2014.1 Multicore Computing
Final Exam (June 17th 11am-12pm)

supervisor signature	

StudentID# : (   ) , Name : (   )

* You may answer in either Korean or English.

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

- In CUDA, threads within a block can cooperate through (a.   ). Threads in different blocks cannot cooperate.
- In CUDA, (b.   ) is a group of threads where multiprocessor executes the same instruction at each clock cycle.
- (c.   ) is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.
- Main advantages of PSRS (Parallel Sorting by Regular Sampling) over parallel quicksort and hyper-quicksort algorithms, are
    - Better (d.   )
    - Repeated communications of a same value are avoided
    - The number of processes does not have to be (e.   ), which is required by the other two algorithms.
- [In OpenMP] By default, all variables declared outside a parallel block are shared variables except (f.   ).

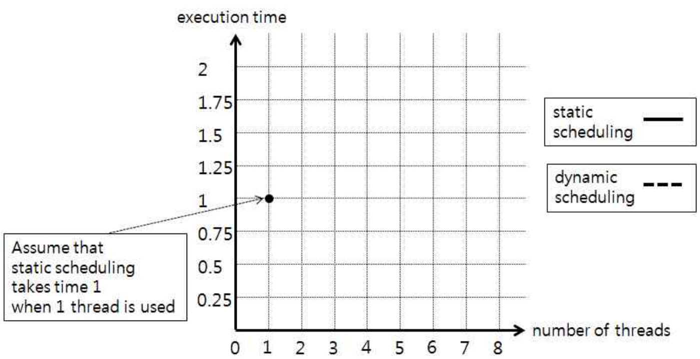
2. (30points) Project 2 in our class was to write 'C with OpenMP' codes that compute the number of prime numbers between 1 and 200000 using **static** scheduling and **dynamic** scheduling just like the source code below.

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

#define NUM_END 200000

int isPrime(int x){
    int i;
    if(x<=1) return 0;
    for(i=2; i<x; i++)
        if((x%i == 0)&&(i!=x)) return 0;
    return 1;
}
```

```
Example of Execution Output Result:
number of threads: 2     <---- user input

Answer: 17984
```

```
int main () {
    int i,NUM_THREADS;
    int counter=0;

    printf("number of threads: ");
    scanf("%d",&NUM_THREADS);

    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        #pragma omp for (c)                  schedule ( static or dynamic )
        for(i=0; i<=NUM_END; i++){
            if(isPrime(i)) counter++;
        }
    }
    printf("Answer: %d\n", counter);
    return 0;
}
```

(a) Draw a graph that has two curves where solid line curve (▬▬) represents the execution time of static scheduling and dashed line curve (▪ ▪ ▪ ▪) represents the execution time of dynamic scheduling. Assume that we execute the codes on a typical PC equipped with Intel i7 quad-core CPU (with hyperthreading ON). Assume that the code using static scheduling takes time 1 when one thread is used. You should draw the graph as accurate as possible. (Give your best prediction!)



(b) Explain why you drew the graph like above for static scheduling curve and dynamic scheduling curve. Your answer should include explanation on the performance characteristics of the two methods with respect to different number of threads and explanation on why such performance characteristics are achieved.

- static scheduling curve : (   )
- dynamic scheduling curve : (   )

(c) Fill out the blank (c) with the most appropriate codes in the above source code. Your answer : (c)

3. (14points) Answer to following questions.

(a) What is GPGPU? Explain. ( )

(b) What does a CUDA synchronization(동기화) function **`__syncthreads()`** do? <u>Explain with sufficient details. (Be specific !)</u>

( )

4. (8points) Fill out the blanks in the following pseudo-code for parallel merge algorithm that takes two sorted array $T[p_1..r_1]$ and $T[p_2..r_2]$ as input, and merge them into one sorted array $A[p_3..]$ as output, which is executed in parallel.

Par-Merge$(T, p_1, r_1, p_2, r_2, A, p_3)$

$n_1 \leftarrow r_1 - p_1 + 1,\ n_2 \leftarrow r_2 - p_2 + 1$

if $n_1 < n2$ then

    $p_1 \leftrightarrow p_2,\ r_1 \leftrightarrow r_2,\ n_1 \leftrightarrow n_2$

if $n_1 = 0$ then return

else

    $q_1 \leftarrow \lfloor (p_1 + r_1)/2 \rfloor$

    $q_2 \leftarrow$ $\boxed{\text{(a)}}$ $(T[q_1], T, p_2, r_2)$

    $q_3 \leftarrow p_3 + (q_1 - p_1) + (q_2 - p_2)$

    $A[q_3] \leftarrow T[q_1]$

    spawn $\boxed{\text{(b)}}$ $(T, p_1, q_1 - 1, p_2, q_2 - 1, A, p_3)$

    $\boxed{\text{(c)}}$ $(T, q_1 + 1, r_1, q_2 + 1, r_2, A, q_3 + 1)$

    sync

6.(30points) Consider following C and CUDA code that adds two vectors using many-core GPU. <u>Write a CUDA kernel function **add** in the box (a) that can handle vectors with arbitrary size 'vec_size'</u>. <u>Insert appropriate code into the box (b) for CUDA kernel function call. Assume that kernel function call 'add' should generate 128 threads per block.</u>

```
#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

      // Alloc space for device copies of a, b, c
      cudaMalloc((void **)&d_a, N*sizeof(int));
      cudaMalloc((void **)&d_b, N*sizeof(int));
      cudaMalloc((void **)&d_c, N*sizeof(int));

      // 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));

      // Copy inputs to device
      cudaMemcpy(d_a, a, N*sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(d_b, b, N*sizeof(int), cudaMemcpyHostToDevice);

      add  (b)
      // Copy result back to host
      cudaMemcpy(c, d_c, N*sizeof(int), cudaMemcpyDeviceToHost);

      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
```