2015.1 Human Media Multicore Computing
Final Exam (June 15th 11am-12:00pm)

supervisor signature	

StudentID# : (   ) ,   Name : (   )

* You may answer in either Korean or English.

1. (30points) Fill out the blanks (a)~(j) with the most appropriate English words.

- In OpenMP, by default all variables declared outside a parallel block are shared, except for (a.   ) variable which is private.

- In OpenMP, values of private variables are (b.   ) on entry and exit of parallel region.

- In GPU, a stream multiprocessor (SM) is basically (c.   ) processor that executes a warp simultaneously.

- [In CUDA] Any call to a __global__ function must specify (d.   ) for that call.
  List three other main characteristics of __global__ function :
  - (e.   )
  - (f.   )
  - (g.   )

- [In CUDA] Threads share data via shared memory within (h.   ).

- In pthread programming, (i.   ) subroutine blocks the calling thread until the specified thread terminates. The programmer is able to obtain the target thread's termination return status if it was specified in the target thread's call to (j.   ).

※ You should fill out the blank (i) and the blank (j) with appropriate pthread library function names.

2. (10points) Fill out empty box with appropriate OpenMP code <u>to execute following code with multithreads.</u>

```
#include <omp.h>
#include <stdio.h>

#define NUM_THREADS 4

int main ()
{
          int i,sum=0;
          omp_set_num_threads(NUM_THREADS);
          ┌──────────────────────────────────────────────────────────────┐
          │                                                              │
          └──────────────────────────────────────────────────────────────┘
                    for (i = 1; i <= 10000; i++) sum+=i;


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

          return 0;
}
```

<u>**Example of Execution Output Result:**</u>
sum = 1+2+..+10000= 50005000

3.(10points) [In CUDA] there are four built-in variables that specify the grid and block dimensions and the block and thread indices. What are they?

(a.   ),   (b.   ),   (c.   ),   (d.   )

4. (10points) In CUDA code, calling cudaMalloc() function and cudaMemcpy() function is often necessary. Fill out the blanks below.

(a) Function cudaMalloc() allocates object in (   ) memory.

(b) cudaMemcpy() is called for memory data transfer. It requires four parameters. What are the four parameters?
Be accurate in your answers.

(i.   ), (ii.   ), (iii.   ), (iv.   )

5. (20points) Answer to following questions that are related to prefix sum by filling out empty box 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 $s_i =$ (a) for $1 \leqq i \leqq n$.

(b) Fill out the empty boxes in the following pseudo-code for parallel prefix sum algorithm, which is executed in parallel.

```
-----------------------------------------------------------------------------
ParallelPrefixSum $(\langle x_1,\ldots,x_n\rangle,\oplus)$
if n=1 then
    $s_1 \leftarrow x_1$
else
    parallel for $i \leftarrow 1$ to n/2 do
        $y_i \leftarrow x_{2i-1} + x_{2i}$
    $\langle z_1,\ldots,z_{n/2}\rangle \leftarrow$ (b)
    parallel for $i \leftarrow 1$ to n do
        if $i=1$ then $s_1 \leftarrow x_1$
        else if $i=even$ then (c)
        else (d)
return $\langle s_1,\ldots,s_n\rangle$
-----------------------------------------------------------------------------
```

6.(20points) Consider following C and CUDA code that adds two vectors using many-core GPU. Write a CUDA kernel function **add** in the box (a) that can handle vectors with arbitrary size ‘vec_size’. Insert appropriate code into the box (b),(c),(e) for memory management, and the box (d) for CUDA kernel function call. Assume that kernel function call ‘add’ should generate 128 threads per block.

```
#include <stdio.h>
#include <stdlib.h>
#define THREAD_NUM 128  // CUDA kernel ‘add’ should generate 128 threads per block

__global__ void add(int *a, int *b, int *c, int vec_size) {

  (a)

}

int main(void) {
       int N, *a, *b, *c, *d_a, *d_b, *d_c;
       printf("vector size :");
       scanf("%d",&N); // get the size of vectors as a user input from keyboard

       (b)

       // Alloc space for host copies of a, b, c and setup input values
       a = (int *)malloc(N*sizeof(int)); vector_init(a, N);
       b = (int *)malloc(N*sizeof(int)); vector_init(b, N);
       c = (int *)malloc(N*sizeof(int));

       (c)

       add (d)

       (e)

       for (int i=0;i<N;i++)
              printf("a[%d]=%d , b[%d]=%d, c[%d]=%d\n",i,a[i],i,b[i],i,c[i]);
       free(a); free(b); free(c); cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
       return 0;
}
```

```
void vector_init(int* x, int size)
{
       int i;
       for (i=0;i<size;i++) {
              x[i]=i;
       }
}
```

Example of Execution Output Result:

```
vector size: 1234567     <---- user input

a[0]=0 , b[0]=0, c[0]=0
a[1]=1 , b[1]=1, c[1]=2
a[2]=2 , b[2]=2, c[2]=4
a[3]=3 , b[3]=3, c[3]=6
a[4]=4 , b[4]=4, c[4]=8
a[5]=5 , b[5]=5, c[5]=10
a[6]=6 , b[6]=6, c[6]=12
a[7]=7 , b[7]=7, c[7]=14
a[8]=8 , b[8]=8, c[8]=16
a[9]=9 , b[9]=9, c[9]=18
a[10]=10 , b[10]=10, c[10]=20
a[11]=11 , b[11]=11, c[11]=22
a[12]=12 , b[12]=12, c[12]=24

...

a[1234564]=1234564  , b[1234564]=1234564,
c[1234564]=2469128
a[1234565]=1234565  , b[1234565]=1234565,
c[1234565]=2469130
a[1234566]=1234566  , b[1234566]=1234566,
c[1234566]=2469132
```