This low level design is common for both method1 and method2

We are having size of array=2^n and number of threads=2^p we will receive n and p from input file,we will assign values to array using random generator,we are having 2^p segments each of size 2^(n-p)

Note : starting position of segment(k) in array will be k*(2^(n-p))
By knowing these starting positions of segments in array and we will also know the list of segments present at every point of time so we can perform sort and merge operations on segments

We will create 2^(p) threads i.e thread0,thread1,.....thread(2^(p)-1)
Thread0 will sort segment0
Thread1 will sort segment1
Thread2 will sort segment2

Thread(2^(p)-1) will sort segment(2^(p)-1)
After finishing their jobs all threads will exit

Continuation of Low level design for method1

We are having segment0,segment1,.....segment((2^p)-1)
Here merging will be done in phases

In phase 1

Segment0 and segment1 will be merged, so segment0 will include both Segment0 and segment1 ,and segment1 will be removed
Now we will have segment0, segment2,.....segment(2^p-1)

In phase 2

Segment0 and segment2 will be merged ,so segment0 will include both Segment0 and segment2 ,and segment2 will be removed
Now We will have segment0,segment3,.....segment(2^p-1)

In phase 2^p-2

Segment0 and segment(2^p-2) will be merged ,so segment0 will include both Segment0 and segment(2^p-2) ,and segment(2^p-2) will be removed
Now We will have segment0,segment(2^p-1)

In phase 2^p-1

Segment0 and segment(2^p-1) will be merged ,so segment0 will include both Segment0 and segment(2^p-1) ,and segment(2^p-1) will be removed

Now We will have segment0 and whole array will be sorted

Note : in every phase we can merge segments because we can find starting location of each segment in an array

Continuation of low level design for method2

Here also merging will be done in phases

Note : in every phase Number of threads which we will create is equal to number of segments/2

In phase1

we will create thread0,thread2,thread4.....thread((2^p)-2)

Thread0 will merge segment0 and segment1

Thread2 will merge segment2 and segment3

……..

thread(2^(p)-2) will merge segment((2^p)-2) and segment((2^p)-1)

After all threads completing their job we will have segment0,segment2, segment4,.....,segment((2^p)-2),so here total number segments will be reduced by half

All threads will exit

In phase 2

we will create thread0,thread4,thread8.....thread((2^p)-4)

Thread0 will merge segment0 and segment2

Thread4 will merge segment4 and segment6

thread((2^p)-4) will merge segment((2^p)-4) and segment((2^p)-2)

After all threads completing their job we will have segment0 segment4,.....,segment((2^p)-4),so here total number segments will be reduced by half

All threads will exit

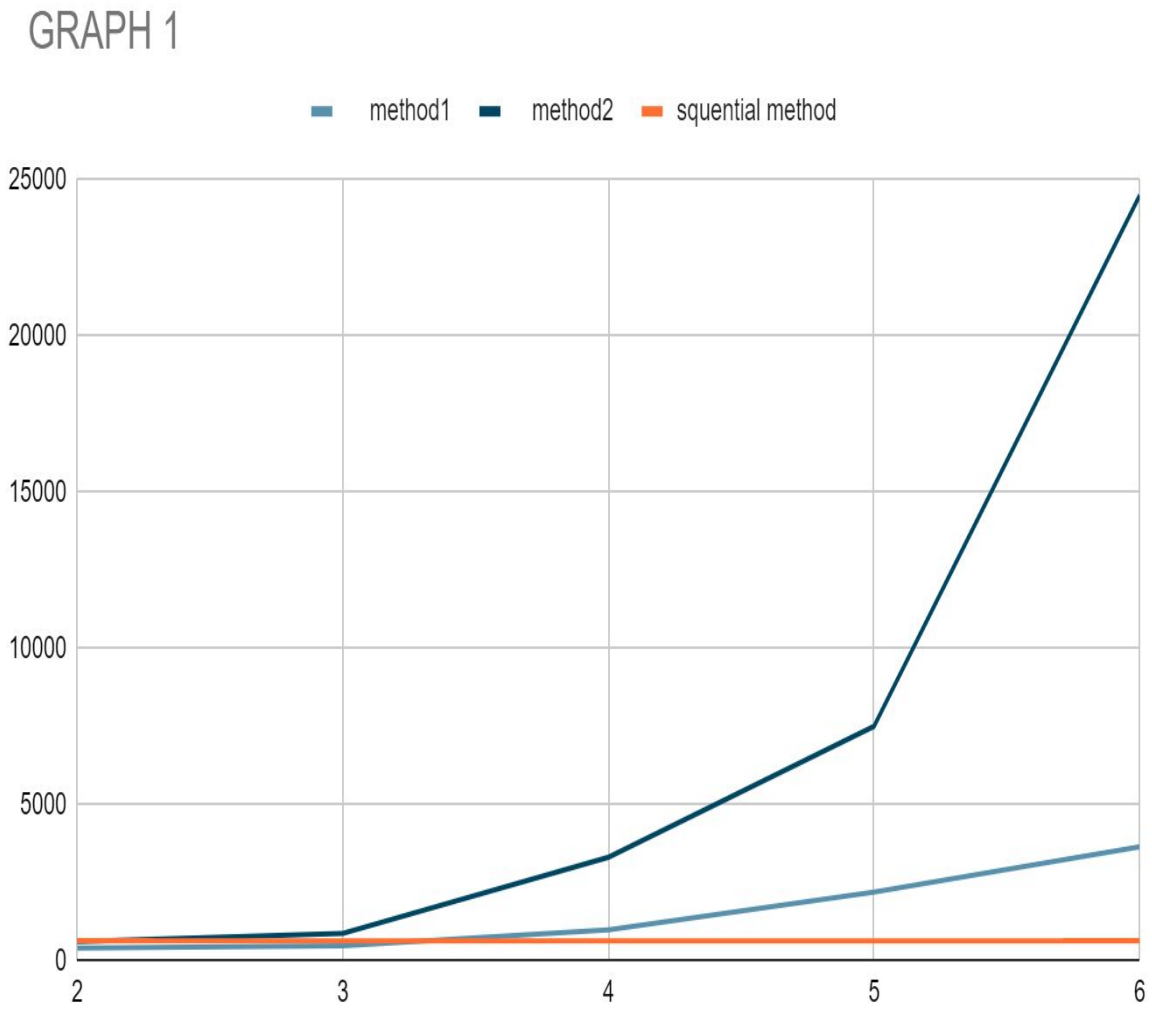
In phase 3

we will create thread0,thread8.....thread((2^p)-8)
Thread0 will merge segment0 and segment4
Thread8 will merge segment8 and segment12
thread((2^p)-8) will merge segment((2^p)-8) and segment((2^p)-4)
After all threads completing their job we will have segment0,
segment8,.....,segment((2^p)-8),so here total number segments will be reduced by half
All threads will exit

In some phase y where number of segments will be 2

We will create thread0
Thread0 will merge segment0 and segment(2^(p-1))
After thread0 completing its job we will left with only segment0
After completing this phase y whole array will be sorted
All threads will exit

Analysis of Graph 1



Fixing parameter n=12,and p varies from 2 to 6
X-axis is for parameter p
Y-axis is for time taken for sorting

Sequential method vs method1

Consider some point k which is in between 3 and 4
According above graph k=3.3 for method1
There is no use if (number of threads created)>(number of cores)
Because there will be same amount of parallelism and more time will be taken for context switching and thread creation so it is not good to create more threads than number of cores

I am having only 2 cores, i will save same amount of time by parallelism when p=2,3,4,5,6, and overhead of thread creation and context switches will keep increasing because we are increasing number of threads

From p= 2 to p= k
Amount of time saved by parallelism will be more than overhead
So method1 will perform better than sequential method
At point k time saved by parallelism and overhead will be equal so
Both method1 and sequential method performs equally

From p=k to p=6
Amount of time saved by parallelism will be less than overhead
So sequential method will perform better than method1

Sequential method vs method2

In above case we got k in between 3 and 4 and now in
This case k is in between 2 and 3 ,because overhead will start
Dominating the time saved by parallelism early than method1.
Here overhead will start dominating early because in method 2
Will create more number of threads than number of threads which were created for method1
According to above graph k will be 2.1
From p= 2 to p=k
Amount of time saved by parallelism will be more than overhead
So method2 will perform better than sequential method

At point k time saved by parallelism and overhead will be equal so
Both method2 and sequential method performs equally

From p=k to p=6
Amount of time saved by parallelism will be less than overhead
So sequential method will perform better than method2

Method1 vs method2

From p = 2 to p= 6

Domination of overhead over time saved by parallelism will be more in method2 compared to method1,so method1 will perform better than method2,and time for both method1 and method2 will keep on increasing because number of threads exceeded number of cores

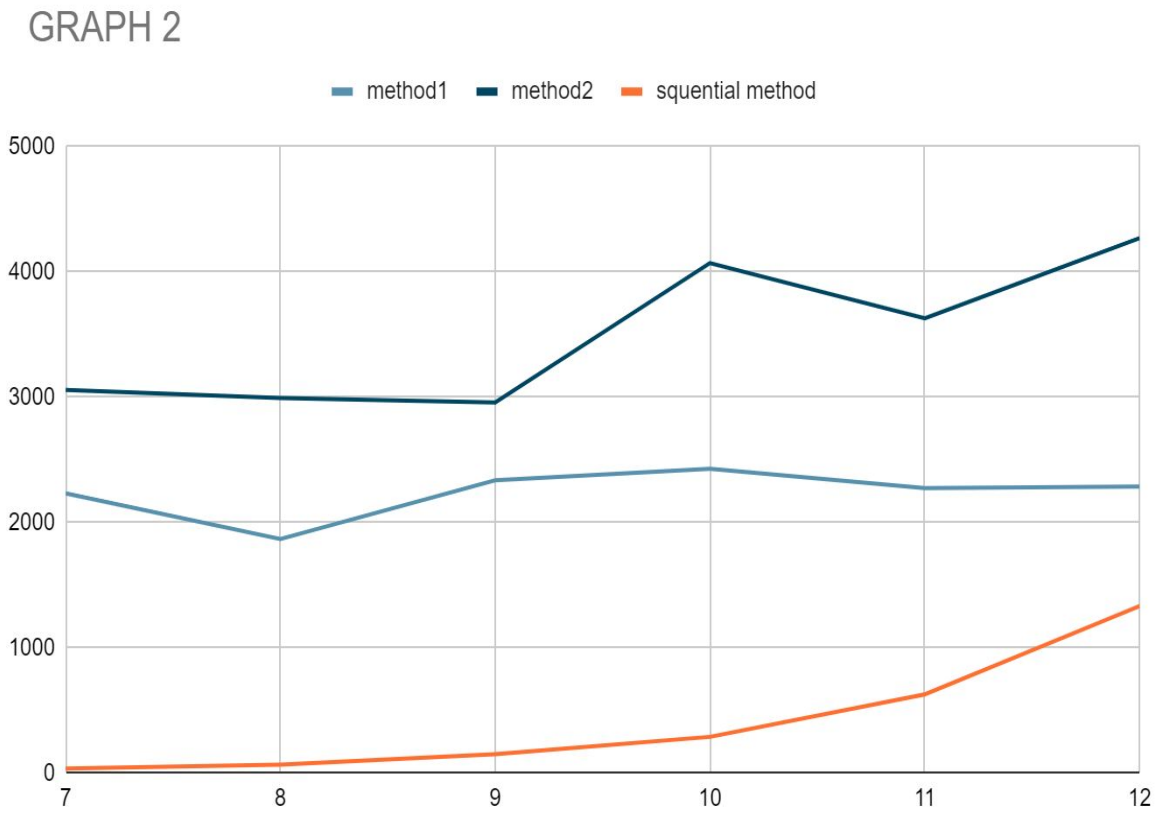
From Graph1 the rankings for these methods are

Rank 1 goes to sequential method

Rank 2 goes to method1

Rank 3 goes to method2

Analysis of graph 2



Here we are fixing number of threads to be 16,n varies from 7 to 12

X-axis is for parameter n
Y-axis is for time taken for sorting

Analysis for method2

From n=7 to n=9 && n=10 to n=11 as n increases timetaken decreases,here parallelism may increases as n increases,so because of more parallelism the time taken may decrease,here time saved by parallelism will dominate the burden created by increasing size of array

From n=9 to n=10 && n=11 to n=12 as n increases timetaken increases Because burden created by increasing size of array dominates time saved by parallelism

Analysis for method1

From n=7 to n=8 && n=10 to n=11 as n increases timetaken decreases,here parallelism may increases as n increases,so because of more parallelism the time taken may decrease,here time saved by parallelism will dominate the burden created by increasing size of array

From n=8 to n=10 && n=11 to n=12 as n increases timetaken increases Because burden created by increasing size of array dominates time saved by parallelism

Analysis for sequential method

As n increases time taken also increases because here we are not creating any threads so parallelism is not possible

Method1 vs method2

Method1 performs better because less overhead in creating threads and context switches compared to method2

Method1 vs sequential method

Sequential method will perform better because method1 will have more Overhead than parallelism

Method2 vs sequential method

Sequential method will perform better because method2 will have more Overhead than parallelism

From Graph2 the rankings for these methods are
Rank 1 goes to sequential method
Rank 2 goes to method1
Rank 3 goes to method2

conclusion

From both graphs we can conclude that sequential method is better than both method1 and method2 ,and method1 is better than method2
But we increase size after some point then method2 may perform better than sequentential and method1,and method1 will perform better than sequential method

Analysis of output for method1 and method2 when n is constant and p varies

We have parameters n and p in our program , let us consider some variable x

Take set r and q which contains sizes of array

assume set r contains sizes from 1 to 2^x

i.e r={1,4,8,.....,2^x}

assume set q contains sizes from 2^(x+1) to 2^n

i.e q={2^(x+1),……,2^n}

for any value of q if p goes from 0 to log(number of cores)base(2) then time will decrease ,here time will decrease because the parallelism increases

for any value of q,if p goes from log(number of cores)base(2) to n-1 then time will increase ,because threads are exceeding number of cores so parallelism cannot increase ,and there will overhead in creating threads and context switches so time will increases

for any value of r ,if p goes from 0 to n-1,then time will increase because size of array is less and overhead of creating threads and context switches dominates the time saved by parallelism

Analysis of output for method1 and method2 when p is constant and n varies

As size of array increases the time taken for both methods will also increases because more burden will be created by increasing size of array but in some times at n=x you may have more parallelism compared when n=x+1 this parallelism will dominates burden created by increasing size of array so from n=x to n=x+1 the time taken may decrease because of more parallelism.
So time may increases as n increases,and time can also decreases as n increases