supervisor signature	

2016.1 Human Media Multicore Computing

Final Exam (June 20th 11am-12:20pm)

StudentID# : ( ) , Name : ( )

* You may answer in either Korean or English.

1. (12points) Fill out the blanks (a)~(f) with the most appropriate English words.

- In CUDA device, threads within the same (a.) can access the same data in a (b.) memory which is much fster than global memory.
- In OpenMP, using (c.) directive, individual code blocks are distributed over threads.
- [OpenMP] By default, all variables declared outside a parallel block are (d.), except the (e.) variable, which is (f.).

2. (10 points) Explain the main difference between static scheduling and dynamic scheduling when executing parallel blocks with multiple threads in OpenMP programming by filling out following blanks.

(i) static scheduling : ( )

(ii) dynamic scheduling : ( )

3.(28points) Consider following CUDA code that multiplies two matrices with arbitrary size. **Insert appropriate code into empty boxes (a), (b), (c), (d), (e), (f), (g) and (h).** 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)
{
  int tx = (a) ;
  int ty = (b) ;
  if ( (c) ) {
    float Pvalue = 0;
    for (int k = 0; k < M.width; ++k)
    {
      float Melement = M.elements[ty * M.width + k];
      float Nelement = N.elements[k * N.width + tx];
      Pvalue += Melement * Nelement;
    }
    P.elements[ (d) ] = Pvalue;
  }
}

Matrix AllocateDeviceMatrix(const Matrix M)
{
  Matrix Mdevice = M;
  int size = M.width * M.height * sizeof(float);
  cudaMalloc((void**)&Mdevice.elements, size);
  return Mdevice;
}

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

void CopyToDeviceMatrix(Matrix Mdevice, const Matrix Mhost)
{
    int size = Mhost.width * Mhost.height * sizeof(float);
    cudaMemcpy(Mdevice.elements, Mhost.elements, size,
          cudaMemcpyHostToDevice);
}
void CopyToDeviceMatrix(Matrix Mdevice, const Matrix Mhost)
{
    int size = Mhost.width * Mhost.height * sizeof(float);
    cudaMemcpy(Mdevice.elements, Mhost.elements, size,
          cudaMemcpyHostToDevice);
}
```

```
void CopyFromDeviceMatrix(Matrix Mhost, const Matrix Mdevice)
{
    int size = Mdevice.width * Mdevice.height * sizeof(float);
    cudaMemcpy(Mhost.elements, Mdevice.elements, size,
          cudaMemcpyDeviceToHost);
}

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

  dim3 dimGrid( (e) , (f) );
  dim3 dimBlock(BLOCK_SIZE,BLOCK_SIZE);

  MatrixMulKernel (g) ;

  (h)

  FreeDeviceMatrix(Md);  FreeDeviceMatrix(Nd);
  FreeDeviceMatrix(Pd);
}

Matrix AllocateMatrix(int height, int width) {
  Matrix M;
  M.width = width;
  M.height = height;
  int size = M.width * M.height;
  M.elements = NULL;
  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) {
  int i,j;
  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;
}
```

4. (30points) 4. (1) 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,p1,r1,p2,r2,A,p3)
1. n1←r1−p1+1, n2←r2−p2+1
2. if n1 < n2 then
3.     p1↔p2, r1↔r2, n1↔n2
4. if n1 = 0 then return
5. else
6.     q1← (a)
7.     q2← Binary-Search(T[q1], T,p2,r2)
8.     q3←p3+(q1−p1)+(q2−p2)
9.     A[q3]←T[q1]
10.    spawn Par-Merge (T, (b) ,A,p3)
11.          Par-Merge (T, (c) ,A,q3+1)
12.    sync
```

---

(2) Fill out empty box to write a pseudo-code for parallel merge sorting algorithm. You may use above `Par-Merge` function.

```
Merge-Sort(A,p,r)        // sort the elements A[p ... r]
```

5. (20points) Consider following C and OpenMP code that computes the approximate value of π(pi). In the program, ten threads simultaneously execute for-loop (line 9~line 12). Fill out empty boxes in the code with appropriate C and OpenMP code. You are supposed to insert code related to computing and displaying execution time into boxes (a) and (c), and code related to specifying OpenMP parallel block into box (b).

```
1: #include <omp.h>
2: #include <stdio.h>
3: #define NUM_THREADS 10
4: long num_steps = 10000000;    double step;

5: void main ()
6: {
7:        long i; double x, pi, sum = 0.0;

   (a)

8:        step = 1.0/(double) num_steps;

   (b)

9:        for (i=0;i< num_steps; i++){
10:               x = (i+0.5)*step;
11:               sum = sum + 4.0/(1.0+x*x);
12:       }
13:       pi = step * sum;

   (c)

14:       printf("pi=%.10lf\n",pi);
15: }
```

Execution Output result:

```
execution time = 0.2637085170 second
pi=3.1415926536
```