Performance enhancements and visualization for RSQSim earthquake simulator

Dmitry Pekurovsky[1], Amit Chourasia[1], Keith B. Richards-Dinger[2], Bruce E. Shaw[3], James H. Dieterich[2], Yifeng Cui[1]

[1]San Diego Supercomputer Center, University of California, San Diego [2]University of California, Riverside [3]Columbia University

Abstract

We report on progress of performance tuning and visualization for earthquake simulator code RSQSim. We have studied performance of the code in detail on supercomputers such as Blue Waters (NCSA/UIUC), Stampede (TACC) and Mira (Argonne). We have continued to improve scalability efficiency of the code, mainly by reducing load imbalance. This is achieved by introducing a new feature of the code which uses an added physical constraint that allows a large reduction in the number of times when an iterative solve is called. This saves on calculation time, as well as makes the code more load-balanced due to less variable numbers of calculation steps. This feature alone leads to up to 33% better performance and substantially improved scalability. Furthermore, we have improved the scaling of the initialization portion of the code and created a Github repository, for the first time, for the SCEC CISM project.

We also report final results from visualization work. RSQSim generates data catalog of millions of seismic events for thousands of years and their associated properties such as changes in slip, stress, etc. Exploring this data is of vital importance to validate the simulator as well as to identify features of interest such as quake time histories, conduct analyses such as calculating mean recurrence interval of events on each fault section. We have created a prototype web based tool for exploring this data interactively. In course of visualization work we recognized the need to transform the data from current form into a more scalable and manageable format. We also summarize data management findings and discuss how the changes impact analysis and its future potential. Interactive visualization is accessible at http://vis.sdsc.edu:5555

Computation

RSQSim Earthquake Simulator

A time-series algorithm

- Each patch/element makes transitions between 3 basic states according to equations of fault friction and based on the state of other elements
- At each time step, go through all elements and calculate the time of next transition based on current configuration.
- Find the minimum transition time and advance all elements. Repeat as desired.

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$$

Computational Needs

We need to scale to O(10K) cores and more, to accommodate memory requirements and achieve faster throughput as

- The code can currently run on O(1K) cores with O(100K) elements.
- Future needs: O(1M) elements
- Memory usage scales as N_{elem}^2

Parallel Performance

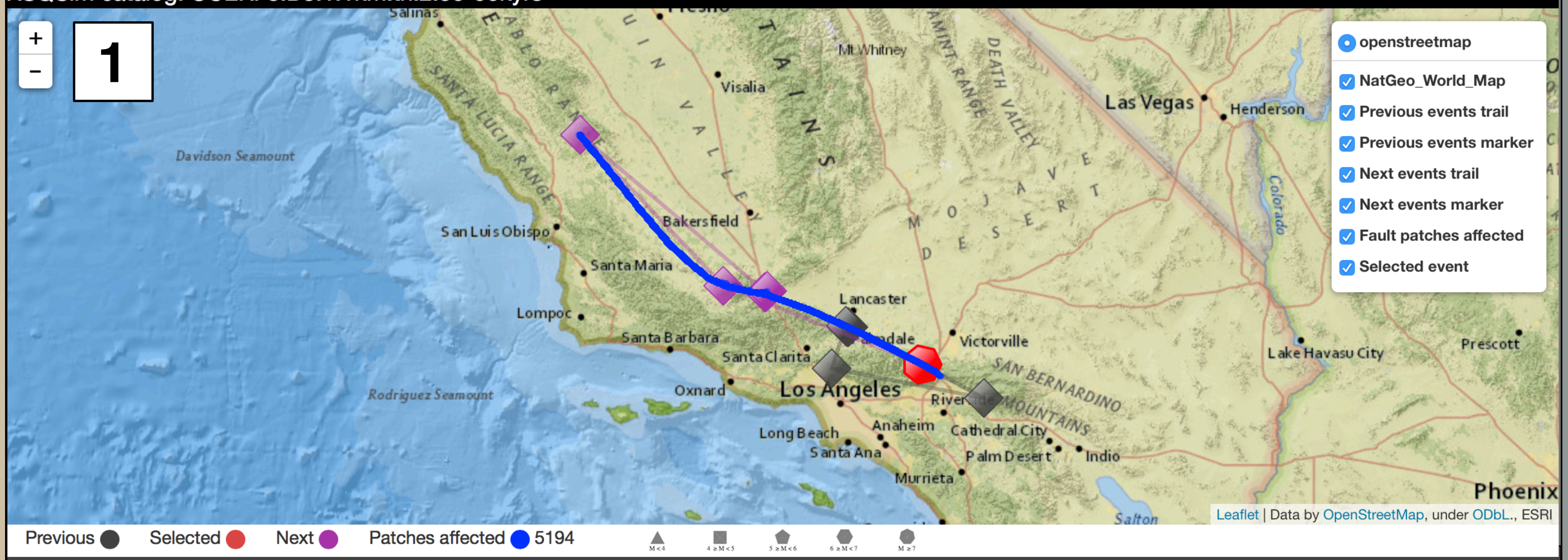
- Patches are distributed among compute cores
- Wall clock time to compute transition time is highly uneven among the patches.
- Some cores take much longer to compute this step than others, resulting in other cores idly waiting for them.
- Therefore finding global minimum transition time is a time-consuming part of the algorithm and does not scale well with core count.

Algorithm Optimization

- Load imbalance arises from finding the global minimum time of transition
- Mainly expensive solves of a root finding (zbrent) and an allreduce operation in MPI.
- Address load imbalance by limiting the number of times the zbrent() function is called. This is controlled by a parameter ZBrentUpperBracket, set to a default value that precludes solving for the minimum time in cases during dynamic rupture.
- This saves computation time (up to 33%) as well as improves scalability by substantially reducing load imbalance (since very few tasks will have significant numerical solve times). See figure 4 on the right.

RSQSim catalog: UCERF3.D3.1.1km.tri.2.50-90kyrs

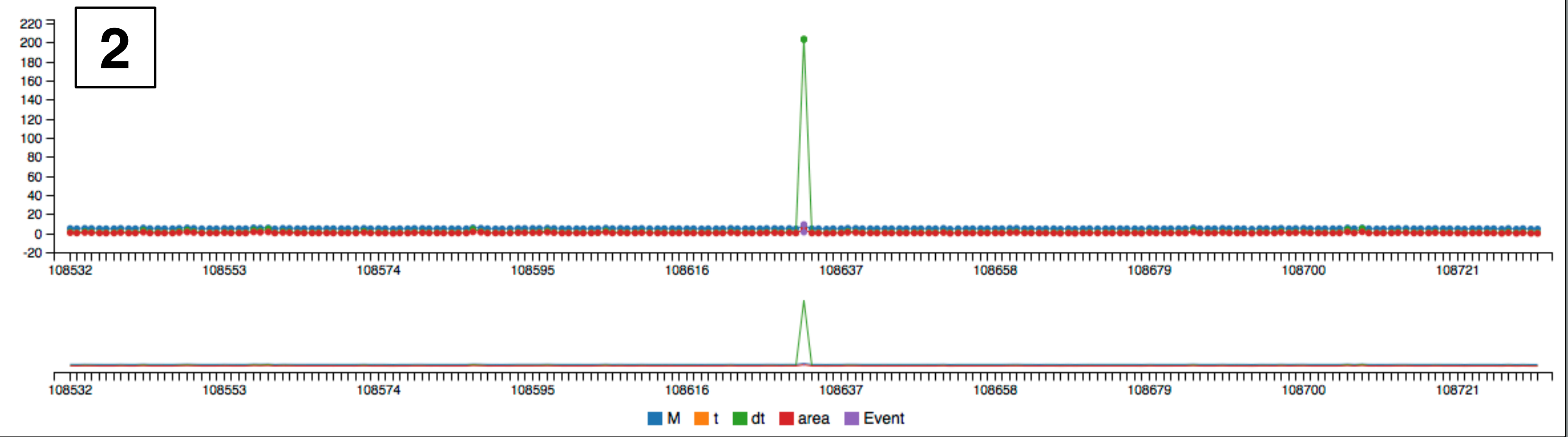
1



Map showing selected event with a red circle and corresponding affected fault patches in blue. Previous events are displayed with grey circles connected by a grey line that indicates their chronology with respect to the selected event. Similarly next events are displayed with purple circles connected by a purple line. Visibility control toggles for various layers are on top right.

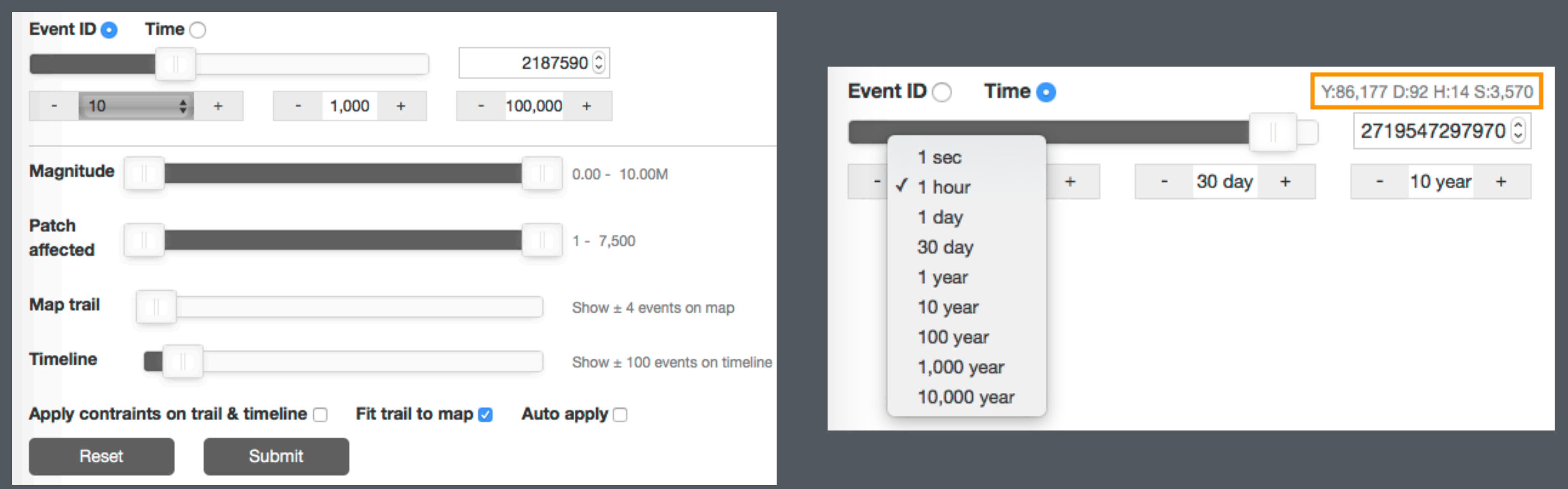
Timeline : ± 100 adjacent events

2



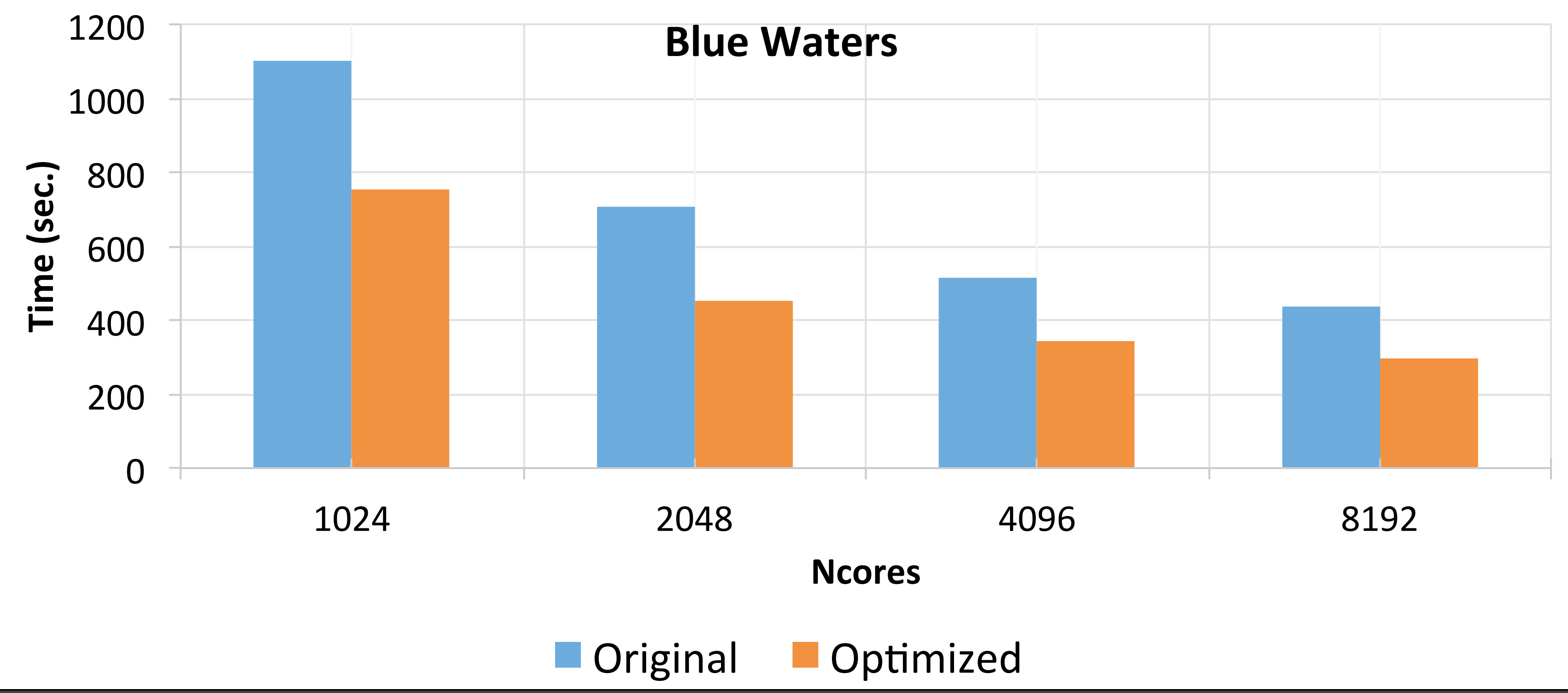
Line plot shows magnitude, duration and area of 100 previous and 100 next events that are adjacent the selected one shown in the center. Clicking on any event on the time series loads the chosen event in the map above, enabling swift exploration.

3



Snapshot of full graphical user interface. Left image showing selection by event ID. The event can be further filtered via magnitude and patch sliders. Trail and time series sliders allow customization of ancillary items for display. Right image shows selection by event time. The interface allows users to provide input via a slider (for easy scrubbing), text input box (for precise numeric input) as well as increment and decrement buttons at preselected scales for swift exploration. A drop down selection list enables choice for additional scales.

4



This figure shows strong scaling of RSQSim performance of original and optimized code. This is defined as wall clock time change as we increase the number of cores while keeping the problem size constant. Both absolute performance and scaling improves with the optimization reducing the number of root solves. The optimized code is scalable up to 8k cores.

Visualization & analysis

Visualization of RSQSim data

Developed an interactive web based application, that allows event selection and filtering of large earthquake catalog data.

Challenges

- What and how to display?
- Combining different plots and linking them
- Capable user interface to select events from 6 million records both via id and time as well filtering them on magnitude and affected patches
- Fast querying and quick round trip communication for generating and presenting visualization

Data wrangling

- Normalize index to 1 based
- Normalize variable names
- Translate 3D data to 2D
- (triangle patches stored with vertex + 3D rotations)
- (rectangle patches stored with center + length, width + 3D rotations)
- Add geographic projections from UTM to EPSG 4236
- Compare using raw data vs database

Application design

Developed a web app using the following

- Python
  Flask framework : server/client brokering
  Folium : map plotting
- JavaScript
  Leaflet : mapping
  C3JS : time series plots
  noUISliderJS : mobile friendly sliders
- HTML

User interface design

- Map : Layers, Markers pop-ups, Link with time series plot. See figure 1
- Time series: Linked with map plot, Enable filter constraints. See figure 2
- Selection : Event or Time. See figure 3
- Filters : Magnitude, Number of patches, Trail events. See figure 3

Database findings

- Fast and easy querying
- Single file, requires less management
- Does not require loading all data in memory

Conclusions

In summary we streamlined data management, analysis and visualization process

Transformed data to a database

- Normalized data from mixed index in source data
- Validated data transformations
- Easier data management/sharing, only one file
- Scalable, as data does not need to be loaded into memory
- Fast and easy querying

Developed interactive visualization web application

- Enables quake exploration in rich geographic context for a large catalog
- Can support multiple concurrent users

Analysis

Calculated mean recurrence interval with nucleation and with nucleation + participation

Acknowledgments

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575. This work also utilized expertise on code optimization, workflows and visualization provided via the XSEDE Extended Collaborative Support Service (ECSS) program. This work is also supported by National Science Foundation grant number EAR-1135455 and Keck Foundation grant number 005590-00001.