Karmaşık Algoritmaların Gerçek Zamanlı Gömülü Sistemlerde Gerçeklenmesi

Sevda ERDOĞDU

ASELSAN AŞ., Savunma Sistem Teknolojileri (SST) Grubu, Ankara

e-posta: erdogdu@aselsan.com.tr

Özetçe

Gerçek zamanlı gömülü sistemlerde basitten karmaşığa farklı seviyelerde algoritmaya ihtiyaç duyulabilmektedir. Bu algoritmaların tasarlanması, üzerinde koşacakları en uygun donanım yapısının oluşturulması ve bu donanım üzerinde gerçeklenmesi özen isteyen bir süreçtir. Bu çalışmada ASELSAN Savunma Sistem Teknolojileri (SST) Grubu algoritma geliştirme çalışmaları özetlenecek, gerçek zamanlı gömülü sistemler için görüntü işleme algoritmalarının gerçeklenmesi esnasında elde edilen tecrübeler paylaşılacaktır.

1. Giriş

Algoritma bir problemi çözmek için açık ve net olarak tanımlanmış bir yöntem olarak ifade edilmektedir. Gömülü Sistemler ise genel amaçlı işlerden çok kendisi için önceden özel olarak tanımlanmış görevleri yerine getiren, sınırlı kaynakları olan sistemlerdir. Gömülü sistemlere günlük hayattan, tıp, telekomünikasyon, hava, uzay, savunma sanayine kadar birçok alanda rastlamak mümkündür. Tahmin edilebileceği üzere gömülü sistemlerde pek çok problemi çözmek için çeşitli algoritmalara ihtiyaç duyulabilmektedir. Bu algoritmalar ihtiyaca göre kontrol algoritmalarından, sinyal işleme algoritmalarına (radar, sonar, görüntü), şifreleme algoritmalarından, kaynaşım (fusion) algoritmalarına kadar değişiklik gösterirler.

Gerçek zamanlı gömülü sistemlerin en belirgin özellikleri sıkı gerçek zamanlı olmalarıdır, bu sistemlerin sorumlu oldukları işleri gerçekleştirip, uygun çıktıları/ tepkileri üretmeleri için belli bir süreleri vardır. Çıktıların üretilmesinin bu süreyi aşması durumunda çıktılar kullanışsız hale gelir, çok kritik hatalar oluşabilir. Bundan dolayı bu sistemler için geçerli olan sıkı gerçek zamanlılık gereksinimi, bu sistemlerde kullanılacak olan algoritmalar için de geçerlidir.

Bu çalışmada, ASELSAN SST Grubu'nda gerçek zamanlı gömülü sistemler için algoritma geliştirme ile başlayan, algoritmanın üzerinde koşacağı donanımı belirleme çalışmalarından, algoritma gerçekleme çalışmalarına ve sistem testlerine kadar edinilen tecrübeler anlatılacaktır.

2. Algoritma Geliştirme Çalışmaları

ASELSAN SST Grubu'nda algoritma geliştirme çalışmalarında genel olarak izlenen yöntem şöyledir:

İlk olarak sistem gereksinimlerini karşılamak için ihtiyaç duyulan algoritmaların kapsamı belirlenir. Bu kapsamda problem tanımı, literatür araştırması, algoritmanın başarım kriterleri belirlenir. Algoritma kapsamı belirlendikten sonra tasarım çalışmalarına geçilir. Tasarım aşamasında algoritma akışı belirlenir, algoritma benzetim kodları oluşturulur. Tasarlanan algoritmalar, farklı kullanım senaryolarını içeren test verileri ile test edilir.

Testlerden sonra algoritmadan uygun bir performans alındı ise algoritmanın yazılım/donanım yapılandırma (configuration) birimleri içinde gerçeklenmesi yapılır. Gerçekleme esnasında ortaya çıkan problemler algoritmayı tasarlayan ekibe iletilmeli ve birlikte gerekli değişiklikler yapılmalıdır.

Son olarak algoritmaların gerçeklendikleri donanım ile birlikte gerçek ortamında test edilmesi vardır. Bu testler esnasında da problemler / eksiklikler gözlemlenebilir ve bunların giderilmesi konusunda da yine algoritma ekibi ile birlikte çalışılır.

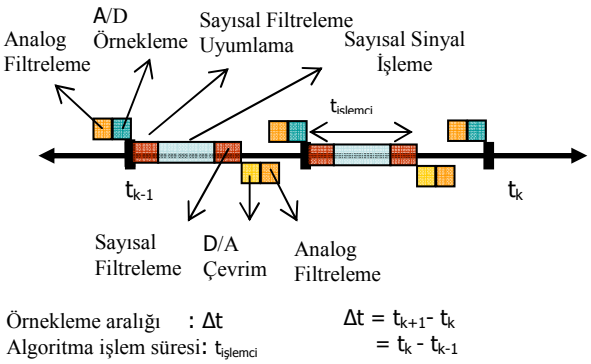
Sinyal işleme algoritmalarının gerçeklenmesinde de benzer bir yöntem önerilmiştir.[1]

3. Algoritma Gerçekleme Çalışmaları

ASELSAN SST Grubu'nda bir sistem için ihtiyaç duyulan algoritmaların tasarımı aşamasında çoğunlukla üniversitelerle birlikte çalışılmaktadır. Uygun olan algoritmaların belirlenmesinden sonra bu algoritmaların benzetim kodları geliştirilir. Gerekli testler yapılır ve algoritmalar belli bir olgunluğa getirilir. Benzetim kodlarının geliştirilmesi için Matlab/Scilab gibi matematiksel araçlar kullanılabileceği gibi, algoritmalar Visual C++, .Net gibi araçlar kullanılarak C, C++ gibi üst seviye programlama dillerinde de geliştirilebilir.

Geliştirilen algoritmalar uygun veriler ile test edilmelidir. Test verileri iki gruba ayrılır. İlk grupta algoritmanın geliştirilmesi esnasında kullanılan veriler, ikinci grupta ise kontrol verileri adı verilen, geliştirilen algoritmaların doğruluğunu test etmek için kullanılan veriler vardır. Test verilerinin olası tüm senaryoları içermesi önemlidir.

Yukarıda sözü geçen ortamlarda geliştirilen benzetim kodlarının, gerçek zamanlı gömülü sistemler üzerinde sınırlı sürede koşacak şekilde düzenlenmesi gerekir. Daha önce de belirtildiği gibi bu sürede oluşturulamayan algoritma çıktıları kullanılamazlar.

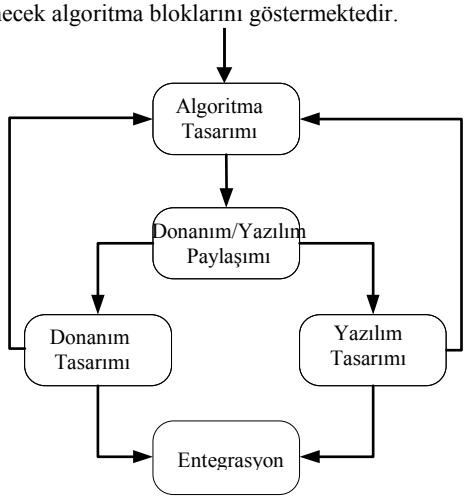


Şekil 1: Gerçek Zamanlı Sinyal İşleme.

Şekil 1'de gerçek zamanlı bir sinyal işleme sistemi için örnek zamanlama bilgisi gösterilmiştir.
Algoritma tasarım çalışmaları esnasında, paralel yürütülen çalışmalar ile algoritmanın üzerinde koşacağı donanım da kabaca belirlenir. Eğer donanım özel olarak tasarlanacaksa, yazılım çalışmalarına başlamak için bu donanımı beklemek büyük bir vakit kaybına neden olacaktır. Bunu engellemek için tasarlanacak donanımda kullanılacak işlemciye benzer ya da aynı işlemciyi içeren değerlendirme kartları (demo / evaluation boards) kullanarak yazılım çalışmalarına başlamak, olgunlaşmış algoritma bloklarını bu donanım üzerinde gerçeklemek, kabaca zaman bilgilerini almak ve dar boğazları belirlemek açısından faydalı olacaktır. Ayrıca yazılımın algoritma dışındaki genel yapısının oluşturulması için de değerlendirme kartlarını kullanmak yazılım geliştirme çalışmalarını hızlandıracaktır.

3.1. Algoritma Paylaşımı

Algoritmaların gerçeklenmesi beklendiği üzere, üzerinde koşacağı donanıma bağlıdır. Özellikle yüksek işlem gücü, yüksek hafıza bandı gerektiren algoritmaların gerçeklenmesinde sayısal sinyal işlemciler veya genel amaçlı işlemcilerle beraber FPGA (Programlanabilir Tümleşik Devre-Field Programable Gate Array) ve/veya ASIC (Application Spesific Integrated Circuit) gibi yapılardan oluşan donanımlar sıklıkla kullanılmaya başlanmıştır. FPGA'ler işlemlerin paralellenebilmesi ve derin "pipeline"lar oluşturulabilmesi yetenekleri ve dağıtık hafıza kaynakları ile bu tip algoritmalar için uygun gerçekleme ortamları sağlamaktadırlar. Bu tip donanımlar üzerinde gerçeklenecek algoritmaların donanım / yazılım paylaşımı, karmaşık algoritmaların en doğru ve kayıpsız bir şekilde gerçeklenmesi için önem taşımaktadır.
Şekil 2'de donanım yazılım paylaşım süreci gösterilmiştir. Burada yazılım olarak adlandırılan kısım PowerPC ya da DSP (Digital Signal Processor) gibi bir işlemci üzerinde C / C++ gibi bir dille gerçeklenecek algoritma bloklarını gösterirken, donanım ile adlandırılan kısım FPGA, ASIC, CPLD (Complex Programmable Logic Device) gibi yapılar üzerinde gerçeklenecek algoritma bloklarını göstermektedir.



Şekil 2: Algoritma paylaşım süreci.

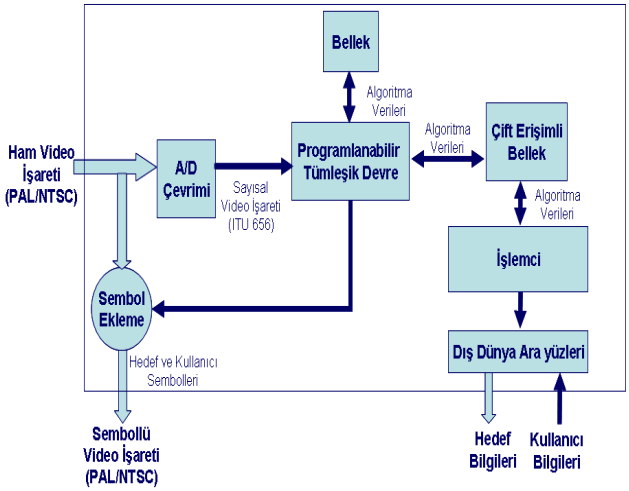
Algoritma, uygun fonksiyonlara ayrılmalı, her bir fonksiyonun donanım ya da yazılım üzerinde mi gerçekleneceğine karar verilmelidir. Tüm algoritma bloklarının girdi çıktıları netleştirilmeli, donanım yazılım arayüzleri ve bunlar arasındaki haberleşme yöntemi belirlenmelidir. Örnek bir donanım yapısı Şekil 3'te görülmektedir [2].
Algoritma paylaşımında genel yaklaşımda aşağıda belirtilen özelliklerdeki işlemler FPGA üzerinde gerçeklenir:

- Veri alınırken uygulanabilecek filtreleme ve türevi işlemler,
- Doğrusal fonksiyonlar,
- Paralellenebilecek bloklar,
- Dar boğaz oluşturan\oluşturabilecek bloklar,
- Değişmesi öngörülmeyen bloklar.

İşlemci üzerinde yani yazılım ile gerçeklenecek bloklar ise genelde aşağıdaki özellikleri taşırlar:

- Doğrusal olmayan bloklar,
- Döngü içerisinde döngüler vs. içeren karmaşık fonksiyonlar,
- Sık sık değişiklik ya da ayar gerektirebileceği düşünülen bloklar.

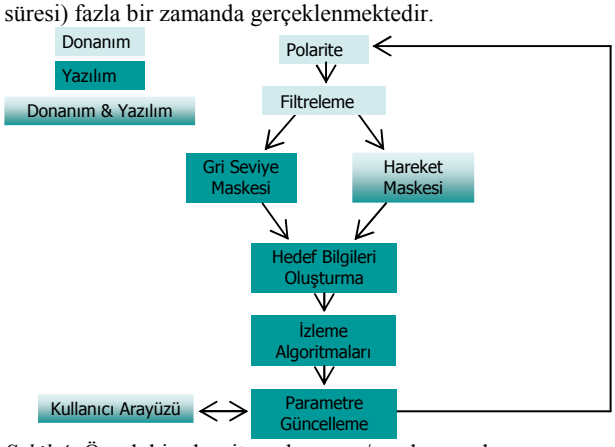
Algoritma paylaşımında genel yaklaşım, veri alınırken uygulanabilecek filtreleme ve türevi işlemlerin, doğrusal fonksiyonların FPGA üzerinde gerçeklenmesi, doğrusal olmayan, döngü içerisinde döngüler vs. içeren karmaşık fonksiyonların işlemci üzerinde yani yazılım ile gerçeklenmesi yönündedir. Algoritma tasarlanırken hangi algoritma bloğunun nerede gerçekleneceği fikri de oluşmaya başlar. Tüm algoritma bloklarının işlemcide gerçeklenmesi ve her biri için zaman bilgisinin oluşturularak, tek başına işlemcide gerçeklendiğinde dar boğaz oluşturan blokların tespit edilmesi, paylaşımda üzerine odaklanılması gereken algoritma bloklarını gösterir. İşlemcide gerçeklenmesinin çok uzun süreceği algoritma blokları karmaşık da olsalar FPGA'de ya da FPGA ile işlemci tarafından ortaklaşa gerçeklenebilirler.



Şekil 3: Örnek bir algoritma gerçekleme donanım yapısı

Algoritma bloklarından paralel çalışabilecek olanların belirlenmesi, uygun bir paylaşımla FPGA ve işlemci üzerinde aynı anda çalıştırılarak paralel koşturulması kısıtlı olan zamandan kazandıracaktır.
Algoritma paylaşımı yapılırken göz önünde bulundurulması gereken noktalardan bir tanesi de yazılım yapısının daha esnek olduğudur. FPGA ile geliştirme yapmak, yazılıma göre daha zordur ve donanım üzerinde çalışır hale getirmek daha fazla

zaman alır. Bu nedenle, değişebileceği ön görülen algoritma bloklarının işlemci üzerinde gerçeklenmesine çalışılabilir.
Şekil 3'te örnek bir görüntü işleme donanımı, bu donanım için yapılmış algoritma paylaşımı ise Şekil 4'te verilmiştir[1].
Şekil 4'te verilen yapıda donanım ile FPGA, yazılım ile kast edilen ise işlemcidir. Açık yeşil ile gösterilen algoritma blokları FPGA, koyu yeşil ile gösterilenler işlemci tarafından gerçeklenen algoritma bloklarıdır. İki renkli bloklar FPGA ve işlemci tarafından ortaklaşa gerçeklenen blokları, okların yönü ise haberleşme yönünü göstermektedir. Alınan görüntü sayısala çevrilmekte, FPGA'de gürültü giderme filtrelerinden geçirilerek, işlemci ile FPGA arasında bilgi ve veri alışverişinin gerçekleşmesini sağlayan sağlayan çift erişimli belleğe (DPSRAM-Dual Port Static Random Access Memory) yazılmaktadır. Gürültü giderme filtresi olarak 3x3'lük median filtre işlemci tarafından gerçeklendiğinde görüntü çerçevesi (frame) işleme süresinden (Şekil 1'de gösterilen tişlemci süresi) fazla bir zamanda gerçeklenmektedir.



Şekil 4: Örnek bir algoritma donanım/yazılım paylaşımı

Görüntünün alınıp, filtrelenmesinden sonra görüntü işleme algoritmaları devreye girmektedir. Bu algoritmaların bir kısmı FPGA, bir kısmı işlemci tarafından gerçeklenmektedir. Şekilden de anlaşıldığı üzere Hareket Maskesi algoritma bloğu FPGA ile işlemci arasında paylaşımlı olarak gerçeklenmektedir ve Gri Seviye Maskesi bloğu ile paralel olarak çalışmaktadır.
Şekil 3'te gösterilen donanımda FPGA ile işlemci arasındaki haberleşme çift erişimli bellek üzerinden sağlanmaktadır. Burada tutulacak verinin boyutu FPGA ve işlemci arasındaki algoritma paylaşıma bağlıdır. FPGA ya da işlemci tarafından bu alandan okunacak verilerin güncel olmasını sağlamak gerekir. Verinin büyüklüğüne göre veriyi yazmak ve güncel olmasını sağlamak belli bir zaman alacaktır.
Algoritma paylaşımı tamamlandıktan sonra yazılım ve donanım tasarım çalışmalarına başlanır. Yazılım ve donanım tasarım çalışmaları esnasında algoritmada değişiklik yapılması ihtiyacı doğabilir. Bu durumda algoritmada uygun değişiklik yapılır, test edilir ve donanım /yazılım paylaşımı tekrar değerlendirilir. Donanım/yazılım paylaşımı başlangıçta ne kadar doğru yapılırsa geri dönüşler o kadar az olur ve ürün daha hızlı bir şekilde ortaya çıkarılır. Bu süreci hızlandırmak için farklı yöntemler önerilmektedir.[3]

3.2. Eniyileme Çalışmaları

Algoritmanın donanım / yazılım paylaşımının yapılmasından sonra, ilgili algoritma bloklarının yazılımda gerçeklenmesi çalışmaları başlar. İlk aşamada benzetim kodunun aynı şekilde işlemci üzerinde gerçeklenmesi sağlanır. Burada önemli olan gerçeklenen kodun benzetim koduyla aynı şekilde çalışıyor, aynı sonuçları (ara ve son değerleri) üretiyor olmasıdır[1]. Bu aşamadan sonra eniyileme (optimization) çalışmaları başlar.
Eniyileme yöntemlerini aşağıdaki gibi gruplamak mümkündür:

- Kod eniyileme
- Algoritma eniyileme
- Derleyici bayraklarının kullanımı
- İşlemciye özel kodlama

3.2.1. Kod eniyileme

Kod eniyileme ile algoritmanın işleyişinde, girdi ve çıktılarında herhangi bir değişiklik yapmadan, sadece kod üzerinde yapılan eniyilemeler kast edilmektedir,

- Saymaç (register) kullanımı: İşlemcilerdeki genel amaçlı saymaçlar kısıtlı sayıda olduğu için, saymaçlara atanan değişkenlerin tanımı en çok kullanılandan başlamalıdır. Derleyici değişkenleri kodda yazılan sıraya göre atayacaktır.
- Karışık tipte aritmetik operasyonlardan sakınma: Aritmetik bir operasyon içerisinde farklı tipler kullanıldığında, otomatik olarak tip çevrimi yapılacaktır ve bu da fazladan işlem demektir.
- Döngü kodu içinde, döngü parametrelerine bağımlı olmayan hesaplamalardan kaçınılmalıdır, aksi takdirde N-defa aynı sonuç hesaplanacaktır.
- İşlemcilerde tamsayı, kayan nokta, aritmetik, vektör v.s. ve bellek işlemcileri için ayrı ayrı ve sayıları birden fazla olabilen işlem üniteleri vardır. Bu özellik nedeniyle, aynı anda birkaç işlem paralel olarak yapılabilmektedir. Örneğin tamsayı ünitesi dolu iken bekleyen komut da tamsayı işlemi ise diğer üniteler kullanılamamaktadır. İşlemcilerin bu özelliğinden yararlanabilmek için, kod içinde aynı tipte olan işlemlerin arka arkaya sıralanmaması, eşit olarak dağıtılması gerekmektedir.
- İşlemcilerde genellikle tek bir bellek okuma/yazma ünitesi mevcuttur. Kod içinde bellek işlemlerinin arka arkaya dizilmesi işlemci performansını azaltacaktır. Önce bellek işlemlerinin tamamlanıp daha sonra işlenmesi yerine sadece gerekli olan bellek işlemi yapılıp işlenmesi durumunda performans artacaktır.

Şekil 3'te verilen görüntü işleme donanımı için hazırlanan BSP(Board Support Package) tarafından sağlanan yüksek çözünülürlüklü zaman ölçme fonksiyonları kullanılarak dar boğaz oluşturabilecek, çok zaman alan fonksiyonlar belirlenmiş ve yukarıda anlatılan eniyileme yöntemleri uygulanmıştır.

3.2.2. Algoritma eniyileme

Algoritma eniyileme ile algoritmada yapılabilecek değişiklikler kast edilmektedir. Verileri 32 bit yerine 8 bitlik alanlarda tutmakla performanstan ne kadar kayıp olur? Bu kayıp önemli midir? Bir fonksiyonun içerisinde hesaplanan bir parametre gerçekten gerekli midir? Kullanılmaz ise kayıp ne olur? Kısacası her bir algoritma bloğunun veri ihtiyacının, girdi / çıktı parametrelerinin gözden geçirilmesi, vazgeçilebilecek, dolayısı ile algoritmayı hızlandıracak parametrelerin belirlenmesi gerekir. Örneğin eşiklenmiş bir görüntüde kümeleme algoritması ile hedefe / hedeflere ait ağırlık merkezi, sınırları, ortalama gri seviye değeri, minimum ve maksimum gri seviye değerleri, gri seviye standart sapması

vs. gibi tüm bilgileri bulmak mümkündür. Bu parametrelerden hayati bir önem taşımayanlar (minimum, maksimum gri seviye ve gri seviye standart sapma değerleri gibi) hesaplanmayabilir.
Bulunmak istenen bilgilerin tamlık derecesi önemlidir. Kabaca ama hızlı bir hesaplama ile ihtiyaç karşılanabiliyor ise ve olası kayıplar bir problem oluşturmuyor ise neden daha karışık ve daha fazla zaman alan bir algoritma kullanılsın? 8 komşuluğa bakan kümeleme algoritması yerine 4 komşuluğa bakan kümeleme algoritmasının kullanılması tercih edilebilir.
Yukarıda bahsedildiği gibi algoritma eniyileme ile çeşitli kayıplar söz konusudur, dolayısı ile bu kayıpların iyi değerlendirilmesi gerekir.

3.2.3. Derleyici eniyileme bayraklarının kullanımı

Yeniden kullanımı sağlamak için bütün platformlarda değiştirilmeden kullanılabilecek yazılımlar geliştirmek amaçlanmaktadır. Dolayısıyla platforma özel yapılabilecek eniyilemeler yazılım içerisinde yapılamamakta bu tür eniyilemeler derleyicilere bırakılmaktadır. Derleyici bayraklarının etkin kullanımı ile yazılım içerisinde değişiklik yapılmadan istenilen performansa ulaşılabilmektedir [4]
İşlemci mimarilerindeki gelişim ancak derleyicilerin işlemci mimarisini en uygun kullanması ile yazılım performansına yansıtılabilmektedir. Derleyicilerin çalışma ilkelerinin bilinmesi ve derleyici dostu kod geliştirilmesi ile derleyiciden en yüksek verimi almak mümkün olmaktadır.
Derleyici bayrakları genel olarak üçe ayrılabilirler [4]:

- C++ ve kod üretim bayrakları:
- Genel eniyileme bayrakları
- İşlemci tipine özel eniyileme bayrakları

3.2.4. İşlemciye özel kodlama

Kod ve algoritma eniyilemesi işlemciden bağımsız eniyileme yöntemleridir. Algoritma bloklarının daha hızlandırılması için bir sonraki adım, algoritmanın koşacağı işlemciye göre kodlama yapmak, işlemcinin sağladığı yetenekleri kullanmaktadır. Çevirici dili (assembly) seviyesinde kod yazarak ya da örneğin PowerPC işlemcilerde Altivec alt yapısını kullanarak algoritmaları daha hızlandırmak mümkün olabilir. Çevirici dili seviyesinde kod yazmak zordur, Altivec kullanımı ise her algoritmaya uygun olmayabilir. Bu kodlar elle yazılabileceği gibi, hazır kütüphane (DSP ya da PowerPC'ler için sinyal işleme, görüntü işleme kütüphaneleri, Altivec kütüphaneleri vs.) kullanımı da tercih edilebilir.
Yukarıda bahsedilen eniyileme yöntemleri ile kümeleme algoritması bloğu için elde edilen hızlanma düzeyleri Tablo 1'de gösterilmektedir.

Eniyileme Yöntemi	Süre (µs)	Hızlanma (%)
Standart	78459	
Kod eniyileme	6581	91.61
Algoritma eniyileme	5830	11.41
İşlemciye özel kodlama (Altivec kullanımı)	3382	41.99
Derleyici bayraklarının kullanımı (O2)	3182	5.91

Tablo 1: Kümeleme algoritması bloğu için eniyileme yöntemleri ile elde edilen hızlanma miktarı

Tablo1'de gösterilen sonuçlar, PPC604 çekirdekli bir işlemci üzerinde Tornado 2.2 aracı ile birlikte sağlanan GCC ver:2.96 derleyicisi ve vxWorks 5.5 işletim sistemi kullanılarak aynı görüntünün işlenmesiyle elde edilen ortalama süreleri göstermektedir. Tabloda en fazla hızlanmanın kod eniyilemesi ile sağlandığı görülmektedir. Burada önemli olan, standart olarak yazılan kodda algoritmanın yerine getirilmiş olmasıdır, verilere ulaşım, indeksleme, for döngüleri vs. gibi unsurlara dikkat edilmemiştir. Kod eniyilemesi ile algoritma da herhangi bir değişiklik yapılmamıştır. Algoritma eniyilemesi ile minimum, maksimum gri seviye değerleri ile gri seviye standart sapma değeri hesaplanmamıştır. Tabloda derleyici bayraklarından O2'nin kullanımıyla %5.91'lik bir artış görülmektedir. Bu artış sadece kümeleme algoritması için geçerlidir, aynı görüntü için tüm algoritma zamanına bakıldığında yaklaşık %25'lik bir iyileşme görülmüştür. Eniyileme yöntemleri ile elde edilecek zaman kazanımının kullanılan algoritmaların yapısına bağlı olduğu göz önünde bulundurulmalıdır.

4. Testler

Gerçek zamanlı gömülü sistemleri için testler algoritma tasarımı ile başlar ve gerçek donanım üzerinde donanım/ yazılım paylaşımı ve eniyileme çalışmaları sonrasında devam eder. Bu kapsamdaki testleri laboratuar testleri olarak tanımlayabiliriz. Laboratuar testlerinden sonra, donanım üzerinde gerçeklenmiş algoritmaların gerçek senaryolar ile gerçek ortamında test edilmesi önemlidir. Bu testler yapılmadan sistem sadece masa üzerinde test edilmiş olacak, gerçek kullanım amacına hizmet edip etmediği tam olarak anlaşılamayacaktır. Bu nedenle sistemi mümkün olduğunca gerçek verilerle, gerçek ortamında test etmek gerekir. Arazi testleri ile sistemin ya da algoritmaların masa üzerinde görülmeyen eksiklikleri / hataları tespit edilebilir ve düzeltme çalışmaları sonrası bu eksikler algoritmada yapılabilecek değişiklikler / eklemeler ile giderilir.
Donanım / yazılım paylaşımı yapılmış, eniyilemelerle gerçek zamanlı olarak koşar hale getirilmiş, arazi testlerine çıkılmış bir sistemde algoritma değişiklikleri / eklemeleri nasıl yapılır, yapılan değişiklikler nasıl test edilir? Bu önemli bir konudur. Algoritmada yapılacak herhangi bir değişikliğin / eklemenin algoritma ekibi ile birlikte yapılması gerektiği belirtilmişti. Yapılacak değişiklikler / eklemeler algoritma ekibi ile birlikte belirlenir, algoritma kodu (benzetim kodu) denenir ve donanım üzerinde gerçeklenir. Bu, algoritma kodu ile algoritmanın gerçeklendiği uygulamanın (donanım / yazılım) paralel götürülmesi gerektiği anlamına gelmektedir. Arazi testleri esnasında uygulama kodunda değişiklik yapılması tercih edilen bir durum değildir, ama bazen gerekmektedir. Bu durumda yapılan değişikliklerin algoritma koduna yansıtılması gerekir ki paralellik sağlanabilsin.
Uygulama yazılımı, donanımdan bağımsız, test verileri ile bilgisayar üzerinde çevrim dışı (off-line) olarak çalışabilir şekilde geliştirilebilir. Bu durumda uygulama yazılımı algoritma geliştirme / ekleme / değiştirme amaçlarıyla da kullanılabilir. PowerPC'ler için model tabanlı araçlar (Rhapsody vs.) kullanılarak bu gerçeklenebilir. Bunun için hem donanım üzerinde hem işlemci üzerinde koşacak olan algoritma blokları C ya da C++ ile gerçeklenir, donanım / yazılım paylaşımına göre bloklar arası haberleşmeyi benzeten yazılım blokları, test verilerini okumak, yazmak için alt yapılar eklenir. Gerçek donanım kullanıldığında sadece yazılım

blokları çalışacak, çevrim dışı çalışma durumunda ise tüm algoritma blokları çalışacak şekilde yapılandırmalar oluşturulur. Bu şekilde hem algoritma değiştirme / ekleme ve test etme kolaylaşır, çünkü uygulama yazılımı donanımdan bağımsız olarak test verileri ile çevrim dışı çalışabilir durumdadır, hem de hızlıca yapılan değişikliklerin gerçek zaman performansını görmek mümkün olur. Burada sözü geçen değişiklikler /eklemeler, yazılım ile gerçeklenen yani işlemci üzerinde koşan algoritma bloklarında yapılmaktadır, donanım üzerinde koşan algoritma blokları için farklı bir yol izlemek gerekir. Ayrıca arazi testleri sonrasında yapılacak değişiklikler daha çok yazılım ile gerçeklenir.

Simulink, Labview, Gedae gibi model tabanlı tasarım araçları kullanarak hem algoritma geliştirme hem de geliştirilen algoritmayı doğrudan gerçek donanım üzerinde koşturmak mümkündür [1] . Bu araçları kullanabilmek için seçilen işlemcinin / donanımın bu araçlar tarafından destekleniyor olması gerekir. Bu araçlar ile algoritma kodundan işlemci / donanım üzerinde koşacak kod otomatik olarak oluşturulduğu için, çok optimize kod elde edilemeyebilir, bu kullanılan algoritmanın yapısına ve karmaşıklığına bağlıdır.

5. Tartışma ve Sonuç

Gerçek zamanlı gömülü sistemlerde basitten karmaşığa pek çok algoritmaya ihtiyaç duyulabilmektedir. Basit algoritmalar için farklı bir yöntem izlemeye gerek yokken, karmaşık algoritmalar için donanım seçiminden algoritma tasarımına, gerçeklemeye kadar dikkat edilmesi gereken pek çok nokta vardır. Bu bildiride daha çok algoritma gerçekleme süreci üzerinde durulmuş, algoritmanın donanım / yazılım paylaşımı, uygulama yazılımı geliştirme, kod, algoritma ve diğer eniyileme yöntemlerinden bahsedilmiştir. Örnek bir donanım üzerinde, örnek bir donanım /yazılım paylaşımı aktarılmıştır, bu konuda göz önünde bulundurulması gereken konular belirtilmiş, eniyileme çalışmalarının ne kadar önemli ve gerekli olduğu gösterilmiştir.

Sistemi gerçek ortamında gerçek veriler ile test etmenin önemi üzerinde durulmuş, testler esnasında bulunan hataların / eksikliklerin algoritma ekibi ile birlikte çalışılarak giderilmesi gerektiği belirtilmiştir.

6. Kaynaklar:

[1] Abkairov N., Nazarov A., Tools of the trade: successful DSP-software development and testing, Spirit Corp, EDN Magazine, p.71-74, 02/21/2002.

[2] Erdoğdu S., Özer E., Kürekli, K., Gürel, İ., Akçay K., Algoritmadan Uygulamaya Video Hedef İzleyici Birimi, ASELSAN A.Ş Savunma Sistem Teknolojileri (SST) Grubu, Savtek 2008 Bildiriler, Cilt-II, s.139-146, 26-27/06/2008

[3] Hein C., Real-time implementation of super-resolution imaging algorithm, Lockheed Martin Advanced Technology Laboratories, Proc. SPIE Vol. 3461, p.501-511, Advanced Signal Processing Algorithms, Architectures, and Implementations VIII.

[4] İyidir B., Kalay A., İşbitiren G., Yılmaz Ö., Gerçek Zamanlı Gömülü Yazılımlarda Derleyici Bayraklarının Etkin Kullanımı, ASELSAN A.Ş Savunma Sistem Teknolojileri (SST) Grubu, Yazılım Kalitesi ve Yazılım Geliştirme Araçları 2008 YKGS2008, 9-10 Ekim, TC İstanbul Kültür Üniversitesi, İstanbul.

[1] http://www.mathworks.com/products/simulink/
http://www.gedae.com/
http://www.ni.com/labview/