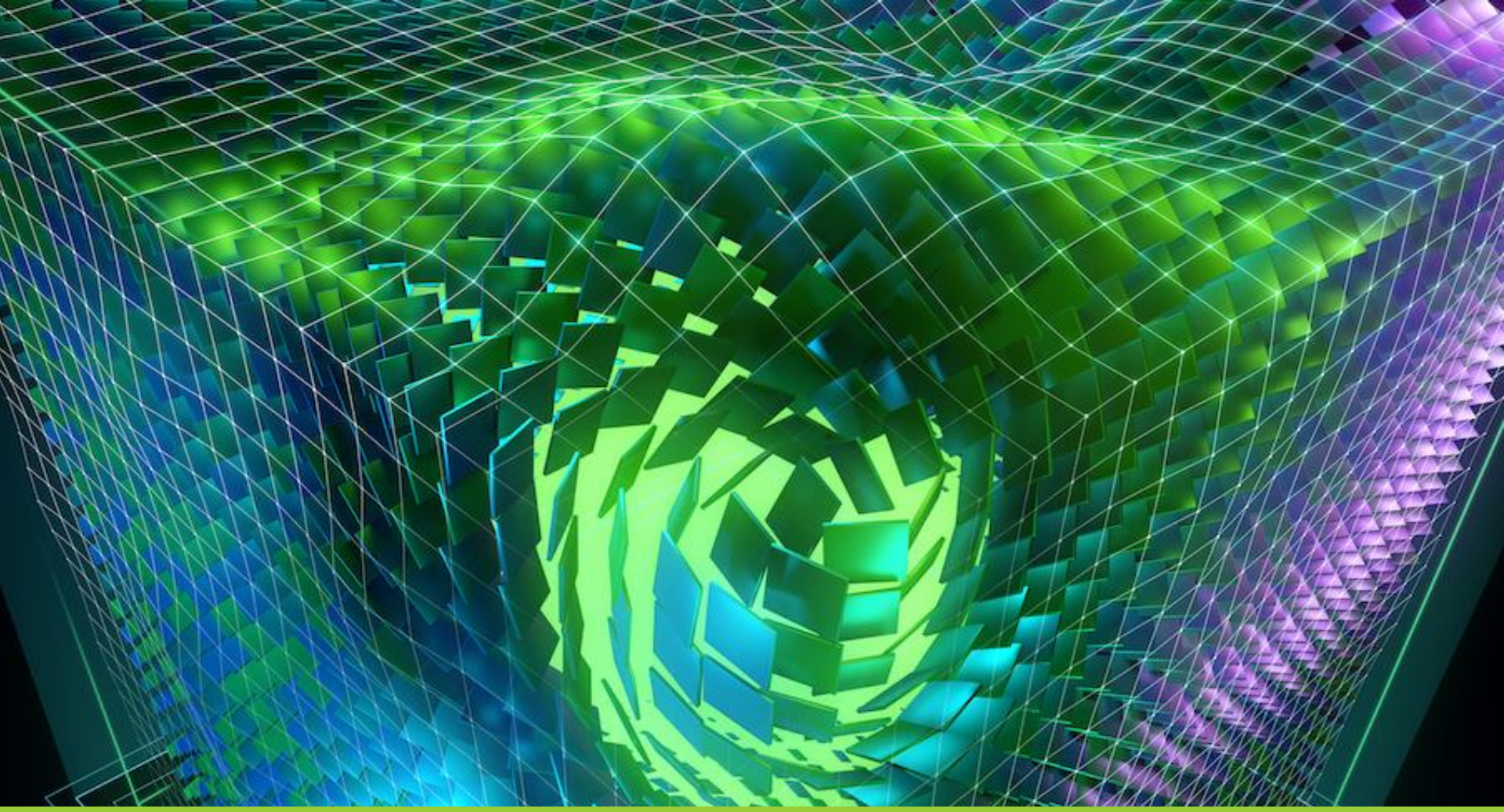

INF351 – Computación de Alto Desempeño

Compilación

PROF. ÁLVARO SALINAS

NVCC

Ahora que ya sabemos como crear nuestro código de CUDA, solo nos falta saber como compilarlo. Por suerte es un proceso idéntico a lo que conocemos de C/C++.

Para usuarios de Visual Studio, la compilación resulta trivial, pues es una opción en el menú del programa.

Para usuarios de Linux o usuarios de Windows que prefieran utilizar la consola, el compilador es nvcc y funciona del mismo modo que gcc y g++.

Ejemplos básicos:

```
nvcc main.cu
```

```
nvcc -o ejecutable main.cu
```

NVCC - Microarquitectura

Con una compilación simple como la anterior, estamos permitiendo que el compilador asuma la microarquitectura (compute capability) de la GPU que utilizaremos.

Es buena práctica especificar qué GPU estamos utilizando. La forma más simple de realizar esto es con la opción -arch:

```
nvcc -o ejecutable -arch sm_61 main.cu
```

Donde sm_XY es utilizado para una GPU con compute capability X.Y (microarquitectura Pascal con compute capability 6.1 para el ejemplo anterior).

NVCC - Opciones

Al igual que con cualquier compilador, existe un gran número de opciones que podemos utilizar al compilar códigos en CUDA.

Para averiguar más sobre este tema, diríjanse a la documentación oficial:

https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#nvcc-command-options

Aquí podrán encontrar opciones para:

- Manejo de directorios
- Compatibilidad
- Optimizaciones
- Control de tamaño de memorias
- Uso de librerías
- Otras opciones

NVCC - ptxas

Una opción que nos puede ser de utilidad es --ptxas-options o -Xptxas que junto a -v nos entrega información sobre el uso de registros de nuestros kernels.

Un ejemplo de compilación sería:

```
nvcc -o ejecutable -arch sm_61 -Xptxas -v main.cu
```

Pudiendo arrojar el siguiente resultado:

```
ptxas info    : Compiling entry function '_Z10kernelPiii' for 'sm_61'
ptxas info    : Function properties for _Z10kernelPiii
        0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 8 registers, 336 bytes cmem[0]
```

CUDA-MEMCHECK

Una herramienta que nos ayudará mucho al encontrar un bug en nuestra aplicación es cuda-memcheck. Ésta nos permitirá generar un reporte con los errores de nuestro código con respecto al manejo de memoria.

Para utilizarla, simplemente debemos ejecutarla con nuestra aplicación ya compilada:

cuda-memcheck ejecutable

Los errores que pueden ser detectados son:

- Errores de accesos de memoria
- Errores reportados por el hardware (exceptions)
- Errores al utilizas malloc()/free() dentro de un kernel
- Asignaciones de memoria (cudaMalloc() en host y malloc() en device) que no han sido liberadas