Using code-transformation and CUDA graphs to optimize 3D elastic-acoustic simulations on Nvidia DGX A100

Ravil Dorozhinskii[1], Lukas Krenz[1], Sebastian Wolf[1], Gregor Hillers[2], Alice-Agnes Gabriel[3], Piero Lanucara[4], Michael Bader[1]
[1]Technical University of Munich, [2]University of Helsinki, [3]Ludwig Maximilian University of Munich, [4]CINECA Supercomputing Center
October 22, 2021

Abstract

We make use of the DGX A100 platform for solving a 3D, fully-coupled Earth-Air wave propagation model using the open-source software SeisSol. Each A100 GPU is managed by a single MPI process with 2 dedicated threads: for control and progressing non-blocking MPI communication. In this work, we apply our new GPU code generation approach for batched GEMM computations. We also use CUDA graphs and capturing for reducing kernel launch overheads while working with our Local Time Stepping scheme. Additionally, we show that Singularity containerization leads to a negligible performance loss of ≈ 1.5% compared to a bare-metal installation of SeisSol for our scenario.

Model

Elastic & Acoustic Wave Propagation

We model the Earth with the **elastic wave equation** and the air above with the **acoustic wave equation**. Between both parts, we ensure that the physical **interface conditions** hold (continuity of normal velocity and traction). With this we are able to capture the entire physics in one **fully-coupled model** [3]. Both equations can be written in standard linear hyperbolic form:

$$\frac{\partial}{\partial t}\boldsymbol{q} + \boldsymbol{A}(\boldsymbol{x})\frac{\partial}{\partial x}\boldsymbol{q} + \boldsymbol{B}(\boldsymbol{x})\frac{\partial}{\partial x}\boldsymbol{q} + \boldsymbol{C}(\boldsymbol{x})\frac{\partial}{\partial x}\boldsymbol{q} = \boldsymbol{S}(\boldsymbol{q}) \tag{1}$$

where $\boldsymbol{q}$ is a vector of unknowns and $\boldsymbol{A}(\boldsymbol{x}), \boldsymbol{B}(\boldsymbol{x}), \boldsymbol{C}(\boldsymbol{x})$ are flux matrices that depend on position $\boldsymbol{x}$ but not on time t.
The source term $\boldsymbol{S}(\boldsymbol{q})$ models the earthquake using a **kinematic point source** model.

ADER-DG

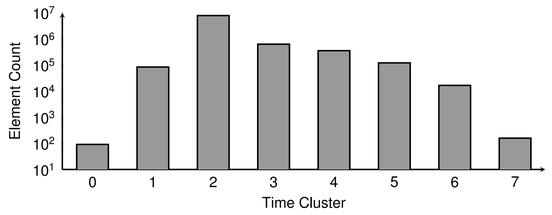
An application of Cauchy-Kovalewski procedure and Taylor expansion allows us to obtain an arbitrary order time-integration predictor, which can be evaluated in the reference element coordinate system as follows:

$$\begin{aligned}\frac{\partial^{j+1}}{\partial t^{j+1}}Q_k^n &= \tilde{K}^{\xi}\left(\frac{\partial^j}{\partial t^j}Q_k^n\right)A_k^* + \tilde{K}^{\eta}\left(\frac{\partial^j}{\partial t^j}Q_k^n\right)B_k^* + \tilde{K}^{\zeta}\left(\frac{\partial^j}{\partial t^j}Q_k^n\right)C_k^* \\ \mathcal{D}_k &= \sum_{j=0}^{\mathcal{O}-1}\frac{(t^{n+1}-t^n)^{j+1}}{(j+1)!}\frac{\partial^j}{\partial t^j}Q_k^n\end{aligned} \tag{2}$$

Then, DG machinery is applied to obtain an element-local update scheme which includes evaluations of *local*, *neighbor* flux integrals as well as the *volume* integral. The scheme mainly consist of **small and dense matrix multiplication chains**.

Local Time Stepping (LTS)

LTS aims at improving time-to-solution by reducing the total amount of floating point operations. We cluster elements such that they get updated **as seldom as possible** ensuring that the **stability limit** is valid. In each cluster, all elements have the same timestep size: In the smallest cluster, all elements have a timestep size of Δt_{min}, in the n^{th} cluster all elements have a stepsize of $R^n \Delta t_{\text{min}}$, where R is the so-called LTS-rate. For our scenario with **9.8 million elements**, the distribution for $R = 3$ looks like:

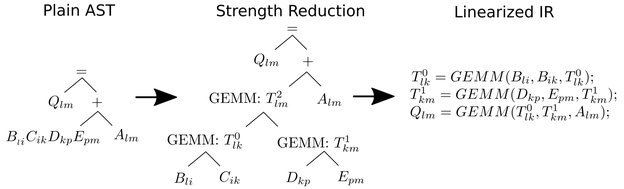


Here, LTS resulted in a speed-up factor of ≈ 9.75 compared to global time stepping (i.e., using a single cluster). We parallelize over all elements that are in a cluster. The LTS scheme needs to ensure that dependencies resulting from flux contributions from neighboring elements are resolved. Hence, we **update clusters sequentially**.

Code Generation

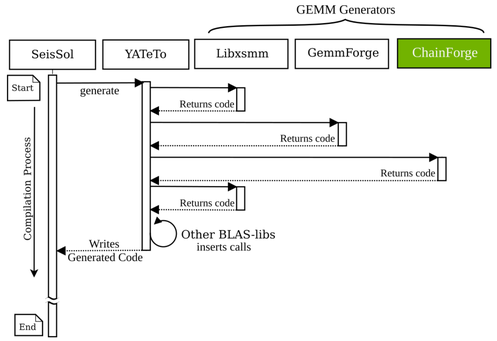
YATeTo [4] is a source code generator for tensor expressions which implements the expressions as sequences of loops over GEMMs. The figure below illustrates the generation process of Eq. 3 written using Einstein notation as an example.

$$Q_{lm} = A_{lm} + B_{li} \cdot C_{ik} \cdot D_{kp} \cdot E_{pm} \tag{3}$$



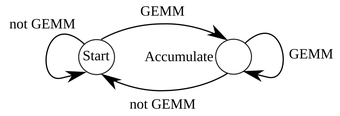
SeisSol expresses wave propagation numerical schemes with YATeTo's DSL. YATeTo:

- gets invoked during SeisSol compilation with specific:
  wave propagation *model*, *order* of convergence, host and device *architectures*
- solves the matrix chain ordering problem
- generates code using:
  BLAS operations for CPUs, i.e., element-wise evaluations
  batched BLAS operations for GPUs
- can make use of external, architecture specific GEMM generators



ChainForge

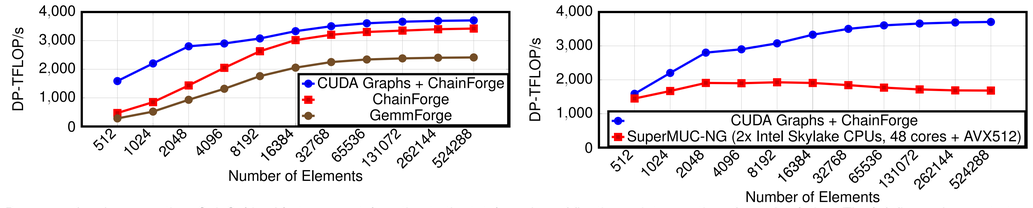
Idea: unlike GemmForge [1], the ChainForge generator is designed to the improve arithmetic intensity of **batched** DG-like computations by **fusing** several consecutive GPU GEMM kernels into a single one. ChainForge has a language front-end but the main interaction is intended to be through its high-level intermediate representation (IR), i.e., lists of GEMM descriptions. The generation process starts by extracting intermediate results from its IR, which reside in shared memory together with the right-hand sides of each individual GEMM operation. ChainForge solves a **graph coloring problem** (similar to register allocation) to reduce shared memory usage per block. Additionally, block-level synchronizations are resolved by a **data flow analysis** performed at a low-level IR.



Integration: a new analysis phase was added to YATeTo which uses a **2-state deterministic Finite Automata (FA)** applied to YATeTo's linearized IR to recognize **subsequent GEMM operations**. Whenever the FA comes back to the initial state, a **Fused-GEMM** operation is generated and a sequence of GEMM operations is substituted with a Fused-GEMM operation which stores the sequence. During code generation, the operation invokes ChainForge and passes the sequence and GPU architecture as parameters.

CUDA Graphs

Problem: Clusters with the highest update frequency tend to be extremely small and thus introduce significant overheads of launching the corresponding GPU kernels.
Solution: Capturing CUDA Graphs for all time clusters and macro-kernels i.e., *volume*, *local* and *neighbour* integrals, as well as *ader* scheme.

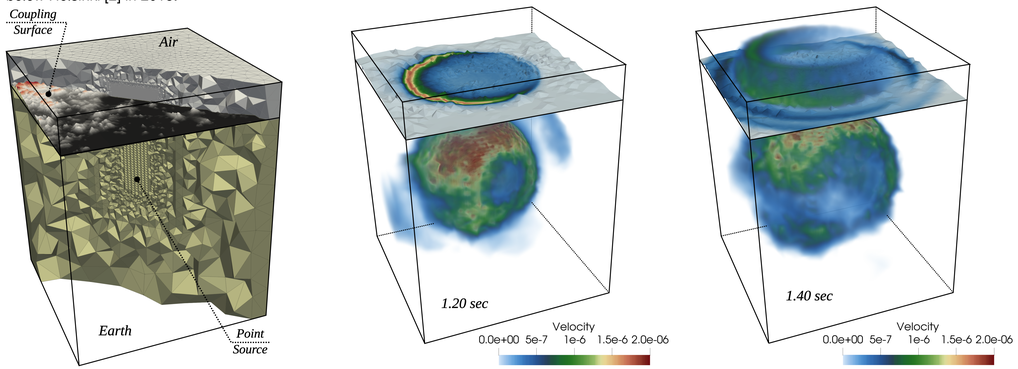


Because of code generation, SeisSol is able to accurately estimate the total number of floating point operations for a simulation. The left figure demonstrates strong scaling of a single time cluster using a performance proxy compiled with Double Precision (DP). **Fused-GEMM** operators resulted in an ≈ 770 DP-TFLOP/s increase of average GPU performance. Application of **CUDA Graphs** helped to significantly increase performance within the [512, 4096] LTS range and thus resulted in an average ≈ 630 DP-TFLOP/s of additional performance.

The right figure demonstrates the performance difference between **a single A100 GPU** and a single compute-node of LRZ SuperMUC-NG. On average, an A100 GPU **outperformed** 2 Intel Skylake Xeon Platinum 8174 CPUs by factors of ≈ 1.75 and ≈ 1.55 with respect to DP-TFLOP/s and time-to-solution, respectively.

Results

The overall computational domain is $12 \times 12 \times 15$ km^3. The solid Earth domain extends 13 km downwards, the atmospheric layer is 2 km thick. We use an unstructured tetrahedral mesh to enable realistic topography. The mesh consists of 9.8 million elements and a local refinement region along the zone of interest. The point source at 6.5 km depth has the same mechanism and size as the largest M_L1.8 earthquake that was induced by a geothermal stimulation below Helsinki [2] in 2018.



Achievements

1. fully-coupled simulation in Air and Earth
2. setup with a realistic topography [2]
3. obtained ≈ **26.2** and ≈ **25.8** DP-TFLOP/s on a single DGX A100 node (i.e., 8x GPUs) with a bare-metal installation and Singularity container, respectively.

Acknowledgements

This project received funding from the European Union's Horizon 2020 research and innovation programme under the ChEESE project, grant agreement No. 823844. We acknowledge the CINECA supercomputing centre for giving us access to their DGX A100 system.

References

[1] Ravil Dorozhinskii and Michael Bader. "SeisSol on Distributed Multi-GPU Systems: CUDA Code Generation for the Modal Discontinuous Galerkin Method". In: *The International Conference on High Performance Computing in Asia-Pacific Region*. 2021, pp. 69–82.

[2] Gregor Hillers et al. "The 2018 Geothermal Reservoir Stimulation in Espoo/Helsinki, Southern Finland: Seismic Network Anatomy and Data Features". en. In: *Seismological Research Letters* (2020). DOI: 10.1785/0220190253. (Visited on 02/10/2020).

[3] Lukas Krenz et al. "3D Acoustic-Elastic Coupling with Gravity: The Dynamics of the 2018 Palu, Sulawesi Earthquake and Tsunami". In: *arXiv preprint arXiv:2107.06640* (2021).

[4] Carsten Uphoff and Michael Bader. "Yet Another Tensor Toolbox for discontinuous Galerkin methods and other applications". en. In: *ACM Transactions on Mathematical Software* 46.4 (2020). DOI: 10.1145/3406835.

GPU TECHNOLOGY CONFERENCE