Shared Memory Programming with OpenMP (2)

2014 Spring

Jinkyu Jeong
(jouncy@skku.edu)
“PARALLEL FOR” DIRECTIVE
Parallel for Directive (1)

- Parallelization directive specialized to for loop
  - `#pragma omp parallel for num_threads(n)`

```c
#pragma omp parallel num_threads(thread_count)
{
    my_rank = omp_get_thread_num();
    my_start = n/thread_count * my_rank;
    my_end = my_start + n/thread_count;
    for(i=my_start; i<my_end; ++i) {
        ...
    }
}
```

```c
#pragma omp parallel for num_threads(thread_count)
for(i=0; i<n; ++i) {
    ...
}
```
Parallel for Directive (2)

- Cannot parallelize while or do-while loops
- For loop should be deterministic
  - Legal form of for loop

\[
\text{for} \quad \begin{aligned}
\text{index} &= \text{start} \quad ; \quad \text{index} \geq \text{end} \quad ; \\
\text{index} &= \text{end} \\
\end{aligned}
\begin{aligned}
\text{index} &= \text{index} + \text{incr} \\
\text{index} &= \text{index} - \text{incr} \\
\text{index} &= \text{index} + \text{incr} \\
\text{index} &= \text{inc}r + \text{index} \\
\text{index} &= \text{index} - \text{incr} \\
\end{aligned}
\]

- \text{index} should be an \text{integer} or a \text{pointer} (not float)
- \text{start}, \text{end}, \text{incr} should not change inside loop
- \text{index} should only be modified by the increment expression in the for statement
Simplifying Trapezoidal Rules using parallel for

- Serial code

```
    h = (b-a)/n;
    approx = (f(a) + f(b))/2.0;
    for (i = 1; i <= n-1; i++)
        approx += f(a + i*h);
    approx = h*approx;
```

- Parallel code using OpenMP `parallel for + reduction`
  - `parallel for` parallelizes area calculation of individual trapezoids
  - `reduction` performs global sum

```
    h = (b-a)/n;
    approx = (f(a) + f(b))/2.0;
    # pragma omp parallel for num_threads(thread_count) \ 
        reduction(+: approx)
    for (i = 1; i <= n-1; i++)
        approx += f(a + i*h);
    approx = h*approx;
```
Data Dependency (1)

- Parallelizing Fibonacci number generator

```c
fibo[0] = fibo[1] = 1;
for (i=2; i<n; i++)
    fibo[i] = fibo[i-1]+fibo[i-2];
```

```c
fibo[0] = fibo[1] = 1;
#pragma omp parallel for num_threads(2)
for (i=2; i<n; i++)
    fibo[i] = fibo[i-1]+fibo[i-2];
```

Correct result: 1 1 2 3 5 8 13 21 34 55

Incorrect result: 1 1 2 3 5 8 0 0 0 0

Sometimes, we get Incorrect result.
Data Dependency (2)

- Initial state of the fibo[] shared array

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
<th>Thread 0 computes</th>
<th>Thread 1 computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2 0 0 0 0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3 0 0 0 0</td>
<td>6 0 0 0 0</td>
</tr>
</tbody>
</table>

  - Correct result: 1 1 2 3 5 8 13 21 34 55

- Otherwise
  - Incorrect result: 1 1 2 3 5 8 0 0 0 0
Data Dependency (3)

- Loop-carried dependency
  - Data dependency existing between iterations
    - Ex) computation of fibo[n] is dependent to the data stored in fibo[n-1] and fibo[n-2]

- When loop-carried dependency exists, parallel for cannot be used

- How to parallelize a for loop having loop-carried dependency
  - Modify the algorithm
Example of Resolving Loop-carried dependency

- Estimating $\pi$

```c
double factor = 1.0;
double sum = 0.0;
for (k = 0; k < n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}
pi_approx = 4.0*sum;
```

$$\pi = 4 \left[ 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \right] = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

- loop-carried dependency

```c
double sum = 0.0;
#pragma omp parallel for num_threads(thread_count) \ reduction(+:sum) private(factor)
for (k = 0; k < n; k++) {
    if (k % 2 == 0)
        factor = 1.0;
    else
        factor = -1.0;
    sum += factor/(2*k+1);
}
```

- Insures factor has private scope
- Eliminating loop-carried dependency
More about Scope of Variables

- **default**(none|shared|private) clause
  - Set default scope of variables declared outside of the parallelized structured block
  - When default is none, programmer have to specify scope of all variables declared outside
  - Ex) default(none), default(shared)

- **private**(variable list) clause
  - Set the scope variables to be private
  - Ex) private(a, b)

- **shared**(variable list) clause
  - Set the scope of variables to be shared

```c
main() {
    int a, b;
    a = b;
    # pragma omp parallel \
    default(none) private(a, b) \
    {
        int x;
    }
}```
Using **default**(none) in Estimating $\pi$

```c
double sum = 0.0;
#pragma omp parallel for num_threads(thread_count) default(none) reduction(sum) private(k, factor) shared(n)
for (k = 0; k < n; k++) {
    if (k % 2 == 0)
        factor = 1.0;
    else
        factor = -1.0;
    sum += factor/(2*k+1);
}
```

- **default**(none)
  - To specify the scopes of all variables
- **reduction**(sum)
  - sum itself is shared but used for reduction operator +
- **private**(k, factor)
  - k and factor variables are private in each thread
- **shared**(n)
  - n is shared (read-only) among threads
ADDITIONAL DIRECTIVE FOR CONTROLLING THREADS
### Serial Odd-Even Transposition Sort

<table>
<thead>
<tr>
<th>Phase</th>
<th>Subscript in Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0: 9 ↔ 7, 1: 8 ↔ 6, 2: 7, 3: 6, 8</td>
</tr>
<tr>
<td>1</td>
<td>0: 7, 1: 9 ↔ 6, 2: 8, 3: 7, 8</td>
</tr>
<tr>
<td>2</td>
<td>0: 7 ↔ 6, 1: 9, 2: 8, 3: 7, 9</td>
</tr>
<tr>
<td>3</td>
<td>0: 6, 1: 7 ↔ 8, 2: 9, 3: 6, 8, 9</td>
</tr>
</tbody>
</table>

```c
for (phase = 0; phase < n; phase++)
    if (phase % 2 == 0)
        for (i = 1; i < n; i += 2)
            if (a[i-1] > a[i]) Swap(&a[i-1], &a[i]);
    else
        for (i = 1; i < n-1; i += 2)
            if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);
```
for (phase = 0; phase < n; phase++) {
    if (phase % 2 == 0)
        pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
    for (i = 1; i < n; i += 2) {
        if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
        }
    }
else
    pragma omp parallel for num_threads(thread_count) \
        default(none) shared(a, n) private(i, tmp)
    for (i = 1; i < n-1; i += 2) {
        if (a[i] > a[i+1]) {
            tmp = a[i+1];
            a[i+1] = a[i];
            a[i] = tmp;
        }
    }
}
Excessive fork&join is inefficient
for Directive

- Parallelize a loop using threads that are forked in advance

```c
#pragma omp parallel num_threads(n)
{
    #pragma omp for
    for (i=0; i<n; ++i) {
        ...
    }
    #pragma omp for
    for (i=0; i<n; ++i) {
        ...
    }
}
```

Unnecessary fork&join is eliminated
Second OpenMP Odd-Even Sort

```c
#pragma omp parallel num_threads(thread_count) \ 
    default(none) shared(a, n) private(i, tmp, phase)
for (phase = 0; phase < n; phase++) {
    if (phase % 2 == 0)
        #pragma omp for
        for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
                tmp = a[i-1];
                a[i-1] = a[i];
                a[i] = tmp;
            }
        }
    else
        #pragma omp for
        for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
                tmp = a[i+1];
                a[i+1] = a[i];
                a[i] = tmp;
            }
        }
}
```

<table>
<thead>
<tr>
<th>thread_count</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two parallel for directives</td>
<td>0.770</td>
<td>0.453</td>
<td>0.358</td>
<td>0.305</td>
</tr>
<tr>
<td>Two for directives</td>
<td>0.732</td>
<td>0.376</td>
<td>0.294</td>
<td>0.239</td>
</tr>
</tbody>
</table>

Execution time comparison between parallel for and for
Parallel Sections Directive

- Different threads execute different works
- Different works should be specified using *section* directive

```c
#pragma omp parallel sections
{
    #pragma omp section
    A();

    #pragma omp section
    B();
}
/* end of parallel sections */
```

$n$ threads available  1 thread available
Single Directive

- Executed by one thread within a parallel region
  - Any thread can execute the single region
  - Implicit barrier synchronization at the end

```c
#pragma omp parallel
{
    #pragma omp single
    {
        a = 10;
    } /* implicit barrier */

    #pragma omp for
    for (i=0; i<N; i++)
        B[i] = a;

    /* end of parallel region */
```
**Master Directive**

- Executed by the master thread
  - No implicit barrier
  - If a barrier is needed for correctness, must specify one

```c
#pragma omp parallel
{
    #pragma omp master
    {
        a = 10;
    } /* no barrier */
    #pragma omp barrier

    #pragma omp for
    for (i=0; i<N; i++)
    {
        B[i] = a;
    }
} /* end of parallel region */
```