Multiprocesorski sistemi

Domaći zadatak 4
CUDA – osnove
(10 poena)

Uvod

Cilj zadatka je da studente obuči da samostalno razvijaju osnovne CUDA programe.

Podešavanje okruženja

Detaljna uputstva za instaliranje, podešavanje i prvo izvršavanje CUDA programa se mogu naći na adresi http://developer.nvidia.com/nvidia-gpu-computing-documentation ili na sajtu predmeta pod nazivom CUDA Getting Started Guide (Windows) ili CUDA Getting Started Guide (Linux) u zavisnosti koji operativni sistem se koristi. Po tom uputstvu podesiti okruženje za razvoj i kontrolisano izvršavanje (engl. debugging) CUDA programa na lokalnom računaru. Alternativno, koristiti CUDA (**`nvcc`**) na računaru **`rtidev5.etf.rs`**. Prevodilac se nalazi u direktorijumu: **`/usr/local/cuda/bin/`**.

Zadaci

Svi programi treba da koriste GPU za bilo koju obradu. Smatrati da je broj GPU niti na nivou jednog bloka niti određen konstantom **`NUM_OF_GPU_THREADS`**, čija je vrednost za sve zadatke 256. Obezbediti da niti koje u nekom koraku nemaju posla na korektan način stignu do kraja tela CUDA jezgra.

Kod zadataka gde je to zahtevano, korisnik zadaje samo dimenzije nizova/matrica, a sve potrebne ulazne podatke generisati u operativnoj memoriji uz pomoć generatora slučajnih brojeva iz biblioteke jezika C, a zatim prebaciti u GPU memoriju. Generisani brojevi treba da budu odgovarajućeg tipa u opsegu od **`-MAX`** do **`+MAX`**, gde **`MAX`** ima vrednost 1024. Za sve zadatke je potrebno napisati ili iskoristiti zadatu sekvencijalnu (CPU) implementaciju odgovarajućeg problema koja će biti korišćena kao referentna (*gold*) implementacija prilikom testiranja programa.

Svaki program treba da:

- Generiše ili koristi već obezbeđene ulazne test primere.
- Kopira test primere u GPU memoriju i rezultat iz GPU memorije.
- Izvrši CUDA jezgro nad zadatim test primerom.
- Izvrši sekvencijalnu implementaciju nad zadatim test primerom.
- Ispiše vreme izvršavanja CUDA i sekvencijalne implementacije problema.
- Uporedi rezultat CUDA i sekvencijalne implementacije problema.
- Ispiše "**`Test PASSED`**" ili "**`Test FAILED`**" u zavisnosti da li se rezultat izvršavanja CUDA implementacije podudara sa rezultatom izvršavanja sekvencijalne implementacije.

Kod zadataka koji koriste realne tipove (**`float`**, **`double`**) tolerisati maksimalno odsupanje od **`±ACCURACY`** prilikom poređenja rezultata CPU i GPU implementacije. Smatrati da konstanta **`ACCURACY`** ima vrednost 0.01. **Prilikom rešavanja zadataka voditi računa da se postigne maksimalni mogući paralelizam.** Dozvoljeno je ograničeno preuređivanje dostupnih sekvencijalnih implementacija prilikom paralelizacije. **Ukoliko u nekom zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku i da nastavi da izgrađuje svoje rešenje na temeljima uvedene pretpostavke.**

Dostupne sekvencijalne implementacije se nalaze u arhivi **`MPS_DZ4_CUDA.zip`** ili **`MPS_DZ4_CUDA.tar.bz2`** koje se mogu preuzeti na adresi http://mups.etf.rs/dz/2015-2016/. Na **`rtidev5.etf.rs`** računaru arhiva se može dohvatiti i raspakovati sledećim komandama:

Dohvatanje: **`wget http://mups.etf.rs/dz/2015-2016/MPS_DZ4_CUDA.tar.bz2`**

Raspakivanje: **`tar xjvf MPS_DZ4_CUDA.tar.bz2`**

1. Paralelizovati program koji vrši računanje aditivne perzistencije za svaki element niza celih brojeva. Aditivna perzistencija nekog celog broja je jednaka broju koraka koji je potreban da se početna vrednost tog broja svede na jednocifren broj uzastopnim računanjem zbira cifara (http://en.wikipedia.org/wiki/Persistence_of_a_number). Sekvencijalni program se nalazi u datoteci **`persistence.c`** u arhivi koja je priložena uz ovaj dokument.
2. Izmeniti prethodni program tako da se umesto niza celih brojeva koristi dvodimenzionalna matrica celih brojeva proizvoljnih dimenzija. Prilikom zadavanja izvršne konfiguracije jezgra, koristiti 2D rešetku (*grid*).
3. Paralelizovati program koji vrši obilazak grafa po širini. Program se nalazi u datoteci **`bfs.cpp`** u arhivi koja je priložena uz ovaj dokument. Ulazni test primeri se nalaze u direktorijumu **`data`**, a način pokretanja programa u datoteci **`run`**.
4. Paralelizovati program koji rešava *pathfinder* problem. *Pathfinder* problem koristi dinamičko programiranje da pronađe put sa najmanjom akumuliranom težinom u 2D mreži (matrici). Put započinje od početne i ide do krajnje vrste matrice krećući se preko elemenata sa najmanjom težinom. U svakom koraku, put se pomera ili pravo ili dijagonalno u odnosu na trenutni element. Program se nalazi u datoteci **`pathfinder.cpp`** u arhivi koja je priložena uz ovaj dokument. Prilikom paralelizacije, obratiti pažnju na zavisnosti koje postoje između iteracija petlje. Za izradu rešenja koristiti deljenu memoriju za smeštanje podataka. Program testirati sa parametrima koji su dati u datoteci **`run`**.
5. Paralelizovati program koji vrši *k-means* klasterizaciju podataka. Klasterizacija metodom *k-srednjih vrednosti* (eng. *k-means clustering*) je metod koji particioniše *n* objekata u *k* klastera u kojem svaki objekat pripada klasteru sa najbližom srednjom vrednošću. Objekat se sastoji od niza vrednosti - osobina (eng. *features*). Podelom objekata u podklastere, algoritam predstavlja sve objekte pomoću njihovi srednjih vrednosti (tzv. centroida podklastera). Inicijalni centroid za svaki podklaster se bira ili nasumično ili pomoću odgovarajuće heuristike. U svakoj iteratiji, algoritam pridružuje svaki objekat najbližem centroidu na osnovu definisane metrike. Novi centroidi za sledeću iteraciju se izračunavaju usrednjavanjem svih objekata unutar podklastera. Algoritam se izvršava sve dok se makar jedan objekat pomera iz jednog u drugi podklaster. Program se nalazi u direktorijumu **`kmeans`** u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke **`kmeans.c`**, **`cluster.c`** i **`kmeans_clustering.c`**. Ulazni test primeri se nalaze u direktorijumu **`data`**, a način pokretanja programa u datoteci **`run`**.