Porting Radio Astronomy Correlation to Setonix, a HPE Cray EX system powered by AMD GPUs

Cristian Di Pietrantonio
cristian.dipietrantonio@csiro.au
Pawsey Supercomputing Research Centre
Perth, Western Australia

Marcin Sokolowski
marcin.sokolowski@curtin.edu.au
Curtin Institute of Radio Astronomy
Perth, Western Australia

Christopher Harris
christopher.harris@csiro.au
Pawsey Supercomputing Research Centre
Perth, Western Australia

Daniel Price
daniel.price@skao.int
SKAO
Perth, Western Australia

Randal Wayth
randal.wayth@skao.int
SKAO
Perth, Western Australia

Abstract

In low-frequency radio astronomy, correlation of signals coming from hundreds of radio antennas is an early and fundamental step to create science-ready data products such as images of the sky at radio wavelengths. Because of the high volume of data to process and the rate at which they are produced, correlation is typically performed in real time by dedicated hardware, a FPGA or GPU cluster, installed near the telescope. However, there are science cases when an astronomer would like to correlate data later with customised settings such as time and frequency averaging of signals. Setonix, Pawsey Supercomputing Centre's HPE Cray Ex supercomputer based on AMD CPUs and GPUs, provides radio astronomers with enough computational power for such processing, but the only established GPU correlator works on NVIDIA GPUs and proved difficult to port. In this paper we discuss the process of providing Australian astronomers an implementation of the correlation algorithm that harnesses the computational power of Setonix.

ACM Reference Format:
Cristian Di Pietrantonio, Marcin Sokolowski, Christopher Harris, Daniel Price, and Randal Wayth. 2025. Porting Radio Astronomy Correlation to Setonix, a HPE Cray EX system powered by AMD GPUs. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Radio astronomy is a thriving science field that studies the Universe through observation at radio frequencies. Interferometers are complex instruments used to create images of the radio sky just like traditional telescopes produce images of the sky at optical wavelength. They are made up of multiple radio antennas distributed over a large area, each of which records high time resolution series of complex voltages induced by incoming radio signals. The Square Kilometre Array (SKA) [1], currently under construction in Western Australia and South Africa, will be the most sensitive radio telescope ever built and it is expected to produce more than 700 PB of data a year.

The first step to produce radio sky images is to cross correlate and time-average complex voltages from each antenna pair to obtain the cross-power spectrum. The computational complexity of correlation scales quadratically with the number of antennas making up the interferometer and it is usually implemented on hardware using FPGAs or through software running on GPUs. In both cases, dedicated compute infrastructure is typically deployed in physical proximity of the telescope computing the correlation products to be then sent to an archival facility for later science processing. In the case of the Murchison Widefield Array (MWA) [8, 10], low-frequency (70-300 MHz) interferometer and precursor to the SKA low-frequency component, a cluster equipped with 24 NVIDIA M2070 Fermi GPUs executes in real-time the xGPU [2] correlation engine with predefined time and frequency resolutions.

Some science cases such as pulsars [4] and Fast Radio Bursts (FRBs) [6] require data processing at finer resolutions and correlation must be run offline on archived raw voltages. In this situation a program called offline correlator, still relying on the xGPU library, is used. Being more than ten years old and highly optimised for the Fermi architecture, xGPU only supports NVIDIA GPUs and porting attempts to AMD GPUs by the authors failed, with the latest one crashing due to internal HIP errors in the deprecated texture reference API. The library makes heavy use of compile-time optimisations and requires parameters such as output time resolution and number of antennas to be known at compilation time. Performance is of uttermost importance in a real-time setting and led to such design choice. In offline processing, researchers exploring various scenarios desire more flexibility by making these configuration decisions at run time.

Pawsey's Setonix [3] is a HPE Cray EX system containing more than 200.000 AMD Milan CPU cores and 750 AMD MI250X GPUs. As radio astronomers prepared to transition to Setonix from an NVIDIA-based GPU system, they found themselves lacking a correlation software that could harness its computational power. This work introduces a new correlation library and command-line application, the Breakthrough Low-latency Imaging with Next-generation Kernels (BLINK) correlator, developed as part of the Pawsey Centre for Extreme Scale Readiness (PaCER) project [7]. The BLINK correlator provides the same functionalities as the traditional offline correlator with a simplified command line interface and API, and comparable performance. Parameters, such as time resolution, which play an important role in the scientific analysis outcomes can be set at run time and explored more easily. It can be executed on both NVIDIA and AMD GPUs, as well as on CPU cores if the user desires to do so, or if GPUs are not available. The BLINK correlator is being adopted by Australian MWA radio astronomers to leverage Pawsey's latest HPE Cray supercomputer to analyse vast amount of data, with the outlook of it being adopted to support some SKA's high time resolution science in the future.

2 Background

2.1 The correlation problem

A radio interferometer is an instrument composed of N_a antennas, each recording a time series $V_a(t)$ of complex voltages describing the radio signal observed over a predefined frequency bandwidth. The correlation process [5] combines the time series from each pair of antennas, called baseline, to form the cross power spectral density, which describes the distribution of power over the spatial frequencies of the radio signal. It provides information on the emission mechanisms of the signal and reveals higher level spatial features of the radio sky.

The first step is to transform each time series $V_a(t)$ into the frequency domain to obtain its spectra:

$$S_a(\nu) = \int_{-\infty}^{\infty} V_a(t) e^{-i2\pi\omega t} dt. \tag{1}$$

Then, spectra from each unordered pair $\{a, b\}$ of antennas are conjugated multiplied to compute signal correlation in the frequency domain:

$$C_{\{a,b\}} = S_a(\nu)\overline{S_b}(\nu). \tag{2}$$

The first step is accomplished in practice using a Fast Fourier Transform of finite length F on a discrete sample of $V_a(t)$, resulting in the original signal being decomposed in multiple discrete time series $S_a^\nu[t]$, one for each contiguous fine channel, with centre frequency ν, within the original frequency bandwidth.

The correlation step multiplies the fine channelised signals from each non-redundant pair of antennas. It also performs time averaging, over a time interval of t_{int} seconds, of the resulting cross-power spectrum to reduce noise and increase sensitivity to signal:

$$C_{\{a,b\}}^\nu = \left\langle S_a^v[t]\overline{S_b^v}[t] \right\rangle. \tag{3}$$

The correlation output is also commonly referred to as visibilities. In practice, because $C_{\{a,b\}}^\nu = \overline{C_{\{b,a\}}^\nu}$, only the lower triangular matrix needs to be computed. Furthermore, each antenna captures the orthogonal polarisations of the radio signal, commonly referred to X and Y, which are recorded in two independent streams of voltage data. Hence, for each baseline, the algorithm computes a 2×2 correlation matrix of the polarisations of the antennas.

2.2 The Murchison Widefield Array

The Murchison Widefield Array (MWA) is a low-frequency radio telescope located in Western Australia, 800 km north of Perth. It observes frequencies in the 70-300 MHz range, and is able to instantly capture a frequency bandwidth of 30.72 MHz at any given time. The MWA is composed of 144 tiles, of 16 dipole antennas, able to record two orthogonal polarisations of the radio wave as separate signals. The first incarnation of the instrument, referred to as Phase I, saw 128 tiles installed in 2013, along with power, computational, and communication infrastructure [8]. In 2018, the Phase II expansion added 16 more tiles (the original design envisioned 128 more [10]). For several years the receivers stayed the same, as it did the correlation software, an so there could only be 128 tiles operating at any given time. While this is not the case any more, we will focus on this representative configuration for clarity.

Each tile acts as a logical antenna combining the signal from its 16 antennas and sending it to the associated receiver. Signal from all receivers is digitally-sampled at 8 bits and filtered into 24, possibly non-contiguous, coarse channels of 1.28 MHz bandwidth. Each of these data streams is further filtered into 128, 10 kHz fine channels. At this point, data is sent to the online correlator to be further processed and to produce science-ready datasets with the default operations pipeline at the predefined 0.5 s time and 40 KHz frequency resolutions. A system component named Voltage Capture System (VCS) can also save channelised voltages to enable more flexible processing at the highest time and frequency resolutions [9]. Its data rate, r_{vcs}, is given by

$$r_{vcs} = N_c N_a N_p N_s N_b, \tag{4}$$

where N_c is the total number of fine channels (= 3092), N_a is the number of tiles (= 128), N_p is the number of polarisations (= 2), N_s is the sampling frequency of the system (= 10^4 Hz), and N_b is the number of bytes needed to represent one data item. The MWA VCS data rate r_{vcs} is then 7.37 GiB/s.

2.3 The xGPU correlation library

The online and offline correlators for the MWA take advantage of the GPU correlation engine implemented in the xGPU library[1]. It is a CUDA code targeting the NVIDIA Fermi GPU architecture, the latest at the time of initial development, where it achieves up to 79% of peak FLOPS. This is achieved by adopting several optimisation strategies. The correlation problem is decomposed in independent subdomains such that memory accesses align with the Fermi memory hierarchy, down to the register level, maximising the efficient use of memory bandwidth. Arithmetic intensity of the code is increased by programming shared memory to cache and reuse data. Instruction level parallelism is enabled by the extensive use of loop unrolling and preprocessor macros. Problem domain parameters like number of antennas and time samples are specified at compile time to allow further optimisations of this kind. Finally, texture memory is employed to take advantage of its linear interpolation unit and to reduce the need of explicit indexing computations in the kernel. The xGPU library is compiled for a target telescope, time and frequency resolutions by specifying the `NSTATION`, `NTIME`, and

[1]https://github.com/GPU-correlators/xGPU/tree/master

NFREQUENCY Makefile parameters. Then, it is linked to the main software that implements the full correlation software pipeline.

3 Porting attempt of xGPU to Setonix

As part of the procurement process, the Pawsey Supercomputing Research Centre funded the PaCER program to help porting scientific codes and workflows to Setonix, its latest supercomputer based on AMD GPUs and CPUs. The xGPU library was identified as one of the NVIDIA GPU software at heart of several radio astronomy workflows to process high time resolution observations captured with the MWA VCS. The porting activity was captured under the PaCER BLINK project that looked at developing a GPU software pipeline to detect Fast Radio Bursts within MWA archival data. The first porting attempt took place in 2022 using earlier versions, 4.7 and 5.0, of ROCm on MI50 and MI100 GPU prototypes provided by AMD. Another more recent attempt used ROCm 6.3.2 on Setonix.

The xGPU code must be rewritten using the AMD HIP programming language for it to be compiled for AMD GPUs. The `hipify-perl` script provided with HIP simplifies the task by replacing every occurrence of a CUDA API call with its HIP equivalent:

```
hipify-perl cuda_xengine.cu > cuda_xengine.cpp
```

Because the API function signatures differ only in their prefixes, the operation is as simple as replacing the `cuda` prefix with `hip`. The file extension has been changed to `.cpp` so that the HIP compiler recognises it contains HIP code. Any other extension, such as `.c`, will not enable the HIP language extensions unless the `-x hip` option is passed to `hipcc`.

While all the high-level API calls are translated correctly, xGPU also makes use of inline PTX assembly that is not translated by the `hipify-perl` script. The following line,

```
asm("dp4a.s32.s32 %0, %1, %2, %3;" : "+r"(c) : "r"(a),
                        "r"(b), "r"(c));
```

implements the dot product of four 8-bit elements stored contiguously on a 32-bit integer, as a single instruction. At the time of the first porting attempt it was unclear whether the MI50 and MI100 supported such instruction and the software-level implementation, also present in xGPU, was used as replacement. Today we know that the AMD CDNA2 ISA contains the `V_DOT4_U32_U8` instruction, equivalent to `dp4a`, based on the CDNA2 Instruction Set Architecture (ISA) specification document.

Other minor compilation errors arose due to source code being processed by a newer clang-based compiler than the one originally used for development. Once these were addressed, we generated a shared library and linked it against the offline correlator program for testing. In the first 2022 attempt, the execution of the program ended prematurely with an insufficient texture memory error.

The same error cannot be reproduced today on Setonix's MI250X GPUs. Instead, the program crashes with the following error:

```
  :0:/[...]/src/clr/hipamd/src/hip_global.cpp:78 : 1506433737274d us:
Cannot create GlobalVar Obj for symbol: _ZL11tex1dfloat2
```

The software was recompiled in debug mode so we could run `rocgdb` to investigate where the error occurs. The stack trace shown in Figure 1 points to an internal HIP error within the `hipBindTexture` function.

The xGPU library uses texture references and associated APIs `hipBindTexture` is part of. Texture references must be declared and instantiated at global scope. The message presented by the program points to an error in doing so. Importantly, the texture reference API has been marked as deprecated both in CUDA and HIP, with the former dropping support since CUDA 12. Texture objects were introduced as replacement but a refactoring of the xGPU code to make use of them has been deemed counter productive. Instead, we opted to develop a more flexible and maintainable implementation that fits the requirements for offline correlation.

4 The BLINK correlator implementation

The BLINK project[2] is developing a GPU-accelerated software pipeline to process petabytes of archival MWA data on Setonix. Correlation is one of the fundamental steps during which voltage time series from every two antennas are combined to form visibilities by implementing Equation 3. Fine channelisation of MWA VCS data is handled on a dedicated FPGA system installed near the telescope, leaving the Conjugate Multiply Accumulate (CMAC) operation to be developed.

The correlation software is designed as a C++ library of functions to be invoked in other programs. A command-line interface, built on top of the library, is also provided to replace the legacy offline correlator used in some of the MWA software pipelines. Many of the science domain parameters, notably the integration time, are now determined at runtime from metadata information or user input. Then, the same installation can be used for multiple science cases with a varying time and frequency resolution requirements. The library supports execution on CPU with parallelisation over multiple cores handled with OpenMP. The GPU implementation is based on a system of macros resolving to CUDA or HIP API calls and syntax based on whether the code is compiled for NVIDIA or AMD GPUs, respectively.

4.1 Data ingesting

The input complex voltage data is organised in a multidimensional array whose dimensions are, from slowest to fastest moving:

$$\text{time step} \times \text{channel} \times \text{antenna} \times \text{polarisation}.$$

Because the BLINK correlator also performs time averaging on the correlation products, the code accesses voltage samples consecutively in time from the same channel, antenna, and polarisation. A corner turn on the multidimensional data layout is required for the memory access pattern to make efficient use of the memory cache. In particular, consecutive time samples from the same integration interval are placed closest in memory, resulting in the following layout, from slowest to fastest moving:

$$\text{integration time interval} \times \text{channel} \times \text{antenna} \times \text{polarisation} \times \text{time sample}.$$

The reordering is performed when data is first read from disk into memory. In addition, each 8 bit complex sample is expanded to 2 bytes for native instructions support. Voltages can be made accessible to the GPU correlation kernel in various ways. One is to read data from disk to host pinned memory, then transferred

[2] https://github.com/PaCER-BLINK-Project

```
Thread 1 "cuda_correlator" received signal SIGABRT, Aborted.
0x00001555539c4d2b in raise () from /lib64/libc.so.6
(gdb) bt
#0 0x00001555539c4d2b in raise () from /lib64/libc.so.6
...
#5 0x00001555541a5786 in ?? () from /software/setonix/rocm/6.3.2/lib/libamdhip64.so.6
#6 0x00001555554ee835 in hipBindTexture<HIP_vector_type<char, 2u>, 1, (hipTextureReadMode)1>
(offset=0x0, tex=..., devPtr=0x155548600000, desc=..., size=1310720)
at /software/setonix/rocm/6.3.2/lib/llvm/bin/../../../include/hip/hip_runtime_api.h:9078
#7 0x00001555554ee1eb in xgpuCudaXengine (context=0x7fffffff4ae8, syncOp=1) at cuda_xengine.cpp:615
#8 0x00000000002026b8 in main (argc=1, argv=0x7fffffff4d28) at cuda_correlator.cpp:203
```

Figure 1: Stack trace of the correlation program at the moment of the crash caused by a runtime error in using texture memory.

to a memory buffer allocated on the GPU. Another is to use Managed Memory, where the allocation residing on the host is also mapped to the memory address space of a GPU. In the latter case, GPU memory acts as a cache for the host memory, which is larger on Setonix, and data transfers are handled by the GPU hardware. Finally, with a Linux kernel supporting the Heterogeneous Memory Management, the MI250X's CDNA2 architecture allows the host to write to memory allocated on the accelerator with a similar page migration mechanism to the one of Unified Memory. The last option is not portable to other systems and GPUs, hence it was discarded during development. We opted for explicit, asynchronous memory transfers overlapped with kernel execution. This is the same strategy adopted in xGPU.

4.2 CPU implementation

The starting point to develop a new correlator is to reproduce the output of xGPU with a C++ function whose implementation is presented in Algorithm 1. The input is a multidimensional array as described in Section 4.1. The output are correlation matrices, one for each pair of output frequency channel and integration time interval. A correlation matrix is stored as an 1D array where the elements at position i, $i+1$, $i+2$, and $i+3$, $i \in \{4j : j \in \{0,\dots,B-1\}\}$, correspond to the polarization pairs XX, XY, YX, and YY of the i-th baseline, and B is the number of baselines.

The outer for loops in the implementation iterate over the output dimensions to select the correlation matrix and the baseline, whose visibilities are to be computed next. The innermost block of code multiplies and accumulates complex samples from the selected baseline within the same integration time interval. Additionally, there is an option to accumulate values from a predefined number of contiguous frequency channels, which helps reduce the output data rate. Then, the visibility value is averaged by dividing it by the integration time and the number of accumulated frequency channels. Finally, the resulting visibility is stored in the corresponding output matrix.

The execution of Algorithm 1 is sped up over multiple CPU cores by processing each integration interval and baseline, which are independent of one another, in parallel. While the number of integration intervals may vary substantially, parallelisation is always retained because the number of MWA baselines is at least 8256. The C++ implementation adopts the OpenMP standard to speed up the code execution with just one line of code.

```
Data: Fine channelised voltage time series
Result: Correlation lower triangular matrices
$output\_matrix_{ic}$ ← lower triangular matrix for integration
  interval $i$ and output channel $c$;
for integration interval i do in parallel
    for baseline b do in parallel
        foreach output channel c do
            foreach polarisation pair p do
                $accum \leftarrow 0$;
                foreach channel to average do
                    foreach integration step do
                        $s_a, s_b$ ← samples from current
                          baseline, channel, and integration
                          step;
                        $CMAC(accum, s_a, s_b)$;
                    end
                end
                Divide accum by integration time and
                  averaged channels;
                $output\_matrix_{ic}[b][p] \leftarrow accum$;
            end
        end
    end
end
```

Algorithm 1: Cross correlation algorithm on CPU.

4.3 GPU implementation

The GPU implementation of the BLINK correlator derives directly from the CPU version and it is illustrated at a high level in Algorithm 2. The problem decomposition strategy adopted to parallelise the computation reflects the computational model GPUs are programmed with. Firstly, the same correlation kernel is executed concurrently on all integration time intervals, with each execution assigned to a CUDA/HIP stream. This allows computation and data transfer activities to overlap and reduce the overall execution time. Within the GPU kernel, the output frequency channel and baseline dimensions are flattened into one. That is, the implementation processes baselines across all output frequency channels at the same time. It does so by evenly distributing them, with their associated pairs of voltage time series, across all thread warps in the grid of thread blocks. Warps are scheduled for execution over all Streaming Multiprocessors (SMs) of the GPU.

Threads within each warp correlate the two time series of the associated baseline in parallel. First, each thread independently computes the CMAC on disjoint subsets of the time series. The partial results are then combined together by means of a parallel reduction operation spanning all threads in the warp. Finally, the result is divided by the integration time and number of averaged frequency channels.

```
Data: Fine channelised voltage time series
Result: Correlation lower triangular matrices
$output\_matrix_{ic}$ ← lower triangular matrix for integration interval $i$ and output channel $c$;
for baseline $b$ in all output channels and integration intervals do in parallel over warps in grid blocks
    foreach polarisation pair $p$ do
        $accum$ ← 0;
        foreach channel to average do
            for integration step do in parallel over threads in warp
                $s_a, s_b$ ← samples from current baseline, channel, and integration step;
                $CMAC(accum, s_a, s_b)$;
            end
        end
        $WarpReduction(accum)$;
        Divide $accum$ by integration time and averaged channels;
        $output\_matrix_{ic}[b][p]$ ← $accum$;
    end
end
```

Algorithm 2: Cross correlation algorithm on GPU.

5 Benchmarking

The test case adopted for the benchmarking task is the correlation of one coarse channel and one second of a MWA observation at 50 ms time and 40 kHz frequency resolutions. It is representative of actual MWA VCS processing workloads during which many of these files, making up entire observations, are processed in parallel. We first established the correctness of our CPU and GPU implementations by comparing the produced output matrices with the ones generated by xGPU and observing they are identical. Figure 2 presents a sample correlation matrix computed from antenna signals that are integrated for 50 ms and span a frequency band of width 40 kHz, centered at 139 MHz.

The performance of the BLINK GPU correlator against the one of xGPU is evaluated in terms of execution times of both the computational kernel and the overall program, including I/O operations. The execution environment for xGPU is a GPU node of Garrawarla, a now decommissioned Pawsey cluster, featuring an NVIDIA V100 GPU. The BLINK GPU correlator is run on a Setonix GPU node using a single MI250X GCD, whereas the CPU version is run on a CPU node equipped with 2 AMD Milan CPUs for a total of 128 cores. We did not benchmark the BLINK correlator on Garrawarla before its decommissioning, hence the lack of comparison on the same GPU hardware. However, because radio astronomers transitioned from Garrawarla to Setonix, the comparison presented in this work is more relevant.

The execution times are reported in Table 1. The BLINK GPU kernel is 10 times slower than xGPU. However, the total execution time is 6 seconds for both GPU applications. In other words, computation only represents a small percentage of the overall execution, 0.5% and 5% for xGPU and BLINK GPU respectively. The majority of time is spent reading the 313 MB input file and writing the correlation matrices to the `/scratch` filesystem, for a total of 162 MB of data. The CPU implementation of the correlation routine is 13 times slower than the GPU counterpart. The factor goes down to 1.6 when the total execution time, which includes I/O operations, is considered.

Setonix nodes are equipped with hardware counters that keep track of power and energy consumption. In addition to providing figures related to the entire node, the counters allow to monitor energy consumption of the single hardware components such as CPU and GPU cards. Figure 3a shows the power consumption of the CPU correlation job over time. The first 4 seconds are spent reading in the input data and so the CPU activity is minimal. The power consumption surges once the multi-threaded computation starts and drops down to the previous level when it ends.

Figure 3b depicts the power consumption of a GPU node running the BLINK GPU correlation software. Measuring the energy consumption of a process using a single MI250X GCD in a shared GPU node setting results in inaccurate figures. Hence, an entire GPU node is allocated to 8 correlation jobs executed in parallel, one for each GCD. The total figures are then scaled down to obtain the cost of a single BLINK GPU correlation task. In this case multiple input files are retrieved from the filesystem, leading to a longer read times. After 6 seconds all the GPUs are executing the correlation kernel, resulting in a steep surge in the power consumption, above 2 kW. The narrow peak reflects the sub-second duration of the computation, followed by a decrease in power usage by all GPUs and the CPU after the output is written to the filesystem.

Table 2 reports the energy spent executing the jobs, in joules, as well as the amount of Service Units (SU) charged for the use of resources. A GPU node completely allocated for 7.5 seconds to correlate 8 coarse channel data files, each representing an observing time of 1 second, consumes 4.7 kJ and it is billed 1.06 SU. These values are divided by eight to obtain the cost of correlating a single coarse channel. The third row in the table indicates that it takes 4.6 times more energy to do the same on a CPU node, at 2.3 times the SU cost.

6 Discussion and conclusions

A growing interest in high time resolution radio astronomy science, such as pulsars and FRBs, has let to the xGPU library being adopted for offline processing of archival, high resolution voltage data. With the transition of MWA researchers to Setonix, a HPE Cray EX supercomputing system based on AMD GPUs, porting xGPU to the new hardware vendor ecosystem was identified as a critical task during the migration process. While code translation from CUDA to HIP had been relatively easy in other scenarios, xGPU represented an exception where developing a completely new and simpler code proved to be a more efficient and timely response to the computing needs of the radio astronomy community.

Software	Total (s)	Kernel total (s)	Kernel total (%)
xGPU	6.0	0.032	0.53
BLINK GPU	6.0	0.336	5.6
BLINK CPU	10.0	4.4	44

Table 1: Execution times of xGPU, and BLINK CPU and GPU correlators on the proposed test case. The first column reports the execution time of the entire program, including I/O and memory allocations. The second column is the time spent in the correlation routine, either the GPU kernel instances or the CPU function. The third column reports the same information as a percentage of the total time.

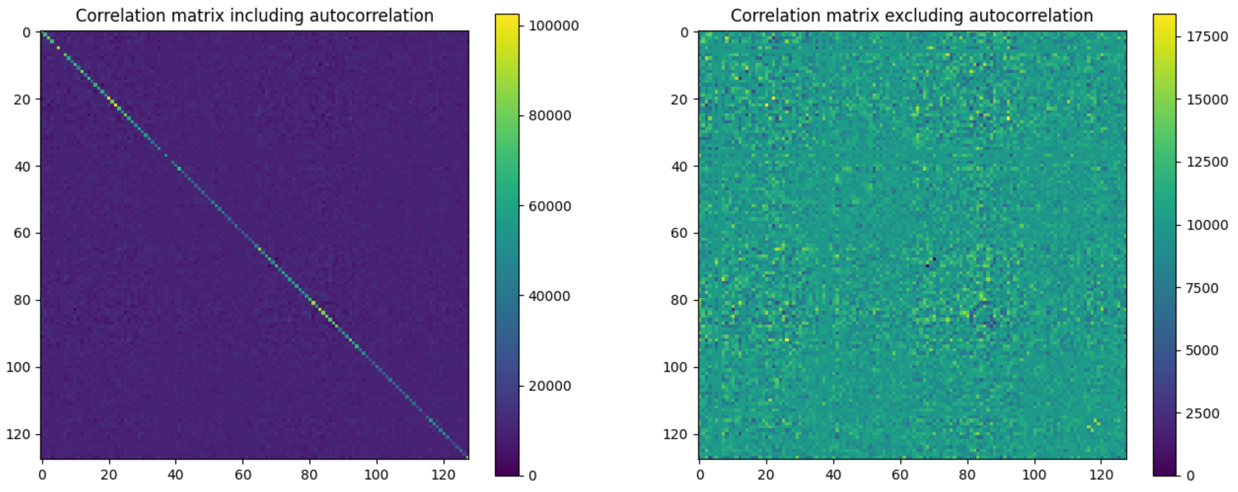


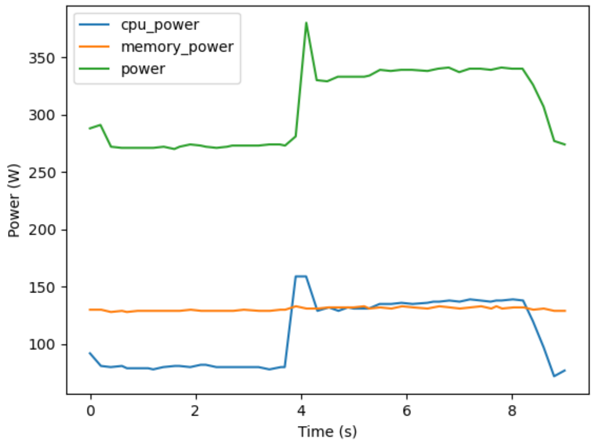
Figure 2: The output of the correlation operation is a correlation (or visibility) matrix. The left panel shows the correlation matrix including autocorrelation products, that is, the correlation of the signal from an antenna with itself. Often these are excluded, resulting in the correlation matrix on the right panel. In this example we only plotted the XX polarisation pair.

Software	Consumed Energy (J)	Billing (SU)
BLINK GPU (full node)	4755	1.06
BLINK GPU (1 task)	594.3	0.13
BLINK CPU	2756	0.3

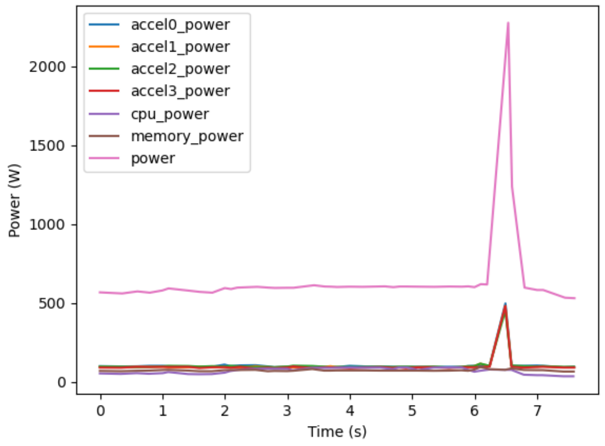
Table 2: Consumed energy and service unit cost of running CPU and GPU correlation jobs on Setonix to process one second, one coarse channel of MWA VCS data. A precise measurement of energy consumption by a single BLINK GPU correlation job can not be obtained when the GPU node is shared with other users. Hence, 8 GPU correlation jobs are submitted to occupy an entire GPU node, and the measured energy consumption and billing information are divided by 8 to get the cost per single correlation task.

The porting attempt started using MI50 and MI100 cards prototypes when both the AMD hardware and software were in early development stages, subject to an unstable and rapidly changing programming environment. Furthermore, the extensive use of optimisation techniques in xGPU such as loop unrolling, inline assembly, and macro expansions, makes any non-trivial change to the code time consuming and prone to programming errors. While we managed to compile a xGPU port based on HIP, its execution terminated with errors due insufficient texture memory on MI50 and MI100 cards, and internal errors in the now deprecated texture reference API when run on Setonix. Further work on xGPU could have led to extensive software changes, moving away from what initially had been identified as code porting activity.

We developed a flexible, architecture-agnostic correlation library designed to promptly meet the radio astronomy community computational needs, effectively removing one of the biggest blockers of the migration process to Setonix. The development focus is shifted to providing a working solution with acceptable performance in a timely manner. While the computational kernel lacks of sophisticated optimisations that would allow it to match xGPU performance, I/O is by far the dominant component of the total execution time.



(a) BLINK CPU power consumption



(b) BLINK GPU power consumption

Figure 3: Reported power consumption of Setonix nodes as a function of time during the execution of the BLINK CPU and GPU correlators. Panel (a) illustrates the power consumption of a CPU node during correlation implemented using OpenMP. Panel (b) shows the energy consumption of a GPU node packed with 8 correlation tasks, one for each GCD.

Further investment into optimisation would present diminishing returns. With offline correlation being the main use case, usability gains higher priority by allowing important domain parameters, such as integration time and channel averaging factor, to be specified at run time. The addition of parallel CPU implementation enables researchers to choose between the CPU and GPU partition of Setonix depending on their availability and other software the BLINK correlator is used in conjunction with.

7 Acknowledgements

This scientific work uses data obtained from Inyarrimanha Ilgari Bundara / the Murchison Radio-astronomy Observatory. We acknowledge the Wajarri Yamaji People as the Traditional Owners and native title holders of the Observatory site. Establishment of CSIRO's Murchison Radio-astronomy Observatory is an initiative of the Australian Government, with support from the Government of Western Australia and the Science and Industry Endowment Fund. Support for the operation of the MWA is provided by the Australian Government (NCRIS), under a contract to Curtin University administered by Astronomy Australia Limited.

This work was supported by resources provided by the Pawsey Supercomputing Research Centre's Setonix Supercomputer (https://doi.org/10.48569/18sb-8s43), with funding from the Australian Government and the Government of Western Australia. Authors also acknowledge the Pawsey Centre for Extreme Scale Readiness (PaCER) for funding and support.

References

[1] C.L. Carilli and S. Rawlings. "Science with the Square Kilometer Array: Motivation, Key ScienceProjects, Standards and Assumptions". In: *New Astronomy Reviews* 48.11-12 (Dec. 2004), pp. 979–984. DOI: 10.1016/j.newar.2004.09.001. URL: https://doi.org/10.1016%2Fj.newar.2004.09.001.

[2] M. A. Clark, P. C. La Plante, and L. J. Greenhill. "Accelerating Radio Astronomy Cross-Correlation with Graphics Processing Units". In: (July 2011). DOI: 10.48550/ARXIV.1107.4264. arXiv: 1107.4264 `[astro-ph.IM]`.

[3] Hewlett-Packard Enterprise. *Setonix*. 2022. DOI: 10.48569/18SB-8S43.

[4] D Lorimer and M Kramer. "Handbook of Pulsar Astronomy". In: *Cambridge observing handbooks for research astronomers*. Vol. 4. Cambridge, UK: Cambridge University Press, 2004.

[5] S. M. Ord et al. "The Murchison Widefield Array Correlator". In: *Publications of the Astronomical Society of Australia* 32 (2015), e006. DOI: 10.1017/pasa.2015.5.

[6] E. Petroff, J.W.T. Hessels, and D.R. Lorimer. "Fast radio bursts". In: *The Astronomy and Astrophysics Review* (2019). DOI: 10.1007/s00159-019-0116-6.

[7] Marcin Sokolowski et al. "High-time resolution GPU imager for FRB searches at low radio frequencies". In: *Publications of the Astronomical Society of Australia* 41 (2024), e042. DOI: 10.1017/pasa.2024.46.

[8] S. J. Tingay et al. "The Murchison Widefield Array: The Square Kilometre Array Precursor at Low Radio Frequencies". In: *Publications of the Astronomical Society of Australia* 30 (2013), e007. DOI: 10.1017/pasa.2012.007.

[9] S. E. Tremblay et al. "The High Time and Frequency Resolution Capabilities of the Murchison Widefield Array". In: *Publications of the Astronomical Society of Australia* 32 (2015). ISSN: 1448-6083. DOI: 10.1017/pasa.2015.6.

[10] Randall B. Wayth et al. "The Phase II Murchison Widefield Array: Design overview". In: *Publications of the Astronomical*

Society of Australia 35 (2018). ISSN: 1448-6083. DOI: 10.1017/pasa.2018.37.