Algoritmi Paraleli si distribuiti

Dijkstra

Rezultate si concluzii

24 ianuarie 2022

Dinu Ion George
Calculatoare Romana Anul III

1 Introducere. Motivatie.

Principala idee a acestui proiect a fost ca plecand de la algoritmul clasic al lui Dijkstra pentru gasirea celor mai scrute cai intre nodurile unui graf, sa proiectam un algoritm paralel mai eficient din punct de vedere al timpului de executie. Pentru a ajunge la o accelerare notabila, nu a fost suficienta doar elaborarea algoritmului, dar si alegerea unei tehnologii pentru a demonstra cu adevarat eficienta acestuia. Astfel in urmatoarele pagini vom prezenta rezultatele experimentelor si vom alege in final cea mai buna implementare.

Testele algoritmilor au fost executate pe urmatorul sitem:

- SO: CUDA/STLParallel - Windows 10; C/PThreads - KaliLinux2022.1
- Procesor: Intel(R)Core(TM)i5-8265U CPU@1.60GHz
- RAM: 8.00 GB
- System type: 64-bit operating system, x64-based processor
- GPU: NVIDIA GeForce MX110 2GB 64 Bit @ 1.8 GHz 28 nm

2 Rezultate.

Observatii

Rezultate urmatoare au fost colectate in urma a 8 teste folosind grafuri cu un numar de noduri intre 1024 si 131072, iar numarul de muchii intre 1032 si 17177532. (Din cauza limitarilor hardware nu s-au putut genera date de intrare mai mari, dar au fost suficiente pentru a demonstra eficienta algoritmilor.) Datele de intrare au fost generate aleator. Deoarece in urma a mai multor exeprimente am observat ca algoritmii paralelizati au dat rezultate mai bune pe grafuri rare, am ales ca numarul de muchii ce pleaca dintr-un nod sa fie cat mai mic cu putinta. Astfel grafurile, au fost generate astfel incat dintr-un nod, va exista o probabilitate de 0.001% sa porneasca o muchie. Aceasta proprietate ne asigura o complexitate mai mare a datelor de intrare in cazul in care dimensiunea acestora este foarte mare.

De altfel, trebuie mentionat ca metodele de paralelizare implementate difera, in functie de tehnologia folosita. Pentru demonstrarea eficientei algoritmului, am ales implementarea a 2 variante seriale, si 3 variante paralele. Implementarile prin CUDA si PThreads utilizeaza un algoritm diferit de algoritmul clasic al lui Dijkstra, dar asa cum o sa vedem ulterior, acestea dau rezultate promitatoare.

Primele rezultate

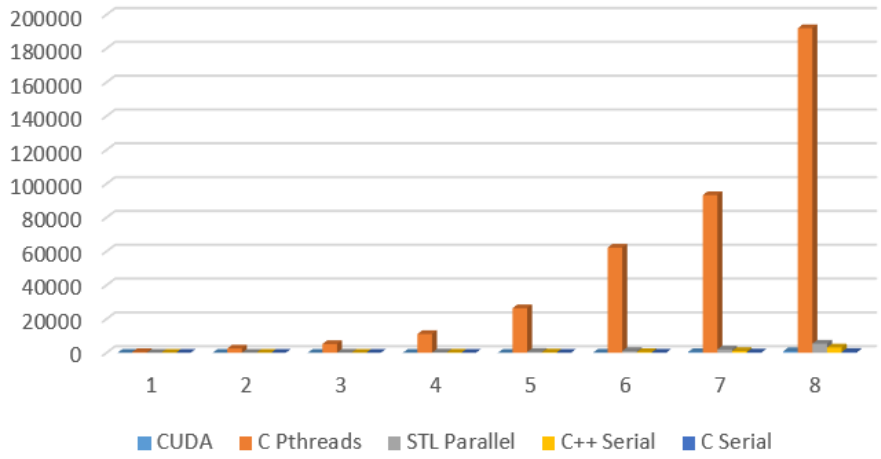

Figure 1: Optimizarea slaba a PThreads

Se observa faptul ca desi implementarea algoritmului este aceeasi ca in cazul CUDA, varainta folosind PThreads ofera cele mai slabe rezultate, motivul fiind implementarea neoptima a mecanizmelor de sincronizare, si folosire a thread-urilor

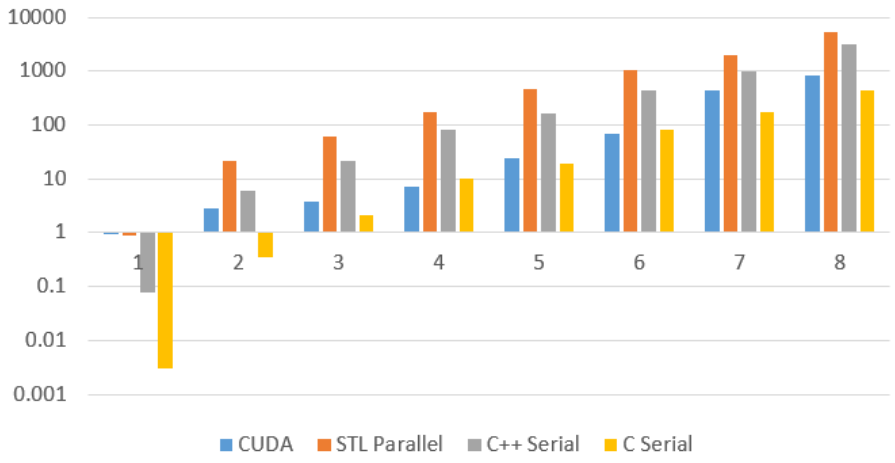

Figure 2: Analiza rezultatelor

In continuare, putem observa acelasi fenomen ca in cazul anterior. In acest caz, putem considera implementarea in C++ folosind Stl, si implementarea in C++ folosind Stl Parallel. Desi in teorie, implementarea in paralel ar trebui sa fie mai optima din punct de vedere al timpului de executie, in practica acest fenomen nu se intampla. De altfel putem observa din graficul precedent faptul ca timpul de executie in ambele cazuri are o crestere liniara, iar diferenta dintre timpii de executie, scade odata cu cresterea datelor de intrare. Pornind de la acesta ipoteza, putem presupune ca pentru un set date de intrare mult mai complexe decat cele actuale, varianta folosind STL Parallel va oferi timpi de executie mai buni decat cei oferiti de varianta seriala, a akgoritmului in C++, dar va fi departe de a depasi varianta in CUDA, sau varianta seriala in C.

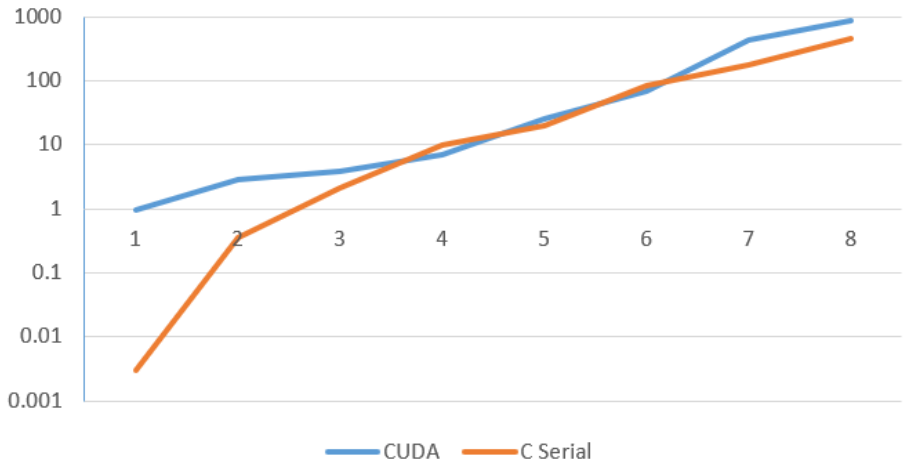

Figure 3: CUDA vs C

Asa cum am observat din graficile anterioare, implementarile cele mai optime au fost: implementare seriala din C, si implementarea paralelizata in CUDA C. Acum ne mai ramane o singura intrebare: Are sanse algoritmul paralel din CUDA, sa depaseasca varianta seriala in C a algoritmului lui Dijkstra ? Raspunsul la aceasta intrebare este mai complex decat pare. Pentru reprezentarea rezultatelor de mai sus, am folosit mai mutle seturi de date de intrare, iar in urma rezultatelor am observat ca varianta seriala este mult mai optima, folosita pentru grafuri dense. Dar, am observat si faptul ca timpul de executie al variantei din CUDA, ramane aproximativ constant, chiar depasind in cazul grafurilor rare, in cele mai multe cazuri varianta seriala. De altfel un alt element ce influenteaza executia algoritmilor reprezinta numarul de noduri al grafurilor. Din graficul de mai sus se observa ca performanta algoritmului din CUDA, apare doar de la testul cu numarul 3 unde dimensiunea unui graf va depasi 8000 de noduri.

Analizand graficul de mai sus, putem observa ca timpul de executie al algoritmului din CUDA are o tendinta descrescatoare in comparatie cu timpul de executie al variantei seriale.

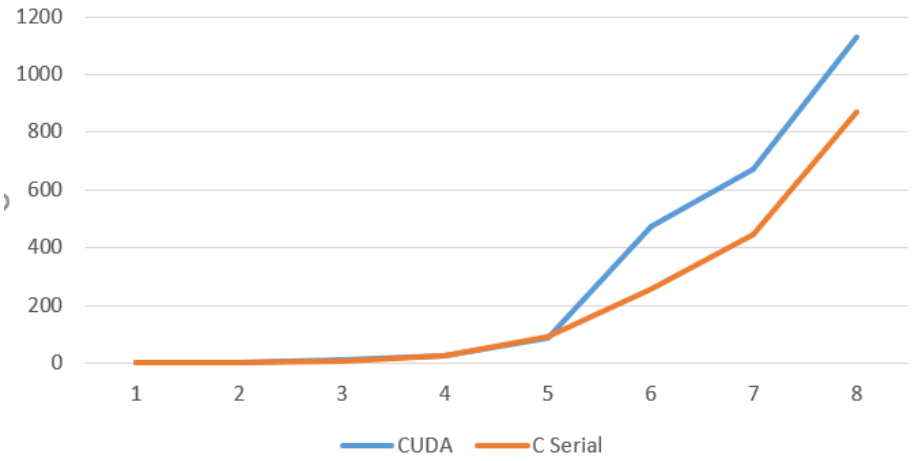

Figure 4: CUDA vs C - grafuri dense

In cazul grafurilor dense, putem observa ca timpul de executie al celor doi algoritmi pana la testul cu numarul 5 este aproximativ identic, dupa care varianta din CUDA incepe sa consume mai mult. Motivul din spatele acestei cresteri o reprezinta sincronizarile efectuate. Cu cat numarul de muchii va fi mai mare, cu atat numarul de sicronizari pe care va trebui sa le realizeze algoritmul va fi mai mare, ceea ce reprezinta un consumator de timp.

Desi exista o diferenta de timp, intre cele doua implementari, asa cum am precizat anterior, se observa o tendinta a timpului de executie al variantei implementate in CUDA de a scadea, odata cu cresterea dimensiunii datelor de intrare. De altfel trebuie precizat faptul ca hardware-ul in cazul de fata poate reprezenta o influenta destul mare asupra algoritmului din CUDA. Diferenta dintre timpi masurati este foarte mica, iar in cazul variantei in CUDA, se pot inca implementa anumite optimizari pentru accelerarea algoritmului. Cu cat GPU-ul pe care ruleaza algoritmul va fi mai performant, cu atat vom avea rezultate mai bune.

Tinand cont de datele prezentate mai sus, dar si de mentiunile legate de platforma hardware pe care sunt rulati algoritmii, consider ca varianta CUDA, are potentialul de a depasi din punct de vedere al performantei, varianta seriala, atat in cazul grafurilor rare cat si dense.

3 Concluzii

Paralelizarea unui algoritm neregulat, foarte sincronizat cu sarcini mici, reprezinta una dintre cele mai grele probleme paralele pe care le putem intalnii. Echipe de cercetare se pot dedica unor astfel de sarcini si pot obtine acceleratii limitate, sau deloc. Adesea astfel de algoritmi functioneaza numai pe arhitecturi specifice, care sunt adaptate pentru paralelizare, iar costurile pot deveni foarte ridicate.

Un astfel de algoritm necesita mult timp și efort pentru a obtine rezultate, ce pot fi luate in considerare.

References

[1] https://github.com/Pithikos/C-Thread-Pool

[2] Accelerating large graph algorithms on the GPU using CUDA Pawan Harish and P. J. Narayanan

[3] CUDA Solutions for the SSSP Problem Pedro J. Martín, Roberto Torres, and Antonio Gavilanes