Lab3 izvještaj

Uvod

Za pokretanje optimizirane verzije programa korištena je ista SLURM skripta kao i u drugoj laboratorijskoj vježbi. Skripta pokreće OpenMP program na sustavu BURA, koristeći različit broj dretvi kako bi se moglo analizirati skaliranje performansi.

```
#!/bin/bash
#SBATCH --job-name=rgb2yuv
#SBATCH --output=rgb2yuv_%j.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --time=00:10:00

export OMP_NUM_THREADS=1
./parallel

export OMP_NUM_THREADS=2
./parallel

export OMP_NUM_THREADS=4
./parallel

export OMP_NUM_THREADS=8
./parallel

export OMP_NUM_THREADS=16
./parallel
```

U nastavku je prikazana tablica s rezultatima mjerenja vremena izvođenja programa na BURI iz druge laboratorijske vježbe, za različit broj dretvi:

Riješenje 2.lab

Broj procesora	Vrijeme izvođenja	Ubrzanje	Efikasnost
1	17.646700 s	1x	100%
2	13.456743 s	1.311x	65.55%
4	10.043702 s	1.757x	43.93%
8	8.355344 s	2.112x	26.4%
16	7.399746 s	2.385x	14.91%

1. Optimizacija

Kao prvi pokušaj optimizacije odlučio sam paralelizirati obradu frameova dodavanjem OpenMP direktive #pragma omp parallel for prije glavne petlje u funkciji main

```
80      #pragma omp parallel for
81      for (int frame = 0; frame < 60; frame++) {
```

Cilj ove optimizacije bio je omogućiti paralelnu obradu više frameova u isto vrijeme, čime bi se postiglo značajno ubrzanje, posebno na sustavima s većim brojem jezgri. Kod se nakon ove izmjene uspješno kompajlirao i izvodio, a vrijeme izvođenja se smanjilo.

Broj procesora	Vrijeme izvođenja	Ubrzanje	Efikasnost
1	18.658335 s	1x	100%
2	12.512059 s	1.49x	74.5%
4	8.317757 s	2.243x	56.08%
8	6.685395 s	2.791x	34.89%
16	5.81045 s	3.211x	20.07%

Pri analizi izlazne datoteke ustanovio sam da slike nisu ispravno spremljene – dolazi do oštećenja ili zamjene redoslijeda frameova. Problem je u tome što svaka dretva istovremeno otvara iste izlazne datoteke i upisuje podatke, podaci se međusobno preklapaju i izlaz više ne predstavlja ispravan video.

Iako je ova paralelizacija naizgled jednostavna i učinkovita, zbog dijeljenja istih datoteka među dretvama ona dovodi do nepravilnih rezultata. Zbog toga sam ovu optimizacija odbacio u konačnoj implementaciji.

2. Optimizacija

Nakon što se pokazalo da paralelizacija po frameovima jednostavnim #pragma omp parallel for direktivama ne daje ispravne rezultate zbog I/O ograničenja i potrebe za redosljednim zapisivanjem u .yuv datoteku, pokušao sam optimizirati program korištenjem OpenMP taskova.

```
64      #pragma omp parallel
65      {
66          #pragma omp single
67          {
68              for (int frame = 0; frame < 60; frame++) {
69                  #pragma omp task firstprivate(frame)
```

U ovoj verziji koda svaki frame se obrađuje u zasebnom OpenMP tasku, dok su sve datoteke dijeljene. Za svaki task:

- koristi se zasebna memorija za RGB i YUV podatke da bi se izbjeglo preklapanje,
- RGB→YUV konverzija, kao i pod/naduzorkovanje se paralelno izvode unutar taska,
- čitanje iz i pisanje u .yuv datoteke odvija se u critical sekcijama da bi se izbjegli race condition problemi.

```
82              #pragma omp critical
83              {
84                  fread(R, 1, FRAME_SIZE, input);
85                  fread(G, 1, FRAME_SIZE, input);
86                  fread(B, 1, FRAME_SIZE, input);
87              }
```

Broj procesora	Vrijeme izvođenja	Ubrzanje	Efikasnost
1	21.185975 s	1x	100%
2	12.455623 s	1.701x	85.05%
4	8.979593 s	2.359x	58.98%
8	7.159784 s	2.959x	36.99%
16	6.852989 s	3.092x	19.33%

Nakon usporedbe s početnom verzijom koda možemo zaključiti da je izvedba s jednom dretvom sporija u odnosu na originalni paralelni kod zbog overheada kreiranja taskova i potrebe za sinkronizacijom, pri izvođenju na više jezgri taskovi omogućuju veće ukupno ubrzanje i bolju efikasnost. To se postiže boljim iskorištavanjem paralelizma između frameova, za razliku od prvotne verzije koja paralelizira isključivo unutar jednog framea.

Dodatna prednost ove optimizacije je da konačni generirani video ima ispravan izgled, za razliku od ranijeg pokušaja paralelizacije preko frameova pomoću #pragma omp parallel for, koji je uzrokovao vizualne greške zbog nekonzistentnog redoslijeda zapisivanja frameova u izlazne datoteke.

3.Optimizacija

Treća optimizacija bila je nastavak druge s ciljem uklanjanja direktive #pragma omp critical koja je ograničavala paralelizaciju prilikom upisa u izlazne .yuv datoteke.

Umjesto sinkroniziranog upisa unutar OpenMP taskova, IO operacije su potpuno uklonjene iz paralelnog dijela koda. Taskovi su zaduženi isključivo za obradu svakog framea i alociranje pripadajućih YUV podataka u zajedničke nizove (Y_all, U_all, itd.).

Nakon završetka paralelne obrade, svi dobiveni podaci se sekvencijalno zapisuju u datoteke iz glavne niti.

Na taj je način izbjegnuta potreba za zaključavanjem dretvi i upis je postao jednostavan i siguran. Međutim, ova optimizacija nije rezultirala dodatnim ubrzanjem u odnosu na prethodnu verziju, nego blagom pogoršanju performansi. To se pripisuje dodatnom vremenu potrebnom za kopiranje podataka u zajedničke strukture i većem memorijskom opterećenju.

Broj procesora	Vrijeme izvođenja	Ubrzanje	Efikasnost
1	19.487620 s	1x	100%
2	13.296030 s	1.466x	73.3%
4	10. 116180 s	1.926x	48.15%
8	8.708998 s	2.238x	27.98%
16	7.473578 s	2.608x	16.3%

Zaključak

Tijekom treće laboratorijske vježbe ispitane su različite tehnike optimizacije paralelnog programa za obradu YUV video podataka korištenjem OpenMP-a. Početni kod (parallel_orig_lab2.c) već je uključivao osnovnu paralelizaciju unutar obrade jednog framea, ali nije iskorištavao mogući paralelizam između frameova.

Prva optimizacija (parallel_opt1.c) pokušala je iskoristiti taj paralelizam dodavanjem direktive #pragma omp parallel for nad glavnu petlju koja iterira kroz frameove. Iako je postignuto značajno ubrzanje, rezultati su bili neispravni zbog istovremenog pristupa istim izlaznim datotekama, što je dovelo do narušenog redoslijeda i korumpiranih podataka.

Druga optimizacija (parallel_opt2.c) riješila je problem dijeljenja I/O resursa uvođenjem OpenMP *taskova*. Svaki frame se obrađuje u vlastitom tasku, a pristup zajedničkim datotekama zaštićen je pomoću #pragma omp critical. Ovaj pristup omogućio je korektno izvođenje i najbolji odnos između brzine i ispravnosti rezultata. Iako su performanse za 1 dretvu nešto slabije zbog overheada taskova i sinkronizacije, ubrzanjem na više jezgri nadmašuje sve ostale verzije.

Treća optimizacija (parallel_opt3.c) dodatno je pokušala izbjeći korištenje critical sekcija tako da se svi I/O postupci obave izvan paralelnog dijela koda. Time se eliminiraju race condition problemi, ali uz cijenu dodatnog memorijskog opterećenja i nešto slabije performanse zbog kopiranja podataka u zajedničke strukture.

Kao konačni zaključak, najbolje rješenje pokazalo se parallel_opt2.c, koje koristi OpenMP taskove s pažljivo sinkroniziranim pristupom zajedničkim resursima. Ova

implementacija ostvaruje najbolji balans između točnosti rezultata i paralelne učinkovitosti.

Student: Boris Boronjek

JMBAG: 0036531473