

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**JAVA İÇİN NESNE TABANLI MESAJ AKTARIM
KÜTÜPHANESİ**

**YÜKSEK LİSANS TEZİ
Müh. Çağlar KUMANLI**

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Bilgisayar Mühendisliği

EYLÜL 2008

**JAVA İÇİN NESNE TABANLI MESAJ AKTARIM
KÜTÜPHANESİ**

**YÜKSEK LİSANS TEZİ
Müh. Çağlar KUMANLI
(504041543)**

**Tezin Enstitüye Verildiği Tarih : 15 Eylül 2008
Tezin Savunulduğu Tarih : 25 Eylül 2008**

**Tez Danışmanı : Prof. Dr. Nadia ERDOĞAN (İ.T.Ü.)
Diğer Jüri Üyeleri : Yrd. Doç. Dr. A. Şima ETANER UYAR (İ.T.Ü.)
Yrd. Doç. Dr. Yunus Emre SELÇUK (Y.T.Ü.)**

EYLÜL 2008

ÖN SÖZ

Tez çalışmam süresince göstermiş olduğu anlayış ve yardımlarından ötürü tez danışmanım Sayın Prof. Dr. Nadia Erdoğan'a ve eğitim hayatım süresince desteklerini hiçbir zaman esirgemeyen aileme teşekkürü bir borç bilirim.

Eylül 2008

Çağlar KUMANLI

İÇİNDEKİLER

KISALTMALAR	v
ŞEKİL LİSTESİ	vi
ÖZET	vii
SUMMARY	ix
1 GİRİŞ	1
2 PARALEL PROGRAMLAMA KAVRAMI	4
2.1 Paralel Programlamaya Duyulan İhtiyaç	4
2.2 Paralel Programlama Modelleri	5
2.2.1 Ortak Bellek	5
2.2.2 Mesaj Aktarımı	5
2.2.3 Veri Paylaşımı	6
2.2.4 Kanallar	7
2.3 Mesaj Aktarım Arayüzü	7
2.3.1 Temel Fonksiyonlar	8
2.3.2 Mesaj Yapısı	9
2.3.3 MPI Gerçekleştirimleri	10
2.3.4 Java İçin MPI Benzeri Gerçekleştirimler	11
2.3.4.1 mpiJava	12
2.3.4.2 CCJ	12
2.3.4.3 JOPI	13
2.3.5 Java İçin Mesaj Aktarım Arayüzü	14
2.3.5.1 Tarihçe	14
2.3.5.2 JMPI	14
3 JAVA İÇİN NESNE TABANLI MESAJ AKTARIM KÜTÜPHANESİ	16
3.1 Genel Özellikler	17
3.1.1 İstemci – Sunucu Mimarisi	17
3.1.2 Kaynak Yönetimi	19
3.1.3 İletişime Yardımcı Yapılar (Communicators)	19
3.2 Uygulama Geliştirme Arayüzü	20
3.2.1 Fonksiyonlar	21
3.2.1.1 Start	21
3.2.1.2 GetProcessID	21
3.2.1.3 GetCommSize	21
3.2.1.4 Send	21
3.2.1.5 Multicast	22
3.2.1.6 Broadcast	22
3.2.1.7 Receive	23
3.2.1.8 InformClient	23
3.2.1.9 Finish	24
3.2.2 Örnek Uygulamalar	24
3.2.2.1 Pi Sayısının Hesaplanması	24

3.2.2.2	Isı Denklemi Uygulaması	27
3.2.2.3	Sıralama Uygulaması	30
3.2.2.4	Matris Çarpımı	32
3.3	Uygulamaların Çalıştırılması	34
3.3.1	Konfigürasyon Dosyaları	36
3.3.2	Sunucu Seçimi	38
3.4	Araçlar	40
3.4.1	Konfigürasyon Dosyası Oluşturucu	40
3.4.2	Etmen Durumu Bildirici	42
3.5	Günlük Tutma (Logging) Sistemi	45
3.6	Sistem Gereksinimleri ve Kurulum	46
4	SONUÇ	49
	KAYNAKLAR	51
	ÖZ GEÇMİŞ	54

KISALTMALAR

CCJ	: Collective Communication in Java
HPF	: High Performance FORTRAN
IANA	: Internet Assigned Numbers Authority
IEEE	: Institute of Electrical and Electronics Engineers
IP	: Internet Protocol
ISO	: International Organization for Standardization
JDOM	: Java Document Object Model
JMPI	: Java Message Passing Interface
JNI	: Java Native Interface
JOPI	: Java Object Passing Interface
JRE	: Java Runtime Environment
LAM/MPI	: Local Area Multicomputer / Message Passing Interface
MPI	: Message Passing Interface
MPICH	: Message Passing Interface Chameleon
MPJ	: Message Passing Interface for Java
OpenMP	: Open Multi-Processing
POSIX	: Portable Operating System Interface
PPPJ	: Parallel Programming Platform for Java
RMI	: Remote Method Invocation
SPMD	: Single Program Multiple Data
XML	: Extensible Markup Language

ŞEKİL LİSTESİ

	<u>Sayfa No</u>
Şekil 2.1 : Mesaj kısımlarının fonksiyon imzaları üzerinde gösterimi.....	9
Şekil 3.1 : Örnek istemci-sunucu mimarisi.....	18
Şekil 3.2 : TriplePJAssistant sınıfının örneğinin oluşturulması.....	21
Şekil 3.3 : Send fonksiyonunun imzası	22
Şekil 3.4 : Multicast fonksiyonunun imzası.....	22
Şekil 3.5 : Broadcast fonksiyonunun imzası.....	23
Şekil 3.6 : Receive fonksiyonunun imzası.....	23
Şekil 3.7 : InformClient fonksiyonunun imzası	24
Şekil 3.8 : İki boyutlu düzlem üzerine yerleştirilmiş kare ve daire.....	25
Şekil 3.9 : Düzlem üzerinde (x,y) koordinatlarındaki bölgenin komşuları	27
Şekil 3.10 : Sıcaklığın başlangıçtaki dağılımı.....	28
Şekil 3.11 : Prosesler arası paylaşım ve iletişim.....	30
Şekil 3.12 : Prosesler arası paylaşım	32
Şekil 3.13 : Örnek matrisin prosesler arası paylaşımı	34
Şekil 3.14 : İstemci uygulamanın arayüzü.....	35
Şekil 3.15 : Örnek konfigürasyon dosyası.....	37
Şekil 3.16 : Kullanıcı tarafından seçilmiş sunucuları içeren örnek dosya.....	39
Şekil 3.17 : Konfigürasyon dosyası oluşturan programın arayüzü	41
Şekil 3.18 : Örnek AgentWorld dosyası.....	43
Şekil 3.19 : Etmen durumu bildirici programın kullanıcı arayüzü.....	44
Şekil 3.20 : Sunucu etmeni yapılandırma dosyası örneği.....	48

JAVA İÇİN NESNE TABANLI MESAJ AKTARIM KÜTÜPHANESİ

ÖZET

Günümüzde paralel programlama, özellikle sunduğu yüksek başarımlı ve verimlilik sayesinde, popüler programlama tekniklerinden biri durumundadır. Paralel algoritmaların oluşturulması sürecinde farklı programlama modellerinden yararlanılabilmektedir. Başlıca paralel programlama modelleri arasında ortak bellek kullanımına dayalı, mesaj aktarımına dayalı ve veri paylaşımına dayalı modeller sayılabilir. Bunlar içerisinde, ayrışık sistemler üzerinde çalışabilme, esneklik ve fonksiyonellik gibi önemli özelliklere sahip olan mesaj aktarımı modeli öne çıkmaktadır. 90'lı yılların ortalarında ortaya konan MPI (Message Passing Interface) belirtimi ile, bu modeli temel alan geliştirimlerin standartlaştırılması hedef alınmıştır. Bununla da kalmamış, beraberinde C, C++ ve Fortran gibi bazı programlama dilleri için uygulama geliştirme arayüzleri de sunulmuştur. Ne var ki, en çok tercih edilen programlama dillerinden biri olan Java, bu diller arasında yer almamıştır. Bu durumun bir sonucu olarak, Java programcıları kendi MPI benzeri iletişim kütüphanelerini geliştirmişlerdir. Birbirinden bağımsız ekipler tarafından yürütülen çalışmalar sonucunda farklı özelliklere sahip geliştirimler ortaya çıkmıştır.

Bu tez çalışmasıyla, mesaj aktarımına dayalı paralel programlama alanında Java için daha önce yapılmış geliştirmelere alternatif olabilecek bir paralel programlama platformu oluşturulmuştur. Geliştirilen platforma “Parallel Programming Platform for Java”, kısaca “PPPJ”, adı verilmiştir. PPPJ, nesne tabanlı mesaj aktarımı yaklaşımını temel alan bir paralel iletişim kütüphanesidir. Kullanıcılara MPI benzeri, sade ve kullanımı kolay bir arayüz sunar. Bu arayüz, bir paralel iletişim ortamının hazırlanmasını, sonlandırılmasını, içeriği hakkında bilgi edinilmesini ve içerisindeki prosesler arasında veri alışverişi yapılmasını sağlayan temel fonksiyonları içerir. PPPJ kütüphanesi, nesne tabanlı oluşu sayesinde, yalnızca uygulama geliştirme sürecini kolaylaştırmakla kalmaz, aynı zamanda ele alınan problemin çözümü ile paralel çalışmanın sağlanması süreçlerinin birbirinden ayrılabilmesine yardımcı olur. Böylelikle, yazılan kaynak kodların farklı problemlerin çözümünde yeniden kullanılabilmesine de zemin hazırlar. Ayrıca, tamamen Java programlama dili kullanılarak geliştirildiğinden, platform bağımsızlık özelliğine de sahiptir. Bir başka deyişle, güncel bir JRE (Java Runtime Environment) yazılımının bulunduğu her bilgisayarda çalışabilir.

PPPJ kütüphanesi istemci-sunucu mimarisi üzerine kurulmuştur. Buna göre sistemdeki bilgisayarlar istemci ve/veya sunucu olarak görev alabilmektedirler. Genel olarak, sunucular kullanıcı uygulamalarının yürütülmesinden sorumlu iken, istemciler bu uygulamaların sunuculara yüklenmesinden sorumludurlar. PPPJ kütüphanesi, bir kullanıcı uygulamasının yürütülmesinde kullanılacak sunucuların seçimi noktasında benzeri geliştirmelere oranla daha çeşitli seçenekler sunar. Kullanıcı, görev alacak sunucuları bizzat kendisi belirleyebileceği gibi, seçimin başarımlı ya da yoğunluk ölçütüne göre kütüphanenin ilgili bileşenleri tarafından

yapılmasını da sağlayabilir. Seçimi kendisi yapmak istediğinde kullanıcıdan o iş için hangi sunucu üzerinde kaç adet proses çalıştırmak istediği bilgisi alınır ve seçim işlemi buna göre yapılır. Seçimin başarımlar ölçütüne göre yapılması istendiğinde, sistemdeki tüm sunucuların başarımları değerlendirilir ve başarımları yüksek olan sunuculara öncelik tanınır. Seçimin yoğunluk ölçütüne göre yapılması istendiğinde ise, sistemdeki sunucuların o anki iş yükleri değerlendirilir ve yoğunluğu düşük olan sunuculara öncelik tanınır. Kütüphane bir çeşit “yük dengeleme” mekanizmasına da sahiptir. Bu özelliği sayesinde, seçim sırasında öncelik tanınan sunuculardan birinde gerekli kaynak ayrımı yapılamadığı takdirde, otomatik olarak bir sonraki yüksek öncelikli sunucu devreye sokulur.

PPPJ kütüphanesinde verilerin bir noktadan diğerine, sıkıştırılarak aktarılması da mümkün kılınmıştır. Böylelikle, özellikle büyük boyutlarda veri transferleri söz konusu olduğunda iletim sürelerinin kısaltılabilmesi hedeflenmiştir.

Ayrıca, kütüphanenin kendisinin ve üzerinde kurulacak kullanıcı uygulamalarının yapılandırılması safhalarında çeşitli konfigürasyon dosyalarından faydalanılarak, bu süreçlere de esneklik kazandırılmıştır. Kütüphane kapsamında sunulan yardımcı araçlarla da söz konusu dosyaların yönetimi kolaylaştırılmıştır.

OBJECT-BASED MESSAGE PASSING LIBRARY FOR JAVA

SUMMARY

Thanks to its high performance and efficiency, parallel programming is currently one of the most popular programming technics. Several programming models can be used to develop parallel algorithms. Major parallel programming models are shared memory, message passing and data parallel models. Message passing model, which has significant features such as portability, flexibility and functionality, comes into prominence among these models. With the MPI (Message Passing Interface) specification which was introduced in the middle '90s, it is aimed to standardize softwares based on this model. The specification also provides bindings for some languages such as C, C++ and Fortran. Nevertheless, it does not provide an official binding for Java which is one of the most popular programming languages. Thus, Java programmers have been developed their own MPI-like communication libraries. Efforts given by separate developer teams led to emergence of different libraries.

In this thesis, an alternative message passing based parallel programming platform is proposed. The platform, which is developed in and for Java, is named as "PPPJ". PPPJ is short for "Parallel Programming Platform for Java". Actually, PPPJ is a parallel communication library which is based on object passing approach. It provides an MPI-like, simple and easy to use interface for users. The interface contains basic functions that perform a variety of operations related with a parallel communication environment. These operations include initializing and finalizing the environment, giving information about it and transferring data between processes in it. Thanks to its object based approach, PPPJ not only simplifies development process but also assists in separation of problem solving and parallelization processes. In this way, it also increases reusability of source codes. Because PPPJ was developed in Java, it is a platform independent library. In other words, it can run on any machine that includes an up-to-date version of Java Runtime Environment.

PPPJ is based on client-server architecture. Accordingly, each computer in the system may work as client and/or server. Servers are responsible for running user programs whereas clients are responsible for loading these programs to servers. PPPJ provides more options than similar libraries when selecting servers on which a user program will run. Users are not only able to select servers manually, but also have the library make selection via performance or occupancy criteria. When user wants to select servers manually, he/she specifies which servers are going to be used and how many processes will be employed on each one. Then the selection is done by his/her choices. When user wants to make selection via performance criteria, performance of each server is evaluated by special components of the library and servers with higher performance value are given higher priority. Similarly, when user wants to make selection via occupancy criteria, occupancy of each server is evaluated and servers with lower occupancy value are given higher priority. Besides, PPPJ library have a kind of "load balancing" mechanism. Owing to this mechanism, if resource

allocation fails on a server during selection phase, it is possible to employ the next high priority server automatically.

PPPJ also supports compressed data transfer between peers. In this way it is aimed to cut transfer times down especially when transferring large data sets.

PPPJ library itself and user programs that will be run on it can be configured with special configuration files. This feature adds flexibility to configuration phases. Moreover, some helpful tools which make management of such files easier are presented within the library.

1 GİRİŞ

Paralel programlama, sağladığı yüksek başarımlar ve düşük maliyeti ile günümüzün popüler programlama tekniklerinden biri durumundadır. Buna paralel olarak, bu tekniğin kullanımını kolaylaştıran yazılım sistemlerine sürekli yenileri eklenmektedir. Mesaj aktarımına dayalı geliştirimler söz konusu yazılım sistemleri içerisinde öne çıkmaktadır. Bu durumun başlıca nedenleri arasında, ayrışık bilgisayar sistemleri ve süper bilgisayarlar üzerinde kullanılmaya elverişli olmaları, türlü paralel algoritmaların geliştirilmesine olanak tanınmaları, bir başka deyişle ifade güçlerinin yüksek olması ve özellikle sergiledikleri esnek bellek yönetimi yaklaşımıyla yüksek başarımlar sağlamaları sayılabilir [1].

Günümüzde mesaj aktarımı modelini temel alan yazılım sistemleri içerisinde en çok rağbet görenler, mesaj aktarımı ve nesne yönelimli programlama kavramlarını bütünleştirenlerdir. Şüphesiz bunda, nesne yönelimli programlama dillerinin popüleritelerindeki artışın payı büyüktür. Bu dillerin en önemlilerinden biri de Java'dır. Mesaj aktarımı modelini temel alan geliştirimlerin standartlaştırılması amacıyla 90'lı yılların ortalarında MPI (Message Passing Interface) [2,3] belirtimi ortaya konulmuştur. Bu belirtimle beraber Java'ya özel bir uygulama geliştirme arayüzü sunulmamıştır. Bununla birlikte, bugüne dek Java için birçok MPI benzeri paralel programlama kütüphanesi geliştirilmiştir. Nesne yönelimli oluşunun yanı sıra, taşınabilirlik özelliği, soket kullanımına ve uzak metot çağrılarına verdiği destek Java'nın paralel programlama alanında da çokça tercih edilmesinin nedenlerindendir.

Bu tez çalışmasıyla, Java dili kullanılarak, nesne tabanlı mesaj aktarımı yaklaşımını temel alan alternatif bir paralel programlama kütüphanesi oluşturulması amaçlanmıştır. Tasarlanan kütüphane kullanıcılara MPI benzeri, ancak sadece temel fonksiyonları içeren, sade ve kullanımı kolay bir arayüz sunar. Nesne tabanlı iletişime verdiği destek ile hem paralel uygulamaların geliştirilmesi sürecini kolaylaştırır hem de ele alınan problemin çözümü ile paralel çalışmanın sağlanması süreçlerinin birbirinden ayrılabilmesine zemin hazırlar. Tamamen Java programlama dili kullanılarak geliştirilmiş olması sayesinde taşınabilirlik özelliğine de sahiptir. Bir

başka deyişle, JRE (Java Runtime Environment) yazılımının güncel bir sürümünün bulunduđu her bilgisayarda kullanılabilir.

Kütüphane istemci-sunucu mimarisi esas alınarak tasarlanmıştır. Buna göre sistemdeki bilgisayarlar istemci, sunucu ya da hem istemci hem sunucu olarak kullanılabilir. Genel olarak istemciler yapılacak işlerin (kullanıcı programlarının) sunuculara yüklenmesinden, sunucular ise söz konusu işlerin yürütülmesinden sorumludurlar. Tasarlanan kütüphane, kullanıcı programlarının yürütüleceğı sunucuların seçilmesi noktasında benzeri geliştirmelere oranla daha fazla seçenek sunar. Kullanıcı, görev alacak sunucuları bizzat kendisi belirleyebileceğı gibi, seçim işleminin başarımı ya da yoğunluk ölçütüne göre kütüphanenin ilgili bileşenleri tarafından yapılmasını da sağlayabilir. Seçimin başarımı ya da yoğunluk ölçütüne göre yapılması talep edildiğinde, kütüphane bünyesinde geliştirilen bir mekanizma ile sistemdeki tüm sunucuların o anki başarımı ya da yoğunlukları ölçülerek, duruma göre başarımı yüksek ya da yoğunluğu düşük olan sunuculara öncelik tanınır. Tasarlanan kütüphane bir çeşit yük dengeleme özelliğine de sahiptir. Bu sayede, seçim esnasında bir sunucu üzerinde gerekli kaynakların ayrılamaması durumunda bir sonraki yüksek öncelikli sunucu devreye alınır.

Tasarlanan kütüphanede, verilerin (nesnelerin) bir noktadan diğerine aktarılmadan önce sıkıştırılarak boyutlarının küçültülebilmesi de öngörülmüştür. Bu nitelik, özellikle büyük boyutlarda veri aktarımları söz konusu olduğunda aktarımlar esnasında oluşabilecek yüksek zaman maliyetlerinin düşürülmesi açısından önemlidir.

Sade ve kullanımı kolay bir uygulama geliştirme arayüzüne sahip olmasının yanı sıra, bu kütüphaneyi kullanıcı dostu yapan etkenlerden bir diğeri ise, hem kendisinin ve hem de üzerinde koşturulacak kullanıcı uygulamalarının çeşitli konfigürasyon dosyaları aracılığıyla kolaylıkla yapılandırılabilir olmasıdır. Bu özellik aynı zamanda onu daha esnek kılar. Örneğin bir kullanıcı programında verilerin daima sıkıştırılarak aktarılması isteniyorsa, bunun için programı oluşturan kaynak kodlar içerisinde herhangi bir değişiklik yapılmasına gerek yoktur. Kullanıcı programının yapılandırılmasında kullanılan konfigürasyon dosyasında bu isteğin belirtilmesi yeterlidir. Ayrıca kütüphane kapsamında sunulan yardımcı araçlarla, bu gibi dosyaların yönetiminin kolaylaştırılması amaçlanmıştır.

Tez çalışması kapsamında tasarlanan kütüphaneye “Parallel Programming Platform for Java”, kısaca “PPPJ”, adı verilmiştir. Takip eden bölümlerde, öncelikle paralel programlama kavramına, özellikle de mesaj aktarımı modeline ve bu modeli temel alan bazı geliştirmelere ilişkin bilgiler verilecek, ardından PPPJ kütüphanesinin özellikleri ve kullanımı örnek uygulamalarla desteklenerek anlatılacaktır.

2 PARALEL PROGRAMLAMA KAVRAMI

Paralel programlama, bir uygulamanın parçalara ayrılarak, her bir parçasının çok işlemcili bir bilgisayar üzerinde eş zamanlı olarak çalıştırılmasına olanak tanıyan bir programlama tekniğidir. Birbirleriyle iletişim halindeki birden fazla bilgisayar üzerinde yapılan paralel programlamaya ise “dağıtık programlama” adı verilir.

Paralel programlamada, yapılacak işlem öncelikle birbirinden bağımsız olarak hesaplanacak parçalara ayrılır ve hesaplamayı gerçekleştirecek birimlere dağıtılır. Sonrasında her bir birimden elde edilen sonuçlar birleştirilerek nihai sonuca ulaşılır. Dolayısıyla, bu teknik genellikle matematiksel problemlerin çözümü gibi kolaylıkla parçalara ayrılabilen işlemlerin yapılmasında kullanılır.

2.1 Paralel Programlamaya Duyulan İhtiyaç

Paralel programlamada temel amaç performans ve verimliliğin artırılmasıdır. Bunlara ek olarak, uzaktaki kaynakların kullanımına olanak tanıdığından, problemlerin büyük veri kümeleri üzerinde çözümü sağlanabilir.

Bu tekniğin başlıca tercih edilme nedenleri şu şekilde sıralanabilir:

- Problemlerin daha kısa sürede çözümüne imkan tanınması.
- Büyük (çözümü zor) problemlerin çözümünü kolaylaştırması.
- Aynı anda birden fazla işin yapılabilmesine olanak tanınması.
- Bir bilgisayar sistemi (ağı) üzerinde kullanıldığında uzaktaki kaynaklara erişime imkan tanınması.
- Maliyetinin (süper bilgisayarlar gibi alternatiflere oranla) düşük olması.

Son yıllarda dağıtık sistemlere ve çok işlemcili bilgisayarlara duyulan eğilim, paralel programlamanın geleceğin programlama tekniği olduğuna işaret etmektedir.

2.2 Paralel Programlama Modelleri

Belli bařlı paralel programlama modelleri olarak, ortak bellek kullanımı, mesaj aktarımı, veri paylaşımı (data parallel) ve kanal (thread) kullanımı sayılabilir. Modellerin birbirlerine göre üstünlüklerini deęerlendirmekten ziyade, uygulamalarının başarımlarını kıyaslamak daha doęru olur. İsteęe baęlı olarak, birden fazla modelin birlikte “karma model” oluřturacak řekilde kullanılması da mümkündür [4].

2.2.1 Ortak Bellek

Ortak bellek kullanımına dayanan yöntemde prosesler, üzerinde asenkron olarak okuma ve yazma işlemleri yapabilecekleri bir ortak adres uzayını kullanırlar. Ortak alana erişimin denetimi için semafor ve kilit gibi yapılar kullanılabilir. Bu modelin avantajı, prosesler arasında veri iletişiminin sağlanması için özel düzenekler geliştirilmesine ihtiyaç bırakmamasıdır. Bu nitelik, özellikle uygulama geliştirim aşamasında kolaylık sağlar. Ne var ki, ortak bellek kullanımı, yerel kaynak yönetiminin ve kararlı uygulamalar geliřtirmenin zorlařması gibi dezavantajlara da sahiptir.

2.2.2 Mesaj Aktarımı

Mesaj aktarımı modeli, birbirlerinden baęımsız çalışan ve her birinin kendisine ait yerel bilgi bankası bulunan proseslerin, kendi aralarında mesaj alışveriři yaparak iletişim sağlanması esasına dayanır.

Mesaj aktarımına dayalı paralel programlamada, işlemciler arasında veri alışveriřini sağlayan kütüphanelerden faydalanılır. Modern programlama dillerinin bir çoęu bu tür kütüphanelerin kullanımını desteklemektedir. Bu amaçla geliştirilen kütüphanelerin standartlařtırılması için 1994 yılında ilk mesaj aktarım arayüzü (MPI) belirtimi ortaya atılmıřtır. Günümüzde MPI (Message Passing Interface), bir IEEE ya da ISO standardı olmamakla birlikte, endüstriyel standart konumundadır [5]. Bir sonraki kısımda MPI konusu daha ayrıntılı olarak incelenecektir.

Mesaj aktarımı modelinin başlıca avantajları řunlardır [1]:

- Farklı donanım altyapıları üzerinde kullanılmaya elverişlidir. Bir bilgisayar ağı üzerinde uygulanabileceği gibi, çok işlemcili bir süper bilgisayar üzerinde de uygulanabilir.
- Çok çeşitli paralel algoritmalar geliştirmeye olanak tanıyan, kullanışlı ve fonksiyonel bir modeldir. Veri paylaşımı modeli gibi derleyici tabanlı modellerde bulunmayan denetim olanağını sunar.
- Bellek yönetimi, günümüzde işlemcilerden yüksek verim almak, bir başka deyişle yüksek başarımlar sağlamak, açısından büyük önem taşımaktadır. Mesaj aktarımı modeli programcıya, kaynakları doğrudan kontrol etme, istediği veriyle istediği prosesi dilediğince ilişkilendirme imkanı tanıyarak bunu sağlar.
- Ortak bellek kullanımı modelini temel alan sistemlere göre hata ayıklama süreci daha kolaydır.

Diğer yandan, mesaj aktarımı modelinde programcının üzerine yüklenen sorumluluğun yüksek oluşu bir dezavantaj olarak düşünülebilir. Bu modelde programcı, veri paylaşımının, prosesler arası iletişimin ve senkronizasyonun sağlanmasından bizzat sorumludur. Bunların yanı sıra, çıkmaz (deadlock) ve yarış koşulu (race condition) oluşumunun önüne geçmek gibi görevler de yine programcıya bırakılmıştır [6].

2.2.3 Veri Paylaşımı

Veri paylaşımına dayanan yöntemde prosesler, aynı veri yapısının farklı kısımları üzerinde eşzamanlı olarak çalışırlar. Bir başka deyişle, her bir proses aynı işlemi eldeki veri kümesinin farklı bir bölümüne uygular. Bu yönüyle veri paylaşımı modeli tipik bir SPMD (Single Program Multiple Data) yaklaşımı sergiler. Dolayısıyla bu yöntemde yapılması gereken öncelikli iş, verinin prosesler için bölümlenmesidir [7].

Bu yöntemle programlama genellikle veri paylaşımına olanak tanıyan programlama dilleri kullanılarak yapılır. Bu dillere örnek olarak Fortran 90 ve High Performance Fortran (HPF) verilebilir.

Veri paylaşımı yaklaşımının en önemli özelliği iletişimi sağlayan program kodunun derleyici tarafından oluşturuluyor olmasıdır. Bu özellik programcının yükünü hafifletir ve alternatif paralel algoritmaların geliştirilmesini kolaylaştırır. Bu modelin

en önemli dezavantajı ise eldeki veri yapısının bölümlenmeye elverişli olmadığı durumlarda kullanılmasının mümkün olmayışdır.

2.2.4 Kanallar

Kanal kullanımıyla bir prosesin birden fazla çalışma rotası tanımlaması mümkündür. Ana proses ihtiyaçlar nispetinde “çocuk” prosesler oluşturarak, birden fazla işin paralel olarak yürütülmesini sağlayabilir.

Kanal kullanımına standart getirmeye yönelik bağımsız çalışmalar sonucunda iki farklı uygulama geliştirme arayüzü ortaya çıkmıştır [4]: OpenMP [8] ve POSIX Threads.

OpenMP, ortak bellek kullanımını destekleyen çok işlemcili bilgisayar sistemleri için tasarlanmıştır. Programcıya, farklı kanallar içerisinde çalıştırılacak paralel kod kısımları tanımlama imkanı verir. Yeni bir kanalın açılması işlemi arkaplanda, programcıdan “gizli” olarak gerçekleştirilir. Öğrenilmesinin ve kullanımının kolay oluşu OpenMP arayüzünün en önemli avantajıdır. Buna karşın, paralellik anlayışının çok yönlülüğünden yoksun oluşu ise en büyük dezavantajıdır. POSIX Threads alternatifi daha esnek bir yaklaşım sergilediğinden, OpenMP ile zor çözülebilecek, hatta çözülemeyecek problemlerin çözümünde kullanılabilir.

POSIX Threads arayüzü, kanalların açılması ve yönetilmesi için C programlama dilinde oluşturulmuş veri tipleri ve fonksiyonlar tanımlar [9]. Altyapısal işlemler programcıdan gizlenmez. Aksine, bir kanalın açılması, sonlandırılması, bekletilmesi gibi işlemler, sağlanan fonksiyonlar yardımıyla bizzat programcı tarafından gerçekleştirilir. Bu özellik, POSIX Threads arayüzünün sağladığı esnek yaklaşımın temelini oluşturur. Diğer yandan, programcının üzerindeki yükü artırdığından, daha fazla dikkat göstermesini gerektirir [4].

2.3 Mesaj Aktarım Arayüzü

Mesaj aktarım arayüzü, kısa adıyla MPI, bilgisayar sistemleri üzerinde mesaj aktarımı modeline dayalı paralel programların geliştiriminin standartlaştırılması amacıyla ortaya atılmış bir belirtimdir. Bir programlama dili değil, bir programlama arayüzüdür. Özellikle dağıtık bellekli sistemler üzerinde kullanılır. Temel amacı, taşınabilir, verimli ve esnek programların geliştirimine zemin hazırlamaktır.

MPI belirtiminin MPI-1 ve MPI-2 olarak adlandırılan iki sürümü bulunmaktadır. MPI-2 sürümü, sağladığı fonksiyon adedinin oldukça fazla (500’den fazla) olmasının yanı sıra, dinamik proses yönetimine ve paralel okuma yazma işlemlerine olanak tanınması gibi MPI-1 belirtiminde bulunmayan özelliklere sahiptir. Bununla birlikte, “geriye uyumluluk” özelliği sayesinde MPI-1 standardına uygun yazılmış uygulamalar, MPI-2 standardı altında da çalışabilmektedir [10].

2.3.1 Temel Fonksiyonlar

MPI belirtimine uygun bir programın hazırlanması basitçe üç ana safhada tamamlanır:

1. İletişim altyapısının hazırlanması,
2. Prosesler arasında veri alışverişi yoluyla iletişimin sağlanması,
3. İletişimin sonlandırılması.

MPI-2 belirtimi çok sayıda fonksiyona sahiptir. Diğer yandan, bu fonksiyonlar içinden “temel fonksiyonlar” olarak adlandırılacak altı tanesinin bilinmesi, ortalama bir MPI programı yazmak için yeterlidir. Bu fonksiyonlar şöyle sıralanabilir:

1. MPI_INIT
2. MPI_COMM_SIZE
3. MPI_COMM_RANK
4. MPI_SEND
5. MPI_RECV
6. MPI_FINALIZE

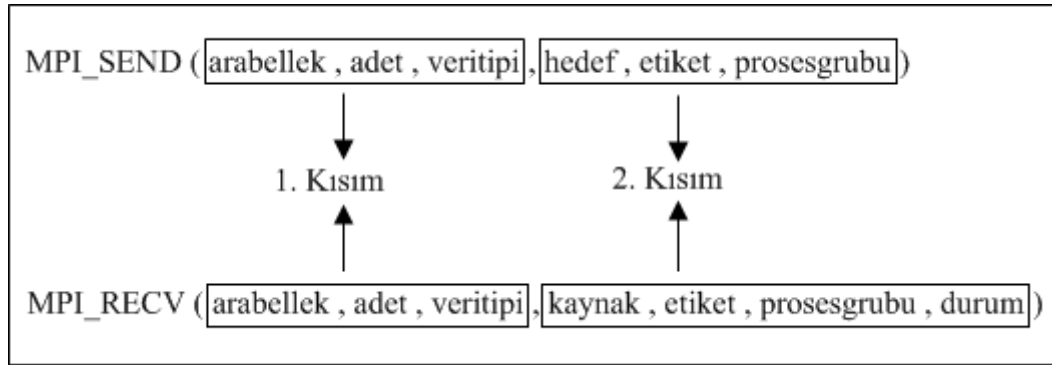
Bunlardan MPI_INIT, MPI_COMM_SIZE ve MPI_COMM_RANK fonksiyonları, iletişim altyapısının hazırlanması safhasında kullanılır. MPI_INIT fonksiyonunun görevi mesaj aktarım sistemini hazır duruma getirmektir ve ilk çağırılması gereken MPI fonksiyonudur. MPI_COMM_SIZE fonksiyonu sistemdeki toplam proses sayısı hakkında bilgi verir. MPI_COMM_RANK ise geçerli prosesin sıra numarasını döndürür.

MPI_SEND ve MPI_RECV fonksiyonları prosesler arasında veri alışverişinin yapıldığı ikinci safhada kullanılır. MPI_SEND mesaj gönderimini, MPI_RECV ise mesaj alımını sağlar.

MPI_FINALIZE fonksiyonu ise üçüncü ve son safhada kullanılır. Görevi, mesaj aktarımına dayalı iletişimin sonlandırılmasıdır.

2.3.2 Mesaj Yapısı

MPI mesajları basitçe iki ana kısımdan oluşur. Birinci kısım gönderilen veya alınan veriyle ilgili bilgileri, ikinci kısım ise gönderim ve alım işlemlerinin eşleştirilmesi için gerekli bilgileri içerir [11]. Söz konusu kısımlar Şekil 2.1’de gösterilmiştir.



Şekil 2.1 : Mesaj kısımlarının fonksiyon imzaları üzerinde gösterimi

“Arabellek” parametresi MPI_SEND fonksiyonunda gönderilecek verinin bellekteki adresini, MPI_RECV fonksiyonunda ise okunacak verinin bellekte yazılmaya başlayacağı adresi gösterir. “Adet” parametresi MPI_SEND fonksiyonunda gönderilecek veri miktarını belirtir. MPI_RECV fonksiyonunda ise yazma işlemi yapılacak bellek bölgesinin uzunluğuna işaret eder. Bu uzunluk, gönderilen verinin uzunluğuna eşit ya da ondan büyük olmalıdır, aksi takdirde hata oluşur. “Veritipi” parametresi ise gönderilen ya da alınan verinin tipini belirtir. Bu parametre MPI_CHAR, MPI_INT, MPI_FLOAT gibi MPI belirtiminde tanımlı değerleri alabilir.

İkinci kısmı oluşturan parametrelerden, MPI_SEND fonksiyonunda yeralan “hedef” parametresi, mesajın iletileceği prosesin sıra numarasını gösterir. MPI_RECV fonksiyonundaki “kaynak” parametresi ise kendisinden mesaj beklenen prosesin sıra numarasına işaret eder. Bu parametreye MPI_ANY_SOURCE değeri verildiğinde herhangi bir kaynaktan gelen mesaj kabul edilir. “Etiket” parametresi ise gönderilen

ve alınan mesajların eşleştirilmesinde kullanılan bir tamsayı değeri içerir. MPI_RECV fonksiyonunda etiket değeri olarak MPI_ANY_TAG kullanıldığında, gelen mesaj etiket değerine bakılmaksızın kabul edilir.

MPI belirtiminde, kararlı programların oluşturulabilmesi için mesaj alışverişinde hedef ve kaynak belirtimi, etiket kullanımı gibi düzenekler geliştirilmiştir. Ne var ki, bu denetim mekanizmaları hata ihtimalini tamamen ortadan kaldırmaz. Örneğin, belli bir işlemi gerçekleştirirken bağımsız bir kütüphaneden faydalanan bir MPI programı, kütüphane içerisindeki mesaj alışverişlerinde hangi etiketlerin kullanıldığını bilemeyebilir. Bir önlem alınmadığında, bu durum düşük de olsa mesajların karışması ihtimalini beraberinde getirir. MPI belirtiminin sağladığı “communicator” yapısı bu ve benzeri sorunlara çözüm getirmektedir [12]. Communicator yapısı birbirleriyle iletişim halindeki bir proses grubunu tanımlar. Yeni bir proses grubunun oluşturulması, aynı zamanda yeni bir bağlam oluşturulması anlamına gelir. Farklı bağlamlarda faaliyet gösteren prosesler, mesaj alışverişleri esnasında aynı hedef, kaynak ve etiket değerlerini kullansalar bile, mesajların karışması gibi bir ihtimal söz konusu değildir. MPI_SEND ve MPI_RECV fonksiyonlarındaki “prosesgrubu” parametresi, iletişimin gerçekleştirileceği proses grubunun (bağlamın) belirtilmesi içindir. Varsayılan bağlam belirteci, oturumdaki proseslerin tümünü birden temsil eden MPI_COMM_WORLD değeridir.

MPI_RECV fonksiyonundaki “durum” parametresinin alacağı değer, mesaj alım işlemi gerçekleştikten sonra belirlenir. Bu parametre kullanılarak, alınan mesajın boyutu, kaynağı ve etiketi hakkında bilgi edinilebilir. Örneğin, mesaj alımı sırasında kaynak parametresine MPI_ANY_SOURCE değeri verilmişse, işlem tamamlandıktan sonra, mesajın hangi kaynaktan geldiği bilgisi durum parametresine bakılarak öğrenilebilir.

2.3.3 MPI Gerçekleştirimleri

MPI belirtimi ortaya çıkarılırken, bir yandan da taşınabilir bir uygulamasının geliştirilmesine yönelik çalışmalar başlatılmış, böylelikle hazırlanan kısımlar için erkenden geri bildirim alınması hedeflenmiştir [13]. Birçok MPI gerçekleştirimi bulunmakla birlikte, en önemlileri arasında MPICH (MPI Chameleon) [14], LAM/MPI (Local Area Multicomputer) [15] ve Open MPI [16] sayılabilir.

MPICH, MPI belirtiminin ilk gerekleřtirmisi olmasının yanı sıra, belirtimi tmyle uygulayan, tařınabilir ve verimli bir gerekleřtirmedir. İsmindeki bu kalemin kelimesi tařınabilirlik zelliđine atfen konulmuřtur [13]. Dahası MPICH, kendisinden sonra geliřtirilen mesaj aktarımına dayalı birok sistemin temelini oluřturmaktadır.

LAM/MPI, ayrıřık bilgisayar sistemleri zerinde mesaj aktarımına dayalı uygulamalar geliřtirmek iin tasarlanmıř bir programlama platformudur. LAM/MPI sayesinde btn bir bilgisayar sistemi, tek bir paralel bilgisayar gibi kullanılabilir. LAM/MPI, MPI-1 ve MPI-2 belirtimlerini tmyle gerekleřtirir. Ayrıca, geliřmiř bir denetleme ve hata ayıklama sistemine sahiptir [13].

Open MPI, varolan teknolojilerin stn zelliklerinin birleřtirilerek, en iyi MPI ktphanesinin geliřtirilmesi fikrinden ortaya ıkmıřtır. LAM/MPI gerekleřtirmisi de Open MPI projesine katılan alıřmalar arasında yer almaktadır. Open MPI gerekleřtirmisinin bařlıca zellikleri řunlardır [16]:

- MPI-2 belirtimini tmyle gerekleme,
- Dinamik proses ynetimi,
- ok sayıda iřletim sistemi desteđi,
- Tm platformlarda yksek bařarım,
- Ayrıřık sistemler zerinde alıřabilme,
- Kanal gvenliđi,
- Hata oluřumuna karřı tolerans.

2.3.4 Java İin MPI Benzeri Gerekleřtirmeler

MPI bir gerekleřtirmisi deđil, bir belirtimdir. Bununla birlikte, C, C++ ve Fortran gibi bazı programlama dilleri iin uygulama geliřtirme arayzleri de sunar. Ne var ki, Java bu diller arasında yer almaz. Buna karřın, Java iin birbirinden bađımsız ekipler tarafından geliřtirilen mesaj aktarım ktphaneleri bulunmaktadır. Bu blmde, sz konusu ktphanelere iliřkin rnekler verilecektir.

2.3.4.1 mpiJava

Java, dağıtık sistemler üzerinde iletişimin sağlanabilmesi için soket kullanımı ve RMI (Remote Method Invocation) gibi çözümler sunar. Ne var ki bunlar daha çok, istemci-sunucu iletişimine dayalı çalışan sistemler için düşünülmüş mekanizmalardır. Paralel programlamanın odağında ise birimler arasında bire bir ve toplu iletişimin sağlanması vardır. MPI sistemi bu tarz iletişime destek verdiği için, Java dili için paralel programlama arayüzü geliştirilirken doğrudan bir yerel MPI kütüphanesinden yararlanılabileceği düşünülmüştür. Bu yöntemin en önemli getirilerinden biri yüksek başarımlı sağlamasıdır.

mpiJava [17], JNI (Java Native Interface) aracılığıyla, yerel MPI geliştirimine Java katmanı ekleyen bir yazılım paketidir. İlk resmi sürümü 1998 yılında yapılmış, takip eden yıllarda özellikle yerel MPI geliştirimi ile Java katmanı arasında ortaya çıkabilecek sorunların giderilmesine yönelik çalışmalar yapılmıştır. mpiJava arayüzü 1997 yılında MPI-2 belirtiminde ortaya konan C++ sürümünü temel almıştır.

mpiJava, platform bağımlı yerel bir MPI geliştiriminin varlığına gereksinim duyar. Bu durum bazı dezavantajları beraberinde getirir [18]:

- Kurulum süreci iki aşamalıdır. Öncelikle yerel MPI kütüphanesi, sonra Java katmanı kurulur.
- Java sanal makinesi ortamı ile yerel kütüphane arasında uyumsuzluklar çıkabilmektedir.
- Java platform bağımsız bir programlama dili olduğundan, izlenen yol genel anlamda Java'nın ilkeleriyle çatışmaktadır.

2.3.4.2 CCJ

CCJ [19], tamamen Java dilinde yazılmış RMI tabanlı bir toplu iletişim kütüphanesidir. Mesaj aktarımı ve toplu iletişim kavramlarını, Java'nın nesne yönelimli yaklaşımıyla bütünleştirir. MPI belirtimindeki benzer şekilde, toplu iletişim sağlayan fonksiyonlar ile noktalar arası iletişime olanak tanıyan az sayıda fonksiyona sahiptir. Ayrıca, MPI belirtimindeki communicator yapısıyla benzer işleve sahip olan "kanal grubu" yapısını tanımlar.

CCJ, kullanımı kolay bir kütüphanedir. Başarımı, kullanılan RMI sisteminin başarımıyla doğru orantılıdır. Dolayısıyla, verimli bir RMI gerçekleştirimiyle

kullanıldığında başarımı artar. CCJ, her türlü veri yapısının aktarımına olanak tanır. Özellikle karmaşık veri yapılarının aktarımı söz konusu olduğunda, alternatifleriyle boy ölçüşebilecek düzeyde yüksek performans gösterir [19].

2.3.4.3 JOPI

JOPI [20], Java programcılarına dağıtık sistemler üzerinde nesne alışverişine dayalı paralel programlar yazma imkanı sunan bir kütüphanedir. Tamamen Java dilinde yazılmıştır ve iletişim altyapısı soket kullanımına dayanır. Nesneler, Java'nın serileştirme (serialization) özelliği kullanılarak eş zamanlı çalışan prosesler arasında aktarılır. Nesne aktarımı yalnızca iki proses arasında gerçekleşebileceği gibi aynı anda birden fazla prosese gönderim yapılması da mümkündür.

Prosesler arasında nesne aktarımına verdiği destek, JOPI kütüphanesini klasik mesaj aktarımına dayalı mimarilerden daha üstün kılar. Bu önemli niteliğin sağladığı faydalar şöyle sıralanabilir:

- Programlamayı ve yazılan programların denetimini kolaylaştırır.
- Prosesler arasında yalnızca veri değil, verinin nasıl kullanılacağı bilgisini saklayan program kodları da taşınır.
- Problemin çözümü ile paralel çalışmanın sağlanması süreçleri birbirinden ayrılabilir. Böylelikle, yazılan program kodlarının farklı problemlerin çözümünde tekrar kullanılması sağlanabilir.
- Karmaşık veri yapılarının prosesler arasında aktarımı kolaylıkla sağlanabilir.

Öte yandan JOPI, mesaj aktarımına dayalı çalışan alternatiflerine göre bazı dezavantajlara da sahiptir [20]:

- Prosesler arasında aktarılan verinin boyutu küçük olduğunda nesne kullanımı getirdiği sarmalama (encapsulation) yükü nedeniyle başarımı düşürür.
- Java dilinde dizilerin yönetimi C diline göre daha güç ve maliyetlidir.
- Java dili genel itibarıyla C diline göre daha yavaştır.

Sonuç olarak, JOPI özellikle büyük boyutlarda veri aktarımı söz konusu olduğunda tercih edilebilecek ve farklı platformlarda kolaylıkla kullanılabilir (taşınabilir) bir paralel programlama kütüphanesidir.

2.3.5 Java İçin Mesaj Aktarım Arayüzü

Bu bölümde, MPI belirtimi örnek alınarak Java için hazırlanmış standart mesaj aktarım arayüzü tanıtılacaktır.

2.3.5.1 Tarihçe

90'lı yılların ikinci yarısında, Java'nın popüler bir programlama dili haline gelmesiyle birlikte, çeşitli gruplar tarafından bu dilde yazılmış MPI benzeri kütüphaneler oluşturulmaya başlandı. O dönemde Java için tasarlanmış standart bir mesaj aktarım arayüzünün olmayışı bu tarz çalışmalar önündeki en önemli engeldi. 1998 yılında toplanan "Java Grande" forumunda oluşturulan çalışma grubu tarafından ilk taslak belirtim ortaya kondu. Aynı grubun çalışmaları sonucunda MPJ (Message Passing Interface for Java) [21] belirtimi ortaya çıktı.

MPJ belirtiminin gerçekleşmesi konusunda iki farklı yaklaşım benimsenebilirdi; varolan yerel MPI kütüphanelerinin üzerine bir yapı inşa etmek ya da geliştirimi tamamen Java dilinde yapmak. Birinci seçenek daha yüksek başarımlar sunmakla birlikte, Java programlama dilinin değerler sistemiyle çatışmaktaydı. Çünkü Java'nın en önemli özelliklerinden biri platform bağımsız bir dil oluşuydu ve bunun bir sonucu olarak, Java'da yazılan bir program, üzerinde herhangi bir değişiklik yapılmasına gerek kalmadan (Java sanal makinesi barındıran) farklı platformlar üzerinde çalışabilmeliydi. Bu nedenledir ki, MPJ belirtiminin gerçekleştirilmesine yönelik çalışmalar daha çok ikinci seçenek üzerinde yoğunlaştı.

2000 yılında, mpiJava kütüphanesini oluşturan takım tarafından, tamamen Java kullanımına dayalı bir arayüz tasarlandı [18]. 2002 yılında ise Massachusetts üniversitesinden bir grup tarafından JMPI kütüphanesi bir MPJ gerçekleştirimi olarak ortaya kondu [22].

2.3.5.2 JMPI

JMPI (Java Message Passing Interface) [22], MPJ belirtimini gerçekleyen, Java dilinde yazılmış bir paralel programlama kütüphanesidir. Tamamen Java'da yazılmış olması nedeniyle, Java sanal makinesinin bulunduğu her türlü platformda çalışabilen, yani taşınabilir bir gerçekleştirmedir. Java'nın uzak metot çağırımı (remote method invocation), nesne serileştirme (object serialization) ve iç gözlem (introspection) özelliklerini kullanır. Nesne aktarımı sırasında uzak metot çağırımı ve nesne

serileştirme özelliklerinden faydalanır. İç gözlem özelliği ise çok boyutlu dizilerin aktarımı sırasında dizinin boyutunun belirlenmesi gibi işlemler için kullanılır.

Nesne aktarımı sırasında nesne serileştirme işlemi ek yük getirmekte, RMI kullanımı ise zaman maliyetini artırmaktadır. Büyük boyutlardaki verilerin aktarımı söz konusu olduğunda bu maliyetler göz ardı edilebileceğinden JMPI gerçekleştiriminin başarımı artar. Ayrıca başarımı, kullanılan RMI sisteminin başarımıyla doğru orantılıdır [22].

3 JAVA İÇİN NESNE TABANLI MESAJ AKTARIM KÜTÜPHANESİ

Karmaşık, büyük problemlerin kısa sürede çözümüne olanak tanınması ve yüksek başarımların sağlanması, paralel programlamaya duyulan ilgiyi gün geçtikçe arttırmaktadır. Buna paralel olarak, bu tekniğin kullanımını kolaylaştıran yazılımların sayısı da artış göstermektedir. Kimi programlama dilleri (Occam, Ada gibi) paralel programlamaya doğrudan destek verirken, kimileri de bu iş için özel olarak geliştirilmiş kütüphanelerden yararlanmaktadırlar. Söz konusu kütüphaneler içerisinde, özellikle mesaj aktarımı modelini temel alanlar öne çıkmaktadır.

Java, mesaj aktarımına dayalı paralel programlama kütüphanelerinin geliştirilmesinde sık kullanılan diller arasındadır. Nesne yönelimli, güvenli, taşınabilir, kısacası güçlü ve yazılım dünyasında çok kullanılan bir dil oluşu, paralel programlama alanında da tercih edilmesinin başlıca nedenlerindendir. Java, soket kullanımına ve uzak metot çağrılarının imkanı tanınması gibi paralel programlama tekniğinin uygulanması aşamasında doğrudan kullanılacak özelliklere de sahiptir. Ne var ki, bu yapıların tek başlarına kullanımı bazı dezavantajları beraberinde getirir. Örneğin doğrudan soketler kullanılarak iletişim sağlanması yazılım sürecini karmaşılaştırırken, uzak metot çağrıları başarımları düşürür [20]. Bu gibi nedenlerden dolayı, kullanımı kolay ve başarımları yüksek kütüphanelerin oluşturulması yoluna gidilmiştir. Bu kütüphanelerden bazıları varolan yerel mesaj aktarım sistemlerinin üzerine inşa edilmiş (örneğin mpiJava), bazıları (örneğin JOPI, CCJ, JMPI) ise Java'nın sunduğu imkanlardan faydalanmak suretiyle tümüyle bu dilde geliştirilmiştir.

Bu çalışmanın amacı da, Java dilinde, nesne tabanlı mesaj aktarımı modelini temel alan bir paralel programlama kütüphanesinin geliştirilmesidir. Hazırlanan kütüphaneye İngilizce “Parallel Programming Platform for Java” ismi verilmiştir. Takip eden kısımlarda kolaylık olması açısından bu ismin kısaltması olan “PPPJ” adıyla anılacaktır.

3.1 Genel Özellikler

PPPJ, kullanıcılara MPI benzeri - ancak sadece temel fonksiyonları içeren - basit bir arayüz sunan bir iletişim kütüphanesidir. Temel amacı, dağıtık bilgisayar sistemleri üzerinde nesne aktarımına dayalı paralel programlamaya imkan tanımadır. Tümüyle Java'da yazılmıştır ve bu dilin sahip olduğu soket desteği, iç gözlem, nesne serileştirme gibi özelliklerden faydalanır. Sahip olduğu sade ve basit arayüz sayesinde, kullanımı klasik MPI geliştirimlerine oranla daha kolaydır. En önemli özelliği ise mesaj aktarım altyapısının nesne tabanlı oluşudur. Çalışma zamanında, iletişim halindeki prosesler aralarında nesne alışverişi yaparak haberleşirler. İletişim esnasında nesnelerin kullanılması, yazılım geliştirme sürecini kolaylaştırır ve yazılan program kodlarının tekrar tekrar kullanılabilmesine zemin hazırlar. PPPJ'nin bir diğer önemli özelliği ise, platform bağımsız bir kütüphane oluşudur. Bu özelliği sayesinde, Java Çalışma Zamanı Ortamı (Java Runtime Environment) yazılımının güncel bir sürümünü barındıran her platformda çalışabilir.

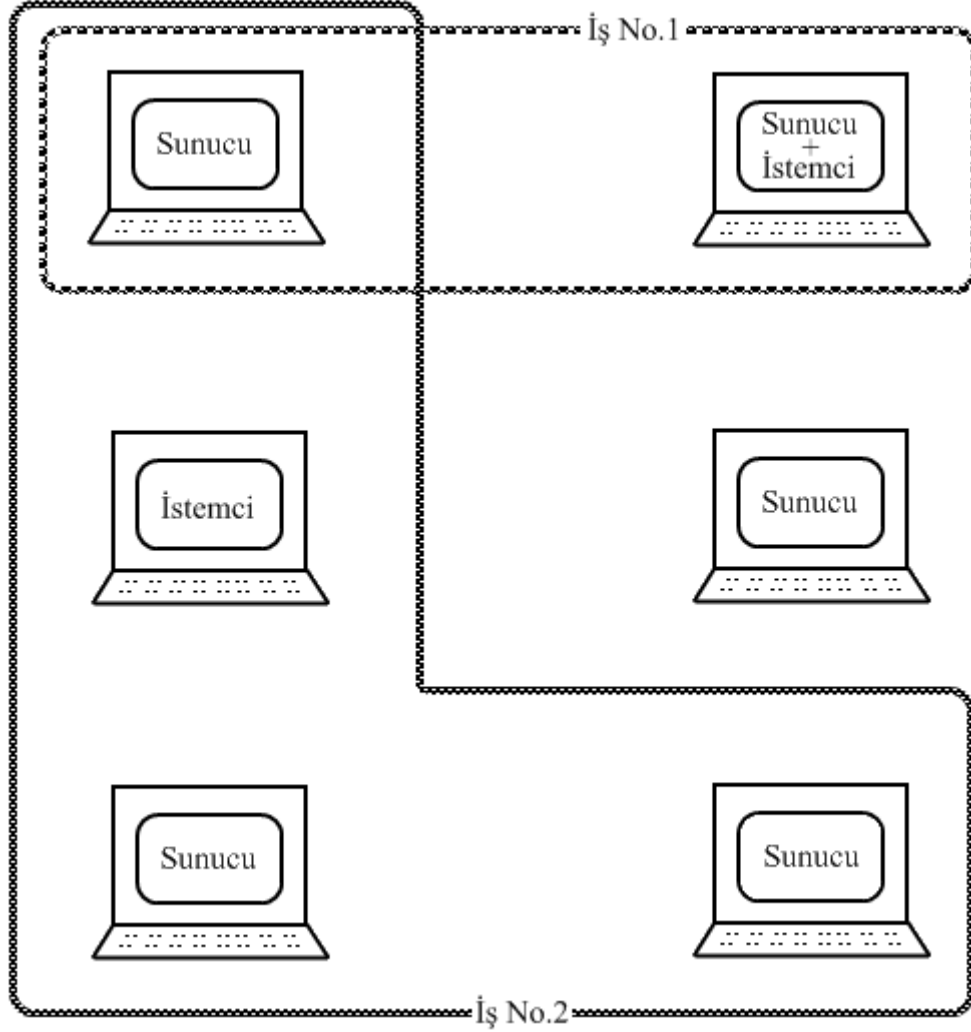
3.1.1 İstemci – Sunucu Mimarisi

PPPJ kütüphanesinin kullanıldığı bir sistemde bilgisayarlar üç farklı rol üstlenebilirler. Sunucu, istemci ya da hem sunucu hem istemci olabilirler. Sunucu olarak faaliyet gösterecek olan bir bilgisayarın üzerinde, kütüphanenin “sunucu etmeni” bileşeninin sürekli olarak çalışır durumda olması gerekir. Bu etmenin temel görevi, istemcilerden gelecek olan isteklerin dinlenerek cevaplanmasıdır. İstemcilerden gelebilecek isteklerin bazıları şunlardır:

- Yeni bir işin başlatılması.
- Devam etmekte olan bir işin sonlandırılması.
- Yeni bir iş için kaynak ayrılması.
- Sunucunun başarımının değerlendirilmesi.
- Sunucunun yük durumunun değerlendirilmesi.

Bunların yanı sıra, sisteme yeni eklenen bir etmenin tanınması ya da varlığı önceden bilinen bir etmenin tanınanlar listesinden çıkartılması gibi istekler de sunucu etmeni tarafından yerine getirilir. Şekil 3.1'de de görüldüğü gibi, bir sunucu etmeni aynı anda birden fazla işi yürütebilir. Hatta gerektiğinde ya da istendiğinde belli bir iş için

aynı anda birden fazla proses çalıştırabilir. Teorik olarak aynı anda yürütülebilecek maksimum iş (proses) adedi sınırsızdır. Ancak, isteğe bağlı olarak bir üst sınır belirlenir ve sunucu etmeninin konfigürasyon dosyasına kaydedilir. Etmen üzerinde aynı anda aktif durumda bulunan iş (proses) adedi söz konusu üst sınıra ulaştığında, işlerden en az bir tanesi sonlanana dek istemcilerden gelen yeni iş istekleri reddedilir.



Şekil 3.1 : Örnek istemci-sunucu mimarisi

Genel itibariyle, sistemdeki sunucu sayısının, sistemin hesaplama gücüyle doğru orantılı olduğu söylenebilir. Öte yandan, sistemde birden fazla sunucu olabileceği gibi, birden fazla istemci de bulunabilir. İstemcilerin başlıca görevleri sunucularla olan eş güdümü sağlamak ve işlerin başlangıcından bitişine kadar olan süreci yönetmektir. Ayrıca, yapılan işlerle ilgili olarak sunuculardan gelebilecek mesajların dinlenmesi ve gerektiğinde kullanıcıya gösterilmesi de görevleri arasındadır.

3.1.2 Kaynak Yönetimi

PPPJ kütüphanesinde sunucu etmenleri, istemcilerin kullanımına sunulmuş en önemli kaynaklardır. Dolayısıyla, istemci tarafındaki kaynak yönetimi, genel itibariyle, sistemdeki sunucuların kayıtlarının tutulmasından ibarettir. Sistemdeki sunucu etmenlerinin listesi “AgentWorld” isimli bir XML dosyasında tutulur. İstemciler bu dosyayı okuyarak, sistemde aktif durumda bulunan sunuculara ilişkin bilgi edinirler. Ayrıca, söz konusu dosya üzerindeki değişiklikleri takip ederek, gerektiğinde kendi kayıtlarını güncellerler.

Sunucu tarafındaki kaynak yönetimi ise nispeten daha karmaşıktır. Sunucu etmenleri, barındırdıkları kaynak yöneticisi bileşeni sayesinde sistemdeki iş yükünü denetim altında tutarlar. Kaynak yöneticisi, ait olduğu etmen üzerinde faaliyet gösteren proseslerin ve iletişim esnasında kullanılan kapı (port) numaralarının listesini tutar. Gerektiğinde - genelde bir istemcinin bilgilendirilmesi amacıyla - etmen üzerindeki iş yükünü hesaplar. Yeni bir iş için gerekli kaynakların ayrılmasından sorumlu olduğu gibi, iş sonlandıktan sonra kaynakların geri alınmasından da sorumludur.

3.1.3 İletişime Yardımcı Yapılar (Communicators)

PPPJ kütüphanesi MPI belirtimindekine benzer bir communicator yapısı tanımlar. Bu yapılar “Communicator” sınıfının örneği olarak yaratılan nesneler ile temsil edilir. Bu nesnelerin her biri, belli bir işi yerine getirmek için kurulmuş bir paralel iletişim ortamı hakkında detaylı bilgiler saklar. Söz konusu bilgiler üç ana grupta toplanabilir:

- Yapılacak işi tanımlayan bilgiler.
- İşin sahibi olan istemciye ilişkin bilgiler.
- İş yerine getirmekle görevlendirilmiş prosesler ile ilgili bilgiler.

Communicator nesneleri istemci tarafında oluşturulur ve ilgili işi yapacak olan sunucu etmenlerine gönderilir. Sunucu etmenleri bu nesneleri gerektiğinde kullanmak üzere dosyalara kaydederler.

3.2 Uygulama Geliştirme Arayüzü

PPPJ kütüphanesinin “TriplePJAssistant” sınıfı nesne alışverişine dayalı iletişim için gerekli temel fonksiyonları barındırır. Bu fonksiyonların isimleri, genel itibariyle program kodları içerisinde kullanılabilecekleri yerler göz önüne alınarak şöyle sıralanabilir:

- Start
- GetProcessID
- GetCommSize
- Send
- Multicast
- Broadcast
- Receive
- InformClient
- Finish

Genel olarak yerine getirdikleri görevler ise şunlardır:

- Paralel iletişim ortamının kurulması ve kaldırılması.
- Paralel iletişim ortamı hakkında bilgi sağlanması.
- Mesaj alışverişinin sağlanması.
- İstemciye (kullanıcıya) bilgi mesajları gönderilmesi.

Fonksiyonları kullanabilmek için öncelikle söz konusu sınıfın bir örneğinin oluşturulması gerekir (Şekil 3.2). Şekildeki “args” parametresi bir karakter katarı (String) dizisine işaret eder. Bu dizi iki eleman içerir. Birinci eleman geçerli prosesin sıra numarasını saklar. İkinci eleman ise, paralel iletişim ortamı hakkında çeşitli bilgiler barındıran Communicator nesnesinin yazıldığı (serileştirildiği) dosyanın yolunu içerir. Bu bilgiler iletişim ortamının hazırlanması için kullanılır.

```
TriplePJAssistant assistant = new TriplePJAssistant(args);
```

Şekil 3.2 : TriplePJAssistant sınıfının örneğinin oluşturulması

Bir sonraki kısımda yukarıda bahsi geçen fonksiyonlar daha ayrıntılı olarak incelenecektir.

3.2.1 Fonksiyonlar

Bu bölümde, paralel iletişimin sağlanabilmesi için PPPJ kütüphanesi tarafından uygulama geliştiricilere sunulan fonksiyonlar daha detaylı olarak ele alınacaktır.

3.2.1.1 Start

TriplePJAssistant sınıfının örneklenmesinin ardından ilk çağrılması gereken kütüphane fonksiyonudur. Görevi, paralel iletişim ortamını hazırlamaktır. Bunu yaparken özellikle Communicator nesnesinden faydalanır. “Start” fonksiyonu herhangi bir parametre almaz. Öte yandan, iletişim ortamının başarıyla hazırlanıp hazırlanamadığını bildiren bir değer döndürür.

3.2.1.2 GetProcessID

Geçerli prosesin, paralel iletişim ortamı içerisindeki sıra numarasını döndürür. Proseslerin sıra numaraları istemci tarafından belirlenerek, Communicator nesnesi içerisinde sunucu etmenlerine bildirilir.

3.2.1.3 GetCommSize

Aynı işin kotarılması için eş zamanlı olarak çalışan proseslerin oluşturduğu bir paralel iletişim ortamındaki toplam proses sayısını döndürür. Bu bilgi de Communicator nesnesinden faydalanılarak edinilir.

3.2.1.4 Send

“Send” fonksiyonu bir prosten diğerine nesne aktarımı için kullanılır. Sırasıyla, hedef prosesin sıra numarasını, etiket bilgisini ve gönderilecek olan nesneyi parametre olarak alır (Şekil 3.3). Etiket bilgisi, aynı iki proses arasında birden fazla mesaj (nesne) alışverişi yapılması durumunda, mesajların birbirine karışmaması için kullanılır.

```
boolean send(short targetProcessID, String label, Object actualData)
```

Şekil 3.3 : Send fonksiyonunun imzası

Send fonksiyonu, gönderim işleminin başarıyla tamamlanıp tamamlanamadığını bildiren bir değer döndürür. İşlemin başarıyla tamamlanamaması durumunda, meydana gelen hata, günlük tutma sistemi tarafından kayıt edilir.

Send fonksiyonunda, gönderimden önce verilerin (nesnelerin) sıkıştırılması (compression) da mümkündür. Bu özellik, büyük boyutlarda veri aktarımı söz konusu olduğunda zamandan kazanmak için kullanılabilir. Tek dezavantajı, alıcı tarafın, gelen veriyi doğrudan kullanamaması, öncelikle eski haline çevirmek (decompression) zorunda olmasıdır. Kullanıcı, söz konusu özellikten faydalanmak isteyip istemediğini, çalıştırmak istediği programın konfigürasyon dosyasında belirtir.

3.2.1.5 Multicast

“Multicast” fonksiyonu, aynı send fonksiyonu gibi, prosesler arasında nesne aktarımı amacıyla kullanılır. Ancak bu fonksiyon, öncekinden farklı olarak, bir nesnenin aynı anda birden fazla prosese iletilmesine olanak tanır. Sırasıyla, hedef proseslerin sıra numaralarını içeren bir diziyi, etiket bilgisini ve gönderilecek olan nesneyi parametre olarak alır (Şekil 3.4).

```
void multicast(short[] targetProcessList, String label, Object actualData)
```

Şekil 3.4 : Multicast fonksiyonunun imzası

3.2.1.6 Broadcast

“Broadcast” fonksiyonu, önceki iki fonksiyon gibi, prosesler arasında nesne aktarımı için kullanılır. Diğerlerinden farklı olarak, bir işi yerine getirmek için birlikte (eş zamanlı) çalışan tüm proseslere aynı anda gönderim yapılmasına olanak tanır. Sırasıyla, etiket bilgisini ve gönderilecek olan nesneyi parametre olarak alır (Şekil 3.5).

```
void broadcast(String label, Object actualData)
```

Şekil 3.5 : Broadcast fonksiyonunun imzası

Paralel iletişim ortamındaki proseslerin listesi Communicator nesnesi kullanılarak edinilebildiğinden, aldığı parametreler arasında hedef proseslerin sıra numaraları bulunmaz.

3.2.1.7 Receive

“Receive” fonksiyonu, başka bir procesten nesne almak için kullanılır. Sırasıyla, kaynak prosesin sıra numarasını, etiket bilgisini ve alınacak nesnenin ait olduğu sınıfın adını parametre olarak alır (Şekil 3.6). Söz konusu parametrelerin her biri için, TriplePJAssistant sınıfı içerisinde tanımlı, alternatif bir sabit vardır. Örneğin belli bir procesten değil de, herhangi bir procesten gelen mesaj (nesne) alınacaksa, kaynak prosesin sıra numarası olarak “ANY_PROCESS” sabiti kullanılır. Benzer şekilde, alınacak mesajın belli bir etikete sahip olması şartı aranmıyorsa, etiket bilgisi olarak “ANY_LABEL” sabiti, alınacak nesnenin belli bir sınıfa ait olması şartı aranmıyorsa, sınıf ismi olarak “ANY_CLASS” sabiti kullanılır.

```
Object receive(short sourceProcessID, String label, String className)
```

Şekil 3.6 : Receive fonksiyonunun imzası

Prosesler kendilerine gönderilen mesajları gerektiğinde kullanmak (almak) üzere bir kuyrukta bekletirler. Aslında receive fonksiyonunun yaptığı iş, söz konusu kuyruktan, parametrelerle belirtilen kriterlere uygun bir mesaj çekmekten ibarettir. Eğer kuyrukta kriterlere uygun bir mesaj bulunmuyorsa, receive fonksiyonunu çağıran proses, böyle bir mesaj kuyruğa eklenene dek bloke olur.

3.2.1.8 InformClient

“InformClient” fonksiyonu, istemciye bilgi ve uyarı mesajları göndermekte kullanılır. Parametre olarak yalnızca gönderilecek olan mesaj metnini alır (Şekil 3.7). Geriye ise, mesajın istemciye başarıyla iletilip iletilemediğini bildiren bir değer döndürür.



```
boolean informClient(String message)
```

Şekil 3.7 : InformClient fonksiyonunun imzası

Gönderilen bilgi ya da uyarı mesajı, istemcinin sunuculardan gelen mesajları dinleyen arabirimi tarafından alınarak, kullanıcıya gösterilir.

3.2.1.9 Finish

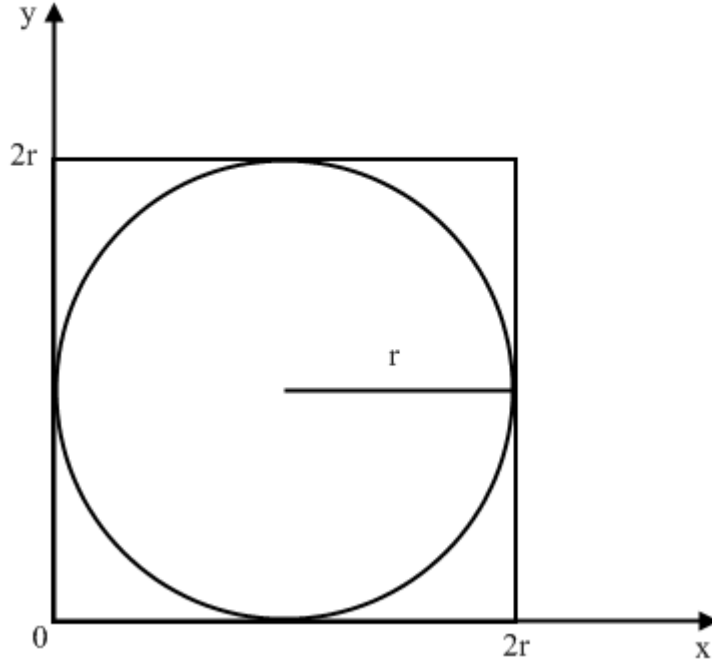
Her zaman son olarak çağırılması gereken kütüphane fonksiyonudur. Görevleri, paralel iletişim ortamını kapatmak ve istemciye işin başarıyla tamamlandığına dair bilgi mesajı göndermektir. Herhangi bir parametre almaz ve geriye bir değer döndürmez.

3.2.2 Örnek Uygulamalar

Bu kısımda, PPPJ kütüphanesi kullanılarak oluşturulmuş örnek paralel uygulamalara ilişkin bilgiler verilecektir. Bahsi geçen örnek programların kodlarına Ek 1'den erişilebilir.

3.2.2.1 Pi Sayısının Hesaplanması

Pi sayısı, bir çemberin çevresinin uzunluğunun, çapının uzunluğuna bölünmesiyle elde edilen sabit sayıdır ve yaklaşık değeri 3,14'tür. Bu değeri, Monte Carlo yöntemini uygulamak suretiyle yaklaşık olarak hesaplamak mümkündür. Bunun için, iki boyutlu bir düzlem üzerine yerleştirilmiş bir kare ve bu kare içerisine yerleştirilmiş bir daire kullanılır (Şekil 3.8).



Şekil 3.8 : İki boyutlu düzlem üzerine yerleştirilmiş kare ve daire

Önce, düzlem üzerinde, karenin kapladığı alanın dışına çıkmayacak şekilde, rastgele noktalar seçilir. Sonrasında, seçilen noktaların dairenin içerisine denk gelme sıklığı (oranı) göz önüne alınarak pi sayısının yaklaşık değeri hesaplanır. Bu yöntemin geçerliliği, aşağıdaki denklemlerle ispatlanabilir.

$$\text{Karenin Alanı} = (2 * r)^2 = 4 * (r)^2 \quad (3.1)$$

$$\text{Dairenin Alanı} = \pi * r^2 \quad (3.2)$$

$$\pi = 4 * (\text{Dairenin Alanı} \div \text{Karenin Alanı}) \quad (3.3)$$

Denklem 3.3'ten de anlaşılabileceği üzere, pi sayısının değeri, dairenin alanının, kendisini çevreleyen karenin alanına oranıyla doğru orantılıdır. Bu denklemden hareketle, daire içerisine isabet eden noktaların sayısı, seçilen toplam nokta sayısına bölündükten sonra, elde edilen sonucun 4 sabiti ile çarpılmasıyla, pi sayısı yaklaşık olarak hesaplanabilir. Öte yandan, bu yöntemin sağlıklı sonuç verebilmesi için düzlem üzerinde çok sayıda nokta seçilmesi gerekmektedir. Bu durum, optimum değere ulaşmak için fazla zamana ihtiyaç olduğu anlamına gelir. Neyse ki, paralel

programlama tekniğinin sunduğu imkanlardan faydalanarak, daha kısa sürede hesaplama yapmak mümkündür.

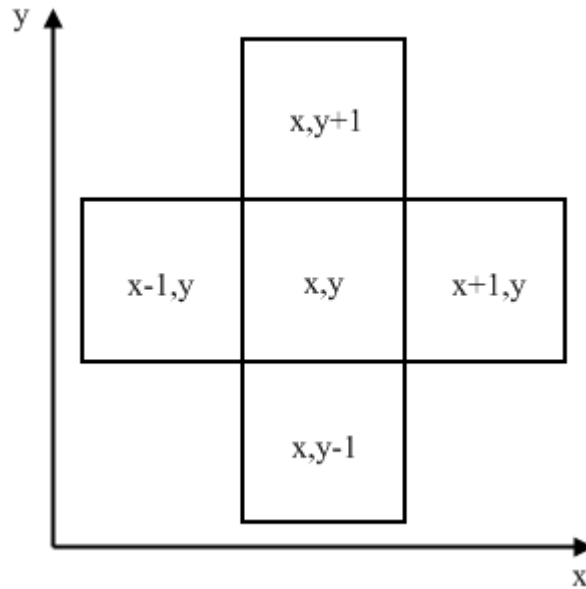
Ek 1’de yer verilen “PiCalculation” programı, Monte Carlo yöntemi ve PPPJ kütüphanesi kullanılarak, pi sayısının değerinin nasıl daha kısa sürede (yaklaşık olarak) hesaplanabileceğini göstermektedir. Söz konusu programı eş zamanlı olarak yürüten proseslerin gerçekleştirdiği işlemler şöyle sıralanabilir:

1. Prosesler paralel iletişim ortamına dahil olurlar ve ortam hakkında bilgi edinirler. Kendi sıra numaralarını ve ortamda toplam kaç proses bulunduğunu öğrenirler.
2. Prosesler, düzlem üzerinde seçilecek toplam nokta sayısını, ortamdaki toplam proses sayısına bölerek, kendi paylarına düşen nokta seçimi adedini hesaplarlar.
3. PiCalculation programında, kare ve dairenin düzlem üzerine Şekil 3.8’de görüldüğü gibi yerleştirildiği ve karenin bir kenar uzunluğunun 1 birim olduğu farzedilmektedir. Dolayısıyla prosesler, düzlem üzerinde $0 \leq x \leq 1$ ve $0 \leq y \leq 1$ koşullarını sağlayan noktalar seçerler. Rastgele nokta seçimi işlemini kendi paylarına düşen miktarda tekrarlar ve her seferinde seçilen noktanın dairenin içine mi yoksa dışına mı isabet ettiğini hesaplarlar. Her proses, dairenin içine isabet eden noktaların sayısını tutan bir sayaç değişkeni kullanır.
4. PiCalculation programında, 0 numaralı proses “ana” proses olarak belirlendiğinden, diğer tüm prosesler işlerini bitirdiklerinde hesapladıkları sayaç değerini ona iletirler. Sayaç değerleri birer tamsayı nesnesi (java.lang.Integer) içerisinde ana prosese gönderilir. Ana proses, kendi sayaç değeri ile diğer proseslerden gelen sayaç değerlerini toplar ve Denklem 3.3’e göre pi sayısının yaklaşık değerini hesaplar.
5. Ana proses hesaplanan yaklaşık değeri istemciye bildirir.
6. Prosesler, istemciye görevlerini tamamladıklarını bildirir ve paralel iletişim ortamından çıkarlar.

Sonuç olarak, rastgele seçilecek ve durumu değerlendirilecek olan toplam nokta adedi prosesler arasında bölüşüldüğünden, hesaplama işleminin daha kısa sürede tamamlanması mümkün olur.

3.2.2.2 Isı Denklemi Uygulaması

Isı denklemi, ortamdaki sıcaklığın zamanla değişimini tanımlayan bir diferansiyel denklemdir [23]. Denklemin basitleştirilmiş formunda (Denklem 3.4), ele alınan ortam eşit büyüklükteki bölgelere ayrılır. Bir bölgenin içerisindeki sıcaklığın her noktada eşit olduğu ve yalnızca birbirine komşu olan bölgeler arasında ısı alışverişi yaşandığı kabul edilir. Bir bölgenin doğusunda, batısında, kuzeyinde ve güneyinde yer alan bölgeler onun komşularıdır (Şekil 3.9).



Şekil 3.9 : Düzlem üzerinde (x,y) koordinatlarındaki bölgenin komşuları

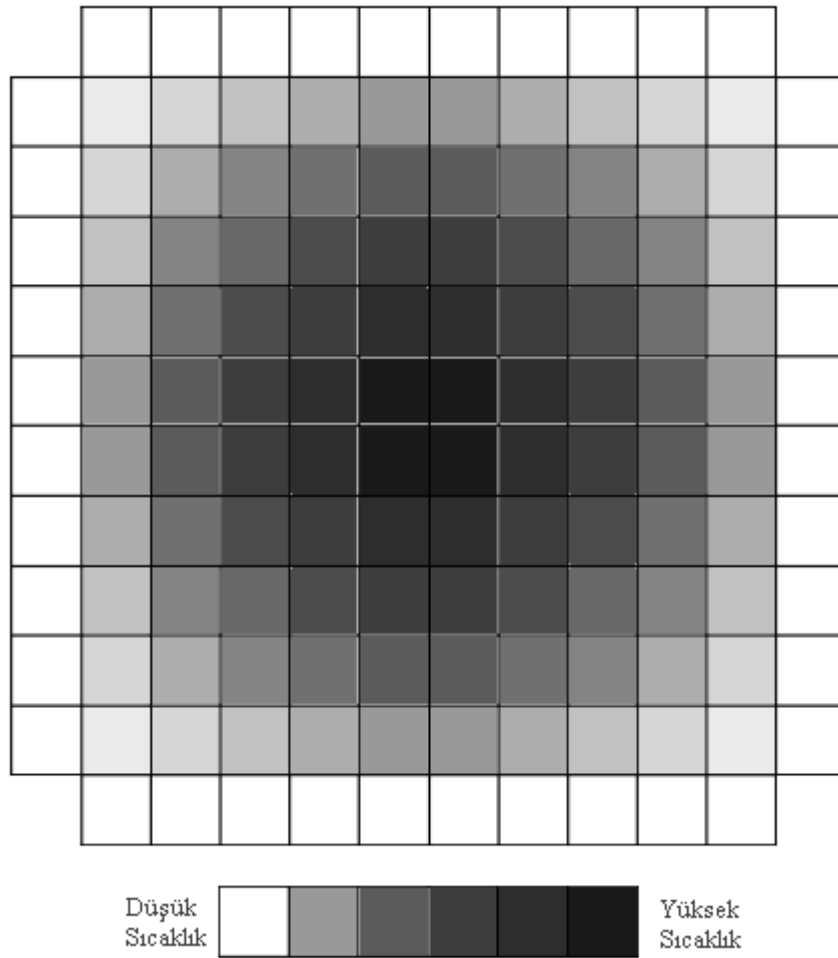
Birim zamanda meydana gelen ısı alışverişi sonucunda, bölgelerin sahip olacağı yeni sıcaklık değerleri Denklem 3.4'e göre hesaplanır [4].

$$S_{x,y} = S_{x,y} + K_x * (S_{x+1,y} + S_{x-1,y} - 2 * S_{x,y}) + K_y * (S_{x,y+1} + S_{x,y-1} - 2 * S_{x,y}) \quad (3.4)$$

Denklemden yer alan $S_{x,y}$ terimi (x,y) koordinatlarında bulunan bölgenin sıcaklığını ifade eder. K_x ve K_y terimleri ise, sırasıyla, x ve y boyutlarının hesaplamadaki ağırlıklarını ifade eden sabitlerdir. Bu denklemden faydalanılarak bir ortamdaki sıcaklığın belli bir zaman dilimi sonundaki durumu hesaplanabilir. Ancak, belirlenen zaman dilimi ve ortamın boyutları büyüdükçe sonuca ulaşmak için harcanan zaman

uzayacaktır. Bu noktada, paralel programlama tekniğinden faydalanarak, sırasal programlama (sequential programming) tekniği uygulandığında harcanan süreden daha kısa sürede sonuca ulaşmak mümkündür.

Ek 1’de yer verilen “HeatEquation” programı Denklem 3.4’ten yola çıkarak, PPPJ kütüphanesinin de yardımıyla, bu hesaplamanın nasıl daha kısa sürede yapılabileceğini göstermektedir. Söz konusu programda, ele alınan ortamdaki sıcaklığın sınır bölgelerinde her zaman 0 derece olduğu ve merkeze yaklaştıkça yükseldiği kabul edilmiştir (Şekil 3.10).

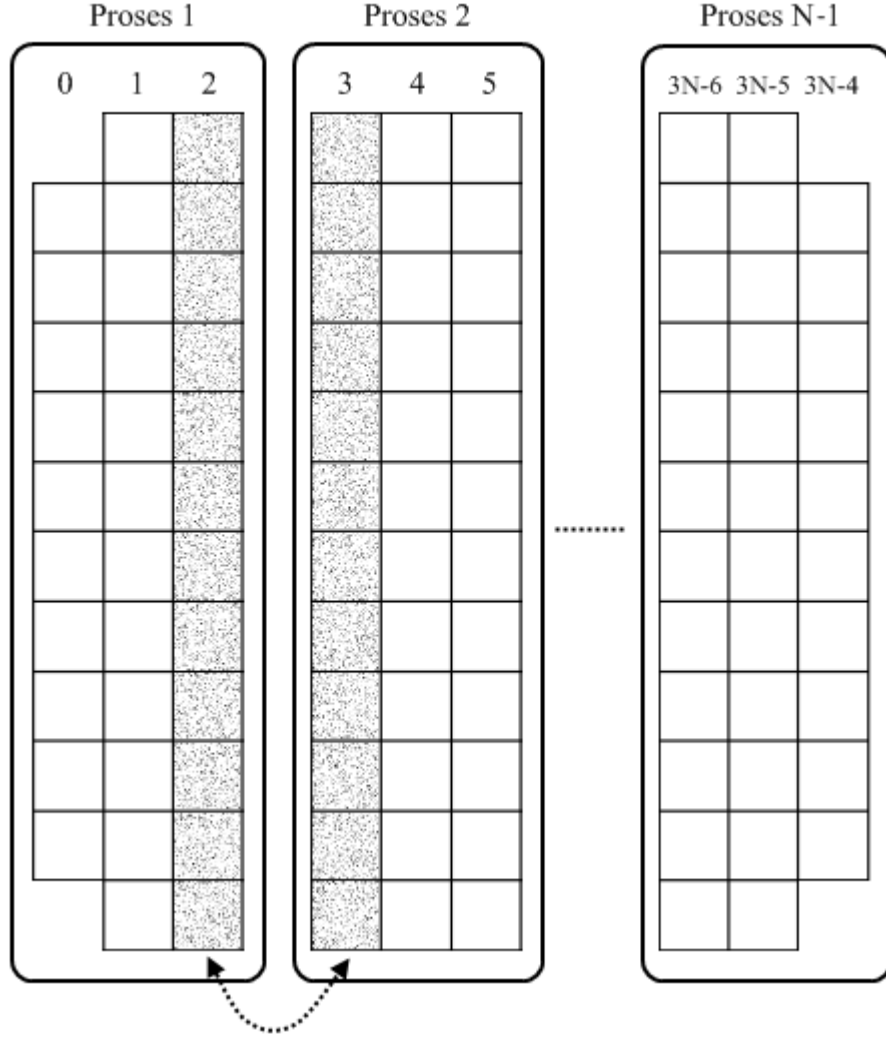


Şekil 3.10 : Sıcaklığın başlangıçtaki dağılımı

HeatEquation programını yürüten prosesler sırasıyla şu işlem adımlarını gerçekleştirirler:

1. Prosesler paralel iletişim ortamına dahil olurlar ve ortam hakkında bilgi edinirler. Kendi sıra numaralarını ve ortamdaki toplam proses sayısını öğrenirler.
2. 0 sıra numaralı proses, diğer bir deyişle ana proses, ele alınan ortamı bölgelere ayıran ızgarayı (grid) oluşturur.
3. Ana proses, ızgarayı sütun bazında bölümleyerek, kendisi dışında kalan prosesler (işçi prosesler) arasında paylaştırır (Şekil 3.11).
4. İşçi prosesler, ızgaranın kendi paylarına düşen parçası üzerindeki sıcaklık değişimini hesaplarlar. Bunu yaparken, ilgilendikleri ızgara parçasının, solundaki ve/veya sağındaki parça üzerinde çalışan proses(ler) ile iletişim kurarlar. Örneğin, Şekil 3.11’de görülen 2 numaralı proses, ızgaranın kendisine ait olan parçasında bulunan 3 numaralı sütun üzerindeki sıcaklık değişimini hesaplamak için, 2 numaralı sütun üzerindeki sıcaklık değerlerine ihtiyaç duyar. Benzer şekilde, 1 numaralı proses, kendisine ait olan 2 numaralı sütun üzerindeki sıcaklık değişimini hesaplamak için 3 numaralı sütun üzerindeki sıcaklık değerlerine ihtiyaç duyar. Sonuç olarak bu iki proses, her hesaplamadan önce, ilgili sütunlar üzerindeki güncel sıcaklık değerlerini bir “HeatGrid” nesnesi içerisinde birbirleriyle paylaşırlar.
5. İşçi prosesler, kendi paylarına düşen ızgara parçası üzerinde sıcaklığın zamanla değişimini hesapladıktan sonra, elde ettikleri son sıcaklık değerlerini, yine bir HeatGrid nesnesi içerisinde ana prosese gönderirler.
6. Ana proses diğer proseslerden gelen ızgara parçalarını birleştirir ve oluşan bütün üzerindeki nihai sıcaklık değerlerini, kullanıcıya sunulmak üzere, bir dosyaya kaydeder.
7. Tüm prosesler, görevlerini tamamladıklarını istemciye bildirirler ve paralel iletişim ortamından çıkarlar.

Sonuç olarak, ele alınan ortamın farklı bölgelerindeki sıcaklık değişimi, farklı prosesler tarafından eş zamanlı olarak hesaplandığından, işlemin daha kısa sürede tamamlanması mümkün olur.



Şekil 3.11 : Prosesler arası paylaşım ve iletişim

Ek 1’de HeatEquation programının bir de görsel sürümüne yer verilmiştir. “VisualHeatEquation” adı verilen bu program, temelde HeatEquation programıyla aynı işi yapmakta, ayrıca ondan farklı olarak sıcaklığın zamanla değişimini temsili ortam üzerinde gözlemleme olanağı sunmaktadır.

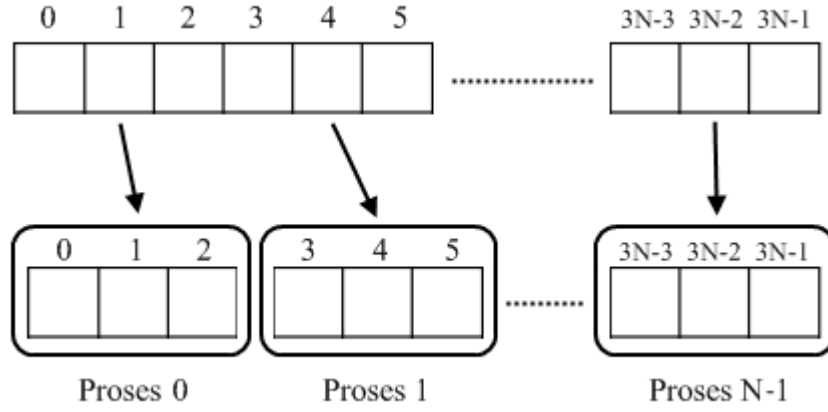
3.2.2.3 Sıralama Uygulaması

Sıralama algoritmaları, bir listeyi oluşturan öğeleri, aralarında belli bir ilişki (büyüklük-küçüklük, öncelik-sonralık vb.) kurarak yeniden yerleştirmekte kullanılan algoritmalar. Günümüze dek çok sayıda sıralama algoritması geliştirilmiş olmasına karşın bu alanda yapılan çalışmalar devam etmektedir. Bu durumun başlıca nedenlerinden biri de başarıyı daha yüksek algoritmalar oluşturma isteğidir. Bir

algoritmanın başarımı değerlendirilirken zaman ve alan karmaşıklığı gibi ölçütler göz önünde bulundurulur.

Özellikle zaman karmaşıklığı açısından başarımı daha yüksek algoritmalar geliştirmek için paralel programlama tekniğinden faydalanılabilir. Ek 1’de yer alan “ParallelSort” programı PPPJ kütüphanesi kullanılarak bunun nasıl yapılabileceğini göstermektedir. ParallelSort programını yürüten prosesler sırasıyla şu işlem adımlarını gerçekleştirirler:

1. Prosesler paralel iletişim ortamına dahil olurlar. Kendi sıra numaralarını ve ortamdaki toplam proses adedini öğrenmek suretiyle ortam hakkında bilgi edinirler.
2. Ana proses olarak belirlenen 0 numaralı proses rastgele sayılar seçerek sıralanacak olan diziyi (listeyi) oluşturur.
3. Ana proses, sıralanacak olan diziyi ortamdaki toplam proses adedince bölümleyerek işçi prosesler arasında paylaştırır (Şekil 3.12). Dizinin bir bölümünü de kendisi sıralamak üzere ayırır.
4. Ana proses, kullanılacak olan sıralama yöntemi (kabarcık, ekleme ya da seçme) hakkında işçi prosesleri bilgilendirir.
5. Ana proses, dizinin kendisi için ayırdığı bölümünü sıralar ve işçi proseslerden gelecek diğer sıralanmış bölümleri beklemeye koyulur.
6. Dizinin kendi paylarına düşen bölümünü ve kullanılacak olan sıralama yöntemini öğrenen işçi prosesler, sıralama işlemini gerçekleştirdikten sonra, sıralanmış dizi bölümlerini ana prosese geri gönderirler.
7. Ana proses, elde ettiği sıralı dizi bölümlerini uygun şekilde birleştirerek nihai sıralı diziyi oluşturur ve kullanıcıya sunulmak üzere bir dosyaya kaydeder.
8. Tüm prosesler, görevlerini tamamladıklarını istemciye bildirirler ve paralel iletişim ortamından çıkarlar.



Şekil 3.12 : Prosesler arası paylaşım

3.2.2.4 Matris Çarpımı

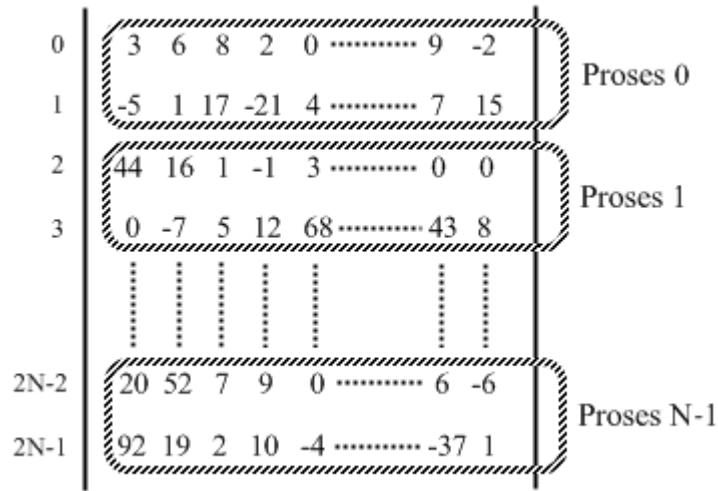
Matris çarpımı, bir matrisin sayısal bir değer ile ya da başka bir matris ile çarpılması işlemidir [24]. Çarpanlardan her ikisinin de birer matris olması durumunda, birinci matrisin sütun sayısı ikinci matrisin satır sayısına eşit olmak zorundadır. İki matrisin çarpımı işlemi Denklem 3.5’te görülen formül ile tanımlanmıştır.

$$C_{ij} = \sum_{r=1}^n A_{i,r} * B_{r,j} \quad (3.5)$$

Denklemde çarpım işlemine tabi tutulan matrisler A ve B sembolleriyle, sonuç matrisi ise C sembolüyle ifade edilmiştir. Denklemde yer alan $A_{i,r}$ terimi A matrisinin i numaralı satırında ve r numaralı sütununda bulunan elemanı temsil etmektedir. Benzer şekilde, $B_{r,j}$ terimi de B matrisinin r numaralı satırının j numaralı sütununda bulunan elemanı temsil eder. n sembolü ise, birbirine eşit olan, birinci matrisin sütun sayısını ve ikinci matrisin satır sayısını ifade eder. Söz konusu denklemden de anlaşılabileceği üzere, matris çarpımı yoğun hesaplama gerektiren, karmaşıklığı yüksek bir işlemdir. Buna karşın, paralel programlama tekniğinden faydalanarak başarımların artırılması sağlanabilir. Ek 1’de yer verilen “MatrixMultiplication” programı, PPPJ kütüphanesinden faydalanılarak bu hesaplamaların nasıl yapılabilirliğini göstermektedir. Söz konusu programda, çarpma işlemine giren matrislerden birinin prosesler arasında paylaştırılması suretiyle sonuç matrisinin farklı bölümlerinin aynı anda farklı prosesler tarafından hesaplanması

hedeflenmiştir (Şekil 3.13). MatrixMultiplication programını yürüten prosesler sırasıyla şu işlem adımlarını gerçekleştirirler:

1. Prosesler paralel iletişim ortamına dahil olurlar ve ortam hakkında bilgi edinirler. Kendi sıra numaralarını ve ortamda toplam kaç proses bulunduğunu öğrenirler.
2. Ana proses olarak belirlenen 0 numaralı proses, içeriklerini ilgili dosyalardan okumak suretiyle, çarpma işlemine girecek olan matrisleri oluşturur.
3. Ana proses, ilk matrisi satır bazında ve ortamdaki proses adedince bölümleyerek işçi prosesler arasında paylaştırır. Ana prosesin kendisi de doğrudan çarpma işlemine katılacağından, matrisin bir bölümünü kendisi için ayırır.
4. Ana proses, çarpma işlemine girecek ikinci matrisi bir bütün halinde tüm işçi proseslere gönderir.
5. Ana proses, çarpma işleminin kendisine ait bölümünü tamamlar ve işçi proseslerden gelecek sonuçları beklemeye başlar.
6. Ana procesten ilk matrisin kendi paylarına düşen bölümünü ve ikinci matrisi alan işçi prosesler çarpma işlemini yaparlar ve hesapladıkları matrisi ana prosese gönderirler.
7. Ana proses, elde ettiği matris parçalarını birleştirerek nihai sonuç matrisini oluşturur ve kullanıcıya sunulmak üzere bir dosyaya kaydeder.
8. Tüm prosesler, görevlerini tamamladıklarını istemciye bildirirler ve paralel iletişim ortamından çıkarlar.



Şekil 3.13 : Örnek matrisin prosesler arası paylaşımı

3.3 Uygulamaların Çalıştırılması

PPPJ kütüphanesinden faydalanılarak oluşturulmuş kullanıcı programlarının (user program) çalıştırılabilmesi için öncelikle istemci ve/veya sunucu olarak çalışan bilgisayarların varlığına ihtiyaç duyulur.

Sunucu olarak görev yapan bilgisayarlarda, kütüphaneye ait “RunServerAgent” uygulaması çalıştırılır. RunServerAgent uygulamasının temel işlevi sunucu etmenini harekete geçirmektir. Sunucu bilgisayar, söz konusu uygulama marifetiyle, istemcilerden gelen talepleri dinler ve yanıtlar. Uygulamayı çalıştırmak için kütüphanenin kurulu olduğu dizine gidilerek aşağıdaki komutun verilmesi yeterlidir.

```
> java server.RunServerAgent
```

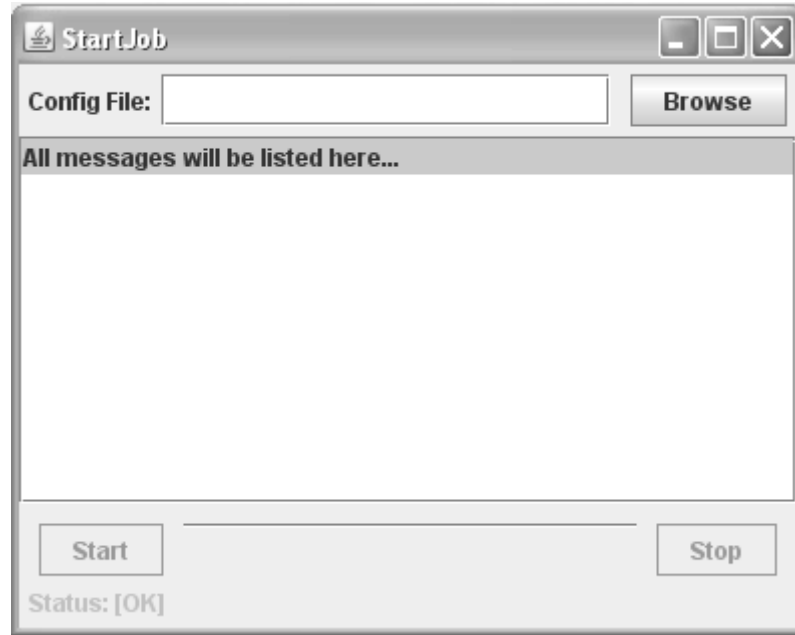
İstemci tarafında ise, kullanıcı programını sunuculara yüklemek için “StartJob” isimli kütüphane uygulaması kullanılır. Söz konusu uygulamayı çalıştırmak için kütüphanenin kurulu olduğu dizine gidilerek aşağıdaki komut verilir.

```
> java client.StartJob
```

StartJob uygulaması, kullanıcı programını, sunuculara yüklenecek yeni bir “iş” olarak değerlendirir ve bu işi kullanıcının tercihlerini göz önünde bulundurarak sistemdeki sunuculara dağıtır. Kullanıcılar, tercihlerini konfigürasyon dosyaları aracılığıyla bildirirler. Konfigürasyon dosyaları, yürütülecek olan kullanıcı

programlarına ilişkin detayları içerir. Bir sonraki kısımda konfigürasyon dosyalarının içeriğine ilişkin ayrıntılı bilgi verilecektir.

StartJob uygulaması çalıştırıldığında Şekil 3.14'te görülen arayüz ekrana gelir. Kullanıcı öncelikle yürütülecek olan paralel programa ilişkin konfigürasyon dosyasının yerini belirtir. Sonrasında “Start” (Başlat) düğmesine basarak paralel uygulamanın çalıştırılması sürecini başlatır.



Şekil 3.14 : İstemci uygulamanın arayüzü

StartJob uygulaması, kullanıcının başlatma düğmesine basmasıyla birlikte, konfigürasyon dosyasında belirtilen stratejiyi izleyerek, kullanılacak olan sunucuları belirler ve bu sunucular üzerinde gerekli kaynakların ayrılması için girişimlerde bulunur. Bu sırada, bir ya da daha fazla sunucudan olumsuz yanıt alınması gibi beklenmedik bir durum meydana gelirse kullanıcı uyarılarak, işleme devam etmek isteyip istemediği sorulur. Kullanıcının işleme devam etmek istememesi durumunda, o ana kadar yapılan tüm işlemler geri alınır ve çalışma sonlandırılır. Benzer şekilde, kullanıcı programının sunuculara yüklenmesi aşamasında ya da çalışması esnasında bir sistem hatası oluşması durumunda, kullanıcı bir uyarı mesajı ile bilgilendirilir. Bizzat istemciden ya da sunuculardan gelen bilgi ve uyarı mesajları StartJob uygulamasına ait arayüzün (Şekil 3.14) orta kısmındaki kutucukta listelenir. Arayüzdeki “Stop” (Durdur) düğmesi, kullanıcı programını, bitmesini beklemeden sonlandırmak için kullanılır. Ne var ki, çalışmakta olan bir kullanıcı programının

durdurulması, işlemekte olduğu verileri tutarsız durumda bırakabileceğinden güvenli bir işlem değildir. Bunun yanı sıra, prosesler arasında kapatılamamış bağlantıların kalmasına da neden olabilir. Bu durum, daha sonra aynı etmenler üzerinde koşturulacak kullanıcı programlarının çalışması esnasında sorun oluşturabileceğinden, ilgili sunucu etmenlerinin (uygun görülen bir zamanda) yeniden başlatılmasında fayda vardır.

3.3.1 Konfigürasyon Dosyaları

Konfigürasyon dosyaları, yürütülecek olan kullanıcı programlarına, diğer bir deyişle yapılacak işlere, ilişkin detayları barındıran, XML formatındaki dosyalardır. İçerdikleri bilgiler şöyle sıralanabilir:

- Yapılacak olan işin ismi.
- İşi yerine getirmekte kullanılacak toplam proses sayısı.
- Sunucular arasında seçim yaparken kullanılacak olan yöntem.
- Eğer sunucu seçimi bizzat kullanıcı tarafından yapılacak ise, buna ilişkin detayları içeren yardımcı konfigürasyon dosyasının bulunduğu yer.
- Kullanıcı programını (yapılacak işi) oluşturan Java sınıflarının isimleri, sınıf dosyalarının (*.class) bulunduğu yerler ve ana sınıfın ismi.
- Prosesler arasındaki nesne alışverişi sırasında verilerin sıkıştırılmış formatta aktarımının tercih edilip edilmediği.

Şekil 3.15’te daha önce bahsedilen PiCalculation isimli kullanıcı programının çalıştırılması için kullanılabilecek örnek bir konfigürasyon dosyasına yer verilmiştir.

```

<?xml version="1.0" encoding="UTF-8"?>
<new_job>
  <name>pi hesaplama</name>
  <server_selection>
    <number_of_processes>6</number_of_processes>
    <type>PERFORMANCE</type>
    <manual_selection_file></manual_selection_file>
  </server_selection>
  <classes primary_class="user_apps.PiCalculation">
    <class directory="C:\PPPJ\user_apps">user_apps.PiCalculation</class>
  </classes>
  <compression>YES</compression>
</new_job>

```

Şekil 3.15 : Örnek konfigürasyon dosyası

Örnek dosyada görüldüğü gibi, yapılacak işe “pi hesaplama” ismi verilmiştir. Söz konusu işin toplam 6 proses tarafından aynı anda yürütülmesi ve bu 6 prosesin bulunacağı sunucuların, başarımlar (performance) kriterine göre belirlenmesi istenmiştir. Ayrıca, kullanıcı programının yalnızca “PiCalculation” isimli bir Java sınıfından oluştuğu belirtilmiş ve adı geçen sınıfa ait derlenmiş dosyanın, PiCalculation.class dosyasının, bulunduğu dizin belirtilmiştir. Örnekte de görüldüğü gibi, bir Java paketi (package) içerisinde tanımlı olan sınıfların isimlerinin, ait oldukları paketin ismi ile birleştirilerek belirtilmesi gereklidir. Şekil 3.15’te, PiCalculation sınıfının “user_apps” isimli bir Java paketi içerisinde tanımlı olduğu görülmektedir. Son olarak, örnek konfigürasyon dosyasında, nesne alışverişi sırasında verilerin sıkıştırılmış formatta iletilmesinin istendiği belirtilmiştir.

Sunucu seçimi için üç farklı yöntem izlenebilir. Seçim tamamen kullanıcıya bırakılabileceği gibi, sistemdeki sunucuların başarımlarının ya da yoğunluğunun değerlendirilmesi suretiyle istemci uygulama tarafından da gerçekleştirilebilir. Sunucu seçiminin tamamen kullanıcı tarafından gerçekleştirilmesi durumunda, konfigürasyon dosyasındaki “type” etiketine “MANUAL” değeri verilir. Ayrıca “manual_selection_file” etiketinde, kullanıcı tarafından seçilen sunuculara ilişkin detayları içeren yardımcı XML dosyasının yeri belirtilir. Bu ikinci dosyada, hangi sunucuda kaç adet proses çalıştırılmak istendiği bilgisi de yer aldığından, konfigürasyon dosyasındaki “number_of_processes” etiketi anlamını yitirir ve görmezden gelinir. Seçim işleminin, istemci program tarafından, sunucuların başarımlarına göre yapılması isteniyorsa, type etiketine “PERFORMANCE” değeri

verilir. Seçimin, sunucuların yoğunluğu göz önünde bulundurularak yapılması istendiğinde ise bu etiket “OCCUPANCY” değerini alır. Bir sonraki kısımda sunucu seçimi konusu daha ayrıntılı olarak ele alınacaktır.

Konfigürasyon dosyasındaki “classes” etiketinin “primary_class” isimli özniteliği, kullanıcı programının başlangıç noktası durumundaki “main” metodunu içeren Java sınıfının, bir başka deyişle ana sınıfın, adını belirtir.

Konfigürasyon dosyasındaki “compression” etiketinde, veri aktarımı sırasında sıkıştırma işlemi yapılmasının istenip istenmediği belirtilir. Bu etikete “YES” değeri verildiğinde, ilgili kullanıcı programının çalıştırılması esnasında veriler (nesneler) her zaman sıkıştırılarak aktarılır, “NO” değeri verildiğinde ise sıkıştırma işlemi yapılmaz.

Konfigürasyon dosyaları el ile (manual) oluşturulabileceği gibi, PPPJ kütüphanesinin sahip olduğu “JobConfigCreator” isimli uygulama yardımıyla otomatik olarak da oluşturulabilir. Bu uygulamanın kullanımına ilişkin detaylı bilgi “Araçlar” bölümünde verilmektedir.

3.3.2 Sunucu Seçimi

Bir önceki kısımda belirtildiği gibi, kullanıcı programlarının yürütüleceği sunucuların seçimi konusunda üç farklı yöntem benimsenebilir:

1. Seçim bizzat kullanıcı tarafından yapılır.
2. İstemci uygulama, sunucuların başarımını ölçüp birbirleriyle kıyaslayarak seçimi yapar.
3. İstemci uygulama, sunucuların yoğunluğunu ölçüp birbirleriyle kıyaslayarak seçimi yapar.

Kullanıcı, seçimi kendisi yapmak istediğinde, seçtiği sunuculara ilişkin bilgileri içeren bir XML dosyası (Şekil 3.16) oluşturur ve bu dosyanın yerini konfigürasyon dosyasında belirtir.

```
<?xml version="1.0" encoding="UTF-8"?>
<select>
  <agent>
    <name>bornova</name>
    <process_count>2</process_count>
  </agent>
  <agent>
    <name>konak</name>
    <process_count>1</process_count>
  </agent>
  <agent>
    <name>buca</name>
    <process_count>1</process_count>
  </agent>
</select>
```

Şekil 3.16 : Kullanıcı tarafından seçilmiş sunucuları içeren örnek dosya

Şekil 3.16’da görüldüğü gibi, söz konusu dosyada, seçilen her bir sunucunun (sunucu etmeninin) ismi ve üzerinde kaç adet proses çalıştırılmak istendiği belirtilir.

Sunucu seçimi başarıma göre yapılmak istendiğinde, istemci uygulama sistemdeki tüm aktif sunuculara birer “PerformanceEvaluator” nesnesi göndererek başarımlarını ölçer. Söz konusu nesneyi alan sunucular, onun başarımlar ölçücü metodunu çağırırlar. Bu metod bir matris çarpımı işlemi gerçekleştirir ve bu işlem için harcanan süreyi PerformanceEvaluator nesnesine kaydeder. Daha sonra sunucular, güncelledikleri PerformanceEvaluator nesnesini istemciye geri gönderirler. İstemci uygulama, sunuculardan gelen güncel PerformanceEvaluator nesnelerini toplar ve matris çarpımı için harcadıkları süreleri bakarak başarımlarını değerlendirir. Söz konusu işlem için harcanan süre ne kadar kısa olursa başarımın o kadar yüksek olduğu kabul edilir. Kullanıcı programının koşturulacağı sunucular seçilirken, başarımı yüksek olan sunuculara öncelik verilir.

Sunucu seçimi yoğunluğa göre yapılmak istendiğinde ise, istemci uygulama sistemdeki tüm aktif sunuculara birer “OccupancyEvaluator” nesnesi göndererek, onların yoğunluğunu, diğer bir deyişle yeni bir işi “hazmetme kapasitelerini” ölçer. Bu ölçüm, sunucuların hızı ve üzerlerindeki iş yükü göz önünde bulundurularak yapılır. İş yükü, belli bir anda sunucu etmeni üzerinde etkin durumda bulunan proses adedinin, etmen üzerinde aynı anda çalışabilecek azami proses adedine oranıyla ifade

edilir (Denklem 3.6). Yoğunluk ise sunucunun iş yüküyle doğru, hızıyla ters orantılıdır (Denklem 3.7).

$$\text{İş yükü} = \text{Etkin proses adedi} / \text{Azami proses adedi} \quad (3.6)$$

$$\text{Yoğunluk} = \text{İş yükü} / \text{Hız katsayısı} \quad (3.7)$$

Kullanıcı programlarının koşturulacağı sunucular seçilirken yoğunluğu düşük olan sunuculara öncelik verilir.

3.4 Araçlar

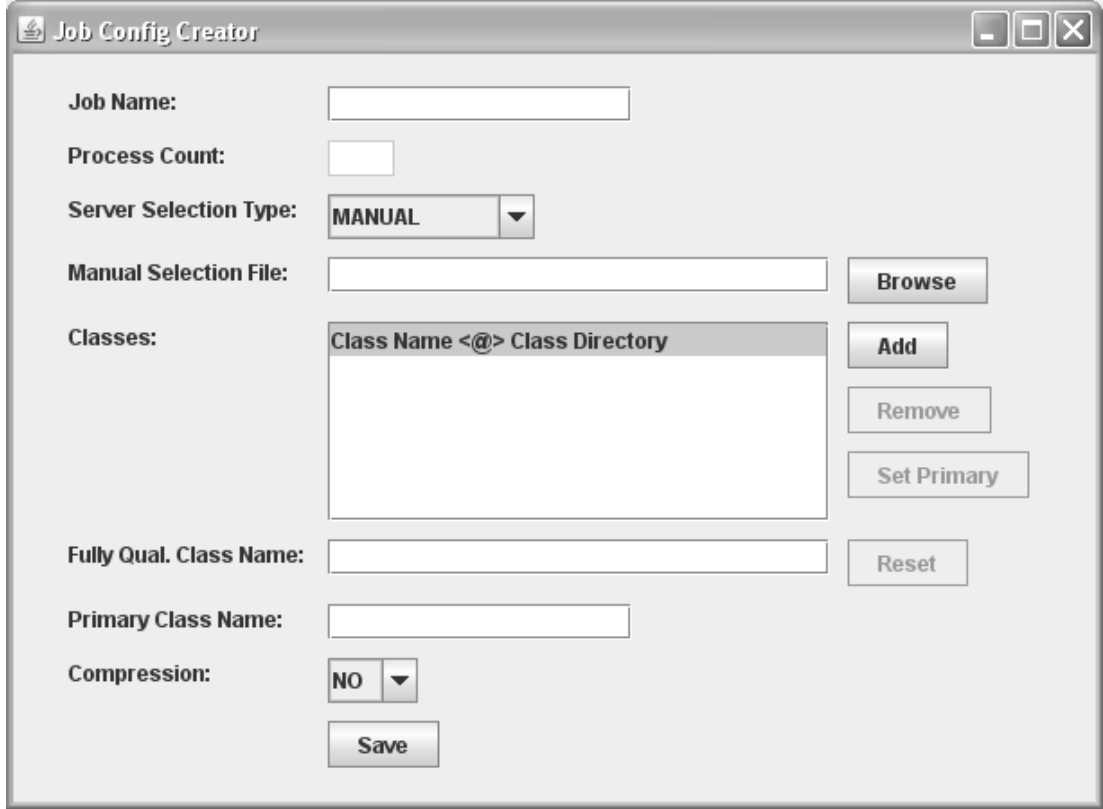
Bu bölümde, PPPJ kütüphanesi tarafından sunulan iki araç tanıtılacaktır. Bunlardan ilki, kullanıcı programlarının yürütülebilmesi için gerekli konfigürasyon dosyalarının oluşturulması sürecini, ikincisi ise sistemdeki sunucuların etkinlik durumundaki olası değişikliklerden diğer sunucuları haberdar etmesi sürecini kolaylaştırmaktadır.

3.4.1 Konfigürasyon Dosyası Oluşturucu

PPPJ kütüphanesine ait JobConfigCreator programı konfigürasyon dosyalarının oluşturulmasında kullanıcılara kolaylık sağlar. Söz konusu programı çalıştırmak için kütüphanenin kurulu olduğu dizine gidilerek aşağıdaki komut verilir:

```
> java tools.JobConfigCreator
```

Programın kullanıcı arayüzü Şekil 3.17’de gösterilmektedir. Bu arayüzde kullanıcıdan sırasıyla, yürütülecek olan işin ismini, çalıştırılacak proses adedini, sunucu seçim yöntemini, sunucu seçimini kendisi yapacaksa ilgili (yardımcı) XML dosyasının yerini, kullanıcı programını (yapılacak işi) oluşturan Java sınıflarına ilişkin bilgileri, ana sınıfın ismini ve veri aktarımı sırasında sıkıştırma yapılmasını tercih edip etmediğini belirtmesi istenir.



Şekil 3.17 : Konfigürasyon dosyası oluşturan programın arayüzü

Arayüzde, kullanılacak proses adedinin girildiği “Process Count” alanı ile kullanıcı tarafından seçilen sunuculara ilişkin bilgileri içeren dosyanın yerinin girildiği “Manual Selection File” alanı, sunucu seçim yöntemine bağlı olarak etkinleşir ya da etkinliğini yitirir. Kullanıcı, sunucu seçimini bizzat kendisi yapmak istediğinde, tercihlerini belirttiği XML dosyasında (Şekil 3.16) yalnızca sunucu isimlerine değil, hangi sunucuda kaç adet proses çalıştırılmasını istediği bilgisine de yer verdiğinden, arayüzdeki Process Count alanı işlevini yitirir. Bunun bir sonucu olarak da etkinliğini yitirir. Buna karşın Manual Selection File alanı etkinlik kazanır. Sunucu seçimi başarıma ya da yoğunluğa göre yapılmak istendiğinde ise Process Count alanı etkin konuma geçerken, işlevini yitiren Manual Selection File alanı etkinliğini kaybeder.

Kullanıcı programını oluşturan Java sınıflarına ilişkin bilgilerin, arayüze, dolayısıyla da konfigürasyon dosyasına eklenmesi için “Add” (Ekle) düğmesi kullanılır. Bu düğmeye basıldığında ekrana gelen pencere yardımıyla, gerekli sınıf dosyaları bilgisayar üzerinde bulunarak, arayüzdeki ilgili listeye kolayca eklenebilir. Arayüzdeki “Primary Class Name” alanında ana sınıfın ismi belirtilir. Bu işlem, ana sınıfın isminin doğrudan ilgili kutucuğa yazılması yoluyla yapılabileceği gibi,

“Classes” alanındaki listeye eklenen sınıflardan birinin seçilip, “Set Primary” düğmesine basılmasıyla da yapılabilir. “Classes” alanındaki listeden bir sınıf çıkarılmak istendiğinde “Remove” (Çıkar) düğmesi kullanılır.

Konfigürasyon dosyasında belirtilen sınıf isimlerinin tam nitelendirilmiş (fully qualified) olması gerektiğinden, arayüzde sınıf isimleri girilirken buna göre hareket edilmelidir. Bir Java sınıfının tam nitelendirilmiş ismi, (eğer varsa) içerisinde tanımlı olduğu Java paketinin ismi ile kendi isminin birleşiminden oluşur. Classes alanındaki listede belirtilen sınıf isimleri, gerektiğinde “Fully Qual. Class Name” alanı kullanılarak düzeltiler. Bunun için öncelikle söz konusu listede ismi düzeltilmek istenen sınıfın bulunduğu satır seçilir. Sonra, “Fully Qual. Class Name” alanındaki kutucuğa sınıfın tam nitelendirilmiş ismi yazılır ve “Reset” (Yeniden Ayarla) düğmesine basılır.

Kullanıcı, Şekil 3.17’de görülen arayüzde istenen bilgileri girdikten sonra “Save” (Kaydet) düğmesine basarak, konfigürasyon dosyasını oluşturur. Oluşturulan XML dosyası, “Job Name” alanında belirtilen isimle, JobConfigCreator programının bulunduğu dizine kaydedilir.

3.4.2 Etmen Durumu Bildirici

PPPJ kütüphanesinin kullanıldığı bir bilgisayar sisteminde, belli bir anda etkin durumda bulunan sunucu etmenlerinin listesi AgentWorld isimli bir XML dosyasında tutulur. Bu dosya, sistemde sunucu ve/veya istemci olarak görev yapan tüm bilgisayarlarda bulunur. Her bir sunucu etmeni için aşağıdaki bilgileri barındırır:

- Sunucu etmeninin ismi.
- Sunucu etmeninin üzerinde koştığı bilgisayarın IP numarası.
- Sunucu etmeninin istemcilerden gelecek talepleri almak için dinlediği kapı (port) numarası.
- Sunucu etmeninin üzerinde koştığı bilgisayarın hız katsayısı.

Bilgisayarların hız katsayıları, donanımsal özellikleri göz önünde bulundurularak sistem yöneticisi tarafından belirlenir. Hız katsayısının değeri 1 ile 3 arasında değişir. Yavaş bir bilgisayar için 1, ortalama bir bilgisayar için 2, hızlı bir bilgisayar için ise 3 değerini alır. Şekil 3.18’de örnek bir AgentWorld dosyası gösterilmektedir.

```

<?xml version="1.0" encoding="UTF-8"?>
<world>
  <agent>
    <name>buca</name>
    <ip>10.0.1.50</ip>
    <port>10067</port>
    <speed>2</speed>
  </agent>
  <agent>
    <name>bornova</name>
    <ip>10.0.1.59</ip>
    <port>10052</port>
    <speed>3</speed>
  </agent>
  <agent>
    <name>karsiyaka</name>
    <ip>10.0.1.55</ip>
    <port>10071</port>
    <speed>3</speed>
  </agent>
  <agent>
    <name>konak</name>
    <ip>10.0.1.51</ip>
    <port>10053</port>
    <speed>1</speed>
  </agent>
</world>

```

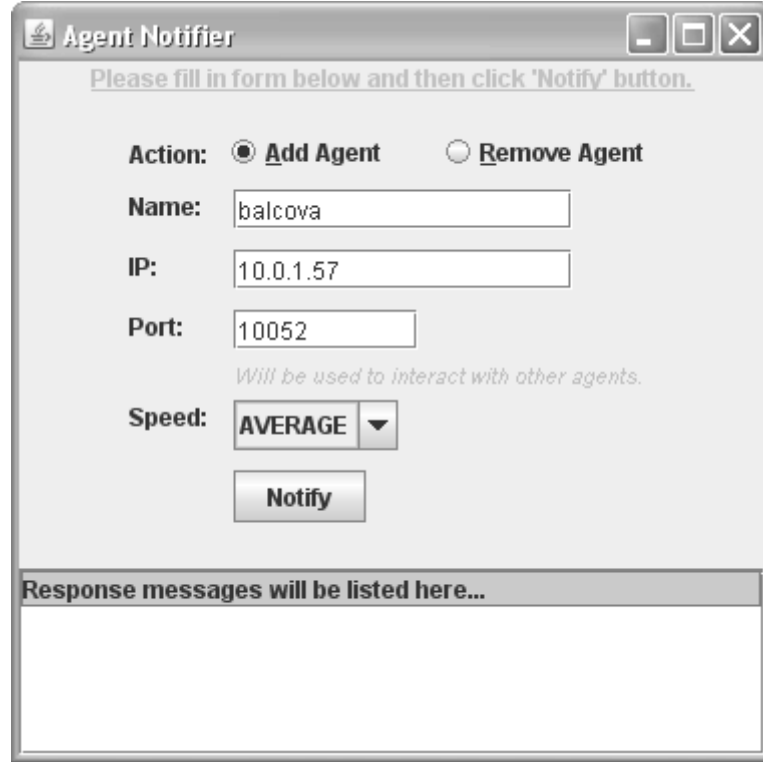
Şekil 3.18 : Örnek AgentWorld dosyası

Söz konusu dosyanın güncellenmesi işlemi, gerektiğinde sunucu etmeni tarafından gerçekleştirilir. Sisteme yeni bir sunucu eklenmesi ya da varolan bir sunucunun sistemden çıkarılması durumunda, diğer sunucu etmenlerine birer “AgentNotification” mesajı (nesnesi) gönderilerek durum hakkında bilgi verilir. Bu mesajı alan sunucu etmenleri AgentWorld dosyalarını duruma uygun şekilde güncellerler. AgentNotification mesajının gönderimi “AgentNotifier” isimli kütüphane programı kullanılarak yapılır. Bu programı çalıştırmak için kütüphanenin kurulu olduğu dizine gidilerek aşağıdaki komut verilir:

```
> java tools.AgentNotifier
```

Programın kullanıcı arayüzü Şekil 3.19’da gösterilmektedir. Buna göre, kullanıcıdan - bu kimsenin sistem yöneticisi olması beklenir - ilk olarak yapmak istediği işlemi seçmesi istenir. Eğer sisteme yeni bir sunucu etmeni eklenecekse “Add Agent” (Etmen Ekle), varolan bir sunucu etmeni çıkarılacaksa “Remove Agent” (Etmen

Çıkar) seçeneği işaretlenir. Daha sonra eklenecek ya da çıkarılacak sunucu etmenine ilişkin isim, IP numarası, kapı numarası ve hız düzeyi bilgileri girilir.

The image shows a Windows-style dialog box titled "Agent Notifier". At the top, there is a text instruction: "Please fill in form below and then click 'Notify' button." Below this, there are two radio buttons for "Action": "Add Agent" (which is selected) and "Remove Agent". Underneath, there are three text input fields: "Name:" with the value "balcova", "IP:" with the value "10.0.1.57", and "Port:" with the value "10052". Below the "Port" field, there is a small italicized text: "Will be used to interact with other agents." Then, there is a "Speed:" label followed by a dropdown menu currently showing "AVERAGE". At the bottom of the form area is a "Notify" button. Below the form area, there is a section with a grey header that says "Response messages will be listed here..." and a large empty white box for the response messages.

Şekil 3.19 : Etmen durumu bildirici programın kullanıcı arayüzü

Add Agent seçeneği işaretlendiğinde, AgentNotifier programı, üzerinde çalıştığı sunucunun sisteme eklenmek istendiğini varsayar ve eğer sunucu etmeninin konfigürasyon dosyası (ServerAgent.properties) daha önce oluşturulmuş ise, o dosyadaki bilgilerden de faydalanarak arayüzdeki isim, IP numarası ve kapı numarası alanlarını kendiliğinden doldurur. Benzer şekilde, Remove Agent seçeneği işaretlendiğinde, AgentNotifier programı, üzerinde çalıştığı sunucunun sistemden çıkarılmak istendiğini varsayar ve AgentWorld dosyasından sunucu etmeniyle ilgili bilgileri okuyarak arayüzdeki alanları doldurur. Kullanıcının sisteme eklemek ya da sistemden çıkarmak istediği sunucunun farklı olması durumunda, arayüzdeki alanlar bizzat kullanıcının kendisi tarafından yeniden doldurulur.

Söz konusu arayüzdeki tüm alanlar doldurulduktan sonra “Notify” (Bildir) düğmesine basılır. Böylelikle AgentNotification mesajı sistemdeki tüm sunucu etmenlerine gönderilmiş olur. Mesajı alan sunucu etmenleri, gerekli denetimleri yaptıktan sonra herhangi bir sorunla karşılaşmazlarsa kendi AgentWorld dosyalarını güncellerler ve işlemin başarıyla tamamlandığına dair yanıt dönerler. Denetimler

esnasında bir sorunla karşılaşmaları durumunda ise, uygun bir hata mesajı dönerler. Örneğin sistemde zaten var olan bir sunucu etmeni ismiyle, başka bir sunucu etmeni daha eklenmeye çalışılırsa hata oluşur. Çünkü sistemde aynı anda etkin durumda bulunan sunucu etmenlerinin isimleri eşsiz olmak zorundadır. Herhangi bir sorundan ötürü yeni sunucu etmenini tanımayan, diğer bir deyişle AgentWorld dosyalarına kaydetmeyen sunucu etmenlerinin bulunduğu bilgisayarlar istemci olarak kullanılmak istendiklerinde bu yeni etmene iş (kullanıcı programı) yüklemeleri mümkün olmayacaktır.

Sunucu etmenlerinin AgentNotification mesajına döndüğü cevaplar, Şekil 3.19’da görülen arayüzün alt kısmındaki kutucukta listelenir.

3.5 Günlük Tutma (Logging) Sistemi

PPPJ kütüphanesinde günlük tutma sistemi, Java için geliştirilmiş en popüler günlükleme yazılımı olan “log4j” [25] üzerine kurulmuştur. Bilgi, hata ve uyarı düzeyindeki tüm günlükleme istekleri bu yazılımdan faydalanılarak yerine getirilir.

Log4j yazılımı kendisine has konfigürasyon dosyalarından yararlanılarak yapılandırılır. PPPJ kütüphanesinde söz konusu dosyalar “configuration” dizini içerisinde yer alır. Adı geçen dizinde log4j yazılımıyla ilgili üç farklı konfigürasyon dosyası bulunur. Bunlardan “Log4jClient.properties” isimli olanı istemci uygulamanın, “Log4jServer.properties” isimli olanı sunucu etmeninin, “Log4jNotifier.properties” isimli olanı ise etmen durumu bildirici uygulamanın günlük tutma sistemini yapılandırmakta kullanılır. Bu dosyalar üzerinde değişiklik yapılarak, günlük çıktısının formatı, düzeyi, yazılacağı yer(ler) gibi birçok konuda düzenleme yapılabilir.

PPPJ kütüphanesinde günlük çıktıları, kullanıcı tarafından aksi belirtilmedikçe, işletmen uçbirimine (console) ve dosyalara yazılır. Varsayılan günlükleme dizini “log” dizinidir. Bu dizin içerisinde, sunucu, istemci ve etmen durumu bildirici uygulamalar için ayrı birer günlük dosyası tutulur. Sunucu etmenine ait günlük dosyasının varsayılan ismi “ServerAgent.log”, istemci uygulamaya ait günlük dosyasının varsayılan ismi “ClientAgent.log”, etmen durumu bildirici uygulamaya ait günlük dosyasının varsayılan ismi ise “AgentNotifier.log” olarak belirlenmiştir.

3.6 Sistem Gereksinimleri ve Kurulum

PPPJ kütüphanesinin kullanılacağı bir bilgisayarda JRE yazılımının 1.5.0 veya daha güncel bir sürümünün kurulu olması gerekir. Bilgisayar üzerinde kullanıcı programları geliştirilmeyecekse yalnızca bu yazılımın kurulu olması yeterlidir. Aksi durumda, uygulama geliştirme paketinin de (Java Development Kit) kurulması gerekmektedir.

PPPJ kütüphanesinin kurulum sürecinin ilk adımı, kütüphaneyi oluşturan program ve konfigürasyon dosyalarını içeren dizinlerin hedef bilgisayara kopyalanmasıdır. Söz konusu dizinlere ve dosyalara Ek 1’den erişilebilir. Dizinlerin isimleri ve içerikleri şöyledir:

- “assist” dizini: Kullanıcı programları oluşturulurken faydalanan TriplePJAssistant sınıfına ve ona yardımcı diğer sınıflara ilişkin program dosyalarını içerir.
- “cache” dizini: Çalışma zamanında oluşturulan bazı geçici dosyaların tutulduğu dizindir. Örneğin Communicator nesnelerinin yazıldığı dosyalar bu dizinde saklanır. Geçici dosyalar, zamanı geldiğinde, kendilerini yaratan program tarafından silinirler.
- “client” dizini: İstemci uygulamayı oluşturan program dosyalarını içerir.
- “common” dizini: İstemci ve sunucu uygulamalar ile araçlar tarafından ortak olarak kullanılan program dosyalarını içerir.
- “configuration” dizini: Çeşitli konfigürasyon dosyaları ile AgentWorld dosyasını içerir. Sunucu etmeninin ve log4j yazılımının yapılandırılmasında kullanılan konfigürasyon dosyaları bu dizinde saklanır.
- “lib” dizini: Kullanılan üçüncü parti yazılımlara (kütüphanelere) ilişkin program dosyalarını içerir. Bu dizindeki log4j yazılımı günlükleme için, “JDOM” [26] yazılımı ise XML dosyalarının yönetimi için kullanılır.
- “log” dizini: Kütüphanenin varsayılan günlükleme dizinidir. log4j yazılımından faydalanılarak oluşturulan günlük dosyaları bu dizinde saklanır.
- “server” dizini: Sunucu uygulamayı oluşturan program dosyalarını içerir.

- “tools” dizini: Kütüphanenin kullanımını kolaylaştıran araçlara (AgentNotifier ve JobConfigCreator) ilişkin program dosyalarını içerir.
- “user_apps” dizini: Kütüphanenin kullanımını gösteren örnek uygulamalara ilişkin program dosyalarını içerir.

Sistemde sadece sunucu olarak görev yapacak bir bilgisayara cache, common, configuration, lib, log, server ve tools dizinlerinin (içerikleriyle birlikte) kopyalanması yeterlidir. Hedef bilgisayarın hem sunucu hem de istemci olarak kullanılması düşünüldüğünde ise assist ve client dizinlerinin de kopyalanması gerekir. Sadece istemci rolünü üstlenecek bir bilgisayarda assist, client, common, configuration, lib, log ve tools dizinlerinin bulunması yeterlidir.

PPPJ kütüphanesi sistemde sunucu rolünü üstlenecek bir bilgisayara kuruluyorsa, ikinci adım, configuration dizininde yer alan “ServerAgent.properties” dosyasının içeriğinin düzenlenmesidir. Bu dosya sunucu etmeninin yapılandırılmasında kullanılır. İçerisinde sunucu etmenine dair aşağıdaki bilgileri barındırır:

- Sunucu etmeninin ismi.
- Sunucu etmeninin istemcilerden gelen talepleri almak için dinlediği kapı (port) numarası.
- Sunucu etmeni üzerinde çalışan proseslerin, başka proseslerden gelebilecek mesajları almak için dinleyebileceği en küçük kapı numarası.
- Sunucu etmeni üzerinde aynı anda çalışabilecek azami proses adedi.

Şekil 3.20’de örnek bir sunucu etmeni yapılandırma dosyası gösterilmektedir. Dosyadaki “port_to_listen” alanı, istemcilerden gelecek talepleri dinlemek için kullanılacak olan kapı numarasını ifade eder. Bu alanın değeri, söz konusu sunucu etmeni için AgentWorld dosyasında belirtilen değer ile aynı olmalıdır. Ayrıca bu değer belirlenirken, aynı bilgisayar üzerinde çalışan diğer uygulamalar tarafından kullanılmayacak bir tane olmasına özen gösterilmelidir. Aynı durum prosesler arası iletişimde kullanılabilir en küçük kapı numarasının (smallest_port_number_for_ipc) belirlenmesi aşamasında da geçerlidir. Çünkü sunucu etmeni, bu kapı numarasından başlayarak azami proses adedince (maximum_number_of_processes) kapı numarasını prosesler arası iletişimde kullanmak üzere ayırır. Örneğin Şekil 3.20’de görülen konfigürasyon dosyasına göre

yapılandırılan bir sunucu etmeni, prosesler arası iletişim için 10545 ila 10644 arasındaki kapı numaralarını kullanır. Uygun kapı numaralarının belirlenmesi sürecinde, kapı numaralarını denetim altında tutan IANA (Internet Assigned Numbers Authority) isimli kuruluşun internet sitesindeki bilgilerden yararlanılabilir [27].

```
agent_name = bornova  
port_to_listen = 10052  
smallest_port_number_for_ipc = 10545  
maximum_number_of_processes = 100
```

Şekil 3.20 : Sunucu etmeni yapılandırma dosyası örneği

Öte yandan, Windows işletim sistemine sahip bilgisayarlarda 5000’in üzerindeki kapı numaralarının kullanılabilmesi için sistem kütüğünde güncelleme yapılması gerekebilmektedir. Konuyla ilgili detaylı bilgi Microsoft’un internet sitesindeki ilgili yardım sayfasından [28] edinilebilir.

Kurulum sürecinin son adımını, çevre değişkenlerinin düzenlenmesi işlemi oluşturur. Öncelikle PPPJ kütüphanesinin kurulduğu dizinin yerini gösteren “PPPJ_PATH” isimli çevre değişkeninin oluşturulması gerekmektedir. Bunun yanı sıra, Java sanal makinesi için özel bir anlama sahip olan “CLASSPATH” çevre değişkeninin, PPPJ kütüphanesini ve kullanılan üçüncü parti kütüphaneleri (log4j ve JDOM) barındıran dizinleri de içerecek şekilde güncellenmesi gerekmektedir.

4 SONUÇ

Bu tez çalışmasıyla, Java için geliştirilmiş ve nesne tabanlı mesaj aktarımı yaklaşımını temel alan bir paralel programlama kütüphanesi sunulmaktadır. Kısaca PPPJ olarak isimlendirilen bu kütüphane, Java'nın soket ve kanal yapıları ile nesne serileştirme ve iç gözlem gibi özelliklerinden faydalanılarak tamamen bu dilde geliştirilmiştir. Bu sayede, yalnızca homojen sistemlerde değil, ayrışık bilgisayar sistemlerinde de kullanılabilir, platform bağımsız bir iletişim kütüphanesidir.

Uygulama geliştiriciler PPPJ kütüphanesini kullanarak kendi paralel programlarını kolaylıkla geliştirebilirler. Üçüncü bölümde açıkça ortaya konulduğu üzere, uygulama geliştirme arayüzünde yalnızca temel fonksiyonlara yer veren, nesne tabanlı oluşu sayesinde hem uygulama geliştirme sürecini kolaylaştıran hem de yazılan kaynak kodların tekrar kullanılabilmesine zemin hazırlayan ve çeşitli konfigürasyon dosyaları aracılığıyla esnek bir biçimde yapılandırılabilen PPPJ kütüphanesi, kullanıcı dostu bir yazılım sistemidir. Yalnızca kütüphanenin kendisi değil, üzerinde çalıştırılacak kullanıcı uygulamaları da özel konfigürasyon dosyaları aracılığıyla yapılandırılabilir. Ayrıca, kütüphane için özel anlam taşıyan bu gibi dosyaların yönetimini kolaylaştırmak amacıyla grafik kullanıcı arabirimine sahip iki de araç geliştirilmiştir. JobConfigCreator isimli araç vasıtasıyla kullanıcı uygulamalarının koşturulabilmesi için gerekli konfigürasyon dosyaları kolaylıkla oluşturulabilmekte, AgentNotifier aracıyla ise sistemdeki bilgisayarlar üzerinde bulunan AgentWorld dosyalarının yönetimi (güncellenmesi) tek noktadan sağlanabilmektedir.

PPPJ kütüphanesinde prosesler arası iletişim esnasında verilerin (nesnelerin) bir noktadan diğerine sıkıştırılarak aktarılabilmesi de sağlanmıştır. Bu özelliğin bilhassa büyük boyutlarda veri aktarımları söz konusu olduğunda ortaya çıkabilecek yüksek zaman maliyetlerini düşürmek açısından faydalı olacağı açıktır. Üstelik kullanıcının, bu özellikten faydalanmak isteyip istemediğini, çalıştırmak istediği programın konfigürasyon dosyasında belirtmesi yeterlidir. Bunun için kaynak kodları içerisinde herhangi bir değişiklik yapmasına gerek yoktur.

Kütüphane istemci-sunucu mimarisi üzerine kurulmuş, sistemdeki bilgisayarların isteğe bağlı olarak istemci ve/veya sunucu rollerini üstlenebilmeleri sağlanmıştır. Sistemde birden fazla sunucu bilgisayar olabileceği gibi birden fazla istemci de bulunabilmektedir. Genel olarak istemciler, görev alacak sunucuların belirlenerek kullanıcı uygulamalarının onlara yüklenmesinden, sunucular ise söz konusu uygulamaların yürütülmesinden sorumludurlar. Kütüphane, kullanıcı uygulamalarının çalıştırılacağı sunucuların seçimi için üç farklı seçenek sunmaktadır. Herşeyden önce kullanıcı, görevlendirilecek sunucuları kendisi belirleme hakkına sahiptir. Bunun yanı sıra, seçim işleminin kütüphane tarafından, sunucuların başarımlarına ya da yoğunluk düzeylerinin ölçülmesinden sonra yapılmasını da sağlayabilir. Gerektiğinde sistemdeki sunucuların başarımlar ve yoğunluk düzeylerinin ölçülmesi işlemleri PPPJ kütüphanesi bünyesinde geliştirilen ilgili bileşenlerce gerçekleştirilerek sunucular arasında bir öncelik sıralaması yapılmaktadır. Duruma göre yoğunluğu düşük ya da başarımlı yüksek olan sunuculara seçim sırasında öncelik tanınır. Ayrıca, seçim esnasında bir sunucu üzerinde herhangi bir nedenden ötürü gerekli kaynakların ayrılamaması durumunda, otomatik olarak bir sonraki yüksek öncelikli sunucunun devreye alınabilmesi sağlanmıştır. Kısacası PPPJ kütüphanesi kullanıcıya, hiçbir işlem yapmasına gerek kalmadan, sistemde en uygun durumda bulunan sunucuları seçme olanağı da sunmaktadır.

Sonuç olarak, kullanıcılara esneklik, taşınabilirlik ve kullanım kolaylığı gibi önemli özellikler sunan PPPJ kütüphanesi benzeri geliştirmelere iyi bir alternatif olmuştur.

KAYNAKLAR

- [1] NPACI Parallel Computing Institute. (2002). *Message Passing Interface - Part 1 of 3*. 24 Ağustos 2008 tarihinde <http://www-rohan.sdsu.edu/~stewart/cs575/npaci2002parallel/MPI1/index.html> adresinden erişildi.
- [2] *Message Passing Interface Forum*. (2008). 16 Mart 2008 tarihinde <http://www.mpi-forum.org/> adresinden erişildi.
- [3] Argonne National Laboratory Mathematics and Computer Science Division. (2008). *The Message Passing Interface (MPI) Standard*. 16 Mart 2008 tarihinde <http://www-unix.mcs.anl.gov/mpi/> adresinden erişildi.
- [4] **Barney, B.** (2007). *Introduction to Parallel Computing*. 8 Mart 2008 tarihinde https://computing.llnl.gov/tutorials/parallel_comp/ adresinden erişildi.
- [5] **Bruck, J., Dolev, D., Ho, C.-T., Roşu, M.-C., Strong, R.**, 1995. Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations, Technical Report, **TR95-1474**, Cornell University.
- [6] Maui High Performance Computing Center. (1997). *Message Passing Overview*. 26 Ağustos 2008 tarihinde http://www.mhpcc.edu/training/tutorials/html/message_passing/ adresinden erişildi.
- [7] Maui High Performance Computing Center. (2003). *Parallel Programming Introduction*. 22 Mart 2008 tarihinde http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html adresinden erişildi.
- [8] OpenMP. (2008). *The OpenMP API Specification for Parallel Programming*. 22 Mart 2008 tarihinde <http://www.openmp.org> adresinden erişildi.
- [9] **Barney, B.** (2008). *POSIX Threads Programming*. 22 Mart 2008 tarihinde <https://computing.llnl.gov/tutorials/pthreads/> adresinden erişildi.
- [10] **Sundararajan, H.** (2007). *Celeritas – A Reference*. 30 Mart 2008 tarihinde <http://www.cct.lsu.edu/~hsunda3/doc/> adresinden erişildi.
- [11] **Woodard, L.** (2008). *Basics of MPI Programming*. 23 Mart 2008 tarihinde <http://www.tc.cornell.edu/Services/Education/Topics/MPI/Basics.bak/Introduction.htm> adresinden erişildi.
- [12] **Pacheco, P. S.**, 1998. A User's Guide To MPI, Technical Report, California, ABD.
- [13] **Ginies, A., & Velu, E.** (2006). *Chapter 18. MPI Implementations*. 29 Mart 2008 tarihinde <http://iggi.mandriva.com/ch18.html> adresinden erişildi.

- [14] Argonne National Laboratory Mathematics and Computer Science Division. (2008). *MPICH2*. 29 Mart 2008 tarihinde <http://www.mcs.anl.gov/research/projects/mpich2/> adresinden erişildi.
- [15] *LAM/MPI Parallel Computing*. (2007). 29 Mart 2008 tarihinde <http://www.lam-mpi.org/> adresinden erişildi.
- [16] *Open MPI: Open Source High Performance Computing*. (2008). 29 Mart 2008 tarihinde <http://www.open-mpi.org/> adresinden erişildi.
- [17] **Baker, M., Carpenter, B., Fox, G., Ko, S. H., Li, X.**, 1998. mpiJava: A Java Interface To MPI, *1st UK Workshop on Java for High Performance Network Computing (Euro-Par'98)*, Southampton, Birleşik Krallık, Eylül 1998.
- [18] **Baker, M., Carpenter, B.**, 2000. MPJ: A Proposed Java Message Passing API and Environment for High Performance Computing, *International Workshop on Java for Parallel and Distributed Computing*, Cancun, Meksika, Mayıs 2000.
- [19] **Nelisse, A., Maassen, J., Kielmann, T., Bal, H. E.**, 2001. Ccj: Object-based Message Passing and Collective Communication in Java, *Joint ACM JavaGrande – ISCOPE 2001 Conference*, Stanford University, California, ABD, Haziran 2-4.
- [20] **Mohamed, N., Al-Jaroodi, J., Jiang, H., Swanson, D.**, 2002. JOPI: A Java Object-Passing Interface, *ACM Java Grande/ISCOPE*, Seattle, Washington, Kasım 2002.
- [21] **Carpenter, B., Getov, V., Judd, G., Skjellum, A., Fox, G.**, 2000. MPJ : MPI-like Message Passing for Java, *Concurrency: Practice and Experience*, **12(11)**, 1019-1038.
- [22] **Morin, S., Koren, I., Krishna, C. M.**, 2002. JMPI: Implementing the Message Passing Standard in Java, *Proceedings of the International Parallel and Distributed Processing Symposium*, Florida, ABD, Nisan 2002.
- [23] **Harrell, E. M., & Herod, J. V.** (1996). *Linear Methods of Applied Mathematics - Derivation of Heat Equation*. 24 Nisan 2008 tarihinde <http://www.mathphysics.com/pde/HEderiv.html> adresinden erişildi.
- [24] **Stapel, E.** (2008). *Scalar and Matrix Multiplication*. 28 Temmuz 2008 tarihinde <http://www.purplemath.com/modules/mtrxmult.htm> adresinden erişildi.
- [25] Apache Logging Services. (2007). *Apache log4j*. 25 Ağustos 2008 tarihinde <http://logging.apache.org/log4j/> adresinden erişildi.
- [26] *JDOM*. (2007). 25 Ağustos 2008 tarihinde <http://www.jdom.org/> adresinden erişildi.
- [27] Internet Assigned Numbers Authority. (2008). *Port Numbers*. 30 Ağustos 2008 tarihinde <http://www.iana.org/assignments/port-numbers> adresinden erişildi.

- [28] Microsoft Yardım ve Destek. (2007). *When you try to connect from TCP ports greater than 5000 you receive the error 'WSAENOBUFFS (10055)'*. 30 Ağustos 2008 tarihinde <http://support.microsoft.com/kb/196271> adresinden erişildi.

ÖZ GEÇMİŞ

Çağlar Kumanlı 1981 yılında İstanbul’da doğdu. Lise eğitimini Avcılar 50. Yıl İnsa Lisesi’nde tamamladı. 1999 yılında girdiği Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği Bölümü’nden 2004 yılında mezun oldu. 2005 yılından bu yana İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği programında yüksek lisans eğitimi alıyor. Ayrıca, 2004 yılında Tikle YBS A.Ş. bünyesinde başladığı yazılım mühendisliği görevine 2006 yılından bu yana Mobilera Bilişim ve İletişim Teknolojileri A.Ş. çatısı altında devam ediyor.