Threaded Programming coursework: benchmarking OpenMP schedules

Jonas Fassbender jonas@fassbender.dev

Abstract

Keywords:

1. Introduction

This paper documents the results of a benchmark performed on a scientific program. The program is written in the Fortran programming language and performs element-wise computations on matrices (using nested do-loops, not array operations). It contains two of these matrix operations—in this paper called critical sections.

Both critical sections are suitable for speeding up with OpenMP's loop construct, distributing the computation on multiple threads of execution. The loop construct provides the `schedule` clause, which determines the divide of the iterations of the loop among the OpenMP threads (see OpenMP Architecture Review Board, 2015, Chapter 2.7.1).

The benchmark consists of two phases. The goal of the first phase is to compare different scheduling options of the OpenMP library and how they effect the execution speed of the two critical sections of the program. The second phase provides data on how well the fastest scheduling options for both critical sections scale with different amounts of threads.

OpenMP version 4.5 was used and the benchmark was performed on the backend of the Cirrus supercomputer (see OpenMP Architecture Review Board, 2015; EPCC, 2019). The program was compiled with the Intel Fortran Compiler (`ifort`) version 17.0.2 (see Intel, 2016).

First, this paper describes the conducted benchmark, before presenting the results. At last the results are discussed and a conclusion is drawn.

2. Experiment

This chapter will describe the performed benchmark. First the two critical sections are described, followed by a description of the benchmark.

Let $A : n \times n$ and $B : n \times n$ be two matrices, $A, B \in \mathbb{R}^n \times \mathbb{R}^n$. Let $A(i, j)$ be the element of A in the ith row and the jth column. Every element in A is initialized to 0 and every element in B is set according to: $B(i, j) = \pi(i + j); i, j = 1, \ldots, n$.

The first critical section updates A:

$$A(i,j) = A(i,j) + \cos(B(i,j)); i,j = 1,\dots,n. \tag{1}$$

Both critical sections are executed multiple times, which is the reason $A(i,j)$ on the right-hand side of (1) can not be substituted to 0.

For the second critical section, let $x = n^{-2}$ and $\vec{c}$ be the zero vector of size n. Let $\vec{j}_{\max} \in \mathbb{R}^n$ be another n-sized vector. $\vec{j}_{\max}$ is set to:

$$i = 1,\dots,n : \vec{j}_{\max}(i) = \begin{cases} n & \text{if } i \bmod \lfloor \frac{i}{10} \rfloor + 1 = 0 \\ 1 & \text{if } i \bmod \lfloor \frac{i}{10} \rfloor + 1 \neq 0 \end{cases}.$$

The matrix B' is set to $B'(i,j) = (ij+1)n^{-2}; i,j = 1,\dots,n$.

The second critical section updates $\vec{c}$:

$$\vec{c}(i) = \sum_{j=1}^{\vec{j}_{\max}(i)} \sum_{k=1}^{j} \vec{c}(i) + kx \ln(B'(j,i)), i = 1,\dots,n. \tag{2}$$

Since both (1) and (2) are element-wise independent, the computation of every element can be distributed over multiple processes.

The benchmark consists of two phases. In the first phase, different scheduling options are compared using four threads and the fastest scheduling option for each critical section is determined. The second part of the benchmark tests how well the fastest scheduling options scale with different amounts of threads. For the benchmark n was set to 729.

The different scheduling options used in the first phase are:

- Auto
- Static
- Static, Dynamic, Guided, all with different chunk sizes of: 1, 2, 4, 8, 16, 32, 64

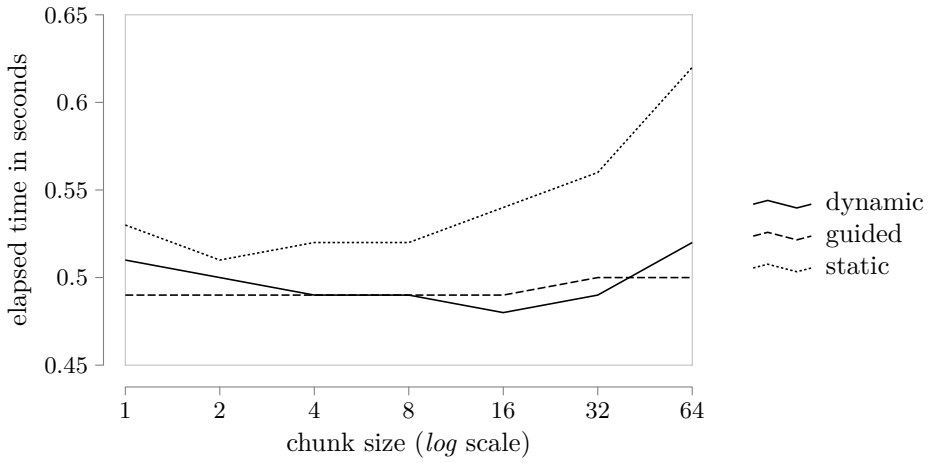
The fastest scheduling options for the critical sections are then run with 1, 2, 4, 6, 8, 12 and 16 threads during the second phase of the benchmark.

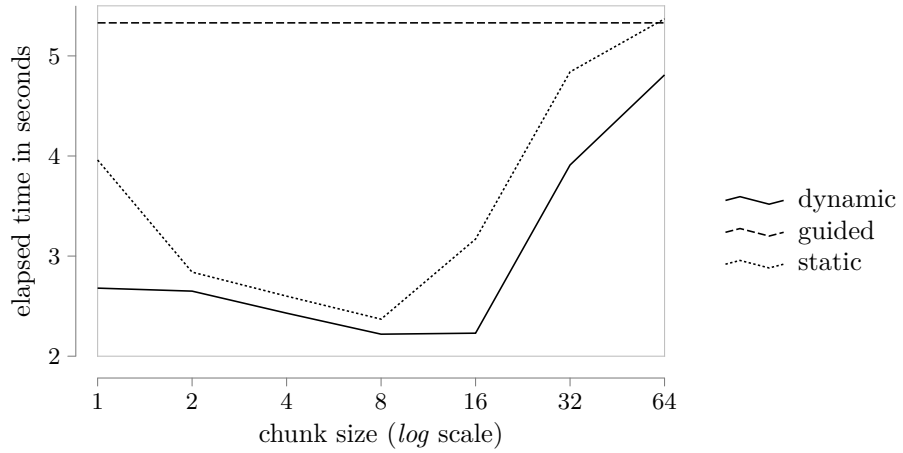
Like stated in the introduction, both benchmark phases are executed on the Cirrus backend with exclusive access to one node.

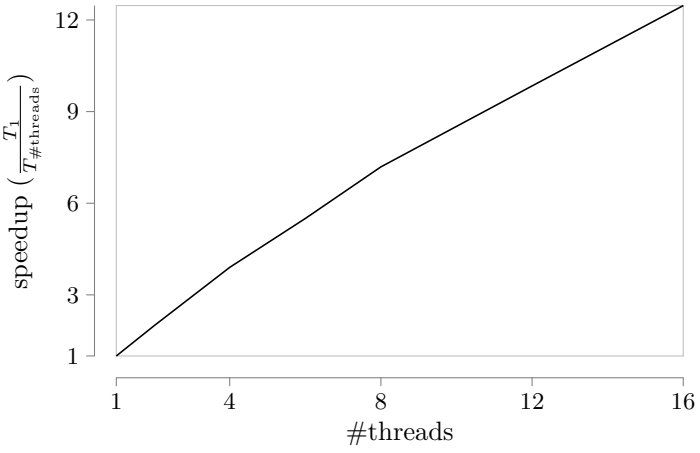
3. Results

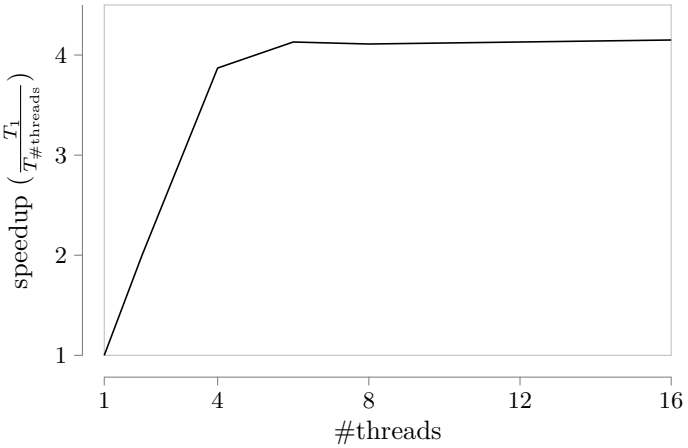
schedule	loop 1		loop 2	
	mean	median	mean	median
Sequential	1.62	1.61	8.57	8.56
Auto	0.49	0.48	5.32	5.32
Static	0.83	0.84	6.18	6.19
Dynamic, 1	0.51	0.51	2.68	2.57
Dynamic, 2	0.50	0.49	2.65	2.57
Dynamic, 4	0.49	0.49	2.43	2.39
Dynamic, 8	0.49	0.48	**2.22**	**2.22**
Dynamic, 16	**0.48**	**0.48**	2.23	2.23
Dynamic, 32	0.49	0.48	3.91	3.91
Dynamic, 64	0.52	0.51	4.81	4.81
Guided, 1	0.49	0.49	5.33	5.34
Guided, 2	0.49	0.48	5.33	5.33
Guided, 4	0.49	0.49	5.33	5.33
Guided, 8	0.49	0.48	5.33	5.33
Guided, 16	0.49	0.48	5.33	5.33
Guided, 32	0.50	0.49	5.33	5.33
Guided, 64	0.50	0.49	5.33	5.33
Static, 1	0.53	0.53	3.96	3.93
Static, 2	0.51	0.51	2.84	2.81
Static, 4	0.52	0.52	2.60	2.57
Static, 8	0.52	0.52	2.37	2.37
Static, 16	0.54	0.53	3.17	3.18
Static, 32	0.56	0.56	4.84	4.84
Static, 64	0.62	0.63	5.37	5.38

#threads		1	2	4	6	8	12	16
loop 1	mean	1.87	0.94	0.48	0.34	0.26	0.19	0.15
	median	1.87	0.93	0.48	0.34	0.26	0.18	0.14
loop 2	mean	8.59	4.30	2.22	2.08	2.09	2.08	2.07
	median	8.59	4.30	2.22	2.08	2.10	2.08	2.07


0.65
0.6
0.55
0.5
0.45
elapsed time in seconds
1
2
4
8
16
32
64
chunk size (log scale)
dynamic
guided
static


5
4
3
2
elapsed time in seconds
1
2
4
8
16
32
64
chunk size (log scale)
dynamic
guided
static


12
9
6
3
1
speedup ($\frac{T_1}{T_{\#\text{threads}}}$)
1
4
8
12
16
#threads



4. Conclusion

References

EPCC. Cirrus, 2019. URL https://www.cirrus.ac.uk.

Intel. Intel Fortran Compiler 17.0 for Linux, 2016. URL https://software.intel.com/en-us/articles/intel-fortran-compiler-170-for-linux-release-notes-for-intel-parallel-studio-xe-2017.

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