Homework #2 – OpenMP Programming

Yoke Kai Wen, 2020280598

March 5, 2021

1 Introduction

In this assignment, I implemented a parallellised version of the **PageRank** graph algorithm on a multi-core CPU using OpenMP.

2 Brief description of implementation

I added OpenMP directives before three `for` loops in `page_rank.cpp`.

1. The first directive, `#pragma omp parallel for schedule (static)`, was added before the `for` loop initialising each graph node.
2. The second directive, `#pragma omp parallel for reduction(+:broadcastScore) schedule (guided)`, was added before the `for` loop computing the new scores for each node in one iteration.
3. The third directive, `#pragma omp parallel for reduction(+:globalDiff) schedule (static)`, was added before the `for` loop computing how much the scores for each node has changed after that iteration.

Note that the `reduction` clause was included in the second and third directives because their `for` loops involved the successive accumulation of value in a variable shared across different threads and iterations of the `for` loop. I also experimented with different OpenMP schedule strategies at each directive, with time results shown in the next section.

3 Time results

Table 1 shows the time results across different OpenMP schedule strategies and different number of threads used. Note that 'static' strategy indicates that all three directives use static scheduling,while 'static-guided-static' means that the first, second and third directives use static, guided and static scheduling respectively.

	Number of threads				
Schedule strategy	**1**	**2**	**3**	**4**	**8**
No schedule	11.593	6.134	4.855	3.288	-
static	12.140	6.348	5.090	3.501	-
dynamic	14.639	11.765	9.678	8.176	-
guided	12.732	5.868	3.906	2.895	2.465
static-dynamic-static	14.160	8.620	6.485	5.117	-
static-guided-static	*11.672*	*5.838*	*3.855*	*2.901*	*2.466*

Table 1: Table showing time results (seconds) of different implementations with different threadnums

3.1 Comparison across different thread numbers

The greater the number of threads, the greater the extent of parallelisation, and the shorter the time required to run the same piece of code. In fact for the most optimal schedule strategies, 'no schedule', 'guided' and 'static-guided-static', the time taken is almost proportional to the number of threads used (for 1-4 threads) i.e. there was a speedup of 1,2,3,4 times corresponding to the number of threads. This suggests that the `page_rank.cpp` code was almost 100% parallelisable with very little sequential component.

3.2 Comparison across different OpenMP schedule strategies

The default OpenMP `parallel for` schedule seems to be static, as seen from the similar time results for 'no schedule' and 'static' strategies. Any implementation involving 'dynamic' scheduling performs poorly, due to high overhead from decision making about which thread gets which chunk of iterations. Guided scheduling performs the best as it can handle imbalanced loads with less overhead, especially at the second OpenMP directive computing the scores for each graph node which has widely varying workload at each node.

However, I was surprised that 'static-guided-static' did not perform significantly better than the 'guided' implementation. I thought that static scheduling would be the most optimal for the first and third directives, which were placed at `for` loops involving uniform work load at each iteration (initialising each node score, computing difference in score at each node), but it seems like it makes not much of a difference if static or guided scheduling was used in the first and third directive. This is probably because the processing workload at the first and third directives was also small compared to the workload at the second directive, and hence the scheduling only matters significantly for the second directive.

4 Discussion on performance loss (speedup vs threadnum)

The speedup for 4 threads for implementations with guided scheduling was near optimal which indicates almost 100% parallelisability of the code, but for this section I will be discussing other non-optimal scheduling strategies that suffered significant performance loss.

4.1 Performance loss with static scheduling

The second directive involves relatively heavier calculations of scores at each node, with calculation complexity per iteration depending on number of neighbouring nodes of each node. Nodes with more neighbours would require more work, while static scheduling assigns the same amount of work to each thread even though some threads do more work than the others. Therefore, this results in sub-optimal performance.

4.2 Performance loss with dynamic scheduling

While dynamic scheduling is able to flexibly assign varying workload to each thread which works well when workload is highly variable per iteration, it comes at the cost of higher overhead to decide which thread gets each chunk. In the case of the `com-orkut_117m.graph` input, the cost of overhead outweighs by far the non-uniformity of the workload per iteration. Perhaps, if the input graph dataset has drastically varying density for each node, dynamic scheduling could perform better than guided scheduling.

5 Effects of hyperthreading on performance

The server cluster node provided has hyperthreading, with each physical core divided into two virtual cores resulting in two threads. Since there are four cores, this results in 8 threads on the node. In Table 1, I recorded the time taken for 8 threads for the optimal implementations. Indeed, there is significant performance improvement compared to 4 threads, but nowhere near the expected 8x speedup. This is because compared to a real physical core, the virtual logical core only has about 30% of the actual performance, but it still offers greater parallelism and hence reduces the running time of the code but not as much as 8 physical cores would.