FFT Optimizations for GPU and Many-Core Architectures

By Seth Hall

AUT

High Performance Computing Research Lab

AUT

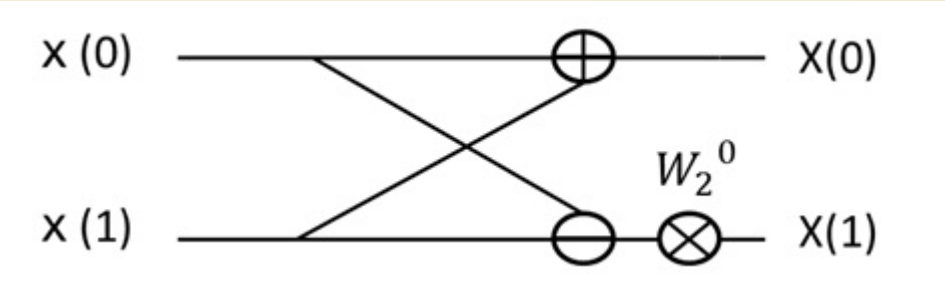
UNIVERSITY

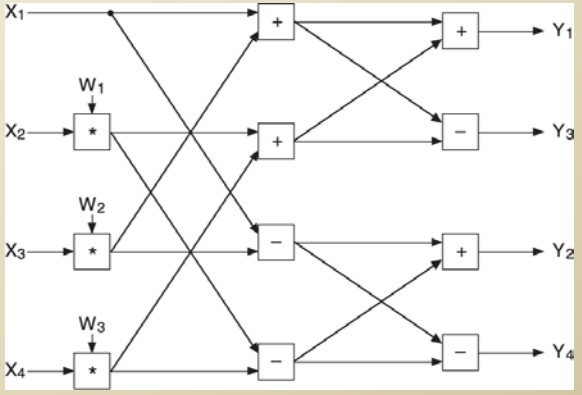
Who am I?

- New-ish AUT Lecturer
- 0.2 FTE on SKA under the HPC Research Lab
- Involved in software development
- Prototyping in GPU computing, many-core architectures and low-power parallelization.
- Document prototyping test reports during the course of the project.

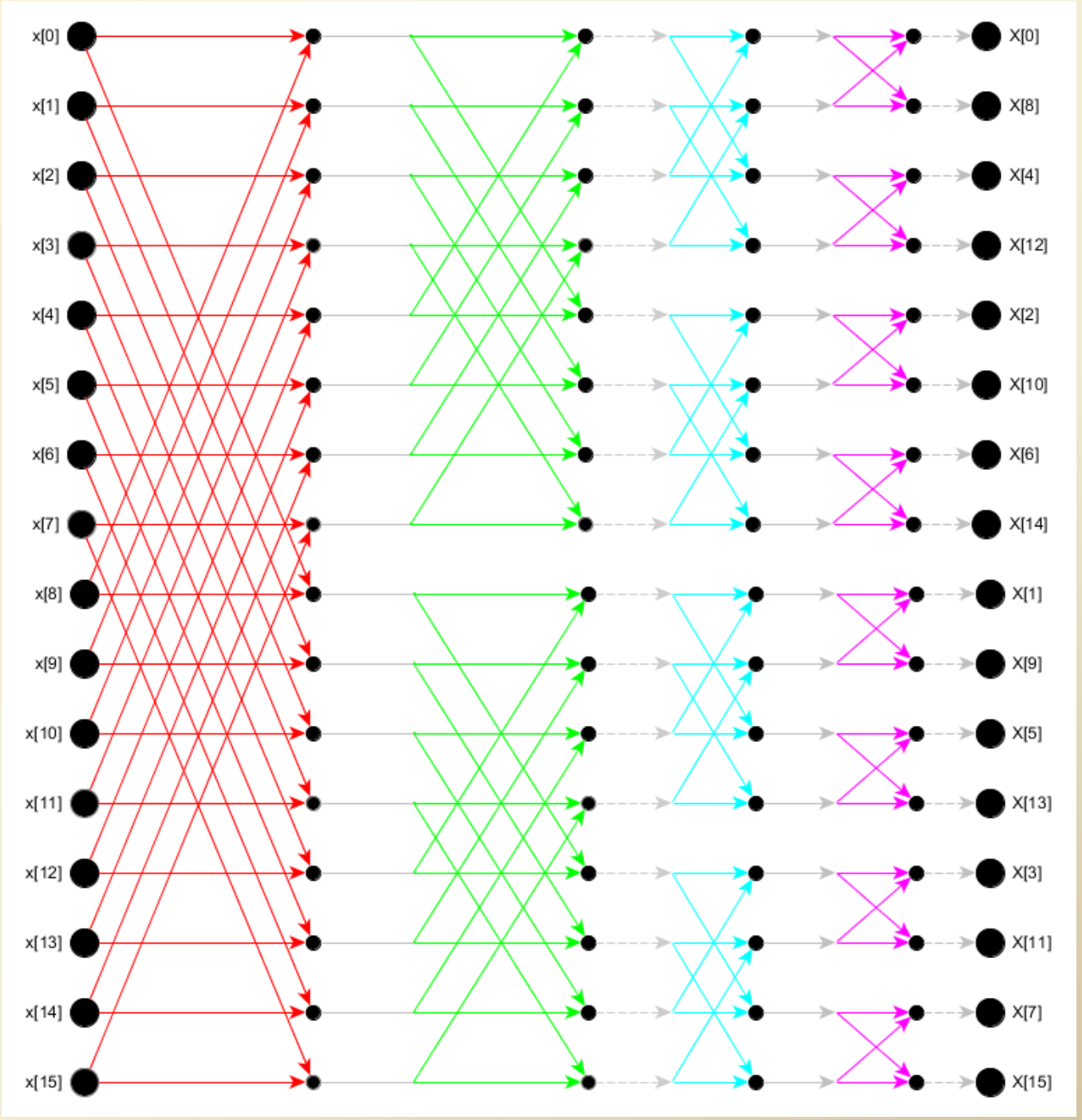
Fast Fourier Transforms (FFT)

- Main algorithm I am looking at is FFT
- FFT is an efficient way to do Fourier transforms which convert a signal between its **original domain** (often time or space) and its representation in the **frequency domain**.
- Used in the Correlator where digitized data from the radio telescope is collected and processed using FFT. Also used in Pulsar Searching and the Imaging Pipeline.





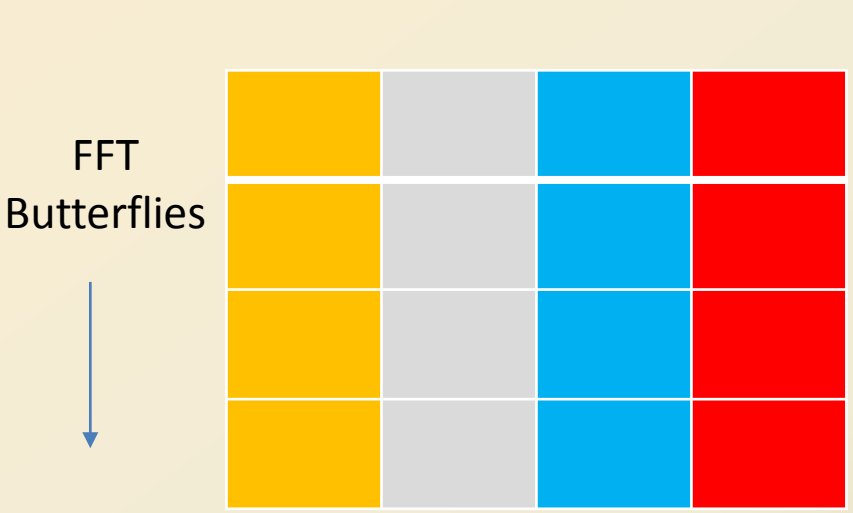
RADIX-2 Decimation In Frequency FFT

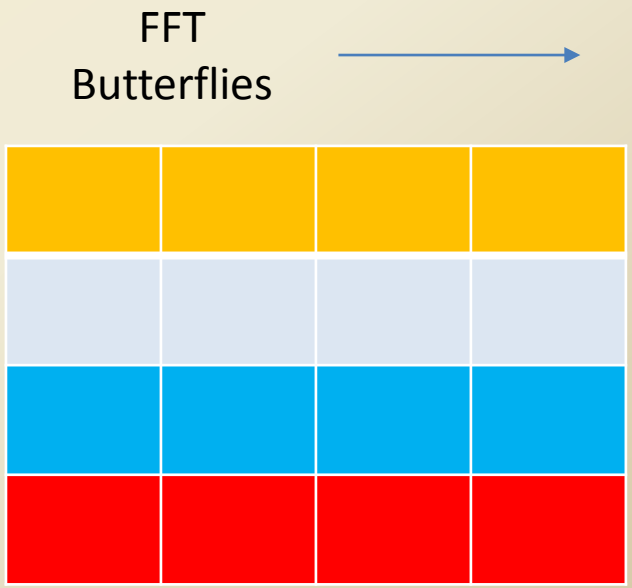


6 Step FFT

- For very large FFT's, Can be more efficient to break it up using a "6 Step" FFT
- For an *N* size FFT we can break it up into two sets of *m* x *n* number of FFTs. Eg for a 16 point FFT we can beak into a 4x4 or 2 x 8.
- Most of the testing in this work is done on 2^{18} point complex FFT. So 6 step done with 512x512 point FFTs
- We can choose the value of *m* and *n* based on hardware architecture (eg amount of memory on processing core cache).

1. Arrange data (Group same colours together).
2. Perform *m* lots of *n* point FFTs
3. Rearrange data (corner turn,gsame colours together).
4. Multiply all values by twiddle correction
5. Perform *n* lots of *m* point FFTs
6. Rearrange data back (corner turn)





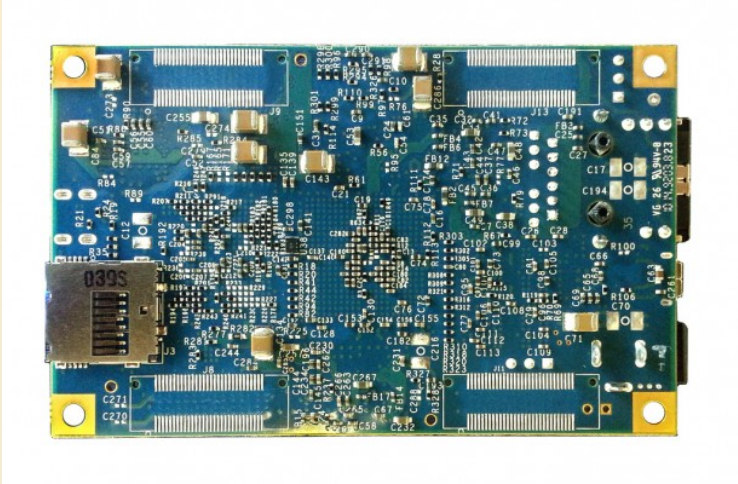
Low Power Devices Being Tested

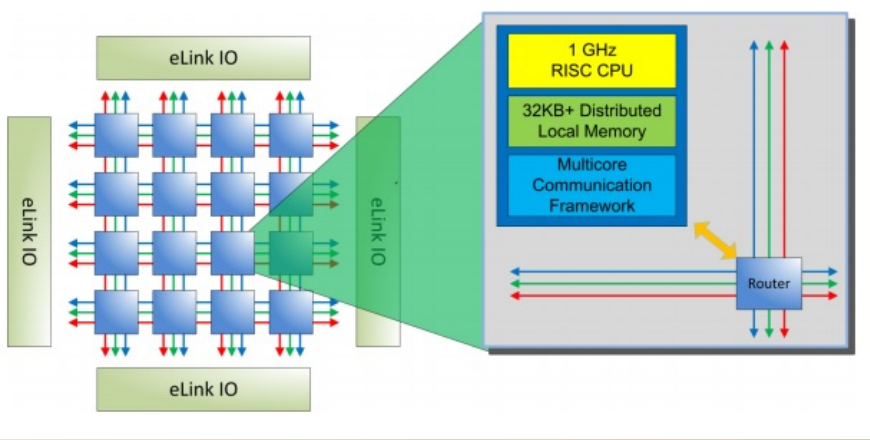
- Adapteva Epiphany Parallella
- NVIDIA Jetson TK1
- Kalray MPPA

NVIDIA®

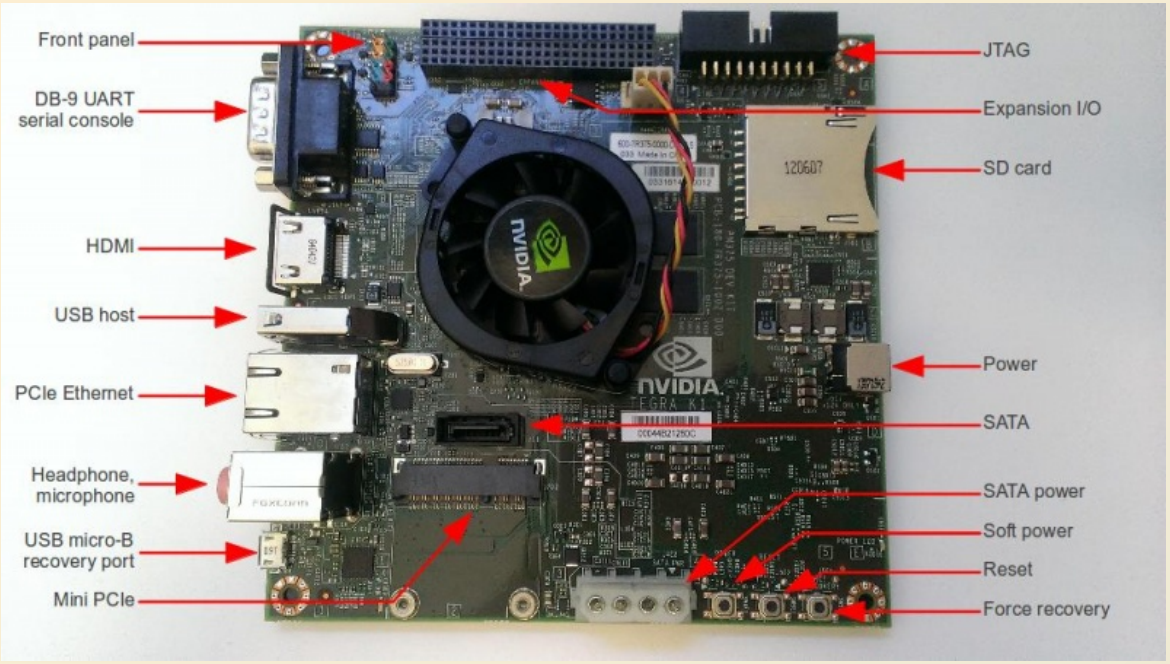
Adaptiva Epiphany Parallella

- Company founded via a Kick Starter campaign
- Dual Core ARM A9 Processor
- Epiphany coprocessor with 16 high performance RISC cores
- 1 GB RAM
- < 5 watt power consumption
- They have a 64 core product and developing a board with 4096 RISC cores





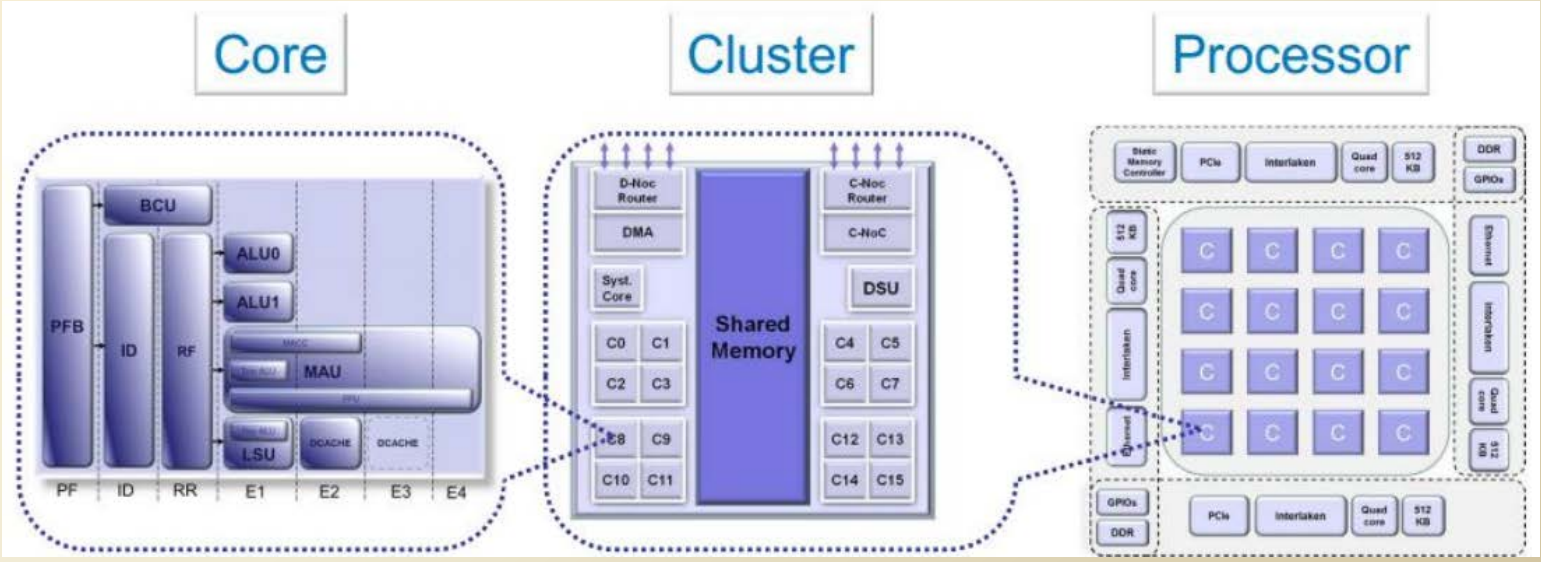
NVIDIA Jetson TK1



- Tegra K1 SOC
- NVIDIA Kepler GPU with 192 CUDA cores
- NVIDIA quad-core ARM Cortex-A15 CPU + low power companion core
- 2 GB RAM
- Power consumption: 11 watts

Kalray MPPA

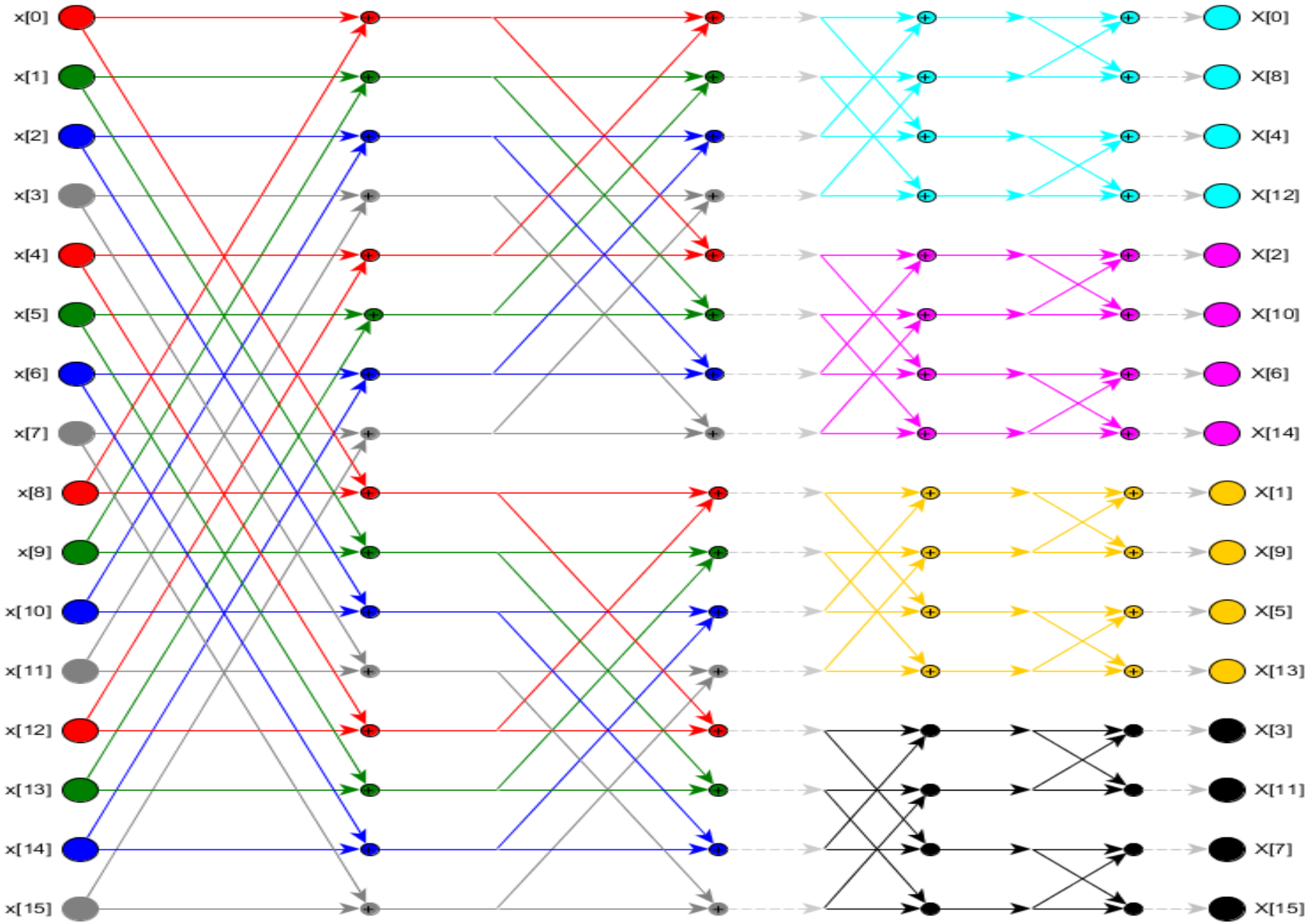
- Massively Parallel Processing Array.
- 5-10 watts power consumption.
- 256 cores.
- Work undertaken by Julien and Julien



Epiphany FFT

- Work in Progress
- Promising Architecture, Company claims to be #1 in terms of throughput / power consumption
- Claims 5 GigaFLOPS per watt on 16 core and 50 GigaFLOPS per watt on 64 core.
- Had several problems with the OS, lots of bugs!!
- Development environment and build tool problems.
- Very bad documentation.
- Tried several languages and libraries for FFT development on Parallella board including Epiphany SDK, Open CL, E-Python and **Epiphany BSP** (Bulk Synchronous Parallel) – works well…… so far

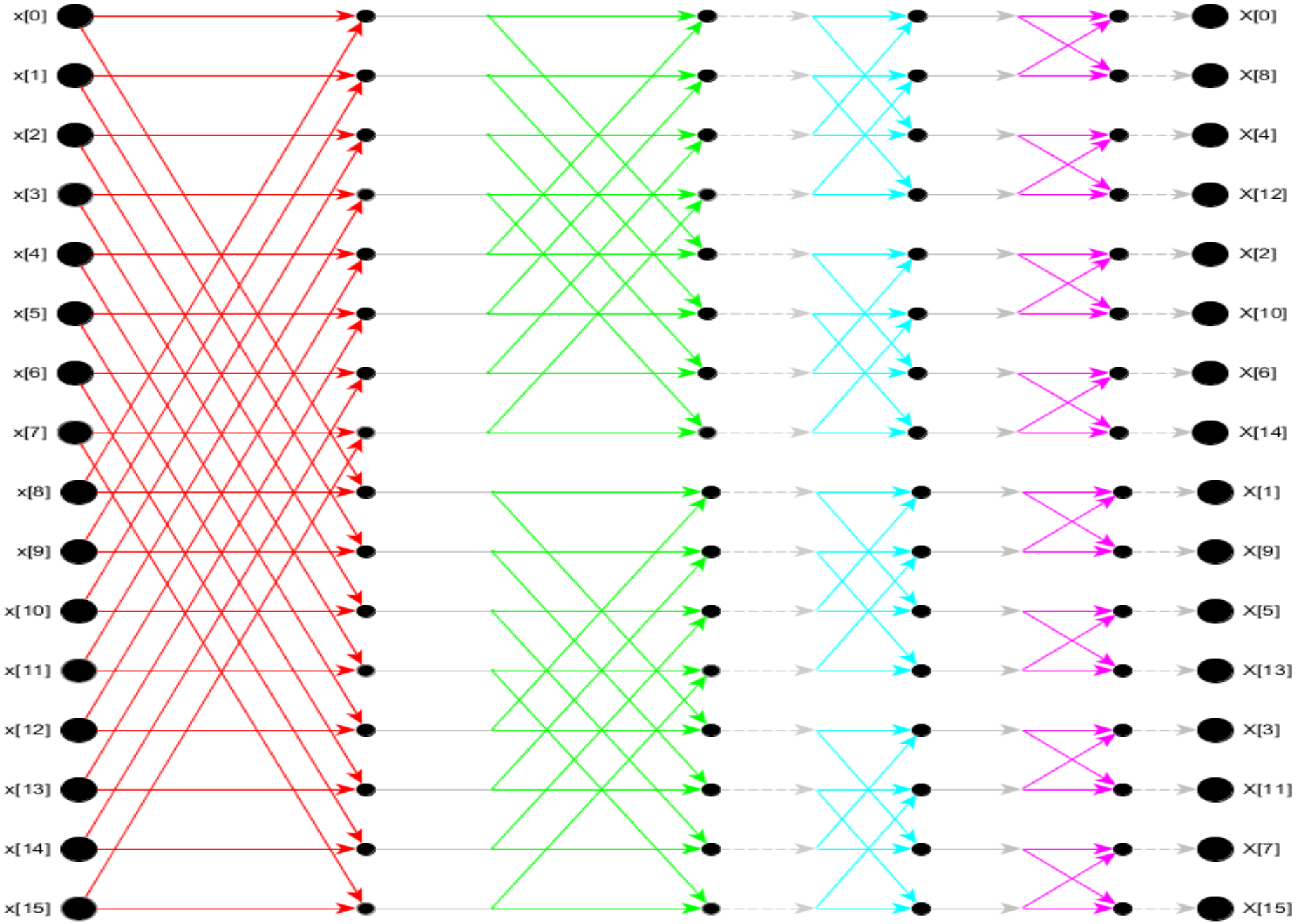
Multicore Parallelization



How things are implemented on GPU

- GPU's have many more cores to do computations in parallel.
- Great for mathematical computations
- Not good for code branching.
- GPU can load in blocks of data to onboard memory. However for large FFT's the Big butterflies can be a bit slow because GPU needs to "lookup" values across multiple blocks of memory.
- Six step help reduce memory requirements because of the smaller sized FFT

GPU Parallelization of FFT



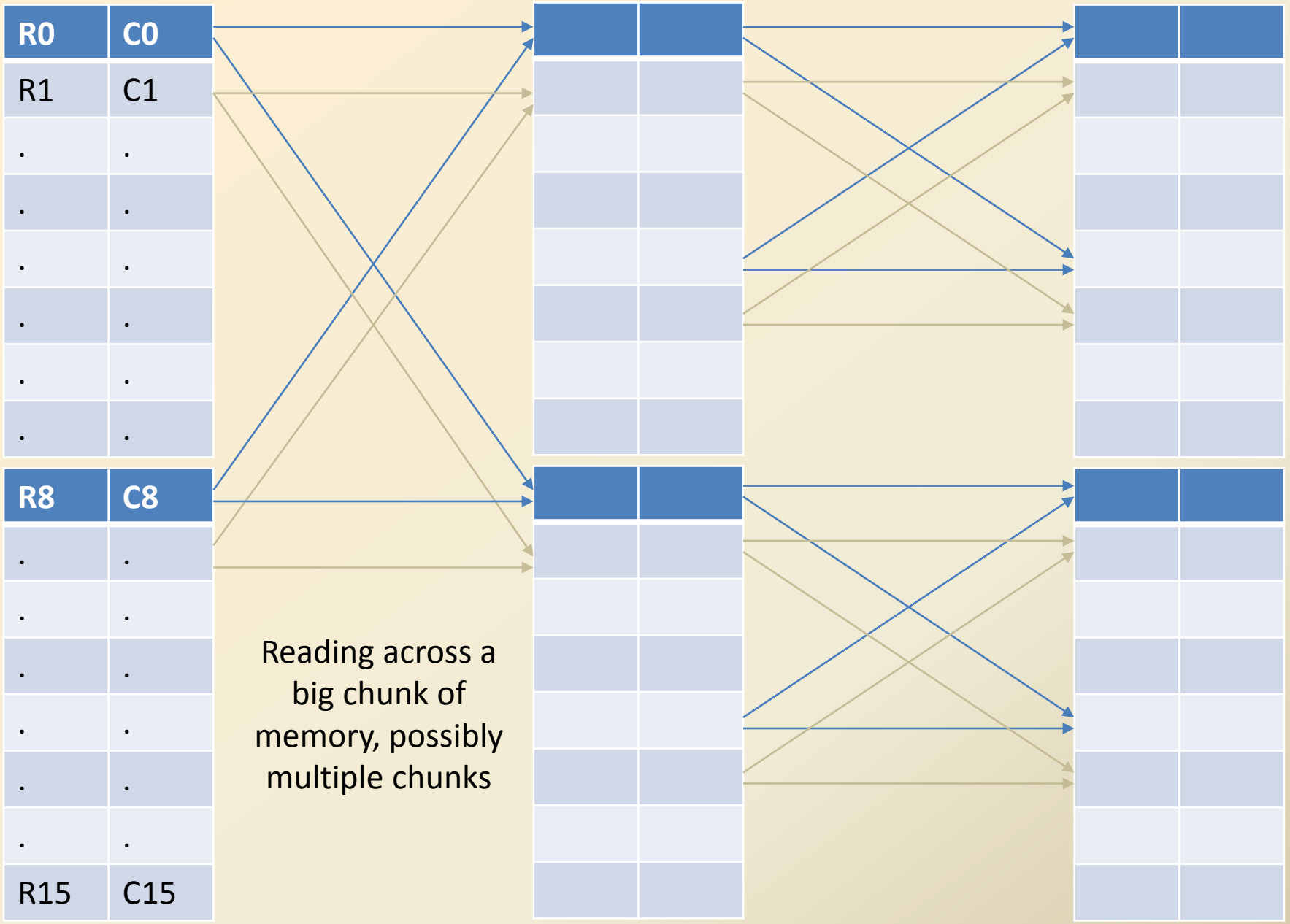
CUDA vs Fragment Shaders/Compute Shaders

- **CUDA** platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements
- On NVIDIA GPU architectures CuFFT library can be used to perform FFT
- Development very easy and the hard parts of FFT are already done.
- **Disadvantages**: CuFFT is closed source. CUDA only available with NVIDIA GPU

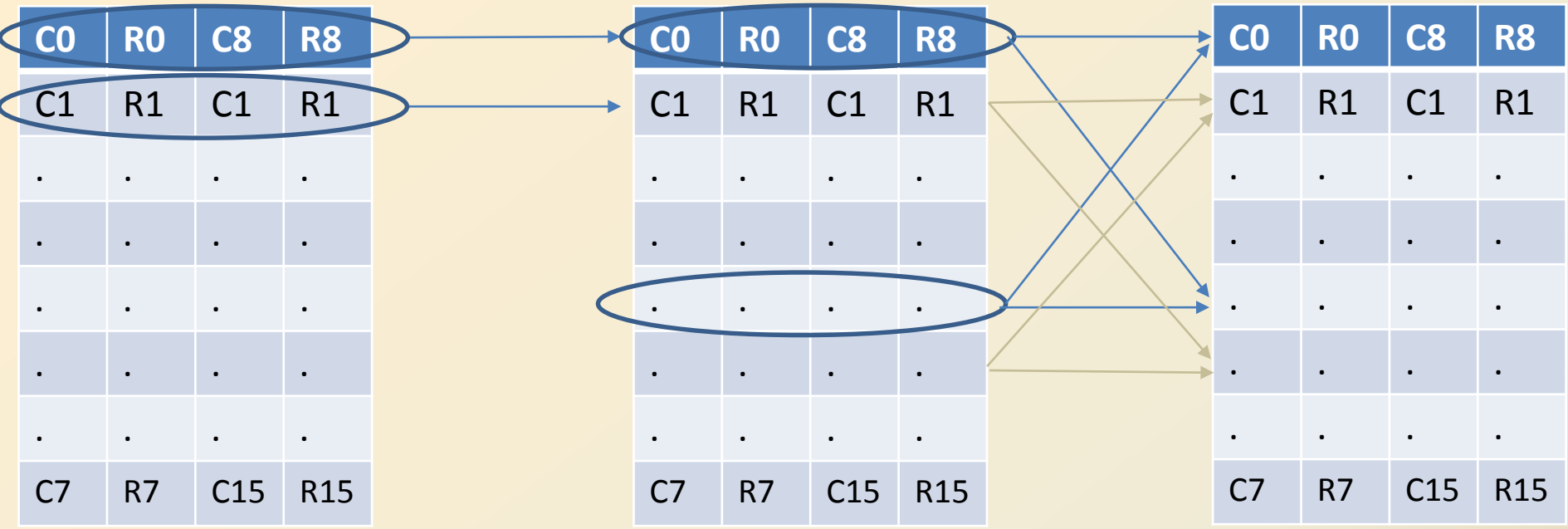
- Alternative: **Vertex/Fragment Shaders**
- Programmed with C/C++ and OpenGL 2.0 libraries and above
- “All” GPU can support.
- **Disadvantages**: Vertex/Fragment shader truly designed for graphics, so effectively we “disguise” our data as RGBA values. Development and debugging a bit of a nightmare. Struggles to run in “headless” mode. Must have display attached.
- Alternative: **Compute Shaders**! Very Open CL-esque. But only supported on “new” GPU. OpenGL 4.3 and above.

GPU Compute Shader Implementation

- A few optimizations I have done with Compute Shader FFTs
- Pre-computed Omega table and Bit Reversal Table
- GPU's can perform SIMD (single instruction multiple data) calculations. Hardware optimized to calculate 4 floating point values (RGBA pixel data, remember GPU's really designed for graphics)
- GPU's better with performing multiple calculations more than data lookups in the butterflies. So Radix 2 is used instead of Radix 4.
- Input Complex Data compacted into a VEC4. Reduces further the amount of data read across multiple memory chunks.. Each core can do 2 sets of butterflies at the same time.


R0
C0
R1
C1
R8
C8
R15
C15
Reading across a big chunk of memory, possibly multiple chunks

Stored as Vec4



Butterfly on first pass in same chunk of memory, can save on computation times by performing SIMD vec4 operations on complex numbers

Next step, each core processing the data can do two butterflies at the same time. Butterflies now smaller so less chance data across different chunks of memory

Performance of Compute Shader vs CUDA CuFFT

- Some good news, execution timing of optimized Compute Shader FFT seems very fast and possibly could be a little faster than CuFFT
- Bad news… There are some technicalities to solve to efficiently transfer data between GPU – CPU. ☹

Why Embedded GPU/Multicore?

- Why not a more powerful machine? EG For GPU Computing a K40…
- For SKA, we also need to worry about power efficiency.
- Tests have shown that embedded mobile GPU such as the Tegra K1 is more efficient in terms of FLOPS per Watt than the Tesla K40

GPU Performance Comparison

	Tesla K40 + CPU	Tegra K1 SOC
Single Precision Peak	4.2 TeraFLOPS	326 GigaFLOPS
Single Precision Matrix Multiply	3.8 TeraFLOPS	290 GigaFLOPS
Memory	12GB @ 288 GB/s	2GB @ 14.9 GB/s
Power (CPU + GPU)	385 Watts	11 Watts
FLOPS PER WATT	10 GigaFLOPS	26 GigaFLOPS

262144 point complex to complex FFT (single precision) (CUDA)

GPU Test	FFT Throughput	Power
Tegra K1 SOC	9.81 mS	11 Watts
Tesla K40 + Xeon 85 CPU	0.64 mS	300 Watts

K40 ~15 times FFT throughput efficiency over TK1

K40 ~27 times power consumption of TK1

What Next, Future Work

- Continue developing FFT for Epiphany
- For TK1 utilizing Compute Shaders, work out delay in data transfer from CPU to GPU.
- Test performance on NVIDIA X1 (next step up from TK1)
- Finalize which low power multicore architecture is best in terms of FFT throughput performance and power consumption.

Questions?