Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

pawsey

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

Cristian Di Pietrantonio

Overview of the Murchison Widefield Array (MWA)

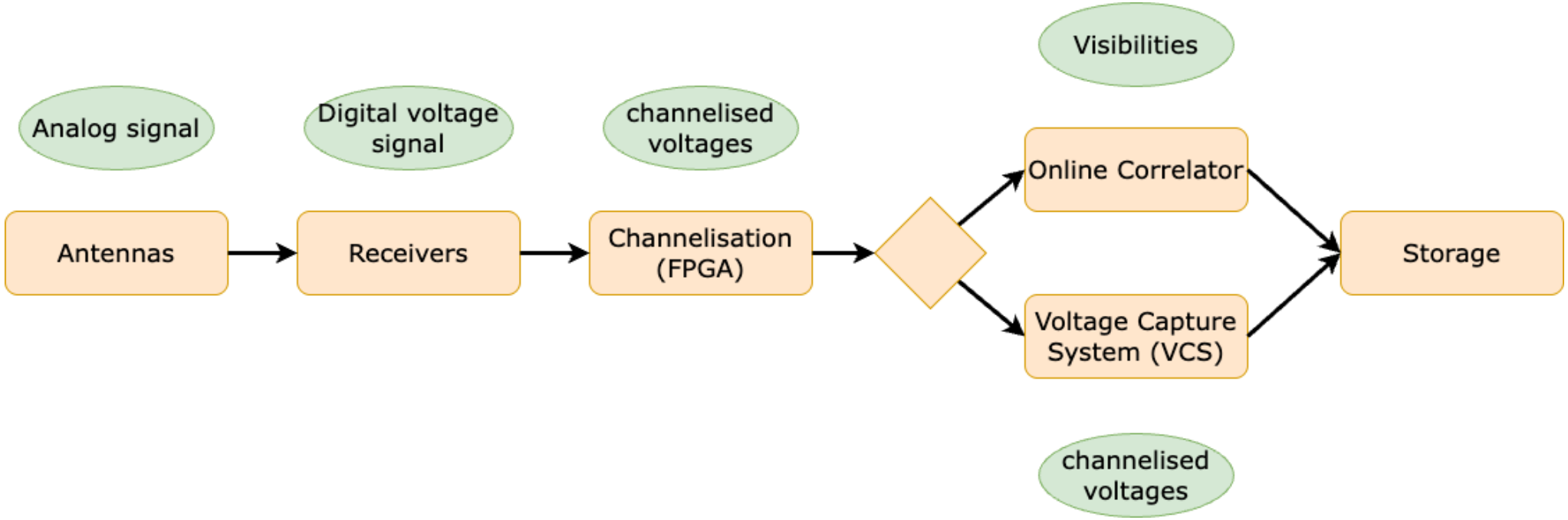


Figure 1. High level overview of the Murchison Widefield Array (MWA) modes of operation.

- The online correlator produces science-ready data for the most common use cases for astronomers to use.
  - Reduces data volume and complexity but limits the “resolution” of the data.
- The Voltage Capture System allows researchers to access channelized voltages to run custom processing.
  - Gives access to data at highest resolution, but volume of data is larger.

The Correlation Problem

Correlation combines the time series from each pair of antennas to form the cross power spectral density, which describes the distribution of power over the spatial frequencies of the radio signal. In practice, for each pair of antennas,

1. Cross correlation of digitalized voltages, then FFT to get the spectral density, or
2. FFT on digitalized voltages, then complex conjugate multiplication in the frequency space. This option is computationally more efficient.

The FFT is implemented using FPGA and the multiplication of the signals is done on GPU.

- Time averaging allows to reduce noise and to increase sensitivity.
- Channel averaging is used to reduce the data rate but has an impact on sensitivity.

Only a small set of time averaging and channel averaging combinations are available during online correlation.

Offline correlation allows to study very short transient signals by reducing integration time and avoid signal dilution.

The Correlation Problem

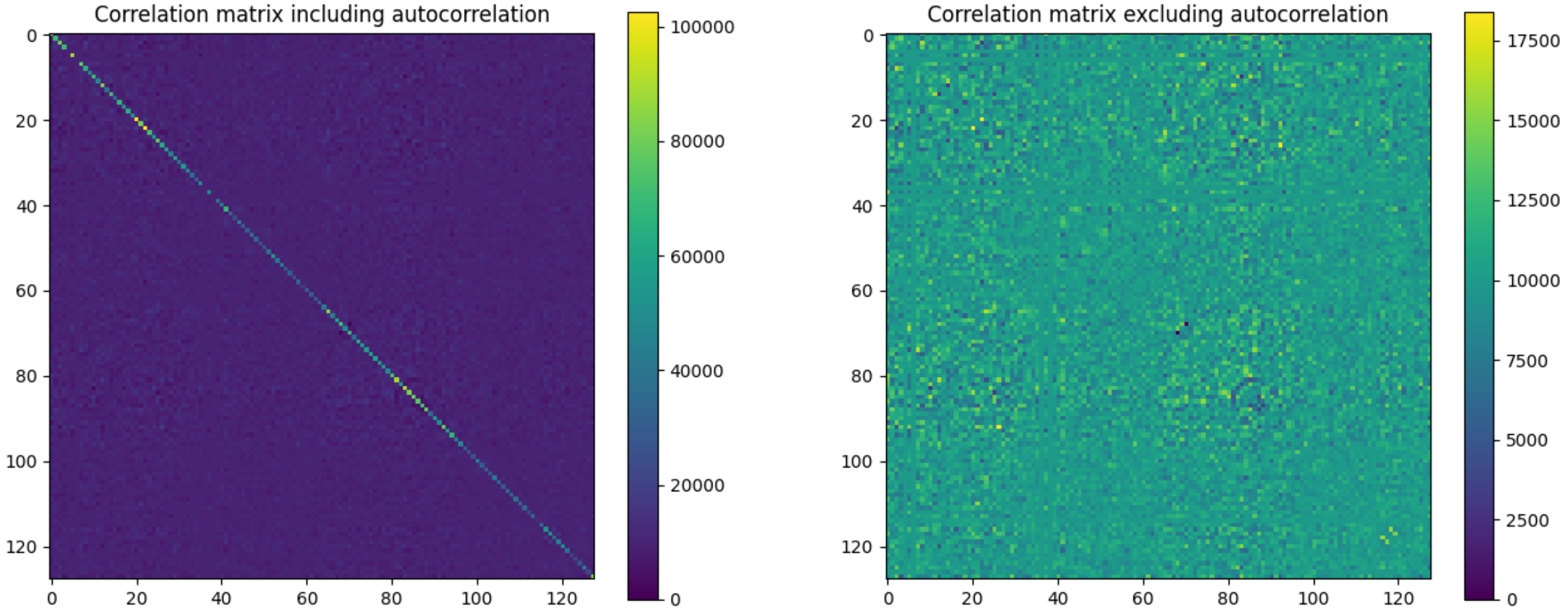


Figure 2. Correlation products (that is, visibilities) for a given frequency and integration interval.

The xGPU library

xGPU is a very efficient NVIDIA GPU implementation of correlation used by the online correlation system of the MWA.

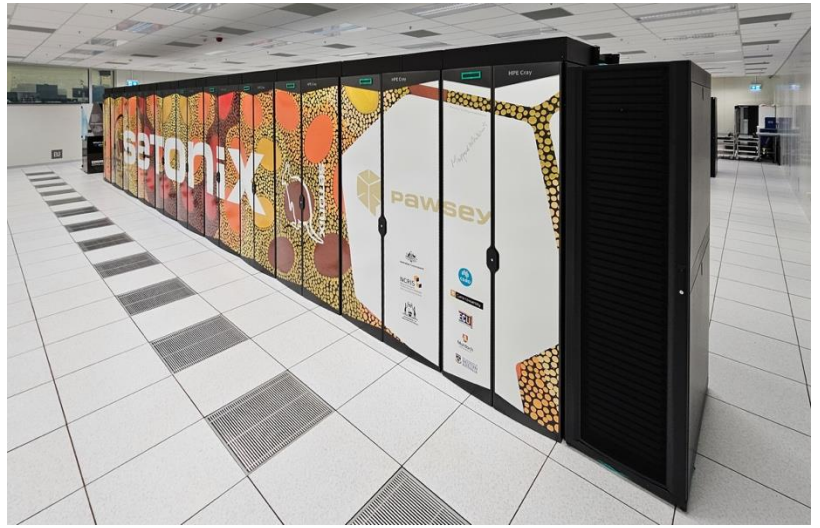
- Developed in 2011, targets the NVIDA Fermi architecture.
- Problem decomposition to align with the memory hierarchy and to maximise use of memory bandwidth
- Arithmetic intensity increased using shared memory
- Instruction level parallelism via loop unrolling and preprocessor macros
- Domain parameters specified at compile time
- Texture memory is employed to speed up indexing calculations
- Texture References are not supported starting from CUDA 12
- No unit tests to verify correctness of the program

```
#pragma unroll 2
#else
#pragma unroll 1
#endif
  for(unsigned int t=0; t<NTIN

    __syncthreads();

#if BUFFER_DEPTH==2
    TWO_BY_TWO_COMPUTE(0);
    LOAD(1, t+1);
#elif BUFFER_DEPTH==4
    TWO_BY_TWO_COMPUTE(0);
    TWO_BY_TWO_COMPUTE(1);
    LOAD(2, t+2);
    LOAD(3, t+3);
#endif

    __syncthreads();


#if BUFFER_DEPTH==2
    TWO_BY_TWO_COMPUTE(1);
    LOAD(0, t+2);
#elif BUFFER_DEPTH==4
    TWO_BY_TWO_COMPUTE(2);
    TWO_BY_TWO_COMPUTE(3);
    LOAD(0, t+4);
    LOAD(1, t+5);
```

```
#if COMPLEX_BLOCK_SIZE == 1
#define TWO_BY_TWO_PRELOAD(s)                                          \
 {float col1Xreal = input[s][4*tx                                   ];  \
  float col1Ximag = input[s][4*tx     + 4*TILE_WIDTH                ];  \
  float col1Yreal = input[s][4*tx + 1                               ];  \
  float col1Yimag = input[s][4*tx + 1 + 4*TILE_WIDTH                ];  \
  float col2Xreal = input[s][4*tx + 2                               ];  \
  float col2Ximag = input[s][4*tx + 2 + 4*TILE_WIDTH                ];  \
  float col2Yreal = input[s][4*tx + 3                               ];  \
  float col2Yimag = input[s][4*tx + 3 + 4*TILE_WIDTH                ];  \
  float row1Xreal = input[s][4*ty                          + 8*TILE_WIDTH];  \
  float row1Ximag = input[s][4*ty     + 4*TILE_HEIGHT + 8*TILE_WIDTH];  \
  float row1Yreal = input[s][4*ty + 1                      + 8*TILE_WIDTH];  \
  float row1Yimag = input[s][4*ty + 1 + 4*TILE_HEIGHT + 8*TILE_WIDTH];  \
  float
  float
  float
  float
```

```
#define TWO_BY_TWO_PRELOAD(s)                                          \
 {float col1Xreal = input[s][2*tx                                   ];  \
  float col1Ximag = input[s][2*tx     + 2*TILE_WIDTH                ];  \
  float col1Yreal = input[s][2*tx     + 4*TILE_WIDTH                ];  \
  float col1Yimag = input[s][2*tx     + 6*TILE_WIDTH                ];  \
  float col2Xreal = input[s][2*tx + 1                               ];  \
  float col2Yreal = input[s][2*tx + 1 + 4*TILE_WIDTH                ];  \
  float col2Ximag = input[s][2*tx + 1 + 2*TILE_WIDTH                ];  \
  float col2Yimag = input[s][2*tx + 1 + 6*TILE_WIDTH                ];  \
  float row1Xreal = input[s][2*ty                          + 8*TILE_WIDTH];  \
  float row1Ximag = input[s][2*ty     + 2*TILE_WIDTH + 8*TILE_WIDTH];  \
  float row1Yreal = input[s][2*ty     + 4*TILE_WIDTH + 8*TILE_WIDTH];  \
  float row1Yimag = input[s][2*ty     + 6*TILE_WIDTH + 8*TILE_WIDTH];  \
  float row2Xreal = input[s][2*ty + 1                      + 8*TILE_WIDTH];  \
  float row2Yreal = input[s][2*ty + 1 + 4*TILE_WIDTH + 8*TILE_WIDTH];  \
  float row2Ximag = input[s][2*ty + 1 + 2*TILE_WIDTH + 8*TILE_WIDTH];  \
  float row2Yimag = input[s][2*ty + 1 + 6*TILE_WIDTH + 8*TILE_WIDTH];
#else
#error COMPLEX_BLOCK_SIZE must be 1 or 32
#endif // COMPLEX_BLOCK_SIZE

#define TWO_BY_TWO_COMPUTE(s)                                          \
  TWO_BY_TWO_PRELOAD(s)                                                \
  sum11XXreal += row1Xreal * col1Xreal;                                \
  sum11XXreal += row1Ximag * col1Ximag;                                \
  sum11XXimag += row1Ximag * col1Xreal;                                \
  sum11XXimag -= row1Xreal * col1Ximag;                                \
  sum11XYreal += row1Xreal * col1Yreal;                                \
  sum11XYreal += row1Ximag * col1Yimag;                                \
```

Porting xGPU to Setonix



- The xGPU correlator was also used on the now-decommissioned Garrawarla, a NVIDIA V100 GPU cluster, for offline correlation.
- Critical software component for high time resolution astronomy that represented a blocker to the migration process.
- Radio astronomers were very reluctant to move away from Garrawarla, and xGPU became a political symbol of ”the resistance”.

Porting xGPU to Setonix

We tried to port xGPU to AMD GPUs in two occasions by mainly using hipify-perl and then replacing some inline assembly code.

- In 2022 using ROCm 4.x in combination with the MI50 and MI100 GPUs. It resulted in runtime errors claiming not enough texture memory was available.
- In 2025, on Setonix's MI250X, using ROCm 5.7.x and 6.x, another runtime error occurring in the deprecated Texture Reference API.

```
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 3. GDB stack trace demonstrating the error comes from the TexRef API.

The BLINK implementation

```
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.

We decided to write a simpler correlator from scratch. It is part of the BLINK project that aims at running radio astronomy imaging pipelines entirely on GPU.

- CPU implementation parallelised using OpenMP.
- GPU implementation using CUDA/HIP (abstracted using macros).
- Simpler implementation than xGPU, but more flexible.
- Domain parameters can be specified at runtime, allowing researchers to use a single binary for difference science cases.

- Existing focus is on integrating the library within the existing ecosystem of software and data formats, rather than refine the implementation.

Benchmarking

To compare the BLINK implementation against xGPU, we correlate one coarse channel, 128 fine channels, for a second worth of VCS data. Integration time is 50 ms and channel averaging factor is 4.

- xGPU run on a NVIDIA V100 GPU
- BLINK GPU run on MI250X GCD
- BLINK CPU run on 2 AMD Milan 64-core CPUs (total: 128 cores)

Property	Value
Number of antennas	128
Fine channel resolution	10 kHz
Coarse channel resolution	1.28 MHz
Number of coarse channels	24
Sampling frequency	10 kHz
Data rate	7.37 GiB/s

Table 1. Main characteristics of a MWA VCS observation.

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 2. Execution times for the various correlation implementations tested in the paper.

The BLINK GPU kernel is slower than the xGPU one, but it hardly impacts the total execution time of the offline correlation program. Hence, ulterior investments into optimisations are not justified.

Power consumption

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 3. Energy consumption as measured by the HPE Cray hardware counters. When utilizing the whole GPU node, the energy consumption normalized per correlation task.

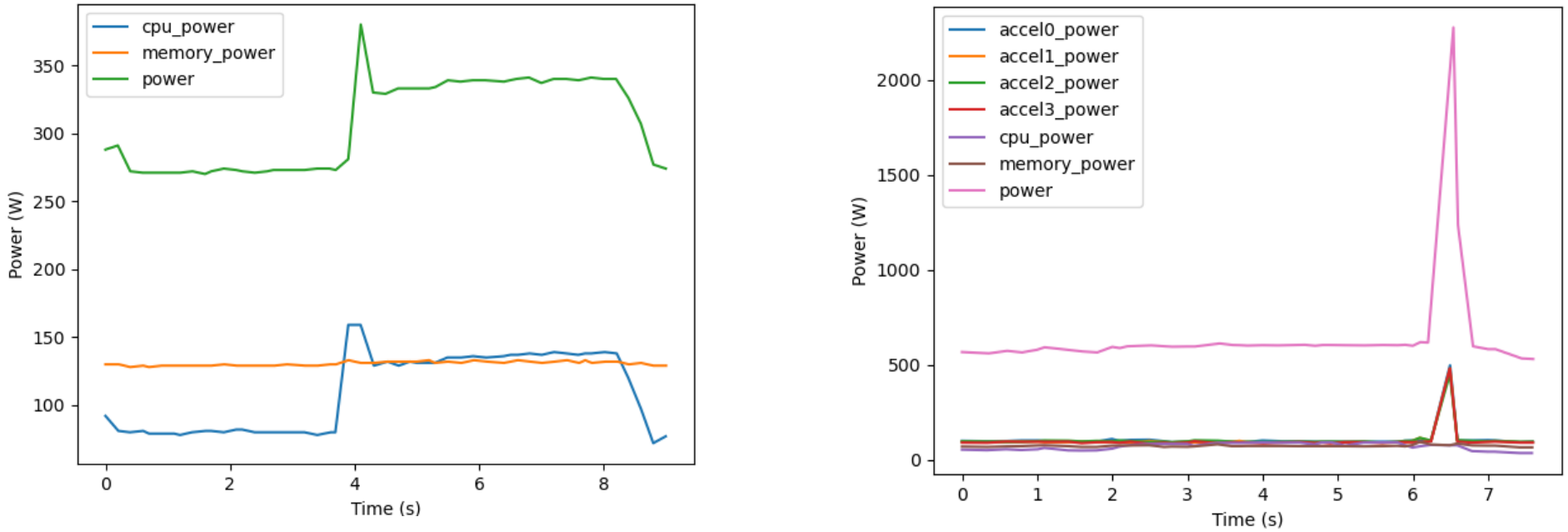


Figure 3. Power consumption over time of the BLINK correlator when run on a CPU node (left) and GPU node (right).

Conclusions

- xGPU is a very fast NVIDIA GPU implementation of the correlation algorithm but it has not been maintained for a long time.
- Highly optimized code can be challenging to port to new architectures and the effort required might not be justified compared to writing a simpler implementation (at least to start with).
- Development is focused on providing a good interface to other software and on handing different data formats.