B160509

Threaded Programming coursework II: the affinity schedule for scheduling the OpenMP loop construct

Abstract

Keywords: Scientific programming, parallelization, performance optimization, OpenMP

1. Introduction

OpenMP version 4.5 supports various scheduling options for its loop construct, for example `static`, `dynamic` or `guided` (see OpenMP Architecture Review Board, 2015, Chapter 2). This paper presents an alternative schedule, called the affinity schedule. The affinity schedule combines some properties of the three above mentioned scheduling options into one schedule.

This paper is a follow-up of a benchmark presented in B160509 (2019). B160509 (2019) presents a scientific program written in the Fortran programming language, containing two loops performing matrix and vector operations. These two loops were parallelized using built-in scheduling options of OpenMP version 4.5 and then benchmarked in order to determine the best schedule for the two loops. The here presented affinity schedule is benchmarked the same way, as are the built-in schedules in B160509 (2019).

This paper begins by describing two versions of the affinity schedule. Afterwards the benchmark is described and its results are presented. The benchmark of the affinity schedule is then compared to the best schedule for both loops determined in B160509 (2019). At last the results are discussed and a conclusion is drawn.

2. Method

Like stated in the previous chapter, the affinity schedule combines properties from the already built-in schedules `static`, `dynamic` and `guided`. Let n be the amount of iterations of the loop the instance of the affinity schedule is applied to and let p be the amount of OpenMP threads. The affinity schedule splits the n iterations of the loop into p splits, all having approximately $\frac{n}{p}$ iterations. This is the same way the `static` schedule splits the iterations, if no chunk size is provided (see OpenMP Architecture Review Board, 2015, Chapter 2).

Every split is owned by one OpenMP thread. But, unlike the `static` schedule, the split is not executed as a whole. Rather, it is split again into smaller chunks, which are executed after another. The size of the chunks gets smaller, the more iterations of the split are already executed:

$$\mathit{chunk\ size} := \lceil \mathit{remaining\ iterations} \cdot p^{-1} \rceil.$$

The decreasing chunk sizes are a property of the `guided` schedule. While the `guided` schedule takes all n iterations and dynamically assigns the splits in a first come first serve order to the OpenMP threads (like the `dynamic` schedule), the affinity schedule uses the same property of decreasing chunk size, just local to each split (see OpenMP Architecture Review Board, 2015, Chapter 2).

Once a thread has finished all chunks of the split it owns, it is not simply idle until all threads have finished their splits. Instead, the thread determines the split that has still the most iterations left and takes the next chunk from it. Therefore, once the owned split of a thread is finished, the thread changes from behaving like it is part of the `guided` scheduling strategy on the local split to being `dynamic` in the fact that the split with the most iterations left is dynamically executed by all threads that have finished their split, plus the owner thread.

The affinity schedule therefore combines the strengths, concerning the execution speed, of the aforementioned built-in schedules over the phases of its execution. It starts executing like the `static` schedule, which produces very low overhead of synchronization (in fact none, but the affinity schedule must synchronize access to the split), moving over to a `dynamic` scheduling strategy, which has a higher overhead of synchronization, but produces no idle threads. This effectively reduces the synchronization overhead of the `dynamic` schedule while also removing the idleness of threads during the execution with the `static` schedule.

3. Results

4. Discussion

5. Conclusion

References

B160509. Threaded Programming coursework I: benchmarking OpenMP schedules, 2019.

OpenMP Architecture Review Board. OpenMP application program interface version 4.5, 2015. URL `https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`.