Apuntes programación CUDA

Mauricio Vanzulli

May 10, 2021

1 Clase de CUDA

Características del device:

- Es un coprocesador de la CPU
- Posee una memoria DRAM
- Ejecuta muchos threads en paralelo.
- Cada mutli-procesador procesa un bloque con un programa único (kernel) en muchos hilos. Cada CUDA-core procesa muchos hilos, uno a la vez.
- Este paradigma de programación recibe la sigla SPMT (single program multiple threads).
- Cada kernel ejecuta un array de hilos, es importante el identificador de hilo respecto al dato al que se le quiere ejecutar el kernel.

Algoritmo básico de programación

1. Instrucciones en el host.
2. Enviar los datos al device.
3. Procesa en GPU
4. Recuperar los datos de la GPU.
5. Continuar procesamiento en el host.

Existe determinada jerarquía de threads. Una grilla en 3D agrupa un conjunto de bloques y dentro de cada bloque se tienen múltiples hilos también en 3D. No se puede asumir a priori que el bloque 1 se ejecute antes del 2.

Las funciones en CUDA

Dominio de funciones	*Ejecuta en:*	*Se invoca desde:*
__device__	device	device
__global__	device	host
__host__	host	hostt

Función	*Variables de entrada:*	*Proposito:*
dim3 DimGrid	(Cant_Bloq_x,Cant_Bloq_y,Cant_Bloq_z)	Crea una grilla de esas dimensio de bloques
dim3 DimBlock	(Cant_Hilos_x, Cant_Hilos_y, Cant_Hilos_z)	Crea las dimensiones de los threads en cada bloque
KernelFunc <<<DimGrid,DimBlock,SharedMemBytes>>>(...)	Input Kernel, grilla, y memoria compartida	Ejecuta el kernel en device

Se disponen de distintos espacios de memoria:

- __global__ : memoria global en el host
- __device__ : memoria global en device
- __shared__ : reside en la memoria compartida host-device (se usa para allocamiento dinámico)
- __constant__ reside en momoria cosntante del device (allocamiento estático)

Localizadores de hilos y bloques:

- threadIdx :Ubicación de ese thread y se accede con .x .y .z
- blockIdx :Ubicación del bloque en el que me encuentro y se acccede con .x .y .z
- blockDim : Tamaño del bloque (en cantidad de hilos) y se accede con .x .y .z
- gridDim :Tamaño de la grilla (en cantidad de blques)y se accede con .x .y .z

Funciones intrínsecas al GPU

- cudaDeviceSynchronize :Sincroniza todos hilos en el device y se ejecuta desde el host.
- __syncthreads :Permite sincronizar los threads de un mismo bloque.
- cudaClock y otro miden tiempos

Reservar memoria en la tarjeta y transferir datos es fundamental para comunicar las datos y sus procesamientos, esto es bastante costoso en términos computacionales. Me interesa comunicar host-cpu (se ejecutan desde el CPU):

- cudaMalloc(Puntero,Tamaño de memoria) Reserva memoria en la global memory de la GPU

- cudaFree (Puntero) libera el espacio reservado
- cudaMemcpy (Puntero de destino, puntero origen,numero de bytes a copiar, Tipo de transferencia) . Los diferentes tipos de transferencia son: Host 2 Device (cudaMemcpyHostToDevice), Device 2 Host(cudaMemcpyDeviceToHost). Además están (Host 2 Host) y Device 2 Device.
- cudaMemSet (Variable a iniciar,0 o 1, sizevector): Setea el inicio del vector en el device.

La clave es la relación entre los identificadores y los particionamientos de datos

Si en el kernel el procesamiento no depende del thread puede haber problemas de condición de carrera... para solucionar esto se usa atomicadd Esto sucede cuando mas de un hilo deben escribir a la misma entrada de memoria golbal. Osea dentro de un mismo warp múltiples hilos deben escribir en el mismo lugar, ¿cuál escribe primero?, hace falta ordenarlos serialmente. Este tipo de problemas se llama racecondition y CUDA escribe unicamente un hilo aleatorio.

2 Clase de CUDA

El acceso a memoria global es donde se encuentra el cuello de botella, el proceso más costos en términos computacionales la transferencia de datos de la CPU al device y luego del device a CPU, es importante realizar esta transferencia de forma eficiente.

Memoria Coalesced Acceso "coalesced" a memoria global: Consiste en fusionar los accesos a memorias de un warp (conjunto de hilos) para optimizar la petición a memoria. Si cada hilo necesita un dato contiguo a otro en memoria en vez de realizar dos accesos, los une en uno solo.

- El acceso a memoria global es por segmentos incluso cuando se quiere leer una palabra. A veces esos datos no son útiles y se desperdicia el ancho de banda. También sucede en CPU pero el cache y el preload vulneran estos problemas intrínsecamente."Cuando se solicita una dirección del segmento me devuelve todo el segmento"
- Los segmentos están alineados en múltiplos de 128 bytes. "Me traigo trozos o bloques de 128 bytes". Además estos segmentos están fijos.
- Es necesario utilizar Structs of Array. Osea si quiero guardar múltiples datos del mismo objeto guardo los arrays de eses datos para los múltiples entes. Como los arrays son secuenciales se puede utilizar el coalessed, contrario a lo que sucede con Array of Structs

En la memoria global queremos que cada lectura utiliza la mayor cantidad de datos en cada ejecución. En el caso de memoria compartida es deseable que cada bloque use la mayor cantidad de memoria

Memoria Compartida Un mismo bloque dispone de una memoria compartida en el multi-procesador que se ejecuta. Esto se tiene que especificar en el kernel y cuando termine de ejecutar ese bloque hay que copiarlos a la memoria global ya que sino mueren con su bloque. Esta memoria se usa como una especie de cache, permite evitar accesos no coalesced a la memoria global. Se copia de forma coalessed de la global y luego se accede de forma desordenada en la memoria compartida.

Cuando se invoca al kernel hay que hacer referencia a que cantidad de memoria compartida se quiere reservar en memoria Función¡¡Dim Grid,Dim BLoc, Tamaño de memoria compartida¿¿(output, input)

Puede existir un problema cuando en un ciclo de reloj los hilos no pegan en bancos distintos puede suceder que: dos hilos traten del leer el mismo dato de banco tenemos un bank conflict. La clave es que el indice de acceso a banco sea con modo impar en módulo 32(cantidad de bytes por banco). Resumen:

- No hay confitico y el acceso es rápido cundo:
  1. Todos los hilos del warp acceden a diferentes bancos.
  2. Todos los hilos del warp leen la misma palabra de un banco.
- El acceso es ineficiente cuando:
  1. Varios hilos del warp acceden a parlabas diferentes del mismo banco.
  2. Se produce un conflicto y se debe serializar el acceso.
  3. Se requieren tantos ciclos como el numero máximo de accesos al mismo banco.

Transferencias a memoria Para hacer una transferencia es necesario que la memoria sea no paginable (estable y no tomado el control por el SO). La memoria paginable hace funcionar la memoria virtual porque cuando se ejecuta una nueva aplicación el SO copia esos datos de memoria paginable al disco y luego vuelve a cargarlas. La transferencia a GPU debe ser no paginable ya que la CPU no debe tocarla. Cuando se ejecuta cudaMemCpy() se copia de paginable a no paginable y luego a la memoria global de la GPU. Para eso se usa cudaHostAlloc, pero este bloque la memoria no paginable.

Se puede separar el plano físico con el estado lógico. Osea hacer una única transferencia y luego particionar esos datos con la lógica.