Northwestern University

GPU Accelerated Parameter Estimation of Gravitational Waves from Binary Black Hole Mergers

Brandon B. Miller, Masters Thesis

August 22, 2016

1 Background and Motivation

Gravitational waves are predicted as a mathematical consequence of the linearized Einstein Field Equations. The Einstein Field Equations themselves are a set of ten unique, coupled, nonlinear partial differential equations.

1.1 Pieces of the Einstein Equation

1.1.1 Reimann Tensor

The Reimann Tensor is a mathematical object that encapsulates the geometric *curvature* of spacetime. The concept of a curved spacetime is fundamentally no different from the concept of a curved space, and from examining the behavior of vectors in such a space we can address both the purpose and properties of the Reimann tensor without departing from the intuitive picture of spheres and arrows.

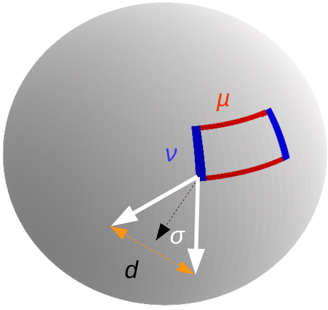


Figure 1.1: A visual depiction of the Reimann Curvature tensor. Parallel transporting a vector in the σ direction both ways around the loop shown yields two different vectors. The orange vector $\vec{d}$ points between the tips. The magnitude of the components of this vector are $R_{\rho\sigma\mu\nu}$.

The Reimann tensor is traditionally derived by examining the local behavior of geodesics subject to a metric tensor that describes curved space. Here we motivate the concept with a visual explanation of parallel transport. Consider a vector pointing in some arbitrary direction σ somewhere on the sphere, as depicted in figure 3.1. If we maintain the orientation of the vector relative to the surface of the sphere as we traverse the path, we note that the vector behaves differently depending on the order in which we choose to traverse the directions μ and ν. This difference is representable by a vector that points between the two tips - one that is dependent on the initial direction σ and both the transport directions μ and ν. Let this vector be denoted $\vec{d}$. Then the Reimann tensor, $R_{\rho\sigma\mu\nu}$, is simply d_ρ, the ρ'th component of this difference vector. From this geometric picture some of the classical symmetries that are often introduced abreast the definition of the Reimann tensor are immediately intuitive, such as the skew symmetry depicted in figure **FIGURE HERE**. Since it is somewhat unfair (worse - *incorrect*) to characterize the entire geometry of the space with looping paths over large areas, the formal definition of the Reimann tensor is in fact the infinitesimal version of the concept above. Indeed, the concept of a single direction one may traverse is only valid in a small region of curved space, so the real Reimann tensor must describe the difference between two vectors as parallel transported around an infinitesimally small version of the original loop. It follows then how there must be a (possibly different) Reimann tensor at each point on the manifold in question. More remarkable however is the simplicity of the formal definition within this picture:

$$R^\rho_{\sigma\mu\nu}\partial_\rho = \left(\nabla_\mu \nabla \nu - \nabla_\nu \nabla_\mu\right)\partial_\sigma \tag{1.1}$$

It should be noted that the above formula is true only modulo a term containing $\nabla_{[\mu,\nu]}$ except in the special case of coordinate vector fields, where it is zero. Although it does little to redeem it as a first explanation of the concept, this also illuminates why the Reimann tensor is sometimes described as encoding the non-commutativity of covariant derivatives on a manifold. Somewhat more subtle is the explicit role of the Reimann tensor in describing geodesic deviation. Crucial to a straightforward linearization of the Reimann tensor itself is the expression in terms of the Christoffel Symbols **AS DEFINED IN APPENDIX**

$$R^\rho_{\sigma\mu\nu} = \partial_\mu \Gamma^\rho_{\nu\sigma} - \partial_\nu \Gamma^\rho_{\mu\sigma} + \Gamma^\rho_{\mu\lambda}\Gamma^\lambda_{\nu\sigma} - \Gamma^\rho_{\nu\lambda}\Gamma^\lambda_{\mu\sigma} \tag{1.2}$$

The christoffel symbols contain some information about the curvature of space as a function of the coordinates, and thus in the framework of linearized gravity products of Christoffel symbols are second order:

$$R^\rho_{\sigma\mu\nu} = \partial_\mu \Gamma^\rho_{\nu\sigma} - \partial_\nu \Gamma^\rho_{\mu\sigma} \tag{1.3}$$

If $x^\rho(\tau)$ are the components of a geodesic parameterized by τ, then $\dot{x^\rho}(\tau) \equiv T^\rho$ are the components of a vector tangent to the geodesic. Suppose there is a family such nearby geodesics indexed by some other parameter, s, and that we can define *deviation vector* $S^\rho(\tau) = \partial_s x^{rho}(s,\tau)$ which characterizes the separation between nearby geodesics. It can be shown that the second derivative of this vector with respect to τ is related to the Reimann tensor through the differential equation

$$\ddot{S}^{\rho} = R^{\rho}_{\sigma\mu\nu} T^{\sigma} T^{\mu} S^{\nu} \tag{1.4}$$

2 Linearized Gravity

In linearized gravity we assume the spacetime is described by a flat spacetime $g_{\mu\nu}$ plus a small perturbation

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu} \tag{2.1}$$

Since the other pieces of the Einstein field equations are directly derived from the metric, they must be individually linearized to form a description of spacetime around weak sources. The standard treatment of linearized gravity assumes that to first order we may raise and lower indices using the Minkowski metric. If we tried to use the full form of equation **EQUATION**, we would have

$$\begin{aligned} g^{\mu}_{\nu} &= g^{\mu\alpha} g_{\alpha\nu} && (2.2)\\ &= (\eta^{\mu\alpha} + h^{\mu\alpha})(\eta_{\alpha\nu} + h_{\alpha\nu}) && (2.3)\\ &= \eta^{\mu\alpha}\eta_{\alpha\nu} + \eta^{\mu\alpha} h_{\alpha\nu} + h^{\mu\alpha}\eta_{\alpha\nu} + h^{\mu\alpha} h_{\alpha\nu} && (2.4) \end{aligned}$$

The second two terms being second order in h, ther peturbation, are neglected in linearized gravity. We assume from here on that indices can be freely raised and lowered to first order using only the Minkowski metric $\eta_{\mu\nu}$.

$$h^{\mu}_{\nu} = \eta^{\alpha\mu} h_{\alpha\nu} \quad \text{and} \quad h^{\mu\nu} = \eta^{\alpha\mu}\eta^{\beta\nu} h_{\alpha\beta} \tag{2.5}$$

The next natural line of logic to pursue is to write down the Christoffel symbols for such a spacetime

$$\begin{aligned} \Gamma^{\mu}_{\alpha\beta} &= \frac{1}{2} g^{\nu\mu} \left[\partial_{\beta} g_{\alpha\nu} + \partial_{\alpha} g_{\beta\nu} - \partial_{\nu} g_{\alpha\beta}\right] && (2.6)\\ &= \frac{1}{2}\left(\eta^{\mu\nu} + h^{\mu\nu}\right)\left[\partial_{\beta} g_{\alpha\nu} + \partial_{\alpha} g_{\beta\nu} - \partial_{\nu} g_{\alpha\beta}\right] && (2.7) \end{aligned}$$

Inserting the expanded form of the metric into the term within the brackets gives us six terms times two at the front, for twelve total. Thankfully, three correspond to the standard flat-space Christoffel symbols which can be made zero through a choice of coordinates (in fact, orthonormal Cartesian coordinates), three involve derivatives of $\eta_{\mu\nu}$ which are zero, and three are second order in $h_{\mu\nu}$, which we assume are zero. This leaves us with three:

$$\Gamma^{\mu}_{\alpha\beta} = \frac{1}{2}\eta^{\mu\nu}\left[\partial_\beta h_{\alpha\nu} + \partial_\alpha h_{\beta\nu} - \partial_\nu h_{\alpha\beta}\right] \tag{2.8}$$

$$= \frac{1}{2}\left[\partial_\beta h^{\mu}_{\alpha} + \partial_\alpha h^{\mu}_{\beta} - \partial^{\mu} h_{\alpha\beta}\right] \tag{2.9}$$

Where in the last term $\partial^\mu = \eta^{\mu\nu}\partial_\nu$ was used to raise the index on the actual partial derivative operator itself. It should be noted at this point that since we neglect terms that are second order in the Christoffel symbols themselves, in linearized gravity, the covariant derivative of a metric perturbation is equivalent to the normal partial derivate. The same assumption allows for simplification of the Reimann and Ricci tensors:

$$R^{\rho}_{\mu\sigma\nu} = \partial_\sigma \Gamma^{\rho}_{\mu\nu} - \partial_\nu \Gamma^{\rho}_{\mu\sigma} \tag{2.10}$$

$$R_{\mu\nu} = R^{\rho}_{\mu\rho\nu} = \partial_\rho \Gamma^{\rho}_{\nu\mu} - \partial_\mu \Gamma^{\rho}_{\mu\rho} \tag{2.11}$$

Plugging in equation **EQUATION** we have

$$R_{\mu\nu} = \frac{1}{2}\partial_\alpha\left[\partial_\nu h^{\alpha}_{\mu} + \partial_\mu h^{\alpha}_{\nu} - \partial^\alpha h_{\mu\nu}\right] - \frac{1}{2}\partial_\nu\left[\partial_\alpha h^{\alpha}_{\mu} + \partial_\mu h^{\alpha}_{\alpha} - \partial^\alpha h_{\mu\alpha}\right] \tag{2.12}$$

$$= \frac{1}{2}\left[\cancel{\partial_\alpha\partial_\nu h^{\alpha}_{\mu}} + \partial_\alpha\partial_\mu h^{\alpha}_{\nu} - \partial_\alpha\partial^\alpha h_{\mu\nu} - \cancel{\partial_\nu\partial_\alpha h^{\alpha}_{\mu}} - \partial_\nu\partial_\mu h^{\alpha}_{\alpha} + \partial_\nu\partial^\alpha h_{\mu\alpha}\right] \tag{2.13}$$

$$= \frac{1}{2}\left[\partial_\alpha\partial_\mu h^{\alpha}_{\nu} + \partial_\nu\partial^\alpha h_{\mu\alpha} - \partial_\alpha\partial^\alpha h_{\mu\nu} - \partial_\nu\partial_\mu h^{\alpha}_{\alpha}\right] \tag{2.14}$$

The quantity h^{α}_{α} is the trace of the pertubation metric and represents, in a sense, an overall measure of the *strength* or *amplitude* of the perturbation, this is often called just h. The term $\partial^\alpha\partial_\alpha$ is an inner product that when formed through the Minkowski metric becomes the wave operator, $\Box = -\partial_t^2 + \nabla^2$. The Ricci tensor is obtained by contracting the Reimann tensor over the two free indices:

$$R \equiv \eta^{\mu\nu} R_{\mu\nu} = \frac{1}{2}\left[\partial_\alpha\partial^\mu h^{\alpha}_{\mu} + \partial_\mu\partial^\alpha h^{\mu}_{\alpha} - \Box h - \Box h\right] \tag{2.15}$$

Since α and μ are summed indices, the first two terms are the same. Dropping the $\frac{1}{2}$, the Ricci scalar curvature is

$$R = \partial_\alpha\partial_\mu h^{\mu\alpha} - \Box h \tag{2.16}$$

Many authors choose to simplify the expression for the Ricci Tensor by defining the quantity

$$V_\nu = \partial_\alpha h^{\alpha}_{\nu} - \frac{1}{2}\partial_\nu h \tag{2.17}$$

With which the first two terms of the linearized Ricci tensor can be reorganized as

$$\partial_\mu \partial_\alpha h^\alpha_\nu - \partial_\mu \partial_\nu h = \partial_\mu \left[\partial_\alpha h^\alpha_\nu - \partial_\nu h\right] \tag{2.18}$$

$$= \partial_\mu V_\nu - \frac{1}{2}\partial_\mu \partial_\nu h \tag{2.19}$$

Which upon substitution into the **EQUATION** for $R_{\mu\nu}$ yields

$$R_{\mu\nu} = \frac{1}{2}\left[\partial_\mu V_\nu - \frac{1}{2}\partial_\mu \partial_\nu h + \partial^\alpha \partial_\nu h_{\mu\alpha} - \Box h_{\mu\nu}\right] \tag{2.20}$$

The two unsimplified terms form the quantity $\partial_\nu \left[\partial^\alpha h_{\mu\alpha} - \frac{1}{2}\partial_\mu h\right]$ which we recognize as $\partial_\nu V_\mu$, leaving us with

$$R_{\mu\nu} = \frac{1}{2}\left[\partial_\mu V_\nu + \partial_\nu V_\mu - \Box h_{\mu\nu}\right] \tag{2.21}$$

Setting the right hand side of this equation gives us the linearized, vacuum Einstein Equation $R_{\mu\nu} = 0$:

$$\frac{1}{2}\left[\partial_\mu V_\nu + \partial_\nu V_\mu - \Box h_{\mu\nu}\right] = 0 \tag{2.22}$$

$h_{\mu\nu}$ is a two index object in a four dimensional spacetime, which in the general case would present with sixteen independent components each of which would need to be explicitly determined to form a complete picture of the physics. However in our specific case it will turn out that there are far fewer real independent components then this, which can be observed by chiseling away at the above expression in the following manner. Firstly, the metric of general relativity is taken to be symmetric, because if we assume that the tensor $dx^\mu dx^\nu$ is symmetric (i.e. dx^ν and dx^ν commute) then only the symmetric part of any $g_{\mu\nu}$ contributes when the quantity $g_{\mu\nu} dx^\mu dx^\nu$ is computed. Thus we have

$$h_{\mu\nu} = h_{\nu\mu} \tag{2.23}$$

Which, for a sixteen component tensor, makes six of the components redundant. This leaves us with ten. Furthermore, there is *gauge freedom* present in the linearized vacuum Einstein equation. This is best illustrated with reference to the equations of classical electrodynamics, which describe the motion of charged particles subject to the electric and magnetic vector potentials, V and $\vec{A}$ respectively. It is somewhat easily demonstrable that the same equations of motion for electrically charged particles are unchanged when V and $\vec{A}$ are subject to transformations of the form

$$\vec{A} \to \vec{A} + \nabla\Psi \tag{2.24}$$

$$V \to V - \frac{\partial \Psi}{\partial t} \tag{2.25}$$

The function *Psi* is known as the *gauge function* and it can be absolutely anything we want, as long as we adjust the potentials accordingly. Within such a framework, one can impose certain conditions on the potentials that simplify the mathematics, such as requiring that

$$\nabla \cdot \vec{A} + \frac{1}{c}\frac{\partial V}{\partial t} = 0 \tag{2.26}$$

$$\to \partial^{\mu} A_{\mu} = 0 \tag{2.27}$$

It is somewhat counterintuitive why imposing this particular condition on $\vec{A}$ and V themselves is equivalent to transforming the potentials via the gauge function, but indeed it is, it is just hidden! While we have made no reference to Ψ, we have *implicitly* assumed that there is such a Ψ that will *make* the restriction on $\vec{A}$ and V true. It actually is there inside the V and $\vec{A}$ written in equation **EQUATION**. We do not have to state *what it is*, as long as we know that it exists. A proof that there legitimately is such a gauge function exists but is out of the scope of this outline. There is a one-to-one correspondance of this formalism with the equivalent gauge in general relativity, which is known as the Lorentz gauge. In the same sense as equation **EQUATION**, it turns out that the equations of motion for massive particles in a slightly perturbed spacetime are invariant under coordinate transformations of the form

$$x^{\mu} \to x'^{\mu} - \xi^{\mu} \tag{2.28}$$

In particular, we can show that there exists a coordinate transformation such that

$$\partial_{\mu} V_{\nu} = \partial_{\mu}\partial_{\alpha} h^{\alpha}_{\nu} - \frac{1}{2}\partial_{\mu}\partial_{\nu} h = 0 \tag{2.29}$$

PUT PROOF HERE

$$\text{PUT PROOF HERE} \tag{2.30}$$

Thus coordinate transformations that obey $\Box\xi = 0$ will allow us to impose the Lorentz gauge. Within this gauge, the linearized, vacuum Einstein equation becomes simply

$$\Box h_{\mu\nu} = 0 \tag{2.31}$$

This is a four dimensional wave equation for the remaining components of $h_{\mu\nu}$. Moreover, we have shown that under coordinate transformations that do not change the underlying physics, there exists an explicit relationship between components of the metric given by equation **EQUATION**. In the same way that symmetry shows that six of the metric components to be functions of the others, The index ν indicates that in four dimensions there are four additional explicit relationships between metric components that correspond to a loss of four additional degrees of freedom. This leaves us with six.

3 Bayesian Statistics

We select a Bayesian posterior probability as our main analysis technique as its ability to quantify and propagate uncertainties in parameter estimates is well documented by **CITATION** and **PROBABLY OTHER CITATIONS**. Bayes' Theorem follows from the symmetric definition of conditional probability depicted visually in figure **??**. Let the joint probability evaluated at any point (x, y) in the plane be denoted $P(x, y)$. We may take a cross section through the joint probability distribution along any direction we choose, let us call such a cross section parallel to the x axis the conditional probability of x given y, and denote it $P(x|y)$. A perpandicular cross section would simply be $P(y|x)$. We see that we can compute $P(x, y)$ in two equivalent ways: scaling $P(x|y)$ by $P(y)$ or by scaling $P(y|x)$ by $P(x)$. Thus we have that

$$P(x|y)P(y) = P(y|x)P(x) \tag{3.1}$$

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{3.2}$$

Which is Bayes' theorem. The denominator of the expression on the right hand side is often rewritten as a sum over all possible situations in which y is an outcome. Reading $\neg x$ as "not x", we have

$$P(y) = P(y|x)P(x) + P(y|\neg x)P(\neg x) \tag{3.3}$$

$$= \sum_{k=1}^{K} P(y|x_k)P(x_k) \tag{3.4}$$

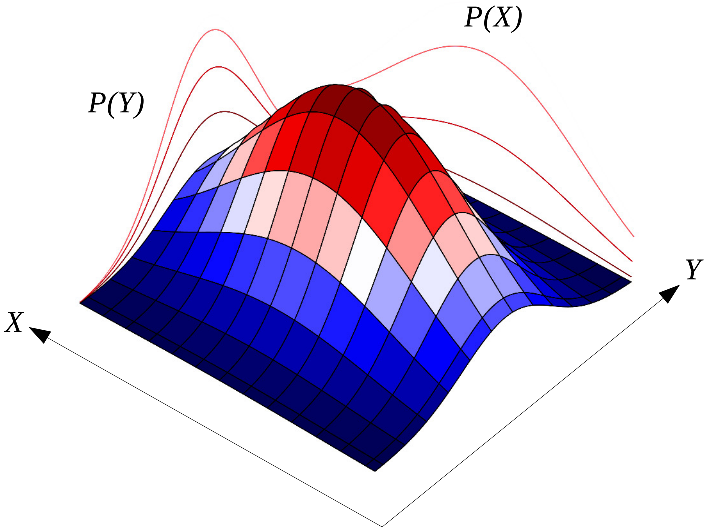


Figure 3.1: A joint probability distribution $P(x, y)$. The contours parallel with the x direction represent different $P(y|x)$.

This allows us to rewrite the posterior probability as

$$P(x|y) = \frac{P(y|x)P(x)}{\sum_{k=1}^{K} P(y|x_k)P(x_k)} \tag{3.5}$$

Multiplying the right hand side of this equation by the quantity $\frac{1}{P(x)P(y|\neg x)}$ one has

$$P(x|y) = \frac{\Lambda(x|y)}{\Lambda(x|y) + \frac{P(\neg x)}{P(x)}} \tag{3.6}$$

Where we have used the *likelihood ratio* $\Lambda(x|y) \equiv \frac{P(y|x)}{P(y|\neg x)}$ to simplify the expression. The likelihood ratio has a simple interpretation, it is simply the ratio of the probability of event y occuring in the case event x occurs and the probability of event y occuring given that event x *does not* occur. In fact, this form of Bayes theorem indicates that the posterior probability for x given y is directly proportional to the likelihood ratio. This is particularly important in the Bayesian treatment of gravitational waves where the central question for study is the probability of observing the data in question given that there may or may not be a signal present.

3.1 Signal Processing

Noise in the LIGO detectors is modeled as *Gaussian Noise*, which describes a length N set of time series samples $\{x_j\}$ with the following probability of observation:

$$P(\{x_j\}) = \left[\frac{1}{\sqrt{2\pi}\sigma}\right]^N e^{-\frac{1}{2\sigma^2}\sum_{j=0}^{N-1} x_j^2} \tag{3.7}$$

If we allow the $\{x_j\}$ to be weighted by the noise sensitivity in frequency space (also known as the *power spectral density*) $S(f) = \sum_{k=0}^{K} n^*(f_k)n(f_k) \equiv \langle n(f)|n(f)\rangle$ then we may write the probability density for any Gaussian noise process as

$$P_x(\{x(t)\}) \propto e^{-\frac{1}{2}\langle x(t)|x(t)\rangle} \tag{3.8}$$

For gravitational wave signal processing we are interested in evaluating the equivalent of the likelihood ratio in equation 3.6 for the null and signal present hypotheses, $\mathcal{H}_0$ and $\mathcal{H}_1$ respectively. This is

$$\Lambda(\mathcal{H}_1|d) = \frac{P(d|\mathcal{H}_1)}{P(d|\mathcal{H}_0)} \tag{3.9}$$

In the above equation $\mathcal{H}_0$ is in direct correspondence with $\neg x$ and $\mathcal{H}_1$ is in direct correspondence with x. If there is no signal present in the data, then the data and noise are equal, meaning

$$d(t) = n(t) \tag{3.10}$$

$$P(d(t)|\mathcal{H}_0) \propto e^{-\frac{1}{2}\langle d(t)|d(t)\rangle} \tag{3.11}$$

Whereas if there is a signal present in the data, then the noise is the *residual* signal after the gravitational wave potion of the signal $h(t)$ is subtracted from the total signal $d(t)$:

$$d(t) = n(t) + h(t) \tag{3.12}$$

$$p(d(t)|\mathcal{H}_1) \propto e^{-\frac{1}{2}\langle d(t)-h(t)|d(t)-h(t)\rangle} \tag{3.13}$$

So the likelihood ratio for gravitational wave signal processing is exactly equal to

$$\Lambda(\mathcal{H}_1|d(t)) = \frac{e^{-\frac{1}{2}\langle h(t)-d(t)|h(t)-d(t)\rangle}}{e^{-\frac{1}{2}\langle d(t)|d(t)\rangle}} \tag{3.14}$$

By equation 3.1 the posterior probability of a gravitational wave signal with parameters μ is given by

$$P[\vec{\mu}, \mathcal{H}_1|d(t)] = \frac{P[d(t)|\vec{\mu}, \mathcal{H}_1]P(\vec{\mu})}{P[d(t), \mathcal{H}_1]} \tag{3.15}$$

Extending equation 3.3 to the case of a continuous state space we have for the denominator

$$P[d(t), \mathcal{H}_1] = \int_\Omega P[d(t)|\vec{\mu}, \mathcal{H}_1]P(\vec{\mu})d\vec{\mu} \tag{3.16}$$

Where Ω indicates integration over the entire probabilistic state space. Dividing both the top and bottom of the right hand side of equation 3.15 by $P(d(t)|\vec{\mu}, \mathcal{H}_1)$ gives us

$$P(\vec{\mu}, \mathcal{H}_1|d(t)) = \frac{\Lambda(d(t)|\vec{\mu}, \mathcal{H}_1)P(\vec{\mu})}{\int_\Omega \Lambda(d(t)|\vec{\mu}, \mathcal{H}_1)P(\vec{\mu})d\vec{\mu}} \tag{3.17}$$

Note that the integral in the denominator may be factored into components that are functions of independent groups of parameters. For the purposes of gravitational wave data analysis it is factored into intrinsic and extrinsic parameters, the former (masses, spins) being responsible for the actual orbital dynamics of the system, and the latter (distance, inclination, right ascension, declination) depending only on the nature of the observer relative to the source. We label these sets of paramters $\vec{\lambda}$ and $\vec{\theta}$ respectively. The denominator of equation 3.17 is thus factorizable as

$$\int_{\Omega} \Lambda(d(t)|\vec{\mu}, \mathcal{H}_1) d\vec{\mu} = \int_{\Omega} \Lambda(d(t)|\vec{\lambda}, \mathcal{H}_1) P(\vec{\lambda}) d\vec{\lambda} \int_{\Omega} \Lambda(d(t)|\vec{\theta}, \mathcal{H}_1) P(\vec{\theta}) d\vec{\theta} \tag{3.18}$$

Note that while evaluating the likelihood for a single set of parameters in the numerator of the above equation may not be any serious challenge, computing the integral in the denominator is. Many schemes **CITE BEN** have been developed to tackle this particular problem and generally involve simulating the posterior through MCMC sampling. These techniques are highly developed and at the time of writing can provide converged results in **SOME TIMESCALE**, however they are limited in their parallelism due fundamental serialization of the algorithm. Cutting edge implementations involving parallel tempering and some forms of ensemble MCMC have shown promising results, however none can take advantage of the many thousands of cores availble across the LIGO data grid. A new and more embarassingly parallel scheme is needed.

3.2 Harmonic Mode Decomposition

For a single detector k We can write our gravitational wave signal $h_k(t) = h_{+,k}(t) - ih_{\times,k}(t)$ as a sum over products of harmonic mode strain values $h_{l,m}(\vec{\lambda})$ and corresponding spin-weighted spherical harmonics, scaled by the reference distance and antenna factor:

$$h_k(t) = \frac{D_r}{D} F_k(\theta) \sum_{(l,m)} h_{k,(l,m)}(t, \vec{\lambda}) Y_{(l,m)}(\vec{\theta}) \tag{3.19}$$

Where D_r is the luminosity reference distance to the binary. We substitute this expression for the signal into the expression for the likelihood ratio we find, taking the logarithm, that

$$\mathcal{L} = \prod_k e^{-\frac{1}{2}\langle d-h_k|d-h_k\rangle + \langle d|d\rangle} \tag{3.20}$$

$$\ln \mathcal{L} = -\frac{1}{2} \sum_k \left[\langle d - h_k|d - h_k\rangle + \langle d|d\rangle\right] \tag{3.21}$$

$$= \frac{1}{2} \sum_k \left[-\cancel{\langle d|d\rangle} + \langle d|h_k\rangle + \langle h_k|d\rangle + \langle h_k|h_k\rangle + \cancel{\langle d|d\rangle}\right] \tag{3.22}$$

$$\tag{3.23}$$

Now, since for complex vector spaces $\langle a|b\rangle = \langle b|a\rangle *$, we have that $\langle d|h_k\rangle + \langle h_k|d\rangle = 2\Re \langle h_k|d\rangle$. Then,

$$\ln \mathcal{L} = \frac{1}{2}\sum_k \left[2\left\langle h_k|d\right\rangle + \left\langle h_k|h_k\right\rangle\right] \tag{3.24}$$

$$= \frac{1}{2}\sum_k \Re\left[\langle 2\frac{D_r}{D}F_k(\vec{\theta})\sum_{(l,m)} h_{k,(l,m)}(\lambda)Y_{(l,m)}(\vec{\theta})|d\rangle\right] \tag{3.25}$$

$$+\left[\frac{D_r}{D}\right]^2 F_k^2(\vec{\theta})\sum_{(l,m),(l',m')} h_{(l,m)}(\vec{\lambda})h_{(l',m')}(\vec{\lambda})Y_{(l,m)}(\vec{\theta})Y_{(l',m')}(\vec{\theta}) \tag{3.26}$$

The defining feature of this form of the likelihood is the separation between intrinsic and extrinsic parameters. The intrinsic parameters of the binary exist only within the $h_{k,(l,m)}$, whereas the parameters that depend on the observer are contained entirely within the antenna factor and spherical harmonics. Work by **CITE CHRIS AND RICHARD** has shown that this can be exploited to accelerate computation of the likelihood by precomputing the orbital dynamics of the binary and marginalizing out all of the extrinsic parameters with a Monte Carlo integration.

4 Implementation

The main weight of the computational work described above falls under an expensive Monte Carlo integration of the evidence as part of the Bayesian posterior. Zooming in it quickly becomes apparent that the main challenge to be overcome is efficient evaluation of the function to be integrated, in this case the factored likelihood. Prior to this work, the factored likelihood is was computed with a series of looping structures that individually compiled all the terms of the equation in an independent and serial manner. The process took substantial advantage of many high level operations available within Python to match up the correct terms across various harmonic mode time series, spherical harmonics, and antenna patterns. Thanks to clever optimization and gratuitous use of fast numerical Python libraries such as NumPy, the code was able to compute approximately 10^3 likelihood evaluations on the order of seconds, the limiting factor being the serial nature of loops and the inherent lethargy of high level languages such as Python. Although NumPy commonly passes target data to compiled routines written in faster languages such as C, some portions of the computation (in particular the marginalization over time) benefit little from this capability and form a bottleneck for efficient evaluation of the likelihood. Furthermore the original implementation heavily relied upon older structures within the existing LIGO Algorithms Library (LAL), a set of C routines bound to Python through Swig, to perform many intermediate calculations, such as computing spherical harmonics. The goal of the first phase of this work was thorough vectorization of the process used to build the terms of the likelihood utilizing the BLAS subroutines available through NumPy as often as possible, as well as reducing dependency on LAL. To that end, the components of the factored likelihod were reworked.

4.1 Complex Antenna Factor

The complex antenna factor is a complex number that modulates the gravitational wave signal as a function of incoming direction. This is necessary due to the non isotropic sensitivity of the LIGO interferometers themselves. F_+ and $F_\times$.

$$\begin{aligned}
F_+ = {} & [R_{0,0}X_0 + R_{0,1}X_1 + R_{0,2}X_2]X_0 - [R_{0,0}Y_0 + R_{0,1}Y_1 + R_{0,2}Y_2]Y_0 && (4.1)\\
& + [R_{1,0}X_0 + R_{1,1}X_1 + R_{1,2}X_2]X_1 - [R_{1,0}Y_0 + R_{1,1}Y_1 + R_{1,2}Y_2]Y_1 && (4.2)\\
& + [R_{2,0}X_0 + R_{2,1}X_1 + R_{2,2}X_2]X_2 - [R_{2,0}Y_0 + R_{2,1}Y_1 + R_{2,2}Y_2]Y_2 && (4.3)
\end{aligned}$$

Collecting all the positive terms of this expression gives

$$\begin{aligned}
F_+ = {} & R_{0,0}X_0X_0 + R_{0,1}X_0X_1 + R_{0,2}X_0X_2 && (4.4)\\
& + R_{0,0}X_0X_0 + R_{0,1}X_0X_1 + R_{0,2}X_0X_2 && (4.5)\\
& + R_{0,0}X_0X_0 + R_{0,1}X_0X_1 + R_{0,2}X_0X_2 && (4.6)
\end{aligned}$$

At which point we see that the same result is computable as an outer product of the vector $\vec{X}$ using

$$F_+ = \vec{X}\mathbf{R}\vec{X} - \vec{Y}\mathbf{R}\vec{Y} \tag{4.7}$$

And the complex part of the gravitational wave is thus

$$F_\times = \vec{X}\mathbf{R}\vec{Y} + \vec{Y}\mathbf{R}\vec{X} \tag{4.8}$$

One a sample-to-sample basis, the numbers that vary are the components of the vectors $\vec{X}$ and $\vec{Y}$. Thus we need a function that takes as input a *list* vectors (or *vector*) of vectors and produces a vector with the right components as output. To that end we define the tensor X^i_j where

$$X^i = \begin{bmatrix} X^i_0 \\ X^i_1 \\ X^i_2 \end{bmatrix} \tag{4.9}$$

As well as the tensor R^i_{jk} where

$$R^i = \begin{bmatrix} R^i_{00} & R^i_{01} & R^i_{02} \\ R^i_{10} & R^i_{11} & R^i_{12} \\ R^i_{20} & R^i_{21} & R^i_{22} \end{bmatrix} \tag{4.10}$$

The tensor X is like a stack of all the different possible $\vec{X}$ coming out of the page. The Tensor R is like n copies of the matrix $\mathbf{R}$ stacked on top of each other. In this way the desired vector is obtainable with the tensor contraction

$$
\begin{align}
F_{+}^{i} &= X^{lm} R_{lj}^{i} X_{m}^{j} - Y^{lm} R_{lj}^{i} Y_{m}^{j} \tag{4.11}\\
F_{\times}^{i} &= X^{lm} R_{lj}^{i} Y_{m}^{j} + Y^{lm} R_{lj}^{i} X_{m}^{j} \tag{4.12}\\
& \tag{4.13}
\end{align}
$$

4.2 Vectorized Single Detector Log Likelihood

With the spherical harmonics and antenna factor in hand the following set of operations yield the factored likelihood for a single detector, the network likelihood being a simple sum over all of the detectors.
Equation 24 from Arxiv 15502.05370$v1.pdf$ for the network log likelihood reads

$$
\begin{align}
\ln \mathcal{L} =& \frac{D_{ref}}{D} Re \sum_{k} \sum_{(l,m)} \left[F_k Y_{lm}\right]^{*} Q_{k,lm} \tag{4.14}\\
&- \left[\frac{D_{ref}}{2D}\right]^2 \sum_{k} \sum_{(l,m),(l',m')} \left[|F_k|^2 Y_{l,m}^{*} Y_{l',m'} U_{k,(l,m),(l'm')}\right] \tag{4.15}\\
&- \left[\frac{D_{ref}}{2D}\right]^2 \sum_{k} \sum_{(l,m),(l',m')} Re\left[F_k^2 Y_{l,m} Y_{l'm'} V_{k,(l,m),(l'm')}\right] \tag{4.16}
\end{align}
$$

Consider a single detector, thus dropping the sum over k. The first term is of the form $\vec{A} \cdot \vec{B} = \sum_{i=0}^{d} A_i B_i$, so if $Q_{k,(l,m)}$ were a simple vector, we could write it as

$$
- \left[\frac{D_{ref}}{2D}\right]^2 \sum_{k} \sum_{(l,m),(l',m')} \left[|F_k|^2 Y_{l,m}^{*} Y_{l',m'} U_{k,(l,m),(l'm')}\right] = - \left[\frac{D_{ref}}{2D}\right]^2 F * \vec{Y} \cdot \vec{Q} \tag{4.17}
$$

However the $Q_{k,(l,m)}$ are actually harmonic mode time series and not single values. We desire a vector whose values are the likelihoods at each point in the time series.
Consider the case where we have only the $(2,-2),(2,0)$ and $(2,2)$ modes. If we write all the mode time series $Q^0, Q^1, Q^2 ...$ as the columns of a matrix , then the desired result is obtained with

$$
F * \begin{bmatrix} Q_{2,-2}^{0} & Q_{2,+0}^{0} & Q_{2,+2}^{0} \\ Q_{2,-2}^{1} & Q_{2,+0}^{1} & Q_{2,+2}^{1} \\ Q_{2,-2}^{2} & Q_{2,+0}^{2} & Q_{2,+2}^{2} \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} (Y_{2,-2}) \\ \\ (Y_{2,+0}) \\ \\ (Y_{2,+2}) \end{bmatrix} = \begin{bmatrix} Q_{2,-2}^{0} Y_{2,-2} + Q_{2,-2}^{0} Y_{2,+0} + Q_{2,+2}^{0} Y_{2,+2} \\ Q_{2,-2}^{1} Y_{2,-2} + Q_{2,-2}^{1} Y_{2,+0} + Q_{2,+2}^{1} Y_{2,+2} \\ Q_{2,-2}^{2} Y_{2,-2} + Q_{2,-2}^{2} Y_{2,+0} + Q_{2,+2}^{2} Y_{2,+2} \\ \vdots \end{bmatrix} \tag{4.18}
$$

With $\vec{Y}$ and $\mathbf{Q}$ defined as the matrix and vector above respectively, we have for the first term

$$\frac{D_{ref}}{D} Re \left[\mathbf{Q}\left(F\vec{Y}\right)^*\right] \tag{4.19}$$

The second term is a sum once over all the possible combinations of $(l, m), (l', m')$ pairs using the $U_{(l,m),(l',m')}$ cross terms. Its result is a scalar quantity made up of terms like

$$\begin{aligned} &Y^*_{2,-2}Y_{2,-2}U_{(2,-2),(2,-2)} + Y^*_{2,-2}Y_{2,+0}U_{(2,-2),(2,+0)} + Y^*_{2,-2}Y_{2,+2}U_{(2,-2),(2,+2)} && (4.20)\\ +\, &Y^*_{2,+0}Y_{2,-2}U_{(2,+0),(2,-2)} + Y^*_{2,+0}Y_{2,+0}U_{(2,+0),(2,+0)} + Y^*_{2,+0}Y_{2,+2}U_{(2,+0),(2,+2)} && (4.21)\\ +\, &Y^*_{2,+2}Y_{2,-2}U_{(2,+2),(2,-2)} + Y^*_{2,+2}Y_{2,+0}U_{(2,+2),(2,+0)} + Y^*_{2,+2}Y_{2,+2}U_{(2,+2),(2,+2)} && (4.22) \end{aligned}$$

If we pack the $U_{(l,m),(l',m')}$ into the matrix $\mathbf{U}$ as defined below then the following set of matrix operations produces the same sum

$$\begin{bmatrix} Y^*_{2,-2} & Y^*_{2,+0} & Y^*_{2,+2} \end{bmatrix} \begin{bmatrix} U_{(2,-2),(2,-2)} & U_{(2,-2),(2,+0)} & U_{(2,-2),(2,+2)} \\ U_{(2,+0),(2,-2)} & U_{(2,+0),(2,+0)} & U_{(2,+0),(2,+2)} \\ U_{(2,+2),(2,-2)} & U_{(2,+2),(2,+0)} & U_{(2,+2),(2,+2)} \end{bmatrix} \begin{bmatrix} Y_{2,-2} \\ Y_{2,+0} \\ Y_{2,+2} \end{bmatrix}$$

because when you multiply $\mathbf{U}$ into $\vec{Y}$ this simplifies to

$$\begin{bmatrix} Y^*_{2,-2} & Y^*_{2,+0} & Y^*_{2,+2} \end{bmatrix} \begin{bmatrix} U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} \\ U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} \\ U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} + U_{(2,-2),(2,-2)}Y_{2,-2} \end{bmatrix}$$

Which becomes the desired scalar. This allows us to write the second term as

$$-\left[\frac{D_{ref}}{2D}\right]^2 \sum_k \sum_{(l,m),(l',m')} \left[|F_k|^2 Y^*_{l,m} Y_{l',m'} U_{k,(l,m),(l'm')}\right] = -\left[\frac{D_{ref}}{2D}\right]^2 |F^2|\vec{Y}^*\mathbf{U}\vec{Y} \tag{4.23}$$

We must always set up the spherical harmonic vectors based on the value of m and the cross terms in row-major form based first on m_2 and then on m_1. If we organize the matrix $\mathbf{V}$ in the same way then the same set of steps will lead us to conclude that

$$-\left[\frac{D_{ref}}{2D}\right]^2 \sum_k \sum_{(l,m),(l',m')} Re\left[F_k^2 Y_{l,m} Y_{l'm'} V_{k,(l,m),(l'm')}\right] = -\left[\frac{D_{ref}}{2D}\right]^2 Re\left[F^2\vec{Y}\mathbf{V}\vec{Y}\right] \tag{4.24}$$

Combining the results the single detector log likelihood is

$$\ln\mathcal{L} = \frac{D_{ref}}{D}\Re\left[\mathbf{Q}\left(F\vec{Y}\right)^*\right] - \left[\frac{D_{ref}}{2D}\right]^2 \left[|F|^2\vec{Y}^*\mathbf{U}\vec{Y} - \Re\left(F^2\vec{Y}\mathbf{V}\vec{Y}\right)\right] \tag{4.25}$$

4.3 GPU Implementation

Vectorization of the code produces a performance improvement due to SIMD instruction sets available on many modern microprocessor architectures. This allows for some on-chip parallelism that the compiler may use to unroll loops within the machine code and increase efficiency. Beyond this however the process of recasting the computation into clear matrix and tensor operations illuminates a higher level of parallelism possible on the hardware level. A substantial amount of both academc and industrial research and development has centered around efficient parallel implementations of the equivalent operations, with impressive results. Reaching the limit of the sample throughput with conventional serial python libraries, our investigation turned to GPU based acceleration as the only means forward. The following is the result of this investigation.

4.4 Hardware Limitations and Logical Programming

GPUs work significantly differently from CPUs and must be paired with one to operate, they serve only as accelerators for complex operations and not as standalone units. Data must be passed back and forth from host memory on the node to RAM placed directly on the card itself through expensive transfers via the PCI bus. This is known as device global memory. While the onboard RAM on bleeding edge cards exceeds 12GB, memory transfers are the dominant cost of many GPU programs and must be minimized. Once the data exists on the device, it is accessed and manipulated by anywhere between nine and fifteen logical cores called Streaming Multiprocessors. These cores operate concurrently and contain a smaller granularity of parallel processing units that *may* operate concurrently depending on the individual resources required by the requested operations. Programming in CUDA involves launching groups of threads called *grids* that are further subdivided into *blocks.* The blocks themselves are indexed by local variables that are rapidly accessible by all the contained threads, and the threads themselves all have access to indices placed within registers that exist for the lifetime of the thread in question. Both the blocks and and grids can be up to three dimensional and thus each block and thread may have access to up to three registers containing block indices and three containing thread indices. In addition thread blocks have access to an extremely high speed shared memory space where ideally most of the computations are performed, however it is limited in size to **SIZE** and is not designed to hold the entire target dataset. One major design goal for CUDA programs is implementing a process which copies portions of memory from the global space to the shared, processes it, and copies it back, while using the thread and block indices to blanket the dataset with threads and control which threads access what data. The current CUDA API allows for the logical launch of **HOW MANY** blocks with a maximum of 1024 threads per block, for a total of *HOW MANY* possible "concurrent" threads. It is important to note that these threads are not *physically* concurrent. Indeed, if each of the logical threads containing six integers in registers were to exist simultaneously, the GPU would be storing **HOW MANY** registers at once! Similarly perposterous amounts of shared memory would have to exist. In actuality, the shared memory and registers are located on the logical cores themselves, and blocks thread blocks line up to be processed. How many are processed at once is heavily dependent on the resources requested by the blocks themselves. Defining too many local variables in a kernel function can lead to *register pressure* and decreased *occupancy* of the multiprocessors on the GPU, a similar effect can come from requesting too much shared memory. Furthermore, the blocks are not guaranteed to execute in any particular order. While the compiler is responsible for sorting out how many blocks can be processed at a time as well as the order, the programmer is responsible for performance tuning with regard to optimizing block occupancy. This is often a late stage design consideration, but also leads to some of the greatest returns after algorithmic optimization has been completed.

4.5 PyCuda

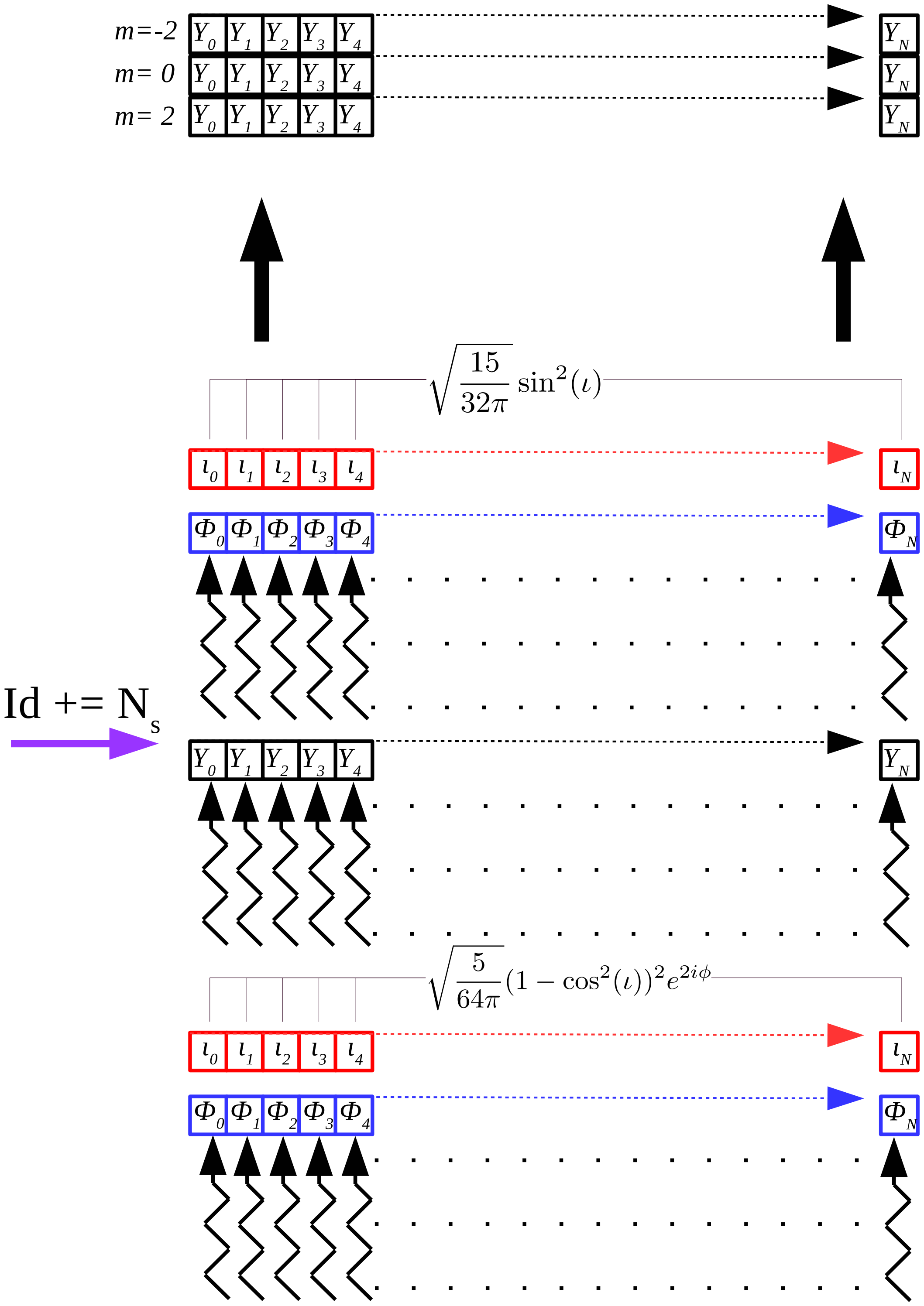
PyCuda is a flexible library for GPU scripting that includes a broad spectrum of GPU based array operations built to mimic the usage of NumPy arrays within a program. It allows for relatively easy access to the Nvidia CUDA API directly from within Python, and many of its operations are directly compatible with NumPy arrays with careful casting to the correct data types. Certainly one of the most challenging aspects of incorporating GPU code into a higher level language such as Python is the interface layer. As it stands much of the widely used LAL software consists of C code bound to Python through complex Swig interfaces, and while this is a perfectly valid way to access faster libraries, it can often be deceptively difficult to work with as many of the resulting Python objects contain read-only data structures masquerading as dynamic Python objects. This sometimes yields unpredictable behavior and it was decided this should be avoided as a way to interface with CUDA C. PyCuda is a very elegant solution to this problem as it seamlessly bridges this gap using the native GPUArray class, and a set of functions that create GPUArray copies of NumPy arrays that are built initially on the node CPU hosts. With a smaller group of developers, the built-in GPUArray class contained within PyCuda does not yet support *all* of the rich spectrum of high level functions available within the NDArray NumPy class. Perhaps more importantly however, PyCuda allows for custom CUDA kernel functions to be written as pure CUDA C directly within special containers called source modules, that are contained within the main body of the Python code. PyCuda actually calls the real Nvidia compiler at runtime and links the resulting temporary executable to the main program. One may call these functions with NumPy arrays as arguments, as if they were normal Python functions. This is perhaps the most important feature of PyCuda as it allows the developer to tune the customization of the code to whatever level is desired. Those intermediate operations that may be handled with the built in methods of the GPUArray are performed as such. More complex operations that are not supported can be custom built with source modules. If desired, the entire program may consist of CUDA C with no more then basic startup routines written in actual Python meant to instantiate the original data and pass it down to the GPU. This particular work used mostly custom kernels, for the simple reason that applying the same matrix operation to a contiguous block of memory containing multiple slightly different copies constitutes tensor operations similar in nature to the results of section **SECTION**, for which there is no general support for in PyCuda.

4.6 General Design Considerations

Computing the time marginalized likelihood for a single sample involves processing the likelihood for each point in a signal time series, multiplied again by the number of associated harmonic modes. As the number of samples selected by the integrator as an adaptation chunk increases, the global memory required for the program to execute becomes dominated by these time series. Consequently the code is built upon managing and manipulating a large contiguous block of memory whose rows contain the harmonic mode time series for each sample, this lends itself naturally to using a two dimensional grid of one dimensional blocks as the launch configuration for most kernel functions. The one dimensional blocks lie conceptually along the rows and may line up if the number of time series samples exceeds the maximum threds per block. The grid extends downwards in the y-direction and will generally assign a row of blocks to each sample, or in some cases blanket the memory entirely in threads. It is the nature of CUDA programs that thread blocks operate most efficiently when the total number of threads per block is a power of two. Aside from the microprocessor architecture optimizing for these cases, memory reads by groups of thirty two threads (known as a *Warp*) are coalesced by the compiler and recieve the value to be read as a broadcast. This massively boosts throughput if used correctly and it is thus advantageous in most cases to cast the problem dimensions into some workable multiple of the maximum threads per block on the hardware. For most Nvidia cards this is 1024. It is substantially easier to pad the time series with zeros out to the next greatest multiple of this number then to design specialized cleanup blocks whose sole purpose is to process the remainder. In the worse case this corresponds to holding **NUMBER** extra zeros in memory, which can be a painful amount of unecessary overhead in some scenarios. These factors come together to make memory management the main bottleneck for the entire algorithm, and a main area of focus for further development of this method.

4.7 Spherical Harmonics

The spin-weighted spherical harmonics are precomputed at startup. There is no analytic formula that returns the spin-weighted spherical harmonic for arbitrary quantum numbers l, m and s. The most efficient way of retrieving them closely follows the method used by the current, serial LAL functions, which implement a lookup table that contains the explicit formulas. This is often sufficient as most waveform templates require only the $l = 2$ modes, and thus only the corresponding spherical harmonics. Indeed, the LSC has yet to require support for $s \neq 2$, so the corresponding CUDA function does not attempt to extend this capability.


m=-2
m= 0
m= 2
Y_0 Y_1 Y_2 Y_3 Y_4
Y_N
$\sqrt{\frac{15}{32\pi}}\sin^2(\iota)$
ι_0 ι_1 ι_2 ι_3 ι_4
ι_N
Φ_0 Φ_1 Φ_2 Φ_3 Φ_4
Φ_N
Id += N_s
$\sqrt{\frac{5}{64\pi}}(1-\cos^2(\iota))^2e^{2i\phi}$

Computation of Spherical Harmonics, $l = 2$.

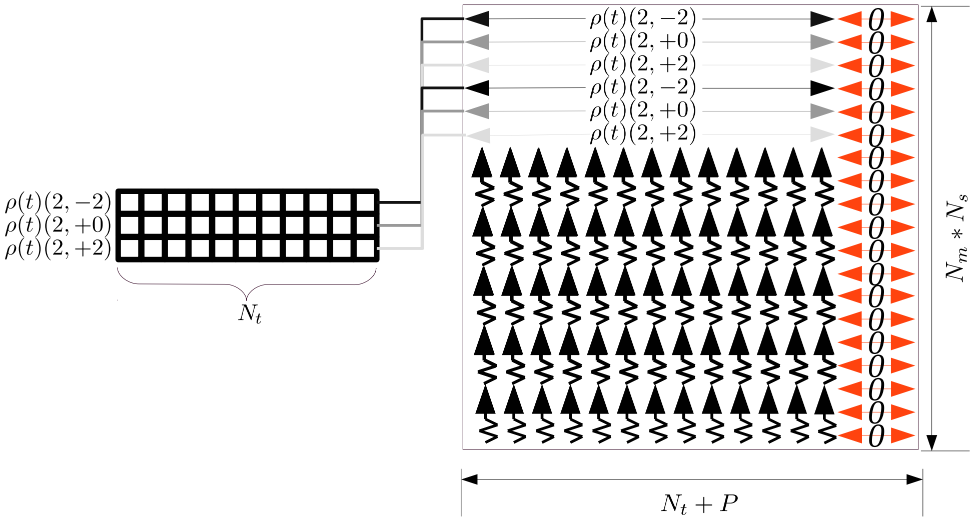
Instead, it takes as argument an ordered list of m values and executes the following set of steps to return a row-major ordered block of results whose rows correspond to values of m: Launch a one dimensional grid of one dimensional blocks, with one thread per sample. Compute the answer for this value of m and place the result in global memory. Increment the local thread index by the number of samples and repeat. As long as it loops through the m values in order, the results will be placed correctly, as per figure 4.7. As the extrinsic parameter values used in computation are accessed relatively few times, this function does not use shared memory. Regardless as a standalone function it is able to compute several billion results per second, so little further optimization is necessary.

4.8 Complex Antenna Factor

The antenna factor requires somewhat less fancy footwork then the spherical harmonics, as there is only one per sample and not an arbirary multiple depending on the number of selected modes (which are not necessarily all the values between $-l$ and l. The position vectors corresponding to the extrinsic parameters are contracted with the detector response tensor in a thread-wise fashion: one thread for each sample. We avoid using BLAS based routines to perform the contraction due to the small size of the matrices involved, startup routines for BLAS might take longer then a simple loop over nine elements of the detector tensor. We again use a one dimensional grid of one dimensional blocks, however since the position vectors contain commonly accessed elements, we choose to store these independently in device shared memory as well as the detector tensor itself in even higher speed device constant memory. This makes the antenna factor computation one of the most efficient steps of the process.

4.9 Main Routine

Due to the variation in extrinsic parameters over the samples, it is necessary for each to manipulate a separate group of copies of the harmonic mode time series. The number of samples may be large, so the main block of memory intended to hold these time series is handed down to the GPU initialized to zeros and a GPU function is used to expanded the time series into a vertial stack of copies, per figure 4.9. To streamline the marginalization process this block of memory is padded with zeros out to the nearest multiple of T_{max}, which is generally either 512 or 1024 depending on the GPU architecture.



Expansion of time series. One thread per relevant item in resulting memory block. Zeros are padded out to the nearest multiple of T_{max}.

The result of this expansion of the time series into GPU memory is the desired vertical stack of time series copies.

The next step is to transform these time series into term one of equation **EQUATION**. This involves combining the antenna factor (for which there is one complex number per sample) with the spin-weighted spherical harmonics (for which there are N_m complex numbers per sample) into a single quantity and inserting them into the correct positions within the time series block. It is more efficient to complete the combination of the antenna factor and spherical harmonics prior to multiplying them into the time series, and this action is easily completed in an elementwise fasion using the built in machinery of PyCuda. It is steps like these for which PyCuda is an essential tool for performing what would otherwise be complex GPU operations with a single line of code.

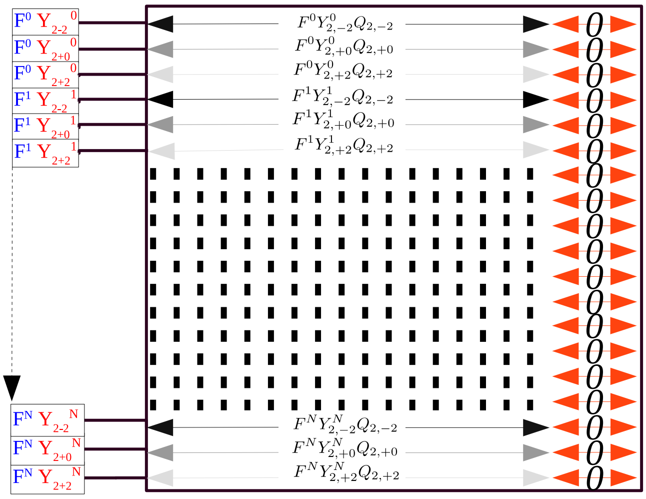


Figure 4.1: Insertion of the antenna factor-spherical harmonic pairs.

This step coupled with the previous is all that is necessary to build the first term of the likelihood for all of the samples on the GPU, the rest of the work is simply a series of summing operations. These must be handled somewhat carefully due to the possibility of race conditions and queued memory accesses, both of which can hinder or dangerously skew the results of a computation. The main issue stems from the fact that there is no guarantee of the order of execution of threadblocks within a program. This means that any program that depends upon memory reads and writes occuring in a specific order is prone to producting incorrect results. For this reason it is important to design the summation process in a way that does not exhibit this behavior: each thread should only read and write to memory locations unique to it. Note that each of the functions described thus far follows this paradigm. Queued memory are serialized by the CUDA compiler. This means that memory reads to the same location by multiple threads take place one by one. While not as dangerous as a race condition, it is a nontrivial performance consideration. Note that again the process thus far has been designed to avoid these situations. This is particularly important for justifying the nature of the next series of steps.

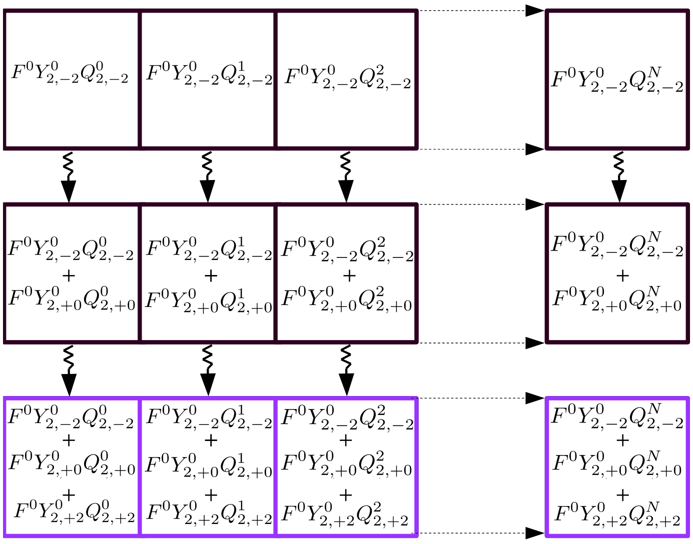
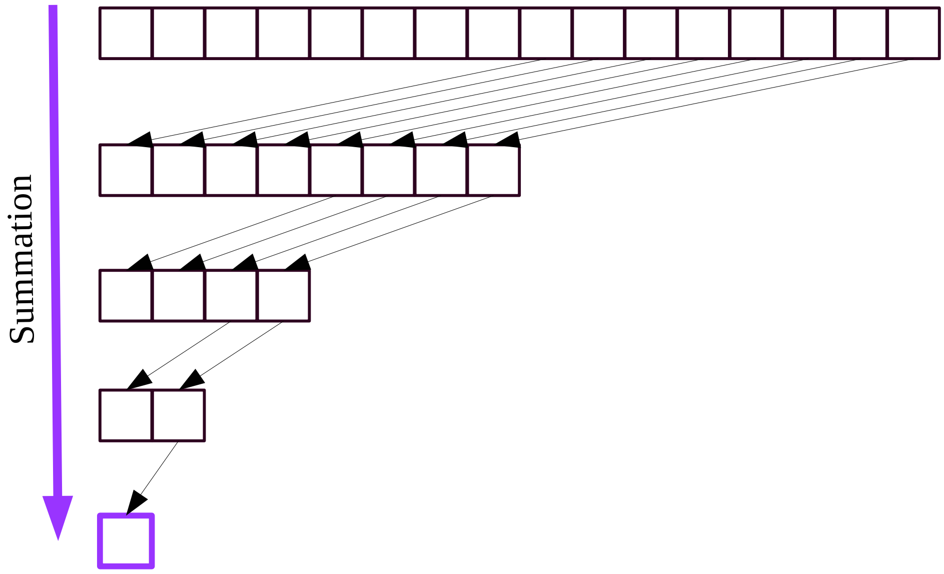

$F^0Y^0_{2,-2}Q^0_{2,-2}$
$F^0Y^0_{2,-2}Q^1_{2,-2}$
$F^0Y^0_{2,-2}Q^2_{2,-2}$
$F^0Y^0_{2,-2}Q^N_{2,-2}$
$F^0Y^0_{2,-2}Q^0_{2,-2}$ + $F^0Y^0_{2,+0}Q^0_{2,+0}$
$F^0Y^0_{2,-2}Q^1_{2,-2}$ + $F^0Y^0_{2,+0}Q^1_{2,+0}$
$F^0Y^0_{2,-2}Q^2_{2,-2}$ + $F^0Y^0_{2,+0}Q^2_{2,+0}$
$F^0Y^0_{2,-2}Q^N_{2,-2}$ + $F^0Y^0_{2,+0}Q^N_{2,+0}$
$F^0Y^0_{2,-2}Q^0_{2,-2}$ + $F^0Y^0_{2,+0}Q^0_{2,+0}$ + $F^0Y^0_{2,+2}Q^0_{2,+2}$
$F^0Y^0_{2,-2}Q^1_{2,-2}$ + $F^0Y^0_{2,+0}Q^1_{2,+0}$ + $F^0Y^0_{2,+2}Q^1_{2,+2}$
$F^0Y^0_{2,-2}Q^2_{2,-2}$ + $F^0Y^0_{2,+0}Q^2_{2,+0}$ + $F^0Y^0_{2,+2}Q^2_{2,+2}$
$F^0Y^0_{2,-2}Q^N_{2,-2}$ + $F^0Y^0_{2,+0}Q^N_{2,+0}$ + $F^0Y^0_{2,+2}Q^N_{2,+2}$

Figure 4.2: Downwards summation of the harmonic modes to form term one of the factored likelihood for a group of samples.

The time series are summed downwards within memory by single threads. One row of threads is launched per sample, this leaves N_m values for each thread to collect in a summation that resides in the bottom row. The memory locations accessed and written to by the individual threads in the grid are unique to that thread, avoiding races and queued memory accesses. The result are rows that are the same as those that would have been formed taking the product $\mathbf{Q}(F\vec{Y})$, this is the first term of the factored likelihood. It remains only to build the second term and subtract it from the rows of this matrix to complete the computation.


Summation