# Can we beat FFTs computational speed for one-way wavefield propagation?

*Biondo Biondi*

## ABSTRACT

I compare the computational efficiency of one-way wavefield propagation by FFT-based methods and by implicit finite-differences methods. On modern computer architectures, the theoretical computational advantages of implicit finite-differences methods are not realized in practice. The speed tests that I present indicate that FFTs run much more efficiently than sparse linear solvers and thus they should be the foundation for efficient waveform inverse solutions based on one-way propagation.

## INTRODUCTION

Imaging the full bandwidth of seismic data by using full waveform inversion (FWI) and linearized waveform inversion (LWI) is hampered by the computational cost of two-way propagation that scales with the fourth power of the frequency. The computational advantages of one-way propagation compared to two-way propagation suggest its use for imaging subsurface structures that are not sufficiently complex to require full two-way propagation. Furthermore, Shan and Biondi (2008) showed that by combining plane-wave decomposition and one-wave propagation we can image steeply dipping reflectors and overturned events.

For high-resolution imaging, the main advantage of one-way propagation is that its computational cost grows slower with frequencies than the cost of two-way propagation. This is particularly true when we are interested in imaging with anisotropic models because one-way operators can be generalized to approximate the kinematics of wave propagation in anisotropic media more cost effectively than two-way propagators. The computational-cost advantages becomes even more attractive when considering waveform inversion with extended model such as tomographic FWI (Biondi and Almomin, 2014; Barnier et al., 2018). Sarkar and Biondi (2018) showed the time-domain velocity model extension using correlation-time is equivalent to a simple extension in frequencies when the wave-equation is solved in the frequency domain. Therefore, no additional correlations are needed for extended-model frequency-domain methods, although frequency-domain extended models are still substantially larger than not extended model and their handling may still lead to additional I/O costs.

*October 5, 2018*

Because accurate wide-angle imaging requires the propagation of wavefield by an operator with large spatial width, the computational kernels of the most commonly used one-way propagation methods require the application of FFTs or the solution of large linear system of equations (implicit methods). Mixed-domain methods that require both the application of FFTs and solution of linear systems are also often implemented (Gazdag and Sguazzero, 1984; Stoffa et al., 1990; Ristow and Ruhl, 1994; Biondi, 2006). A theoretical analysis of the computational complexity suggests that implicit methods might be particularly efficient because the linear system is solved by inverting a matrix that depends only on the velocity model. Consequently the cost of factorizing this matrix can be amortized an all the shots that share the same velocity model. Furthermore, for methods such as LWI and TFWI (Barnier et al., 2018) that require several linearized iterations with the same background model, the factorization does not change with iterations and thus can be reused across iterations, further decreasing the relative computational burden of matrix factorization.

However, comparative theoretical efficiency advantages may not translate into practical ones on modern computer architecture. In this report Akhmadiev et al. (2018) present the beginning of a project on waveform inversion by using one-way propagation. Therefore, I investigated the efficiency of implicit one-way propagation compared to FFT-based one-way propagation to determine which type of one-way extrapolator we may want to base that project. In my comparison I used the FFTW3 library to perform FFTs and the Pardiso solver part of the Intel Math Kernel Library (MKL) to factorize the matrix and recursively solve the system (Schenk and Gartner, 2004).

## NUMERICAL TESTS

I run my speed tests on a 2.3 GHz dual-CPU Intel Xeon "Haswell" processor (E5-2670-v3) with 12 cores per CPU for a total of 24 cores and a maximum of 48 threads. I run my downward continuation code by parallelizing the computations in two ways: 1) across temporal frequencies by using OpenMP threading and 2) within a single solver call using the intrinsic parallelism controlled by the MKL library. The coarse frequency parallelism that is exploited by OpenMP threading is the most natural and efficient way of parallelizing both the FFTs and the linear solver solution. I report the FFT speed only using this parallelization mode, whereas I measured and report the linear solver speed using both parallelization modes.

Vectorization is also essential to achieve efficiency with modern CPUs. When implicit one-way propagation is used to solve a waveform inversion problem, both parallelization and vectorization are possible when solving the same linear system with many right hand sides (RHS). The multiple RHS correspond to multiple shot gathers that share the same computational domain that may encompass the spatial extent of the whole survey, or a shared subset of it. Therefore, I run tests using different number of right hand sides ($N_{RHS}$) as well as three different lengths of the horizontal axis ($N_x$): two short ones ($Nx$=512 for OpenMP parallelization and $Nx$=544 for MKL

internal parallelization) and a long one ($N_x$=65,536 only for OpenMP parallelization). I report only the solver speeds and not of the matrix factorization because the same factorization of the matrix is used for all RHS. The cost of the matrix factorization is amortized on all RHS, and thus the controlling speed is the one of the solver, not the factorization.

The computational throughputs shown in the figures below were measured using the system clock. They show how many millions of horizontal grid points can be propagated by one depth level per second. The measured throughputs were averaged over 220 depth steps and 240 temporal frequencies.

## Parallelism across temporal frequencies by OpenMP threads

Figures 1–3 show the throughputs of the FFTs parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) used. The asterisks correspond to actual measurements, whereas the solid lines interpolate between measurements. The dashed lines show the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. The results shown in these figures correspond to different values of $N_x$ and $N_{RHS}$ to test the scalability in different situations.

Figure 1 shows the results for $N_x$=512 and $N_{RHS}$=48. Computational throughput scales almost linearly up to 12 OpenMP threads, and then starts flattening out most likely because memory bandwidth is increasingly saturated. The maximum throughput is achieved for 48 threads. When this measurement is converted to flop/s using the "classical" formula for estimating the number of flops needed to compute FFTs ($N_{flop} = 5N_x \log N_x$), it results in an impressive 478 Gflop/s. This speed is about 48% of the maximum theoretical speed of the processor when the clock frequency is "turbo" boosted to 2.6 GHz and 16 flop are performed for each cycle of all the 24 cores available. When the number of right hand-sides is scaled by a factor 20 to $N_{RHS}$=960, the behavior is similar but the maximum speed of 416 Gflop/s is slightly lower because of memory-bandwidths limitations, as shown in Figure 2.

Figure 3 shows that the throughput decreases when $N_x$ is scaled by a factor of 128; that is $N_x = 65,536$ and $N_{RHS}$=2. This decrease is caused by the more-than-linear increase in computational complexity of FFTs. The actual computational speed with $N_{OMP}$=48 is 425 Gflop/s; that is still a high percentage of the maximum theoretical speed of the hardware. It is unlikely that in a real 2D problem the horizontal axis is as long as 65,536 samples. However, my code is currently limited to 2D wave propagation, and this choice of parameters was motivated to simulate the computational complexity of 2D FFTs when performing 3-D wave propagation on a fairly small problem (65,536=256*256).

Figures 4–6 show the throughputs of the implicit propagator parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) used. As for the previous figures, the asterisks correspond to actual measurements, whereas the

solid lines interpolate between measurements. The dashed lines show the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads.

The general scaling trend in Figure 4 is similar to the trend in Figure 1, however the throughput axis is now scaled down by a factor of 100. The maximum throughput achieved by FFTs with 48 OpenMP threads is higher by a factor 75 than the one achieved by the linear solver using the same number of threads. When the number of RHS is scaled by 20 (Figure 5) the comparison with FFTs throughput is slightly less unfavorable because the computations are better vectorized over the RHS axis. The linear solver throughput scales better than before up to a 24 OpenMP threads and the ratio between the maximum throughputs achieved with 48 OpenMp threads is "only" 65.

Figure 6 shows that the throughput substantially decreases when $N_{RHS}$ decreases to 2 and $N_x$ is scaled up by a factor of 128. The ratio between the maximum throughputs achieved by FFTs and linear solver with 24 OpenMp threads is 78. This decrease in relative efficiency of the linear solver is caused by the lack of vectorization over the RHS axis; it more than compensates for the relative advantage that the solution phase of the linear scales linearly with $N_x$, in contrast with the superlinear increase of computational complexity of FFTs.

## Parallelism within solver call by MKL threads

The sparse-matrix solver in the Pardiso library has been developed to take advantage of multi-threaded computers for parallelizing computations within each library call. This capability is accessed by setting the number of MKL threads ($N_{MKL}$) using an environment variable. Therefore, I tested this alternative way of parallelizing the solution of the linear system to verify whether it leads to higher performances than the coarse parallelization over temporal frequencies achieved by calling the library solver from different OpenMP threads that I discussed in previous subsection. Figures 7 and 8 show the throughputs measured with different MKL threads and with different number of RHS. Notice that I show the results with $N_x$=544 because performances were slightly higher than with my first choice ($N_x$=512). Performances improve when the dimension of the system is not a power-of-two because the reuse of in-cache data increases (Stew Levin, personal communication). The maximum number of threads that the Pardiso library can utilize is equal to the number of cores available in hardware, therefore I measured performances using up to 24 threads.

Figure 7 shows that show the throughputs of the implicit propagator as a function of $N_{MKL}$ used, with $N_{RHS}$=1920; that is the maximum I could fit into memory. As for the previous figures, the asterisks correspond to actual measurements, whereas the solid line interpolates between measurements. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. The performance scales almost linearly with

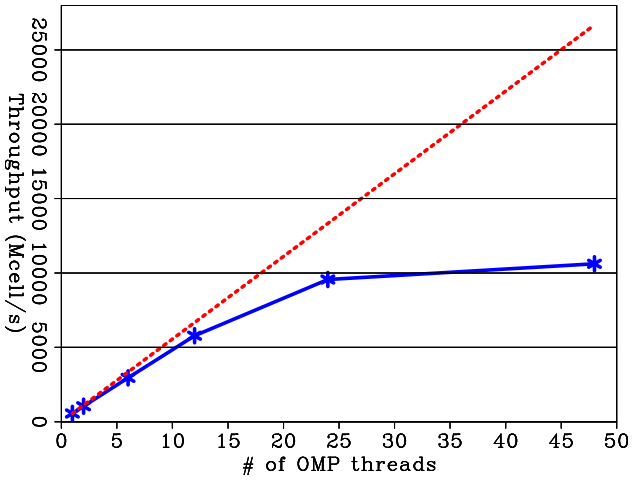


Figure 1: Throughputs of FFTs parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=512 and $N_{RHS}$=48. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

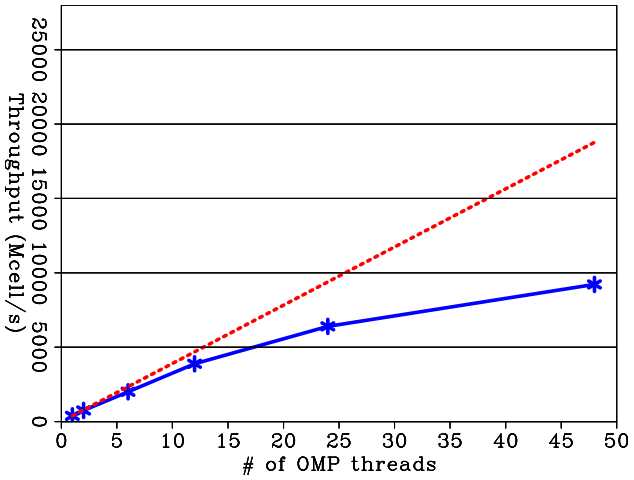


Figure 2: Throughputs of FFTs parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=512 and $N_{RHS}$=960. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

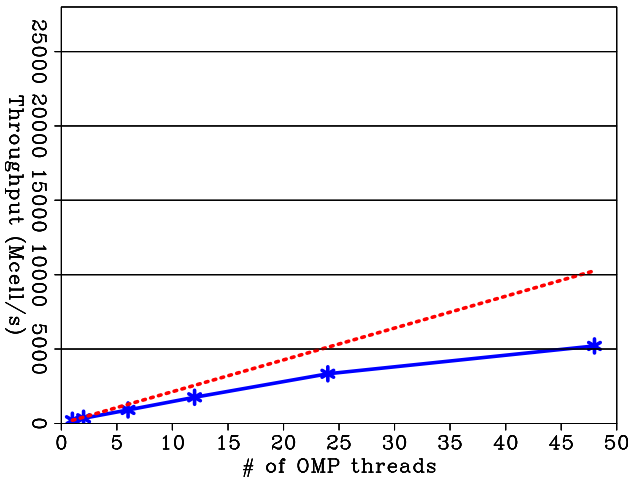


Figure 3: Throughputs of FFTs parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=65,536 and $N_{RHS}$=2. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

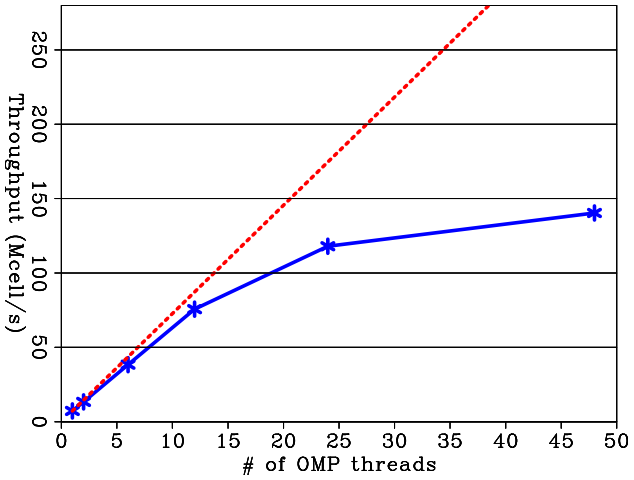


Figure 4: Throughputs of Pardiso solver parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=512 and $N_{RHS}$=48. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

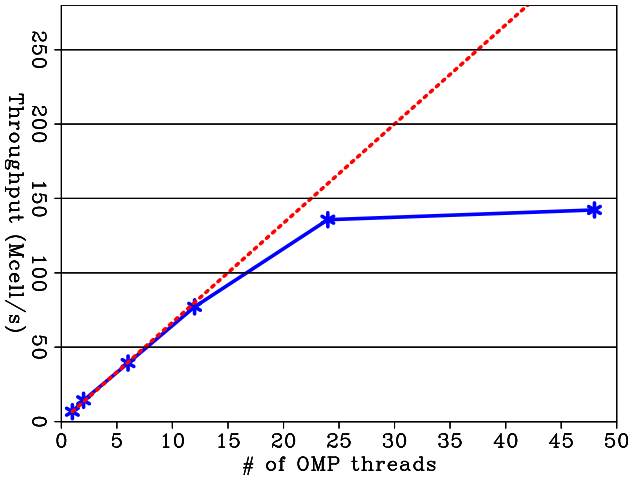


Figure 5: Throughputs of Pardiso solver parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=512 and $N_{RHS}$=960. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

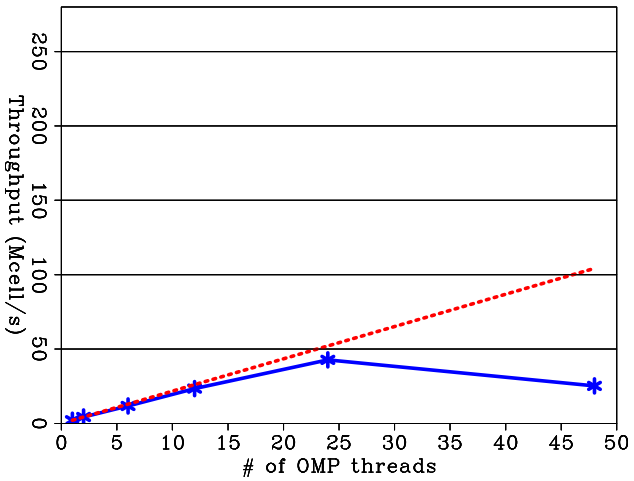


Figure 6: Throughputs of Pardiso solver parallelized over frequencies as a function of the number of OpenMP threads ($N_{OMP}$) with $N_x$=65,536 and $N_{RHS}$=2. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

the number of threads up to $N_{MKL} = 12$ but it decreases with $N_{MKL} = 24$. This is probably caused by saturation of the memory bandwidth.

Figure 8 demonstrates that the solver efficiency strongly depends on vectorization as well as parallelization by showing that throughput substantially improves as $N_{RHS}$ increases up to 960, although it starts to flatten out after 960. It shows these gains for both $N_{MKL} = 12$ and $N_{MKL} = 24$. However, by comparing Figure 8 with Figure 5 we can observe that the maximum throughput achieved by the intrinsic parallelism within each solver call is lower by a factor of two with respect to the maximum throughput achieved by parallelizing over temporal frequencies. Consequently we can conclude that parallelizing over frequencies is a better choice than using the intrinsic parallelization in the Pardiso library, even when the number of RHS is sufficiently large to enable full vectorization. Some mixed-domain methods require both the performance of FFTs and the solutions of linear systems of equations (Ristow and Ruhl, 1994; Biondi, 2006); therefore, parallelization over frequencies seems to be the optimal choice when these methods are implemented.

## DISCUSSION AND CONCLUSIONS

The throughput results presented in the previous section clearly show that the computational efficiency of FFTs on modern architectures tilts the plane in favor of one-way propagation algorithms that use FFTs to perform the computational heavy lifting. Even when solving relatively large problems for which FFTs computational cost scales faster than linearly, the measured throughput is still faster by a factor between 50 and 100. In practice, we can use many reference velocities in FFT-based methods to approximate propagation with strong lateral variations and still achieve higher throughput than by using implicit methods.

The computational efficiency of the Pardiso solvers that I describe in this paper might be improved. First I may not have correctly used all the parallelizing options provided by the MKL library. However, an analysis of the graphs in Figure 7 and Figure 8 clearly shows that I succeeded to achieve a high degree of parallelization inside each solver call. Second, the MKL library provides the option of supplying user-written customized solver based on the matrix factorization performed by the library. Probably some efficiency gain could be achieved by using a customized solver that takes advantage of problem-specific knowledge of the structure of the matrix factors to improve the use of available memory bandwidth. However, it is doubtful that those gains would be large enough to change the balance in favor of implicit methods, and justify the additional code development efforts required for developing customized solvers.

When anisotropic propagation is implemented with FFTs, it is important to use smart reference velocity-parameters selection methods, such as Lloyd algorithm (Clapp, 2004; Tang and Clapp, 2006). Efficient algorithms may also include correction terms analogous to the ones presented by Ristow and Ruhl (1994), but applied as con-

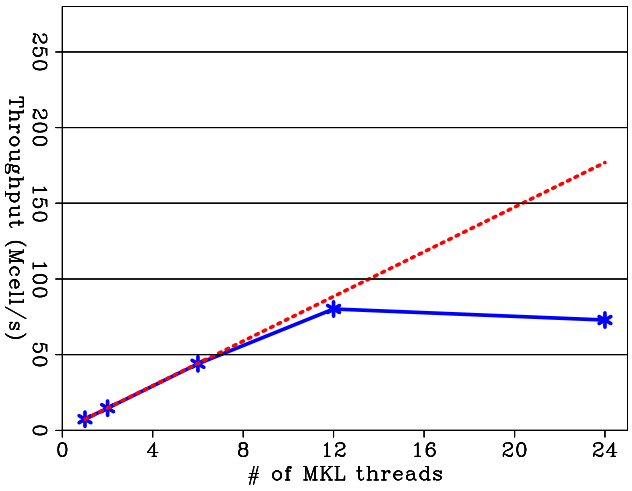


Figure 7: Throughputs of Pardiso solver parallelized within the library call using the intrinsic parallelization as a function of the number of MKL threads ($N_{MKL}$) with $N_x$=544 and $N_{RHS}$=920. The dashed line shows the theoretical throughput that would be achieved if the throughput measured for one thread scaled-up linearly with the number of threads. [**NR**]

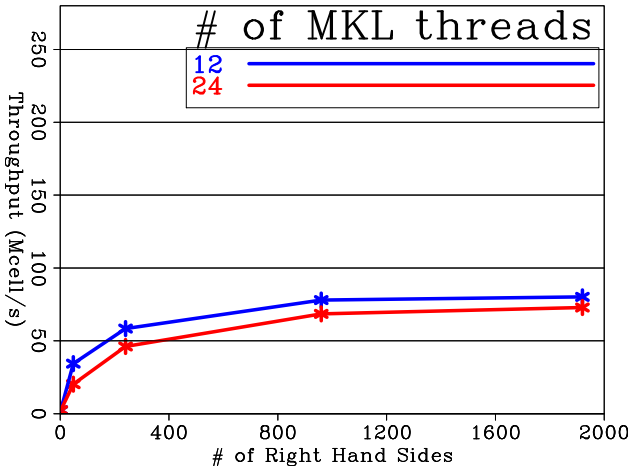


Figure 8: Throughputs of Pardiso solver parallelized within the library call using the intrinsic parallelization as a function of the number RHS with $N_x$=544 and for $N_{MKL}$=12 (blue line) and $N_{MKL}$=24 (red line). [**NR**]

volutional filters instead of as recursive linear-systems solutions. These convolutional correction terms would perform more efficiently than implicit methods on modern architectures; the challenge would be to assure unconditional stability of wavefield propagation using convolutional correction terms.

## ACKNOWLEDGMENTS

I would like to thank Stew Levin and Bob Clapp for sharing their in-depth knowledge of modern architectures to improve the significance of the tests I present in this paper.

## REFERENCES

Akhmadiev, R., B. Biondi, and R. G. Clapp, 2018, Full-waveform inversion problem using one-way wave extrapolation operators: SEP-Report, **174**, 31–40.

Barnier, G., E. Biondi, and B. Biondi, 2018, Full waveform inversion by model extension: SEP-Report, **172**, 153–171.

Biondi, B. and A. Almomin, 2014, Simultaneous inversion of full data bandwidth by tomographic full waveform inversion: Geophysics, **79**, WA129–WA140.

Biondi, B. L., 2006, 3D Seismic Imaging: Society of Exploration Geophysicists.

Clapp, R. G., 2004, Reference velocity selection by a generalized Lloyd method: 74th Ann. Internat. Mtg., Expanded Abstracts, 981–984, Soc. of Expl. Geophys.

Gazdag, J. and P. Sguazzero, 1984, Migration of seismic data by phase-shift plus interpolation: Geophysics, **49**, 124–131.

Ristow, D. and T. Ruhl, 1994, Fourier finite-difference migration: Geophysics, **59**, 1882–1893.

Sarkar, R. and B. Biondi, 2018, Frequency domain tomographic full waveform inversion: SEP-Report, **172**, 173–191.

Schenk, O. and K. Gartner, 2004, Solving unsymmetric sparse systems of linear equations with pardiso: Future Generation Computer Systems, **20**, 475–487.

Shan, G. and B. Biondi, 2008, Plane-wave migration in tilted coordinates: Geophysics, **73**, S185–S194.

Stoffa, P. L., J. Fokkema, R. M. de Luna Freire, and W. P. Kessinger, 1990, Split-step Fourier migration: Geophysics, **55**, 410–421.

Tang, Y. and R. G. Clapp, 2006, Selection of reference-anisotropy parameters for wavefield extrapolation by Lloyd's algorithm: 76th Ann. Internat. Mtg., Expanded Abstracts, ANI 2.7, Soc. of Expl. Geophys.