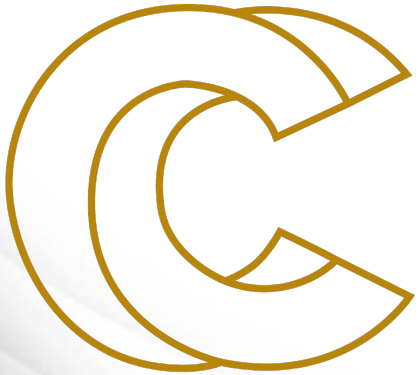


EURO

OpenMP
Kuramsal Bilgiler

Giriş

Paralel Programlama Nedir?

- Amaç
  - sonuca ulaşma süresini kısaltmak
    - daha büyük boyutta veri
    - daha yüksek çözünürlük
    - daha çok deney
    - süre kısıtlarına uyma
- Yöntem
  - daha çok kaynak kullanmak
    - işi kendi içinde bağımsız parçalara ayırmak
    - parçaları birlikte çalışan birimlere dağıtmak

Yaklaşımlar

- Yazılımda gerekli değişikliğe göre …
  - otomatik paralelleştirme
    - derleyici desteği ile
  - hazır kütüphane kullanımı
    - *BLAS*
  - dil içerisindeki olanakların kullanımı
    - *C++17*
  - özelleşmiş donanıma devretme
    - çeşitli yöntemler
  - **paralel bölgeleri belirtme**
    - **OpenMP**
  - **yeniden yapılandırma**
    - **MPI**

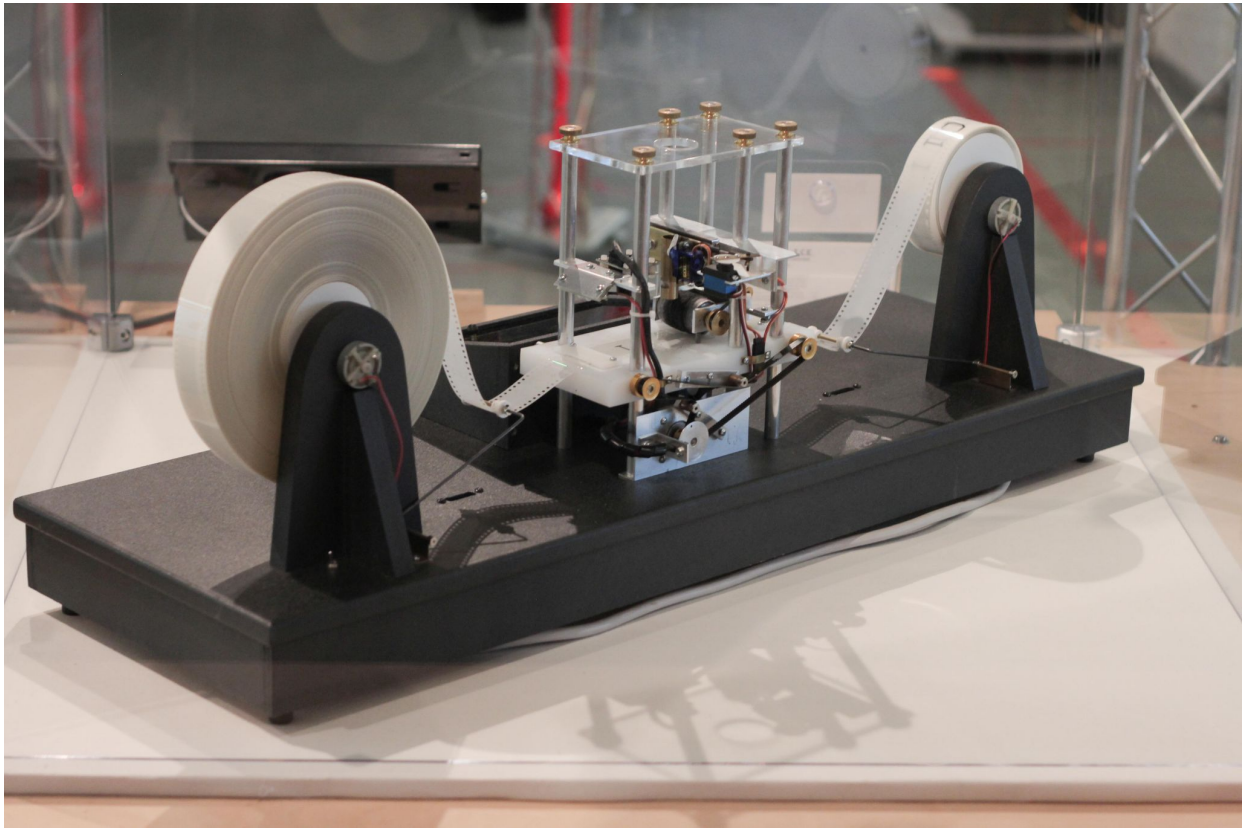
Yaklaşımlar

- Veriye erişim biçimine göre …
  - ortak bellek alanı
    - *multithreading*
      - OpenMP
  - dağıtık bellek alanı
    - *message passing*
      - MPI

Program Nedir?

- İş akış bilgisi
  - işlemler
    - okuma
    - yazma
    - dönüşüm
  - dallanma
    - koşullu
    - koşulsuz

- Veri alanı
  - konum bilgileri
    - ad
    - uzaklık
  - değerler
    - girdiler
    - ara değerler
    - çıktılar



Turing Düzeneği
Rocky Acosta, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>,
via Wikimedia Commons

Ortak Bellek / Dağıtık Bellek

- Ortak bellek
  - tüm işlem birimleri aynı veri alanına erişiyor
  - erişim anlık, ek çaba gerektirmiyor
  - yarış durumları (ing. *race condition*) sorun yaratıyor
- Dağıtık bellek
  - her işlem biriminin kendi veri alanı var
  - veri paylaşımı ayrıca çaba gerektiriyor
  - kördüğümler (ing. *deadlock*) sorun yaratıyor

OpenMP

OpenMP'ye Genel Bakış

- Varolan dili kullanarak
  - C/C++, Fortran
- Program üzerinde ek yönergeler ile
  - #pragma
- Ortak bellek alanında
  - değişkenlerde aynı ad, ayrı konum
- Genellikle SIMD (Single Instruction Multiple Data) yöntemi ile
  - *parallel for*
- Yaygın derleyici desteğine güvenerek
  - GCC, LLVM/CLang, …

OpenMP'nin Geçmişi

- v1.0
  - Fortran (1997), C/C++ (1998)
- v2.0
  - Fortran (2000), C/C++ (2002)
- v2.5
  - Fortran ve C/C++ (2005)
  - akış paylaşımı üzerine çalışmalar (2007, eskiden sadece döngü paylaşımı odaklı)
- v3.0
  - *Task* kavramı (2008)
- v4.0
  - özel donanıma devretme, *atomics*, kullanıcı tanımlı indirgeme işlevleri, … (2013)
- v5.0
  - *task reductions*, akış yönetimi ve takımlar ile ilgili geliştirmeler (2018)
- v5.2
  - güncel sürüm

Derleyici Desteği

- v3.1
  - GCC 4.3
  - LLVM/CLang 3.7
- v4.5
  - GCC v6
  - LLVM/Clang v12
- v5.0
  - GCC v9
  - LLVM/Clang v12

Kavramlar

Kavramlar: *pragma*

- tanım
  - derleyici için yönlendirme
  - programın algoritmasını değiştirmiyor
  - akış ve veri paylaşımı ile ilgili ek bilgi sağlıyor
  - çalıştırılma biçimini değiştiriyor
- örnek
  - C/C++
    - `#pragma omp parallel`
  - Fortran
    - `!$OMP PARALLEL`

Kavramlar: *region*

- tanım
  - program içerisinde sınırlı bir bölge
  - giriş ve çıkışta ek işlemler gerçekleşebilir
    - *thread*'lerin ayrılması/birleşmesi
    - yerel değişkenler tanımlama
      - bellekte ad-konum ilişkilerinin değiştirilmesi
    - indirgeme işlemler
- örnek
  - C/C++
    - `if { … }`
    - `for { … }`

Kavramlar: *parallel / critical region*

- tanım
  - parallel region
    - aynı anda birden çok kopyası çalışan bölge
  - critical region
    - aynı anda sadece bir kopyası çalışan bölge
- örnek
  - parallel region
    - aynı işlemi farklı veri aralıklarına uygulayan işlevler
  - critical region
    - ortak bir kaynağa (örn: çıktı dosyası) erişen işlevler

Kavramlar: *atomic*

- tanım
  - “kesilmeyen”, yarıda kalmayacak
  - işlem ya da bölge için geçerli olabilir
  - donanım desteği ile sağlanıyor olabilir
- örnek
  - bir sayacın okunup güncellenmesi

Kavramlar: *barrier*

- tanım
  - tüm kopyaların, diğerleri gelene kadar birbirini beklediği adım
  - çeşitli işlemlerin yan etkisi olarak ortaya çıkabilir
  - özel olarak belirtilebilir
- örnek
  - paralel bölge sonunda

Kavramlar: *reduction*

- tanım
  - birden çok değeri tek değere indirgeyen işlem
  - eski sürümlerde ön tanımlı bir küme ile sınırlı
  - OpenMP v4.0 ve sonrasında kullanıcı tanımlayabilir
- örnek
  - bir dizinin en küçük değerini bulma

Kavramlar: *fork / join model*

- tanım
  - fork
    - programın, akışın bir aşamasında birden çok kopyaya ayrılması
  - join
    - birden çok kopyanın, tek kopyaya indirgenmesi
  - “aynı akış, ayrı veri”
- örnek
  - *non-deterministic finite automaton*

Kavramlar: *shared / private variable*

- tanım
  - shared variable
    - bir adın, her kopyada aynı konumla eşleşmesi
  - private variable
    - bir adın, her kopyada özgün bir bellek konumunu göstermesi
- örnek
  - yerel toplamlar

Kavramlar: *scheduling*

- tanım
  - dağıtılan işlerin boyut ve zamanlamasının yönetimi
    - kaç kopya başlatılacak?
    - her kopyaya ne kadar iş düşecek?
    - iş parçası sayısı çalışan kopya sayısından fazla ise?
  - çeşitli yaklaşımlar olası
  - kullanıcı yönlendirebilir
- örnek
  - *round robin*
  - 10 iş parçası, 3 kopya
    - 4+3+3
    - (2+2+2) + (2+2+0)

Kavramlar: *race condition*

- tanım
  - sonucun bir yarış ile belirlendiği durumlar
  - kazanan öngörülemeyen kurallarla belirlenir
    - bilgisayarın yük durumu
    - kopyaların çalışma sırası
    - önbelleğin etkileri
  - incelenmesi en zor sorunlar arasındadır
- örnek
  - aynı konuma aynı anda birden çok kişinin veri yazması

Kavramlar: *deadlock / livelock*

- tanım
  - döngüsel bağımlılık nedeniyle sürecin kördüğüm olması
  - *deadlock*
    - bağımlı süreçlerin birbirini bekleyerek aynı durumda kalması
  - *livelock*
    - süreçlerin işlem yapmaya devam ederek aynı döngüde kalmaları
- örnek
  - deadlock
    - kapıda diğerinin geçmesini bekleyen kişiler
    - aynı anda birbirine yol veren kişiler

Örnek

```
#include <omp.h>
#include “islevler.h”

int main(int argc, char* argv[]) {
    int x=0, y=0;

    x = oku();
    #pragma omp parallel shared(x) reduction(+:y)
    {
        y = isle(x, omp_get_thread_num());
    }
    yaz(y);

    return 0;
}
```

```
#include <omp.h>
#include “islevler.h”

int main(int argc, char* argv[]) {
    int x=0, y=0;

    x = oku();
    #pragma omp parallel shared(x) reduction(+:y)
    {
        y = isle(x, omp_get_thread_num());
    }
    yaz(y);

    return 0;
}
```

İşlevler ve Çevre değişkenleri

İşlevler: omp_get_num_threads

- int omp_get_num_threads(void);

- çalışan kopya sayısını verir
- paralel bölge dışında 1 değerini verecektir

İşlevler: omp_set_num_threads

- void omp_set_num_threads(int num_threads);

- çalışacak kopya sayısını belirler

İşlevler: omp_get_thread_num

- int omp_get_thread_num(void);

- çalıştıran kopyanın sıra numarasını verir
  - sıralama 0’dan başlar
- paralel bölge dışında 0 değerini verecektir

Çevre değişkenleri

- OMP_NUM_THREADS
  - çalışacak kopya sayısını belirler

- OMP_SCHEDULE
  - iş bölümü seçimlerini yönetir

omp parallel

omp parallel

```
#pragma omp parallel
    if(scalar-expression)
    num_threads(integer-expression)
    default(shared | none)
    private(list)
    firstprivate(list)
    shared(list)
    copyin(list)
    reduction(operator: list)
```

omp parallel if(*scalar-expression*)

- bölge, ancak *scalar-expression* 0 olmayan bir değer alırsa paralel olarak çalıştırılır
- sadece bir adet *if* belirtilebilir

omp parallel num_threads(*integer-expression*)

- bölgenin *integer-expression* tane kopyası çalıştırılır
- *dynamic scheduling* belirtildiyse, aynı anda en fazla çalışacak kopya sayısını belirtir

omp parallel default(shared | none)

- bölge içerisindeki değişkenler için varsayılan paylaşım türünü seçer
- default(shared)
  - tüm değişkenler tüm kopyalar için aynı bellek konumunu gösterir
- default(none)
  - bölge içerisinde erişilmesi istenen tüm değişkenlerin belirtilmesi gerekir
  - aşağıda sayılanlar bu kuralın dışındadır
    - const olarak belirtilen değişkenler
    - *parallel for* içinde kullanılan sayaç
- sadece bir kez *default* belirtilebilir

omp parallel private(*list*)

- *list* içerisinde belirtilen değişkenler, her kopya için ayrı bir bellek konumunu gösterir
- değişken adları, virgül ile ayırılarak belirtilir
- değişkenlerin ilk değerleri belirlenmemiştir
- bölge içerisinde yeni tanımlanan değişkenler *private* sayılırlar

omp parallel firstprivate(*list*)

- *list* içerisinde belirtilen değişkenler, her kopya için ayrı bir bellek konumunu gösterir
- değişken adları, virgül ile ayırılarak belirtilir
- değişkenlerin ilk değerleri, değişkenin bölge öncesindeki değerine eşittir

omp parallel shared(*list*)

- *list* içerisinde belirtilen değişkenler, her kopya için aynı bir bellek konumunu gösterir
- değişken adları, virgül ile ayırılarak belirtilir
- değişkenlerin ilk değerleri, değişkenin bölge öncesindeki değerine eşittir

omp parallel copyin(*list*)

- *list* içerisinde belirtilen değişkenler, her kopya için ayrı bir bellek konumunu gösterir
- değişken adları, virgül ile ayırılarak belirtilir
- değişkenlerin ilk değerleri, değişkenin *master thread* içindeki değerinden kopyalanır

omp parallel reduction(*operator*: *list*)

- *list* içerisinde belirtilen değişkenler, belirtilen *operator* ile indirgenir
  - değişkenler her kopya için ayrı bir bellek konumunu gösterir
  - indirgeme işlemi bölge sonunda gerçekleştirilir
  - indirgeme sonucu çıkışta aynı adlı değişkene yazılır
- değişken adları, virgül ile ayırılarak belirtilir
- örnek işlemler ile ilk değerleri
  - + (0)
  - * (1)
  - max (veri tipindeki en küçük sayı)
  - min (veri tipindeki en büyük sayı)

Örnek

```
#include <omp.h>
#include “islevler.h”

int main(int argc, char* argv[]) {
    int *x = olustur_x();
    int *y = olustur_y();

    #pragma omp parallel for
    for (int i=0; i<4096; i++) {
        y[i] = isle(x[i]);
    }
    yaz(y);

    return 0;
}
```

```
#include <omp.h>
#include “islevler.h”

int main(int argc, char* argv[]) {
    int *x = olustur_x();
    int *y = olustur_y();

    #pragma omp parallel for
    for (int i=0; i<4096; i++) {
        y[i] = isle(x[i]);
    }
    yaz(y);

    return 0;
}
```

omp parallel for

omp parallel for

```
#pragma omp parallel
{
    #pragma omp for
    {
        for (int i=0; i<N; i++) islem(i);
    }
}
```

omp parallel for

```
#pragma omp parallel for
{
    for (int i=0; i<N; i++) islem(i);
}
```

omp parallel for

- biçim sıradan olmalı
  - for (sayaç **=** ilkdeğer; sayaç **karşılaştırma** sınırdeğer; **değişim**) { … }
    - döngünün tekrar sayısı öngörülebilir olmalı
    - karşılaştırma **<**, **<=**, **>**, **>=** arasından seçilmeli
    - **değişim**, “sayaç = sayaç + adımboyu” ya da “sayaç++” (ve - karşılıkları) olmalı
    - sayaç, döngü içerisinde değiştirilmemeli
- parallel için kullanılan tanımlar (private, reduction, …) geçerli
- sayaç *private* olarak tanımlanır

schedule

schedule

- schedule(tür[, boyut])
- tür
  - static
  - dynamic
  - guided
  - auto
  - runtime
- boyut
  - artı değerli bir tamsayı
  - isteğe bağlı

schedule(static) / schedule(dynamic)

- static
  - boyut belirtilmezse iş eşit parçalara bölünür
  - boyut belirtildiyse iş boyut büyüklüğünde parçalara bölünür
    - artan parçalar çalışanlara sıra ile dağıtılır
- dynamic
  - boyut belirtilmediyse 1 alınır
  - boyut belirtildiyse iş boyut büyüklüğünde parçalara bölünür
    - artan parçalar çalışanlara sırayla elindeki işi ilk bitirene verilir

schedule(guided) / schedule(auto)

- guided
  - dynamic seçeneğine benzer
    - ama eşit boyutlar yerine giderek azalan boyutlar kullanılır
    - boyut, en küçük parça boyu boyutunu belirler
  - boyut belirtilmediyse 1 olarak alınır
- auto
  - dağıtım, tamamen çalışma ortamı tarafından belirlenir

schedule seçimi

- static
  - dengeli dağılım olası ise en uygun seçimdir
- dynamic
  - parça işleme süreleri değişken ise önerilir
  - ardışık veri az olduğundan *cache* kullanımı kötü olabilir
- guided
  - dynamic'e göre daha iyi bir dağılım sağlar
  - ilk adımlarda geçen süre ilerideki kararları belirler
- auto
  - adım sayısı çok ise sistem yükü tanıyarak daha doğru karar verebilir

single / master

omp single

- sadece bir kez çalışır
- bu bölgeye ulaşan ilk kopya tarafından çalıştırılır
- diğer tüm kopyalar doğrudan bölgenin sonuna gider
- çıkışı bir buluşma noktası oluşturur

omp master

- sadece bir kez çalışır
- sıra sayısı 0 olan (master) kopya tarafından çalıştırılır
- diğer tüm kopyalar doğrudan bölgenin sonuna gider
- çıkışı bir buluşma noktası oluşturmaz

omp single örneği

```
#pragma omp  parallel
{
    x = ozgun_girdi();

    #pragma omp single
    {
        y = ortak_girdi();
    }

    isle(x,y);
}
```

barrier

omp barrier

- kopyalar, diğerleri bariyere ulaşana kadar birbirini bekler
- **for** ve **single** bölgeleri sonunda kendiliğinden oluşur
- el ile de belirtilebilir

omp barrier örneği

```
#pragma omp parallel
{
    …
    x[i] = islem();
    …
    #pragma omp barrier
    y = x[j]
    …
}
```

critical / atomic

omp critical

- #pragma omp critical[(özgünad)]
- verilen bir anda sadece bir kopya çalıştırabilir
- program içerisinde birden çok *critical* bölge olabilir
  - bölgeler ad verilerek birbirinden ayırılabilir
- ortak kaynaklara erişimi denetlemek için kullanılır

omp atomic

- #pragma omp atomic
- tek bir işlemin kesintisiz gerçekleşmesini sağlar
- işlem, aşağıdaki biçimlerden birinde olmalıdır
  - x OP y
    - OP: +=, *=, -=, /=, &=, ^=, <<=, >>=
  - x++
  - x--
  - ++x
  - --x
- *critical* bölge kullanmaktan daha verimli olabilir

omp critical örneği

```
#pragma omp parallel private(x)
{
    x = islem();
    #pragma omp critical(cikti)
    {
        dosyaya_yaz(x);
    }

}
```

Sorular ve Tartışma

EURO

Thanks!

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, United Kingdom, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, Republic of North Macedonia, Iceland, Montenegro