Modello di Esecuzione CUDA - Riassunto

Slides

Indice

Introduzione al Modello di Esecuzione CUDA

Il modello di esecuzione CUDA è progettato per massimizzare il parallelismo hardware delle GPU NVIDIA. Le sue caratteristiche principali includono:

- **Astrazione GPU NVIDIA**: Mantiene concetti fondamentali invariati tra generazioni.
- **Parallelismo Massivo**: Basato sul modello SIMT (Single Instruction, Multiple Threads).
- **Ottimizzazione del codice**: Migliora throughput e accessi alla memoria.

Streaming Multiprocessor (SM)

Gli SM sono i blocchi fondamentali delle GPU NVIDIA, ognuno con:

- **CUDA Cores**: Eseguono operazioni aritmetiche e logiche.
- **Shared Memory e Registri**: Memoria veloce condivisa tra i thread di uno stesso blocco.
- **Warp Scheduler**: Gestisce i gruppi di thread chiamati warp.

Ogni SM supporta migliaia di thread grazie alla replicazione dell'architettura.

CUDA Core

Il CUDA Core è l'unità di elaborazione base, specializzata in:

- Operazioni in virgola mobile (FP32/FP64) e intere (INT).
- Tensor Core (da Volta): Accelerano calcoli di AI e HPC tramite operazioni su matrici (es. moltiplicazioni).

Architetture GPU

Evoluzione Principale:

- **Fermi (2010)**: Prima architettura per HPC; fino a 512 CUDA Cores.
- **Kepler (2012)**: Introduzione di SMX potenziati e Dynamic Parallelism.
- **Volta (2017) e oltre**: Tensor Core per AI, gestione indipendente dei thread.

Gestione dei Thread e Warp

- **Thread Block**: Gruppo di thread che collaborano tramite memoria condivisa e sincronizzazione.
- **Warp**: Gruppo di 32 thread che eseguono insieme istruzioni. La divergenza nel flusso dei thread riduce l'efficienza.

Ogni thread ha identificatori univoci (gridDim, blockIdx, blockDim, threadIdx) per accedere ai dati.

Modello di Esecuzione SIMT

Il modello SIMT combina parallelismo tra thread e approccio SIMD:

- Ogni thread possiede un proprio **Program Counter** e stato indipendente.
- I thread di un warp iniziano con lo stesso indirizzo, ma possono divergere.

Esempio di somma di array:

```
__global__ void array_sum(float *A, float *B, float *C, int N) {
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
   if (idx < N) C[idx] = A[idx] + B[idx];
}
```

Distribuzione dei Thread Block

- I blocchi di thread sono distribuiti dinamicamente agli SM dal **GigaThread Engine**.
- Ogni blocco esegue indipendentemente sugli SM disponibili, massimizzando la scalabilità.

Warp Scheduler e Ottimizzazione

Il Warp Scheduler seleziona i warp pronti per l'esecuzione, garantendo l'utilizzo ottimale delle risorse:

- **TLP (Thread Level Parallelism)**: Più warp eseguiti contemporaneamente.
- **ILP (Instruction Level Parallelism)**: Esecuzione parallela di istruzioni indipendenti.

Sincronizzazione e Divergenza

- **Sincronizzazione**: I thread di un blocco si sincronizzano tramite barriere (es. `__syncthreads()`).
- **Divergenza**: Gestita automaticamente dall'hardware, ma penalizza le prestazioni.

Parallelismo Avanzato e Tensor Core

- **Tensor Core**: Specializzati per calcoli su matrici, riducono memoria e consumo energetico.

- **CUDA Dynamic Parallelism**: Kernel che lanciano altri kernel, migliorando la flessibilità.

Consigli di Ottimizzazione

- Usare dimensioni dei blocchi multipli di 32 per evitare warp incompleti.
- Massimizzare l'occupazione ("occupancy") utilizzando risorse hardware in modo efficiente.
- Minimizzare la divergenza dei warp per migliorare le prestazioni.

Conclusione

Il modello CUDA è progettato per sfruttare il parallelismo massivo delle GPU NVIDIA, garantendo flessibilità e prestazioni elevate attraverso un'architettura scalabile e ottimizzata.