Final Report

Parallel Computing for Computational Mechanics

Johannes Grafen 380149
johannes.grafen@rwth-aachen.de

Abstract: Lorem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut lre et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1 Introduction

The parallelisation of code for the simulation of complex mechanical systems is a key-topic in modern computational mechanics. Constantly increasing computing power of high-performance clusters allow for the simulation of large and complex systems. This requires an efficient parallelisation approach, where different parallelisation paradigms can be of use.

In this report the parallelisation of a finite element code for the simulation of the stationary temperature distribution on a two-dimensional disk is investigated. For the shared memory approach the OpenMP library is used, where as the distributed memory approach is assessed by using a one-sided parallelisation approach with MPI. The different parallelisation approaches are discussed and compared for different number of CPUs. This report is structured as follows. In the second chapter *Theory and Methods* are presented. The *Implementation and Validation* of the presented problem is discussed in chapter 3. The results of the optimisation of the runtime of the code is illustrated in chapter 4 *Results* and the assessment of the different optimisation techniques is discussed in chapter 5 *Discussion*. Finally, this report is closed with a *Conclusion* in chapter 6.

serial optimisation is performed by

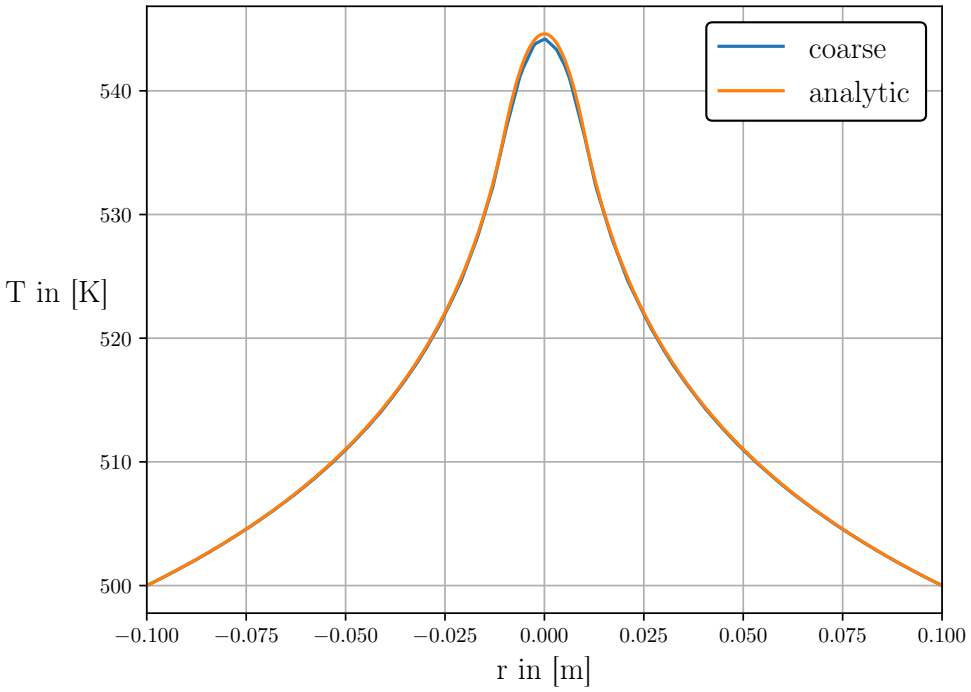


Figure 1: Comparison of the steady-state temperature distribution of the analytical solution (refEQ) and the numerical solution on the coarse mesh

compiler flag	time 1	time 2	time 3	average time
-O0 (no optimisation)	233.71	241.5	243.02	239.41
-O1	62.41	63.71	62.76	62.96
-O2	56.96	55.39	57.11	56.49
-O3	55.49	55.23	57.8	56.17
-O3 -axSSE4.2, SSSE3, SSE2 -fp-model fast=2	56.17	56.21	56.46	56.18
-O3 -fp-model fast=2	57.69	56.19	57.89	57.26
-O3 -ipo	54.57	57.26	60.38	57.40
-unroll	55.17	57.01	55.94	56.04
-Ofast	55.43	56.17	56.27	**55.96**

Table 1: Timings of the various compiler flags for serial optimisation

2 Theory and Methods

3 Implementation and Validation

Parallelisation with OpenMP

Parallelisation with MPI

4 Results

4.1 Serial Solver

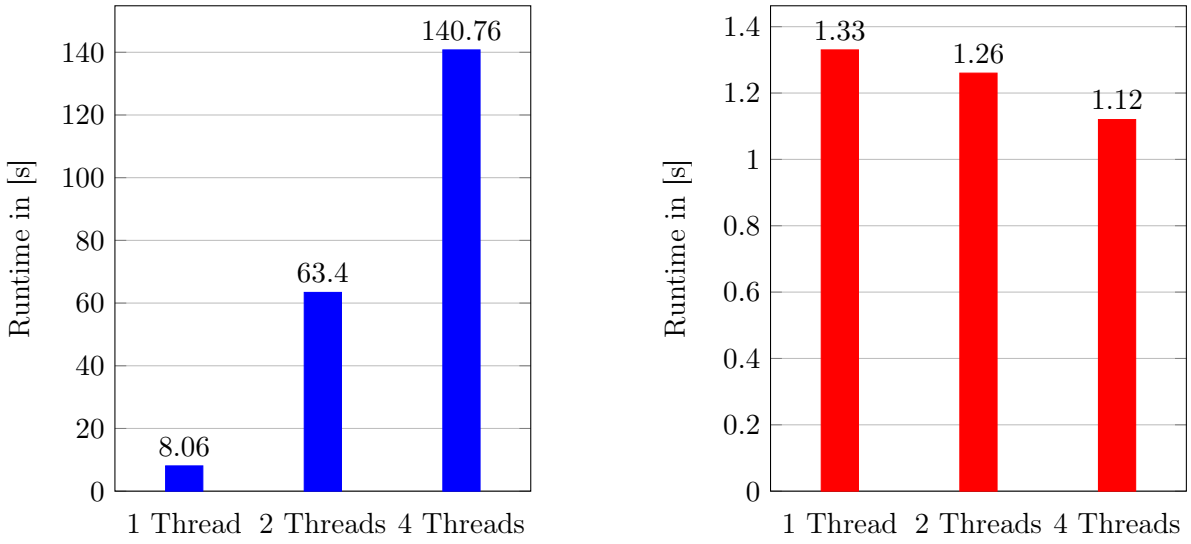
R1 a

R1 b

- debugging - the code can't be properly debugged as the compiler changes complete code structure

4.2 Parallel OpenMP Solver

R2 a



(a) OpenMP task A

(b) OpenMP task B

Figure 2: Timings for solver with different OpenMP optimisations using the coarse mesh

R2 b

R2 c

R2 d

- look into presentation

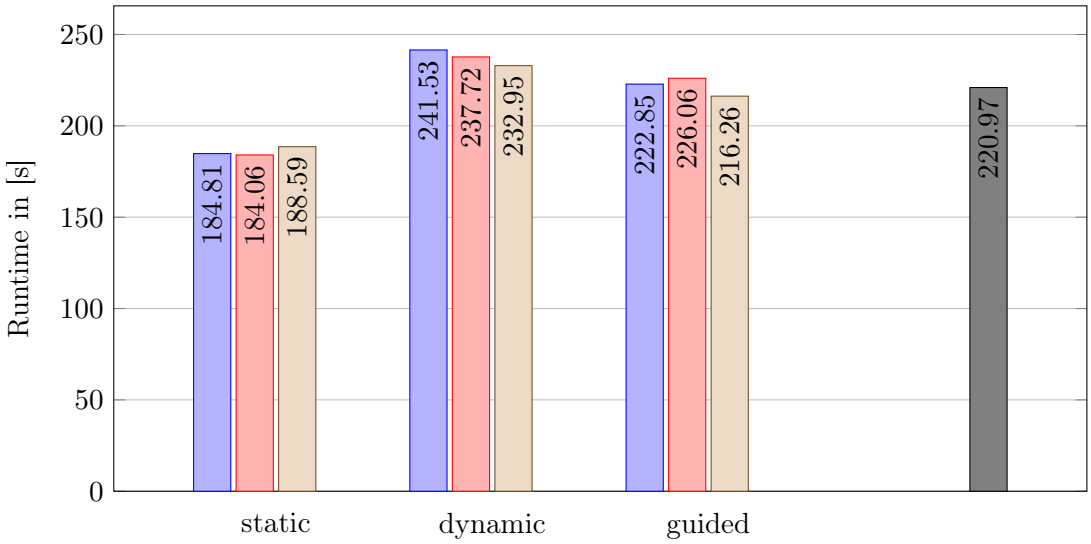
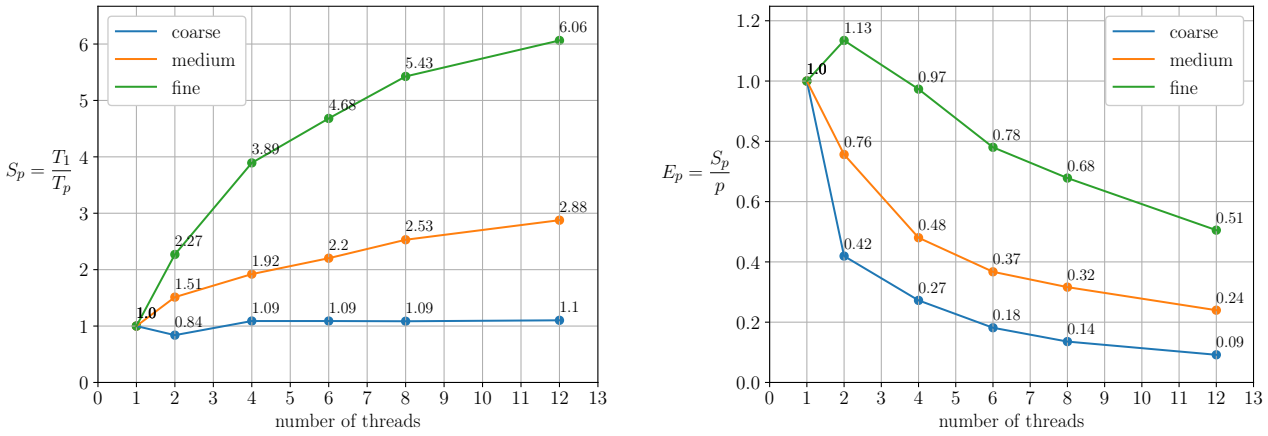


Figure 3: Comparison of different scheduling options (static, dynamic, guided, auto) for different chunk sizes(128, 256, 512) in first parallelised loop for 4 threads using the finest mesh



(a) Speed-up **(b)** Efficiency

Figure 4: Speed-up and efficiencies for coarse, medium and fine mesh for different numbers of threads

4.3 Parallel MPI Solver

R3 a

R3 b

R3 c

R3 d

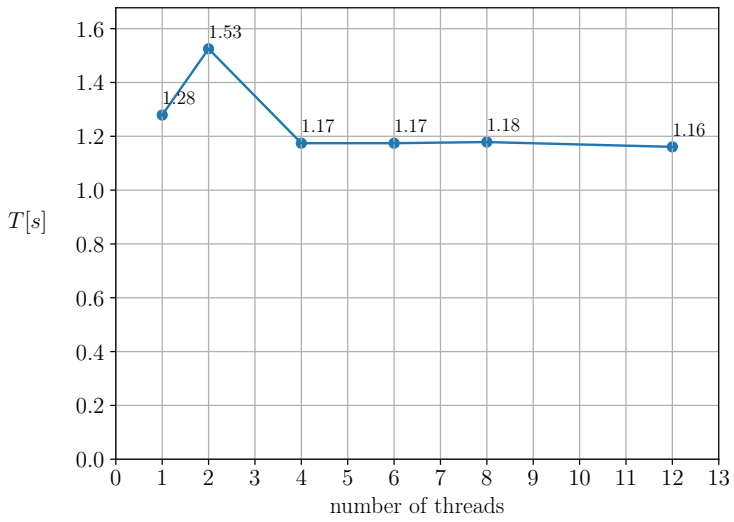
5 Discussion

6 Conclusion

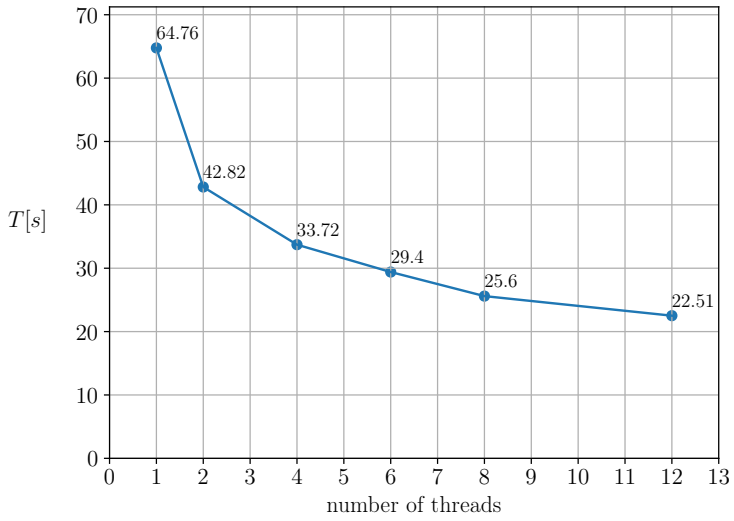
Lorem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut

lre et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

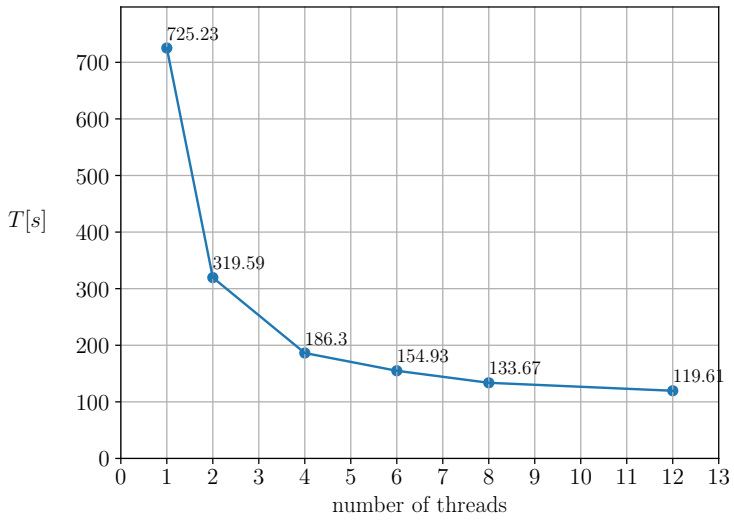
References



(a) coarse mesh

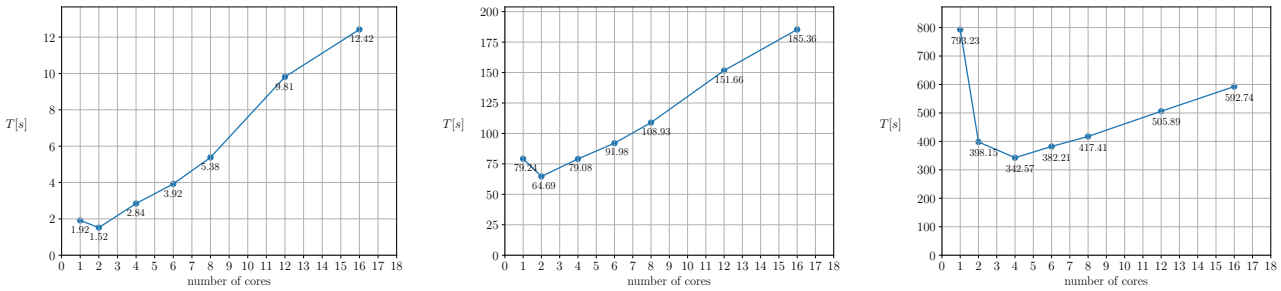


(b) medium mesh



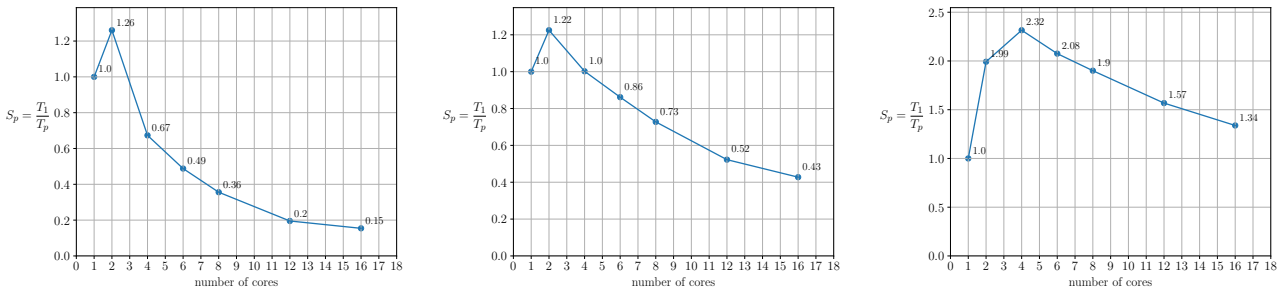
(c) fine mesh

Figure 5: Runtime for coarse, medium and fine mesh for different number of threads



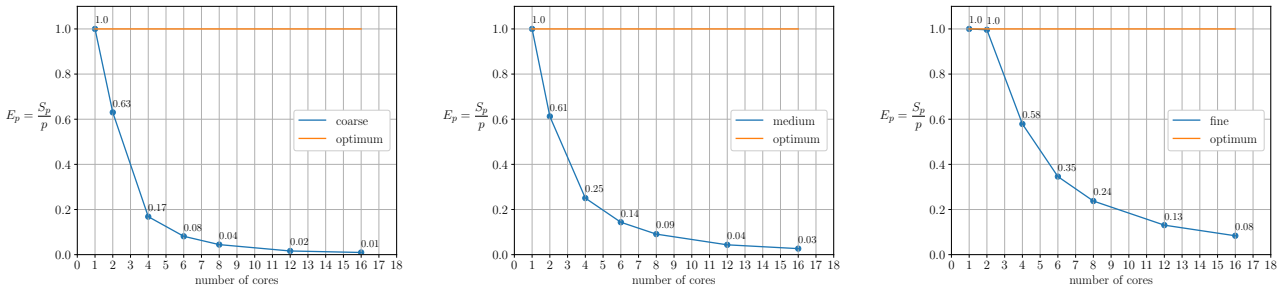
(a) coarse mesh **(b)** medium mesh **(c)** fine mesh

Figure 6: Runtime for coarse, medium and fine mesh for different number of cores



(a) coarse mesh **(b)** medium mesh **(c)** fine mesh

Figure 7: Speed-up for coarse, medium and fine mesh for different number of cores



(a) coarse mesh **(b)** medium mesh **(c)** fine mesh

Figure 8: Efficiency for coarse, medium and fine mesh for different number of cores

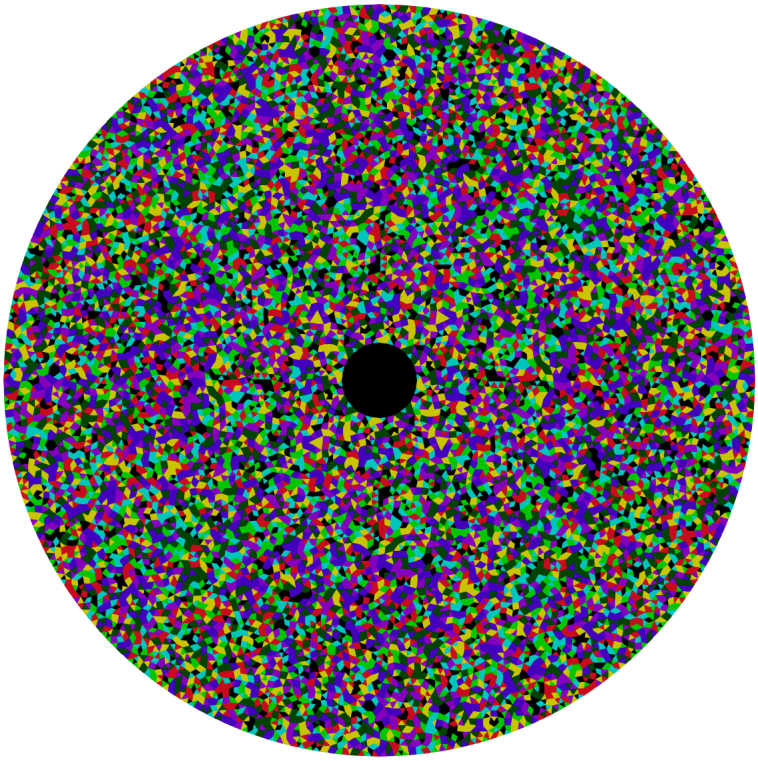


Figure 9: Partitioning of medium mesh using 8 cores