cal@di

Implementation and Evaluation of DWT on Contemporary NVIDIA GPUs and x86 CPUs

MoMA LAB | NEW YORK UNIVERSITY ABU DHABI

Vasilis Dimitsas[1], Olympia Kremmyda[1], Dimitris Gizopoulos[1], Anastasis Keliris[2], Michail Maniatakos[3]
[1] University of Athens, [2] New York University Polytechnic School of Engineering, [3] New York University Abu Dhabi

DWT (Discrete Wavelet Transform)

DWT is a wavelet transformation method in which the pixels of an image, as wavelets, are discretely sampled. DWT is used in many research and industrial areas:

- image compression algorithms such as JPEG 2000 and CCSDS-122.0-B-1
- biomedical applications for edge detection
- digital watermarking

DWT performs a three-level, two-dimensional (2D) wavelet transformation and splits the image into ten subareas (sub-bands)

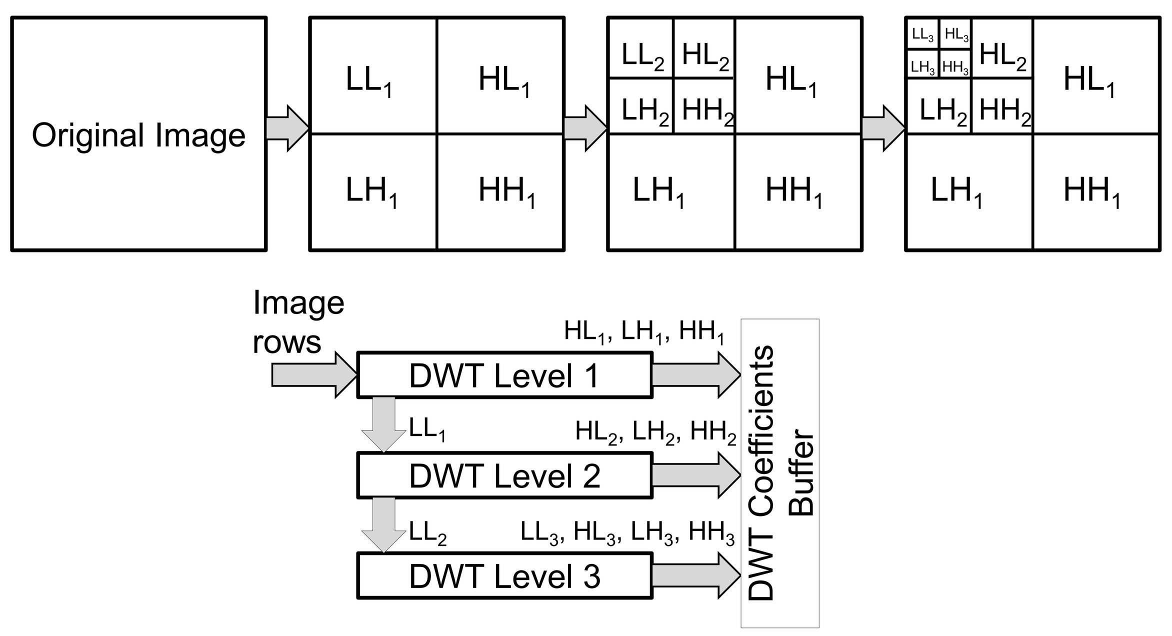

Fig. 1: DWT separates the image in 10 sub-bands (upper) and operates in three levels (lower).

The image pixels are filtered into two sets of integer wavelet coefficients: set *D* which represents the high frequency of a pixel and set *C* which represents the low frequency.

$$D_0 = x_1 - \left\lfloor \frac{9}{16}(x_0 + x_2) - \frac{1}{16}(x_2 + x_4) + \frac{1}{2} \right\rfloor$$

$$D_j = x_{2j+1} - \left\lfloor \frac{9}{16}(x_{2j} + x_{2j+2}) - \frac{1}{16}(x_{2j-2} + x_{2j+4}) + \frac{1}{2} \right\rfloor$$

$$D_{N-2} = x_{2N-3} - \left\lfloor \frac{9}{16}(x_{2N-4} + x_{2N-2}) - \frac{1}{16}(x_{2N-6} + x_{2N-2}) + \frac{1}{2} \right\rfloor$$

$$D_{N-1} = x_{2N-1} - \left\lfloor \frac{9}{8} x_{2N-2} - \frac{1}{8} x_{2N-4} + \frac{1}{2} \right\rfloor$$

$$C_0 = x_0 - \left\lfloor - \frac{D_0}{2} + \frac{1}{2} \right\rfloor$$

$$C_j = x_{2j} - \left\lfloor - \frac{D_{j-1} D_j}{4} + \frac{1}{2} \right\rfloor$$

CUDA Implementation

- Transfer data from global to shared memory into blocks (Every block of threads copies the corresponding chunk of input data into its shared memory):

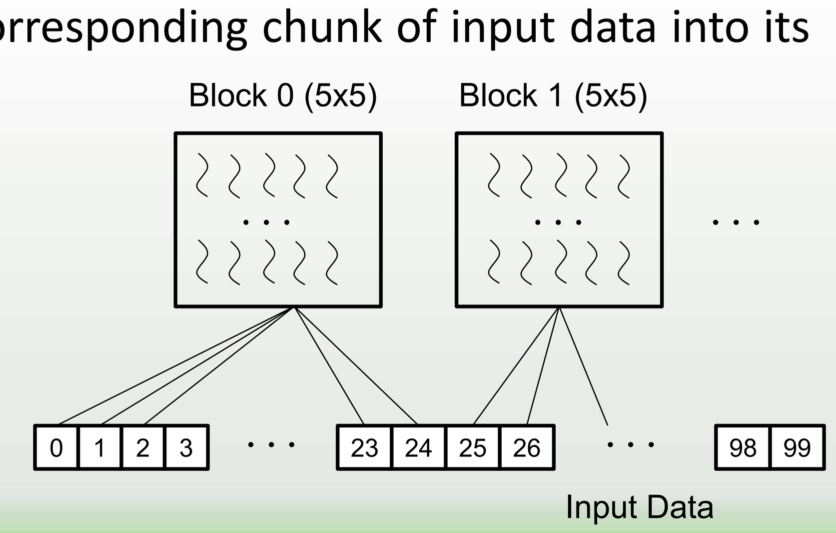

Fig. 2: Assignment of blocks to the image input array.

- Extend the boundary pixels (To eliminate time consuming calculations for the pixels at the boundary of the image which require values for pixels "out" of the image)

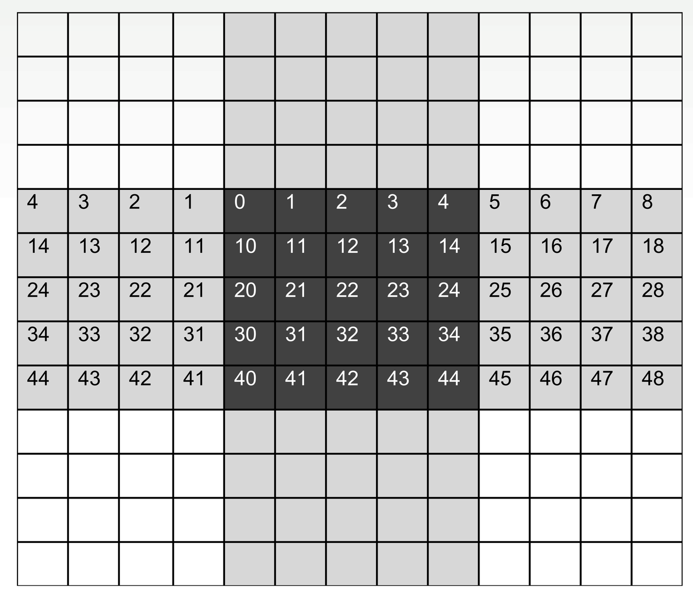

Fig. 3: Shared memory layout after the transfer and the extension of the chunk of data of block 0 (for the row processing part of DWT). The numbers in the cells correspond to the original location of pixels into the input array.

- Calculation: Each thread t_i focuses on a central pixel x_{2i} and uses six more pixels (x_{2i-4}, x_{2i-2}, x_{2i-1} and x_{2i+1}, x_{2i+2}, x_{2i+4}) for the calculation of the corresponding coefficients.

- DWT operation includes two steps:
  - row transform
  - column transform

- For column transform the same steps are followed except that the calculations are done column-wise.
  - shared memory is used in order to efficiently coalesce global memory accesses, while processing the image in columns.

Results

The experiments of the DWT were conducted in three different systems.

The measurements focus on the raw processing time of each hardware component i.e. in case of GPU we measure the kernel execution time and for CPUs we take into consideration the DWT processing time.

	System 1	System 2	System 3
CPU	AMD Phenom™ II X4 965 4 cores, @ 3.4GHz, 45nm (Released **November 2009**) 8 GB main memory	Intel® Core™ i7-3970X 6 cores, @ 3.50GHz, 32nm (Released **November 2012**) 32 GB main memory	Intel® Xeon® E5-2680 8 cores @ 2.7GHz, 32nm (Released **March 2012)** 32 GB main memory
GPU/Coprocessor	NVIDIA Tesla C2070 (Fermi architecture), 6GB GDDR5 RAM, 448 CUDA cores, 40nm (Released **July 2010**)	NVIDIA Tesla K20 (Kepler architecture), 5GB GDDR5 RAM, 2496 CUDA cores, 28nm (Released **November 2012**)	Intel® Xeon Phi™ Coprocessor 5110P (8GB Memory, 60 cores @ 1.053 GHz) 22nm (Released **November 2012)**

TABLE 1: CONFIGURATIONS OF THE THREE EVALUATION SYSTEMS AND THEIR RELEASE DATES

To optimize the program on multicore CPUs we first analyze and optimize the serial execution of the program and then identify sections that can be executed in parallel. For the parallel execution we investigate performance gains using both OpenMP and CilkPlus.
The results in Table 2 below represent the best cases of the DWT algorithm execution across the various platforms.

	256 x 256	512 x 512	1024 x 1024	2048 x 2048	4096 x 4096	8192 x 8192
Phenom	1.673	6.88	32.038	144.38	577.239	2586.208
i7	1.1375	3.7565	20.6411	72.3541	270.02	951.019
E5	1.0213	3.4224	14.4339	54.9691	234.315	953.823
Fermi	0.19	0.5	1.19	3.88	14.58	58.35
Kepler	0.19	0.45	1.03	3.14	11.66	45.88
Xeon Phi	6.63	43.92	235.5	749.17	1806.4	4128.95

TABLE 2: THE EXECUTION TIMES (IN MS) OF THE DWT ALGORITHM IN EVERY PROCESSING COMPONENT.

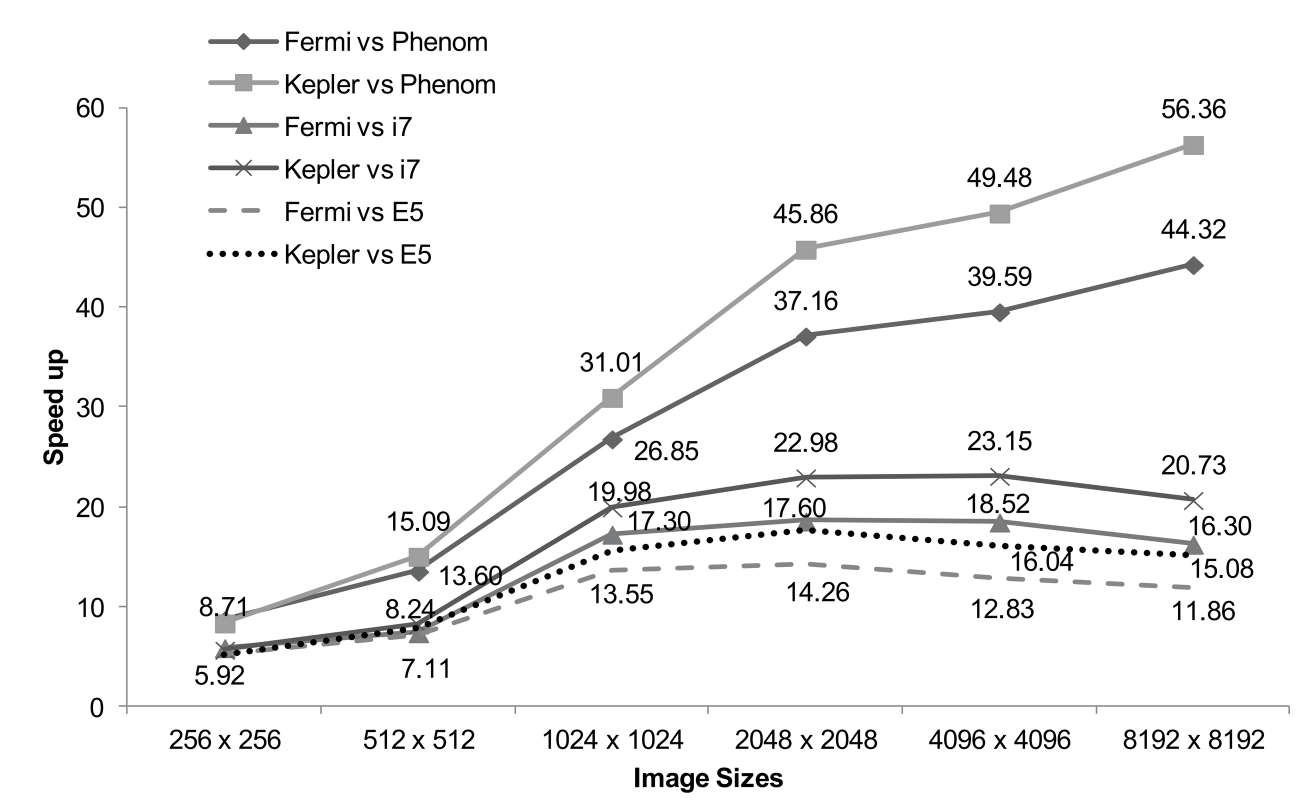

Fig. 4: The speedups of the GPUs against the CPUs for the DWT algorithm execution (CPU/GPU execution times ratio).

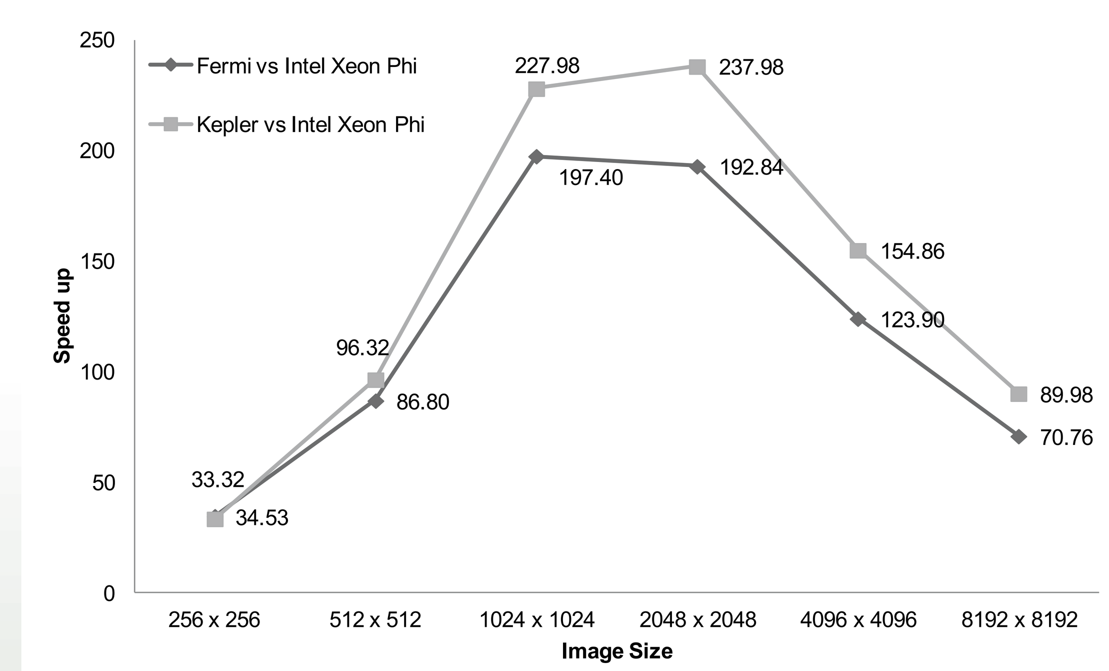

Fig. 5: GPUs vs. Intel Xeon Phi overall speedup diagram (Phi/GPU execution times ratio).

References

1. Image Data Compression CCSDS-120.1-G-1. 2007. http://public.ccsds.org/publications/archive/120x1g1e2.pdf
2. CCSDS Image Data Compression Implementation. Nebraska – Lincoln University. http://hyperspectral.unl.edu/index.html
3. Sanders J., E.Kandrot, CUDA by Example – An Introduction to General-Purpose GPU Programming. Addison-Wesley, 2010.
4. NVIDIA CUDA HTTP://WWW.NVIDIA.COM/OBJECT/CUDA_HOME_NEW.HTML