Parallel Matrix Squaring with Thread Affinity

SURBHI(CS22BTECH11057)

February 24, 2024

1 Code Overview

The provided code is a C++ program that performs matrix multiplication using two different techniques: the chunk technique and the mixed technique. It reads input parameters such as matrix size(A), number of threads(K), number of cores(C), and bounded threads(BT) from an input file, performs matrix multiplication using both techniques, and writes the resulting matrix along with the time taken for each technique to an output file.

2 Implementation

2.1 Input Handling

The input parameters required for matrix multiplication are read from an input file named `inp.txt`.

2.2 Matrix Multiplication Techniques

- **Chunk Technique:** The matrix multiplication task is divided into chunks, and each chunk is assigned to a separate thread for computation.
- **Mixed Technique:** The matrix multiplication task is distributed among threads in a mixed fashion, where each thread handles a portion of the rows.

2.3 Thread Affinity

The code sets thread affinity on Linux using pthread_setaffinity_np to bind threads to specific CPU cores.

2.4 Output Handling

The resulting matrix and the time taken for each technique are written to an output file named `output_input.txt`.

3 Analysis

3.1 Chunk Technique

- **Pros:**
  - Effective for large matrices as it utilizes multiple threads to process chunks simultaneously.
  - Allows for better load balancing when the number of threads is a multiple of the matrix size.
- **Cons:**
  - May suffer from load imbalance when the matrix size is not evenly divisible by the number of threads.
  - High memory consumption due to storing the entire result matrix in memory.

3.2 Mixed Technique

- **Pros:**
  - Offers better load balancing as threads handle different portions of the matrix.
  - Suitable for matrices where load imbalance is a concern.
- **Cons:**
  - May not fully utilize available CPU resources when the number of threads is greater than the matrix size.

4 Experiment 1: Time Taken vs Bounded Threads

4.1 Experiment Specifications

In this experiment, the y-axis of this graph represents the time taken to compute the square matrix. The x-axis will vary with bounded threads BT in terms of a parameter 'b'. The value of BT will vary as b, 2b, 3b, and so forth, up to K, where b = K/C.

4.2 Experimental Setup

For this experiment, fix N as 1024, and K as 32. The total number of cores C depends on your laptop. Based on the number of cores C, you can decide b. For instance, if C is 16, then have b as $K/C = 2$.

4.3 Experiment Details

The graph for this experiment will consist of four curves:

1. Chunk algorithm without threads being bound to cores (as done in the assignment1 experiments)
2. Chunk algorithm with a given b
3. Mixed algorithm without threads being bound to cores (as done in the assignment1 experiments)
4. Mixed algorithm with a given b

Note that since nothing changes for algorithms 1 and 3, the performance of these algorithms should remain the same. The value of b in both of these cases will be 0.

4.4 Graph

Blue curve represents Chunk without Affinity.
Red curve represents Chunk with b (2).
Green curve represents Mixed without Affinity.
purple curve represents Mixed with b (2).

5 Graph

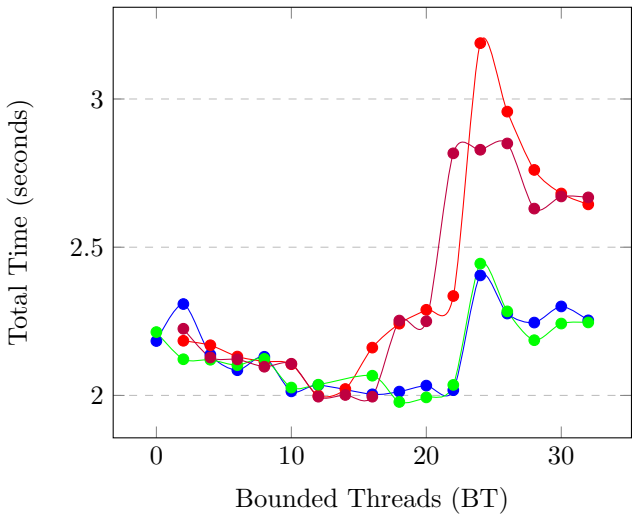

Figure 1: Total Time vs Bounded Threads (BT) for Experiment 1

Table 1: Experiment Results

Threads	**Chunk Algorithm**		**Mixed Algorithm**	
BT	Without Binding	With Binding	Without Binding	With Binding
2	2.18 s	2.18 s	2.21 s	2.22 s
4	2.31 s	2.17 s	2.12 s	2.13 s
6	2.14 s	2.13 s	2.12 s	2.12 s
8	2.09 s	2.11 s	2.10 s	2.10 s
10	2.13 s	2.11 s	2.12 s	2.11 s
12	2.01 s	2.00 s	2.03 s	1.99 s
14	2.03 s	2.02 s	2.04 s	2.00 s
16	2.00 s	2.16 s	2.07 s	2.00 s
18	2.01 s	2.24 s	1.98 s	2.25 s
20	2.03 s	2.29 s	1.99 s	2.25 s
22	2.02 s	2.34 s	2.04 s	2.82 s
24	2.40 s	3.19 s	2.44 s	2.83 s
26	2.28 s	2.96 s	2.28 s	2.85 s
28	2.25 s	2.76 s	2.19 s	2.63 s
30	2.30 s	2.68 s	2.24 s	2.67 s
32	2.25 s	2.64 s	2.25 s	2.67 s

5.1 Observations

- After Reflecting on the effectiveness of the chunk, mixed, and mixed-chunks techniques along with thread affinity ones,
- Getting Random patterns which is observed in the above results.

6 Experiment 2: Time vs Number of Threads

6.1 Experimental Setup

Determine the number of bounded threads (BT) as $\frac{K}{2}$. For example, if K is 32, set BT as $\frac{32}{2} = 16$.
Assign BT threads equally to $\frac{C}{2}$ cores.
The remaining $\frac{K}{2}$ threads will be scheduled by the OS scheduler.

6.2 Graph

Six curves for each input on the x-axis:

1. Average time execution without threads being bound to cores - Chunk algorithm. (Blue)
2. Average time execution of Core-bounded threads - Chunk algorithm.(Red)
3. Average time execution of Normal threads - Chunk algorithm.(Green)
4. Average time execution without threads being bound to cores - Mixed algorithm.(Orange)
5. Average time execution of Core-bounded threads - Mixed algorithm.(Purple)
6. Average time execution of Normal threads - Mixed algorithm.(Pink)

7 Graph

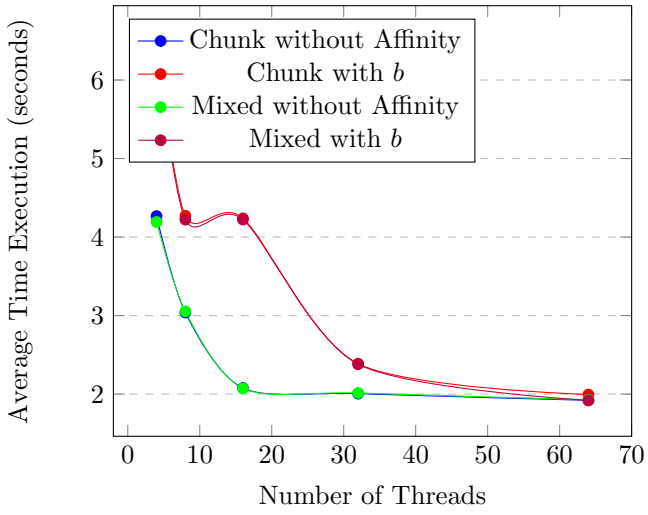

Figure 2: Average Time Execution vs Number of Threads for Experiment 2

Table 2: Experiment Results

Threads	**Chunk Algorithm**		**Mixed Algorithm**	
	Without Binding	Core-bounded	Without Binding	Core-bounded
4	4.27 s	6.49 s	4.19 s	6.43 s
8	3.04 s	4.27 s	3.05 s	4.22 s
16	2.08 s	4.24 s	2.07 s	4.22 s
32	2.01 s	2.39 s	2.01 s	2.38 s
64	1.92 s	1.99 s	1.93 s	1.92 s

7.1 Results

Include six curves as mentioned in the experiment description.

7.2 Observations

- After Reflecting on the effectiveness of the chunk, mixed, and mixed-chunks techniques.
- Getting Random patterns which is observed in the above results.

8 Conclusion

- The provided code efficiently utilizes multiple threads for matrix multiplication using both the chunk and mixed techniques.
- The choice between techniques depends on factors such as matrix size, number of threads, and expected load balance.
- Further optimization can be achieved by implementing dynamic load balancing algorithms or using GPU acceleration for matrix multiplication.
- **Time Taken (Chunk):** 2.5655 seconds
- **Time Taken (Mixed):** 2.48161 seconds