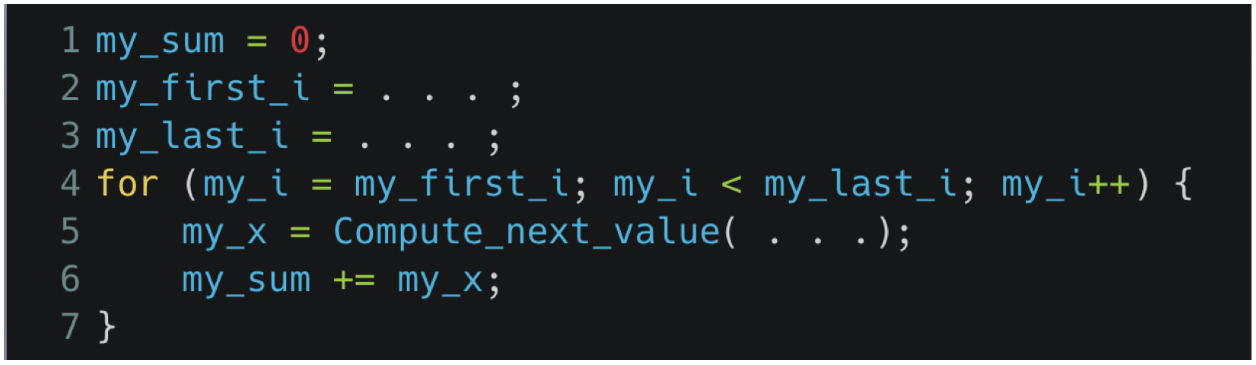
Assignment 1

Due: 11:59 PM, February 20

Solve Problem 1.1, 1.3, and 1.4 in the textbook (1st edition).

1.1 Devise formulas for the functions that calculate `my_first_i` and `my_last_i` in the global sum example. Remember that each core should be assigned roughly the same number of elements of computations in the loop. *Hint*: First consider the case when n is evenly divisible by p.



```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
    my_x = Compute_next_value( . . .);
    my_sum += my_x;
}
```

1.3 Try to write pseudo-code for the tree-structured global sum illustrated in Figure 1.1. Assume the number of cores is a power of two (1, 2, 4, 8, . . .).

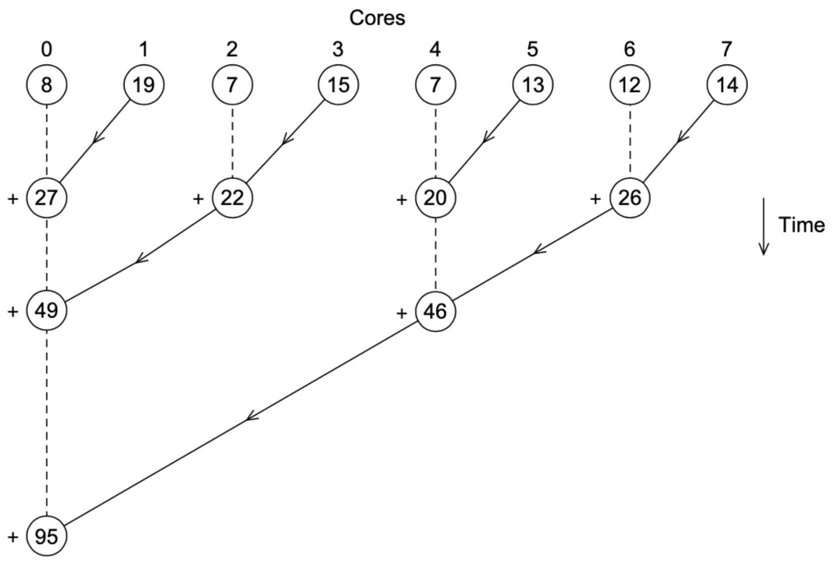


FIGURE 1.1

Multiple cores forming a global sum

Hints: Use a variable `divisor` to determine whether a core should send its sum or receive and add. The `divisor` should start with the value 2 and be doubled after each iteration. Also use a variable `core_difference` to determine which core should be partnered with the current core. It should start with the value 1 and also be doubled after each iteration. For example, in the first iteration `0 % divisor = 0` and `1 % divisor = 1`, so 0 receives and adds, while 1 sends. Also in the first iteration `0 + core_difference = 1` and `1 − core_difference = 0`, so 0 and 1 are paired in the first iteration.

1.4 As an alternative to the approach outlined in the preceding problem, we can use C's bitwise operators to implement the tree-structured global sum. In order to see how this works, it helps to write down the binary (base 2) representation of each of the core ranks, and note the pairings during each stage:

	Stages		
Cores	*1*	*2*	*3*
$0_{10} = 000_2$	$1_{10} = 00\boxed{1}_2$	$2_{10} = 010_2$	$4_{10} = 100_2$
$1_{10} = 001_2$	$0_{10} = 00\boxed{0}_2$	×	×
$2_{10} = 010_2$	$3_{10} = 011_2$	$0_{10} = 000_2$	×
$3_{10} = 011_2$	$2_{10} = 010_2$	×	×
$4_{10} = 100_2$	$5_{10} = 101_2$	$6_{10} = 110_2$	$0_{10} = 000_2$
$5_{10} = 101_2$	$4_{10} = 100_2$	×	×
$6_{10} = 110_2$	$7_{10} = 111_2$	$4_{10} = 100_2$	×
$7_{10} = 111_2$	$6_{10} = 110_2$	×	×

0 shift   1 shift   2 shift

From the table we see that during the first stage each core is paired with the core whose rank differs in the rightmost or first bit. During the second stage cores that continue are paired with the core whose rank differs in the second bit, and during the third stage cores are paired with the core whose rank differs in the third bit. Thus, if we have a binary value `bitmask` that is 001_2 for the first stage, 010_2 for the second, and 100_2 for the third, we can get the rank of the core we're paired with by "inverting" the bit in our rank that is nonzero in `bitmask`. This can be done using the bitwise exclusive or ∧ operator.

Implement this algorithm in pseudo-code using the bitwise exclusive or and the left-shift operator.