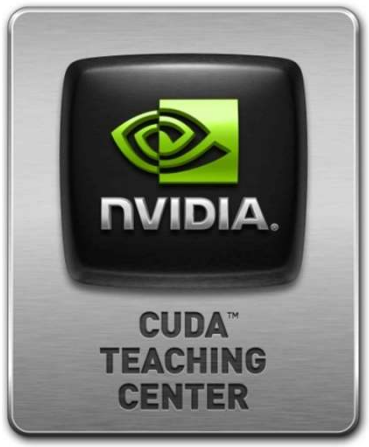
Computación de Propósito General en Unidades de Procesamiento Gráfico (GPGPU)

E. Dufrechou, P. Ezzatti y M. Pedemonte



Clase 10
Ecosistema CUDA (parte 2)

Contenido

- Introducción
- Bibliotecas
  - Thrust
  - CUB
  - Modern GPU
- Ejemplo

Introducción

- Existen tipos de problemas muy repetidos por lo que es interesante hacer soluciones estándar (patrones de cómputo). Ej: Scan, Reduce
- La programación en CUDA es compleja y requiere una gran curva de aprendizaje
- Tiene sentido hacer implementaciones (relativamente) eficientes y genéricas que puedan ser usadas por programadores con poca experiencia

Thrust

Conceptos generales

- Extensión de C++ Standard Template Library (STL) para usar en GPUs (y otros dispositivos de cómputo heterogéneo)
- Abstrae el manejo de memoria utilizando wrappers de punteros (device, host) y contenedores (vector)
- Para CUDA existen implementaciones eficientes de los principales algoritmos (merge, sort, scan, reduce, etc.)
- Interoperabilidad con el resto de CUDA

Manejo de memoria

- Puede hacerse de manera abstracta (con contenedores) o utilizando punteros (usando un wrapper para device)
- Al usar contenedores la reserva y liberación se hace automática (aunque es más eficiente liberar manual!!!)
- Los wrappers mantienen información (tipo, dispositivo, etc), no son únicamente punteros.
- Siempre se puede obtener un puntero con los datos utilizando la función data() (y casteando a raw pointer)

Algoritmos

Se desarrolla mediante el uso de algoritmos de alto nivel, delegando a la biblioteca como se implementa.

Los algoritmos básicos:

- **for_each**
- **scan**
- **sort**
- **reduce**
- **merge**
- **find**
- **transformaciones**
- **etc.**

No se especifica configuración de ejecución !!!

Cub

Conceptos generales

- Surge de Thrust por necesidades de mantenimiento y portabilidad de su rendimiento reusando primitivas a niveles más bajos (block, warp)
- Ligeramente más bajo nivel que thrust
- Es una librería de rutinas implementada específicamente para CUDA C++
- El manejo de memoria es idéntico al de CUDA (cudaMalloc, cudaFree, punteros) y corre por el programador
- Implementaciones (más) eficientes de los principales algoritmos (merge, sort, scan, reduce, etc.)

Algoritmos

- Se dividen en tres grupos según su granularidad (warp, block y device)
- Además de los mencionados anteriormente presenta otros "específicos" tanto a nivel de bloque (discontinuity, adjacent difference) como device (select, runLength encoding, SpMV)
- Se puede trabajar en problemas a batch (segmented sort)

ModernGPU

Modern GPU

- Biblioteca pensada para un uso educacional
- Prioriza la claridad y el reuse por sobre la eficiencia computacional
- ModernGPU no ejecuta con valores “grandes”
- No es utilizada en la práctica

Ejemplo práctico

Ejemplo práctico

- **Se eligió la primitiva reduce por su extensa utilizacion**
- **Recordemos la implementación**

```
extern __shared__ long int intermedio[];
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if(idx < sz)
    intermedio[threadIdx.x] = input[idx];
else{
    intermedio[threadIdx.x] = 0;
}
__syncthreads();
int i = blockDim.x/2;
while (i != 0) {
if (threadIdx.x < i)
    intermedio[threadIdx.x] =
    intermedio[threadIdx.x] +
    intermedio[threadIdx.x + i];
__syncthreads();
i = i / 2;
}//endwhile

__syncthreads();
if (threadIdx.x==0 && idx < sz)
   output[blockIdx.x] = intermedio[0];
```

Ejemplo práctico

- **Se eligió la primitiva reduce por su extensa utilización**
- **¿Cómo queda en Thrust?**

```
thrust::device_ptr<long int> temp_ptr(d_in);
thrust::device_vector<long int> in_thrust(temp_ptr, temp_ptr + size);
d_out_h[0]= thrust::reduce(in_thrust.begin(),in_thrust.begin()+size);
cudaDeviceSynchronize();
```

Ejemplo práctico

- **Se eligió la primitiva reduce por su extensa utilización**
- **¿Cómo queda en Cub?**

```
struct CustomSum{
   template <typename T>
   __device__ __forceinline__
   T operator()(const T &a, const T &b) const {
      return a+b;
   }
};
CustomSum    sum_op;

void     *d_temp_storage = NULL;
size_t   temp_storage_bytes = 0;
cub::DeviceReduce::Reduce(d_storage, storage_bytes, d_in, d_out, size, sum_op, 0);
cudaMalloc(&d_storage, storage_bytes);

cub::DeviceReduce::Reduce(d_storage,storage_bytes, d_in, d_out, size, sum_op, 0);
cudaDeviceSynchronize();
```

Ejemplo práctico

- **Se eligió la primitiva reduce por su extensa utilización**
- **¿Cómo queda en ModernGPU?**

```
ContextPtr context = CreateCudaDevice(0)
MGPU_MEM(int) data = context.GenRandom<int>(size, 0, size);
int total = Reduce(data->get(), size, context);
```

Ejemplo práctico

- Se ejecutó cada algoritmo 500 veces en una tarjeta Nvidia RTX 3090 Ti
- Se evaluó el tiempo de ejecución ignorando la primera ejecución (overhead de las bibliotecas)
- Se probaron distintos tamaños medios y grandes (1024*2^N)
- ModernGPU no ejecuta con valores de N>10
- Thrust no ejecuta con el valor de N=30

Ejemplo práctico