ENCODING HIGH-THROUGHPUT JPEG2000 (HTJ2K) IMAGES ON A GPU

Presenter: Aous Naman
Authors: Aous Naman & David Taubman

Session: COM-05.11 -- Lossy Coding of Images & Video

School of Electrical Engineering and Telecommunications,
The University of New South Wales (UNSW), Sydney, Australia

Synopsis

- High-throughput JPEG2000 (HTJ2K)
- A new addition to the J2K suite of image coding tools – JPEG2000-Part 15
- HTJ2K introduces a new block coder (entropy coding)
  - More parallelism, lower complexity
  - Significantly faster than conventional; $\approx$ 10x faster block coding.
  - Overall speedup >6x at low-bit rates to >30x for lossless
  - More efficient than JPEG, faster on a single core, and highly parallelizable to multiple cores
  - On i7 6700, can encode 12bit 4K 4:4:4 @ 2bits/pixel at 123fps, decode at 126fps
  - Lower coding efficiency: BD-Bitrate $\approx +7\%$ or BD-PSNR $\approx -0.7\text{dB}$
  - Limited quality scalability; we still have accessibility and resolution scalability
- HTJ2K maintains transcoding compatibility with J2K
- Supported by Kakadu 8.0, and OpenJPH (github.com/aous72/OpenJPH)
- A previous work explored HTJ2K decoding on a GPU.
  - For a 12bit 4K 4:4:4 @ 1bits/pixel, up to 770 fps on GTX1080 (Today's mid-range).
- This work explores HTJ2K encoding on a GPU.
  - Same sequence, up to 450 fps on GTX1080 (Today's mid-range).

UNSW SYDNEY

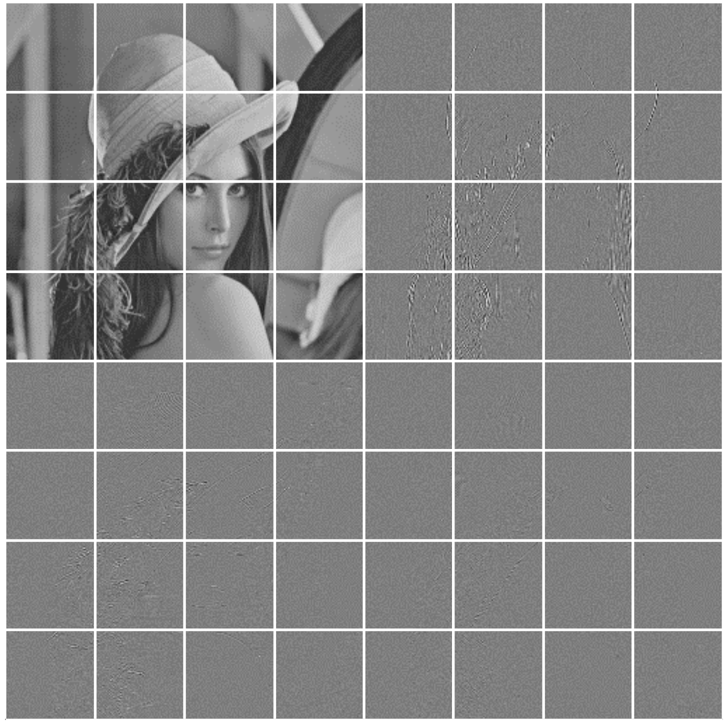
JPEG2000 Pipeline

- JPEG2000 pipeline comprises
  - Color transform – to represent image in a form more amenable to compression
  - Wavelet transform – exploit spatial redundancy
  - Subbands are subdivided into codeblocks – say 64x64 wavelet coefficients.
  - The codeblock coder (entropy coding) operates on codeblocks.

Original
RGB

Y CB CR

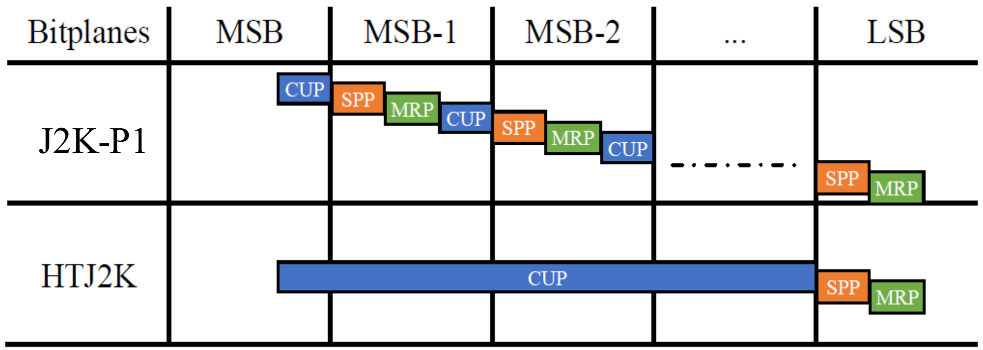
64x64
Codeblock



After Wavelet Transform

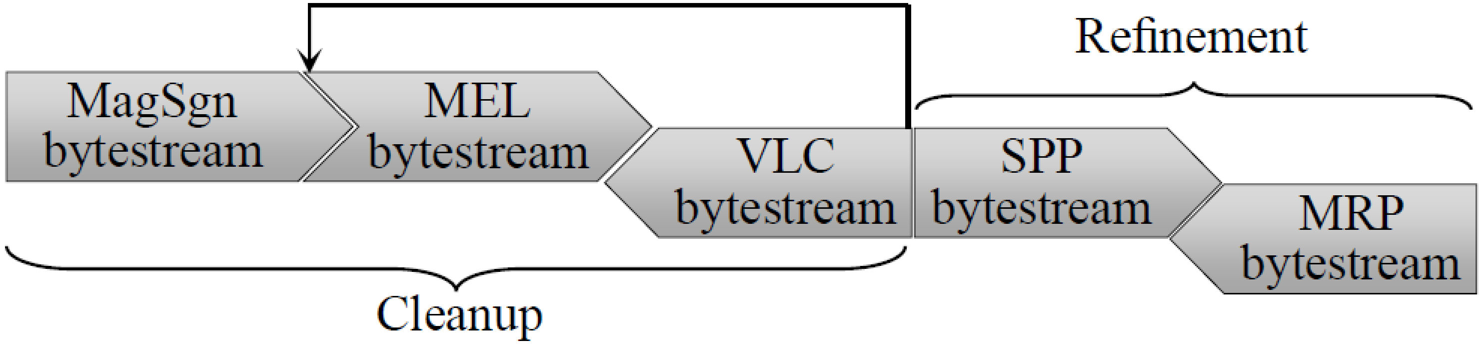
HTJ2K Coding Passes

- JPEG2000-Part1,2 employs a fractional bit plane adaptive arithmetic coder
  - Bit-planes are coded in three passes, known as
  - Significance propagation pass (SPP)
  - Magnitude Refinement pass (MRP)
  - Cleanup pass (CUP)
  - This provides many truncation points for the codeblock bitstream during RD optimization
- HTJ2K employs a different block coder
  - The cleanup pass encodes many bitplanes
  - Optional SPP, MRP – enables transcoding, and finer truncation point granularity.
  - This work employs the cleanup pass only.
  - No rate control is employed, but HTJ2K supports single-pass precise rate control



HTJ2K Codestream Segments

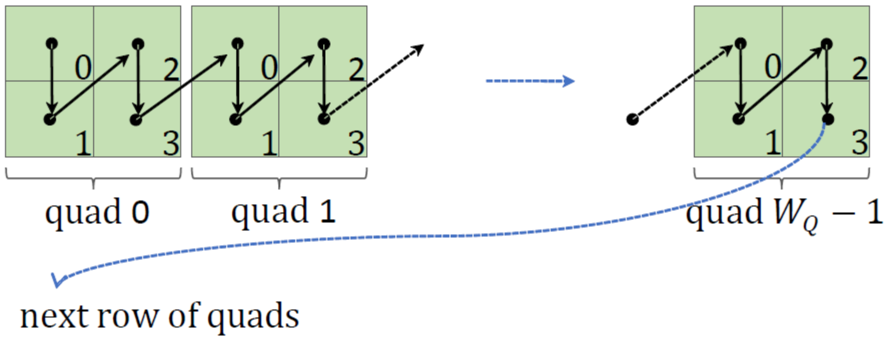
- HTJ2K CUP codestream is made up of
  - A magnitude-sign segment (MagSgn)
  - A MEL segment
  - A VLC segment
- The HTJ2K can also have SPP and MRP



- Having multiple segments give the encoder/decoder opportunity to concurrently work on different segments – better parallelism.
- Coding efficiency → efficiently coding locations of non-zero coefficients, and information about coefficient magnitude.

The VLC segment of HTJ2K

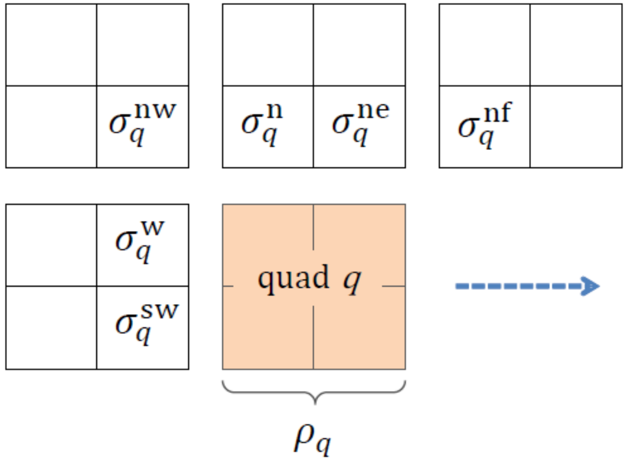
- HTJ2K cleanup pass encodes coefficients in 2x2 groups, known as quads



- The VLC segment interleaves
  - CxtVLC: Context adaptive variable-to-variable code
    - at most 7 bits/quad.
  - UVLC : u_q values – next slide

- The context is made of previous causal quads

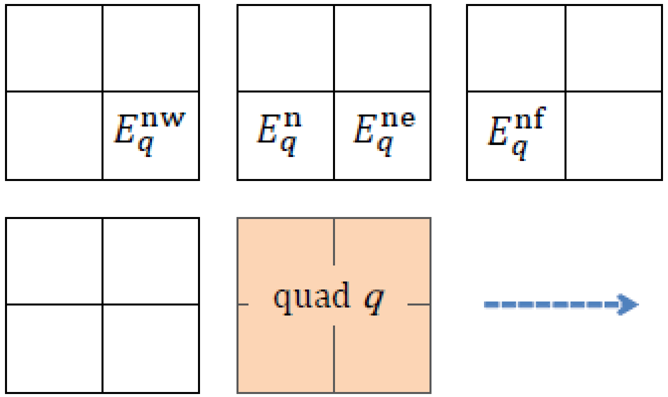
$$c_q = (\sigma_q^{\mathrm{nw}} | \sigma_q^{\mathrm{n}}) + 2(\sigma_q^{\mathrm{w}} | \sigma_q^{\mathrm{sw}}) + 4(\sigma_q^{\mathrm{ne}} | \sigma_q^{\mathrm{nf}})$$



- Decoding CxtVLC produces
  - ρ_q (4 bits): locations of non-zero samples $\mu_p \neq 0$ in quad q
  - u_q^{off} (1 bit): existence of u_q for quad q
  - $\bar{\epsilon}_q^{\mathrm{k}}, \bar{\epsilon}_q^{1}$ (4 bits each): EMB code – next slides

The MgnSgn Segment (1/2)

- This segment communicates coefficient values – bit packed
- Quantized coefficient is written as an unsigned values μ_p, and a sign $s_p \in \{0,1\}$
- The encoder encodes $2(\mu_p - 1) + s_p$
- We define
  - an exponent E_p as the number of bits needed for $2(\mu_p - 1) + s_p$
  - the maximum exponent $E_p^{\max}$ in quad Q_q, given by $E_p^{\max} = \max_{p \in Q_q} E_p$

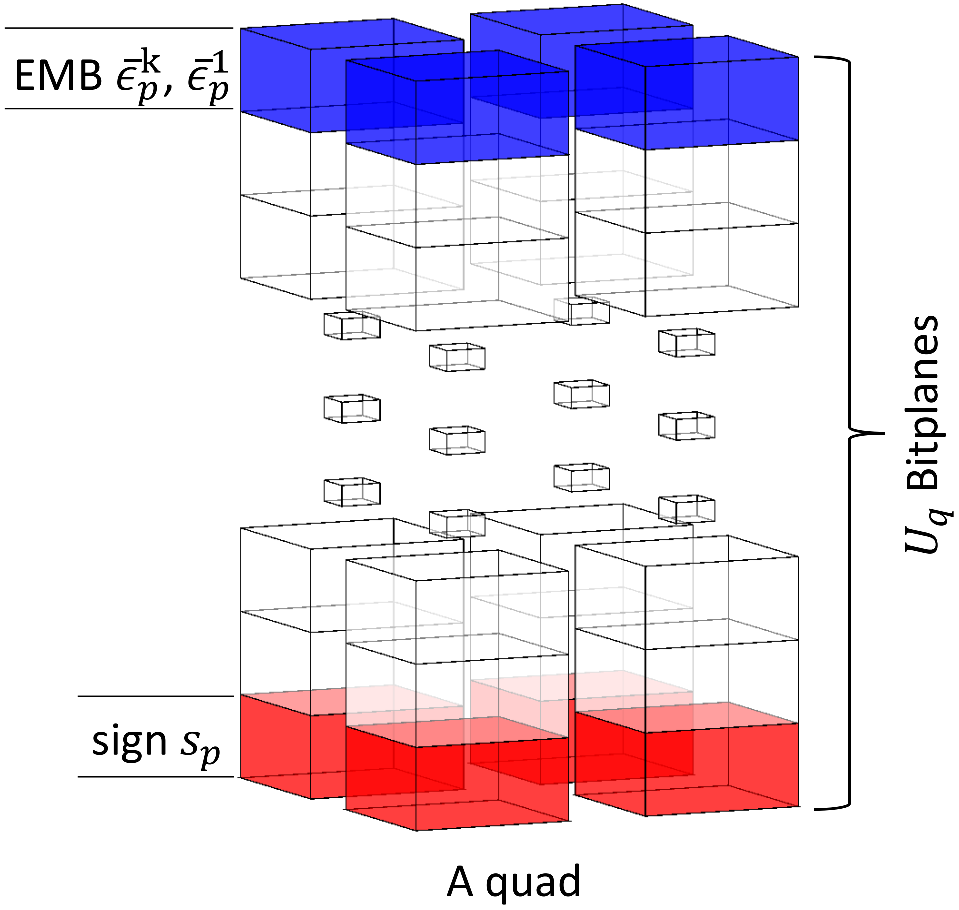


- We do **not** communicate $E_p^{\max}$
- We **indirectly** communicate an upper bound U_q, where $U_q \geq E_p^{\max}$
  - The idea is to try to predict U_q and increment it if it is not large enough
  - We generate a predictor κ_q from exponents E_q^{xx} in the previous row, then
  - If $\kappa_q \geq E_p^{\max}$, set $U_q = \kappa_q$, $u_q^{\mathrm{off}} = 0$, do not communicate u_q, else
  - If $\kappa_q < E_p^{\max}$, set $U_q = E_p^{\max}$, $u_q^{\mathrm{off}} = 1$, communicate $u_q = E_p^{\max} - \kappa_q$
  - We communicate u_q^{off} in the CxtVLC code, and u_q in the UVLC

The MgnSgn Segment (2/2)

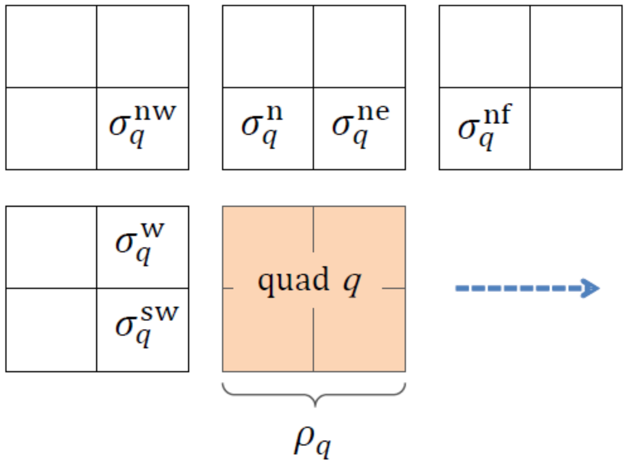
- U_q is the number of bits that need to be communicated.
- If $U_q > E_p^{\max}$, not ideal
  - MSBs are all zero.
  - CxtVLC communicate locations of zero samples
- If $U_q = E_p^{\max}$, ideal

- EMB $\bar{\epsilon}_q^{\mathrm{k}}, \bar{\epsilon}_q^1$ can communicate some MSBs.



The MEL Segment of HTJ2K

- An adaptive run-length coder that efficiently encodes runs of “0” events
- For a quad Q_q with zero context $c_q = 0 = \left(\sigma_q^{\mathrm{nw}} | \sigma_q^{\mathrm{n}}\right) + 2\left(\sigma_q^{\mathrm{w}} | \sigma_q^{\mathrm{sw}}\right) + 4\left(\sigma_q^{\mathrm{ne}} | \sigma_q^{\mathrm{nf}}\right)$
  - a “0” event means a quad with all zero coefficients.
  - A “1” event means one or more samples is not zero.
- Enables efficient coding of runs of all zero quads.



Recap:

- Notice that, for the VLC and MagSgn segments, producing data for a quad depends only on the adjacent quads → more opportunities for parallelism.
- Coding efficiency → efficiently coding locations of non-zero coefficients, using CxtVLC, and information about coefficient magnitude in the form of U_q.

GPU Implementation Overview

- HTJ2K is very suitable for GPU implementation.
  - The wavelet transform is highly parallelizable – 1 thread per 1 or 2 columns.
  - Many blocks per image – processed all in parallel. 4:4:4, 4K image has more than 6000 codeblocks
  - A codeblock bytestream has 3 segments – process one segment in one CUDA kernel
  - Segments, except MEL, are highly vectorizable – 1 CUDA thread per 1 or 2 columns
  - MEL can be efficiently implemented – needed for a small number of quads, produce very little data

- Usage scenarios:
  - CPU upload uncompressed images that are compressed on the GPU
    - requires high upload bandwidth on the PCIe interface – can limit the number of frames per sec, still 100s of frames per second on PCI 3.0.
    - This year's PCIe 4.0 interface supports more than 1000 4K frames/sec.
    - See Table 1 in the paper.
  - CPU upload HTJ2K images, which are decompressed, processed and compressed again on GPU.
    - requires lower per-frame bandwidth on the PCIe interface.
    - More frames can be processed per second if enough compute resources are available on the GPU.

CPU downloads compressed codeblocks, and packages them into files.

Colour & Wavelet Transform on a GPU

- The first wavelet decomposition employs a “special kernel” that
  - performs color transform
  - performs wavelet transform on 3 colour components in one kernel – this saved memory bandwidth
  - needs 113 registers
- Subsequent wavelet decomposition kernels operate on one component
  - Third dimension of the thread grid is used for components.
  - need 56 registers
- These Kernels produce 32-bit
  - Floats for wavelet coefficients – awaits further decomposition.
  - Integers in sign-magnitude format for quantized coefficient – awaits entropy coding
- Kernel properties
  - Each CUDA thread operates on 2 columns
  - Each kernel invocation operates on 64 rows – user configurable
  - We refer to these kernels by KCT+DWT

GPU kernel for the MagSgn segment

- This kernel is named KMagSgn; it
  - reads quantized samples
  - produces bit-stuffed MagSgn segment, storing it in Global memory
  - produces state info used by subsequent kernels; CxtVLC codewords, offsets u_q, and ρ_q
  - This is the only kernel that reads uncompressed data; this saves bandwidth.

- Kernel properties
  - Each CUDA thread processes 2 columns
  - Codeblocks wider than 64 are scanned by a single warp
  - For narrower codeblocks, one warp concurrently operates on multiple codeblocks
  - Uses one-byte shared memory for context c_q and 64 registers

UNSW SYDNEY

Other GPU Kernels

- KVLC kernel – for VLC segment
  - reads state information
  - generates bitstuffed VLC segment and store it in global memory
  - packs MEL events into a contiguous stream
  - one CUDA thread processes 2 quads, because of interleaving
  - uses 40 registers
- The KMEL kernel – for MEL segment
  - reads packed MEL events
  - produces bitstuffed MEL segment
  - one CUDA thread processes one codeblock, because of the serial nature of MEL coding
  - uses 30 registers
- The KVCPY kernel – for VLC segment
  - copies the VLC segment to the end of the MEL segments, potentially overlapping the two segments
  - produces the pointer at then end of the VLC segment
  - uses 26 kernels
- No mechanism yet to detect zero codeblocks (non significant samples)

UNSW SYDNEY

Experimental Results

- We tested with the 3 GPUs shown next

Card	CUDA Cores	Boost Clock (MHz)	Mem. BW (GB/s)	Attainable Mem. BW (GB/s)	PCIe 3.0 Lanes	Compute Capability
GT1030	384	1468	48	~40	x4	6.1
GTX1660Ti	1536	1845	288	~240	x16	7.5
GTX1080	2560	1847	320	~240	x16	6.1

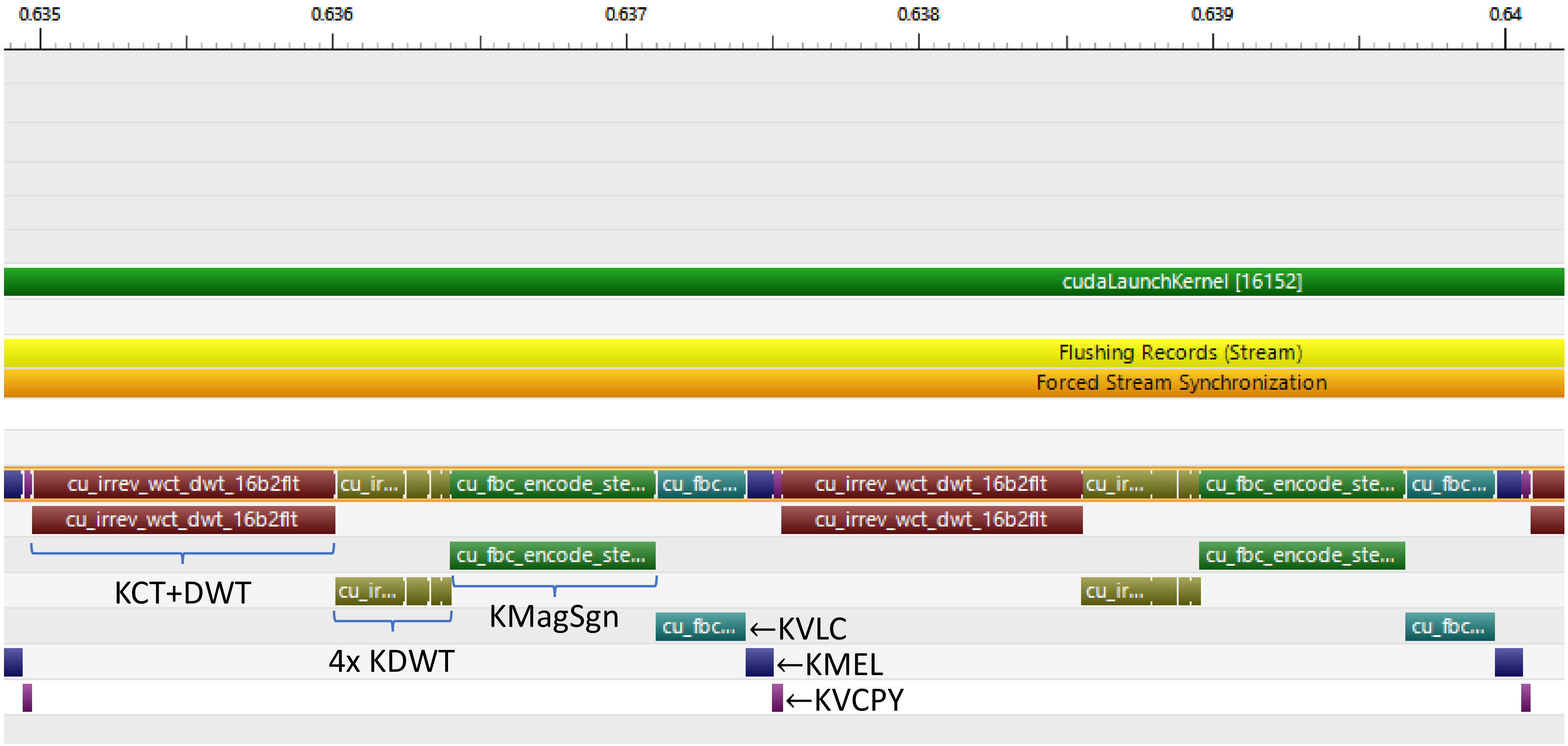
- The next page shows kernel times and coding performance for encoding 4K 4:4:4 12bit video test sequence “ARRI_AlexaDrums”

UNSW SYDNEY

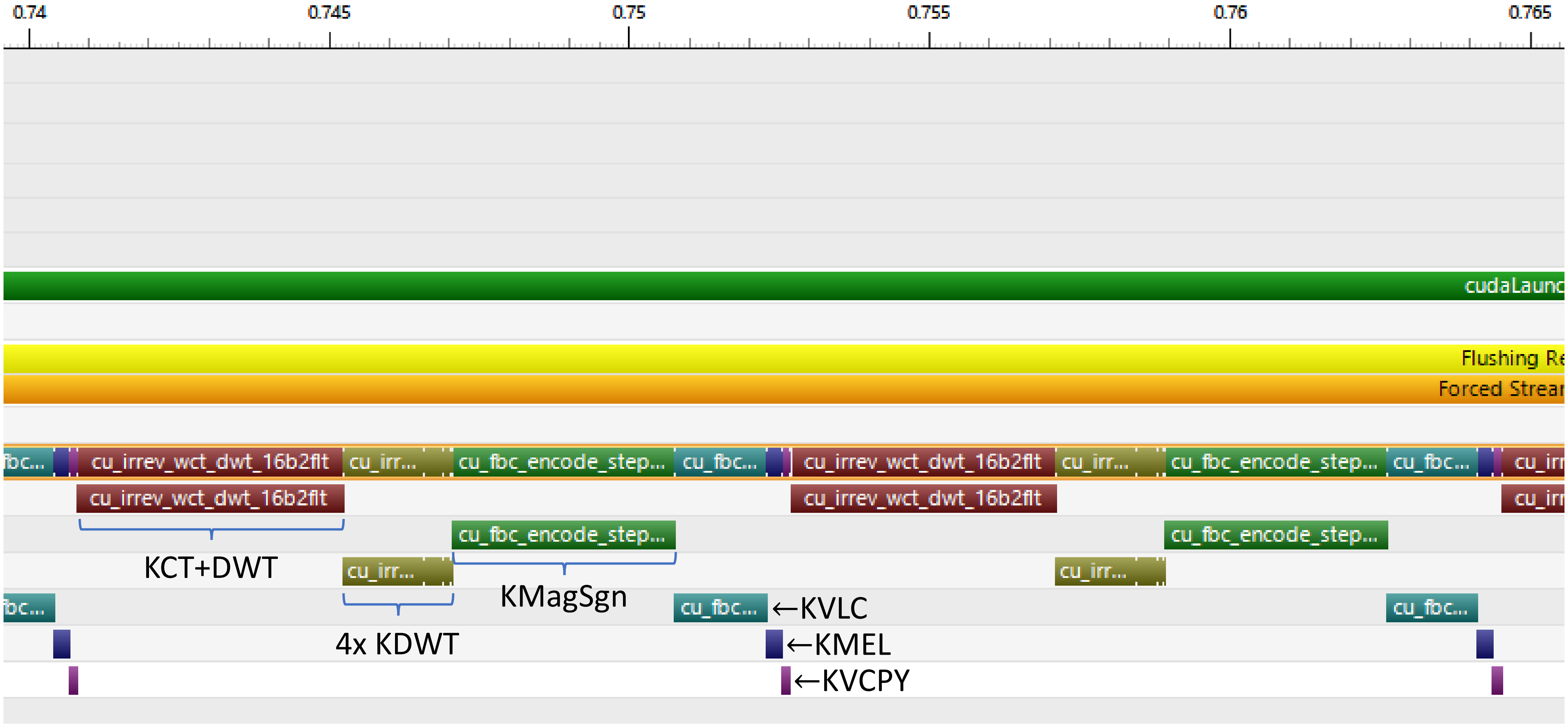
Kernel	GT1030 1bit/pixel	GT1030 lossless	GTX1660Ti 1bit/pixel	GTX1660Ti lossless	GTX1080 1bit/pixel	GTX1080 lossless
	KCT+DWT time to decompose one frame (ms)					
KCT+DWT	6.233	6.233	1.410	1.410	1.304	1.304
	Time to encode one frame (ms) using **64x64** codeblocks					
KMagSgn	3.243	4.338	0.698	1.089	0.551	0.647
KVLC	1.105	1.432	0.307	0.381	0.195	0.224
KMEL	0.275	0.303	0.092	0.026	0.102	0.026
KVCPY	0.115	0.096	0.028	0.079	0.022	0.076
Frames per second	90	80	391	332	455	435
	Time to encode one frame (ms) using **32x32** codeblocks					
KMagSgn	3.263	4.350	0.794	2.013	0.576	0.815
KVLC	1.434	1.530	0.377	0.630	0.374	0.463
KMEL	0.496	0.366	0.107	0.093	0.125	0.100
KVCPY	0.370	0.568	0.077	0.129	0.064	0.126
Frames per second	84	76	358	230	405	353

	Frames per second		
JPEG2K [7]	NA	NA	40†

Timeline plot for GTX 1660Ti at 1bpp



Timeline plot for GT1030 at 1bit/pixel



Conclusions and Future work

- HTJ2K standard is an exciting new addition to JPEG2000
- HTJ2K has significantly lower complexity and enables fast and parallelisable implementations – an order of magnitude
- Block coding is very fast – similar complexity to colour/wavelet transforms
    - Rate-control can be very fast – two cleanup passes, with SPP & MRP, are sufficient
- HTJ2K is transcodable to and from conventional JP2000 (Parts 1 and 2)
- Very fast GPU implementation is possible, encoding 100s of frames per sec.

Future work includes

- the addition of SPP and MRP
- the implementation of rate-control
- Publishing complete results for encoder/decoder implementation

Thank you!