KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROJENİN KONUSU

MANDELBROT SETİ PARALELLEŞTİRMESİNİN PERFORMANS ARTIŞI VE BOYUT SEÇİMİ ANALİZİ

Numara:	**Ad-Soyad:**
394811	Ufuk BULUT
394797	Osman Can AKSOY
394753	Hüdahan ALTUN
407632	MEHMET ALİ AKAN
394775	Mustafa Görgün ÖZDEMİR

Ders Sorumlusu: ***Prof. Dr. Cemal KÖSE***

TRABZON _ 2024

İÇİNDEKİLER

1. ÖZET

Bu projede, Mandelbrot Seti'nin paralelleştirilmesi ve performans analizi üzerine odaklanıyoruz. İki farklı yöntem kullanarak, openMP kütüphanesi ve grafik işlemci birimi kullanarak Mandelbrot Seti'nin hesaplanmasını paralelleştireceğiz. Ardından, her iki yöntemin sağladığı performans artışını ve hızlanmayı gösterecek şekilde uygun beş farklı boyut belirleyip, bu boyutlarda zaman, hızlanma (speed-up) ve verimlilik (efficiecy) eğrilerini çıkaracağız. Paralelleştirme ile sağlanan veya sağlanamayan performans artışının nedenlerini, her iki yöntem için ayrı ayrı değerlendirerek raporumuzda sunacağız. Bu analizler, paralelleştirme tekniklerinin Mandelbrot Seti hesaplamasındaki etkinliğini ve farklı boyutlardaki performansını açıklayacaktır.

2. GİRİŞ

Bu projemizde, Mandelbrot Seti'nin paralelleştirilmesi ve performans analizi üzerine yoğunlaşırken, geliştirme sürecinde Qt framework'ünden faydalandık. Qt, birden çok platformu destekleyen ve verimli bir grafiksel kullanıcı arayüzü geliştirme araç takımıdır. Farklı dillerle olan bağlantıları sayesinde esneklik sunan bu araç takımı, projenin geliştirilmesini kolaylaştırdı. Ayrıca, yazılımın farklı bileşenlerinde CUDA ve SFML gibi teknolojileri kullandık.

SFML (Simple and Fast Multimedia Library), bilgisayar bileşenleri için basit arayüz sağlayan platform bağımsız bir yazılım geliştirme kütüphanesidir. Bu kütüphane, grafik, ses, ağ ve giriş işlevselliği gibi çeşitli bileşenler için basit ve etkili bir arayüz sunar. CUDA ise, GPU için NVIDIA tarafından sunulan bir teknolojidir ve paralel hesaplama işlemlerini hızlandırmak için kullanılır. Bu projede, SFML ve CUDA gibi teknolojileri kullanarak grafiksel arayüz geliştirmeyi ve paralel hesaplama işlemlerini optimize etmeyi hedefledik.

3. YAPILACAK ÇALIŞMA PLANI

- Projenin kapsamı ve hedefleri belirlendi, paralelleştirme süreci için bir çalışma planı oluşturuldu.
- Mandelbrot Seti'nin paralelleştirilmesi için kullanıcı dostu bir arayüz geliştirilmesi planlandı. Bu arayüz, Qt framework'ü kullanılarak tasarlanacak.
- İlk aşamada, Python kütüphaneleriyle basit bir Mandelbrot Seti hesaplama işlemi gerçekleştirilecek ve sonuçlar grafik arayüzde gösterilecek.
- Paralelleştirme işlemi için multi-threading özelliği uygun kütüphaneler kullanılarak projeye entegre edilecek ve birden fazla işlemcinin koşması sağlanacak. Bu sayede hızlanma ve verimlilik artışı elde edilecek.

4. İÇERİK

Mandelbrot Kümesi;

Mandelbrot kümesi, karmaşık düzlemdeki karmaşık sayılarla tanımlanan ve büyütüldüğünde sonsuz derecede karmaşık detaylar içeren iki boyutlu bir kümedir. Bu kümenin estetik çekiciliği ve fraktal yapısı popülerlik kazanmıştır. Mandelbrot kümesi, ilk olarak Robert W. Brooks ve Peter Matelski tarafından 1978'de Kleinian grupları üzerine yapılan bir çalışmanın parçası olarak tanımlanmış ve çizilmiştir. Daha sonra, Benoit Mandelbrot 1980 yılında IBM'in Thomas J. Watson Araştırma Merkezi'nde çalışırken kümenin yüksek kaliteli görselleştirmelerini elde etmiştir.

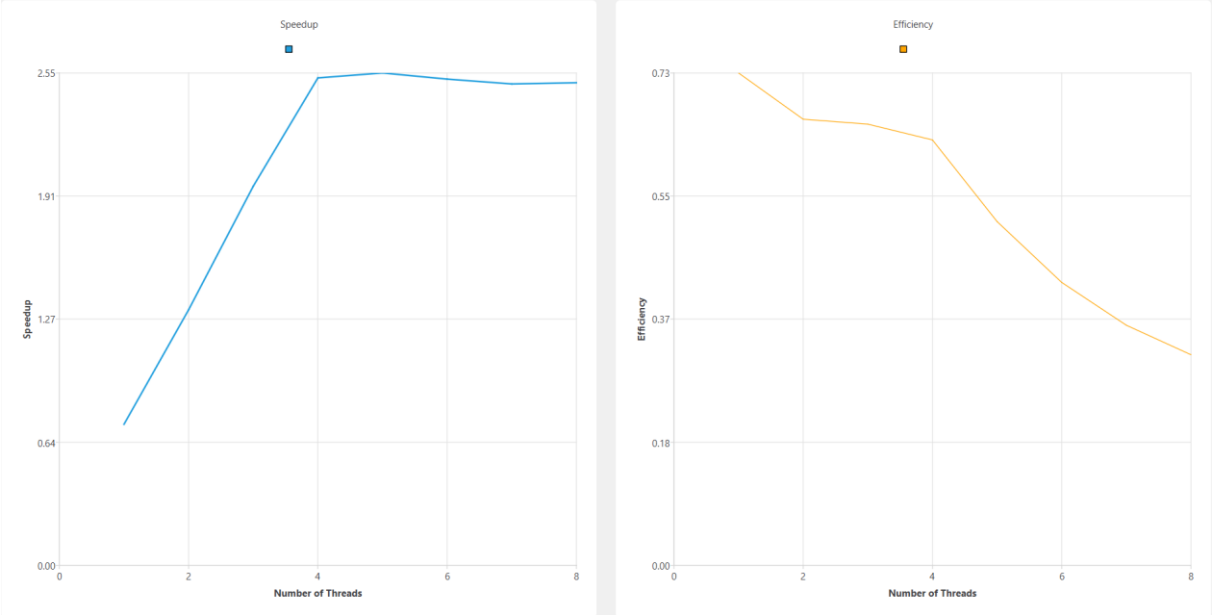
Mandelbrot kümesinin görüntüleri, artan büyütmelerde giderek daha ince özyinelemeli ayrıntıları ortaya çıkaran sonsuz derecede karmaşık bir sınır sergiler. Bu özyinelemeli ayrıntının tarzı, incelenen bölgeye bağlıdır. Mandelbrot kümesi görüntüleri, karmaşık sayılar örneklenerek ve her örnek nokta için test edilerek oluşturulabilir. Bu küme, matematiksel güzellik ve görselleştirme açısından önemli bir örnektir.

Paralel hesaplama, birçok hesaplama veya işlemin aynı anda gerçekleştirildiği bir hesaplama türüdür. Genellikle büyük problemler daha küçük problemlere bölünerek aynı anda çözülür. Paralel hesaplamanın farklı biçimleri vardır ve yüksek performanslı bilgi işlemde uzun zamandır kullanılmaktadır. Son yıllarda, bilgisayarların güç tüketimi ve ısı üretimi gibi fiziksel kısıtlamalar nedeniyle, paralel hesaplama özellikle çok çekirdekli işlemciler biçiminde baskın bir paradigma haline gelmiştir. Bu proje kapsamında, Mandelbrot kümesinin hesaplanması paralelleştirilecek ve farklı hesaplama kaynaklarının aynı anda kullanılmasıyla performans artışı sağlanacaktır.

5. SONUÇLAR

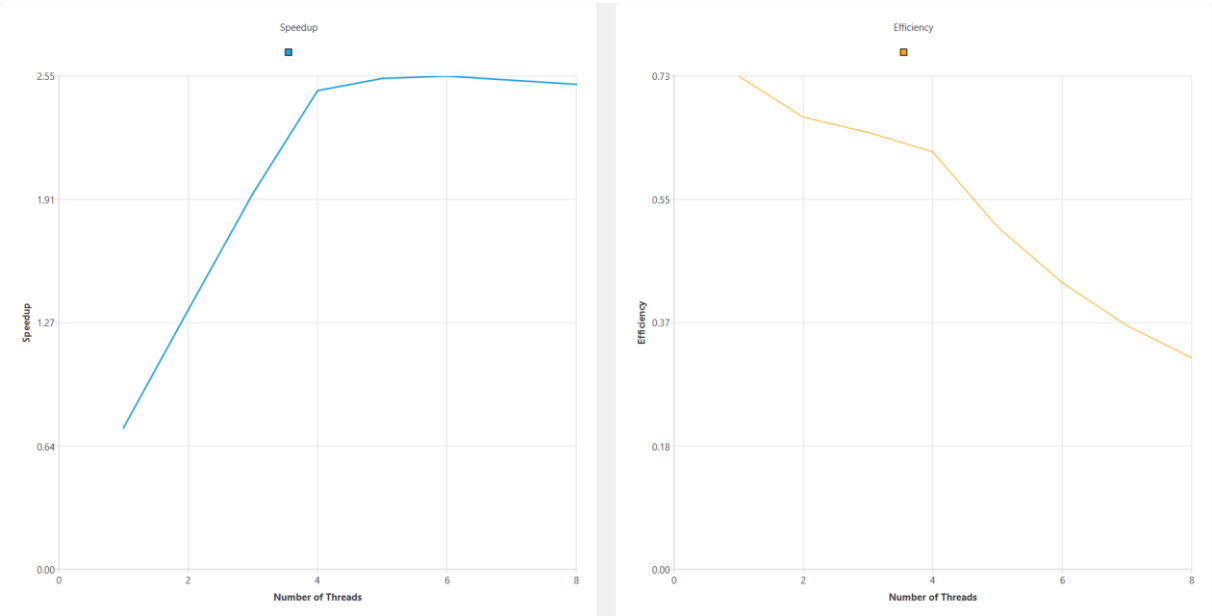
5.1 İşlemci üzerinde paralelleştirme

800x600



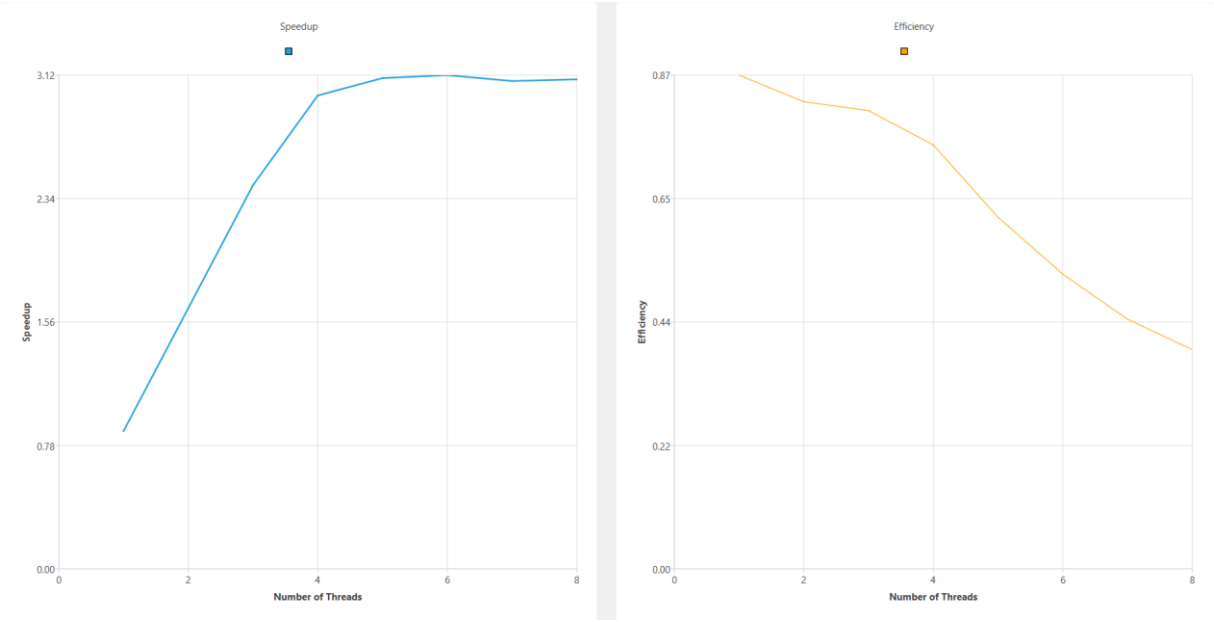
Sequential Time = 1.396sn **Parallel Time** = 0.402sn

1000x1000



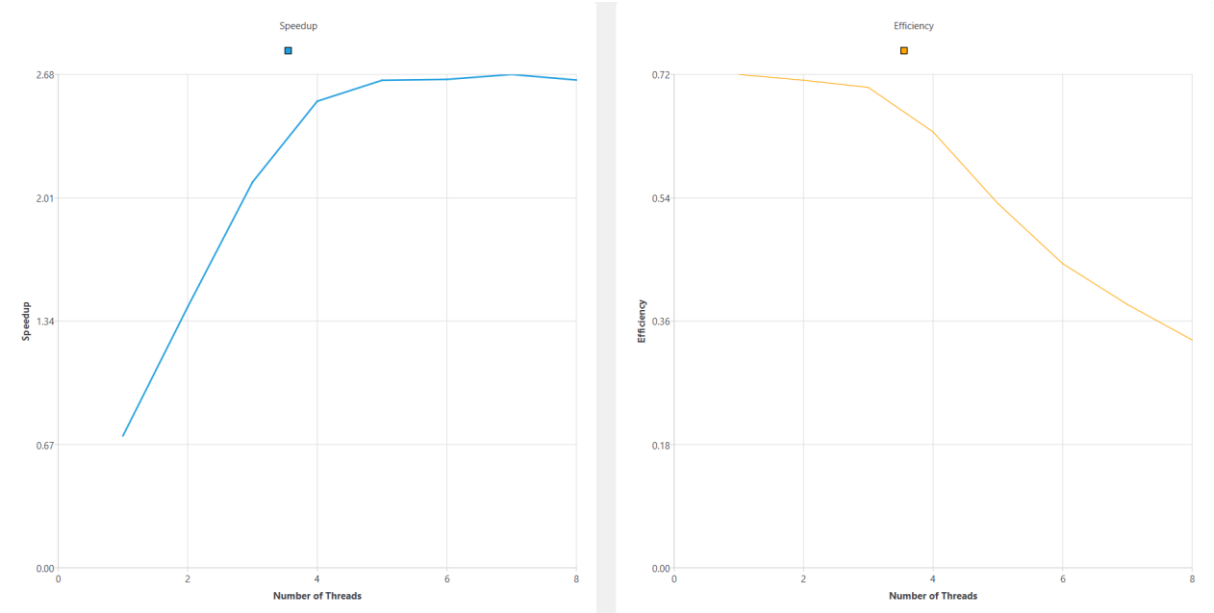
Sequential Time = 2.121sn **Parallel Time** = 0.841sn

1200x1200



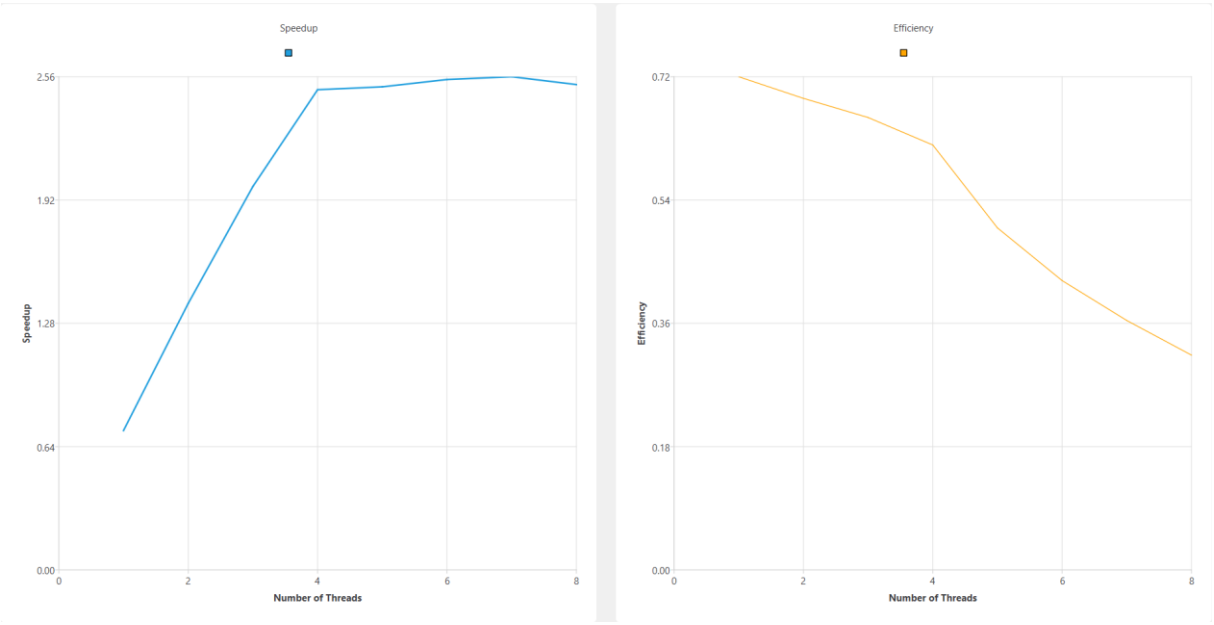
Sequential Time = 3.046sn **Parallel Time** = 1.176sn

1400x1400



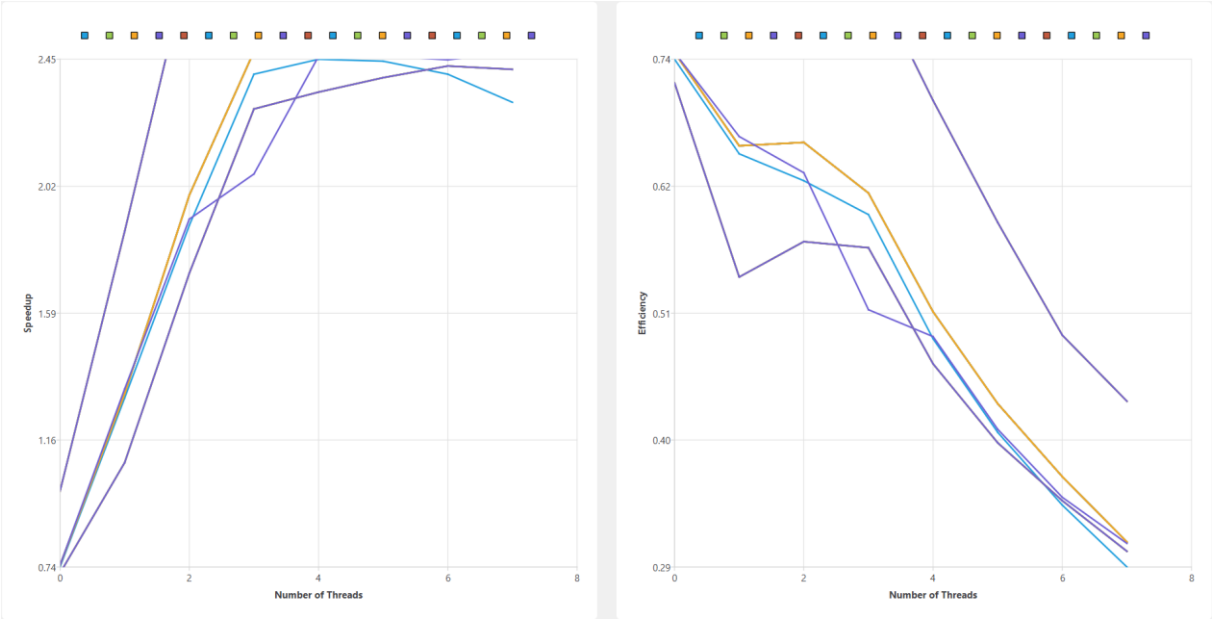
Sequential Time = 4.121sn **Parallel Time** = 1.601sn

1600x1600



Sequential Time = 5.406sn **Parallel Time** = 2.17sn

Aşağıda 5 farklı boyut bir arada gösterilmektedir.

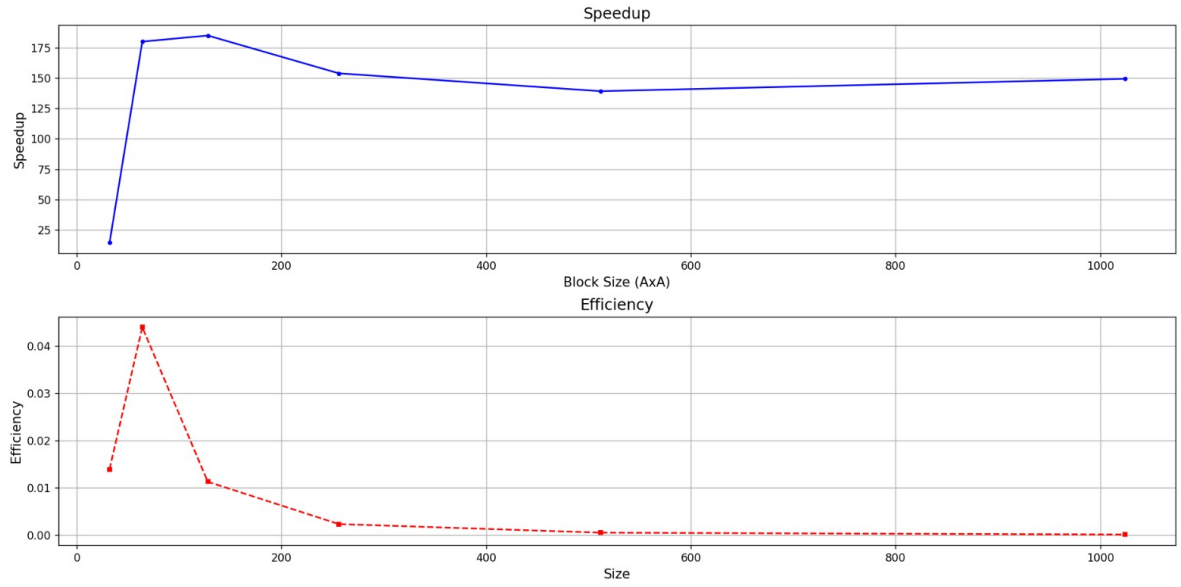


5.1.1 Programın toplama ile ilgili kod kısmı :

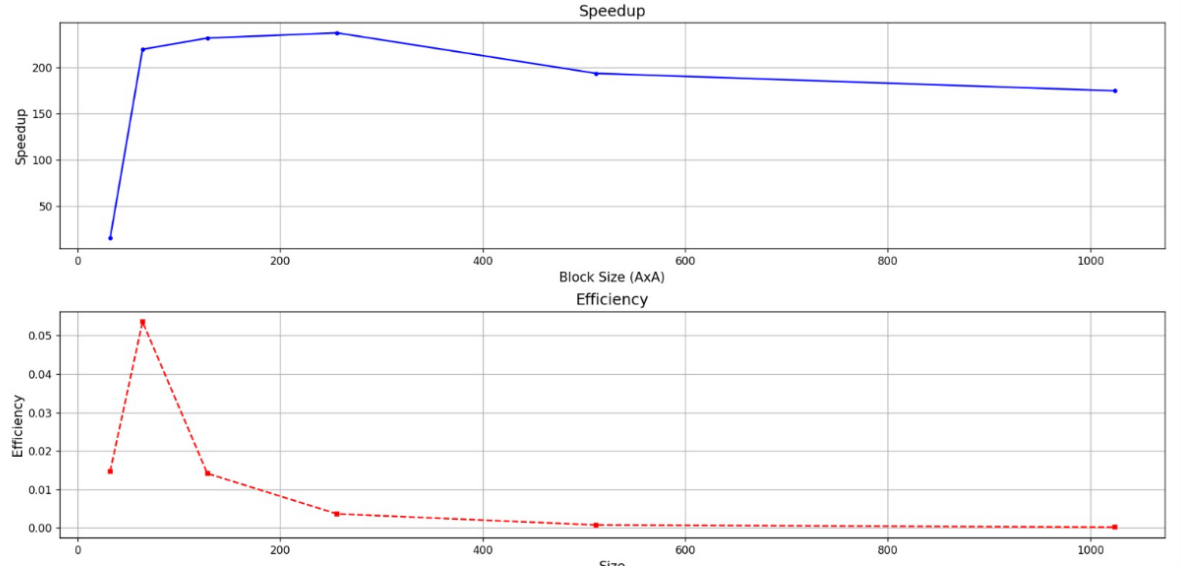
```
void MainWindow::computeMandelBrotParallel()
{
    QImage image(IMAGE_WIDTH, IMAGE_HEIGHT, QImage::Format_RGB32);

    int maxThreads = omp_get_max_threads();

    for(int numThreads = 1; numThreads <= maxThreads; numThreads++) {
        omp_set_num_threads(numThreads);

        double startTime = omp_get_wtime();

#pragma omp parallel for schedule(dynamic)
        for (int y = 0; y < IMAGE_HEIGHT; y++)
        {
            for (int x = 0; x < IMAGE_WIDTH; x++)
            {
                long double real = MIN_REAL + (MAX_REAL - MIN_REAL) * x / IMAGE_WIDTH;
                long double imaginary = MIN_IMAGINARY + (MAX_IMAGINARY - MIN_IMAGINARY) * y / IMAGE_HEIGHT;

                std::complex<double> z(0, 0);
                std::complex<double> c(real, imaginary);
                int iterations = 0;
                while (std::abs(z) < 2 && iterations < MAX_ITERATIONS)
                {
                    z = pow(z, POWER_Z) + c;
                    iterations++;
                }

                if (iterations < MAX_ITERATIONS)
                {
                    long double hue = 360.0 * iterations / MAX_ITERATIONS;
                    QColor color = QColor::fromHsvF(hue / 360.0, 1.0, 1.0);

#pragma omp critical
                    {
                        image.setPixelColor(x, y, color);
                    }
                }
                else
                {
#pragma omp critical
                    {
                        image.setPixelColor(x, y, Qt::black);
                    }
                }
            }
        }

        double endTime = omp_get_wtime();
        parallelTime = endTime - startTime;

        double speedup = sequentialTime / parallelTime;
        speedups.push_back(speedup);

        double efficiency = speedup / numThreads;
        efficiencies.push_back(efficiency);
    }

    scene->clear();
    scene->addPixmap(QPixmap::fromImage(image));
}
```

5.2 Gpu üzerinde paralleştirme:

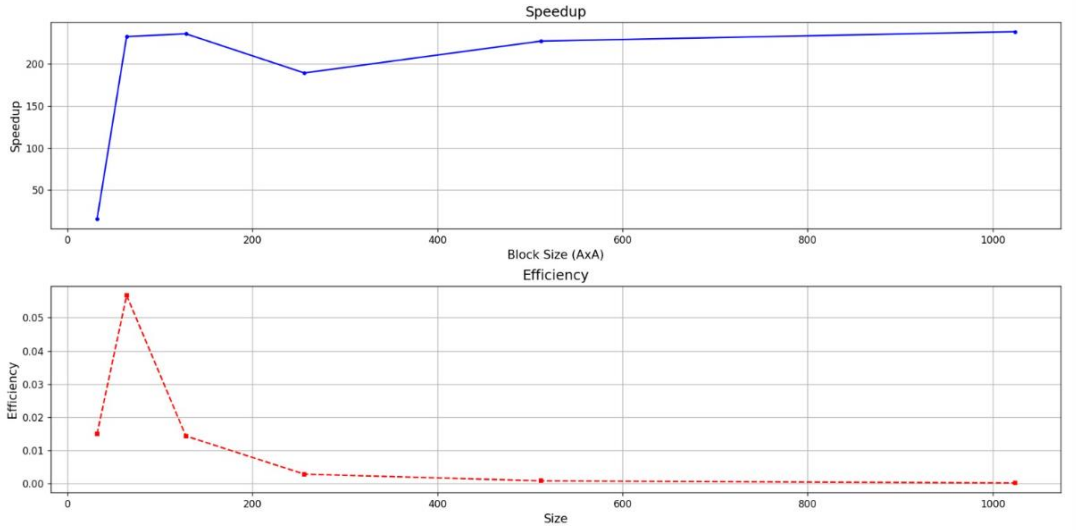
800x600



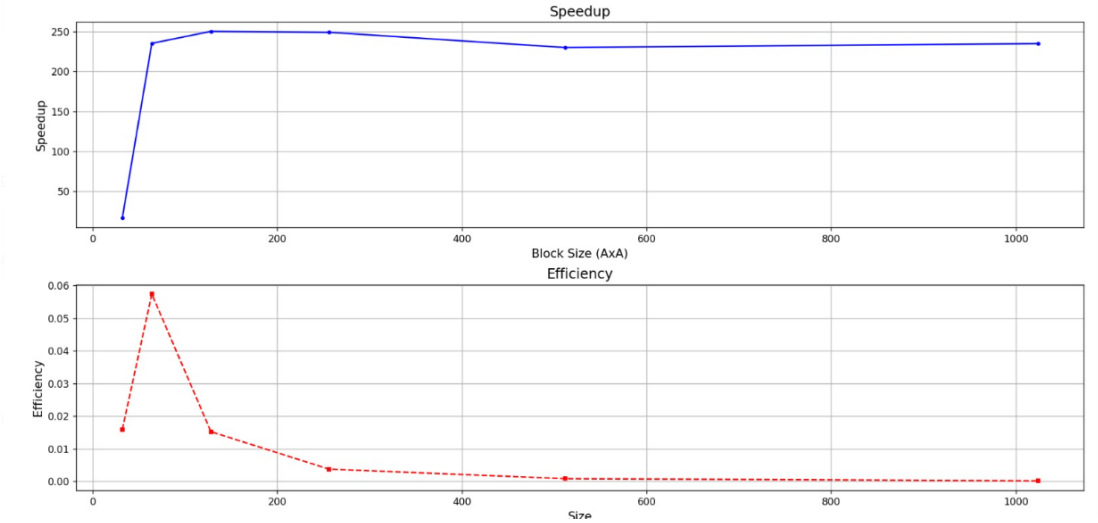
1000x1000



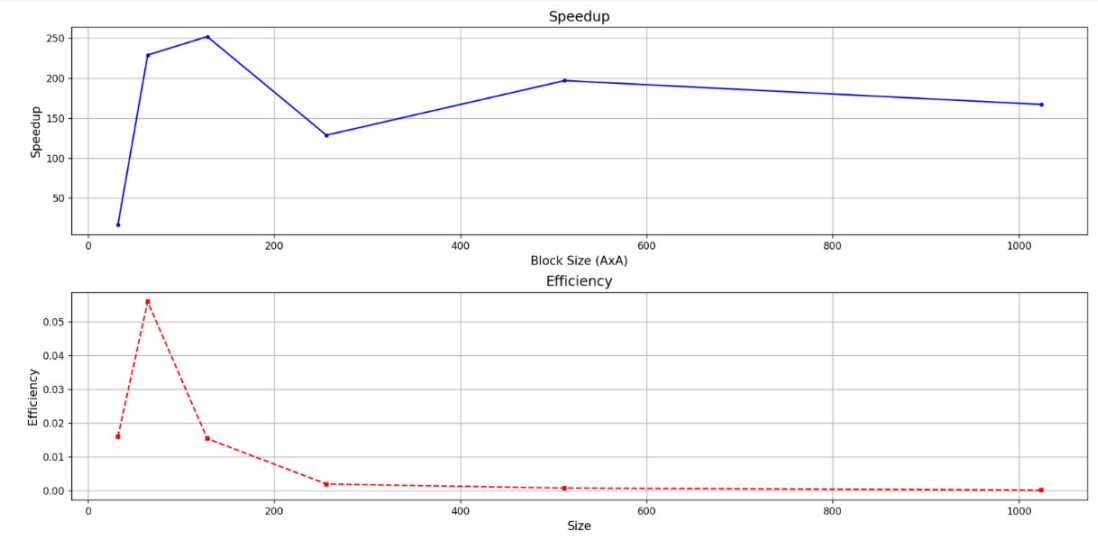
1200x1200



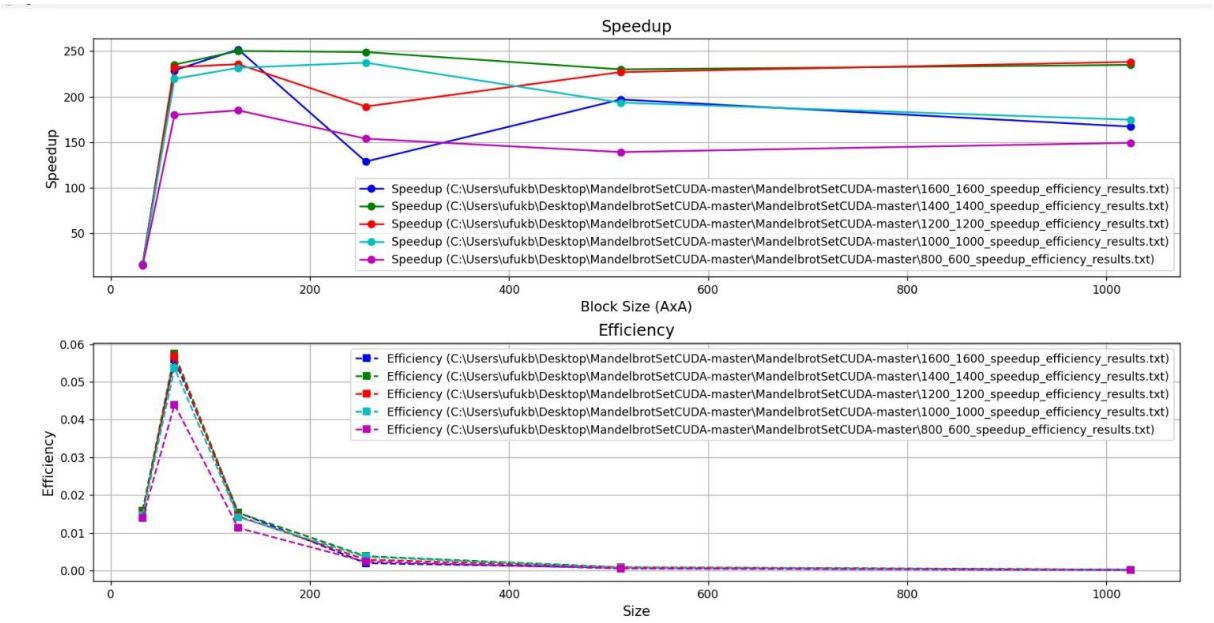
1400x1400



1600x1600



Aşağıda 5 farklı boyut bir arada gösterilmektedir.

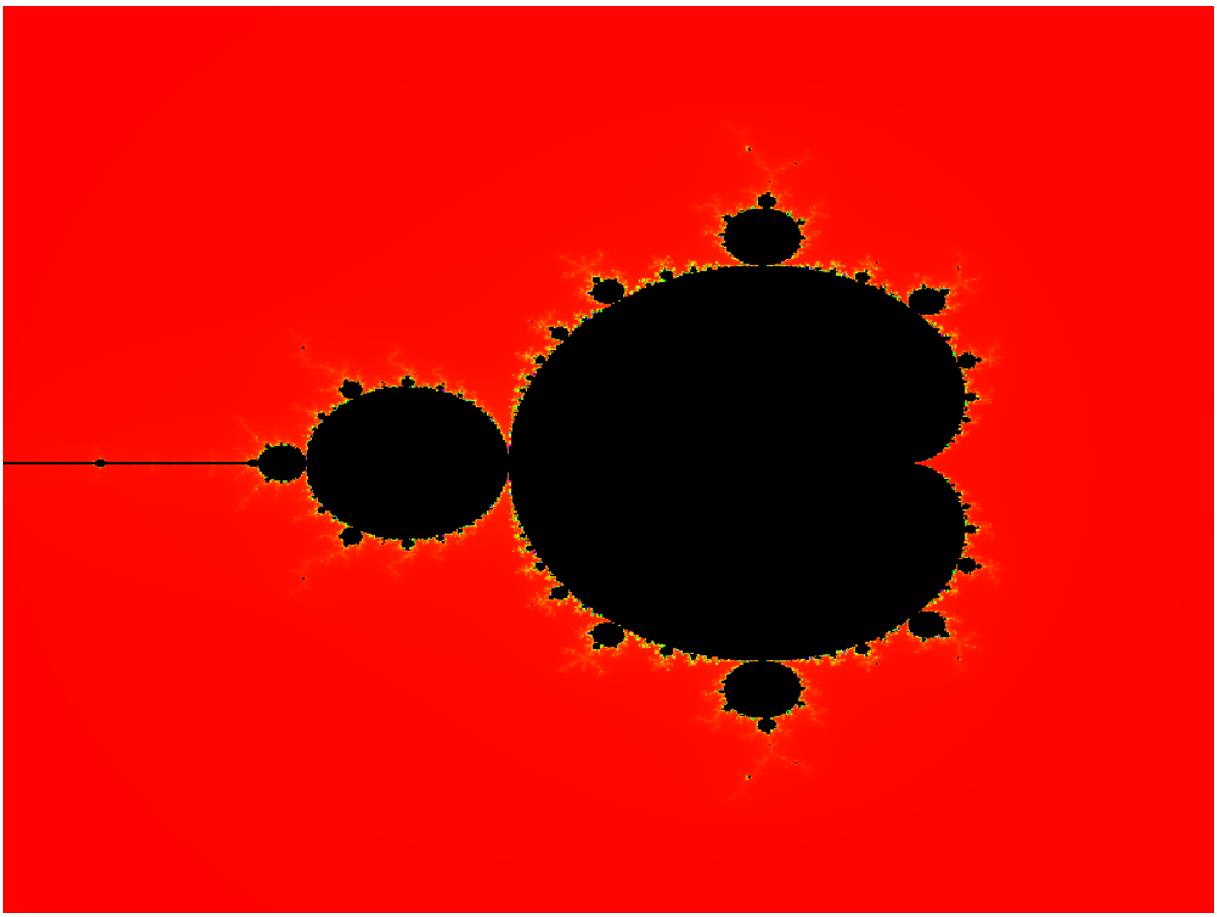


GPU üzerinde yapılan paralelleştirme çalışmalarında farklı blok boyutlarıyla performans analizi gerçekleştirilmiştir.
Blok boyutları, 16x16'dan 32x32'ye kadar artırılarak, Nvidia GTX 1650 grafik kartı kullanılarak testler yapılmıştır. Özellikle 32x32 blok boyutunda performansın en yüksek verimlilik seviyesine ulaştığı gözlemlenmiştir.
Grafiklerde görülen dalgalanmalar, farklı blok boyutlarının performans üzerindeki etkilerini açıkça göstermektedir. Bu sonuçlar, GPU performansını optimize etme sürecinde blok boyutunun önemli bir faktör olduğunu ve belirli iş yükleri için en uygun blok boyutlarının seçilmesinin performans üzerinde belirgin bir etkisi olduğunu ortaya koymaktadır.

5.2.1 Programın toplama ile ilgili kod kısmı :

```
int ms::Mandelbrot_set::calculate_paralell_CUDA()
{
    std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

    int* result = mendelbrot_kernel(height, width, maxIter, startX, startY, zoom,32,32);
    for (int x = 0; x < (*window).getSize().x; ++x)
    {
        for (int y = 0; y < (*window).getSize().y; ++y)
        {
            int i = x + y * width;

#ifdef __STATIC_MAX_ITER
            vertices[i].color.r = palette[result[i]].r;
            vertices[i].color.g = palette[result[i]].g;
            vertices[i].color.b = palette[result[i]].b;
#else
            double iterationNorm = 360.0 * result[i] / maxIter;
            if (result[i])
                vertices[i].color = hsv2rgb_fast(iterationNorm);
            else
                vertices[i].color = sf::Color::Black;
#endif
        }
    }
    cudaFreeHost(result);

    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
    int paralell = std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count();
    return paralell;
}
__device__ unsigned long long totThr = 0;
__global__ void fractal(int nrows, int ncols, int max_iter, double startX, double startY, double zoom, int* result) //373 - 326
{
    atomicAdd(&totThr, 1);
    int x_idx = threadIdx.x + blockIdx.x * blockDim.x;
    int y_idx = threadIdx.y + blockIdx.y * blockDim.y;
    if (x_idx >= ncols || y_idx >= nrows)
        return;

    int l = x_idx + y_idx * ncols;
    // Pikselin koordinatlarını belirle
    double x = ((x_idx - ncols / 2) * zoom + startX) / ncols * 3.5 - 0.75;
    double y = ((y_idx - nrows / 2) * zoom + startY) / nrows * 2.0;
    double re = x, im = y;

    //MANDELBROT SETİ HESAPLAMASI
    for (int i = 1; i < max_iter; ++i)
    {
        if (re * re + im * im >= 4)
        {
            result[l] = i;
            return;
        }
        double reTemp = re * re - im * im + x;
        im = 2 * re * im + y;
        re = reTemp;
    }
    result[l] = 0;
}
```

6. MANDELBROT KÜMESİNİN ÇİZİLMİŞ HALİ



7. DEĞERLENDİRME

Projede Mandelbrot kümesinin paralelleştirilmesi için farklı yöntemler kullanılmıştır. Öncelikle openMP kütüphanesi kullanılarak işlemci/çekirdek sayısına göre paralelleştirme yapılmıştır. Bu yöntemde, 1, 2, 3 ve 4 işlemci/çekirdek üzerinde Mandelbrot setinin hesaplanması sağlanmıştır. Ayrıca, aynı problemi grafik işlemci (GPU) kullanarak da paralelleştirilmiştir.

İlk olarak, zaman, hızlanma (speed-up) ve verimlilik (efficiency) eğrileri her iki yöntem için çizilmiştir. Bu eğrilerin incelenmesi, paralelleştirme ile sağlanan performans artışını göstermektedir. Paralelleştirme sayesinde hesaplama süresinde belirgin bir düşüş görülmüştür, bu da işlemlerin aynı anda gerçekleştirilmesiyle elde edilen performans artışını göstermektedir.

Ancak, grafik işlemci kullanımında daha belirgin bir hızlanma (speed-up) elde edilmiştir. Bunun nedeni, grafik işlemcilerin paralel hesaplama için optimize edilmiş olması ve büyük veri setlerini daha hızlı işleyebilme yeteneğidir. Bu durum, paralelleştirme ile sağlanan performans artışının, grafik işlemcisi kullanılarak daha etkin bir şekilde gerçekleştirilebileceğini göstermektedir.

Özetle, her iki yöntem de paralelleştirme ile belirgin bir performans artışı sağlamıştır. Ancak, grafik işlemci kullanımı daha yüksek bir hızlanma ve verimlilik sağlamıştır, çünkü grafik işlemciler paralel hesaplama için daha uygun bir mimariye sahiptir.

8. KAYNAKÇA

1-https://tr.wikipedia.org/wiki/Mandelbrot_k%C3%BCmesi

2-https://www.openmp.org/

3-https://en.wikipedia.org/wiki/Simple_and_Fast_Multimedia_Library

4-https://doc.qt.io/qt-6/examples-threadandconcurrent.html

5-https://tr.wikipedia.org/wiki/OpenMP

6-https://www.geeksforgeeks.org/introduction-to-cuda-programming/

7-https://paulbourke.net/fractals/mandelbrot/Ruben_van_Nieuwpoort.html