Parallel computing HW 2

Akshay Kumar Varanasi
(av32826)

May 5, 2018

Introduction and Ideology

The way I parallelized this code using MPI is as follows.

1. Initialize MPI.
2. Allocate both the arrays.
3. Initialize the array x only on root.
4. Broadcast the array x
5. Make each processor do smoothing on certain part of x and value is stored in y.
6. Counting done by each processor on certain part of x,y.
7. We call reduce function to add the count from each processor.

I have used Fortran 90 to do this assignment and the value of number of elements I used in these simulation is 16388. I ran my code on 3 Skylake nodes with different number of tasks so that there won't be any issues with memory. I have used `MPI_WTIME()` instead of `system_clock()`for calculating time as it has better precision.

Assignment 1

After parallelizing the code and running the code with different settings we get following result for counting kernel.

Execution	Run-times for x	Run-times for y
Parallel with 2 thread	0.14909	9.3494E-02
Parallel with 8 threads	3.8999E-002	2.5743E-02
Parallel with 16 threads	2.0119E-002	1.3396E-02

Table 1: Comparision of run-times of different no.of threads.

We can see that as the no.of tasks increased the run-times for both x and y reduced by same factor and hence we can see the speed up in counting kernel i.e when tasks become 8 times then times becomes 1/8th. Coming to number of elements below the threshold in serial and parallel, they are same as we used `MPI_REDUCE`.

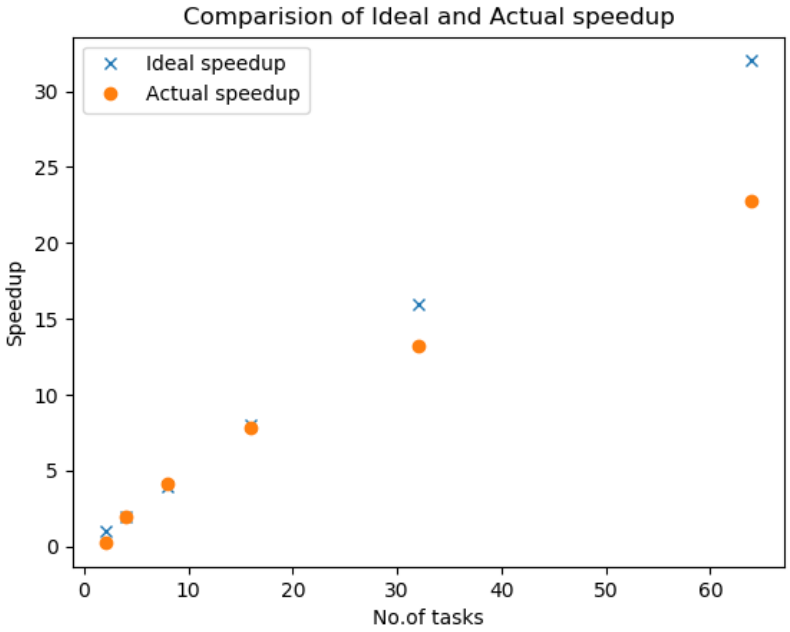

Figure 1: Speedup vs No. of Proc.

Assignment 2

The speed up in smoothing due to parallelization is shown in the following figure 1 and figure 2.

The speedup seems to be good initially but after sometime the speedup starts to deviate from the ideal speedup. In this case, we can see that best speedup is at 16 tasks but after that there is not much difference. So scaling is good till 16 but not after that. Deviation arises after some number of task (in our case it is 16) because as the tasks increase, other costs such as the cost of communication becomes significant.

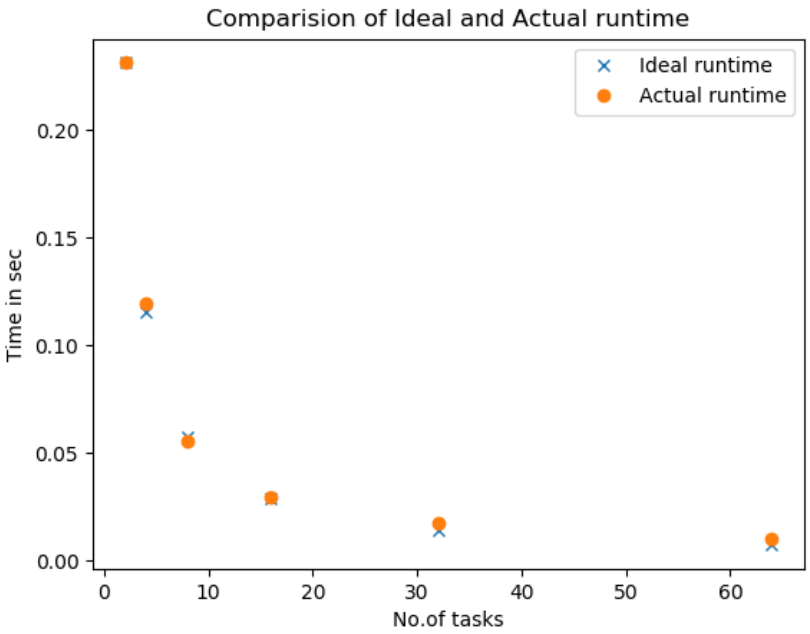

Figure 2: Runtime vs No. of Proc.

The output of the code with 8 tasks

```
This is thread           0
This is thread           1
This is thread           4
This is thread           5
This is thread           2
This is thread           3
This is thread           6
This is thread           7
The time taken to allocate x is  1.096725463867188E-005
The time taken to allocate y is  4.198551177978516E-004
The time taken to initialize x is   1.29591012001038
The time taken to smooth y is  5.530500411987305E-002
The time taken to count x is  3.899908065795898E-002
The time taken to count y is  2.574300765991211E-002
Summary
-------
Number of tasks used ::              8
Number of elements in a row/column ::         16388
Number of inner elements in a row/column ::          16386
Total number of elements ::      268566544
Total number of inner elements ::      268500996
Memory (GB) used per array ::  1049088.00000
Threshold ::           0.10
Smoothing constants (a, b, c) :: 0.05 0.10 0.40
Number of elements below threshold (X) ::       26850816
Fraction of elements below threshold ::    1.00003E-01
Number of elements below threshold (Y) ::           2981
Fraction of elements below threshold ::    1.11024E-05
```