FINITE-DIFFERENCE SEISMIC MODELLING ON CPUS AND GPUS USING MATRIX-VECTOR PRODUCTS

F. Wittkamp, T. Steinweg, and T. Bohlen

email: *tilman.steinweg@kit.edu*
keywords: *seismic modelling, matrix-vector formalism, CPU/GPU*

ABSTRACT

Modern seismic imaging methods such as reverse-time migration (RTM) or full-waveform inversion (FWI) require large high-performance computing (HPC) systems to provide enough computational power to solve a large number of forward problems based on the wave equation. These wavefield simulations are conventionally performed by explicit time-domain finite-difference (FD) methods on regular numerical grids, where the parallelization is often based on a fixed and rather inflexible decomposition of the computational domain. However, such parallelization cannot exploit the computing capacities of modern and especially future exascale HPC architectures, which are expected to become more and more hierarchical and non-uniform. For this purpose, we developed a matrix-vector formulation of the explicit time-domain FD method solving the 3D elastic wave equation. To implement the matrix-vector formalism, we chose the open-source framework LAMA, which allows the development of hardware-independent code. We found that the implementation of such a matrix-vector based 3D elastic forward solver is straightforward. In a strong and weak scaling benchmark, we subsequently explored the scaling behavior of our implementation. The overall scaling performance shows the large potential of our method, which can be improved even further by tuning on the application and framework level.

INTRODUCTION

Modern seismic imaging methods such as reverse-time migration (RTM) or full-waveform inversion (FWI) require large high-performance computing (HPC) systems to provide enough computational power to solve a large number of forward problems based on the wave equation. These wavefield simulations are conventionally performed by explicit time-domain finite-difference (FD) methods on regular numerical grids, where the parallelization is often based on a fixed and rather inflexible decomposition of the computational domain. Such parallelization cannot exploit the computing capacities of modern and especially future exascale HPC architectures, which are expected to become more and more hierarchical and non-uniform. In recent years open-source HPC libraries have become available that allow to run the same software code on different hardware architectures. However, these libraries typically require the problem to be formulated using matrices and vectors. We therefore developed a matrix-vector formulation of the explicit time-domain FD method solving the 3D elastic wave equation. This formulation allows us to hide and outsource most of the HPC issues, such as efficient domain decomposition, inter-node communication and load balancing, to an HPC library. We chose the open-source framework LAMA, which allows the development of hardware-independent code.

METHODOLOGY

We demonstrate the matrix-vector formalism using the 1D acoustic wave equation in stress-velocity formulation:

$$\frac{\partial v(x,t)}{\partial t} = \gamma(x)\,\frac{\partial p(x,t)}{\partial x}, \qquad \frac{\partial p(x,t)}{\partial t} = \lambda(x)\,\frac{\partial v(x,t)}{\partial x}, \tag{1}$$

where p is the pressure wavefield, v is the particle velocity, λ is the first Lamé parameter and γ is the inverse density ρ^{-1}. We solve the wave equations on a staggered grid by an explicit FD time-stepping scheme, which is second-order accurate in time and higher-order accurate in space (Virieux, 1986). We first review the application of the conventional second-order accurate FD operator to the first-order spatial derivative of a differentiable function f which is evaluated on discrete grid points:

$$\frac{\partial f(x)}{\partial x}\Big|_{\frac{x_1+x_2}{2}} = \frac{f_{x_2} - f_{x_1}}{\Delta x} \tag{2}$$

The discrete points x_1 and x_2 are spaced by the distance Δx. For FD modeling this operator has to be evaluated individually for each grid point. Therefore, we can express it as a matrix-vector product:

$$\begin{pmatrix} \frac{\partial f(x)}{\partial x}\Big|_{\frac{x_1+x_2}{2}} \\ \frac{\partial f(x)}{\partial x}\Big|_{\frac{x_2+x_3}{2}} \\ \vdots \end{pmatrix} = \frac{1}{\Delta x} \begin{pmatrix} -1 & 1 & 0 & \dots \\ 0 & -1 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \cdot \begin{pmatrix} f_{x_1} \\ f_{x_2} \\ f_{x_3} \\ \vdots \end{pmatrix}, \tag{3}$$

where we obtain the derivatives by a multiplication of a matrix with a vector containing the functional values of f at the discrete grid points. This can be formulated in matrix-vector notation as follows:

$$\frac{\partial \vec{f}}{\partial x} = \frac{1}{\Delta x}\,\underline{D}_f \cdot \vec{f} \tag{4}$$

The derivative matrix $\underline{D}_f$ contains the FD coefficients. It exhibits strong sparsity because it only contains n non-zero entries per row (resp. grid point), where n is the order of the spatial operator (e.g., $n = 2, 4$ or 6). The sparsity allows an efficient storage and optimized access pattern for the multiplication. Using this approach we obtain the explicit second-order time-stepping (leapfrog) scheme for the 1D acoustic wave equation in matrix-vector notation:

$$\vec{v}^{\,n+1} = \vec{v}^{\,n} + \frac{\Delta t}{\Delta x}\,\vec{\gamma} \circ \left(\underline{D}_f \cdot \vec{p}^{\,n+\frac{1}{2}}\right), \qquad \vec{p}^{\,n+\frac{1}{2}} = \vec{p}^{\,n-\frac{1}{2}} + \frac{\Delta t}{\Delta x}\,\vec{\lambda} \circ \left(\underline{D}_b \cdot \vec{v}^{\,n}\right), \tag{5}$$

where Δt is the temporal sampling interval, $\vec{\gamma}$ and $\vec{\lambda}$ are the vectorized material parameters and $\underline{D}_b = -\underline{D}_f^T$, which accounts for the spatial grid staggering. The symbol $\circ$ denotes the elementwise product. Higher-order accuracy in space can be achieved by adjusting the FD coefficients contained in $\underline{D}$. Moreover, the derivative matrix allows to explicitly consider boundary conditions, like the free-surface condition, by modification of the relevant rows at initialization.

3D ELASTIC IMPLEMENTATION

To transfer the methodology to the 3D elastic wave equation it is necessary to store the 3D parameter fields, such as the wavefields, in 1D vectors. We obtain this transfer by a row-major-wise mapping of 3D coordinates to 1D indices. By applying the shown approach to the 3D elastic wave equation in stress-

velocity formulation, which is straightforward, we obtain:

$$\vec{v}_x^{n+1} = \vec{v}_x^n + \frac{\Delta t}{\Delta h}\,\vec{\gamma} \circ \left(\underline{D}_{x,f} \cdot \vec{\sigma}_{xx}^{n+\frac{1}{2}} + \underline{D}_{y,b} \cdot \vec{\sigma}_{xy}^{n+\frac{1}{2}} + \underline{D}_{z,b} \cdot \vec{\sigma}_{xz}^{n+\frac{1}{2}}\right), \tag{6a}$$

$$\vec{v}_y^{n+1} = \vec{v}_y^n + \frac{\Delta t}{\Delta h}\,\vec{\gamma} \circ \left(\underline{D}_{x,b} \cdot \vec{\sigma}_{yx}^{n+\frac{1}{2}} + \underline{D}_{y,f} \cdot \vec{\sigma}_{yy}^{n+\frac{1}{2}} + \underline{D}_{z,b} \cdot \vec{\sigma}_{yz}^{n+\frac{1}{2}}\right), \tag{6b}$$

$$\vec{v}_z^{n+1} = \vec{v}_z^n + \frac{\Delta t}{\Delta h}\,\vec{\gamma} \circ \left(\underline{D}_{x,b} \cdot \vec{\sigma}_{zx}^{n+\frac{1}{2}} + \underline{D}_{y,b} \cdot \vec{\sigma}_{zy}^{n+\frac{1}{2}} + \underline{D}_{z,f} \cdot \vec{\sigma}_{zz}^{n+\frac{1}{2}}\right), \tag{6c}$$

$$\vec{\sigma}_{xx}^{n+\frac{1}{2}} = \vec{\sigma}_{xx}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\lambda} \circ \left(\underline{D}_{x,b} \cdot \vec{v}_x^n + \underline{D}_{y,b} \cdot \vec{v}_y^n + \underline{D}_{z,b} \cdot \vec{v}_z^n\right) + 2 \cdot \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \underline{D}_{x,b} \cdot \vec{v}_x^n, \tag{6d}$$

$$\vec{\sigma}_{yy}^{n+\frac{1}{2}} = \vec{\sigma}_{yy}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\lambda} \circ \left(\underline{D}_{x,b} \cdot \vec{v}_x^n + \underline{D}_{y,b} \cdot \vec{v}_y^n + \underline{D}_{z,b} \cdot \vec{v}_z^n\right) + 2 \cdot \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \underline{D}_{y,b} \cdot \vec{v}_y^n, \tag{6e}$$

$$\vec{\sigma}_{zz}^{n+\frac{1}{2}} = \vec{\sigma}_{zz}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\lambda} \circ \left(\underline{D}_{x,b} \cdot \vec{v}_x^n + \underline{D}_{y,b} \cdot \vec{v}_y^n + \underline{D}_{z,b} \cdot \vec{v}_z^n\right) + 2 \cdot \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \underline{D}_{z,b} \cdot \vec{v}_z^n, \tag{6f}$$

$$\vec{\sigma}_{xy}^{n+\frac{1}{2}} = \vec{\sigma}_{xy}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \left(\underline{D}_{y,f} \cdot \vec{v}_x^n + \underline{D}_{x,f} \cdot \vec{v}_y^n\right), \tag{6g}$$

$$\vec{\sigma}_{xz}^{n+\frac{1}{2}} = \vec{\sigma}_{xz}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \left(\underline{D}_{z,f} \cdot \vec{v}_x^n + \underline{D}_{x,f} \cdot \vec{v}_z^n\right), \tag{6h}$$

$$\vec{\sigma}_{yz}^{n+\frac{1}{2}} = \vec{\sigma}_{yz}^{n-\frac{1}{2}} + \frac{\Delta t}{\Delta h}\,\vec{\mu} \circ \left(\underline{D}_{z,f} \cdot \vec{v}_y^n + \underline{D}_{y,f} \cdot \vec{v}_z^n\right), \tag{6i}$$

where $\vec{\sigma}$ is the stress tensor, $\vec{v}_i$ are the particle velocities, Δh is the equidistant grid spacing and $\vec{\mu}$ is the second Lamé parameter. The symbol $\circ$ denotes again the elementwise product. The matrix-vector formalism for the 3D elastic wave equation requires six separate derivative matrices, since two matrices are needed per direction in space, due to the spatial grid staggering. Similar to the 1D case, the relation between the forward (f) and backward (b) derivative matrix reads $\underline{D}_b = \underline{D}_f^T$.

We implemented the shown matrix-vector formalism of the explicit staggered-grid time-domain 3D elastic FD method with the open-source framework LAMA, which is designed for writing hardware-independent software code (Brandes et al., 2017). It facilitates the development of fast and scalable software for distributed HPC systems currently including multicore CPUs, Nvidia® GPUs and Intel® Xeon® Phi™. Besides, it allows a seamless integration of future architectures while offering an independent formulation of algorithms, which makes rewriting code for new hardware obsolete. Routines like the presented FD update (equation 6) operate on matrix and vector structures with a high level of abstraction, solving the general problem without giving considerations to the actual implementation details.

BENCHMARKS

To explore the performance of the presented implementation, we perform strong and weak scaling benchmarks based on the 3D elastic wave propagation. The benchmarks are run on the JURECA HPC system (Krause and Thörnig, 2016), where each CPU node consists of two Intel Xeon E5-2680 v3 Haswell CPUs (2x12 cores at 2.5 GHz) and each GPU node of two Nvidia Tesla K80. In the following discussion we address both a single CPU node (with 24 cores) and a single GPU device as a processing unit (PU). A GPU node thus holds two GPU PUs, while a CPU node is a single PU. To address multiple processes we use an MPI-based parallelization, where we choose an MVAPICH2 implementation in combination with the GNU compiler suite. For the GPU runs we additionally use asynchronous execution for overlapping halo communication with computation to also benefit from the host CPU. To minimize the required communication overhead, we manually optimize the partitioning of the 3D wavefields by assigning 3D sub-cubes to individual processes. Each benchmark is measured at least three times to ensure stable results.

For the benchmark we use a homogeneous 3D elastic model with a P-wave velocity of 3500 m/s , an S-wave velocity of 2500 m/s and a density of 2000 kg/m^3. We propagate the waves over 1000 time steps and set the temporal sampling to $2 \cdot 10^{-3}$ s and the spatial sampling equidistantly to 50 m. For the time-stepping we use the classical second-order leapfrog scheme, where we calculate the spatial derivatives with 8th-order FD stencils. We place a 5 Hz pressure source (Ricker wavelet) in the middle of the modeling

domain and record the wavefield with 101 pressure receivers, which we line up on the 3D diagonal.

Strong scaling benchmark

For the strong scaling benchmark the 3D model contains a constant number of 300 grid points (GPs) in each direction, which results in a total of $27 \cdot 10^6$ GPs. Figure 1 illustrates the results. The CPU benchmark shows a nearly linear scaling behavior up to four CPU nodes. For a larger number of CPU nodes we observe efficiency below linear scaling, most likely due to the decreased workload per PU combined with the increased communication. Comparing the results of the CPU nodes to the same number of GPU devices, we find the GPU runtimes to be between 34% and 46% lower. However, the GPU scaling efficiency is lower than its CPU counterpart. This is due to the fact that the GPU performance depends more on its utilization than the CPU, which causes the strong scaling behavior to stop earlier.

Weak scaling benchmark

For the weak scaling benchmark we increase the problem size with the number of PUs. Thereby, the workload and utilization is kept constant at $13.5 \cdot 10^6$ GPs per PU during the benchmark. We start with the grid dimensions of $150 \times 300 \times 300$ GPs for a single PU and end with the dimensions of $600 \times 600 \times 600$ GPs for 16 PUs. Figure 2 shows the results. When comparing CPU to GPU performance, we observe 40% to 45% lower runtimes when using GPU devices compared to the same number of CPU nodes. This supports the statement that the GPU's strong scaling performance is effected by the decreased workload. In the case of our weak scaling benchmark the total runtime should ideally stay constant with the number of PUs; the upper limit would be reached if the parallelization over the PUs introduces no overhead. In our results we see an increase of the runtime between the single PU and the 16 PU execution, for the CPU by 17% and for the GPU by 27%. In both cases the introduced overhead comes by virtue of the needed halo communication, which is particularly bad for 2 GPU PUs because of higher communication needs, since in this case the manual partitioning cannot be done as 3D cubes, but only as 2D slices. Furthermore, the GPU execution benefits from the asynchronous execution of communication and computation.

CONCLUSION

We have developed an efficient and hardware-independent 3D elastic finite-difference forward solver based on matrix-vector products. We use the HPC framework LAMA for the execution on CPUs and GPUs. The framework relieves us from the actual HPC implementation, such as parallelization and hardware-dependent adaption, and consequently it makes maintenance of our software relatively easy. In a strong

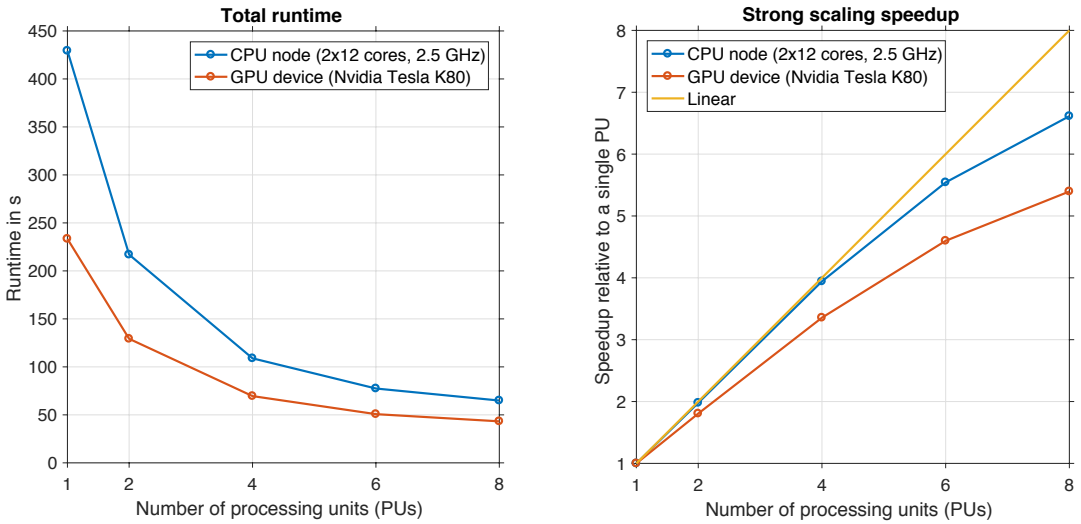

Figure 1: Results of the strong scaling benchmark.

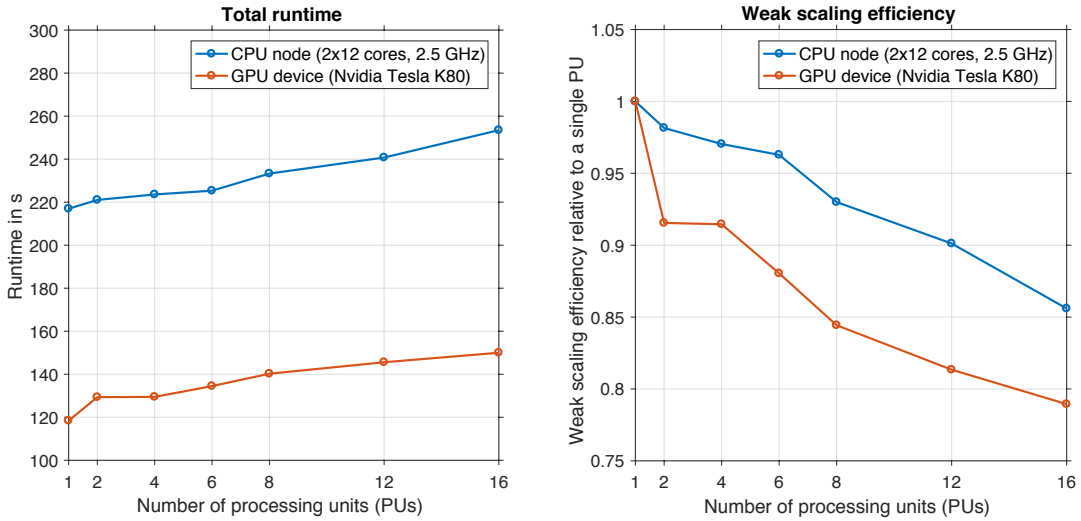

Figure 2: Results of the weak scaling benchmark.

and weak scaling benchmark we found that the time-to-solution can be decreased by either increasing the number of CPUs/GPUs or by replacing CPU nodes with GPU devices. We observed good overall performance and convenient execution of the same software code on CPU- and GPU-based architectures.

ACKNOWLEDGMENTS

This work is financially supported by the German Ministry of Education and Research (BMBF) through the project WAVE, grant 01IH15004A. The authors gratefully acknowledge the computing time granted by the John von Neumann Institute for Computing (NIC), provided on the supercomputer JURECA at Jülich Supercomputing Centre (JSC). Furthermore, we thank Fraunhofer SCAI for providing the open-source framework LAMA as well as for extensive support. This work was kindly supported by the sponsors of the *Wave Inversion Technology (WIT) Consortium.*

REFERENCES

Brandes, T., Schricker, E., and Soddemann, T. (2017). The LAMA approach for writing portable applications on heterogenous architectures. *Projects and Products of Fraunhofer SCAI, Springer. Accepted, to be published*.

Krause, D. and Thörnig, P. (2016). Jureca: General-purpose supercomputer at Jülich supercomputing centre. *Journal of large-scale research facilities JLSRF*, 2:62.

Virieux, J. (1986). P-SV wave propagation in heterogeneous media: Velocity-stress finite-difference method. *Geophysics*, 51(4):889–901.