Universidad Nacional de San Agustin

Escuela Profesional de
Ciencia de la Computación

Algoritmos Paralelos

CUDA : Multiplicación de Matrices

Alumna :
Rosa Yuliana Gabriela
Paccotacya Yanque
Profesor:
Mg.Alvaro Hen-ry Mamani
Aliaga

Índice

1. Multiplicación de matrices **2**
1.1. Multiplicación de matrices simple y con Tiles 2
1.2. Código 2

2. Características del Dispositivo **4**

3. Análisis de Factores limitantes del dispositivo **5**
3.1. Registros 5
3.2. Memoria Compartida 5
3.3. Otros 5

4. Experimentos **6**
4.1. Tabla de resultados (ms) 6
4.2. Capturas de Pantalla 6

5. Conclusiones **7**

1. Multiplicación de matrices

1.1. Multiplicación de matrices simple y con Tiles

En la multiplicación de matrices simple, cada hebra calcula el valor de la salida, llamando a la fila y a la columna correspondiente y realizando el producto punto. La fila y columna son datos comunes para realizar el producto punto, por loq ue pueden ser almacenados en memoria compartida en vez de ser llamados de memoria local cada vez que se ejecuta una hebra.

1.2. Código

```
1
2  __global__
3  void matrixMulti(int *c, int *a, int *b,int n)
4  {
5      int row = blockIdx.y * blockDim.y + threadIdx.y ;
6      int col = blockIdx.x * blockDim.x + threadIdx.x ;
7      if ((row <n) && (col<n))
8      {
9      int suma=0;
10         for(int i=0;i<n;++i)
11         {
12           suma+=a[row*n+i]*b[i*n+col];
13         }
14         c[row*n+col] = suma;
15     }
16 }
17
18 __global__ void MatrixMulTiled(int * d_P, int * d_M, int* d_N,int Width
       )
19 {
20   __shared__ int Mds[TILE_WIDTH][TILE_WIDTH];
21   __shared__ int Nds[TILE_WIDTH][TILE_WIDTH];
22   int bx = blockIdx.x; int by = blockIdx.y;
23   int tx = threadIdx.x; int ty = threadIdx.y;
24   // Identify the row and column of the d_P element to work on
25   int Row = by * TILE_WIDTH + ty;
26   int Col = bx * TILE_WIDTH + tx;
27   int Pvalue = 0;
28   // Loop over the d_M and d_N tiles required to compute d_P element
29   for (int ph = 0; ph < Width/TILE_WIDTH; ++ph)
30   {
31     // Collaborative loading of d_M and d_N tiles into shared memory
32     if ((Row< Width) && (ph*TILE_WIDTH+tx)< Width)
```

```
33        Mds[ty][tx] = M[Row*Width + ph*TILE_WIDTH + tx];
34      if ((ph*TILE_WIDTH+ty)<Width && Col<Width)
35        Nds[ty][tx] = N[(ph*TILE_WIDTH + ty)*Width + Col];
36      __syncthreads();
37      for (int k = 0; k < TILE_WIDTH; ++k)
38      {
39        Pvalue += Mds[ty][k] * Nds[k][tx];
40      }
41      __syncthreads();
42    }
43    d_P[Row*Width + Col] = Pvalue;
44  }
45
46  int main(int argc, char *argv[])
47  {
48    srand (time(NULL));
49    int  N= strtol(argv[1], NULL, 10);
50    int THREADS_PER_BLOCK=32;
51    int *a, *b, *c; // host copies of a, b, c
52    int *d_a, *d_b, *d_c; //device copies of a,b,c
53    //int size = N*N*sizeof(int);
54    int size=N*N*sizeof(int);
55    cudaMalloc((void **)&d_a, size);
56    cudaMalloc((void **)&d_b, size);
57    cudaMalloc((void **)&d_c, size);
58    a = (int *)malloc(size);
59    llenar(a, N);
60    b = (int *)malloc(size);
61    llenar(b, N);
62    c = (int *)malloc(size);
63    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
64    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
65    int blocks= (N + THREADS_PER_BLOCK -1)/THREADS_PER_BLOCK;
66    dim3 dimGrid(blocks, blocks, 1);
67    dim3 dimBlock(THREADS_PER_BLOCK,THREADS_PER_BLOCK, 1);
68    cout<<"N: "<<N<<"\tBloques : "<<blocks<<"\t Hebras: "<<
       THREADS_PER_BLOCK<<endl;
69    cudaEvent_t start, stop;
70    float elapsedTime;
71    cudaEventCreate(&start);
72    cudaEventRecord(start,0);
73      matrixMulti<<<dimGrid,dimBlock>>>(d_c, d_a, d_b, N);
74      //MatrixMultTiled<<<dimGrid,dimBlock>>>(d_c, d_a, d_b, N);
75    cudaEventCreate(&stop);
76    cudaEventRecord(stop,0);
77    cudaEventSynchronize(stop);
78    cudaEventElapsedTime(&elapsedTime, start,stop);
79    printf("Tiempo  : %f ms\n" ,elapsedTime);
80    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
```

```
81    free(a); free(b); free(c);
82    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
83    return 0;
84  }
85  }
```

Código 1: Multiplicación de Matrices

2. Características del Dispositivo

Se trabajó en una máquina con tarjeta gráfica NVIDIA GT740-M y Cuda 8.0. Las características se detallan a continuación:

```
 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GT 740M"
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 2004 MBytes (2101542912 bytes)
  ( 2) Multiprocessors, (192) CUDA Cores/MP:     384 CUDA Cores
  GPU Max Clock rate:                            1032 MHz (1.03 GHz)
  Memory Clock rate:                             900 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 524288 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Dev
ice0 = GeForce GT 740M
```

3. Análisis de Factores limitantes del dispositivo

3.1. Registros

El Kernel de la multiplicación de matrices con tiles requiere almacenar 9 variables de tipo int(4 bytes) en registro, por lo que cada hebra requeriria 36 bytes en registros, cada registro es 4 bytes, necesitandose así como mínimo 9 registros(16, por cercanía de potencias de 2, siendo 4096 hebras por bloque como máximo,pero el dispositivo solo permite 1024 hebras por bloque, así que el número de registros no limitará las operaciones.

3.2. Memoria Compartida

En memoria compartida se necesitará la cantidad de bytes que usarán Mds y Nds (el tiplo int es de 4 bytes) $tile * tile * 4 * 2 = 8 * tile^2$ bytes.

tile=16 $16^2 * 8 = 2048bytes = 2Kb$

tile=32 $32^2 * 8 = 8192bytes = 8Kb$

tile=64 $64^2 * 8 = 32768bytes = 32Kb$

El dispositivo cuenta con 49152 bytes de memoria compartida por bloque, así la memoria compartida no limitará las operaciones.

3.3. Otros

Las limitaciones del hardware son principalmente:

- 2048 hebras por multiprocesador
- 1024 hebras por bloque
- 32 hebras en el warp

Lo recomendable es que el tamaño del bloque sea múltiplo del warp, para asi tener todas las hebras activas.Por lo que usando un blocksize de 32 o 16 maximizaría el número de hebras en el SM.

BlockSize	Nro Blocks	Nro Hebras en SM
32x32	2	2048
16x16	8	2048

Se realizarán experimentos con estas dos dimensiones para evaluar su ejecución.

4. Experimentos

4.1. Tabla de resultados (ms)

Matriz	256x256		1024x1024		4096x4096		8192x8192	
Block/Tile Size	**16x16**	**32x32**	**16x16**	**32x32**	**16x16**	**32x32**	**16x16**	**32x32**
Mult. Simple	2.28	0.59	150.6	145.04	8745.83	8370.59	77907.42	69143.46
Mult. Tiled	0.73	2.21	51.23	43.94	2764.95	2383.15	22154.74	19305.83

4.2. Capturas de Pantalla

```
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16tiled 256
N: 256  Bloques : 16     Hebras: 16
Tiempo  : 0.734656 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16simple 256
N: 256  Bloques : 16     Hebras: 16
Tiempo  : 2.282912 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16tiled 1024
N: 1024 Bloques : 64     Hebras: 16
Tiempo  : 51.235870 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16simple 1024
N: 1024 Bloques : 64     Hebras: 16
Tiempo  : 150.601761 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16tiled 4096
N: 4096 Bloques : 256    Hebras: 16
Tiempo  : 2764.952637 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16simple 4096
N: 4096 Bloques : 256    Hebras: 16
Tiempo  : 8745.834961 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16tiled 8192
N: 8192 Bloques : 512    Hebras: 16
Tiempo  : 22154.744141 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./16simple 8192
N: 8192 Bloques : 512    Hebras: 16
Tiempo  : 77907.421875 ms
```

```
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32tiled 256
N: 256  Bloques : 8      Hebras: 32
Tiempo  : 0.599552 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32simple 256
N: 256  Bloques : 8      Hebras: 32
Tiempo  : 2.210816 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32tiled 1024
N: 1024 Bloques : 32     Hebras: 32
Tiempo  : 43.943970 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32simple 1024
N: 1024 Bloques : 32     Hebras: 32
Tiempo  : 145.044449 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32tiled 4096
N: 4096 Bloques : 128    Hebras: 32
Tiempo  : 2383.159180 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32simple 4096
N: 4096 Bloques : 128    Hebras: 32
Tiempo  : 8370.592773 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32tiled 8192
N: 8192 Bloques : 256    Hebras: 32
Tiempo  : 19305.832031 ms
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/cuda$ ./32simple 8192
N: 8192 Bloques : 256    Hebras: 32
Tiempo  : 69143.468750 ms
```

5. Conclusiones

La multiplicación con tiles y memoria compartida es más rápida que la mulplicacion simple, disminuyendo la lectura de datos en un radio de 1 a 16 o 32(según sea el tamaño del Tile).

Tener un Tile múltiplo del warp mejora el tiempo ya que siempre se tiene las hebras activas.

Para el dispositivo GeForce 740M, el mejor tamaño de tile es de 32x32.