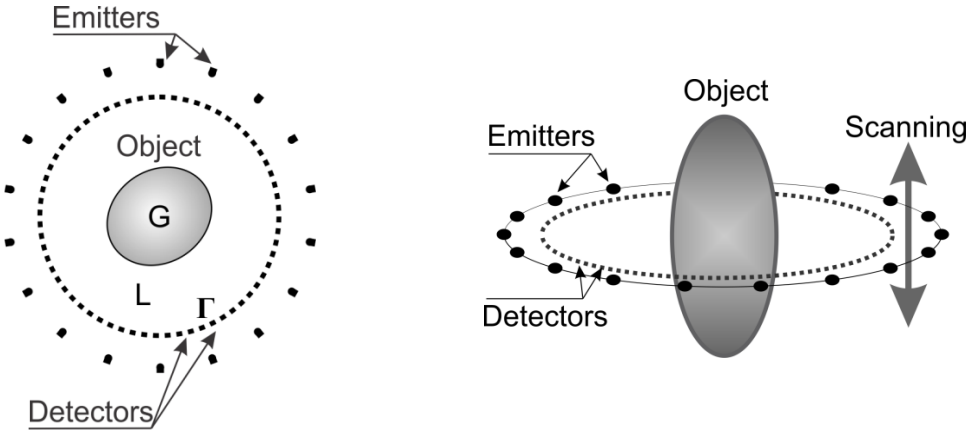
WaveTomography 1.0 software
User manual

1 Formulation of the inverse problem of wave tomography and its solution method

The wave tomography technology aims to obtain an image of the internal structure of an object via sounding the object with waves and measuring transmitted and scattered waves around the object. Main applications are medical imaging, nondestructive testing, seismic studies. The basic scheme of a tomographic examination is shown in Figure 1a. The object being imaged occupies region G. In medical imaging, region L is filled with water with known properties. The detectors are located on circle Γ. The objective is to reconstruct the speed of sound in region G using the ultrasonic waves radiated from the emitters, scattered by the object and registered by the detectors.



(a) (b)

Figure 1: Scheme of tomographic examination (a), layer-by-layer 3D tomography (b)

Figure 1b shows the scheme of the layer-by-layer 3D wave tomography. Emitters and detector are located in a horizontal plane that can shift vertically. The images are acquired in multiple horizontal imaging planes, yielding a representation of a 3D object in the form of a stack of 2D cross-sections.

The inverse problem of wave tomography is posed as a coefficient inverse problem, in which the unknowns are the speed of sound and the absorption factor at each point of the object. A scalar wave model based on a second-order hyperbolic differential equation (1) is used to simulate the process of wave propagation numerically. This model accounts for diffraction, refraction, multiple scattering and absorption of ultrasound waves.

$$c(\boldsymbol{r})u_{tt}(\boldsymbol{r},t) + a(\boldsymbol{r})u_t(\boldsymbol{r},t) - \Delta u(\boldsymbol{r},t) = 0; \tag{1}$$

$$u(\boldsymbol{r},t)|_{t=0} = F_0(\boldsymbol{r}), \quad u_t(\boldsymbol{r},t)|_{t=0} = F_1(\boldsymbol{r}). \tag{2}$$

Here, $u(\boldsymbol{r},t)$ is the acoustic pressure; $c(\boldsymbol{r}) = 1/v^2(\boldsymbol{r})$, where $v(\boldsymbol{r})$ is the speed of sound; $a(\boldsymbol{r})$ is the absorption factor; $\boldsymbol{r} = \{x, y\}$ is a point in the imaging plane, and Δ is the Laplacian with respect to $\boldsymbol{r}$. Initial conditions (2) represent the wavefield at the initial time of the numerical simulation.

Finite-difference time-domain method (FDTD) is employed to solve equations (1) – (2). We define a uniform rectangular finite difference grid: $x_i = ih, y_j = jh, t_k = k\tau; i, j =$

$1, ..., N, k = 1, ..., M$, where h is the spatial discretization step, and τ is the time step. A second-order finite difference scheme approximates equation (1):

$$c_{ij}\frac{u_{ij}^{k+1} - 2u_{ij}^{k} + u_{ij}^{k-1}}{\tau^2} + a_{ij}\frac{u_{ij}^{k+1} - u_{ij}^{k-1}}{2\tau} - \frac{\boldsymbol{L}_{ij}^{k}}{h^2} = 0. \tag{3}$$

Here, $u_{ij}^{k} = u(x_i, y_j, t_k)$ are the values of $u(\boldsymbol{r}, t)$ at point (i, j) at the time step k; c_{ij} and a_{ij} are the values of $c(\boldsymbol{r})$ and $a(\boldsymbol{r})$ at point (i, j). The first term approximates $c(\boldsymbol{r})u_{tt}(\boldsymbol{r}, t)$, the second term approximates $a(\boldsymbol{r})u_t(\boldsymbol{r}, t)$. The discrete Laplacian is denoted by $\boldsymbol{L}_{ij}^{k}$. A fourth-order numerical approximation on a 5×5-point stencil is used for the discrete Laplacian:

$$\boldsymbol{L}_{ij}^{k} = \sum_{i=i_0-2}^{i_0+2}\sum_{j=j_0-2}^{j_0+2} v_{ij}u_{ij}^{k}. \tag{4}$$

Collecting the terms with u_{ij}^{k+1} in (3), we obtain an explicit finite-difference scheme for the wave equation (1):

$$u_{ij}^{k+1} = \left(2\frac{c_{ij}}{\tau^2}u_{ij}^{k} - \boldsymbol{L}_{ij}^{k} + \left(\frac{a_{ij}}{2\tau} + \frac{c_{ij}}{\tau^2}\right)u_{ij}^{k-1}\right)\left(\frac{a_{ij}}{2\tau} + \frac{c_{ij}}{\tau^2}\right)^{-1}. \tag{5}$$

This scheme allows us to compute the wavefield $u(\boldsymbol{r}, t)$ sequentially in time, starting witn initial conditions (2). The parameters h and τ are related by the Courant stability condition $c^{-0.5}\tau < h/\sqrt{2}$. For the problem considered, we used a time step equal to $\tau = 0.3c_0^{0.5}h$, which ensured the stability of the finite difference method.

Solving two-dimensional equation (1) using explicit difference schemes involves performing $O(N^2T)$ operations to compute the wavefield propagation, where N is the number of grid points along spatial dimensions, and T is the number of time steps. Such amount of computation in a reasonable time is a feasible task for modern supercomputers.

In order to solve the inverse problem of wave tomography, a direct problem is solved to obtain the boundary values and the simulated wavefield $u(\boldsymbol{s}, t)$ at the detectors assuming the current approximate values of the coefficients $c(\boldsymbol{r})$ and $a(\boldsymbol{r})$. The direct problem is solved for each ultrasound emitter at each iteration of the gradient-descent solution method.

The inverse problem of wave tomography is a ill-posed coefficient inverse problem for the wave equation (1). The objective is to determine the speed of sound $c(\boldsymbol{r})$ and absorption factor $a(\boldsymbol{r})$ inside the medium, while the wavefield $u(\boldsymbol{r}, t)$ is known only at the detector positions. The software being ported in this project solves the inverse problem via minimizing the residual functional

$$\Phi(u(c, a)) = \frac{1}{2}\int_0^T\int_S (u(\boldsymbol{s}, t) - U(\boldsymbol{s}, t))^2 \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}t \tag{6}$$

for its argument (c, a). Here $U(\boldsymbol{s}, t)$ are the data measured at surface Γ for the time period $(0, T)$, $u(\boldsymbol{s}, t)$ is the solution of the direct problem (1)–(2) for the given $c(\boldsymbol{r}) = 1/v^2(\boldsymbol{r})$ and $a(\boldsymbol{r})$. The residual functional is the sum of the residuals (6) obtained for each ultrasound emitter.

The gradient $\Phi'(u(c,a)) = \{\Phi'_c(u), \Phi'_a(u)\}$ of the functional (6) with respect to the variation of the sound speed and absorption factor $\{dc, da\}$ has the form:

$$\Phi'_c(u(c)) = \int_0^T w_t(\boldsymbol{r},t)u_t(\boldsymbol{r},t)\,\mathrm{d}t, \quad \Phi'_a(u(a)) = \int_0^T w_t(\boldsymbol{r},t)u(\boldsymbol{r},t)\,\mathrm{d}t. \tag{7}$$

Here $u(\boldsymbol{r},t)$ is the solution of the direct problem (1)–(2), and $w(\boldsymbol{r},t)$ is the solution of the "conjugate" problem with the given $c(\boldsymbol{r})$, $a(\boldsymbol{r})$, and $u(\boldsymbol{r},t)$:

$$c(\boldsymbol{r})w_{tt}(\boldsymbol{r},t) - a(\boldsymbol{r})w_t(\boldsymbol{r},t) - \Delta w(\boldsymbol{r},t) = E(\boldsymbol{r},t); \tag{8}$$

$$w(\boldsymbol{r},t=T) = 0, \quad w_t(\boldsymbol{r},t=T) = 0; \tag{9}$$

$$E(\boldsymbol{r},t) = \begin{cases} u(\boldsymbol{r},t) - U(\boldsymbol{r},t), \text{ where } \boldsymbol{r} \in \Gamma \text{ and } U(\boldsymbol{r},t) \text{ is known;} \\ 0, \text{ otherwise.} \end{cases} \tag{10}$$

An approximate solution to the inverse problem can be obtained via an iterative gradient descent method. The gradient of the residual functional is computed, and the current approximation of coefficients $c(\boldsymbol{r})$ and $a(\boldsymbol{r})$ is updated: $\{c^{(n+1)}, a^{(n+1)}\} = \{c^{(n)}, a^{(n)}\} - \alpha \cdot \{\Phi'_c(u), \Phi'_a(u)\}$, where n is the iteration number. The process stops when the residual value reaches the level determined by measurement errors and numerical simulation errors and thus does not decrease any further. Each iteration involves solving direct (1)–(2) and conjugate (8)–(10) problems, which require simulating the wave propagation process in forward and reverse time. This technique is also known as propagation-backpropagation (PBP).

The gradient descent method involves computing successive approximations of unknown coefficients over many iterations. Under certain conditions, the gradient descent process converges to the global minimum of the residual functional, which represents an approximate solution to the inverse problem.

2 Overview of the software

Open-source "WaveTomography" software can be used in research and educational projects on wave tomography, computational diagnostics, numerical simulation and supercomputer technology. The software implements the algorithms for direct and inverse problems of wave tomography for Intel x86-64, ARM and GPU computing platforms and runs under Linux-compatible operating systems.

The software is implemented in C++ using open-source "vectorclass" library for Intel-compatible processors and OpenCL interface for GPU computing. The process is designed to run on a single CPU socket. Apply `taskset` or an analogous command as necessary to select a CPU socket on a particular system. The computations are parallelized over multiple computing cores using OpenMP library. The software is free, distributed under GNU General Public License v.3.

The software performs the following functions:

- Computing the simulated wavefield at the detectors using a pre-defined numerical phantom that specifies the parameters $c(\boldsymbol{r})$ and $a(\boldsymbol{r})$ in a single imaging plane.
- Computing an approximate solution to the inverse problem of wave tomography using the previously computed wavefield. The software performs a limited number of iterations of the gradient descent method of minimizing the residual functional and stops when the minimum is found or the limit on the number of iterations is reached.
- Generating parametrized test phantoms. The automatic routines ensure that the generated model problems are solvable in a single run. The auto-generated phantoms correspond to soft tissues in application to medical imaging. The phantom consists of a circular mass approximately 8 cm in diameter with a sound speed lower than that in water (1.5 km/s). The phantom contains randomly placed inclusions of varying sound speed and absorption factor. Two small inclusions 2 mm in diameter in the center of the image serve as a resolution target.

  The phantoms can be accessed by number, which serves as a seed to the random number generator. The phantoms with the same number are the same on all systems.
- During the process, the program reports iteration number and total time spent. The process can be stopped at the next iteration by creating a file named 'stop' in the current directory.

3 Implemented algorithms

"WaveTomography" software implements the algorithms for solving direct and inverse problems of wave tomography on model problems. Figure 2a illustrates the direct solution algorithm. The direct problem solution algorithm takes an image as input, simulates wave propagation through the object and produces a dataset representing the signals recorded by the detectors. Initial pulses, emitter and detector placement are specified by model parameters. By default the software provides randomly generated sample images and reasonable simulation parameters so that these images can be reconstructed.

Figure 2b illustrates the direct solution algorithm. The inverse problem solution algorithm is an iterative gradient descent method of minimizing the residual functional

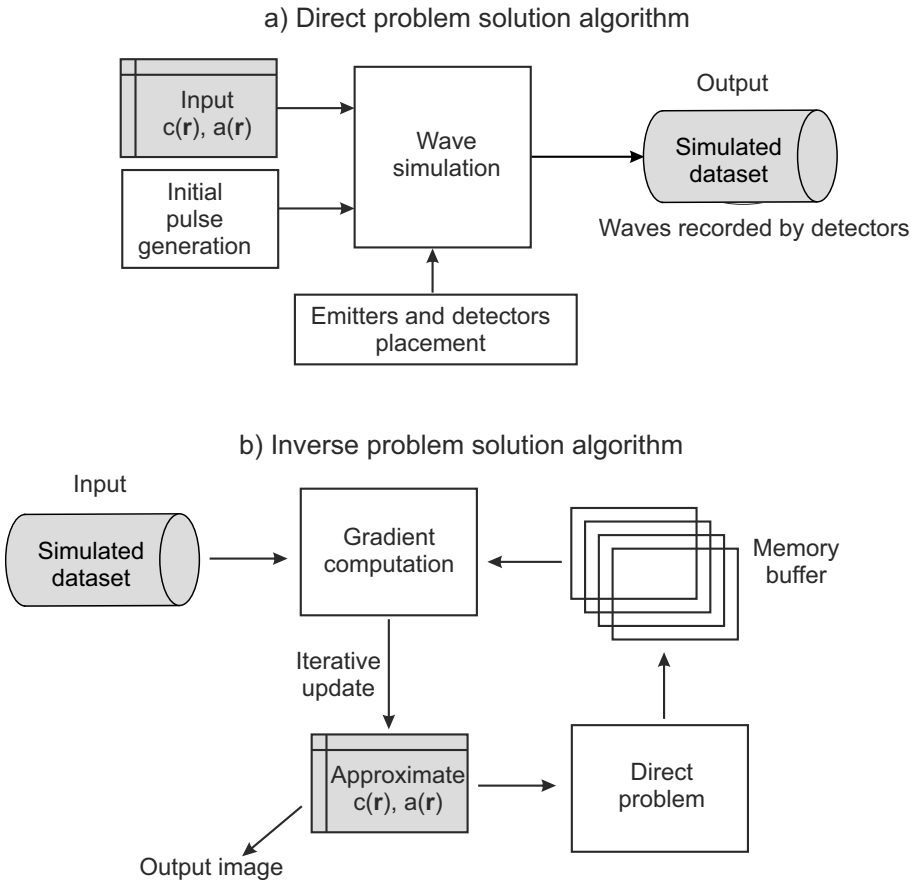


Figure 2: Flowchart of the direct (a) and inverse (b) problem solution algorithms

between the simulated wave and the input data. The algorithm takes the previously computed dataset as input and attempts to reconstruct the image of the object using the waveform data. This is accomplished via assuming a constant initial approximation of the coefficients $c(\boldsymbol{r})$ and $a(\boldsymbol{r})$ and iteratively improving this approximation by solving the direct problem and computing the gradient of the residual functional. The memory buffer holds the boundary values of the wave field. It reverses the wave propagation direction in order to apply formula (7) for gradient computation. Under some conditions that are met by default simulation parameters, the iterative process converges to the original image.

The direct problem solution algorithm implemented in the software consists of the following steps:

- Generating a random numerical phantom for the simulation
- Generating emitters and detectors and setting the simulation parameters — emitters and detectors are placed in a circular formation around the computational domain (Fig. 1a).
- Generating the sounding pulse — computing the initial conditions for the wave equation corresponding to a pulse of a certain wavelength and bandwidth.
- The process of pressure wave propagation through the phantom is simulated
- The wave data at the detectors are saved to the dataset.

  The inverse problem solution algorithm implemented in the software consists of the following steps:

- The input data are read from the dataset computed via solving the direct problem
- Generating emitters and detectors and setting the simulation parameters
- A number of gradient descent iterations is performed. At each iteration, the initial pulse is generated, the wave propagation through the phantom is simulated as in the direct problem, then the gradient of the residual functional is computed and the coefficients of the wave equation are updated according to the gradient.
- The approximate solution obtained as a result of the gradient descent method is saved.

4 Quick start guide

4.1 Compiling

The software was tested with GCC compiler 8.4.1, 10.2.0 and 12.0.0. For compiling the software, makefiles for GNU make utility are included:

- `makefile_arm` for ARM 64-bit systems
- `makefile_gpu` for Intel Haswell-EP with GPU support.
  For GPU support, the path to OpenCL library must be specified.
  The default path is set to `/opt/cuda/cuda-11.1/lib64`
- `makefile_avx512`: for Intel AVX512-capable processors
- `make_auto.sh`: Builds a version suitable for the local system. CPU type is determined via `lscpu`. MPI and OpenCL support is enabled if detected on the system. Individual build options are listed in the makefile.

`wavetm` executable and the files needed for operation are placed in `bin/` directory. MPI version executable is named `wavetm_mpi`.

4.2 Setting up computing devices

For ARM processors, there are no special settings. ARM CPU should run in little-endian mode for correct operation of the software.

GPU can be selected via option `-GPU` *`device_id`* (default: 0) on the command line or in `wavetm.ini` file. If GPU support is compiled in, by default the program attempts to use the GPU at OpenCL Platform 0, Device 0 (equivalent to `-GPU 0`). To use another device, specify `-GPU 0x`*`PPDD`*, where *PP* is the hexadecimal platform number, *DD* is the hexadecimal device number. Example `-GPU 0x0201` uses OpenCL platform #2, device #1 To disable GPU and use CPU only, specify `-GPU -1` For GPU operation, the files `gpu.h` and `k2d.cl` must be present in the current directory.

4.2.1 Multiprocessor operation

On a system with multiple devices, one MPI process should be run for each CPU socket or GPU device in the system. For a multi-GPU system, `-NGPUs n` option specifies the number of GPU devices per computing node. Device IDs for these GPUs are incremented sequentially starting from the ID specified by `-GPU` option. The number of MPI processes launched per node should be equal to the number of GPUs per node for correct operation.

For multi-CPU operation, one process per socket should be launched, and OpenMP should be configured for one thread per processing element. A typical command line for this case is:

```
OMP_PROC_BIND=true OMP_PLACES=threads
mpirun --map-by ppr:1:socket -bind-to socket ./wavetm_mpi $@
```

The total number of processes should not exceed the number of emitters in the simulation. Over-subscribed configuration is not supported.

4.3 Running the simulations

4.3.1 Interactive demo mode

By default, if no options are specified, the program will ask for parameters in interactive mode.

```
*** Interactive demo ***
Image size (480/640/800) ?  640
-size 640 selected
Number of emitters (10 - 20) ?  10
10 selected
Phantom number (any) ?  55555
#55555 selected
Saving auto-phantom 'auto_#55555'
Generating dataset '#55555'
Loading dataset '#55555'
...
```

User input is underlined. Interactive input may be unavailable on some remote-operated computers. In this case, specify the parameters on the command line or in `wavetm.ini` file. The interactive demo can be disabled by setting `-DemoMode 0` option. Demo mode restricts the simulation parameters to safe values and limits the image resolution so that the computations wouldn't take a very long time.

4.3.2 Quick start example

Quick example: `wavetm -direct '#8' -inverse '#8'` — run direct and then inverse problem solution with auto-generated phantom #8 and all default parameters.

Set image size: `-size N` (default: 640) Valid range: 480, 640, 800

Set number of emitters: `-nsources N` (default:15) Valid range: 10 to 20

Examples:

```
wavetm -size 480 -nsources 10 -direct '#1234' -inverse '#1234'
wavetm -size 800 -nsources 20 -direct '#567' -inverse '#567'
```

If interactive demo option is compiled in, run `wavetm` without parameters for the program to ask for the parameters interactively. All command-line parameters can also be specified in `wavetm.ini` file. The parameters on the command line take precedence (overwrite) the parameters specified in .ini file.

4.3.3 Solving the direct problem of wave tomography

Option: `-direct` *<phantom_name>*
`'#N'` as a name uses internally generated random phantoms, where N is a seed number.
Example: `wavetm -direct '#8'`
Output: Dataset files are created (`'#8.exd.NNN'`, etc.) The corresponding dataset file names all start with `#8` in this case. The auto-generated phantom is saved as `auto__8.mat` (the # symbol is not allowed in MATLAB matrix names. Such symbols

are replaced with '_', or with 'x' if it is the first symbol). The auto-generated phantom can be viewed in MATLAB or GNU Octave using the script `view_i.m`. Example: run `view_i('auto__8');` from MATLAB or GNU Octave. The scripts are supplied in `scripts/`directory.

4.3.4 Solving the inverse problem of wave tomography

Option: `-inverse` *<dataset_name>*
Specify the same name as in `-direct` option to use the dataset created earlier.

Example: `wavetm -inverse '#8'`
No additional options are necessary. The image size and emitters settings are taken from the dataset.

Output: Every iteration, the program prints iteration number, gradient descent rates and elapsed time. By default, every 10 iterations the approximate solution reached to this point is saved in MAT format as *<name>_iNNN.**, where *NNN* is the iteration number. The solution gradually improves with more iterations. At the end of the process, the solution reached is saved as `finished_`*<name>*`.mat` By default, the number of iterations is limited to 250. The process can be stopped at the next iteration by creating a file named `stop` in the current directory. The approximate solution obtained can be viewed in MATLAB or GPU Octave using the script `view_i.m`. Example: run `view_i('finished__8')` from MATLAB or GNU Octave.

Both `-direct` and `-inverse` options can be sulpplied at the same time. In this case, `-direct` is executed first, regardless of the order of appearance.

5 Command line options

5.1 Task control

`-direct` *name*
Solve direct problem and create the simulated dataset. *name* may be an input file name or an internal phantom number if starts with #

`-inverse` *name*
Solve inverse problem for the simulated dataset with a corresponding name. Both `-direct` and `-inverse` may be specified, `-direct` is executed first.

`-restart` *file*
Supply the data file for an initial approximation for the inverse problem (raw data) or a RST file created by the program (includes metadata to continue an interrupted process). User-data file is an array of $[2 \times ImageSize \times ImageSize]$ 32-bit float items. First element in a pair for each pixel represents the speed of sound in km/s, second element represents the attenuation factor

`-GPU` *0xPPDD*
Specify OpenCL device number in hex. PP=platform number, DD=device. To disable GPU, specify `-GPU -1`. Default=0 (first available platform and device).

`-NGPUs` *n*
Specify the number of devices per computing node.

`-DemoMode` *0/1*
Ask for parameters interactively if not supplied and restrict the parameters to safe limits for a quick start. Default=1 (set by a build option). Interactive selection might not work on remote-operated systems or under MPI.

5.2 Simulation parameters

`-size` *pixels*
Image size

`-nsources` *n*
Number of ultrasound emitters

`-SrcRadius` *mm*
Emitter placement radius

`-RcvRadius` *mm*
Detector placement radius

`-RcvSpacing` *pixels*
Detector spacing

`-SolutionRange` *Rmin, Rmax*
The solution is confined to the area $R < Rmin$ (millimeters). The area $R > Rmax$ is assumed empty. A smooth transition is applied between $Rmin$ and $Rmax$. Negative values are relative to the field size. For example -0.8 is 80% of hte field radius. Defaults= -0.64, -0.8

`-TimeBuffer` k
Specify the simulation time. $k = 1$ corresponds to the time for the wave to reach the detectors. Default=1.5

`-impulse` l, k
Specify initial pulse length (mm) and the number of waves k. The formula for the initial pulse is $f(x) = \sin(\pi\, x/l) \cdot \sin(\pi\, k\, x/l), \quad x = 0...1$. Default: $k = 2$, length determined automatically from image resolution.

`-impstart` *mm*
Distance from the emitter (center) to the initial pulse wavefront.

`-impclip` *wmin, wmax*
Beam width control. The wavefront for $\cos(\alpha) < wmin$ is cut off, where $\alpha = 0$ is the direction from the emitter to the center of the field. The wavefront for $\cos(\alpha) > wmax$ is fully preserved. A smooth transition is applied between $wmin$ and $wmax$. Defaults=0.73, 0.95. If both values are below -1 the initial pulse is a spherical wave.

`-nframes` n
Number of simulation time frames. Overrides the auto-computed value.

5.3 Generated phantoms

Auto-generated phantoms by default are created so that the inverse problem is solvable in a single run of the gradient-descent algorithm using default simulation parameters.

`-PhantomSize` *mm*
Phantom size

`-PhantomContrast` k
Increases contrast by k

5.4 Numerical method parameters

`-v0` *km/s*
Speed of sound in the environment. Default=1.5

`-dx` *mm*
Pixel size. Default pixel size is computed so that the field size is 250 mm.

`-dt` *usec*
Time step in microseconds. Negative values are relative to dx. For example `-dt -0.3` specifies $dt = 0.3 \cdot dx$

`-LAPL` *k01, k11, k02, k12, k22*
Specify discrete Laplacian coefficients for a 5x5 stencil.

`-LimitVel` *Vmin,Vmax*
Limit the wave velocity in the sought-for solution of the inverse problem.

`-LimitAtt` *Amax*
Limit the attenuation factor in the the sought-for solution of the inverse problem.

5.5 Iterative method parameters

`-MaxIter` n
Maximum number of iterations.

`-SaveIter` n
Save output file every n iterations.

`-TargetGrad` *dv*
Specify the starting value of gradient descent step as a sound speed difference in km/s. Default=0.004

`-StepUpCoeff` k
Increase step by k if the residual value decreases. Default=1.25

`-StepDownCoeff` k
Decrease step by k if the residual value increases. Default=0.3

`-MinThreshold` p
Stop the process if the residual decreases by less than p per iteration. Default=0.0003.

`-AttGradScale` k
Modify the gradient for the attenuation factor by k. $k = 0$ disables attenuation in the model. Other values can be used in research to boost or dampen attenuation factor reconstruction. Default=1

5.6 Output control

`-dump` *nframes*
Outputs the wave data every *nframes* time frames.

`-WaveExportFormat` *PMB,MAT,RAW*
A parameter string specifies the file format(s) for wave images created by `-dump` option. Default=*MAT*

`-ImageExportFormat` *PMB,MAT,RAW,RST*
A parameter string specifies the file format(s) for output images. .RST files contain the reconstructed image and iteration parameters. The iterative process can be continued from a .RST file using `-restart` option Default=*MAT*

`-PBMScale` *Vrange, Arange*
Sound speed and attenuation factor ranges represented in .PBM images. Defaults=0.2, 0.015 Red channel represents sound speeds greater than $v0$, blue channel represents sound speeds less than $v0$, green channel represents the attenuation factor.

`-log` n
n=0 (default) – minimal, n=1 – verbose logging, other values are bit masks that enable various debug logging features. Refer to the source for details.

5.7 Computation control

`-nthreads` n
Specify the number of OpenMP threads per process.

`-CellW` n
GPU block width in 4-element vectors. Default=32

`-CellY` n
Block height in pixels. Default=20

`-vector` n
CPU SIMD vector size in 32-bit words. Possible values – 4,8,16. Not available on ARM CPUs. Default=16 (for Intel AVX-512 CPUs). The specified vector size may be different

from the native CPU vector. Various SIMD modes performance can be tested with this option.

`-CPUMemory` *MBytes*
Specify the cache size. Can be determined automatically by a performance test.

`-memalign` *bytes*
Specify the memory alignment factor. Values unsupported by the machine may result in a bus error or a segmentation fault.

`-ialign` *align shear*
Specify the image width alignment and shift factors in pixels.

5.8 Performance testing

The tests are run only for a single CPU.

`-perftest` *type, param*

type = 1 - determine CPU cache size. *param*=image size for testing.
Example `-perftest 1 480`
Upon completion, this test writes `-CPUMemory` option to wavetm.ini file.
type = 0 execute performance tests:
param = 1 create optimized settings (autotune.dat) – runs multiple tests to determine optimal settings
param = 2 apply optimized settings

`-TestIncrSize` *Increment, MaxSize*
The tests are executed for image size increasing from a size given by `-size` by *Increment* up to and including *MaxSize* (pixels).
Example: `-perftest 0 1 -size 640 -IncrSize 320 1280`
-runs the test for image sizes 640, 960 and 1280.

`-TestIncrSrc` *n*
The tests are executed for the number of emitters decreasing from given by `-nsources` by *n*. Example: `-perftest 0 1 -size 640 -nsources 12 -TestIncrSrc 3`
-runs the test for 12, 9, 6 and 3 emitters

`-TestVolume` *pixels*
Specify the test volume, total pixels computes for all simulation frames. Reasonable values are about $1e10...2e10$ (10–20 gigapixels) for the tests to take a few seconds each.

`-TestDuration` *sec*
Specify the duration of each test in seconds. Actual duration can be longer for large image sizes.

`-SkipScalars` *0/1*
Skip scalar functions in the test (test wave simulations only)

`-autotune` *0/1*
Uses optimized settings in inverse problem. Create optimized settings if they do not exist, use if they exist. The settings are separate for each image size and for CPU/GPU. New settings can be created only in single-process mode.

6 Image reconstruction examples

Wave tomography image reconstruction fundamentally differs from X-ray tomographic imaging. While in X-ray imaging the detectors record a single value, in wave tomography the data recorded by detectors is a continuous waveform for some time period. The inverse problem of wave tomography is nonlinear and ill-posed. The reconstructed images represent two variables — sound speed and sound absorption factor inside the object. The images demonstrate that it is possible to reconstruct such images from waveform data with high resolution and high sensitivity to changes in sound speed and absorption factor.

WaveToography software is aimed to aid the research in medical imaging for breast cancer diagnosis, and the parameters of the simulations are set accordingly. In medical imaging, soft tissues have low contrast, but the imaging method must detect small inclusions. Auto-generated sample phantoms contain inclusions with sizes of 2 mm and larger. Inclusions 2 mm in size are resolved, which is a typical target resolution for breast cancer diagnosis.

Computational complexity of the inverse problem of wave tomography is proportional to the image size, the number of time frames, the number of emitters, and the number of gradient descent iterations. Computation time per iteration rises at least as a third power of the image length (not counting the number of emitters), which means approximately 10-fold increase per each doubling of the resolution.

The number of iterations required to solve the inverse problem (which means to obtain an approximate solution that cannot be improved any further under the current simulation setup) is not known a priori and may vary arbitrarily from tens to hundreds depending on the data, simulation setup and reconstruction precision.

The dataset generated by solving a direct problem consists of the waves recorded by the detectors for the duration of the measurements (in this case, the numerical simulation). The data can be examined in MATLAB or GNU octave using the `viewdata.m` script supplied.

Figure 3 shows a sample data file viewed. The image shows a wave scattered off a simulated phantom. The phantom itself is unknown at this point, these data serve as input to the inverse problem of tomographic image reconstruction. The detector number (horizontal) is proportional to angular direction from the center to a detector and the vertical axis is simulation time. Obtain this image: `viewdata('#8.exd.000');` from

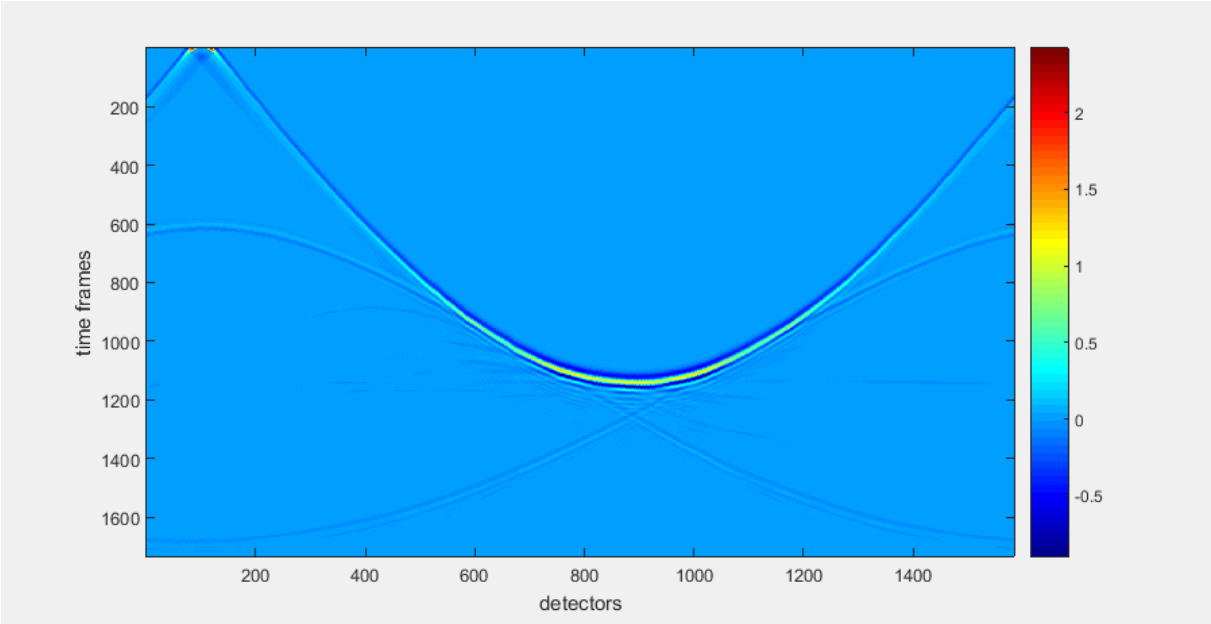


Figure 3: Input data to the inverse problem

within MATLAB or GNU Octave after running the direct problem: `wavetm -nsources 20 -size 800 -direct '#8'`

The size of the input data for each ultrasound emitter is roughly proportional to the size of the reconstructed image. In addition, the higher the resolution, the more emitters are typically needed, making the dataset to grow faster than the image size. Actual spatial resolution obtained in wave tomography depends mainly on the wavelength (and measurement precision in physical experiments).

The image size is chosen so that all the details revealed by a given wavelength fit in the image. The presented examples have the following parameters:

Dataset	Image size	Emitters	Wavelength	Time frames	Files
Small:	480x480	10	9 mm	779	35 MB
Medium:	640x640	16	7 mm	1040	91 MB
Large:	800x800	20	5.6mm	1300	166 MB

Typical computing time is 2 to 10 seconds per iteration on a Kunpeng-920 CPU (single socket). The maximum computing time is 45 minutes (the iterative process is limited to 250 iterations by default).

The simulated phantom used in the reconstructions was generated automatically by the software. It has a diameter of 72 mm and random inclusions of varying sound speed and absorption factor. Two dots in the center of the image are 2 mm in size, which is much smaller than a wavelength of 9 mm. A remaining effect of the waves can be seen at the perimeter of the phantom.

The following images were obtained on different systems — NVidia Tesla V100 GPU, Intel Cascade Lake 6240R CPU and ARM-based Kunpeng-920 CPU.

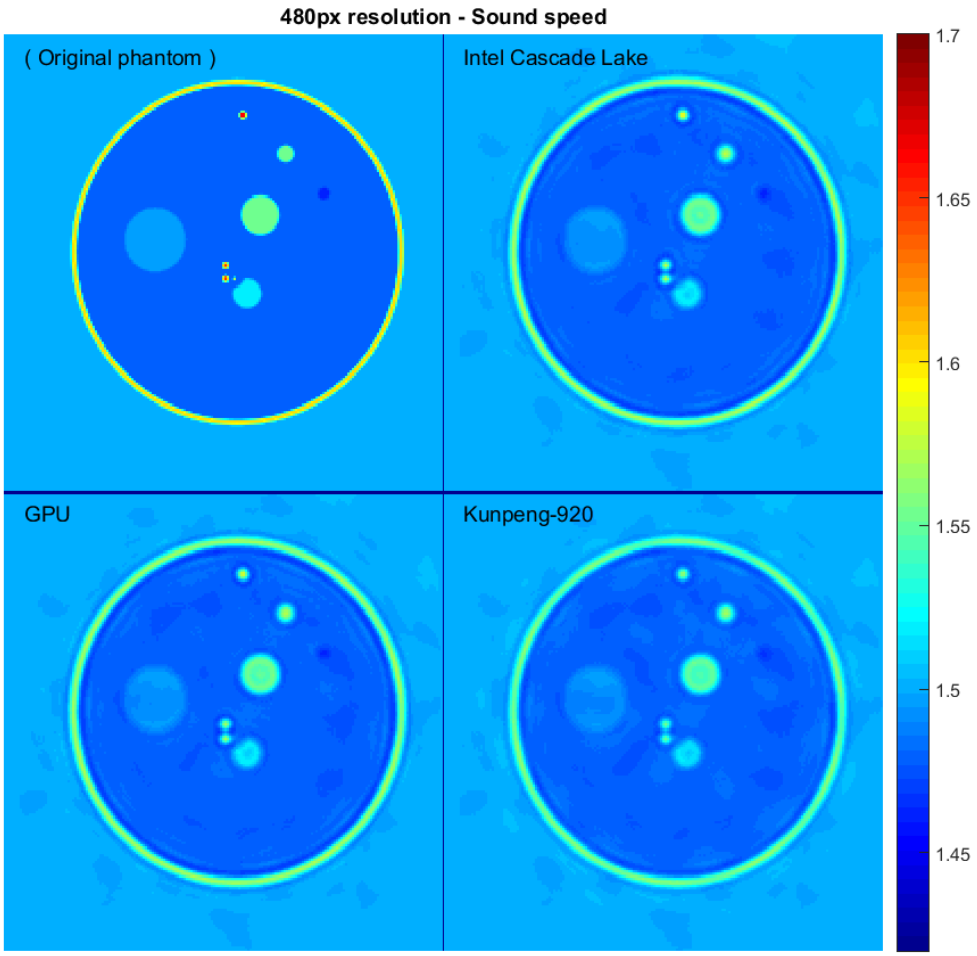


Figure 4: Sound speed reconstruction for a small dataset

To obtain these images, run:

```
wavetm -size 480 -nsources 10 -direct '#8' -inverse '#8'
```

After the process finishes, view the data in Matlab or GNU Octave:

View original phantom: `view_i ('auto__8');`

View reconstructed image: `view_i ('finished__8');`

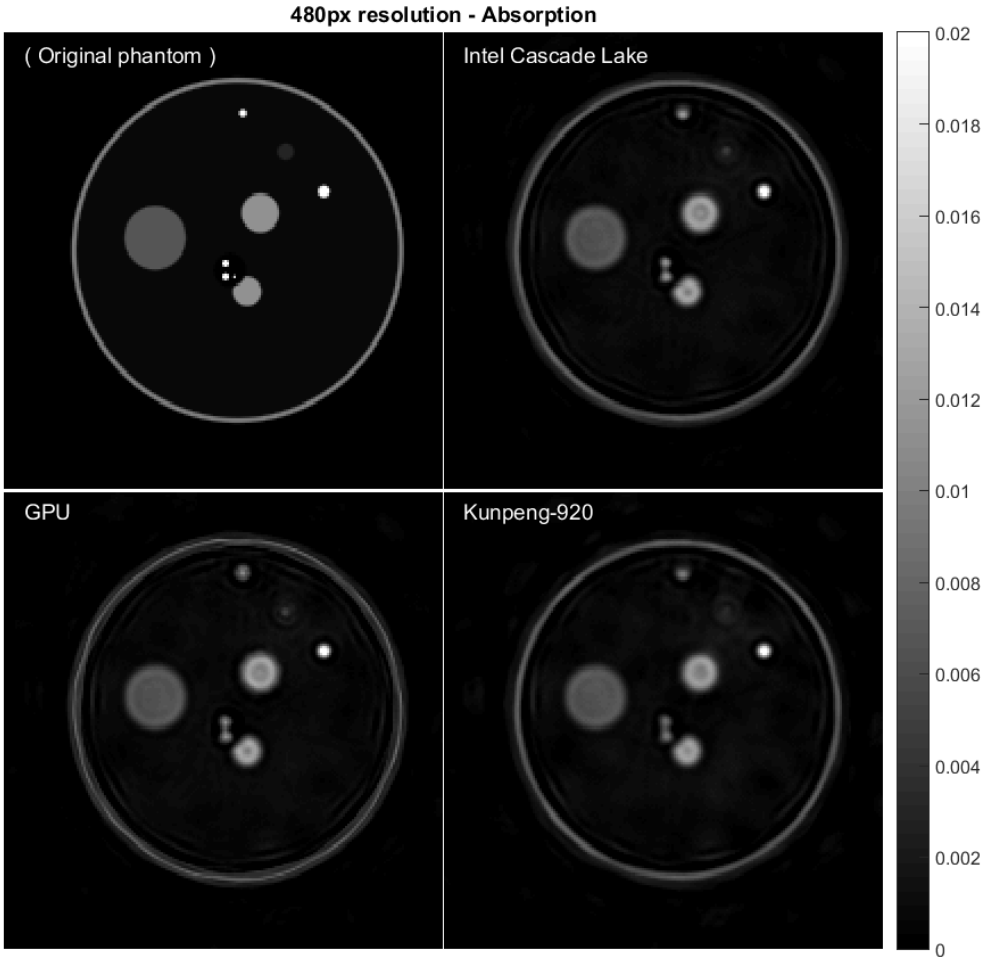


Figure 5: Absorption factor reconstruction for the small dataset

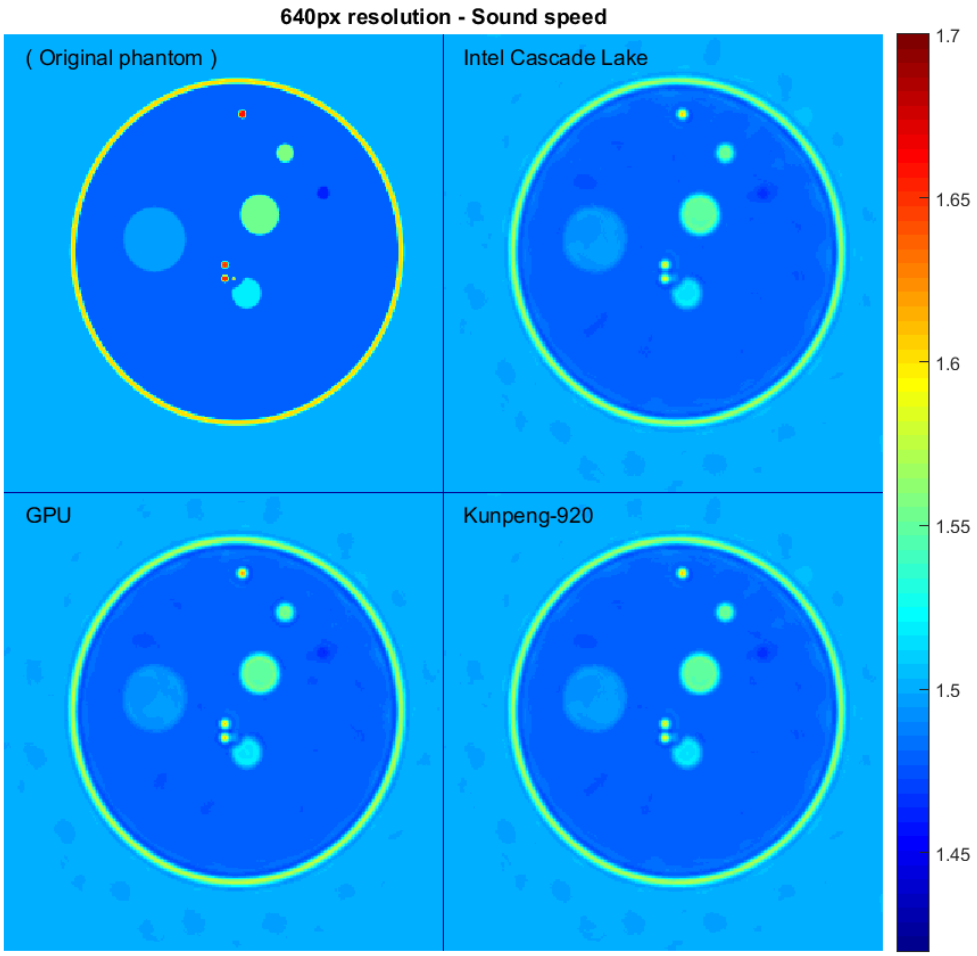


Figure 6: Sound speed reconstruction for the medium dataset

To obtain these images, run:

```
wavetm -size 640 -nsources 16 -direct '#8' -inverse '#8'
```

After the process finishes, view the data in Matlab or GNU Octave:

View original phantom: `view_i ('auto__8');`

View reconstructed image: `view_i ('finished__8');`

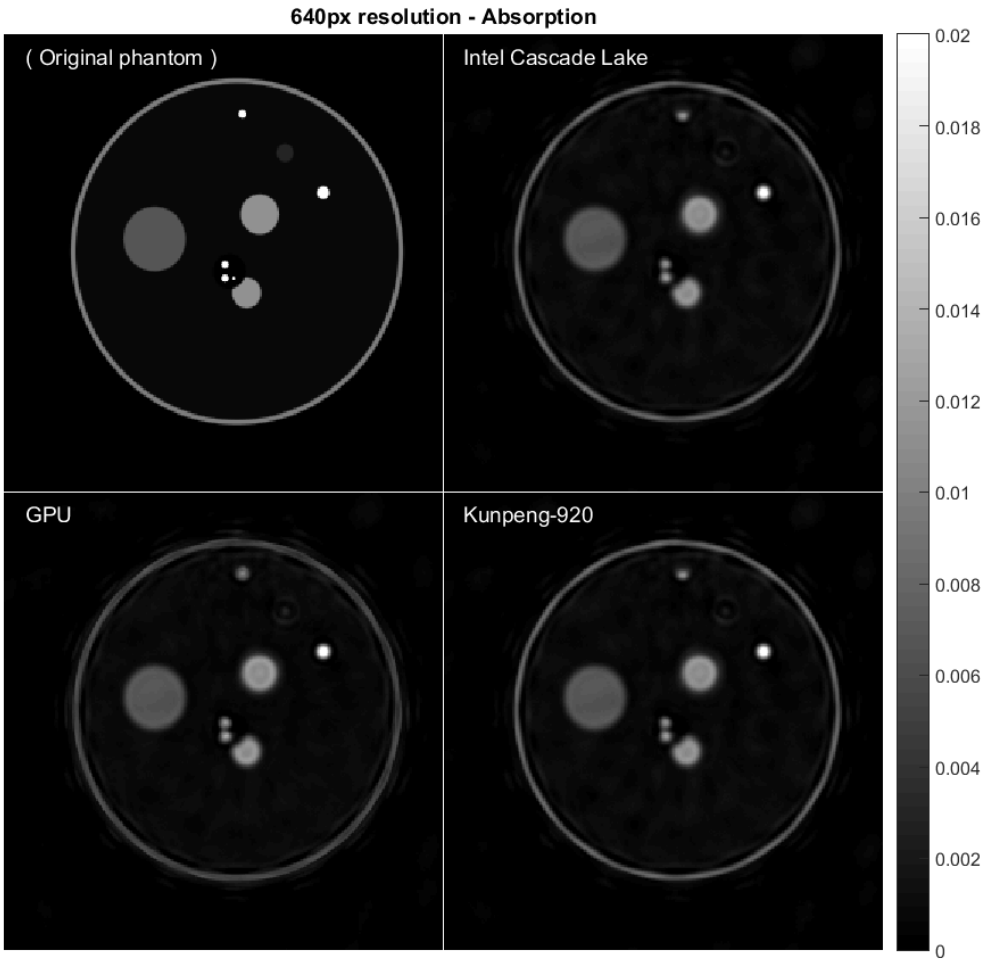


Figure 7: Absorption factor reconstruction for the medium dataset

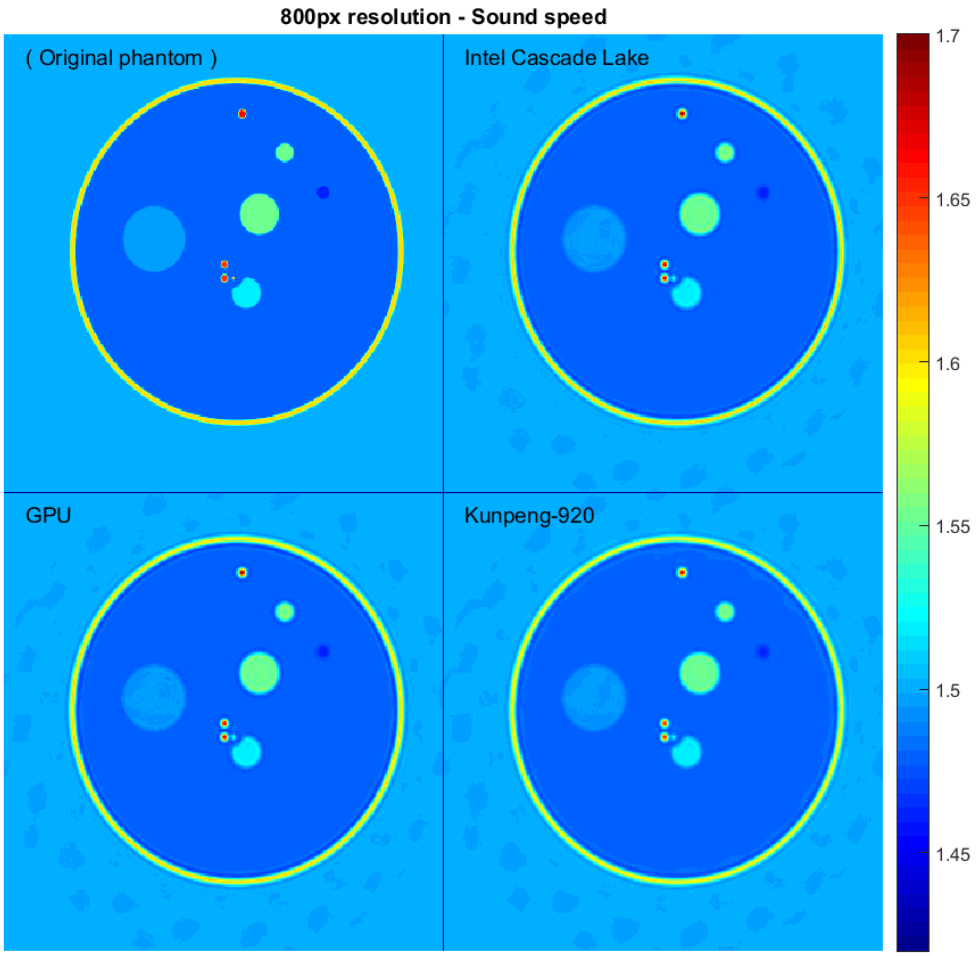


Figure 8: Sound speed reconstruction for the large dataset

To obtain these images, run:

```
wavetm -size 800 -nsources 20 -direct '#8' -inverse '#8'
```

After the process finishes, view the data in Matlab or GNU Octave:

View original phantom: `view_i ('auto__8');`

View reconstructed image: `view_i ('finished__8');`

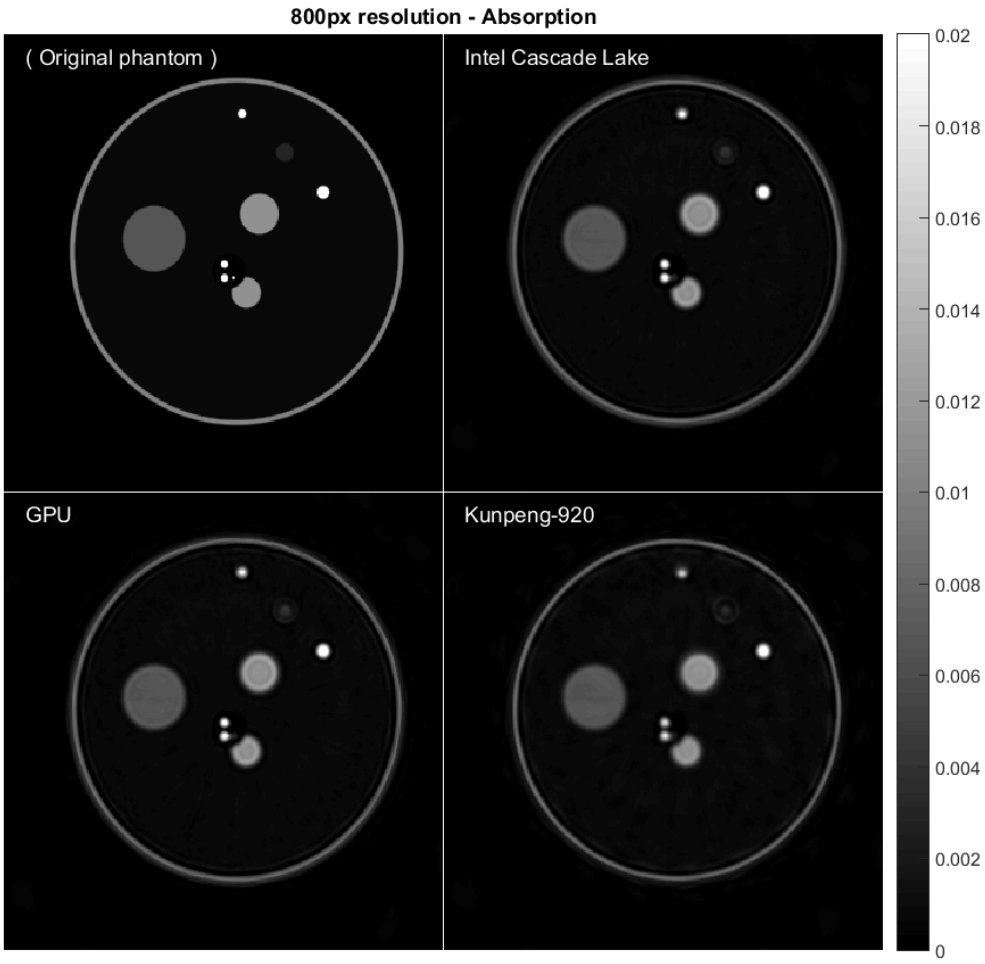


Figure 9: Absorption factor reconstruction for the large dataset