GPU FX Spectrometer using CUDA

Abstract

The next generation of radio telescopes, such as Square Kilometer Array and the associated Pathfinder arrays, require vast amounts of computation due to the extremely large number of interferometers and the imaging requirements. The hardware for this computation is becoming a significant consideration in array design, both in terms of initial cost and power consumption. Graphics Processing Units (GPU) provide power efficiency and affordability as well as the flexibility of general purpose hardware. This work implements a GPU-based FX spectrometer, which processes four streams of 8-bit interferometer data, for a variable number of frequency channels. This approach scales well with frequency channels, with a computation speed up to 18 times faster than those of a CPU implementation. Further work is in progress to scale the algorithm with the number of interferometer streams, and to investigate optimisation of the GPU algorithm.

Background

Interferometry combines the signals of multiple telescopes to obtain higher angular resolution than that which could be produced by a single telescope. The processing involved in combining these streams, correlation, is computationally intensive. Correlation can consist of a number of stages starting with the packed data streams from the telescopes through to obtaining the final image. The GPU FX Spectrometer presented here starts with the packed data streams, and produces accumulated output, ready for gridding.

Algorithm

The algorithm performs three main steps on the GPU: unpacking, the fast Fourier transform, and accumulation. These steps are shown in Figure 2. Unpacking the data consists of converting the data from its 8 bit representation to a 32 bit floating point. The fast Fourier transform is applied in segments to all streams, the length of the transform is set at the beginning of the algorithm. In the final step, the data is cross-multiplied per frequency channel for each pair of streams (including self pairings) and then accumulated. Two approaches were tried in this stage: 1:n, where each thread calculates all pairings for all n incoming streams for a frequency, and 1:k where each thread calculates all pairings for a subset k < n of the incoming streams.

Results

The GPU FX algorithm was implemented in CUDA. Testing was performed on 1 GB of data obtained from the MWA as described in Figure 1. Results were obtained for a range of transform lengths. These are shown in Figure 3, as the rate of samples from each individual tile stream that the GPU can keep up with for the given parameters.

Discussion

As this algorithm has the capacity to be made parallel in time using time-slicing, additional performance may be attained by using multiple GPUs. The 1:n approach was limited to a small number of streams, because the number of registers required scaled beyond those available on current hardware. The 1:k approach uses shared memory, and can scale until the algorithm is limited by the total amount of global memory. The poor performance for low length is caused by a low number of blocks, and for the 1:k method this effect will diminish as the number of streams increases.

Acknowledgements

The authors thank Frank Briggs of the Research School of Astronomy and Astrophysics at the Australian National University for providing the MWA data and comparative CPU code used in this research.