Advantages of High-Level Synthesis in an OpenCL Based FPGA Programming Methodology

Alex Bartzas, George Economakos and Dimitrios Soudris
Microprocessors and Digital Systems Laboratory,
National Technical University of Athens, Greece

HLS4HPC Workshop @ HiPEAC 2013

Outline

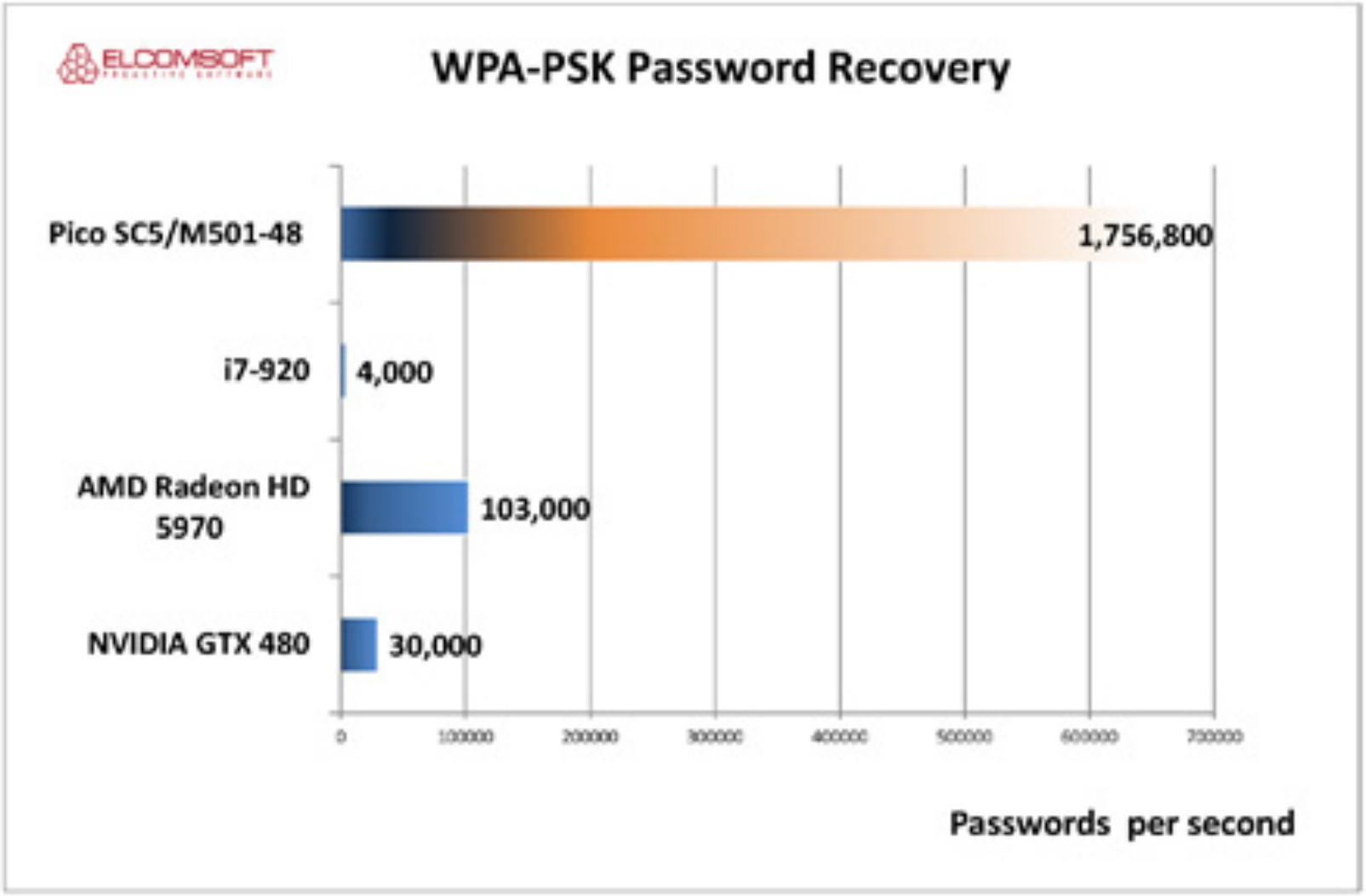
- Motivation
- Methodology
- Experimental results
- Conclusions and future work

Motivation – FPGAs in Parallel Programming

Press Release, Moscow, Russia – July 17, 2012 - ElcomSoft Co. Ltd. releases world's fastest password cracking solutions by supporting Pico's range of high-end hardware acceleration platforms. ElcomSoft updates its range of password recovery tools, employing Pico FPGA-based hardware to greatly accelerate the recovery of passwords.

At this time, two products received the update: Elcomsoft Phone Password Breaker and Elcomsoft Wireless Security Auditor. Users of these products can now recover Wi-Fi WPA/WPA2 passwords as well as passwords protecting Apple and Blackberry offline backups even faster than with the already supported clusters of high-end video accelerators produced by AMD and NVIDIA. Pico support is planned for Elcomsoft Distributed Password Recovery.

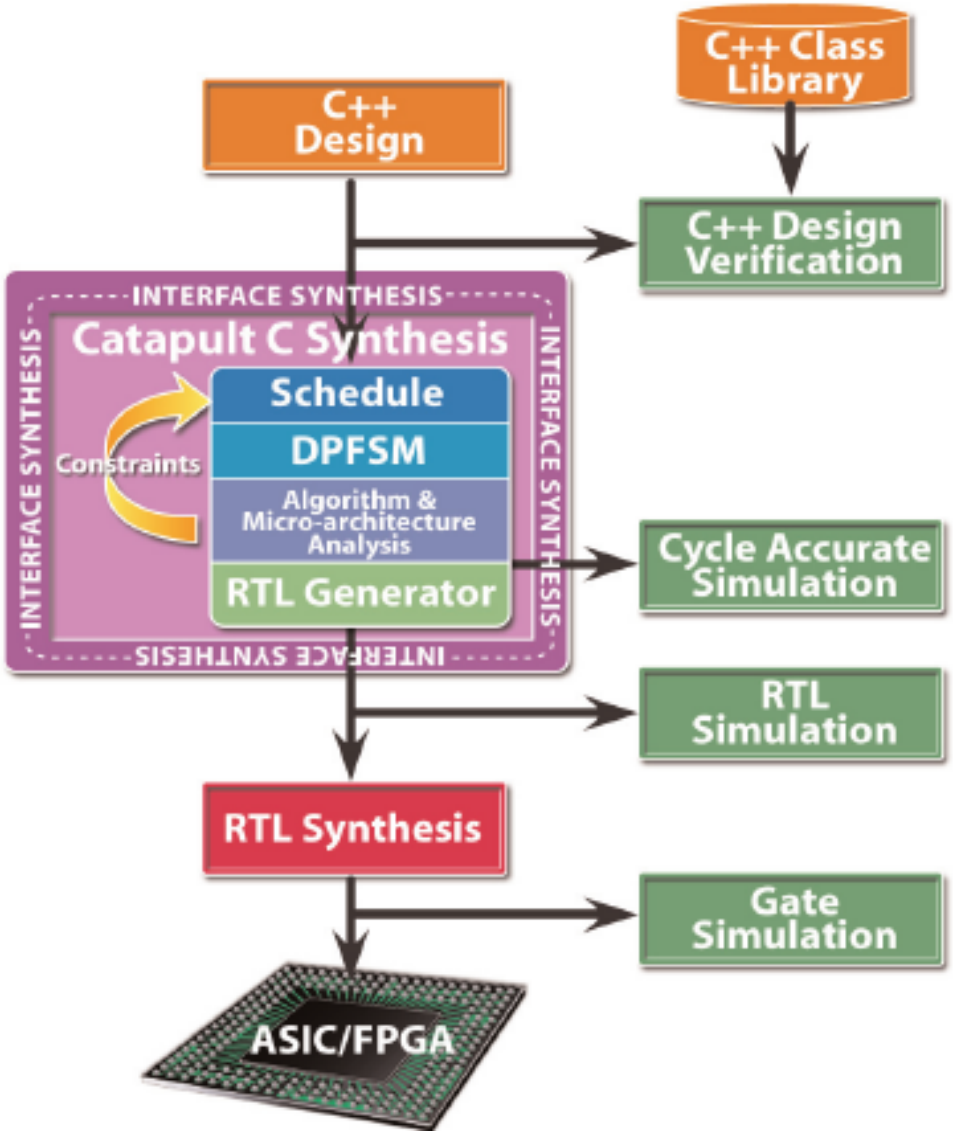
Motivation – FPGAs in Parallel Programming



Motivation – OpenCL Adoption

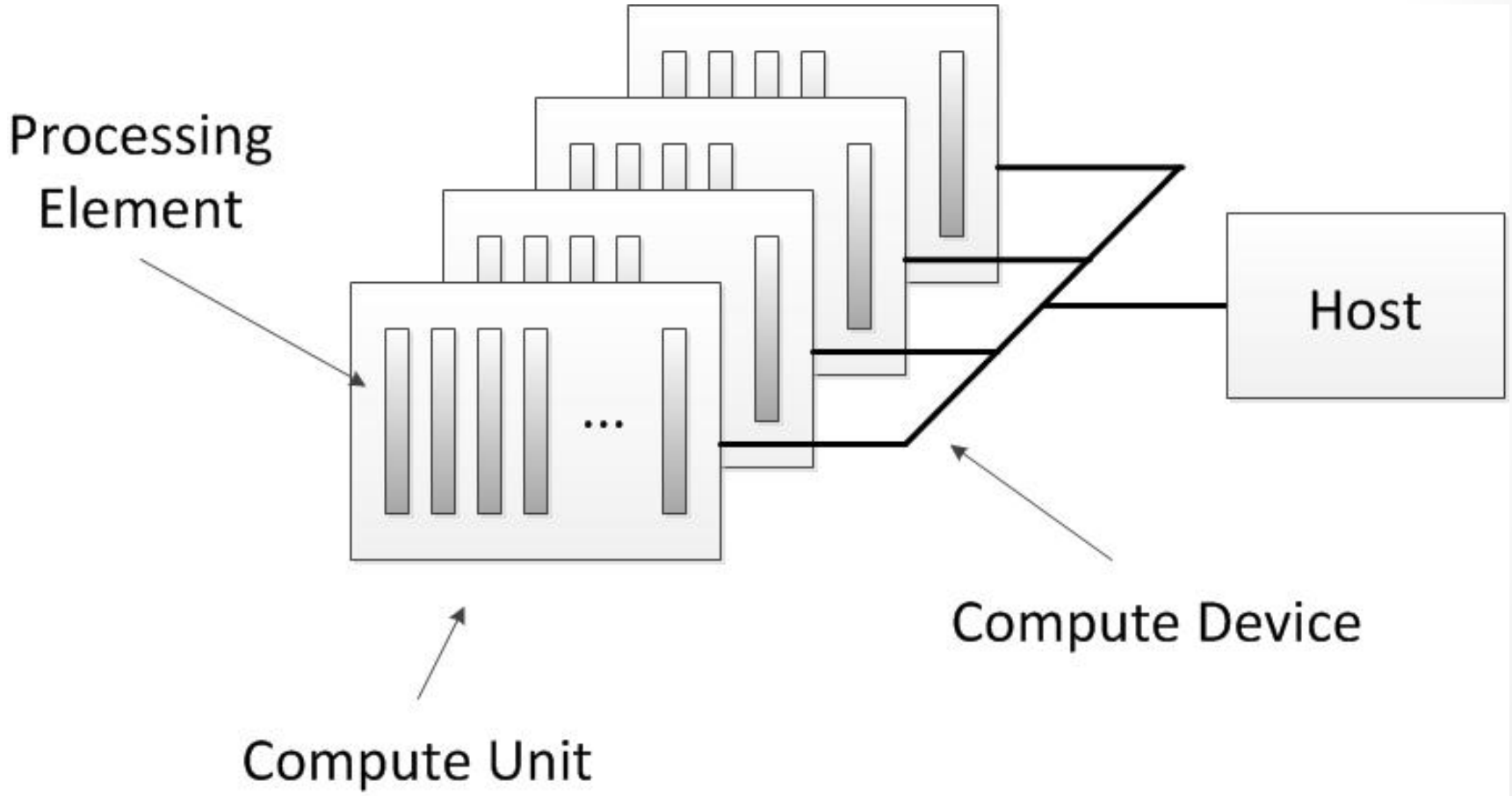
	Intel CPUs	AMD CPUs	NVIDIA Tesla GPUs	AMD GPUs	IBM Power Systems	Altera/ Xilinx FPGAs
C/C++	Yes	Yes	No	No	Yes	No
OpenGL SL	No	No	Yes/No	Yes	No	No
OpenCL	Yes	Yes	Yes	Yes	Yes	TBD
Intel TBB	Yes	Yes	No	No	No	No
CUDA	No	No	Yes	No	No	No

Motivation – ESL & HLS

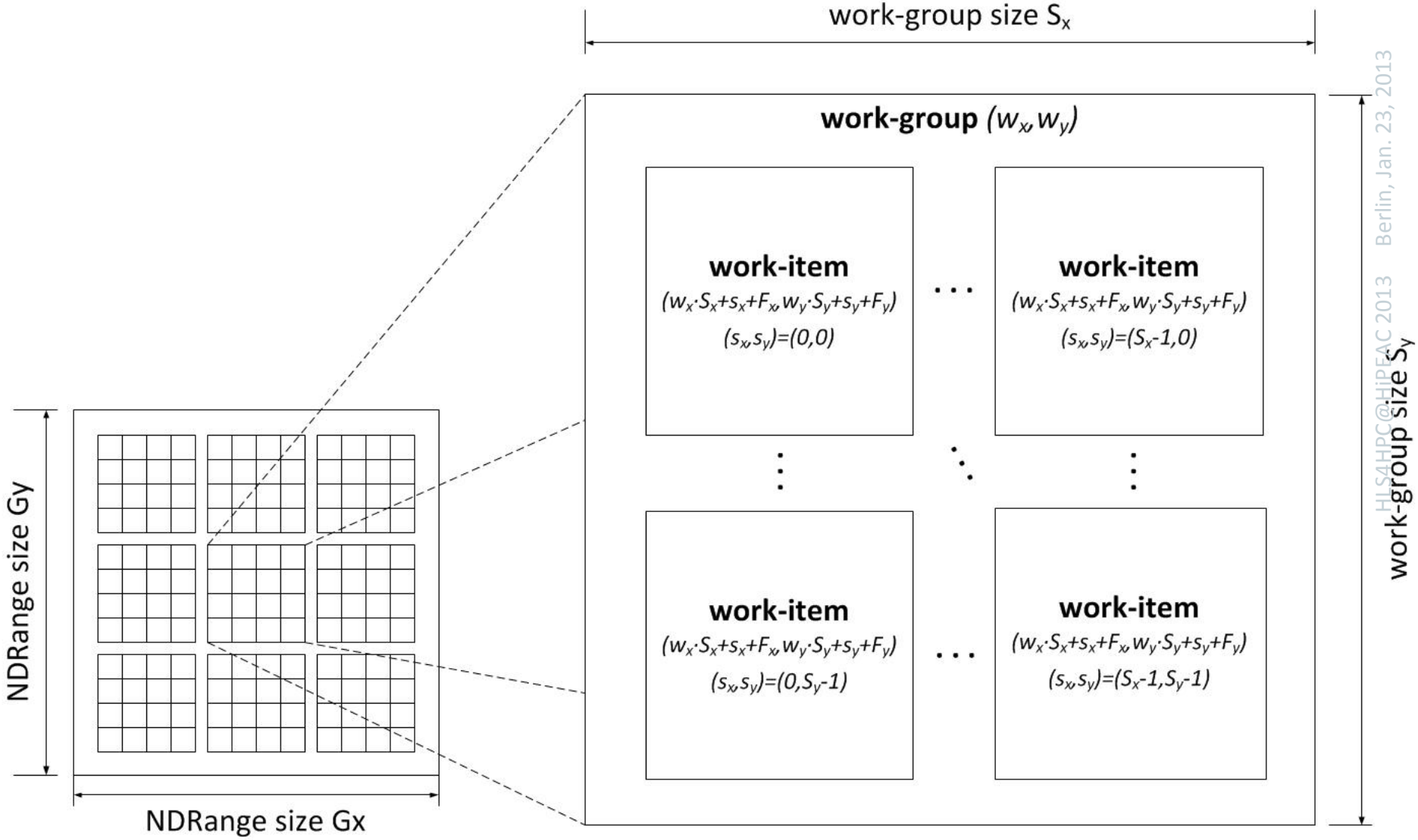


Source: Calypto Design Systems

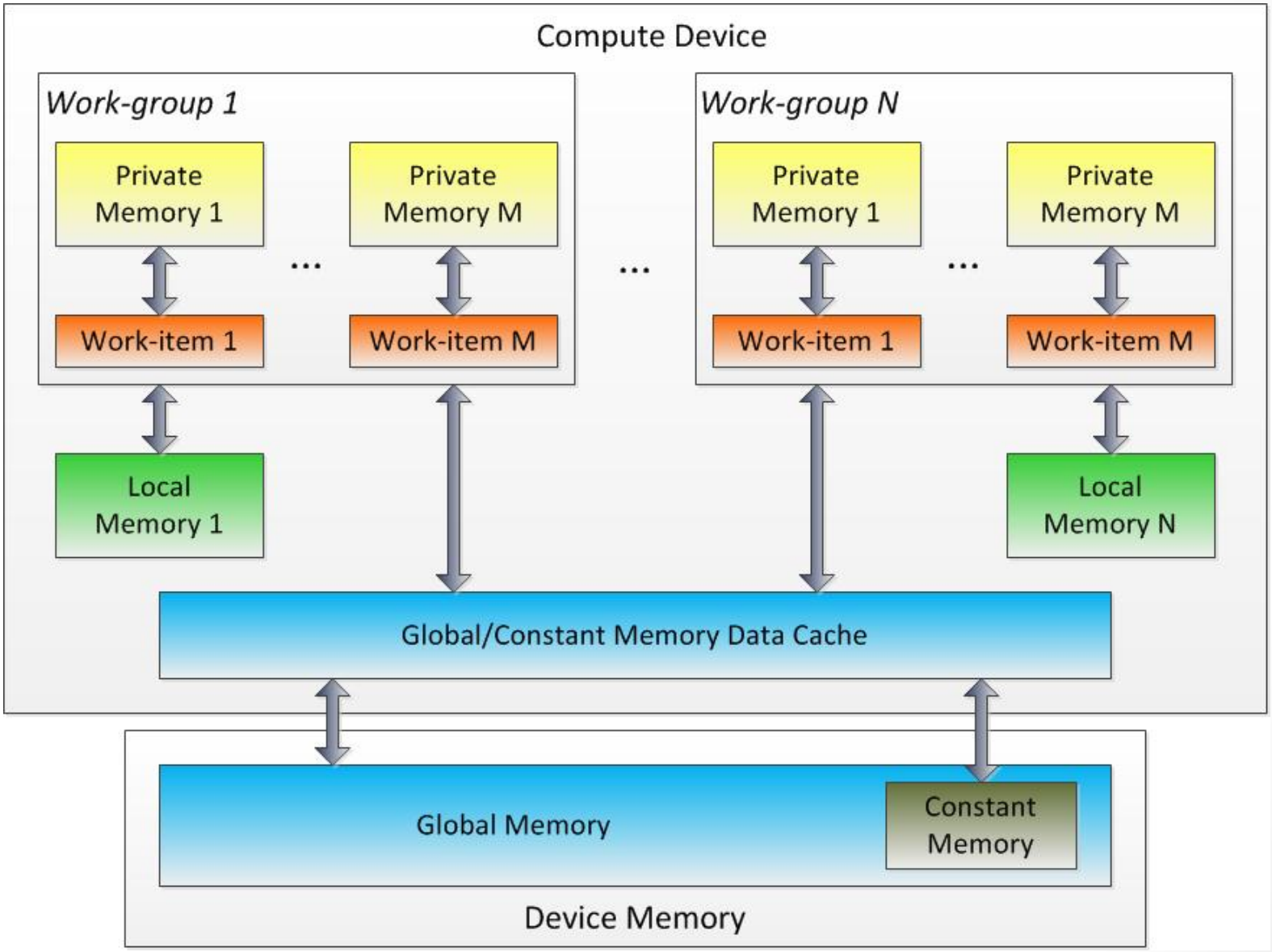
OpenCL Platform Model



OpenCL Execution Model



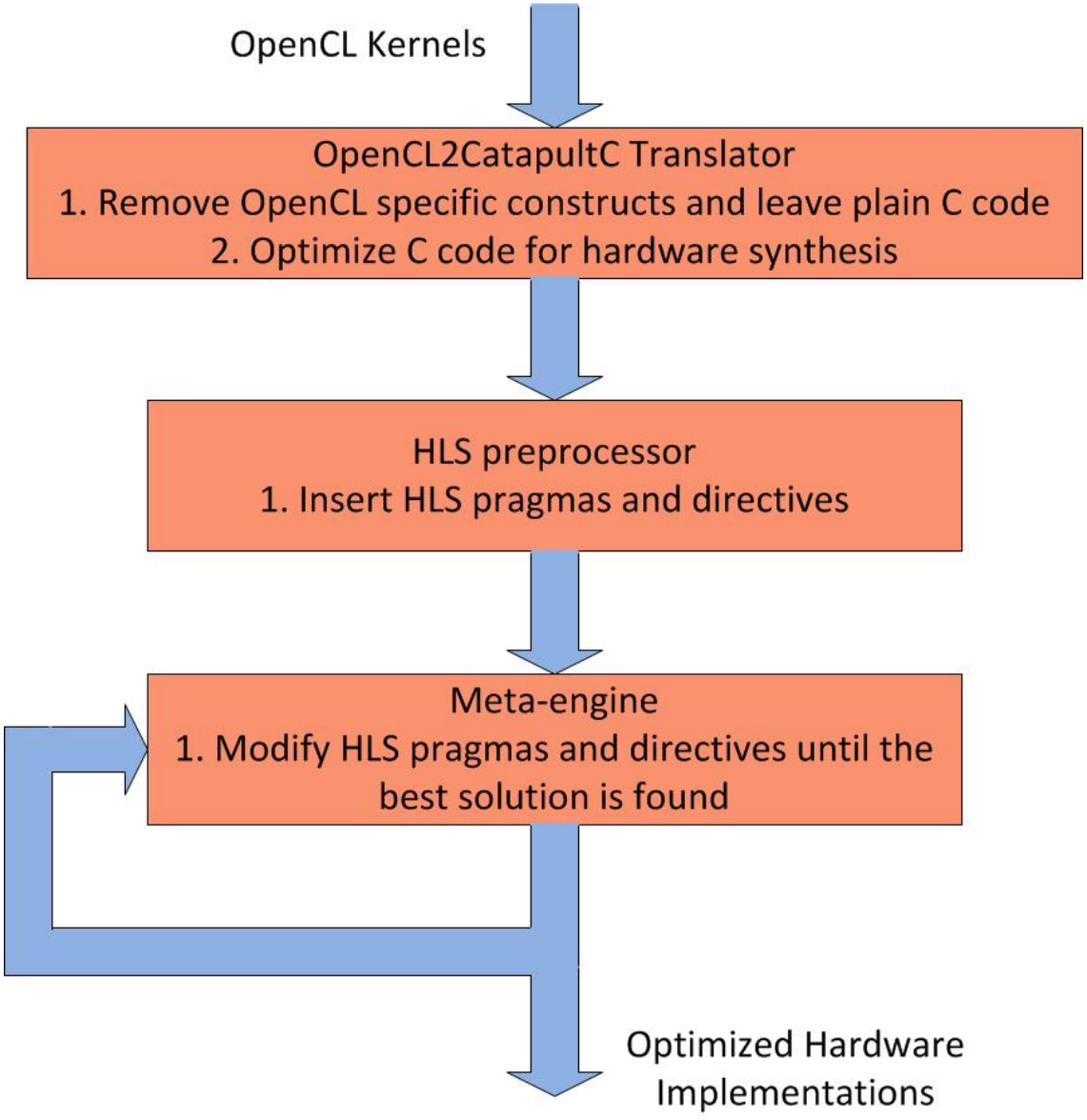
OpenCL Memory Model



Difference with Related Approaches

- Other related approaches are template based, i.e. they recognize OpenCL constructs and map them into HDL code previously filled into corresponding templates
  - Jaaskelainen, de La Lama, Huerta and Takala, "OpenCL-based Design Methodology for Application-Specific Processors"
  - Mingjie, Lebedev and Wawrzynek, "OpenRCL: Low-Power High-Performance Computing with Reconfigurable Devices"
  - Owaida, Bellas, Antonopoulos, Daloukas and Antoniadis, "Massively Parallel Programming Models Used as Hardware Description Languages: The OpenCL Case"
  - http://www.altera.com/opencl
- The proposed work is synthesis based, searching for different microarchitectural styles and generating application specific kernels through HLS
- The same difference is found between IP based design and HLS in ESL environments.

Proposed Methodology



Proposed Methodology Steps

1. Translate OpenCL kernels into CatapultC ready code.
2. Iteratively apply HLS transformations (exhaustive application/exploration) to find the best FPGA based implementation (meta-engine), with respect to performance and area consumption.
3. Manually transform host OpenCL code into an FPGA based controller, to control kernel deployment (number of kernels and memory architecture), invocation (parameter passing) and synchronization, on selected FPGA devices.

Work-in-Progress Steps

1. Apply heuristics to the meta-engine for run time efficiency.
2. Consider FPGA based power consumption.
3. Automate the transformation of the host code into either small scale hardware controllers or OpenCL code for an embedded processor.

Translation Methodology

- Each kernel is isolated and HLS synthesizes a hardware component for it.
- Pointers used as formal parameters in functions are converted to arrays with specific dimensions, for correct memory allocation.
- Return values are inserted as formal pointer parameters in the kernel function. This coding technique generates output registers for them.
- Barrier OpenCL instructions are converted into CatapultC I/O transactions with ready/acknowledge interfaces.
- Array sizes are enlarged to reach powers of 2, when feasible. This simplifies synthesis of memory access related hardware.

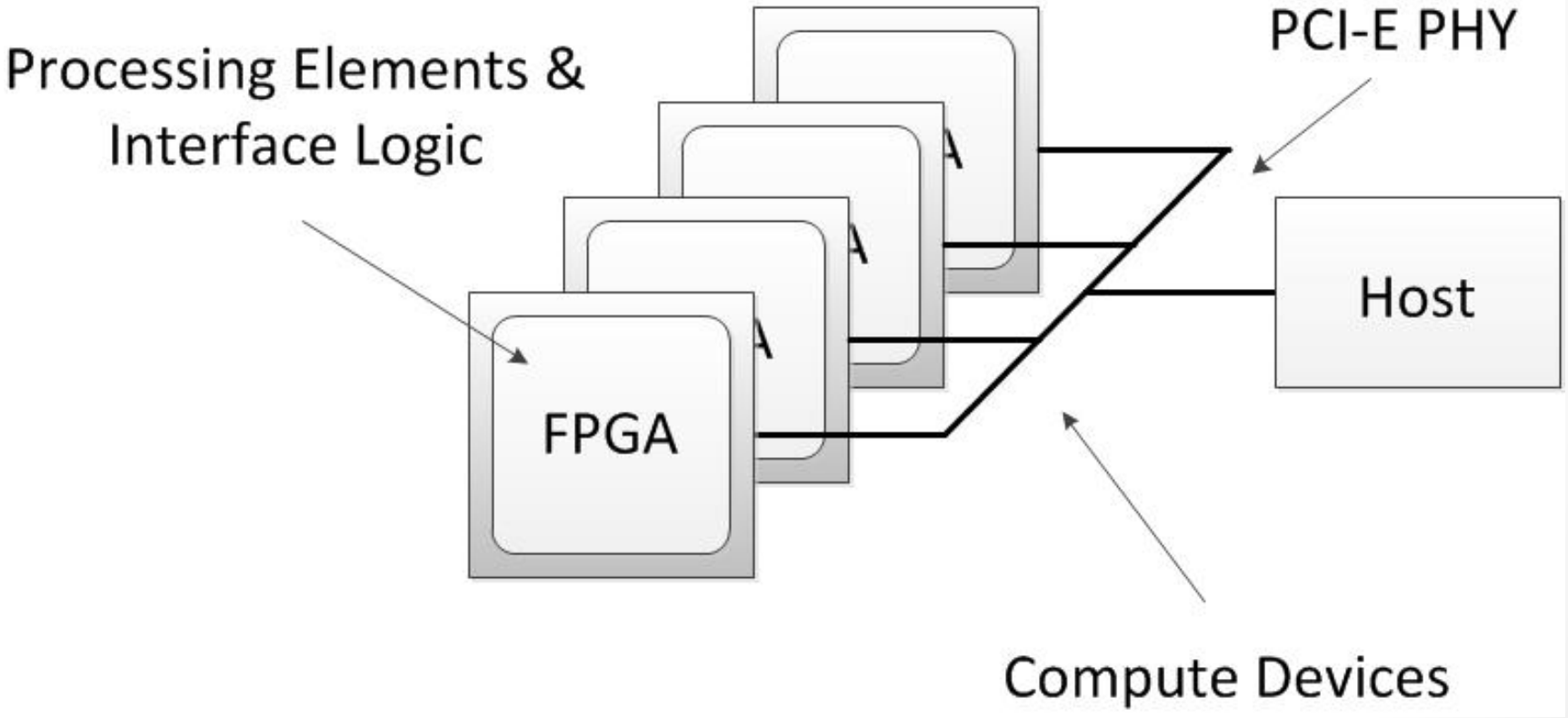
Translation Methodology

- Data types are changed into bit accurate and simulation efficient types supported by CatapultC.
  - For example, integer data types can be changed into ac_int<16,false> (16 bit unsigned integer).
- Conditional statements are supplemented so that all mutually exclusive paths are clearly defined.
  - For example, if statements are supplemented with else clauses when possible. This helps {CatapultC} schedule them correctly.
- OpenCL specific directives are temporary removed. They are taken into account later, during system integration.
- CatapultC pragmas and directives are inserted. These pragmas and directives control all HLS transformations, acting as either on-off switches (the corresponding transformation is performed only if the directive is present) or value holding elements (the corresponding transformation is performed with respect to the given value).

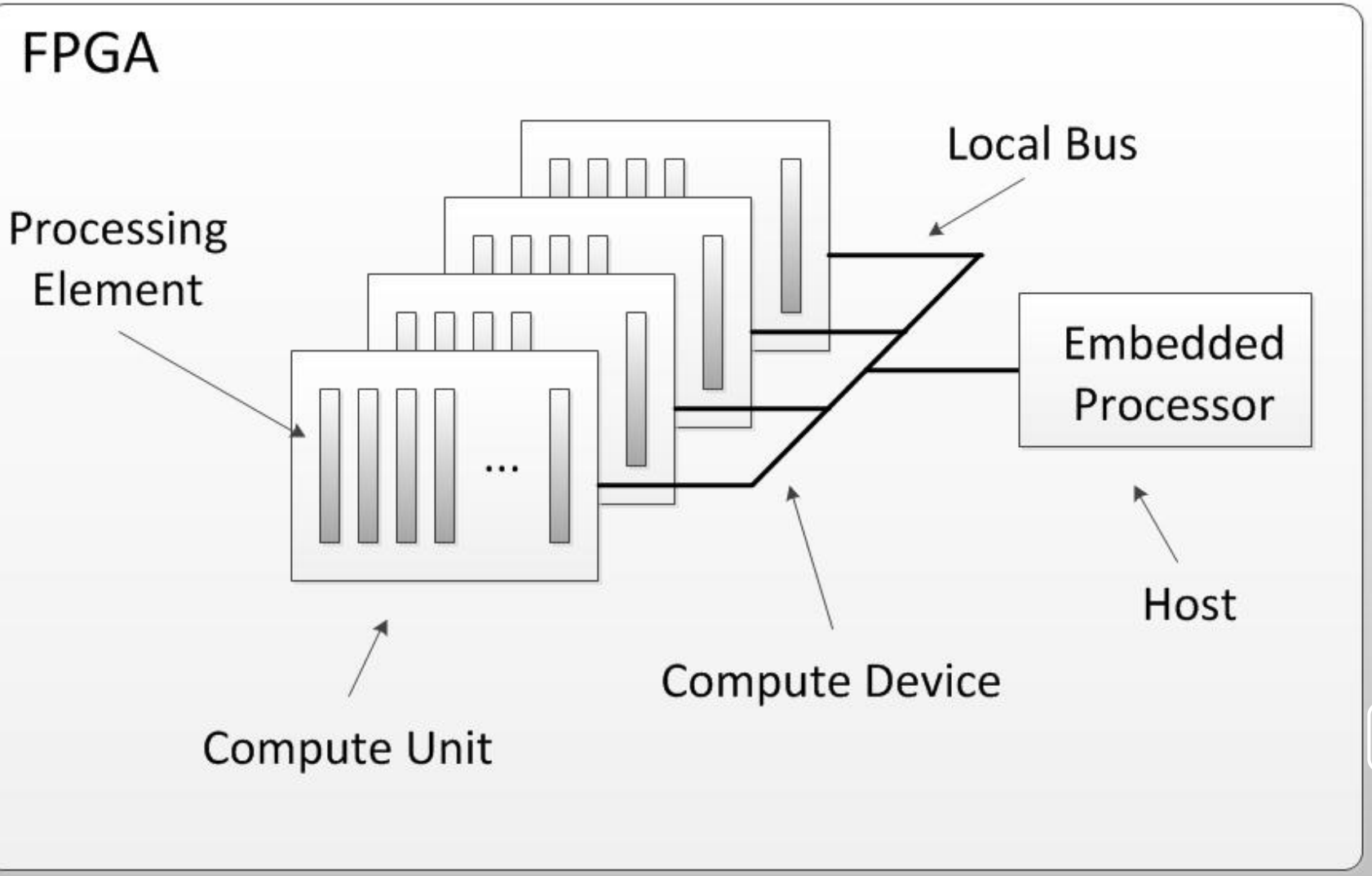
HLS optimizations

- Loops
  - Pipelining
  - Unrolling
  - Merging
- Memories
  - Register files
  - On-chip memories
  - Off-chip memories
  - Single or dual port
  - Interleaved blocks
- Synchronization
  - Barriers changed into I/O ready/acknowledge signals

System Integration



System Integration



Experimental results

Parallel Matrix Multiplication

Solution	Performance (throughput ns)	LUTs	DFFs	BRAMs	DSPs
S1	1295	85(0.02%)	102(0.01%)	0(0.00%)	4(0.46%)
S2	640	84(0.02%)	102(0.01%)	0(0.00%)	4(0.46%)
S3	320	113(0.02%)	118(0.01%)	0(0.00%)	8(0.93%)
S4	160	213(0.04%)	191(0.02%)	0(0.00%)	16(1.85%)
S5	80	335(0.07%)	292(0.03%)	0(0.00%)	32(3.70%)

S1 corresponds to no optimizations selected. Solution S2 corresponds to initiation interval set to 1, while solutions S3, S4 and S5 keep this value and add an unrolling factor of 2, 4 and 8 respectively.

Experimental results

Parallel Discrete Cosine Transform

Solution	Performance (throughput ns)	LUTs	DFFs	BRAMs	DSPs
S1	455	4158(0.88%)	1702(0.18%)	1(0.14%)	37(4.28%)
S2	640	4194(0.88%)	2084(0.22%)	1(0.14%)	48(5.56%)
S3	110	3563(0.75%)	2354(0.25%)	1(0.14%)	23(2.66%)
S4	30	3602(0.76%)	2377(0.25%)	1(0.14%)	68(7.87%)
S5	30	3649(0.77%)	2261(0.24%)	0(0.00%)	46(5.32%)
S6	15	5273(1.11%)	4339(0.46%)	0(0.00%)	62(7.18%)
S7	10	5453(1.15%)	6292(0.66%)	0(0.00%)	64(7.41%)

Experimental results

Parallel Inverse Discrete Cosine Transform

Solution	Performance (throughput ns)	LUTs	DFFs	BRAMs	DSPs
S1	450	3002(0.63%)	1688(0.18%)	1(0.14%)	38(4.40%)
S2	800	4703(0.99%)	2001(0.21%)	1(0.14%)	52(6.02%)
S3	70	3331(0.70%)	1859(0.20%)	1(0.14%)	34(3.94%)
S4	35	2499(0.53%)	1521(0.16%)	1(0.14%)	54(6.25%)
S5	35	2489(0.52%)	1519(0.16%)	0(0.00%)	54(6.25%)
S6	15	5329(1.12%)	4259(0.45%)	0(0.00%)	56(6.48%)
S7	10	5498(1.16%)	5491(0.58%)	0(0.00%)	56(6.48%)

Experimental results

FPGA and GPU comparison

Xilinx Virtex-6 6VLX760 at 600MHz vs Radeon HD 6970 GPU at 850MHz

	Execution time (ns)			
Platform	256x256	512x512	1024x1024	2048x2048
Virtex-6 (S1)	662102	1216167	2324299	4540563
Virtex-6 (S6)	399822	772103	1510840	2988349
Radeon	755398	1225752	2958031	10160484
Speedup:	1.8	1.5	1.9	3.4

Conclusions and future work

- Methodology for the adoption of OpenCL as an FPGA programming environment, based on the systematic application of HLS transformations by a meta-engine.
  - Even though HLS tools can produce hardware from C, efficient hardware needs effort and some architectural synthesis expertise.
  - This expertise is captured in the meta-engine, which iterates through different possible and feasible directive applications, and generates optimal hardware implementations.
- Use of both CUDA and OpenCL under the same environment
- Use of heuristics in the meta-engine iterations, to speed up the process and produce better results

Thank you!
Questions?

More info:
Alex Bartzas
alexis@microlab.ntua.gr