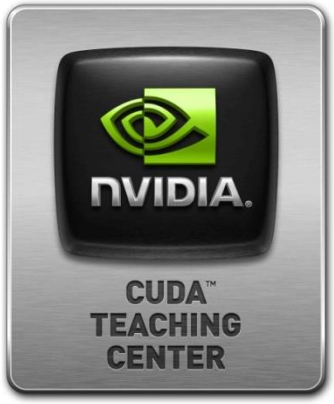
Programación masivamente paralela en procesadores gráficos (GPUs)

E. Dufrechou , P. Ezzatti y M. Pedemonte



Clase 7
Programación CUDA III

Contenido

- Transferencias a memoria
- Ejemplos de memoria compartida

Transferencias a Memoria

Transferencias a memoria

Memoria no paginable

- cudaHostAlloc()
  - Mayor ancho de banda
  - Posibilidad de mapear memoria del host en el device.
  - Es un recurso limitado
  - Evita copia de paginable a no paginable.

- Optimización de transferencias
  - Agrupar transferencias
  - Evitar transferencia de poca memoria

Transferencias a memoria

Transferencias asincrónicas

- **Utilizan streams (no entraremos en detalles)**

```
cudaMallocHost( (void **)& hostPtr, NUM_STREAMS * size) ;

cudaStreamCreate( &stream [ i ] );

cudaMemcpyAsync ( inputDevPtr + i * size,

                  hostPtr + i * size, size,

                  cudaMemcpyHostToDevice,  stream [ i ] );
```

- **Permiten solapar cálculos con transferencias.**
- **Ojo! Ahora las transferencias no sincronizan!**

Ejemplos Memoria Compartida

Lanzamiento del kernel (5) versión del kernel (3) con memoria compartida

```
// configuración de la ejecución
int chunk = 32;
dim3 tamGrid(M, 1); //Grid dimensión
dim3 tamBlock(N/chunk,1,1); //Block dimensión
// lanzamiento del kernel
SumaFilaMatrizKernel<<<tamGrid, tamBlock>>>(M, Md, Nd);
```

Programación en CUDA

```
__global__ void SumaFilaMatrizKernel(int M, float*Md, float* Nd){
__shared __float Nds[DIMBLOCKX];
    float Pvalue = 0;
    int aux = blockIdx.x*M + threadIdx.x*(M/blockDim.x);
    int aux2 = aux + (M/blockDim.x);
    for (int k = aux; k < aux2; ++k) {
      Pvalue = Pvalue + Md[k];
    }
    Nds[threadIdx.x] = Pvalue;

    if (threadIdx.x == 0){
      for (int i = 1; i < blockDim.x; ++i) {
          Nds[0] = Nds[0]+ Nds[i];
      }
      Nd[blockIdx.x] = Nds[0];
    }
}
```

Programación en CUDA

Recordemos ….

- `__syncthreads`: permite sincronizar threads del mismo bloque.

Se invoca dentro de un kernel.

Programación en CUDA

```
__global__ void SumaFilaMatrizKernel(int M, float* Md, float*
Nd){
__shared __float Nds[DIMBLOCKX];
    float Pvalue = 0;
    int aux = blockIdx.x*M + threadIdx.x*(M/blockDim.x);
    int aux2 = aux + (M/blockDim.x);
    for (int k = aux; k < aux2; ++k) {
      Pvalue = Pvalue + Md[k];
    }
    Nds[threadIdx.x] = Pvalue;
    __syncthreads();

    if (threadIdx.x == 0){
      for (int i = 1; i < blockDim.x; ++i) {
          Nds[0] = Nds[0]+ Nds[i];
      }
      Nd[blockIdx.x] = Nds[0];
    }
}
```

Programación en CUDA

Queda como tarea domiciliaria:

- Extender el kernel 4 de la clase 5 para que utilice memoria compartida (kernel 6).

- Modificar los kernels 5 y 6 para que las lecturas de memoria global sean coalesced.