Parallel Processing Project: Solving the Heat Equation

Aman Hogan-Bailey

University of Texas at Arlington

CSE-5351: Parallel Processing

Contents

Part 1: Solving 2D Equation for different Grid Sizes

The results of solving the heat equation for grid sizes N=100, 200, 400 are shown in Table 1. As expected, the error decreases by approximately 4x with each doubling of the grid size, which is consistent with the scheme outlined in the project description. Specifically, the error decreases from 0.000070 for N =100 to 0.000017 for N=200, and further to 0.000004 for N=400, representing factors of 4.12 and 4.25, respectively. This confirms that the implementation is functioning correctly, as a 4x decrease in error is expected when the grid size is doubled for such a method.

Part 2: Effect of Column Major Ordering

In column-major ordering, where elements in the same column are stored contiguously in memory, the nested for loops should have `j` as the outer loop and `i` as the inner loop. This loop ordering ensures efficient memory access, as each iteration through `i` accesses consecutive memory locations. By iterating over rows within each column, this takes advantage of spatial locality, reducing cache misses and improving performance. If the loops were reversed, the program would access non-contiguous memory locations, leading to poorer cache performance. This was also elaborated on in project 2, where the loop ordering had to be consistent with the way the matrix elements were stored.

Part 3: Parallelization for Nested Loops

Parallelizing the inner loop of nested for loops with OpenMP yields lower performance than parallelizing the outer loop. This is due to the fact that parallelizing the inner loop causes each thread to handle smaller tasks, which increases the overhead associated with thread synchronization. Also, the inner loop accesses non-contiguous memory locations in column-major ordering, which results in poor cache locality and more cache misses. Conversely, by

parallelizing the outer loop, more work is divided among the threads and better memory access patterns are maintained, which results in increased efficiency and quicker execution.

Part 4: Parallelization for Nested Loops using different threads

Based on the timings obtained for N=400, there was a significant reduction in execution time as the number of OpenMP threads increases. With 1 thread, the runtime is 116.51 seconds, which decreases to 7.34 seconds with 24 threads. However, while the performance scales well up to 12 threads (11.57 seconds), the improvement between 12 and 24 threads is smaller, showing diminishing returns. This decline in parallel performance efficiency is likely due to factors such as thread contention, memory bandwidth limitations, and increased overhead for managing more threads. As more threads are added, they compete for shared resources like cache and memory, which can reduce the scalability of the parallelization beyond a certain point.

Part 5: Scheduling Performance differences

The results show a performance difference between the static, dynamic, and guided scheduling strategies when using 24 OpenMP threads. Static scheduling is the fastest, with a time of 0.51 seconds, followed by guided scheduling at 0.64 seconds, and dynamic scheduling being the slowest at 0.71 seconds.

The way that each scheduling method divides up the work among threads is what causes the performance disparity. Because no additional scheduling decisions are required, static scheduling lowers overhead by distributing the iterations equally among the threads at the beginning. Better performance results from this when the workload is relatively constant throughout each repetition.

Work is divided into smaller segments as threads become available in dynamic scheduling, which adds overhead for task assignment and improves load balancing in the event that certain iterations take longer than others. It is slower than static scheduling because of this overhead.

Guided scheduling starts with larger chunks that decrease in size as threads finish, offering a compromise between static and dynamic scheduling. It balances load while reducing scheduling overhead compared to dynamic, but it's still slower than static because of the task management overhead.

Tables

Table 1

Errors for differing grid sizes.

Grid	Error
100	0.000070
200	0.000017
400	0.000004

Table 2

Time taken in seconds based on number of threads being run.

Threads	Time (seconds)
1	116
3	42
6	22
12	12
24	7

Table 3

Time taken in seconds based on different scheduling.

Scheduling	Time (seconds)
Static	.51
Dynamic	.71
Guided	.63