Operating Systems–2: Spring 2024
Programming Assignment 2:
Thread Affinity
Report

Prabhat - CO22BTECH11009

Program Overview:

Below is an overview of the program:

- In the previous programming assignment, we performed matrix multiplication on a matrix A to find its square. There, we directly assigned the number of threads to be created, letting the Linux system handle the assignment of cores to the threads using load balancing.
- In this assignment, we will make some changes to the previous assignment, and as part of this assignment, our objective is to assign threads to specific cores and measure performance.
- Assigning threads to specific CPU cores, known as thread affinity, can enhance cache utilization and reduce cache coherence overhead.
- In the assignment, I have implemented both the chunk and mixed designs with thread affinity.

Specifications of the System:

- ❒ Hardware Information:
    1. Hardware Model: ASUSTeK COMPUTER INC. ASUS TUF Gaming A15 FA506ICB_FA506ICB
    2. Memory: 16.0 GiB
    3. Processor: AMD Ryzen™ 7 4800H with Radeon™ Graphics × 16
    4. Graphics: AMD Radeon™ Graphics
    5. Disk Capacity: 512.1 GiB
- ❒ Software Information:
    1. Firmware Version: FA506ICB.304
    2. OS Name: Ubuntu 23.10
    3. OS Build: (null)
    4. OS Type: 64-bit
    5. Windowing System: Wayland
    6. Kernel Version: Linux 6.5.0-15-generic

Chunk Design:

The low-level design for the chunk program is as follows:

1. Input Parameters:
   - ❒ N: Size of the square matrices (NxN).
   - ❒ K: Number of chunks/tasks to divide the matrix multiplication into.
   - ❒ C: Number of CPU cores available for computation.
   - ❒ BT: Number of bounded threads that need to be executed first.
2. Matrix Multiplication Function (multiply):
   - ❒ It takes two matrices, A and B, along with their size N.
   - ❒ Computes the matrix multiplication A * A and stores the result in matrix B.
3. Chunk Function (chunk):
   - ❒ Divides the matrix multiplication task into chunks and assigns each chunk to a thread for parallel processing.
   - ❒ Determines the number of rows each thread will handle (p = N/K).
   - ❒ Initializes a vector of threads.
   - ❒ Sets CPU affinity for each thread to distribute them across different cores.
   - ❒ Depending on the value of BT, some threads may get CPU affinity set to prioritize big tasks.
   - ❒ Each thread executes the multiply function for its assigned chunk.
4. Main Function (main):
   - ❒ Reads input parameters and matrix from the input file (inp.txt).
   - ❒ Records the start time for performance measurement.
   - ❒ Calls the chunk function to perform matrix multiplication.
   - ❒ Records the end time and calculates the duration of matrix multiplication.
   - ❒ Writes the result matrix B to an output file (out_chunk.txt).
   - ❒ Writes the duration of matrix multiplication to the output file.

5. Thread Execution:
    - ❒ Each thread runs the multiply function with its assigned range of rows to compute.
    - ❒ Threads join back in the main function after completing their tasks.
6. Performance Measurement:
    - ❒ The time taken for matrix multiplication is measured using the chrono library.
    - ❒ The elapsed time is calculated and written to the output file.
7. File Handling:
    - ❒ Input matrices are read from the input file (inp.txt).
    - ❒ The resultant matrix and performance information are written to the output file (out_chunk.txt).
8. Resource Management:
    - ❒ Input and output file streams are properly opened at the beginning of main and closed at the end to ensure proper resource management.

Overall, the program efficiently utilizes multiple threads to perform matrix multiplication in parallel, potentially reducing the overall computation time for large matrices. The CPU affinity setting helps distribute threads across different CPU cores, maximizing parallelism.

Mixed Design:

The low-level design for the chunk program is as follows:

1. Input Parameters:
   - ❒ N: Size of the square matrices (NxN).
   - ❒ K: Number of chunks/tasks to divide the matrix multiplication into.
   - ❒ C: Number of CPU cores available for computation.
   - ❒ BT: Number of bounded threads that need to be executed first.
2. Matrix Multiplication Function (multiply):
   - ❒ It takes two matrices, A and B, along with their size N.
   - ❒ Computes the matrix multiplication for rows assigned to the thread identified by threadID, with a stride of K.
   - ❒ Each thread processes a subset of rows of matrix A and updates corresponding rows of matrix B.
3. Mixed Function (mixed):
   - ❒ Initializes a vector of threads.
   - ❒ For each thread (i), determine the core it will be assigned to using round-robin scheduling (core_id = i % C).
   - ❒ Sets CPU affinity for each thread to ensure distribution across different CPU cores.
   - ❒ Some threads may have their CPU affinity set if BT is greater than 0, prioritizing them for bounded threads.
   - ❒ Each thread executes the multiply function with its assigned thread ID and stride K.
   - ❒ All threads are joined back in the main function after completing their tasks.
4. Main Function (main):
   - ❒ Reads input parameters and matrices from the input file (inp.txt).
   - ❒ Records the start time for performance measurement.
   - ❒ Calls the mixed function to perform matrix multiplication.
   - ❒ Records the end time and calculates the duration of matrix multiplication.
   - ❒ Writes the result matrix B to an output file (out_mixed.txt).
   - ❒ Writes the duration of matrix multiplication to the output file.

5. Thread Execution:
    - ❒ Each thread runs the multiply function with its assigned range of rows (threadID) and a stride of K.
    - ❒ Threads handle disjoint subsets of rows of matrix A to perform matrix multiplication concurrently.
6. Performance Measurement:
    - ❒ The time taken for matrix multiplication is measured using the chrono library.
    - ❒ The elapsed time is calculated and written to the output file.
7. File Handling:
    - ❒ Input matrices are read from the input file (inp.txt).
    - ❒ The resultant matrix and performance information are written to the output file (out_mixed.txt).
8. Resource Management:
    - ❒ Input and output file streams are properly opened at the beginning of the main and closed at the end to ensure proper resource management.

Overall, this code similarly leverages multi-threading for parallel matrix multiplication but with a different thread distribution strategy. It uses round-robin scheduling to distribute threads across CPU cores and allows for prioritizing certain threads for big tasks.

Experiment 1:

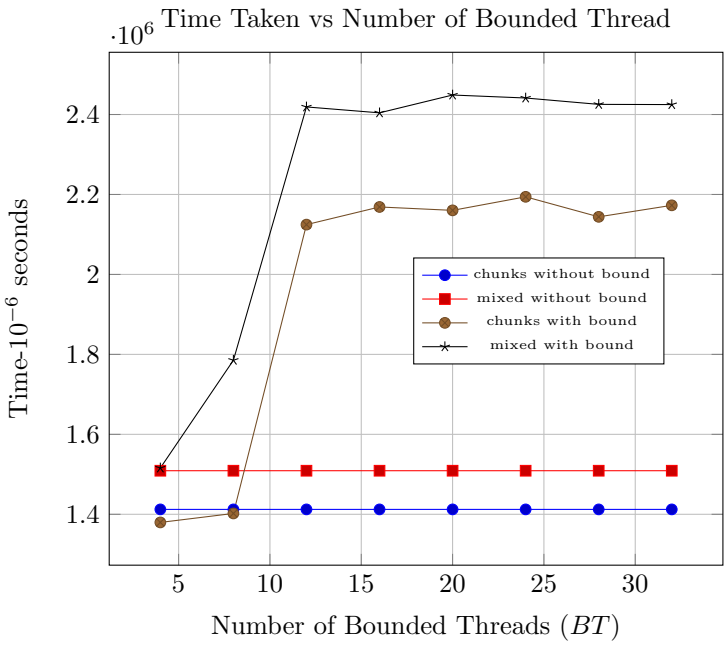

Figure 1: Time vs Number of Bounded Thread Graph

Observations:

- ❒ The performance is not improving due to the bounding of threads.
- ❒ No fixed pattern is observed.
- ❒ The reason for this might be extra time needed for thread binding and basic scheduling.

Experiment 2:

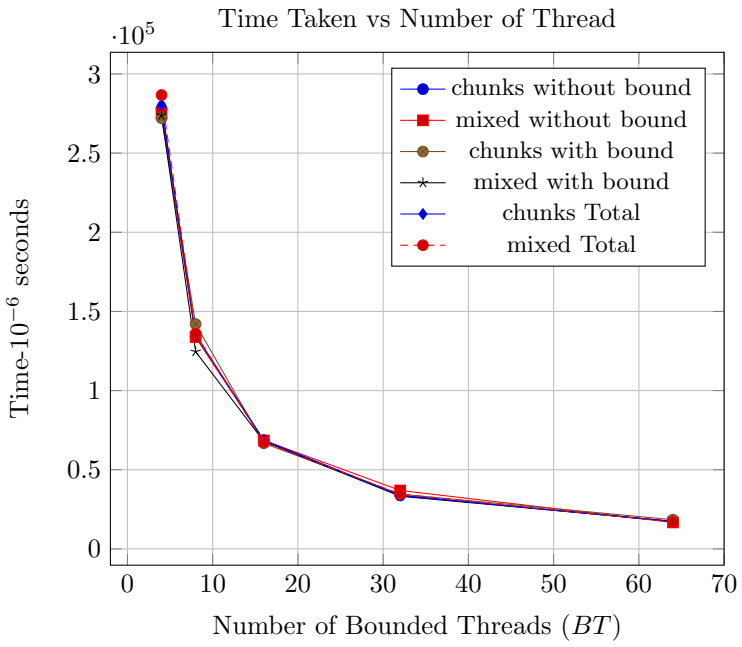

Figure 2: Time vs Number of Thread

Observations:

- ❒ The performance is improving due to the bounding of threads.
- ❒ The pattern is the same as that of the performance of non-bounded threads.
- ❒ But bounding is not coming out to be better than non-bounding.