UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR11023 PARALLEL PROGRAMMING LANGUAGES AND SYSTEMS (LEVEL 11)

Friday 9 th May 2014

14:30 to 16:30

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 4 Courses

Convener: I. Stark
External Examiners: A. Cohn, T. Field

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Discuss what it means to say that "Java's `Object` class was designed to support parallel programming in a monitor style, with signal-and-continue semantics." Your answer should discuss relevant keywords and methods, and should include and explain a state diagram for the given semantics. [*6 marks*]

   (b) Consider the following code, which implements a producers-consumers buffer.

```
public class Buffer extends Vector {
  public synchronized void putLast (Object obj) {
     addElement(obj);     // add to the end of the buffer
     notify();
  }
  public synchronized Object getFirst () {
     while (isEmpty())
       try {wait();}
       catch (InterruptedException e) {return null;}
     Object obj = elementAt(0);
     removeElementAt(0);  //remove from the front of the buffer
     return obj;
  }
}
```

   i. Explain why there is a loop around the call to the **wait** method. [*3 marks*]

   ii. What constraints does this implementation impose upon concurrency between and amongst producers and consumers? Is this reasonable? [*3 marks*]

   (c) Write a Java monitor implementation of a re-usable counter barrier. You will not be penalised for minor syntactic errors. If you can't remember the Java name of some standard monitor method then make up your own name and explain its effect. [*6 marks*]

   (d) Consider the following scenario: a set of Pthreads are working independently, except that occasionally a thread needs to find another thread with which it can swap an integer value. The thread pairings are arbitrary: a thread swaps with whichever other thread becomes available first. Write a Pthreads monitor which allows pairs of threads to meet and exchange integers in this way. The monitor should have a single operation called **swap**, which is called by a thread when it wishes to participate in such a swap. [*7 marks*]

2. (a) The algorithm template section of Threading Building Blocks (TBB) is an example of a task-based parallel programming library. Explain what is meant by the phrase "task-based" in the previous sentence. Using TBB's `parallel_for` template as an example, explain how opportunities for parallelism are identified by the TBB programmer, and exploited by the TBB run-time. Your answer should refer to the TBB concepts of `Range` and `Body`, and anything else you feel is relevant. [*6 marks*]

(b) Consider the following code, which implements the naive Fibonacci function.

```
class FibRange {
public:
  int n_ ;
  FibRange(int n) : n_(n) { }

  FibRange(FibRange& other, split)
  : n_(other.n_ - 2) { other.n_ = other.n_ - 1;}

  bool is_divisible() const { return (n_ > 10); }
  bool is_empty() const { return n_ < 0; };
};

class Fib {
public:
  int fsum_ ;
  Fib() : fsum_(0) { }
  Fib(Fib& other, split) : fsum_(0) { }

  void operator() (FibRange& range) { fsum_ += fib(range.n_ ); }

  int fib(int n) {if (n < 2) return n; else return fib(n-1)+fib(n-2);}

  void join(Fib& rhs) { fsum_ += rhs.fsum_; };
};

int main( int argc, char* argv[] ) {
 task_scheduler_init init(NPROCS);
 Fib f;

 parallel_reduce(FibRange(FIBSEED), f, simple_partitioner());
 cout << "Fib " << FIBSEED < " is " << f.fsum_ << "\n";
 return 0;
}
```

QUESTION CONTINUES ON NEXT PAGE

Drawing attention to specific parts of the code, explain how the TBB `parallel_reduce` template has been used to create and control the massive potential parallelism inherent in naive Fibonacci computation. (Note: this question is about TBB mechanisms, so the fact that Fibonacci can be computed much more efficiently by other means is not relevant). [*6 marks*]

(c) A simple Pthreads version of the naive Fibonacci computation would create a thread for each recursive call above some threshold argument value.

i. Sketch Pthreads pseudocode for such an implementation (you don't have to worry about precise Pthreads syntax, as long as the concepts are clear). [*4 marks*]

ii. Contrast the behaviour of the Pthreads and TBB versions in terms of the creation of threads and scheduling of work. [*2 marks*]

(d) Consider the following pseudocode, taken from the discussion of the Adaptive Quadrature (AQ) problem as discussed in the course.

```
double quad (double left, right, fleft, fright, lrarea) {
  double mid = (left + right) / 2;
  double fmid = f(mid);
  double larea = (fleft+fmid) * (mid-left) / 2;
  double rarea = (fmid+fright) * (right-mid) / 2;
  if (abs((larea+rarea) - lrarea) > EPSILON) {
    larea = quad(left, mid, fleft, fmid, larea);
    rarea = quad(mid, right, fmid, fright, rarea);
  }
  return (larea + rarea);
}

area = quad (a, b, f(a), f(b), (f(a)+f(b))*(b-a)/2);
```

Explain how you would use this algorithm to develop a TBB `parallel_reduce` solution to the AQ problem. You should explain clearly how you would use TBB mechanisms to address the various issues. You may find it helpful to write small code fragments for illustration, but a complete syntactically correct program is not required. [*7 marks*]

3. (a) Explain the purpose and behaviour of the MPI operation `MPI_Comm_split`, concentrating on the concepts involved rather than syntax. [*3 marks*]

(b) Consider the following MPI program, executed with three processes in `MPI_COMM_WORLD`.

```
int main(int argc, char *argv[]) {
  int my_rank, p, x, y, z;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &p);

  switch(my_rank) {
    case 0: x=0; y=1; z=2;
      MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
      MPI_Send(&y, 1, MPI_INT, 2, 43, MPI_COMM_WORLD);          // line Z
      MPI_Bcast(&z, 1, MPI_INT, 1, MPI_COMM_WORLD);
    break;
    case 1: x=3; y=8; z=5;
      MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
      MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);
    break;
    case 2: x=6; y=7; z=8;
      MPI_Bcast(&z, 1, MPI_INT, 0, MPI_COMM_WORLD);
      MPI_Recv(&x, 1, MPI_INT, 0, 43, MPI_COMM_WORLD, &status);  // line X
      MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);              // line Y
    break;
  }
  MPI_Finalize();
}
```

i. Explain (for example, with the help of a simple table), the sequence of changes which occur to variables `x`, `y` and `z` as the program executes. [*3 marks*]

ii. Consider the following variations, and explain what difference each would make to the behaviour of the program. Justify your answer in each case.

A. What difference would it make if the value `43` on line X was changed to `42`?

B. What difference would it make if line X and line Y were switched (so that line Y occurs before line X)?

QUESTION CONTINUES ON NEXT PAGE

C. What difference would it make if line X and line Y were switched (so that line Y occurs before line X) and line Z was changed to use `MPI_Ssend`?

D. What difference would it make if the program, unchanged, was executed with four processes? [*7 marks*]

(c) Discuss the extent to which the Linda coordination model could be used as a substitute for a message passing library such as MPI. Your answer should concentrate on what you believe to be the key conceptual capabilities of MPI, rather than details of syntax or underlying implementation. [*5 marks*]

(d) i. Explain what it means for a shared variable programming model to exhibit sequential consistency. You should give a simple example which illustrates the difference between sequential consistency and weaker models, and explain why such weaker models may still be attractive. [*3 marks*]

ii. Discuss the extent to which the Linda coordination model could be used to implement shared variable style programs. [*4 marks*]