MPI'da Uygulama Seviyesinde Aksaklığa Dayanıklılık

Özet: MPI(message passing interface), paralel programlama yapmak için süreçler arası mesaj geçiş ara yüzü sağlayan bir kütüphanedir. Paralel işlemlerin üzerinde çalıştığı işlemciler kullanıcının isteği ile veya haberi olmaksızın azalabilir veya artabilir. Bu durumda donanımı algılayıp ona göre işin alt işlemlere bölünmesi ve donanımdaki değişikliği algılayıp uygun algoritmanın işletilmesi performansı artıracaktır. Bu çalışmada bu işlemin MPI uygulama seviyesinde nasıl bir tasarımla gerçeklendiği anlatılmıştır.

Anahtar Sözcükler: MPI, Aksaklığa Dayanıklılık, Öz Uyarlama.

A New Approach for Quality Function Deployment: An Application

Abstract: MPI is a library to offer message passing interface for developing parallel programming. The processors on which parallel tasks are running, can increase or decrease according to user request or without informing user. In this case, to monitor system and splitting task to sub task according to changing on the system and the number of the processors increase performance. In this paper, Fault tolerant on MPI Application Level is discussed.

Keywords: MPI, Fault Tolerant, Self-Adapting.

1. Giriş

Donanımın gelişmesi ve ucuzlaması ile birlikte uzun sürecek veya yüksek performans gerektiren uygulamalar, birden fazla makine, küme üzerinde paralel bir şekilde işletilirler.

Çoğu optimizasyon probleminin (örneğin gezgin satıcı probleminin 10,000 şehirli örneği[1]) paralel işletilmesinin, ardışık işletilmesine oranı 1'e yakındır.

Dinamik programlama tekniklerinin de büyük hesaplama karmaşıklığından ötürü dezavantajları kümeler üzerinde paralel işlerle azaltılabilir.[2]

FlowVR[3], sanal gerçeklik uygulamalarının küme ve gridler üzerinde çalıştırılmasına ve geliştirilmesine izin veren bir kütüphanedir. Sanal gerçeklik uygulamaları oluşturmak için GrImage platformu ve paralellik için FlowVR kütüphanesi kullanan MOAIS projesi de paralel uygulamalara odaklanmış bir projedir.

Optimizasyon, sanal gerçeklik uygulamaları, dinamik programlama teknikleri gibi paralelliğe uygun problemler için donanımın sağlanmasının yanı sıra bu donanımı etkin bir şekilde kullanabilecek, donanımı algılayabilen öz-uyarlamalı sistemler geliştirmek zorunluluğu ortaya çıkmıştır. Uygun algoritma alternatifleri yaygınlaşırken karmaşık hesaplanabilir ortamın başarılı bir şekilde yönetilmesi problemi Self Adapting Numerical Software[4] ile aşılmaya çalışılır. Üç seviyede çözüm getirilmeye çalışılır. Bunlardan ilki algoritma kararı, ikincisi paralel ortamın yönetilmesi ve sonuncusu işlemci-çekirdeğin spesifik uygunluğudur.

Bilimsel problemlerin verimli çözülmesine engel olan ana etkenlerden biri kullanıcı problemine ve eldeki mimariye uygun yazılım problemidir. Otomatik olarak uygun algoritmanın seçimi için ve uygun yüksek performanslı çekirdeklerin elde edilmesi için çeşitli yaklaşımlar söz konusudur. (ATLAS ve BeBOP örnekleri olan AEOS yaklaşımı gibi)[5]

İhtiyaçlar arttıkça paralel makine modelleri artık daha gerçekçi çözümler vermek için tasarlanmaya başlamıştır. Bu modellerden biri olan LogP[6], dört parametreye dayanmaktadır; iletişim bant genişliği, hesaplama bant genişliği, iletişim gecikmesi, iletişimin verimliliği. Modelin hedefi, hız için analiz ve tasarım ve şimdinin ve geleceğin paralel makineleri üzerinde verimli bir şekilde uygulama yapılmasına izin verecek paralel algoritmalardır. Şimdiki paralel modeller için geliştirilen paralel algoritmalara ana hatlarıyla bakacak olursak, pratik olmadıklarını söyleyebiliriz. Gerçekçi sonuçlar vermezler çünkü sıfır iletişim gecikmesi ve sonsuz bant genişliği gibi yapay faktörler hesaba alınır. Tüm işlemcilerin eş zamanlı bir şekilde çalıştığını, işlemciler arasındaki iletişimin serbest olduğunu varsaydığı için PRAM da gerçekçi olmayan modellerden biridir. BSP (bulk-synchronous paralel model) PRAM'dan yola çıkarak ama teoriyle pratik arasında bağlantı sağlamak için girişimde bulunulmuş daha radikal geliştirilmiş paralel modeldir. LogP için de BSP bir başlangıç noktasıdır.

Yüksek performanslı hesaplamalı sistemler için paralel programlamanın mesaj geçişine dayandığından bahsetmiştik. MPI kullanan sistemlerde tek bir algoritma, topoloji söz konusu değildir. MPI'ın ortak işlemlerini gerçekleştirmek için çok sayıda algoritmanın ayarlanabilir modüller tarafından desteklenmesi gerekir. Açık MPI ayarlanabilir ortak bileşenin (Open MPI tuned collectives component)[7] çoklu ortak uygulamalarına ve çoklu mantık topolojilerine destek vermek gibi bir çok hedefi vardır. Geniş aralıkta ayarlanabilir parametrelere ve kararların verimliliğine kullanıcı tarafından alternatif karar

fonksiyonlarının belirlenmesine, kullanıcı destekli karar parametrelerinin seçimine, dinamik olarak fonksiyonların yaratılması ve silinmesine de destek vermek amaçları arasındadır.

Mesaj geçiş paradigması, uygulama geliştiricilerinden mimari kararların saklanmasını ve soyutlanmasını sağlar ve ölçeklenebilir algoritmalar yazmaya izin verir. Çalışma zamanında bir veya daha fazla süreç erişilemez olduğunda MPI kullanıcıya hatanın nasıl çözülebileceğine ilişkin iki olasılıktan birini seçme şansı verir. İlki MPI'ın default modu, uygulama sonlandırılır. İkinci olasılık ise daha esnektir, kontrolü kullanıcı programına geri verir.[8]

Yüksek performanslı hesaplama sistemlerinin uygulama geliştiricileri ve son kullanıcıları bugün daha büyük makinelere ve çok daha fazla işlemciye bağlanır. Dünya Simülatörü gibi sistemler binlerce veya on binlerce işlemciliden oluşur. Bu tip çok işlemcili sistemlerin karşılaşacağı kritik durum süreç hatalarıdır. 100,000 işlemcili sistemde her dakikada bir süreç hatası ile karşılaşılır. Bu nedenle aksaklığa dayanıklılık gerekir. Aksaklığa dayanıklılık, öz uyarlamalının (self-adaptable) özel bir halidir. İşlemcilerden biri çalışmaz erişilemez duruma geçtiğinde sistemin bu hatayı algılayıp geriye kalan süreç ve işlemcilerle işin yürütülmesi sağlanmalıdır. Aksaklığa dayanıklılık üç adımdan oluşur. İlki hatanın tespiti ikincisi bilgilendirme ve sonuncusu düzeltmedir.

Fazla sayıda işlemcili sistemlerde makinelerin yeniden başlatılması nedeniyle işlemci sayısı azalabilir veya artabilir ve makinedeki hata nedeniyle o makine üzerinde çalışan süreçler sonlanabilir. Sürecin kendisi hatalı olduğunda ise işin paylaştırılmış olduğu süreçlerden birisi artık işi yapamaz olur ve geriye kalan diğer süreçler tarafından o sürecin işi yürütülür. Aksaklığa dayanıklılık da bu nedenle işlemci sayısını algılamalı, işlemcilerden veya süreçlerden biri hatalı oluğunda bunu tespit etmeli, diğer işlemcilere işin dağıtılmalı ve hatalı işlemci veya süreçten haberdar etmelidir. Bu haliyle donanımı algılaması ve ona uygun algoritmayı işletmesi yönüyle öz uyarlamanın özel bir halidir.

İşlemci sayısındaki değişime rağmen paralel algoritmaların kalan işlemci üzerinde verimli bir şekilde işletilmesi, yapılması gereken işin gerçekleştirilmesi ve geniş ağlarda kümelerde çalışan süreçlerin hatalı olma olasılığının yüksek olması nedeniyle paralel ortamı sağlayan kütüphaneler, ara yüzler aksaklığa dayanıklılığı sağlamak zorundadır. MPI da en yaygın kullanılanlardan biri olduğundan yüksek performanslı hesaplamalı sistemlerde aksaklığa dayanıklılık için FT-MPI spesifikasyonu belirlenmiştir. [8] FT-MPI hatanın nasıl tespit edileceği ile ilgilenmez. Bildirimden kasıt, hata hakkında paralel işleyen diğer MPI süreçlerinin bilgilendirilmesidir. Eş zamanlı olarak tüm süreçlerin bilgilendirildiği varsayılır ve süreçlerin ne zaman bilgilendirilmesi gerektiği ile de ilgilenmez. Düzeltme prosedürü iki adımda gerçekleşir; MPI kütüphanesi ve çalışma zamanı ortamının düzeltilmesi ikincisi de uygulamanın ve uygulama verisinin düzeltilmesidir.

FT-MPI uygulama seviyesinde aksaklığa dayanıklılığın nasıl çözüleceğini belirtmez. Bizim yaptığımız çalışma, daha yüksek performans ve hız için günümüzde daha çok ortaya çıkan küme ve gridler üzerinde çalışan ve geliştirilen yazılımların en çok karşılaşacağı problemlerden biri olan işlemci veya süreç hatalarından ötürü oluşabilecek aksaklıklara uygulama seviyesinde bir dayanıklılık sağlanıp sağlanamayacağına ilişkindir. Bu modelin detayları Bölüm 2'de anlatılmıştır. Son olarak Bölüm 3'de yapılan çalışma değerlendirilmiş ve öneriler sunulmuştur.

2. Uygulama

Öz uyarlamalı algoritmalar, ortamı algılar donanıma uygun bir şekilde çalışırlar. Hedef, gerçekleştirilmesi istenen uygulamanın kaynakları optimum bir şekilde kullanmasıdır. MPI, işin süreçlere dağıtılmasını ve süreçlerin paralel bir şekilde işletilmesini ve mesajlaşmasını sağlayan bir kütüphanedir. Klasik iki seviyeli dağıtık sistem modeli olan sunucu istemci modeli olarak düşünülebilir. Süreçlerden biri sunucu görevi görür ve istemci süreçler sunucu tarafından gönderilen görevi yapıp sonuçları sunucu sürece göndermekle sorumludurlar. İstemci süreçlerin sayısı kullanıcı tarafından girilen bir parametre ile belirlenir. Kullanıcının girdiği değer kadar süreç oluşturulur ve her birine bir id atanır. Genellikle id'si 0 olan süreç sunucu süreç görevi görür.

Kullanıcıların birçoğu uygulamasını işletirken donanımdan haberdar olmayacak ya da olmak istemeyeceklerdir. Daha önce de bahsedildiği gibi istenen, donanımın algılanması ve ona uygun algoritmanın işletilmesidir. Süreçler eldeki işlemcilere sırayla dağıtılırlar, her bir işlemciye başlangıçta bir süreç atanır. Süreç sayısı işlemci sayısından fazlaysa örneğin n işlemci, n+2 süreç varsa n süreç n işlemciye dağıtıldıktan sonra n+1. süreç 1 numaralı işlemciye, n+2. süreç ise 2 numaralı işlemciye atanır. Yapılması istenen görev birden fazla ardışık seviyeden oluşuyorsa ve her bir sürecin bir işlemcide işletilmesi t zaman alıyorsa, n+1. ve n+2. süreçlerden ötürü yapılması istenen görev 2t zaman alacaktır ve diğer işlemciler bu süreçler işletilirken boşta bekletilecektir. Oysa görevin her seviyesi eldeki işlemci sayısı kadar sürece bölünürse seviyenin bitirilme süresi (< 2t) ilk duruma göre daha kısa olacaktır ve bu şekilde boşta bekleyen işlemcilerin önüne geçilmiş olunur.

Aşılması gereken diğer bir nokta süreçlerde veya süreçlerin üzerinde çalıştığı makinelerde problem olması durumudur. Bu durumda çeşitli yaklaşımlar ile öz uyarlama yapılabilir. Yöntemlerden biri, denetçi bir sürecin oluşturulmasını içerir. Denetçi süreç belirli zaman aralıklarında istemci süreçlerin ayakta olup olmadığını anlamak için onlardan mesaj bekler. Süreçlerden birinden bu tip bir mesaj, beklenen sürede gelmezse denetçi süreç sunucuya bilgi mesajı gönderir. Sunucu bu durumda bilgi alınamayan istemci sürecin işletmesi gereken görevi diğer işlemcilere dağıtır. İkinci bir yaklaşımda ise işlemci sayısının belirli sayıda katı kadar süreç oluşturulur. Bunlardan işlemci sayısı kadar olanı, aktif istemci süreç diğerleri ise pasif istemci süreç sınıfına ayrılır. Aktif istemci süreçlerden biri çalışamaz hale geldiğinde, denetçiye mesaj atmadığında veya atamadığında pasif durumda olan süreçlerden biri aktif hale getirilip bu süreçten beklenen görev bu istemciye atanır. Böylelikle görevin devamı sağlanır ve işlemcilerin boşta bekleme süresi olabildiğince minimize edilmeye çalışılır. Bir diğer yaklaşımsa yapılacak görev belirli ardışık süreçlerden oluşuyorsa her süreç bitiminde aktif işlemci sayısı yeniden öğrenilir ve bir sonraki seviye aktif işlemci sayısına bölünür. Eğer alınan sonuç sayısı başta belirlenen aktif işlemci sayısına eş değer değilse bir takım süreçlerde hata oluştuğu açıktır. Bu durumda alınan sonuçların ortalaması bulunup cevap alınamamış süreçlerin sonucu bu ortalama olarak kabul edilebilir.

İşlemci sayısı kadar sürecin aktif hale getirilmesi için MPI'ın MPI_Get_processor_name(char *name, int *resultlen) fonksiyonu kullanılır. name işlemcinin adını döndürür. Ne kadar farklı işlemci adı varsa o kadar işlemci vardır. Bu yolla işlemci sayısı öğrenilir ve o kadar sürece aktif geriye kalanlara da uyumaları için pasif mesajı gönderilir. Sunucu aktif süreçleri ve hangi işlemci üzerinde çalıştıklarını bir yapıda tutarken aynı şekilde pasif olanların da bilgilerini saklar. Süreçlerden biri denetçi, biri sunucu süreç

olarak belirlenir. Geriye kalanlar ise aktif istemci ve pasif istemci süreçler olarak ikiye ayrılırlar. Sunucu sürecin iki tür iletişimi söz konusudur. Birincisi istemcilerle olan iletişimi diğeri denetçiyle olan iletişimidir. Denetçiden herhangi bir zamanda süreçlerden birinde hata tespit edildiğinde sunucuya haber gelir. Bu aksaklığa dayanıklılığın hata tespiti aşamasıdır. Sunucu istemcileri, işlemci sayısı konusunda bilgilendirmelidir. Böylelikle tüm bilginin ve görevin ne kadar işlemciye dağıtıldığına ve işin kendilerine düşen bölümüne hakim olmaları sağlanır. Bu şekilde aksaklığa dayanıklılığın bildirim kısmı gerçeklenmiş olur. Son olarak yukarda sayılan yöntemlerden herhangi biriyle düzeltme aşaması da gerçeklenir.

Böylelikle, yapılması gereken görev optimum bir şekilde paralelleştirilmiş ve işlemci sayısı kadar sürece bölünmüş olur. Ardışık bir şekilde işletilmesine oranla daha kısa bir cevap süresinde görev paralel bir şekilde işletilir. Birden fazla işlemci üzerinde çalıştırılan işlemlerde karşılaşılabilecek aksaklıklara karşı da dayanıklı bir model MPI'ın uygulama seviyesinde bu yolla gerçeklenmiş olur.

3. Sonuç ve Öneriler

MPI, paralel süreçler arası mesajlaşmayı sağlayan yaygın olarak kullanılan bir kütüphanedir. Günümüz uygulamaları, gelişen teknoloji ve artan beklentilere uygun olarak birden fazla makine, küme ve gridler üzerinde geliştirilir ve çalıştırılırlar.

İşlemcilerin bazıları uygulama çalıştırılırken uygulamadan ayrılabilir, çalışmaz duruma geçebilirler. İşlemci sayısının algılanıp ona uygun bir şekilde işin süreçlere bölünmesi ve en iyi performansın gerçekleştirilmesi bir zorunluluktur.

MPI'ın aksaklığa dayanıklılık spesifikasyonu uygulama seviyesinde değildir. Biz bu çalışmada paralelleştirilen algoritmalardan, değişen yazılım bakış açısından bahsederek bu çözümün neden gerektiği hakkında bilgi verdik. MPI'ın uygulama seviyesinde işlemci sayısının nasıl algılanabileceği, işin bu işlemci sayısına nasıl bölünebileceği, denetçi süreç sayesinde işlemcilerin ayakta olup olmadığının nasıl anlaşılacağı ve buna uygun öz uyarlamalı algoritmaların nasıl oluşturulabileceğini tasarım aşamasında anlattık.

Burada bahsedilen paralelleştirilmiş optimizasyon problemlerinin, sanal gerçeklik algoritmalarının ve diğer örneklerin ardışık olana oranla nasıl bir performans göstermiş olduğu başka çalışmalarda gösterilmiştir. Biz bunu doğru varsayarak paralelleşen algoritmaların ortamı algılayıp, ortama uygun bir şekilde çalıştırılmasının MPI kullanan sistemlerde uygulama seviyesinde nasıl bir tasarımla başarılabileceğini gösterdik.

4. Kaynaklar

[1] Talbi, E.G., "Parallel Combinatorial Optimization", John Wiley & Sos, Inc, Publication, (2006)

[2] Canto, S.D., Madrid, A.P. and Bencomo, S.D., "Parallel Dynamic Programming on Clusters of Workstation", Proceedings of the Parallel and Distributed Systems. Volume 16, Issue 9, 785-798 (2005)

[3] Vernizzi, D., "Self-adaptive parallel algorithms for computer vision applications", (2005)

[4] Bosilca, G., Zizhong, C., Dongarra, J., Eijkhout, V., Fagg, G.E., Euntes, E., Langou, J., Luszczek, P., Pjesivac-Grbovic, J., Seymour, K., You, H. and Vahiyar, S.S., "Self Adapting Numerical Software (SANS) Effort" , IBM Journal of Research and

Development. Volume 50, Number 2/3, Page 223-238, (2006).

[5] Demmel, J., Dongarra, J., Eijkhout, V., Feuntes, E., Petiet, A., Vudue, R., Whaley, RR.C. and Yelick, K., "Self Adapting Linear Algebra Algorithms and Software", Proceedings of the IEEE 93, 2, 293–312.(2005)

[6] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R. and Eicken, T., "LogP: Towards a realistic Model of Parallel Computation", In Fourth ACM SIGPLAN Symposiumon Principles and Practice of Parallel Programming, pp. 262–273, (1993)

[7] Fagg, G.E., Bosilca, G., Pjesivac-Grbovic, J., Angskun, T. and Dongarra, J.J., "Tuned: An Open MPI Collective Communications Component", In Distributed and Paral-
lel Systems, pages 65□ 72. Springer US, (2007)

[8] Fagg, G.E., Bosilca, G., Angskun, T., Chen, Z., Presivac-Grbovic, J., London, K. and Dongarra, J.J. "Extending the MPI Specification for Process Fault Tolerance on High Performance Computing Systems.", Proceedings of the 19th International Supercomputer Conference (ISC2004), Heidelberg, German, June 21-24,(2004)