

CS 292

Introduction to Parallel Computing

Spring 2008

VANDERBILT



School of Engineering

Announcements

- Programming assignment #1
 - Two programs using Pthreads
 - Due Friday, 2/29/08

- Questions??

Chapter 7 Topic Overview

Shared memory programming:

- Thread Basics ✓
- The POSIX Thread API ✓
- Synchronization Primitives in Pthreads ✓
- Controlling Thread and Synchronization Attributes ✓
- Composite Synchronization Constructs ✓
- OpenMP: a Standard for Directive Based Parallel Programming

OpenMP Programming Model

- OpenMP directives in C and C++ are based on the `#pragma` compiler directives.
- A directive consists of a directive name followed by clauses.

```
#pragma omp directive [clause list]
```
- OpenMP programs execute serially until they encounter the `parallel` directive, which creates a group of threads.

```
#pragma omp parallel [clause list]  
/* structured block */
```
- Each thread that is created executes the code in the structured block.

OpenMP Programming: Example

```
/* *****  
An OpenMP version of a threaded program to compute PI.  
***** */  
#pragma omp parallel \  
private(num_threads, sample_points_per_thread, i, rand_no_x, rand_no_y) \  
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++;  
    }  
}
```

Note: default(private) is not supported in C/C++. Red font indicates a difference between lecture notes and text.

OpenMP Storage Association

- Private variables are undefined on entry and exit of the parallel region.
- The value of the original variable (before the parallel region) is undefined after the parallel region.
- A private variable within a parallel region has no storage association with the same variable outside the region.
- You can use the `firstprivate` and `lastprivate` clause to override this behavior
 - `firstprivate` will initialize the private objects with the value the original object had before entering the parallel construct
 - `lastprivate` causes the thread that executes the sequentially last iteration or section to update the value of objects on exit

Specifying Concurrent Tasks in OpenMP

- The `parallel` directive can be used in conjunction with other directives to specify concurrency across iterations and tasks. Known as *work-sharing*.
- OpenMP provides two directives - `for` and `sections` - to specify concurrent iterations and tasks.
- The `for` directive is used to split parallel iteration spaces across multiple threads. The general form of a `for` directive is as follows:

```
#pragma omp for [clause list]
/* for loop */
```
- The clauses that can be used in this context are: `private`, `firstprivate`, `lastprivate`, `reduction`, `schedule`, `nowait`, **and** `ordered`.

Specifying Concurrent Tasks in OpenMP: Example

```
#pragma omp parallel private(rand_no_x,rand_no_y) \  
    shared(npoints) reduction(+: sum) num_threads(8)  
{  
    sum = 0;  
    #pragma omp for  
    for (i = 0; i < npoints; i++) // i is private by default  
    {  
        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++;  
    }  
}
```

- The iterations of the loop following the directive are automatically split across the various threads. There is an implied barrier at the end of the loop.
- Note that the only difference between this code and the serial version is the addition of the directives.

Assigning Iterations to Threads

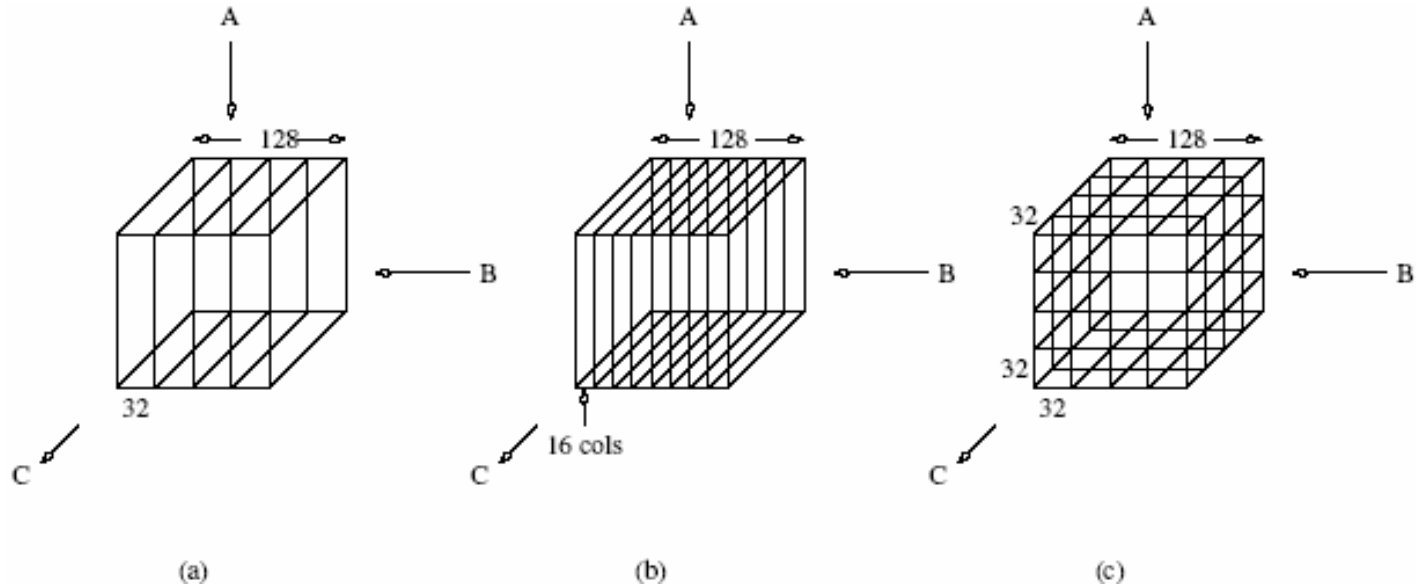
- The `schedule` clause of the `for` directive deals with the assignment of iterations to threads.
- The general form of the `schedule` directive is
`schedule(scheduling_class[, parameter]).`
- OpenMP supports four scheduling classes: `static`, `dynamic`, `guided`, and `runtime`.
 - We will only look at `static`, but will look at reasons why `static` is not always the best choice

Assigning Iterations to Threads: Example

```
/* static scheduling of matrix multiplication loops */
#pragma omp parallel private(i, j, k) 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);
        }
    }
}
```

- Each thread will be give a ***dim/4*** chunk of the iteration space of the i-loop

Assigning Iterations to Threads: Example



- Three different schedules using the static scheduling class of OpenMP.
 - a) `schedule(static)`
 - b) `schedule(static, 16)`
 - c) when each for loop is parallelized with `schedule(static)`

Assigning Iterations to Threads

- `static` scheduling works very well for “*regular*” computation, where each iteration of a loop does approximately the same amount of work.
- For less regular workloads, care needs to be taken in distributing the work over the threads. This is known as *load balancing*.
 - Examples of irregular workloads include: multiplication of triangular matrices, searching linked lists of varying lengths, scatter/gather operations, etc.
- `dynamic` scheduling creates fixed-sized chunks of work (similar to `static`), but the chunks are not pre-assigned to threads, rather a new chunk is given to a thread whenever it finishes an old chunk.
- `guided` scheduling is similar to `dynamic` expect that chunk sizes decrease as chunks are dispatched.

Parallel For Loops

- Often, it is desirable to have a sequence of `for`-directives within a parallel construct that do not execute an implicit barrier at the end of each `for` directive.
- OpenMP provides a clause - `nowait`, which can be used with a `for` directive.

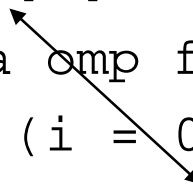
Parallel For Loops: Example

```
#pragma omp parallel firstprivate(name) private(i) \  
    shared(current_list, past_list, nmax, mmax)  
{  
    #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);  
}
```

Parallel For Loops: Example

It is not always safe to use `nowait`. Consider:

```
#pragma omp parallel private(i) shared(a,b,c,d)
{
    #pragma omp for nowait
        for (i = 0; i < nmax; i++)
            a[i] = b[i] + c[i];
    #pragma omp for
        for (i = 0; i < mmax; i++)
            d[i] = a[i] + b[i];
}
```



We need to have updated all of `a[]` first before using `a[]`. It is not safe to use `nowait` in this situation.

The `sections` Directive

- OpenMP supports non-iterative parallel task assignment using the `sections` directive.
- The general form of the `sections` directive is as follows:

```
#pragma omp sections [clause list]
{
    [#pragma omp section
        /* structured block */
    ]
    [#pragma omp section
        /* structured block */
    ]
    ...
}
```

The sections Directive: Example

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

Merging Directives

```
#pragma omp parallel ...
{
  #pragma omp for
  for (i=0; i<n; i++)
  {
    /* body */
  }
}
```

Is identical to

```
#pragma omp parallel for ...
{
  for (i=0; i<n; i++)
  {
    /* body */
  }
}
```

```
#pragma omp parallel ...
{
  #pragma omp sections
  {
    #pragma omp section
    { ...
    }
    #pragma omp section...
  }
}
```

Is identical to

```
#pragma omp parallel sections...
{
  #pragma omp section
  { ...
  }
  #pragma omp section...
}
```

Nesting `parallel` Directives

- Nested parallelism can be enabled using the `OMP_NESTED` environment variable.
- If the `OMP_NESTED` environment variable is set to `TRUE`, nested parallelism is enabled.
- In this case, each parallel directive creates a new team of threads, otherwise the logical team is executed by the originating thread

Nesting parallel Directives

```
/* nested parallelism of matrix multiplication loops */
#pragma omp parallel for private(i, j, k) \
    shared (a, b, c, dim) num_threads(2)
for (i = 0; i < dim; i++) {
    #pragma omp parallel for private(j, k) \
        shared (a, b, c, dim) num_threads(2)
    for (j = 0; j < dim; j++) {
        c(i, j) = 0;
        #pragma omp parallel for private(k) \
            shared (a, b, c, dim) num_threads(2)
        for (k = 0; k < dim; k++) {
            c(i, j) += a(i, k) * b(k, j);
        }
    }
}
```