CSE 443/543: High Performance Computing & Parallel Programming
Miami University

U

Exam 1

Spring 2015

There are 13 questions for a total of 110 points.
Maximum Possible Score: 100 points

NAME (PRINT IN ALL CAPS): _____

General Instructions

- **Wait until you are instructed to start.**
- First, ensure you have all the 20 pages of this exam booklet before starting.
- This exam is closed notes and closed books. No discussions are permitted.
- Even if your final answers are incorrect, you will get partial credit if intermediate steps are clearly shown to highlight thought process. This applies to program tracing questions as well.
- All work must be written on the exam pages in order to be graded. Any scrap paper used, must be the extra sheets provided during the exam period.
- For programming questions: Be accurate with your C++ syntax. This includes appropriate use of braces, semicolons, and the proper use of upper/lowercase letters. Points will be deducted for significant syntax errors in programs and C++ statements that you write in this exam. Exception handling is not required.
- You are permitted to use a simple calculator. You calculator cannot be: a graphing calculator, cannot have a QWERTY keyboard, or be integrated with other electronic devices such as: iPods, PDAs, and cellular phones.
- You have about two hours to complete the exam.

GOOD LUCK!

Multiple Choice Questions [30 Points]

1. Clearly circle only the best response for each question below. If your choice is not clearly indicated or if multiple choices seem to be selected then you will not earn any points for that question. If you change your answer, ensure that your final answer is clearly highlighted. **Each question is worth 2 points**.

i. The primary objective of high performance computing is to:

A. Reduce time complexity of programs

B. Redesign programs to make them more user friendly

C. Improve efficiency of programs

D. Facilitate effective interaction between computers and scientists

ii. The expansion of the acronym RAM is:

A. Random Access Memory

B. Real Automatic Memory

C. Repeatable Access Memory

D. RAM is not acronym but an abbreviation for "Real/Actual Memory"

iii. The net theoretical FLOPS of a super computer with 400 CPUs, each CPU having eight 2-GFLOPS cores (@ 1 CPI) would be:

A. 3200 GFLOPS

B. 640 GFLOPS

C. 6400 GFLOPS

D. 320 GFLOPS

iv. A 1 GHz CPU has a clock frequency of:

A. 1 millisecond

B. 1 microsecond

C. 1 nanosecond

D. 1 picosecond

v. The minimum number of ALUs in a SSE capable CPU-core is:

A. 0

B. 1

C. 2

D. 3

vi. A programmer coded 4 instructions in the order: I1, I2, I3, and I4. However, a microprocessor executed the instructions always in the reverse order: I4, I3, I2, and I1. This is because:

A. The microprocessor has SIMD instructions.

B. The microprocessor is superscalar.

C. The microprocessor has SSE instructions.

D. All of the above

E. None of the above

vii. The fragment of C++ shown in the adjacent figure was optimized by a compiler to ran on a CPU in just 2 instruction cycles. This is because:

```
int a[3], b[3], c[3];
c[0] = a[0] + b[0];
c[1] = a[1] + b[1];
c[2] = a[2] + b[2];
```

A. The CPU is superscalar

B. The CPU has SIMD instructions

C. Both A and B have to be true

D. Just A or B is true

viii. A new startup company has designed a new quad core CPU that can automatically change clock frequency to improve performance of every core thereby improving performance of a multithreaded program. Such a CPU is an example of:

A. Instruction Level Parallelism (ILP)

B. Automatic Hardware Parallelism

C. Implicit parallelism

D. Explicit parallelism

E. None of the above

ix. Under Flynn's taxonomy, A superscalar CPU is:

A. SISD

B. SIMD

C. MISD

D. MIMD

E. None of the above

x. In order to improve program performance, a SISD CPU that requires X µsec to execute 1 instruction, was redesigned to operate as a 4-way superscalar processor. The theoretical CPI of the new CPU is:

A. 4 (increases 4 times)

B. 1 (remains unchanged)

C. 0.25 (decreases to a quarter)

D. 0.5 (decreases to half)

E. None of the above

xi. In order to improve performance, a SISD CPU that requires X µsec to execute 1 instruction, was redesigned to operate as a 4-way superscalar processor. The time to execute one instruction in the new CPU would be:

A. 4 X µsec

B. 1 X µsec

C. 0.25 X µsec

D. 0.5 X µsec

E. None of the above

xii. When the `-fopenmp -D_GLIBCXX_PARALLEL` flag is used to compile C++ programs that use standard algorithms, the runtime of the program decreases because:

A. The compiler make the program simpler

B. The compiler utilizes multiple threads to improve performance

C. The compiler makes better use of caches to improve performance

D. The compiler eliminates hazards in the pipeline.

xiii. The best metric as reported by `/usr/bin/time` that provides a clear and conclusive information that a program is utilizing SIMD instructions is:

```
$ /usr/bin/time ./magic
52.37user 3.45system 0:20.79elapsed 268%CPU
```

A. User time
B. System time
C. Elapsed time
D. The %CPU used
E. None of the above

xiv. The relationship between user time and elapsed time of a single threaded program that is optimized to use SSE instructions is:

A. User time is always less than or equal to elapsed time
B. User time is always greater than or equal to elapsed time
C. The user time and elapsed time are unrelated runtime metrics of a program
D. All of the above.
E. None of the above.

xv. A C++ program compiled with `-O2` and `-O3` optimization takes x_2 and x_3 seconds to run respectively. Which one of the following relationships between x_2 and x_3 will always be true?

A. $x_2 \leq x_3$
B. $x_3 \leq x_2$
C. Relationship between x_2 and x_3 is undefined
D. None of the above.

Short Answer Questions [15 Points]

2. Briefly (2 to 3 sentences) the concept of SIMD instruction. What is the advantage of SIMD instructions? [**3 points**]

   SIMD is an acronym for Single Instruction Multiple Data. It refers to a family of instructions that can perform the same operation (such as: addition, multiplication, comparison etc.) on more than 1 pair of data items. For example, in x86 CPUs one SIMD instruction can replace 8 instructions (used to add 8 pairs of numbers) thereby improving performance.

3. What are SSE instructions? Compare and contrast SSE instructions with SIMD instructions. [**4 points**]

   SSE is an acronym for Streaming SIMD extensions. SSE refers to a family of instructions that repeatedly use SIMD operations to process a vector or array of values. In SSE the data is streamed into the CPU by streamlining the memory fetching operations improving efficiency of RAM and cache operations.

   Unlike SIMD operations that deal with fixed data sizes, SSE instructions handles a varying number of values. Unlike SIMD or Superscalar operations, SSE instructions interact with CPU caches to streamline/optimize access to RAM.

4. In the space below tabulate two significant differences contrasting implicit vs. explicit parallelism. [**4 points**]

Implicit Parallelism	***Explicit Parallelism***
Parallelism is automatically extracted by CPU. Typically little or no programmer intervention is needed.	The programmer is responsible for suitably designing the program to extract and realize parallelism.
Limited to a single core or CPU.	Can utilize multiple cores, multiple CPUs, and even distributed memory architectures.

5. With reference to the runtime statistics reported by `/usr/bin/time` program, briefly (1 or 2 sentences) is describe what the following outputs signify: [**4 points**]

   a. What does the value of "`System time`" indicate?

   The "System time" value indicates the time spent by the operating system executing instructions on behalf of the program to perform various operations such as I/O, interacting with hardware devices (GPU), creating/stopping threads, or synchronizing threads etc.

   b. What does the value of "`Maximum resident set size`" indicate?

   The "Maximum resident set size" indicates the peak memory utilized by the program. It indicates the minimum amount of virtual memory that would be needed to run the program and the minimum amount of physical memory that would be needed to run the program with peak performance.

Quantitative Analysis Questions [10 points]

6. As a HPDC specialist you have been assigned the task to evaluate the effectiveness of two different optimizations, code named Opt-α and Opt-β, based on the following timing measurements.

Execution Timings with Opt-α (in seconds)
8
9
10
11
12

Execution Timings with Opt-β (in seconds)
10
19
15
17
14

Given the above raw execution timings, you must now decide which of the two optimizations is better by answering the following questions:

a. From the above data compute and report the average execution time and 95% CI for Opt-α below. [**3 points**]

USE THIS SPACE (AS NEEDED) FOR ANY INTERMEDIATE STEPS OR OTHER CALCULATIONS

The average (and 95%) CI for Opt-α = 10.0±1.76 seconds

b. From the above data compute and report the average execution time and 95% CI for Opt-β below. [**3 points**]

USE THIS SPACE (AS NEEDED) FOR ANY INTERMEDIATE STEPS OR OTHER CALCULATIONS

The average (and 95%) CI for Opt-β = 15.0±3.7 seconds

c. Based on the above statistics (that you calculated and reported in the previous page) which optimization is better? [**1 points**]

Optimization Opt-β is better than Opt-α.

d. Provide a brief (no more than 2-3 sentences) justification for the above choice/decision in the space below. [**3 points**]

Although the averages are different, when the 95% CI values are accounted for, there is a little bit of overlap between the two observations:

1. 10.0 + 1.12 = 11.76 seconds
2. 15.0 - 3.70 = 11.30 seconds

However the overlap is not of practical significance. Consequently it can be concluded that Opt-β is better.

Code Analysis Questions [20 points]

8. Briefly describe the source of control hazard in the `getName` method below and rewrite the method to suitably resolve the control hazard. Briefly describe how your revised solution resolves the control hazard [**5 points**]

```
const std::unordered_map<int, std::string> Name =
    { "mon", "tue", "wed", "thu", "fri", "sat", "sun"};

std::string getName(unsigned day) {
    return Name[day % 7];
}
```

The source of the control hazard arises from accessing elements in the `unordered_map` which requires computation of hash values and managing the hash map.

```
const std::string Name[] =
    { "mon", "tue", "wed", "thu", "fri", "sat", "sun"};

std::string getName(unsigned day) {
    return Name[day % 7];
}
```

The above code has been simply rewritten to use an array of values to resolve the control hazard as array access does not require calculation of hash values etc and are the fastest and simplest form of memory access.

9. It is hypothesized that `std::vector` provides the same performance as a standard array to randomly access various entries. Implement a benchmark program that can be used to test the hypothesis. Indicate lines to comment/uncomment to perform the different benchmarking. [**8 points**]

```
const int Size = 100000;
int arr[Size];
std::vector <int> vec(Size);

int main() {
	for (int i = 0; (i < 10000000); i++) {
		const int index = std::rand() % Size;
		// One of the following 2 lines must be uncommented
		arr[index]++;
		// vec[index]++;
	}
	return 0;
}
```

10. Briefly (1 to 2 sentences) describe the shortcoming(s) [say why it is a shortcoming/issue] of the `isPallindrome` method that checks to see if a long/large string is a palindrome. Next rewrite the method to make it efficient by addressing the shortcomings you identified but without changing its functionality. Simplicity & efficiency of your revised solution is key to earn full credit. [**7 points**]

```
bool isPallindrome(std::string bigStr) {
  std::string rev(bigStr.rbegin(),
                  bigStr.rend());
  return (bigStr == rev);
}
```

There are two shortcomings with the method:

- The parameter must be accepted as a reference rather than a value to avoid making temporary copy of a big string, which increases memory footprint and negatively impacts performance/efficiency.
- A second copy of the big string is made in `rev`, which can be avoided using a more efficient palindrome detection algorithm shown below.

```
bool isPalli(const std::string& bigStr) {
  size_t i = 0, j = bigStr.size() - 1;
  for(; (i > j) && (bigStr[i] == bigStr[j]); i++, j--) {}
  return (i >= j);
}
```

Programming Questions [35 Points]

11. Complete the `Rational` class (also used in lab exercises) that represents a rational number (with a numerator and denominator value) with the following operations. Ensure you use the `const` keyword appropriately (each method has at least 1 point reserved for appropriate use of the `const` keyword). A skeleton of the `Rational` class is provided in the next page with certain selected methods **[Total: 16 Points]**

Instance variables	**Description**
`int num`	The numerator of the rational number
`int den;`	The denominator of the rational number (never zero)
Methods	**Description**
Default constructor	Sets numerator to 0, denominator to 1 **[2 points]**
Copy constructor	Standard copy constructor **[2 points]**
Constructor that accepts numerator and denominator as parameters	Initializes the corresponding instance variables with appropriate parameters. **[2 points]**
Addition operator (`operator+`)	Returns a new `Rational` object by appropriately adding the numbers together using the formula: $\frac{a}{b}+\frac{c}{d}=\frac{a*d+c*b}{b*d}$ **[3 points]**
Less-than operator (`operator<`)	Suitably compares another `Rational` number (with `this`) using normalized denominators similar to addition operator. **[3 points]**
Stream insertion operator (`operator<<`)	Inserts numerator and denominator separated by a "/" (between them) to the given output stream. **[2 points]**
Stream extraction operator (`operator>>`)	Reads numerator and denominator from a given input stream (assuming they are separated by a white space) **[2 points]**

A skeleton definition for the `Rational` class with selected methods is included on the next page to help you get started with this question. For other methods (such as the constructors and `operator<`) you are expected to write the methods from scratch.

```
#include <iostream>

class Rational {
  friend std::ostream& operator<<(std::ostream&, const Rational&);
  friend std::istream& operator>>(std::istream&, Rational&);
private:
  int num, den;
public:

  Rational operator+(const Rational& rhs) const {


     return Rational(num * rhs.den + rhs.num * den,
                     den * rhs.den);









  }


  bool operator<(const Rational& rhs) const {


      double d1 = num / (double) den;
      double d2 = rhs.num / (double) rhs.den;
      return (d1 < d2);









  }


  // Use space on the next page to appropriately implement
  // other methods in Rational class. End the Rational
  // class with }; as necessary.
```

```
// Implement default constructor below

Rational::Rational() : num(0) , den(1) {}
```

```
// Implement copy constructor below

Rational::Rational(const Rational& src) :
          num(src.num), den(src.den) {}
```

```
// Implement constructor to initialize numerator & denominator

Rational::Rational(int num, int den) :
          num(num), den(den) {}
```

12. Assuming that the various methods in the `Rational` class in the previous question are correctly implemented provide short answers to the following questions:
    a. Complete the following `main` method that demonstrates the use of the copy constructor of `Rational` class. **[2 points]**

```
int main() {

   Rational r1(1, 1), r2(r1);









  return 0;
}
```

    b. Complete the following `main` method that demonstrates the use of the addition operator of `Rational` class. **[3 points]**

```
int main() {


   Rational r1(1, 1), r2(2, 3);
   Rational r3 = r1 + r2;









  return 0;
}
```

13. Assuming that the various methods in the `Rational` class in the previous question are correctly implemented (and those are the only methods in `Rational` class) complete the following `loadCloseTo` method that loads all `Rational` numbers from a given text file that no more than 0.25 away from a given `value` into a vector. For example, if the given `value` is 0/4 then the `loadCloseTo` method loads all rational numbers greater than -0.25 but less than +0.25. You may assume that the data in the text file is valid -- *i.e.*, numerator and denominator values for each rational number is separated by one or more white spaces. Your solution must use only algorithms and iterators (without using any traditional looping constructs) in order to ensure good performance and earn more than 50% of the points. [**9 points**]

```
std::vector<Rational>
loadCloseTo(const std::string& srcFilePath, const Rational& value) {

    Rational fourth(1, 4);
    auto nearBy = [&](const Rational &r) { return (value < r + fourth) &&
                                           (r < value + fourth); };
    std::vector<Rational> result;
    std::ifstream srcFile(srcFilePath);
    std::istream_iterator<Rational> src(srcFile), eof;
    std::copy_if(src, eof, std::back_inserter(result), nearBy);
    return result;









}
```

14. Rewrite the following method such that it will make the best possible use of an CPU which is a 8-way superscalar processor without relying on compiler optimizations [**5 points**]

```
int sum(const std::vector<int>& vec) {
    int sum = 0;
    // Assume the vector has 999 entries…
    for(int i = 0; (i < 999); i++) {
        sum += vec[i];
    }
    return sum;
}
```

```
int sum(const std::vector<int>& vec) {
    int sum = 0;
    for(int i = 0; (i < 999); i += 3) {
        sum += vec[i] + vec[i + 1] + vec[i + 2];
    }
    return sum;
}
```