Computación de Altas Prestaciones

Ejercicios de repaso del Tema 3

P1. Para el siguiente kernel de suma de dos vectores y el código correspondiente que se utiliza para su ejecución, conteste justificadamente las siguientes preguntas:

```
__global__
void vecAddKernel(float* A_d, float* B_d, float* C_d, int n)
{
1.      int i = threadIdx.x + blockDim.x * blockIdx.x;
2.      if(i<n) C_d[i] = A_d[i] + B_d[i];
}

int vectAdd(float* A, float* B, float* C, int n)
{
        //assume that size has been set to the actual length of
        //arrays A, B, and C
3.      int size = n * sizeof(float);
4.
5.      cudaMalloc((void **) &A_d, size);
6.      cudaMalloc((void **) &B_d, size);
7.      cudaMalloc((void **) &C_d, size);
8.      cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
9.      cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
10.     vecAddKernel<<<ceil(n/256), 256>>>(A_d, B_d, C_d, n);
11.     cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
}
```

a. Si el tamaño n de los vectores A, B y C es de 1,000 elementos, cada uno. ¿Cuántos bloques de threads se generan?

   4

b. Si el tamaño n de los vectores A, B y C es de 1000 elementos, cada uno. ¿Cuántos warps hay en cada bloque de threads?

   8

c. Si el tamaño n de los vectores A, B y C es de 1000 elementos, cada uno. ¿Cuántos threads en total se generan en el grid lanzado en la línea 10?

   1024

d. Si el tamaño n de los vectores A, B y C es de 1000 elementos, cada uno. ¿Hay divergencia de threads en la ejecución del kernel? Explique cuando se produce y cuando no, identificando el número de bloques y de warps con divergencia. Justifique identificando las líneas de código que hayan generado la divergencia para cada caso.

   Yes, there is control divergence, which is caused by the if statement in line 2. Since the total number of threads in the grid will be 1024, larger than the size of arrays, the last warp will have divergence: the first 8 threads in the warp will take the true path and the other 24 threads will take the false path.

e. Assume that the size of A, B, and C is 768 elements. Is there any control divergence during the execution of the kernel? If so, identify the line number of the statement that causes the control divergence. Explain why or why not.

No, there is no control divergence. Since the total number of threads in the grid will be 768, the same as the size of the arrays, the last warp will not have divergence: all 32 threads will take the true path.

f. As we discussed in class, data structure padding can be used to eliminate control divergence. Assuming that we will keep the host data structure size the same but pad the device data structure. Declare and initialize a new variable padded_size in line 4 and make some minor changes to statements in lines 5, 6, 7 and 10.

```
int vectAdd(float* A, float* B, float* C, int n)
{
        //assume that size has been set to the actual length of
        //arrays A, B, and C
3.      int size = n * sizeof(float);
4.      int padded_size = (ceil(n/256.0)*256) * sizeof(float);
5.      cudaMalloc((void **) &A_d, padded_size);
6.      cudaMalloc((void **) &B_d, padded_size);
7.      cudaMalloc((void **) &C_d, padded_size);
8.      cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
9.      cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
10.     vecAddKernel<<<ceil(n/256.0), 256>>>(A_d, B_d, C_d,
ceil(n/256.0)*256);
11.     cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
```

g. With this change to the host code, do you think that the “if (i<n)” is still needed in Line 2 of the original kernel? Why or why not?

It is no longer needed. Since n will always to multiples of block sizes with padding, all threads will always have an i value smaller than n. There is no longer need for the test.

h. For large vector sizes, say greater than 1,000,000 elements, do you expect that the padded code will have significant impact on performance? Why or why not?

i. There should be little performance impact. The only warp that has control divergence in the original code was the last warp. With only one out of 30000 warps affected by control divergence, there was not much

P2. Para el siguiente kernel de suma de dos vectores y el código correspondiente que
Considera el siguiente kernel CUDA y responda a las siguientes preguntas

```
__global__ void mykernel(double *in, double *out, unsigned int size) {
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    double result = 0;
    int w;

    for (w = -3 ; w <= 3 ; w++) {
        if ((index+w < size) && (index+w > 0)) {
            result += in[index + w];
        }
    }
    if (index < size)
       out[index] = result;
}
```

Considere además que la GPU de la que dispone tiene:

- 128KB de registros por SM.
- 64KB de memoria compartida por SM.
- 8GB de memoria global.
- 32 warps como máximo por bloque.

a) Si size=2^{20}, ¿cuál es el uso de memoria global? ¿Cuántos kernels simultáneos podríamos lanzar considerando solo la memoria global?

El consumo de memoria global es $2Nsizeof(double) = 2^{24} = 16MiB$. Si la memoria global es 8GiB=2^{33}, entonces, $\frac{2^{33}}{2*2^{23}} = 2^9 = 512$

b) ¿Cuántos registros (en bytes) utiliza cada hilo según las variables empleadas? En base a ello, proponga un tamaño máximo de bloque.

2*int+1*double=16B/hilo
128KiB/16B=8Ki hilos, esto es, 256 warps. Pero el máximo de warps por bloque es 32, luego esto no puede exceder de 1024.

c) Varios hilos dentro del mismo bloque acceden a las mismas posiciones de memoria. Para hacer un mejor uso de la memoria, se propone la siguiente versión **incompleta** del programa. En esta se pretende que cada hilo del bloque copie su elemento asignado a la región de memoria compartida y los hilos 0, 1 y 2 se encarguen de los elementos extra de forma equilibrada.

```
__global__ void mykernel2(double *in, double *out, unsigned int size) {
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    __shared__ double temp[*TODO1*];
    double result = 0;
    int w;

    if (index < size)
       temp[*TODO2*] = in[index];
    else
       temp[*TODO2*] = 0;

    if (threadIdx.x < 3) {
       if ((index-3+threadIdx.x < size) && ((index-3+threadIdx.x > 0))
           temp[threadIdx.x] = in[index-3+threadIdx.x];
       else
           temp[threadIdx.x] = 0;

       if ((index+blockDim.x+threadIdx.x < size) &&
                 ((index+blockDim.x+threadIdx.x > 0))
           temp[threadIdx.x+3+blockDim.x] = in[index+blockDim.x+threadIdx.x];
       else
           temp[threadIdx.x+3+blockDim.x] = 0;

    }
    __syncthreads();
    for (w = -3 ; w <= 3 ; w++) {
        if ((index+w < size) && (index+w > 0)) {
            result += temp[threadIdx.x + 3 + w];
        }
    }
    if (index < size)
       out[index] = result;
}
```

¿Cuál es el tamaño de la memoria compartida (*TODO1*)? ¿Cuál es la fórmula para rellenar la región de memoria compartida (*TODO2*)? Para tamaño de bloque 8, dibuje temp e indique qué hilo (threadIdx) escribe cada posición.

TODO1 = blockDim.x + 6
TODO2 = threadIdx.x + 3

0	1	2	0	1	2	3	4	5	6	7	0	1	2

d) Razone si hay o no divergencia en bloques distintos del primero (blockIdx.x=0) y del último (blockIdx.x=gridDim.x).

Sí, el primer warp de todos los bloques tendrá divergencia por los tres hilos que tienen que copiar elementos extra.