COMS3008A: Parallel Computing
Introduction to OpenMP: Part III

Hairong Wang

School of Computer Science,
University of the Witwatersrand, Johannesburg

April 9, 2025

WITS UNIVERSITY

Contents

Outline

Outline

Sections/Section Construct

- `sections` directive enables specification of **task parallelism**
- The `sections` worksharing construct gives a different structured block to each thread.
- Syntax:

```
#pragma omp sections [clause[[,] clause]...]
{
  [#pragma omp section]
    structured block
  [#pragma omp section]
  structured block
  ...
}
```

- clauses: `private`, `firstprivate`, `lastprivate`, `reduction`, `nowait`
- Each section must be a structured block of code that is independent of other sections.
- There is an implicit barrier at the end of a sections construct unless a `nowait` clause is specified.

Examples — firstprivate

```
1  #include <omp.h>
2  #include <stdio.h>
3  #define NT 4
4  int main( ) {
5    int section_count = 0;
6    omp_set_dynamic(0);
7    omp_set_num_threads(NT);
8    #pragma omp parallel
9    #pragma omp sections firstprivate( section_count )
10   {
11     #pragma omp section
12     {
13       section_count++;
14       printf( "section_count %d\n", section_count );
15     }
16     #pragma omp section
17     {
18       section_count++;
19       printf( "section_count %d\n", section_count );
20     }
21   }
22   return 0;
23 }
```

Example – Parallel quicksort

Example 1

Parallelize the sequential *quicksort* program (`qsort_serial.c`) using OpenMP `sections/section` construct.

```
1   q_sort(left, right, data) {
2     if(left < right){
3       q = partition(left, right, data);
4       q_sort(left, q-1, data);
5       q_sort(q+1, right, data);
6     }
7   }
8   partition(left, right, float *data) {
9     x = data[right];
10    i = left-1;
11    for(j=left; j<right; j++) {
12      if(data[j] <= x){
13        i++;
14        swap(data, i, j);
15      }
16    }
17    swap(data, i+1, right);
18    return i+1;
19  }
```

Outline

Task worksharing construct

- Tasks are independent units of work.
- Threads are assigned to perform the work of each task.
  - Tasks may be deferred
  - Tasks may be executed immediately
- The runtime system decides which of the above
- A *task* is composed of:
  - Code to execute
  - A data environment
  - Internal control variables, such as
    - `OMP_NESTED` – `TRUE` or `FALSE`, controls whether nested parallelism is enabled.
    - `omp_set_dynamic()`, `omp_get_dynamic` – controls the dynamic adjustment of threads.
    - `omp_set_num_threads()`, etc.

Task construct cont.

Task Construct Syntax

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

where clause can be

- if(*expr*): if *expr*=FALSE, then the task is immediately executed.
- shared
- private
- firstprivate
- default(shared|none)
- tied/untied
- final(expr)

Task construct cont.

- Variables that are shared in all scopes enclosing the task construct remain shared in the generated task. All other variables are implicitly determined firstprivate.
- `tied/untied`: Upon resuming a suspended task region, a `tied` task must be executed by the same thread again. With an `untied` task, there is no such restriction and any thread in the team can resume execution of the suspended task.
- **if** `(expr)` clause - If `expr` is evaluated to false, the task is undeferred and executed immediately by the thread that was creating the task.
- `final(expr)` clause - For recursive and nested applications, it stops task generations at a certain depth where we have enough tasks (or parallelism). If `expr` is evaluated to true, the task is considered to be final and no additional tasks are generated.

Task construct cont.

- Two activities: *packaging* and *execution*
  - Each encountering thread packages a new instance of task
  - Some thread in the team executes the task at some time later or immediately.
- Task barrier: The **taskwait** directive:

Task construct example

Example 2

```
1  ......
2  #pragma omp parallel
3  {
4    #pragma omp single private(p)
5    {
6      p=list_head;
7      while(p){
8        #pragma omp task
9          processwork(p);
10       p=p->next;
11     }
12   }
13 }
```

Task construct cont.

When tasks are guaranteed to be completed?

- At thread or task barriers
- At the directive: `#pragma omp barrier`
- At the directive: `#pragma omp taskwait`

Example 3

```
1  #pragma omp parallel
2  {
3    #pragma omp task
4      foo();
5    #pragma omp barrier
6    #pragma omp single
7    {
8      #pragma omp task
9        bar();
10   }
11 }
```

Task construct example

Example 4

Understanding Task Construct

```
1  int main(int argc, char *argv[]){
2    #pragma omp parallel num_threads(2)
3    {
4      printf("A ");
5      printf("soccer ");
6      printf("match ");
7    }
8    printf("\n");
9    return 0;
10 }
```

Task construct example

Example 5

Understanding Task Construct

```
1  int main(int argc, char *argv[]){
2    #pragma omp parallel
3    {
4      #pragma omp single
5      {
6        printf("A ");
7        printf("soccer ");
8        printf("match ");
9      }
10   }
11   printf("\n");
12   return 0;
13 }
```

Task construct example

Example 6

Understanding Task Construct

```
1  int main(int argc, char *argv[]){
2    #pragma omp parallel
3    {
4      #pragma omp single
5      {
6        printf("A ");
7        #pragma omp task
8        printf("soccer ");
9        #pragma omp task
10       printf("match ");
11     }
12   }
13   printf("\n");
14   return 0;
15 }
```

Task construct example

Example 7

Understanding Task Construct

```
1  int main(int argc, char *argv[]){
2    #pragma omp parallel
3    {
4      #pragma omp single
5      {
6        printf("A ");
7        #pragma omp task
8        printf("soccer ");
9        #pragma omp task
10       printf("match ");
11       printf("is fun to watch ");
12     }
13   }
14   printf("\n");
15   return 0;
16 }
```

Task construct example

Example 8

Understanding Task Construct

```
1  int main(int argc, char *argv[]){
2    #pragma omp parallel
3    {
4      #pragma omp single
5      {
6        printf("A ");
7        #pragma omp task
8        printf("soccer ");
9        #pragma omp task
10       printf("match ");
11       #pragma omp taskwait
12       printf("is fun to watch ");
13     }
14   }
15   printf("\n");
16   return 0;
17 }
```

Task construct example

Example 9

Tree traversal using task

```
2  void traverse(node *p){
3    if(p->left)
4      #pragma omp task
5        traverse(p->left);
6    if(p->right)
7      #pragma omp task
8        traverse(p->right);
9    process(p->data);
10 }
```

Task construct example

Example 10

Tree traversal using task

```
2  void traverse(node *p){
3    if (p->left)
4      #pragma omp task
5        traverse(p->left)
6    if (p->right)
7      #pragma omp task
8        traverse(p->right)
9    #pragma omp taskwait
10   process(p->data);
11 }
```

Task construct example

Example 11

Write an OpenMP parallel program for computing the *n*th Fibonacci number. Compare the performance of the parallel implementation to the sequential one.

Task construct cont.

Task switching: *untied*:

```
1  #define ONEBILLION 1000000000L
2  #pragma omp parallel
3  {
4    #pragma omp single
5    {
6      for(i=0; i<ONEBILLION; i++)
7        #pragma omp task
8          process(item[i]);
9    }
10   ......
11   /* Untied task: any other thread is eligible to resume
12   the task generating loop*/
13   #pragma omp single
14   {
15     for(i=0; i<ONEBILLION; i++)
16       #pragma omp task untied
17         process(item[i]);
18   }
19 }
```

References

- Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), by Barbara Chapman, Gabriele Jost and Ruud van der Pas. The MIT Press, 2007.
- https://hpc-tutorials.llnl.gov/openmp/, accessed April 5.

WITS UNIVERSITY