Intro to High-Performance Computing May 4, 2016
University of California, Irvine EECS 221
Prof. Aparna Chandramowlishwaran Midterm

Algorithms, Performance Analysis, Programming Models, and Parallelism

Name: _____

UCInetID: _____

Instructions

- DO NOT open this quiz booklet until you are instructed to do so.
- This quiz booklet contains 10 pages, including this one. You have 50 minutes to earn 50 points.
- This quiz is closed book.
- When the quiz begins, please write your name and UCInetID on this coversheet.
- Some of the questions are true/false, and some are multiple choice. You are not required to explain these answers unless you are explicitly asked to do so, although including some justification for your answers maximizes your potential for partial credit. Since
- Good luck!

	Question	**Parts**	**Points**	**Score**
1	True or False	5	12	
2	Multiple Choice	4	8	
3	Analysis of Parallelism	4	20	
4	Distributed Memory	3	10	
	Total		50	

1 True or False (5 parts, 12 points)

Indicate whether each statement below is true or false by circling the correct answer.

1.1 [2 points]

A 4-D hypercube has 8 nodes.

True False

1.2 [2 points]

GPUs use extensive branch prediction to improve performance.

True False

1.3 [2 points]

CUDA supports global synchronization.

True False

Indicate whether each statement below is true or false by circling the correct answer, and justify your answer in at most one or two sentences. **Your justification is worth more than your true-false designation.**

1.4 [3 points]

Consider the `scale()` function:

```
void scale(double *A, double *B, double c, size_t n)
{
  for (size_t i = 0; i < n; ++i)
    A[i] += c * B[i];
}
```

One can parallelize the `scale()` function by simply adding a `#pragma omp parallel for` above the loop.

True False

1.5 [3 points]

Suppose that the running time of a recursive program satisfies the recurrence $T(n) = 2T(n/2) + \Theta(n^2)$ (with a base condition of $T(n) = \Theta(1)$ for sufficiently small n). Optimizing the leaves of the recursion should result in a significant speedup.

True **False**

2 Multiple Choice (4 parts, 8 points)

2.1 [2 points]

Consider the snippet of code below that computes the matrix-vector product of a matrix A of size $n \times n$ and a vector x of size n.

```
// Computes: y <- y + A . x
for (i = 0; i < n; ++i)  // Loop 1
  for (j = 0; j < n; ++j)  // Loop 2
    y[i] += A[i,j] * x[j];
```

Which for-loops may be **safely** converted into parallel-for loops.

A Only loop 1
B Only loop 2
C Both loops 1 and 2
D Neither loop 1 nor 2

2.2 [2 points]

How much memory does the SUMMA algorithm need compared to the 1-D matrix multiply algorithm discussed in class?

A More than 1D
B Less than 1D
C Same as 1D

2.3 [2 points]

What is the **diameter** defined as the maximum number of message hops between processors in a $N \times N$ processor mesh, with $N = \sqrt{P}$?

A $\sqrt{P}$
B $2(\sqrt{P} - 1)$
C $2\sqrt{P}$
D P
E P^2

2.4 [2 points]

Ben is worried about races and uses a critical section to synchronize the code snippet below.

```
#pragma omp parallel for
for (i = 0; i < n; ++i) {
  #pragma omp critical
  sum += a[i];
}
```

The code does not achieve any speedup. (Circle all that apply)

A The code still has a race and the race slows down the program
B The critical section only allows one iteration to execute the body of the loop at any time
C Ben could have used OpenMP reduction to achieve the same result and could have been faster
D All of the above

3 Analysis of Parallelism (4 parts, 20 points)

3.1 [5 points]

Five students have implemented recursive Fibonacci programs, where the base case of each program returns 1 if the program input is $n = 0$ or $n = 1$. For $n > 1$, the various students calculate Fibonacci using the code snippets for the recursive cases shown below:

a:

```
x = fib(n-1);
y = fib(n-2);
```

b:

```
x = spawn fib(n-1);
y = spawn fib(n-2);
sync;
```

c:

```
x = fib(n-1);
y = spawn fib(n-2);
sync;
```

d:

```
y = spawn fib(n-2);
x = fib(n-1);
sync;
```

e:

```
x = spawn fib(n-1);
y = fib(n-2);
sync;
```

Assume that the overhead of spawning a function is about 10 times the cost of an ordinary function call. Rank these codes in order of the performance you would expect on a 32-core machine (e.g., fastest > second fastest > $\cdots$ > slowest):

_____ > _____ > _____ > _____ > _____

For each of the multiple-choice questions below, circle the letter corresponding to the correct answer. **You need to explain your answers.**

3.2 [5 points]

Consider the following multithreaded function, which implements a dot product of two vectors, each of size n, in parallel:

```
double dot_product(double *A, double *B, int n)
{
  if (n == 1)
    return *A * *B;
  else
  {
    int mid = n / 2;
    double p1 = spawn dot_product(A, B, mid);
    double p2 = dot_product(A + mid, B + mid, n - mid);
    sync;
    return p1 + p2;
  }
}
```

What is the parallelism of the `dot_product` function?

A $\Theta(\lg n)$
B $\Theta(n/\lg n)$
C $\Theta(n)$
D $\Theta(n^2/\lg n)$
E None of the above

3.3 [5 points]

A matrix L is ***lower triangular*** if $L(i,j) = 0$ for $i < j$. A lower-triangular matrix can be stored compactly in a one-dimensional array by storing row i, which has length $i+1$ starting in location $i(i+1)/2$.

Consider the following function that computes the matrix-vector product of a lower triangular matrix L of size $n \times n$ and a vector X of size n:

```
void lt_matrix_vector_product(double *L, double *X, double *B, int n)
{
  parallel for (int i = 0; i < n; i++)
  {
    int offset = (i * (i + 1)) / 2;
    B [i] = dot_product(L + offset, X, i + 1);
  }
}
```

For the purposes of analysis, assume that the grain size for the `parallel for` is 1. What is the parallelism of `lt_matrix_vector_product` function?

A $\Theta(n/\lg^2 n)$
B $\Theta(n)$
C $\Theta(n^2/\lg^2 n)$
D $\Theta(n^2/\lg n)$
E None of the above

3.4 [5 points]

Now consider a different way of computing the matrix-vector product for a lower-triangular matrix:

```
void new_lt_matrix_vector_product(double *L, double *X, double *B, int n)
{
  for (int i = 0; i < n; i++)
  {
    int offset = (i * (i + 1)) / 2;
    B [i] = spawn dot_product(L + offset, X, i + 1);
  }
  sync;
}
```

What is the parallelism of `new_lt_matrix_vector_product` function?

A $\Theta(n/\lg n)$
B $\Theta(n)$
C $\Theta(n^2/\lg^2 n)$
D $\Theta(n^2/\lg n)$
E None of the above

4 Distributed Memory (3 parts, 10 points)

4.1 [2 points]

Draw a picture to show how a 128 x 128 matrix A would be distributed among 16 processors using a 2-D block-cyclic distribution with a block size of 32 x 32 .

4.2 [4 points]

Sketch and describe an algorithm for multiplying the matrix with a 128 x 1 vector x using collective communication algorithms. You may assume that the vector x and the result vector b are stored on the diagonal processors.

4.3 [4 points]

Write expressions for the computation and communication costs of your algorithm.