PARALEL GENELLEŞTİRİLMİŞ TENSÖR ÇARPIMI
PARALLEL GENERALIZED TENSOR MULTIPLICATION

Can Kavaklıoğlu, A. Taylan Cemgil

Bilgisayar Mühendisliği Bölümü
Boğaziçi Üniversitesi
{can.kavaklioglu, taylan.cemgil}@boun.edu.tr

ÖZETÇE

Saklı Tensör Ayrışımı çerçevesi çok miktarda ve çok boyutlu veri içeren tensör ayrışımı problemlerine olasılıksal bir bakış açısıyla yaklaşır. Çerçevenin tanımladığı güncelleme operasyonlarında kullanılan genelleştirilmiş tensör çarpımı operasyonları çok miktarda benzer yapıda aritmetik işlemler yapılmasını gerektirir. Bu çalışma tekrar eden bağımsız işlemleri grafik işlemci üniteleri (GPU) üzerinde paralel çalıştırarak elde ettiğimiz sonuçları göstermektedir.

ABSTRACT

Tensor factorization is a frequently used modelling tool in problems involving large amounts of n-way data. Probabilistic Latent Tensor Factorization framework provides a probabilistic approach to solve the tensor factorization problem. The iterative algorithms use generalized tensor multiplication operations involving large amounts of arithmetic operations with similar structures. This work shows the performance improvements achieved by performing the independent operations on a graphical processing unit (GPU).

1. GİRİŞ

Günümüzdeki hesaplama problemlerinde kullanılan veri kaynaklarının ürettikleri veri kümeleri her geçen gün hem miktar hem de boyut sayısı açısından artış göstermektedirler. Gerekli boyut sayısını ve her bir boyutun büyüklüğünü belirterek tanımladığımız tensör veri yapıları, herhangi bir hesaplama probleminde kullanılan veriyi büyüklük ya da boyut sınırı olmaksızın ifade edebilmektedirler.

Literatürdeki birçok veri analizi problemi tensörleri veri modeli olarak kullanmaktadır. Tensörleri modelleme aracı olarak kullanan çalışmalar pek çok farklı literatürde yer almaktadır. Örnek olarak yüz tanıma [1], kimya [2], sinyal işleme [3], veri madenciliği [4] sıralanabilir. Bu çalışmaların bir çoğunda tensör ayrışımı yöntemleri kullanılmaktadır[5].

Saklı Tensör Ayrışımı (STA) çerçevesi [6] literatürdeki tensör ayrışımı yöntemlerini genelleştiren bir hesaplama çerçevesidir. STA çerçevesi tensör ayrışımı ile grafik modeller [7] arasındaki benzerliği [8] kullanarak, çok boyutlu tensör ayrışımı problemlerini grafik modellerde kullanılan mesaj iletim yöntemlerine benzer bir şekilde formüle eder. Çerçevenin önerdiği güncelleme yöntemleri verilen modelin tanımladığı saklı tensörleri genel tensör çarpımı operasyonu sonucunda, gözlemlenen tensöre yakınsayan bir tensör oluşturacak şekilde günceller.

STA çerçevesinin çözmeyi hedeflediği problemler yapısal olarak bağımsız bir şekilde tekrar eden ve ağır hesaplama gerektiren veri yapıları içerir. Bu tür problemleri ekran kartlarının genel amaçlı hesaplama araçlarını kullanarak, veriyi paralel bir şekilde işleyen yaklaşımlar popülerleşmektedir [9] [10] [11] [12]. Bu çalışmada STA çerçevesinin en temel operasyonunu paralelleştirerek elde ettiğimiz başarım artışından bahsedeceğiz.

2. SAKLI TENSÖR AYRIŞIMI ÇERÇEVESİ

STA çerçevesi tensörleri çok yollu dizilimler (multi-way array) olarak tanımlar. Dizilimlerin adreslenmesi bir indis kümesi $V = \{i_1, i_2 \dots i_N\}$ aracılığı ile yapılır. Kümedeki her bir indis $i_n = 1 \dots |i_n|$, $n = 1 \dots N$ ayrı ayrı tanımlanır. Burada i_n indisinin alabileceği en büyük değer, $|i_n|$, indisin kardinalitesini tanımlar. İndis kümesi V ve kümedeki indislerin kardinaliteleri tanımlanan bir modeldeki bütün tensörler için ortaktır. Tensörlerin elemanları $T(i_1, i_2 \dots i_n)$ skaler değerlerdir. Bir tensörün indis kümesinden seçilen bir v konfigürasyonu ilgili tensörün skaler elemanlarına ulaşmak için kullanılabilir: $T(v)$. v indis kümesinde bulunmayan indisleri, $\bar{v}$, kullanarak da tensörleri $T(\bar{v})$ şeklinde adreslemek mümkündür. STA çerçevesinin hedefi tensör notasyonunu kullanarak aşağıdaki en-iyileme problemlini çözmektir [6].

$$\text{en-küçültme } D(X||\hat{X}) \; s.t. \; \hat{X} = \sum_{\bar{v}_0} \prod_{\alpha} Z_\alpha(v_\alpha) \qquad (1)$$

Problemin çözümünde gözlemlenen verilen modelde $\alpha = 1 \dots N$ için tanımlanmış Z_α saklı tensörlerinin çıkarımı yapılır. Bu işlemi yaparken gözlemlenen X tensörüne belli bir ıraksaklık ölçütü D'ye göre yakınsayan yakınsama tensörü $\hat{X}$ hesaplanır. Bu çalışma için ıraksaklık ölçütü, Kullback-Leibler ıraksaklığı (KL) olarak seçilmiştir. Problemde tanımlanan X ve $\hat{X}$ tensörleri aynı V_0 indis kümesini paylaşırlar ve ıraksaklık ölçümleri eleman-eleman yapılır. Problemin indis kümesi V ise kullanılacak bütün tensörlerin indis kümelerinin birleşimine, $V = \cup_\alpha V_\alpha$ eşittir.

978-1-4673-0056-8/12/$26.00

KL ıraksaklığı kullanıldığında beklenti-enbüyütme yöntemi ile aşağıdaki güncelleme denklemlerini elde etmek mümkündür[6]. Gözlemlenemeyen saklı tensörler rastsal olarak başlatıldıktan sonra denklem 1'de belirtilen şekilde yakınsama tensörü $\hat{X}$ hesaplanır. Sonrasında her bir saklı tensör için güncelleme işlemi yapılır. Her güncelleme işleminden sonra yakınsama tensörü tekrar hesaplanır.

$$Z_\alpha = Z_\alpha * \frac{\Delta_\alpha(M * X/\hat{X})}{\Delta_\alpha(M)} \tag{2}$$

$$\Delta_\alpha(A) = \sum_{\bar{v}_\alpha} \left(A(v_0) \prod_{\alpha' \neq \alpha} Z_{\alpha'}(v_{\alpha'}) \right) \tag{3}$$

Denklem 2'deki M tensörü gözlemlenen tensör ile aynı boyutlara sahip bir maske tensörü olarak tanımlanmıştır. Eksik veri ile karşılaşılan durumlarda istenen indisler 0 olarak belirtilir. Böylece güncelleme işleminde eksik veri olan indisler dikkate alınmamış olur. Eksik veri durumu göze alınmadığında M tensörünün bütün elemanları 1 olarak seçilir.

Denklem 3'teki Δ_α operasyonu STA çerçevesinin en temel işlemidir. Bu işlem Z_α saklı tensörünü, $\hat{X}$ tekrar hesaplandığında, $D(X||\hat{X})$ ıraksaklığını azaltacak şekilde günceller. Güncelleme işleminin iki alt işlemi vardır. Birincisi bir tensörün belirtilen bir indis üzerinden toplanması ya da diğer adıyla *tensör daraltması işlemi*, ikincisi ise iki tensörün çarpılmasıdır. Diğer bir bakış açısıyla bu iki işlemin tek bir adımda ard arda yapılmasına *genelleştirilmiş tensör çarpımı* adını veriyoruz.

Güncelleme işleminin sağ tarafındaki toplama ifadesini parantezin içerisine dağıtarak hesap miktarını azaltmak mümkündür. Örnek olarak Şekil 1'de verilen literatürde çok kullanılan Tucker tensör ayrışımı modelinin güncelleme işlemlerinden Δ_4'ü aşağıdaki şekilde yazabiliriz: $\Delta_4(A) = \sum_i Z_1(i,z) \sum_j Z_2(j,f) \sum_k Z_3(k,l) A(i,j,k)$. Bu ifadedeki tensör çarpımı işlemlerinin sıralamasını seçerek işlem hızı ve kullanılan hafıza miktarı arasında tercih yapmak [13] da mümkündür. Bu çalışmada güncelleme işlemlerinde çok sayıda kullanılan tensör çarpımı ve daraltması operasyonlarının paralel gerçellemesi anlatılmaktadır.

Aşağıdaki bölümde genelleştirilmiş tensör çarpımının örnekleri ile birlikte detaylı anlatımı mevcuttur. Ardından bu operasyonun paralelleştirilmesi anlatılacaktır.

$$\begin{aligned}
X(i,j,k) &\approx \hat{X}(i,j,k) \\
\hat{X}(i,j,k) &= \sum_{z,f,l} Z_1(i,z) Z_2(j,f) Z_3(k,l) Z_4(z,f,l) \\
A &= M * X/\hat{X} \text{ (bölünen için)} \\
A &= M \text{ (bölen için)} \\
\Delta_1(A) &= \sum_{j,k,f,l} A(i,j,k) Z_2(j,f) Z_3(k,l) Z_4(z,f,l) \\
&\cdots \\
\Delta_4(A) &= \sum_{i,j,k} A(i,j,k) Z_1(i,z) Z_2(j,f) Z_3(k,l)
\end{aligned} \tag{4}$$

Şekil 1: Tucker tensör ayrışımı modeli ve güncelleme işlemleri

3. GENELLEŞTİRİLMİŞ TENSÖR ÇARPIMI

Genelleştirilmiş tensör çarpımı iki adımlı bir işlem yapmamız gereklidir. Öncelikle verilen modelde tanımlanmış çarpım işlemine soktuğumuz iki tensörü kullanarak F *bileşik tensörünü* hesaplarız. Ardından eğer A çıktı tensöründe bulunmayan ancak bileşik tensörde bulunan indisler varsa, bu indisler üzerinden daraltma işlemi uygularız.

Bir başka değişle aşağıdaki modelde tanımlanan B ve C tensörlerini genelleştirilmiş tensör çarpımı işlemine girdi olarak alıp, A çıktı tensörünü hesapladığımızda, matris çarpımı işlemi uygulamış oluruz. Örnek olarak matris çarpımı işlemi $V = \{i,j,k\}$, $V_A = \{i,j\}$, $V_B = \{i,k\}$, $V_C = \{k,j\}$ indis kümelerini seçerek gösterilebilir. Bunu sağlayan genelleştirilmiş tensör çarpımı işleminin çarpım $F(v_F) = B(v_B)C(v_C)$, $V_F = \{i,k,j\}$ ve daraltma $A(v_A) = \sum_{\bar{v}_F} F(v_F)$ işlemleridir.

4. PARALELLEŞTİRME

Veri analizi problemlerinde artan boyut sayısı ve veri miktarı yüksek başarımlı hesaplamaya yapmayı güçleştirir. Hesaplanabilirlik açısından ağır problemlerin çözümünde literatürde çok sayıda bulunan yazılım kütüphanelerinin matris işlevleri sıkça kullanılırlar. Bu işlevler matris veri yapıları üzerinde çalışan yüksek başarımlı hesaplama yöntemlerini kullanarak verilen problemlerin hızlı bir şekilde çözülmesini sağlarlar. Ancak bu en-iyileme yönteminin çok boyutlu problemlerin çözümünde kullanılabilmesi için çok boyutlu modellerin matris veri yapısı ile iki boyutlu olarak yeniden düzenlenmesi gereksinimi ortaya çıkmaktadır. STA çerçevesinin paralel gerçellemesinin hedefi yeniden modelleme sürecine gerek duymadan, belirtilen çok boyutlu modeli kullanarak paralel çalışan ekran kartı mimarileri üzerinde yüksek başarım elde etmektir.

Bu bölümde genel çarpım operasyonunun en temel paralelleştirme fikirleri kullanılarak tasarlanmış gerçellemesi anlatılmaktadır. İlk adım olarak bileşik tensörün, F, her bir elemanı, $F(v_F)$, paralel olarak hesaplanır. Sonrasında (eğer gerekli ise) $A(v_A)$ çıktı tensörünün hesaplanması için bileşik tensör üzerinde paralel tensör daraltması operasyonu gerçekleştirilir. Daraltma operasyonu da çıktı tensörün her bir elemanı için paralel olarak hesaplanır.

İşlemlerin, sonuçta oluşacak tensörlerin her bir elemanı için paralel olarak yapılması sayesinde, hesaplamalar sırasında oluşabilecek hafıza yazma/okuma çakışmaları engellenmiştir. Bu yöntem her ne kadar hafıza kullanımı açısından en iyi algoritma olmasa da, çözümü hesaplanabilirlik açısıdan pahalı olan, global senkronizasyon problemi ile karşılaşmamak için kullanılan çözümlerden birisidir.

Bu çalışmanın temel varsayımı gerçekleştirilen her bir genelleştirilmiş tensör çarpımı işleminde kullanılan iki girdi tensörü, çıktı tensörü ve bileşik tensörü saklamak için GPU üzerinde gerekli hafıza miktarının mevcut olmasıdır.

4.1. Bileşik Tensörün Paralel Hesaplanması

GPU mimarileri nispeten basit ve birbirinden bağımsız işlemleri çok fazla sayıda paralel olarak çalıştırmak üzere

geliştirilmişlerdir. Bu mimarilerin sağladığı avantajları en verimli şekilde kullanabilmek için algoritmaların paralel olarak aynı anda çalıştırılabilir şekilde tasarlanması gereklidir.

Bileşik tensörün paralel hesaplanmasını GPU mimarileri üzerinde gerçekleştirmek için bileşik tensörün her bir elemanını, $F(v_F)$, paralel hesaplayacak bir yöntem tasarlamak uygun olacaktır. Algoritma 1 bileşik tensörün hesaplanmasında kullanılan algoritmayı göstermektedir. Algoritmanın paralelleştirilmesi için *parfor* anahtar kelimesi ile belirtilen bölümün her bir iterasyonunun ayrı bir işlemci birimi tarafından hesaplanması gereklidir. Dolayısıyla her bir işlemci bileşik tensörün bir elemanını hesaplar ve hafıza okuma/yazma çakışması gerçekleşmez.

Çok boyutlu tensör verisi bilgisayar hafızasında tek boyutlu bir dizilim olarak saklanmaktadır. Algoritma 2'deki hafıza_indisi_bul işlevi bir tensörün istenen bir elemanının hafıza üzerindeki adresini hesaplamak için kullanılır. Bu hesaplamayı yapabilmek için tensörün tanımının yanı sıra tensörün herhangi bir indisinde bir sonraki elemana erişmek istediğimizde bilgisayar hafızasındaki dizi üzerinde kaç eleman ilerlenmesi gerektiği bildiren *kaydırma listesine* (stride list), σ, ihtiyacımız vardır. Örnek olarak tanımlayabileceğimiz bir A tensörünün indis kümesi $V_A = \{i, j, k\}$ ve $|i| = 2$, $|j| = 3$, $|k| = 4$ ise bu tensör için $\sigma_A = [1, |i|, |i| * |j|] = [1, 2, 6]$ olur. Bu notasyonu kullanarak herhangi bir indis konfigürasyonun hafıza üzerinde hangi adreste bulunduğunu hesaplamamak mümkündür. Aşağıda adres hesaplama işlemi örnekleri verilmiştir.

İndis konfigürasyonu	$v_A . * \sigma_A$	Adres
$v_A = \{0, 1, 3\}$	$0 * 1 + 1 * 2 + 3 * 6$	20
$v_A = \{0, 2, 3\}$	$0 * 1 + 2 * 2 + 3 * 6$	22
$v_A = \{0, 2, 4\}$	$0 * 1 + 2 * 2 + 4 * 6$	28

Anlatılan notasyonu kullandığımızda F bileşik tensörünün hesaplanması, Algorima 1'de gösterildiği gibi, B, C girdi tensörlerinin eşlenen elemanlarının indislerini hafıza_indisi_bul işlevi ile hesaplayıp, ilgili değerleri çarpmak ve sonucu F çıktı tensörünün ilgili elemanına kaydetmekten ibarettir.

4.2. Çıktı Tensörün Hesaplanması

İkinci adımda tam tensör kullanılarak çıktı tensör hesaplanır. Çıktı tensörü ile tam tensörün kullandıkları indislerin aynı olması durumunda bu adım atlanır, zaten çıktı tensörü tam tensöre eşittir. Aksi halde tam tensörde bulunan ve çıktı tensöründe bulunmayan indisler üzerinden toplama işlemi yapılır.

Çıktı tensörün elemanlarının her birisi de paralel olarak hesaplanır. Algoritma 3 çıktı tensörünün paralel olarak hesaplanması için gerekli adımları göstermektedir. İlk olarak daraltma indisleri hesaplanır. Daraltma işlemi gerçekleştirilmesi gereken indislerin, $V_F \setminus V_C$, alabileceği bütün indis değerleri kartezyen çarpımı ile bulunur. Daraltma işleminde her bir C[indis_C] çıktı tensör elemanının hesaplanması için, F bileşik tensörünün toplanacak elemanlarının indislerinin hesaplanması gereklidir. Bunun için C tensörünün verilen bir j elemanının daraltılan indisler D_i ile beraber kullanılarak, $j \oplus i$, F bileşik tensörünün adreslenmesi gereklidir. Bu toplama işlemi seri olarak gerçekleştirilmektedir. Sonraki çalışmalarımızda bu işlem de paralelleştirilecektir[14].

Algoritma 1 Bileşik tensörün hesaplanması

F.elnum $= \prod_{i \in V_F} |V_i|$
parfor $j = 0 : F$.elnum **do**
  indis_F = hafiza_indisi_bul(j, F)
  indis_B = hafiza_indisi_bul(j, B)
  indis_C = hafiza_indisi_bul(j, C)
  F[indis_F] $= B$[indis_B] $* C$[indis_C]
end parfor

Algoritma 2 hafiza_indisi_bul(elnum, T)

// T: bir tensör
// elnum: adresi bulunacak tensör elemanının sıra numarası
hafiza_indis $= 0$
for boyut $=$ boyut_say -1;boyut$>= 0$;boyut $=$ boyut -1 **do**
  if elnum / $T.\sigma$[boyut] > 0 **then**
    gecici $=$ elnum / $T.\sigma$[boyut]
    elnum $=$ elnum $-$ gecici $* T.\sigma$[boyut]
  else
    gecici $= 0$
  end if
  hafiza_indis $=$ hafiza_indis $+$ gecici $* T.\sigma$[boyut]
end for
return hafiza_indis

Algoritma 3 Çıktı tensörünün hesaplanması

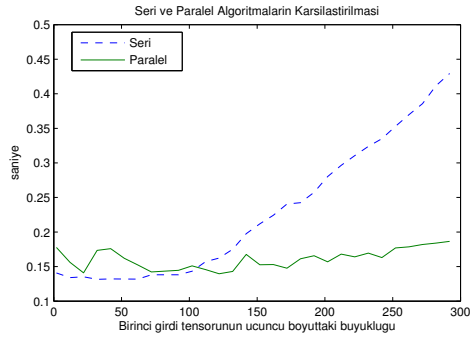
daraltma_indisleri $= V_F \setminus V_C$
for $i \in$ daraltma_indisleri **do**
  $D_i = \{1 \dots |i|\}$
end for
$D = D_1 \times \cdots \times D_n$
C.elnum $= \prod_{i \in V_c} |i|$
parfor $j = 0 : C$.elnum-1 **do**
  gecici $= 0$
  for all i $\in D$
    indis_F = hafiza_indisi_bul$(j \oplus i, F)$
    gecici $=$ gecici $+F$[indis_F]
  end for
  indis_C= hafiza_indisi_bul(j, C)
  C[indis_C]=gecici
end parfor

5. SONUÇLAR ve VARGILAR

Paralelleştirmenin sonuçlarını görebilmek için genel çarpım operasyonu hem işlemci üzerinde seri olarak çalışan hem de ekran kartı üzerinde paralel olarak çalışan sürümleri gerçellenmiştir.

Aşağıda tanımlanan modeldeki A tensörünün m boyutundaki büyüklüğü değiştirdiğimizde paralel ve seri sürümlerde gerçekleşen başarım değişikliğini Şekil 2 göstermektedir. Test modeli: $V = \{i, k, j, m\}$, $V_C = \{i, m\}$, $V_A = \{i, j\}$, $V_B = \{i, k, j\}$, $|i| = |k| = |j| = 100$.

Şekilde görülebileceği gibi m boyutundaki büyüklük 100'den az olduğu sürece seri çalışan işlemci sürümünün başarımı ekran kartında çalışan paralel sürümünün başarımına kıyasla daha yüksektir. Ancak bu değerden sonra işlemcide çalışan seri operasyonunun aldığı süre doğrusal olarak artmaya



Şekil 2: Seri and paralel algoritmaların karşılaştırılması

başlamaktadır. Bu kırılma noktası bize aynı zamanda veriyi ekran kartına gönderme için harcanan sürenin amorti edildiği noktayı göstermektedir.

Aşağıdaki tablo literatürde çok kullanılan PARAFAC modelinin STA çerçevesi kullanılarak gerçellenmiş sürümü ile popüler tensör hesaplama kütüphanelerinden Tensor Toolbox v2.5 [15] kullanılarak gerçellenmiş sürümününde iterasyon sayısı arttıkça değişen çalışma sürelerini saniye cinsinden göstermektedir. Kullanılan modelin boyutları aşağıda verilmiştir: $V_0 = \{i, j, k\}$, $V_A = \{i, a\}$, $V_B = \{j, a\}$, $V_C = \{k, a\}$, $|i| = 10$, $|j| = 11$, $|k| = 12$, $|a| = 20$. Tensor Toolbox'ın kullandığı yöntem (Dalgalı En Küçük Kareler - Alternating Least Squares) çok farklı olsa da bu tablo paralel algoritmamızın başarımını endüstri standardı ile karşılaştırmasını göstermektedir. Sonuçların analizinde önemli bir nokta STA çerçevesinin eksik veri ile karşılaşılan problemlerde de çalışabiliyor olmasıdır. Bütün testler Intel Xeon E5530 işlemcili NVIDIA Quadro FX 4800 ekran kartına sahip bir sistem üzerinde gerçekleştirilmiştir.

İterasyon	Seri STA (sn)	Paralel STA (sn)	Tensor Toolbox (sn)
100	127.1	10.1	2.5
200	255.4	19.8	4.1
300	376.6	29.3	5.8

Bu sonuçlar STA çerçevesinin paralelleştirme sonucu daha yüksek başarımlı bir şekilde çalışacağını göstermiştir. Yeterince büyük tensörler söz konusu olduğunda her ne kadar gelişmiş olsalar da seri çalışmaları sebebiyle, ana işlemcilerin (CPU) başarımları girdi objelerin eleman sayısına doğrusal oranda düşmektedir.

6. GELECEK ÇALIŞMALAR

Bileşik tensörün hesaplanmadan daha küçük ara tensörleri kullanarak sonuca ulaşmak genelleştirilmiş çarpımın paralel gerçellemesinin daha hızlı çalışmasını sağlamak sonraki çalışmalarımızın konusu olacaktır.

7. TEŞEKKÜR

Bu çalışma Türkiye Bilimsel ve Teknik Araştırmalar Kurumu (TÜBİTAK) tarafından 110E292 nolu araştırma projesi ve Boğaziçi Araştırma Projeleri (BAP) tarafından P5723 nolu araştırma projesi kapsamında desteklenmektedir.

8. KAYNAKÇA

[1] M. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," *Computer Vision—ECCV 2002*, pp. 447–460, 2002.

[2] AK Smilde, "Theory of medium-rank second-order calibration with restricted-Tucker models," *Journal of chemometrics*, 1994.

[3] N.D. Sidiropoulos, R. Bro, and G.B. Giannakis, "Parallel factor analysis in sensor array processing," *Signal Processing, IEEE Transactions on*, vol. 48, no. 8, pp. 2377–2388, 2000.

[4] B. Savas, "Analyses and tests of handwritten digit recognition algorithms," *LiTH-MAT-EX-2003-01, Linköping University, Department of Mathematics*, 2003.

[5] T.G. Kolda and B.W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[6] Y.K. Yilmaz and A.T. Cemgil, "Probabilistic latent tensor factorization," in *Proceedings of the 9th international conference on Latent variable analysis and signal separation*. Sept. 2010, pp. 346–353, Springer-Verlag.

[7] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2007.

[8] C. Fevotte and A.T. Cemgil, "Nonnegative matrix factorizations as probabilistic inference in composite models," in *Proc. EUSIPCO*, 2009, vol. 47, pp. 1913–1917.

[9] M.A. Suchard, C. Holmes, and Mike West, "Some of the What?, Why?, How?, Who? and Where? of Graphics Processing Unit Computing for Bayesian Analysis," *ISBA Bull*, vol. 17, pp. 12–16, 2010.

[10] Lu Zheng, O.J. Mengshoel, and J. Chong, "Belief Propagation by Message Passing in Junction Trees: Computing Each Message Faster Using GPU Parallelization," in *Proc. of the 27th Conference on Uncertainty in Artificial Intelligence*, 2011.

[11] Wenjing M., S. Krishnamoorthy, O. Villay, and K. Kowalski, "Acceleration of Streamed Tensor Contraction Expressions on GPGPU-Based Clusters," in *International Conference on Cluster Computing*, 2010, pp. 207–216.

[12] Hyeran Jeon, Yinglong Xia, and V.K. Prasanna, "Node Level Primitives for Exact Inference using GPGPU," *Int. Conf. on Systems Signals and Image Processing*, 2010.

[13] G. Baumgartner, Auer, D.E. Bernholdt, Bibireata, V. Choppella, Cociorva, and et al, "Synthesis of High-Performance Parallel Programs for a Class of ab Initio Quantum Chemistry Models," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 276–292, Feb. 2005.

[14] Hubert Nguyen, *GPU Gems 3*, Addison-Wesley Professional, 2007.

[15] Tamara G. Kolda Brett W. Bader et al., "Matlab tensor toolbox version 2.5," Available online, January 2012.