

OpenMP

Exercises

OpenMP Tasks

Tasks in OpenMP are code blocks that the compiler wraps up and makes available to be executed in parallel.

```
1 #pragma omp parallel
2 {
3     #pragma omp task
4     printf("hello world from a random thread\n");
5 }
```

Data Sharing Attributes for Tasks

The tasks so far haven't involved any variables. Like other OpenMP constructs, variables that are used inside a task can be specified explicitly with clauses (shared, firstprivate, private, etc.)

OpenMP tasks

Fibonacci Sequence: 1 1 2 3 5 8 13 21 ...

$$f(n) = f(n-2) + f(n-1)$$

Example: fibonnachi_1.c

Exercise 9a

Create parallel program for finding sum of numbers from 0 to N. Create at least two different approaches.

Sequential: `sum.c`

Exercise 9b

1. Create parallel program with two threads:
 - First thread in infinity loop increment shared var and sleep for one second
 - Second thread every 5 seconds output current value of shared var

*Use **sections** for this task*

2. Rewrite program that there are multiple worker thread (worker - thread that increment shared var)

*Use **any** type of directives*

Exercise 9c

Create program for multithreaded Prime number search based on OpenMP.
Output the number of Prime numbers between 1 to N.

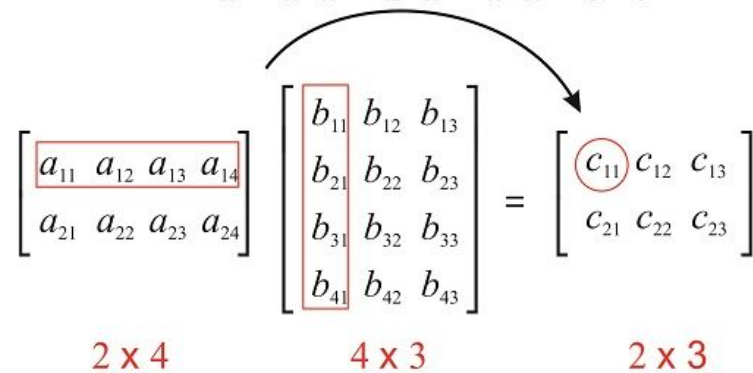
Question:

1. Compare time consummation with different for loop scheduler type.
2. Compare with PThread implementation.

Sample code: `find_prime_number_seq_4.c`

Exercise 9d

Create program for matrix multiplication. First make the sequential version and then parallel with OpenMP.

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

2×4 4×3 2×3

Helpers: `omp_matrix_gen.c`

Exercise 9e

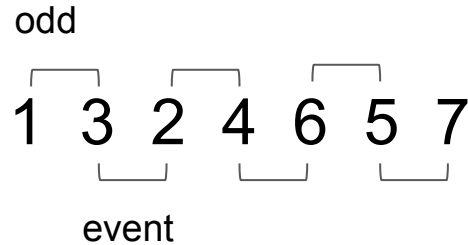
Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent pairs and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

6 5 3 1 8 7 2 4

```
BubbleSort(double A[], int n) {  
    for (i=0; i<n-1; i++)  
        for (j=0; j<n-i; j++)  
            compare exchange(A[j], A[j+1]);  
}
```

Odd even sort

The sorting algorithm introduces two different rules for iterating the method – depending on the even or odd number of the sorting iteration, elements with even or odd indexes are selected for processing, respectively, the comparison of the allocated values is always carried out with their right neighboring elements.



Odd even sort

```
1  OddEvenSort (a[], int n)
2  {
3      for (int i = 0 ; i < n ; i++)
4      {
5          if (i % 2 == 0)
6          {
7              for (int j = 2 ; j < n ; j += 2)
8              {
9                  compare_exchange(a[j-1], a[j])
10             }
11         }
12         else
13         {
14             for (int j = 1 ; j < n ; j += 2)
15             {
16                 compare_exchange(a[j-1], a[j])
17             }
18         }
19     }
20 }
```

Comparisons of value pairs on sorting iterations are independent and can be performed in parallel.

Exercise 9f

Create odd-even version of bubble sort and then transform it to parallel program with help of OpenMP.

Example: bubble_sort.c (seq version)