IMAGING ON THE EDGE: GPU-ACCELERATED TRAVELTIME TOMOGRAPHY ON A JETSON NANO

Ban-Sok Shin, Luis Wientgens, Dmitriy Shutin

German Aerospace Center
Institute of Communications and Navigation
E-Mail: {ban-sok.shin, luis.wientgens, dmitriy.shutin}@dlr.de

ABSTRACT

We present a GPU-accelerated, computationally efficient seismic imaging approach implemented in CUDA for use on NVIDIA's edge device, the Jetson Nano. The presented implementation intends to enable fast seismic imaging in autonomous exploration tasks on robotic platforms where electrical power and weight are limiting factors. For the imaging method we consider traveltime tomography, a seismic imaging technique based on first-arrival traveltimes. To this end, we rely on the fast iterative method that solves the eikonal equation in a parallel fashion and provides first-arrival traveltime maps. Furthermore, we employ a gradient-based ray tracer to reconstruct wave paths through the subsurface. We implement ray tracer and parts of the tomography update in a parallel fashion for efficient computation on the GPU of the Jetson Nano. We demonstrate the imaging capability of our implementation for synthetic as well as real seismic data. We also show that our edge device implementation achieves imaging results in a comparable time range as a state-of-the-art geophysical inversion tool running on a powerful desktop CPU.

Index Terms— Seismic imaging, traveltime tomography, edge devices, CUDA, GPGPU

1. INTRODUCTION

Seismic imaging is done in an offline-processing manner where measurement data from receivers are processed at a remote location with high computational capabilities. This is mainly due to the high amount of data that is recorded. Recently, powerful and compact edge devices with high computational capabilities such as Nvidia's Jetson Nano are available on the market. Such a device is equipped with a graphics processing unit (GPU) that allows for highly parallelized computations. In light of such edge devices the question arises to which extent seismic imaging methods can be performed on them. In this work, we intend to shed light on this question.

Performing imaging on such devices comes with a variety of advantages. Due to their small form factor they are highly suited for applications where mobility, portability and on-site processing play an important role. Furthermore, such implementation will enable faster imaging at the location where the seismic survey takes place. One specific application example is the autonomous exploration of planetary subsurfaces by multiple robotic agents [1, 2, 3]. Here, a network of multiple robotic agents conducts a seismic survey in a cooperative manner to image interesting features within the near-surface. This shall be done without forwarding measurement data via a remote connection to Earth for processing but directly within the network of agents. To this end, the complete imaging task needs to be conducted locally on the agents' computing boards. Furthermore, since the agents need to be mobile, heavy computing hardware is not suited for such a task. Hence, imaging on a portable device such as the Jetson Nano is highly relevant for such an application.

In this work, we specifically focus on traveltime tomography (TT), an imaging method in geophysical exploration. We present an implementation of TT that performs the imaging task with reasonable spatial resolution within a few seconds on a Jetson Nano device. To this end, we implement the complete TT on the GPU of the Jetson Nano and exploit parallelization where possible. The related kernels are written in the CUDA programming language to enable direct implementation on NVIDIA GPUs. We test our implementation with both synthetic as well as real seismic data. For the real data set we employ measurement data from a seismic refraction survey over a highway tunnel. We show that our proposed implementation recovers the main structure of the tunnel. With regard to timing we compare our implementation to a state-of-the-art inversion package that runs in parallel over multiple CPU cores on a powerful desktop machine. We demonstrate that tomographic imaging is realizable in a computationally efficient way on a compact, mobile, low-power edge device such as the Jetson Nano.

2. TRAVELTIME TOMOGRAPHY

2.1. Main Algorithm

TT is a seismic imaging technique that employs first-arrival traveltimes to reconstruct a subsurface image w.r.t. the P-wave velocity [4]. To acquire seismic data we consider N_R geophones or receivers and N_S sources which are released in a sequential manner. A seismic source can be e.g. a hammer strike, an explosive or a vibrating source [5]. Receivers as well as sources are placed on the surface.

In its ordinary form TT deals with the minimization of a least-squares cost between measured traveltimes $\boldsymbol{t}_\mathrm{obs} \in \mathbb{R}^{N_\mathrm{R}}$ and synthesized traveltimes $\boldsymbol{t}_\mathrm{syn} \in \mathbb{R}^{N_\mathrm{R}}$ of all N_R receivers w.r.t. the P-wave slowness $\boldsymbol{m}(\boldsymbol{x})$ (i.e., inverse velocity) over spatial coordinate $\boldsymbol{x} \in \mathbb{R}^2$. Assuming a spatial discretization using e.g. finite differences of the two-dimensional subsurface domain $\Omega \subset \mathbb{R}^2$ into $N_x \times N_z$ grid points, the P-wave slowness has dimension $\boldsymbol{m} \in \mathbb{R}^{N_x N_z}$. The optimization problem of TT reads

$$\min_{\boldsymbol{m} \in \mathbb{R}^{N_x N_z}} \mathcal{J}(\boldsymbol{m}) = \frac{1}{2} \sum_{s=1}^{N_\mathrm{S}} ||\boldsymbol{t}_{\mathrm{syn},s}(\boldsymbol{m}) - \boldsymbol{t}_{\mathrm{obs},s}||_2^2. \qquad (1)$$

The measured traveltimes $\boldsymbol{t}_\mathrm{obs}$ are obtained by picking first-arrival events of the P-wave in the measured seismograms, either manually

The work leading to this publication was partially funded by the German Research Foundation (DFG) under grants SH 1975/1-1 and SH 1743/3-1.

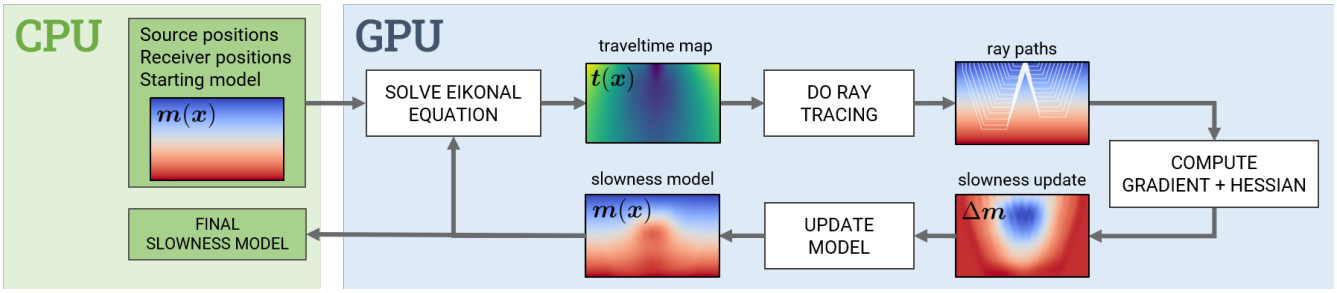

Fig. 1: Schematic overview of proposed implementation of the traveltime tomography on a GPU. The CPU generates the starting model that is used to solve the eikonal equation on the GPU. Then rays are traced using Runge-Kutta on the traveltime map $\boldsymbol{t}(\boldsymbol{x})$. The rays are used to compute the gradient and Hessian of the inverse problem. From this an update is calculated in form of a slowness perturbation that is applied to the model. This process is repeated until a satisfactory imaging result is obtained.

or by a picker algorithm. To synthesize traveltimes $\boldsymbol{t}_{\text{syn}}(\boldsymbol{m})$ the *eikonal equation*

$$|\nabla \boldsymbol{t}(\boldsymbol{x})|^2 = \boldsymbol{m}(\boldsymbol{x})^2, \quad \text{s.t.} \quad \boldsymbol{t}(\boldsymbol{x}_s) = 0, \boldsymbol{x} \in \Omega. \tag{2}$$

needs to be solved w.r.t. $\boldsymbol{t}(\boldsymbol{x})$. This provides a map of first-arrival traveltimes based on $\boldsymbol{m}$ which is sampled at receiver positions $\boldsymbol{x}_r, r = 1, \ldots, N_{\text{R}}$ to give $\boldsymbol{t}_{\text{syn}}(\boldsymbol{m})$. The operator ∇ is a discretized spatial gradient, while $\boldsymbol{t}(\boldsymbol{x})$ is the discretized traveltime map. As initial condition, the traveltime has to be zero at source position $\boldsymbol{x}_s$. The eikonal equation (2) can be solved via the fast marching method [6] or fast sweeping method [7]. However, both these methods do not allow for a parallelized computation of the traveltime map $\boldsymbol{t}(\boldsymbol{x})$. In contrast, the fast iterative method (FIM) proposed in [8] solves the eikonal equation in a parallel fashion allowing for an efficient implementation on a GPU. In more detail, grid points where traveltimes need to be updated are stored in a structure called *active list*. This list contains grid points where the traveltimes have not converged yet. The key step here is that all grid points in the active list are updated at once without any special order. This allows for easy parallelization on a GPU. This is in contrast to FMM which uses a heap that sorts the update order of the grid points [9]. Thus, in our implementation we employ the FIM as forward solver of the eikonal equation.

For TT, problem (1) needs to be solved subject to the sampled traveltime map obtained by (2). Due to the absolute norm included in the eikonal equation (2) TT is a nonlinear inverse problem w.r.t. $\boldsymbol{m}$. Furthermore, cost (1) is highly non-convex containing multiple local minima. To obtain a subsurface image $\boldsymbol{m}$, TT employs a steepest-descent strategy. To this end, we require the gradient of $\mathcal{J}(\boldsymbol{m})$ w.r.t. $\boldsymbol{m}$ which can be obtained by ray tracing [10], by the adjoint-state method [11, 12] or by calculating sensitivities [6]. Here, we employ a ray tracer that provides ray paths between source and receivers through the subsurface domain Ω since it can be easily parallelized over the receivers as proposed in [13]. Using ray tracing the forward problem of synthesizing traveltimes $\boldsymbol{t}_{\text{syn}}(\boldsymbol{m})$ can be linearized. From ray theory it is known that by integrating over a ray path one obtains the time the ray travels along that path [10]. For the discrete case, this line integral can be approximated by a sum through all grid points $i = 1, \ldots, N_x N_z$ in the discretized spatial domain. For each grid point i, the contribution of the ray within the current point l_i is multiplied by the respective slowness value at point i: $t_{\text{ray}} \approx \sum_{i=1}^{N_x N_z} l_i \boldsymbol{m}(\boldsymbol{x}_i)$. If for each ray r and source s the respective segments in each grid cell i are collected in a matrix $\boldsymbol{G}_s \in \mathbb{R}^{N_{\text{R}} \times N_x N_z}$ the synthesized traveltimes can be approximated via the linear model

$$\boldsymbol{t}_{\text{syn},s}(\boldsymbol{m}) \approx \boldsymbol{G}_s \boldsymbol{m}. \tag{3}$$

Matrix $\boldsymbol{G}_s$ is the *sensitivity matrix* and its r-th row contains all ray segments over all grid cells for receiver and ray r through the subsurface domain for source s. By replacing $\boldsymbol{t}_{\text{syn},s}(\boldsymbol{m})$ in (1) with (3) gradient and Hessian can be computed directly via

$$\frac{\partial \mathcal{J}(\boldsymbol{m})}{\partial \boldsymbol{m}} = \sum_{s=1}^{N_{\text{S}}} \boldsymbol{G}_s^{\mathsf{T}} (\boldsymbol{G}_s \boldsymbol{m} - \boldsymbol{t}_{\text{obs}}), \; \frac{\partial^2 \mathcal{J}(\boldsymbol{m})}{\partial \boldsymbol{m} \, \partial \boldsymbol{m}^{\mathsf{T}}} = \sum_{s=1}^{N_{\text{S}}} \boldsymbol{G}_s^{\mathsf{T}} \boldsymbol{G}_s. \tag{4}$$

To improve estimates of the slowness model $\boldsymbol{m}$ iteratively Newton's method can be applied using gradient and Hessian information. For numerical stability in the inversion of the Hessian a scaled identity matrix is added:

$$\begin{aligned} \boldsymbol{m} &\leftarrow \boldsymbol{m} - \Delta \boldsymbol{m} \\ &= \boldsymbol{m} - \sum_{s=1}^{N_{\text{S}}} (\boldsymbol{G}_s^{\mathsf{T}} \boldsymbol{G}_s + \gamma \boldsymbol{I})^{-1} \boldsymbol{G}_s^{\mathsf{T}} (\boldsymbol{t}_{\text{syn},s} - \boldsymbol{t}_{\text{obs},s}). \end{aligned} \tag{5}$$

In each iteration, the eikonal equation (2) needs to be solved for the current slowness model $\boldsymbol{m}$. Afterwards, rays between source and receivers are traced to generate an updated sensitivity matrix $\boldsymbol{G}_s$. Using $\boldsymbol{G}_s$ the gradient and Hessian are computed following (4) and the slowness model $\boldsymbol{m}$ is updated (5), cf. Fig. 1. In a practical implementation a smoothing filter is applied on the update term in (5) to obtain slowness changes over a broader spatial area. Without applying such filter the slowness model $\boldsymbol{m}$ will only be changed along the ray paths in the sensitivity matrix $\boldsymbol{G}_s$.

2.2. Gradient-based Ray Tracer

In the following, we introduce the ray tracer that is used in our TT implementation. Besides various possibilities to implement a seismic ray tracer (see [10] for an overview), we rely on a gradient-based method since FIM provides a traveltime map $\boldsymbol{t}(\boldsymbol{x})$ in an efficient manner which can be fed into the ray tracer. In particular, we rely on our previous work [13] where a gradient-based, parallelized ray tracer has been successfully implemented on the GPU of a Jetson Nano module. However, in [13] an ordinary steepest-descent technique has been used. In this work, we employ a 4th-order Runge-Kutta scheme to enhance accuracy and stability of the ray tracer.

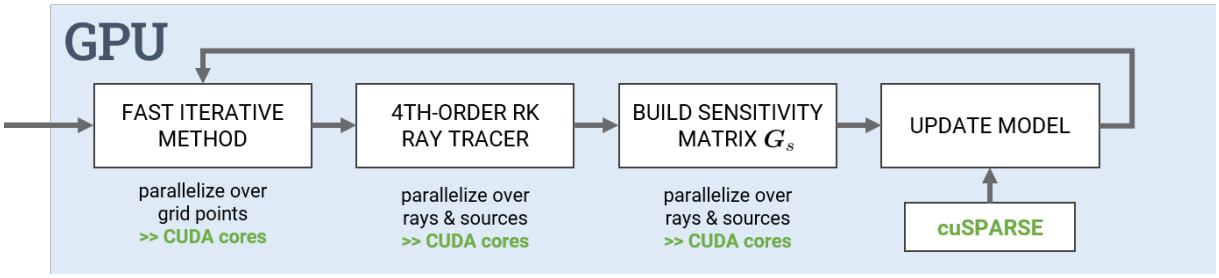

Fig. 2: Overview of building blocks for the implementation on the GPU with details w.r.t. parallelization. In the FIM, parallelization is implemented over the grid points in the active list. For the ray tracer and the sensitivity matrix, we utilize parallelization over the rays/receivers r and the sources s. To efficiently update the slowness model following (5) we employ a conjugate gradient solver implemented by using the `cuSPARSE` library to accelerate the involved matrix inversion.

In the traveltime map $\boldsymbol{t}(\boldsymbol{x})$ the source location $\boldsymbol{x}_s$ represents a global minimum. Hence, a steepest-descent scheme can be applied to trace a ray path from a receiver location to the source location. If $\boldsymbol{x}_k$ denotes the current coordinate of the ray path, the steepest-descent strategy is expressed by $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k + \alpha \nabla \boldsymbol{t}(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_k}$ where $\nabla \boldsymbol{t}(\boldsymbol{x})$ is the gradient of the traveltime map and α is a positive step size. Since each ray path is independent of any other ray path all rays can be traced in parallel, i.e., computation of gradient, angle and decision on the next ray coordinate are conducted on one thread in one CUDA core. For details the reader is referred to [13].

In the steepest-descent strategy the gradient of the traveltime map $\nabla \boldsymbol{t}(\boldsymbol{x})$ is evaluated at a single coordinate $\boldsymbol{x}_k$ only. This can result in inaccurate and diverging ray paths especially in highly heterogeneous media. To enhance accuracy of the ray tracer we use the 4th-order Runge-Kutta method [14] where the gradient $\nabla \boldsymbol{t}(\boldsymbol{x})$ is evaluated at four different positions. Afterwards gradient information from all four positions is used to obtain the next ray coordinate. The respective four positions are calculated via

$$\boldsymbol{x}_{k,1} = \boldsymbol{x}_k, \quad \boldsymbol{x}_{k,2} = \boldsymbol{x}_k + \frac{\alpha}{2} \nabla \boldsymbol{t}(\boldsymbol{x}_{k,1}), \tag{6}$$

$$\boldsymbol{x}_{k,3} = \boldsymbol{x}_k + \frac{\alpha}{2} \nabla \boldsymbol{t}(\boldsymbol{x}_{k,2}), \quad \boldsymbol{x}_{k,4} = \boldsymbol{x}_k + \alpha \nabla \boldsymbol{t}(\boldsymbol{x}_{k,3}). \tag{7}$$

The ray coordinate is then updated by

$$\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k + \frac{\alpha}{6} \big(\nabla \boldsymbol{t}(\boldsymbol{x}_{k,1}) + \nabla \boldsymbol{t}(\boldsymbol{x}_{k,2}) + \nabla \boldsymbol{t}(\boldsymbol{x}_{k,3}) + \nabla \boldsymbol{t}(\boldsymbol{x}_{k,4}) \big) \tag{8}$$

Again, since computation of a ray path is independent of any other ray the algorithm can be easily parallelized over the rays.

3. IMPLEMENTATION DETAILS

To use the parallel compute power of the Jetson Nano we implement critical sections of the imaging scheme as GPU kernels using CUDA. We exploit the CUDA programming model and GPU capabilities at four different instances in the imaging process, i.e., in the eikonal solver, the ray tracer, the construction of the sensitivity matrix and the update of the slowness model. Fig. 2 gives an overview of the different processing blocks with information about parallelization.

3.1. Eikonal solver

As described in Section 2 the FIM can update traveltimes at multiple grid points in parallel in every iteration, leading to faster computation of the traveltime map $\boldsymbol{t}(\boldsymbol{x})$ compared to sequential solvers such as FMM and FSM. One iteration in FIM comprises three parallel kernels that conduct the following operations: Firstly, updating travel times in the active list, then reducing the list by removing converged grid points and lastly propagating changes to directly neighbouring grid points. As soon as the active list is empty, all computed traveltimes have converged and FIM outputs the final traveltime map. We use the original FIM code which can be accessed on GitHub[1].

3.2. Ray tracing

The ray tracer is implemented in a data parallel way over the total number of rays which is equal to the number of receivers N_{R}. Each block in the kernel grid on the GPU is responsible for one source s and executes with the number of receivers N_{R} as thread count. Therefore, each thread processes tracing of one ray. The resulting rays are written into a pre-allocated continuous linear memory region. Since in CUDA kernels use fixed size static arrays sufficient memory space needs to be allocated in advance for the rays. To this end, we conservatively assume three times the largest edge of the subsurface domain Ω, i.e., either N_x or N_z. In scenarios with obstacles that could lead to rays longer than traversing the edges of the domain (e.g. obstacles that have to be circumvented), a different heuristic is needed to assure appropriate space for each ray. We choose the described approach to avoid having to reallocate device memory during ray tracing. By parallelizing the ray tracer we can consider measurement scenarios with more receivers participating, without incurring significant, additional run time cost. Following the 4th-order Runge-Kutta in Section 2 the next point in the ray is determined by the direction of steepest descent. In the discrete domain we need to map the next point of the ray to the grid point that coincides best with the calculated descent angle. To this end, we partition the unit circle into multiple sections that define decision boundaries for the next point. We implemented the option to use either 8 or 16 sections, respectively. In general, the utilized ray tracing approach works best for smooth velocity variations.

3.3. Sensitivity matrix and model update

In TT, the matrix $\boldsymbol{G}_s$ needs to be rebuilt for each source s in each iteration based on the updated slowness model $\boldsymbol{m}$. This impact can also be offset by parallelization using the inherent data parallelism in the building process. Each row in each $\boldsymbol{G}_s$ corresponding to one ray r traced through the domain for source s can be independently

[1] https://github.com/SCIInstitute/StructuredEikonal

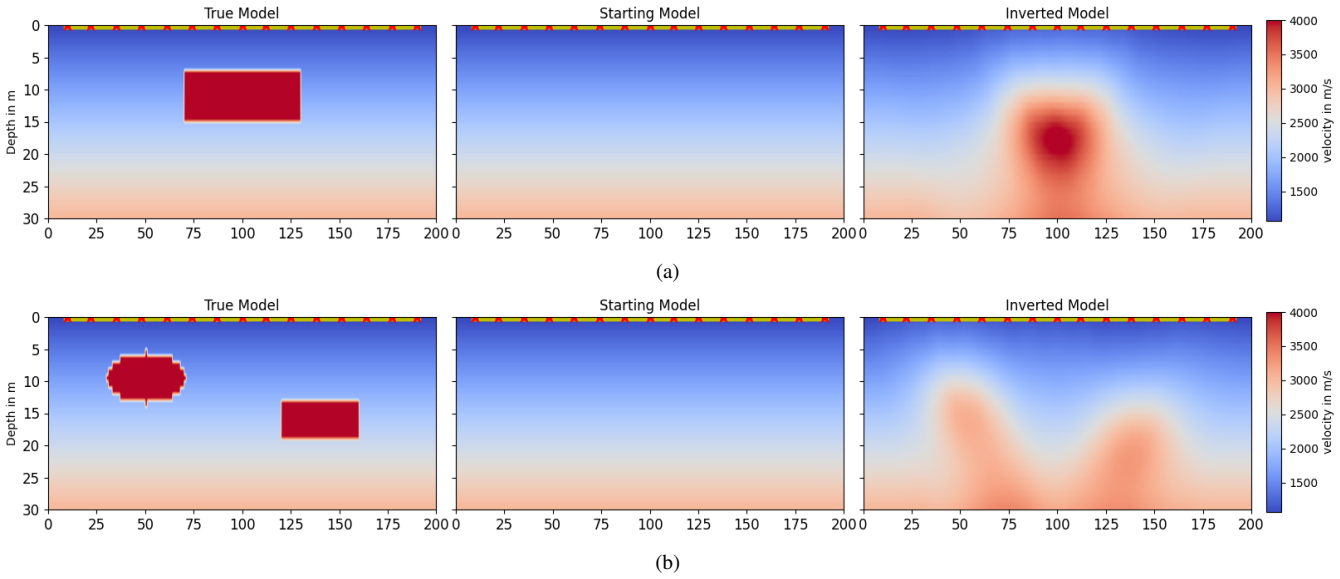

Fig. 3: Imaging results of our implementation onb the Jetson Nano for two scenarios with synthetic velocity anomalies: (a) symmetric rectangular high velocity region, (b) asymmetric scenario with two high velocity regions. Receiver and source positions are depicted by yellow and red markers, respectively.

calculated by its own thread. One thread scans the ray r corresponding to the respective row and populates it with values for the segment lengths of the ray. For this purpose we interpret the grid points as nodes forming square cells. A row of $\boldsymbol{G}_s$ is then a representation of the segment length for each cell in the computational domain.

In the update (5) of the subsurface model $\boldsymbol{m}$ a matrix of dimension $N_x N_z \times N_x N_z$ needs to be inverted for each source s. Clearly, such computation will require long processing time. To reduce the computation time we use a general matrix-vector linear solver in each iteration for each source s. More specifically, we employ parallel sparse matrix-vector multiplication and sparse decomposition for the system matrix $(\boldsymbol{G}_s^\mathsf{T}\boldsymbol{G}_s+\gamma\boldsymbol{I})^{-1}$ from the cuSPARSE library provided by NVIDIA to implement a conjugate gradient (CG) method [15]. The CG method is known to be particularly suited for large sparse linear systems as we encounter here. Due to memory limitations of the Jetson Nano we solve the linear system of equations for each source s, separately. In general, forming a block matrix over all sources and applying the CG method is possible but can quickly occupy the complete memory of the Jetson Nano. In this case, swapping data needs to be used which again increases computation time.

In the implementation, the system matrix in (5) is only invertible for a high regularization parameter γ due to the singular matrix product $\boldsymbol{G}_s^\mathsf{T}\boldsymbol{G}_s$. To compensate for the strong regularization we build the mean over the contributions from all sources in the update vector $\Delta\boldsymbol{m}$ and add a scaling factor α that decays exponentially to enable a smooth cost minimization, i.e., $\boldsymbol{m} \leftarrow \boldsymbol{m} - \frac{\alpha}{N_\mathrm{S}}\sum_{s=1}^{N_\mathrm{S}}(\boldsymbol{G}_s^\mathsf{T}\boldsymbol{G}_s+\gamma\boldsymbol{I})^{-1}\boldsymbol{G}_s^\mathsf{T}(\boldsymbol{t}_{\mathrm{syn},s}-\boldsymbol{t}_{\mathrm{obs},s})$. Such modification of the subsurface model update leads to a numerically stable implementation of TT.

4. PERFORMANCE EVALUATION

To evaluate our implementation we focus on imaging results using synthetic as well as real seismic data recorded over a highway tunnel. We employ a Jetson Nano with 2GB RAM and 2 streaming multiprocessors (SM). Furthermore, we compare timing performance against a CPU implementation using the pyGIMLi package [16] that does not have to abide by power or space constraints as our implementation on the Jetson Nano. The desktop machine uses an i7-1185G7 CPU and has 16GB RAM available.

4.1. Imaging synthetic velocity anomalies

We report results for two scenarios with synthetic subsurface models. The first scenario is a symmetric setup with a rectangular high velocity zone. Secondly we consider an asymmetric setup with a rectangular and an ellipsoidal anomaly. For both scenarios we use $N_\mathrm{R} = 128$ receivers and $N_\mathrm{S} = 15$ sources placed on the surface. We assume a regular grid spacing with a cell size of $\Delta x = \Delta z = 1\,\mathrm{m}$ that results in grid size of 30×200 points. For TT we set $\gamma = 1000, \alpha = 10$ and $N_\mathrm{TT} = 4$. Furthermore, for the Gaussian filter we choose a high standard deviation in the range of $\sigma = 3, \ldots, 5$ to smooth out the model update $\Delta\boldsymbol{m}$.

Fig. 3 depicts the imaging results for both scenarios with the respective true velocity model and the starting model that is used by our TT implementation on the Jetson Nano. One can observe that for both cases the anomalies are visible in the reconstructed images in particular for the rectangular anomaly. In both images the resolution is low. This is due to the resolution limits imposed by the first-arrival traveltime data [17]. In Tab. 1 we compare computation time of our implementation against PyGIMLi running on a desktop machine. Here we additionally report computation times for models with a lower resolution to demonstrate the impact of model scale. One can see that PyGIMLi consumes lower computation time for the considered examples than our implementation on the Jetson Nano. However, one should note that PyGIMLi runs on a powerful desktop machine that consumes a multiple of power compared to the Jetson Nano that draws $5 - 10\,\mathrm{W}$ only. Furthermore, PyGIMLi exploits parallel computing over multiple CPU cores. Under this consideration, our implementation achieves reasonable computation times.

Scenario	Resolution	Method	Time
Box	140×30	Ours	22.25 s
		PyGIMLi	4.27 s
	200×30	Ours	51.32 s
		PyGIMLi	7.68 s
Box + Ellipse	140×30	Ours	21.76 s
		PyGIMLi	7.18 s
	200×30	Ours	42.04 s
		PyGIMLi	14.15 s

Table 1: Timing results for our implementation on the Jetson Nano and PyGIMLi running on a desktop CPU.

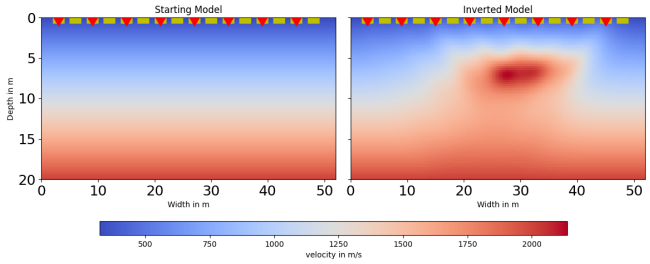

Fig. 4: Imaging result for real traveltime data recorded over a highway tunnel [18]. The source and receivers are placed at the surface and marked in red and yellow, respectively.

4.2. Imaging a highway tunnel

In this example, we use real seismic measurement data that was recorded over a highway tunnel using hammer strikes as a source [18]. For the measurements we used a receiver line of $N_{\mathrm{R}} = 16$ geophones with $N_{\mathrm{S}} = 8$ shot positions. To image the tunnel we use a discretization of $\Delta x = \Delta z = 0.25\,\mathrm{m}$ for the spatial domain. We use $\gamma = 100$, $\alpha = 3$, $\sigma = 5$ and $N_{\mathrm{TT}} = 6$ as parameters for TT. In this scenario, our Jetson Nano implementation requires $39.60\,\mathrm{s}$ of computation time. The imaging results using our proposed TT implementation are illustrated in Fig. 4. One can clearly see that the tunnel structure is recovered by TT. Based on ground truth information of the measurement site the highway tunnel is located at approximately $4 - 5\,\mathrm{m}$ depth and $18\,\mathrm{m}$ distance from the first geophone of the array. These data also coincide well with the reconstructed image. Our imaging result shows the ability of TT to resolve the top and boundaries of the tunnel. Resolving the lateral boundaries is possible albeit blurred.

5. CONCLUSION

We presented an efficient implementation of the traveltime tomography on the NVIDIA Jetson Nano, a GPU-equipped edge device. Our proposed implementation places the complete tomography method on the GPU where forward solver, ray tracing and slowness model update are all implemented in a parallelized fashion to run efficiently over multiple CUDA cores. With regard to timing performance, we compared our implementation with the package PyGIMLi that is suited for desktop CPUs. Our implementation on the compact and power-limited Jetson Nano, although slower, showed to be in comparable range with PyGIMLi. Further optimization of the inversion involved in the slowness model update will decrease this gap.

6. REFERENCES

[1] A. Wedler et al., "German Aerospace Center's advanced robotic technology for future lunar scientific missions," *Phil. Trans. R. Soc. A*, vol. 379, no. 2188, Nov. 2020.

[2] S. W. Courville et al., "ARES: An Autonomous Roving Exploration System for Planetary Active-Source Seismic Data Acquisition," in *AGU Fall Meeting*, Dec. 2018.

[3] B.-S. Shin and D. Shutin, "Joint distributed traveltime and full waveform tomography for enhanced subsurface imaging in seismic networks," *IEEE Trans. Comput. Imaging*, 2024.

[4] D. J. White, "Two-Dimensional Seismic Refraction Tomography," *Geophysical Journal International*, vol. 97, no. 2, pp. 223–245, 1989.

[5] A. E. Mussett et al., *Looking into the Earth*, Cambridge University Press, Oct. 2000.

[6] E. Treister and E. Haber, "A fast marching algorithm for the factored eikonal equation," *J. Comput. Phys.*, vol. 324, pp. 210–225, 2016.

[7] Y. R. Tsai et al., "Fast Sweeping Algorithms for a Class of Hamilton–Jacobi Equations," *SIAM Journal on Numerical Analysis*, vol. 41, no. 2, pp. 673–694, 2003.

[8] W. K. Jeong and R. Whitaker, "A Fast Iterative Method for Eikonal Equations," *SIAM Journal of Scientific Computing*, 2008.

[9] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, Jan. 1999.

[10] N. Rawlinson and M. Sambridge, "Seismic Traveltime Tomography of the Crust and Lithosphere," *Advances in Geophysics*, vol. 46, pp. 81–198, 2003.

[11] R. Plessix, "A review of the adjoint-state method for computing the gradient of a functional with geophysical applications," *Geophysical Journal International*, vol. 167, no. 2, pp. 495–503, 2006.

[12] S. Leung and J. Qian, "An adjoint state method for three dimensional refraction tomography," *Commun Math Sci*, vol. 4, no. 1, pp. 249–266, 2006.

[13] B.-S. Shin et al., "Parallel 2D Seismic Ray Tracing Using CUDA on a Jetson Nano," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, June 2023.

[14] W. Press et al., *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, 2007.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, 2 edition, 1999.

[16] C. Rücker et al., "pyGIMLi: An open-source library for modelling and inversion in geophysics," *Computers and Geosciences*, vol. 109, pp. 106–123, 2017.

[17] F. A. Dahlen, "Resolution limit of traveltime tomography," *Geophysical Journal International*, vol. 157, no. 1, pp. 315–331, 04 2004.

[18] B.-S. Shin et al., "Near-Surface Seismic Measurements in Gravel Pit, over Highway Tunnel and Underground Tubes with Ground Truth Information as an Open Data Set," *Sensors*, vol. 22, no. 17, 2022.