* 1. p: number of cores  
     n: number of elements  
     int quotient = n / p  
     int remainder = n % p  
     if (my\_core\_number < remainder) {  
      my\_n\_count = quotient + 1  
      my\_first\_i: my\_n\_count \* my\_core\_number  
     } else {  
      my\_n\_count = quotient  
      my\_first\_i = my\_core\_number \* my\_n\_count + remainder

}  
my\_last\_i: my\_first\_i + my\_n\_count

* 1. example:   
      if n = 12, number of cores = 4  
      instead of giving :  
      core 0: i = 0 (2 ms), 1 (4 ms), 2 (6 ms)   
      core 1: i = 3 (8 ms), 4 (10 ms), 5 (12 ms)  
      …  
      we can divide the work more efficiently by giving:  
      core 0: i = 0, 3, 6, 9  
      core 1: i = 1, 4, 7, 10  
      …  
     /\* assign[c][j] is the jth value of i assigned to core c \*/  
     /\* work[c] is the total amount of work assigned to core c \*/  
     c = j = 0;  
     for (i = 0, i < n; i++) {  
      work [c] = 2 \* (i + 1);  
      assign[c][j] = i;  
      c = (c + 1) % p;  
      if (c == 0) j++;  
     }
  2. divisor = 2  
     core\_difference = 1  
     sum = my\_value  
     while (divisor <= p) {  
      if (my\_rank % divisor == 0) {  
      partner = my\_rank + core\_difference  
      receive data from partner  
      sum += received data  
      } else {  
      partner = my\_rank – core\_difference  
      send data to partner  
      return  
      }  
      divisor \*= 2  
      core\_difference \*= 2  
     }
  3. divisor = 2  
     bitmask = 1  
     sum = my\_value  
     while (bitmask < p) {  
      partner = myrank ^ bitmask  
      if (my\_rank % divisor == 0) {   
      receive data from partner  
      sum += received data  
      } else {   
      send data to partner  
      return  
      }  
      divisor <<= 2  
      bitmask <<= 1  
     }
  4. divisor = 2  
     core\_difference = 1  
     sum = my\_value  
     while (divisor <= p) {  
      if (my\_rank % divisor == 0) {  
      partner = my\_rank + core\_difference  
      if ( partner < p ) {  
      receive data from partner  
      sum += received data  
      }  
      } else {  
      partner = my\_rank – core\_difference  
      send data to partner  
      return  
      }  
      divisor \*= 2  
      core\_difference \*= 2  
     }
  5. original: nb\_of\_cores – 1

tree structured: log2(nb\_of\_cores)

|  |  |  |
| --- | --- | --- |
| Receives/addition (p) (cores) | Original | Tree structured |
| 2 | 1 | 1 |
| 4 | 3 | 2 |
| 8 | 7 | 3 |
| … | … | … |
| 1024 | 1023 | 10 |

* 1. The tree structured program is an example of both task and data parallelism.

Since in each phase:

* each core adds its assigned computed values which is called as data-parallelism,
* while some cores have to send their assigned computed values to other cores which is called as task parallelism.
  1. a) making posters, cleaning the place, bringing food, …  
     b) we can assign each faculty a portion of the job  
     c) for task parallelism, we can partition the task of preparing food and drinks among the faculty. For data parallelism, we can partition this group according to the type of food (some can do the grilling while others can do the frying, …).