Çizge Uygulamalarına Özel İşlemci Tasarımı

Gülce PULAT, Aamir SAEED, Mehmetali Semi YENİMOL, Utku GÜLGEÇ ve Özcan ÖZTÜRK
Bilgisayar Mühendisliği Bölümü
Bilkent Üniversitesi
Bilkent, Ankara, Türkiye
gulce.pulat, aamir.saeed, semi.yenimol, utku.gulgec, ozturk@cs.bilkent.edu.tr

2022 30th Signal Processing and Communications Applications Conference (SIU) | 978-1-6654-5092-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SIU55565.2022.9864699

***Özetçe* — Bir çok büyük veri işleme uygulaması "PageRank", "İşbirliğine Dayalı Filtreleme" ve "Betweenness Centrality" gibi düzensiz ve yinelemeli çizge algoritmalarından yararlanmaktadır. Çizge yapılarında her bir düğüm bir kişiye ya da nesneye karşılık gelirken, her bir ayrıt da bir kişi ya da nesne çifti arasındaki ilişkiye karşılık gelmektedir. Böylesi büyük verileri işleyen algoritmaları yürütecek genel amaçlı işlemciler yetersiz kalabilmektedir. Özellikle, güç kısıtları sebebiyle işlemci alt parçalarından sadece bir kısmı aynı anda aktif olarak kullanılabilmektedir. Bu da işlemcinin etkin olarak kullanılmasına engel olarak aktif olmayan "karanlık silikon"lar ortaya çıkarmaktadır. Bu bildirinin amacı da büyük verili ve düzensiz çizge uygulamalarını hızlı, verimli ve kolay programlanabilir biçimde çalıştıracak bir işlemci mimarisi tasarlamaktır. Literatürde çizge uygulamaları için daha önce sunulmuş çalışmalar çoğunlukla sadece yazılım ya da hızlandırıcı seviyesindedir. İşlemci seviyesi tasarımlar ise donanımsal maliyet, talep ettikleri mimari destek ve komut seti değişiklikleri açılarından bu bildiri kapsamında tanıtılacak işlemciden farklıdır.**

Anahtar Kelimeler* — *Açık kaynak kodlu işlemciler, uygulamaya özel komut genişletme, derleyici, yazılım geliştirme ortamı, çizge uygulamaları, işlemci, tasarım, FPGA

***Abstract* — Many widely used big data applications make use of irregular and iterative graph processing algorithms like PageRank, collaborative filtering and betweenness centrality. In a graph structure, each vertex corresponds to a person or object, while each edge corresponds to the relationship between a pair of persons or objects. In executing algorithms that process such large datasets, general-purpose processors may be inefficient. In particular, due to power constraints, only a portion of the processor can be used simultaneously. This, in turn, prevents the efficient use of the processor, which leads to inactive areas termed as "dark silicone". The aim of this paper is to design a processor architecture that will run large datasets on irregular graphs quickly, efficiently and in an easily programmable fashion. Most of the studies previously presented for graph applications are only at the software or accelerator level, and the processor-level designs are different from the processor to be introduced for this paper in terms of hardware cost, required architectural support, and instruction set modifications.**

Keywords* — *Open source processors, application specific instruction set extension, compiler, software development kit, graph applications, processor design, FPGA.

I. GİRİŞ

2000'li yılların başlangıcı, entegre devre imalatında teknoloji boyutlarında da nanometre ölçülerine geçildiği dönem olmuştur [1]. Milenyumun ilk yılında 180 nm seviyesinde imal edilmiş olan Intel'in Pentium 4 işlemcisi piyasaya sürülmüş, 2006 yılında ise 90 nm teknolojisinin kullanıldığı Itanium işlemcisinin tanıtılmasıyla ilk kez 100 nm seviyesinin altına inilmiştir. Teknoloji boyutlarındaki bu küçülme, aynı boyuttaki bir entegreye yerleştirilebilecek transistör sayısını da doğal olarak artırmış, aynı boyutlara çok daha fazla fonksiyon eklenebilmiştir. Aslında, transistör sayısındaki artış miktarını yıllar önce Intel'in kurucularından olan Gordon E. Moore, daha sonra Moore'un kanunu adıyla anılacak olan "Entegre devrelerdeki transistör sayısı her iki yılda iki katına çıkacaktır." ifadesiyle tahminde bulunmuş ve bu tahmin yaklaşık olarak belli bir süre doğrulanmıştır [2]. Son yıllara kadar, transistörler küçüldükçe, harcadıkları güç miktarı da benzer bir oranda düşmekteydi (Dennard ölçeklemesi [3]). Son yıllarda ise, transistörler küçülmeye devam ettiği halde, her bir transistörün harcadığı güç miktarı fiziksel nedenlerden dolayı aynı oranda azalmamaya başlamıştır. Bu da "karanlık silikon" adı verilen bir kavramın ortaya çıkmasına yol açmıştır [4]. Buna göre, bir yonga içinde çok miktarda hesaplama modülü olabilir. Fakat yonga tarafından harcanabilen güç kısıtlı olduğundan, bu modüllerin sadece bir kısmı aynı anda aktif olarak kullanılabilir. Burada aktif olmayan kısımlara "karanlık silikon" denmektedir. Mesela 5nm transistör teknolojilerinde, bir yonganın yaklaşık yüzde 80'inin karanlık olması beklenmektedir [5].

Enerji verimliliğini arttırmak için, değişik işlemci ve entegre birimlerinden oluşan heterojen sistemler kullanılabilir [6]. Değişik mimarilerden oluşan çekirdekler kullanmak enerji verimini ciddi bir miktarda arttırabilir. Örneğin, 16 çekirdekten oluşan bir mimaride çekirdeklerin bir kısmı genel kullanıma yönelik skaler üstü ("superscalar") işlemci mimarisine sahipken başka bir kısmı, tek-komut-çok-veri ("SIMD") programlarını yüksek performansta çalıştırabilecek vektör işlemcilerinden oluşabilir. Veya çizge algoritmalarına yönelik özel bir mimariye sahip çekirdek işlemcileri böyle bir yapıya eklenebilir ve uygulamalar kendilerine en uygun

978-1-6654-5092-8/22/$31.00 ©2022 IEEE

çekirdeklerde çalıştırılırken, kullanılmayan çekirdekler kapatılabilir. Bununla birlikte son yıllarda, GPU ve FPGA gibi hızlandırıcı temelli mimariler skaler üstü işlemcilerin yanında sunucu bilgisayarlara da eklenmeye başlamıştır [7, 8]. Birçok büyük veri uygulamasının bu tip heterojen sistemlerde çok daha verimli olarak çalıştığı da gösterilmiştir [9, 10, 11].

Çizge uygulamalarının kullanım alanının sürekli genişlediği ve ilgilendiği verilerin hacmi ile çeşitliliğinin giderek arttığı düşünüldüğünde, bu uygulamalara yönelik özel donanımların performans ve enerji verimliliği açısından gündeme alınması gerekmiştir. Enerji verimliliğini arttırmak için değişik işlemci ve entegre birimlerinden oluşan heterojen sistemler kullanılabilir. Bu sayede hem büyük verinin donanımsal ihtiyaçları gözetilebilir hem de karanlık silikonun kapsamı düşürülebilir. Birçok büyük veri uygulamasının bu tip heterojen sistemlerde çok daha verimli çalıştığı da gösterilmiştir.

Bu bildirinin amacı da büyük verili ve düzensiz çizge uygulamalarına yönelik bir işlemci mimarisi tasarlamaktır. Burada ana hedef, genel kullanıma yönelik işlemcilere göre çok daha yüksek performans ve enerji verimi elde etmektir. Böyle bir işlemci, heterojen veri merkezlerinde çizge algoritmalarını çalıştırmak için kullanılabilir. Hali hazırda kullanılmakta olan birçok büyük veri uygulaması, çizge algoritmalarından yararlanmaktadır. Çizge yapılarında her bir düğüm bir kişiye ya da nesneye karşılık gelirken, her bir ayrıt da bir kişi ya da nesne çifti arasındaki ilişkiye karşılık gelmektedir. Bu tip uygulamalarda düğüm ve ayrıt sayıları milyarlarla ya da trilyonlarla ölçülmektedir. Günümüzde birçok şirket bu tip çizgeleri analiz ederek ticari açıdan önemli bilgilere erişebilmektedir. Bunun için veri madenciliği ve makine öğrenimi algoritmaları kullanılmaktadır. Örneğin, sosyal ağ analizi yapan bir şirket sosyal ağdaki her bir kullanıcıyı bir düğüm, arkadaşlık ilişkilerini de ayrıtlar olarak tanımlayabilir. Arkasından da gruplamalar ve davranış analizleri böyle bir çizge üzerinde gerçekleştirilebilir.

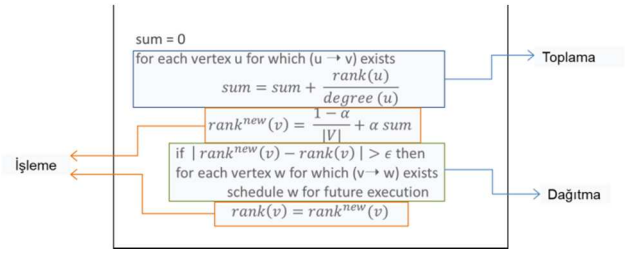
Çizge işlemcisinin tasarımı, gerçekleştirilmesi ve test edilmesi süreçlerinin aşamaları sırayla şu şekildedir:

1) Düzensiz ve yinelemeli çizge uygulamaları hakkında istatistikler toplanacak, bu uygulamaların başvurduğu algoritmalar analiz edilecektir. Daha önceki çizge algoritma analizlerinde sadece saniyede yapılan iş miktarları değerlendirilmiştir. Bu bildiride hem saniyede yapılan iş miktarı hem de iş verimliliği göz önünde bulundurulacaktır. Böylece bir çizge işlemcisinin hangi alanlarda daha yüksek performanslı olması gerektiği açığa çıkacaktır.
2) Genel bir işlemciyi özel bir çizge işlemcisine çevirmek için mimarisine eklenecek yeni komutlar ve mimari tasarımında yapılacak değişiklikler belirlenecektir.
3) Çizge işlemcisinin tasarımı yapılacaktır. Bunun için yeni komutların komut setine eklenmesi ve işlemcinin kaynak kodundaki donanım birimlerinin güncellenmesi gereklidir.
4) Çizge işlemcisini desteklemek ve test etmek için gerekli derleyici, benzetim aygıtı, test kodları ve yazılım kütüphaneleri yazılacaktır.
5) Çizge işlemcisi FPGA üzerinde gerçeklenecektir.
6) Hem benzetim yoluyla işlemci yazılımı üzerinde, hem de FPGA'de gerçeklenmiş işlemci donanımı üzerinde doğruluk, performans ve verimlilik testleri gerçekleştirilecektir.
7) Çizge uygulamaları çizge işlemcisi için optimize edilecektir.

Ana hedefimiz çizge algoritmalarında genel kullanıma yönelik işlemcilere göre çok daha yüksek performans ve enerji verimi elde etmektir. Böyle bir işlemci, heterojen veri merkezlerinde çizge algoritmalarını kullanan uygulamaları çalıştırmak için kullanılabilecektir.

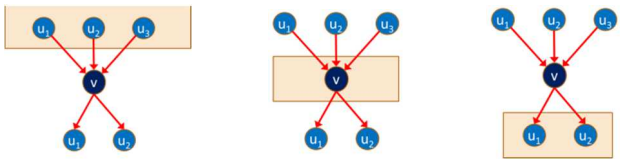
II. Düzensiz ve Yinelemeli Çizge Uygulamaları

Düzensiz ve yinelemeli çizge uygulamaları, özellikle büyük veri işleme alanlarında kullanılmaktadır. Bunlara örnek olarak arama motorlarında kullanılan PageRank algoritması [13], tavsiye sistemlerinde kullanılan İşbirliğine Dayalı Filtreleme ("Collaborative Filtering") algoritması [14] ve sosyal ağ analizi için kullanılan "Betweenness Centrality" [15] algoritması gösterilebilir. Bu tip uygulamaların genel özelliklerini temsilen anlatmak için kullanılan en yaygın uygulama PageRank'tir. Şekil 1'de bu algoritmanın genel yapısı verilmiştir.



Şekil 1. PageRank Algoritması.

PageRank algoritmasında adından da anlaşılacağı üzere her bir düğümün önemine karşılık gelen bir "rank" değişkeni hesaplanmaktadır. Aktif durumdaki bütün v düğümleri işleme tabii tutularak sayfaların sıralaması belirlenir. Her bir yineleme adımında ("iteration") komşu düğümlerin değerleri alınarak yeni değer hesaplanır. Arkasından yeni değer komşu düğümlerle paylaşılır. Bu hesaplama eski ve yeni değerler arasındaki fark yeterince küçük oluncaya kadar devam eder. Bu düğüm yakınsamış ("converged") oluncaya kadar toplama-işleme-dağıtma ("GAS - gather-apply-scatter") fonksiyonunu Şekil 2'de gösterildiği gibi sürekli yineler.



Şekil 2. Her bir düğümde gerçekleşen toplama-işleme-dağıtma ("GAS - gather-apply-scatter").

Bu örnekten de anlaşılacağı üzere çizge uygulamalarında her bir düğüme karşılık gelen çekirdek ("kernel") program

birbirinden bağımsız çalıştırılabilmektedir. Ancak "toplama" yapılırken komşu düğümlerdeki en yeni değerler alınmalı ve yapılacak olan "işlem"de son değerler kullanılmalıdır. Bu "işlem" bir döngü içinde yapılmaktadır ve bu döngü bütün düğümlerin değerleri yakınsayıncaya kadar devam etmektedir. Aslında birçok çizge algoritması böyle bir çalışma prensibine sahiptir [16, 17].

Bu tip algoritmaların etkin bir biçimde çalıştırılabilmesi için gerekli özelliklerden bazıları şöyle sıralanabilir:

1) Aktif düğümler var olan bütün düğümlerden azdır ve aktif düğümler üzerine işlem yapılması daha etkili olacaktır.

2) Düğümler komşusunun en son hesapladığı veriyle işlem yapabilmelidir. Böylece her bir veri için ayrı bir kilit kullanmaya gerek kalmaz.

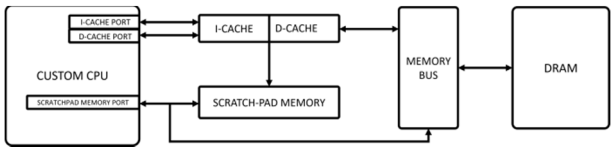
3) Önbellekte bulunmayan her veri parçası için ana belleğe erişim gereklidir. Dolayısıyla, çizge algoritmalarına uygun bir mimaride çok sayıda bellek isteği aynı anda yapılabilmelidir.

4) Büyük veri uygulamalarında kullanılan çizgelerin çoğunda düzenli bir yapı yoktur. Dinamik yük dengeleme yapılmasını destekleyen mimariler yararlı olacaktır.

III. MİMARİ TASARIM

Bu bildiride, büyük verili çizge algoritmalarına yönelik bir işlemci mimarisi ve ek donanım birimleri önerilmektedir. Önerilen bu mimari, Şekil 3'te kısaca özetlenmiştir. Buradaki işlemci için, aşağıda daha detaylı anlatılacağı gibi, çizge algoritmalarına yönelik bazı özellikler önerilmektedir. Bu işlemcide yapılan hesaplamaların geçici verileri geçici bellekte (GB - "scratch pad memory (SPM)") kaydedilecektir. Önerilen donanım modülü, standart bir önbellek modülü üzerinden sistemin geri kalanına bağlanacaktır. Bu donanım modülünün çalışma prensipleri aşağıda daha detaylı olarak anlatılmaktadır.

Önerilen sistemin mimarisi, geçici bellek eklenmesini içerdiği için tasarımda birinci seviye önbellek ile birlikte tasarlanmıştır. GB için varsayılan erişim süresi ve ilgili tüm veri okuma ve yazma parametreleri ön-bellek ile aynı olacak şekilde alınmıştır.



Şekil 3. Önerilen işlemci mimarisi ve ek donanım birimleri.

GB hem işlemci, hem önbellek, hem de ana bellek veriyolu ile bağlantılı bir şekilde tasarlanmıştır. GB'de saklanması gereken veriler seviye 1 önbellekte de bulunabilir, böylece benzer verileri almak için ana belleğe gitmeyerek zamandan tasarruf edilmesi mümkün olabilecektir. Ana bellek ile GB arasındaki iletişim önbellek arayüzünde olduğu gibi bellek veriyolu üzerinden sağlanmaktadır. Sistemimizin bellek veri yolu hem önbellekten hem de GB'den aynı anda birden çok isteği işleyebilir. GB'ye veri yazarken, işlemci ilk önce verilerin önbellekte bulunup bulunmadığını kontrol eder. Eğer mevcut değilse önbelleğe alınması için oluşturulan istek belleğe iletilir. Aksi takdirde, eğer mevcutsa, veriler önbellekten GB'ye kopyalanır. Bu nedenle, GB'ye yazmanın üç olası yolu vardır: 1) önbellekten GB'ye yazmamız gerektiğinde , 2) GB'ye ana bellekten yazarken, ve 3) işlemcinin GB'ye yazması. Her üçü de diğelerinden bağımsız olarak GB'ye yazabilir.

Bu mimari yapı GEM5 [12] simülasyon ortamında gerçekleştirilerek deneysel sonuçlar alınmıştır. GEM5'te bulunan örnek sıralı işlemci (in-order) mimarisi Minor-CPU değiştirilip gerekli ek donanım parçaları ilave edilmiştir. 4 evreden (4-stage) oluşan boru hattı (pipeline) yapısında Fetch1, Fetch2, Decode ve Execute evreleri bulunmaktadır. Bu mimarinin bizim istediğimiz işlemleri gerçekleştirebilmesi için özellikle Decode ve Execute evrelerinde değişiklikler yapılmıştır. Alana özel tanımladığımız komutlar Execute evresinde uygulanmıştır.

Yürütme aşaması, tüm komut yürütme ve bellek erişimini sağlayan mekanizmaları içerir. Bu noktada bellek erişim mekanizması ikiye ayrılır: biri normal erişim için bellek yönergeleri ve diğeri uzun gecikmeli bellek (long-latency memory) yönergeleri içindir. GB'ye gitmesi zorunlu olan komutlar SPM komutları olarak ifade edilecektir. İşlemcide kullanılan SPM komutları herhangi bir normal komut gibi verilir ancak hafıza adreslerini hesapladıktan sonra fonksiyonel üniteden çıktıklarında içinde tanımlanmalarını kolaylaştırmak için SPM komutları olarak etiketlenmiştir.

Hafızadan getirme komutları yalnızca bellek yanıtı alındığında işlendiğinden yürütme aşamasında komut penceresinde kalma eğilimindedirler. Bu da boru hattının durmasına neden olur. Bu nedenle, tasarımımıza göre tüm SPM komutları ALU işlemleri tamamlandıktan sonra komut penceresinden dışarı alınarak ayrı bir pencere ve kuyrukta tutulur. Bu şekilde boru hattında bulunan diğer komutlar duraklamaya gerek kalmadan yürütmeye ve sonlandırmaya (commit) devam edebilirler.

Minor-CPU modeli, hafızadan veri getirme ve saklama işlemlerinden sorumlu olan bir kuyruğa (LSQ – Load Store Queue) sahiptir. Buna ek olarak SPM komutlarını işlemek üzere yeni bir kuyruk oluşturularak bellek isteklerini ve SPM komutlarını saklamak mümkün olmuştur. Bu "SPM Buffer" iki ayrı parçaya bölünür. Bunlar sırasıyla SPM isteği (request) ve SPM aktarımıdır (transfer). SPM istek kuyruğu ana bellekten veri transferi talep eden SPM komutlarını tutar. Veri ana bellekten geldiğinde SPM komutu istek kuyruğundan alınarak transfer kuyruğuna aktarılır. Burada "Barrier" komutunu bekleyerek sırası geldiğinde sonlandırılır.

SPM yanıt kuyruğu, ana bellekten birden çok çakışan bellek erişimi yapabilir. Bellek erişimleri toplu olarak verilerek SPM yükleme talimatları için tüm yanıtların geri gelip gelmediğini anlamak için bir kayıt (flag) tutulur. Bu kayıt etkinleştiğinde de ilgili "Barrier" komutu çalıştırılarak bekleyen komutların çalıştırılabilmesi sağlanır. Barrier komutu her zaman SPM'den okumaya çalışmadan önce verilir ve amacı, karşılığında kontrol eden SPM kuyruğunun kayıtlarını kontrol etmektir.

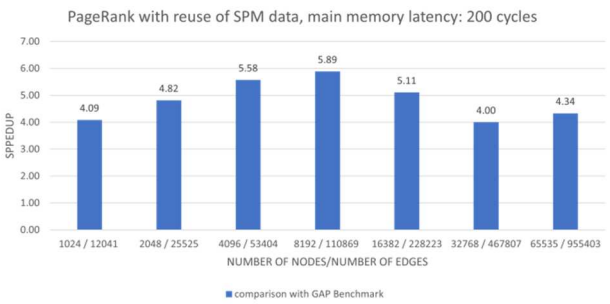
IV. SONUÇLAR

Çalışmada, tercih edilen simülatör GEM5, komut seti mimarisi X86, ve ek donanım ise SPM olarak belirlenmiştir. Düğüm merkezli çizge algoritmalarını adil bir şekilde değerlendirmek için sonuçlarımız önceden belirlenmiş bir standart olan GAP ile karşılaştırılmıştır. Bu bildiride bazı sonuçlar verilmiş olmakla birlikte tüm detaylı sonuçlar kapsamı büyütmemek adına verilmemiştir.

Tüm kıyaslamalar çizge işlemcisi ve normal bir CPU arasında yapılmıştır. Normal CPU'da SPM gibi bir geçici bellek bulunmadığı gibi özel bellek komutlarına erişim de yoktur. Ancak bunun dışında hem çizge işlemcisi hem de genel CPU aynı özelliklere sahip olacak şekilde tasarlanmıştır, tek fark SPM ve ek komutlardır. İlave edilen SPM ve ek komutlarla ilgili gerekli olacak ek donanım maliyetinin %1'ler mertebesinde olması beklenmektedir.

Testlerin yapılması sırasında dört farklı uygulama kullanılmış olup bunlar farklı alanlarda yaygın bir şekilde kullanılan çizge uygulamalarından seçilmiştir. Özel olarak GAP benchmarklarından 4 çekirdek seçilmiştir: Sayfa Sıralama (PR –Page Rank), Tek Kaynaklı En Kısa Yol (SSSP – Single Source Shortest Path), Bağlantılı Bileşenler (CC – Connected Components) ve Üçgen Sayma (TC – Triangle Counting). Bunlar özellikle iletişim merkezli bir iş yüküne sahip ve düğüm merkezli çizgeler oldukları için seçilmiştir.

Şekil 4'te çizge işlemcisinin farklı parametrelerle ve çizgelerle denenen PageRank uygulaması için verdiği hızlandırma sonuçları görülmektedir. Bu sonuçlardan da anlaşılacağı üzere SPM ve ilgili komutlar potansiyel iyileştirmeler içermektedir. Diğer uygulamalarda da iyileştirmeler olmakla birlikte en belirgin iyileştirmeler PageRank uygulamasında görülmüştür.

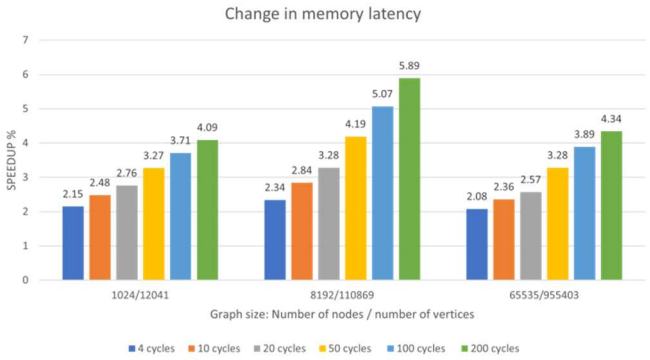


Şekil 4. Çizge işlemcisinin PageRank uygulamasında kullanılmasıyla ortaya çıkan hızlandırmalar.

Yapılan farklı deneylerde hafıza zamanlarının artmasıyla çizge işlemcisinin elde ettiği hızlandırmaların arttığı gözlemlenmiştir. Şekil 5'te anabelleğe erişimin 4 saat çeviriminden 200 saat çevirimine kadar değiştirildiğinde ortaya çıkan hızlandırmalar görülmektedir.

BİLGİLENDİRME

Bu çalışma Türkiye Bilimler Akademisi (TÜBA) tarafından ve Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) 1001 programı EEEAG 119E559 projesi ile desteklenmiştir.



Şekil 5. Anabelleğe erişimin 4 saat çeviriminden 200 saat çevirimine kadar değiştirildiğinde ortaya çıkan hızlandırmalar.

KAYNAKLAR

[1] *"Integrated circuit". Industrialin. https://industrialin.com/integrated-circuit*

[2] Bradley, D. "Keeping up with Moore's Law". phys.org http://phys.org/news/2018-12-law.html

[3] Bassous, E., Dennard, R. H., Gaensslen, F., LeBlanc, A., Rideout, L., Yu, N. 1974."Design of ion-implanted MOSFETs with very small physical dimensions", IEEE Journal of Solid State Circuits, 12, 38-50.

[4] Burger, D., Blem, E., Esmaeilzadeh, H., Sankaralingam, K., St.Amant, R. 2011."Dark Silicon and the End of Multicore Scaling", in Proc. of Int'l Symp. On Computer Architecture (ISCA).

[5] Yeric, G. 2014. "Moore's Law at 50: Are we planning for retirement?", International Electron Devices Meeting (IEDM).

[6] Taylor, M. 2012. "Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse", Proc. of Design Automation Conference.

[7] Wile, B. "Coherent Accelerator Processor Interface (CAPI) for Power8 Systems". IBM. ibm.com/webapp/set2/sas/f/capi/CAPI_POWER8.pdf.

[8] Putnam, A. ve diğerleri. 2014. "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services", 41st Annual International Symposium on Computer Architecture (ISCA).

[9] Blott, M. 2014. "Key Value Store Acceleration with OpenPower", IEEE/ACM Workshop on Heterogeneous Computing Platforms.

[10] Hagleitner, C. 2014. "Text-Analytics Acceleration with OpenPower", IEEE/ACM Workshop on Heterogeneous Computing Platforms.

[11] Beece, D. 2014. "FPGA-Based Monte-Carlo Simulation Acceleration on Power8", IEEE/ACM Workshop on Heterogeneous Computing Platforms.

[12] Beckmann, B. ve diğerleri. 2011. "The Gem5 simulator", ACM SIGARCH Computer Architecture News, 39(2), 1-7.

[13] Brin, S., Motwani, R., Page, L., Winograd, T. 1999. "The PageRank citation ranking: bringing order to the Web", Stanford InfoLab.

[14] Gemulla, R., Haas, P. J., Nijkamp, E., Sismanis, Y. 2011. "Large-scale matrix factorization with distributed stochastic gradient descent", Proc. of the 17th ACM SIGKDD International Conference on Konwledge Discovery and Data Mining (KDD).

[15] Brandes, U. 2001. "A faster algorithm for betweenness centrality", Journal of Mathematical Sociology, 25(2), 163-177.

[16] Bickson, D., Gonzalez, J., Guestrin, C., Hellerstein, J. M., Kyrola, A., Low, Y. 2012. "Distributed Graphlab: A framework for machine learning in the cloud," Proc. of the VLDB Endowment.

[17] Austern, M. H., Bik, A. J., Czajkowski, G., Dehnert, J. C., Horn, I., Leiser, N., Malewicz, G. 2010."Pregel: A system for large-scale graph processing", Proc. of International Conference on Management of Data (SIGMOD).