**Fall 2024**

**Parallel Programming and Algorithms**

**Multi-threaded parallel programming assignment**

|  |  |  |
| --- | --- | --- |
| **Problem** | **Full points** | **Points obtained** |
| **1** | **10** |  |
| **2** | **10** |  |
| **3** | **5** |  |
| **4** | **10** |  |
| **Total** | **35** |  |

**Objective**: The previous assignment was on static load balancing ideas. The objective of this assignment is to learn about dynamic load balancing in shared memory context. The assignments are on implementing two different methods of dynamically distributing tasks among threads.

A sample program called Tasks.cpp has been provided to get you started. This program shows a method “double compute( int i )” which is a sequential compute-intensive method. The run-time of this method is data-dependent (depends on value of i). This method is being called from the main method in a for loop. **This for loop of the main method can benefit from parallelizing the loop iterations among threads in a dynamic fashion.** Since you have already done static methods of load balancing, now you have to develop dynamic methods of load balancing. A pseudo-code has been provided below for you to work on dynamic workload load balancing.

**Problem 1)** Submit your final program in Dynamic.cpp

In dynamic method, all tasks are shared among threads using a queue. Each thread executes a chunk of the total tasks, and when a thread finishes a chunk, it requests another one from the queue.

Dynamic scheduling is shown below where each thread takes 2 tasks or loop iterations (among 32 loop iterations). There are 4 threads shown here. This is one possible distribution of tasks among four threads.

A grid with black squares

Description automatically generated

A sample pseudo-code is provided below to do load balancing among threads :

|  |  |
| --- | --- |
| Load balancing at the Master  Master thread  main()  {    main thread creates worker threads.    main thread creates a shared task queue Q.    main thread displays the partial and final results.  } | Load balancing at the workers  Worker threads  Each thread runs a while loop  while(1)  {     get a chunk of tasks from the shared task queue Q       if ( all tasks done )           break;     else          while( still more tasks pending )          {               process tasks;          }  } |

**Problem 2)** A new method of dynamic load balancing needs to be implemented here aka DecreasingChunks method. DecreasingChunks method of load balancing is also dynamic because each thread executes a chunk and requests another one when it’s finished. However, as chunks are completed, the size of the new chunks decreases. For 10,000 tasks and 2 threads, the table below shows the assignment of chunks to the threads with id 0 and 1. We see that the size of the chunk is approximately the number of iterations remaining divided by the number of threads. The first chunk has size 10000/2 = 5000, since there are 10,000 unassigned iterations. The second chunk has size 4999/2 = around 2500, and so on.

Test this load balancing strategy by parallelizing the tasks in the for loop of the main method provided in Tasks.cpp (same for loop as in the previous problem). Submit your program in DecreasingChunks.cpp.

**Example of a decreasing chunk size for 10,000 tasks (loop iterations) with 2 threads**

A table with numbers and a few words

Description automatically generated with medium confidence

**Problem 3) Compare the performance of the two solutions – simple dynamic load balancing vs Decreasing chunk methods.**

**Problem 4)** For this assignment, first you have to read the false sharing example from a pdf (a book chapter) [**falseSharing\_count3s.pdf**](https://umsystem.instructure.com/courses/272277/modules/items/8937091). This is shared in Canvas course modules under “C++ multi-threading”. In class, I covered the false sharing problem in the context of Matrix Vector Multiplication. This is described in the pdf [**falseSharing\_MatVec.pdf**](https://umsystem.instructure.com/courses/272277/modules/items/8937112) and my “False Sharing” slides. Both the pdfs are in Modules => C++ Multi-threading.

After understanding the "False sharing problem" in the context of counting 3s in parallel by multiple threads, explain the problem, how the problem manifests itself in experiments and provide a potential fix to this false sharing problem. There are three parts in this question. Total points = 4 + 4 + 2 = 10 points.