Masters Dissertation

THE SUPPORT OF ANY NUMBER OF DETECTORS IN A GRAVITATIONAL WAVE DETECTION PIPELINE

Thomas Hill Almeida (21963144)*
Supervisors: Prof. Linqing Wen†, Qi Chu†

2020-10-19

Word count: 7518

*Software Engineering, University of Western Australia
†Department of Physics, University of Western Australia

Contents

1 Introduction **4**

2 Literature Review **7**
2.1 Supporting any number of detectors work in other gravitational wave detection pipelines . 7
2.2 CUDA 8
2.3 Parallelised Complexity Analysis 8

3 Design process **10**
3.1 Design Constraints 11
3.2 Employed tools 12
3.3 Relevant code 13
3.4 Evaluation criteria 14

4 Final Design **14**
4.1 Patches 14
4.1.1 Mapping combinations of detectors to integers 14
4.1.2 Replacing individual detector variables with C arrays 15
4.2 Testing 16
4.3 Evaluation 16

5 Discussion **17**
5.1 Comparison to previous "state of the art" 18
5.1.1 SPIIR pipeline 18
5.1.2 Other pipelines 18
5.2 A complexity analysis of the parallel post-processing of the SPIIR pipeline 18
5.2.1 Maximum element reduction 19
5.2.2 Determining the number of samples over a signal-to-noise threshold 20
5.2.3 Transposing the input matrices 20
5.2.4 Determining the coherent correlation and statistical value of data points 21
5.2.5 Calculating heat skymaps 22
5.2.6 Overall complexity 22
5.2.7 Implications 23

6 Future work **23**

7 Conclusion **24**

References **24**

A Patches **26**
A.1 Making `IFOComboMap` be sums of powers of two 26
A.2 Removing hard-coded detector names 44

List of Figures

1 The structure of the SPIIR pipeline. 6
2 An example of calculating work for a merge-sort like algorithm. 10
3 An example of calculating span for a merge-sort like algorithm. 11
4 Simplified callgraph for coherent post-processing in the SPIIR pipeline. 19

List of Tables

1 An evaluation of the implementation of supporting any number of detectors against the criteria specified in section 3.4 17

Abstract

A large number of gravitational-wave detectors will become operational in the next few years, which presents a great opportunity for existing gravitational-wave detection pipelines to reduce their false detection rate and improve the accuracy of localisation. The ability to easily support the addition of new detectors to the pipelines, however, is an area that requires much improvement, with the SPIIR pipeline potentially requiring the modification of approximately 1000 lines of code across 16 files to add the support of one additional detector. This presents a challenge for the existing gravitational pipelines, as development work that could be spent on improving the latency, precision or accuracy of the pipeline would instead have to be spent preparing the pipeline for the introduction of the new detectors. A solution proposd by this dissertation reduced the development cost of supporting additional detectors in the SPIIR pipeline to just 9 lines of code across 2 files by introducing a method of mapping combinations of detectors to integers based on powers of two and replacing individual detector variables with C arrays. In addition, this work explores the consequences of adding new detectors to the potential latency of the pipeline by applying parallelised asymptotic complexity analysis to its gravitational-wave searching algorithm, finding that the runtime of the algorithm scales cubically with the number of detectors.

1 Introduction

Gravitational waves have been postulated to exist since Albert Einstein's publication of his general theory of relativity, as massive accelerating objects would cause 'ripples' in the curvature of spacetime [1]. Direct detection of gravitational waves, however, remained beyond the reach of the scientific community until 2015, when the Laser Interferometric Gravitational-Wave Observatory (LIGO [see 2]) reported a direct observation of a gravitational wave on the 14th of September [3, 4].

Due to their design, the detectors in use for gravitational wave detection are affected by a significant amount of noise from other sources, whilst the gravitational waves themselves have very weak signals. As such, a large amount of data processing must be done to the outputs produced by the detectors in order to filter and extract any possible gravitational waves. These data processors are known as "pipelines", and have historically been created by research groups that are a part of the LIGO Scientific Collaboration (LSC [see 5]), and are used throughout observation runs for real-time data analysis.

The Summed Parallel Infinite Impulse Response (SPIIR [see 6]) pipeline, based on the SPIIR method originally implemented by Shaun Hooper in 2012, uses a number of IIR (infinite impulse response) filters

to approximate possible gravitational wave signals for detection [7]. The output of the ith IIR filter can be expressed with the equation [8]:

$$y_k^i = a_1^i y_{k-1}^i + b_0^i x_{k-d_i}, \tag{1}$$

where a_1^i and b_0^i are coefficients, k is time in a discrete form and x_{k-d_i} denotes input with some time delay d_i. After summing the output of the filters, the resulting signal undergoes coherent post-processing (see section 5.2 and [9, chapter 4]) to determine the likelihood of an event having occurred.

The pipeline is currently thought to be the fastest of all existing pipelines, is the only pipeline that implements coherent search, and has participated in every observation run since November 2015, successfully detecting most of the events seen in more than one detector.

The SPIIR pipeline uses GStreamer, a library for composing, filtering and moving around signals, in addition to the GStreamer LIGO Algorithm Library (`gstlal`) [10]. After receiving data from the detectors, the pipeline performs data conditioning and data whitening, followed by the usage of the IIR filters. The data is then combined for post-processing, where events are given sky localization and then inserted into the LIGO event database [8].

The structure of the SPIIR pipeline can be seen in figure 1.

At the time of the start of this research, the SPIIR pipeline supported the use of two or three detectors for gravitational wave detection — the two American LIGO detectors and the Italian Virgo detector — although additional interferometers are likely to be introduced soon. This presents several issues with the existing pipeline design. As with many of the other gravitational wave detection pipelines, providing support for additional detectors is a significant undertaking for the development team, with many hours of work and testing needing to be completed. As the number of available interferometers continues to grow, development work that could be spent on improving the optimisation, precision, or accuracy of the pipeline would instead have to be spent allowing for those detectors to be used.

Section 2 shall explore the existing literature on the implementation of any number of detectors in other gravitational wave detection pipelines, in addition to exploring CUDA and complexity analysis, two tools that will be used in the analysis of the final design. Section 3 will look at the existing pipeline structure and its deficiencies for supporting any number of detectors. It will also determine any constraints a design for implementing the support of any number of detectors would have, and will provide a framework for evaluating the success of the resulting design. The final design will be presented in section 4, and will explore the implementation details of the design and how it interacts with the existing programming interface that SPIIR provides. A discussion about the implications of this design will be done in section 5, with additional research on those implications being presented there. Finally, section 6 will provide some suggestions for future research that can be based on this research.

This dissertation aims to layout the design and implementation work done to provide the SPIIR pipeline with the ability to easily support the addition of any number of detectors for gravitational-wave searching without hindering the existing functionality of the pipeline.

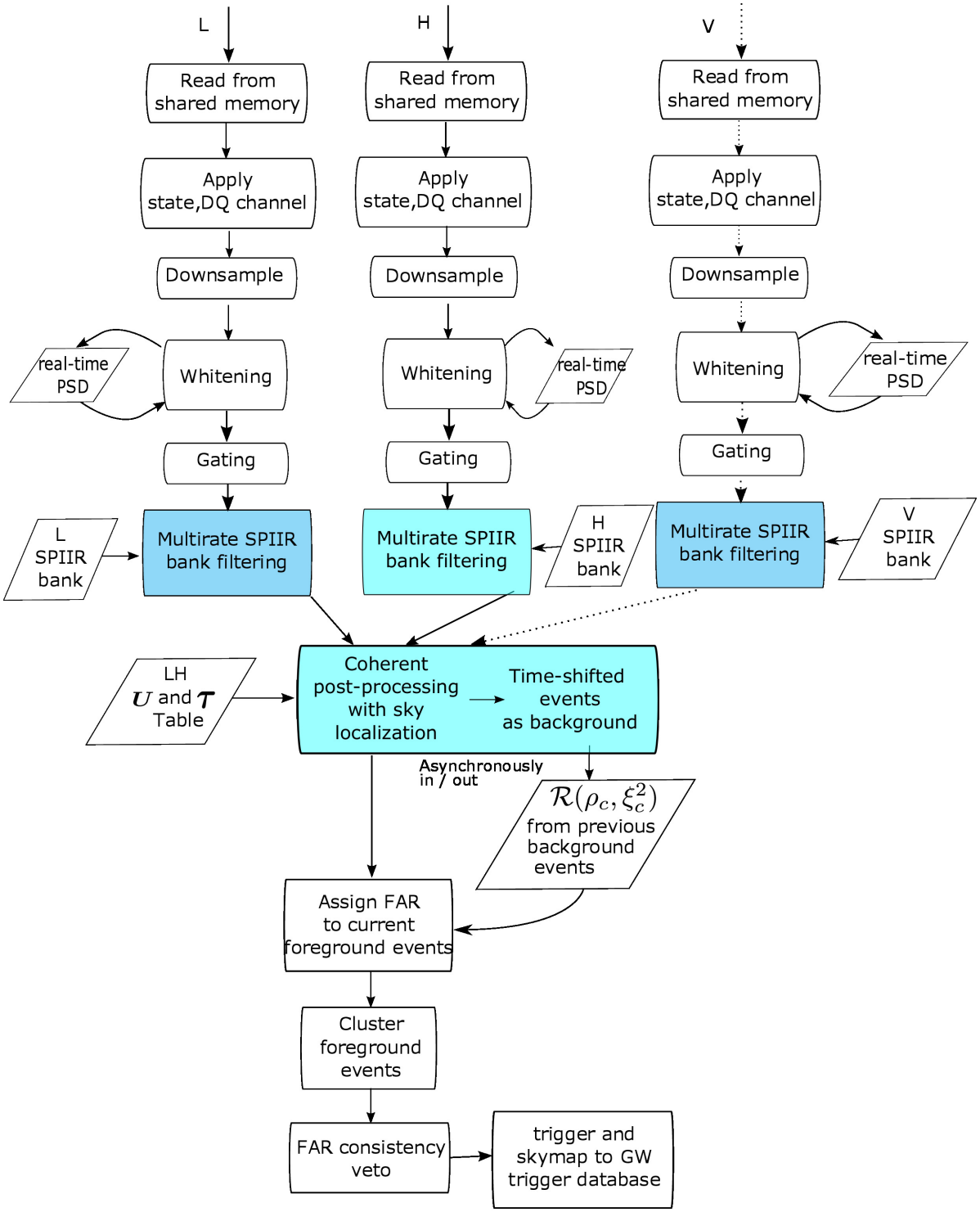

Figure 1: The structure of the SPIIR pipeline.

2 Literature Review

This section explores the existing research that is relevant to this project. Section 2.1 will explore the implementations that other gravitational wave detection pipelines have used to deal with a growing number of detectors in their algorithms and interfaces, whilst section 2.2 will outline the computational model of CUDA, the library that SPIIR uses for parallelism. Finally, section 2.3 will summarise the literature about parallel complexity analysis, which will allow for the later analysis of the pipeline in section 5.2.

2.1 Supporting any number of detectors work in other gravitational wave detection pipelines

There are, of course, other gravitational wave detection pipelines that may also have to consider the issue of dealing with a growing number of detectors. In the process of designing a new architecture to allow for any number of detectors to be used, it is well worth examining the methods that the other pipelines may use to shape the resulting design.

Most detection pipelines use a "coincidence" search to determine whether a gravitational wave event has occurred. Coincidence search, is defined as a process which includes finding candidates for gravitational waves from individual detectors, identifying temporal coincidences and producing measures to rank candidate events [9, chapter 3]. In more simple terms, a coincidence search considers events which can be seen in a single detector, and then sanity checks that other detectors may have seen them at the same time.

In contrast, the SPIIR pipeline uses a "coherent" search to determine whether a gravitational wave event has occurred, using the maximum likelihood ratio principle to consider specific parameters of a potential signal [9, chapter 4]. As such, it is unlikely that all of the principles for dealing with additional detectors may translate directly to being able to be used for the SPIIR pipeline, as the method of search differs.

PyCBC is one of the best known toolkits for gravitational wave astronomy, and was one of the pipelines used in the original 2015 gravitational wave detection [11]. From an examination of the codebase of the latest version of PyCBC [12, October 2020], it can be observed that the codebase itself makes no direct mention of detectors — instead it provides a generic `Detector` class as a wrapper around LALSuite's [13] `LALDetector` structure for validation, which in turns provides utilities for returning information about the detector as well as methods for getting readings from it.

This allows PyCBC to provide an entirely generic gravitational wave searching algorithm library for any input detectors (although the algorithm only supports using two detectors at a time) providing that the detectors are in the LALSuite library — and adding support for additional detectors is a simple matter of updating the dependency on LALSuite.

The approach to supporting new detectors that PyCBC uses is close to ideal — it shifts the burden of the support to a well-maintained third-party library. PyCBC's lack of support, however, for using any more than two detectors at a time for gravitational-wave searching limits the applicability of its approach for the SPIIR pipeline, as SPIIR must be able to search using any subset of the supported detectors.

GstLAL is gravitational wave detection library, that exposes components of LALSuite [13] as GStreamer elements for use in other analysis pipelines — including the SPIIR pipeline — as well as providing its own pipeline for processing raw signals from detectors into lists of gravitational wave candidates [14]. The GstLAL's pipeline hard-codes the detector names into both its inputs and its outputs, however the algorithm used for detection itself is actually generic on which detectors are used [15, 16].

This means that the process of adding support for detectors involves changing a number of different files, as well as modifying several internal data structures [17] — so the existing GstLAL design is non-optimal in terms of the ease of adding new detectors for use.

2.2 CUDA

CUDA [18] is an extension of the C++ programming language created by NVIDIA that allows for the development of GPU-accelerated applications. In 2018, the SPIIR pipeline had multiple components rewritten in CUDA to take advantage of the high number of simultaneous threads available compared to CPUs [8]. As such, it is worth understanding the computational model of CUDA for the analysis of the SPIIR pipeline.

In CUDA, each individual sequence of instructions being executed is called a *thread*. By its nature, a highly-parallelised environment such as GPUs will run many individual threads, which are partitioned into *warps*, a group of (typically 32) threads. Warps are the smallest unit that GPUs schedule, and all threads in a warp must execute the same instruction – although each thread maintains its own instruction pointer and can branch independently from the warp at a small performance cost. The performance cost of branching within a warp means that a major optimisation that does not affect computational complexity in CUDA can be simply reducing the number of branches. Warps are further organised into thread blocks, which contain a small amount of fast memory shared between the threads in the block. Blocks in CUDA are typically executed on the same Simultaneous Multiprocessor (SM). The CUDA Programming Guide states that the number of blocks and warps that can reside and be processed together on an SM depends on the number of registers and shared memory available on the SM, as well as on a CUDA defined maximum number of blocks and warps [19].

For the purpose of actual time-based computation, the maximum number of threads that can run at any given time is determined by a few factors of the CUDA runtime; the maximum number of resident warps per SM; the maximum number of resident threads per SM; the number of 32-bit registers per thread; the number of 32-bit registers per SM; the number of 32-bit registers per thread block; and the amount of shared memory in each of those divisions. Thus, one major determining factor in any speed-up given by a CUDA operation can be determined by the ability to split the workload across threads and thread blocks so that the number of registers and used memory is well balanced across threads.

2.3 Parallelised Complexity Analysis

As the pipeline changes to accommodate additional detectors, it is important that the impact that this has on the pipeline's runtime is considered. The SPIIR pipeline is designed to be as low latency as possible, and an asymptotic complexity analysis of components that may be impacted by the addition of new detectors allows for the measurement of the potential runtime cost of doing so. The SPIIR pipeline

has been parallelised using CUDA [8], and thus determining the asymptotic complexity of components of the pipeline requires different considerations to that of a sequential program.

According to [20], the theoretical efficiency of a multi-threaded or parallelised algorithm can be measured using the metrics of 'span', 'work', 'speed-up' and 'parallelism', all of which should be considered in the context of a directed acyclic graph (DAG) of operations in the algorithm. The ***work*** of a parallelised computation is the total time to execute the entire computation sequentially on a single processor, and can be found by summing the total work of every vertex in the DAG. An example of ***work*** for a merge-sort like algorithm can be seen in figure 2, which has a work of $O(N \log N)$. In comparison, the ***span*** of a parallelised computation is the maximum time taken to complete any path in the DAG. An example of ***span*** for a merge-sort like algorithm can be seen in figure 3, which has a work of $O(N)$. It should be noted that the actual running time of a parallelised computation also depends on the number of processors available for computation and how they are allocated to perform different tasks in the DAG, and thus denoting the running time of parallelised computation on P processors as T_P is also common practice. This leads to work being denoted at T_1 (the time taken to run on a single processor) and span being denoted as T_∞ (the time taken on an infinite number of processors). Another helpful metric is ***speed-up***, which shows how the algorithm scales with additional processors as $S_P = \frac{T_1}{T_P}$. ***Parallelism*** can thus be defined as the maximum possible speed-up on an infinite number of processors, and thus as $p = \frac{T_1}{T_\infty}$.

Using the above definitions, several laws that provide lower bounds on the running time of T_P can be re-derived.

In one step, a computer with P processors can do P units of work, and thus in T_P time can perform PT_P units of work. As the total work to be done as per above is T_1, the ***work law*** states that [20]:

$$T_P \geq \frac{T_1}{P}. \tag{2}$$

It is also evident that a computer with P processors cannot run any faster than a computer with an infinite number of processors, as the computer with an infinite number of processors can emulate a computer with P processors by using a subset of its processors, leading to the ***span law*** [20]:

$$T_P \geq T_\infty. \tag{3}$$

It is also useful to use the metrics of 'cost' and 'efficiency' when analysing parallel algorithms [21]. The ***cost*** of a parallel algorithm is minimised when all of processors are used at every step for useful computation and thus can be defined as $C_P = P \times T_P$. ***Efficiency*** is closely related to cost and describes speed-up per processor and can be defined as:

$$e_P = \frac{S_P}{P} = \frac{T_1}{C_P}. \tag{4}$$

Another helpful theorem for analysis is ***Brent's Theorem***, which states that for an algorithm that can run in parallel on N processors can be executed on $P < N$ processors in a time of approximately [22]:

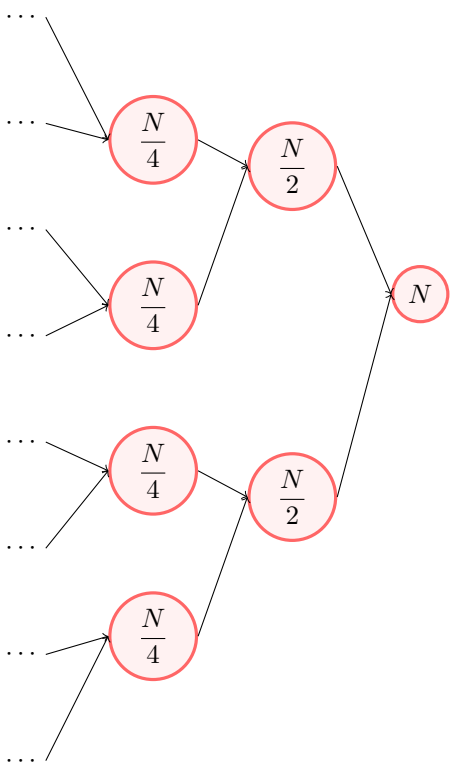

Figure 2: An example of calculating work for a merge-sort like algorithm. Summed nodes are in red.

$$T_P \leq T_N + \frac{T_1 - T_N}{P}. \tag{5}$$

This can be approximated with the upper bound of $O(\frac{T_1}{P} + T_N)$ [21].

Determining the span, work, parallelism, efficiency and cost, and examining the application of Brent's theorem to the computations at hand will allow for the computational complexity of the SPIIR pipeline to be analyse.

3 Design process

This section aims to layout and discuss the design process used to arrive at the final design and implementation to easily support any number of detectors. Section 3.1 will discuss the various design constraints that should be factored in any design implementation, and section 3.2 will discuss the tools used throughout the design process. The code sections that will be impacted by the implementation will be explored in section 3.3, and a framework for evaluating the success of the implementation will be discussed in section 3.4.

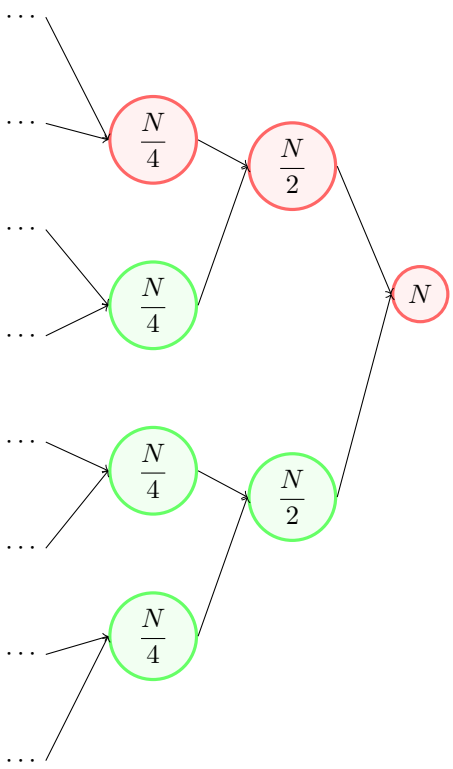

Figure 3: An example of calculating span for a merge-sort like algorithm. Summed nodes are in red.

3.1 Design Constraints

In the process of designing any system or any modifications thereof, constraints on that design must be considered and factored in. This section aims to lay out and discuss the various design constraints that should be imposed on any final design.

The below constraints are ordered in importance.

- **Output results must be unchanged**
  A major measure of a gravitational-wave detection pipeline is its accuracy. Any design should not impact or modify the output results of the existing pipeline on the same data.

- **The output table format should be unchanged**
  Events that are detected by the SPIIR pipeline are uploaded to the gravitational-wave candidate event database (GraceDB) using a unified table format to represent event data [23]. It is of critical importance that the SPIIR pipeline is still able to interact with GraceDB so that it can continue to have detected events considered by the wider gravitational-wave community. In addition, the pipeline outputs a number of files that are automatically scanned to ensure that the pipeline is functioning correctly. The format of these output files should not change.

- **Adding new detectors should be able to be done by a non-technical individual**
  The wider gravitational-wave detection community is largely composed of physicists as opposed to software engineers. As such, any changes to the number of detectors are likely to be best tested by

non-technical members of the gravitational-wave community, as the expertise to ensure that the values output by the pipeline are correct are more likely to come from physicists. Any design for supporting any number of detectors should ensure that the process of adding detectors should be easy enough to be completed and tested by a non-technical individual.

- **The external interface of the pipeline should not change**
  There are many developers and scientists that are using the existing SPIIR pipeline and use customised scripts that automate gravitational wave searches. The ability to support any number of detectors should be a largely internal change, without needing to modify the way that other users interact with the pipeline and force modifications to existing search scripts.

- **Individual detectors should still be exposed in Python code**
  There are several reasons for this constraint to be factored into any implementation. First, exposing individual detectors in the Python code allows for easier programming debugging of any values that may be incorrect, as the values of each detector would be immediately evident by the variable names instead of having be indirectly referenced by knowledge of the data structure used to hold the group of detectors. In addition, the Python code that interacts with GraceDB iterates through the list of detectors and uses the individual detector names to collect the data required for each event. By exposing individual detectors to the Python code, none of the code that interacts with GraceDB would need to change.

3.2 Employed tools

There are a number of tools that were employed in the process of designing and testing the implementation of supporting any number of detectors. These were:

- **OzStar [24]**
  "OzStar" is a computing cluster hosted at the University of Swinburne for use in gravitational-wave discovery and theoretical astrophysics. Much of the SPIIR development process occurs on OzStar, and its head nodes were used for both building and testing the modifications made to the SPIIR pipeline.

- **LIGO DataGrid (LDG) [25]**
  The LDG is the collection of clusters that the LSC uses for running the various gravitational-wave detection pipelines on live data. Testing and benchmarking (see section **??**) were performed on the LDG as the available nodes there closely mirror those that the SPIIR pipeline will run on.

- **SPIIR scripts [26]**
  "SPIIR scripts" is a collection of scripts maintained by Patrick Clearwater for use in the SPIIR development process. The `build_spiir` script from this collection was used to build the modifications made to the pipeline.

- **GWDC utils [27]**
  "GWDC utils" is a collection of utilities maintained by Alex Codoreanu that are written to perform various tasks for the SPIIR pipeline. The unit tests that are exposed as a part of GWDC utils were

used for testing that the modifications made to the pipeline did not affect its output (see section 3.1).

- **OzGrav Research utils [28, `utils/`]**
  "OzGrav Research utils" is a collection of utilities created by Thomas Almeida to automate the process of creating callgraphs from C or CUDA code on the OzStar cluster. These were used in the generation of callgraphs which were used for the complexity analysis in section 5.2.

3.3 Relevant code

The relevant code sections that may need to be modified to support any number of detectors is any step after the individual detector SPIIR filtering is combined, as observed in figure 1. This includes:

- Coherent post-processing with sky localization
  [6, `gstlal-spiir/gst/cuda/postcoh/`]
- $\mathcal{R}(\rho_c, \xi_c^2)$ from previous background events
  [6, `gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c`]
- False Alarm Rate (FAR) assignment to foreground events
  [6, `gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c`]
- Trigger and skymap to GW trigger database
  [6, `gstlal-spiir/python/pipemodules/postcoh_finalsink.py`]

In addition, the data structures that are passed between the components also need to be modified. This includes the `PostcohInspiralTable` [6, `gstlal-spiir/python/pipemodules/postcohtable/`] and the `PeakList` [6, `gstlal-spiir/gst/cuda/postcoh/postcoh.h`].

These files are mixed between several different languages; C, CUDA (see section 2.2), and Python. As such, any implementation of support for any number of detectors must consider the ability to:

- **Move data structures easily between GPU and CPU memory**
  CUDA provides two interfaces for allocating and managing memory across a GPU and CPU; managed memory and memory copies. Managed memory automatically propagates changes to allocated memory in a GPU or CPU to the other address space, whilst memory copies copy a block of memory from one address space to another.

  The existing SPIIR pipeline uses memory copies for transferring data to and from the GPU address space, and as such the implementation of supporting any number of detectors should ensure that the existing memory copies will still function as expected. This means that flat data structures would be generally preferred, as they can be copied with a single memory copy.

- **Interoperability between Python and C code**
  The `PostcohInspiralTable` is a C data structure that exposes a Python interface and the final trigger generation and uploading to GraceDB takes the data structure as an input. As such, it is vital that any changes to the `PostcohInspiralTable` are able to be used in the Python interface that it exposes.

3.4 Evaluation criteria

The success of a design and implementation needs to be evaluated according to some predefined criteria. The final design will be evaluated according to the following criteria, with results displayed in section 4.3.

1. Output results must be unchanged
2. The output table format should be unchanged
3. Adding new detectors should be able to be done by a non-technical individual
   (a) Maximum number of required edited files: 3
4. Unchanged external interface
5. Exposing of individual detectors in Python
6. Changing number of detectors compiles correctly

4 Final Design

This section aims to describe and expand on the final design and implementation chosen for allowing the SPIIR pipeline to work with any number of detectors.

Section 4.1 discusses the major components and changes that each source code patch makes to the codebase, and the testing that the patches went through will be described in section 4.2. The final design and implementation will be evaluated in section 3.4, according to the criteria laid out in section 4.3.

4.1 Patches

The patches referred to in this section can be found in appendix A.

4.1.1 Mapping combinations of detectors to integers

The patch for this section can be found in appendix A.1.

Internally, the pipeline uses a static constant array defined in `gstlal-spiir/include/pipe_macro.h` called the "`IFOComboMap`" to keep track of which detectors it is processing. The original ordering of the array is as follows:

- All single detectors
- All combinations of two detectors
- All combinations of three detectors

This presents a problem when adding detectors, as the indexing of the array would change such that; all combinations of two detectors are shifted by at least 1; and, all combinations of three detectors are shifted by at least $\binom{n}{2} - \binom{n-1}{2}$, where n is the number of detectors; and so on. If the number of detectors changes, then all of the locations that index the `IFOComboMap` will also need to change,

to ensure that they continue to index the right combinations. For example, the `scan_trigger_ifos` function in `gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c` determines which detectors should be active by checking the indexes of the `IFOComboMap`, and if the number of detectors changes, all of the indexing done in the function will also have to change.

The `IFOComboMap` is, as the name suggests, a list of all of the combinations of detectors, not including combinations of zero detectors. As a special case of the binomial expansion; $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$ when $x = y = 1$, we can note that [29];

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n. \tag{6}$$

As per equation 6, the sum of the number of combinations of n elements is equal to 2^n. This lends itself well to binary numbers, as an 8-bit binary number can represent 2^8 numbers or all the combinations of 8 elements — which means that we can represent all combinations of N detectors with N bits.

By having each detector represented as a single bit in a N-bit number, there are a number of advantages across the codebase that are evident as well. `popcount` (or `POPCNT`) is an instruction for x86 processors and NVIDIA GPUs that returns the number of set (1) bits in an integer, and thus the number of detectors being used can be determined with a single instruction. In addition, whether a detector is being used can be determined using a bitwise `AND` of the combination and 2^i, where i is the index of the detector. These both simplify large areas of the codebase, and make up the majority of the patch in appendix A.1.

By implementing this change, adding a new detector does not modify the existing indexing into the `IFOComboMap`, significantly reducing the number of files and code locations that would need to be modified to support a new detector.

4.1.2 Replacing individual detector variables with C arrays

The patch for this section can be found in appendix A.2.

As mentioned in section 3.3, there are two data structures that are passed between the components of the pipeline that can be seen in figure 1 — the `PostcohInspiralTable` and the `PeakList`. These data structures include detector specific variables that are cloned for each detector. For example, both structures have `snglsnr_H` data members to refer to the signal-to-noise ratio from the Hanford LIGO detector.

Within the `PostcohInspiralTable` and `PeakList` themselves, these can simply be moved from detector specific variables to being arrays with a length equal to the number of detectors, however this change causes some issues with the way the C code interacts with both CUDA and Python.

Arrays in C are simply pointers to the first element of a range of contiguous memory [30], however the CUDA programming model assumes by default that both the host and the GPU maintain their own separate memory spaces as well as separate allocation and deallocation routines [19]. This means that any multi-dimensional arrays (such as the afore mentioned transformation to `snglsnr_H`, which was previously a pointer to a float in the `PeakList` structure), require a additional memory copies per element per dimension, which is both unergonomic and difficult to implement correctly due to each dimension needing to be synchronised before the next dimension can be copied. Since CUDA 6.0, CUDA also provides the `cudaMallocManaged` function, which is accessible from the CPUs and GPUs a system as

a common address space. By using `cudaMallocManaged`, the complexity of handling multi-dimensional arrays can be largely elided and passed to the CUDA runtime without performance impairment.

There are also several challenges that using C arrays to hold detector specific variables presents when trying to provide interoperability with Python. First, Python does not have a native contiguous-memory array data type, and instead usually uses linked lists as its array-like data structure. This can be rectified by using the array API of NumPy, an open source project for numerical computing with Python, which allows for C arrays to be used within a Pythonic API [31]. Second, as per section 3.4 the individual detector variables should be exposed in Python, despite them no longer existing as singular variables. This can be resolved by using Python's attribute getters and setters, which allow for customized functions to be run for getting and setting class attributes [32]. When used on a data type, an array of getters and setters is used [33], and the modification of this array at runtime allows for the programmatic creation of the individual detector variables.

After implementing this change, the only modifications required to add support for new detectors are to the `IFOComboMap` mentioned in section 4.1.1, and to the list of data sources to ensure that the data from the new detector can be read.

4.2 Testing

As per section 3.2, the `build_spiir` script from "SPIIR scripts" was used for building the changes to the code, and the unit tests from "GWDC utils" were used to ensure that the output of the pipeline was not affected by the code modifications.

The unit test in "GWDC utils" runs a comparison on a known good output and the signal-to-noise ratio series for each detector in a new build. Both patches were built, run and passed the "GWDC utils" unit test, ensuring that there were no regressions in output. In addition, the outputs of existing combinations of detectors were manually checked against a build of the SPIIR pipeline without the patches to ensure that fields other than the signal-to-noise ratio series were consistent across pipeline versions.

The ease of supporting additional detectors was tested by doing the necessary modifications to support one additional detector, ensuring that the build process worked correctly, and then running the same unit tests as above to ensure no regressions.

All testing of the unit tests and of additional detector support were done successfully with no faults observed on the OzStar cluster.

4.3 Evaluation

A summary of this section can be found in table 1.

1. **Output results must be unchanged**
   As per section 4.2, unit testing and manual comparison of the pipeline showed that the output results were unchanged.

2. **The output table format should be unchanged**
   Whilst the table that is sent to GraceDB was unchanged, the files output by the pipeline had their

Criteria	Success
Output results must be unchanged	✔
The output table format should be unchanged	✗
Adding new detectors should be able to be done by a non-technical individual	✔
Maximum number of required edited files	2
Unchanged external interface	✔
Exposing of individual detectors in Python	✔
Changing number of detectors compiles correctly	✔

Table 1: An evaluation of the implementation of supporting any number of detectors against the criteria specified in section 3.4

header names modified as the full name of the detector was appended to detector specific variables instead of the one representative letter.

3. **Adding new detectors should be able to be done by a non-technical individual**
   This is subjective, but arguably achieved.

   (a) **Maximum number of required edited files: 3**
   The number of files that need to be edited to add new detectors is 2, `gstlal-spiir/include/pipe_macro.h` for the `IFOComboMap` and `gstlal/python/datasource.py` for getting detector data.

4. **Unchanged external interface**
   The API for the SPIIR pipeline was not modified, with the only visible changes being internal to the pipeline.

5. **Exposing of individual detectors in Python**
   As per section 4.1.2, this was done using Python's attribute getters and setters.

6. **Changing number of detectors compiles correctly**
   As per section 4.2, the modified pipeline compiles and runs correctly with an additional detector added.

As per the criteria laid out in section 3.4, the implementation to easily support any number of detectors has been successful, however there are still areas for improvement. These areas will be outlined in section 6.

5 Discussion

This section aims to discuss the wider implications of the design chosen in section 4 for both the wider gravitational-wave detection pipeline community and the SPIIR pipeline specifically.

Section 5.1 will compare the new SPIIR pipeline to the previous state of the SPIIR pipeline and other gravitational-wave detection pipelines, and section 5.2 will consider the implications of supporting any number of detectors on the runtime of the SPIIR pipeline by performing a complexity analysis.

5.1 Comparison to previous "state of the art"

5.1.1 SPIIR pipeline

It is difficult to compare the difference between adding a detector before these changes and after, as the pipeline was originally created with support for all three currently supported detectors so there has never been a previous requirement to include additional detectors. An estimate of the magnitude of work required for supporting additional detectors can, however, be done using the changes made in section 4.1, as there would be significant overlap between the files that would have to change to support additional detectors and the files modified in those patches. Thus, we can estimate that the number of files that would have to change would be 16 files, with a total number of lines changed at approximately 1000.

In comparison, as stated in section 4.3, the number of files that would need to change to support an additional detector is only 2, representing a large leap forward in ease of modification.

5.1.2 Other pipelines

PyCBC

In comparison to PyCBC, the process of adding support for new detectors is still more difficult — it's hard to beat "just update the library" for ease of use! This work, however, does continue to allow the SPIIR pipeline to continue supporting the use of any subset of supported detectors for gravitational-wave searching, representing a significant continuing advantage that the SPIIR pipeline has over PyCBC.

GstLAL

The implementation to easily support any number of detectors in SPIIR makes significant improvements of the existing support in GstLAL. The most recent detector addition (GEO600, June 2020), involved modifying 9 files with 187 new lines of code — and the detector addition didn't include being able to get the data from the detector, just support for the coincidence search [17]. In comparison, the equivalent change in the SPIIR pipeline would modify only one file (as getting data from the detector isn't required) and would involve adding approximately 8 lines of code, a significant improvement.

5.2 A complexity analysis of the parallel post-processing of the SPIIR pipeline

As discussed in 2.3, the SPIIR pipeline is designed to be as low latency as possible, and an asymptotic complexity analysis of components that may be impacted by the addition of new detectors allows for the measurements of the potential cost of doing so. This subsection aims to determine the complexity analysis of the coherent post-processing step of the SPIIR pipeline to understand the impact of increasing the number of detectors.

Coherent post-processing was introduced in 2017 by Qi Chu et al as an alternative to the coincidence post-processing used by all other pipelines (see section 2.1) [9]. In the original work, the computational cost of the coherent search is estimated to be $O(2N_d^3 N_m N_p)$, where N_d is the number of detectors, N_m is the number of sets of IIR filters (called templates), and N_p is the number of potential sky locations [9]. Further optimisations were made to the pipeline in 2018, including moving to using GPU acceleration, and whilst a paper outlining the changes made discusses a number of constant time optimisations made

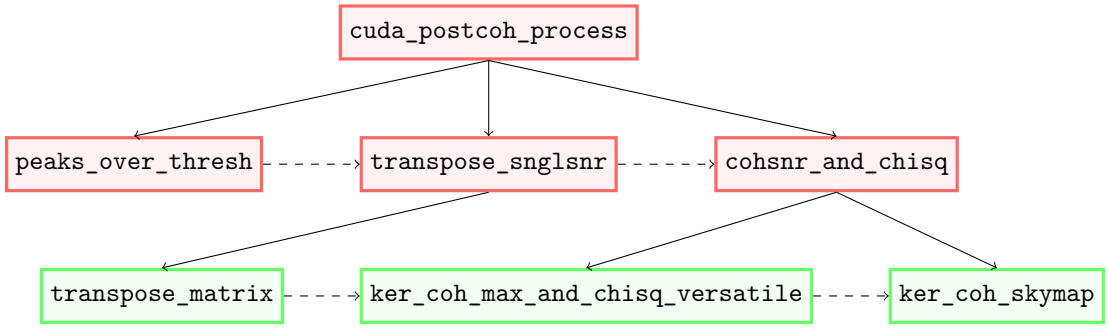

Figure 4: Simplified callgraph for coherent post-processing in the SPIIR pipeline.
Function calls are represented by solid arrows, the ordering of the calls is represented by dashed arrows.
GPU functions are in green boxes, and CPU functions are in red boxes.

to the pipeline, the computational cost of the overall process is not discussed, despite the parallelisation of GPU acceleration leading to additional changes to the overall potential cost [8].

"OzGrav Research utils" (see section 3.2) were used to generate a callgraph to determine the data flow for this analysis [see 28, `resources/callgraph.png`]. A simplified version of the callgraph can be seen in figure 4.

5.2.1 Maximum element reduction

One of the more common operations in the SPIIR pipeline is the concept of a "maximum element reduction". Reduction is the idea of taking some array of data and producing a single summary output from that array, whether it is the total sum of the array or the maximum value of the array and its index in the array as it is in this case.

Harris discusses the computational complexity of reduction algorithms in a parallelised context, noting that the best complexity according to Brent's Law is $O(\frac{N}{\log N})$ threads each doing $O(\log N)$ sequential work, resulting in a total overall cost of $O(\frac{N}{\log N} \times \log N) = O(N)$ [34].

We can note from our own analysis, that the process of reduction can be parallelised by the use of a binary tree of operations, where each vertex in the binary tree combines the results of the two parent vertices. In the case of determining the maximum of two numbers, each vertex is identical in the amount of work done, and thus we can determine each vertex to be a unit of work. As there are N elements in the original array, we can note that the height of the binary tree is $\log N$, and each level of the binary tree has $N_l/2$ vertices, thus the total number of vertices in the binary tree is $\sum_{i=0}^{\log N} 2 \times i = N$. Using this information, we can determine that the ***work*** of a parallelised reduction is $T_1 = O(N \times 1) = O(N)$, and that the ***span*** of the reduction is $T_\infty = O(\log N \times 1) = O(\log N)$. Thus, the ***parallelism*** of the reduction is:

$$p = \frac{N}{\log N}.$$

Using the span and work laws, we can observe that any algorithm using the above method is bounded by the inequalities $O(\log N) \leq T_P, \frac{O(N)}{P} \leq T_P$. This means that best possible time complexity with P processors is $O(\log N)$ (equation 3). We can determine the minimum number of processors required to

achieve this runtime using the formula $T_P = O(\log N) = \frac{O(N)}{P}$, which can be rearranged to

$$P = \frac{N}{\log N},$$

thus the time complexity cannot improve past $P = N/\log N$ processors. We can also observe that using $P = N/\log N$ processors gives a ***cost*** of $C_P = N$, which is identical to the sequential algorithm.

Functions that include maximum element reduction will be denoted for clarity with $M(x)$, where x is the size of the array being reduced.

5.2.2 Determining the number of samples over a signal-to-noise threshold

The coherent post-processing in SPIIR determines the number of samples over a signal-to-noise (SNR) threshold in order to not do more work than is required. The function that is used for determining the number of samples over the threshold (`peaks_over_thresh`) is a sequential algorithm that runs on the CPU, and shall be analysed as such, although there is an alternative GPU-based implementation that is not used.

Initially, the function performs a maximum element reduction to get the maximum SNR from the combined IIR filters (templates) for each sample. Recalling from section 5.2.1 that for maximum element reduction $T_1 = O(N)$, and that this operation is performed S times, where S is the number of samples, we can determine that this initial reduction has a time complexity of $O(ST)$, where T is the number of templates.

The function then determines the maximum SNR across the templates found from the previous step by stepping through every combination of samples and removing SNR samples that are using the same template and have a lower SNR, resulting in a step with a time complexity of $O(S^2)$.

The function then determines the maximum overall SNR for the input samples ($O(S)$) and cycles through every maximum SNR to cluster maxima that are close together to be a single combined maximum. The number of maxima is bounded by ($O(\min\{S, T\})$) as there cannot be more maxima than there are samples or templates.

This gives the overall function a time complexity of $O(ST + S^2 + S + \min\{S, T\})$, which can be reduced to the dominating terms of:

$$O(ST + S^2).$$

5.2.3 Transposing the input matrices

The full post-processing function requires that the input matrix is transposed for better memory access such that each row is a different template, and each column is a different sample. To transpose the matrix, the GPU function `transpose_matrix` is used, thus this should be analysed as a parallel algorithm.

The algorithm in use works by breaking the original array into tiles of size 32×32, and then inserting the transpose of the tile into an output array. The tiles are further broken down eight processors per row, so each thread does four copies. We can conceptualise this as a DAG by observing that each tile does

not depend on any other tile to be completed, and that each tile is composed of 32×8 interdependent processors, each doing 4 units of work.

Using this observation, we can see that the ***span*** of the algorithm is $T_\infty = (32 \times 8) \times 4 = O(1024) = O(1)$, and the ***work*** is $T_1 = O(ST)$, where S is the number of samples and T is the number of templates. Thus, the ***parallelism*** of the transpose is $p = ST$.

Using the span and work laws (equations 3 and 2), we can observe that the above method is bounded by the inequalities $O(1) \leq T_P, \frac{O(ST)}{P} \leq T_P$. Thus it can be determined that the best possible time complexity with P processors is bounded by ratio of available processors to the size of the transposed matrix (the work law). This gives the function an overall time complexity of:

$$O(\frac{ST}{P}).$$

5.2.4 Determining the coherent correlation and statistical value of data points

The scoring metric of different templates and times is determined using coherent correlation and determining their statistical value using a chi squared-based distribution. These scoring metrics are performed using the GPU function `ker_coh_max_and_chisq_versatile`, and thus should be analysed as a parallelised function.

In this function, each block looks at a different SNR maximum (as discussed in section 5.2.2) and splits the threads within the blocks for operations on that peak.

Determining the sky direction of the SNR maximum

Initially, each thread within a block looks at a different sky direction and determines the total signal-to-noise ratio (SNR) by summing the SNR of each of the detectors at that given sky direction with the relevant detector arrival time offsets. The time complexity for the calculation of SNR for a given time offset is $O(D + D^2)$, where D is the number of detectors. The maximum SNR for all the sky directions is then spread across each warp and placed into shared memory before being shared across every thread in the block, which is an application of the parallelised maximum element reduction function discussed in section 5.2.1.

Thus, the ***span*** of determining the sky direction with the highest signal to noise ratio is $T_\infty = O(D + D^2 + M_{T_\infty}(S))$ and the ***work*** is $T_1 = O(S(D + D^2) + M_{T_1}(S)$, where S is the number of sky directions and $M(x)$ is the complexity of the parallelised maximum element reduction function. We can further state that the ***parallelism*** of this is equivalent to the number of sky directions, $S + S/\log S$.

Calculating signal consistency statistics

After having determined the sky direction with the highest SNR for a given maximum, the function then calculates a signal-morphology based statistic ξ_D^2 for each detector D. The statistic is a reduced χ^2 distribution with $D \times 2 - 4$ degrees of freedom and a mean value of 1, and is given in the discrete form by:

$$\xi_D^2 = \frac{\sum_{j=-m}^{m} |\varrho_D[j] - \varrho_D[0] A_D[j]|^2}{\sum_{j=-m}^{m} (2 - 2|A_D[j]|^2)}, \tag{7}$$

where ϱ is the coherent SNR, A_D is the vector of the correlation of the given template with the output from the detector and $2 \times m$ is the number of samples.

The numerator of the statistic is calculated by splitting the number of samples between the threads of a block, followed by combining the results of the statistic across each warp and then each block. The combination of the statistic across each warp and block is a modification of the parallelised maximum element reduction discussed in section 5.2.1 that uses addition instead of maximum as the combining binary function. Thus the ***span*** of calculating the statistic is $T_\infty = O(D \times M_{T_\infty}(N))$ and its ***work*** is $T_1 = O(D \times M_{T_1}(N))$, where N is the number of samples. We can then state that the ***parallelism*** of calculating the statistic is equivalent to the parallelism of the reduction, $O(N/\log N)$.

Generating time-shifted background noise statistics

The function then performs a number of time shifts on background noise for use with the significance estimation. The generation of a single background statistical variant is equal to the total work of the function so far, save that instead of using blocks for every peak, each warp looks at a different time shift. Thus, whilst the theoretical time complexity does not change, the number of processors available is smaller, so the actual runtime each loop is approximately the warp size slower.

Overall computational cost

Overall, this function has a ***span*** of $T_\infty = 2(D + D^2 + M_{T_\infty}(S) + DM_{T_\infty}(N))$, and has $T_1 = P(S(D + D^2) + M_{T_1}(S) + DM_{T_1}(N) + B(S(D + D^2) + M_{T_1}(S)))$ ***work***, where P is the number of SNR maxima and B is the number of times shifts made to background noise.

5.2.5 Calculating heat skymaps

If the coherent SNR exceeds a threshold, the post-processing produces a skymap of the highest SNR in the GPU function `ker_coh_skymap`.

The function determines the highest maximum SNR by using the maximum element reduction technique discussed in section 5.2.1. Following this, the function re-performs the process discussed in section 5.2.4 with additional sky directions and without the reduction to generate the final skymap.

As such, this function has a ***span*** of $T_\infty = M_{T_\infty}(P) + D + D^2$ and total ***work*** of $T_1 = M_{T_1}(P) + S(D + D^2)$.

5.2.6 Overall complexity

The total span and work of the coherent post-processing step in the SPIIR pipeline is the sum of the total spans and works of the internal functions. Conversely, we cannot determine the overall parallelism as the post-processing step spans a number individual functions that can each be run with a different set of processors. As the step to determine the number of peaks over a threshold (see section 5.2.2) is sequential, we can consider its time complexity as contributing to both the span and work of the total

pipeline. Another thing to note is that the steps for determining the coherent correlation, statistic value and skymaps (sections 5.2.4 and 5.2.4) will be run for every detector.

With this in mind, we can determine that the ***span*** of the post-processing is:

$$\begin{aligned} T_\infty &= O(NT + N^2 + 1 + D(2(D + D^2 + \log S + D \log N) + \log P + D + D^2)) \\ &= O(NT + N^2 + D^3 + D^2 \log N + D \log S + D \log P), \end{aligned} \tag{8}$$

where D is the number of detectors, S is the number of sky directions, T is the number of templates, N is the number of samples and $P = \max\{S, T\}$.

The total ***work*** of the post-processing is:

$$\begin{aligned} T_1 &= O(NT + NT + S^2 + D(P + S(D + D^2) + P(S(D + D^2) + S + DN + B(S(D + D^2) + S)))) \\ &= O(NT + N^2 + SPD^3 + SPBD^3 + ND^2), \end{aligned} \tag{9}$$

where D is the number of detectors, S is the number of sky directions, T is the number of templates, N is the number of samples, B is the number of is the number of times shifts made to background noise and $P = \max\{S, T\}$

5.2.7 Implications

It can be noted from equation 9 that with a single processor, the expected runtime of the SPIIR pipeline's coherent search scales with the number of detectors cubically, and most of the other terms also scale with the number of detectors. As equation 8 shows that even with the use of an infinite number of processors there are still terms that scale cubically with the number of detectors, it can be assumed that the real runtime of the coherent post-processing will increase in a cubic manner relative to the number of detectors.

As such, it would be the recommendation of the author that work is done to mitigate the increased latency that would be introduced by the addition of new detectors.

6 Future work

The change to C arrays from individual detector variables in section 4.1.2 uses fixed-size arrays, which means that the memory usage of the pipeline will be invariant to the number of detectors being used, instead it will depend on the number of detectors supported. This may present issues with a large number of detectors as memory, and GPU memory in particular, is generally at a premium and should be conserved if possible. A potential future research direction in the realm of memory optimisation would be to modify the algorithms and data structures to size the arrays to the number of detectors being used.

One method that could be used to counter the potential runtime cost of adding new detectors — as discussed in section 5.2.7 — is to have some detectors skip the coherent search step. The detectors skipping the coherent search step could be reintroduced later in the pipeline to help improve the accuracy of the generated skymaps whilst not causing a noticeable impairment of performance. Research into a potential restructuring to support this is already under way, with a working prototype expected to be

completed within the coming months.

Another potential region for research would be to investigate alternative parallelisation techniques and algorithms in the SPIIR pipeline's coherent post-processing. If the cubic term about the number of detectors can be reduced to even a squared term, the runtime cost of adding new detectors would be significantly reduced.

7 Conclusion

This dissertation addresses the issue of a large development cost to add support for new gravitational-wave detectors to gravitational-wave detection pipelines as the number of detectors continues to grow. The development cost of adding support for just a single detector can be as high as requiring 1000 lines of code changed across 16 files.

By implementing both the powers of two combination map and by removing individual detector variables by replacing them with C arrays, the development cost of supporting additional detectors has been reduced to be able to be completed by even non-technical individuals, with only two files requiring modifications. This represents a large leap forward in the accessibility of adding and testing new detectors in gravitational-wave detection pipelines for the wider scientific community.

References

[1] *Introduction to LIGO & Gravitational Waves*. URL: https://www.ligo.org/science/GW-GW2.php.

[2] *LIGO Lab: Caltech: MIT*. URL: https://www.ligo.caltech.edu/.

[3] B. P. Abbott et al. "Observation of Gravitational Waves from a Binary Black Hole Merger". In: *Phys. Rev. Lett.* 116 (6 2016), p. 061102. DOI: 10.1103/PhysRevLett.116.061102. URL: https://link.aps.org/doi/10.1103/PhysRevLett.116.061102.

[4] *LIGO Detected Gravitational Waves from Black Holes*. URL: https://www.ligo.caltech.edu/detection.

[5] *LIGO Scientific Collaboration*. URL: https://ligo.org/.

[6] *Summed Parallel Infinite Impulse Response Pipeline Codebase*. URL: https://git.ligo.org/lscsoft/spiir/.

[7] Shaun Hooper et al. "Summed parallel infinite impulse response filters for low-latency detection of chirping gravitational waves". eng. In: *Physical Review D - Particles, Fields, Gravitation and Cosmology* 86.2 (2012). ISSN: 1550-7998.

[8] Xiaoyang Guo et al. "GPU-Optimised Low-Latency Online Search for Gravitational Waves from Binary Coalescences". eng. In: vol. 2018-. EURASIP, 2018, pp. 2638–2642. ISBN: 9082797011. URL: https://ieeexplore.ieee.org/document/8553574.

[9] Q. Chu. *Low-latency detection and localization of gravitational waves from compact binary coalescences*. eng. 2017.

[10] *LSC Algorithm Library for GStreamer*. URL: https://git.ligo.org/lscsoft/gstlal/.

[11] *PyCBC - Analyze gravitational-wave data, find signals, and study their parameters.* URL: https://pycbc.org/.

[12] Alex Nitz et al. "gwastro/pycbc: PyCBC release v1.16.11". In: (2020). DOI: 10.5281/zenodo.4075326.

[13] LIGO Scientific Collaboration. *LIGO Algorithm Library - LALSuite.* free software (GPL). 2018. DOI: 10.7935/GT1W-FZ16.

[14] *GstLAL documentation.* URL: https://lscsoft.docs.ligo.org/gstlal/.

[15] Cody Messick et al. "Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data". In: *Physical Review D* 95.4 (2017). ISSN: 2470-0029. DOI: 10.1103/physrevd.95.042001. URL: http://dx.doi.org/10.1103/PhysRevD.95.042001.

[16] *gstlal-inspiral/gst/lal/gstlal_itacac.c.* URL: https://git.ligo.org/lscsoft/gstlal/-/blob/master/gstlal-inspiral/gst/lal/gstlal_itacac.c.

[17] Cody Messick. *Added GEO600 support ("G1") to itacac (d43bcfd6).* URL: https://git.ligo.org/lscsoft/gstlal/-/commit/d43bcfd6096ac4fab33114848b2d5f9ffaf6ca86.

[18] *CUDA Toolkit.* URL: https://developer.nvidia.com/cuda-toolkit.

[19] *CUDA Programming Guide.* URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#hardware-multithreading.

[20] Thomas H. Cormen et al. *Introduction to algorithms.* eng. 3rd ed. MIT electrical engineering and computer science series. Cambridge, Mass: MIT Press, pp. 779–781. ISBN: 0070131430.

[21] Henri Casanova, Arnaud Legrand, and Yves Robert. *Parallel Algorithms.* eng. CRC Press, 2008, pp. 10–12. DOI: 10.1.1.466.8142.

[22] John L. Gustafson. "Brent's Theorem". In: *Encyclopedia of Parallel Computing.* Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 182–185. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_80. URL: https://doi.org/10.1007/978-0-387-09766-4_80.

[23] *GraceDB | The Gravitational-Wave Candidate Event Database.* URL: https://gracedb.ligo.org/.

[24] *OzStar – Supercomputing at Swinburne University of Technology.* URL: https://supercomputing.swin.edu.au/.

[25] *LIGO Data Grid.* URL: https://computing.docs.ligo.org/lscdatagridweb/.

[26] Patrick Clearwater. *Build SPIIR | SPIIR scripts.* URL: https://git.ligo.org/patrick.clearwater/spiir-scripts/.

[27] Alex Codoreanu. *GWDC Utils.* URL: https://git.ligo.org/alex.codoreanu/gwdc_utils/.

[28] Thomas Hill Almeida. "Tommoa/ozgrav-research". In: (2020). DOI: 10.5281/zenodo.4075219.

[29] Thomas H. Cormen et al. *Introduction to algorithms.* eng. 3rd ed. MIT electrical engineering and computer science series. Cambridge, Mass: MIT Press, pp. 1185–1186. ISBN: 0070131430.

[30] Bjarne Stroustrup. *The C++ Programming Language.* 4th ed. Addison-Wesley, 2013, pp. 179–182. ISBN: 0321563840.

[31] *Array API - NumPy v1.19 Manual*. URL: https://numpy.org/doc/stable/reference/c-api/array.html.

[32] *Common Object Structures - Python 2.7.18 documentation*. URL: https://docs.python.org/2.7/c-api/structures.html?highlight=pygetsetdef#c.PyGetSetDef.

[33] *Type Objects - Python 2.7.18 documentation*. URL: https://docs.python.org/2.7/c-api/typeobj.html#c.PyTypeObject.tp_getset.

[34] Mark Harris. *Optimizing Parallel Reduction in CUDA*. eng. URL: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf.

A Patches

A.1 Making `IFOComboMap` be sums of powers of two

```
From b4f563eb358b77f2e130aad6655db566a2f25b5a Mon Sep 17 00:00:00 2001
From: Tom Almeida <tommoa256@gmail.com>
Date: Thu, 20 Aug 2020 12:37:16 +0800
Subject: [PATCH 1/2] pipe_macro: Make IFOComboMap index with bits indicating
 which IFOs are being used

Previously IFOComboMap was indexed as [single detector, two detectors,
three detectors, ...]. This means that checking to see how many and
which detectors are in use can't be easily done without referring back
to the table, and any changes to the table can potentially break other
parts of the code.

This patch changes IFOComboMap to be indexed by powers of two. That
means that any index of (2^i - 1) is a single detector, indicated by
index into IFOMap. Now when using IFOComboMap, the number of detectors
in use can be determined using `__builtin_popcount(icombo + 1)`, and
whether a single detector is in use can be done with the check `index &
(icombo + 1)`.
---
 .../gst/cuda/cohfar/background_stats_utils.c  | 50 +++++++++++++++++++
 .../gst/cuda/cohfar/cohfar_accumbackground.c  |  8 ++-
 gstlal-spiir/gst/cuda/postcoh/postcoh.c       | 12 +++--
 gstlal-spiir/include/pipe_macro.h             | 11 ++--
 4 files changed, 45 insertions(+), 36 deletions(-)

diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c b/gstlal-spiir/gst/cuda/cohfar/background_
index 4a5b7ec8..45c57ec6 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
@@ -39,38 +39,34 @@
 #define EPSILON         1e-6

 int scan_trigger_ifos(int icombo, PostcohInspiralTable *trigger) {
-    int cur_ifo = 0, one_ifo_size = sizeof(char) * IFO_LEN;
+    int nifo = 0, one_ifo_size = sizeof(char) * IFO_LEN;
     char final_ifos[MAX_ALLIFO_LEN];
     gboolean pass_test = TRUE;
-    if ((icombo == 6 || icombo == 3 || icombo == 4)) { // H1L1, H1V1 or H1L1V1
-        if (trigger->snglsnr_H > EPSILON) {
-            strncpy(final_ifos + IFO_LEN * cur_ifo, "H1", one_ifo_size);
-            cur_ifo++;
-        } else
-            pass_test = FALSE;
-    }
-
-    if ((icombo == 6 || icombo == 3 || icombo == 5)) { // H1L1, L1V1 or H1L1V1
-        if (trigger->snglsnr_L > EPSILON) {
-            strncpy(final_ifos + IFO_LEN * cur_ifo, "L1", one_ifo_size);
-            cur_ifo++;
-        } else
-            pass_test = FALSE;
-    }
```

```
-
-    if ((icombo == 6 || icombo == 4 || icombo == 5)) { // H1V1, L1V1 or H1L1V1
-        if (trigger->snglsnr_V > EPSILON) {
-            strncpy(final_ifos + IFO_LEN * cur_ifo, "V1", one_ifo_size);
-            cur_ifo++;
-        } else
-            pass_test = FALSE;
+    // [THA]: Because icombo is sum(1 << index) - 1, we should add one to it
+    // so that we don't need to add one in the loop.
+    ++icombo;
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        // [THA]: We can determine if the IFO at IFOMap[i] is in the icombo by
+        // checking if that power of two exists in the combo
+        if (icombo & (1 << i)) {
+            // [THA]: This is a check that the data from this IFO is actually
+            // valid. If it's not valid, the number will be very *very* small
+            if (*(&trigger->snglsnr_H + i) > EPSILON) {
+                strncpy(final_ifos + IFO_LEN * nifo, IFOMap[i].name,
+                        one_ifo_size);
+                nifo++;
+            } else {
+                pass_test = FALSE;
+            }
+        }
     }
     if (pass_test != TRUE) {
-        strncpy(trigger->ifos, final_ifos, cur_ifo * sizeof(char) * IFO_LEN);
-        trigger->ifos[IFO_LEN * cur_ifo] = '\0';
+        strncpy(trigger->ifos, final_ifos, nifo * one_ifo_size);
+        trigger->ifos[IFO_LEN * nifo] = '\0';
         return get_icombo(trigger->ifos);
-    } else
-        return icombo;
+    } else {
+        return icombo - 1;
+    }
 }
 int get_icombo(char *ifos) {
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_accun
index ef3136e0..154cbda1 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
@@ -223,6 +223,7 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
     int isingle, nifo;
     for (; intable < intable_end; intable++) {
         icombo = get_icombo(intable->ifos);
+        // The combination of IFOs is invalid
         if (icombo < 0) {
             LIGOTimeGPS ligo_time;
             XLALINT8NSToGPS(&ligo_time, GST_BUFFER_TIMESTAMP(inbuf));
@@ -245,7 +246,12 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
             outtable++;
         } else {
             /* increment livetime if participating nifo >= 2 */
-            if (icombo > 2) {
+            // If icombo is a power of two, then there is only one participating
+            // IFO, thus we can use the property of `(x & (x-1)) != 0` to
+            // determine if we have more than one IFO participating
+            //
+            // Note that this is inverted because icombo is sum(1 << index) - 1
+            if ((icombo + 1) & icombo) {
                 nifo = strlen(intable->ifos) / IFO_LEN;
                 /* add single detector stats */
                 get_write_ifo_mapping(IFOComboMap[icombo].name, nifo,
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh.c b/gstlal-spiir/gst/cuda/postcoh/postcoh.c
index f724477b..85f071ff 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh.c
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh.c
@@ -667,16 +667,20 @@ static gboolean cuda_postcoh_sink_setcaps(GstPad *pad, GstCaps *caps) {
         data        = gst_pad_get_element_private(pad);
         set_offset_per_nanosecond(data, postcoh->offset_per_nanosecond);
```

```
        set_channels(data, postcoh->channels);
-       // FIXME: need to consider non-standard ifo indexing, like HV, need
-       // testing
+       // [THA]: Non-standard IFO indexing (e.g. VH) works because `get_icombo`
+       // doesn't care about the ordering of IFOs
        strncpy(state->all_ifos + IFO_LEN * i, data->ifo_name,
                sizeof(char) * IFO_LEN);
    }
    state->all_ifos[IFO_LEN * nifo] = '\0';
-   state->ifo_combo_idx            = get_icombo(state->all_ifos);
+   // [THA]: This is the only place that ifo_combo_idx is used. Perhaps remove
+   // it later to save space?
+   state->ifo_combo_idx = get_icombo(state->all_ifos);
+   // [THA]: sizeof() only works for arrays that we've statically created, so
+   // we use strlen() to get the length of the combination name
    /* overwrite all_ifos to be the same with the combo in the IFOComboMap */
    strncpy(state->all_ifos, IFOComboMap[state->ifo_combo_idx].name,
-           sizeof(IFOComboMap[state->ifo_combo_idx].name));
+           strlen(IFOComboMap[state->ifo_combo_idx].name));
    state->all_ifos[IFO_LEN * nifo] = '\0';

    /* initialize input_ifo_mapping, snglsnr matrix, and peak_list */
diff --git a/gstlal-spiir/include/pipe_macro.h b/gstlal-spiir/include/pipe_macro.h
index a6f20486..b6e0fc9a 100644
--- a/gstlal-spiir/include/pipe_macro.h
+++ b/gstlal-spiir/include/pipe_macro.h
@@ -13,13 +13,16 @@ typedef struct _IFOType {
 } IFOType;

 static const IFOType IFOMap[MAX_NIFO] = {
-    { "H1", 0 },
-    { "L1", 1 },
-    { "V1", 2 },
+    { "H1", 0 }, // 1 << 0 = 1
+    { "L1", 1 }, // 1 << 1 = 2
+    { "V1", 2 }, // 1 << 2 = 4
 };
 #define MAX_IFO_COMBOS 7 // 2^3-1
+// A combination is sum(1 << index) - 1
+// This gives us some nice mathematical properties that we can use to check
+// if an IFO exists in a given ComboMap
 static const IFOType IFOComboMap[MAX_IFO_COMBOS] = {
-    { "H1", 0 },   { "L1", 1 },   { "V1", 2 },     { "H1L1", 3 },
+    { "H1", 0 },   { "L1", 1 },   { "H1L1", 2 },   { "V1", 3 },
     { "H1V1", 4 }, { "L1V1", 5 }, { "H1L1V1", 6 },
 };
 /* function given a random ifo, output the index in the IFOComboMap list,
--
GitLab

From 6f7f2e00c9630a4a615d6bec63953342f93d023f Mon Sep 17 00:00:00 2001
From: Tom Almeida <tommoa256@gmail.com>
Date: Wed, 26 Aug 2020 13:48:03 +0800
Subject: [PATCH 2/2] cohfar: Reduce the size of outputted XML files by only
 using active IFOs

Previously, every possible combination of existing IFO was outputted as
a part of the XML dump from `cohfar`. This meant that the size of the
XML dump wasn't tied to the actual number of IFOs in use, and would
continue to exponentially increase in size as we increased the number of
IFOs that might be in play (e.g. KAGRA, LIGO-India, etc.).

To solve this, we have moved to using the total combination (as an index
into IFOComboMap) as a store of the active IFOs, and using
`__builtin_popcount()` to get the total number of IFOs that are in use
in that combination. This means that we can quickly and efficiently
determine the maximum number of detectors that we actually need to print
stats for, with no additional information required.

-- Tom Almeida
---
 .../gst/cuda/cohfar/background_stats.h        |   9 +-
 .../gst/cuda/cohfar/background_stats_utils.c  | 552 ++++++++++++++----
 .../gst/cuda/cohfar/background_stats_utils.h  |   5 +-
 .../gst/cuda/cohfar/cohfar_accumbackground.c  |  67 +--
```

```
 .../gst/cuda/cohfar/cohfar_accumbackground.h  |   4 +-
 .../gst/cuda/cohfar/cohfar_assignfar.c        |   6 +-
 .../gst/cuda/cohfar/cohfar_assignfar.h        |   2 +-
 .../gst/cuda/cohfar/cohfar_calc_fap.c         |  90 ++--
 8 files changed, 356 insertions(+), 379 deletions(-)
diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats.h b/gstlal-spiir/gst/cuda/cohfar/background_stats.h
index 40f50290..7f88f80e 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats.h
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats.h
@@ -96,14 +96,7 @@ typedef struct {
     TriggerStats **multistats;
     GString *rank_xmlname;
     GString *feature_xmlname;
-    int ncombo;
+    int icombo;
 } TriggerStatsXML;

-typedef TriggerStats **TriggerStatsPointer;
-typedef struct {
-    TriggerStatsPointer *plist;
-    int size;
-    int pos;
-} TriggerStatsPointerList;
-
 #endif /* __BACKGROUND_STATS_H__ */
diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c b/gstlal-spiir/gst/cuda/cohfar/background_
index 45c57ec6..449cf376 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
@@ -1,5 +1,6 @@
 /*
- * Copyright (C) 2015 Qi Chu <qi.chu@ligo.org>
+ * Copyright (C) 2015 Qi Chu <qi.chu@ligo.org>,
+ *               2020 Tom Almeida <tom@tommoa.me>,
  *
  * Permission is hereby granted, free of charge, to any person obtaining a
  * copy of this software and associated documentation files (the
@@ -98,11 +99,6 @@ int get_icombo(char *ifos) {
     return -1;
 }

-int get_ncombo(int nifo) {
-    g_assert(pow(2, nifo) - 1 <= MAX_IFO_COMBOS);
-    return MAX_IFO_COMBOS;
-}
-
 Bins1D *bins1D_long_create(double cmin, double cmax, int nbin) {
     Bins1D *bins = (Bins1D *)malloc(sizeof(Bins1D));
     bins->cmin   = cmin;
@@ -187,22 +183,23 @@ void bins2D_long_destroy(Bins2D *bins) {
     free(bins);
 }

-void trigger_stats_reset(TriggerStats **multistats, int ncombo) {
-    int icombo;
+void trigger_stats_reset(TriggerStats **multistats, int nifo) {
+    int ifo;
     FeatureStats *feature;
-    for (icombo = 0; icombo < ncombo; icombo++) {
-        feature = multistats[icombo]->feature;
+    for (ifo = 0; ifo <= nifo; ifo++) {
+        feature = multistats[ifo]->feature;
         gsl_vector_long_set_zero((gsl_vector_long *)feature->lgsnr_rate->data);
         gsl_vector_long_set_zero(
           (gsl_vector_long *)feature->lgchisq_rate->data);
         gsl_matrix_long_set_zero(
           (gsl_matrix_long *)feature->lgsnr_lgchisq_rate->data);
-        multistats[icombo]->nevent   = 0;
-        multistats[icombo]->livetime = 0;
+        multistats[ifo]->nevent   = 0;
+        multistats[ifo]->livetime = 0;
     }
 }
 void trigger_stats_xml_reset(TriggerStatsXML *stats) {
```

```
-    trigger_stats_reset(stats->multistats, stats->ncombo);
+    trigger_stats_reset(stats->multistats,
+                        __builtin_popcount(stats->icombo + 1));
 }
 FeatureStats *feature_stats_create() {
@@ -254,32 +251,59 @@ void rank_stats_destroy(RankingStats *rank) {
     free(rank);
 }
-TriggerStats **trigger_stats_create(int ncombo) {
+TriggerStats **trigger_stats_create(int icombo) {
+    // [THA]: We can see the number of detectors in a interferometer combination
+    // by checking the number of set bits in `icombo + 1`. We can do this
+    // because icombo is one less than the power of two combination of detectors
+    // (see `include/pipe_macro.h`)
+    int nifo = __builtin_popcount(icombo + 1);
+    // We only create TriggerStats for each individual IFO and their final
+    // total combination (e.g. (H1, L1, H1L1) or (H1, L1, V1, H1L1V1))
+    // Thus, the total number of combinations is the number of individual IFOs
+    // in the combo + 1
     TriggerStats **multistats =
-      (TriggerStats **)malloc(sizeof(TriggerStats *) * ncombo);
-
-    int icombo = 0;
-    for (icombo = 0; icombo < ncombo; icombo++) {
-        multistats[icombo]       = (TriggerStats *)malloc(sizeof(TriggerStats));
-        TriggerStats *cur_stats = multistats[icombo];
-        // FIXME: what if HV or LV combo
-        // printf("len %s, %d\n", IFOComboMap[icombo].name,
-        // strlen(IFOComboMap[icombo].name));
-        cur_stats->ifos =
-          malloc(strlen(IFOComboMap[icombo].name) * sizeof(char));
-        strncpy(cur_stats->ifos, IFOComboMap[icombo].name,
-                strlen(IFOComboMap[icombo].name) * sizeof(char));
-        // create feature
-        cur_stats->feature = feature_stats_create();
-        // our rank, cdf
-        cur_stats->rank     = rank_stats_create();
-        cur_stats->nevent   = 0;
-        cur_stats->livetime = 0;
+      (TriggerStats **)malloc(sizeof(TriggerStats *) * (nifo + 1));
+
+    // Allocate for the final combination (all IFOs together)
+    multistats[nifo]        = (TriggerStats *)malloc(sizeof(TriggerStats));
+    TriggerStats *cur_stats = multistats[nifo];
+    cur_stats->ifos =
+      malloc(strlen(IFOComboMap[icombo].name) * sizeof(char) + 1);
+    strncpy(cur_stats->ifos, IFOComboMap[icombo].name,
+            strlen(IFOComboMap[icombo].name) * sizeof(char) + 1);
+    // create feature
+    cur_stats->feature = feature_stats_create();
+    // our rank, cdf
+    cur_stats->rank     = rank_stats_create();
+    cur_stats->nevent   = 0;
+    cur_stats->livetime = 0;
+
+    // Individual IFOs
+    int ifo = 0, index = 0;
+    ++icombo;
+    for (ifo = 0; ifo < MAX_NIFO; ifo++) {
+        // Is this IFO in the combo?
+        if (icombo & (1 << ifo)) {
+            multistats[index] = (TriggerStats *)malloc(sizeof(TriggerStats));
+            cur_stats         = multistats[index];
+            cur_stats->ifos =
+              malloc(strlen(IFOMap[ifo].name) * sizeof(char) + 1);
+            strncpy(cur_stats->ifos, IFOMap[ifo].name,
+                    strlen(IFOMap[ifo].name) * sizeof(char) + 1);
+            // create feature
+            cur_stats->feature = feature_stats_create();
+            // our rank, cdf
+            cur_stats->rank     = rank_stats_create();
```

```
+            cur_stats->nevent   = 0;
+            cur_stats->livetime = 0;
+            ++index;
+        }
     }
     return multistats;
 }
 TriggerStatsXML *trigger_stats_xml_create(char *ifos, int stats_type) {
+    // Create the XML document for tracking trigger stats
     TriggerStatsXML *stats = (TriggerStatsXML *)malloc(sizeof(TriggerStatsXML));
     if (stats_type == STATS_XML_TYPE_BACKGROUND) {
         stats->feature_xmlname = g_string_new(BACKGROUND_XML_FEATURE_NAME);
@@ -293,18 +317,15 @@ TriggerStatsXML *trigger_stats_xml_create(char *ifos, int stats_type) {
         printf("create sgstats %s\n", stats->feature_xmlname->str);
     }
-    int nifo          = strlen(ifos) / IFO_LEN;
-    int ncombo        = get_ncombo(nifo);
-    stats->multistats = trigger_stats_create(ncombo);
-    stats->ncombo     = ncombo;
+    int icombo        = get_icombo(ifos);
+    stats->multistats = trigger_stats_create(icombo);
+    stats->icombo     = icombo;
     return stats;
 }
-void trigger_stats_destroy(TriggerStats **multistats, int ncombo) {
-    int icombo = 0;
-
-    for (icombo = 0; icombo < ncombo; icombo++) {
-        TriggerStats *cur_stats = multistats[icombo];
+void trigger_stats_destroy(TriggerStats **multistats, int nifo) {
+    for (int ifo = 0; ifo <= nifo; ifo++) {
+        TriggerStats *cur_stats = multistats[ifo];
         feature_stats_destroy(cur_stats->feature);
         cur_stats->feature = NULL;
         rank_stats_destroy(cur_stats->rank);
@@ -320,41 +341,9 @@ void trigger_stats_destroy(TriggerStats **multistats, int ncombo) {
 void trigger_stats_xml_destroy(TriggerStatsXML *stats) {
     g_string_free(stats->feature_xmlname, TRUE);
     g_string_free(stats->rank_xmlname, TRUE);
-    trigger_stats_destroy(stats->multistats, stats->ncombo);
-}
-
-TriggerStatsPointerList *trigger_stats_list_create(char *ifos) {
-    int nifo = 0, ncombo = 0, icombo = 0;
-    nifo = strlen(ifos) / IFO_LEN;
-
-    ncombo = get_ncombo(nifo);
-    TriggerStatsPointerList *stats_list =
-      (TriggerStatsPointerList *)malloc(sizeof(TriggerStatsPointerList));
-    stats_list->size  = NSTATS_TO_PROMPT;
-    stats_list->pos   = 0;
-    stats_list->plist = (TriggerStatsPointer *)malloc(
-      sizeof(TriggerStatsPointer) * NSTATS_TO_PROMPT);
-
-    int ilist           = 0;
-    TriggerStats **stats = NULL;
-
-    for (ilist = 0; ilist < NSTATS_TO_PROMPT; ilist++) {
-        stats = (TriggerStats **)malloc(sizeof(TriggerStats *) * ncombo);
-        for (icombo = 0; icombo < ncombo; icombo++) {
-            stats[icombo] = (TriggerStats *)malloc(sizeof(TriggerStats));
-            TriggerStats *cur_stats = stats[icombo];
-            // printf("len %s, %d\n", IFOComboMap[icombo].name,
-            // strlen(IFOComboMap[icombo].name));
-            cur_stats->ifos =
-              malloc(strlen(IFOComboMap[icombo].name) * sizeof(char));
-            strncpy(cur_stats->ifos, IFOComboMap[icombo].name,
-                    strlen(IFOComboMap[icombo].name) * sizeof(char));
-            cur_stats->nevent   = 0;
-            cur_stats->livetime = 0;
-        }
-        stats_list->plist[ilist] = stats;
```

```
-    }
-    return stats_list;
+    trigger_stats_destroy(stats->multistats,
+                          __builtin_popcount(stats->icombo + 1));
+    free(stats);
 }
 /*
@@ -434,14 +423,14 @@ void trigger_stats_feature_rate_add(FeatureStats *feature1,
 void trigger_stats_livetime_add(TriggerStats **stats_out,
                                 TriggerStats **stats_in,
-                                const int icombo) {
-    stats_out[icombo]->livetime += stats_in[icombo]->livetime;
+                                const int index) {
+    stats_out[index]->livetime += stats_in[index]->livetime;
 }
 /*
  * background pdf direnctly from rate
  */
-void trigger_stats_livetime_inc(TriggerStats **stats, const int icombo) {
-    stats[icombo]->livetime += 1;
+void trigger_stats_livetime_inc(TriggerStats **stats, const int index) {
+    stats[index]->livetime += 1;
 }
 void trigger_stats_feature_rate_to_pdf_hist(FeatureStats *feature,
@@ -751,10 +740,10 @@ static void signal_stats_gen_ratemap_from_pdf(FeatureStats *feature) {
 }
 void signal_stats_init(TriggerStatsXML *sgstats, int source_type) {
-    int icombo, ncombo = sgstats->ncombo;
+    int ifo, nifo = __builtin_popcount(sgstats->icombo + 1);
     if (source_type == SOURCE_TYPE_BNS) {
-        for (icombo = 0; icombo < ncombo; icombo++) {
-            TriggerStats *stats = sgstats->multistats[icombo];
+        for (ifo = 0; ifo <= nifo; ifo++) {
+            TriggerStats *stats = sgstats->multistats[ifo];
             signal_stats_gen_pdfmap(stats->feature->lgsnr_lgchisq_pdf);
             signal_stats_gen_ratemap_from_pdf(stats->feature);
         }
@@ -999,99 +988,108 @@ gboolean trigger_stats_xml_from_xml(TriggerStatsXML *stats,
     if (!g_file_test(filename, G_FILE_TEST_EXISTS)) { return FALSE; }
     int nelem  = 10; // 4 for feature, 4 for rank, 2 for nevent,livetime
-    int ncombo = stats->ncombo;
-    int nnode  = ncombo * nelem + 1, icombo; // 1 for hist_trials
+    int icombo = stats->icombo;
+    int nifo   = __builtin_popcount(icombo + 1);
+    int nodes  = nifo + 1; // top level nodes
+    int nnode  = nodes * nelem + 1, combo; // 1 for hist_trials
     /* read rate */
     XmlNodeStruct *xns = (XmlNodeStruct *)malloc(sizeof(XmlNodeStruct) * nnode);
-    XmlArray *array_lgsnr_rate = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_lgchisq_rate =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+    XmlArray *array_lgsnr_rate   = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+    XmlArray *array_lgchisq_rate = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
     XmlArray *array_lgsnr_lgchisq_rate =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+      (XmlArray *)malloc(sizeof(XmlArray) * nodes);
     XmlArray *array_lgsnr_lgchisq_pdf =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_map  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlParam *param_nevent    = (XmlParam *)malloc(sizeof(XmlParam) * ncombo);
-    XmlParam *param_livetime  = (XmlParam *)malloc(sizeof(XmlParam) * ncombo);
-    XmlArray *array_rank_rate = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_pdf  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_fap  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-
-    int pos_xns;
-    for (icombo = 0; icombo < ncombo; icombo++) {
-        pos_xns = icombo;
+      (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+    XmlArray *array_rank_map  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+    XmlParam *param_nevent    = (XmlParam *)malloc(sizeof(XmlParam) * ncombo);
+    XmlParam *param_livetime  = (XmlParam *)malloc(sizeof(XmlParam) * ncombo);
+    XmlArray *array_rank_rate = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+    XmlArray *array_rank_pdf  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+    XmlArray *array_rank_fap  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+
+    int pos_xns;
+    for (combo = 0; combo < ncombo; combo++) {
+        pos_xns = combo;
```

```
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name,
-                 SNR_RATE_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_lgsnr_rate[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name,
-                 CHISQ_RATE_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_lgchisq_rate[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name,
-                 SNR_CHISQ_RATE_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_lgsnr_lgchisq_rate[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name,
-                 SNR_CHISQ_PDF_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_lgsnr_lgchisq_pdf[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->rank_xmlname->str, IFOComboMap[icombo].name,
-                 RANK_MAP_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_rank_map[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_nevent:param",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name);
-         xns[pos_xns].processPtr = readParam;
-         xns[pos_xns].data       = &(param_nevent[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_livetime:param",
-                 stats->feature_xmlname->str, IFOComboMap[icombo].name);
-         xns[pos_xns].processPtr = readParam;
-         xns[pos_xns].data       = &(param_livetime[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->rank_xmlname->str, IFOComboMap[icombo].name,
-                 RANK_RATE_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_rank_rate[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->rank_xmlname->str, IFOComboMap[icombo].name,
-                 RANK_PDF_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_rank_pdf[icombo]);
-
-         pos_xns += ncombo;
-         sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
-                 stats->rank_xmlname->str, IFOComboMap[icombo].name,
-                 RANK_FAP_SUFFIX);
-         xns[pos_xns].processPtr = readArray;
-         xns[pos_xns].data       = &(array_rank_fap[icombo]);
+       (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+     XmlArray *array_rank_map  = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+     XmlParam *param_nevent    = (XmlParam *)malloc(sizeof(XmlParam) * nodes);
+     XmlParam *param_livetime  = (XmlParam *)malloc(sizeof(XmlParam) * nodes);
+     XmlArray *array_rank_rate = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+     XmlArray *array_rank_pdf  = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
+     XmlArray *array_rank_fap  = (XmlArray *)malloc(sizeof(XmlArray) * nodes);
```

```
+
+    // [THA]: We hold `index` to be the index into the various arrays that we
+    // index for printing. We also only have this many (+1)s because icombo
+    // starts from 0 and we need to make sure that the 0th combo & icombo != 0
+    // if its not in the actual combo. Thus we add 1 to get the "actual"
+    // combination and just use combo for indexing the combomap
+    int pos_xns, index;
+    for (combo = 0, index = 0; combo < icombo + 1; combo++) {
+        if ((combo + 1) & (icombo + 1) == combo + 1) {
+            pos_xns = index;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name,
+                    SNR_RATE_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_lgsnr_rate[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name,
+                    CHISQ_RATE_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_lgchisq_rate[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name,
+                    SNR_CHISQ_RATE_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_lgsnr_lgchisq_rate[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name,
+                    SNR_CHISQ_PDF_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_lgsnr_lgchisq_pdf[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->rank_xmlname->str, IFOComboMap[combo].name,
+                    RANK_MAP_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_rank_map[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_nevent:param",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name);
+            xns[pos_xns].processPtr = readParam;
+            xns[pos_xns].data       = &(param_nevent[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_livetime:param",
+                    stats->feature_xmlname->str, IFOComboMap[combo].name);
+            xns[pos_xns].processPtr = readParam;
+            xns[pos_xns].data       = &(param_livetime[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->rank_xmlname->str, IFOComboMap[combo].name,
+                    RANK_RATE_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_rank_rate[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->rank_xmlname->str, IFOComboMap[combo].name,
+                    RANK_PDF_SUFFIX);
+            xns[pos_xns].processPtr = readArray;
+            xns[pos_xns].data       = &(array_rank_pdf[index]);
+
+            pos_xns += nifo;
+            sprintf((char *)xns[pos_xns].tag, "%s:%s_%s:array",
+                    stats->rank_xmlname->str, IFOComboMap[combo].name,
```

```
+                   RANK_FAP_SUFFIX);
+           xns[pos_xns].processPtr = readArray;
+           xns[pos_xns].data       = &(array_rank_fap[index]);
+           index += 1;
+       }
    }
    XmlParam *param_hist_trials = (XmlParam *)malloc(sizeof(XmlParam) * 1);
-   pos_xns             = nelem * ncombo;
+   pos_xns             = nelem * nifo;
    GString *hist_name = g_string_new(NULL);
    g_string_printf(hist_name, "%s:hist_trials:param",
                    stats->feature_xmlname->str);
@@ -1116,32 +1114,32 @@ gboolean trigger_stats_xml_from_xml(TriggerStatsXML *stats,
    g_assert(array_lgsnr_rate[0].dim[0] == nbin_x);
    g_assert(array_lgchisq_rate[0].dim[0] == nbin_y);
-   for (icombo = 0; icombo < ncombo; icombo++) {
-       TriggerStats *cur_stats = multistats[icombo];
+   for (index = 0; index < nifo; index++) {
+       TriggerStats *cur_stats = multistats[index];
        FeatureStats *feature   = cur_stats->feature;
        RankingStats *rank      = cur_stats->rank;
        memcpy(((gsl_vector_long *)feature->lgsnr_rate->data)->data,
-              (long *)array_lgsnr_rate[icombo].data, x_size);
+              (long *)array_lgsnr_rate[index].data, x_size);
        memcpy(((gsl_vector_long *)feature->lgchisq_rate->data)->data,
-              (long *)array_lgchisq_rate[icombo].data, y_size);
+              (long *)array_lgchisq_rate[index].data, y_size);
        memcpy(((gsl_matrix_long *)feature->lgsnr_lgchisq_rate->data)->data,
-              (long *)array_lgsnr_lgchisq_rate[icombo].data, xy_size);
+              (long *)array_lgsnr_lgchisq_rate[index].data, xy_size);
        memcpy(((gsl_matrix *)feature->lgsnr_lgchisq_pdf->data)->data,
-              array_lgsnr_lgchisq_pdf[icombo].data, xy_size);
+              array_lgsnr_lgchisq_pdf[index].data, xy_size);
        memcpy(((gsl_matrix *)rank->rank_map->data)->data,
-              array_rank_map[icombo].data, xy_size);
+              array_rank_map[index].data, xy_size);
        memcpy(((gsl_vector_long *)rank->rank_rate->data)->data,
-              (long *)array_rank_rate[icombo].data, y_size);
+              (long *)array_rank_rate[index].data, y_size);
        memcpy(((gsl_vector *)rank->rank_pdf->data)->data,
-              (long *)array_rank_pdf[icombo].data, y_size);
+              (long *)array_rank_pdf[index].data, y_size);
        memcpy(((gsl_vector *)rank->rank_fap->data)->data,
-              (long *)array_rank_fap[icombo].data, y_size);
-       cur_stats->nevent   = *((long *)param_nevent[icombo].data);
-       cur_stats->livetime = *((long *)param_livetime[icombo].data);
+              (long *)array_rank_fap[index].data, y_size);
+       cur_stats->nevent   = *((long *)param_nevent[index].data);
+       cur_stats->livetime = *((long *)param_livetime[index].data);
        // printf("filename %s, icombo %d, fap addr %p\n", filename, icombo,
-       // ((gsl_matrix *)cur_stats->fap->data)->data); printf("icombo %d, nevent
-       // addr %p, %p\n", icombo, (param_nevent[icombo].data),
+       // ((gsl_matrix *)cur_stats->fap->data)->data); printf("icombo %d,
+       // nevent addr %p, %p\n", icombo, (param_nevent[icombo].data),
        // (&(param_nevent[icombo]))->data);
    }
    *hist_trials = *((int *)param_hist_trials->data);
@@ -1160,17 +1158,17 @@ gboolean trigger_stats_xml_from_xml(TriggerStatsXML *stats,
    /*
     * free the allocated memory for xml reading
     */
-   for (icombo = 0; icombo < ncombo; icombo++) {
-       free(array_lgsnr_rate[icombo].data);
-       free(array_lgchisq_rate[icombo].data);
-       free(array_lgsnr_lgchisq_rate[icombo].data);
-       free(array_lgsnr_lgchisq_pdf[icombo].data);
-       free(param_nevent[icombo].data);
-       free(param_livetime[icombo].data);
-       free(array_rank_map[icombo].data);
```

```
-        free(array_rank_rate[icombo].data);
-        free(array_rank_pdf[icombo].data);
-        free(array_rank_fap[icombo].data);
+    for (index = 0; index < nifo; index++) {
+        free(array_lgsnr_rate[index].data);
+        free(array_lgchisq_rate[index].data);
+        free(array_lgsnr_lgchisq_rate[index].data);
+        free(array_lgsnr_lgchisq_pdf[index].data);
+        free(param_nevent[index].data);
+        free(param_livetime[index].data);
+        free(array_rank_map[index].data);
+        free(array_rank_rate[index].data);
+        free(array_rank_pdf[index].data);
+        free(array_rank_fap[index].data);
     }
     free(array_lgsnr_rate);
     free(array_lgchisq_rate);
@@ -1293,18 +1291,19 @@ gboolean trigger_stats_xml_dump(TriggerStatsXML *stats,
     }
     printf("write %s\n", stats->rank_xmlname->str);
     xmlTextWriterPtr writer = *pwriter;
-    int icombo = 0, ncombo = stats->ncombo;
-    XmlArray *array_lgsnr_rate = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+    int ifo = 0, nifo = __builtin_popcount(stats->icombo + 1);
+    int nnodes                = nifo + 1;
+    XmlArray *array_lgsnr_rate = (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
     XmlArray *array_lgchisq_rate =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+      (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
     XmlArray *array_lgsnr_lgchisq_rate =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+      (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
     XmlArray *array_lgsnr_lgchisq_pdf =
-      (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_map  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_rate = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_pdf  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
-    XmlArray *array_rank_fap  = (XmlArray *)malloc(sizeof(XmlArray) * ncombo);
+      (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
+    XmlArray *array_rank_map  = (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
+    XmlArray *array_rank_rate = (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
+    XmlArray *array_rank_pdf  = (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
+    XmlArray *array_rank_fap  = (XmlArray *)malloc(sizeof(XmlArray) * nnodes);
     TriggerStats **multistats = stats->multistats;
     int nbin_x = multistats[0]->feature->lgsnr_lgchisq_pdf->nbin_x,
@@ -1312,61 +1311,61 @@ gboolean trigger_stats_xml_dump(TriggerStatsXML *stats,
     int x_size = sizeof(double) * nbin_x, y_size = sizeof(double) * nbin_y;
     int xy_size = sizeof(double) * nbin_x * nbin_y;

-    for (icombo = 0; icombo < ncombo; icombo++) {
-        TriggerStats *cur_stats = multistats[icombo];
+    for (ifo = 0; ifo < nnodes; ifo++) {
+        TriggerStats *cur_stats = multistats[ifo];
         FeatureStats *feature   = cur_stats->feature;
         RankingStats *rank      = cur_stats->rank;
         // assemble lgsnr_rate
-        array_lgsnr_rate[icombo].ndim   = 1;
-        array_lgsnr_rate[icombo].dim[0] = nbin_x;
-        array_lgsnr_rate[icombo].data   = (long *)malloc(x_size);
-        memcpy(array_lgsnr_rate[icombo].data,
+        array_lgsnr_rate[ifo].ndim   = 1;
+        array_lgsnr_rate[ifo].dim[0] = nbin_x;
+        array_lgsnr_rate[ifo].data   = (long *)malloc(x_size);
+        memcpy(array_lgsnr_rate[ifo].data,
               ((gsl_vector_long *)feature->lgsnr_rate->data)->data, x_size);
         // assemble lgchisq_rate
-        array_lgchisq_rate[icombo].ndim   = 1;
-        array_lgchisq_rate[icombo].dim[0] = nbin_y;
-        array_lgchisq_rate[icombo].data   = (long *)malloc(y_size);
-        memcpy(array_lgchisq_rate[icombo].data,
+        array_lgchisq_rate[ifo].ndim   = 1;
```

```
+       array_lgchisq_rate[ifo].dim[0] = nbin_y;
+       array_lgchisq_rate[ifo].data   = (long *)malloc(y_size);
+       memcpy(array_lgchisq_rate[ifo].data,
               ((gsl_vector_long *)feature->lgchisq_rate->data)->data, y_size);
        // assemble lgsnr_lgchisq_rate
-       array_lgsnr_lgchisq_rate[icombo].ndim   = 2;
-       array_lgsnr_lgchisq_rate[icombo].dim[0] = nbin_x;
-       array_lgsnr_lgchisq_rate[icombo].dim[1] = nbin_y;
-       array_lgsnr_lgchisq_rate[icombo].data   = (long *)malloc(xy_size);
-       memcpy(array_lgsnr_lgchisq_rate[icombo].data,
+       array_lgsnr_lgchisq_rate[ifo].ndim   = 2;
+       array_lgsnr_lgchisq_rate[ifo].dim[0] = nbin_x;
+       array_lgsnr_lgchisq_rate[ifo].dim[1] = nbin_y;
+       array_lgsnr_lgchisq_rate[ifo].data   = (long *)malloc(xy_size);
+       memcpy(array_lgsnr_lgchisq_rate[ifo].data,
               ((gsl_matrix_long *)feature->lgsnr_lgchisq_rate->data)->data,
               xy_size);
        // aseemble lgsnr_lgchisq_pdf
-       array_lgsnr_lgchisq_pdf[icombo].ndim   = 2;
-       array_lgsnr_lgchisq_pdf[icombo].dim[0] = nbin_x;
-       array_lgsnr_lgchisq_pdf[icombo].dim[1] = nbin_y;
-       array_lgsnr_lgchisq_pdf[icombo].data   = (double *)malloc(xy_size);
-       memcpy(array_lgsnr_lgchisq_pdf[icombo].data,
+       array_lgsnr_lgchisq_pdf[ifo].ndim   = 2;
+       array_lgsnr_lgchisq_pdf[ifo].dim[0] = nbin_x;
+       array_lgsnr_lgchisq_pdf[ifo].dim[1] = nbin_y;
+       array_lgsnr_lgchisq_pdf[ifo].data   = (double *)malloc(xy_size);
+       memcpy(array_lgsnr_lgchisq_pdf[ifo].data,
               ((gsl_matrix *)feature->lgsnr_lgchisq_pdf->data)->data, xy_size);
        // assemble rank_map
-       array_rank_map[icombo].ndim   = 2;
-       array_rank_map[icombo].dim[0] = nbin_x;
-       array_rank_map[icombo].dim[1] = nbin_y;
-       array_rank_map[icombo].data   = (double *)malloc(x_size * y_size);
-       memcpy(array_rank_map[icombo].data,
+       array_rank_map[ifo].ndim   = 2;
+       array_rank_map[ifo].dim[0] = nbin_x;
+       array_rank_map[ifo].dim[1] = nbin_y;
+       array_rank_map[ifo].data   = (double *)malloc(x_size * y_size);
+       memcpy(array_rank_map[ifo].data,
               ((gsl_matrix *)rank->rank_map->data)->data, xy_size);
        // assemble rank_rate
-       array_rank_rate[icombo].ndim   = 1;
-       array_rank_rate[icombo].dim[0] = nbin_x;
-       array_rank_rate[icombo].data   = (long *)malloc(x_size);
-       memcpy(array_rank_rate[icombo].data,
+       array_rank_rate[ifo].ndim   = 1;
+       array_rank_rate[ifo].dim[0] = nbin_x;
+       array_rank_rate[ifo].data   = (long *)malloc(x_size);
+       memcpy(array_rank_rate[ifo].data,
               ((gsl_vector_long *)rank->rank_rate->data)->data, x_size);
        // assemble rank_pdf
-       array_rank_pdf[icombo].ndim   = 1;
-       array_rank_pdf[icombo].dim[0] = nbin_x;
-       array_rank_pdf[icombo].data   = (double *)malloc(x_size);
-       memcpy(array_rank_pdf[icombo].data,
+       array_rank_pdf[ifo].ndim   = 1;
+       array_rank_pdf[ifo].dim[0] = nbin_x;
+       array_rank_pdf[ifo].data   = (double *)malloc(x_size);
+       memcpy(array_rank_pdf[ifo].data,
               ((gsl_vector *)rank->rank_pdf->data)->data, x_size);
        // assemble rank_fap_
-       array_rank_fap[icombo].ndim   = 1;
-       array_rank_fap[icombo].dim[0] = nbin_x;
-       array_rank_fap[icombo].data   = (double *)malloc(x_size);
-       memcpy(array_rank_fap[icombo].data,
+       array_rank_fap[ifo].ndim   = 1;
+       array_rank_fap[ifo].dim[0] = nbin_x;
+       array_rank_fap[ifo].data   = (double *)malloc(x_size);
+       memcpy(array_rank_fap[ifo].data,
```

```
                ((gsl_vector *)rank->rank_fap->data)->data, x_size);
    }
@@ -1409,63 +1408,58 @@ gboolean trigger_stats_xml_dump(TriggerStatsXML *stats,
    GString *array_name = g_string_new(NULL);
    GString *param_name = g_string_new(NULL);
-   for (icombo = 0; icombo < ncombo; icombo++) {
+   for (ifo = 0; ifo < nnodes; ifo++) {
        // write features
        g_string_printf(array_name, "%s:%s_%s:array",
-                       stats->feature_xmlname->str, IFOComboMap[icombo].name,
+                       stats->feature_xmlname->str, multistats[ifo]->ifos,
                        SNR_RATE_SUFFIX);
-       ligoxml_write_Array(writer, &(array_lgsnr_rate[icombo]),
-                           BAD_CAST "int_8s", BAD_CAST " ",
-                           BAD_CAST array_name->str);
+       ligoxml_write_Array(writer, &(array_lgsnr_rate[ifo]), BAD_CAST "int_8s",
+                           BAD_CAST " ", BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array",
-                       stats->feature_xmlname->str, IFOComboMap[icombo].name,
+                       stats->feature_xmlname->str, multistats[ifo]->ifos,
                        CHISQ_RATE_SUFFIX);
-       ligoxml_write_Array(writer, &(array_lgchisq_rate[icombo]),
+       ligoxml_write_Array(writer, &(array_lgchisq_rate[ifo]),
                            BAD_CAST "int_8s", BAD_CAST " ",
                            BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array",
-                       stats->feature_xmlname->str, IFOComboMap[icombo].name,
+                       stats->feature_xmlname->str, multistats[ifo]->ifos,
                        SNR_CHISQ_RATE_SUFFIX);
-       ligoxml_write_Array(writer, &(array_lgsnr_lgchisq_rate[icombo]),
+       ligoxml_write_Array(writer, &(array_lgsnr_lgchisq_rate[ifo]),
                            BAD_CAST "int_8s", BAD_CAST " ",
                            BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array",
-                       stats->feature_xmlname->str, IFOComboMap[icombo].name,
+                       stats->feature_xmlname->str, multistats[ifo]->ifos,
                        SNR_CHISQ_PDF_SUFFIX);
-       ligoxml_write_Array(writer, &(array_lgsnr_lgchisq_pdf[icombo]),
+       ligoxml_write_Array(writer, &(array_lgsnr_lgchisq_pdf[ifo]),
                            BAD_CAST "real_8", BAD_CAST " ",
                            BAD_CAST array_name->str);

        // write rank
        g_string_printf(array_name, "%s:%s_%s:array", stats->rank_xmlname->str,
-                       IFOComboMap[icombo].name, RANK_MAP_SUFFIX);
-       ligoxml_write_Array(writer, &(array_rank_map[icombo]),
-                           BAD_CAST "real_8", BAD_CAST " ",
-                           BAD_CAST array_name->str);
+                       multistats[ifo]->ifos, RANK_MAP_SUFFIX);
+       ligoxml_write_Array(writer, &(array_rank_map[ifo]), BAD_CAST "real_8",
+                           BAD_CAST " ", BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array", stats->rank_xmlname->str,
-                       IFOComboMap[icombo].name, RANK_RATE_SUFFIX);
-       ligoxml_write_Array(writer, &(array_rank_rate[icombo]),
-                           BAD_CAST "int_8s", BAD_CAST " ",
-                           BAD_CAST array_name->str);
+                       multistats[ifo]->ifos, RANK_RATE_SUFFIX);
+       ligoxml_write_Array(writer, &(array_rank_rate[ifo]), BAD_CAST "int_8s",
+                           BAD_CAST " ", BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array", stats->rank_xmlname->str,
-                       IFOComboMap[icombo].name, RANK_PDF_SUFFIX);
-       ligoxml_write_Array(writer, &(array_rank_pdf[icombo]),
-                           BAD_CAST "real_8", BAD_CAST " ",
-                           BAD_CAST array_name->str);
+                       multistats[ifo]->ifos, RANK_PDF_SUFFIX);
+       ligoxml_write_Array(writer, &(array_rank_pdf[ifo]), BAD_CAST "real_8",
+                           BAD_CAST " ", BAD_CAST array_name->str);
        g_string_printf(array_name, "%s:%s_%s:array", stats->rank_xmlname->str,
-                       IFOComboMap[icombo].name, RANK_FAP_SUFFIX);
-       ligoxml_write_Array(writer, &(array_rank_fap[icombo]),
+                       multistats[ifo]->ifos, RANK_FAP_SUFFIX);
+       ligoxml_write_Array(writer, &(array_rank_fap[ifo]),
```

```
-                               BAD_CAST "real_8", BAD_CAST " ",
-                               BAD_CAST array_name->str);
+                             multistats[ifo]->ifos, RANK_FAP_SUFFIX);
+       ligoxml_write_Array(writer, &(array_rank_fap[ifo]), BAD_CAST "real_8",
+                               BAD_CAST " ", BAD_CAST array_name->str);
        g_string_printf(param_name, "%s:%s_nevent:param",
-                               stats->feature_xmlname->str, IFOComboMap[icombo].name);
-       ((long *)param_nevent.data)[0] = multistats[icombo]->nevent;
+                               stats->feature_xmlname->str, multistats[ifo]->ifos);
+       ((long *)param_nevent.data)[0] = multistats[ifo]->nevent;
        ligoxml_write_Param(writer, &param_nevent, BAD_CAST "int_8s",
                                BAD_CAST param_name->str);
        g_string_printf(param_name, "%s:%s_livetime:param",
-                               stats->feature_xmlname->str, IFOComboMap[icombo].name);
-       ((long *)param_livetime.data)[0] = multistats[icombo]->livetime;
+                               stats->feature_xmlname->str, multistats[ifo]->ifos);
+       ((long *)param_livetime.data)[0] = multistats[ifo]->livetime;
        ligoxml_write_Param(writer, &param_livetime, BAD_CAST "int_8s",
                                BAD_CAST param_name->str);
    }
@@ -1501,15 +1495,15 @@ gboolean trigger_stats_xml_dump(TriggerStatsXML *stats,
    free(param_nevent.data);
    free(param_livetime.data);
    free(param_hist_trials.data);
-   for (icombo = ncombo - 1; icombo >= 0; icombo--) {
-       freeArray(array_lgsnr_rate + icombo);
-       freeArray(array_lgchisq_rate + icombo);
-       freeArray(array_lgsnr_lgchisq_rate + icombo);
-       freeArray(array_lgsnr_lgchisq_pdf + icombo);
-       freeArray(array_rank_map + icombo);
-       freeArray(array_rank_rate + icombo);
-       freeArray(array_rank_pdf + icombo);
-       freeArray(array_rank_fap + icombo);
+   for (int node = nnodes - 1; node >= 0; node--) {
+       freeArray(array_lgsnr_rate + node);
+       freeArray(array_lgchisq_rate + node);
+       freeArray(array_lgsnr_lgchisq_rate + node);
+       freeArray(array_lgsnr_lgchisq_pdf + node);
+       freeArray(array_rank_map + node);
+       freeArray(array_rank_rate + node);
+       freeArray(array_rank_pdf + node);
+       freeArray(array_rank_fap + node);
    }
    free(array_lgsnr_rate);
    free(array_lgchisq_rate);
diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h b/gstlal-spiir/gst/cuda/cohfar/background_
index 4c5990d4..9eda6a4a 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h
@@ -1,5 +1,6 @@
 /*
- * Copyright (C) 2015 Qi Chu <qi.chu@uwa.edu.au>
+ * Copyright (C) 2015 Qi Chu <qi.chu@uwa.edu.au>,
+ *               2020 Tom Almeida <tom@tommoa.me>,
  *
  * Permission is hereby granted, free of charge, to any person obtaining a
  * copy of this software and associated documentation files (the
@@ -48,7 +49,7 @@ Bins2D *bins2D_create_long(double cmin_x,
                            double cmax_y,
                            int nbin_y);

-TriggerStats **trigger_stats_create(int ncombo);
+TriggerStats **trigger_stats_create(int icombo);

 int bins1D_get_idx(double val, Bins1D *bins);

diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_accun
index 154cbda1..c742a658 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
@@ -1,5 +1,6 @@
 /*
```

```
- * Copyright (C) 2015  Qi Chu  <qi.chu@uwa.edu.au>
+ * Copyright (C) 2015  Qi Chu  <qi.chu@uwa.edu.au>,
+ *               2020  Tom Almeida <tom@tommoa.me>,
   *
   * This program is free software; you can redistribute it and/or modify
   * it under the terms of the GNU General Public License as published by
@@ -103,30 +104,27 @@ static gboolean cohfar_accumbackground_sink_event(GstPad *pad, GstEvent *event);
 static void cohfar_accumbackground_dispose(GObject *object);

 static void update_stats_icombo(PostcohInspiralTable *intable,
-                                int icombo,
-                                int ncombo,
                                 TriggerStatsXML *stats) {
-    int nifo, isingle, write_ifo_mapping[MAX_NIFO];
-    // update the multi-IFO background at the last bin.
-    if (icombo > -1) {
+    int nifo, ifo;
+    nifo = __builtin_popcount(stats->icombo + 1);
+
+    if (stats->icombo > -1) {
+        // update the multi-IFO background at the last bin.
         trigger_stats_feature_rate_update(
           (double)(intable->cohsnr), (double)intable->cmbchisq,
-          stats->multistats[ncombo - 1]->feature,
-          stats->multistats[ncombo - 1]);
+          stats->multistats[nifo]->feature, stats->multistats[nifo]);

-        nifo = strlen(intable->ifos) / IFO_LEN;
         /* add single detector stats */
-        get_write_ifo_mapping(IFOComboMap[icombo].name, nifo,
-                              write_ifo_mapping);
-
         // update single-IFO background according the single-IFO decomposition
-        for (isingle = 0; isingle < nifo; isingle++) {
-            int write_isingle = write_ifo_mapping[isingle];
-            trigger_stats_feature_rate_update(
-              (double)(*(&(intable->snglsnr_H) + write_isingle)),
-              (double)(*(&(intable->chisq_H) + write_isingle)),
-              stats->multistats[write_isingle]->feature,
-              stats->multistats[write_isingle]);
+        int index;
+        for (ifo = 0, index = 0; ifo < MAX_NIFO; ifo++) {
+            if ((stats->icombo + 1) & (1 << ifo)) {
+                trigger_stats_feature_rate_update(
+                  (double)((&intable->snglsnr_H)[ifo]),
+                  (double)((&intable->chisq_H)[ifo]),
+                  stats->multistats[index]->feature, stats->multistats[index]);
+                ++index;
+            }
         }
     }
 }
@@ -171,14 +169,14 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
     // TriggerStats **stats_prompt = element->stats_prompt;
     // TriggerStatsPointerList *stats_list = element->stats_list;
     // /* reset stats_prompt */
-    // trigger_stats_reset(stats_prompt, element->ncombo);
+    // trigger_stats_reset(stats_prompt, element->nifo);

     /*
      * reset stats in the stats_list in order to input new background points
      */
     // int pos = stats_list->pos;
     // TriggerStats **cur_stats_in_list = stats_list->plist[pos];
-    // trigger_stats_reset(cur_stats_in_list, element->ncombo);
+    // trigger_stats_reset(cur_stats_in_list, element->nifo);

     /*
      * calculate number of output postcoh entries
@@ -203,7 +201,7 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
     /* allocate extra space for prompt stats */
     // int out_size = sizeof(PostcohInspiralTable) * outentries +
-    // sizeof(TriggerStats) * ncombo;
```

```
+    // sizeof(TriggerStats) * (nifo + 1);
     int out_size = sizeof(PostcohInspiralTable) * outentries;
     result       = gst_pad_alloc_buffer(srcpad, 0, out_size, caps, &outbuf);
     if (result != GST_FLOW_OK) {
@@ -235,13 +233,13 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
         }
         if (intable->is_background == FLAG_BACKGROUND) {
             update_stats_icombo(
-              intable, icombo, element->ncombo,
-              bgstats); // update the last ncombo and single IFO stats
+              intable,
+              bgstats); // update the last combination and single IFO stats
         } else if (intable->is_background
                    == FLAG_FOREGROUND) { /* coherent trigger entry */
             update_stats_icombo(
-              intable, icombo, element->ncombo,
-              zlstats); // update the last ncombo and single IFO stats
+              intable,
+              zlstats); // update the last combination and single IFO stats
             memcpy(outtable, intable, sizeof(PostcohInspiralTable));
             outtable++;
         } else {
@@ -252,22 +250,15 @@ static GstFlowReturn cohfar_accumbackground_chain(GstPad *pad,
             //
             // Note that this is inverted because icombo is sum(1 << index) - 1
             if ((icombo + 1) & icombo) {
-                nifo = strlen(intable->ifos) / IFO_LEN;
+                nifo = __builtin_popcount(icombo + 1);
                 /* add single detector stats */
                 get_write_ifo_mapping(IFOComboMap[icombo].name, nifo,
                                       element->write_ifo_mapping);
-                for (isingle = 0; isingle < nifo; isingle++) {
-                    int write_isingle = element->write_ifo_mapping[isingle];
-                    trigger_stats_livetime_inc(bgstats->multistats,
-                                               write_isingle);
-                    trigger_stats_livetime_inc(zlstats->multistats,
-                                               write_isingle);
+                for (isingle = 0; isingle <= nifo; isingle++) {
+                    trigger_stats_livetime_inc(bgstats->multistats, isingle);
+                    trigger_stats_livetime_inc(zlstats->multistats, isingle);
                 }
-                trigger_stats_livetime_inc(bgstats->multistats,
-                                           element->ncombo - 1);
-                trigger_stats_livetime_inc(zlstats->multistats,
-                                           element->ncombo - 1);
             }
             memcpy(outtable, intable, sizeof(PostcohInspiralTable));
             outtable++;
@@ -422,7 +413,7 @@ static void cohfar_accumbackground_set_property(GObject *object,
     case PROP_IFOS:
         element->ifos    = g_value_dup_string(value);
         element->nifo    = strlen(element->ifos) / IFO_LEN;
-        element->ncombo = get_ncombo(element->nifo);
+        element->icombo = get_icombo(element->ifos);
         element->bgstats =
           trigger_stats_xml_create(element->ifos, STATS_XML_TYPE_BACKGROUND);
         element->zlstats =
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.h b/gstlal-spiir/gst/cuda/cohfar/cohfar_accun
index c3480659..07cf0a5e 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.h
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.h
@@ -32,8 +32,8 @@
 #include <glib.h>
 #include <gst/base/gstbasetransform.h>
 #include <gst/gst.h>
-#include <postcohtable.h>
 #include <libxml/xmlwriter.h>
+#include <postcohtable.h>
 G_BEGIN_DECLS
 #define COHFAR_ACCUMBACKGROUND_TYPE (cohfar_accumbackground_get_type())
@@ -59,7 +59,7 @@ typedef struct {
```

```
     char *ifos;
     int nifo;
-    int ncombo; // ifo combination
+    int icombo; // ifo combination
     int write_ifo_mapping[MAX_NIFO];
     TriggerStatsXML *bgstats;
     TriggerStatsXML *zlstats;
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
index ef5be930..8590bf2a 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
@@ -293,9 +293,9 @@ static GstFlowReturn cohfar_assignfar_transform_ip(GstBaseTransform *trans,
                 fprintf(stderr, "icombo not found, cohfar_assignfar\n");
                 exit(0);
             }
-            cur_stats = element->bgstats_1w->multistats[element->ncombo - 1];
+            cur_stats = element->bgstats_1w->multistats[element->nifo];
             if (icombo > -1 && cur_stats->nevent > MIN_BACKGROUND_NEVENT) {
-                update_trigger_fars(table, element->ncombo - 1, element);
+                update_trigger_fars(table, element->nifo, element);
             }
         }
     }
@@ -344,7 +344,7 @@ static void cohfar_assignfar_set_property(GObject *object,
     case PROP_IFOS:
         element->ifos   = g_value_dup_string(value);
         element->nifo   = strlen(element->ifos) / IFO_LEN;
-        element->ncombo = get_ncombo(element->nifo);
+        element->icombo = get_icombo(element->ifos);
         element->bgstats_1w =
           trigger_stats_xml_create(element->ifos, STATS_XML_TYPE_BACKGROUND);
         element->bgstats_1d =
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.h b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.h
index c3fb3223..0471da19 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.h
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.h
@@ -54,7 +54,7 @@ typedef struct {
     char *ifos;
     int nifo;
-    int ncombo; // ifo combination
+    int icombo; // ifo combination
     int hist_trials;
     TriggerStatsXML *bgstats_2h;
     TriggerStatsXML *bgstats_1d;
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_calc_fap.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_calc_fap.c
index 8a3fc991..49dcb142 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_calc_fap.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_calc_fap.c
@@ -1,5 +1,6 @@
 /*
- * Copyright (C) 2015 Qi Chu <qi.chu@uwa.edu.au>
+ * Copyright (C) 2015 Qi Chu <qi.chu@uwa.edu.au>,
+ *               2020 Tom Almeida <tom@tommoa.me>,
  *
  * Permission is hereby granted, free of charge, to any person obtaining a
  * copy of this software and associated documentation files (the
@@ -44,13 +45,21 @@ static void parse_opts(int argc,
     *ptype                    = g_strdup("all");
     *update_pdf               = 0;
     int option_index          = 0;
-    struct option long_opts[] = { { "input", required_argument, 0, 'i' },
-                                  { "input-format", required_argument, 0, 'f' },
-                                  { "output", required_argument, 0, 'o' },
-                                  { "ifos", required_argument, 0, 'd' },
-                                  { "type", required_argument, 0, 'u' },
-                                  { "update-pdf", no_argument, 0, 'p' },
-                                  { 0, 0, 0, 0 } };
+    struct option long_opts[] = {
+        // A comma separated list of files to use for input.
+        { "input", required_argument, 0, 'i' },
+        // The format of the input files. One of "data" or "stats".
```

```
+        { "input-format", required_argument, 0, 'f' },
+        // The name of the file to output.
+        { "output", required_argument, 0, 'o' },
+        // The IFOs to use.
+        { "ifos", required_argument, 0, 'd' },
+        // The type of stat. One of "background", "zerolag", "signal" or "all".
+        { "type", required_argument, 0, 'u' },
+        // Should we update the PDF?
+        { "update-pdf", no_argument, 0, 'p' },
+        { 0, 0, 0, 0 }
+    };
     int opt;
     while (
       (opt = getopt_long(argc, argv, "i:f:o:d:u:p:", long_opts, &option_index))
@@ -131,19 +140,18 @@ void cohfar_get_stats_from_file(gchar **in_fnames,
                                 TriggerStatsXML *stats_out,
                                 int *hist_trials) {
     gchar **ifname;
-    int icombo;
+    int ifo;
     for (ifname = in_fnames; *ifname; ifname++) {
 #ifdef __DEBUG__
         printf("%s\n", *ifname);
 #endif
         trigger_stats_xml_from_xml(stats_in, hist_trials, *ifname);
-        for (icombo = 0; icombo < stats_in->ncombo; icombo++) {
-            trigger_stats_feature_rate_add(
-              stats_out->multistats[icombo]->feature,
-              stats_in->multistats[icombo]->feature,
-              stats_out->multistats[icombo]);
+        for (ifo = 0; ifo <= __builtin_popcount(stats_in->icombo + 1); ifo++) {
+            trigger_stats_feature_rate_add(stats_out->multistats[ifo]->feature,
+                                           stats_in->multistats[ifo]->feature,
+                                           stats_out->multistats[ifo]);
             trigger_stats_livetime_add(stats_out->multistats,
-                                       stats_in->multistats, icombo);
+                                       stats_in->multistats, ifo);
         }
     }
 }
@@ -157,7 +165,7 @@ static int get_type(gchar **ptype) {
 static int process_stats_full(
   gchar **in_fnames, int nifo, gchar **pifos, gchar **pout, int *update_pdf) {
-    int icombo, ncombo = get_ncombo(nifo), hist_trials;
+    int ifo, hist_trials;
     TriggerStatsXML *zlstats_in =
       trigger_stats_xml_create(*pifos, STATS_XML_TYPE_ZEROLAG);
     TriggerStatsXML *zlstats_out =
@@ -180,24 +188,21 @@ static int process_stats_full(
     cohfar_get_stats_from_file(in_fnames, bgstats_in, bgstats_out,
                                &hist_trials);
     if (*update_pdf == 1) {
-        for (icombo = 0; icombo < ncombo; icombo++) {
+        for (ifo = 0; ifo < nifo; ifo++) {
             trigger_stats_feature_rate_to_pdf(
-              sgstats_out->multistats[icombo]->feature);
-            trigger_stats_feature_to_rank(
-              sgstats_out->multistats[icombo]->feature,
-              sgstats_out->multistats[icombo]->rank);
+              sgstats_out->multistats[ifo]->feature);
+            trigger_stats_feature_to_rank(sgstats_out->multistats[ifo]->feature,
+                                          sgstats_out->multistats[ifo]->rank);
 
             trigger_stats_feature_rate_to_pdf(
-              zlstats_out->multistats[icombo]->feature);
-            trigger_stats_feature_to_rank(
-              zlstats_out->multistats[icombo]->feature,
-              zlstats_out->multistats[icombo]->rank);
+              zlstats_out->multistats[ifo]->feature);
+            trigger_stats_feature_to_rank(zlstats_out->multistats[ifo]->feature,
+                                          zlstats_out->multistats[ifo]->rank);
```

```
            trigger_stats_feature_rate_to_pdf(
-             bgstats_out->multistats[icombo]->feature);
-           trigger_stats_feature_to_rank(
-             bgstats_out->multistats[icombo]->feature,
-             bgstats_out->multistats[icombo]->rank);
+             bgstats_out->multistats[ifo]->feature);
+           trigger_stats_feature_to_rank(bgstats_out->multistats[ifo]->feature,
+                                         bgstats_out->multistats[ifo]->rank);
        }
    }
@@ -234,18 +239,17 @@ static int process_stats_single(gchar **in_fnames,
                                gchar **pout,
                                int type,
                                int *update_pdf) {
-    int icombo, ncombo = get_ncombo(nifo), hist_trials;
+    int ifo, hist_trials;
    TriggerStatsXML *stats_in  = trigger_stats_xml_create(*pifos, type);
    TriggerStatsXML *stats_out = trigger_stats_xml_create(*pifos, type);
    cohfar_get_stats_from_file(in_fnames, stats_in, stats_out, &hist_trials);
    if (*update_pdf == 1) {
-        for (icombo = 0; icombo < ncombo; icombo++) {
+        for (ifo = 0; ifo < nifo; ifo++) {
            trigger_stats_feature_rate_to_pdf(
-             stats_out->multistats[icombo]->feature);
-           trigger_stats_feature_to_rank(
-             stats_out->multistats[icombo]->feature,
-             stats_out->multistats[icombo]->rank);
+             stats_out->multistats[ifo]->feature);
+           trigger_stats_feature_to_rank(stats_out->multistats[ifo]->feature,
+                                         stats_out->multistats[ifo]->rank);
        }
    }
    xmlTextWriterPtr stats_writer = NULL;
@@ -288,24 +292,18 @@ int main(int argc, char *argv[]) {
          trigger_stats_xml_create(*pifos, STATS_XML_TYPE_BACKGROUND);
        TriggerStatsXML *bgstats_out =
          trigger_stats_xml_create(*pifos, STATS_XML_TYPE_BACKGROUND);
-        int ncombo = get_ncombo(nifo);
        // FIXME: hardcoded to only update the last stats
        trigger_stats_feature_rate_update_all(
-         data_dim1, data_dim2, bgstats_out->multistats[ncombo - 1]->feature,
-         bgstats_out->multistats[ncombo - 1]);
+         data_dim1, data_dim2, bgstats_out->multistats[nifo]->feature,
+         bgstats_out->multistats[nifo]);
        trigger_stats_feature_rate_to_pdf(
-         bgstats_out->multistats[ncombo - 1]->feature);
-        trigger_stats_feature_to_rank(
-         bgstats_out->multistats[ncombo - 1]->feature,
-         bgstats_out->multistats[ncombo - 1]->rank);
+         bgstats_out->multistats[nifo]->feature);
+        trigger_stats_feature_to_rank(bgstats_out->multistats[nifo]->feature,
+                                      bgstats_out->multistats[nifo]->rank);
        if (data_dim1) {
            free(data_dim1);
            free(data_dim2);
        }
-
-        // trigger_stats_pdf_from_data(data_dim1, data_dim2,
-        // stats_out[ncombo-1]->rate->lgsnr_bins,
-        // stats_out[ncombo-1]->rate->lgchisq_bins, stats_out[ncombo-1]->pdf);
    } else if (g_strcmp0(*pfmt, "stats") == 0) {
        if (type == STATS_XML_TYPE_ALL) {
            rc = process_stats_full(in_fnames, nifo, pifos, pout, update_pdf);
```

GitLab

A.2 Removing hard-coded detector names

From 8ff7b39c3d8d3db77006e3690a04631de69df752 Mon Sep 17 00:00:00 2001

```
From: Tom Almeida <tommoa256@gmail.com>
Date: Sun, 6 Sep 2020 12:15:59 +0800
Subject: [PATCH] postcohtable: Remove detector names from table

This change removes detector names from the various data structures that
are used throughout the pipeline and ensures that all pointers into the
data structures have their offsets based upon the number of
interferometers.

Previously, adding support for an additional interferometer to the
pipeline would involve modifying all of the below files, and would often
require extensive testing to ensure that the massive number of changes
that would need to be done would be correct.

With these changes, when adding interferometers, the only file that
needs to be changed is `gstlal-spiir/include/pipe_macro.h`, with the
modification of `MAX_NIFO`, `IFOMap` and `IFOComboMap`. By changing
these variables, the next compilation of the pipeline will automatically
support the new interferometer.

--- Tom Almeida
---
 .../gst/cuda/cohfar/background_stats_utils.c  |   2 +-
 .../gst/cuda/cohfar/background_stats_utils.h  |  19 +-
 .../gst/cuda/cohfar/cohfar_accumbackground.c  |   4 +-
 .../gst/cuda/cohfar/cohfar_assignfar.c        |  86 +--
 gstlal-spiir/gst/cuda/cohfar/ssvkernel.c      |  16 +-
 gstlal-spiir/gst/cuda/cohfar/ssvkernel.h      |   2 +-
 .../gst/cuda/cohfar/test_knn_pipeline.c       |   4 +-
 gstlal-spiir/gst/cuda/cuda_plugin.c           |   2 +-
 gstlal-spiir/gst/cuda/postcoh/postcoh.c       | 107 ++--
 gstlal-spiir/gst/cuda/postcoh/postcoh.h       |  68 +--
 .../gst/cuda/postcoh/postcoh_kernel.cu        | 647 +++++++++++++++++++
 gstlal-spiir/gst/cuda/postcoh/postcoh_utils.c | 207 +++++-
 gstlal-spiir/gst/cuda/postcoh/postcoh_utils.h |   2 +-
 .../gst/cuda/postcoh/postcohtable_utils.c     | 241 +++++++
 gstlal-spiir/gst/cuda/spiir/spiir_kernel.cu   |   4 +-
 gstlal-spiir/include/pipe_macro.h             |  16 +-
 gstlal-spiir/include/postcohtable.h           |  36 +-
 .../python/pipemodules/postcoh_finalsink.py   |  35 +-
 .../pipemodules/postcohtable/Makefile.am      |   2 +-
 .../postcohtable/_postcohtable.c              | 389 ++++++++++++
 .../postcohtable/postcoh_table_def.py         | 132 ++--
 .../pipemodules/postcohtable/postcohtable.py  |   4 +-
 22 files changed, 860 insertions(+), 1165 deletions(-)

diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c b/gstlal-spiir/gst/cuda/cohfar/background_
index 449cf376..4c6f049d 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.c
@@ -52,7 +52,7 @@ int scan_trigger_ifos(int icombo, PostcohInspiralTable *trigger) {
         if (icombo & (1 << i)) {
             // [THA]: This is a check that the data from this IFO is actually
             // valid. If it's not valid, the number will be very *very* small
-            if (*(&trigger->snglsnr_H + i) > EPSILON) {
+            if (trigger->snglsnr[i] > EPSILON) {
                 strncpy(final_ifos + IFO_LEN * nifo, IFOMap[i].name,
                         one_ifo_size);
                 nifo++;
diff --git a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h b/gstlal-spiir/gst/cuda/cohfar/background_
index 9eda6a4a..abe7fc7f 100644
--- a/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h
+++ b/gstlal-spiir/gst/cuda/cohfar/background_stats_utils.h
@@ -32,6 +32,7 @@
 #include <LIGOLwHeader.h>
 #include <cohfar/background_stats.h>
 #include <glib.h>
+#include <postcohtable.h>

 Bins1D *bins1D_create_long(double cmin, double cmax, int nbin);

@@ -53,10 +54,20 @@ TriggerStats **trigger_stats_create(int icombo);

 int bins1D_get_idx(double val, Bins1D *bins);

-void trigger_stats_feature_rates_update(double snr,
-                                        double chisq,
-                                        FeatureStats *feature,
```

```
-                                        TriggerStats *cur_stats);
+void trigger_stats_feature_rate_update(double snr,
+                                       double chisq,
+                                       FeatureStats *feature,
+                                       TriggerStats *cur_stats);
+
+double trigger_stats_get_val_from_map(double snr, double chisq, Bins2D *bins);
+
+int scan_trigger_ifos(int icombo, PostcohInspiralTable *trigger);
+
+void trigger_stats_livetime_inc(TriggerStats **stats, const int index);
+
+void trigger_stats_xml_reset(TriggerStatsXML *stats);
+
+void signal_stats_init(TriggerStatsXML *sgstats, int source_type);
 void trigger_stats_feature_rates_add(FeatureStats *feature1,
                                      FeatureStats *feature2,
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_accun
index c742a658..6f8f3124 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_accumbackground.c
@@ -120,8 +120,8 @@ static void update_stats_icombo(PostcohInspiralTable *intable,
         for (ifo = 0, index = 0; ifo < MAX_NIFO; ifo++) {
             if ((stats->icombo + 1) & (1 << ifo)) {
                 trigger_stats_feature_rate_update(
-                  (double)((&intable->snglsnr_H)[ifo]),
-                  (double)((&intable->chisq_H)[ifo]),
+                  (double)(intable->snglsnr[ifo]),
+                  (double)(intable->chisq[ifo]),
                   stats->multistats[index]->feature, stats->multistats[index]);
                 ++index;
             }
diff --git a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
index 8590bf2a..cdb5648d 100644
--- a/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
+++ b/gstlal-spiir/gst/cuda/cohfar/cohfar_assignfar.c
@@ -133,72 +133,30 @@ static void update_trigger_fars(PostcohInspiralTable *table,
                                                 cur_stats->rank->rank_map);
     table->rank = MAX(MAX(rank_1w, rank_1d), rank_2h);

-    /* FIXME: currently hardcoded for single detectors FAR */
-    cur_stats = element->bgstats_1w->multistats[0];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_H,
-                                                    (double)table->chisq_H, cur_stats)
-                              * cur_stats->nevent
-                              / (cur_stats->livetime * hist_trials));
-    table->far_h_1w = far;
-
-    cur_stats = element->bgstats_1w->multistats[1];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_L,
-                                                    (double)table->chisq_L, cur_stats)
-                              * cur_stats->nevent
-                              / (cur_stats->livetime * hist_trials));
-    table->far_l_1w = far;
-
-    cur_stats = element->bgstats_1w->multistats[2];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_V,
-                                                    (double)table->chisq_V, cur_stats)
-                              * cur_stats->nevent
-                              / (cur_stats->livetime * hist_trials));
-    table->far_v_1w = far;
-
-    cur_stats = element->bgstats_1d->multistats[0];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_H,
-                                                    (double)table->chisq_H, cur_stats)
-                              * cur_stats->nevent
-                              / (cur_stats->livetime * hist_trials));
-    table->far_h_1d = far;
-
-    cur_stats = element->bgstats_1d->multistats[1];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_L,
-                                                    (double)table->chisq_L, cur_stats)
-                              * cur_stats->nevent
-                              / (cur_stats->livetime * hist_trials));
```

```
-    table->far_l_1d = far;
-
-    cur_stats = element->bgstats_1d->multistats[2];
-    far       = BOUND(FLT_MIN, gen_fap_from_feature((double)table->snglsnr_V,
-                                                    (double)table->chisq_V, cur_stats)
-                           * cur_stats->nevent
-                           / (cur_stats->livetime * hist_trials));
-    table->far_v_1d = far;
-
-    cur_stats = element->bgstats_2h->multistats[0];
-    if (cur_stats->livetime > 0) {
-        far = BOUND(
-          FLT_MIN, gen_fap_from_feature((double)table->snglsnr_H,
-                                        (double)table->chisq_H, cur_stats)
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        cur_stats = element->bgstats_1w->multistats[i];
+        far       = BOUND(
+          FLT_MIN, gen_fap_from_feature((double)table->snglsnr[i],
+                                        (double)table->chisq[i], cur_stats)
                       * cur_stats->nevent / (cur_stats->livetime * hist_trials));
-        table->far_h_2h = far;
-    }
-    cur_stats = element->bgstats_2h->multistats[1];
-    if (cur_stats->livetime > 0) {
-        far = BOUND(
-          FLT_MIN, gen_fap_from_feature((double)table->snglsnr_L,
-                                        (double)table->chisq_L, cur_stats)
-                      * cur_stats->nevent / (cur_stats->livetime * hist_trials));
-        table->far_l_2h = far;
-    }
-    cur_stats = element->bgstats_2h->multistats[2];
-    if (cur_stats->livetime > 0) {
-        far = BOUND(
-          FLT_MIN, gen_fap_from_feature((double)table->snglsnr_V,
-                                        (double)table->chisq_V, cur_stats)
+        table->far_1w_sngl[i] = far;
+
+        cur_stats = element->bgstats_1d->multistats[i];
+        far       = BOUND(
+          FLT_MIN, gen_fap_from_feature((double)table->snglsnr[i],
+                                        (double)table->chisq[i], cur_stats)
                       * cur_stats->nevent / (cur_stats->livetime * hist_trials));
-        table->far_v_2h = far;
+        table->far_1d_sngl[i] = far;
+
+        cur_stats = element->bgstats_2h->multistats[i];
+        if (cur_stats->livetime > 0) {
+            far                   = BOUND(FLT_MIN,
+                        gen_fap_from_feature((double)table->snglsnr[i],
+                                             (double)table->chisq[i], cur_stats)
+                          * cur_stats->nevent
+                          / (cur_stats->livetime * hist_trials));
+            table->far_2h_sngl[i] = far;
+        }
     }
 }
diff --git a/gstlal-spiir/gst/cuda/cohfar/ssvkernel.c b/gstlal-spiir/gst/cuda/cohfar/ssvkernel.c
index e9c3a6a0..1b53804f 100644
--- a/gstlal-spiir/gst/cuda/cohfar/ssvkernel.c
+++ b/gstlal-spiir/gst/cuda/cohfar/ssvkernel.c
@@ -39,7 +39,7 @@
 // y_hist_result: one-dimensional histogram
 // result: a L*L matrix. result(i,j) =
 // count(Bin_i)/h(x_j)*K[x-point(Bin_i)/h(x_j)] (each variable's meaning: see
-//document of ''Shimazakis method' part.).
+// document of ''Shimazakis method' part.).
 //
 // int main()
@@ -282,7 +282,8 @@ double CostFunction(gsl_vector *y_hist,
     // for (i = 0; i < y_hist->size; i++) {
     //         if (gsl_vector_get(y_hist, i) != 0) {
```

```
     //                 gsl_vector_set(y_hist_nz, idx, gsl_vector_get(y_hist,
-    //i));                    gsl_vector_set(t_nz, idx, gsl_vector_get(t, i));                                    idx++
+    // i));                   gsl_vector_set(t_nz, idx, gsl_vector_get(t, i));
+    // idx++;
     //         }
     // }
     //
@@ -905,8 +906,8 @@ void ssvkernel(gsl_vector *x,
     //         gsl_vector * xb = gsl_vector_alloc(Nb);
     //         for (j = 0; j < Nb; j++) {
     //                 gsl_vector_set(xb, j,
-    //                                gsl_vector_get(x_ab, floor(gsl_ran_flat(r, 0,
-    //N))));
+    //                                gsl_vector_get(x_ab, floor(gsl_ran_flat(r,
+    //0, N))));
     //         }
     //
     //         gsl_vector_histc(xb, t_dt2, y_histb);
@@ -923,7 +924,8 @@ void ssvkernel(gsl_vector *x,
     //         for (j = 0; j < tin->size; j++) {
     //                 gsl_matrix_set(yb, i, j,
     //                                gsl_interp_eval(linear, t->data,
-    //yb_buf->data,                                                   gsl_vector_get(tin, j), acc));
+    // yb_buf->data,                                                  gsl_vector_get(tin, j),
+    // acc));
     //         }
     //         gsl_vector_free(xb);
     // }
@@ -934,8 +936,8 @@ void ssvkernel(gsl_vector *x,
     //         gsl_sort_vector(yb_col);
     //         gsl_vector_set(lower_bound, i,
     //                        gsl_vector_get(yb_col, floor((1 - confidence) *
-    //nbs)));          gsl_vector_set(upper_bound, i,                          gsl_vector_get(yb_col,
-    //floor((confidence) * nbs)));
+    // nbs)));                gsl_vector_set(upper_bound, i,
+    // gsl_vector_get(yb_col, floor((confidence) * nbs)));
     // }
     // gsl_interp_init(linear, t->data, yv->data, t->size);
     // for (i = 0; i < tin->size; i++) {
diff --git a/gstlal-spiir/gst/cuda/cohfar/ssvkernel.h b/gstlal-spiir/gst/cuda/cohfar/ssvkernel.h
index b1beda94..71d0ae82 100644
--- a/gstlal-spiir/gst/cuda/cohfar/ssvkernel.h
+++ b/gstlal-spiir/gst/cuda/cohfar/ssvkernel.h
@@ -19,7 +19,7 @@
 // y_hist_result: one-dimensional histogram
 // result: a L*L matrix. result(i,j) =
 // count(Bin_i)/h(x_j)*K[x-point(Bin_i)/h(x_j)] (each variable's meaning: see
-//document of ''Shimazakis method' part.).
+// document of ''Shimazakis method' part.).
 //
 // int main()
diff --git a/gstlal-spiir/gst/cuda/cohfar/test_knn_pipeline.c b/gstlal-spiir/gst/cuda/cohfar/test_knn_pipeline
index d42e1ee5..2432b025 100644
--- a/gstlal-spiir/gst/cuda/cohfar/test_knn_pipeline.c
+++ b/gstlal-spiir/gst/cuda/cohfar/test_knn_pipeline.c
@@ -144,7 +144,7 @@ void getTemp(int *tempPtr, float *histgt0Ptr) {
     }
     // tempPtr = {2 5 7 9 11 15 16 18 20 22 29 31 35}
     // histgt0Ptr = {0.0714 0.0714 0.0714 0.0714 0.0714 0.0714 0.0714 0.0714
-    //0.0714 0.0714 0.0714 0.0714 0.1429 }
+    // 0.0714 0.0714 0.0714 0.0714 0.1429 }
 }
 void getHistaxis(int *tempPtr, int **histaxisPtr) {
@@ -217,7 +217,7 @@ void getHwidth(float *kthDistPtr, float GlobalHband, float *hwidthPtr) {
     // For K=2
     // kthDistPtr =
     //{1.4142 1.0000 1.4142 1.0000 1.0000 1.0000 1.0000 2.0000 1.4142 1.0000 1.0000
-    //2.2361 1.0000}
+    // 2.2361 1.0000}
```

```
 }
 void getPDF(float *hwidthPtr, float *histgt0Ptr, int **histaxisPtr) {
diff --git a/gstlal-spiir/gst/cuda/cuda_plugin.c b/gstlal-spiir/gst/cuda/cuda_plugin.c
index 6b19b0b1..bb3cc412 100644
--- a/gstlal-spiir/gst/cuda/cuda_plugin.c
+++ b/gstlal-spiir/gst/cuda/cuda_plugin.c
@@ -61,7 +61,7 @@ static gboolean plugin_init(GstPlugin *plugin) {
         { "cuda_iirbank", CUDA_IIRBANK_TYPE },
         //              {"cuda_audioresample", CUDA_AUDIO_RESAMPLE_TYPE},
         //              {"gstlal_multidownsample",
-        //GSTLAL_MULTI_DOWNSAMPLE_TYPE},
+        // GSTLAL_MULTI_DOWNSAMPLE_TYPE},
         { "cuda_multiratespiir", CUDA_TYPE_MULTIRATESPIIR },
         { "cuda_postcoh", CUDA_TYPE_POSTCOH },
         { "postcoh_filesink", POSTCOH_TYPE_FILESINK },
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh.c b/gstlal-spiir/gst/cuda/postcoh/postcoh.c
index 85f071ff..d51bff60 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh.c
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh.c
@@ -73,8 +73,8 @@ static gboolean need_flag_gap(GstPostcohCollectData *data,
     for (i = 0; i < flag_segments->len; i++) {
         this_segment = &((FlagSegment *)flag_segments->data)[i];
         /*              | start                                 | stop
-         *                                                                  | this_start
-         *(1) | s | e (2)
+         *                                                                  |
+         *this_start (1) | s | e (2)
          * | s                                                  | e
          * | s          | e
          *            |s | e
@@ -1144,10 +1144,11 @@ static int cuda_postcoh_select_background(PeakList *pklist,
         for (itrial = 1; itrial <= hist_trials; itrial++) {
             background_cur = (itrial - 1) * max_npeak + peak_cur;
             // FIXME: consider a different threshold for 3-detector
-            //                  if (sqrt(pklist->cohsnr_bg[background_cur]) > cohsnr_thresh
+            //                  if (sqrt(pklist->cohsnr_bg[background_cur]) >
+            // cohsnr_thresh
             //* pklist->snglsnr_H[iifo*max_npeak + peak_cur])
             if (sqrt(pklist->cohsnr_bg[background_cur])
-                > 1.414 + pklist->snglsnr_H[write_ifo * max_npeak + peak_cur]) {
+                > 1.414 + pklist->snglsnr[write_ifo][peak_cur]) {
                 left_backgrounds++;
                 GST_LOG("mark back,%d ipeak, %d itrial", ipeak, itrial);
             } else
@@ -1207,9 +1208,7 @@ static int cuda_postcoh_select_foreground(PostcohState *state,
             peak_cur = peak_pos[ipeak];
             // FIXME: consider a different threshold for 3-detector
             if (sqrt(pklist->cohsnr[peak_cur])
-                > 1.414
-                    + pklist->snglsnr_H[write_ifo * (state->max_npeak)
-                                        + peak_cur]) {
+                > 1.414 + pklist->snglsnr[write_ifo][peak_cur]) {
                 cluster_peak_pos[final_peaks++] = peak_cur;
             } else
                 bubbled_peak_pos[bubbled_peaks++] = peak_cur;
@@ -1290,33 +1289,22 @@ static int cuda_postcoh_write_table_to_buf(CudaPostcoh *postcoh,
             len_cur = pklist->len_idx[peak_cur];
             XLALGPSAdd(&(end_time), (double)len_cur / exe_len);
             output->end_time = end_time;
-            XLALGPSAdd(&(end_time),
-                       (double)pklist->ntoff_H[peak_cur] / exe_len);
-            output->end_time_H = end_time;
-            end_time           = output->end_time;
-            XLALGPSAdd(&(end_time),
-                       (double)pklist->ntoff_L[peak_cur] / exe_len);
-            output->end_time_L = end_time;
-            end_time           = output->end_time;
-            XLALGPSAdd(&(end_time),
-                       (double)pklist->ntoff_V[peak_cur] / exe_len);
-            output->end_time_V = end_time;
-            output->snglsnr_H  = pklist->snglsnr_H[peak_cur];
```

```
-               output->snglsnr_L  = pklist->snglsnr_L[peak_cur];
-               output->snglsnr_V  = pklist->snglsnr_V[peak_cur];
-               output->coaphase_H = pklist->coaphase_H[peak_cur];
-               output->coaphase_L = pklist->coaphase_L[peak_cur];
-               output->coaphase_V = pklist->coaphase_V[peak_cur];
-               output->chisq_H    = pklist->chisq_H[peak_cur];
-               output->chisq_L    = pklist->chisq_L[peak_cur];
-               output->chisq_V    = pklist->chisq_V[peak_cur];
+               for (int i = 0; i < MAX_NIFO; ++i) {
+                   XLALGPSAdd(&(end_time),
+                              (double)pklist->ntoff[i][peak_cur] / exe_len);
+                   output->end_time_sngl[i] = end_time;
+                   end_time                 = output->end_time;
+
+                   output->snglsnr[i]  = pklist->snglsnr[i][peak_cur];
+                   output->coaphase[i] = pklist->coaphase[i][peak_cur];
+                   output->chisq[i]    = pklist->chisq[i][peak_cur];
+               }
                for (jifo = 0; jifo < nifo; jifo++) {
                    int write_ifo = state->write_ifo_mapping[jifo];
-                   *(&output->deff_H + write_ifo) =
+                   output->deff[write_ifo] =
                      sqrt(state->sigmasq[jifo][cur_tmplt_idx])
-                     / *(pklist->snglsnr_H + write_ifo * state->max_npeak
-                         + peak_cur); // in MPC
+                     / pklist->snglsnr[write_ifo][peak_cur]; // in MPC
                }
                output->is_background = FLAG_FOREGROUND;
                output->livetime      = livetime;
@@ -1384,20 +1372,20 @@ static int cuda_postcoh_write_table_to_buf(CudaPostcoh *postcoh,
                    output->skymap_fname[0] = '\0';
                output->rank = 0;
-               GST_LOG_OBJECT(postcoh,
-                              "end_time_L %d, ipeak %d, peak_cur %d, len_cur %d, "
-                              "tmplt_idx %d, pix_idx %d \t,"
-                              "snglsnr_L %f, snglsnr_H %f, snglsnr_V %f,"
-                              "coaphase_L %f, coaphase_H %f, coa_phase_V %f,"
-                              "chisq_L %f, chisq_H %f, chisq_V %f,"
-                              "cohsnr %f, nullsnr %f, cmbchisq %f\n",
-                              output->end_time_L.gpsSeconds, ipeak, peak_cur,
-                              len_cur, output->tmplt_idx, output->pix_idx,
-                              output->snglsnr_L, output->snglsnr_H,
-                              output->snglsnr_V, output->coaphase_L,
-                              output->coaphase_H, output->coaphase_V,
-                              output->chisq_L, output->chisq_H, output->chisq_V,
-                              output->cohsnr, output->nullsnr, output->cmbchisq);
+               GST_LOG_OBJECT(
+                 postcoh,
+                 "end_time_sngl_0 %d, ipeak %d, peak_cur %d, len_cur %d, "
+                 "tmplt_idx %d, pix_idx %d \t,"
+                 "snglsnr_0 %f, snglsnr_1 %f, snglsnr_2 %f,"
+                 "coaphase_0 %f, coaphase_1 %f, coa_phase_2 %f,"
+                 "chisq_0 %f, chisq_1 %f, chisq_2 %f,"
+                 "cohsnr %f, nullsnr %f, cmbchisq %f\n",
+                 output->end_time_sngl[0].gpsSeconds, ipeak, peak_cur, len_cur,
+                 output->tmplt_idx, output->pix_idx, output->snglsnr[0],
+                 output->snglsnr[1], output->snglsnr[2], output->coaphase[0],
+                 output->coaphase[1], output->coaphase[2], output->chisq[0],
+                 output->chisq[1], output->chisq[2], output->cohsnr,
+                 output->nullsnr, output->cmbchisq);
                XLALINT8NSToGPS(&output->epoch, ts);
                output->deltaT = 1. / postcoh->rate;
@@ -1424,15 +1412,12 @@ static int cuda_postcoh_write_table_to_buf(CudaPostcoh *postcoh,
                            state->all_ifos + IFO_LEN * iifo, one_ifo_size);
                    output->pivotal_ifo[IFO_LEN] = '\0';
                    output->tmplt_idx            = pklist->tmplt_idx[peak_cur];
-                   output->snglsnr_H  = pklist->snglsnr_bg_H[peak_cur_bg];
-                   output->snglsnr_L  = pklist->snglsnr_bg_L[peak_cur_bg];
-                   output->snglsnr_V  = pklist->snglsnr_bg_V[peak_cur_bg];
-                   output->coaphase_H = pklist->coaphase_bg_H[peak_cur_bg];
```

```
-                       output->coaphase_L = pklist->coaphase_bg_L[peak_cur_bg];
-                       output->coaphase_V = pklist->coaphase_bg_V[peak_cur_bg];
-                       output->chisq_H    = pklist->chisq_bg_H[peak_cur_bg];
-                       output->chisq_L    = pklist->chisq_bg_L[peak_cur_bg];
-                       output->chisq_V    = pklist->chisq_bg_V[peak_cur_bg];
+                       for (int i = 0; i < MAX_NIFO; ++i) {
+                           output->snglsnr[i] = pklist->snglsnr_bg[i][peak_cur_bg];
+                           output->coaphase[i] =
+                             pklist->coaphase_bg[i][peak_cur_bg];
+                           output->chisq[i] = pklist->chisq_bg[i][peak_cur_bg];
+                       }
                        // output->pix_idx = pklist->pix_idx[itrial*max_npeak +
                        // peak_cur];
@@ -1448,16 +1433,16 @@ static int cuda_postcoh_write_table_to_buf(CudaPostcoh *postcoh,
                          postcoh,
                          "ipeak %d, itrial %d, len_cur %d, tmplt_idx %d, pix_idx "
                          "%d,"
-                         "snglsnr_L %f, snglsnr_H %f, snglsnr_V %f,"
-                         "coaphase_L %f, coaphase_H %f, coa_phase_V %f,"
-                         "chisq_L %f, chisq_H %f, chisq_V %f,"
+                         "snglsnr[0] %f, snglsnr[1] %f, snglsnr[2] %f,"
+                         "coaphase[0] %f, coaphase[1] %f, coa_phase[2] %f,"
+                         "chisq[0] %f, chisq[1] %f, chisq[2] %f,"
                          "cohsnr %f, nullsnr %f, cmbchisq %f\n",
                          ipeak, itrial, len_cur, output->tmplt_idx,
-                         output->pix_idx, output->snglsnr_L, output->snglsnr_H,
-                         output->snglsnr_V, output->coaphase_L, output->coaphase_H,
-                         output->coaphase_V, output->chisq_L, output->chisq_H,
-                         output->chisq_V, output->cohsnr, output->nullsnr,
-                         output->cmbchisq);
+                         output->pix_idx, output->snglsnr[0], output->snglsnr[2],
+                         output->snglsnr[2], output->coaphase[0],
+                         output->coaphase[1], output->coaphase[2],
+                         output->chisq[0], output->chisq[1], output->chisq[2],
+                         output->cohsnr, output->nullsnr, output->cmbchisq);
                        /* do not dump snr for background */
                        XLALINT8NSToGPS(&output->epoch, ts);
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh.h b/gstlal-spiir/gst/cuda/postcoh/postcoh.h
index 0845168e..32d58dbf 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh.h
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh.h
@@ -88,29 +88,15 @@ typedef struct _PeakList {
     int *tmplt_idx;
     int *pix_idx;
     int *pix_idx_bg; // background Ntoff needs this, do not remove
-    int *ntoff_H;
-    int *ntoff_L;
-    int *ntoff_V;
-
-    float *snglsnr_H;
-    float *snglsnr_L;
-    float *snglsnr_V;
-    float *coaphase_H;
-    float *coaphase_L;
-    float *coaphase_V;
-    float *chisq_H;
-    float *chisq_L;
-    float *chisq_V;
-
-    float *snglsnr_bg_H;
-    float *snglsnr_bg_L;
-    float *snglsnr_bg_V;
-    float *coaphase_bg_H;
-    float *coaphase_bg_L;
-    float *coaphase_bg_V;
-    float *chisq_bg_H;
-    float *chisq_bg_L;
-    float *chisq_bg_V;
+    int *ntoff[MAX_NIFO];
+
+    float *snglsnr[MAX_NIFO];
+    float *coaphase[MAX_NIFO];
+    float *chisq[MAX_NIFO];
```

```
+
+    float *snglsnr_bg[MAX_NIFO];
+    float *coaphase_bg[MAX_NIFO];
+    float *chisq_bg[MAX_NIFO];
     float *cohsnr;
     float *nullsnr;
@@ -124,35 +110,25 @@ typedef struct _PeakList {
     float *nullsnr_skymap;
     /* structure on GPU device */
+    // [THA]: It is important to note that pointers on the host device are not
+    // exposed to the GPU device. For this reason, we can't allocate d_ntoff,
+    // d_snglsnr, etc. here on the stack with sized arrays. Instead, we need
+    // to malloc is when PeakList is built.
     int *d_npeak;
     int *d_peak_pos;
     int *d_len_idx;
     int *d_tmplt_idx;
     int *d_pix_idx;
     int *d_pix_idx_bg; // background Ntoff needs this, do not remove
-    int *d_ntoff_H;
-    int *d_ntoff_L;
-    int *d_ntoff_V;
-
-    float *d_snglsnr_H;
-    float *d_snglsnr_L;
-    float *d_snglsnr_V;
-    float *d_coaphase_H;
-    float *d_coaphase_L;
-    float *d_coaphase_V;
-    float *d_chisq_H;
-    float *d_chisq_L;
-    float *d_chisq_V;
-
-    float *d_snglsnr_bg_H;
-    float *d_snglsnr_bg_L;
-    float *d_snglsnr_bg_V;
-    float *d_coaphase_bg_H;
-    float *d_coaphase_bg_L;
-    float *d_coaphase_bg_V;
-    float *d_chisq_bg_H;
-    float *d_chisq_bg_L;
-    float *d_chisq_bg_V;
+    int **d_ntoff; // size (MAX_NIFO)
+
+    float **d_snglsnr; // size (MAX_NIFO)
+    float **d_coaphase; // size (MAX_NIFO)
+    float **d_chisq; // size (MAX_NIFO)
+
+    float **d_snglsnr_bg; // size (MAX_NIFO)
+    float **d_coaphase_bg; // size (MAX_NIFO)
+    float **d_chisq_bg; // size (MAX_NIFO)
     float *d_cohsnr;
     float *d_nullsnr;
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh_kernel.cu b/gstlal-spiir/gst/cuda/postcoh/postcoh_kernel.cu
index 8c6c6d9a..513a226e 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh_kernel.cu
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh_kernel.cu
@@ -190,8 +190,8 @@ __global__ void
             peak_pos[index] = i;
             /* FIXME: could be many peak positions for one peak */
             //                  peak[templ] = -1;
-            //                  printf("peak tmplt %d, time %d, maxsnr %f, snr %f\n", templ,
-            //i, max_snr, snr);
+            //                  printf("peak tmplt %d, time %d, maxsnr %f, snr
+            //%f\n", templ, i, max_snr, snr);
         }
         // restemplate[i]  = ((-1 + templ) + (-1 - templ) * ((max_snr > snr) * 2
         // - 1)) >> 1; ressnr[i]       = (-1.0f + snr) * 0.5 + (-1.0f - snr) *
@@ -222,24 +222,27 @@ __device__ float gser(float x, float a) {
 }
 __global__ void ker_coh_skymap(
-    float
```

```
-    *cohsnr_skymap, /* OUTPUT, of size (num_triggers * num_sky_directions) */
-   float
-    *nullsnr_skymap, /* OUTPUT, of size (num_triggers * num_sky_directions) */
-  COMPLEX_F **snr, /* INPUT, (2, 3) * data_points */
+   float *restrict
+    cohsnr_skymap, /* OUTPUT, of size (num_triggers * num_sky_directions) */
+   float *restrict
+    nullsnr_skymap, /* OUTPUT, of size (num_triggers * num_sky_directions) */
+  COMPLEX_F *restrict *restrict snr, /* INPUT, (2, 3) * data_points */
    int iifo, /* INPUT, detector we are considering */
    int nifo, /* INPUT, all detectors that are in this coherent analysis */
-   int *peak_pos, /* INPUT, place the location of the trigger */
-   float *snglsnr_H, /* INPUT, maximum single snr    */
+   int *restrict peak_pos, /* INPUT, place the location of the trigger */
+   float *restrict *restrict snglsnr, /* INPUT, maximum single snr    */
+   float *restrict cohsnr, /* INPUT, coherent snr */
    int npeak, /* INPUT, number of triggers */
-   float *u_map, /* INPUT, u matrix map */
-   float *toa_diff_map, /* INPUT, time of arrival difference map */
+   float *restrict u_map, /* INPUT, u matrix map */
+   float *restrict toa_diff_map, /* INPUT, time of arrival difference map */
    int num_sky_directions, /* INPUT, # of sky directions */
    int max_npeak, /* INPUT, max number of peaks */
    int len, /* INPUT, snglsnr length */
    int start_exe, /* INPUT, snglsnr start exe position */
    float dt, /* INPUT, 1/ sampling rate */
-   int ntmplt /* INPUT, number of templates */
+   int ntmplt, /* INPUT, number of templates */
+   int *restrict len_idx, /* INPUT, the length of the peaks */
+   int *restrict tmplt_idx /* INPUT, the template used for this peak */
 ) {
     int peak_cur, tmplt_cur, len_cur, ipeak_max;
@@ -248,7 +251,6 @@ __global__ void ker_coh_skymap(
     int map_idx, ipix, i;
     float real, imag;
     float snr_tmp, al_all = 0.0f;
-    float *cohsnr = snglsnr_H + 9 * max_npeak;
     extern __shared__ float smem[];

     /* find maximum cohsnr and swope to the first pos */
@@ -304,9 +306,9 @@ __global__ void ker_coh_skymap(
     if (npeak > 0) {
         peak_cur = peak_pos[0];
         // find the len_cur from len_idx
-        len_cur = peak_pos[peak_cur + max_npeak];
+        len_cur = len_idx[peak_cur];
         // find the tmplt_cur from tmplt_idx
-        tmplt_cur = peak_pos[peak_cur + 2 * max_npeak];
+        tmplt_cur = tmplt_idx[peak_cur];

         for (int seed_pix = threadIdx.x; seed_pix < num_sky_directions;
              seed_pix += blockDim.x) {
@@ -348,23 +350,19 @@ __global__ void ker_coh_skymap(
 }

 __global__ void ker_coh_max_and_chisq_versatile(
-  COMPLEX_F **snr, /* INPUT, (2, 3) * data_points */
+  COMPLEX_F *restrict *restrict snr, /* INPUT, (2, 3) * data_points */
    int iifo, /* INPUT, detector we are considering */
    int nifo, /* INPUT, all detectors that are in this coherent analysis */
    int cur_nifo, /* INPUT, all detectors that are in this coherent analysis */
    int
     cur_ifo_bits, /* INPUT, all detectors that are in this coherent analysis */
-   int *write_ifo_mapping, /* INPUT, write-ifo-mapping */
-   int *peak_pos, /* INPUT, place the location of the trigger */
-   float *snglsnr_H, /* INPUT, maximum single snr    */
-   float *snglsnr_bg_H, /* INPUT, maximum single snr    */
-   float *cohsnr_skymap,
-   float *nullsnr_skymap,
-   int output_skymap, /* INPUT, whether to write to cohsnr_skymap and
-                         nullsnr_skymap */
+   int *restrict write_ifo_mapping, /* INPUT, write-ifo-mapping */
```

```
+  int *restrict peak_pos, /* INPUT, place the location of the trigger */
+  float *restrict *restrict snglsnr, /* INPUT, maximum single snr    */
+  float *restrict *restrict snglsnr_bg, /* INPUT, maximum single snr    */
   int npeak, /* INPUT, number of triggers */
-  float *u_map, /* INPUT, u matrix map */
-  float *toa_diff_map, /* INPUT, time of arrival difference map */
+  float *restrict u_map, /* INPUT, u matrix map */
+  float *restrict toa_diff_map, /* INPUT, time of arrival difference map */
   int num_sky_directions, /* INPUT, # of sky directions */
   int len, /* INPUT, snglsnr length */
   int max_npeak, /* INPUT, snglsnr length */
@@ -372,13 +370,32 @@ __global__ void ker_coh_max_and_chisq_versatile(
   float dt, /* INPUT, 1/ sampling rate */
   int ntmplt, /* INPUT, number of templates */
   int autochisq_len, /* INPUT, auto-chisq length */
-  COMPLEX_F *
-    *autocorr_matrix, /* INPUT, autocorrelation matrix for all templates */
-  float **autocorr_norm, /* INPUT, autocorrelation normalization matrix for all
-                            templates */
+  COMPLEX_F *restrict *restrict
+    autocorr_matrix, /* INPUT, autocorrelation matrix for all templates */
+  float *restrict *restrict autocorr_norm, /* INPUT, autocorrelation
+                            normalization matrix for all templates */
   int hist_trials, /* INPUT, trial number */
-  int trial_sample_inv /* INPUT, trial interval in samples */
-) {
+  int trial_sample_inv, /* INPUT, trial interval in samples */
+  int *restrict pix_idx, /* OUTPUT, sky direction index */
+  int *restrict pix_idx_bg, /* OUTPUT, sky direction index for the background */
+  float *restrict
+    cohsnr, /* OUTPUT, the coherent SNR for combination of detectors */
+  float *restrict
+    nullsnr, /* OUTPUT, the nullsnr for the combination of detectors */
+  float *restrict
+    cmbchisq, /* OUTPUT, the chisq for the combination of detectors */
+  float *restrict
+    cohsnr_bg, /* OUTPUT, the coherent SNR for the background noise */
+  float *restrict nullsnr_bg, /* OUTPUT, the nullsnr for the background noise */
+  float *restrict
+    cmbchisq_bg, /* OUTPUT, the combined chisq for the background noise */
+  float *restrict *restrict coaphase,
+  float *restrict *restrict coaphase_bg,
+  int *restrict *restrict ntoff,
+  float *restrict *restrict chisq,
+  float *restrict *restrict chisq_bg,
+  int *restrict len_idx,
+  int *restrict tmplt_idx) {
   int bid = blockIdx.x;
   int bn  = gridDim.x;
@@ -390,10 +407,10 @@ __global__ void ker_coh_max_and_chisq_versatile(
   // store snr_max, nullstream_max and sky_idx, each has (blockDim.x /
   // WARP_SIZE) elements
   extern __shared__ float smem[];
-  volatile float *stat_shared       = &smem[0];
-  volatile float *snr_shared        = &stat_shared[wn];
-  volatile float *nullstream_shared = &snr_shared[wn];
-  volatile int *sky_idx_shared      = (int *)&nullstream_shared[wn];
+  volatile float *restrict stat_shared       = &smem[0];
+  volatile float *restrict snr_shared        = &stat_shared[wn];
+  volatile float *restrict nullstream_shared = &snr_shared[wn];
+  volatile int *restrict sky_idx_shared      = (int *)&nullstream_shared[wn];
   // float    *mu;    // matrix u for certain sky direction
   int peak_cur, tmplt_cur, ipeak_max = 0;
@@ -408,21 +425,13 @@ __global__ void ker_coh_max_and_chisq_versatile(
   float nullstream_max, nullstream_max_tmp;
   int sky_idx = 0, sky_idx_tmp = 0;
   int i, itrial, trial_offset, output_offset, len_cur;
-  int *pix_idx      = peak_pos + 3 * max_npeak;
-  int *pix_idx_bg   = peak_pos + 4 * max_npeak;
-  float *cohsnr     = snglsnr_H + 9 * max_npeak;
-  float *nullsnr    = snglsnr_H + 10 * max_npeak;
-  float *cmbchisq   = snglsnr_H + 11 * max_npeak;
```

```
-    float *cohsnr_bg   = snglsnr_H + (12 + 9 * hist_trials) * max_npeak;
-    float *nullsnr_bg  = snglsnr_H + (12 + 10 * hist_trials) * max_npeak;
-    float *cmbchisq_bg = snglsnr_H + (12 + 11 * hist_trials) * max_npeak;

     for (int ipeak = bid; ipeak < npeak; ipeak += bn) {
         peak_cur = peak_pos[ipeak];
         // find the len_cur from len_idx
-        len_cur = peak_pos[peak_cur + max_npeak];
+        len_cur = len_idx[peak_cur];
         // find the tmplt_cur from tmplt_idx
-        tmplt_cur = peak_pos[peak_cur + 2 * max_npeak];
+        tmplt_cur = tmplt_idx[peak_cur];

         itrial        = 0;
         stat_max      = 0.0;
@@ -553,14 +562,12 @@ __global__ void ker_coh_max_and_chisq_versatile(
             * does not participate */
            /* set the ntoff; snglsnr; and coa_phase for each detector */
            /* set the ntoff; this is actually d_ntoff_*  */
-           peak_pos[peak_cur
-                    + (4 + hist_trials + write_ifo_mapping[j]) * max_npeak] =
-             NtOff;
+           ntoff[write_ifo_mapping[j]][peak_cur] = NtOff;
            /* set the d_snglsnr_* */
-           snglsnr_H[peak_cur + write_ifo_mapping[j] * max_npeak] = sqrt(
+           snglsnr[write_ifo_mapping[j]][peak_cur] = sqrt(
              tmp_maxsnr.re * tmp_maxsnr.re + tmp_maxsnr.im * tmp_maxsnr.im);
            /* set the d_coa_phase_* */
-           snglsnr_H[peak_cur + (3 + write_ifo_mapping[j]) * max_npeak] =
+           coaphase[write_ifo_mapping[j]][peak_cur] =
              atan2(tmp_maxsnr.im, tmp_maxsnr.re);

            for (int ishift = threadIdx.x - autochisq_half_len;
@@ -589,15 +596,14 @@ __global__ void ker_coh_max_and_chisq_versatile(
                if (srcLane == 0) {
                    chisq_cur = laneChi2 / autocorr_norm[j][tmplt_cur];
                    // the location of chisq_* is indexed from maxsnglsnr
-                   snglsnr_H[peak_cur
-                             + (6 + write_ifo_mapping[j]) * max_npeak] =
-                     chisq_cur;
+                   chisq[write_ifo_mapping[j]][peak_cur] = chisq_cur;

                    if (((1 << j) & cur_ifo_bits) > 0)
                        cmbchisq[peak_cur] += chisq_cur;
                    // printf("peak %d, itrial %d, cohsnr %f, nullstream %f,
                    // ipix %d, chisq %f\n", ipeak, itrial, cohsnr[peak_cur],
-                   // nullsnr[peak_cur], pix_idx[peak_cur], cmbchisq[peak_cur]);
+                   // nullsnr[peak_cur], pix_idx[peak_cur],
+                   // cmbchisq[peak_cur]);
                }
            }
            __syncthreads();
@@ -752,14 +758,11 @@ __global__ void ker_coh_max_and_chisq_versatile(
                tmp_maxsnr =
                  snr[j][len * tmplt_cur + ((peak_pos_tmp + len) % len)];
                /* set the d_snglsnr_* */
-               snglsnr_bg_H[output_offset
-                            + write_ifo_mapping[j] * hist_trials * max_npeak] =
+               snglsnr_bg[write_ifo_mapping[j]][output_offset] =
                  sqrt(tmp_maxsnr.re * tmp_maxsnr.re
                       + tmp_maxsnr.im * tmp_maxsnr.im);
                /* set the d_coa_phase_* */
-               snglsnr_bg_H[output_offset
-                            + (3 + write_ifo_mapping[j]) * hist_trials
-                              * max_npeak] =
+               coaphase_bg[write_ifo_mapping[j]][output_offset] =
                  atan2(tmp_maxsnr.im, tmp_maxsnr.re);

                for (int ishift = srcLane - autochisq_half_len;
@@ -783,10 +786,8 @@ __global__ void ker_coh_max_and_chisq_versatile(

                if (srcLane == 0) {
                    chisq_cur = laneChi2 / autocorr_norm[j][tmplt_cur];
-                   // set d_chisq_bg_* from snglsnr_bg_H
```

```
-                       snglsnr_bg_H[output_offset
-                                    + (6 + write_ifo_mapping[j]) * hist_trials
-                                       * max_npeak] = chisq_cur;
+                       // set d_chisq_bg_* from snglsnr_bg
+                       chisq_bg[write_ifo_mapping[j]][output_offset] = chisq_cur;
                        cmbchisq_bg[output_offset] += chisq_cur;
                }
@@ -794,481 +795,16 @@ __global__ void ker_coh_max_and_chisq_versatile(
                        __syncthreads();
                }
-
-                __syncthreads();
-        }
-    }
-}
-
-__global__ void ker_coh_max_and_chisq(
-  COMPLEX_F **snr, /* INPUT, (2, 3) * data_points */
-  int iifo, /* INPUT, detector we are considering */
-  int nifo, /* INPUT, all detectors that are in this coherent analysis */
-  int *write_ifo_mapping, /* INPUT, write-ifo-mapping */
-  int *peak_pos, /* INPUT, place the location of the trigger */
-  float *snglsnr_H, /* INPUT, maximum single snr    */
-  float *snglsnr_bg_H, /* INPUT, maximum single snr    */
-  float *cohsnr_skymap,
-  float *nullsnr_skymap,
-  int output_skymap, /* INPUT, whether to write to cohsnr_skymap and
-                        nullsnr_skymap */
-  int npeak, /* INPUT, number of triggers */
-  float *u_map, /* INPUT, u matrix map */
-  float *toa_diff_map, /* INPUT, time of arrival difference map */
-  int num_sky_directions, /* INPUT, # of sky directions */
-  int len, /* INPUT, snglsnr length */
-  int max_npeak, /* INPUT, snglsnr length */
-  int start_exe, /* INPUT, snglsnr start exe position */
-  float dt, /* INPUT, 1/ sampling rate */
-  int ntmplt, /* INPUT, number of templates */
-  int autochisq_len, /* INPUT, auto-chisq length */
-  COMPLEX_F *
-    *autocorr_matrix, /* INPUT, autocorrelation matrix for all templates */
-  float **autocorr_norm, /* INPUT, autocorrelation normalization matrix for all
-                            templates */
-  int hist_trials, /* INPUT, trial number */
-  int trial_sample_inv /* INPUT, trial interval in samples */
-) {
-    int bid = blockIdx.x;
-    int bn  = gridDim.x;
-
-    int wn  = blockDim.x >> LOG_WARP_SIZE;
-    int wID = threadIdx.x >> LOG_WARP_SIZE;
-
-    int srcLane = threadIdx.x & 0x1f, ipix; // binary: 11111, decimal 31
-    // store snr_max, nullstream_max and sky_idx, each has (blockDim.x /
-    // WARP_SIZE) elements
-    extern __shared__ float smem[];
-    volatile float *stat_shared       = &smem[0];
-    volatile float *snr_shared        = &stat_shared[wn];
-    volatile float *nullstream_shared = &snr_shared[wn];
-    volatile int *sky_idx_shared      = (int *)&nullstream_shared[wn];
-
-    // float    *mu;    // matrix u for certain sky direction
-    int peak_cur, tmplt_cur, ipeak_max = 0;
-    COMPLEX_F dk[MAXIFOS];
-    int NtOff;
-    int map_idx;
-    float real, imag;
-    float al_all = 0.0f, chisq_cur;
-
-    float stat_max, stat_tmp;
-    float snr_max, snr_tmp;
-    float nullstream_max, nullstream_max_tmp;
-    int sky_idx = 0, sky_idx_tmp = 0;
```

```
-    int i, itrial, trial_offset, output_offset, len_cur;
-    int *pix_idx       = peak_pos + 3 * max_npeak;
-    int *pix_idx_bg    = peak_pos + 4 * max_npeak;
-    float *cohsnr      = snglsnr_H + 9 * max_npeak;
-    float *nullsnr     = snglsnr_H + 10 * max_npeak;
-    float *cmbchisq    = snglsnr_H + 11 * max_npeak;
-    float *cohsnr_bg   = snglsnr_H + (12 + 9 * hist_trials) * max_npeak;
-    float *nullsnr_bg  = snglsnr_H + (12 + 10 * hist_trials) * max_npeak;
-    float *cmbchisq_bg = snglsnr_H + (12 + 11 * hist_trials) * max_npeak;
-
-    for (int ipeak = bid; ipeak < npeak; ipeak += bn) {
-        peak_cur = peak_pos[ipeak];
-        // find the len_cur from len_idx
-        len_cur = peak_pos[peak_cur + max_npeak];
-        // find the tmplt_cur from tmplt_idx
-        tmplt_cur = peak_pos[peak_cur + 2 * max_npeak];
-
-        itrial         = 0;
-        stat_max       = 0.0;
-        snr_max        = 0.0;
-        nullstream_max = 0.0f;
-        sky_idx        = 0;
-
-        for (int seed_pix = threadIdx.x;
-             seed_pix < num_sky_directions / NSKY_REDUCE_RATIO;
-             seed_pix += blockDim.x) {
-            ipix = seed_pix * NSKY_REDUCE_RATIO;
-
-            snr_tmp = 0.0;
-            al_all  = 0.0;
-            // matrix u is stored in column order
-            // mu = u_map + nifo * nifo * i;
-
-            for (int j = 0; j < nifo; ++j) {
-                /* this is a simplified algorithm to get map_idx */
-                map_idx = iifo * nifo + j;
-                NtOff =
-                  round(toa_diff_map[map_idx * num_sky_directions + ipix] / dt);
-                NtOff = (j == iifo ? 0 : NtOff);
-                // dk[j] = snr[j][((start_exe + len_cur + NtOff + len) % len) *
-                // ntmplt + tmplt_cur ];
-                dk[j] = snr[j][tmplt_cur * len
-                               + ((start_exe + len_cur + NtOff + len) % len)];
-            }
-
-            for (int j = 0; j < nifo; ++j) {
-                real = 0.0f;
-                imag = 0.0f;
-                for (int k = 0; k < nifo; ++k) {
-                    // transpose of u_map
-                    real += u_map[(k * nifo + j) * num_sky_directions + ipix]
-                            * dk[k].re;
-                    imag += u_map[(k * nifo + j) * num_sky_directions + ipix]
-                            * dk[k].im;
-                }
-                (j < 2 ? snr_tmp : al_all) += real * real + imag * imag;
-            }
-
-            stat_tmp = snr_tmp - 0.0;
-            if (stat_tmp > stat_max) {
-                stat_max       = stat_tmp;
-                snr_max        = snr_tmp;
-                nullstream_max = al_all;
-                sky_idx        = ipix;
-            }
-        }
-
-        for (i = WARP_SIZE / 2; i > 0; i = i >> 1) {
-            stat_tmp           = __shfl_xor(stat_max, i);
-            snr_tmp            = __shfl_xor(snr_max, i);
-            nullstream_max_tmp = __shfl_xor(nullstream_max, i);
-            sky_idx_tmp        = __shfl_xor(sky_idx, i);
-
-            if (stat_tmp > stat_max) {
-                stat_max       = stat_tmp;
```

```
-                 snr_max         = snr_tmp;
-                 nullstream_max  = nullstream_max_tmp;
-                 sky_idx         = sky_idx_tmp;
-             }
-         }
-
-         if (srcLane == 0) {
-             stat_shared[wID]        = stat_max;
-             snr_shared[wID]         = snr_max;
-             nullstream_shared[wID]  = nullstream_max;
-             sky_idx_shared[wID]     = sky_idx;
-         }
-         __syncthreads();
-
-         for (i = wn >> 1; i > 0; i = i >> 1) {
-             if (threadIdx.x < i) {
-                 stat_tmp = stat_shared[threadIdx.x + i];
-                 stat_max = stat_shared[threadIdx.x];
-
-                 if (stat_tmp > stat_max) {
-                     stat_shared[threadIdx.x] = stat_tmp;
-                     snr_shared[threadIdx.x]  = snr_shared[threadIdx.x + i];
-                     nullstream_shared[threadIdx.x] =
-                       nullstream_shared[threadIdx.x + i];
-                     sky_idx_shared[threadIdx.x] =
-                       sky_idx_shared[threadIdx.x + i];
-                 }
-             }
-             __syncthreads();
-         }
-         if (threadIdx.x == 0) {
-             cohsnr[peak_cur]  = snr_shared[0];
-             nullsnr[peak_cur] = nullstream_shared[0];
-             pix_idx[peak_cur] = sky_idx_shared[0];
-         }
-         __syncthreads();
-
-         /* chisq calculation */
-
-         COMPLEX_F data, tmp_snr, tmp_autocorr, tmp_maxsnr;
-         float laneChi2          = 0.0f;
-         int autochisq_half_len = autochisq_len / 2, peak_pos_tmp;
-
-         cmbchisq[peak_cur] = 0.0;
-
-         for (int j = 0; j < nifo; ++j) {
-             laneChi2 = 0.0f;
-             /* this is a simplified algorithm to get map_idx */
-             map_idx = iifo * nifo + j;
-
-             NtOff = round(
-               toa_diff_map[map_idx * num_sky_directions + pix_idx[peak_cur]]
-               / dt);
-             peak_pos_tmp = start_exe + len_cur + (j == iifo ? 0 : NtOff + len);
-
-             // tmp_maxsnr = snr[j][((peak_pos_tmp + len) % len) * ntmplt +
-             // tmplt_cur];
-             tmp_maxsnr = snr[j][len * tmplt_cur + ((peak_pos_tmp + len) % len)];
-             /* set the ntoff; snglsnr; and coa_phase for each detector */
-             /* set the ntoff; this is actually d_ntoff_*  */
-             peak_pos[peak_cur
-                      + (4 + hist_trials + write_ifo_mapping[j]) * max_npeak] =
-               NtOff;
-             /* set the d_snglsnr_* */
-             snglsnr_H[peak_cur + write_ifo_mapping[j] * max_npeak] = sqrt(
-               tmp_maxsnr.re * tmp_maxsnr.re + tmp_maxsnr.im * tmp_maxsnr.im);
-             /* set the d_coa_phase_* */
-             snglsnr_H[peak_cur + (3 + write_ifo_mapping[j]) * max_npeak] =
-               atan2(tmp_maxsnr.im, tmp_maxsnr.re);
-
-             for (int ishift = threadIdx.x - autochisq_half_len;
-                  ishift <= autochisq_half_len; ishift += blockDim.x) {
-                 tmp_snr = snr[j][len * tmplt_cur
```

```
-                                   + ((peak_pos_tmp + ishift + len) % len)];
-                 tmp_autocorr =
-                   autocorr_matrix[j][tmplt_cur * autochisq_len + ishift
-                                      + autochisq_half_len];
-                 data.re = tmp_snr.re - tmp_maxsnr.re * tmp_autocorr.re
-                           + tmp_maxsnr.im * tmp_autocorr.im;
-                 data.im = tmp_snr.im - tmp_maxsnr.re * tmp_autocorr.im
-                           - tmp_maxsnr.im * tmp_autocorr.re;
-                 laneChi2 += (data.re * data.re + data.im * data.im);
-             }
-             for (int k = WARP_SIZE >> 1; k > 0; k = k >> 1) {
-                 laneChi2 += __shfl_xor(laneChi2, k, WARP_SIZE);
-             }
-             if (srcLane == 0) { snr_shared[wID] = laneChi2; }
-             __syncthreads();
-             if (threadIdx.x < wn) {
-                 laneChi2 = snr_shared[srcLane];
-                 for (i = wn / 2; i > 0; i = i >> 1) {
-                     laneChi2 += __shfl_xor(laneChi2, i);
-                 }
-                 if (srcLane == 0) {
-                     chisq_cur = laneChi2 / autocorr_norm[j][tmplt_cur];
-                     // the location of chisq_* is indexed from maxsnglsnr
-                     snglsnr_H[peak_cur
-                               + (6 + write_ifo_mapping[j]) * max_npeak] =
-                       chisq_cur;
-
-                     cmbchisq[peak_cur] += chisq_cur;
-                     // printf("peak %d, itrial %d, cohsnr %f, nullstream %f,
-                     // ipix %d, chisq %f\n", ipeak, itrial, cohsnr[peak_cur],
-                     // nullsnr[peak_cur], pix_idx[peak_cur], cmbchisq[peak_cur]);
-                 }
-             }
-             __syncthreads();
-         }
-
-         __syncthreads();
-
-         /*
-          *
-          * Generate background cohsnr; nullsnr; chisq
-          *
-          */
-
-         int ipix = 0, rand_range = trial_sample_inv * hist_trials - 1;
-         for (itrial = 1 + threadIdx.x / WARP_SIZE; itrial <= hist_trials;
-              itrial += blockDim.x / WARP_SIZE) {
-             snr_max        = 0.0;
-             nullstream_max = 0.0;
-             sky_idx        = 0;
-
-             // FIXME: try using random offset like the following
-             // trial_offset = rand()% rand_range + 1;
-             trial_offset  = itrial * trial_sample_inv;
-             output_offset = peak_cur + (itrial - 1) * max_npeak;
-             for (int seed_pix = srcLane;
-                  seed_pix < num_sky_directions / NSKY_REDUCE_RATIO;
-                  seed_pix += WARP_SIZE) {
-                 snr_tmp = 0.0;
-                 al_all  = 0.0;
-                 // matrix u is stored in column order
-                 // mu = u_map + nifo * nifo * i;
-
-                 ipix = (seed_pix * NSKY_REDUCE_RATIO)
-                        + (itrial & (NSKY_REDUCE_RATIO - 1));
-                 for (int j = 0; j < nifo; ++j) {
-                     /* this is a simplified algorithm to get map_idx */
-                     map_idx = iifo * nifo + j;
-
-                     NtOff = round(
-                       toa_diff_map[map_idx * num_sky_directions + ipix] / dt);
-                     // The background cohsnr should be obtained coherently as
-                     // well.
-                     int offset =
```

```
-                     (j == iifo ? 0 : NtOff - (trial_offset * (j - iifo)));
-                 // dk[j] = snr[j][((start_exe + len_cur + offset + len) %
-                 // len) * ntmplt + tmplt_cur ];
-                 dk[j] =
-                   snr[j][len * tmplt_cur
-                          + ((start_exe + len_cur + offset + len) % len)];
-             }
-
-             for (int j = 0; j < nifo; ++j) {
-                 real = 0.0f;
-                 imag = 0.0f;
-                 for (int k = 0; k < nifo; ++k) {
-                     // transpose of u_map
-                     real +=
-                       u_map[(k * nifo + j) * num_sky_directions + ipix]
-                       * dk[k].re;
-                     imag +=
-                       u_map[(k * nifo + j) * num_sky_directions + ipix]
-                       * dk[k].im;
-                 }
-                 (j < 2 ? snr_tmp : al_all) += real * real + imag * imag;
-             }
-
-             stat_tmp = snr_tmp - 0.0;
-             if (stat_tmp > stat_max) {
-                 stat_max        = stat_tmp;
-                 snr_max         = snr_tmp;
-                 nullstream_max = al_all;
-                 sky_idx         = ipix;
-             }
-         }
-
-         for (i = WARP_SIZE / 2; i > 0; i = i >> 1) {
-             stat_tmp           = __shfl_xor(stat_max, i);
-             snr_tmp            = __shfl_xor(snr_max, i);
-             nullstream_max_tmp = __shfl_xor(nullstream_max, i);
-             sky_idx_tmp        = __shfl_xor(sky_idx, i);
-
-             if (stat_tmp > stat_max) {
-                 stat_max        = stat_tmp;
-                 snr_max         = snr_tmp;
-                 nullstream_max = nullstream_max_tmp;
-                 sky_idx         = sky_idx_tmp;
-             }
-         }
-         if (srcLane == 0) {
-             cohsnr_bg[output_offset]  = snr_max;
-             nullsnr_bg[output_offset] = nullstream_max;
-             /* background need this for Ntoff */
-             pix_idx_bg[output_offset] = sky_idx;
-         }
-         __syncthreads();
-
-         /* c code here
-         COMPLEX_F data;
-         chisq[peak_cur] = 0.0;
-         int autochisq_half_len = autochisq_len /2;
-         for (int j = 0; j < nifo; ++j)
-         {   data = 0;
-             // this is a simplified algorithm to get map_idx
-             map_idx = iifo * nifo + j;
-             NtOff = round (toa_diff_map[map_idx * num_sky_directions + ipix]
-         / dt); for(int ishift=-autochisq_half_len;
-         ishift<=autochisq_half_len; ishift++)
-             {
-
-             data += snr[j][((start_exe + peak_cur + NtOff + ishift) % len) *
-         ntmplt + tmplt_cur] - maxsnglsnr[peak_cur] * autocorr_matrix[j][
-         tmplt_cur * autochisq_len + ishift + autochisq_half_len];
-             }
-             chisq[peak_cur] += (data.re * data.re + data.im * data.im) /
-         autocorr_norm[j][tmplt_cur];
-         }
```

```
-            */
-            COMPLEX_F data, tmp_snr, tmp_autocorr, tmp_maxsnr;
-            float laneChi2          = 0.0f;
-            int autochisq_half_len = autochisq_len / 2, peak_pos_tmp;
-
-            cmbchisq_bg[output_offset] = 0.0;
-
-            for (int j = 0; j < nifo; ++j) {
-                laneChi2 = 0.0f;
-                /* this is a simplified algorithm to get map_idx */
-                map_idx = iifo * nifo + j;
-
-                NtOff = round(toa_diff_map[map_idx * num_sky_directions
-                                           + pix_idx_bg[output_offset]]
-                              / dt);
-
-                peak_pos_tmp =
-                  start_exe + len_cur
-                  + (j == iifo ? 0 : NtOff - (trial_offset * (j - iifo)) + len);
-
-                // tmp_maxsnr = snr[j][((peak_pos_tmp + len) % len) * ntmplt +
-                // tmplt_cur];
-                tmp_maxsnr =
-                  snr[j][len * tmplt_cur + ((peak_pos_tmp + len) % len)];
-                /* set the d_snglsnr_* */
-                snglsnr_bg_H[output_offset
-                             + write_ifo_mapping[j] * hist_trials * max_npeak] =
-                  sqrt(tmp_maxsnr.re * tmp_maxsnr.re
-                       + tmp_maxsnr.im * tmp_maxsnr.im);
-                /* set the d_coa_phase_* */
-                snglsnr_bg_H[output_offset
-                             + (3 + write_ifo_mapping[j]) * hist_trials
-                                 * max_npeak] =
-                  atan2(tmp_maxsnr.im, tmp_maxsnr.re);
-
-                for (int ishift = srcLane - autochisq_half_len;
-                     ishift <= autochisq_half_len; ishift += WARP_SIZE) {
-                    // tmp_snr = snr[j][((peak_pos_tmp + ishift + len) % len) *
-                    // ntmplt + tmplt_cur];
-                    tmp_snr = snr[j][len * tmplt_cur
-                                     + ((peak_pos_tmp + ishift + len) % len)];
-                    tmp_autocorr =
-                      autocorr_matrix[j][tmplt_cur * autochisq_len + ishift
-                                         + autochisq_half_len];
-                    data.re = tmp_snr.re - tmp_maxsnr.re * tmp_autocorr.re
-                              + tmp_maxsnr.im * tmp_autocorr.im;
-                    data.im = tmp_snr.im - tmp_maxsnr.re * tmp_autocorr.im
-                              - tmp_maxsnr.im * tmp_autocorr.re;
-                    laneChi2 += (data.re * data.re + data.im * data.im);
-                }
-                for (int k = WARP_SIZE >> 1; k > 0; k = k >> 1) {
-                    laneChi2 += __shfl_xor(laneChi2, k, WARP_SIZE);
-                }
-
-                if (srcLane == 0) {
-                    chisq_cur = laneChi2 / autocorr_norm[j][tmplt_cur];
-                    // set d_chisq_bg_* from snglsnr_bg_H
-                    snglsnr_bg_H[output_offset
-                                 + (6 + write_ifo_mapping[j]) * hist_trials
-                                     * max_npeak] = chisq_cur;
-
-                    cmbchisq_bg[output_offset] += chisq_cur;
-                }
-                __syncthreads();
-            }
-
             __syncthreads();
         }
     }
-    /* find maximum cohsnr and swope to the first pos */
-    volatile float *cohsnr_shared = &smem[0];
-    volatile float *ipeak_shared  = &smem[blockDim.x];
-    float cohsnr_max              = 0.0, cohsnr_cur;
-
```

```
-    if (bid == 0 && npeak > 1) {
-        /* clean up smem history */
-        cohsnr_shared[threadIdx.x] = 0.0;
-        ipeak_shared[threadIdx.x]  = 0;
-        __syncthreads();
-        for (i = threadIdx.x; i < npeak; i += blockDim.x) {
-            peak_cur   = peak_pos[i];
-            cohsnr_cur = cohsnr[peak_cur];
-            if (cohsnr_cur > cohsnr_max) {
-                cohsnr_shared[threadIdx.x] = cohsnr_cur;
-                ipeak_shared[threadIdx.x]  = i;
-                cohsnr_max                 = cohsnr_cur;
-            }
-        }
-        __syncthreads();
-        for (i = blockDim.x >> 1; i > 0; i = i >> 1) {
-            if (threadIdx.x < i) {
-                cohsnr_cur = cohsnr_shared[threadIdx.x + i];
-                cohsnr_max = cohsnr_shared[threadIdx.x];
-
-                if (cohsnr_cur > cohsnr_max) {
-                    cohsnr_shared[threadIdx.x] = cohsnr_cur;
-                    ipeak_shared[threadIdx.x]  = ipeak_shared[threadIdx.x + i];
-                }
-            }
-            __syncthreads();
-        }
-
-        /* swope the first and max peak_cur in peak_pos */
-
-        if (threadIdx.x == 0) {
-            ipeak_max            = ipeak_shared[0];
-            peak_cur             = peak_pos[ipeak_max];
-            peak_pos[ipeak_max] = peak_pos[0];
-            peak_pos[0]          = peak_cur;
-        }
-    }
 }
 #define TRANSPOSE_TILE_DIM   32
 #define TRANSPOSE_BLOCK_ROWS 8
-__global__ void transpose_matrix(COMPLEX_F *idata,
-                                 COMPLEX_F *odata,
+__global__ void transpose_matrix(COMPLEX_F *restrict idata,
+                                 COMPLEX_F *restrict odata,
                                  int in_x_offset,
                                  int in_y_offset,
                                  int in_width,
@@ -1370,42 +906,20 @@ void cohsnr_and_chisq(PostcohState *state,
     MAX(2 * threads * sizeof(float), 4 * threads / WARP_SIZE * sizeof(float));
     PeakList *pklist = state->peak_list[iifo];
     int npeak        = pklist->npeak[0];
-    /*
-            ker_coh_max_and_chisq<<<npeak, threads, sharedsize, stream>>>(
-                                                                          state->dd_snglsnr,
-                                                                          iifo,
-                                                                          state->nifo,
-                                                                          state->d_write_ifo_mapping,
-                                                                          pklist->d_peak_pos,
-                                                                          pklist->d_snglsnr_H,
-                                                                          pklist->d_snglsnr_bg_H,
-                                                                          pklist->d_cohsnr_skymap,
-                                                                          pklist->d_nullsnr_skymap,
-                                                                          output_skymap,
-                                                                          pklist->npeak[0],
-                                                                          state->d_U_map[gps_idx],
-                                                                          state->d_diff_map[gps_idx],
-                                                                          state->npix,
-                                                                          state->snglsnr_len,
-                                                                          state->max_npeak,
-                                                                          state->snglsnr_start_exe,
-                                                                          state->dt,
-                                                                          state->ntmplt,
```

```
-                                                                   state->autochisq_len,
-                                                                   state->dd_autocorr_matrix,
-                                                                   state->dd_autocorr_norm,
-                                                                   state->hist_trials,
-                                                                   state->trial_sample_inv);
-    */
+
     ker_coh_max_and_chisq_versatile<<<npeak, threads, sharedsize, stream>>>(
       state->dd_snglsnr, iifo, state->nifo, state->cur_nifo,
       state->cur_ifo_bits, state->d_write_ifo_mapping, pklist->d_peak_pos,
-      pklist->d_snglsnr_H, pklist->d_snglsnr_bg_H, pklist->d_cohsnr_skymap,
-      pklist->d_nullsnr_skymap, output_skymap, pklist->npeak[0],
+      pklist->d_snglsnr, pklist->d_snglsnr_bg, pklist->npeak[0],
       state->d_U_map[gps_idx], state->d_diff_map[gps_idx], state->npix,
       state->snglsnr_len, state->max_npeak, state->snglsnr_start_exe, state->dt,
       state->ntmplt, state->autochisq_len, state->dd_autocorr_matrix,
-      state->dd_autocorr_norm, state->hist_trials, state->trial_sample_inv);
+      state->dd_autocorr_norm, state->hist_trials, state->trial_sample_inv,
+      pklist->d_pix_idx, pklist->d_pix_idx_bg, pklist->d_cohsnr,
+      pklist->d_nullsnr, pklist->d_cmbchisq, pklist->d_cohsnr_bg,
+      pklist->d_nullsnr_bg, pklist->d_cmbchisq_bg, pklist->d_coaphase,
+      pklist->d_coaphase_bg, pklist->d_ntoff, pklist->d_chisq,
+      pklist->d_chisq_bg, pklist->d_len_idx, pklist->d_tmplt_idx);

     CUDA_CHECK(cudaStreamSynchronize(stream));
     CUDA_CHECK(cudaPeekAtLastError());
@@ -1413,10 +927,11 @@ void cohsnr_and_chisq(PostcohState *state,
     if (output_skymap && state->snglsnr_max[iifo] > output_skymap) {
         ker_coh_skymap<<<1, threads, sharedsize, stream>>>(
           pklist->d_cohsnr_skymap, pklist->d_nullsnr_skymap, state->dd_snglsnr,
-          iifo, state->nifo, pklist->d_peak_pos, pklist->d_snglsnr_H,
-          pklist->npeak[0], state->d_U_map[gps_idx], state->d_diff_map[gps_idx],
-          state->npix, state->max_npeak, state->snglsnr_len,
-          state->snglsnr_start_exe, state->dt, state->ntmplt);
+          iifo, state->nifo, pklist->d_peak_pos, pklist->d_snglsnr,
+          pklist->d_cohsnr, pklist->npeak[0], state->d_U_map[gps_idx],
+          state->d_diff_map[gps_idx], state->npix, state->max_npeak,
+          state->snglsnr_len, state->snglsnr_start_exe, state->dt,
+          state->ntmplt, pklist->d_len_idx, pklist->d_tmplt_idx);

         CUDA_CHECK(cudaStreamSynchronize(stream));
         CUDA_CHECK(cudaPeekAtLastError());
@@ -1427,7 +942,7 @@ void cohsnr_and_chisq(PostcohState *state,
     }

     /* copy the snr, cohsnr, nullsnr, chisq out */
-    CUDA_CHECK(cudaMemcpyAsync(pklist->snglsnr_H, pklist->d_snglsnr_H,
+    CUDA_CHECK(cudaMemcpyAsync(pklist->snglsnr[0], pklist->d_snglsnr[0],
                                sizeof(float) * (pklist->peak_floatlen),
                                cudaMemcpyDeviceToHost, stream));
```

```
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.c b/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.c
index 555b6818..c04803ef 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.c
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.c
@@ -169,71 +169,107 @@ PeakList *create_peak_list(PostcohState *state, cudaStream_t stream) {
 #endif
     PeakList *pklist = (PeakList *)malloc(sizeof(PeakList));

-    int peak_intlen      = (7 + hist_trials) * max_npeak + 1;
-    int peak_floatlen    = (12 + hist_trials * 12) * max_npeak;
+    int peak_intlen = (4 + MAX_NIFO + hist_trials) * max_npeak + 1;
+    int peak_floatlen =
+      ((4 * MAX_NIFO) + (hist_trials * 4 * MAX_NIFO)) * max_npeak;
     pklist->peak_intlen   = peak_intlen;
     pklist->peak_floatlen = peak_floatlen;

+    // [THA]: Why do we use `cudaMallocManaged()` sometimes below? Well, a large
+    // number of the below pointers are to 2D arrays that we won't be accessing
+    // after setting up, but whilst we're setting them up they need to be able
+    // to be accessed on the CPU. `cudaMallocManaged()` allows us to access the
+    // memory involved on both the CPU and GPU, which means that we can assign
+    // the pointers here instead of having to do an awkward `cudaMemcpyAsync()`
+    // to copy across the right pointer.
+    //
+    // It's likely that there will be a small performance hit for having used
```

```
+    // `cudaMallocManaged()` instead of doing an async copy, so if its needed
+    // to, the below code can definitely can be changed and will work with
+    // `cudaMemcpyAsync()`.
+    //
+    // FIXME: Move to `cudaMemcpyAsync()` instead of `cudaMallocManaged()` for a
+    // slight performance bump
+
     /* create device space for peak list for int-type variables */
     CUDA_CHECK(
       cudaMalloc((void **)&(pklist->d_npeak), sizeof(int) * peak_intlen));
     CUDA_CHECK(
       cudaMemsetAsync(pklist->d_npeak, 0, sizeof(int) * peak_intlen, stream));
-    pklist->d_peak_pos   = pklist->d_npeak + 1;
-    pklist->d_len_idx    = pklist->d_npeak + 1 + max_npeak;
+    pklist->d_peak_pos   = pklist->d_npeak + 1 + 0 * max_npeak;
+    pklist->d_len_idx    = pklist->d_npeak + 1 + 1 * max_npeak;
     pklist->d_tmplt_idx  = pklist->d_npeak + 1 + 2 * max_npeak;
     pklist->d_pix_idx    = pklist->d_npeak + 1 + 3 * max_npeak;
     pklist->d_pix_idx_bg = pklist->d_npeak + 1 + 4 * max_npeak;
-    pklist->d_ntoff_H    = pklist->d_npeak + 1 + (4 + hist_trials) * max_npeak;
-    pklist->d_ntoff_L    = pklist->d_npeak + 1 + (5 + hist_trials) * max_npeak;
-    pklist->d_ntoff_V    = pklist->d_npeak + 1 + (6 + hist_trials) * max_npeak;
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_ntoff),
+                                 sizeof(int *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        pklist->d_ntoff[i] =
+          pklist->d_npeak + 1 + ((4 + hist_trials + i) * max_npeak);
+    }
     // printf("d_npeak %p\n", pklist->d_npeak);
     // CUDA_CHECK(cudaMemsetAsync(pklist->d_npeak, 0, sizeof(int), stream));
     /* create device space for peak list for float-type variables */
-    CUDA_CHECK(cudaMalloc((void **)&(pklist->d_snglsnr_H),
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_snglsnr),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_coaphase),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_chisq),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_snglsnr_bg),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_coaphase_bg),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+    CUDA_CHECK(cudaMallocManaged((void **)&(pklist->d_chisq_bg),
+                                 sizeof(float *) * MAX_NIFO,
+                                 cudaMemAttachGlobal));
+
+    CUDA_CHECK(cudaMalloc((void **)&(pklist->d_snglsnr[0]),
                           sizeof(float) * peak_floatlen));
-    CUDA_CHECK(cudaMemsetAsync(pklist->d_snglsnr_H, 0,
+    CUDA_CHECK(cudaMemsetAsync(pklist->d_snglsnr[0], 0,
                                sizeof(float) * peak_floatlen, stream));
-    pklist->d_snglsnr_L  = pklist->d_snglsnr_H + max_npeak;
-    pklist->d_snglsnr_V  = pklist->d_snglsnr_H + 2 * max_npeak;
-    pklist->d_coaphase_H = pklist->d_snglsnr_H + 3 * max_npeak;
-    pklist->d_coaphase_L = pklist->d_snglsnr_H + 4 * max_npeak;
-    pklist->d_coaphase_V = pklist->d_snglsnr_H + 5 * max_npeak;
-    pklist->d_chisq_H    = pklist->d_snglsnr_H + 6 * max_npeak;
-    pklist->d_chisq_L    = pklist->d_snglsnr_H + 7 * max_npeak;
-    pklist->d_chisq_V    = pklist->d_snglsnr_H + 8 * max_npeak;
-    pklist->d_cohsnr     = pklist->d_snglsnr_H + 9 * max_npeak;
-    pklist->d_nullsnr    = pklist->d_snglsnr_H + 10 * max_npeak;
-    pklist->d_cmbchisq   = pklist->d_snglsnr_H + 11 * max_npeak;
-
-    pklist->d_snglsnr_bg_H = pklist->d_snglsnr_H + 12 * max_npeak;
-    pklist->d_snglsnr_bg_L =
-      pklist->d_snglsnr_H + (12 + hist_trials) * max_npeak;
```

```
-     pklist->d_snglsnr_bg_V =
-       pklist->d_snglsnr_H + (12 + 2 * hist_trials) * max_npeak;
-     pklist->d_coaphase_bg_H =
-       pklist->d_snglsnr_H + (12 + 3 * hist_trials) * max_npeak;
-     pklist->d_coaphase_bg_L =
-       pklist->d_snglsnr_H + (12 + 4 * hist_trials) * max_npeak;
-     pklist->d_coaphase_bg_V =
-       pklist->d_snglsnr_H + (12 + 5 * hist_trials) * max_npeak;
-     pklist->d_chisq_bg_H =
-       pklist->d_snglsnr_H + (12 + 6 * hist_trials) * max_npeak;
-     pklist->d_chisq_bg_L =
-       pklist->d_snglsnr_H + (12 + 7 * hist_trials) * max_npeak;
-     pklist->d_chisq_bg_V =
-       pklist->d_snglsnr_H + (12 + 8 * hist_trials) * max_npeak;
+
+     for (int i = 0; i < MAX_NIFO; ++i) {
+         pklist->d_snglsnr[i] = pklist->d_snglsnr[0] + (max_npeak * i);
+         pklist->d_coaphase[i] =
+           pklist->d_snglsnr[0] + (max_npeak * (i + MAX_NIFO));
+         pklist->d_chisq[i] =
+           pklist->d_snglsnr[0] + (max_npeak * (i + 2 * MAX_NIFO));
+
+         pklist->d_snglsnr_bg[i] =
+           pklist->d_snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 0 * MAX_NIFO))));
+         pklist->d_coaphase_bg[i] =
+           pklist->d_snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 1 * MAX_NIFO))));
+         pklist->d_chisq_bg[i] =
+           pklist->d_snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 2 * MAX_NIFO))));
+     }
+     pklist->d_cohsnr   = pklist->d_snglsnr[0] + (3 * MAX_NIFO + 0) * max_npeak;
+     pklist->d_nullsnr  = pklist->d_snglsnr[0] + (3 * MAX_NIFO + 1) * max_npeak;
+     pklist->d_cmbchisq = pklist->d_snglsnr[0] + (3 * MAX_NIFO + 2) * max_npeak;
+
      pklist->d_cohsnr_bg =
-       pklist->d_snglsnr_H + (12 + 9 * hist_trials) * max_npeak;
+       pklist->d_snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (0 + 3 * MAX_NIFO))));
+
      pklist->d_nullsnr_bg =
-       pklist->d_snglsnr_H + (12 + 10 * hist_trials) * max_npeak;
+       pklist->d_snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (1 + 3 * MAX_NIFO))));
      pklist->d_cmbchisq_bg =
-       pklist->d_snglsnr_H + (12 + 11 * hist_trials) * max_npeak;
+       pklist->d_snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (2 + 3 * MAX_NIFO))));
      /* create host space for peak list for int-type variables */
-     // pklist->npeak = (int *)malloc(sizeof(int) * peak_intlen);
      CUDA_CHECK(
        cudaMallocHost((void **)&(pklist->npeak), sizeof(int) * peak_intlen));
      memset(pklist->npeak, 0, sizeof(int) * peak_intlen);
@@ -242,49 +278,46 @@ PeakList *create_peak_list(PostcohState *state, cudaStream_t stream) {
      pklist->tmplt_idx  = pklist->npeak + 1 + 2 * max_npeak;
      pklist->pix_idx    = pklist->npeak + 1 + 3 * max_npeak;
      pklist->pix_idx_bg = pklist->npeak + 1 + 4 * max_npeak;
-     pklist->ntoff_H    = pklist->npeak + 1 + (4 + hist_trials) * max_npeak;
-     pklist->ntoff_L    = pklist->npeak + 1 + (5 + hist_trials) * max_npeak;
-     pklist->ntoff_V    = pklist->npeak + 1 + (6 + hist_trials) * max_npeak;
+     for (int i = 0; i < MAX_NIFO; ++i) {
+         pklist->ntoff[i] =
+           pklist->npeak + 1 + (4 + hist_trials + i) * max_npeak;
+     }
      /* create host space for peak list for float-type variables */
-     // pklist->snglsnr_L = (float *)malloc(sizeof(float) * peak_floatlen);
-     CUDA_CHECK(cudaMallocHost((void **)&(pklist->snglsnr_H),
+     CUDA_CHECK(cudaMallocHost((void **)&(pklist->snglsnr[0]),
                                sizeof(float) * peak_floatlen));
-     memset(pklist->snglsnr_H, 0, sizeof(float) * peak_floatlen);
```

```
-     pklist->snglsnr_L  = pklist->snglsnr_H + max_npeak;
-     pklist->snglsnr_V  = pklist->snglsnr_H + 2 * max_npeak;
-     pklist->coaphase_H = pklist->snglsnr_H + 3 * max_npeak;
-     pklist->coaphase_L = pklist->snglsnr_H + 4 * max_npeak;
-     pklist->coaphase_V = pklist->snglsnr_H + 5 * max_npeak;
-     pklist->chisq_H    = pklist->snglsnr_H + 6 * max_npeak;
-     pklist->chisq_L    = pklist->snglsnr_H + 7 * max_npeak;
-     pklist->chisq_V    = pklist->snglsnr_H + 8 * max_npeak;
-     pklist->cohsnr     = pklist->snglsnr_H + 9 * max_npeak;
-     pklist->nullsnr    = pklist->snglsnr_H + 10 * max_npeak;
-     pklist->cmbchisq   = pklist->snglsnr_H + 11 * max_npeak;
-     //
-     pklist->snglsnr_bg_H = pklist->snglsnr_H + 12 * max_npeak;
-     pklist->snglsnr_bg_L = pklist->snglsnr_H + (12 + hist_trials) * max_npeak;
-     pklist->snglsnr_bg_V =
-       pklist->snglsnr_H + (12 + 2 * hist_trials) * max_npeak;
-     pklist->coaphase_bg_H =
-       pklist->snglsnr_H + (12 + 3 * hist_trials) * max_npeak;
-     pklist->coaphase_bg_L =
-       pklist->snglsnr_H + (12 + 4 * hist_trials) * max_npeak;
-     pklist->coaphase_bg_V =
-       pklist->snglsnr_H + (12 + 5 * hist_trials) * max_npeak;
-     pklist->chisq_bg_H = pklist->snglsnr_H + (12 + 6 * hist_trials) * max_npeak;
-     pklist->chisq_bg_L = pklist->snglsnr_H + (12 + 7 * hist_trials) * max_npeak;
-     pklist->chisq_bg_V = pklist->snglsnr_H + (12 + 8 * hist_trials) * max_npeak;
-     pklist->cohsnr_bg  = pklist->snglsnr_H + (12 + 9 * hist_trials) * max_npeak;
+     memset(pklist->snglsnr[0], 0, sizeof(float) * peak_floatlen);
+     for (int i = 0; i < MAX_NIFO; ++i) {
+         pklist->snglsnr[i]  = pklist->snglsnr[0] + (max_npeak * i);
+         pklist->coaphase[i] = pklist->snglsnr[0] + (max_npeak * (i + MAX_NIFO));
+         pklist->chisq[i] =
+           pklist->snglsnr[0] + (max_npeak * (i + 2 * MAX_NIFO));
+
+         pklist->snglsnr_bg[i] =
+           pklist->snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 0 * MAX_NIFO))));
+         pklist->coaphase_bg[i] =
+           pklist->snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 1 * MAX_NIFO))));
+         pklist->chisq_bg[i] =
+           pklist->snglsnr[0]
+           + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (i + 2 * MAX_NIFO))));
+     }
+     pklist->cohsnr   = pklist->snglsnr[0] + (3 * MAX_NIFO + 0) * max_npeak;
+     pklist->nullsnr  = pklist->snglsnr[0] + (3 * MAX_NIFO + 1) * max_npeak;
+     pklist->cmbchisq = pklist->snglsnr[0] + (3 * MAX_NIFO + 2) * max_npeak;
+
+     pklist->cohsnr_bg =
+       pklist->snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (0 + 3 * MAX_NIFO))));
+
      pklist->nullsnr_bg =
-       pklist->snglsnr_H + (12 + 10 * hist_trials) * max_npeak;
+       pklist->snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (1 + 3 * MAX_NIFO))));
      pklist->cmbchisq_bg =
-       pklist->snglsnr_H + (12 + 11 * hist_trials) * max_npeak;
+       pklist->snglsnr[0]
+       + (max_npeak * ((4 * MAX_NIFO) + (hist_trials * (2 + 3 * MAX_NIFO))));
-     //         printf("set peak addr %p, d_npeak addr %p\n", pklist,
-     //pklist->d_npeak); printf("hist trials %d, peak_intlen %d, peak_floatlen
-     // %d\n", hist_trials, peak_intlen, peak_floatlen);
      /* temporary struct to store tmplt max in one max_npeak data */
      CUDA_CHECK(cudaMalloc((void **)&(pklist->d_peak_tmplt),
                            sizeof(float) * state->ntmplt));
@@ -820,12 +853,24 @@ static void autocorr_destroy(PostcohState *state) {
  void peak_list_destroy(PeakList *pklist) {

      CUDA_CHECK(cudaFree(pklist->d_npeak));
-     CUDA_CHECK(cudaFree(pklist->d_snglsnr_L));
+     CUDA_CHECK(cudaFree(pklist->d_snglsnr[0]));
+     CUDA_CHECK(cudaFree(pklist->d_snglsnr));
```

```
+    CUDA_CHECK(cudaFree(pklist->d_coaphase));
+    CUDA_CHECK(cudaFree(pklist->d_chisq));
+    CUDA_CHECK(cudaFree(pklist->d_snglsnr_bg));
+    CUDA_CHECK(cudaFree(pklist->d_coaphase_bg));
+    CUDA_CHECK(cudaFree(pklist->d_chisq_bg));
     CUDA_CHECK(cudaFree(pklist->d_peak_tmplt));
     CUDA_CHECK(cudaFree(pklist->d_cohsnr_skymap));

     CUDA_CHECK(cudaFreeHost(pklist->npeak));
-    CUDA_CHECK(cudaFreeHost(pklist->snglsnr_L));
+    CUDA_CHECK(cudaFreeHost(pklist->snglsnr[0]));
+    CUDA_CHECK(cudaFreeHost(pklist->snglsnr));
+    CUDA_CHECK(cudaFreeHost(pklist->coaphase));
+    CUDA_CHECK(cudaFreeHost(pklist->chisq));
+    CUDA_CHECK(cudaFreeHost(pklist->snglsnr_bg));
+    CUDA_CHECK(cudaFreeHost(pklist->coaphase_bg));
+    CUDA_CHECK(cudaFreeHost(pklist->chisq_bg));
     CUDA_CHECK(cudaFreeHost(pklist->cohsnr_skymap));
 }
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.h b/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.h
index d9837f1c..41e2bdd9 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.h
+++ b/gstlal-spiir/gst/cuda/postcoh/postcoh_utils.h
@@ -15,6 +15,8 @@ void cuda_device_print(int deviceCount);

 PeakList *create_peak_list(PostcohState *state, cudaStream_t stream);

+void get_write_ifo_mapping(char *ifo_combo, int nifo, int *write_ifo_mapping);
+
 void cuda_postcoh_map_from_xml(char *fname,
                                PostcohState *state,
                                cudaStream_t stream);
diff --git a/gstlal-spiir/gst/cuda/postcoh/postcohtable_utils.c b/gstlal-spiir/gst/cuda/postcoh/postcohtable_
index db53a829..87ecf159 100644
--- a/gstlal-spiir/gst/cuda/postcoh/postcohtable_utils.c
+++ b/gstlal-spiir/gst/cuda/postcoh/postcohtable_utils.c
@@ -26,136 +26,148 @@ void postcohtable_init(XmlTable *table) {
     table->delimiter = g_string_new(",");

-    table->names = g_array_new(FALSE, FALSE, sizeof(GString));
+    table->names      = g_array_new(FALSE, FALSE, sizeof(GString));
+    table->type_names = g_array_new(FALSE, FALSE, sizeof(GString));
+
     g_array_append_val(table->names, *g_string_new("postcoh:end_time"));
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
     g_array_append_val(table->names, *g_string_new("postcoh:end_time_ns"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_L"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_ns_L"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_H"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_ns_H"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_V"));
-    g_array_append_val(table->names, *g_string_new("postcoh:end_time_ns_V"));
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:end_time_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("int_4s"));
+        g_array_append_val(
+          table->names, *g_string_append(g_string_new("postcoh:end_time_ns_"),
+                                         IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("int_4s"));
+    }
     g_array_append_val(table->names, *g_string_new("postcoh:is_background"));
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
     g_array_append_val(table->names, *g_string_new("postcoh:livetime"));
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
     g_array_append_val(table->names, *g_string_new("postcoh:ifos"));
+    g_array_append_val(table->type_names, *g_string_new("lstring"));
     g_array_append_val(table->names, *g_string_new("postcoh:pivotal_ifo"));
+    g_array_append_val(table->type_names, *g_string_new("lstring"));
     g_array_append_val(table->names, *g_string_new("postcoh:tmplt_idx"));
```

```
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
     g_array_append_val(table->names, *g_string_new("postcoh:pix_idx"));
-    g_array_append_val(table->names, *g_string_new("postcoh:snglsnr_L"));
-    g_array_append_val(table->names, *g_string_new("postcoh:snglsnr_H"));
-    g_array_append_val(table->names, *g_string_new("postcoh:snglsnr_V"));
-    g_array_append_val(table->names, *g_string_new("postcoh:coaphase_L"));
-    g_array_append_val(table->names, *g_string_new("postcoh:coaphase_H"));
-    g_array_append_val(table->names, *g_string_new("postcoh:coaphase_V"));
-    g_array_append_val(table->names, *g_string_new("postcoh:chisq_L"));
-    g_array_append_val(table->names, *g_string_new("postcoh:chisq_H"));
-    g_array_append_val(table->names, *g_string_new("postcoh:chisq_V"));
+    g_array_append_val(table->type_names, *g_string_new("int_4s"));
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:snglsnr_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:snglsnr_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+    }
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:coaphase_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:coaphase_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+    }
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:chisq_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:chisq_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+    }
     g_array_append_val(table->names, *g_string_new("postcoh:cohsnr"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:nullsnr"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:cmbchisq"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:spearman_pval"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:fap"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:far"));
-    g_array_append_val(table->names, *g_string_new("postcoh:far_l"));
-    g_array_append_val(table->names, *g_string_new("postcoh:far_h"));
-    g_array_append_val(table->names, *g_string_new("postcoh:far_v"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:far_sngl_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:far_sngl_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_4"));
+    }
     g_array_append_val(table->names, *g_string_new("postcoh:far_2h"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
     g_array_append_val(table->names, *g_string_new("postcoh:far_1d"));
+    g_array_append_val(table->type_names, *g_string_new("real_4"));
```

```
        g_array_append_val(table->names, *g_string_new("postcoh:far_1w"));
+       g_array_append_val(table->type_names, *g_string_new("real_4"));
        g_array_append_val(table->names, *g_string_new("postcoh:skymap_fname"));
+       g_array_append_val(table->type_names, *g_string_new("lstring"));
        g_array_append_val(table->names,
                           *g_string_new("postcoh:template_duration"));
-       g_array_append_val(table->names, *g_string_new("postcoh:mchirp"));
-       g_array_append_val(table->names, *g_string_new("postcoh:mtotal"));
-       g_array_append_val(table->names, *g_string_new("postcoh:mass1"));
-       g_array_append_val(table->names, *g_string_new("postcoh:mass2"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin1x"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin1y"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin1z"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin2x"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin2y"));
-       g_array_append_val(table->names, *g_string_new("postcoh:spin2z"));
-       g_array_append_val(table->names, *g_string_new("postcoh:ra"));
-       g_array_append_val(table->names, *g_string_new("postcoh:dec"));
-       g_array_append_val(table->names, *g_string_new("postcoh:deff_L"));
-       g_array_append_val(table->names, *g_string_new("postcoh:deff_H"));
-       g_array_append_val(table->names, *g_string_new("postcoh:deff_V"));
-
-       table->type_names = g_array_new(FALSE, FALSE, sizeof(GString));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("lstring"));
-       g_array_append_val(table->type_names, *g_string_new("lstring"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("int_4s"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("real_4"));
-       g_array_append_val(table->type_names, *g_string_new("lstring"));
        g_array_append_val(table->type_names, *g_string_new("real_8"));
+       g_array_append_val(table->names, *g_string_new("postcoh:mchirp"));
        g_array_append_val(table->type_names, *g_string_new("real_4"));
+       g_array_append_val(table->names, *g_string_new("postcoh:mtotal"));
        g_array_append_val(table->type_names, *g_string_new("real_4"));
+       g_array_append_val(table->names, *g_string_new("postcoh:mass1"));
        g_array_append_val(table->type_names, *g_string_new("real_4"));
+       g_array_append_val(table->names, *g_string_new("postcoh:mass2"));
        g_array_append_val(table->type_names, *g_string_new("real_4"));
+       g_array_append_val(table->names, *g_string_new("postcoh:spin1x"));
        g_array_append_val(table->type_names, *g_string_new("real_4"));
```

```
+    g_array_append_val(table->names, *g_string_new("postcoh:spin1y"));
     g_array_append_val(table->type_names, *g_string_new("real_4"));
+    g_array_append_val(table->names, *g_string_new("postcoh:spin1z"));
     g_array_append_val(table->type_names, *g_string_new("real_4"));
+    g_array_append_val(table->names, *g_string_new("postcoh:spin2x"));
     g_array_append_val(table->type_names, *g_string_new("real_4"));
+    g_array_append_val(table->names, *g_string_new("postcoh:spin2y"));
     g_array_append_val(table->type_names, *g_string_new("real_4"));
+    g_array_append_val(table->names, *g_string_new("postcoh:spin2z"));
     g_array_append_val(table->type_names, *g_string_new("real_4"));
+    g_array_append_val(table->names, *g_string_new("postcoh:ra"));
     g_array_append_val(table->type_names, *g_string_new("real_8"));
+    g_array_append_val(table->names, *g_string_new("postcoh:dec"));
     g_array_append_val(table->type_names, *g_string_new("real_8"));
-    g_array_append_val(table->type_names, *g_string_new("real_8"));
-    g_array_append_val(table->type_names, *g_string_new("real_8"));
-    g_array_append_val(table->type_names, *g_string_new("real_8"));
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:deff_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_8"));
+        g_array_append_val(
+          table->names,
+          *g_string_append(g_string_new("postcoh:deff_"), IFOMap[i].name));
+        g_array_append_val(table->type_names, *g_string_new("real_8"));
+    }
 }
 void postcohtable_set_line(GString *line,
                            PostcohInspiralTable *table,
                            XmlTable *xtable) {
-    g_string_append_printf(line, "%d%s", table->end_time_L.gpsSeconds,
-                           xtable->delimiter->str); // for end_time_ns
-    g_string_append_printf(line, "%d%s", table->end_time_L.gpsNanoSeconds,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_L.gpsSeconds,
+    g_string_append_printf(line, "%d%s", table->end_time.gpsSeconds,
                            xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_L.gpsNanoSeconds,
+    g_string_append_printf(line, "%d%s", table->end_time.gpsNanoSeconds,
                            xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_H.gpsSeconds,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_H.gpsNanoSeconds,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_V.gpsSeconds,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%d%s", table->end_time_V.gpsNanoSeconds,
-                           xtable->delimiter->str);
-
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%d%s", table->end_time_sngl[i].gpsSeconds,
+                               xtable->delimiter->str);
+        g_string_append_printf(line, "%d%s",
+                               table->end_time_sngl[i].gpsNanoSeconds,
+                               xtable->delimiter->str);
+    }
     g_string_append_printf(line, "%d%s", table->is_background,
                            xtable->delimiter->str);
     g_string_append_printf(line, "%d%s", table->livetime,
@@ -167,24 +179,19 @@ void postcohtable_set_line(GString *line,
                            xtable->delimiter->str);
     g_string_append_printf(line, "%d%s", table->pix_idx,
                            xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->snglsnr_L,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->snglsnr_H,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->snglsnr_V,
-                           xtable->delimiter->str);
```

```
-    g_string_append_printf(line, "%g%s", table->coaphase_L,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->coaphase_H,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->coaphase_V,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->chisq_L,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->chisq_H,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->chisq_V,
-                           xtable->delimiter->str);
+
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%g%s", table->snglsnr[i],
+                               xtable->delimiter->str);
+    }
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%g%s", table->coaphase[i],
+                               xtable->delimiter->str);
+    }
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%g%s", table->chisq[i],
+                               xtable->delimiter->str);
+    }
     g_string_append_printf(line, "%g%s", table->cohsnr, xtable->delimiter->str);
     g_string_append_printf(line, "%g%s", table->nullsnr,
@@ -195,9 +202,11 @@ void postcohtable_set_line(GString *line,
                            xtable->delimiter->str);
     g_string_append_printf(line, "%g%s", table->fap, xtable->delimiter->str);
     g_string_append_printf(line, "%g%s", table->far, xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->far_l, xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->far_h, xtable->delimiter->str);
-    g_string_append_printf(line, "%g%s", table->far_v, xtable->delimiter->str);
+
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%g%s", table->far_sngl[i],
+                               xtable->delimiter->str);
+    }
     g_string_append_printf(line, "%g%s", table->far_2h, xtable->delimiter->str);
     g_string_append_printf(line, "%g%s", table->far_1d, xtable->delimiter->str);
     g_string_append_printf(line, "%g%s", table->far_1w, xtable->delimiter->str);
@@ -217,12 +226,10 @@ void postcohtable_set_line(GString *line,
     g_string_append_printf(line, "%g%s", table->spin2z, xtable->delimiter->str);
     g_string_append_printf(line, "%lg%s", table->ra, xtable->delimiter->str);
     g_string_append_printf(line, "%lg%s", table->dec, xtable->delimiter->str);
-    g_string_append_printf(line, "%lg%s", table->deff_L,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%lg%s", table->deff_H,
-                           xtable->delimiter->str);
-    g_string_append_printf(line, "%lg%s", table->deff_V,
-                           xtable->delimiter->str);
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        g_string_append_printf(line, "%lg%s", table->deff[i],
+                               xtable->delimiter->str);
+    }
     g_string_append(line, "\n");
     // printf("%s", line->str);
diff --git a/gstlal-spiir/gst/cuda/spiir/spiir_kernel.cu b/gstlal-spiir/gst/cuda/spiir/spiir_kernel.cu
index 0c09d485..f84080f1 100644
--- a/gstlal-spiir/gst/cuda/spiir/spiir_kernel.cu
+++ b/gstlal-spiir/gst/cuda/spiir/spiir_kernel.cu
@@ -332,8 +332,8 @@ __global__ void cuda_iir_filter_kernel(COMPLEX8_F *cudaA1,
         for (i = 0; i < step_points; i += nb) {
             for (j = 0; j < nb; ++j) {
                 // data = 0.01f;
-                // data = tex1Dfetch(texRef, shift+i+j);                //use texture,
-                // abandon now
+                // data = tex1Dfetch(texRef, shift+i+j);                //use
+                // texture, abandon now
```

```
                data              = cudaData[shift + i + j];
                fltrOutptReal[tx] = a1.re * previousSnr.re
                                  - a1.im * previousSnr.im + b0.re * data;
diff --git a/gstlal-spiir/include/pipe_macro.h b/gstlal-spiir/include/pipe_macro.h
index b6e0fc9a..f3d82196 100644
--- a/gstlal-spiir/include/pipe_macro.h
+++ b/gstlal-spiir/include/pipe_macro.h
@@ -6,17 +6,20 @@
 #define MAX_IFO_LEN 4
 #endif

-#define MAX_NIFO 3
 typedef struct _IFOType {
     const char *name;
     int index;
 } IFOType;

-static const IFOType IFOMap[MAX_NIFO] = {
-    { "H1", 0 }, // 1 << 0 = 1
-    { "L1", 1 }, // 1 << 1 = 2
-    { "V1", 2 }, // 1 << 2 = 4
-};
+#define MAX_NIFO 3
+#ifdef __cplusplus
+constexpr
+#endif
+  static const IFOType IFOMap[MAX_NIFO] = {
+      { "H1", 0 }, // 1 << 0 = 1
+      { "L1", 1 }, // 1 << 1 = 2
+      { "V1", 2 }, // 1 << 2 = 4
+  };
 #define MAX_IFO_COMBOS 7 // 2^3-1
 // A combination is sum(1 << index) - 1
 // This gives us some nice mathematical properties that we can use to check
@@ -79,7 +82,6 @@ int get_icombo(char *ifos);
 #define STATS_XML_WRITE_END   3
 #define STATS_XML_WRITE_FULL  4

-#define MAX(a, b)         (a > b ? a : b)
 #define PNOISE_MIN_LIMIT -30
 #define PSIG_MIN_LIMIT   -30
 #define LR_MIN_LIMIT     -30
diff --git a/gstlal-spiir/include/postcohtable.h b/gstlal-spiir/include/postcohtable.h
index e7833c57..f101886b 100644
--- a/gstlal-spiir/include/postcohtable.h
+++ b/gstlal-spiir/include/postcohtable.h
@@ -34,9 +34,7 @@ typedef struct tagPostcohInspiralTable {
     long process_id;
     long event_id;
     LIGOTimeGPS end_time;
-    LIGOTimeGPS end_time_H;
-    LIGOTimeGPS end_time_L;
-    LIGOTimeGPS end_time_V;
+    LIGOTimeGPS end_time_sngl[MAX_NIFO];
     INT4 is_background;
     INT4 livetime;
     CHAR ifos[MAX_ALLIFO_LEN];
@@ -44,32 +42,18 @@ typedef struct tagPostcohInspiralTable {
     INT4 tmplt_idx;
     INT4 bankid;
     INT4 pix_idx;
-    REAL4 snglsnr_H;
-    REAL4 snglsnr_L;
-    REAL4 snglsnr_V;
-    REAL4 coaphase_H;
-    REAL4 coaphase_L;
-    REAL4 coaphase_V;
-    REAL4 chisq_H;
-    REAL4 chisq_L;
-    REAL4 chisq_V;
+    REAL4 snglsnr[MAX_NIFO];
+    REAL4 coaphase[MAX_NIFO];
+    REAL4 chisq[MAX_NIFO];
     REAL4 cohsnr;
     REAL4 nullsnr;
```

```
     REAL4 cmbchisq;
     REAL4 spearman_pval;
     REAL4 fap;
-    REAL4 far_h;
-    REAL4 far_l;
-    REAL4 far_v;
-    REAL4 far_h_1w;
-    REAL4 far_l_1w;
-    REAL4 far_v_1w;
-    REAL4 far_h_1d;
-    REAL4 far_l_1d;
-    REAL4 far_v_1d;
-    REAL4 far_h_2h;
-    REAL4 far_l_2h;
-    REAL4 far_v_2h;
+    REAL4 far_sngl[MAX_NIFO];
+    REAL4 far_1w_sngl[MAX_NIFO];
+    REAL4 far_1d_sngl[MAX_NIFO];
+    REAL4 far_2h_sngl[MAX_NIFO];
     REAL4 far;
     REAL4 far_2h;
     REAL4 far_1d;
@@ -89,9 +73,7 @@ typedef struct tagPostcohInspiralTable {
     REAL4 eta;
     REAL8 ra;
     REAL8 dec;
-    REAL8 deff_H;
-    REAL8 deff_L;
-    REAL8 deff_V;
+    REAL8 deff[MAX_NIFO];
     REAL8 rank;
     REAL4 f_final;
     LIGOTimeGPS epoch;
diff --git a/gstlal-spiir/python/pipemodules/postcoh_finalsink.py b/gstlal-spiir/python/pipemodules/postcoh_f
index e01d9021..ec1d07e1 100644
--- a/gstlal-spiir/python/pipemodules/postcoh_finalsink.py
+++ b/gstlal-spiir/python/pipemodules/postcoh_finalsink.py
@@ -468,7 +468,8 @@ class FinalSink(object):
         else:
             self.gracedb_offline_annote = False
         if GraceDb:
-            self.gracedb_client = GraceDb(gracedb_service_url, reload_certificate=True)
+            self.gracedb_client = GraceDb(gracedb_service_url,
+                                          reload_certificate=True)

         # keep a record of segments and is snapshotted
         # our segments is determined by if incoming buf is GAP
@@ -554,29 +555,20 @@ class FinalSink(object):
         # single far veto for high-significance trigger
         # add an upper limit for the chisq for uploaded event compared to the last line, hardcoded to have u
         ifo_active = [
-            self.candidate.chisq_H != 0 and self.candidate.chisq_H < 3,
-            self.candidate.chisq_L != 0 and self.candidate.chisq_L < 3,
-            self.candidate.chisq_V != 0 and self.candidate.chisq_V < 3
+            chisq != 0 and chisq < 3 for chisq in self.candidate.chisq
         ]
         ifo_fars_ok = [
-            self.candidate.far_h < self.singlefar_veto_thresh
-            and self.candidate.far_h > 0.,
-            self.candidate.far_l < self.singlefar_veto_thresh
-            and self.candidate.far_l > 0.,
-            self.candidate.far_v < self.singlefar_veto_thresh
-            and self.candidate.far_v > 0.
-        ]
-        ifo_chisqs = [
-            self.candidate.chisq_H, self.candidate.chisq_L,
-            self.candidate.chisq_V
+            far < self.singlefar_veto_thresh and far > 0.
+            for far in self.candidate.far_sngl
         ]
         if self.candidate.far < self.superevent_thresh:
             return sum([
                 i for (i, v) in zip(ifo_fars_ok, ifo_active) if v
             ]) >= 2 and all(
                 (lambda x:
```

```
-                [i1 / i2 < self.chisq_ratio_thresh for i1 in x for i2 in x])(
-                    [i for (i, v) in zip(ifo_chisqs, ifo_active) if v]))
+                [i1 / i2 < self.chisq_ratio_thresh for i1 in x for i2 in x])([
+                    i for (i, v) in zip(self.candidate.chisq, ifo_active) if v
+                ]))
     def appsink_new_buffer(self, elem):
         with self.lock:
@@ -739,12 +731,11 @@ class FinalSink(object):
     def __set_far(self, candidate):
         candidate.far = (max(candidate.far_2h, candidate.far_1d,
                              candidate.far_1w)) * self.far_factor
-        candidate.far_h = (max(candidate.far_h_2h, candidate.far_h_1d,
--                               candidate.far_h_1w)) * self.far_factor
-        candidate.far_l = (max(candidate.far_l_2h, candidate.far_l_1d,
--                               candidate.far_l_1w)) * self.far_factor
-        candidate.far_v = (max(candidate.far_v_2h, candidate.far_v_1d,
--                               candidate.far_v_1w)) * self.far_factor
+        candidate.far_sngl = [
+            (max(fars) * self.far_factor)
+            for fars in zip(candidate.far_2h_sngl, candidate.far_1d_sngl,
+                            candidate.far_1w_sngl)
+        ]
     # def __lookback_far(self, candidate):
     # FIXME: hard-code to check event that's < 5e-7
diff --git a/gstlal-spiir/python/pipemodules/postcohtable/Makefile.am b/gstlal-spiir/python/pipemodules/postco
index 7038ccb4..23df433d 100644
--- a/gstlal-spiir/python/pipemodules/postcohtable/Makefile.am
+++ b/gstlal-spiir/python/pipemodules/postcohtable/Makefile.am
@@ -17,7 +17,7 @@ postcohtable_PYTHON = \
 postcohtable_LTLIBRARIES = _postcohtable.la

 _postcohtable_la_SOURCES = _postcohtable.c
-_postcohtable_la_CPPFLAGS = $(AM_CPPFLAGS) $(PYTHON_CPPFLAGS) -DMODULE_NAME="\"gstlal.pipemodules.postcohtab
+_postcohtable_la_CPPFLAGS = $(AM_CPPFLAGS) $(LAL_CFLAGS) $(GSL_CFLAGS) $(gstreamer_CFLAGS) $(GSTLAL_CFLAGS) $
 _postcohtable_la_CFLAGS = $(AM_CFLAGS) $(LAL_CFLAGS) $(GSL_CFLAGS) $(gstreamer_CFLAGS) $(GSTLAL_CFLAGS) $(ADI
 _postcohtable_la_LDFLAGS = $(AM_LDFLAGS) $(LAL_LIBS) $(GSL_LIBS) $(GSTLAL_LDFLAGS) $(PYTHON_LIBS) $(ADD_LIBS)
diff --git a/gstlal-spiir/python/pipemodules/postcohtable/_postcohtable.c b/gstlal-spiir/python/pipemodules/p
index 5ec05962..6ce527a0 100644
--- a/gstlal-spiir/python/pipemodules/postcohtable/_postcohtable.c
+++ b/gstlal-spiir/python/pipemodules/postcohtable/_postcohtable.c
@@ -24,6 +24,7 @@
  * ============================================================================
  */

+#include <string.h>
 #define NPY_NO_DEPRECATED_API NPY_1_7_API_VERSION

 #define PY_SSIZE_T_CLEAN
@@ -31,6 +32,7 @@
 #include <lal/TimeSeries.h>
 #include <lal/Units.h>
 #include <numpy/ndarrayobject.h>
+#include <pipe_macro.h>
 #include <postcohtable.h>
 #include <structmember.h>

@@ -49,6 +51,15 @@
 typedef struct {
     PyObject_HEAD PostcohInspiralTable row;
     COMPLEX8TimeSeries *snr;
+    PyObject *end_time_sngl;
+    PyObject *snglsnr;
+    PyObject *coaphase;
+    PyObject *chisq;
+    PyObject *far_sngl;
+    PyObject *far_1w_sngl;
+    PyObject *far_1d_sngl;
+    PyObject *far_2h_sngl;
+    PyObject *deff;
 } gstlal_GSTLALPostcohInspiral;

 // static PyObject *row_event_id_type = NULL;
@@ -58,49 +69,14 @@ typedef struct {
  * Member access
```

```
  */
-static struct PyMemberDef members[] = {
+static PyMemberDef members[] = {
+    // Not dependent on the number of detectors
     { "end_time", T_INT,
       offsetof(gstlal_GSTLALPostcohInspiral, row.end_time.gpsSeconds), 0,
       "end_time" },
     { "end_time_ns", T_INT,
       offsetof(gstlal_GSTLALPostcohInspiral, row.end_time.gpsNanoSeconds), 0,
       "end_time_ns" },
-    { "end_time_L", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_L.gpsSeconds), 0,
-      "end_time_L" },
-    { "end_time_ns_L", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_L.gpsNanoSeconds), 0,
-      "end_time_ns_L" },
-    { "end_time_H", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_H.gpsSeconds), 0,
-      "end_time_H" },
-    { "end_time_ns_H", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_H.gpsNanoSeconds), 0,
-      "end_time_ns_H" },
-    { "end_time_V", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_V.gpsSeconds), 0,
-      "end_time_V" },
-    { "end_time_ns_V", T_INT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_V.gpsNanoSeconds), 0,
-      "end_time_ns_V" },
-    { "snglsnr_L", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.snglsnr_L), 0, "snglsnr_L" },
-    { "snglsnr_H", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.snglsnr_H), 0, "snglsnr_H" },
-    { "snglsnr_V", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.snglsnr_V), 0, "snglsnr_V" },
-    { "coaphase_L", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.coaphase_L), 0, "coaphase_L" },
-    { "coaphase_H", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.coaphase_H), 0, "coaphase_H" },
-    { "coaphase_V", T_FLOAT,
-      offsetof(gstlal_GSTLALPostcohInspiral, row.coaphase_V), 0, "coaphase_V" },
-    { "chisq_L", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.chisq_L),
-      0, "chisq_L" },
-    { "chisq_H", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.chisq_H),
-      0, "chisq_H" },
-    { "chisq_V", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.chisq_V),
-      0, "chisq_V" },
     { "is_background", T_INT,
       offsetof(gstlal_GSTLALPostcohInspiral, row.is_background), 0,
       "is_background" },
@@ -131,30 +107,6 @@ static struct PyMemberDef members[] = {
       "far_1w" },
     { "far", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far), 0,
       "far" },
-    { "far_h", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_h), 0,
-      "far_h" },
-    { "far_l", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_l), 0,
-      "far_l" },
-    { "far_v", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_v), 0,
-      "far_v" },
-    { "far_h_1w", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_h_1w),
-      0, "far_h_1w" },
-    { "far_l_1w", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_l_1w),
-      0, "far_l_1w" },
-    { "far_v_1w", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_v_1w),
-      0, "far_v_1w" },
-    { "far_h_1d", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_h_1d),
-      0, "far_h_1d" },
-    { "far_l_1d", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_l_1d),
-      0, "far_l_1d" },
```

```
-    { "far_v_1d", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_v_1d),
-      0, "far_v_1d" },
-    { "far_h_2h", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_h_2h),
-      0, "far_h_2h" },
-    { "far_l_2h", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_l_2h),
-      0, "far_l_2h" },
-    { "far_v_2h", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.far_v_2h),
-      0, "far_v_2h" },
     { "rank", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.rank), 0,
       "rank" },
     { "template_duration", T_DOUBLE,
@@ -185,12 +137,6 @@ static struct PyMemberDef members[] = {
     { "ra", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.ra), 0, "ra" },
     { "dec", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.dec), 0,
       "dec" },
-    { "deff_L", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.deff_L), 0,
-      "deff_L" },
-    { "deff_H", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.deff_H), 0,
-      "deff_H" },
-    { "deff_V", T_DOUBLE, offsetof(gstlal_GSTLALPostcohInspiral, row.deff_V), 0,
-      "deff_V" },
     { "f_final", T_FLOAT, offsetof(gstlal_GSTLALPostcohInspiral, row.f_final),
       0, "f_final" },
     { "_process_id", T_LONG,
@@ -198,9 +144,28 @@ static struct PyMemberDef members[] = {
       "process_id (long)" },
     { "_event_id", T_LONG, offsetof(gstlal_GSTLALPostcohInspiral, row.event_id),
       0, "event_id (long)" },
-    {
-      NULL,
-    }
+
+    // Things that are done single detector are ndarrays
+    { "end_time_sngl", T_OBJECT_EX,
+      offsetof(gstlal_GSTLALPostcohInspiral, end_time_sngl), 0,
+      "end_time_sngl" },
+    { "snglsnr", T_OBJECT_EX, offsetof(gstlal_GSTLALPostcohInspiral, snglsnr),
+      0, "snglsnr" },
+    { "coaphase", T_OBJECT_EX, offsetof(gstlal_GSTLALPostcohInspiral, coaphase),
+      0, "coaphase" },
+    { "chisq", T_OBJECT_EX, offsetof(gstlal_GSTLALPostcohInspiral, chisq), 0,
+      "chisq" },
+    { "far_sngl", T_OBJECT_EX, offsetof(gstlal_GSTLALPostcohInspiral, far_sngl),
+      0, "far_sngl" },
+    { "far_1w_sngl", T_OBJECT_EX,
+      offsetof(gstlal_GSTLALPostcohInspiral, far_1w_sngl), 0, "far_1w_sngl" },
+    { "far_1d_sngl", T_OBJECT_EX,
+      offsetof(gstlal_GSTLALPostcohInspiral, far_1d_sngl), 0, "far_1d_sngl" },
+    { "far_2h_sngl", T_OBJECT_EX,
+      offsetof(gstlal_GSTLALPostcohInspiral, far_2h_sngl), 0, "far_2h_sngl" },
+    { "deff", T_OBJECT_EX, offsetof(gstlal_GSTLALPostcohInspiral, deff), 0,
+      "deff" },
+    { NULL },
 };
 struct pylal_inline_string_description {
@@ -210,7 +175,7 @@ struct pylal_inline_string_description {
 static PyObject *pylal_inline_string_get(PyObject *obj, void *data) {
     const struct pylal_inline_string_description *desc = data;
-    char *s = (void *)obj + desc->offset;
+    char *s = (char *)obj + desc->offset;
     if ((ssize_t)strlen(s) >= desc->length) {
         /* something's wrong, obj probably isn't a valid address */
@@ -222,7 +187,7 @@ static PyObject *pylal_inline_string_get(PyObject *obj, void *data) {
 static int pylal_inline_string_set(PyObject *obj, PyObject *val, void *data) {
     const struct pylal_inline_string_description *desc = data;
     char *v                                             = PyString_AsString(val);
-    char *s = (void *)obj + desc->offset;
+    char *s = (char *)obj + desc->offset;
     if (!v) return -1;
```

```
     if ((ssize_t)strlen(v) >= desc->length) {
@@ -238,7 +203,7 @@ static int pylal_inline_string_set(PyObject *obj, PyObject *val, void *data) {
 static PyObject *snr_component_get(PyObject *obj, void *data) {
     COMPLEX8TimeSeries *snr = ((gstlal_GSTLALPostcohInspiral *)obj)->snr;
-    const char *name        = data;
+    const char *name        = (const char *)data;
     if (!snr) {
         PyErr_SetString(PyExc_ValueError, "no snr time series available");
@@ -274,7 +239,8 @@ static PyObject *snr_component_get(PyObject *obj, void *data) {
     return NULL;
 }
-static struct PyGetSetDef getset[] = {
+#define SINGLE 11
+static struct PyGetSetDef getset[SINGLE + 10 * MAX_NIFO + 1] = {
     { "ifos", pylal_inline_string_get, pylal_inline_string_set, "ifos",
       &(struct pylal_inline_string_description) {
         offsetof(gstlal_GSTLALPostcohInspiral, row.ifos), MAX_ALLIFO_LEN } },
@@ -301,11 +267,215 @@ static struct PyGetSetDef getset[] = {
         "_snr_data_length" },
     { "_snr_data", snr_component_get, NULL, ".snr.data", "_snr_data" },
-    {
-      NULL,
-    }
+    { NULL }
 };
+struct lal_array {
+    Py_ssize_t offset;
+    Py_ssize_t index;
+};
+
+static PyObject *pylal_double_array_get(PyObject *obj, void *data) {
+    const struct lal_array *desc = data;
+    double *d = (double *)((char *)obj + desc->offset) + desc->index;
+    if (!d) {
+        PyErr_Format(PyExc_ValueError, "float doesn't exist!");
+        return NULL;
+    }
+    return PyFloat_FromDouble(*d);
+}
+
+static int pylal_double_array_set(PyObject *obj, PyObject *val, void *data) {
+    const struct lal_array *desc = data;
+    double v                     = PyFloat_AsDouble(val);
+    double *d = (double *)((char *)obj + desc->offset) + desc->index;
+    if (!d) {
+        PyErr_Format(PyExc_ValueError, "float doesn't exist!");
+        return -1;
+    }
+    *d = v;
+    return 0;
+}
+
+static PyObject *pylal_float_array_get(PyObject *obj, void *data) {
+    const struct lal_array *desc = data;
+    float *f = (float *)((char *)obj + desc->offset) + desc->index;
+    if (!f) {
+        PyErr_Format(PyExc_ValueError, "float doesn't exist!");
+        return NULL;
+    }
+    return PyFloat_FromDouble((double)*f);
+}
+
+static int pylal_float_array_set(PyObject *obj, PyObject *val, void *data) {
+    const struct lal_array *desc = data;
+    double v                     = PyFloat_AsDouble(val);
+    float *f = (float *)((char *)obj + desc->offset) + desc->index;
+    if (!f) {
+        PyErr_Format(PyExc_ValueError, "float doesn't exist!");
+        return -1;
+    }
```

```
+    *f = (float)v;
+    return 0;
+}
+
+static PyObject *pylal_int_array_get(PyObject *obj, void *data) {
+    const struct lal_array *desc = data;
+    int *i = (int *)((char *)obj + desc->offset) + desc->index;
+    if (!i) {
+        PyErr_Format(PyExc_ValueError, "int doesn't exist!");
+        return NULL;
+    }
+    return PyInt_FromLong((long)*i);
+}
+
+static int pylal_int_array_set(PyObject *obj, PyObject *val, void *data) {
+    const struct lal_array *desc = data;
+    int v                        = (int)PyInt_AsLong(val);
+    int *i = (int *)((char *)obj + desc->offset) + desc->index;
+    if (!i) {
+        PyErr_Format(PyExc_ValueError, "float doesn't exist!");
+        return -1;
+    }
+    *i = (int)v;
+    return 0;
+}
+
+void prepare_getset() {
+    int offset = SINGLE;
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        char *var  = "chisq_";
+        char *name = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+        struct lal_array *data =
+          (struct lal_array *)malloc(sizeof(struct lal_array));
+        data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.chisq);
+        data->index  = i;
+        strcpy(name, var);
+        strcat(name, IFOMap[i].name);
+        PyGetSetDef def  = { name, pylal_float_array_get, pylal_float_array_set,
+                             name, data };
+        getset[offset++] = def;
+
+        var          = "snglsnr_";
+        name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+        data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+        data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.snglsnr);
+        data->index  = i;
+        strcpy(name, var);
+        strcat(name, IFOMap[i].name);
+        def.name         = name;
+        def.doc          = name;
+        def.closure      = data;
+        getset[offset++] = def;
+
+        var          = "coaphase_";
+        name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+        data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+        data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.coaphase);
+        data->index  = i;
+        strcpy(name, var);
+        strcat(name, IFOMap[i].name);
+        def.name         = name;
+        def.doc          = name;
+        def.closure      = data;
+        getset[offset++] = def;
+
+        var          = "far_sngl_";
+        name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+        data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+        data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.far_sngl);
+        data->index  = i;
+        strcpy(name, var);
+        strcat(name, IFOMap[i].name);
+        def.name         = name;
+        def.doc          = name;
```

```
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var          = "far_1d_sngl_";
+       name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.far_1d_sngl);
+       data->index  = i;
+       strcpy(name, var);
+       strcat(name, IFOMap[i].name);
+       def.name         = name;
+       def.doc          = name;
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var          = "far_1w_sngl_";
+       name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.far_1w_sngl);
+       data->index  = i;
+       strcpy(name, var);
+       strcat(name, IFOMap[i].name);
+       def.name         = name;
+       def.doc          = name;
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var          = "far_2h_sngl_";
+       name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.far_2h_sngl);
+       data->index  = i;
+       strcpy(name, var);
+       strcat(name, IFOMap[i].name);
+       def.name         = name;
+       def.doc          = name;
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var          = "deff_";
+       name         = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data         = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset = offsetof(gstlal_GSTLALPostcohInspiral, row.deff);
+       data->index  = i;
+       strcpy(name, var);
+       strcat(name, IFOMap[i].name);
+       def.name         = name;
+       def.get          = pylal_double_array_get;
+       def.set          = pylal_double_array_set;
+       def.doc          = name;
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var  = "end_time_sngl_";
+       name = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset =
+         offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_sngl);
+       data->index = i * 2;
+       strcpy(name, var);
+       strcat(name, IFOMap[i].name);
+       def.name         = name;
+       def.get          = pylal_int_array_get;
+       def.set          = pylal_int_array_set;
+       def.doc          = name;
+       def.closure      = data;
+       getset[offset++] = def;
+
+       var  = "end_time_ns_sngl_";
+       name = (char *)malloc(strlen(IFOMap[i].name) + strlen(var) + 1);
+       data = (struct lal_array *)malloc(sizeof(struct lal_array));
+       data->offset =
+         offsetof(gstlal_GSTLALPostcohInspiral, row.end_time_sngl);
+       data->index = i * 2 + 1;
+       strcpy(name, var);
```

```
+        strcat(name, IFOMap[i].name);
+        def.name         = name;
+        def.get          = pylal_int_array_get;
+        def.set          = pylal_int_array_set;
+        def.doc          = name;
+        def.closure      = data;
+        getset[offset++] = def;
+    }
+    PyGetSetDef def = { NULL };
+    getset[offset]  = def;
+}
+
 // static Py_ssize_t getreadbuffer(PyObject *self, Py_ssize_t segment, void
 // **ptrptr)
 //{
@@ -339,10 +509,10 @@ static struct PyGetSetDef getset[] = {
   */
 static PyObject *__new__(PyTypeObject *type, PyObject *args, PyObject *kwds) {
-    gstlal_GSTLALPostcohInspiral *new =
+    gstlal_GSTLALPostcohInspiral *ret =
       (gstlal_GSTLALPostcohInspiral *)PyType_GenericNew(type, args, kwds);
-    if (!new) return NULL;
+    if (!ret) return NULL;
     /* link the event_id pointer in the row table structure
      * to the event_id structure */
@@ -352,13 +522,22 @@ static PyObject *__new__(PyTypeObject *type, PyObject *args, PyObject *kwds) {
     // new->event_id_i = 0;
     /* done */
-    return (PyObject *)new;
+    return (PyObject *)ret;
 }
 static void __del__(PyObject *self) {
-    if (((gstlal_GSTLALPostcohInspiral *)self)->snr)
-        XLALDestroyCOMPLEX8TimeSeries(
-          ((gstlal_GSTLALPostcohInspiral *)self)->snr);
+    gstlal_GSTLALPostcohInspiral *typed_self =
+      (gstlal_GSTLALPostcohInspiral *)self;
+    if (typed_self->snr) XLALDestroyCOMPLEX8TimeSeries(typed_self->snr);
+    Py_DECREF(typed_self->end_time_sngl);
+    Py_DECREF(typed_self->snglsnr);
+    Py_DECREF(typed_self->coaphase);
+    Py_DECREF(typed_self->chisq);
+    Py_DECREF(typed_self->far_sngl);
+    Py_DECREF(typed_self->far_1w_sngl);
+    Py_DECREF(typed_self->far_1d_sngl);
+    Py_DECREF(typed_self->far_2h_sngl);
+    Py_DECREF(typed_self->deff);
     Py_TYPE(self)->tp_free(self);
 }
@@ -366,6 +545,8 @@ static PyObject *from_buffer(PyObject *cls, PyObject *args) {
     const char *data;
     Py_ssize_t length;
     PyObject *result;
+    npy_intp dims[1]          = { MAX_NIFO };
+    npy_intp end_time_dims[2] = { 2, MAX_NIFO };
     if (!PyArg_ParseTuple(args, "s#", (const char **)&data, &length))
         return NULL;
@@ -397,6 +578,36 @@ static PyObject *from_buffer(PyObject *cls, PyObject *args) {
          * gstlal_GSTLALPostcohInspiral item*/
         ((gstlal_GSTLALPostcohInspiral *)item)->row =
           (PostcohInspiralTable)*gstlal_postcohinspiral;
+
+        // Set the single-detector arrays
+        ((gstlal_GSTLALPostcohInspiral *)item)->end_time_sngl =
+          PyArray_SimpleNewFromData(1, end_time_dims, NPY_INT,
+                                    gstlal_postcohinspiral->end_time_sngl);
+        ((gstlal_GSTLALPostcohInspiral *)item)->snglsnr =
+          PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
```

```
+                                       gstlal_postcohinspiral->snglsnr);
+       ((gstlal_GSTLALPostcohInspiral *)item)->coaphase =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->coaphase);
+       ((gstlal_GSTLALPostcohInspiral *)item)->chisq =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->chisq);
+       ((gstlal_GSTLALPostcohInspiral *)item)->far_sngl =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->far_sngl);
+       ((gstlal_GSTLALPostcohInspiral *)item)->far_1w_sngl =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->far_1w_sngl);
+       ((gstlal_GSTLALPostcohInspiral *)item)->far_1d_sngl =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->far_1d_sngl);
+       ((gstlal_GSTLALPostcohInspiral *)item)->far_2h_sngl =
+         PyArray_SimpleNewFromData(1, dims, NPY_FLOAT,
+                                       gstlal_postcohinspiral->far_2h_sngl);
+       ((gstlal_GSTLALPostcohInspiral *)item)->deff =
+         PyArray_SimpleNewFromData(1, dims, NPY_DOUBLE,
+                                       gstlal_postcohinspiral->deff);
+
        /* duplicate the SNR time series if we have length? */
        if (gstlal_postcohinspiral->snr_length) {
            const size_t nbytes = sizeof(gstlal_postcohinspiral->snr[0])
@@ -479,8 +690,20 @@ PyMODINIT_FUNC init_postcohtable(void) {
     PyObject *module = Py_InitModule3(
       MODULE_NAME, NULL, "Wrapper for LAL's PostcohInspiralTable type.");
+    prepare_getset();
     import_array();
+    PyObject *ifo_map = PyList_New(MAX_NIFO);
+    Py_INCREF(ifo_map);
+    for (int i = 0; i < MAX_NIFO; ++i) {
+        PyObject *str =
+          PyString_FromStringAndSize(IFOMap[i].name, strlen(IFOMap[i].name));
+        assert(str);
+        Py_INCREF(str);
+        PyList_SetItem(ifo_map, i, str);
+    }
+    PyModule_AddObject(module, "ifo_map", ifo_map);
+
     /* Cached ID types */
     // process_id_type = pylal_get_ilwdchar_class("process", "process_id");
     // row_event_id_type = pylal_get_ilwdchar_class("postcoh", "event_id");
diff --git a/gstlal-spiir/python/pipemodules/postcohtable/postcoh_table_def.py b/gstlal-spiir/python/pipemodu
index e7d88349..45e7034f 100644
--- a/gstlal-spiir/python/pipemodules/postcohtable/postcoh_table_def.py
+++ b/gstlal-spiir/python/pipemodules/postcohtable/postcoh_table_def.py
@@ -4,84 +4,78 @@ from glue.ligolw import ilwd
 from glue.ligolw import dbtables
 from lal import LIGOTimeGPS
 from xml.sax.xmlreader import AttributesImpl
+from itertools import chain
 # so they can be inserted into a database
 dbtables.ligolwtypes.ToPyType["ilwd:char"] = unicode

 PostcohInspiralID = ilwd.get_ilwdchar_class(u"postcoh", u"event_id")

+import postcohtable
+
 # need to be consistent with the table defined in postcohinspiral_table.h
 class PostcohInspiralTable(table.Table):
     tableName = "postcoh"
-    validcolumns = {
-        "process_id": "ilwd:char",
-        "event_id": "ilwd:char",
-        "end_time": "int_4s",
-        "end_time_ns": "int_4s",
-        "end_time_L": "int_4s",
-        "end_time_ns_L": "int_4s",
-        "end_time_H": "int_4s",
```

```
-         "end_time_ns_H": "int_4s",
-         "end_time_V": "int_4s",
-         "end_time_ns_V": "int_4s",
-         "snglsnr_L": "real_4",
-         "snglsnr_H": "real_4",
-         "snglsnr_V": "real_4",
-         "coaphase_L": "real_4",
-         "coaphase_H": "real_4",
-         "coaphase_V": "real_4",
-         "chisq_L": "real_4",
-         "chisq_H": "real_4",
-         "chisq_V": "real_4",
-         "is_background": "int_4s",
-         "livetime": "int_4s",
-         "ifos": "lstring",
-         "pivotal_ifo": "lstring",
-         "tmplt_idx": "int_4s",
-         "bankid": "int_4s",
-         "pix_idx": "int_4s",
-         "cohsnr": "real_4",
-         "nullsnr": "real_4",
-         "cmbchisq": "real_4",
-         "spearman_pval": "real_4",
-         "fap": "real_4",
-         "far_h": "real_4",
-         "far_l": "real_4",
-         "far_v": "real_4",
-         "far_h_1w": "real_4",
-         "far_l_1w": "real_4",
-         "far_v_1w": "real_4",
-         "far_h_1d": "real_4",
-         "far_l_1d": "real_4",
-         "far_v_1d": "real_4",
-         "far_h_2h": "real_4",
-         "far_l_2h": "real_4",
-         "far_v_2h": "real_4",
-         "far": "real_4",
-         "far_2h": "real_4",
-         "far_1d": "real_4",
-         "far_1w": "real_4",
-         "skymap_fname": "lstring",
-         "template_duration": "real_8",
-         "mass1": "real_4",
-         "mass2": "real_4",
-         "mchirp": "real_4",
-         "mtotal": "real_4",
-         "spin1x": "real_4",
-         "spin1y": "real_4",
-         "spin1z": "real_4",
-         "spin2x": "real_4",
-         "spin2y": "real_4",
-         "spin2z": "real_4",
-         "eta": "real_4",
-         "f_final": "real_4",
-         "ra": "real_8",
-         "dec": "real_8",
-         "deff_L": "real_8",
-         "deff_H": "real_8",
-         "deff_V": "real_8",
-         "rank": "real_8"
-     }
+     validcolumns = dict(
+         chain(
+             [
+                 ("process_id", "ilwd:char"),
+                 ("event_id", "ilwd:char"),
+                 ("end_time", "int_4s"),
+                 ("end_time_ns", "int_4s"),
+                 ("is_background", "int_4s"),
+                 ("livetime", "int_4s"),
+                 ("ifos", "lstring"),
+                 ("pivotal_ifo", "lstring"),
+                 ("tmplt_idx", "int_4s"),
+                 ("bankid", "int_4s"),
+                 ("pix_idx", "int_4s"),
```

```
+                ("cohsnr", "real_4"),
+                ("nullsnr", "real_4"),
+                ("cmbchisq", "real_4"),
+                ("spearman_pval", "real_4"),
+                ("fap", "real_4"),
+                ("far", "real_4"),
+                ("far_2h", "real_4"),
+                ("far_1d", "real_4"),
+                ("far_1w", "real_4"),
+                ("skymap_fname", "lstring"),
+                ("template_duration", "real_8"),
+                ("mass1", "real_4"),
+                ("mass2", "real_4"),
+                ("mchirp", "real_4"),
+                ("mtotal", "real_4"),
+                ("spin1x", "real_4"),
+                ("spin1y", "real_4"),
+                ("spin1z", "real_4"),
+                ("spin2x", "real_4"),
+                ("spin2y", "real_4"),
+                ("spin2z", "real_4"),
+                ("eta", "real_4"),
+                ("f_final", "real_4"),
+                ("ra", "real_8"),
+                ("dec", "real_8"),
+                ("rank", "real_8"),
+            ],
+            list(("deff_" + name, "real_8") for name in postcohtable.ifo_map),
+            list(("far_sngl_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("far_2h_sngl_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("far_1d_sngl_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("far_1w_sngl_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("snglsnr_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("coaphase_" + name, "real_4")
+                 for name in postcohtable.ifo_map),
+            list(("chisq_" + name, "real_4") for name in postcohtable.ifo_map),
+            list(("end_time_sngl_" + name, "int_4s")
+                 for name in postcohtable.ifo_map),
+            list(("end_time_ns_sngl_" + name, "int_4s")
+                 for name in postcohtable.ifo_map),
+        ))
      constraints = "PRIMARY KEY (event_id)"
      next_id = PostcohInspiralID(0)
diff --git a/gstlal-spiir/python/pipemodules/postcohtable/postcohtable.py b/gstlal-spiir/python/pipemodules/p
index d1a2c31b..b43183f3 100644
--- a/gstlal-spiir/python/pipemodules/postcohtable/postcohtable.py
+++ b/gstlal-spiir/python/pipemodules/postcohtable/postcohtable.py
@@ -21,7 +21,9 @@ from glue.ligolw import lsctables
 import lal
 from . import _postcohtable
-__all__ = ["GSTLALPostcohInspiral"]
+__all__ = ["GSTLALPostcohInspiral", "ifo_map"]
+
+ifo_map = _postcohtable.ifo_map
 class GSTLALPostcohInspiral(_postcohtable.GSTLALPostcohInspiral):
--
GitLab
```