COMS3008A: Parallel Computing
Introduction to OpenMP: Part II

Hairong Wang

School of Computer Science,
University of the Witwatersrand, Johannesburg

2021-9-16

Contents

1. Worksharing
    - Sections/Section Construct
    - Task worksharing construct

WITS UNIVERSITY

Outline

1. Worksharing
   - Sections/Section Construct
   - Task worksharing construct

Outline

1. Worksharing
   - Sections/Section Construct
   - Task worksharing construct

WITS UNIVERSITY

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.

WITS UNIVERSITY

Examples — firstprivate

```c
#include <omp.h>
#include <stdio.h>
#define NT 4
int main( ) {
  int section_count = 0;
  omp_set_dynamic(0);
  omp_set_num_threads(NT);
  #pragma omp parallel
  #pragma omp sections firstprivate( section_count )
  {
    #pragma omp section
    {
      section_count++;
      printf( "section_count %d\n", section_count );
    }
    #pragma omp section
    {
      section_count++;
      printf( "section_count %d\n", section_count );
    }
  }
  return 0;
}
```

WITS UNIVERSITY

Example – Parallel quicksort

Example 1

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

```
    q_sort(left, right, data) {
      if(left < right){
        q = partition(left, right, data);
        q_sort(left, q-1, data);
        q_sort(q+1, right, data);
      }
    }
    partition(left, right, float *data) {
      x = data[right];
      i = left-1;
      for(j=left; j<right; j++) {
        if(data[j] <= x){
          i++;
          swap(data, i, j);
        }
      }
      swap(data, i+1, right);
      return i+1;
    }
```

Outline

1. Worksharing
   - Sections/Section Construct
   - Task worksharing construct

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.

WITS UNIVERSITY

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)
- untied
- final(expr)

Task construct cont.

- `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).

WITS UNIVERSITY

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:

WITS UNIVERSITY

Task construct example

Example 2

```
......
#pragma omp parallel
{
  #pragma omp single private(p)
  {
    p=list_head;
    while(p){
      #pragma omp task
        processwork(p);
      p=p->next;
    }
  }
}
```

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

```
#pragma omp parallel
{
  #pragma omp task
    foo();
  #pragma omp barrier
  #pragma omp single
  {
    #pragma omp task
      bar();
  }
}
```

Task construct example

Example 4

Understanding Task Construct

```
int main(int argc, char *argv[]){
  #pragma omp parallel num_threads(2)
  {
    printf("A ");
    printf("soccer ");
    printf("match ");
  }
  printf("\n");
  return 0;
}
```

Task construct example

Example 5

Understanding Task Construct

```c
int main(int argc, char *argv[]){
  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("A ");
      printf("soccer ");
      printf("match ");
    }
  }
  printf("\n");
  return 0;
}
```

Task construct example

Example 6

Understanding Task Construct

```c
int main(int argc, char *argv[]){
  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
      printf("soccer ");
      #pragma omp task
      printf("match ");
    }
  }
  printf("\n");
  return 0;
}
```

WITS UNIVERSITY

Task construct example

Example 7

Understanding Task Construct

```
int main(int argc, char *argv[]){
  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
      printf("soccer ");
      #pragma omp task
      printf("match ");
      printf("is fun to watch ");
    }
  }
  printf("\n");
  return 0;
}
```

Task construct example

Example 8

Understanding Task Construct

```
int main(int argc, char *argv[]){
  #pragma omp parallel
  {
    #pragma omp single
    {
      printf("A ");
      #pragma omp task
      printf("soccer ");
      #pragma omp task
      printf("match ");
      #pragma omp taskwait
      printf("is fun to watch ");
    }
  }
  printf("\n");
  return 0;
}
```

Task construct example

Example 9

Tree traversal using task

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

WITS UNIVERSITY

Task construct example

Example 10

Tree traversal using task

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

Task construct example

Example 11

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

Task construct cont.

Task switching: *untied*:

```
#define ONEBILLION 1000000000L
#pragma omp parallel
{
  #pragma omp single
  {
    for (i=0; i<ONEBILLION; i++)
      #pragma omp task
        process(item[i]);
  }
  ......
  /* Untied task: any other thread is eligible to resume
  the task generating loop*/
  #pragma omp single
  {
    #pragma omp task untied
      for (i=0; i<ONEBILLION; i++)
        #pragma omp task
          process(item[i]);
  }
}
```

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.
- Using OpenMP—The Next Step: Affinity, Accelerators, Tasking, and SIMD, by Ruud van der Pas, Eric Stozer, and Christian Terboven. The MIT Press, 2017.
- `https://hpc.llnl.gov/tuts/openMP/`