Application of a hybrid parallelization technique to accelerate the numerical simulation of nonlinear mechanical problems

Thomas Most and Stefan Eckardt, Institute of Structural Mechanics,
Bauhaus-University Weimar, Marienstrasse 15, 99423 Weimar, Germany
(thomas.most@uni-weimar.de, stefan.eckardt@uni-weimar.de)

Summary

This paper presents the combination of two different parallelization environments, OpenMP and MPI, in one numerical simulation tool. The computation of the system matrices and vectors is parallelized with OpenMP and the solution of the system of equations is done with the MPI-based solver MUMPS. The efficiency of both algorithms is shown on several linear and non-linear examples using the Finite Element Method and a meshless discretization technique.

1 Introduction

In the last two decades Galerkin Methods like the Finite Element Method have become popular tools to handle large mechanical problems numerically. Constitutive models can be used to simulate the nonlinear material behavior, e.g. cracking or plasticity. Due to the application of this material models the system of equations becomes nonlinear and an iterative solution scheme is required. The Newton-Raphson iteration procedure is the usually used tool to solve nonlinear quasi-static problems. In this method the external load will be applied incrementally, whereby for each increment the nodal displacement vector is iterated as long as internal and external forces become equilibrated. In each iteration step a linear system of equations has to be solved.

If we use standard single processor computers for nonlinear calculations with a large number of degrees of freedom (e.g. 10^6 to 10^7 d.o.f.), the computational time can easily increase to some hours or days. For this reason the parallelization of the calculation can be useful. In general two different standard parallelization environments are available for Unix-like computer systems. The first one is the MPI (Message Passing Interface) environment (Gropp 1996), which can be used on distributed and on shared memory systems. The other standard tool is OpenMP (OpenMP 2002) which can be utilized only on shared memory multiprocessor computers. The computation of the stiffness matrices and internal force vectors, which is done via numerical integration over integration cells, is parallelized directly with this tool due to the straightforward application of OpenMP to loops. The duplication of memory entries, in these cases the calculated matrices and vectors, which is necessary for MPI, is a disadvantage of the environment, thus it is not applied for this purpose.

The solution of the system of equations can be parallelized by using a domain decomposition method (Patzak and Bittnar 2001) or a multifrontal direct solving algorithm (Amestoy et al. 2000). In the World Wide Web many implementations of such solvers are available. Some examples are PSPASES, PIM and MUMPS. The authors decided to use one of these libraries instead of implementing a parallel algorithm themselves. One of the best documented and most efficient direct solver is the so-called “MUltifrontal Massively Parallel Solver” (MUMPS) (Amestoy et al. 2003) which is applied by the authors.

In this paper the parallelization of the numerical computation of the stiffness matrix and the internal force vector using OpenMP and the integration of the external MPI-based library MUMPS into the SLang Software package (Bucher et al. 2002) is presented. The efficiency and scalability of both algorithms will be shown first on different linear benchmark calculations. The applicability for nonlinear problems will be verified on a mesoscale fracture analysis of a

concrete specimen using a smeared crack model (Eckardt et al. 2004) and on a simulation of a plastification process using a meshless discretization. For this purpose a rotating crack model within the Finite Element Method and an adapted Natural Element Method (Unger 2004) are used, respectively.

2 Treatment of nonlinear mechanical problems

Most quasi-static mechanical problems can be solved sufficiently accurate within a Galerkin method by using a linearization of the system equations. The solution is obtained directly from the following equation

$$\mathbf{K}\mathbf{d} = \mathbf{F} \tag{1}$$

where $\mathbf{K}$ is the tangential symmetric stiffness matrix of the assumed system model, $\mathbf{d}$ contains the unknown nodal displacements and $\mathbf{F}$ is the external load vector. Nonlinear structural behavior caused by path dependent geometrical or material nonlinearities, which has to be considered for stability, fracture or plasticity analyses, can be investigated incrementally. An iteration procedure like the Newton-Raphson method (Bathe 1996) can be applied to obtain the nodal displacements at the end of the load increment $n+1$ under the condition that the external and internal forces $\mathbf{F}_{\text{ext}}$ and $\mathbf{F}_{\text{int}}$ are in equilibrium

$$\mathbf{F}_{\text{int},n+1} = \mathbf{F}_{\text{ext},n+1} \tag{2}$$

This procedure approximates the unknown internal forces as

$$\mathbf{F}_{\text{int},n+1} \approx \mathbf{F}_{\text{int},n} + \Delta\mathbf{F}_{\text{int},n+1} \;,\; \Delta\mathbf{F}_{\text{int},n+1} = \mathbf{K}_n \Delta\mathbf{d}_{n+1} \tag{3}$$

The iteration is done for $i = 1,2,3,...$

$$\begin{aligned} \mathbf{K}_{n+1,i-1}\Delta\mathbf{d}_{n+1,i} &= \mathbf{F}_{\text{ext},n+1} - \Delta\mathbf{F}_{\text{int},n+1,i-1} \\ \mathbf{d}_{n+1,i} &= \mathbf{d}_{n+1,i-1} + \Delta\mathbf{d}_{n+1,i} \end{aligned} \tag{4}$$

taking into account the initial conditions for each incremental step

$$\mathbf{d}_{n+1,0} = \mathbf{d}_n \,, \qquad \mathbf{K}_{n+1,0} = \mathbf{K}_n \,, \qquad \mathbf{F}_{\text{int},n+1,0} = \mathbf{F}_{\text{int},n} \tag{5}$$

until Eq. (2) is fulfilled with a given accuracy. Within this iteration the incremental linear system of equations in Eq. (4) can be solved similarly to Eq. (1). For larger systems the stiffness matrix $\mathbf{K}$ has sparse character due to the limited number of interacting nodes per integration cell. Standard procedures for the efficient solution are given in (Bathe 1996) by the $\mathbf{L}\mathbf{D}\mathbf{L}^T$ and the Cholesky factorization ($\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^T$) of the matrix $\mathbf{K}$, where $\mathbf{L}$ and $\widetilde{\mathbf{L}}$ are triangular matrices and $\mathbf{D}$ has diagonal form. The solution itself is obtained after the factorization via Gauss elimination.

3 Solution of large systems of equations with sequential solvers

In this section different sequential sparse matrix solvers will be presented very briefly. The traditional solver which is available in the applied program system SLang uses the already mentioned $\mathbf{L}\mathbf{D}\mathbf{L}^T$ factorization technique with following Gauss elimination. The necessary computing time for this method depends mainly on the structure of the stiffness matrix. A reduction of the matrix band width can lead to a much more efficient solution process then using the unstructured matrix. A simple ordering algorithm is given with the geometrical band width reduction (e.g. Bucher et al. 2002), where the nodes will be renumbered along a straight line given by two structural nodes. Another preconditioner available in SLang is a spectral reduction technique (Barnand et.al. 1993) which uses a nodal connectivity matrix to obtain the optimal node num-

bering. Within this method the computation of the second eigenvector of the stiffness matrix is necessary, which leads to some more computational costs. Due to the fact, that the preconditioner has to be applied only once at the beginning of a nonlinear calculation (if the discretization will not change during the simulation), these costs remain very small compared to the savings which can be obtained in solving the equation system in many load increments and iteration steps. Another sequential solver which is available in SLang is a sparse direct solver based on $\mathbf{LDL}^T$ factorization (Vondracek 2003). The sparsity of the final matrix **L** is ensured by initial approximate minimum degree ordering (Davis et al. 1994). This solver uses in contrast to the above mentioned a compressed column storage instead of the skyline format. The sequential solvers will be used for comparison with the MUMPS solver in the examples in section 6.

4 Implementation of the Multifrontal Massively Parallel Solver

The multifrontal method for the solution of sparse linear equations is a direct method based on the **LU** factorization of the matrix, which is analyzed first to determine an ordering that sparsity in the factors will preserved. MUMPS offers several built-in ordering algorithms, like an approximate minimum degree ordering and an approximate minimum fill-in ordering, and furthermore an interface to some external ordering packages such as PORD (Schulze 2001) and METIS (Karypis and Kumar 1998). Within the analysis an ordering and an assembly tree is produced. The assembly tree is then used to organize the subsequent numerical factorization and solution phases on the different processors. The solution is obtained by using the distributed factors of the frontal matrices on the different processors and their results are assembled at the host. For details of the theoretical background the reader is referred to (Amestoy2000). MUMPS uses the MPI message passing and requires the BLAS, BLACS and ScaLAPACK libraries. Within the downloadable parallel MUMPS source code package a sequential version is also available which only requires the standard BLAS library.

The integration of the sequential MUMPS version in SLang was done directly by linking the compiled library to the other required SLang libraries. Some more effort had to be spend to implement the parallel solver: Due to the fact, that SLang is an interactive structural language, controlled via a text based input file with flow control such as loops and conditional jumps and with an graphical user interface, it was not reasonable to transform the SLang main routine to a MPI routine, which is executed by all processors simultaneously with CPU identity based controlling. A more efficient possibility was chosen, by calling the parallel MUMPS solver as an external executable, which is a full MPI application. The required input and output data for and from the solver are stored in binary files on the local hard disk of the corresponding platform. By using this method a hybrid parallelization in combination with OpenMP is possible, due to the fact that the slave processors will only be used if the solver is called and released afterwards (analogous to OpenMP). The time required for writing and reading the data did not exceed 2 % of the total solver time for one CPU for all investigated examples. This can be tolerated compared to the performance increase due to the parallelization. In (Schrader 2004) a detailed description of the implementation of the sequential and the parallel version in SLang and the generation of the required libraries BLACS and ScaLAPACK can be found. Investigations in (Schrader 2004) and by the authors have shown, that the PORD and METIS ordering algorithms lead to the most efficient solving. For this reason only the results obtained with these algorithms will be presented for the numerical examples. The sequential MUMPS solver is used in the same section to point out how small the additional time requirement is, due to the data storage concept for the external solver.

5 Parallel computation of system matrices and vectors using OpenMP

By using OpenMP (OpenMP 2002) , which can only be applied on shared memory systems, time consuming loops can be parallelized with little programming effort. The computation of the stiffness matrices and internal force vectors of finite elements is realized in SLang independently for each element and the resulting arrays are stored in the element structure. Therewith the main loop summing up all element contributions can be easily parallelized. OpenMP simply needs the following parallel construct for ANSI-C programming directly in front of the loop:

```
#pragma omp parallel shared(counter, error, ...) private(...)
#pragma omp for schedule(dynamic,1) nowait
   for(counter = 0; counter < element_number; counter ++)
   {
      /* compute single finite element matrix */
      ...
   }
```

The loop counter, an error identifier and other controlling variables are defined as shared variables for all processors. Private variables are these, which have to be a local copy for each processor. The dynamic schedule type enables OpenMP to choose the optimal processor usage. During the meshless calculations done by the authors the stiffness matrix and internal force vector are computed via numerical integration over triangular integration cells (Most and Bucher 2003). In general the domain is discretized with a few meshless zones, each containing a large number of nodes and integration cells. The resulting objects are stored in compact form for each meshless zone. This means that the parallelization of the loop over the integration cells can not be done independently as for finite elements. The problem can be solved by defining a critical region, where the local stiffness matrix of one integration cell is transferred onto the stiffness matrix of the meshless zone, so that only one processor executes this part in a given time:

```
#pragma omp parallel shared(counter, error, ...) private(...)
#pragma omp for schedule(dynamic,1) nowait
   for(counter = 0; counter < triangle_number; counter ++)
   {
      /* compute local matrix of integration cell*/
      ...
      #pragma omp critical
      {
         /* transfer integration cell matrix to matrix of meshless zone */
         ...
      }
   }
```

6 Examples

To analyze the efficiency of the presented hybrid parallelization technique several numerical examples have been investigated on two different hardware configurations. The first platform was a Pentium 4 based 32-bit Linux cluster with three nodes, each having two CPUs with 2.4 GHz and 4 GB shared memory, connected via a Gigabit Ethernet switch. The second system was the Altix 3000 super computer manufactured from SGI with 40 Itanium II processors (64 bit) with 900 MHz, connected with high speed Numa Link and 40 GB shared memory. The calculations have been carried out on the first system by using up to 4 processors for the MPI solver and the 2 processors of one node for the OpenMP parallelized system matrix computation and on the second system with up to 32 processors for both parallel algorithms.

As comparison to the efficiency of the parallel solver the above mentioned sequential solvers with $\mathbf{LDL}^T$ factorization and geometrical (Geom.) and spectral (Spect.) band width reduction and approximate minimum degree ordering (Vondr.) are used.

6.1 Linear finite element analysis of three-dimensional problems

6.1.1 Cantilever subjected to a single load

This example investigates the scalability of the presented parallel solver and the parallel finite element matrix and vector computation by studying a simple cantilever with a length of $1m$ and a quadratic cross section of $0.1m \times 0.1m$. Material properties are assumed as $E = 2 \cdot 10^6 N/m^2$ for the Young's modulus and $\nu = 0.3$ for the Poisson's ratio. The cantilever is loaded by a point load of $1N$ as shown in Figure 1. Due to the slenderness of the system the application of the preconditioning algorithms lead to a stiffness matrix with small band width as illustrated in Figure 2. Two different finite element models have been investigated: first a discretization with 640 27-node hexahedral elements and 6561 nodes, resulting in 19612 degrees of freedom, and another model using 5120 27-node elements with 46529 nodes and 139516 degrees of freedom.

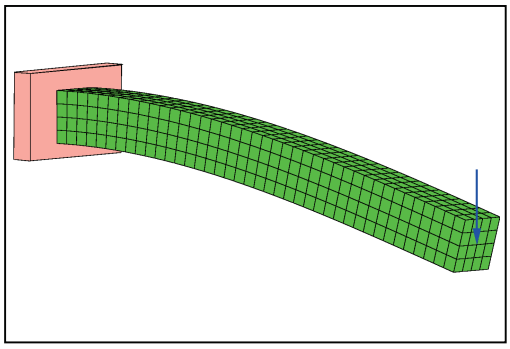

Figure 1: Cantilever finite element model with load and boundary conditions for 640 elements

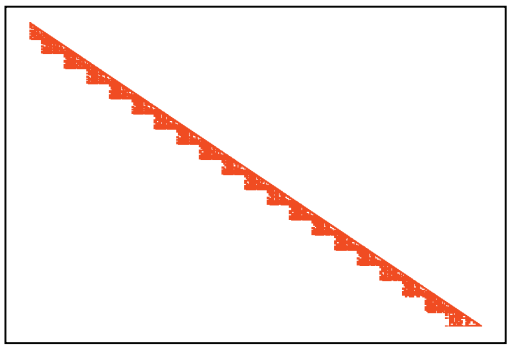

Figure 2: Optimized stiffness matrix by using spectral reduction

The scalability of the parallel algorithms was investigated first on the Altix 3000 computer for both models. In Figure 3 the computing time required for a single factorization and solution step is shown as a function of the applied number of processors. For this analysis the external ordering algorithms PORD and METIS have been used and are compared to the theoretical ideal line obtained from the computational time by using one processor. Both algorithms lead to a very good speed-up for the larger model. For the smaller model a good efficiency up to 4 CPUs can be seen. In Figure 4 the scalability of the stiffness matrix computation is shown, which corresponds to an almost perfect speed-up until 16 processors.

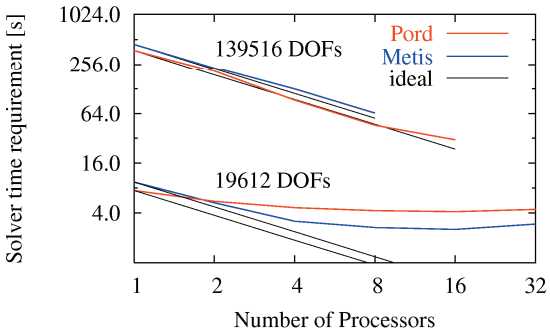

Figure 3: Computational time for one factorization and solution step on the Altix 3000

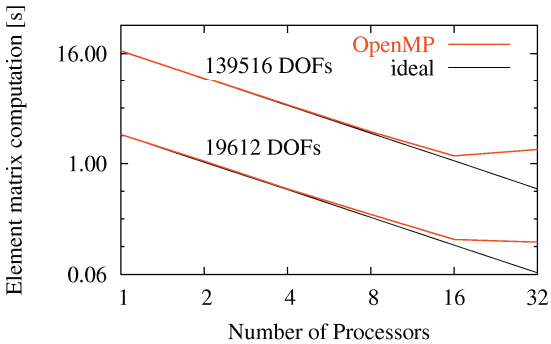

Figure 4: Computational time for the stiffness matrix calculation on the Altix 3000

In Table 1 a detailed list of the computing time for one solver call is given for all presented sequential and parallel solvers. It is distinguished between the time needed for the factorization and the solution and the preconditioning and the solver handling, such as the storage of the data and the call of the MPI environment for the external MPI solver. The empty fields indicate that the calculation for this configuration was stopped because of limited memory. It can be seen that

the sequential MUMPS solver is already faster than all the other presented solvers. The scalability of the parallel solver for the larger model on the Altix 3000 is almost ideal and on the Linux Cluster it is about $1.5^{\log_2(n)}$ which is still a very good speed-up, in consideration of the required communication between the Cluster CPUs. The solver handling for the MPI version takes on the Altix 3000 nearly 1.5 % for the smaller and 0.2 % for the larger model and on the Cluster nearly 8 % and 1 %, respectively, which shows that the presented external concept does not lead to remarkable higher computational costs. Table 2 shows the efficiency of the OpenMP parallelized stiffness matrix and internal force vector computations. As already mentioned in Figure 3 the speed-up on the Altix 3000 platform is almost ideal, which corresponds to the results obtained on the Linux Cluster, by using both processors of one cluster node. This example shows that the independent usage of MPI and OpenMP parallelization techniques does increase the efficiency of the presented numerical simulation on both platforms as expected.

Table 1: Comparison of computational time in seconds for the different matrix solvers
(a: time for factorization and solution, b: time for preconditioning and solver handling)

DOFs	Geom.	Spect.	Vondr.	MUMPS seq.		MUMPS parallel					
				Metis	Pord	Metis			Pord		
						1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
Linux Cluster											
19612 a)	29.21	24.11	4.87	5.33	4.49	5.36	3.17	3.24	4.65	3.50	5.37
19612 b)	0.07	2.44	0.21	1.12	0.62	1.43	1.46	1.62	0.98	1.01	1.08
139516 a)	-	-	399.44	268.00	207.94	276.66	183.93	118.63	203.47	-	122.29
139516 b)	-	-	16.88	13.09	6.86	15.39	15.11	15.06	9.01	-	8.96
Altix 3000											
19612 a)	21.66	14.86	15.72	9.46	7.49	9.47	5.35	3.16	7.50	5.50	4.66
19612 b)	0.12	8.72	0.35	1.68	1.06	1.83	1.88	1.89	1.18	1.27	1.35
139516 a)	4665.0	3200.5	3028.9	449.08	379.84	449.05	232.57	127.13	378.84	210.91	93.08
139516 b)	1.93	300.40	52.13	19.27	11.54	19.95	20.42	20.45	12.36	12.80	13.05

Table 2: Comparison of computational time in seconds for the system matrix and vector calculation

DOFs	Linux Cluster				Altix 3000					
	Stiffness matrix		Internal forces		Stiffness matrix			Internal forces		
	1 CPU	2 CPU	1 CPU	2 CPU	1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
19612	2.06	1.04	0.48	0.24	2.06	1.05	0.54	0.81	0.44	0.23
139516	17.13	8.36	3.76	1.94	16.52	8.54	4.30	6.68	3.67	1.89

6.1.2 Investigation of a compact 3D block

This example is used to present the efficiency of the parallel solver on a compact 3D structure with observable higher band width of the stiffness matrix than in the previous example. For this purpose a cubic block with $1m$ edge length is investigated by using two finite element models with 512 elements, 4913 nodes and 14739 degrees of freedom and with 4096 elements, 35937 nodes and 106928 degrees of freedom, respectively. 27-node hexahedral finite elements have been used again. The load and boundary conditions are shown for the smaller model in Figure 5. The displayed linear distributed load is assumed to be 16 N/m. The Young's modulus and

Poisson's ratio have been taken as $E = 1.0 \cdot 10^3 \, N/m^2$ and $\nu = 0.3$, respectively. Figure 6 shows the optimized stiffness matrix, which is much more compact than in the previous example.

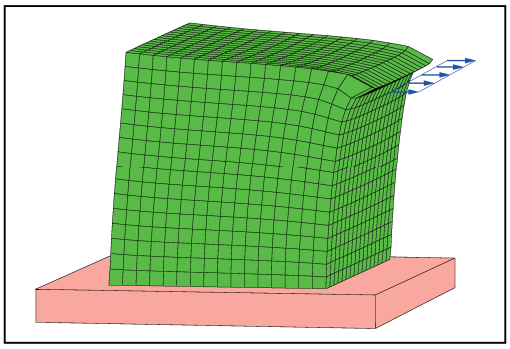

Figure 5: Compact block model with load and boundary conditions for 512 elements

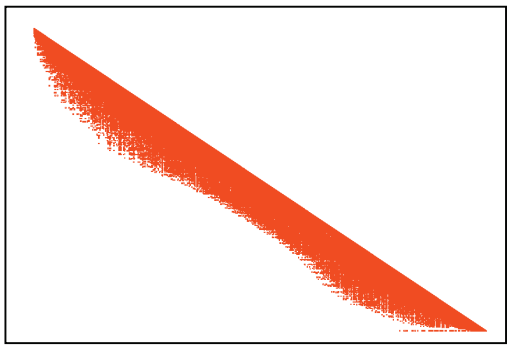

Figure 6: Optimized stiffness matrix by using spectral reduction

In Figures 7 and 8 as well as in Tables 3 and 4 the obtained numerical results are shown analogous to the previous example. It can be concluded, that the scalability for the solver and the matrix computation is very good again. The application of the presented parallel solver leads to a significant efficiency increase compared to the other solvers. Both numerical examples in section 6.1 verify the applicability of the hybrid parallelization technique for linear problems.

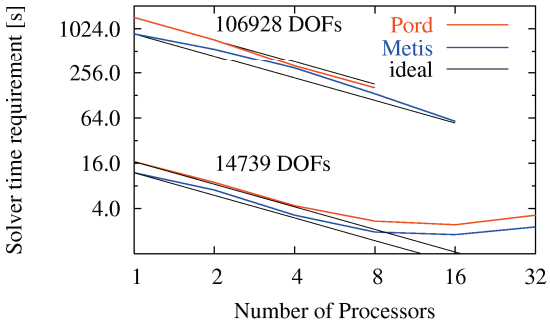

Figure 7: Computational time for one factorization and solution step on the Altix 3000

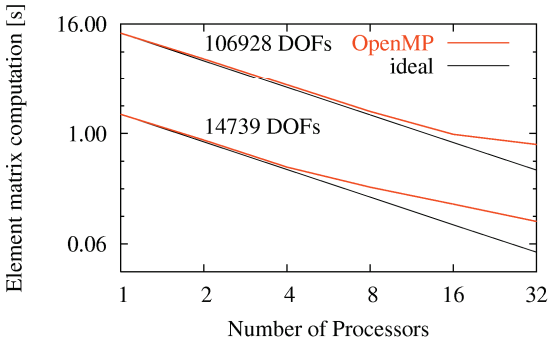

Figure 8: Computational time for the stiffness matrix calculation on the Altix 3000

Table 3: Comparison of computational time in seconds for the different matrix solvers (a: time for factorization and solution, b: time for preconditioning and solver handling)

DOFs	Geom.	Spect.	Vondr.	MUMPS seq.		MUMPS parallel					
				Metis	Pord	Metis			Pord		
						1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
Linux Cluster											
14739 a)	210.40	160.36	9.73	7.19	10.34	8.15	4.79	3.28	11.12	7.56	7.04
14739 b)	0.19	1.69	1.12	1.52	1.19	1.87	1.80	1.85	1.45	1.52	1.55
106928 a)	-	-	-	679.28	1278.2	689.26	582.64	221.35	1300.8	-	-
106928 b)	-	-	-	16.53	15.00	18.27	17.27	17.27	16.87	-	-
Altix 3000											
14739 a)	293.85	219.30	37.28	11.95	16.82	11.95	7.04	3.25	16.80	8.91	4.35
14739 b)	0.37	3.75	2.56	2.09	1.97	2.19	2.25	2.29	2.10	2.15	2.16
106928 a)	81305	107746	41647	873.17	1446.3	878.07	544.91	298.29	1448.1	727.63	318.41
106928 b)	11.7	76.8	47.51	24.06	26.33	26.35	24.46	24.83	26.81	27.23	27.35

Table 4: Comparison of computational time in seconds for the system matrix and vector calculation

DOFs	Linux Cluster				Altix 3000					
	Stiffness matrix		Internal forces		Stiffness matrix			Internal forces		
	1 CPU	2 CPU	1 CPU	2 CPU	1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
14739	1.65	0.83	0.37	0.18	1.62	0.85	0.43	0.64	0.36	0.20
106928	13.78	6.96	3.01	1.52	12.77	6.69	3.40	5.10	2.89	1.49

6.2 Mesoscale fracture analysis of concrete with finite elements and rotating crack model

In this example the applicability of the parallelization techniques to simulations of concrete structures at the mesoscale is shown. At this length scale the material structure is explicitly represented by the numerical model, e.g. concrete is separated into three main components: the homogeneous mortar matrix, the aggregates and interfacial zone between them. A finite element discretization, which explicitly represents the boundary between aggregates and matrix, is used for numerical simulations. Linear elastic material behavior is assumed for the aggregates and the standard rotating crack model (Jirásek and Zimmermann 1998) is applied to the mortar matrix to simulate tensile failure. Rigid bond between the constituents is assumed. A detailed description of this mesoscale model for concrete, e.g. the geometry generation, the applied material models and the numerical simulations, is given in (Eckardt et al. 2004). The resolution of the material structure depends on the number of finite elements. Numerical models for the simulation of concrete at the mesoscale with aggregates from a diameter of 2 *mm* normally have a large number of degrees of freedom. Due to the applied nonlinear material law an iterative solution scheme is required. The numerical effort increases with the number of degrees of freedom and the number of cracked elements. The use of parallel solvers and parallel computation of system matrices is required to reduce the time effort of the simulation.

In Figure 9 the specimen geometry, material parameters and the distribution of the aggregates is illustrated. A constant displacement u_y is applied in increments along the upper edge. For the

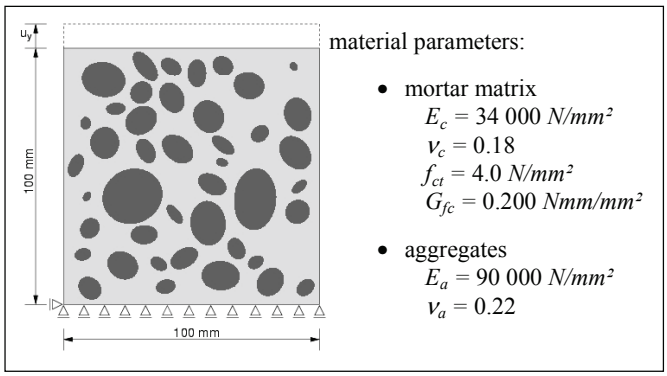

Figure 9: uniaxial tension test - concrete specimen, particle distribution, dimensions, material properties

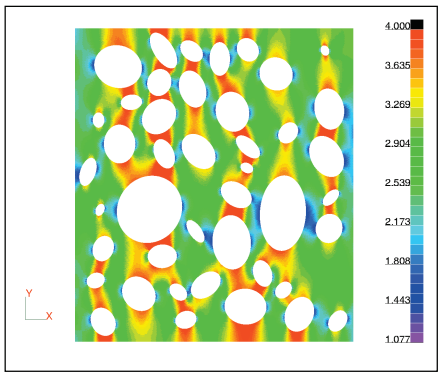

Figure 10: principal stress σ_{11} [N/mm²] in the mortar matrix

two dimensional simulation plane stress is assumed. The thickness of the specimen is 1 *mm*. Two triangular finite element meshes, with different element sizes, are used to show the scalability of the parallelization techniques. The coarse model has 92506 degrees of freedom, 91906 elements and 46354 nodes, and the fine mesh consists of 367624 elements and 184613 nodes and has 368824 degrees of freedom. In Figure 10 the first principal stress within the mortar matrix under uniaxial tension is plotted immediately after first cracks initiate. It shows the inhomogeneous stress distribution in the material structure under uniform loading conditions.

In Table 5 the time effort for the computation of the stiffness matrix and the internal force vector is given. The usage of two processors significantly reduces the computational time compared to one CPU. Due to the simplicity of the material law and the small number of cracked elements, the utilization of more than two processors does not improve the efficiency of the computation any further.

Table 5: Comparison of computational time in seconds for the system matrix and vector calculation

DOFs	Linux Cluster				Altix 3000					
	Stiffness matrix		Internal forces		Stiffness matrix			Internal forces		
	1 CPU	2 CPU	1 CPU	2 CPU	1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
92 506	3.63	2.04	2.88	1.52	5.45	3.14	3.01	4.39	2.66	2.45
368 824	14.42	8.95	11.45	6.42	21.91	12.59	11.95	17.40	10.41	8.37

The time for the factorization and the solution step is illustrated in Table 6. The computational effort and the memory supply of the Vondráček solver is significantly larger compared to the MUMPS package. Two preconditioner METIS and PORD are analyzed for MUMPS. The difference in the computational time for solving is negligible. For preconditioning METIS normally requires less time than PORD. For the fine finite element mesh a good scalability of the solver is observed up to 8 CPUs. On the shared memory system Altix 3000 the usage of 4 processors decreases the computational time nearly by a factor of 4 compared to 1 CPU. The time for preconditioning becomes eminent. An application of a parallel preconditioner would be reasonable. The example shows that the utilization of the parallel MUMPS package can significantly reduce the computing time for mesoscale simulations.

Table 6: Comparison of computational time in seconds for the different matrix solvers
(a: time for factorization and solution, b: time for preconditioning and solver handling)

DOFs	Vondr.	MUMPS seq.		MUMPS parallel							
		Metis	Pord	Metis				Pord			
				1 CPU	2 CPU	4 CPU	8CPU	1 CPU	2 CPU	4 CPU	8 CPU
Linux Cluster											
92 506 a)	71.14	3.09	3.22	3.12	1.83	1.69	-	3.26	2.14	1.89	-
92 506 b)	-	1.03	1.50	1.20	1.30	1.45	-	1.69	1.78	1.92	-
368 824 a)	-	23.18	24.25	23.02	12.84	8.73	-	23.99	13.84	13.01	-
368 824 b)	-	5.44	6.76	5.95	6.27	6.85	-	7.26	7.14	7.68	-
Altix 3000											
92 506 a)	306.31	5.23	5.52	5.32	2.92	1.80	1.49	5.68	3.40	1.97	1.69
92 506 b)	-	1.33	1.96	1.43	1.63	1.65	1.74	2.04	2.25	2.30	2.49
368 824 a)	9790.38	39.32	41.90	41.62	22.99	10.63	6.12	42.24	19.01	10.44	6.59
368 824 b)	-	6.84	9.13	7.15	7.82	7.98	8.28	9.42	10.30	10.46	10.75

6.3 Simulation of a plastification process using a 2D meshless dicretization of the domain

Within this example the presented parallelization techniques will be verified for a nonlinear simulation of a plastification process by using a meshless discretization. The standard v. Mises yield criterion with linear hardening and associated flow rule is used in the material model. As

meshless method an adapted version of the “Natural Element Method” (Unger 2004) is applied. The system with loading and boundary conditions is shown in Figure 11. The edge length and the thickness of the quadratic panel are $1m$, respectively. The material is assumed to be homogeneous with $E = 3 \cdot 10^{10} N/m^2$ for the Young’s modulus, $\nu = 0.3$ for the Poisson’s ratio, $\sigma_Y = 2 \cdot 10^6 N/m^2$ for the yield stress and $H = 1 \cdot 10^{10} N/m^2$ for the plastic hardening modulus. Due to the loading and boundary conditions a uniaxial stress state is implied within the structure. Two models with 427 nodes / 8319 degrees of freedom and 16643 nodes / 33023 degrees of freedom have been analyzed for a complete linear and for a complete plastified material state. The numerical effort for the stress integration on the integration point level is much higher for the plastic state than for the linear state. This can be seen in Figure 12, where the required time for the stiffness matrix computation of the larger model on the Altix 3000 platform is shown for both cases as a function of the CPU number.

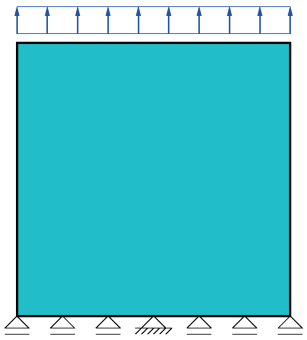

Figure 11: Quadratical panel with load and boundary conditions

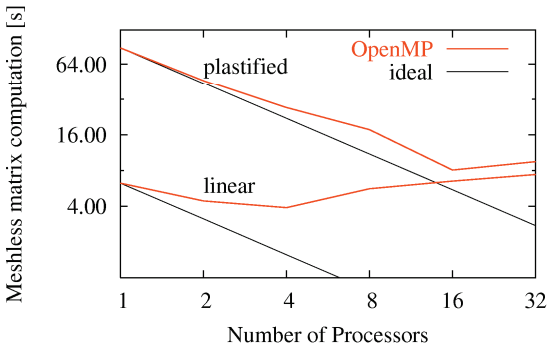

Figure 12: Computational time requirement for stiffness matrix calculation

The figure indicates a good scalability for the plastified state up to 16 processors. For the linear model a speed-up slightly below $1.5^{\log_2(n)}$ can be seen until 4 CPUs. The influence of the critical region in the OpenMP parallelized loop, mentioned in section 5 is much higher for the linear state. By using more than 4 CPUs, this critical region results in a decrease of the efficiency due to the interference of the processors. If the material is completely plastified, this influence remains small enough to obtain an almost ideal scalability because of the time consuming stress integration. In Table 7 a detailed list of the determined computing times for the calculation of the stiffness matrix and the internal force vector is given for the Altix 3000 and the Linux Cluster platform.

Table 7: Comparison of computational time in seconds for the system matrix and vector calculation (a: material complete linear, b: material complete plastified)

DOFs	Linux Cluster				Altix 3000					
	Stiffness matrix		Internal forces		Stiffness matrix			Internal forces		
	1 CPU	2 CPU	1 CPU	2 CPU	1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
8319 a)	0.66	0.43	0.33	0.22	1.52	1.02	1.03	1.13	0.64	0.45
8319 b)	14.09	7.56	13.43	7.10	16.48	8.10	4.14	15.82	7.69	3.74
33023 a)	2.48	1.64	1.27	0.91	6.25	4.43	3.89	4.57	2.88	1.76
33023 b)	56.23	32.00	57.38	30.66	87.85	46.46	27.10	86.28	44.67	24.29

In Table 8 the time for a single factorization and solution step for the different solvers is shown for linear and plastified material. All solvers need the same time for both states, which shows that they perform independently from the material state and the stiffness matrix computation.

This example points out, that the presented hybrid parallelization technique can be successfully applied for nonlinear problems.

Table 8: Comparison of computational time in seconds needed for factorization and solution (a: material complete linear, b: material complete plastified)

DOFs	Geom.	Spect.	Vondr.	MUMPS seq.		MUMPS parallel					
				Metis	Pord	Metis			Pord		
						1 CPU	2 CPU	4 CPU	1 CPU	2 CPU	4 CPU
Linux Cluster											
8319 a)	4.08	3.33	1.75	0.67	0.64	0.68	0.40	0.37	0.66	0.45	0.52
8319 b)	4.08	3.34	1.80	0.67	0.65	0.68	0.36	0.38	0.66	0.44	0.53
33023 a)	65.57	50.97	15.57	5.12	6.08	5.14	2.97	2.35	5.99	3.66	3.20
33023 b)	66.14	50.99	15.92	5.13	6.12	5.15	3.02	2.34	6.01	3.64	3.14
Altix 3000											
8319 a)	2.17	1.77	5.06	1.15	1.12	1.15	0.64	0.44	1.12	0.72	0.51
8319 b)	2.17	1.76	5.29	1.14	1.11	1.15	0.64	0.44	1.12	0.73	0.52
33023 a)	60.78	46.59	55.88	8.97	10.16	8.97	4.96	2.74	10.15	5.55	2.93
33023 b)	60.76	46.52	57.24	8.95	10.14	8.95	4.95	2.74	10.14	5.52	2.97

7 Conclusions

In this paper a hybrid parallelization technique using OpenMP and MPI is presented. OpenMP is used to parallelize the computation of the stiffness matrices and internal force vectors of the integration cells. If a finite element discretization is used, this cells coincide with the elements and the main loop can directly be parallelized. For a meshless discretization the loop over the triangle integration zones is taken. There a critical region has to be defined for the transfer of the local matrices of the cells to the matrix of the meshless zone. The solution of the systems of equations is obtained by calling the MPI-based MUMPS solver as an external application. The interaction between SLang and the solver is realized via an input and output binary file. The MPI-used processors will be released after each solver call and can be used for OpenMP in the next iteration step.

The presented algorithms have been verified for linear simulations for two examples using three-dimensional finite element models. The parallel solver and the computation of the system matrices and vectors show a very good scalability for large system sizes. The usage of the parallel solver on a distributed memory system is less efficient than on a shared memory computer, because of the additional network communication between the cluster nodes. For nonlinear applications the authors performed two further examples. The first one investigated the meso-scale fracture behavior of concrete with finite elements and a rotating crack model and the second example was the simulation of a plastification process, using a meshless dicretization based on the Natural Element Method. Both examples could verify the independency of the OpenMP and MPI parallelized SLang commands and the applicability of the hybrid method for nonlinear simulations.

8 Acknowledgement

This research has been supported by the German Research Council (DFG) through Collaborative Research Center 524, which is gratefully acknowledged by the authors.

9 References

Amestoy, P.R., Duff, I.J. and L'Excellent, J.-Y. (2000), Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng., 184: 501-520

Amestoy, P.R., Duff, I.J., L'Excellent, J.-Y. and Kloster, J. (2003), MUltifrontal Massively Parallel Solver (MUMPS). ENSEEIHT-IRIT, Toulouse, France, Users' guide 4.3 edition

Barnard S.T., Pothen A., and Simon H.D. (1993), A spectral algorithm for envelope reduction of sparse matrices. In Proc. Supercomputing '93, 93-502

Bucher C. et al. (2002), SLang – The Structural Language. Institute of Structural Mechanics, Bauhaus-University Weimar, Germany, 5.0 edition

Bathe, K.J. (1996), Finite Element Procedures. Prentice Hall, Englewood Cliffs

Davis, T.A., Amestoy, P. and Duff, I.S. (1994), An Approximate Minimum Degree Ordering Algorithm. Comp. Inform. Sciences Dept., University of Florida, Technical Report TR-94-039

Eckardt, S., Haefner, S. and Koenke, C. (2004), Simulation of the fracture behaviour of concrete using continuum damage models at the mesoscale. In P. Neittaanmäki et.al. editors, Proc. 4th European Congress on Comp. Mechanics in Appl. Sciences and Eng., Jyväskylä, Finland, July 24-28

Gropp, W., Lusk, E., Doss, N. and Skjellum, A. (1996), A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, 22(6): 789-828

Jirásek, M. and Zimmermann, T. (1998), Analysis of rotating crack model. Journal of engineering mechanics, 124(8):842-851

Karypis, G. and Kumar, V. (1998), METIS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0, University of Minnesota

Most, T. and Bucher, C. (2003), Moving Least Squares-elements for stochastic crack propagation simulations coupled with stochastic finite elements. In A. Der Kiureghian et.al. (Eds.), Proc. 9th Int. Conf. Appl. of Stat. Prob. Civil Eng., San Francisco, July 6-9, Balkema

OpenMP Architecture Review Board (2002), OpenMP C and C++ Application Program Interface. Version 2.0, www.openmp.org

Patzák, B. and Bittnar, Z. (2001), Design of object oriented finite element code. Advances in Engineering Software, 32(10-11): 759-767

Schrader, K. (2004), Implementation des „MUltifrontal Massively Parallel Solvers" über ein C-Interface in SLang zur Lösung großer linearer Gleichungssysteme mit schwachbesetzten Matrizen. Student research project, Institute of Struct. Mechanics, Bauhaus-University Weimar

Schulze, J. (2001), Towards atighter coupling of bottom-up and top-down sparse matrix ordering methods. BIT, 41(4): 800-841

Unger, J.F., Most, T., Bucher, C. and Koenke, C. (2004), Adaptation of the Natural Element Method for crack growth simulations. In P. Neittaanmäki et.al. editors, Proc. 4th European Congress on Comp. Mechanics in Appl. Sciences and Eng., Jyväskylä, Finland, July 24-28

Vondráček, R. (2003), Application of Sparse Direct Solver on Large Systems of Linear Equations. In J. Vrba et al. (Eds.), Book of Abstracts, 2nd PhD Workshop, Brno University of Technology, 20.-22. November