Michael Thorman

CSCI490/Fall 2021

Lab 4

The purpose of this lab was to compute double-precision matrix multiplication with 1xP processes using MPI to implement the 1D Blocked Column Algorithm. I was also required to describe how the algorithm was implemented, determine the best performance in Gflop/s given different matrix sizes and processes, and determine the parallel efficiency of the program. The machine used was Big Red 3. Also, the number or processes ranged from 1 through 64, the matrix size was between 100 and 1000, and nodes =3, ntasks-per-node = 22.

The 1D blocked column algorithm partitions matrices along a single dimension, and p processors are logically arranged into a 1D topology, much like a ring. In each round, each process multiplies its stored column of matrix A with a corresponding available n/p x n/p submatrix of matrix B. The results are then added to matrix C and matrix A is shifted cyclically by one per round. During each round, a message is sent of size nnz(A)/p words to left neighbor. The communication cost of each processor therefore turns out to be: #Messages = O(p), #Words=O(nnz(A)).

1D Blocked Column Algorithm Pseudocode:

*For all processes Pj in parallel do*

*Local\_A=A(:,j) //dimensions: n x n/p*

*Local B=B(:,j) //dimensions: n x n/p*

*Local C=C(:,j) //dimensions: n x n/p*

*For k=j to j+p-1 do*

*Local C += Local\_A x Local\_B(k % p)*

*Pj sends its local\_A to its left neighbor (n\*n/p message size)*

*Local\_A = message that is sent from its right neighbor*

*Local\_C on each process Pj stores a portion of the final output, or C(:, j)*

The optimization tool for this lab is MPI, or message passing interface, which provides a standard for writing message passing programs on parallel computers. MPI is a popular message passing library because it has good portability, performance, and functionality. I used the following MPI functions in this lab: MPI\_Init(), MPI\_Finalize(), MPI\_Comm\_rank(), MPI\_Comm\_size(), MPI\_Send(), and MPI\_Recv(). And the type of communication was Point-to-Point. In this lab, the advantages of MPI include: execution time reduction and speedup increase, simple computation for each processor, and only elements of matrix A are distributed, while the columns of B matrix are traversed for matrix multiplication. Furthermore, results show that it may not be completely beneficial to increase the number of processes proportional to N size. Also, in some cases, less processes mean faster computation speed for N size compared to more processes, larger N values can be divided and conquered more efficiently using more processes, and smaller process numbers work better on smaller N sizes. The following table shows the computation time for a specific N size and process number. The first observation is that as the size of N and number of processes went up, so did the computation time. In the case of N=100, as the number of processes increased, so did the time in seconds.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Computation time: Matrix size vs. # of Processes* | *2 processes* | *4 processes* | *12 processes* | *24 processes* | *48 processes* | *64 processes* |
| *N=100* | 0.005689 | 0.007738 | 0.010959 | 0.016206 | 0.059958 | 0.078928 |
| *200* | 0.010342 | 0.009536 | 0.012813 | 0.020096 | 0.066257 | 0.089216 |
| *300* | 0.023930 | 0.014429 | 0.017851 | 0.028215 | 0.083295 | 0.113845 |
| *400* | 0.045520 | 0.024115 | 0.024287 | 0.035340 | 0.100977 | 0.141271 |
| *500* | 0.083828 | 0.038834 | 0.032681 | 0.048810 | 0.125256 | 0.175466 |
| *600* | 0.025853 | 0.020803 | 0.031317 | 0.056155 | 0.153669 | 0.218479 |
| *700* | 0.035299 | 0.025190 | 0.041145 | 0.071580 | 0.187539 | 0.269300 |
| *800* | 0.054534 | 0.033018 | 0.048567 | 0.093484 | 0.229143 | 0.327200 |
| *900* | 0.065203 | 0.039922 | 0.058969 | 0.106869 | 0.273394 | 0.395705 |
| *1000* | 0.093196 | 0.050492 | 0.071646 | 0.170720 | 0.324288 | 0.469852 |

*Parallel efficiency = speedup/number of processors*

Below are the computation speeds converted to Gflop/s. In almost every case, the Gflop/s were at their greatest with the number of processes equal to 4. As the number of processes increased, the Gflop/s decreased. Moreover, the best Gflop/s was at N=1000, P=4, and I am guessing that as N increases and processes are equal to 4, then the Gflop/s will continue to increase to a certain point.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0.3516** | 0.2585 | 0.1825 | 0.1234 | 0.0334 | 0.0253 |
| 1.5471 | **1.6779** | 1.2487 | 0.7962 | 0.2415 | 0.1793 |
| 2.2566 | **3.7425** | 3.025 | 1.9139 | 0.6483 | 0.4743 |
| 2.812 | **5.3079** | 5.2703 | 3.622 | 1.2676 | 0.9061 |
| 2.9823 | 6.4377 | 7.6497 | 5.1219 | 1.9959 | 1.4248 |
| 16.7099 | **20.7662** | 13.7944 | 7.693 | 2.8112 | 1.9773 |
| 19.434 | **27.233** | 16.6727 | 9.5837 | 3.65791 | 2.54734 |
| 18.7773 | **31.0134** | 21.0843 | 10.9538 | 4.46882 | 3.12958 |
| 22.3609 | **36.5212** | 24.7249 | 13.6429 | 5.33296 | 3.68456 |
| 21.4601 | **39.6102** | 27.915 | 11.7151 | 6.16736 | 4.25666 |

Comparing these Gflop/s to the Gflop/s found in Lab 2 for IJK, shows a significant increase in Gflop/s as N increases. The following table shows Gflop/s vs. N size for IJK:

|  |  |
| --- | --- |
| 100 | **3.7244** |
| 200 | 2.7787 |
| 300 | 2.9937 |
| 400 | 1.9997 |
| 500 | 2.2781 |
| 600 | 2 |
| 700 | 2.0052 |
| 800 | 1.5673 |
| 900 | 1.7525 |
| 1000 | 1.3286 |

With N=100 and 4 processes, the Gflop/s are similar around ~3.7 Gflop/s. However, as the N size increases the Gflop/s decrease in the original IJK, but the optimized IJK with MPI and column blocking algorithm has an increase in Gflop/s as N increase, if processes remain at 4. As the number of processes increase the Gflop/s go down. Furthermore, with N=1000, and process count = 64, the Gflop/s are still higher than the original IJK version, which ended at 1.3286 Gflop/s.

In conclusion, MPI can help increase the performance of matrix multiplication because of efficient communication. Speedup is determined by number of processes and data size. 2 processes were found to give the best performance.