2017.1 Human Media Multicore Computing
Final Exam (June 16th 10am-11:20am)

supervisor signature	

StudentID# : ( ) , Name : ( )

* You may answer in either Korean or English language unless instructed to answer in English.

1. (18 points) Fill out the blanks (a)~(i) with the most appropriate English words.

- GPGPU stands for (a. G____________) (b. P____________) computing on Graphics Processing Units. GPGPU is the use of GPU, which typically handles computation only for graphics, to perform computation in applications traditionally handled by CPU.
- CUDA C/C++ keyword **__global__** indicates a function
  - is executed on (c.), and
  - is called from (d.).

  Any call to a **__global__** function must specify (e.) for that call.
- In GPU, a stream multiprocessor (SM) is basically (f.) processor that executes a warp simultaneously.
- [In OpenMP] By default, all variables declared outside a parallel block are (g.), except (h.) variable, which is (i.).

2. (10 points) [In pthread library] Consider a program statement "**pthread_join(A,B)**". Answer to following questions.

(1) What does the function **pthread_join** do? Explain **pthread_join** with sufficient details.
( )
(2) What is the purpose of the argument variable **A**? Explain **A** with sufficient details. ( )
(3) What is the purpose of the argument variable **B**? Explain **B** with sufficient details. ( )

3. (12 points) Following program in the left box intends to compute the sum between 1 and 10000 with multiple threads using OpenMP. However, the program is not correct and may generate a wrong result.
(1) Why is the program (in the left box) wrong? Explain with sufficient details. ( )
(2) Insert a correct code in the right empty box.
(3) Explain how/why your code can make the program correct. ( )

```
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 4

int main ()
{
   int i,sum=0;
   omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel for
      for (i=1;i<=10000;i++) {
         sum += i;
      }
   printf("sum = 1+2+..+10000 = %d\n",sum);
   return 0;
}
```

```
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 4

int main ()
{
   // insert your correct code here



   printf("sum = 1+2+..+10000 = %d\n",sum);
   return 0;
}
```

4.(15 points) Following pseudocode, which we learned in our class, describes parallel mergesort algorithm **Par-Merge-Sort** using a devide-and-conquer approach. Assume that the function **Par-Merge** correctly defines the parallel merge algorithm. **spawn** means creating and starting a new thread. Also, assume that we use a typical multicore computer.

(1) What is the most serious problem of the following parallel mergesort algorithm **Par-Merge-Sort**, when we run the algorithm in real software? Explain with sufficient details. ( )

(2) How can you modify the algorithm to solve the problem? Please modify or insert your code directly in the following pseudocode.

```
Par-Merge-Sort ( A, p, r )   { sort the elements in A[ p … r ] }

1.  if p < r then
2.     q ← ⌊ ( p + r ) / 2 ⌋
3.     spawn Par-Merge-Sort ( A, p, q )
4.           Par-Merge-Sort ( A, q + 1, r )
5.     sync
6.     Par-Merge ( A, p, q, r )
```

5. (20 points) Answer to following questions that are related to prefix sum by filling out empty boxes with appropriate pseudocodes.

(a) In prefix sum algorithm, input is a sequence of n elements $\{x_1, x_2, \ldots, x_n\}$ with a binary associative operation (binary addition) denoted by $\oplus$, and output is $\{s_1, s_2, \ldots, s_n\}$, where $s_i =$ [ ] for $1 \leqq i \leqq n$.

(b) Fill out the empty box in the following pseudocode for parallel prefix sum algorithm, which uses divide-and-conquer approach.

ParallelPrefixSum $(\langle x_1, \ldots, x_n \rangle, \oplus)$
  if n=1 then
    $s_1 \leftarrow x_1$;
  else {

[ ]

  }
  return $\langle s_1, \ldots, s_n \rangle$ **;**

6.(25 points) Consider following CUDA code that multiplies two matrices, M and N, with arbitrary size. (1) <u>Insert appropriate code into empty boxes (a) ~ (h).</u> In the code, BLOCK_SIZE and WIDTH represents the size of a block and the width of matrices.

```
#include <stdio.h>
#include <sys/time.h>
#define BLOCK_SIZE 32
#define WIDTH 1027

typedef struct {
   int width;
   int height;
   float* elements;
} Matrix;

__global__ void MatrixMulKernel(Matrix M, Matrix N, Matrix P)
{
  (a)
}

Matrix FuncA(const Matrix M) {
  (b)
}

void FuncB(Matrix Mdevice, const Matrix Mhost) {
  (c)
}

void FuncC(Matrix M) {
  (d)
}
```

```
void FuncD(Matrix Mhost, const Matrix Mdevice){
  (e)
}

void FreeMatrix(Matrix M) { free(M.elements); }

void MatrixMulOnDevice(const Matrix M, const Matrix N, Matrix P) {
  Matrix Md = FuncA(M);
  FuncB(Md, M);
  Matrix Nd = FuncA(N);
  FuncB(Nd, N);
  Matrix Pd = FuncA(P);
  FuncB(Pd, P);

  dim3 dimGrid( (f) , (g) );

  dim3 dimBlock(BLOCK_SIZE,BLOCK_SIZE);

  MatrixMulKernel (h) ;

  FuncD(P, Pd);

  FuncC(Md);  FuncC(Nd); FuncC(Pd);
}

Matrix AllocateMatrix(int height, int width) {
  Matrix M;  M.width = width;   M.height = height;
  int size = M.width * M.height;
  M.elements = (float*) malloc(size*sizeof(float));
  for (unsigned int i = 0; i < M.height * M.width; i++)
    M.elements[i] = 1.0;
  return M;
}

int main(void) {
  Matrix  M  = AllocateMatrix(WIDTH, WIDTH);
  Matrix  N  = AllocateMatrix(WIDTH, WIDTH);
  Matrix  P  = AllocateMatrix(WIDTH, WIDTH);
  MatrixMulOnDevice(M, N, P);
  cudaDeviceSynchronize();
  FreeMatrix(M);   FreeMatrix(N);   FreeMatrix(P);
  return 0;
}
```