Document de Briefing Detaliat: Programare Paralelă și Arhitecturi de Calcul

Sinteză Concepte Cheie

16 iunie 2025

Rezumat

Acest briefing sintetizează concepte cheie din programarea paralelă, arhitecturile de calcul, modelele de performanță și implementările practice, cu un accent deosebit pe CUDA de la NVIDIA și pe modelele teoretice precum BSP și PRAM.

Cuprins

1 Arhitectura și Programarea GPU cu NVIDIA CUDA

NVIDIA a revoluționat calculul de înaltă performanță (HPC) prin introducerea platformei Tesla și a arhitecturii de calcul unificat a dispozitivului (CUDA). Această platformă a permis programatorilor să utilizeze eficient paralelismul masiv al GPU-urilor pentru scopuri generale de calcul.

1.1 NVIDIA Tesla Platformă și Specificații GPU

1.1.1 Piața HPC

Seria NVIDIA Tesla a fost prima platformă destinată pieței calculului de înaltă performanță, urmată de serii precum Kepler, Maxwell, Pascal, Turing și Volta.

1.1.2 Specificații GTX 480

Un GPU emblematic la vremea sa, GTX 480 demonstra capacități impresionante de procesare paralelă:

- **Tranzistori:** 3 miliarde
- **Nuclee de calcul:** 480
- **Frecvență:** 1.401 GHz
- **Performanță în virgulă mobilă (precizie simplă):** 1.35 TFLOPs
- **Performanță în virgulă mobilă (precizie dublă):** 168 GFLOPs
- **Memorie RAM internă (VRAM):** 1.5 GB DDR5, cu o viteză de 177.4 GB/sec, semnificativ mai rapidă decât RAM-ul obișnuit (21-25 GB/sec)
- **Interfață:** Slot PCIe (până la 8 GB/sec per placă GPU)
- **Consum de putere:** 250 W

1.1.3 Modelul de Programare SIMT

Single-Instruction Multiple Thread (SIMT): Un principiu fundamental al arhitecturii GPU, permițând executarea aceleiași instrucțiuni pe multiple fire de execuție în paralel, ceea ce este esențial pentru procesarea datelor în masă.

1.2 Compute Unified Device Architecture (CUDA)

CUDA este o extensie a limbajului C/C++ care permite programatorilor să acceseze și să gestioneze direct resursele GPU.

1.2.1 Obiectiv și Principii

- **Obiectiv:** "Expose GPU parallelism for general-purpose computing" și "Retain performance"
- **Programare eterogenă:** CUDA C/C++ permite programarea sistemelor eterogene (CPU + GPU)

1.2.2 Cuvinte Cheie și Concepte

- Funcțiile executate pe GPU sunt marcate cu cuvântul cheie `__global__`
- "CUDA C/C++ keyword global indicates a function that: Runs on the device, Is called from host code"
- Apelarea funcțiilor kernel se face prin "triple angle brackets" (`<<<...>>>`), numite și "kernel launch"

1.2.3 Compilarea și Execuția

- **NVCC:** Compilatorul NVIDIA (nvcc) separă codul sursă în componente pentru gazdă și dispozitiv
- **Parallel Thread eXecution (PTX):** O mașină virtuală și un ISA pentru GPU, servind ca un cod intermediar generic

1.2.4 Tipuri de Memorie CUDA

Global memory: Alocată prin `cudaMalloc`, este mare, dar lentă (deși are cache). Este vizibilă tuturor firelor de execuție și blocurilor.

Shared memory: Declarată cu `__shared__`, este o memorie extrem de rapidă, pe cip, gestionată de utilizator. "Extremely fast on-chip memory, user-managed".

1.2.5 Sincronizare și Operații Atomice

- **Sincronizare:** `__syncthreads()` sincronizează toate firele de execuție dintr-un bloc. "Used to prevent RAW / WAR / WAW hazards"
- **Atomic Operations:** "An atomic operation guarantees that only a single thread has access to a piece of memory while an operation completes." Sunt mai lente decât operațiile normale: "Atomics are slower than normal load/store"

1.3 Modelul Ierarhic al Firelor de Execuție CUDA

Programarea CUDA se bazează pe o ierarhie de fire de execuție organizate în blocuri și grile:

Fire de execuție (Threads): Unități fundamentale de execuție, care pot comunica și se pot sincroniza în cadrul unui bloc. "Unlike parallel blocks, threads have mechanisms to: Communicate, Synchronize"

Blocuri de fire de execuție (Blocks): Un grup de fire de execuție care rulează împreună și pot accesa memoria partajată și se pot sincroniza

Grilă (Grid): O colecție de blocuri

1.3.1 Adresarea Elementelor

Adresarea elementelor într-un spațiu de date multi-dimensional se realizează prin:

```
int index_x = threadIdx.x + blockIdx.x * blockDim.x;
int index_y = threadIdx.y + blockIdx.y * blockDim.y;
```

2 Modele de Calcul Paralel

Două modele teoretice importante pentru înțelegerea și analiza algoritmilor paraleli sunt PRAM (Parallel Random Access Machine) și BSP (Bulk Synchronous Parallel).

2.1 PRAM (Parallel Random Access Machine)

PRAM este un model idealizat de calcul paralel cu memorie partajată.

2.1.1 Arhitectură

- **Structură:** "n RAM processors connected to a common memory of m cells"
- **Memorie:** Accesul la orice celulă de memorie partajată se face în timp unitar
- **Execuție sincronizată:** "Instructions are synchronized across the processors"

2.1.2 Faze de Execuție

O instrucțiune PRAM se execută în trei faze:

1. Citire din memorie partajată
2. Calcul local
3. Scriere în memorie partajată

2.1.3 Variante PRAM

EREW: Exclusive Read Exclusive Write - Fiecare celulă poate fi citită sau scrisă de cel mult un procesor

CREW: Concurrent Read Exclusive Write - Mai multe procesoare pot citi aceeași celulă concurent

CRCW: Concurrent Read Concurrent Write - Citiri și scrieri concurente sunt permise

2.1.4 Teorema lui Brent

Brent's Theorem: "Let N be a computational network with n interior nodes and depth d, and bounded fan-in. Then the computations performed by N can be carried out by a CREW-PRAM computer with p processors in time O(n/p+d)."

2.2 BSP (Bulk Synchronous Parallel)

BSP este un model de calcul paralel cu granularitate grosieră, axat pe superpași de sincronizare.

2.2.1 Structură

Superpași: "Sequential composition of 'supersteps'." Fiecare superpas este compus din:

1. **Calcul local:** Procesoarele execută operații locale
2. **Comunicare:** Procesoarele pot iniția schimburi de date
3. **Sincronizare cu barieră:** "A mechanism for the efficient barrier synchronization for all or a subset of the processes"

2.2.2 Costul unui Superpas

$$\text{Cost} = \max(w_i) + \max(h_i \cdot g) + l$$

unde:

- w_i = lucrul local
- h_i = numărul maxim de mesaje de intrare/ieșire per procesor
- g = permeabilitatea rețelei la trafic continuu
- l = timpul necesar pentru sincronizare

2.2.3 Comunicare și Localitate

- **Comunicare:** BSP tratează comunicația "en masse" (în bloc)
- **h-relation:** "If the maximum number of incoming or outgoing messages per processor is h, then such a communication pattern is called an h-relation"
- **Locality:** "Renounces locality as a performance optimization"

2.2.4 Relația cu PRAM

"BSP can be regarded as a generalization of the PRAM model." Dacă $g = 1$ (cost de comunicare zero), un computer BSP se poate comporta ca un PRAM idealizat.

2.3 LogP Model

LogP este un alt model de performanță pentru mașinile paralele, care ia în considerare patru parametri cheie:

L (Latency): Întârzierea maximă pentru trimiterea unui mesaj

o (Overhead): Timpul petrecut de procesor în procesarea unui mesaj

g (Gap): Intervalul minim de timp între trimiterile succesive

P (Processors): Numărul de procesoare

Relația cu BSP: "LogP + barriers - overhead = BSP"

3 Metricile de Performanță și Scalabilitate

Evaluarea eficienței unui algoritm paralel pe o arhitectură dată necesită metrici specifice.

3.1 Metricile Standard de Performanță

Timp Serial (T_S): Timpul de execuție al celui mai rapid algoritm secvențial cunoscut

Timp Paralel (T_P): Timpul de execuție al algoritmului paralel pe p procesoare

Speedup (S): $S(n) = T_S/T_P$ - Exemplu: adunarea n numere cu p=n procesoare are $S = \Theta(n/\log n)$

Eficiență (E): $E = T_S/(p \cdot T_P)$ - Pentru sisteme cu cost optim, $E = O(1)$

Cost (Work): Produsul dintre numărul de procesoare și timpul paralel ($p \cdot T_P$)

Un algoritm este **cost-optimal** dacă costul său paralel este asimptotic egal cu timpul serial al celui mai rapid algoritm serial.

3.2 Granularitatea (Granularity)

- "Granularity measures the amount of computation in relation to communication"
- Poate fi aproximată ca "ratio of computation to the amount of communication"
- Construirea granularității implică gruparea sarcinilor pentru a reduce overhead-ul de comunicare

3.3 Isoeficiența (Isoefficiency)

O funcție care descrie modul în care dimensiunea problemei (W) trebuie să crească în raport cu numărul de procesoare (p) pentru a menține o eficiență fixă.

Exemplu: "The isoefficiency function f(p) of this parallel system is $\Theta(p \log p)$" pentru adunarea numerelor.

3.4 Alte Metrici de Scalabilitate

- **Fracția Serială (f):** Derivată din Legea lui Amdahl, cuantifică proporția de cod care rămâne serial
- **Scalabilitatea Isospeed:** Definește un sistem ca scalabil dacă viteza medie pe unitatea de procesor poate rămâne constantă
- **Scalabilitatea Isospeed-e:** Introduce conceptul de "viteză marcată" pentru medii eterogene

4 Algoritmi Paraleli pentru Operații cu Matrici

Matricele oferă un caz de studiu excelent pentru aplicarea principiilor de programare paralelă și analiză a performanței.

4.1 Descompuneri și Distribuția Datelor

Dezcompunerea în fâșii pe coloane: Matricea A este împărțită pe coloane, iar vectorii b și c sunt distribuiți pe elemente sau blocuri

Dezcompunerea 2D pe blocuri: Matricea este împărțită în blocuri 2D, mai eficientă pentru înmulțirea matricilor

4.2 Operații Comune și Comunicare

Replicate block vector: Transformă un vector dintr-o distribuție pe blocuri într-o distribuție replicată, utilizând `MPI_Allgatherv`

MPI_Alltoallv: "MPI_Alltoallv() is used to do this all-to-all exchange" - funcție esențială pentru schimbul complex de date

4.3 Algoritmi de Înmulțire a Matricilor

4.3.1 Înmulțirea Simplă Paralelă

- Împarte matricile A și B în p blocuri pătrate
- Fiecare proces $P_{i,j}$ calculează blocul $C_{i,j}$
- Necesită difuzare de tip "all-to-all"
- Timpul paralel: $T_p = \frac{n^3}{p^{3/2}} + t_s \log(\sqrt{p}) + t_w \frac{n^2}{p}$

4.3.2 Cannon's Algorithm

- Algoritm pentru înmulțirea matricilor care minimizează comunicarea
- Utilizează aliniere inițială a blocurilor și cicluri de calcul și deplasări
- "Shift $A_{i,j}$ one step left (with wraparound) and $B_{i,j}$ one step up (with wraparound)"

- Utilizează `MPI_Sendrecv_replace` pentru deplasări eficiente
- Timpul paralel: $T_p = \frac{n^3}{p} + 2\sqrt{p} \cdot t_s + 2t_w \frac{n^2}{\sqrt{p}}$
- Superior în complexitatea comunicării (reduceri de $\sqrt{p}$ în loc de p)