Факултет техничких наука

Универзитет у Новом Саду

Рачунарски системи високих перформанси

Имплементација Кули-Тукеј алгоритма за брзе Фуријеове трансформација

Аутор:
Никола Калинић

Индекс:
E2 41/2023

17. јануар 2024.

Сажетак

Фуријеове трансформације кључне су за обраду сигнала, али се суочавају с изазовима временске сложености, посебно код дискретних Фуријеових трансформација које имају сложеност $O(n^2)$. Да би се превазишао овај изазов, развијени су бржи алгоритми, попут брзих Фуријеових трансформација, које смањују временску сложеност на $O(n\ logn)$ и ефикасно оптимизују анализу података. Овај рад истражује могућности додатног убрзања Кули-Тукеј алгоритма кроз примену $OpenMP$ и $OpenMPI$ алата. Показујући да је ефикаснија паралелизација постигнута радом с матрицом података која има независне редове, уместо с једним низом.

Садржај

1 Увод

Фуријеове трансформације[1] су кључни математички алат који има широку примену у различитим дисциплинама, од инжењеринга до физике, рачунарства и многих других области. Ове трансформације омогућавају анализу сигнала у фреквенцијском домену, што значајно олакшава проучавање њихових карактеристика. Фуријеове трансформације омогућавају прелазак из временског домена у фреквенцијски домен, што је од суштинског значаја за анализу сигнала. На пример, у електроинжењерингу, могу се користити за анализу електричних сигнала, док се у телекомуникацијама користе за обраду и пренос информација. У области обраде слика, Фуријеове трансформације се користе за анализу и манипулацију слика. Трансформације омогућавају екстракцију информација о фреквенцијама присутним у слици, што може бити корисно за детекцију узорака, филтрирање и друге методе обраде слика. Фуријеове трансформације имају кључну улогу у многим алгоритмима у рачунарству. На пример, брза Фуријеова трансформација (ФФТ) се често користи за ефикасно израчунавање Фуријеове трансформације у контексту дигиталне обраде сигнала. У области електронике, Фуријеове трансформације су од суштинског значаја за анализу и пројектовање електричних кола, посебно у вези са обрадом сигнала. У телекомуникацијама се често користе за модулацију, демодулацију и анализу сигнала. Фуријеове трансформације се користе у математичком моделирању како би се репрезентовале функције и сигнали у фреквенцијском домену. У квантној механици, Фуријеове трансформације се користе за анализу таласних функција и вероватноћа расподела у простору и импулсу. Ове трансформације су кључне за разумевање понашања микрочестица на квантном нивоу. Укратко, Фуријеове трансформације су незаобилазан инструмент у многим дисциплинама које се баве анализом и обрадом сигнала. Њихова способност да пренесу информације из временског у фреквенцијски домен чини их драгоценим алатом за истраживање и примену у различитим научним и инжењерским подручјима.

Кули-Тукеј алгоритам представља ефикасан начин за рачунање брзе Фуријеове трансформације (ФФТ), која је кључна операција у дигиталној обради сигнала, телекомуникацијама, рачунарству и многим другим областима. Овај алгоритам, назван по својим творцима Џејмсу Кулију и Џону Тукеју, значајно убрзава израчунавање Фуријеове трансформације, посебно када се примењује на велике сетове података.

У овом раду биће истражени детаљи имплементације Кули-Тукеј алгоритма за брзу Фуријеову трансформацију (ФФТ) у његовој секвенцијалној верзији, а затим ћемо проћи кроз процес паралелизације коришћењем $OpenMP$ [2]и $OpenMPI$ [3] за побољшање перформанси на вишејезгарним и дистрибуираним системима.

2 Приказ полинома у рачунарству

У рачунарству, представљање полинома има значајан утицај на различите области, укључујући математику, инжењеринг, рачунарску графику, анализу података и многе друге[4] .Полиноми су математички изрази који се састављају од различитих потенција променљиве, а њихово ефикасно и прецизно приказивање у рачунару кључно је за разноврсне апликације. Постоје два основна начина за представљање полинома у рачунарском окружењу: коначни низ коефицијената и представљање помоћу чворова.

2.1 Приказивање полинома кроз коначан низ коефицијента

Један од најефикаснијих начина за представљање полинома у рачунару је коришћење коначног низа коефицијената. Овај приступ се заснива на идеји чувања коефицијената полинома у облику низа бројева. Узмимо, на пример, полином:

$$P(x) = 2x^3 + 4x^2 - 3x + 7$$

Овај полином се може представити кроз низ коефицијената [2,4,−3,7]. Сваки елемент низа одговара коефицијенту уз одговарајући степен променљиве x. Први елемент низа се односи на коефицијент уз x^3, други на коефицијент уз x^2, трећи на коефицијент уз x^1, и четврти на слободни члан или коефицијент уз x^0.

Предности оваквог начина коришћења су следеће:

- **Једноставност аритметичких операција**: Аритметичке операције над полиномима, као што су сабирање, одузимање и множење, постају једноставне итерацијом кроз низ коефицијената. На пример, да би се сабрала два полинома, једноставно се сабирају одговарајући коефицијенти.
- **Лака имплементација алгоритама**: Алгоритми који се користе за манипулацију полиномима често су једноставни и ефикасни када се примењују директно на низ коефицијената.
- **Ефикасност коришћења меморије**: Представљање полинома низом коефицијената често заузима мање меморије у односу на алтернативне структуре података.

2.2 Представа полинома помоћу чворова

Други значајан приступ представљању полинома у рачунару укључује коришћење структуре података познате као чворови. Овај метод користи бинарно стабло, где

сваки чвор представља један члан полинома. Сваки чвор садржи два кључна елемента: експонент потенције променљиве и коефицијент тог члана. На пример, полином:

$$P(x) = 3x^4 - 2x^2 + 5x + 1$$

Се може представити помоћу чворова: (3,4), (−2,2), (5,1), (1,0). Где први елемент у сваком пару означава коефицијент, а други експонент потенције променљиве.

Предности оваквог начина коришћења су следеће:

- **Ефикасно дељење и факторизација**: Представљање полинома помоћу чворова олакшава операције дељења и факторизације. Бинарно стабло омогућава брз приступ члановима, што је од суштинског значаја за ефикасно извођење ових операција.
- **Отпорност на велике експоненте**: У ситуацијама где полиноми имају велике експоненте, представљање чворовима може значајно смањити простор за чување информација, посебно у односу на низ коефицијената.
- **Ефикасна манипулација структуром стабла**: Бинарно стабло омогућава ефикасно претраживање и манипулацију структуром, што је корисно за разне алгоритме у анализи и манипулацији полиномима.

3 Кули-Тукеј алгоритам

3.1 Проблем са дискретним Фуријеовим трансформација

Постоје два кључна типа Фуријеових трансформација: дискретна Фуријеова трансформација (ДФТ) и брза Фуријеова трансформација (ФФТ). Дискретна Фуријеова Трансформација (ДФТ) представља кључан математички алат за анализу сигнала у фреквенцијском домену. ДФТ се примењује на дискретне секвенце података, често на временске серије, како би се прешло из временског домена у фреквенцијски. Класична ДФТ се рачуна према следећој формули за N-тачкасти сигнал $x[n]$:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}kn}$$

Где су $X(k)$ резултујуће фреквенције, $x[n]$ вредности сигнала у временском домену, k индекс фреквенције, а N број тачака у сигналу. Међутим, класична имплементација ове трансформације има временску сложеност од $O(n^2)$, што постаје непрактично за велике сетове података.

Да би се ублажио проблем временске сложености ДФТ-а, развијена је Брза Фуријеова Трансформација (ФФТ). ФФТ је ефикасан алгоритам који драматично смањује временску сложеност на $O(n \log n)$, захваљујући стратегији подели и владај. Постоје различити алгоритми за ФФТ, а један од најпознатијих је Кули-Тукеј алгоритам.

3.2 Основна идеја алгоритма

Кули-Тукеј алгоритам је ефикасан алгоритам за израчунавање брзе Фуријеове трансформације (ФФТ) и представља једну од најважнијих метода у нумеричкој анализи. Ова техника убрзава израчунавање Дискретне Фуријеове Трансформације, посебно када се користи на секвенцама података чија је дужина степен броја 2.

Кораци самог алгоритма:

- **Рекурзивна подела**: Основна идеја Кули-Тукеј алгоритма лежи у рекурзивној подели ДФТ-а на мање подпроблеме. Ако имамо N-тачкасти сигнал, алгоритам га дели на два подпроблема, сваки са $N/2$ тачака. Ова подела се рекурзивно примењује на сваки од подпроблема све док се не дође до тачака где је $N = 1$. Тада се директно израчунавају ДФТ-ови за сваку од тих тачака.
- **Комбинација резултата**: Након што се рекурзивно израчунају ДФТ-ови за мање подпроблеме, резултати се комбинују на ефикасан начин како би се добила

коначна ДФТ секвенце. Ова фаза комбинације подразумева сабирање чланова добијених из рекурзивних позива и формирање коначног резултата.

- **Употреба периодичних односа**: Кључна карактеристика Кули-Тукеј алгоритма лежи у коришћењу периодичних односа у тригонометрији, посебно у односима који се појављују у комплексним експоненцијалама. Ови периодични односи омогућавају ефикасно комбиновање резултата и смањују укупан број потребних операција.

Кули-Тукеј алгоритам значајно побољшава временску сложеност израчунавања ДФТ-а, смањујући је са $O(N^2)$ на $O(n \log n)$, где је N број тачака у сигналу. Ова ефикасност чини овај алгоритам кључним за многе примене, посебно у контексту брзе анализе сигнала у реалном времену и дигиталне обраде података.

4 Секвенцијална имплементација

У овом поглављу, описана је секвенцијална имплементација Кули-Тукеј алгоритма, након што смо размотрили основну идеју овог алгоритма. У секвенцијалном извршавању, Кули-Тукеј алгоритам придржава се принципа рекурзивне подела и комбинације резултата, али детаљи имплементације доносе додатне нијансе у разумевању како се ова магија убрзања дешава.

```
void fft(complex double *X, int N) {
    if (N <= 1) return;


    complex double even[N/2];
    complex double odd[N/2];
    for (int i = 0; i < N/2; i++) {
        even[i] = X[i*2];
        odd[i] = X[i*2 + 1];
    }


    fft(even, N/2);
    fft(odd, N/2);


    for (int i = 0; i < N/2; i++) {
        complex double t = cexp(-2.0 * I * PI * i / N) * odd[i];
        X[i] = even[i] + t;
        X[i + N/2] = even[i] - t;
    }
}
```

Listing 1: Секвенцијална имплементација

Када погледамо рекурзивну поделу у секвенцијалном контексту што је призано на листингу 1, неколико кључних тачака додатно се истиче. Прво, ефикасно управљање меморијом и итерацијама пружа основу за оптимално дељење низа у мање подпроблеме. У секвенцијалном току, овај корак захтева пажљив рад са поднизовима, узимајући у обзир ограничења меморије и приступ до података.Након рекурзивне подела, хармонично комбиновање резултата постаје кључно. У секвенцијалном окружењу, ефикасност ове фазе зависи од пажљивог управљања радом са комплексним бројевима. Правилно прилагођавање резултата рекурзивних позива, узимајући у обзир оптималну манипулацију комплексним факторима, доприноси бржем формира-

њу крајње ДФТ секвенце. Секвенцијална имплементација Кули-Тукеј алгоритма у реалном свету често захтева додатне оптимизације како би се постигла најбоље могуће перформансе. Прилагођавање величине поднизова, коришћење ефикасних меморијских структура и разматрање специфичности платформе играју кључну улогу у постизању оптималне секвенцијалне изведбе.

Секвенцијална имплементација Кули-Тукеј алгоритма задржава есенцијалне кораке који чине основну идеју, али додатно наглашава потребу за тактичким приступом код управљања меморијом и оптимизацијама. Ово омогућава да алгоритам покаже своју праву моћ у реалним сценаријима, посебно у областима где брза анализа сигнала и дигитална обрада података захтевају равнотежу између ефикасности и ресурса.

5 Паралелна имплементација

Покушај паралелизације Кули-Тукеј алгоритма, упркос својој обећавајућој природи, суочава се с изазовима услед специфичности рекурзивне структуре алгоритма. У наредном разматрању, истражићемо кључне изазове који произлазе из рекурентне природе алгоритма и како ова међузависност ствара значајне тешкоће у ефикасној паралелизацији, нарочито када користимо технологије као што су $OpenMP$ и $OpenMPI$.

5.1 Паралелна имплементација помоћу OpenMP

На листингу 2 приказан је Кули-Тукеј алгоритам за брзу Фуријеову трансформацију (ФФТ) користећи $OpenMP$ за паралелизацију.

```
void fft(complex double *X, int N) {
    if (N <= 1) return;

    complex double even[N/2];
    complex double odd[N/2];

    #pragma omp parallel for
    for (int i = 0; i < N/2; i++) {
        even[i] = X[i*2];
        odd[i] = X[i*2 + 1];
    }

    #pragma omp parallel sections
    {
        #pragma omp section
        fft(even, N/2);

        #pragma omp section
        fft(odd, N/2);
    }

    #pragma omp parallel for
    for (int i = 0; i < N/2; i++) {
        complex double t = cexp(-2.0 * I * PI * i / N) * odd[i];

        #pragma omp critical
        {
```

```
            X[i] = even[i] + t;
            X[i + N/2] = even[i] - t;
        }
    }
}
```

Listing 2: Паралелизација помоћу *OpenMP*

Кључни аспекти и проблеми укључени у имплементацију су следећи:

- **Рекурзивна структура алгоритма**: Кули-Тукеј алгоритам се заснива на рекурзивној подели низа. *OpenMP*, док подржава паралелизацију, није оптимално прилагођен за рекурзивне задатке. У овом коду, рекурзивни позиви реализовани су кроз `#pragma omp sections` и `#pragma omp section` како би се омогућила паралелизација на мањим подпроблемима.
- **Синхронизација и критичне сецкије**: Како бисмо избегли трке између нити приликом комбиновања резултата, користимо критичне секције `#pragma omp critical`. Ипак, овај приступ може донети значајан додатни трошак, посебно када имамо велики број нити.
- **Динамичко прилагођавање броја нити**: Код користи `#pragma omp parallel for` што је приказано на листингу 3 како би дозволио да свака нит засебно, покрене свој рекурзивни позив. Динамичко прилагођавање броја нити може бити неопходно како би се постигао оптималан баланс између оптерећења и брзине извршавања.

```
#pragma omp parallel for
for (int i = 0; i < rows; i++) {
    fft(X[i], cols);
}
```

Listing 3: Покретање паралелизација помоћу *OpenMP*

Ова имплементација покушава адекватно адресирати ове изазове, али постизање оптималне паралелизације може захтевати даље прилагођавање у зависности од конкретних захтева проблема и рачунарске архитектуре.

5.2 Паралелна имплементација помоћу OpenMPI

Кули-Тукеј алгоритам је структуриран за рад над низом података који су обликовани као степен двојке. Ова специфичност чини га мање погодним за обраду једног јединственог низа података, посебно у контексту паралелизације. Овај изазов произлази из чињенице да би сваки подниз, који се добија дељењем по два, требало да

буде независан од других, што није могуће када се ради са једним низом. Да бисмо превазишли овај изазов, уводимо концепт обраде матрице података уместо једног низа. Сваки ред матрице посматраћемо као један независан низ, над којим ћемо применити Кули-Тукеј алгоритам. Ово нам омогућава да паралелизујемо обраду сваког реда, с обзиром да су они независни од других. Овим прилагођавањем постижемо ефикасније искоришћење паралелног извршавања, истовремено смањујући природна ограничења алгоритма.

```
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rows = 50000;
    int cols = 8;

    int total_elements = rows * cols;

    int local_rows = rows / size;
    int local_elements = local_rows * cols;

    int xyz = local_elements * sizeof(complex double);
    complex double* local_X = (complex double*)malloc(xyz);

    complex double* gathered_X = NULL;

    if (rank == 0) {
        complex double X[rows][cols];
        for (int i = 0; i < rows; i++) {
            X[i][0] = 1;
            X[i][1] = 2;
            X[i][2] = 3;
            X[i][3] = 4;
            X[i][4] = 4;
            X[i][5] = 3;
            X[i][6] = 2;
            X[i][7] = 1;
        }
```

```
        MPI_Scatter(X, local_elements, MPI_C_DOUBLE_COMPLEX,
                    local_X, local_elements, MPI_C_DOUBLE_COMPLEX,
                    0, MPI_COMM_WORLD);

        int buffer_size = total_elements * sizeof(complex double);
        gathered_X = (complex double*)malloc(buffer_size);
    } else {
        MPI_Scatter(NULL, 0, MPI_DATATYPE_NULL, local_X,
        local_elements, MPI_C_DOUBLE_COMPLEX, 0, MPI_COMM_WORLD);
    }

    for (int i = 0; i < local_rows; i++) {
        fft(&local_X[i * cols], cols);
    }

    MPI_Gather(local_X, local_elements, MPI_C_DOUBLE_COMPLEX,
                gathered_X, local_elements, MPI_C_DOUBLE_COMPLEX,
                0, MPI_COMM_WORLD);

    if (rank == 0) {
        free(gathered_X);
    }

    free(local_X);
    MPI_Finalize();
    return 0;
}
```

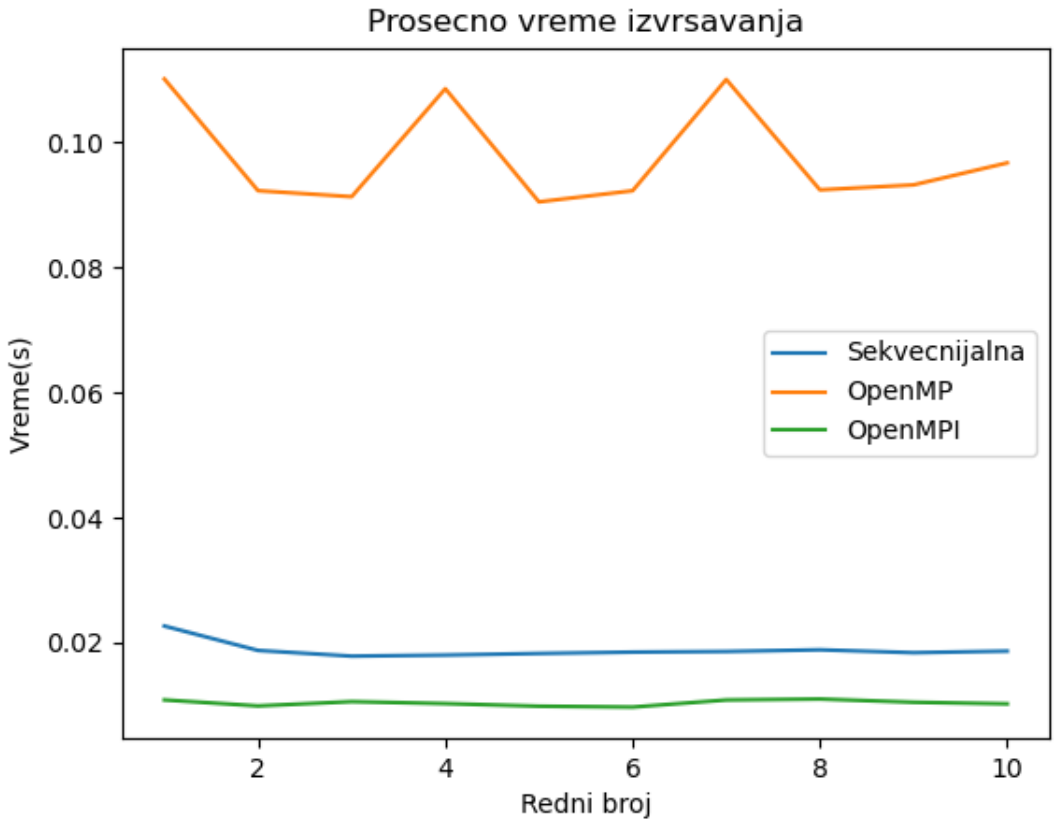
Listing 4: Паралелизација помоћу $OpenMPI$

За реализацију паралелизације Кули-Тукеј алгоритма користимо $OpenMPI$, библиотеку која омогућава дистрибуирано извршавање кода на више процесорских чворова. Сваки чвор процесорског кластера обрађује одређени део матрице података. Кроз $OpenMPI$ функције попут `MPI_Scatter` и `MPI_Gather`, расподељујемо и прикупљамо податке међу процесима. Паралелизација Кули-Тукеј ФФТ алгоритма са $OpenMPI$ и коришћењем матрице као улаза представља ефикасно решење за побољшање перформанси израчунавања брзе Фуријеове трансформације. Ова адаптација омогућава нам да ефикасно искористимо потенцијал вишепроцесорских архитектура, превазилазећи ограничења која произлазе из секвенцијалне природе алгоритма.

6 Преглед резултата

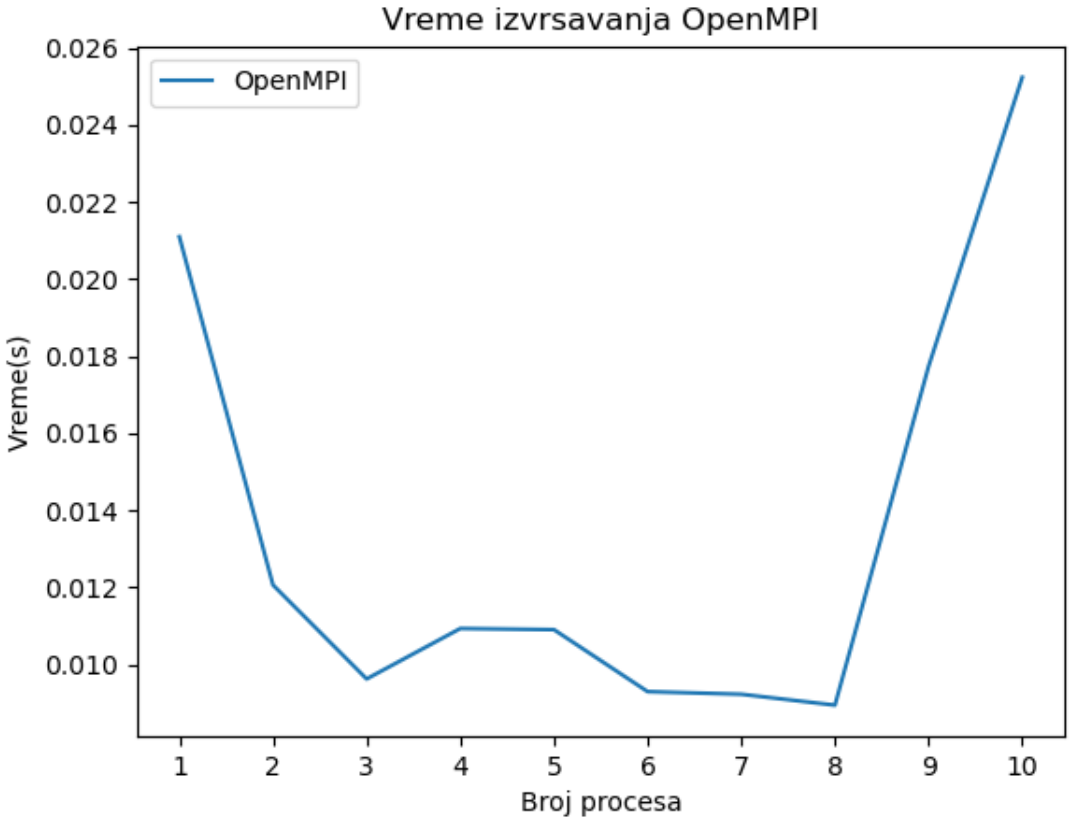
Као што је наведено у претходном поглављу имплементација Кули-Тукеј алгоритма у оригиналном облику суочава се са значајним изазовима када је реч о паралелизацији, због рекурзивне природе алгоритма и његове зависности између итерација. У намери да превазиђемо ове изазове, одабрана је матрица података као алтернативна репрезентација сигнала. Сваки ред матрице представља појединачни низ података, што омогућава симулацију ефикасне паралелизације. Ова прилагођена структура матрице дозвољава независност између редова, чиме се елиминише потреба за синхронизацијом између итерација и омогућава ефикасно паралелно извршавање. Величина матрице која се користи за тестирање овог приступа је 50000 редова по 8 колона, пружајући адекватно окружење за евалуацију перформанси.

Запажамо значајне разлике у просечном времену извршавања између секвенцијалног и паралелног решења Кули-Тукеј алгоритма над матрицом димензија 50000x8 што се може видети на слици 1. Секвенцијално решење остварује изузетно брзе перформансе са просечним временом од 0.0183 секунде. С друге стране, паралелно решење путем $OpenMP$-а показује дуже време извршавања са просеком од 0.093 секунде, док је паралелно решење путем $OpenMPI$-ја изузетно ефикасно с просечним временом од само 0.01 секунди. Овај феномен може произаћи из различитих приступа паралелизацији, као и утицаја различитих карактеристика техника управљања паралелним извршавањем. Важно је напоменути да је 50000x8 највећа величина матрице која је могла да се тестира на тестној машини, те би повећање саме матрице могло имати утицај на перформансе алгоритма.



Слика 1: Графикон перформанси у зависности од броја тестирања

На слици 2 је призана завиност времена извршавања $OpenMPI$ имплементације од броја процеца на којим се извршава. Са повећањем броја процеса, можемо приметити да се време извршавања смањује до одређене тачке, након чега се повећава. Ово је у складу са очекивањем, јер паралелна обрада може побољшати перформансе за одређени број процеса, али претерано паралелизовање може изазвати додатне трошкове синхронизације и комуникације између процеса.



Слика 2: Резултати MPI имплементације

Важно је напоменути да ефикасна паралелизација често захтева пажљиво подешавање и оптимизацију како би се смањили трошкови паралелизације и искористиле предности више процесорских ресурса. Овај резултат може послужити као полазна тачка за даље истраживање и оптимизацију паралелне имплементације како би се постигло очекивано побољшање перформанси.

7 Закључак

У раду је проучаван Кули-Тукеј алгоритам и имплементација секвенцијалне верзије, као и паралелне верзије коришћењем $OpenMP$ и $OpenMPI$ технологија. Сукобљени с изазовима рекурзивне природе алгоритма, одабрали смо матрицу података као алтернативну репрезентацију сигнала, омогућавајући ефикасну паралелизацију. Резултати су показали значајне разлике у извршавању између секвенцијалног и паралелног приступа. Важно је напоменути да разлике у перформансама између $OpenMP$ и $OpenMPI$ могу произаћи из различитих техника управљања паралелним извршавањем. Параметри попут величине матрице, степена паралелизације и ефикасности имплементације играју кључну улогу у добивеним резултатима.

Закључујемо да ефикасна паралелизација захтева пажљиво подешавање и оптимизацију како би се смањили трошкови паралелизације и искористиле предности више процесорских ресурса. Ови резултати пружају полазну тачку за даља истраживања и оптимизацију паралелних имплементација с циљем постизања очекиваног побољшања перформанси.

Будући правци истраживања могли би укључивати детаљније анализе утицаја различитих параметара на перформансе, истраживање других техника паралелизације или примену додатних оптимизација. Такође, истраживање могућности примене алгоритма на различитим архитектурама и хардверским платформама допринело би ширем разумевању његове применљивости и ефикасности.

Списак изворних кодова

Списак слика

Библиографија

[1] Fourier transform. https://en.wikipedia.org/wiki/Fourier_transform. Последњи приступ: 16.01.2024.

[2] Openmp. https://hpc-tutorials.llnl.gov/openmp/. Последњи приступ: 16.01.2024.

[3] Openmpi. https://www.open-mpi.org/doc/. Последњи приступ: 16.01.2024.

[4] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. Introduction to algorithms, 2001.