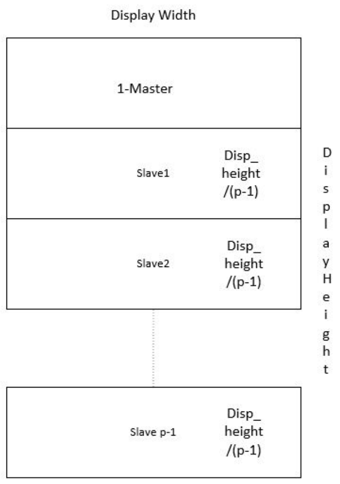
Multiprocessor Programming Parallel Mandelbrot Computation and Load Balancing.

Fig. 1. Static task assignment



I. Introduction

Embarrassingly parallel computation involves computing a data point without dependence of other data points. They are highly optimizable applications. Each processor needs different data and computation does not depend on result from other processes. So, the performance is mainly based on partitioning of data such that the computaion time for processing data by each process is roughly the same. The process is called load balancing and there are multiple techniques for balancing load across processes. Two of those techniques are discussed below for the generation of Mandelbrot set.

II. Approach 1- Static task assignment

For this approach, the display window for Mandelbrot is equally divided into *p-1* parts as shown in figure 1. Such approach is easy to implement and does not involve high communication overhead. However, the load on each processor may not be equal. In case of Mandelbrot calculation, a large portion of the set require much less than 255 iterations and in case of static assignment, such portion may entirely be allocated to a single process leading other processes to perform longer calculations. So, the master has to wait for the slowest process to end to display the output. This approach is not very efficient.

III. Setup for approach 1

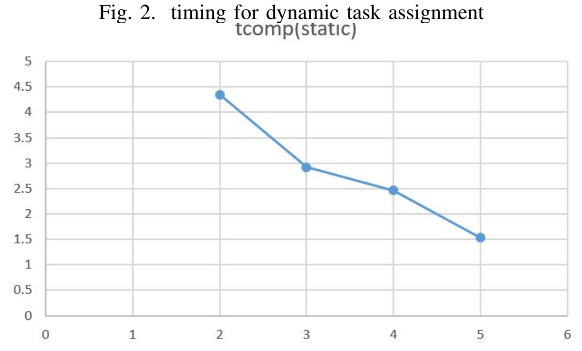
- number of processors=2,3,4,5
- number of rows calculated per request=display height / (p-1)
- display width=2000
- display height =1500

IV. Results for Approach 1

- average time for sequential mandelbrot (*tseq*)=5.24
- average time for parallel mandelbrot (p=5) (*tpar=tcomp+tcomm+tio*)=1.54+1.53=3.07
- speedup=5.24/3.07=1.7

Figure 2 summarizes the results for approach 2.

Fig. 2. timing for dynamic task assignment



V. Approach 2- Dynamic task assignmrnt

Since, the number of iterations for each pixel is different, the processes may run at different speeds and some complete tasks before others. In such case a dynamic way of task assignment can be done for comparitively more efficient load balancing than static assignment. For dynamic task assignment, the master process has to know when the salve is idle and assign task accordingly (Figure 2). This results in higher communication overhead as multiple messages have to be passed between processes not only for data but also status of the process. However, the processes can complete tasks together resulting in higher system efficiency.

VI. Setup for approach 2

- number of processors=2,3,4,5
- number of rows calculated per request=100
- display width=2000
- display height =1500

Fig. 3. Dynamic task assignment

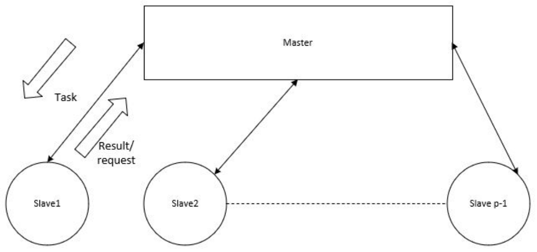
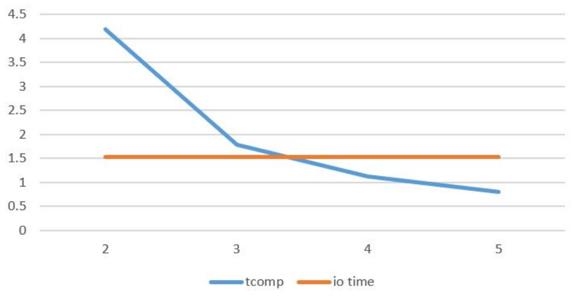


Fig. 4. timing for dynamic task assignment



VII. Results for Approach 2

- average time for sequential mandelbrot (*tseq*)=5.24
- average time for parallel mandelbrot (p = 5) (*tpar*=*tcomp*+*tcomm*+*tio*)=0.81+1.53=2.34
- speedup=5.24/2.34=2.24

Figure 4 summarizes the results for approach 2.

VIII. Discussion

For convinience, the comm time is considered for *p=2* case in both approach 1 and 2. The time reported is time for returning results for each request. The average communication time in both approach are listed below:

- tcomm for approach 1=0.35
- tcomm for approach 2=0.098

Comments on Tcomp, Tcomm and io time

- For smaller batch, the communication time is also small.
- The communication time for requests is negligible.
- Since, the process needs to write to file after all the calculations serially, the IO time is almost same for sequential, approach 1 and approach 2.
- The computation time for approach 2 is less than other two because the load is balanced more efficiently.

IX. Conclusion

A more efficient way to balance the load would be to send the requests for processing row by row to the idle available processes. In such case, we can be pretty sure that all the processes finish computation almost at the same time. However, the communication overhead would be very high.