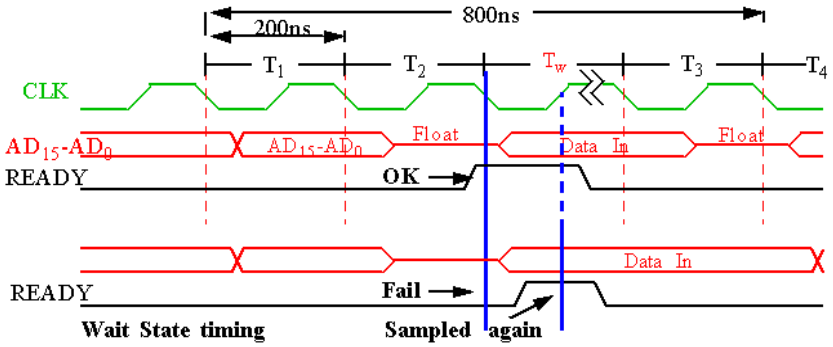
KTÜ Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü
Bilgisayar Organizasyonu Laboratuvarı

BEKLEMELİ ÇALIŞMA ve ZAMAN SINIRLI İŞLER

1. Beklemeli Çalışma

Yavaş yanıt verebilen bellek veya dış cihazlara işlemcinin erişmesi sırasında yaşadığı gecikmeye Bekleme Durumu (Wait State) denir. İşlemciler genelde verileri geçici olarak saklama için kullanılan ana bellek (DRAM) ve kalıcı olarak saklama için kullanılan katı hal disk (SSD), melez disk (SSHD), manyetik disk (HDD) ve optik disklerden (CD, DVD, BD) çok daha hızlı çalışır. Birkaç GHz hızında çalışan işlemciler bile, ~14 ns erişim süresine sahip 8-derinlikli ön-erişim tamponlu (8-deep prefetch buffer) DDR3 veya DDR4 belleğe erişirken onlarca saat dönemi (periyodu) süresince beklemek zorunda kalmaktadır. İşlemci saat dönemi ve bellek erişim süreleri arasındaki bu büyük farklılık nedeniyle işlemcinin bellekten gecikmesiz (zero wait state) okuma/yazma yapamayacağı açıktır. Bu nedenle işlemcinin okuma/yazma süresini uzatmak için READY girişine T2 evresi sonunda uygulanan düşük seviyeli bir işaret ile okuma/yazma çevrim süresinin uzatılması sonucunda yavaş bellek ile hızlı bir işlemcinin çalıştırılması mümkündür.



READY girişinden uygulanan işaret ile okuma/yazma çevrim süresinin uzatılması işlemcinin birim zamanda çalıştıracağı emir sayısını ve başarımını azaltır. Yüksek başarım beklentisi ile çelişen bu basit çözüm yerine, Intel 8086'da, mikroişlemcinin bellekten emirleri okumasını beklemeden program sayacının (Program Counter, PC) bellekte gösterdiği adresten başlayarak bellekten işlemci içindeki 6 byte'lık bir ilk-giren-ilk-çıkar (First In First Out, FIFO) kuyruğa emir okuyan bir birim kullanılmıştır. Intel tarafından Ortak-yol Arayüz Birimi (Bus Interface Unit, BIU) adı verilen bu birim, çalıştırılacak 6 emiri Çalıştırma Birimine (Execution Unit, EU) hazır sunduğu için işlemci yavaş belleği beklemeksizin daha hızlı çalışarak READY işareti ile okuma/yazma süresi uzatılmasına kıyasla önemli bir başarım artışı sağlamaktadır. BIU'nun, işlemci tarafından okunması beklenen emirleri bellekten erken okuyarak EU için hazır hale getirmesinin tek sakıncası, dallanma (jmp veya br emirleri) ve alt program çağırma (call emri) gibi PC içeriğinin değiştiği durumlarda daha önce getirilmiş olan verinin anlamını yitirmesi nedeniyle FIFO'nun boşaltılarak yeni PC değerinden başlayarak yeniden doldurulmasıdır. Ön bellek (cache) türünün ilk örneği olan BIU ve 6 byte'lık FIFO yerine günümüz işlemcilerinde, yüksek kapasiteli ana belleğe kıyasla yüksek erişim hızı için sınırlı kapasiteli statik bellek (SRAM) hücreleri kullanan, çok seviyeli (L0, L1, L2, L3 ..), emir ve veriyi ayrıştırabilen paylaşımlı veya paylaşımsız ön bellekler ve ön bellek denetleyicileri kullanılmaktadır. Ön bellek kullanımı ve ardışıl adreslerdeki bilgilerin daha hızlı okunmasına olanak tanıyan 2 (DDR), 4 (DDR2), 8 (DDR3) ve 8 (DDR4) derinlikte ön-erişim tamponu kullanan belleklerin kullanımı sonucunda, bellek kod çözücülerinin neden olduğu

gecikmenin başarımı sınırlandırması ardışık adreslere erişim için azaltılmaktadır. Programlarda verinin zaman ve mekânda yerel (yakın) olması varsayımı ile tasarlanan bu başarım artırıcı tekniklerin kullanımı birçok uygulamada başarımı çok yükseltmesine karşılık, ön bellek kullanımı ve 8 derinlikte ön-erişim tamponu ile gerçekleştirilen hızlı ardışık okumanın uygulama başarımını çok az artırabileceği uygulama türleri de vardır. Veri ve emirler için, bellekte yerel olmayan yani öngörülemez adreslere erişen uygulamalar, önbellek ve ön-erişim tamponundan yeterince yararlanamamaktadır. Güncel işlemcilerde L0, L1, L2, L3 ön bellekler, ön bellek denetleyici ve bellek denetleyici doğrudan işlemci bünyesinde bulunduğu için READY işaretini üreterek işlemciyi beklemeye alma sorumluluğunu bu denetleyiciler üstlenmektedir. İşlemciden daha yavaş bir bellek için bekleme işaretinin üretilmesi ve işlemcinin okuma/yazma çevrimini uzatarak işlemci bellek uyumunun sağlanması, ön bellek ve bellek denetleyicisi işlemciyle bütünleşik olmayan işlemciler dışında kullanılamamaktadır.

Intel, How to Use Loop Blocking to Optimize Memory Use on 32-Bit Intel® Architecture adlı dokümanda, ön bellek kapasite sınırlamasına dikkat edilerek kodun nasıl daha verimli (kısa sürede) çalıştırılabileceği konusunda bilgi vermektedir. Aşağıdaki örnek kod parçasında, boyutları N x N olan bir A matrisinin uygun boydaki B matrisi devriği (transpozu) ile nasıl toplanabileceği gösterilmektedir. Örnek kod parçasından da anlaşılacağı gibi, sonucu değiştirmese bile yapılacak bir işlemin ön bellek kapasite sınırlamasına dikkat edilerek yazılması çalışma hızını artırabilmektedir.

Ön bellek kapasite sınırlaması dikkate alınmadan:

```
float A[N, N], B[N, N];
for (i=0; i< N; i++)
     for (j=0; j< N; j++)
          A[i,j] = A[i,j] + B[j, i];
```

Ön bellek kapasite sınırlaması dikkate alınarak:

```
float A[N, N], B[N, N];
for (i=0; i< N; i+=blok_boyu)
     for (j=0; j< N; j+=blok_boyu)
         for (ii=i; ii<i+blok_boyu; ii++)
             for (jj=j; jj<j+blok_boyu; jj++)
                 A[ii,jj] = A[ii,jj] + B[jj, ii];
```

2. Zaman Sınırlı İşler

Yüksek hız için tasarlanan işlemciler birçok uygulamada hedeflenen başarımı göstermesine karşılık bazı uygulamalarda işlemleri zamanında tamamlamakta yetersiz kalabilmektedir. Belirli bir zaman aralığında yapılması gereken işlere örnek olarak, akıcı algılanması istenen bir oyunda saniyede en az 25 görüntü üretme, bir video oynatıcıda yüksek oranda sıkıştırma için kodlanmış görüntü karelerini bir sonraki karenin gösterilmesinden önce (~1/25 s) kodunu çözme, iki bilgisayar arasında gerçek zamanlı (düşük gecikmeli) ses iletimi yapılabilmesi için bir sonraki ses örneğinin kullanılmasından önce (~1/8000 s) hesaplama gibi örnekler verilebilir. Sınırlı sürede yapılması beklenen işlemlerin zamanında bitirilememesi durumunda kötü sonuçlar doğmuyorsa bu tür gerçek zaman gereksinimine **yumuşak gerçek zamanlı** (soft real-time) uygulamalar denmektedir. Yumuşak gerçek zamanlı uygulamalara örnek olarak, 3B (3D) bilgisayar oyunlarında cisimleri oluşturan çok sayıdaki poligonların tüm pikselleri için görünmeyen yüzeyleri yok etme amacı ile derinlik tamponu (depth buffer, Z-Buffer) algoritmasının 1/25 s içinde uygulanmasının mümkün olmaması veya karmaşık bir videodaki bir görüntü karesinin kodlanmış halinin 1/25 s süresi içinde kod çözülememesi verilebilir. İşlemci başarım sınırlaması sonucunda ortaya çıkabilecek durumlarda hesaplanamayan bir örneğin atlanması veya

önceki örneğin zamanında hesaplanamayan örnek yerine tekrar kullanılması sıklıkla kullanılan ve kullanıcıları çok rahatsız etmeyen bir çözümdür.

Endüstriyel bir otomasyon sisteminin gerek duyduğu denetim bilgilerinin zamanında hesaplanamaması, bir uçağın uçuş denetim sisteminin gerek duyduğu denetim işaretlerinin zamanında üretilememesi veya sürücüsüz bir aracın önünde algıladığı bir yayaya çarpmamak için gerek duyduğu acil denetim bilgisinin zamanından geç üretilmesinin çok kötü sonuçları olabileceği için bu uygulamalara katı gerçek zamanlı (hard real-time) uygulamalar denmektedir. Yumuşak gerçek zaman uygulamalarında kullanılan ve zamanında hesaplanamayan örnek yerine bir önceki örneği kullanmak katı gerçek zamanlı uygulamalarda çözüm olarak kabul edilemeyeceği için, gereken hesapları genel amaçlı bir işlemciye göre çok daha yüksek verimle gerçekleştirebilecek, Uygulamaya Özgü Entegre Devreler (Application Specific Integrated Circuit, ASIC) tasarlanmaktadır. Örneğin, Taylor serisine açılımı ile hesaplanan açısal, üstel ve daha karmaşık işlevleri gerçekleştiren Kayan noktalı Aritmetik İşlemciler (Floating Point Unit, FPU) genel amaçlı işlemcilere kıyasla çok daha kısa sürede bu işlevleri hesaplayabilmektedir. Benzer şekilde algoritması standartlar tarafından belirlenen mpeg2, mpeg4, h264 ve h265 gibi video kodlama algoritmalarını gerçekleştiren adanmış bir yardımcı işlemci (co-processor) birimi güncel işlemcilerin çoğunda bulunmaktadır. Üç boyutlu bilgisayar oyunlarında cisimlerin görünmeyen yüzeylerini yok etmek için kullanılan derinlik tamponu algoritması, işlemci içinde veya işlemci dışında Grafik İşlem Birimi (Graphics Processing Unit, GPU) tarafından çalıştırılmakta ve gerçekçi görüntüler gerçek zamanda üretilebilmektedir.

Ana bilgisayar (Mainframe) sistemlerinin büyük işletmeler ve kampüslerde kullanıldığı yıllarda bilim ve mühendislik problemlerinde sıklıkla karşılaşılan vektör ve matris işlemlerini daha hızlı yapabilmek için vektör işlem (Single Instruction Multiple Data, SIMD) birimleri kullanımı oldukça yaygındı. Üniversitemizde de 90'lı yıllarda satın alınarak 2005'e kadar kullanılmış olan DEC firmasının VAX 6000 Model 510 ana bilgisayar sisteminde de vektör birim vektör veri ile çalışan akademik personel tarafından kullanılmıştır. Ana bilgisayar sistemlerinden sunucu (server), iş istasyonu (workstation) ve kişisel bilgisayar çağına geçildikten sonra, çoklu ortam (Multimedia) uygulamaları tarafında yoğun olarak kullanılan vektör işlemleri için 1997 yılından sonra üretilen x86 mimarili işlemcilerin tümünde vektör işlem yapabilen SIMD birimleri ve emir kümeleri (2 adet 32-bit, 4 adet 16-bit veya 8 adet 8-bit tamsayı işlemi aynı anda yapabilen Intel Multi Media eXtensions MMX, 32-bit kayan noktalı aritmetiği de MMX'e ekleyen AMD 3DNow!, 128-bit kaydedici ve 32-bit kayan noktalı işlem yapabilen Intel Streaming SIMD Extensions SSE, 64-bit kayan noktalı işlem yapabilen SSE2, SSE2'ye 13 yeni emir ekleyen SSE3 ve emir sayısını 54'e çıkaran SSE4, SandyBridge/IvyBridge/Haswell serisi işlemcilerde 256-bit kaydedici kullanabilen Advanced Vector eXtensions AVX ve AVX2 ve Knight'sCorner serisi Xeon Phi işlemcisinde kullanılan 512-bit kaydedicili AVX-512) bulunmaktadır. Vektör işlem birimi x86 mimarisi dışında da AIM olarak bilinen Apple, IBM ve Motorola (Freescale) firmalarının kurduğu birlik tarafından üretilen PowerPC mimarisinde AltiVec adı ile 32-bit tamsayı ve kayan noktalı sayı işlem yapabilen VMX128, VSX, VMX 2 ve VMX 3 emir kümeleri olarak bulunmaktadır. Vektör işlemler, çoğunlukla vektör ve matris formunda ifade edilen Sayısal İşaret İşleme (Digital Signal Processing, DSP) algoritmalarını, aynı anda işleyebildiği eleman sayısı oranında paralelleştirebildiği için kayda değer hızlanma sağlayabilmektedir.

Bilim ve mühendislik problemlerinde sık karşılaşılan ve gerçek zamanda yapılması istenen vektör çarpımı, işlemcinin vektörün tüm elemanlarını sıra ile tek tek işlemesi yerine vektör işlem emirleri ile vektör boyu oranında hızlandırabilmektedir. Ses işaretinin gerçek zamanda işlenmesi için vektör işlem emirlerini kullanılması zaman sınırlı işlere basit örnektir. Sesin işlenme gerekçelerinden birisi de sesteki bazı frekans bileşenlerinin genliklerinin azaltılması veya artırılması yani süzgeçten geçirilmesidir. Örneğin, bilgisayarın ses kartı ile örneklenen ses, hard disk dönüş sesi veya soğutucu

pervane sesi gibi istenmeyen bileşenleri de içerdiği için bu bileşenlerin geçemeyeceği bir süzgeçten geçirilerek disk veya pervane gürültüsü azaltılabilir. Süzgeçler, analog yani devre elemanları kullanılarak yapılabildiği gibi sayısal olarak ta gerçekleştirilebilir. Sayısal süzgeçler birçok açıdan eşdeğer analog süzgeçlere göre üstün özellikler göstermektedir. Sayısal bir süzgeçin çıkışı, süzgeç frekans tepkesinin giriş işareti frekans gösterimi ile çarpımının ters Fourier'i olarak yazılabilir. Frekans domenindeki yapılan çarpma işleminin, zaman domenindeki katlama işlemine eşdeğer olduğu da bilinmektedir. Katlama, süzgeç zaman tepkesini oluşturan katsayı vektörünün, örneklenmiş giriş işareti vektörü ile çarpım sonuçlarının toplanması ile hesaplanmaktadır.

3. Deneyin Yapılışı

Deneyde geliştirilecek programın Assembly dilinde yazılması mümkün olsa bile, geliştirme kolaylığı ve farklı derleyiciler (gcc, clang, icc ve MSVC) arasında deneyde kullanılacak kodun taşınabilirliği için, C/C++ dili ve vektör işlem emirlerine ortak arayüz sağlayan başlık dosyasında tanımlı olan isimlerle (intrinsics) vektör emirler kullanılarak işlemler gerçekleştirilecektir. Deneyde, masaüstü ve dizüstü bilgisayarlarda yaygın olarak kullanılan güncel işlemcilerde bulunan AVX/AVX2/FMA gelişmiş vektör komut setlerini çalıştırabilecek şekilde, 256-bit uzunluktaki kaydediciler ile AVX vektör komutlarını, Intel Sandy Bridge veya Ivy Bridge mimarili Celeron ve Pentium dışındaki i3/i5/i7 veya AMD Bulldozer, Piledriver ve Steamroller veya daha yeni işlemcilerde ve AVX2/FMA vektör komutlarını Intel Haswell mimarili Celeron ve Pentium dışındaki i3/i5/i7 veya AMD Excavator veya daha yeni işlemcilerin kullanılması gerekmektedir. Deney yapacağınız bilgisayar işlemcisinin desteklediği vektör komut kümesini Windows'da **CPU-Z** yazılımı veya Linux'de **cat /proc/cpuinfo** komutu ile öğrenebilirsiniz.

Deneyde, AVX/AVX2 ve FMA gelişmiş vektör işlem komut seti ile vektör hesaplamaları basit örnekler ile açıklayan [4] kaynağında verilen örnekler üzerinden gidilerek karmaşık sayıların çarpımı vektör işlemler ile yapılacak ve skaler (ALU+FPU) ile gerçekleştirilmesine kıyasla başarım artışı ölçülecektir. İlgili kaynak, AVX/AVX2 ve FMA komutlarını ve bu komutlara C/C++ programlardan erişmek için kullanılacak **immintrin.h** başlık dosyasında tanımlanmış olan **intrinsic** işlev adlarını ve argümanlarını listelemektedir. Deney sırasında kaynakta verilen C++ kodları yazmak için Ubuntu ile gelen grafiksel kullanıcı arayüzlü **gedit** veya komut satırından kullanılabilen **pico** gibi düzenleyiciler kullanabilirsiniz. Koda web tarayıcıdan bakarak klavyeden eksiksiz/hatasız yazmaya çalışmak yerine kaynaktan kopyala/yapıştır ile alabilirsiniz. Yazdığınız kodu kaynakta belirtilen komut satırı parametreleri ile **gcc** derleyicisi kullanarak derleyiniz ve çalışma süresini not ediniz.

Denediğiniz kodun çalışma süresini ölçmek için POSIX uyumu için tanımlı olan **gettimeofday** işlevini aşağıda verilen kod örneğindeki gibi kullanabilirsiniz.

```
#include <time.h>
#include <sys/time.h>
int main(int argc, char* argv[])
{
  struct timeval t_before, t_after, t_diff;
  ...
  gettimeofday(&t_before, NULL);
/*  çalışma süresi ölçülmek istenen kod satırları */
  gettimeofday(&t_after, NULL);
  ...
  timersub(&t_after, &t_before, &t_diff);
  printf("%ld.%06ld s\n",(long int)t_diff.tv_sec, (long int)t_diff.tv_usec);
```

a. Kaynakta 3.1 de açıklanan vektör veri tipleri, 3.2 deki intrinsic işlev adları kabulleri ve 3.3 deki AVX programların derlenmesi aşamalarını okuduktan sonra örnek kodları derleyerek çalıştırınız.

b. Kaynakta 4.1 de açıklanan vektör kaydedicilere ilk değerlerin atanması ve 4.2 de açıklanan programda tanımlanan ve bellekteki değişkenlerden vektör kaydedicilere yükleme/saklama örneklerini okuduktan sonra örnek kodları derleyerek çalıştırınız.

c. Kaynakta 5.1 de açıklanan aritmetik işlemlerden toplama/çıkarma, 5.2 de açıklanan çarpma/bölme ve 5.3 de açıklanan Birleştirilmiş Çarpma ve Toplama (Fused Multiply and Add, FMA) örneklerini okuduktan sonra örnek kodları derleyerek çalıştırınız.

d. Kaynakta 6.1 de açıklanan permute etme ve 6.2 de açıklanan karıştırma (shuffle) örneklerini okuduktan sonra örnek kodları derleyerek çalıştırınız. Bu aşamada derleyiciye daha önce kullandığınız -mavx veya -mavx2 parametresi yerine -mfma ile FMA için derlemesini belirtmeniz gerekmektedir.

e. Kaynakta 7. alt başlıkta açıklanan ve bilim ve mühendislikte sık kullanılan karmaşık sayı çarpımı için önerilen algoritmayı inceledikten sonra örnek kodları derleyerek çalıştırınız. Çarpım için vektör kaydedici ve gelişmiş vektör işlem komutlarının sağladığı hızlanmayı belirlemek için aynı çarpmayı gelişmiş vektör komutları kullanmadan yapınız.

Not : Üreticiden bağımsız olarak x86-64 mimarili işlemcilerin desteklediği tüm vektör komut kümeleri için **x86intrin.h** başlık dosyası, gcc (v. 4.5.0+), clang (v 3.x+), icc (v. 16.0+) gibi tüm modern derleyiciler tarafından ortak olarak kullanılmaktadır. MicroSoft ise Visual Studio 2013'de MSVC 12.0 sürümü derleyicisine **intrin.h** başlık dosyasını dahil etmiştir.

Kaynaklar :

0. Web: UNM, The University of New Mexico, ECE 310 ders notları
1. Web: wikipedia.org SIMD
2. Web: UAF, University of Alaska Fairbanks, CS 441 ders notları
3. Web: stackoverflow.com, Header files for x86 SIMD intrinsics
4. Web: codeproject.com, Crunching Numbers with AVX and AVX2