CS4961 Parallel Programming

Lecture 4:
CTA, cont.
Data and Task Parallelism

Mary Hall
September 2, 2010

Homework 2, Due Friday, Sept. 10, 11:59 PM

- To submit your homework:
  - Submit a PDF file
  - Use the "handin" program on the CADE machines
  - Use the following command:
    "handin cs4961 hw2 <prob1file>

Problem 1 (based on #1 in text on p. 59):

Consider the Try2 algorithm for "count3s" from Figure 1.9 of p.19 of the text. Assume you have an input array of 1024 elements, 4 threads, and that the input data is evenly split among the four processors so that accesses to the input array are local and have unit cost. Assume there is an even distribution of appearances of 3 in the elements assigned to each thread which is a constant we call NTPT. What is a bound for the memory cost for a particular thread predicted by the CTA expressed in terms of λ and NTPT.

Homework 2, cont

Problem 2 (based on #2 in text on p. 59), cont.:

Now provide a bound for the memory cost for a particular thread predicted by CTA for the Try4 algorithm of Fig. 114 on p. 23 (or Try3 assuming each element is placed on a separate cache line).

Problem 3:

For these examples, how is algorithm selection impacted by the value of NTPT?

Problem 4 (in general, not specific to this problem):

How is algorithm selection impacted by the value of λ?

Brief Recap of Course So Far

- Technology trends that make parallel computing increasingly important
- History from scientific simulation and supercomputers
- Data dependences and reordering transformations
  - Fundamental Theory of Dependence
- Tuesday, we looked at a lot of different kinds of parallel architectures
  - Diverse!
  - Shared memory vs. distributed memory
  - Scalability through hierarchy

Today's Lecture

- How to write software for a moving hardware target?
  - Abstract away specific details
  - Want to write machine-independent code
- Candidate Type Architecture (CTA) Model
  - Captures inherent tradeoffs without details of hardware choices
  - Summary: Locality is Everything!
- Data parallel and task parallel constructs and how to express them
- Sources for this lecture:
  - Larry Snyder, http://www.cs.washington.edu/education/courses/524/08wi/
  - Grama et al., Introduction to Parallel Computing, http://www.cs.umn.edu/~karypis/parbook

Parallel Architecture Model

- How to develop portable parallel algorithms for current and future parallel architectures, a moving target?
- Strategy:
  - Adopt an abstract parallel machine model for use in thinking about algorithms

1. Review how we compare algorithms on sequential architectures
2. Introduce the CTA model (Candidate Type Architecture)
3. Discuss how it relates to today's set of machines

How did we do it for sequential architectures?

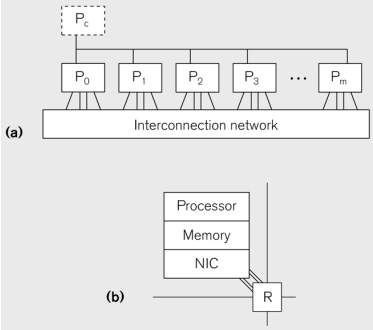
- Sequential Model: Random Access Machine
  - Control, ALU, (Unlimited) Memory, [Input, Output]
  - Fetch/execute cycle runs 1 inst. pointed at by PC
  - Memory references are "unit time" independent of location
    - Gives RAM it's name in preference to von Neumann
    - "Unit time" is not literally true, but caches provide that illusion when effective
  - Executes "3-address" instructions
- Focus in developing sequential algorithms, at least in courses, is on reducing amount of computation (useful even if imprecise)
  - Treat memory time as negligible
  - Ignore overheads

Interesting Historical Parallel Architecture Model, 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 to capture true performance behavior
  - Synchronous execution w/ unit cost memory reference does not scale
  - Therefore, parallel hardware typically implements non-uniform cost memory reference

Candidate Type Architecture (CTA Model)

- A model with *P standard processors, d degree, λ latency*
- Node == processor + memory + NIC
- Key Property: Local memory ref is 1, global memory is λ



Estimated Values for Lambda

- Captures inherent property that data locality is important.
- But different values of Lambda can lead to different algorithm strategies

CMP	AMD	100
SMP	Sun Fire E25K	400-660
Cluster	Itanium + Myrinet	4100-5100
Super	BlueGene/L	5000

Lg λ range => cannot be ignored

Locality Rule

- Definition, p. 53:
  - Fast programs tend to maximize the number of local memory references and minimize the number of non-local memory references.
- Locality Rule in practice
  - It is usually more efficient to add a fair amount of redundant computation to avoid non-local accesses (e.g., random number generator example).

This is the most important thing you need to learn in this class!

Memory Reference Mechanisms

- Shared Memory
  - All processors have access to a global address space
  - Refer to remote data or local data in the same way, through normal loads and stores
  - Usually, caches must be kept *coherent* with global store
- Message Passing & Distributed Memory
  - Memory is partitioned and a partition is associated with an individual processor
  - Remote data access through explicit communication (sends and receives)
  - Two-sided (both a send and receive are needed)
- One-Sided Communication (a hybrid mechanism)
  - Supports a global shared address space but no coherence guarantees
  - Access to remote data through gets and puts

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

Conceptual: CTA for Shared Memory Architectures?

- CTA is not capturing global memory in SMPs
- Forces a discipline
  - Application developer should think about locality even if remote data is referenced identically to local data!
  - Otherwise, performance will suffer
  - Anecdotally, codes written for distributed memory shown to run faster on shared memory architectures than shared memory programs
  - Similarly, GPU codes (which require a partitioning of memory) recently shown to run well on conventional multi-core

Definitions of Data and Task Parallelism

- Data parallel computation:
  - Perform the same operation to different items of data at the same time; the parallelism grows with the size of the data.
- Task parallel computation:
  - Perform distinct computations -- or tasks -- at the same time; with the number of tasks fixed, the parallelism is not scalable.
- Summary
  - Mostly we will study data parallelism in this class
  - Data parallelism facilitates very high speedups; and scaling to supercomputers.
  - Hybrid (mixing of the two) is increasingly common

Parallel Formulation vs. Parallel Algorithm

- Parallel Formulation
  - Refers to a parallelization of a serial algorithm.
- Parallel Algorithm
  - May represent an entirely different algorithm than the one used serially.
- In this course, we primarily focus on "Parallel Formulations".

Steps to Parallel Formulation (refined from Lecture 2)

- Computation Decomposition/Partitioning:
  - Identify pieces of work that can be done concurrently
  - Assign tasks to multiple processors (processes used equivalently)
- Data Decomposition/Partitioning:
  - Decompose input, output and intermediate data across different processors
- Manage Access to shared data and synchronization:
  - coherent view, safe access for input or intermediate data

UNDERLYING PRINCIPLES:

- Maximize concurrency and reduce overheads due to parallelization!
- Maximize potential speedup!

Concept of Threads

- This week's homework was made more difficult because we didn't have a concrete way of expressing the parallelism features of our code!
- Text introduces Peril-L as a neutral language for describing parallel programming constructs
  - Abstracts away details of existing languages
  - Architecture independent
  - Data parallel
  - Based on C, for universality
- Next week we will instead learn OpenMP
  - Similar to Peril-L

Common Notions of Task-Parallel Thread Creation (not in Peril-L)

- cobegin/coend

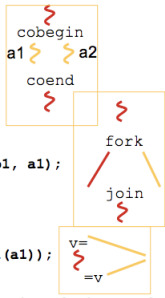
```
cobegin
    job1(a1);
    job2(a2);
coend
```

- fork/join

```
tid1 = fork(job1, a1);
job2(a2);
join tid1;
```

- future

```
v = future(job1(a1));
… = …v…;
```



- Statements in block may run in parallel
- cobegins may be nested
- Scoped, so you cannot have a missing coend

- Forked procedure runs in parallel
- Wait at join point if it's not finishedC

- Future expression evaluated in parallel
- Attempt to use return value will wait

- Cobegin cleaner than fork, but fork is more general
- Futures require some compiler (and likely hardware) support

Examples of Task and Data Parallelism

- Looking for all the appearances of "University of Utah" on the world-wide web
- A series of signal processing filters applied to an incoming signal
- Same signal processing filters applied to a large known signal
- Climate model from Lecture 1

Summary of Lecture

- CTA model
- How to develop a parallel code
  - Locality is everything!
  - Added in data decomposition
- Next Time:
  - Our first parallel programming language!
  - Data Parallelism in OpenMP