Parallel Computing
平行計算

Tien-Hsiung Weng
翁添雄
大葉大學資工系
thweng@pu.edu.tw

Lecture 4

Programming Shared-Address Space with OpenMP

Lecture 4

Topics

- Introduction to OpenMP
- OpenMP directives
  - specifying concurrency
    - parallel regions
    - loops, task parallelism
- Synchronization directives
  - reductions, barrier, critical, ordered
- Data handling clauses
  - shared, private, firstprivate, lastprivate
- Library primitives
- Environment variables
- Example of SPMD style OpenMP program

Introduction to OpenMP

- Open specifications for Multi Processing
- An API for explicit multi-threaded, shared memory parallelism
- Three components
  - compiler directives
  - runtime library routines
  - environment variables
- Higher-level programming model than Pthreads
  - support for concurrency, synchronization, and data handling
  - not mutexes, condition variables, data scope, and initialization
- Portable
  - API is specified for C/C++ and Fortran
  - implementations on many platforms (most Unix, Windows NT)
- Standardized

Introduction to OpenMP

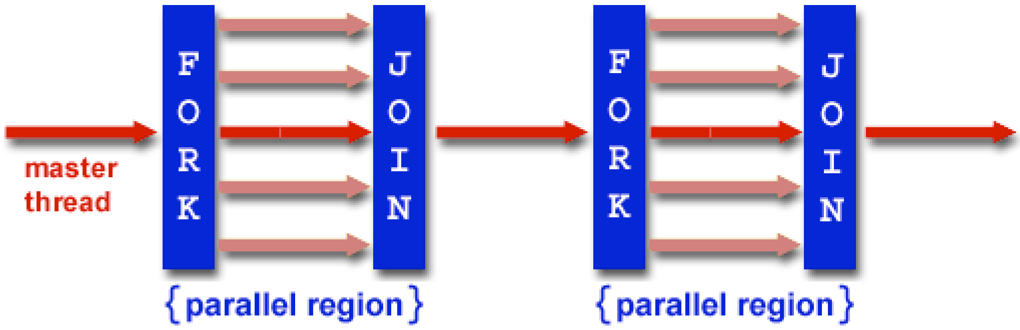
- **Parallelism is explicit**
  - **It is not an automatic programming programming model**
  - **programmer full control (and responsibility) over parallelization**
- **No data locality control**
  - **No guaranteed to make the most efficient use of shared memory**
- Not n**ecessarily implemented identically by all vendors**
- **designed for shared address spaced machines**
  - **Not for distributed memory parallel systems (by itself)**

Introduction to OpenMP

- **Advantages**
  - **Ease of use**
  - **Enables incremental parallelization of a serial program**
  - **Supports both coarse-grain and fine-grain parallelism**
  - **Portable**
  - **Standard**

OpenMP: Fork-Join Parallelism

- **OpenMP program begins execution as a single master thread**
- **Master thread executes sequentially until 1st parallel region**
- **When a parallel region is encountered, master thread**
  - **creates a group of threads**
  - **becomes the master of this group of threads**
  - **is assigned the thread id 0 within the group**



OpenMP Directive Format

C and C++ use compiler directives
  - **prefix: #pragma …**
- **Fortran uses significant comments**
  - **prefixes: !OMP, COMP, *$OMP**
- **A directive consists of a directive name followed by clauses**
- **C:**
  - **#pragma omp parallel default(shared) private(i,j)**
- **Fortran:**
  - **!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(i,j)**

OpenMP parallel Region Directives

#pragma omp parallel [clause list]

Possible clauses in [clause list]

- **Conditional parallelization**
  - **if (scalar expression)**
    - **determines whether the parallel construct creates threads**
- **Degree of concurrency**
  - **num_threads(integer expression)**
    - **Specifies the number of threads to create**
- **Data Handling**
  - **private (variable list)**
    - **specifies variables local to each thread**
  - **firstprivate (variable list)**
    - **similar to the private**
    - **private variables are initialized to variable value before the parallel directive**
  - **shared (variable list)**
    - **specifies that variables are shared across all the threads**

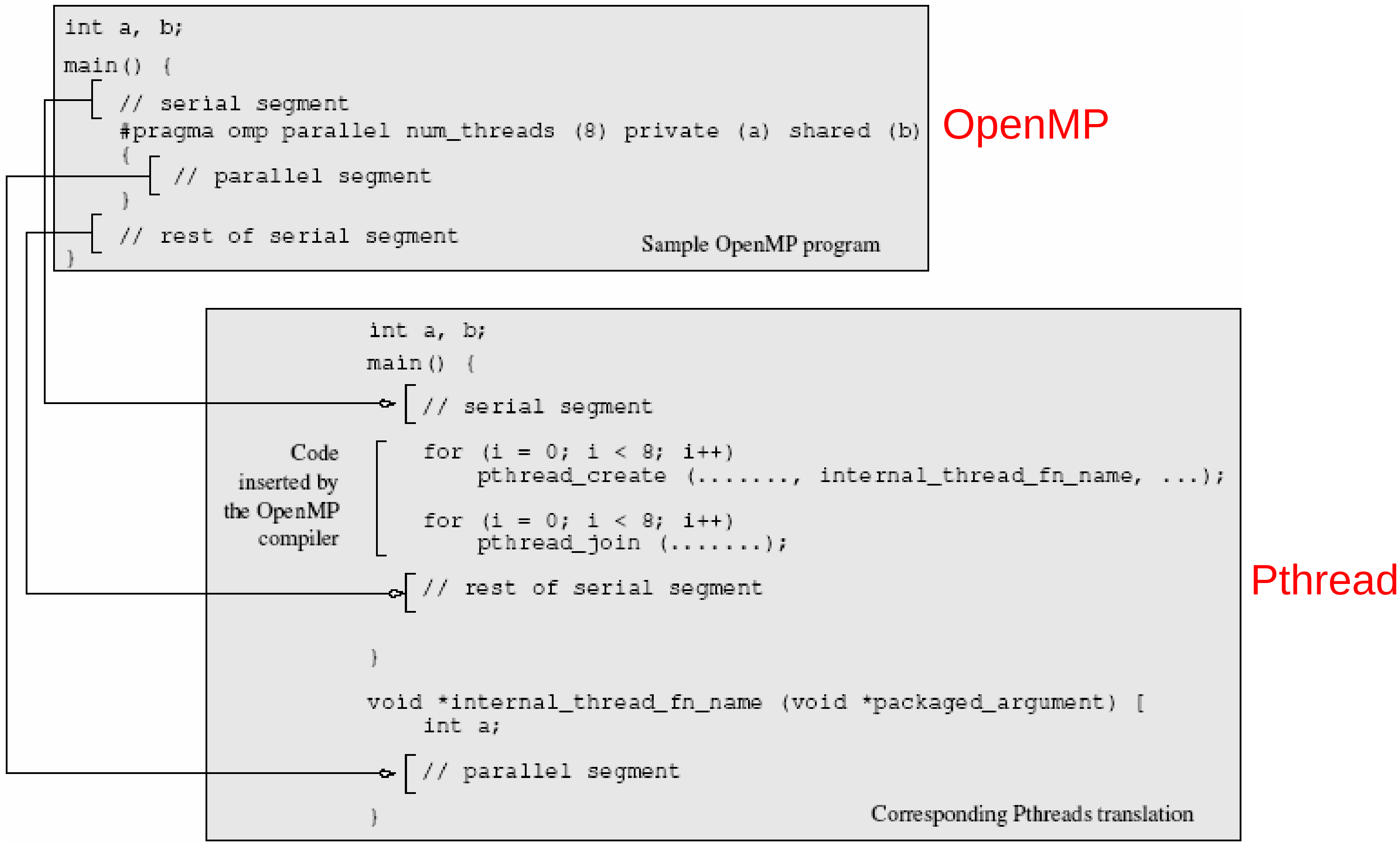
Interpreting an OpenMP Parallel Directive

```
#pragma omp parallel if (is_parallel==1) num_threads(8) \
                     private (a) shared (b) firstprivate(c) default(none)
{
    /* structured block */
}
```

Meaning

- if (is_parallel== 1) num_threads(8)
  - If the value of the variable is_parallel is one, create 8 threads
- private (a) shared (b)
  - each thread gets private copies of variables a and c
  - each thread shares a single copy of variable b
- firstprivate(c)
  - each private copy of c is initialized with the value of c in main thread when the parallel directive is encountered
- default(none)
  - default state of a variable is specified as none (rather than shared)
  - signals error if not all variables are specified as shared or private

OpenMP Programming Model



- A sample OpenMP program along with its Pthreads translation that might be performed by an OpenMP compiler.

Reduction Clause in OpenMP

- The *reduction* clause specifies how multiple local copies of a variable at different threads are combined into a single copy at the master when threads exit.
- The usage of the *reduction* clause is *reduction (operator: variable list).*
- The variables in the list are implicitly specified as being private to threads.
- The *operator* can be one of +*, *, -, &, |, ^, &&, and* ||*.*

```
#pragma omp parallel reduction(+: sum) num_threads(8) {
/* compute local sums here */
}
/*sum here contains sum of all local instances of sums */
```

OpenMP Programming: Example

```
/* ******************************************************************
An OpenMP version of a threaded program to compute PI.
****************************************************************** */
#pragma omp parallel default(private) shared (npoints) \
    reduction(+: sum) num_threads(8)
{
    num_threads = omp_get_num_threads();
    sample_points_per_thread = npoints / num_threads;
    sum = 0;
    for (i = 0; i < sample_points_per_thread; i++) {
        rand_no_x =(double)(rand_r(&seed))/(double)((2<<14)-1);
        rand_no_y =(double)(rand_r(&seed))/(double)((2<<14)-1);
        if (((rand_no_x - 0.5) * (rand_no_x - 0.5) +
            (rand_no_y - 0.5) * (rand_no_y - 0.5)) < 0.25)
            sum ++;
    }
}
```

Worksharing DO/for Directive

- **for directive partitions parallel iterations across threads**
- **DO is the analogous directive for Fortran**
- **Usage:**

```
#pragma omp for [clause list]
/* for loop */
```

- **Possible clauses in [clause list]**
  - **private, firstprivate, lastprivate**
  - **reduction**
  - **schedule, nowait, and ordered**
- **Implicit barrier at end of for loop**

Using Worksharing for Directive

```
#pragma omp parallel default(private) shared (npoints) \
reduction(+: sum) num_threads(8)
{
    sum = 0;
    #pragma omp for
     for (i = 0; i < npoints; i++) {
          rand_no_x =(double)(rand_r(&seed))/(double)((2<<14)-1);
          rand_no_y =(double)(rand_r(&seed))/(double)((2<<14)-1);
          if (((rand_no_x - 0.5) * (rand_no_x - 0.5) +
             (rand_no_y - 0.5) * (rand_no_y - 0.5)) < 0.25)
             sum ++;
      }
}
```

worksharing for divides work

Implicit barrier at end of loop

Example: Matrix Multiply

```
#pragma omp parallel for
   for(i=0; i<n; i++)
      for(j=0; j<n; j++) {
         c[i][j] =0.0;
         for (k=0; k<n; k++)
            c[i][j] += a[i][k]*b[k][i];
      }
```

a,b,c are shared

i,j,k are private

Private Variables

#pragma omp parallel for private(list)

- **Compiler sets up a private copy of each variable in the list for each thread**
- **Our examples use OpenMP for and DO**
- **But these apply to other region and worksharing directives**
- **For compiler: thread has its own stack**

Example: Private Variables

```
for (i=0; i<n; i++) {
    tmp = a[i];
    a[i] = b[i];
    b[i] = tmp;
}
```

Swaps the values of a and b

Loop-carried dependence on tmp

Easily fixed by privatizing tmp

Example: Private Variables

```
#pragma omp parallel for private(tmp)
for (i=0; i<n; i++) {
    tmp = a[i];
    a[i] = b[i];
    b[i] = tmp;
}
```

Removes dependence on tmp

Would be more difficult to do in Pthreads

Example: Private Variables

```
for (i=0; i<n; i++) {
    tmp[i] = a[i];
    a[i] = b[i];
    b[i] = tmp[i];
}
```

Requires sequential program change

Wasteful in space, O(n) vs O(p)

Example: Private Variables

```
F()
{   int tmp;   /* local allocation on stack */
    for (i=0; i<n; i++) {
        tmp[i] = a[i];
        a[i] = b[i];
        b[i] = tmp[i];
    }
}
```

So, tmp is local to each thread

Firstprivate and Lastprivate

The initial and final values of private variables are unspecified

A firstprivate variable is private, and the private copies are initialized using its value before the loop

A lastprivate variable is private, and the thread executing the {sequentially last iteration/lexically last section} updates the version of the object outside the parallel region

Example: Firstprivate and Lastprivate

```
for(i=0; (i<n) && b[i]; i++)
    a[i] = b[i];
for(i=0; j<n; j++)
    a[j] = 1.0;
```

Sets all elements of a to the value of the corresponding element in b, up to first zero value in b

Sets all further elements of a to 1.0

Example: Firstprivate and Lastprivate

```
#pragma omp parallel for lastprivate(i)
for(i=0; (i<n) && b[i]; i++)
    a[i] = b[i];
#pragma omp parallel for firstprivate(i)
for(i=0; j<n; j++)
    a[j] = 1.0;
```

Sets all elements of a to the value of the corresponding element in b, up to first zero value in b

Sets all further elements of a to 1.0

Data Environment Directives

Private
Firstprivate
Lastprivate
Reduction
Threadprivate
Copyin

For good performance, OpenMP code should use private variables whenever possible

- **Reduces cache problems**

However, this could waste a lot of memory
Use of reductions also extremely important

Mapping Iterations to Threads

schedule clause of the for directive

- **Recipe for mapping iterations to threads**
- **Usage: schedule(scheduling_class[, parameter]).**
- **Four scheduling classes**
  - **static: work partitioned at compile time**
    - **iterations statically divided into pieces of size *chunk***
    - **statically assigned to threads**
  - **dynamic: work evenly partitioned at run time**
    - **iterations are divided into pieces of size *chunk***
    - **chunks dynamically scheduled among the threads**
    - **when a thread finishes one chunk, it is dynamically assigned another**
    - **default chunk size is 1**
  - **guided: guided self-scheduling**
    - **chunk size is exponentially reduced with each dispatched piece of work**
    - **the default chunk size is 1**
  - **runtime:**
    - **scheduling decision from environment variable OMP_SCHEDULE**
    - **illegal to specify a chunk size for this clause.**

Statically Mapping Iterations to Threads

- /* static scheduling of matrix multiplication loops */
- #pragma omp parallel default(private) \
-                 shared (a, b, c, dim) num_threads(4)
- #pragma omp for schedule(static)
- for (i = 0; i < dim; i++) {
-     for (j = 0; j < dim; j++) {
-         c(i,j) = 0;
-         for (k = 0; k < dim; k++) {
-             c(i,j) += a(i, k) * b(k, j);
-         }
-     }
- }

- static schedule maps iterations to threads at compile time

Avoiding Unwanted Synchronization

- **Default: worksharing for loops end with an implicit barrier**
- **Often, less synchronization is appropriate**
  - **series of independent for-directives within a parallel construct**
- **nowait clause**
  - **modifies a for directive**
  - **avoids implicit barrier at end of for**

Avoiding Synchronization with nowait

```
#pragma omp parallel
{
#pragma omp for nowait
  for (i = 0; i < nmax; i++)
      if (isEqual(name, current_list[i])
         processCurrentName(name);
#pragma omp for
  for (i = 0; i < mmax; i++)
      if (isEqual(name, past_list[i])
          processPastName(name);
}
```

any thread can begin second loop immediately without waiting for other threads to finish first loop

Using the sections Directive

```
#pragma omp parallel
{
   #pragma omp sections
     {
        #pragma omp section
         {     taskA();
         }
       #pragma omp section
        {     taskB();
        }
      #pragma omp section
        {     taskC();
        }
     }
}
```

parallel section encloses all parallel work

sections: task parallelism

three concurrent tasks

Synchronization Constructs in OpenMP

#pragma omp barrier

#pragma omp single [clause list]

structured block

#pragma omp master

structured block

Use MASTER instead of SINGLE wherever possible

- **MASTER = IF-statement with no implicit BARRIER**
  - **equivalent to IF(omp_get_thread_num() == 0) {...}**
- **SINGLE: implemented like other worksharing constructs**
- **keeping track of which thread reached SINGLE first adds overhead**

Synchronization Constructs in OpenMP

#pragma omp critical [(name)]
structured block
#pragma omp ordered
structured block

Similar to Pthreads mutex locks

critical section: like a named lock
for loops with carried dependences

Example Using critical

```
#pragma omp parallel
{
#pragma omp for nowait shared(best_cost)
  for (i = 0; i < nmax; i++) {
        int my_cost;
        …
        #pragma omp critical
         { if (best_cost <my_cost)
              best_cost = my_cost;
         } …
  }
}
```

critical ensures mutual exclusion when accessing shared state

Example Using ordered

```
#pragma omp parallel
{
#pragma omp for nowait shared(best_cost)
   for (k = 0; k < nmax; k++) {
     …
     #pragma omp ordered
       { a[k] =  a[k-1] +
         …;
       } …
   }
}
```

ordered ensures carried dependence does not cause a data race

OpenMP Library Functions

Processor count

```
  int omp_get_num_procs(); /* # PE currently available */
  int omp_in_parallel(); /* determine whether running in parallel */
```

Thread count and identity

```
/* max # threads for next parallel region. only call in serial region */
   void omp_set_num_threads(int num_threads);
   int omp_get_num_threads(); /*# threads currently active */
   int omp_get_max_threads(); /* max # concurrent threads */
   int omp_get_thread_num(); /* thread id */
```

OpenMP Environment Variables

- **OMP_NUM_THREADS**
  - **specifies the default number of threads for a parallel region**
- **OMP_SET_DYNAMIC**
  - **specfies if the number of threads can be dynamically changed**
- **OMP_NESTED**
  - **enables nested parallelism**
- **OMP_SCHEDULE**
  - **specifies scheduling of for-loops if the clause specifies runtime**

OpenMP SPMD Style

- SPMD (Single Program Multiple Data)
- The same program on each CPU accessed different data

OpenMP SPMD Style

```
#include <omp.h>
main()
{    long int i;
     long int A[1000000];
     float B[1000000];
     float c[1000000];
     printf("omp_get_num_procs = %4d \n",omp_get_num_procs());
     printf("omp_get_max_threads = %4d \n",omp_get_max_threads());
     #pragma omp parallel
     {
          #pragma omp for
        for (i=1; i<=1000000; i++)
          { A[i] = i;
            B[i] = A[i] *2.3;
}
     }
     for(i=10; i<=100; i++) printf(" %7d ",A[i]);
     printf("\n");
}
```

OpenMP SPMD Example

```
#include <omp.h>
long int A[1000000];
int mystart, myend;
float B[1000000];
void mywork(int, int);

#pragma omp threadprivate(mystart, myend)
main()
{    long int i, iam;
     int N, nthreads, chunk,temp;

     #pragma omp parallel private(iam, nthreads, chunk)
     {  nthreads = omp_get_num_threads();
        iam = omp_get_thread_num();
        chunk = (N + nthreads - 1) / nthreads;
        mystart = iam * chunk + 1;
        temp = (iam+1) * chunk;
        myend =  (temp <= N) ? temp : N;
        mywork(mystart, myend);
     }
     for(i=1; i<=100; i++) printf(" %7d ",A[i]);
     printf("\n");
}
```

```
void mywork(int mystart, int myend)
{  int i;
        for (i=mystart; i<=myend; i++)
          { A[i] = i;
            B[i] = A[i] *2.3;
          }
}
```