Reduction of computing time for least-squares migration based on the Helmholtz equation by graphics processing units

Knibbe, Hans; Vuik, Kees; Oosterlee, Kees

DOI
10.1007/s10596-015-9546-z

Publication date
2015

Document Version
Final published version

Published in
Computational Geosciences: modeling, simulation and data analysis

Comput Geosci (2016) 20:297–315
DOI 10.1007/s10596-015-9546-z

ORIGINAL PAPER

Reduction of computing time for least-squares migration based on the Helmholtz equation by graphics processing units

H. Knibbe[1] · C. Vuik[1] · C. W. Oosterlee[1,2]

Received: 17 May 2015 / Accepted: 25 October 2015 / Published online: 30 December 2015

Abstract In geophysical applications, the interest in least-squares migration (LSM) as an imaging algorithm is increasing due to the demand for more accurate solutions and the development of high-performance computing. The computational engine of LSM in this work is the numerical solution of the 3D Helmholtz equation in the frequency domain. The Helmholtz solver is Bi-CGSTAB preconditioned with the shifted Laplace matrix-dependent multigrid method. In this paper, an efficient LSM algorithm is presented using several enhancements. First of all, a frequency decimation approach is introduced that makes use of redundant information present in the data. It leads to a speedup of LSM, whereas the impact on accuracy is kept minimal. Secondly, a new matrix storage format Very Compressed Row Storage (VCRS) is presented. It not only reduces the size of the stored matrix by a certain factor but also increases the efficiency of the matrix-vector computations. The effects of lossless and lossy compression with a proper choice of the compression parameters are positive. Thirdly, we accelerate the LSM engine by graphics cards (GPUs). A GPU is used as an accelerator, where the data is partially transferred to a GPU to execute a set of operations or as a replacement, where the complete data is stored in the GPU memory. We demonstrate that using the GPU as a replacement leads to higher speedups and allows us to solve larger problem sizes. Summarizing the effects of each improvement, the resulting speedup can be at least an order of magnitude compared to the original LSM method.

Keywords Least-squares migration · Helmholtz equation · Wave equation · Frequency domain · Multigrid method · GPU acceleration · Matrix storage format · Frequency decimation

Mathematics Subject Classifications (2010) 65-04 · 65N55 · 86A15 · 65Y05

1 Introduction

In the oil and gas industry, one of the challenges is to obtain an accurate image of the subsurface to find hydrocarbons. A source, for instance an explosion, sends acoustic or elastic waves into the ground. Part of the waves is transmitted through the subsurface, another part of the waves is reflected at the interfaces between layers with different properties. Then the wave amplitude is recorded at the receiver locations, for example, by geophones. The recorded signal in time forms a seismogram. The data in frequency domain can be easily obtained by the Fourier transform of the signal in time. Using the recorded data, there are several techniques, called depth migration, to map it to the depth domain, given a sufficiently accurate velocity model. The result is a reflectivity image of the subsurface. The techniques include ray based and wave equation based algorithms and can be formulated in time or in frequency domain.

An alternative to the depth migration is least-squares migration (LSM). Least-squares migration [22] has been shown to have the following advantages: (1) it can reduce

✉ H. Knibbe
hknibbe@gmail.com

[1] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

[2] Centrum Wiskunde & Informatica, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

migration artifacts from a limited recording aperture and/or coarse source and receiver sampling; (2) it can balance the amplitudes of the reflectors; and (3) it can improve the resolution of the migration images. However, least-squares migration is usually considered expensive, because it contains many modeling and migration steps.

Originally, ray-based Kirchhoff operators have been proposed for the modeling and migration in LSM (see, e.g., Schuster [27], Nemeth et al. [22]). Recently, in least-squares migration algorithms, wave-equation based operators were used in the time domain (see, e.g., Tang [29], Wei and Schuster [32]) and in the frequency domain (see, e.g., Plessix and Mulder [23], Kim et al. [14], Ren et al. [24]). The major advantage of a frequency domain engine is that each frequency can be processed independently in parallel.

With the recent developments in high-performance computing, such as increased memory and processor power of central processing units (CPUs) and the introduction of general purpose graphic processing units (GPGPUs), it is possible to compute larger and more complex problems and use more sophisticated numerical techniques. For example, in migration, the wave equation has been traditionally solved by an explicit time discretization scheme in the time domain requiring large amounts of disk space. In Knibbe et al. [17], we have shown that solving the wave equation in the frequency domain, i.e., the Helmholtz equation, can compete with a time domain solver given a sufficient number of parallel computational nodes with a limited usage of disk space. The Helmholtz equation is solved using iterative methods. Many authors showed the suitability of preconditioned Krylov subspace methods to solve the Helmholtz equation, see, for example, Gozani et al. [10], Kechroud et al. [12]. Especially, the shifted Laplace preconditioners improve the convergence of the Krylov subspace methods, see Laird and Giles [19], Turkel [31], Erlangga et al. [6], Erlangga et al. [7].

These methods have shown their applicability on traditional hardware such as a multi-core CPUs, see, e.g., Riyanti et al. [25]. However, the most common type of cluster hardware consists nowadays of a multi-core CPU connected to one or two GPUs. In general, a GPU has a relatively small memory compared to the CPU.

A GPU can be used as a *replacement* for the CPU or as an *accelerator*. In the first case, the data lives in GPU memory to avoid memory transfers between CPU and GPU memory. We have already investigated this approach for the Helmholtz equation in the frequency domain in Knibbe et al. [15, 16]. The advantage of the migration with a frequency domain solver is that it does not require large amounts of disk space to store the snapshots. However, a disadvantage is the memory usage of the solver. As GPUs have generally much less memory available than CPUs, this impacts the size of the problem significantly.

In the second case, the GPU is considered as an accelerator, which means that the problem is solved on the CPU while off-loading some computational intensive parts of the algorithm to the GPU. Here, the data is transferred to and from the GPU for each new task. This approach has been investigated for the wave equation in the time domain in Knibbe et al. [17]. While the simplicity of the time domain algorithm makes it easy to use GPUs of modest size to accelerate the computations, it is not trivial to use GPUs as accelerators for the Helmholtz solver. By using the GPU as an accelerator, the Helmholtz matrices are distributed across two GPUs. The vectors would "live" on the CPU and are transferred when needed to the relevant GPU to execute matrix-vector multiplication or Gauss-Seidel iterations. As a parallel Gauss-Seidel iteration is generally more expensive than a matrix-vector multiplication, it would still pay off to transfer the memory content back-and-forth between GPU and CPU. For a frequency domain solver, the off-loaded matrix-vector multiplication in the well-known Compressed Sparse Row (CSR) format does not result in any significant improvements compared to a many-core CPU due to the data transfer.

The goal of this paper is to accelerate the least-squares migration algorithm in frequency domain using three different techniques. Firstly, a decimation algorithm is introduced using the redundancy of the data for different frequencies. Secondly, we introduce a VCRS format and consider its effect on the accuracy and performance of the Helmholtz solver, which is our numerical engine for each source and frequency of the LSM algorithm. The third goal is to achieve an improved performance of LSM by using GPUs either as accelerators or as replacements for CPUs.

2 Least-squares migration

2.1 Description

The solution for a wave problem in a heterogeneous medium is given by the Helmholtz wave equation in three dimensions

$$A\phi = g, \qquad A = -k^2\sigma^2 - \Delta_h \tag{1}$$

where $\phi = \phi(x, y, z)$ is the pressure wavefield, $g = g(x, y, z)$ is a source function, and L is the Helmholtz operator with the spatially-dependent frequency $k = k(x, y, z)$ and the *slowness* $\sigma = 1/c^2$ which is the inverse of the square velocity $c = c(x, y, z)$. Here, Δ denotes the discrete spatial Laplace operator. The problem is defined in a rectangular domain $\Omega = [(0, 0, 0), (X, Y, Z)]$, $X, Y, Z \in \mathbb{R}$. A first-order radiation boundary condition is applied

$\left(-\frac{\partial}{\partial \eta} - ik\right)\phi = 0$, where η is the outward normal vector to the boundary (see Engquist and Majda [5]).

The slowness σ can be split into $\sigma = \sigma_0 + r\sigma_0$, where the perturbation of slowness r denotes reflectivity and σ_0 ideally does not produce reflections in the bandwidth of the seismic data. Then, the Helmholtz operator in Eq. 1 can be written as

$$A = -k^2\sigma_0^2 - 2k^2 r\sigma_0^2 - k^2 r^2\sigma_0^2 - \Delta_h. \tag{2}$$

Assuming reflectivity being very small $r << 1$ gives

$$A = -k^2\sigma_0^2 - 2k^2 r\sigma_0^2 - \Delta_h. \tag{3}$$

The wavefield $\phi = \phi_0 + \phi_1$ can be split accordingly into a reference and a scattering wavefield, respectively. The reference wavefield ϕ_0 describes the propagation of a wave in a smooth medium without any hard interfaces. The scattering wavefield ϕ_1 represents a wavefield in a medium which is the difference between the actual and the reference medium. Substituting the split to Eq. 1 gives

$$\left(-k^2\sigma_0^2 - 2k^2 r\sigma_0^2 - \Delta_h\right)(\phi_0 + \phi_1) = f. \tag{4}$$

Wave propagation in the reference medium is described by $A_0\phi_0 = g$ with $A_0 = -k^2\sigma_0^2 - \Delta_h$. Then, the Helmholtz equation can be written as

$$A_0\phi_0 - 2k^2 r\sigma_0^2\phi_0 + A_0\phi_1 - 2k^2 r\sigma_0^2\phi_1 = g. \tag{5}$$

In the Born approximation, the term $2k^2 r\sigma_0^2\phi_1$ is assumed to be negligible, leading to the system of equations

$$\begin{cases} A_0\phi_0 = g, \\ A_0\phi_1 = 2\omega^2 r\sigma_0^2\phi_0, \end{cases} \tag{6}$$

which represents the forward modeling. Let us denote $\hat{\phi}(\omega, x_s, x_r)$ the solution of the wave Eq. 6 from the source g at the position x_s and recorded at the receiver positions x_r

$$\hat{\phi}(\omega, x_s, x_r) = R(x_r)(\phi_0(\omega, x_s) + \phi_1(\omega, x_s, r)). \tag{7}$$

Here, R can be seen as a projection operator to the receiver positions. Then, migration becomes the linear inverse problem of finding the reflectivity r that minimizes the difference between the recorded data $d(\omega, x_s, x_r)$ and the modeled wavefield $\hat{\phi}(\omega, x_s, x_r)$ dependent on the reflectivity r, in a least-squares sense

$$J(r) = \frac{1}{2}\sum_{\omega}\sum_{x_s, x_r} \|d(\omega, x_s, x_r) - \hat{\phi}(\omega, x_s, x_r)(r)\|^2. \tag{8}$$

Removing the first arrival from the recorded data and denoting it by d_1, the previous equation is equivalent to

$$J(r) = \frac{1}{2}\sum_{\omega}\sum_{x_s, x_r} \|d_1(\omega, x_s, x_r) - R(x_r)\phi_1(\omega, x_s, r)\|^2. \tag{9}$$

Equation 9 can be also written in a matrix form, as

$$J(r) = \frac{1}{2}\left(\mathbf{d} - \mathbf{RFr}\right)^H\left(\mathbf{d} - \mathbf{RFr}\right), \tag{10}$$

where $\mathbf{d}$ contains the recorded data without first arrival for each source and receiver pair, $\mathbf{R}$ denotes the projection matrix, $\mathbf{F}$ is the modeling operator from Eq. 6, and $\mathbf{r}$ contains reflectivity. By setting the gradient of the Jacobian in Eq. 10 to zero, we obtain the solution to the least-squares problem in a matrix form,

$$\mathbf{F}^H\mathbf{R}^H\mathbf{R}\,\mathbf{F}\,\mathbf{r} = \mathbf{F}^H\mathbf{R}^H\mathbf{d}. \tag{11}$$

Here, the operator $\mathbf{R}^H$ denotes the adjoint of the projection operator $\mathbf{R}$ and is defined as "extending the data $\mathbf{d}$ given at the receiver positions to the whole computational domain". The right-hand side is the sum over each source of its subsurface image, that is obtained by migration $\mathbf{F}^H$ of the data at the receiver position corresponding to the given source. Note that migration in the frequency domain is described in detail in our previous work Knibbe et al. [17]. The left-hand side consists of a sum over the forward modeling (6) for a given set of reflectivity coefficients for each source, consecutively followed by the migration.

2.2 CG and frequency decimation

Equation 11 represents the normal equation that can be solved iteratively, for example, with a conjugate gradient method (CGNR, see, e.g., Saad [26]), which belongs to the family of Krylov subspace methods. For each iteration of the CGNR method, a number of matrix-vector multiplications and vector operations are performed. Usually, the iteration matrix is constructed once before the start of the iteration. However, to construct the matrix in Eq. 11 is very costly, since it requires the number of sources times the number of frequencies of matrix-matrix multiplications. Therefore, we only compute the vector by matrix-vector operations, where the used parts of the matrix are constructed on the fly. Since we consider the problem in the frequency domain, the iteration matrix consists of Helmholtz matrices for each source and a corresponding set of frequencies:

$$\mathbf{F}^H\mathbf{R}^H\mathbf{R}\,\mathbf{F}\,\mathbf{r} =: \sum_{s}\sum_{\omega}(F_{s,\omega}^H R_{s,\omega}^H R_{s,\omega} F_{s,\omega}\mathbf{r}). \tag{12}$$

Next, we assume that there is a redundancy in the seismic data (both modeled and observed) with respect to the frequencies. This assumption has been suggested for migration in frequency domain in Mulder and Plessix [21].

The idea is to reduce the number of frequencies in such a way that for each source several frequencies are discarded. Therefore, we benefit from the redundancy of seismic data, so that the total amount of computations is reduced. We introduce *decimation* over the frequencies and the sources

by choosing subsets ω' and s', respectively, and decimation parameter δ. The decimation parameter is defined as a factor by which the original set of frequencies and sources is reduced. Note that the decimation factor also indicates a reduction of the computational effort. The subset of frequencies and sources, which has size of ω' times s', is constructed by applying a mask consisting of zeros and ones to the original set of size ω times s. The number of ones is δ times smaller than the total size of the original subset. The positions of the ones are randomly generated with a normal distribution. A similar technique has been used for random shot decimation for the full-waveform inversion in the time domain in two dimensions (see Guitton and Diaz [11]). The subset of size ω' times s' is changing for each iteration of the CGNR method. In this way, the decimation of frequencies and sources is compensated for by the redundancy of the data.

The frequency decimation is only applied to the left-hand side of Eq. 11, the right-hand side that represents input data is not decimated. Therefore, the iteration matrix with the frequency decimation is given by

$$\mathbf{F}^H\mathbf{R}^H\mathbf{R}\,\mathbf{F}\,\mathbf{r} =: \delta \sum_{\omega'\times s'} (F^H_{s',\omega'} R^H_{s',\omega'} R_{s',\omega'} F_{s',\omega'}\mathbf{r}). \tag{13}$$

Here, the decimation parameter is used to compensate for the energy of the sum in case of reduction over frequencies and sources. Let us explain it for a simple example with decimation factor $\delta = 2$. In this case, there are two subsets of equal size: one with decimated sources and frequencies, $\Omega_{\text{decimated}} = \omega' \times s'$, and a second one with the removed sources and frequencies, $\Omega_{\text{removed}} = \omega \times s - \omega' \times s'$. The iteration matrix can then be presented as a sum of the two matrices

$$\begin{aligned}\mathbf{F}^H\mathbf{R}^H\mathbf{R}\,\mathbf{F}\,\mathbf{r} \quad &= A_{\text{decimated}} + A_{\text{removed}} \\ &:= \textstyle\sum_{i\in\Omega_{\text{decimated}}}(F_i^H R_i^H R_i F_i) \\ &\quad + \textstyle\sum_{i\in\Omega_{\text{removed}}}(F_i^H R_i^H R_i F_i).\end{aligned}$$

Here, the randomness of the decimated subset is important because for a given source the randomly selected frequencies are assumed to represent the spectrum of the source. Randomly selected sources are collected without affecting the total signal energy. We can assume that

$$A_{\text{decimated}} \approx A_{\text{removed}}, \tag{14}$$

which leads to

$$\mathbf{F}^H\mathbf{R}^H\mathbf{R}\,\mathbf{F}\,\mathbf{r} = 2A_{\text{decimated}}. \tag{15}$$

Note that the matrices do not have to be assembled. In our matrix notation, each matrix is implemented as an operator.

2.3 Helmholtz solver

The computational engine of the least-squares migration is the damped Helmholtz equation in three dimensions. Let us revise (1), as

$$-\frac{\partial^2\phi}{\partial x^2} - \frac{\partial^2\phi}{\partial y^2} - \frac{\partial^2\phi}{\partial z^2} - (1-\alpha i)\omega^2\sigma^2\phi = g, \tag{16}$$

and introduce a damping coefficient $\alpha \ll 1$. The closer the damping parameter α is set to zero, the more difficult it is to solve the Helmholtz equation, as shown in Erlangga et al. [7]. In this case, we choose $\alpha = 0.05$. From our experiments, we have observed that this choice of α does not affect the quality of the image significantly within the LSM framework, however, it leads to faster computational times of the Helmholtz solver, see Knibbe et al. [15].

As the solver for the discretized Helmholtz equation, we have chosen the Bi-CGSTAB method preconditioned by a shifted Laplacian multigrid method with matrix-dependent transfer operators and a multi-colored Gauss-Seidel smoother (see Erlangga et al. [7] and Knibbe et al. [16]). The preconditioner for the system (16) is given by

$$M = -\Delta - (\beta_1 - i\beta_2)k^2 I, \qquad \beta_1, \beta_2 \in \mathbb{R} \tag{17}$$

where Δ is the discretized Laplace operator, I is the identity matrix, and β_1, β_2 can be chosen optimally. Depending on β_1 and β_2, the spectral properties of the matrix AM^{-1} change. In Erlangga et al. [7], the Fourier analysis shows that (17) with $\beta_1 = 1$ and $0.4 \le \beta_2 \le 1$ gives rise to favorable properties that give rise to considerably improved convergence of the Krylov subspace method (e.g., Bi-CGSTAB), see also van Gijzen et al. [9]. For the LSM framework, we have chosen $\beta_2 = 0.8$ for robustness reasons.

In the coarse grid correction phase, the Galerkin method is used in order to get coarse grid matrices:

$$M_l = R_{l-1} M_{l-1} P_{l-1}, \tag{18}$$

where M_l and M_{l-1} are matrices on the coarse and fine grids, respectively, P_{l-1} is prolongation and R_{l-1} is restriction, $l = 0, \ldots, m$. The finest grid is denoted by the index 0 and the coarsest with m, respectively. The prolongation P_{l-1} is based on the 3-D matrix-dependent prolongation, described in Zhebel [34] for real-valued matrices. Since the matrix M_{l-1} is a complex-valued symmetric matrix, the prolongation is adapted for this case. This prolongation is also valid at the boundaries.

The restriction R_{l-1} is chosen as full weighting restriction and not as the adjoint of the prolongation. It provided a robust convergence for several complex-valued Helmholtz problems in Erlangga et al. [7].

As the smoother within the preconditioner, the multi-colored Gauss-Seidel method has been used. In particular, for 3D problems, the smoother uses eight colors, so that the color of a given point will be different from the color of its neighbors.

It has been shown in Knibbe et al. [15] that the preconditioned Helmholtz solver is parallelizable on CPUs as well as on a single GPU and provides an interesting speedup on parallel architectures.

3 Model problems

Before we dive into the acceleration techniques, let us consider three model problems: one with a "close to constant" velocity field, a second one with significant velocity variation, and a realistic third velocity field. These model problems will be used further for illustration and comparison purposes.

The first model problem **MP1** represents a wedge that consists of two dipping interfaces separating in the medium different constant velocities. Figure 1 (top) shows the velocities in **MP1**. This model problem represents a very smooth medium with two contrast interfaces. Since the velocities in both parts of the model are constant, the coefficients in the discretization and the prolongation matrices are mainly constant too.

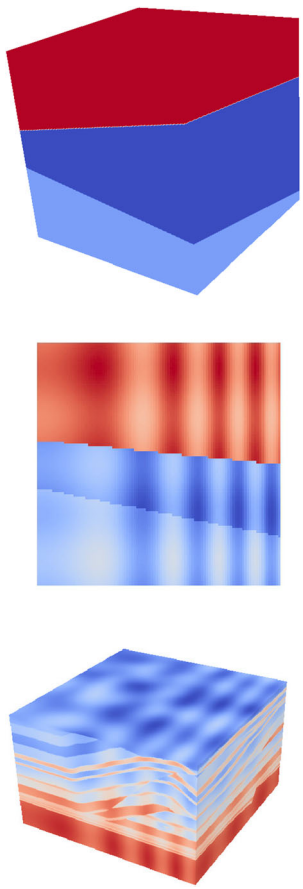


Fig. 1 Velocity functions for the wedge model problem **MP1** (*top*), for the modified wedge problem **MP2** (*center*), and for the modified Overthrust model problem **MP3** (*bottom*)

The second model **MP2** is based on the previous model with additional smooth sinusoidal velocity oscillations in each direction, shown in Fig. 1 (center). The velocity model is heterogeneous, thus, the coefficients of the discretization and prolongation matrices are not constant anymore. Since we can vary easily the problem size, this model problem is used to study the effects of compression on the convergence of the preconditioned Bi-CGSTAB method. For our experiments, we use this problem in three- and two-dimensions $\mathbf{MP2_{3d}}$ and $\mathbf{MP2_{2d}}$, respectively.

As the background velocity for the third model problem **MP3**, the SEG/EAGE Overthrust velocity model has been chosen, described in Aminzadeh et al. [1]. On top of it, additional smooth sinus oscillations are added in each direction. This model problem is close to a realistic problem. The velocity model is heterogeneous as shown in Fig. 1 (bottom), so that matrix entries of the discretization and prolongation matrices exhibit many variations and are far from constant. A reason for choosing additional smooth oscillations is to simulate a smooth update in the case of the full waveform inversion algorithm. This way the robustness of the proposed scheme can be validated for this application area.

4 Very Compressed Row Storage (VCRS) format

As already known, an iterative solver for the wave equation in frequency domain requires more memory than an explicit solver in time domain, especially for a shifted Laplace multigrid preconditioner based on matrix-dependent prolongation. Then, the prolongation and coarse grid-correction matrices need to be stored in memory. Since we are focusing on sparse matrices, in this section, we suggest a new format to store the sparse matrices that reduces memory and speeds up the matrix-vector operations.

4.1 VCRS description

First of all, let us briefly describe the well-known CSR format for storage of sparse matrices, e.g., Bai et al. [2], Saad [26]. It consists of two integers and one floating point array. The non-zero elements $a_{i,j}$ of a matrix A are consecutively, row by row, stored in the floating point array `data`. The column index j of each element is stored in an integer array `cidx`. The second integer array `first` contains the location of the beginning of each row. To illustrate this storage format, let us consider a small matrix from a

one-dimensional Poisson equation, with Dirichlet boundary conditions,

$$A = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix}.$$

The CSR format of this matrix is given by

$$\texttt{first} = \{0\ 2\ 5\ 8\ 10\},$$
$$\texttt{cidx} = \{0\ 1 \mid 0\ 1\ 2 \mid 1\ 2\ 3 \mid 2\ 3\},$$
$$\texttt{data} = \{2\ -1 \mid -1\ 2\ -1 \mid -1\ 2\ -1 \mid -1\ 2\}.$$

Note that the count starts at zero, which can however be easily adjusted to a starting index equal to 1.

To take advantage of the redundancy in the column indices of a matrix constructed by a discretization with finite differences or finite elements on structured meshes, we introduce a *new sparse storage format* inspired by the CSR format. The first array contains the column indices of the first non-zero elements of each row

$$\texttt{col\_offset} = \{0\ 0\ 1\ 2\}.$$

The second array consists of the number of non-zero elements per row

$$\texttt{num\_row} = \{2\ 3\ 3\ 2\}.$$

From an implementation point of view, if it is known that a row does not have more than 255 non-zero elements, then 8 bits integers can be used to reduce the storage of `num_row`. The third array is `col_data` which represents a unique set of indices per row, calculated as the column indices of the non-zero elements in the row `cidx` minus `col_offset`

$$\texttt{col\_data} = \{0\ 1 \mid 0\ 1\ 2\}.$$

Here, the row numbers 0 and 3 have the same set of indices, $\texttt{col\_data} = \{0\ 1\}$, and the row numbers 1 and 2 have the same set of indices as well, $\texttt{col\_data} = \{0\ 1\ 2\}$. To reduce redundancy in `col_data`, we introduce a fourth array `col_pointer` that contains an index per row, pointing at the starting positions in the `col_data`, i.e.,

$$\texttt{col\_pointer} = \{0\ 2\ 2\ 0\}.$$

This approach is also applied to the array containing values of the non-zero elements per row, i.e., the set of values is listed uniquely,

$$\texttt{data} = \{2\ -1 \mid -1\ 2\ -1 \mid -1\ 2\}.$$

Therefore, also here we need an additional array of pointers per row pointing at the positions of the first non-zero value in a row in `data`, i.e.,

$$\texttt{data\_pointer} = \{0\ 2\ 2\ 5\}.$$

For ease of notation, let us call the new format *VCRS*. At a first glance, it seems that the VCRS format is based on more arrays than the CSR format, six versus three, respectively. However, the largest arrays in the CSR format are `cidx` and `data`, and they contain redundant information of repeated indices and values of the matrix. For small matrices, the overhead can be significant, however, for large matrices, it can be beneficial to use the VCRS, especially on GPU hardware with limited memory.

Summarizing, the following factors contribute to the usage of the VCRS format:

1. The CSR format of a large matrix contains a large amount of redundancy, especially if the matrix arises from a finite-difference discretization;
2. The amount of redundancy of a matrix can vary depending on the accuracy and storage requirements, giving the opportunity to use a lossy compression;
3. The exact representation of matrices is not required for the preconditioner, an approximation might be sufficient for the convergence of the solver.

The lossy compression of preconditioners can be very beneficial for a GPU-implementation, as it allows to store the data on hardware with a limited amount of memory, but at the same time takes advantage of its speed compared to CPU hardware.

In this paper, we use two mechanisms to adjust the data redundancy: quantization and row classification. Note that these mechanisms can be used separately or in combination.

Quantization is a lossy compression technique that compresses a range of values to a single value, see, e.g., Gersho and Grey [8]. It has well-known applications in image processing and digital signal processing. By lossy compression, as opposed to lossless compression, some information will be lost. However, we need to make sure that the effect of the data loss in lossy compression does not affect the accuracy of the solution. The simplest example of quantization is rounding a real number to the nearest integer value. A similar idea applied to the lossless compression of the column indices was described in Kourtis et al. [18]. The quantization technique can be used to make the matrix elements in different rows similar to each other for better compression. The quantization mechanism is based on the maximum and minimum values of a matrix and on a number of so-called bins or sample intervals. Figure 2 illustrates the quantization process of a matrix with values on the interval [0, 1]. In this example, the number of bins is set to 5, meaning there are five intervals $[\ 0.2(i-1),\ 0.2i\ ),\ i = 1, \ldots, 5$. The matrix entries are normally distributed between 0 and 1, as shown by the black dots connected with the solid line. By applying quantization, the matrix values that fall in a bin are assigned to be a new value equal to the bin center. Therefore, instead of the whole range of matrix entries, we only get 5 values. Obviously, the larger number of bins, the more accurate is the representation of matrix entries.

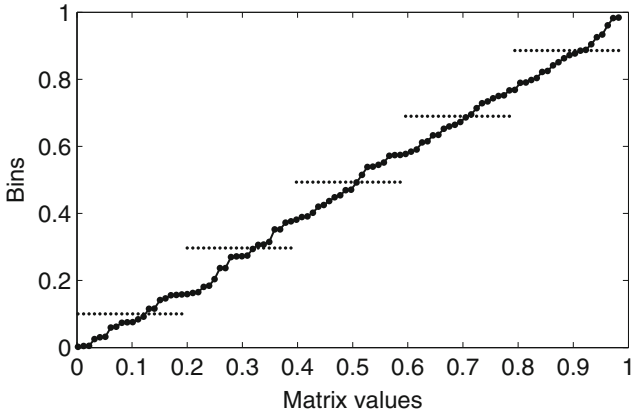


Fig. 2 Quantization of a matrix with normal distribution of entries in the interval [0, 1]. The number of bins is equal to 5

Next, we introduce *row classification* as a mechanism to define similarity of two different matrix rows. Given a sorted array of rows and a tolerance, we can easily search for two rows that are similar within a certain tolerance. The main assumption for row comparison is that the rows have the same number of non-zero elements. Let $R_i = \{a_{i1}\ a_{i2}\ \ldots\ a_{in}\}$ be the i-th row of matrix A of length n and $R_j = \{a_{j1}\ a_{j2}\ \ldots\ a_{jn}\}$ be the j-th row of A.

Algorithm 1 Comparison of two matrix rows

```
Procedure: IsRowSmaller(R_i,R_j,λ)
for each integer k = 1,...,n do
    if IsComplexValueSmaller(a_ik,a_jk,λ) then
        return true ;
    end
    if IsComplexValueSmaller(a_jk,a_ik,λ) then
        return false ;
    end
    a_jk,a_ik are equal, continue ;
end
rows R_i,R_j are equal ;
return false ;
```

Algorithm 2 Comparison of two complex numbers

```
Procedure: IsComplexValueSmaller(x,y,λ)
if IsValueSmaller(Re(x),Re(y),λ) then
    return true ;
end
if IsValueSmaller(Re(y),Re(x),λ) then
    return false ;
end
return IsValueSmaller(Im(x),Im(y),λ);
```

Algorithm 3 Comparison of two floating-point numbers

```
Procedure: IsValueSmaller(x,y,λ)
return x + λ max(|x|,|y|) < y ;
```

The comparison of two rows is summarized in Algorithm 1. If R_i is not smaller than R_j and R_j is not smaller than R_i, then the rows R_i and R_j are *"equal within the given tolerance* λ*"*. Algorithm 2 then describes the comparison of two complex values and Algorithm 3 compares two floating-point numbers. Figure 3 illustrates the classification of a complex number a_{ij}. Within a distance λ, the numbers are assumed to be equal to a_{ij}. Then, a_{ij} is smaller than the numbers in the dark gray area in Fig. 3, and larger than the numbers in the light gray area.

The number of bins and tolerance have influence on

1. the compression factor $c = m/m_c$, which is ratio between the memory usage of the original matrix m and the memory usage of the compressed matrix m_c;
2. the maximum norm of the compression error $\|e\|_\infty = \max_{i,j}(|a_{ij}| - |\bar{a}_{ij}|)$, where a_{ij} are the original matrix entries and $\bar{a}_{ij}$ are entries of the compressed matrix;
3. the computational time;
4. the memory usage;
5. the speedup on modern hardware which is calculated as a ratio of the computational time of the algorithm using the original matrix and of the computational time using the compressed matrix.

Next, we consider the effect of the VCRS format for matrix-vector multiplication, on the multigrid preconditioner, and the preconditioned Bi-CGSTAB method.

4.2 Matrix-vector multiplication

Two parameters, the number of bins from quantization and the tolerance λ from row classification, have an impact on

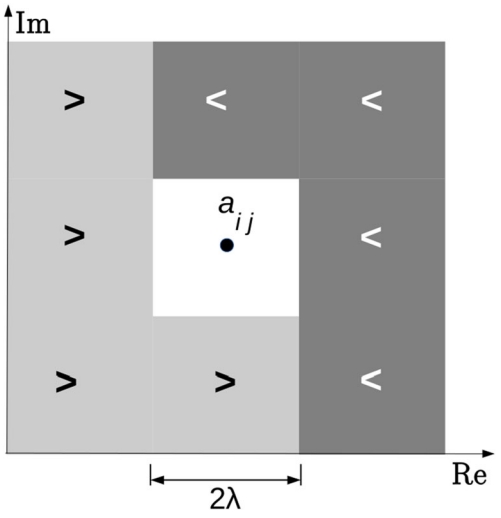


Fig. 3 Classification of a complex number a_{ij}. The numbers falling in the *white square* around a_{ij} are assumed to be equal to a_{ij}. Then, a_{ij} is smaller than the numbers in the *dark gray area* and larger than the numbers in *light gray area*

the accuracy, performance, and memory usage. To illustrate the effect, let us first consider model problem **MP1**. We compare the matrix-vector multiplication in VCRS format with that in the standard CSR format on a CPU. As we have mentioned, matrix-vector multiplication for the discretization matrix on the finest level is performed in a matrix-free way. Therefore, the prolongation matrix on the finest level has been chosen as the test example, since it is the largest matrix that needs to be kept in memory.

The results are shown in Fig. 5(left), where the maximum error is given in subfigure (a), the computational time in seconds in subfigure (b), the memory in GB in subfigure (c), the speedup in subfigure (d), and the compression factor in subfigure (e), respectively. As expected, the smaller the tolerance λ used, the more accurate is the representation of the compressed matrix, and the maximum error is reduced. Since model problem **MP1** has two large areas with constant velocity, the entries of the prolongation matrix are mostly constant. Even if the entries are not represented accurately because of the larger tolerance or the smaller number of bins, the number of non-zero elements does not change significantly. Therefore, the computational time and thus speedup, memory usage, and compression factor are very similar for all combinations of λ and number of bins.

Model problem **MP3** has significant velocity variation, so that the prolongation matrix has different coefficients in each row, as shown in Fig. 4. Note that the quantization has been done on the real and imaginary parts separately. Obviously, for the real part, the quantization will have a smaller effect than for the imaginary part, since the real values are clustered whereas the imaginary values are distributed over a larger interval.

Figure 5(right) shows the accuracy (a), the computational time in seconds (b), the memory in GB (c), the speedup (d), and the compression factor (e). It can be seen in Fig. 5a that by increasing the number of bins, the accuracy of the matrix-vector multiplication is also increasing, since two rows will less likely be similar. Reduced tolerance λ also contributes to the decrease of the maximum error, because the values of entries in rows are becoming closer to each other. With an increasing number of bins and a decreasing tolerance, the compression factor (Fig. 5e) and speedup (Fig. 5d) decrease and, therefore, the computational time increases (Fig. 5b). The compression factor is decreasing, because the more bins are used, the closer the matrix resembles its uncompressed form. Therefore, the memory usage increases (see Fig. 5c). With larger compression factor, the matrix has a smaller size in the memory, so that the cache effect contributes to the performance increase.

Obviously, there is a trade-off between performance and accuracy. The more accurate the compressed matrix, the slower the matrix-vector multiplication. Based on our experiments above, the most reasonable parameter choice would

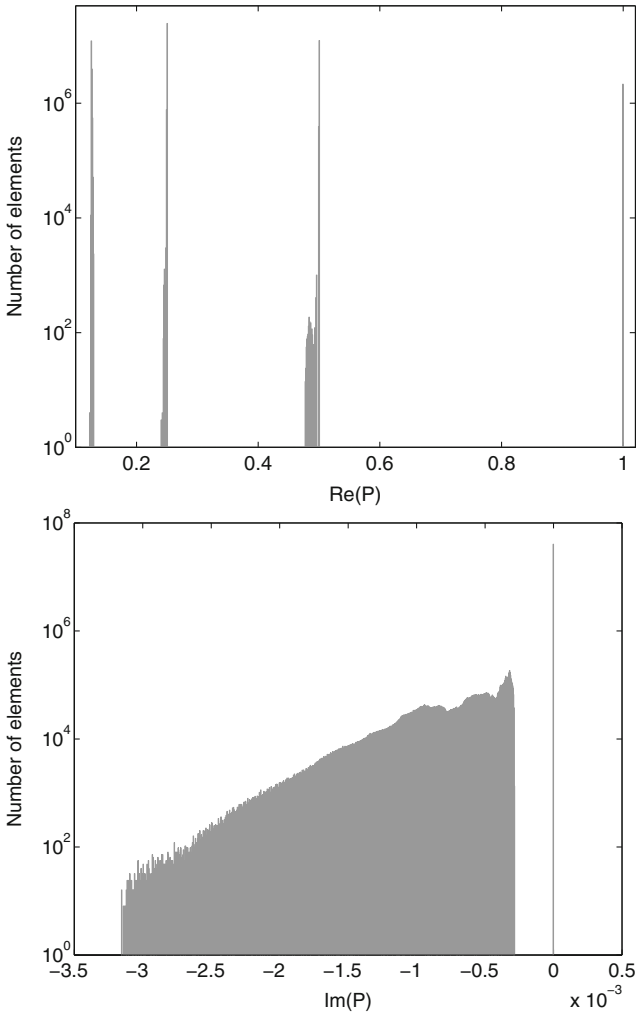


Fig. 4 Value distribution of the prolongation matrix on the finest level for the model problem **MP3**

be a tolerance $\lambda = 0.1$ and number of bin equals to 10^5. We will investigate the effect of this parameter choice on the multigrid preconditioner and the complete Helmholtz solver.

4.2.1 Different application, reservoir simulation

The VCRS compression can also be used in other applications where the solvers are based on a preconditioned system of linear equations. For example, an iterative solver for a system of linear equations is also an important part of a reservoir simulator, see, e.g., Chen et al. [4]. It appears within a Newton step to solve discretized nonlinear partial differential equations describing the fluid flow in porous media. The basic partial differential equations include a mass-conservation equation, Darcy's law, and an equation of state relating the fluid pressure to its density. In its original form, the values of the discretization matrix are scattered, see Fig. 6(top). Although the matrix looks full

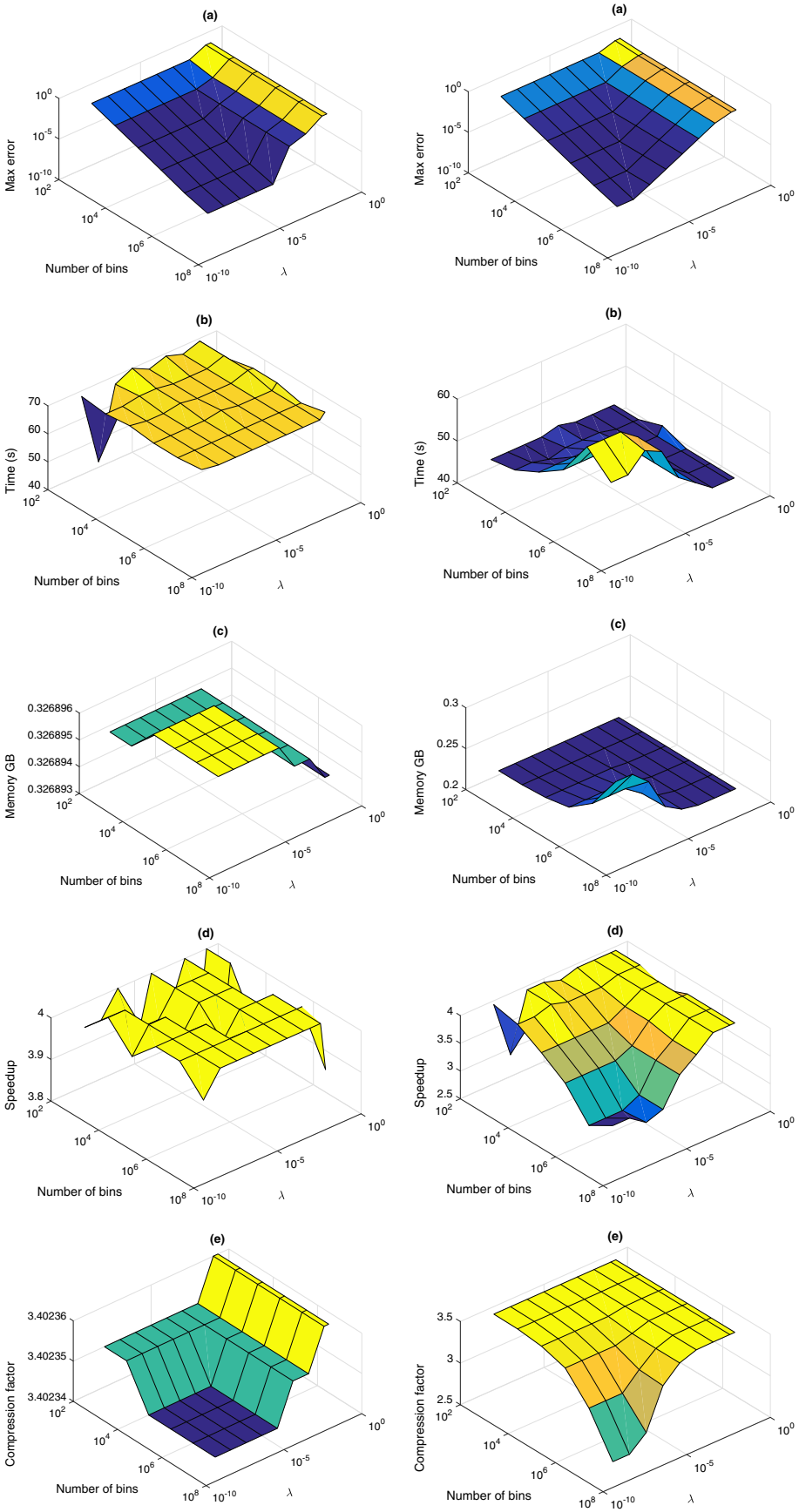


Fig. 5 Effect of the VCRS format for the matrix-vector multiplication of **MP1** (*left*) and of **MP3** (*right*) on the maximum error (**a**), computational time (**b**), memory (**c**), speedup (**d**), compression factor (**e**)

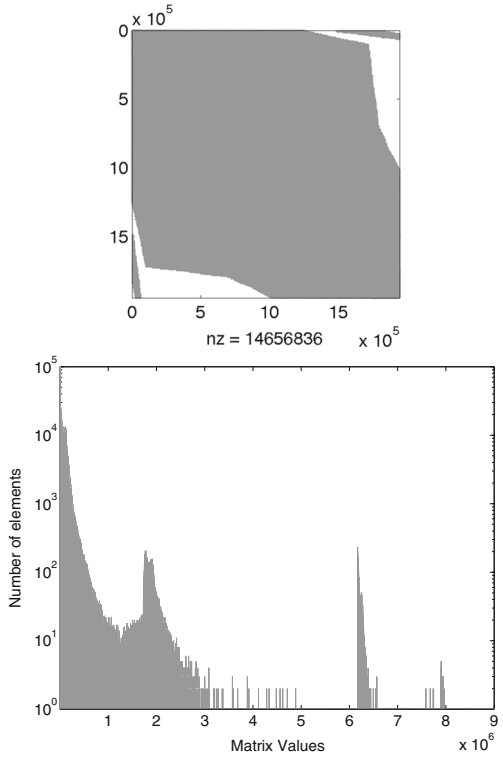


Fig. 6 The original matrix from pressure solver (*top*) and its value distribution (*bottom*)

due to the scattered entries, the most common number of non-zero elements per row is equal to 7, however, the maximum number of elements per row is 210. The distribution of the matrix values is shown in Fig. 6 (bottom). Note that the matrix has real-valued entries only. It can be seen that there is a large variety of matrix values that makes the quantization and row classification effective. Using the VCRS format to store this matrix results in two to three times smaller memory requirements and two to three times faster matrix-vector multiplication, depending on the compression parameters. Of course, the effect of the compression parameters on the solver still needs to be investigated.

4.3 Multigrid method preconditioner

Since matrix A is implemented in stencil version, it does not require any additional storage. However, the lossy VCRS format may be very useful for the preconditioner, as, because of the matrix-dependent preconditioner, the prolongation and coarse grid correction matrices have to be stored at each level in the multigrid preconditioner. Note that recomputing those matrices on the fly will result in doubling or even tripling the total computational time.

The multigrid method consists of a setup phase and a run phase. In the setup phase, the prolongation, restriction and coarse grid correction matrices are constructed, and the matrices are compressed. In the run phase, the multigrid algorithm is actually applied. We assume that the multigrid method is acting on grids at levels $l, l = 0, \ldots, m$, the finest grid being $l = 0$ and the coarsest grid being $l = m$. On each level, except for $l = m$, the matrix-dependent prolongation matrix has to be constructed. The restriction in our implementation is the standard restriction and can be easily used in a matrix-free fashion. At each level, except for $l = 0$, the coarse grid matrix has to be constructed using the Galerkin method (18).

We suggest to construct all the coarse-grid matrices exactly in the setup phase and use compression afterwards. This way, errors due to lossy compression will not propagate to the coarse-grid representation of the preconditioned matrix. The algorithm is summarized in Table 1. Matrix M_m, on the coarsest level m, is not compressed, since the size of M_m is already small.

To illustrate the compression factor at each level for the matrices M_l and P_l, $l = 1, \ldots, m$, for different values of the row classification parameter λ, let us fix the quantization parameter, i.e., the number of bins is set to 10^3. From Fig. 5, this choice of bins seems to be reasonable. Table 2 shows the compression factors for M and P on each multigrid level. The first column lists the values of the tolerance λ. The multigrid levels are shown in the second column. The third multi-column presents the compression factor, i.e., the percentage of total memory usage of the preconditioner after compression and the maximum error in the matrix elements of the compressed matrix P compared to its exact representation at each multigrid level. The fourth multi-column represents the compression factor, the percentage of the preconditioner memory usage after compression and the maximum error in the matrix elements of the compressed matrix M compared to its exact representation at each multigrid level. The last column shows the total memory usage of the preconditioner for the given tolerance λ. Note that the total memory usage for the uncompressed matrix is 810 MB.

Table 1 Summary of the exact and compressed matrices used in the matrix-dependent multigrid preconditioner

Level	Setup (exact)	Run	
		Matrix-free	Compressed
0	M_0, P_0, R_0	M_0, R_0	P_0
1	M_1, P_1, R_1	R_1	M_1, P_1
...	...	...	...
l	M_l, P_l, R_l	R_l	M_l, P_l
...	...	...	...
$m-1$	$M_{m-1}, P_{m-1}, R_{m-1}$	R_{m-1}	M_{m-1}, P_{m-1}
m	M_m		

Table 2 Model problem **MP3** (3D Overthrust), nbins = 10^3

λ	Level	P			M			Memory
		Factor	Memory	Error	Factor	Memory	Error	
0.2	0	3.4	76%	2.7e-2				276 MB
	1	3.2	10%	5.1e-2	24.1	10 %	3.5e-3	
	2	2.9	1 %	1.0e-1	19.3	1.5 %	1.6e-3	
	3	2.3	< 1%	1.6e-1	5.3	< 1%	7.3e-4	
	4				1.9	< 1%	2.4e-4	
0.1	0	3.4	71%	2.3e-2				293 MB
	1	3.3	10%	5.1e-2	20.1	11 %	3.1e-3	
	2	3.1	1.4 %	5.1e-2	6.8	3.4 %	1.5e-3	
	3	2.2	< 1%	6.8e-2	2.4	< 1%	1.0e-3	
	4				1.5	< 1%	2.6e-4	
0.01	0	3.4	57%	4.8e-3				367 MB
	1	2.7	9%	4.8e-3	9.3	19 %	1.8e-4	
	2	2.1	1.4 %	5.1e-3	2.1	10 %	4.7e-5	
	3	1.7	< 1%	4.8e-3	1.6	2 %	1.5e-5	
	4	1.4	< 1%		1.4	< 1%	3.7e-6	
0.001	0	3.4	54%	4.4e-4				382 MB
	1	2.4	9%	4.4e-4	8.4	21 %	4.2e-5	
	2	2.1	1.4 %	5.0e-4	2.1	11 %	1.0e-5	
	3	1.8	< 1%	1.1e-3	1.5	2 %	5.1e-6	
	4	1.4	< 1%	9.2e-4	1.4	< 1%	3.6e-6	

The compression factor for the prolongation matrix almost does not change on the finest level for different tolerance parameters, since the prolongation coefficients are constructed from the discretization matrix based on the 7-point discretization scheme. On the coarsest level, the stencil becomes a full 27-point stencil and the effect of the compression is more pronounced for smaller tolerance parameter λ. Clearly, the most memory consuming part is the prolongation matrix P_0 on the finest level. However, with decreased tolerance λ, the coarse-grid correction matrices need more memory due to larger variety of the matrix entries. As expected, the maximum error is reduced when the tolerance is decreasing in both cases for P and M, respectively. Due to the 27-point stencil on the coarser grids, the coarse-grid correction matrix has a wider spread of the matrix values, which affects the compression factor with respect to the tolerance parameter. The more accurate compression is required, the more memory is needed to store coarse-grid correction matrices. Therefore, to compromise between the accuracy and the memory usage for the multigrid preconditioner, the tolerance λ is chosen equal to 0.1.

A prolongation matrix stored in the VCRS format with lossy compression uses less memory than the original matrix, which can be seen as an approximation. However, the matrix-dependent characteristics of the original prolongation must be preserved for satisfactory convergence of the Helmholtz solver, otherwise, it would be easier to just use a standard prolongation matrix. A standard prolongation matrix can be implemented in a matrix-free manner.

4.4 Preconditioned Bi-CGSTAB

Let us consider the convergence of the preconditioned system,

$$AM^{-1}v = f, \qquad u = Mv,$$

starting with an exact preconditioner and followed by the preconditioner with lossy VCRS compression. From the previous sections, it is clear that standard analysis on a simple homogeneous problem is not sufficient, because here the tolerance and the number of bins will have a minimum effect on the lossy compression.

4.4.1 Two-dimensional problem

Let us first consider a two-dimensional variant of the heterogeneous model problem **MP2$_{2d}$**. In this case, we can vary

the tolerance and the number of bins and observe the effect on the convergence properties of the preconditioned system. Also, an analytic derivation of the spectral radius of the multigrid iteration matrix is not possible, and therefore, we will use numerical computations to determine it.

In our work, we focus on quantitative estimates of the convergence of the multigrid preconditioner, see, e.g., Trottenberg et al. [30]. To do this, we construct and analyze the shifted Laplace multigrid operator M^{-1} from Eq. 17 with $\beta_1 = 1$ and $\beta_2 = 0.8$. Two-grid analysis has been widely described in the literature, see Brackenridge [3], Stüben and Trottenberg [28], Trottenberg et al. [30]. Three-grid analysis has been done in Wienands and Oosterlee [33]. To see the effect of the lossy VCRS compression, a true multigrid matrix needs to be constructed. The four-grid operator with only pre-smoothing for the F-cycle is given by

$$
\begin{aligned}
M^{-1} := T_4 &= S_0\,(I_0 - P_0F_1V_1R_0M_0)\,S_0, \qquad \text{with}\\
V_1 &= S_1(I_1 - P_1V_2R_1M_1)S_1\\
F_1 &= S_1(I_1 - P_1F_2V_2R_1M_1)S_1\\
V_2 &= S_2(I_2 - P_2M_3^{-1}R_2M_2)S_2\\
F_2 &= S_2(I_2 - P_2M_3^{-1}M_3^{-1}R_2M_2)S_2,
\end{aligned}
$$

where S_0, S_1, S_2 are smoothers on the finest, first, and second grid, P_0, P_1, P_2, and R_0, R_1, R_2 are prolongation and restriction matrices, respectively. M_0 is the discretization matrix on the finest grid and M_1, M_2, M_3 are respective coarse-grid correction matrices.

Next, we compute the spectral radius ρ of AM^{-1}, which is the maximum of the absolute eigenvalues, using the four-grid operator as the preconditioner. The results are summarized in Table 3 for different values of λ and different numbers of bins, that are given in the first and second rows. The third row presents the number of iterations of Bi-CGSTAB applied to the preconditioned system, the stopping criterion is 10^{-7} for the relative residual. The fourth row shows the spectral radius of the compressed matrices with VCRS preconditioner. The second column with $\lambda = 0.0$ and #bins$= 0.0$ represents results for the exact preconditioner. Note that the spectral radius is larger than 1 which means that the multigrid does not converge without the Krylov subspace method for model problem **MP2$_{2d}$**.

However, for illustration purposes of the compression, it is interesting to consider the spectral radius too. For large tolerance $\lambda = 1.0$, the spectral radius is far from the exact one, meaning that the compressed preconditioned system does not resemble the original preconditioned system and the number of iterations may rapidly increase. For smaller tolerance, the spectral radius of the compressed preconditioned system is very similar to the exact spectral radius and the iteration numbers are almost the same. Note thatthe number of bins does not have a significant influence on the number of iterations of the Helmholtz solver in this case.

Figure 7 illustrates the two extreme cases for **MP2$_{2d}$**, where the number of iterations is most and least affected by the compression. The eigenvalues of the exact operator are shown by blue crosses and of the compressed operator by red circles. On the left, the compression parameters are $\lambda = 1.0$, #bins$= 10^2$, and on the right, $\lambda = 0.01$, #bins$=10^6$, respectively. Clearly, the more accurate compression (shown on the bottom) gives a better approximation of the eigenvalues of the exact operator, therefore, the convergence is only slightly affected by the compression. The least accurate compression (shown on the top) affects the eigenvalues of the preconditioned system and therefore, Bi-CGSTAB needs many more iterations.

4.4.2 Three-dimensional problem

The number of iterations for the more realistic example **MP3** are given in Table 4. The results are presented for different values of λ and numbers of bins, that are given in the first and second row. The third row shows the number of iterations of Bi-CGSTAB applied to the preconditioned system, the stopping criterion is 10^{-7} for the relative residual. The second column with $\lambda = 0.0$ and #bins$= 0.0$ represents results for the exact preconditioner. For tolerance $\lambda = 1.0$, the preconditioned Bi-CGSTAB method does not converge anymore. Therefore, it is advised to use the tolerance smaller than 1.0. In case of **MP3**, the number of bins influences the iterations number of the preconditioned Bi-CGSTAB. This happens because the spreading of the matrix entries for the realistic three-dimensional problem is large, therefore, the quantization affects many matrix entries significantly. For a relatively large number of bins, the number of iterations is close to the uncompressed case.

Table 3 Spectral radius ρ for preconditioned system AM^{-1} for four-grids using VCRS format for several λ and number of bins for model problem **MP2$_{2d}$**, $\omega = 10$ Hz

λ	0.0	1.0	0.2	0.2	0.1	0.1	0.01	0.01
# bins	0.0	10^2	10^2	10^6	10^2	10^6	10^2	10^6
# iter	24	124	24	24	24	24	23	24
ρ	1.149	1.966	1.158	1.151	1.155	1.150	1.158	1.150

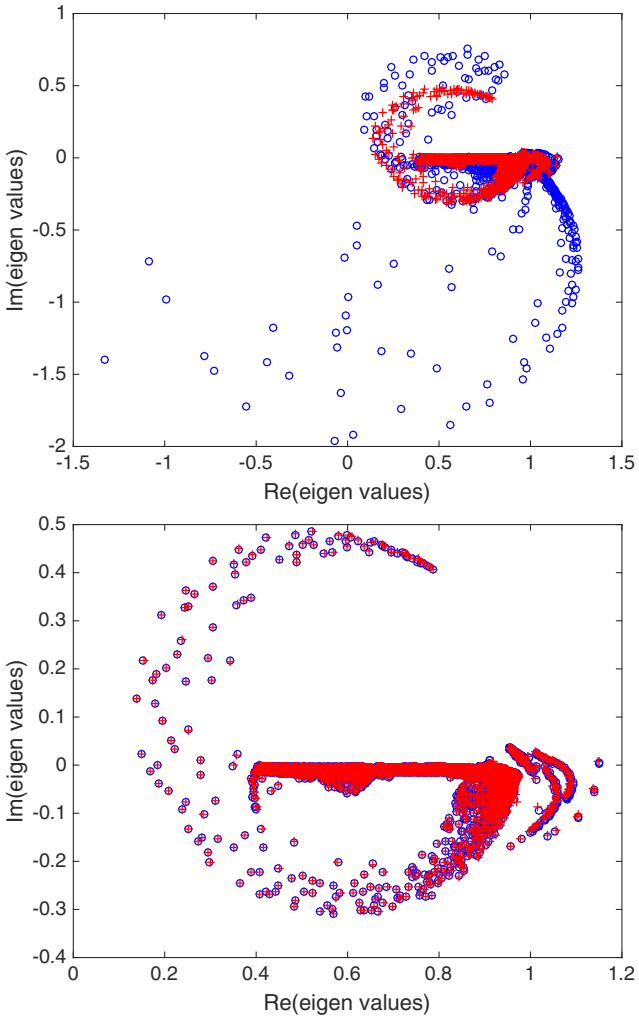


Fig. 7 Comparison of exact eigenvalues (*red crosses*) for the preconditioned system AM^{-1} by a four-grid method with approximate eigenvalues (*blue circles*) using the VCRS format for $\lambda = 1.0$, bins=10^2 (*top*) and $\lambda = 0.01$, bin=10^6 (*bottom*), model problem $\mathbf{MP2_{2d}}$, $\omega = 10$ Hz

5 Implementation details

Presently, a common hardware configuration is a CPU connected to two GPUs that contain less memory than the CPU. By a "CPU", we refer here to a multi-core CPU and by a "GPU" to an NVidia general purpose graphics card. We identified the parts of the algorithms that can be accelerated on a GPU and implemented them in CUDA 5.0.

5.1 GPU

We consider the GPU as a replacement for the CPU and as an accelerator. In both cases, the Bi-CGSTAB algorithm is executed on the CPU, since the storage of temporary vectors takes the most of the memory space. Executing the Bi-CGSTAB method on a GPU would significantly limit the problem size. Therefore, we split the algorithm, where the Krylov solver runs on a CPU and the preconditioner runs on one or more GPUs. We exploited this technique in Knibbe et al. [16] and concluded that it reduces the communication between different devices.

When the GPU is used as a *replacement*, the hardware setup consists of one multi-core CPU connected to one GPU. Then the data for the preconditioner, i.e., prolongation and coarse-grid correction matrices, lives in GPU memory to avoid memory transfers between CPU and GPU memory. The process is illustrated in Fig. 8. A Bi-CGSTAB vector is transferred from the CPU to the GPU, then the multigrid preconditioner is applied. The prolongation and coarse-grid correction matrices are already located in the GPU memory, since they have been transferred in the setup phase. After the preconditioner is applied, the resulting vector is copied back to the CPU memory and the iterations of Bi-CGSTAB continue. As GPUs have generally much less memory available than CPUs, this impacts the size of the problem under consideration. The VCRS format can be used to increase the size of the problem that would fit into GPU memory, moreover, it also leads to increased performance.

When the GPU is used as an *accelerator*, then a multi-core CPU is connected to one or more GPUs, see Fig. 8. This means that part of the data for the preconditioner lives in the CPU memory, and it is copied to the GPU only for the duration of the relevant computational intensive operation, for example, a multi-colored Gauss-Seidel iteration or matrix-vector multiplication. The data is copied back to the CPU memory once the operation is finished. Note that only the vectors are transferred back-and-forth between the CPU and GPU memories, the matrices stay in the GPU memory. Also, on each GPU, memory for an input and an output vector on the finest grid needs to be allocated to receive vectors from the CPU. Then all the vectors from the preconditioner will fit into this allocated memory. This approach takes advantage of the memory available across GPUs. Using the VCRS format, the matrices become small enough so that they can be evenly distributed across two GPUs. For example, in case

Table 4 Number of iterations for preconditioned system AM^{-1} using VCRS format for several λ and number of bins for model problem **MP3**, $\omega = 20$ Hz

λ	0.0	1.0	1.0	0.2	0.2	0.1	0.1	0.01	0.01
# bins	0.0	10^2	10^6	10^2	10^6	10^2	10^6	10^2	10^6
# iter	18	> 400	> 400	60	20	59	20	58	19

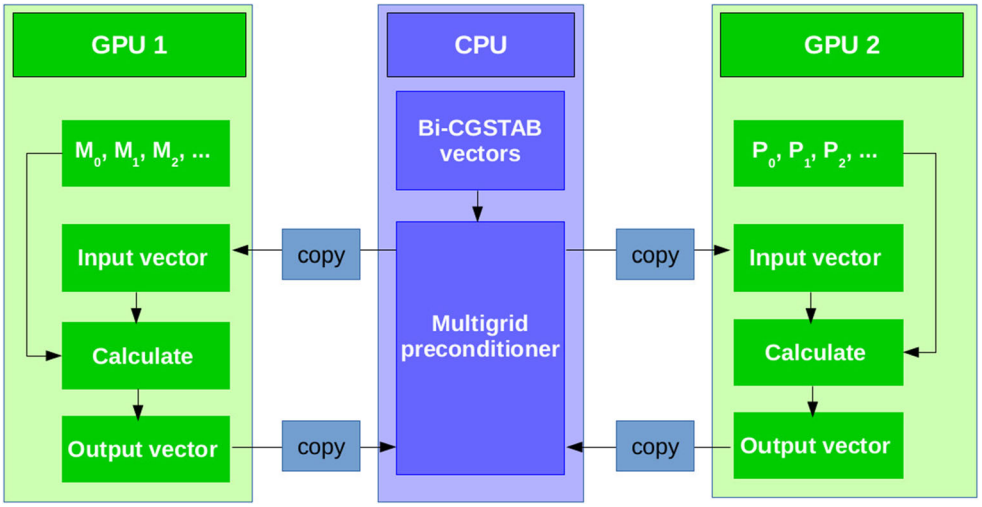


Fig. 8 GPU as a replacement (**a**) and as an accelerator (**b**)

of two GPUs connected to one CPU, we suggest to store the prolongation matrices on one GPU and the coarse-grid correction matrices on the other GPU.

Table 5 shows the performance of the preconditioned Bi-CGSTAB method for $\mathbf{MP2_{3d}}$ on CPU, GPU as an accelerator and GPU as a replacement. The first column shows the chosen matrix storage format. The second column lists the used hardware. The compression parameters are given in the third and fourth columns. If no compression parameters are given, then lossless compression is applied, where no information is lost due to the matrix compression and the matrix entries are unchanged. Otherwise, the compression parameters belong to lossy compression, where some loss of information is unavoidable. The setup phase, number of iterations, and time per iteration are shown in the last three columns of the table. The setup phase for the VCRS format takes longer than the setup phase for the standard CSR because of the construction of additional arrays. The number of iterations does not change significantly for different formats, however, it increases slightly in the case of lossy compression. The VCRS format with lossy compression gives the best performance time per iteration without affecting the iteration numbers significantly.

The maximum number of unknowns for the CSR and VCRS formats with lossless and lossy compression are shown for $\mathbf{MP2_{3d}}$ in Table 6. The CPU, the GPU as an accelerator and the GPU as a replacement are considered. The first column shows the chosen matrix storage format. The second column lists the used hardware. The compression parameters are given in the third column. If no compression parameters are given, then lossless compression is applied. Across the different hardware platforms, the VCRS format increases the maximum size of the problem compared to the CSR matrix storage. Using GPU as a replacement leads to solving larger problem sizes than using GPU as an accelerator, because the memory needed to store preconditioner matrices is distributed across the GPUs.

Summarizing we can conclude that the VCRS format can be used to reduce the memory for the preconditioner as well as to increase the performance of the preconditioned Bi-CGSTAB on different hardware platforms.

5.2 Common code

The idea of one common hardware code on CPU and GPU has a number of benefits. Just to name few of them, code duplication is kept to a minimum, reducing the possibility to make mistakes, easier maintainability and extensibility. There have been attempts to create a common high-level language for hybrid architectures, for example, OpenCL

Table 5 Performance of preconditioned Bi-CGSTAB on CPU, GPU as accelerator and GPU as a replacement for $\mathbf{MP2_{3d}}$ of size 250^3

Format	Hardware	# bins	λ	Setup (s)	# iter	Time per iter (s)
CSR	CPU	–	–	88	73	5.8
VCRS	CPU	–	–	175	73	4.9
VCRS	CPU	10^3	0.1	150	80	4.6
VCRS	GPU accel	–	–	180	73	3.4
VCRS	GPU accel	10^3	0.1	149	78	3.1
VCRS	GPU repl	–	–	148	73	2.8
VCRS	GPU repl	10^3	0.1	149	76	2.4

Table 6 Maximum number of unknowns for $\mathbf{MP2_{3d}}$ for given storage format on different hardware

Format	Hardware	Compression parameters	Maximum size
CSR	CPU	–	74,088,000
VCRS	CPU	$\lambda = 0.1$, #bins=10^3	94,196,375
VCRS	GPU accel	–	19,683,000
VCRS	GPU accel	$\lambda = 0.1$, #bins=10^3	27,000,000
VCRS	GPU repl	–	23,149,125
VCRS	GPU repl	$\lambda = 0.1$, #bins=10^3	32,768,000

that has been introduced by the Khronos group [13]. For our research, the idea of a common code has always been attractive, but its development really started when NVIDIA stopped to support the CUDA-emulator on CPU hardware. By the time our research had started, OpenCL was not commonly available. Therefore, we used our own approach for a common code on CPU and GPU. We assume the code on the CPU is using C++ and the code on the GPU is using CUDA, respectively.

Our implementation is based on the fact that CPUs and GPUs have multiple threads. Therefore, the multi-threading mechanism can be made abstract on the highest level of the program that describes the numerical algorithms. Code to setup the information about threads is abstracted in macros. On the device level, OpenMP is used for parallel computations on a CPU and CUDA is used on a GPU, respectively. Depending on the device where the part of the program is executed, the high-level functions call sub-functions specific for the device. For example, the synchronization of the threads after computations uses the so-called `omp_barrier()` function for a CPU and `cudaThreadsSynchronize()` for a GPU.

Another abstraction technique we use is to simplify argument handling. It uses one structure that contains all arguments of a function. This allows to copy all arguments with one command to a GPU. Memory transfers to the right hardware remain the responsibility of the developer.

Finally, there is one code that describes a numerical algorithm, for example, the multigrid preconditioner, that compiles for two different architectures, CPU and GPU. The code developments are done on a CPU in debug mode, where multi-threading is switched off and only one thread is used. This allows to develop on less powerful hardware or when CUDA is not available. The code can be easily expanded to other architectures as long as a subset of all the programming languages is used, in our case meaning the intersection of CUDA, C++, etc. Currently, CUDA is a limiting factor for a program in how many C++ language-specific features it can have. As soon as CUDA releases a version that supports more C++ features, it can be immediately used in the common code.

5.3 Task system

To run the least-squares migration in parallel, we have developed a *task system* that allows to split the work amongst compute nodes and monitor the execution. By a *compute node*, we assume a multi-core CPU connected to one or more GPUs, where GPUs can be used as a replacement or as an accelerator.

As we have seen in Eq. 11, the least-squares migration algorithm consists of forward modeling and migrations in frequency domain for each source. Therefore, the highest level of parallelism for LSM consists of parallelization over all sources and frequencies. That means one task consists of computations of one frequency ω_{s_i} for a given source s_i from the set of size ω times s, $s_i \in s$ on one compute node, $\omega_{s_i} \in \omega$, $i = 1, \ldots, N_s$ with the number of sources N_s. In total, we have $N_\omega N_s$ tasks, where N_ω is number of frequencies.

For each frequency, a linear system of equations needs to be solved. We have shown in Knibbe et al. [17] that the matrix size and memory requirements are the same for each frequency, but the lower frequencies require less compute time than the higher ones [7]. Here, we assume that one shot for one frequency in the frequency domain fits in one compute node.

For the LSM task system, we adapt a client-to-server approach, described in Knibbe et al. [17], where clients request tasks to the server. GPU clusters are either heterogeneous or they have to be shared simultaneously amongst the users. For example, the cluster at our disposition, Little Green Machine [20], has the same hardware (with the exception of one node), see Appendix. Similarly, the GPUs have the same specifications, but one GPU can already be used by a user while the other one remains available. The task system addresses the issue of load balancing. This is also handy when compute nodes are shared between users.

For each CGNR iteration, the server or *master node* creates one task per shot per frequency. Each task is added to a queue. When a client requests a task, a given task is moved from the queue to the active list as illustrated in Fig. 9. This example shows computation of the right-hand side in Eq. 11. The active list contains tasks in the table "Jobs being processed". The queue is given in the table "Remaining jobs". The column "Description" shows the task for a given frequency and corresponding source with related receivers called "panel". It can happen that a node will crash due to a hardware failure. In that case, the task will remain on the active list until all the other tasks have finished. Once

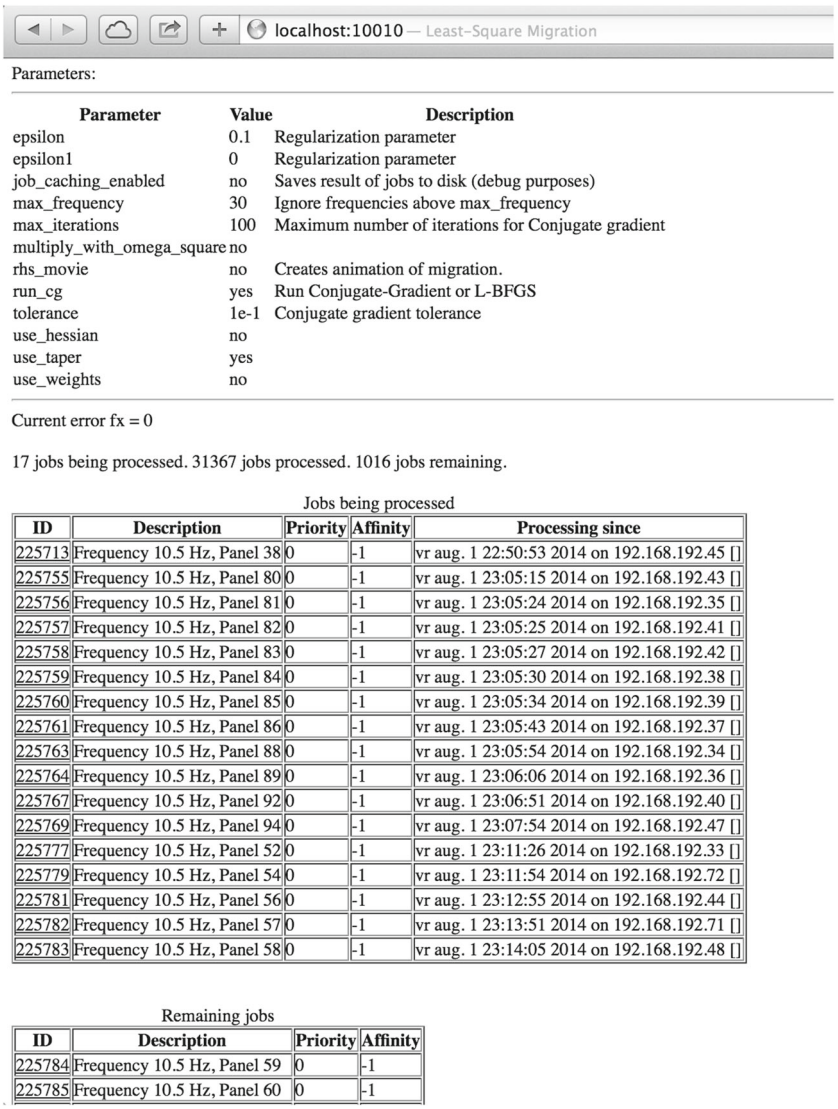
that happens, any unfinished task will be moved back to the queue, so that another compute node can take over the uncompleted work. When all tasks have been processed, the master node proceeds to the next CGNR iteration or stops if convergence is achieved.

Using the task system, the frequency decimation in Eq. 13 can be easily applied, since only the content of the queue will change, the implementation will stay the same. Our implementation has only a single point of failure: the master process. Furthermore, it is possible to adjust parameters on the fly by connecting into the master program using Telnet without having to restart the master program. The Telnet session allows the master program to process commands as well (load/save of restart points, save intermediate results, display statistics, etc.). It is also possible to manipulate the queues with an Internet browser.

6 Results

Combining the techniques described above, let us compare the results of the least-squares migration method with and without decimation for the original Overthrust velocity model problem [1]. Both implementations ran on the Little Green Machine [20]. Due to limited resources because of sharing with other students, we take a two-dimensional vertical slice of the overthrust velocity model for our LSM experiments to make sure it will fit in the available memory. The domain size of the test problem is $[0,\ 20000\ ] \times [0,\ 4650]$, that is discretized on a regular orthogonal grid with 4000 points in x-direction and 930 points in depth.

Both implementations use the VCRS format with the tolerance $\lambda = 0.1$ and the number of bins of 10^7. The decimation is applied with the decimation parameter $\delta = 10$,


localhost:10010 — Least-Square Migration

Parameters:

Parameter	Value	Description
epsilon	0.1	Regularization parameter
epsilon1	0	Regularization parameter
job_caching_enabled	no	Saves result of jobs to disk (debug purposes)
max_frequency	30	Ignore frequencies above max_frequency
max_iterations	100	Maximum number of iterations for Conjugate gradient
multiply_with_omega_square	no	
rhs_movie	no	Creates animation of migration.
run_cg	yes	Run Conjugate-Gradient or L-BFGS
tolerance	1e-1	Conjugate gradient tolerance
use_hessian	no	
use_taper	yes	
use_weights	no	

Current error fx = 0

17 jobs being processed. 31367 jobs processed. 1016 jobs remaining.

Jobs being processed

ID	Description	Priority	Affinity	Processing since
225713	Frequency 10.5 Hz, Panel 38	0	-1	vr aug. 1 22:50:53 2014 on 192.168.192.45 []
225755	Frequency 10.5 Hz, Panel 80	0	-1	vr aug. 1 23:05:15 2014 on 192.168.192.43 []
225756	Frequency 10.5 Hz, Panel 81	0	-1	vr aug. 1 23:05:24 2014 on 192.168.192.35 []
225757	Frequency 10.5 Hz, Panel 82	0	-1	vr aug. 1 23:05:25 2014 on 192.168.192.41 []
225758	Frequency 10.5 Hz, Panel 83	0	-1	vr aug. 1 23:05:27 2014 on 192.168.192.42 []
225759	Frequency 10.5 Hz, Panel 84	0	-1	vr aug. 1 23:05:30 2014 on 192.168.192.38 []
225760	Frequency 10.5 Hz, Panel 85	0	-1	vr aug. 1 23:05:34 2014 on 192.168.192.39 []
225761	Frequency 10.5 Hz, Panel 86	0	-1	vr aug. 1 23:05:43 2014 on 192.168.192.37 []
225763	Frequency 10.5 Hz, Panel 88	0	-1	vr aug. 1 23:05:54 2014 on 192.168.192.34 []
225764	Frequency 10.5 Hz, Panel 89	0	-1	vr aug. 1 23:06:06 2014 on 192.168.192.36 []
225767	Frequency 10.5 Hz, Panel 92	0	-1	vr aug. 1 23:06:51 2014 on 192.168.192.40 []
225769	Frequency 10.5 Hz, Panel 94	0	-1	vr aug. 1 23:07:54 2014 on 192.168.192.47 []
225777	Frequency 10.5 Hz, Panel 52	0	-1	vr aug. 1 23:11:26 2014 on 192.168.192.33 []
225779	Frequency 10.5 Hz, Panel 54	0	-1	vr aug. 1 23:11:54 2014 on 192.168.192.72 []
225781	Frequency 10.5 Hz, Panel 56	0	-1	vr aug. 1 23:12:55 2014 on 192.168.192.44 []
225782	Frequency 10.5 Hz, Panel 57	0	-1	vr aug. 1 23:13:51 2014 on 192.168.192.71 []
225783	Frequency 10.5 Hz, Panel 58	0	-1	vr aug. 1 23:14:05 2014 on 192.168.192.48 []

Remaining jobs

ID	Description	Priority	Affinity
225784	Frequency 10.5 Hz, Panel 59	0	-1
225785	Frequency 10.5 Hz, Panel 60	0	-1

Fig. 9 Example of a task system monitoring during the LSM execution. The active list contains tasks in table "Jobs being processed". The queue is given in the table "Remaining jobs"

meaning that the least-squares migration matrix is computed using 10 times less information, and therefore, 10 times faster than without decimation. The stopping criterion for the Helmholtz solver is 10^{-5}. The results for LSM are presented in Fig. 10, where the implementation without decimation is shown at the top, the implementation with decimation at the center and difference between those two at the bottom, respectively. The results represent the reflectivity of the overthrust velocity model. The color scale is the same for the three pictures. The results for both methods are very similar, even with the large decimation factor chosen.

The speedup of the LSM with decimation algorithm compared to the original LSM can be calculated theoretically. Assuming the computational time of the right-hand side operator $\mathbf{F^H R^H d}$ from Eq. 12 is equal to t, then the computational time to calculate the least-squares matrix equals $2t$ per iteration of the CGNR method. In total, the computational time of the original LSM is given by

$$T_{original} = t(2n_{iter} + 1), \tag{19}$$

where the n_{iter} denotes the total number of CGNR iterations. There is a possibility to decrease the total time of the LSM. This can be achieved by saving the smooth solution u_0 in Eq. 6 for each source and frequency to disk while computing the right-hand side and reusing it instead of recomputing for the construction of the LSM matrix. Assuming the time to read the solution u_0 from the disk negligible, the total time of the LSM reads

$$\hat{T}_{original} = t(n_{iter} + 1). \tag{20}$$

Of course, in this case, the compression of the solution on the disk becomes important, since the disk space is also usually limited. We recommend to use lossless compression of u_0 to avoid any effects of the lossy compression. This could be investigated in the future.

In case of the LSM with decimation, the time to compute the right-hand side is the same as for the original LSM. However, the computational time of the matrix on the left-hand side of the Eq. 13 is equal to t/δ, where δ is the decimation parameter. Then the total computational time of the LSM with decimation is given by

$$T_{decimation} = t(\frac{n_{iter}}{\delta} + 1). \tag{21}$$

The speedup can be defined as the computational time of the original LSM algorithm divided by the computational time of the LSM with decimation:

$$\text{Speedup} = \frac{T_{original}}{T_{decimation}} = \frac{2n_{iter} + 1}{\frac{n_{iter}}{\delta} + 1} \tag{22}$$

For a large number of CGNR iterations, the speedup is approaching the decimation factor 2δ.

Using the VCRS format for the Helmholtz solver gives additional speedup of 4 compared to the standard CSR matrix format. If a GPU is used to improve the performance of the preconditioned Helmholtz solver, then the total speedup can be increased approximately by another factor 3. This brings the total speedup of the decimated LSM to the value of 24δ. In the case $\delta = 10$, the LSM with the above

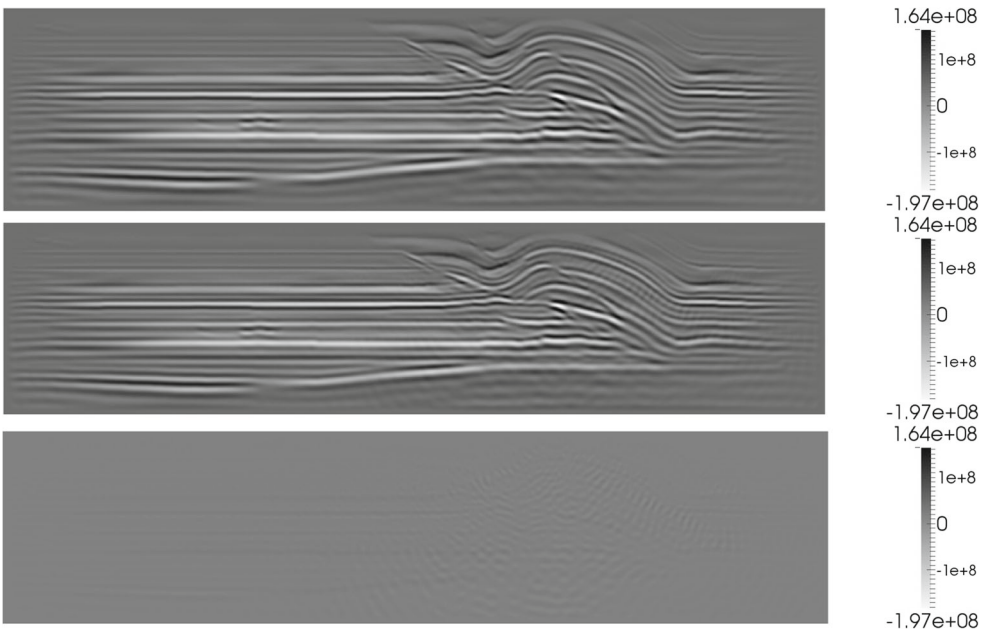


Fig. 10 The second iteration of LSM for the original Overthrust velocity model without decimation (*top*), with decimation $\delta = 10$ (*center*) and the difference between them (*bottom*)

improvements is about 240 times faster than the original algorithm.

7 Conclusions

In this work, we presented an efficient least-squares migration algorithm using several improvements.

Firstly, a decimation was done over sources and frequencies to take advantage of the redundant information present in the data during the CGNR iterations, which is used to solve the optimization problem within the LSM framework. This leads to a speedup of the LSM algorithm by the decimation parameter, whereas the impact is kept minimal.

Secondly, we introduced a VCRS format. The VCRS format not only reduces the size of the stored matrix by a certain factor but also increases the efficiency of the matrix-vector computations. We have investigated the lossless and lossy compression and shown that with the proper choice of the compression parameters the effect of the lossy compression is minimal on the Helmholtz solver which is the Bi-CGSTAB method preconditioned with the shifted Laplace matrix-dependent multigrid method. Also, we applied the VCRS format to a problem arising from a different application area and showed that the compression may be useful in this case as well.

Moreover, using VCRS allows to accelerate the least-squares migration engine by GPUs. A GPU can be used as an accelerator, in which case the data is partially transferred to a GPU to execute a set of operations, or as a replacement, in which case the complete data is stored in the GPU memory. We have demonstrated that using GPU as a replacement leads to higher speedups and allows to use larger problem sizes than when used as an accelerator. In both cases, the speedup is higher than for the standard CSR matrix format.

Summarizing the effects of each used improvement, it has been shown that the resulting speedup can be at least an order of magnitude compared to the original LSM method, depending on the decimation parameter.

Appendix

The Little Green Machine configuration consists of the following nodes interconnected by 40 Gbps Infiniband:

- 1 head node
  - 2 Intel hexacore X5650
  - 24 GB memory
  - 24 TB disk (RAID)
- 1 large RAM node
  - 2 Intel quadcore E5620
  - 96 GB memory
  - 8 TB disk
  - 2 NVIDIA C2070
- 1 secondary Tesla node
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 8 TB disk
  - 2 NVIDIA GTX480
- 1 test node
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 2 TB disk
  - 1 NVIDIA C2050
  - 1 NVIDIA GTX480
- 20 LGM general computing nodes
  - 2 Intel quadcore E5620
  - 24 GB memory
  - 2 TB disk
  - 2 NVIDIA GTX480

References

1. Aminzadeh, F., Brac, J., Kunz, T.: 3-D Salt and Overthrust Models. Society of Exploration Geophysicists, Tulsa (1997)
2. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia (2000)
3. Brackenridge, K.: Multigrid and cyclic reduction applied to the Helmholtz equation. In: Melson, N.D., Manteufel, T.A., McCormick, S.F. (eds.) 6th Cooper Mountain Conf. on Multigrid Methods, pp. 31–41 (1993)
4. Chen, Z., Huan, G., Ma, Y.: Computational Methods for Multiphase Flows in Porous Media. Society for Industrial and Applied Mathematics, Philadelphia (2006). http://opac.inria.fr/record=b1120110
5. Engquist, B., Majda, A.: Absorbing boundary conditions for numerical simulation of waves. Math. Comput. **31**, 629–651 (1977)
6. Erlangga, Y.A., Vuik, C., Oosterlee, C.W.: On a class of preconditioners for solving the discrete Helmholtz equation. In: Cohen, G., Heikkola, E., Joly, P., Neittaanmakki, P. (eds.) Mathematical and Numerical Aspects of Wave Propagation, pp. 788–793. Univ Jyväskylä, Finnland (2003)
7. Erlangga, Y.A., Oosterlee, C.W., Vuik, C.: A novel multigrid based preconditioner for heterogeneous Helmholtz problems. SIAM J. Sci. Comput. **27**, 1471–1492 (2006)

8. Gersho, A., Grey, R.M.: Vector Quantization and Signal Compression. Springer Science+Business Media (1992). doi:10.1007/978-1-4615-3626-
9. van Gijzen, M.B., Erlangga, Y.A., Vuik, C.: Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplace. SIAM J. Sci. Comput. **29**, 1942–1958 (2007)
10. Gozani, J., Nachshon, A., Turkel, E.: Conjugate gradient coupled with multigrid for an indefinite problem. In: Vichnevestsky R, Tepelman, R.S. (eds.) Advances in Computational Methods for PDEs V, pp. 425–427. IMACS, New Brunswick (1984)
11. Guitton, A., Diaz, E.: Attenuating crosstalk noise with simultaneous source full waveform inversion. Geophys. Prosp. **60**, 759–768 (2012). doi:10.1111/j.1365-2478.2011.01023.x
12. Kechroud, R., Soulaimani, A., Saad, Y., Gowda, S.: Preconditioning techniques for the solution of the Helmholtz equation by the finite element method. Math. Comput. Simul. **65**(4-5), 303–321 (2004). doi:10.1016/j.matcom.2004.01.004
13. Khronos Group (2014) www.khronos.org
14. Kim, Y., Min, D.J., Shin, C.: Frequency-domain reverse-time migration with source estimation. Geophysics **76**(2), S41–S49 (2011)
15. Knibbe, H., Oosterlee, C.W., Vuik, C.: GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. J. Comput. Appl. Math. **236**, 281–293 (2011). doi:10.1016/j.cam.2011.07.021
16. Knibbe, H., Vuik, C., Oosterlee, C.W.: 3D Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method on multi-GPUs. In: Cangiani, A., Davidchack, R.L., Georgoulis, E., Gorban, A.N., Levesley, J., Tretyakov, M.V. (eds.) Proceedings of ENUMATH 2011, the 9th European Conference on Numerical Mathematics and Advanced Applications, Leicester, pp. 653–661. Springer-Verlag, Berlin Heidelberg (2011)
17. Knibbe, H., Mulder, W.A., Oosterlee, C.W., Vuik, C.: Closing the performance gap between an iterative frequency-domain solver and an explicit time-domain scheme for 3-d migration on parallel architectures. Geophysics **79**, 47–61 (2014)
18. Kourtis, K., Goumas, G., Koziris, N.: Optimizing sparse matrix-vector multiplication using index and value compression. In: Proceedings of the 5th Conference on Computing Frontiers CF '08, pp. 87–96. ACM, New York (2008)
19. Laird, A.L., Giles, M.B.: Preconditioned iterative solution of the 2D Helmholtz equation. Tech. Rep. 02/12, Oxford Computing Laboratory, Oxford, UK (2002)
20. LGM (2012) The Little Green Machine: Massive many-core supercomputer at low environmental cost. http://www.littlegreenmachine.org
21. Mulder, W.A., Plessix, R.E.: How to choose a subset of frequencies in frequency-domain finite-difference migration. Geophys. J. Int. **158**(3), 801–812 (2004). doi:10.1111/j.1365-246X.2004.02336.x
22. Nemeth, T., Wu, C., Schuster, G.T.: Least-squares migration of incomplete reflection data. Geophysics **64**(1), 208–221 (1999)
23. Plessix, R.E., Mulder, W.A.: Frequency-domain finite-difference amplitude-preserving migration. Geophys. J. Int. **157**, 975–987 (2004)
24. Ren, H., Wang, H., Chen, S.: Least-squares reverse time migration in frequency domain using the adjoint-state method. J. Geophys. Eng. **10**(3), 035, 002 (2013). http://stacks.iop.org/1742-2140/10/i=3/a=035002
25. Riyanti, C.D., Kononov, A., Erlangga, Y.A., Vuik, C., Oosterlee, C.W., Plessix, R.E., Mulder, W.A.: A parallel multigrid-based preconditioner for the 3D heterogeneous high-frequency Helmholtz equation. J. Comput. Phys. **224**(1), 431–448 (2007). doi:10.1016/j.jcp.2007.03.033
26. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)
27. Schuster, G.T.: Least-squares crosswell migration. In: SEG Expanded Abstracts 12, 63 Annual International Meeting, pp. 25–28 (1993)
28. Stüben, K., Trottenberg, U.: Multigrid methods: Fundamental algorithms, model problem analysis and applications. In: Hackbush, W., Trottenberg, U. (eds.) Lecture Notes in Math, vol. 960, pp. 1–176 (1982)
29. Tang, Y.: Wave-equation Hessian by phase encoding. In: 78 Annual International Meeting, SEG, Expanded Abstracts, vol. 27, pp. 2201–2205 (2008)
30. Trottenberg, U., Oosterlee, C.W., Schüller, A.: Multigrid. Academic Press, New York (2001)
31. Turkel, E.: Numerical methods and nature. J. Sci. Comput. **28**, 549–570 (2006)
32. Wei, D., Schuster, G.T.: Least-squares migration of multisource data with a deblurring filter. Geophysics **76**(5), R135–R146 (2011)
33. Wienands, R., Oosterlee, C.W.: On three-grid Fourier analysis of multigrid. SIAM J. Sci. Comp. **23**, 651–671 (2001)
34. Zhebel, E.: A multigrid method with matrix-dependent transfer operators for 3D diffusion problems with jump coefficients, PhD thesis, Technical University Bergakademie Freiberg, Germany (2006)