Parallel Matrix Squaring with Thread Affinity

POOJA MANE(CS22BTECH11035)

February 23, 2024

1 Low-level Design

1.1 Overview

The program aims to perform parallel matrix multiplication with Thread Affinity using two different approaches: chunk method and mixed method.

1.2 Implementation

1.2.1 Input and Output Handling

Input Matrix: The program reads the input matrix from a file. Where it represents the matrix size as (N) and the number of threads as (K). As mentioned above, the input will consist of four parameters: N, K, C, the matrix A, and the number of bounded threads BT.
Output Files: One output files is generated: contains executing times one for the chunk method and one for the mixed method. File includes the resulting square matrix and the time taken for computation.

1.2.2 Matrix Multiplication Methods

Chunk Method (`matrixMultiplyChunk`): The chunk method divides the input matrix into contiguous chunks, distributing the workload among multiple threads along with thread affinity.

Mixed Method (`matrixMultiplyMixed`): The mixed method distributes the computation across multiple threads along with thread affinity by assigning each thread a specific row.

2 Experiment 1: Time Taken vs Bounded Threads

2.1 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.

2.2 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 $K/C = 2$.

2.3 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.

2.4 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).

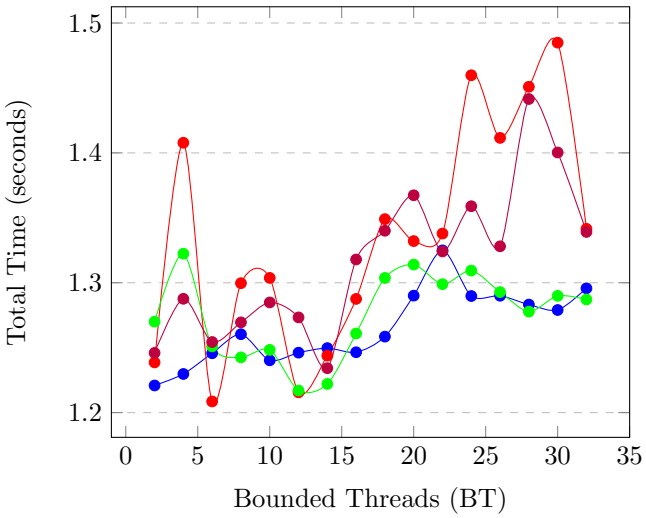

Figure 1: Total Time vs Bounded Threads (BT) for Experiment 1

2.5 Chunk Algorithm Without Threads Being Bound (Blue Curve)

As expected, the performance of the Chunk algorithm without thread affinity remains consistent, and the total time increases with the increase in the number of bounded threads (BT). This behavior is consistent with the assignment1 experiments, where the lack of thread binding results in the OS scheduler distributing threads across available cores.

2.6 Chunk Algorithm with a Given b (Red Curve)

The Chunk algorithm with a given b (where $b = K/C$) shows fluctuations in the total time as the number of bounded threads increases. The fluctuations may be attributed to the binding strategy, where the first b threads are bound to core1, the next b threads to core2, and so forth. This distribution might lead to variations in core performance or contention.

2.7 Mixed Algorithm Without Threads Being Bound (Green Curve)

Similar to the Chunk algorithm without thread affinity, the Mixed algorithm without thread binding maintains consistent performance. The total time increases with the increase in the number of bounded threads.

2.8 Mixed Algorithm with a Given b (Purple Curve)

The Mixed algorithm with a given b displays variations in the total time as b increases. Like the Chunk algorithm with b, these fluctuations may be due to the binding strategy and the distribution of b threads across cores.

2.9 Conclusions

- The chunk and mixed algorithms without thread affinity (Blue and Green Curves) exhibit consistent behavior, as expected, since there is no specific binding strategy.
- The introduction of thread binding with a given b introduces variability in performance, likely influenced by the binding strategy.

- Fluctuations in the total time for algorithms with b may indicate issues related to core distribution or contention.

2.10 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.

3 Experiment 2: Time vs Number of Threads

3.1 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.

3.2 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)

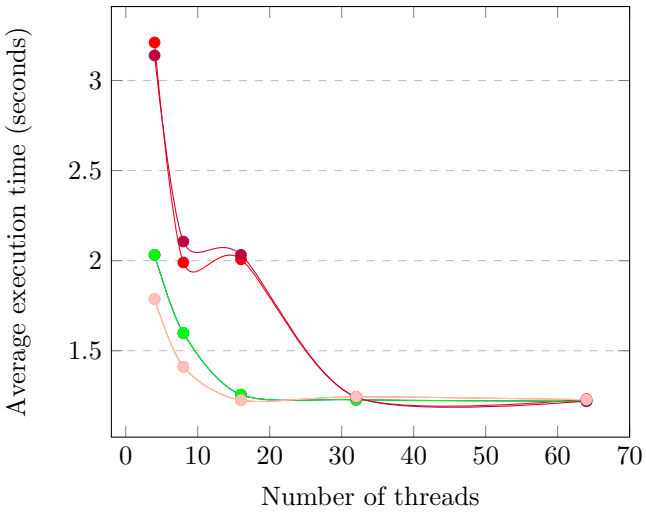

Figure 2: Average Time Vs Number of Threads for Experiment 2

3.3 Results

Number of Threads	Algorithm	Average Time (seconds)
4	Chunk Without Binding	2.03214
4	Chunk With Binding	3.21128
4	Normal Threads - Chunk	2.03214
8	Normal Threads - Mixed	1.78625
8	Mixed Without Binding	3.13939
16	Mixed With Binding	1.78625
16	Chunk Without Binding	1.25622
16	Chunk With Binding	2.00764
16	Normal Threads - Chunk	1.25622
32	Mixed With Binding	1.22534
32	Chunk Without Binding	1.22802
64	Chunk With Binding	1.23155
64	Normal Threads - Chunk	1.21961
64	Mixed With Binding	1.22855

Table 1: Experimental Results

3.4 Observation

Chunk Algorithm Without Binding (Blue Curve):

- The average execution time increases as the number of threads increases.
- Lack of thread binding results in the OS scheduler distributing threads across available cores.
- Performance degradation with an increase in the number of threads is observed.

Core-Bounded Threads - Chunk Algorithm (Red Curve):

- Initially, a significant decrease in average execution time compared to the non-bound threads (Blue Curve).
- As the number of threads increases, a relatively stable performance is maintained.
- The binding strategy of assigning threads to specific cores contributes to improved performance.

Normal Threads - Chunk Algorithm (Green Curve):

- Similar behavior to Chunk Algorithm Without Binding (Blue Curve).
- No specific core assignment results in performance degradation with an increase in the number of threads.

Mixed Algorithm Without Binding (Orange Curve):

- Similar behavior to Chunk Algorithm Without Binding (Blue Curve).
- No specific core assignment leads to performance degradation.

Core-Bounded Threads - Mixed Algorithm (Purple Curve):

- Similar behavior to Core-Bounded Threads - Chunk Algorithm (Red Curve).
- Stable performance as the number of threads increases.

Normal Threads - Mixed Algorithm (Pink Curve):

- Similar behavior to Mixed Algorithm Without Binding (Orange Curve).
- No specific core assignment results in performance degradation with an increase in the number of threads.

3.5 Conclusion

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

4 Analysis

4.1 Correctness

4.1.1 Matrix Multiplication Accuracy

Ensure that the matrix multiplication results are correct for both the chunk and mixed methods done with thread affinity.

4.2 Efficiency

4.2.1 Large Matrix Sizes

It evaluates the efficiency of matrix multiplication for large matrix sizes.

5 Conclusion

- To draw a conclusion based on the whole program and experiment, it is clearly seen that Thread Affinity does not help to improve the performance of parallel matrix multiplication by either of the two methods which are chunk and mixed. It increases the time taken to perform parallel matrix multiplication.
- Thus, we conclude that thread affinity doesn't enhance cache utilization and doesn't reduce cache coherence overhead.