Optimized Kernels for Large Scale Earthquake Simulations with SeisSol, an Unstructured ADER-DG Code

Alexander Heinecke, Alexander Breuer, Sebastian Rettenberger, Michael Bader
Technische Universität München
{heinecke-breuera}@in.tum.de
{rettenbs-bader}@in.tum.de

Alice Gabriel, Christian Pelties
Stefan Wenk
Ludwig-Maximillians-Universität München
{gabriel}@geophysik.uni-muenchen.de
{pelties-wenk}@geophysik.uni-muenchen.de

Introduction

The software package SeisSol is one of the leading codes for earthquake scenarios, in particular for simulating dynamic rupture processes and for problems that require discretization of very complex geometries. Spatial adaptivity in 3D is realized by flexible unstructured tetrahedral meshes. SeisSol uses the discontinuous Galerkin (DG) method for spatial and Arbitrary high order DERivatives (ADER) for time discretization. Due to it's high degree of locality the ADER-DG method shows near-optimal speedups on supercomputing architectures, [3, 2]. We present optimizations for the inner, cell-local kernel routines of SeisSol, where SeisSol spends most of the total computing time. The kernels are implemented as a sequence of matrix-matrix-multiplications of relatively small size and varying sparsity patterns (ranging from "very" sparse to dense).

1. SPARSITY PATTERNS OF THE ADER-DG METHOD

The elastic wave equations are the basic set of model equations in SeisSol, details about extension to complex rheologies and dynamic rupture coupling of this set can be found in [4]. In velocity-stress formulation the homogenous elastic wave equations are given by a quasi-linear hyperbolic system of partial differential equations:

$$q_t + A(x,y,z)q_x + B(x,y,z)q_y + C(x,y,z)q_z = 0, \qquad (1)$$

where $q(t,x,y,z)$ is the nine-dimensional space-time-dependent vector of unknowns and A, B and C are the space-dependent Jacobians. By applying the ADER-DG scheme for space and time discretization, we obtain from Eq. (1) an explicit updating scheme from time step t^n to t^{n+1} [1]:

$$\begin{aligned} Q_k^{n+1} = Q_k^n - \frac{|S_k|}{|J_k|} M^{-1} \Bigg(&\sum_{i=1}^{4} F^{-,i} I(t^n, t^{n+1}, Q_k^n) N_{k,i} A_k^+ N_{k,i}^{-1} \\ &+ \sum_{i=1}^{4} F^{+,i,j,h} I(t^n, t^{n+1}, Q_{k(i)}^n) N_{k,i} A_{k(i)}^- N_{k,i}^{-1} \Bigg) \\ &+ M^{-1} K^{\xi} I(t^n, t^{n+1}, Q_k^n) A_k^* \\ &+ M^{-1} K^{\eta} I(t^n, t^{n+1}, Q_k^n) B_k^* \\ &+ M^{-1} K^{\zeta} I(t^n, t^{n+1}, Q_k^n) C_k^*. \end{aligned} \qquad (2)$$

For an ADER-DG method of convergence order $\mathcal{O}$, we require $B = \frac{1}{6}\mathcal{O}(\mathcal{O}+1)(\mathcal{O}+2)$ distinct basis functions in the discontinuous Galerkin approximation. Typical orders of production runs in SeisSol are five or six, resulting in $B = 35$ or $B = 56$ basis functions. The individual matrices in Eq. (2) are given by:

- Q_k^{n+1}, Q_k^n – Unknowns for each basis function at time t^{n+1} and t^n in tetrahedron k: $B \times 9$
- $N_{k,i} A_k^+ N_{k,i}^{-1}$, $N_{k,i} A_{k(i)}^- N_{k,i}^{-1}$ – Flux solver for element k and face i: 9×9
- A_k^*, B_k^*, C_k^* – Linear combinations of the Jacobians in tetrahedron k with respect to the corresponding transformation to the reference tetrahedron: 9×9
- K^{ξ}, K^{η}, K^{ζ} – Stiffness matrices over the reference tetrahedron: $B \times B$
- $F^{-,i}$, $F^{+,i,j,h}$ – Flux matrices over the reference tetrahedron: $B \times B$
- M^{-1} – Inverse of the diagonal mass matrix over the reference tetrahedron; stored implicitly as part of the flux and stiffness matrices: $B \times B$

The operator $I(t^n, t^{n+1}, Q_k^n)$ in Eq. (2) represents the ADER time integration:

$$I(t^n, t^{n+1}, Q_k^n) = \sum_{j=0}^{\mathcal{O}-1} \frac{(t^{n+1} - t^n)^{j+1}}{(j+1)!} \frac{\partial^j}{\partial t^j} Q_k(t^n), \qquad (3)$$

where the time derivatives are defined as a recursive scheme with initial values $\frac{\partial^0}{\partial t^0} Q_k = Q_k^n$:

$$\frac{\partial^{j+1}}{\partial t^{j+1}} Q_k = -M^{-1} \left((K^\xi)^T \left(\frac{\partial^j}{\partial t^j} Q_k \right) A_k^* \right. \\ \left. + (K^\eta)^T \left(\frac{\partial^j}{\partial t^j} Q_k \right) B_k^* + (K^\zeta)^T \left(\frac{\partial^j}{\partial t^j} Q_k \right) C_k^* \right). \tag{4}$$

$|J_k|$ and $|S_k|$ are scalars stored implicitly as part of the flux solver, which account for the transformation to the reference element and the corresponding transformation of the element's boundary.

ADER time integration: Fig. 1 shows all involved sparsity patterns of the ADER time integration for a fifth order ($B = 35$) scheme.

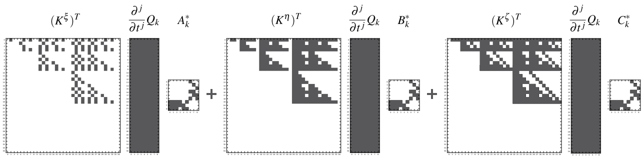

Figure 1: Sparsity patterns appearing in the ADER time integration for a fifth order method.

Volume integration: Except for the non-transposed stiffness matrices, the sparsity pattern of the volume integration is identical to the ADER time integration, illustrated in Fig. 1.

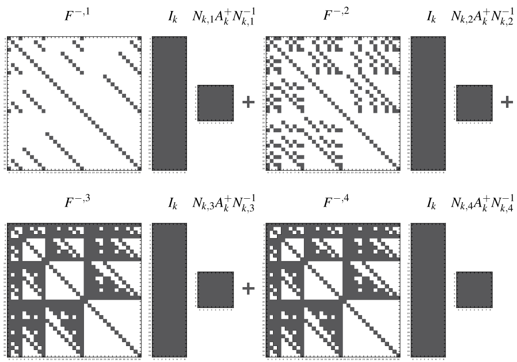

Figure 2: Sparsity patterns appearing in the element contribution to the flux computation.

Boundary integration: Fig. 2 shows all element local matrix patterns ($B = 35$). The contribution of the neighboring elements to the numerical flux, given by the second sum of Eq. (2), is mathematically similar to the elements' contribution. The major difference are the 48 different flux matrices $F^{+,i,j,h}$ appearing throughout the sum. The sparsity patterns of the flux matrices are highly varying and can even be near-dense as the examples in Fig. 3 show.

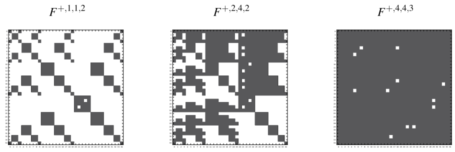

Figure 3: Further examples of sparsity patterns appearing in the neighboring element contribution.

2. MATRIX ROUTINES AND RESULTS

Since the sparsity patterns described in Sect. 1 are statically known, we can generate code exactly matching our needs through an enriched offline code generation phase before calling the routines (note that calling optimized library functions provided by vendors and academia is not suitable due to the small matrix sizes). During this step, vector instructions are employed as much as possible by generating kernels for *sparse* × *dense* = *dense*, *dense* × *sparse* = *dense* and *dense* × *dense* = *dense* which are required according to Figs. 1-3. When running the LOH.1 benchmark [1] with convergence order $\mathcal{O} = 6$, see Fig. 4, in a strong scaling setting on up to 32,000 Sandy Bridge cores of SuperMUC[1], we achieved a maximum performance of 153 TFLOPS, which corresponds to 21.6% of the theoretical peak performance. By comparing our optimized version to the classic version of SeisSol we obtained a total application speed-up of roughly 4.5 (depending on number of cores due to overhead of communication) which nicely fits to the vector length of four provided by Sandy Bridge's vector-extension AVX.

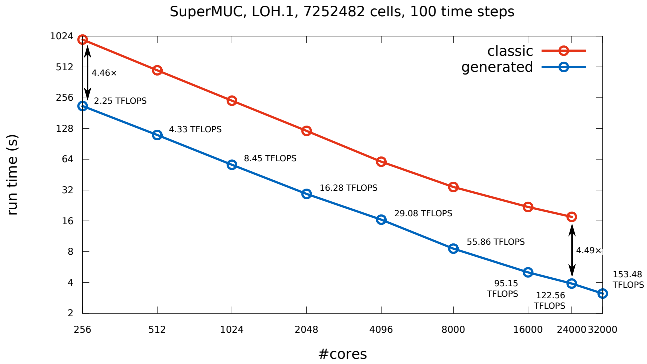

Figure 4: Runtime enhancement due to code generation on SuperMUC.

3. REFERENCES

[1] M. Dumbser et al. An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes – II. The three-dimensional isotropic case. *Geophysical Journal International*, 2006.

[2] M. Käser et al. SeisSol–A Software for Seismic Wave Propagation Simulations. *High Performance Computing in Science and Engineering, Garching/Munich 2009*, 2010.

[3] M. Käser et al. Wavefield modeling in exploration seismology using the discontinuous Galerkin finite-element method on HPC infrastructure. *The leading edge*, 2010.

[4] Christian Pelties et al. Three-dimensional dynamic rupture simulation with a high-order discontinuous Galerkin method on unstructured tetrahedral meshes. *Journal of Geophysical Research: Solid Earth*, 2012.

[1] `http://www.lrz.de/services/compute/supermuc/`