Contenidos

Contenidos

3.2. Depuración y medición de uso de recursos
Medición recursos al compilar

- Compilar con opción "`--ptxas-options -v`"

- Para saber qué recursos va a usar un kernel:
  - Memoria global → de toda la grid
  - Memoria constante → de toda la grid
  - Memoria compartida → de cada bloque
  - Los registros → de cada thread

- **ATENCION. Obviamente:**
  - **Memoria compartida**
    - **No** informa sobre la **dinámica** (`extern _shared_`),
    - **Sí** informa sobre la **estática** (`_shared_`).

  - **Memoria global**
    - **No** informa sobre la **dinámica** (reservada con `cudaMalloc`())
    - **Sí** informa sobre la **estática** (variables definidas como `__device__`)

3.2. Depuración y medición de uso de recursos
Medición recursos al compilar

```
////////////////////////////////////////////////////////////////////////////
  // Dummy kernel
////////////////////////////////////////////////////////////////////////////
__constant__ int constante_d[100*CM_SIZE];      //CM_SIZE=8
__device__ int basura[16];

__global__ void foo(int *gid_d)
{
  __shared__ int shared_mem[100];
  …
}

////////////////////////////////////////

$ export dbg=1
$ make
…

ptxas info    : 64 bytes gmem, 3200 bytes cmem[3]   // 64 bytes global
…                                                   // 3200 bytes constante

ptxas info    : Function properties for _Z3fooPi
ptxas info    : Used 13 registers, 400 bytes smem, …
                                                    // 13 registros
                                                    // 400 bytes shared
```

Contenidos

3.2. Depuración y medición de uso de recursos
Depuración de código

Método básico tradicional: mensajes espías

- **`printf`** desde el código del *kernel*
  - En `Makefile` indicar:
    - compatibilidad con arquitectura 20, pero no con 10

```
GENCODE_FLAGS := $(GENCODE_SM20)
```

  - ATENCIÓN: cada *thread* muestra su propio mensaje

3.2. Depuración y medición de uso de recursos
Depuración de código: cuda-gdb

- **cuda-gdb**

(https://developer.nvidia.com/cuda-gdb)

(https://docs.nvidia.com/cuda/cuda-gdb/index.html)

  - Opciones de compilación: `-g -G`
    - `-g`: generar información depuración código de la CPU
    - `-G`: generar información depuración código de la GPU

- **ATENCIÓN**: posteriormente, para prestaciones, quitar: `-g -G`

3.2. Depuración y medición de uso de recursos
Depuración de código: cuda-gdb

```
$ cuda-gdb ./vectorAdd
    ...
    Reading symbols from …

    (cuda-gdb) run
    Starting program:
    …

    Copy input data from the host memory to the CUDA device
    CUDA kernel launch with 2 blocks of 256 threads
    Copy output data from the CUDA device to the host memory

    Test PASSED
    …
```

3.2. Depuración y medición de uso de recursos
Depuración de código: cuda-gdb

`(cuda-gdb) break vectorAdd`

→ poniendo un *breakpoint* en el inicio del kernel vectorAdd

`(cuda-gdb) break 80`

→ poniendo un *breakpoint* en la línea 80 del código

`(cuda-gdb) run`

→ ejecutar desde el principio hasta el primer *breakpoint*

`(cuda-gdb) display threadIdx.x`

→ ver el valor de la variable `threadIdx.x` en ese *breakpoint*

`(cuda-gdb) next`

→ avanza ejecución (a nivel de *warp*)

`(cuda-gdb) continue`

→ continuar ejecución hasta siguiente *breakpoint* o el final

3.2. Depuración y medición de uso de recursos

Depuración de código: cuda-gdb

```
__global__ void
vectorAdd(const float *A, const float *B, float
*C, int numElements)


{
    int i = blockDim.x * blockIdx.x +
threadIdx.x;
        int bdx = blockDim.x;
        int bix = blockIdx.x;
        int tix = threadIdx.x;
    if (i < numElements)
        C[i] = A[i] + B[i];
}
```

```
$ cuda-gdb ./vectorAdd
(cuda-gdb) break vectorAdd
(cuda-gdb) run
(cuda-gdb) display bdx
1: bdx = 0
(cuda-gdb) display bix
2: bix = 0
(cuda-gdb) display tix
3: tix = 0
(cuda-gdb) next
42              int bdx =
blockDim.x;
3: tix = 0
2: bix = 0
1: bdx = 0
(cuda-gdb) next
43              int bix =
blockIdx.x;
3: tix = 0
2: bix = 0
1: bdx = 256
…
47          if (i < numElements)
3: tix = 32
2: bix = 0
```

Contenidos

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar

CUDA *Profiler* (VERSIONES ANTIGUAS)
Estadísticas de la ejecución de una aplicación CUDA

Versión intermedia: **nvprof**
(https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview)

Versión gráfica del CUDA profiler: **nvvp** (NVIDIA Visual Profiling)
https://developer.nvidia.com/nvidia-visual-profiler

Versión tarjetas más actuales: **Nsight Compute**
https://developer.nvidia.com/nsight-compute

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: CUDA Profiler

(VERSIONES ANTIGUAS)

```
$ vi /home/..../cuda_profile_config
     threadblocksize
     regperthread
     stasmemperblock
     dynsmemperblock

$ export CUDA_PROFILE=1

$ export CUDA_PROFILE_CONFIG=/home/..../cuda_profile_config

$ ./cuda_vectorReduce -n=1024 -bsx=64

$ cat cuda_profile_0.log
...
method=[ memcpyHtoD ] gputime=[ 1.312 ] cputime=[ 22.000 ]
method=[ memset32_post ] gputime=[ 2.560 ] cputime=[ 32.000 ] threadblocksize=[ 64, 1, 1 ]
    dynsmemperblock=[ 0 ] stasmemperblock=[ 0 ] regperthread=[ 6 ] occupancy=[ 0.042 ]
method=[ _Z12vectorReducePfS_i ] gputime=[ 7.648 ] cputime=[ 17.000 ] threadblocksize=[ 64, 1, 1
    ] dynsmemperblock=[ 256 ] stasmemperblock=[ 0 ] regperthread=[ 18 ] occupancy=[ 0.333 ]
method=[ memcpyDtoH ] gputime=[ 1.504 ] cputime=[ 24.000 ]
```

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: nvprof

```
$ nvprof ./cuda_vectorReduce -n=1024 -bsx=128
==29419== NVPROF is profiling process 29419, command: ./cuda_vectorReduce -n=1024 -bsx=128
==29419== Warning: Unified Memory Profiling is not supported on devices of compute capability less than 3.0
---> Running configuration: grid of 8 blocks of 128 threads (TOTAL: 1024 threads)
Processing time: 0.223296 (ms)
Test PASSED

==29419== Profiling application: ./cuda_vectorReduce -n=1024 -bsx=128
==29419== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 37.71%  3.0080us         1  3.0080us  3.0080us  3.0080us  [CUDA memset]
 31.40%  2.5050us         1  2.5050us  2.5050us  2.5050us  vectorReduce(float*, float*, int)
 18.85%  1.5040us         1  1.5040us  1.5040us  1.5040us  [CUDA memcpy DtoH]
 12.03%     960ns         1     960ns     960ns     960ns  [CUDA memcpy HtoD]

==29419== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 56.73%  83.330ms         2  41.665ms  413.16us  82.917ms  cudaMalloc
 42.55%  62.501ms         1  62.501ms  62.501ms  62.501ms  cudaDeviceReset
  0.24%  355.94us         2  177.97us  147.08us  208.86us  cudaFree
  0.16%  231.25us        91  2.5410us     289ns  85.403us  cuDeviceGetAttribute
```

NOTA: en este ejemplo hemos puesto el calculo de tiempo, usando eventos, en el codigo solamente para la ejecución del kernel, por lo que no queda reflejado el tiempo de comunicación en “Processing time”

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: nvprof

Muestra cronológicamente la actividad en la GPU y datos de uso de recursos en kernel lanzado (regs, SSMem, Dsmem): `--print-gpu-trace`

```
$ nvprof --print-gpu-trace ./cuda_vectorReduce -n=1000000 -bsx=128
```

```
==18535== NVPROF is profiling process 18535, command: ./cuda_vectorReduce -n=1000000 -bsx=128
==18535== Warning: Unified Memory Profiling is not supported on devices of compute capability less than 3.0
---> Running configuration: grid of 7813 blocks of 128 threads (TOTAL: 1000064 threads)
Processing time: 0.247648 (ms)
Test PASSED
==18535== Profiling application: ./cuda_vectorReduce -n=1000000 -bsx=128

==18535== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput           Device   Context    Stream  Name
208.58ms  1.2246ms                    -               -        -         -         -  3.8147MB  3.0420GB/s  GeForce GTX 480         1         7  [CUDA memcpy HtoD]
209.80ms  3.7120us                    -               -        -         -         -  30.520KB  7.8410GB/s  GeForce GTX 480         1         7  [CUDA memset]
209.81ms  236.98us           (7813 1 1)       (128 1 1)        8        0B      512B         -           -  GeForce GTX 480         1         7  vectorReduce(float*,
210.09ms  6.9440us                    -               -        -         -         -  30.520KB  4.1915GB/s  GeForce GTX 480         1         7  [CUDA memcpy DtoH]
```

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.

SSMem: Static shared memory allocated per CUDA block.

DSMem: Dynamic shared memory allocated per CUDA block.

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: nvprof

Muestra cronológicamente la actividad en la GPU y datos de uso de recursos en kernel lanzado (regs, SSMem, Dsmem): `--print-gpu-trace`

```
$ nvprof --print-gpu-trace ./cuda_vectorReduce -n=8000000 -bsx=128
```

```
==22291== NVPROF is profiling process 22291, command: ./cuda_vectorReduce -n=8000000 -bsx=128
==22291== Warning: Unified Memory Profiling is not supported on devices of compute capability less than 3.0
Running configuration: grid of 62500 blocks of 128 threads (8000000 threads)
Processing time: 6.844576 (ms)
Test PASSED
==22291== Profiling application: ./cuda_vectorReduce -n=8000000 -bsx=128
==22291== Profiling result:
```

Start	Duration	Grid Size	Block Size	Regs*	SSMem*	DSMem*	Size	Throughput	Device	Context	Stream	Name
209.32ms	15.079ms	-	-	-	-	-	30.518MB	1.9764GB/s	GeForce GTX 480	1	7	[CUDA memcpy HtoD]
224.40ms	9.4080us	-	-	-	-	-	244.14KB	24.748GB/s	GeForce GTX 480	1	7	[CUDA memset]
224.46ms	**6.7866ms**	**(62500 1 1)**	**(128 1 1)**	**13**	**0B**	**512B**	**-**	**-**	**GeForce GTX 480**	**1**	**7**	**vectorReduce(float*,**
231.29ms	44.033us	-	-	-	-	-	244.14KB	5.2876GB/s	GeForce GTX 480	1	7	[CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.

SSMem: Static shared memory allocated per CUDA block.

DSMem: Dynamic shared memory allocated per CUDA block.

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: nvprof

Muestra cronológicamente las llamadas CUDA runtime y API: `--print-api-trace`

```
$  nvprof --print-api-trace ./cuda_vectorReduce -n=1024 -bsx=128
```

```
…
   Start  Duration  Name
120.63ms  2.8480us  cuDeviceGetCount
120.63ms     273ns  cuDeviceGetCount
120.63ms     471ns  cuDeviceGet
120.64ms     708ns  cuDeviceGetAttribute
120.65ms     317ns  cuDeviceGet
…
121.09ms  84.290ms  cudaMalloc
205.39ms  137.08us  cudaMalloc
205.53ms  24.258us  cudaMemcpy
205.56ms  24.304us  cudaMemset
205.58ms  8.0190us  cudaEventCreateWithFlags
205.59ms  1.2180us  cudaEventCreateWithFlags
205.60ms  5.9480us  cudaEventRecord
205.64ms  1.9540us  cudaConfigureCall
205.64ms  2.1260us  cudaSetupArgument
205.64ms     336ns  cudaSetupArgument
205.65ms     581ns  cudaSetupArgument
205.65ms  15.578us  cudaLaunch (vectorReduce(float*, float*, int) [111])
205.66ms  6.9180us  cudaDeviceSynchronize
205.67ms  3.1010us  cudaEventRecord
205.68ms  3.9450us  cudaEventSynchronize
205.68ms  2.1460us  cudaEventElapsedTime
205.70ms  18.705us  cudaMemcpy
205.72ms  78.481us  cudaFree
205.80ms  54.854us  cudaFree
211.24ms  53.676ms  cudaDeviceReset
```

3.2. Depuración y medición de uso de recursos
Medición de recursos al ejecutar: nvprof + nvvp

nvvp:

- Por ejemplo, en Laboratorio 2.1:

```
$ nvvp -vm /usr/local/jdk1.8.0_202/bin/java
```

- Una linea de tiempo que muestra gráficamente las actividades de CPU y de la GPU

- Análisis automático de prestaciones que ayuda a optimizar las aplicaciones:
  - Tiempo de ejecución, uso de registros, memoria compartida,…
  - Desglose por:
    - Por stream
    - Por thread de CPU
    - Por GPU

nvprof + nvvp:

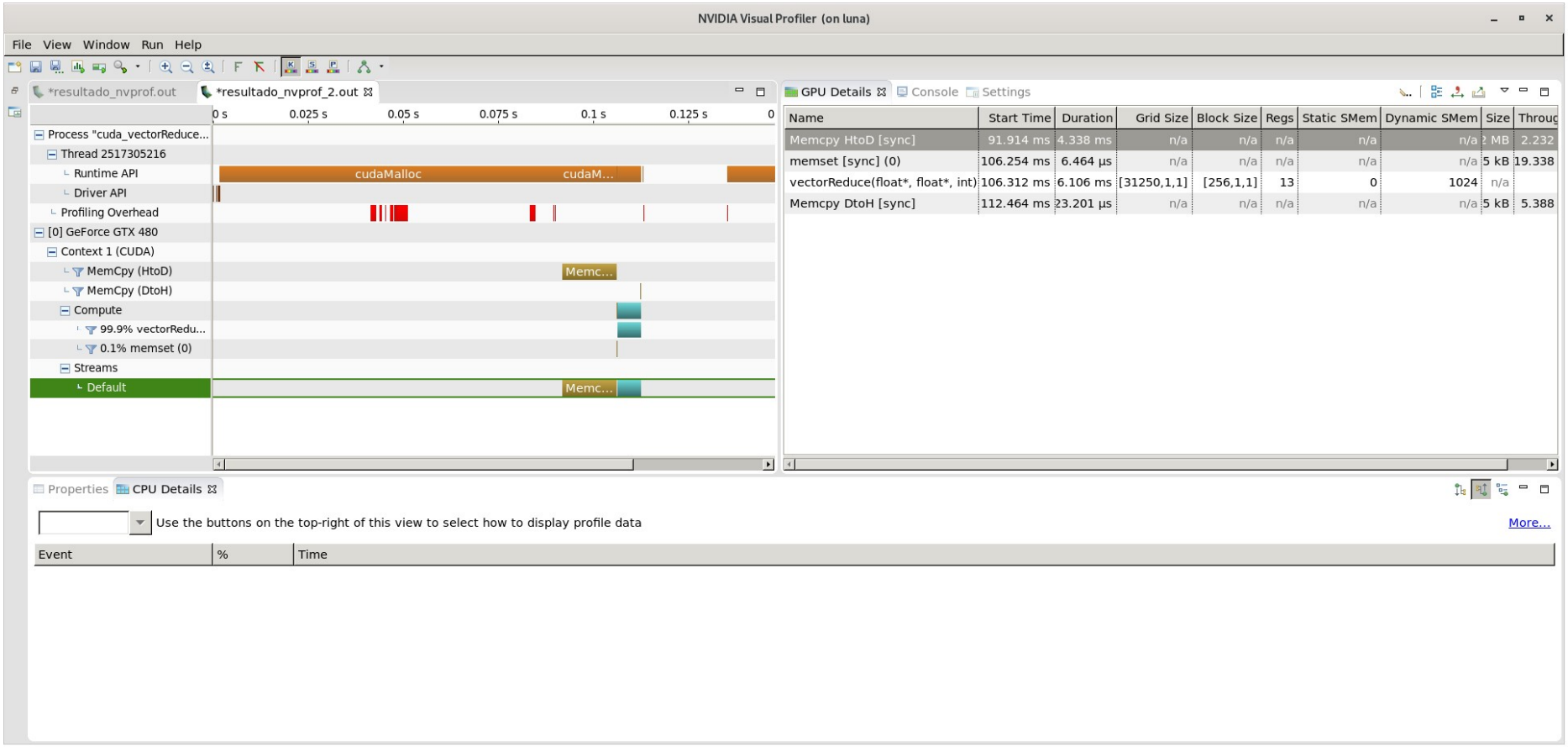
Se puede generar un fichero de salida desde nvprof en un nodo de cómputo

```
$ nvprof --export-profile resultado_nvprof.out ./cuda_vectorReduce -n=8000000 -bsx=128
```

Y, después, abrirlo/importarlo desde `nvvp` para visualizar el *timeline*

3.2. Depuración y medición de uso de recursos

Medición de recursos al ejecutar: nvprof + nvvp

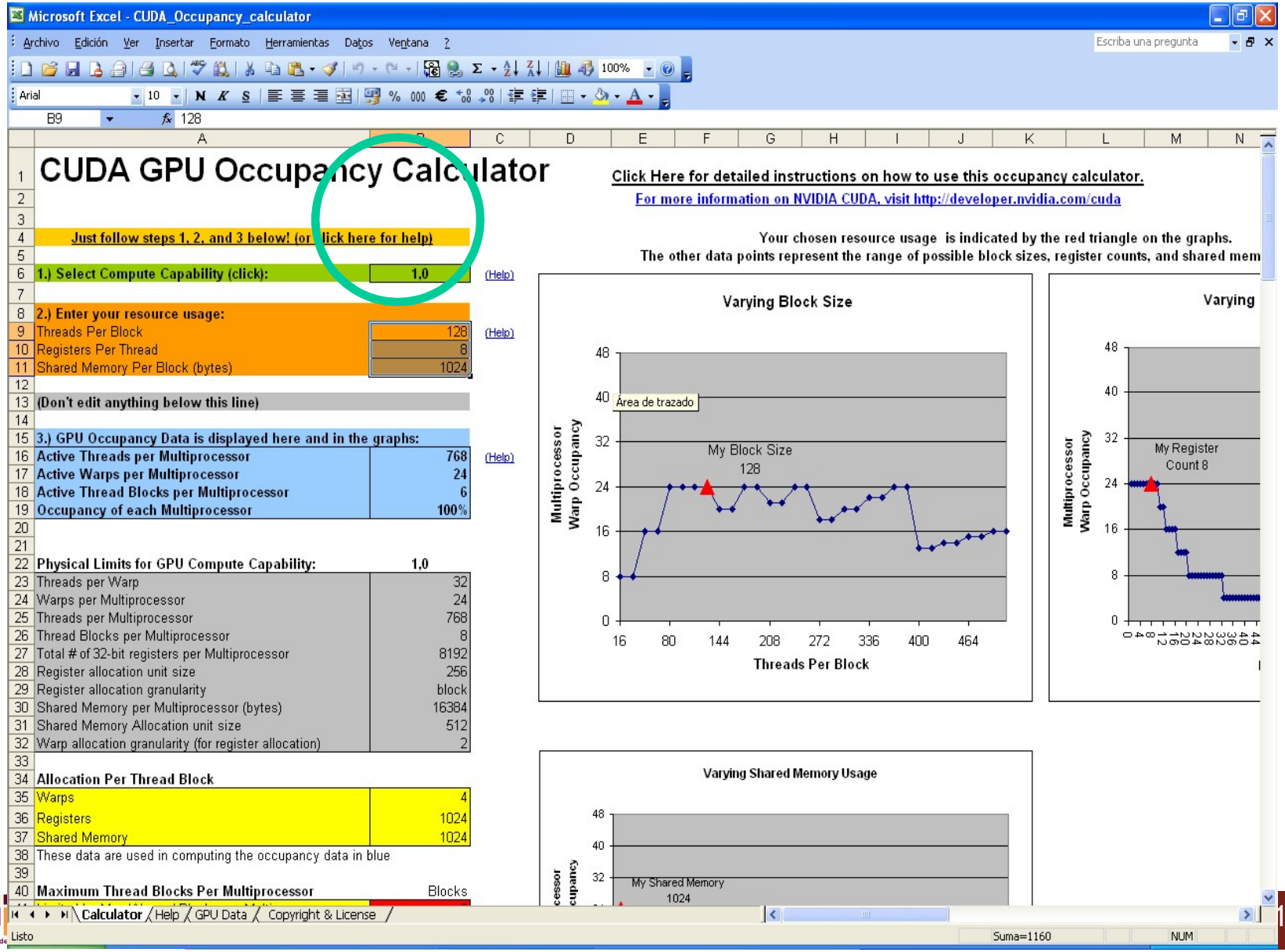


Contenidos

3.2. Depuración y medición de uso de recursos
Configuración de la ejecución: ocupación de la GPU

- **Ocupación de la GPU**
  - La **ocupación** se define como el ratio entre el número de *threads* activos por SM y el número máximo posible
    - Mayor ocupación no garantiza mejor rendimiento pero…
    - …baja ocupación implica normalmente peor rendimiento
      - Incapacidad para ocultar las largas latencias de acceso a memoria global
  - El número máximo de *threads* activos está limitado por el consumo de registros y de memoria compartida
  - Para determinar la configuración óptima NVIDIA proporciona:
    CUDA *Occupancy Calculator*

3.2. Depuración y medición de uso de recursos
Configuración de la ejecución: ocupación de la GPU



3.2. Depuración y medición de uso de recursos

Configuración de la ejecución: ocupación de la GPU

Microsoft Excel - CUDA_Occupancy_calculator

Archivo Edición Ver Insertar Formato Herramientas Datos Ventana ?

Escriba una pregunta

Arial Unicode MS 10 150%

A2 Compute Capability

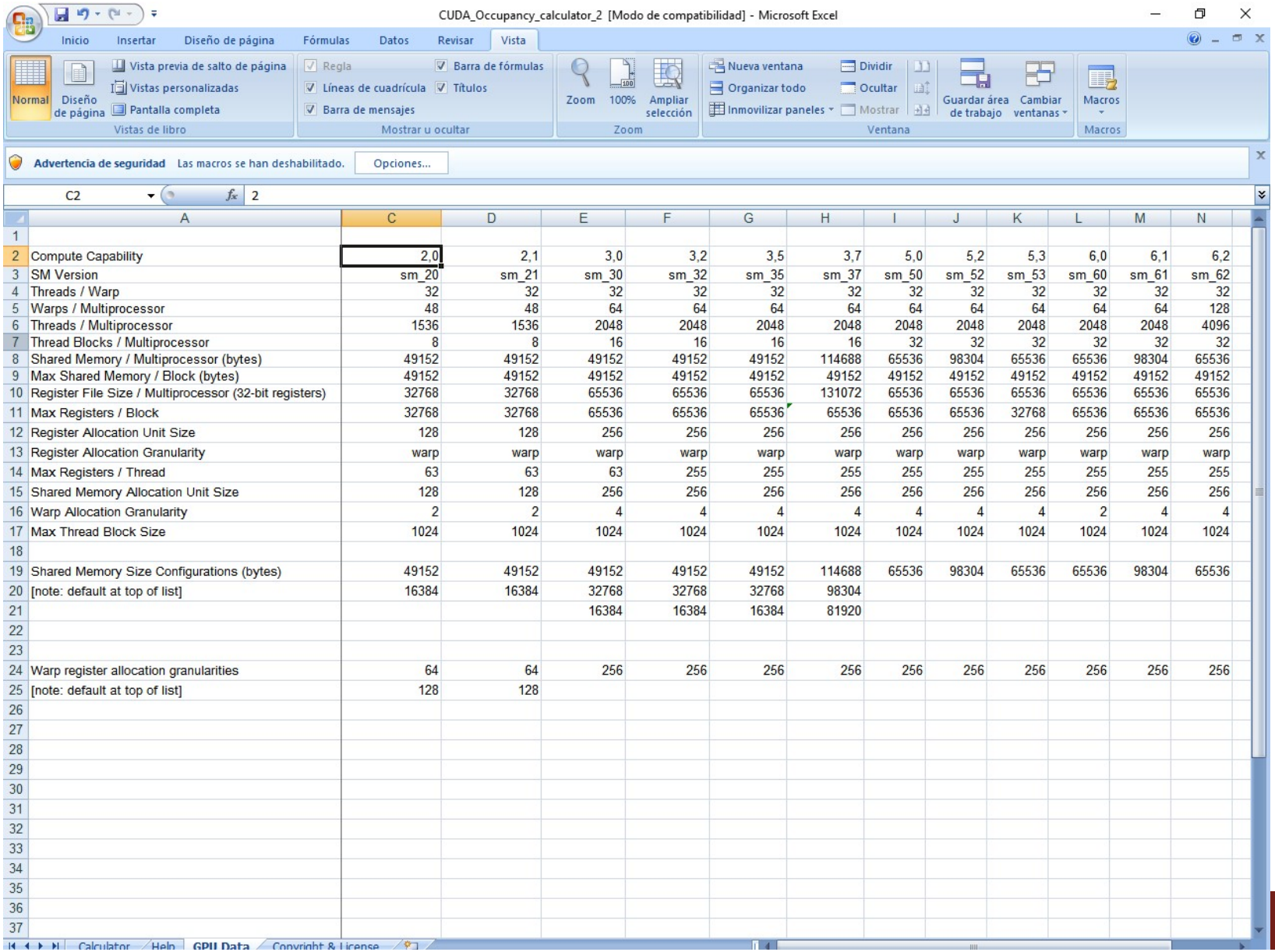
	A	B	C	D	E	F
1						
2	Compute Capability	1,0	1,1	1,2	1,3	2,0
3						
4	Threads / Warp	32	32	32	32	32
5	Warps / Multiprocessor	24	24	32	32	48
6	Threads / Multiprocessor	768	768	1024	1024	1536
7	Thread Blocks / Multiprocessor	8	8	8	8	8
8	Shared Memory / Multiprocessor (bytes)	16384	16384	16384	16384	49152
9	Register File Size	8192	8192	16384	16384	32768
10	Register Allocation Unit Size	256	256	512	512	64
11	Allocation Granularity	block	block	block	block	warp
12	Shared Memory Allocation Unit Size	512	512	512	512	128
13	Warp allocation granularity (for registers)	2	2	2	2	
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						

Calculator Help GPU Data Copyright & License

Listo Suma=205878,6 NUM

3.2. Depuración y medición de uso de recursos

Configuración de la ejecución: ocupación de la GPU



CUDA_Occupancy_calculator_2 [Modo de compatibilidad] - Microsoft Excel

Inicio | Insertar | Diseño de página | Fórmulas | Datos | Revisar | Vista

Normal | Diseño de página | Vista previa de salto de página | Vistas personalizadas | Pantalla completa — Vistas de libro

Regla | Barra de fórmulas | Líneas de cuadrícula | Títulos | Barra de mensajes — Mostrar u ocultar

Zoom | 100% | Ampliar selección — Zoom

Nueva ventana | Organizar todo | Inmovilizar paneles | Dividir | Ocultar | Mostrar | Guardar área de trabajo | Cambiar ventanas — Ventana

Macros — Macros

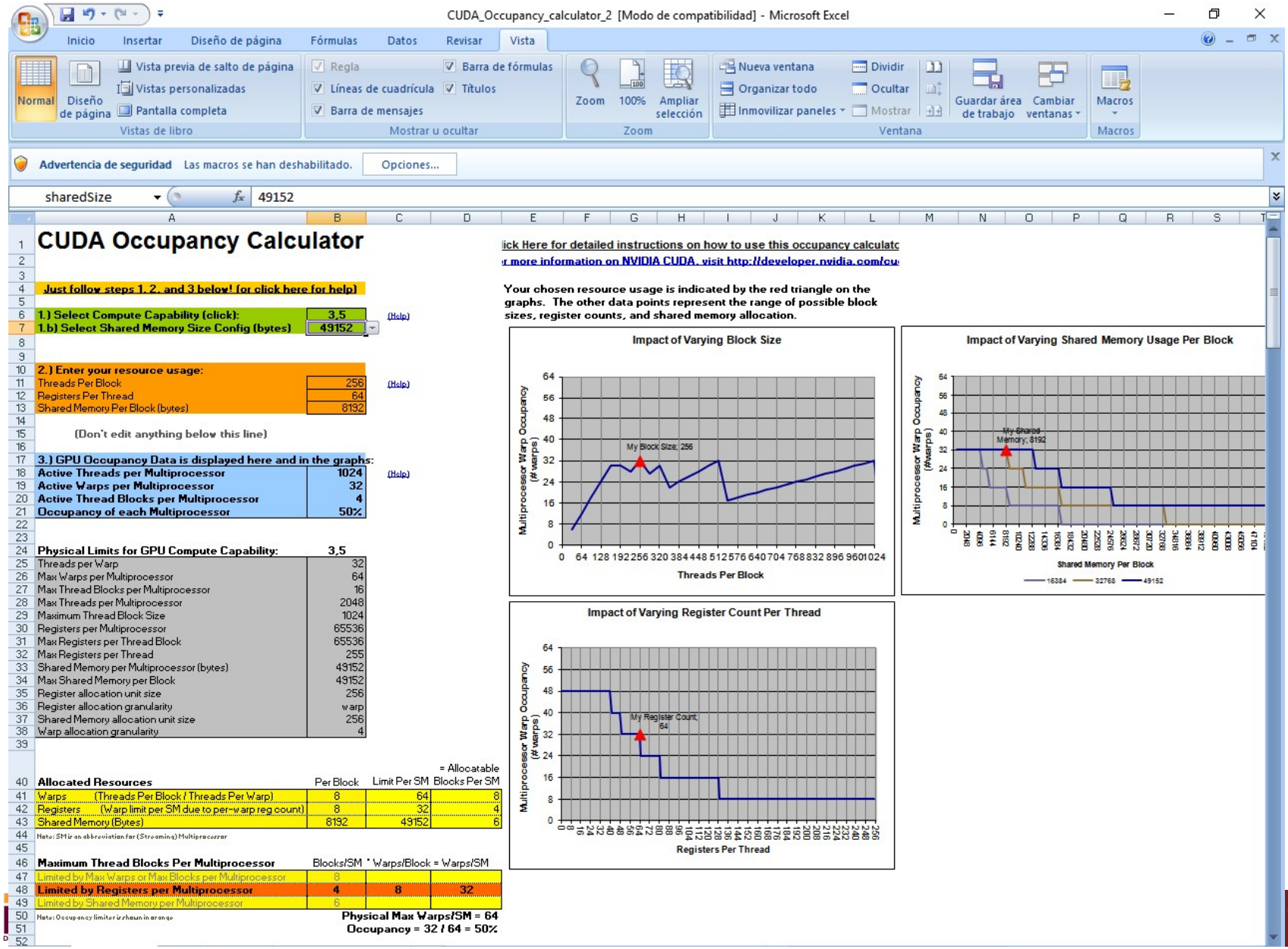
Advertencia de seguridad Las macros se han deshabilitado. Opciones...

C2 | 2

	A	C	D	E	F	G	H	I	J	K	L	M	N
1													
2	Compute Capability	2,0	2,1	3,0	3,2	3,5	3,7	5,0	5,2	5,3	6,0	6,1	6,2
3	SM Version	sm_20	sm_21	sm_30	sm_32	sm_35	sm_37	sm_50	sm_52	sm_53	sm_60	sm_61	sm_62
4	Threads / Warp	32	32	32	32	32	32	32	32	32	32	32	32
5	Warps / Multiprocessor	48	48	64	64	64	64	64	64	64	64	64	128
6	Threads / Multiprocessor	1536	1536	2048	2048	2048	2048	2048	2048	2048	2048	2048	4096
7	Thread Blocks / Multiprocessor	8	8	16	16	16	16	32	32	32	32	32	32
8	Shared Memory / Multiprocessor (bytes)	49152	49152	49152	49152	49152	114688	65536	98304	65536	65536	98304	65536
9	Max Shared Memory / Block (bytes)	49152	49152	49152	49152	49152	49152	49152	49152	49152	49152	49152	49152
10	Register File Size / Multiprocessor (32-bit registers)	32768	32768	65536	65536	65536	131072	65536	65536	65536	65536	65536	65536
11	Max Registers / Block	32768	32768	65536	65536	65536	65536	65536	65536	32768	65536	65536	65536
12	Register Allocation Unit Size	128	128	256	256	256	256	256	256	256	256	256	256
13	Register Allocation Granularity	warp	warp	warp	warp	warp	warp	warp	warp	warp	warp	warp	warp
14	Max Registers / Thread	63	63	63	255	255	255	255	255	255	255	255	255
15	Shared Memory Allocation Unit Size	128	128	256	256	256	256	256	256	256	256	256	256
16	Warp Allocation Granularity	2	2	4	4	4	4	4	4	4	2	4	4
17	Max Thread Block Size	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
18													
19	Shared Memory Size Configurations (bytes)	49152	49152	49152	49152	49152	114688	65536	98304	65536	65536	98304	65536
20	[note: default at top of list]	16384	16384	32768	32768	32768	98304						
21				16384	16384	16384	81920						
22													
23													
24	Warp register allocation granularities	64	64	256	256	256	256	256	256	256	256	256	256
25	[note: default at top of list]	128	128										
26													
27													
28													
29													
30													
31													
32													
33													
34													
35													
36													
37													

Calculator | Help | GPU Data | Copyright & License

3.2. Depuración y medición de uso de recursos
Configuración de la ejecución: ocupación de la GPU



Contenidos

4. Librerías basadas en CUDA

- **CUBLAS**: *Basic Linear Algebra Subprograms* (BLAS)
- **CUFFT**: *Fast Fourier Transform* (FFT)
- **CUSPARSE**: BLAS para matrices dispersas
- **Thrust**: data parallel primitives: scan, sort, reduce,...
- **NVIDIA *Performance Primitives* (NPP)**
  - Implementación de funciones de diversa naturaleza para procesamiento de imágenes, video y señal
- **CULA**: Linear Algebra PACKage (LAPACK)
- **NVIDIA cuDNN**:primitivas para redes neuronales profundas)

4. Librerías basadas en CUDA

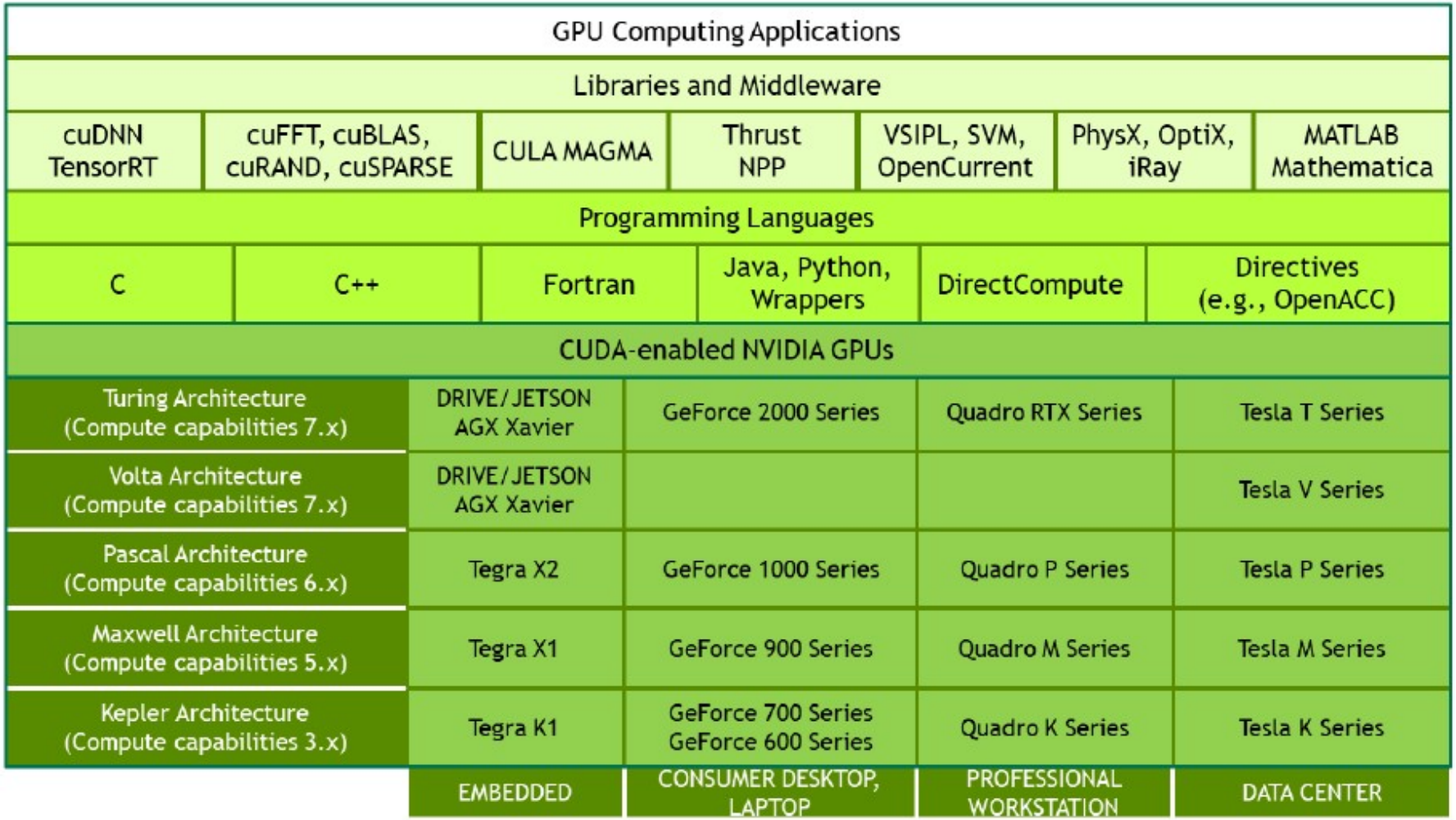


Figure 4 GPU Computing Applications

CUDA is designed to support various languages and application programming interfaces.

Contenidos

5. Alternativas a NVIDIA/CUDA

- ***Open Computing Language* (OpenCL)**
  - Estándar abierto de Khronos OpenCL *working group*
    - NVIDIA, ATI/AMD, Intel, IBM, etc.
  - Solución basada en una librería cuyo objetivo principal es conseguir soportar plataformas heterogéneas así como facilitar la portabilidad entre plataformas
    - CPUs, GPUs, Cell BE, DSPs, etc.
  - Modelos de memoria, ejecución y programación similares a CUDA

5. Alternativas a NVIDIA/CUDA

- ***Open Computing Language* (OpenCL)**

```
__kernel /* Código GPU */

void saxpy_parallel(__global float* Y,
                    __global float A,
                    __global float* X)
{
   int tid = get_global_id(0);
   Y[tid] = A * X[tid] + Y[tid];
}
```

5. Alternativas a NVIDIA/CUDA

- **Open Accelerators (OpenACC)**
  (http://www.openacc.org/)

  - Colección de directivas (al estilo OpenMP)
  - Portabilidad entre CPUs, GPUs y otros aceleradores
  - Desarrollado por Cray, CAPS, NVIDIA y PGI
  - El programador crea programas sin necesidad de inicializar el acelerador, gestionar datos o indicar transferencias CPU-acelerador
  - El programador es el responsable de mantener la coherencia de los datos GPU/CPU

5. Alternativas a NVIDIA/CUDA

- **O**pen **Acc**elerators (OpenACC)

```
#pragma acc data create(a[0:n]) present(x[0:n],b[0:n])
{
   // following loop executed on device
   #pragma acc parallel loop
   for(i=0;i<n;++i) a[i] = b[i];

   // following loop executed on host
   for(i=0;i<n;++i) a[i] = c[i];

   // following loop executed on device
   #pragma acc parallel loop
   for(i=0;i<n;++i) x[i] = a[i];

   ...
}
```

5. Alternativas a NVIDIA/CUDA

- **Compitiendo…**
  - nVIDIA CUDA
  - AMD HIP-ROCm
    - Objetivo: Un "clon" de CUDA multiplataforma
    - Funciona también en tarjetas nVIDIA, pero con menos prestaciones que CUDA
  - Intel oneAPI
    - Objetivo: soporte multiplataforma
    - Compilador DPC++

      (basado en los estándares ISO C++ y Khronos Group SYCL)

Contenidos

6. Conclusiones

- La paralelización de aplicaciones con CUDA difiere significativamente de la misma tarea con OpenMP sobre procesadores multinúcleo convencionales
  - Descomposición en subproblemas INDEPENDIENTES
    - Paralelismo de datos que debe identificar el programador
  - *# threads*
  - *C*oste de creación y destrucción de los *threads*
  - Coste de los cambios de contexto
  - Comunicación y sincronización de *threads*
  - Tipos y gestión de la memoria disponible
  - Ausencia de bloqueos si se siguen unas reglas básicas

6. Conclusiones

- La mejora de rendimiento obtenida dependerá del porcentaje del tiempo de ejecución que supongan los *kernels* sobre toda la aplicación (Ley de Amdahl)

- Las aplicaciones paralelizadas con CUDA escalan conforme aumentamos el número de SMs y/o GPUs

  – ¿Adaptación de la configuración de ejecución?

6. Conclusiones

- ¿Qué aplicaciones son susceptibles de ser paralelizadas con éxito utilizando CUDA?
  - Paralelismo de datos mejor que paralelismo de tareas
    - SIMT vs. SIMD
      - La divergencia de *threads* penaliza el rendimiento
  - Procesamiento de bloques de datos independientes
    - El flujo de control debe ser independiente de los datos
      - Algoritmos con muchas dependencias de datos serán difíciles de paralelizar de manera eficiente

6. Conclusiones

- ¿Qué aplicaciones son susceptibles de ser paralelizadas con éxito utilizando CUDA?
  - Patrones de acceso a memoria regulares o susceptibles de ser adaptados usando la memoria compartida
  - Intensidad aritmética suficiente para amortizar las copias y mantener ocupados todos los recursos hardware de la GPU
  - IEEE 754 SP proporcionará una mayor mejora que DP (especialmente si tenemos una tarjeta antigua)

Contenidos

7. Bibliografía

- Manuales
  - *CUDA Programming Guide*
  - *CUDA C Programming Best Practices Guide*
  - *CUDA Reference Manual*

- *Websites*
  - NVIDIA *Developer Zone*:
    http://developer.nvidia.com/object/gpucomputing.html (CUDA)

7. Bibliografía

- Artículos
  - Dinesh Manocha, General-Purpose Computations using Graphics Procesors, IEEE Computer, vol. 38 no. 8, pp. 85–88, August 2005
  - Erik Lindholm et al., NVIDIA Tesla: A Unified Graphics and Computing Architecture, IEEE Micro, vol. 28 no. 2, pp. 39–55, March 2008
  - Tom R. Halfhill, Parallel Processing with CUDA, MicroProcessor Report online (http://www.mpronline.com), January 2008
  - John Nickolls et al., Scalable Parallel Programming, ACM Queue, vol. 6 no. 2, pp. 40–53, March/April 2008
  - Wen-mei Hwu et al., Compute Unified Device Architecture Application Suitability, Computing in Science and Engineering, vol. 11 no. 3, pp. 16, 26, May/June 2009
  - Michael Garland et al., Parallel Computing Experiences with CUDA, IEEE Micro, vol. 28 no. 4, pp. 13–27, July 2008

Contenidos

8. ANEXO
Programación híbrida CPU+multiGPU (OpenMP+CUDA)

AYUDAS/CONSEJOS:

- En `Makefile`:

```
NVCCFLAGS := ... -Xcompiler -fopenmp
```

- En `programa.cu`:

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

- Definición para cada GPU de sus variables de memoria global

- `cudaSetDevice(v)`: Establece que el código actúe en GPU `v-esima` (reserva de memoria, copia de datos, invocación kernel,...)

8. ANEXO
Programación híbrida CPU+multiGPU (OpenMP+CUDA)

AYUDAS/CONSEJOS:

- Organización de trabajo, por ejemplo:
  - Primeros `v` threads de CPU hacen actuar a las `v` GPUs:
    - (envía datos entrada, lanza el kernel, recoge resultados)
  - Resto de threads de CPU: realizan cálculo que corresponde a CPU