

T.C.
KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ



Paralel ve Dağıtık Sistem Programlama
İşlem Yoğunluğuna Sahip Matrislerin MPI Kütüphanesi ile İşlem Sürelerinin Hesaplanması

Oğulcan Uçarsu
185172014

ÖĞRETİM GÖREVLİSİ
DOÇ. DR. Hikmet Hakan GÜREL

KASIM 2019

1.Problem Tanımı

Matrisler, bilim ve bilimin alt dallarında kullanılan matematiksel hesaplama araçlarından bir tanesidir[1]. Matrislerin çarpımları, dağıtık şekilde farklı çekirdekler üzerinden yapılarak Paralel Programlama çalışma prensibi ve performansı gözlemlenmiştir. İşlemler 1000 - 10000 boyutlu matrisler üzerinden 1000'er arttırarak sağlanmıştır.

2.Yöntem

Paralel Programlama, Windows bir bilgisayarın üzerinde, kurulan Ubuntu işlem sisteminde ki sanal bilgisayarda denenmiştir. Ubuntu işletim sistemi, 'Oracle VM Virtual Box Programı' kullanılarak kurulmuştur. MPI kütüphanesi ile Matris çarpımı yapacak olan Ubuntu Bilgisayar'ın özellikleri şu şekildedir.

- İşletim Sistemi: Ubuntu 64 bit
- Bellek: 8 GB

Ubuntu sanal bilgisayarı kurduktan sonra, Matris çarpımlarını yapabilmek için, 'GCC' derleyicisinin ve 'Open MPI' kütüphanesinin indirilmesi gerekmektedir[2]. Sırası ile terminal ekranına,

- 'sudo apt install gcc' komutu ile GCC derleyicisi indirilir.
- 'sudo apt install mpich' komutu ile Open MPI kütüphanesi ile indirilir.

2.1 Paralel programlama ile Matris çarpımı yapacak olan kodun oluşturulması

Open MPI ile matris çarpımı yapacak olan kod yazılırken, C programlama dili kullanılmıştır. Sebebi ise C programlama dilinin, paralel ve seri programlama için uygun olması hem de MPI vb. paralel programlama kütüphaneleri ile uyumlu çalıştığı için tercih edilmiştir.

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define SIZE 9000

MPI_Status status;

static int first_matrix[SIZE][SIZE],second_matrix[SIZE][SIZE],multiply_matrix[SIZE][SIZE];

int main(int argc, char **argv)
{
    int number_core,core_id,number_workers;
    int source,destination,satir,offset;
    int i,j,k;
    double start_time, end_time;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &core_id);
    MPI_Comm_size(MPI_COMM_WORLD, &number_core);
```

Şekil – 1 Paralel Programlama ile Matris çarpımı yapacak olan kodun değişkenlerinin tanımlanması

İlk önce Open MPI kütüphanesini kullanabilmek için, 'include<mpi.h>' kodu yazılarak Open MPI kütüphanesi projeye import edilir. Daha sonra matris çarpımı yapacak olan matrisler, matris boyutları ve değişkenler tanımlanır. 'MPI' ile başlayan kod parçaları ise Paralel programlama yapabilmek için, Open MPI kütüphanesinin ayağa kalkmasını sağlamaktadır. 'Init' komutu, Open MPI

kütüphanesini kullanan bütün kodlarda bulunması zorunludur. Kütüphaneyi ayağa kaldırıyor diyebiliriz. ‘rank’ komutu ise ‘processlere’ sıra vermektedir. ‘size’ komutu ise ‘processlerinin’ sayısını verir[3,4].

```
if(number_core > 1){
    number_workers = number_core-1;
} else {
    printf ("Lutfen cekirdek sayisini arttiriniz.\n");
}
```

Şekil – 2 ‘core’ sayılarının kontrol edilmesi

Matris çarpımının birden fazla çekirdek üzerinde yapılması için ‘core’ sayısının kontrolü yapılır.

```
if (core_id == 0) {
    for (i=0; i<SIZE; i++) {
        for (j=0; j<SIZE; j++) {
            first_matrix[i][j]= rand()%21+10;
        }
    }
    for (i=0; i<SIZE; i++) {
        for (j=0; j<SIZE; j++) {
            second_matrix[i][j]= rand()%21+10;
        }
    }
    satir = SIZE/number_workers;
    offset = 0;
    start_time = MPI_Wtime();
    printf("Basladi. Core sayisi: %d , Boyut: %d \n",number_core,SIZE);
    for (destination=1; destination<=number_workers; destination++)
    {
        MPI_Send(&offset, 1, MPI_INT, destination, 1, MPI_COMM_WORLD);
        MPI_Send(&satir, 1, MPI_INT, destination, 1, MPI_COMM_WORLD);
        MPI_Send(&first_matrix[offset][0], satir*SIZE, MPI_INT,destination,1, MPI_COMM_WORLD);
        MPI_Send(&second_matrix, SIZE*SIZE, MPI_INT, destination, 1, MPI_COMM_WORLD);
        offset = offset + satir;
    }
    for (i=1; i<=number_workers; i++)
    {
        source = i;
        MPI_Recv(&offset, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
        MPI_Recv(&satir, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
        MPI_Recv(&multiply_matrix[offset][0], satir*SIZE, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
    }
}
```

Şekil – 3 Master çekirdeğin matrisleri oluşturmaları ve görevleri işçi çekirdeklere dağıtması

Id’si sıfır olan çekirdek, master(ana) çekirdek kabul edilerek, matris çarpımı yapacak olan değerlere ‘random’ değerler atayarak, diğer görev yapacak olan çekirdeklere ilk ‘for’ döngüsünün içerisinde çarpacak oldukları matris satırları gönderilir. 2. ‘for’ döngüsünde ise, görevlerini tamamlayan işçi çekirdeklerden matris çarpım sonuçlarının değerleri alınarak işlem tamamlanır[3,4].

```
if (core_id > 0) {
    source = 0;
    MPI_Recv(&offset, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&satir, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&first_matrix, satir*SIZE, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&second_matrix, SIZE*SIZE, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
    for (i=0; i<SIZE; i++){
        for (j=0; j<SIZE; j++){
            for (k=0; k<SIZE; k++){
                multiply_matrix[i][j] += first_matrix[i][k] * second_matrix[k][j];
            }
        }
    }
    MPI_Send(&offset, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Send(&satir, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Send(&multiply_matrix, satir*SIZE, MPI_INT, 0, 2, MPI_COMM_WORLD);
}
```

Şekil – 4 İşçi çekirdeklerin, kendilerine verilen matris çarpımı görevini yapan kod

Master(Ana) çekirdekten görevler dağıtıldıktan sonra işçi çekirdekler sırası ile kendilerine verilen görevleri gerçekleştirir.

Bütün işlemler bittikten sonra ise, MPI kütüphanesinin 'finalize' komutu ile kod sonlandırılır[3,4].

```
    printf ("Lutfen cekirdek sayi\n");
}
MPI_Finalize();
}
```

Şekil – 5 MPI 'finalize'komutunun kullanımı

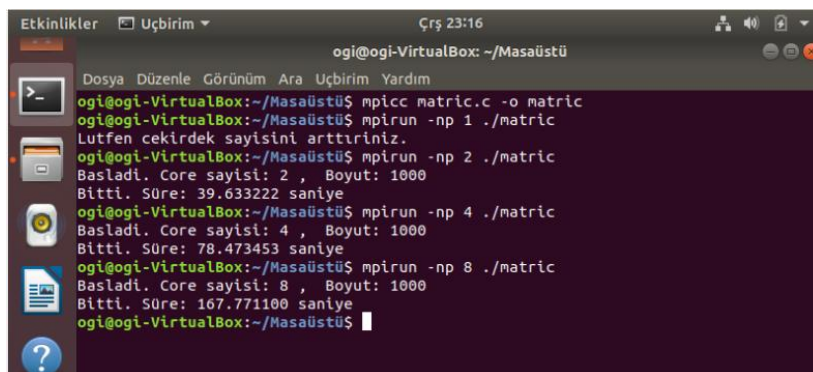
2.2 Paralel programlama ile çarpılan matrislerin zamansal sonuçları

Aşağıda 1000*1000'den 10000*10000'e kadar 1000'er 1000'er artan matrislerin çarpım sonuçlarının değerlendirilmesi yapılacaktır. Yazılan kodun çalıştırılması için ilk önce derlenmesi gerekmektedir. Derledikten sonra kaç çekirdekte çalışacağını belirterek programı çalıştırırız.

- 'mpicc matric.c -o matric' kodu derler.
- 'mpirun -np 2 ./matric' programı çalıştırır. 2 parametresi kaç çekirdek üzerinde çalışılacağını belirtir. 2, 4, 8, 16 parametreleri verilebilir.

En performanslı matris çarpımının kaç tane 'core' üzerinde çalıştığını bulmak için 1000*1000 matrisinde 2, 4 ve 8 'core' çalıştırılmıştır.

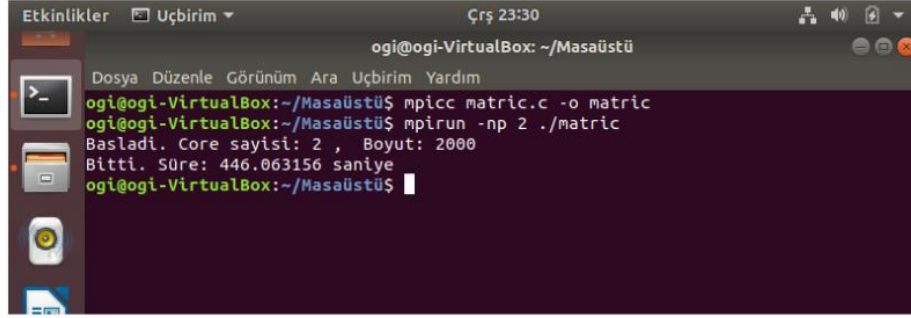
- 1000*1000 matris
 - En hızlı çarpım 2 'core' üzerinde olmuştur. En yavaş ise 8 'core' üzerinde olmuştur.
 - 2 'core' -> 39.6 saniye
 - 4 'core' -> 78.4 saniye
 - 8 'core' -> 167.7 saniye sürmüştür.



```
ogi@ogi-VirtualBox: ~/Masaüstü
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 1 ./matric
Lutfen cekirdek sayisini arttiriniz.
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 1000
Bitti. Süre: 39.633222 saniye
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 4 ./matric
Basladi. Core sayisi: 4 , Boyut: 1000
Bitti. Süre: 78.473453 saniye
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 8 ./matric
Basladi. Core sayisi: 8 , Boyut: 1000
Bitti. Süre: 167.771100 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 5 1000*1000 işlem süresi

- 2000*2000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 446.06 Saniye’de işlem tamamlanmıştır.



```
ogi@ogi-VirtualBox: ~/Masaüstü
Dosya Düzenle Görünüm Ara Uçbirim Yardım
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 2000
Bitti. Süre: 446.063156 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

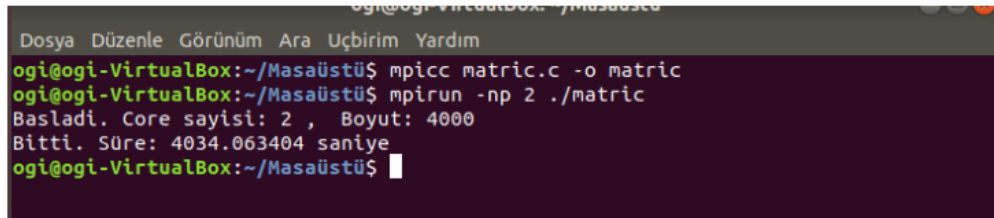
Şekil – 6 2000*2000 İşlem süresi

- 3000*3000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 1739.51 Saniye’de işlem tamamlanmıştır

```
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 3000
Bitti. Süre: 1739.512950 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 7 3000*3000 İşlem süresi

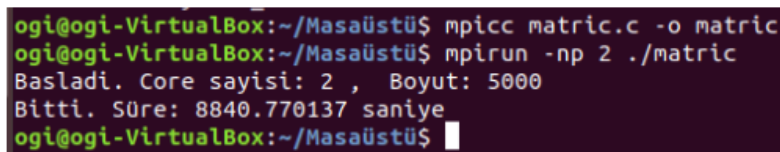
- 4000*4000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 4034.06 Saniye’de işlem tamamlanmıştır.



```
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 4000
Bitti. Süre: 4034.063404 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 8 4000*4000 İşlem süresi

- 5000*5000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 8840.77 Saniye’de işlem tamamlanmıştır.



```
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 5000
Bitti. Süre: 8840.770137 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 9 5000*5000 İşlem süresi

- 6000*6000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 14.903,97 Saniye’de işlem tamamlanmıştır.

```
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 6000
Bitti. Süre: 14903.970491 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 10 6000*6000 İşlem süresi

- 7000*7000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 25.461,29 Saniye’de işlem tamamlanmıştır.

```
Dosya Düzenle Görünüm Ara Uçbirim Yardım
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 7000
Bitti. Süre: 25461.299881 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 11 7000*7000 İşlem süresi

- 8000*8000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 38631.96 Saniye’de işlem tamamlanmıştır.

```
ogi@ogi-VirtualBox: ~/Masaüstü
Dosya Düzenle Görünüm Ara Uçbirim Yardım
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 8000
Bitti. Süre: 38631.968185 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 12 8000*8000 İşlem süresi

- 9000*9000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 56.388,97 Saniye’de işlem tamamlanmıştır.

```
ogi@ogi-VirtualBox: ~/Masaüstü
Dosya Düzenle Görünüm Ara Uçbirim Yardım
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 9000
Bitti. Süre: 56388.978568 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 13 9000*9000 İşlem süresi

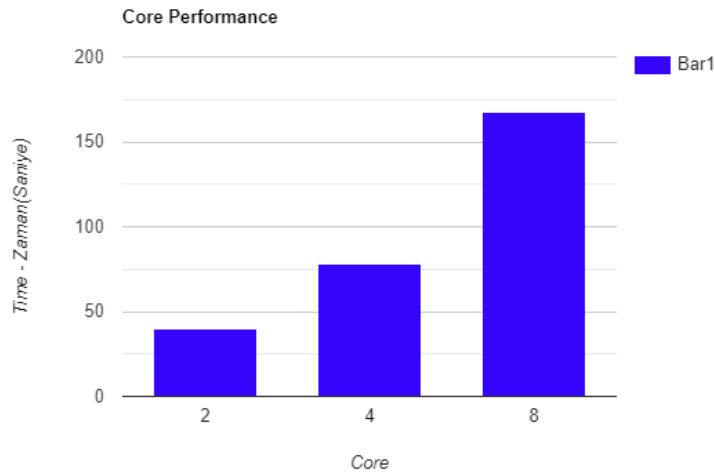
- 10000*10000 matris
 - İşlem süresi aşağıda ki şekilde gösterilmiştir. 75.808,15 Saniye’de işlem tamamlanmıştır.

```
ogi@ogi-VirtualBox:~/Masaüstü$ mpicc matric.c -o matric
ogi@ogi-VirtualBox:~/Masaüstü$ mpirun -np 2 ./matric
Basladi. Core sayisi: 2 , Boyut: 10000
Bitti. Sure: 75808.156162 saniye
ogi@ogi-VirtualBox:~/Masaüstü$
```

Şekil – 14 10000*10000 İşlem süresi

2.2 MPI ile Paralel programlama yaparken çekirdek sayısının etkisi

Linux ortamında MPI kütüphanesi ile matris çarpımı yaparken, en performanslı çarpımın kaç çekirdekte çalıştığını bulmak için, 1000*1000 boyutunda ki matrisleri çarparken 2, 4 ve 8 çekirdek üzerinde çalıştırılarak zamanları ölçüldü.



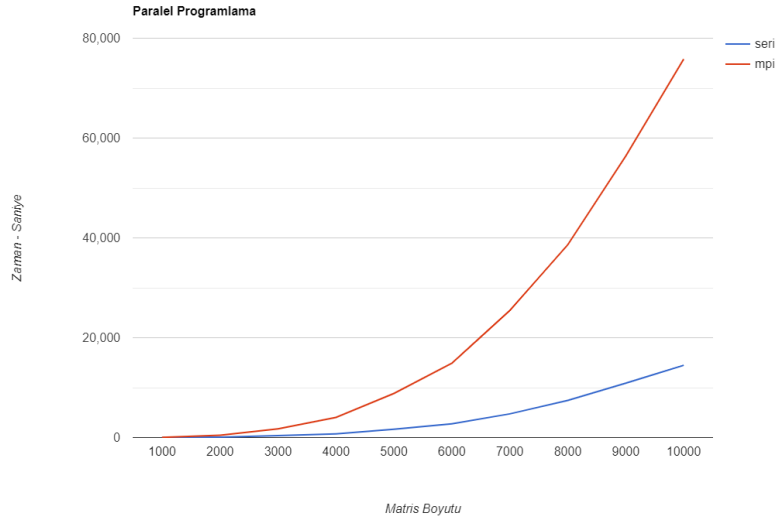
Şekil – 15 Çekirdek sayına göre zaman grafiği

Yapılan çalışma ile en performanslı matris çarpımının 2 çekirdekte olduğu anlaşıldı. Şekil – 15’de bu durum incelenebilir. Çekirdek sayısı arttıkça, zamanda çekirdek sayısına göre doğrusal artmaktadır.

2.3 Paralel programlama ile Seri Programların karşılaştırılması

MPI kütüphanesi ile Paralel programlama yapmadan önce yapılan çalışmalarda, seri programlama ile matris çarpımlarının zamanları ölçülmüştü. Seri programlama ve Paralel programlama ile yapılan matris çarpımlarında, seri programlama daha hızlı çalışmıştır. Sebebi ise Paralel programlama yaptığımız Linux ortamı, Windows bilgisayara üzerine kurulmuş sanal bir işletim sistemidir. Ayrıca Linux bilgisayara ayırmış olduğumuz bellek Windows bilgisayara göre daha düşüktür. Ayrıca MPI kütüphanesinde çekirdekler arasında işlem yaparken haberleşme zamanı da olduğu için performans anlamında Seri programlama daha hızlı çalışmıştır.

Küçük boyutlu matrislerde, performans anlamında, zaman aralıkları birbirine yakındır. Fakat matris boyutu büyüdükçe arada ki zaman farkı 'eksponansiyel' olarak büyümektedir. Seri ve Paralel programlamayı karşılaştırdığımız matris boyutu – zaman grafiği aşağıda ki şekildedir.



Şekil – 16 Seri ve Paralel Programlamanın zamansal karşılaştırılması

Referanslar

1. Young C.Y. Precalculus. John Wiley Sons, 2010.
2. <https://www.open-mpi.org/faq/?category=mpi-apps>
3. <http://www.csharpnedir.com/articles/read/?id=473>
4. <http://www.csharpnedir.com/articles/read/?id=481>

