Workshop:
Programming FPGAs with OpenCL

David Castells-Rufas
Microelectronics & Electronics Systems Department
Universitat Autònoma de Barcelona
david.castells@uab.cat

Centre de Prototips i Solucions Hardware - Software
Center for Hardware – Software Prototypes & Solutions

What is CEPHIS?

Cephis

- Center for Hardware / Software Prototypes

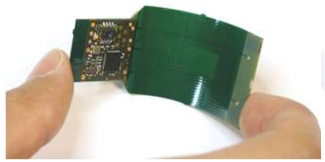
- Focussed on Technology Transfer
- Part of the TECNIO network
  http://www.accio.gencat.cat/ca/serveis/innovacio/tecnologia-per-a-lempresa/tecnio/

Cephis

- Flexible electronics
  - Research on Physically & Functionally flexible embedded systems

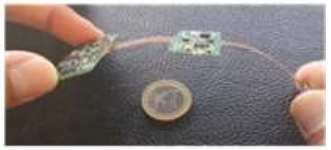
Exemples:

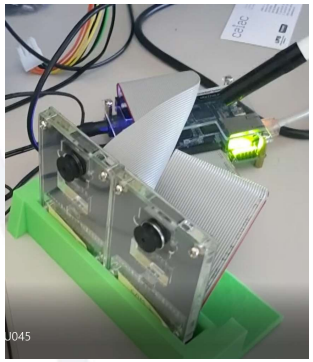
Humidity RF Sensor

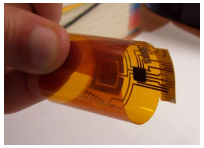


Wearables
(càrrega Qi)

Hybrid systems
(wireless BLE insoles)



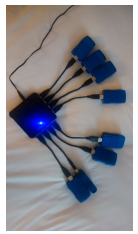


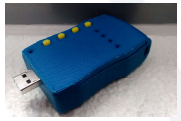


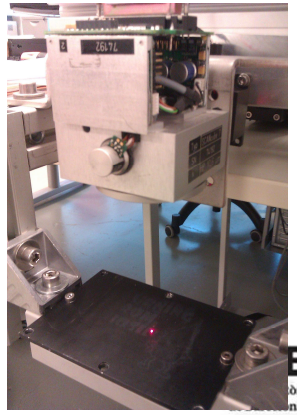
FPGA based
Embedded
Vision

Augmented Reality

FPGA based MEMS microphones



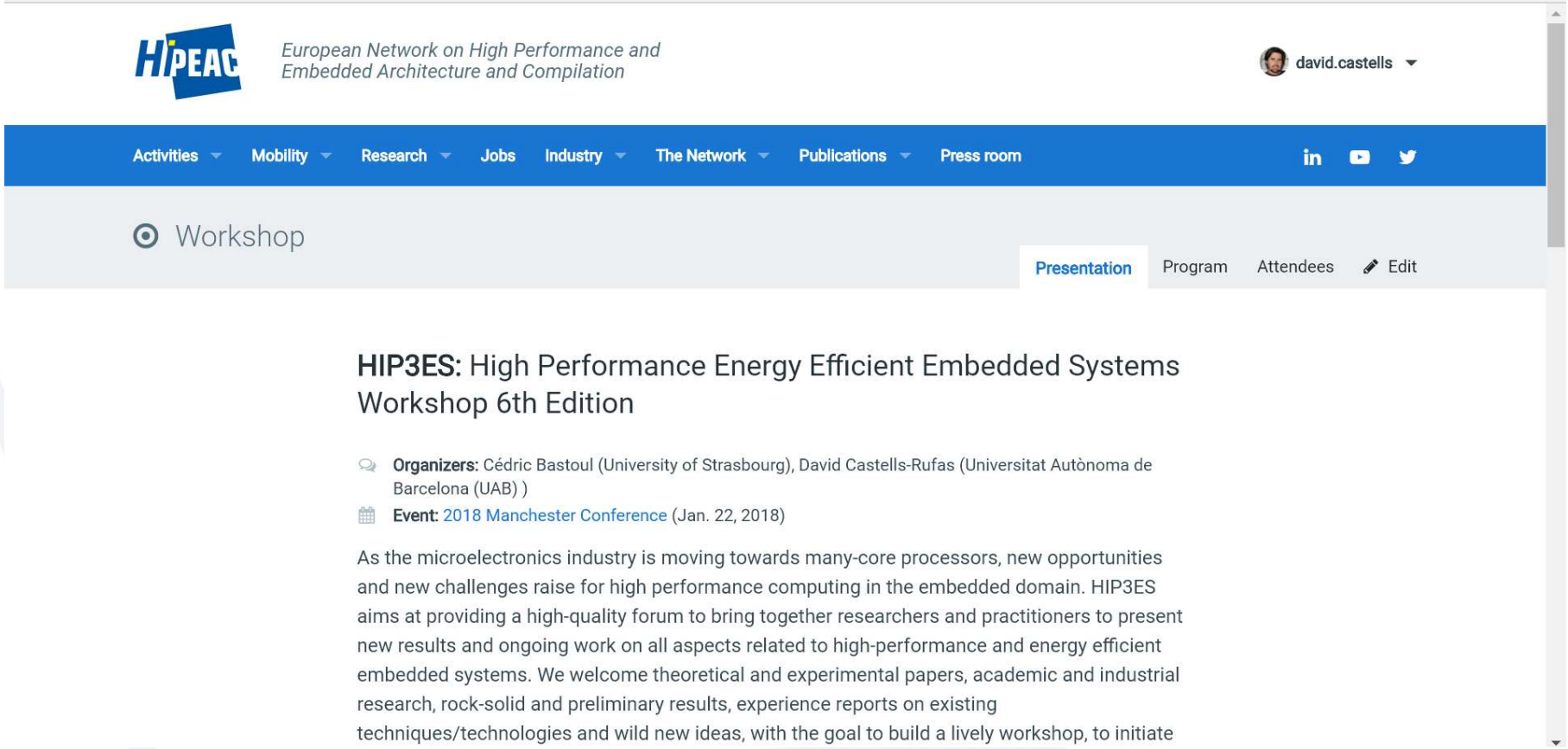




FPGA based
laser
controller

We are part of the HIPEAC NoE

- Organizing the HIP3ES Workshop



Introduction

History of Energy Efficiency (G from Greeness)

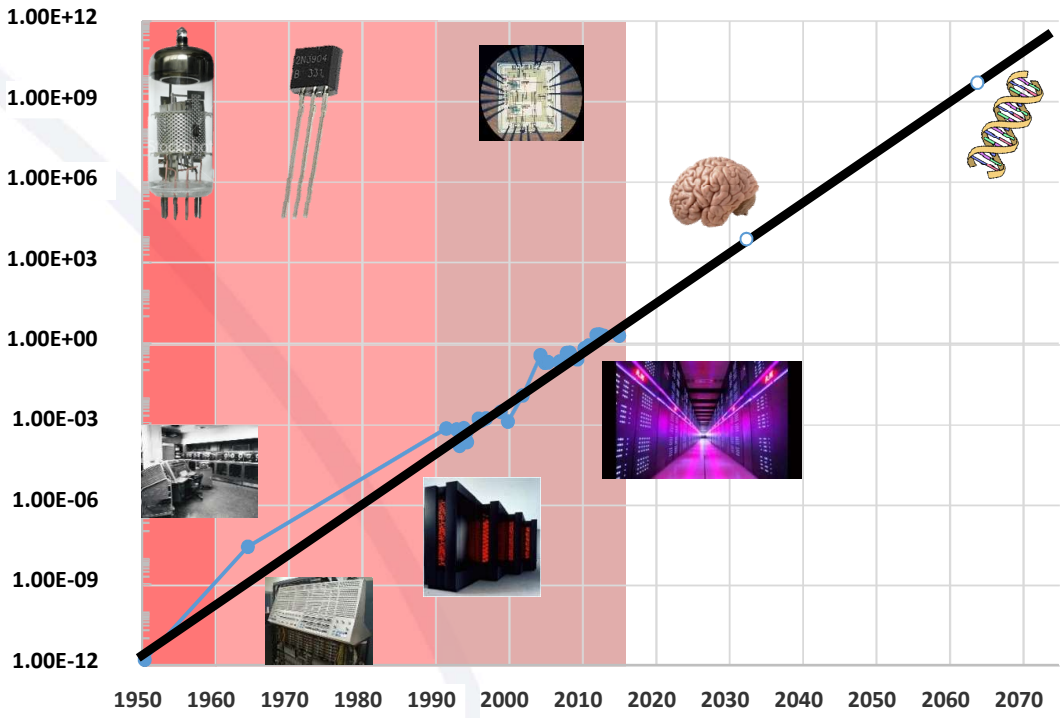
- Computing Eras
- Operations per Joule

$$G = \frac{Op}{E} = \frac{Op}{T}\frac{1}{P}$$

- OPS per Watt

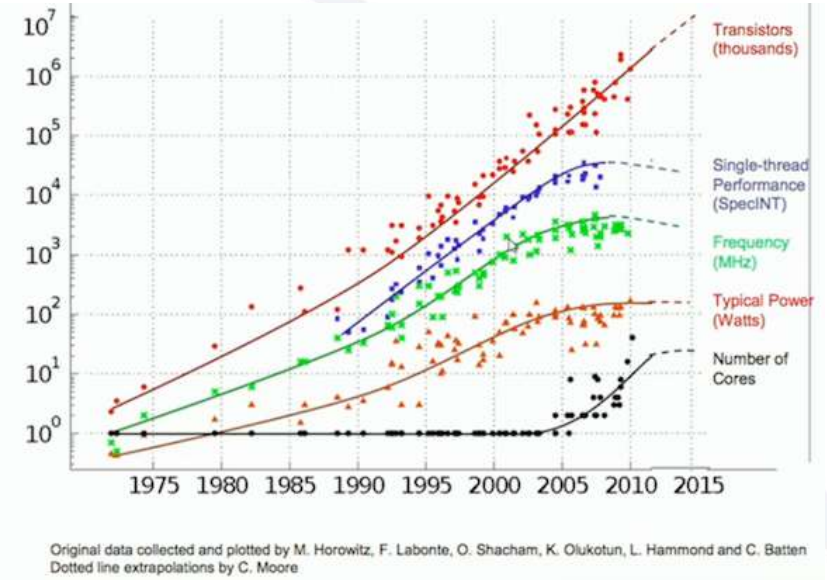
  THE GREEN 500 uses GFLOPS/W

- 12 orders of magnitude in 65 yr.
  - –Disruptive changes
  - –Architectural changes



Rules changed before 2010

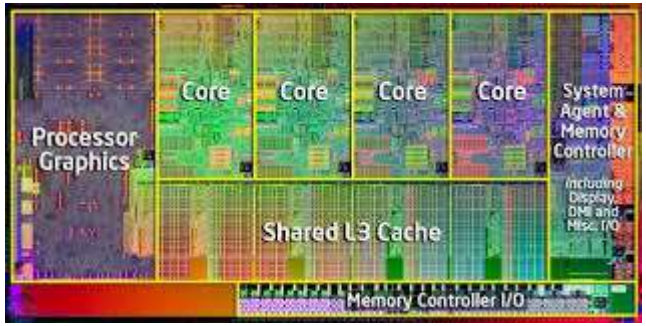
- Dennard Scaling Rules were no longer valid
- Thermal Density

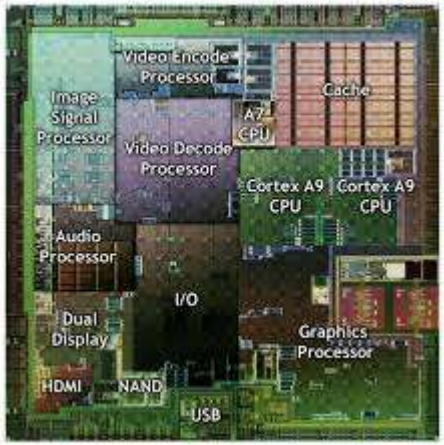


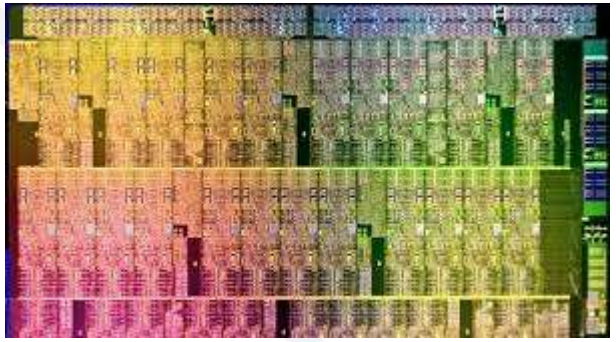
	Dennard (1974)	Taylor (2013)
Tran. Size tox, W, L	$1/s$	$1/s$
Devices	s^2	s^2
Voltage V	$1/s$	1
Current I	$1/s$	1
Capacitance C	$1/s$	$1/s$
Intrinsic delay CV/I	$1/s$	$1/s$
Power dissipation VI	$1/s^2$	1
Power density	1	s^2
Frequency	s	1

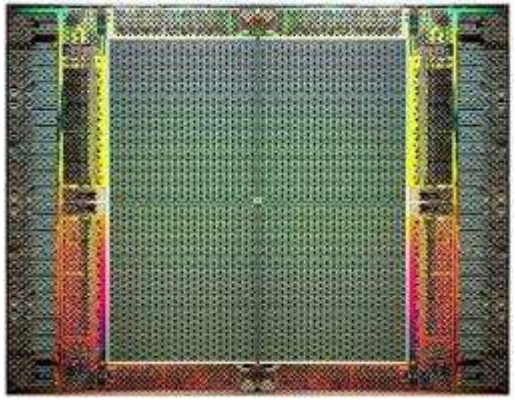
Alternatives

- Multicores
- HW Coprocessors
- GPUs
- FPGAs

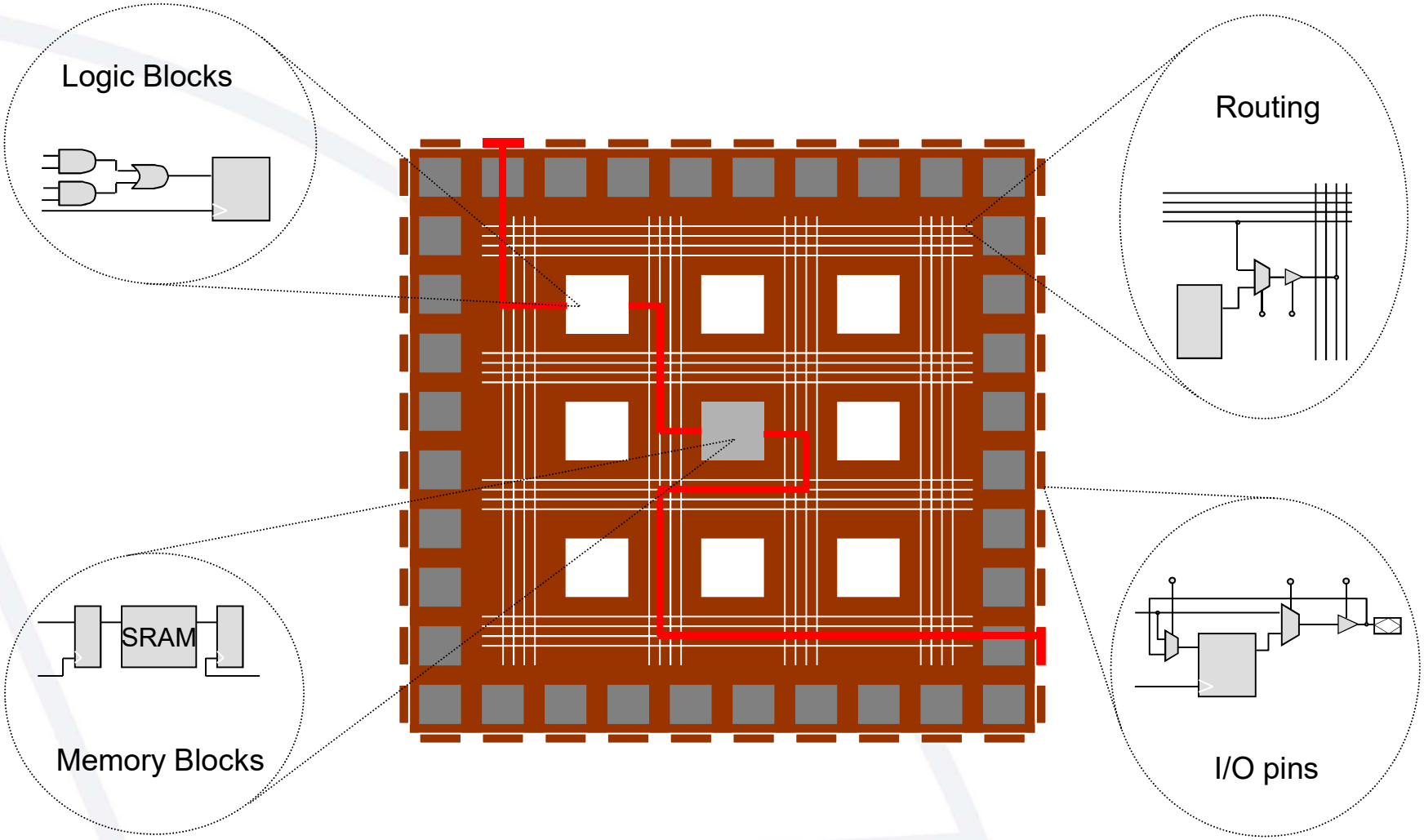






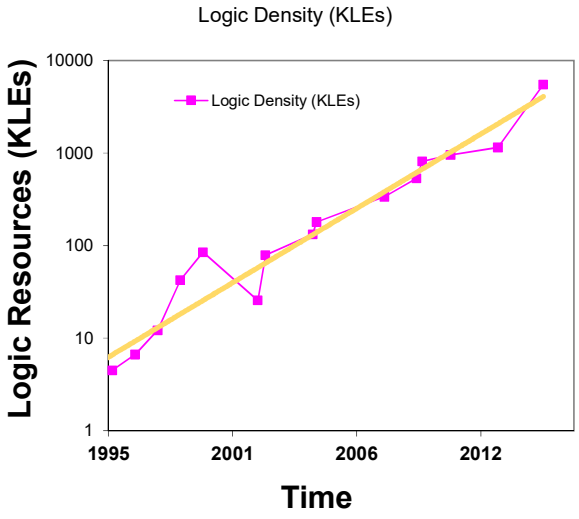


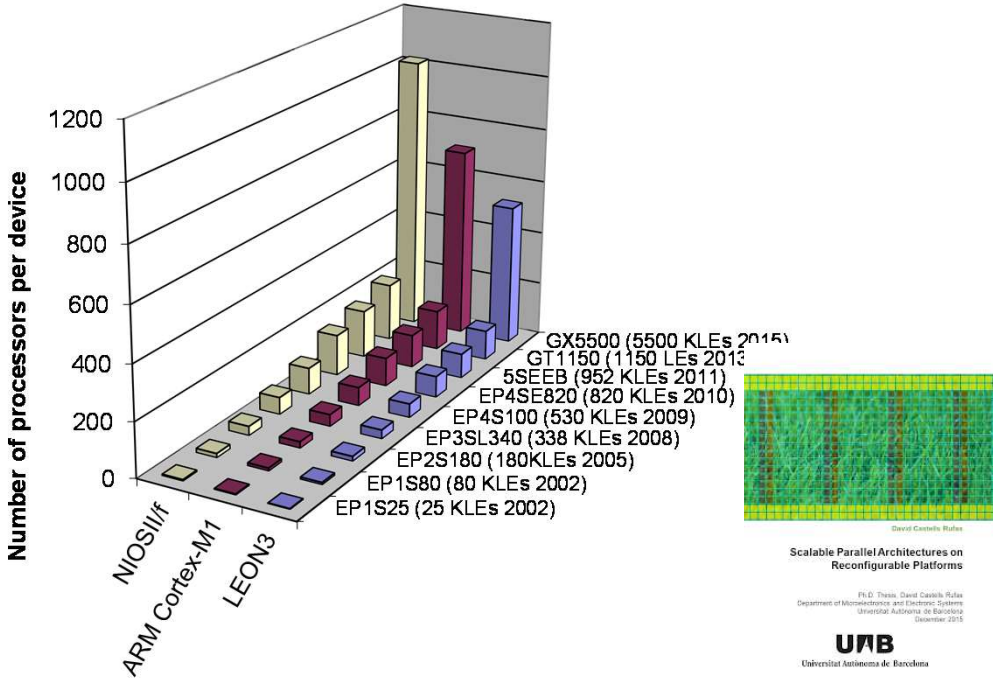
What is an FPGA?



How “Big” is an FPGA?

- A simple RISC processor takes about 6 KLE
- FPGAs are early adopters of last technology nodes (11nm Intel node)
- Hundreds of soft-core processors can already be embedded in current big FPGA devices





Why FPGAs can be Energy Efficient ?

Energy Efficiency Drivers in CMOS era

$$G = \frac{\mathrm{OPC} \cdot f_{clk}}{P} = \frac{\mathrm{OPC} \cdot f_{clk}}{P_{dyn} + P_{sta}} = \frac{\mathrm{OPC} \cdot f_{clk}}{N \cdot \alpha \cdot f_{clk} \cdot C \cdot V^2 + N \cdot V \cdot I_{leakage}}$$

$$G_{dyn} = \frac{OPC}{N \cdot \alpha \cdot C \cdot V^2}$$

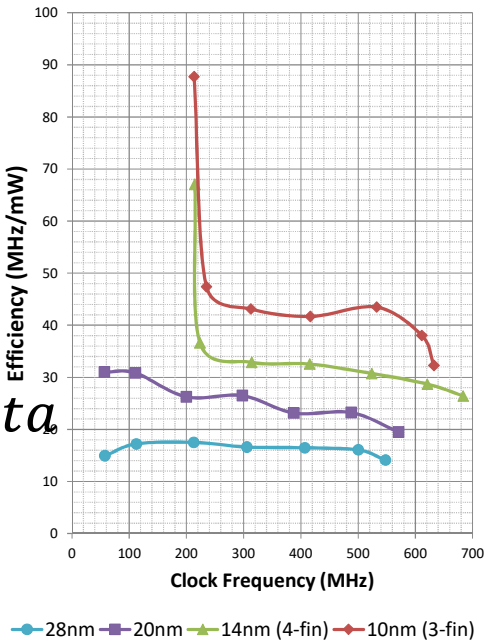
$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (C \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}}\right)}$$

UAB Universitat Autònoma de Barcelona

TECNIO

Why FPGAs can be Energy Efficient ?

$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (C \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}}\right)}$$



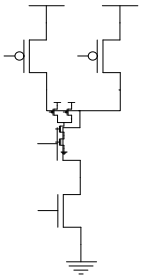
- Before Synthesis
  - $\nabla f_{clk} \Rightarrow \nabla C, \nabla N \Rightarrow \nabla P_{dyn}, \nabla P_{sta}$
- After Synthesis
  - $\nabla f_{clk} \Rightarrow \Delta \int P_{sta}$
- Empirical f_{max}

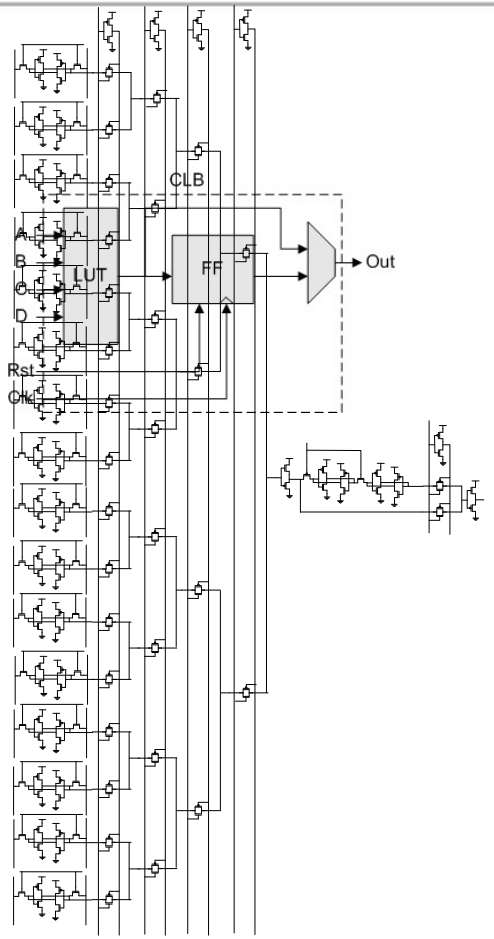
$$K_f = \frac{f_{clk\ IC}}{f_{clk\ FPGA}} \approx 20$$

Why FPGAs can be Energy Efficient ?

$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (C \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}}\right)}$$

- FPGAs use more transistors





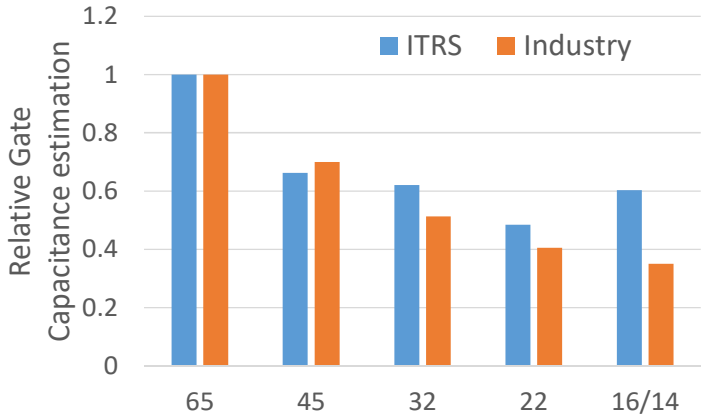
Circuits	Technology	Area Overhead
Small Benchmarks [Kuon07]	90 nm	23-55
Pentium [Lu07]	65 nm	53
OpenSparc, Atom, Nehalem [Wong11]	65 nm	17-27

$$K_N = \frac{N_{FPGA}}{N_{IC}} \approx 40$$

Why FPGAs can be Energy Efficient ?

$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (\boldsymbol{C} \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}}\right)}$$

- C is complex
  - –Transistor tech. & sizing, Fan-out, Wiring
- Isolate Tech. Node contribution
- ITRS predictions are not exactly inline with industry

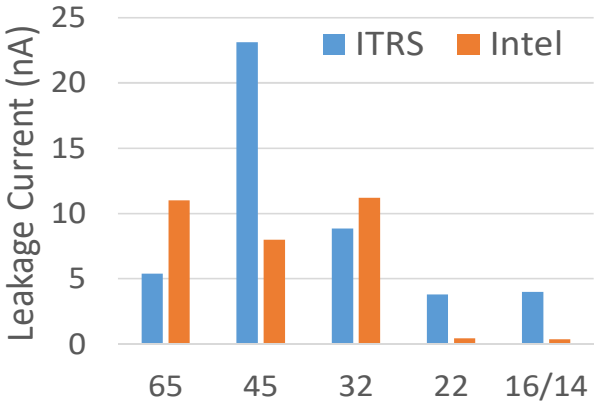


Node Name	65nm	45nm	32nm	22nm	14nm
C_g (aF)	72.9	49.65	41.32	32.44	34.78
Relative C_g	1	0.68	0.56	0.44	0.47

Why FPGAs can be Energy Efficient ?

$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (C \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}} \right)}$$

- Leakage power is a major concern
  - –But due to N
- Fixed by tech. Node (sleep modes)
- Isolate Node Contribution.



Node Name	65nm	45nm	32nm	22nm	14nm
$I_{leakage}$ (nA)	11	8	11.2	0.45	0.35
Relative $I_{leakage}$	0.98	0.71	1	0.04	0.03

Why FPGAs can be Energy Efficient ?

$$G = \frac{OPC}{N \cdot \left(\alpha \cdot (C \cdot V^2) + \frac{V \cdot I_{leakage}}{f_{clk}}\right)}$$

>1

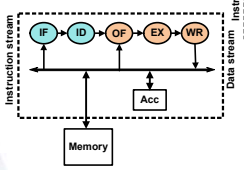
1

<1

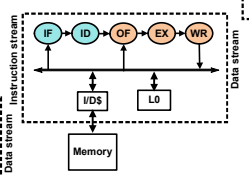
1/5

1/10

1/769

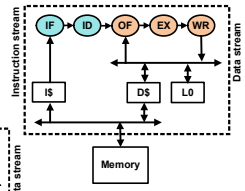


Von-Neumann Machine

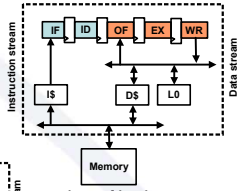


Register File

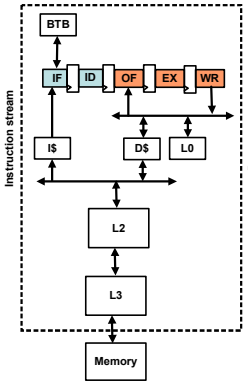
Cache (L0)



Harvard Architecture

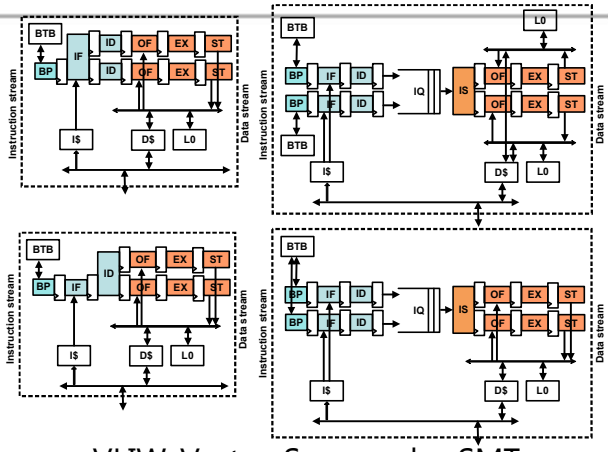


Pipelining



Branch Prediction

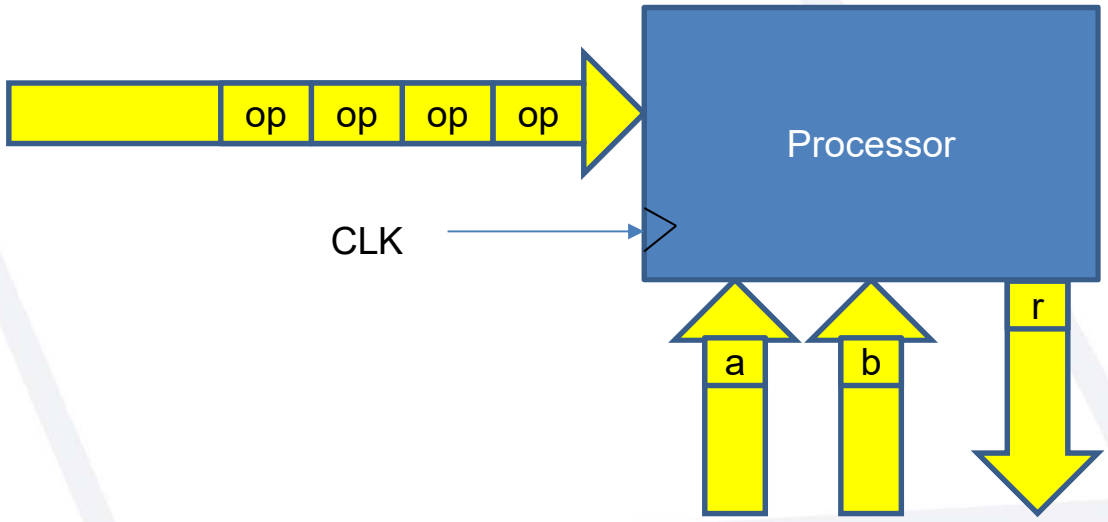
Cache Hierarchy



VLIW, Vector, Superscalar, SMT

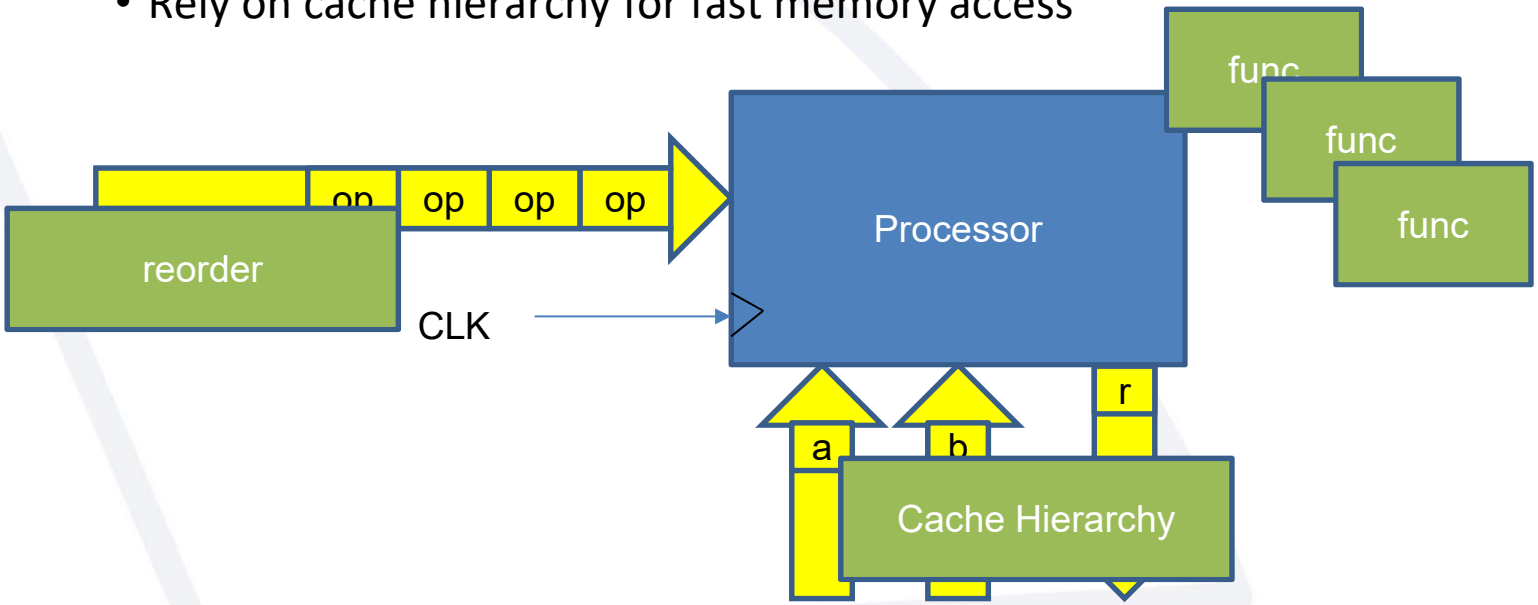
The Basic Fundamental Problems to increase OPC

- Frequency is limited
- Memory Access Latency > several clock cycles
- Data Dependency prevents parallelization



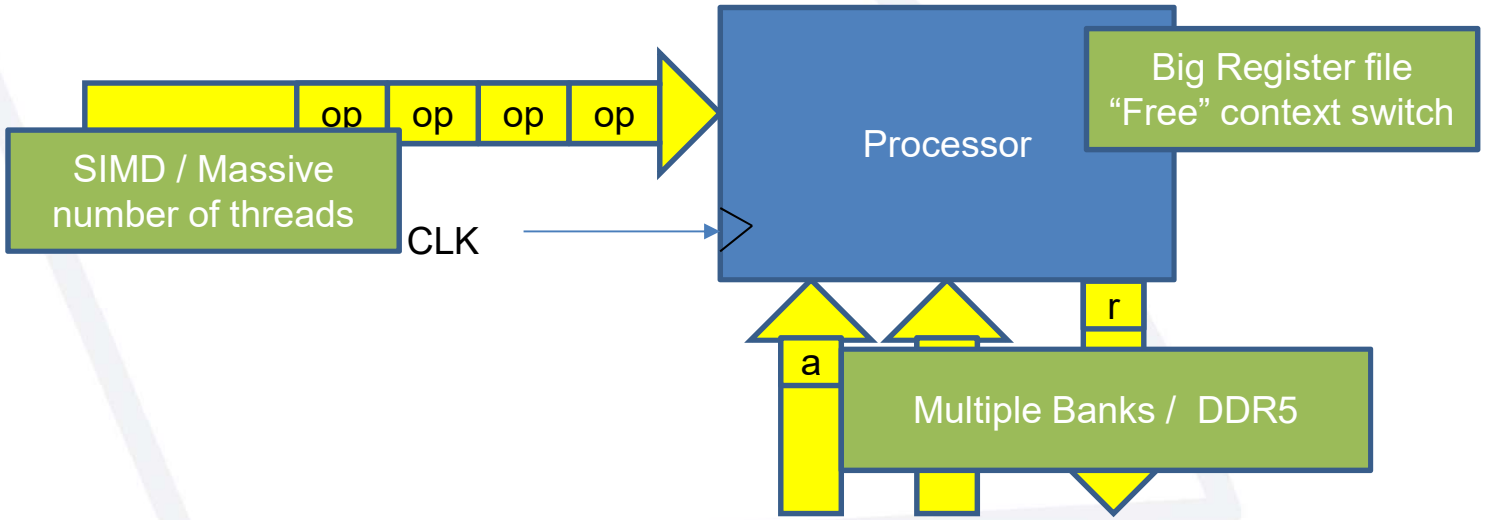
How Superscalar uP get the Job done?

- High frequency (3GHz)
- Have multiple functional units
- Have long instructions queues so that dependency analysis can be done and operations can be scheduled simultaneously
- Rely on cache hierarchy for fast memory access



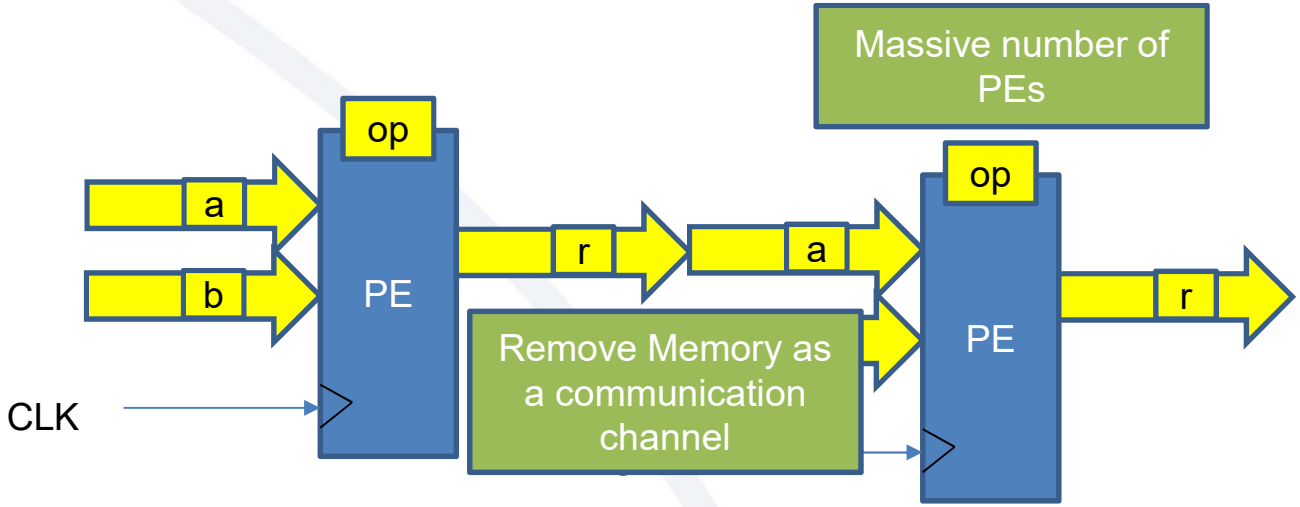
How GPUs get the Job done ?

- High frequency (2GHz)
- Have multiple SIMD processors and support for massive number of threads with fast context switch (1 clk)
- Very High Memory Bandwidth (DDR5), multiple Banks
- Avoid data dependencies by running different threads
  - data accesses from different threads are not dependent



What FPGAs can provide?

- Remove intermediate memory from computation datapaths
- Allow much higher number simultaneous computation units
- Fine grain (bit level) computation



- Problems
  - Lower frequency (due to overheads)
  - Difficult to program (HDL)

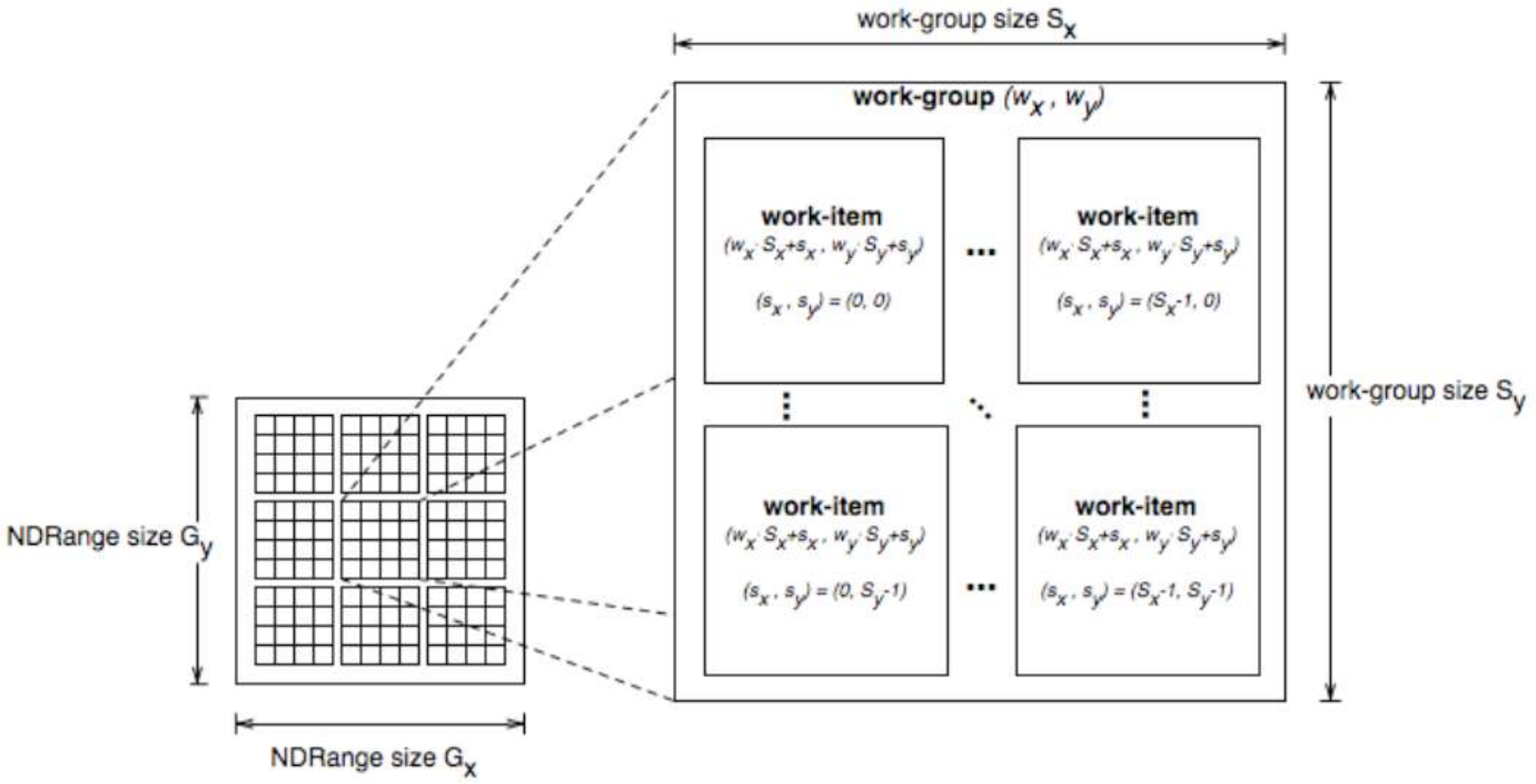
OpenCL for FPGAs

OpenCL

- API for the execution of “parallel” code (kernels) to be run in accelerators
- Programmers take care of parallelism
- OpenCL runtime
  - Compiles the code targeting the accelerator platform
  - Programs and executes (communicates with) the kernels in the accelerator platform

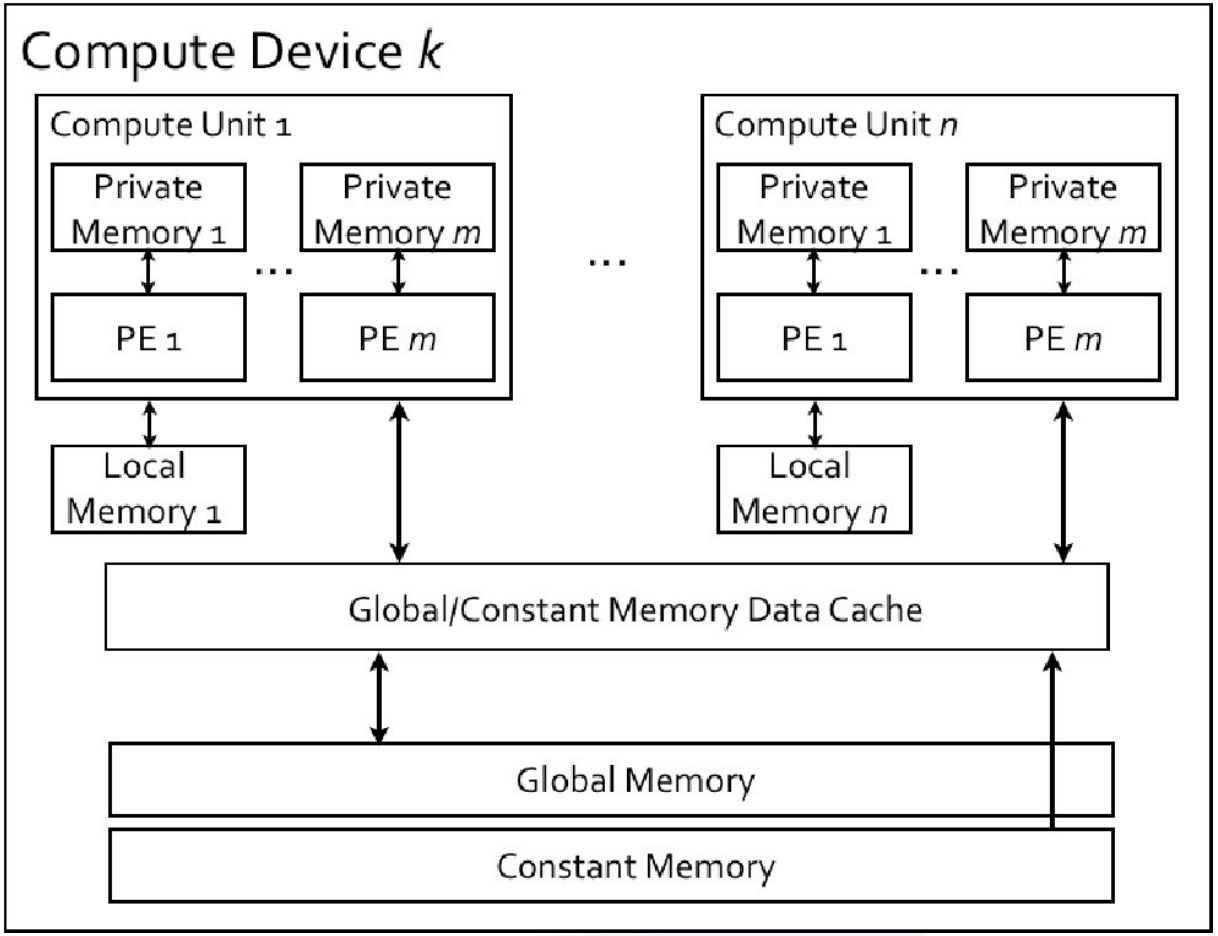
Logical organization of work

- Work Items (like threads)
- Work Groups (groups of threads executed in the same computing unit)



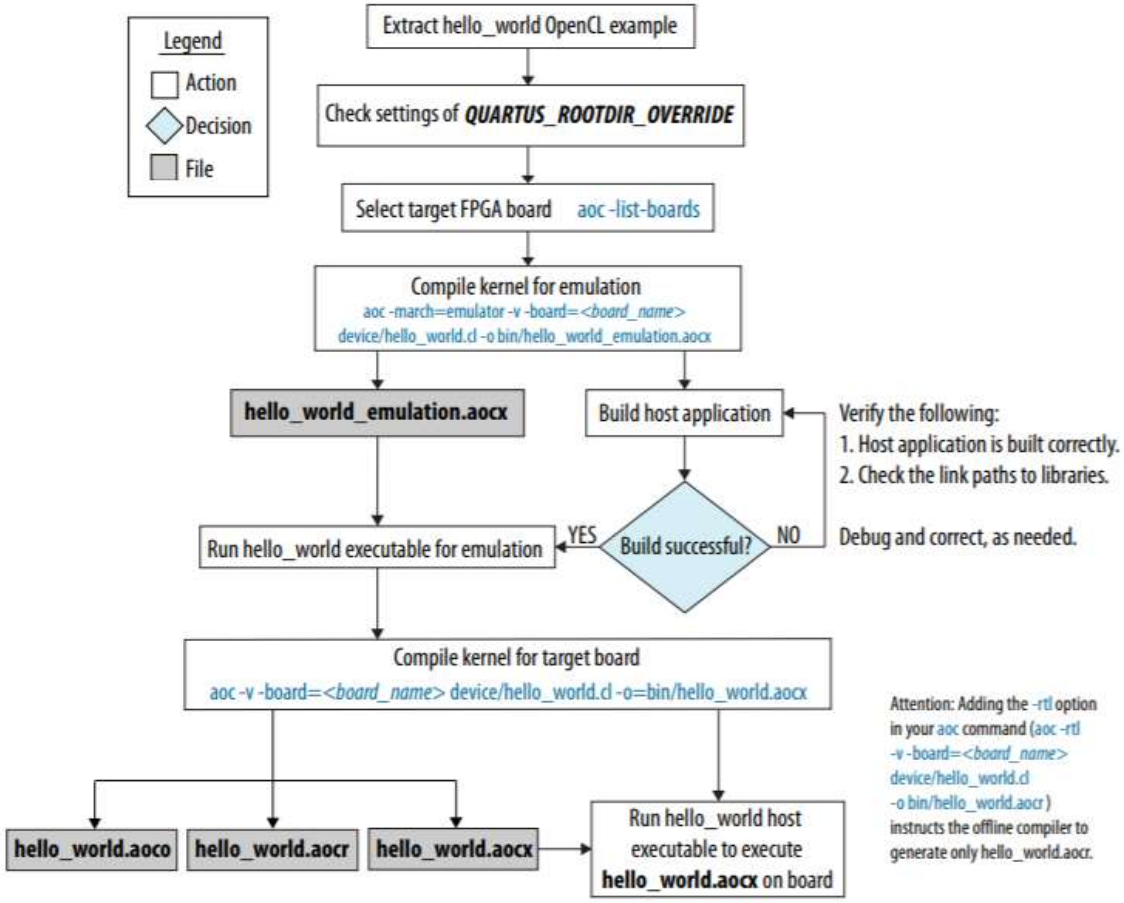
Conceptual OpenCL device architecture

- Host is not shown



The Intel FPGA OpenCL flow

- You need a Board Support Package



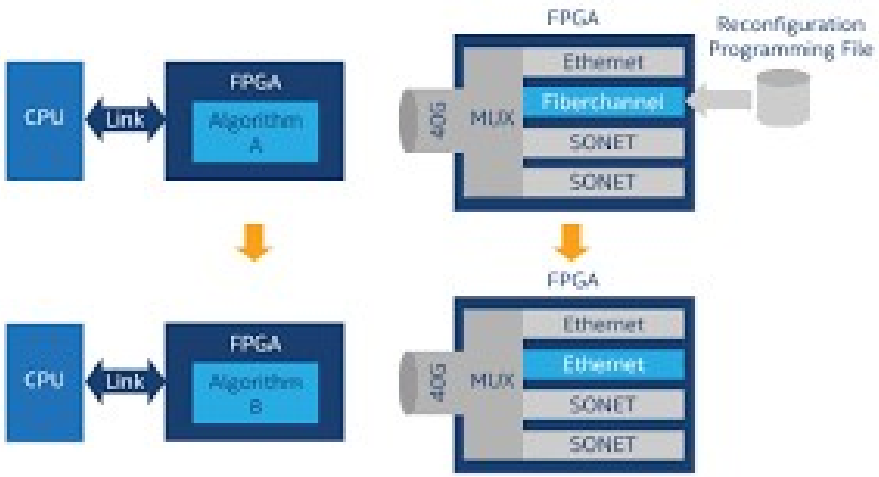
The Intel FPGA OpenCL Flow

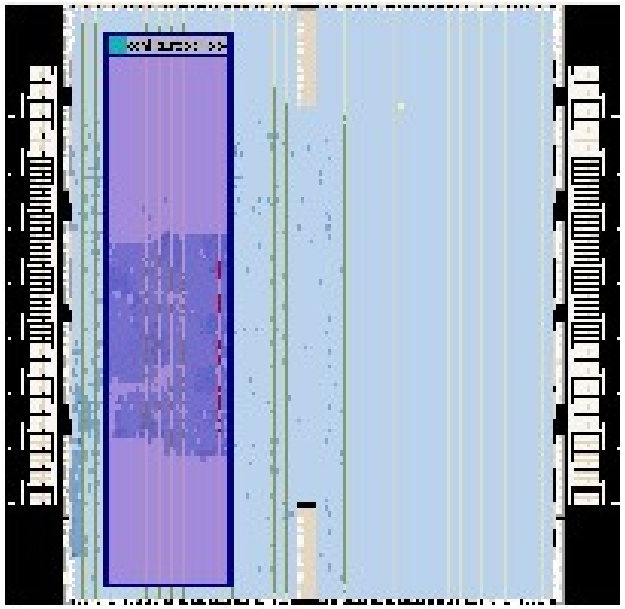
- You have **MUCH MORE** freedom when you design hardware
- OpenCL
  - –Allows fast Design Space Exploration
  - –Easier to code than HDL (Verilog, VHDL)

- FPGA compilation takes long time
  - – No runtime compilation (unlike GPUs)
  - – Ahead of time compiler (clCreateProgramWithBinary)

The Intel FPGA OpenCL Flow

- Partial Reconfiguration
  - Reserve a portion of the FPGA for future use
  - Change it during runtime

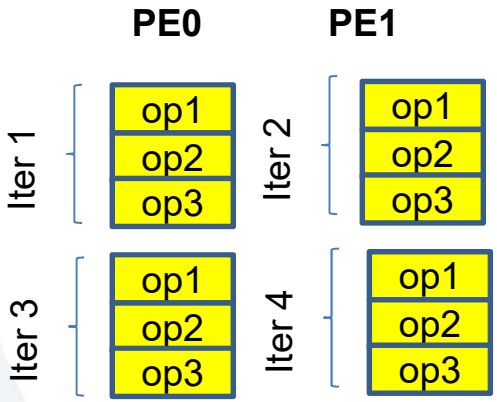




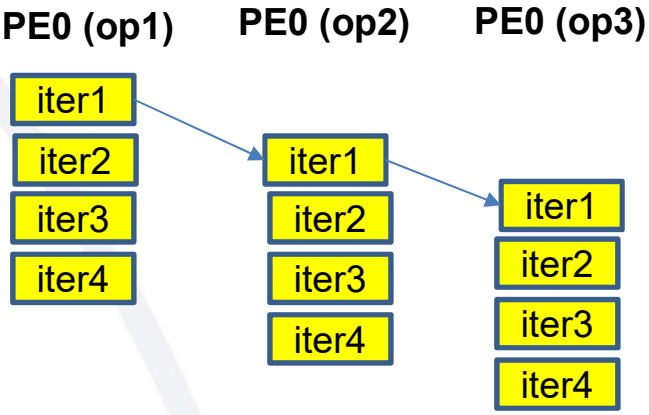
Loop Unrolling and Loop pipelining

```
for each (iter ){
    op1;
    op2;
    op3;
}
```

- Unrolling



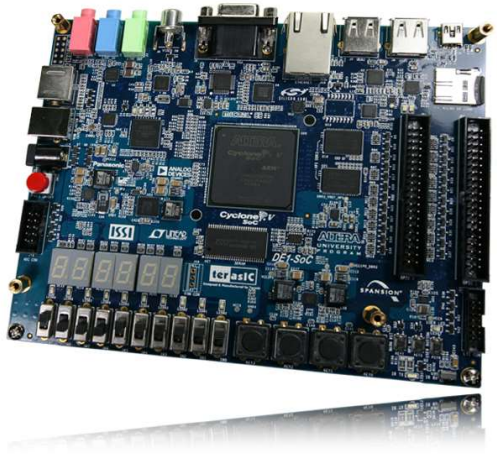
- Pipelining



Preliminar
The Platform

The Board

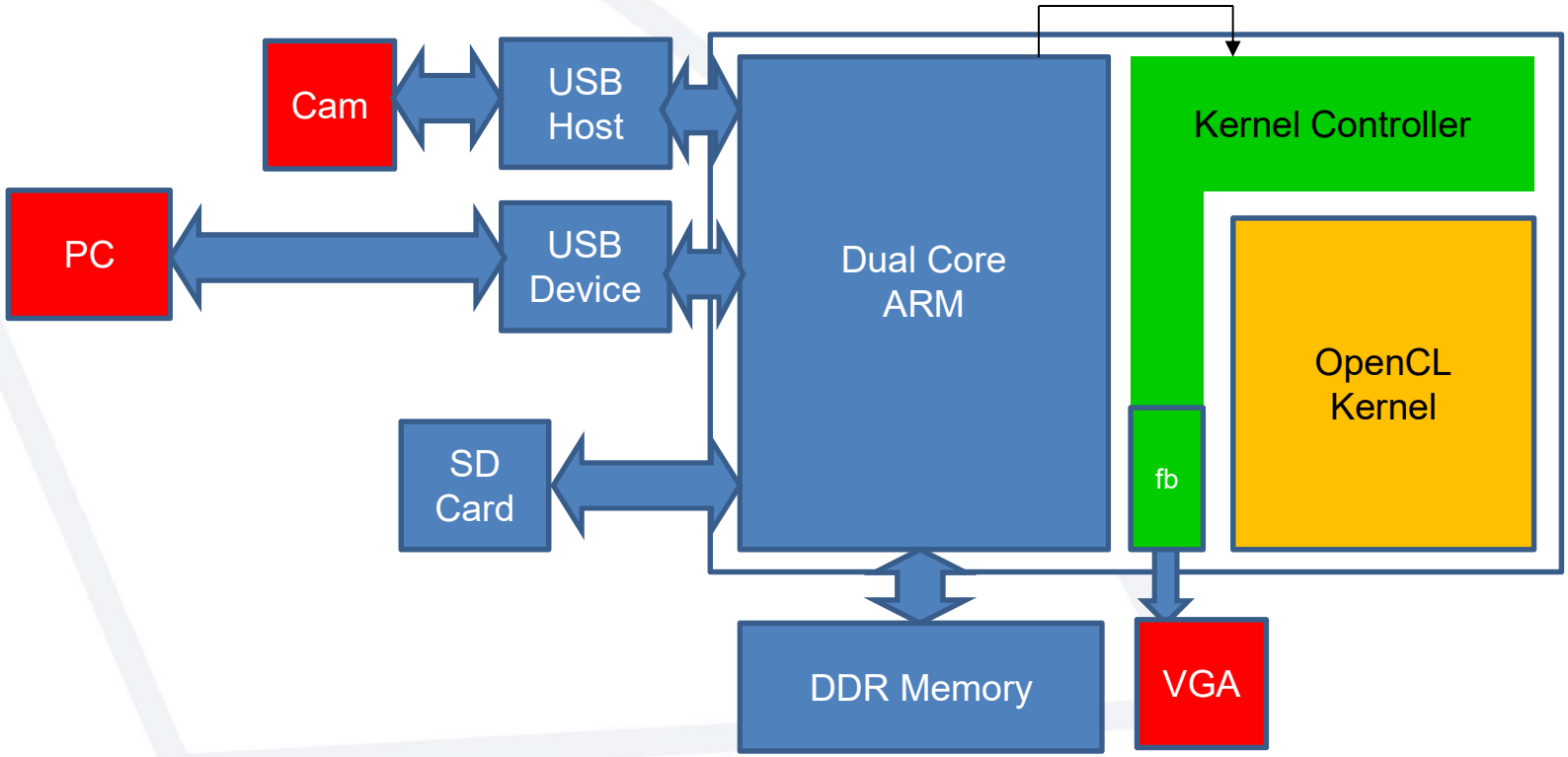
- Terasic DE1-SoC (~200€)
- Cyclone V SOC
  - 85 KLEs
  - 4 MBits
- 2 USB Master ports
- VGA connector
- USB (device) Serial Port communication



Infrastructure

- ARM processor configures the FPGA part

- Cyclone V does not support Dynamic Reconfiguration (reboot needed)



Example 1: Sobel Edge Detector

- Approximation to derivate
- Computer Image processing Basic Step
- Edge detector
- Simple Kernel

$$\frac{\partial f}{\partial x}[x, y] \approx f[x+1, y] - f[x, y]$$

-1	0	1
-2	0	2
-1	0	1

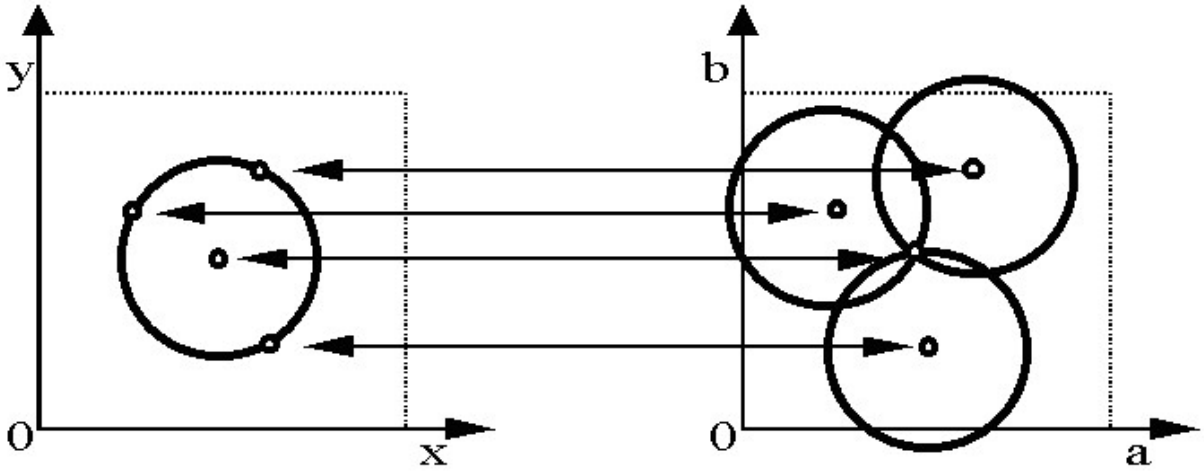
s_x

1	2	1
0	0	0
-1	-2	-1

s_y

Example 2: Circle Detection

- Hough Transform for circle detection
- For each point we must vote for all possible candidates
  - The maximum value will be found at the center



Hands-on

First Kernel: Hello World

KERNEL

```
__kernel void hello_world(int thread_id_from_which_to_print_message) {
  // Get index of the work item
  unsigned thread_id = get_global_id(0);

  if(thread_id == thread_id_from_which_to_print_message) {
    printf("Thread #%u: Hello from Altera's OpenCL Compiler!\n", thread_id);
  }
}
```

HOST

```
…
int id = 2;
status = clSetKernelArg(kernel, 0, sizeof(cl_int), (void*)&id);

// Configure work set over which the kernel will execute
size_t gSize[3] = {10, 1, 1};
size_t lSize[3] = {1, 1, 1};

status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, gSize, lSize, 0, NULL, NULL);
…
```

Compile

```
g++ -std=c++11 src/*.cpp common/src/AOCLUtils/*.cpp -I/root/aocl-rte-17.1.0-590.arm32/host/include  -
Icommon/inc -L/root/aocl-rte-17.1.0-590.arm32/board/c5soc/arm32/lib -L/root/aocl-rte-17.1.0-
590.arm32/host/arm32/lib -Wl,--no-as-needed -lalteracl -lintel_soc32_mmd -lelf -lstdc++ -
lacl_emulator_kernel_rt -lalterahalmmd -lc_accel_runtime -lOpenCL -o hello_world.exe
```

Hello World Results

- Early Compilation Report

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/hello_world/reports/report.html

- Compilation Summary

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/hello_world/top.fit.summary

C Implementation

```
unsigned char getc(unsigned char* img, int x, int y, int w, int h, int c) {
    if ((x < 0 || x >= w)|| (y < 0 || y >= h)) return 0; return img[(y*w+x)*3 +c];}

void putc(unsigned char* img, int x, int y, int w, int c, unsigned char v ) {
    img[(y*w+x)*3 +c] = v;}

void sobel2D(unsigned char* img, unsigned char* dst) {
    const int w = 640; const int h = 480;

    int kernelx[3][3] = {{-1, 0 , 1}, {-2, 0,  2}, {-1, 0,  1}};
    int kernely[3][3] = {{-1, -2 , -1}, { 0,  0,  0}, { 1, 2,  1}};

    for (int c= 0;  c < 3; c++)
        for (int y = 0;  y < h; y++)
            for (int x = 0;  x < w; x++) {
                int sumx = 0; int sumy = 0;

                for (int ky = 0; ky < 3; ky++)
                    for (int kx = 0; kx < 3; kx++) {
                        sumx += kernelx[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                        sumy += kernely[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                    }
                int nv = sumx+sumy;
                if (nv < 0) nv = 0;
                if (nv > 255) nv = 255;
                putc(dst, x, y, w, c, nv);
            }
```

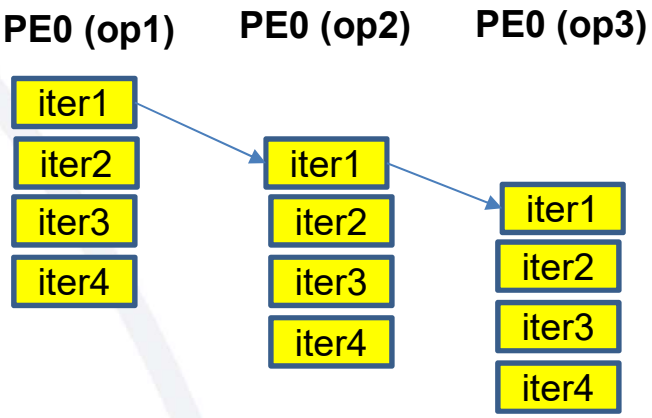
Results

- ARM is running at 200MHz
- Using some optimizations
- FPS 1.1

Review: Pipelined Kernels

- In the host we ask for 1 work item
- In the kernel We do not use get_global_id()
- The important factor is the Interval Innitiation Factor (II)

- Pipelining

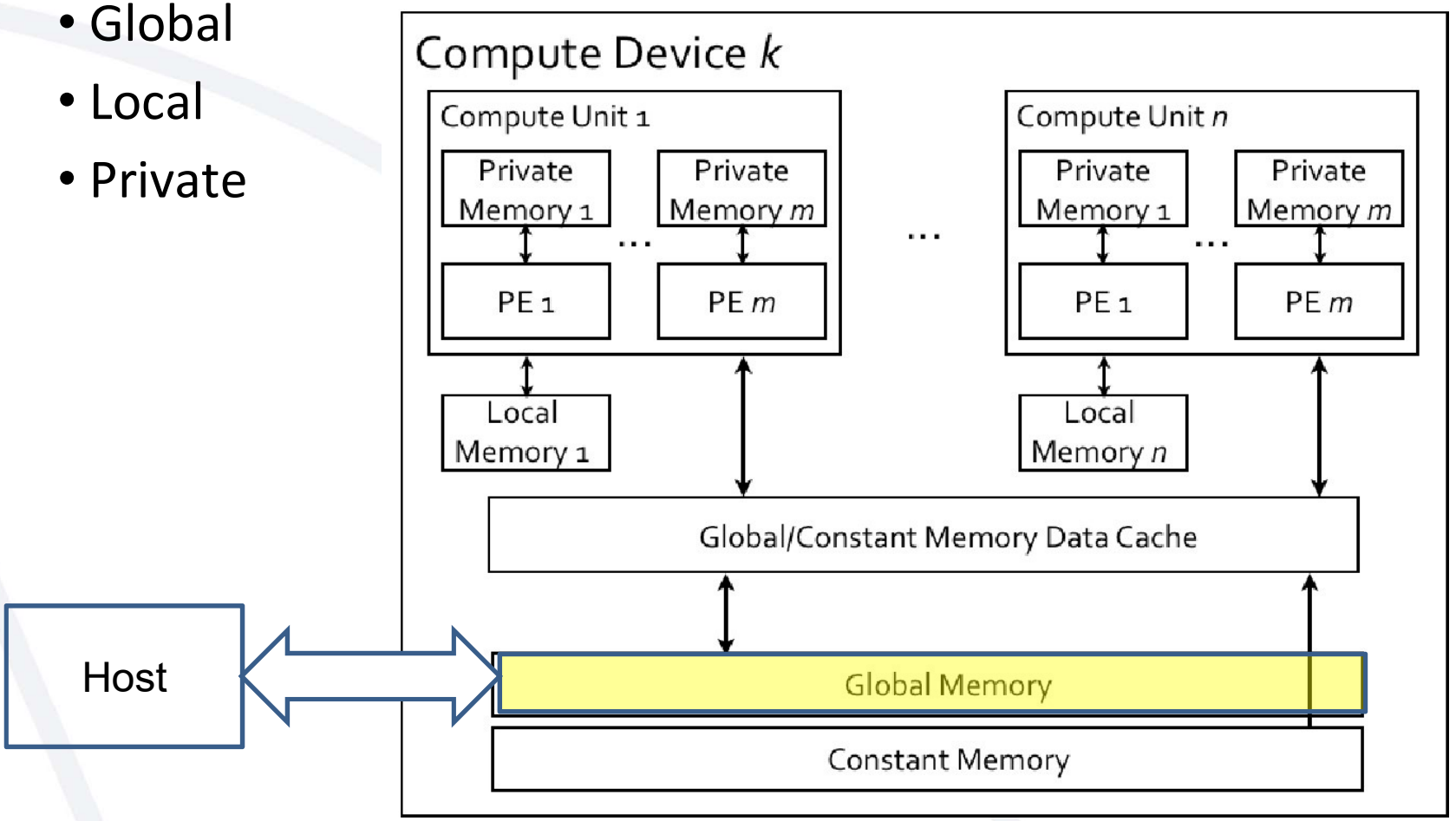


Sobel2D_1 As a Kernel

```
unsigned char getc(__global unsigned char* img, int x, int y, int w, int h, int c) {
    if ((x < 0 || x >= w)|| (y < 0 || y >= h)) return 0; return img[(y*w+x)*3 +c];}

void putc(__global unsigned char* img, int x, int y, int w, int c, unsigned char v ) {
    img[(y*w+x)*3 +c] = v;}

__kernel void sobel2D(__global unsigned char* img, __global unsigned char* dst){
    const int w = 640; const int h = 480;

    int kernelx[3][3] = {{-1, 0 , 1}, {-2, 0,  2}, {-1, 0,  1}};
    int kernely[3][3] = {{-1, -2 , -1}, { 0,  0,  0}, { 1, 2,  1}};

    for (int c= 0;  c < 3; c++)
        for (int y = 0;  y < h; y++)
            for (int x = 0;  x < w; x++) {
                int sumx = 0; int sumy = 0;

                for (int ky = 0; ky < 3; ky++)
                    for (int kx = 0; kx < 3; kx++) {
                        sumx += kernelx[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                        sumy += kernely[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                    }
                int nv = sumx+sumy;
                if (nv < 0) nv = 0;
                if (nv > 255) nv = 255;
                putc(dst, x, y, w, c, nv);
            }
```

Review OpenCL Memory regions

- Global
- Local
- Private



Sobel2D_1: The Host

```
status = clEnqueueWriteBuffer(queue, m_memImgSrc, CL_TRUE, 0, 640*480*3, img, 0, NULL, NULL);
status = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&m_memImgSrc);

status = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void*)&m_memImgDst);

size_t wgSize[3] = {1, 1, 1};
size_t gSize[3] = {1, 1, 1};

status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, gSize, wgSize, 0, NULL, NULL);

status = clEnqueueReadBuffer(queue, m_memImgDst, CL_TRUE, 0, 640*480*3, dst, 0, NULL, NULL);
```

Sobel2D_1: Results

- Early Compilation Report

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_1/reports/report.html

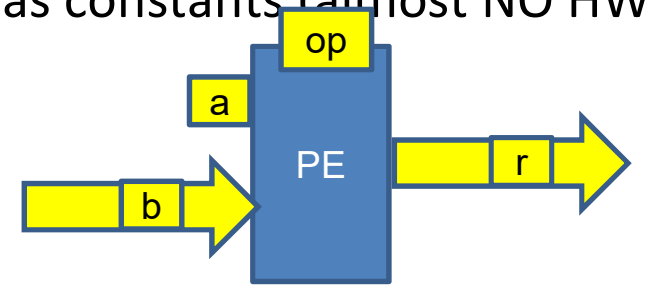
- Compilation Summary

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_1/top.fit.summary

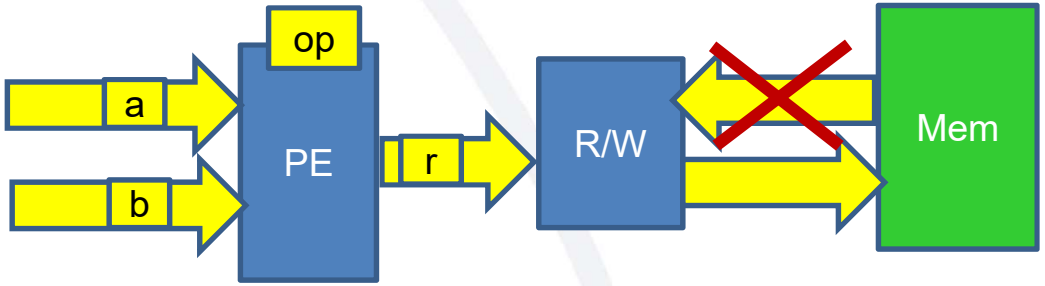
- FPS 1.2
- 22% ALMs, 28% BRAMs, Clk 157 MHz
- Coalesced Memory Accesses for inner kernel loop !
- Pipelinging for outer kernel loop

Improvement: Constant variables

- __constant
  - Ensures that variables are used as constants (almost NO HW COST)



- __restrict
  - Ensures that only a Load or Store unit is implemented for a memory region



Sobel2D_3: As a Kernel

```
unsigned char getc(__global unsigned char* img, int x, int y, int w, int h, int c) {
    if ((x < 0 || x >= w)|| (y < 0 || y >= h)) return 0; return img[(y*w+x)*3 +c];}

void putc(__global unsigned char* img, int x, int y, int w, int c, unsigned char v ) {
    img[(y*w+x)*3 +c] = v;}

__constant const int kernelx[3][3] = {{-1, 0 , 1}, {-2, 0,  2}, {-1, 0,  1}};
__constant const int kernely[3][3] = {{-1, -2 , -1}, { 0,  0,  0}, { 1, 2,  1}};

__kernel void sobel2D(__global unsigned char* restrict img, __global unsigned char* restrict dst){
    const int w = 640; const int h = 480;


    for (int c= 0;  c < 3; c++)
        for (int y = 0;  y < h; y++)
            for (int x = 0;  x < w; x++) {
                int sumx = 0; int sumy = 0;

                for (int ky = 0; ky < 3; ky++)
                    for (int kx = 0; kx < 3; kx++) {
                        sumx += kernelx[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                        sumy += kernely[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                    }
                int nv = sumx+sumy;
                if (nv < 0) nv = 0;
                if (nv > 255) nv = 255;
                putc(dst, x, y, w, c, nv);
            }
```

Sobel2D_3: Results

- Early Compilation Report

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_3/reports/report.html

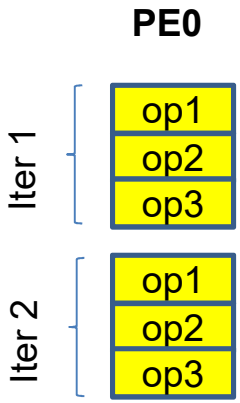
- Compilation Summary

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_3/top.fit.summary

- FPS 6-7
- 21% ALMs, 25% BRAMs, Clk 150 MHz
- Coalesced Memory Accesses for inner kernel loop !
- Pipelinging for outer kernel loop

Alternative Implementation

- Using NDRange kernel (GPU like)
- Going to assign a Work Item (thread) to every line of the image



Sobel2D_4 As a Kernel

```
unsigned char getc(__global unsigned char* img, int x, int y, int w, int h, int c) {
    if ((x < 0 || x >= w)|| (y < 0 || y >= h)) return 0; return img[(y*w+x)*3 +c];}

void putc(__global unsigned char* img, int x, int y, int w, int c, unsigned char v ) {
    img[(y*w+x)*3 +c] = v;}

__kernel void sobel2D(__global unsigned char* img, __global unsigned char* dst){
    const int w = 640; const int h = 480;

    int kernelx[3][3] = {{-1, 0 , 1}, {-2, 0,  2}, {-1, 0,  1}};
    int kernely[3][3] = {{-1, -2 , -1}, { 0,  0,  0}, { 1, 2,  1}};

    int y = get_global_id(0);

    for (int c= 0;  c < 3; c++)
   //      for (int y = 0;  y < h; y++)
            for (int x = 0;  x < w; x++) {
                int sumx = 0; int sumy = 0;

                for (int ky = 0; ky < 3; ky++)
                    for (int kx = 0; kx < 3; kx++) {
                        sumx += kernelx[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                        sumy += kernely[ky][kx] * getc(img, x+kx-1, y+ky-1, w,h, c);
                    }
                int nv = sumx+sumy;
                if (nv < 0) nv = 0;
                if (nv > 255) nv = 255;
                putc(dst, x, y, w, c, nv);
            }
```

Sobel2D_4: The Host

```
status = clEnqueueWriteBuffer(queue, m_memImgSrc, CL_TRUE, 0, 640*480*3, img, 0, NULL, NULL);
status = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&m_memImgSrc);

status = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void*)&m_memImgDst);

size_t gSize[3] = {480, 1, 1};
size_t lSize[3] = {1, 1, 1};

status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, gSize, lSize, 0, NULL, NULL);

status = clEnqueueReadBuffer(queue, m_memImgDst, CL_TRUE, 0, 640*480*3, dst, 0, NULL, NULL);
```

Sobel2D_4: Results

- Early Compiler Report

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_4/reports/report.html#view1

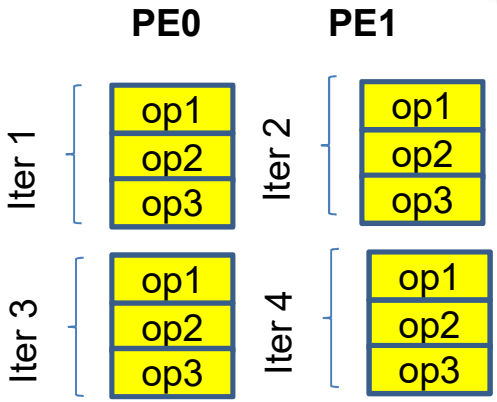
- Compilation Summary

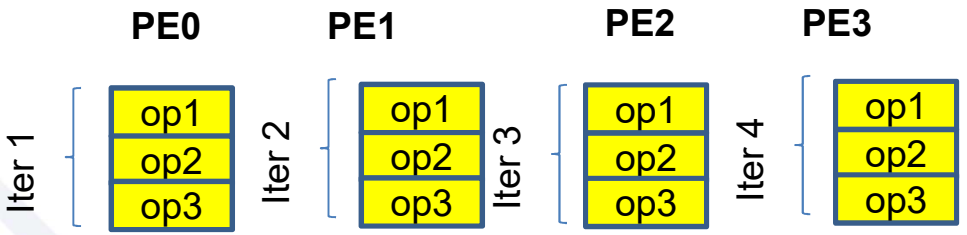
file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_4/top.fit.summary

- 24% ALMs, 24% BRAMs, clk 149 MHz
- Similar performance / **Better performance with version 5 (iterating on X)**

Alternative Implementation

- Increment the number of Computing Units
- 2 and 4





Sobel2D_6 and _7: Results

- Not very good results
- Replication of compute units does not have any benefit! We have a memory access bottleneck

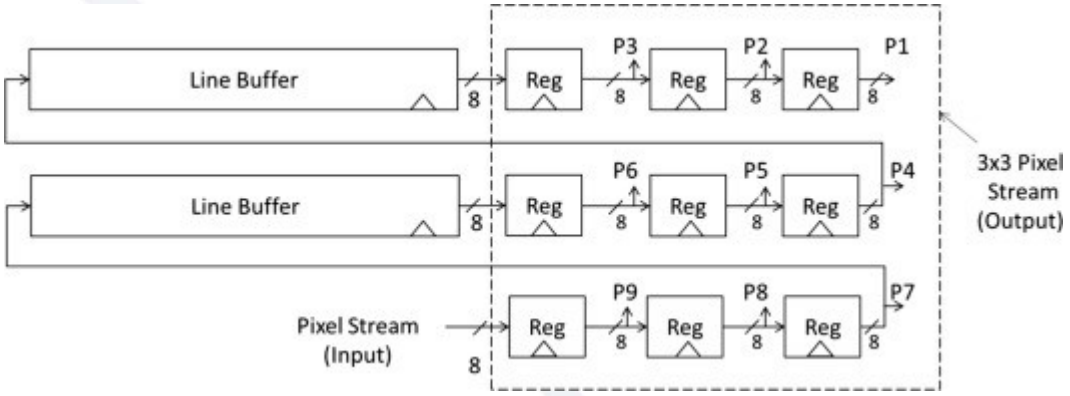
Version	Details	FPS	ALMs	BRAM s
Sobel2D_6	ND Range on Y (2 compute Units)	5	37%	24%
Sobel2D_7	ND Range on Y (4 compute Units)	5	65%	49%

- Early Report (4 PE)

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_7/reports/report.html

Alternative

- Implement a line buffer to reduce memory access / 9
- Common hardware design strategy for image processin



- A line buffer is a shift register! **We can express it using OpenCL!**
- **We must use a Pipelined Kernel Design (Single WorkItem)**

Sobel2D_8: Inferring Line Buffer

```
// FIFO of the last 2 rows
   unsigned char fifor[fifo_size]; unsigned char fifog[fifo_size]; unsigned char fifob[fifo_size];

   for (int index = -(fifo_size); index < (640*480) ; index++)
   {
         // Create a FIFO
         #pragma unroll
         for (int i = 0; i < (fifo_size-1); i++)
         {
            fifor[i] = fifor[i+1];
            fifog[i] = fifog[i+1];
            fifob[i] = fifob[i+1];
         }

         int fifox = (index+fifo_size) % 640;
         int fifoy = (index+fifo_size) / 640;

         fifor[fifo_size-1] = getc(img, fifox, fifoy, w, h, 0);
         fifog[fifo_size-1] = getc(img, fifox, fifoy, w, h, 1);
         fifob[fifo_size-1] = getc(img, fifox, fifoy, w, h, 2);
```

Sobel2D_8: Kernel Processing

```
 int computeKernel(unsigned char fifo[fifo_size])
{
    int sumx = 0;
    int sumy = 0;

    #pragma unroll
    for (int ky = 0; ky < 3; ky++)
        for (int kx = 0; kx < 3; kx++)
        {
            sumx += kernelx[ky][kx] * get_fifo(kx, ky);
            sumy += kernely[ky][kx] * get_fifo(kx, ky);
        }
    int nv = sumx+sumy;

    if (nv < 0) nv = 0;
    if (nv > 255) nv = 255;

    return nv;
}
```

Sobel2D_8: Results

•Early Compilation Report

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_8/reports/report.html

•Compilation Summary

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/sobel2d_8/top.fit.summary

- 27% ALMs, 18% BRAMs, clk 160MHz
- Kernel loop executed in Parallel!
- **Main loop Initiation Interval = 1!!**
- Line buffer implemented in FFs **(no local memory!)**

Increasing Complexity, a second Kernel

- Hough Transform
- Use 3D addressing {x, y, color cannel} to identify work ítems
- Sofware version FPS: 0.42

Circle_3: Kernel

```
__kernel void computeVotingSpace(__global unsigned char* restrict img, int radius, __global int* votes)
{

          const int w = 640;
          const int h = 480;

          int x = get_global_id(0);
          int y = get_global_id(1);
          int c = get_global_id(2);

/*        for (int y = 0;  y < h; y++)
            for (int x = 0;  x < w; x++)
                    for (int c=0; c < 3; c++)
*/
            {
                unsigned char value = getc(img, x, y, w, h, c);

                if (value > MIN_VOTING_THRESHOLD)
                    voteForCircle(votes, x, y, w, h, radius, value);
            }

}
```

Circles_3: Results

- Compilation Early Report:

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/circles_3/reports/report.html

- Compilation Summary

file:///C:/Projects/Cephis/INT_SIE_2018_HPCAdmintech/SourceCode/circles_3/top.fit.summary

- 62% ALMs, 97% BRAMs, clk 127MHz
- We almost exhausted BRAMs
- FPS, Sobel: 30 Hough: 7

Other

Other Interesting Features

- Arbitrary Big Number operations

```
ap_uint<2048> a, b, r;
r= a + b;
```

- Channels
  - Direct communication from kernel to kernel (avoid going to host)
  - Direct communication with host (always-on kernel)

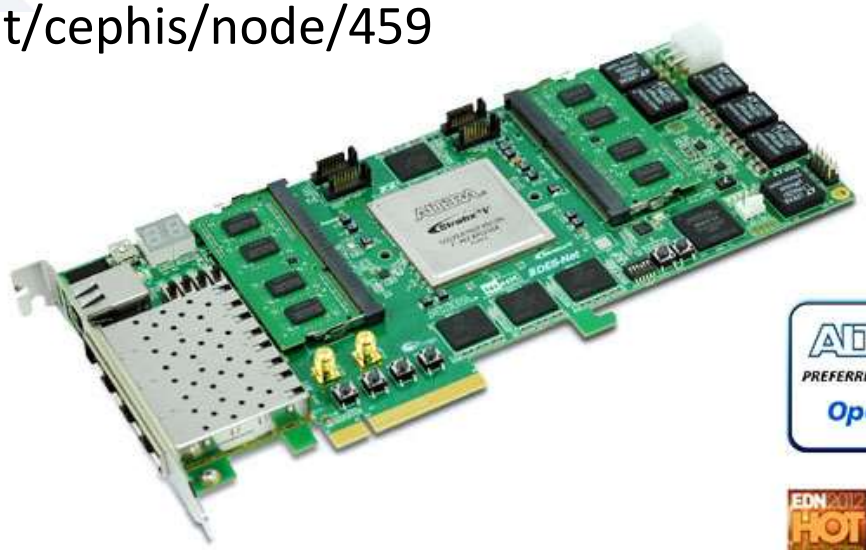
- Custom Hardware blocks
  - Interfacing with High Efficciency HDLs designs

Our current Technology Transfer Projects

- eVoting application
  - using big integers to do encryption algorithms
  - check https://github.com/davidcastells/BigInteger

- Music Identification
  - RETOS call from Ministerio de Economia, Industria y Competitividad
  - http://grupsderecerca.uab.cat/cephis/node/459

GOBIERNO DE ESPAÑA
MINISTERIO DE ECONOMÍA, INDUSTRIA Y COMPETITIVIDAD



ALTERA PREFERRED BOARD FOR OpenCL™

EDN's Hot Products of 2012

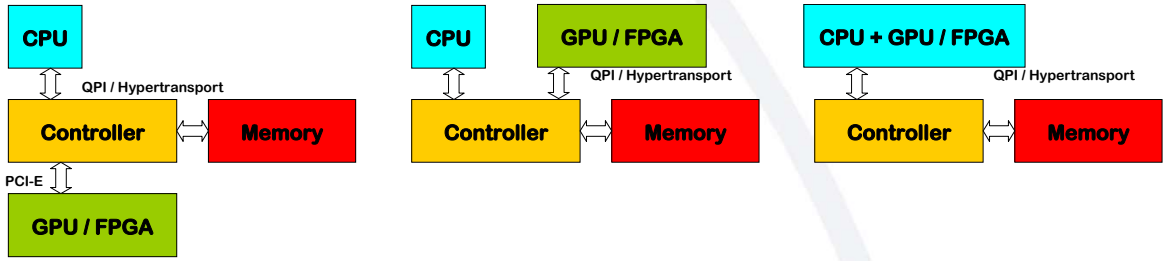
Doing Research using new devices

- Comparing OpenCL toolchain with Many-Soft-Core architectures (OpenMP / MPI)

https://github.com/davidcastells/ocmpi

- OPAE Accelerators (cache coherent with Host)

- Future (?) (reasonable) Roadmap



CEPHIS

Thanks for your attention!

work with us ;-)

Kernel Versions

Version	Details	FPS	ALMs	BRAM s
Sobel2D_1	Pipelined Basic	1.2	22%	28%
Sobel2D_2	Pipelined With Constants (Kernel)	1.2	22%	28%
Sobel2D_3	Pipelined With Restrict	7	21%	25%
Sobel2D_4	ND Range on Y	5-6	24%	24%
Sobel2D_5	ND Range on X	12	23%	23%
Sobel2D_6	ND Range on Y (2 compute Units)	5	37%	24%
Sobel2D_7	ND Range on Y (4 compute Units)	5	65%	49%
Sobel2D_8	Line Buffer	32	27%	18%

Kernel Versions

Version	Details	FPS	ALMs	BRAMs
circles_1	Pipelined Basic	?		
circles_2		Not working	100%	
circles_3	ND Range 3D		61%	97%
circles_4	NR Range 3D with mem			