Optimizing the computational performance of time-domain modelling—leveraging multiple right-hand-sides

Mathias Louboutin[1]*, Gerard Gorman[2] and Felix J. Herrmann[1]
[1]Seismic Laboratory for Imaging and Modeling (SLIM), University of British Columbia
[2]Department of Earth Science & Engineering, Imperial college

Abstract

Exploration geophysics heavily relies upon fast solvers for the wave-equation and its adjoint. The main computational cost of a wave-equation solver is to compute the Laplacian, or more complex finite-difference operators, at every time step. The performance of many discretizations is limited by the relatively low operational intensity (number of floating point operations divided by memory traffic) of the finite-difference stencil. Solving the wave-equation for multiple sources/right-hand-sides (RHSs) at once mitigates this problem by increasing the operational intensity. This is implemented by rewriting the classical matrix-vector product into a matrix-matrix product where each column of the second matrix represent the solution wavefield for each given source. This minor modification to the solver is shown to achieve a $2-4$ times speedup compared to a single source solver. We concentrate in this paper on acoustic modelling, but our approach can easily be extended to anisotropic or elastic cases for both forward and adjoint modelling.

1 Introduction

The computational cost of time-domain inversion workflow in seismic exploration is nearly entirely contained within the wave-equation solves cost. Even though the time-marching structure of the discretized wave-equation allows relatively straightforward implementations, problem sizes of practical interest (e.g. over 400^3 grid points) require highly optimized propagators. For this reason a diverse range of solvers has emerged, focusing on issues such as: the trade-off between the completeness of the physics being modelled and the cost of computing that solution (e.g. many more degrees of freedom are required to model elastic waves due to the much smaller wavelength of shear waves); wide range of different numerical discretizations; and software implementations with consideration to different parallel programming models and computer architectures. Often the result is difficult to develop and maintain software that lacks performance portability, and that restricts innovation on the level of numerical methods and inversion algorithms.

Finite-difference schemes generally used in time-stepping codes are characterized as structure grid stencil codes, and typically have a very low operational intensity (OI; number of floating point operations divided by memory traffic) [Asanovic et al., 2006, Colella, 2004], that is to say that the performance of the solver is limited by the memory latency. The roofline model [Williams et al., 2009] provides a perfect visual tool to compare the maximum achievable performance for a given OI and architecture and illustrates this limitation. It is also for this reason that SIMD vectorization has limited impact on the performance of the code as the CPU cores still have to wait for data to be transferred from memory. While parallelizing the code will ensure that all the available memory bandwidth and cores are utilized it does not overcome the fundamental performance restriction related to the OI of the computational kernel.

To address the performance issues with stencils, polyhedral programming methods such as tiling [Kamil et al., 2006] and vector folding

[Yount, 2015] have been developed to reduce the number of cache misses in practice. While these methods can increase the code's performance, the changes to the software can be highly invasive. What we propose here is rather than computing the solution for a single shot per compute node, we instead compute the solution for multiple shots simultaneously, whereby the multiple solutions are stored continuously at the grid points. By choosing an appropriate number of shots this modification changes the computational kernel from being a stencil acting on scalar values to being a stencil acting on SIMD vectors.

While this approach is straightforward to implement, we show that we can readily double the performance of the solver. In this case study, we used a high-level programming language (MATLAB) interspersed with C functions that implement the time-stepping for multiple sources. OpenMP is used to parallelize the solver over space. The inner loop is over a fixed number of shots, therefore the compiler can readily vectorize it.

2 Seismic modelling

2.1 Single source modelling

For a spatially varying velocity model, c, the acoustic wave-equation in the time domain is given by:

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \nabla^2 u = q, \tag{1}$$

where u is the wavefield, q is the source, ∇^2 is the Laplacian and $\frac{\partial^2}{\partial t^2}$ is the second-order time derivative. The aim is to model the acoustic wavefield and to record it at the receiver locations:

$$\begin{aligned} \mathbf{d}_{syn} &= \mathbf{P}_r \mathbf{u}, \\ \mathbf{A}(\mathbf{m})\mathbf{u} &= \mathbf{q}_s, \end{aligned} \tag{2}$$

where $\mathbf{P}_r$ is the projection operator onto the receiver locations, $\mathbf{A}(\mathbf{m})$ is the discretized wave-equation matrix, $\mathbf{m}$ is the set of model parameters (for this simple acoustic case $\mathbf{m} := \frac{1}{\mathbf{c}^2}$), $\mathbf{u}$ is the discrete synthetic pressure wavefield, $\mathbf{q}_s$ is the corresponding source.

2.2 Matrix-vector to matrix-matrix products

For n_{RHS} multiple RHSs, we can rewrite the wave-equation as

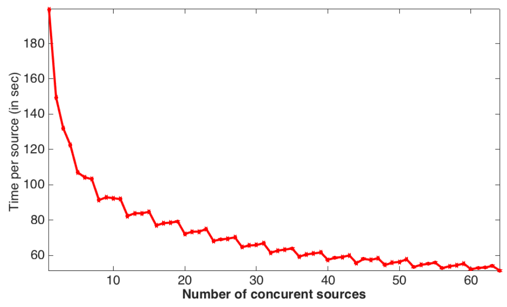
$$A(\mathbf{m})\mathbf{U} = \mathbf{Q}, \tag{3}$$

where $\mathbf{U}$ is now a matrix with one wavefield for each source per column with the corresponding source located in the same column in $\mathbf{Q}$. This is implemented by arranging the wavefield matrix $\mathbf{U}$ as a flat $n_{RHS} \times N$ matrix while making sure that every column is memory aligned and contiguous so the compiler can readily vectorize the code. This approach can be easily implemented for arbitrary time-stepping codes by adding an additional inner loop, which applies the stencil to a SIMD vector instead of a single scalar for each time step.

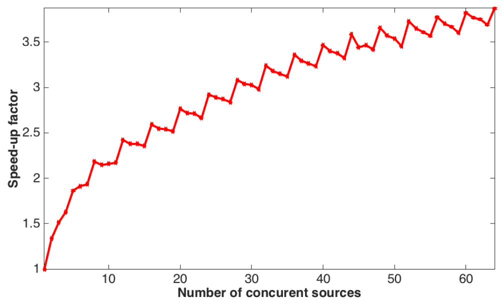
3 Computational efficiency

3.1 Number of sources

To evaluate the performance of our implementation, we start by measuring the relative speed up compared to modellings of one single RHS of our implementation designed to work with multiple RHSs. For this purpose, we use a 10 Hz Ricker wavelet as a source and generate 4 s of data. The velocity model is a two layer cube of size 4.75 km discretized with a 19 m grid in every direction (250^3 grid points). We simulate 1 to 64 sources concurrently and we look at the computation time per individual source by dividing the simulation time for the concurrent simulations by the number of concurrently computed sources. All experiments ran with the same computational resources on a 20 cores CPU. We can see from Figure 1 that we obtained a speedup of a factor of 3.8 between operating on a single source and 64 sources. Even for 20 concurrent sources, we already have a speedup by 2.5. We observe a staircase behavior where the performance increases every four sources due to non aligned access.



(a) Runtime for different number of concurrent RHSs.



(b) Speedup factor for different number of RHSs compared to a single source run.

Figure 1: Single source versus multi-source speedup factor for a fixed model size.

3.2 Model size

Now that we know that our method improve the performances as a function of the number of RHSs for a fixed domain size, we ensure this behavior scales with the size of the model. For this purpose, we fix the number of RHSs to 20 as this number already gives a significant speedup. This choice also allows us to go to relatively large models given our memory of 256 GB of RAM. For a fixed source and receiver setup, we increase the model size from 20^3 to 700^3 including an absorbing boundary layer of 40 grid points. The effective computational domain ranges from 100^3 to 780^3 grid points (2 km cube to 15 km cube with a 19 m grid at 10 Hz). The results are presented in Figure 2, which shows the simulation times for 4 s recordings of 3D shots as the model size increases. From this figure, we observe that the computational times for our multiple RHS approach grow moderately fast as a function of the model size.

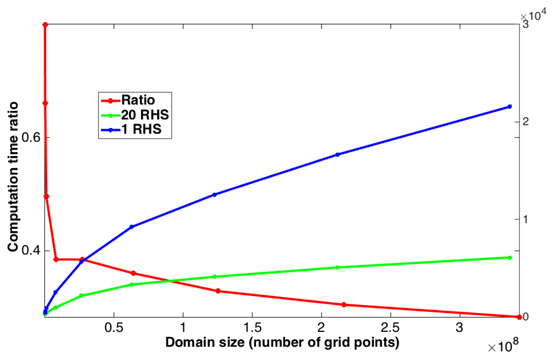


Figure 2: Single source versus multi-source timing comparison for increasing model sizes.

4 Roofline analysis

We previously derived estimates of the maximum achievable performance for an acoustic modelling kernel [Louboutin et al., 2016]. The method presented can be extended to multiple RHSs analysis to support our results. The paper shows that the operational intensity for a single source is given by

$$\mathcal{I}_{alg}(k) = \frac{3k}{8} + \frac{1}{4}, \tag{4}$$

where k is the stencil size (space order+1). From the derivation of this expression, we can rewrite the equivalent expression for a given number n_{RHS} of RHSs as:

$$\mathcal{I}^s_{alg}(k) = \frac{n_{RHS}(3k+2)}{2(3n_{RHS}+1)} = \frac{3ks + 2n_{RHS}}{6n_{RHS}+2}, \tag{5}$$

and we can easily verify that we obtain the same operational intensity for $n_{RHS} = 1$. In the previous examples we used a 4th order discretization in space ($k = 5$). Using equation 5 we show in the next table the operational intensity for different number of RHS compared to a single source solver. We obtain a 30% increase of the OI with only 16 RHSs leading to at least 30% performance improvement for a fixed implementation (the roofline model shows that performance is linear as a function of the OI [Asanovic et al., 2006]). The SIMD instruction over the RHSs gives us the extra improvement.

RHS	$I^s_{alg}(5)$	$I_{alg}(5)$	ratio	% improvement
1	2.1250	2.1250	1	0%
2	2.4286	2.1250	1.1429	15%
4	2.6154	2.1250	1.2308	23%
8	2.72	2.1250	1.28	28%
16	2.7755	2.1250	1.3061	30%
32	2.8041	2.1250	1.3196	32%

Table 1: Operational intensity for different number of RHS.

5 Conclusions

The balance between productivity and performance is a key concern in rapidly evolving fields such as geophysical inverse problems. This is even more pronounced for any gradient based method where both the forward and adjoint wave-equation have to be solved. In this work we took a first principles look at both a performance model for the underlying numerical method, namely the roofline model, and considered this in the specific context of FWI where sources (RHS) are typically processed using task parallelism. We were readily able to double the speed of the code by redesigning the computational kernel so that its operational intensity was increased by processing multiple shots at once. This directly alleviates the key performance barrier for this problem, making the solver highly vectorized with an higher operational intensity. The changes required to the source code were minimal and did not require any platform specific tuning. While this work only considered the acoustic wave-equation, the method has broad applicability and will have even greater impact as the number of physical parameters involved increases. Even though this method requires higher memory resources to store the wavefield for each source, the latest in-core and out-of-core hardware and software developments will make it easily feasible.

6 Acknowledgements

This research was carried out as part of the SINBAD II project with the support of the member organizations of the SINBAD Consortium and the Imperial College London Intel Parallel Computing Centre.

References

[Asanovic et al., 2006] Asanovic, K., R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al., 2006, The landscape of parallel computing research: A view from berkeley: Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

[Colella, 2004] Colella, P., 2004, Defining software requirements for scientific computing.

[Kamil et al., 2006] Kamil, S., K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, 2006, Implicit and explicit optimizations for stencil computations: Proceedings of the 2006 Workshop on Memory System Performance and Correctness, ACM, 51–60.

[Louboutin et al., 2016] Louboutin, M., M. Lange, F. J. Herrmann, N. Kukreja, and G. Gorman, 2016, Performance prediction of finite-difference solvers for different computer architectures. Submitted to the Computer and Geoscience Journal on September 16, 2016.

[Williams et al., 2009] Williams, S., A. Waterman, and D. Patterson, 2009, The roofline model offers insight on how to improve the performance of software and hardware.: communications of the acm, **52**.

[Yount, 2015] Yount, C., 2015, Vector folding: Improving stencil performance via multi-dimensional simd-vector representation: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, 865–870.