Parallelizing Accelerographic Records Processing

Ronaldo Canizales
Department of Computer Science
Colorado State University
Colorado, USA

Luis Mixco
Observatorio de Amenazas y Recursos Naturales
Ministerio de Medio Ambiente y Recursos Naturales
San Salvador, El Salvador

Jedidiah McClurg
Department of Computer Science
Colorado State University
Colorado, USA

***Abstract*—Strong-motion processing holds paramount importance in earthquake engineering and disaster risk management systems. By leveraging parallel loops and task-parallelism techniques, we address computational challenges posed by large-scale accelerographic datasets. Through experimentation with more than one million data points from six real-world seismic events, our approach achieved speedups of up to 2.9x, demonstrating the effectiveness of parallel programming in accelerating seismic data processing. Our findings highlight the significance of parallel programming techniques in advancing seismological research and enhancing earthquake mitigation strategies.**

***Index Terms*—strong-motion data, parallel processing, natural disaster management, OpenMP, Fortran, C++**

I. Introduction

Earthquakes are well-known for their unpredictability and potential to cause significant damage. These events have profound and far-reaching impacts on both human societies and the environment. One of the most devastating earthquakes in recent history occurred in 2004 in Sumatra, Indonesia, triggering a massive tsunami that affected the entire Indian Ocean region, and resulting in approximately 230,000 fatalities [13]. The 2015 Nepal earthquake caused significant damage to the Himalayan region, with economic losses amounting to one-third of its gross domestic product for that year [19]. Similarly, in February 2023, a powerful earthquake struck southeastern Turkey and Syria, causing over 50,000 fatalities and widespread destruction [11]. Furthermore, seismic events such as the Gyeongju and Pohang earthquakes in 2016 and 2017 in the Korean Peninsula have underscored the importance of *ground-motion data collection and analysis* for ensuring the stability of critical infrastructure such as nuclear power plants [20], and have highlighted the critical need for seismic hazard assessment and disaster risk reduction strategies in earthquake-prone areas, such as the Indian subcontinent [18].

Earthquake-resistant structural design is primarily concerned with the tradeoff between potential for local earthquake activity levels and the ability of structures to resist damage, and relies on seismic data collection and analysis. With numerous data analysis schemes available, ensuring accurate interpretation and utilization of the data is crucial [3]. Large seismic monitoring networks generate vast amounts of data that require efficient archiving, dissemination, visualization, and real-time analysis [6]. These records, obtained from *strong-motion accelerograph machines*, are indispensable for hazard estimation and site-effect studies, forming the backbone of seismic research [18]. *Processing strong ground motion data efficiently can mitigate seismic hazards and gain valuable insights into site-specific effects [5].*

Fig. 1. Seismic station and accelerograph [26, 22]

In this paper, we explore a parallelization-based approach for improving the performance of vital strong-motion processing software used in El Salvador. Although various techniques for parallelizing generic programs exist, these approaches are not straightforward to use in the context of real-world seismological software because of complex process dependencies, legacy programming languages and APIs, mixed input/output data formats, and intrinsically distributed data sources.

Specifically, we focus on software used in El Salvador's Observatory of Natural Threats. In this context, despite using modern technology for sensing, distributing, storing, and visualizing seismic-related data, the core calculations needed for processing strong-motion records are performed using legacy Fortran code. In this paper, we show how to perform a careful analysis of input/output data dependencies, and apply techniques such as parallelizing loops and adding task-parallelism, producing an optimized version of the existing sequential implementation.

To our knowledge, our approach is the first to fully parallelize the processing of accelerographic records when all three signal components are analyzed, leveraging the robust Salvadoran strong-motion sensor network. We demonstrate the utility of our approach by processing data from 6 strong-motion events in El Salvador. On this data, our work outperforms the original sequential implementation by a factor of 2.6x to 2.9x, benefiting seismology research and natural disaster management, and aiding impact mitigation on society.

II. Background: Accelerographic Records Processing

When a seismic event occurs, proximal strong-motion sensors register its effects – Figure 1 shows an example of such sensors. Each sensor produces data files capturing three distinct components: longitudinal, transversal, and vertical motions. These raw data files, denoted as uncorrected versions, are stored with a *V1* extension, and contain the ground's acceleration, velocity, and displacement over a defined temporal window. The number of files needing to be processed per seismic event depends on nearby sensors' availability and the event's magnitude. In the following, we will outline the processing steps for each file.

The components of the accelerographic data (e.g., Figure 2) are individually stored in files named according to the format $[station][comp]$.v1. Subsequently, a Hamming band-pass filter is applied to each component, utilizing default parameters. This processing step generates corrected versions of the signals, which are then saved with a *V2* extension.

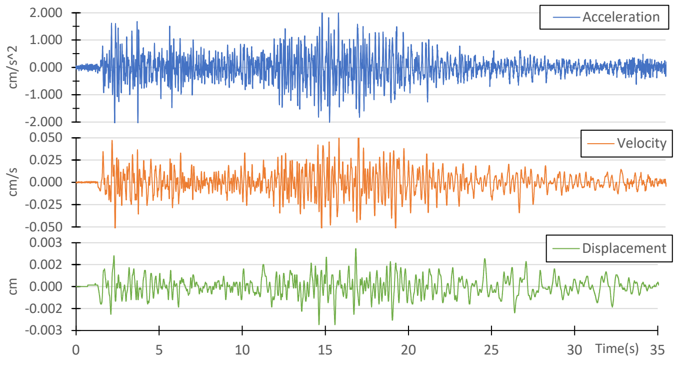

Fig. 2. Accelerographic data (one component)

Following this correction, a Fourier transformation is performed on each signal (e.g, Figure 3) resulting in files marked with an *F* extension. Notably, the velocity Fourier spectrum of each signal holds significant importance, as its analysis yields the low-pass frequency (FPL) and low-stop frequency (FSL) parameters, highlighted in red in Figure 3. These are used in the definitive acceleration baseline correction of each signal. Moreover, the peak ground acceleration (PGA) values are extracted and archived.

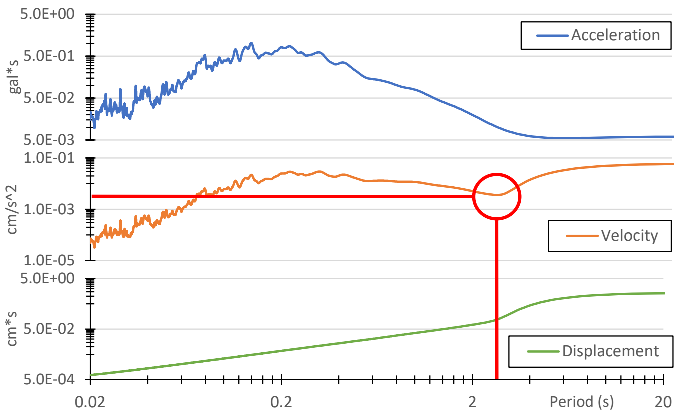

Fig. 3. Inflection point in the velocity Fourier spectrum (one component), indicates the values of the FPL & FSL signal filtering parameters.

The final corrected signals are derived by applying another Hamming band-pass filter, tailored with the appropriate FPL and FSL parameters obtained from the Fourier analysis, and are subsequently stored in *V2* files.

The most computationally intensive operation involves calculating the Response spectrum for each corrected signal (e.g., Figure 4), saved with an *R* extension. This spectrum provides valuable insights into the response characteristics of various building types during seismic events.

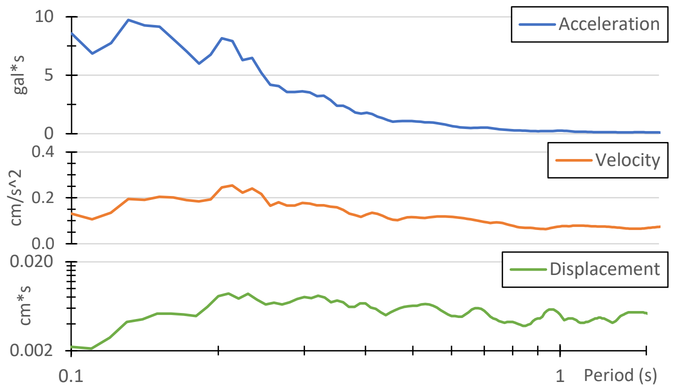

Fig. 4. Response spectrum (one component)

Finally, 18 Global Earthquake Model (*GEM*) files are created from the *V2* and *R* files. These serve as crucial inputs for subsequent processes. This information resulting from this processing of strong-motion files is of considerable significance to structural engineers during the design phase of new buildings, informing their decisions and enhancing structural integrity.

III. Sequential Implementation

The entire process used in the sequential version is segmented into 20 sequential steps, shown in Figure 5. Each step, which we will refer to as a *process*, is either a function embedded within C++ code, or an entire Fortran program.

While certain processes are lightweight, others entail substantial input/output operations, calculations, or plotting tasks. The following section describes how to optimize this sequential implementation (Figure 6).

IV. Optimizing the Sequential Implementation

Initial optimization involves the elimination of unnecessary processes. Detailed analysis of the process uncovered the following redundancies.

1) The plotting of uncorrected signals (process #6) is unnecessary, as the generated plots are not utilized within the program. Additionally, the $[station]$.ps files produced are subsequently overwritten in process #15.
2) Segmentation of each component of uncorrected signals into individual files (process #12) is superfluous, since no modifications are applied to *V1* files during execution.
3) Overwriting intermediate files (process #14) is redundant, as these files mirror those obtained earlier in process #5.

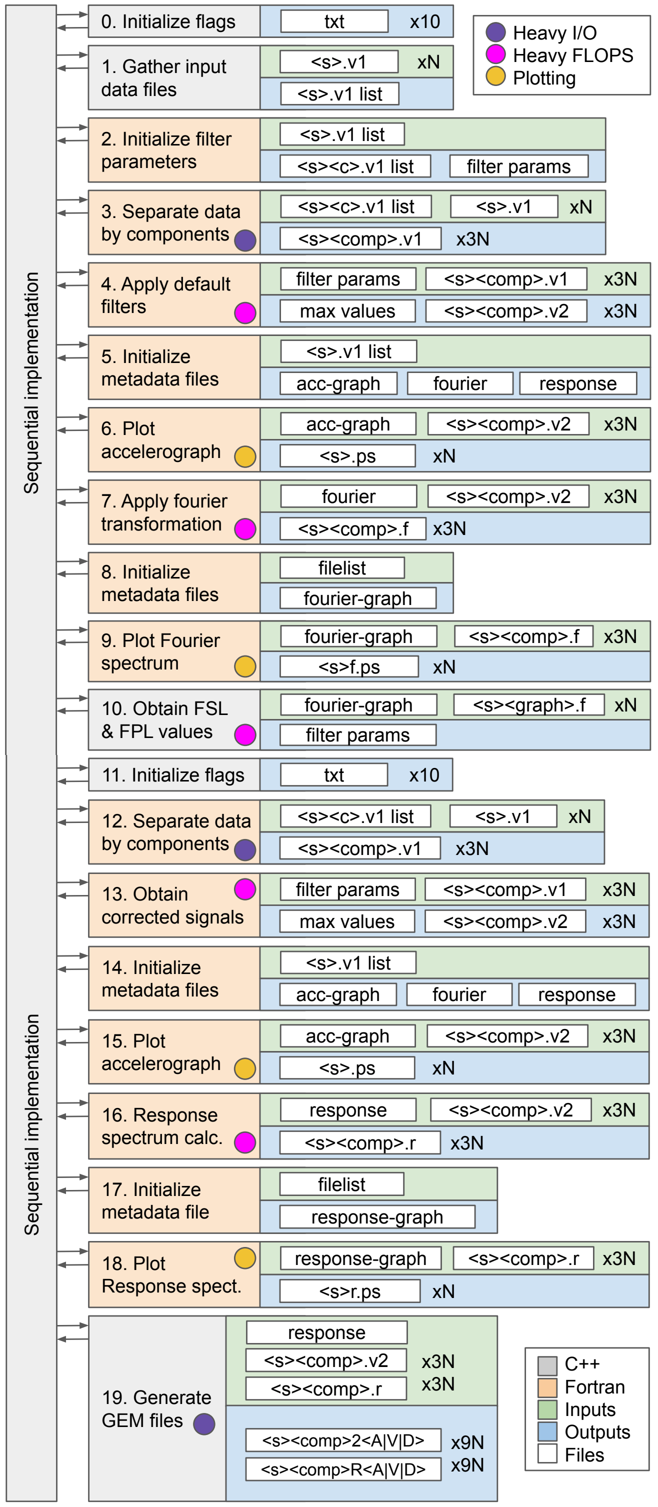

Fig. 5. Sequential accelerographic records processing implementation

In summary, the exclusion of processes #6, #12, and #14 is feasible, and has no impact on the final output. This optimization reduces the overall execution time and avoids overwriting data. The resulting optimized sequential version is shown in Figure 7.

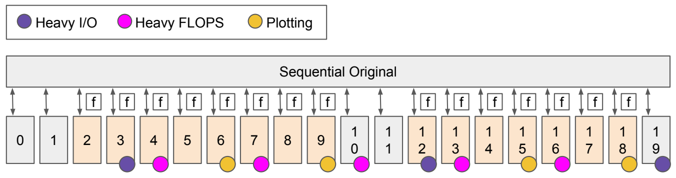

Fig. 6. Original sequential version: processes overview

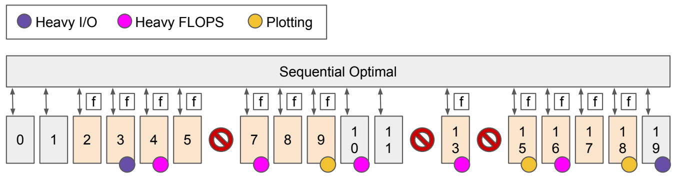

Fig. 7. Optimized sequential version: processes overview

V. Partially Parallelized Implementation

Establishing the foundation for parallelizing the workflow involves delineating all processes and their dependencies, encompassing both inputs and outputs, as shown in Figure 9. These processes have been categorized into nine stages, with each stage slated for parallelization employing the specific strategies outlined on the right-hand side of the figure (we will elaborate further on these strategies shortly). We analyzed the reordering of processes to ensure that it upholds valid execution sequences while maximizing processor utilization across each stage.

We employed three distinct parallelization strategies, delineated as follows.

1) Processes #0, #1, #10, and #19 are exclusively implemented in C++. Thus, the initial focus was on parallelizing these processes. Detailed explanations of how this was performed are furnished in subsequent sections.
2) Processes #2, #5, #8, and #17 are implemented in Fortran, and exhibit minimal execution times, prompting their parallelization through C++ OpenMP tasks.
3) Processes #9, #15, and #18, also implemented in Fortran, are characterized by the independent processing of distinct signal types, and plot generation. Consequently, these tasks were parallelized using C++ OpenMP tasks.

Using this strategy, we were able to parallelize 5 out of 11 stages, as depicted in Figure 8. The specifics of the parallelization for each stage are detailed below.

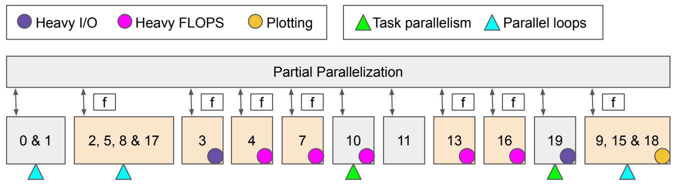

Fig. 8. Partial parallelization: 5 out of 11 stages are executed in parallel.

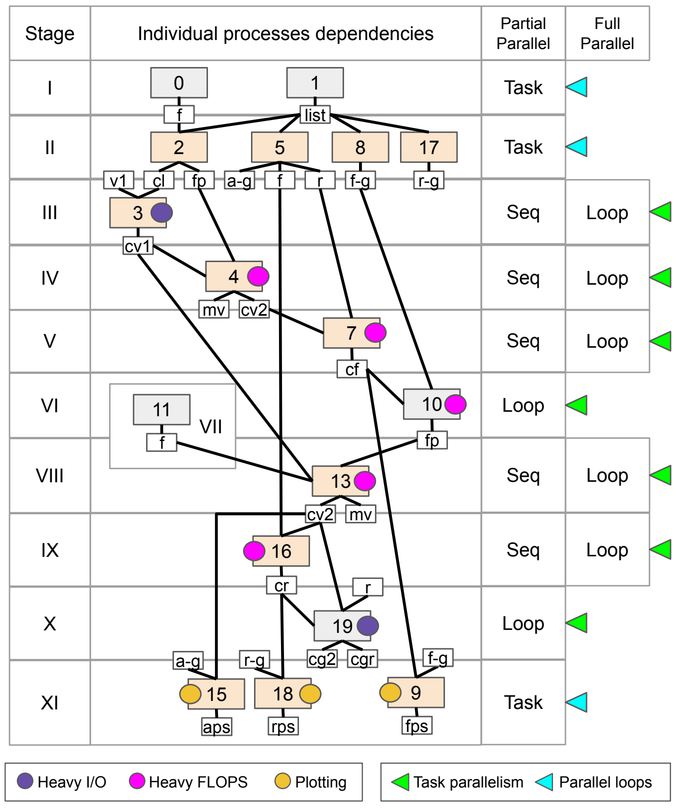

Fig. 9. Processes reordering into 11 stages. Parallel approaches for each individual stage (rows) for the first and second parallel versions (right-most columns) are detailed, as well as input/output dependencies for each process.

A. Parallelizing Stages I and II

Since a limited number of equally lightweight processes are involved, we chose a task-parallelization approach. A parallel pragma is configured to utilize between 2 and 4 processors for task execution. The single pragma restricts task assignment to only the main processor. A task-wait pragma is necessary to synchronize the execution between both stages.

In Stage I, processes operate independently and do not require inputs. In Stage II, all processes create metadata files used in further stages based on the $[station][comp]$.v1 files list generated during Stage I.

```
#pragma omp parallel{
    #pragma omp single{
        #pragma omp task//Stage I processes:
        InitializeFlags();//P#0
        #pragma omp task
        GatherInputFiles();//P#1
        #pragma omp taskwait

        #pragma omp task//Stage II processes:
        InitializeFilterParams();                 //P#2
        #pragma omp task
        InitMetadata(<acc-graph>,<fou>,<res>);//P#5
        #pragma omp task
        InitMetadata(<fourier-graph>);            //P#8
        #pragma omp task
        InitMetadata(<response-graph>);          //P#17
        #pragma omp taskwait
}    }
```

B. Parallelizing Stage VI

Process #10 identifies inflection points within the velocity Fourier spectrum for each of the three components. Concurrent analysis is conducted for each component.

`CalculateInflectionPoint` within the parallel for-loop employs an early-termination strategy while searching for slope changes in data points for periods greater than one second. Although additional parallelization opportunities exist, they were not pursued, since execution time was already small.

```
void AnalyzeFourier(){
    //Read metadata (fourier-graph file)
    for(int i=0; i<N; i++){
        string files[3];//Input: <comp>.f files
        float fsl[3], fpl[3];
        //Analyze L, T, and V plots
        #pragma omp parallel for
        for(int j=0; j<3; j++)
            CalculateInflectionPoint(
                files[j], fsl[j], fpl[j]);
        //Output: FSL & FPL values (filter param)
}    }
```

C. Parallelizing Stage X

Process #19 generates 18 *GEM* files for each input file. Specifically, six files are generated per *V2* and *R* file pair. Concurrent reading of each batch of files is facilitated in this stage, leveraging the maximum number of available processors, given the typically substantial quantity of *GEM* files produced.

The function `SetDataApart` operates on each *V2* or *R* file within the parallel for-loop, generating three corresponding *GEM* files. This parallelized approach optimizes efficiency by distributing the workload across all available processors.

```
void GenerateGEMFiles(){
    //Read metadata (response file)
    string files[N*2];
    for(int i=0; i<N; i++){
        files[i*2] = //Input: <s>.v2 files
        files[i*2+1] = //Input: <s>.r files
    }
    #pragma omp parallel for
    for(int i=0; i<N*2; i++){
        bool isR = //flag (odd/even)
        SetDataApart(files[i], isR);
        //Output: <s><comp>GEM<2|R><A|V|D> files
}    }
```

D. Parallelizing Stage XI

These plotting processes operate independently from one another and operate on different input data. They receive *V2*, *F*, and *R* files as inputs and generate corresponding plot files, namely $[station]$.ps, $[station]$f.ps, and $[station]$r.ps.

```
#pragma omp parallel{
    #pragma omp single{
        #pragma omp task//Stage XI processes:
        PlotFourierSpectrum();//P#9
        #pragma omp task
        PlotAccelerograph();//P#15
        #pragma omp task
        PlotResponseSpectrum();//P#18
        #pragma omp taskwait
}    }
```

Expanding on this, the *V2* files contain corrected signals, the *F* files store Fourier-transformed signals, and the *R* files comprise Response spectrum data. The resultant plots provide visual representations of the analyzed accelerographic data, aiding in interpreting and understanding the seismic event under investigation.

VI. Fully Parallelized Implementation

The parallelization of the remaining processes requires either Fortran OpenMP pragmas or executing binary files concurrently within temporary folders. This implies the transfer of all input and output data to and from these folders. We employed two distinct approaches, outlined as follows:

- The parallelization approach for Processes #3 and #16 is via Fortran OpenMP do-loops, enabling efficient concurrency within the existing Fortran codebase.
- Concurrent execution of processes #4, #7, and #13 inside temporary folders with required data transfer is achieved through C++ OpenMP for-loops.

All processes except #11 are parallelized (Fig. 10), due to its execution time being less than two milliseconds on average. We describe specific approaches used in each stage.

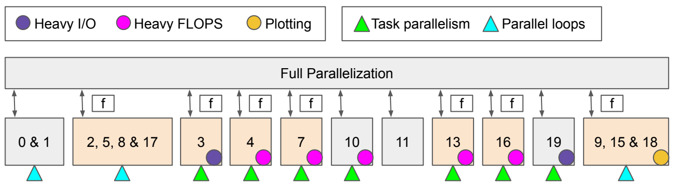

Fig. 10. Full parallelization: all stages are executed in parallel.

A. Parallelizing Stage III

Process #3 concurrently collects uncorrected acceleration data from all $[station]$.v1 files and generates an individual file $[station][comp]$.v1 for each component.

This task is accomplished using Fortran OpenMP pragmas, including `omp parallel` and `omp do`, which leverage all available processors for enhanced parallel execution efficiency.

```
c   Read data (<station><comp>.v1 list file)
C$OMP PARALLEL
C$OMP DO PRIVATE(<variables>)
do i=1,N
c   Read acceleration data
open(unit=i*4,file=..,status='old') -> Input:<s>.v1
c   Write acceleration data per component
open(unit=i*4+1,file=...) -> Output: <s>l.v1
open(unit=i*4+2,file=...) -> Output: <s>t.v1
open(unit=i*4+3,file=...) -> Output: <s>v.v1
end do
C$OMP END DO
C$OMP END PARALLEL
```

B. Parallelizing Stage IX

Process #16 starts by extracting metadata from the response file, that is, the list of individual component $[station][comp]$.v2 file names. Then, each file is read in parallel. The elastic response spectra for acceleration, velocity and displacement are calculated for each component.

It is important to note that this process has a sequential complexity of $O(9000 * N * D^2)$ where N is the number of input *V1* files, and D is the average number of individual data points contained in each *V1* file.

Lastly, output data is stored in individual $[station][comp]$.r files for each component. The response spectrum data is used in processes #18 and #19, which generate plots and create individual *GEM* files, respectively.

This parallelization is particularly significant for this process, because it not only takes the longest to execute, but also achieves the highest speedup, as shown in Figure 11 of the experimental results. We use all available processors to maximize efficiency in this optimization effort.

```
c   Read metadata (response file)
C$OMP PARALLEL
C$OMP DO PRIVATE(<variables>)
do i=1,<3N> -> Each component in parallel
c   Read acceleration data -> Input: <s><comp>.v2
open(unit=2*i,file=...,status='old')
c   Calculate Response Spectrum
c   Write output data
open(unit=2*i+1,file=...) -> Output: <s><comp>.r
end do
C$OMP END DO
C$OMP END PARALLEL
```

C. Parallelizing Stages IV and VIII

Processes #4 and #13 share the same functionality. Specifically, they read the filter parameters file to extract the values of FPL and FSL, and take all the individual $[station][comp]$.v1 files as input. Subsequently, a Hamming band-pass filter is applied to each signal, resulting in a corrected version of the acceleration values. These corrected values are then stored in individual $[station][comp]$.v2 files.

```
void ParallelizeCorrection(){
  //Read data (filter params file)
  string files[N*10];
  for (int i = 0; i < N; i++)
      files[10*i] = //Input: <s><comp>.v1 files
      for (int j = 0; j < 3; j++)
          for (int k = 1; k <= 3; k++)
              getline(i_data, files[10*i+3*j+k]);
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
      //Create temp 10*i folder and params file
      for (int j = 0; j < 3; j++)
          for (int k = 1; k <= 3; k++)
              //Move 10*i+3*j+k <s><comp>.v1 file
  for (int i = 0; i < N; i++)//Seq. to avoid races
      //Move EXE to 10*i folder
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
      for (int j = 0; j < 3; j++)
          //Input: Move 10*i+3*j+2 <s>.v1 file
      //Apply filters to signals on 10*i folder
      for (int j = 0; j < 3; j++)
          //Output: Move 10*i+3*j+3 <s>.v2 file
  //Output: (max values file)
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
      //Delete remaining temp files
}
```

However, the parallelization strategy for these processes differs significantly from the approaches used in the other stages. Due to the impracticality of modifying the original Fortran programs, multiple instances are executed concurrently within separate folders. The primary task involves creating these folders with all the necessary files, and subsequently copying the results back. This approach maximizes processor utilization by harnessing all available processors.

D. Parallelizing Stage V

Process #7 is responsible for computing the Fourier spectra for acceleration, velocity, and displacement of each component stored in all $[station][comp]$.v2 files, with the results saved in $[station][comp]$.f files.

The parallelization strategy employed for this process mirrors that of Stages IV and VIII, as detailed in the preceding section. All available processors are utilized to optimize computational efficiency.

```
void ParallelizeFourier(){
  //Read data (fourier file)
  string files[N*3];
  for (int i = 0; i < N; i++)
      files[3*i] = //Input: Folder name
      files[3*i+1] = //Input: <s><comp>.v2 files
      files[3*i+2] = //Input: <s><comp>.f files
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
      //Create temp 3*i folder & fourier file
  for (int i = 0; i < N; i++)//Seq. to avoid races
      //Move EXE to 3*i folder
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
      //Input: Move 3*i+1 <s><comp>.v2 file
      //Apply Fourier Transform on 3*i folder
      //Output: Move 3*i+2 <s><comp>.f file
      //Delete remaining temp files
}
```

VII. Experiments and Results

A. Experimental Setup

Our experimental dataset comprises 71 unprocessed accelerograph files from 6 seismic events that occurred in El Salvador over the past decade. These events represent a diversity of stations affected, and have varying numbers of data points within each raw file, ranging from 7,300 to 35,000.

The original sequential implementation consisted of 2,297 lines of Fortran code. Our fully-parallelized version was implemented with 653 lines of C++ code and an additional 337 lines of Fortran code.

The experimental platform used for the experiments was a 12th Gen Intel Core i5-12450H, 2.00 GHz, 8 Cores, 12 Logical Processors, 16 GB of RAM, 640 KB of L1 cache, 6.2 MB of L2 cache, and 12 MB of L3 cache.

B. Results per Stage

For this experiment, we assessed the parallel performance of each individual stage using data from the seismic event with the most data in the experimental dataset, i.e., $384,000$ data points distributed across 19 *V1* files.

Figure 11 illustrates that stage IX exhibits the longest execution time among all stages, accounting for 57.2% of the original sequential implementation. However, Stage IX also achieves the highest speedup, reaching 5.14x.

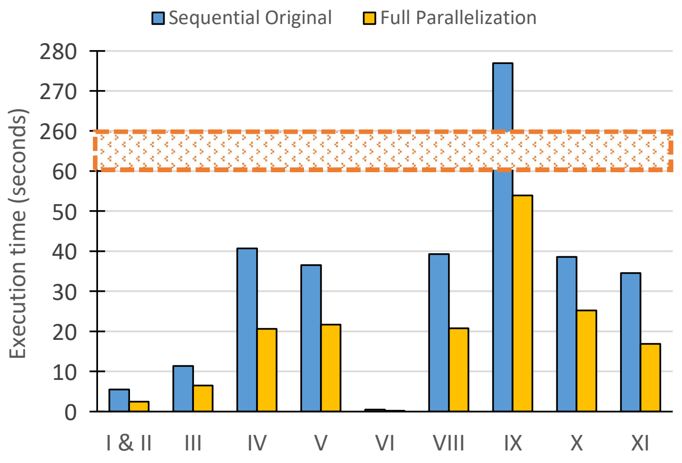

Fig. 11. Speedup per individual Stage (19 Files, 384k Data points)

Other parallel stages successfully reduce the sequential execution time by more than half, achieving speedups of 2.2x, 2.0x, 2.6x, and 2.1x for stages I-II, IV, VI, and XI, respectively. Furthermore, the remaining parallel stages achieve speedups of 1.8x, 1.7x, 1.9x, and 1.5x for stages III, V, VIII, and X, respectively. Overall, for this event, the speedup stands at 2.88x, as indicated in the bottom row of Table I.

TABLE I
Experimental Results

Event	V1 Files	Data Points	Seq. Ori.*	Seq. Opt.*	Part. Par.*	Full Par.*	Speed Up
Nov'18	5	56K	76.6	64.1	61.9	32.1	2.39x
Apr'18	5	115K	149.6	127.1	126.4	56.5	2.65x
Jul'19	9	145K	174.9	161.3	154.8	68.1	2.57x
Apr'17	15	309K	358.6	351.2	327.9	131.5	2.73x
May'19	18	361K	439.5	392.6	378.9	155.3	2.83x
Jul'19	19	384K	483.7	426.0	412.2	168.1	2.88x

*Execution times are measured in seconds.

C. Results per Event

All available uncorrected accelerographs (*V1* files) for each seismic event in the experimental dataset were processed using the four implementations explained in sections III to VI:

- *Sequential Original*: contains 20 sequential processes.
- *Sequential Optimized*: contains 17 sequential processes.
- *Partially Parallelized*: utilizes 5 parallel stages.
- *Fully Parallelized*: utilizes 10 parallel stages.

Figure 12 shows a visual representation of each implementation's execution times, and full data is shown in Table I. Execution time is linearly proportional to the total amount of data points. While each implementation offers a performance improvement over its predecessor, the fully-parallelized version demonstrates the most efficiency.

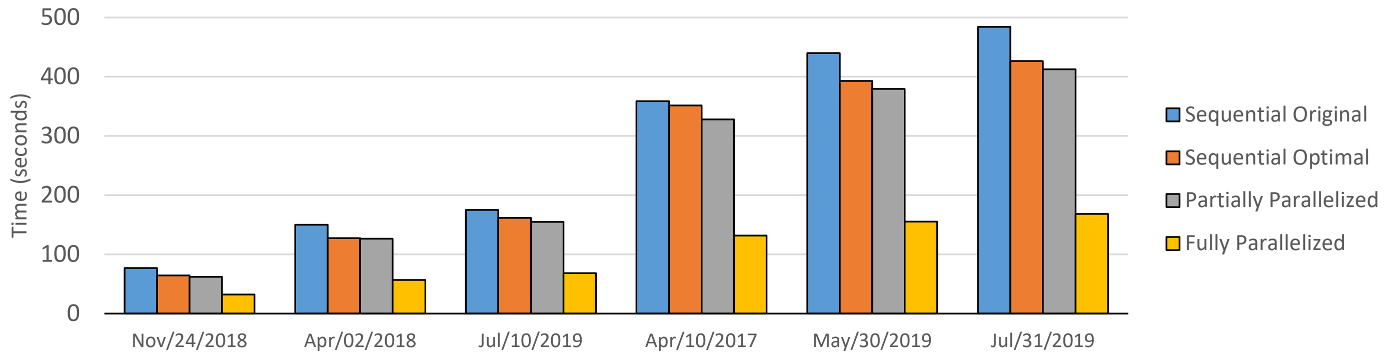

Fig. 12. Speedup per Event: execution time is proportional to the number of data points in each seismic event.

We observe that the speedup increases proportionally with the number of total input data points, as illustrated in Figure 13. The overall speedup ranges from 2.4x to 2.9x and appears to follow a quasi-logarithmic trend (Amdahl's effect).

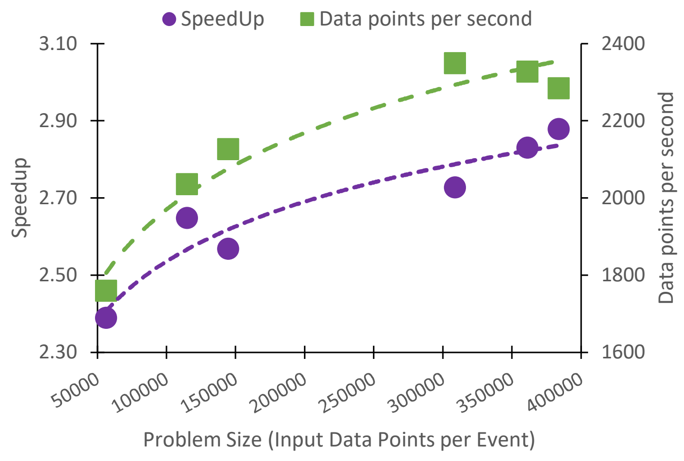

Fig. 13. Overall speedup in parallel accelerographic records processing: problem size vs. speedup (purple) and vs. data points per second (green).

The fully-parallelized implementation was able to process between 1,700 to 2,300 data points per second, a significant improvement over the original sequential version, which processed 800 data points per second on average. This improvement also follows a quasi-logarithmic trend on the problem size, as shown in Figure 13.

VIII. Discussion & Future Work

Although the number of seismic events contained in our experimental dataset is relatively small, it offers a comprehensive range of real-world seismic data, generated in different geographic locations with a variety of equipment types and sampling rates. Although we used a relatively low-end machine for the experiments, and performance may be further improved on a higher-performance machine, our experimental platform is reflective of the types of machines used in practice in the Salvadoran Observatory of Natural Threats. It is worth noting that the engineering work invested in parallelizing each stage does not always directly translate to performance gains. Despite similar levels of effort, Stage IX provides the most significant improvements. Additionally, different stages leverage available processors differently, with some using only a fraction while others fully utilize all available resources.

Our findings indicate potential for leveraging tools like OpenMP to enhance Accelerographic Records Processing, though communication overhead remains a concern. We also observed similarities in array management techniques across stages IV, VIII, and V, resembling principles seen in MPI or CUDA programming.

While we believe our approach to be an important step toward the high-level goal of scaling up strong-motion records processing, especially in regard to large-scale real-world datasets, there are engineering and research challenges we plan to address in future work. Enhancing strong-motion record processing could involve automatic translation of diverse legacy code into a single programming language, such as C++, and using search-based techniques [25, 10, 24] to assist in parallelization of the code, which could streamline development and improve overall efficiency. With adequate time and resources, similar enhancements could extend to other seismic-related applications, as well as processes related to other natural threats, like vulcanology and landslides.

Additionally, there is room for further optimization through the exploration of advanced parallel processing techniques like tiling, wavefront scheduling, and the polyhedral model. Furthermore, scaling our approach to larger experimental accelerographic datasets presents an exciting opportunity to assess its performance and scalability in real-world scenarios.

IX. Related Work

A. *Strong-Motion Records Databases*

Several initiatives worldwide have been dedicated to collecting and disseminating strong-motion data to advance seismic research and hazard assessment. For instance, the ITACA project focused on gathering, standardizing, and sharing strong motion data acquired in Italy since 1972. By 2010, this database encompassed 7,038 waveforms from analog and digital instruments recorded during 1,019 earthquakes with magnitudes reaching 6.9 [23]. Similarly, the Salvadoran Accelerographic Repository stores 6,787 strong motion records from 1,615 seismic events between the years 1966 and 2019. It scales up quickly; the most recent report shows 241 seismic events recorded just during December 2023 [21]. Another example is the Indian Strong Motion Instrumentation Network, with around 220 accelerograph stations, which provides data from approximately 300 strong ground motion records from 130 earthquakes [18]. Furthermore, a comprehensive database has been established to study induced earthquakes in the Groningen gas field, Netherlands. This repository houses over 8,500 processed ground motion recordings from 87 earthquakes, serving as a resource for refining seismic hazard and risk models and conducting research in site response and ground motion characteristics [27].

B. Earthquake Engineering and Urban Planning

Despite efforts to enhance safety through urban planning, indiscriminate development and structural failures persist, leading to significant loss of life and economic damage [1]. To address these challenges, there is a need to prioritize the development of resilient buildings and emergency response efforts by mapping vulnerable urban areas and populations. While progress has been made in evaluating individual facilities and distributed systems, greater integration and collaboration are necessary to improve overall seismic resilience [34]. In El Salvador, the original sequential code discussed in this paper was a component of a large project that united academia, industry, and governmental agencies with the goal of updating building design codes. Various software tools are utilized in earthquake engineering and seismic risk assessment, enabling different aspects of the analysis process. High-performance computing clusters enable parallel job configurations for handling buildings independently [34]. Neural networks like the multi-layer perceptron assess urban block vulnerability to earthquakes [1]. Additionally, software tools such as Obspy [16, 7] and TSPP [2] are employed for ground-motion processing, aiding in structural design and seismic evaluation [20].

C. Seismic and Volcanic Observatories

Software plays a crucial role in supporting natural-threat observatories by processing strong-motion records, like the ones from El Salvador used in our experiments. A parallel approach, similar to ours, using Python and MPI, enables efficient processing of strong-motion files, albeit from volcanoes, demonstrating scalability with the number of events and processor cores [9]. In contrast, sequential software like Earthworm (developed by the USGS) and packages from the Alaska Volcano Observatory provide robust tools for seismic data analysis and real-time monitoring [28, 6]. Additionally, the waveform suite from the University of Alaska Geophysical Institute offers MATLAB code for waveform data manipulation, ensuring data integrity and program stability [32]. These software tools are vital for monitoring seismic and volcanic activity and conducting research in these fields.

D. Early Warning Systems

Software supporting early warning systems for earthquakes and tsunamis plays a pivotal role in mitigating the impact of these natural disasters. The German Indonesian Tsunami Early Warning System (GITEWS) Project has developed the SeisComP3 software package, which offers reliable and fast earthquake location and magnitude estimation capabilities [13]. SeisComP3 is also used in El Salvador as a real-time support tool that allows seismic technicians to visualize and handle customized parameters for magnitude calculations. Furthermore, machine learning approaches, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, are utilized in Early Earthquake Warning (EEW) systems to issue alerts promptly [4]. Early warning systems require rapid data processing and transmission to provide timely alerts, with mere seconds dedicated to sensor data acquisition, processing, magnitude estimation, and decision transmission to the warning server [19, 8, 31]. Overall, developing and deploying advanced software solutions are essential for enhancing the effectiveness of early warning systems, thereby reducing the impact of earthquakes and tsunamis on vulnerable communities.

E. General purpose seismology research

Parallel programming plays a crucial role in advancing seismology research by enabling the efficient processing of strong-motion datasets. Computing Grids provide a platform for sharing seismic data from diverse sources, facilitating seismic waveform analysis, and uncovering observed regions' geological features. The authors of [15] propose a framework for massively parallel wavelet data processing of seismic waveforms. Software tools like *Seismic Analysis Code* offer capabilities for processing multiple signals concurrently, enhancing research productivity and enabling detailed seismic event analysis [12, 33]. A server-side tool [29] allows parallel processing of accelerometric waveforms through a combination of Python for signal processing, Fortran for parallel acceleration and displacement response spectrum calculation, and PHP for dynamic plot generation. Moreover, similar to our approach's architecture, efforts to parallelize source code using High-Performance Fortran and OpenMP demonstrate a faster and more accessible approach to simulating seismic radiation interactions with near-surface geological structures [5].

Nowadays, a considerable portion of scientific software operates sequentially. For instance, the SEISAN seismic analysis system provides a comprehensive suite of programs written primarily in Fortran, supplemented by some C code, facilitating earthquake analysis from both analog and digital data sources [14]. Another example is the open-source software PASCAL Quick Look eXtended (PQLX), which allows for detailed analysis of seismographic data records to study ambient noise and detect earthquake events reliably [30]. Additionally, MATLAB-based software like EQK_SRC_PARA enables estimation of earthquake source spectrum spectral parameters, which are essential for analyzing seismic events and developing scaling laws for specific study regions [17].

These programming tools offer valuable support for seismologists in analyzing seismic data, understanding earthquake characteristics, and improving earthquake detection systems.

X. Conclusion

Our paper introduces a fully-parallelized approach for strong-motion record processing, marking a significant advancement over the original sequential version. Leveraging parallel loops and task parallelization, our method effectively addresses the challenge of efficiently processing accelerographic data, providing scalability and speedup roughly proportional to problem size. This demonstrates the potential for parallel programming techniques to enhance scientific software, playing a pivotal role in advancing seismology research. By enabling efficient data processing, analysis, and simulation of seismic events, such software contributes to a deeper understanding of seismic phenomena and aids in mitigating their impact on society.

Acknowledgments

We thank Sanjay Rajopadhye for helpful feedback on an earlier version of this work. We are grateful to the anonymous reviewers for their detailed feedback on this paper. The Colorado State University co-authors were partially supported by the US National Science Foundation under grant No. 2318970.

References

[1] Rasoul Afsari et al. "Using Artificial Neural Networks to Assess Earthquake Vulnerability in Urban Blocks of Tehran". In *Remote. Sens.* 15.5 (2023), p. 1248.

[2] Jan Baczek et al. "TSPP: A Unified Benchmarking Tool for Time-series Forecasting". In *CoRR* abs/2312.17100 (2023). URL: https://doi.org/10.48550/arXiv.2312.17100.

[3] David Boore and Julian Bommer. "Processing of strong-motion accelerograms: Needs, options and consequences". In *Soil Dynamics and Earthquake Engineering* 25 (Feb. 2005), pp. 93–115.

[4] Marco Carratù et al. "A deep learning approach for the development of an Early Earthquake Warning system". In *IEEE International Instrumentation and Measurement Technology Conference, I2MTC 2022, Ottawa, ON, Canada, May 16-19, 2022*. IEEE, 2022, pp. 1–6.

[5] A. Caserta, V. Ruggiero, and P. Lanucara. "Numerical modelling of dynamical interaction between seismic radiation and near-surface geological structures: a parallel approach". In *Computers & Geosciences* 28.9 (2002), pp. 1069–1077.

[6] Denise Cervelli, P. Cervelli, and T. Murray. "New Software for Long-Term Storage and Analysis of Seismic Wave Data". In *AGU Fall Meeting Abstracts* -1 (Nov. 2004), p. 0705.

[7] Derrick J. A. Chambers, M. Shawn Boltz, and Calum J. Chamberlain. "ObsPlus: A Pandas-centric ObsPy expansion pack". In *J. Open Source Softw.* 6.60 (2021), p. 2696. URL: https://doi.org/10.21105/joss.02696.

[8] Bhanu Chamoli et al. "Development of earthquake early warning system". In *6th Annual Conference of the International Society for Integrated Disaster Risk Management*. Oct. 2015.

[9] Guillermo Corneio-Surez et al. "Using Parallel Computing for Seismo-Volcanic Event Location based on Seismic Amplitudes". In *2018 IEEE 38th Central America and Panama Convention (CONCAPAN XXXVIII)*. 2018.

[10] Ismet Dagli et al. "AxoNN: energy-aware execution of neural network inference on multi-accelerator heterogeneous SoCs". In *DAC*. ACM, 2022, pp. 1069–1074.

[11] Ocione D. Filho et al. "Assessment Of The Impacts Of The 2023 Earthquake In Diyarbakir, Turkey With CBERS-4A Satellite Images". In *XXIV Brazilian Symposium on Geoinformatics - GEOINFO 2023, São José dos Campos, SP, Brazil, December 4-6, 2023*. Ed. by Flávia F. Feitosa and Lúbia Vinhas. MCTI/INPE, 2023, pp. 167–174.

[12] Peter Goldstein and A Snoke. "SAC availability for the IRIS community". In *Incorporated Institutions for Seismology Data Management Center Electronic Newsletter* 7 (Jan. 2005).

[13] W. Hanka et al. "Real-time earthquake monitoring for tsunami warning in the Indian Ocean and beyond". In *Natural Hazards and Earth System Sciences* 10.12 (2010), pp. 2611–2622.

[14] Jens Havskov, Peter H. Voss, and Lars Ottemöller. "Seismological Observatory Software: 30 Yr of SEISAN". In *Seismological Research Letters* 91.3 (Mar. 2020), pp. 1846–1852.

[15] Ljupco Jordanovski, Boro Jakimovski, and Anastas Misev. "Massively Parallel Seismic Data Wavelet Processing Using Advanced Grid Workflows". In *ICT Innovations 2009, Ohrid, Macedonia, 28-30 September, 2009*. Ed. by Danco Davcev and Jorge Marx Gómez. Springer, 2009, pp. 411–419.

[16] Lion Krischer et al. "ObsPy: A bridge for seismology into the scientific Python ecosystem". In *Computational Science & Discovery* 8 (May 2015), p. 014003.

[17] Arjun Kumar et al. "Software to Estimate Spectral and Source Parameters." In *International Journal of Geosciences* 3 (Nov. 2012), pp. 1142–1149.

[18] Ashok Kumar et al. "Indian Strong Motion Instrumentation Network". In *Seismological Research Letters* 83 (Jan. 2012), pp. 59–66.

[19] Pankaj Kumar et al. "Successful Alert Issuance with Sufficient Lead Time by Uttarakhand State Earthquake Early Warning System: Case Study of Nepal Earthquakes". In *Journal of the Geological Society of India* 99 (Mar. 2023), pp. 303–310.

[20] Jin Koo Lee, Jeongbeom Seo, and Sung Whang. "A study on the design of ground motion database and processing for input seismic evaluation and nuclear power plant safety". In *Transactions of the Korean Nuclear Society Spring Meeting* (May 2023).

[21] Dennis Lemus. *Monthly Seismic Activity Bulletin*. Ed. by Ministerio de Medio Ambiente y Recursos Naturales Observatory of Natural Threats. Dec. 2023. URL: https://www.snet.gob.sv/informacion/?area=sismologia (visited on 03/07/2024).

[22] Axel Lloret. *Basalt Accelerograph Photo*. Ed. by National University of Cuyo Digital Photo Repository Argentina. Sept. 2020. URL: https://fotos.uncuyo.edu.ar/piwigo/picture.php?/3266/category/161 (visited on 02/23/2024).

[23] Marco Massa et al. "The ITalian ACcelerometric Archive (ITACA): processing of strong-motion data". In *Bulletin of Earthquake Engineering* 8.5 (Oct. 2010), pp. 1175–1187.

[24] Daniel Mawhirter et al. "Dryadic: Flexible and Fast Graph Pattern Matching at Scale". In *PACT*. IEEE, 2021, pp. 289–303.

[25] Jedidiah McClurg et al. "Optimizing Regular Expressions via Rewrite-Guided Synthesis". In *PACT*. ACM, 2022, pp. 426–438.

[26] Kinemetrics Newsletter. *Kinemetrics executes 3rd contract with Caltech for California's earthquake early warning system*. Mar. 2017. URL: https://kinemetrics.com/news/kinemetrics-executes-third-contract-with-caltech-for-californias-earthquake-early-warning-system/ (visited on 02/23/2024).

[27] Michail Ntinalexis et al. "A database of ground motion recordings, site profiles, and amplification factors from the Groningen gas field in the Netherlands". In *Earthquake Spectra* 39.1 (2023), pp. 687–701.

[28] Marco Olivieri and John Clinton. "An almost fair comparison between Earthworm and SeisComp3". In *Seismological Research Letters* 83.4 (2012), pp. 720–727.

[29] R. Puglia et al. "Strong-motion processing service: a tool to access and analyse earthquakes strong-motion waveforms". In *Bulletin of Earthquake Engineering* 16.7 (July 2018), pp. 2641–2651.

[30] Sepideh J. Rastin et al. "PQLX noise model for the URZ station in the Matata region of New Zealand". In *10th International Conference on Information Sciences, Signal Processing and their Applications, ISSPA 2010, Kuala Lumpur, Malaysia, 10-13 May, 2010*. IEEE, 2010, pp. 17–20.

[31] Govind Rathore et al. "Development and Implementation of a Regional Earthquake Early Warning System in Northern India". In. *Proceedings of 17th Symposium on Earthquake Engineering (Vol. 4)*. July 2023, pp. 537–544.

[32] Celso Reyes and Michael West. "The Waveform Suite: A robust platform for manipulating waveforms in MATLAB". In *Seismological Research Letters* 82 (Jan. 2011), pp. 104–110. DOI: 10.1785/gssrl.82.1.104.

[33] Brian Savage. "sacio: A library for Seismic Analysis Code data files". In *J. Open Source Softw.* 6.67 (2021), p. 3619. URL: https://doi.org/10.21105/joss.03619.

[34] A Zsarnóczay and GG Deierlein. "PELICUN: A Computational Framework for Estimating Damage, Loss and Community Resilience". In *Proceedings, 17th World Conference on Earthquake Engineering, (Sendai: WCEE)*. 2020.