Sanal Paralel Makina

Burhanettin DURMUŞ

YÜKSEK LİSANS TEZİ

Elektrik-Elektronik Anabilim Dalı

Haziran-2003

SANAL PARALEL MAKİNA

Burhanettin Durmuş

DUMLUPINAR ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik-Elektronik Anabilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır.

Danışman : Yrd. Doç. Dr. Ahmet Özmen

Haziran - 2003

KABUL VE ONAY SAYFASI

Burhanettin Durmuş'un YÜKSEK LİSANS tezi olarak hazırladığı SANAL PARALEL MAKİNA başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir

09/07/2003

Üye : Yrd.Doç.Dr. Ahmet Özmen

Üye : Prof.Dr. Abdurrahman Karamancıoğlu

Üye : Yrd.Doç.Dr. Ahmet ALTUNCU

Fen Bilimleri Enstitüsün Yönetim Kurulu'nun 18/07/2003 gün ve 11 sayılı kararıyla onaylanmıştır.

Prof.Dr.M.Sabri ÖZYURT

Fen Bilimleri Enstitüsü Müdürü

SANAL PARALEL MAKİNA

Burhanettin DURMUŞ
Elektrik – Elektronik Mühendisliği, Yüksek Lisans Tezi, 2003
Danışman : Yrd.Doç.Dr. Ahmet ÖZMEN

ÖZET

Bazı uygulamalar için yüksek performanslı bilgisayarlara ihtiyaç vardır. Bu uygulamalar gerçek zamanlı kontrol sistemleri, gerçek zaman kriterlerine esnek ancak yine de zamana karşı hassas olan simülatörler veya icrası çok uzun süren hesaplama ağırlıklı programlardır. Bu çalışmada bu tür ihtiyaçlara cevap verebilecek, dağıtık hesaplama mimarisi altında, bir işletim sistemine ek olarak tasarlanmış sanal bir paralel bilgisayar (PVM: Parallel Virtual Machine) ve grafiksel arayüzü XPVM tanıtılmış, PVM ile oluşturulan dağıtık hesaplama ortamında paralel programlar çalıştırılmış, programların icra süreleri ölçülerek gerçeklenen paralellikten doğan performans gözlemlenmiştir. PVM, birbirlerine ağ ile bağlı, farklı mimari ve işletim sistemleri ile çalışan dağınık halde duran bilgisayarları bir amaç için ve aynı anda çalıştırmaya yarayan yazılım paketidir. PVM ile sisteme katılan bilgisayarlar görünürde birer iş istasyonu ya da kişisel bilgisayar olsalar da, sistem sonunda sanal bir paralel makineye dönüşmektedir. Dolayısıyla PVM, oldukça pahalı yüksek performanslı bilgisayar sistemleri alamayan üniversiteler ve araştırma kurumları için ekonomik bir çözümdür.

Anahtar Kelimeler: Ağ, Arayüz, Dağıtık Hesaplama, Dağıtık Paylaşımlı Hafıza, Mesaj Geçişi, Sanal Paralel Makina, XPVM

PARALLEL VİRTUAL MACHİNE

Burhanettin DURMUŞ
Electric & Electronic Engineering, M.S. Thesis, 2003
Thesis Supervisor : Asst.Prof.Dr. Ahmet ÖZMEN

SUMMARY

Some applications require high performance computing environment. These applications can be real-time control systems, soft real-time simulators or long running non-interactive compute bound tasks. In this paper, a parallel virtual machine (PVM) and its graphical interface (XPVM) have been introduced to meet high performance computing demands, parallel programs have been run in distributed computing media that is formed with PVM, performance by paralellizm is examined with measuring the execution times. PVM is a software package designed as a library extension to an operating system. A virtual parallel system can easily be build from networked computers using PVM. These computers can also be heterogeneous according to their architecture and operating systems. Therefore, PVM makes it possible to build affordable high performance computers for universities and research companies using ordinary workstations and personal computers.

Key Words: Distributed Computing, Distributed Shared Memory, Interface, Massage Passing, Network, Parallel Virtual Machine, XPVM

TEŞEKKÜR

Çalışmalarım sırasında yardımlarını hiçbir zaman esirgemeyen danışman hocam Yrd.Doç.Dr. Ahmet ÖZMEN'e, çalışmalarımda beni sabırla destekleyen kardeşim Semih'e, hiçbir zaman maddi ve manevi desteklerini esirgemeyen aileme, bölümümüzün değerli öğretim elemanlarından Yrd.Doç.Dr. A.İhsan ÇANAKOĞLU, Arş.Gör. Gültekin KUVAT, Arş.Gör.Hasan TEMURTAŞ ve diğer tüm mesai arkadaşlarıma sonsuz teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

ŞEKİLLER DİZİNİ (devamı)

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
ALU	Aritmetik Lojik Birim
ANN	Yapay Sinir Ağı
DM	Veri Hafıza
DSM	Dağıtık Paylaşımlı Hafıza
FDDI	Fiber Dağıtık Veri Arayüzü
IM	Komut Belleği
IN	Bağlantı Ağı
LAN	Yerel Alan Ağı
MIMD	Çoklu Komut Çoklu Veri Kümesi
MISD	Çoklu Komut Tek Veri Kümesi
MPI	Mesaj Geçiş Arayüzü
PAG	Program Aktivite Grafiği
PE	Proses Element
PN	İşlemci Ünitesi
PVM	Sanal Paralel Makina
SIMD	Tek Komut Çoklu Veri Kümesi
SISD	Tek Komut Tek Veri Kümesi
TID	Görev Kimlik Numarası
IM/DD	Direkt Dedeksiyon ile Modülasyon Şiddeti
LAN	Yerel Alan Ağı
WAN	Geniş Alan Ağı
XPVM	PVM Grafiksel Arayüzü

1. GİRİŞ

Yüksek performanslı hesaplama yeteneği olan bilgisayar sistemlerine ihtiyaç gün geçtikçe artmaktadır. Tek işlemcili bir sistemden yüksek performans elde etmek hem fiziksel ve hem de ekonomik nedenlerden dolayı çok cazip değildir. Bunun yerine bir süredir çok işlemcili paralel sistemler, ortak bellekli ve dağıtık olarak gerçeklenmektedir [3]. Ortak bellekli paralel sistemlerin programlanması kolay, ancak üretilmesi zor ve bu nedenle pahalıdır. Buna bir alternatif ise, piyasadan kolaylıkla temin edilebilen bilgisayarların hızlı bir ağ ile irtibatlandırılması sonucu elde edilen dağıtık paralel sistemlerdir. Bu tür sistemler, yüksek performans ihtiyacına ekonomik çözümler sunmakla birlikte programlanması zordur.

1.1. Tek İşlemcili Sistemler

Genel olarak bir bilgisayarın performansını belirleyen en önemli unsur icra süresidir. Performansı arttırmak icra süresinin kısaltılması ile mümkündür. Bu ise, işlem hızının arttırılması veya icra edilen programda iyileştirmelerle gerçeklenebilir. İşlem hızının arttırılması yüksek hızlı işlemci ve bellek kapasitesinin arttırılması ile mümkündür. Öte yandan icra edilen programın herhangi bir noktasında yapılacak olan bir iyileştirmede programın icra süresini kısaltabilir. Bu iyileştirmeler için yeni algoritmalar geliştirilse de tek işlemcili sistemlerde bu sınırlı kalmaktadır. Tek işlemcili sistemlerde karşımıza çıkan bu performans problemleri paralel bilgi işleme ile aşılabilir.

1.2. Paralelliğe Olan İhtiyaç

Bilgisayar sistemlerinin performansı, işlemci saat frekansı ve bellek kapasitesi ile doğrudan orantılıdır. Birim zamanda işlenen komut sayısını arttıracağından daha yüksek frekanslarda çalışabilen işlemciler kullanılarak performans arttırılabilir. Ancak işlemci frekanslarını arttırmak fiziksel sebeplerden dolayı sınırlıdır. Diğer yandan performans, ilave bellek kullanılarak, bellek kapasitesinin genişletilmesiyle de arttırılabilir. Ancak bu da belirli değerlerden sonra ekonomik değildir.

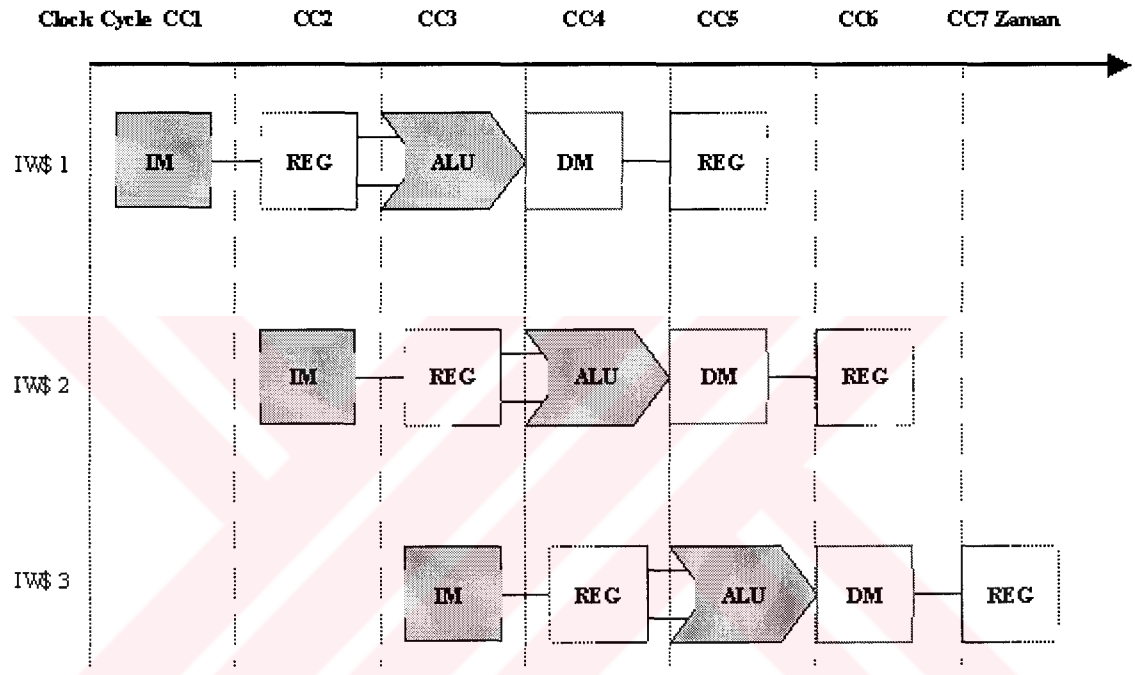
Görülüyor ki hem fiziksel hem de ekonomik nedenlerden dolayı tek işlemcili bilgisayar sistemlerinin performansı sınırlıdır. Sonuç olarak, performansı arttırmak için çeşitli boyutlarda paralellik işlemci içinde veya dışında ekonomik çözümler için kaçınılmaz olmuştur.

1.3. Paralellik

Paralellik donanımda çoklama ile sağlanabilir. Örneğin; ideal şartlarda bir işlemci ile sonuçlar N saatte alınıyorsa N işlemci kullanılarak sonuçlar 1 saatte alınabilir. Bu sayede

işlemci başına düşen iş yükü düşeceğinden işlemler daha hızlı sonuçlanacaktır. Paralellik çipiçi ve çipdışı olmak üzere ikiye ayrılır.

Çipiçi paralellik; günümüz işlemcilerinde yoğun olarak kullanılan komut bazında gerçekleşen paralelliktir. Bu paralelliğin pipeline mimarisinde nasıl gerçekleştiği Şekil 1.1'de gösterilmiştir.

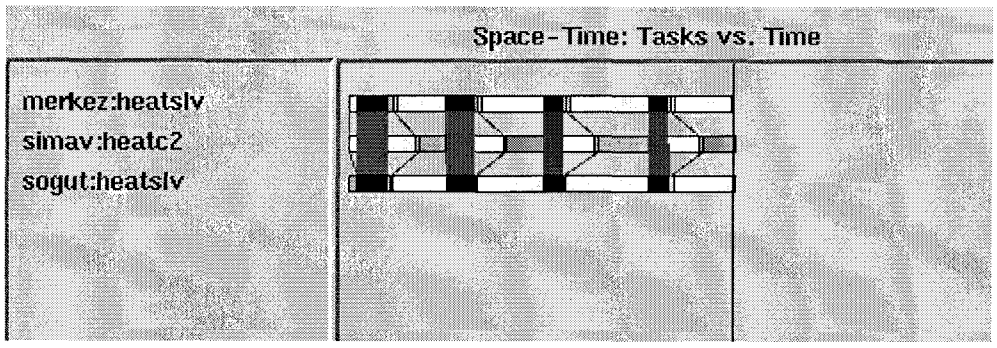


Şekil 1.1 Pipeline mimaride çipiçi paralellik [8]

Her bir komut, programın icrası sırasında Komut Bellği (IM), Komut Saklayıcı (Register), Aritmetik Lojik Birim (ALU), Veri Hafıza (DM) ve Veri Saklayıcılar (Register) olmak üzere çeşitli işlem basamaklarından geçirilmektedir. Ardışık saat darbeleri ile her bir komut zincirleme olarak bu işlem basamaklarında ilerler. Eğer çipiçi paralellik olmasaydı her komut kendinden önceki komutun icrasının tamamlanmasını bekleyecekti. Paralellik sayesinde komutların icrası zincirleme adımlarla daha kısa sürede tamamlanmaktadır. Pipeline, superpipeline ve vektör işlemciler gibi yapılar bu paralelliğe örnek gösterilebilir. Bu tür yapıda çip içine daha fazla donanım konulması gerekmektedir. Örneğin Şekil 1.1'deki sistemde aynı çip içinde iki adet ALU bulunması gerekmektedir. Bunlardan birisi aritmetik-lojik işlemler için, diğeri ise dallanmalarda yeni adres hesabı için kullanılmaktadır.

Çipdışı paralellik; birden fazla işlemci ile program bazında gerçekleştirilen paralelliktir. Bu tür paralellikte program parçalanarak farklı işlemcilerde icra edilmektedir. İşlemciler kendi

aralarında haberleşerek koordineli bir şekilde programı icra ederler ve sonuçları ana işlemciye gönderirler. Bu tür sistemlerin en büyük sorunu programlanmasındaki zorluktur. Şekil 1.2'de paralel bir programın çok işlemcili ortamda icrası gösterilmiştir.



Şekil 1.2 Çok işlemcili ortamda paralellik

1.4. Sanal Paralel Makina; PVM (Parallel Virtual Machine)

PVM bir paralel bilgisayar gibi görünecek şekilde ağdaki bilgisayarların heterojen olarak kullanılmasına imkan veren bir yazılım sistemidir. Merkezi bir kontrol altında farklı mimarileri bir araya getirmeye yönelik bu kabiliyet sayesinde PVM kullanıcısı, bir problemi alt görevlere bölebilir ve en uygun işlemci mimarisine atayabilir. Yüklenmesi ve kullanımı kolay olduğu için popülerdir. Küçük ve büyük ölçekli uygulamaları geliştirmede dünya çapında kullanılmaktadır.

Çok sayıda işlemcili sistemlerde paralel programların çalıştırılabilmesine olanak sağlayan bu yazılım paketi, paralel programların geliştirilmesinde önemli rol oynamaktadır. PVM, bilgisayar kaynaklarının bağlanması için tasarlanmakta ve kullanıcılara bilgisayar uygulamalarını çalıştırmaları için, kullandıkları bilgisayarların sayısından ve yerlerinden bağımsız olarak bir paralel platform sunmaktadır. Yüksek düzeyde performans ve fonksiyonellik vermek üzere heterojen şebekelenmiş hesaplama platformlarının birleştirilmiş kaynaklarını organize etme kabiliyetine sahiptir. Sistemin esas hedefi sistemden en iyi performansı almaktır.

1.5. PVM İçin Grafiksel Arayüz; XPVM

XPVM, PVM konsol komutlarına grafiksel ara yüz olanağı tanır. PVM programlarının icrası sırasında ağın durumu, uzay-zaman bakışı ve görev takibi gibi bazı animasyonların on-line veya off-line olarak gözlemlenmesine imkan sağlar. Performansın belirlenmesi ve programın hatalardan arındırılması için bu gözlemleme aracı, PVM uygulamalarında çalışan işlerin birbirleriyle etkileşimini hakkında bilgi kaynağı sağlar.

1.6. Tezde Sunulan Çalışma

Bu tezde, paralel bilgi işlemenin gerekliliği anlatılmış, paralel bilgi işleme modelleri, dağıtık paralel bilgi işleme, PVM ve PVM grafik arayüzü XPVM tanıtılmıştır. Mimari ve işletim sistemi bakımından heterojen bir ortamda dağıtık paralel sistem oluşturulması, PVM' in kurulması, dağıtık bilgi işleme modelinde örnek uygulamaların geliştirilmesi ve sistemde çalıştırılması, paralel uygulamaların performans problemlerinin gözlemlenmesi, yorumlanması ve çözülmesi gerçeklenmiştir. Örnek uygulama olarak bakır bir levhadaki ısı yayılımını hesaplayan Heat adlı program paralel ortamda çalıştırılmış ve hedeflenen paralellikten doğan performans artışı gözlenmiştir.

1.7. Araştırma Hedefleri ve Tezin Yapısı

Bu çalışmada dağıtık paralel sistemin oluşturulması, analizi ve dağıtık paralel programların performans problemlerinin yorumlanması ve giderilmesi hedeflenmiştir.

Bölüm 2'de SIMD, SISD, MIMD gibi Paylaşılmış Bellekli ve Dağıtılmış Bellekli Sistem modellerinin yapıları incelenmiş, günümüzde kullanılan çok işlemcili sistemler örneklendirilmiştir. Bölüm 3'de PVM ve grafiksel ara yüzü XPVM yazılım paketleri tanıtılmıştır.

Bölüm 4'de PVM uygulamalarına yer verilmiş, programların işlemci sayısı-problem boyutuna karşılık icra süreleri ölçülmüştür. Sonuçlar, Matlab ile grafiğe dökülmüştür. Grafiklerden performanstaki değişim gözlenmiş ve performansın optimum noktaları saptanmıştır. Bölüm 5'de PVM' e benzer diğer paralel ortamlar tanıtılmıştır. Bölüm 6'da yapılan çalışmanın sonuçları yorumlanmış, yoğun programlama gücü olan sistemlerin tasarımında neler yapılabileceğine dair öneriler getirilmiştir.

2. PARALEL MİMARİLER

Genel olarak binalar amaçlara göre odalara bölünmüştür. Bu odalar birbirlerine kapılar, koridorlar ve merdivenlerle bağlanmıştır. Bu organizasyonlar mimarinin özünde vardır. Aslında bina mimarisinde mühendislikten öte kavramlarda önemlidir. Zira güzellik ve estetik bina mimarisi için önemlidir. Bu özellik bilgisayar mimarileri içinde önemlidir [10].

Bilgisayar mimarileri işlemciler, bellek giriş-çıkış alt sistemleri gibi temel işlem bloklarının seçimiyle ve bu blokların birbirleri ile olan etkileşimi ile ilgilenir. Bilgisayar mimarı, belirli kriterlere göre bu blokları seçer ve bağlar. Buradaki amaç maliyeti düşürmek, hız ve güvenilirliği arttırmaktır. Bilgisayar mimarı, bilgisayarın ne gibi foksiyonları yerine getireceğini belirlemeli ve işlenen verinin türüne göre en uygun mimariyi seçmelidir.

Bilgisayar mimarileri çok hızlı bir biçimde değişmektedir ve bu alanda kısa zamanda çok fazla yol katedilmiştir. Sonuç olarak, bilgisayarlar daha hızlı ve daha esnek olmaktadır. Bugünün bir çipi, 40 yıl önce bir sinema salonunu kaplayan bilgisayardan 100.000 kat daha hızlı çalışmaktadır.

2.1. Bilgisayar Mimari Sınıflandırması

Bilgisayar mimarisi sınıflandırmasında Flyn Sınıflandırması temel teşkil eder. Ancak, bilgisayar teknolojisindeki gelişmeler bu sınıflandırma ile tanımlanamamaktadır. Örneğin, vektör işlemcilerini ve hybrid mimarilerini yeterince sınıflandıramamaktadır. Bu sorunun üstesinden gelebilmek için DAS 90, HOC 87, SKI 88, BEL 92 gibi bazı yeni sınıflandırmalar önerilmiştir [10]. Önerilen bu sınıflandırmaların çoğu Flyn Sınıflandırmasını SIMD ve MIMD açısından muhafaza eder. Önerilen bu sınıflandırmaların özellikleri bilgisayar işlemci mimarilerinde anlatılmıştır.

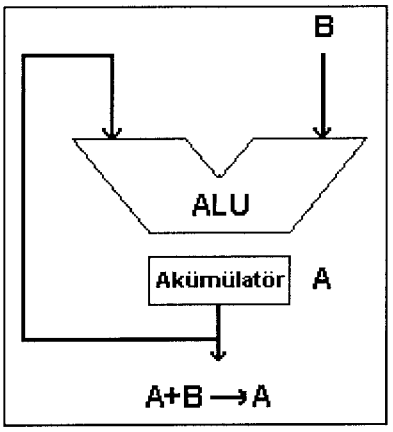
2.1.1. Flyn sınıflandırması

En çok tanınmış bilgisayar mimarisi sınıflandırmalarından birisi Flyn Sınıflandırması'dır. Micheal Flyn, mimariyi dört farklı katagoride sınıflandırmıştır. Bunlar tekli yada çoklu veri ve komut kümesine göre değerlendirilir. Flyn'ın dört sınıflandırması şunlardır:

2.1.1.1. Tekli komut-tekli veri kümesi (SISD)

Bu model Von Neumann'ın seri bilgisayar dizayn modelidir. Herhangi bir anda tek komut işlenir. Bu nedenle SISD, çoğunlukla seri bilgisayar olarak tanımlanır. Bütün SISD makinalarda tek akümülatör kullanılır ve komutların işlenmesi seri olarak sağlanılır. Herbir komut bellekten okunduğunda program sayıcı sonraki komutun adres alanını içerecek şekilde

güncellenir. Oysa günümüz kişisel bilgisayarlarında performans için az da olsa paralellik kullanılır ve çoğu durumunda iki veya daha fazla komut eşzamanlı olarak işlenir. Bu sebeple günümüzde SISD bilgisayarların üretimi azalmıştır [10].



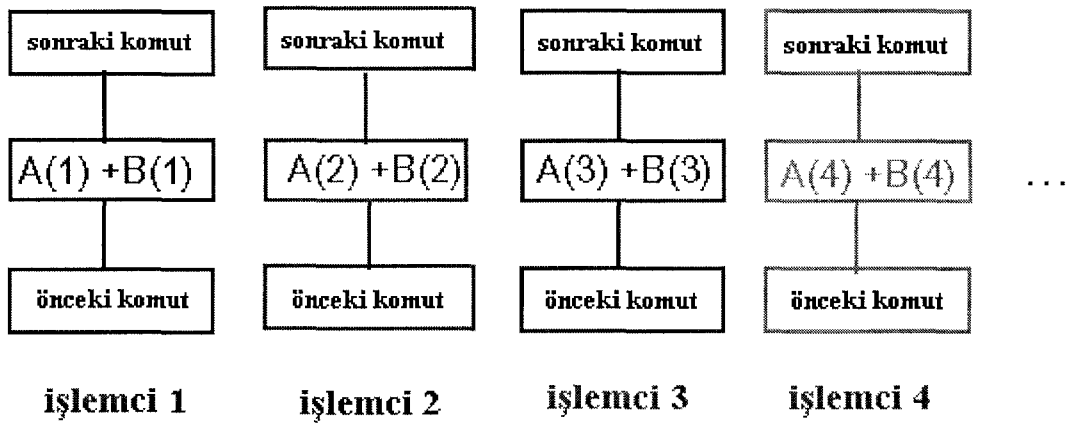
Şekil 2.1 SISD Model: Von Neumann mimarisinde işlemci

2.1.1.2. Çoklu komut-tek veri kümesi (MISD)

Birden fazla komutun tek parça veri üzerinde operasyon yapmasıdır. MISD makinaların organizasyonu için iki yöntem vardır. İlk yöntem; farklı işlemci ünitelerine ait farklı komutların aynı veri üzerinde operasyon yapmasıdır. Bu tür mimarinin gerçeklenmesi mümkün değildir. Diğer yöntem ise verinin seri bir işlemci ünitesi boyunca aktığı makina sınıfıdır. Pipeline mimarilerinde, örneğin vektör işlemciler çoğunlukla bu tip makinalar arasında sınıflandırılır. Pipeline mimarileri vektör işlemlerini seri adımlar boyunca yerine getirir. Herbiri kendine ait bir fonksiyonu icra eder ve ara adım sonuçlarını üretir. Bu tip mimarilerin MISD sistemler olarak tanımlanmasının sebebi, vektör elemanlarının aynı veri parçasına ait olmasının varsayılması ve tüm pipeline adımlarının çoklu komut kümesini temsil etmesidir [10].

2.1.1.3. Tekli komut-çoklu veri kümesi (SIMD)

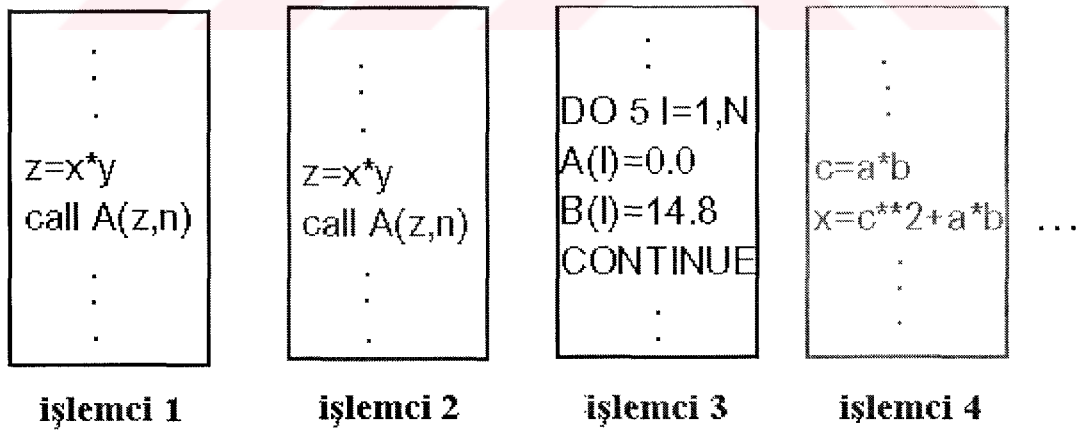
Bu model bir komutun farklı verilere eşzamanlı uygulanması ile elde edilir. Bu modelde birçok farklı işlem üniteleri tek kontrol ünitesi tarafından yönlendirilir. MISD'de olduğu gibi SIMD makinalarda vektör işlemeyi destekler. Vektör elemanlarını eşzamanlı hesap için herbir işlemciye atayarak hedefe ulaşır. Örneğin, 1000 çalışanın aylık ve haftalık kazancını hesaplayan programı düşünelim. SISD makinasında bu işlem 1000 adet seri iterasyonu gerektirir. Oysaki SIMD makinasında bu hesap eşzamanlı ve paralel olarak 1000 farklı veri akışı için bir iterasyonda yapılabilir. Yani tek komut ile 1000 çalışanın kazançları hesaplanabilir [10].



Şekil 2.2 SIMD Model: Vektör işlemci

2.1.1.4. Çoklu komut-çoklu veri kümesi (MIMD)

Bu modelde makinalar birden fazla işlemci ünitelerine sahiptir ve çoklu komutlar çoklu veriye uygulanır. İşlemciler hesaplamayı kendi komut girişine göre yaparlar, eşzamanlı değildir. Çok işlemcili, çoklu bilgisayar ve veri akış makinaları bu modele örnektir. MIMD, dağıtık hesaplamanın temelini oluşturur. Birbirine ağ ile bağlı bilgisayarlar, birbirlerinden bağımsız olarak hesaplamalarını yapabilirler. MIMD, mimarilerin en karmaşığıdır, öte yandan erişilebilen en büyük performansa ulaşma özelliğini sağlar.



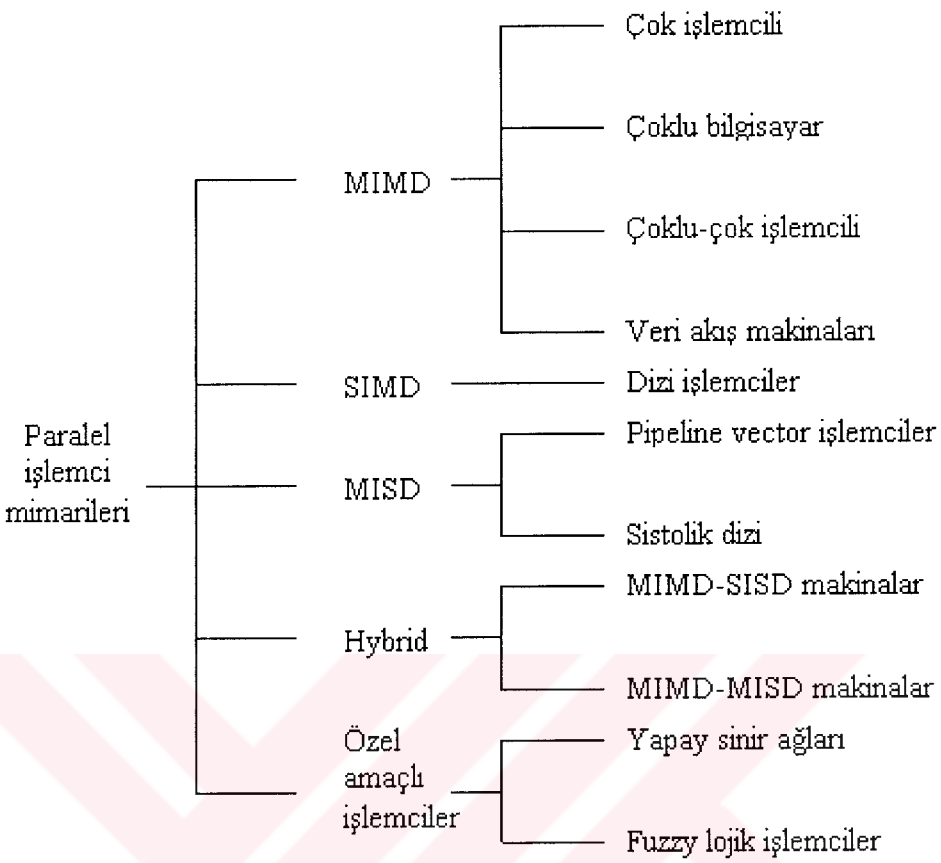
Şekil 2.3 MIMD Model: Dağıtık paralel mimari

Flyn'ın sınıflandırmasını otomobil üreten bir fabrikaya benzetebiliriz. SISD mantığına göre bir otomobilin üretimi tamamen bir kişi tarafından yapılmaktadır. MISD ise her bir işçi kendine özgü görevleri kendinden önceki işçinin sonuçlarını devralarak tamamlanmasına benzer. SIMD mimarisinde ise birden fazla işçinin aynı işi eşzamanlı olarak yapmasıdır. Bütün

işçilerin işi bittiğinde herbirine farklı bir görev verilir. Herbir işçi aynı zamanda aynı işi yaparak otomobili meydana getirir. Bir sonraki komut bütün işçilere aynı zamanda ve aynı kaynak tarafından verilir. MIMD ise SIMD'e benzemektedir. Tek farkı herbir işçi aynı işi eşzamanlı olarak yapmaz; herbiri otomobili kendi komut girişine bağlı olarak yapar.

2.1.2. Paralel işlemci mimarileri

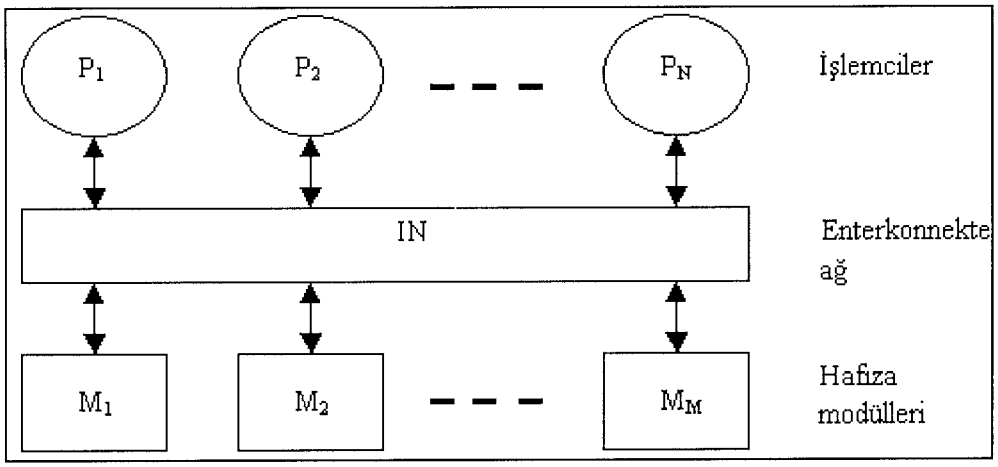
Geçen otuz yıl boyunca bilgisayar mimarilerinin sınıflandırılmasında Flyn Sınıflandırması iyi bir yöntem olarak görülmüştür. Bilgisayar mimarları tarafından geniş bir biçimde kullanılması bunun bir delilidir. Bununla birlikte bilgisayar teknolojilerindeki yeni gelişmeler Flyn Sınıflandırması ile tanımlanamamaktadır. Örneğin, vektör işlemcilerini ve hybrid mimarilerini yeterince sınıflandıramamaktadır. Bu sorunun üstesinden gelebilmek için bazı yeni sınıflandırmalar önerilmiştir [DAS 90, HOC 87, SKI 88, BEL 92]. Önerilen bu sınıflandırmaların çoğu Flyn Sınıflandırmasını SIMD ve MIMD açısından muhafaza eder. Bu iki özellik birçok mimariyi karakterize etmek için kullanılabilir. Şekil 2.4, önerilen yeni sınıflandırmaların özelliklerini göstermektedir. Bu sınıflandırma ile yakın geçmişteki mimarilerin çoğunun kapsanması amaçlanmıştır [10].



Şekil 2.4 Paralel işlemci mimarilerinin sınıflandırılması [10]

Şekil 2.4'de gösterildiği gibi MIMD bilgisayarlar daha sonra dört ayrı parallel makinalara ayrılmıştır: çok işlemcili, çoklu bilgisayar, çoklu çok işlemcili ve veri akış makinaları. SIMD sınıfı için sadece dizi işlemciler sınıflandırmaya dahil edilmiştir. Geriye kalan paralel mimariler iki sınıf halinde gruplandırılmıştır: Hybrid makinalar ve özel amaçlı işlemciler.

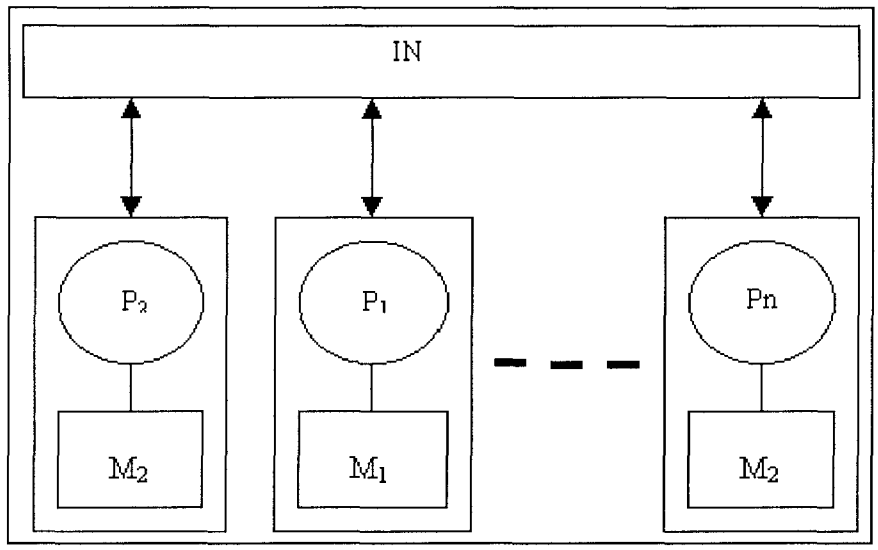
Çok işlemcililer tek bellek sistemini paylaşan, birbirleri ile etkileşimli işlemcilerden oluşan paralel bilgisayar görünümündedir. Her işlemci programın farklı bir tarafında çalıştırılacak şekilde görevlendirilir. Böylece farklı programlar eşzamanlı olarak çalışır.



Şekil 2.5 Çok işlemcili bir sistemin blok diyagramı [10]

Şekil 2.5'de çok işlemcili bir sistem görülmektedir. Bu çok işlemcili sistem N adet işlemci ve M adet bellek modülünden oluşmaktadır. İşlemciler P_1, P_2,... ve P_N, bellek modülleri M_1, M_2, ... ve M_M olarak adlandırılsın. İşlemciler arası bağlantı ağı (IN;Interconnection Network), herbir işlemciyi bellek modülünün kendisine ayrılmış birimine bağlar. Bir transfer komutu, verinin herbir işlemciden bağlı bulunduğu belleğe yönlendirilmesini sağlar. İki işlemci arasında veri transferi için, veriyi arabellek ve işlemcilere ileten programlanmış veri transfer sırası çalıştırılmaktadır.

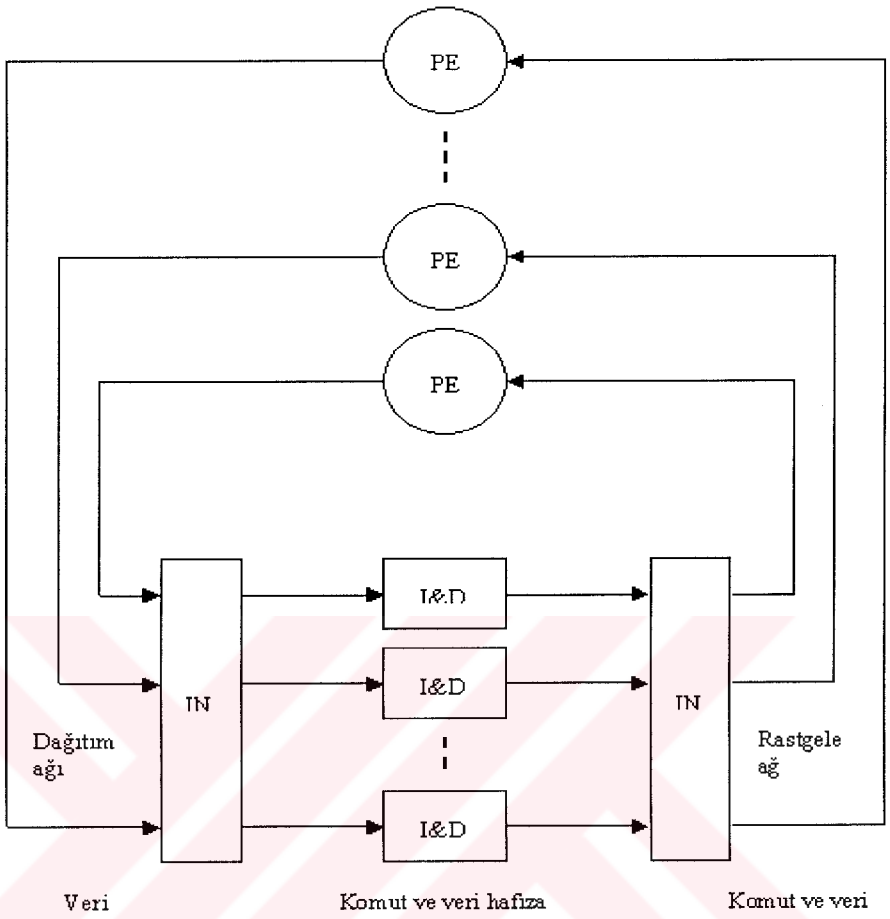
Çok işlemcililerden farklı olarak çoklu bilgisayarlarda, herbir işlemcinin kendi lokal belleği vardır. Çoklu bilgisayarlarda ana bellek işlemciler arasında dağıtılmıştır. Her işlemci kendi lokal belleğine doğrudan erişme hakkına sahiptir ve diğer işlemcilerin belleklerini adresleyemez. Bu özellik, çoklu bilgisayarları çok işlemcililerden ayıran en büyük farktır. Bu mimarinin blok diyagramı Şekil 2.6'da gösterilmiştir. Şekilde N tane işlemci ünitesi (PN:Processing node) vardır. Her PN bir işlemci ve bir lokal bellekten oluşur. İşlemciler arası bağlantı ağı (IN), herbir PN'i diğer PN alt kümesine bağlar.



Şekil 2.6 Çoklu bilgisayarın blok diyagramı [10]

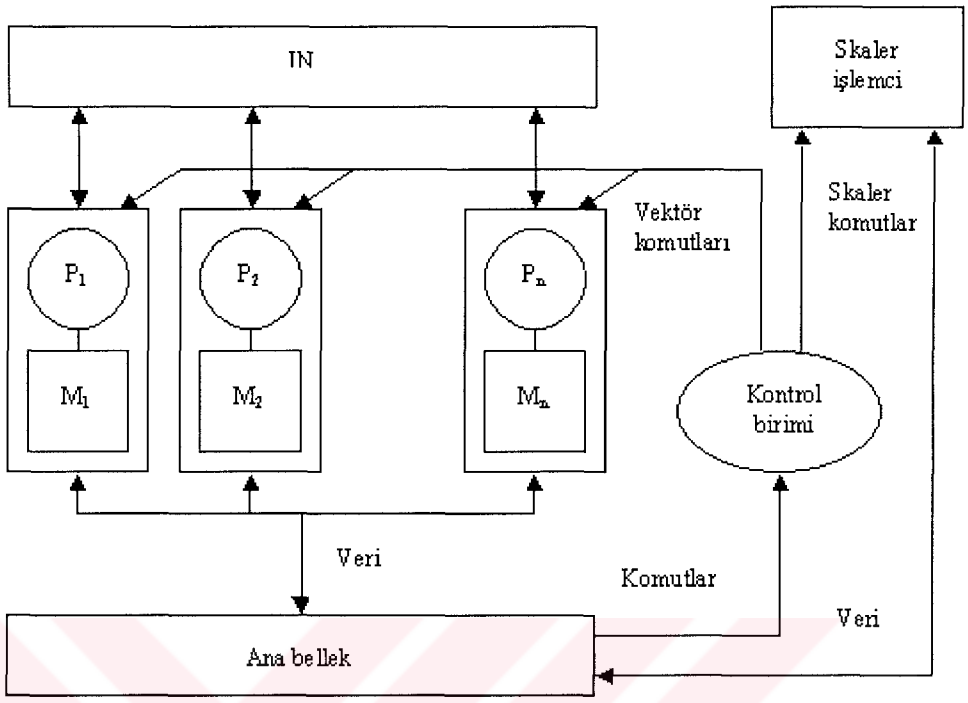
Çoklu çok işlemcililer ise, çok işlemcililer ve çoklu bilgisayarların özelliklerini birleştirir. Her bir işlemcinin kendi yerel belleği ve ağ ile bağlandığı bellek modülünde kendisine ayrılmış bellek birimi vardır. Her işlemci kendi lokal belleğine doğrudan erişme hakkına sahiptir ve diğer işlemcilerin belleklerini adresleyemez.

Veri akış makinalarında (data-flow machine) ise, bir önceki çalıştırılmış komut kümesinden gelen sonuçlar alınarak sıradaki komutun işlenenlerine verilir ve bir sonraki komutun icrası başlatılır. Bu mantık ile bir veri akışı oluşur ve komutların çalıştırılması tetiklenir. Veri akış komutları paylaşımlı bellekte adresleme yapmaz. Bunun yerine değişkenlerin değerini kendilerinde saklar (self-contained). Veri akış makinalarında bir komutun çalışması, çalışmaya hazır diğer komutları etkilemez. Bu sayede birden fazla hazır komut eşzamanlı olarak çalıştırılabilir. Şekil 2.7'de veri akış mimarisinin blok diyagramı gösterilmiştir.



Şekil 2.7 Veri akış makinasının blok diyagram [10]

Veri akış makinalarında komutlar ve bunların işlenenleri, komut ve veri belleğinde(I&D) saklanır. Herhangi bir komut çalışmaya hazır olduğunda, ağ üzerinde rastgele seçilen işlemci elementlerinden birine gönderilir. Herbir işlemci, sınırlı lokal diske sahiptir. İşlemci bir komut aldığında, gerekli operasyonu hesaplar ve sonucu hedef belleğe yazılmak üzere dağıtım ağına yollar.



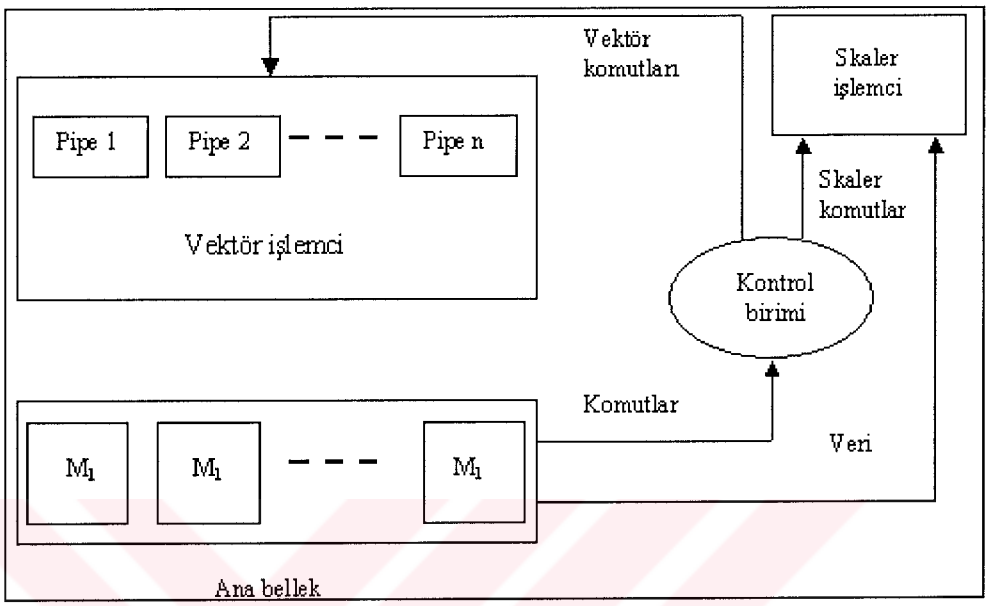
Şekil 2.8 Dizi işlemcinin blok diyagramı [10]

Bir dizi işlemcisi, PN'lerden ve merkezi kontrol ünitesi altında operasyon yapan skaler işlemciden oluşur. Kontrol ünitesi ana bellekten talimatları alır, bunları çözer ve daha sonra türüne göre skaler işlemcilere veya PN'lere gönderir. Eğer alınan talimat skaler bir talimat ise skaler işlemciye gönderir. Aksi taktirde talimatların hepsini PN'lere gönderilir. Bütün PN'ler aynı talimatı eşzamanlı olarak kendi lokal belleklerindeki farklı veriler üzerinde çalıştırır. Böylece, dizi işlemcisi sistemdeki tüm PN'leri kontrol için tek programa ihtiyaç duyar. Sonuçta her PN'de aynı program kodu kullanılmış olmaktadır. Dizi işlemcilerin yapısı Şekil 2.8'de görülmektedir.

Dizi işlemcilerdeki amaç, problemin komut çalıştırma sırasını paralelleştirmekten çok verilen problemin veri kümesini paralelleştirilmesidir. Her bir işlemciyi bir veri kümesine atarsak paralel hesap kavramı ile karşı karşıya kalırız. Eğer veri kümesi bir vektör ise sonuçta bir vektör elemanı olur. Dizi işlemciler tüm veri bölümlerini eşzamanlı işleyerek performansı arttırır. Dizi işlemciler ile vektörler üzerinde aritmetik ve lojik işlemler yapılabilir. Bu sebepten dolayı vektör işlemciler olarak da adlandırılır.

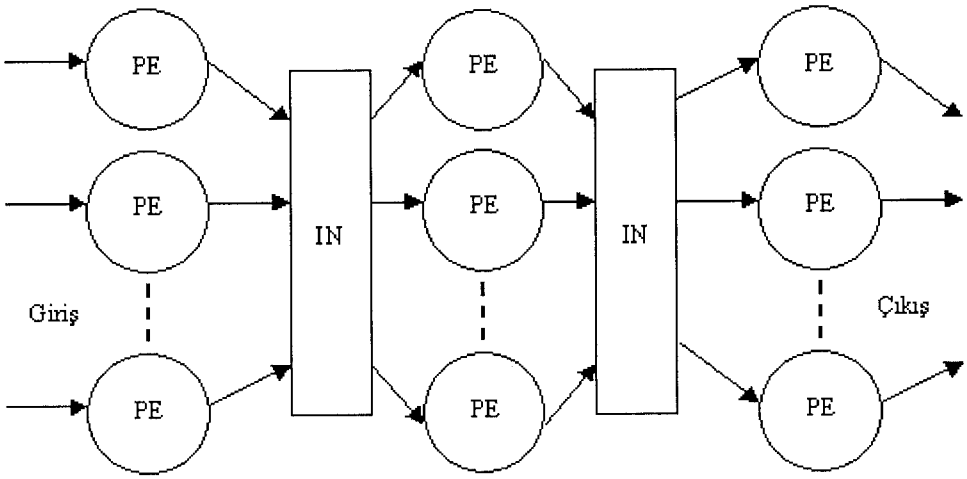
Pipeline vektör işlemciler, vektör işlenenlerini verimli olarak işleyebilir. Pipeline vektör işlemci ile dizi yada vektör işlemci arasındaki temel fark budur. Dizi işlemler komut tabanlıdır,

öte yandan pipeline vektör işlemler sürekli veri akışı tabanlıdır. Şekil 2.9 pipeline vektör işlemcilerin temel yapısını temsil eder.



Şekil 2.9 Pipeline vektör işlemcinin blok diyagramı [10]

Pipeline vektör işlemciler, pipeline'ları sürekli veri akışı sağlayacak birkaç hafıza modülünü kullanır. Verinin donanım tarafından daha sonra kullanılabilmesi ve veriyi akış içersinde idare için çoğunlukla vektörize edilmiş derleyici kullanılır.

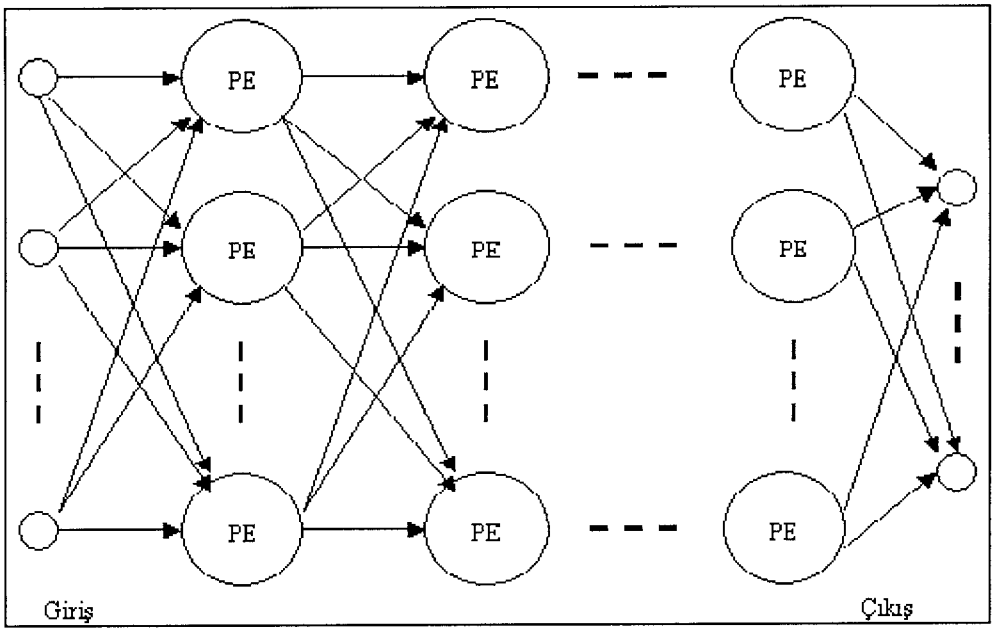


Şekil 2.10 Sistolik dizinin blok diyagramı [10]

Şekil 2.10, sistolik dizilerin yapısını göstermektedir. Bir sistolik dizide çok sayıda aynı özelliklere sahip işlemci elemanları (PE) bulunur. Her bir PE lokal depolama ile sınırlıdır ve her bir PE IN'de sadece komşu PE'lere bağlanabilir. Bundan amaç diziye yerleştirilmiş PE sayısının sınırlandırılmamasıdır. Böylece lineer yada iki boyutlu dizi olarak tüm PE'ler pipeline yapıda organize edilmiş olur.

Hybrid mimariler daha iyi performans sağlamak için farklı mimari türlerini içerir. Genel olarak iki farklı paralel hesap yöntemi vardır: Kontrol paralelliği (kontrolü dağıtmak) ve veri paralelliği. Kontrol paralelliğinde iki veya daha fazla paralel uygulamalar için MIMD makinalar idealdir. Farklı operasyonların farklı veriler üzerinde eşzamanlı icrası gerektiğinde MIMD uygundur. Bir MIMD bilgisayarda, her bir komut işlemci kendi komut işlem sırasını bağımsız olarak icra eder. Öte yandan, SIMD makinaları paralel veri uygulamaları için idealdir ve aynı operasyonların farklı veriler üzerinde eşzamanlı icrası için uygundurlar. MISD makinaları veri paralelizmi için uygundur. Pratikte asıl kazanç verinin paralelleştirilmesinden gelmiştir. Hesaptaki veri miktarına bağlı olarak paralellikten faydalanıldığı için veri paralelliği bu sonuca ulaşmıştır. Bununla beraber, uygulama programlarının çoğunda veri paralelizmden tam olarak faydalanmak mümkün olmamaktadır. Bundan dolaya kontrol ve veri paralelizminin ikisini birlikte kullanmak ihtiyaç olmaktadır. Örneğin, bazı uygulamalarda programlar veri paralelizminden faydalanan alt kısımlara ayrılmıştır ve tüm alt kısımlarla birlikte pipeline formunda kontrol paralelizminden faydalanmaktadır. İşlemcilerin bir grubu verileri toplamakta ve bazı öncü hesaplar yapmaktadır. Daha sonra bu işlemciler sonuçları daha detaylı hesap yapmak üzere ikinci grup işlemcilere gönderir. İkinci grup da kendi sonuçlarını üçüncü bir grup işlemcilere ve böylece nihai sonuçlar elde edilir. Böylece hem MIMD hem de SIMD mimari özelliklerini içeren paralel bilgisayarlarla geniş kapsamlı problemlere çözüm bulunabilmektedir.

Özel amaçlı mimarilere örnek, yapay sinir ağlarıdır (ANN: Artificial Neural Network). Yapay sinir ağları paralel işlem yapan, çok sayıda işlemci elemanlarından (PE) oluşur. Doğal bilgi takibi model tanımı gibi Von Neumann bilgisayarın bilgisayarların düşük performans gösterdiği durumlarda umut verici sonuçlar üretmektedir. Bu tip problemler insan performansına ulaşabilmek için yüksek işlem gücüne ihtiyaç duymaktadır. Gereken işlem gücünü ise şu şekilde elde eder: Paralel işlem yapan çok sayıda işlemci elemanını kullanarak, öğrenme, değişken koşullara adapte olma yeteneğine sahiptir. Şekil 2.11 bir yapay sinir ağı yapısını temsil eder.



Şekil 2.11 Bir yapay sinir ağının blok diyagramı [10]

Her bir PE biyolojik sinir hücrelerin bazı karakteristiklerini andırır. Çok sayıda girdi alır yada çok sayıda çıktı verir. Sayısal bir ağırlık her bir girdiye atanmıştır. Biyolojik nöronların benzer PE'nin tüm girdileri ağırlıklarıyla çarpılır ve nöronların aktivasyon seviyelerini belirlemek için toplanır. Aktivasyon seviyeleri belirlendiğinde, aktivasyon fonksiyonu çıktı sinyalini üretmek üzere uygulanır. Bir önceki katmanın tüm çıktıları sonraki katmanın girdileri olmaktadır. Bu işlem tüm ağ dolaşıldığında ve bazı kararlara erişilene kadar tekrar edilir.

Diğer bir özel amaç cihazı örneği bulanık mantıktır (fuzzy logic). Klasik iki değerli lojik (doğru-yanlış) gerekçelendirmenin formal prensipleriyle ilgileniyorsa bulanık mantık da yaklaşık gerekçelendirme prensipleri ile ilgilenir. Fuzzy lojik insan tarzı işlemlerle uğraşır ve klasik doğru-yanlış mantığının iyi sonuç vermediği sorunların üstesinden gelmeyi hedefler (doğru-yanlış mantığı insan tarzı işlemleri yansıtmayı hedeflemez). Ev uygulamalarından karar-destek sistemlerine kadar fuzzy lojik sistemleri geniş bir alanda kullanılmaktadır. Her ne kadar fuzzy lojik yazılım uygulamaları iyi sonuçlar verse de, fuzzy işlemcilere bağlı olarak; yüksek performanslı uygulamalar için bu tip hızlandırıcılar gerekmektedir [10].

2.2. Paylaşılmış Bellekli Sistemler (Shared Memory Systems)

Dağıtık sistem mimarisinin ilk zamanlarında herkesin hemfikir olduğu bir nokta vardı; fiziksel bellek paylaşımı olmayan makina programlarının farklı adres alanlarında icra olanağı. Bu bakış açısı üzerinde haberleşmenin ayrık adresler arasında mesaj iletimi yoluyla

yapılacağıdır. 1986 lar'da Kai Lee tarafından bugün dağıtık paylaşımlı bellek olarak bilinen DSM (Distrubitud Shared Memory) önerildi.

Özetle, bir yerel ağ tarafından bağlanmış iş istasyonları kümesinin tek sayfalı, sanal adres alanlı paylaşımı önerilmiştir. Basitçe, her bir sayfa sadece bir makina içersinde yer almaktaydı. Lokal sayfaya referans, etkin bellek hızında donanımda sağlanmaktaydı. Bir başka makinadaki sayfaya referans ise donanım sayfa hatasına sebep olmaktaydı ve bu sorun işletim sistemine gönderilmekteydi. Daha sonra işletim sistemi mesajı uzak ara makinaya göndermekte, gerekli sayfa bulununca ilgili işlemciye cevap gönderilmekteydi. Böylece hatalı komut işlemi yeniden başlamakta ve icra edilmekteydi [9].

Bu tasarım tarzı, geleneksel sanal bellek sistemlerine benzemektedir: Bir işlemci kendi içinde mevcut olmayan sayfaya erişim talebinde bulunduğunda hata oluşur ve işletim sistemi sayfayı bulur ve kendi içinde adresler. Kai Lee örneğinde ise sabit diskten bir sayfa getirmek yerine, işletim sistemi ağ üzerindeki diğer bir işlemciden getirilir. Kullanıcı işlemi için ise sistem geleneksel çoklu işlemcilerdeki gibi birden fazla işlemin istem durumunda paylaşımlı bellekten bilgi okuma ve yazma işlemine olanak sağlar. Tüm haberleşme ve senkronizasyon bellek vasıtası ile ve kullanıcı işlemlerinin gözlemlerinden uzak olarak sağlanır. Kai Lee 'nin sistemi hem kolay programlanabilir (mantıksal paylaşımlı bellek) hem de gerçeklenebilir yapıdadır (fiziksel paylaşımlı bellek içermiyor).

Her ne kadar böyle bir sistemin gerçeklenmesi ve programlanması kolay olsa da, çoğu uygulamalarda düşük performans gösterir. Zira ağ boyunca sayfaların iletimi performansı düşürür. Yakın zamanda dağıtık paylaşımlı bellek sistemlerini geliştirmek için birçok araştırma yapılmış ve çok sayıda teknik geliştirilmiştir. Önerilen tüm tekniklerde amaç, ağ trafiğini minimum yapmak ve bellek talebi ile cevabın gelmesi arasındaki süreyi kısaltmaktır.

Bir diğer yaklaşım, tüm adres alanını paylaşmak yerine sadece seçilmiş bir kısmını paylaşıma açmaktadır. Örneğin, birden fazla işlemci tarafından kullanılan değişkenler ya da veri yapıları paylaşıma açılır. Bu modelde her bir makinanın herhangi bir belleğe doğrudan erişimi düşünülmez. Bunun yerine paylaşımlı değişkenler kümesi erişim olanağı tanınır. Bu strateji, paylaşılması gereken veri miktarını azaltmakla kalmaz, çoğu durumda bu verilerin tipleri, hangilerinin uygulamada optimizasyon sağlanacağı bildiride hazır olur.

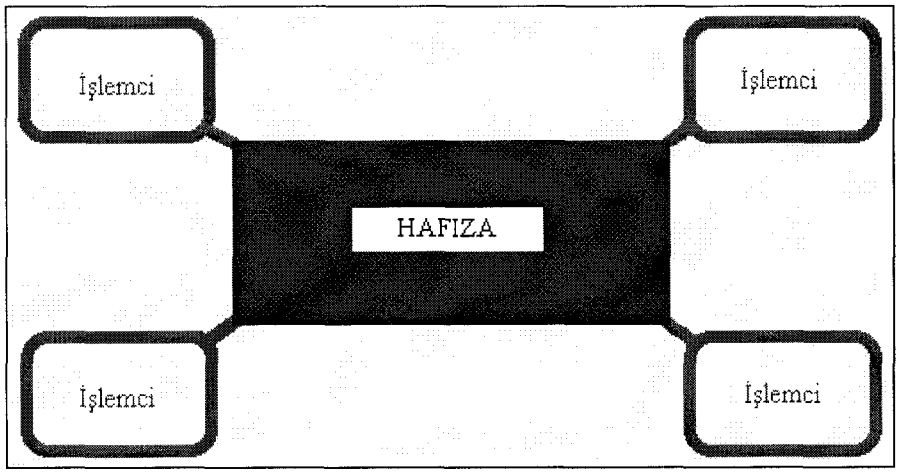
Bir diğer optimizasyon ise, paylaşımlı değişkenleri birden fazla makinaya kaydetmektir (çoğaltılmış değişken). Tüm sayfa yerine çoğaltılmış değişkenleri paylaşıma açarak çoklu işlemci simülasyon programı basitleştirilir. Ancak, kopya değişkenlerinin sürekli güncellenmesi gerekir. Potansiyel olarak okuma işlemleri çoklu kopya güncelleme protokolünü gerektirir.

Adres alanlarının yapılandırılması konusunda, sadece değişkenleri değil bunun yerine nesneleri paylaşıma sunması mümkündür. Paylaşımlı değişkenlerden farklı olarak her bir nesnede verilerin yanında metot olarak tanımlanan prosedürleri de içerir. Programlar nesneye ait verileri ancak nesnenin kendi metotlarını çağırarak değiştirilebilir. Bu tip bir kısaltma sayesinde çok çeşitli optimizasyonlar mümkün olmaktadır.

Her şeyi yazılım üzerine inşa etmenin sayfalandırma donanımı mantığı kullanmaya göre avantaj ve dezavantajları vardır. Çoğunlukla programcıya daha çok kısaltmalar getirir fakat daha iyi performans sağlar. Bu kısaltmaların çoğunluğu (örneğin nesneler üzerinde çalışmak) çok güzel yazılım mühendisliği pratiği sağlar.

2.2.1. Paylaşımlı bellek nedir?

Paylaşımlı bellek birden fazla işlemcinin belleği ortak kullanmasıdır. Bu sistemlerin bazıları tek bir yol üzerinde basit bir paylaşım yaparken, diğerleri oldukça karmaşık ön bellek (cache) yöntemleri kullanır. Bu makine türleri, DSM mimarisini anlama açısından oldukça önemlidir. Zira çoklu işlem mimarisindeki gelişmeler DSM mimarisine ilham vermiştir. Daha da ötesi, bir çok algoritma birbirine öyle çok benzemektedir ki, geliştirilen makinenin çoklu işlemci mi yoksa DSM 'in donanım uygulaması olan bir çoklu bilgisayar mı olduğunu ayırabilmek bile zordur. Çoklu bilgisayar sistemleri ile dağıtık paylaşımlı bellek sistemlerini karşılaştırıldığında, yazılım ağırlıklıdan donanım ağırlıklıya kadar çok farklı tasarım spektrumunun mevcuttur. DSM 'in nerede daha uyumlu olabileceği, bütün spektrum incelenmesiyle ortaya çıkar. Şekil 2.12 paylaşılmış belleğin genel yapısını temsil eder.



Şekil 2.12 Paylaşılmış belleğin genel yapısı

2.2.2. Çip üstü bellek (On-Chip)

Bilgisayarların dış belleklerinin olduğu bilinse de, işlemci ve kendine ait bellekleri olan entegreler mevcuttur. Bu tip entegreler, yaygın biçimde üretilmekte ve arabalarda, tezgahlarda hatta oyuncaklarda bile kullanılmaktadır. Bu tür tasarımlarda, entegrenin işlemci kısmı bellek kısmına veri ve adres hatları ile bağlıdır.

Öte yandan, birden fazla işlemcinin aynı bellek alanını paylaştığını varsayılırsa, bu tip bir tasarım mümkün olmakla birlikte, sistem oldukça karmaşık, pahalı ve uygunsuz olacaktır. 100 tane işlemcinin tek bir belleğe eriştiği bir tasarım ise, mühendislik sebeplerden dolayı olanaksızdır.

2.2.3. Bus tabanlı çoklu işlemciler

İşlemci ile bellek arasında bağlantı kabloları mevcuttur. Bunların bazıları işlemcinin okuma ve yazma operasyonlarını sağlamak için, bazıları veri alışverişinde, kalanları ise transferin kontrolünde kullanılmaktadır. Bu tür kablo kümesine "bus" denir. Çip üstü bellekte bus entegre üzerindedir. Oysa çoğunlukla "bus" entegre dışındadır ve kartlar üzerinde işlemci, bellekler ve I/O kontrolleri arasında bağlantıyı sağlarlar. Kişisel bilgisayarda ise bus ana kart üzerinde işlemci, bellek, I/O ve diğer cihazlar arasında bağlantıyı sağlar.

Herhangi bir işlemci bellekten okuma işlemi yapmak istediğinde, okumak istediği kelimenin adresini bus'a göndermekte ve bus kontrol alanına okuma işlemi yapmak istediğine dair bir işaret göndermektedir. Bellek, talebi aldığında, cevaben kelimeyi bus'a vermekte ve diğer bir kontrol alanından talep edilen kelimenin hazır olduğunu duyurmaktadır. Nihayetinde işlemci istenen kelimeyi okumaktadır. Yazma işlemleri de benzer şekilde gerçeklenir.

Birden fazla işlemcinin aynı anda belleğe erişimini engellemek için bus yönetim mekanizması gerekecektir. Bunun için bir çok yöntem mevcuttur. Örneğin, işlemci belleğe erişim için önce izin alma şartı konulabilir. Öncelikle belleğe erişim izni alınmak şartı ile bus cihazlar tarafından kullanıma açılabilir. Erişim izni için nasıl bir yöntem kullanılacağı ise ayrıca belirlenmelidir. Merkezi bir yöntemle bus yönetim cihazı kullanılacağı gibi, merkezi olmayan bir yöntem ile ilk talebi gönderen işlemciye öncelik verme mantığı da kullanılabilir.

Bus yükünü azaltmanın bilinen bir yöntemi, her bir işlemciyi snoopy cache ile donatmaktır (bus'u gizlice izlediği için böyle bir isim verilmiştir). Caching konusunda farklı işlemcilere ait önbelleklerde aynı adres için farklı değerler bulundurulmaması önerilmektedir [9].

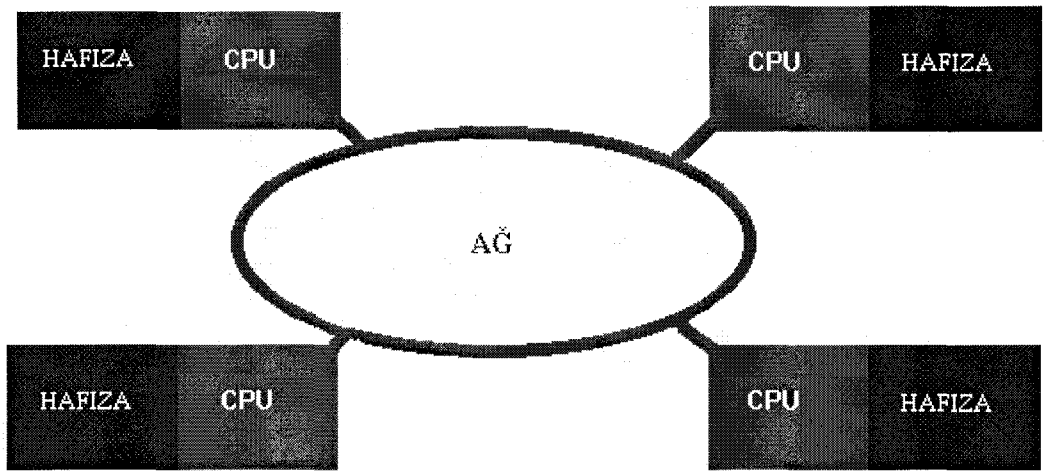
2.3. Dağıtılmış Bellekli Sistemler

1945'den günümüze bilgisayar sistemlerinde ciddi devrimler meydana gelmiştir. İlk bilgisayarlar büyük yer kaplıyorlardı ve pahalıydı. Sonuç olarak, organizasyonlarda sınırlı sayıda bilgisayar bulunuyordu ve bunları birbirlerine bağlamak ihtiyacı yoktu. 1980'lerin ortalarından günüm üze, teknolojideki iki önemli değişiklik bu durumun değişmesine neden oldu. Birincisi, güçlü mikroişlemcilerin geliştirilmesidir. Başlangıçta 8 bitlik işlemciler kullanılırken daha sonra 16, 32 ve 64 bit işlemciler yaygın hale geldi. Bilgisayar teknolojisindeki gelişmeler herhangi bir endüstri ile kıyaslanamayacak şekilde hızlı olmuştur. Saniyede bir komut icra edebilen 10.000.000$'lık bir makinadan, günümüzde saniyede 10.000.000 komut icra eden ve maliyeti 1000$ mertebesine düşen makinalara ulaştık. İkinci önemli gelişme ise yüksek hızlı bilgisayar ağlarının gelişmesidir. Yerel alan ağları (LAN; Local Area Network), bir bina içersinde onlarca bilgisayar arasında milisaniyeler mertebesinde veri iletimine imkan tanımaktadır. Geniş alan ağları (WAN; Wide Area Network) ise dünya üzerindeki milyonlarca bilgisayarı 64kb/ps mertebesinde veri transferine imkan tanımaktadır. Bütün bu teknolojilerin sonucu, çok sayıdaki işlemcinin birbirlerine yüksek hızlı ağ oluşturacak şekilde bağlanabilmesidir. Bunlara merkezi sistemden farklı olarak dağıtık sistemler denir.

2.3.1. Dağıtık sistem nedir?

Bir dağıtık sistem, kullanıcılarının her birine tek bilgisayar gibi görünen birbirinden bağımsız bilgisayarlar kümesidir. Bu tanımı daha iyi anlamak için örneklerle açıklayabiliriz. Bir üniversite kampüsündeki ağı varsayalım. Her bir kullanıcı kendi iş istasyonu bilgisayarının yanı sıra, ağa bağlı bilgisayarları, hiçbir kullanıcıya bağlı olmayan ancak gerektiğinde kullanıcılara dinamik olarak atanabilen işlemciler havuzu olarak ta kullanabilir. Böyle bir sistemde, bilgisayarların tümü aynı yolu kullanarak bütün dosyalara ulaşabilir. Bunun yanı sıra, kullanıcı bir komut yazdığında sistem, komutu icra için en uygun işlemciye gönderir.

Diğer bir örnek: Dünya çapında yüzlerce şubesi olan bir banka. Her bir şubenin yerel hesapları kaydeden ve yerel işlemleri yapan ana bilgisayarı olsun. Buna ek olarak her bilgisayarın ana merkezdeki bir bilgisayar arcılığı ile diğer şubeler ile haberleşebildiğini varsayalım. Eğer hesap işlemleri müşterinin yerinden bağımsız olarak yapılabiliyorsa ve müşteri bu sistemle merkezi sistem arasındaki farkı hesap işlemleri arasında fark edemiyorsa, bu sistem dağıtık sisteme örnek kabul edilebilir. Şekil 2.12 dağıtık sistem modelini göstermektedir.



Şekil 2.13 Dağıtık Sistem Modeli

2.3.2. Dağıtık sistemlerin merkezi sistemlere göre avantajları

Merkezi olmayan sistemlere yönelişin temel amacı ekonomik sebeplerdir. 25 yıl önce Herb Grosch şu kuralı ileri sürmüştür: Bir işlemcinin hesap gücü fiyatının karesi ile doğru orantılıdır [9]. İki kat daha fazla ödeyerek dört kat daha güçlü performans elde edilebilir. Bu gözlem eski sistemler için makul olmuştur ve birçok organizasyonun ödeyebilecekleri en büyük maliyetlerde bilgisayarlar alınmıştır. Ancak, mikroişlemci teknolojisindeki ilerlemeler ile Grosch kanunu geçerliliğini yitirmiştir. Artık birkaç yüz dolarlık bir işlemci alarak 1980'lerin sistemlerinden daha fazla komut icrasına sahip olunabilir. Eğer iki kat fazla ödemek istiyorsanız, aynı işlemciyi almakla birlikte daha yüksek saat darbesine sahip olursunuz. Sonuç olarak, en iyi maliyet-performans çözümü ucuz maliyetli işlemcileri tek çatı altında toplamaktır ve dolayısıyla dağıtık sistemler daha iyi fiyat/performans oranına imkan tanımaktadır.

2.3.3. Bilgisayar ağ yapıları

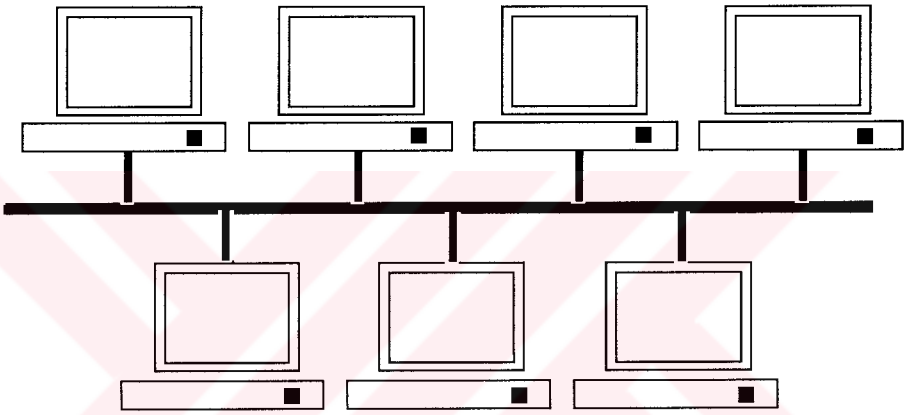
Bilgisayar ağları, bilgisayarları birbirine bağlayan ve bilgisayarlar arası veri iletişimini gerçekleyen yapılardır. Kullanılan pek çok bağlantı şekli vardır. Buna ağ mimarisi denir. En çok kullanılan ağ mimarileri şunlardır:

- Doğrusal yapı (bus)
- Yıldız yapı (star)
- Halka yapı (ring)

2.3.3.1. Doğrusal (Bus) yapı

Tüm bilgisayarlar sıra ile tek kablonun üzerinden bağlanır. Kablonun her iki ucunda elektriksel sinyallerin kablo dışına çıkmasını önlemek amacı ile sonlandırıcılar bulunur.

Ethernet sistemlerinde bu yapı kullanılır. Bilgisayarlar Ethernet kartlarına bağlanan T konnektörler (BNC) ile ana kabloya bağlanır. Kabloda meydana gelen en ufak bir sorunda tüm ağ kullanışsız hale gelir. Doğrusal yapı kullanan bir ağda veri paketleri doğrudan tüm ağa gönderilir. Veri paketlerindeki adres bilgileri sayesinde, ilgili hedef bilgisayar ağdaki bu verinin kendine gönderildiğini anlar ve veriyi alır. Ağda dolaşan paketlerin tüm bilgisayarlar tarafından dinlenebildiğini göz önüne alırsak, bu yapının güvenlik sorunları doğuracağı düşünülebilir. Doğrusal yapı, birçok küçük yerel ağda (LAN) kullanılan nispeten ucuz yapılardır. Şekil 2.13'de bu tür bir ağ şeması gösterilmiştir.

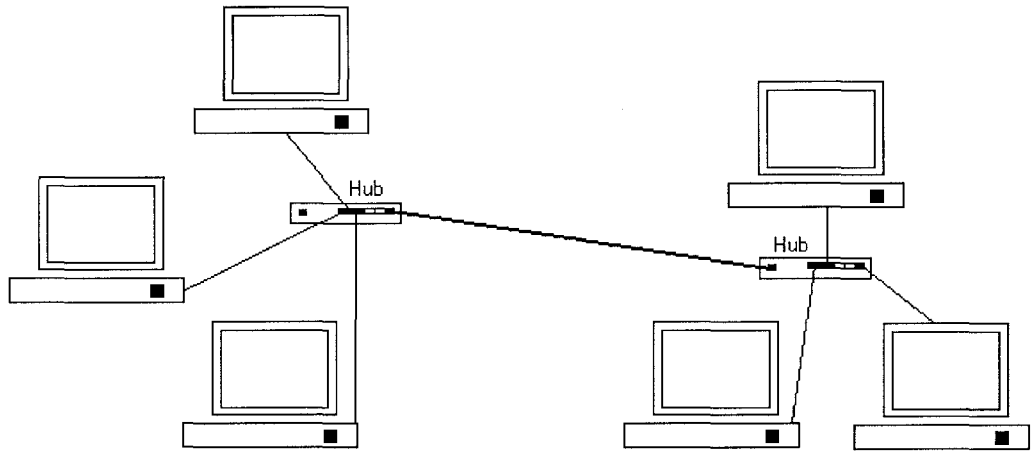


Şekil 2.14 Doğrusal yapıya sahip ağ modeli [1]

2.3.3.2. Yıldız (Star) yapı

Bu tür sistemde tüm bilgisayarlar merkezi bir üniteye (hub) bağlanır. Tüm paketler bu merkez üzerinden geçmektedir. Kablolarda bir sorun çıkarsa sadece sorun çıkan kabloya bağlı bilgisayar bağlantısı kesilir.

Bu yapıyı bir otoyol kavşağına benzetebiliriz. Bir araba, kavşağı oluşturan yolların birinden diğerine geçmek için mutlaka kavşağı kullanmak zorundadır. Kavşak bu modelde bir nevi denetleyici vazifesi görür. Şekil 2.14'de de görüldüğü gibi bu sisteme donanımın elverdiği sürece ek bilgisayarlar bağlanabilir.

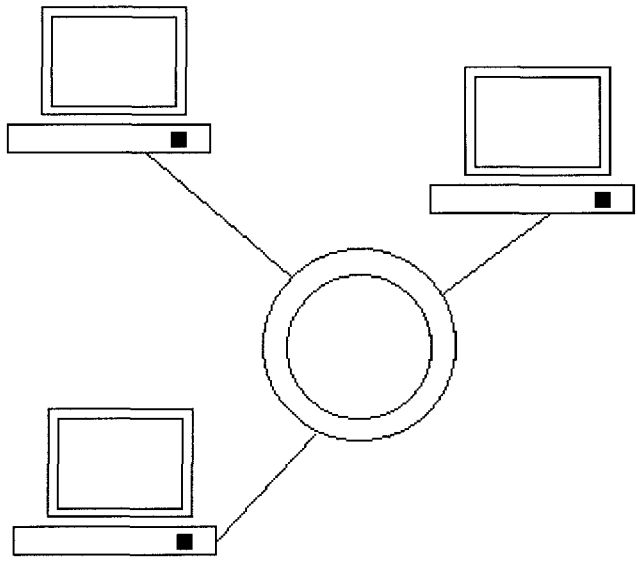


Şekil 2.15 Yıldız yapıya sahip ağ modeli [1]

Yıldız ağlarda ortadaki trafik akışını düzenleyen birim, hub yerine başka bir bilgisayar da olabilir. Merkezdeki birim trafiğin akışını yönlendirdiği için bu bilgisayarın ya da hub'ın bozulması ve görevini yapamaz hale gelmesi tüm iletişimi felç eder. Yıldız ağ mimarisinde bir bilgisayardan giden veri, merkezi birime ulaştırıldıktan sonra hedefe yönlendirildiği için diğer bilgisayarların kendilerine ait olmayan bilgileri okumaları mümkün olmaz.

2.3.3.3. Halka (Ring) yapı

Tüm bilgisayarlar tek kablo üzerinde bağlıdır, fakat doğrusal yapıda olduğu gibi uçlarda sonlandırıcılar yoktur, uçlar birleştirilerek bir halkayı tamamlarlar. Kablo kopmasında tüm ağ kullanım dışı kalır. Token Ring bu yapıyı kullanır. Şekil 2.15'de halka yapıya sahip bir ağ modeli gösterilmiştir.

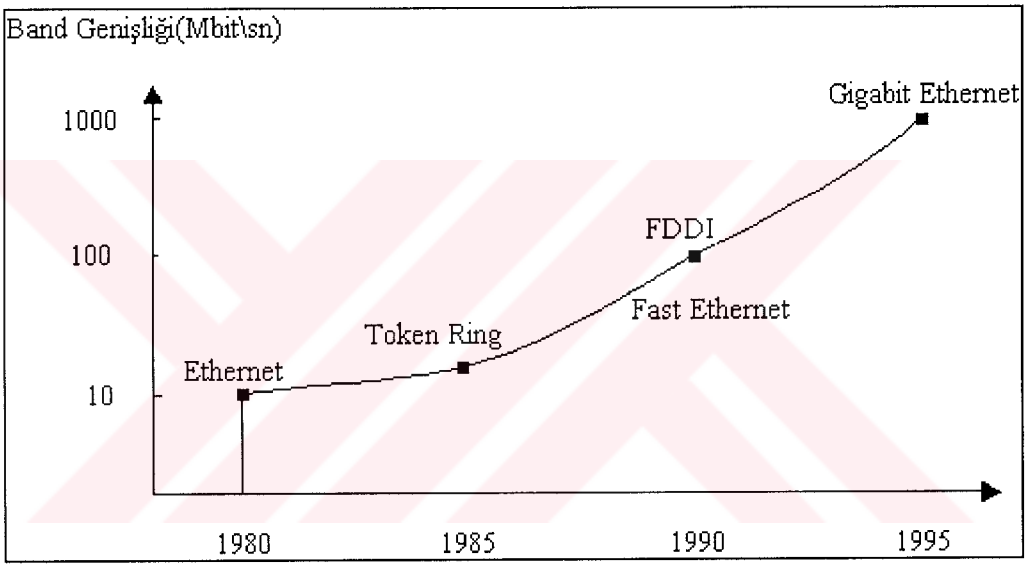


Şekil 2.16 Halka yapıya sahip ağ modeli [1]

Halka mimarinin en büyük özelliği her bilgisayarın ağdan eşit faydalanma hakkı olmasıdır. Bu ağlarda bir jeton (token) yardımı ile iletişim yapılır. Ağ üzerinde bir jeton kalıbı sürekli döner. Eğer bir bilgisayar diğerine veri yollamak istiyorsa bu jetonu bekler. Jeton kendisine ulaştığı anda bu yakalanan jeton ile birlikte veriyi ağ üzerindeki hedef bilgisayara yollar.

3. SANAL PARALEL MAKİNA; PVM (Parallel Virtual Machine)

Saniyede milyonlarca işlem yapan tek işlemcili iş istasyonları yaygındır ve hesap yetenekleri her geçen gün güçlenmektedir. Uygun bir yüksek hızlı ağ ile bu bilgisayarlar bağlandığında müşterek hesap gücü, birçok yoğun uygulamanın çözümünde kullanılabilir. Hızlı ağ ile birbirine bağlanmış bilgisayarlardan oluşan bu tür bir bilgisayar sistemi süper bilgisayar düzeyinde hesaplama gücü bile sağlayabilir. Dağıtılmış hesaplama modeline uyan bu sistemin etkin olabilmesi için yüksek iletişim hızları gereklidir. Şekil 3.1'de ağ hesaplama hızının yıllara göre gelişimi görülmektedir.



Şekil 3.1 Ağ hesaplama hızının gelişimi [2]

- Ethernet: Popüler yerel alan paket anahtarlama teknolojisi olan Ethernet, 10 Mbit/sn dağıtım özelliğine sahiptir.
- Token Ring: Halka şeklindeki hat üzerinde bir jeton dolaştırılır ve paket bu jeton ile taşınır. 4-16 Mbit/sn hıza sahiptir.
- Fiber Dağıtılmış Veri Arayüzü (FDDI; Fiber Distributed Data Interface): İki istasyon arasında optik fiber iletimi kullanan 100Mbit/sn token ring modelidir ve güvenilirliği sağlamak için yedek ring içerir.
- Fast Ethernet: 100Mbit/sn hıza sahiptir.
- Gigabit Ethernet: 1Gbit/sn hıza sahiptir. Fiyatı yüksek olduğundan henüz geniş bir alanda kullanılmamaktadır [2].

Tek işlemcili sistemler seri bilgisayar olarak bilinir. Bu tür sistemler için yazılan programlar ile birim zamanda bir talimat gerçekleştirilir. İşlemci hızı donanıma bağlıdır. Elektrik akımının hızı bakır tel limitleri (9 cm/nsn) için düşünülürse; en hızlı bilgisayarlar bir talimatı saniyenin milyarda dokuzu sürede gerçekleştirir [4]. Bir talimatın bu kadar kısa bir sürede gerçekleştirilmesini yeterince hızlı olduğu düşünülebilir ancak, daha hızlı işlem gerektiren birçok problem sınıfı vardır:

- Simülasyon ve modelleme problemleri; başarılı yaklaşıklara dayalıdır, daha fazla hesaplama ve hassasiyet ister.
- Büyük miktarda verinin hesaplamasına dayalı problemler; görüntü işleme, veri tabanı oluşturma, yarı iletken ve süper iletken modellemesi, kuantum mekanikleri, sismik, görünüm v.s. [4].

Yüksek hesap gücü isteyen bu problemlerin çözümü için dağıtık hesaplama modeli cazip hale gelmekte ve bu doğrultuda paralel programlar yazılmaktadır.

Genel olarak paralel programlama modeli şunları içerir:

- Bir algoritmayı veya veriyi parçalara bölmek
- Farklı işlemcilere eşzamanlı çalışan görev parçalarını dağıtmak
- İşlemcilerin iletişimini koordine etmek

Paralel programlamada problem parçalara bölünür. Daha sonra bu görev parçaları sisteme bağlı diğer işlemcilere dağıtılmaktadır. Bu görev, sistemi yöneten bir servis programı tarafından yapılır. Servis programı, dağıtılan görev parçalarının diğer işlemcilerde icrasını başlatır ve görevleri tamamlandığında sonuçları geri toplar. Paralel programlamadaki en büyük zorluk, bütün bu olayları gerçekleyebilecek paralel programların yazılmasıdır.

3.1. PVM Yazılım Paketi

PVM, paralel bir bilgisayar gibi görünecek şekilde bir ağa bağlanan bilgisayarların heterojen olarak toplanmasına imkan veren bir yazılım paketidir. Hedef, ağa bağlı bilgisayarları bir araya getirip hesap yeteneği güçlü sanal paralel bir bilgisayar oluşturmaktır. Kullanıcının sanal makinasında toplanan donanım tek işlemcili bilgisayarlar, vektör makinaları veya paralel süper bilgisayarlar olabilir. Sistemi oluşturan makinaların hepsi tek tip (homojen) veya farklı tiplerde (heterojen) mimarilere sahip olabilir. PVM sistemi aynı odada bilgisayarları bağlayan bir yerel ağ (LAN) kadar küçük veya dünya çapında bilgisayarları bağlayan İnternet kadar büyük olabilir. Merkezi bir kontrol altında farklı bilgisayar mimarilerini bir araya getirmeye yönelik bu kabiliyet sayesinde PVM kullanıcısı, bir problemi alt görevlere bölebilir ve her birini

icra edilmek üzere en uygun işlemci mimarisine atayabilir. Kullanıcının görevleri prosesleri başlatıp bitirmek, veri paylaşımı ve senkronizasyondur. Görev boyutuyla ilgili bir sınırlama olmamakla birlikte iki bilgisayar arasında veri göndermek için gerekli zaman nedeniyle görevlerin boyutlarına göre PVM'in etkin kullanılması önerilir [2].

3.1.1. PVM'in tarihçesi

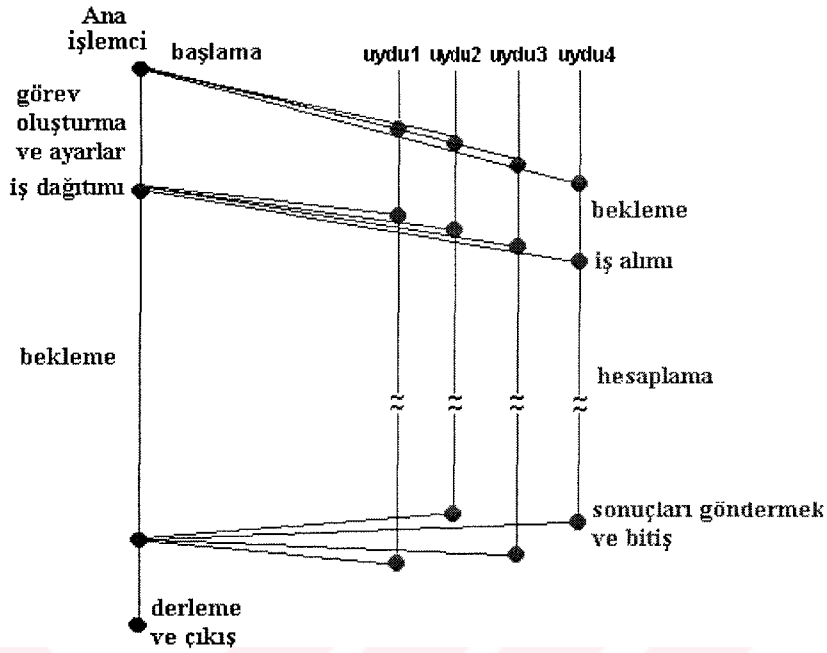
PVM'in ilk sürümü 1989 yazında Oak Ridge Ulusal Laboratuarı'nda yazıldı. Dağıtım olmadı ama laboratuardaki uygulamalarda kullanıldı. 2. sürüm Şubat 1991'de Tennesse Üniversitesi'nde yazıldı ve aynı yılın mart ayında dağıtıldı. 1.5 yıl sonra 2. sürüme ilavelere ihtiyaç duyuldu ve 3. sürüm Eylül 1992'de yazılmaya başlandı. Yazılımın ilk kullanımı 1993 Mart'ıdır. Hali hazırdaki sürüm 3.4 olup müteakip sürümler beklenmektedir [2].

PVM'in yüklenmesi özel izin veya ücret ödemek gerekmemektedir. Yüklenmesi kolaydır ve sadece birkaç Mbyte disk alanı gerektirir. C ve Fortran'ı destekleyen PVM, geleceğin hesaplama gereksinimleri için dağıtılmış hesaplama imkanları sunar. Dağıtılmış hesaplama, mevcut donanımı kullanır ve maliyetleri düşük tutar. Her görev en uygun mimariye atanarak performans optimize edilir. Sanal bilgisayar kaynakları aşama aşama büyüyebilir, en son hesap ve ağ teknolojilerinin avantajlarını alabilir.

PVM, süratle standart hale gelen Mesaj Geçiş Kütüphanesini (MPI) kullanmaktadır. Paralel işlemlerde veri, ortak çalışan görevler arasında değiş tokuş edilir. Standartlaşan bu yapı ile geliştirilen uygulamaların diğer mimarilere taşınması da standart hale gelmektedir. PVM'in mesaj geçiş kütüphanesini kullanması onu taşınabilir yapmıştır. Paralel program uygulamaları için birçok imkan sunan PVM, dünya çapında kabul görmekte ve büyük problemlerin çözümünde kullanılmaktadır.

3.1.2. PVM uygulamalarının yazılması

PVM ile programlama iki temel modelde yapılmaktadır. İlki, ana/uydu (master-slave) modelidir. Ana görev veya makine problem üzerinde çalışacak diğer görevleri veya makinaları başlatır, başlangıç verilerini her göreve veya makinaya gönderir, çalışmalarına koordine eder ve işi biten görev veya makinalardan sonuçları toplar. Bu modele göre ana program uygulamayı başlatır, dağıtılacak görevleri oluşturur ve bunları uydulara dağıtır, sonuçları toplar ve derler. Uydu veya işçi programları ise kendilerine verilen hesaplamaları yaparlar. Şekil 3.2'de ana/uydu modeline ait paralel çalışma (senkronizasyon) gösterilmiştir.



Şekil 3.2 Senkronizasyon

İkincisi evsiz (hostless) modeldir. Bu modelde başlangıç görevi, kendi kopyalarını üretir ve ardından problemin kendi kısmı üzerinde çalışmaya başlar. Giriş ve çıkış şartları genellikle ayrık görevlerce yürütülür. Ancak sonuçların tüm görevlere geri yollanması gerekiyorsa tek görevle tüm sonuçları toplatmak daha faydalı olabilir [2].

PVM uygulamaları hem C, hem de Fortran 77'de yazılabilir. PVM altında bir programı çalıştırmak için kullanıcı, görevleri diğer bilgisayarlara dağıtan PVM kütüphane rutinlerine çağrılar ekler.

Çalışma boyunca diğer herhangi bir PVM fonksiyonu çağrılmadan önce bir görevin kendisini PVM'e kaydetmesi gerekir. Böylece her bir görev ayrı bir kimlik numarası alır. Bu kimlik numaraları mesaj iletişiminde kullanılır. Herhangi bir görev PVM altında işini bitirdiğinde PVM yöneticisini sanal makinayı terk ettiğine dair bilgilendirmelidir. Bu olay halihazırda çalışmakta olan diğer görevleri etkilemez.

3.1.3. PVM konsolu

PVM uygulamaların çalıştırılması ve konfigürasyon için PVM konsolu kullanılır. Konsol ile sanal makinanın konfigürasyonu, PVM görevlerin başlatılması ve durdurulması, bilgi ve hata mesajlarının alınması yapılabilir. Konsol, PVM'in çalıştığı herhangi bir makinada başlatılabilir ve durdurulabilir. Konsol '*pvm>*' prompt ile cevap verir.

Bir PVM uygulamasının çalıştırılması için uygulamanın tüm kısımlarının her mimari için derlenmesi gerekir. Derlenen uygulama uygun dizinlere yerleştirildikten sonra sanal makina ayarlanır (konfigürasyon yapılır). Bunun en uygun yolu; sanal makinaya dahil edilecek bilgisayarların listesini içeren bir makina (host) dosyası oluşturmaktır. Bu dosya kullanıcının ev dizininde olmalıdır ve adı '.rhosts' şeklinde olmalıdır. Dosyanın yapısı şöyledir: Her makina ismi ayrı satırda olmalıdır. Boş satırlar görmezden gelinir ve '#' ile başlayan her satır bir yorumdur. Şekil 3.5'de örnek bir host dosyası görülmektedir.

```
# Örnek host dosya
simav.dumlupinar.edu.tr
emet.dumlupinar.edu.tr
gediz.dumlupinar.edu.tr
sogut.dumlupinar.edu.tr
merkez.dumlupinar.edu.tr
```

Şekil 3.3 Örnek host dosyası

Sanal makinaya konsoldan bilgisayar eklenip çıkarılabilir, sanal makina dağılımı istenebilir ve çalışan görevlerin durumu kontrol edilebilir. Hali hazırdaki sanal makine konfigürasyonu görmek için 'conf' komutu kullanılır. Bu listede host ismi, PVM yönetici görev kimliği (DTID), mimari tipi ve bilgisayarların hız oranı bulunur. Şekil 3.4'de örnek bir dağılım listesi verilmiştir.

```
pvm> conf
6 hosts, 1 data format
        HOST        DTID        ARCH      SPEED
     makina1       40000        SUN4       1000
     makina2       80000        SUN4       1000
     makina3       c0000        SUN4       1000
     makina4      100000        SUN4       1000
     makina5      140000        SUN4       1000
     makina6      180000        SUN4       1000
```

Şekil 3.4 Örnek sanal makina dağılımı (konfigürasyonu)

Görevler kullanıcının seçtiği makina kümesinde çalışır. Kümedeki makinalar çalışma esnasında değiştirilebilir. Sanal makinaya yeni bilgisayarlar eklemek yada çıkarmak için 'add' ve 'delete' komutları kullanılır. Şekil 3.5.a ve Şekil 3.5.b'de bu komutlara örnek verilmiştir.

```
pvm> add makina7
                    HOST         DTID
                 makina7       1c0000
```

Şekil 3.5.a Makina ekleme

```
pvm> delete makina2
                    HOST       STATUS
                 makina2      deleted
```

Şekil 3.5.b Makina çıkarma

PVM uygulaması sanal paralel bilgisayardaki makinaların herhangi birinde başlatılabilir. Bunu PVM'in başlatıldığı makinadan yapmak için 'ctrl z' ile PVM konsolu arka plana atılabilir veya 'quit' komutu kullanılır. Quit komutu sadece konsol programını bitirir, PVM görevi çalışmaya devam eder. Kullanıcı uygulamanın bir kopyasını başlattığında uygulamanın icrası başlar. Ana işlemci PVM görevlerini çalıştırır. Bu görevler içinde problemi çözmek üzere haberleşip hesaplama yapan birçok aktif görevler olabilir. Görev, PVM 'de paralellik birimidir. İletişim ve hesaplama arasında gidip gelen bağımsız bir kontroldür. Görevler, sistem tarafından verilen görev kimlik numaraları (TID) ile etkileşirler. Son olarak görevler işini bitirdiğinde sahip ile birlikte PVM' den çıkarlar [2].

3.1.4. PVM uygulamalarının icrası

Bir PVM uygulamasının çalıştırılması için uygulamanın tüm bilgisayarlarda derlenmesi veya icra edilebilir dosyaların her bilgisayar kopyalanması gerekir. Derleme için 'aimk' komutundan yararlanılır. Komut satırına '*aimk program ismi*' yazılır. Derlenen uygulamanın icra edilebilir dosyası pvm kütüphanesi altındaki bin/LINUX dizinine kopyalanmalıdır. Derleme yapıldıktan sonra pvm konsolu başlatılır ve sanal makina ayarlanır. Uygulamayı konsoldan başlatmak için '*spawn*' komutu kullanılır. '*pvm> spawn program-ismi*' komutu program çıktısını ekrana yönlendirecektir.

Sanal makinanın işi bittiğinde 'halt' komutu girilmelidir. Bu komut tüm PVM görevlerini sonlandırır, sanal makinayı kapatır ve konsoldan çıkar. PVM' i durdurmanın tavsiye edilen yolu budur.

3.1.5. Mesaj geçişi

PVM uygulamalarında işlemciler arasında veri iletişimi mesaj geçişi ile gerçekleşmektedir. Böylece müşterek görevler arasında veri paylaşılır. PVM' de mesaj geçişi üç aşamada gerçekleşir:

- *pvmf_init_send* fonksiyonu kullanılarak bir buffer başlatılır.
- *pvmfback* komutu kullanılarak buffer'a veri konur.
- *pvmfsend* veya *pvmfcast* kullanılarak buffer'ın içeriği diğer bir prosese gönderilir.

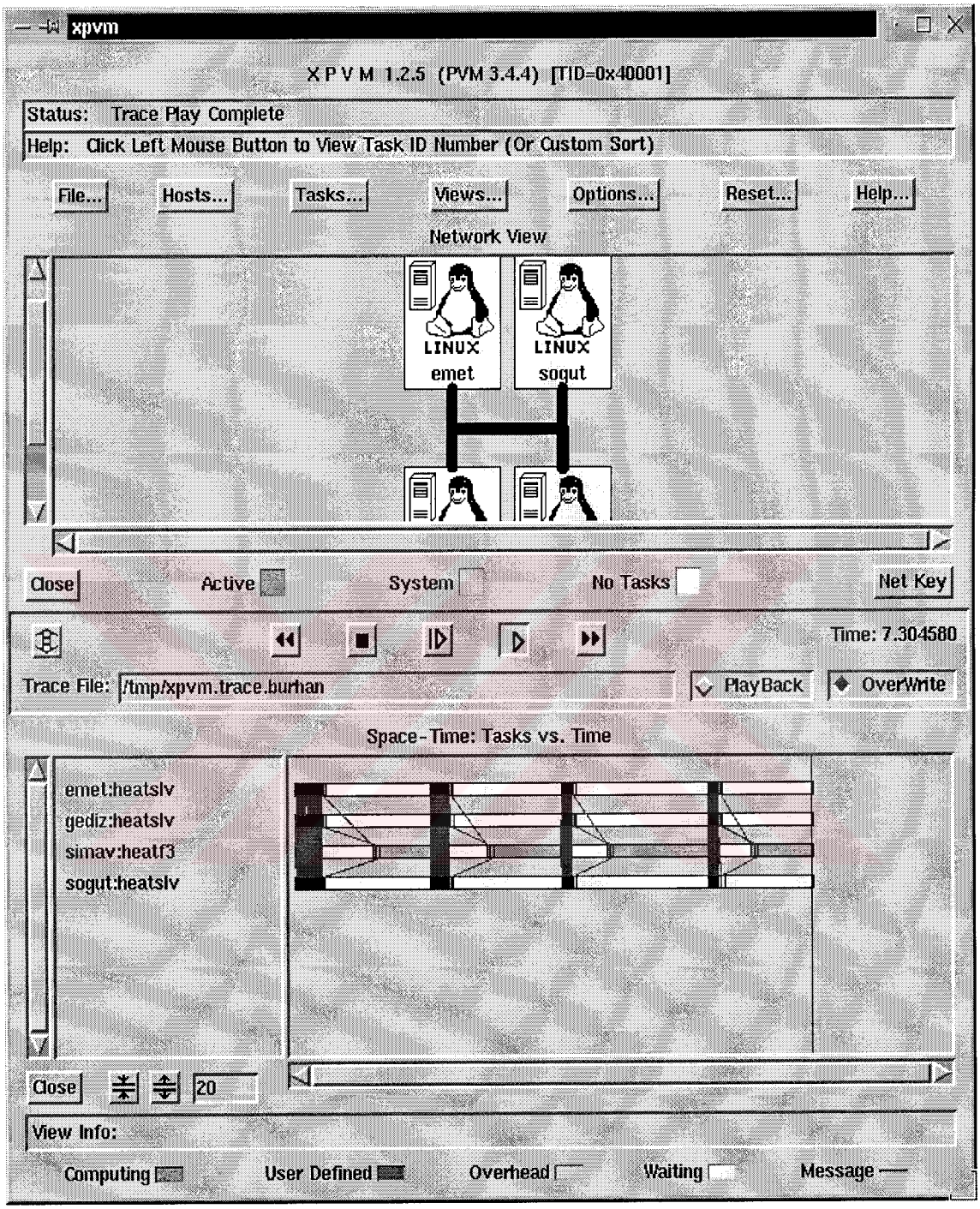
Mesaj alışı ise iki aşamalı bir işlemdir:

- *pvmrecv* veya *pvmnrecv* komutları çağrılır.
- *pvmunpack* kullanılarak mesaj alınır.

3.2. XPVM: PVM İçin Grafiksel Konsol ve Gözlemleme

XPVM, PVM için hazırlanmış bir grafiksel arayüzdür. PVM programlarının icrası sırasında ağın durumu, uzay-zaman bakışı ve görev takibi gibi bazı animasyonların on-line veya off-line olarak gözlemlenmesine imkan sağlar. Performansın gözlemlenmesi veya programın hatalardan arındırılması için grafiksel arayüz çok kıymetli bilgiler sunar.

XPVM kullanan bir programda, kullanıcının ihtiyaç duyduğu tek şey 3.3 ya da yukarı versiyon bir PVM kütüphanesidir. PVM 3.3 versiyon ve yukarısı çalışma zamanında iz-sürme bilgisi ile donatılmıştır. İz-sürme bilgisi ile kullanıcı gerçek zamanda analiz yapma olanağına sahiptir. Şekil 3.6'da XPVM ara yüzü gösterilmiştir.



Şekil 3.6 XPVM Arayüzü

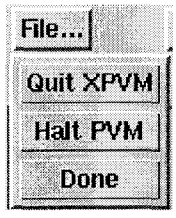
3.2.1. Konsol komut menüsü

XPVM, PVM konsol komutlarına erişim için 'işaretle ve tıkla' imkanı sağlar. Bu erişim ile birçok komutun icrası menüden başlatılabilir. Şekil 3.7'de konsol komut menüsü görülmektedir.

Şekil 3.7 Konsol komut menüsü

3.2.1.1. File menüsü

File menüsü PVM'in durdurulmasını sağlayan görev çubuğudur.



Şekil 3.8 File menüsü

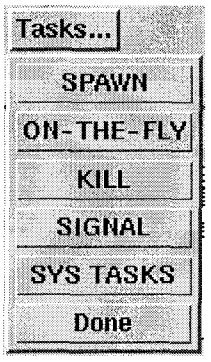
3.2.1.2. Hosts menüsü

Host menüsü sanal makineyi düzenlemek üzere kullanıcılara terminal ekleme ve çıkarma imkanı sağlar. Eklenmek istenen makina ismi listeye eklenerek ayarlanır.

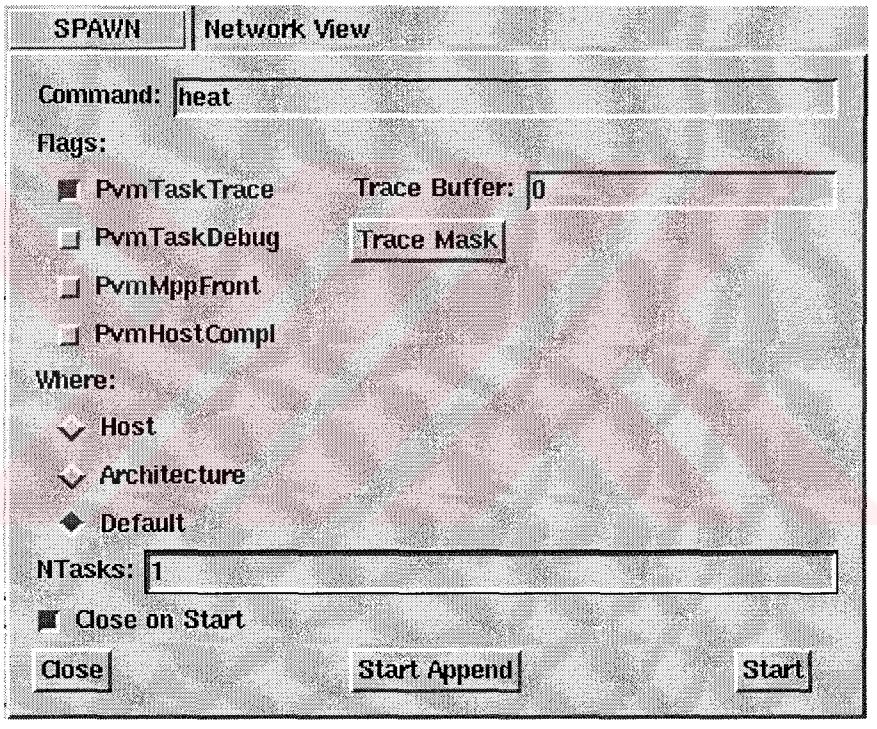
Şekil 3.9 Host menüsü

3.2.1.3. Görev (Task) menüsü

Görev menüsünde PVM uygulamasının başlatılması, uygulamadaki sinyalleşme ve uygulamanın durdurulması gibi görev çubukları mevcuttur. PVM uygulamasını başlatmak için '*spawn*' butonu ile gelen pencerede command boşluğuna uygulamanın ismi yazılır ve '*start*' butonu ile uygulama başlatılır.



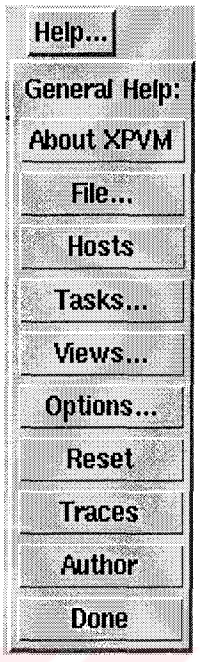
Şekil 3.10 Görev menüsü



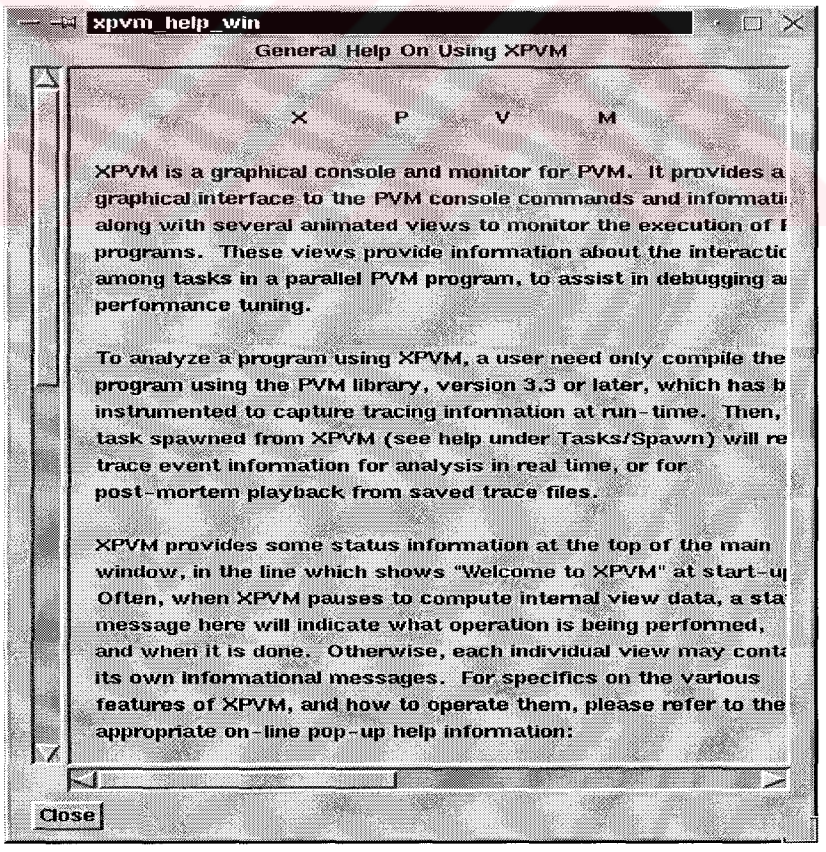
Şekil 3.11 Spawn menüsü

3.2.1.4. Yardım menüsü

XPVM kullanımı ve özellikleri için tanıtıcı bilgilerin yer aldığı menüdür.



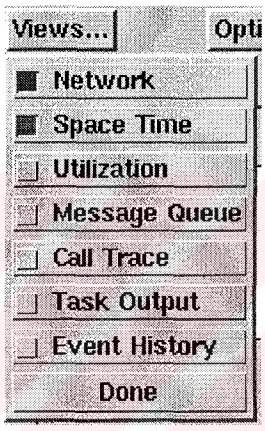
Şekil 3.12 Yardım menüsü



Şekil 3.13 XPVM 'in tanıtımı

3.2.2. XPVM monitör bakış açısı

XPVM, PVM görevlerinin gerçek zamanlı performanslarını ölçen bir servis olarak düşünülebilir. XPVM' den türetilen işler, istenilen herhangi bir PVM aktivitesi hakkında durum bilgilerini belirleme yeteneğine sahiptir. PVM 3.3 kütüphanesi izleme imkanına sahip olduğundan, kullanıcı programların tekrar derlenmesine gerek kalmamaktadır. Monitör bakışı ahenk içinde ve zamana bağlı olarak çalışmaktadır. Yani, faklı verilerin programın belirlenmiş bir noktasında icrası esnasında kullanıcıya farkı sunabilmektedir. "View" menüsü ile ağın durumu, uzay-zaman bakışı, kullanım oranı, mesajlaşma gibi bilgilere ulaşılabilir.

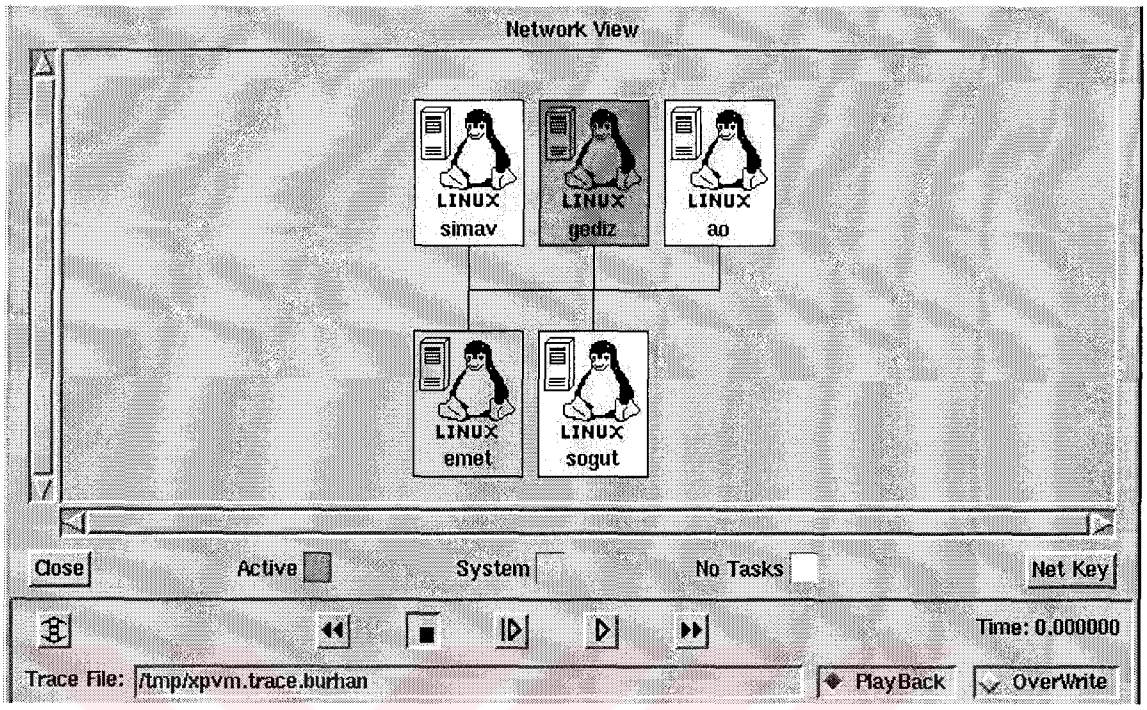


Şekil 3.14 View menüsü

3.2.2.1. Ağa bakış açısı

Sanal makinada terminal (host) üzerinde ağ bakış açısı yüksek seviye bilgi sağlar. Her bir makina, ismi ve mimarisi belirlenmiş olacak şekilde bir simge (icon) resmi ile temsil edilir. Bu simgeler her bir makinada çalışan işleri gösterecek şekilde farklı renkler ile gösterilir.

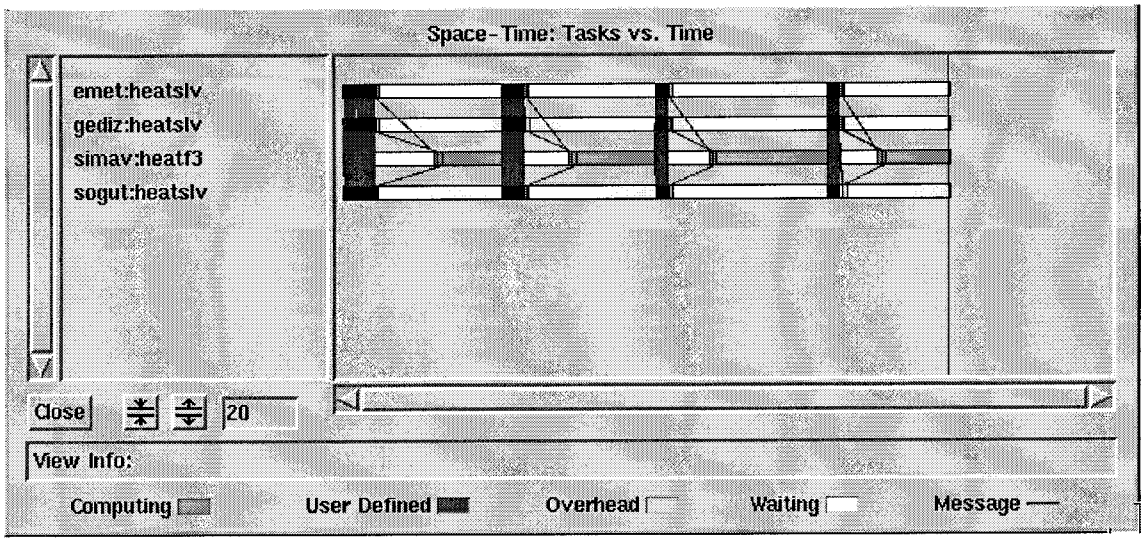
"Aktif" renk en azından bir tane işin ilgili hostta yararlı bir görev yaptığını gösterir. "Sistem" rengi kullanıcı ile ilgili bir işlemin yapılmadığını, öte yandan en az bir işin PVM sistem rutinleri ile ilgili icra görevi gördüğünü ifade eder. Terminal üzerinde hiç bir iş yoksa, simge renksiz yada beyaz ile ifade edilir. Terminallerin her bir durum karşısında alacağı renkler belirlidir.Örneğin yeşil aktif terminali, sarı ise sistem işlemlerinin yapıldığını temsil eder.Ancak renk ayarları kullanıcı tarafından yeniden düzenlenebilir.



Şekil 3.15 Ağın durumu

3.2.2.2. Uzay-zaman bakışı (Space-Time View)

Uzay-Zaman Bakışı, her bir görevin icrası sırasındaki bilgisayarların durumlarını gösterir. Her görev yatay bir çubukla temsil edilir ve çubuğun o anki rengi görevin durumunu belirtilir. Makine hesaplama yapıyorsa (computing) görev çubuğun rengi hesaplama rengini alacaktır. Sistem zamanı (overhead) rengi, işin haber amaçlı yada görev kontrol amaçlı PVM sistem komutları ile meşgul olduğunu temsil eder. Bekleme (waiting) rengi diğer görevlerden mesaj beklediğini gösterir. Mesajlaşma rengi ise mesaj iletimi ve alma işlemlerini temsil eder. İlgili durumlar için kullanılan renkler; hesaplama için yeşil, sistem zamanı için sarı, bekleme için beyaz ve mesajlaşma için kırmızıdır. Dikey mavi çizgi ise uygulamanın sona erdiğini gösterir. Şekil 3.14'de örnek bir uzay-zaman bakışı gösterilmiştir.

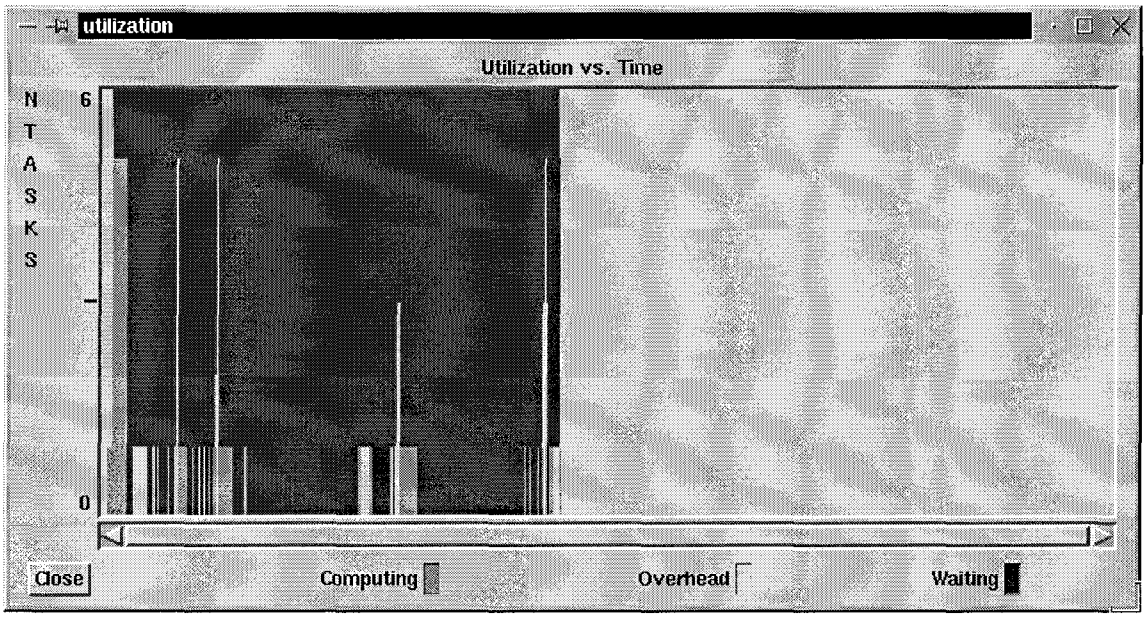


Şekil 3.16 Uzay- zaman bakışı

Görev durumları ve mesajlara ilişkin daha detaylı bilgi, uzay-zaman bakışı içinde farenin sol butonuna basılarak elde edilebilir. Görev çubuklarına basıldığında ekranın alt kısmındaki bilgi şeridi görevin ne zaman başladığını, ne zaman bittiğini gösterir. Buna ilaveten durum başlangıcındaki yada bitişindeki çağrılan PVM sistem komutlarını belirtir.

3.2.2.3. Kullanım-oran bakışı (Utilization View)

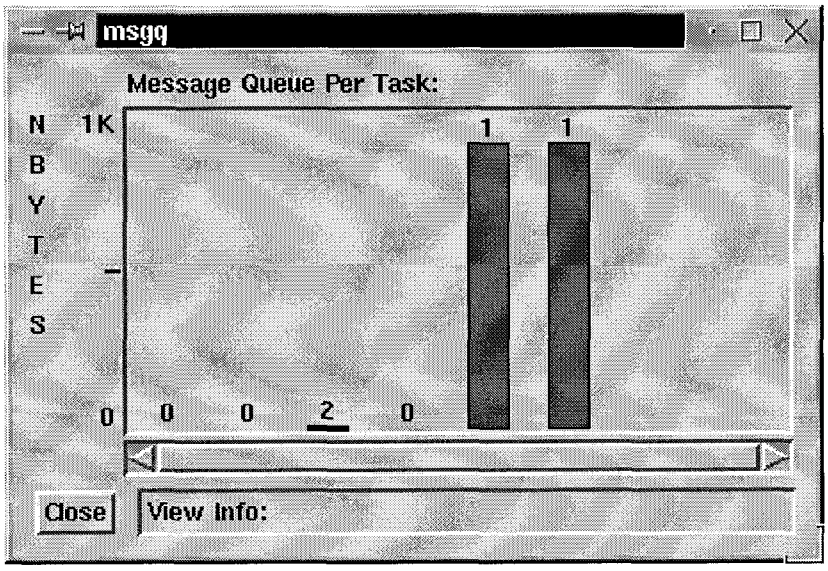
Verilen bir zaman diliminde arta kalan işlemler veya mesaj bekleme konumu işlevleri ile ilgili özet bilgiyi verir. Bu bilgiler üç renkli dikdörtgenlerin yığına dikey olarak koyulması şeklinde temsil edilir. En alttaki hesaplama, ortadaki arta kalan görev süresini ve en üsteki beklemeyi gösterir. Belirlenen dikdörtgen renkleri: hesaplama için yeşil, arta kalan için sarı ve bekleme için kırmızıdır.



Şekil 3.17 Kullanım oranı grafiği

3.2.2.4. Mesaj yığınları

Bu özellik ile uygulamanın icrası esnasında makinaların yığınlarına (buffer) gelen ve bekleyen mesajların durumu izlenebilir. Mesajlar, boyutlarına göre grafiksel olarak temsil edilmektedir.



Şekil 3.18 Mesaj yığınları

Uygulamanın icra esnasında mesaj yığınlarının durumunu izleyen bu gözlenebilirlik ile makinalara dağıtılan işin eşit paylaşımı gözlenebilir ve paylaşımın optimalliği sağlanabilir.

Sonuç olarak, XPVM'in sunduğu gözlenebilirlik imkanları paralel programın icrası esnasındaki olayların takibi, hesaplama ortamının kontrolü ve performans analizini daha kolay kılmaktadır. Performansın belirlenmesi veya programın performans hatalarından arındırılması için bu görsel araçlar vazgeçilmezdir.

4. PVM UYGULAMALARI

Bu bölümde Heat adlı ısı yayılım programının farklı işlemci sayısı ve problem boyutları için icra süreleri ölçülerek performans analizi yapılmıştır. Program konsoldan icra edilmiş ve cevap süreleri ölçülmüştür. Cevap süresi XPVM ile de ölçülebilirdi, ancak XPVM'in monitör özelliği icra süresini olması gerekenden fazla arttırdığından ölçümler sırasında kullanılmamıştır. Programlar İnternet'ten hazır alınmıştır. İşlemci sayısı ve problem boyutları için koda müdahale edilmiştir.

Deney ortamı, Linux işletim sistemi yüklü Pentium III- 800 Mhz işlemcili, 128 Mbyte bellekli beş bilgisayar kullanılmıştır. Bilgisayarlar 100/10 Mbit/sn hızlı Ethernet kartı ve 16 portlu blokajsız anahtar elemanı ile bağlanmıştır.

4.1. Isı Yayılımı Uygulaması (Heat-Flow)

Heat-Flow adlı program, ısı yayılımı diferansiyel denklemini ana-uydu modeliyle PVM ortamında çözen bir programdır. Program bir tel kablo üzerinde yayılan (difüzyon) sıcaklığı hesaplamaktadır. İnce bir tel üzerine yayılan sıcaklığı düşünürsek:

$$\frac{\partial A}{\partial t} = \frac{\partial^2 A}{\partial x^2} \tag{4.1.1}$$

Denklem farklı bir şekilde yazılırsa:

$$\frac{A_{i+1,j} - A_{i,j}}{\Delta t} = \frac{A_{i,j+1} - 2A_{i,j} + A_{i,j-1}}{\Delta x^2} \tag{4.1.2}$$

açık formülü verilirse:

$$A_{i+1,j} = A_{i,j} + \frac{\Delta t}{\Delta x^2}(A_{i,j+1} - 2A_{i,j} + A_{i,j-1}) \tag{4.1.3}$$

başlangıç ve sınır şartları:

$$A(t,0) = 0,\ A(t,1) = 0\ \textit{for all } t \tag{4.1.4}$$

$$A(0,x) = Sin(\pi\ x) \quad \textit{for}\ \ 0 \le x \le 1 \tag{4.1.5}$$

Bu programda ana makina (master) ilk veriyi hazırladıktan sonra uydularla haberleşir ve sonuçları uydulardan bekler. Uydular (slaves) ise ana makinadan ilk verileri aldıktan sonra sınır koşullarını kendi komşularıyla değişir ve bir teldeki ısı değişimini hesaplar. Bu işlem ana makina tarafından gönderilen birkaç iterasyonla devam ettirilir [2].

Heat-Flow'da ısı dağılımı ile ilgili hesaplamalar, uydu makinalarda çalışan programlar tarafından yapılmaktadır. Uydu programda var olan bir sonsuz döngü ile ilk veriler ana makinadan alınır, sınır koşulları ise komşu makinalarla haberleşilerek belirlenir ve iteratif olarak sonuçlar hesaplanır. Daha sonra bu kısmi sonuçlar ana makinaya geri gönderilir. Sonuçlar ana makinaya ulaştığı zaman, ana makinada çalışan program sonuçları çözüm matrisine yerleştirir, geçen süreyi hesaplar ve sonucu xgraph (koordinatları belli veriler için kullanılan bir program) dosyası şeklinde yazdırır.

Uydu makinalarda çalışan Heatslv programı telin üzerindeki ısı yayılımının hesaplamasını yapar. Uydu programı; başlangıç verilerini alan, bu verilerin üzerinde iteratif yöntemle hesaplama yapan (her bir iterasyonda komşularıyla sınır bilgilerini değiştiren) ve çözümün sonuç kısmını ana makinaya geri gönderen sonsuz sayıda döngüden oluşmaktadır. Uydu, görevlerini bir sonsuz döngüde kullanmak yerine, ana makinadan uyduya programdan çıkmasını emreden özel bir mesaj gönderilebilirdi. Ancak programda mesaj iletimini karmaşık hale getirmek yerine uydu görevlerinde basitçe sonsuz döngü kullanılarak hepsi ana programdan kapatılmaktadır.

Her iterasyonda hesaplama fazından önce, ısı sınır değerleri matrisi karşılıklı değiştirilir. Sol taraftaki sınır elemanları önce sol taraftaki komşuya gönderilir ve sağ sınır elemanları alınır simetrik olarak sağ taraftaki komşuya gönderilir ve oradan sol sınır değerleri alınır. Her alma ve gönderme işlemlerinde olmayan bir komşuyla iletişim kurulmaya çalışılmaması için görev numaraları (TID) kontrol edilir.

Yukarıda anlatılanlardan da anlaşılacağı üzere Heat-Flow oldukça mesajlaşma yoğunluklu bir paralel programdır. Bu nedenle yapılan deneylerde işlemci sayısı arttırıldığında icra süresinin çok fazla düşmesi beklenmemektedir.

4.2. Deneyler ve Ölçüm Sonuçları

Deneyler sırasında program içindeki problem boyutu ve işlemci sayısı değişkenleri uygun şekilde değiştirilerek her konfigürasyon için derlenmiş, ana ve uydu bilgisayarlara icra edilebilir dosyalar kopyalanmıştır. Daha sonra konsoldan PVM metin tabanlı olarak çalıştırılmış ve her konfigürasyon için yeterli sayıda işlemci paralel ortama dahil edilmiştir.

Heat-Flow programı her bir konfigürasyon için en az 5 defa çalıştırılmış ve icra süreleri ölçülmüştür. Ölçülen değerlerin ortalaması alınarak ilgili konfigürasyon için icra süresi belirlenmiştir. İcra süreleri LINUX komutu 'time' ile ölçülmüştür. Bu komut, programın icra

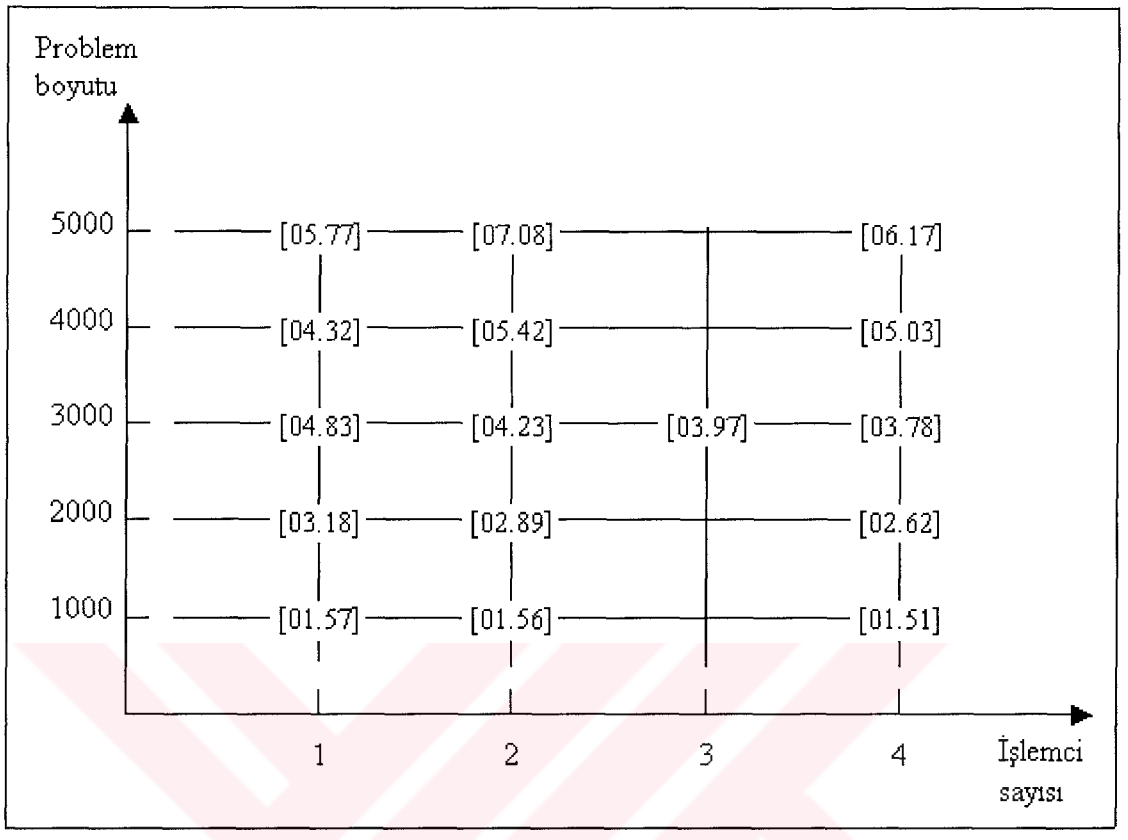
süresini, işlemci kullanım süresini ve işlemci kullanım oranını verilerini göstermektedir. Time komutuna ait örnek çıktı verisi Şekil 4.1'de gösterilmiştir.

```
0.360u 0.010s 0:01.11 %33.3
```

Şekil 4.1 Time komutu veri çıktısı

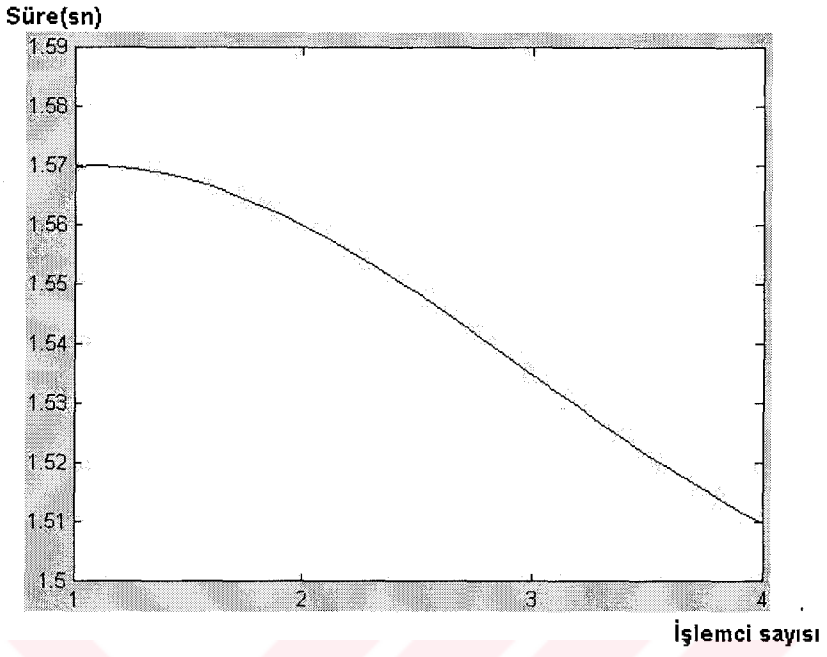
Komutun çıktı verisinde ilk kısım işlemci kullanım süresidir. Daha sonraki veri, uygulamanın icra sırasında işletim sisteminde harcanan süreyi gösterir. 3. veri uygulamanın toplam icra süresini, en son veri ise hesaplama süresi oranın yüzdelik ifadesini gösterir. Hesaplama süresi oranı, işlemci ve sistemdeki harcanan sürelerin toplamının icra süresine oranın yüzdelik ifadesidir.

Konfigürasyon denemelerinde işlemci boyutu-problem boyutu-icra süresi tablosu Şekil 4.2'de verilmiştir. Tabloda bazı problem boyutları için icra süreleri verilmemiştir. Bunun nedeni; program içinde işlemci sayısı 'NPROC' ve problem boyutu 'SIZE' tanımları yapılmış ve işin paylaşımında SIZE/NPROC tanımında boyutun işlemci sayısına bölümünün tamsayı olması gerektiğinden bu tür konfigürasyonlar için uygulama çalıştırılmamıştır. Konfigürasyon denemelerinde elde edilen veriler Ek 1'de verilmiştir.

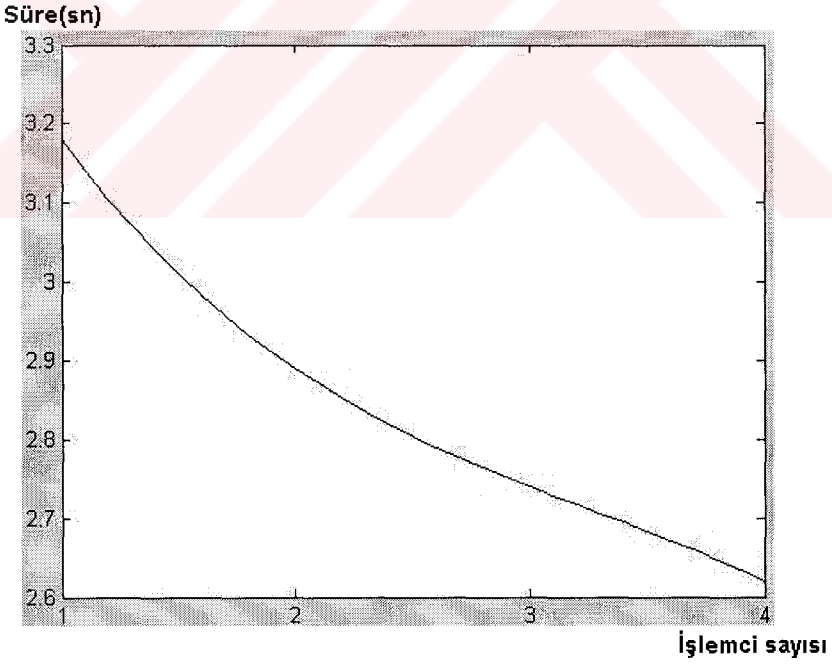


Şekil 4.2 Heat-Flow'a ait işlemci sayısı-problem boyutu-icra süreleri

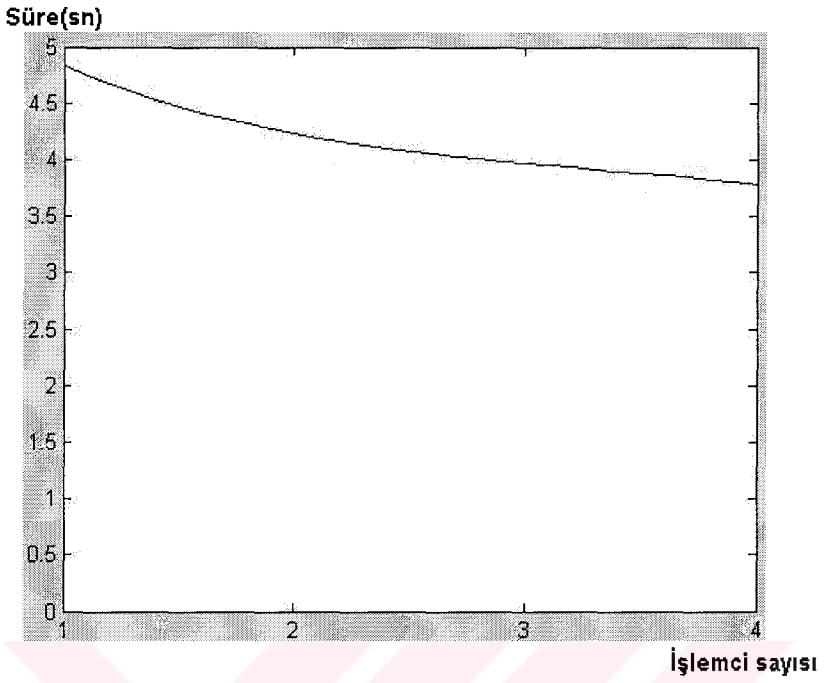
Elde edilen bu tablodaki verilere göre Matlab 6.0 kullanılarak icra süresi grafikleri çizilmiştir. Simülasyonda ilk olarak farklı problem boyutları için işlemci sayısı-icra süresi grafikleri çizilmiştir. Bu grafikler Şekil 4.3, 4.4, 4.5, 4.6 ve 4.7'de gösterilmiştir. Grafiklerde ara değerler için küçük kareler yöntemi kullanılmıştır.



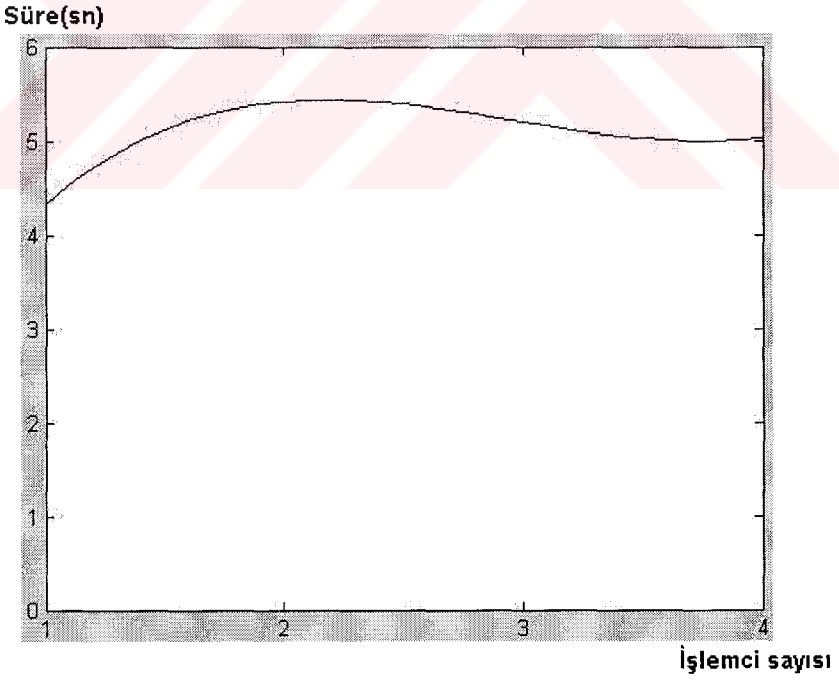
Şekil 4.3 Problem boyutu 1000 için işlemci sayısı-icra süresi grafiği



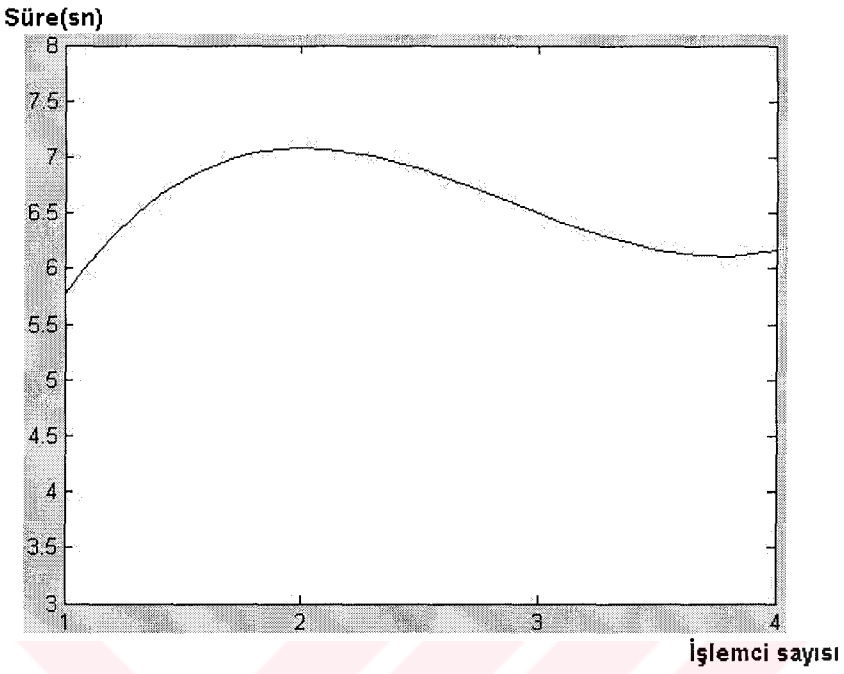
Şekil 4.4 Problem boyutu 2000 için işlemci sayısı-icra süresi grafiği



Şekil 4.5 Problem boyutu 3000 için işlemci sayısı-icra süresi grafiği

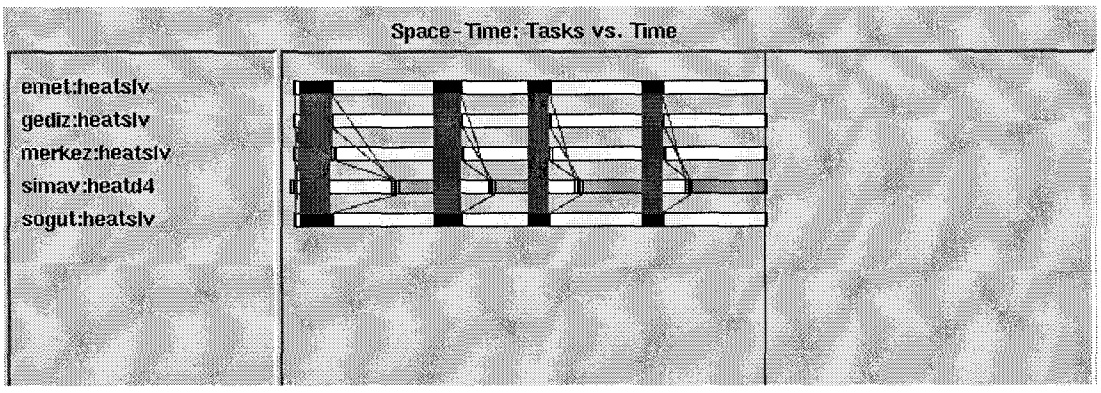


Şekil 4.6 Problem boyutu 4000 için işlemci sayısı-icra süresi grafiği

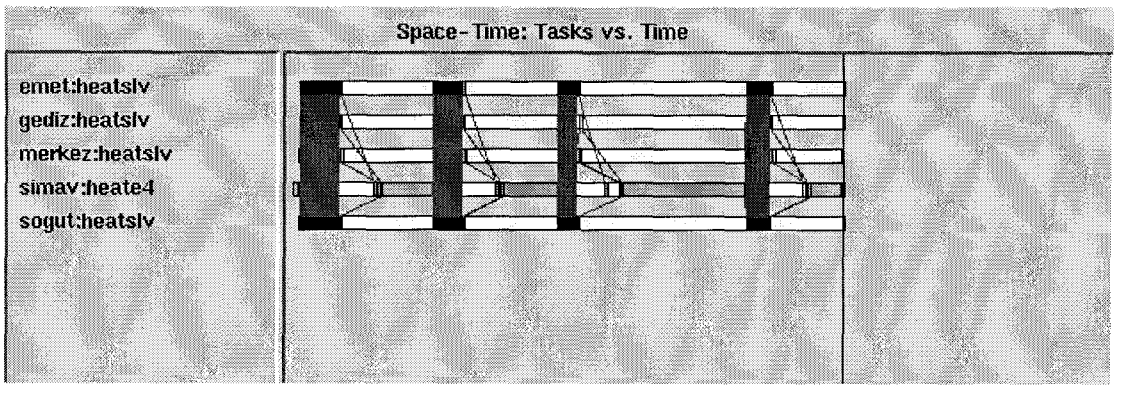


Şekil 4.7 Problem boyutu 5000 için işlemci sayısı-icra süresi grafiği

Grafiklerde görüldüğü gibi problem boyutları 1000, 2000 ve 3000 için icra süresi artan işlemci sayısına göre azalmaktadır. Buna göre performans için başvurulan paralel hesaplama hedefine ulaşmıştır. Ancak, problem boyutunun 4000 ve 5000 olduğu konfigürasyonlarda artan işlemci sayısına karşın icra süresinin azalması beklenirken sürenin arttığı gözlenmiştir. Bunun nedeni ise programa ait PAG'in (Program Aktivite Grafiği) XPVM yardımıyla elde edilmesi ve incelenmesi sonucu seçilen uygulamanın haberleşme ağırlıklı olduğu anlaşılmıştır. Bu nedenle problem boyutundaki büyüme beraberinde haberleşme yükünü de arttırdığından icra süresi uzamıştır. Uygulamanın problem boyutları 4000 ve 5000 için program aktivite grafikleri Şekil 4.8 ve Şekil 4.9'da gösterilmiştir.

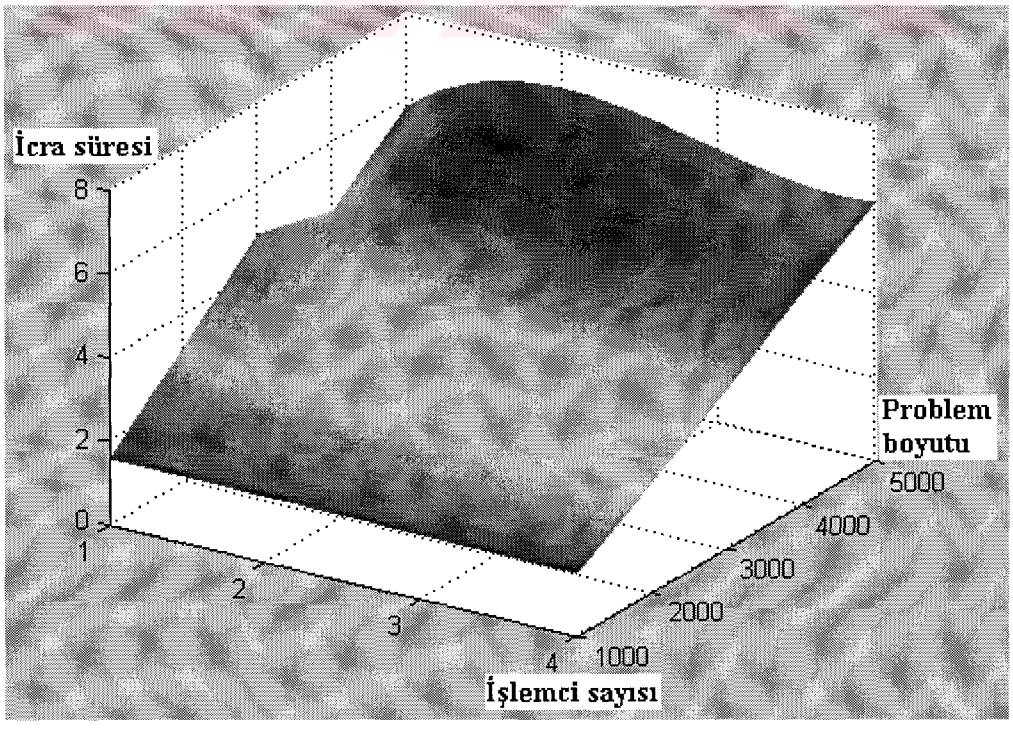


Şekil 4.8 Problem boyutu 4000 için program aktivite grafiği

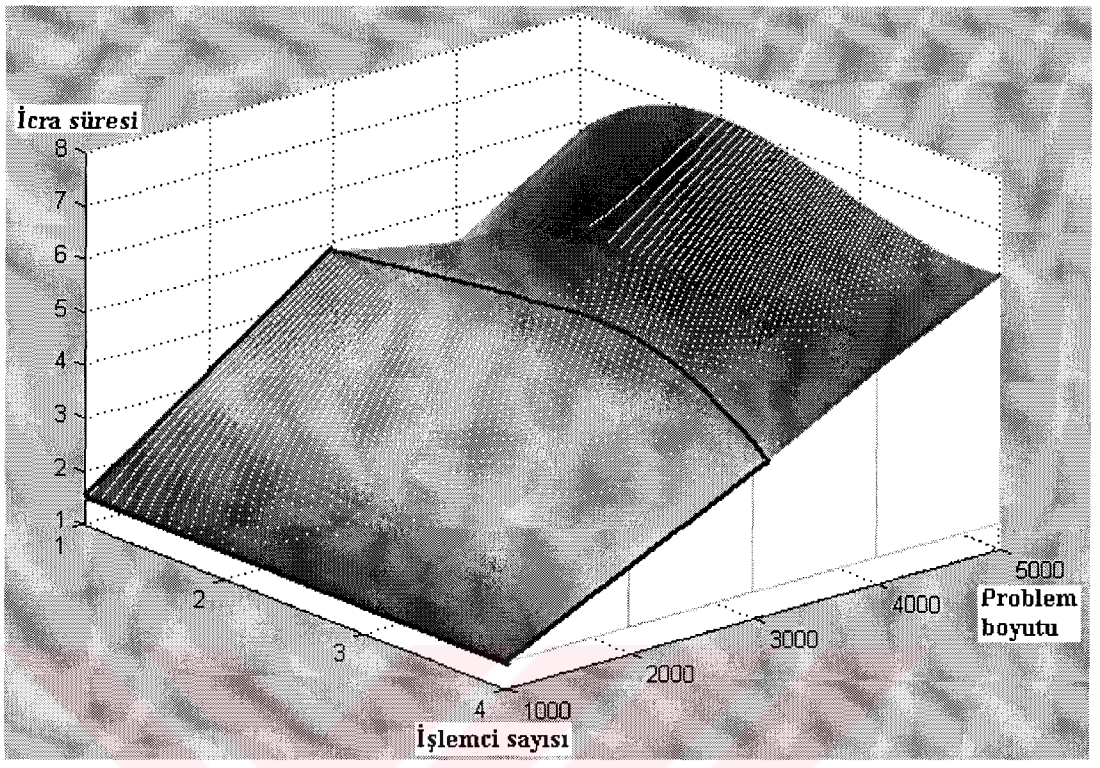


Şekil 4.9 Problem boyutu 5000 için program aktivite grafiği

Program aktivite grafiklerinde görüldüğü gibi işlemciler arasındaki haberleşmede çok fazla zaman harcanmaktadır. Bu nedenle bazı problem boyutları için işlemci sayısını arttırmak icra süresini kısaltmamış aksine uzatmıştır. Özellikle problem boyutu 3000' den büyük değerler için işlemci sayısı arttırmak performansı düşürmektedir. Bu doğrultuda performansın optimum noktalarının tespiti için işlemci sayısı-problem boyutu-icra süresi değerlerinin üç boyutlu grafiği çizilmiştir. Grafik hem Matlab ile hem de Excel ile çizilmiş olup, performansın optimum noktaları grafikler üzerinde işaretlenmiştir. Matlab grafikleri Şekil 4.10 ve Şekil 4.11'de gösterilmiştir.

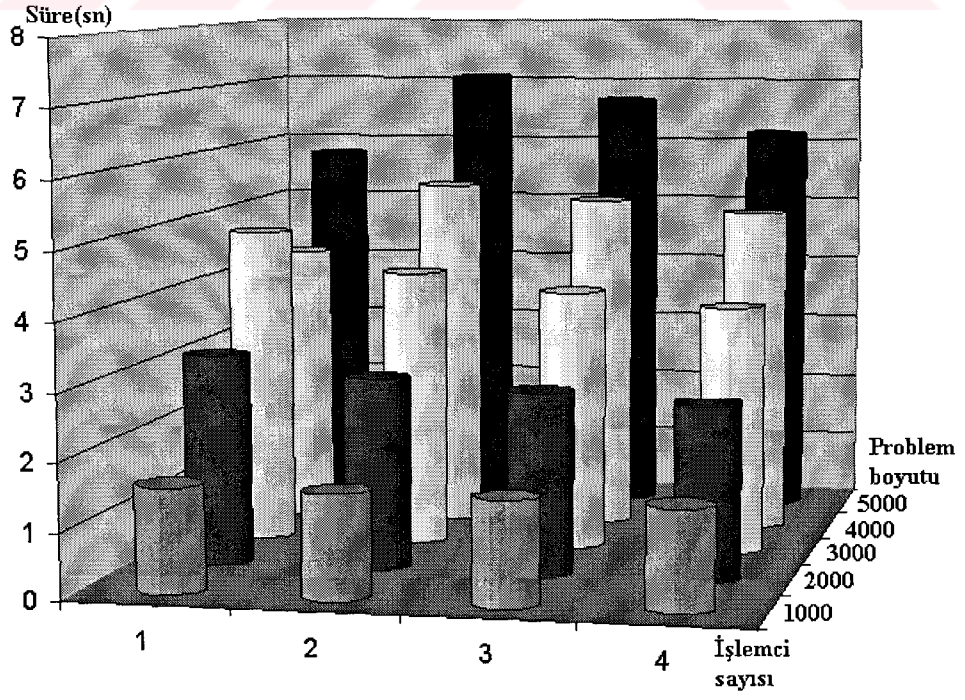


Şekil 4.10 Problem boyutu-işlemci sayısı-icra süresi grafiği

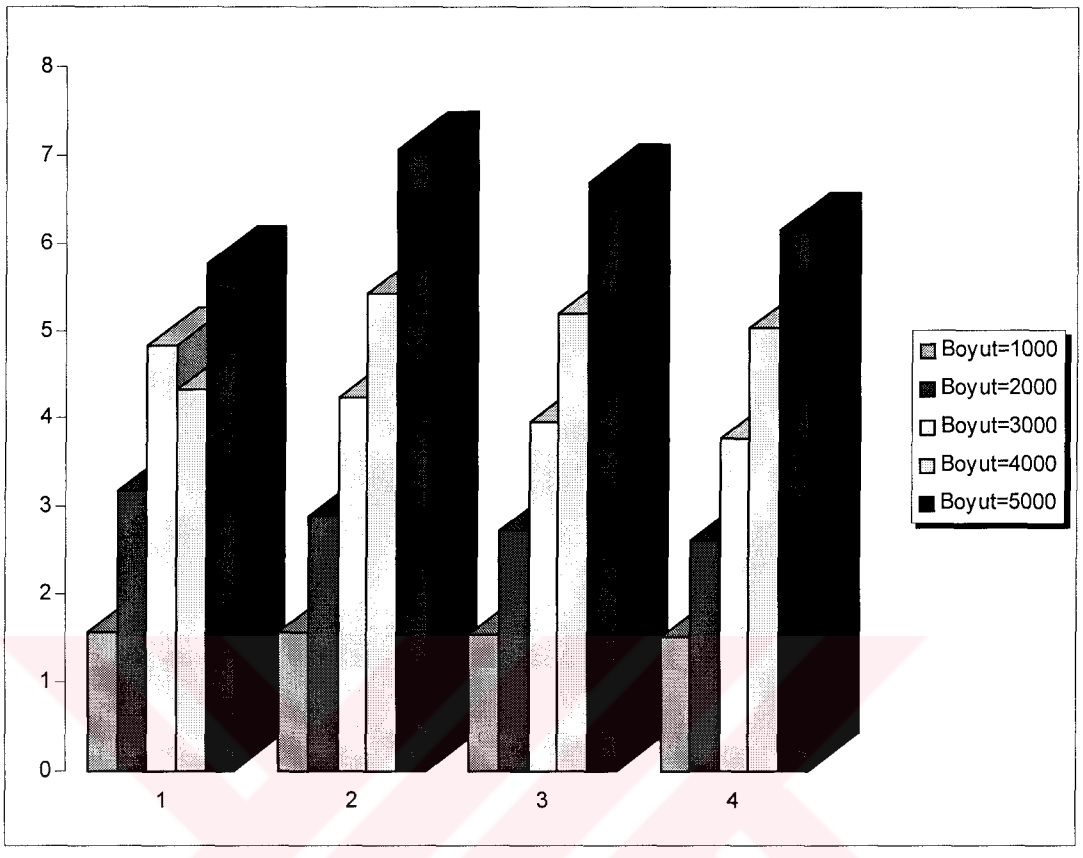


Şekil 4.11 Problem boyutu-işlemci sayısı-icra süresi grafiği

Grafiklerin Excel ile çizimi Şekil 4.12 ve Şekil 4.13'de gösterilmiştir.



Şekil 4.12 Problem boyutu-işlemci sayısı-icra süresi Excel silindir grafiği



Şekil 4.13 Problem boyutu-işlemci sayısı-icra süresi Excel sütun grafiği

Grafiklerde görüldüğü gibi problem boyutu 1000, 2000 ve 3000 için artan işlemci sayısı ile icra süresinin kısaldığı problem boyutu 4000 ve 5000 için sürenin uzadığı açıktır. Seçilen bu uygulama için Matlab grafiklerinde işaretlenen bölgede(problem boyutu 1000, 2000 ve 3000 için) paralel hesaplamanın performansı arttırdığı gözlemlenmiştir. Diğer bölgelerde ise paralel hesaplama performansı düşürmüştür.

Sonuç olarak, paralel hesaplamada problem boyutu ile kullanılacak işlemci sayısı uygun seçilmesi gerekir. Haberleşme ağırlıklı uygulamalarda işlemci sayısını arttırmak, uygulamamızın problem boyutu 4000 ve 5000 konfigürasyonlarında olduğu gibi haberleşmede harcanan süreyi arttıracağından icra süresini uzatabilir ve performansın düşmesine neden olabilir. Bu nedenle uygun işlemci sayısı seçilmeli ve uygulama performansın optimum olduğu bölgelerde çalıştırılmalıdır. Aksi halde performansın arttırılması için başvurulan paralellik hedefinden uzaklaşacaktır.

5. DİĞER PARALEL ORTAMLAR

Bu bölümde PVM' e benzer yazılım paketleri tanıtılmıştır. Çeşitli araştırma grupları dağıtık hesaplama uygulamalarında bilgisayar programcılarına yardım edebilmek için PVM' e benzer yazılım paketleri geliştirdiler. Bunların arasında en iyi bilinenler; P4, Express, MPI, Linda, Unify' dır [2].

5.1. P4 Sistemi

Çeşitli paralel makinaların programlanabilmesi amacıyla Argonne Ulusal Laboratuarı'nda geliştirilen altprogramların bir kütüphanesidir. P4 sistemi hem monitör tabanlı paylaşılan hafıza modelini hem de mesaj geçişini kullanarak dağıtık hafıza modelini destekler. P4 paralel hesaplamaların paylaşılan hafıza modelinde, en az kurulabilen ilkel (primitif) monitörlerin seti kadar iyi kullanışlı bir monitör seti sağlar. Dağıtık hafıza modeli için ise grup ve işlem yapılarını tanımlayan bir text dosyasına göre işlemlerin gönderme ve alma operasyonlarını sağlar.

P4 sistemindeki işlem yönetimi makina (host) havuzunu, her bir makinada hesaplanan nesne dosyasını, ilk önce çok işlemcili sistemler için tasarlanan her bir makinada başlayan işlem sayısını ve diğer yardımcı bilgileri açıkça belirten bir konfigürasyon dosyası tabanlıdır. Şekil 5.1, bu konfigürasyon dosyasına bir örnektir.

```
# start one slave on each of sun2 and sun3
    local 0
    sun2   1   /home/mylogin/p4pgms/sr_test
  sun3 1 /home/mylogin/p4pgms/sr_test
```

Şekil 5.1 P4 konfigürasyon dosyası [2]

P4 içindeki işlem yönetim mekanizmasını gerçeklemenin dikkate değer iki çeşidi vardır. Birincisi, bir ana işlem ve uydu işlemler içeren çok dereceli ve küme (cluster) modelinde hesaplama yapan türüdür. İkincisi ise işlem oluşumunun konfigürasyon dosyası sayesinde statik olduğu (değişmediği) ilk modudur. Değişken (dinamik) işlem oluşumu ise, sadece lokal makina üzerinde yeni bir işlem yapmak için mutlaka özel 'o4' fonksiyonu isteyen statik olarak üretilmiş bir işlem sayesinde mümkündür. Bu sınırlamalara rağmen çeşitli uygulama paradigmaları doğru bir şekilde P4 sistemi içinde gerçekleştirilebilir.

P4 sistemi içindeki mesaj geçişi doğrudan geleneksel gönderme ve alım ilkeleri kullanılarak başarılabilir ve hemen hemen diğer mesaj geçiş sistemlerinin aynısı gibi parametrize edilebilir [2].

5.2. Express

Express programı diğer paralel işletim sistemlerinin tersine, o andaki hesaplamanın çeşitli yönlerini tek tek adresleyen program-komut topluluğundan oluşur. Bu program, Caltech eşzamanlı hesaplama projesinin üyeleri tarafından kurulmuş olan ParaSoft Corporation şirketi tarafından geliştirilmiş ve ticari olarak piyasaya sürülmüştür.

Express sisteminin özü haberleşme, giriş/çıkış ve paralel grafiklerin bulunduğu kütüphane setidir. Haberleşme ilkeleri diğer mesaj geçiş sistemlerine benzer. Ayrıca global operasyonların çeşitliliğini ve veri dağıtım ilkelerini içerir. Genişletilmiş giriş/çıkış komutları paralel giriş ve çıkışları kullanıma açar ve çoklu eşzamanlı işlemlerden grafiksel gösterim için komutların benzer bir seti sağlanır. Express ayrıca popüler "dbx" arayüzü üstüne kurulan komutları kullanan paralel bir hata arındırıcı (debugger) olan NDB programını da içerir [2].

5.3. Mesaj Geçiş Arayüzü (MPI)

MPI, diğer mesaj-iletim sistemlerinin evrimsel bir toplamıdır. MPI standartı açık bir standarttır ve herkes kendi MPI işletimini yapabilir. MPI bir paralel programlama birimi değildir ve paralel programlamanın bazı özellikleri standarda eklenmemiştir. Bundan dolayı MPI, dinamik işlem kontrolü, paralel giriş-çıkış ve paralel görevlerin başlatılması gibi özellikleri içermez. Her MPI işletimi kendi paralel programlama birimlerini sunar. MPI işletiminin içindekileri programlayıcının düzenlemesine bağlıdır. Öncelikle MPI mesaj iletiminin güvenliğini garanti altına alır. Özel bir görevden başka bir göreve gönderilen mesajlar gönderiliş sırasına göre ulaşır. Böylece programcı kontrol işlemlerini yapmak zorunda kalmaz [13].

MPI, dağıtık hesaplama için kullanılan bütün ve kendiliğinden içerikli yazılım altyapı sistemi olması amacıyla yapılamamıştır. MPI işlem yönetimi, sanal makina konfigürasyonu, giriş ve çıkış için destek gibi gereksinimleri içermez. Sonuç olarak MPI, donanıma yakın bir seviyede kullanıcının istekleri üzerine kurulacak olan haberleşme arayüzü olanağı sağlar [2].

5.4. Linda Sistemi

Linda, 1980 lerin ortasında tanınan süper bilgisayar ve geniş iş istasyonu grupları için sanal paylaşımlı bellek (VSM) işletimini sağlayan ilk ticari üründür. Linda, içerik-

adreslenebilir, adres-tabanlı değildir, uygulamaları kolayca yapılandırır ve donanım kapasitesinden maksimum yaralanır. Lindanın temel özellikleri şunlardır:

- Taşınabilirlik: Linda, paylaşımlı bellek bilgisayarları, dağıtık-bellek bilgisayarları ve ağa bağlı çok sayıdaki bilgisayar sistemlerinde kullanılabilir.
- Kullanım kolaylığı: Linda paralelliği, mantıksal-dağıtık bellek yoluyla az sayıdaki basit fakat güçlü operasyonları kullanır.
- Yatırım korunumu: Linda, C, C++, Fortran gibi dilleri paralel koda kolay ve hızlı bir şekilde dönüştürmeye imkan verir.
- Özel paralel mimarilerde yüksek performans: IBM, Cray, SGI, HP ve Hitachi gibi tüm paylaşımlı-bellek makinaları ve paralel-bellek makinalarının temel mimarilerinden faydalanır.
- Dinamik yük dengelemesi: Sanal paylaşımlı bellek veri modeli paralel heterojen işlemciler arasında otomatik yük dengelemeye rehberlik eden teknikleri kullanarak yazılım yapmayı kolaylaştırır.

Linda, paralel hesaplamayı zorlaştıran geleneksel engellerin (pahalı donanım, programlama zorluğu, kod uygunluğunun zayıflığı) üstesinden gelerek birçok alanda (hava boşluğu, otomotiv, petrol, yarıiletken, ecza, finans) büyük kabul görmüştür [12].

5.5. Unify

Unify projesi, Kentucky Üniversitesi Bilgisayar Bilimi Bölümünde dağıtık hesaplama sistemleri (DSM) üzerine yürütülen bir araştırma projesidir. Unify projesinin konusu, coğrafi olarak birbirine uzak olan çok sayıdaki yüksek performanslı makinaların çoklu bilgisayar bağlantılarını geliştirmektir. Projenin amacı, yüksek ölçeklenebilir paylaşımlı hafıza programlama paradigmasını (uygun programlama modelleri sağlayan) desteklemektir. Geleneksel DSM yaklaşımları, böyle geniş coğrafi alana yayılmış düzenekler için uygun değildir. Unify projesi, böyle bir çevreye yayılabilirliği başarmak için kaynak paylaşımını gizleyen, transfer edilen verileri ve haberleşme frekansını düşüren ve sınırlayan, senkronizasyonu serbest uygun eski metotlar yolu ile geniş ölçekli uygunluğu destekleyen yeni paylaşımlı hafıza düşünceleri ve mekanizmaları destekler. Sistemin amacı, performans gözlenebilirliği ve ölçeklenebilirliği PVM ve MPI gibi geniş amaçlı mesaj ileten çoklu bilgisayarlardaki ile aynı olan uygun veri paylaşımını sağlamaktır. Unify projesinin göze çarpan özellikleri şunlardır:

- Tek Adres Alanı: Tek adres alanı, tüm uygulamalar tarafından paylaşılan yapısal adres bağımlı verilerin uygun ve verimli paylaşımına izin verir.

- Çoklu Hafıza Tipleri: Sistem paylaşılmış veriler için üç temel hafıza düşüncesini destekler.Sırasıyla Erişimli Hafıza (SAM) oku/ön, yaz/ek biçiminde erişilir. Sıralı erişim ve birleşebilen hafıza RAM parçalarından daha verimli işlenebilen daha zayıf alana ilişkin garantiler tarafından genellikle desteklenebilir.
- Çoklu Tutarlılık Dereceleri: Unify tasarımı, işletim sisteminin tutarlılığı sağlamlaştırdığı otomatik metotları ve kullanıcının tutarlılık kontrol noktalarını tanımladığı uygulama destekli metotları içeren tutarlılık protokollerinin spekturumundan uygun tutarlılık anlamları seçmek için bir uygulamaya izin veren bir dizi tutarlılık yönetim metodunu sağlar. Bazı DSM sistemleri zayıf uygulama destekli geçici tutarlılık modellerinin faydalarını göstermiştir. Bunlara ek olarak, Unify tasarımı, hafızanın bazı T zaman ayrımlarından sonra tutarlı olduğu yerlerdeki zayıf otomatik metotları destekler.
- Ölçeklenebilir Senkronizasyon Metotları: Birçok DSM tasarımı basit senkronizasyon metotları olarak kilitleri (lock), ve/veya (and/or) bariyerlerini destekler. Geleneksel senkronizasyonlama metotları senkronizasyonlama olayını eş zamanlı olarak gözlemleyen tüm katılımcılara gereksinim duyar. Olay sayıları katılımcılara olayı farklı zamanlarda gözlemlemeleri için, haberleşme sıkıntılarını rahatlatmak için ve daha iyi uygunluk için izin verir. Bundan başka, geleneksel senkronizasyonlama metotları olay sayıları yolu ile işlenebilir.
- Domain Paylaşımının Hiyerarjisi: Geniş ölçekli paylaşımlı çoklu bilgisayarlarda paylaşım, yerleşim ilkesini takip etmektedir. Yerleşmiş paylaşım ve haberleşmeyi değerlendirebilmek için hostları 'domain paylaşım' larına bölümlenmektedir. Her bir domain paylaşımı, ayrı multicast grupları, intra-domain bilgi paylaşımının maliyetini düşürmek için kullanılır. Domain paylaşımı, bilgiyi tekrar elde edebilecek durumda dağıtır ve inter-domain gereksinimlerini karşılamak için domain'in herhangi bir parçasına izin verir. Her host bilgi için domainin dışına gitmeden önce yerel paylaşımlardaki hostlara danışır. Bundan dolayı bir host uzak domaindeki bir siteden cross_domain bilgi alır almaz bilgi domain paylaşımında verimli olarak ulaşabilir hale gelir.
- Güvenilir Multicast Desteği: Dağıtık hesaplama uygulamaları, sadece veriyi güvenli iletmez aynı zamanda veriyi alacaklara senkronize olarak dağıtacak güvenli bir multicast mekanizmasına ihtiyaç duyar. Paylaşımlı verinin veya senkronize bilginin güvenilir, ölçeklenebilir ve etkili dağıtımını sağlamak için Unify, ağaç tabanlı multicast taşıyıcı protokol (TMTP) yolu ile güvenli multicast i sağlar. TMTP geniş alanda

ulaşılabilir olan IP multicast 'in etkili dağıtımı üzerine kuruludur. TMTP güvenirliliği sağlamak için, hata ve akış kontrolünü yönetmek için ayrık kontrol ağaçları yapılandıran gönderici ve alıcı yaklaşımları kombinasyonunu kullanır. Tekrar iletimler zaman biçiminde yerel olarak yönlendirilir. Sonunda, yığılmış pozitif sinyaller İnternet trafiğini azaltır ve paket problemlerini yok eder [10].

Geniş çaplı çoklu bilgisayarların yararlılığı ve ölçeklenebilirliği sadece gerçek-hayat, geniş ölçekli paralel ve dağıtık uygulamalar aracılığı ile gösterilebilir. DSM uygulamaları kütüphanelere uygun hafıza çeşitleri ve tutarlılık anlamlarıyla paylaşımlı parçalar oluşturmak için bağlanır. Unify sistemi Unix iş istasyonlarında gerçek zaman kütüphanesi olarak çok sayıda host içeren testlerde çalıştırılmıştır.

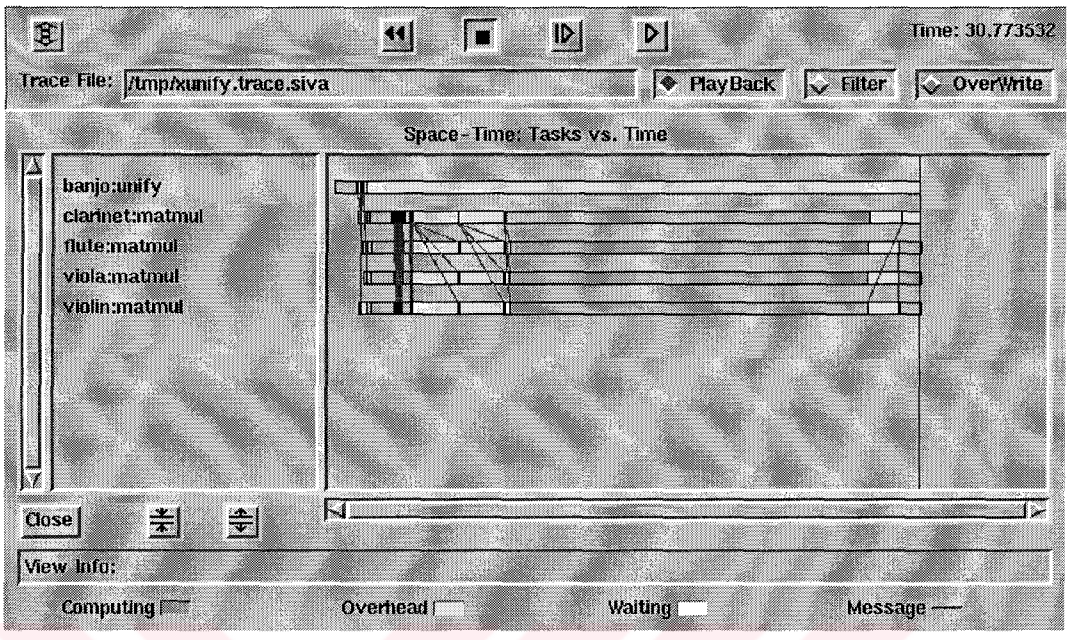
5.5.1. Xunify: Unify için performans gözlenebilirliği

Xunify, DSM sistemi için bir performans gözlenebilirliği olarak geliştirilmiştir. Xunify, programlamacıya yardım etmek ve onun uygulamalarındaki performans problemlerini anlaması için grafiksel görümünde bir çok program uygulama imkanı sağlar. Bu uygulamalar uzay-zaman bakışı, ağ bakışı, olay filtreleme bakışı, kullanım bakışı, yüzde kullanım bakışı, çıktı izleme bakışı ve belirli çıktısı bakışı olarak sıralanabilir. Bu araçlar, uygulamacıya programdaki performans problemlerini gidermeye yardımcı olur. Uygulama çalışırken bu araç, izleme verisini işletir ve ekranda görüntüler. Hatta toplanan izleme verisi birçok kere tekrar oynatılabilen izleme dosyası içine yazılabilir.

Xunify, PVM sistemine geçen mesaj için kullanılan XPVM'in değişimi olarak geliştirilmiştir. Hatta bu araç izleme mesajlarını üretmek için PVM ile sağlanan izleme kolaylıklarına adapte olur. C ve TCL/TK kullanılarak geliştirilmesine devam edilmektedir [11].

5.5.1.1. Uzay-zaman bakışı

Uzay zaman bakışı yatay eksendeki zaman ve dikey eksendeki host lar için uygulama programının yürütümünü gösterir. Bu durum Şekil 5.2'de gösterilmiştir. Ayrıca bu araç Unify sistemindeki ortak işlemlerin nasıl çalıştığının anlaşılmasına yardımcı olmaktadır. Bu görüşle çalışmakta olan tüm Unify görevleri görülebilir. Her Unify çalışanı yatay bir çubuk ile temsil edilir. Çubuk yeşil, sarı ve beyaz renklere sahip olabilir. Yeşil renk çalışanın bir hesap yaptığını, sarı renk kişinin bağlantıda olduğunu veya bazı sistem servislerine giriyor olduğunu ve beyaz renk ise çalışanın senkronizasyonda kilitleniyor olduğunu gösterir. Çalışanlar arasındaki gelen giden mesajlar gönderenden alana doğru olarak kırmızı oklarla ifade edilir.

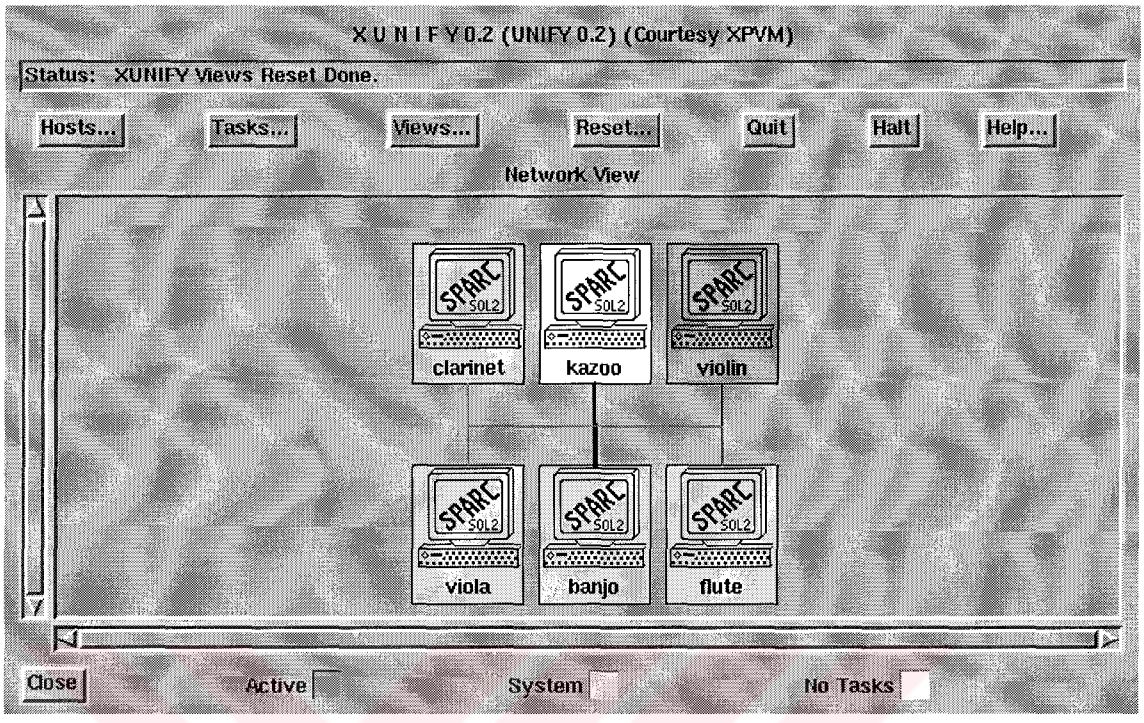


Şekil 5.2 Xunify uzay-zaman bakışı [11]

Farklı çalışanların zaman ekseni boyunca nasıl etkileştiğini göstermesinden farklı olarak bu görüş, kullanıcı uygulamasının kullandığı farklı Unify olayları hakkında bilgi sağlar. Bu olay çalışanın işleyişini gösteren çubuğun düzenini tanımlar. Yatay çubuk Unify olaylarına karşılık gelen ufak dörtgenlerden oluşur. Bu çubuklar veya çubukların arasındaki oklara tıklamak, olayın tipi hakkında daha fazla bilgi verir; olayın ne kadar sürdüğü ve aramayı yapmak için kullanılan anahtar parametre değerlerinin ne olduğu hakkında bilgi verir. Uygulamanın çalıştığı toplam zamanı gösteren izleme toplam zamanı pencerenin hemen üstünde sol tarafta gösterilir. Her olayın geçen zamanı pencerenin altında gösterilir. Detaylı fikir sahibi olmak için yaklaştırma ve uzaklaştırma da mümkündür.

5.5.1.2. Ağ bakışı

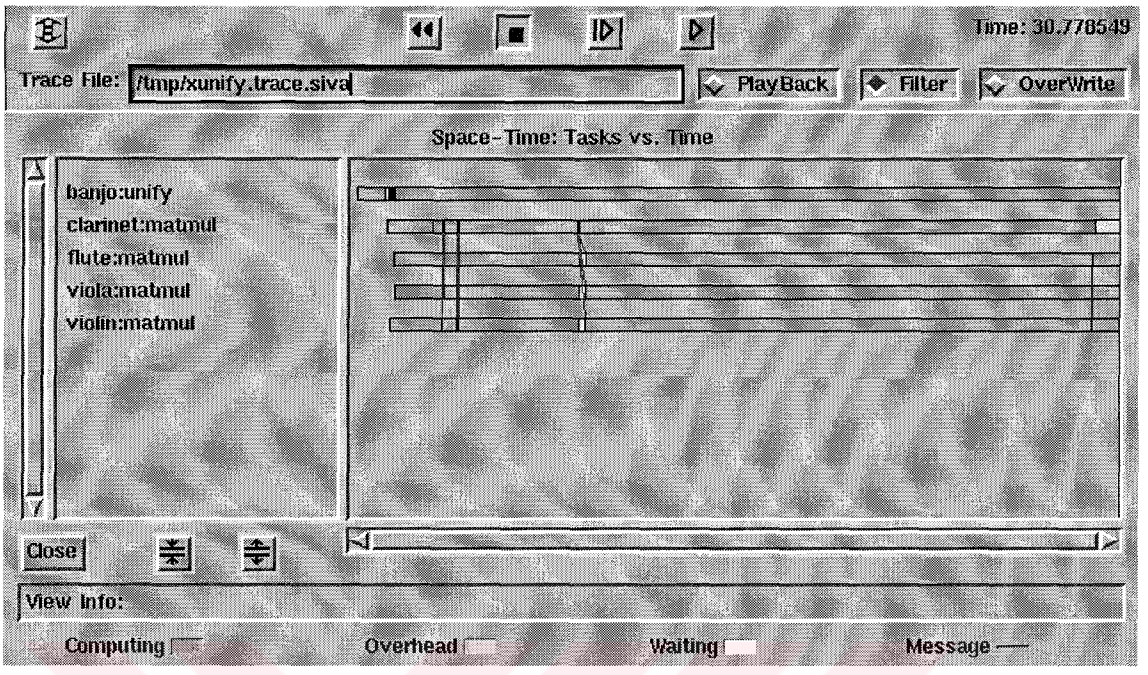
Bu bakış açısı, konfigürasyonu yapılandıran farklı makinaları gösterir. Makinalar, ağ topolojisine bakılmaksızın basit bir bus ile bağlanmış şekilde gösterilir. Makinalar, mimarisi ismini içeren simge tarafından gösterilmiştir. Şekil 5.3 ağ kuramının bir fotoğrafını gösterir. Ağ kuramı, uygulama çalıştığı zaman harekete geçer. Tuşlar makinanın durumuna göre renklenmiştir. Herhangi bir zamanda yeşil renk hesaplamayı, sarı artık zamanı ve beyaz host üzerinde herhangi bir görev olmadığını gösterir. Host' a bağlanan hattın kalınlığı , gönderilen veya alınan en son mesajda beliren ağ band genişliğine bağlı olarak değişir.



Şekil 5.3 Xunify ağ bakışı [11]

5.5.1.3. Olay filtreleme bakışı

Bu kuram tek bir parçada veya parçalar kümesinde yer alan olayları gösterir. Bu kuram izleme dosyasını geri isterken otopsi analizi sırasında kullanabilir. Uygulama çalışmayı bir kere durdurduğunda playback analizi sırasında kullanıcı bir parça kümesini tanımlayabilir ve sonra filtre düğmesini açtıktan sonra izlemeyi tekrar çalıştırabilir. Bu filtre olacak ve sadece kullanıcının seçtiği parçalar üzerinde olayları gösterecektir. Şekil 5.4 olay filtrelemesi kuramını gösterir.



Şekil 5.4 Xunify olay filtreleme kuramı [11]

Yararlanma kuramı, yüzde yaralanma kuramı, senkronizsyon gibi diğer kuramlar halen geliştirilmektedir [11].

5.6. PVM Sistemi

Geliştirilmesi programcılar ve onların kuruluşları tarafından gerçekleştirilen ve halen gelişimine devam eden PVM projesi, heterojen ağ hesaplamasının eski bir ürünüdür. Bu projenin genel amaçları heterojen ve eşzamanlı hesaplamanın önemli sorunlarını araştırmak ve bu sorunlara karşı çözümler üretmektir. PVM, birbirine bağlanmış çeşitli mimarideki bilgisayarlar arasında esnek bir heterojen eşzamanlı hesaplama çatısı oluşturmayı ilke edinen kütüphane ve yazılım araçlarının bütünleşmiş bir setidir. PVM sisteminin genel olarak amacı ise eşzamanlı veya paralel hesaplama için bilgisayarlar topluluğunun işbirliği içinde kullanılmasına olanak sağlamaktır.

Özet olarak PVM aşağıdaki temel özelliklere özellikler üzerine kurulmuştur:

- Kullanıcı İsteğine Bağlı Host Havuzu: PVM programında çalışılan uygulamaların hesaplama görevleri kullanıcı tarafından seçilen makina seti sayesinde yürütülür. Tek işlemcili bilgisayarların ve paylaşılabilen hafıza, dağıtık hafıza gibi özellikleri olan çok işlemcili bilgisayarların her ikisi de host havuzunun bir parçası olabilir. Hesaplama

boyunca host havuzuna makinalar eklenebilir veya çıkarılabilir ki bu, hata toleransı açısından çok önemli bir özelliktir.

- Donanıma Yarı Saydam Erişim: Uygulama programları ya donanım çevresini sanal işlem elemanlarının doğal özelliği olmayan bir topluluğu gibi görebilir veya kesin hesaplama işlemini yürüten en uygun bilgisayarların yerini belirleyerek host havuzundaki özel makinaların kapasitelerini kendi çıkarları için kullanmayı seçebilir.
- İşlem Tabanlı Hesaplama: PVM içindeki paralelliğin birimi, haberleşme ve hesaplama arasındaki değişimi sağlayan kontrolün bağımsız bir ardışık görevidir. İşlem-işlemci haritalaması içerilmez veya PVM tarafından zorlanmaz. Özel görevler tek bir işlemci üzerinde yürütülebilir.
- Açık Mesaj Geçiş Modeli : Her biri fonksiyonel veri veya hibrid dekompozisyon kullanan uygulamaların iş yükünün bir parçası bir diğerine açıkça gönderilen veya alınan mesajlara destek olan ölçümlemesel görevlerin kolleksiyonlarıdır.
- Heterojenetik Destek : PVM sistemi makinalar, ağlar ve uygulamalar bakımından heterojenliği destekler. Mesaj geçişini göz önüne alırsak, PVM faklı veri temsillerini içeren makinalar arasında değiştirebilen birden fazla veri tipi içeren mesajlara izin verir.
- Çoklu İşlemci Desteği : PVM temeldeki donanımın avantajını kullanabilmek için çoklu işlemci üzerindeki yerli mesaj geçiş olanaklarını kullanır. Satıcılar çoğunlukla kendi sistemleri için optimize edilmiş olan ve hala diğer genel PVM versiyonlarıyla haberleşebilen kendi PVM 'lerini savunurlar [11].

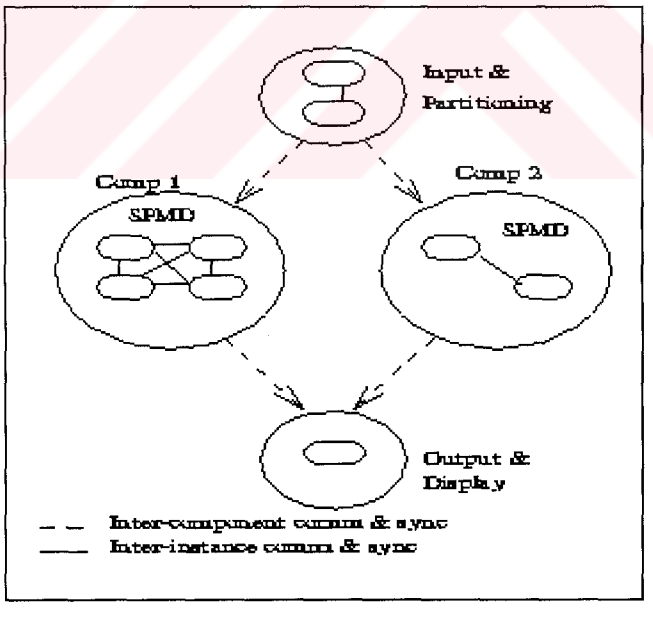
PVM sistemi iki parçadan meydana gelir. Birinci parça pvmd3 diye bilinen ve bazen pvmd diye kısaltılan DAEMON' dur ki sanal makinelerin oluşturduğu bütün mimarilerde bulunmaktadır. Daemon programına bir örnek olarak bir bilgisayarın bütün gelen ve giden elektronik mailleri tutması ve mail programının arka planda çalışmasıdır.

PVM herhangi bir kullanıcının geçerli bir şifreyle bir makine üzerine dameon kurabileceği şekilde tasarlanmıştır. Bir kullanıcı PVM uygulamasını çalıştırmak istediği zaman ilkönce PVM in başlangıcındaki sanal makineyi oluşturmalıdır. PVM uygulaması hostların herhangi birindeki Unix promptundan başlatılabilir. Çoklu kullanıcı sanal makinaların üst üste binmesini önleyebilir ve her bir kullanıcı çeşitli eşzamanlı PVM uygulamalarını yürütebilir.

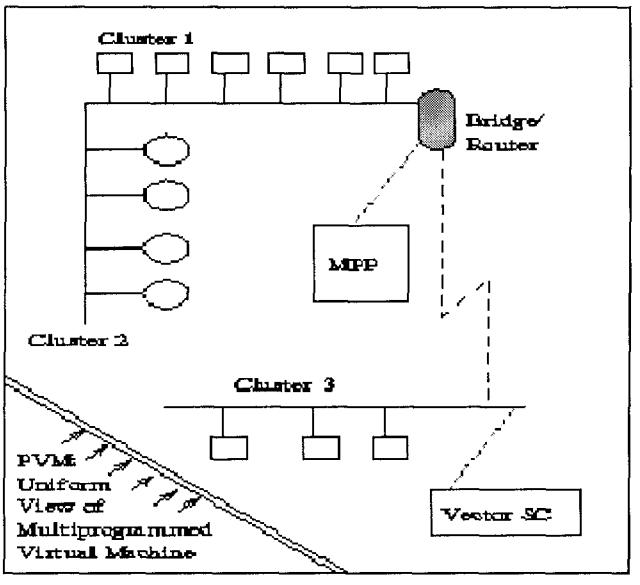
Sistemin ikinci parçası PVM ara yüz komutlarının bir kütüphanesidir. Bu kütüphane bir uygulamanın görevleri arasındaki işbirliği için gerekli olan ilkelerin fonksiyonel olarak tümünü

içermektedir. Ayrıca mesaj geçişi paylaşım işlemleri, işbirliği isteyen görevler ve sanal makinayı modifiye etmek için kullanıcı komutları içerir.

PVM hesaplama modeli çeşitli görevlerden oluşan bir uygulama fikri temeline oturtulmuştur. Her bir görev uygulamanın ölçümsel iş yükünün bir parçasından sorumludur. Bazen bir uygulama onun fonksiyonları boyunca paralelleştirilmiştir. Yani her bir görev giriş, problem kuruluşu, çözüm, çıkış ve göstermek gibi farklı bir fonksiyonu yerine getirir. Bu işlem sık sık fonksiyonel paralelizm olarak adlandırılır. Paralellemenin daha yaygın bir metodu veri paralelliği denilen bir uygulamadır. Bu metotta bütün görevler aynıdır ama her biri verinin sadece küçük bir parçasını bilir ve çözer. Bu aynı zamanda hesaplamanın tek-program çok-veri (SPMD) modeli olarak da bilinir. PVM bu metotların bir karışımını da destekler. Onların fonksiyonlarına bağlı olarak, görevler paralel olarak yürütülebilir ve genellikle bu bir olay olmamasına rağmen eşzamanlı olmaya veya veri değişimine ihtiyaç duyabilir. PVM hesaplama modelinin örnek bir diyagramı Şekil 5.5'de gösterilmiştir. Bir PVM sisteminin mimarisel görünüşü ve PVM tarafından desteklenen hesaplama platformlarının heterojenik görüntüsü Şekil 5.6'da verilmiştir.



Şekil 5.5 PVM hesaplama modeli [2]



Şekil 5.6 PVM' in mimarisel görünüşü [2]

PVM sistemi günümüzde C, C++ ve Fortran dillerini desteklemektedir. Bu dil arayüzlerinin setinin temelini; nesne-tabanlı diller ile yapılan deneylerde ortaya çıkan eğilim sayesinde hedef uygulamaların büyük çoğunluğu C ve Fortran dillerinde yazılması görüşü oluşturmaktadır.

PVM kullanıcı arayüz kütüphanesi için C ve C++, birçok C sistemlerinde kullanılan genel geleneklerin takip ettiği ve operasyon sistemleri gibi Unix içeren fonksiyonlar olarak gerçekleştirilir. Uygulama programları PVM kütüphane fonksiyonlarına standart dağıtımın bir parçası olan arşivsel bir kütüphane olan libpvm3 ile bağlantı kurarak erişmek için C ve C++ dillerinde yazılırlar.

Fortran dili ile fonksiyonlardan daha çok altprogramlar gibi gerçeklenirler. Bunun sebebi desteklenen mimarideki bazı derleyiciler Fortran fonksiyonları ile C fonksiyonları arasında güvenli bir arayüz oluşturamayacak olabilmesi, istenilen programa dönebilmek için durum sonuçlarının her bir PVM kütüphane araması içine ek bir tez tanıtılması gerekliliğidir. Ayrıca, mesaj tamponları içindeki şekillendirilmiş verilerin yerleşme ve düzenleme için bulunan kütüphane komutlarının veri tipini gösteren bir ek parametre ile birleştirilmiştir. Bu farklılıklardan başka, iki dil arasında birebir bir uyumluluk vardır. Fortranın PVM ile olan arayüzleri, uyumlu C komutlarına dönen kütüphane koçanları (stubs) gibi gerçekleştirilirler. Bundan dolayı Fortran uygulamaları en az C kütüphanesi kadar iyi olan stubs(koçan) kütüphanesi libfpvm3' e bağlanmaya ihtiyaç duyar [2].

Bütün PVM görevleri integer görev tanımlayıcısı (TID) tarafından tanımlanır. Mesajlar TID 'lere gönderilir ve TID 'ler den alınır. TID 'ler bütün sanal makineleri üzerinde eşsiz olmak zorunda olduğundan dolayı bölgesel (local) pvmd tarafından desteklenir ve kullanıcı seçimli değildir.

PVM ile yapılan uygulamalarda genel paradigma şu şekildedir. Bir kullanıcı PVM kütüphanesine yerleşen aramaları içeren bir veya daha fazla olan ardışık programları C, C++ veya Fortran 77 dillerinde yazar. Her bir program uygulamayı gerçekleyen bir görevle uyumludur. Bu programlar host havuzundaki her bir mimari için derlenir ve sonuç dosyaları host havuzundaki makinelerden erişebileceği bir yere yerleştirilir. Bir uygulamayı yürütmek için, bir kullanıcı bir görevin bir kopyasını host havuzu içindeki bir makine yardımıyla tipik olarak başlatır. Bu işlem diğer PVM görevlerini de arkasından gelecek şekilde başlatır. Sonunda aktif görevlerin bir koleksiyonu sonuçlanır. Böylece problemin çözülmesi için bölgesel hesaplamalar ve mesajların birbiriyle değiştirilmesi yapılmış olur.

PVM yazılımı, mevcut donanımı kullanarak paralel programların etkili ve doğru tarzda geliştirildiği birleştirilmiş bir yapı oluşturur. PVM, heterojen bilgisayar sistemlerinin tek bir paralel makina gibi görünmesini mümkün kılar. PVM, farklı mimarilere sahip bir ağda tüm mesaj yönlendirmelerini, veri dönüşümlerini ve veri şemalarını şeffaf bir şekilde yönetebilir.

PVM hesaplama modeli genel olarak basittir ve geniş bir uygulama programı çeşitliliği sağlar. Kullanıcı uygulamasını birlikte çalışan(işbirliği) görevlerin topluluğu olarak yazar. Görevler PVM kaynaklarına standarttır ve göreneklere arayüz kütüphanesi içinden ulaşır. Bu görenekler hem ağ hem de haberleşme ve görevler arası senkronizasyonda görevlerin başlamasına ve sonlandırılmasına izin verir.

İlk PVM mesaj ileticileri heterojen işletimlerle bağlantılıydı ve depolama ve iletim için güçlü yapılar içeriyordu. Haberleşme yapıları bunları veri yapılarının iletimi ve alımı için içerir. PVM görevleri keyfi kontrol ve bağımlılık yapıları içerebilir. Diğer bir deyişli aynı anda çalışan uygulamanın her hangi bir noktasında, mevcut olan herhangi bir görev diğer görevleri başlatabilir veya durdurabilir, gerçek makinadan bilgisayarları silebilir veya ekleyebilir. Herhangi bir işlem diğer biriyle haberleşebilir veya senkronizeleşebilir. Herhangi bir özel kontrol ve bağımlı yapı PVM sistemi altında uygun PVM yapıları ve host dili kontrol-akış yapıları kullanarak işletilebilir [2].

PVM gibi diğer yazılım paketleri de dağıtık hesaplama uygulamalarında kullanılmakta olup, birbirleri arasında benzerlikler vardır. Diğer bir tespit ise, aynı anda her yerde mevcut

olmasından (özellikle gerçek makina konsepti) ve de basit fakat tam programlama arayüzünden dolayı PVM sistemi, yüksek performans uygulamalarında geniş bir kabul görmektedir.

6. SONUÇLAR VE ÖNERİLER

Bu çalışmada, dağıtık hesaplama imkanı sunan PVM yazılım paketi kullanılarak bir sanal paralel makine oluşturulmuş ve paralel hesaplama yapan uygulamalar çalıştırılmıştır. Uygulamalarda farklı problem boyutu-işlemci sayısı için icra süreleri ölçülmüştür. Ölçüm sonuçları Matlab 6.0 ile grafiğe dökülmüş, paralellikten doğan performans değişimi gözlemlenmiştir. Bölüm 4'de anlatıldığı üzere, ölçüm sonuçları bazı tespitlerin yapılmasını gerekli kılmıştır. Bunlar, artan işlemci sayısının belli değerlerden sonra performansı düşürdüğü, bunun nedenin haberleşme ağırlıklı uygulamalarda artan işlemci sayısının haberleşmeden harcanan süreyi arttırdığı XPVM ile hesaplamanın izlenmesi sonucunda görülmüştür. Buna göre performansın optimum olduğu noktaların belirlenmesi ve performansın arttırılması için uygulamaların bu noktalarda çalıştırılması gerekir.

PVM ortamında çözülecek bir problem önce analiz edilmeli ve ortamı oluşturan hesaplama kaynakları arasında görev dağıtımı yapılmalıdır. Dengeli bir çalışma için dağıtılan görevlerin icra sürelerinin işlemciler arasında çok farklı olmamasına dikkat edilmelidir. Bunun için ortamı oluşturan işlemcilerin performansları da dikkate alınmalıdır. Örneğin yüksek performanslı bir işlemciye daha çok görev, eski ve düşük performanslı, veya aşırı yüklü işlemciye ise daha az görev verilebilir. Ayrıca, işlemciler arası mesajlaşma, görev süreleri mümkün mertebe büyük seçilerek en aza indirilmelidir. Ayrıca sanal paralel ortam değiştiğinde (işlemci sayısı veya yüksek performanslı bir işlemci yerine düşük performanslı bir işlemci konması gibi), programın tekrar elden geçmesi gerekebilir. Bu nedenle PVM gibi dağıtılmış paralel ortamlarda verimli çalışacak program yazmak oldukça zahmetli bir iştir.

Paralel sistemlerde performansın gözlemlenmesi ve analizi kaynakların verimli kullanılması için önemlidir. Paralel sistemi oluşturan işlemcilerin sayısı, performansı, paralel sistemde çözülmek istenen problemin boyutu ve iletişim elemanlarının performansı genel performansı doğrudan etkilediğinden paralel sistemlerde performans analizi çok faktörlü karmaşık bir probleme dönüşmektedir.

Çok işlemcili bir sistemde, problemin boyutuna göre kaç tane işlemci kullanılırsa kaynaklar en verimli bir şekilde kullanılmış olur. Bazı problemlerde çok fazla işlemci kullanmak performansın artmasından ziyade azalmasına sebep olabilir. Karmaşıklık analizi denilen bu tür çalışmalarda problemin boyutuyla kullanılan işlemci sayısı arasındaki ilişki çıkartılmaya çalışılır [6]. Biz bu çalışmada yukarıda bahsedilen örnek uygulama için en ideal işlemci sayısı-problem boyutu konfigürasyonunu, problem boyutu 2000 ve 2 işlemci kullanılarak yapılan hesaplama ile olduğunu bulduk.

Paralel sistemlerde performans analizi ve iyileştirmesi tek işlemcili sistemlere göre daha önemlidir. Amaç yüksek performans elde etmek olduğundan, pahalı kaynakların en verimli şekilde kullanılmasına dikkat edilmelidir. Ancak, paralel sistemlerde performans birçok faktöre bağlı olduğundan, dikkat edilmezse tek işlemcili sistemden bile kötü olabilir.

Tek işlemcili sistemlerde, programı oluşturan komutlar ardışık olarak icra edilirler. Performans işlemci hızıyla doğru orantılı olarak değiştiğinden, program içinde icrası en uzun süren fonksiyonda bir iyileştirme yapıldığında genel performans da iyileştirilmiş olur. Oysa dağıtık sistemlerde durum bundan farklıdır. Aynı anda birden fazla noktada programın komutları farklı işlemciler tarafından icra edildiğinden, performans iyileştirmesi için programın neresinde iyileştirme yapılması gerektiği hemen anlaşılamaz ve seri sistemlerde icra süresi ölçümü için kullanılan "gprof" (GNU-profiler) gibi araçlar da performans analizi için yetersiz kalır [7].

Oysa, paralel performans analizi sonucunda, programın herhangi bir yerinde yapılan bir iyileştirmenin genel performansı ne kadar etkileyeceğinin bilinmesi istenir. Bu tür sistemlerin performansı, sadece işlemcilerin performansına bağlı olmadığından, işlemciler arasındaki koordinasyon ve iletişimin de hesaba katılması gereklidir. Herhangi bir fonksiyonda yapılan iyileştirme genel performansı iyi yönde etkilemeyebilir. Bu amaçla, paralel sistemlerde performans analizi için "Kritik Yol Analizi", "Slack" gibi bazı metotlar geliştirilmiştir [6].

Paralel performans ölçümleri için çok çeşitli metrikler kullanılmakla beraber en temel performans metriği cevap süresidir. Cevap süresi, bir programın başlangıcından o programa ait son prosesin bitişine kadar geçen süreye denir. Ölçülen bu süre programın gerçek performans ölçüsüdür. Bununla beraber, eğer performansın bedeli de ölçülmek istenirse, hızlanma ve efektiflik kullanılabilecek diğer metriklerdir [6].

Bu çalışmada PVM: "Parallel Virtual Machine" deneysel ortamını kullanarak paralel programlarda performans ve karmaşıklık analizleri yapılmıştır. Performans ölçümleri için cevap süresi metriğinden yararlanılmış, ancak problemleri açıklayabilmek için ise program aktivite grafiğine bakılmıştır. Karmaşıklık analizi sırasında ise bir problemin farklı boyut-işlemci sayıları için icra sürelerinin grafikleri elde edilmiştir.

Performans ölçümleriyle paralel sistemler daha verimli çalışacak şekilde yönetilebilirler veya paralel programlar değişik giriş ve işlemci konfigürasyonlarında daha verimli çalışacak şekilde ayarlanabilirler [7]. Gözlemleme sistemine bağlı performans araçları, istenilen metrik için toplanan performans verisini işleyerek, ölçüm sonuçlarını rapor ederler [7]. Ayrıca, az sayıda giriş ve işlemci konfigürasyonuyla denenen bir paralel programın, farklı sayıda giriş

büyüklüğü ve farklı sayıda işlemcili sistemlerde nasıl çalışacağı öngörülebilir ve performansının giriş büyüklüğü ve işlemci sayısı açısından maksimum olacağı çalışma bölgesi bulunabilir.

Bu çalışmaya ilave olarak, programın uygun bölümlerine sensörler ilave edilebilir ve performans verisi toplanabilir. Elde edilen veriden kritik yol, slack, gprof gibi paralel performans metrikleri belirlenerek uygulamanın performansı daha detaylı irdelenebilir.

KAYNAKLAR DİZİNİ

[1] Çetin Görkem, Çelik Kaan Güneş, Linux Ağ Yönetimi, Seçkin Yayıncılık, Ağustos 2000. pp.30-32.

[2] Geist Al, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994.

[3] Griffioen J., Yavatkar R., Finkel R., Unify: A Scalable, Loosely-Coupled, Distributed Shared Memory Multicomputer, University of Kentucky, Computer Science Dept. Technical Report, Ocak 1993.

[4] Manchek B., Heterogeneous Distributed Computing, 1995.

[5] Manchek Robert, "Workstation Solutions: Opening the Door to Heterogeneous Network Supercomputing",Supercomputing Review Volume 4, Number 9, Utah Department of Mathematics - University of Utah, USA, September, 1991, pp.44-46.

[6] Özmen A., Durmuş B., "Dağıtık Sistemlerde Paralel Performans ve Karmaşıklık Analizi",1.Ulusal Yüksek Performanslı Bilişim Sempozyumu, Gebze-Kocaeli, 24-25 Ekim 2002, pp 45-49.

[7] Özmen A., Durmuş B., "Paralel Performans Metrikleri", Elektrik-Elektronik-Bilgisayar Mühendisliği 9. Ulusal Kongresi, Kocaeli, Eylül 2001, pp 269-272.

[8] Patterson David A., Hennessy John L., Computer Organization & Design The Hardware/Software Interface , University of Wisconsin, 1994.

[9] Tanenbaum Andrew S., Distributed Operating Sysstems, 614 Seiten-Prentice Hall. January 1995.

[10] Zargham Mehdi R., Computer Architecture single and parallel systems Prentice Hall. Upper saddle river. New Jersey 07458, 1996.

[11] www.dcs.uky.edu/~unify/.

[12] http://www.lindaspaces.com/products/linda.html

[13] http://radio.weblogs.com/0112083/stories/2002/08/22/usingMpiInParallelProgramm.html

EK 1: Farklı problem boyutu-işlemci sayısı konfigürasyon denemelerinden elde edilen sonuçlar

Çizelge 4.1.a İşlem boyutu 1000 ve tek işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	0.360u	0.010s	0:01.11	%33.3
2. Deneme	0.360u	0.040s	0:01.11	%36.0
3. Deneme	0.380u	0.030s	0:01.12	%36.6
4. Deneme	0.360u	0.030s	0:03.87	%10.0
5. Deneme	0.380u	0.020s	0:00.66	%60.6
Ortalama	0.368u	0.026s	0:01.57	%25.1

Çizelge 4.1.b Problem boyutu 1000 ve 2 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi	Hesaplama süresi oranı
1. Deneme	0.380u	0.030s	0:00.95	%43.1
2. Deneme	0.370u	0.040s	0:02.62	%15.6
3. Deneme	0.360u	0.030s	0:00.95	%41.0
4. Deneme	0.360u	0.030s	0:02.37	%16.0
5. Deneme	0.380u	0.050s	0:00.94	%45.7
Ortalama	0.370u	0.034s	0:01.56	%25.9

Çizelge 4.1.c Problem boyutu 1000 ve 4 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	0.370u	0.030s	0:01.71	%23.3
2. Deneme	0.390u	0.060s	0:01.91	%23.5
3. Deneme	0.370u	0.040s	0:01.59	%25.7
4. Deneme	0.360u	0.060s	0:01.43	%25.3
5. Deneme	0.360u	0.050s	0:00.89	%46.0
Ortalama	0.370u	0.044s	0:01.51	%27.42

Çizelge 4.2.a Problem boyutu 2000 ve tek işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	0.700u	0.080s	0:08.04	%9.7
2. Deneme	0.720u	0.070s	0:01.28	%61.7
3. Deneme	0.670u	0.090s	0:02.19	%34.7
4. Deneme	0.730u	0.060s	0:02.18	%36.2
5. Deneme	0.690u	0.030s	0:02.20	%32.7
Ortalama	0.700u	0.066s	0:03.18	%24.09

Çizelge 4.2.b Problem boyutu 2000 ve 2 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	0.700u	0.060s	0:04.83	%15.7
2. Deneme	0.750u	0.060s	0:01.62	%50.0
3. Deneme	0.690u	0.070s	0:04.69	%16.2
4. Deneme	0.730u	0.040s	0:01.64	%46.9
5. Deneme	0.700u	0.060s	0:01.67	%45.5
Ortalama	0.714u	0.058s	0:02.89	%26.71

Çizelge 4.2.c Problem boyutu 2000 ve 4 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	0.710u	0.110s	0:02.66	%30.8
2. Deneme	0.730u	0.080s	0:01.66	%48.7
3. Deneme	0.700u	0.100s	0:03.02	%26.4
4. Deneme	0.700u	0.050s	0:02.92	%25.6
5. Deneme	0.710u	0.050s	0:02.82	%26.9
Ortalama	0.710u	0.078s	0:02.62	%30.07

Çizelge 4.3.a Problem boyutu 3000 ve tek işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.110u	0.080s	0:03.40	%35.0
2. Deneme	1.150u	0.020s	0:03.35	%34.9
3. Deneme	1.120u	0.020s	0:03.35	%34.0
4. Deneme	1.120u	0.060s	0:12.07	%9.7
5. Deneme	1.100u	0.040s	0:02.00	%57.0
Ortalama	1.120u	0.044s	0:04.83	%24.1

Çizelge 4.3.b Problem boyutu 3000 ve 2 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.140u	0.090s	0:02.51	%49.0
2. Deneme	1.130u	0.110s	0:06.48	%19.1
3. Deneme	1.090u	0.090s	0:02.44	%48.3
4. Deneme	1.100u	0.190s	0:02.45	%52.6
5. Deneme	1.150u	0.030s	0:07.27	%16.2
Ortalama	1.122u	0.102s	0:04.23	%28.94

Çizelge 4.3.c Problem boyutu 3000 ve 3 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.090u	0.100s	0:02.31	%51.5
2. Deneme	1.130u	0.090s	0:05.13	%23.7
3. Deneme	1.130u	0.120s	0:04.92	%25.4
4. Deneme	1.140u	0.120s	0:02.33	%54.0
5. Deneme	1.120u	0.030s	0:05.16	%22.2
Ortalama	1.122u	0.092s	0:03.97	%30.56

Çizelge 4.3.d Problem boyutu 3000 ve 4 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.090u	0.090s	0:03.81	%30.9
2. Deneme	1.130u	0.110s	0:02.46	%50.4
3. Deneme	1.140u	0.030s	0:04.30	%27.2
4. Deneme	1.130u	0.090s	0:04.25	%28.7
5. Deneme	1.150u	0.070s	0:04.10	%29.7
Ortalama	1.128u	0.078s	0:03.78	%31.9

Çizelge 4.4.a Problem boyutu 4000 ve tek işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.470u	0.060s	0:05.07	%30.1
2. Deneme	1.440u	0.080s	0:04.41	%34.4
3. Deneme	1.430u	0.040s	0:04.41	%33.3
4. Deneme	1.430u	0.520s	0:03.13	%62.3
5. Deneme	1.450u	0.050s	0:04.59	%32.6
Ortalama	1.444u	0.150s	0:04.32	%36.9

Çizelge 4.4.b Problem boyutu 4000 ve 2 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.440u	0.110s	0:03.20	%48.4
2. Deneme	1.420u	0.090s	0:08.98	%16.8
3. Deneme	1.460u	0.090s	0:03.18	%48.7
4. Deneme	1.470u	0.080s	0:08.52	%18.1
5. Deneme	1.410u	0.170s	0:03.23	%48.9
Ortalama	1.440u	0.108s	0:05.42	%28.56

Çizelge 4.4.c Problem boyutu 4000 ve 4 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.420u	0.170s	0:05.44	%29.2
2. Deneme	1.440u	0.090s	0:05.08	%30.1
3. Deneme	1.430u	0.120s	0:03.18	%48.7
4. Deneme	1.470u	0.100s	0:05.85	%26.8
5. Deneme	1.420u	0.130s	0:05.60	%27.6
Ortalama	1.436u	0.122s	0:05.03	%30.97

Çizelge 4.5.a Problem boyutu 5000 ve tek işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.820u	0.100s	0:05.56	%34.5
2. Deneme	1.840u	0.100s	0:06.53	%29.7
3. Deneme	1.830u	0.160s	0:04.60	%43.2
4. Deneme	1.840u	0.190s	0:06.41	%31.6
5. Deneme	1.820u	0.120s	0:05.77	%33.6
Ortalama	1.830u	0.134s	0:05.77	%34.04

Çizelge 4.5.b Problem boyutu 5000 ve 2 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.810u	0.130s	0:10.70	%18.1
2. Deneme	1.860u	0.150s	0:04.26	%47.1
3. Deneme	1.770u	0.230s	0:04.24	%47.1
4. Deneme	1.840u	0.100s	0:11.26	%17.2
5. Deneme	1.890u	0.300s	0:04.94	%44.3
Ortalama	1.834u	0.182s	0:07.08	%28.47

Çizelge 4.5.c Problem boyutu 5000 ve 4 işlemci için icra süreleri

	İşlemci süresi	Sistem süresi	İcra süresi(sn)	Hesaplama süresi oranı
1. Deneme	1.850u	0.110s	0:06.95	%28.2
2. Deneme	1.820u	0.180s	0:06.70	%29.8
3. Deneme	1.870u	0.180s	0:06.42	%31.9
4. Deneme	1.860u	0.170s	0:03.57	%56.8
5. Deneme	1.860u	0.300s	0:07.22	%29.9
Ortalama	1.852u	0.188s	0:06.17	%33.06