ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/321487962

Fast matched filter (FMF): An efficient seismic matched-filter search for both CPU and GPU architectures

Preprint *in* Seismological Research Letters · November 2017

CITATIONS
0

READS
1,008

3 authors:

Eric Beaucé
Massachusetts Institute of Technology
12 PUBLICATIONS **84** CITATIONS
SEE PROFILE

William Benjamin Frank
Massachusetts Institute of Technology
45 PUBLICATIONS **683** CITATIONS
SEE PROFILE

Alexey Romanenko
Novosibirsk State University
46 PUBLICATIONS **195** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Recursive detection of swarms of volcanic long-period seismicity in Marie Byrd, Antarctica View project

Project Automated earthquake detection View project

All content following this page was uploaded by Eric Beaucé on 04 December 2017.

The user has requested enhancement of the downloaded file.

Fast matched-filter (FMF): an efficient seismic matched-filter search for both CPU and GPU architectures

Eric Beaucé[a], William B. Frank[a], Alexey Romanenko[b]

[a]*Department of Earth, Atmospheric, and Planetary Sciences, Massachusetts Institute of Technology, Cambridge, MA, United States*
[b]*Novosibirsk State University, Novosibirsk, Russia*

Abstract

Matched-filter searches are an important tool in modern seismology to detect seismic events. The algorithm functions by computing the correlation coefficient between a template earthquake and a sliding window of continuous seismic records. A detection is recorded when the correlation coefficient crosses an established threshold. We present an optimized program, called Fast Matched-Filter (FMF), to efficiently run a network-based matched-filter search with either central processing units (CPUs) or Nvidia graphics processing units (GPUs). Wrappers for both Python and Matlab (CPU only) are provided to easily run FMF on a wide range of computational resources, from multi-core laptops to specialized computing clusters with GPUs. Both implementations leverage the embarrassingly parallel structure of the continuous computation of correlation coefficients in the time domain to achieve rapid performance. The highly parallel architecture of GPUs lends itself perfectly to the matched-filter algorithm, and we achieve the fastest run times with our GPU implementation. FMF allows for seismic network-based matched-filtering between a large set of template waveforms and a large continuous dataset in a reasonable amount of time. Such fast run times are an important step in expanding the scope of earthquake detection and fostering the reproducibility of such studies.

Keywords: matched-filter search, earthquake detection, high-performance computing (HPC), graphics processing unit (GPU)

1. Introduction

Many studies in seismology are based on detecting earthquakes and gathering them into a catalog of their locations and timing. Earthquake catalogs provide a window into the solid Earth, including but not limited to, the Earth's mechanical properties, [e.g. Dziewonski and Anderson, 1981; Van der Hilst et al., 1997], the amount of tectonic stress released at plate margins [e.g. Rogers and Dragert, 2003; Frank, 2016], and the statistical laws that govern earthquake occurrence [e.g. Gutenberg and Richter, 1944; Utsu, 1961; Frank et al., 2016]. Systematically generated earthquake catalogs are thus an important resource in seismology. A key to constructing such catalogs is the development of algorithms that can detect earthquakes over a wide range of magnitudes within large volumes of data in a feasible amount of time.

Earthquake catalogs can be made with simple power-based detection algorithms, like the short-term average/long-term average (STA/LTA) detector [Allen, 1982], but such methods often have difficulty discriminating earthquakes from transient noise sources. Earthquake detection is further complicated when the amplitudes of target seismic events are on the same order of magnitude as the ambient seismic noise, measured as the signal-to-noise ratio (SNR). Because the SNR is higher when the magnitude of the event is larger and/or the source-receiver distance is shorter, detection capabilities can vary greatly based on both source position and mechanism, and the geometry of the recording seismic network [e.g. Kwiatek and Ben-Zion, 2016]. Through the analysis of multiple seismic stations together, one can lower the SNR threshold for detection by leveraging the coherent information over the seismic network. A method that has proven itself capable of detecting earthquakes with low SNRs on seismic networks is template matched-filtering [e.g. Gibbons and Ringdal, 2006; Shelly et al., 2007; Frank et al., 2014].

Template matched-filtering is motivated by the fact that when two colocated earthquakes occur with the same focal mechanism, their waveforms are very similar to each other and should have a high correlation coefficient. Matched-filtering algorithms compute the average correlation coefficient (CC) between the waveforms of a reference event, or template, and a sliding window of the continuous seismic records at multiple stations:

$$\mathrm{CC}(t) = \frac{1}{N_s N_c} \sum_{s,c} \frac{\sum_{n=1}^{N} T_{s,c}(t_n) S_{s,c,}(t_n + \tau_{s,c})}{\sqrt{\sum_{n=1}^{N} T_{s,c}^2(t_n) \sum_{n=1}^{N} S_{s,c}^2(t_n + \tau_{s,c})}} \tag{1}$$

In Equation 1, s, c and n are respectively the indices for the station, the component and the time sample, and N_s, N_c and N are the respective number of stations, components and the template's length in samples. T, S are respectively the template and the continuous seismic data, and $\tau_{s,c}$ is the moveout on station s and component c. The moveout $\tau_{s,c}$ describes the delay with which an event is observed on every station relative to the beginning of the earliest template's window. The sliding window of continuous data that produces a high CC is considered a newly detected seismic event that is colocated with the template earthquake.

It is important to note that we assume the waveforms are centered around zero in Equation 1, which differs from other authors' definition that removes explicitly the mean from each sliding window. This last assumption allows for a large speed up of the algorithm, but makes it inappropriate for matched-filtering on raw data. However, if the data are highpass filtered such that periods longer than the template's duration are removed, this assumption is valid most of the time. The zero-mean assumption can break down when high energy signals are recorded, but the effect on the correlation coefficients is very limited (error $< 1\%$). This small error on the correlation coefficient can be shown to only bias the CC toward zero. Thus, we state that the zero-mean assumption neither affects the detection capability of the algorithm, nor favors the detection of unwanted events.

An existing earthquake catalog can be expanded by using its constituent earthquakes as templates in a matched-filter search. Each template will detect similar events over a wide range of SNRs [Warren-Smith et al., 2017]. This can, however, be very costly if the catalog contains a large number of events. Making this method computationally efficient is therefore essential to avoid any limitations related to long computation times, such as limiting the number of templates or limiting the duration of the continuous search. One example that demonstrates this is recent work on the postseismic sequence of the 2015 M_w 8.3 Illapel earthquake in Central Chile. Huang et al. [2017] performed a matched-filter search for aftershocks during the month that followed the mainshock. They were not capable of capturing the log-time spatial expansion of aftershock seismicity, which is indicative of an afterslip-driven postseismic phase [e.g. Perfettini and Avouac, 2004]. However, [Frank et al., 2017] observed this behavior by processing a longer detection period of 10 months.

The ability to run an algorithm several times within a reasonable amount of time is also crucial in earthquake detection, where studies must often

choose some empirical parameters. One common example is choosing a CC threshold to determine when a new event is detected. This threshold represents a trade-off between detecting false positives (detecting non-seismic events that are unwanted) and detecting false negatives (missing target seismic events). Depending on the overarching goal of the detection, one threshold might be favored over another. If the detection algorithm is fast enough, different parameters such as template length, frequency band, or a subset of the seismic network, can be evaluated to select the best ones for the job. We believe that the open-source distribution of a fast detection algorithm, along with the careful reporting of chosen detection parameters, will encourage independent groups to reproduce and verify future studies.

In this context we propose an optimized program, which we call Fast Matched-Filter (FMF), to execute a matched-filter search and compute Equation 1. We provide two versions implemented for different computational resources. One version is written in C and can run on compute nodes typically comprised of several central processing units (CPUs), but can also run on any ordinary multi-core laptop or desktop. We refer to this as the CPU implementation. We also provide an Nvidia CUDA C version that runs on computers with Nvidia graphics processing units (GPU). We refer to this version as the GPU implementation. We note that FMF's GPU implementation does not function with AMD GPUs, as CUDA C is developed for Nvidia GPUs. Both versions are open-source and are hosted on Github: `https://github.com/beridel/fast_matched_filter`.

2. CPU Implementation

The CPU implementation only requires a C compiler (*e.g.* the GNU compiler collection *gcc*). The core of the program is written in C , and wrapper functions are available for Python and Matlab to allow for easy use from a higher level programming interface. The continuous computation of the correlation coefficient (Equation 1) is an embarrassingly parallel algorithm: the computations at different times t and for different channels or templates are completely independent. For that reason, the parallelization is only limited by the number of threads we can create, each independently working through Equation 1 at the same time. We parallelize the program over the temporal loop, meaning that different sliding windows of our data are assigned to different threads.

In this massive computation, the input/output operations (I/O) to access the data are a non-negligible source of slowdown, whose impact depends on the computer's or computation node's architecture. Before the CPU can continue computational operations, it must wait during I/O operations for the disk access to finish. We develop the scheme described below and shown in Figure 1 that focuses on an efficient parallelization and only performs a single I/O operation:

1. in the Python and Matlab wrapper functions, a single I/O operation is achieved at the beginning of the algorithm,
2. noting that the denominator in Equation 1 involves redundant operations, the Python and Matlab wrapper functions pre-compute the sums of the squared template waveforms and a C function pre-computes the cumulative sum of the squared seismograms within each sliding window,
3. the heavy computations are performed in C: for each template the temporal loop is parallelized into several threads using OpenMP [Dagum and Menon, 1998] (OpenMP automatically determines the number of threads that can be created given the available resources).

The arithmetic mean is used to average the correlation coefficients over the stations and components in Equation 1. When dealing with real data, however, it might be useful to attribute different weights to different stations and components. For example, one could attribute the highest weights to the stations/components that record the highest SNR waveforms. We modify Equation 1 to take into account a weight per station/component:

$$\mathrm{CC}(t) = \sum_{s,c} w_{s,c} \frac{\sum_{n=1}^{N} T_{s,c}(t_n) S_{s,c,}(t_n + \tau_{s,c})}{\sqrt{\sum_{n=1}^{N} T_{s,c}^2(t_n) \sum_{n=1}^{N} S_{s,c}^2(t_n + \tau_{s,c})}} \tag{2}$$

where $w_{s,c}$ is the user-designated weight associated with station s and component c. If the sum of the weights is normalized to one, then attributing equal weights to every station and component results in computing the arithmetic mean.

The Python and Matlab wrappers ensure the inputs are well formatted for the C functions, and reduce the arguments to:

- templates: array with dimensions[1] (N_t, N_s, N_c, N) storing the template waveforms, where N_t is the number of templates and the other parameters are the same as in Equation 1.

- moveouts: array with dimensions[1] (N_t, N_s, N_c) storing the moveouts in number of samples.

- weights: array with dimensions[1] (N_t, N_s, N_c) storing the weights attributed to the stations for each template.

- data: array with dimensions[1] $(N_s, N_c, N_{\text{data}})$ storing the seismograms, where N_{data} is the length of a seismogram in number of samples.

- step: number of samples between each computation of the CC.

The data array has to fit in the CPU's memory. Therefore, to process a long period of continuous data, the user has to call consecutively the wrapper function with successive chunks of data. Typically, continuous seismic data is divided into daily traces and the wrapper function can be used on each of them.

3. GPU Implementation

Computational problems arising from gaming and 3D graphics motivated the development of GPUs: in order to scale and rotate a scene, one has to perform millions of simple matrix operations on the polygons that make up the 3D visualization. GPUs are designed to address such problems with most of their transistors devoted to computation rather than I/O operations, as is the case for CPUs. Thanks to this specialized architecture, GPUs can be much more powerful than CPUs for specific applications [Mu et al., 2017]. The embarrassingly parallel nature of matched-filter searches and the fact that our algorithm only performs one I/O operation make this computation well-suited for the use of GPUs. The GPU version of FMF requires a computation node equipped with at least one Nvidia GPU. The code is based on CUDA C [Nickolls et al., 2008] and requires the user to install the CUDA toolkit, which includes the compiler nvcc.

One key aspect of the GPU architecture is the organization of the computation into blocks and threads. Each block is subdivided into threads

[1]These dimensions are for C-contiguous arrays. The order is reversed in Matlab, as Matlab uses Fortran-contiguous dimensions.

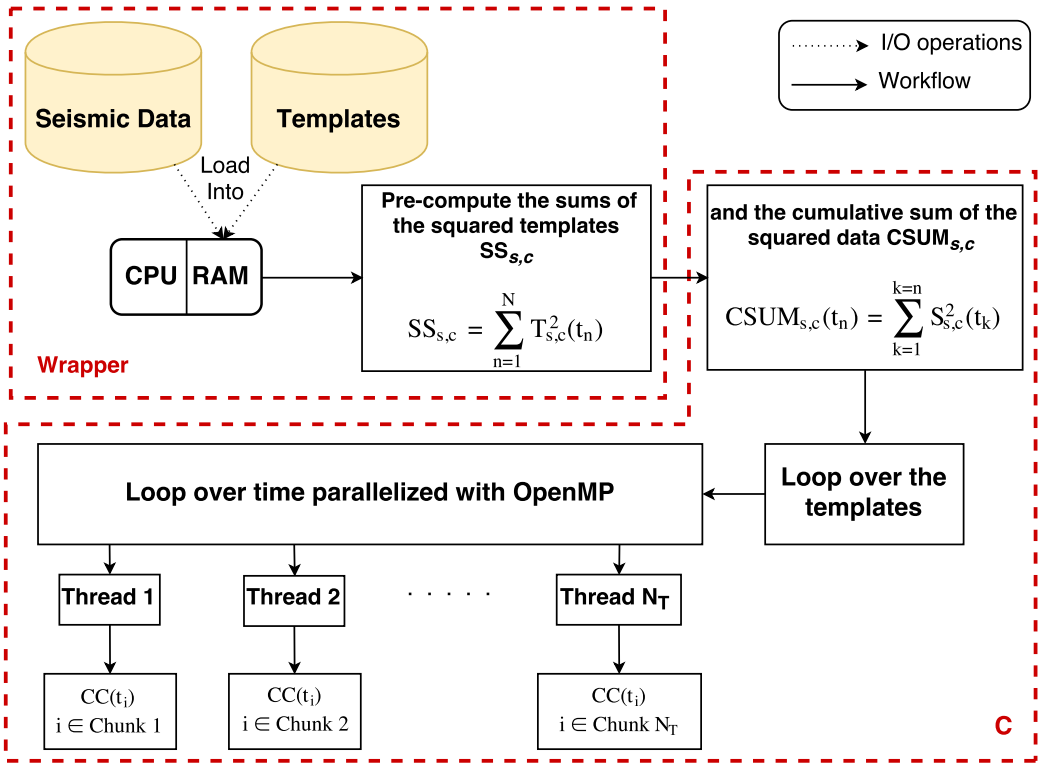

Figure 1: Workflow of the CPU implementation of our program Fast Matched-Filter (FMF). A single large I/O operation is achieved at the beginning by reading the templates' waveforms and the continuous data. Beside, the sums of the squared templates and the cumulative sum of the squared data are computed before entering the loops to avoid redundant operations. After that, the iterative computation of the CC starts and is parallelized with OpenMP: different sliding windows (chunks) of the data are assigned to different threads. The two dashed boxes indicate which part is executed by the wrapper, and which one by the C code.

that each run a single task; in our case, this task is computing the correlation coefficient between a template and one sliding window of continuous data. This structure allows for a much greater degree of parallelization than on CPUs. On a single GPU one typically runs several blocks at the same time, each associated with several hundreds of threads. This amounts to $\sim 1,000 - 10,000$ threads running at the same time, compared with the $\sim 2 - 100$ threads usually running on CPUs. To further optimize the use of GPUs to the matched-filtering algorithm, we leverage the many redundant memory accesses involved in the continuous computation of Equation 2. Dur-

ing the whole matched-filter search, a data sample is not accessed by a single thread, but by N different threads in total (each thread reads N data samples to compute a single correlation coefficient, N being the template length). These redundant memory accesses can be exploited through the use of shared memory. Each thread of the same block (*cf.* Figure 2) loads a data or template sample from the GPU's global memory (its equivalent of RAM) into the block's shared memory; this sample is then no longer revisited in global memory by other threads. Because shared memory has a throughput $\sim 10\times$ larger than the throughput of global memory, this optimization represents a significant speedup of the algorithm. Beside, this transfer from global memory to shared memory is such that neighboring threads read data/template samples that are contiguous in the global memory. This organization allows to access the memory in a coalesced manner: neighboring threads read samples at the cost of one memory transaction instead of one transaction each.

The algorithm's workflow is described below and illustrated in Figure 2:

1. in the Python function (no GPU implementation is currently available for Matlab), a single I/O operation is performed at the beginning of the algorithm,
2. the wrapper function computes the sums of the squared templates, and formats the inputs for the CUDA C code,
3. the communication between the host (the CPUs) and the device (the GPUs) is achieved in CUDA C. The templates and the data (plus the other inputs necessary to compute Equation 2) are transferred to the device. If multiple GPUs are available, an OpenMP parallelization distributes the templates to different GPUs.
4. Many computation blocks, each of them involving many threads, are simultaneously (up to a given number of blocks that is hardware dependent) set up to compute the CCs for different templates/stations/components and at different times. This process is repeated until the entire data has been scanned (*cf.* Figure 3).
5. The CCs are transferred back from the device (GPU) to the host (CPU).

In this algorithm, we skipped the computation of the cumulative sum of the squared data because GPUs typically feature limited amounts of memory (*e.g.* $\sim$ 6Gbytes or less) that cannot store the vector of cumulative sums in addition to the vector of continuous data. This does not change, however,

the arguments that are passed to the GPU wrapper function, which are the same as the CPU implementation.

4. Speed Comparison

We test FMF running a series of numerical experiments that generate synthetic data and templates, and compute the correlation coefficient between them (*i.e.* the continuous computation of Equation 2). We ran our tests on a compute node composed of 2 x 12-core 2.50GHz CPUs (Intel Xeon Processor E5-2680 v3), 256GB RAM and two Nvidia K80 Tesla dual-GPUs, amounting to a total of twenty-four CPU cores and four GPUs.

The first test we present used the following inputs: 5 stations, 3 components, between 1 and 10 templates with a waveform duration of 8 s, one-day long seismograms at a sampling rate of 10 Hz. The correlations are computed with a temporal step equal to one sample, or 0.1 s. A Matlab code that leverages the same optimizations as CPU implementation (through Matlab's Parallel Processing Toolbox's *parfor* loop) and the EQcorrscan program ([Chamberlain et al., 2017], we used the version 0.2.7 and the back-end FFTW) are compared with the CPU and GPU versions of our program. Figure 4 shows the large difference of performance between the Matlab code and the other programs, emphasizing that Matlab's high-level language is not suited to high performance computing. To further discriminate the two versions of our program and EQcorrscan, we choose to present another test using more realistic input parameters. Matched-filtering is often performed with more than five stations to take advantage of the seismic network and to obtain more statistically robust correlation coefficients. Sampling rates are also typically higher than 10 Hz.

The second test used the following input parameters: 12 stations, 3 components, between 1 and 50 templates with duration 8 s, one-day long seismograms at a sampling rate of 50 Hz and the cross-correlation is computed with different temporal steps (1, 5 and 10 samples). We ran our CPU version and EQcorrscan on 24 cores and our GPU version on 1 and 4 GPUs; Figure 5 shows all the results. As an example, running 20 templates in a matched-filter search with those parameters and a temporal step of 1 takes:

- with our CPU code: 55.5 s,

- with EQcorrscan: 15.8 s,

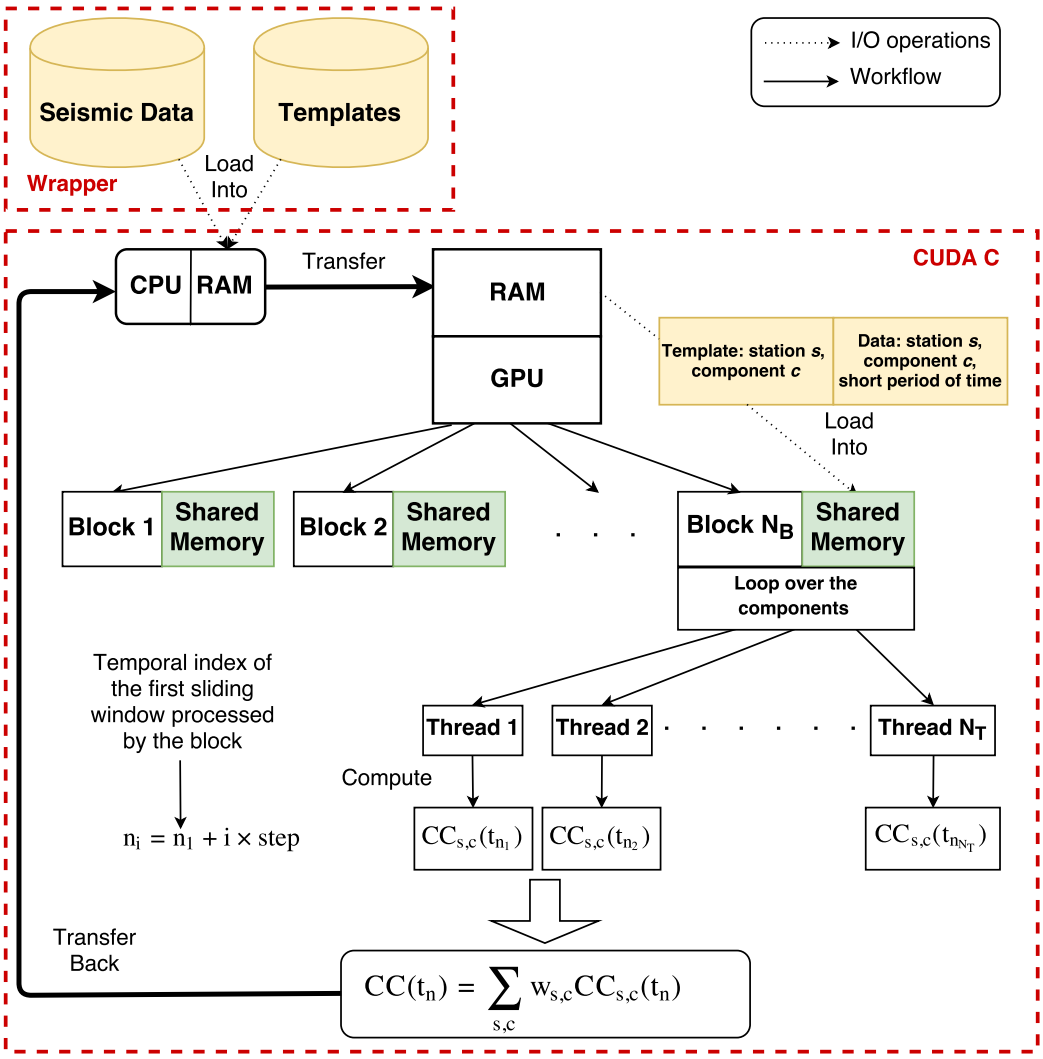

Figure 2: Workflow of the GPU implementation of FMF. The continuous seismic data and the templates are first read from the disk by the CPUs and transferred to the GPUs. The GPUs take advantage of the collective behavior of the threads and their quick access to shared memory to efficiently parallelize the computation. One GPU sets up several blocks (~ 10) whom each creates many threads (512) that computes the CC on all the components of a given template/station and at different times ($\mathrm{CC}_{s,c}(t_{n_i})$). The average correlation coefficients are computed through a weighted average, and are eventually transferred back to the CPUs. The two dashed boxes indicate which part is executed by the wrapper, and which one by the CUDA C code.

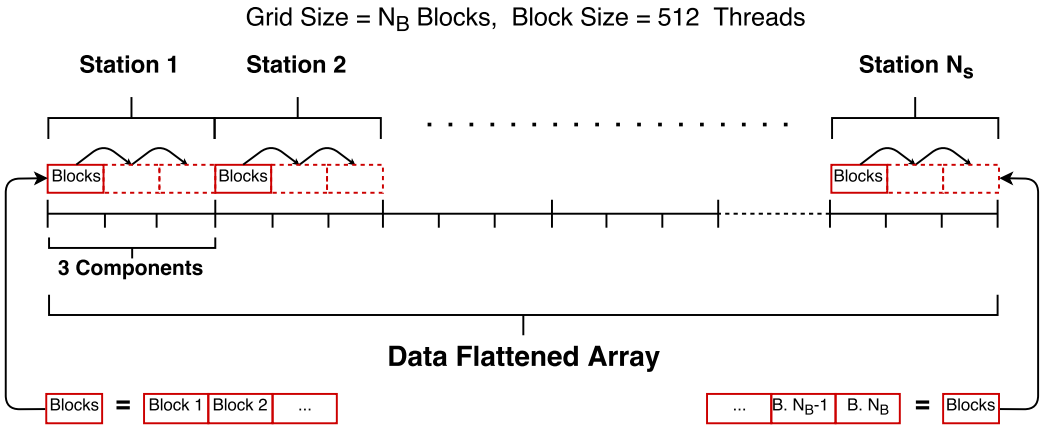

Figure 3: Organization of the grid of thread blocks set up by the GPU to compute the correlation between one template and the data. The block size (the number of threads per block) is 512 and the grid size (the total number of blocks) is N_B. The grid size depends on the length of the data array and on the temporal step; N_B is just enough so that all the correlation coefficients are computed. Note that this scheme holds for a single template, but the same scheme runs in parallel when several GPUs are available.

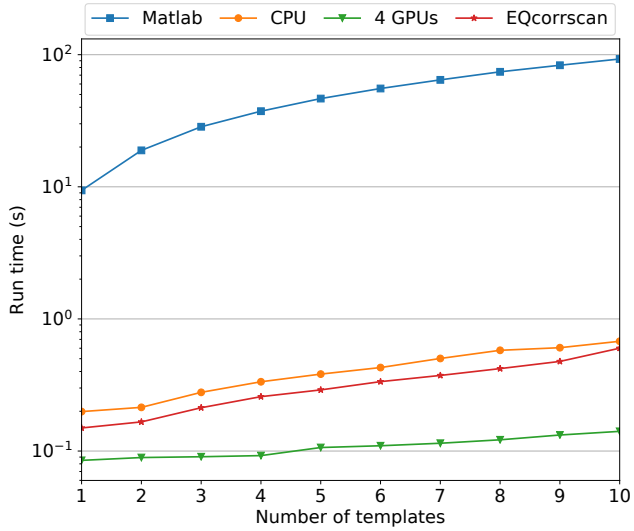

Figure 4: Comparison of the run times of matched-filter searches with different codes. Matched-filtering is achieved between one-day long synthetic seismograms on 5 stations, 3 components and a set of 8-second long templates whose size varies from 1 to 10. The sampling rate is 10 Hz and the temporal step used in the CC computation is 1 sample. Note the log scale on the y-axis.

- with our GPU code running on 1 GPU: 11.1 s,

- with our GPU code running on 4 GPUs: 3.5 s.

The GPU code running on a single GPU is respectively 5× and 2× faster than our CPU code and EQcorrscan running on 24 cores. In other words, one GPU is here equivalent to 48 or 120 cores, depending on the program that is running. The speedup associated with the increase of GPUs' number is almost linear (11.1/3.5=3.2), making the use of many GPUs very attractive.

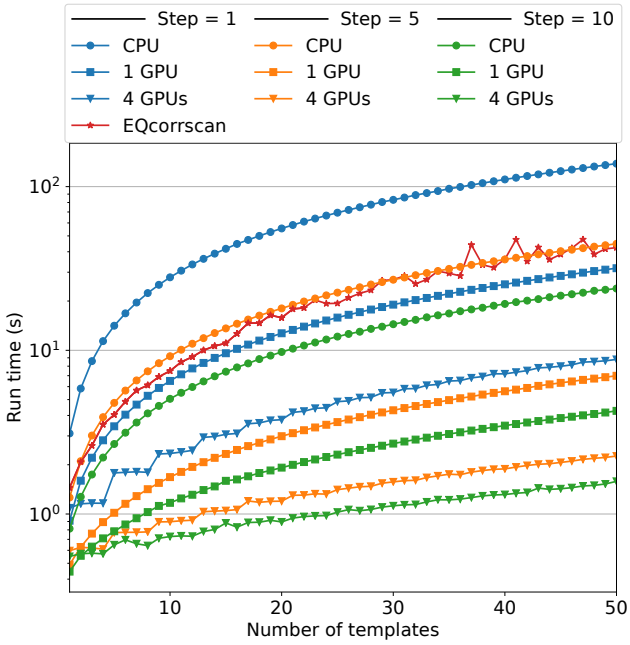

Figure 5: Comparison of the run times of several matched-filter searches using the CPU and the GPU implementations of our FMF and EQcorrscan. Matched-filtering is achieved between a set of one-day long synthetic seismograms on 12 stations, 3 components and a set of 8-second long templates whose size varies from 1 to 50. The sampling rate is 50 Hz and the temporal step takes the values 1,5 and 10. Note the log scale on the y-axis.

EQcorrscan, which computes the correlation coefficients in the spectral domain, is $\sim 3.5\times$ faster than the CPU version of our program for this choice of parameters. However this speedup is associated with a very memory-consuming algorithm. Indeed, the complex numbers required to perform the computation in the spectral domain are 2x larger than double precision

float numbers (16 bytes per element) whereas we only use single precision float numbers in our algorithm. Even though our node has enough RAM to run EQcorrscan with 50 templates, we note that memory problems can arise due to insufficient resources on other machines. We also emphasize that using single precision float numbers is part of the optimization on GPUs because Nvidia GPUs are designed to work efficiently on 32-bit numbers. The memory gain and speedup are only at the cost of a small loss of precision (typically of $\sim 10^{-3}$ when large numbers appear in the data). Another advantage of performing matched-filtering in the temporal domain is that we are not constrained to a temporal step of one, whereas it is the only option in the spectral domain because of the intrinsic mathematical formulation of the problem.

Increasing the temporal step between correlation coefficients, or decreasing the number of sliding windows of the continuous dataset, is a great source of speedup, but decreases the time precision of detected events and the overall detection capability. Depending on the frequency content of the events being searched for and the sampling rate, then increasing the temporal step might be a reasonable choice. For the CPU code, the speedup is inversely proportional to the temporal step increase; in Figure 5 the 10-step run takes $10\times$ shorter than the 1-step run. However, the speedup is not as efficient for the GPU version, due to the fact that a temporal step of one makes an optimal use of the shared memory in our code.

To summarize the results of Figure 5 in another way, if one were to run a matched-filter search with 50 8-second long templates on 12 stations, 3 components, a sampling rate of 50 Hz and one-year-long period of data with a temporal step of one sample, it would take:

- with our GPU code running on 4 GPUs:

$$7.95\text{s} \times 365\text{days} = 2901\text{s} = 48\text{min}20\text{s}$$

- with our CPU code running on 24 cores:

$$136.48\text{s} \times 365\text{days} = 49,815\text{s} = 13\text{h}50\text{min}$$

5. Application to Real Data

We present here an application to real data, where we use one template representing near-repeating seismic events from a source in the Western Alps, France to scan one day of data. The data come from the temporary experiment CIFALPS (China-Italy-France Alps Seismic Survey [CIFALPS, 2012-2013; Zhao et al., 2015]) that involved 55 broadband velocimeters and also from 32 permanent stations (from French and Italian networks). Figure 6 summarizes this matched-filter search with the template's location, the correlation coefficients we computed and some example detections that were found with the template. The correlation coefficient computation is turned into an earthquake detector by establishing a threshold of $10 \times \mathrm{MAD}\{\mathrm{CC}(t)\}$ (MAD stands for Median Absolute Deviation). All time windows with a CC greater than the threshold correspond to detections of earthquakes with waveforms similar to those of the template. We only show a single channel on Figure 6, but the template used in the matched-filter search is composed of waveforms on 12 stations and 3 components. The data was downsampled to 50 Hz and filtered in the band 1.5-25 Hz, thus corresponding to the one-template test case shown in Figure 5.

6. Conclusion

We provide an open-source code called FMF to efficiently compute the normalized correlation coefficient between template waveforms and continuous seismic records to achieve earthquake detection. Our open-source code comes with two implementations: a version written in C working with Python or Matlab wrappers that is designed to run on any multi-core computer, and a second version written in CUDA C working with a Python wrapper that is designed to run on computers with an Nvidia GPU.

FMF is faster than parallel codes written in high-level languages like Matlab or Python. Such performance enables large-scale detection via template matched-filtering in a reasonable amount time. For example, running 50 8-second long templates with waveforms on 12 stations, 3 components on a one-year period of data with sampling rate 50 Hz takes 48min with the GPU version running on 4 GPUs and 14h with the CPU version running on 24 cores. The use of GPUs dramatically decreases the computation time, and

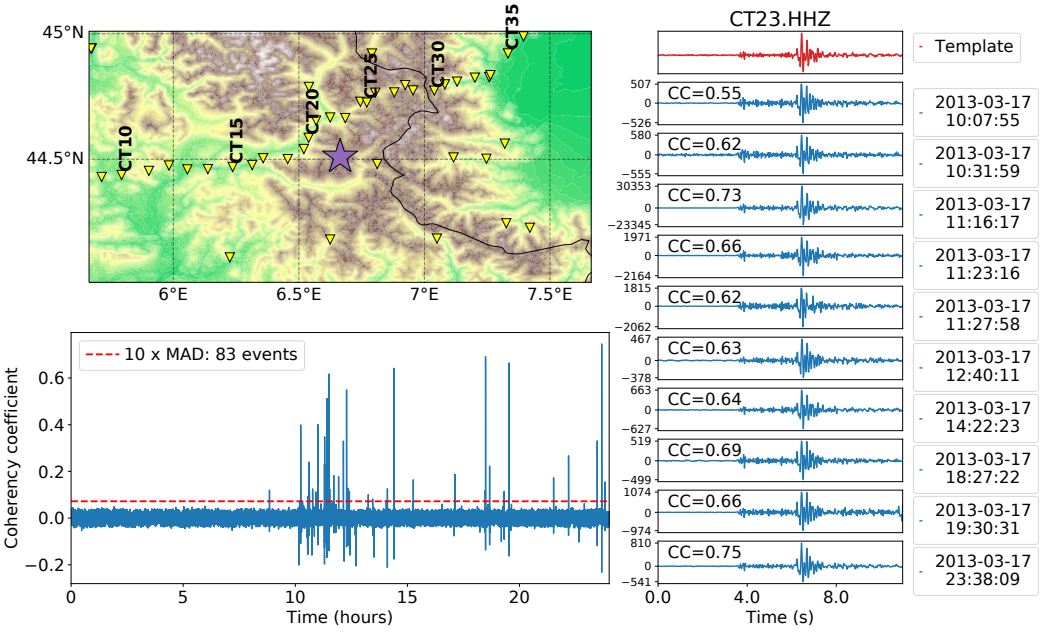

Figure 6: Matched-filter search of one template in the Western Alps, France over a single day. **Top left panel:** Map of the region showing the template's location (star) and the neighboring stations (inverted triangles). **Bottom left panel:** Correlation coefficients (Equation 2) over the whole day. The threshold of $10 \times \mathrm{MAD}\{\mathrm{CC}(t)\}$ is plotted with the *dashed line*. A total of 83 earthquakes was detected with constraining each event to be separated by at least 3 seconds. **Right panel:** Template's channel CT23.HHZ (top record) and 10 examples of detections on this same channel.

allows to efficient performances with relatively low cost hardware: our test machine cost about US$15,000 in 2015.

Processing large volumes of data is an important challenge when the number of instruments and data are rapidly increasing. We show here how our program can be used to systematically enhance existing catalogs of seismic events. Finally, we hope that the distribution of this software will help make detection studies more easily reproducible by different groups within the scientific community.

Data and resources

Our program FMF is available at `https://github.com/beridel/fast_matched_filter`. The topographic data used to generate the map in Figure 6 was extracted from the SRTM 90m database (`http://www.cgiar-csi.org/data/srtm-90m-digital-elevation-database-v4-1`). The data used in the example come from the CIFALPS temporary array (China-Italy-Alps Seismic Survey, doi=`http://dx.doi.org/10.15778/RESIF.YP2012`), and some permanents stations belonging to French networks (FR, RD) and Italian networks (GU, IV, MN, MT). EQcorrscan is a Python package for the detection and analysis of repeating and near-repeating seismicity, we used the version 0.2.7 to run our tests (`https://doi.org/10.5281/zenodo.893621`).

Acknowledgements

We thank Nikolaï Shapiro for the inspiration behind our adventure into the world of GPUs. W.B.F. was supported by NSF grant EAR-PF 1452375 and E.B. by the Theodore R. Madden Fellowship.

References

Allen, R., 1982. Automatic phase pickers: their present use and future prospects. Bulletin of the Seismological Society of America 72 (6B), S225–S242.

Chamberlain, C. J., Hoop, C. J., Boese, C. M., Warren-Smith, E., Chambers, D., Chu, S. X., Michailos, K., Townend, J., 2017. Eqcorrscan: Repeating and near-repeating earthquake detection and analysis in python. Seismological Research Letters *in review*.

CIFALPS, 2012-2013. China-Italy-France seismic survey: 55 broadband velocimeters.

Dagum, L., Menon, R., Jan. 1998. Openmp: An industry-standard api for shared-memory programming. IEEE Comput. Sci. Eng. 5 (1), 46–55. URL `https://doi.org/10.1109/99.660313`

Dziewonski, A. M., Anderson, D. L., 1981. Preliminary reference earth model. Physics of the earth and planetary interiors 25 (4), 297–356.

366 Frank, W. B., 2016. Slow slip hidden in the noise: The intermittence of
367 tectonic release. Geophysical Research Letters 43 (19).

368 Frank, W. B., Poli, P., Perfettini, H., 2017. Mapping the rheology of the Cen-
369 tral Chile subduction zone with aftershocks. Geophysical Research Letters
370 44 (11), 5374–5382, 2016GL072288.
371 URL http://dx.doi.org/10.1002/2016GL072288

372 Frank, W. B., Shapiro, N. M., Husker, A. L., Kostoglodov, V., Gusev, A. A.,
373 Campillo, M., 2016. The evolving interaction of low-frequency earthquakes
374 during transient slip. Science advances 2 (4), e1501616.

375 Frank, W. B., Shapiro, N. M., Husker, A. L., Kostoglodov, V., Romanenko,
376 A., Campillo, M., 2014. Using systematically characterized low-frequency
377 earthquakes as a fault probe in Guerrero, Mexico. Journal of Geophysical
378 Research: Solid Earth 119 (10), 7686–7700.

379 Gibbons, S. J., Ringdal, F., 2006. The detection of low magnitude seismic
380 events using array-based waveform correlation. Geophysical Journal Inter-
381 national 165 (1), 149–166.

382 Gutenberg, B., Richter, C. F., 1944. Frequency of earthquakes in california.
383 Bulletin of the Seismological Society of America 34 (4), 185–188.

384 Huang, H., Xu, W., Meng, L., Bürgmann, R., Baez, J. C., 2017. Early after-
385 shocks and afterslip surrounding the 2015 Mw 8.4 Illapel rupture. Earth
386 and Planetary Science Letters 457, 282–291.

387 Kwiatek, G., Ben-Zion, Y., 2016. Theoretical limits on detection and analysis
388 of small earthquakes. Journal of Geophysical Research: Solid Earth 121 (8),
389 5898–5916.

390 Mu, D., Lee, E.-J., Po, C., 2017. Rapid earthquake detection through gpu-
391 based template matching. Computers and Geoscience.

392 Nickolls, J., Buck, I., Garland, M., Skadron, K., Mar. 2008. Scalable parallel
393 programming with cuda. Queue 6 (2), 40–53.
394 URL http://doi.acm.org/10.1145/1365490.1365500

395 Perfettini, H., Avouac, J.-P., 2004. Postseismic relaxation driven by brittle
396 creep: A possible mechanism to reconcile geodetic measurements and the

decay rate of aftershocks, application to the Chi-Chi earthquake, Taiwan. Journal of Geophysical Research: Solid Earth 109 (B2).

Rogers, G., Dragert, H., 2003. Episodic tremor and slip on the cascadia subduction zone: The chatter of silent slip. Science 300 (5627), 1942–1943.

Shelly, D. R., Beroza, G. C., Ide, S., 2007. Non-volcanic tremor and low-frequency earthquake swarms. Nature 446 (7133), 305.

Utsu, T., 1961. A statistical study on the occurrence of aftershocks. Geophys. Mag. 30, 521–605.

Van der Hilst, R., Widiyantoro, S., Engdahl, E., 1997. Evidence for deep mantle circulation from global tomography. Nature 386, 578–584.

Warren-Smith, E., Chamberlain, C. J., Lamb, S., Townend, J., 2017. High-Precision Analysis of an Aftershock Sequence Using Matched-Filter Detection: The 4 May 2015 M L 6 Wanaka Earthquake, Southern Alps, New Zealand. Seismological Research Letters.

Zhao, L., Paul, A., Guillot, S., Solarino, S., Malusà, M. G., Zheng, T., Aubert, C., Salimbeni, S., Dumont, T., Schwartz, S., et al., 2015. First seismic evidence for continental subduction beneath the Western Alps. Geology 43 (9), 815–818.

View publication stats