Stanford CS149: Parallel Computing
Written Assignment 2

Sending Messages

Problem 1. (30 points):

A. (10 pts) Your friend suspects that their program is suffering from high communication overhead, so to overlap the sending of multiple messages, they try to change their code to use asynchronous, non-blocking sends instead of synchronous, blocking sends. The result is this code (assume it is run by thread 1 in two-thread program).

```
float mydata[ARRAY_SIZE];
int dst_thread = 2;

update_data_1(mydata); // updates contents of mydata
async_send(dst_thread, mydata, sizeof(float) * ARRAY_SIZE);

update_data_2(mydata); // updates contents of mydata
async_send(dst_thread, mydata, sizeof(float) * ARRAY_SIZE);
```

Your friend runs to you to say "my program no longer gives the correct results." What is their bug?

B. (20 pts) In class we talked about the `barrier()` synchronization primitive. No thread proceeds past a barrier until all threads in the system have reached the barrier. (In other words, the call to `barrier()` will not return to the caller until its known that all threads have called `barrier()`. Consider implementing a barrier in the context of a message passing program that is only allowed to communicate via **blocking sends and receives**. Using only the helper functions defined below, implement a barrier. Your solution should make no assumptions about the number of threads in the system. **Keep in mind that all threads in a message passing program execute in their own address space—there are no shared variables.**

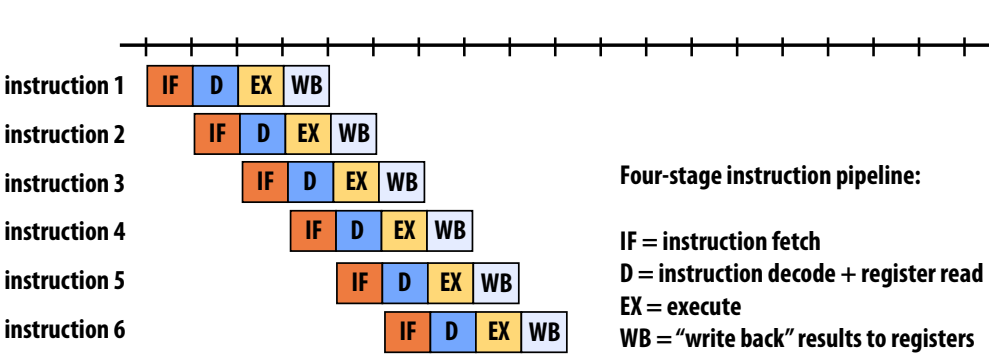
```
// send msg with id msgId and contents msgValue to thread dstThread
void blockingSend(int dstThread, int msgId, int value);

// recv message from srcThread. Upon return, msgId and msgValue are populated
void blockingRecv(int srcThread, int* msgId, int* msgValue);

// returns the id of the calling thread
int getThreadId();

// returns the number of threads in the program
int getNumThreads();
```

A Cardinal Processor Pipeline

Problem 2. (40 points):

The fast-growing startup Cardinal Processors, Inc. builds a single core, single threaded processor that executes instructions using a simple four-stage pipeline. As shown in the figure below, each unit performs its work for an instruction **in one clock**. To keep things simple, assume this is the case for all instructions in the program, including loads and stores (memory is infinitely fast).

The figure shows the execution of a program with six **independent instructions** on this processor. *However, if instruction B depends on the results of instruction A, instruction B will not begin the IF phase of execution until the clock after WB completes for A.*



A. (4 pts) Assuming all instructions in a program are **independent** (yes, a bit unrealistic) what is the instruction throughput of the processor?

B. (4 pts) Assuming all instructions in a program are **dependent** on the previous instruction, what is the instruction throughput of the processor?

C. (4 pts) What is the latency of completing an instruction?

D. (4 pts) Imagine the EX stage is modified to improve its throughput to two instructions per clock. What is the new overall maximum instruction throughput of the processor?

E. (8 pts) Consider the following C program:

```
float A[100000];
float B[100000];
// assume A is initialized here

for (int i=0; i<100000; i++) {
    float x1 = A[i];
    float x2 = 2*x1;
    float x3 = 3 + x2;
    B[i] = x3;
}
```

Assuming that we consider only the four instructions in the loop body (for simplicity, disregard instructions for managing the loop), what is the average instruction throughput of this program? (Hint: You should probably consider instruction dependencies, and at least two loop iterations worth of work).

F. (10 pts) Modify the program to achieve peak instruction throughput on the processor. Please give your answer in C-pseudocode.

G. (6 pts) Now assume the program is reverted to the original code, and the for loop is parallelized using OpenMP.

```
// assume iterations of this FOR LOOP are parallelized across multiple
// worker threads in a thread pool.
#pragma omp parallel for
for (int i=0; i<100000; i++) {
  float x1 = A[i];
  float x2 = 2*x1;
  float x3 = 3 + x2;
  B[i] = x3;
}
```

Given this program, imagine you wanted to add multi-threading to the **single-core processor** to obtain **peak instruction throughput** (100% utilization of execution resources). What is the smallest number of threads your processor could support and still achieve this goal? You may not change the program.

Be a Parallel Processor Architect

Problem 3. (30 points):

You are hired to start the parallel processor design team at Lagunita Processors, Inc. Your boss tells you that you are responsible for designing the company's first shared address space multi-core processor, which will be constructed by cramming multiple copies of the company's best selling uniprocessor core on a single chip. Your boss expects the project to yield at least a $5\times$ speedup on the performance of the program given below. You are not allowed to change the program, and assume that:

- Each Lagunita core can complete one floating point operation per clock
- Cores are clocked at 1 GHz, and each have a 1 MB cache using LRU replacement.
- All Lagunita processors (both single and multi-core) are attached to a 100 GB/s memory bus
- Memory latency is perfectly hidden (Lagunita processors have excellent pre-fetchers)

```
float A[N]; // let N = 100 million elements
float total = 0;

// ASSUME TIMER STARTS HERE ////////////////////////////////

for (int i=0; i<N; i++)
   total += A[i];

for (int i=0; i<9; i++) {

   // made up syntax for brevity: 'parallel_for'
   // Assume iterations of this loop are perfectly partitioned
   // using blocked assignment among X pthreads each running on
   // one of the processor's X cores.
   parallel_for(int j=0; j<N; j++) {
      A[j] = A[j] / total;
   }
}

// ASSUME TIMER STOPS HERE /////////////////////////////////
```

A. (10 pts) How do you respond to your boss' request? Do you believe you can meet the performance goal? If yes, how many cores should be included in the new multi-core processor? If no, explain why.

B. (10 pts) You tell your boss that if you were allowed to make a few changes to the code, you could deliver a much better speedup with your parallel processor design. How would you change the code to improve its performance by improving *speedup*? (A simple description of the new code is fine). If your answer was NO in part one, how many processors are required to achieve 5× speedup now? If your answer was YES, approximately what speedup do you expect from your previously proposed machine on the new code? (*Note: we are NOT looking for answers that optimize the program by rolling multiple divisions into one.*)

C. (10 pts) Assume that the following year, Lagunita Processors, Inc. decides to produce a 32-core version of your parallel CPU design. In addition to adding cores, your boss gives you the opportunity to further improve the processor through one of the following three options.

- You may double each processor's cache to 2 MB.
- You may increase memory bandwidth by 50%
- You may add a 4-wide SIMD unit to the core so that each core can perform 4 floating point operations per clock.

If each of these options has the same cost, given the code you produced in part B (and what you learned from assignment 1), which option do you recommend to your boss? Why?