Yüksek Başarımlı Hesaplama ve Paralel Programlama Yaz Çalıştayı

MPI Kütüphanesi Kullanarak Paralel Matris Çarpımı

1. Öncelikle seri matris çarpımı kodu oluşturulmuş ve performansı ölçülmüştür. Çalıştayın son günlerinde anlatılan '*fox*' ve '*cannon*' algoritmalarıyla karşılaştırılmıştır. Hazırlanan standart matris çarpma algoritması her ikisinden de çok daha hızlı çalıştığından, diğer algoritmalar kullanılmamıştır.

Tablo 1: Seri Matris Çarpımı Algoritması

```
for(i=0; i<K; ++i)
        for(j=0; j<L; ++j)
                for(k=0; k<M; ++k)
                        c[i][k] += a[i][j] * b[j][k]
```

Kodu yukarıda verilen standart matris çarpımı algoritması iki şekilde hızlandırılmıştır:

i. Veriyi yerelleştirme: ön belleğe alınan veriyi olabildiğince fazla kullan

ii. İşaretçiler yardımıyla hızlı gezme

Hızlandırılmış olan seri kod aşağıda verilmiştir.

NOT: Yapılan yerelleştirme, sadece matris boyutunun 40'ın katları şeklinde seçildiğinde işimizi görür. Aksi taktirde hesaplanmayan bölümler kalacaktır!

Tablo 2: Hızlandırılmış Seri Matris Çarpımı Algoritması

```
for(ii=0; ii<n/40; ++ii)
        for(jj=0; jj<n/8; ++jj)
                for(i=40*ii; i<40*(ii+1); ++i)
                {
                        aa = &a[i * n + jj*8];
                        for(j=jj*8; j<(jj+1)*8; ++j)
                        {
                                cc = &c[i * n];
                                bb = &b[j * n];

                                for(k=0; k<n; ++k)
                                        *cc++ += *aa * *bb++;
                                ++aa;
                        }
                }
```

ser klasörünün altındaki kodun derlenmesi, çalıştırılması ve elde edilen sonuçlar aşağıdaki tabloda verilmiştir.

Tablo 3: Seri Kodun Çalıştırılması

```
[du025@d180 ser]$ icc du025seri.c -o du025seri_icc.x
[du025@d180 ser]$ icc -O3 du025seri.c -o du025seri_icc_O3.x
[du025@d180 ser]$ gcc -O3 du025seri.c -o du025seri_gcc_O3.x
[du025@d180 ser]$ gcc du025seri.c -o du025seri_gcc.x
[du025@d180 ser]$ ./du025seri_gcc.x ; ./du025seri_gcc_O3.x ; ./du025seri_icc.x ; ./du025seri_icc_O3.x
  (1000 x 1000) -> duration = 5290      ms
  (2000 x 2000) -> duration = 42220     ms
  (3000 x 3000) -> duration = 142530    ms
  (4000 x 4000) -> duration = 337940    ms
  (5000 x 5000) -> duration = 659530    ms

  (1000 x 1000) -> duration = 1330      ms
  (2000 x 2000) -> duration = 10470     ms
  (3000 x 3000) -> duration = 35330     ms
  (4000 x 4000) -> duration = 83760     ms
  (5000 x 5000) -> duration = 164300    ms

  (1000 x 1000) -> duration = 1330      ms
  (2000 x 2000) -> duration = 10480     ms
  (3000 x 3000) -> duration = 35320     ms
  (4000 x 4000) -> duration = 83740     ms
  (5000 x 5000) -> duration = 163990    ms

  (1000 x 1000) -> duration = 1000      ms
  (2000 x 2000) -> duration = 7740      ms
  (3000 x 3000) -> duration = 27440     ms
  (4000 x 4000) -> duration = 61580     ms
  (5000 x 5000) -> duration = 134880    ms
```

2. Denenmesi istenen değerler herhangi bir bellek sorununa sebep olmadığından, paralelleştirme işlemi hesaplanacak matrisin satırları işlemciler arasında paylaştırılarak gerçeklenmiştir.
   Paralel algoritmanın detayları aşağıda verilmiştir:
   i. Ana işlemci, A ve B matrislerini rasgele olarak oluşturur
   ii. Ana işlemci, A ve B matrislerini **BROADCAST** ile diğer işlemcilere dağıtır
   iii. Her işlemci, kendi satırlarını hesaplar
   iv. İşlemciler sırayla elde ettikleri değerleri ana işlemciye **MPI_Bsend** ile gönderirler
   v. Ana işlemci toplam süreyi hesaplar ve ekrana basar

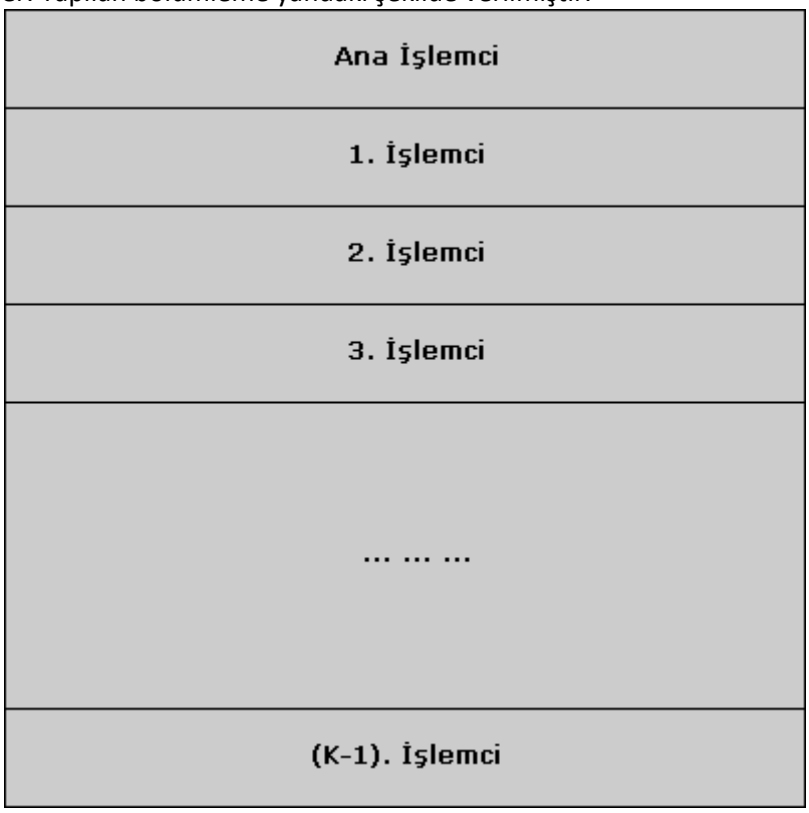
   Algoritmada yer alan for döngülerinin sınırları üzerinde titizlikle çalışılmış ve her türlü farklı işlemci sayısı için çalışması sağlanmıştır.

   par klasörünün altındaki kodun derlenmesi, çalıştırılması ve sonuçların incelenmesi aşağıdaki tabloda verilmiştir.

Tablo 4: Paralel Kodun Çalıştırılması

```
[du025@d180 par]$ icc -O3 du025par.c -o du025par_icc_O3.x
[du025@d180 par]$ for j in `seq 1 20`; do bsub -e error -o output -q workshop.q -a intelmpi -n $j -m anadolu_all
mpirun.lsf ./du025par.x; done
.....................................
[du025@d180 par]$ nano output
```

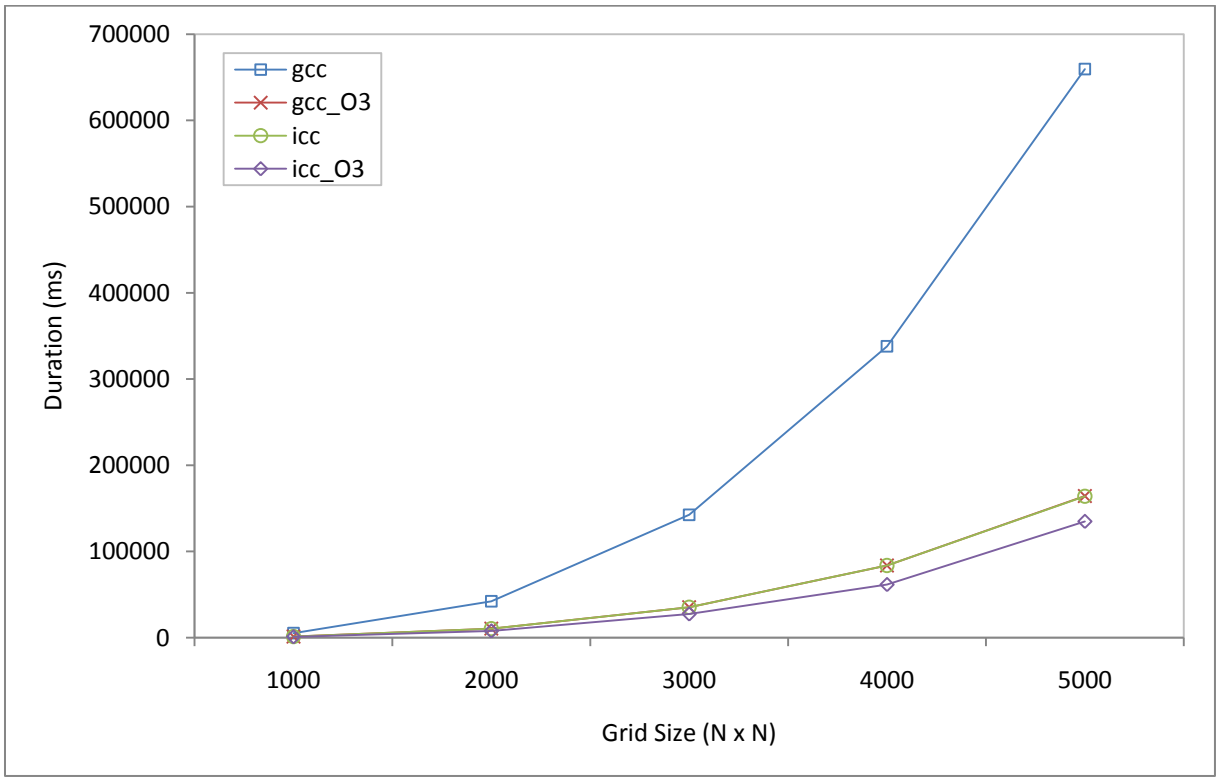
3. Hazırlanan paralel algoritma, basit bir şekilde hesaplanacak olan sonuç matrisini eşit parçalara böler. Yapılan bölümleme yandaki şekilde verilmiştir.



Şekil 1: Hesaplanacak Matrisin İşlemciler Arası Paylaştırılması

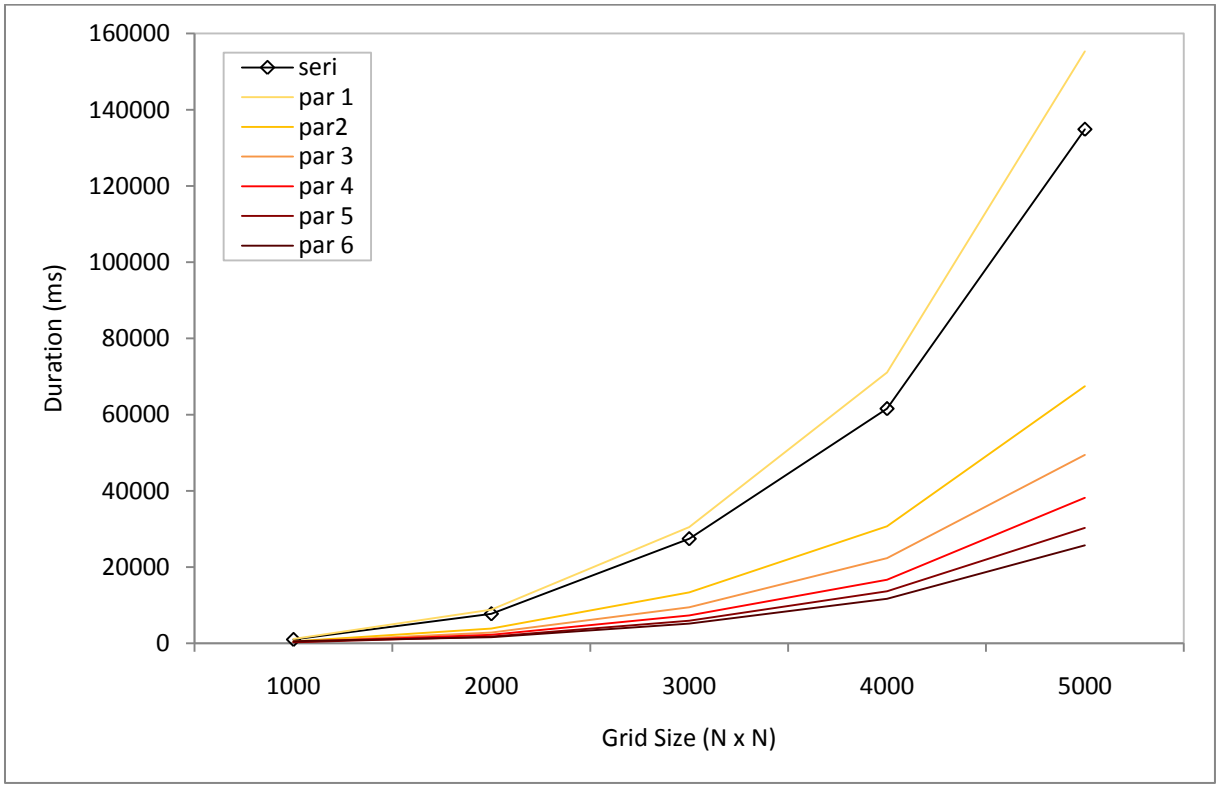
4. 1000 ile 5000 arasında yer alan farklı matrisler için matris çarpımı sonuçları elde edilmiştir. Elde edilen sonuçlar 4 farklı derlemenin sonuçlarıdır. ICC ve GCC derleyicilerinin O2 ve O3 iyileştirmeleri için sonuçlar elde edilmiş ve kıyaslanmıştır. Bu problem için, en hızlı sonuçları O3 iyileştirme seviyesindeki ICC derleyicisi vermiştir.

5. Seri algoritma için elde edilen sonuçlar aşağıdaki grafikte verilmiştir.



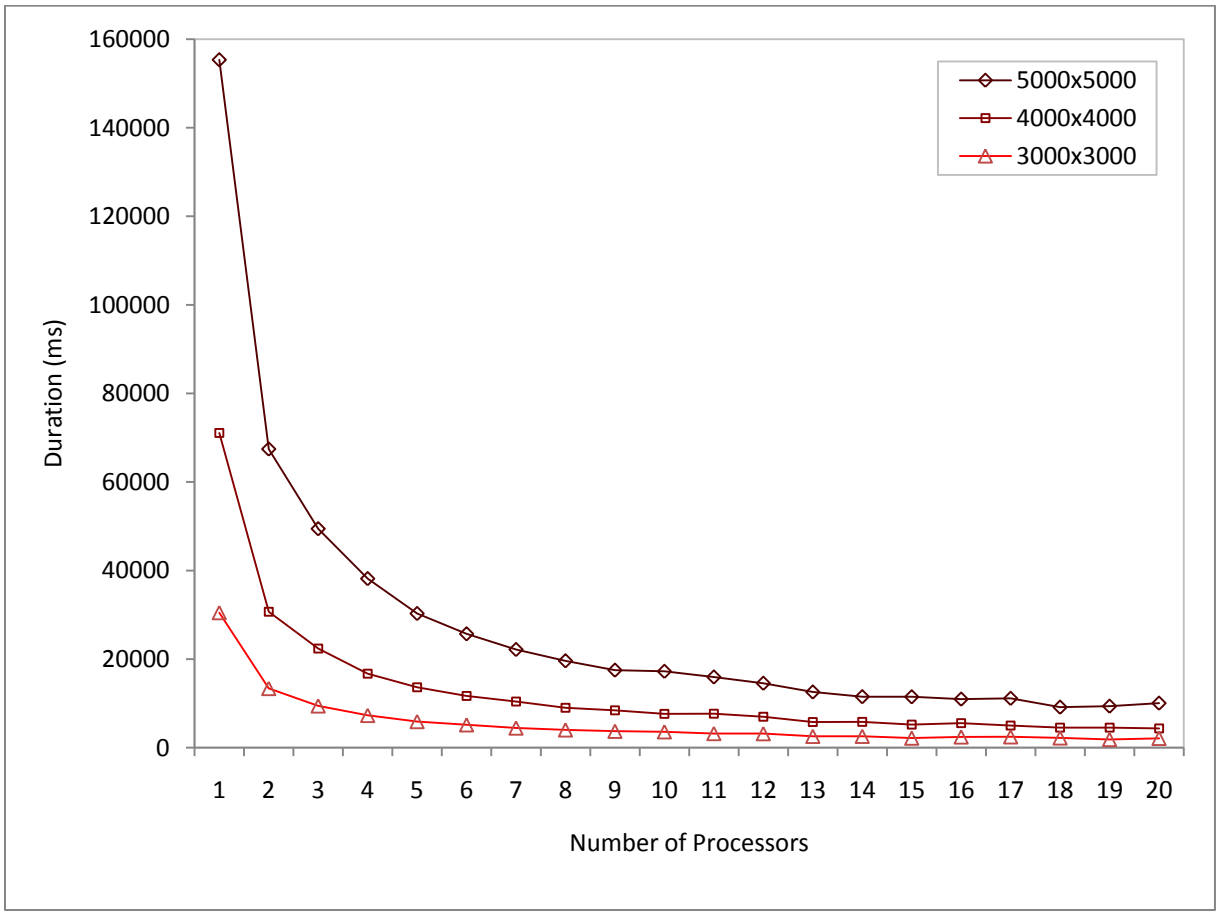
Şekil 2: Seri Algoritma Sonuçları

6. Paralel algoritma için, matris boyutuna göre elde edilen sonuçlar aşağıdaki şekilde verilmiştir.



Şekil 3: Paralel Algoritma Sonuçları

Paralel algoritma için, işlemci sayısının değişiminin etkisi aşağıdaki grafikte verilmiştir.



Şekil 4: Paralel Ortamda İşlemci Sayısının Etkisi

7. Elde edilen ve edilmesi beklenen sonuçları maddeler halinde verelim:

   i. Matris çarpımı problemi (küçük boyutlar için) oldukça iyi parallelleşiyor.
   ii. İşlemci sayısı arttıkça, haberleşmenin artmasından dolayı hızlanma doyuma ulaşıyor.
   iii. Daha büyük matrisler için doyum noktası daha yüksek miktardaki işlemci sayısı olacaktır.
   iv. İşlemci içi matris çarpımı OMP kütüphanesinden yararlanılarak hızlandırılabilir.
   v. Belli bir yerden sonra, işlemcilerin bellekleri matrisin tamamını tutmak için yetersiz olduğunda büyük bir performans düşüşü yaşanacaktır.
   vi. Çok büyük matrisler için; matrisin tamamını paylaştırmak yerine, matrisleri parça parça kullanabiliriz.

8. Gönderiyorum ☺