Parallelized inference for gravitational-wave astronomy

Colm Talbot,[*] Rory Smith, and Eric Thrane
School of Physics and Astronomy, Monash University, Vic 3800, Australia
and OzGrav: The ARC Centre of Excellence for Gravitational Wave Discovery,
Clayton VIC 3800, Australia

Gregory B. Poole
Astronomy Data and Computing Services (ADACS); the Centre for Astrophysics & Supercomputing,
Swinburne University of Technology, P.O. Box 218, Hawthorn, VIC 3122, Australia

(Received 5 April 2019; published 28 August 2019)

Bayesian inference is the workhorse of gravitational-wave astronomy, for example, determining the mass and spins of merging black holes, revealing the neutron star equation of state, and unveiling the population properties of compact binaries. The science enabled by these inferences comes with a computational cost that can limit the questions we are able to answer. This cost is expected to grow. As detectors improve, the detection rate will go up, allowing less time to analyze each event. Improvement in low-frequency sensitivity will yield longer signals, increasing the number of computations per event. The growing number of entries in the transient catalog will drive up the cost of population studies. While Bayesian inference calculations are not entirely parallelizable, key components are embarrassingly parallel: calculating the gravitational waveform and evaluating the likelihood function. Graphical processor units (GPUs) are adept at such parallel calculations. We report on progress porting gravitational-wave inference calculations to GPUs. Using a single code—which takes advantage of GPU architecture if it is available—we compare computation times using modern GPUs (NVIDIA P100) and CPUs (Intel Gold 6140). We demonstrate speed-ups of $\sim 50\times$ for compact binary coalescence gravitational waveform generation and likelihood evaluation, and more than $100\times$ for population inference within the lifetime of current detectors. Further improvement is likely with continued development. Our python-based code is publicly available and can be used without familiarity with the parallel computing platform, CUDA.

DOI: 10.1103/PhysRevD.100.043030

I. INTRODUCTION

In the first two observing runs of Advanced LIGO/Virgo, ten binary black hole mergers were detected along with one binary neutron star inspiral [1]. These observations allowed us to measure the Hubble parameter [2], to study matter at extreme densities [3], and to probe the underlying distribution of black holes in merging binaries [4]. Within the lifetime of advanced detectors, we conservatively estimate that hundreds of such observations will be made given inferred merger rates and projected sensitivity [5].

Compact binary coalescences are analyzed with Bayesian inference (see e.g., [6] for a general introduction or [7] for applications to gravitational waves.). We distinguish between two kinds. We refer to inferring the properties (e.g., the masses, spins, and location) of individual binaries as single-event inference. Hierarchical Bayesian inference is then used to infer the ensemble properties (e.g., the shape of the binary black hole mass distribution) of the observed binaries in population inference. These are typically performed with stochastic sampling algorithms such as Markov chain Monte Carlo [8,9] or nested sampling [10]. These algorithms generate samples from the posterior distribution and possibly a Bayesian evidence which can be used for model selection.

Both single-event inference and population inference require the computation of likelihood functions consisting of many independent operations. For single-event inference, the number of operations per likelihood evaluation is determined by the length of the signals being analyzed; see Eq. (1). As the low-frequency sensitivity of detectors increases, binaries will spend longer in our sensitive frequency range, leading to fast-increasing computational demands. For population inference, the number of operations required per likelihood calculation is proportional to the number of binaries in the population; see Eq. (3). The growing number of observations and the growing duration of the longest signals in the catalog require improved speed for inference algorithms.

Most previous methods for accelerating inference for compact binary coalescences have sped up calculations by reducing the amount of data required to represent the

[*]colm.talbot@monash.edu

gravitational-wave signal, thereby reducing the number of operations required to evaluate the likelihood, e.g., reduced order methods [11–13], multibanding [14], and relative binning [15]. Another approach, which we investigate here, is to parallelize the most time-consuming calculations in the likelihood evaluation. While it is difficult to parallelize the actual sampling algorithm, there are embarrassingly parallel calculations within the likelihoods. In this paper, we explore how astrophysical inference can be accelerated by executing parallelizable calculations on graphical processor units.

While we focus here on inference using stochastic samplers, e.g., [16–18], it bears mentioning that there are alternative inference schemes, which also face computational challenges; e.g., iterative fitting [19,20] is one such alternative. This method evaluates far fewer gravitational waveforms, which can significantly reduce computation times for inference when the waveform evaluation is very time consuming. Recently, it has been shown that this algorithm can also be significantly accelerated by parallelization [21].

Modern computation is mostly performed on either a central processing unit (CPU) or a graphics processing unit (GPU). CPUs consist of a relatively small number of cores optimized to perform all the tasks necessary for a computer in serial. GPUs, on the other hand, consist of hundreds or thousands of cores, enabling evaluation of many numerical operations simultaneously. This makes them ideal for embarrassingly parallel operations such as manipulating large arrays of numbers. Low-level GPU programming is carried out using the parallel computing platform, CUDA. In this work, we take advantage of the library, CUPY [22].

We present python packages with parallelized versions of the likelihoods necessary for performing both single-event and population inference. The code is designed to run on either a CPU or a GPU, depending on the available hardware. Our GPU-compatible code for single-event inference is available at [23], and our CUDA compatible version of IMRPHENOMPV2 at [24]. Our GPU population inference code GWPOPULATION is available from the python package manager PYPI and from git at [25].

Using our GPU-accelerated code, we investigate the speed-up achieved carrying out gravitational-wave inference calculations using GPUs versus traditional CPUs. We carry out a benchmarking study in which we compare inference code using GPUs to identical code running on CPUs. We compare the run times for various tasks including (i) evaluating a gravitational waveform, (ii) evaluating the single-event likelihood, and (iii) evaluating population likelihood. To carry out this comparison, we use NVIDIA P100 GPUs and Intel Gold 6140 CPUs available on the OzStar supercomputing cluster.

In Sec. II we describe methods for parallelizing the evaluation of gravitational waveforms. In Sec. III we use these parallelized waveforms to consider the possible acceleration for single-event inference. In Sec. IV, we investigate the possible acceleration from parallelizing the population inference likelihood. Finally, we provide a summary of our findings and future work in Sec. V.

II. WAVEFORM ACCELERATION

A key ingredient for single-event inference is a model for the waveform, $\tilde{h}(f,\theta)$. Here, $\tilde{h}$ is the discrete Fourier transform of the gravitational-wave strain time series, f is frequency, and θ is the set of parameters (typically 15–17), which determine the waveform, e.g., the masses, spins, and orientation of the binary. This theoretical waveform is compared with the data every time the likelihood function is evaluated. Many commonly used waveform models can be directly evaluated in the frequency domain and the value at each frequency can be evaluated independently (e.g., [26–29]). This makes the waveform model evaluation embarrassingly parallel.

We design two different codes for performing waveform generation on a GPU. First, we implement a GPU version of a simple, post-Newtonian inspiral-only waveform TAYLORF2 [26] directly in python using CUPY. Secondly, we convert the C code for a phenomenological waveform IMRPHENOMPV2 [27] into CUDA [30]. Both of these methods are then compared to the time required to evaluate the same waveform implemented in C within LALSUITE [31].

Technically, the CUDA version of IMRPHENOMPV2 does not adhere to our philosophy of keeping the GPU programming entirely "under the hood"; however, writing custom CUDA kernels allows increased optimization of the GPU code. We consider the effect of preallocating a reusable memory "buffer" on the GPU. During parameter estimation waveforms of the same length are generated many times as a waveform evaluation is required for every likelihood evaluation. Since the waveform always has the same size, a predefined spot in memory can be reused for every waveform.

In Fig. 1, we plot the speed-up (defined as the CPU computation time divided by the GPU computation time) for both of our waveforms. The dashed line indicates no speed-up. In blue we show the comparison of our TAYLORF2 waveform using CUPY with the C version available in lalsuite [31]. We find that, for signals longer than ~ 10 s, we achieve faster waveform evaluation. The speed-up scales roughly linearly with signal duration so that the waveforms of duration 100 s are sped up by a factor of ≈ 10. For signals longer than ~ 1000 s the GPU queue saturates and the rate of increase of the speed-up decreases.

In orange, we plot the speedup for our CUDA implementation of IMRPHENOMPV2 compared to the C implementation of the same waveform model in lalsuite. The solid curve includes the use of a preallocated memory buffer, while the dashed curve does not. We find that preallocating memory buffer increases the performance when the number of frequencies is $\lesssim 10^5$ and leads to accelerations for all waveforms when the number of frequency bins is larger than 100.

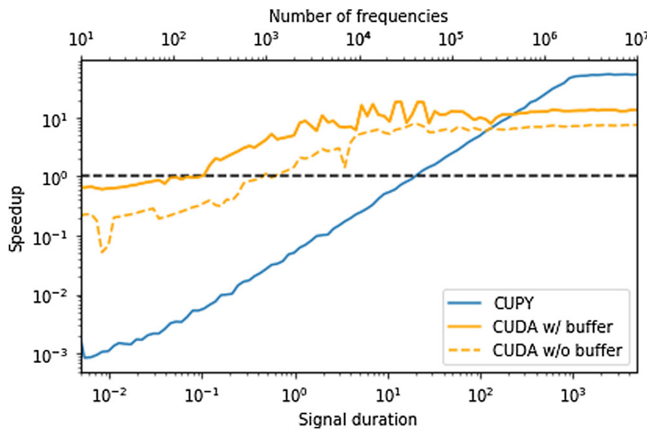

FIG. 1. Speed-up comparing our GPU accelerated waveforms with the equivalent versions in LALSuite as a function of the number of frequencies. The corresponding signal duration is shown for comparison assuming a maximum frequency of 2048 Hz. The blue curve shows the comparison of our CUPY implementation of TAYLORF2 with the version available in lalsuite. The GPU version is slower for shorter waveform durations; this is likely because of overheads in memory allocation in CUPY. For binary neutron star inspirals the signal duration from 20 Hz is ∼100 s, at which signal durations we see a speed-up of ∼10×. For even longer signals we find an acceleration of up to 80×. The orange curves show a comparison of our CUDA implementation of IMRPHENOMPV2; the solid curve reuses a preallocated memory buffer while the dashed line does not. We see that the memory buffer makes the GPU waveform faster for shorter signals.

This suggests that the performance of our CUPY waveform could be similarly improved for short signal durations using more sophisticated waveform allocation.

III. SINGLE-EVENT LIKELIHOOD ACCELERATION

The standard likelihood used in single-event inference is

$$\mathcal{L}(d|\theta) = \prod_{i}^{N_{\rm det}} \prod_{j}^{M} \frac{1}{2\pi S_{ij}} \exp\left(-\frac{2}{T}\frac{|\tilde{d}_{ij} - \tilde{h}_{ij}(\theta)|^2}{S_{ij}}\right), \quad (1)$$

where d is the detector strain data, θ are the binary parameters, $\tilde{h}$ is the template for the response of the detector to the gravitational-wave signal as described in Sec. II, S is the strain power spectral density, the products over i and j are over the M frequency bins and $N_{\rm det}$ detectors in the network, respectively.

Different signals have different durations, and thus, different values for the number of frequency bins M. The more frequency bins, the more embarrassingly parallel calculations to perform, and the more we expect to gain from the use of GPUs. For example, systems with lower masses produce longer duration signals than systems with higher masses, all else equal. In previously published

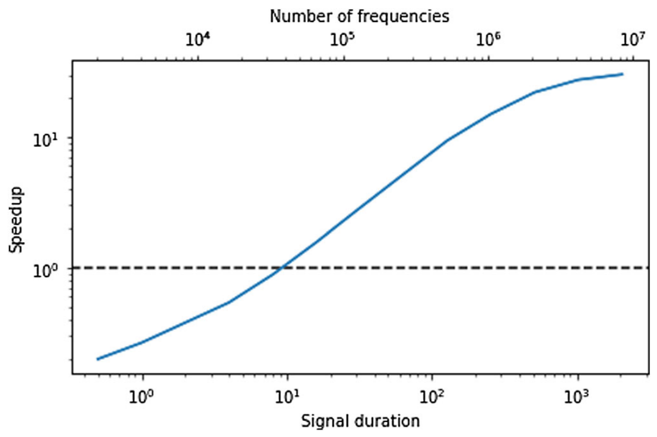

FIG. 2. Speed-up of the compact binary coalescence likelihood as a function of the number of frequencies with and without a GPU using our accelerated TaylorF2. The corresponding signal duration is shown for comparison assuming a maximum frequency of 2048 Hz. We see similar performance as for the waveform generation. We find a speed-up of an order of magnitude for 128 s signals.

observational papers, the minimum frequency is typically set to 20 Hz for binary black holes, although a larger minimum frequency was used in the analysis of GW170817 [32]. For high mass binary black holes, this corresponds to $M = 4,000$–$32,000$ bins. Binary neutron star signals are much longer in duration, requiring $M \approx 10^6$ bins. When current detectors reach design sensitivity, frequencies down to 10 Hz are used, leading to substantially longer signals and therefore many times more bins. Future detectors are expected to be sensitive to frequencies as low as 5 Hz [33].

During each likelihood evaluation, the most expensive step is usually calculating the template. The next most expensive operation is applying a time translation to the frequency-domain template in order to align the template with the merger signal. To do this, the frequency-domain waveform is multiplied by a factor of $\exp(-2i\pi f\delta t_{\rm det})$, where $t_{\rm det}$ is different for each detector in the network. This becomes increasingly expensive (with linear scaling) as more detectors are added to the network.

In Fig. 2 we show the speed-up achieved evaluating the single-event likelihood function using our TAYLORF2 implementation compared to the LALSIMULATION implementation. We find a speed-up of an order of magnitude for a 128 s-long signal, the typical duration for a binary neutron star analysis with a minimum frequency of 20 Hz. This reduces the total calculation time from a few weeks to a few days.

When the number of frequency bins is relatively small, $M < 10^5$, we see a slowdown rather than a speed-up. This is due to the same overheads as seen in Sec. II. Using the CUDA implementation of IMRPHENOMPV2, we would expect to see an acceleration for much shorter signal durations.

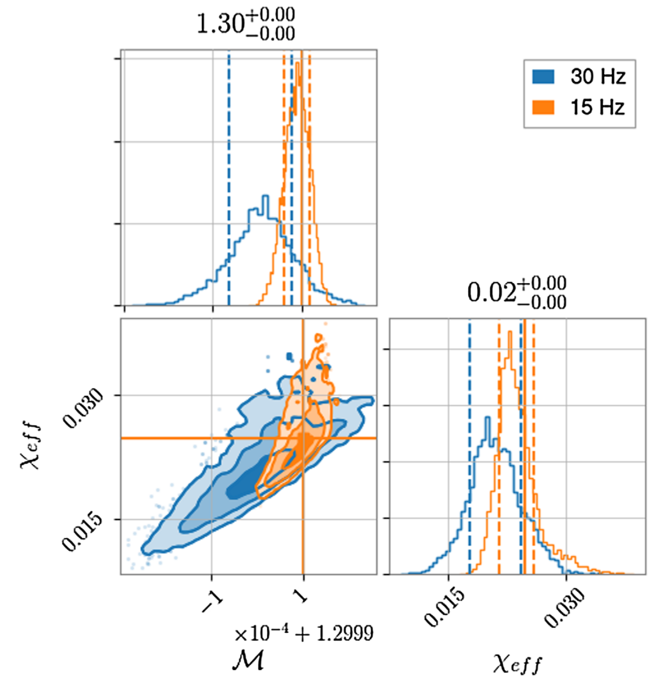

FIG. 3. Posterior distributions for the chirp mass and effective spin of a binary neutron star inspiral similar to GW170817 when beginning the analysis at 30 (blue) and 15 Hz (orange). Analyzing more of the early inspiral enables better measurement of the chirp mass, which leads to an improved measurement of the neutron star spins.

In order to demonstrate the improvement achieved analyzing longer-duration signals, we analyze a synthetic binary neutron star inspiral two different ways. First, we analyze the final 128 s of inspiral from a frequency of 30 Hz as was done for the LIGO/Virgo analysis of GW170817 [32]. Then we repeat the same analysis including 512 s worth of the inspiral with a minimum frequency of 15 Hz.

The one- and two-dimensional posterior distributions for chirp mass and effective aligned spin $\chi_{\rm eff}$ are shown in Fig. 3. Observing more of the early inspiral improves our measurement of both of these parameters. A factor of $\sim$2 improvement of the effective spin facilitates future comparisons with the galactic pulsar population [34].

IV. POPULATION ACCELERATION

In population inference, we are interested in measuring hyperparameters, Λ, describing a population of binaries (e.g., minimum/maximum black hole mass) rather than the parameters, θ, of each of the individual binaries. The population properties are often described by either phenomenological models (e.g., [35–44]) or by the results of detailed physical simulations, e.g., population synthesis or N-body dynamical simulations (e.g., [45–55]). In this work, we use the former for examples. However, our methods apply equally to both. The formalism for hierarchical inference including a discussion of selection effects is briefly described below. See, e.g., [7,56,57] for detailed derivations.

In order to analyze a population of binary black holes, we typically use the following likelihood [56],

$$\mathcal{L}_{\rm tot}(\{d_i\}|\Lambda, R) = R^N e^{-RVT(\Lambda)} \prod_i^N \int d\theta_i \mathcal{L}(d_i|\theta_i) p(\theta_i|\Lambda). \tag{2}$$

Here, $\mathcal{L}(d_i|\theta)$ is the likelihood of obtaining strain data d_i given binary parameters θ_i as in Eq. (1), $p(\theta|\Lambda)$ is our population model, and $VT(\Lambda)$ is the total observed space-time volume if the population is described by Λ. See, e.g., [58–61] for discussions of methods to calculate $VT(\Lambda)$.

Within GWPopulation we currently support the calculation of VT on a regular grid with GPU acceleration. The calculation of VT does not depend on the number of events. However, this grid-based integration is limited to small dimensional spaces, and so the subdominant effects of spin on detectability cannot be included in this method. This is mitigated by performing Monte Carlo integration. However, the cost of this computation scales as $O(N)$ [62]. We are currently developing a method which will enable spin effects to be included in the calculation of VT with no increase in cost at run-time [63]. For the benchmarking performed in this work, we ignore this quantity, but it may be added back in without significant impact on performance.

Since $\mathcal{L}(d_i|\theta)$ is independent of the population model, it can be evaluated independently for each observed binary, as described in Sec. III. Since we are interested in evaluating an integral of the likelihood over the binary parameters, it is convenient to perform a Monte Carlo integral using samples from the likelihood. This process is known as "recycling." The Bayesian inference algorithms used for single-event inference typically generate samples from the posterior distribution. Therefore, it is necessary to weight each of the samples by the prior probability distribution $p(\theta_{ij}|\varnothing)$ used during the initial inference step. This yields the following likelihood:

$$\mathcal{L}_{\rm tot}(\{d_i\}|\Lambda, R) \propto R^N e^{-RVT(\Lambda)} \prod_i^N \sum_j^{n_i} \frac{p(\theta_{ij}|\Lambda)}{p(\theta_{ij}|\varnothing)}, \tag{3}$$

where $\{\theta_j\}_i$ is a set of n_i samples drawn from the posterior distribution $p(\theta_i|d_i)$. The evaluation of the population model for each of the posterior samples is embarrassingly parallel.

The first step is to draw an equal number of samples from each posterior so that for each binary parameter we have a single $N \times n_i$ array. These arrays are then transferred to the GPU to evaluate the probabilities, sums, and logarithms. We then need to only transfer a single number, the (log) likelihood, back to the CPU when the likelihood is evaluated.

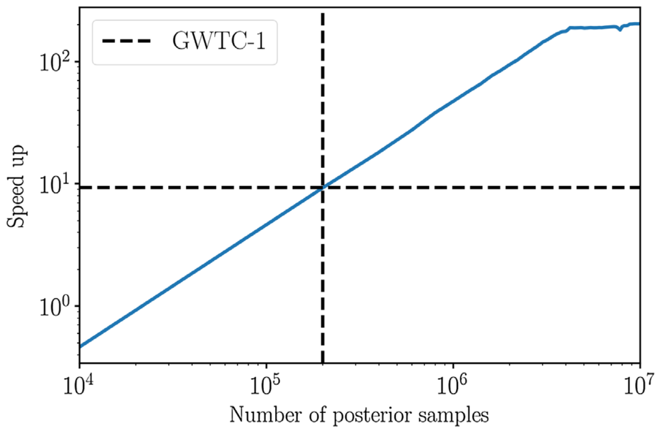

FIG. 4. Ratio of the likelihood evaluation time (top) and likelihood evaluation time (bottom) on CPU and GPU as a function of number of samples. The vertical lines indicate (left to right): the number of samples released as part of the GWTC-1 data release, the anticipated number of samples at design sensitivity, the number of samples for a week of data. With the number of samples available as part of GWTC-1, the speed-up is -up. When the number of samples exceeds 4 million we reach the limits of the available GPUs and the likelihood evaluation time begins to increase. As GPU technology improves we expect that the maximum speed-up over single-threaded code will continue to increase.

We calculate the likelihood evaluation time as a function of the number of posterior samples being recycled. The tests performed in this work use the mass distribution proposed in [40], the spin magnitude distribution from [42], and the spin orientation model from [36]. Figure 4 shows the expected linear scaling in the speed-up obtained by using the GPU. At around 3×10^6 samples, the GPU likelihood evaluation time begins increasing and the growth of the relative speed-up slows. This is due to GPU queue saturation.

The data released after the second observing run of advanced LIGO/Virgo include ten binary black hole systems and the shortest posterior contains $\sim 2 \times 10^4$ posterior samples. Using 2×10^5 samples the GPU code is $\approx 10\times$ faster, reducing run times from a week to less than a day.

During Advanced LIGO/Virgo's third observing run, beginning April 2019, we can expect to detect tens more binary black hole mergers [5]. Given the current performance, we would expect the relative speed of the GPU code and the CPU code to continue to scale linearly with the size of the observed population. Within the lifetime of current detectors, we can conservatively assume that we will detect hundreds of events. At this stage using a GPU will accelerate population inference by more than 2 orders of magnitude.

V. DISCUSSION

As the field of gravitational-wave astronomy grows, the quantity of data to be analyzed is rapidly increasing. Thus, it is necessary to constantly improve and accelerate inference algorithms. In this paper, we demonstrate multiple ways in which GPUs can aid in this endeavor. We show that multiple orders of magnitude speed-up can be achieved within the lifetime of current detectors in three areas:

(i) waveform evaluation,
(ii) CBC likelihood evaluation,
(iii) population inference.

Most of these improvements use CUPY, a python interface to CUDA, which acts as a GPU wrapper for existing C code. CUPY has also recently been used for other parameter estimation methods in gravitational-wave astronomy [21].

We provide two complementary GPU versions of commonly used waveforms, a CUDA implementation of IMRPhenomPv2 and a python implementation of TaylorF2. We find that the performance of the CUDA waveform exceeds that of the pure-python waveform for short waveforms when efficient memory allocation is vital. For longer waveforms, memory allocation is less important and the python waveforms give similar or greater speed-ups than the CUDA implementation. The CUDA implementation of IMRPhenomPv2 is available at [64]. Future development of parallelized waveforms may enable rapid evaluation of waveforms encoding more of the phenomenology of general relativity, e.g., higher-order modes [65], gravitational-wave kicks [66], or gravitational-wave memory [67].

Other than waveform evaluation, the dominant cost for the likelihood used in inference for compact binary coalescences is exponentials to perform frequency domain time shifts. This is another operation which drastically benefits from parallelization. Using these two methods, we reduce the likelihood evaluation time for binary neutron star mergers by an order of magnitude at current sensitivity and more when current detectors reach their design sensitivity. The code for performing GPU-accelerated single-event parameter estimation can be found at [23].

Other methods for speeding up likelihood evaluation for long signals include reduced order quadrature methods [13] and relative binning [15,68]. These methods rely on the waveform being sufficiently well described by a small set of unevenly sampled frequencies. For binary neutron star mergers like GW170817, the signal from ~30 Hz to the merger can be described with only $\sim 10^3$ frequencies. Additionally, these methods do not require computing any exponentials at run time. GPU waveforms will have less of a benefit for these cases. However, we may be able to accelerate parameter estimation by an additional factor of a few. This facilitates more rapid production of sky maps for electromagnetic observers following up on gravitational-wave events.

The computational cost of performing population inference increases linearly with the size of the observed population. Using a GPU to perform the embarrassingly parallel likelihood evaluation we find an acceleration of up using the data in GWTC-1 [1] compared to the CPU code. We additionally find that the GPU implementation will outperform the CPU code by more than 2 orders of magnitude during the lifetime of current detectors. We therefore present GWPopulation [25]: a CPU/GPU agnostic framework for performing gravitational-wave population inference. Both of these packages use the framework available within Bilby [18].

Within GWPopulation, we include CPU/GPU tools for performing population inference. We provide

(i) implementations of many previously proposed binary black hole population models,
(ii) the likelihoods commonly used in gravitational-wave population inference,
(iii) methods for computing selection biases.

Using our GPU-enabled implementation of a binary neutron inspiral waveform we demonstrate that beginning the analysis at lower frequencies improves our measurements of the intrinsic parameters of the system. While these results present a tantalizing glimpse of the physics that are enabled through GPU acceleration, more work is required to realize these gains.

(i) When analyzing signals for many minutes it is necessary to include the effect of the Earth's rotation in our analysis. Including the effect of the Earth's rotation will improve sky localization since the movement of the detector allows triangulation from data taken at different times. While progress on this front has been made in recent years, e.g., [69,70], a working implementation is not available at this time.
(ii) Central to the likelihood we use [Eq. (1)] is an assumption of Gaussianity and stationarity of the noise. These assumptions are not generally valid over the lengths of time considered in this paper.

The improvements between current sensitivity and the projected design sensitivities of advanced LIGO/Virgo are largest at low frequencies, so additional upgrades may be required in order to achieve the improvements shown here with real data.

As we enter "the data era" of gravitational-wave astronomy, optimizing Bayesian inference codes becomes ever more important. When the likelihood evaluation requires a large number of independent operations, GPUs can yield significant benefits.

ACKNOWLEDGMENTS

C. T., R. S., and E. T. are supported by the Australian Research Council (ARC) Grant No. CE170100004. E. T. is supported by ARC Grant No. FT150100281. This work used the OzStar supercomputing cluster and the LIGO Data Grid. This work was supported by a Software Support grant from Astronomy Data And Computing Services (ADACS). This is LIGO document P1900101.

[1] B. P. Abbott *et al.*, arXiv:1811.12907.
[2] B. P. Abbott *et al.*, Nature (London) **551**, 85 (2017).
[3] B. P. Abbott *et al.*, Phys. Rev. Lett. **121**, 161101 (2018).
[4] B. P. Abbott *et al.*, arXiv:1811.12940.
[5] B. P. Abbott *et al.*, Living Rev. Relativity **19**, 1 (2016).
[6] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian Data Analysis*, Chapman & Hall/CRC Texts in Statistical Science, 3rd ed. (Taylor & Francis, London, 2013).
[7] E. Thrane and C. Talbot, Pub. Astron. Soc. Aust. **36**, e010 (2019).
[8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
[9] W. K. Hastings, Biometrika **57**, 97 (1970).
[10] J. Skilling, AIP Conf. Proc. **735**, 395 (2004).
[11] M. Pürrer, Classical Quantum Gravity **31**, 195010 (2014).
[12] P. Canizares, S. E. Field, J. Gair, V. Raymond, R. Smith, and M. Tiglio, Phys. Rev. Lett. **114**, 071104 (2015).
[13] R. Smith, S. E. Field, K. Blackburn, C.-J. Haster, M. Purrer, V. Raymond, and P. Schmidt, Phys. Rev. D **94**, 044031 (2016).
[14] S. Vinciguerra, J. Veitch, and I. Mandel, Classical Quantum Gravity **34**, 115006 (2017).
[15] B. Zackay, L. Dai, and T. Venumadhav, arXiv:1806.08792.
[16] J. Veitch *et al.*, Phys. Rev. D **91**, 042003 (2015).
[17] C. M. Biwer, C. D. Capano, S. De, M. Cabero, D. A. Brown, A. H. Nitz, and V. Raymond, Publ. Astron. Soc. Pac. **131**, 024503 (2019).
[18] G. Ashton *et al.*, Astrophys. J. Suppl. Ser. **241**, 27 (2019).
[19] C. Pankow, P. Brady, E. Ochsner, and R. O'Shaughnessy, Phys. Rev. D **92**, 023002 (2015).
[20] J. Lange, R. O'Shaughnessy, and M. Rizzo, arXiv:1805.10457.
[21] D. Wysocki, R. O'Shaughnessy, Y.-L. L. Fang, and J. Lange, Phys. Rev. D **99**, 084026 (2019).
[22] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, in *Workshop of ML Systems, NIPS 2017* (2017), http://learningsys.org/nips17/assets/papers/paper_16.pdf.
[23] https://github.com/ColmTalbot/gpucbc.
[24] https://github.com/ADACS-Australia/ADACS-SS18A-RSmith
[25] https://github.com/ColmTalbot/gwpopulation, GWPopulation is also available through pypi.
[26] P. Ajith, M. Hannam, S. Husa, Y. Chen, B. Brügmann, N. Dorband, D. Müller, F. Ohme, D. Pollney, C. Reisswig, L. Santamaría, and J. Seiler, Phys. Rev. Lett. **106**, 241101 (2011).
[27] M. Hannam, P. Schmidt, A. Bohé, L. Haegel, S. Husa, F. Ohme, G. Pratten, and M. Pürrer, Phys. Rev. Lett. **113**, 151101 (2014).

[28] S. Khan, S. Husa, M. Hannam, F. Ohme, M. Pürrer, X. J. Forteza, and A. Bohé, Phys. Rev. D **93**, 044007 (2016).
[29] S. Khan, K. Chatziioannou, M. Hannam, and F. Ohme, Phys. Rev. D **100**, 024059 (2019).
[30] Specifically, we rewrite in CUDA the function lalsimulation.SimIMRPhenomPFrequencySequence branched from LALSuite at SHA:8cbd1b7187.
[31] Lalsuite, git.ligo.org/lscsoft/lalsuite.
[32] B. P. Abbott *et al.*, Phys. Rev. Lett. **119**, 161101 (2017).
[33] H. Yu, D. Martynov, S. Vitale, M. Evans, D. Shoemaker, B. Barr, G. Hammond, S. Hild, J. Hough, S. Huttner, S. Rowan, B. Sorazu, L. Carbone, A. Freise, C. Mow-Lowry, K. Dooley, P. Fulda, H. Grote, and D. Sigg, Phys. Rev. Lett. **120**, 141102 (2018).
[34] X. Zhu, E. Thrane, S. Osłowski, Y. Levin, and P. D. Lasky, Phys. Rev. D **98**, 043002 (2018).
[35] S. Vitale, R. Lynch, R. Sturani, and P. Graff, Classical Quantum Gravity **34**, 03LT01 (2017).
[36] C. Talbot and E. Thrane, Phys. Rev. D **96**, 023012 (2017).
[37] M. Fishbach and D. E. Holz, Astrophys. J. **851**, L25 (2017).
[38] E. D. Kovetz, I. Cholis, P. C. Breysse, and M. Kamionkowski, Phys. Rev. D **95**, 103010 (2017).
[39] D. Wysocki, arXiv:1712.02643.
[40] C. Talbot and E. Thrane, Astrophys. J. **856**, 173 (2018).
[41] M. Fishbach, D. E. Holz, and W. M. Farr, Astrophys. J. **863**, L41 (2018).
[42] D. Wysocki, J. Lange, and R. O'Shaughnessy, arXiv: 1805.06442.
[43] J. Roulet and M. Zaldarriaga, Mon. Not. R. Astron. Soc. **484**, 4216 (2019).
[44] S. M. Gaebel, J. Veitch, T. Dent, and W. M. Farr, Mon. Not. R. Astron. Soc. **484**, 4008 (2019).
[45] I. Mandel and R. O'Shaughnessy, Classical Quantum Gravity **27**, 114007 (2010).
[46] S. Stevenson, F. Ohme, and S. Fairhurst, Astrophys. J. **810**, 58 (2015).
[47] K. Belczynski, A. Heger, W. Gladysz, A. J. Ruiter, S. Woosley, G. Wiktorowicz, H.-Y. Chen, T. Bulik, R. OShaughnessy, D. E. Holz, C. L. Fryer, and E. Berti, Astron. Astrophys. **594**, A97 (2016).
[48] D. Gerosa and E. Berti, Phys. Rev. D **95**, 124046 (2017).
[49] M. Fishbach, D. Holz, and B. Farr, Astrophys. J. Lett. **840**, L24 (2017).
[50] S. Stevenson, C. P. L. Berry, and I. Mandel, Mon. Not. R. Astron. Soc. **471**, 2801 (2017).
[51] M. Zaldarriaga, D. Kushnir, and J. A. Kollmeier, Mon. Not. R. Astron. Soc. **473**, 4174 (2018).
[52] M. Zevin, C. Pankow, C. L. Rodriguez, L. Sampson, E. Chase, V. Kalogera, and F. A. Rasio, Astrophys. J. **846**, 82 (2017).
[53] D. Wysocki, D. Gerosa, R. OShaughnessy, K. Belczynski, W. Gladysz, E. Berti, M. Kesden, and D. E. Holz, Phys. Rev. D **97**, 043014 (2018).
[54] J. W. Barrett, S. M. Gaebel, C. J. Neijssel, A. Vigna-Gómez, S. Stevenson, C. P. L. Berry, W. M. Farr, and I. Mandel, Mon. Not. R. Astron. Soc. **477**, 4685 (2018).
[55] Y. Qin, T. Fragos, G. Meynet, J. Andrews, M. Sørensen, and H. F. Song, Astron. Astrophys. **616**, A28 (2018).
[56] W. M. Farr, J. R. Gair, I. Mandel, and C. Cutler, Phys. Rev. D **91**, 023005 (2015).
[57] I. Mandel, W. M. Farr, and J. R. Gair, arXiv:1809.02063.
[58] L. S. Finn and D. F. Chernoff, Phys. Rev. D **47**, 2198 (1993).
[59] M. Dominik, E. Berti, R. O 'shaughnessy, I. Mandel, K. Belczynski, C. Fryer, D. E. Holz, T. Bulik, and F. Pannarale, Astrophys. J. **806**, 263 (2015).
[60] D. Wycoski and R. O'Shaughnessy, Calibrating semi-analytic VT's against reweighted injection VT's (2018), https://dcc.ligo.org/LIGO-T1800427/public.
[61] V. Tiwari, Classical Quantum Gravity **35**, 145009 (2018).
[62] W. M. Farr, Res. Notes AAS **3**, 66 (2019).
[63] C. Talbot *et al.* (to be published).
[64] adacs-ss18a-rsmith-python.readthedocs.io/en/latest/.
[65] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, C. D. Ott, M. Boyle, L. E. Kidder, H. P. Pfeiffer, and B. Szilágyi, Phys. Rev. D **96**, 024058 (2017).
[66] D. Gerosa, F. Hébert, and L. C. Stein, Phys. Rev. D **97**, 104049 (2018).
[67] C. Talbot, E. Thrane, P. D. Lasky, and F. Lin, Phys. Rev. D **98**, 064031 (2018).
[68] N. J. Cornish, arXiv:1007.4820.
[69] S. Marsat and J. G. Baker, arXiv:1806.10734.
[70] D. Liang, Y. Gong, A. J. Weinstein, C. Zhang, and C. Zhang, Phys. Rev. D **99**, 104027 (2019).