**T.C.**

**KOCAELİ ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

**BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ**

****

**Paralel ve Dağıtık Sistem Programlama**

**İşlem Yoğunluğuna Sahip Matrislerin İşlem Sürelerinin Hesaplanması**

**Oğulcan Uçarsu**

**185172014**

**ÖĞRETİM GÖREVLİSİ**

**DOÇ. DR. Hikmet Hakan GÜREL**

**KASIM 2019**

**1.Giriş**

Matrisler, bilim ve alt dallarında kullanılan matematiksel hesaplama araçlarından bir tanesidir. Matris çarpımı, boyutları birbiri ile uyumlu 2 matrisin çarpılarak yeni bir matris oluşturulduğu matematiksel işlemdir. Matrislerin satır ve sütunlarının sayısı matrisin boyutu olarak bilinmektedir. Yüksek işlem sayısı ve performans içeren algoritmaların optimizasyonu yapılırken matris çarpımları kullanılmaktadır. 2 tane matrisi çarpabilmek için 1. matrisin sütün sayısı ile 2. matrisin satır sayısı aynı olmak zorundadır[1,2,3].

**2.Yöntem**

2 Kare matrisin çarpımı C programlama dili yazılacak kod parçacığı ile hesaplanacaktır. Matrisler çarpılırken kullanılan Bilgisayar’ın Bellek ve İşlemcisi üzerinde ki işlem seviyeleri ve yoğunlukları izlenecektir. Matris çarpımı yapacak olan Bilgisayar’ın özellikleri şu şekildedir.

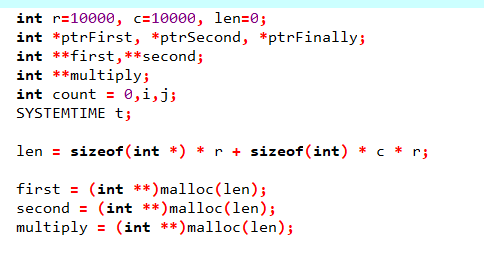
* Marka: Asus
* İşlemci: Intel Core i7-4700HQ CPU 2.40 Ghz
* Bellek: 32 GB
* Sistem Türü: 64 bit, İşletim Sistemi Windows 10

Çarpılacak olan kare matrislerin, çarpım sürelerini en iyi şekilde görebilmek ve optimizasyonu sağlamak için aşağıda ki matris boyutlarına göre çarpımlar yapılacaktır. Çarpılacak 2 matrisinde boyutları aynı olacak şekilde belirlenmiştir.

* 500\*500
* 1000\*1000
* 1500\*1500
* 2000\*2000
* 2500\*2500
* 3000\*3000
* 3500\*3500
* 4000\*4000
* 4500\*4500
* 5000\*5000
* 6000\*6000
* 7000\*7000
* 8000\*8000
* 9000\*9000
* 10000\*10000

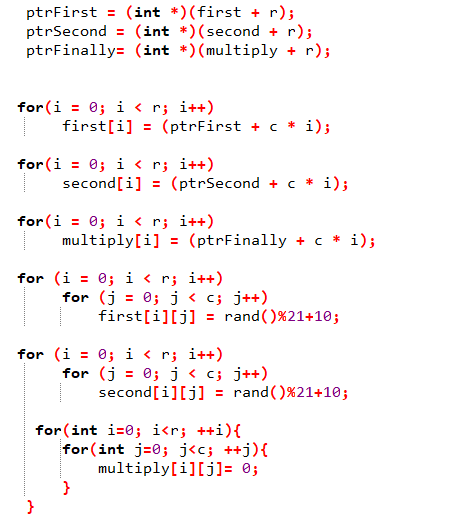
**2.1 Matris çarpımı yapacak olan kodun oluşturulması**

Matris çarpımını yapacak olan kod yazılırken, C programlama dili kullanılmıştır. Sebebi ise C, C++, Fortran gibi programlama dilleri, paralel ve seri programlama yaparken de kullanılabildiği için ve MPI vb. paralel programlama kütüphaneleri ile uyumlu çalıştığı için tercih edilmiştir. C kodu yazılırken, ‘Pointer’ ve ‘Malloc’ mantıksal yapıları kullanılmıştır. Bu mantıksal yapıların kullanılma sebebi ise büyük matrislerin çarpımları yapıldığı için, matris elemanlarının ‘stack’ yerine ‘heap’ üzerinde tutularak bellek problemi yaşamamasının önüne geçilmiştir. ‘Pointer’ ve ‘Malloc’ mantıksal yapıları kullanılmadığında, matris elemanları ‘stack’ üzerinde tutulacaktır. ‘stack’ üzerinde ki boyut sınırlı olduğu için program çalıştırıldığında hata alınabilir. Aşağıda ki şekillerde ilgili kod parçacıkları anlatılacaktır.



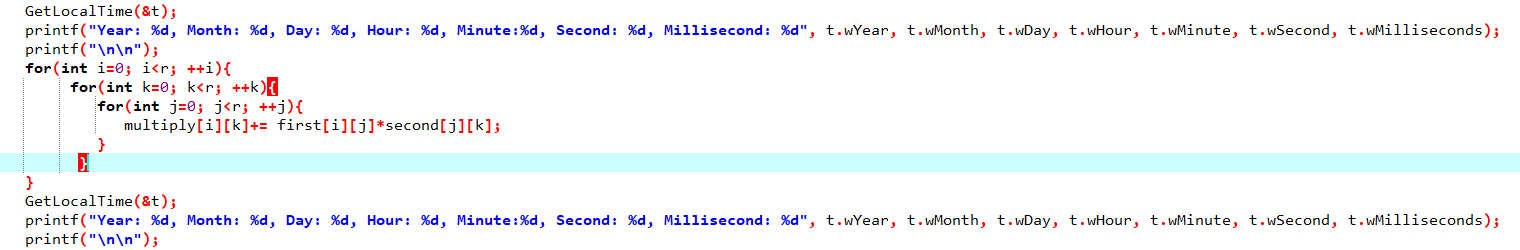
Şekil – 1 Matris çarpımında kullanacağımız elemanların tanımlaması

İlk önce kullanılacak değişkenler tanımlanır. ‘r’ ve ‘c’ değişkenleri matrisin satır ve sütün sayısını ifade eder, ‘len’ olarak tanımlanan değişken ise matris çarpımında satır ve sütün elemanları sırasıyla çarpıldıktan sonra toplanarak sonuç matrisinde ki ilgili satır ve sütün elemanını bulmamıza yardımcı olacağı için bellekte o kadarlık yer ayrılır.



Şekil – 2 Matrislere bellekte oluşturulması ve değerlerin atanması

Çarpım yapacak olan matrislerin ve çarpım sonucunu gösterecek olan matrisler içinde ilk önce bellek üzerinde değer ataması yapılır, daha sonra çarpım yapacak olan matrislere 21 ile 30 arasında ki sayılar ‘rand’ fonksiyonu kullanılarak random şekilde değerler atanır. Ayrıca sonucu gösterecek olduğumuz matris içinde bütün değerler ilk önce sıfır atanır.



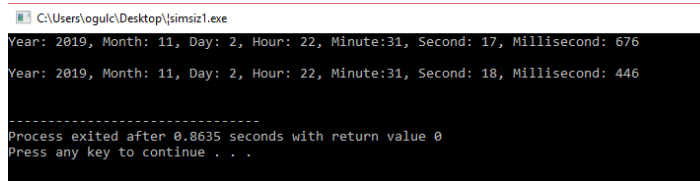
Şekil – 3 Matris çarpımı yapan kod parçacığı.

500\*500’den 10000\*10000’e kadar sırayla belirlenen matrisler yukarıda ki matris çarpımı yapan kod parçacığı ile sonuç matrisini oluşturacaktır. En içte bulunan ‘for’ döngüsü içerisinde ki kod parçası her bir adımda sonuç matrisinin bir elemanını hesaplar.

**2.2 Belirlenen matrislerin çarpım sonuçları ve işlemci üzerinde ki performans durumu**

Aşağıda 500\*500’den 10000\*10000’e kadar belirlenen matrislerin çarpım sonuçlarının değerlendirilmesi yapılacaktır.

* 500\*500 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 770 Milisaniye’de işlem tamamlanmıştır.



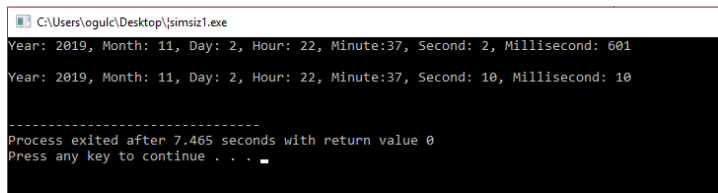
Şekil – 4 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



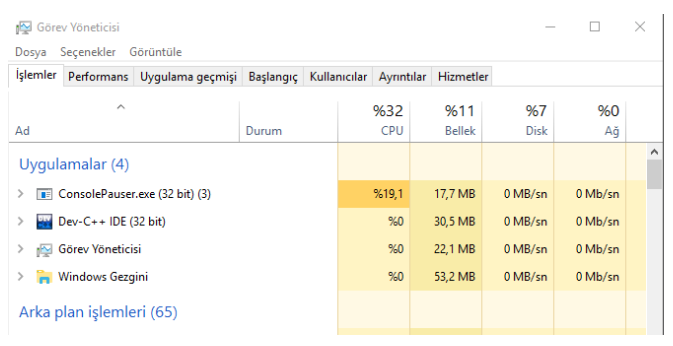
Şekil – 5 Bellek ve işlemci performansı

* 1000\*1000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 8 Saniye 409 Milisaniye’de işlem tamamlanmıştır.



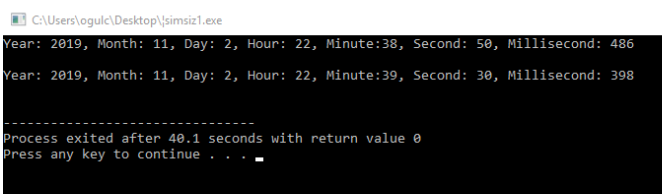
Şekil – 6 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



Şekil – 7 Bellek ve işlemci performansı

* 1500\*1500 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 39 Saniye 912 Milisaniye’de işlem tamamlanmıştır.



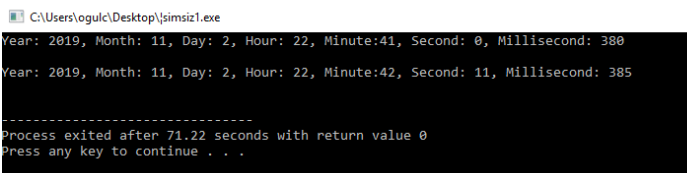
Şekil – 8 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



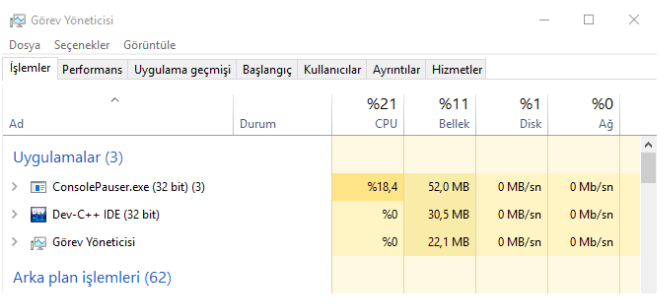
Şekil – 9 Bellek ve işlemci performansı

* 2000\*2000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 1 Dakika 11 Saniye 5 Milisaniye’de işlem tamamlanmıştır.



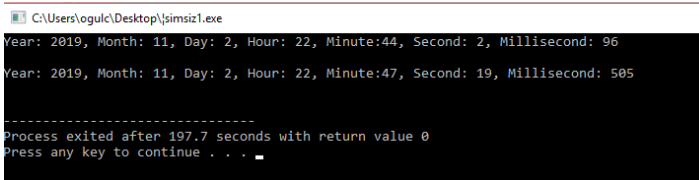
Şekil – 10 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



Şekil – 11 Bellek ve işlemci performansı

* 2500\*2500 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 3 Dakika 17 Saniye 409 Milisaniye’de işlem tamamlanmıştır.



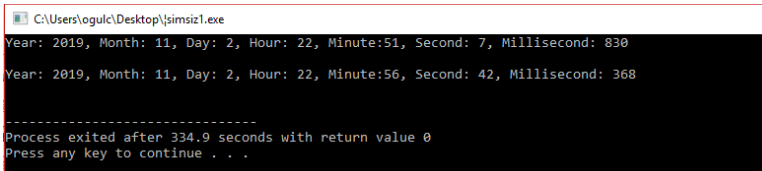
Şekil – 12 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



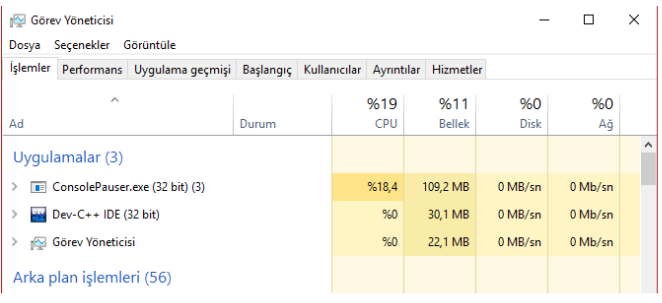
Şekil – 13 Bellek ve işlemci performansı

* 3000\*3000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 5 dakika 36 Saniye 538 Milisaniye’de işlem tamamlanmıştır.



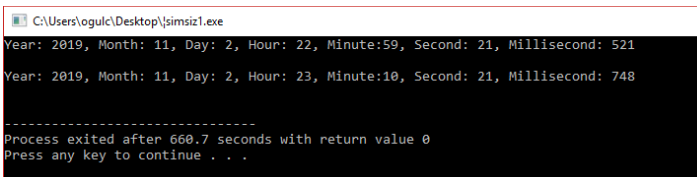
Şekil – 14 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



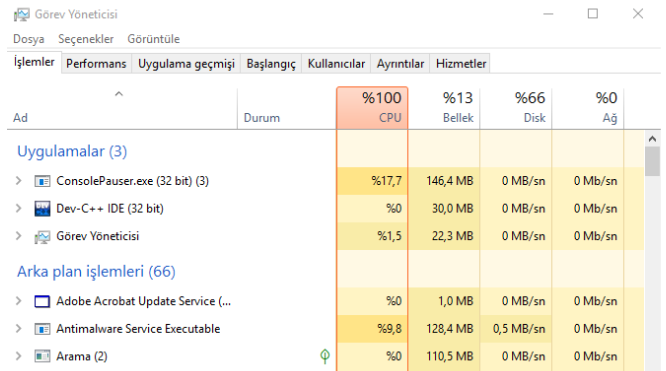
Şekil – 15 Bellek ve işlemci performansı

* 3500\*3500 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 11 Dakika 227 Milisaniye’de işlem tamamlanmıştır.



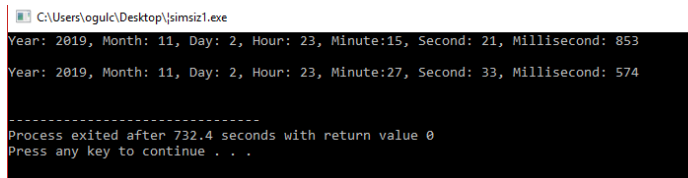
Şekil – 16 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



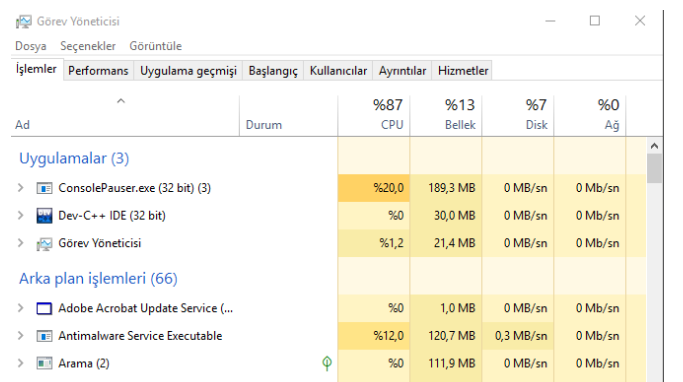
Şekil – 17 Bellek ve işlemci performansı

* 4000\*4000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 12 Dakika 1 Saniye 227 Milisaniye’de işlem tamamlanmıştır.



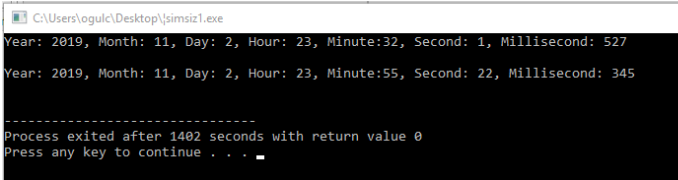
Şekil – 18 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



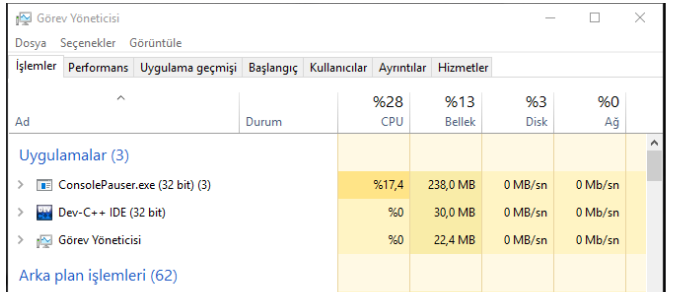
Şekil – 19 Bellek ve işlemci performansı

* 4500\*4500 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 23 Dakika 30 Saniye 818 Milisaniye’de işlem tamamlanmıştır.



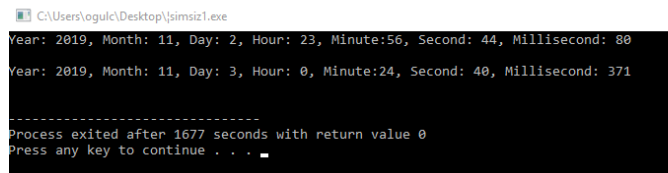
Şekil – 20 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



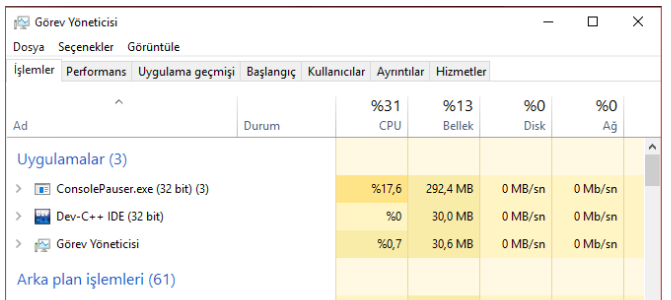
Şekil – 21 Bellek ve işlemci performansı

* 5000\*5000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 27 Dakika 56 Saniye 291 Milisaniye’de işlem tamamlanmıştır.



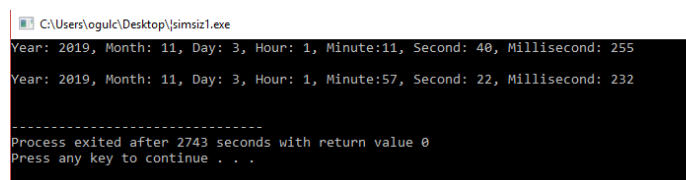
Şekil – 22 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



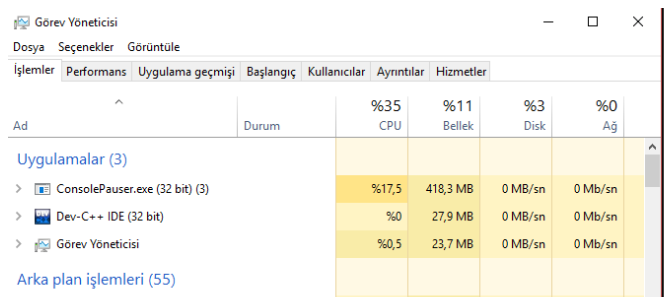
Şekil – 23 Bellek ve işlemci performansı

* 6000\*6000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 45 Dakika 41 Saniye 977 Milisaniye’de işlem tamamlanmıştır.



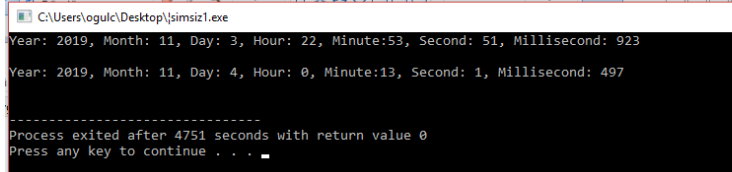
Şekil – 24 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



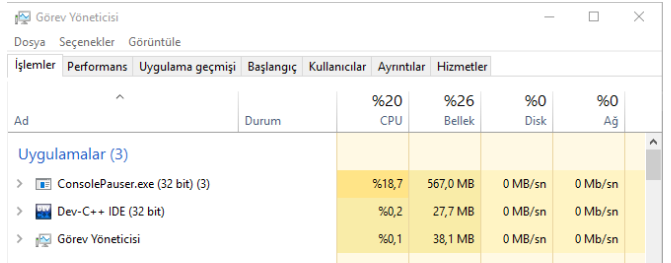
Şekil – 25 Bellek ve işlemci performansı

* 7000\*7000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 1 Saat 19 Dakika 9 Saniye 574 Milisaniye’de işlem tamamlanmıştır.



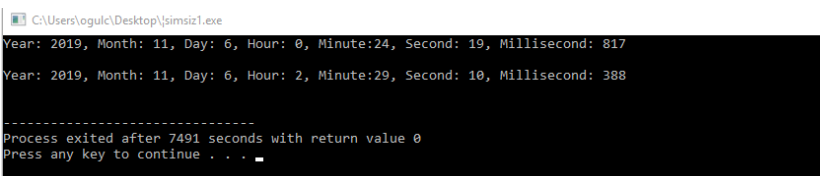
Şekil – 26 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



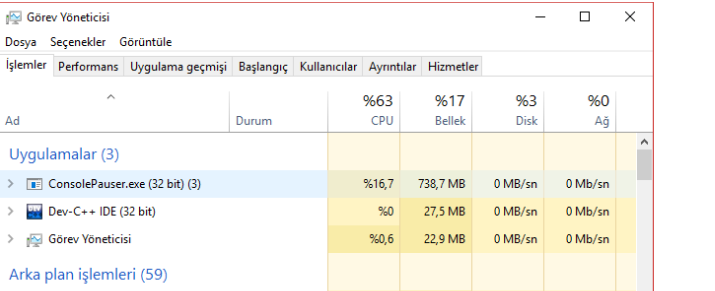
Şekil – 27 Bellek ve işlemci performansı

* 8000\*8000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 2 Saat 4 Dakika 50 Saniye 571 Milisaniye’de işlem tamamlanmıştır.



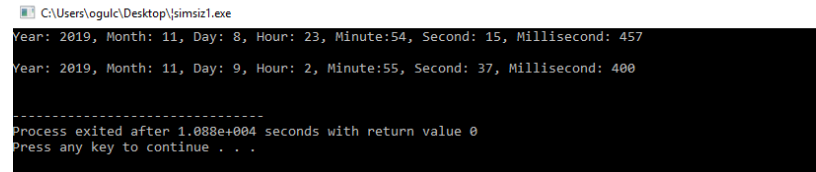
Şekil – 28 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



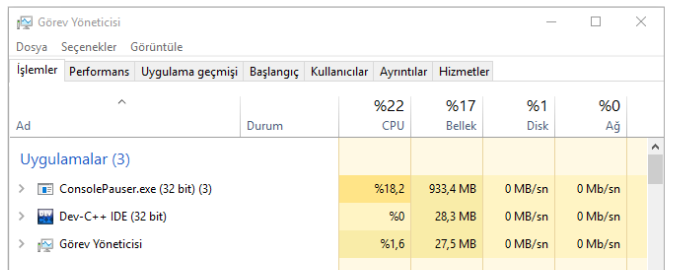
Şekil – 29 Bellek ve işlemci performansı

* 9000\*9000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 3 Saat 1 Dakika 21 Saniye 949 Milisaniye’de işlem tamamlanmıştır.



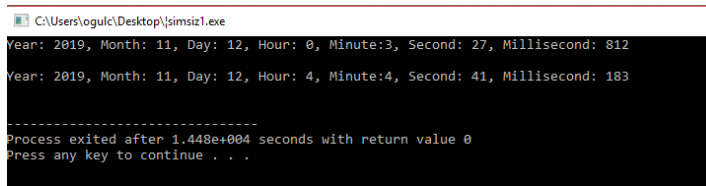
Şekil – 30 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.



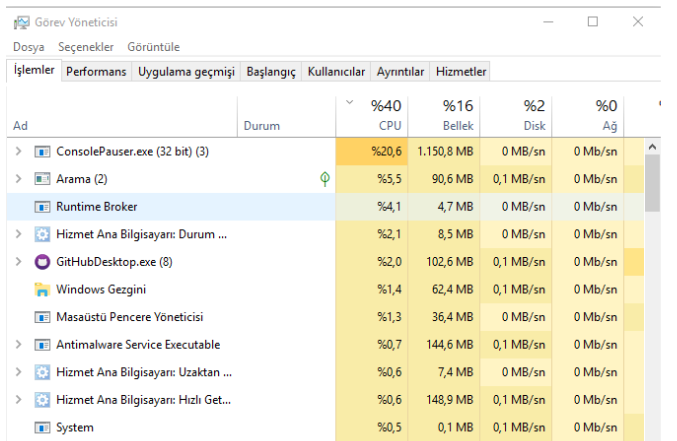
Şekil – 31 Bellek ve işlemci performansı

* 10000\*10000 matris
  + İşlem süresi aşağıda ki şekilde gösterilmiştir. 4 Saat 1 Dakika 13 Saniye 271 Milisaniye’de işlem tamamlanmıştır.



Şekil – 32 İşlem süresi

* + Çarpım anında bellek ve işlemcinin durumu aşağıda ki şekilde gösterilmiştir.

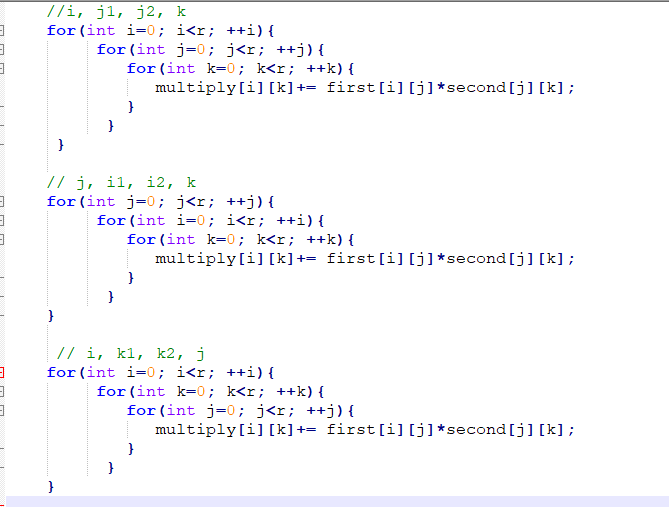


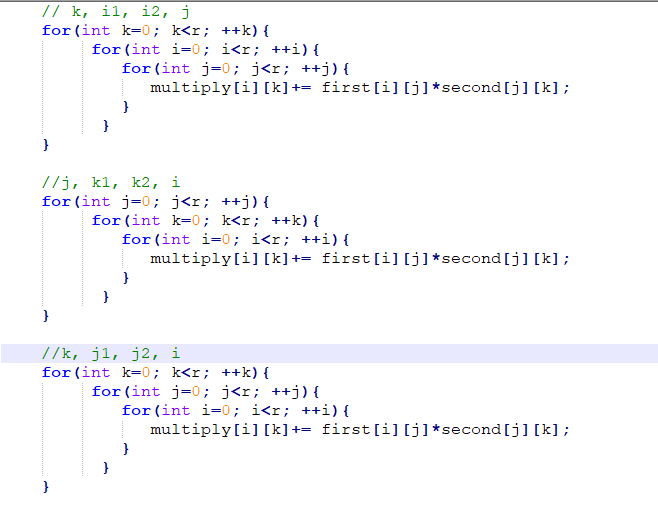
Şekil – 33 Bellek ve işlemci performansı

İşlem süreleri katlayarak artan bir süre biçimde artmaktadır. Bellek ve işlemci performansları incelendiğinde bütün çarpımlarda, çarpım esnasında işlemci %17-20 arasında çalışmaktadır. Sürekli bu aralıkta çalışma sebebi ise Windows işletim sistemlerinde tek ‘thread’ üzerinde işlemi yapıyor olmasından dolayı %20 sınırını geçmemektedir. Çarpım işlemi olduğu sırada bilgisayara ‘ I/O’ işlemi geldiğinde işlemci tavan yapmaktadır. Bellekte ise ilgili matrislerin boyutları kadar yer ayırdığımız için boyut büyüdükçe bellekte ayrılan miktar büyümektedir.

**2.2 En hızlı çarpım yapan ‘for’ döngüsünün bulunması**

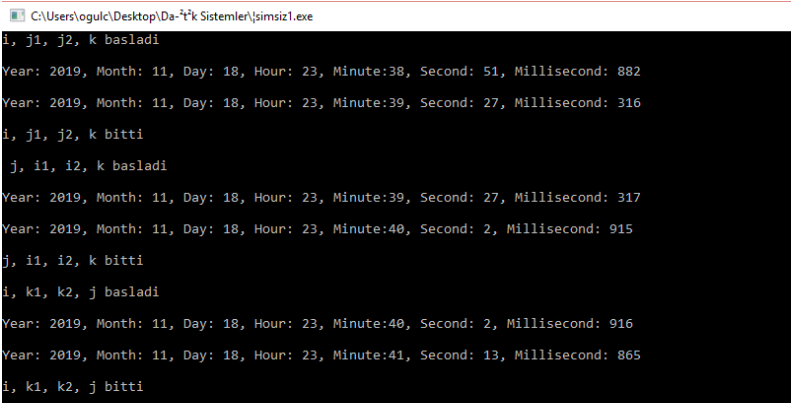
2 Matrisin çarpımını gerçekleştiren kod parçasında, iç içe 3 tane ‘for’ döngüsü bulunmaktadır. Çarpma işlemi bu 3 ‘for’ döngüsünün bitmesi ile sonuçlanmaktadır. ‘for’ döngüsünde bulunan kod parçacığında ki basit değişiklikler ile ön bellek kullanım optimizasyonu değiştirilerek matrislerin daha hızlı çarpması sağlanmaktadır. 3 ‘for’ döngüsünde bulunan i,j,k elemanlarının sıraları değiştirilerek 6 farklı sonuç elde edilmektedir. Aşağıda ilk önce 6 farklı çarpma işleminin kod parçası ve 2000, 6000, 10000 boyutlu dizilerin 6 farklı çarpma sırasına göre çarpma süreleri bulunmaktadır.

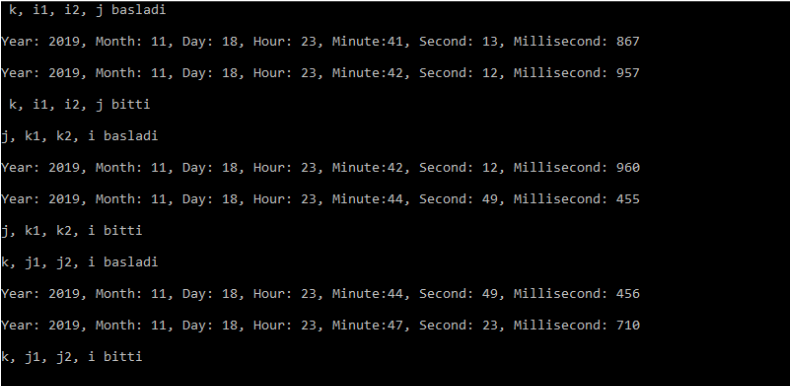




Şekil – 34 6 farklı varyasyona göre matris işlemi

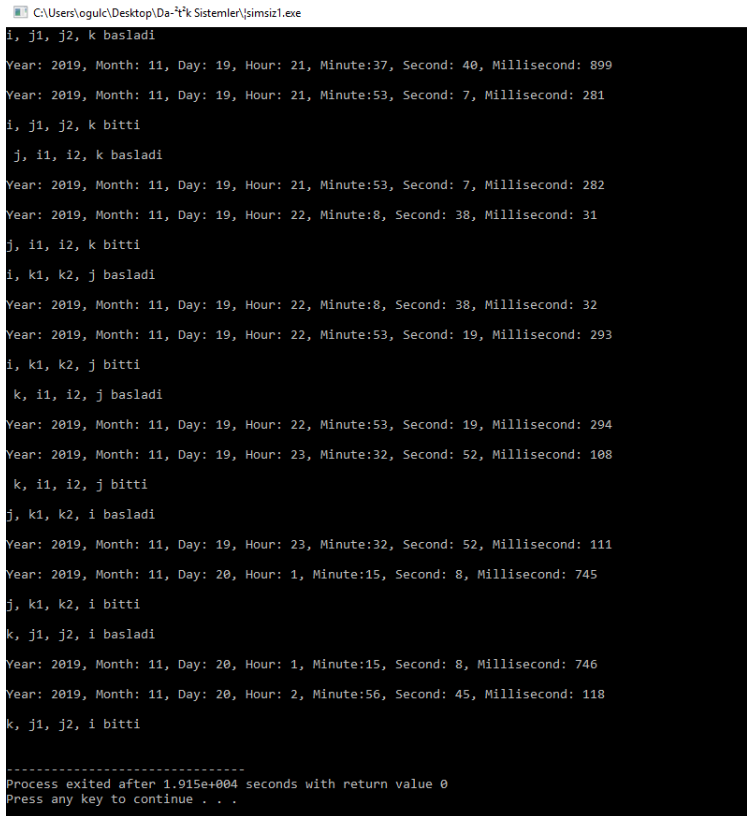
* 2000\*2000 matrisin 6 farklı ‘for’ döngüsüne göre işlem süresi sonuçları şu şekildedir.
  + i, j1, j2, k -> 35 Saniye 434 Milisaniye (1)
  + j, i1, i2, k -> 35 Saniye 598 Milisaniye (2)
  + i, k1, k2, j -> 1 Dakika 10 Saniye 949 Milisaniye (4)
  + k, i1,i2, j -> 59 Saniye 194 Milisaniye (3)
  + j, k1, k2, i -> 2 Dakika 36 Saniye 495 Milisaniye (6)
  + k, j1, j2, i -> 2 Dakika 34 Saniye 254 Milisaniye (5)





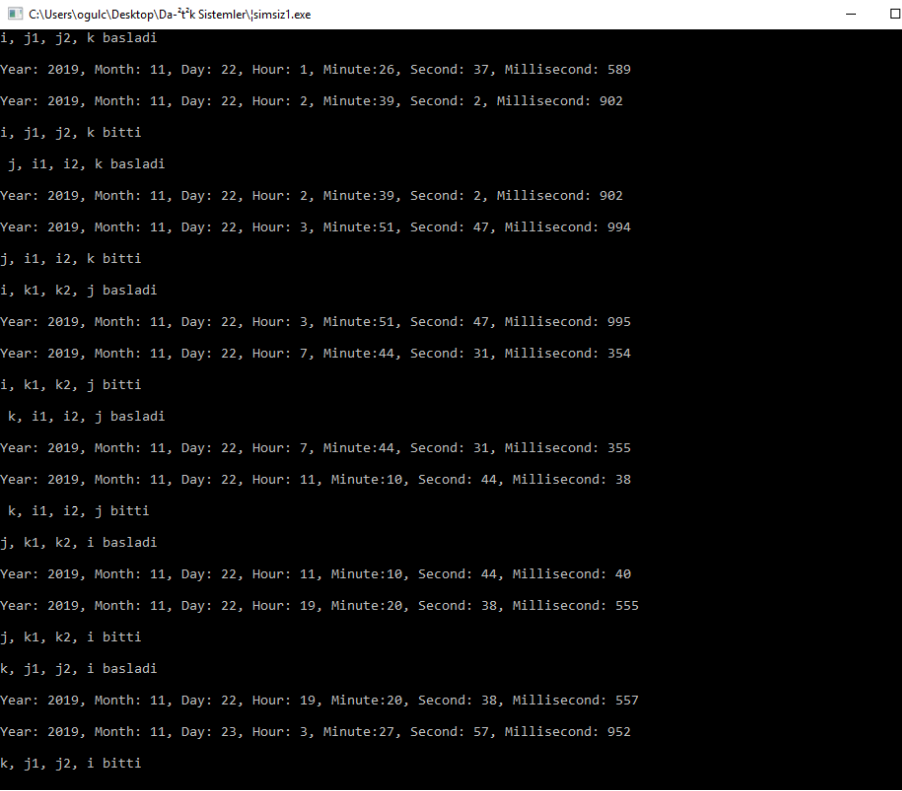
Şekil – 35 6 farklı varyasyona göre çarpım sonucu

* 6000\*6000 matrisin 6 farklı ‘for’ döngüsüne göre işlem süresi sonuçları şu şekildedir.
  + i, j1, j2, k -> 15 Dakika 26 Saniye 382 Milisaniye (1)
  + j, i1, i2, k -> 15 Dakika 30 Saniye 749 Milisaniye (2)
  + i, k1, k2, j -> 44 Dakika 41 Saniye 261 Milisaniye (4)
  + k, i1,i2, j -> 39 Dakika 32 Saniye 814 Milisaniye (3)
  + j, k1, k2, i -> 1 Saat 42 Dakika 16 Saniye 634 Milisaniye (6)
  + k, j1, j2, i -> 1 Saat 41 Dakika 36 Saniye 372 Milisaniye (5)



Şekil – 36 6 farklı varyasyona göre çarpım sonucu

* 10000\*10000 matrisin 6 farklı ‘for’ döngüsüne göre işlem süresi sonuçları şu şekildedir.
  + i, j1, j2, k -> 1 Saat 2 Dakika 25 Saniye 313 Milisaniye (1)
  + j, i1, i2, k -> 1 Saat 12 Dakika 45 Saniye 092 Milisaniye (2)
  + i, k1, k2, j -> 3 Saat 52 Dakika 43 Saniye 359 Milisaniye (4)
  + k, i1,i2, j -> 3 Saat 26 Dakika 12 Saniye 683 Milisaniye (3)
  + j, k1, k2, i -> 8 Saat 9 Dakika 54 Saniye 515 Milisaniye (6)
  + k, j1, j2, i -> 8 Saat 7 Dakika 19 Saniye 395 Milisaniye (5)



Şekil – 37 6 farklı varyasyona göre çarpım sonucu

3 farklı Matris çarpımı incelendiğinde, en hızlı çarpım yapan ‘for’ döngüsünün **i, j1, j2, k** sırasında olan döngü olduğu anlaşılmıştır. En yavaş çarpım yapan ‘for’ döngüsünün **j, k1, k2 i** sırasında olan döngü olduğu anlaşılmıştır. Ayrıca bütün matris çarpımlarında, 6 tane for döngüsünün en hızlıdan en yavaşa olan sırası hiçbir zaman değişmemesi gerekmektedir.

**2.2 Seri programın hızlandırılması ve optimizasyonu**

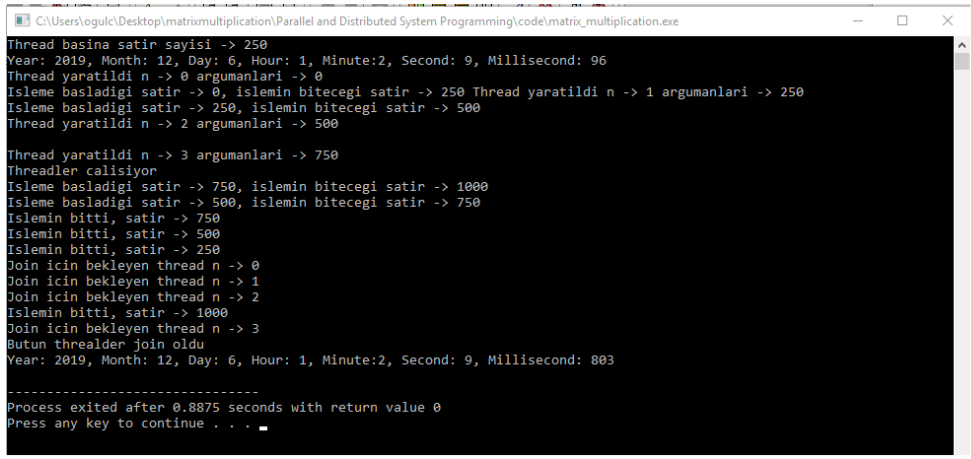
İlgili matris çarpımı yapılırken, sırası ile 3 tane ‘for’ döngüsü çalışarak, sonuç matrisinde ki her bir satır ve sütün elemanları bulunmaktadır. Matris çarpımının hızlandırılıp optimize edilmesi için, bir çok yol bulunmaktadır. Bu bölümde optimizasyon işlemi ‘loop-unrolling’ metodu ile sağlanacaktır.

‘loop-unrolling’ optimizasyonu Türkçe’de döngü açma olarak bilinmektedir. Programın çalışma süresini en iyiye getiren bir tekniktir. Temel olarak döngülerin sayısını azaltır veya kaldırır[4,5,6]. Matris çarpımında ise, ‘threadler’ kullanılarak döngülerin sayısı farklı ‘threadlere’ paylaştırılarak hızlanma kaydedilmesi hedeflenmektedir. ‘threadler’ kullanılarak, optimizasyon sağlanacak matris çarpımları aşağıdaki şekildedir.

* 1000\*1000
* 2000\*2000
* 2500\*2500
* 3000\*3000
* 4000\*4000
* 5000\*5000
* 6000\*6000
* 10000\*10000

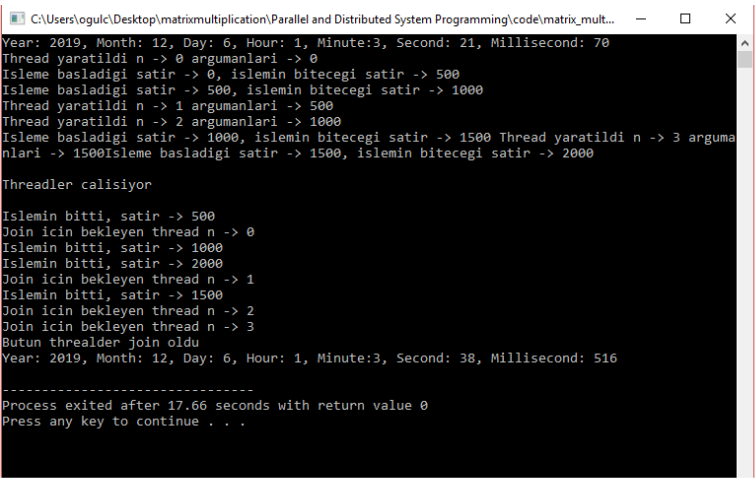
Matrisler çarpılırken, 4 tane ‘thread’ üzerinde işlemler yapılacaktır. Her bir ‘threadin’ oluşturulmasında, işlemin yapılmasında ve ‘threadlerin’ sonucu oluştururken hata durumları kontrol edilecektir. Her bir ‘threade’ hangi satır ve sütunları çarpılacağı bilgisi verilerek, aynı anda birden fazla satır ve sütun üzerinde çarpma işlemi olacağı için döngü sayısı azalmış olacaktır[7,8].

* 1000\*1000 2 matris çarpımının 4 ‘thread’ ile çarpım sonucu
  + İşlem süresi 707 Milisaniye.



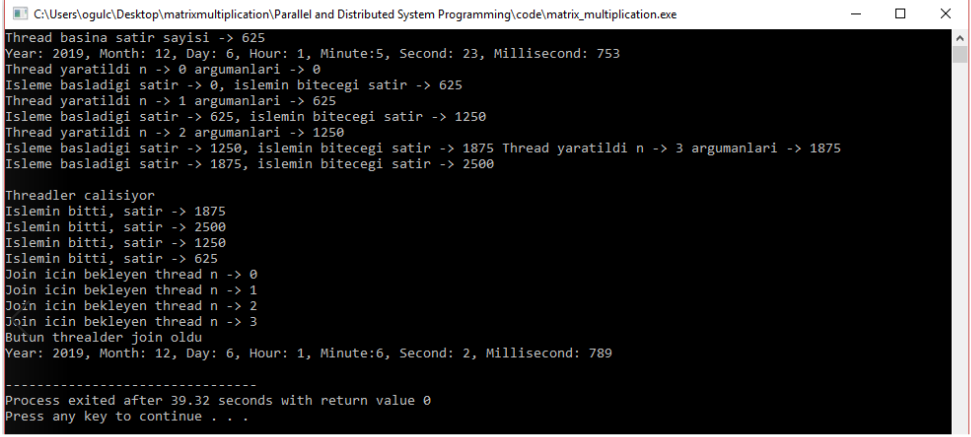
Şekil – 38 1000\*1000 matrisin 4 ‘thread’ ile çarpım sonucu

* 2000\*2000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 17 Saniye 446 Milisaniye.
  + CPU durumu, çarpım işlemi süresince %65-70 arasında çalışmıştır.
  + En iyi i,j,k sıralamasına göre 2 kat daha hızlı çalışmıştır.



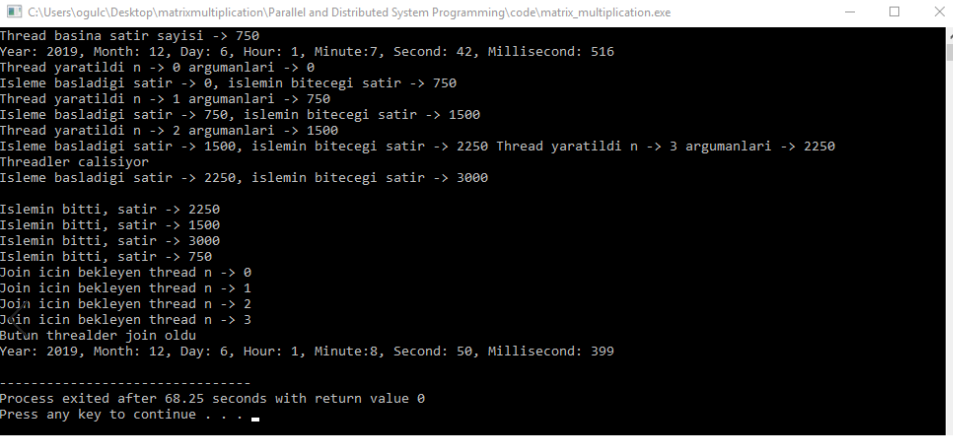
Şekil – 39 2000\*2000 matrisin 4 ‘thread’ ile çarpım sonucu

* 2500\*2500 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 39 Saniye 36 Milisaniye
  + CPU durumu, çarpım işlemi süresince %65-70 arasında çalışmıştır.



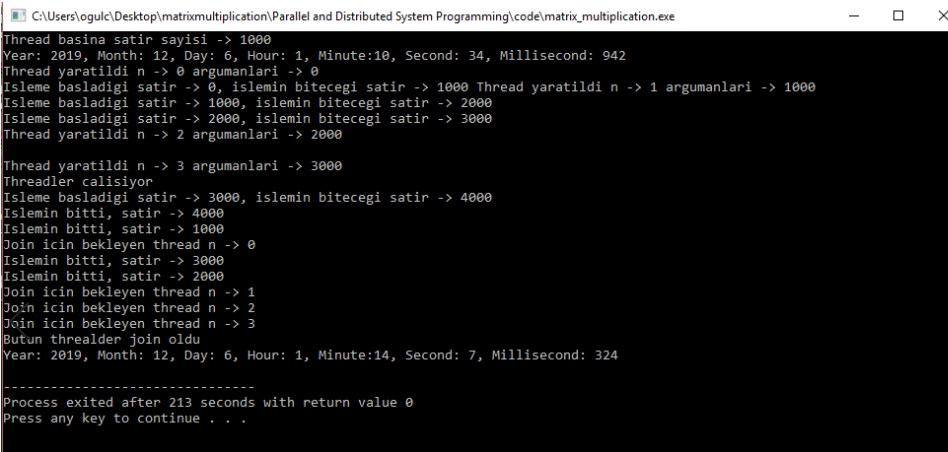
Şekil – 40 2500\*2500 matrisin 4 ‘thread’ ile çarpım sonucu

* 3000\*3000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 1 Dakika 7 Saniye 883 Milisaniye
  + CPU durumu, çarpım işlemi süresince %70-75 arasında çalışmıştır.



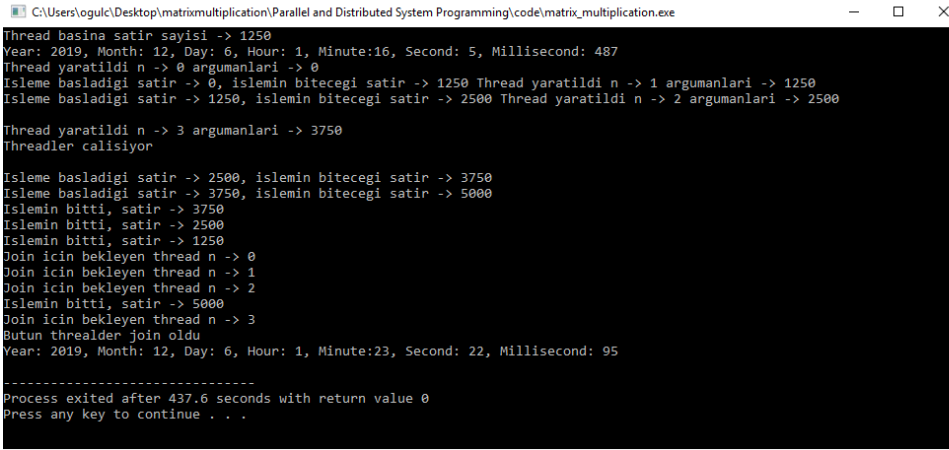
Şekil – 41 3000\*3000 matrisin 4 ‘thread’ ile çarpım sonucu

* 4000\*4000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 3 Dakika 32 Saniye 372 Milisaniye
  + CPU durumu, çarpım işlemi süresince %70-100 arasında çalışmıştır.



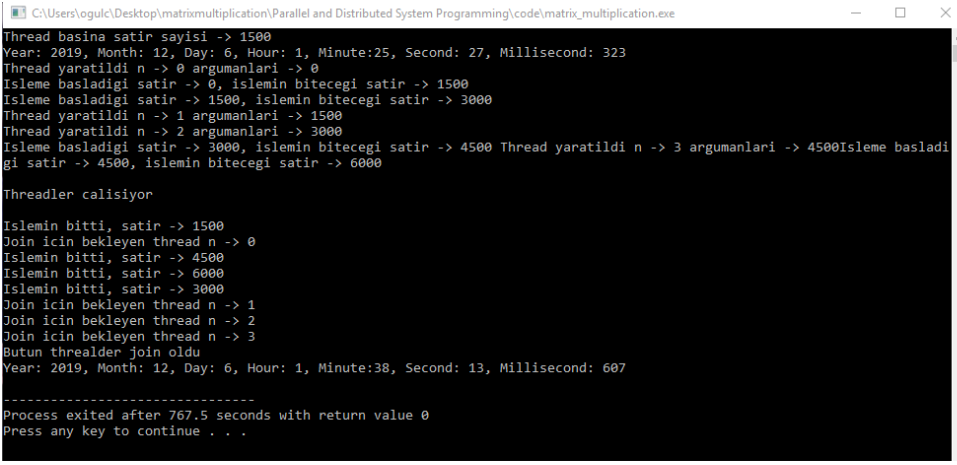
Şekil – 42 4000\*4000 matrisin 4 ‘thread’ ile çarpım sonucu

* 5000\*5000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 7 Dakika 16 Saniye 608 Milisaniye
  + CPU durumu, çarpım işlemi süresince %70-100 arasında çalışmıştır.



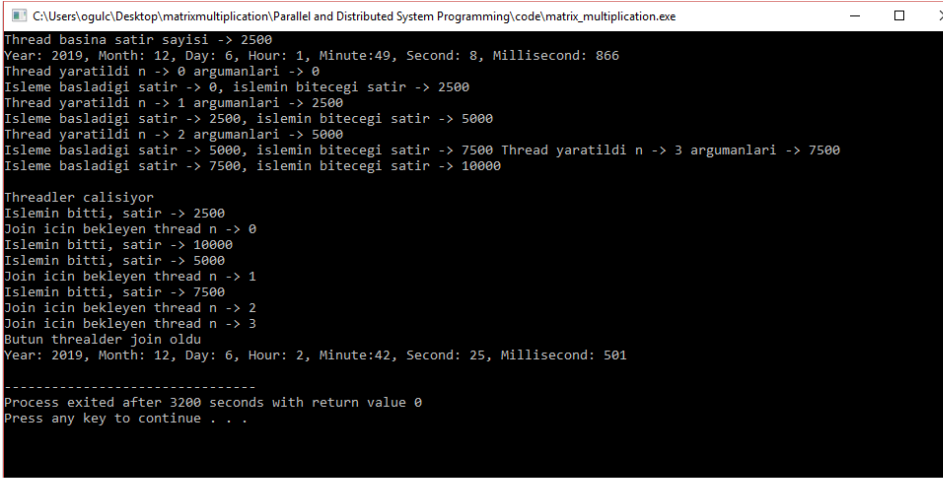
Şekil – 43 5000\*5000 matrisin 4 ‘thread’ ile çarpım sonucu

* 6000\*6000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 12 Dakika 46 Saniye 284 Milisaniye
  + CPU durumu, çarpım işlemi süresince %70-100 arasında çalışmıştır.
  + En iyi i,j,k sıralamasına göre 3 dakika daha hızlı çalışmıştır.



Şekil – 44 6000\*6000 matrisin 4 ‘thread’ ile çarpım sonucu

* 10000\*10000 2 matris çarpımının 4 ’thread’ ile çarpım sonucu
  + İşlem süresi 54 Dakika 8 Saniye 635 Milisaniye
  + CPU durumu, çarpım işlemi süresince %70-100 arasında çalışmıştır.
  + En iyi i,j,k sıralamasına göre 8 dakika daha hızlı çalışmıştır.



Şekil – 45 10000\*10000 matrisin 4 ‘thread’ ile çarpım sonucu

**Referanslar**

1. Young C.Y. Precalculus. John Wiley Sons, 2010.

2. Leise K.B.T. The linear algebra behind Google, SIAM Review, 48:3 569-581, 2006

3. Godsil C.R. Gordon Algebraic Graph Theory, Springer-Verlag, 2004.

4. <https://www.d.umn.edu/~gshute/arch/loop-unrolling.xhtml>

5.<http://astroa.physics.metu.edu.tr/MANUALS/intel_ifc/mergedProjects/optaps_for/common/optaps_hlo_unrl.htm>

6. <https://www.geeksforgeeks.org/loop-unrolling/>

7. <https://www.includehelp.com/articles/threading-in-c-programming-language-with-gcc-linux.aspx>

8. <https://www.tutorialspoint.com/multithreading-in-c>