Abstract

The contribution proposes efficient computational strategies for large scale numerical simulations of the shield tunnelling process using the finite element method. In the simulation, the ground, the tunnel lining, the shield machine, and the support measures are modelled as distinct components. Their interactions are accounted for by means of contact conditions and other constraints. The material model of the soil considers the soil skeleton, pore water and air as separate material phases. The numerical characteristics of the presented model imposes particular challenges on the solvers used to solve the linearised system equations. The coefficient matrices lack some properties that are vital to many iterative solvers, such as positive definiteness, symmetry, and low condition numbers.

To reduce computing times, several parallelisation techniques have been investigated. A parallelisation concept for distributed memory computers by means of the Trilinos libraries is presented as well as an openCL-based implementation for GPGPU systems using the ViennaCL framework. A comparative study reviews the parallel efficiency of different linear solvers and assembling procedures for the finite element model with respect to both speedup and applicability for the simulation of highly demanding numerical models in geotechnics and tunnelling.

In order to handle the spatial search involved in the solution of contact conditions, the standard domain decomposition scheme has been adopted. An additional, dedicated contact domain is specified that contains all nodes in the contact surfaces. A two-staged decomposition allows for dissimilar decompositions for the assembling and the solving phase. Thus, arbitrary linear solvers can be applied regardless of the partition method used to decompose the model for the assembling process.

Keywords: distributed memory, GPGPU, parallelisation, simulation, finite element method, tunnelling

1 Introduction

Mechanised tunnelling is the most wide-spread tunnelling method today. Its flexibility allows to construct tunnels in various conditions including difficult ground conditions as well as tunnels in vulnerable urban areas, often characterised by low overburden and sensitive existing buildings. The tunnelling process, however, is associated with considerably high risks. In particular, changing and unforeseen ground conditions may require to adapt the original design and to react in a suitable manner with a readjustment of process parameters. Furthermore, surface settlements induced by the tunnelling process may lead to damage of existing infrastructure and buildings in the vicinity of the tunnel.

Numerical simulations can provide valuable insight in the behaviour and the interactions of the excavation process and its environment. These simulations, however, are characterised by complex and large numerical models that are expensive to compute and may have to be run multiple times in order to study different design or condition variants. For this reason, it is inevitably important to minimise the computational effort and the time needed to run such simulations.

Parallelisation of the simulation software is a prerequisite to achieve the goal of providing accurate predictions of a complex construction process in reasonably short computation times. In this contribution parallelisation concepts and their implementation in a recently developed numerical simulation software for the mechanised shield tunnelling process [1] are presented. Particular focus is put on the measures taken to overcome problems in the parallel efficiency and in the parallelisation strategy as such in view of the specific features of the computational model.

2 Simulation model for shield tunnelling

The mechanised shield tunnelling process is characterised by a continuous support of the excavated ground by means of a steel shield. Within the shelter of this shield, the ground is excavated by means of a revolving cutting wheel and the tunnel lining is constructed from precast concrete segments. By this, the excavated tunnel is sealed against inflowing water and ground material at any time during the complete construction process. At the heading face, the ground needs to be supported by appropriate measures to prevent failure and, hence, uncontrolled inflow of the material. This is usually realised by means of a bentonite support liquid (hydro shield) or by a pressurised muck that consists of the excavated material (earth pressure balanced shield). The construction follows a repeated sequential order where, in the excavation phase, the shield is pushed forward by means of hydraulic jacks that are mounted on the shield and use the recently completed ring of lining segments as counter bearing. After the length of one lining ring has been excavated, the machine is stopped and the hydraulic jacks are retracted to allow for the erection of the next lining ring. The gap that arises between the ground and the lining tube is grouted with mortar to ensure the bedding of the lining and to prevent settlements.

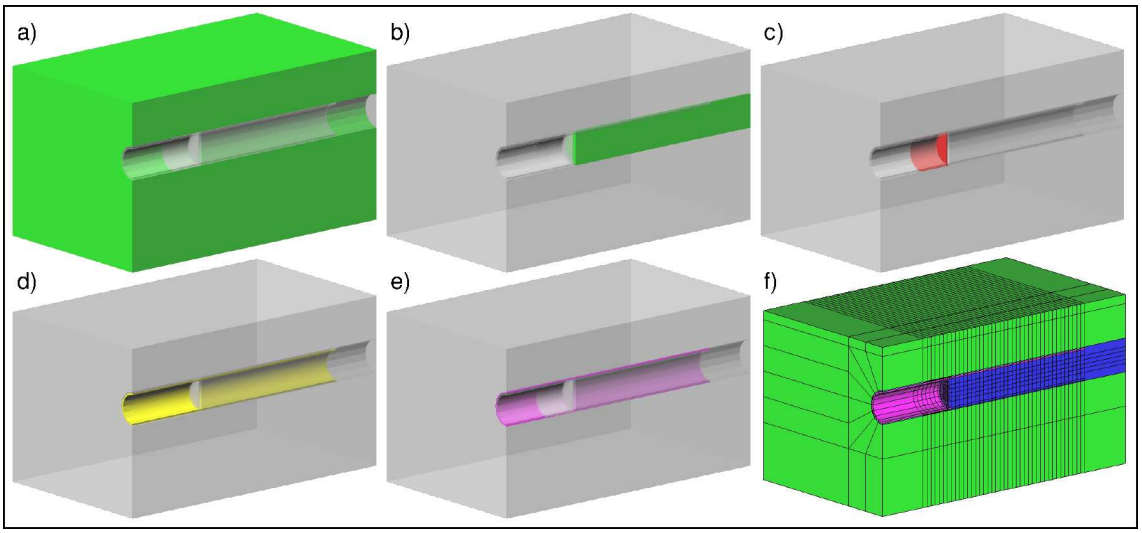

Figure 1: Components of the simulation model for shield tunnelling: a) ground model; b) excavation volume; c) shield model; d) segmented lining; e) tail gap grouting; f) complete discretised model

The numerical model for the simulation of the shield tunnelling process allows for a transient, process-oriented simulation of all relevant components of the excavation, the lining installation and process-related impacts on the environment. The shield machine is represented as a separate model component that is capable to move freely in the ground. The model components and their interactions with each other are represented by various sub-models that are specifically designed to represent the system behaviour of the tunnel construction process. These sub-models are: the shield machine, the surrounding ground, the segmented lining, the hydraulic thrust jacks, the heading face support and the tail gap grouting (see Fig 1).

Each of these sub-models is characterised by specific features that are non-standard in the context of the finite element method and that need to be addressed with respect to parallelisation of the simulation software. The shield machine is modelled as a deformable body representing the main load bearing components of the shield and the machinery. Its skin is in frictional contact with the surrounding soil. For this purpose a surface-to-surface contact algorithm, based on a formulation presented in [2], is implemented in the simulation software that allows to model also the flow of process fluids (grouting mortar and support liquids) in the steering gap between the shield and the ground [3]. This contact algorithm requires spatial search over the complete contact domain in order to select points in opposing contact surfaces that can be linked with each other. The ground model is characterised by a three phase formulation in the framework of the theory of porous media [4] that considers water and air as separate fluid phases in a porous, solid soil skeleton to model partially saturated ground conditions. Here, the coupling of fluid and solid mechanics as well as the numerical properties of a nonlinear constitutive model for the soil skeleton impose severe challenges

to the linear solver used to solve the system equations. In particular, ill-conditioned, non-symmetric matrices have to be considered. The segmented lining and the shield machine are coupled by means of truss elements representing the hydraulic thrust jacks. These truss elements are not connected to the nodes but to the faces of the respective lining and shield elements. Therefore, a tying algorithm has been employed that undergoes the same spatial search algorithm as the contact algorithm. For the heading face support, either mechanical or pressure boundary conditions are applied to model both slurry shields and EPB shields. To model an impermeable filter cake that evolves during standstill of the machine in the lining construction phase and that is destroyed in the excavation phase, these boundary conditions are applied in an alternating way. Here, different permeabilities of the soil contribute ill-conditioning of the system matrix. Finally, the excavation of the ground as well as the installation of the lining and the tail void grouting are implemented by means of de- and reactivation of elements during the simulation procedure. This fact has to be dealt with properly also in the case of distributed memory parallelisation using domain decomposition methods.

The simulation model is implemented in the object-oriented multi-physics finite element framework KRATOS [5]. The aim of KRATOS is to allow for an easy, yet efficient, implementation of arbitrary finite element formulations and algorithms for the solution of coupled problems. For this purpose, all interfaces have been designed such that internal data handling, including the communication processes required for parallelisation, are encapsulated. Through this software design, it is possible to use all features of KRATOS in arbitrary combinations without the need to implement the same algorithms in different routines.

3 Parallelisation concept

For the parallelisation of the simulation software three different concepts have been applied: shared memory parallelisation by means of the openMP framework, distributed memory parallelisation using MPI and domain decomposition methods, and parallel solving on graphics processors (GPGPU) using the openCL interface. The basic idea of the parallelisation concept is to keep the basic implementation of the finite element technology independent from the parallelisation method. Thus, an element, a constitutive law or auxiliary algorithms such as time integration schemes or the contact formulation only has to be implemented once and works in all parallel setups. This requires, however, a number of specific measures in the design of the software framework as well as in the parallelisation concept that are explained in the following.

3.1 Shared memory parallelisation

The parallelisation on shared memory computers follows a straight forward implementation scheme frequently used in applications that run on multi-core computers:

all loops that can be trivially parallelised are marked with a respective compiler directive (pragma) to allow for automatic parallelisation by openMP. In the case of the finite element simulation software employed for the simulation of the shield tunnelling process, this approach has been applied for the assembling of the system matrix and in the linear solvers. During the assembling phase, all elemental contributions to the global stiffness matrix and the load vector are computed element-wise in parallel, including the computation of the material response and all contributions from the contact and tying algorithms. By this, approximately half of the workload of the complete program has been parallelised. The second large part of the overall workload is the solution of the system of linear equations. Here, parallel linear solvers have been employed. Since on shared memory systems all stored variables are accessible by each processor, no particular measures have to be taken to ensure the applicability of the parallelisation to deactivated elements, spatial search algorithms or the setup of the system. It has to be noted, though, that the allocation of memory follows the distribution of workload to the different processors in order to prevent communication bottlenecks that arise from the ccNUMA architecture used in many shared memory computers. This is achieved by partitioning the nodes container according to the number of processors used. This split container is then used to allocate the memory for the global system equations. By this, each partition of the shared variables is allocated in the local memory of a different processor. While the global system of equations is assembled in course of the simulation, the memory access is then distributed among all processors thus preventing the access by all processes to the local memory of one single processor.

3.2 Distributed memory parallelisation

While the parallelisation concept for shield tunnelling simulations on shared memory computers has already been presented in [6], an extension for distributed memory computers has been recently developed and is described in the following. On distributed memory computers, domain decomposition methods are applied to divide the complete model into multiple parts of similar workload to be assigned to each processor. In this case each processor has access to its private subdomain only. To solve the interface problem such that a valid solution for the complete domain is achieved, all processors communicate by means of message passing. This renders the solution of contact problems difficult, if bodies that are in contact are not in the same subdomain. For this reason, all contact surfaces of the complete model are assigned to one particular subdomain. While the number of nodes in the two-dimensional contact surface is small compared to the number of nodes of the three-dimensional simulation model, this does not yield an imbalanced workload. A second issue to be addressed is the handling of deactivated elements. The deactivation of elements is controlled by a flag that determines whether an element contributes to the global system matrix. This flag can be controlled independent from the domain decomposition.

All communication between the processes is carried out by means of communica-

tor objects that are inherently connected to each mesh entity (nodes, elements, conditions). By this, the algorithmic implementations are identical for the shared memory and the distributed memory versions of the software. Thus, no additional maintenance effort needs to be taken to keep the software coherent. Since the model is decomposed already in the initialisation phase, each processor is related to only one subdomain of the complete model. During the assembling of the global system of equations, each sub-model is assembled separately, but stored in a distributed data structure using the Epetra library [7]. Epetra matrices are stored in a distributed manner, yet appear as a monolithic data structure. All communication that is necessary to handle cross-process access to these resources are encapsulated in the Epetra library such that no communication functions have to be directly implemented in the finite element code. A second advantage of this approach is that the domain decomposition of the global stiffness matrix may be independent of the decomposition of the mesh. As a consequence, for both the assembling and the solving process, optimal domain decompositions can be applied.

For the simulation of tunnelling problems, an iterative GMRES solver is applied to solve the global interface problem in parallel. As a preconditioner to this iterative solution, the matrices arising from the local sub-domains are treated with a block LU decomposition solver (KLU, [8]).

3.3 GPGPU parallelisation

A more recent approach to exploit parallelism is to use general purpose graphics processors (GPGPU). These processors are massively parallel and well suited for floating-point computations. Wide-spread iterative linear solvers such as CG, BiCGStab or GMRES repeatedly perform sparse matrix-vector products. This operation can be conducted row-wise independently and is therefore applicable for massive parallelisation on GPGPUs. By means of the generic programming interface openCL [9], linear solvers and other parallel algorithms can be conveniently implemented to run on graphics processors. In KRATOS, this has been realised employing the ViennaCL library [10] that provides data types for sparse matrices and vectors as well as implementations of level 1 and 2 BLAS operations.

As can be seen from the benchmarks presented in Section 4, the performance of the implemented preconditioned iterative solvers is good for basic structural problems. However, the challenging properties of the coefficient matrices arising from shield tunnelling problems require specific preconditioners that are hard to implement efficiently on GPGPU architectures due to their algorithmic formulation and memory requirements. In particular, the lack of heap allocation on runtime in the openCL framework renders this a difficult task at present.

4 Benchmark applications

The performance of the GPGPU solvers is demonstrated by means of a benchmark example related to a linear structural analysis in Subsection 4.1. Subsection 4.2 shows the effect of an optimised allocation strategy for ccNUMA architectures on the speedup of the assembling process. To demonstrate the parallel performance of the presented simulation software, an exemplary simulation of a tunnel excavation has been computed using both the shared memory and the distributed memory version of the software. The results are presented in Subsection 4.3.

4.1 Benchmark 1: Elastic analysis of a Cooling tower

As a simple benchmark structure, a cooling tower shell (see Fig. 2) has been analysed using all parallelisation approaches available in KRATOS. A comparison of turnover times for different linear solvers are illustrated in Fig. 3. Each of the investigated solvers has been tested in its standard form and together with the application of an in-matrix JACOBI-preconditioning ("Scaling Solver"). The results show that for structural problems the direct solver PARDISO [11] can clearly compete with the openCL implementation of an iterative CG solver. Even the CPU-based CG solver is slower for the considered problem.

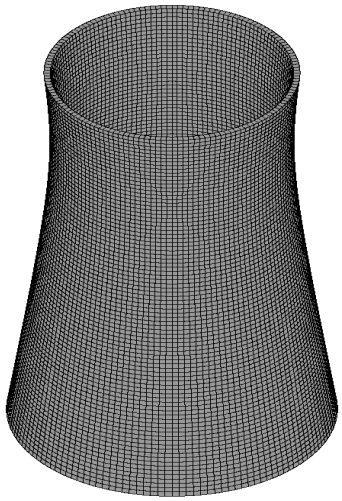

Figure 2: Benchmark structure: a cooling tower shell under gravity load

For the distributed memory version, the cooling tower structure has been computed using different numbers of processes on a computer cluster. Fig. 4 shows the speedup and the final parallel efficiency that has been achieved for a model of 66000 degrees of freedom. The linear solver used in this example is an iterative GMRES solver, preconditioned by means of a block-wise application of the direct LU-decomposition solver KLU.

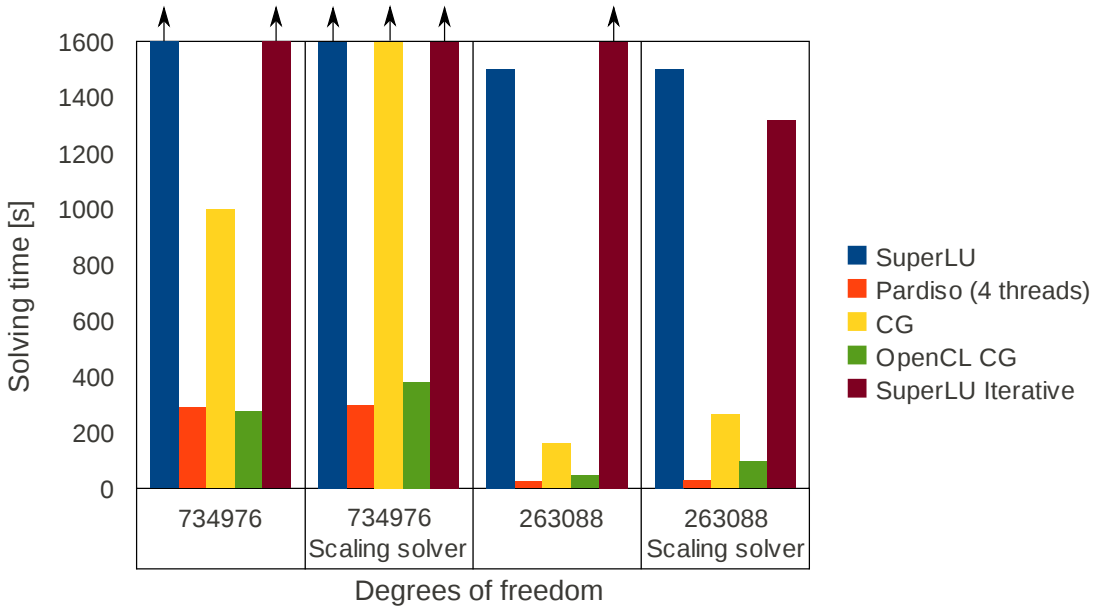

Figure 3: Turnover time of various linear solvers for the cooling tower benchmark in different discretisations (The results exceeding by far a reasonable solving time are indicated by arrows)

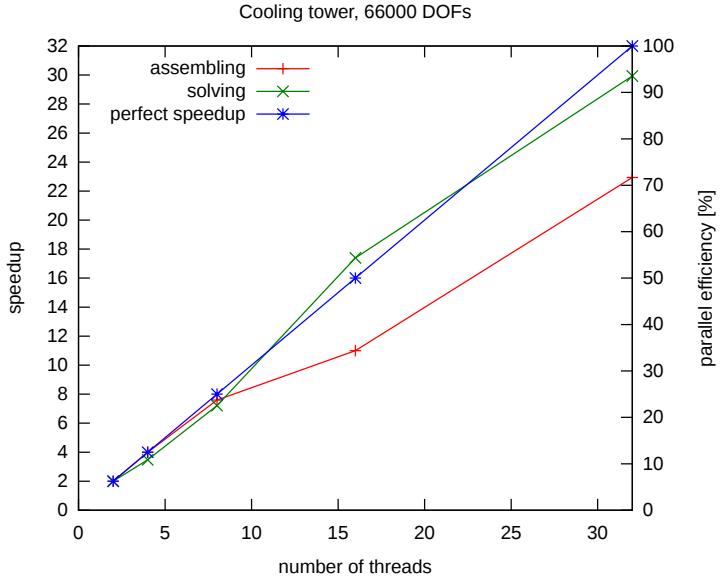

Figure 4: Speedup of the assembling and the solving process for a small cooling tower benchmark computed on a distributed memory cluster

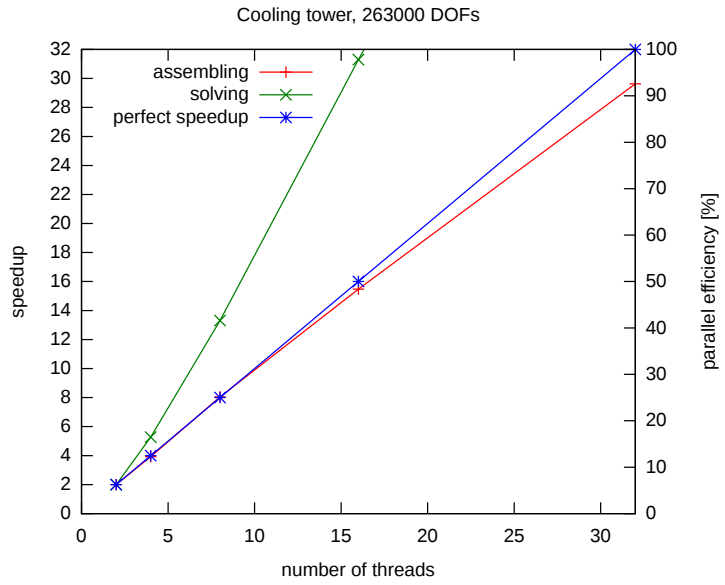

Figure 5: Speedup of the assembling and the solving process for a large cooling tower benchmark computed on a distributed memory cluster

It can be observed, that the solver exhibits a nearly perfect speedup on up to 32 threads whereas the assembling process performance drops for more than 8 threads and reaches a final efficiency of 70%. This can be explained by the communication overhead in the assembling of the global system matrix. For larger number of threads, the size of each subdomain becomes too small to take advantages from the parallelisation. This effect vanishes when a larger system is computed, as shown in Fig. 5. Here it can also be seen that the solver shows a superlinear speedup. This, as well as the optimal speedup in the smaller example can be attributed to the memory requirements of the KLU preconditioner and its effectiveness in the improvement of the matrix condition that depends on the block size. While the preconditioner has an algorithmic complexity of greater than $\mathcal{O}(n^2)$, the iterative solver for the global problem has a complexity of approx. $\mathcal{O}(n)$. Thus, the larger sub-domains in the case of fewer numbers of threads are much more expensive in terms of the preconditioning than the increasing size of the interface problem to be solved iteratively with growing numbers of threads.

4.2 Benchmark 2: Contact analysis using ccNUMA memory allocation

The effect of an optimised allocation strategy for ccNUMA architectures is demonstrated by means of a simple contact benchmark. Here, the assembling of the global system of linear equations requires a large share of the total turnover time since the computation of the element stiffness matrices follows a complex algorithm. Thus, the

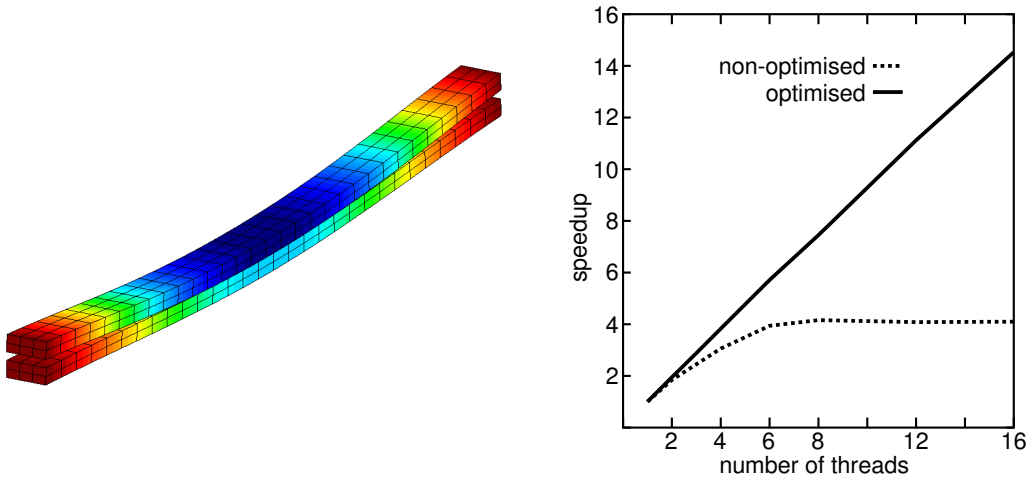

Figure 6: Effect of ccNUMA-aware optimisation of the memory allocation on the speedup of the assembling process: left: discretised benchmark system; right: speedup plot

effect of access to a single processor's local memory by all threads becomes most evident. The example has been computed on a 16-core AMD Opteron system that uses a hypercube ccNUMA architecture. Fig. 6 shows the effect this optimised allocation has on the parallel performance of the assembling process. It can be seen that without the optimised allocation strategy, the memory access bottleneck prevents the speedup from growing beyond 4 threads, whereas with a distributed memory allocation the speedup is nearly optimal up to 16 threads.

4.3 Benchmark 3: Parallel performance of tunnelling simulations

On a distributed memory computer cluster, two shield tunnelling simulations of different sizes have been computed at different numbers of threads. The smaller model features 8 steps of excavation at a total model length of 24 m whereas the larger model consists of 56 excavation steps at a length of 96 m. The domain decomposition of the larger model at 64 threads is shown in Fig. 7. As linear solver, analogous to the cooling tower example, the KLU-preconditioned iterative GMRES solver has been employed. The speedup plots in Figs. 8 and 9 show that the larger model exhibits a much better parallel performance than the smaller model, as was else observed in the cooling tower example. It can be further noted, that the additional complexity of the model in comparison to the simple structural model above reduces the parallel performance significantly. Yet, for the larger model a satisfactory parallel efficiency of approx. 60% can be achieved.

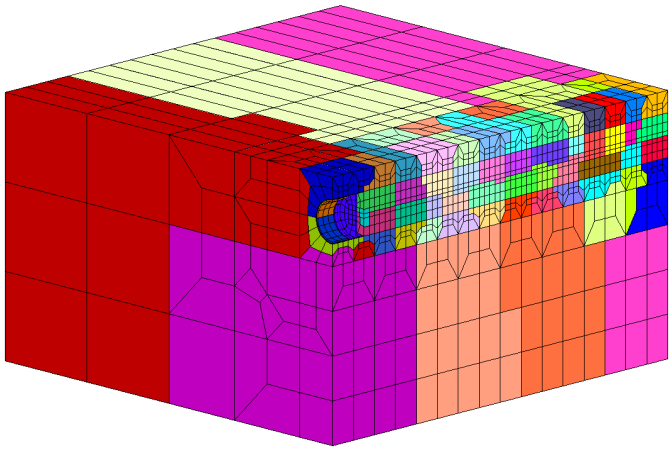

Figure 7: Domain decomposition of the larger shield tunnelling example running at 64 threads on a computer cluster.

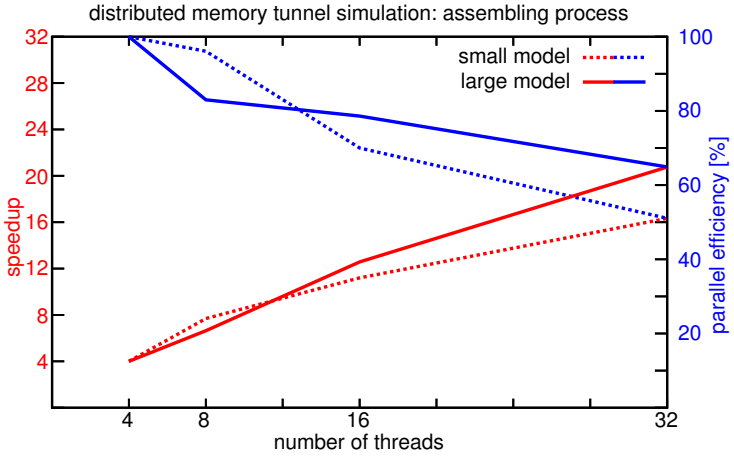

Figure 8: Speedup of the shield tunnelling simulations on a distributed memory cluster: assembling process

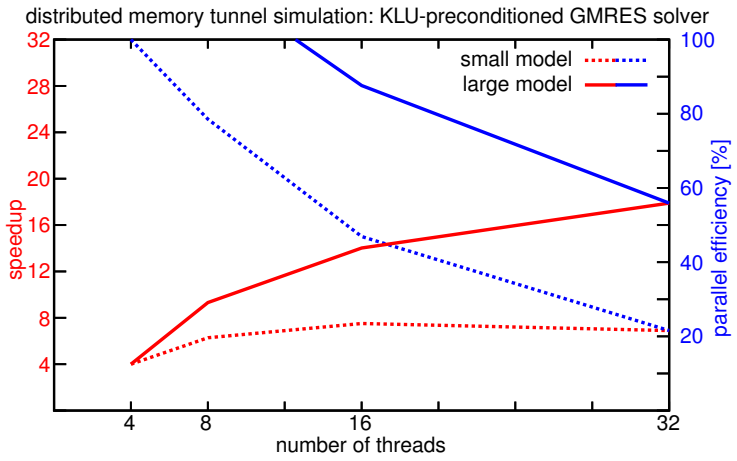

Figure 9: Speedup of the shield tunnelling simulations on a distributed memory cluster: solving process

5 Concluding remarks

Several parallelisation concepts have been presented and demonstrated for the solution of finite element models in structural mechanics. The capability of the presented software framework KRATOS to combine arbitrary algorithmic implementations, as for example contact algorithms, multi-phase and multi-physics formulations, or mesh coupling techniques, with each other regardless of the parallelisation method used, constitutes a considerable advantage for the testing of different linear solvers. In the context of the numerical simulation of the shield tunnelling process, the employed finite element model imposes high challenges for the efficient solution of the system equations.

It has been shown that problems in structural mechanics up to a relatively large size can be solved with similar efficiency with a good direct solver as well as with an iterative solver on shared memory systems. Considerable speedup could only be achieved on a distributed memory system using an iterative solver that is preconditioned by the block-wise application of a direct solver. However, also here the complexity added by the tunnelling simulation compared to a simple structural problem drops the parallel efficiency considerably.

The applicability of effective preconditioners for structural problems on GPGPUs is still limited. Here, an improvement in the memory allocation capabilities on graphics processors may allow for a performance that goes far beyond the performance of CPU architectures.

6 Acknowledgements

This work has been supported by the German Science Foundation (DFG) in the framework of the Collaborative Research Center SFB 837 “Interaction Modelling in Mechanised Tunnelling”. This support is gratefully acknowledged.

References

[1] F. Nagel, J. Stascheit and G. Meschke, “Process-oriented numerical simulation of shield tunneling in soft soils”, Geomechanics and Tunnelling, 3, 268-282, 2010.

[2] T. Laursen, “Computational Contact and Impact Mechanics”, Springer, 2002.

[3] F. Nagel, A. Bezuijen, J. Stascheit and G. Meschke, “Measurements and simulations of fluid and ground pressures around a TBM”, International Conference on Computational Methods in Tunnelling (EURO:TUN 2009), 61-70, 2009.

[4] J. Bluhm and R. de Boer, “The Volume Fraction Concept in the Porous Media Theory”, Zeitschrift fur angewandte Mathematik und Mechanik, 77, 563-577, 1997.

[5] Pooyan Dadvand, Riccardo Rossi and Eugenio Oñate, “A Framework for Developing Finite Element Codes for Multi-disciplinary Applications”, Proceedings of the 8th World Congress on Computational Mechanics, 2008.

[6] J. Stascheit, P. Dadvand and G. Meschke, “Parallelisation techniques for the numerical simulation of shield tunnelling processes”, in: International Conference on Computational Methods in Tunnelling (EURO:TUN 2009), 1032-1038, 2009.

[7] M. A. Heroux, “Epetra Performance Optimization Guide”, Technical Report, Sandia National Laboratories, 2005.

[8] E. P. Natarajan, “KLU - A high performance sparse linear solver for circuit simulation problems”m Master’s Thesis, University of Florida, 2005.

[9] `http://www.khronos.org/opencl/`

[10] `http://viennacl.sourceforge.net/`

[11] O. Schenk and K. Gärtner, “Solving unsymmetric sparse systems of linear equations with PARDISO”, Future Generation Computer Systems, 20, 475-487, 2004.