COMP4300 Assignment 1

Part 1: Replicated Data

Q1: Comparison of Naïve and BLAS DGEMM

Performance Testing Approach

- 20 iterations per m,n,k combination
- Record minimum running time
- No. of Processors is 1, since Naïve and DGEMM are both sequential.

$$\mathbf{GFLOPS} = \frac{\mathbf{2MNK}}{\frac{\mathbf{time\ in\ seconds}}{\mathbf{1000000000}}}$$

Table 1 - Measured execution times on Raijin and GFLOP/s for sequential Naïve and DGEMM matrix multiplication (entries in blue signify M,N,K are different)

Test#	M	N	K	Naïve Time	DGEMM Time	Naïve GFLOP/s	DGEMM GFLOP/s
1	10	10	10	0.000000953674	0.00000095367	2.10	2.10
2	100	100	100	0.00105596	0.000105858	1.89	1.96
3	1000	1000	100	0.120953	0.00883198	1.65	22.64
4	1000	100	1000	0.740265	0.0166318	0.27	12.03
5	1000	1000	1000	7.4889	0.085113	0.27	23.50
6	4000	4000	1000	155.406	1.45727	0.21	21.96
7	4000	1000	4000	188.168	1.32818	0.17	24.09
8	4000	4000	4000	N/A – too long	5.23946	N/A	24.43

Discussion

From the table, it is clear that the Naïve implementation performs very poorly in comparison to DGEMM. Although both Naïve and DGEMM perform similarly for small matrices (N=10 and N=100), the performance varies significantly in larger matrices (N=1000, N=4000). Tests 3 and 4, and tests 6 and 7, produce the same values for 2MNK. However, differences in performance can be explained by which dimension of the matrices is lower. Performance is better for matrices with a smaller shared dimension (i.e. smaller k) than for matrices with a shared larger dimension (larger k), as expected.

Q2: Replicated Matrix Implementation

Selected Approach

1. Matrix A is broadcast to all processors
2. Columns of matrix B are distributed over (no. of processors-1) by processor 0 using MPI_Send and MPI_Recv
3. Processors 1..n calculate multiply AB to find results for their assigned columns of C
4. Cblas_dgemm is used for multiplication
5. Processors 1..n send their results back to processor 0

Purpose

1. Simplification: Master/Slave approach and distributed data approach
2. Efficiency: Parallel computing using p-1 processors

Issues

1. Efficiency: Could be more efficient.
    a. It could completely utilise p processors instead of p-1 – given that the Master process would be available for computation while waiting for other processes to complete.
2. Using an algorithm that splits up matrices into blocks could be more efficient (i.e. SUMMA algorithm)

Q3: Model the computation and communication time of your algorithm

Model of Computation and Communication

1. Broadcast matrix A (mxn) to all processes

$$t_{comm1} = (p) \times (t_{stup} + (m \times k \times t_{data}))$$

2. Send $\frac{n}{p-1}$ columns of matrix B (kxn) to all other processes

$$t_{comm2} = (p-1) \times \left(t_{stup} + \left(\frac{n}{p-1} \times k \times t_{data}\right)\right)$$

3. Compute matrix multiplication using CBLAS_DGEMM

$$t_{comp} = 2\left(\frac{n}{p} \times (k \times m)\right) FLOP$$

4. Each process sends its own results $\left(\text{m rows}, \frac{n}{p-1} \text{ columns}\right)$ to process 0

$$t_{comm7} = (p-1) \times \left(t_{stup} + \left(m \times \frac{n}{p-1} \times t_{data}\right)\right)$$

Note: $t_{comm3,4,5,6}$ pass one integer each to another process and hence are trivial.

$total = t_{comm1} + t_{comm2} + t_{comp} + t_{comm7}$

$$total = (p) \times \left(t_{stup} + (m \times k \times t_{data})\right) + (p-1) \times \left(t_{stup} + \left(k \times \frac{n}{p-1} \times t_{data}\right)\right) + 2\left(\frac{n}{p} \times (k \times m)\right) + (p-1) \times \left(t_{stup} + \left(m \times \frac{n}{p-1} \times t_{data}\right)\right)$$

When multiplying small matrices, the algorithm is limited by computation. However, as matrix sizes increase, communication becomes a bottleneck. However, performance continues to increase due to processing power derived from extra processors.

Predicted Execution Time

Table 2 - Predicted execution time and GFLOP/s for square matrix multiplication with replicated data

No. Processes	N	Time	GFLOP/s
1	1000	0.1000	20.0000
2	1000	0.0505	39.6039
4	1000	0.0257	77.8210
8	4000	0.8181	156.460
16	4000	0.4338	295.067
32	4000	0.26131	489.8397

Justification of Parameters

The parameters I have used for calculations are as follows:

- Latency = 0.1us = 0.0000001 seconds
- Peak bandwidth = 8.18 GB/s = 1.222e-10 seconds per bit
- FLOP: 20 GFLOP/s = 5e-11 seconds per FLOP

Latency and peak bandwidth were derived from the results of lab1. GFLOP/s were derived from the fact that the Raijin processors are 2.6GHz, and estimating 4 FLOP per cycle.

Q4: Actual Performance Data for NxN Matrices

Performance Testing Approach

- 50 Iterations
- Record minimum running time

$$\mathbf{GFLOPS} = \frac{\mathbf{2MNK}}{\frac{\mathbf{time\ in\ seconds}}{\mathbf{1000000000}}}$$

Table 3 - Measured execution times on Raijin and GFLOP/s for square matrix multiplication with replicated data

Test #	No. Processes	N	Min Time	Max GFLOP/s
1	1	10	0.000000953674	2.10
2	1	100	0.000113964	17.55
3	1	1000	0.088353	22.64
4	2	10	0.00000286102	0.70
5	2	100	0.000159025	12.58
6	2	1000	0.049497	40.41
7	4	1000	0.037024	27.00
8	8	4000	0.996095	128.50
9	16	4000	0.601080	212.95
10	32	4000	0.354333	361.24

Note

- If no. of processes > 1, take a master/slave approach where the master hands out portions and waits to receive results;
- Otherwise, compute using 1 process with no communications.
- Time was calculated from start of the implementation till the very end. This includes computations that are not part of the replicated algorithm, but are required such as initialising variables and assigning columns of matrix A to different processors.

Discussion

At small matrix sizes, the performance of the implementation is limited by communication. For example, note that when NP=1 and N=100, GFLOP/s is 17.55. When NP=2 and N=100, GFLOP/s is 12.58 – a slowdown of 5GFLOP/s mostly due to communication. However, as matrix sizes increase, communication cost becomes trivial and the implementation's performance is limited by computation. The more processes dedicated to the task, the greater the performance. For example, as NP increases from 8 to 16 to 32, and as N remains equal at 4000, performance increases from 128.50 to 212.95 to 361.24 GFLOP/s, almost tripling from the first to the last.

Measured vs. Predicated values

The predicted values are much better than those measured. The difference is due to multiple factors which are all implications of the fact that the model is an optimal, unrealistic state. Firstly, the model is an abstraction away from reality. For example, it does not account for execution-time issues such as initialisation, accessing data, etc. Secondly, the model does not account for communication issues and error handling. Thirdly, the model does not account for extra communications required to implement the algorithm. Rather, it focuses, abstractly, on the algorithm itself rather than what it takes to implement it. Fourthly, speed would also be language-specific. However, the impact of this would have been minimal. Overall, the differences in the performance between that predicted and measured can be explained by the fact that the model abstracts away from reality and presents an optimal state that is unrealistic. However, models are beneficial and allow us to grasp the concepts and apply it in order to achieve an implementation.

Q5: Non-Square Matrix Multiplication

Table 4 - Measured execution times on Raijin and GFLOP/s for square matrix multiplication with replicated data (entries in blue signify M,N,K are different. Entries in red are used to show poorer performance than expected).

Test #	No. Processes	M	N	K	Min Time	Max GFLOP/s
1	1	100	10	10	0.000000286102	6.99
2	1	10	100	10	0.0000038147	5.24
3	1	10	10	100	0.00000286102	6.99
4	1	100	1000	1000	0.00997901	20.04
5	1	1000	100	1000	0.0100598	19.88
6	1	1000	100	100	0.00952387	20.10
7	2	100	1000	100	0.0114858	17.40
8	2	1000	100	1000	0.011961	16.72
9	2	1000	1000	100	0.011101	18.02
10	4	100	1000	1000	0.00512004	39.06
11	4	1000	100	1000	0.00875306	22.85
12	4	1000	1000	100	0.00531507	37.63
13	8	1000	4000	4000	0.259571	123.28
14	8	4000	1000	4000	0.308756	103.64
15	8	4000	4000	1000	0.261995	122.14
16	16	1000	4000	4000	0.164556	194.46
17	16	4000	1000	4000	0.236042	135.57
18	16	4000	4000	1000	0.154098	207.66
19	32	1000	4000	4000	0.106391	300.77
20	32	4000	1000	4000	0.19039	168.08
21	32	4000	4000	1000	0.108151	295.88

Discussion

The performance for the mult_replicated implementation shows that performance is lower for non-square matrix when M=K but N is smaller. This may be due to the fact that the amount of calculations undertaken that use the B matrix which has dimensions (KxN) of the calculations prior to multiplication occurs on the B matrix. Further, N is the no. of columns of the B matrix, so the higher the N, the more multiplication operations there will be.

Part 2: SUMMA

Q1: Model the computation and communication time for the algorithm provided.

Model of Computation and Communication

Note

$p = r \times c$

$where\ p\ is\ no.\ of\ processors; r\ is\ no.\ of\ processor\ rows; c\ is\ no.\ of\ processor\ columns.$

1. Send $\frac{m \times k}{p}$ elements of matrix A and $\frac{k \times n}{p}$ elements of matrix B to each processer except root

$$t_{comm1} = (p-1) \times \left(t_{stup} + \left(\frac{m \times k}{p} \times t_{data}\right) + \left(\frac{k \times n}{p} \times t_{data}\right)\right)$$

2. Compute matrix multiplication using PGEMM (from SUMMA paper)

$$t_{comp} = k\left(\log(c)\left(t_{stup} + \frac{m}{rt_{data}}\right) + \frac{2kn}{p}FLOP + \log(r)(t_{stup} + \frac{k}{ct_{data}} + \frac{n}{c}FLOP)\right)$$

3. Each process sends its own results $\left(\frac{m}{p}\ \text{rows}, \frac{n}{p}\ \text{columns}\right)$ to process 0

$$t_{comm2} = (p-1) \times \left(t_{stup} + \left(\frac{m \times n}{p} \times t_{data}\right)\right)$$

$total = t_{comm1} + t_{comp} + t_{comm2}$

$$\begin{aligned} total = {} & (p-1) \times \left(t_{stup} + \left(\frac{m \times k}{p} \times t_{data}\right) + \left(\frac{k \times n}{p} \times t_{data}\right)\right) \\ & + k\left(\log(c)\left(t_{stup} + \frac{m}{rt_{data}}\right) + \frac{2kn}{p}FLOP + \log(r)(t_{stup} + \frac{k}{ct_{data}} + \frac{n}{c}FLOP)\right) \\ & + (p-1) \times \left(t_{stup} + \left(\frac{m \times n}{p} \times t_{data}\right)\right) \end{aligned}$$

When multiplying small matrices, the algorithm is limited by computation. However, as matrix sizes increase, the algorithm becomes limited by communication.

Predicted Execution Time

Table 5 - Predicted execution time and GFLOP/s for square matrix multiplication with replicated data

No. Processes	N	Time	GFLOP/s
1	1000	N/A (algorithm produces negative number)	N/A
2	1000	0.2001	9.995
4	1000	0.6001	3.333
8	4000	1.4016	91.324
16	4000	3.0015	42.645
32	4000	6.2015	20.640

Justification of Parameters

The parameters I have used for calculations are as follows:

- Latency = 0.1us = 0.0000001 seconds
- Peak bandwidth = 8.18 GB/s = 1.222e-10 seconds per bit
- FLOP: 20 GFLOP/s = 5e-11 seconds per FLOP

Note

- The majority of time in predicted values is utilised by communication. I calculated using the above values and considering that communications of matrix occur as arrays of MPI_DOUBLE which is a 64-bit integer.
- I also considered that r=c and hence p/2=r=c.

Q2: Actual Performance Data for NxN Matrices

Performance Testing Approach

- 50 Iterations
- Record minimum running time

$$\mathbf{GFLOPS} = \frac{\mathbf{2MNK}}{\frac{\mathbf{time\ in\ seconds}}{\mathbf{1000000000}}}$$

Table 6 - Measured execution times on Raijin and GFLOP/s for square matrix multiplication with SUMMA

Test #	No. Processes	N	Min Time	Max GFLOP/s
1	1	10	0.0000119	0.17
2	1	100	0.00022006	9.07
3	1	1000	0.11123	17.98
4	2	10	0.0000222	0.09
5	2	100	0.000251055	7.97
6	2	1000	0.129423	15.45
7	4	1000	0.045337	44.11
8	8	4000	2.43372	52.59
9	16	4000	1.33499	95.88
10	32	4000	0.640764	199.76

Note

- Time was calculated from start of the implementation till the very end. This includes computations that are not part of the algorithm, but are required such as initialising variables.

Discussion

At small matrix sizes, the performance of the implementation is extremely slow and limited by computation. For example, note that when NP=1 and N=10, GFLOP/s is 0.16. When NP=2 and N=10, GFLOP/s is 0.09. However, as matrix sizes increase, computation cost is of less importance and the implementation's performance is limited by communication. The more processes dedicated to the task, the greater the performance. For example, as NP increases from 8 to 16 to 32, and as N remains equal at 4000, performance increases from 52 to 199 GFLOP/s, almost quadrupling from the first to the last.

<u>Measured vs. Predicated values</u>

The predicted values of GLOP/s are much lower than those measured. This is quite unexpected as it generally tends to be. There is likely an error in the calculations or the model is inaccurate.

Q3: Non-Square Matrix Multiplication

Table 7 - Measured execution times on Raijin and GFLOP/s for square matrix multiplication with replicated data (entries in blue signify M,N,K are different. Entries in red are used to show poorer performance than expected).

Test #	No. Processes	M	N	K	Min Time	Max GFLOP/s
1	1	100	10	10	0.0000210	0.953252
2	1	10	100	10	0.0000229	0.873813
3	1	100	1000	1000	0.018273	10.94505
4	1	1000	100	1000	0.018466	10.83072
5	2	100	1000	1000	0.021524	9.291867
6	2	1000	100	1000	0.016447	12.1602
7	4	100	1000	1000	0.011081	18.04891
8	4	1000	100	1000	0.011803	16.94456
9	8	1000	4000	4000	0.648462	49.34753
10	8	4000	1000	4000	0.659691	48.50756
11	16	1000	4000	4000	0.291741	109.6863
12	16	4000	1000	4000	0.268958	118.9777
13	32	1000	4000	4000	0.20377	157.0398
24	32	4000	1000	4000	0.196212	163.0889

Discussion

The performance for the SUMMA implementation shows that performance is quiet consistent for for non-square matrix when M, K, and N are different sizes. As matrix sizes increase, however, there is notable differences in GFLOP/s between similar-sized non-square matrices. The performance of SUMMA for non-square matrix multiplication is very similar to square matrix multiplication.