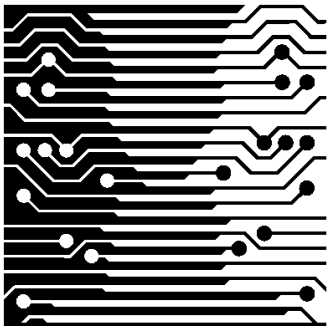
CUDA-Squeeze

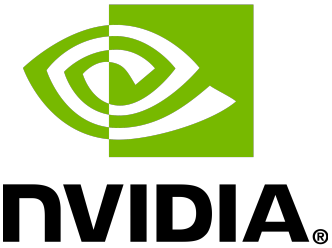
Benjamin Hugo
Department of Computer Science
University of Cape Town
bhugo@cs.uct.ac.za

Brandon Talbot
Department of Computer Science
University of Cape Town
btalbot@cs.uct.ac.za

Heinrich Strauss
Department of Computer Science
University of Cape Town
hstrauss@cs.uct.ac.za

10 June 2013





1 Project Description

The SKA (Square Kilometer Array) South Africa requires a compression algorithm that offers reasonable compression ratios, as well as being fast enough to keep up with the antenna output rates. This need arrises due to the requirement of saving streaming data onto a storage system and reading queries from that system simultaneously. This is where the focus of our project lies. We will be taking streaming data from correlators and beamformers (which combines pairs of data channels from the antennae) and compress the data-stream using GPGPU (General-Purpose Computing on Graphical Processing Units) in order to achieve the speeds that SKA requires, while maintaining reasonable compression ratios. Refer to Figure 1 for an overview of the data pipeline.

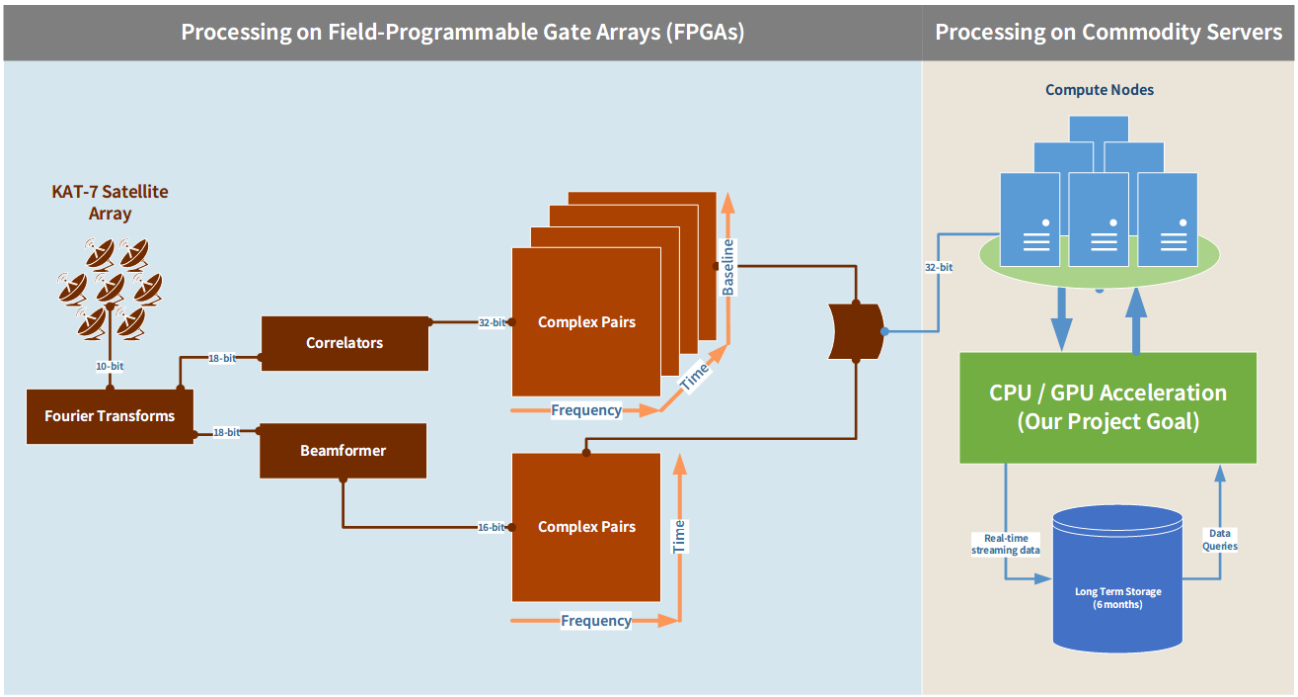


Figure 1: Simplified overview of the SKA data pipeline

CUDA-Squeeze is a scientific experiment to test which compression method run on an optimised CPU- or GPU-code base would be the most efficient for the frequency data produced by the SKA radio telescope array. We will be testing 3 different types of algorithms between the 3 team members. For each of the algorithms we will be creating a CPU (Central Processing Unit) P-Threaded (Processor Thread) version of the algorithm with optimizations for SSE and AVX and a GPU-based algorithm. We will then determine if the GPU-based versions have the required speedup factor. Finally we will determine which of the 3 algorithms is best suited to SKA's intended purpose of real-time, in-path data compression and decompression.

1.1 Problem Statement

The sample rate of the incoming 32-bit floating-point data is expected to be in excess of 15 terabits per second for the KAT-7 array alone. This sample rate has a quadratic growth rate as the number of dishes is increased (since the input data is correlated between pairs of antennae). We are testing the feasibility of using a highly multithreaded coprocessor such as GPU to perform bulk real-time compression where realtime data throughput enjoys priority over excellent compression ratios. Due to this quadratic growth rate even a small percentage saving will have a significant impact on I/O capacity, both in terms of transfer rates and storage capacity. Our stated goal is to determine whether this is possible, given the nature of the data.

2 Procedures and Methods

2.1 Predictive Methods

Predictive methods use a predictor function to attempt to predict what the next value would be. They tend to use two or more of the previous values to determine what the next value could be. The algorithms then find the difference between the predicted value and the actual value for the next number. Knowing how many zeroes before the initial one in the binary representation means they can shorten those zeroes by encoding them as a run-length encoder would, if run on that sequence.

This means that for data that has many values that are close to each other, we could achieve a very good compression ratio. A problem arises if the data has huge differences and spikes that ends up either not compressing at all or actually increasing the size of the data.

SKA has said their data will mainly low values very close to each other, and there should be very few spikes in the numbers. This means that predictive methods are viable for our circumstances.

Additionally, floating-point numbers are not denormalized at any point in the processing pipeline, yielding a consistent number format. This will improve the predictability of the data.

2.2 Adaptive Huffman Coding

Huffman coding allocates codes to each unique value in the data, this code being shorter (smaller) than the value it is allocated to. It creates a table that stores the codes allocated to the specified value. It allocates a shorter code to the most commonly seen value in the data.

Once it has created all the codes for all unique values in the file, it replaces the value with the code, thus shortening (compressing) the data. It also stores the table in the beginning in order for it to decompress the data.

Huffman coding is seen in many aspects of compression. The most common of these compression algorithms being video compression algorithms, in conjunction with DXT trees.

2.3 Zero-Length Encoding

Zero-Length Encoding is basically run-length encoding, just performed on the binary representation of data. In Zero-Length Encoding you run through the data and replace sections of the binary stream where there are multiple 0s or 1s with the number of repetitions and what is repeating, Eg. 1000001 becomes $1(5)01$. where the (5) is a tag to say it is a number representation and the 5 to tell how many 0s there are.

This is a viable option because the data samples we receive are skewed towards the least-significant bits. Most of the time they will be fairly small numbers and therefore will not use many of the mantissa bits. Values that cannot be fitted into 32-bits are flagged as unusable in the context of radio astronomy (these spikes are for the most part caused by ground-based or satellite interference), so these are excluded from compression in most cases.

2.4 Data Throughput

Data throughput through the processing pipeline will be a limiting factor. At present, commodity hardware allows for around 16 gigabits per second (per direction) across the PCI Express Bus (theoretical). With data rates per antenna approaching 2 terabits/s, there will likely not be enough available bandwidth to read the data in real-time. This is noted as a potential risk to the project. Future hardware evolution may negate this concern.

The stated aim is to process the data at a line-rate of 90% of 40 gigabits per second, as that is the network I/O rate at the Compute Nodes. It is, however, unlikely that these rates will be sustained for a significant time during a single astronomical observation.

We intend to perform testing on hardware in the Honours laboratory and, potentially, the compute clusters available. The Lab hardware is based on commodity-hardware, generally an Intel DQ57TML mainboard, using the onboard Intel 82578DM 1 gigabit per second NIC. The data will be processed through a Quad-Core Intel Core i3 or i5 3.00 GHz processor and GPU tasks will be offloaded to a CUDA Compute 2.0 capable device,

generally an nVidia GeForce GTX 560 Ti or equivalently equipped machine. While the available bandwidth across the PCI Express bus is high in comparison to the rest of the hardware, we need to efficiently transfer that to the GPU in chunks of $1 - 3$ gibibytes, due to current hardware limitations. This will likely introduce latency into the computation pipeline, which we will have to mitigate as best possible.

As part of the GPGPU processing paradigm, technologies are currently being developed to transfer data directly from I/O devices to the GPU to bypass the throughput deterioration through the mainboard RAM. These technologies are not yet commercially available. Addressing the data-throughput issues arising from these should, therefore, be considered outside the scope of our proposal.

Given that the data rate needs to approach 40 gigabits per second, reading from Hard-Disk or Network Interface Card will not be enough to keep data buffers full. We will therefore read the data from disk into Host memory and compress permutations along Floating-Point boundaries to "randomize" the input. This will allow for the highest data-throughput rate to the CPU and GPU, where the compression rates will be benchmarked.

We may assume that data-errors do not occur, since ECC RAM is used everywhere within the Compute Nodes, as stated by the SKA Office.

We do not consider the speed of the disk subsystem in this project, as this will be handled outside of the input and output streams for our implementations. As such, storage and database storage components are a specific exclusion to this project.

2.5 Testing and Evaluation Methods

A few samples of raw data (in $1 - 3$ gibibyte blocks) will be run through each CPU and GPU implementation of each algorithm. This will give us the compression ratio *compressed data*:*uncompressed data*. This will be used to gauge the improvement of the algorithm over raw throughput for data storage. More importantly, the algorithm has to run at real-time rates. This will be evaluated by subtracting the time to copy data to and from the system memory from the time to execute the execution run in entirety.

2.6 Ethical, Professional and Legal Issues

The sample data will remain intellectual property of the SKA. There are no known U.S. or international patent restrictions on the methods we mentioned above. No ethics clearance will be required for our project.

2.7 Related Work

The zero-length compression scheme we described has been used successfully employed by Abadi et al. [1] to reduce database sizes, under similar circumstances (where the run-lengths of leading zeroes is sufficiently large to truncate normal 4 byte-integer values down to 3 or fewer bytes).

The predictive methods described by O'Neil et al. [6] and Ratanaworabhan et al. [3] are two parallel implementations of 64-bit double compression that will work on chunks of streaming data. The GFC algorithm specified by O'Neil achieves 75 gigabits per second but trades compression ratios for speed, where the parallel FPC compressor achieves much better compression ratios at half less than half that speed. However, the FPC compressor uses a lookup table to predict, which makes it too memory intensive for our purposes.

Huffman coding is a well-researched technique, used in many contemporary compression implementations. Knuth[5] and Gallagher[4] independently suggested various ways of localising the Huffman lookup-table, and Vitter[8] shows that the lookup size is bounded by a factor of two and that incrementing or decrementing the lookup-table indices can be accomplished in $O(1)$[2]. This implies that a compression factor would likely be established for Adaptive Huffman Coding, even with a single pass.

2.8 Dynamic Algorithm Selection

Should the compression rates be different between different kinds of observation, it would be useful to SKA South Africa to dynamically select the compression algorithm based on observed data.

The feasibility study for this is considered an inclusion in the scope of our project.

2.9 Altera OpenCL compiler

Altera, the FPGA hardware vendor, have plans to introduce an OpenCL compiler to program the FPGAs in use by SKA. This would unify their code-base and they will investigate this once it becomes available.

This, and the associated OpenCL implementation, is considered outside the scope of our project.

3 Project Plan

3.1 Deliverables

There are several major milestones in our project. They primarily consist of building 3 viable GPU-based compressors and 3 multi-threaded CPU implementations to use as a baseline in further analysis. The basic CPU versions will serve as a judgement of the feasibility of the undertaking and has to be completed by the 1^{st} of July. Work on optimization will be undertaken while developing the basic CPU versions and will be due by the 30^{th} of July. Work on the GPU implementations will commence after the individual literature surveys have been expanded and some initial architecture and experimental designs have been established. The GPU versions will require a significant investment of time and profiling to optimise and will run throughout much of the second semester. We will be testing our solutions concurrently, as well as documenting the required results. These will range from unit testing to profiling. The final comparison writeup is due on the 25^{th} of September. In this we hope to combine our primary results in terms of compression throughput and ratios. See the Gantt chart (Figure 2) for the other less significant deliverables.

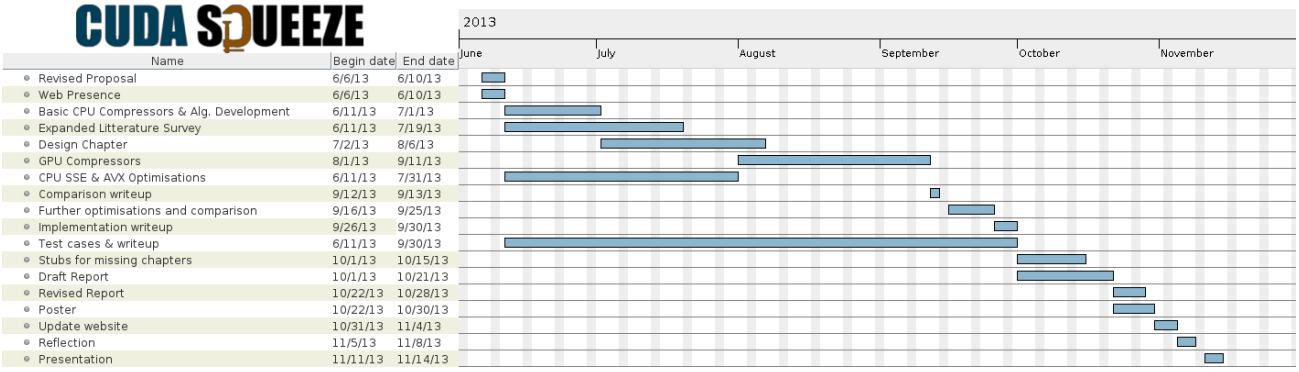


Figure 2: Project Timeline and Deliverables

3.2 Work Allocation

At the end of the project we expect to have both GPU-based and P-Threaded (CPU) versions (SSE- and AVX-optimised, where applicable) of the following algorithms:

- **Implementation of a GFC predictor:** to be investigated by *Benjamin Hugo*
- **Parallel block-based Huffman coding:** to be investigated by *Brandon Talbot*
- **Zero-Length Encoding:** to be investigated by *Heinrich Strauss*

One of the major design issues is deciding on how to divide the data up per graphics card, where we are limited in terms of L1 cache size and data transfer rates through the PCI-Express bus. These constraints play a major role in achieving any significant speedup from the GPU.

We will compare performance between the 3 algorithms in terms of throughput, memory usage and compression ratios. These results will determine whether it is feasible to use GPUs as compression co-processors in the data cluster.

The project will be deemed successful if we can determine which one of the basic algorithms is a good candidate for compression on a co-processor or whether moving the compression onto a co-processor is

feasible. Our focus is specifically on the compression itself on a single co-processor without consideration of scaling this over multiple nodes.

3.3 Technical Resources

The GPU implementations will be run on CUDA compute capability >2.0 devices, such as those found in the CS Honours laboratory. It will be preferable to run the profiling on the systems with higher capability in terms of number of CPU cores and newer CUDA compute capabilities, such as the ICTS HEX compute cluster or the Center for High Performance Computing cluster.

Obtaining enough raw testing data should not considered an issue. The SKA Office have committed to providing us with very large samples (in excess of 100 gibibytes, if necessary).

3.4 Human Resources

Associate-Professor James Gain and Dr. Patrick Marais will co-supervise our project. They have co-authored several papers on compression schemes including compression on irregularly sampled height-fields, point clouds and molecular dynamics trajectory files.

Jason Manley is a Digital Design Engineer and Digital Signal Processing specialist at the SKA office in Pinelands. Jason has provided valuable insight into the SKA architecture and serves as our external advisor. Jason has kindly agreed to provide the sample data for our experiment.

3.5 Risks and Mitigation

We consider the complexity of implementation and optimization as the biggest risk to our project. As already mentioned the project is divided into 3 separate algorithm implementations. Each implementation will have its own CPU baseline written to which comparisons can be drawn. This ensures that the project is loosely coupled and that failure in some of the components can be tolerated.

The Risk Matrix (Figure 3) for the project is as follows:

impact \ *likelihood*	**Low**	**Medium**	**High**
Low	Benchmarking Resources		
Medium		Implementation	Optimisation
High			

Figure 3: Risk Matrix (Impact vs. Likelihood)

There is some technical risk to the project. Although there is sufficient hardware resources within the Honours laboratory for the scope of development, any comparison and benchmarking will require access to machines with larger capability in terms of CPU cores and GPUs with higher capacity, such as those provided on the “HEX” HPC cluster. Additional resources can be provided by the SKA Office, as they have communal compute nodes available at their offices in Pinelands. These resources will be arranged well ahead of analysis, and if they cannot be obtained we will perform benchmarking using the hardware we have at our disposal.

4 Glossary

- **AVX** - Advanced Vector Extensions; an extension to the Intel architecture implemented in Intel Sandy-Bridge and newer processors. The SIMD registers' sizes are expanded to 256 bits, instead of the 128-bit registers found in SSE. Requires a recent operating system to utilize (Windows 7+; Linux 2.6.30+; MacOS 10.6.8+).
- **Compression ratio** - compressed size / uncompressed size.
- **CPU** - Central Processing Unit
- **DXT Tree** - A data-structure used in lossy compression derived from S3 Texture Compression for GPUs.
- **Entropy** - (Information Theory) Entropy is the measurement of the information contained in a single base-n symbol (as transmitted per time unit by some source)[7, p. 46 - 47]
- **Gibibytes** - A measure of data size, using the binary-prefix definition of Giga-; 2^{30} bytes.
- **Gigabits per second** - A measure of throughput speed, using the SI definition of Giga-; 10^9 bits per second.
- **GPGPU** - General-Purpose computing on Graphical Processing Units
- **GPU** - Graphical Processing Unit
- **L1 Cache** - Level 1 Cache; In the Intel x86 architecture, the fastest, but smallest, CPU cache memory.
- **KAT / KAT-7** - A 7-dish Radio Telescope Array in the Northern Cape, ZA
- **MeerKAT** - A proposed 64-dish Radio Telescope Array planned for completion by 2016
- **P-Thread** - Processor Thread
- **Real-time Rates** - For the purposes of this project, being able to attain over 36 gigabits per second is considered "real-time"
- **Redundancy** - (Information Theory) The difference between the largest possible entropy of a symbol set and its actual entropy. Mathematically redundancy is defined as follows (n is the size of a symbol set and P_i is the probability that a symbol c_i is transmitted from a source)[7, p. 46 - 47]:

$$R = \log_2 n + \sum_{1}^{n} P_i \log_2 P_i$$

- **SKA** - Square Kilometer Array, the largest radio-telescope array in the Southern Hemisphere (over 3000 dishes) planned for first-operation in 2020.
- **SSE** - Streaming SIMD Extensions; A set of additional instructions introduced to Intel processors from the Pentium 3 onwards which allows the same instruction to be performed on multiple data targets, thus improving performance of an optimised program.

References

[1] D. Abadi, S. Madden, and M. Ferreira, *Integrating compression and execution in column-oriented database systems*, in Proceedings of the 2006 ACM SIGMOD international conference on Management of data, SIGMOD '06, New York, NY, USA, 2006, ACM, pp. 671–682.

[2] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, *A locally adaptive data compression scheme*, Communications of the ACM, 29 (1986), pp. 320–330.

[3] M. Burtscher and P. Ratanaworabhan, *pFPC: A Parallel Compressor for Floating-Point Data*, in Data Compression Conference, 2009. DCC '09., 2009, pp. 43–52.

[4] R. Gallager, *Variations on a theme by Huffman*, Information Theory, IEEE Transactions on, 24 (1978), pp. 668–674.

[5] D. E. Knuth, *Dynamic Huffman coding*, Journal of algorithms, 6 (1985), pp. 163–180.

[6] M. A. O'Neil and M. Burtscher, *Floating-point data compression at 75 Gb/s on a GPU*, in Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4, New York, NY, USA, 2011, ACM, pp. 7:1–7:7.

[7] D. Salomon, *Data Compression: The Complete Reference*, Springer-Verlag New York Incorporated, 2004.

[8] J. S. Vitter, *Design and analysis of dynamic Huffman codes*, Journal of the ACM (JACM), 34 (1987), pp. 825–845.