Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi
Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering

Basılı/Printed ISSN: 1302-9304. Elektronik/Online ISSN: 2547-958X

Duff Aygıtı Tabanlı Seyrek Matris-Vektör Çarpımı

Sparse Matrix-Vector Multiplication Based on Duff's Device

Barış Aktemur [1*]

[1] Özyeğin Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul, TÜRKİYE
Sorumlu Yazar / Corresponding Author *: aktemur@gmail.com

Geliş Tarihi / Received: 26.01.2019 *Araştırma Makalesi/Research Article*
Kabul Tarihi / Accepted: 13.11.2019 *DOI: 10.21205/deufmd.2020226501*
Atıf şekli/ How to cite: AKTEMUR, B.(2020). Duff Aygıtı Tabanlı Seyrek Matris-Vektör Çarpımı DEUFMD 22(65), 315-324.

Öz

Seyrek matris-vektör çarpımı (SpMV) pek çok mühendislik probleminde ve bilimsel hesaplamada sıklıkla kullanılan bir işlemdir. SpMV'nin hızlandırılması geniş bir yelpazedeki uygulamaları olumlu etkiler. Bu makalede Duff aygıtı olarak bilinen döngü açılımının SpMV'nin başarımına etkisini irdeliyoruz. Önerdiğimiz Duff aygıtı tabanlı SpMV gerçeklemesi, en geçerli seyrek matris saklama formatı olan CSR formatının düşük maliyetli bir ön işlemesi sonrası kullanılabilmektedir. Gerçek problemlerde kullanılan matrislerden oluşan veri kümesi ile deneysel bir değerlendirme yaptık ve önemli derecede hızlanma kaydedilebileceğini gözlemledik.

Anahtar Kelimeler: *Seyrek matris-vektör çarpımı, yüksek başarımlı hesaplama, Duff aygıtı*

Abstract

Sparse matrix-vector multiplication (SpMV) is used frequently in several engineering problems and scientific computation. Optimizing SpMV positively impacts a wide range of applications. In this paper, we investigate the effect of a loop-unrolling method known as Duff's device on the performance of SpMV. The Duff's device-based SpMV implementation that we propose can be used after a low-cost preprocessing of the CSR representation – the most common sparse matrix storage format – of a matrix. We have evaluated the approach on a dataset consisting of matrices from real-world problems, and observed that substantial speedup can be achieved.

Keywords: *Sparse matrix-vector multiplication, high performance computing, Duff's device*

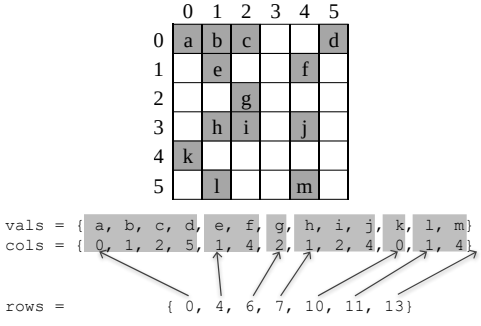
1. Giriş

Seyrek matrisler, sonlu eleman modelleri, bilimsel benzetimler, makine öğrenmesi uygulamaları gibi pek çok bağlamda karşımıza çıkan, elemanlarının önemli bir kısmı sıfır olan matrislerdir. Seyrek matrislerle en sık yapılan işlemlerden birisi matrisin bir vektörle çarpımıdır. Özellikle yinelemeli çözümleyiciler için bu işlem kilit konumdadır. Bu nedenle seyrek matris-vektör çarpımının (SpMV) eniyilemesi üzerine yoğun araştırma yapılagelmiştir (örn. [1–10]).

Sıfır olan elemanların çarpma işleminin sonucuna bir katkısı olmadığı için ve de bellek alanından tasarruf etmek adına, seyrek matrisler sadece sıfır olmayan elemanlarının tutulduğu formatlarda saklanırlar. Bu formatların en yaygını *Compressed Sparse Row* (CSR) adı verilendir (*Compressed Row Storage* – CRS olarak da bilinir) [11]. Bu, genel-geçer bir formattır; bütün seyrek matrisler bu formatta tutulabilir. Doğrusal cebir kütüphanelerinin büyük çoğunluğu seyrek matris girdilerini CSR

formatında kabul ettiklerinden, bu format *defakto* standart konumundadır.

Örnek bir seyrek matris ve onun CSR gösterimi Şekil 1'de verilmiştir. CSR formatında matrisin sıfır olmayan elemanları `vals` adıyla anacağımız bir dizide tutulurlar; `cols` olarak anacağımız ikinci bir dizide ise bu elemanların sütun indisleri yer alır. Sıfır olmayan eleman sayısı NZ ise, bu iki dizinin uzunluğu da NZ'dir. Son olarak, `rows` ismiyle anacağımız üçüncü bir dizi ise her bir satırda yer alan elemanların `vals` ve `cols` dizilerinde hangi indisler arasında yer aldığı bilgisini tutar. Öyle ki, ardışık `rows` değerleri arasındaki fark, karşılık gelen matris satırının uzunluğunu verir. Yani, *i*'ninci satırın uzunluğu `rows`[*i*+1]-`rows`[*i*] ifadesi ile bulunabilir. Matrisin satır sayısı N ise, rows dizisinin uzunluğu N+1'dir.



Şekil 1. Seyrek bir matris ve CSR gösterimi

CSR formatı kullanarak yapılan SpMV işleminin kodu ise Şekil 2'de gösterilmektedir (C/C++ dilinde). Burada `x` girdi vektörü, `y` çıktı vektörüdür.

```
void spmv(double *vals, int *cols,
          int *rows, int N,
          double *x, double *y) {
  for(int i = 0; i < N; i++) {
    double sum = 0.0;
    int j;
    for(j=rows[i]; j < rows[i+1]; j++)
      sum += vals[j] * x[cols[j]];
    y[i] += sum;
  }
}
```

Şekil 2. CSR formatı ile yapılan SpMV'nin kodu

SpMV'de genel olarak yüksek başarım elde etmek zordur. Bunun nedeni, matris elemanlarının düzensiz bir şekilde konumlanmış olması ve `x` vektörünün elemanlarına erişimin dolaylı olmasıdır. Bu düzensiz ve dolaylı erişim, işlemci mimarilerinde başarımı etkileyen ana unsurlardan olan önbelleğin kullanımı, verinin önceden yüklenmesi gibi konularda verimsizliğe neden olmaktadır; ayrıca az elemanlı satırlarda iç döngünün yineleme sayısı çok düşük kalacağından dallanma kestirimi mekanizması etkisini gösterememektedir [4]. Belli eleman örüntülerine sahip birtakım seyrek matriste bu sorunların üstesinden gelmek ve başarımı artırmak için CSR'den başka matris saklama formatları kullanılabilir. Ancak CSR'den bu formatlara dönüşüm koşut-zamanda vakit kaybına neden olmakta ve çoğu zaman astarı yüzünden pahalıya gelmektedir. Pratikte, özel durumlara uygun çoğu format sadece matris örüntüsü koşut-zamandan önce (yani durağan zamanda) belli ise faydalı olabilmektedir. Bu nedenle koşut-zaman maliyeti düşük olan yöntemlere ihtiyaç duyulmaktadır [12–17].

Bu makalede, Duff aygıtı olarak adlandırılan döngü açılımı marifetinin SpMV işleminin başarımına etkisini irdeliyoruz. Duff aygıtı yöntemiyle SpMV kodundaki iç döngüyü açıyor, ayrıca matrisin `rows` dizisi üzerinde bir veri küçültme işlemi yapıyoruz. Bu işlem sadece `rows` dizisi üzerinde değişiklik gerçekleştirdiğinden CSR'den dönüşüm maliyetleri düşük olmaktadır.

Bölüm 2'de Duff aygıtını ve SpMV kodunda nasıl kullanılabileceğini anlatıyoruz. Bölüm 3'te deneysel değerlendirmemizi sunuyoruz. Bölüm 4'te ilgili diğer çalışmaları değerlendiriyor, Bölüm 5'te makalemizi sonlandırıyoruz.

2. Yöntem

Döngü açılımı derleyiciler tarafından sıklıkla uygulanan eniyilemelerden biridir. Amaç, bir yanda döngünün sayaç masrafını azaltmak, bir yandan da tekrarlanan döngü vücudunda daha fazla eniyileme imkânı ortaya çıkarmaktır. Ancak bir dezavantajı vardır, o da döngünün yineleme sayısı açılım miktarının tam katı olmadığı durumlarda arta kalan yinelemeleri bitirmek için ikinci bir döngü eklemek gerekliliğidir. Şöyle ki, Şekil 2'de bulunan iç döngüyü 4 kere açmak istediğimizde aşağıdaki gibi bir kod elde ederiz:

```
int j = rows[i];
for (; j < rows[i+1] - 3; j += 4){
  sum += vals[j]   * x[cols[j]];
  sum += vals[j+1] * x[cols[j+1]];
  sum += vals[j+2] * x[cols[j+2]];
  sum += vals[j+3] * x[cols[j+3]];
}
for (; j < rows[i+1]; j++)
  sum += vals[j] * x[cols[j]];
```

Burada, ikinci döngü ek kontrol yükü getirmektedir. Kısa matris satırları için bu yükler hissedilebilir seviyededir. İkinci döngüden kurtulmak adına literatürde Duff aygıtı [18] (*Duff's device*) olarak bilinen kodlama marifeti uygulanabilir. Bu yönteme göre

```
do
  s
while(--count > 0);
```

biçimindeki bir döngü, şu şekilde 4 kere açılabilir:

```
int trips = (count + 3) / 4;
switch (count % 4) {
  case 0: do { s;
  case 3: s;
  case 2: s;
  case 1: s;
  } while(--trips > 0);
}
```

Burada uygulanan marifet, *do-while* döngüsü ile *switch-case* komutlarını iç-içe yazarak döngüye ilk giriş noktasının döngü vücudunun ortasında bir yer olmasını sağlamaktır.

Duff aygıtından ilham alarak SpMV kodunu yeniden yazalım. Bunu iki sürüm halinde yapacağız. Her bir sürümü aşağıda bir alt başlıkta inceliyoruz.

Duff aygıtının SpMV için kullanılması fikri yeni değildir. Bildiğimiz kadarıyla daha önce sadece Youssefi'nin yüksek lisans tez çalışmasında incelenmiştir [19]. Duff aygıtının özgün hali döngünün en az bir kere çalışacağı varsayımına göre yazılmıştır. Youssefi'nin tezinde de boş satır içermeyen, bantlı yapıdaki 13 matris için denemeler yapılmıştır; kullanılan kod boş satır içeren matrisler için hata vermektedir. Ancak genel olarak seyrek bir matrisin hiç elemanı olmayan satırlarının olması mümkündür. Makalemizde verdiğimiz kodlar bu durum gözetilerek yazılmıştır. Bölüm 2.2'de sunduğumuz *DuffCompressed*$_k$ yöntemi tamamen özgündür; bildiğimiz kadarıyla daha önce hiçbir çalışmada irdelenmemiştir.

2.1. Birinci sürüm: *DuffCSR*$_k$

Yukarıdaki Duff aygıtından ilham alarak, Şekil 2'deki SpMV kodunu Şekil 3'teki gibi yazabiliriz. Her ne kadar Şekil 3'te verdiğimiz kod, örnek teşkil etmesi açısından, döngünün 4 kere açılım uygulanmış hali olsa da bunu rahatlıkla genelleyebiliriz. Döngüye *k* kere açılım yapılması ile elde edilen koda *DuffCSR*$_k$ diyelim. Bu durumda Şekil 3'te verdiğimiz kod *DuffCSR*$_4$ olmaktadır.

```
// Beklenen matris formatı: CSR
for (int i = 0; i < N; i++) {
  double sum = 0.0;
  int length = rows[i + 1] - rows[i];
  int trips = length / 4;
  int entry = length % 4;
  vals += entry;
  cols += entry;

  switch (entry) {
    do {  vals += 4; cols += 4;
          sum += vals[-4]*x[cols[-4]];
  case 3: sum += vals[-3]*x[cols[-3]];
  case 2: sum += vals[-2]*x[cols[-2]];
  case 1: sum += vals[-1]*x[cols[-1]];
  case 0: ;
    } while (--trips >= 0);
  }
  y[i] += sum;
}
```

Şekil 3. Duff aygıtı ile yazılan *DuffCSR*$_4$ yöntemi

DuffCSR$_k$ kodu ile CSR formatı hiç değiştirilmeden SpMV yapılabilmektedir. Youssefi'nin tezinde ayrıca Duff aygıtı yaklaşımının LCSR seyrek matris formatı [20] kullanan hali de incelenmiştir. LCSR formatında matrisin elemanları yeniden sıralanmaktadır. Bu nedenle CSR formatından LCSR elde etmek için hem `vals` hem de `cols` dizilerinin yeniden oluşturulması gerekir. Bu durum en baştan belirlediğimiz "düşük maliyet" kısıtına uymadığı için LCSR tabanlı yaklaşımları kapsam dışı tutuyoruz.

2.2. İkinci sürüm: *DuffCompressed*$_k$

DuffCSR$_k$ yönteminde her bir satırın uzunluğunu buluyor, sonra da bu değeri kullanarak döngü vücuduna giriş noktasını (`entry`) ve döngü yineleme sayısını (`trips`) hesaplıyoruz. Bu hesaplamalardan kurtulmak adına `entry` ve `trips` değerlerini bir ön işleme safhasında önceden hesaplayıp `rows` dizisinde tutabiliriz. Üstelik bunu daha düşük boyut kullanarak yapabiliriz. Şöyle ki, `entry` değeri her zaman *k*'den küçük olacaktır; `trips`'in alabileceği en yüksek değer ise matrisin azami satır uzunluğu bölü *k* kadardır. Bu değerler yeterince küçükse, 4 baytlık `int` değerleri tutmak yerinde 1 veya 2 baytlık iki değer tutabiliriz. Böylece veri boyutunu zahmetsiz bir şekilde küçültme imkânından faydalanmış oluruz.

Bir matrisin azami satır uzunluğuna ASU diyelim. ASU/*k* değeri 2^8'den küçük ise, olası tüm `trips`

değerleri 1 baytlık alanda gösterilebilir. Benzer şekilde, *k* değeri 2^8'den küçükse, tüm `entry` değerleri 1 baytlık alanda tutulabilir. Bu durumda matrisin her bir satırı için iki adet `unsigned char` verisi tutarız. Orijinal `rows` dizisinde ise her bir satır için 4 baytlık bir `int` değer tutuluyordu. (`unsigned char` veri tipi günümüz işlemcilerinin çoğunda, örn: X86 mimarisi, 1 bayt iken, `int` veri tipi 4 bayttır.) Böylece `rows` dizisi için gereken alan yarıya indirilmiş olacaktır. Bu dönüşümü CSR formatındaki `rows` dizisini Şekil 4'teki gibi ön işlemeden geçirerek yapabiliriz. Dikkat ediniz ki bu dönüşümde orijinal `rows` dizisi üzerinde değişiklik yapıyoruz, ancak istenirse yeni bir alan da açılabilir.

```
unsigned char *R
     = (unsigned char *)rows;
for (int i = 0; i < N; i++) {
 int length = rows[i + 1] - rows[i];
 R[i*2] = (unsigned char)(length/k);
 R[i*2+1] = (unsigned char)(length%k);
}
```

Şekil 4. CSR formatındaki `rows`'un ön işlenmesi

Yeni `rows` dizisini kullanarak SpMV işlemini Şekil 5'teki gibi yapabiliriz. Bu yönteme *DuffCompressed*$_k$ diyoruz. *DuffCSR*$_k$ yönteminin koduna kıyasla değişim gösteren yerler **koyu** yazı tipi ile gösterilmiştir.

```
unsigned char *rowPtr = rows;
for (int i = 0; i < N; i++) {
  double sum = 0.0;
  unsigned char trips = *rowPtr++;
  unsigned char entry = *rowPtr++;
  vals += entry;
  cols += entry;
  ... // Gerisi Şekil 3 ile aynı
```

Şekil 5. Ön işlemeden geçirilmiş `rows` dizisi kullanan *DuffCompressed*$_4$ kodu

Eğer ASU/*k* değeri 2^8'den küçük değil ama 2^{16}'dan küçükse, `entry` ve `trips` değerlerini tutmak için `unsigned char` yerine 2 baytlık `unsigned short` veri tipi kullanırız. (Burada `entry` değerlerini `unsigned char`, `trips`'i ise `unsigned short` olarak tutarak 1 + 2 bayt tarzında bir veri düzenini hafıza hizalama sorunlarına yol açacağı için kullanmadık.) Bu durumda her bir satır için 2 + 2 = 4 baytlık alan kullanacağımız için yerden kazanç sağlamayız, fakat yine de `entry` ve `trips` değerlerini bölme ve kalan alma işlemleri ile hesaplamak yerine hazır hesaplanmış olarak okuma imkânımız vardır. Eğer ASU/k değeri 2^{16}'dan da küçük değilse *DuffCompressed*$_k$ yönteminde `entry` ve `trips` değerlerini önceden hesaplamıyoruz; yeni `rows` dizisinde sadece satır uzunlarını (`length`) saklıyor, `entry` ve `trips`'i SpMV çalışırken `length` değerinden hesaplıyoruz.

3. Bulgular

Bu bölümde Duff aygıtı tabanlı SpMV gerçeklemesinin başarımını değerlendiriyoruz. Bunun için oluşturduğumuz bir veri kümesindeki her bir matris için SpMV yöntemlerini çalıştırıp sürelerini ölçtük. Deneyleri iki farklı bilgisayarda yaptık. Aşağıda deney detaylarını veriyor ve elde edilen sonuçları tartışıyoruz.

3.1. Dayanak yöntemler

Duff aygıtı tabanlı SpMV gerçeklememizin başarımını ölçmek için referans noktası olarak öncelikle CSR yöntemini kullanacağız. Buna ek olarak ikinci bir yöntem daha kullanacağız. Hatırlayalım ki, *DuffCompressed*$_k$ yönteminde `rows` dizisi için kullanılan veri alanını küçültme imkanını değerlendiriyoruz. *DuffCompressed*$_k$ yöntemiyle elde edilebilecek hızlanmanın ne kadarı veri küçültmesinden, ne kadarı ise Duff aygıtı kullanımından kaynaklanıyor, bunu anlayabilmek adına, *CSRLen* ismini verdiğimiz, CSR'ye çok benzeyen ve veri küçültme kullanan bir yöntemi daha kıyaslama için kullanacağız. Bu yöntemde öncelikle `rows` dizisini aşağıdaki ön işlemeden geçirip her satırın uzunluğunu buluyor, rows dizisinde bu değerleri tutuyoruz.

```
T *R = (T *)rows;
for (int i = 0; i < N; i++) {
  int length = rows[i+1] - rows[i];
  R[i] = (T)length;
}
```

Burada `T` veri tipi olarak, eğer azami satır uzunluğu 2^8'den küçükse `unsigned char`, 2^8 ile 2^{16} arasındaysa `unsigned short`, o da değilse `unsigned int` kullanıyoruz. Bu değişiklik sonucu CSRLen yöntemi Şekil 6'daki gibi yapılır.

CSRLen'de, *DuffCompressed*$_k$ yöntemindeki gibi, hatta daha üstün, alan kazanımları elde edilebilir. Bu sebeple, *DuffCompressed*$_k$ yönteminin başarımını CSRLen ile kıyaslayarak başarımın ne kadarının veri küçülmesinden ne kadarının Duff aygıtı yaklaşımından kaynaklandığı hakkında fikir yürütebiliriz.

```
int j = 0;
for (int i = 0; i < N; i++) {
  double sum = 0.0;
  const T length = rows[i];
  for (T k = 0; k < length; k++, j++)
    sum += vals[j] * x[cols[j]];
  y[i] += sum;
}
```

Şekil 6. CSRLen formatı ile yapılan SpMV

CSRLen yöntemi ICRS formatına [10,21] benzerlik göstermektedir. CSRLen formatında ardışık değerler arasındaki farkın tutulması sadece `rows` dizisine uygulanırken, ICRS formatında hem `rows` hem de `cols` dizisine uygulanmaktadır. Düşük maliyet politikamız gereği `rows` dizisinden çok daha uzun olması beklenen `cols` dizisini ön işlemeden geçiren ICRS yöntemini kapsam dışı tuttuk.

3.2. Deney kurulumu

Deneyleri iki farklı bilgisayarda gerçekleştirdik. Bilgisayarların özellikleri Tablo 1'de verilmiştir. Kodları derlemek için bilgisayarda mevcut derleyiciyi `-O3` eniyileme seviyesinde kullandık.

Tablo 1. Deney bilgisayarları.

İşlemci	Önbellek (Bayt)			Bellek	Derleyici
	L1 (I/D)	L2	L3		
AMD FX 8350 *PileDriver*	64K/16K	2M	8M	8GB	gcc 5.4.0
Intel Xeon E5-2620 *SandyBridge*	32K/32K	256K	15M	16GB	icc 17.0.0

Deneyleri Tablo 2'de listelenen matrisleri kullanarak gerçekleştirdik. Bu matrisler başka çalışmalarda da veri kümesi içinde kullanılmıştır [1,7,9,12]; özellikleri geniş bir yelpazeye yayılmaktadır. Tabloda N matrisin satır ve sütun sayısını (tüm matrisler karedir), NZ sıfır olmayan eleman sayısını, NZ/N ortalama satır uzunluğunu (yani satır başına ortalama eleman sayısını), ASU ise azami satır uzunluğunu vermektedir. Matrislerin tümü SuiteSparse (eski adıyla University of Florida) matris koleksiyonundan indirilebilir [22].

Her bir matris için CSR, CSRLen, $DuffCSR_k$, $DuffCompressed_k$ yöntemleriyle SpMV işlemi yaptık ve aldıkları süreyi ölçtük. Duff tabanlı yöntemlerde k değeri olarak 4, 8, 16 kullandık. Ölçümleri şu şekilde gerçekleştirdik: Her bir matris için ölçüm yapılacak SpMV yöntemini bir döngü içinde çok defa yineledik (yineleme sayısı matris büyüklüğüne göre belirlenmektedir). Geçen zamanı ölçtük ve yineleme sayısına bölerek tek bir SpMV için geçen ortalama süreyi bulduk. Bunu 3 defa tekrarladık ve en düşük süreyi kaydettik. Böylece olası veri gürültüsünü asgari hale getirmeyi amaçladık. Her bir SpMV yönteminin başarımını CSR yöntemine göreceli hale getirdik. Bunun için, CSR yönteminin aldığı süreyi o yöntemin aldığı süreye böldük. Bu nedenle, başarım değeri 1'den büyük olduğunda CSR'ye kıyasla hızlanma kaydedildiğini, 1'den küçükse yavaşlama görüldüğünü anlıyoruz.

3.3. Sonuçlar

AMD işlemcili deney bilgisayarında ölçülen başarım değerleri Şekil 7'de, Intel işlemcili bilgisayarda alınan sonuçlar ise Şekil 8'de yer almaktadır.

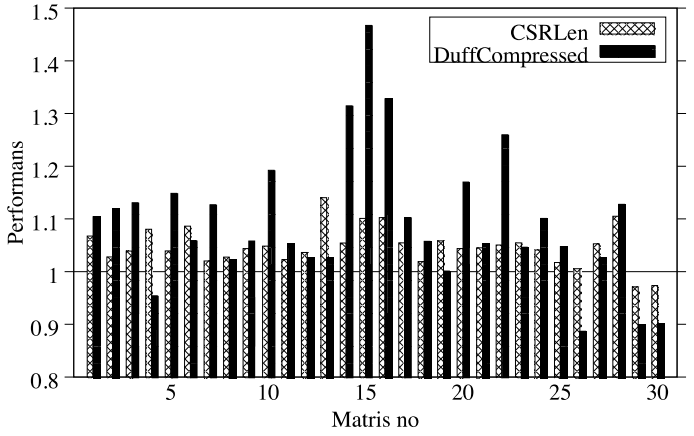
Genel olarak, $DuffCompressed_k$ yöntemi $DuffCSR_k$'den, k=4 durumu ise k=8 ve k=16 durumlarından daha iyi sonuçlar verdiğinden dolayı, sunumun sadeliği ve okunaklığını artırmak adına Şekil 7 ve 8'de Duff tabanlı yöntem olarak sadece $DuffCompressed_4$'ün başarımını gösteriyoruz. Bu yöntemde dört matris için (ASIC_680k, circuit5M, FullChip ve ins2) ASU/k değeri 2^{16}'dan büyüktür. Diğer matrislerden sekiz tanesi için (dc2, eu-2005, in-2004, Stanford, Stanford_Berkeley, webbase-1M, wikipedia-20051105, wikipedia-20060925) ASU/k değeri 2^8 ile 2^{16} arasında, kalan 18 matris içinse 2^8'den küçüktür.

AMD işlemcili bilgisayarda $DuffCompressed_4$ yöntemi 4 matris için CSR yönteminden daha kötü başarım göstermiş, diğer matrisler için hızlanma sağlamıştır. En düşük başarım 0,88x, en yüksek başarım 1,47x, ortalama başarım ise 1,09x olarak görülmektedir. Aynı bilgisayar üzerinde CSRLen yöntemi ile elde edilen başarımın en yükseği 1,14x, en düşüğü 0,97x, ortalama ise 1,04x olmuştur.

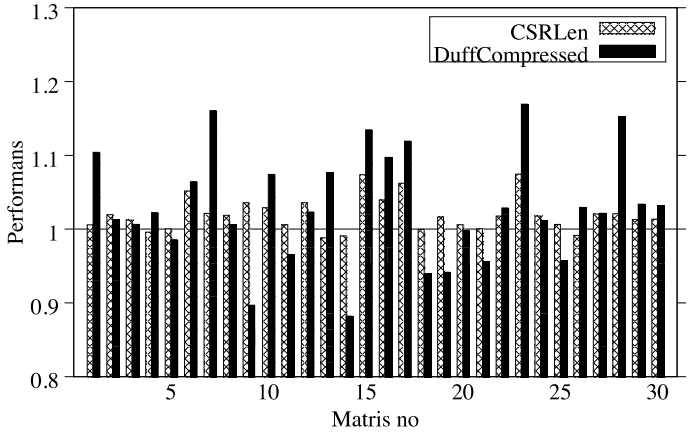
Intel işlemcili bilgisayarda genel olarak daha düşük hızlanmalar görülmüştür. $DuffCompressed_4$ yöntemi 21 matris için hızlanma kaydetmiştir. En düşük başarım 0,88x, en yüksek başarım 1,17x, ortalama başarım ise 1,03x olmuştur. CSRLen yöntemi ile alınan en yüksek başarım 1,07x, en düşük 0,98x, ortalama ise 1,02x'tir.

Tablo 2. Başarım ölçümlerinde kullandığımız matrisler.

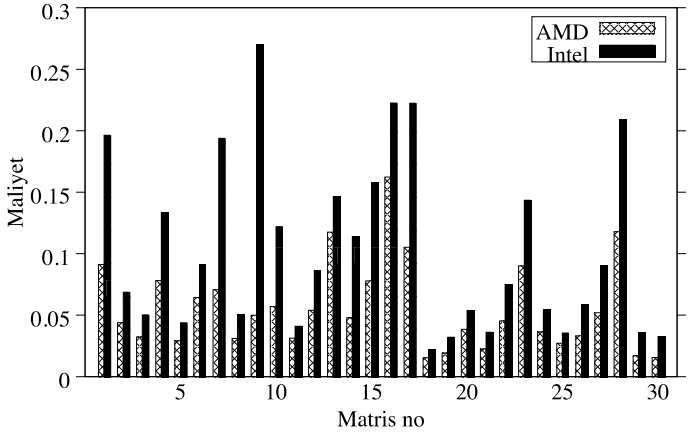
No	Matris	N	NZ	NZ/N	ASU
1	ASIC_680k	682.862	3.871.773	5.7	395.259
2	cage14	1.505.785	27.130.349	18.0	41
3	cant	62.451	2.034.917	32.6	40
4	circuit5M	5.558.326	59.524.291	10.7	1.290.501
5	consph	83.334	3.046.907	36.6	66
6	cop20k_A	121.192	1.362.087	11.2	24
7	dc2	116.835	766.396	6.6	114.190
8	eu-2005	862.664	19.235.140	22.3	6.985
9	fidap037	3.565	67.591	19.0	85
10	FullChip	2.987.012	26.621.990	8.9	2.312.481
11	Ga41As41H72	268.096	9.378.286	35.0	472
12	in-2004	1.382.908	16.917.053	12.2	7.753
13	ins2	309.412	1.530.448	4.9	303.876
14	lhr34	35.152	764.014	21.7	63
15	mac_econ_fwd500	206.500	1.273.389	6.2	44
16	mc2depi	525.825	2.100.225	4.0	4
17	memplus	17.758	126.150	7.1	574
18	mip1	66.463	5.209.641	78.4	713
19	pdb1HYS	36.417	2.190.591	60.2	184
20	pwtk	217.918	5.926.171	27.2	180
21	rma10	46.835	2.374.001	50.7	145
22	s3dkt3m2	90.449	1.921.955	21.2	24
23	scircuit	170.998	958.936	5.6	353
24	shipsec1	140.874	3.977.139	28.2	84
25	Si41Ge41H72	185.639	7.598.452	40.9	531
26	Stanford	281.903	2.312.497	8.2	38.606
27	Stanford_Berkeley	683.446	7.583.376	11.1	83.448
28	webbase-1M	1.000.005	3.105.536	3.1	4.700
29	wikipedia-20051105	1.634.989	19.753.078	12.1	4.970
30	wikipedia-20060925	2.983.494	37.269.096	12.5	5.852



Şekil 7. AMD işlemcili bilgisayarda ölçülen başarım



Şekil 8. Intel işlemcili bilgisayarda ölçülen başarım



Şekil 9. Ön işlemenin bir CSR SpMV işlemi cinsinden maliyeti

Her iki bilgisayarda alınan sonuçlara göre görülmektedir ki Duff aygıtı tabanlı yöntemle belirgin hızlanmalar almak mümkündür. CSRLen yöntemiyle yapılan kıyaslamadan anlaşılıyor ki bu hızlanmalar sadece veri sıkıştırmasından kaynaklanmamaktır.

Son olarak, *DuffCompressed*$_k$ yöntemi için uygulanan ön işlemenin, yani CSR formatındaki `rows` dizisinin içeriğinin dönüştürülmesinin masrafını inceleyelim. Her bir matris için ön işlemede harcanan sürenin CSR yöntemiyle yapılan tek bir SpMV işleminin süresine göre normalleştirilmiş hali Şekil 9'da yer almaktadır. AMD işlemcili bilgisayarda ortalama maliyet bir SpMV koşumunun yalnızca 0,06'sı kadardır. En yüksek maliyet 0,16 SpMV koşumuna denktir. Intel üzerinde ise ortalama maliyet 0,10 iken, en yüksek 0,27 olarak ölçülmüştür. Buradan da görülmektedir ki sadece `rows` dizisini işleyen bir ön işlemenin maliyeti oldukça düşüktür.

4. İlgili Çalışmalar

SpMV işleminin çok sayıda bilim alanında kullanımı olduğu için hızlandırılması halinde yaygın etkisi geniş olmaktadır. Bu nedenle SpMV'nin hem CPU hem de GPU mimarilerinde eniyilenmesi araştırmacıların uzun süredir üzerinde çalışageldiği bir problemdir (örneğin [1,4–10,12,14,20,23,24]). Konuyla ilgili yakın zamanda yayınlanmış iki tarama çalışması bulunmaktadır [2,3]. Makalemizde SpMV'nin CPU'larda çalıştırılması üzerine eğildik; GPU makalemizin kapsamı dışındadır.

Çalışmamızda CSR tabanlı SpMV işleminin maliyeti düşük bir şekilde hızlandırılması üzerine odaklandık. Matris elemanlarının yeniden sıralanması ve CSR'den daha sofistike saklama formatı kullanılmasıyla SpMV işlemini belirgin şekilde hızlandırmak mümkün olabilir (örneğin [1,6,7,20,24,25]). Fakat bu tip yaklaşımlarda ön işleme ve format dönüşümünün maliyeti onlarca hatta yüzlerce SpMV işlemine denk olabilmektedir [6,7,13,24,26]. Alınan hızlanmanın bu maliyeti karşılaması için SpMV işleminin aynı matris için yüzlerce veya binlerce kez yapılması gerekebilir. Aynı matrisle tekrar tekrar SpMV işleminin yapıldığı bir bağlam olan yinelemeli çözücülerde yineleme sayısı kimi zamanlar iki basamaklı sayılar mertebesinde kalabilmektedir. Böyle bağlamlarda yüksek ön işleme maliyetli SpMV yöntemleri pratik olarak faydasız hale gelmektedir. Bu nedenle son zamanlarda CSR formatını hiç değiştirmeden veya düşük maliyetli bir ön işlemeden geçirerek SpMV'yi hızlandırma üzerine odaklanan çalışmalar olmuştur [12–17,27]. Bunlardan Ashari *vd.*'nin çalışmasında SpMV'nin GPU üzerinde koşumunun hızlandırılması için CSR formatındaki matrisin aynı uzunluktaki satırlarının bir araya getirilmesi önerilmektedir [13]. Greathouse ve Daga, yine GPU için, CSR formatındaki matris verisini iş parçacıklarına dinamik olarak dağıtan ve GPU'nun müsvedde belleğini verimli bir şekilde kullanan bir yaklaşım sunmaktadır [12]. Liu ve Schmidt de LightSpMV adını verdikleri GPU kütüphanelerinde CSR formatındaki matrisi hiçbir ön işlemeden geçirmemekte, hızlanmayı dinamik yük dağılımı yaparak ve GPU'nun atomik işlemleri ile *warp* karma komutlarını kullanarak sağlamaktadırlar [15]. Bir başka GPU odaklı yöntem olan LSRB-CSR formatında matris datası GPU'nun mimari özelliklerine ve *warp* yapısına uygun olarak dilimlere ayrılmaktadır [17]. Merrill ve Garland'ın makalesinde CSR formatı aynen korunmakta, eşzamanlı SpMV koşumu için matrisi mevcut yaklaşımlardan daha dengeli bölüntüleyen bir yaklaşım önerilmektedir [14]; önerdikleri teknik hem GPU hem CPU için uygundur. Ohshima *vd.* farklı OpenMP çizelgeleme ayarlarının CSR tabanlı SpMV başarımına etkisini incelemişlerdir [16]. Aktemur'un çalışmasında ise CSR yöntemindeki iç döngünün tamamen açılması yaklaşımı önerilmektedir [27]. Bu yöntem SpMV kodunun belli bir CPU mimarisi için çevirim dilinde yazılmasını zorunlu kılmıştır. Bizim sunduğumuz Duff tabanlı yaklaşımda gerçekleme kaynak kod seviyesindedir. Bu nedenle farklı mimarilerde çalıştırmak ve derleyici optimizasyonlarından faydalanmak mümkündür.

Duff aygıtı tabanlı SpMV gerçeklemesinin başarıma etkisi daha önce Youssefi'nin tezinde incelenmiştir [19]. Bu tez ile bizim makalemizin karşılaştırmasını Bölüm 2'de yapmıştık.

SpMV işlemi, matrisin her bir satırının birbirinden bağımsız olarak işlenebileceğinden yüksek seviyede eşzamanlı hale getirilebilen bir hesaplamadır. Eşzamanlılık için uygulanan genel yaklaşım matrisin mümkün olduğunca dengeli bir şekilde eldeki iş parçacığı sayısı kadar bölüntüye ayrılmasıdır. Bunun için tek boyutlu, satır tabanlı bölüntülemeler yanında iki boyutlu yaklaşımlar da bulunmaktadır [14,28–31]. Bölüntüleme sonrasında her bir matris parçası

tek iş parçacıklı bir SpMV kodu ile işlenir. Bu sebeple, her ne kadar tek iş parçacıklı bir SpMV gerçeklemesi sunmuş olsak da önerdiğimiz teknik mevcut bölüntüleme yaklaşımları ile birleştirilerek eşzamanlı hale getirilebilir.

Duff tabanlı SpMV yönteminde `rows` dizisi üzerinde veri küçültmesi yapmıştık. Böylece hafızadan işlemciye taşınan verinin boyutu azalmaktadır. SpMV başarımına veri küçültmenin ve sıkıştırmanın etkisinin incelendiği çalışmalar bulunmaktadır [23,32]. Bu çalışmalarda sunulan teknikler vasıtasıyla `vals` ve `cols` dizileri de sıkıştırılabilir, fakat ön işleme safhasına bu dizilerin de eklenmesi ön işlemenin maliyetini artıracağı için bu yaklaşımları kullanmadık.

5. Sonuç

Yaygın etkisi yüksek olan seyrek matris-vektör çarpımı (SpMV) işlemi için Duff aygıtı ismi verilen döngü açılımı tekniğine dayalı bir gerçekleme sunduk. Bu gerçekleme kaynak kod seviyesinde yapılabilir ve farklı mimarilere derlenebilir durumdadır. Deneysel değerlendirmemizde biri AMD diğeri Intel işlemcili iki bilgisayar üzerinde, gerçek mühendislik probleminden alınmış matrislerle başarım ölçümü yaptık; Duff tabanlı yöntem ile AMD'de 1,47, Intel'de 1,17 kata kadar hızlanma gözlemledik.

Önerdiğimiz yöntem maliyeti oldukça düşük olan bir ön işleme akabinde kullanılabilmektedir. Bu maliyet genel geçer CSR formatıyla yapılan bir SpMV işleminin ortalama 0,06'sına (AMD bilgisayarda) ve 0,10'una (Intel bilgisayarda) denktir. Düşük maliyetli ön işleme yaklaşımı bilhassa az sayıda yinelemenin yapılacağı ve dolayısıyla yüksek ön işleme maliyetlerinin amortize edilemediği bağlamlar için uygun olmaktadır.

Sunduğumuz çalışmanın kaynak kodunu aşağıdaki web adresinde kamuya açık olarak sunuyoruz:

https://github.com/ozusrl/thundercat

Kaynakça

[1] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J. Optimization of Sparse Matrix-vector Multiplication on Emerging Multicore Platforms. Parallel Comput 2009;35:178–94. doi:10.1016/j.parco.2008.12.006.

[2] Filippone S, Cardellini V, Barbieri D, Fanfarillo A. Sparse Matrix-Vector Multiplication on GPGPUs. ACM Trans Math Softw 2017;43:30:1--30:49. doi:10.1145/3017994.

[3] Langr D, Tvrdik P. Evaluation Criteria for Sparse Matrix Storage Formats. IEEE Trans Parallel Distrib Syst 2016;27:428–40. doi:10.1109/TPDS.2015.2401575.

[4] Goumas GI, Kourtis K, Anastopoulos N, Karakasis V, Koziris N. Understanding the Performance of Sparse Matrix-Vector Multiplication. 16th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process., 2008, p. 283–92. doi:10.1109/PDP.2008.41.

[5] Liu X, Smelyanskiy M, Chow E, Dubey P. Efficient Sparse Matrix-vector Multiplication on x86-based Many-core Processors. Proc. 27th Int. ACM Conf. Int. Conf. Supercomput., New York, NY, USA: ACM; 2013, p. 273–82. doi:10.1145/2464996.2465013.

[6] Belgin M, Back G, Ribbens CJ. A Library for Pattern-based Sparse Matrix Vector Multiply. Int J Parallel Program 2011;39:62–87. doi:10.1007/s10766-010-0145-2.

[7] Liu W, Vinter B. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. Proc. 29th ACM Int. Conf. Supercomput., New York, NY, USA: ACM; 2015, p. 339–50. doi:10.1145/2751205.2751209.

[8] Vuduc R, Demmel JW, Yelick KA. OSKI: A library of automatically tuned sparse matrix kernels. J Phys Conf Ser 2005;16:521. doi:10.1088/1742-6596/16/1/071.

[9] Bell N, Garland M. Implementing Sparse Matrix-vector Multiplication on Throughput-oriented Processors. Proc. Conf. High Perform. Comput. Networking, Storage Anal., New York, NY, USA: ACM; 2009, p. 18:1--18:11. doi:10.1145/1654059.1654078.

[10] Yzelman AN, Roose D. High-Level Strategies for Parallel Shared-Memory Sparse Matrix-Vector Multiplication. IEEE Trans Parallel Distrib Syst 2014;25:116–25. doi:10.1109/TPDS.2013.31.

[11] Saad Y. Iterative Methods for Sparse Linear Systems. SIAM; 2003. doi:10.1137/1.9780898718003.

[12] Greathouse JL, Daga M. Efficient Sparse Matrix-vector Multiplication on GPUs Using the CSR Storage Format. Int. Conf. High Perform. Comput. Networking, Storage Anal. (SC '14), 2014, p. 769–80. doi:10.1109/SC.2014.68.

[13] Ashari A, Sedaghati N, Eisenlohr J, Parthasarathy S, Sadayappan P. Fast Sparse Matrix-vector Multiplication on GPUs for Graph Applications. Int. Conf. High Perform. Comput. Networking, Storage Anal. (SC '14), 2014, p. 781–92. doi:10.1109/SC.2014.69.

[14] Merrill D, Garland M. Merge-based Parallel Sparse Matrix-vector Multiplication. Int. Conf. High Perform. Comput. Networking, Storage Anal. (SC '16), 2016, p. 58:1--58:12. doi:10.1109/SC.2016.57.

[15] Liu Y, Schmidt B. LightSpMV: Faster CSR-based sparse matrix-vector multiplication on CUDA-enabled GPUs. Proc. Int. Conf. Appl. Syst. Archit. Process., vol. 2015–Septe, 2015, p. 82–9. doi:10.1109/ASAP.2015.7245713.

[16] Ohshima S, Katagiri T, Matsumoto M. Performance Optimization of SpMV Using CRS Format by Considering OpenMP Scheduling on CPUs and MIC. IEEE Int. Symp. Embed. Multicore/Manycore SoCs, 2014, p. 253–60. doi:10.1109/MCSoC.2014.43.

[17] Liu L, Liu M, Wang C, Wang J. LSRB-CSR: A low overhead storage format for SpMV on the GPU systems. Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS, vol. 2016–Janua, 2016, p. 733–41. doi:10.1109/ICPADS.2015.97.

[18] Duff T. Duff's Device 1988. http://www.lysator.liu.se/c/duffs-device.html.

[19] Youssefi A. Exploring the Potential for Accelerating Sparse Matrix-Vector Product on a Processing-in-Memory Architecture. Rice University, 2008.

[20] Mellor-Crummey J, Garvin J. Optimizing Sparse Matrix-Vector Product Computations Using Unroll and Jam. Int J High Perform Comput Appl 2004;18:225–36. doi:10.1177/1094342004038951.

[21] Koster J. Parallel Templates for Numerical Linear Algebra, A High-Performance Computation Library. Utrecht University, 2002.

[22] Davis TA, Hu Y. The University of Florida Sparse Matrix Collection. ACM Trans Math Softw 2011;38:1:1--1:25. doi:10.1145/2049662.2049663.

[23] Kourtis K, Goumas G, Koziris N. Exploiting Compression Opportunities to Improve SpMxV Performance on Shared Memory Systems. ACM Trans Arch Code Optim 2010;7:16:1--16:31. doi:10.1145/1880037.1880041.

[24] Karakasis V, Gkountouvas T, Kourtis K, Goumas G, Koziris N. An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication. IEEE Trans Parallel Distrib Syst 2013;24:1930–40. doi:10.1109/TPDS.2012.290.

[25] Kamin S, Garzarán MJ, Aktemur B, Xu D, Yılmaz B, Chen Z. Optimization by Runtime Specialization for Sparse Matrix-vector Multiplication. Gener. Program. Concepts Exp. (GPCE '14), 2014, p. 93–102. doi:10.1145/2658761.2658773.

[26] Yılmaz B, Aktemur B, Garzarán MJ, Kamin S, Kıraç F. Autotuning Runtime Specialization for Sparse Matrix-Vector Multiplication. ACM Trans Arch Code Optim 2016;13:5:1--5:26. doi:10.1145/2851500.

[27] Aktemur B. A sparse matrix-vector multiplication method with low preprocessing cost. Concurr Comput Pract Exp 2018;30:e4701. doi:10.1002/cpe.4701.

[28] Yang W, Li K, Mo Z, Li K. Performance Optimization Using Partitioned SpMV on GPUs and Multicore CPUs. IEEE Trans Comput 2015;64:2623–36. doi:10.1109/TC.2014.2366731.

[29] Yzelman AN, Bisseling RH. Cache-Oblivious Sparse Matrix--Vector Multiplication by Using Sparse Matrix Partitioning Methods. SIAM J Sci Comput 2009;31:3128–54. doi:10.1137/080733243.

[30] Martone M. Efficient multithreaded untransposed, transposed or symmetric sparse matrix--vector multiplication with the Recursive Sparse Blocks format. Parallel Comput 2014;40:251–70. doi:10.1016/j.parco.2014.03.008.

[31] Çatalyürek Ü V., Aykanat C. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans Parallel Distrib Syst 1999;10:673–93. doi:10.1109/71.780863.

[32] Willcock J, Lumsdaine A. Accelerating Sparse Matrix Computations via Data Compression. Proc. 20th Annu. Int. Conf. Supercomput., New York, NY, USA: ACM; 2006, p. 307–16. doi:10.1145/1183401.1183444.