Threads coordinate through **locks** and **barriers** implemented using shared variables.

#### Shared Memory Architecture 1:
Intel i7 860 Nehalem (CADE LAB1)

- 256KB L2 Unified Cache
- 32KB L1 Instr Cache
- 32KB L1 Data Cache
- Up to 16 GB Main Memory (DDR3 Interface)
**More on Nehalem and Lab1 machines -- ILP**

- Target users are general-purpose
- Personal use
- Games
- High-end PCs in clusters
- Support for SSE 4.2 SIMD instruction set
- 8-way hyperthreading (executes two threads per core)
- Multiscalar execution (4-way issue per thread)
- Out-of-order execution
- Usual branch prediction, etc.

**Shared Memory Architecture 2: Sun Ultrasparc T2 Niagara (water)**

**More on Niagara**

- Target applications are server-class, business operations
- Characterization:
  - Floating point?
  - Array-based computation?
- Support for VIS 2.0 SIMD instruction set
- 64-way multithreading (8-way per processor, 8 processors)
- ...

**Shared Memory Architecture 3: GPUs**

Lab1 has Nvidia GTX 260 accelerators

24 Multiprocessors, with 8 SIMD processors per multiprocessor

- SIMD Execution of warpsize threads
  (from single block)
- Multithreaded Execution across different instruction streams

Complex and largely programmer-controlled memory hierarchy

- Shared Device memory
- Per-multiprocessor “Shared memory”
- Some other constrained memories (constant and texture memories/caches)
- No standard data cache
Jaguar (3rd fastest computer in the world)

Peak performance of 2.33 Petaflops
224,256 AMD Opteron cores

http://www.olcf.ornl.gov/computing-resources/jaguar/

Shared Memory Architecture 4:
Each Socket is a 12-core AMD Opteron Istanbul

- 6-core “Processor”
- 6-core “Processor”
- Hyper Transport Link (Interconnect)
- Shared 6MB L3 Cache
- 8 GB Main Memory (DDR3 Interface)

Shared Memory Architecture 3:
Each Socket is a 12-core AMD Opteron Istanbul

- 64KB L1 Data Cache
- 64KB L1 Data Cache
- 64KB L1 Data Cache
- 64KB L1 Data Cache
- 64KB L1 Data Cache
- 64KB L1 Data Cache

- 64KB L1 Instr Cache
- 64KB L1 Instr Cache
- 64KB L1 Instr Cache
- 64KB L1 Instr Cache
- 64KB L1 Instr Cache
- 64KB L1 Instr Cache

- Proc
- Proc
- Proc
- Proc
- Proc
- Proc

- 512 KB L2 Unified Cache
- 512 KB L2 Unified Cache
- 512 KB L2 Unified Cache
- 512 KB L2 Unified Cache
- 512 KB L2 Unified Cache
- 512 KB L2 Unified Cache

Hyper Transport Link (Interconnect)

Shared 6MB L3 Cache

8 GB Main Memory (DDR3 Interface)

Jaguar is a Cray X15 (plus X14)
Interconnect is a 3-d mesh

3-dimensional toroidal mesh

Summary of Architectures

Two main classes

• Complete connection: CMPs, SMPs, X-bar
  - Preserve single memory image
  - Complete connection limits scaling to small number of processors (say, 32 or 256 with heroic network)
  - Available to everyone (multi-core)

• Sparse connection: Clusters, Supercomputers, Networked computers used for parallelism
  - Separate memory images
  - Can grow "arbitrarily" large
  - Available to everyone with LOTS of air conditioning

• Programming differences are significant

Brief Discussion

• Why is it good to have different parallel architectures?
  - Some may be better suited for specific application domains
  - Some may be better suited for a particular community
  - Cost
  - Explore new ideas

• And different programming models/languages?
  - Relate to architectural features
  - Application domains, user community, cost, exploring new ideas

Summary of Lecture

• Exploration of different kinds of parallel architectures
  - And impact on programming models

• Key features
  - How processors are connected?
  - How memory is connected to processors?
  - How parallelism is represented/managed?

• Next Time
  - Memory systems and interconnect
  - Models of memory and communication latency