Parallel Algorithms and Programming

Master 1 Informatique and Master 1 MOSIG

Lab 2 – 2019

Salman Farhat
Ivan Zabrodin

1 OpenMP Loop Scheduling

CPU in MHz in my machine is between 800 and 900

Below is a graph captured, x axis is number of threads and y axis represents number of cycles at the end of the program.

Number of cycles depends on the schedule if it is dynamic, static, run time or guided and also depends on the chunk size that have been selected.

As we see below, number of cycles is decrease as we increase number of threads from 2 to 4 but then increased a little bit as we increase from 4 to 8 threads etc… .

When no `chunk-size` is specified, OpenMP divides iterations into chunks that are approximately equal in size and it distributes at most one chunk to each thread.

Choosing the best chunk size isn't easy it depends on the threads, which threads are fast in doing the work also it depend on the schedule type for example in dynamic schedule threads can help other threads in there work if they finish early while in static schedule threads cannot get other threads work, each thread should finish it's own work.

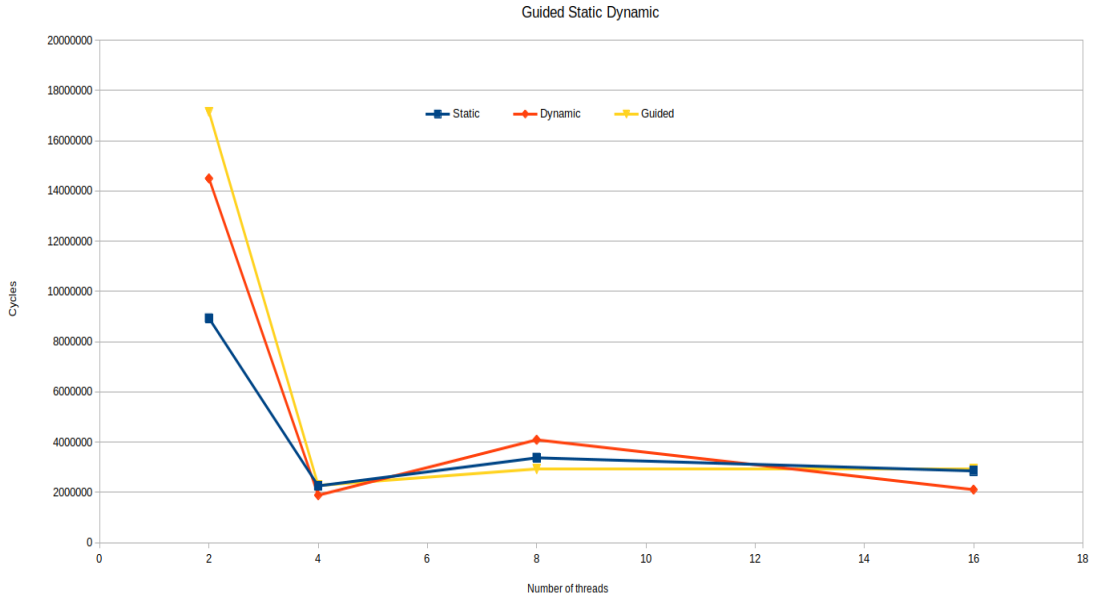

Figure 1

2 Bubble Sort

Parallel Bubble Sort algorithm:

- Each thread takes a chunk and apply sort for it's own chunk.
- after all threads Sorts there chunks they do swapping for the last element of the chunk with the first element of the next chunk.
- repeat until no thread do swap.

Figure below shows how many cycles was needed by the sequential and parallel bubble sort with respect of Array Size.

As we See Below first if the Array Size is less than 1024 parallel is meaning less but we will observe the power of parallelism when we have a big Array Size.

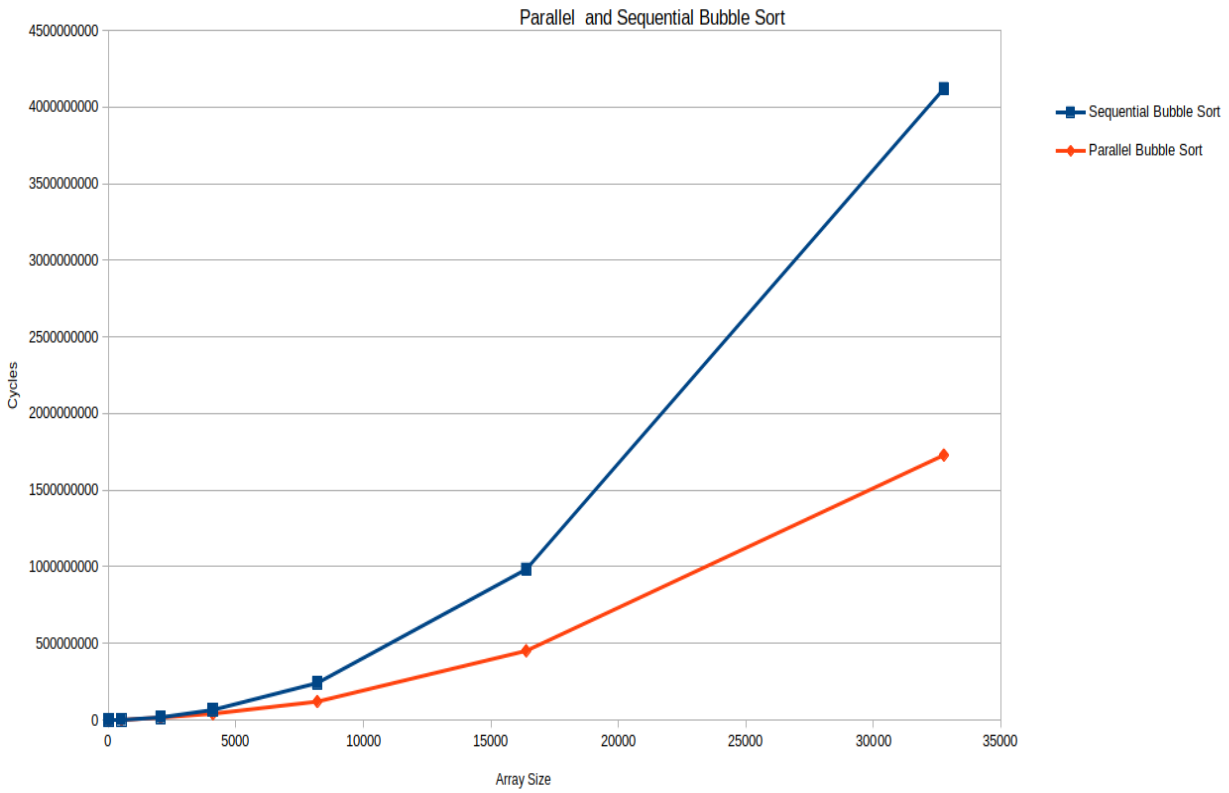

Figure 2

3 Quick Sort

Parallel Quick Sort algorithm <u>without</u> parallel merging:

- Each thread Sort it's own chunk by calling the sequential function
- In the step before I didn't use schedule policy I did it manually on purpose just to understand how can I do it without using the schedule.
- Chunk size is Array size over number of threads
- After sorting chunks we merge 2 elements then 4 then 8 etc… till end which is sorting from 0 and size/2 (hole array).

Parallel Quick Sort algorithm <u>with</u> parallel merging:

- Each thread Sort it's own chunk by calling the sequential function
- Chunk size is Array size over number of threads
- Then threads in a certain way (See the code commented) apply together sorting with size 2 once all finished then apply all 4 together till end which is thread 0 apply merge with size/2 where size is the array size.

Figure bellow shows number of cycles of sequential and parallel algorithms with and without parallel merging.

As we See in figure-3 yellow line which is for the parallel sort **<u>with</u>** merging is way efficient than sequential sort and parallel **<u>without</u>** parallel merging.

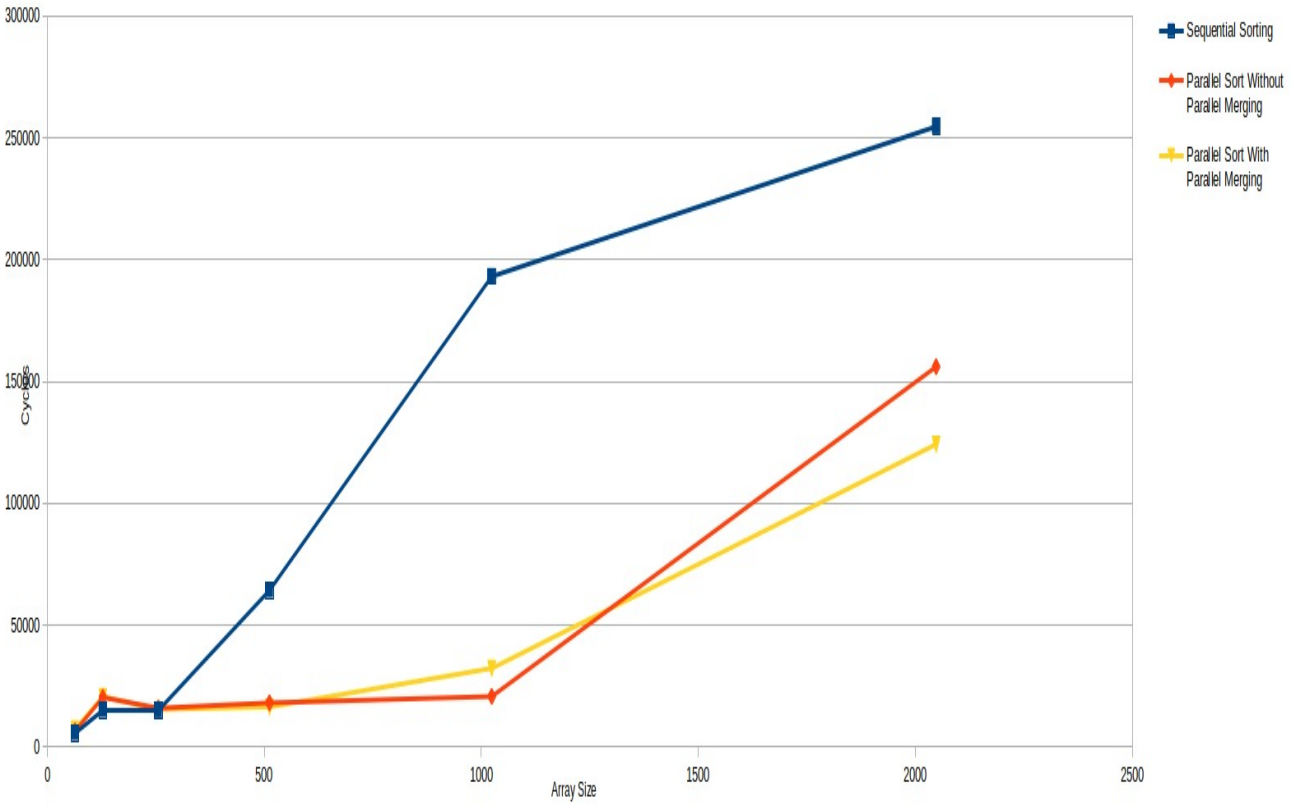

Figure 3

4 MergeSort with tasks

Parallel Merge Sort algorithm :

- We first open a parallel region in main where we call the recursive function and call the function by using pragma omp single

```
#pragma omp parallel
{
#pragma omp single
parallel_merge_sort (X, N) ;
}
```

- Then in the recursive function by using tasks we call recursively on first half and second have then wait by using taskwait then we merge the two sorted chunks.

```
}
void parallel_merge_sort (int *T, const int size)
{
if (size==1)
return;
#pragma omp task
merge_sort(T,size/2);
#pragma omp task
merge_sort(T+size/2,(size+1)/2);
#pragma omp taskwait
merge(T,size/2);
}
```

➔ The number of tasks created by the program with N Size of the array is $\sum_{i=1}^{\log 2(N)} 2^i$

➔ for example if N = 8, number of tasks created is 8 + 4 + 2 = 14 tasks, if N = 16 it is 2 +14 +14 = 30 (16+8+4+2 = 30).

New algorithm to reduce number of Tasks is:

- My algorithm is to let one task call merge sort on one half and then the call the other merge without task
- Then make task wait so we can apply merging both sorted halves.

```
void parallel_merge_sort_reduced_tasks (int *T, const int size)
{
if (size==1)
return;
#pragma omp task
{
merge_sort(T,size/2);
}
merge_sort(T+size/2,(size+1)/2);
#pragma omp taskwait
merge(T,size/2);
}
```

Graph below in figure-4 shows number of cycles in terms of Array Size of each algorithm

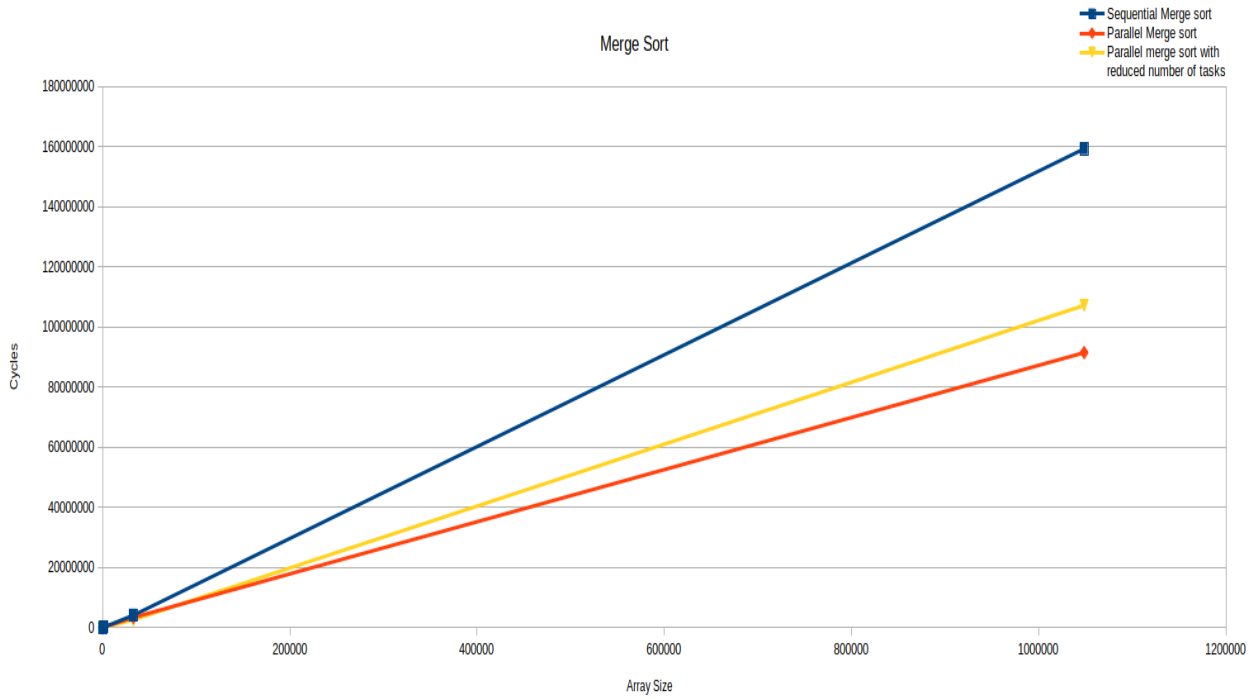

Figure 4