OPERATING SYSTEMS-II

PROGRAMMING ASSIGNMENT-2: THREAD AFFINITY

PRATHAMESH MALVE

CO22BTECH11010

Low-level design:: Chunk:

- The following code is written in c++.
- N,K,C,BT are defined as global variables
- The matrices (`matrixA` and `matrixC`) are represented using vectors of vectors to dynamically store the matrix elements.
- **ThreadData Structure:**
  - A structure `ThreadData` is defined to store information about each thread, including the start and end row indices it is responsible for and its thread ID.
- **matmul Function:**
  - The `matmul` function performs the matrix multiplication for a specific range of rows. It is designed to be executed by multiple threads.
  - The processor affinity for a thread is set inside this function when a thread calls the function.

- **Main function**:
  - ❖ The code reads the matrix from an input file named "matrix.txt"
  - ❖ matrixA is initialized and filled with the values from the input files.
  - ❖ Threads are created and each thread is assigned particular number of rows to compute matrix multiplication.

    - ❖ The main thread wiats for all the threads to finish execution using join() function.
- The reusltant MatrixC is written to an output file ‘output.txt’ along with the time taken by the code for execution.
- The time taken by the entire code is measured using the `<chrono>` library
- The code attempts to bind threads to specific cores based on the specified algorithm when `affinity` is set to 1.

Low-level design:: Mixed:

The given C++ code performs matrix multiplication using a specified number of threads (`K`) and implements thread affinity to control the assignment of threads to CPU cores. Below is the low-level design explanation:

1. **Input Reading:**
    - The code starts by reading input parameters from a file named "matrix.txt".
    - Parameters include the size of the matrix (`N`), the number of threads (`K`), the number of cores (`C`), and a variable `BT`.
    - The matrix `matrixA` is dynamically allocated and filled with values from the input file.
2. **Thread Affinity Setup:**
    - Thread affinity is controlled by the `affinity` flag, which is set to 1 in this case.
    - Affinity settings are done inside the `matmul` function for each thread.
    - If affinity is enabled (`affinity == 1`), and the thread id is less than `BT` (a condition on the number of threads allowed affinity), and `b` is not equal to 0, the thread's affinity is set.
    - Affinity is set to distribute threads uniformly among available cores (`(data->id / b) % C`), using the `pthread_setaffinity_np` function.
3. **Matrix Multiplication:**

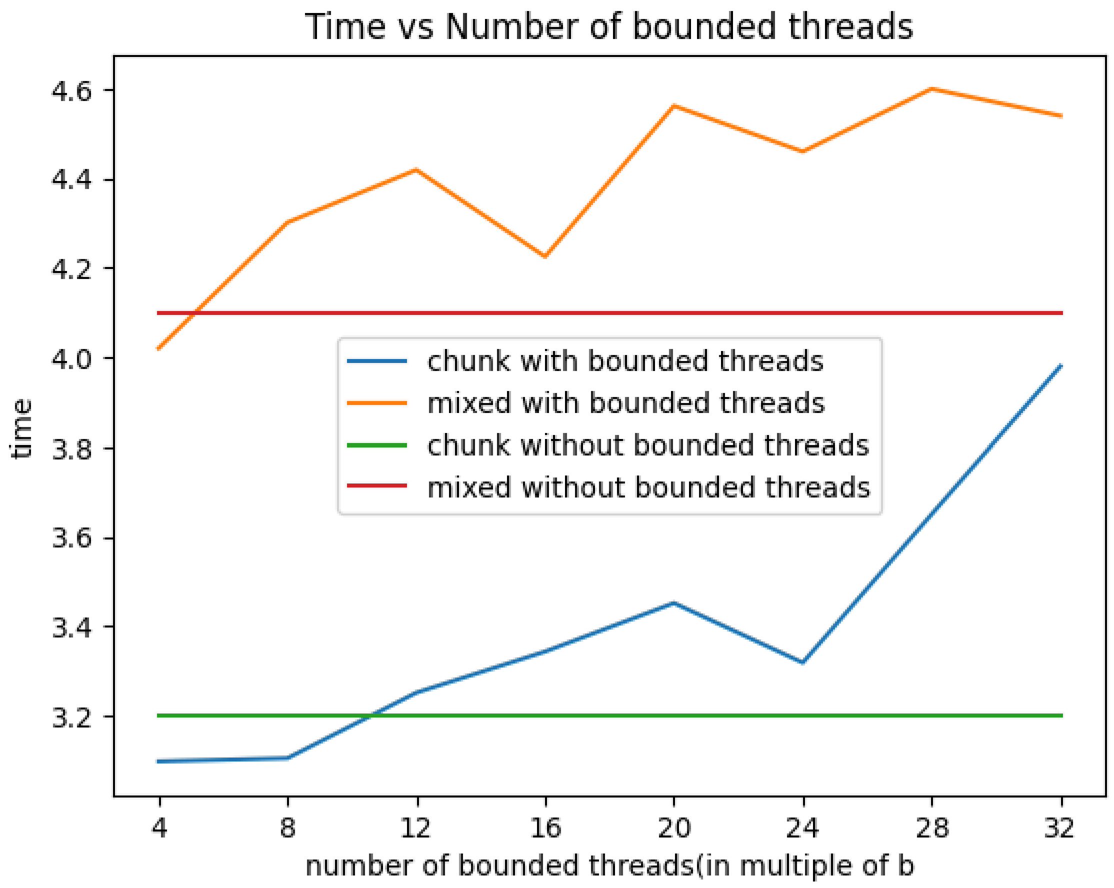
- The main work is done in the `matmul` function, which performs matrix multiplication for a subset of rows assigned to each thread.
- The rows assigned to a thread are determined based on the thread id and the total number of threads (`K`).
- The multiplication result is stored in the `matrixC`.

4. **Thread Creation and Joining:**
   - Threads are created inside the main function, and each thread is assigned a portion of the matrix multiplication task.
   - Thread creation is done using `std::thread` and a custom data structure `ThreadData` to pass relevant information to each thread.
   - After creating all threads, the main function waits for each thread to finish using `join`.
5. **Output Writing:**
   - The result matrix (`matrixC`) is written to an output file named "out2.txt".
   - The output file includes the calculated values of the matrix after multiplication and the time taken for code execution.

System specifications:

- Architecture : x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU(s): 8
- On-line CPU(s) list: 0-7
- Model name: 11th Gen Intel(R) Core(TM) i5-11320H @3.20GHz
- Thread(s) per core: 2
- Core(s) per socket: 4
- Socket(s): 1

Experiment 1:

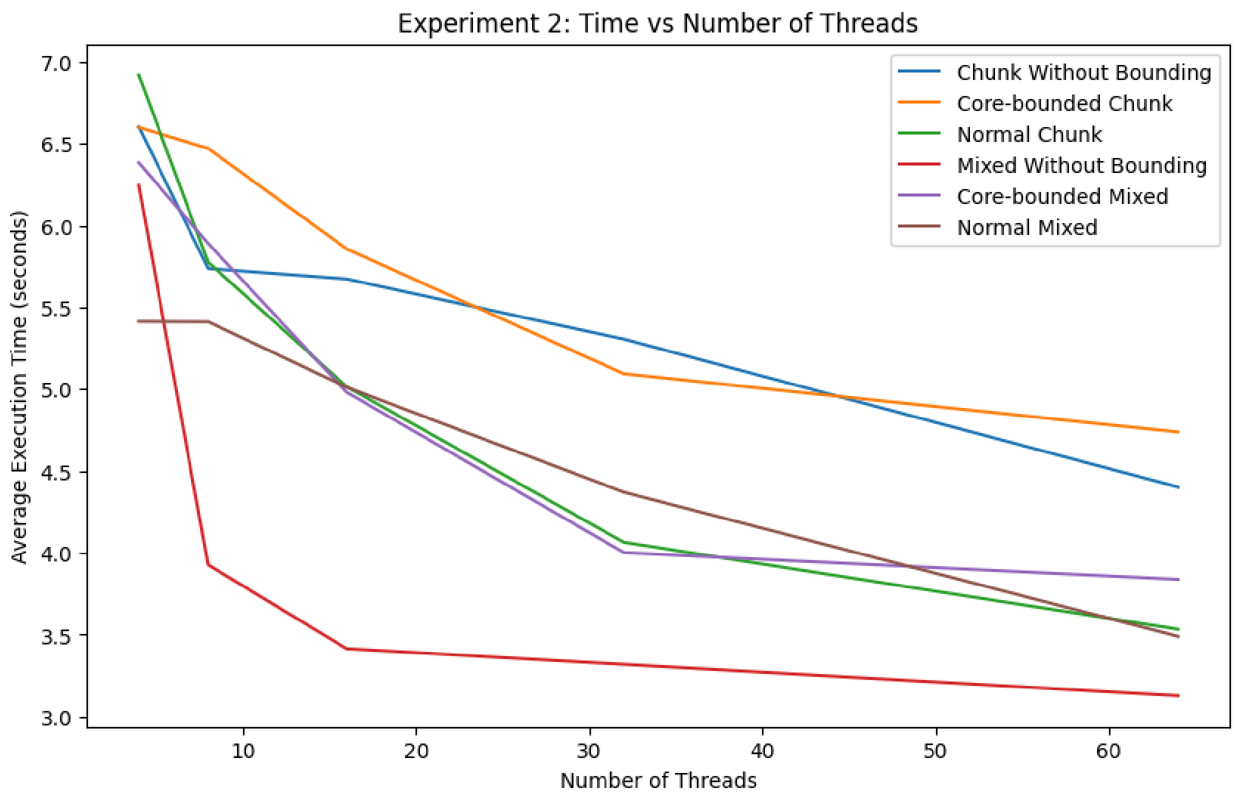


According to the book Operating system concepts, as we increase the number of bounded threads, the execution time should reduce as we save upon the work of shifting a thread from one core to another and reallocating its current cache memory to new thread. However, from our graph we can see that as we increase the number of bounded threads, the execution time is fluctuating(not uniformly increasing or decreasing).This might be because of many reasons. Some of the reasons are:

Increasing the number of threads may lead to increased contention for shared resources, such as memory or synchronization mechanisms. This contention can degrade performance.

The size of the matrices being multiplied might be too small to fully exploit parallelism with the increased number of threads. For smaller matrices, the overhead of thread creation and synchronization may dominate.

Moreover, it also depends on the inputs we take and CPU condition.



The execution time is likely to vary based on the system's scheduling and load. However, binding threads to cores increases the overall performance of the program.The execution time of threads without explicit affinity (Normal Threads) may vary based on the system's scheduling decisions.