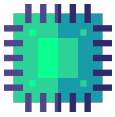

Risposte - Sistemi di Elaborazione Accelerata

Ingegneria Informatica M

Matteo Fontolan

2025/2026

Disclaimer & Info

Questo documento é una raccolta **personale** basata sulle lezioni e/o sui materiali didattici forniti dai docenti e/o raccolti negli anni dagli studenti.

> ❗ **Disclaimer**
>
> *L'autore non si assume alcuna responsabilità per eventuali errori, omissioni o inesattezze contenuti in questo documento. Il materiale è fornito «così com'è» a solo scopo di supporto allo studio.*

Se vuoi **aggiungere** materiale utile o **segnalarmi** errori o correzioni, fallo qui.

Spero che queste risorse ti siano utili.
Buono studio e in bocca al lupo! 🤞

Se ti va di offrirmi una cioccolata 🍫 puoi farlo qui.

Bento

Indice

1. MODULO 1: Architetture CPU, Memoria, Numeri Reali e FPGA

1.1. Cache e Memoria CPU

1.1.1. Architettura della Cache

La memoria cache è una memoria SRAM (Static RAM) piccola e veloce interposta tra la CPU e la memoria principale (DRAM) per sfruttare il principio di località. Esistono tre principali strategie di mappatura:

1. **Direct Mapped (Mappatura Diretta):**
   - **Funzionamento:** Ogni blocco di memoria principale può essere mappato in una sola, specifica linea di cache. L'indirizzo viene diviso in *Tag*, *Index* e *Offset*.
   - **Vantaggi:** Semplicità hardware (1 comparatore), basso consumo, velocità.
   - **Svantaggi:** Alta probabilità di *conflict miss* (collisioni) se due indirizzi mappano sulla stessa riga.
2. **Fully Associative (Completamente Associativa):**
   - **Funzionamento:** Un blocco può essere collocato in **qualsiasi** linea. Non esiste l'Index; l'indirizzo è diviso solo in *Tag* e *Offset*.
   - **Vantaggi:** Massima flessibilità, minimizza i *conflict miss*.
   - **Svantaggi:** Complessità hardware elevata. Richiede un comparatore per **ogni** linea per il confronto parallelo (consumo e latenza alti).
3. **Set-Associative (Associativa a Insiemi):**
   - **Funzionamento:** Compromesso tra le precedenti. La cache è divisa in S insiemi. Un blocco mappa su un insieme specifico (tramite *Index*), ma può occupare una qualsiasi delle N vie (*ways*) in quell'insieme.
   - **Vantaggi:** Bilancia flessibilità e semplicità.

1.1.2. Set-Associative Cache: Funzionamento

In una cache Set-Associative a N-vie, l'indirizzo fisico è suddiviso in **Tag**, **Index**, **Offset**.

1. **Index:** Seleziona il *Set* specifico tra quelli disponibili.
2. **Confronto Parallelo (Tag):** Il *Tag* dell'indirizzo richiesto viene confrontato simultaneamente con i Tag memorizzati in tutte le N vie dell'insieme selezionato (N comparatori hardware).
3. **Hit/Miss:**
   - **Hit:** Se un comparatore rileva uguaglianza (e valid bit = 1). Il dato viene estratto usando l'*Offset*.
   - **Miss:** Se nessun comparatore rileva uguaglianza.

1.1.3. Line Fill

Operazione di riempimento di un'intera linea di cache quando si verifica un miss.

- **Convenienza:** Vantaggioso per il **principio di località spaziale** (probabile richiesta di dati contigui).
- **Implementazione:** Le memorie DRAM usano il **Burst Access**. Una volta aperta una riga (**Row Open**), leggere le colonne successive è veloce. Si trasferisce un blocco intero (es. 64 byte) in un solo colpo per ammortizzare la latenza.

1.2. Rappresentazione dei Numeri Reali (Focus E5M2)

1.2.1. Standard E5M2

Formato *minifloat* a 8 bit per Deep Learning.

- **Struttura:** 1 bit Segno (S), 5 bit Esponente (E), 2 bit Mantissa (M).
- **Bias:** $2^{k-1} - 1$. Con $k = 5$, il **Bias = 15**.

Formula valore normale:

$$\text{Valore} = (-1)^S \times 2^{E-\text{ Bias}} \times \left(1 + \frac{M}{2^2}\right)$$

Esempio Conversione (Binario `0100 0110` a Decimale):
1. $S = 0$ (Positivo).
2. $E = 17 \rightarrow 17 - 15 = 2$.
3. $M = 2$ (binario `10`) $\rightarrow 1 + \left(\frac{2}{4}\right) = 1.5$.
4. Risultato: $1.5 \times 2^2 = 6$.

Numeri Subnormali: Si hanno quando $E = 0$. L'esponente implicito è $1 - \text{Bias} = -14$. Niente 1 implicito nella mantissa.

$$\text{Valore} = (-1)^S \times 2^{-14} \times \left(0 + \frac{M}{4}\right)$$

Calcoli Limite:
- **Max Subnormale ($E = 0, M = 3$):** $2^{-14} \times 0.75$.
- **Min Normale ($E = 1, M = 0$):** $2^{1-15} \times 1.0 = 2^{-14}$.

Casi Limite Esponente:
- $E = 0$: Zero (se $M = 0$) o Subnormali.
- $E = 31$: Infinito (se $M = 0$) o NaN (se $M \neq 0$).

1.2.2. Operazioni e Precisione
- **Problematiche:** Violazione proprietà associativa e distributiva. Cancellazione catastrofica («vanishing») sommando numeri con esponenti molto diversi.
- **Bit di Arrotondamento:** Guard (G), Round (R), Sticky (S). Usati per garantire l'arrotondamento corretto (es. *Round to Nearest Even*). S diventa 1 se c'è qualsiasi bit non nullo a destra.
- **Vantaggi FMA (Fused Multiply-Add):** Esegue $d = a \times b + c$ con **un solo arrotondamento** finale. Più preciso e veloce del MAC (Multiply-Accumulate).

1.3. FPGA e Accelerazione Hardware
Una FPGA è una matrice di blocchi configurabili.
- **CLB (Configurable Logic Blocks):** I mattoni base. Contengono **Slice** con:
  - **LUT (Look-Up Table):** Piccola RAM che implementa funzioni logiche combinatorie (tavola di verità).
  - **Flip-Flop:** Per logica sequenziale.
  - **Mux:** Per il routing.
- **Block RAM (BRAM):** Blocchi di RAM dedicati integrati nel chip (es. 36Kb), molto più densi della Distributed RAM (fatta con le LUT).

1.4. Ottimizzazione Reti Neurali
- **Palettizzazione (Weight Sharing):** Raggruppamento dei pesi simili tramite clustering (es. K-means).
  - **Funzionamento:** Si salvano solo i centroidi (pochi float) in una **LUT**. La matrice dei pesi diventa una matrice di **indici** (pochi bit) che puntano alla LUT, riducendo drasticamente l'occupazione.

1.5. Altro (SIMD e Sensing)
- **SIMD:** *Single Instruction Multiple Data*. Una sola istruzione controlla l'elaborazione simultanea di un vettore di dati (es. registri a 128/256 bit) in lockstep.
- **3D Sensing:**
  - **Stereo Vision:** Usa due camere e la disparità d. Profondità $Z = \frac{b \cdot f}{d}$.
  - **Time of Flight (ToF):** Misura il tempo Δt di volo della luce. Distanza $= c \cdot \Delta \frac{t}{2}$.

2. MODULO 2: CUDA (Modelli e Ottimizzazione)

2.1. Modelli Fondamentali

2.1.1. Modelli CUDA

1. **Programmazione (Host/Device):** Host (CPU) gestisce flusso e I/O; Device (GPU) esegue kernel paralleli. Memorie logicamente separate.
2. **Esecuzione:** Gerarchia Thread → Block → Grid.
3. **Memoria:** Gerarchia esposta: Registri (privati), Shared (blocco), Global (grid), Constant/Texture.

2.1.2. SIMD vs SIMT

- **SIMD:** Un thread controlla un vettore. Larghezza vettore esposta al programmatore. Divergenza gestita manualmente.
- **SIMT (NVIDIA):** Programmatore scrive codice per **singolo thread**. Hardware raggruppa thread in **Warp** (32). Divergenza gestita dall'hardware (mascheramento), permettendo flussi logici indipendenti.

2.1.3. Architettura SM

Unità costruttiva della GPU contenente: CUDA Cores (INT/FP), LD/ST Units, SFU (funzioni speciali), Register File (enorme), Shared Memory/L1 Cache, Warp Schedulers.

2.2. Modello di Esecuzione e Scheduling

2.2.1. Warp Scheduling

- **Context Switch a costo zero:** Lo stato di tutti i warp attivi è mantenuto nei registri hardware. Non serve salvare/ripristinare in RAM.
- **Latency Hiding:** Se un warp stalla (es. attesa memoria), lo scheduler seleziona istantaneamente un altro warp pronto (*Eligible*).

2.2.2. Divergenza e Sincronizzazione

- **Branch Efficiency:**

$$\text{Eff} = 100 \times \frac{\text{Branches} - \text{Divergent}}{\text{Branches}}$$

- **Independent Thread Scheduling (ITS - Volta+):** Ogni thread ha il suo PC e Stack. Permette esecuzione interlacciata e sincronizzazione fine.
- **Sincronizzazione:**
  - `__syncthreads()`: Barriera per il **blocco**.
  - `__syncwarp()`: Barriera per il **warp** (necessaria con ITS).
  - **Parallelismo Dinamico:** Kernel lancia altri kernel direttamente dal Device.

2.3. Modello di Memoria e Accessi

2.3.1. Shared Memory

- **Struttura:** On-chip, bassa latenza, 32 banchi da 4 byte.
- **Bank Conflicts:** Se più thread di un warp accedono a indirizzi diversi sullo **stesso banco**, l'accesso è serializzato.
- **Broadcast:** Se tutti accedono allo **stesso indirizzo**, non è conflitto.

2.3.2. Global Memory e Coalescing

- **Coalescing:** Quando 32 thread di un warp accedono a un blocco contiguo e allineato.
- **Effetto:** Una singola transazione (es. 128 byte) serve tutti i thread (Efficienza 100%). Accessi sparsi (stride) causano spreco di banda (molte transazioni per pochi dati).

2.3.3. Strutture Dati

- **Matrici:** Linearizzate **Row-major**. Indice: `idx = iy * width + ix`.

- **Ottimizzazione Trasposizione:** Usare **Shared Memory** come buffer (tile) per garantire lettura coalescente da Global a Shared e scrittura coalescente da Shared a Global, evitando stride ampi.

3. Tipi di Memoria Host-Device (Completamento Modulo 2 Base)

3.1. Differenze tra Pinned, Zero-Copy e Unified Memory

3.1.1. Pinned Memory (Page-Locked)

- **Definizione:** Memoria allocata sull'Host (CPU) che viene «bloccata» (locked) dal Sistema Operativo. Non può essere spostata fisicamente né scambiata su disco (paged out).
- **Vantaggi:**
  - Permette l'uso del **DMA (Direct Memory Access)** per trasferimenti diretti verso la GPU, bypassando la CPU.
  - Offre la banda di trasferimento PCIe più alta possibile.
  - È obbligatoria per eseguire trasferimenti asincroni (concurrent copy and execution).
- **Svantaggi:**
  - L'allocazione (`cudaMallocHost`) è lenta e costosa rispetto a `malloc` .
  - Riduce la memoria fisica disponibile per il sistema operativo e altri processi, rischiando di degradare le prestazioni generali del sistema se abusata.

3.1.2. Zero-Copy Memory

- **Definizione:** È una particolare configurazione della Pinned Memory che viene mappata direttamente nello spazio di indirizzamento del Device.
- **Funzionamento:** La GPU accede ai dati direttamente nella RAM della CPU tramite PCIe. Non avviene una copia esplicita iniziale; i dati viaggiano sul bus al momento dell'accesso (on-demand).
- **Vantaggi:** Utile per dati letti una sola volta o per dataset troppo grandi per la memoria della GPU. Elimina l'overhead di latenza della copia esplicita.
- **Svantaggi:** Latenza altissima per ogni accesso (deve attraversare il bus PCIe) e banda limitata rispetto alla GDDR/HBM. Sconsigliata per accessi frequenti (coalescing obbligatorio per non uccidere le performance).

3.1.3. Unified Memory (UM)

- **Definizione:** Crea uno spazio di indirizzamento virtuale unico condiviso tra CPU e GPU.
- **Vantaggi:** Semplifica drasticamente il codice (un solo puntatore).
- **Funzionamento:** Si basa sul **Page Migration Engine** (vedi sezione Advanced). I dati migrano fisicamente dove servono (località dei dati), offrendo le prestazioni della memoria locale dopo la migrazione.

3.2. Funzionamento a basso livello (Paginazione e Mapping)

- **Staging Buffer:** La memoria standard allocata con `malloc` è **Pageable** (paginabile). Il motore DMA della GPU richiede indirizzi fisici contigui e stabili per trasferire dati. Poiché il SO può spostare la memoria paginabile, il driver CUDA non può permettere alla GPU di accedervi direttamente.
- **Processo di Trasferimento (Memoria Paginabile):**
  1. Il driver alloca implicitamente un buffer temporaneo **Pinned** (Staging Buffer).
  2. La CPU copia i dati dalla memoria paginabile allo Staging Buffer.
  3. Il motore DMA trasferisce i dati dallo Staging Buffer alla GPU.
  4. Questo comporta una doppia copia e overhead di CPU. L'uso diretto di memoria Pinned evita i passaggi 1 e 2.

3.3. Performance e Metriche

3.3.1. Occupancy

Rapporto tra warp attivi e massimi supportati per SM.

$$\text{Occupancy} = \frac{\text{Active Warps}}{\text{Max Warps per SM}}$$

- **Teorica vs Effettiva:** Teorica limitata da risorse statiche (registri/shared mem). Effettiva dipende dal runtime.
- **Scopo:** Tenere occupato l'SM per il **Latency Hiding**.

3.3.2. Roofline Model

Grafico logaritmico: Asse X = Intensità Aritmetica (FLOPs/Byte), Asse Y = GFLOPS.

- **Intensità Aritmetica (AI):**

$$\frac{\text{FLOPs}}{\text{Bytes Trasferiti}}$$

.

- **Memory Bound:** AI bassa (sotto la pendenza). Limitato dalla banda. Soluzione: aumentare AI, compressione, cache.
- **Compute Bound:** AI alta (sotto il tetto piatto). Limitato dal calcolo. Soluzione: istruzioni più veloci, Tensor Cores.

4. APPROFONDIMENTI E DETTAGLI TECNICI

4.1. Approfondimento Cache e Memoria CPU

- **SRAM vs DRAM:** SRAM usa flip-flop (veloce, costosa, no refresh). DRAM usa condensatori (lenta, alta densità, refresh necessario).
- **Memory Wall:** Divario crescente tra velocità CPU (esponenziale) e memoria (lineare).
- **3 C dei Miss:**
  1. **Compulsory:** Primo accesso.
  2. **Capacity:** Cache troppo piccola.
  3. **Conflict:** Collisione su set/linea.
- **Loop Tiling:** Divide matrici in blocchi che stanno in cache per massimizzare il riutilizzo dei dati, migliorando l'ordine degli accessi (es. evitare stride elevati).

4.2. Rappresentazione e Calcolo

- **ULP (Unit in the Last Place):** Valore del bit meno significativo. La distanza tra due float consecutivi aumenta al crescere dell'esponente.
- **Rounding Modes:** «Round to Nearest Even» evita il bias statistico nelle somme lunghe (es. $1.5 \to 2$, $2.5 \to 2$).
- **BF16 (Brain Float):** Stesso esponente di FP32 (range dinamico intatto), mantissa ridotta (precisione minore). Ideale per AI.
- **Saturazione:** Evita il wrap-around (overflow riparte da zero). I valori restano al massimo (es. 255). Utile nel pixel processing.

4.3. 3D Sensing e Acquisizione

- **Rolling vs Global Shutter:** Rolling espone righe in sequenza (distorsione su oggetti veloci). Global espone tutto insieme.
- **Event Cameras:** Pixel indipendenti, asincroni, attivati solo da variazioni di luminosità. Altissimo range dinamico, micro-latenza.

4.4. FPGA

- **Workflow HLS:** C/C++ $\to$ Analisi $\to$ Scheduling $\to$ Generazione RTL. Le direttive `#pragma` guidano l'architettura (pipeline, unrolling).

4.5. CUDA Advanced

- **SM Sub-partitioning:** SM diviso in 4 SMSP, ognuno con scheduler e unità dedicate.
- **Little's Law:** Per nascondere latenza L, servono warp attivi N:

$$N_{\text{warp}} \times \text{IssueRate} \geq L$$

- **Unified Memory:** Page Migration Engine hardware sposta le pagine tra RAM e VRAM su richiesta (Page Faults).
- **Pinned Memory:** Memoria bloccata (non swappabile). La GPU può usare il DMA diretto (niente buffer intermedio nel driver).
- **Arithmetic Intensity Esempio:** Kernel somma vettoriale ($C = A + B$).
  - Dati: 2 Load + 1 Store (12 byte per float). Op: 1 Somma.
  - AI $= \frac{1}{12} = 0.083$ FLOP/Byte. Estremamente **Memory Bound**.

5. MODULO 1: Architetture, Sensing e Numeri Reali (Extra)

5.1. Approfondimento Cache e Memoria CPU

5.1.1. SRAM vs DRAM

- **SRAM (Static RAM):** L'elemento di memorizzazione è un **Latch** (flip-flop) composto da 4-6 transistor. Mantiene il dato finché c'è corrente.
  - *Caratteristiche:* Velocissima (latenza 1ns), ma bassa densità (occupa molto spazio sul chip) e costosa. Usata per Cache e Registri.
- **DRAM (Dynamic RAM):** L'elemento è un singolo **condensatore** controllato da 1 transistor. Il dato è la carica elettrica.
  - *Caratteristiche:* Il condensatore si scarica nel tempo, quindi necessita di **Refresh** periodico (lettura e riscrittura), che consuma energia e banda. Alta densità, economica, ma lenta (latenza 50ns).

5.1.2. Il problema del «Memory Wall»

È il divario crescente tra la velocità di elaborazione della CPU e la velocità di accesso alla memoria (DRAM). Storicamente, la frequenza delle CPU è cresciuta esponenzialmente (60% anno), mentre la latenza della memoria è migliorata linearmente (9% anno). Questo rende la memoria il principale collo di bottiglia (**bottleneck**) nelle prestazioni di sistema.

5.1.3. Le 3 «C» dei Cache Miss

1. **Compulsory (Cold):** Il miss avviene al primissimo accesso a un blocco. È inevitabile.
2. **Capacity:** La cache è troppo piccola per contenere l'intero *working set* (tutti i dati necessari al programma). I dati vengono espulsi e poi richiamati.
3. **Conflict:** Avviene in cache non fully-associative (Direct Mapped o Set Associative). Più dati mappano sullo stesso indice (Set), costringendo all'espulsione di un dato anche se la cache ha spazio libero in altri Set.

5.1.4. Ottimizzazione del Codice (Loop Tiling)

- **Ordine dei Loop:** In C (row-major), accedere a una matrice per righe (`a[i][j]` , `j` varia velocemente) sfrutta la località spaziale (cache hit). Accedere per colonne (`a[j][i]`) causa salti in memoria (stride) e continui cache miss.
- **Loop Tiling:** Tecnica che divide le iterazioni sui grandi array in «blocchi» (tiles) piccoli che entrano nella cache. Si completano tutte le operazioni su un tile prima di passare al successivo, massimizzando il riutilizzo dei dati caricati.

5.2. Rappresentazione e Calcolo (E5M2 e Oltre)

5.2.1. ULP (Unit in the Last Place)

È il valore del bit meno significativo della mantissa per un dato esponente. Rappresenta la distanza tra due numeri floating point consecutivi.

- **Impatto:** Poiché

$$\text{ULP} \approx 2^E$$

, l'errore assoluto (precisione) peggiora man mano che il numero cresce (l'esponente aumenta). I numeri grandi sono «più radi» dei numeri piccoli.

5.2.2. Rounding Modes: Round to Even

La modalità standard IEEE 754 è «Round to Nearest, Ties to Even». Se un numero è esattamente a metà tra due rappresentabili (es. xxx.5), si arrotonda al numero con l'ultimo bit pari (es. 1.5 → 2, 2.5 → 2).

- **Perché:** Evita il **bias statistico** (drift) che si avrebbe arrotondando sempre per eccesso (es. 0.5 → 1) in lunghe sequenze di somme.

5.2.3. Format Trade-offs (FP16, BF16, TF32)

- **BF16 (Brain Float):** Ha 8 bit di esponente (come FP32) e solo 7 di mantissa.
- **TF32:** Formato ibrido NVIDIA (19 bit totali: 8 esponente, 10 mantissa).
- **Motivo AI:** L'IA (training reti neurali) tollera bassa precisione (pochi bit mantissa) ma richiede un ampio **range dinamico** (tanti bit esponente) per rappresentare gradienti molto piccoli o molto grandi senza andare in underflow/overflow. FP16 classico ha poco range (5 bit exp) e rischia di far divergere il training.

5.2.4. Saturazione nel calcolo SIMD

Nell'aritmetica standard, un overflow di un intero a 8 bit ($255 + 1$) genera 0 (wrap-around). Nell'elaborazione immagini/pixel, questo crea artefatti visivi gravi (bianco diventa nero).

- **Saturazione:** Fissa il risultato al massimo rappresentabile ($255 + 1 = 255$). È preferibile per mantenere la coerenza visiva.

5.3. 3D Sensing e Acquisizione

5.3.1. Rolling vs Global Shutter

- **Rolling:** Espone le righe del sensore in tempi diversi sequenzialmente. Oggetti veloci o movimenti di camera causano distorsioni geometriche (skew, wobble, «jello effect»).
- **Global:** Espone tutti i pixel nello stesso istante. Elimina le distorsioni temporali.

5.3.2. Event Cameras (Sensori Neuromorfici)

Differiscono dalle camere standard perché non catturano frame a intervalli fissi. Ogni pixel è indipendente e genera un evento asincrono (x, y, t, p) solo quando rileva una variazione di luminosità (logaritmica).

- **Vantaggi:** Latenza nell'ordine dei microsecondi, nessun motion blur, Dynamic Range altissimo (>140dB), basso consumo e basso data rate (se la scena è statica).

5.3.3. Principi di Profondità

- **Stereo Vision:**

$$Z = \frac{b \cdot f}{d}$$

  . La profondità Z è inversamente proporzionale alla disparità d. Errori piccoli nella disparità causano grandi errori in profondità per oggetti lontani.
- **Time of Flight (ToF):** Emette un impulso di luce e misura il tempo di volo Δt per tornare al sensore.

$$\text{Distanza} = c \cdot \frac{\Delta t}{2}$$

  .

5.3.4. deBayering (Demosaicizzazione)

I sensori a colori usano una griglia (Bayer Pattern: RGGB) dove ogni pixel cattura solo 1 colore. Il deBayering è l'interpolazione software per stimare gli altri 2 colori per ogni pixel.

- **Metrologia:** Si preferiscono camere monocromatiche perché ogni pixel misura l'intensità reale della luce senza interpolazioni, garantendo fedeltà geometrica e radiometrica superiore (assenza di artefatti colore o sfocature da filtro).

5.4. FPGA

5.4.1. LUT come Memoria

Le **Look-Up Tables (LUT)** nei CLB normalmente memorizzano funzioni logiche. Tuttavia, essendo piccole RAM, possono essere configurate come **Distributed RAM** (o Distributed Memory). Questo permette di creare piccole memorie (es. registri, shift register, FIFO) direttamente dentro la logica, molto veloci, senza consumare le preziose Block RAM.

5.4.2. Workflow HLS (High Level Synthesis)

Trasforma codice C/C++ in una descrizione hardware (RTL - Register Transfer Level) e poi in bitstream.

- **`#pragma:`** Sono direttive date al compilatore HLS per guidare la sintesi dell'hardware. Esempi: `#pragma HLS PIPELINE` (per parallelizzare loop), `#pragma HLS UNROLL` (per replicare hardware), `#pragma HLS ARRAY_PARTITION` (per dividere memorie e aumentare la banda).

6. MODULO 2: CUDA (Extra)

6.1. Architettura SM e Scheduling

6.1.1. SM Sub-partitioning

Un SM (Streaming Multiprocessor) moderno è diviso in 4 partizioni chiamate **SMSP** (SM Sub-Partitions).

- Ogni SMSP ha il proprio **Warp Scheduler**, la propria Dispatch Unit e un set dedicato di unità di calcolo (CUDA Cores) e Register File.
- I warp vengono assegnati staticamente a un SMSP e restano lì fino al termine.

6.1.2. Independent Thread Scheduling (ITS) - Volta+

- **Pre-Volta:** Esisteva un solo Program Counter (PC) e Stack per l'intero Warp. Se i thread divergevano, si eseguiva un ramo (es. `if`) disabilitando gli altri thread, poi l'altro (es. `else`).
- **ITS (Volta e succ.):** Ogni **singolo thread** mantiene il proprio stato di esecuzione (PC e Stack), pur condividendo le unità di calcolo.
- **Vantaggio:** Permette l'esecuzione interlacciata di thread divergenti e la sincronizzazione a grana fine. Risolve i deadlock in algoritmi (es. «starvation-free») che richiedono che i thread si scambino dati in loop divergenti, cosa impossibile con il vecchio modello lock-step rigido.

6.1.3. Costo dei Registri (Resource Partitioning)

I registri sono la risorsa più veloce ma limitata (es. 64K registri a 32-bit per SM).

- Se un kernel usa troppi registri per thread, l'SM può ospitare meno warp contemporaneamente.
- **Effetto:** Questo riduce l'**Occupancy** e la capacità di fare *Latency Hiding*. Se i registri non bastano, avviene il *Register Spilling* in memoria locale (lenta), crollando le performance.

6.2. Modello SIMT e Warp

6.2.1. Little's Law

$$N \text{ warp} = \text{Latenza} \times \text{Throughput}$$

Serve per stimare quanti warp attivi servono per nascondere la latenza.

- **Esempio:** Se la latenza memoria è 500 cicli e la GPU può emettere 1 warp/ciclo, servono 500 warp attivi (o istruzioni indipendenti) per tenere l'hardware occupato durante l'attesa.

6.2.2. Branch Efficiency

Se un warp diverge in 2 rami (es. `if-else`) di lunghezza N, l'esecuzione diventa seriale: tempo totale $2N$.

- **Efficienza:** I thread utili per ciclo sono la metà (16 su 32). L'efficienza è il 50%.

6.2.3. Warp Shuffle/Sync e `___syncwarp()`

- **Shuffle:** Permette ai thread di un warp di scambiarsi dati direttamente (registri) senza passare dalla Shared Memory.
- `__syncwarp()` : Nelle architetture post-Volta (con ITS), i thread non marciano più necessariamente in *lock-step* implicito. Se un algoritmo si basa sul fatto che i thread di un warp siano sincronizzati (es. per scambiare dati), è obbligatorio usare `__syncwarp()` per forzare la barriera intra-warp.

6.3. Memoria e Trasferimenti (Advanced)

6.3.1. PCIe Bottleneck

Il bus PCIe ha una latenza di setup fissa per ogni transazione e una banda limitata (molto inferiore alla VRAM).

- Inviare tanti piccoli pacchetti paga il costo di latenza molte volte.
- **Aggregazione:** Unire i dati in un unico grande buffer paga la latenza una volta sola e permette di saturare la banda passante del bus (Throughput massimo).

6.3.2. Unified Memory (UM) e Page Migration

- **Page Migration Engine:** Hardware dedicato che gestisce i Page Fault sulla GPU.
- **Pre-Pascal:** La Unified Memory era gestita via software/driver (spesso pinned memory su host o copie al lancio del kernel). Se la GPU accedeva a memoria non residente, il kernel falliva o era lentissimo.
- **Post-Pascal:** Supporto hardware al **Page Faulting**. Se la GPU tocca una pagina residente su CPU, il motore hardware mette in pausa il warp, migra la pagina via PCIe nella VRAM, aggiorna la tabella delle pagine e riprende l'esecuzione. Permette di allocare più memoria della VRAM disponibile (oversubscription).

6.3.3. Pinned Memory (Staging)

Vedi sezione iniziale «Tipi di Memoria». Concetto chiave: il driver non può fare DMA da memoria paginabile perché l'indirizzo fisico può cambiare. Deve copiare in un buffer **interno** (staging) bloccato. Usare memoria già Pinned evita questa «copia nascosta».

6.3.4. Bank Conflicts

- **Regola:** Si ha conflitto se thread diversi accedono a indirizzi diversi che cadono nello **stesso banco** (modulo 32).
- **Eccezione Broadcast:** Se tutti i 32 thread (o un sottoinsieme) accedono allo **stesso identico indirizzo**, non c'è conflitto. L'hardware legge il dato una volta e lo invia a tutti (Broadcast).

6.4. Metriche e Profiling

6.4.1. Nsight Systems vs Nsight Compute

- **Nsight Systems:** Profiler a livello di sistema/timeline. Si usa per vedere **quando** avvengono i trasferimenti, le copie, i lanci dei kernel e se ci sono gap (GPU idle) o sovrapposizioni CPU-GPU.
- **Nsight Compute:** Profiler a livello di singolo kernel. Si usa per analizzare **perché** un kernel è lento: stalli delle pipeline, **bank conflicts**, uso registri, throughput delle istruzioni.

6.4.2. Arithmetic Intensity

$$\text{AI} = \frac{\text{FLOPs}}{\text{Bytes (DRAM Traffic)}}$$

- **Kernel Somma Vettoriale ($C = A + B$):**
  - Operazioni: 1 somma (1 FLOP).
  - Memoria: Legge A (4B), Legge B (4B), Scrive C (4B) = 12 Bytes.
  - $\text{AI} = \frac{1}{12} = 0.083$.
- **Analisi:** Il valore è bassissimo. La GPU passa tutto il tempo ad aspettare i dati dalla memoria; le unità di calcolo sono scariche. È *Memory Bound*.

6.4.3. Occupancy 100%: Non sempre ideale

Un'occupancy del 100% non è garanzia di massime performance perché:

1. Per raggiungerla, potrei dover limitare i registri per thread, causando **Register Spilling** (lento).
2. Potrebbe esserci saturazione di altre risorse (es. banda Shared Memory).
3. A volte basta un'occupancy minore (es. 50-60%) per nascondere completamente la latenza, se c'è abbastanza ILP (Instruction Level Parallelism) dentro ogni thread. Aumentare oltre non dà benefici e aumenta la contesa per la cache L1.