Name: Ahmad Farhan

Roll no. 21I-1366

Assignment 1

Thread Mapping Plan

Available Cores: 4 Cores – 8 Threads

System Architecture: Core i5 1135G7 QuadCore

Problem Analysis:

The following steps will be needed to solve the given problem:

1. Reading of integer data from a file into an array
2. Conversion of array into a list
3. Sorting of list using merge sort

The available system is capable of running 8 hardware threads simultaneously on 4 cores with intel's hyperthreading technology. Since we need to apply same operations (i.e., creation and sorting of lists) across different parts of the data (array of integer numbers), we will resort to Domain Decomposition strategies for parallelization.

Decomposition Strategy:

Given that the array is 1 dimensional, so our options are:

1. Block Decomposition
2. Cyclic Decomposition

In this case, Block Decomposition is preferred due to its simpler implementation and the degree of independence of each partition of data. In this strategy, each thread will be dedicated a CPU (hardware thread) and a partition of the array to perform the required operations on. Using block decomposition will also allow us to assign each thread a dedicated CPU using set affinity.

Parallelization Plan:

After data has been read from file into a single integer array, parallelization will be applied to the following two step:

1. **Conversion of Array to List:**
    1. Divide array into n near equal divisions.
    2. Assign each of the n threads one partition of array.
    3. Set Affinity to dedicate each thread to one CPU.
    4. Each thread converts its partition of array into a list.
2. **Sorting Using Merge Sort**
    1. Each thread executes Merge sort operation on the list it converted.
    2. Main thread will then Merge the returned lists of each thread in a sorted way.

Justification of Strategy:

1D Block Domain Decomposition is chosen because the given data can easily be divided into multiple independent parts where each thread will solve a particular part. The result can then be combined to get the desired output. 1D Cyclic Domain Decomposition was not chosen because that would require the implementation of a thread pool and would result in unneeded switching overhead. Functional Decomposition was not chosen because same operations are to be performed on different parts of data.

Testing and Analysis

The following table enlists execution times (in milliseconds) of serial, parallel with affinity and parallel without affinity version of the program for different data sizes i.e., number of elements.

Data Size	Serial Version (ms)	Parallel Version (ms)	Parallel – Affinity (ms)
500	0.612	0.570	0.526
1000	2.070	0.618	0.781
5000	43.63	1.466	1.337
10000	161.5	2.351	2.073
50000	3883.	6.055	5.938

Speed Up Metrics:

Using Actual Speed up formula to calculate speedup of parallel version with and without manual affinity:

Data Size	Parallel Version (ms)	Parallel – Affinity (ms)
500	1.074	1.163
1000	3.349	2.650
5000	29.76	32.63
10000	63.80	77.90
50000	641.3	654.0

Analysis:

Serial and Parallel Comparison:

As is apparent from the above given table, the execution time of the parallel version is exponentially faster than the serial version. Furthermore, it can also be observed that as the data size increases this the difference in execution times between the parallel and serial version becomes more drastic. It must be noted that a further analysis of the program showed that the majority of the gains were obtained from the array to list conversion step.

Parallel Versions with and without setting Affinity:

The above table shows that while the gains are minimal, manually setting the affinity does optimize the thread execution time. Notably, the gains show a linear pattern as well indicating that OS dependent scheduling is good enough to offer near similar performance.

The hardware architecture used for this assignment offered same clock speeds (2.4 GHz) across each core which reduced the possibility of bottlenecks due to workload imbalance. However, changes in system resources may still cause underutilization of certain cores and potentially degrade performance.

Discussion

Setting the CPU Affinity of the parallel version of the program does speed up the program by an average of 4%. However, it must be noted that dynamic workload variations or changes in system resources may invalidate static CPU affinity settings. The approach employed in this program divides the data into 8 portions, creates 8 threads and binds each thread to a dedicated CPU. In theory, assigning each thread a dedicated portion of data and binding it to a CPU does drastically improve cache locality leading to improved performance. This binding of threads to specific cores also reduces the need for frequent context switches thereby reducing the context switching overhead and improving performance further.

Code Execution Instructions:

1. Filename may be passed from terminal as a parameter : ./exefile datafile.txt
   Or execute as ./execfile.out to give the filename as console input.
2. Elements provided in data file must be delimited by a \n (newline) character