EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

ÇARPANLARINA AYIRMA ALGORİTMALARININ PARALELLEŞTİRİLMESİ

Selçuk KESKİN

Tez Danışmanı: Prof. Dr. Urfat NURİYEV

Matematik Anabilim Dalı

Bilim Dalı Kodu: 619.03.03
Sunuş Tarihi: 09.06.2011

Bornova – İZMİR
2011

Selçuk KESKİN tarafından YÜKSEK LİSANS tezi olarak sunulan **"ÇARPANLARINA AYIRMA ALGORİTMALARININ PARALELLEŞTİRİLMESİ"** başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **09.06.2011** tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri: **İmza**

Jüri Başkanı **: .....** **.....**

Raportör Üye **: .....** **.....**

Üye **: .....** **.....**

ÖZET

ÇARPANLARINA AYIRMA ALGORİTMALARININ PARALELLEŞTİRİLMESİ

KESKİN, Selçuk

Yüksek Lisans Tezi, Matematik Bölümü
Tez Yöneticisi: Prof. Dr. Urfat NURİYEV
Haziran 2011, 85 sayfa

Açık anahtarlı şifreleme algoritmalarının güvenliği çarpanlara ayırma probleminin matematiksel zorluğuna dayanır. Günümüzün en sağlam veri şifreleme algoritmalarının başında gelen RSA Algoritması Asal çarpanlarına ayırma probleminin büyük sayılar için bilgisayarda etkin bir şekilde çözülememesi prensibine dayanmaktadır.

Çarpanlara ayırma problemi, “p” ve “q” gibi iki büyük asal sayının çarpımından oluşan “n” sayısı verildiğinde, “p” ve “q” sayılarının bulunmasıdır. Asal çarpanların bulunması problemi sayılar büyüdükçe çok karmaşık bir hal almaktadır.

Bu çalışmada şifreleme algoritmalarına karşı yapılan saldırıların (kriptanaliz) temelinde duran tamsayıların çarpanlarına ayırma algoritmaları üstünde durulmuş, farklı çarpanlara ayırma yöntemleri incelenerek yeni çarpanlara ayırma algoritmaları geliştirilmiştir ve onların paralel versiyonları tasarlanmıştır.

Önerilen algoritmaların MPI ve GMP kütüphaneleri kullanılarak C dilinde programları tasarlanmış ve hesaplama denemeleri yapılmıştır. Çıkan sonuçlar grafik ve tablolarla gösterilmiştir.

Anahtar Kelimeler: Asal sayılar, Çarpanlarına Ayırma Algoritmaları, Paralel Algoritmalar, GMP Kütüphanesi, MPI Kütüphanesi

ABSTRACT

ON PARALLELIZATION OF PRIME FACTORIZATION ALGORITHMS

KESKİN, Selçuk

MSc. in Mathematics
Supervisor: Professor Dr. Urfat NURİYEV
June 2011, 85 pages

Security of the public key encryption algorithms are based on the fact that the prime factorization problems is mathematically hard. RSA, which is one of the most reliable encryption algorithm, also relies on the principle that prime factorization of the big numbers can not be computed efficiently with computers.

Prime factorization problem is the problem of finding the numbers “p” and “q”, given the number “n” which is consists of the product of two big prime numbers “p” and “q”. The prime factorization problems is too complex when the numbers are big enough.

In this work, the prime factorization algorithms, the center of the attacks against the encryption algorithms, are emphasized, different prime factorization algorithms are studied and new prime factorization algorithms are proposed. The parallel versions of the algorithms are designed.

Proposed algorithms were implemented in C using MPI and GMP libraries and computational experiments were done. The results shown in graphs and tables.

Keywords: Prime numbers, prime factorization algorithms, parallel algorithms, GMP Library, MPI Library

TEŞEKKÜR

Bu tez çalışması süresince tecrübelerini ve bilgilerini paylaşan danışman hocam sayın Prof. Dr. Urfat NURİYEV'e, desteğini esirgemeyen hocam sayın Prof. Dr. Alpay KIRLANGIÇ'a, maddi-manevi her türlü desteği sunduğu için aileme, hiçbir zaman yanımdan ayrılmayan benimle birlikte aynı stresi çeken yardımlarını esirgemeyen Melek'ime SONSUZ teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ÇİZELGELER DİZİNİ

Çizelge Sayfa

KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
Aİ	Associative (Çağrışımlı) Sistemler
ALU	Arithmetic Logic Unit (Aritmetik Mantık Birimi)
B	Bit (Verilerin İşlemcilerde bitlerle İşlemi)
CPU	Central Processing Unit (Merkezi İşlem Birimi)
ÇAH	Çok Ara Hatlı Sistemler (Çok Girişli Bellek Kullanılır)
ÇBS	Çapraz Bağlantı Santralli Sistemler.
ÇEÇV	Çok Emir ve Çok Veri Akışlı Sistemler
ÇETV	Çok Emir ve Tek Veri Akışlı Sistemler
DB	Bilgisayar Sistemi Elemanlarının Düşük Bağlantı Düzeyi
EB	Emirler Belleği
GDBB	Genel Dış Bellek Biriminin Yardımı ile Bağlantılı Sistemler
GNU	Genel Halka Açık Lisans (Linux İşletim Sistemi kavramı)
İB	Veri İşleme Birimi
İ-İ	İşlemciler Arasında Direk Bağlantılı Sistemler
KB	Tek Kontrol Birimi
KK	Kanal -Kanal Bağlantılı Sistemler

KISALTMALAR DİZİNİ (devam)

Kısaltmalar	Açıklama
LAM	Yerel Alan Çoklu-bilgisayarları (Local Area Multicomputers)
MD	Multiple Data Stream (Çoklu Veri Akışı)
MI	Multiple Instruction Stream (Çok Emir Akışı)
MPI	Mesaj-Geçme Arayüzü (Message Passing Interface)
RSA	Rivest, Shamir and Adleman (ilk tarif edenler)
SD	Single Data Stream (Tek Veri Akışı)
SI	Single Instruction Stream (Tek Emir Akışı)
TAH	Tek Genel Ara Hatlı Sistemler
TEÇV	Tek Emir ve Çok Veri Akışlı Sistemler
TETV	Tek Emir ve Tek Veri Akışlı Sistemler
VB	Veriler Belleği
W	Word (Verilerin İşlemcilerde Sözlerle İşlemi)
YB	Bilgisayar Sistemi Elemanlarının Yüksek Bağlantı Düzeyi

1. GİRİŞ

Günümüzün en sağlam veri şifreleme algoritmalarının başında gelen RSA Algoritması Asal çarpanlarına ayırma probleminin büyük sayılar için bilgisayarda etkin bir şekilde çözülememesi prensibine dayanmaktadır. (Nabiyev, 2007) Çarpanlara ayırma problemi, p ve q gibi iki büyük asal sayının çarpımından oluşan n sayısı verildiğinde, p ve q sayılarının bulunmasıdır (Pomerance et al., 1984). Asal çarpanların bulunması problemi sayılar büyüdükçe çok karmaşık bir hal almaktadır. (Riesel, 1985) Bu sebepten günümüzde en yaygın kullanıma sahip ağ güvenliği protokolleri genellikle RSA tabanlıdır. (Cormen et al., 2002) Bu nedenle çarpanlara ayırma problemi pratikte çok önem kazanmıştır. (Giblin, 1993)

İlk bilgisayarlar hem aritmetik işlemleri yapmak hem de grafik çizmek gibi birçok özellikleri aynı anda kullanıcıya sağlamak yerine, sadece tek bir işlemi yapmak için özel olarak tasarlanmıştır. Örneğin hesap makinesi, aritmetik işlem yapmasına rağmen bir kelime-işlemcinin bize sağladığı metin yazma ve düzeltme avantajını sağlayamaz. Bu tarz ilk bilgisayarların tasarlandıkları amacın dışında başka işleri de yapmaları için bütün parçalarının baştan inşa edilmesi gerekiyordu. Ünlü Amerikalı matematikçi ve bilgisayar bilimcisi John Von Neumann'ın bilgisayarın iç yapısına ve çalışma prensibine dair tasarladığı "Von Neumann Mimarisi" bilgisayarın işlemcisini hafızadan ayırarak bu soruna çözüm bulmuştur. Bu mimari şaşırtıcı bir biçimde hala günümüz bilgisayarlarının bir anlamda temelini oluşturmaktadır. (Leighton, 1992)

1980'lerin öncesinde, bilgisayar mimarisi bigisayardaki hız ve bellek kapasitesi olarak sınırlıydı. Bu zamanın bazı bilgisayarları belli programları kullanırlar ve 64KB hafızaya sahiptirler. Günümüzdeki kişisel bilgisayarlar ise o zamanki bilgisayarlardan 1000 kat daha fazla belleğe sahiptir. Günümüz yazılımlarının daha karmaşık ve günümüz bilgisayarlarının daha verimli olmasına rağmen bu değişiklikler program dizaynındaki verimliliği gözardı edebileceğimiz anlamına gelmez. (Mattson et al., 2004)

Bu tezde tamsayıların çarpanlara ayrılması problemi, büyük sayılarla işlem yapmak için kullanılan GMP kütüphanesi ve paralel programlama için MPI kütüphanesi ele alınmış, Fermat'ın çarpanlara ayırma yöntemi incelenerek yeni çarpanlara ayırma algoritmaları geliştirilmiş, onların paralel versiyonları tasarlanarak hesaplama denemeleri yapılmıştır.

Tez; Giriş, 7 Bölüm, Sonuç ve Kaynaklar dizininden oluşmaktadır.

Giriş'de problemin önemine değinerek genel tanımlar verilmiş ve izleyen bölümlerin içerikleri özet olarak açıklanmıştır.

İkinci bölüm'de Paralel Programlama hakkında genel bilgi verilmiş ve paralel bilgisayar yapılarından, paralel algoritmalardan bahsedilmiştir.

Üçüncü bölüm'de GMP Kütüphanesi tanıtılmıştır. Kullanılan veri tipleri ve hazır fonksiyonlar detaylı açıklanmıştır. Özel tanımlı fonksiyonlara da yer verilmiştir.

Dördüncü bölüm'de MPI Kütüphanesi hakkında genel bilgi verilmiştir. Tez çalışmasında kullanılan fonksiyonlar incelenmiştir.

*Beşinci bölüm'*de çarpanlarına ayırma problemi ele alınmış, Basit Bölme ve Fermat'ın çarpanlara ayırma yönteminden bahsedilmiştir. Bu algoritmaların paralel çözümlerinin nasıl olabileceği hakkında açıklama yapılmıştır.

Altıncı bölüm'de Sayıların Çarpanlarına göre yapısı verilmiş, yeni parametreler tanımlanmış, bu parametrelerin hesaplanmasına dayanan farklı algoritmaların paralel versiyonları geliştirilmiştir.

Yedinci bölüm'de diğer bölümlerde bahsedilen algoritmaların karşılaştırılması yapılmıştır. Sonuçlar grafikler ile gösterilmiştir.

Sonuç bölümü'nde tezde elde edilen sonuçlardan özet olarak bahsedilmiştir.

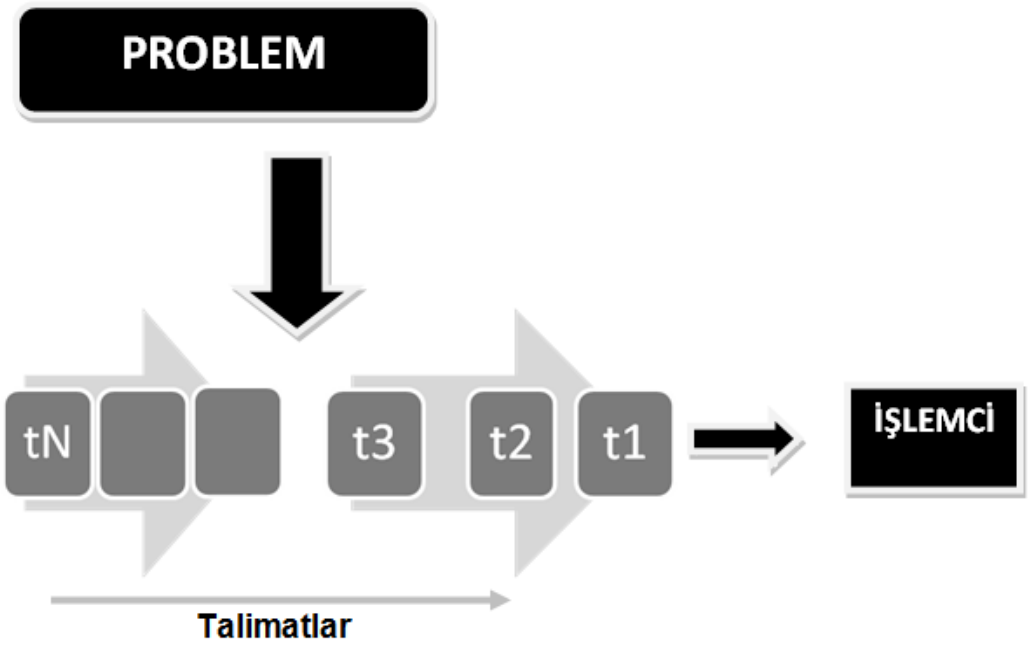
Tezin son bölümünde ise *kaynaklar dizini* verilmiştir.

EK bölümü'nde tezde bahsedilen programların kodları verilmiştir.

2. PARALEL PROGRAMLAMA

Paralel hesaplama, aynı görevin (parçalara bölünmüş ve uyarlanmış), sonuçları daha hızlı elde etmek için çoklu işlemcilerde eş zamanlı olarak işletilmesidir. Bu fikir, problemlerin çözümünün küçük görev parçalarına bölünmesi ve bunların eş zamanlı olarak koordine edilmesine dayanır. (Barney, 2011)

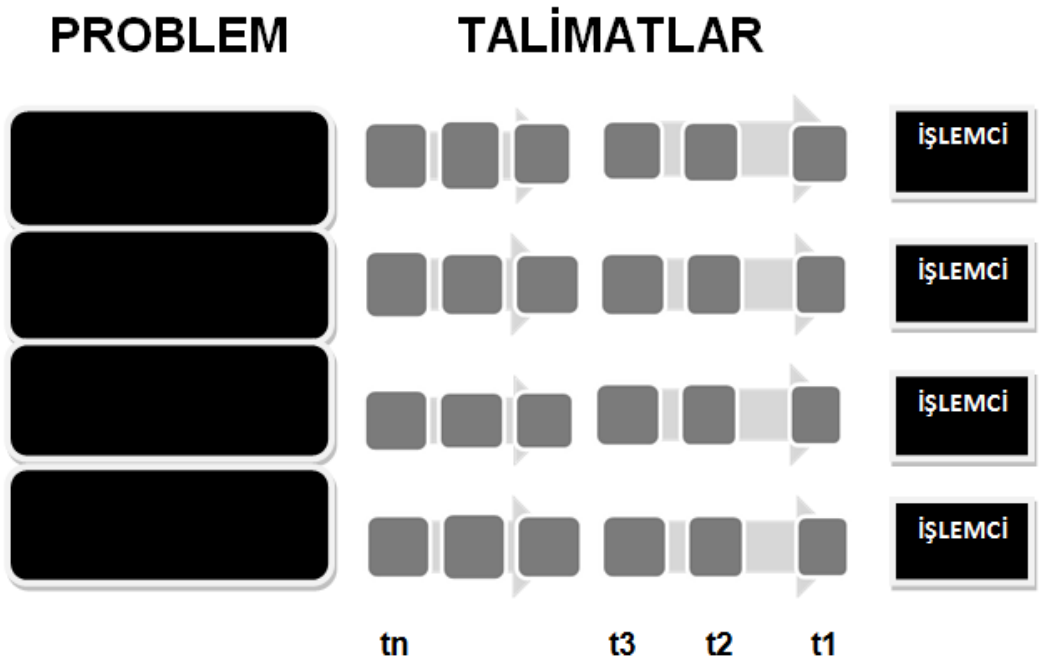
- Genellikle, programlar sequential (sıralı/ardışıl) hesaplama için yazılır: (Xavier et al., 1998)
    - Tek işlemcisi olan tek bir bilgisayarda çalıştırılır.
    - Problem talimatların ayrık bir serisine bölünür.
    - Talimatlar ardışık (birinden sonra diğeri) şekilde yürütülür.
    - Aynı zamanda sadece bir talimat yürütülebilir.



Şekil 2.1 Ardışıl program akış diyagramı

- Kolaylık açısından, *paralel hesaplama* çoklu hesaplama kaynaklarının eşzamanlı kullanılarak problemin çözülmesidir. (Rauber, 2010)
    - Birden fazla işlemci kullanılarak yürütülür.
    - Problem aynı anda çözülebilecek ayrık parçalara bölünür.
    - Ayrıca herbir parça ardışıl haldeki talimatlara bölünür.

- Herbir parçadaki talimatlar eşzamanlı olarak farklı işlemcilerde yürütülür.



Şekil 2.2 Paralel program akış diyagramı

2.1 Paralel Programlamanın Amacı

Gerçek Zaman Problemi, bilgisayar ve bilgisayar sistemlerinin kullanım alanlarının günden güne hızla arttığı bir dönemde, kullanım alanlarından bir çoğu, bilgisayarın son derece yüksek karakteristiklere ve kaliteye sahip olmasını gerektirir. Çok geniş olarak ortaya çıkan talep, yüksek performanstır. Performans talebi, genellikle bilgisayarlar kontrol ve modelleme amaçları için kullanıldığında ortaya çıkar. (Miller et al., 2006)

Eğer bilgisayarlar, herhangi bir hedefi, örneğin; uçakları, kozmik gemileri, su gemilerini, robotları, teknolojik işlemleri, füzeleri vb. kontrol etmek için kullanılırsa, şüphesiz, onlar, kontrol hedeflerindeki gerçek işlemleri önlemekle çalışmalıdırlar veya en azından gerçek zaman ölçüsünde işlem yapmalıdırlar. Aksi halde büyük faciaların, kayıpların ortaya çıkmasına neden olabilirler.

Çoğu zaman çok hızlı değişen ve çok kısa zamanda oluşan işlemleri kontrol etmek gerekir. Bu durumda, yüksek performansla beraber yüksek doğruluk istenir.

Gereken doğruluk ne kadar yüksek olursa, hesaplama hacmi bir o kadar fazla olacak ve bu hesaplamayı yapmak için fazla zaman talep edilecektir. Bir çok durumda gerçek zaman ölçüsünde çalışırken doğruluk ve hesaplama hızı ters orantılı olur.

Öte yandan, kontrol edilecek hedef ne kadar hızlı değişirse, kontrol sistemi, yani bilgisayar bir o kadar hızlı çalışmalıdır. Buna göre, eğer yüksek doğruluk ve hızlı kontrol etmek gerekirse, o zaman kontrol eden bilgisayarın çok yüksek performansa sahip olması gerekir. Bilgisayar yüksek performansa sahip olduğunda ise gereken doğruluğu ve gerçek zaman ölçüsünde kontrolü gerçekleştirebilecektir.

Aynı problem, bilgisayarların karmaşık hedefleri ve çok hızlı değişen süreçler ve olayları modellendirmek amacıyla kullanıldığında da ortaya çıkar. Bu modeller gerçek zaman ölçüsünde çalışmalıdırlar. Bir çok durumda hedeflenen amaca ulaşmak için modellenme yüksek doğrulukla yapılmalıdır. (Samedov, 2002)

Bir çok durumda modellenecek işlemler ve hedefler yüksek hızla değişirler. Modellerin gerçek işlemlere ve hedeflere maksimum yaklaşması için sayısal modellerin yüksek hıza ve doğruluğa sahip olması gerekir. Böyle durum, benzer işlem ve hedeflerin modellenmesi için yüksek hıza sahip olan ve hesaplamaları yüksek doğrulukla yapabilen bilgisayarların kullanılmasını gerektirir.

Bir çok modern kontrol etme ve modellenme problemlerinde kontrol edilecek veya modellenecek işlem ve hedefler binlerle ve hatta on binlerle Hz frekansla değişecek parametrelerle karakterize edilirler. Öte yandan, oluşacak hata, kontrol edilen ve modellenen parametrelerin maksimum değerinin 10-5 - 10-6‘sından büyük olamaz.

En yüksek performansa sahip olan çağdaş tek bilgisayarlı veya tek mikroişlemcili hesaplama makineleri gösterilen şartlarla, gereken bilgiyi gerçek zaman ölçüsünde hesaplayamazlar. Buna göre hesaplama sistemlerinin gerçek zaman şartlarında çalışabilecek performansa erişmesi için başka metotlar aramak gerekir.

Bilgisayar gücü; bir çok alanlarda çok büyük hacimdeki bilgi üzerinde işlem yapmak gerekmektedir. Bu işlemi gerçekleştirmek için yeterli zaman süresi ayrılmaktadır. Böyle alanlara karmaşık konstrüksiyonların hesaplanması (örneğin; uçakların, büyük ısı makinelerinin, büyük gemilerin, nükleer reaktörlerinin, mühendislik projelerinin) ve diğer problemler (aerohoparlör, meteorolojik problemler, elektromıknatıs, ısı ve mekanik sahaların hesaplanması, uydu bilgisinin analizi, nükleer santralarda oluşan işlemlerin araştırılması vb.) aittirler.

Böyle problemlerin çözümü karmaşık algoritmaların yapılmasını ve büyük hacimdeki bilginin hesaplanmasını ister. Çoğu zaman çağdaş bilgisayarların gücü, bu işlemleri uygun zaman süresinde yapmakta yetersizdir.

Bunlardan şu sonuç çıkarılabilir; gösterilen problemleri çözmek için bilgisayarların gücünü artırabilecek metotlar ve yollar gerekmektedir.

Geliştirilebilme; çoğu zaman bilgisayarın gücünü büyük hacimli hesaplamalara sahip problemleri çözmek için artırmak gerekir. Yani bir kaç bilgisayar (aynı veya değişik) kullanılarak gereken güce erişmek mümkündür. Bunun için bilgisayar sistemleri geliştirilebilme özelliğine sahip olmalıdırlar. Bu özellik, bilgisayar sisteminin yapısını değiştirmekle gerçekleştirilebilir.

Güvenilirlik, Arıza-Kaldırılabilirlik; modern bilgisayar sistemlerinde güvenilirlik ve arıza-kaldırılabilirlik talepleri yüksektir. Böyle talepler özellikle bilgisayar sistemleri önemli hedefleri kontrol ettikçe (insanlardan çok uzak mesafede bulunan, insanların yaşamı için çok tehlikeli olan, büyük facialara neden olabilecek hedefleri, vb.) ortaya çıkabilir.

Güvenilirlik ve Arıza-kaldırılabilirlik bilgisayar sisteminin ortaya çıkabilecek arızalara rağmen asıl kontrol fonksiyonlarını doğru yapabilme özelliğidir. Bilgisayar sistemlerinin yüksek güvenilirliğe ve arıza-kaldırılabilirliğe sahip olması için onların yapı elemanlarının karşılıklı yerine geçme özelliğine sahip olması şarttır. Aynı tip bilgisayarların yardımı ile böyle bir özelliğe erişmek mümkündür.

Yukarıdaki problemlerin hiçbiri en yüksek performansa ve imkana sahip tek bilgisayar tarafından çözülemez. Bu problemleri çözmek için tek bilgisayarın performansını yükseltmek yeterli değil, aynı zamanda bilgisayar sisteminin yapısını ve iş prensibini değiştirmek gerekir.

Bilgisayar sisteminin gerçek zaman problemini çözebilmesi için iki yol vardır:

1. Bilgisayar sisteminin elemanlarının çalışma frekansının yükseltilmesi;

2. Hesaplamaların paralel yapılması.

Birinci yol çok sınırlı imkanlara sahiptir. Halen bilgisayar elemanlarının çalışma frekansı MHz seviyelerinden fazla değildir. Bu durumda en perspektifli projeler bilgisayar elemanlarının çalışma frekansını 50-100 kat değiştirebilirler. Bu frekans bilgisayar hızının en yüksek değeridir ve bilgisayarın performansına, doğruluğuna gereken artışı yapamayacaktır.

İkinci yol tek bilgisayar (işlemci) sisteminden çok bilgisayarlı (çok işlemcili) sisteme geçmekle gerçekleştirilir. Bu en perspektifli yoldur, performansı ve doğruluğu yükseltmek için hiç bir prensipsel sınırlara sahip değildir.

Bilgisayarın büyük hacimde bilgiyi yeterli zaman süresinde işleyebilecek güce sahip olabilmesi için de tek yol, paralel bilgisayar sistemlerinin yapılmasıdır. Paralel (çok bilgisayarlı ve çok işlemcili) sistemler, çağdaş bilgisayarların zor yaptıkları veya yapamadıkları işlemleri çok kolaylıkla yapabilme imkanlarına sahiptirler.

Geliştirilebilme problemi de paralel sistemlerin yardımı ile çözülür. *Paralel sistemler, n* (*n=1,...,N*) sayıda bilgisayarlardan veya işlemcilerden oluşabilir. Eğer sistemin performansını artırmak veya azaltmak istesek, aynı tip bilgisayarı veya işlemciyi sisteme eklemek veya sistemden çıkartmak gerekir.

Güvenilirlik ve Arıza-kaldırılabilirlik problemi de kendi çözümünü paralel sistemlerde bulmuştur. Aynı tip bilgisayar ve işlemciden yapılmış paralel sistemlerde bu parametrelerin istenilen değerine erişmek mümkündür. Örneğin, aynı hesaplamalar paralel olarak ayrı bilgisayarlarda veya işlemcilerle yapılarak, çıkışta oylama prensibi ile düzgün değer elde edilebilir. Bu durumda ortaya çıkabilecek arızalar etkisiz hale getirilecektir. Öte yandan arızalı bilgisayar veya işlemci sistemdeki diğer düzgün çalışan bilgisayar veya işlemci ile değiştirilebilir bu da arıza-kaldırılabilirlik özelliğini oluşturur.

Sonuç olarak şunu söyleyebiliriz ki, gerçek zaman, bilgisayar gücü, geliştirilebilme, güvenilirlik ve arıza-kaldırılabilirlik problemlerinin çözülmesinin tek metodu tek bilgisayarlı (işlemcili) sistemden, çok bilgisayarlı (işlemcili) sisteme geçmektir. Yani paralel çalışan bilgisayarlardan ve işlemcilerden oluşan sistemler yapmaktır.

2.2 Flynn Sınıflandırma Şeması

Paralel bilgisayarları sınıflandırmak için çeşitli yollar vardır. Bunlardan biri 1966'dan beri sıkça kullanılan *Flynn's Taxonomy* (*Flynn Sınıflandırması*) dır. Bu şemada sınıflandırma belirtileri olarak aşağıdakiler seçilmiştir: (El-Rewini et al., 2005)

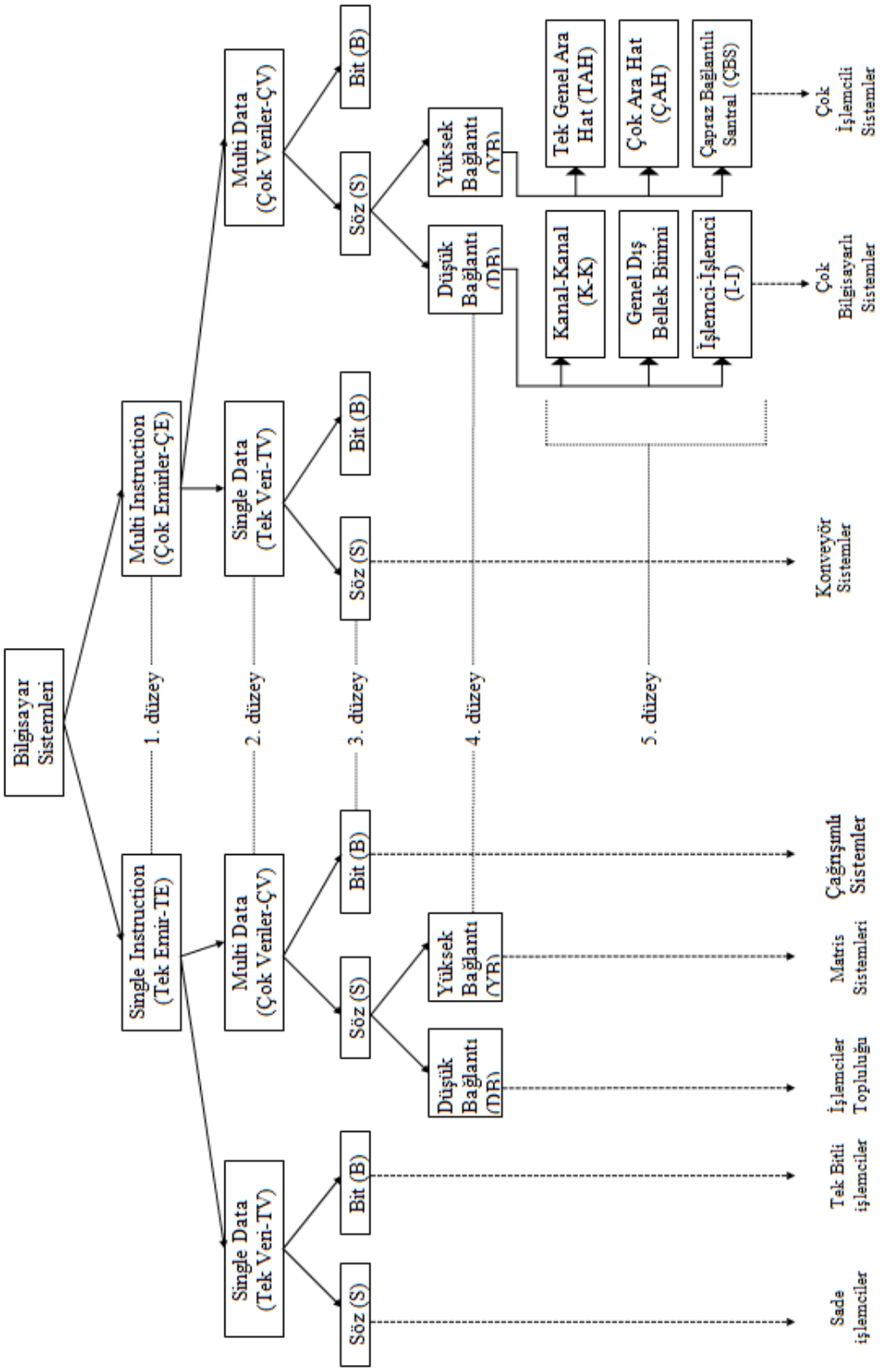
1. Emirler Akışı'nın tipi;
2. Veriler Akışı'nın tipi;
3. Verileri işleme metodu;
4. Fonksiyonel elemanların bağlantı düzeyi;
5. Elemanlar arasındaki bağlantıların tipi.

Bu belirtilere uygun yapılmış sınıflandırmanın temel şeması Şekil 2.3 'de verilmiştir. Şemada 6 düzey bulunmaktadır. 0. düzey mevcut bütün bilgisayar sistemlerini içerir. 0. düzeyden 1. düzeye geçiş *Emirler Akışı'nın tipi* ile belirtilir. 1. düzeyden 2. düzeye geçiş *Veriler Akışı'nın tipi* ile belirtilir vb. Herhangi bir somut bilgisayar sistemini isimlendirmek için aşağıdaki kısaltmalar kullanılabilir:

- SI - Single Instruction Stream (Tek Emir Akışı);
- MI - Multiple Instruction Stream (Çok Emir Akışı);
- SD - Single Data Stream (Tek Veri Akışı);
- MD - Multiple Data Stream (Çoklu Veri Akışı);
- W - Word (Verilerin İşlemcilerde Sözlerle İşlemi);
- B - Bit (Verilerin İşlemcilerde Bitlerle İşlemi);
- DB - Bilgisayar Sistemi Elemanlarının Düşük Bağlantı Düzeyi;
- YB - Bilgisayar Sistemi Elemanlarının Yüksek Bağlantı Düzeyi;

- KK - Kanal -Kanal Bağlantılı Sistemler;
- GDBB - Genel Dış Bellek Biriminin Yardımı ile Bağlantılı Sistemler;
- İ-İ - İşlemciler Arasında Direk Bağlantılı Sistemler;
- TAH - Tek Genel Ara Hatlı Sistemler;
- ÇAH - Çok Ara Hatlı Sistemler (Çok Girişli Bellek Kullanılır);
- ÇBS - Çapraz Bağlantı Santralli Sistemler.

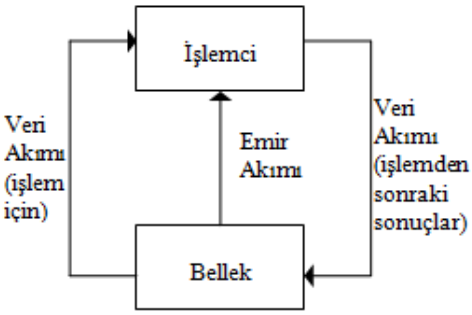
Bilgisayar sistemlerinin her sınıfının ismi sınıflama şemasının birinci üç düzeyindeki işaretlerden oluşur. Bilgisayar sistemleri 0. düzey düğümünden başlayarak ibre yönündeki somut sınıf düğümüne kadar bulunan işaretlerdir. Yazı rahatlığı için birinci üç düzey düğümlerinin işaretleri son iki düzey düğümlerinin işaretlerinden (/) ile ayrılırlar.



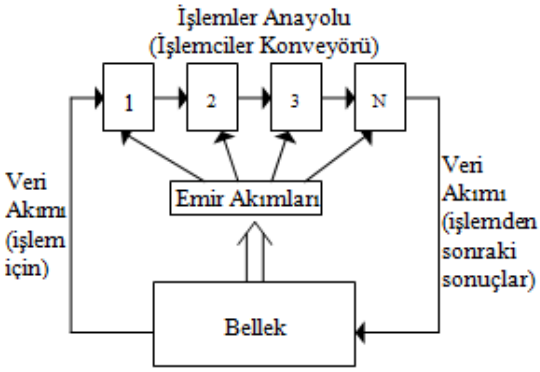
Şekil 2.3 Flynn sınıflandırma şeması (Samedov, R., 2002)

Flynn sınıflandırma şemasının birinci ve ikinci belirtilerine uygun olarak, Emir ve Veri Akışı'nın tiplerine (TE, ÇE, TV, ÇV) göre bilgisayar sistemleri dört esas sınıfa ayrılır: (Culler et al., 1997)

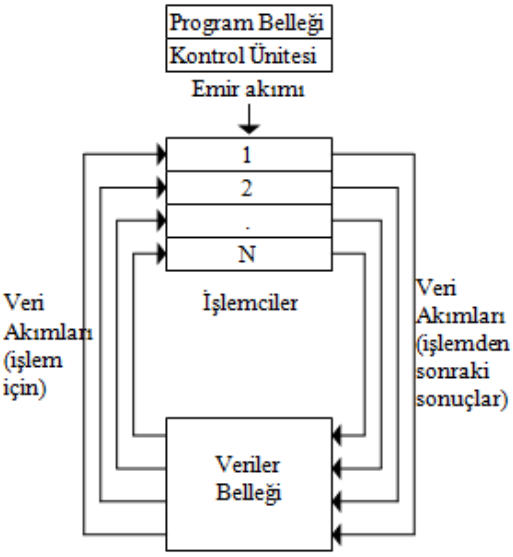
1) TETV - Tek Emir ve Tek Veri Akışlı sistemler (Şekil 2.4a)
2) ÇETV - Çok Emir ve Tek Veri Akışlı sistemler (Şekil 2.4b)
3) TEÇV - Tek Emir ve Çok Veri Akışlı sistemler (Şekil 2.4c)
4) ÇEÇV - Çok Emir ve Çok Veri Akışlı sistemler (Şekil 2.4d)



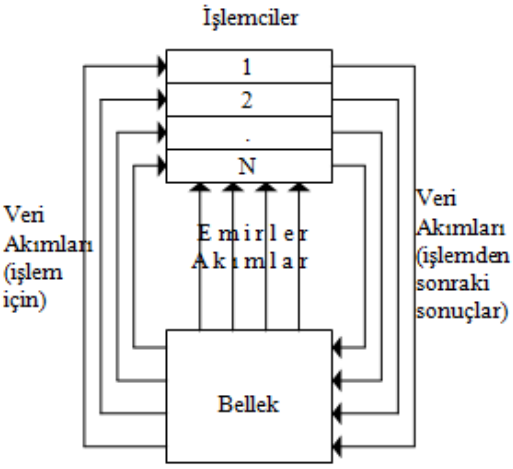
Şekil 2.4a TETV (Tek Emir Akışı ve Tek Veri Akışı) Sistemi



Şekil 2.4b ÇETV (Çok Emir Akışı ve Tek Veri Akışı) Sistemi



Şekil 2.4c TEÇV (Tek Emir Akışı ve Çok Veri Akışı) Sistemi



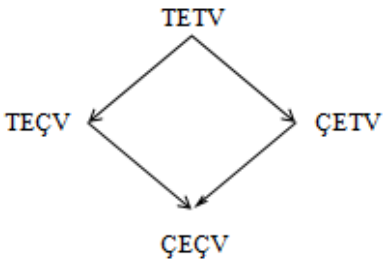
Şekil 2.4d ÇEÇV (Çok Emir Akışı ve Çok Veri Akışı) Sistemi

Şekil 2.4 Bilgisayar sistemlerinin tiplerinin yapı şemaları (Samedov, R., 2002)

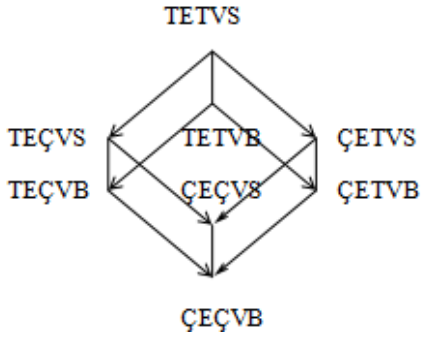
Bilgisayar sistemlerinin bu dört sınıfı Şekil 2.5'de görülmektedir. Şemanın en üst düğümü TETV sınıfına uygundur. Sol taraftaki düğüme geçerken veriler akışının sayısı artmaktadır. Bu da TEÇV sınıfını oluşturur. Üstteki düğümden sağ alttaki düğüme geçerken emirler akışı artmakta, bu da ÇETV sistemini oluşturmaktadır. Sağ ve sol taraftaki düğümlerden alttaki düğüme geçerken veri ve emirler akışının arttığı görülür. Bu ise ÇEÇV sistemini oluşturur.

Bilgisayar sistemlerinin dört sınıfta incelenmesi oldukça geneldir. Bir çok sistem farklı yapılara sahip olmalarına rağmen aynı sınıfa dahil edilebilir. Sınıflandırmanın daha da netleşmesi için sınıflar sözlerle yahut bitlerle işlem yapan gruplara ayrılır. (Herlihy, 2008)

Şekil 2.6'da bilgisayar sistemlerinin sekiz sınıfı verilmiştir. Bu şema kombine ve değişik yapılı sistem sınıflarını netleştirmek için imkan sağlar. Şekil 2.7'da, 8 sınıfın temsilcilerinin genel yapı şeması verilmiştir. Bu yapıda bellek program ve veri olmak üzere ikiye ayrılır.



Şekil 2.5 Bilgisayar Sisteminin Dört Sınıfının Şeması



Şekil 2.6 Bilgisayar Sisteminin Sekiz Sınıfının Şeması

Tek Emir ve Tek Veri Akışlı, Sözler ile işlem yapan sistemler - TETVS - *klasik paralel işlemcileri* temsil ederler.

Tek Emir ve Tek Veri Akışlı, Bitler ile işlem yapan sistemler - TETVB - *tek bitli ardışıl işlemcileri* temsil ederler.

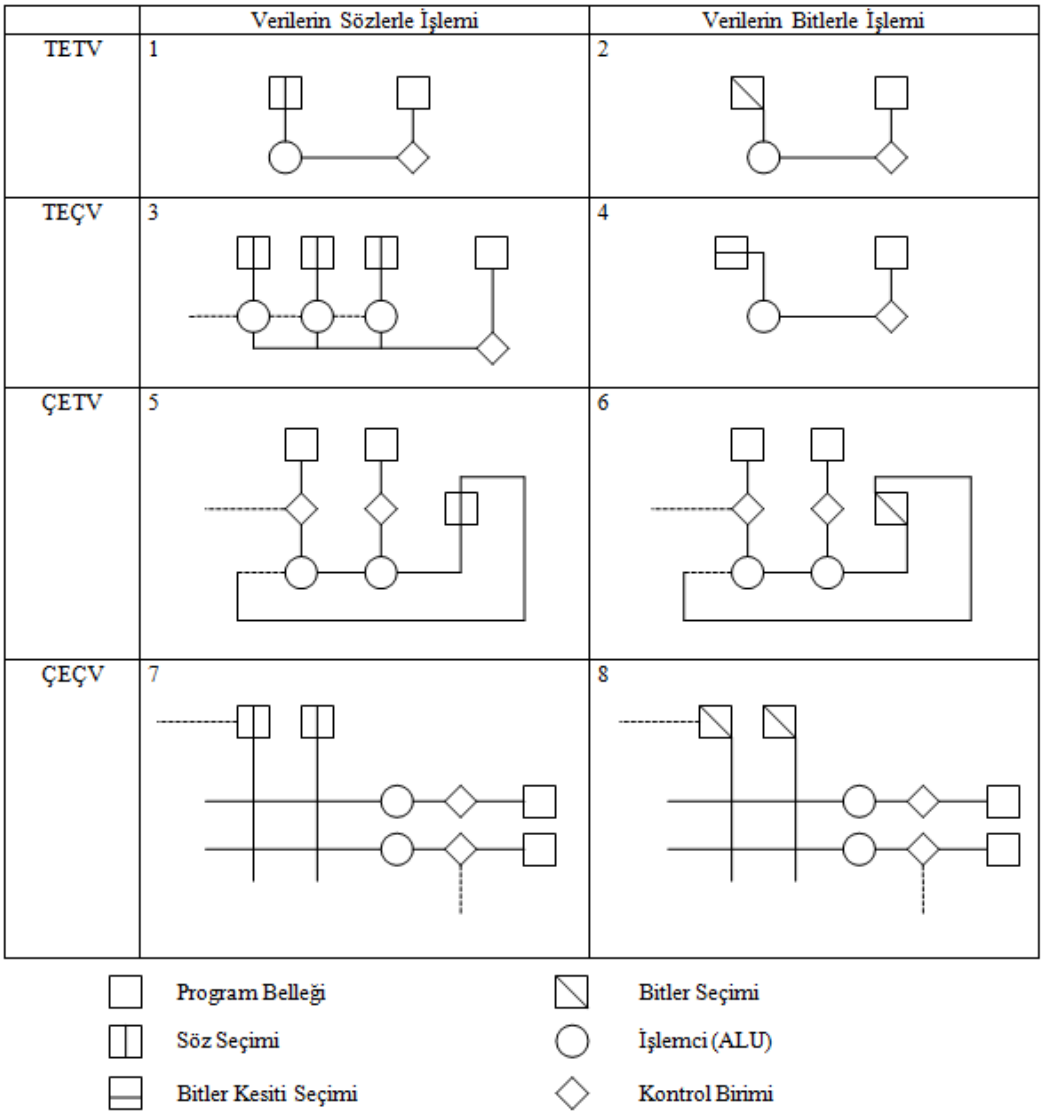
Tek Emir ve Çok Veri Akışlı, Sözler ile işlem yapan sistemler - TEÇVS - *matris sistemleri ve işlemciler topluluğunu* temsil ederler.

Tek Emir ve Çok Veri Akışlı, Bitler ile işlem yapan sistemler - TEÇVB - *associative (çağrışımlı) sistemleri* temsil ederler.

İşlemciler topluluğu paralel tipli sistemlerden oluşur. Bu sistemlerde işlemci üniteleri genel kontrol ve işlem akışları ile çalışırlar. Böyle sistemlerde işlemciler kendi aralarında bağlantıya sahip değildirler veya düşük bağlantıya sahiptirler.

Matris sistemlerinde işlemci üniteleri genel kontrol ve işlem akışları ile çalışırlar, lakin işlemciler yüksek bağlantı düzeyine sahiptirler.

TEÇVS tipli sistemler için düşük bağlantının anlamı: İşlemciler bireysel belleklere sahip olup kontrol ünitesine bağlanmış fakat kendi aralarında bağlantıya sahip değillerdir. İşlemcilerin bireysel belleklere sahip olduğu ve komşu işlemciler arasında bağlantıların bulunduğu sistemlere, *yüksek bağlantılı sistemler* denir.



Şekil 2.7 Sekizli bilgisayar sistemlerinin genel yapı şemaları (Samedov, R., 2002)

Çağrışımlı sistemler iki gruba ayrılırlar: *çağrışımlı bellekli sistemler* ve *çağrışımlı işlemcili sistemler* (çağrışımlı matris sistemleri).

İlk sistemde veriye ulaşma associative veriye göre olur (adrese göre değil). İkincisi ise veri bitlerinin kesitleri ile işlem yapar. Veri bit kesitleri veri sözlerindeki aynı sütunda bulunan bitlerden oluşurlar.

Çok Emir ve Tek Veri Akışlı, Sözler ile işlem yapan sistemler - ÇETVS - *konveyör sistemlerini* temsil ederler. Bu sistemde aynı anda bir kaç emir yapılır. Bu işlem veri akışının sözlerinin seri halinde birkaç özel işlemciden geçmesi ile olur. Bu işlem esnasında emir uygun etaplara ayrılır, etaplardan herbiri özel işlemcilerin birinde yapılır.

Çok Emir ve Tek Veri Akışlı, Bitlerle işlem yapan sistemler - ÇETVB - tek bit işlemcili konveyör sistemlerini temsil ederler.

Çok Emir ve Çok Veri Akışlı, Sözler ile işlem yapan sistemler - ÇEÇVS - çok işlemcili ve çok bilgisayarlı sistemleri temsil ederler.

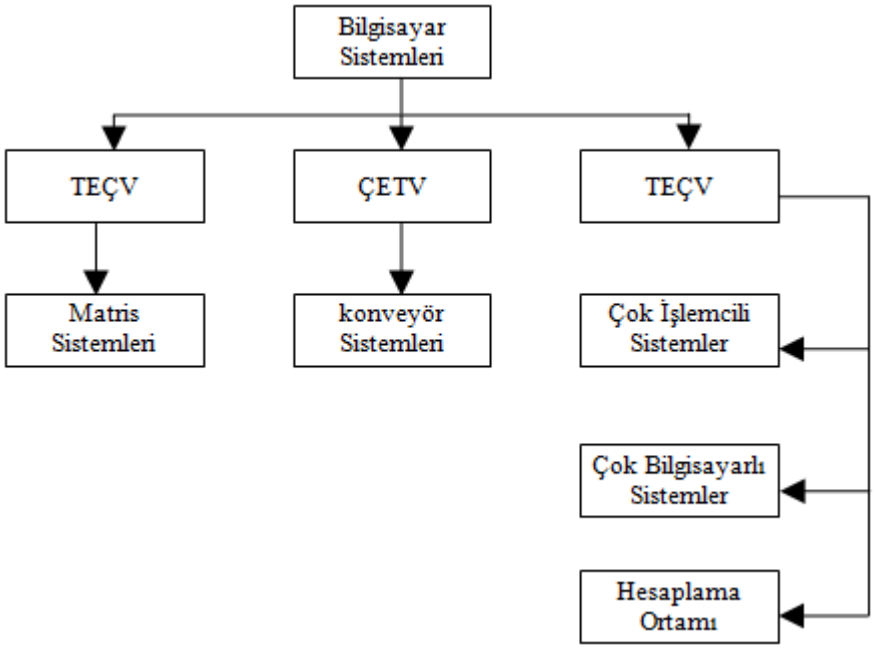
Çok Emir ve Çok Veri Akışlı, Bitler ile işlem yapan sistemler - ÇEÇVB - düşük bağlantıya sahip sistemleri temsil ederler.

Çok bilgisayarlı sistemler düşük, çok işlemcili sistemler ise yüksek bağlantıya sahiptirler. Yüksek bağlantı genel belleğin yardımı ile yapılır.

ÇEÇV sistemler iki gruba ayrılır: Hesaplama Ağları ve Hesaplama Ortamı. *Hesaplama Ağları* uzak mesafelere yerleştirilmiş merkezlerden (düğümlerden) ve sistem bağlantılarından oluşur. Merkezlerde *n* çeşit bilgisayar sistemi bulunabilir. Bağlantılar, ağları her hangi bir somut yapıya göre bağlar. Hesaplama merkezleri bağımsız olarak çalışırlar ve kendi aralarında sistem bağlantılarının yardımı ile bilgi değişimi yaparlar.

ÇEÇVB sınıfına dahil olan bilgisayar sistemlerinin gelişmesinin en perspektif yönü *hesaplama ortamıdır*. Bu paralel çalışan toplu bilgi işlemi yapan üniteler topluluğudur. Hesaplama ortamı, basit ve aynı mantıklı otomatik birimler topluluğunu oluşturur. Otomatik birimler kendi aralarında eşit bağlantıya sahiptirler ve her hangi bir fonksiyonun yapılması için programla düzenlenirler. Değişken yapılı sistemlerde, işlem zamanı yapı tipini değiştirme imkanına sahiptir ve bir sınıftan diğerine geçiş yapılabilir.

Şekil 2.8’ de günümüzde çok kullanılan bilgisayar sistemlerinin sınıfları görülmektedir.



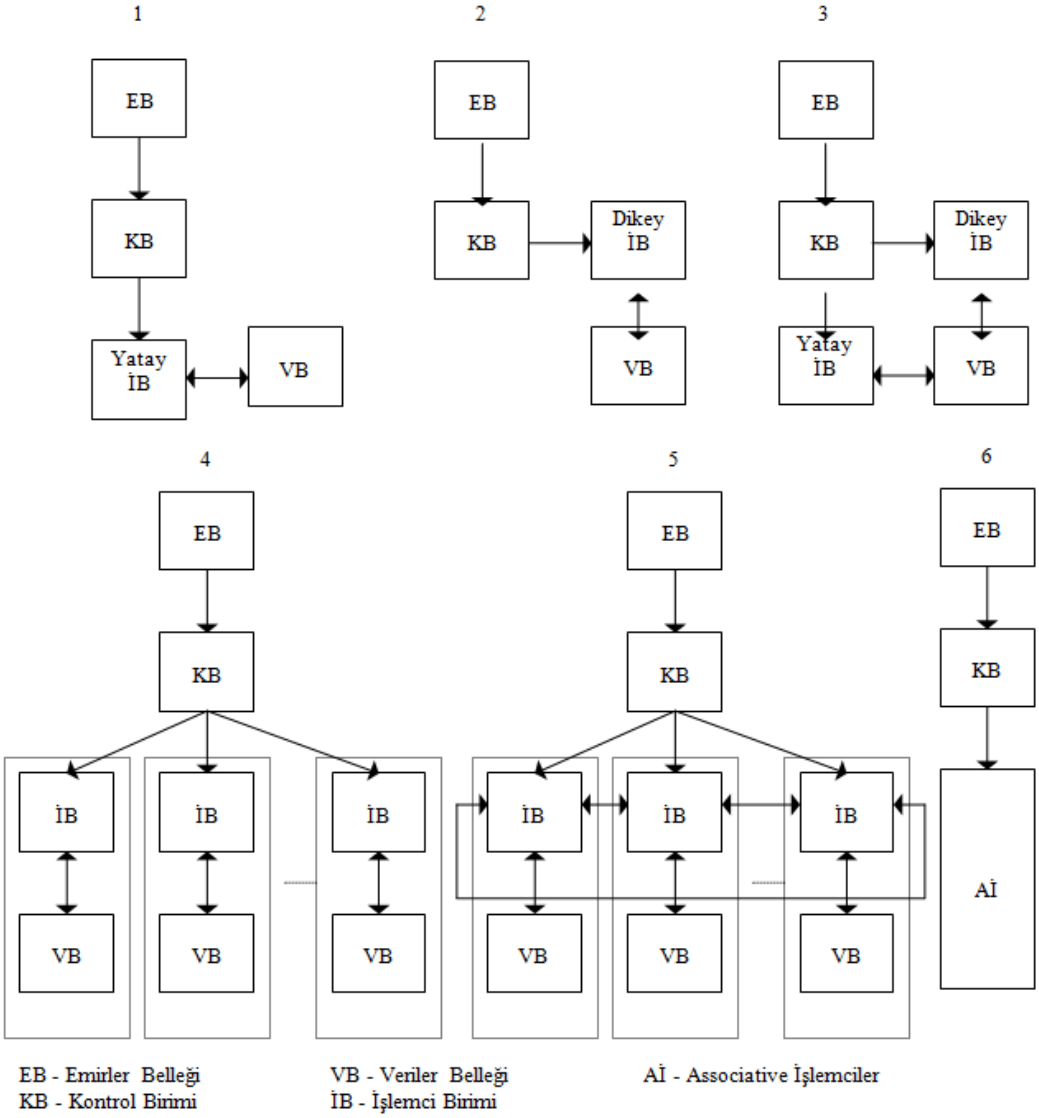
Şekil 2.8 Halen sık sık kullanılan sistemler

2.3 Shor Sınıflandırma Şeması

Shor sınıflandırılmasında iki belirti kullanılır: (Parhami, 2002)

1. Bilgisayar sistemleri temel birimlerinin bağlantı metotları belirtisi
2. Verileri işleme metotları belirtisi

Bu sınıflandırma şeması bütün bilgisayar sistemlerini 6 sınıfa ayırır (Şekil 2.9):



Şekil 2.9 Shor sınıflandırma şeması (Samedov, R., 2002)

1. Klasik von Neumann yapılı bilgisayarlardır: Tek Kontrol Birimi (KB), Veri İşleme Birimi (İB), Emirler Belleği (EB) ve Veriler Belleği (VB)'den oluşurlar. Bu sınıf bilgisayarlarda veriler üzerinde sözler ile (yatay) işlem yapan paralel İB'ler kullanılır ve işlemci birimi bir veya birkaç ALU tipli fonksiyonel birim içerebilir.

2. Bu sınıfa giren bilgisayarların yapısı ile birinci sınıfa giren bilgisayarların yapısı arasındaki fark verileri işleme metodudur. Bu sınıf bilgisayarlarda veriler üzerinde bitlerle (dikey) işlem yapılır. Veri işleme biriminde aynı zamanda VB'de bulunan bütün sözlerin aynı isimli bitleri üzerinde işlem yapılır.

3. Birinci ve ikinci sınıfın birleşmesi ile oluşur. Bu bilgisayarlarda iki tip veri işleme birimi bulunur; yatay ve dikey İB. Aynı zamanda VB'den veriler iki metotla çıkarılır; sözlerle ve bitlerle.

4. Bu sınıf çok sayıda işlemci elemanlarının kullanılmasından oluşur ve her işlemci elemanı çift birim şeklinde yapılır; İB ve VB. Böyle işlemciler topluluğunu tek KB yönetir. Bütün işlemci elemanları birbirinden bağımsızdırlar ve senkron olarak çalışırlar. İşlemci elemanların arasında bağlantının olmaması bu sınıf bilgisayarların kullanımını sınırlar. Bunun yanında sistemdeki işlemci elemanların sayısını artırmak çok kolaydır.

5. Bu sınıf bilgisayarların yapısında, işlemciler arasında bağlantı kurulması ile oluşur. Böyle bir sistemde her hangi bir işlemci birimi hem kendi VB'nin sözlerini hem de başka VB'lerinin sözlerini kullanabilir.

6. Bu sınıf bilgisayarların yapısı İB mantığının sistemin bellek birimi boyunca dağıtımı prensibine dayanır. Bu sınıfa örnek olarak basit associative bellekli bilgisayarları ve karmaşık associative işlemcileri göstermek mümkündür.

Her sınıflandırma şeması belli belirtilere dayanarak yapıldığından çeşitli şemalara ait sistemler arasında tam uygunluk bulmak mümkün değildir. Bazı durumlarda çeşitli sınıflandırma şemalarına ait sınıflar arasında özel bağlantılar ortaya çıkabilir. Örneğin; Shor sınıflandırma şemasının 1. sınıfı Flynn sınıflandırma şemasının TETVS sınıfına uygun gelir, 2. sınıfı -TETVB ve 3, 4, 5 sınıfları - TEÇV. Bu çok sayıdaki mevcut sınıflandırma şemasından halen en yaygın olarak kullanılan Flynn şemasıdır.

2.4 Paralel Algoritmalar

Paralel algoritmaların analizi yapılırken iki kavram göz önünde bulundurulur: hız ve maliyet. (Carriero, 1992) Örneğin, sıralama algoritması için *O(nlogn)* değerinin optimum sonuç olduğunu biliyoruz. Eğer elimizde *O(n)* gibi bir paralel algoritma mevcutsa algoritmamız *O(logn)* kadar hızlanmış oluyor. Ama bunu elde edebilmek için *n* adet işlemci kullanmamız gerekiyor, bu da bize $O(n^2)$ bellek kapasitesi gerektirdiğini gösteriyor. Oysa ki *O(nlogn)* sıralama algoritması için *O(n)* bellek kapasitesi yeterli oluyor. (McConnell, 2001)

İlişkili diğer bir sorun ise problemin ölçülebilirliği. Eğer paralel sıralama algoritmamız giriş eleman sayısı kadar işlemci gerektiriyorsa bu algoritmanın kullanışsız olduğu boyutta eleman içeren problemler bulabiliriz. Oysa ki ardışıl algoritma giriş eleman sayısına bağlı olmadan sonuç bulmaktadır. Giriş eleman sayısına bağımlı olmayan, işlemci sayısını arttırmadan problemleri çözebilen paralel algoritmalar geliştirmek gerekiyor. (Levitin, 2006)

2.4.1 Arama algoritması

Bilindik en iyi ardışıl arama algoritması *O(logn)* karmaşıklığa sahiptir. *n* işlemcimiz olduğu takdirde paralel arama algoritmasının çalışma zamanı karmaşıklığı *O(1)*, maliyeti *O(n)* olacaktır. Ölçülebilir bir algoritma düşünecek olursak, *p* işlemcimiz ve her biri *n/p* adet eleman içeren listeler mevcut iken zaman karmaşıklığı *O(log(n/p))*, maliyet ise *O(p.log(n/p))* olacaktır. *p=logn* seçildiğinde zaman karmaşıklığı *O(logn)*, maliyet $O(\log^2 n)$ dir. Ardışıl arama algoritması ile aynı zamanda çözmesine rağmen daha fazla maliyet gerektirmektedir. Fakat çok fazla bir fark yoktur. Bu da bize maliyetteki bu değişim önemsenmediği takdirde arama algoritmaların paralelleştirilebildiğini göstermektedir. (Lin et al., 2008)

2.4.2 Sıralama Algoritması

Girişte de bahsedildiği gibi *n* adet işlemcimiz olduğu takdirde zaman karmaşıklığı *O(n)* dir, fakat maliyetimiz $O(n^2)$ olur.

n/lgn işlemci kullanarak iki listeyi *O(n)* karmaşıklıkta birleştirebilen paralel merge algoritmalar mevcuttur. Eğer sıralacak listeyi *n/lgn* parçaya bölersek ve her

parçada *QuickSort* gibi hızlı ardışıl bir algoritma ile sıralarsak bu paralel merge algoritmalarını kullanabiliriz. Böylece oluşacak maliyeti azaltmış oluruz.

2.4.3 Matris çarpımı

Matris çarpımı, graflar bitişiklik veya ağırlık matrisleri ile temsil edildiklerinde graf problemlerin çözümünde oldukça kullanışlıdır.

n x *n* boyutlu *A* ve *B* gibi iki matris verildiğinde ardışıl çarpım $O(n^3)$ karmaşıklığa sahip olacaktır. n^2 adet işlemcimiz olduğu zaman karmaşıklık *O(n)* dir. Bu da ardışıl algoritma ile aynı sonucu elde ettiğimizi gösterir. Büyük sayılarda bu sistemin aslında ne kadar avantajlı olduğu daha iyi anlaşılmaktadır. Örneğin, 512x512 boyutlu bir matriste 5x5 lik birimlerin çarpımını yapalım. Ardışıl algoritma ile 32,258,000 adet çarpma işlemini 1 işlemci ile 32,258,000 turda yapmamız gerekmektedir. Oysa 5x5 grid işlemcide (25 işlemci) 3,612,896 turda aynı sayıda çarpma işlemini gerçekleştirebilir. Nerdeyse %90 zamandan kazanç sağlanmış oluyor.

2.5 Paralel Programlama Modelleri

Bir paralel programlama modeli, paralel algoritmaları açıklayan bir yazılım teknolojileri kümesidir. Bu model, uygulamalar, diller, derleyiciler, kütüphaneler, iletişim sistemleri ve paralel giriş/çıkış alanlarını kapsar. Programcılar, kendileri ve uygulamaları için uygun bir model veya karma bir model seçip, uygulamalarını geliştirirler. (Stein, 2011)

Paralel modeller çok farklı şekillerde uyarlanırlar: klasik sıralı dillerden çağrılan kütüphaneler şeklinde, dil uzantıları şeklinde ya da tamamen yeni işleme modelleriyle. Bu modeller kabaca ikiye ayrılırlar: paylaşımlı hafıza sistemleri ve dağıtık hafıza sistemleri. Günümüzde bu iki sistem arasındaki çizgi oldukça bulanıklaşmıştır. (NZMATH development group, 2011)

GMP, NZMATH, FLINT kullanılan kütüphanelere örnek verilebilir. MPI, OpenMP, LAM/MPI, CUDA, PVM, Global Arrays, Co-Array Fortran, UPC, HPF, SHMEM, Occam, Linda, Cilk ise sık kullanılan paralel programlama modelleridir. Aşağıdaki bölümlerde kullandığım GMP kütüphanesi ve MPI modelinden bahsedilmiştir.

3. GMP KÜTÜPHANESİ

Richard Sallman tarafından geliştirilen GNU işletim sisteminin (Unix tabanlı) bir kütüphanesidir. (The GNU Multiple Precision Arithmetic Library) İlk olarak 1991 yılında geliştirilmektedir. Yılda en az bir kez güncelleştirilerek yayınlanmaya devam ediliyor. Bazı uygulamalar birkaç yüz bit duyarlılık kullanırken, bazıları bin hatta milyon duyarlılık kullanır. GMP bu iki farklı uygulama için en iyi performans sağlamak için geliştirilmiştir. Bünyesinde çok fazla sayıda fonksiyon bulunmaktadır. (Granlund, 2011)

3.1 Temel Kavramlar

3.1.1 Başlık ve kütüphaneler

GMP kullanmak için gerekli tüm bildirimler “gmp.h” dosyasında toparlanmıştır. C ve C++ derleyicilerinde çalışabilmektedir.

#include <gmp.h>

GMP kullanılan bütün programlar “libgmp” kütüphanesiyle ilişkilendirilmelidir. Unix sistemde bu işlem “-lgmp” parametresi ile yapılabilir, örneğin:

gcc program.c –lgmp

GMP C++ programları farklı bir kütüphane, “libgmpxx”, ile ilişkilendirilmiştir, örneğin:

g++ program.cc –lgmpxx –lgmp

3.1.2 Veri tipleri

C dilinde kullanılan *integer* için *mpz_t* kullanılır. Tanımlama için birkaç örnek:

```
mpz_t veri1;

struct foo { mpz_t x, y; };
```

```
mpz_t veri2[40];
```

Reel sayılar için *mpf_t* ifadesi kullanılır.

```
mpf_t veri3;
```

3.2 Tamsayı Fonksiyonlar

Tamsayılar için kullanılan fonksiyonlar *“mpz_”* ön takısı ile başlar.

3.2.1 Tanımlama fonksiyonları

Tamsayılarda aritmetiksel işlem yapan fonksiyonlar tamsayı değerlerinin ilk değerlerini aldığını varsayarlar. Bunun için tanımladığımız değişkenlere ilk atamalarını yapmamız gerekir. Bu işlem için “mpz_init” fonksiyonu çağrılır.

```
{
   mpz_t integ;
   mpz_init(integ);
   …
   mpz_add(integ, …);
   …
   mpz_sub(integ, …);
   …
}
```

Görüldüğü gibi değişkene ilk atamasını yaptığımız andan itibaren istediğimiz kadar kullanabiliriz.

void mpz_init(mpz_t x) [fonksiyon]

Değişkene ilk ataması 0 olarak yapılır.

void mpz_clear(mpz_t x) [fonksiyon]

Değişken için bellekten ayrılan alan boşaltılır. Değişkenin kullanımı bittiğinde bu fonksiyon çağrılmalı. Çünkü belleğin fazla kullanılması programın yavaşlamasına sebep olacaktır.

3.2.2 Değer atama fonksiyonları

İlk değeri atanmış değişkenler için kullanılan fonksiyonlardır.

void mpz_set(mpz_t ydeg, mpz_t deg) [fonksiyon]

void mpz_set_ui(mpz_t ydeg, unsigned long int deg) [fonksiyon]

void mpz_set_si(mpz_t ydeg, signed long int deg) [fonksiyon]

void mpz_set_d(mpz_t ydeg, double deg) [fonksiyon]

void mpz_set_f(mpz_t ydeg, mpf_t deg) [fonksiyon]

deg değişkenindeki değeri *ydeg* değişkenine aktarılır. mpz_set_d ve mpz_set_f fonksiyonları *deg* değişkeninin yuvarlanmış halini aktarır.

*int mpz_set_str(mpz_t ydeg, char *str, int taban)* [fonksiyon]

str değişkeninde bulunan string ifadeyi *taban* değişkeni ile girilen sayı tabanda *ydeg* değişkenine aktarmayı sağlar.

void mpz_swap(mpz_t ydeg, mpz_t deg) [fonksiyon]

ydeg ve *deg* değişkenlerindeki değerleri etkili şekilde değiştirir.

3.2.3 Tanımlama ve değer atama fonksiyonlarının birleşimi

Kolaylık olması için GMP kütüphanesi tanımlama-değer atama fonksiyonlarının paralel çalışmasını sağlamaktadır. Bir değişkenin önce ilk

ataması yapılır daha sonra verilen değer atanır. Bu fonksiyonlar “mpz_init_set” olarak adlandırılır. Kullanımı için bir örnek vermek gerekirse:

```
{
   mpz _t pie;
   mpz_init_set_str(pie,"3141592653589793238462643383279502 8", 10);
   …
   mpz_sub(pie, …);
   …
   mpz_clear(pie);
}
```

3.2.4 Aritmetiksel işlem fonksiyonları

void mpz_add(mpz_t rop, mpz_t op1, mpz_t op2) [fonksiyon]
void mpz_add_ui(mpz_t rop, mpz_t op1, unsigned long int op2)

rop = op1 + op2 işlemini gerçekleştirir.

void mpz_sub(mpz_t rop, mpz_t op1, mpz_t op2) [fonksiyon]
void mpz_sub_ui(mpz_t rop, mpz_t op1, unsigned long int op2)
void mpz_ui_sub(mpz_t rop, unsigned long int op1, mpz_t op2)

rop = op1 - op2 işlemini gerçekleştirir.

void mpz_mul(mpz_t rop, mpz_t op1, mpz_t op2) [fonksiyon]
void mpz_mul_si(mpz_t rop, mpz_t op1, long int op2)
void mpz_mul_ui(mpz_t rop, mpz_t op1, unsigned long int op2)

rop = op1 * op2 işlemini gerçekleştirir.

void mpz_addmul(mpz_t rop, mpz_t op1, mpz_t op2) [fonksiyon]

rop = rop + op1 * op2 işlemini gerçekleştirir.

void mpz_submul(mpz_t rop, mpz_t op1, mpz_t op2) [fonksiyon]

rop = rop - op1 * op2 işlemini gerçekleştirir.

void mpz_neg(mpz_t rop, mpz_t op) [fonksiyon]

rop değişkenine op değişkeninin negatif değerini(-op) atar

void mpz_abs(mpz_t rop, mpz_t op) [fonksiyon]

op değişkeninin mutlak değerini rop değişkenine atar.

3.2.5 Sayı Teorisi Fonksiyonları

int mpz_probab_prime_p(mpz_t deg, int reps) [fonksiyon]

deg ifadesinin asal sayı olup olmadığının kontrolünü yapar. 2 değeri döndüğü takdirde *deg* kesinlikle asal sayıdır, 1 değeri döndüğün takdirde asal sayı olma ihtimali var, 0 değeri döndüğünde ise kesinlikle bileşik sayı demektir. Miller-Rabin asallık testini kullanır. *reps* değişkeni ile bu testin kaç kez yapılacağı belirlenir.

void mpz_nextprime(mpz_t ydeg, mpz_t deg) [fonksiyon]

deg değişkeninden sonraki ilk asal sayıyı *ydeg* değişkenine aktarır.

void mpz_gcd(mpz_t ydeg, mpz_t deg1, mpz_t deg2) [fonksiyon]

deg1 ve *deg2* ifadelerinin en büyük ortak bölenini *ydeg* değişkenine aktarır.

void mpz_fac_ui(mpz_t rop, unsigned long int op) [fonksiyon]

deg ifadesinin faktöriyelini *ydeg* ifadesine aktarır.

3.2.6 Karşılaştırma Fonksiyonları

int mpz_cmp(mpz_t deg1, mpz_t deg2) [fonksiyon]

deg1 ve *deg2* değişkenlerini kıyaslar. *deg1>deg2* ise pozitif, *deg1<deg2* ise negatif, *deg1=deg2* ise sıfır değeri döndürür.

int mpz_cmpabs(mpz_t deg1, mpz_t deg2) [fonksiyon]

deg1 ve *deg2* değişkenlerini mutlak değerlerini kıyaslar.

int mpz_sgn(mpz_t deg) [fonksiyon]

deg ifadesi pozitif ise +1, negatif ise -1, sıfır ise 0 değeri döndürür.

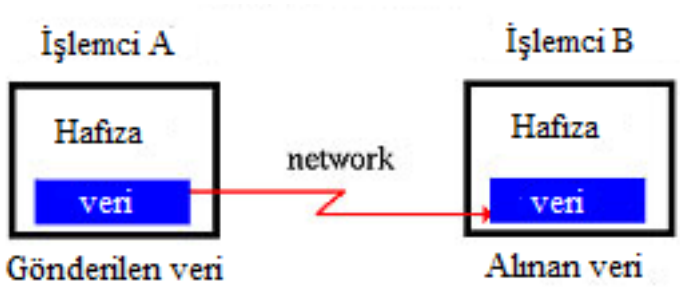
4. MPI KÜTÜPHANESİ

MPI (Message Passing Interface) dağıtık bellekli, mesaj geçmeli paralel hesaplamalarda standart oluşturabilecek mesaj geçme kütüphaneleri için bir tanımlamadır. (Trustees of Indiana University, 2011) Kütüphanenin hedefi mesaj geçmeli programlar yazmak için geniş kullanıma sahip bir standart sağlamaktır. Arayüz; mesaj geçme için pratik, taşınabilir, etkin ve esnek bir standart kurmaya çalışır. Satıcılar, uygulamacılar, ve kullanıcılardan oluşan bir komite tarafından geliştirilmektedir. (Pacheco, 1996)

4.1 Mesaj Geçme Modeli

Her işlemci sadece kendi CPU'su ile direk olarak erişilebileceği yerel belleğe sahiptir. Bir işlemciden diğerine veri iletimi ağ üzerinden gerçekleştirilir. Mesaj Geçme, verinin bir işlemcinin belleğinden diğer bir işlemcinin belleğine kopyalandığı yöntemdir. Dağıtık bellekli sistemlerde veri genellikle bir işlemciden diğerine ağ üzerinden bilgi paketleri olarak yollanır. Bir mesaj bir veya daha çok bellekten oluşabilir ve çoğunlukla yönlendirme ya da başka kontrol bilgileri içerir. (Snir et al., 1998)

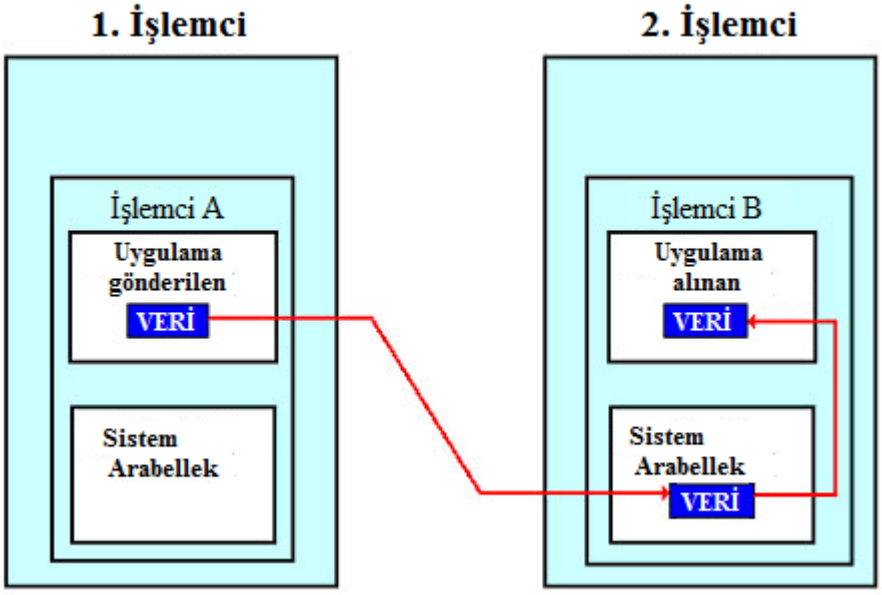
Mesaj geçme verinin bir süreçten diğer bir sürece transfer olmasını icap ettirir. Yollayan ve alan süreçlerin birlikte çalışmasını gerektirir. "Send" işlemleri genellikle yollayan sürecin verinin yeri, boyu, tipi ve hedefi bilgilerini belirtmesini gerektirir. "Receive" işlemleri ilişkili "Send" işlemi ile eşleşmelidir.



Şekil 4.1 Temel mesaj iletimi

Senkron send işlemi alıcı process'in mesajı güvenli bir şekilde aldığını belirten onayından sonra sonlanacaktır. Asenkron send işlemleri alıcı süreç mesajı almadan tamamlanabilir. Uygulama Tamponu, yollanacak veya alınacak veriyi tutan adres uzayıdır. Sistem Tamponu ise mesajları tutmak için kullanılan sistem uzayıdır. Send/Receive işlemine bağlı olarak uygulama tamponundaki veri, sistem

tampon uzayından kopyalanmak zorunda kalabilir. Haberleşmenin asenkron olmasına izin verir.

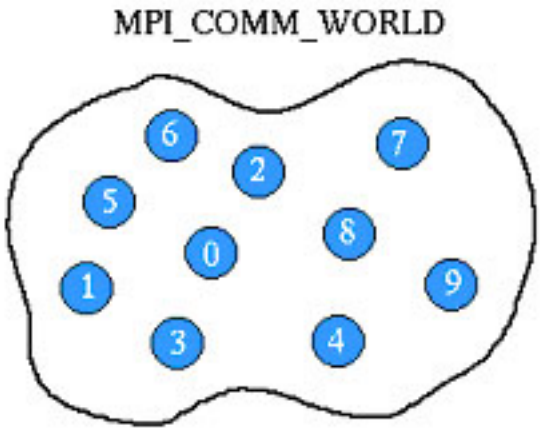


Şekil 4.2 Arabellek yardımıyla iletilen bir mesajın izlediği yol

Eğer çağrının tamamlanması belirli bir olaya bağlı ise haberleşme rutini bloklanır. Göndermeler için, veri başarı ile yollanmalı veya sistem tampon uzayına güvenli bir şekilde kopyalanmalıdır. Alıcı için veri, alıcı tamponunda depolanmalıdır. Böylece veri kullanmak için hazır olacaktır. Eğer çağırma herhangi bir haberleşme olayının tamamlanmasını beklemeden dönüyorsa haberleşme rutini bloklanmayandır. Bloklanmayan haberleşmeler performansı arttırmak amacıyla hesaplamayı haberleşme ile üst üste getirmek için kullanılırlar. (Kanmaz, 2003)

MPI hangi süreç kümelerinin hangileri ile haberleşeceğini belirtmek için, communicator ve group adında nesneler kullanırlar. Çoğu MPI rutinleri communicator'ı argüman olarak belirtmenizi gerektirir.

MPI_COMM_WORLD, bütün MPI süreçlerini içeren önceden tanımlı ileticidir.



Şekil 4.3 Önceden tanımlı iletici dünyası

İleticinin içinde, her süreç başlatımlandığında sistem tarafından atanan kendi eşsiz integer belirtecine sahiptir. Rank bazen “process ID” olarak da adlandırılabilirler. Ranklar sıralıdır ve 0’dan başlarlar.

Programcı tarafından mesajın kaynağını ve hedefini belirtmek için kullanılırlar. Bir çok zaman program icrasını kontrol için uygulama tarafından şartlı olarak kullanılırlar. (if rank==0 do this / if rank==1 do that)

4.2 MPI’a Giriş

MPI kütüphane fonksiyonlarını kullanmak için başlık dosyasını dahil etmeliyiz.

#include "mpi.h"

MPI Çağrılarının formatı şu şekildedir:

rc = MPI_Xxxxx(parameter, ...)

4.2.1 Yaygın kullanılan fonksiyonlar

MPI_Init(&argc, &argv)

MPI_Init fonksiyonunu bütün kodlarınıza eklemeniz gerekmektedir. MPI_Init fonksiyonuna komut satırından da parametre verebilirsiniz ama mecbur değilsiniz. MPI_Init size bir değer dönmektedir bu “MPI_SUCCESS” olursa kod

geri kalan MPI_X fonksiyonlarını kullanabilecektir, eğer bu değer dönmezse MPI_X fonksiyonlarını kullanamazsınız.

MPI_Comm_size(comm, &size)

Bu fonksiyonumuz süreç sayısını bize vermektedir. Genellikle “comm” kısmına MPI_COMM_WORLD üst fonksiyonu yazılmaktadır.

MPI_Comm_rank(comm, &rank)

Bu MPI fonksiyonu çağıran sürecin sırasını vermektedir. Başlangıçta bütün süreçler 0 veya -1 değerlerini almaktadır. Daha sonra MPI tarafından sıraya sokulunca bütün süreçler numaralandırılmaktadır. Aşağıda ki örneğimizde göreceğimiz gibi makineler 0 dan başlayarak rank(sıra) alacaklardır.

MPI_Abort (comm, errorcode)

Adından da anlaşıldığı gibi bütün süreçleri durdurmaktadır. Istenmeyen durumlarda başvurulabilecek bir fonksiyondur.

MPI_Get_processor_name(&name, &resultlength)

İşlemcinin adını ve adın uzunluğunu dönmektedir.

MPI_Initialized(&flag)

MPI_Init fonksiyonun çağırılıp çağırılmadığını kontrol etmektedir ve çağırıldıysa “1” dönmektedir. MPI_Init her süreç tarafından sadece bir kere çağırılması gerektiği için MPI_Initialized fonksiyonu olası çakışmaları önlemektedir.

MPI_Wtime( )

Double deger olarak paralel kodun çalışmaya başlamasından sonra geçen zamanı vermektedir.

MPI_Finalize( )

Bütün işlemlerimizden sonra MPI_Finalize diyerek işlemlerimizi sonlandırıyoruz.

MPI_Send(&buffer, count, datatype, dest, tag, comm)

MPI_Recv(&buffer, count, datatype, source, tag, comm, status)

Fonksiyonlarımız burada değişik parametreler almaktadır bunlardan *buffer* yollanacak veya alınacak verinin adresini göstermektedir.

Count parametresi yollanacak verilerin sayısını belirtmektedir.

Datatype parametresi gidecek verinin türünü belirtmektedir (*int, float, double, char, ...*).

Dest parametresi verinin hangi makineye gideceğini belirtmektedir. *Source* parametresi verinin hangi makineden alacağını belirtmektedir.

Tag parametresi ise 0-32767 arası sayısal bir değer alır ve kullanıcı tarafından yanlış makinelerden mesajlar gelmesini önlemek için verilebilecek bir tür güvenlik kodudur. SEND ve RECV fonksiyonlarında tag' lerin eşit olması gerekmektedir.

Comm değişkeni olarak genelde MPI_COMM_WORLD kullanılmaktadır. Grup haberleşme fonksiyonlarında değişik "comm" değerleri kullanılmaktadır. Grup haberleşme fonksiyonlarında ise comm parametresi birden fazla değer alabilir.

En son parametre değerimiz ise RECV fonksiyonundaki *status* değeridir. *Status* mesajın geldiği kaynağı bize bildirmektedir.

SEND/RECV fonksiyonlarımızın dışında işimize yarayabilecek başka fonksiyonlarda vardır bunları kısaca inceleyecek olursak;

MPI_Isend(&buffer, count, datatype, dest, tag, comm)

MPI_Irecv(&buffer, count, datatype, source, tag, comm, status)

Yukarıda gördüğünüz komutların Send ve Recv fonksiyonlarından tek farkı "I" (Immediate) yani beklemeksizin veri alışverişlerin olmasıdır. Sizin de tahmin edebileceğiniz gibi, Send/Recv fonksiyonlarında var olan otomatik olarak bekleme işlemi burada kullanıcı inisiyatifine bırakılmaktadır. Bu fonksiyonlar tek başlarına kullanılmazlar. Muhakkak MPI_WAIT veya MPI_TEST gibi yardımcı fonksiyonları kullanmaları gerekmektedir.

MPI_Wait(&request, &status)

ISEND ve IRECV fonksiyonları verileri yollayana veya alana kadar beklemelerini sağlayan fonksiyondur. *Status* istenilen mesajın adresini ve tag' ini dönmektedir.

MPI_Test (&request, &flag, &status)

Bu fonksiyon da istenilen verinin ulaşıp ulaşmadığını kontrol etmek için kullanılır. Eğer verimiz istenilen yere ulaşmışsa *flag* olarak *true* değeri dönmektedir. *Status* istenilen mesajın adresini ve tag' ini dönmektedir.

MPI_Reduce(data, result, count, type, op, root,comm)

Bu fonksiyon N tane makinede oluşmuş sonuçları derleyerek ana süreç' te *result* dizisinde saklamaktadır. Geri kalan değişkenlerden *count* "reduce"(alınmış veri) sayısını, *type* alınan verilerin türlerini, *op* yapılan alınma işlemlerinin sonuçlarını, *root* verilerin alındığı süreçleri ve son olarak *comm* haberleşme ortamının adını (MPI_COMM_WORLD) almaktadır.

Yukarıda da gördüğümüz gibi 5 tane işlemci ayrı data yani veriler üretmektedir. İstersek bunları ayrı ayrı yollayabiliriz. Bunun için her makine send/recv fonksiyonlarını kullanabiliriz. Biz bunun yerine MPI_REDUCE fonksiyonunu kullanıyoruz. Elimizde ki bütün "*data*" verilerini MPI_REDUCE aracılığı ile ana işlemcide "*result*" dizisine gönderiyoruz. (Moody, 2011)

MPI_AllReduce(data, result, count, type, op, root, comm)

ALLREDUCE fonksiyonu REDUCE ile işlevsel olarak aynıdır. Tek farkı bütün süreçlere *result* değerini göndermektedir.

MPI_Barrier(comm)

Global haberleşme fonksiyonlarının en önemlilerindendir. Bütün süreçlerin işlemleri tamamlanana kadar devam etmelerini engelleyen senkronizasyon fonksiyonudur.

MPI_Bcast(&buffer, count, datatype, root, comm)

BCAST, *Broadcasting* kelimesinin kısaltılmış halidir. Root olarak sayılan süreçten arabellekteki (buffer) değerleri "comm" ortamında ki bütün süreçlere yollamaktadır. MPI paralel makineler arası haberleşme kütüphanesidir ve 100 den fazla fonksiyonu vardır.

*MPI_Scatter(*sendbuf, sendcnt, sendtype, *recvbuf, recvcnt, recvtype, root, comm)*

Tek bir kaynak task'tan gruptaki her bir task'a ayrı mesajların dağıtılmasını sağlar.

*MPI_Gather (*sendbuf, sendcnt, sendtype, *recvbuf, recvcount, recvtype, root, comm)*

Gruptaki her bir tasktan ayrı mesajları alır ve tek hedefe toplar. Bu rutin MPI_Scatter'ın ters işlemidir

4.2.2 MPI argümanları

Buffer: Yollanacak veya alınacak veriyi referanslayan program adres uzayıdır.

Data Count: Yollanacak olan belirli bir tipteki veri elemanı sayısıdır.

Data Type: Taşınabilirlik için MPI kendi veri tiplerini tanımlamıştır:

MPI	C/C++
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	8 binary digits

Destination: Send rutinlerinde kullanılan bir argüman olup, mesajın nereye ulaştırılacağını belirtir. Alıcı process'in rank'ı ile tanımlanır.

Source: Mesajın nereden geldiğini belirtir. Bu argüman herhangi bir süreçten mesaj almak için MPI_ANY_SOURCE wild card'ına ayarlanabilir.

Tag: Mesajı unique olarak tanımlamak için programcı tarafından gelişigüzel atanan, negatif olmayan bir sayıdır. Send ve receive işlemleri mesaj tag'larında eşleşmelidir. Receive işlemi için MPI_ANY_TAG wild card'ı kullanılabilir. MPI standart 0-32767 arasındaki sayıların kullanılabilmesini garanti eder.

Communicator: Haberleşme içeriğini veya kaynak ve hedef alanları geçerli bir grup süreci işaret eder. Programcı açıkça yeni bir communicator oluşturmadıkça önceden tanımlanmış MPI_COMM_WORLD kullanılır.

Status: Receive işlemi için mesajın kaynağını ve mesajın tag'ını belirtir. C'de bu argüman önceden tanımlı MPI_Status yapısına (MPI_SOURCE ve MPI_TAG elemanlarını içerir) bir pointer'dır.

5. BAZI BİLİNEN ÇARPANLARA AYIRMA ALGORİTMALARININ PARALELLEŞTİRİLMESİ

Genelde şifreleme sistemlerinde kullanılan *n* (çarpanlarına ayrılacak tamsayı) sayısı iki büyük asal sayının çarpımı şeklindedir. (Arjen, 2000) Bu tip sayılara zor sayılar (*hard number*) denir. (Caldwell, 1994) Özel amaçlı çarpanlara ayırma algoritmaları zor sayılar için etkisizdirler. *Basit Bölme* ve *Fermat'ın çarpanlara ayırma yöntemi* özel amaçlı çarpanlara ayırma algoritmalarına örnek verilebilir. (Cormen et al., 2002)

Bu algoritmalar paralelleştirmeye elverişli olduklarından GMP ve MPI kütüphaneleri kullanılarak daha etkin bir hale getirilebilir. (Brent, 1990)

5.1 Basit Bölme Algoritmasının Paralelleştirilmesi

Basit bölme özel amaçlı çarpanlara ayırma algoritmalarından biridir. Bu algoritmada *n*'nin potansiyel bölenleri $d = 2,3,4,...$ aşağıdaki durumlardan biri sağlanana kadar uygulanır.

(1) Eğer $d > n^{1/2}$ ise, *n* asal bir sayıdır.
(2) Eğer $d < n$ ve $d \mid n$ ise, *d* *n*'nin bölenlerinden biridir.
(3) Eğer *d* önceden belirlenen sınır *B*'yi $\left(B < n^{1/2}\right)$ geçerse, o zaman *n*'nin asal çarpanlarından biri *B*'den büyük denir ($p>B$).

Bu algoritmayı daha etkin hale getirmek mümkündür. Eğer *n* tek bir sayıysa, sadece tek *d* değerleri için *n*'nin çarpanları aranır. Diğer bir yol ise, *n*'nin asal çarpanları sadece $n^{1/2}$'den küçük asal sayılar olacağından *d* değerleri bu sayılardan seçilir. Fakat bu yöntemde $n^{1/2}$'den küçük asal sayıların bulunması gerekir. (Matthews, 2011)

Tarayacağımız aralığı elimizdeki işlemci sayı kadar alana bölerek bu işlemi hızlandırmak mümkündür. *p* işlemci sayısı olmak üzere, *n* sayısının potansiyel bölenleri;

$d_1 = 2, 3, 4, ...$
$d_2 = 2 + \lceil n/p \rceil, 3 + \lceil n/p \rceil, 4 + \lceil n/p \rceil, ...$
...
$d_p = 2 + (p-1).\lceil n/p \rceil, 3 + (p-1).\lceil n/p \rceil, 4 + (p-1).\lceil n/p \rceil, ...$

gruplara ayrılmış olur. Eğer n sayısının sadece asal çarpanlarını bulmak istiyorsak, $d_1, d_2, ..., d_p$ değişkenlerinin başlangıç değerleri bulunduktan sonra GMP kütüphanesinin fonksiyonlarından yararlanarak etkili bir şekilde asal sayıları kontrol etmiş oluruz.

5.2 Fermat'ın Çarpanlara Ayırma Algoritmasının Paraleleştirilmesi

Teorem 5.1: $n > 1$ tek tamsayısının asal olmaması için gerek ve yeter koşul, $\exists p, q \in Z^+$ ve $n = q^2 - p^2$ için $q - p > 1$ olmasıdır.

Yani, n pozitif tamsayısı bize verilip, Fermat Çarpanlara Ayırma Yöntemi ile çarpanlarına ayırmamız istendiğinde eğer n asal değilse; mutlaka $n = q^2 - p^2$ olacak şekilde uygun bir çözüm vardır. Dolayısıyla, $n + p^2$ değeri tam kareye eşit olanadek n sayısına, 1'den başlayarak sırasıyla $1, 2, ..., (q-p)/2$ 'ye kadar tüm sayıların karesi tek tek eklenir. Bu işlem tam olarak $(q-p)/2$ devam eder

$n = q^2 - p^2$ olsun. O halde, $n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{q-p}{2}\right)^2$ olur. Bu durumda çözüm için $\left(\frac{q-p}{2}\right)^2$ ifadesini eşitliğin sağ tarafına geçirirsek, eşitlik $n + \left(\frac{q-p}{2}\right)^2 = \left(\frac{p+q}{2}\right)^2$ şeklinde yazılabilir. Bu durumda algoritmanın tek yapması gereken 1'den başlayarak birer birer $\left(\frac{q-p}{2}\right)$ değerine kadar tek tek tüm sayıların karesini almaktır.

$n + 1^2$ tam kare mi? (Hayır)
$n + 2^2$ tam kare mi? (Hayır)
……
$n + \left(\frac{q-p}{2}\right)^2$ tam kare mi? (Evet)

Eklenecek sayıları *t* işlemci sayısı aralığında gerçekleştirirsek daha hızlı ve etkili bir algoritma elde etmiş oluruz. Yani;

1. işlemci	$n+1^2$	tam kare mi?
	$n+(t+1)^2$	tam kare mi?
	$n+(2t+1)^2$	tam kare mi?
	…	
2. işlemci	$n+2^2$	tam kare mi?
	$n+(t+2)^2$	tam kare mi?
	$n+(2t+2)^2$	tam kare mi?
	…	
…		
t. işlemci	$n+t^2$	tam kare mi?
	$n+(t+t)^2$	tam kare mi?
	$n+(2t+t)^2$	tam kare mi?
	…	

“Evet” cevabını bulan işlemci diğer işlemcileri uyararak işlemi sonlandırır. Böylece *n* sayının çarpanları olan *p* ve *q* asal sayılarını bulmuş oluruz. Bu yöntem bize birim zamanda taradığımız sayı adetini arttırmış oluyor. Kullandığımız sistem ne kadar iyiyse işlemci adeti yani *t* sayısı o kadar fazla oluyor.

6. YENİ ÇARPANLARINA AYIRMA YÖNTEMLERİ VE PARALELLEŞTİRİLMESİ

Bu bölümde önerilen yöntem ve algoritmalarda yeni parametreler tanımlanmış ve bu parametrelerin daha küçük değerler aldığı izlenmiştir ve böylece de bu parametreleri belirlemek için daha az zaman gerektiği gösterilmiştir. (Kutucu vd, 2008; Keskin, 2011)

6.1 Sayıların Çarpanlarına Göre Yapıları Üzerine

$n \in Z^+$ sayısının $p, q \in P$ gibi iki asal sayının çarpımından oluştuğunu varsayalım. $n = p \cdot q$, $m = \lfloor \sqrt{n} \rfloor$ olarak işaretleyelim.

Teorem 4.1'den aşağıdaki sonucu yazabiliriz.

Sonuç 6.1: $\exists t > m,\ \exists k \geq 1,\ n = t^2 - k^2$'dir.

Buradan aşağıdakileri yazabiliriz.

$$n = t^2 - k^2 = (t-k)(t+k) = p \cdot q$$

$t > m$ olduğu için, $\exists s,\ t = m + s$ şeklinde gösterebiliriz.

Bunları göz önüne alırsak, *n*'i aşağıdaki gibi yazabiliriz.

$$\Rightarrow n = (\underbrace{m+s-k}_{p})(\underbrace{m+s+k}_{q}) \qquad (6.1)$$

Diğer taraftan *n*'i $n = m^2 + \Delta n$ şeklinde de yazabiliriz. *n*'i *m*'ye bağımlı olarak

$$\exists l,\ f, \qquad n = (\underbrace{m-l}_{p})(\underbrace{m+f}_{q}).$$

şeklinde de gösterebiliriz.

Burada her zaman $f > l$ için $\exists g,\ f = l + g$ yazabiliriz.

Buradan ise,

$$n = (\underbrace{m-l}_{p})(\underbrace{m+l+g}_{q}) \tag{6.2}$$

şeklinde gösterebiliriz.

n sayısının *p* ve *q* sayılarının çarpımı şeklinde iki farklı gösterimden ((6.1) ve (6.2)) aşağıdaki sistem denkliğini elde ederiz.

$$\left.\begin{aligned} m-l &= m+s-k \\ m+l+g &= m+s+k \end{aligned}\right\}$$

Bu denklemleri taraf tarafa toplayıp, çıkarırsak

$$\left.\begin{aligned} -l &= s-k \\ l+g &= s+k \end{aligned}\right\}$$

alırız. Buradan,

$$\left.\begin{aligned} l &= k-s \\ g &= 2s \end{aligned}\right\}$$

Sonuçta

$$\left.\begin{aligned} p &= m-l \\ q &= m+l+2s \end{aligned}\right\} \tag{6.3}$$

buluruz.

Teorem 4.4'ü göz önüne alırsak,

$$t = \frac{p+q}{2} \quad \text{ve} \quad k = \frac{q-p}{2}$$

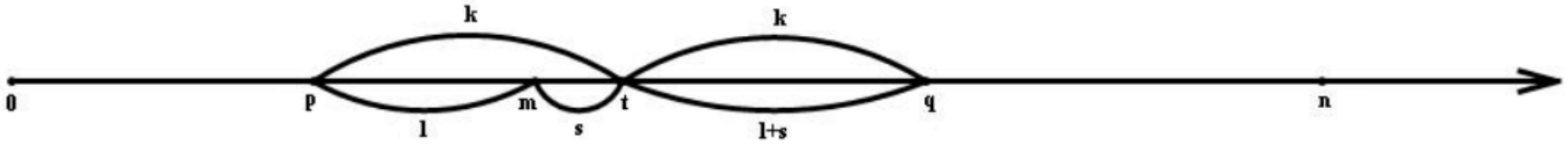
olur. Burada *p* ve *q*'nü *t* ve *k* ile ifade edip (6.3)'de yerine yazarak, *l* ve *s*'nin *m*, *p*, *t*'ye göre ifadesini elde ederiz:

$$l = m - p$$
$$s = t - m$$
$$k = l + s$$

Yukarıda yazdıklarımızı göz önüne alarak bütün parametreleri (Şekil 6.1)'deki gibi gösterebiliriz.

Fermat'ın çarpanlarına ayırma yönteminde $n+k^2=t^2$ formülü göz önüne alınarak $k=1,2,...$ değerleri için $n+k^2$ toplamının tam kare (t^2) olup-olmadığı kontrol edilirdi. Böyle bir k bulunduğunda iterasyon bitiyordu. Bu koşulu sağlayan k ve ona uygun t'ye göre $p=t-k$ ve $q=t+k$ olarak belirlenir. Göründüğü gibi algoritmanın iterasyonlarının sayısı k'nın boyutuna bağımlıdır.

Yukarıdaki formüllerden de göründüğü gibi l ve s parametreleri k'ya göre daha küçüktürler. Bizim geliştirdiğimiz algoritmalarda l veya s (veya bunlardan daha küçük olan u, v_i) parametrelerinin değerlerine göre kontrol yapılır. Bu parametreler her zaman k'dan küçük olduğu için daha az zaman gerektirir.



Burada : $n = pq, \quad n \in \mathbb{Z}^+ ; \quad p,q \in P , \; m = \lfloor \sqrt{n} \rfloor$

$$t = \frac{(p+q)}{2}, \quad k = \frac{(q-p)}{2}$$

$$l = m - p, \quad s = t - m, \quad k = l + s$$

Şekil 6.1 n sayısının p ve q çarpanlarına göre yapısal gösterimi

6.2“*s*” Parametresine Dayalı Algoritmalar

$n = p \times q,\ m = \lfloor \sqrt{n} \rfloor,\ t = \dfrac{p+q}{2}$ olsun.

Bu bölümde iki yöntem ele alınacak. Birinci yöntemde $t = m+1, m+2, \ldots$ değerlerini vererek, $S = t^2 - n$ farkının tam kare olup-olmadığı kontrol edilir, yani $\exists s$ aranır ki, $t = m+s$ olduğunda $S = (m+s)^2 - n$ tam kare olur, buradan da $k = \sqrt{S}$ ve $p = t-k,\ q = t+k$ olarak belirlenir.

İkinci yöntemde ise (6.3) formülleri göz önüne alınarak 2. dereceden denklem çözülür.

6.2.1 Birinci yöntem

n, *p* ve *q* olmak üzere iki asal sayının çarpımı, *q* *p*'ye eşit veya *p*'den küçük bir asal sayı, *p* ve *q* arasındaki fark 2*k* olmak üzere, $\lfloor \sqrt{n} \rfloor = m$ ve $n - m^2 = d$ olarak belirtilsin.

$$n = p \times q,\ \ p \le q,\ \ p = q - 2k,\ \lfloor \sqrt{n} \rfloor = m,\ \ n - m^2 = d$$

Durum 1. Eğer $d = 0 \Rightarrow n = q \times q \Rightarrow q = p, k = 0$

Durum 2. $k \ge 1 \Rightarrow q(q-2k) = n \Rightarrow$

$$\Rightarrow q^2 - 2kq - n = 0$$

$$\Rightarrow q = k \pm \sqrt{k^2 + n}$$

1.durumda $d = 0$ olsun. O halde, $n = q \times q$ olur, dolayısıyla $q = p$ ve $k = 0$ değerine eşittir.

2.durumda $k \ge 1$ için $q(q-2k) = n$'dir.

O halde, $\sqrt{k^2+n} = t \Rightarrow t^2 = k^2 + n \Rightarrow m^2 \le t^2 < (m+1)^2$

$$n - m^2 = d \Rightarrow n = m^2 + d \Rightarrow t^2 = k^2 + m^2 + d$$

$$\Rightarrow k^2 = t^2 - m^2 - d \text{ olur.} \qquad (6.4)$$

(6.4) formülünde $t = m+1, m+2, ...$ değerlerini vererek

$$S = t^2 - m^2 - d$$

ifadesinin değeri tam kare olana kadar prosedürü devam ettiririz, yani $\exists t = m + j$ ve $\exists k$ değeri bulunur ki, $S = k^2$ şartı sağlanır. O halde,

$$q = k + t \text{ ve } p = t - k$$

olduğu görülür.

ALGORİTMA-1

$p, q, n, k, m, d, j, t \in N$ olsun

1.1 $m = \lfloor \sqrt{n} \rfloor$, $d = n - m^2$

1.2 $j = 1$

1.3 $t = m + j$

1.4 $(S = t^2 - m^2 - d)$, $S = t^2 - n$

1.5 Eğer $\sqrt{S}$ tamsayı ise 1.7. adıma git

1.6 $j = j + 1$ ve 1.3. adıma git

1.7 $k = \sqrt{S}$, $q = t + k$, $p = t - k$.

1.8 SON.

Algoritmada yer alan *j* değişkeni bir bir arttığı için işlem yükü fazla olacak. Burada j değişkeni paralel sistemde daha hızlı artacaktır. Yani elimizde var olan işlemci sayısı kadar artış gösterecek. *ps* işlemci sayısını göstermek üzere algoritmayı şöyle düzenleyebiliriz:

ALGORİTMA-1P

$p, q, n, k, m, d, i, j_i, s, t, tk, ps \in N$ olsun

2.1 $m = \lfloor \sqrt{n} \rfloor,\ tk = (n+1)/2$

2.2 Paralel ortamı başlat

2.3 $j_i = i \quad , i = 1, ..., ps$ // ps işlemci adeti olmak üzere her i değeri bir işlemciyi temsil eder

2.4 $t = m + j_i$

2.5 Eğer $t > tk$ ise 2.11. adıma git

2.6 $(S = t^2 - m^2 - d),\ S = t^2 - n$ // her işlemci hesapladığı t değerini dener

2.7 Eğer $\sqrt{S}$ tamsayı ise 2.8. adıma git

2.8 $j_i = j_i + ps$ ve 2.4. adıma git

2.9 Paralel ortamı sonlandır.

2.10 $k = \sqrt{S}, q = t + k,\ p = t - k.$

2.11 SON.

ÖRNEK 6.1:

$p = 13, q = 89$ asal sayıları, $n = 13 \cdot 89 = 1157, m = \lfloor \sqrt{1157} \rfloor = 34$ değerleri için algoritma çalıştırıldığında, $j = 17$ iken $t = 51$ ve $k = 38$ için $p = 51 - 38 = 13, q = 51 + 38 = 89$ değerleri elde edilir (Çizelge 6.1) :

Çizelge 6.1 Birinci yöntemle 1157 sayısının çarpanlarına ayrılması örneği

j	$t = m + j = 34 + j$	$S = t^2 - n = (m + j)^2 - n = (34 + j)^2 - n$	$\sqrt{S} \in N\,?,\ k = \sqrt{S}$
1	34 + 1 = 35	35^2 - 1157 = 1225 – 1157 = 68	8,24… -
2	34 + 2 = 36	36^2 - 1157 = 1296 – 1157 = 139	11,79… -
3	34 + 3 = 37	37^2 - 1157 = 1369 – 1157 = 212	14,56… -
4	34 + 4 = 38	38^2 - 1157 = 1444 – 1157 = 287	16,94… -
5	34 + 5 = 39	39^2 - 1157 = 1521 – 1157 = 364	19,07… -
6	34 + 6 = 40	40^2 - 1157 = 1600 – 1157 = 443	21,04… -
7	34 + 7 = 41	41^2 - 1157 = 1681 – 1157 = 524	22,89… -
8	34 + 8 = 42	42^2 - 1157 = 1764 – 1157 = 607	24,63… -
9	34 + 9 = 43	43^2 - 1157 = 1849 – 1157 = 692	26,30… -
10	34 + 10 = 44	44^2 - 1157 = 1936 – 1157 = 779	27,91… -
11	34 + 11 = 45	45^2 - 1157 = 2025 – 1157 = 868	29,46… -
12	34 + 12 = 46	46^2 - 1157 = 2116 – 1157 = 959	30,96… -
13	34 + 13 = 47	47^2 - 1157 = 2209 – 1157 = 1052	32,43… -
14	34 + 14 = 48	48^2 - 1157 = 2304 – 1157 = 1147	33,86… -
15	34 + 15 = 49	49^2 - 1157 = 2401 – 1157 = 1244	35,27… -
16	34 + 16 = 50	50^2 - 1157 = 2500 – 1157 = 1343	36,64… -
17	34 + 17 = 51	51^2 - 1157 = 2601 – 1157 = 1444	38 +

$p = 13, q = 89$ asal sayıları, $n = 13 \cdot 89 = 1157$, $m = \left\lfloor \sqrt{1157} \right\rfloor = 34$ değerleri için algoritma 2, 8 işlemci için çalıştırıldığında 1. işlemci 3 adım sonra $k = 51$ sayısını elde ettiğinden sonuca daha çabuk ulaşmış oluyor (Çizelge 6.2) :

Çizelge 6.2 Algoritma 2 de 1. işlemcinin çalışması örneği

J	$t = m + j = 32 + j$	$S = t^2 - n = (m + j)^2 - n = (34 + j)^2 - n$	$\sqrt{S} \in N\,?,\ k = \sqrt{S}$
1	34 + 1 = 35	35^2 - 1157 = 1225 – 1157 = 68	8,24… -
9	34 + 9 = 43	43^2 - 1157 = 1849 – 1157 = 692	26,30… -
17	34 + 17 = 51	51^2 - 1157 = 2601 – 1157 = 1444	38 +

6.2.2 İkinci yöntem

Bu yöntemde aşağıdaki işaretlemeler göz önüne alınarak *l*'e göre ikinci dereceden denklem çözülür.

$$n = p \cdot q, \quad m = \left\lfloor \sqrt{n} \right\rfloor, \quad t = (p+q)/2, \quad k = (q-p)/2,$$

$$s = t - m, \quad l = m - p, \quad u = l - s, \quad v = s - u$$

$$p = m - l, \quad q = m + l + 2s$$

$$(m-l)\,(m+l+2s) = n$$

$$m^2 - l^2 + 2s(m-l) = n \qquad (6.5)$$

$$l^2 + 2sl - 2sm + n - m^2 = 0$$

$$l^2 + 2sl - 2sm + \Delta n = 0 \qquad // \;\; \Delta n = n - m^2$$

$$l = -s \pm \sqrt{s^2 + 2sm - \Delta n}$$

Burada *s*'e sıfırdan başlayarak değerler verilir ve karekök tamsayılı değer alana kadar iterasyona devam edilir. Bulunmuş *s* ve *l*'e göre (6.3) formülü ile *p* ve *q* belirlenir. Algoritma-1p de yaptığımız gibi elimizdeki işlemci sayısı kadar aralıklarla *s* değişkeni arttırıldığında algoritmamız daha verimli bir hal almaktadır.

ALGORİTMA-2P

$p, q, n, k, m, d, i, \; j, s_i, sk, ps \in N$ olsun

3.1 $m = \left\lfloor \sqrt{n} \right\rfloor, \; sk = \left(n+1\right)/2 - m$

3.2 Paralel ortamı başlat

3.3 $s_i = i - 1 \quad , i = 1, ..., ps$ // *ps* işlemci adeti olmak üzere her *i* değeri bir işlemciyi temsil eder

3.4 Eğer $s_i > sk$ ise 3.11. adıma git

3.5 $A = s_i^{\,2} + 2s_i m - \Delta n \quad // \Delta n = n - m^2$ // her işlemci hesapladığı *s* değerini dener

3.6 Eğer $\sqrt{A}$ tamsayı ise 3.7. adıma git

3.7 $s_i = s_i + ps$ ve 3.4. adıma git

3.8 Paralel ortamı sonlandır.

3.9 $l = -s_i \pm \sqrt{A}$

3.10 $p = m - l, \quad q = m + l + 2s_i$

3.11 SON.

6.3 *“l”* Parametresine Dayalı Yöntem

Bu yöntemde yukarıdaki (6.5) denkleminden *s* için kesirli bir ifade belirlenerek bu ifadenin tam değer olması koşulu aranır.

$$n = p \cdot q, \quad p = m - l, \quad q = m + l + 2s$$

$$2s(m-l) = l^2 + \Delta n$$

$$s = \frac{l^2 + \Delta n}{2(m-1)}$$

Burada *l*’ye 0’dan başlayarak değerler verilir ve kesrin değeri tamsayı olana kadar iterasyonlar devam ettirilir. Bölünmüş *l* ve *s*’ye göre *p* ve *q* belirlenir. Diğer yöntemlerde olduğu gibi *l* ’ye işlemci sayısı kadar iterasyon uygulanır. Böylece etkin bir algoritma elde etmiş oluruz.

ALGORİTMA-3P

$p, q, n, k, m, d, i, j, s, l, lk, ps \in N$ olsun

4.1 $m = \lfloor \sqrt{n} \rfloor$, $lk = m - 1$

4.2 Paralel ortamı başlat

4.3 $l_i = i - 1$, $i = 1, ..., ps$ // *ps* işlemci adeti olmak üzere her *i* değeri bir işlemciyi temsil eder

4.4 Eğer $l_i > lk$ ise 4.10. adıma git

4.5 $s = \left(l_i^2 + \Delta n\right) \big/ 2\left(m - l_i\right)$ // $\Delta n = n - m^2$ // her işlemci hesapladığı *l* değerini dener

4.6 Eğer *s* tamsayı ise 4.7. adıma git

4.7 $l_i = l_i + ps$ ve 4.4. adıma git

4.8 Paralel ortamı sonlandır.

4.9 $p = m - l_i, \quad q = m + l_i + 2s$

4.10 SON.

6.4 “*u*” Parametresine Dayalı Yöntem

Bu yöntemde aşağıdaki işaretlemeler göz önüne alınarak *s*’e göre ikinci dereceden denklem çözülür.

$$u = l - s \Rightarrow l = u + s \Rightarrow p = m - l, \quad q = m + l + 2s \Rightarrow$$
$$\Rightarrow p = m - u - s, \quad q = m + u + s + 2s = m + u + 3s$$

$$p \cdot q = n \Rightarrow (m-u-s)(m+u+3s) = \big((m-u)-s\big)\big((m+u)+3s\big) =$$
$$= m^2 - u^2 - s(m+u) + 3s(m-u) - 3s^2 =$$
$$= m^2 - u^2 + 2ms - 4su - 3s^2 = n \Rightarrow$$
$$\Rightarrow 3s^2 - 2s(m-2u) + n - m^2 + u^2 = 0$$

$$s = \frac{1}{3}\left((m-2u) \pm \sqrt{(m-2u)^2 - 3(n - m^2 + u^2)}\right)$$

$$D = m^2 - 4mu + 4u^2 - 3n + 3m^2 - 3u^2 =$$
$$= 4m^2 - 4mu + u^2 - 3n = (3m^2 - 3n) + m^2 - 4mu + u^2$$
$$= (3\Delta n + m^2) - 4mu + u^2$$

$$s = \frac{1}{3}\left((m-2u) \pm \sqrt{(3\Delta n + m^2) + u^2 - 4mu}\right)$$

Burada *u*’ya sıfırdan başlayarak karekök tamsayılı değer alana kadar değerler verilerek iterasyona devam edilir. İterasyon paralel ortamda işlemci sayısına göre yapılır.

ALGORİTMA-4P

$p, q, n, k, m, d, i, j, s, u_i, uk, ps \in N$ olsun

5.1 $m = \lfloor \sqrt{n} \rfloor$, $uk = (4m - n - 3)/2$

5.2 Paralel ortamı başlat

5.3 $u_i = i - 1$, $i = 1, ..., ps$ // *ps* işlemci adeti olmak üzere her *i* değeri bir işlemciyi temsil eder

5.4 Eğer $u_i > uk$ ise 5.11. adıma git

5.5 $A = \sqrt{(3\Delta n + m^2) + u_i{}^2 - 4mu_i}$ // $\Delta n = n - m^2$ // her işlemci hesapladığı *l* değerini dener

5.6 Eğer *A* tamsayı ise 5.7. adıma git

5.7 $u_i = u_i + ps$ ve 5.4. adıma git

5.8 Paralel ortamı sonlandır.

5.9 $s = \frac{1}{3}\left((m - 2u_i) \pm \sqrt{A}\right)$

5.10 $p = m - u_i - s, \quad q = m + u_i + 3s$

5.11 SON.

6.5 "v_i" Parametrelerine Dayalı Yöntem

Yukarıda gösterdiğimiz gibi yeni tanımladığımız *l* ve *s* parametreleri *k*'ya göre daha küçük değerler alır ve bu yüzden de *p* ve *q* çarpanlarını bulmak için daha hızlı algoritma tasarlamak imkanı verir. Sonradan tanımladığımız $v_0 = u$ parametresi ise *l* ve *s*'ye göre daha küçük değerler alıyor. Bu bölümde tanımlayacağımız v_i parametreleri ise $v_{i+1} < v_i$ koşulunu sağlayacak ve böylece her bir yeni v_i daha küçük olmakla daha hızlı algoritma tasarlamak imkanı verir.

$$v_0 = u = l - s \Rightarrow p = m - u - s, \quad q = m + u + 3s$$

$$s = \frac{1}{3}((m - 2u) \pm \sqrt{(3\Delta n + m^2) + u^2 - 4mu}$$

$$v_1 = u - s \Rightarrow p = m - v - 2s, \quad q = m + v + 4s$$
$$s = \frac{1}{8}((m-3v) \pm \sqrt{(8\Delta n + m^2) + v^2 - 6mv}$$

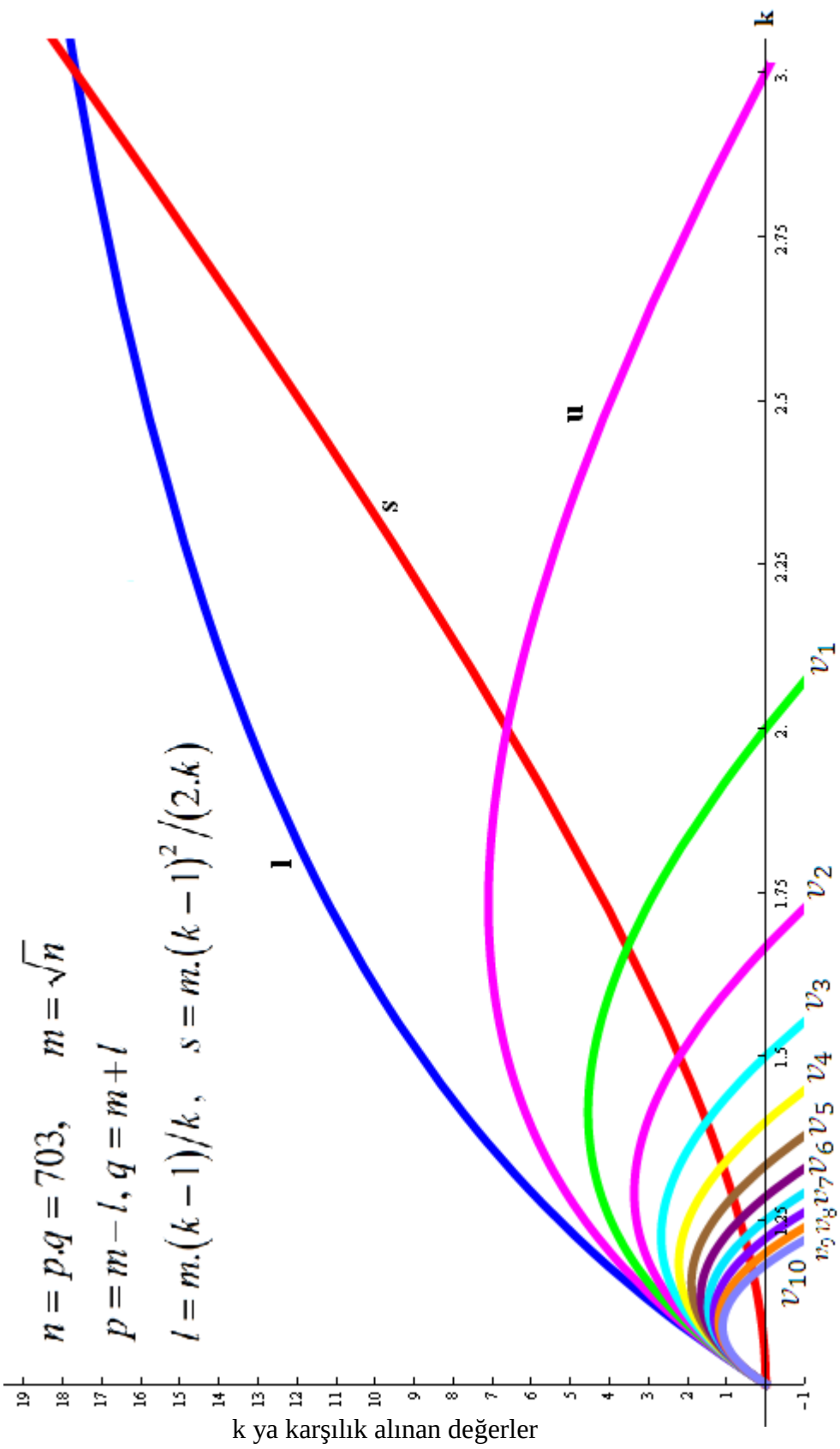
$$v_2 = v_1 - s \Rightarrow p = m - v - 3s, \quad q = m + v + 5s$$
$$s = \frac{1}{15}((m-4v) \pm \sqrt{(15\Delta n + m^2) + v^2 - 8mv}$$

$$v_3 = v_2 - s \Rightarrow p = m - v - 4s, \quad q = m + v + 6s$$
$$s = \frac{1}{24}((m-5v) \pm \sqrt{(24\Delta n + m^2) + v^2 - 10mv}$$

$$v_4 = v_3 - s \Rightarrow p = m - v - 5s, \quad q = m + v + 7s$$
$$s = \frac{1}{35}((m-6v) \pm \sqrt{(35\Delta n + m^2) + v^2 - 12mv})$$

Böylece v_i 'ler için genel formülü aşağıdaki gibi yazabiliriz.

$$v_i = v_{i-1} - s \Rightarrow p = m - v - (i+1)s, \quad q = m + v + (i+3)s$$
$$s = \frac{1}{(i+1)(i+3)}((m-(i+2)v) \pm \sqrt{((i+1)(i+3)\Delta n + m^2) + v^2 - 2(i+2)mv})$$



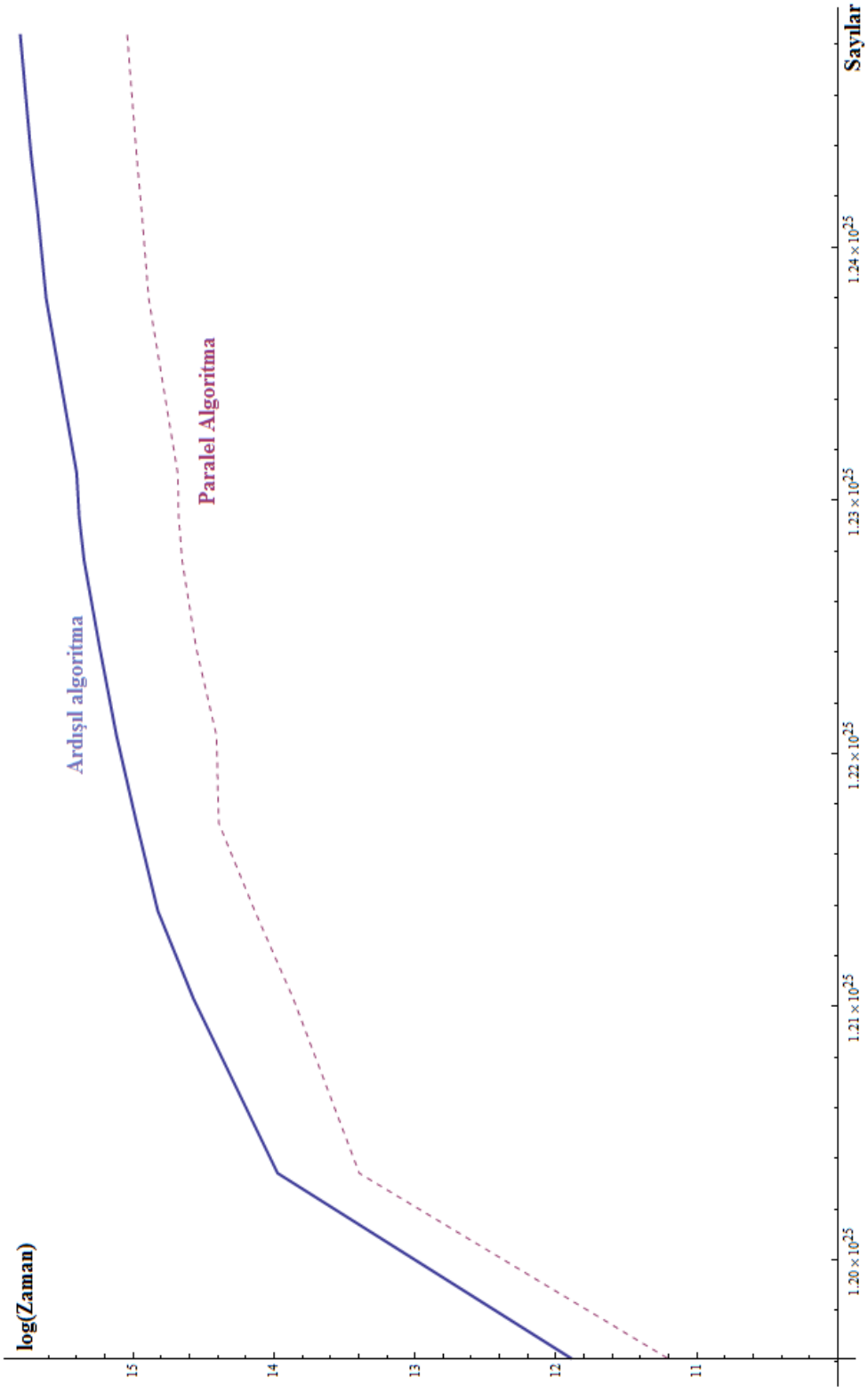
Şekil 6.2 v_i parametreleri arasındaki bağlantı

7. HESAPLAMA DENEMELERİ

Hazırlanan programlar yardımıyla Fermat algoritması ve yeni çarpanlara ayırma algoritmalarının ardışıl ve paralel çalışma süreleri karşılaştırılmış ve önerilen algoritmaların etkinliğinden bahsedilmiştir. (Keskin, 2011) Bu hesaplama denemelerinde çift çekirdek bir işlemci kullanıldığından 2 süreç ile çalışmıştır.

Çizelge 7.1 Fermat'ta kullanılan sayılar ve çözüm zamanları

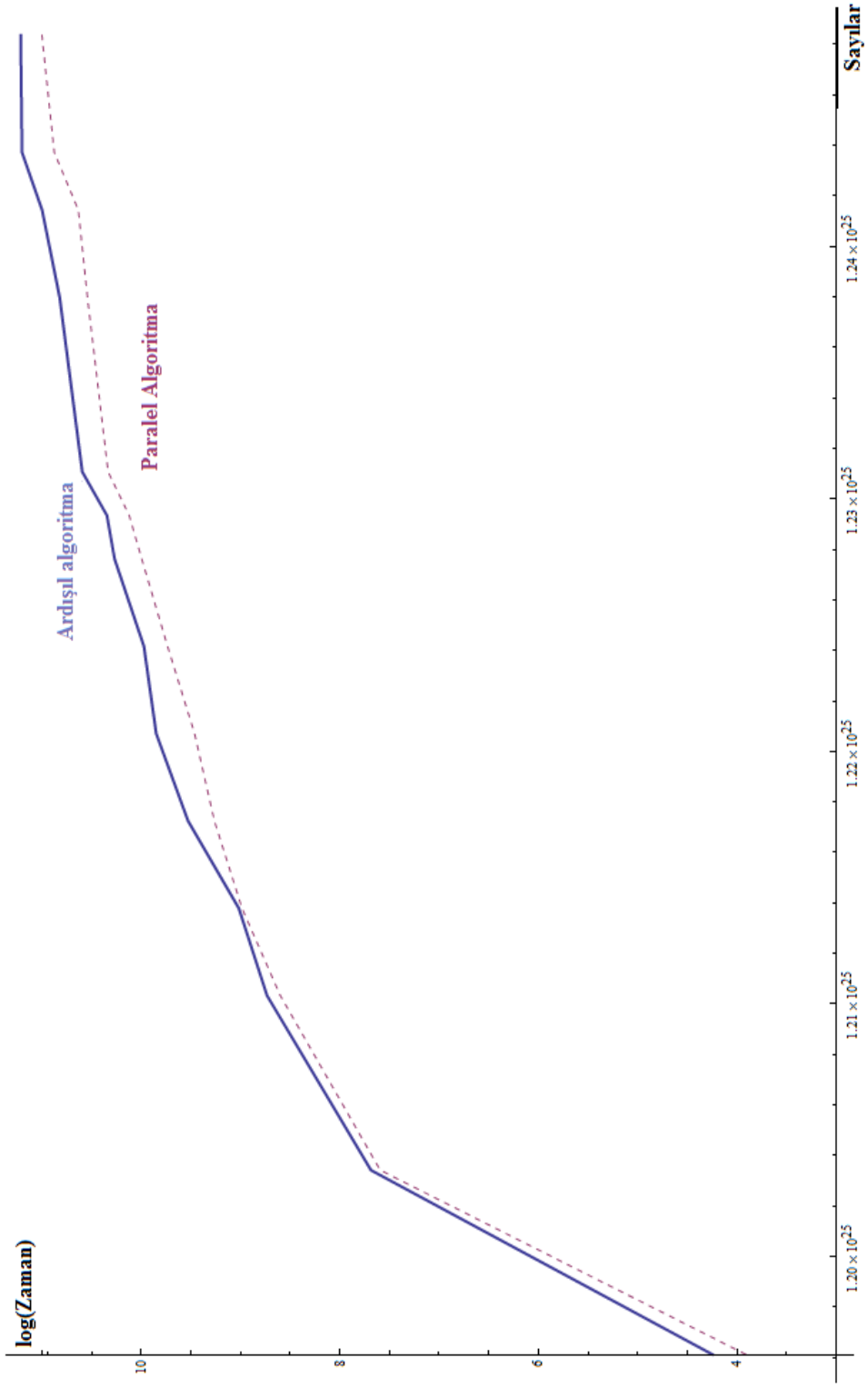
Kullanılan Sayılar			**Paralel**		**Ardışıl**
n = p.q	**p**	**q**	**Zaman(ms)**	**Süreç**	**Zaman(ms)**
11961099199121134769563567	**3456754259947**	**3460211024461**	74010	0	146853
12034074185291329380242507	**3456754259947**	**3481321864481**	656985	0	1172826
12103209270531750431361871	**3456754259947**	**3501321864493**	1050665	0	2133568
12137776813089739380242507	**3456754259947**	**3511321864481**	1380078	0	2744253
12172344355848220076200069	**3456754259947**	**3521321864527**	1780038	1	3188808
12206911898240284820603249	**3456754259947**	**3531321864467**	1809275	0	3677319
12241479441199257263637737	**3456754259947**	**3541321864571**	2089289	1	4139856
12276046983494532888762401	**3456754259947**	**3551321864483**	2308680	0	4625370
12293330754725132803563461	**3456754259947**	**3556321864463**	2365969	1	4792203
12310614526211532533600599	**3456754259947**	**3561321864517**	2382017	1	4877118
12379749611486521127319433	**3456754259947**	**3581321864539**	2927280	0	6057936
12414317153837104820603249	**3456754259947**	**3591321864467**	3077515	0	6436557
12437131731973495546162931	**3456754259947**	**3597921864473**	3193898	0	6750243
12483452239008390786523673	**3456754259947**	**3611321864459**	3406591	1	7274718



Şekil 7.1 Fermat'ın ardışıl ve paralel algoritmalarının karşılaştırılması

Çizelge 7.2 Algoritma-1p'de kullanılan sayılar ve çözüm zamanları

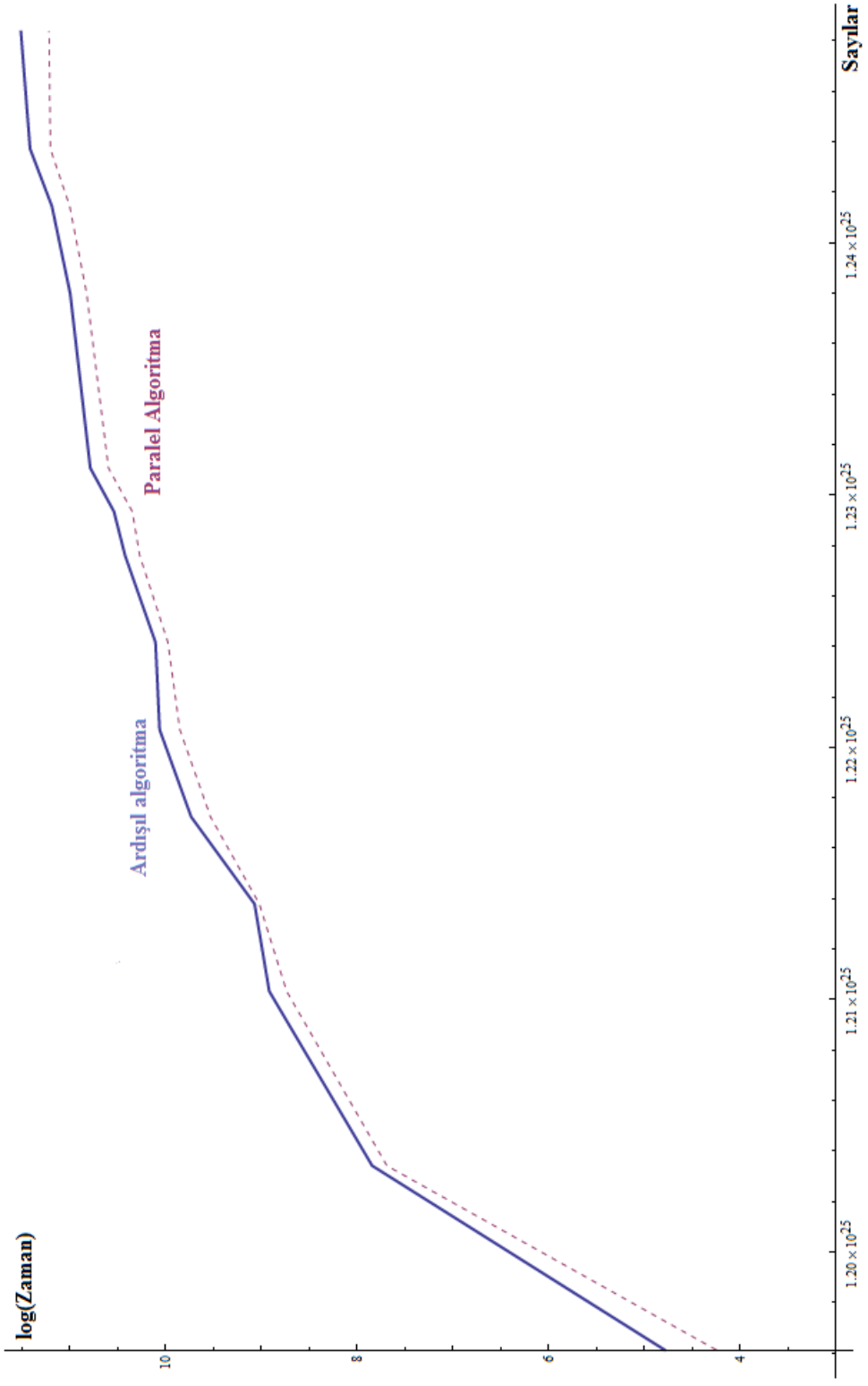
Kullanılan Sayılar			Paralel		Ardışıl
n = p.q	**p**	**q**	**Zaman(ms)**	**Süreç**	**Zaman(ms)**
11961099199121134769563567	**3456754259947**	**3460211024461**	50	0	86
12034074185291329380242507	**3456754259947**	**3481321864481**	1986	0	2316
12103209270531750431361871	**3456754259947**	**3501321864493**	5432	1	6514
12137776813089739380242507	**3456754259947**	**3511321864481**	7987	1	8401
12172344355848220076200069	**3456754259947**	**3521321864527**	10546	0	12903
12206911898240284820603249	**3456754259947**	**3531321864467**	12873	1	15902
12241479441199257263637737	**3456754259947**	**3541321864571**	16901	0	19234
12276046983494532888762401	**3456754259947**	**3551321864483**	21912	0	25546
12293330754725132803563461	**3456754259947**	**3556321864463**	24876	1	30125
12310614526211532533600599	**3456754259947**	**3561321864517**	30765	1	37210
12379749611486521127319433	**3456754259947**	**3581321864539**	37889	1	45010
12414317153837104820603249	**3456754259947**	**3591321864467**	41532	0	50001
12437131731973495546162931	**3456754259947**	**3597921864473**	52916	1	65412
12483452239008390786523673	**3456754259947**	**3611321864459**	59910	0	72345



Şekil 7.2 Algoritma-1p'nin ardışıl ve paralel algoritmalarının karşılaştırılması

Çizelge 7.3 Algoritma-2p'de kullanılan sayılar ve çözüm zamanları

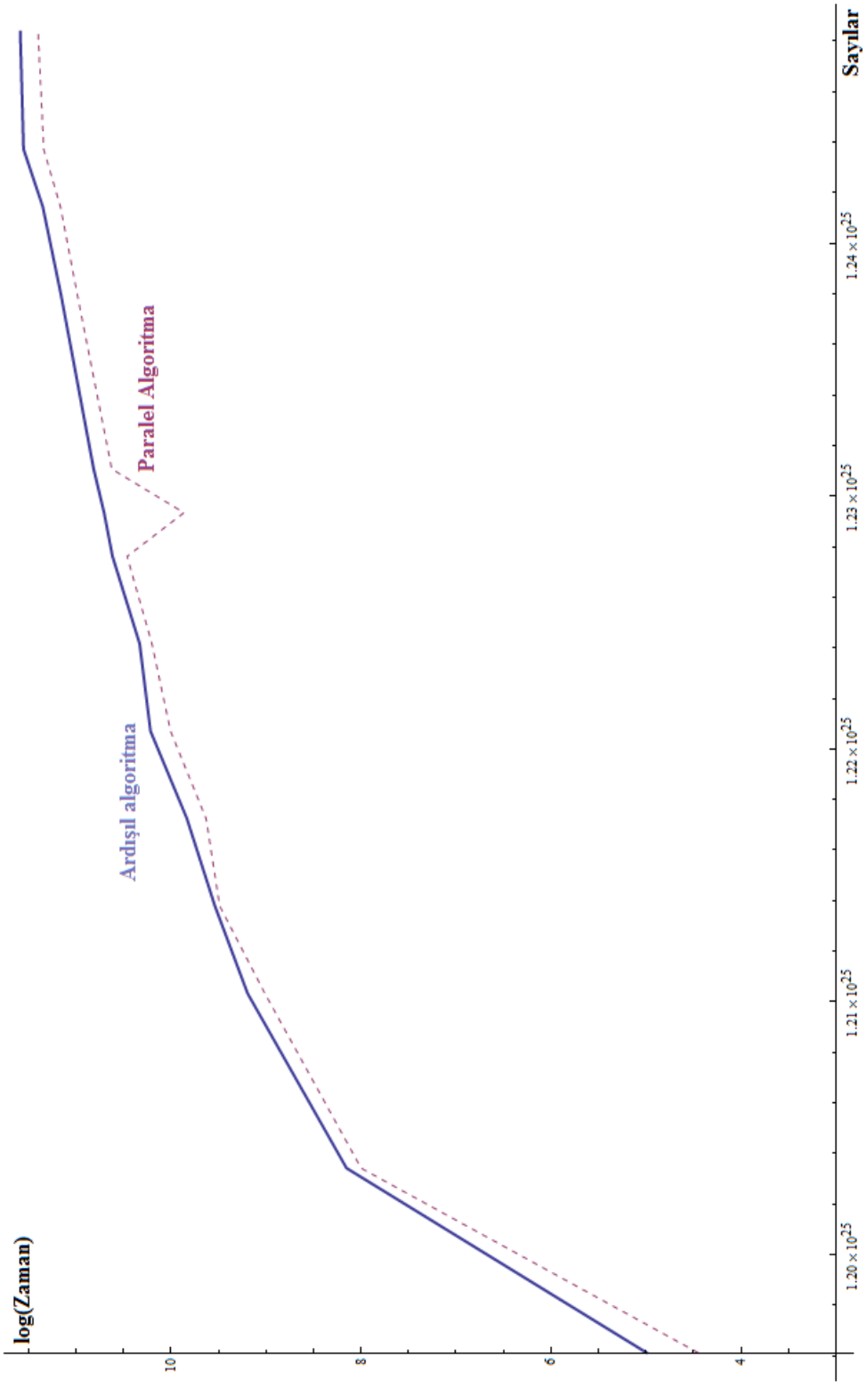
Kullanılan Sayılar			Paralel		Ardışıl
n = p.q	**p**	**q**	**Zaman(ms)**	**Süreç**	**Zaman(ms)**
11961099199121134769563567	**3456754259947**	**3460211024461**	70	0	120
12034074185291329380242507	**3456754259947**	**3481321864481**	2180	1	2542
12103209270531750431361871	**3456754259947**	**3501321864493**	6210	1	7447
12137776813089739380242507	**3456754259947**	**3511321864481**	8250	0	8678
12172344355848220076200069	**3456754259947**	**3521321864527**	13753	0	16827
12206911898240284820603249	**3456754259947**	**3531321864467**	18945	1	23403
12241479441199257263637737	**3456754259947**	**3541321864571**	21452	0	24413
12276046983494532888762401	**3456754259947**	**3551321864483**	28819	1	33598
12293330754725132803563461	**3456754259947**	**3556321864463**	31120	0	37687
12310614526211532533600599	**3456754259947**	**3561321864517**	39910	1	48271
12379749611486521127319433	**3456754259947**	**3581321864539**	50123	1	59543
12414317153837104820603249	**3456754259947**	**3591321864467**	59821	0	72019
12437131731973495546162931	**3456754259947**	**3597921864473**	73212	0	90501
12483452239008390786523673	**3456754259947**	**3611321864459**	74213	1	99617



Şekil 7.3 Algoritma-2p'nin ardışıl ve paralel algoritmalarının karşılaştırılması

Çizelge 7.4 Algoritma-3p'de kullanılan sayılar ve çözüm zamanları

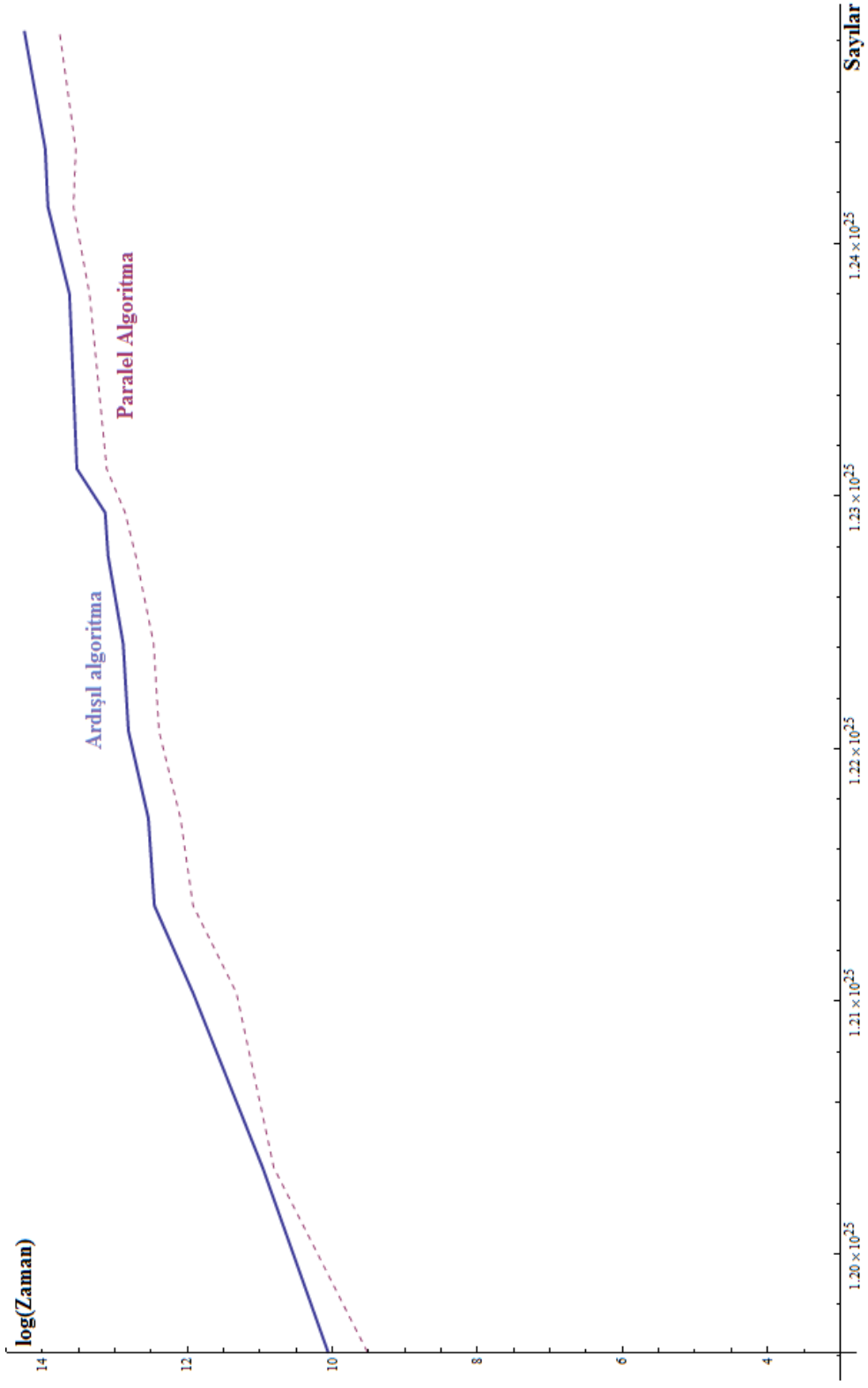
Kullanılan Sayılar			Paralel		Ardışıl
n = p.q	**p**	**q**	**Zaman(ms)**	**Süreç**	**Zaman(ms)**
11961099199121134769563567	**3456754259947**	**3460211024461**	86		148
12034074185291329380242507	**3456754259947**	**3481321864481**	2980		3475
12103209270531750431361871	**3456754259947**	**3501321864493**	8210		9845
12137776813089739380242507	**3456754259947**	**3511321864481**	13201		13885
12172344355848220076200069	**3456754259947**	**3521321864527**	15235		18640
12206911898240284820603249	**3456754259947**	**3531321864467**	22135		27343
12241479441199257263637737	**3456754259947**	**3541321864571**	26932		30650
12276046983494532888762401	**3456754259947**	**3551321864483**	34983		40785
12293330754725132803563461	**3456754259947**	**3556321864463**	19235		44563
12310614526211532533600599	**3456754259947**	**3561321864517**	41120		49734
12379749611486521127319433	**3456754259947**	**3581321864539**	59129		70242
12414317153837104820603249	**3456754259947**	**3591321864467**	70540		84924
12437131731973495546162931	**3456754259947**	**3597921864473**	84195		104077
12483452239008390786523673	**3456754259947**	**3611321864459**	89215		107733



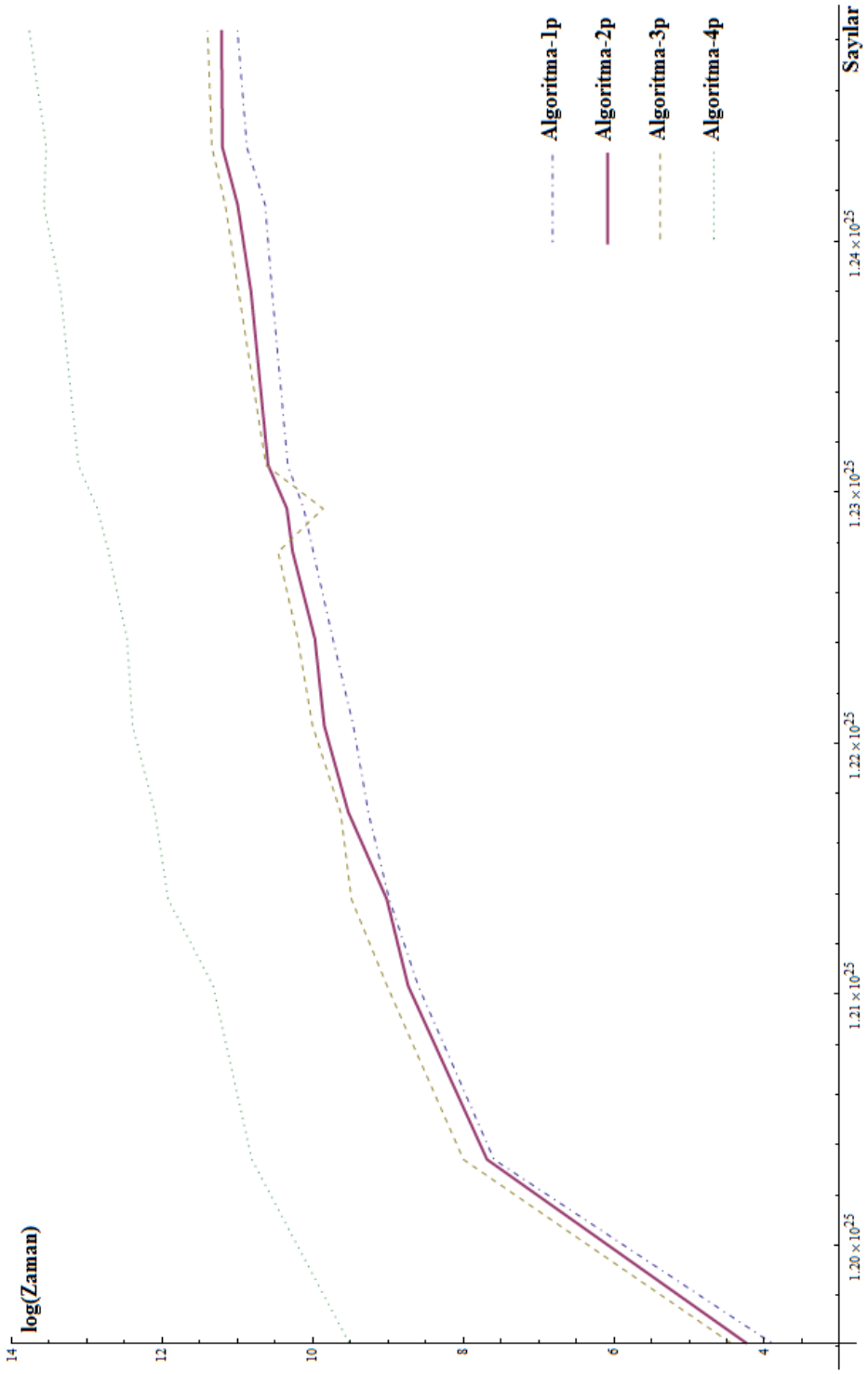
Şekil 7.4 Algoritma-3p'nin ardışıl ve paralel algoritmalarının karşılaştırılması

Çizelge 7.5 Algoritma-4p'de kullanılan sayılar ve çözüm zamanları

Kullanılan Sayılar			Paralel		Ardışıl
n = p.q	**p**	**q**	**Zaman(ms)**	**Süreç**	**Zaman(ms)**
11961099199121134769563567	**3456754259947**	**3460211024461**	13640	0	23473
12034074185291329380242507	**3456754259947**	**3481321864481**	49585	1	57821
12103209270531750431361871	**3456754259947**	**3501321864493**	82986	1	151027
12137776813089739380242507	**3456754259947**	**3511321864481**	150988	1	257462
12172344355848220076200069	**3456754259947**	**3521321864527**	179587	1	279724
12206911898240284820603249	**3456754259947**	**3531321864467**	241667	0	368187
12241479441199257263637737	**3456754259947**	**3541321864571**	260078	1	395982
12276046983494532888762401	**3456754259947**	**3551321864483**	332445	0	487315
12293330754725132803563461	**3456754259947**	**3556321864463**	385567	1	507702
12310614526211532533600599	**3456754259947**	**3561321864517**	496776	1	749876
12379749611486521127319433	**3456754259947**	**3581321864539**	628998	0	829519
12414317153837104820603249	**3456754259947**	**3591321864467**	786544	0	1117200
12437131731973495546162931	**3456754259947**	**3597921864473**	760700	1	1159013
12483452239008390786523673	**3456754259947**	**3611321864459**	952345	1	1543519



Şekil 7.5 Algoritma-4p'nin ardışıl ve paralel algoritmalarının karşılaştırılması



Şekil 7.6 Önerilen paralel algoritmalarının aralarında karşılaştırılması

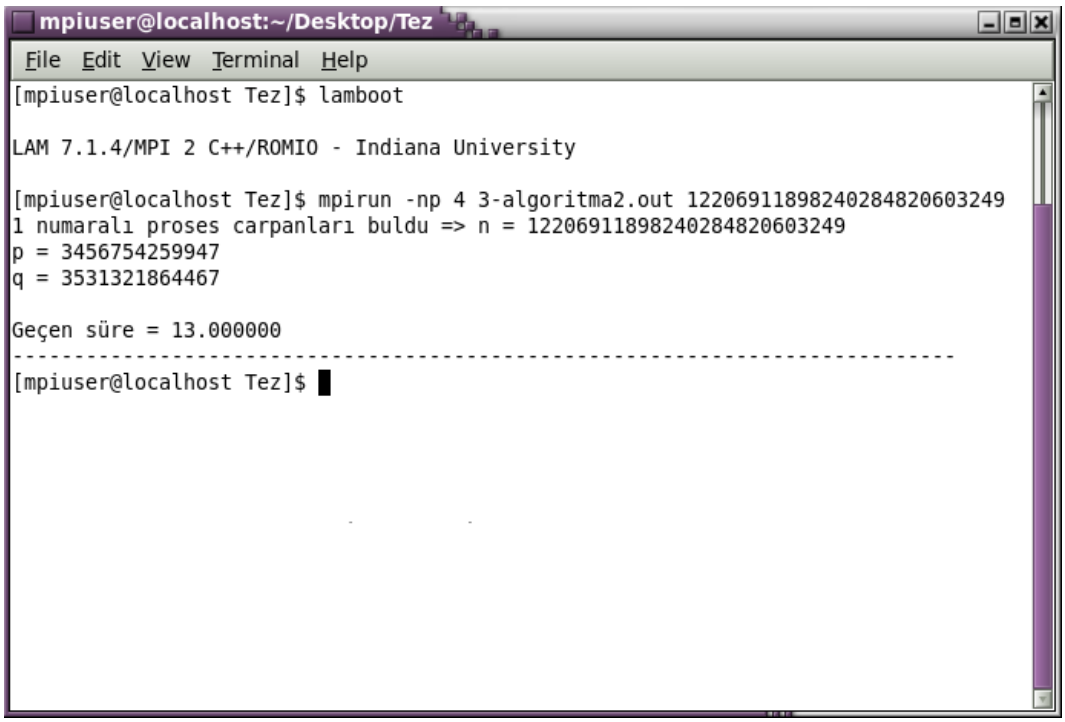
Algoritmaların performans kriterleri, zaman, bellek gereksinimi ve işlem sayısıdır. Bu çalışmada çarpanlara ayırma algoritmalarının performanslarını ölçmek için kriterler içerisindeki zaman kavramı yalnız ele alınmıştır.

Programların içerisindeki “difftime” fonksiyonunun yardımıyla sayının çarpanlarını bulmak için geçen süre hesaplanmaktadır

Grafiklerde de görüldüğü gibi Fermat’ın çarpanlara ayırma yönteminde sayı büyüdükçe çarpanların bulunma süresi çok fazla artmaktadır. Ama önerilen paralel programların çalışma süreleri Fermat’ın çarpanlara ayırma yönteminden oldukça verimlidir. “u” parametresine bağlı algoritma diğerlerine göre biraz daha yavaş sonucu elde ediyor olsa bile daha verimlidir.

Sayıların karakteristiğine göre algoritmaların hızlı çözümleri değişebilir, yani “s” parametresine bağlı algoritma her zaman daha verimlidir diyemeyiz. Şekil 7.4de de görüldüğü gibi “l” parametresine bağlı algoritma dalgalanma yaşamıştır.

Var olan süreç sayısı arttırıldığı takdirde paralel programlar daha büyük sayılarda da verimli bir şekilde çalışacaktır.



Şekil 7.7 Program çalışma örneği

8. SONUÇ

Böylece bu tezde günümüzün veri şifreleme algoritmaları için çok büyük öneme sahip olan tamsayıların çarpanlara ayrılması ile ilgili yöntemler detaylı olarak incelenmiş, paralel programlama hakkında ayrıntılı bilgi verilmiştir.

Kullanılan GMP ve MPI kütüphanesi detaylı olarak incelenmiş, önerilen yöntem ve algoritmalarda yeni parametreler tanımlanmış ve bu parametrelerin daha küçük değerler aldığı izlenmiştir ve böylece de bu parametreleri belirlemek için daha az zaman gerektiği gösterilmiştir.

Tanımlanmış parametrelerin hesaplanmasına dayanan yeni algoritmalar anlatılan kütüphaneleri kullanılarak C programlama dilinde kodlanarak hesaplama denemeleri yapılmıştır. Sonuçlar grafik ve tablo olarak sunulmuş, algoritmaların verimli olduğu gösterilmiştir.

Gelecekte önerilen yöntemler geliştirilerek süper bilgisayarlar için tasarlanması ve daha detaylı hesaplama denemelerinin yapılması düşünülmektedir.

KAYNAKLAR DİZİNİ

Arjen K. L., 2000, Integer Factoring, Designs, Codes and Cryptography, Kluwer Academic Publishers, 19:101–128pp.

Brent, Richard P., 1990, Parallel Algorithms for Integer Factorization, Computer Sience Laboratory, Australian National Universite Canbera, New York, 12pp.

Caldwell, C. K., 1994, Finding Primes and Proving Primality, The Prime Pages.

Carriero, N., 1992, How To Write Parallel Programs A First Course, MIT Press, USA, 250p.

Cormen, T.H., Leiserson, C. E., Rivest, R. L. and Stein, C., 2002, Introduction to Algorithms, McGraw – Hill, MIT Press, England, 1312p.

Culler, D. and Pal Singh, J., 1997, Parallel Computer Architecture A Hardware/Software Approach, Morgan Kaufmann Publishers, USA, 1056p.

El-Rewini, H. and Abd-El-Barr, M., 2005, Advanced Computer Architecture and Parallel Processing - A John Wiley & Sons, Inc Publicatıon, Canada, 288p.

Giblin, P., 1993, Primes and Programming-An Introduction to Number Theory with Computing, Cambridge University Press, Cambridge, 252p.

Herlihy, M., 2008, The Art of Multiprocessor Programming, Morgan Kaufmann Publishers, USA, 528p.

Kanmaz, Abbas A., 2003, Paralel Hesaplama Kütüphaneleri (Mpi, Pvm) ve Linux Ağında Çalıştırılmaları, İstanbul Teknik Üniversitesi Bilişim Enstitüsü, Adana.

Keskin, S., Berberler, M.E. ve Nuriyev, U.G., 2011, "Faktöriyel Yardımıyla Çarpanlara Ayırma Üzerine", 6. Ankara Matematik Günleri konferansı bildiri özetleri kitapçığı, Ankara, 151s (Baskıda).

Keskin, S., Nuriyeva, F. and Nuriyev, U.G., 2011 "On Parallelization of Prıme Factorizatıon Algorithms", The 4th Congress of the Turkic World Mathematical Society (TWMS), Azerbaijan (in press).

Kutucu, H. ve Nuriyeva, F., 2008, "Tamsayıların Çarpanlara Ayrılması Algoritmaları Üzerine", XXI. Ulusal Matematik Sempozyumu, Koç Üniversitesi, İstanbul.

KAYNAKLAR DİZİNİ (devam)

Leighton, F., 1992, Introduction To Parallel Algorithms And Architectures, Morgan Kaufmann Publishers, California, 831p.

Levitin, A., 2006, Introduction to The Design & Analysis of Algorithms, Addison Wesley, USA, 592p.

Lin, C. and Snyder, L., 2008, Principles of Parallel Programming, Addison Wesley, USA, 352p.

Mattson, T., Sanders, B. and Massingill B., 2004, Patterns for Parallel Programming, Addison Wesley, USA, 384p.

McConnell, J.J., 2001, Analysis of Algorithms An Active Learning Approach, Jones & Bartlett Publishers, USA.

Miller, R., and Boxer, L., 2006, Algorithms Sequential & Parallel A Unified Approach, Prentice Hall, New Jersey.

Nabiyev, V. V., 2007, Algoritmalar. Teoriden Uygulamalara, Seçkin Yayıncılık, Ankara, 799s.

Pacheco, P., 1996, Parallel Programming with MPI, Morgan Kaufmann Publishers, USA, 500p.

Parhami, B., 2002, Introduction To Parallel Processing - Algorithms And Architectures, Kluwer Academic Publishers, New York, 556p.

Pomerance, C., Smith, J. W. and Wagstaff, S. S., 1984, New ideas for factoring large integers, Advances in Cryptology, Plenum Press, New York.

Rauber, T., 2010, Parallel Programming: for Multicore and Cluster Systems, Springer, Heidelberg, 450p.

Riesel, H., 1985, Prime Numbers and Computer Methods for Factorization, Progress in Mathematics, Birkhäuser, 480p.

Samedov, R., 2002, Paralel (Çok İşlemcili ve Çok Bilgisayarlı) Sistemler, Pamukkale Üniversitesi Yayınları, Denizli.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J., 1998, MPI-The Complete Reference Volume 1, The MPI Core Second Edition, MIT press, England.

Xavier, C. and Iyengar, S. S., 1998 Introduction To Parallel Algorithms, John Wiley & Sons Inc, USA, 384p.

KAYNAKLAR DİZİNİ (devam)

Barney, B., Introduction to Parallel Computing, http://www.llnl.gov/computing/tutorials/parallel_comp/ (Erişim Tarihi 12 Nisan 2011)

Barney, B., Message Passing Interface (MPI), http://www.llnl.gov/computing/tutorials/mpi/ (Erişim Tarihi 12 Nisan 2011)

Granlund, T., The GNU MP Bignum Library, http://gmplib.org/ (Erişim Tarihi 24 Mayıs 2011)

Matthews K., Number Theory Web, http://www.numbertheory.org/ntw/web.html (Erişim Tarihi 1 Haziran 2011)

Moody, A., MPI at LLNL, https://computing.llnl.gov/mpi/, (Erişim Tarihi 1 Haziran 2011)

NZMATH development group, NZMATH, http://tnt.math.se.tmu.ac.jp/nzmath/ (Erişim Tarihi 1 Haziran 2011)

Stein, W., FLINT: Fast Library for Number Theory, http://www.flintlib.org/ (Erişim Tarihi 1 Haziran 2011)

Trustees of Indiana University, LAM/MPI Parallel Computing, http://www.lam-mpi.org (Erişim Tarihi 12 Nisan 2011)

ÖZGEÇMİŞ

KESKİN, Selçuk, Mustafa oğlu, 24/01/1986 tarihinde Kocaeli'nde doğdu. Osmangazi İlköğretim Okulu'nda okudu. 2000 yılında Kocaeli Anadolu Lisesi'ne başladı ve 2004 yılında bu liseyi bitirdi. Aynı yıl Ege Üniversitesi Fen Fakültesi Matematik Bölümüne girdi.

3. sınıfta Bilgisayar Bilimleri opsiyonunu seçti ve 2009 yılında mezun oldu. Aynı yıl İstanbul Teknik Üniversitesi'nin Yüksek Başarımlı Hesaplama ve Paralel Programlama Yaz Okulu'na katıldı ve başarıyla tamamladı. 2009 yılında E.Ü. Fen Bilimleri Enstitüsü Matematik Anabilim dalı Bilgisayar Bilimleri bilim dalında Yüksek Lisans eğitimine başladı. 2009-Eylül ile 2011-Şubat tarihleri arasında E.Ü. Ege Meslek Yüksekokulu'nda Öğretim Elemanı olarak görev aldı.

Makaleleri:

Keskin, S., Berberler, M.E., Nuriyev, U.G., 2011, "Faktöriyel Yardımıyla Çarpanlara Ayırma Üzerine", 6. Ankara Matematik Günleri konferansı bildiri özetleri kitapçığı, s. 151, Hacettepe Üniversitesi.

Keskin, S., Nuriyeva, F., Nuriyev, U.G., 2011 "On Parallelization of Prıme Factorizatıon Algorithms", The 4th Congress of the Turkic World Mathematical Society (TWMS).(kabul edilmiş)

EK 1. PROGRAM KODLARI

EK 1.1 Basit Bölme Yardımı İle Çarpanlara Ayırma

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, adet, ilk, son;

  mpf_t nf, mf;

  int procnumber, proc_id;
  unsigned long adetl;
  time_t now,end;

  time(&now);

  // kullanılacak değişkenler için bellekten yer alıyoruz
  mpz_init_set_str(n,argv[1],10);
  mpz_init(adet);
  mpz_init(ilk);
  mpz_init(son);

  mpf_init(nf);
  mpf_init(mf);

  MPI_Init(&argc, &argv);     // paralel ortamın başlatılması
  MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
  MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

// işlemi sadece seçtiğimiz, iletişimi sağlayacak proses e yaptırıyoruz.
  if(proc_id==0)
{
    //adeti bulma
    mpf_set_z(nf,n);
```

```
   mpf_div_ui(mf,nf,procnumber-1);
   mpf_ceil(mf,mf);
   mpz_set_f(adet,mf);

   gmp_printf("n == %Zd parça adeti == %Zd\n",n,adet);

   adetl = mpz_get_ui(adet);  // bulduğumuz parça sayısını aktarabilmek için MPI kütüphanesinin tanıyabileceği long tipine dönüştürüyoruz.
  }

  MPI_Bcast(&adetl,1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD); // belirlediğimiz proses (0 numaralı) bulmuş olduğu değeri bütün proseslere aktarıyor Bu seviyeye gelen diğer prosesler bilgi gelene kadar bekliyorlar.

  if(proc_id!=0)
  {
   // maalesef gmp kütüphanesinin en kötü yanı birden fazla aritmetiksel işlemi aynı anda yapamamamız. ilk=(proc_id-1)*adetl+2 gibi bir işlemi adım adım yapmamız gerekiyor

   mpz_set_ui(ilk,adetl);        // öncelikle ilk değişkenine adetl değerini atadık
   mpz_set_ui(son,adetl);

   mpz_mul_ui(ilk,ilk,proc_id-1);     // daha sonra adetl değeri ile proses numarasını çarptık.
   mpz_mul_ui(son,son,proc_id);

   mpz_add_ui(ilk,ilk,2);                  // bulduğumuz bu değere 2 sayısını ekleyerek çarpanları 2 den başlatmış oluyoruz
   mpz_add_ui(son,son,2);

   if(proc_id!=1)mpz_add_ui(ilk,ilk,1);        // bulduğumuz son değer ile bir sonraki prosesin ilk değeri aynı olmaması için 1 ekledik

   while(mpz_cmp(ilk,son)<=0)
   {
    if(mpz_divisible_p(n,ilk)!=0)         // döngüdeki değişkenin n sayısını bölüp bölmediği kontrol edilir
```

```
        gmp_printf("proses%3d => %Zd sayısının bir böleni :
%Zd\n",proc_id,n,ilk);
     mpz_add_ui(ilk,ilk,1);
   }
  }

  MPI_Barrier(MPI_COMM_WORLD);    // diğer işlemciler işini bitirene kadar
işi biten işlemcileri bu seviyede bekletiyoruz.
  time(&end);
  if(proc_id==0) printf("Geçen süre = %f\n",difftime(end,now));

  MPI_Finalize();       // paralel ortamın sonlandırılması

  mpz_clear(n);         // bellekten alınan yerler boşaltılır
  mpz_clear(adet);
  mpz_clear(ilk);
  mpz_clear(son);

  mpf_clear(nf);
  mpf_clear(mf);

}
```

EK 1.2 Fermat'ın Çarpanlara Ayırma Yöntemi

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, nkontrol, r, rp, z, p, q;

  int procnumber, proc_id;

  time_t now,end;

  time(&now);
```

```
  // kullanılacak değişkenler için bellekten yer alıyoruz
  mpz_init_set_str(n,argv[1],10);
  mpz_init(nkontrol);
  mpz_init(r);
  mpz_init(rp);
  mpz_init(z);
  mpz_init(p);
  mpz_init(q);

  MPI_Init(&argc, &argv); // paralel ortamın başlatılması
  MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
  MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

  mpz_set_ui(r,proc_id);
  mpz_add_ui(r,r,1);

bas:
  mpz_mul(rp,r,r);              // n + r^2 ifadesini hesaplanmak için önce r^2 yi bulduk
  mpz_add(nkontrol,n,rp);   // daha sonra n sayısını ekledik.
  if(mpz_perfect_square_p(nkontrol)!=0)   // tamkare olup olmadığını kontrol ettik
  {
    mpz_sqrt(z,nkontrol);       // ifade bir tam kare olduğu için fonksiyon doğru
    mpz_add(p,z,r);               // sonuç
    mpz_sub(q,z,r);
    if( mpz_cmp_ui(p,1)!=0 || mpz_cmp_ui(q,1)!=0 )  goto son;     // 1 çarpanlarını
vermemesi için kontrol ediyoruz
  }
  mpz_add_ui(r,r,procnumber);   // artış sayısını proses sayısına göre yapıyoruz
  goto bas;

son:
  time(&end);
  printf("Geçen süre = %f\n",difftime(end,now));
  gmp_printf("p = %Zd   q = %Zd    r = %Zd z = %Zd        id = %d\n", p, q, r, z,
proc_id);     // n sayısının p ve q çarpanlarını yazdırıyoruz
  MPI_Abort(MPI_COMM_WORLD,0); // paralel ortamın sonlandırılması
```

```
  mpz_clear(n); // bellekten alınan yerler boşaltılır
  mpz_clear(nkontrol);
  mpz_clear(r);
  mpz_clear(rp);
  mpz_clear(z);

}
```

EK 1.3 “*s*” Parametresine Dayalı Birinci Yöntem

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, ilk, son, m, j, t, S, k, p, q;

  mpf_t nf, mf;

  int procnumber, proc_id;

  time_t now,end;

  time(&now);

  // kullanılacak değişkenler için bellekten yer alıyoruz
  mpz_init_set_str(n,argv[1],10);
  mpz_init(ilk);
  mpz_init(son);
  mpz_init(m);
  mpz_init(j);
  mpz_init(t);
  mpz_init(S);
  mpz_init(k);
  mpz_init(p);
  mpz_init(q);
```

```
  mpf_init(nf);
  mpf_init(mf);

  MPI_Init(&argc, &argv); // paralel ortamın başlatılması
  MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
  MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

  mpf_set_z(nf,n);
  mpf_sqrt(mf,nf);
  mpf_ceil(mf,mf);
  mpz_set_f(m,mf);

  mpz_set_ui(j,proc_id);
  mpz_add_ui(j,j,1);

bas:
  mpz_add(t,m,j);         // t = m + j_i ifadesini hesapladık
  mpz_mul(S,t,t);         // S = t^2 - n ifadesini için önce t^2 yi bulduk
  mpz_sub(S,S,n);         // daha sonra bu ifadeden n sayısını çıkardık

  if(mpz_perfect_square_p(S)!=0)   // Bulduğumuz bu S değerinin tamkare olup
              goto son;            // olmadığını kontrol ediyoruz
  mpz_add_ui(j,j,procnumber);
  goto bas;

son:
  mpz_sqrt(k,S);
  mpz_add(q,t,k);
  mpz_sub(p,t,k);

  if(mpz_cmp_ui(p,1)<=0||mpz_cmp_ui(q,1)<=0)
  {
    mpz_add_ui(j,j,procnumber);
    goto bas;
  }

  gmp_printf("%d numaralı proses carpanları buldu => n = %Zd\np = %Zd\nq =
%Zd\n\n",proc_id,n,p,q);
```

```
  time(&end);
  printf("Geçen süre = %f\n",difftime(end,now));

  MPI_Abort(MPI_COMM_WORLD,0); // paralel ortamın sonlandırılması

  mpz_clear(n); // bellekten alınan yerler boşaltılır
  mpz_clear(m);
  mpz_clear(ilk);
  mpz_clear(son);
  mpz_clear(j);
  mpz_clear(t);
  mpz_clear(S);
  mpz_clear(k);
  mpz_clear(p);
  mpz_clear(q);

  mpf_clear(nf);
  mpf_clear(mf);

}
```

EK 1.4 “*s*” Parametresine Dayalı İkinci Yöntem

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, dn, ilk, son, s, m, j, t, A, k, p, q, l1, l2;

  mpf_t nf, mf;

  int procnumber, proc_id;

  time_t now,end;
```

```
time(&now);

// kullanılacak değişkenler için bellekten yer alıyoruz
mpz_init_set_str(n,argv[1],10);
mpz_init(dn);
mpz_init(ilk);
mpz_init(son);
mpz_init(s);
mpz_init(m);
mpz_init(j);
mpz_init(t);
mpz_init(A);
mpz_init(k);
mpz_init(p);
mpz_init(q);
mpz_init(l1);
mpz_init(l2);

mpf_init(nf);
mpf_init(mf);

MPI_Init(&argc, &argv); // paralel ortamın başlatılması
MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

mpf_set_z(nf,n);
mpf_sqrt(mf,nf);
mpf_ceil(mf,mf);
mpz_set_f(m,mf);
mpz_mul(t,m,m);
mpz_sub(dn,n,t);
mpz_set_ui(s,proc_id);

bas:
mpz_mul(t,s,s);        //  A = s^2 + 2sm - Δn  ifadesini hesaplayabilmek için daha
mpz_mul(k,s,m);        // önceden de bahsedildiği gibi işlemleri adım adım
mpz_mul_ui(k,k,2);     // yapmamız gerekiyor. Geçici değişkenler kullanmak
mpz_add(A,k,t);        // işimizi kolaylaştırabilir
```

```
  mpz_sub(A,A,dn);

  if(mpz_perfect_square_p(A)!=0)
                goto son;
  mpz_add_ui(s,s,procnumber);
  goto bas;

son:
  mpz_sqrt(k,A);          // l = -s ± √(s^2 + 2sm - Δn) ifadesinde iki tane l değeri
  mpz_neg(l1,s);          // elde etmiş oluyoruz. Her iki kök de bize sonucu
  mpz_set(l2,l1);         // verecektir. Sadece p ve q değerlerinin yeri değişmiş
  mpz_add(l1,l1,k);       // oluyor
  mpz_sub(l2,l2,k);

  mpz_sub(p,m,l1);
  mpz_mul_ui(q,s,2);
  mpz_add(q,q,l1);
  mpz_add(q,q,m);

  if(mpz_cmp_ui(p,1)<=0||mpz_cmp_ui(q,1)<=0)
  {
   mpz_add_ui(s,s,procnumber);
   goto bas;
  }

  gmp_printf("%d numaralı proses carpanları buldu => n = %Zd\np = %Zd\nq =
%Zd\n\n",proc_id,n,p,q);

  mpz_sub(p,m,l2);
  mpz_mul_ui(q,s,2);
  mpz_add(q,q,l2);
  mpz_add(q,q,m);

  gmp_printf("%d numaralı proses carpanları buldu => n = %Zd\np = %Zd\nq =
%Zd\n\n",proc_id,n,p,q);

  time(&end);
  printf("Geçen süre = %f\n",difftime(end,now));
```

```
  MPI_Abort(MPI_COMM_WORLD,0); // paralel ortamın sonlandırılması

  mpz_clear(n); // bellekten alınan yerler boşaltılır
  mpz_clear(dn);
  mpz_clear(ilk);
  mpz_clear(son);
  mpz_clear(s);
  mpz_clear(m);
  mpz_clear(j);
  mpz_clear(t);
  mpz_clear(A);
  mpz_clear(k);
  mpz_clear(p);
  mpz_clear(q);
  mpz_clear(l1);
  mpz_clear(l2);

  mpf_clear(nf);
  mpf_clear(mf);

}
```

EK 1.5 “*l*” Parametresine Dayalı Yöntem

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, dn, ilk, son, m, j, t, s, l, p, q;

  mpf_t nf, mf;

  int procnumber, proc_id;

  time_t now,end;
```

```
time(&now);

// kullanılacak değişkenler için bellekten yer alıyoruz
mpz_init_set_str(n,argv[1],10);
mpz_init(dn);
mpz_init(ilk);
mpz_init(son);
mpz_init(m);
mpz_init(j);
mpz_init(t);
mpz_init(s);
mpz_init(l);
mpz_init(p);
mpz_init(q);

mpf_init(nf);
mpf_init(mf);

MPI_Init(&argc, &argv); // paralel ortamın başlatılması
MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

mpf_set_z(nf,n);
mpf_sqrt(mf,nf);
mpf_ceil(mf,mf);
mpz_set_f(m,mf);
mpz_mul(t,m,m);
mpz_sub(dn,n,t);

mpz_mul_ui(l,m,2);      // l_i = ⌊√(2m−n)⌋ + i − 1 ifadesini bulmak için
mpz_sub(l,n,l);         // önce karekökün içini hesapladık. Karakökten
mpf_set_z(mf,l);        // sağlıklı bir sonuç elde edebilmek için ifadeyi önce
mpf_sqrt(nf,mf);        // reel sayıya çevirdik saha sonra kare kökü alındı.
mpf_ceil(mf,nf);        // reel kısım tam sayıya çevrilerek yuvarlama  yapıl-
mpz_set_f(l,mf);        // mış oldu.

mpz_add_ui(l,l,proc_id);
```

```
bas:
  mpz_mul(t,l,l);       // s = (l^2 + Δn) / 2(m - l) ifadesini hesaplamak için paydanın
  mpz_add(t,t,dn);      // payı tam bölüp bölmediğini kontrol etmemiz gerekiyor
  mpz_sub(j,m,l);
  mpz_mul_ui(j,j,2);

  if(mpz_divisible_p(t,j)!=0)     // tam bölme işlemi gerçekleştiği takdirde sonuca
                goto son;         // erişmiş oluyoruz
  mpz_add_ui(l,l,procnumber);
  goto bas;

son:
  mpz_divexact(s,t,j);
  mpz_sub(p,m,l);
  mpz_mul_ui(q,s,2);
  mpz_add(q,q,l);
  mpz_add(q,q,m);

  if(mpz_cmp_ui(p,1)<=0||mpz_cmp_ui(q,1)<=0)
  {
    mpz_add_ui(l,l,procnumber);
    goto bas;
  }

  gmp_printf("%d numaralı proses carpanları buldu => n = %Zd\np = %Zd\nq =
%Zd\n\n",proc_id,n,p,q);

  time(&end);
  printf("Geçen süre = %f\n",difftime(end,now));

  MPI_Abort(MPI_COMM_WORLD,0); // paralel ortamın sonlandırılması

  mpz_clear(n); // bellekten alınan yerler boşaltılır
  mpz_clear(dn);
  mpz_clear(ilk);
  mpz_clear(son);
  mpz_clear(m);
  mpz_clear(j);
```

```
  mpz_clear(t);
  mpz_clear(s);
  mpz_clear(l);
  mpz_clear(p);
  mpz_clear(q);

  mpf_clear(nf);
  mpf_clear(mf);

}
```

EK 1.6 “*u*” Parametresine Dayalı Yöntem

```
#include<stdio.h>
#include<gmp.h>
#include<mpi.h>
#include<time.h>

int main(int argc, char *argv[])
{
  mpz_t n, dn, ilk, son, m, j, t, s1, s2, l, p, q, u, A;

  mpf_t nf, mf;

  int procnumber, proc_id;

  time_t now,end;

  time(&now);

  // kullanılacak değişkenler için bellekten yer alıyoruz
  mpz_init_set_str(n,argv[1],10);
  mpz_init(dn);
  mpz_init(ilk);
  mpz_init(son);
  mpz_init(m);
  mpz_init(j);
  mpz_init(t);
  mpz_init(s1);
```

```
mpz_init(s2);
mpz_init(l);
mpz_init(u);
mpz_init(A);
mpz_init(p);
mpz_init(q);

mpf_init(nf);
mpf_init(mf);

MPI_Init(&argc, &argv); // paralel ortamın başlatılması
MPI_Comm_size(MPI_COMM_WORLD,&procnumber);
MPI_Comm_rank(MPI_COMM_WORLD,&proc_id);

mpf_set_z(nf,n);
mpf_sqrt(mf,nf);
mpf_ceil(mf,mf);
mpz_set_f(m,mf);
mpz_mul(t,m,m);
mpz_sub(dn,n,t);

gmp_printf("n == %Zd    m == %Zd    t == %Zd\n",n,m,t);

mpz_set_ui(u,proc_id);

bas:
mpz_mul_ui(j,dn,3);         //  A = (3Δn + m^2) + u^2 − 4mu  ifadesini adım adım
mpz_add(j,j,t);             // hesaplıyoruz.
mpz_addmul(j,u,u);
mpz_mul_ui(p,m,4);
mpz_mul(p,p,u);
mpz_sub(A,j,p);

if(mpz_perfect_square_p(A)!=0)
        goto son;
mpz_add_ui(u,u,procnumber);
goto bas;
```

```
son:
 mpz_mul_ui(t,u,2);            // s = 1/3 ( (m - 2u) ± sqrt((3Δn + m^2) + u^2 - 4mu) )
 mpz_sub(j,m,t);               // ifadesini hesaplıyoruz
 mpz_sqrt(t,A);
 mpz_add(s1,j,t);
 mpz_sub(s2,j,t);

 if(mpz_divisible_ui_p(s1,3)!=0)      // bulduğumuz ifade 3 e bölünüyor mu ?
 {
            mpz_divexact_ui(s1,s1,3);
            mpz_sub(p,m,u);
            mpz_sub(p,p,s1);
            mpz_mul_ui(q,s1,3);
            mpz_add(q,q,m);
            mpz_add(q,q,u);

            if(mpz_cmp_ui(p,1)<=0||mpz_cmp_ui(q,1)<=0)
            {
             mpz_add_ui(l,l,procnumber);
             goto bas;

            }

            gmp_printf("%d numaralı proses carpanları buldu s1 => n = %Zd\np
= %Zd\nq = %Zd\n\n",proc_id,n,p,q);
 }else
            gmp_printf("s1, 3 e tam bölünmüyor\n");

 if(mpz_divisible_ui_p(s2,3)!=0)
 {
      mpz_divexact_ui(s2,s2,3);
      mpz_sub(p,m,u);
      mpz_sub(p,p,s2);
      mpz_mul_ui(q,s2,3);
      mpz_add(q,q,m);
      mpz_add(q,q,u);

      if(mpz_cmp_ui(p,1)<=0||mpz_cmp_ui(q,1)<=0)
```

```
        {
            mpz_add_ui(l,l,procnumber);
            goto bas;

        }

        gmp_printf("%d numaralı proses carpanları buldu s2 => n = %Zd\np =
%Zd\nq = %Zd\n\n",proc_id,n,p,q);
  }else
        gmp_printf("s1, 3 e tam bölünmüyor\n");

  time(&end);
  printf("Geçen süre = %f\n",difftime(end,now));

  MPI_Abort(MPI_COMM_WORLD,0); // paralel ortamın sonlandırılması

  mpz_clear(n); // bellekten alınan yerler boşaltılır
  mpz_clear(dn);
  mpz_clear(ilk);
  mpz_clear(son);
  mpz_clear(m);
  mpz_clear(j);
  mpz_clear(t);
  mpz_clear(s1);
  mpz_clear(s2);
  mpz_clear(l);
  mpz_clear(u);
  mpz_clear(A);
  mpz_clear(p);
  mpz_clear(q);

  mpf_clear(nf);
  mpf_clear(mf);

}
```

EK 2. YABANCI TERİMLER SÖZLÜĞÜ

Associative	Çağrışımlı
Asynchronous	Asenkron
Conveyor	Konveyör
Communicator	İletişimci
Composite Number	Bileşik Sayı
Computing Environment	Hesaplama Ortamı
Computing Network	Hesaplama Ağı
Cryptanalysis	Kriptanaliz
Distributed Memory	Dağıtık Bellek
Hard number	Zor sayı
Immediate	Derhal
Incidence	Bitişiklik
Initialize	İlk Atama
Interface	Arayüz
Optimal	Optimum
Precision	Duyarlılık
Prefix	Ön Takı
Prime Number	Asal Sayı
Rank	Sınıf
Real-time Problem	Gerçek Zaman Problemi
Reliability	Güvenilirlik
Sequential	Sıralı
Synchronous	Senkron
Truncate	Yuvarlamak
Upgradable	Geliştirilebilme
Weight Matrice	Ağırlık Matrisi