Programming Assignment 1: Parallel Computation of Polynomial Evaluation

Wenqing Shen (wshen35); Qi Zheng (qzheng61)

Feb 28, 2017

Abstract

CSE6220 programming assignment 1 asks to implement parallel computation of polynomial evaluation. This report gives a brief description of the parallel algorithm and the method to implement it at first. The computation results using cluster are then shown and discussed. At the end, ideas of optimizing the computing performance are proposed.

Method

The sequential algorithm is based on Horner's rule.

The main steps of parallel algorithm are shown in Fig. 1 and also listed below

1. Broadcast x value from source rank to each processor, **MPI_Send** and **MPI_Recv** are used;
2. Scatter constants a_i from source rank to each processor, **MPI_Isend** and **MPI_Irecv** are used;
3. Now we have a sequence $[1, x, x, x, \dots, x]$. Locally multiply the local values and get prefix value P_i;
4. Run parallel prefix of the last P_i on each processor using multiply operation, and get prefix result q_{rank};
5. Locally multiply P_i with q_{rank} divided by the last local P_i, get updated P_i;
6. Locally multiply P_i with constant a_i, get a sequence PP_i;
7. Run parallel prefix sum of sequence PP_i, get prefix results Pre_i and total sum S
8. The total sum S is the value of polynomial.

Results and Discussion

In this project, there are $n = 5{,}000{,}000$ constants, $m = 100$ values. In order to get proper polynomial value, x is in the range of $0.99995 \sim 1.00005$. Constants are in the range of $-0.5 \sim 0.5$. All the constants and x values are generated randomly by MATLAB. The MATLAB code is included in the submission. The n is large enough to have a run time at least one millisecond. The correctness of parallel computation is confirmed by comparing with sequential results.

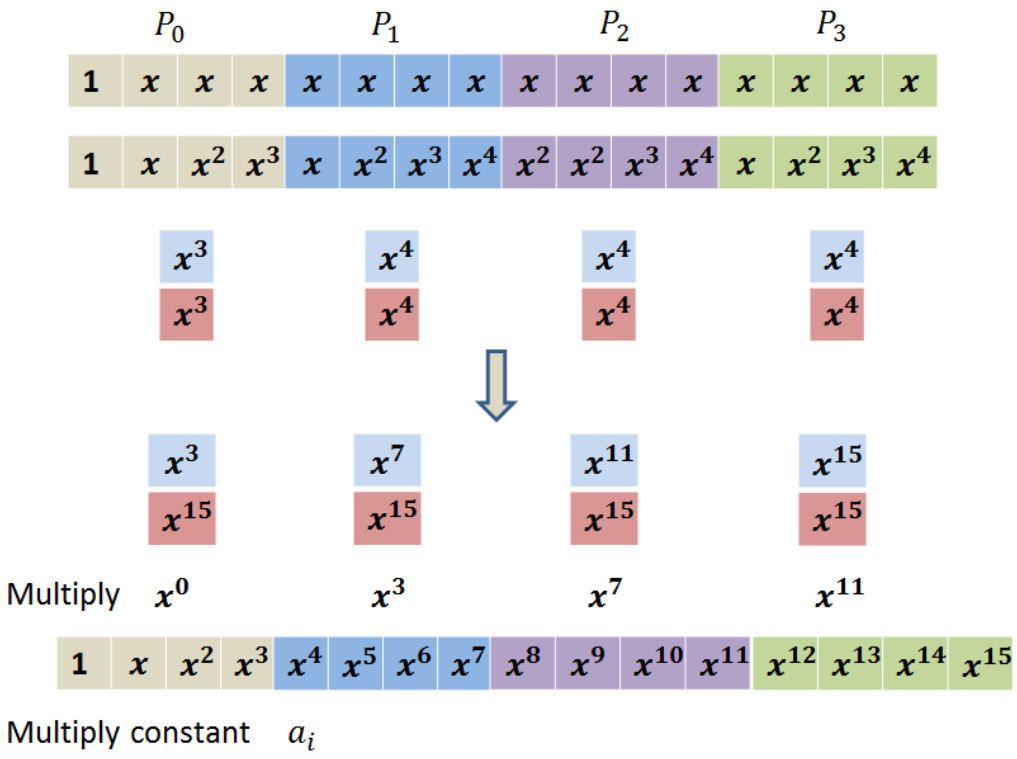

Figure 1. Scheme of parallel algorithm step 3~6.

Effect of processor number on runtime of Broadcast and PolyEvaluator

To investigate the effect of processor number p on the runtime, the code were run on $2\sim64$ cores. Cluster used is provided by PACE (The partnership for an Advanced Computing Environment). The code collects the runtime of both Broadcast and polyEvaluator. For each p value, the runtime is calculated by averaging results from all 100 x values. Total runtime of Broadcast and polyEvaluator for different p is plotted in Fig. 2. The runtime for sequential is also indicated. From the figure we can see that parallel runtime decreases with processor number. From the theoretical analysis, computation time is $\Theta(\frac{n}{p}+\log p)$, communication time is $\Theta((\tau+\mu)\log p)$, the total runtime is $\Theta(\frac{n}{p}+(\tau+\mu)\log p)$. Assumed τ and μ is fixed, for fixed n, with increasing p, the runtime decreases first and then increase. In this project, however, due to the limitation of cluster resources, p cannot be large enough to have $p\log p=\Theta(n)$ to get minimum runtime.

The runtime of Broadcast and polyEvaluator are separately shown in Fig. 3. Since the n is very large, the poyEvaluator takes much longer time than Broadcast. The Broadcast time increase with processor number and the trend is following $\Theta((\tau+\mu)\log p)$, which is the theoretical runtime. The polyEvaluator runtime decrease with processor number, and the trend is very similar to the total runtime as shown in Fig. 2 and the reason is the same as above analysis.

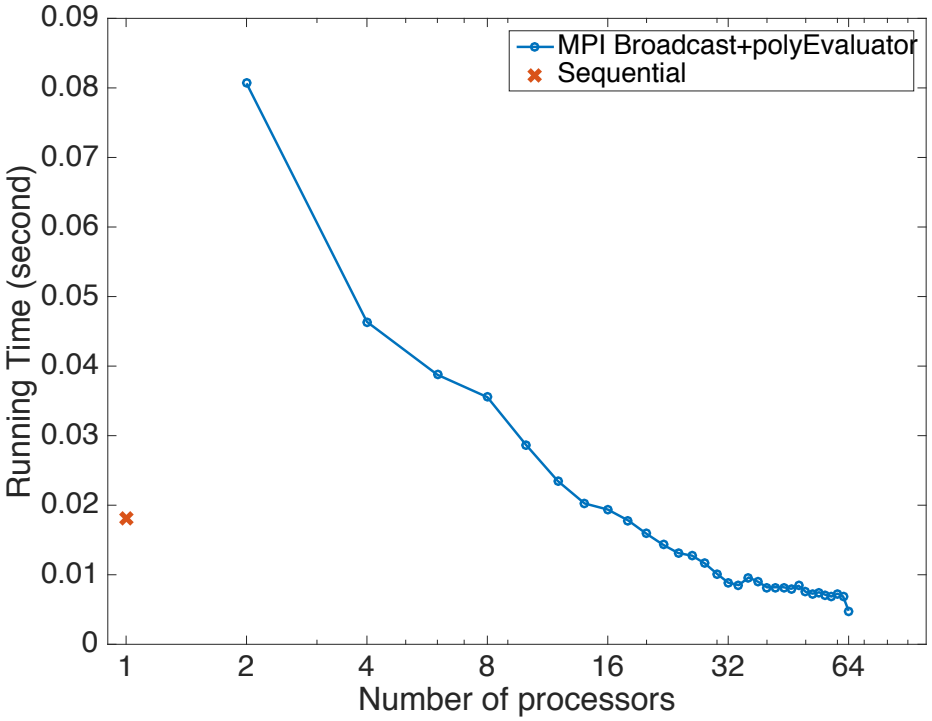

Figure 2. Total runtime of parallel and sequential algorithm

Effect of processor number on Speedup and Efficiency

The speedup of the parallel algorithm is plotted in Fig. 4. It was calculated by $S(n,p) = \frac{T(n,1)}{T(n,p)}$. The efficiency is shown in Fig. 5, which was calculated by $E(n,p) = \frac{T(n,1)}{p\,T(n,p)}$. We can see that only when $p > 18$, the speedup is larger than 1. When p is small, the computing time takes the majority of runtime. Differentiate the function of speedup $S(n,p) = \Theta\left(\frac{n}{\frac{n}{p}+(\tau+\mu)\log p}\right)$ with p, the maximum speedup could be achieved when $p = \Theta(n)$. From Fig. 5, the parallel efficiency is much smaller than 1, and decreases with processor number. In theory, speedup should increase first and then decrease with increasing processor number, but here the p cannot be too large and we only see the trend when p is small. Theoretically, $E(n,p) = \frac{T(n,1)}{p\,T(n,p)} = \Theta\left(\frac{n}{n+p(\tau+\mu)\log p}\right)$ decreases with p, the computation results match well with the trend.

From the analysis of speedup and efficiency, we can see that parallel computation could be optimized if it parallelizes Horner's rule. To get larger speedup, processor number should be large up to $p = \Theta(n)$. To get better efficiency, less processor should be used. The algorithm is efficient when $p = O(\frac{n}{\log n})$.

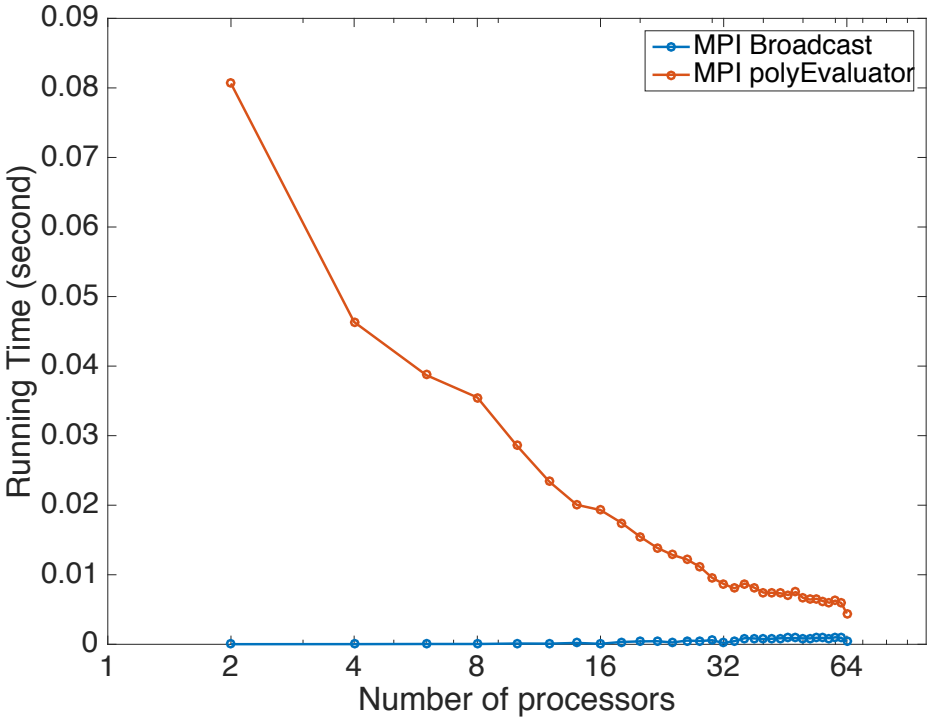

Figure 3. Runtime of Broadcast and polyEvaluator

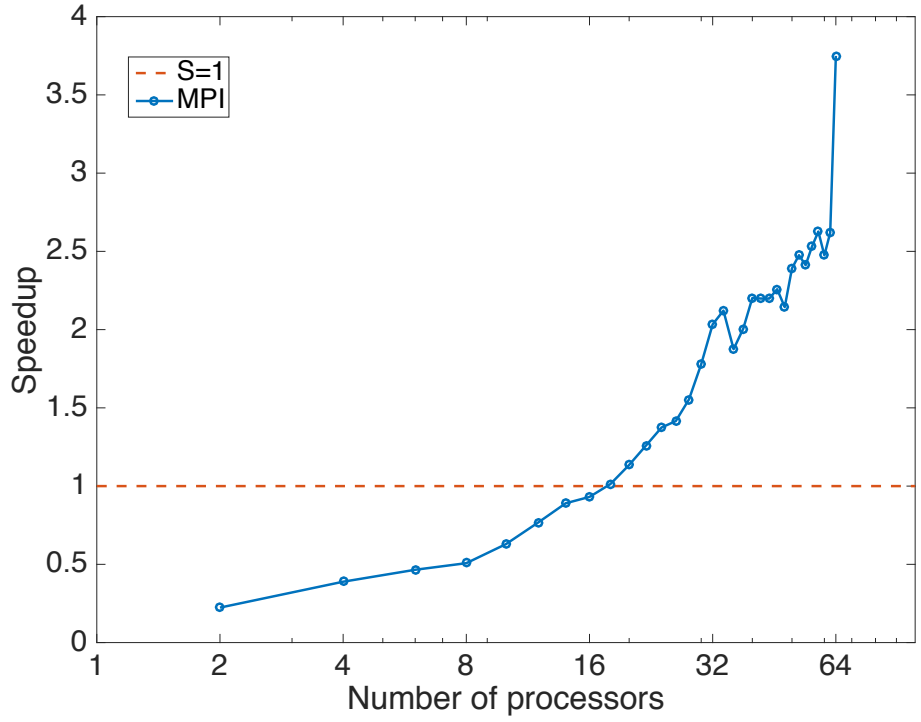

Figure 4. Speedup of parallel algorithm with different processor number

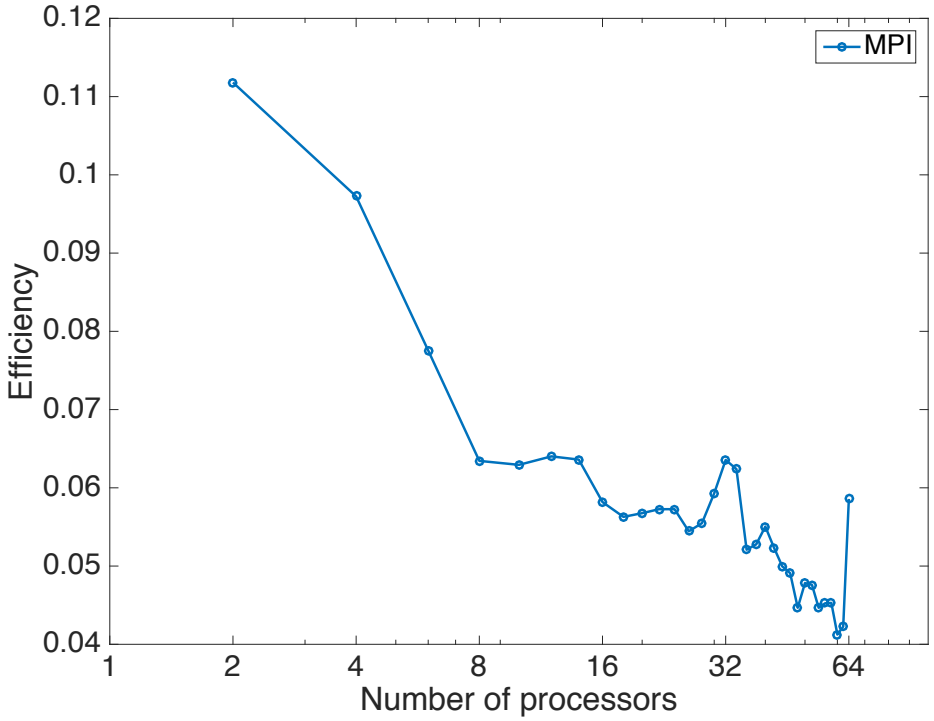

Figure 5. Efficiency with different processor number

Conclusion

In this project, parallel evaluation of polynomial was implemented. The effect of processor number on running time, speedup and efficiency was investigated. The results match well with theoretical runtime. From the analysis of speedup and efficiency, we can see that parallel computation could be optimized if it parallelizes Horner's rule. To get larger speedup, processor number should be large up to $p = \Theta(n)$. To get better efficiency, less processor should be used. The algorithm is efficient when $p = O(\frac{n}{\log n})$.