CS601: Software Development for Scientific Computing

Maximum Points: 25 | Mid-semester examination | 25/09/2024, 2PM to 4PM

Instructions: This exam has two parts. Part I is open-book, open-notes (printed/written). No electronic devices allowed. Part II is take-home. The submission instructions for part II are the same as in programming assignment 1. State your assumptions (if any) clearly.

Part I **(19 points)**:

1. From naïve python to the multi-core C++ program execution of matrix-multiplication kernel, you have seen multiple optimizations in class. Name any seven generic optimization ideas discussed in class. Note that you should write generic answers and not specific ones such as -O0, -O1, -O2, -O3, -O4 and count 5. **(3.5 points)**
2.

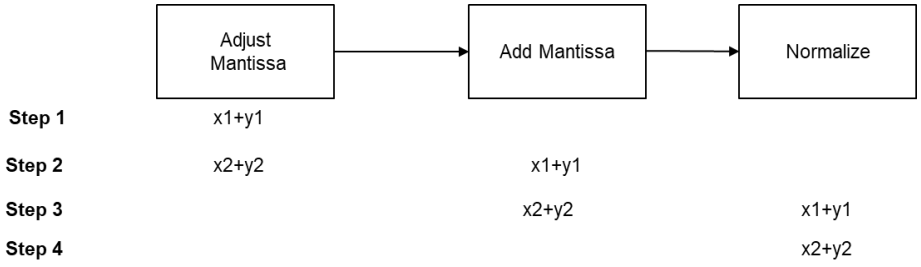
```
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/testcode/midsem/q2$ ls
Makefile  clean
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/testcode/midsem/q2$ cat Makefile
CXX=g++
CFLAGS=-I./inc
ifeq ($(DEBUG), 1)
CFLAGS += -g
endif

clean:
        rm -f clean
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/testcode/midsem/q2$ make clean
make: 'clean' is up to date.
nikhilh@ndhpc01:/mnt/c/temp/Nikhil/testcode/midsem/q2$
```

a. In the adjacent figure, the command `make clean` doesn't execute the recipe for rule named `clean`. Why? How do you make it to fire the rule `clean`? Note that you can only modify the `Makefile` and nothing else in the directory. **(2 points)**

b. How do you invoke the make command to include the `-g` option in CFLAGS **(1 point)**

3. a. Consider how addition of floating-point numbers is done: firstly, the mantissas of the two numbers to be added are adjusted so that the exponents are aligned, secondly, the mantissas are added, lastly the added number is 'normalized'. Assume that these three steps are implemented in hardware as three independent pipelined units each. The figure shows how two floating point additions x1+y1 and x2+y2 are done. Assuming that it takes 3-cycles to complete one floating point addition, how many floating point additions can you accomplish in 11 cycles? Illustrate. **(3 points)**



b. Assume that the three units mentioned previously support vector hardware and there are 32 vector registers of width 512-bit in a processor architecture. Also assume that each unit now operates only on vector registers and consumes exactly one cycle each while operating on the data held in those registers. How many IEEE-754 double-precision floating point additions (of the form x1=x1+y1) can you accomplish at the end of 3rd cycle? Explain. **(2 points)**

c. Consider computing dot product using two code snippets as shown. Assume that each fused multiply-add operation takes 3 cycles and there is a 3-stage pipelined fused multiply-add hardware, where the pipelined units are independent. Also assume that n is always a multiple of 4. Which code snippet (left or right) would execute faster and why? **(1.5 points)**

```
sum0 = sum1 = sum2 = sum3 = 0.0;
 for (i = 0; i < n; i += 4 ){
  sum0 = sum0 + a[i ]*b[i ];
  sum1 = sum1 + a[i+1]*b[i+1];
  sum2 = sum2 + a[i+2]*b[i+2];
  sum3 = sum3 + a[i+3]*b[i+3];
 }

sum = sum0 + sum1 + sum2 + sum3;
```

```
sum = 0.0;
   for (i = 0; i < n; i++ )
       sum = sum + a[i]*b[i];
```

4. Consider the program below and the gdb session for the same shown adjacently. Fill in the blanks for A, B, C, and D shown in the gdb session. **(2 points)**

```
1 #include<iostream>
2 int fact(int n){
3  if(n<0)
4     return -1;
5  if(n==0)
6     return 1;
7  return n*fact(n-1);
8 }
9
10 int main() {
11  std::cout<<fact(5);
12 }
```

```
(gdb) b fact
(gdb) r
(gdb) c
(gdb) c
(gdb) c
(gdb) bt
#0  fact (n=2) at fact.cpp:3
#1          A          in fact (n=D) at fact.cpp:  B
#2          A          in fact (n=C) at fact.cpp:  B
#3  0x00000001004010b2 in fact (n=5) at fact.cpp:7
#4  0x00000001004010d3 in main () at fact.cpp:11
(gdb)
```

4. You have implemented a recursive matrix multiplication that works perfectly when the matrix size is in power-of-two (shown below). You wish to store the input matrices in Z-order to get excellent locality. Here, A_{ij} and B_{ij} are block matrices of size n/2 x n/2 $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$

For an example 4x4 matrix shown, how would you store the matrix elements in memory? Explain. **(2 points)**

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

5. `const int x=10; int const *const y=&x; *y=100;` **(2 points)**

The above code is buggy. Does it produce warning or compiler error or runtime error? Explain your answer.

Part II – take home: (6 points) Visit the discussion forum to receive the question paper and instructions.