IF3230
Sistem Paralel dan Terdistribusi
Shared Address Space Model – OpenMP
Additional topics

Achmad Imam Kistijantoro (imam@informatika.org)

Februari 2021

Informatika – STEI – ITB

Standar OpenMP

Version 3.0 (2008)	Introduction of task parallelism, and improvements to loop parallelism. These improvements to loop parallelism include loop collapse and nested parallelism.
Version 3.1 (2011)	Adds reduction `min` and `max` operators to C and C++ (other operators already in C and C++; `min` and `max` already in Fortran) and thread binding control
Version 4.0 (2013)	Adds OpenMP SIMD (vectorization) directive, target directive for offloading to GPUs and other accelerator devices, and thread affinity control
Version 4.5 (2015)	Substantial improvements to accelerator device support for GPUs
Version 5.0 (2018)	Further improvements to accelerator device support

Pragma: single

- Pragma `single` digunakan di dalam blok parallel untuk memberitahu kompiler bahwa hanya 1 thread yang akan menjalankan blok/statement yg mengikuti pragma ini

- Pragma `master` (`masked`) bermakna sama, namun lebih efisien, dan tidak ada implisit barrier

Clause: nowait

- Klausa `nowait` digunakan untuk memberitahu kompiler bahwa tidak ada barrier synchronization di akhir parallel `for` loop atau block `single`

Case: parallel, for, single Pragmas

```
for (i = 0; i < N; i++)
   a[i] = alpha(i);
if (delta < 0.0) printf (“delta < 0.0\n”);
for (i = 0; i < N; i++)
   b[i] = beta (i, delta);
```

Case: parallel, for, single Pragmas

```
#pragma omp parallel for
for (i = 0; i < N; i++)
   a[i] = alpha(i);
if (delta < 0.0) printf (“delta < 0.0\n”);
#pragma omp parallel for
for (i = 0; i < N; i++)
   b[i] = beta (i, delta);
```

Solution: parallel, for, single Pragma

```
#pragma omp parallel
{

   #pragma omp for
   for (i = 0; i < N; i++)
      a[i] = alpha(i);
   #pragma omp single nowait
   if (delta < 0.0)
     printf (“delta < 0.0\n”);
   #pragma omp for
   for (i = 0; i < N; i++)
      b[i] = beta (i, delta);
}
```

```
#include <math.h>
void nowait_example2(int n, float *a, float *b, float *c, float
*y, float *z)
{
  int i;
#pragma omp parallel
  {
#pragma omp for schedule(static) nowait
  for (i=0; i<n; i++)
    c[i] = (a[i] + b[i]) / 2.0f;
#pragma omp for schedule(static) nowait
  for (i=0; i<n; i++)
    z[i] = sqrtf(c[i]);
#pragma omp for schedule(static) nowait
  for (i=1; i<=n; i++)
    y[i] = z[i-1] + a[i];
  }
}
```

Contoh lain

```
#pragma omp parallel
{
#pragma omp for schedule(static) nowait
  for(i=0;i<N;i++){
    a[i] = ....
  }
#pragma omp for schedule(static)
  for(i=0;i<N;i++){
    ... = a[i]
  }
}
```

Aman sepanjang jumlah iterasi dan schedule sama di kedua loop

Extended Example

```
for (i = 0; i < m; i++) {
   low = a[i];
   high = b[i];
   if (low > high) {
      printf ("Exiting during iteration %d\n", i);
      break;
   }
   #pragma omp parallel for
   for (j = low; j < high; j++)
      c[j] += alpha (i, j);
}
```

Extended Example

```
#pragma omp parallel private (i, j, low, high)
for (i = 0; i < m; i++) {
   low = a[i];
   high = b[i];
   if (low > high) {
      printf ("Exiting during iteration %d\n", i);
      break;
   }
   #pragma omp for nowait
   for (j = low; j < high; j++)
      c[j] += alpha (i, j);
}
```

Extended Example

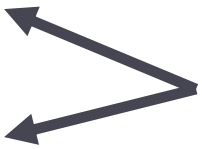
```
#pragma omp parallel private (i, j, low, high)
for (i = 0; i < m; i++) {
   low = a[i];
   high = b[i];
   if (low > high) {
      #pragma omp single nowait
      printf ("Exiting during iteration %d\n", i);
      break;

   }
   #pragma omp for nowait
   for (j = low; j < high; j++)
      c[j] += alpha (i, j);
}
```

Parallel Sections

digunakan untuk mendefinisikan blok yang dapat dieksekusi paralel

```
#pragma omp parallel sections
{
<code block A>
#pragma omp section
<code block B>
#pragma omp section
<code block C>
}
```

setiap blok dijalankan oleh 1 thread



pemisah antar blok

Contoh Model pipeline

```
#pragma omp parallel sections
{
   #pragma omp section
   {
     for (int i=0; i<N; i++) {
        (void) read_input(i);
        (void) signal_read(i);
     }
   }
   #pragma omp section
   {
     for (int i=0; i<N; i++) {
        (void) wait_read(i);
        (void) process_data(i);
        (void) signal_processed(i);
     }
   }
   #pragma omp section
   {
     for (int i=0; i<N; i++) {
        (void) wait_processed(i);
        (void) write_output(i);
     }
   }
} /*-- End of parallel sections --*/
```

Input Thread

Processing Thread(s)

Output Thread

Task

- task adalah unit kerja yang independen, terdiri atas:
  - kode yang akan dieksekusi
  - data environment
  - internal control variable
- thread adalah entitas aktif yang mengerjakan setiap task
  - eksekusi task dapat ditunda (deferred), atau langsung dieksekusi

Task

- task construct: sebuah blok yang diawali dengan direktif task dan diikuti blok

- eksekusi task dipastikan akan selesai saat melalui thread barrier atau task barrier

```
int fib ( int n ) {
  int x,y;
  if ( n < 2 ) return n;
  #pragma omp task
  x = fib(n-1);
  #pragma omp task
  y = fib(n-2);
  #pragma omp taskwait
  return x+y
}
```

```
int main() {
  #omp parallel num_thread(3)
    #omp single nowait
    int result = fib(10);
}


int fib ( int n ) {
  int x,y;
  if ( n < 2 ) return n;
  #pragma omp task
  x = fib(n-1);
  #pragma omp task
  y = fib(n-2);
  #pragma omp taskwait
  return x+y
}
```

```
struct node {
  struct node *left;
  struct node *right;
};
extern void process(struct node *);
void traverse( struct node *p ) {
  if (p->left)
#pragma omp task
    traverse(p->left);
  if (p->right)
#pragma omp task
    traverse(p->right);
  process(p);
}
```

```
struct node {
  struct node *left;
  struct node *right;
};
extern void process(struct node *);
void postorder_traverse( struct node *p ) {
  if (p->left)
#pragma omp task
    postorder_traverse(p->left);
  if (p->right)
#pragma omp task
    postorder_traverse(p->right);
#pragma omp taskwait
  process(p);
}
```

```
typedef struct node node;
struct node {
  int data;
  node * next;
};
void process(node * p){
/* do work here */
}
void increment_list_items(node * head) {
  #pragma omp parallel
  {
    #pragma omp single
    {
      node * p = head;
      while (p) {
#pragma omp task
        process(p);
        p = p->next;
      }
    }
  }
```

Contoh kasus

```
#pragma omp parallel for private(temp)
for(i=0;i<N;i++){
  for (j=0;j<M;j++){
    temp = b[i]*c[j];
    a[i][j] = temp * temp + d[i];
  }
}
```