Multiprocesorski sistemi

Domaći zadatak 4
CUDA – osnove
(10 poena)

Uvod

Cilj zadatka je da studente obuči da samostalno razvijaju osnovne CUDA programe za izvršavanje na grafičkom procesoru.

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. 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/`**.

Izveštaj

Uz predati domaći zadatak (izvorne kodove) treba napisati i priložiti kratak izveštaj o izvršenoj paralelizaciji i dobijenim ubrzanjima u odnosu na sekvencijalnu verziju koda. Za svaki rešeni zadatak treba kratko opisati uočena mesta koja je moguće paralelizovati i način paralelizacije. Takođe, potrebno je dati logove izvršenog koda za sve test primere koji se izvršavaju i nalaze se u **`run`** datoteci i nacrtati grafike ubrzanja u odnosu na sekvencijalnu verziju. Na graficima je potrebno dati i rezultate poređenja različitih načina paralelizacije za isti broj niti, ukoliko postoje takvi zahtevi u okviru teksta zadatka. Šablon za pisanje izveštaja se nalazi u okviru sekcije za domaće zadatke predmetnog sajta.

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 1024. 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/2021-2022/. Na **`rtidev5.etf.rs`** računaru arhiva se može dohvatiti i raspakovati sledećim komandama:

Dohvatanje: **`wget http://mups.etf.rs/dz/2021-2022/MPS_DZ4_CUDA.tar.bz2`**

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

1. **[3p]** Paralelizovati program koji računa integral funkcije F na osnovu unutrašnjosti *simplex*-a (https://en.wikipedia.org/wiki/Simplex) u 20 dimenzija korišćenjem Monte Carlo metode. Program se nalazi u datoteci **`simplex.c`**. U izvornom kodu data je matrica eksponenata jednačine i ivica *simplex*-a. Ulazni parametar programa je broj iteracija aproksimacije. Program testirati sa parametrima koji su dati u datoteci **`run`**.

2. **[3p]** Paralelizovati program koji implementira simulaciju ćelijskog automata *Game of Life*. Simulacija je predstavljena dvodimenzionalnom matricom dimenzija *w x h*, a svaka ćelija *c* može uzeti vrednost 1 ukoliko predstavlja živu ćeliju, a 0 ukoliko je mrtva. Za svaku ćeliju se vrši izračunavanje vrednosti *n* koja predstavlja zbir živih ćelija u susedstvu posmatrane ćelije. Posmatra se osam suseda. Ćelije se rađaju i umiru prema pravilima iz sledeće tabele.

Vrednost C	**Vrednost N**	**Nova vrednost C**	**Komentar**
1	0, 1	0	Usamljena ćelija umire
1	4, 5, 6, 7, 8	0	Ćelija umire usled prenaseljenosti
1	2,3	1	Ćelija živi
0	3	1	Rađa se nova ćelija
0	0, 1, 2, 4, 6, 7, 8	0	Nema promene stanja

   Može se smatrati da su ćelije van opsega posmatrane matrice mrtve. Kod koji treba paralelizovati se nalazi u datoteci **`gameoflife.c`** u arhivi koja je priložena uz ovaj dokument. Program se može prevesti u dve konfiguracije: sa vizuelnim prikazom i bez vizuelnog prikaza, u zavisnosti da li je definisan makro **`LIFE_VISUAL`**. Prevođenje sa vizuelnim prikazom se može izvršiti naredbom **`make visual`**. Paralelizovati konfiguraciju bez vizuelnog prikaza, a vreme meriti na nivou cele simulacije i na nivou jednog izvršavanja funkcije **`evolve`**. Koristiti 2D organizaciju jezgra, ukoliko je moguće. Program testirati sa parametrima koji su dati u datoteci **`run`**.

3. **[3p]** Paralelizovati program koji rešava problem promene temperature na čipu procesora u dvodimenzionalnom prostoru kroz vreme, ako su poznati početna temperatura i granični uslovi. Simulacija rešava seriju diferencijalnih jednačina nad pravilnom mrežom tačaka kojom se aproksimira površina procesora. Svaka tačka u mreži predstavlja prosečnu temperaturu za odgovarajuću površinu na čipu. Mreža tačaka je predstavljena odgovarajućom matricom koja opisuje trenutne temperature. Program se nalazi u direktorijumu **`hotspot`** u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih je od interesa datoteka **`hotspot.c`**. Verifikaciju paralelizovanog rešenja vršiti nad dobijenim temperaturama u poslednjem stanju sistema. Koristiti 2D organizaciju jezgra, ukoliko je moguće. Način pokretanja programa se nalazi u datoteci **`run`**. Kao pomoćno sredstvo, data je i **`python`** skripta koja izlaznu datoteku formatira u **`heatmap`** sliku u PNG formatu.

   [1p] U okviru rešenja sva tri zadatka, obratiti pažnju na efikasnost paralelizacije, mogućnost upotrebe deljene memorije, efikasnost pristupa memoriji, problem divergencije grananja (skokova), proračun indeksa i optimizaciju sekvencijalnog dela koda.