CS6550 Scaling to Big Data 2018
Homework 1

Max marks: 40

1. [4 marks] Consider the decimal number Q= 54.560024261474609375. The IEEE-754 single-precision representation of Q is

01000010010110100011110101110111 as can be seen from here: https://www.h-schmidt.net/FloatConverter/IEEE754.html

Now, assume Q is stored in a 32b register whose two bit storage locations are faulty. The index of these bits are 3 and 9. Here, counting starts from 0 to 31, where bit-index 0 is the left-most (most significant).

Assume that a faulty bit-location reverses the bit-value stored in it.

If Q is stored in this register, find the faulty value of Q' and absolute error (abs(Q-Q')). Now, apply the strategy of circularly shifting the value to reduce error magnitude (see the slide "using faulty/inexact hardware" in Approximate Computing presentation). In this strategy, the value is shifted right by K bits at the time of writing and shifted left by K bits at the time of reading. K is decided by the user. If Q is written, Q" is read. Find the value of Q" and abs(Q-Q"). Write values of Q' etc. in decimal notation.

2. [4 marks] Consider sharing 2000 iterations among 10 threads in OpenMP using "dynamic" scheduling. Each iteration has fixed amount of work. Lets use that as the unit of time. Assume thread 0 arrives late by 110 units and thread 7 arrives late by 50 units. Find the completion time for this whole task for a chunk size of (a) 2 and (b) 10.

3. [2 marks] Consider the following OpenMP program.

As you can see, Program1 has a parallel region, then a sequential region and then again a parallel region.

We would like to have as large parallel regions as possible and hence, we change Program1 to Program2 below.

```
//Program1
#pragma omp parallel
{
   #pragma omp for
   for (…) {  /* Parallel loop 1 */ }
}
opt = opt + N; //sequential
#pragma omp parallel
{
   #pragma omp for
   for(…) { /* Parallel loop 2 */ }
   #pragma omp for
   for(…) { /* Parallel loop N */}
}
```

```
//Program2
#pragma omp parallel
{
   #pragma omp for
   for (…) {  /* Parallel loop 1 */ }
   #pragma omp XYZ
   opt = opt + N; //sequential
   #pragma omp for
   for(…) { /* Parallel loop 2 */ }
   #pragma omp for
   for(…) { /* Parallel loop N */}
}
```

Find the directive we should use in place of XYZ.

4. [3 marks] In the following program, add an OPENMP parallel region at the line where "TODO" is written. Also, define the variables as shared, private or firstprivate. For each of this, show the output. Discuss why do you get those results.

```
#include <stdio.h>

int main(void) {

   int var1 = 1, var2 = 2;
```

```
   /* TODO:  add pragmas  */
   {
      printf("Region 1: var1=%i, var2=%i\n", var1, var2);
      var1++;
      var2++;
   }
   printf("After region 1: var1=%i, var2=%i\n\n", var1, var2);
   return 0; }
```

---

5. [4 marks] See the program below. Parallelize the loop where "TODO" is written. Each thread should compute its own part of sum in a private variable (say partialSum). After the loop, a critical section should be used to compute the global sum in "sum variable" from all the partial sums.

```
#include <stdio.h>
#define NX 10240
int main(void) {
   double VectorA[NX], VectorB[NX];
   double sum, partialSum;    int i;
   for (i = 0; i < NX; i++) {
      VectorA[i] = 1.0/((double) (NX-i));
      VectorB[i] = VectorA[i] * VectorA[i];
   }
   sum = 0.0;
//TODO add pragmas here
   for (i = 0; i < NX; i++) {
```

```
    sum += VectorA[i] * VectorB[i];
  }
  printf("Sum: %18.1f\n", sum);
  return 0;}
```

6. [4 marks] Consider each of these applications which have unique characteristics

App1: App's branch misprediction rate is high

App2: App is I/O intensive but interrupts are not common

App3:. Kernel code and service deamon code account for 95% of total code executed

App4: It is an iterative algorithm, where the next iteration depends on the previous iteration. The data-transfer between iterations is large. Iteration 100 was executed on GPU. We need to execute iteration 101.

App5: A multimedia-intensive app

App6: The app is composed of many tasks, and the computation-load of TASK(i) increases with square of i. Where should TASK(2) and TASK(100) be executed.

App7: Its data transfer and computation account for 80%:20% of workload respectively.

For App1 to App7: on which PU (CPU or GPU) each of these Apps (or their iteration/tasks) should be executed.

App8: It is a multithreaded app with 8 threads. The threads are numbered 0 to 7 and the computations performed by a thread proportional to their thread-id. If GPU takes half the time compared to CPU in running any thread, then, ignoring data-transfer, find which threads should be run on CPU or GPU to get near-perfect load-balancing.

7. [3 marks] Consider "clustering, encoding and factorized multiplication" approach discussed in class on FPGA-based deep-learning accelerators.

Assume that K=2 (number of clusters). The CNN has 8 weights and 8 inputs which are multiplied with each other.

Here are the weights after encoding: [9, -6, -6 , 9, 9, 9, -6, 9] and here is the input [-4, -7, 9, 10, 13, 15, 3, 1]. With factorized multiplication, how many multiplications and additions are required to compute the final product.

8. [2 marks] Rewrite the following code by tiling only the first loop.

```
for ( row=0; row<R; row++)
 for ( col =0; col<C; col++)
  for ( to=0; to<M; to++)
   for ( ti =0; ti <N; ti++)
 for ( i =0; i<K; i++)
  for ( j =0; j<K; j++)
    Output_fmaps [to] [row] [col]
+= Weights [to] [ti] [i] [j] *
Input_fmaps[ti] [S*row+i] [S*col+j]
```

9. [2 marks] Assuming that memory is laid out such that

Array[z][y][x] is close to Array[z+1][y][x] whereas

Array[z][y][x] is far from Array[z][y+1][x] and Array[z][y][x+1]

Whether this pseudo-code will have good cache hit ratio. If no, rewrite the code to improve cache hit ratio.

```
for(i = 0 to 99)
 for(j =0 to 99)
   for(k=0 to 99)
     sum = sum+ Array[i][j][k];
```

10. [3 marks] Consider two matrices: [1 1 0] and

$$\begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{matrix}$$

Both the matrices are binarized representation of BNN to be stored in the memory. (As you can see, the second matrix is a 3*3 matrix, the first one is a 1*3 matrix).

Show the result of XNOR operation between them and then, how binary-count is performed to get the final answer (product of the two matrices).

11. [3 marks] Assume a GPU SM (streaming multiprocessor) has 64 registers. Each register is 32b. The following code executes on GPU. Write in which location will the variable/array (shown in table below), be stored.

```
device int maxValue ;

__global__ void render ( char * ABC, int width , int height ) {

unsigned int x_dim = blockIdx.x * blockDim.x + threadIdx.x ;

unsigned int y_dim = blockIdx.y_blockDim.y + threadIdx.y;

int iteration = 0 ;

int pqr [ 4 ] ;

//some code which modifies iteration

if ( iteration == 256)

ABC[ x_dim ] = 10 ;

}
```

Variable	Location ?
x_dim	
y_dim	
iteration	
pqr	
ABC	
maxValue	

12. [1 mark] Rewrite the code after loop-unrolling, so that the loop executes only 50 times.

```
int x;

 for (x = 0; x < 1000; x++)  {

    process(x);

 }
```

13. [2 marks] In this code, put correct qualifier (e.g., __global__) in front of each function, i.e., replace __??__ with the suitable qualifier. Do not assume any default qualifier.

```
#define N 16
__??__ void addFunc1(int *a, int *b, int *c) {
 *c = *a + *b; }
__??__ void addFunc2(int *a, int *b, int *c) {
 for(int i=0; i< N; i++)
   addFunc1(a+i, b+i, c+i);  }
__??__ void random_ints(int* x, int size) {
    for (int i=0;i<size;i++)
       x[i]=rand()%50;    }

__??__ int main(void) {
int *a, *b, *c; int *d_a, *d_b, *d_c;
int size = N * sizeof(int);
cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size);
cudaMalloc((void **)&d_c, size);
a = (int *)malloc(size); random_ints(a, N);
b = (int *)malloc(size); random_ints(b, N); c = (int *)malloc(size);
cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
addFunc2<<<1,1>>>(d_a, d_b, d_c);
cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
free(a); free(b); free(c); cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
}
```

14. [2 marks] Consider the 32b value Z=01000000101101100110011001100110, which is single-precision IEEE-754 representation of the decimal number 5.7, as can be seen from here: https://www.h-schmidt.net/FloatConverter/IEEE754.html

This value is stored in a memory location. To reduce the number of bit-accessed from memory, we want to fetch only most-significant bits. Consider the following 4 strategies:

(A) fetching left-most 9 bits

(B) fetching left-most 12 bits

(C) fetching left-most 15 bits

(D) fetching left-most 18 bits

(The bits which are not fetched are assumed to be zero).

For each of the 4 strategies, find the actual (approximate) value of Z we obtain after fetching. Write your answer as a decimal number.

15. [1 mark] For the following matrix, show the result after average pooling. Perform 2*2 pooling with a stride of 2.

50	52	7	11
59	1	19	10
57	2	42	28
3	4	21	35