

**UNIVERSITETI POLITEKNIK I TIRANËS**

**FAKULTETI I TEKNOLOGJISË DHE INFORMACIONIT**

**DEPARTAMENTI I INXHINIERISË INFORMATIKE**

**Punë Laboratori nr. 1**

**Lënda:** Sisteme Te Shperndara

**Grupi:** III-B

**Punoi:**  **Pranoi**:

Piro Gjikdhima MSc.Megi Tartari

**Ushtrimi 1**

Llogaritja e shumes se elementeve te nje vektori ne menyre seriale dhe paralele.

*Kodi*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

#define MAX\_THREADS 6  // Numri maksimal i fijeve që do të testojmë

#define START\_SIZE 100  // Madhësia fillestare e vektorit

#define MAX\_EXPONENT 5 // Numri i rritjeve (p.sh., 100, 1000, 10000, ...)

double array\_sum\_parallel(const int\* array, size\_t size, int num\_threads) {

    double sum = 0.0;

    omp\_set\_num\_threads(num\_threads);

    #pragma omp parallel for reduction(+:sum)

    for (size\_t i = 0; i < size; i++) {

        sum += array[i];

    }

    return sum;

}

int main() {

    printf("Testimi i performancës me ndryshim të madhësisë së vektorit dhe numrit të thredeve\n");

    // Kalojmë madhësinë e vargut nga 100 në 100\*10^MAX\_EXPONENT

    for (int exp = 0; exp <= MAX\_EXPONENT; exp++) {

        size\_t array\_size = START\_SIZE;

        for (int i = 0; i < exp; i++) {

            array\_size \*= 10;

        }

        printf("\nMadhësia e vargut: %zu elemente\n", array\_size);

        // Alokojmë vargun dhe e inicializojmë me vlera të rastësishme

        int\* array = (int\*)malloc(array\_size \* sizeof(int));

        if (!array) {

            fprintf(stderr, "Dështoi alokimi i kujtesës\n");

            return 1;

        }

        srand(time(NULL));

        for (size\_t i = 0; i < array\_size; i++) {

            array[i] = rand() % 100 + 1;

        }

        // Ekzekutojmë mbledhjen seriale

        double start\_time = omp\_get\_wtime();

        double serial\_sum = array\_sum\_parallel(array, array\_size, 1);

        double serial\_time = omp\_get\_wtime() - start\_time;

        printf("Shuma seriale: %.0f (u desh %.6f sekonda)\n", serial\_sum, serial\_time);

        // Testojmë për fije nga 2 deri në 6

        for (int threads = 2; threads <= MAX\_THREADS; threads++) {

            start\_time = omp\_get\_wtime();

            double parallel\_sum = array\_sum\_parallel(array, array\_size, threads);

            double parallel\_time = omp\_get\_wtime() - start\_time;

            printf("Shuma paralele (%d threde): %.0f (u desh %.6f sekonda, përshpejtimi %.2fx)\n",

                   threads, parallel\_sum, parallel\_time, serial\_time / parallel\_time);

        }

        free(array);

    }

    return 0;

}

*Rezultate*

Kur punojmë me vektorë të vegjël (100 - 10,000 elemente), nuk vërejmë ndonjë përmirësim të ndjeshëm në ekzekutimin paralel. Kjo ndodh sepse overhead-i i krijimit dhe menaxhimit të thredeve është më i madh sesa përfitimi i ndarjes së punës. Në këtë rast, koha e shtuar për sinkronizimin dhe koordinimin e thredeve e bën ekzekutimin paralel joefektiv.

Në rastin e vektorëve mesatarë (100,000 elemente), paralelizimi fillon të tregojë përmirësime të dukshme. Procesori shfrytëzon më mirë “multicore” duke ndarë punën në seksione më të mëdha për çdo thred. Kjo redukton kohën totale të ekzekutimit, pasi çdo thred përpunon një pjesë më të madhe të të dhënave në mënyrë të pavarur.

Për vektorë të mëdhenj (1,000,000 elemente), përfitimet e paralelizimit bëhen më të theksuara. Overhead-i i krijimit të thredeve bëhet i papërfillshëm krahasuar me kohën e përgjithshme të përpunimit. Kjo çon në një përshpejtim të ndjeshëm, duke arritur deri në ~4.8x më shpejt me 6 threde në krahasim me ekzekutimin serial.

Në përfundim, paralelizimi është i dobishëm për madhësi të mëdha të të dhënave, ku ndarja e punës sjell një reduktim të ndjeshëm të kohës së ekzekutimit. Megjithatë, për madhësi të vogla, overhead-i i menaxhimit të thredeve e bën ekzekutimin paralel më pak efikas sesa ai serial.

A screenshot of a computer

AI-generated content may be incorrect.

**Ushtrimi 2**

Llogaritja e faktorialit te elementeve te nje vektori ne menyre seriale dhe paralele.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

#define MAX\_THREADS 6  // Numri maksimal i thredeve që do të testojmë

#define START\_SIZE 100 // Madhësia fillestare e vektorit

#define MAX\_EXPONENT 5 // Numri i rritjeve (p.sh., 100, 1000, 10000, ...)

// Llogarit faktorialin e një numri

unsigned long long factorial(int n) {

    if (n <= 1) return 1;

    unsigned long long result = 1;

    for (int i = 2; i <= n; i++) {

        result \*= i;

    }

    return result;

}

// Implementimi serial

void compute\_factorials\_serial(const int\* input, unsigned long long\* output, size\_t size) {

    for (size\_t i = 0; i < size; i++) {

        output[i] = factorial(input[i]);

    }

}

// Implementimi paralel me OpenMP

void compute\_factorials\_parallel(const int\* input, unsigned long long\* output, size\_t size, int num\_threads) {

    omp\_set\_num\_threads(num\_threads);

    #pragma omp parallel for

    for (size\_t i = 0; i < size; i++) {

        output[i] = factorial(input[i]);

    }

}

int main() {

    printf("Testimi i performancës me ndryshim të madhësisë së vektorit dhe numrit të thredeve\n");

    for (int exp = 0; exp <= MAX\_EXPONENT; exp++) {

        size\_t array\_size = START\_SIZE;

        for (int i = 0; i < exp; i++) {

            array\_size \*= 10;

        }

        printf("\nMadhësia e vargut: %zu elemente\n", array\_size);

        // Alokojmë vektoret

        int\* input\_array = (int\*)malloc(array\_size \* sizeof(int));

        unsigned long long\* output\_serial = (unsigned long long\*)malloc(array\_size \* sizeof(unsigned long long));

        unsigned long long\* output\_parallel = (unsigned long long\*)malloc(array\_size \* sizeof(unsigned long long));

        if (!input\_array || !output\_serial || !output\_parallel) {

            fprintf(stderr, "Dështoi alokimi i kujtesës\n");

            return 1;

        }

        srand(time(NULL));

        for (size\_t i = 0; i < array\_size; i++) {

            input\_array[i] = rand() % 20 + 1; // Vlera midis 1 dhe 20 për të shmangur overflow

        }

        // Ekzekutojmë versionin serial

        double start\_time = omp\_get\_wtime();

        compute\_factorials\_serial(input\_array, output\_serial, array\_size);

        double serial\_time = omp\_get\_wtime() - start\_time;

        printf("Koha seriale: %.6f sekonda\n", serial\_time);

        // Testojmë versionin paralel me 2 deri në 6 threde

        for (int threads = 2; threads <= MAX\_THREADS; threads++) {

            start\_time = omp\_get\_wtime();

            compute\_factorials\_parallel(input\_array, output\_parallel, array\_size, threads);

            double parallel\_time = omp\_get\_wtime() - start\_time;

            printf("Koha paralele (%d threde): %.6f sekonda, përshpejtimi %.2fx\n",

                   threads, parallel\_time, serial\_time / parallel\_time);

        }

        free(input\_array);

        free(output\_serial);

        free(output\_parallel);

    }

    return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.*Rezultate*

Të njëjtat rezultate si ushtrimi 1 përftojmë dhe në ushtrimin 2.

Kur punojmë me vektorë të vegjël (100 - 10,000 elemente), paralelizimi nuk tregon përmirësime të dukshme. Kjo ndodh sepse overhead-i i krijimit dhe menaxhimit të thredeve është më i madh sesa përfitimi i ndarjes së punës. Në këtë rast, koha e shtuar për sinkronizimin dhe koordinimin e thredeve e bën ekzekutimin paralel joefektiv.

Në rastin e vektorëve mesatarë (100,000 elemente), paralelizimi fillon të tregojë përmirësime të dukshme. Procesori shfrytëzon më mirë shumëbërthamësinë duke ndarë punën në seksione më të mëdha për çdo thred. Kjo redukton kohën totale të ekzekutimit, pasi çdo thred përpunon një pjesë më të madhe të të dhënave në mënyrë të pavarur.

Për vektorë të mëdhenj (1,000,000 elemente), përfitimet e paralelizimit bëhen më të theksuara. Overhead-i i krijimit të thredeve bëhet i papërfillshëm krahasuar me kohën e përgjithshme të përpunimit. Kjo çon në një përshpejtim të ndjeshëm, duke arritur deri në ~3.11x më shpejt me 6 threde në krahasim me ekzekutimin serial.

Për vektorë shumë të mëdhenj (10,000,000 elemente), përfitimet e paralelizimit bëhen jashtëzakonisht të dukshme. Overhead-i i krijimit të thredeve bëhet i papërfillshëm dhe koha e përpunimit zvogëlohet në mënyrë të konsiderueshme. Kjo çon në një përshpejtim të jashtëzakonshëm, duke arritur deri në ~5.22x më shpejt me 6 threde në krahasim me ekzekutimin serial.

Ky është një konfirmim i parimit të përgjithshëm që paralelizimi është më efektiv kur madhësia e problemit është mjaft e madhe për të justifikuar koston e krijimit dhe menaxhimit të thredeve.