GPU-Optimised Low-Latency Online Search for Gravitational Waves from Binary Coalescences

Xiaoyang Guo*, Qi Chu†, Zhihui Du* and Linqing Wen†
*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Email: xiaoyang.guo1995@gmail.com, duzh@tsinghua.edu.cn
†OzGrav-UWA, Department of Physics and Astrophysics, University of Western Australia, Crawley, WA 6009, Australia
Email: {qi.chu,linqing.wen}@uwa.edu.au

***Abstract*—Low-latency detection of gravitational waves (GWs) from compact stellar mergers is crucial to enable prompt follow-up electro-magnetic (EM) observations, as to probe different aspects of the merging process. The GW signal detection involves large computational efforts to search over the merger parameter space and Graphics Processing Unit (GPU) can play an important role to parallel the process. In this paper, Summed Parallel Infinite Impulse Response (SPIIR) GW detection pipeline is further optimized using recent GPU techniques to improve its throughput and reduce its latency. Two main computational bottlenecks have been studied: the SPIIR filtering and the coherent post-processing which combines multiple GW detector outputs. In the filtering part, inefficient memory access is accelerated by exploiting temporal locality of input data, where the performance over previous implementation is improved by a factor of 2.5-3.5x on different GPUs. The post-processing part is improved by employing multiple strategies and a speedup of 12-25x is achieved. Once again, it is shown that GPUs can be very useful to tackle computational challenges in GW detection.**

I. Introduction

The first direct observation of gravitational waves (GWs) were made in September 2015 by the Laser Interferometric Gravitational-Wave Observatory (LIGO) [1]. Since then, five GWs have been observed with high confidence including the first GW observation from a binary neutron star merger [2]. It is expected there will be more events in the coming years when LIGO and other GW detectors finish upgrading and come into operation in better sensitivities. It is among the highest priorities of the GW community to be able to observe these events without delay, to alert other electromagnetic channels for joint observations.

Several low-latency detection pipelines have been developed in place which include the MBTA pipeline [3], the LLOID pipeline [4], the PyCBC pipeline [5] and the Summed Parallel Infinite Impulse Response (SPIIR) pipeline [6]. Each pipeline has a complete procedure of dealing with raw streaming data and submitting GW triggers to GW database. All of them have achieved latencies of sub-minute during the last LIGO science run.

The SPIIR coherent search pipeline (see Fig. 1) that we are studying in this work mainly consists of six parts: data loading, data whitening, SPIIR filtering, coherent post-processing, veto and clustering of candidate events, candidate submission to the GW database. The SPIIR method utilizes a group of IIR filters to approximate a traditional matched filter. The result of the filtering output is an approximation of the signal-to-noise ratio (SNR). The coherent post-processing will combine SNRs from different detectors coherently to form a single statistic which will be used to evaluate the significance of a candidate. The main computation of the pipeline lies in the filtering part and the coherent post-processing part, which account for 52% and 36% of the total time consumption respectively. Previously, we explored GPUs to accelerate the filtering part of the pipeline [7], [8] and achieved an over 100x speedup on this part [8]. This work will try to further accelerate both parts with recent GPU features.

Nvidia GPUs have evolved from the Tesla architecture to the state of the art Volta architecture with their single-precision performance and memory bandwidth growing almost exponentially. Recently some useful advanced features have been supported, such as half precision arithmetic operations and atomic operations, which will help the flexibility and efficiency of parallel programs. However, it is more difficult to fully utilize GPUs to accelerate an application compared to acceleration using multiple cores of CPUs since it involves utilization of various levels of memory access and synchronizations between different levels of thread aggregation.

The structure of this paper is as follows: In Sec. II, the optimization for the SPIIR filtering is introduced. The bottleneck is inefficient memory access and is accelerated using vectorized memory access, which exploits temporal locality of data access and improves memory access efficiency. Sec. III shows the optimization for the coherent post-processing. The post-processing part is optimized iteratively using multiple strategies. In Sec. IV, detailed experiments are carried out to analyze performance improvement. Conclusions and the future works are given in Sec. V.

II. Optimization on SPIIR Filtering

A. Algorithm

The SPIIR method uses a group of IIR filters with time delays to approximate a given matched filter. Each IIR filter corresponds to a small segment of the matched filter (see details in [6]). The filters will then operate on the data and the filtering output is signal-to-noise ratio (SNR).

For simplicity, we only consider for one template. The output of the lth IIR filter can be expressed as:

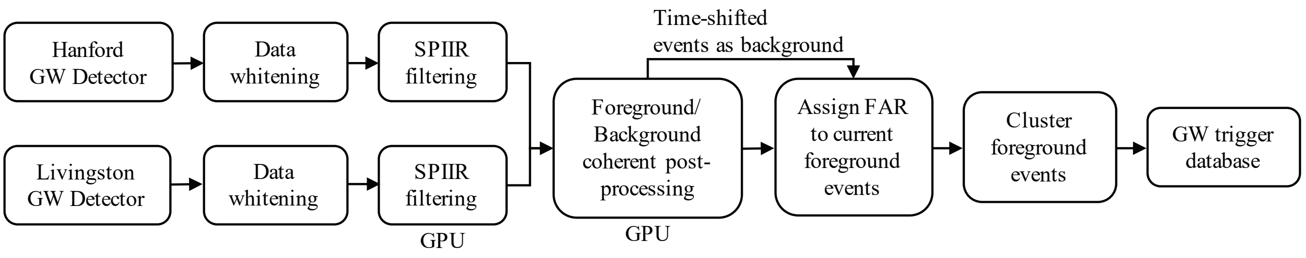

Hanford GW Detector
Livingston GW Detector
Data whitening
Data whitening
SPIIR filtering
SPIIR filtering
GPU
Time-shifted events as background
Foreground/ Background coherent post-processing
GPU
Assign FAR to current foreground events
Cluster foreground events
GW trigger database

Fig. 1. SPIIR gravitational wave detection pipeline

$$y_k^l = a_1^l y_{k-1}^l + b_0^l x_{k-d_l} \tag{1}$$

where a_1^l and b_0^l are IIR coefficients and k denotes time in discrete form. x_{k-d_l} denotes input with a time delay d_l. Then the filtering outputs of the template are given by summing up the outputs of all IIR filters in this template:

$$\hat{z}_k = 2\Delta t \sum_l y_k^l \tag{2}$$

Every time the filtering part is executed, one second whitened data from individual detectors are processed sequentially. N_t times calculations are needed for each filter, where N_t denotes the sampling rate of the input data.

B. Analysis of previous implementation

In previous CUDA implementation [8], one template is allocated to one thread block and the calculation for each thread block is shown in Fig. 2a. Each thread corresponds to one IIR filter and loops for N_t times to get filtering outputs.

Warp-shuffle operations[1] and read-only data cache have been used in previous implementation to improve reduction efficiency and data loading performance. However, the implementation still suffers from memory access issue because the input data $\{x_{k-d_l}\}$ are not continuous in device memory and could not be loaded efficiently by coalesced memory access[2]. As the average difference of time delays of IIR filters $\Delta d = |d_l - d_{l+1}|$ grows, more memory transactions are needed. Read-only data cache can only partially solve the problem because it suffers from the large amount of data and low cache hit rate.

C. Optimization method

Temporal locality is explored to optimize the program. For the lth thread, data inputs x_{k-d_l}, x_{k+1-d_l}, x_{k+2-d_l}, ... are loaded sequentially in the loop and located in continuous memory space. In order to better utilize the temporal locality characteristic, we use vectorized data type to load multiple consecutive data inputs at the same time, as shown in Fig. 2b.

In our implementation, vectorized data type *float4* is used to load 4 data inputs at the same time. The original loop is replaced by a nested loop. Data inputs are batch loaded in the outer loop and IIR filtering for these four inputs is done in the unrolled inner loop sequentially.

The inner loop need to be unrolled to make sure that vectorized data inputs stay in registers instead of local memory. Instruction throughput is also improved by unrolling the loop. By this method, the number of memory transactions is decreased and the memory access efficiency is improved.

III. Optimization on coherent post-processing

A. Algorithm

The principle of frequentist coherent search of GWs over multiple detectors follows the maximum likelihood ratio principle [9]–[11]. Recently it has been shown that four parameters of the binary source, denoted as A_{jk} can be maximized using a singular value decomposition method [11]. The rest of parameters can be searched using brute-force method.

We denote here the multi-detector maximum log likelihood ratio as the coherent SNR ρ_c^2. It can be shown that [11]:

$$\rho_c^2 = \frac{1}{1\text{Mpc}} \underset{max\{A_{jk},\theta,t_c,\alpha,\delta\}}{\ln\mathcal{L}_{NW}}, \tag{3}$$

where 1Mpc is one mega parsec in distance, A_{jk} is the four parameters describing the relative inclination of the detector to the source, θ is the mass of the source, α, β are the sky directions of the source, and $\mathcal{L}_{NW}$ is the network log likelihood ratio. The maximized network log likelihood ratio can be written as (see details in [11]):

$$\underset{max\{A_{jk}\}}{\ln\mathcal{L}_{NW}} = \|IU^T Z\|^2, \tag{4}$$

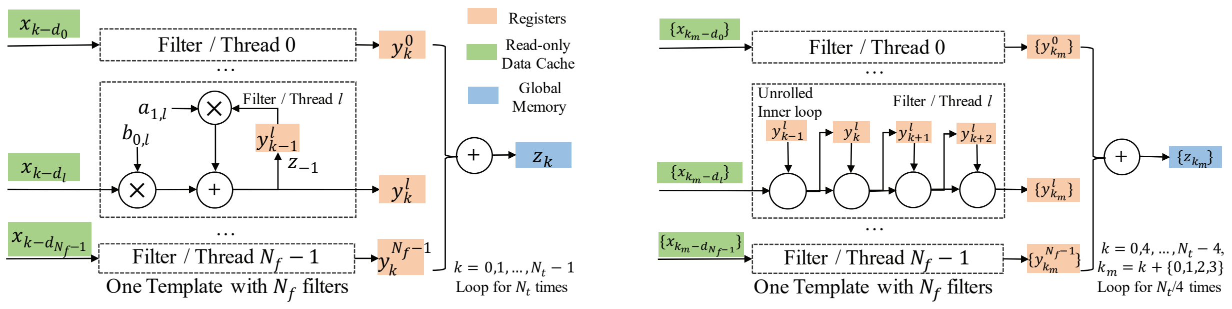
where $I = \text{diag}\{1, 1, 0, 0, \ldots\}$, U is the U matrix of the singular value decomposition of detector response matrix [11]. $Z^T = (z^{(1)}, z^{(2)}, \ldots, z^{(N_d)})$ is the filtering outputs from each detector. N_d denotes the number of detectors in the network.

Another statistic that will be useful for glitch veto from the coherent search is the residual SNR, which is usually called the null SNR. When there is a GW received by multiple detectors, the null SNR should follow central χ^2 distribution. The null SNR is given as:

$$\rho^2_{\text{NULL}} = \frac{1}{1\text{Mpc}} \underset{\{\hat{\theta},\hat{t_c},\hat{\alpha},\hat{\delta}\}}{\|I^\dagger U^T Z\|^2} \tag{5}$$

[1]Warp-shuffle operations provide a faster mechanism to exchange a variable between threads in the same warp without using shared memory.

[2]Memory accesses of threads in a warp can be coalesced to decrease memory transfer times depending on the data address distribution.



(a) Before optimization

(b) After optimization

Fig. 2. SPIIR filtering before and after optimization

where $I^\dagger = \mathrm{diag}\{0,0,1,1,\ldots\}$ and $\hat{\theta}, \hat{t_c}, \hat{\alpha}, \hat{\delta}$ are parameters determined by ρ_c in (3).

B. *Task of coherent post-processing*

In coherent post-processing, for one second filtering results, ten to hundreds of GW event candidates $\{(\hat{t_i}, \hat{\theta_i})\}$ are first proposed for each detector to reduce the brute-force search space of (3). For each candidate, all-sky search is performed for both foreground and background events to get coherent SNR, null SNR and corresponding sky direction. Background events are used to calculate statistics for the following FAR vetoing. For foreground event $(\hat{t}, \hat{\theta})$, corresponding time-shifted background events are $\{(\hat{t} - k\Delta t, \hat{\theta}) | 1 \le k \le N_{bg}\}$.

C. *Optimization method*

The coherent post-processing part is optimized iteratively and detailed optimization analysis is as follows.

1) Remove Synchronization: In previous implementation (see Fig. 3b), foreground and background event search for one candidate are processed in the same CUDA block. For each all-sky search, all threads in the block are used to calculate the maximum coherent SNR ρ_c and synchronizations are required by block-level max-reduction, which consumes a large mount of time.

Considering the computation time for foreground events could be ignored, we only optimize for background events. To avoid synchronization, we rearrange the computation and perform all-sky search only within warps (see Fig. 3c). Then the maximum coherent SNR could be got by warp-shuffle operations, instead of block-level reduction. Then the number of thread blocks for the new background kernel becomes:

$$N_{\mathrm{blocks-new}} = \left\lceil \frac{CN_{bg}}{N_{\mathrm{warps}}} \right\rceil \tag{6}$$

where $N_{\mathrm{warps}} = N_{\mathrm{threads}}/32$ denotes the number of warps per block and C is the number of candidates. There are totally CN_{bg} background event search tasks for all candidates and these tasks are numbered from 0 to $CN_{bg} - 1$. The jth warp of the ith thread block performs the all-sky serach for the $(iN_{\mathrm{warp}} + j)$th task.

After optimization, since all event searches are performed within warps, maximum coherent SNR can be calculated by warp-shuffle operations without explicit block-level synchronizations, by which time wasted by synchronizations is saved.

2) Arithmetic optimization: The program performance is also limited by the hardware arithmetic instruction throughput because of some computationally intensive codes.

In order to optimize arithmetic bottlenecks, the algorithm is first improved to remove useless or repeated calculations. Then intrinsic arithmetic functions are used to replace the regular ones, for example normal division operation *x/y* could be replaced by intrinsic function *__fdividef(x,y)* in CUDA. Intrinsic arithmetic functions are faster but less accurate. After replacing with intrinsic functions, experiments are carried out to verify that the numeric error resulted by intrinsic functions is small enough and would not affect the detection accuracy.

3) Local memory optimization: To calculate the log likelihood ratio in (4) and the null statistics in (5), intermediate results of $U^T Z$ are saved into a local array with the size of N_d. The problem is that the local array is accessed with dynamic loop indexes, which makes the array allocated in local memory instead of registers. Local memory actually resides in device memory, so access to local memory is time-consuming and increases the pressure of device memory.

To avoid using local memory, instead of saving intermediate results in local array, we directly add the results to the final $\mathcal{L}_{NW}$ and ρ^2_{NULL}, which is illustrated in Fig. 4. The number of device memory transactions is decreased by the simple modification.

4) Coalesced memory access: In our original implementation, SNR values from the SPIIR filtering are stored in the form of $N_t \times N_m$, where N_m denotes the number of the templates in a template bank. This means that SNR values of all templates with the same time are contiguous in memory. However, in coherent post-processing, SNR values along time of the same template are accessed at the same time, so memory access cannot be coalesced and the memory efficiency is low.

In order to accelerate memory access, the SNR matrix is transposed into the form of $N_m \times N_t$ before executing coherent post-processing. Then memory access to SNR values can be coalesced and the number of memory transactions is decreased.

5) Improve occupancy: The theoretical occupancy of the program is only 50% because of too much register usage.

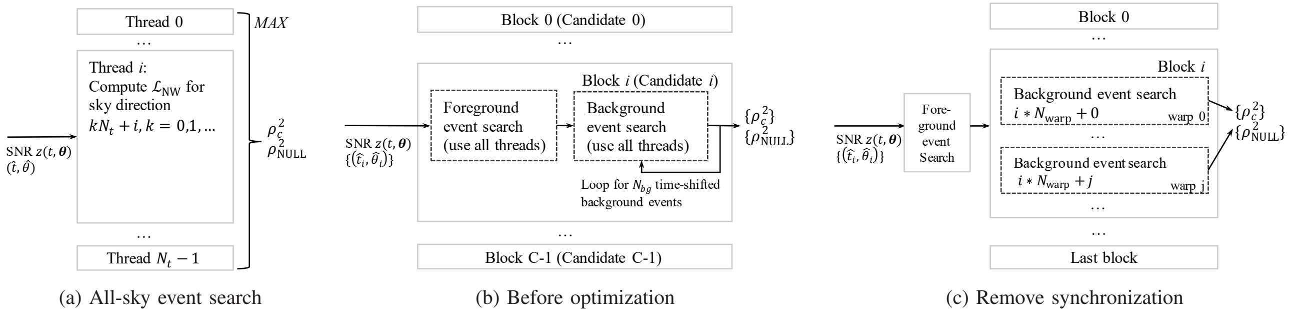

Thread 0
Thread i: Compute $\mathcal{L}_{NW}$ for sky direction $kN_t + i, k = 0,1,\dots$
Thread $N_t - 1$
MAX
SNR $z(t,\theta)$ $(\hat{t},\hat{\theta})$
ρ_c^2 ρ_{NULL}^2
(a) All-sky event search
Block 0 (Candidate 0)
Block i (Candidate i)
Foreground event search (use all threads)
Background event search (use all threads)
Loop for N_{bg} time-shifted background events
Block C-1 (Candidate C-1)
SNR $z(t,\theta)$ $\{(\hat{t}_i,\hat{\theta}_i)\}$
$\{\rho_c^2\}$ $\{\rho_{\mathrm{NULL}}^2\}$
(b) Before optimization
Block 0
Block i
Fore-ground event Search
Background event search $i * N_{\mathrm{warp}} + 0$ warp 0
Background event search $i * N_{\mathrm{warp}} + j$ warp j
Last block
(c) Remove synchronization

Fig. 3. The implementations of the coherent post-processing

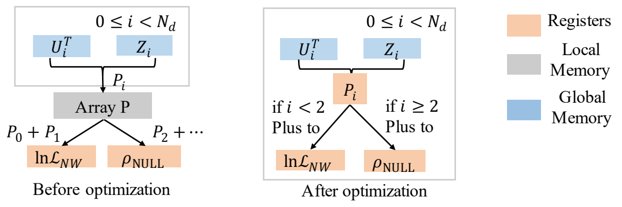


Fig. 4. Schematic of the local memory optimization

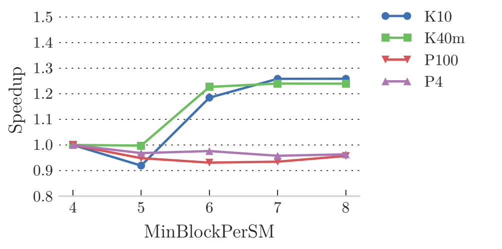


Fig. 5. Speedup ratio by limiting register usage

TABLE I
THEORETICAL OCCUPANCIES FOR DIFFERENT REGISTER USAGE

MinBlocksPerSM	RegistersPerThread	Occupancy
4	$\le$64	$\ge$50.0%
5	$\le$51	$\ge$62.5%
6	$\le$42	$\ge$75.0%
7	$\le$36	$\ge$87.5%
8	$\le$32	$\ge$100.0%

TABLE II
TIME USAGE OF THE SPIIR FILTERING

Method	Time (ms)			
	K10	K40m	P4	P100
Original	860.9	464.9	145.2	70.8
VMA	345.4	184.9	41.7	21.4
RODC	N/A	172.2	127.7	60.6
VMA+RODC	N/A	90.2	41.5	21.4

Since it is very hard to reduce register usage by modifying the algorithm, we consider limiting the register usage by force, which could be achieved by function qualifiers or compiler options. Here we use *__launch_bounds__* in CUDA to constraint the minimum blocks per multiprocessor (MinBlocksPerSM) to restrict register usage and increase theoretical occupancy. We can see from TABLE I how occupancy varies with the register usage.[3]

The disadvantage of limiting register usage by force is that more local memory are used caused by spilled registers. Although the theoretical occupancy is improved, more latencies might be brought in by local memory load and store instructions. The speedup ratios by limiting registers on different GPUs are illustrated in Fig. 5.

Different devices differ in performance. We can see that there could be 1.25 times speedup for some older Kepler devices such as K10 and K40m, while there is no benefit for the new Pascal devices such as P4 and P100.

IV. EXPERIMENTS

The running time of SPIIR filtering and post-processing before and after optimization is demonstrated and analyzed in this part. All the experiments are repeated for 50 times to get average execution time. All the testing results are measured in milliseconds. The detailed results are shown as follows.

A. Results of SPIIR filtering

The number of templates is set to 1024. For each template, there are 512 IIR filters. The average distance of time delays of adjacent IIR filters is set to $\Delta d = 20$.

There are four methods to be compared. The first one is the original implementation without read-only data cache, denoted as *Original*. Our method uses vectorized memory access to reduce the number of memory transactions, denoted as *VMA*. Original code with read-only data cache is denoted as *RODC*. The final experiment is combining read-only data cache and our method, denoted as *VMA+RODC*.

The final results are shown in TABLE II in milliseconds. Since K10 does not support read-only data cache, the corresponding results are not reported. From the results, we can see vectorized memory access improves the speed by a factor of 2.5-3.5x on both Kepler and Pascal GPUs. K40m could also benefit from combining two methods, getting a speedup ratio of over 5.1x.

[3]The number of threads per block is set to 256. For our tested GPUs, the numbers of registers per multiprocessor are all 64K.

TABLE III
TIME USAGE OF THE COHERENT POST-PROCESSING

Experiment	Time (ms)			
	K10	K40m	P4	P100
Original	1268.64	587.31	261.2	119.23
Remove synchronization	365.87	183.13	76.46	47.71
Arithmetic optimization	173.46	75.1	41.77	17.6
Local memory optimization	88.48	41.32	26.97	9.81
Coalesced memory access	63.11	33.73	16.14	9.35
Improve occupancy	49.68	27.25	16.14	9.35
Final speedup	25.54x	21.55x	16.18x	12.75x

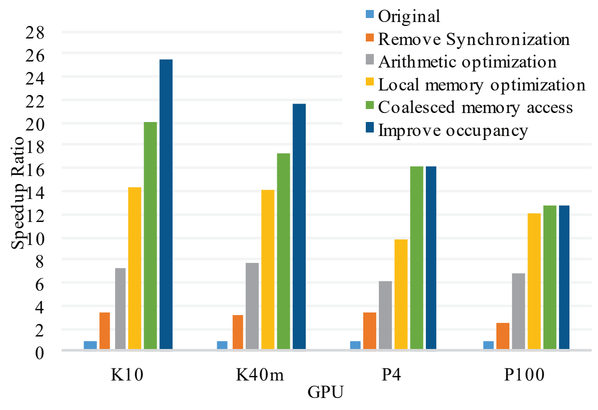


Fig. 6. Speedup ratio of the optimized coherent post-processing

B. Results of coherent post-processing

When testing the coherent post-processing part, the number of event candidates C is set to 1000. The number of time-shifted background events N_{bg} for each candidate is 100. The number of sky directions for brute force searching is 12288.

The optimization of coherent post-processing is divided into five steps: removing synchronizations, arithmetic optimization, decreasing local memory usage, coalesced memory access, improving occupancy. These optimization methods are performed step by step. All time usage results are measured in the same configurations in the above and shown in TABLE III. Better visualization of speedup ratios of each optimization step is shown in Fig. 6.

From Fig. 6, we can see that for first four steps of the optimization, both Kepler and Pascal GPUs benefit a lot and get a speedup of over 12x. Improving occupancy by limiting register usage improves running speed of Kepler GPUs but not for Pascal GPUs. The performance improvement by coalesced memory access for P100 is very small mainly due to the high memory bandwidth and the overhead of SNR matrix transposes.

V. CONCLUSION

In this paper, the SPIIR GW detection pipeline is optimized using recent GPU techniques, to decrease the latency and improve the throughput. The SPIIR filtering part of the pipeline is optimized by improving memory access efficiency with a speedup of more than 2.5x using the same GPU. For the coherent post-processing part, a speedup of 12-25x on the same GPU is achieved by employing multiple strategies to improve memory access and resolve synchronizations between threads. It is also worth noting that the recent Pascal generation GPU cards have significant improvement of performance over the preceding generations. In particular, Pascal P100 card provides an over 10x better performance over the Kepler K10 card on our applications.

ACKNOWLEDGMENT

This research is supported in part by Key Research and Development Program of China (No. 2016YFB1000602, 2017YFB0701501), the Key Laboratory of Space Astronomy and Technology, National Astronomical Observatories, Chinese Academy of Sciences, MOE research center for online education foundation (No. 2016ZD302), and National Natural Science Foundation of China (Nos. 61440057, 61272087 and 61363019).

REFERENCES

[1] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, V. B. Adhikari *et al.*, “Observation of gravitational waves from a binary black hole merger,” *Phys. Rev. Lett.*, vol. 116, p. 061102, Feb 2016.

[2] B. P. Abbott, R. Abbott, T. D. Abbott, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, V. B. Adya *et al.*, “Gw170817: Observation of gravitational waves from a binary neutron star inspiral,” *Phys. Rev. Lett.*, vol. 119, p. 161101, Oct 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.119.161101

[3] D. Buskulic, the LIGO Scientific Collaboration, and the Virgo Collaboration, “Very low latency search pipeline for low mass compact binary coalescences in the ligo s6 and virgo vsr2 data,” *Classical and Quantum Gravity*, vol. 27, no. 19, p. 194013, 2010.

[4] K. Cannon, R. Cariou, A. Chapman, M. Crispin-Ortuzar, N. Fotopoulos, M. Frei, C. Hanna, E. Kara, D. Keppel, L. Liao, S. Privitera, A. Searle, L. Singer, and A. Weinstein, “Toward early-warning detection of gravitational waves from compact binary coalescence,” *The Astrophysical Journal*, vol. 748, no. 2, p. 136, 2012.

[5] S. A. Usman, A. H. Nitz, I. W. Harry, C. M. Biwer, D. A. Brown, M. Cabero, C. D. Capano, T. D. Canton, T. Dent, S. Fairhurst, M. S. Kehl, D. Keppel, B. Krishnan, A. Lenon, A. Lundgren, A. B. Nielsen, L. P. Pekowsky, H. P. Pfeiffer, P. R. Saulson, M. West, and J. L. Willis, “The pycbc search for gravitational waves from compact binary coalescence,” *Classical and Quantum Gravity*, vol. 33, no. 21, p. 215004, 2016.

[6] S. Hooper, S. K. Chung, J. Luan, D. Blair, Y. Chen, and L. Wen, “Summed Parallel Infinite Impulse Response (SPIIR) Filters For Low-Latency Gravitational Wave Detection,” *Phys. Rev.*, vol. D86, p. 024012, 2012.

[7] Y. Liu, Z. Du, S. K. Chung, S. Hooper, D. Blair, and L. Wen, “Gpu-accelerated low-latency real-time searches for gravitational waves from compact binary coalescence,” *Classical and Quantum Gravity*, vol. 29, no. 23, p. 235018, 2012.

[8] X. Guo, Q. Chu, S. K. Chung, Z. Du, and L. Wen, “Acceleration of low-latency gravitational wave searches using maxwell-microarchitecture gpus,” *arXiv preprint arXiv:1702.02256*, 2017.

[9] A. Pai, S. Dhurandhar, and S. Bose, “Data-analysis strategy for detecting gravitational-wave signals from inspiraling compact binaries with a network of laser-interferometric detectors,” *Phys. Rev. D*, vol. 64, no. 4, p. 042004, Aug. 2001.

[10] D. M. Macleod, I. W. Harry, and S. Fairhurst, “A fully-coherent all-sky search for gravitational-waves from compact binary coalescences,” Phys. Rev. D 93, 064004 (2016), 2015. [Online]. Available: http://arxiv.org/abs/1509.03426

[11] Q. Chu, “Low-latency detection and localization of gravitational waves from compact binary coalescences,” Ph.D. dissertation, The University of Western Australia, Australia, 2017.