CSE 443/543: High Performance Computing & Parallel Programming
Miami University
Exam 1

There are 11 questions for a total of 100 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 11 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 syntax errors in programs and C++ statements that you write in this exam.
- 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 motivation for developing multi-core architectures and parallel programs is to:

A. Surmount challenges due to Moore's Law

B. Surmount challenges due to power & thermal issues

C. Redesign programs to make them better

D. Facilitate effective interaction between computers and scientists

ii. The expansion of the acronym FLOPS is:

A. Floats Operated on Per Second

B. Floating Point Operations Per Second

C. Fast Linear Operations Per Second

D. Fast Logical Operations Per Second

iii. The net theoretical FLOPS that 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 10 TFLOP supercomputer is equivalent to

A. A 1000 MFLOP supercomputer

B. A 1024 GFLOP supercomputer

C. A 0.01 PFLOP supercomputer

D. A 0.1 GFLOP supercomputer

v. A 1 GHz can execute 500 million instructions in one second, then its CPI is:

A. 1

B. 2

C. 0.5

D. 4

vi. A microprocessor has three levels of caches, namely Lα, Lβ, and Lδ. The access times (T) of the caches follow the relation T (Lβ) < T (Lα) < T(Lδ) . Then the tier of cache that is closest to the ALU would typically be:

A. Lα

B. Lβ

C. Lδ

D. The supplied information is insufficient to distinguish the caches.

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

A. The microprocessor has is pipelined.

B. The microprocessor is superscalar.

C. The microprocessor has SSE instructions.

D. All of the above

E. None of the above

viii. A microprocessor that has 4 pipelined stages can execute an instruction in 8 nanoseconds on an average. If the design is revised to have 8 pipelined stages then, on an average, the time to execute a single instruction would be:

A. 1 nanosecond

B. 2 nanoseconds

C. 4 nanoseconds

D. 8 nanoseconds

ix. If a 2 GHz CPU with CPI of 1 takes about 10 seconds to run a program, then the number of instructions executed by the CPU would be:

A. $2 * 10^{10}$ instructions

B. $2 * 10^{6}$ instructions

C. $4 * 10^{9}$ instructions

D. $4 * 10^{6}$ instructions

x. The fragment of C++ shown in the adjacent figure was compiled to 15 instruction on machine with CPU-π and 82 instructions on machine with CPU-μ. This most likely implies that

```
double sum = 0;
for(int i = 0;(i < 360);i++){
  sum += sin(tan(i));
}
```

A. CPU-μ is RISC while CPU-π is CISC

B. CPU-μ is CISC while CPU-π is RISC

C. CPU-μ is SIMD while CPU-π is SISD

D. CPU-μ is SISD while CPU-π is SIMD

xi. The expansion for RAM is

A. Random Access Memory

B. Rapid Access Memory

C. Real Automatic Memory

D. Any one of the above depending on context

E. None of the above

xii. A new startup company has designed a special RAM that can deliver 4 or more instructions x86 instructions (that are either generated by `g++` or by a JVM) as they are retrieved from memory. Such a special RAM is a classic example of a device that is performing:

A. Instruction Level Parallelism (ILP)

B. Compiler-assisted Parallelism

C. Implicit parallelism

D. Explicit parallelism

E. None of the above

xiii. A HPC programmer changed a fragment of C++ code so that the revised fragment runs much faster on a given superscalar processor. This is an example of:

A. Instruction Level Parallelism (ILP)

B. Compiler-assisted Parallelism

C. Implicit parallelism

D. Explicit parallelism

E. None of the above

xiv. The minimum number of cores that are required to execute a concurrent program with 4 threads is:

A. 1

B. 2

C. 4

D. 8

E. None of the above

xv. Given a x86 processor CPU-μ that run at 1.5 GHz and CPU-π 2 GHz, a given program would:

A. Run faster on CPU-μ than CPU-π

B. Run faster on CPU-π than CPU-μ

C. It will run at the same speed on both CPUs

D. The supplied information is insufficient to draw conclusions.

Code Analysis Questions

2. Given the definition of the `Mystery` class shown in the adjacent figure, circle the lines of code in the `main` method below that would cause compile errors. For each line of code that will generate a compiler error:
    a. Indicate the line number
    b. Briefly describe the source of the error
    c. Rewrite the line of code so that it will resolve the compiler error [**6 points**]

```
class Mystery {
public:
  Mystery(const std::vector<int>& list);
  Mystery(std::string mutableStr);
  ~Mystery();
  int getValue(int i = 0) const;
};
```

```
1  int main() {
2    const std::string info("{1, 2, 3}");
3    Mystery m1;
4    Mystery m2({1, 2, 3});
5    Mystery m3(info);
6    m1.getValue();
7    std::cout << m2.getValue(1.0) << std::endl;
8    std::cout << Mystery::getValue(-1) << std::endl;
9    return 0;
10 }
```

You may also use the space on the next page to respond to this question.

Line 3 is invalid: No default constructor for `Mystery`. This method can be recoded as: `Mystery({})` or `Mystery("");`

Line 6 is invalid only because Line 3 is invalid. It can be rewritten as `m2.getValue();`

Line 7 is invalid because passing `double` as argument to an `int` parameter. This line can be rewritten as: `std::cout << m2.getValue(1) << std::endl;`

Line 8 is invalid as getValue() is a non-static method. This line can be written as:
`std::cout << m2.getValue(1) << std::endl;`

This space may be used for responding to code analysis question on the pervious page.

Programming Questions

3. Complete the following `digitCount()` function that must return the number of factional digits in a given real number in a memory efficient manner. The adjacent table provides a few examples illustrating the return value from this function for a few sample values of the parameter `num`. [**7 points**]

num (parameter)	**Return Value**
12345.0625	4
-0.5	1
1	0
0.25	2
-0.00390625	8

```
int digitCount(const double num) {

  // To ensure memory efficiency the solution does
  // not use strings.

  double temp   = abs(num - (long) num);
  int    digits = 0;
  while (temp > 0) {
    digits++;
    temp *= 10;
    temp  = temp - (long) temp;
  }
  return digits;









}
```

4. Complete the following matrix transpose function that must transpose a given square matrix. An example 3x3 (`size==3`) square matrix before and after transposing is shown in the adjacent figures for your reference. [**6 points**]

1	2	3
4	5	6
7	8	9

Before

1	4	7
2	5	8
3	6	9

After

```
void transpose(std::vector<std::vector<double>>& matrix) {
    int size = matrix.size();
    for(int r = 0; (r < size); r++) {
        for(int c = 0; (c < size); c++) {
            std::swap(matrix[r][c], matrix[c][r]);
        }
    }

}
```

5. Assuming a given `source` file contains only integers, complete the following `copy` method that must copy all negative numbers from the `source` file to a given `destination` file using only algorithms and integrators (without using any traditional looping constructs) **[8 points]**

```
void copy(const std::string& source, const std::string& destination) {
    std::ifstream inFile(source);
    std::ofstream outFile(destination);
    std::istream_iterator<int> src(inFile), eof;
    std::ostream_iterator<int> dest(outFile);
    std::copy_if(src, eof, dest, std::bind2nd(std::less<int>(), 0));
}
```

6. Complete the following method that must print the most frequently occurring string in the given list. For example if the method is called with the list `{"a", "b", "a", "c", "d", "c", "a"}`, the method `freq` must print `"a"` as the most frequently occurring string. You may assume the list has at least one entry in it. You may use conventional loops and/or algorithms as you see fit [**11 points**]

```
void freq(const std::vector<std::string>& list) {
    std::vector<std::string> copy(list);
    std::sort(copy.begin(), copy.end());
    std::string freqWord = copy[0];
    std::string prevWord = copy[0];
    int maxFreq = 1, currFreq = 1;
    for(size_t i = 0; (i < copy.size()); i++) {
        if (prevWord != copy[i]) {
            if (maxFreq < currFreq) {
                freqWord = prevWord;
                maxFreq  = currFreq;
            }
            currFreq = 0;
        }
        prevWord = copy[i];
        currFreq++;
    }
    std::cout << freqWord << std::endl;
}
```