

Lecture 12

Midterm Return

Threaded program design

Announcements

- Makeup lecture on Friday, CSB 002
 - 2pm to 3.20 pm
 - Podcasted
 - No Quiz
- A4 Due next Thursday 11/13
- You must login to Abe and turnin output: see the recent post of A4
www.cse.ucsd.edu/classes/fa08/cse160/HW/A4

Midterm Return

Threaded program design

Roadmap for next 3 Lectures

- Threaded program design
- Code organization and re-use
- How to parallelize serial code (code reorganization)
- How to design with parallelism in mind
- Idioms - Design patterns
- Performance
- Debugging
- Development tools
- Using OpenMP
- Translation from OpenMP to pthreads
- Performance

Case Study - odd/even sort

Parallel Sorting

- Sorting is fundamental algorithm in data processing
 - Given an unordered set of keys $x_0, x_1, \dots, x_{N-1}$
 - Return the keys in sorted order
- The keys may be character strings, floating point numbers, integers, or any object for which the relations $>$, $<$, and $==$ hold
- We'll assume integers here
- Will talk about other algorithms later on

Compare and exchange sorts

- Simplest sort, based on the bubble sort algorithm
- The fundamental operation is compare-exchange

- **Compare-exchange($a[j]$, $a[j+1]$)**

- swaps its arguments if they are in decreasing order
- satisfies the post-condition that $a[j] \leq a[j+1]$
- Returns FALSE if a swap was made

```
for i = N-1 to 1 by -1 do
```

```
    done = TRUE;
```

```
    for j = 0 to i-1 do // Compare-exchange( $a[j]$  ,  $a[j+1]$ )
```

```
        if ( $a[i] < a[j]$ ) {  $a[i] \leftrightarrow a[j]$ ;
```

```
            done=FALSE; }
```

```
    end do
```

```
    if (done) break;
```

```
end do
```

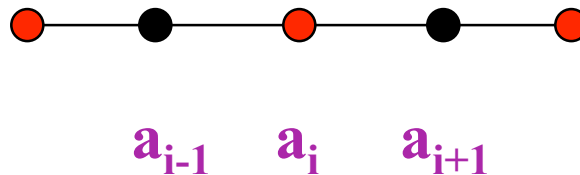
Compare and exchange sorts

- We cannot run bubble sort in parallel owing to the *loop carried dependence* in the inner loop
- The value of $a[j]$ computed in iteration j depends on the $a[i]$ computed in iterations $0, 1, \dots, j-1$

```
for i = N-1 to 1 by -1 do
  done = TRUE;
  for j = 0 to i-1 do
    done = Compare-exchange( $a[j]$  ,  $a[j+1]$ )
  end do
  if (done) break;
end do
```

Odd/Even sort

- If we re-order the comparisons we can parallelize the algorithm
 - number the points as even and odd
 - alternate between sorting the odd and even points
- This algorithm parallelizes since there are no loop carried dependences
- All the odd (even) points are decoupled



The algorithm

```
done = false;
```

```
for i = 0 to n-1 do
```

```
    for j = 0 to n-1 by 2 do // Even
```

```
        done &= Compare-exchange(a[j] , a[j+1]);
```

```
    end do
```

```
    for j = 1 to n-1 by 2 do // Odd
```

```
        done &= Compare-exchange(a[j] , a[j+1]);
```

```
    end do
```

```
    if (done) break;
```

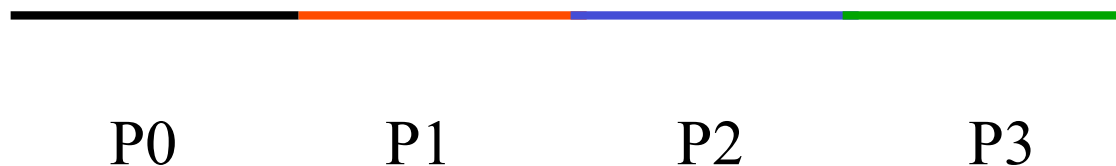
```
end do
```

Parallelizing the algorithm

- Splitting up the data
- Handling communication

Partitioning

- Partitioning is the process of splitting up the data
- We partition the data into intervals, assigning each to a unique thread



Data dependencies

- Each thread updates the interval it owns
- Depends on data owned by other threads needs data “owned” by a neighboring process

