CS4230 Parallel Programming

Lecture 2: Introduction to Parallel Algorithms

Mary Hall
August 23, 2012

Homework 1: Parallel Programming Basics

Due before class, Thursday, August 30

Turn in electronically on the CADE machines using the handin program: "handin cs4230 hw1 <probfile>"

- Problem 1: (from today's lecture) We can develop a model for the performance behavior from the versions of parallel sum in today's lecture based on sequential execution time S, number of threads T, parallelization overhead O (fixed for all versions), and the cost B for the barrier or M for each invocation of the mutex. Let N be the number of elements in the list. For version 5, there is some additional work for thread 0 that you should also model using the variables above. (a) Using these variables, what is the execution time of valid parallel versions 2, 3 and 5; (b) present a model of when parallelization is profitable for version 3; (c) discuss how varying T and N impact the relative profitability of versions 3 and 5.

Homework 1: Parallel Programming Basics

- Problem 2: (#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 `divisor` to determine whether a core should send its sum or receive and add. The `divisor` should start with the value 2 and be doubled after each iteration. Also use a variable `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 % divisor = 0` and `1 % divisor = 1`, so 0 receives and adds, while 1 sends. Also in the first iteration `0 + core_difference = 1` and `1 – core_difference = 0`, so 0 and 1 are paired in the first iteration.

Homework 1, cont.

- Problem 3: What are your goals after this year and how do you anticipate this class is going to help you with that? Some possible answers, but please feel free to add to them. Also, please write at least one sentence of explanation.
  - A job in the computing industry
  - A job in some other industry that uses computing
  - As preparation for graduate studies
  - To satisfy intellectual curiosity about the future of the computing field
  - Other

Today's Lecture

- Aspects of parallel algorithms (and a hint at complexity!)
- Derive parallel algorithms
- Discussion
- Sources for this lecture:
  - Slides accompanying textbook

Reasoning about a Parallel Algorithm

- Ignore architectural details for now (next time)
- Assume we are starting with a sequential algorithm and trying to modify it to execute in parallel
  - Not always the best strategy, as sometimes the best parallel algorithms are NOTHING like their sequential counterparts
  - But useful since you are accustomed to sequential algorithms

Reasoning about a parallel algorithm, cont.

- Computation Decomposition
  - How to divide the sequential computation among parallel threads/processors/computations?
- Aside: Also, Data Partitioning (ignore today)
- Preserving Dependences
  - Keeping the data values consistent with respect to the sequential execution.
- Overhead
  - We'll talk about some different kinds of overhead

Race Condition or Data Dependence

- A ***race condition*** exists when the result of an execution depends on the ***timing*** of two or more events.
- A ***data dependence*** is an ordering on a pair of memory operations that must be preserved to maintain correctness. (More on data dependences in a subsequent lecture.)
- ***Synchronization*** is used to sequence control among threads or to sequence accesses to data in parallel code.

Simple Example (p. 4 of text)

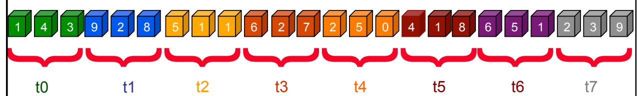
- Compute n values and add them together.
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

- Parallel formulation?

Version 1: Computation Partitioning

- Suppose each core computes a partial sum on n/t consecutive elements (t is the number of threads or processors)
- Example: n = 24 and t = 8, threads are numbered from 0 to 3




```
int block_length_per_thread = n/t;
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    x = Compute_next_value(…);
    sum += x;
}
```


What Happened?

- Dependence on sum across iterations/ threads
  - But reordering ok since operations on sum are associative
- Load/increment/store must be done *atomically* to preserve sequential meaning
- Definitions:
  - Atomicity: a set of operations is atomic if either they all execute or none executes. Thus, there is no way to see the results of a partial execution.
  - Mutual exclusion: at most one thread can execute the code at any time

Version 2: Add Locks

- Insert mutual exclusion (mutex) so that only one thread at a time is loading/incrementing/ storing count atomically

```
int block_length_per_thread = n/t;
mutex m;
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    my_x = Compute_next_value(…);
    mutex_lock(m);
    sum += my_x;
    mutex_unlock(m);
}
```

Correct now. Done?

Version 3: Increase Granularity

- Version 3:
  - Lock only to update final sum from private copy

```
int block_length_per_thread = n/t;
mutex m;
int my_sum;
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    my_x = Compute_next_value(…);
    my_sum += my_x;
}
mutex_lock(m);
sum += my_sum;
mutex_unlock(m);
```


Version 4: Eliminate lock

- Version 4 (bottom of page 4 in textbook):
  - "Master" processor accumulates result

```
int block_length_per_thread = n/t;
mutex m;
shared my_sum[t];
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    my_x = Compute_next_value(…);
    my_sum[id] += my_x;
}
if (id == 0) { // master thread
   sum = my_sum[0];
   for (i=1; i<t; i++) sum += my_sum[i];
}
```

Correct? Why not?

More Synchronization: Barriers

- Incorrect if master thread begins accumulating final result before other threads are done
- How can we force the master to wait until the threads are ready?
- Definition:
  - A ***barrier*** is used to block threads from proceeding beyond a program point until all of the participating threads has reached the barrier.
  - Implementation of barriers?

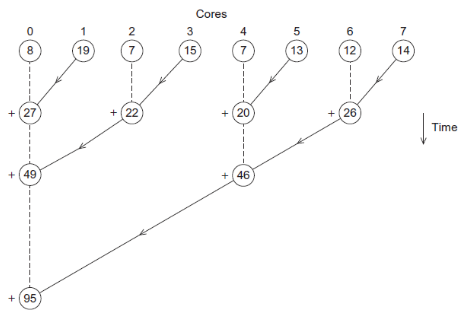
Version 5: Eliminate lock, but add barrier

- Version 5 (bottom of page 4 in textbook):
  - "Master" processor accumulates result

```
int block_length_per_thread = n/t;
mutex m;
shared my_sum[t];
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    my_x = Compute_next_value(…);
    my_sum[t] += x;
}
Synchronize_cores(); // barrier for all participating threads
if (id == 0) { // master thread
   sum = my_sum[0];
   for (i=1; i<t; i++) sum += my_sum[t];
}
```

Now it's correct!

Version 6 (homework): Multiple cores forming a global sum

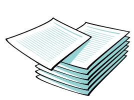


How do we write parallel programs?

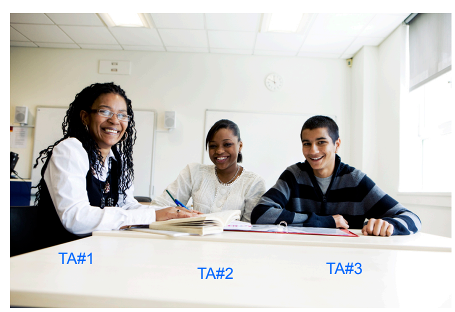
- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.

Professor P

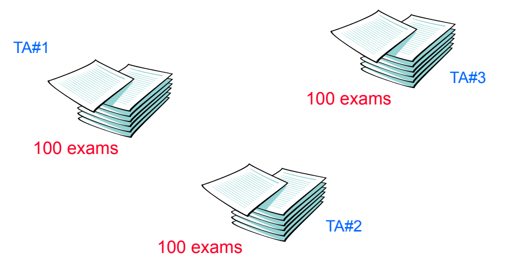
15 questions
300 exams



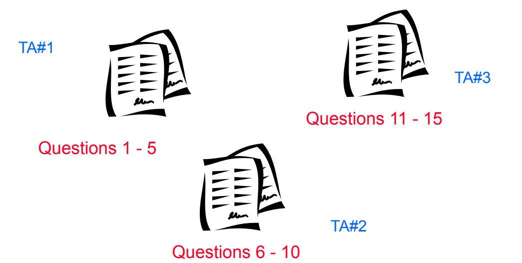
Professor P's grading assistants



Division of work – data parallelism



Division of work – task parallelism



Summary of Lessons from Sum Computation

Data and Task Parallelism: Discussion Problem 1

- Problem 1: Recall the example of building a house from the first lecture.
  - (a) Identify a portion of home building that can employ *data parallelism*, where "data" in this context is any object used as an input to the home-building process, as opposed to tools that can be thought of as processing resources.
  - (b) Identify *task parallelism* in home building by defining a set of tasks. Work out a schedule that shows when the various tasks can be performed.
  - (c) Describe how task and data parallelism can be combined in building a home. What computations can be reassigned to different workers to balance the load?

Data and Task Parallelism: Discussion Problem 2

- 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.