Arhitektura računara

Projektni zadaci – mikroarhitektura/paralelizam (grupa 2)

Zadatak 2.1 – Proširenje ISA simulatora

Proširiti funkcionalnosti iz zadatka 1.1 (simulator arhitekture instrukcijskog skupa).

(10) Ugraditi mehanizam keširanja u simulator. Simulator je potrebno konfigurisati sa veličinom keš memorije, veličinom keš linije i, po potrebi, veličinom radne memorije. Keš memorija treba da radi po *write-back* mehanizmu. Pri svakom pristupu memoriji, potrebno je evidentirati da li je došlo do keš pogotka (*cache hit*) ili promašaja (*cache miss*). Potrebno je voditi evidenciju o ukupnom broja pogodaka i promašaja, što treba biti podatak dostupan upotrebom instrukcije ili debagera. Keš treba podrazumijevano da radi kao direktno-mapirani keš. Zamjena linija se može vršiti primjenom LRU algoritma i primjenom optimalnog (*Bélády*) algoritma.

(5) Omogućiti da korisnik simulatora može konfigurisati broj nivoa keš memorije, te da za svaki nivo može konfigurisati veličinu keš memorije tog nivoa.

(5) Omogućiti da nivo keš memorije bude N-*way associative*, tako da korisnik simulatora za svaki nivo može specificirati vrijednost za N.

(5) Simulirati paralelan rad dva ili viša jezgra upotrebom niti. Jezgra izvršavaju sopstvene tokove instrukcija. Pri pristupu memoriji, svako jezgro treba da pristupa sopstvenoj strukturi koja predstavlja lokalnu keš memoriju, pri čemu je radna memorija dijeljena. Pri tome, na proizvoljan način riješiti problem koherencije keš memorije i demonstrirati rješenje na primjerima u kojima se simulira paralelan rad jezgara koje pristupaju istim keš linijama.

Obezbijediti nekoliko primjera ili jediničnih testova kojima se demonstriraju funkcionalnosti iz stavki u specifikaciji projektnog zadatka.

Sve detalje koji nisu eksplicitno navedeni implementirati na proizvoljan način.

Pridržavati se:

- principa objektno-orijentisanog programiranja i SOLID principa
- principa pisanja čistog koda i pravilnog imenovanja varijabli, funkcija, klasa i metoda
- konvencija za korišteni programski jezik

Zadatak 2.2 – Optimizacija algoritma

(15) U proizvoljnom programskom jeziku realizovati paralelizibilan algoritam koji vrši netrivijalnu obradu podataka. Algoritam odabrati na kursu predmeta.

Program treba da, kao argumente komandne linije, prihvata putanju do ulaznog fajla, putanju do izlaznog fajla, kao i vrijednosti parametara algoritma. Obezbijediti smislene podrazumijevane vrijednosti za sve argumente komandne linije. Analizirati, uporediti, dokumentovati i grafički predstaviti mogućnosti ubrzavanja datog algoritma korištenjem kompajlerskih optimizacija i bar 2 navedena pristupa:

1. (a) SIMD programiranje ili (b) optimizacije za keš memoriju.
    - Za (a) je dozvoljeno koristiti SIMD optimizacije urađene u zadatku 1.2 (asemblerski program za obradu podataka), ako je odabran isti algoritam i za ovaj zadatak.
    - Za (b) je potrebno izvršiti mjerenja (npr. upotrebom *cachegrind* alata) i pokazati da keš optimizacija stvarno doprinosi keš performansama (procentu keš pogodaka).
    - Za (b) je dozvoljeno da se kao inicijalni algoritam iskoristi varijanta algoritma sa lošijim keš performansama (bez dodavanja nepotrebnog koda).
2. Paralelizacija na višejezgarnom procesoru (npr. OpenMP ili sopstveno rješenje).
3. Distribuirano procesiranje (npr. OpenMPI ili sopstveno rješenje).
4. GPGPU programiranje (npr. OpenCL ili CUDA).

(10) Kombinovati dva navedena pristupa u cilju još većeg ubrzanja i dokumentovati rezultate. Pri tome je potrebno zabilježiti i grafički predstaviti rezultate prije i poslije primjene optimizacija i paralelizacije.

Obezbijediti nekoliko primjera ili jediničnih testova kojima se demonstriraju funkcionalnosti iz stavki u specifikaciji projektnog zadatka. Na pravilan način pokazati da će optimizovane varijante algoritma proizvoditi tačan rezultat.

Pri mjerenju vremena izvršavanja:

- Izvršiti mjerenja za veličine ulaza N različitog reda – npr. $N \in \{10^3, 10^4, 10^5, 10^6, 10^7, \dots\}$).
- Izvršiti zagrijavanje prije mjerenja, tj. prije mjerenja izvršiti algoritam više puta bez mjerenja.
- Ako vrijeme izračunavanja algoritma zavisi od sadržaja ulaznih podataka, izvršiti mjerenja za različite slučajeve (npr. nasumični ulazni podaci i granični slučajevi).
- Za svaki ulaz izvršiti mjerenje veći broj puta, a kao konačan rezultat mjerenja dokumentovati prosječnu vrijednost i varijansu.

Pridržavati se:

- principa objektno-orijentisanog programiranja,
- SOLID principa,
- principa pisanja čistog koda i
- konvencija za korišteni programski jezik.

Zadatak 2.3 – Proširenje kernela operativnog sistema

Proširiti funkcionalnosti iz zadatka 1.3 (kernel operativnog sistema), tako da se omogući rad sa više procesorskih jezgara.

(20) Realizovati paralelnu obradu podataka, upotrebom bar dva jezgra. Demonstrirati primjerom.

(5) Implementirati proizvoljnu sinhronizacionu primitivu i demonstrirati način rada na primjeru.

Zadatak 2.4 – Simulator petofazne protočne obrade

(13) Realizovati (u proizvoljnom jeziku) simulator petofazne protočne obrade (IF, ID, EX, MEM, WB). Podržati sljedeće instrukcije: ADD, SUB, MUL, DIV, LOAD i STORE (sa njihovim standardnim značenjem). ADD, SUB, LOAD i STORE zahtijevaju 1 ciklus za izvršavanje (EX faza). MUL i DIV zahtijevaju 2 ciklusa za izvršavanje (EX faza). LOAD i STORE zahtijevaju 3 ciklusa za pristup memoriji. Smatrati da se IF, ID i WB faze uvijek izvrše u jednom ciklusu. Svaka instrukcija raspolaže sa maksimalno tri registarska operanda (odredišni registar i izvorišni registar kod unarnih operacija, odnosno odredišni registar i dva izvorišna registra kod binarnih operacija). Kao ulaz prosljeđuje se niz instrukcija. U tabelarnoj formi prikazati izlaz koji za svaki ciklus (kolonu) prikazuje faze (ili zastoje) u kojima se nalaze ulazne instrukcije (vrste).

(6) Omogućiti da se specificira da li će se vršiti prosljeđivanje podataka između protočnih stepeni ili neće. Omogućiti ispis slučajeva u kojima dođe do prosljeđivanja podatka, sa svim relevantnim informacijama (redni brojevi instrukcija i njihove faze izvršavanja pri prosljeđivanju).

(6) Realizovati detekciju hazarda (WAR, RAW i WAW zavisnosti) između instrukcija u ulaznom nizu.