CS4961 Parallel Programming

Lecture 2: Introduction to Parallel Algorithms

Mary Hall
August 25, 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 `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: 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.

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)

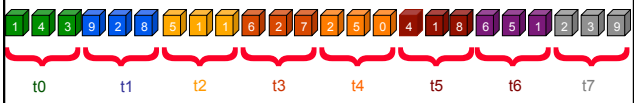
- Compute n values and add them together.
- Serial solution:

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

- Parallel formulation?

Version 1: Computation Partitioning

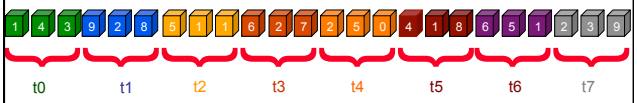
- Suppose each core computes a partial sum on n/t consecutive elements (t is the number of threads or processors)
- Example: n = 24 and t = 8, threads are numbered from 0 to 3

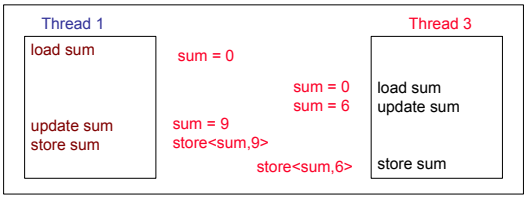



```
int block_length_per_thread = n/t;
int start = id * block_length_per_thread;
for (i=start; i<start+block_length_per_thread; i++) {
    x = Compute_next_value(…);
    sum += x;
}
```


Data Race on Sum Variable

- Two threads may interfere on memory writes





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

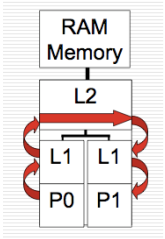
- Insert mutual exclusion (mutex) so that only one thread at a time is loading/incrementing/ storing count atomically

```
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?

Lock Contention and Poor Granularity

- To acquire lock, must go through at least a few levels of cache (locality)
  - Local copy in register not going to be correct
- Not a lot of parallel work outside of acquiring/releasing lock



Slide source: Larry Snyder, "http://www.cs.washington.edu/education/courses/524/08wi/"

Increase Parallelism Granularity

- Private copy of sum for each core
- Accumulate later – how?

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

Accumulating result

- 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);
```

Accumulating result

- 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?

Accumulating result

- 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!

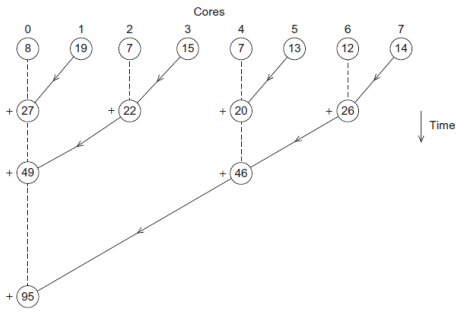
Discussion: Overheads

- What were the overheads we saw with this example?
  - Extra code to determine portion of computation
  - Locking overhead: inherent cost plus contention
  - Load imbalance

Problem #1 on homework (see page 5)

- A strategy to improve load imbalance
- Suppose number of threads is large and master is spending a lot of time in calculating final value
- Have threads combine partial results with their "neighbors"
  - Tree-structured computation sums partial results

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

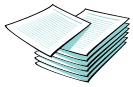


How do we write parallel programs?

- 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.

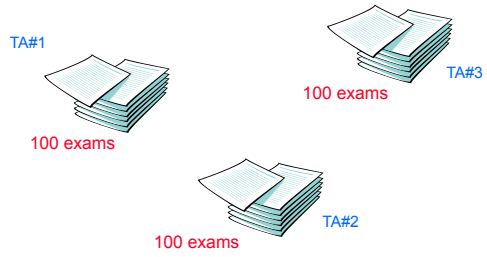
Professor P

15 questions
300 exams

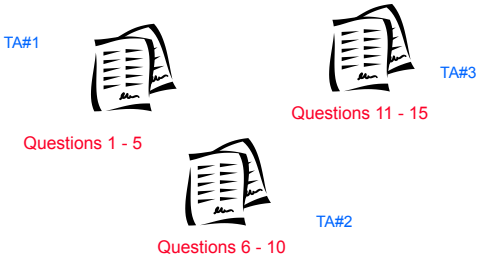


Professor P's grading assistants

Division of work – data parallelism



Division of work – task parallelism



Generalizing from this example

- Interestingly, this code represents a common pattern in parallel algorithms
- A reduction computation
  - From a large amount of input data, compute a smaller result that represents a reduction in the dimensionality of the input
  - In this case, a reduction from an array input to a scalar result (the sum)
- Reduction computations exhibit dependences that must be preserved
  - Looks like *"result = result op …"*
  - Operation *op* must be associative so that it is safe to reorder them
- Aside: Floating point arithmetic is not truly associative, but usually ok to reorder

Examples of Reduction Computations

- Count all the occurrences of a word in news articles (how many times does Libya appear in articles from Tuesday?)
- What is the minimum id # of all students in this class?
- What is the highest price paid for a gallon of gas in Salt Lake City over the past 10 years?

Summary of Lecture

- How to Derive Parallel Versions of Sequential Algorithms
  - Computation Partitioning
  - Preserving Dependences using Synchronization
  - Reduction Computations
  - Overheads

Next Time

- A Discussion of parallel computing platforms
- Go over first written homework assignment