Ottimizzazioni 2

Corso di sviluppo Nvidia CUDA™

Davide Barbieri

Panoramica Lezione

- Parallelismo a livello di Thread
- Occupazione dei multiprocessori
- Latenza
- Parallelismo a livello di istruzione

Parallelismo a livello di thread

- *Ogni SM riesce a tenere attivi N warp*
  - *C.C. 1.0-1.1 → N = 24 warp = 768 thread*
  - *C.C. 1.2-1.3 → N = 32 warp = 1024 thread*
  - *C.C. 2.* → N = 48 warp = 1536 thread*

- Il *warp scheduler* esegue un *issue* di un'istruzione *pronta*
  (Le istruzioni da cui dipende sono state completate)
  - Se non trova istruzioni *pronte* resta in *idle*
  - Più *warp* sono *attivi* → più lo scheduler ha *"scelta"*
  - Altrimenti: latenza di 24 cicli

Warp 1	Warp 2	Warp 3	Warp 4	Warp 5	Warp 6
x = y + z;	x = y + z;	x = y + z;	x = y + z;	x = y + z;	x = y + z;
w = 2*x;	w = 2*x;	w = 2*x;	w = 2*x;	w = 2*x;	w = 2*x;
h = 3*w;	h = 3*w;	h = 3*w;	h = 3*w;	h = 3*w;	h = 3*w;

  - *6 warp sono già sufficienti a nascondere tutte le dipendenze RAW su c.c. 1.**

Parallelismo a livello di thread

- # Massimo di thread per blocco
  - C.C. 1.* → 512 thread
  - C.C. 2.* → 1024 thread
- # Massimo di blocchi per SM
  - 8 (qualsiasi C.C.)
- Un solo blocco non basta per occupare tutto lo SM
- Ogni blocco divide le risorse dello SM con gli altri
  - Registri da dividere tra i thread
  - Memoria Condivisa da dividere tra i blocchi
- Per garantire il massimo # di warp attivi
  - Necessario configurare il blocco di thread

Occupazione

- Definizione
  - *“Warp attivi per SM” / “Max Warp attivi per SM“*
- Fattori limitanti
  - Numero di registri per thread
  - Quantità di memoria condivisa per blocco
  - Grandezza del blocco di thread
- *Occupazione ≠ Utilizzazione*
  - *Alta occupazione non implica alte prestazioni*
  - *Alte prestazioni non implicano alta occupazione*

Fattori limitanti: Grandezza del blocco

- Ottenere il massimo numero di warp
  - Devo allocare un certo numero di blocchi
  - Ogni blocco deve avere un sufficiente numero di thread
  - Max. 8 blocchi per SM

  - c.c. 1.0-1.1 → 24 warp = 768 thread
    - 2 blocchi da 384 thread
    - 3 blocchi da 256 thread
    - 4 blocchi da 192 thread
    - 6 blocchi da 128 thread
    - 8 blocchi da 96 thread
  - c.c. 1.2-1.3 → 32 warp = 1024 thread
    - 2 blocchi da 512 thread
    - 4 blocchi da 256 thread
    - 8 blocchi da 128 thread
  - c.c. 2.* → 48 warp = 1536 thread
    - 2 blocchi da 768 thread
    - 3 blocchi da 512 thread
    - 4 blocchi di 384 thread
    - 6 blocchi di 256 thread
    - 8 blocchi da 192 thread

Fattori limitanti: Memoria Condivisa

- C.C. 1.*
  - 16 KB di Memoria Condivisa
- C.C. 2.*
  - 16 KB o 48 KB di Memoria Condivisa
- La memoria condivisa è allocata per blocco di thread
- La Memoria condivisa viene allocata a segmenti
  - 512B per c.c. 1.*
  - 128B per c.c. 2.*
- Esempio (c.c. 1.*):
  - Il blocco chiede 5 KB
    → Possono essere schedulati massimo 3 blocchi
  - Il numero effettivo di blocchi dipenderà dagli altri fattori limitanti

Fattori limitanti: Registri

- C.C. 1.0 – 1.1
  - 8192 registri a 32 bit (1024 registri per core)
  - Max occupazione (768 thread)
    - ~10 registri per thread
- C.C. 1.2 – 1.3
  - 16384 registri a 32 bit (2048 registri per core)
  - Max occupazione (1024 thread)
    - 16 registri per thread
- C.C. 2.*
  - 32768 registri a 32 bit
  - (1024 registri per core)
  - Max occupazione (1536 thread)
    - 20 registri per thread

Fattori limitanti: Registri

- R*k* : numero di registri utilizzati dal kernel
- W*k* : warps per blocco
- Il numero di registri allocato è:
  - c.c. 1.0-1.1
    - ceil(32 * R*k* * ceil(W*k*,2), *256)*
  - *c.c. 1.2-1.3*
    - *ceil(32 * Rk * ceil(Wk,2), 512)*
  - *c.c. 2.**
    - *ceil(32 * Rk, 64) * Wk*
- *Evitate di fare tutti questi calcoli:*
  - *Esistono dei tool che lo fanno per voi*

CUDA Tools

CUDA Occupancy Calculator

- Foglio *xls nell'SDK*
- Vi permette di calcolare la percentuale di occupazione

CUDA Profiler

- Applicazione multi-piattaforma
- Vi permette di conoscere l'attuale percentuale di occupazione della vostra applicazione
- Accesi coalizzati e non
- Conflitti di banco
- Registri allocati
- Quantità di memoria condivisa
- Warp divergenti

NVCC

- Parametri da passare al compilatore

Occupazione

- Normalmente vi si raccomanda:
  - Aumentate l'occupazione!
    - Aumentate il numero di thread per blocco!
    - E' la priorità per nascondere le latenze!
- I codici più veloci alla base di CUBLAS/CUFFT non massimizzano l'occupazione
  - 67% di occupazione su SGEMM 1.1
  - 33% di occupazione su SGEMM 2.1
    → 60% di prestazioni in più
- Aumentare l'occupazione non è la prima priorità
  - 100% occupazione non vuol dire 100% prestazioni
  - Per raggiungere il picco di op./s
    SP devono lavorare il più possibile

Latenza

- **Tempo che va dalla richiesta di un'operazione alla sua conclusione**
  - ~24 cicli per ALU
  - ~400 – ~800 per accessi a memoria globale
- La *pipeline* della GPU è in grado di ***nasconderla***:
  - Se ho operazioni di qualche warp da avviare, per ogni ciclo di latenza
    - Istruzioni provenienti da warp diversi (anche la stessa) ***(parallelismo a livello di thread)***
    - Istruzioni differenti dello stesso warp (purchè indipendenti) ***(parallelismo a livello di istruzione)***

Parallelismo a livello di thread

- ***Copro la latenza di un'istruzione con istruzioni appartenenti ad altri warp***

Warp 1	Warp 2	Warp 3	Warp 4
x = y + z;	x = y + z;	x = y + z;	x = y + z;

Parallelismo a livello di istruzione

- ***Copro la latenza di un'istruzione con istruzioni dello stesso warp***
- *Purchè non ci sia dipendenza RAW*

```
Warp 1
x = x + a;
y = y + a;
z = z + a;
```

L'uno non esclude l'altro

Throughput

- Numero di operazioni completate nell'unita di tempo (cicli, secondi, ecc.)
  - **Esempio: Nvidia Geforce GTX 580**
    - **512 core**
    - **1544 Mhz clock**
    - Può eseguire una FMA (1 ADD + 1 MUL = **2 FLOP**)
    - Throughput ALU → $512*1544*(10^6)*(2\ Flop)$
      - **~1581 Gflop/s**
    - Clock memoria **GDDR5**: **2004 Mhz**
      - *DDR → doppio data-rate*
    - Ampiezza interfaccia: **384 bit**
    - Throughput Memoria → $2*2004*(10^6)*(384/8\ Bytes)$
      - ~192.4 GB/s

Nascondere la latenza (1/3)

- Più nascondo la latenza, più mi avvicino al picco di performance
  - Bene avere sullo stesso multiprocessore più istruzioni indipendenti
  - Se non ci sono istruzioni disponibili → cicli a vuoto

- Quante istruzioni *Ist* posso utilizzare per nascondere la latenza L?
  - *Legge di **Little***: **N = λ * L = Throughput*Ist* * Latenza**

  - e.g. Istruzioni ADD, MUL, MAD (c.c. 1.*)/FMA (c.c. 2.*)

- c.c. 1.* → λ = ¼ (1 ist. * 1 warp / 4 cicli)
- c.c. 2.0 → λ = 1 (1 ist. * 2 warp / 2 cicli)
- c.c. 2.1 → λ = 3/2 ((2 ist. * 1 warp + 1 ist. * 1 warp) / 2 cicli)

Nascondere la latenza (2/3)

- Qual è il numero di FLOP che ogni SM deve eseguire per nascondere la latenza?

- Operazioni ALU
  - 24 cicli di latenza

- Considerazioni operazioni ALU:
  - Se ho un blocco con # thread ≥ **λ** * **L** warps
    - su (c.c. 1.*) → (¼) * 24 = 6 warps = 192 thread
    - su c.c. 2.0 → 1 * 24 = 24 warps = 768 thread
    - su c.c. 2.1 → (3/2) * 24 = 36 warps → 1152 thread

    Su GPU, la latenza è **automaticamente nascosta**
    - **In pratica (prove empiriche):**
    - **molto meno → (~576 su GF100, ~768 su GF104)**

  - Anche se ho dipendenze RAW!!
    R2 ← R1 + R3
    R4 ← R2 + R3

Nascondere la latenza (3/3)

- Favorisci entrambi i tipi di parallelismo
  - *Instruction Level Parallelism* e *Thread Level Parallelism*
  - Nonostante quel che può sembrare:
    - ILP > TLP
    - G80 con 100% prestazioni possibili (es. di kernel con sole MAD)
      - con 25% di occupazione (6 warps) senza ILP
      - con 8,3% di occupazione (2 warps) con ILP di 3 operazioni/thread
  - Indispensabile su GF104 per prestazioni ≥ 66% del picco
    - 48 core, ma solo 2 warp scheduler
      → 1/3 dei core può essere usato solo se un warp presenta ILP
- Migliore è ILP, minore è l'occupazione necessaria

Latenza di memoria

- La latenza degli accessi in memoria è più grande di quella delle operazioni ALU
  - 400 ↔ 800 cicli di latenza

- Quante operazioni di accesso a memoria riesce a gestire la GPU contemporaneamente?

  All'incirca:
  - **λ * L < 192 GB/s * (800 / 1544Mhz) < 100 KB**

- Per massimizzare l'uso della banda di memoria, dunque, è sufficiente mantenere 100 KB di accesso continui
  - Meno se il kernel ha bottleneck di tipo ALU

- Anche in questo caso, diversi modi per ottenere lo stesso obiettivo

  - Usare TLP per eseguire 100KB di accessi in parallelo continuati

  - Fate eseguire più lavoro ad ogni thread
    - Eseguire accessi da 64 o 128 bit alla volta anziché 32
    - Eseguire più accessi per thread
      - 1 float = 4B per thread → ho bisogno di 25600 thread
      - 16 float = 64B per thread → 1600 thread

  - Anticipate il più possibile gli accessi alla memoria

    - Solo le dipendenze RAW causano stallo

    - Potete coprire così gli accessi alla memoria con operazioni ALU

Registri vs Memoria Condivisa

- Che vantaggio si ha ad utilizzare i registri (rispetto a memoria condivisa)?
  - I registri sono gli unici abbastanza veloci per raggiungere il picco
  - a*b+c (float) su ogni SM:
    (G80-GT200) 8*12 Byte = 96 Byte dai registri
    (GF100) 32*12 Byte = 384 Byte dai registri
    (GF104) 48*12 Byte = 576 Byte dai registri

  - Ogni banco di memoria condivisa ha una banda di 4 Byte su 2 cicli
    La memoria condivisa riesce a fornire solo:
    (Tesla) 16 banchi * 4 Byte / 2 cicli = 32 Byte per ciclo
    (Fermi) 32 banchi * 4Byte / 2 cicli= 64 Byte per ciclo

    → throughput memoria condivisa
    - 1/3 dei registri su Tesla
    - 1/6 dei registri su GF100
    - 1/9 dei registri su GF104

- Utilizzando meno thread → più registri disponibili per ogni thread

Confronto Throughput Memorie (GF100)

- Memoria globale: **192 GB/s**
- Memoria condivisa: **1.5 TB/s** (8x memoria globale)
- Registri: **9 TB/s** (6x memoria condivisa)

- In generale:
  Usate la memoria condivisa se necessaria a diminuire gli accessi alla memoria globale o a renderli coalizzati

- Non usate la memoria condivisa se non vi serve!!
  - Stessa latenza dei registri
  - Minor throughput

- Purtroppo il trend nelle ultime architetture
  - Differenza prestazioni mem. condivisa ↔ registri aumenta
  - Minor numero di registri massimo per thread
    ~128 Tesla
    63 Fermi

Ricapitolando

- Più lavoro per ogni thread
- Minimizzate l'uso di memoria host
- Minimizzate l'uso di memoria globale
- Minimizzate l'uso di memoria condivisa
  - Massimizzate l'uso dei registri

- Massimizzate **PRIMA** l'*ILP*
- *Aumentate* **POI** *il TLP configurando il blocco senza diminuire troppo l'ILP*
  - Provate diverse grandezze del blocco
  - Non è importante avere occupazione altissima

Riferimenti

- CUDA Programming guide
- CUDA Best Practices guide
- *Vasily Volkov – Better Performance at Lower Occupancy*
  - http://nvidia.fullviewmedia.com/gtc2010/0922-a5-2238.html