Pemrograman OpenMP (3)

Kuliah#9 TSK617 Pengolahan Paralel - TA 2011/2012

Eko Didik Widianto

Teknik Sistem Komputer - Universitas Diponegoro

Tentang Kuliah #9 Pemrograman OpenMP

- Akan dibahas tentang **pemrograman paralel dengan OpenMP** menggunakan kompiler directive
  - Arsitektur memori: shared (SMP, symmetric multi-processor)
  - Model programming: thread
- Pokok Bahasan: (kuliah #8 akan membahas item yang ditebalkan)
  1. Pengantar OpenMP
  2. Membuat Thread
  3. Sinkronisasi dengan *critical, atomic*
  4. Loop dan Worksharing
  5. Sinkronisasi dengan *barrier, single, master, ordered*
  6. Environment Data
  7. **Menjadwalkan *for* dan *section***
  8. **Task di OpenMP 3.0**

Kompetensi Dasar

- Setelah mempelajari bab ini, mahasiswa akan mampu:
  1. [C2] Mahasiswa memahami konsep pemrograman paralel menggunakan OpenMP
  2. [C3] Mahasiswa akan mampu membuat program paralel dari program serial menggunakan compiler-directive dan pustaka-pustaka OpenMP
  3. [C5] Mahasiswa akan mampu memprogram suatu aplikasi komputasi matrik menggunakan OpenMP serta menghitung faktor speedupnya
- Link
  - Website: http://didik.blog.undip.ac.id/2012/02/25/kuliah-tsk-617-pengolahan-paralel-2011/
  - Email: didik@undip.ac.id

Acknowledment

- Materi dan gambar didapat dari:
  - Tim Mattson, Larry Meadows (2008): “A Hands-on Introduction to OpenMP“
  - Website: http://openmp.org/wp/resources/
  - OmpSCR: OpenMP Source Code Repository (http://sourceforge.net/projects/ompscr/)

Bahasan

Bahasan

Sections Worksharing Construct

- The Sections **worksharing** construct gives **a different structured block** to each thread

```
#pragma omp parallel
{
#pragma omp sections
  {
#pragma omp section
    X_calculation();
#pragma omp section
    Y_calculation();
#pragma omp section
    Z_calculation();
  }
}
```

- By default, **there is a barrier** at the end of the “omp sections”. Use the “**nowait**” clause to **turn off** the barrier

Sections Example

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp sections nowait
    {
      #pragma omp section
        for (i=0; i<n-1; i++)
            b[i] = (a[i] + a[i+1])/2;

      #pragma omp section
        for (i=0; i<n; i++)
            d[i] = 1.0/c[i];

    } /*-- End of sections --*/

  } /*-- End of parallel region --*/
```

Bahasan

Schedule Clause in Loop Worksharing for

- The schedule clause affects how **loop iterations are mapped onto threads**
  - **schedule(static [,chunk])**
    Deal-out blocks of iterations of size “chunk” to each thread.
  - **schedule(dynamic[,chunk])**
    Each thread grabs “chunk” iterations off a queue until all iterations have been handled.
  - **schedule(guided[,chunk])**
    Threads dynamically grab blocks of iterations. The size of the block starts large and shrinks down to size “chunk” as the calculation proceeds.
  - **schedule(runtime)**
    Schedule and chunk size taken from the OMP_SCHEDULE environment variable (or the runtime library)
    - for OpenMP 3.0

Static, Dynamic and Guided Schedule Clauses

Schedule Clause	When To Use	Notes
STATIC	Pre-determined and predictable by the programmer	**Least work** at runtime: scheduling done at compile-time
DYNAMIC	Unpredictable, highly variable work per iteration	**Most work** at runtime: complex scheduling logic used at run-time
GUIDED	Special case of dynamic to reduce scheduling overhead	

Contoh: Linked List

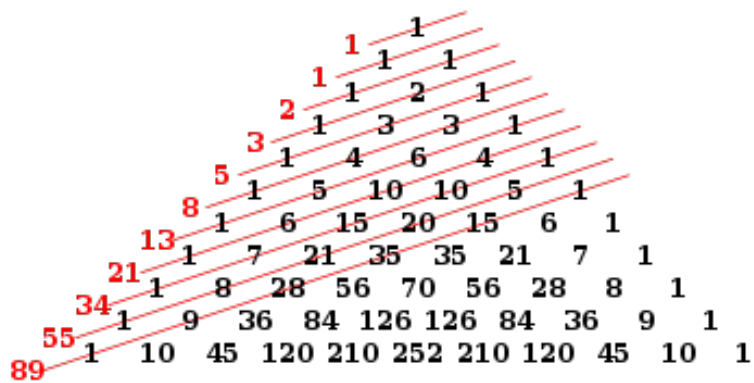
- Program Serial: linked.c
  - Menampilkan urutan bilangan fibonacci:

F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	F_0	...
0	1	1	2	3	5	8	13	21	34	55	89	144	...

$F_n = F_{n-1} + F_{n-2}$ dengan nilai seed: $F_0 = 0$, $F_1 = 1$

  - menggunakan linked list

```
int fib(int n) {
   int x, y;
   if (n < 2) {
      return (n);
   } else {
      x = fib(n - 1);
      y = fib(n - 2);
      return (x + y);
   }
}
```



Pascal Triangle[a]

[a] Sumber: www.wikipedia.org

Linked List (cont)

- Buat program paralelnya (linked_omp25.c)
  - Ubah *loop* ***while*** menjadi ***for***
    - jumlah iterasi dihitung terlebih dahulu

```
p = init_list(p);
head = p;

while (p != NULL) {
   processwork(p);
   p = p->next;
}
```

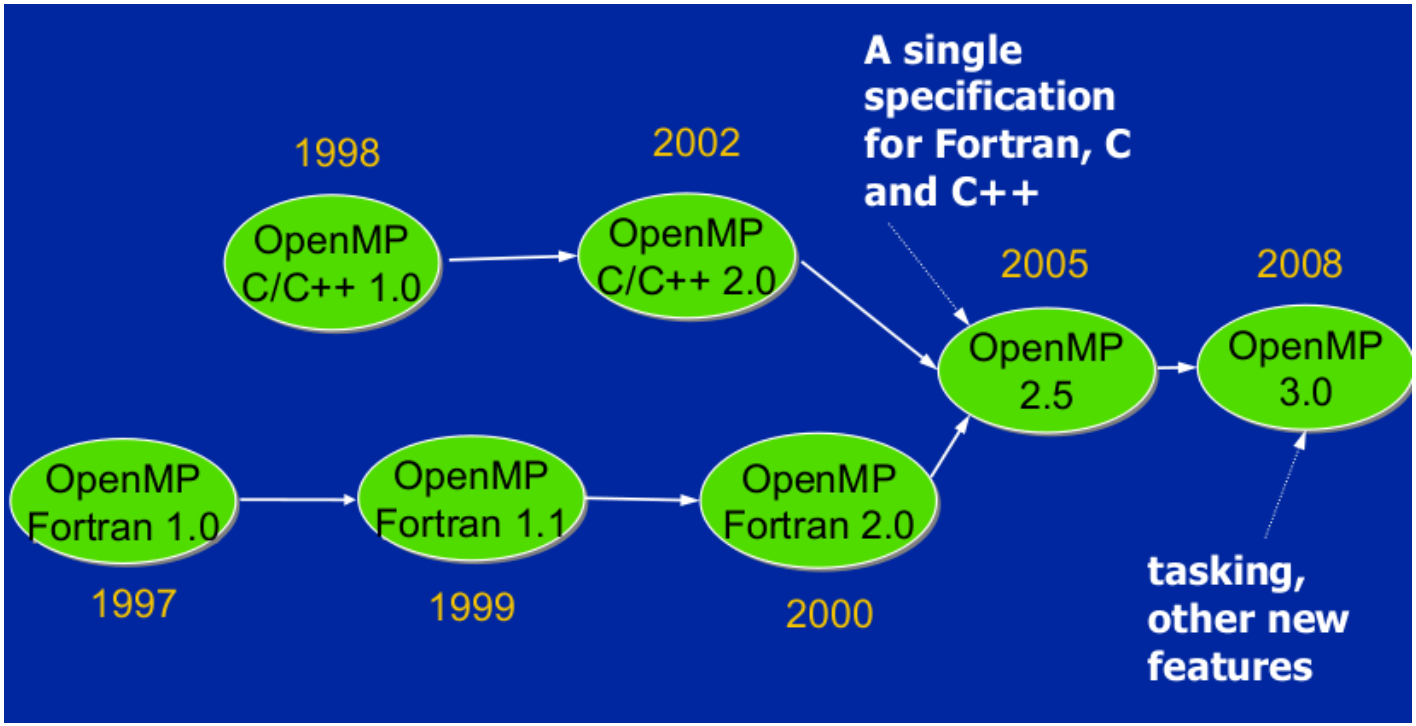
```
p = init_list(p);
head = p;
while (p != NULL) {
   p = p->next;
   count++;
}
p = head;
for(i=0; i<count; i++) {
   parr[i] = p;
   p = p->next;
}
#pragma omp parallel
{
   #pragma omp single
  printf(" %d threads \n",omp_get_num_threads());
   #pragma omp for
   for(i=0; i<count; i++)
      processwork(parr[i]);
}
```

Latihan: Perkalian Matrik

- Program serial perkalian matriks: matmul.c
- Buat program paralelnya
  - Optimasi program dengan menjadwalkan loop

Bahasan

OpenMP Release History



- 2011: Release OpenMP 3.1 (http://www.openmp.org/mp-documents/OpenMP3.1.pdf)

Bahasan

OpenMP Tasks

- Tasking is added since OpenMP 3.0
- A task **has**
  - **Code to execute**
  - A data environment (it **owns** its data)
  - **An assigned thread** that executes the code and uses the data
- Two activities: **packaging** and **execution**
  - Each encountering thread **packages** a new instance of a task (code and data)
  - Some thread in the team **executes** the task

Definitions

- **Task construct**
  - task directive plus structured block
- **Task**
  - **the package** of code and instructions for allocating data
  - **created** when a thread encounters a task construct
- **Task region**
  - the dynamic sequence of instructions **produced b**y the execution of a task by a thread

Tasks and OpenMP

- Tasks have been fully integrated into OpenMP
- Key concept: OpenMP **has always** had tasks, we just never called them that.
  - Thread encountering parallel construct packages up a set of implicit tasks, one per thread
  - Team of threads is created
  - Each thread in team is assigned to one of the tasks (and tied to it)
  - Barrier holds original master thread until all implicit tasks are finished
- We have simply added a way to create **a task explicitly** for the team to execute.

Bahasan

Task Construct

```
#pragma omp task [clause[[,]clause] ...]
    structured-block
```

- where clause can be one of:

  `if (expression)`
  - When the if clause expression evaluates to false, an undeferred task is generated, and the encountering thread must suspend the current task region, for which execution cannot be resumed until the generated task is completed

  `untied`
  - If present, any thread in the team can resume the task region after a suspension

  `shared (list)`

  `private (list)`

  `firstprivate (list)`

  `default( shared | none )`

When/where are tasks complete?

- At thread barriers, explicit or implicit
  - applies to all tasks generated in the current parallel region up to the barrier
  - matches user expectation
- At task barriers
  - i.e. Wait until all tasks defined in the current task have completed.

```
#pragma omp taskwait
```

Bahasan

Task Example

Parallel Pointer Chasing using Tasks

```
#pragma omp parallel
{
  #pragma omp single private(p)
  {
    p = listhead ;

    while (p) {
      #pragma omp task  // p is firstprivate inside  this task
        process (p)
      p=next (p) ;
    }
  }
}
```

Task Example

Parallel Pointer Chasing on Multiple Lists using Tasks

```
#pragma omp parallel
{
  #pragma omp for private(p)
  for ( int i =0; i <numlists ; i++) {
    p = listheads [ i ] ;
    while (p ) {
      #pragma omp task
      process (p)
      p=next (p ) ;
    }
  }
}
```

Bahasan

Task Switching

- When a thread encounters a **task scheduling point**, it is **allowed to suspend** the current task and execute another (called **task switching**)
  - **Generating task** will have to suspend for a while
- It can then return to the original task and resume

```
#pragma omp single
{
   for (i=0; i<ONEZILLION; i++)
   #pragma omp task
   process(item[i]);
}
```

- With task switching, the executing thread can:
  - execute an already generated task (draining the “task pool”)
  - dive into the encountered task (could be very **cache-friendly**)

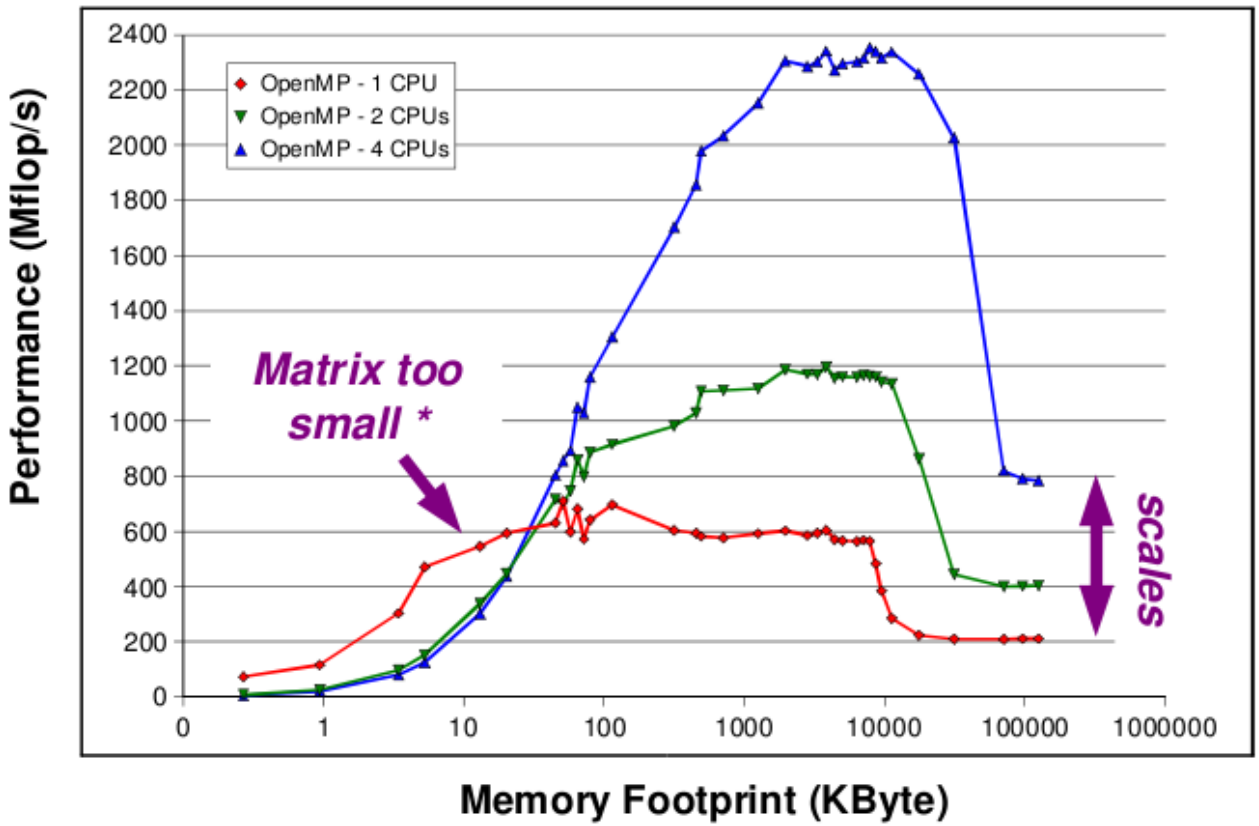
Thread Switching

```
#pragma omp single
{
   #pragma omp task untied
      for (i=0; i<ONEZILLION; i++)
        #pragma omp task
        process(item[i]);
}
```

- Generating task is suspended and executing thread switches to a long task
- With thread switching, the generating task can be resumed by a **different thread**

Bahasan

Profiling



- Bagaimana melakukan profiling kode paralel?

Inserting Profiling Code

- Use OpenMP function **omp_get_wtime()** to get time stamp

```
#include <omp.h>
#define ORDER 1000
double start, end;
....
start = omp_get_wtime();
<paralel code>
end = omp_get_wtime();
printf("Compute Time: %f seconds\n", end - start);
dN = (double)ORDER;
mflops = 2.0 * dN * dN * dN/(1000000.0* run_time);
printf(" Order %d multiplication at %f mflops\n", OR-
DER, mflops);
...
```

Bahasan

OpenMP Resources

- OmpSCR: OpenMP Source Code Repository (http://sourceforge.net/projects/ompscr/)

OpenMP On the Real World

- Christian Terboven, Dieter an Mey, “OpenMP in the Real World”

Lisensi

Creative Common Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

- Anda bebas:
  - untuk **Membagikan** — untuk menyalin, mendistribusikan, dan menyebarkan karya, dan
  - untuk **Remix** — untuk mengadaptasikan karya
- Di bawah persyaratan berikut:
  - **Atribusi** — Anda harus memberikan atribusi karya sesuai dengan cara-cara yang diminta oleh pembuat karya tersebut atau pihak yang mengeluarkan lisensi.
  - **Pembagian Serupa** — Jika Anda mengubah, menambah, atau membuat karya lain menggunakan karya ini, Anda hanya boleh menyebarkan karya tersebut hanya dengan lisensi yang sama, serupa, atau kompatibel.
- Lihat: Creative Commons Attribution-ShareAlike 3.0 Unported License