CS 220: Introduction to Parallel Computing

Distributed Sorting

Lecture 20

Today's Agenda

- Sleep Sort
- Bogosort
- Bubble Sort
- Odd/Even Transposition Sort

Today's Agenda

- **Sleep Sort**
- Bogosort
- Bubble Sort
- Odd/Even Transposition Sort

Sorting

- When we have lists of data, it's very useful to be able to **sort** them
- In Python and Java, sorting is built in!
  - list.sort()
  - So nice!
- But as you already guessed, in C we need to provide our own sorting functions

Complexity

- Sorting algorithms can be costly; it takes time to sort a really large list!
- Splitting the data up across several cores can greatly improve performance
- Each process will sort its own version of the list, and then we'll need to merge everything back together

My Favorite Algorithm: Sleep Sort

- The easiest parallel sort to implement is called **sleep sort**
- It may be the laziest (and possibly the slowest) sorting algorithm out there
- Each process receives a number to sort
  - Then it goes to sleep
  - Sets an alarm for (number) seconds
- Wakes up and prints its value!

Today's Agenda

- Sleep Sort
- **Bogosort**
- Bubble Sort
- Odd/Even Transposition Sort

Bogosort

- The bogosort algorithm consists of:
  - Shuffling the elements
  - Determining whether they're in order or not
- Rinse & Repeat
- Note: bogosort **can** benefit from parallelism, right?
  - Kind of in the same way as our password cracker…

Today's Agenda

- Sleep Sort
- Bogosort
- **Bubble Sort**
- Odd/Even Transposition Sort

Bubble Sort

- Bubble sort compares adjacent elements and then swaps them if necessary
- The bigger elements “bubble” to the top
- After one pass of the algorithm, we know the last element is sorted
  - Then continue on…

Bubble Sort in Action

6 5 3 1 8 7 2 4

Source: https://en.wikipedia.org/wiki/Bubble_sort

Parallelizing Bubble Sort

- How can we parallelize the algorithm?
- It's not easy: we can't do the swaps in parallel because we need to wait for the previous step
- A relative of the bubble sort, odd-even transposition sort, can be parallelized though!

An Aside: Complexity

- Is bubble sort efficient?
- Well, we could always ask Obama:
  - https://www.youtube.com/watch?v=k4RRi_ntQc8
  - (Thanks, Obama)
- The bubble sort is definitely the wrong way to go:
  - So many swaps, so many iterations over the list
- Worst-case performance: $O(n^2)$

Today's Agenda

- Sleep Sort
- Bogosort
- Bubble Sort
- **Odd/Even Transposition Sort**

Odd-Even Transposition Sort

- Consists of two phases: **odd** and **even**
- In the even phases, each odd-subscripted element is compared with its “left” neighbor
  - If they’re out of order, they’re swapped
- In the odd phases, each odd-subscripted element is compared with its “right” neighbor
  - If they’re out of order, they’re swapped
- Repeat until sorted

An Example

- Suppose the list is:

```
[5,   9,   4,   3]
 0    1    2    3  /* (subscripts) */
```

- Even phase:
  - Compare (5, 9) and (4, 3)
  - 4 and 3 are out of order. Swap them!

An Example

- The list:

```
[5,    9,    3,    4]
 0     1     2     3   /* (subscripts) */
```

- Odd phase:
  - Compare (9, 3)
- The list:

```
[5,    3,    9,    4]
 0     1     2     3   /* (subscripts) */
```

An Example

- The list:

```
[5,   3,   9,   4]
 0    1    2    3  /* (subscripts) */
```

- Even phase:
  - Compare (5, 3) and (9, 4)
  - Both pairs are out of order. Swap them!
- The list:

```
[3,   5,   4,   9]
 0    1    2    3  /* (subscripts) */
```

An Example

- The list:

```
[3,   5,   4,   9]
 0    1    2    3  /* (subscripts) */
```

- Odd phase:
  - Compare (5, 4). Swap!

-

```
[3,   4,   5,   9]
 0    1    2    3  /* (subscripts) */
```

Parallelizing It

- Distribute one element to each process
- During each phase, send/receive between neighbors
  - We'll use our MPI functions here
- Compare and keep the appropriate element
- How do we know when we're done?
  - When no swaps occurred across both the **odd** and **even** phases