Assignement 4 - CS633

Parallel Computing

Part 4.1

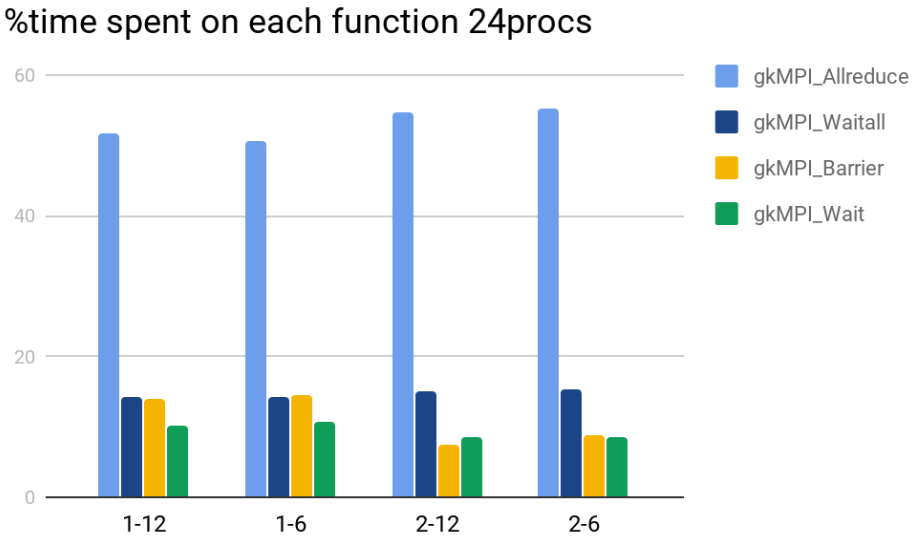
For the first part of the assignment, I ran parmetis-rotor-graph with four different configurations for 24, 48 and 72 processes. I then generated the combined traces for all nodes using *pprof* . On further observation of the traces I could see the percentage time spent for every MPI-function used by parmetis. So I plotted the following four functions:

- MPI_Allreduce
- MPI_Waitall
- MPI_Barrier
- MPI_Wait

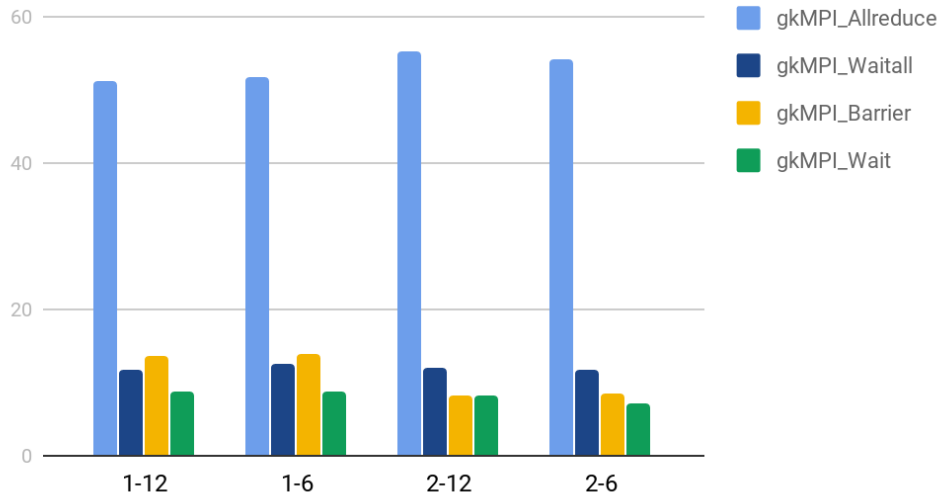
The configurations for the graph that I used are as follows:

1. 1 6 1 1 6 1
2. 1 12 1 1 6 1
3. 2 12 1 1 6 1
4. 2 6 1 1 6 1

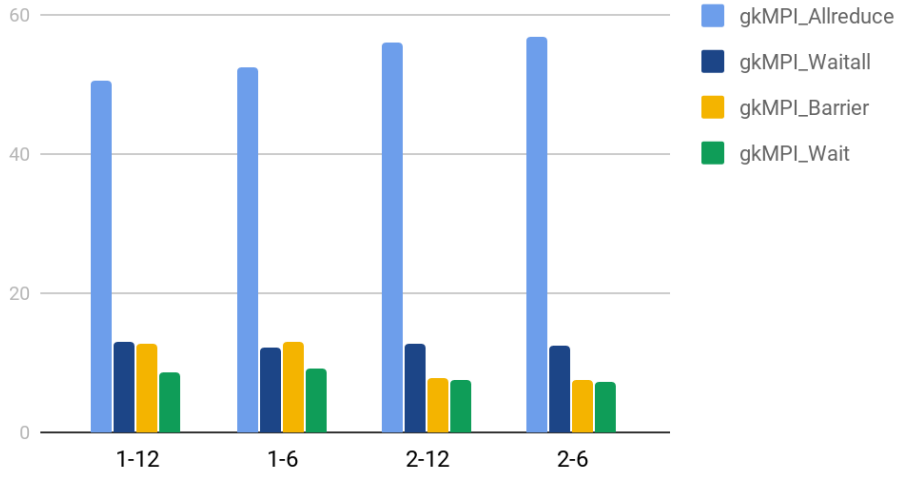
These were the four most time consuming MPI operations in the trace. To verify the same I plotted the percentage time consumed of the said functions; the plots are shown below.



%time spent on each function 48procs



%time spent on each function 72procs



To my surprise, the results were very uniform. As it can be seen from the plots, parmetis is spending the maximum amount of time in MPI_Allreduce, greater than 50%. Going through the code, I could tell that this collective call was in the path of various functions and was required to combine the values from all nodes. This function and its call-path is what I'm going to try and optimize in part two and part three of this assignment.

Part 4.2

From the plots in the previous section, I could clearly see that the MPI_Allreduce's call path was the busiest and most time consuming of all the MPI-collectives used by parmetis. By Amdahl's law, we should try to improve the common case, which in this context will be the most time consuming path.
I can think of the following methods to optimize this code,

- Switch the MPI_Allreduce function with other functions like MPI_Reduce_scatter; or MPI_Reduce and then MPI_Scatter. We could also try MPI_Gather and then do the compute the operation and then do a MPI_Scatter
- Another optimization the I could think of while going through the code was that of the loop_unrolling. There are multiple straight loops in the program call path which can be improved using loop_unrolling.

Part 4.3

Following the two optimizations in the part 4.2, I made the following changes to the parmetis code,

- I replaced the MPI_Allreduce collective call with MPI_Reduce_scatter in *libparmetis/gkmpi.c.* Both functions take the same arguments and it was easy enough to replace.
- The second optimization I made was in *libparmetis/kmetis.c.* I replaced calculation-heavy completely rolled loop with a loop is unrolled by a factor of 2.

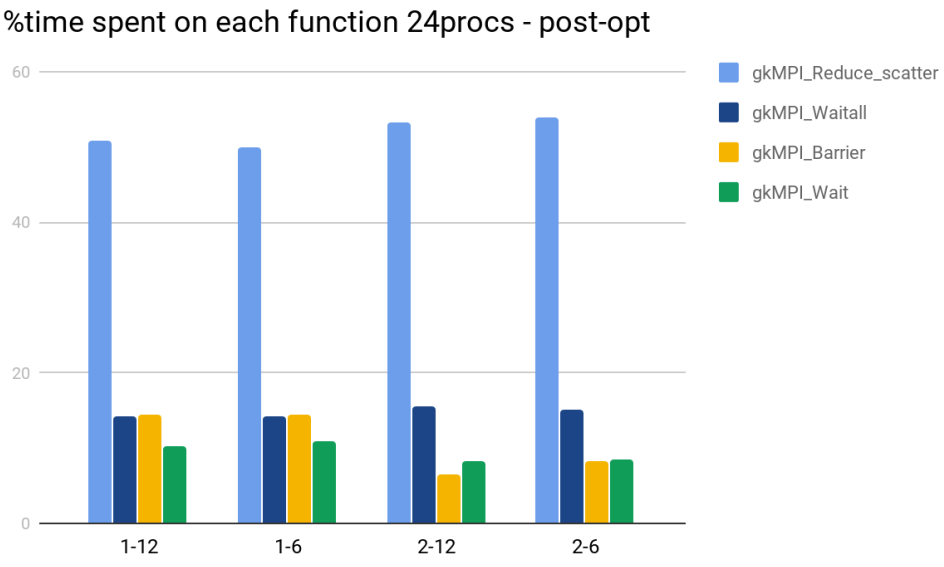
Loop_unrolling:

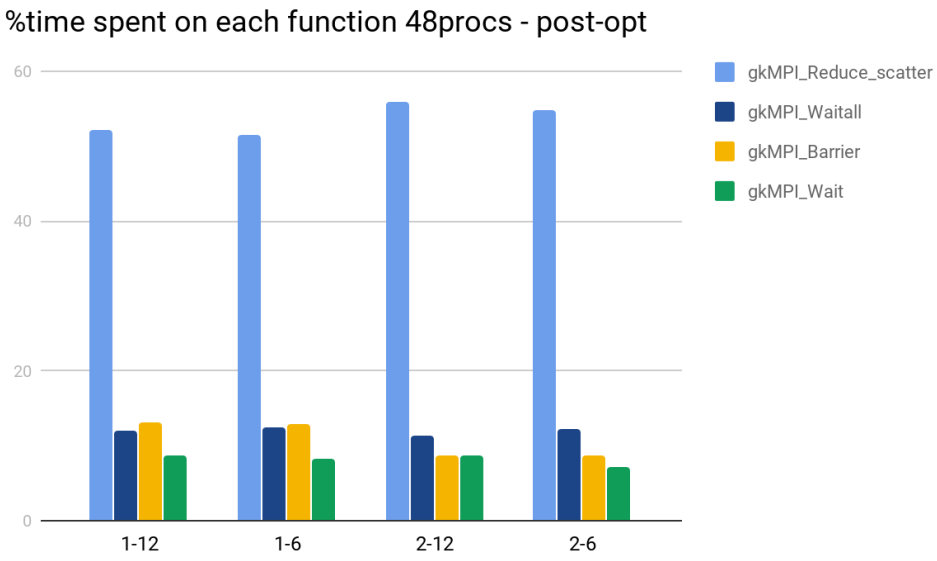
```
int sum = 0;
for(int i = 0; i < 10; i++) {
    sum += a[i];
}
```

The above loop is a normal loop, after an unrolling by the factor of two, it looks like this,

```
int sum = 0;
for(int i = 0; i < 10; i+=2) {
   sum += a[i];
   sum += a[i+1];
}
```

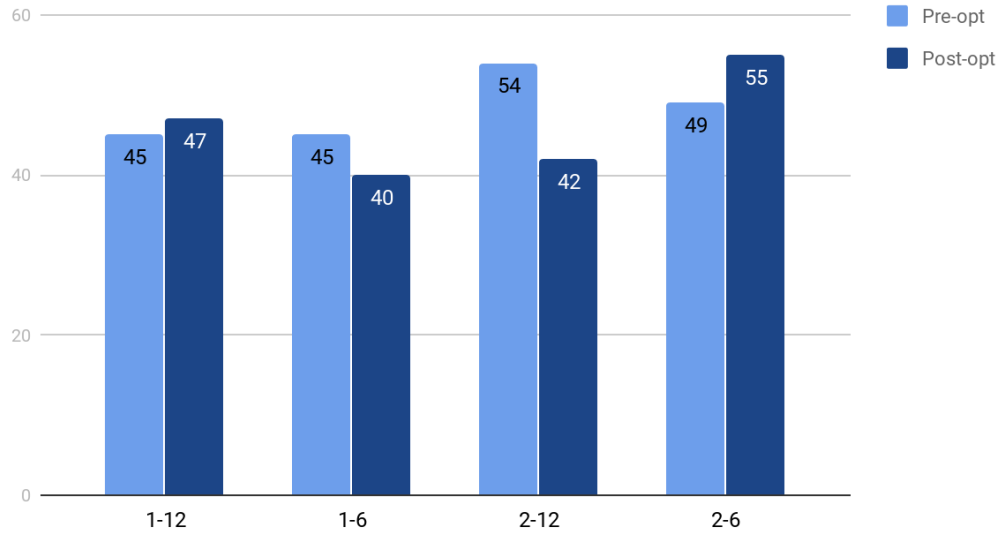
To see the effect of my optimizations I plotted the function-wise percentage time plot again,





As we can see from the plots, there isn't much change in the percentage time spent in the MPI-collectives. To see the actual effect of the optimization I then plotted the total time of the program execution.
The plot is a bit tricky to read. For each configuration on the x-axis, we show the time in milliseconds, the seconds part of execution is same of one configuration.
The plots for 24 and 48 processes are shown below. As we can see, for 24 processes there's a slight improvement of 5milliseconds and 12 milliseconds for the two middle configurations. We can see a slight improvement in 1-6 and 2-6 configuration of the plot for 48 processes as well. Now these plots might not be absolutely accurate owing to the variable load on the systems in the CSE cluster. But in theory, I expect this optimization to work.

Time taken in millisecs - 24 processes



Time taken in millisecs - 48 processes