



Parallel Algorithms

Lecture Objectives:

- 1) Explain the operation of the bubble sort algorithm for sorting a list of numbers
- 2) Explain the difference between the bubble sort and the odd-even transposition sort
- 3) Construct a simple implementation of an odd-even sort which runs in parallel.
- 4) Explain the purpose of the Jacobi algorithm for solving mathematical equations
- 5) Construct a simple parallel implementation of the Jacobi algorithm using OpenMP.

$$O(n \log n) \Rightarrow O(n^2)$$

- **Sorting** takes an unordered collection and makes it an ordered one.

Sorting

1	2	3	4	5	6
77	42	35	12	101	5



1	2	3	4	5	6
5	12	35	42	77	101

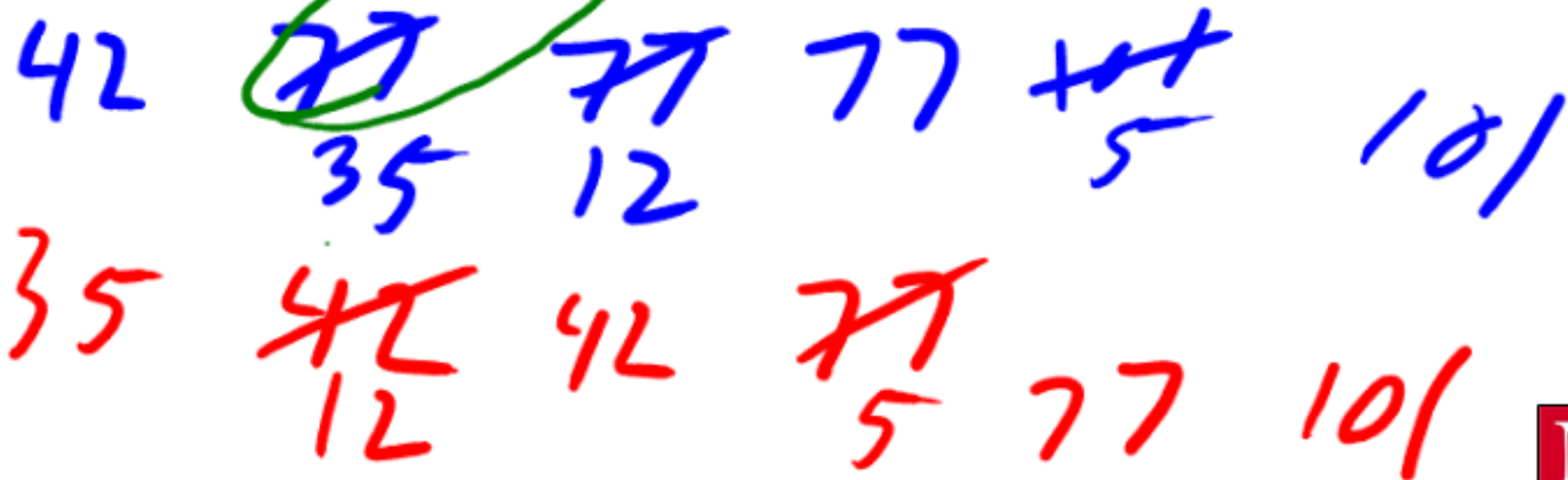
→
In order

Bubble Sort

- Traverse a collection of elements
 - Move from the front to the end
 - “Bubble” the **largest value** to the end using **pair-wise comparisons and swapping**

$$n(n-1) \rightarrow O(n^2)$$

1	2	3	4	5	6
<u>77</u>	<u>42</u>	35	12	101	5



The "Bubble Up" Algorithm

```
index <- 1
last_compare_at <- n - 1

loop
  exitif(index > last_compare_at)
  if(A[index] > A[index + 1]) then
    Swap(A[index], A[index + 1])
  endif
  index <- index + 1
endloop
```

How complex is this?

Complexity

- How complex is this sort?

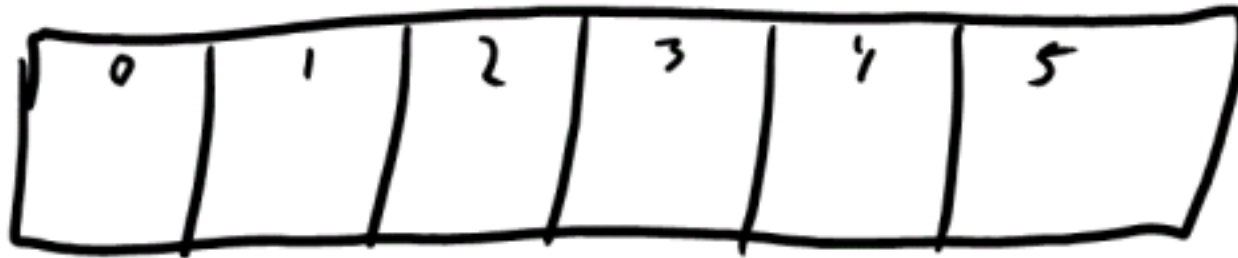
Can we parallelize this?

```
index <- 1
last_compare_at <- n - 1

loop
  exitif(index > last_compare_at)
  if(A[index] > A[index + 1]) then
    Swap(A[index], A[index + 1])
  endif
  index <- index + 1
endloop
```

No! Not
easily parallelizable.

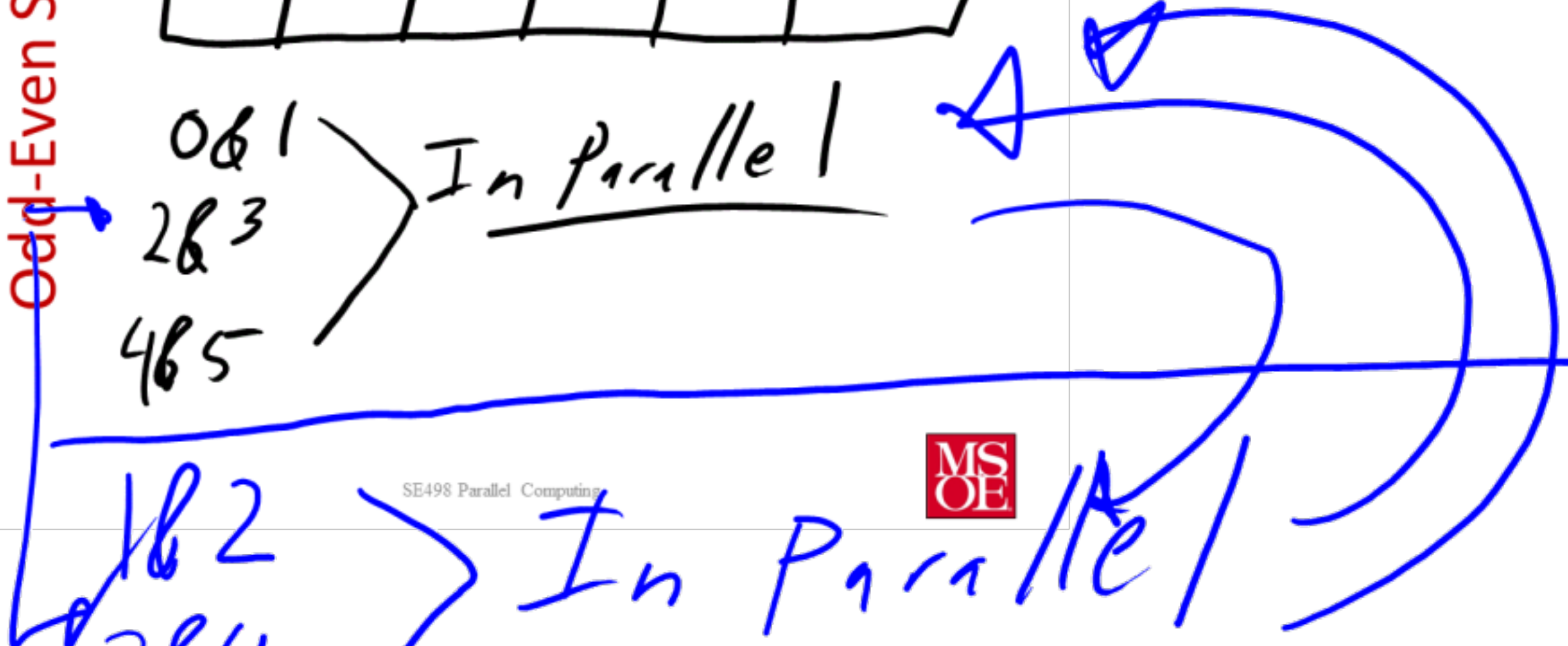
- What if we compared all the odd values together and then compared all of the even values?



Odd-Even Sort

0 & 1
2 & 3
4 & 5
In Parallel

1 & 2
3 & 4
In Parallel



Odd-Even sort algorithm

```
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]);
```


Odd Even Sort Behavior

1	2	3	4	5	6
77	42	35	12	101	5

42 77 12 35 5 101

12 | 77 5 | 35

12 42 | 5 77 | 35 101

5 42 | 35 77 |

5 12 : 35 42 77 101

$n \Rightarrow O(n)$ $O(n^2) \Rightarrow n$ can be done at a time



Lets look at some code

Solving a set of Linear

Equations

- Direct solvers
 - Gauss elimination
 - LU decomposition
 - Iterative solvers
 - Basic iterative solvers
 - Jacobi
 - Gauss-Seidel
 - Successive over-relaxation
 - Krylov subspace methods
 - Generalized minimum residual (GMRES)
 - Conjugate gradient
- Hard*

Jacobi Method

$$\begin{aligned} \alpha_{11} x_1 + \alpha_{12} x_2 + \dots + \alpha_{1j} x_j + \dots + \alpha_{1n} x_n &= \beta_1 \\ \alpha_{21} x_1 + \alpha_{22} x_2 + \dots + \alpha_{2j} x_j + \dots + \alpha_{2n} x_n &= \beta_2 \\ \dots & \dots \\ \alpha_{i1} x_1 + \alpha_{i2} x_2 + \dots + \alpha_{ij} x_j + \dots + \alpha_{in} x_n &= \beta_i \\ \dots & \dots \\ \alpha_{n1} x_1 + \alpha_{n2} x_2 + \dots + \alpha_{nj} x_j + \dots + \alpha_{nn} x_n &= \beta_n \end{aligned}$$

Solution

$$Ax = b$$

Matrix

Sequential Jacobi Algorithm

$$Ax = b$$

$$A = \underline{D} + \underline{L} + \underline{U}$$

$$\underline{x}^{k+1} = \underline{D}^{-1} (\underline{b} - \underline{(L+U)x}^k)$$

- D is diagonal matrix
- L is lower triangular matrix
- U is upper triangular matrix

Example

- Lets solve the following set of simultaneous equations

$$\begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix} x = \begin{bmatrix} 6 \\ 10 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix},$$

$$R = L + U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$y = 6 - 2x$$

$$2x + y = 6$$

$$x + 4y = 10$$

$$x + 4(6 - 2x) = 10$$

$$x + 24 - 8x = 10$$

$$-7x = -14$$

$$\begin{aligned} \therefore x &= 2 \\ \therefore y &= 2 \end{aligned}$$

- Lets solve the following set of simultaneous equations

$$\begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix} x = \begin{bmatrix} 6 \\ 10 \end{bmatrix}$$

$$\rightarrow D = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix},$$

$$R = L + U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Example

$$x^{k+1} = D^{-1}(b - Lx^k + Ux^k)$$

Start: Taking a guess x



Solutions

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9
	1.0000	2.5000	1.8750	2.0625	1.9844	2.0078	1.9980	2.0010	1.9998	2.0001
	1.0000	2.2500	1.8750	2.0313	1.9844	2.0039	1.9980	2.0005	1.9998	2.0001

↑
Guess

↑ ↑ ↑ ↑ ↑ . . .
Pretty
~~Close~~ Answer

Lets look at some implementations using openMP