COMS4040A & COMS7045A: High Performance Computing & Scientific Data Management Introduction to OpenMP: Part II

Hairong Wang

School of Computer Science,
University of the Witwatersrand, Johannesburg

2019-2-28

WITS UNIVERSITY

Contents

Outline

Outline

A Review on OpenMP

The OpenMP constructs we've learned so far:

- To create a team of threads (or a parallel region):
- To share work between threads:
- To prevent race conditions (or conflicts):
- Data environment (or data scoping) clauses:

A Review on OpenMP Contd.

```
1  double compute_pi(double step){
2    double pi, x, sum=0.0;
3    omp_set_num_threads(NUM_THREADS);
4  #pragma omp parallel
5  {
6    #pragma omp for reduction(+:sum) private(x)
7    for (int i=1;i<= num_steps; i++){
8      x = (i-0.5)*step;
9      sum = sum + 4.0/(1.0+x*x);
10   }
11 }
12   pi = step * sum;
13   return pi;
14 }
```

WITS UNIVERSITY

A Review on OpenMP Contd.

```
1 #pragma omp for [clause[[,] clause]...]
2 {...}
```

- Possible clauses
  - `private`, `firstprivate`, `lastprivate`
  - `reduction`
  - `schedule`, `nowait`, `collapsed`, `ordered`
- Implicit barrier at the end of `for` loop
- `nowait`
  - modifies a `for` directive
  - avoids implicit barrier at end of `for`
- `ordered`
  - `for` loops with carried dependence
  - The enclosed block of code is executed in the order in which iterations would be executed sequentially.

The `collapse` Clause

- The iterations of the `k` and `j` loops are collapsed into one loop. and that loop is then divided among the threads in the current team.

```
1  void work(int a, int j, int k);
2  void main()
3  {
4     int j, k, a;
5     #pragma omp parallel num_threads(2)
6     {
7        #pragma omp for collapse(2) ordered private(j,k) schedule
             (static,3)
8        for (k=0; k<3; k++)
9           for (j=0; j<2; j++)
10          {
11             #pragma omp ordered
12             printf("%d %d %d\n", omp_get_thread_num(), k, j);
13             /* end ordered */
14             work(a,j,k);
15          }
16    }
17 }
```

The `collapse` Clause

- The iterations of the `k` and `j` loops are collapsed into one loop. and that loop is then divided among the threads in the current team.

```
1  void work(int a, int j, int k);
2  void main()
3  {
4    int i, j, k, a;
5    #pragma omp parallel num_threads(2)
6    {
7      #pragma omp for collapse(2) private(i,j,k)
8        for (k=0; k<3; k++)
9          for (j=0; j<2; j++)
10           for (i = 0; i<4; i++)
11           {
12             printf("%d %d %d\n", i, k, j);
13             work(a,j,k);
14           }
15
16   }
17 }
```

More on for construct

- OpenMP parallelizes `for` loops that are in canonical form.
- Loops in canonical form take one of the following forms.

```
                                  index++
                                  ++index
                       index < end   index--
                       index <= end  --index
for( index=start;      index >= end; index += incr   )
                       index > end   index -= incr
                                  index=index+incr
                                  index=incr+index
                                  index=index-incr
```

Outline

Sections/Section Construct

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

```
1  #pragma omp sections [clause[[,] clause]...]
2  {
3    [#pragma omp section]
4      structured block
5    [#pragma omp section
6      structured block]
7    ...
8  }
```

- 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

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 }
```

Examples — lastprivate clause

```
1  void lastpriv (int n, float *a, float *b)
2  {
3    int i;
4    ......
5    #pragma omp parallel
6    {
7      #pragma omp for lastprivate(i)
8      for (i=0; i<n-1; i++)
9        a[i] = b[i] + b[i+1];
10   }

12   a[i]=b[i];
13 }
```

Examples — nowait clause

```
1  #include <math.h>
2  void nowait_example2(int n, float *a, float *b, float *c, float
       *y, float *z)
3  {
4     int i;
5  #pragma omp parallel
6     {
7  #pragma omp for schedule(static) nowait
8     for (i=0; i<n; i++)
9        c[i] = (a[i] + b[i]) / 2.0f;
10 #pragma omp for schedule(static) nowait
11    for (i=0; i<n; i++)
12       z[i] = sqrtf(c[i]);
13 #pragma omp for schedule(static) nowait
14    for (i=1; i<=n; i++)
15       y[i] = z[i-1] + a[i];
16    }
17 }
```

Sections/Section Construct Contd

Example (1)

Parallelize the sequential *quicksort* program (`qsort_v00.c`) using OpenMP `sections` construct.

Quick Sort Algorithm

```
1  q_sort(left, right, data){
2    q = partition(left, right, data);
3    q_sort(left, q-1, data);
4    q_sort(q+1, right, data);
5  }
6  partition(left, right, data){
7    x = data[left];
8    p = left+1; r = right;
9    while(p < r){
10     while(data[r] > x)
11       r=r-1;
12     while(data[p] <= x && p < r)
13       p=p+1;
14     if(p <= r)
15       swap(data[p], data[r]);
16       p=p+1; r=r-1;
17   }
18   swap(data[left], data[r]);
19   return r;
20 }
```

Outline

Single Worksharing Construct

- The *single* construct denotes a block of code that is executed by only one thread.
- Syntax:

```
1  #pragma omp single [clause[[,] clause]...]
2      structured block
```

- clauses: `private`, `firstprivate`, `copyprivate`, `nowait`
- A barrier is implied at the end of the *single* block, unless a `nowait` clause is specified.
- This construct is ideally suited for I/O or initialization.

Single Worksharing Construct Contd.

```
1  void work1() {}
2  void work2() {}
3  void single_example()
4  {
5    #pragma omp parallel
6    {
7      #pragma omp single
8      printf("Beginning work1.\n");
9    work1();
10     #pragma omp single
11     printf("Finishing work1.\n");
12     #pragma omp single nowait
13     printf("Finished work1 and beginning work2.\n");
14   work2();
15   }
16 }
```

Excercise 1: For simple worksharing examples, compile and run “worksharing1.c”.

Master construct

`master` construct:

- The `master` construct specifies a structured block that is executed by the master thread of the team.
- There is no implied barrier either on entry to, or exit from, the master construct.

Combined Parallel Work-Sharing Constructs

Combined parallel worksharing constructs are shortcuts that can be used when a parallel region comprises precisely one worksharing construct.

```
1  //Full for and sections
       versions
2  #pragma omp parallel
3  {
4    #pragma omp for
5      for-loop
6  }
7  #pragma omp parallel
8  {
9    #pragma omp sections
10   {
11     [#pragma omp section]
12       structured block
13     [#pragma omp section
14       structured block]
15     ...
16   }
17 }
```

```
1  //Combined for version
2  #pragma omp parallel for
3  {
4    for-loop
5  }
6  //Combined sections version
7  #pragma omp parallel sections
8  {
9    [#pragma omp section]
10     structured block
11   [#pragma omp section
12     structured block]
13     ...
14   }
15 }
```

WITS UNIVERSITY

Outline

Task Worksharing Construct

- Tasks are independent units of work.
- Threads are assigned to perform the work of each task.

Task Construct Contd.

Task Construct Syntax

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

where clause can be

- if(*expression*): if *expression*=TRUE, then the task is immediately executed.
- shared
- private
- firstprivate
- default(shared|none)
- untied

Task Construct Contd.

- 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 Construct Contd.

- 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;

12     }
13   }
14 }
```

Task Construct Contd.

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[]){

3   printf("A ");
4   printf("race ");
5   printf("car ");

7   printf("\n");
8   return 0;
9 }
```

Task Construct Example

Example (5)

Understanding Task Construct

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

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      printf("race ");
8      printf("car ");
9      }
10   }
11   printf("\n");
12   return 0;
13 }
```

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("race ");
9      #pragma omp task
10     printf("car ");
11     }
12   }
13   printf("\n");
14   return 0;
15 }
```

Task Construct Example

Example (9)

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("race ");
9      #pragma omp task
10     printf("car ");
11     printf("is fun to watch ");
12     }
13   }
14   printf("\n");
15   return 0;
16 }
```

Task Construct Example

Example (10)

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("race ");
9      #pragma omp task
10     printf("car ");
11     #pragma omp taskwait
12     printf("is fun to watch ");
13     }
14   }
15   printf("\n");
16   return 0;
17 }
```

Task Construct Example

Example (11)

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 (12)

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 }
```

Exercise 1

Parallelize the program `fib_v00.c` for computing the *n*th Fibonacci number.

Task Construct Contd.

Task switching: *untied*:

```
1  #define ONEBILLION 1000000000L
2  ......
3  #pragma omp parallel
4  {
5    #pragma omp single
6    {
7      for(i=0; i<ONEBILLION; i++)
8        #pragma omp task
9          process(item[i]);
10   }
11   ......
12   /* Untied task: any other thread is eligible to resume
13   the task generating loop*/
14   #pragma omp single
15   {
16     #pragma omp task untied
17       for(i=0; i<ONEBILLION; i++)
18         #pragma omp task
19           process(item[i]);
20   }
21 }
```

Exercise 2:

Parallelize the sequential *quicksort* program (`qsort_v00.c`) using OpenMP `task` construct. Compare the performances of three versions of quicksort implementations, i.e., the sequential implementation, the OpenMP implementation using `sections` construct, and the OpenMP implementation using `task` construct.

Summary for clause applicability

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Figure: Clause applicability for OpenMP constructs

WITS UNIVERSITY

Outline

Data Environment

- Shared memory programming model. Hence most variables are shared by default.
- Global variables are shared among threads. For C, file scope variables and static variables.
- Not everything is shared. Loop indexes are `private` Automatic variables within a statement block are `private`.

Data Environment Contd.

Example

What are the data scoping characteristics of the variables in the following code fragment?

```
1  ......
2  int b[3], i;
3  char *ptr;
4  cptr=malloc(sizeof(char)*32);
5  #pragma omp parallel
6  {
7    #pragma omp for
8      for(i=0; i<3; i++)
9        b[i]=i;
10 }
```

Outline

Changing Storage Attributes

- Changing storage attributes using clauses:
  - shared
  - private: private(var)
- firstprivate: Variables are initialized from shared variable.

Example

What is the result of the following code being executed?

```
1 ......
2 int tmp=0;
3 #pragma omp parallel for
      private(tmp)
4   for(int j=0; j<1000; ++j)
5     tmp+=j;
6 printf("%d\n", tmp);
```

Example (6)

What is the result of the following code being executed?

```
1 ......
2 int tmp=0;
3 /*Each thread gets its own
      copy of tmp with an
      initial value of 0*/
4 #pragma omp parallel for
      firstprivate(tmp)
5   for(int j=0; j<1000; ++j){
6     if((j%2)==0) tmp++;
7     A[j]=tmp;
8   }
```

Changing Storage Attributes Contd.

- *default* clause: The *default* clause allows the user to specify a default scope for all variables in a parallel region.
- For C/C++, the syntax is *default(shared |none)*.

Example

Consider the following variables:

```
int A=1, B=1, C=1;
#pragma omp parallel private(B) firstprivate(C)
```

Questions:

- Are *A, B, C* local to each thread or shared inside the parallel region?
- What are their initial values inside and values after the parallel region?

Changing Storage Attributes Contd.

- lastprivate: Variables update shared variable using value from last iteration.

Example

```
1  void sq2(int n, double *lastterm)
2  {
3    double x; int i;
4    #pragma omp parallel for lastprivate(x)
5    for(int i=0; i<1000; i++){
6      x=a[i]*a[i]+b[i]*b[i];
7      b[i]=sqrt(x);
8    }
9    /*x has the value it held for the last sequential iteration,
         i.e., for i=(n-1)*/
10   *lastterm = x;
11 }
```

More Synchronization Constructs

- **#pragma omp master**
  The *master* construct denotes a structured block that is only executed by the master thread. Other threads just skip it.

```
1  #pragma omp parallel shared(a,b) private(i)
2  {
3     #pragma omp master
4     {
5        a = 10;
6        printf("Master construct is executed by thread %d\n",
7               omp_get_thread_num());
8     }
9     #pragma omp barrier
10    #pragma omp for
11    for (i=0; i<n; i++)
12       b[i] = a;
13 } /*-- End of parallel region --*/

15    printf("After the parallel region:\n");
16    for (i=0; i<n; i++)
17       printf("b[%d] = %d\n",i,b[i]);
18 }
```

More Synchronization Constructs

- **nowait** clause: Removes the implicit barrier at the end of a worksharing construct.

Example

```
1  #pragma omp parallel shared(A,B,C) private(id)
2  {
3    id=omp_get_thread_num();
4    A[id]=big_calc1(id);
5    #pragma omp barrier
6    #pragma omp for
7      for(i=0;i<N;i++){
8        C[i]=big_calc2(i,A);
9      }//Implicit barrier
10   #pragma omp for nowait
11     for(i=0; i< N; i++){
12       B[i]=big_calc3(C,i);
13     }//No implicit barrier due to nowait
14   A[id]=big_calc4(id);
15 }//Implicit barrier at the end of a parallel region
```

More OpenMP Clauses: Copyprivate Clause

- Used with a *single* region (only) to broadcast values of privates from one member of a team to the rest of the team.

Example

Using *copyprivate* clause.

```
1  #include <omp.h>
2  void input_parameters(int, int);
3  void do_work(int, int);
4  void main(){
5    int Nsize, choice;
6    #pragma omp parallel private(Nsize, choice)
7    {
8      #pragma omp single copyprivate(Nsize,choice)
9        input_parameters(Nsize, choice);
10     do_work(Nsize, choice);
11   }
12   ......
13 }
```

Loop Carried Dependency

- Loop carried dependency: Dependencies between instructions in different iterations of a loop;
- What are the dependencies in the following loop?

```
for(i=0; i<N; i++)
{
B[i]=tmp;
A[i+1]=B[i+1];
tmp=A[i];
}
```

Loop Carried Dependency Contd.

It helps to unroll the loop to see the dependencies.

```
1   i=1:
2     B[1]=tmp;
3     A[2]=B[2];
4     tmp=A[1];

6   i=2:
7     B[2]=tmp;
8     A[3]=B[3];
9     tmp=A[2];

11  i=3:
12    B[3]=tmp;
13    A[4]=B[4];
14    tmp=A[3];
15  ......
```

Loop Carried Dependency Example

```
1 //Loop dependency
2 int i, j, A[MAX];
3 j=5;
4 for(i=0; i<MAX; i++){
5   j+=2;
6   A[i]=big(j);
7   }
8 ......
```

Race Condition

- Shared data requires special care.
- A problem may arise in case multiple threads access the same memory section simultaneously.
- This can lead to a race condition.
  - Read only data is no problem
  - Updates have to be checked for race condition.
- Different runs of same program might give different results due to race conditions.

Lab Exercises

Complete all the exercises in the class.

References

- Introduction to Parallel Computing. Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar.
  https://www.cs.purdue.edu/homes/ayg/book/Slides/
- 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://computing.llnl.gov/tutorials/openMP/#Introduction
- OpenMP Application Programming Interface, https://www.openmp.org/resources/refguides/
- OpenMP Application Programming Interface Examples, https://www.openmp.org/specifications/

WITS UNIVERSITY