CpSc 448B **Midterm** October 20, 2011

100 points

Time for the exam: 80 minutes.

Open book: anything printed on paper may be brought to the exam and used during the exam. This includes the textbook, other books, printed copies of the lecture slides, lecture notes, homework and solutions, and any other material that a student chooses to bring.

Calculators are allowed: no restriction on programmability or graphing. There are a few simple calculations needed in the exam, a calculator will be handy, but the fancy features won't make a difference.

No communication devices: That's right. You may not use your cell phone for voice, text, web-surfing, or any other purpose. Likewise, the use of computers, i-pods, etc. is not permitted during the exam.

Good luck! ☺

CpSc 448B **Midterm** October 20, 2011

100 points

Answer **all** of questions 1 through 3. Answer any **four** of questions 4 through 8.

1. **Erlang (14 points).**

   (a) **Read some code** (**5 points**)
   Consider the module `q1a` shown in Figure 1a. Write down an Erlang expression, `E`, your score for this problem will be `q1a:score(E).`

   (b) **Write some code** (**9 points**)
   Consider the module `q1b` shown in Figure 1b. For this problem, you need to write an implementation of function `q1b` such that each call to `q1b:read(P)` returns the last value "written" by a call to `q1b:write(P, Value)`, where P is an Erlang pid returned by a call to `q1b:create()`. For example, here's a transcript of an Erlang session using my solution:

```
1> c(q1b).
{ok,q1b}
2> P = q1b:create().
<0.38.0>

2> q1b:write(P, 2).
{write,2}
3> q1b:read(P).
2
4> q1b:write(P, 17).
{write,17}
5> q1b:write(P, 42).
{write,42}
6> q1b:read(P).
42
7> P2 = q1b:create().
<0.47.0>
8> q1b:read(P2).
empty
```

```
-module q1a.
-export [score/1].

score([H | T]) ->
   MyPid = self(),
   Cpid = spawn(fun() -> MyPid !  helper(0) end),
   send(Cpid, [H | T]),
   receive
      N when N =< 5 -> N;
      _ -> 0
   end;
score(_) -> 0.

helper(V) ->
   receive
      0 -> helper(2*V);
      1 -> helper(2*V + 1);
      '$' -> V
   end.

send(_Pid, []) -> ok;
send(Pid, [H | T]) ->
   Pid !  H,
   send(Pid, T).
```

(a) Module for question 1a

```
-module q1b.
-export [create/0, read/1, write/2].

create() -> spawn(fun() -> q1b(empty) end).

read(Pid) ->
   Pid !  {read, self()},
   receive
      {Pid, read, Value} -> Value
   end.

write(Pid, Value) -> Pid !  {write, Value}.
```

(b) Module for question 1b

Figure 1: Modules for question 1

C:

```
for(int k = 0; k < n; k++)
   sum += a[i,k] * b[k,j];
```

assembly:

```
loop:   ld    $fa, 0($aptr)           // fa ← a[i,k]
        add   $aptr, $aptr, 8         // move aptr to next column
        ld    $fb, $bptr              // fb ← b[k,j]
        add   $bptr, $bptr, $bstride  // move bptr to next row
        fmult $fp, $fa, $fb           // fp ← a[i,k]*b[k,j]
        fadd  $fsum, $fsum, $fp       // sum ← sum + a[i,k]*b[k,j]
        bne   $aptr, $atop, loop      // loop test:  k < n
```

Figure 2: C and assembly for the inner-loop of matrix multiplication

2. **Dependences** (**21 points**)

   (a) (**6 points**)
   What are the three types of "dependences" defined in Lin & Snyder? Describe each with one or two short sentences.

   (b) (**6 points**)
   Figure 2 shows the C and assembly code for the inner-loop of matrix multiplication. Identify one example of each kind of dependence.

   (c) (**3 points**)
   Which of the dependences that you identified for question 2b are true dependences and which are false dependences?

   (d) (**6 points**)
   Briefly explain how a superscalar processor handles each dependence that you identified for question 2b.

3. **Multicore, Transactions and the Future of Distributed Computing** (**5 points**)
   What was the main problem addressed by the talk?

   (a) Persistent memory (e.g. DRAM with battery back-up) will replace disk-based transaction systems in future distributed applications such as web-servers.

   (b) Declaring blocks of operations to be atomic is a more natural way to handle synchronization than either coarse or fine grained locks and should lead to simpler programming and more efficient multicore and distributed systems.

   (c) Programming of multicore and distributed systems should be built on top of a transactional, relational data base layer to provide a higher-level programming model and robustness against system failures.

   (d) Current cache-memory hierarchies work poorly with database servers and web-servers. Memory systems for multicore processors should be designed to directly support database and distributed, web-style transactions.

   (e) Multicore computing is vastly increasing the compute power of individual computer systems. To exploit the extra processing power, future distributed systems will need a transaction-based paradigm to replace the client-server framework that is prevalent today.

Answer **any four** of questions 4 through 8. If you respond to all five, please indicate which four to mark. Otherwise, I will make an **arbitrary** choice.

4. **Sequential Consistency** (**15 points**)
   What is sequential consistency?
   Your answer should be at most 100 words long.

5. **Snooping Caches** (**15 points**)
   What is a snooping cache?
   Your answer should be at most 100 words long.

Questions 6 through 8 address the speed-ups that can be achieved when multiplying two $N \times N$ matrices using P processors.

6. **Amdahl** (**15 points**)
   Amdahl suggested modeling a computation as having some fraction, s that must be performed sequentially, and the rest $(1 - s)$ that can be run P times faster when run on P processors. Assume that for matrix multiply, $s = 0.01$. According to Amdahl's law, how many times faster is the parallel version than the sequential one for $P = 10$, $P = 100$, and $P = 1000$?

7. **Algorithm 1** (**15 points**)
   Most of the overhead for matrix multiplication is from the communication time required to send blocks of matrix elements between processors. In class, I sketched an algorithm where each processor sends and receives $P - 1$ messages of size N^2/P.

   - Assume that a single processor can compute the product of two $N \times N$ matrix in N^3 clock cycles.
   - Assume that each processor for the parallel version requires time N^3/P for computation plus the time required to send and receive $P - 1$ messages of size N^2/P.
   - Assume that the total time consumed at the sender and receiver to convey a message of M matrix elements is $1000 + 10M$ processor cycles.

   According to this model, how many times faster is the parallel version than the sequential one for $P = 10$, $P = 100$, and $P = 1000$? Use $N = 1000$ for all three values of P.

8. **Algorithm 2** (**15 points**)
   I also sketched an implementation that only requires sending $\sqrt{P}$ messages of size N^2/P per processor. Using the same assumptions for compute time and communication time as for question 7, how many times faster is this improved parallel version than the sequential one for $P = 10$, $P = 100$, and $P = 1000$? As before, use $N = 1000$ for all three values of P.