Full-waveform inversion on heterogeneous HPC systems

Alexey Gokhberg, Andreas Fichtner

Department of Earth Sciences, ETH Zurich, Zurich, Switzerland

A R T I C L E   I N F O

Article history:
Received 17 March 2015
Received in revised form 25 November 2015
Accepted 16 December 2015
Available online 28 December 2015

Keywords:
Seismic tomography
Heterogeneous HPC systems
Wave propagation
Full-waveform inversion
Inverse theory
Spectral-element method

A B S T R A C T

We present a spectral-element implementation of full seismic waveform inversion for large heterogeneous HPC systems. In this we address the optimal parallelisation configurations of individual simulations, the large I/O requirements of adjoint simulations, and the scheduling of large numbers of forward and adjoint solves, typical for realistic inversions. Using GPU accelerators allows us to achieve a 3.5–4 times performance improvement over the best known homogeneous implementation. We achieve GPU memory throughput varying from 60 to 80% of the peak bandwidth, thus providing good utilisation of hardware resources. We demonstrate the practical applicability of our developments in a real-data application in the western Mediterranean. With the help of GPU accelerators, we are able to model and invert seismic wave propagation in a frequency band that is broader than permitted by the use of CPUs alone, which helps bridging the traditional gap between crustal and mantle tomography.

1. Introduction

Seismic tomography is in a transition period where methods based on simplified wave propagation physics are successively replaced by fully numerical approaches that allow seismologists to exploit the complete seismic recordings for the benefit of improved tomographic resolution, provided that data quality is sufficient. Conceptualised in the late 1970s and early 1980s (Bamberger et al., 1977, 1982; Lailly et al., 1983; Gauthier et al., 1986) such full-waveform inversions have become applicable to today’s large seismic data sets thanks to steadily increasing computational resources. Applications at local scales in seismic exploration (Pratt et al., 1998; Brenders and Pratt, 2007; Smithyman et al., 2009; Afanasiev et al., 2015), on regional scales (Chen et al., 2007; Tape et al., 2010; Colli et al., 2013; Rickers et al., 2013; Fichtner et al., 2013b; Zhu et al., 2015), and for the entire Globe (Bozdag et al., 2014; Afanasiev et al., 2016) provide new insight into the structure, dynamics and evolution of the Earth. While 3D inversion on standard CPU clusters is still a major computational challenge taken up by few research groups, the emergence of large heterogeneous high-performance computing (HPC) systems has the potential to transform full-waveform inversion into a widely used routine method.

Recognising the potential of heterogeneous HPC systems, several forward wave propagation codes have been implemented on GPU graphic cards (Michea and Komatitsch, 2010; Komatitsch et al., 2010; Komatitsch, 2011), showing promising results. The next step, that is the transition from the forward modelling of a single source (e.g. an earthquake or explosion) to the solution of a seismic inverse problem, is complicated for two reasons: (i) adjoint calculations needed to compute gradients (Tarantola, 1984, 1988; Tromp et al., 2005; Fichtner et al., 2006; Chen, 2011) require that very large amounts of data are stored and read during the simulations. This is in conflict with GPUs that require minimal I/O for optimal use. (ii) Acceptable tomographic resolution can only be achieved when data from many sources, often hundreds to thousands, are used. Therefore, individual simulations must be set up and scheduled such that heterogeneous HPC systems are optimally harnessed. Running simulations sequentially on the largest possible number of nodes is mostly inefficient.

In the following sections, we address these issues of I/O in adjoint simulations, parallelisation setup and scheduling in the context of large heterogeneous HPC systems. In addition to purely computational aspects, we present a real-data full-waveform inversion of the Mediterranean region that underlines the applicability of our developments.

This paper is organised as follows: in Section 2 we provide a condensed review of seismic wave propagation modelling, including the governing equations of forward and adjoint simulations. A specific spectral-element implementation of these equations is presented in Section 3. Following a brief description in Section 4 of the HPC system used for our developments, Section 5 is dedicated to software engineering challenges, optimisation and parallelisation strategies, and the application architecture. A performance analysis and a real-data application are presented in

© 2015 Elsevier Ltd. All rights reserved.
Sections 6 and 7, respectively.

2. Forward and inverse modelling of seismic wave propagation

To set the stage for our computational developments, we provide a condensed review of forward and adjoint seismic modelling, and the spectral-element discretisation of the seismic wave equation. Additional details may be found in Tarantola (1984, 1988), Komatitsch and Vilotte (1998), Komatitsch and Tromp (2002), Tromp et al. (2005), Fichtner et al. (2006), and Fichtner (2010).

2.1. Forward modelling in visco-elastic anisotropic media

The propagation of seismic waves through the Earth is governed by the momentum-balance equation

\[ \rho(x)\ddot{u}(x, t) - \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \sigma_{ji}(x, t) = f_j(x, t), \quad x \in \Omega, t \in [0, \infty) \]

where \( \rho > 0 \) denotes mass density, \( \sigma_{ji} \) the symmetric stress tensor, \( f_j \) an external force density, and \( u_i \) the seismic waveform (Dahlen and Tromp, 1998; Kennett, 2001; Aki and Richards, 2002). Over-dots symbolise differentiation in time. The stress tensor and the wavefield are subject to the boundary and initial conditions

\[ \sum_{j=1}^{3} \sigma_{ji}\nu_j = 0, \quad u_{t\nu_j=0} = 0, \quad u_{t\nu_j=0} = 0, \]

meaning that stresses in the direction of the surface normal \( \nu_j \) vanish along the boundary \( \partial \Omega \), and that the wavefield is at rest for time \( t=0 \). The stress tensor \( \sigma_{ji} \) is related to the gradients of the wavefield via the visco-elastic stress–strain relation

\[ \sigma_{ji}(x, t) = \frac{\partial}{\partial x_j} \sum_{k=1}^{3} C_{ijkl}(x, t) \frac{\partial}{\partial x_l} u_i(x, t), \quad i = 1, 2, 3, \]

where the elastic tensor \( C_{ijkl} \) satisfies the symmetry relations

\[ C_{ijkl} = C_{klij} = C_{ijlk}. \]

The time-dependence of \( C_{ijkl} \) controls the visco-elastic dissipation of the material, i.e. the conversion of elastic energy into heat as a function of frequency or time. To circumvent the numerically expensive solution of the convolution integral (3), the elastic tensor can be written as a superposition of \( N \) standard-linear solids:

\[ C_{ijkl}(x, t) = \sum_{k=1}^{N} D_k(x) e^{-\tau_k t / \rho_k} \]

(4)

The relaxed moduli \( C_{ijkl}^{r} = \lim_{\tau \rightarrow \infty} \frac{\partial}{\partial x_j} C_{ijkl} \) and the relaxation parameters \( \tau_{ijkl} \) and \( \tau_p \) are chosen to match a prescribed dissipation of the material (Emmerich and Korn, 1987; Robertsson et al., 1994; Blanch et al., 1995; Fichtner and van Driel, 2014; van Driel and Nissen-Meyer, 2014). Introducing (4) in (3) transforms the stress–strain relation into

\[ \sigma_{ji}(x, t) = \frac{\partial}{\partial x_j} \sum_{k=1}^{N} C_{ijkl}^{r}(x) \left[ 1 + \tau_{ijkl}(x) \sum_{p=1}^{N} D_p(x) e^{-t / \tau_p} \right] \frac{\partial}{\partial x_l} u_i(x, t) + \sum_{k=1}^{3} C_{ijkl}^{s}(x) u_{ij}(x, t) \sum_{p=1}^{N} M_{p,kl}(x, t), \]

with \( s = \sum_{p=1}^{N} D_p \). The memory variables \( M_{p,kl} \) satisfy the ordinary differential equation

\[ M_{p,kl} = -\frac{1}{\tau_p} M_{p,kl} + D_p \delta_{kl} u_i. \]

Together, Eqs. (1), (5) and (6) constitute the set of differential equations that must be solved numerically for the seismic wavefield \( u_i(x, t) \), subject to the boundary and initial conditions (2).

2.2. Adjoint modelling and full waveform inversion

The solution of a seismic inverse problem rests on the quantification of a misfit \( J \) between waveforms \( u^{\text{obs}}(x, t) \) observed in a time interval \( [0, T] \) at receiver positions \( x_{r1}, \ldots, x_{rR} \) and the corresponding synthetic waveforms \( u_i(x, t, \mathbf{m}) \) computed for an Earth model \( \mathbf{m} = (\rho, C_{ijkl}) \). The specific form of the misfit functional \( J(\mathbf{m}) \) depends on the concrete application and on the nature of the data (Crase et al., 1990; Luo and Schuster, 1991; Gee and Jordan, 1992; Fichtner et al., 2008; van Leeuwen and Mulder, 2010; Brossier et al., 2009, 2010). To minimise \( J \) with respect to \( \mathbf{m} \) using gradient-based methods, we require the variation of \( J \), which can be written in terms of a Fréchet or sensitivity kernel \( \mathbf{K} \):

\[ \delta J = \int_{0}^{T} \mathbf{K}(x, t) \delta m(x) \, dx. \]

(7)

For the parameters \( \rho \) and \( C_{ijkl} \), the Fréchet kernels take the specific forms

\[ K_\rho(x) = -\sum_{i=1}^{3} \int_{0}^{T} \dot{u}_i(x, t) \delta u_i(x, t) \, dt, \quad K_{C_{ijkl}}(x) = \int_{0}^{T} \delta u_i(x, t) \delta u_i(x, t) \, dt, \]

(8)

respectively (Tarantola, 1984, 1988; Tromp et al., 2005; Fichtner et al., 2006; Chen, 2011). The adjoint wavefield \( u_i^*(x, t) \) is governed by the following set of equations: (i) the adjoint momentum-balance equation

\[ \rho(x)\ddot{u}_i^*(x, t) - \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \sigma_{ji}^*(x, t) = f_j^*(x, t), \quad x \in \Omega, t \in [0, \infty), \quad i = 1, 2, 3, \]

(9)

(ii) the adjoint stress–strain relation that couples the adjoint stress tensor \( \sigma_{ji}^* \) to the adjoint field \( u_i^* \),

\[ \sigma_{ji}^*(x, t) = \sum_{k=1}^{3} \int_{t-\tau}^{t} C_{ijkl}^r(x, t' - \tau') \delta u_i(x, t') \, dt', \]

(10)

and (iii) the adjoint subsidiary conditions

\[ \sum_{j=1}^{3} \sigma_{ji}^*\nu_j = 0, \quad u_i^*|_{t=0} = 0, \quad u_i^*|_{t=T} = 0. \]

(11)

The adjoint source \( f_j^* \) is uniquely determined by the misfit functional \( J \), and so are the adjoint wavefield and the Fréchet kernels.

The terminal conditions on the adjoint field (11) impose that the adjoint equations must be solved in reverse time. Consequently, the forward field \( u_i \) must be made available during the adjoint equation solve in order to compute the time integrals in (8) on the fly. To achieve this task in the hypothetical absence of visco-elastic dissipation, the wavefield from the end of the forward simulation can be propagated backwards in time, therefore not requiring significant storage capacities. In realistic dissipative media, backward propagation is numerically unstable, and the forward field must be stored at sufficiently many time steps to enable efficient checkpointing (Griewank and Walther, 2000). The massive I/O related to reading and writing snapshots of the forward field is a major challenge on modern HPC systems that we address in this paper.
2.3. Numerical solution

Depending on the particular application, various techniques have been developed to solve the seismic wave equation numerically. These include finite-difference (Alterman and Karal, 1968; Boore, 1970; Moczo et al., 2002; Bohlen, 2002; Moczo et al., 2014), spectral-element (Seriani and Priolo, 1994; Faccioli et al., 1997; Komatitsch and Vilotte, 1998; Komatitsch and Tromp, 2002; Nissen-Meyer et al., 2014), and discontinuous Galerkin (de la Puente et al., 2007; Dumbser et al., 2007a,b) methods. Regardless of the specific method, the space-discretised equations can generally be cast into the generic form

$$\sum_j M_{ij} u_j + \sum_j K_{ij} u_j = f_j,$$  

with the discretised wavefield $u_j$, the mass matrix $M_{ij}$, the stiffness matrix $K_{ij}$, and the discretised external force $f_j$. In our application we obtain (12) through a spectral-element discretisation of the elastic wave equation. For this, the computational domain is subdivided into non-overlapping elements $C_e$ that are mapped to the unit cube $\Lambda$ (Fig. 1). In $\Lambda$, material parameters and the wavefield are approximated by Lagrange polynomials, where the GLL points serve as interpolation points. GLL quadrature then leads to a discrete system with diagonal mass matrix that allows for efficient explicit time stepping.

Despite the specific choice of a forward problem solver, the following computational developments are equally applicable to time-domain forward and adjoint wave equation solvers where the discrete system is the same as in Eq. (12).

3. SES3D: spectral-element visco-elastic wave propagation in a 3D spherical section

We have implemented the computational approach described above in a software package named SES3D. SES3D operates in the natural spherical coordinates, which is untypical for spectral-element approaches. The advantages are a compact programme code, fast computations for spherical sections that are sufficiently far from the poles and the core, and the easy implementation of 3D models. A single run of SES3D performs either a forward simulation with or without storage of the wavefield, or an adjoint simulation where a previously stored forward field is read and combined with the adjoint field to produce sensitivity kernels. Each run, forward or adjoint, corresponds to an earthquake or other seismic event that is processed independently.

SES3D data are represented as 3D arrays mostly falling into the following categories: (i) physical model parameters including elastic tensor components $C_{ijkl}$, density $\rho$, and relaxation parameters $\tau_{ijkl}$, (ii) source fields for moment tensor and vector forces, including adjoint sources, (iii) snapshots of the forward field required for adjoint calculations, (iv) static fields such as the mass matrix, and (v) dynamics fields that are recomputed in every iteration, including velocities $u_j$, velocity gradients $\phi_j u_i$, stresses $\sigma_{ij}$ and stress rates $\dot{\sigma}_{ij}$. Among these fields the displacements and stress rates are accumulated (integrated) during the entire iterative process.

The time step algorithm exposes a large amount of data-level parallelism that can be harnessed by computing systems with multiple independent processing elements. For this, the computational domain is partitioned into a fixed number of sub-domains, each being assigned to one processing element. Information between elements is exchanged when stress rates at element boundaries are shared between neighbouring sub-domains. If processing elements themselves provide hardware facilities for parallel computations, they can exploit the algorithm data parallelism by performing parallel computations of field values for different grid points within a sub-domain. This is exactly the case for heterogeneous massively parallel computing systems, which we use in the present work.

4. Instrument: heterogeneous massively parallel computing systems

In recent years, the combination of GPU-accelerated computing with more traditional ideas of massively parallel systems, consisting of multiple nodes with distributed memory, led to the introduction of heterogeneous high-performance systems. Such systems consist of large numbers of nodes interconnected via high throughput networks, where each node provides a mix of processing elements with different architectures, like a small number of sequential CPU cores and one or a few GPU accelerators. These emerging systems provide an opportunity for manifold application performance increase and are more energy-efficient. However, since they have higher hardware complexity and are more difficult to program, the transition to heterogeneous computing is highly disruptive from the software engineering perspective. Efficient use of GPU resources requires a programming methodology that is completely different from those used for multiple sequential processors running in parallel, typical for the recent decades. Thus, to exploit potential benefits of heterogeneous computing, novel programming methodologies must be created.

A typical example for massively parallel heterogeneous high-performance computing systems is “Piz Daint”, a supercomputer of the Cray XC 30 family operated by the Swiss National Supercomputing Center (CSCS). It features 5272 computing nodes, each being equipped with an 8-core 64-bit Intel Sandy Bridge CPU (Intel Xeon E5-2670), an NVIDIA Tesla K20x GPU accelerator with 6 GB GDDR5 memory, and 32 GB of host memory. The nodes are connected by the proprietary “Aries” interconnect from Cray. In this study, we aimed at the implementation of SES3D on “Piz Daint”.

![Fig. 1. Schematic illustration of the mapping between the unit cube $\Lambda$ and the physical-domain element $G_e$, which is a small spherical-subsection of the Earth.](image-url)
5. Methodology: software engineering challenges

5.1. Need for a new software engineering methodology

The proposed implementation presented us with various engineering challenges arising from a need to split the code into two distinct parts running on the CPU and GPU devices that require different techniques for programming and performance optimisation. Furthermore, special attention had to be paid to work distribution and interaction between CPU and GPU, especially during I/O intensive adjoint runs.

We based the implementation process on the assumption that the transition to a heterogeneous computing model will have impact on all aspects of the application architecture. This means, in particular, that it would not be sufficient to extract the most critical parts of the code and re-implement them as optimised kernels running on GPUs. Instead, the entire application architecture has to be re-engineered because any inefficiency in the serial part of the application can easily offset the benefits of using GPUs. Therefore, we decided to employ a hybrid programming model based on a combination of several well-established software development technologies, including C as a host programming language, MPI 3.0 as a library for the inter-node communication, and CUDA as a toolkit for programming NVIDIA GPU accelerators. We are using compilers and libraries of Cray Programming Environments 5.1 in combination with NVIDIA CUDA Toolkit 5.5.

5.2. Optimisation methodology

SES3D manages large volumes of data arranged in ~100 arrays. Interchange of these data volumes between GPU memory and CPU memory at every iteration would incur a large performance overhead. Due to the nature of the algorithm, this interchange cannot be done asynchronously. To use GPU capabilities efficiently, it is necessary to host all data arrays on GPU permanently and implement all computations on GPU. The role of the corresponding host CPU should be limited to I/O operations, inter-node communication, and management of the workflow. We thus try to keep data interchange between the GPU and the surrounding environment at the absolutely necessary minimum.

Data interchange between the computing nodes may cause overhead that can offset benefits provided by the GPU. It was therefore essential to tune inter-node communication carefully to keep data interchange between GPU devices of different nodes reasonably fast. Our implementation relies on MPI as a principal communication platform. However, we decided to replace the classic MPI point-to-point communication by one-sided communication based on the MPI Remote Memory Access (RMA) that, according to the results of our empirical study, provide better performance for all parallelisation configurations.

For the optimisation of CUDA kernels, we tuned the layout of data arrays to enable coalesced access to device memory, and we used GPU shared memory to reduce the number of redundant device memory accesses. Whenever feasible, we implemented highly optimised re-usable kernels for commonly used functions like array reductions. We extensively used the NVIDIA profiler `nvprof` to measure performance of individual CUDA kernels and data transfers between CPU hosts and GPU devices.

The algorithm for the forward and adjoint simulations where wavefield snapshots are written and read, respectively, requires a substantial amount of data interchange between the application and external storage. This interchange takes place once per a pre-defined number of iterations, typically 10–30 (Fichtner et al., 2009). As explained in Section 2.2, snapshots of the forward wavefield are written to large external files during the forward simulations; in the adjoint mode they are read from these files in reverse order. The storage requirements for typical applications, such as the one presented in Section 7, are on the order of 1–10 TB.

Adjoint mode I/O operations add penalty to the overall performance. This is tolerable in homogeneous configurations but increases dramatically in heterogeneous cases. This increase occurs because in heterogeneous configurations each GPU device takes over the work that otherwise would be performed by a big number of CPU cores and therefore much larger volumes of data have to be interchanged between each computing node and external storage. Our empirical performance study of interchange with the external storage suggests that the related I/O penalty is tolerable if data are read or written sequentially. Unfortunately, this is not a case in the adjoint mode when data are read in the reverse order. To solve this problem we introduced buffering of read operations in the adjoint mode. For each array to be read the application allocates a buffer large enough to hold approximately 30 consecutive array instances which are pre-fetched via a single read request. This approach allowed us to reduce the run time and make it comparable to that of forward simulations with wavefield storage.

5.3. Choice of parallelisation scheme

Our implementation on massively parallel computing architectures is facilitated by parallelisation configurations that specify 3D processor grids consisting of CPU cores in the homogeneous case or GPU devices (and respective computing nodes) in the heterogeneous case. The whole computational domain is partitioned into sub-domains, each sub-domain being mapped to a single element of the respective processor grid. We make the distinction between “wide” and “compact” parallelisation configurations. Wide configurations are characterised by a relatively large processor grid size and rather small amount of data assigned to each processor grid element. Compact configurations have a substantially smaller grid size and respectively larger size of data sets assigned to each processing element.

The initial implementation of SES3D processed all events sequentially, that is, at each moment all computing nodes were involved in processing of the same event. This sequential scenario appears impractical for the compact parallelisation configurations. These configurations provide superior performance in heterogeneous cases, however, the absolute wall clock time required to process the large number of events in a realistic full-waveform inversion sequentially is too large. On the other hand, the number of computing resources involved in computations for the compact parallelisation configurations is substantially (usually in order of magnitude of more) smaller than in case of wide configurations. (This subject is discussed further in Section 6 dedicated to the application performance.) Therefore, if a comparable number of computing resources is available for both compact and wide configurations, then the compact configuration may utilise these resources in full by splitting them into smaller groups, then distributing all events to be processed evenly among these groups and allowing them to process events in parallel.

We implemented support for parallel event processing in SES3D. If a number of processing elements allocated at application start is a multiple of a number specified by a parallelisation configuration, the application automatically splits available processing elements in groups and partitions a list of events into chunks, assigning each chunk to its own group for parallel processing.

5.4. Application architecture

Each program instance consists of a sequential part running on the CPU core and a massively parallel part running on the GPU
device (see Fig. 2). The sequential part includes modules for application initialisation, inter-node communication, mass storage I/O, and management of control and data flow for operations offload to the GPU. The massively parallel part is represented as a set of CUDA kernels that reside and run on GPU, each kernel implementing a certain computational function applied to a large stream of parallel processing threads. The sequential part orchestrates activation of CUDA kernels. CUDA kernels perform all computations of the SES3D time step algorithm, including the computation of external forces, velocity gradients and stress rates and the time extrapolation of stresses and velocities. Furthermore, there are kernels implementing functionality specific for various computation modes, such as the computation of synthetic seismograms, intermediate wave field snapshots and Fréchet kernels.

All major data sets are replicated between the CPU and GPU memories. The CPU is responsible for initialisation of all data sets at the processing start for each event. These data sets are then copied once to the GPU memory, where they reside until completion of the event processing. Further data interchange between CPU and GPU is kept at minimum and takes place only when inter-node communication or mass storage I/O operation is required. Copying of data between the CPU and GPU is performed using synchronous memory management primitives of CUDA. We keep a separate set of data files per processing element and use plain binary format and standard C I/O libraries. The overall number of files is not performance-critical. We have made experiments keeping values of each parameter for the entire simulation domain in a single file and using the MPI parallel I/O for the shared access, however, this did not provide any performance gain.

Programming instances arranged this way form a gang that performs parallel wave modelling on all sub-domains for a single event. Since usually a single simulation involves tens of events that can be processed independently, a typical application configuration contains multiple gangs, every gang being responsible for processing a distinct part of the entire event set (see Fig. 3). Distribution of work among gangs is done statically at the application initialisation phase based on a simple algorithm aiming at uniform load balancing.

6. Performance

We use a simple yet practical method for performance assessment. The wall clock time for each configuration and test run is measured. Based on this figure, the number of iterations processed per minute per computing node is calculated. While we have chosen several SES3D use cases in the development stage, we limit the following description to a representative scenario featuring a $77 \times 68 \times 77$ grid of blocks, and 10 events of 14,000 iterations each. For this case we used the following parallelisation configurations:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Processor grid</th>
<th>Nodes</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous, compact</td>
<td>$11 \times 1 \times 7$</td>
<td>10</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>Heterogeneous, compact</td>
<td>$11 \times 1 \times 1$</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Homogeneous, wide</td>
<td>$11 \times 17 \times 7$</td>
<td>164</td>
<td>1309</td>
<td></td>
</tr>
<tr>
<td>Heterogeneous, wide</td>
<td>$11 \times 1 \times 11$</td>
<td>121</td>
<td>121</td>
<td>121</td>
</tr>
</tbody>
</table>

We measured performance for both CPU and GPU versions, and for both compact and wide parallelisation configurations. At each run, 10 events were processed in a sequence with 14,000 iterations for each event. We conducted separate tests for each of three supported computation modes: (i) forward simulations without storage of wavefield snapshots, (ii) forward simulations with storage of wavefield snapshots, and (iii) adjoint simulations. The following wall clock times have been recorded in one of the representative test series (results expressed as hours:mins:secs).

Fig. 2. Schematic illustration of the application architecture.

Fig. 3. Schematic illustration of 10 gangs of 11 computing nodes each processing 60 events in parallel.
The following performance figures (iterations/node/minute) have been calculated for forward simulations without wavefield storage:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU, 11 × 1 × 1</td>
<td>140,000 / 11 / 78.5 = 162.1</td>
</tr>
<tr>
<td>CPU, 11 × 1 × 11</td>
<td>140,000 / 121 / 16.5 = 70.1</td>
</tr>
<tr>
<td>CPU, 11 × 17 × 7</td>
<td>140,000 / 10 / 420.5 = 33.3</td>
</tr>
<tr>
<td>CPU, 17 × 7</td>
<td>140,000 / 164 / 21.5 = 39.7</td>
</tr>
</tbody>
</table>

These figures fully support our original assumption stating that wide configurations favour homogeneous (CPU-based) versions since smaller data sets fit CPU caches better. On the other hand, compact configurations provide more benefits for heterogeneous (GPU-based) versions because GPU devices achieve better performance processing reasonably large data sets that suit better for amortisation of latencies in the GPU streaming processors. Furthermore, by comparison of the best performances for CPU and GPU cases, the speedup achieved on the heterogeneous architecture for the given use case can be calculated as 162.1/39.7 = 4.1 times.

There exist certain technical limits for the further performance improvement on the heterogeneous architecture. SES3D is a typical memory-bound application, therefore performance of its CUDA kernels is constrained by the peak memory bandwidth of the GPU device. For NVIDIA K20x this bandwidth equals 250 GB/s. Profiling of our kernels reports the GPU device memory throughput varying between 160 and 200 GB/s, that is, between 60% and 80% of the peak bandwidth.

Performance comparison done for problems with different overall sizes but the same problem size per processing element has demonstrated the weak scaling efficiency of SES3D does not depend significantly on the number of processing elements for configurations consisting of up to a few hundred of computing nodes on “Piz Daint”. The strong scaling efficiency behaves differently for homogeneous and heterogeneous configurations. For homogeneous configurations the strong scaling efficiency is higher for big numbers of processing elements (CPUs), while for heterogeneous configurations it is higher for the smaller number of processing elements (GPUs). Furthermore, simulations with higher frequency resolution will require the proportionally larger number of time steps and grid size in each dimension. Therefore, the computational complexity grows by the factor of $N^4$ with the increase of the simulation frequency by the factor of $N$.

7. Application: full-waveform inversion for the western Mediterranean region

We tested the GPU implementation of SES3D in a full-waveform inversion in the western Mediterranean region. For this, we used 13,089 three-component recordings from 52 regional earthquakes. For broadband stations belonging to permanent national/regional networks we obtained seismic waveform data from global (IRIS, http://www.iris.edu and European (ORFEUS, http://www.orfeus-eu.org) data centres. This was complemented by data from the temporary IberArray (Díaz et al., 2009). The source–receiver distribution is shown in Fig. 4.

We partitioned the computational domain into 1,603,584 elements, within which all fields a represented by tensorised polynomials of degree 4. The total number of grid points was 200,448,000. Taking at least two elements per minimum wavelength to ensure sufficiently accurate solutions, this setup allowed us to model wave propagation at periods as low as 12 s. To optimise GPU usage, each event was processed on only 48 nodes. However, since the 52 earthquakes were subdivided into 26 gangs with 2 earthquakes each, a total of 1248 nodes was used simultaneously. The forward simulation of all earthquakes was typically accomplished within slightly more than 1 h.

For the processing of seismic waveform data and the selection of measurement time windows, we used the Large Scale Seismic Inversion Framework of Krischer et al. (2015). A previous full-waveform inversion for all Eurasia at longer periods (Fichtner et al., 2013a) served as initial model. After 20 conjugate-gradient iterations, the initial misfit was reduced by nearly 80% Fig. 5 shows the final distribution of the isotropic $S$ velocity $v_s$, defined as a weighted average of the velocities of vertically polarised $(v_{sv})$ and horizontally polarised $(v_{sh})$ shear waves: $v_s^2 = \frac{2}{3}v_{sv}^2 + \frac{1}{3}v_{sh}^2$ (Babuska and Cara, 1991; Panning and Romanowicz, 2006).

The final model, discussed in terms of the regional geology in Fichtner and Villasenor (2015), is remarkable in that 3D structure
within both the thin crust at mostly less than 40 km depth and the
underlying mantle can be resolved. This is in contrast to traditional
seismic tomography, where either crust or mantle was fixed, de-
dpending on the specific region of interest. The ability to model and
invert seismic wave propagation at shorter periods with the help
of heterogeneous HPC systems bridges the resolution gap between
crustal and seismic tomography; and it will eventually lead to the
unification of these two seismic imaging branches. The work
presented here constitutes a step in this direction.

8. Discussion

A major challenge in the GPU implementation of full-waveform
inversion is the large I/O caused by the reading and writing of
wavefield snapshots during the simulations. It arises from the
need to have synchronous access to the forward and adjoint fields
in the computation of Fréchet kernels. In media without attenua-
tion, this synchronous access can be achieved by the time reversed
propagation of the forward wavefield from the final state of the
forward simulation to time \( t=0 \) (Rietmann et al., 2012). Since
seismic wavefields in the Earth can be strongly attenuated, re-
verse-time propagation is inherently unstable. The instability can
be avoided by ignoring attenuation in the time reversed simula-
tion, though at the cost of inaccurate kernels that may slow down
or even prevent convergence. To ensure convergence in the pre-
sence of a realistic attenuating Earth, the forward wavefield should
be stored whenever possible.

Using the GPU-based version of SES3D package is not more
difficult than that of the conventional CPU-only version since no
GPU-specific detail is exposed to the end-user and no computer
science background is required for running the application.

However, using the GPU-related technologies for building similar
HPC applications in seismology in general represents a significant
challenge. Transition to heterogeneous architectures has a clear
disruptive effect since it requires a substantial revision of pro-
gramming paradigms and demands higher programming skills
from developers. In particular, we note that substantial code du-
plication took place during the development of SES3D. The same
computation algorithms have been implemented twice, as serial C
code and as code of GPU CUDA kernels. This code duplication is
regarded as a significant disadvantage that cannot be efficiently
addressed within the chosen programming model, which, in our
opinion, leaves substantial room for improvement of programmer
productivity. In the future this improvement might be achieved by
introduction of more advanced programming models that would
replace the present heterogeneous mix of relatively low-level
technologies thus reducing complexity of application code, im-
proving its portability, increasing programmer productivity and
being accessible for developers who are not experts in software
engineering. Acceptability of GPU accelerators for development of
new applications by the wider seismologist community depends
on the availability of such high level programming models and
respective tools. Their creation must involve the multi-disciplinary
collaboration of seismologists and computer scientists.

It must be noted that the waveform simulations in SES3D im-
plement just one of the various performance-critical steps of the
seismic inversion process. Once the substantial performance im-
provement is achieved for the simulation modules, we have to
shift focus to the optimisation of other steps, in particular, com-
putation of misfits and updating the model. We intend to address
these issue at the next stages of our research.

At the high level, our engineering approach is similar to the one
used for the seismic wave propagation package SPECFEM3D
(Rietmann et al., 2012). Like our solution, SPECFEM3D supports large-scale homogeneous and GPU-enabled heterogeneous computers and is implemented using MPI/CUDA. Furthermore, the entire package and not just individual kernels had been ported to CUDA and special attention had been paid to the optimisation of data motions at all levels. Although the authors of SPECFEM3D and SES3D have faced the similar development challenges, in some cases they addressed them in different ways. In particular, in our work we target the issue of updating the shared grid nodes with the purpose built race-free algorithm, thus avoiding the need for using more sophisticated methods like mesh colouring. We have decided not to implement overlapping of data motion for inter-node and host-accelerator transfers because in case of SES3D the volumes of data to be transferred at each iteration are relatively modest and potential performance gains do not justify the respective increase in the code complexity. We employ the various techniques for CUDA optimisations, focusing mainly on mediating the limited global memory throughput by the extensive use of the GPU shared memory; in this respect the SES3D approach apparently differs in detail from that used in SPECFEM3D given the different code structures of two packages. Finally, we note that the parallelisation strategy of SES3D is specifically tailored towards the solution of large inverse problems that involve a large number of events.

9. Conclusions

We present a spectral-element implementation of full seismic waveform inversion for large heterogeneous HPC systems. In our development, we addressed the optimal parallelisation configurations of individual simulations, the large I/O requirements of adjoint simulations, and the scheduling of large numbers of forward and adjoint solves, typical for realistic inversions. We have demonstrated that wide parallelisation configurations are more beneficial for the homogeneous (CPU-only) application version while compact configurations yield better performance in the heterogeneous (GPU-enabled) case.

Adjoint mode I/O operations add a significant penalty to the overall performance of the heterogeneous application version running in compact configurations. We solve this problem by prefetching a reasonably big number of consecutive data records into a large memory buffer via a single I/O request. This approach successfully reduces the run time of adjoint simulations to that of forward simulations. The multi-event processing in the inverse problem solution is most efficient when events are grouped into gangs. While all gangs operate in parallel, events are processed sequentially within each of them. Using GPU accelerators allowed us to achieve 3.5–4 times performance improvement over the best known homogeneous implementation on “Piz Daint”. The new implementation achieves the GPU memory throughput varying from 60% to 80% of the peak bandwidth, thus providing good utilisation of hardware resources.

We demonstrated the practical applicability of our developments in a real-data application in the western Mediterranean. With the help of GPU accelerators, we were able to model and invert seismic wave propagation at periods as short as 12 s over ~2000 km distance, thereby bridging the resolution gap between crustal and seismic tomography.

Acknowledgements

We are grateful to Michele De Lorenzi and Thomas Schulthess for their continuing support and encouragement. Yann Capdeville and an anonymous reviewer provided numerous constructive comments that helped us to improve the manuscript. Our developments would not have been possible without the enthusiasm of our power user Saule Simute. This research was funded by the Swiss National Supercomputing Center (CSCS) in the form of the GeoScale and CH1 projects, by the Swiss National Science Foundation (SNF) under Grant 200021_149143, and by the Netherlands Organisation for Scientific Research (VIDI Grant 864.11.008). The SES3D program package is available from the authors upon request or from the website of the ETH Computational Seismology Group, cos.ethz.ch.

References

Nissen-Meyer, T., van Driel, M., Stähler, S., Hosseini, K., Hempel, S., Auer, L., Four-