

Bilgisayar Tabanlı Görselleştirmede Paralel Uygulamalar

Hüseyin Kaya ve Murat Kaplan

İstanbul Teknik Üniversitesi
Bilişim Enstitüsü
80626, Maslak, İstanbul
{hkaya, mkaplan}@be.itu.edu.tr

Özet Bilgisayar tabanlı görselleştirme çalışmaları genel olarak çok fazla zaman ve işlem gücü gerektiren uygulamalardır. Bu nedenle paralel işlemcilerin kullanılması önemli ölçüde zaman kazanımı getirmektedir. Bu çalışmada kaynak kodu açık bir görselleştirme yazılımı olan **POVRAY**, Linux makinalardan oluşan bir küme üzerinde koşulmuştur. Paralleleştirme tabanı olarak **PVM** ve **MPI** kullanılmıştır. Bu kapsamda çeşitli uygulamalar yapılmış ve sonuçlar sunulmuştur.

1 Giriş

Bu çalışmada **POVRAY**'in paralel hesaplama ortamında kullanımını göstereceğiz. Bu maksatla Linux işletim sistemi yüklü 16 adet bilgisayardan oluşmuş bir küme kullanılmıştır.

POVRAY adlı bölümde **POVRAY**'in ne olduğu, kurulumu ve bazı örnekler gösterilecektir. **PVM** ve **MPI** için özel bir bölüm ayırmadık. İlgilenenler bu konferansta **Abbas Ayhan Kanmaz** tarafından sunulan ve iki yazılımda konu alan çalışmayı okuyabilirler[1]. Bir sonraki bölümde **PVM** ve **POVRAY**'in birlikte kullanıldığı bir yazılımdan (**PVM-POVRAY**) bahsediyoruz. Kurulum ve bir örnek program yer alıyor bu bölümde. Hemen ardından **MPI** ve **POVRAY**'in birlikte kullanıldığı başka bir yazılım anlatılacak. Yine kurulum ve örnek bir uygulama yapılacak. Uygulamalar bölümünde ise **POVRAY** camiası tarafından kabul görmüş bazı örnekler bahsedilen küme üzerinde koşulmuş ve elde edilen sonuçlar açıklanmıştır. Sonsöz ve Sonuç bölümünde ise bütün anlatılanlar toparlanmış ve bazı tesbitler yapılmıştır.

Bu noktadan itibaren **POVRAY**'in **PVM** sürümünü **PVM-POVRAY**, **MPI** sürümünü ise **MPI-POVRAY** adı ile anacağız.

2 **POVRAY**

2.1 Giriş: **POVRAY** Nedir?

POVRAY, (Persistence of Vision Ray-Tracer) üç boyutlu gerçekçi görüntüler oluşturmak için kullanılan bir yazılımdır. “ray-tracing” denen görüntü oluşturma tekniğini kullanır. Temel olarak bir giriş dosyasında tanımlanan şekilleri, ışığı

ve kamerayı üç boyutlu sanal bir ortama yerleştirerek kameranın baktığı açıdan görüntüyü oluşturur. Bu teknikle yüksek başarılı üç boyutlu görüntüler, bütün gerçek görüntü etkileşimleri yada ışık oyunları -yansıma, kırılma, görüş açısı gibi- kullanılarak kolaylıkla oluşturulabilir. Ardışık olarak tasarlanan görüntüler sayesinde hareketli sahneler oluşturmak da olasıdır[2].

POVRAY kullanarak oluşturulabilecek görüntülere yüzlerce örnek verilebileceği gibi, kısaca oluşturulamayacak bir görüntünün olmadığını söylemek de mümkündür. POVRAY ile yapabilecekleriniz sadece hayal gücünüz ile sınırlıdır.

2.2 Kurulumu

POVRAY'ın Linux, Mac OS X yada Windows işletim sistemleri için kaynak kodlarına <http://www.povray.org> sanal yöresinden ulaşabilirsiniz. Adım adım yapmanız gereken herşeyi klavuz dosyalarında bulabilirsiniz. Kaynak kodlarla uğraşmak istemiyorsanız belli başlı işletim sistemleri için derlenmiş çalıştırılabilir sürümlerine de aynı adresten erişebilirsiniz.

RPM dosyalarını yeğleyen bir Linux kullanıcısıysanız onun da kolayı var: <http://www.rpmfind.net>. Belli başlı Linux sürümlerinden birinde tam kurulum yaptıysanız yada CD'leri elinizin altındaysa yapmanız gereken yine çok az şey var demektir:

```
rpm -i povray*.rpm.
```

Yukarıdaki seçeneklerden herhangi birinde başınıza gelebilecek herhangi bir sorunda ilk bakmanız gereken yer yine sıkça sorulan sorular olmalı. Desteklendiği belirtilmeyen ortamlardan birinde çalışıyorsanız sizin için en uygunu Unix ortamı için önerilen kaynak kodları kullanarak sisteminize özel bir kurulumu denemek olacaktır.

2.3 Linux Ortamında POVRAY

Linux sisteminde kurulumu başarıyla tamamlanmış bir POVRAY'iniz varsa komut satırından `povray` komutunu girdiğinizde

```
# povray
Persistence of Vision(tm) Ray Tracer
  Version 3.1g.Linux.gcc
  This is an unofficial version compiled by:
  SuSE GmbH, Nuernberg, Germany
  The POV-Ray Team(tm) is not responsible for supporting
  this version.
Copyright 1999 POV-Ray Team(tm)
```

diye başlayıp devam eden,

```

Redirecting options

GI<name>= write all .INI parameters to file name
Gx<name>= write stream x to console and/or file name
          GA - All streams (except status)
          GD - Debug stream
          GF - Fatal stream
          GR - Render stream
          GS - Statistics stream
          GW - Warning stream

```

satırlarıyla da son bulan bir tepki almış olmalısınız. Bu durumda aşağıdaki dosyalara bakmanızın zamanı gelmiş demektir:

```

# cd /usr/share/doc/packages/povray
# ls
CMPL_Unix.doc  compile.doc  povlegal.doc
README.unix   povdoc.ps.gz  revision.doc

```

dosyalarına ulaşabilirsiniz. Özellikle povdoc.ps.gz yaklaşık beş yüz sayfalık oldukça büyük bir kaynak yazıdır. Kurulumundan kullanımına kadar takıldığınız yada takılabileceğiniz birçok soruya cevap bulabilirsiniz.

/usr/local/lib/povray31 dizininde ise temel bütün “ini” , “scene” ve “include” dosyalarına ulaşabilirsiniz. Bu dizin altındaki scenes dizininde ise sayısız örnek bulabilirsiniz.

Bu dizindeki önemli dosyalardan biri de povray.ini dosyasıdır. Sisteminizde kurulu POVRAY için görüntü boyutu, çıktı formatı, kitaplık dosyalarının yeri gibi ön tanımlı bilgiler bu dosyada belirtilmiştir. Kullanıcılar bu dosyayı kendi alanlarına .povrayrc adıyla kaydedip içinde istedikleri değişiklikleri yapabilirler.

2.4 Kullanımı ve Örnekler

POVRAY 'in üç boyutlu bir koordinat sistemi vardır. Bu sistemde kameranın yeri, bakış açısı, cisimlerin yeri ve ışık kaynağı özenle yerleştirilmelidir. Sıfırdan büyük değerler için Y eksenini yukarı, X eksenini sağ tarafa, Z eksenini de görüntünün içine doğrudur. Bu gösterim, sol elin baş (Y), orta (X) ve işaret (Z) parmaklarının uygun şekilde tutulmasıyla daha kolay anlaşılabilir.

Aşağıda örnek bir POVRAY kodu ve görüntüsü verilmiştir. Örnek üzerinde de kolayca görülebileceği gibi, oluşturulacak görüntüde kullanılan bütün nesnelere ayrı ayrı tanımlanarak kod hazırlanır.

```

background { color red 10 green 10 blue 10 }
camera {
  location < 0, 2, -3 > // kamera'nın yerleştirildiği yer
  look_at < 0, 1, 2 > } // kamera'nın baktığı yer

sphere {
  < 0, 1, 2 >, 2 // küre'nin yerleştirildiği yer ve yarıçapı
  texture { // küre'nin dokusu
    pigment { color red 1 green 0.5 blue 1 }
  } }

light_source { //ışık kaynağı
  < 2, 4, -3 > color rgb < 1, 1, 1 > }

```

Şekil 1. Örnek bir POVray programı.

Bu kod içerisinde `include` dosyalarındaki tanımlamalardan hiçbiri kullanılmamış, `texture` tanımlaması kod içerisinde yapılmıştır. İstenirse `include` dosyaları ikinci örnekteki gibi kodun başına eklenerek sayısız şekil, renk ve doku birleşimi kullanılabilir. `/usr/local/lib/povray31/include/` dizinindeki seçeneklerin yetersiz olduğunu düşünüyorsanız <http://texlib.povray.org> adresindeki doku kütüphanesini de kullanabilirsiniz.

Bu kodu aşağıdaki gibi sadece `povray` komutuyla derleyebilirsiniz.

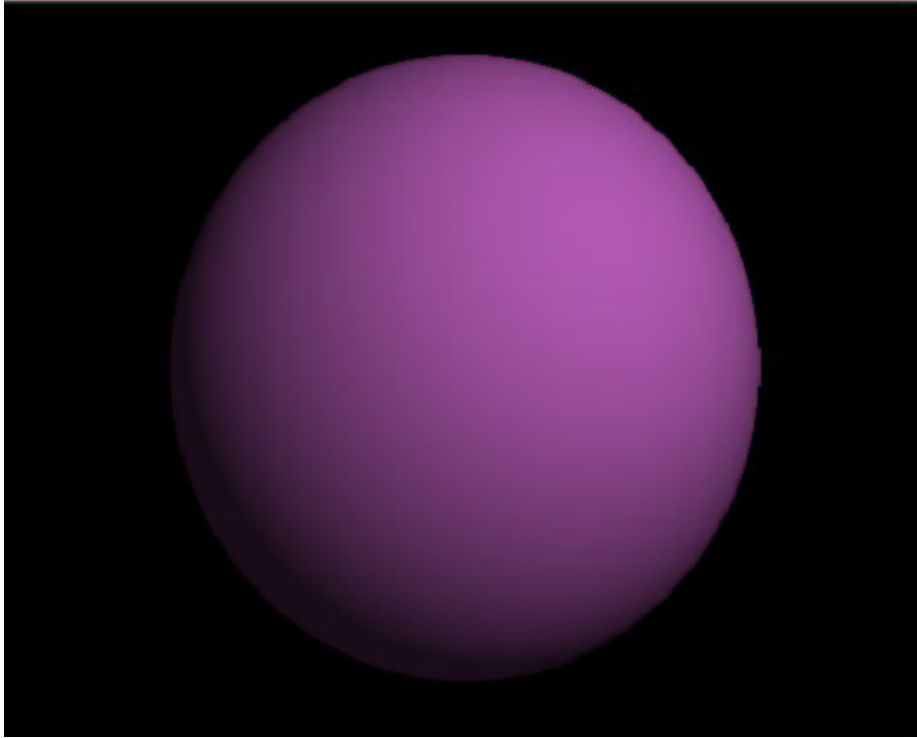
```
# povray -Ikod1.pov
```

Bu durumda `kod1.png` adlı bir görüntü dosyası oluşacaktır. “-O” parametresiyle oluşturulan görüntünün formatını, “-W” ve “-H” ile sırasıyla genişlik ve yüksekliğinin kaç piksel olmasını istediğinizi de belirtebilirsiniz. Örneğin:

```
# povray -Ikod1.pov -Okod1.gif -W500 -H400
```

`include` dosyalarının kullanımı için de Şekil 3 örnek verilebilir. Bu kodu derleyerek de Şekil 4’de görülen resim elde edilir.

Son olarak bir animasyon örneği verelim. İki tane dosya kullanıyoruz. İlki genellikle `ini` uzantısı ile belirtilir. İçine gerekli tanımlamalar yazılır. Örneğimizdeki gibi derlenecek POVray kodunun adı, oluşturulacak çerçevelerin başlangıç ve bitiş sırasayıları bu dosyaya yazılır. Diğerleri ise `pov` uzantılı olur ve gerçek POVray kodu orada bulunur. Dosyanın içindeki parametrelere göre kaç adet çerçeve oluşturulacağı bellidir. Mesela bizim örneğimizde 1’den 30’a kadar yani



Şekil 2. Şekil 1'deki POV-Ray kodunun görüntüsü.

```
#include "colors.inc"
#include "textures.inc"
#include "shapes.inc"
background { color DarkSlateBlue }

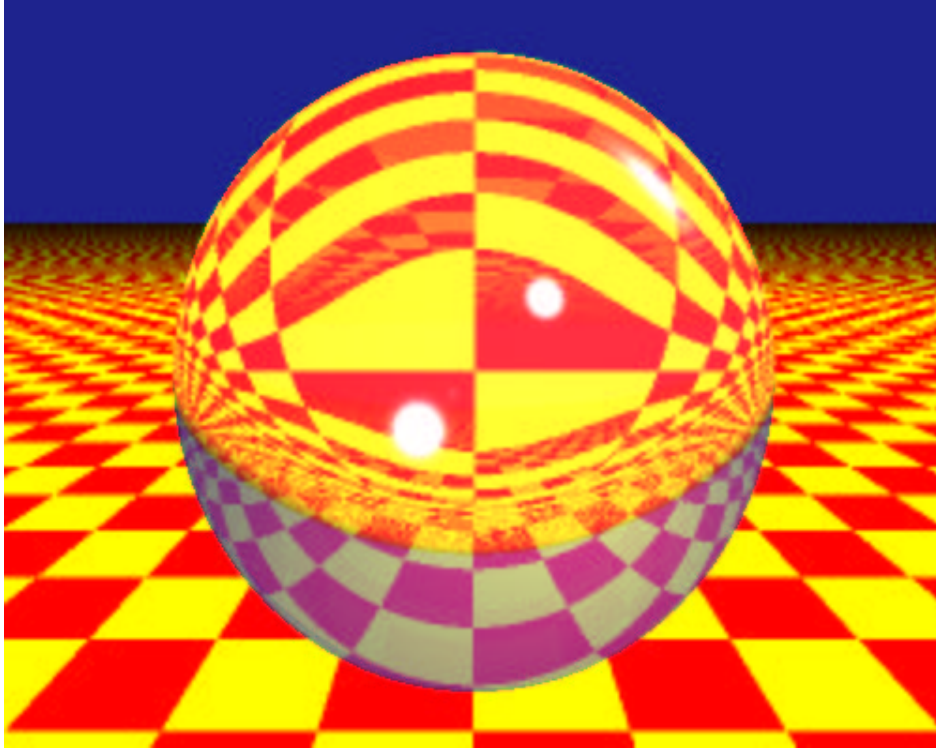
camera { location < 0, 2, -3 >
         look_at < 0, 1, 2 > }

sphere { < 0, 1, 2 >, 2
        texture {Glass3 } }

plane { < 0, 1, 0 >, -1
        pigment { checker color Yellow, color Red }}
light_source { < 2, 4, -3 > color rgb < 10, 10, 10 > }
```

Şekil 3. İkinci örnek POV-Ray programı.

30 adet çerçeve oluşacaktır. Derlerken ini dosyası derlenir. Örneğimiz için Şekil 5'de belirtilen komut yeterli olacaktır.



Şekil 4. Şekil 3'deki POV-Ray kodunun görüntüsü.

```
# povray anim.ini
```

Şekil 5. Animasyonları derlemek için gereken POV-Ray komutu.

```
Input_File_Name=anim.pov  
Initial_Frame=1  
Final_Frame=30  
Initial_Clock=0  
Final_Clock=1  
Pause_when_Done=off  
Cyclic_Animation=on
```

Şekil 6. Örnek bir POV-Ray animasyonu (anim.ini).

```

#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"
camera {
  location < 0, -2, -5 >
  look_at < 0, 1, 4 > }
sphere {
  < 0, 0, 0 >, 1
  texture {
Lightening1 }
}
sphere {
  < 1, 0, 2 >, 0.4
  texture {
  White_Marble }
rotate -y*360*clock }

light_source { < 2, 4, -3 > color rgb < 1, 1, 1 > }
light_source { < 2, 4, -3 > color rgb < 1, 1, 1 > }
light_source { < 2, 4, -3 > color rgb < 1, 1, 1 > }

```

Şekil 7. Örnek bir POV-Ray animasyonu (anim.pov).

3 PVM ve POV-Ray

POV-Ray uygulamaları çok vakit gerektirdiğinden dolayı paralelleştirilmesi önemlidir. Bazı POV-Ray uygulamaları yalnızca bir adet çerçeve resim üretmesine rağmen işlenmesi saatler hatta günler alabilmektedir. Çok ayrıntısı olan veya büyük ölçeklerde işlenmesi gereken resimler bu sınıfa girerler.

POV-Ray'ın PVM kullanılarak paralelleştirilmesi son derece basit bir ilkeye dayanıyor. Buna göre işlenecek resim dikdörtgensel alanlara bölünüyor ve işlemcilerle dağıtılıyor. Her işlemci kendisine düşen parçayı işledikten sonra ana makineye gönderiyor. Ana makine ise parçaları birleştirip resmin son şeklini oluşturuyor. Ana işlemci genelde sadece koordinasyon işini yapar ama istenirse resim işleme de sağlanabilir[3].

3.1 Kurulum

Bu yazılımın yaygınlığı sebebiyle Linux dağıtıcıları genelde RPM paketleri sürümlerinin içerisine koyuyor. Örnekte SuSE benim bildiğim kadarıyla 7.2 serisinden itibaren yazılımın RPM paketini sağlıyor. Paketi kurmak için ise elbette POV-Ray ve PVM yazılımlarının sistemde kurulu olması gerekmektedir. Eğer kendi sisteminiz için uygun bir RPM paketi bulamaz iseniz yazılımın kendi sanal-doku yöresinden paketin kaynak kodlarını indirip derlemeniz mümkün. Fakat POV-Ray'in de kaynak kodlarını indirmeniz gerekecektir.

Bu çalışmada PVM-POVRAY'in 3.1 sürümü kullanıldı. Dolayısıyla POVRA Y'in de 3.1 sürümünü kullandık. En son çıkan POVRA Y sürümü olan 3.5 için yazılım şu an yenileniyor. 3.5 henüz çok yeni olduğu için biz yoğun bir şekilde test edilmiş olan 3.1 sürümünü kullanmayı tercih ettik. Ayrıntılı bilgi için yazılımın ana sayfasına ¹ bir göz atabilirsiniz.

Paket kurulduktan sonra sistemde pvmpov isimli bir komutu görmemiz gerekli.

3.2 Ayarlar

pvmpov komutu standart POVRA Y yazılımının hemen hemen bütün parametlerini desteklemektedir. Ancak ondan önce PVM ortamında hangi makinaların çalışacağını ayarlamamız gerekir. Bunun için komut satırında pvm komutunu yürütün.

```
hkaya@ankara:~ > pvm
pvmd already running.
pvm>
```

Hangi makinaların PVM için hazır olduğunu conf komutuyla görebilirsiniz.

```
pvm> conf
conf
1 host, 1 data format
          HOST      DTID      ARCH      SPEED      DSIG
          ankara    40000    LINUX     1000 0x00408841
pvm>
```

Eğer bu haliyle pvmpov komutunu çalıştırsanız yalnızca ankara isimli makinalarda resim işleme gerçekleşecektir. Eğer işlemi iki makinalara dağıtmak istiyorsanız diğer bilgisayarları eklemelisiniz. Bunun için add komutu kullanılır.

```
pvm> add istanbul
add istanbul
1 successful
          HOST      DTID
          istanbul  100000
pvm>
```

Diyelimki iki makina bizim için yeterli. Çıkmak için quit yazıyoruz. Artık iki makina işlem yapmaya hazır. pvmd still running uyarısı PVM'in kendisini

¹ <http://pvmpov.sourceforge.net/>

```
pvm> quit

quit
Console: exit handler called
pvmd still running.
```

arka plana attığını söylüyor. PVM uygulamasını tamamen durdurmaya istiyorsak `halt` komutunu kullanmamız gerekli.

```
pvm> halt
halt
Terminated
```

PVM-POVRAY'in eksik yanı animasyonlar. Animasyon içerisindeki çerçeveleri üstüste bindiriyor. Hatta bu kaynak yazı içinde kullandığımız 1934'deki zeplin kazasını canlandıran POVRAY çalışmasını PVM-POVRAY ile görüntüleyemedik. Bu yüzden animasyonlar için MPI-POVRAY'i seçtik.

3.3 Örnek Uygulama

Bunu belirttikten sonra resimleri işlemeye başlayabiliriz. POVRAY ile birlikte gelen örnek resimleri kullanabiliriz.

```
# cp /usr/lib/povray31/scenes/advanced/skyvase.pov ~
# /usr/bin/pvmpov +I skyvase.pov +v
Persistence of Vision(tm) Ray Tracer Version 3.1g.Linux.gcc
This is an unofficial version compiled by:
Jakob Flierl <flierl@luga.de> - PVMPov Version 3.1g.1
The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing PVMPov
Spawning /usr/bin/pvmpov with 2 PVM tasks on 2 hosts...
...2 PVM tasks successfully spawned.
Waiting up to 120s for first slave to start...
Slave 0 successfully started.
Parsing Options
Input file: skyvase.pov (compatible to version 3.1)
Remove bounds.....On Split unions.....Off
Library paths: /usr/lib/povray31 /usr/lib/povray31/include
Output Options
Image resolution 320 by 240 (rows 1 to 240, columns 1 to 320).
Output file: skyvase.png, 24 bpp PNG
Graphic display.....Off
Mosaic preview.....Off
CPU usage histogram.Off
```

Continued trace....Off Allow interruption...On
Pause when done....Off
Verbose messages....On

Tracing Options

Quality: 9
Bounding boxes.....On Bounding threshold: 25
Light Buffer.....On Vista Buffer.....On
Antialiasing.....Off
Radiosity.....Off

Animation Options

Clock value.... 0.000 (Animation off)

PVM Options

Block Width.... 32 Block Height... 32
PVM Tasks..... 2
PVM Nice..... 5
PVM Arch.....
PVM Slave..... /usr/bin/pvmpov
PVM WorkingDir. /behome/hkaya

Redirecting Options

All Streams to console.....On
Debug Stream to console.....On
Fatal Stream to console.....On
Render Stream to console.....On
Statistics Stream to console....On
Warning Stream to console.....On

Starting frame 0...

Slave 1 at istanbul successfully started.

Finishing frame 0...rtw. 240

Waiting for remaining slave stats.

PVM Task Distribution Statistics:

host name [done] [late] host name [done] [late]
istanbul [52.67%] [0.17%] ankara [47.33%] [0.00%]

POV-Ray statistics for finished frames:

skyvase.pov Statistics, Resolution 320 x 240

Pixels: 85248 Samples: 85248 Smpls/Pxl: 1.00
Rays: 350023 Saved: 2313 Max Level: 0/5

Ray->Shape Intersection	Tests	Succeeded	Percentage
CSG Intersection	1125202	96435	8.57
CSG Union	143265	31848	22.23
Plane	6751212	3644222	53.98
Quadric	1125202	458986	40.79
Sphere	1125202	151899	13.50

```

Calls to Noise:          364259   Calls to DNoise:  645434
-----
Shadow Ray Tests:       1232769   Succeeded:        28022
Reflected Rays:        264775
-----
Smallest Alloc:         15 bytes   Largest:          40004
Peak memory used:       331698 bytes
-----
Time For Trace:    0 hours 0 minutes 8.0 seconds (8 seconds)
Total Time:       0 hours 0 minutes 8.0 seconds (8 seconds)

```

Gördüğünüz gibi birçok mesaj ekrana yazıldı. Bu program gerçekten paralel çalıştı mı diyorsanız çıktıların içinde yer alan `istanbul` ve `ankara` isimlerine dikkat edin. Bütün işlemin ne kadarının hangi makinede yapıldığı görebilirsiniz. Dikkat edilmesi gereken bir nokta da `pvm` komutunu tam yolu ile yani `/usr/bin/pvm` şeklinde çağırmanız. Öteki türlü köle makineler `pvm` komutunu bulamıyorlar.

Paralel ortamda çalışmanın getirisini anlayabilmek için uzun vakit alan uygulamalara bakmanız gerekli. Çünkü zaten kısa süren uygulamaları paralel ortamlarda koşmanın büyük bir getirisi olmayacaktır. Ayrıca makineler arasındaki haberleşme süresi işlem zamanına görece büyük olacaktır. Bu yüzden gerçekten uzun süren uygulamalarda yararlı olacaktır. Mesela yukarıdaki örneği daha büyük ölçeklerde işlememiz gerekseydi süre çok daha uzun olacaktı.

Süre karşılaştırma işlerini Uygulamalar bölümüne bırakıp burayı geçiyoruz.

4 MPI ve POV-Ray

POV-Ray'in paralelleştirmesi için akla gelen diğer bir seçenek ise MPI. POV-Ray'in MPI versiyonu Leon Verrall tarafından PVM-POV-Ray yazılımından yararlanılarak geliştirilmiştir.

4.1 Kurulum

Ne yazık ki ben MPI-POV-Ray için hazırlanmış bir RPM paketi bulamadım. Bunu yazılımın çok yaygın olmamasına bağlıyorum. Kurulum için POV-Ray'in kaynak kodlarına gereksinimiz var. Gerekli dosyaları <http://www.povray.org> adresinden sağlayabilirsiniz. Ayrıca MPI yazılımının kurulu olması gerekiyor. MPI-POV-Ray'i ise yazılımın ana sayfasından indirebilirsiniz². Bu dosyalar mevcut POV-Ray kodlarına bir yama niteliği taşıyor. Yamayı POV-Ray kodlarının olduğu dizinde uygulamamız gerekli.

```
# gzip -dc mpi-povray-1.0.patch.gz | patch -p1
```

² <http://www.verrall.demon.co.uk/mpipov/>

Bu işlemin ardında `mpi-unix` adlı bir dizin oluşacaktır. Bu dizine gidip `make` yazmanız yeterli olacaktır. Birçok derleme işleminden sonra `mpi-x-povray` çalıştırılabilir dosyası oluşacak. Bu dosyayı herhangi bir yere taşıyıp çalıştırabilirsiniz. Ayrıntılı bilgiyi yukarıda belirttiğimiz `sanaldoku` yöresinde bulabilirsiniz.

4.2 Ayarlar

Herhangi özel bir ayarlama gerekmiyor. `mpirun` komutu ile çalıştırılması yeterli. Ancak dikkat çekici bir özellik `mpi-x-povray` komutunun en az iki işlemci belirtildiği zaman çalışabilmesi. İşlemcilerden bir tanesi resimleri parçalamak, diğer makinelere dağıtmak ve geri alıp resmin son halini oluşturmak için kullanılıyor. Dolayısıyla iki işlemciyle yapılan bir resim işleme tek makina ile yapılacak işlemden bir miktar fazla çıkıyor. Ancak iki makinadan sonra hızlanmalar elde ediliyor.

`MPI-POVRAY` animasyonları işlerken çerveleri işlemcilere dağıtır. Tabii çerçeve sayısı işlemci sayısından fazla ise. Bu açıdan `PVM-POVRAY`'dan başarılı olduğunu söyleyebiliriz. Çünkü `PVM-POVRAY` animasyon sözkonusu ise problemler yaşıyor. Mesela çerçveleri üstüste bindiriyor.

4.3 Örnek Uygulama

`PVM` için yaptığımız uygulamayı bu sefer `MPI` için tekrarlayabiliriz.

```
# mpirun -machinefile ~/.rhosts -np 2 mpi-x-povray \
+I Skyvase.pov
```

Yukarıdaki komut ile `/.rhosts` dosyasının içindeki ilk iki makinaı (`-np 2`) paralel işlem için kullanmış olduk. Bu komutun sorunsuz çalışabilmesi için ilgili makinalara `rsh` komutu ile şifresiz bağlanabilmeniz gerekiyor.

5 Uygulamalar

Bu bölümde `POVRAY` camiası tarafından test amaçlı kullanılan `skyvase.pov` adlı uygulamayı kullandık. Bu uygulama Şekil 8'de görülen resim dosyasını oluşturuyor.

İşlem sürelerini ise Unix işletim sisteminde bulunan `time` komutu ile belirledik. Bu komutun kullanımına bir örnek Şekil 9'da verilmiştir. Gördüğünüz gibi `povray` komutu normal bir şekilde çalıştıktan sonra `time` komutu üç satırdan oluşan bir çıktı üretti. "Real Time", komutun verildiği andan işlemin bitiş anına kadar geçen süreyi verir. Kronometre tutmaya oldukça benziyor. Eğer bilgisayar üstünde yük oluşturan başka ciddi bir program çalışmıyor ise ve anormal



Şekil 8. Skyvase.pov adlı dosyanın işlendikten sonraki görüntüsü.

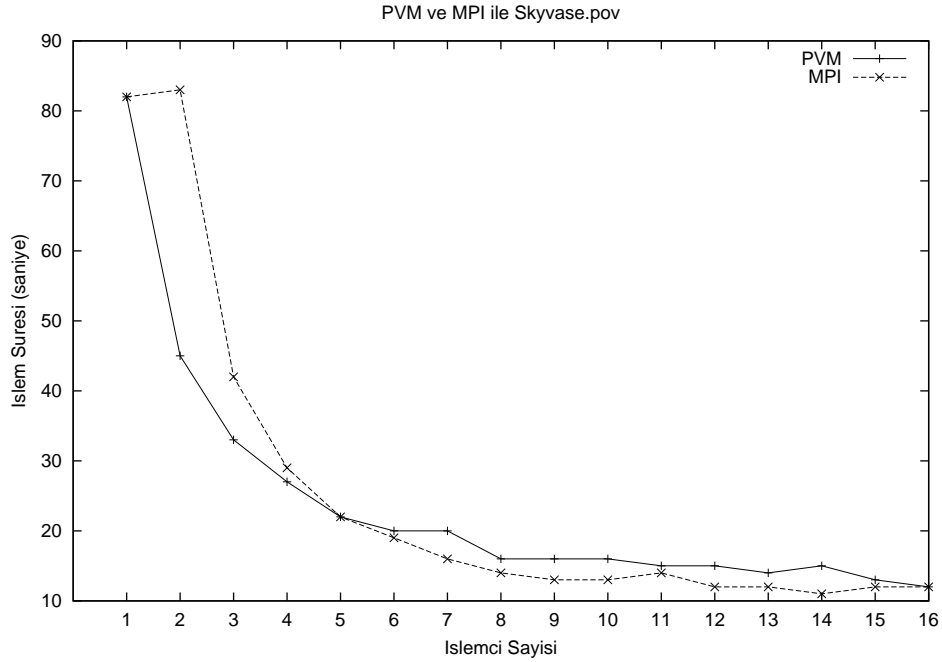
```
# time povray +I skyvase.pov
----> Povray mesajlarını es geçiyorum.
real    0m12.771s
user    0m12.120s
sys     0m0.080s
```

Şekil 9. time komutunun kullanılışı

bir durum yoksa “Real Time” ile “User Time” birbirine yakın çıkmalı. Bilgisayarlar üzerinde süreleri tutarken sadece biz çalıştırdığımızdan, yalnızca “Real Time” seçeneğini esas aldık.

PVM-POVRAY ve MPI-POVRAY için elde ettiğimiz sonuçlar Tablo 10’da gösterilmiştir.

Resim işleme sürelerinde elde edilen kazanç çok açık bir şekilde görülüyor. Normalde tek bilgisayarda 90 saniye süren bir işi 16 bilgisayar kullanarak 12 saniyede yaptık. Ancak grafikten görülebileceği gibi bilgisayar sayısı artırıldıkça hızlanma oranı düşüyor. Hatta 8,9 ve 10 bilgisayar arasında hız farkı olmadığı görülüyor. Bunun sebebi bilgisayarlar arasındaki haberleşmenin resim işleme zamanına baskın gelmesidir. Daha büyük çaplı problemler için haberleşme süresi önemsiz olacaktır.



Şekil 10. Farklı sayıda bilgisayar sayıları için işlem süreleri. Örnek `Skyvase.pov` üzerinde yapılmıştır.

İkinci örnek ise internet üzerinde 1996 yılından itibaren iki ayda bir gerçekleştirilen “Internet Ray Tracing Competition” adlı yarışmadan alındı. 1999 yılının Mart-Nisan döneminde birinci seçilen çalışma Şekil 11’de görüntülenmiştir. Şekil 11’deki resmin işlenmesi için harcanan süre Şekil 12’de gösterilmiştir.

6 Sonsöz ve Sonuç

Bu çalışmada çok popüler olan `POVRAY` adlı görselleştirme yazılımının bir küme üzerinde nasıl kullanılabileceğini göstermeye çalıştık. Bizim kullandığımız küme normal şartlar altında öğrenci bilgisayar laboratuvarı olarak kullanıldığı için makinaların üstünde her türlü donanım ve yazılım mevcuttu. Her ne kadar biz uygulamaları yaparken laboratuvar kullanılmadıysa da makinalarda yüklü olan programların ve donanımın az da olsa performansı kötü yönde etkilediğini düşünüyoruz.

Son zamanlarda geliştirilen yeni bir kümeleme tekniğinin bizim yaptığımız örneklemeler için çok daha iyi sonuçlar vereceğini düşünüyoruz. Yeni tekniğe göre ana makina hariç bütün bilgisayarlarda sadece bir çekirdek çalışıyor. Makinaların sabit diskleri, ekran ve ses kartları, klavye ve fare gibi aksesuarları kullanılmıyor. Sadece anakart, işlemci ve bellek kullanılıyor. Bu yapının en büyük

avantajı bilgisayarların işlemci gücünün en yüksek derecede kullanılması. Ayrıca donanımdan çok büyük oranda tasarruf sağlanıyor.

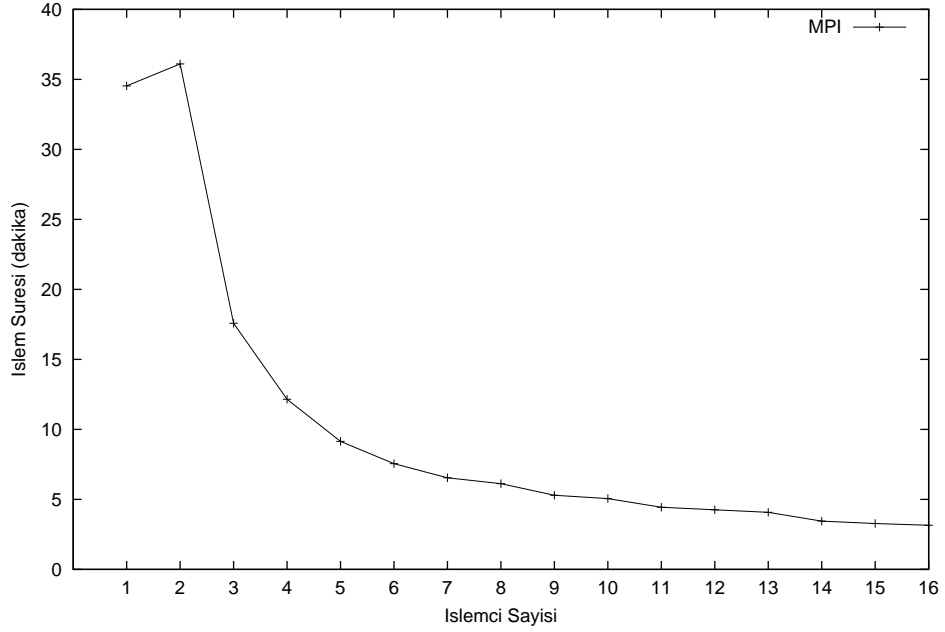
PVM-POVRAY kümeleme teknolojilerinin kazançlarını göstermesi açısından çok güzel bir örnek. Çok basit birkaç adımda onlarca bilgisayar paralel olarak çalıştırılabilir ve çok ucuz bir maliyet ile binlerce dolarlık süperbilgisayarlar ile ancak yapılabilecek uygulamalar gerçekleştirilebilir. Kümeleme teknolojilerindeki gelişmeleri takip ederek bu çalışmada elde edilen sonuçlar iyileştirilebilir.

Kaynaklar

1. ABBAS AYHAN KANMAZ, *Paralel Hesaplama Kütüphaneleri (MPI, PVM) ve Linux Ağında Çalıştırılmaları* <http://www.be.itu.edu.tr/kaynak/kaynak/sy/>
2. *Persistence of Vision TM Ray-Tracer POV-Ray TM Version 3.1g User's Documentation*, May 1999.
3. *PVMPOV HOWTO*, <http://pvmpov.sourceforge.net/PVMPOV-HOWTO.html>, August 22th 2002



Şekil 11. 1937 yılında Hindenburg adlı zeplinin patlamasını ve ardından yanmasını canlandıran POV-Ray çalışması.



Şekil 12. Farklı sayıda bilgisayar sayıları için işlem süreleri. Örnek Şekil 11'de gösterilen resim üzerinde yapılmıştır.