LIGO

Meeting the Computational Challenges of Third-Generation Gravitational-Wave Detectors

Josh Willis

LIGO Laboratory/Caltech

30 November 2021

LIGO

Outline

- Overview of gravitational-wave data analysis computing
- Challenges posed by current and third-generation detectors
- Describe some of the software engineering challenges that flow from this
- What are some ways to address this?

See https://gwic.ligo.org/3Gsubcomm/, particularly Computing report

LIGO

Background

- Staff computational scientist with LIGO project (Caltech) since 2017
  - Support both laboratory and LSC scientific computing, both grid computing and optimization
- Prior to that faculty member at teaching university; joined LIGO Scientific Collaboration in 2011
- Background in classical and quantum general relativity

LIGO

Gravitational-Wave Data Analysis

- Big picture: can break down by both source type or analysis technique
- Searches:
  - Continuous Wave (long signal, matched filtering)
  - Compact binaries (transient signal, matched filtering or unmodeled)
  - Stochastic (background of signals)
  - Burst (transients, generally unmodeled)

LIGO

Gravitational-Wave Data Analysis

- Searches look at all the data, finding and estimating the significance of source candidates
- Significant candidates must then be followed up, and their properties measured (parameter estimation)
- When there are many candidates, we also use their measured properties to make inferences about populations of sources
- All of this analysis requires significant computing!

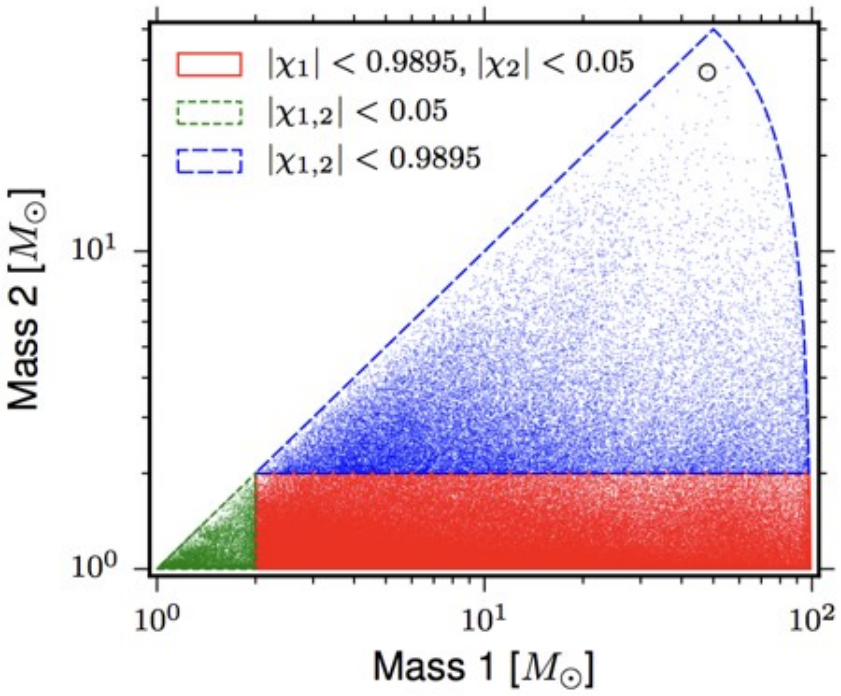
LIGO

Matched Filtering

- When we have good models of a source, the statistically (but not necessarily computationally) optimal is some variation of matched filtering
- Compare each stretch of data (in each detector) to your candidate signal, and search over the parameter space of candidate signals
- For continuous signals, the “stretch” of data would in principle be all of your data, and an exhaustive search is intractable
- For transient modeled sources (CBC) this is the current method used in all production searches by LIGO & Virgo

Matched Filtering

- CBC matched filtering: search against templates in a *bank*
- In 2G (current) detectors, these are up to O(100) seconds long
- For searches, the full parameter space is not necessary, focus on mass and (aligned) spin
- Shift templates in time, maximize over other parameters
- Coincident search (search each detector separately)



LIGO

Parameter Estimation

- See talk by John Veitch later this week
- In brief, a variety of sampling algorithms may be used to explore parameter space. Increasingly, for CBC PE, this must cover the full 15-dimensional space (and possibly more)
- This will certainly be the case for 3G PE
- Thus far, roughly speaking, each event has required ~1 million CPU core hours (see talk earlier during this long program by Peter Couvares)

LIGO

Computational Challenges

- Matched filtering is *embarrassingly parallel*: different parts of the template bank, and different stretches of data, can be analyzed independently of each other. Complex to manage this, but *high throughput computing.*
- Sampling methods for parameter estimation fit this model less well; the primary parallelization is across different realizations of a sampler, and the many different events to be analyzed.
- Some sampling methods can be parallelized with complex communication methods, and are examples of *high performance computing.*

LIGO

Computational Challenges

- For high throughput computing, there is a sophisticated infrastructure to manage the individual jobs in a workflow, and allow those jobs to match to available computing resources throughout the world.
- For LIGO/Virgo/KAGRA, this computing infrastructure is reliant on the Open Science Grid, HTCondor software, and other critical pieces of infrastructure to distribute the data itself as well as other files needed by the analyses, and to bring the results back
- See talk on Friday by Frank Wuerthwein

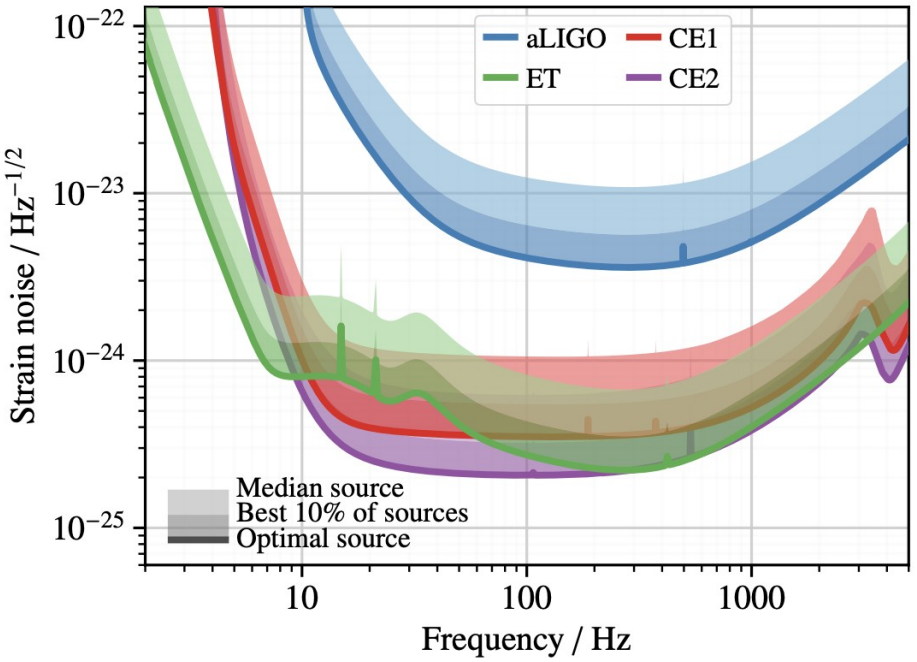
LIGO

3G Challenges

- To date, LIGO & Virgo have detected 90 binary mergers
- Planned 3G detectors could observe hundreds of mergers per day, totaling O(10^4–10^5) per year
- Naive extrapolation would predict ~10^{10} core hours
- Moreover, latency for MMA will be critical for many of the science goals of 3G, as will precision of parameter estimation, tests of general relativity
- Experience in 2G has shown that unexpected results lead inevitably to increased computing: more detailed analyses and comparisons are needed

3G Challenges

- Matched filtering is most sensitive to the *shape* of the noise PSD
- Though matched filtering will undoubtedly require more computing for 3G, its costs scale modestly (Phys. Rev. D 104, 063011 (2021))
- This does assume no increase in dimension of parameter space
- CW searches remain intractable to exhaustive matched filtering search



3G Parameter Estimation

- The naive extrapolation mentioned earlier did not account for the significantly longer (minutes to hours instead of seconds to minutes) waveforms needed for 3G detectors. Yet it was still enormous.
- Many have been working to address this. Some ways to make these calculations more tractable:
  - Relative binning (Zackay+, arxiv:1806.08792; Finstad& Brown, ApJ 905, L9 (2020)
  - Reduced order quadrature (e.g. PRL 127, 081102 (2021); PRL 114, 071104 (2015); PRD 94, 44031 (2016))
- Machine learning: talks this week by Deep Chatterjee, Michael Coughlin, Jonathan Gair

LIGO

3G Parameter Estimation

- Most methods are (at some point) still dominated by the cost of waveform generation. In ML that can be in the training stage.
- There are also applications of ML to waveform generation itself (see talk on Wednesday by Leila Haegel)
- Yet for 3G such waveform models need to be much more accurate---one order of magnitude for numerical relativity, and up to three orders of magnitude for semi-analytical models (Pürrer & Haster, arxiv:1912.10055)
- For the rest of the talk, I'll focus on the *software engineering* challenge this poses

LIGO

Computing Performance Landscape

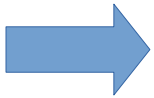
- For decades, *Moore's Law* has predicted that the number of transistors on a CPU doubles roughly every 18 months
- It's end has been predicted for a long time, but not yet realized
- At one point, it was possible to try to write reasonable code in a high-level language, and trust that it would run faster on a faster machine (clock speeds increased regularly)
- But it has not been that simple for a while

LIGO

Computing Performance Landscape

- Modern processors achieve their performance through *vectorization*: when the CPU executes one operation, it is possible for that operation to in fact compute many things at once

- `for (i=0; i<N; i++)`
  - `a[i] = b[i] + c[i]`



- `for (i=0; i<N; i+=2)`
  - `a[i] = b[i] + c[i]`
  - `a[i+1] = b[i+1] + c[i+1]`

- Not every code can take advantage of this

LIGO

Computing Performance Landscape

- In the simplest case, a software developer doesn't really worry about this. Instead, the compiler should do this for them.
- But for complex software this is rarely sufficient (many reasons).
- Also, even when this does work, it requires knowing which kind of hardware the software will execute on---there are many different versions of vectorization (SSE, AVX, AVX2, AVX512, AMD)
- In a distributed computing environment, this is close to impossible

LIGO

Further Complications

- In addition to vectorization, performant code usually has to worry about how it accesses the CPU's memory. Even on machines with the same instruction set, these differences can matter
- Out-of-order execution, memory dereferencing, can all also degrade performance of software that was not prepared for them
- Without accounting for this, there will be a large gap between the performance a machine is capable, and what a particular scientific code achieves

LIGO

Modern Software Engineering

- In practice, the correct way to handle this is for the scientific user to *not* write the performance-critical parts of their code themselves.
- Instead, they should consciously write their software to leverage *libraries*: defined external pieces of software that their code will call. The library will adhere to a specified interface (API) and that is all the scientific user needs to know.
- There are libraries that cover many common scientific computations; generally they build on each other.

LIGO

Example

- If I need to solve a system of linear equations, I should select from among the routines available in the LAPACK interface
- But that library itself will call another library, the Basic Linear Algebra Subroutines (BLAS) library
- My code will set up the linear system, and if I choose an implementation of BLAS that is efficient, it will take care of vectorization, memory access (cache performance), and other similar subtleties.

LIGO

Heterogenous Computing

- Libraries such as BLAS accomplish these optimizations through *runtime dispatching*: they will detect what CPU they are running on and select the best available implementation for that hardware.
- There are often implementations maintained by hardware vendors (Intel, AMD, NVIDIA) or dedicated scientific computing research groups.
- Popular machine learning frameworks (tensorflow, PyTorch) are also doing this internally, supported by their own engineering staff.

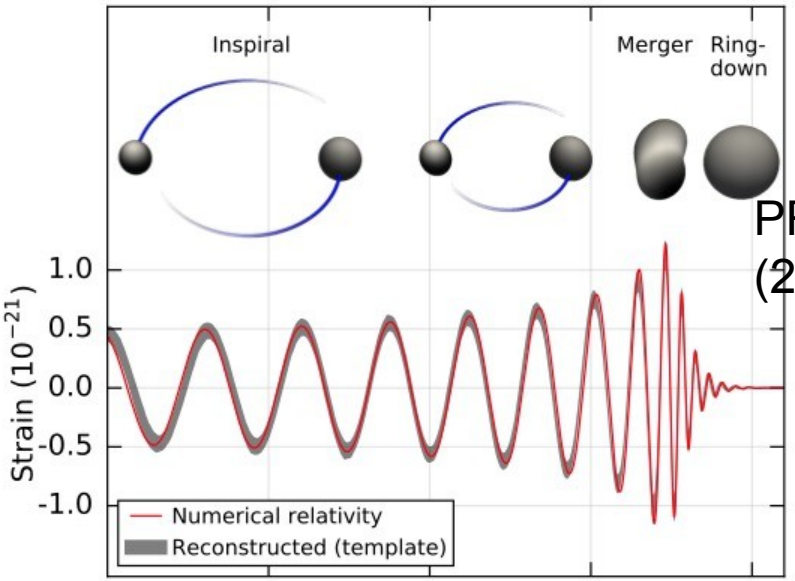
LIGO

When this works and doesn't

- A great many of the gravitational-wave analyses outlined earlier follow this approach, relying on library support for common operations (linear algebra, FFT, etc)
- The primary place this model breaks down is when the core computation (whatever limits the performance) is *domain-specific*: unique to an area of science so that there is no existing library that has an engineering staff behind it, and no external incentive for this

Domain-specific GW analysis

- In both matched-filter CBC searches, and parameter estimation via sampling, the most domain-specific computation is the generation of gravitational wave-forms.



PRL 116, 061102 (2016)

LIGO

Software Engineering Waveform Modeling

- The expertise required to create accurate waveform modeling (which combines both analytical relativity, data analysis techniques, and numerical relativity) can be fairly disjoint from the somewhat esoteric computing issues just discussed
- Yet most of the existing models are not readily defined in such a way that their most computationally intensive functions correspond to standard library functions
- At the same time, already these models tend to evolve quickly to incorporate improved analytical models and updated NR simulations. Makes “write once then optimize” impossible

LIGO

What's Needed

- The models themselves can change often; this is to be expected and planned for
- Hardware capabilities also evolve, though on a slower scale. But they diversify: the number of hardware platforms we may want to support increases over time
- At the same time, the implementations need to be *maintainable*
- There exist some wrappers that can help, depending on the model:
  - https://sleef.org/
  - https://github.com/vectorclass

LIGO

What's Needed

- There are existing frameworks for waveform models (e.g. lalsuite)
- But it has proven challenging to directly integrate SIMD and GPU into some of these general frameworks
- Focused effort required to build the pieces, and to write them in a templated manner that does not require maintenance of hand-crafted vectorized code
- Even for simple operations (`cexp(i*phase)`) can achieve an order of magnitude gain, or more
- Will also require integration with inference code to see this benefits ("loop fusion")

LIGO

Conclusions

- Need software engineering effort focused on domain-specific computing for 3G; maintainability and reliability will be key requirements
- Some planning needs to go to how this effort is organized and funded (compare to “The Astropy Problem” arxiv:1610.03159
- Requires cooperation between domain experts and software engineers