KARADENİZ TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

PARALEL BİLGİSAYAR ÖDEVİ



**MANDELBROT KÜMESİNİN PARALEL HESAPLANMASI VE GÖRSELLEŞTİRİLMESİ**

FEYZULLAH YILDIZ

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

BAHAR 2015

KARADENİZ TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

PARALEL BİLGİSAYARLAR ÖDEVİ

**MANDELBROT KÜMESİNİN PARALEL HESAPLANMASI VE GÖRSELLEŞTİRİLMESİ**

259275 FEYZULLAH YILDIZ

DERSİN SORUMLUSU: PROF. DR. CEMAL KÖSE

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

GÜZ 2015

**ÖNSÖZ**

Bu raporun ilk taslaklarının hazırlanmasından raporun son halini almasına dek yol gösterici olan ve bu ödevi bizlere veren kıymetli hocamız Sayın Prof. Dr. Cemal KÖSE ‘ye şükranlarımı sunarım. Yapılan proje ile paralel programlama bilgi ve becerisi kazanmış bulunduk. Ayrıca OpenCL ile grafik işlemcisini kullanmanın ilk adımlarını attık.

Feyzullah YILDIZ

Trabzon, 2015

**İÇİNDEKİLER**

ÖNSÖZ II

İÇİNDEKİLER III

ÖZET IV

1. GİRİŞ 1

2. KISITLAR 2

3. ÇALIŞMA PLANI 3

4. PROJE YÖNETİMİ 4

4.1. PARALEL PROGRAMLAMA 4

4.2. OPENCL 4

4.3. MANDELBROT KÜMESİ 4

4.3.1. Sıralı Program 4

4.4. SIMD 5

4.4.1. SIMD Nedir? 5

4.4.2. Yazılan SIMD Programı 5

4.5 MIMD 6

4.5.1. MIMD Nedir? 6

4.5.2. Yazılan MIMD Programı 6

4.6 Hem SIMD Hem MIMD 7

4.6.1. SIMD ve MIMD Programı 7

4.7. Zaman Hızlanma ve Verimlilik Bilgilerinin Elde Edilmesi 8

4.8. VERİLERİN GRAFİKLEŞTİRİLMESİ 12

4.8.1. Zaman Grafikleri 12

4.8.2. Hızlanma Grafikleri 15

4.8.3. Verimlilik Grafikleri 18

5. SONUÇLAR. 20

6. KAYNAKÇA 21

**ÖZET**

İnsanlar tarih boyunca yaptıkları işin kısa sürede sona ulaşması için çeşitli taktikler geliştirmişlerdir. Mesela bir çay fabrikasında yaş çay önce soldurma işlemine tabi tutulur sonra buradan çıkan çaylar kesme işlemine tabi tutulmak için ilgili bölüme aktarılır. Ardından kıvırtma mayalanma ve fırınlama işlemleri ile kuru çay elde edilir. Fakat birinci aşama olan soldurma işleminden çıkan çaydan sonra onun kuru çay olması beklenilmeden sıradaki çaylar soldurma işlemine tabi tutulur. Böylelikle fabrikada bütün aşamalar paralel yapılarak daha hızlı kuru çay üretimi sağlanır. Aynen bunun gibi bilgisayar programcılığında da işlerin daha hızlı yapılabilmesi için paralel programlama yapılmaktadır. Bu sayede bir program alt parçalara bölünerek farklı işlemcilerde koşulup performans arttırılır.

Bu projede amaç mandelbrot kümesini SIMD, MIMD, hem SIMD hem MIMD algoritmaları kullanılarak görselleştirip paralel programlamanın avantajını görmek ve farklı paralel progralama algoritmalarının birbirlerine kıyaslanması sağlanacaktır.

**1.GİRİŞ**

Bu projeye öncelikle OpenCL öğrenerek başladım. OpenCL ile grafik işlemci kullanılarak paralel programlama sağlanmaktadır. Bu sayede ekrana bir görüntü basılmak istendiğinde grafik işlemcideki hesaplanan sonuçların ekrana basılması daha hızlı olmaktadır. OpenCL çeşitli diller ile kullanılabilmektedir. Ben projede C programlama dilini tercih ettim.

Projenin yazıldığı sistemin özellikleri:

* Ubuntu 14.04 OS
* Çift Çekirdek Intel Core i5 işlemci
* CUDA destekli Nvidia GeForce Grafik Kartı
* 640GB HDD
* 4GB RAM
* OpenCL için Nvidia CUDA yazılımı
* gcc Derleyici

Benim bilgisayarımın özellikleri bu şekildeydi. Fakat grafik kartı OpenCL uyumlu olmasına rağmen tek çekirdekli olması beni şaşırttı. Bu sistem üzerinde projeyi geliştirip tamamladım.

**2. KISITLAR**

Paralel programlamanın getirdiği avantajlarının yanısıra birde kısıtlamaları mevcuttur. Paralel programlarda oluşturulan çocuk süreçler sonlanmadan ana sürecin sonlanması durumunda çocuk süreçler ölecek ve program yanlış sonuç üretecektir. Bunu engellemek için süreçler arası senkronizasyon sağlanmalıdır. Ana süreçte wait() metodu kullanılırsa bir çocuk sürecin sonlanmadan ana süreçte sonlanmayacaktır. Fakat birden fazla çocuk süreç olduğunda ya süreçler arası haberleşme yapılacak ya da birden fazla wait() metodu kullanılarak çocuk süreçlerin beklenilmesi sağlanacaktır. Bu programa ek yük getirse bile kullanmak zorundayız.

Diğer bir kısıtlama ise süreçler arası haberleşmedir. C dilinde global değişkenleri bütün süreçler görüyor fakat bir sürecin global bir değişken üzerinde yaptığı değişikliği diğer süreçler görememektedir. Bunun için süreçler arası haberleşmeyi sağlamak için pipe() metodunu kullanmaktayız. Bu yöntem ile işletim sistemi aracılığı ile süreçleri haberleştirebiliyoruz.

**3. ÇALIŞMA TAKVİMİ**

Aşağıda projeye başlarken hazırladığım çalışma planı grafiği bulunmaktadır.



Resim-1: Çalışma Planı

**4. PROJE YÖNETİMİ**

**4.1. PARALEL PROGRAMLAMA**

Paralel programlama, aynı anda birden fazla işlemin yapılması veya programın zaman paylaşımlı olarak çalışmasıdır. Zaman paylaşımlı sistemler genel tek işlemcili bilgisayarlarda kullanılmaktadır. Birden fazla çekirdek veya işlemci bulunan bilgisayaralar ve birden fazla bilgisayarın birbirine bağlanmasıyla elde edilen paralel sistemler ise fiziksel olarak aynı anda birden fazla işlem yapma yeteneğine sahiptirler.

**4.2. OPENCL**

OpenCL grafik işlemci ve genel amaçlı işlemciyi kullanarak parallelliği daha etkin hale getirmek için ortaya çıkmıştır. OpenCL programlama mantığı C dilini baz alarak geliştirilmiştir. OpenCL programlamada dinamik dizi, pointer fonksiyonlar ve recursion kaldırılmıştır. Vektör veri sınıfı ve bazı hafıza sıfatları eklenmiştir.

Grafik işlemcideki çekirdek sayısı arttıkça OpenCL programının verimliliği de artmaktadır. Özellikle SIMD türünde paralelleştirme bu yöntem kullanılarak çok verimli hale getirilebilir. Ayrıca burada hesaplamalar grafik işlemcide yapıldığından sonuçların görselleştirilmesi daha hızlı olacaktır.

**4.3. MANDELBROT KÜMESİ**

Mandelbrot kümesi Benoit Mandelbrot tarafından geliştirilen bir kümedir. Karmaşık z sayısının dinamikliğini açıklamak için ortaya atılmıştır. Bunun için başlangıçta bir c sayısı belirlenir ve belli bir iterasyon boyunca fonksiyon tekrar tekrar hesaplanır. Hesaplamada z = z^2 + c fonksiyonu kullanılmaktadır. Bu fonksiyon ile bir pikselin mandelbrot kümesine dahil olup olmadığı pikselin belli bir iterasyon boyunca (programda iterasyon sayısını 200 olarak belirledim) belirlenen daire çapının(programda 2) içerisine dahil olup olmadığı kontrol edilir. Eğer dahil ise o piksel siyah renge boyanır. Değilse beyaz renge boyanır ve bu şekilde mandelbrot kümesi çizdirilmiş olunur.

**4.3.1. Sıralı Program**

Sıralı program, emirlerin icrasının sıralı şekilde gerçekleştirilmesidir. Paralel programlamaya göre dezavantajları vardır. Fakat senkronizasyon gerektirmediği için hem program karmaşıklığı azdır hem okunabilirliği fazladır. Mandelbrot kümesini hesaplayan sıralı C programı ek-1'de verillmiştir.

**4.4. SIMD**

Paralel programlamanın çeşitli türleri vardır. Bunlardan birisi de SIMD programlamadır.

**4.4.1. SIMD Nedir?**

SIMD operasyonu aynı emri farklı işlemcilerde farklı datalar üzerinde uygulamaktır. Bu şekilde paralel olarak yapılan programlar seri programlara göre daha verimli olmaktadır. Fakat işlemciler arası haberleşme maliyetinin yüksek olduğu bazı durumlarda verimli olmamaktadır.

**4.4.2. Yazılan SIMD Programı**

Mandelbrot hesaplamasını C dilinde yaptım. SIMD türü paralelleştirme yapabilmek için fork() kullanarak aynı işi farklı süreçlere paylaştırdım. Sıralı programda yapılan işleri mandelbrot() adında bir fonksiyona yazdım ve bu fonksiyona çeşitli parametreler vererek her bir aralığın bir süreç tarafından yapılmasını sağladım. Süreç sayısını dinamik olarak kullanıcıdan alıp hesaplama yapıyorum.

int ar = 1000 / is;

int bs = 0;

int bts = ar;

for(i = 0; i < is; i++)

{

if(fork()==0){

mandelbrot(bs, bts,1000,1000,fp[i],i);

exit(1);

}

bs = bts;

bts = bts + ar;

}

Programda 1000x1000 çözünürlüğünde bir görüntü elde edilmektedir. Bu programda maksimum 1000 süreç olabilir. Çünkü her bir satırı bir süreç hesaplayacak şekilde ayarladım. Eğer 1000 den daha az süreç sayısı seçilirse bir süreç birden fazla satırı hesaplayıp ekrana basacaktır. Programda fp[] dizisi herbir süreç için dosya gösterici tutmaktadır. Program paralel icra edildiğinden her süreç farklı bir dosya gösterici kullanmaktadır. Fonksiyona verilen birinci parametre hangi satırdan hesaplamaya başlanacağını göstermekte. İkinci parametre ise hangi satıra kadar devam edileceğini göstermektedir. Üçüncü ve dördüncü parametreler çözünürlüğü belirtmektedir. Beşinci parametre dosya gösterici ve altıncı parametre ise yapılan işin kaçıncı iş olduğunu belirtmektedir.

**4.5. MIMD**

**4.5.1. MIMD Nedir?**

MIMD operasyonunda program alt parçalara bölünür ve her süreç programın başka bir alt kısmını icra etmektedir. Böylelikle programın farklı kısımları aynı anda koşabilmektedir.

**4.5.2. Yazılan MIMD Programı**

Mandelbrot programını hesaplama ve yazma olarak iki alt parçaya böldüm. Bir süreç hesaplama yaparken diğer bir süreç yazdırma işlemini yapmaktadır. Burada önemli olan hesaplama yapılmadan yazmaya çalışılmamasıdır. Bu yüzden hesaplama yapan süreç yazan sürece mesaj yollayarak hesaplamayı yaptığını ve yazmaya hazır olduğunu belirtir. Bu sayede senkronizasyon sağlanmış olmaktadır.

if(fork() == 0) hesapla(); else yazdir();

Programda bu şekilde kullanım ile çocuk süreç hesaplamayı ana süreç ise yazdırma işlemini yapmaktadır. Süreçler arası haberleşme için ise pipe mantığı kullanılmıştır. İki adet pipe belirlenmiştir. Birisi ile senkronizasyon sağlanmakta diğeri ise hesaplanan değeri yazıcıya göndermektedir. Pipe değişkenleri global tanımlanmıştır.

int senkpipe[2];

char buf[30];

int sncpipe[2];

char buf2[3000];

Süreçler başlatılmadan önce pipe fonksiyonu ile bu diziler iletişim için kullanıma aktif hale getirilmiştir.

pipe(senkpipe);

pipe(sncpipe);

Hesaplama yapan süreç hesaplama sonucunu ve senkronizasyon bilgisini write fonksiyonu ile şu şekilde yazdırıcıya göndermektedir.

write(senkpipe[1], "bitti", 5);

write(sncpipe[1], color, 3000);

Yazdırıcı süreç ise bu değerleri beklemektedir. Burada eğer senkronizsyon değerinden okunan değer “devam” kelimesi ise hesaplamanın devam ettiği henüz sonuç üretmediği anlamındadır. Sonuç üretilene kadar yazdırıcı süreç kendisini while döngüsü ile bloklamaktadır. Ve sonuç üretildiğinden while döngüsünden kurtulup gelen değeri dosyaya yazdırmaktadır. Eğer p değeri 1000 olduysa bütün satırlar hesaplanmıştır ve program sonlandırılır.

while(1)

{

read(senkpipe[0], buf, 5);

while(!strcmp(buf,"devam")) read(senkpipe[0], buf, 5);

if(p == 1000) break;

write(senkpipe[1], "devam", 5);

read(sncpipe[0], buf2, 3000);

fseek (fp2 , fps, SEEK\_SET );

fwrite(buf2,1,3000,fp2);

fps+=3000;

++p;

}

MIMD programının tamamı ek-2'de verilmiştir.

**4.6. HİBRİT SİSTEM**

Bu programda hem program alt parçalara bölünecek hemde bu alt parçalar birden fazla süreç tarafından paralel olarak yapılacaktır.

**4.6.1. Hibrit Program**

Burada MIMD programında olduğu gibi program hesaplayıcı ve yazıcı olarak iki alt parçaya bölünmüştür. Bu alt parçalarda ayrıca paralelleştirilmiştir. Süreç sayısı dinamik olarak kullanıcıdan alınmaktadır. Burada yine süreçler arası iletişim için pipe kullanılmıştır. Paralelleştirme şu şekilde yapılmaktadır.

for(j=0;j<is;j++)

{

max = min + adim;

if(fork() == 0) { hesapla(min, max,\_ortak[j], \_senk[j]); exit(1); }

if(fork() == 0) { yazdir(fp2[j], adim, \_ortak[j], \_buf[j], \_senk[j], \_senkbuf[j], fps, pfds[j]); exit(1); }

fps = fps + 3000\*(1000/is);

min = min + adim;

}

Burada SIMD programında olduğu gibi min ve max değişkenleri hesaplama yapılacak aralığı belirliyor. Fps değeri ise herbir sürecin dosyaya nereden yazmaya başlayacağını belirtmektedir. Bu sayeye her süreç dosyada yazması gerektiği alana yazma yapmaktadır.

**4.7. Zaman Hızlanma ve Verimlilik Bilgilerinin Elde Edilmesi**

3 farklı paralelleştirme algoritmasının zaman hızlanma ve verimlilik grafiklerini elde ederek programların analizini sağlamayı amaçladım. Elde ettiğim verileri aşağıdaki şekilde tablolaştırdım.

SIMD sistem için fork() ile oluşturulan süreç sayısı baz alınarak tablo oluşturulmuştur. Burada 1000x1000 çözünürlükte bir görüntü elde edilmekte ve her süreç görüntünün bir satırını hesaplayıp yazmaktadır. Bundan dolayı süreç sayısı maksimum 1000 olmaktadır.

Hızlanma ve Verimlilik bilgileri şu şekilde elde edilmektedir.

**Hızlanma = Seri Çalışma Süresi / Paralel Çalışma Süresi**

**Verimlilik = Hızlanma / Çekirdek Sayısı**

Hızlanma bilgisi ile seri yöntemin paralele göre kaç kat hızlı olduğu bulunur. Bu bilgi ne kadar büyükse seri program paralel programa göre o kadar hızlıdır. En düşük olduğu değer ise paralel programın en hızlı olduğu yerdir.

Verimlilik bilgisi ise bir çekirdeğin birim zamanda yaptığı ortalama iştir. Yani bu değere bakılarak paralel veya seri program tercih edilebilir. Benim sistemimde 2 çekirdek olduğu için hızlanma bilgisini ikiye bölerek verimliliği bulmaktayım.

Tablolarda en düşük ve en yüksek değerler koyu italik şeklinde vurgulanmıştır.

,

|  |  |  |  |
| --- | --- | --- | --- |
| **SIMD Sistem için Elde Edilen Veriler** | | | |
| Süreç Sayısı | Zaman (Sn.) | Hızlanma | Verimlilik |
| 2 | 1,250 | 0,160 | 0,08 |
| 4 | 1,147 | 0,175 | 0,087 |
| 6 | 1,084 | 0,185 | 0,032 |
| 8 | 1,102 | 0,182 | 0,091 |
| 10 | 1,072 | 0,187 | 0,093 |
| 20 | 1,043 | 0,192 | 0,096 |
| 30 | 1,031 | 0,194 | 0,097 |
| 40 | 1,045 | 0,192 | 0,096 |
| 50 | 1,029 | 0,195 | 0,098 |
| 60 | 1,998 | 0,201 | 0,1 |
| 70 | 1,031 | 0,194 | 0,097 |
| 80 | 1,009 | 0,199 | 0,099 |
| 90 | 1,045 | 0,192 | 0,096 |
| 100 | 1,052 | 0,191 | 0,095 |
| 250 | 1,087 | 0,184 | 0,092 |
| 500 | 1,139 | 0,176 | 0,088 |
| ***510*** | ***0,630*** | ***0,319*** | ***0,159*** |
| 750 | 0,975 | 0,206 | 0,103 |
| ***1000*** | ***1,488*** | ***0,141*** | ***0,071*** |

|  |  |  |  |
| --- | --- | --- | --- |
| **MIMD Sistem için Elde Edilen Veriler** | | | |
| Çözünürlük | Zaman (Sn.) | Hızlanma | Verimlilik |
| 25x25 | 0,002 | 1 | 0,5 |
| 50x50 | 0,002 | 1 | 0,5 |
| ***100x100*** | ***0,004*** | ***1,33*** | ***0,665*** |
| 125x125 | 0,005 | 1,25 | 0,625 |
| 180x180 | 0,008 | 1,14 | 0,57 |
| 200x200 | 0,009 | 1 | 0,5 |
| 250x250 | 0,014 | 1 | 0,5 |
| 300x300 | 0,020 | 1 | 0,5 |
| 340x340 | 0,025 | 1,04 | 0,52 |
| 390x390 | 0,033 | 1,03 | 0,51 |
| 450x450 | 0,043 | 1,04 | 0,52 |
| 600x600 | 0,077 | 1,11 | 0,55 |
| ***640x640*** | ***0,085*** | ***0,94*** | ***0,48*** |
| 720x720 | 0,102 | 1 | 0,5 |
| 800x800 | 0,130 | 1,04 | 0,52 |
| 860x860 | 0,150 | 1,02 | 0,51 |
| 920x920 | 0,180 | 1,1 | 0,55 |
| 1000x1000 | 0,201 | 1,05 | 0,52 |

MIMD sistemde program iki farklı iş yapan alt parçalara bölünmüştür. Bunlardan birisi piksel değerlerini hesaplamakta diğeri ise hesaplanan değerleri dosyaya yazmaktadır. Bundan dolayı MIMD sistemde süreç sayısını değiştiremediğim için çözünürlüğü değiştirerek farklı değerler elde etmeyi amaçladım. Burada çözünürlük arttıkça iki süreç arası haberleşme de arttığından işlem süre lineer olarak artmamıştır.

|  |  |  |  |
| --- | --- | --- | --- |
| **Hibrit Sistem için Elde Edilen Veriler** | | | |
| Süreç Sayısı | Zaman (Sn.) | Hızlanma | Verimlilik |
| ***2*** | ***0,191*** | ***1,052*** | ***0,526*** |
| 4 | 0,103 | 1,951 | 0,975 |
| 6 | 0,173 | 1,161 | 0,580 |
| 8 | 0,1 | 2,01 | 1,005 |
| 10 | 0,136 | 1,477 | 0,738 |
| 30 | 0,078 | 2,576 | 1,288 |
| 40 | 0,083 | 2,421 | 1,210 |
| ***50*** | ***0,071*** | ***2,830*** | ***1,415*** |
| 60 | 0,076 | 2,644 | 1,322 |
| 70 | 0,078 | 2,576 | 1,288 |
| 80 | 0,078 | 2,576 | 1,288 |
| 90 | 0,075 | 2,680 | 1,34 |
| 100 | 0,078 | 2,576 | 1,288 |
| 150 | 0,084 | 2,392 | 1,196 |
| 250 | 0,092 | 2,184 | 1,092 |
| 290 | 0,092 | 2,184 | 1,092 |

Hibrit sistemde birden fazla süreç hesaplama yapmakta ve birden fazla süreçte dosyaya yazma yapmaktadır. Dolayısıyla birden fazla süreç haberleşme yapmaktadır. Süreçler arası haberleşmeyi pipe yapısı ile sağladığımdan ve sistemin çok fazla pipe kullanıma izin vermemesinden dolayı en fazla 290 süreç kullanabilmekteyim.

Hibrit sistemin verimliliği, SIMD ve MIMD sisteme göre daha yüksek çıkmıştır. Demek mandelbrot programı için en uygun sistem hibrit sistemdir.

**4.8. VERİLERİN GRAFİKLEŞTİRİLMESİ**

**4.8.1. Zaman Grafikleri**

Zaman bilgisi, programın sistem üzerinde kaç saniyede koştuğu bilgisini verir. Programın kaç saniye koştuğunu anlayabilmek için kodun başına ve sonuna sistem saatini nanosaniye cinsinden yazdıran ve bir Linux komutu olan “date +\"%T.%N\” komutunu C kodu içerisinden system("date +\"%T.%N\""); şeklinde çağırdım. Bu şekilde programın başlangıç ve bitiş zamanını öğrenerek toplam koşma zamanını elde ettim.

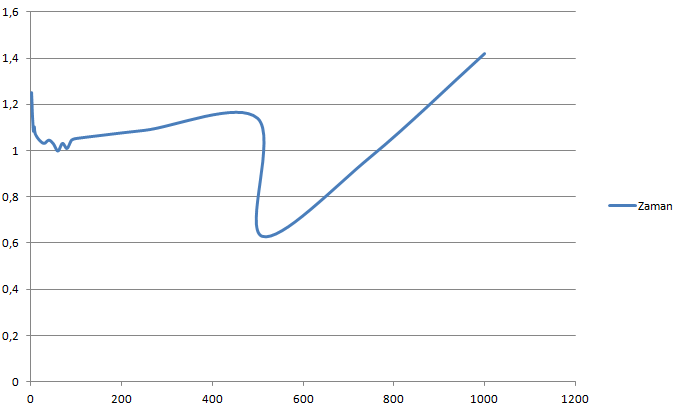
Aşağıda üç sistem için elde ettiğim zaman verilerini kullanarak bunların grafiğini elde ettim.

* **SIMD**

SIMD sistemde her bir process aynı ayna aynı işi yapmaktadır. Yaptıkları iş görüntünün satırlarını oluşturmaktır. Bu programda süreçler arası haberleşme yoktur. Her süreç kendi hesapladığı değeri kendisi dosyaya yazmaktadır. Aşağıda görüldüğü gibi en iyi sonucu 512 süreçte vermiştir. Yani her süreç 2 görüntünün satırından sorumlu olduğunda en iyi sonuç elde edilmiştir. Fakat 512 süreçten sonra zaman sürekli artmaktadır. Görüldüğü gibi süreç sayısının belli bir yerden sonra arttırılmasının bir faydası yoktur.

Y ekseni: Zaman

X ekseni: Süreç Sayısı

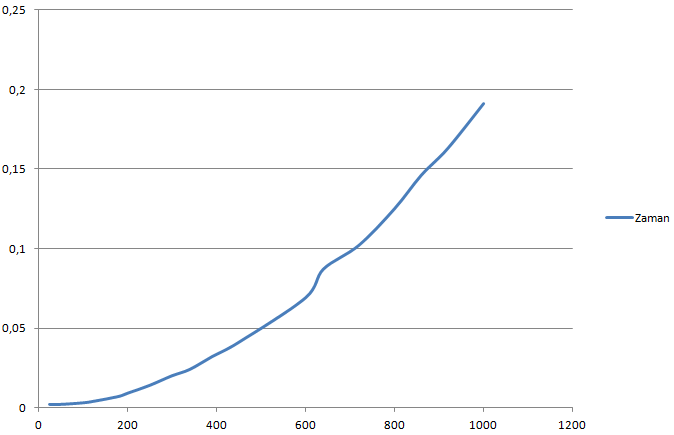


Şekil . SIMD için Zaman - Süreç Sayısı Grafiğ

* **MIMD**

Y ekseni: Zaman

X ekseni: Çözünürlük

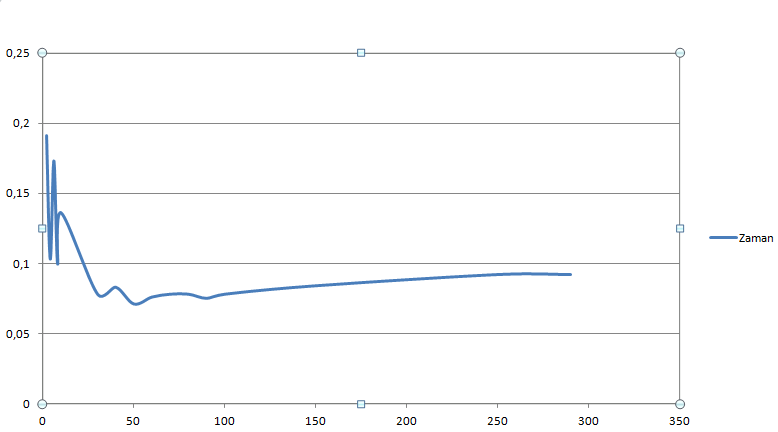
****

Şekil . MIMD için Zaman - Çözünürlük Grafiği

* **HİBRİT**

Y ekseni: Zaman

X ekseni: Süreç Sayısı

****

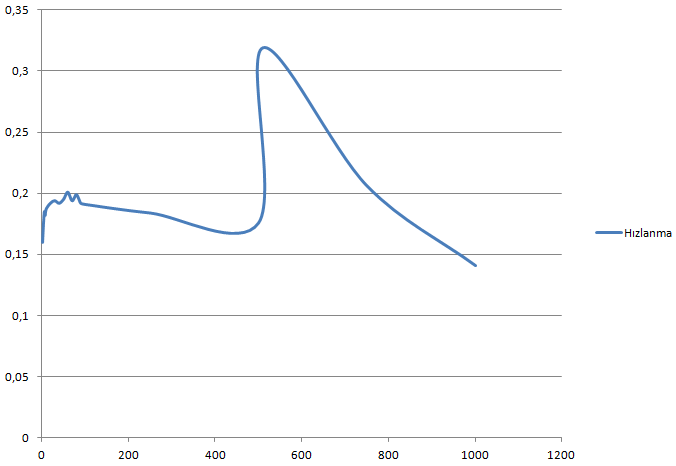
Şekil . Hibrit için Zaman - Süreç Sayısı Grafiği

**4.8.2. Hızlanma Grafikleri**

* **SIMD**

Y ekseni: Hızlanma

X ekseni: Süreç Sayısı

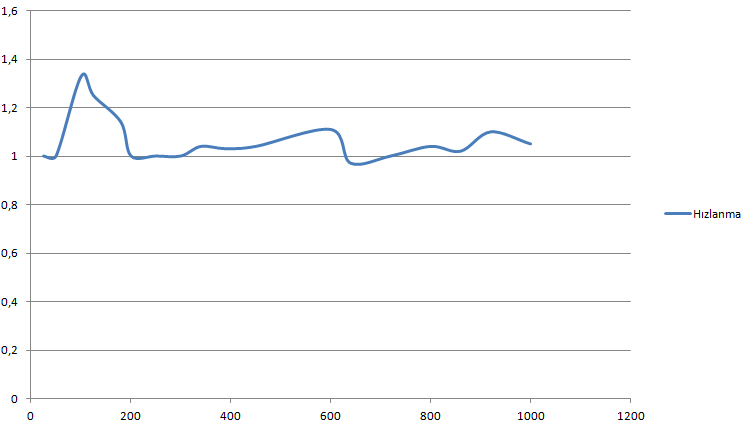


Şekil . SIMD için Hızlanma - Süreç Sayısı Grafiği

* **MIMD**

Y ekseni: Hızlanma

X ekseni: Çözünürlük

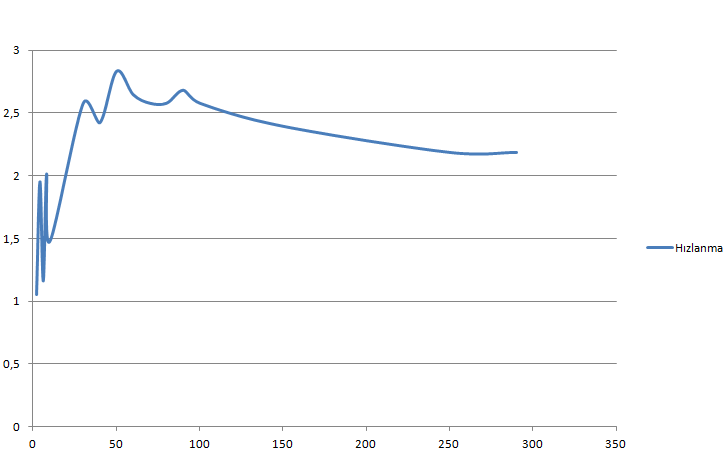


Şekil . MIMD için Hızlanma - Çözünürlük Grafiği

* **HİBRİT**

Y ekseni: Hızlanma

X ekseni: Süreç Sayısı



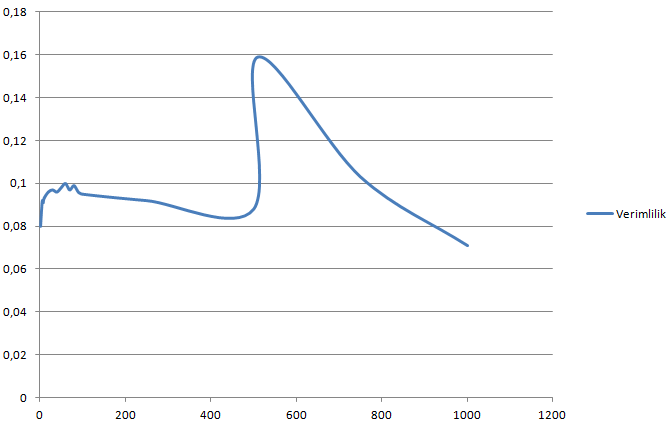
Şekil . Hibrit için Hızlanma - Süreç Sayısı Grafiği

**4.8.3. Verimlilik Grafikleri**

* **SIMD**

Y ekseni: Verimlilik

X ekseni: Süreç Sayısı

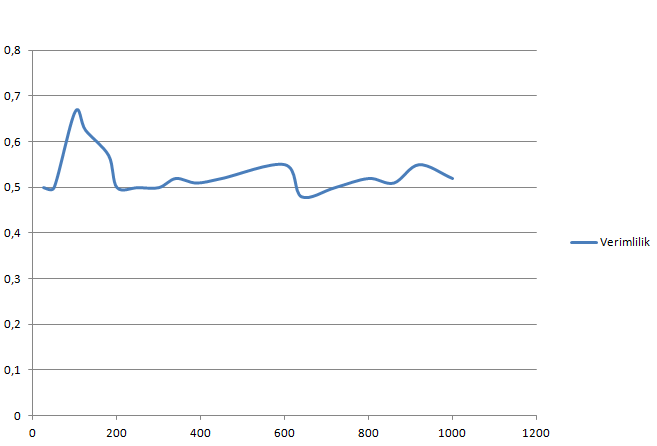


Şekil . SIMD için Verimlilik - Süreç Sayısı Grafiği

* **MIMD**

Y ekseni: Verimlilik

X ekseni: Çözünürlük

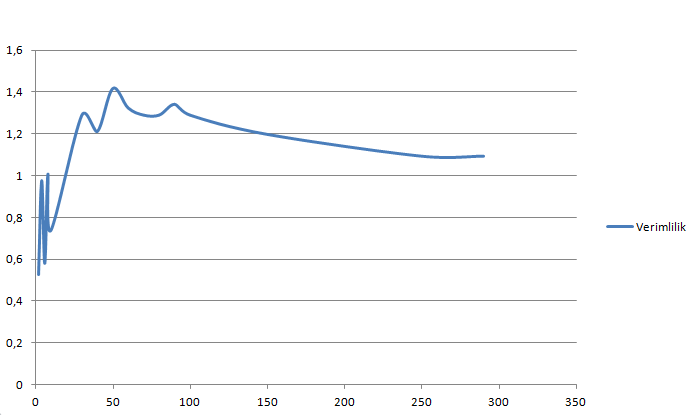


Şekil . MIMD için Verimlilik - Çözünürlük Grafiği

* **HİBRİT**

Y ekseni: Verimlilik

X ekseni: Süreç Sayısı



Şekil . Hibrit için Verimlilik - Süreç Sayısı Grafiği

**5. SONUÇLAR**

Bu proje mesleki gelişimim için çok faydalı olmuştur. Paralel programlamanın temellerini oluşturan yapıları hem öğrenmiş hem de uygulayarak pekiştirmiş oldum.

Projede SIMD ve MIMD yapılarının farklarını kavradım. Eğer programımızı vektörel olarak yazabilirsek SIMD bizim çok avantajlı olmuş olur. Eğer algoritmik veya domain parçalama yapacaksak MIMD daha faydalı olmaktadır. Ayrıca MIMD sisteminde haberleşme ve senkronizasyon sorunlarından dolayı süreç sayısı çok fazla olduğunda verimlilik düşmektedir. Haberleşme ve senkronizasyonun sisteme getirdiği yükü bu projede uygulayarak görmüş oldum.

Her iki yapı birlikte kullanıldığında ise daha yüksek performans elde etmek mümkün olmaktadır. Bunu projede gözlemleme şansım oldu.

Projede verimliliği en yüksek hibrit, verimliliği en düşük SIMD yapısı oldu.

**6. KAYNAKÇA**

1*. Thomas Braunl (1993). Parallel Programming an introduction, Prentice Hall.*

2. *<* *http://tr.wikipedia.org/wiki/Mandelbrot\_k%C3%BCmesi> (2014, Aralık).*

3. *<* *https://developer.nvidia.com/cuda-downloads> (2015).*

4. *<* *http://www.cs.hmc.edu/~keller/cs60book/14%20Parallel%20Computing.pdf >*

*5.< http://tr.wikipedia.org/wiki/OpenCL> (2013, Mayıs)*

**EK-1 ( Sıralı Çalışan Program )**

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <math.h>*

*#include <sys/types.h>*

*int main()*

*{*

*int iX,iY;*

*int cz;*

*printf("Çözürnürlük(AxA): ");*

*scanf("%d",&cz);*

*system("date +\"%T.%N\"");*

*const int iXmax = cz;*

*const int iYmax = cz;*

*double Cx,Cy;*

*const double CxMin=-2.5;*

*const double CxMax=1.5;*

*const double CyMin=-2.0;*

*const double CyMax=2.0;*

*double PixelWidth=(CxMax-CxMin)/iXmax;*

*double PixelHeight=(CyMax-CyMin)/iYmax;*

*const int MaxColorComponentValue=255;*

*FILE \* fp2;*

*char \*filename="sirali.ppm";*

*char \*comment="# ";*

*char color[3];*

*double Zx, Zy;*

*double Zx2, Zy2;*

*int gl = 0;*

*int Iteration;*

*const int IterationMax=200;*

*const double EscapeRadius=2;*

*double ER2=EscapeRadius\*EscapeRadius;*

*fp2= fopen(filename,"wb");*

*fprintf(fp2,"P6\n %s\n %d\n %d\n %d\n",comment,cz,cz,MaxColorComponentValue);*

*for(iY=0;iY<iYmax;iY++)*

*{*

*Cy=CyMin + iY\*PixelHeight;*

*if (fabs((double)Cy)< PixelHeight/2) Cy=0.0;*

*for(iX=0;iX<iXmax;iX++)*

*{*

*Cx=CxMin + iX\*PixelWidth;*

*Zx=0.0;*

*Zy=0.0;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*for (Iteration=0;Iteration<IterationMax && ((Zx2+Zy2)<ER2);Iteration++)*

*{*

*Zy=2\*Zx\*Zy + Cy;*

*Zx=Zx2-Zy2 +Cx;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*};*

*if (Iteration==IterationMax)*

*{*

*color[0]=0;*

*color[1]=0;*

*color[2]=0;*

*}*

*else*

*{*

*color[0]=255;*

*color[1]=255;*

*color[2]=255;*

*};*

*fwrite(color,1,3,fp2);*

*}*

*}*

*fclose(fp2);*

*system("date +\"%T.%N\"");*

*printf("Tamamlandı\n");*

*return 0;*

*}*

**EK-2 (OpenCL Programı)**

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string>*

*#include <iostream>*

*#include <CL/cl.h>*

*#define MEM\_SIZE (750000)*

*#define MAX\_SOURCE\_SIZE (0x100000)*

*int main()*

*{*

*cl\_device\_id device\_id = NULL;*

*cl\_context context = NULL;*

*cl\_command\_queue command\_queue = NULL;*

*cl\_mem memobj = NULL;*

*cl\_program program = NULL;*

*cl\_kernel kernel = NULL;*

*cl\_platform\_id platform\_id = NULL;*

*cl\_uint ret\_num\_devices;*

*cl\_uint ret\_num\_platforms;*

*cl\_int ret;*

*char \*filename="new1.bmp";*

*char \*comment="# ";*

*const int iXmax = 500;*

*const int iYmax = 500;*

*const int MaxColorComponentValue=255;*

*FILE \* fp2;*

*fp2= fopen(filename,"wb");*

*fprintf(fp2,"P6\n %s\n %d\n %d\n %d\n",comment,iXmax,iYmax,MaxColorComponentValue);*

*char string[750000];*

*FILE \*fp;*

*char fileName[] = "./fyz.cl";*

*char \*source\_str;*

*size\_t source\_size;*

*fp = fopen(fileName, "r");*

*if (!fp) {*

*fprintf(stderr, "Failed to load kernel.\n");*

*exit(1);*

*}*

*source\_str = (char\*)malloc(MAX\_SOURCE\_SIZE);*

*source\_size = fread(source\_str, 1, MAX\_SOURCE\_SIZE, fp);*

*fclose(fp);*

*ret = clGetPlatformIDs(1, &platform\_id, &ret\_num\_platforms);*

*ret = clGetDeviceIDs(platform\_id, CL\_DEVICE\_TYPE\_DEFAULT, 1, &device\_id, &ret\_num\_devices);*

*context = clCreateContext(NULL, 1, &device\_id, NULL, NULL, &ret);*

*printf("%d - %d\n", ret\_num\_devices, ret\_num\_platforms);*

*command\_queue = clCreateCommandQueue(context, device\_id, 0, &ret);*

*memobj = clCreateBuffer(context, CL\_MEM\_READ\_WRITE,MEM\_SIZE \* sizeof(char), NULL, &ret);*

*program = clCreateProgramWithSource(context, 1, (const char \*\*)&source\_str,*

*(const size\_t \*)&source\_size, &ret);*

*ret = clBuildProgram(program, 1, &device\_id, NULL, NULL, NULL);*

*kernel = clCreateKernel(program, "feyzullah", &ret);*

*int \*a, \*colormap, nx, ny, offset, lda, maxIt;*

*ret = clSetKernelArg(kernel, 0, sizeof(cl\_mem), (void \*)&memobj);*

*ret = clEnqueueTask(command\_queue, kernel, 0, NULL,NULL);*

*ret = clEnqueueReadBuffer(command\_queue, memobj, CL\_TRUE, 0, MEM\_SIZE \* sizeof(char),string, 0, NULL, NULL);*

*ret = clFlush(command\_queue);*

*ret = clFinish(command\_queue);*

*ret = clReleaseKernel(kernel);*

*ret = clReleaseProgram(program);*

*ret = clReleaseMemObject(memobj);*

*ret = clReleaseCommandQueue(command\_queue);*

*ret = clReleaseContext(context);*

*fwrite(string,1,750000,fp2);*

*fclose(fp2);*

*free(source\_str);*

*return 0;*

*}*

**fyz.cl OpenCL Dosyası**

*\_\_kernel void feyzullah(\_\_global char\* string)*

*{*

*int iX,iY;*

*int iXmax = 500;*

*int iYmax = 500;*

*float Cx,Cy;*

*float CxMin=-2.5;*

*float CxMax=1.5;*

*float CyMin=-2.0;*

*float CyMax=2.0;*

*float PixelWidth=(CxMax-CxMin)/iXmax;*

*float PixelHeight=(CyMax-CyMin)/iYmax;*

*int MaxColorComponentValue=255;*

*float Zx, Zy;*

*float Zx2, Zy2;*

*int gl = 0;*

*int Iteration;*

*int IterationMax=200;*

*float EscapeRadius=2;*

*float ER2=EscapeRadius\*EscapeRadius;*

*for(iY=0;iY<iYmax;iY++)*

*{*

*Cy=CyMin + iY\*PixelHeight;*

*if (fabs(Cy)< PixelHeight/2) Cy=0.0;*

*for(iX=0;iX<iXmax;iX++)*

*{*

*Cx=CxMin + iX\*PixelWidth;*

*Zx=0.0;*

*Zy=0.0;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*for (Iteration=0;Iteration<IterationMax && ((Zx2+Zy2)<ER2);Iteration++)*

*{*

*Zy=2\*Zx\*Zy + Cy;*

*Zx=Zx2-Zy2 +Cx;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*};*

*if (Iteration==IterationMax)*

*{*

*string[gl]=0;*

*gl = gl + 1;*

*string[gl]=0;*

*gl = gl + 1;*

*string[gl]=0;*

*gl = gl + 1;*

*}*

*else*

*{*

*string[gl]=255;*

*gl = gl + 1;*

*string[gl]=255;*

*gl = gl + 1;*

*string[gl]=255;*

*gl = gl + 1;*

*};*

*}*

*}*

*}*

**EK-3 ( SIMD Program )**

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <math.h>*

*#include <sys/types.h>*

*#include <time.h>*

*FILE \*\* fp;*

*int \* fpkonum;*

*void mandelbrot(int min\_y,int max\_y,int width,int height,FILE\* fp,int pid){*

*int iX,iY;*

*int min\_x = 0;*

*int max\_x = 1000;*

*const int iXmax = max\_x;*

*const int iYmax = max\_y;*

*double Cx,Cy;*

*const double CxMin=-2.5;*

*const double CxMax=1.5;*

*const double CyMin=-1.5;*

*const double CyMax=1.5;*

*double PixelWidth=(CxMax-CxMin)/width;*

*double PixelHeight=(CyMax-CyMin)/height;*

*static unsigned char piksel[3];*

*double Zx, Zy;*

*double Zx2, Zy2;*

*int Iteration;*

*const int IterationMax=255;*

*const double EscapeRadius=2;*

*double ER2=EscapeRadius\*EscapeRadius;*

*for(iY=min\_y;iY<height;iY++)*

*{*

*if(iY == max\_y) break;*

*for(iX=min\_x;iX<width;iX++)*

*{*

*Cy=CyMin + iY\*PixelHeight;*

*if (fabs(Cy)< PixelHeight/2) Cy=0.0;*

*Cx=CxMin + iX\*PixelWidth;*

*Zx=0.0;*

*Zy=0.0;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*for (Iteration=0;Iteration<IterationMax && ((Zx2+Zy2)<ER2);Iteration++)*

*{*

*Zy=2\*Zx\*Zy + Cy;*

*Zx=Zx2-Zy2 + Cx;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*}*

*if (Iteration==IterationMax)*

*{*

*piksel[0]=0;*

*piksel[1]=0;*

*piksel[2]=0;*

*}*

*else*

*{*

*piksel[0]=255;*

*piksel[1]=255;*

*piksel[2]=255;*

*}*

*fseek (fp , fpkonum[pid], SEEK\_SET );*

*fwrite(piksel,1,3,fp);*

*fpkonum[pid]+=3;*

*}*

*}*

*}*

*int main()*

*{*

*printf("SIMD Sistem\n");*

*int is;*

*printf("is sayisi girin: ");*

*scanf("%d",&is);*

*system("date +\"%T.%N\""); //Sistemin saati nanosaniye cinsinden yazılır.*

*int j;*

*FILE \*fp2[is]; //*

*fp = fp2; // fp2 global yapılıyor*

*int tryfpkonum[is];*

*fpkonum = tryfpkonum;*

*int gorev = (1000/is)\*1000;*

*fpkonum[0] = 24;*

*int i;*

*for(i = 1; i < is; i++)*

*{*

*fpkonum[i] = (gorev\*3) + fpkonum[i-1];*

*}*

*char \*filename="simd.ppm";*

*for(i = 0; i < is; i++)*

*{*

*fp[i]= fopen(filename,"wb");*

*}*

*fprintf(fp[0],"P6\n %s\n %d\n %d\n %d\n","# ",1000,1000,255);*

*int artis = 1000 / is;*

*int begin = 0;*

*int end = artis;*

*for(i = 0; i < is; i++)*

*{*

*if(fork()==0){*

*mandelbrot(begin, end,1000,1000,fp[i],i);*

*exit(1);*

*}*

*begin = end;*

*end = end + artis;*

*}*

*for(i = 0; i < is; i++) // Çocuk süreçlerin sonlanması bekleniyor.*

*wait(NULL);*

*system("date +\"%T.%N\"");*

*printf("Tamamlandı\n");*

*return 0;*

*}*

**EK-4 ( MIMD Program )**

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <math.h>*

*#include <sys/types.h>*

*#include <unistd.h>*

*#include <time.h>*

*int fpkonum = 24; // İlk 24 karakter resim bilgisi olduğundan*

*// pikseller burdan itibaren yazılmaya başlanacak*

*FILE \* fp;*

*char \*piksel;*

*int senkronpipe[2];*

*char senkbuffer[5];*

*int tasiyicipipe[2];*

*char \*buffer;*

*int cozunurluk;*

*void hesapla()*

*{*

*int iX,iY;*

*const int iXmax = cozunurluk;*

*const int iYmax = cozunurluk;*

*double Cx,Cy;*

*const double CxMin=-2.5;*

*const double CxMax=1.5;*

*const double CyMin=-2.0;*

*const double CyMax=2.0;*

*double PixelWidth=(CxMax-CxMin)/iXmax;*

*double PixelHeight=(CyMax-CyMin)/iYmax;*

*const int MaxColorComponentValue=255;*

*double Zx, Zy;*

*double Zx2, Zy2;*

*int Iteration;*

*const int IterationMax=200;*

*const double EscapeRadius=2;*

*double ER2=EscapeRadius\*EscapeRadius;*

*int syc = 0;*

*for(iY=0;iY<iYmax;iY++)*

*{*

*syc = 0;*

*Cy=CyMin + iY\*PixelHeight;*

*if (fabs((double)Cy)< PixelHeight/2) Cy=0.0;*

*for(iX=0;iX<iXmax;iX++)*

*{*

*Cx=CxMin + iX\*PixelWidth;*

*Zx=0.0;*

*Zy=0.0;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*for (Iteration=0;Iteration<IterationMax && ((Zx2+Zy2)<ER2);Iteration++)*

*{*

*Zy=2\*Zx\*Zy + Cy;*

*Zx=Zx2-Zy2 +Cx;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*};*

*if (Iteration==IterationMax)*

*{*

*piksel[syc++]=0;*

*piksel[syc++]=0;*

*piksel[syc++]=0;*

*}*

*else*

*{*

*piksel[syc++]=255;*

*piksel[syc++]=255;*

*piksel[syc++]=255;*

*};*

*}*

*write(senkronpipe[1], "tamam", 5);*

*write(tasiyicipipe[1], piksel, cozunurluk\*3);*

*}*

*exit(1);*

*}*

*void resimyazdir()*

*{*

*fp= fopen("mandelbrot\_mimd.ppm","wb");*

*fprintf(fp,"P6\n %s\n %d\n %d\n %d\n","# ",cozunurluk,cozunurluk,255);*

*int p = 0;*

*while(1)*

*{*

*read(senkronpipe[0], senkbuffer, 5);*

*// Sonuç hesaplanana kadar yazan süreç bekleyecek.*

*while(!(!strcmp(senkbuffer,"tamam"))) read(senkronpipe[0], senkbuffer, 5);*

*++p;*

*if(p == cozunurluk) break;*

*write(senkronpipe[1], "bekle", 5);*

*read(tasiyicipipe[0], buffer, cozunurluk\*3);*

*fseek (fp , fpkonum, SEEK\_SET );*

*fwrite(buffer,1,cozunurluk\*3,fp);*

*fpkonum+=cozunurluk\*3;*

*}*

*close(senkronpipe[0]);*

*close(senkronpipe[1]);*

*close(tasiyicipipe[0]);*

*close(tasiyicipipe[1]);*

*}*

*int main()*

*{*

*printf("MIMD Sistem\n");*

*printf("Çözünürlük? (AxA): ");*

*scanf("%d",&cozunurluk);*

*char piksel2[cozunurluk\*3]; // her satırın sonucu bu buffer da saklanacak.*

*piksel = piksel2;*

*char trybuffer[cozunurluk\*3];*

*buffer = trybuffer; //Buffer, çocuk süreçte görsün diye global yapılıyor*

*system("date +\"%T.%N\""); //Sistem saatini nanosaniye olarak yazıyor.*

*pipe(senkronpipe);*

*pipe(tasiyicipipe);*

*write(senkronpipe[1], "bekle", 5);*

*if(fork() == 0) hesapla(); else resimyazdir();*

*fclose(fp);*

*system("date +\"%T.%N\"");*

*return 0;*

*}*

**EK-5 (Hibrit Program)**

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <math.h>*

*#include <sys/types.h>*

*#include <unistd.h>*

*#include <time.h>*

*char piksel[3000];*

*void hesapla(int min,int max,int\* \_ortak,int\* \_senk)*

*{*

*int iX,iY;*

*const int iXmax = 1000;*

*const int iYmax = 1000;*

*double Cx,Cy;*

*const double CxMin=-2.5;*

*const double CxMax=1.5;*

*const double CyMin=-2.0;*

*const double CyMax=2.0;*

*double PixelWidth=(CxMax-CxMin)/iXmax;*

*double PixelHeight=(CyMax-CyMin)/iYmax;*

*const int MaxColorComponentValue=255;*

*double Zx, Zy;*

*double Zx2, Zy2;*

*int Iteration;*

*const int IterationMax=200;*

*const double EscapeRadius=2;*

*double ER2=EscapeRadius\*EscapeRadius;*

*int syc;*

*syc = 0;*

*for(iY=min;iY<iYmax;iY++)*

*{*

*if(iY == max) break;*

*syc = 0;*

*Cy=CyMin + iY\*PixelHeight;*

*if (fabs((double)Cy)< PixelHeight/2) Cy=0.0;*

*for(iX=0;iX<iXmax;iX++)*

*{*

*Cx=CxMin + iX\*PixelWidth;*

*Zx=0.0;*

*Zy=0.0;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*for (Iteration=0;Iteration<IterationMax && ((Zx2+Zy2)<ER2);Iteration++)*

*{*

*Zy=2\*Zx\*Zy + Cy;*

*Zx=Zx2-Zy2 +Cx;*

*Zx2=Zx\*Zx;*

*Zy2=Zy\*Zy;*

*};*

*if (Iteration==IterationMax)*

*{*

*piksel[syc++]=0;*

*piksel[syc++]=0;*

*piksel[syc++]=0;*

*}*

*else*

*{*

*piksel[syc++]=255;*

*piksel[syc++]=255;*

*piksel[syc++]=255;*

*};*

*}*

*write(\_senk[1], "bitti", 5);*

*write(\_ortak[1], piksel, 3000);*

*}*

*exit(1);*

*}*

*void yazdir(FILE\* fp,int adim,int\* \_ortak,char\* \_obuf,int\* \_senk,char\* \_senkbuf, int fps, int\* tamam)*

*{*

*int p = 0;*

*int u = 1;*

*while(u)*

*{*

*read(\_senk[0], \_senkbuf, 5);*

*while(!strcmp(\_senkbuf,"bekle")) read(\_senk[0], \_senkbuf, 5);*

*++p;*

*if(p == adim+1) break;*

*write(\_senk[1], "bekle", 5);*

*read(\_ortak[0], \_obuf, 3000);*

*fseek (fp , fps, SEEK\_SET );*

*fwrite(\_obuf,1,3000,fp);*

*fps+=3000;*

*}*

*write(tamam[1], "bitti", 5);*

*}*

*int main()*

*{*

*printf("Hem SIMD hem MIMD Program\nHesaplama farklı çocuk süreçlere ve dosyaya yazma da farklı çocuk süreçlere dağıtılacaktır.\n");*

*int is;*

*printf("hesaplama ve haberleşme kac alt process bolunsun?(x:int): ");*

*scanf("%d",&is);*

*system("date +\"%T.%N\"");*

*int tamam[is][2];*

*char buf[is][30];*

*int fps = 24;*

*int j=0;*

*FILE \*f[is];*

*int \_ortak[is][2];*

*int \_senk[is][2];*

*char \_senkbuf[is][5];*

*int pay = (1000/is)\*3;*

*char \_buf[is][3000];*

*char \*filename="simdvemimd.ppm";*

*for(j = 0; j < is; j++)*

*{*

*f[j] = fopen(filename,"wb");*

*pipe(tamam[j]);*

*pipe(\_ortak[j]);*

*pipe(\_senk[j]);*

*write(\_senk[j][1], "bitti", 5);*

*write(tamam[j][1], "bekle", 5);*

*}*

*char \*comment="# ";*

*fprintf(f[0],"P6\n %s\n %d\n %d\n %d\n",comment,1000,1000,255);*

*int min = 0;*

*int adim = 1000/is;*

*int max;*

*for(j=0;j<is;j++)*

*{*

*max = min + adim;*

*if(fork() == 0) { hesapla(min, max, \_ortak[j], \_senk[j]); exit(1); }*

*if(fork() == 0) { yazdir(f[j],adim, \_ortak[j], \_buf[j], \_senk[j], \_senkbuf[j], fps, tamam[j]); exit(1); }*

*fps = fps + 3000\*(1000/is);*

*min = min + adim;*

*}*

*for(j = 0; j < is; j++)*

*{*

*read(tamam[j][0], buf[j], 5);*

*while(!strcmp(buf[j],"bekle")) read(tamam[j][0], buf[j], 5);*

*}*

*system("date +\"%T.%N\"");*

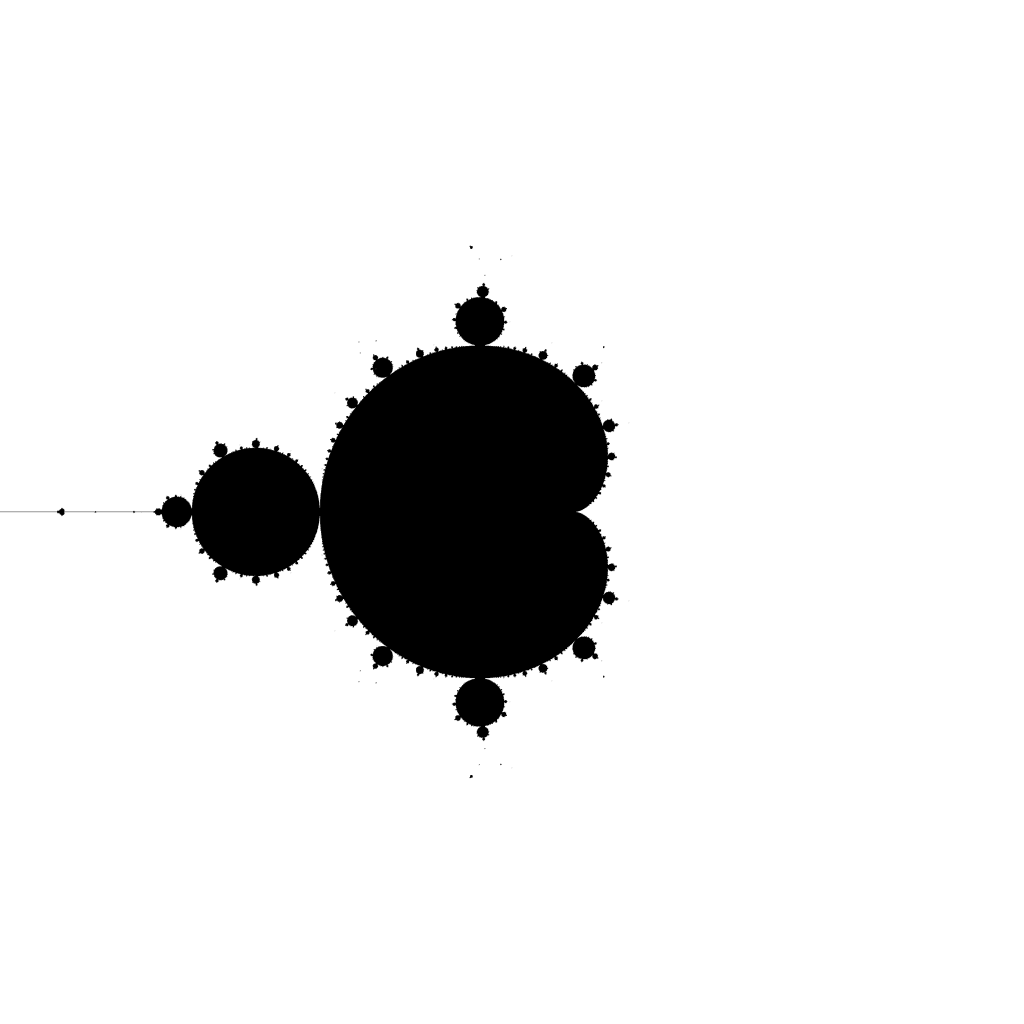
*printf("Tamamlandi\n");*

*getchar();*

*return 0;*

*}*

**EK-6 ( MandelBrot Görüntüsü )**

****