Práctica 1

Programación GPUs con CUDA

Carlos García Sánchez

UCM

9 de marzo de 2020

- "The CUDA handbook: a comprehensive guide to GPU", Nicholas Wilt
- "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", Philipp K. Janert, O'Reilly

Outline

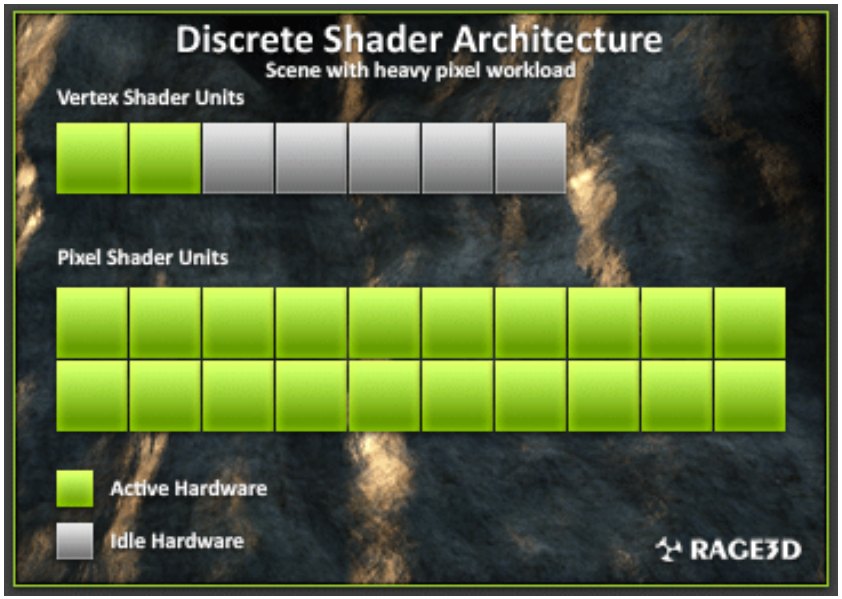
1. Objetivos
2. Arquitectura GPUs
3. CUDA
4. Ejemplos
5. Entrega

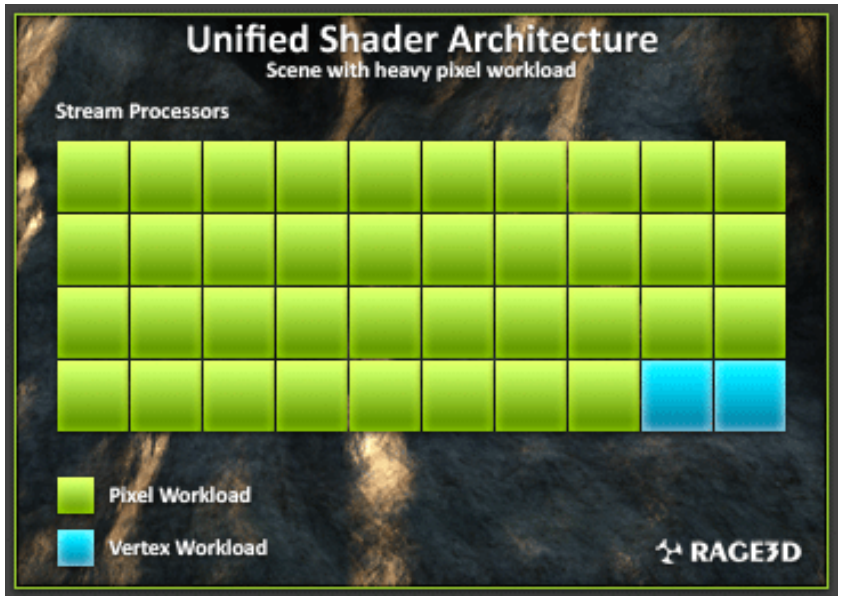
Objetivos

- Familiarizarse con la arquitectura CUDA
- Evaluar las mejoras/speedup de GPU vs CPU

Arquitectura G80/G90/GT200

- Nuevo modelo arquitectura: Arquitectura unificada de shaders
- Solo hay un tipo de unidad programable para pixels/vertex
  - Ejecuta todo los tipos de shaders
- Estructura realimentada

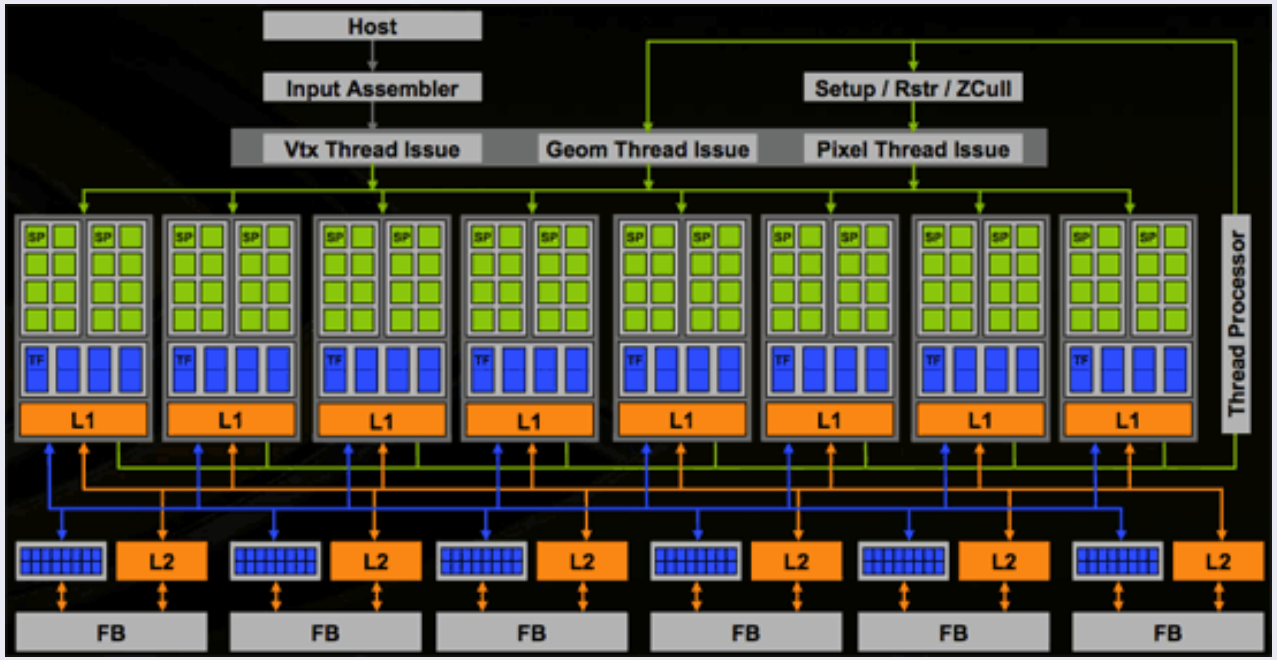




Arquitectura G80/G90/GT200

- 8 clusters
- Cada cluster tiene un planificador compartida por los 16 SP (ALUs), y unidades de textura

GPUs vistas como procesador de hilos



Entorno desarrollo

- Instalado CUDA 10.1
  - Ubuntu 18.04 por defecto instalado gcc-7.5

Entorno desarrollo

nvidia-smi

Terminal #1

```
usuario_local@profess11:~$ nvidia-smi
Fri Feb 14 14:45:31 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 430.64       Driver Version: 430.64       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce RTX 2060    Off  | 00000000:01:00.0  On |                  N/A |
| 26%   37C    P8    13W / 160W |    677MiB /  5931MiB |      2%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      5294      G   /usr/bin/gnome-shell                          97MiB |
|    0      6955      G   /usr/lib/xorg/Xorg                           229MiB |
|    0     11164      G   ...quest-channel-token=7661568818610171974   348MiB |
+-----------------------------------------------------------------------------+
```

Entorno desarrollo

nvcc

Terminal #1

```
usuario_local@profess11:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Apr_24_19:10:27_PDT_2019
Cuda compilation tools, release 10.1, V10.1.168
```

Entorno desarrollo

- Procesador Intel i7-8700 con 6 cores + SMT

cpu_info

Terminal #1

```
usuario_local@profess11:~$ more /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 158
model name  : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
stepping    : 10
microcode   : 0xca
cpu MHz     : 800.073
cache size  : 12288 KB
physical id : 0
siblings    : 12
core id     : 0
cpu cores   : 6
apicid      : 0
initial apicid  : 0
fpu     : yes
fpu_exception   : yes
cpuid level : 22
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
```

Entorno desarrollo

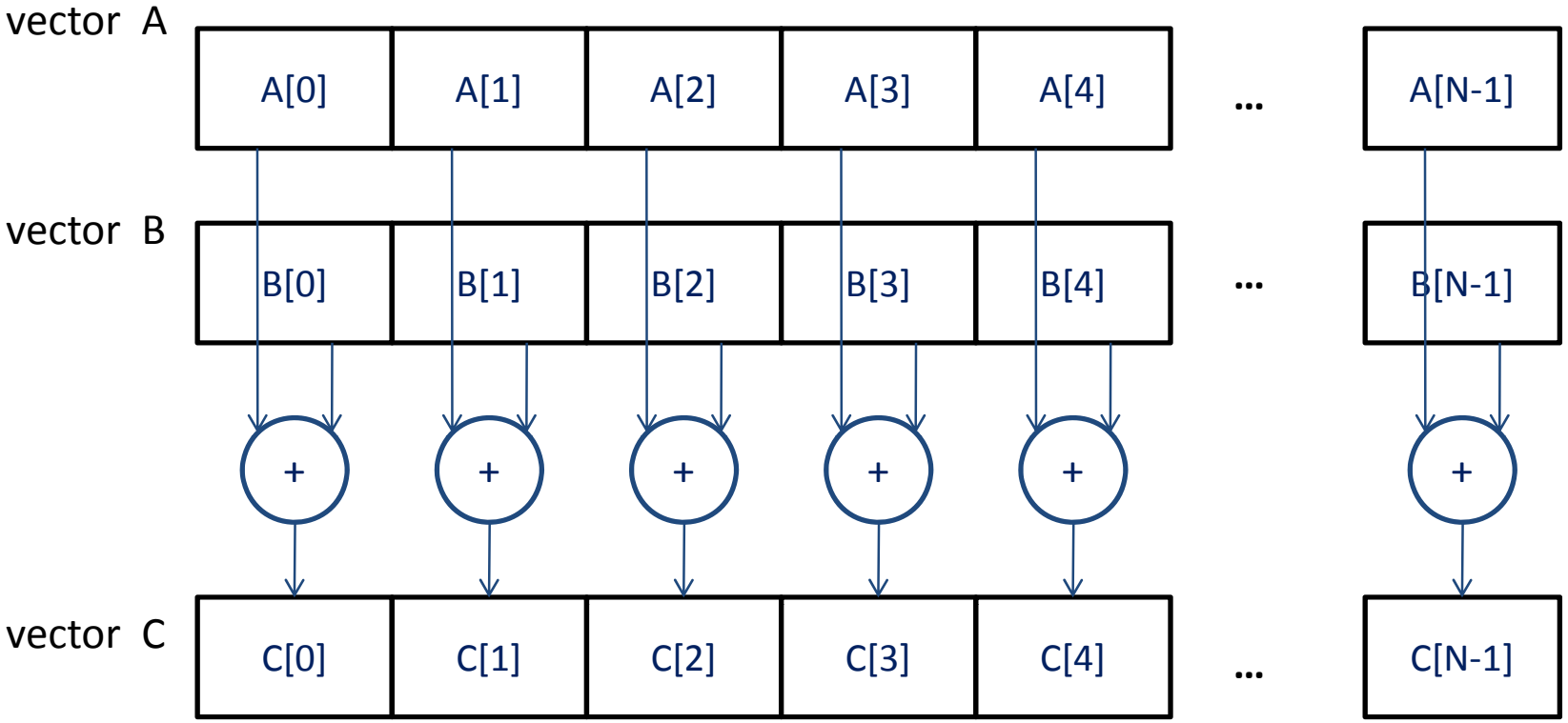
- *nvcc* para compilar los ficheros fuente .cu (kernels)
- *g++/gcc* para compilar los ficheros fuente .cpp/.c y enlazar

Ejemplos a desarrollar

- Ejemplo 1: suma vectores
- Ejemplo 2: Multiplicación matrices sencilla
- Ejemplo 3: Multiplicación matrices usando mem. compartida
  - Optimización memoria
- Ejemplo 4: Multiplicación matrices mediante uso de librerías (cuBLAS)
- Ejemplo 5: Herramientas desarrollo
  - Profiler y depuración
- Ejemplo 6: Transposición matrices
  - Uso de **profiler**
- *Ejemplo 7: Tratamientos de imágenes*

Ejemplo 1

- Suma de matrices
  - ... basándose en la idea de suma de vectores



Ejemplo 1

vectorAdd.c

```
// Compute vector sum C = A+B
void vecAdd(float* A, float* B, float* C,
   int n)
{
   for (i = 0, i < n, i++)
      C[i] = A[i] + B[i];
}

int main()
{
   // Memory allocation for A_h, B_h, C_h
   // I/O to read A_h and B_h, N elements
   ...
   vecAdd(A_h, B_h, C_h, N);
}
```

Ejemplo 1

vectorAdd.cu

```
// Compute vector sum C = A+B

int main()
{
   // Get device memory for A, B, C
   // copy A and B to device memory
   ...

   // Kernel execution in device
   // (vector add in device)
   ...

   // copy C from device memory
   // free A, B, C
   ...
}
```

Ejemplo2

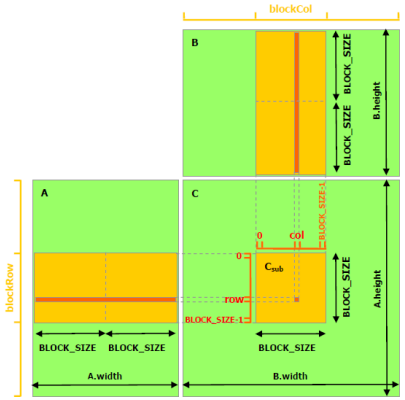
- $C_{NM} = A_{NK} * B_{KM}$

matrix_mult.c

```
for (i = 0; i < N; i++) {
   for (j = 0; j < M; j++) {
      for (k = 0; k < K; k++) {
         C[i][j] += A[i][k]*B[k][j];
      }
   }
}
```

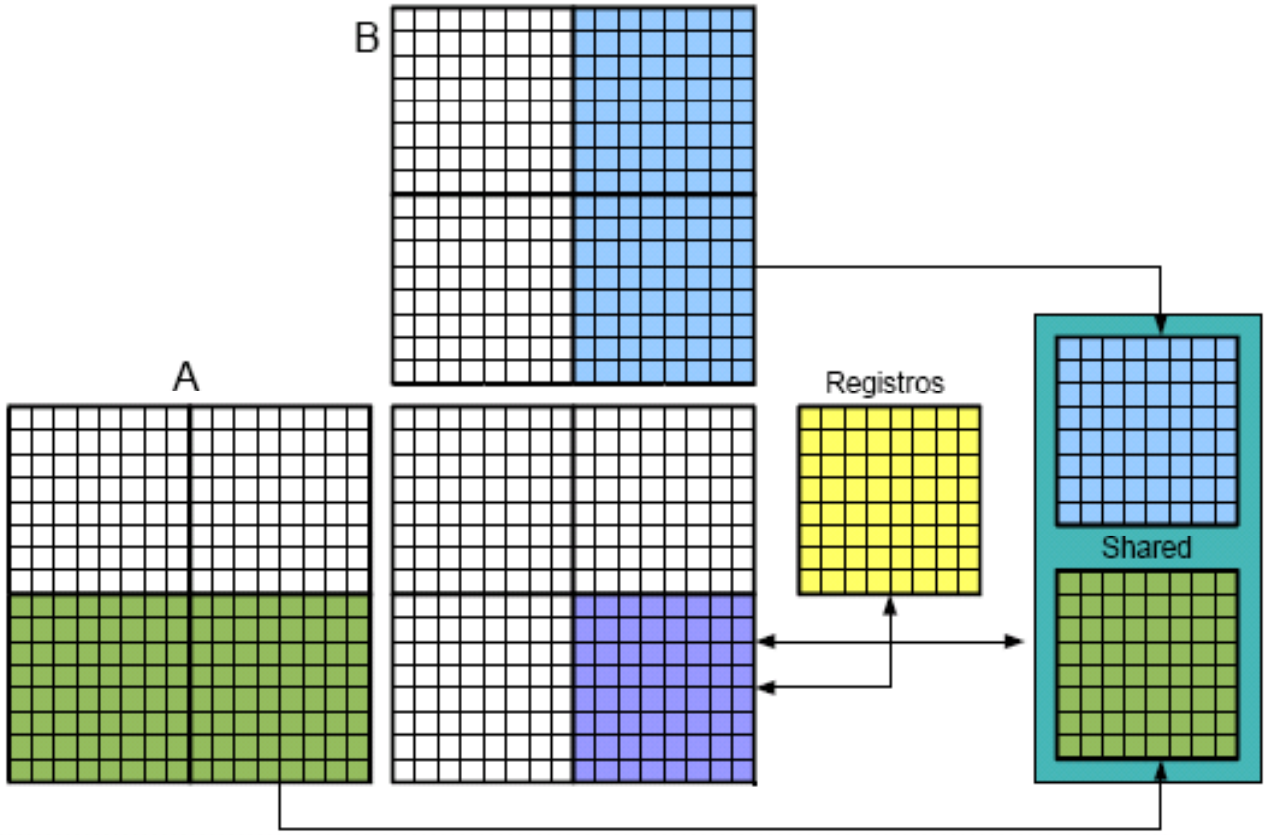
Ejemplo 2

- $C_{NM} = A_{NK} * B_{KM}$
- Muchos threads
  - Un Thread por cada Elemento de C[i][j]
  - Cada Thread necesita Fila A[i][:] y Columna B[:][j]
- Podemos Usar Memoria Compartida
  - Cada Block Recorre Filas A[-][:] y Columnas B[:][-]
  - Hay Localidad a nivel de CUDA Block!!



Ejemplo 3

- $C_{NM} = A_{NK} * B_{KM}$



Ejemplo 3

matrix_mult.cu

```
int main(int argc, char** argv) {
   float* d_A, *d_B, *d_C;
   float* h_A, *h_B, *h_C;
   dim3 block(ABLOQUE, ABLOQUE);
   dim3 grid(2,2);
   cudaMalloc((void**)&d_A,n*k);
   cudaMalloc((void**)&d_B, k*m);
   cudaMalloc((void**)&d_C, n*m);
   h_A = (float*)malloc(n*k);
   ...

   // Inicialización A y B
   cudaMemcpy(d_A,h_A,n*k,cudaMemcpyHostToDevice);
   cudaMemcpy(d_B,h_A,k*m,cudaMemcpyHostToDevice);
   mulMat<<<grid,block>>>(d_C,d_A,d_B);
   cudaMemcpy(h_C,d_C,n*m,cudaMemcpyDeviceToHost);
   cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
   free(h_A); free(h_B); free(h_C);
}
```

Ejemplo 3

matrix_mult.cu

```
__global__ void mulMat(float* out, float* mA, float* mB){
   int bx = blockIdx.x; int by = blockIdx.y;
   int tx = threadIdx.x; int ty = threadIdx.y;
   int aIni = AMATRIZ*ABLOQUE*by; int aFin = aIni+AMATRIZ-1;
   int aSig = ABLOQUE;
   int bIni = ABLOQUE*bx; int bSig = ABLOQUE*AMATRIZ;
   __shared__ float sA[ABLOQUE][ABLOQUE];
   __shared__ float sB[ABLOQUE][ABLOQUE];
   float valor = 0;
   for (int a=aIni,b=bIni;a<=aFin;a+=aSig,b+=bSig){
      sA[ty][tx] = mA[a + AMATRIZ * ty + tx];
      sB[ty][tx] = mB[b + AMATRIZ * ty + tx];
      __syncthreads();
      for (int i = 0; i < ABLOQUE; i++)
         valor += sA[ty][i] * sB[i][tx];
      __syncthreads();
   }
   int bOut = AMATRIZ * ABLOQUE * by + ABLOQUE * bx;
   out[bOut + AMATRIZ * ty + tx] = valor;
}
```

Ejemplo 4

- Utilización de librerias optimizadas
- Haremos uso de la librerías CUBLAS
  - BLAS (Basic Linear Algebra Subroutine)
  - CUBLAS = CUDA + BLAS
  - Operación GEMM:
    - C = α op (A) op (B) + β C
    - α y β son escalares
    - $A_{m\times k}$, $B_{k\times n}$ y $C_{n\times n}$ en (column-major)

cublasSgemm.c

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
    cublasOperation_t transa, cublasOperation_t transb,
    int m, int n, int k,
    const float        *alpha,
    const float        *A, int lda,
    const float        *B, int ldb,
    const float        *beta,
    float              *C, int ldc)
```

Ejemplo 4

- mulMat ⇌ cublasSgemm

cublasSgemm.c

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
    cublasOperation_t transa, cublasOperation_t transb,
    int m, int n, int k,
    const float        *alpha,
    const float        *A, int lda,
    const float        *B, int ldb,
    const float        *beta,
    float              *C, int ldc)
```

Ejemplo 4

- mulMat ⇌ cublasSgemm

matrixMult.c

```
int main(int argc, char** argv) {
   float* d_A, *d_B, *d_C;
   float* h_A, *h_B, *h_C;
   dim3 block(ABLOQUE, ABLOQUE);
   dim3 grid(2,2);
   cudaMalloc((void**)&d_A,n*k);
   cudaMalloc((void**)&d_B, k*m);
   cudaMalloc((void**)&d_C, n*m);
   h_A = (float*)malloc(n*k);
   ...
   // Inicialización A y B
   cudaMemcpy(d_A,h_A,n*k,cudaMemcpyHostToDevice);
   cudaMemcpy(d_B,h_A,k*m,cudaMemcpyHostToDevice);
   mulMat<<<grid,block>>>(d_C,d_A,d_B);
   cudaMemcpy(h_C,d_C,n*m,cudaMemcpyDeviceToHost);
   cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
   free(h_A); free(h_B); free(h_C);
}
```

Ejemplo 4

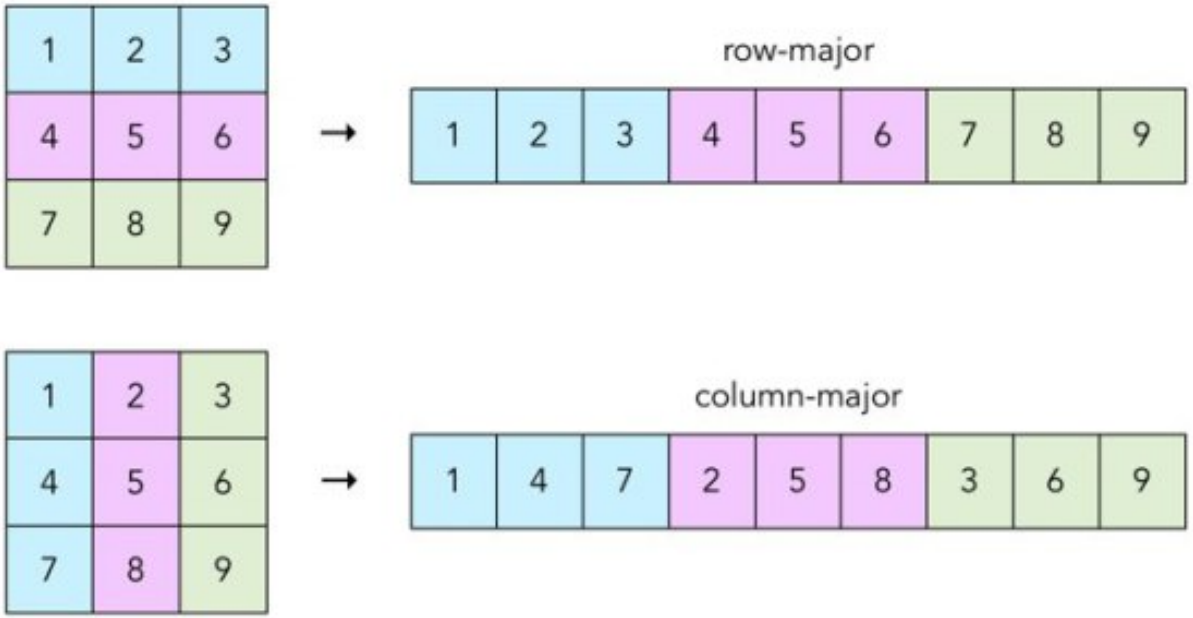
- mulMat ⇌ cublasSgemm

matrixMult.c

```
int main(int argc, char** argv) {
   float* d_A, *d_B, *d_C;
   float* h_A, *h_B, *h_C;
   dim3 block(ABLOQUE, ABLOQUE);
   dim3 grid(2,2);
   cudaMalloc((void**)&d_A,n*k);
   cudaMalloc((void**)&d_B, k*m);
   cudaMalloc((void**)&d_C, n*m);
   h_A = (float*)malloc(n*k);
   ...
   // Inicialización A y B
   cudaMemcpy(d_A,h_A,n*k,cudaMemcpyHostToDevice);
   cudaMemcpy(d_B,h_A,k*m,cudaMemcpyHostToDevice);
   cublasSgemm(.....);
   cudaMemcpy(h_C,d_C,n*m,cudaMemcpyDeviceToHost);
   cudaFree(d_A); cudaFree(d_B); cudaFree(d_C);
   free(h_A); free(h_B); free(h_C);
}
```

Ejemplo 4

- column-major vs. row-major
- $A * B = (B^T * A^T)^T$



Ejemplo 2, 3 y 4

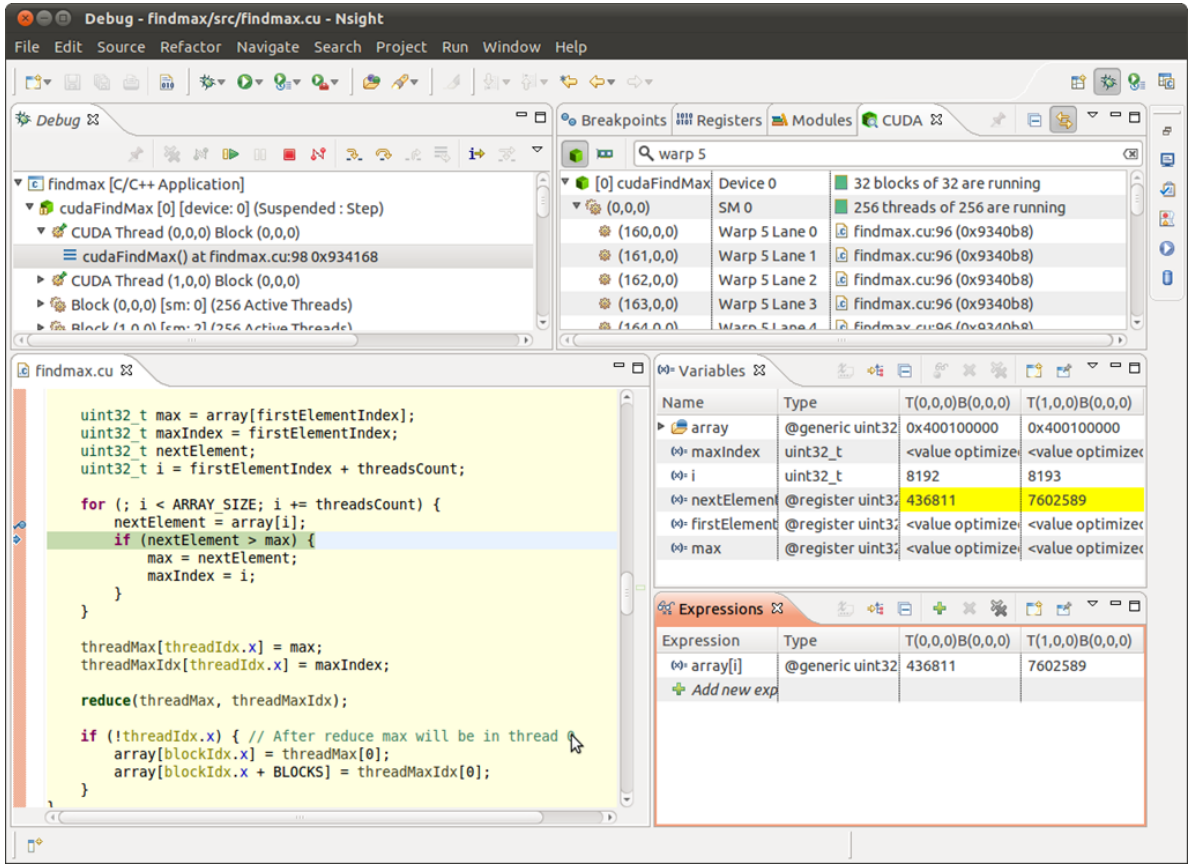
- Comparativa de tiempos de ejecución (GPU-CPU) para la multiplicación de matrices
  - A[4096][1024]*B[1024][2048]
  - A[1024][1024] *B[1024][1024]
  - A[8192][512] *B[512][4096]

Ejemplo 5

- Herrammientas de desarrollo en CUDA
- NVIDIA Nsight Visual Studio Edition | NVIDIA Developer
  - Eclipse: *nsight*
  - Depurador: *cuda-gdb*
  - Profiling: *nvprof* y *nvvp*

Ejemplo 5

- *nsight*



Ejemplo 5

- *cuda-gdb*
- `http://docs.nvidia.com/cuda/cuda-gdb`
- Compilación: *nvcc -g -G fuente.cu -o exec*

Ejemplo 5

- *cuda-gdb*
  - Información del dispositivo, bloque y thread

Terminal #1

```
carlos@posets:~/cuda-gdb ./exec
(cuda-gdb) cuda device sm warp lane block thread
block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0

(cuda-gdb) cuda kernel block thread
kernel 1, block (0,0,0), thread (0,0,0)

(cuda-gdb) cuda kernel
kernel 1
```

- Cambio de bloque y thread

Terminal #1

```
cuda-gdb) cuda block 1 thread 3
[Switching focus to CUDA kernel 1, grid 2, block (1,0,0), thread (3,0,0), device 0, sm 3, warp 0,
```

Ejemplo 5

- *cuda-gdb*
  - Compilar con las opciones de depuración e invocar el depurador (cuda-gdb)
  - Excepción de la línea 17: *CUDA_EXCEPTION_10*
    - Identificar que significa esa excepción `http://docs.nvidia.com/cuda/cuda-gdb`

Terminal #1

```
(cuda-gdb) run
Starting program: /…/example
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7ffff5d0e700 (LWP 22874)]

CUDA Exception: Device Illegal Address
The exception was triggered in device 0.

Program received signal CUDA_EXCEPTION_10, Device Illegal Address.
[Switching focus to CUDA kernel 0, grid 1, block (3,0,0), thread (32,0,0), device 0, sm 7, warp 1,
0x0000000000811a98 in example(int * @global * @global)<<<(8,1,1),(64,1,1)>>> (data=0x902f20000) at
17      *(data[idx1]) = value3;
```

Ejemplo 5

- *cuda-gdb*
  - **gdb cheatsheet** en `http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf`

Ejercicio

- Introducir *breakpoint* en línea anterior (16)
- Cambiar contexto (*block* y *thread*)
- NOTA: el error se muestra para un *warp* en cuestión no para el hilo correspondiente
- ¿Qué *thread* ha sido el causante del error?
- ¿Por qué?

Ejemplo 5

- *profiler*
  - Ver ejemplo 6

Ejemplo 6

- Transposición de Matrices
  - Empleada en multitud de algoritmos: FFT
  - PROBLEMA: Accesos a memoria no eficientes en GPU

Versión CPU vs GPU

- Makefile: *gcc -o exec source.c -fopenmp*

Ejemplo 6

- Paralelismo a nivel de thread: OpenMP

Cuadro: Rendimiento en CPU

CPU 2D	CPU 1D	
380MB/s	1 Core	365 MB/s
	2 Core	696 MB/s
	4 Core	859 MB/s

Ejemplo 6

- Versión 1: esquema 1D

transpose.c

```
void transpose1D(float *in,
    float *out, int n)
{
    int i, j;

    for(j=0; j < n; j++)
       for(i=0; i < n; i++)
          out[j*n+i] = in[i*n+j];
}
```

Ejemplo 6

- Versión 1: esquema 1D

transpose.cu

```
__global__ void transpose_device(float *in, float *out, int rows, int
     cols)
{
   int i, j;
   i = blockIdx.x * blockDim.x + threadIdx.x;

   if (i<rows)
   for ( j=0; j<cols; j++)
      out [ i * rows + j ] = in [ j * cols + i ];
}
```

Ejemplo 6

- Versión 2: esquema 2D

transpose2D.cu

```
__global__ void transpose_device(float *in, float *out, int rows, int
     cols)
{
   int i, j;
   i = ??
   j = ??

   if (i<?? && j<??)
      out [ i * rows + j ] = in [ j * cols + i ];
}
```

Ejemplo 6

- Versión 2: esquema 2D

Cuadro: Rendimiento en GPU

CPU 2D	CPU 1D+OMP	CUDA 1D	CUDA 2D
380MB/s	859 MB/s	1219 MB/s	2014 MB/s

Ejemplo 6

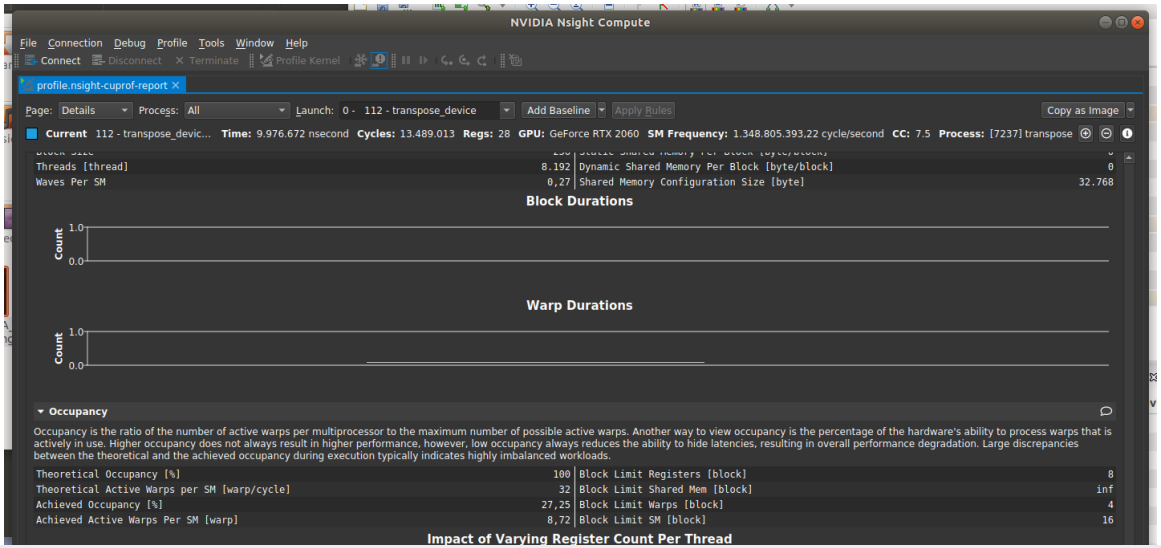
- CUDA profiler: **nvprof**

Terminal #1

```
carlos@7picos: $ nvprof ./transpose
./exec n (by default n=4096)
Transpose version 1D: 381.636146 MB/s
==29936== NVPROF is profiling process 29936, command: ./transpose
Transpose kernel version: 4950.878007 MB/s
==29936== Profiling application: ./transpose
==29936== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 47.10%  10.468ms         1  10.468ms  10.468ms  10.468ms  [CUDA memcpy DtoH]
 42.61%  9.4681ms         1  9.4681ms  9.4681ms  9.4681ms  [CUDA memcpy HtoD]
 10.29%  2.2867ms         1  2.2867ms  2.2867ms  2.2867ms  transpose_device(float*, float*, int, i

==29936== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 85.07%  130.76ms         2  65.381ms  360.05us  130.40ms  cudaMalloc
 14.57%  22.390ms         2  11.195ms  9.5133ms  12.877ms  cudaMemcpy
  0.24%  370.00us        91  4.0650us     122ns  193.84us  cuDeviceGetAttribute
  0.07%  108.08us         1  108.08us  108.08us  108.08us  cuDeviceTotalMem
  0.02%  33.740us         1  33.740us  33.740us  33.740us  cudaLaunch
  0.02%  28.259us         1  28.259us  28.259us  28.259us  cuDeviceGetName
  0.00%  5.4260us         1  5.4260us  5.4260us  5.4260us  cudaThreadSynchronize
  0.00%  5.2270us         4  1.3060us     147ns  4.3870us  cudaSetupArgument
  0.00%  1.9540us         1  1.9540us  1.9540us  1.9540us  cudaConfigureCall
  0.00%  1.5640us         3     521ns     124ns  1.2540us  cuDeviceGetCount
  0.00%     853ns         3     284ns     122ns     487ns  cuDeviceGet
```

Ejemplo 6

- CUDA profiler: **nvvp**
  - Aplicación gráfica con mayor precisión
- Para mayor detalle consultar la opción **Profile in NVIDIA Nsight Compute**
- Versión v1 de la transposición de matrices: *Achieved Occupancy 27,25 %*

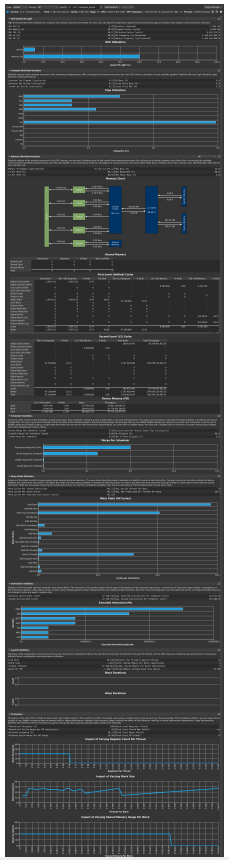


Ejemplo 6

- CUDA profiler: **nvvp**
- **Profile in NVIDIA Nsight Compute**: versión v2 de la transposición de matrices: *Occupancy 59.31 %*

Ejemplo 6

- CUDA profiler: **nvvp**
- **Profile in NVIDIA Nsight Compute**: versión v2



Ejemplo 6

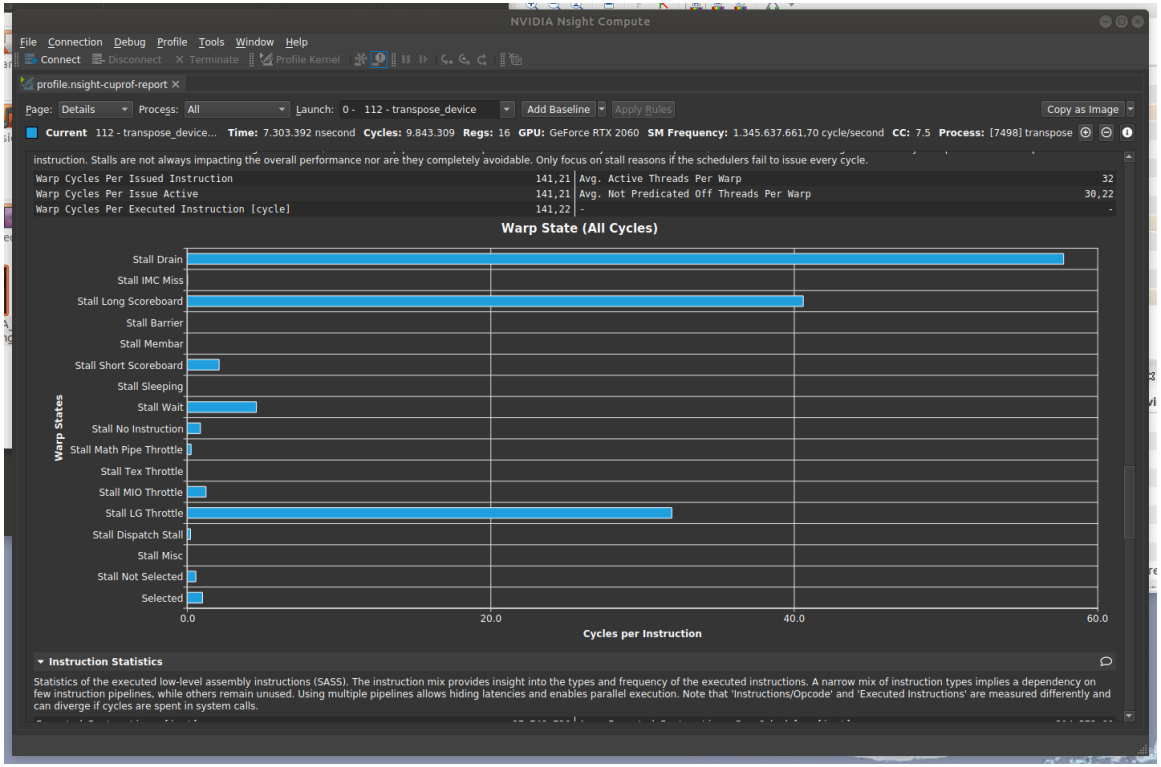
- CUDA profiler: **nvvp**
- Sin embargo el **SOL SM [%] es del 5,70**, quiere decir que el Speed of Light (SOL) solamente es el 5,7 % del máximo
- **¿A qué es debido?**
  - En el apartado *Warp State (All cycles) da una idea de la razón*
  - Consultar la tabla *Table 6. Warp Scheduler States* del manual Nsight `https://docs.nvidia.com/nsight-compute/NsightCompute/index.html#profiler-report-summary-page`

Ejemplo 6

¿A qué es debido?

- En el apartado *Warp State (All cycles) da una idea de la razón*
- Consultar la tabla *Table 6. Warp Scheduler States* del manual Nsight `https://docs.nvidia.com/nsight-compute/NsightCompute/index.html#profiler-report-summary-page`
  - *Stall Drain*: Warp was stalled after EXIT waiting for all memory instructions to complete
  - *Stall long Scoreboard*: Warp was stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, tex) operation. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture

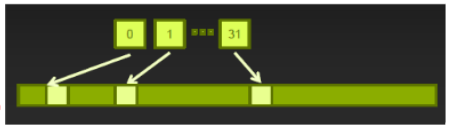
Ejemplo 6

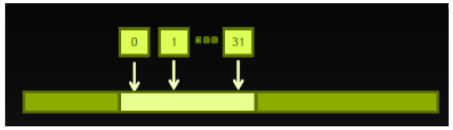

NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel
profile.nsight-cuprof-report
Page: Details Process: All Launch: 0 - 112 - transpose_device Add Baseline Apply Rules Copy as Image
Current 112 - transpose_device... Time: 7.303.392 nsecond Cycles: 9.843.309 Regs: 16 GPU: GeForce RTX 2060 SM Frequency: 1.345.637.661,70 cycle/second CC: 7.5 Process: [7498] transpose
instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle.
Warp Cycles Per Issued Instruction 141,21
Warp Cycles Per Issue Active 141,21
Warp Cycles Per Executed Instruction [cycle] 141,22
Avg. Active Threads Per Warp 32
Avg. Not Predicated Off Threads Per Warp 30,22
Warp State (All Cycles)
Warp States
Stall Drain
Stall IMC Miss
Stall Long Scoreboard
Stall Barrier
Stall Membar
Stall Short Scoreboard
Stall Sleeping
Stall Wait
Stall No Instruction
Stall Math Pipe Throttle
Stall Tex Throttle
Stall MIO Throttle
Stall LG Throttle
Stall Dispatch Stall
Stall Misc
Stall Not Selected
Selected
0.0
20.0
40.0
60.0
Cycles per Instruction
Instruction Statistics
Statistics of the executed low-level assembly instructions (SASS). The instruction mix provides insight into the types and frequency of the executed instructions. A narrow mix of instruction types implies a dependency on few instruction pipelines, while others remain unused. Using multiple pipelines allows hiding latencies and enables parallel execution. Note that 'Instructions/Opcode' and 'Executed Instructions' are measured differently and can diverge if cycles are spent in system calls.

Ejemplo 6

- Versión v2 de la transposición de matrices: **Accesos alineados/desalineados**

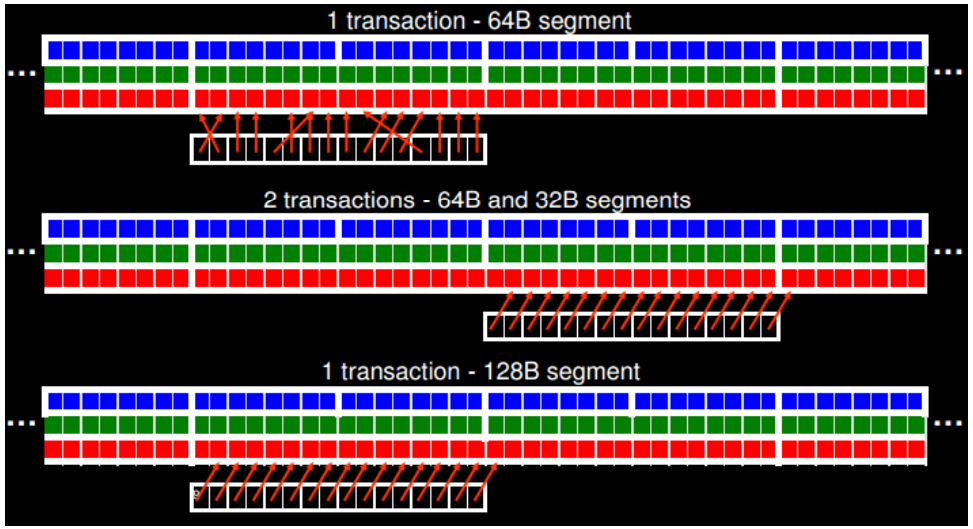
```
__global__ void transpose_device(float *in,
        float *out, int rows, int cols)
{
    int i, j;

   i = blockIdx.x * blockDim.x + threadIdx.x;
   j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i<rows && j<cols)
      out [ i * rows + j ] = in [ j * cols + i ];
}
```





Ejemplo 6

- Versión v2 de la transposición de matrices: **Accesos alineados/desalineados**



Ejemplo 6

- Versión v3 de la transposición de matrices: **shared-memory**
  - Más rápida en comparación con la memoria global
  - Baja latencia
  - Alto Ancho de banda
- Load: Copia de un bloque a Global ⇒ memoria compartida
- Store: Copia de un memoria compartida ⇒ mem Global

Ejemplo 6

- Versión v3 de la transposición de matrices: **shared-memory**

transpose_v3.cu

```
#define TILE_DIM 16

__global__ void transpose_device(float *in, float *out, int rows, int
     cols)
{
   int i, j;
   __shared__ float tile [ TILE_DIM ] [ TILE_DIM ];

   i = blockIdx.x * blockDim.x + threadIdx.x;
   j = blockIdx.y * blockDim.y + threadIdx.y;

   if (i<rows && j<cols) {
      tile[?] [?]...
      __syncthreads();
      i = ? ;
      j = ? ;
      out[ i * rows + j ] ...
   }
}
```

Ejemplo 6

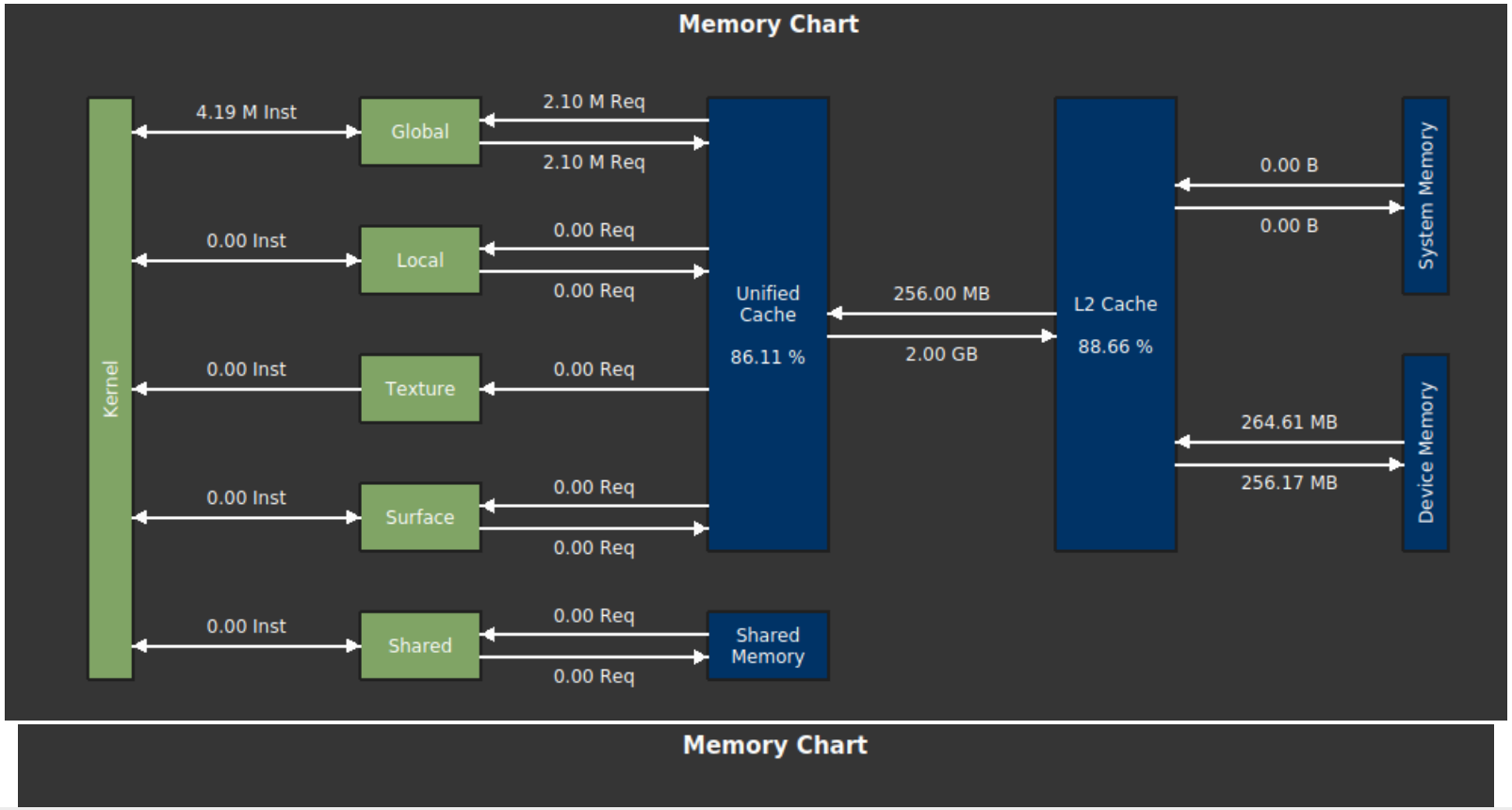
- Versión v3 de la transposición de matrices: **shared-memory**

Cuadro: Rendimiento en GPU

CPU 1D+OMP	CUDA 1D	CUDA 2D	CUDA 2D+Shared
444 MB/s	4471 MB/s	4946 MB/s	5213 MB/s

Ejemplo 6

- Versión v2 vs v3 de la transposición de matrices: **shared-memory**



Ejemplo 6

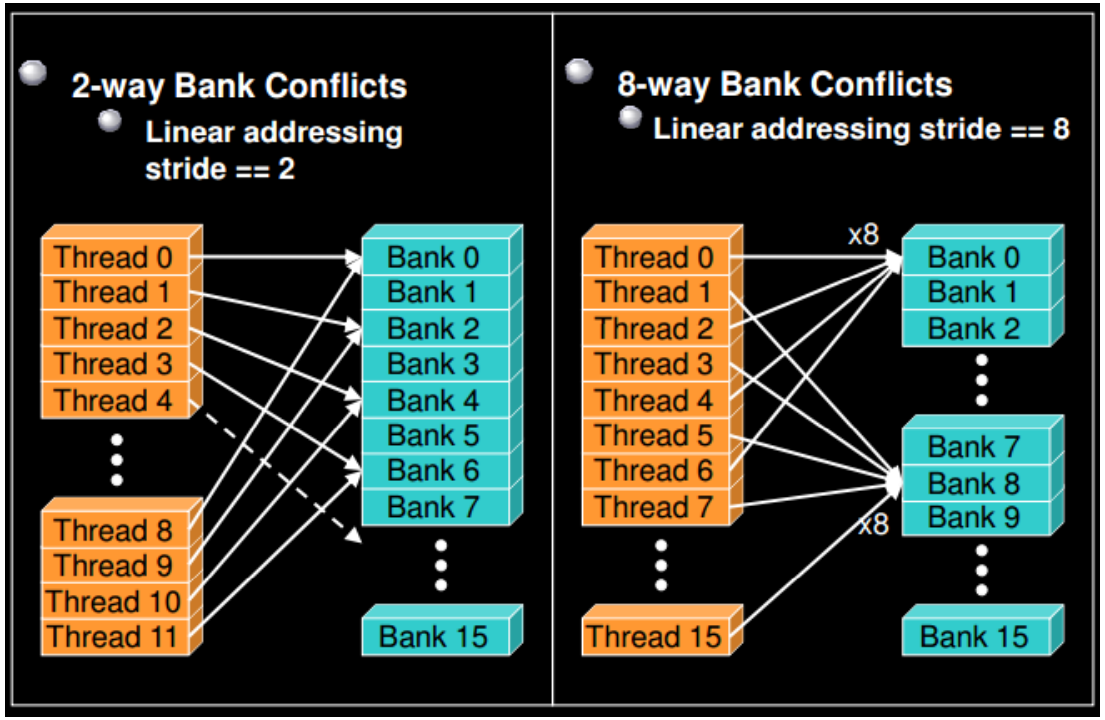
- Versión v3 de la transposición de matrices: **shared-memory**
  - Consultar **Shared Memory** con *Bank Conflicts*

Shared Memory

	Instructions	Requests	% Peak	Bank Conflicts
Shared Load	2.097.152	2.097.152	1,55	0
Shared Store	2.097.152	16.777.216	12,43	14.680.064
Shared Atomic	0	-	-	-
Total	4.194.304	18.874.368	13,99	14.680.064

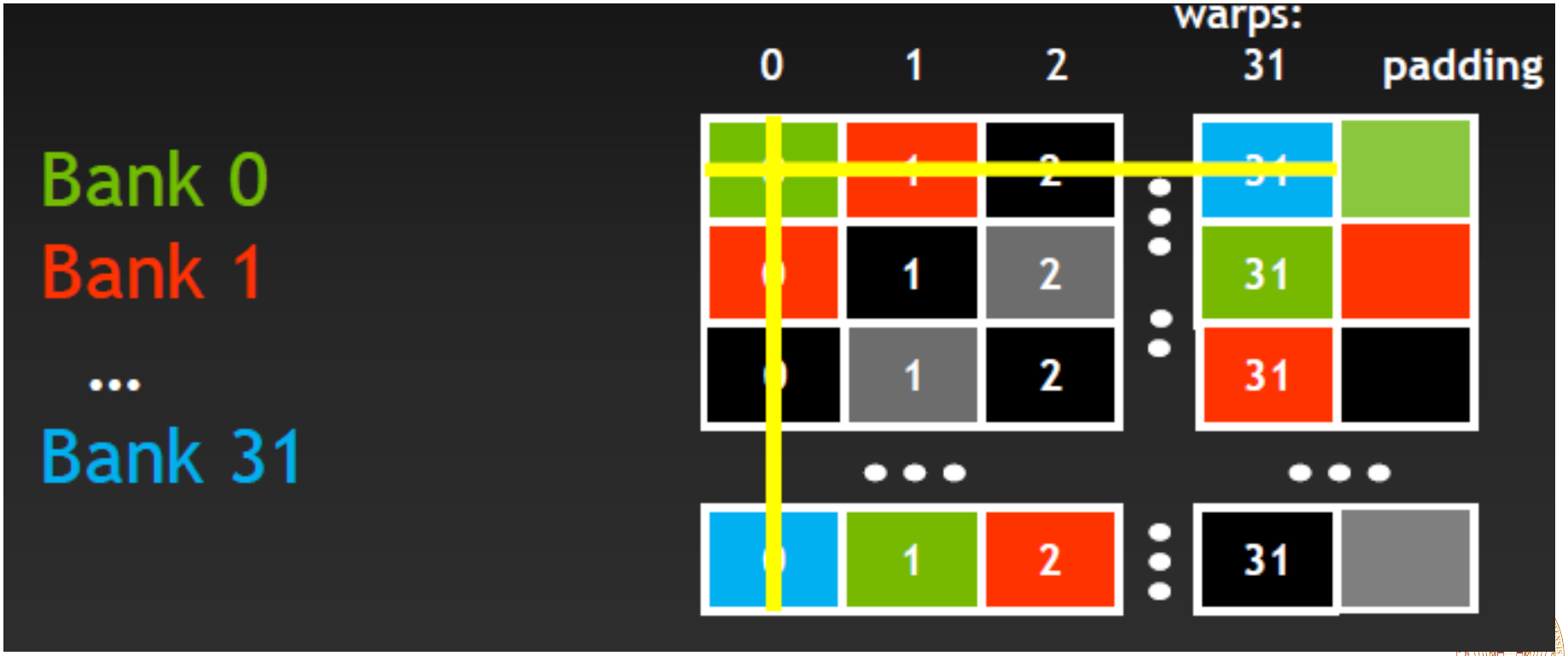
Ejemplo 6

- Versión v3 de la transposición de matrices: **shared-memory**
  - Conflictos de banco



Ejemplo 6

- Versión v3 de la transposición de matrices: **shared-memory**
  - Conflictos de banco



Ejemplo 6

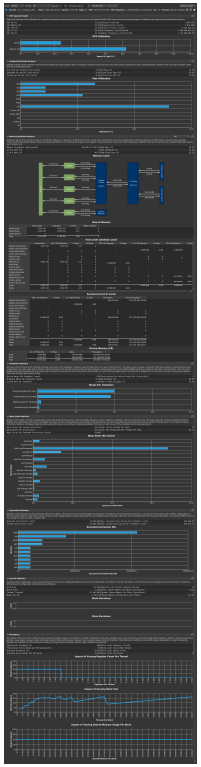
- Versión v4 de la transposición de matrices: **sin conflicto en shared**

transpose_v4.cu

```
#define TILE_DIM 16

__global__ void transpose_device(float *in, float *out, int rows, int
     cols)
{
   int i, j;
   __shared__ float tile [ TILE_DIM ] [ TILE_DIM+1 ];

   i = blockIdx.x * blockDim.x + threadIdx.x;
   j = blockIdx.y * blockDim.y + threadIdx.y;

   if (i<rows && j<cols) {
      tile[?] [?]...
      __syncthreads();
      i = ? ;
      j = ? ;
      out[ i * rows + j ] ...
   }
}
```

Ejemplo 6

- Versión v4 de la transposición de matrices: **sin conflicto en shared**



Ejemplo 7

- Tratamiento de imágenes: **Lane assist**
  - Detección de bordes: **Filtrado canny**
  - Detección de líneas: **Transformada de Hough**

Ejecutable

- Sintasis:
- **./image input_image.png [c,g]**
  - c: código ejecutado en CPU
  - g: código ejecutado en GPU

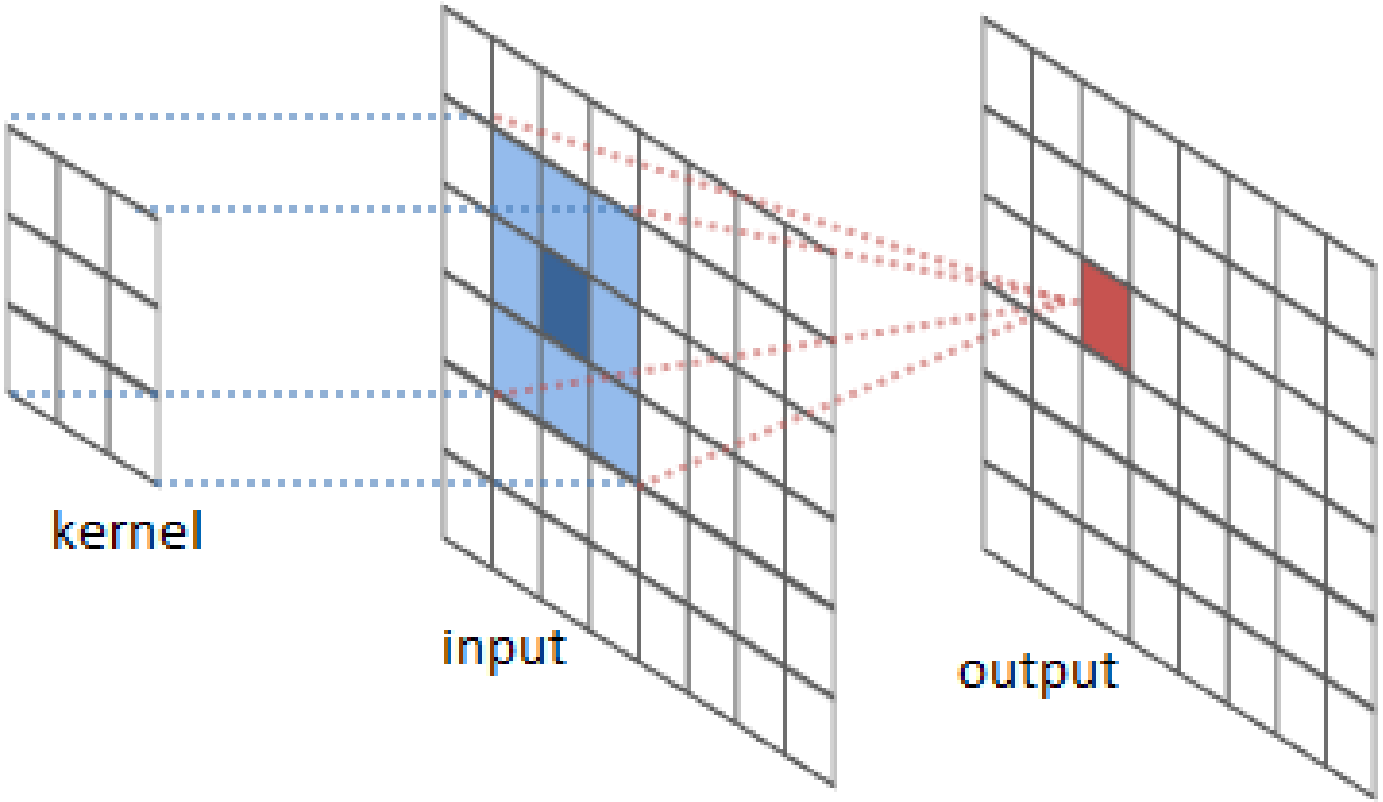
Ejemplo 7

Ejemplo 7

- Detección de bordes: **Filtrado canny**
  - Basado en **convolución 2D**

Ejemplo 7

- Detección de bordes: **Filtrado canny**
  - Basado en **convolución 2D**



Ejemplo 7

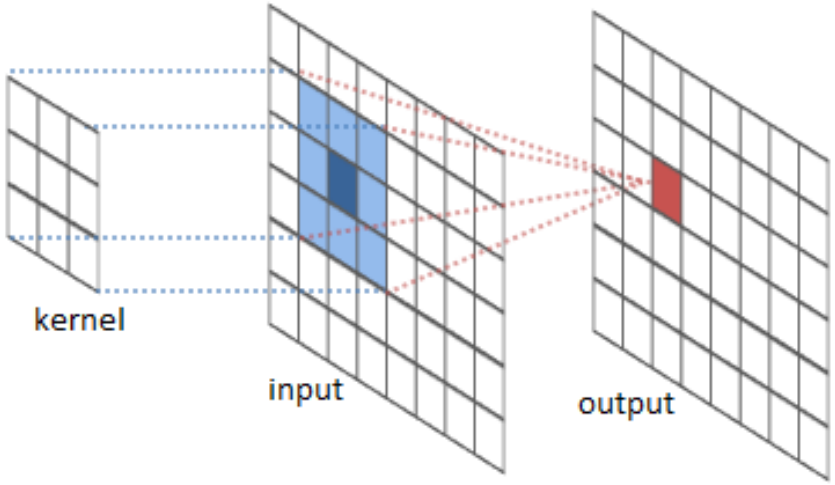
Etapas

1. Reducción de ruido (filtro *blur*)
2. Gradiente de la imagen
3. Supresión no-máximo

Ejemplo 7

1. Reducción de ruido (convolución 2D)

$$Out(i,j) = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} In(i-2:i+2, j-2:j+2)$$



Ejemplo 7

2. Gradiente de la imagen

- $Gx(i,j) = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} NR(i-2:i+2, j-2:j+2)$
- $G = \sqrt{{G_x}^2 + {G_y}^2}$
- $\phi = \arctan 2({G_x}^2, {G_y}^2)$

Ejemplo 7

3. Supresión no-máximo
   - Determina si la magnitud del gradiente (G) es máximo local (redondeo $\phi \in 0°, 45°, 90°, 135°$)
   - Umbral de histéresis: dos umbrales alto y bajo
     - Si el gradiente G>umbral_alto ⇒ **borde**
     - Si el gradiente G>umbral_bajo y **es máximo local** ⇒ **borde**

Máximo local

- Si $\phi = 0°$ (borde N o S) si $G > G_E$ y $G > G_W$
- Si $\phi = 90°$ (borde E o W) si $G > G_N$ y $G > G_S$
- Si $\phi = 135°$ (borde NE o SW) si $G > G_{NW}$ y $G > G_{SE}$
- Si $\phi = 45°$ (borde NW o SE) si $G > G_{NE}$ y $G > G_{SW}$

Ejemplo 7

- La **transformada de Hough** bastante común en tratamiento de imágenes para detección de figuras geométricas
  - Más info: `https://es.wikipedia.org/wiki/Transformada_de_Hough`
- Después de aplicar un detector de bordes a una imagen (ej: operador Sobel, Canny)

Ejemplo 7

- **Transformada de Hough**
- Útil en muchos algoritmo de detección de líneas en una imagen
  - Emplea un sitema votador

Ejemplo 7

- Una recta se puede representar: $y = m * x + n$
  - Donde (x, y) corresponde a cada píxel de la imagen: **problema con líneas verticales**: $m = \infty$
- También se puede representar en coordenadas polares: $y = -\frac{\cos\theta}{\sin\theta} * x + \frac{\rho}{\sin\theta}$
- O finalmente como $\rho = x * \cos\theta + y * \sin\theta$ asociando a cada recta el par (ρ, θ)

Ejemplo 7

- Implementación
  - El algoritmo de Hough almacena el resultado una **matriz acumulador** (votación) para cada píxel:

pseudocode

```
[1] cargar imagen
[2] detectar los bordes en la imagen
[3] por cada punto en la imagen:
[4]   si el punto (x,y) esta en un borde:
[5]      por todos los posibles ángulos theta:
[6]         calcular rho para el punto (x,y) con un ángulo theta
[7]         incrementar la posición (rho, theta) en el acumulador
[8] buscar las posiciones con los mayores valores en el acumulador
[9] devolver las rectas cuyos valores son los mayores en el acumulador.
```

Ejemplo 7

- Esqueleto del código: *main.c* y *kernel.h*
- Modificación del código fuente *kernel.cu*
- Imágenes de prueba en directorio **images**

kernel.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda.h>

#include "routinesGPU.h"

void line_asist_GPU(uint8_t *im, int height, int width,
    uint8_t *imEdge, float *NR, float *G, float *phi, float *Gx, float *
         Gy, uint8_t *pedge,
    float *sin_table, float *cos_table,
    uint32_t *accum, int accu_height, int accu_width,
    int *x1, int *y1, int *x2, int *y2, int *nlines)
{

    /* To do */
}
```