Yazılım Ürün Hattı Yaklaşımı ile Yüksek Başarımlı Uygulama Geliştirme

Developing High Performance Computing Applications by Software Product Lines

Oktay Özgün
Bilişim Ensititüsü, HBM Programı
İstanbul Teknik Üniversitesi, İstanbul
oktay@ieee.org

A. Şima Uyar
Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi, İstanbul
etaner@itu.edu.tr

Hasan Dağ
Enformasyon Teknolojileri Bölümü
Kadir Has Üniversitesi, İstanbul
hasan.dag@khas.edu.tr

Özet

Koşut programlamada kullanılan mevcut yazılım geliştirme araçlarının soyutlama seviyesi yeteri kadar yüksek değildir ve ilgili yazılım geliştime yöntemlerinde yeniden kullanım olanakları kısıtlıdır. Bu çalışmada yüksek başarımlı hesaplama alanındaki yazılım geliştirme süreçlerinde sistematik yeniden kullanımı artırarak, koşut programlama ile ilintili zorlukları azaltacak, yazılım ürün hattı yöntemine dayanan yeni bir yaklaşım önerilmektedir.

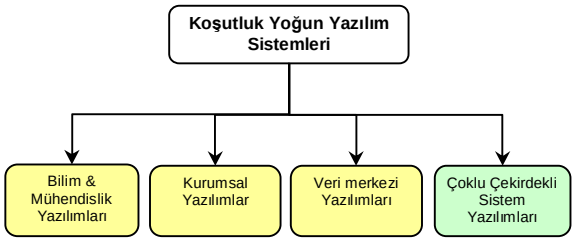
Abstract

Existing parallel programming software tools provide a poor layer of abstraction and the associated methodologies offer very limited reusability and modularity. This study proposes a new programming model based on software product line methodology that will increase systematic reuse and in turn, facilitate parallel programming in high performance computing space.

1. Giriş

Koşut programlama yaygın olarak üniversitelerde ve araştırma merkezlerinde karmaşık bilim ve mühendislik uygulamaları geliştirmede kullanılmaktadır. Kurumsal bilgi sistemlerinin ölçeğinin genişlemesi ve iş zekası uygulamalarının yaygınlaşması ile büyük ölçekli kurumlar da koşut programlamayı uygulamaya başlamıştır. Büyük veri merkezleri ile milyonlarca tüketici ve binlerce kuruma hizmet veren servis sağlayıcılar da yine yaygın şekilde koşut program-lamadan faydalanmaktadır. Son olarak, çoklu çekirdekli mikroişlemcilerin yaygınlaşması ile bu mimarinin bir gerekliliği olarak koşut programlamanın bu sistemler üzerindeki tüm yazılımların gerçeklenmesinde de kullanılması gereği doğmuştur.

Bilgi sistemlerinin yazılım tasarımlarında koşutlaş-tırmanın ön planda olduğu tüm sistemleri “Koşutluk Yoğun Yazılım Sistemleri ” olarak adlandırmanın doğru olacağını düşünüyoruz (Şekil 1):



Şekil 1 Koşutluk Yoğun Yazılım Sistemleri

Bahsi geçen ilk üç sistemde daha çok tasarlanan yazılımların gereği olarak koşutlaştırma uygulanırken, çoklu çekirdekli sistemlerde bu donanımın bir gereği olarak ortaya çıkmaktadır.

Buradan da anlaşılacağı üzere koşut yazılım geliştirme artık sadece seçkin akademik ve devlet destekli araştırma kurumlarında kullanılan bir yöntem değildir. Kurumsal, masaüstü ve İnternet uygulamaları geliştiren geniş bir kitle tarafından çok daha yaygın olarak kullanılmasının gereği ortaya çıkmıştır. Böyle bir resimde bilişim teknolojileri endüstrisi, geniş devlet fonları ile desteklenen ve yüksek başarımlı hesaplama dünyasında “kahraman programcı” olarak adlandırılan, yapısallıktan uzak yazılım geliştirme süreçlerini, büyük yazılım projelerinde tolere edemeyecektir. Diğer yandan, kullanılan araçlardaki yetersiz soyutlama seviyesi ve araç seçeneklerindeki fazlalıktan kaynak-lanan karmaşık durum bireysel gelişticiler için de idealden uzaktır.

Mattson ve Wrinn [1] bu durumu bilgisayar endüstri-sinin karşı karşıya olduğu “koşut programlama sorunu” olarak adlandırmış ve yüksek başarımlı hesaplama alanının yaklaşık olarak 30 yıllık tarihinin bu sorununun çözümünde dersler çıkarılacak bir

kaynak olduğunu vurgulamıştır. ABD'deki Yüksek Üretkenlik Hesaplama Merkezleri (High Productivity Computing Systems - HPCS) Programında yapılan çalışmalar 30 yıllık mazisi olan yüksek başarımlı hesaplama sistemlerindeki sorunun asıl kaynağının yazılım geliştirme araçlarında olduğunu ortaya koymuştur [2]. Bu alandaki araçlar is-tekleri karşılayamamaktadır. Sistemlerin genişleyen ölçeği, artan karmaşıklığı ve görevlerindeki hassasiyetle sorun daha da büyümektedir.

1990'lardan bu yana yüzlerce koşut programlama teknolojisi yaratılmış ve kullanıma sunulmuştur ancak en popüler olanları bile çok düşük kullanım oranına ulaşa-bilmiştir [1, 3]. Bu teknolojiler 3 ana grupta toplanabilir [4]:

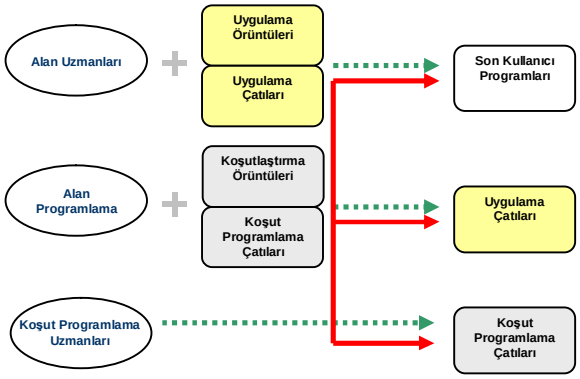
1. *Koşutzamanlı Diller:* Koşutluk özelliklerini dilin tasarımında taşırlar.
2. *Dil Eklentileri:* Sırasal dillere koşutluk eklentileri kazandırılmıştır.
3. *Çalıştırma Kütüphaneleri:* Bir sistem kütüphanesinde saklanan koşut yordamlara yüksek düzeyli arayüzler sağlanmıştır. Uzun yıllar süren ve daha çok "araç odaklı" olarak sürdürülen bu yoğun çalışmalara rağmen, programcıların çok küçük bir oranı koşut programlama yapmak-tadırlar [5].

Diğer yandan yaygın bilgi sistemlerinin (kurumsal, gömülü, masaüstü, mobil sistemler vs.) geliştirilmesinde çalışan milyonlarca yazılım geliştiricisi olduğunu görüyoruz. Bu bilgi sistemlerinin de çözülmesi gereken sorunları [6] olmasına rağmen, yazılım mühendisliğinin ilk olarak teleffuz edildiği 1968 yılındaki NATO Yazılım Konferansı'ndan [7] bu yana, ortaya koyduğu gelişmeleri etkin şekilde kullanarak, dayandıkları yazılımların geliştirme süreçlerinde daha sistematik (yüksek başarımlı hesaplamaya göre) yaklaşımlar barındırdığını söyleyebiliriz [8]. Yazılım mühendisliği 80'li ve 90'lı yıllardaki bireysel üretkenliği artırmaya yönelik yaklaşımlarını 2000'li yıllarla birlikte otomas-yona yönelterek, endüstriyel üretim sistemlerinin verim-lilik kazanımlarını yazılım geliştirme süreçlerine uygulamaya başlamıştır. Bileşen tabanlı üretim, otomasyonla emek yoğun görevlerin azaltılması, ürün hatlarının ve tedarik süreçlerinin kurulması, arayüzlerin somut-laştırılması, mimari ve süreçlerin standartlaşması bu yönde yapılan çalışmalardır [9]. Kısaca bunu "gümüş kurşun" yaklaşımlardan "alana özel" yaklaşımlara doğru bir yönelim olarak niteleyebiliriz.

Bu açıdan bakıldığında özellikle çoklu çekirdekli mikroişlemcilerin yaygınlaşması ile koşut programlama geliştirme süreçlerinde yazılım mühendisliğinin kazanımlarının tersi yönde ilerlemeler olduğunu görüyoruz. Soyutlama seviyelerinin düşmesi ve yeniden kullanım oranlarının azalması bu yöndeki iki temel gözlemdir.

Koşut programlama süreçlerinde yazılım mühendisliği yaklaşımlarının daha etkin şekilde kullanılmasının koşut programalamadaki sorunların çözümünde büyük katkılar sağlayacağını öngürüyoruz. Özellikle yüksek başarımlı hesaplama alanında ortaya konacak yeni yöntemlerin diğer koşutluk yoğun yazılım sistemlerinde de kullanılacağı kesindir.

Kuetzar ve arkadaşları koşut program geliştirme süreçlerindeki araç ve görev dağılımını Şekil 2'deki gibi betimlemişlerdir [10]: Kesikli çizgilerle belirtilen ayrış-mış süreç, alan uzmanlarının minimum programlama bilgisi ile koşut programlar geliştirebildiği ideal ortamı göstermektedir. Ancak maalesef günümüzde bundan uzağız.



Şekil 2 Koşut program geliştirme süreçlerindeki araçlar ve görev dağılımları [10]

Mevcut durumda yer alan ve iç içe geçmiş olan alan analizi, alan gerçekleştirmesi, uygulama geliştirme ve kurulum faaliyetlerinin sistematik şekilde ele alınması gerekmektedir. Yazılım ürün hattı yaklaşımı bu konuda ideal bir çözüm olacaktır ve yöntem gereği zaman içinde yukarıdaki ideal geliştirme yapısına doğru evrimleşecektir.

Buradan hareketle, bu çalışmada yüksek başarımlı hesaplama alanındaki yazılım geliştirme süreçlerinde sistematik yeniden kullanımı artırarak, koşut programlama ile ilintili zorlukları azaltacak, yazılım ürün hattı yöntemine dayanan yeni bir yaklaşım önerilmektedir.

2. Yazılım Ürün Hattı

SEI[1] yazılım ürün hattını şu şekilde tanımlamaktadır: "Belli bir alanın veya görevin ihtiyaçlarını yerine

[1] Software Engineering Institute, Carnegie Mellon University http://www.sei.cmu.edu

getiren ortak özellikleri paylaşan ve ortak temel varlıkları tanımlı şekillerde kullanarak geliştirilmiş yazılım yoğun sistem kümeleri" [11]. Yazılım ürün hattı yaklaşımı tanımdan da anlaşılacağı gibi ortak özellikler üzerine kurgulanmıştır ve değişkenlikleri sistemli şekilde yönetir. Yazılım ürün hattı yaklaşımında üç temel faaliyet vardır:

- *Temel varlık geliştirilmesi*, ürün hattının temel varlıklarının geliştirilmesi için yapılan faaliyetleri içerir. Temel varlıklara ek olarak, bu varlıkların ürün gerçeklemede ne şekilde kullanılacağını tanımlayan ürün planını çıktı olarak verir.
- *Ürün geliştirilmesi*, ürün planında tanımlandığı şekilde temel varlıklardan ürünlerin geliştirilmesini kapsar.
- *Yönetim* de bu süreçlerin teknik ve organizasyonel olarak idaresini içerir.

Yazılım ürün hatları aşağıda belirtilen sebeblerden ötürü yoğun ilgi görmektedir [9]:

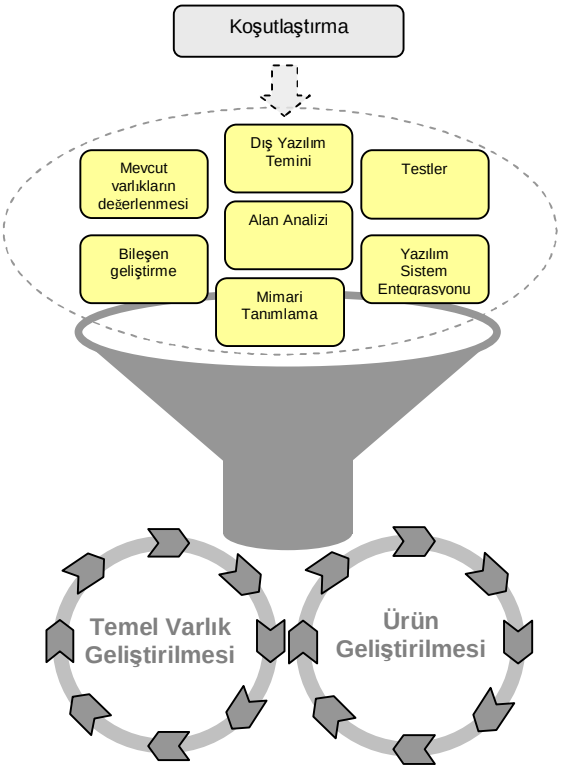
- Yazılım eserlerinin (artifacts) etkin şekilde yeniden kullanımı
- Onaylanmış referans mimarinin farklı ürünlerde kul-lanımı
- Ürünlerin farklı yanlarını gerçekleştirmek için odak-lanma
- Entegrasyonu test edilmiş bileşenler kullanıldığından yazılım entegrasyonunun kolay sağlanması
- Ürün planı sayesinde etkin iş gücü kullanımı
- Yazılım ürün kalitesinin sağlanması

3. Koşut Yazılım Ürün Hattı

Bu çalışmada SEI [11]'nin ürün hattı yöntemi yüksek başarımlı hesaplama alanına uyarlanacaktır. Bunun için temel varlık geliştirilmesi ve ürün geliştirilmesi aşa-malarında kullanılan yazılım mühendisliği çalışma alanları koşutlaştırma gereksinimlerine uygun olarak güncellenecektir (Şekil 3).

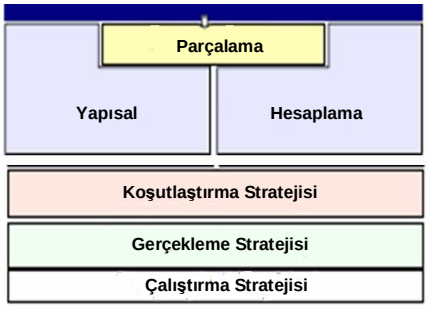
3.1. Mimari tanımlama

Bileşen stratejisi, bileşen arayüzleri, bileşen bağlantı şemaları yazılım ürün hattının referans mimarisini oluşturur. Mimari örüntüler (architectural patterns) yazılım mimarilerinin tanımlanmasında (ve yeniden kullanımında) sıkça kullanılmaktadır.



Şekil 3 Koşut yazılım ürün hattı için izlenecek yol

Kalifornia Üniverstitesi, Berkley 'de yer alan ParLab araştırma grubu koşut bir uygulamanın genel organizas-yonunu tanımlayan yapısal örüntüleri, hesaplama örüntülerini ve koşutlaştırma ile ilgili detayların tümünü kapsayan bir örüntü dili geliştirmiştir [12] (Şekil 4).



Şekil 4 ParLab örüntü dili [12]

Yazılım ürün hattımız için koşutlaştırma perspektifi ile iki temel bileşen tanımlanacaktır:

1. Koşutluk yoğun bileşenler
2. Sırasal (seri) bileşenler

Parlab örüntü dili temel varlıkları oluşturacak bileşenleri tanımlama ve yukarıdaki ayrımı yapma işlerini kolay-laştıracaktır.

3.2. Bileşen geliştirme

Bileşen geliştirme, adından da anlaşılacağı gibi belirli bir işlevi yerine getirecek bileşenleri yazılım mimarisine uygun olarak üretmeyi sağlar [11]. Temel varlık küme-sinin ana bileşenlerini, bu bileşenler ve bunlara ait arayüz belirtimleri, değişkenlik yönetim metodları, test dokumanları ve bu bileşenlerin entegrasyonunun nasıl yapılacağını tanımlayan referans mimari oluşturur.

Koşut yoğun bileşenlerin geliştirilmesi seri bileşenlere göre farklılık gösterecektir. Bu amaçla, Matson, Sanders ve Massingill'in çalışmalarına [13] da-yanan Parlab örüntü dilinin ilk üç katmanını kullanarak bu bileşenlere koşutluk özellikleri kazandırılacaktır.

3.3. Mevcut varlıkların değerlendirilmesi

Mevcut varlıkların değerlendirilmesi eski bir sistemin ortaya çıkış amacı dışında kullanılmasını kapsar. Olgun bir alanda, ihtiyaç duyulan çözümlerin çoğu mevcut uygulamalarda vardır. Bir sistemi oluşturan üç ana yazılım kategorisi tanımlanabilir [14]:

- Hizmet bileşenleri (ortalamada uygulamanın %20'lik bölümü)
- Alana özel bileşenler (ortalamada uygulamanın %65'lik bölümü)
- Uygulamaya özel bileşenler (ortalamada uygulamanın %15'lik bölümü)

Dolayısıyla buradan uygulama ailelerinde ortalamada %85'lik ortak bileşenler olacağı sonucu çıkar.

Parlab örüntü dilinin hesaplama örüntüleri özel bir çözüm için yenilenir uygulama çözümleri önerir ve bunları kısaca 'cüce' (dwarf) olarak adlandırır. Oniki cüce belirlenmiştir şu ana kadar: yoğun doğrusal cebir (dense linear algebra), seyrek doğrusal cebir (sparse linear algebra), tayf yöntemleri (spectral methods), n-cisim yöntemleri (n-body methods), yapısal şebekeler (structured grids), yapısal olmayan şebekeler (unstructured grids), birleştirilmeli mantık (combi-national logic), çizge tarama (graph traversals), dinamik programlama (dynamic programming), geriye dönüşlü dallandırma ve sınırlandırma (backtrack branch and bound) grafiksel modeller (graphical models) ve sonlu hal yöntemleri (finite-state machines) [15].

Yazılım ürün hattı yaklaşımımızda bu "cüce"leri yeniden kullanıma aday bileşenleri bulmada kullanabileceğiz.

3.4. Dış yazılım temini

Dışardan temin edilebilecek yazılımlar temel varlıklar için iyi bir kaynaktır. Birçok alana özel veya genel amaçlı, ticari ve açık kaynak kodlu koşut yazılım geliştirme araçları, bileşenler ve kütüphanler mevcuttur (örn: Intel IPP, Intel MKL, Intel TBB vs.).

Arayüzlerinin, iletişim protokollerinin ve dayandığı standartların hangi ürünlere uyumlu olduğu temel varlık-lara yönelik seçim yaparken göz önünde bulundurul-malıdır.

3.5. Alan analizi

Yazılım ürün hattı sürecinde en önemli çalışmaların başında alan analizi gelir. Bu sayede alan içerisindeki ortaklıklar ve değişkenlikler belirlenmekte ve özellik modeli ortaya çıkartılmaktadır.

3.6. Testler

Ürün hattı yaklaşımında testler temel varlıklardaki yazılım bileşenlerini, ürüne özel yazılımları, aralarındaki etkileşimleri ve son olarak da tamamlanmış ürünleri inceler. Bileşenler üzerindeki test süreçlerinin çıktıları da aslında birer temel varlıktır (test vakaları, test belgeleri, veri setleri ve test yazılımları) ve ürün geliş-tirme aşamasında ürünün gereksinimlerinin karşılan-masının denetiminde kullanılırlar. Testlerin koşut plat-formlara taşınmasındaki sorunların [5] titizlikle ele alın-ması gereklidir.

3.7. Yazılım sistem entegrasyonu

Yazılım sistem entegrasyonu, bağımsız şekilde test edilmiş yazılım bileşenlerinin bir bütün olacak şekilde biraraya getirilmesi çalışmasıdır. Yazılım ürün hattın-daki entegrasyon ise temel varlıkların biraraya gelerek yeni temel varlıklar oluşturulması veya ürünleştirilme-sidir.

Ancak koşut ortamlarda bileşenleri biraraya getirme-nin zorlukları vardır [16] ve tasarlanan ürün bandı yak-laşımında bu konu da ele alanacaktır.

4. Tartışma

Bu çalışma, yazılım ürün hattı yaklaşımının yüksek başarımlı uygulama geliştirilmesine uyarlanmasını önerir. Yazılım ürün hatlarının amacı, benzerliklerden yararlanmak ve değişkenlikleri temel varlıkların sis-tematik bir şekilde yeniden kullanılmasıyla yönetmektir. Bu sistematik yaklaşımın, koşut programlamadaki işlem-leri eş zamanlı bir hale getirmesini ve dolayısıyla ortaya çıkan uygulamaların kalitesini, geliştirici ekiplerin üretkenliğini ve verimliliğini artırmasını bekliyoruz.

5. Sonuç

Koşut programlamanın artık sadece programlama uzmanlarının, araştırma-geliştirme merkezlerinin ve akademinin kullandığı niş bir yöntembilimi olmadığı ortadadır. Koşut programlama, "Koşutluk yoğun yazılım sistemleri" olarak adlandırdığımız bilim-mühendislik, kurumsal, masaüstü ve İnternet uygulamalarını geliştirenler tarafından yaygın bir şekilde kulla-nılmalıdır. Geçtiğimiz otuz yıl içinde kullanılan yakla-şımlar ise koşut programlamadaki sorunların çözümünde yetersiz kalmıştır.

Bu çalışma, yazılım ürün hattı yaklaşımına dayanarak, yüksek başarımlı uygulama geliştirilmesi için yeni bir programlama modeli önermektedir. Herkesçe bilinen yazılım ürün hattı yöntemleri koşutlaştırma gereksinimlerini (kolay koşutlaştırma) karşılayacak şe-kilde güncellenecek ve genişletilecektir. Yüksek başa-rımlı uygulama alanında kazanılan böyle bir başarının diğer koşutlaştırma yoğun yazılım sistemlerindeki engel-lerin aşılmasına da yardımcı olacağını öngörüyoruz.

6. Kaynaklar

[1] Mattson T. and Wrinn M., "Parallel programming: can we PLEASE get it right this time?", *Proceedings of the 45th annual Design Automation Conference*, June 08-13, 2008

[2] VanDeVanter M., Post D. E., and Zosel M. E., "HPC Needs a Tool Strategy", *2nd Int. Workshop on Software Engineering for HPC System Applications*, 2004

[3] Douglass E "Seven Challenges of High Performance Computing", HPCWire - *http://www.hpcwire.com/features/17886204.htm*, July 21, 2006

[4] Bergmark D. and Pancake C. M., "Do Parallel Languages Respond to the Needs of Scientific Programmers?", *Computer-IEEE*, v.23 n.12, p.13-23, December 1990

[5] Wen-mei Hwu, Kurt Keutzer, Timothy G. Mattson, "The Concurrency Challenge," *IEEE Design and Test of Computers*, vol. 25, no. 4, pp. 312-320, July 2008

[6] Standish Group, March 2003, "2003 Chaos Chronicles Report", *http://www.standishgroup.com*, 2010

[7] NATO Science Committee, "NATO Software Engineering Conference", Garmisch, Germany, 1968

[8] Skjellum A., Bangalore P., Gray J. and Bryant B., "Reinventing Explicit Parallel Programming for Improved Engineering of High Performance Computing Software", *International Workshop on Software Engineering for High Performance Computing System Applications*, pp. 59--63, Edinburgh, 2004

[9] Altintas N I, "Feature-based software asset modeling with domain specific kits," *Ph.D. dissertation*, Middle East Technical University, Department of Computer Engineering, 2007.

[10] Keutzer K., Wrinn M., and Mattson T. G., "Architecting Parallel Programs", *International Conference on Computer-Aided Design (ICCAD)*, Tutorial-2, 2008

[11] Carnegie-Mellon Software Engineering Institute, "Software Product Lines" *http://www.sei.cmu.edu/productlines*, 2004

[12] Keutzer K, Mattson T "A design Pattern Language for Engineering (Parallel) Software, *Intel Technology Journal*, Volume 13, Issue 4, 2009

[13] Mattson T. G., Sanders B. A., and Berna L. Massingill B. L., "A Pattern Language for Parallel Programming", *Addison Wesley Software Patterns Series*, 2004

[14] J.S. Poulin. "Measuring Software Re-use: Principles, Practices, and Economic Models". *Addison-Wesley*, Reading, MA, 1997

[15] The Landscape of Parallel Computing Research: A View from Berkeley, *http://view.eecs.berkeley.edu*, 2006

[16] H. Sutter and J. Larus, ''Software and the Concurrency Revolution,'' *ACM Queue*, vol. 3, no. 7, pp. 54-62, Sept. 2005.