2018.1 Human Media Multicore Computing
Final Exam (June 18th, 7pm-8:20pm)

supervisor signature	

StudentID# : (   ) , Name : (   )

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

1. (26 points) Fill out the blanks (a)~(m) with the most appropriate English words.

- In CUDA, (a.   ) is a group of 32 threads where multiprocessors execute the same instruction at each clock cycle.

- CUDA keyword (b.   ) indicates a function that runs on device, can only be callable from host code, and should have (c.   ) return type.
  CUDA keyword (d.   ) indicates a function that runs on host and can only be callable from host code.
  CUDA keyword (e.   ) indicates a function that runs on device and can only be callable from device code.

- [In CUDA] Because it is on-chip, (f.   ) is much faster than local and global memory. In fact, (same as (f)) latency is roughly 100x lower than uncached global memory latency. (same as (f)) is allocated per (g.   ). So, (same as (f)) enables cooperation between threads in (same as (g)).

- In OpenMP, variables declared in a parallel block are always (h.   ).

- The word 'mutex' is from the abbreviation of (i. Mut__________ Ex__________).

- [In pthread] **pthread_mutex_trylock** behaves identically to **pthread_mutex_lock**, except that it does not (j.   ) the calling thread if the given mutex is already locked by another thread. Instead, **pthread_mutex_trylock** (k.   ) immediately with the error code **EBUSY**.

- In pthread programming, (l.   ) 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 (m.   ).

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

2. (12 points) Fill out empty boxes with appropriate OpenMP code to execute following code for parallel matrix-vectoc multiplication.

```
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 4   // number of threads to create
#define M 8
#define N 8
void mxv_row(int m,int n,int *a,int *b,int *c);

int main(){
  int *Mat, * Vec, *Result_Vec, i,j;
  Mat = (int*)malloc(M*N*sizeof(int));
  Vec = (int*)malloc(N*sizeof(int));
  Result_Vec = (int*)malloc(N*sizeof(int));

  for (i=0;i<M;i++)
    for (j=0;j<N;j++) Mat[i*N+j]=i+j; // Mat(i,j) = i+j;
  for (j=0;j<N;j++) Vec[j]=j; // vector initialization

  mxv_row(M,N,Result_Vec,Mat,Vec);

  for (j=0;j<N;j++) printf("%d ",Result_Vec[j]);
  return 0;
}
```

```
void mxv_row(int m,int n,int *a,int *b,int *c)
{
  int i, j;
  int sum;

  #pragma omp [ (e)                                    ]
  [
    (e)
  ]
}
```

```
-----------------------------------------------------------------
Example of Execution Output Result:
140 168 196 224 252 280 308 336
```

3.(8 points) [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. (8 points) In OpenMP code, explain with sufficient details:

(i) the meaning of "**#pragma omp parallel for schedule (static,n)**" :

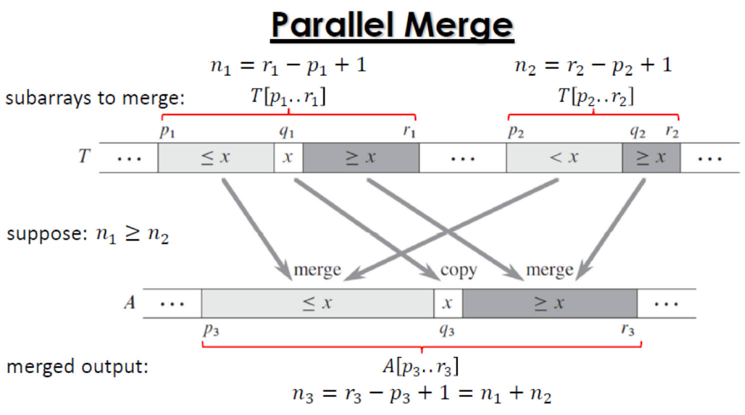
(   )

(ii) the meaning of "**#pragma omp parallel for schedule (dynamic, n)**":

(   )

5. (18 points) Fill out an empty box in the following pseudo-code for parallel-merge algorithm which is used for parallel mergesort.

```
----------------------------------------
ParallelMerge ( $T$ , $p_1$ , $r_1$ , $p_2$ , $r_2$ , $A$ , $p_3$ )
1. $n_1 \leftarrow r_1 - p_1 + 1$ , $n_2 \leftarrow r_2 - p_2 + 1$
2. if  $n_1 < n_2$  then
3.      $p_1 \leftrightarrow p_2$ , $r_1 \leftrightarrow r_2$ , $n_1 \leftrightarrow n_2$
4. if $n_1 = 0$ then return
5. else
----------------------------------------
```



6.(28 points) Consider applying a 1D stencil to a 1D array elements. Each output element is the sum of input elements within a radius. If radius = 3, then each output element is the sum of 7 elements. Elements that are not available because their indices are below 0 or above the size of an array are assumed to be 0. For example, if an input array is {3,1,6,2,4,0,5,1,7,8} and the radius is 2, the output array becomes {10,12,16,13,17,12,17,21,21,16}. Write a CUDA kernel function **stencil1D** 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 '**stencil1D**' should generate **THREAD_NUM** (i.e. 128) threads per a block. For **stencil1D** function implementation, using only global memory is OK. (using shared memory is also OK). Your code should be grammatically correct and handle correct memory management. Be careful!

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

__global__ void stencil1D(int *a, int *b, int *c, int vec_size, int radius) {
  (a)
}

int main(void) {
      int N, Radius, *a, *c, *d_a, *d_c;
      printf("vector size, and radius of stencil :");
      scanf("%d %d",&N, &Radius); // get the size of vectors as a keyboard input
   (b)
      a = (int *)malloc(N*sizeof(int)); vector_init(a, N);
      c = (int *)malloc(N*sizeof(int));
   (c)
      stencil1D (d)
   (e)
      for (int i=0;i<N;i++) printf("a[%d]=%d , c[%d]=%d\n",i,a[i],i,c[i]);
      free(a); free(c); cudaFree(d_a); 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, and radius of stencil :
1234567 , 3   <---- user input

a[0]=0 , c[0]=6
a[1]=1 , c[1]=10
a[2]=2 , c[2]=15
a[3]=3 , c[3]=21
a[4]=4 , c[4]=28
a[5]=5 , c[5]=35
a[6]=6 , c[6]=42
a[7]=7 , c[7]=49
a[8]=8 , c[8]=56
a[9]=9 , c[9]=63
a[10]=10 , c[10]=70

...

a[1234566]=1234566 , c[1234566]=4938258
```