

1 Thread-Level Parallelism

As powerful as data level parallelization is, it can be quite inflexible, as not all applications have data that can be vectorized. Multithreading, or running a single piece of software on multiple hardware threads, is much more powerful and versatile. OpenMP provides an easy interface for using multithreading within C programs. Some examples of OpenMP directives:

The `parallel` directive indicates that each thread should run a copy of the code within the block. If a for loop is put within the block, **every** thread will run every iteration of the for loop.

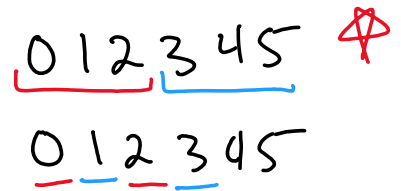
```
#pragma omp parallel {
    ... // threads doing work
}
```



The `parallel for` directive will split up iterations of a for loop over various threads. Every thread will run **different** iterations of the for loop. The following two code snippets are equivalent.

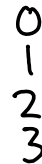
```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    ...
}

#pragma omp parallel {
    #pragma omp for
    for (int i = 0; i < n; i++) { ... }
}
```



There are two functions you can call that may be useful to you:

- `int omp_get_thread_num()` will return the number of the thread executing the code
- `int omp_get_num_threads()` will return the number of total hardware threads executing the code



1.1 For each question below, state and justify whether the program is **sometimes incorrect**, **always incorrect**, **slower than serial**, **faster than serial**, or **none of the above**. Assume the default number of threads is greater than 1. Assume no thread will complete before another thread starts executing. Assume `arr` is an `int[]` of length `n`.

```
(a) // Set element i of arr to i
    #pragma omp parallel
    {
        for (int i = 0; i < n; i++)
            arr[i] = i;
    }
```

correct incorrect
faster slower always sometimes



0 1 2 3 4 5 6 7
└─┘ └─┘ └─┘ └─┘

```
(b) // Set arr to be an array of Fibonacci numbers.
arr[0] = 0;
arr[1] = 1;
#pragma omp parallel for
for (int i = 2; i < n; i++)
    arr[i] = arr[i-1] + arr[i - 2];
```

Correct ~~incorrect~~
always sometimes

```
(c) // Set all elements in arr to 0;
int i;
#pragma omp parallel for
for (i = 0; i < n; i++)
    arr[i] = 0;
```

Correct Incorrect
faster

1.2 What potential issue can arise from this code?

```
1 // Decrements element i of arr. n is a multiple of omp_get_num_threads()
2 #pragma omp parallel
3 {
4 int threadCount = omp_get_num_threads();
5 int myThread = omp_get_thread_num();
6 for (int i = 0; i < n; i++) {
7     if (i % threadCount == myThread) arr[i] -= arr[i];
8 }
9 }
```

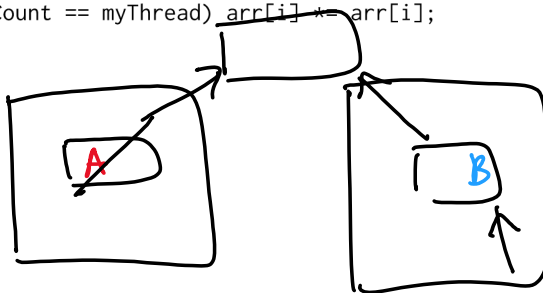


4
0 1 2 3

False sharing

0 1 2 3 4 5
└─┘ └─┘

Block



```
1.3 // Assume n holds the length of arr
2 double fast_product(double *arr, int n) {
3     double product = 1;
4     #pragma omp parallel for
5     for (i = 0; i < n; i++) {
6         product *= arr[i];
7     }
8     return product;
9 }
```

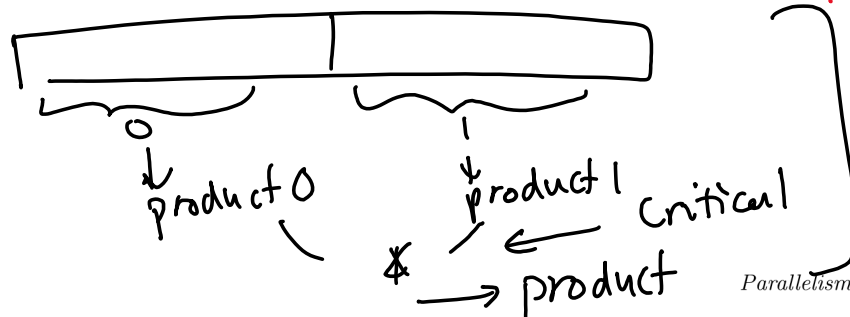
reduction (*: product)
 #pragma omp critical

1 1
 x x
 2 3
 2 3
 product = 3

Data Race

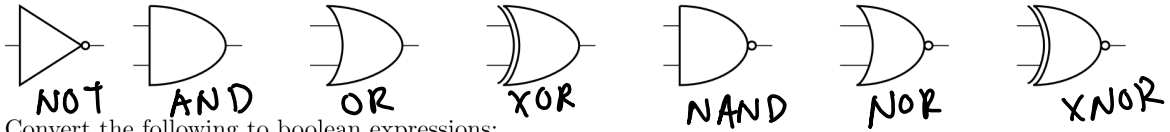
- (a) What is wrong with this code?
- (b) Fix the code using #pragma omp critical
- (c) Fix the code using #pragma omp reduction(operation: var)

- 1) read product
- 2) read arr[i]
- 3) execute
- 4) store back to product



2 Logic Gates

2.1 Label the following logic gates:



2.2 Convert the following to boolean expressions:

(a) NAND $\bar{A}B + A\bar{B} + \bar{A}\bar{B}$

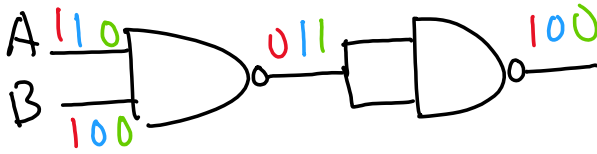
(b) XOR $\bar{A}B + A\bar{B}$

(c) XNOR $AB + \bar{A}\bar{B}$

AND : AB

OR : $A+B$

2.3 Create an AND gate using only NAND gates.



2.4 How many different two-input logic gates can there be? How many n-input logic gates?