**MPI ile Paralel Programlama - 1**

Paralel programlamayı konuştuk tartıştık uygulamaları için gerekli platformları konuştuk şimdi bir örnek inceleyeceğiz.

Paralel programlamanın az kullanılmasının sebebi paralel makinelerin pahalı olması kontrolünün zor olması ve en önemlisi de genel olarak ihtiyacımızın olmamasıdır.

Peki paralel hesaplamalar nerede kullanılmaktadır? Araştırma laboratuarları, üniversiteler, ileri teknoloji kullanan şirketler limitlerinin zorlandığı durumlarda paralel makinelere ihtiyaç duymaktadırlar. Örnek olarak Koç Üniversitesinde “moleküler simülasyonlar”, “quantum computing”, ”computational electromagnetics”, ”bioinformatics” ve “computational flow simulation/modelling” gibi değişik araştırma projeleri üzerinde uğraşılmaktadır. Bu paralel hesaplamalar için de MPI adı verilen Message Passing Interface adlı paralel işlem kütüphanelerini (Library) kullanmaktayız. Bunun dışında değişik kütüphaneler de vardır.

**Donanım / Yazılım**  
  
Donanım olarak en basit makineleri birbirine bağlayarak bile bir cluster elde edebilirsiniz.

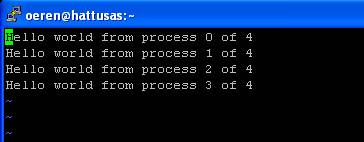
Bunun için bilgisayarınıza gerekli MPI (Message Passing Interface – Paralel kütüphaneler başka standartlar da var ben MPI kullanıyorum) dosyalarını ve bazı programları kurmanız gerekmektedir.

Bunlardan bir tanesi openPBS (Open Portable Batch System ) iş akışını düzenlemektedir ve OSCAR adlı ( MPI,SSH,SSL,MPICH,... gibi gerekli kütüphaneleri, programları bize sağlamaktadır ) programdır. Her ne kadar işletim sistemi olarak unix/linux tabanlı işletim sistemleri desteklense de windows tabanlı işletim sistemleri için de bu yazılımların versiyonları vardır.

Donanım olarak bütün işlemciler desteklenmektedir (Windows işletim sistemlerinde bu destek yoktur). Ayrıca ULAKBIM den (128 node’ luk güzel bir clusterları var) account almanız mümkün, tabi akademik amaçlar için kullanacağınıza ikna etmeniz gerekecektir.   
  
**MPI nedir?**  
  
Kısaca bahsettik ama MPI nedir? Nasıl çalışır? Başka makinelere/cpu lara nasıl sonuç yollar... Biraz bunlardan bahsedelim. Öncelikle yazdığınız her paralel kodun başına **“mpi.h”** headerini eklemeniz gerekmektedir. Bu header dışında kullanmanız gereken başka header yoktur. MPI fonksiyonları makineler arası iletişimi, işlem önceliklerini, yük ayarını vs düzenlemektedir.Format olarak MPI fonksiyonları MPI\_Xxxxx(parameter, ... ) formatını kullanmaktadır. MPI fonksiyonlarını size kısaca tanıtırsak,  
  
***MPI\_Init(&argc,&argv)***  
  
MPI\_Init fonksiyonunu bütün kodlarınıza eklemeniz gerekmektedir. MPI\_Init fonksiyonuna komut satırından da parametre verebilirsiniz ama mecbur değilsiniz. MPI\_Init size bir değer dönmektedir bu “MPI\_SUCCESS” olursa kod geri kalan MPI\_X fonksiyonlarını kullanabilecektir, eğer bu değer dönmezse MPI\_X fonksiyonlarını kullanamazsınız.  
  
***MPI\_Comm\_size(comm,&size)***  
  
Bu fonksiyonumuz proses sayısını bize vermektedir. Genellikle “comm” kısmına MPI\_COMM\_WORLD üst fonksiyonu yazılmaktadır.  
  
  
  
***MPI\_Comm\_rank(comm,&rank)***  
  
Bu MPI fonksiyonu çağıran prosesin sırasını vermektedir. Başlangıçta bütün prosesler 0 veya -1 değerlerini almaktadır. Daha sonra MPI tarafından sıraya sokulunca bütün prosesler numaralandırılmaktadır. Aşağıda ki örneğimizde göreceğimiz gibi makineler 0 dan başlayarak rank(sıra) alacaklardır.  
  
  
  
***MPI\_Abort******(comm,errorcode)***  
  
Adından da anlaşıldığı gibi bütün prosesleri durdurmaktadır. Istenmeyen durumlarda başvurulabilecek bir fonksiyondur.  
  
***MPI\_Get\_processor\_name(&name,&resultlength)***  
  
İşlemcinin adını ve adın uzunluğunu dönmektedir.  
  
***MPI\_Initialized(&flag)***  
  
MPI\_Init fonksiyonun çağırılıp çağırılmadığını kontrol etmektedir ve çağırıldıysa “1” dönmektedir. MPI\_Init her proses tarafından sadece bir kere çağırılması gerektiği için MPI\_Initialized fonksiyonu olası çakışmaları önlemektedir.  
  
  
  
***MPI\_Wtime( )***  
  
Double deger olarak paralel kodun çalışmaya başlamasından sonra geçen zamanı vermektedir.  
  
  
  
***MPI\_Finalize( )***  
  
Bütün işlemlerimizden sonra MPI\_Finalize diyerek işlemlerimizi sonlandırıyoruz.  
  
Yukarıda gördüğünüz fonksiyonlar genel MPI fonksiyonlarıdır ve sadece kodunuzu yönetmek amacıyla kullanılmaktadır. Bunların dışında 100 e yakın daha fonksiyon vardır.

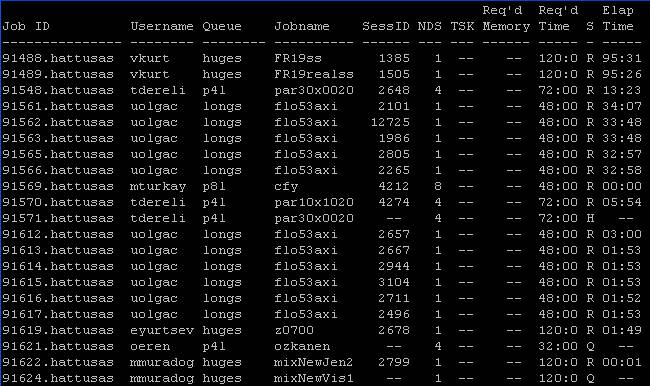
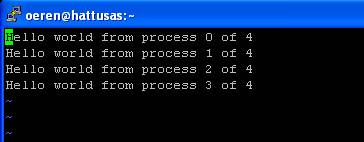
**Hello World 0 Hello World 1 Hello World 2  ....**  
  
Klasikleşmiş “merhaba dünya” örneğimizi bu sefer paralel bilgisayarlarda uygulayacağız. Her bilgisayardan selamlama mesajı göndereceğiz. Bunun için yukarıda açıkladığımız fonksiyonları kullanacağız.

|  |
| --- |
| #include <stdio.h> #include "mpi.h" int main( argc, argv ) int  argc; char \*\*argv; {     int rank, size;     MPI\_Init( &argc, &argv );     MPI\_Comm\_size( MPI\_COMM\_WORLD, &size );     MPI\_Comm\_rank( MPI\_COMM\_WORLD, &rank );     // beklemek istemediğim için fprintf komutu ile myout.dat dosyasına yazdırıyorum     printf( “Hello world from process %d of %d \n", rank,size);     MPI\_Finalize();     return 0; } |

Compile edip çalıştırınca cevap olarak aşağıdaki ekranı görmekteyiz.  
  
  
  
Haklı olarak “sadece bir tane proses gözüküyor” diyeceksiniz. Hatamız burada paralel işlem yapacak bir kodu normal bir C kodunu çalıştırdığımız gibi çalıştırmamızdır. Bunun için paralel makineye iş yüklemek gerekmektedir (işin bu kısmı genelde hiç söylenmez). Bunun için PBS yazılımının manuelinden nasıl komut aldığına bakıyoruz (scriptteki #PBS bu bahsettiğimiz komutlardır). Örnek olarak yüklediğimiz işe isim vermek istersek #PBS –N benim işim diyebiliyoruz. Oluşacak output ve error bilgileri için log dosyaları da oluşturabiliriz. Daha sonra bu yazdığımız 10-15 komutu bir defada çalıştırmak için bir “job script” yazıyoruz. Yukarıda bahsedilen PBS adlı program sizin verdiğiniz komutları değerlendirerek bilgisayarların iş yoğunluğuna göre verilen işleri sıraya sokmaktadır ve ona göre cpu, hard disk, memory tahsisi yapmaktadır.

Aşağıda gördüğünüz ### comment yani yorumlar, geri kalan kısımlar execute edilecek komutlardır. Bunu dışında bir yöntem daha var yöntem sistem admini tarafından görülürse hiç hoş karşılanmayan bir durumdur (ki genelde serverlerde sizin ana makineye PBS aracılığı ile iş yüklemeniz dışında kolay kolay komut kullanamazsınız).

|  |
| --- |
| #!/bin/sh ### iş ismi “ozkanen” ismi verdim #PBS -N ozkanen ### Output files ### olusacak output ve olusacak error loglarını  buraya koyabiliriz #PBS -o ozkanen.out #PBS -e ozkanen.err ### Buraya dikkat edin!!! ### 4 gördüğünüz kısım 4 makine kullanacağım demek ### 1 gördüğünüz kısım 1 makinede 1 işlemci kullanacağım demek (hattusas da her node da bir işlemci var) #PBS -l nodes=4:ppn=1 ### programın ne kadar çalışacağını söylemektedir #PBS -l cput=32:00:00 #PBS -l walltime=32:00:00 ### Queue name ### clusteri paralel olarak kullanacagim icin asagidaki kodu yazıyorum ### clusterları isterseniz tek tek de kullanabilirsiniz #PBS -q parallel #PBS -m abe ### cevap cıkınca bana mail at demek oluyor aşağıda #PBS -M oeren@ku.edu.tr ### Script Commands ### aşağı kısımlar o kadar önemli değil sadece size işlerin çalıştığını ve nerede çalıştığını size söylüyor cd $PBS\_O\_WORKDIR echo "Running test" echo group main > machines cat $PBS\_NODEFILE | awk {print "host " $1} >> machines echo current directory is pwd echo Running test ### aşağıda ./hello adını verdiğimiz dosyayı çalıştırıyoruz ### istersek cevapları output dosyamıza yazdırabiliriz ./hello   out ### işimiz bittiği zaman bize bittiğini haber vermektedir echo "job complete" exit 0 |

Bu job scripti yazdıktan sonra oerentest.job ismi ile kaydediyorum. Çalıştırmak için de qsub oerentest.job diyorum.  
  
  
  
“a” komutu yazdığım zaman ekranda 91621 kodu ile işimin sisteme girildiğini görüyorum. Status anlamına gelen “S” sütunundan yüklediğim işimin Q (yani sırada olduğunu) durumunda olduğunu görüyorum. Kodum saniyelik bir run yapıp bana cevap verecek ama yukarıda gördüğünüz gibi sıra olduğu için PBS sıraya sokuyor (32 node da olsa her zaman fazla iş var). Eğer dikkatlice bakarsanız NDS de “4” yazdığını görürsünüz bu “4” tane makine istiyorum anlamındadır, ayrıca ben dahil kimse memory den kısıtlama yapmadığı için Req memory kısmı boş gözükmektedir. Ayrıca Req Time sütununa  bakarsanız 32 saatlik bir iş verdiğimi ve Elap Time sütununa bakarsanız hiç çalışmadığını göreceksiniz. Bayağı uzun sürdüğü için ben bu işimi bir output dosyama yazdırıp sonraya bırakıyorum ve filmimi seyrediyorum. Aradan bayağı bir zaman geçtikten sonra sıradakilerin işi bitip benim işim de çalışınca sabırsızca beklediğim “mydat.out” dosyam oluşuyor. VI editörü ile mydat.out dosyasına bakarsak :  
  
  
  
Beklediğimiz sonucu görüyoruz. Şimdi kodumuza tekrar dönelim. Temel bir C kullanıcısına yabancı gelen aşağıdaki satırlara bakalım:

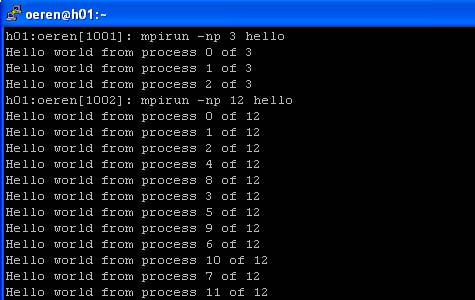
|  |
| --- |
| // verilen proseslerin toplam sayısını bize size olarak dönmektedir MPI\_Comm\_size( MPI\_COMM\_WORLD, &size ); // prosesin hangi bilgisayarda çalıştığı rank olarak dönmektedir MPI\_Comm\_rank( MPI\_COMM\_WORLD, &rank ); // sonuçları print ediyoruz printf( “Hello world from process %d of %d \n", rank,size); |

Yabancı fonksiyonlardan ilki olan Comm\_size fonksiyonunu çağırıyoruz.

|  |
| --- |
| MPI\_Comm\_size( MPI\_COMM\_WORLD, &size ); |

Yukarıda da bahsettiğim gibi MPI\_Comm\_size fonksiyonu bize paralel makineye yüklediğimiz işin kaç bilgisayar tarafından çalıştırılacağını “size”  değeri ile vermektedir. MPI\_COMM\_WORLD değişkeni ise çalıştığımız environmenti (çevre) vermekteyiz. Makinemiz bunu default çalışma parametresi almaktadır. Şimdi elimizde size değeri 4 makineye iş yüklediğimiz için 4 olarak bize dönmektedir. Sırada rank fonksiyonumuz var!

|  |
| --- |
| MPI\_Comm\_rank( MPI\_COMM\_WORLD, &rank ); |

Rank fonksiyonu bize proseslerin MPI tarafından sıraya sokulduktan sonra hangi sırada olduğunu rank ile bize dönmektedir. En başta ki MPI\_COMM\_WORLD değerini de size fonksiyonunda olduğu gibi default olarak giriyoruz. MPI tarafından sıraya biz herhangi bir tanımlama yapmadığımız için ( prosesleri sıraya sokmak için ve bunların arasında etkileşimlere göre sıra önceliği tanımlayabileceğimiz fonksiyonlar vardır. Bunlar arasında MPI\_Send, MPI\_Recv, MPI\_Ssend, MPI\_Bsend, MPI\_Buffer\_attach, MPI\_Rsend fonksiyonlarını sayabilirim bunları gelecek yazımda detaylı olarak anlatacağım ve örnek kodla açıklayacağım. Örnek olarak MPI\_Recv fonksiyonu bir bilgisayardan mesaj aldığı zaman o mesajı gerçekleştirene kadar beklemektedir (eğer bunu kullanmazsak bir makinenin oluşturmadığı verilere ulaşmaya çalışmak hata döndürürdü).  
  
**Kolay Yötem : Job Script yazmadan çalıştırmak**  
  
Başlangıçta size bu job scripti yazmadan daha kolay bir şekilde bu programı çalıştırabileceğimiz söylenmişti (yine açıklayalım çoğu clusterlar buna izin vermemektedir ,PBS ve benzeri uygulamalar üzerinden programlar çalıştırılmaktadır). Şimdi gelelim kolay yönteme! Komut satırına “mpirun –np 3 hello” yazıyoruz. Bunun anlamı şudur “Bana 3 tane makine kullanarak hello uygulamasını çalıştır”. Bu satırı yazdıktan sonra MPI aktif hale geliyor biz iş sırasını düzenlemediğimiz için MPI hangi prosesten önce cevap gelirse onları sıraya sokarak ekrana cevapları döndürüyor. Komutta eğer 3 yerine 12 yazarsak 12 node kullanarak çalıştırıyoruz. Gördüğünüz gibi her defasında ayrı bir sıra oluşuyor hangi işlemci daha hızlı cevabı gönderirse onun prosesi daha önce gözüküyor.  
  


MS MPI kurulum ve ilk basit örnek:

<https://www.youtube.com/watch?v=L-xJreZ55aU&list=PLuuNv043RqhHe7zghCIk1p2E2joHAEOdq&ab_channel=MahmoudEssam>

MPI Fonksiyonları:

https://learn.microsoft.com/en-us/message-passing-interface/mpi-functions

Kaynaklar:

Özkan Eren (Csharpnedir.com)