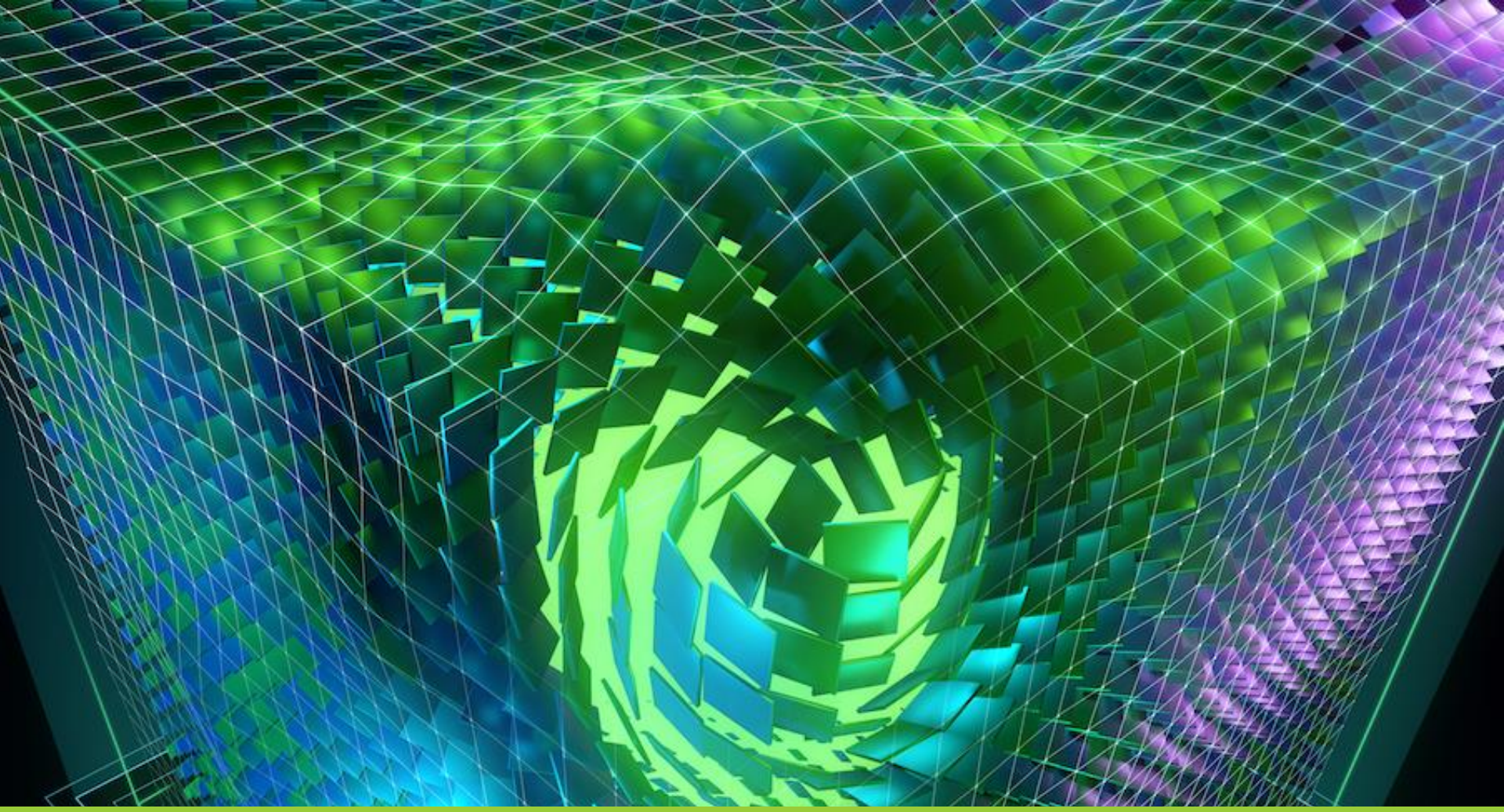


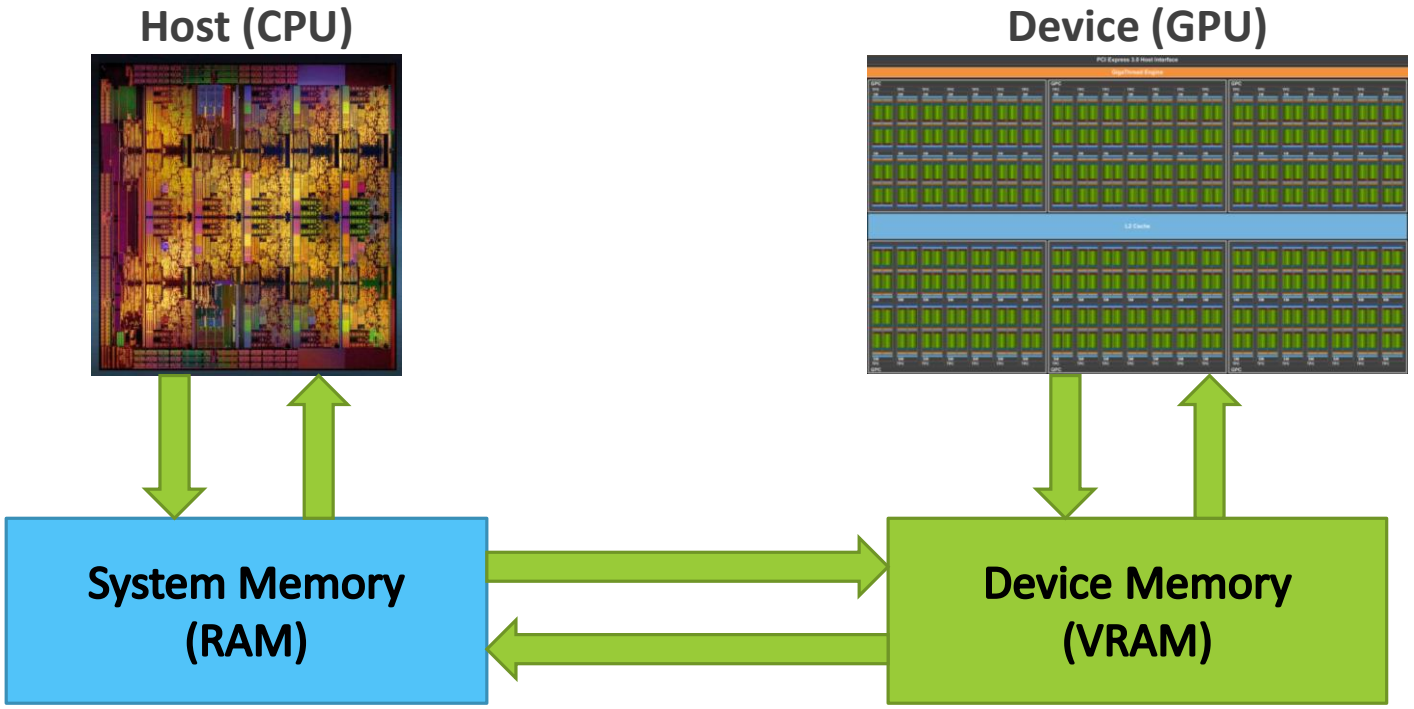
INF351 – Computación de Alto Desempeño

Manejo de Memoria entre CPU y GPU

PROF. ÁLVARO SALINAS

Transferencia de Datos

Pasar datos desde la CPU a la GPU o viceversa es un proceso bastante costoso, por lo que la cantidad de estas transferencias debe ser mínima, en lo posible.



Parámetros Pasados por Valor

Cuando un parámetro es pasado por valor a un CUDA kernel, al momento de la invocación, CUDA copia automáticamente el valor de la variable en el stack de la GPU, por lo que no es necesaria una declaración especial.

```
__global__ void kernel(int a) {
    a = 4;
    printf("%d\n", a); // 4
}

int main() {
    int a = 2;
    kernel<<<1, 1>>>(a);
    std::cout << a << std::endl; // 2
    return 0;
}
```

Parámetros Pasados por Referencia

Cuando necesitamos que los cambios realizados en el kernel a una determinada variable se conserven, es necesario asignarle una dirección de memoria en la GPU.

Esto podemos lograrlo de dos formas:

- Estática
- Dinámica

Declaración Estática

Para declarar una variable de forma estática debemos utilizar el descriptor *__device__*.

La declaración debe estar en global scope, i.e. fuera del main, y antes de la definición de los kernels en donde se utilice la variable.

Cuando declaramos una variable de esta forma, la memoria es asignada y posteriormente liberada automáticamente.

Cuando declaramos un arreglo, debemos saber su dimensión en tiempo de compilación (por algo es estática), por ejemplo:

- *__device__ int a[20];*

Una variable declarada estáticamente es visible por todos los kernels posteriores a su declaración, por lo que no es necesario pasarla como parámetro.

Declaración Estática

La copia de valores desde CPU a GPU y viceversa deben manejarse manualmente.

Para copiar valores desde una variable de CPU a una variable declarada estáticamente en GPU es necesario utilizar:

cudaMemcpyToSymbol(const void symbol, const void* src, size_t count, size_t offset = 0, cudaMemcpyKind kind = cudaMemcpyHostToDevice)*

En el caso contrario, para copiar un valor desde GPU a CPU tenemos:

cudaMemcpyFromSymbol(void dst, const void* symbol, size_t count, size_t offset = 0, cudaMemcpyKind kind = cudaMemcpyDeviceToHost)*

Declaración Estática

```
__device__ int dev_a;

__global__ void kernel() {
    printf("%d\n", dev_a); // 2
    dev_a = 4;
}

int main() {
    int host_a = 2;
    cudaMemcpyToSymbol(dev_a, &host_a, sizeof(int), 0, cudaMemcpyHostToDevice);
    kernel<<<1, 1>>>();
    cudaMemcpyFromSymbol(&host_a, dev_a, sizeof(int), 0, cudaMemcpyDeviceToHost);
    std::cout << host_a << std::endl; // 4
    return 0;
}
```

Declaración Dinámica

Para declarar una variable de forma dinámica es necesario crear un puntero normal y reservar la memoria que ocupará en el dispositivo con *cudaMalloc(void ** devPtr, size_t size).*

La declaración debe estar en código de CPU, i.e. dentro del main u otra función.

Cuando declaramos una variable de esta forma, debemos preocuparnos de liberada posteriormente con *cudaFree(void* devPtr)*.

Una variable declarada dinámicamente debe ser pasada por referencia a los kernels en donde se usará.

Declaración Dinámica

La copia de valores desde CPU a GPU y viceversa también deben manejarse manualmente al declarar una variable de forma dinámica.

Para copiar valores desde una variable de CPU a una variable declarada dinámicamente en GPU, o viceversa, es necesario utilizar:

cudaMemcpy (void dst, const void* src, size_t count, cudaMemcpyKind kind)*

Declaración Dinámica

```
__global__ void kernel(int *dev_a) {
    printf("%d\n", *dev_a); // 2
    *dev_a = 4;
}

int main() {
    int host_a = 2;
    int *dev_a;
    cudaMalloc(&dev_a, sizeof(int));
    cudaMemcpy(dev_a, &host_a, sizeof(int), cudaMemcpyHostToDevice);
    kernel<<<1, 1>>>(dev_a);
    cudaMemcpy(&host_a, dev_a, sizeof(int), cudaMemcpyDeviceToHost);
    std::cout << host_a << std::endl; // 4
    cudaFree(dev_a);
    return 0;
}
```