Parallel Algorithms 2

Outline

Example #1: Matrix Multiplication

Example #2: Trapezoidal Rule

Example #3: Odd-Even Transposition Sort

Example #1: Matrix Multiplication

Many scientific simulations involve modeling using matrices, such as *population studies* and *spread of disease*.

A very common operation on matrices is **multiplying two of them together**.

For example:

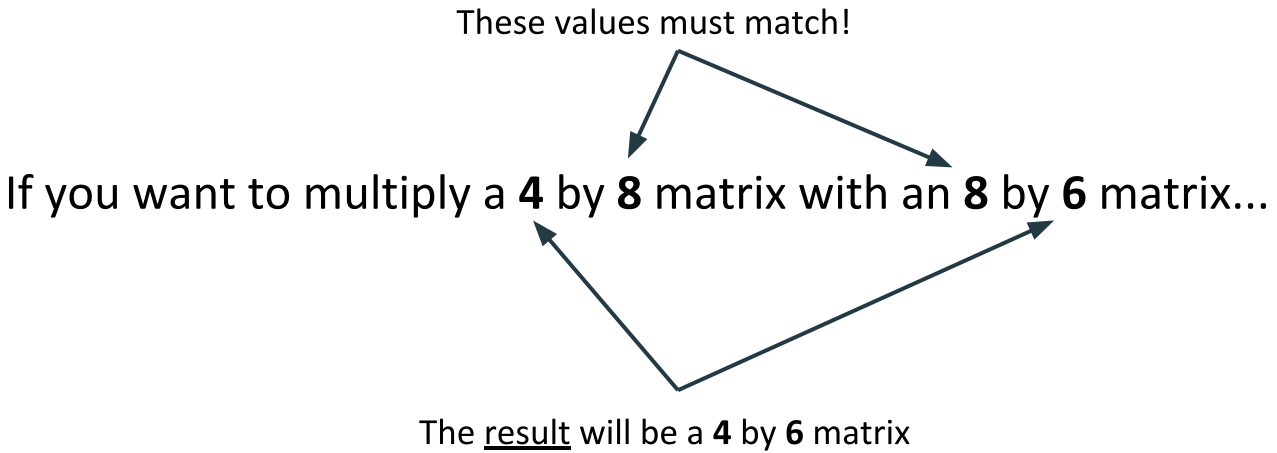
$$\begin{bmatrix} -4 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 9 & -3 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} 15 & 7 \\ 36 & -3 \end{bmatrix}$$

Example #1: Matrix Multiplication

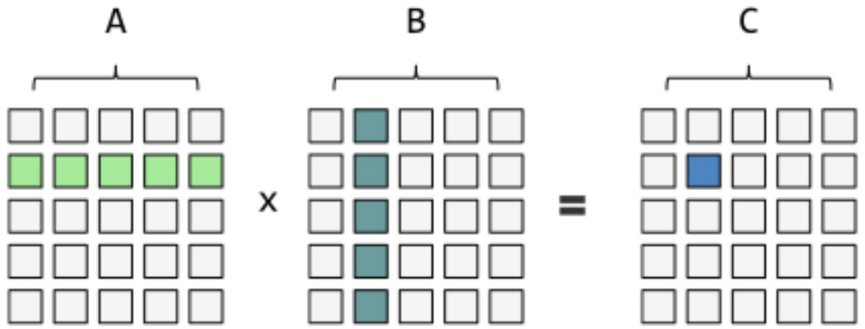
The general process:

1) You can only multiply two matrices if the **number of columns in the first matrix** is the same as the **number of rows in the second matrix**.
2) The resulting matrix will have the same number of *rows as the first matrix* and the same number of *columns as the second matrix*.
3) To compute the value at location **(x,y)** in the result matrix, use row **x** of the first matrix and column **y** of the second, multiplying the corresponding positions and summing them all together.

Example #1: Matrix Multiplication



Example #1: Matrix Multiplication




```
C[i][j] = sum(A[i][k] * B[k][j]) for k = 0 ... n
```

Image from: https://insidehpc.com/2016/01/heterogeneous-streams/

Example #1: Matrix Multiplication

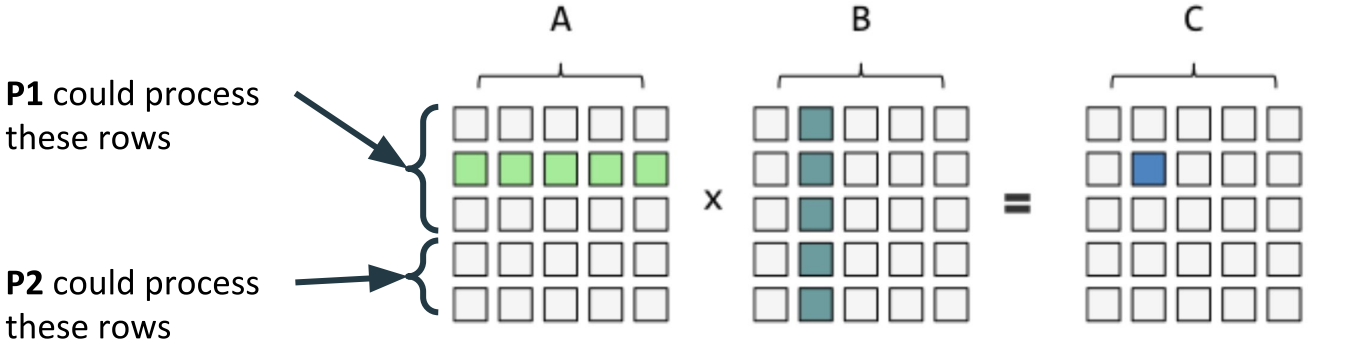
The **sequential** solution for computing result matrix **C** from **A x B**:

```
for each row r in matrix A:
    for each column c in matrix B:
        total = 0
        for each corresponding position i between row r and column c:
            total = total + A[r, i] * B[i, c]
        C[r, c] = total
```

Could this be *parallelized*, and if so, how might you do it? If it helps, imagine you are multiplying two matrices that are each 10,000 x 10,000 elements. That's **a lot** of numbers!

Example #1: Matrix Multiplication

The **parallel** solution: have each process take a portion of the rows!



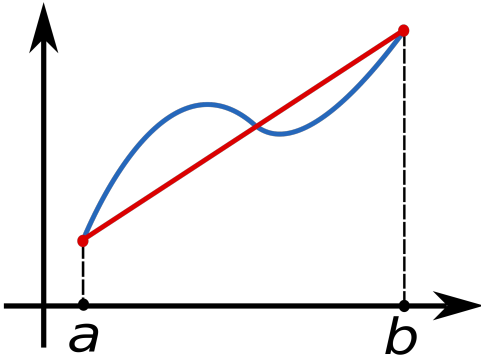

```
C[i][j] = sum(A[i][k] * B[k][j]) for k = 0 ... n
```

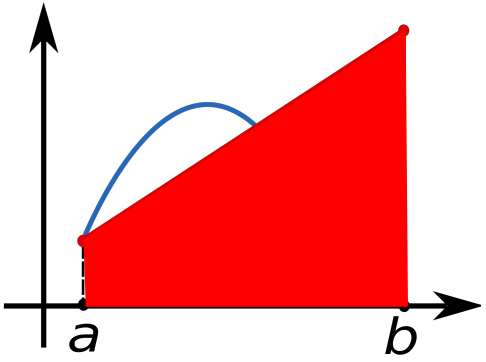
Image from: https://insidehpc.com/2016/01/heterogeneous-streams/

Example #2: Trapezoidal Rule

The **trapezoidal rule** is a numerical technique for approximating the area underneath a curve.

Trapezoids are used as the shape because they fit fairly well, and it is extremely easy to compute the area of a trapezoid..

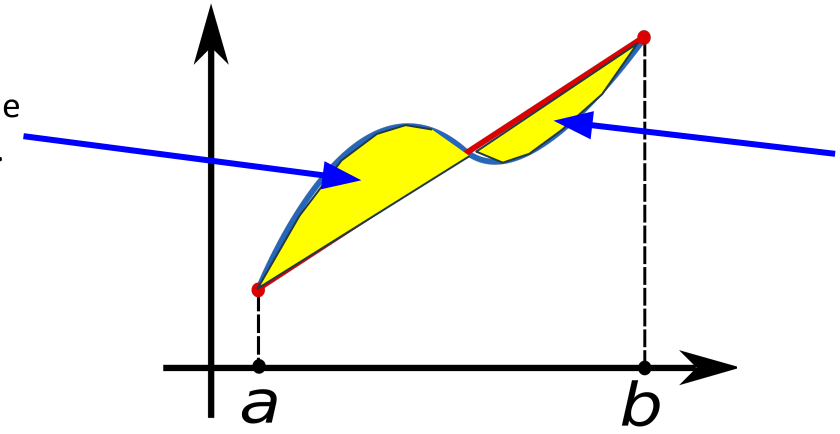




Example #2: Trapezoidal Rule

Remember, this is only an **approximation**!

This isn't included in the result, but it should be.

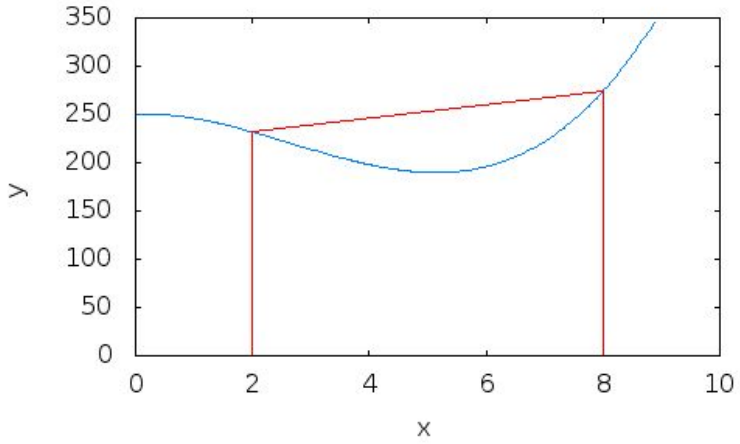


This is included in the result, but it shouldn't be.

Example #2: Trapezoidal Rule

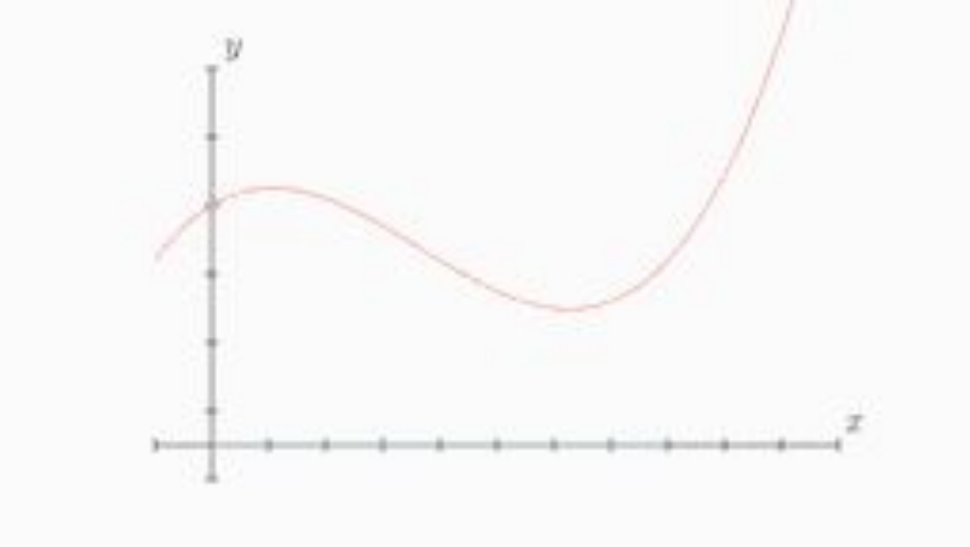
The **trapezoidal rule** works really well as you keep dividing the interval in which you are interested into more and more trapezoids.

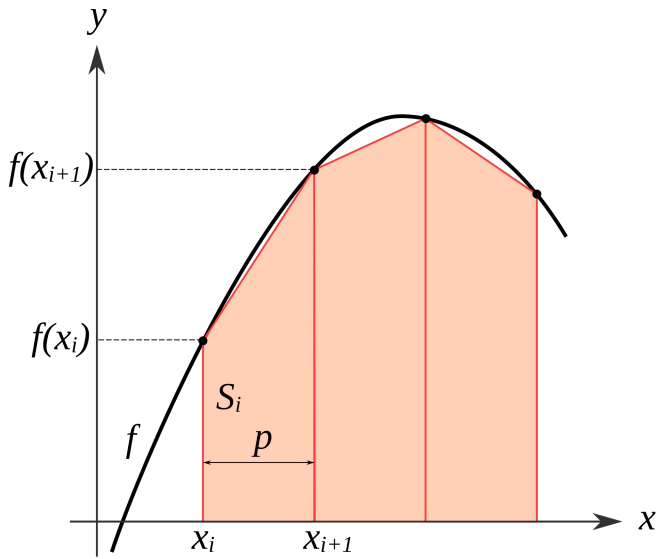
The smaller they are, the ***closer*** they fit the curve, and thus the more accurate the approximation!



Example #2: Trapezoidal Rule

As the shapes are still all trapezoids, it simply turns into a straightforward probably using basic algebra to compute each individual area, and then sum them all up.





Example #2: Trapezoidal Rule

The **sequential** solution:

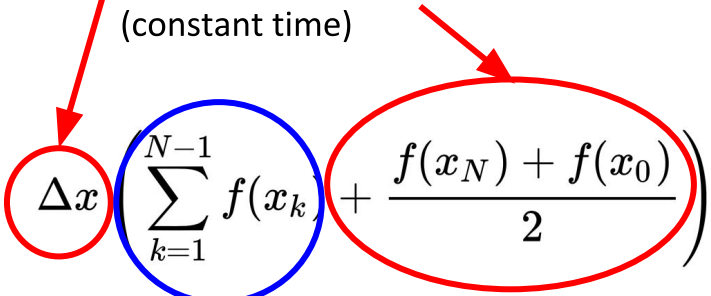
If you choose to use N trapezoids (the more you use, the better the solution), where each one has a width of Δx, then the math can be simplified to the following equation:

$$\Delta x \left(\sum_{k=1}^{N-1} f(x_k) + \frac{f(x_N) + f(x_0)}{2} \right)$$

Example #2: Trapezoidal Rule

The **sequential** solution:

These pieces are one time computations (constant time)

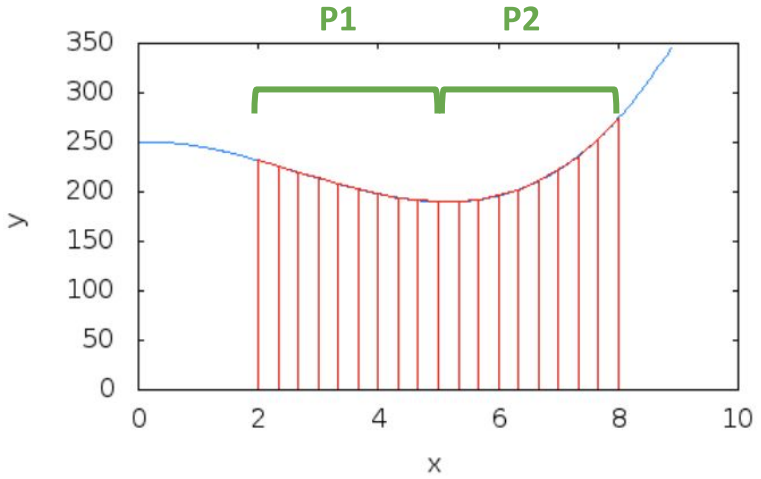


This piece will involve more computation time as the number of trapezoids increases (can be implemented in code with a loop)

Example #2: Trapezoidal Rule

The **parallel** solution:

- Each process can take a **portion of the trapezoids** (*or in terms of the equation we have seen, each process could process their own piece of the summation/loop*)
- Combine the partial sums at the end



Example #3: Odd-Even Transposition Sort

Sorting is also a common task in many applications, and so we ask a similar question here: **If there are a lot of items to sort, can we utilize parallel algorithms to speed up the process?**

We will first look at a simple sequential sorting algorithm, **Bubble Sort**.

Example #3: Odd-Even Transposition Sort

Bubble Sort works by comparing each pair of neighbors, left to right, and swapping them if they are out of order. The name comes from the idea that for each complete pass over all the numbers, the next largest value “bubbles” its way to the top (*or technically the right side of the list of values*)

You can guarantee they will be sorted if you do *N* passes over the numbers, where *N* is the total number of items to be sorted.

6 5 3 1 8 7 2 4

Example #3: Odd-Even Transposition Sort

Odd-Even Transposition Sort is an extension of Bubble Sort, where each phase/pass only compares odd positions with the value to the right of each, or even positions with the value to the right of each, *alternating* between **odd** and **even** each pass.

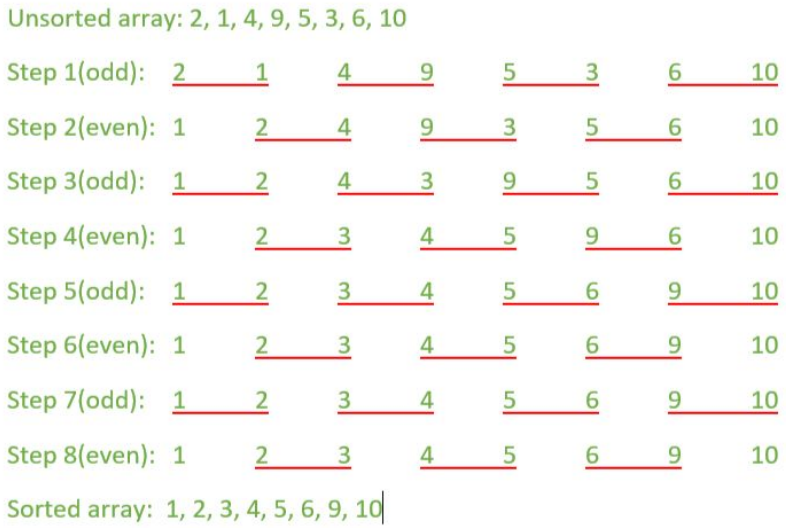
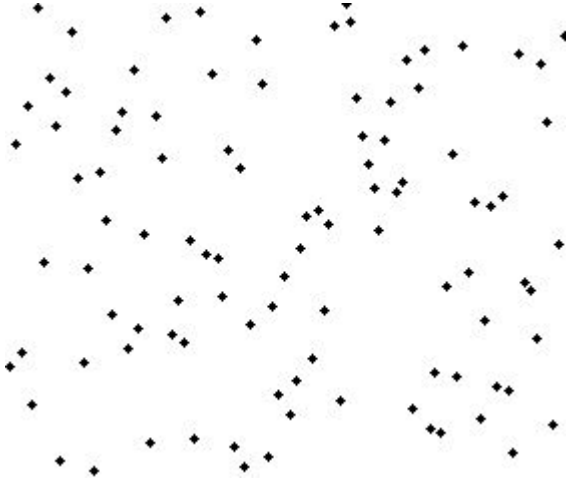


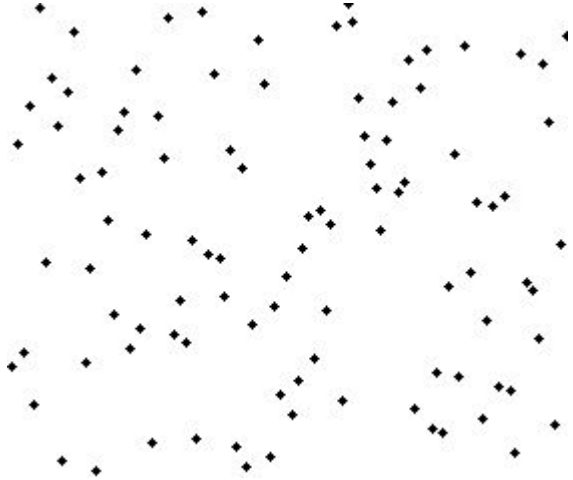
Image from: https://www.geeksforgeeks.org/odd-even-transposition-sort-brick-sort-using-pthreads/

Example #3: Odd-Even Transposition Sort

Visualizing **Bubble Sort**



Visualizing **Odd-Even Transposition Sort**



Example #3: Odd-Even Transposition Sort

Bubble Sort *cannot* be parallelized in a straightforward manner, because issues arise if a process is comparing two values next to another process comparing values

- For example, if **P1** is swapping *positions 3 and 4* while **P2** is swapping *positions 4 and 5*, incorrect results could occur.

Odd Even Transposition Sort solves this by assigning portions of the comparisons during ***each odd/even phase*** to each contributing process, and it is guaranteed that there is no overlapping locations being compared while in the same phase.

Conclusion

You have seen three examples where parallelization can be achieved to increase performance

In general, when coming up with a parallel algorithm, you want to *avoid* situations where **two processes could be using a memory location at the same time**

You will learn that there are ways using code to prevent two processes from using the same memory location simultaneously