NEED TO WRITE PARALLEL PROGRAMS

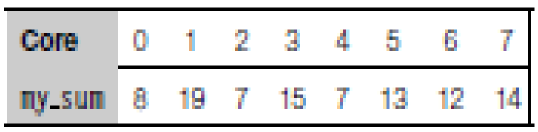
- Most programs that have been written for conventional, single-core systems cannot exploit the presence of multiple cores. We can run multiple instances of a programon a multicore system, but this is often of little help.
- As an example, suppose that we need to compute n values and add them together.We know that this can be done with the following serial code:

```
sum = 0;
for (i = 0; i < n; i++)
{
        x = Compute next value(. . .);
        sum += x;
}
```

- Now suppose we also have p cores and p is much smaller than n. Then each core canform a partial sum of approximately n=p values:

```
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;
        }
```

- Here the prefix my indicates that each core is using its own, private variables, andeach core can execute this block of code independently of the other cores.After each core completes execution of this code, its variable my-sum will storethe sum of the values computed by its calls to Compute_ next_ value.
- For example,if there are eight cores, n =24, and the 24 calls to Compute-next- value return thevalues
  1, 4, 3, 9, 2, 8, 5, 1, 1, 6, 2, 7, 2, 5, 0, 4, 1, 8, 6, 5, 1, 2, 3, 9,
  then the values stored in my sum might be



Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

- Here we’re assuming the cores are identified by nonnegative integers in the range0, 1, : : : ,p-1, where p is the number of cores.

- When the cores are done computing their values of my sum, they can form a

global sum by sending their results to a designated “master” core, which can addtheir results:

```
if (I’m the master core)
        {
        sum = my- x;
        for each core other than myself
        {
        receive value from core;
        sum += value;
        }
}
 Else
 {

        send my x to the master;
}
```

In our example, if the master core is core 0, it would add the values

8+19+7+15+7+13+12+14= 95.

- But you can probably see a better way to do this—especially if the number ofcores is large. Instead of making the master core do all the work of computing thefinal sum, we can pair the cores so that while core 0 adds in the result of core 1, core2 can add in the result of core 3, core 4 can add in the result of core 5 and so on. Thenwe can repeat the process with only the even-ranked cores: 0 adds in the result of 2,4 adds in the result of 6, and so on.
- Now cores divisible by 4 repeat the process, andso on. The circles contain the current value of each core’s sum, andthe lines with arrows indicate that one core is sending its sum to another core. Theplus signs indicate that a core is receiving a sum from another core and adding thereceived sum into its own sum.
- For both “global” sums, the master core (core 0) does more work than any othercore, and the length of time it takes the program to complete the final sum shouldbe the length of time it takes for the master to complete.

- However, with eight cores,the master will carry out seven receives and adds using the first method, while withthe second method it will only carry out three. So the second method results in animprovement of more than a factor of two.
- The difference becomes much more dramatic with large numbers of cores. With 1000 cores, the first method will require 999 receives and adds, while the second will only require 10, an improvement of almost a factor of 100!

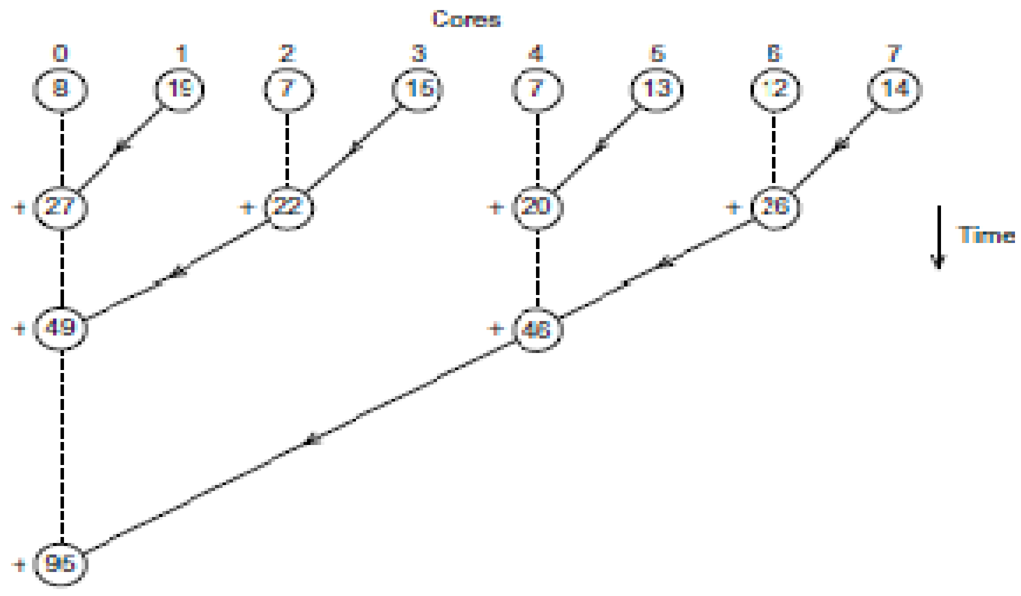


Fig : Multiple cores forming a global sum