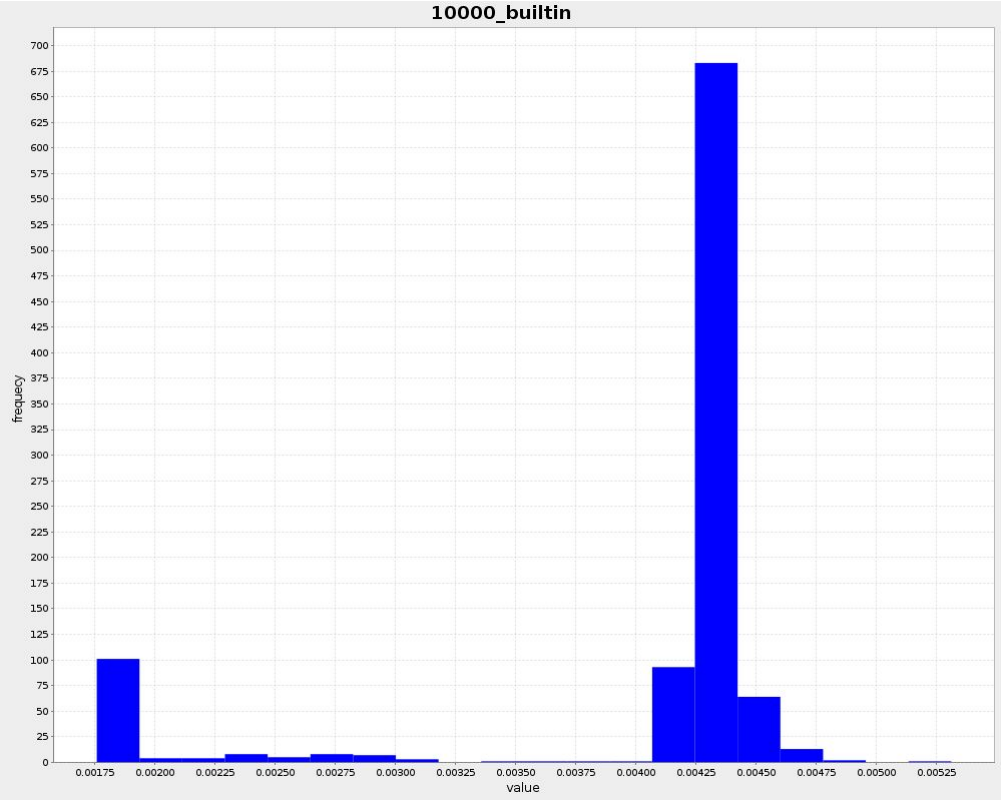
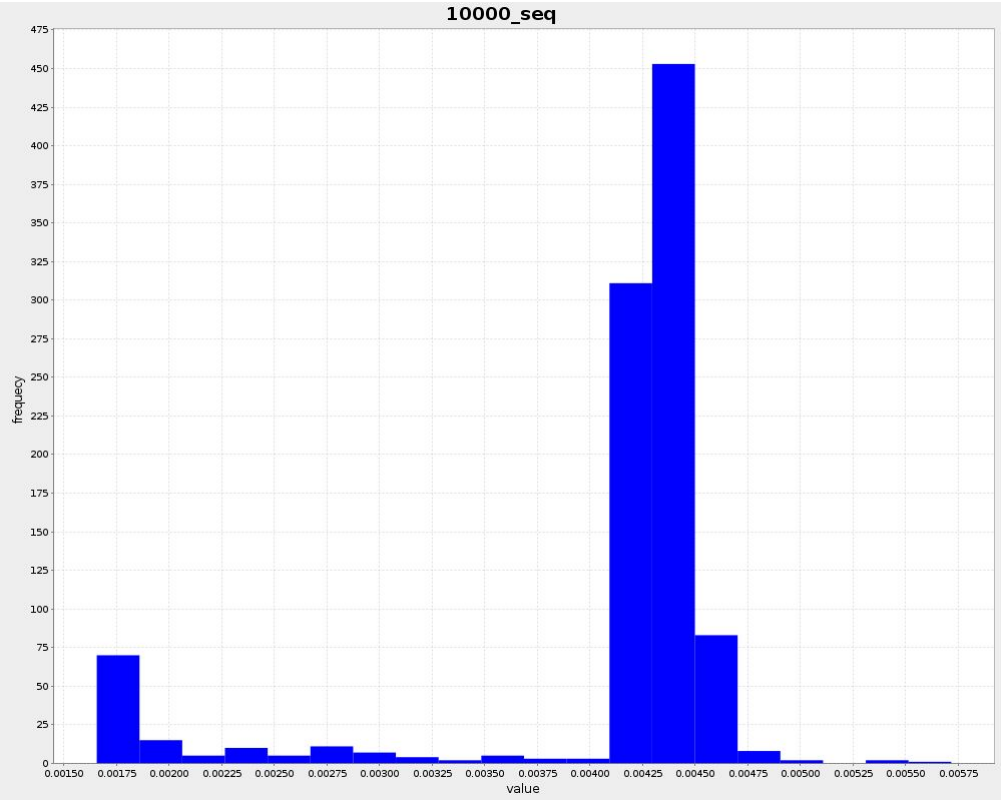
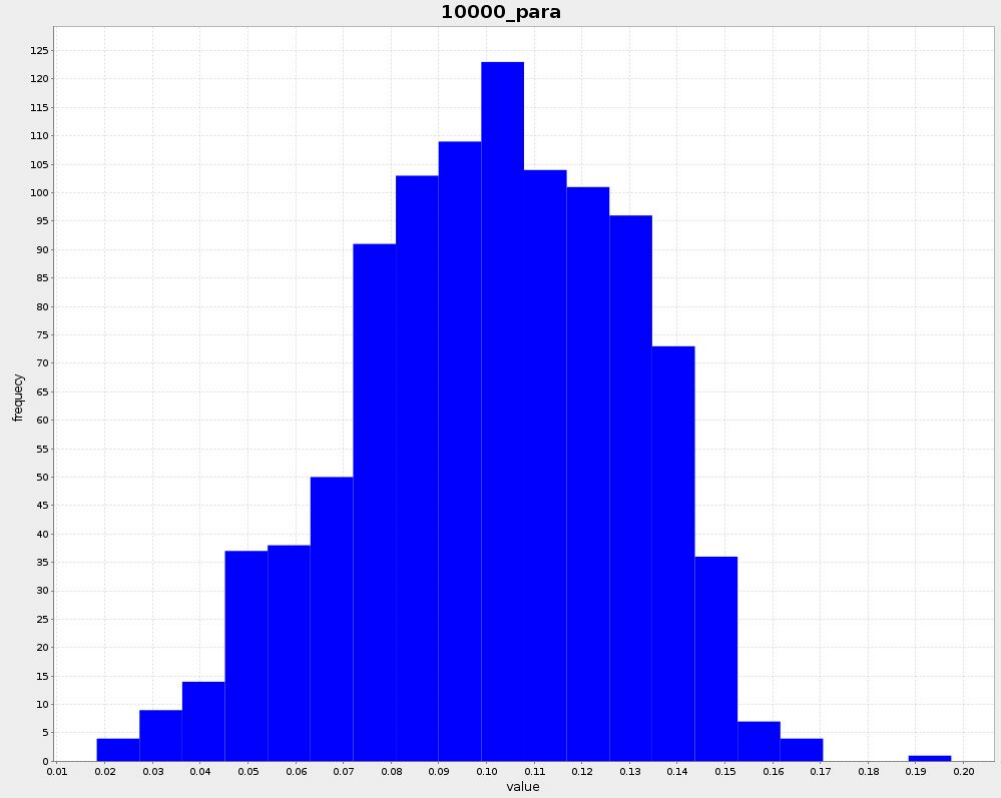
**Distribution Study**

For better understanding, firstly, on my laptop (intel i5, 2 cores), I've tested 1000 samples of each algorithm with different size of arrays and studied their distribution of execution time. When the size of the array is less than 10000, for the libc function and the sequential implementation, their distributions are generally centralized, but I found peaks on smaller time, as the following graphs:

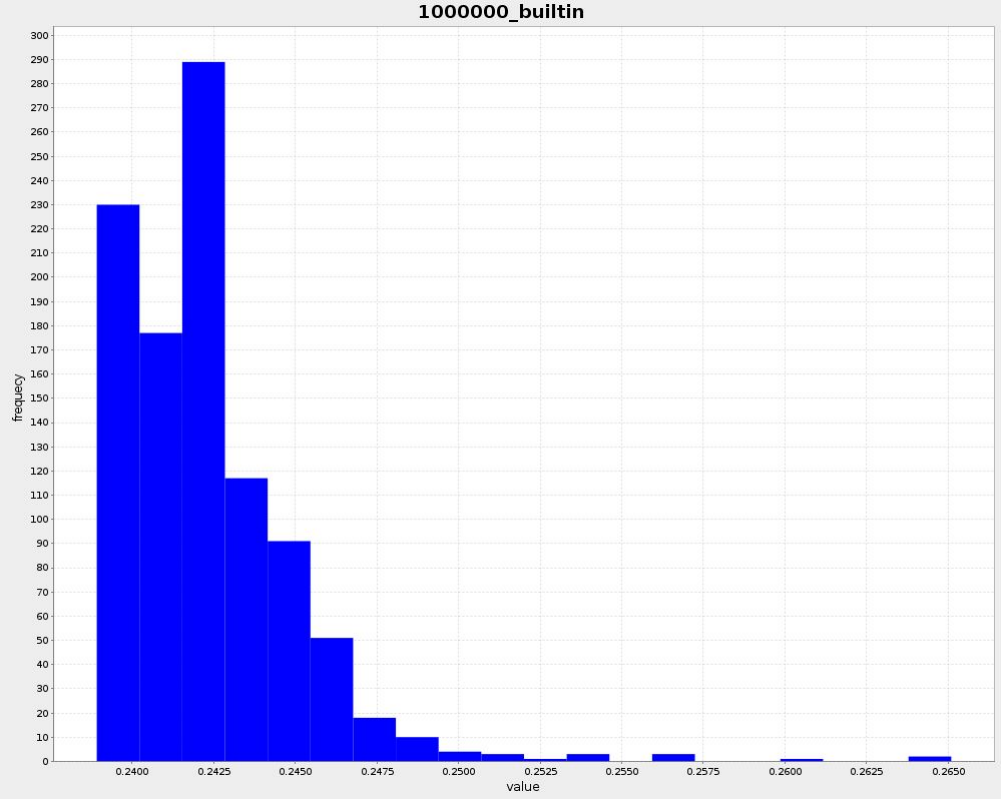


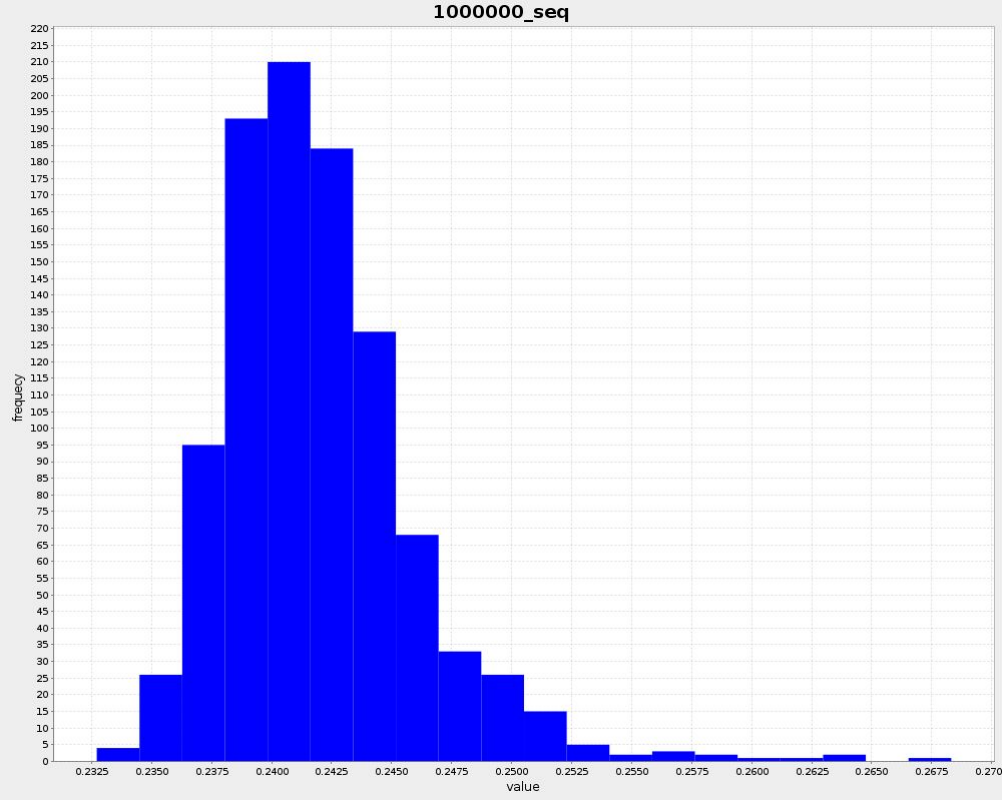



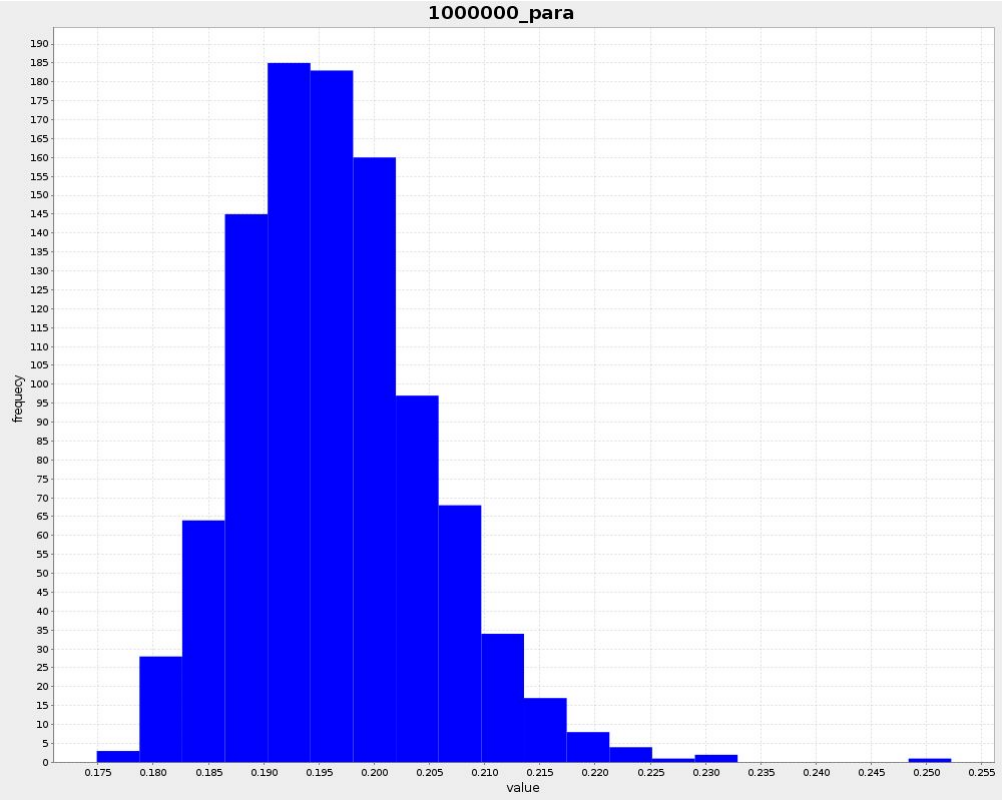
Different with the above ones, the distribution of parallel implementation generally follows normal distribution. As the following :




With bigger array, the trend of distribution is more clear, but the peak could still be found, as the following:





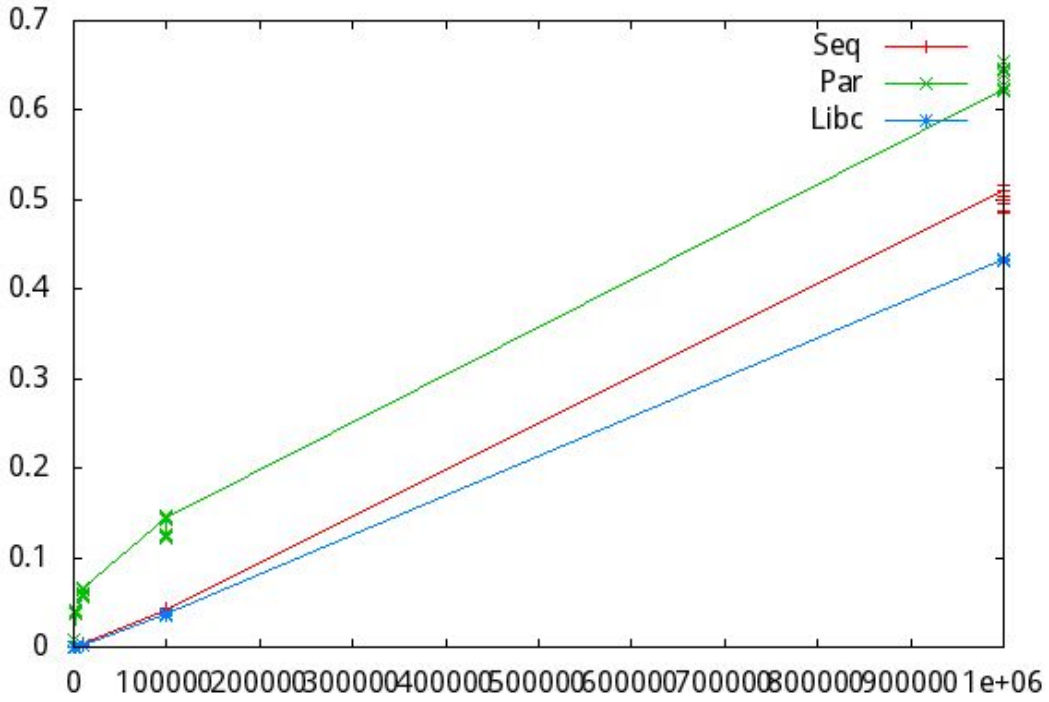




I’ve drawn these graphs with jfreechart library of java, to my point of view, this could be easier to control...

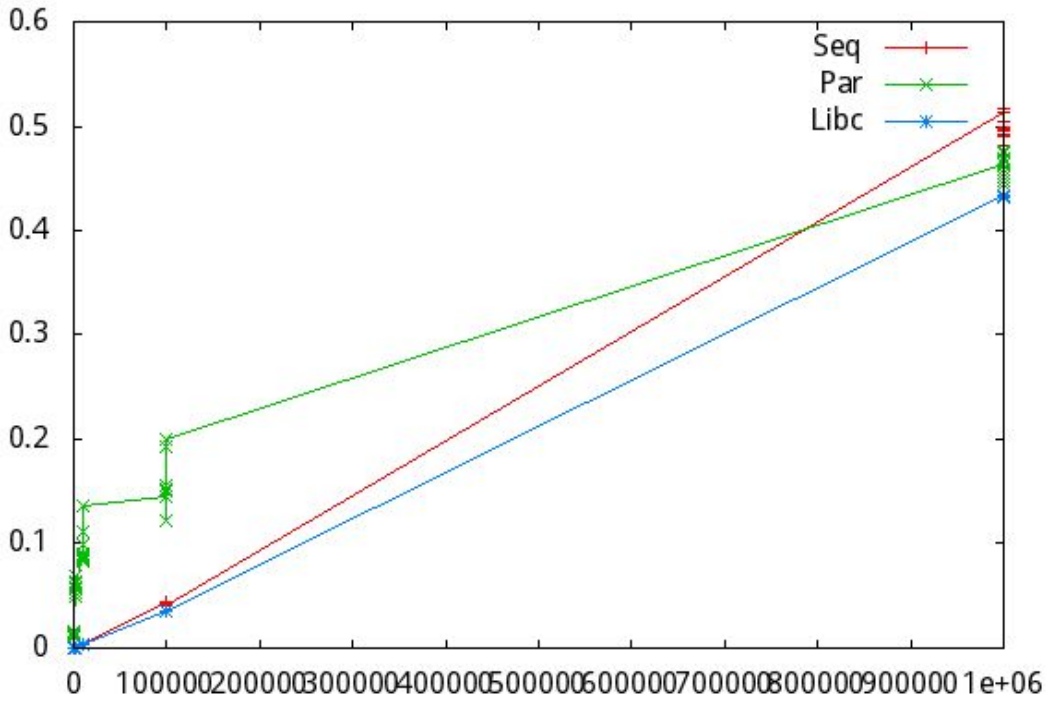
**G5k**

Then I've executed on G5K, I've tested with a different host number and core number, I've got the result as the following :

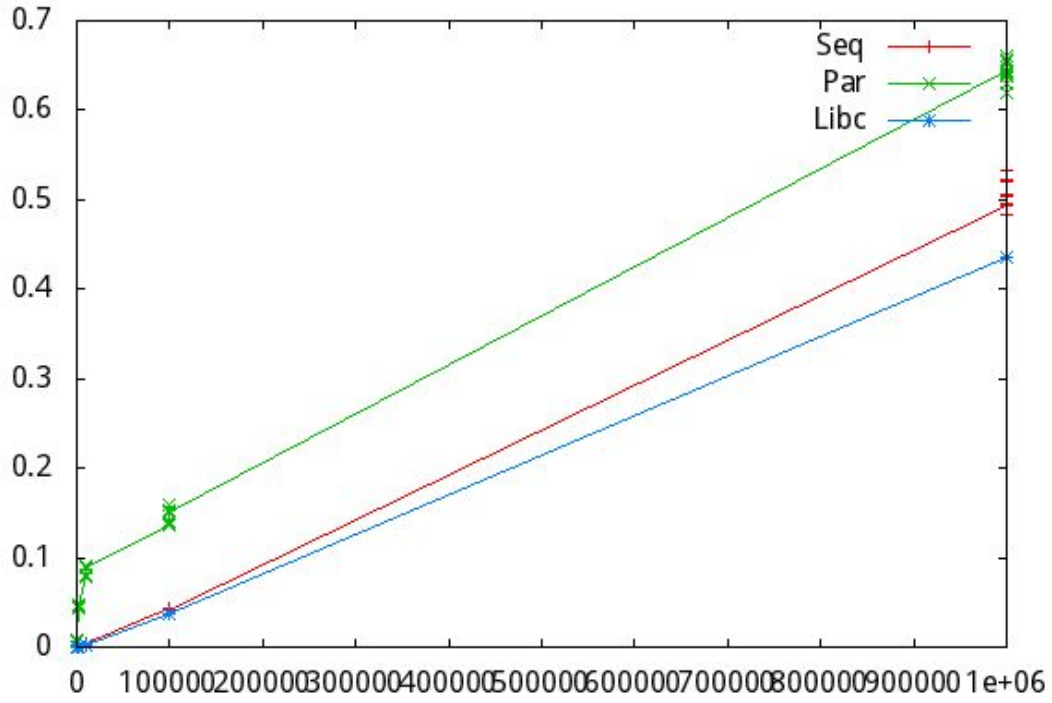
**Host1core1** (*oarsub -I -l **host=1/core=1***)



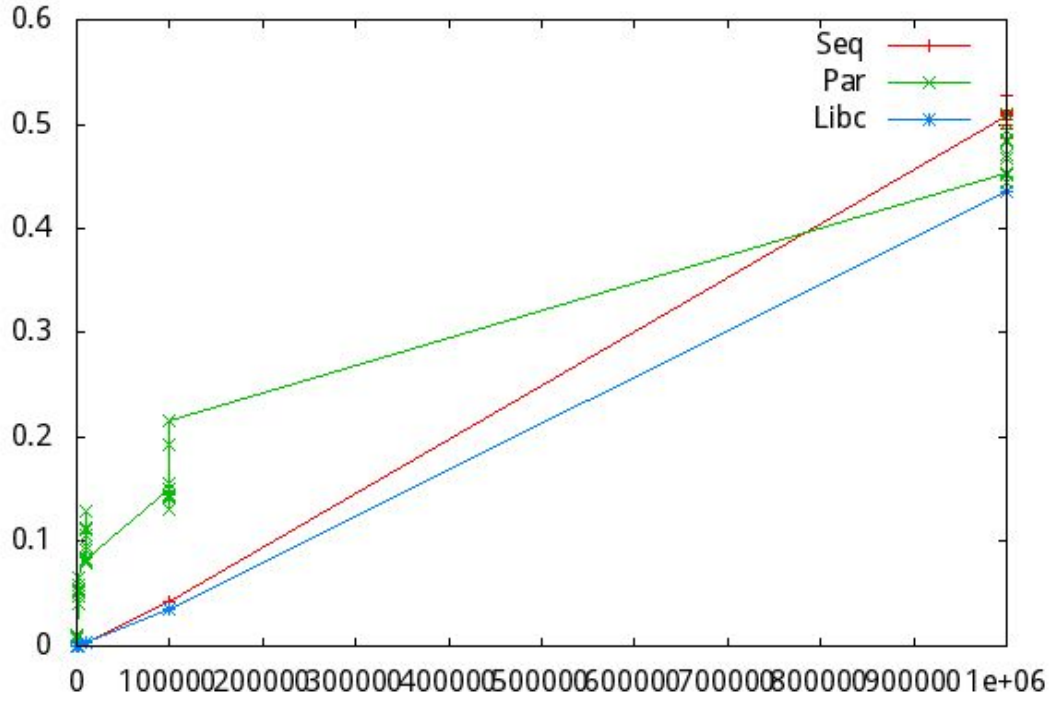

**Host1core2** (*oarsub -I -l **host=1/core=2***)



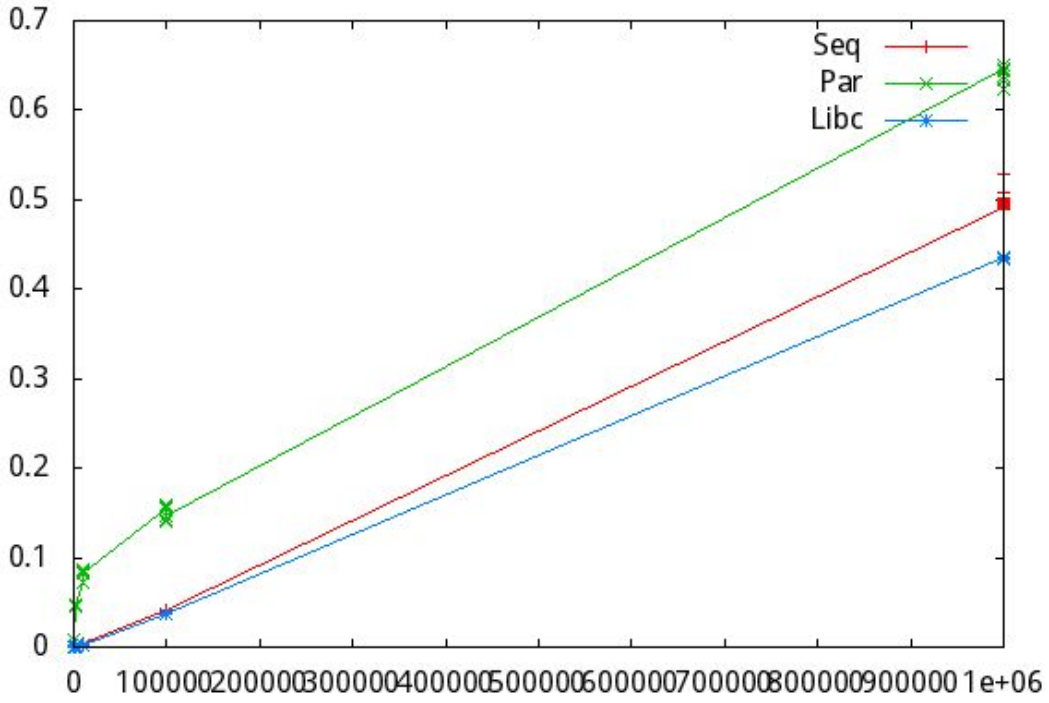
**Host4core1**



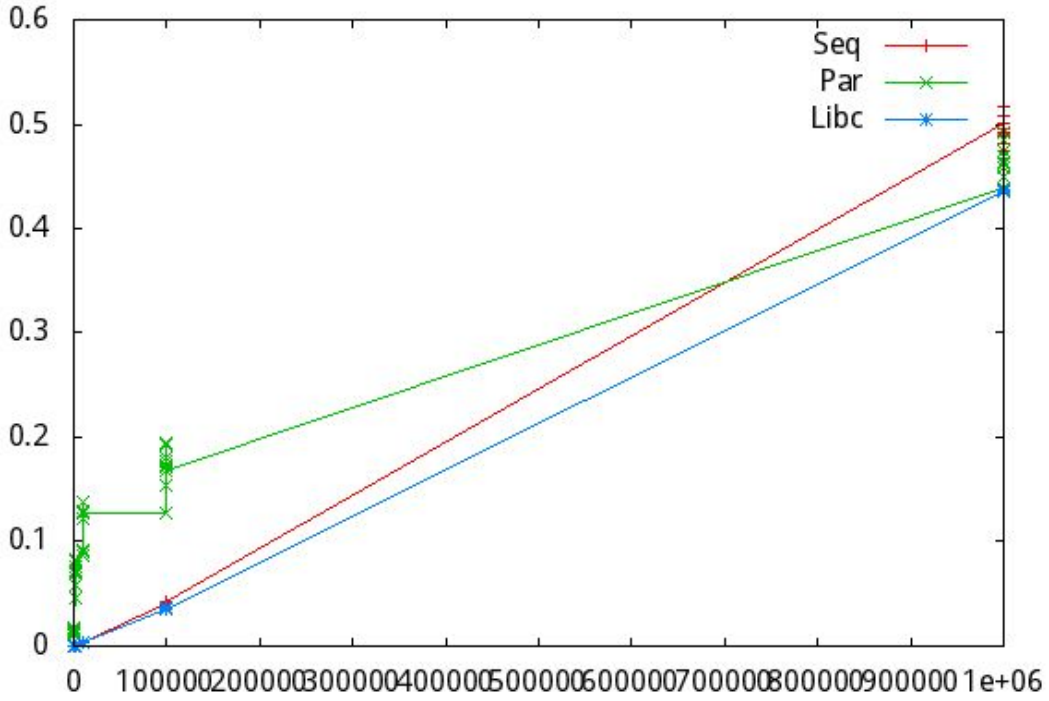

**Host4core2**



**Host10core1**



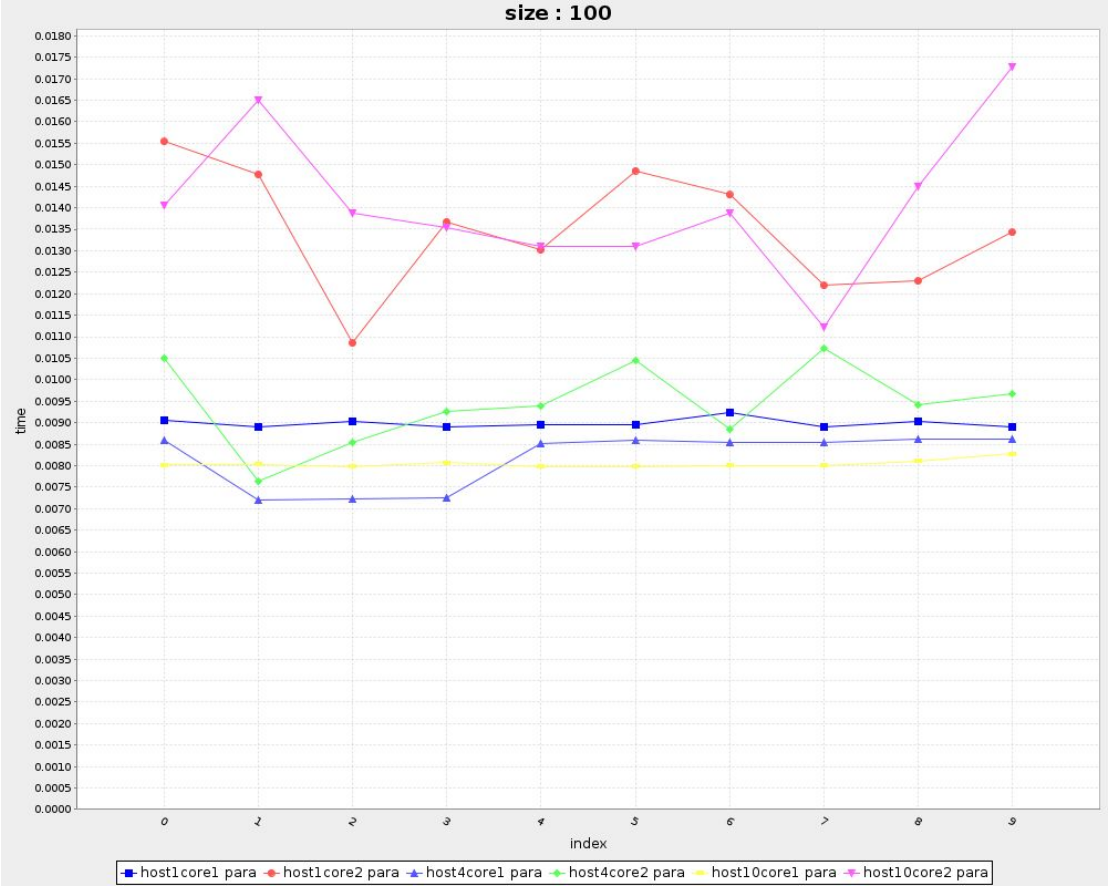

**Host10core2**



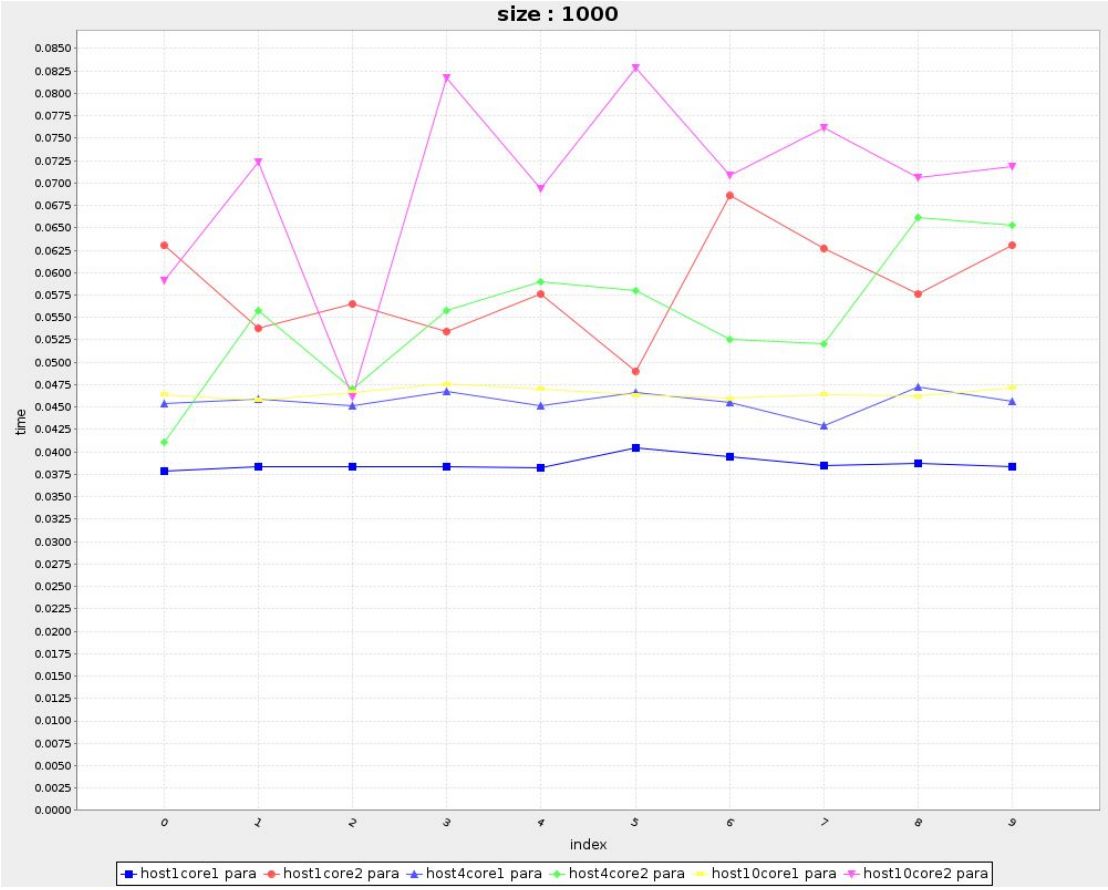

It could be found that with bigger array size, the performance of parallel implementation is relatively better. When we increase the core number the performance is better than to increase the host number.

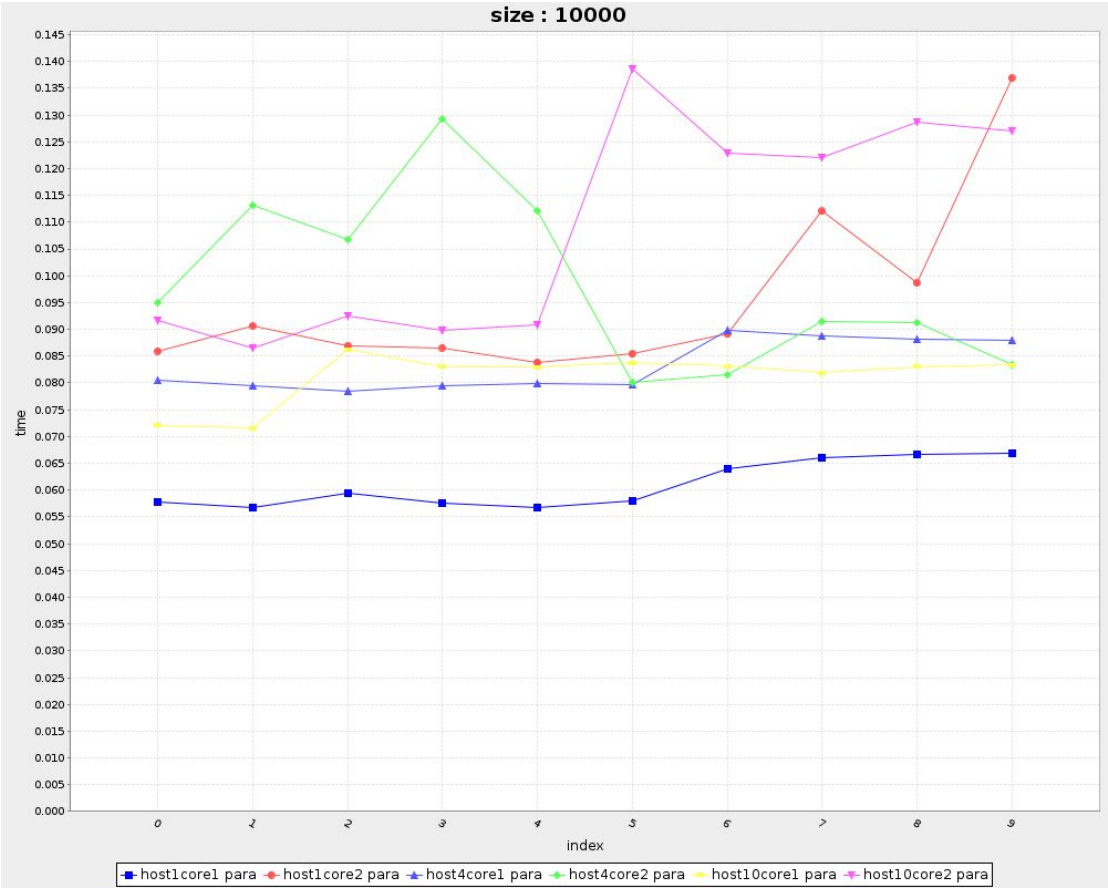
**parallel implementation compare**

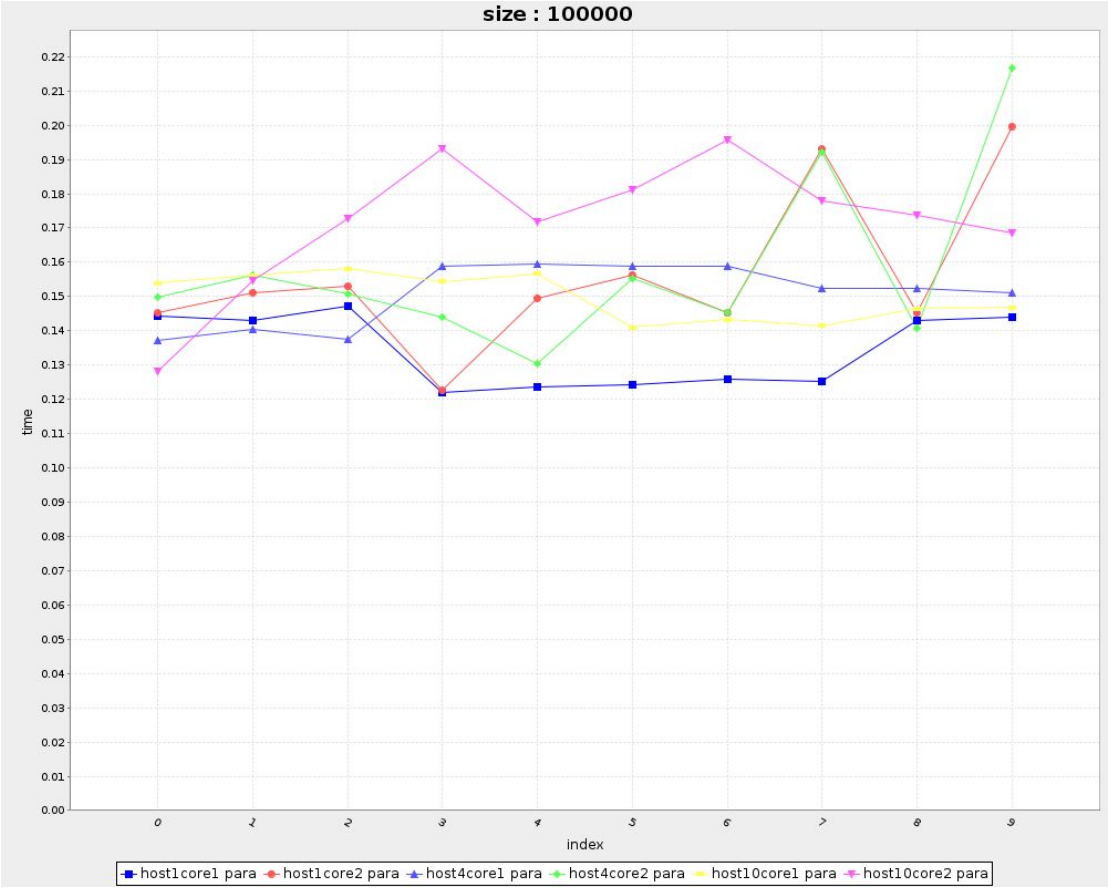
Then I've compared the performance of parallel implementation on different environments, x-axis means the index of test, y-axis is the time of execution, the environments of each color of the lines are shown in the bottom of the graph, and the array size is in the top of the graph, as the following :



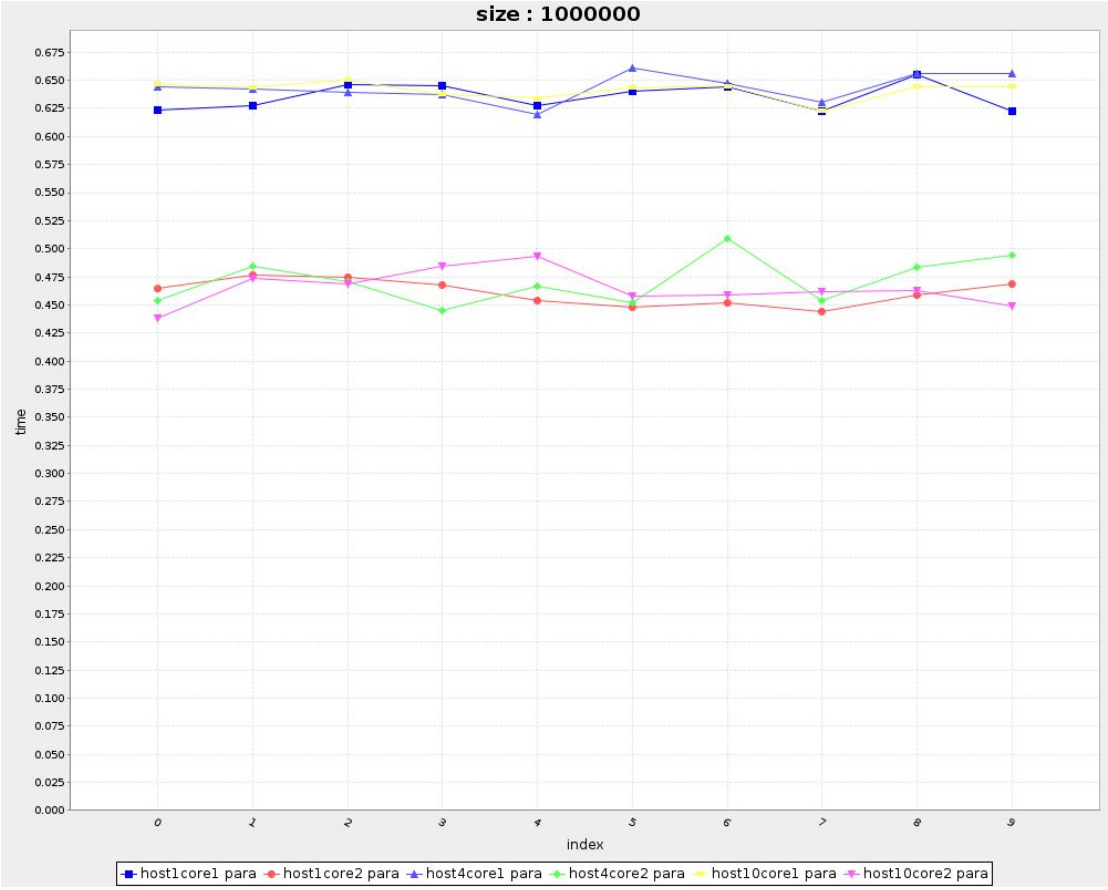

For array size 100, host4core1 and host10core1 are with the best performance.


size : 1000
time
index
host1core1 para
host1core2 para
host4core1 para
host4core2 para
host10core1 para
host10core2 para


size : 10000
time
index
host1core1 para
host1core2 para
host4core1 para
host4core2 para
host10core1 para
host10core2 para




For array size 1000, 10000 and 100000, the graphs show us host1core1 performs the best.




For array size 1000000, host1core2, host4core2 and host10core2 get better performance. It seems that the multicore environment gets advantage when the array size is big enough.

**Average execution time**

The average execution times are as the following:

size : 100

| | |
|---|---|
| host1core1 | **0.0089868** |
| host1core2 | 0.0134968 |
| host4core1 | **0.0081727** |
| host4core2 | **0.0094407** |
| host10core1 | **0.0080471** |
| host10core2 | 0.014098100000000002 |

size : 1000

| | |
|---|---|
| host1core1 | **0.0386717** |
| host1core2 | 0.058543 |
| host4core1 | **0.0456303** |
| host4core2 | 0.055259899999999994 |
| host10core1 | **0.046554** |
| host10core2 | 0.0701014 |

size : 10000

| | |
|---|---|
| host1core1 | **0.0609802** |
| host1core2 | 0.09563569999999999 |
| host4core1 | 0.08323289999999998 |
| host4core2 | 0.098415 |
| host10core1 | 0.08111160000000002 |
| host10core2 | 0.1090425 |

size : 100000

| | |
|---|---|
| host1core1 | **0.13425800000000002** |
| host1core2 | 0.1561003 |
| host4core1 | 0.1507127 |
| host4core2 | 0.1581778 |
| host10core1 | 0.14984 |
| host10core2 | 0.17169629999999997 |

size : 1000000

| | |
|---|---|
| host1core1 | 0.6354127999999999 |
| host1core2 | **0.460987** |
| host4core1 | 0.6433924 |
| host4core2 | **0.471392** |
| host10core1 | 0.6411336 |
| host10core2 | **0.4648397** |

This shows us again the result we get from the graphs, and I found that in this experiment, I didn't get a better performance by increasing host number.