Pemrograman Memori Shared: Thread dan OpenMP

Eko Didik Widianto (didik@undip.ac.id)

Lab Embedded, Siskom - Undip

Pengolahan Paralel

- Pokok Bahasan
  - **Konsep dan terminologi**
  - **Arsitektur memori paralel**
  - **Model pemrograman paralel**
  - **Desain Program Memori Shared: Thread dan OpenMP**
  - Memprogram paralel
- Referensi:
  - Introduction to Parallel Computing: https://computing.llnl.gov/tutorials/parallel_comp/

Bahasan

Pemrograman Shared Memory

- Proses dan Memori
- Memori Shared dan Pemrograman

Pemrograman Thread

- Pustaka Thread (Sistem)
- POSIX Threads

Pemrograman OpenMP

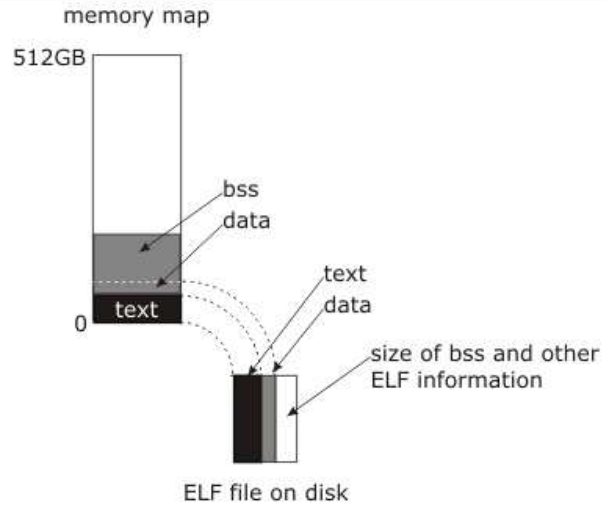
- Tentang OpenMP
- Model Eksekusi
- Sharing Data

Pemrograman Paralel di Arsitektur Memori Shared

Proses dan Memori

- Program dan Process
  - Program: source code dengan bahasa pemrograman tertentu atau file executable
  - Process/task: konsep OS tentang program yang sedang berjalan
  - Program yang dijalankan di shell akan membuat satu proses baru (spawn, fork)
- Program berisi statemen executable dan deklarasi data
  - Tiap data mempunyai properti *storage class* (durasi: permanen/statik, automatic/temporary) dan *scope* (global, lokal)
  - Properti menentukan penempatannya di virtual memori
  - Data yang dideklarasikan di luar fungsi: scope global, permanen
  - Data yang dideklarasikan di dalam fungsi (termasuk *main*): scope lokal, temporary
  - Data dapat dideklarasikan permanen dengan keyword *static*

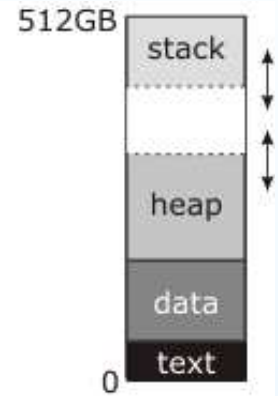
Peta Memori: Text, Data, BSS



- Kompiler menerjemahkan statemen executable ke instruksi CPU, dan deklarasi data ke bentuk sesuai spesifikasi mesin

- Linker membuat file executable dan menghimpun instruksi dan data di segment yang berbeda
  - Instruksi dimasukkan ke **segmen text** (RO)
  - Data terinisialisasi, data statik, konstanta masuk ke **segmen data** (RW)
  - Data tidak terinisialisasi masuk ke **bss** (*block started from symbol*)
- Contoh program

Peta Memori: Heap, Stack

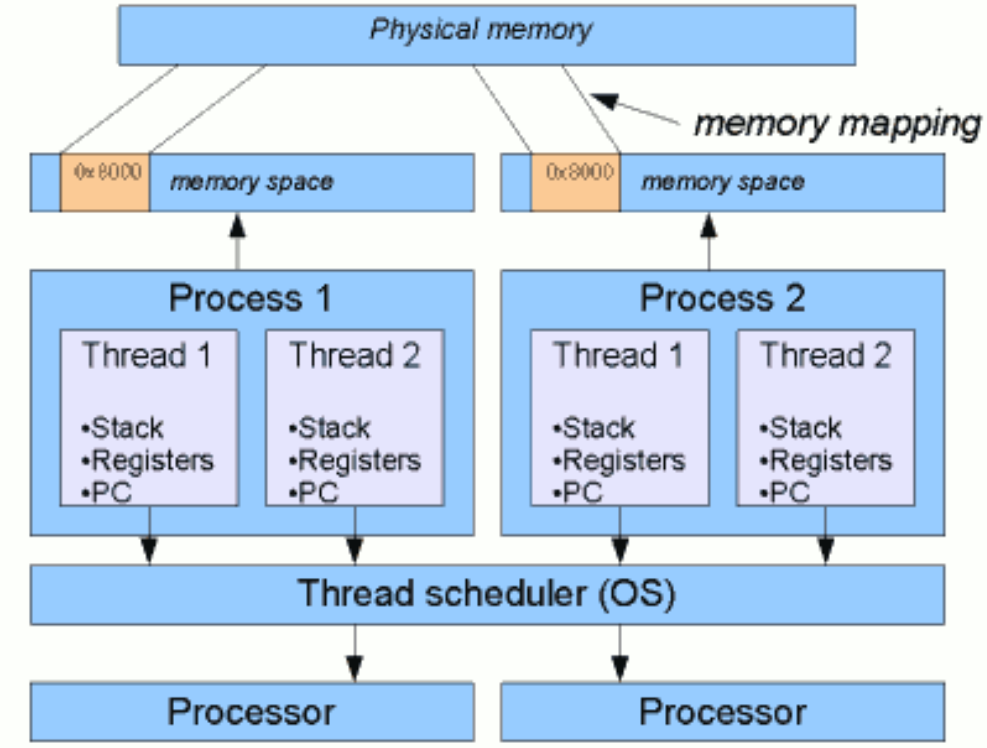


- Alamat memori virtual mulai dari 0 sampai 512GB. Alamat di atas 512GB reserved untuk kernel Linux. Tiap arsitektur mungkin limitnya beda
- size dari process (text+data+bss) ditentukan saat kompilasi dan konstan selama eksekusi
- Segmen heap untuk alokasi memori dinamik (misalnya dengan fungsi *malloc*)
- Segmen stack untuk variabel lokal fungsi/sub-rutin

- Text, data, bss dipetakan ke memori fisik dengan page table
- Heap dan stack: memori secara dinamis dialokasikan dan dibebaskan (allocate, deallocate), sehingga page table juga diupdate secara dinamis
- Baca: http://www.ualberta.ca/CNS/RESEARCH/LinuxClusters/mem.html

Memori Shared dan Model Pemrograman

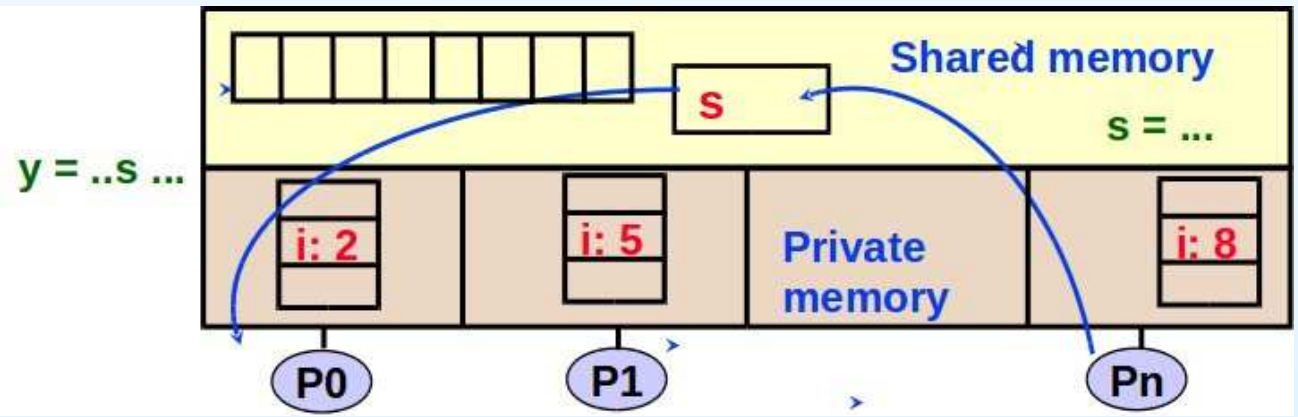
Recall Thread:



(source: http://www.javamex.com/tutorials/threads/how_threads_work.shtml)

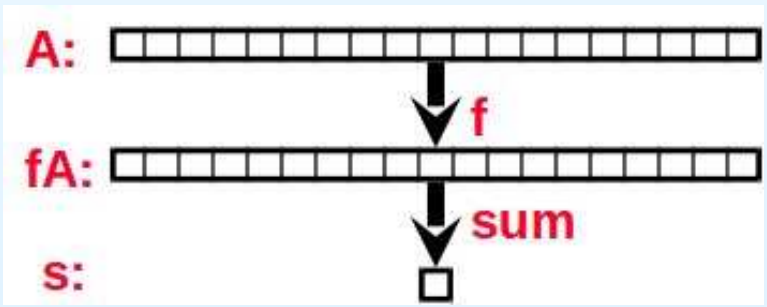
Memori Shared dan Model Pemrograman

- Process merupakan kumpulan dari thread
  - Dapat dibuat secara dinamik, di tengah eksekusi
  - Tiap thread mempunyai variabel private (local stack)
  - Juga mempunyai variabel shared (variabel statik atau heap global), misalnya s
  - Antar thread berkomunikasi secara implisit dengan menulis/membaca variabel shared
  - Antar thread berkoordinasi dengan sinkronisasi variabel shared



Contoh Problem

- Misalnya: buat program untuk $\sum_{i=0}^{n-1} f(A[i])$
- Dimana akan menempatkan A? semuanya di memori? atau dipecah?
- Operasi mana yang akan dilakukan oleh tiap prosesor?
- Bagaimana mengkoordinasi tiap hasil pecahan operasi?
  - A = array dari semua data
  - fA = $f(A)$
  - s = sum(fA)



Strategi Memori Shared

- Strategi: jumlah prosesor p << n, menggunakan memori tunggal
- Dekomposisi paralel: tiap evaluasi dan tiap penjumlahan parsial adalah sebuah task
- Berikan tugas sejumlah $\frac{n}{p}$ bilangan ke tiap prosesor
  - Tiap prosesor menghitung secara independen hasil (privat) dan jumlah parsialnya
  - Mengumpulkan p jumlah parsial dan menghitung jumlah global
- Kelas data:
  - Shared: jumlah global, nomor n
  - Private: evaluasi fungsi, bagimana dengan jumlah parsial?

Kondisi Race

- Apa masalah di program berikut?

```
fork(sum,a[0:n/2-1]);
sum(a[n/2,n-1]);

static int s = 0;

Thread 1

  for i = 0, n/2-1
      s = s + f(A[i])

Thread 2

  for i = n/2, n-1
      s = s + f(A[i])
```

- Kondisi **race** atau data **race** dapat terjadi
  - Saat dua prosesor (dua thread) mengakses variabel yang sama dan setidaknya satu thread melakukan operasi penulisan
  - Saat akses-akses tersebut terjadi bersamaan (tidak disinkronkan)

Kondisi Race: Hasil Tidak Seperti Yang Diinginkan

A= | 3 | 5 | f (x) = x^2

static int s = 0;

Thread 1		Thread 2	
....		...	
compute f([A[i]) and put in reg0	9	compute f([A[i]) and put in reg0	25
reg1 = s	0	reg1 = s	0
reg1 = reg1 + reg0	9	reg1 = reg1 + reg0	25
s = reg1	9	s = reg1	25
...		...	

- Misalnya A=[3,5], $f(x) = x^2$, dan s=0 (init)
- Di akhir eksekusi, nilai s seharusnya $3^2 + 5^2 = 34$
  - Tapi nilainya bisa 34, 9 atau 25
- Operasi **atomik** adalah pembacaan dan penulisan
  - Tidak ada 1/2 dari satu bilangan, operasi += adalah atomik
  - Semua komputasi terjadi di register private

Perbaikan Kode

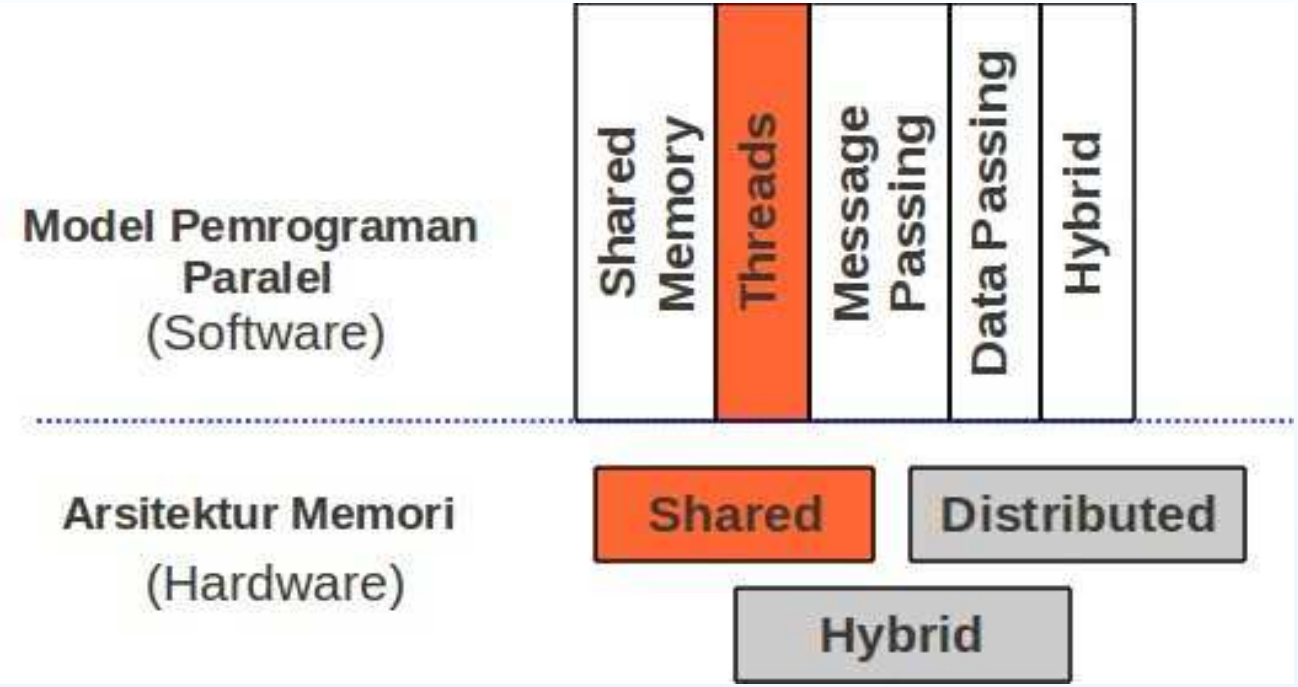

```
static int s = 0;
static lock lk;
```

```
Thread 1

  local_s1= 0
  for i = 0, n/2-1
     local_s1 = local_s1 + f(A[i])
  lock(lk);
  s = s + local_s1
  unlock(lk);
```

```
Thread 2

  local_s2 = 0
  for i = n/2, n-1
     local_s2= local_s2 + f(A[i])
  lock(lk);
  s = s +local_s2
  unlock(lk);
```

- Karena penjumlahan bersifat asosiatif, maka urutan penjumlahan tidak jadi masalah
- Sebagian komputasi menggunakan variabel private
  - Frekuensi sharing berkurang, sehingga mungkin meningkatkan kecepatan
  - Masih terdapat peluang kondisi race saat update s shared. Race dapat diatasi dengan menambahkan **lock** (mutex)
    - Hanya satu thread yang bisa memegang **lock**, lainnya menunggu

Model Thread di Shared Memory



Pemrograman Paralel dengan Thread

Pustaka Thread (Sistem)

1. Pthreads: POSIX Standar
   - Lebih low level
   - Portabel tapi mungkin lebih lambat
2. OpenMP: standar untuk programming level aplikasi
   - Mendukung pemrograman saintifik di memori shared
   - http://www.openMP.org
3. Java Threads
   - Dibangun di atas thread POSIX
   - Object di bahasa Java

Implementasi Model Thread

- Dari perspektif programmer, implementasi thread dapat dilakukan dari:
  1. **Pustaka** dari subrutin yang dipanggil dari dalam source code paralel
  2. Set **compiler directive** yang di-*embed*-kan di source code (serial atau paralel)
- Implementasi thread: POSIX Threads dan OpenMP
  - Posix (pthread) menggunakan pustaka, memerlukan pengkodean paralel
    - Paralelisme eksplisit, memerlukan programmer yang mahir di paralel
  - OpenMP, menggunakan compiler directive; dapat menggunakan kode serial

POSIX Threads

- POSIX: Portable Operating System Interface for UNIX
  - IEEE Std 1003.1c-1995 mendefinisikan API untuk membuat dan memanipulasi thread
  - System call untuk membuat dan mensinkronkan thread
  - Implementasi API: FreeBSD, NetBSD, GNU/Linux, Mac OS X dan Solaris
  - Windows? pthreads-w32 (third party)
- Support PThread
  - Membuat program paralel
  - Sinkronisasi
  - Memori shared implisit, pointer ke data shared diberikan ke suatu thread

Fungsi Threads: Creating Thread

*int pthread_create(pthread_t * thread,*
*const pthread_attr_t * attr,*
*void * (*start_routine)(void *),*
*void *arg);*

- **Contoh:**

```
err = pthread_create(&thread_id,
    &thread_attr,
    PrintHello,
    &thread_arg);
```

- Lihat program sample!

Manajemen Thread

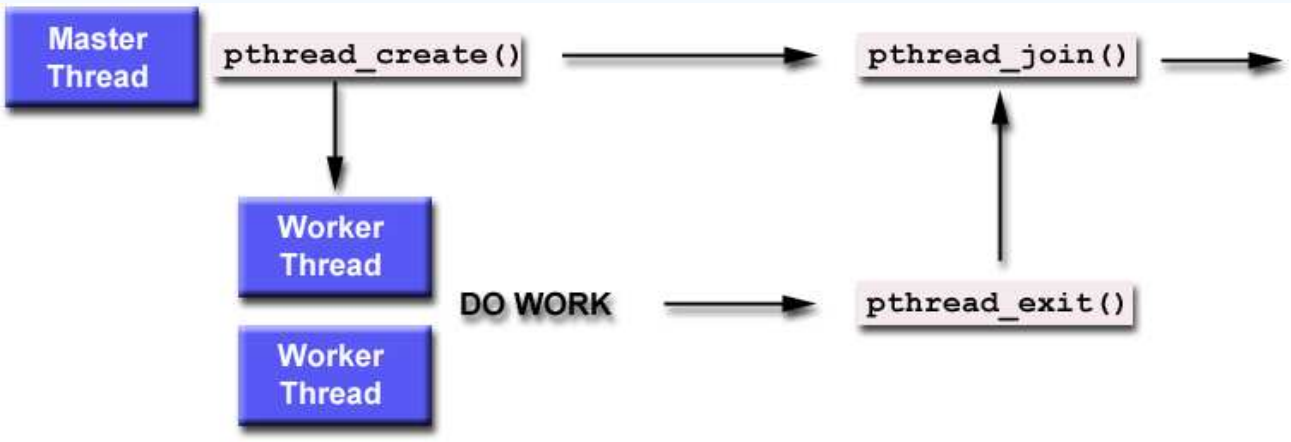
Joining and Detaching Threads

pthread_join (threadid,status)

pthread_detach (threadid)

pthread_attr_setdetachstate (attr,detachstate)

pthread_attr_getdetachstate (attr,detachstate)



- Joining adalah salah satu cara untuk sinkronisasi antar thread
  - Metode lainnya: mutex

Manajemen Thread: Mutex

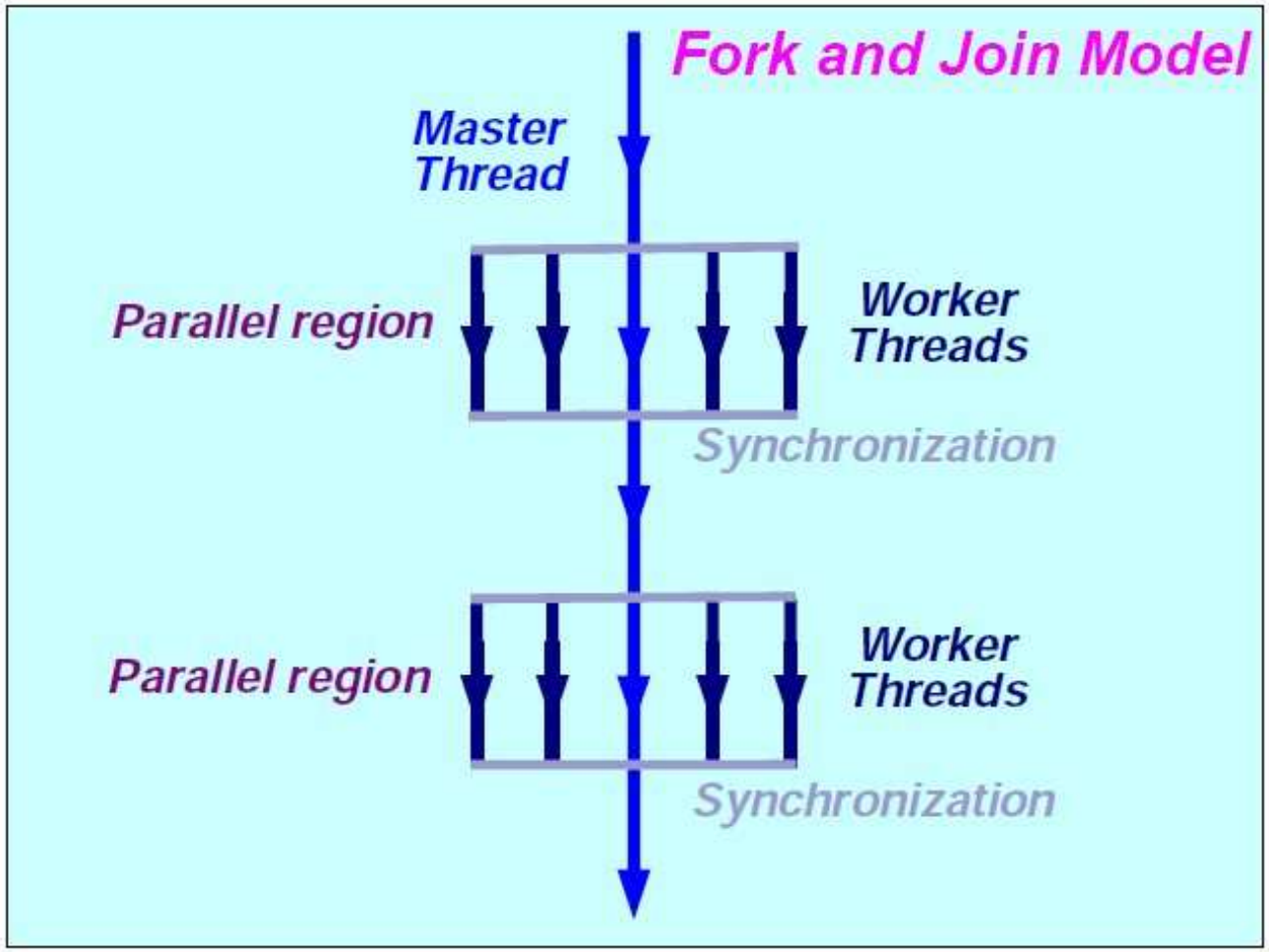
- Code!

Pemrograman Paralel dengan OpenMP

Tentang OpenMP

- OpenMP
  - Spesifikasi open untuk Multi-Processing
  - API standard untuk mendefinisikan program multi-threaded shared-memory
  - Resources: http://openmp.org/wp/resources/ - talks, examples, forum
  - Baca:
    - Ruud van des Pas: An Overview of OpenMP
    - materi selanjutnya diambil dari slide Pas
  - Tugas #1 diambil dari program example di openmp.org
- High-level API
  - Preprocessor (compiler) directive (~80%)
  - Library calls (~19%)
  - Variabel environment (~1%)

Model Eksekusi OpenMP



Model Sharing Data OpenMP

- Program paralel seringkali mempunyai 2 tipe data:
  1. Data shared, dapat diakses oleh semua thread, mempunyai nama yang sama
  2. Private data, hanya dapat diakses oleh satu thread (biasanya dialokasikan di stack)
- Pthreads:
  - Variabel dengan scope global adalah shared (data, bss, global heap)
  - Variabel dengan alokasi stack adalah private
- OpenMP:
  - Variabel shared adalah shared. Dinyatakan dengan *shared(list)*
  - Variabel private adalah private. Dinyatakan dengan *private(list)*

Model Sharing Data OpenMP

Pthreads:

```
// shared, globals
   int bigdata[1024];
   void* foo(void* bar) {
// private, stack
   int tid;
   /* Calculation goes here */
}
```

OpenMP:

```
int bigdata[1024];
void* foo(void* bar)
{
   int tid;
#pragma omp parallel shared
(bigdata) private (tid) {
   /* Calc. here */ }
}
```

Contoh OpenMP

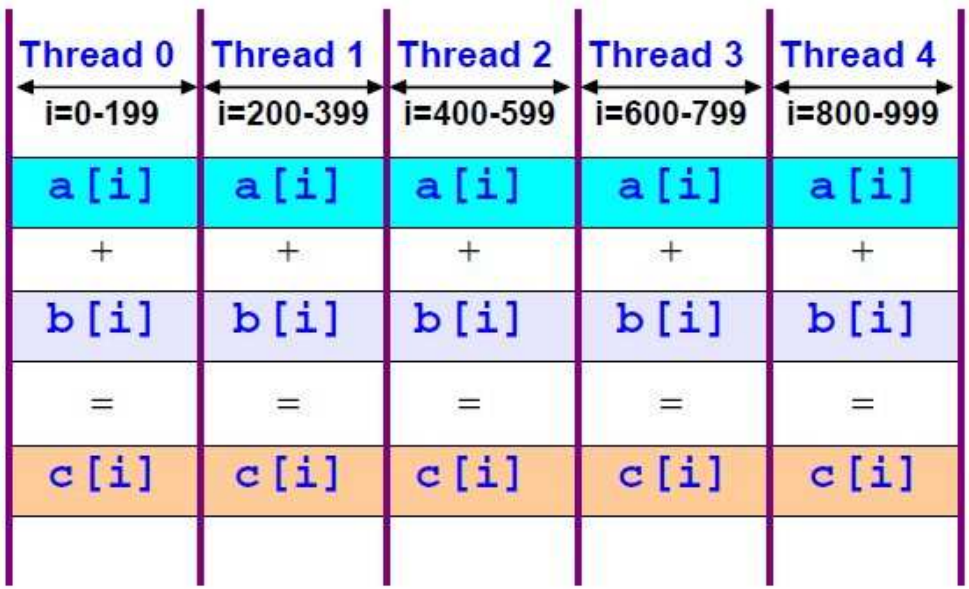
For-loop with independent iterations

```
for (int i=0; i<n; i++)
  c[i] = a[i] + b[i];
```

For-loop parallelized using an OpenMP pragma

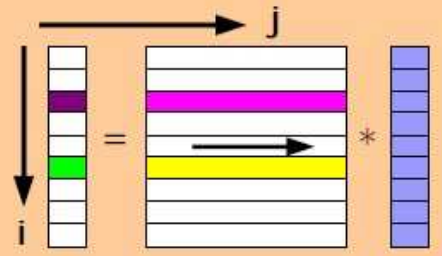
```
#pragma omp parallel for
for (int i=0; i<n; i++)
    c[i] = a[i] + b[i];
```

```
$ cc -xopenmp source.c
$ export OMP_NUM_THREADS=5
$ ./a.out
```



Contoh Perkalian Matriks dan Vektor


```
#pragma omp parallel for default(none) \
            private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
   sum = 0.0;
   for (j=0; j<n; j++)
     sum += b[i][j]*c[j];
   a[i] = sum;
 }
```



TID = 0

```
for (i=0,1,2,3,4)
 i = 0
sum =    b[i=0][j]*c[j]
   a[0] = sum
 i = 1
sum =    b[i=1][j]*c[j]
   a[1] = sum
```

TID = 1

```
for (i=5,6,7,8,9)
 i = 5
sum =    b[i=5][j]*c[j]
   a[5] = sum
 i = 6
sum =    b[i=6][j]*c[j]
   a[6] = sum
```

Baca Hands-on Introduction to OpenMP

- Tim Mattson dan Larry Meadows, Intel Corporation
- Tutorial dan Example