

MPI ile Paralel Programlamanın Temelleri

Devkan Kaleci¹, Ali Şahin², Olgun Adem Kaya¹

¹ İnönü Üniversitesi, Eğitim Fakültesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü

² İnönü Üniversitesi, Fen Edebiyat Fakültesi Fizik Bölümü

devkan@inonu.edu.tr, asahin@inonu.edu.tr, oakaya@inonu.edu.tr

Özet: Günümüzde artan talep ve teknolojik gelişmelere bağlı olarak bilgisayar teknolojilerinde de önemli gelişmeler olmaktadır. Örneğin işlemci hızları GHz mertebesine ulaşmıştır. Buna rağmen bazı özel uygulamada dahi bu hızlar yetersiz kalmaktadır. Bu nedenle süper bilgisayarlara olan ihtiyaç ortaya çıkmıştır. Süper bilgisayarlar, birden fazla bilgisayarın bir ağ üzerinde birbirleri ile iletişim halinde buldukları sistemlerdir. Paralel programlama ise, bu bilgisayarların birbirleri ile haberleşmelerini sağlayan algoritmaların tümüdür. Bu çalışmada Linux işletim sistemi tabanlı paralel programlama yöntemlerinden bir olan MPI'ın (Message Passing Interface) temelleri ve MPI ile hazırlanan paralel programlama örnekleri sunulmuştur.

Abstract: Nowadays, there has been a significant improvement in the computer technology with regard to increasing demand and technological development. For example, processing speed reached GHz level. Despite this high level processing speed, it is still not adequate for some particular operations. Therefore, needs for super computers have been emerged. Super computer concept is a system in which more than one computer communicates with each other. Parallel programming, on the other hand, is a set of algorithms which enables computers to communicate with each other. In this paper, fundamentals of Message Passing Interface (MPI) methods for parallel programming based on Linux operating system and parallel programming examples with the application of MPI were presented.

Anahtar Kelimeler: Paralel Programming, MPI, Super Computer.

1. Giriş

İlk süper bilgisayar çalışmalarına 1990'lı yılların başlarında başlanmıştır ve bu ilk çalışmalar İngiliz edebiyat kahramanı olan Beowulf adı altında toplanmıştır [1]. Beowulf İngiliz edebiyatında tüm büyük zorlukları aşan bir şövalyedir. Süper bilgisayarlar da aynı düşünce doğrultusunda tüm büyük zorlukları aşabilmeleri amacıyla geliştirilmişlerdir. Günümüzde farklı amaçlar için binlerce bilgisayardan oluşan süper bilgisayarlar kurulmuştur [2].

Bir süper bilgisayarın temel amacı, komplike ve uzun süre alan işlemleri kısa sürede çözülmesini mümkün kılmasıdır. Bu nedenle yapılan işlemlerin en kısa zamanda çözümü için süper bilgisayarların bir algoritma içinde programlanması gerekir.

Paralel programlama bir ağ üzerinde birden fazla bilgisayarın haberleşmelerini temel alacak şekilde geliştirilmiştir. Bu tür sistemler yardımı ile mevcut işler parçalara bölünerek ağ üzerindeki her bir bilgisayar paylaşılabilir. Böylece her bilgisayar düşen işlem sayısı azalmış olur ve işlem süresi kısalmıştır [3-5].

MPI, temel olarak süper bilgisayar üzerinde yürütülen bir işlemin tüm bilgisayarlara parçalanarak ayrı ayrı olarak dağıtılması için kullanılan mesaj gönderme arayüzüdür [6,7]. MPI, C'de bir fonksiyon, Fortran'da bir alt rutindir ve işlemciler arasında haberleşmeyi sağlamak amacıyla kullanılır. Tüm işlem süreci boyunca işlemciler aralarında haberleşirler. Bu nedenle, MPI kütüphanesi bu haberleşmeleri sağlayacak fonksiyon ve alt rutinleri de içermektedir.

İlk MPI çalışmalarına Kasım 1992’de resmi olarak ilk sunum yapılması ile başlanmıştır. MPI standardı yaklaşık elli kişilik bir grup tarafından birçok farklı organizasyonda tartışılarak geliştirilmiştir. 1993 yılında süper bilgisayarlar için geliştirilen ilk programların taslakları sunulmuş ve bir yıl sonra ilk MPI standardı olan MPI-1.0 bir çok farklı platformda denerek tamamlanmıştır. Bu standart, Fortran 77 ve C’de çağrılacak alt rutin ve fonksiyon isimlerini ve sonuçlarını içermektedir. Nisan 1995’de MPI-2 çalışmalarına başlamıştır. Haziran 1995’de MPI-1.1, Temmuz 1997’de MPI-1.2 ve 1997’nin sonlarında ise, MPI-2 sunulmuştur. MPI-2, MPI-1.1’den farklı olarak paralel I/O işlemleri için paketleri, Fortran 90 dosyalarını ve dinamik işlem yöneticisi gibi ek özellikleri içermektedir.

2. Genel MPI Program Yapısı

Bir C programı düşünüldüğünde MPI fonksiyonlarının yer aldığı bir programın yapısı aşağıdaki gibidir.

1. Header dosyalarının eklenmesi
2. MPI veri türlerinin belirlenmesi
3. MPI’ın başlatılması
4. MPI haberleşmelerinin yapılması
5. Program içinde yer alan gerekli hesaplamaların yapılması
6. MPI’ın kapatılması
7. Programın sonlandırılması

MPI fonksiyonlarının kullanılabilmesi için C kodlarına başına `#include <mpi.h>` yada `#include "mpi.h"` şeklinde MPI header eklenmelidir.

Başlangıç MPI fonksiyonu, `MPI_INIT(&argc,&argv)` fonksiyonudur ve bu fonksiyon MPI programın çalışıp çalışmadığını kontrol eder.

Prosesler arası haberleşmeler, `MPI_Comm_size (comm,&size)` ve `MPI_Comm_rank(comm,&rank)` fonksiyonları ile sağlanır.

Birinci fonksiyon, sistemde bulunan işlemci (node) sayısını verir. İkinci fonksiyon ise, işlemcilerin 0’dan N’e kadar sayısal olarak sıralanmasını sağlar. Bu fonksiyon kullanılmazsa, bütün işlemciler 0 ya da -1 değerini alır.

MPI’in sonlandırılması, `MPI_Finalize()` fonksiyonu ile sağlanır. Bu nedenle bu fonksiyon kodun en sonuna eklenmelidir.

3. MPI ile Haberleşme

MPI bir haberleşme protokolüdür ve MPI fonksiyonları temel olarak node’ların aralarında haberleşmelerini sağlar. MPI ile iki temel haberleşme şekli vardır. Birincisi, herhangi bir node’un bir mesajı göndermesi ve diğer node’un ya da node’ların bu mesajı alması şeklinde gerçekleşen haberleşme türüdür. İkincisi ise, node’lar arasında verilerin gönderilmesini ve alınmasını içeren haberleşme türüdür.

3.1. Point-to-Point Haberleşme

Point-to-point haberleşme bir node’un herhangi bir mesajı diğer node ya da node’lara gönderilmesi ve bu mesajın tüm node ya da node’lar tarafından alınması sürecini kapsar. Point-to-point haberleşmesi, bir node’un bir mesajı gönderip diğer node’ların bu mesajı alması prensibine göre çalıştığından işlem süresince senkronizasyon yoktur. Birinci node başlangıç olarak mesaj gönderir. Daha sonra ikinci node mesajı alarak işlemi sürdürür. Yani point-to-point haberleşmesi mesaj gönderme ve alma şeklinde iki yönlü işler. Bu haberleşme boyunca her iki node’un olaya katılması gerekir. Aksi takdirde veri transferi gerçekleşemez.

MPI’da mesajlar, mesaj zarfı ve mesaj gövdesi olmak üzere iki kısımdan oluşur. MPI mesaj zarfı, üzerinde gideceği hedefin adresinin, dönüş adresinin, göndermek ve alınmak için gerekli diğer bilgilerin yer aldığı günlük hayatımızda kullandığımız mektup zarfına benzer. MPI mesaj zarfı, gönderilecek işlemciyi gösteren kaynak, alıcı işlemciyi gösteren

hedef, kaynak ve hedef işlemcilerle ait özel haberleşme özelliklerini içeren haberleştirici ve mesajı etiketlendirmek amacıyla kullanılan işaretçi olmak üzere toplam dört kısımdan oluşur. Mesaj gövdesi ise, mesaj verisi, mesaj veri tipi ve mesaj veri sayısı olmak üzere üç kısımdan oluşur.

Point-to-point haberleşmesinde, mesaj gönderilmesi “*send*” argümanı ile, alınması ise “*receive*” argümanı ile gerçekleştirilir.

SEND Parametresi

MPI_Send fonksiyonu, istenilen bir veriyi herhangi bir işlemciye göndermek amacıyla kullanılır. Bu fonksiyonla tek bir değişken gönderilebileceği gibi bir dizi ya da bir dizinin herhangi bir kısmı da gönderilebilmektedir.

Bir mesaj MPI_Send fonksiyonu kullanılarak gönderildiğinde, mesaj ilk olarak hafızaya alınır ve bloklama fonksiyonları kullanılmamış ise, senkronize olmaksızın hedef işlemciye gönderilir. Gönderilen mesaj mevcut hafızadan daha fazla yer kaplıyorsa, hafıza alanı kullanılamaz ve mesajı göndermekte olan işlemci yerli hafıza alanı sağlanana kadar mesajı bloklar.

MPI_SEND parametresinin yapısı aşağıdaki gibidir.

```
MPI_Send(&input_veri, input_veri_sayısı,  
         input_veri_tipi, hedef_node'un_rankı,  
         işaret_sayısı, MPI_COMM_WORLD)
```

RECIEVE Parametresi

MPI_Recv fonksiyonu, node'lar tarafından mesajın gönderildiğini ve bu mesajın kullanıma hazır olduğunu belirtmesi amacıyla kullanılır. Bu amaçla MPI_Recv fonksiyonu, MPI_Send fonksiyonundan farklı olarak status şeklinde ekstradan bir argüman içerir. Bir veri hedef node'a ulaştığında, mesajı işaretine, kaynağına ve alınan veri sayısına status argümanı yardımı ile ulaşılabilir. Eğer alınan mesaj gön-

derilen mesajdan fazla ise, mesaj gönderme alma sürecinde hata oluştuğu anlamına gelir. Yani her durumda alınan mesaj ile gönderilen mesaj miktarı aynı olmalıdır. Ayrıca dikkat edilmesi gereken diğer önemli bir nokta ise, gönderilen ve alınan mesaj veri tiplerinin aynı olmasıdır.

MPI_Recv parametresinin yapısı aşağıdaki gibidir.

```
MPI_Recv(&output_veri, output_veri_sayısı,  
         output_veri_tipi, kaynak_node'un_rankı,  
         işaret_sayısı, MPI_COMM_WORLD &status)
```

3.2. Kolektif Haberleşme

Kolektif haberleşme, bir grup node arasında verilerin gönderilmesini ve alınmasını kapsar. Genel olarak düşünüldüğünde, kolektif haberleşme rutinleri kullanılmaksızın yapılacak tüm işlemler MPI_Send ve MPI_Recv fonksiyonları kullanılarak yapılabilir. Fakat kolektif haberleşme fonksiyonları, bu işlemlerin çok daha kısa bir şekilde yapılmasını sağlar. Ayrıca point-to-point haberleşmeden farklı olarak kolektif haberleşme fonksiyonları yardımıyla, bir node'un birden fazla node'a (one-to-several) ya da diğer node'ların herhangi bir node'a (several-to-one) veri transferi yapmasına olanak sağlar.

Kolektif haberleşmenin kullanılması ile program içinde yazılacak MPI kodlarında önemli ölçüde bir azalma olur. Bu ise, program yazılması sırasında yapılacak hatalarının azalması anlamına gelir. Ayrıca kolektif haberleşme rutinleri kullanılması ile node'lar arası yapılan haberleşmeler azaldığından çalıştırılan programın daha kısa sürede tamamlanması sağlanabilmektedir.

Barrier Senkronize Edici Fonksiyon:

Bu fonksiyon, node'lar arasındaki senkronizeliği sağlamak amacıyla kullanılır ve her bir node herhangi bir işlem yapmadan önce diğer node'ların belirlenen sürece kadar beklemesini sağlar.

MPI_Barrier fonksiyonunun yapısı aşağıdaki gibidir.

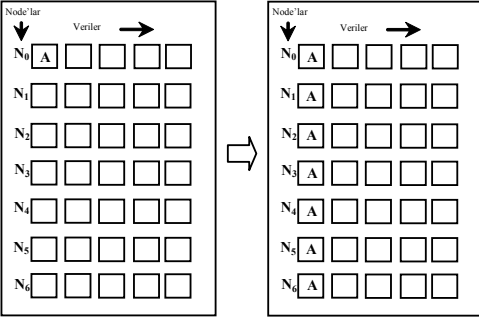
.....
.....

`MPI_Barrier (MPI_COMM_WORLD)`

.....
.....

Broadcast Fonksiyonu:

Şekil 1’de görüldüğü gibi broadcast fonksiyonu ile ana node’daki istenilen bir veri, haberleşme grubu içinde yer alan diğer tüm node’lara dağıtılır.



Şekil 1. Broadcast fonksiyon örneği.

MPI_Bcast fonksiyonunun yapısı aşağıdaki gibidir.

.....
.....

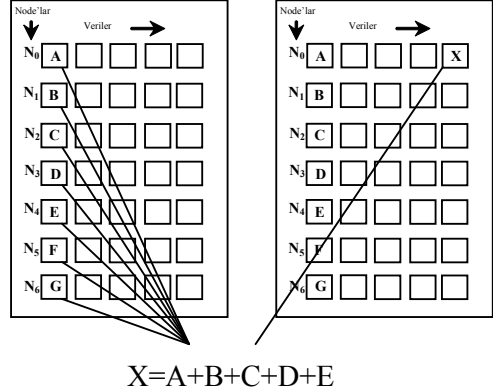
`MPI_Bcast (&input_veri,input_veri_sayısı,
input_veri_tipi,ana_node'un_rankı,
MPI_COMM_WORLD)`

.....
.....

Reduction Fonksiyonu

Reduction fonksiyonu, her bir node’dan istenilen verilerin alınarak temel toplam çarpma

ya da maksimum değeri bulma gibi işlemleri gerçekleştirerek ana node’da saklanmasını sağlar. Şekil 2’de verilen reduction fonksiyonu tüm node’lardan aldığı verileri toplayarak ana node’da aktarmaktadır.



Şekil 2. Reduction fonksiyon örneği.

Çizelge 1’de Reduction fonksiyonu içinde kullanılan operatörler verilmiştir. Bu operatörler yardımı ile node’lardan gelen veriler basit bir şekilde aşağıda verilen işlemlere tabi tutulması sağlanabilmektedir.

Operatör	Açıklaması
MPI_MAX	Maksimum Değer
MPI_MIN	Minimum Değer
MPI_SUM	Toplama işlemi
MPI_PROD	Çarpma işlemi
MPI_LAND	Mantıksal AND
MPI_LOR	Mantıksal OR

Çizelge 1. Reduction fonksiyonunda kullanılan bazı operatörler.

MPI_Reduce fonksiyonunun yapısı aşağıdaki gibidir.

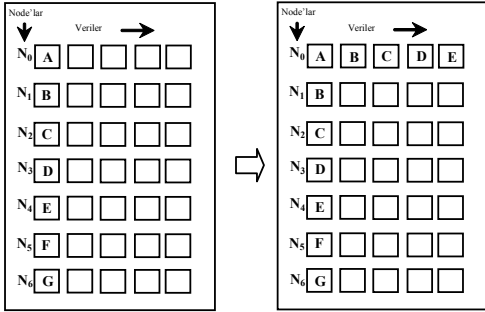
.....
.....

`MPI_Reduce (&input_veri,&output_veri,veri_sayısı,
veri_tipi,reduce_fonk,ana_node'un_rankı, MPI_COMM_WORLD)`

.....
.....

Gather Fonksiyonu

MPI_Gather fonksiyonu her bir node'daki verileri ana node'a kopyalanmasını sağlar. Şekil 3'de gather fonksiyon örneği verilmiştir. Dikkat edilecek olursa, diğer node'larda bulunan veriler ana node'daki verilere eklenmektedir.



Şekil 3. Gather fonksiyon örneği.

MPI_Gather fonksiyonunun kullanımı:

.....
.....

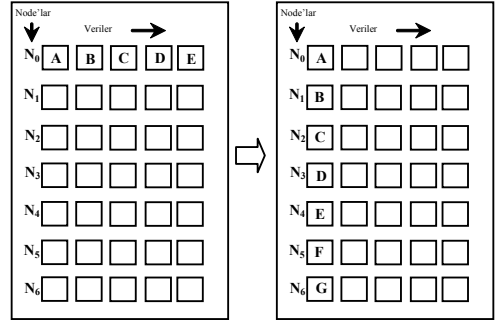
```
MPI_Gather(&input_veri, input_veri_
sayısı,
         input_veri_tipi, &output_
veri,
         output_veri_sayısı, output_
veri_tipi,
         ana_node'un_rankı, MPI_COMM_
WORLD)
```

.....
.....

Scatter Fonksiyonu

MPI_Scatter fonksiyonu, ana node'da bulunan dizi elemanlarının sırası ile diğer node'lara dağıtılması amacıyla kullanılır. Burada dağıtma işlemi, ana node'dan başlar ve node rankına göre sırası ile devam eder. Şekil 4'den de görül-

düğü gibi, bu fonksiyon yardımıyla sadece ana node'daki verilerin paylaşılması mümkündür.



Şekil 4. Scatter fonksiyon örneği.

MPI_Scatter fonksiyonunun kullanımı:

.....
.....

```
MPI_Scatter(&input_veri, input_veri_
sayısı,
           input_veri_tipi, &output_veri, output_
veri_sayısı,
           output_veri_tipi, gönderilecek_
node'un_rankı,
           MPI_COMM_WORLD)
```

.....
.....

5. Sonuç ve Tartışma

İlk bilgisayarlar 1950'li yıllarda yapılmaya başladığında, onlarca ton ağırlığında ve çok düşük işlem hızlarına sahiptiler. Bugün ise, her ofise ya da eve girebilecek kadar küçülmüş ve saniyede milyarlarca işlem yapabilecek kapasiteye ulaşmışlardır. Teknolojinin her geçen gün gelişmesi ile birlikte bilgisayarın hızlarında önemli artışlar olmaktadır. Fakat bu artışın nerede son bulacağı sorusu her zaman olacaktır. Süper bilgisayarlar da tam bu noktada ortaya çıkmaktadır. Bilgisayarlar ne kadar hızlandırılırsa hızlandırılırsın, birden fazla bilgisayarın aralarında haberleşmelerini sağlayan bir algoritma geliştirilerek en hızlı bilgisayardan dahi kat kat hızlı sistemler oluşturulabilir.

Süper bilgisayarlar bugün hızla gelişen bir çalışma alanı durumuna gelmiştir. Başta büyük bilgisayar ve yazılım şirketleri olmakla beraber, teknolojiye dayalı firmalar ve üniversiteler süper bilgisayar çalışmalarında yer almaktadırlar. IBM [8,9], SUN Microsystems [10], Cray [11] gibi dünya çapında büyük bilişim şirketleri süper bilgisayar sistemleri pazarındadır. Bu tip şirketlerin kendilerine ait işletim sistemleri vardır ve süper bilgisayarların çalışması için bu işletim sistemleri üzerinde çalışan özel yazılımlar geliştirmişlerdir. Açık kaynak kodlu ücretsiz olan Linux işletim sistemleri bu noktada önem kazanmaktadır. Bir Linux işletim sistemi ve MPI yardımı ile herhangi bir üniversitede ya da kurumda ücretsiz olarak süper bilgisayar kurulabilir ve paralel programlamanın kullanılabilirliği birçok işlem yapılabilir.

Kaynaklar

- [1] <http://www.beowulf.org>
- [2] <http://www.top500.org/>
- [3] Kaleci, D., Kaya O. A., Karakaplan, M., Şahin A., ‘The Linear Acoustic Field Calculation on 15 node Linux Cluster’, Balkan Physics Letters, 2008 Special Issue, Boğaziçi University Press, ISSN 1301-8329, 2008, p.215-219.
- [4] Uehara, H., Tamura M., Yokokawa M., ‘An MPI Benchmark Program Library and Its Application to the Earth Simulator’, Lecture Notes in Computer Science, vol.2327, January 2002,pp 351-356.
- [5] Stefano Cozzini, Axel Kohlmeyer, and Roger Rousseau, ‘Benchmark Analysis of 64-bit Servers for Linux Clusters for Application in Molecular Modeling and Atomistic Simulations’, Proceedings of the 7th LCI International Conference on Clusters, 2006.
- [6] <http://www-unix.mcs.anl.gov/mpi/>
- [7] <http://lam-mpi.org>
- [8] <http://www.lanl.gov/roadrunner/>
- [9] http://domino.research.ibm.com/comm/research_projects.nsf/pages/bluegene.index.html
- [10] <http://www.tacc.utexas.edu/resources/hpcsystems/#ranger>
- [11] <http://www.cray.com/Home.aspx>