Webinar III

HPP-Heterogeneous Processing Platform

Computer Vision Using OpenCV/OpenCL Targeting the HPP

Yassine Hariri
Senior Engineer, Platform Design
Hariri@cmc.ca

Hugh W. Pollitt-Smith
Senior System Design Engineer
Pollitt-smith@cmc.ca

13-1-2016

Agenda

- Overview
- emSYSCAN Development Systems Update
- HPP: Heterogeneous processing platform
- Computer Vision challenges and opportunities
- OpenCV
- HPP GPU
  - OpenCV and CUDA
  - Live demo: OpenCV/CUDA targeting the HPP
- HPP FPGA
  - OpenCL
  - Live demo: OpenCL/OpenGL targeting the HPP
- Q&A

EMSYSCAN DEVELOPMENT SYSTEMS UPDATE

Embedded Systems Canada (emSYSCAN)
$54M investment in Canada's National Design Network
37+ universities, 250+ faculty, 5 years

Microsystems Rapid-Prototyping, Characterization and Integration Labs

4 Universities:
- UBC
- U Manitoba
- Queen's
- École Polytechnique

Multi-Technology Design Environment
- System architecture exploration
- Multi-technology simulation
- Design of custom devices for manufacturing

Development Systems
- System validation and proof-of-concept demonstration

License Management Appliance

Installed Design Environment

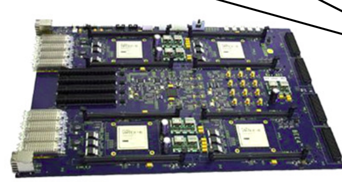
Development System Hardware

Real-Time Embedded Software Lab

University of Waterloo

- Design, analysis, debug of real-time software on next-generation processor systems

Common, Shared Platforms
Interconnected Community of Users
Knowledge Repository
Centralized Management & Operations



License Management Server (LMS)

Development Systems for Proof of Concept

Images courtesy of National Instruments, Xilinx, BEECube,, NVIDIA

National Research Platform: Enriched Projects; Results Sooner

- Common set of programmable research platforms with proof of concept features
- Pooled equipment timeshared among users
- Sharing of knowledge on equipment usage
- Adaptive over time in terms of equipment quantities and equipment features
- Large community of users, institutions
- Leveraged industrial partners (e.g., STMicro.)

National project scope and sizeable outcomes enabled by centralized project implementation and management by CMC Microsystems

Installation and usage

- Shared access systems can be accessed at no charge but require Designer level subscription
  - Subscription provides access to support, tools, reference designs, forums, workshops, travel, select/swap, training, additional discounts
- Systems delivered on site, remote access
- Designated Development System coordinator(s) at each site
  - Communicate institutional needs for purchase specifications
  - Local advocate, information source
  - Encourage participation in National Project

emSYSCAN Development Systems delivered (Gen1)

- Embedded Systems Platform:
  - Xilinx ML605, Altera DE4-530
- Advanced Processing Platform
  - BEEcube BEE3, BEE4, miniBEE
- Software-Defined Radio Platform
  - BEEcube miniBEE, RF daughtercard
- Simulation Acceleration Platform
  - Nallatech P385-D5 (Altera Stratix V, OpenCL)
- Multiprocessor Array Platform
  - NVIDIA Tesla K20 GPU
  - Intel Xeon Phi
- Microsystems Integration Platform
  - National Instruments PXI-based, FPGA, MEMS, microfluidics, RF, photonics features

NDN Development Systems Community

https://community.cmc.ca/community/development-systems

Welcome, Guest Login

Home | Browse | Search

/lore documents in Development Systems

Actions

Development Systems National Catalog

Version 19

created by hugh on Jun 20, 2012 2:30 PM, last modified by hugh on Jan 18, 2013 12:34 PM

The following Development Systems have been delivered to the National Design Network (NDN) for shared access. The designated Coordinator/Contact can provide additional details on availability and how to access:

System Product	Location/University	Quantity	Coordinator/Contact
BEEcube BEE3 (Advanced Processing Platform)			
	CMC Microsystems (online access)	2	Hugh Pollitt-Smith
	McMaster University	1	Dr. Nicola Nicolici
	University of Guelph	1	Dr. Stefano Gregori
	McGill University	1	Dr. Zeljko Zilic
	University of New Brunswick	1	Dr. Kenneth Kent
	University of Saskatchewan	1	Dr. Seok-Bum Ko
	Université du Québec à Chicoutimi	1	Dr. Hung-Tien Bui
	Université du Québec à Trois-Rivières	1	Dr. Adel Omar Dahmane
	Université du Québec à Outaouais	1	Dr. Ahmed Lakhssassi
	University of Windsor	1	Dr. Rashid Rashidzadeh

More Like This

- Introductory setup instructions for the BEEcube miniBEE
- Advanced Processing Platform (BEE3) Design Environment
- Development Systems Inventory
- Advanced Processing Platform
- Setup Information for the HP Z400 workstations (ML605, DE4-530, BEE3)

Upcoming emSYSCAN Development Systems deployments

- Embedded Systems Platform
  - Xilinx Virtex-7, Ultrascale, Zynq options
  - Altera Arria 10 and Arria 10 SoC options
  - Shipping Q1 2016
- Advanced Processing Platform
  - RFP currently in evaluation
  - Shipping Q1/Q2 2016
- Software-defined Radio
  - BEEcube nano/megaBEE (2x2 up to 16x16 MIMO options)
  - Shipping Q1 2016

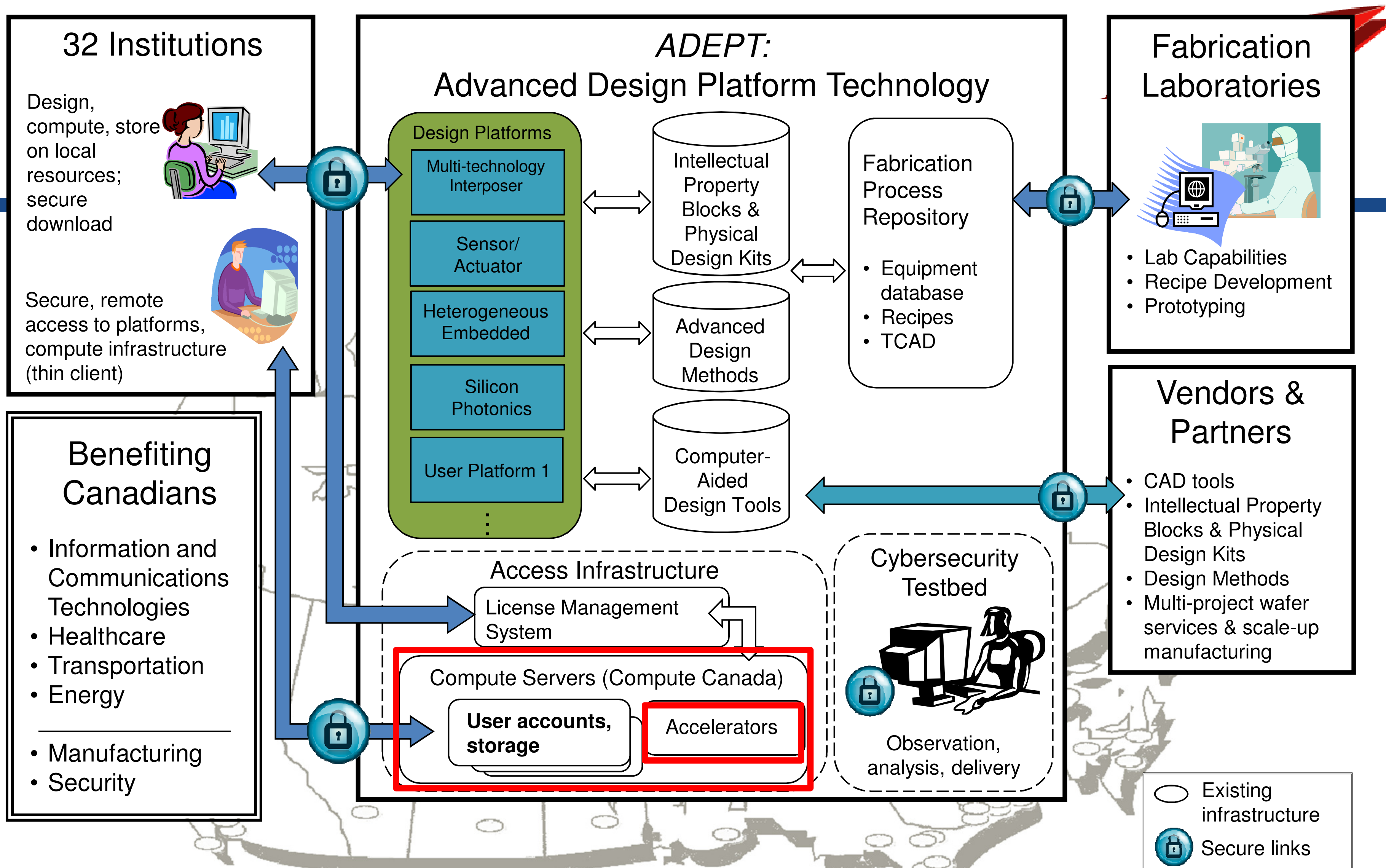
emSYSCAN
Wireless Sensor Network Kits

- For researchers interested in topics related to the Internet of Things, healthcare, smart sensor systems & algorithms & wireless communications.

- Zigbee and Bluetooth based components including:
  - IAR Embedded Workbench for ARM Cortex M
  - Mpression Odyssey MAX 10 FPGA with Bluetooth
  - Freescale Freedom, Beaglebone, Raspberry Pi,
  - CC2538DK, CC2650STK, programmers
  - 10-port USB Industrial Charger (power source)

- Timeline:
  - IAR Embedded Workbench for ARM Cortex-M, dev kits, & programmers delivered Jan 7, 2016 to 14 universities
  - Zigbee connectivity documentation, end of Feb 2016.

- Details at:
  - www.cmc.ca | Products & Services | Development Systems | Sensor Platforms | Wireless Sensor Network Kits



Canada's National Design Network – ADEPT Management & Operations

Includes software procurement, configuration, installation and delivery. Access and utilization management, engineering/technical support. Cybersecurity installations, secure testbed assistance and demonstrations. Train-the-trainer events. Advisory Group coordination. Governance, reporting, legal and financial administration.

HPP Distribution

- Based on Development Systems Coordinator consultations in April 2014:
  - Generation 1 (2014/15): 18 systems
    - USask, UQTR, Outaouais, McGill, York, Windsor, Waterloo, Western, Ottawa, Ryerson, RMC, Victoria
  - Generation 2 (2016/17): 12 systems
    - Memorial, Guelph, McMaster, Toronto, Polytechnique, UQTR, Outaouais

HPP: HETEROGENEOUS PROCESSING PLATFORM

HPP Project status

Status (12 universities selected 18 systems G1, 7 universities selected 8 G2)

- Assembled, cloned, tested and shipped 18/18 units

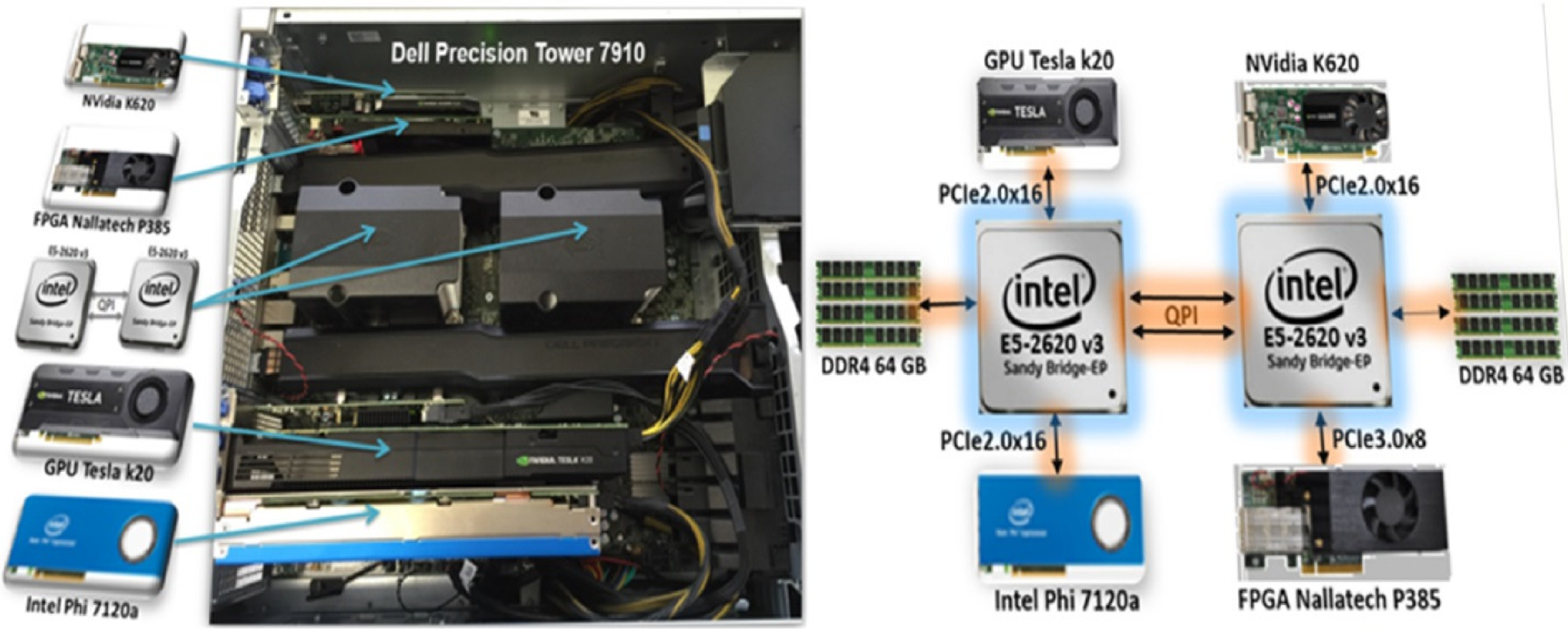
Tutorials and reference designs

- Quick start guide : Heterogeneous Parallel Platform (HPP) **Available online**
- User Guide: Performance and Power profiling for the HPP **Release Date: early Feb. 2016**
- Computer vision using OpenCV/OpenCL targeting the HPP- Heterogeneous Processing Platform **Release Date: early Feb. 2016**

Webinars series for the HPP

- Introduction to the HPP-Heterogeneous Parallel Platform: A combination of Multicores, GPUs, FPGAs and Many-cores accelerators (August 26th) **Available online**
- Programming models, performance and power profiling for the HPP-Heterogeneous Parallel Platform (December 2nd) Available online **Available online**
- Computer vision using OpenCV/OpenCL targeting the HPP- Heterogeneous Processing Platform (January 13th)

HPP fully installed system



Computer Vision

- What is Computer Vision?
  - "**Computer vision** is the transformation of data from a still or video camera into either a decision or a new representation"
  - " **Learning OpenCV: Computer Vision with the OpenCV Library** Paperback – Oct 4 2008 by Gary Bradski (Author), Adrian Kaehler (Author)

- Computer vision is a rapidly growing field for many reasons:
  - High quality Cameras availability and low cost,
  - Increasing processing power,
  - Mature computer vision algorithms.

- Various applications of computer vision, including:
  - objects detection, inspection and tracking, lane tracking and pedestrian detection in automotive, robotic systems, video surveillance, biometrics, augmented reality gaming, new user interfaces, and many more.

Computer Vision challenges and opportunities

Challenges

- Compute-intensive, therefore require high compute capabilities.
- Many computer-vision scenarios must be executed in real-time
  - The processing of a single frame ~30-40 milliseconds.
- In the embedded space, Power consumption is a huge issue

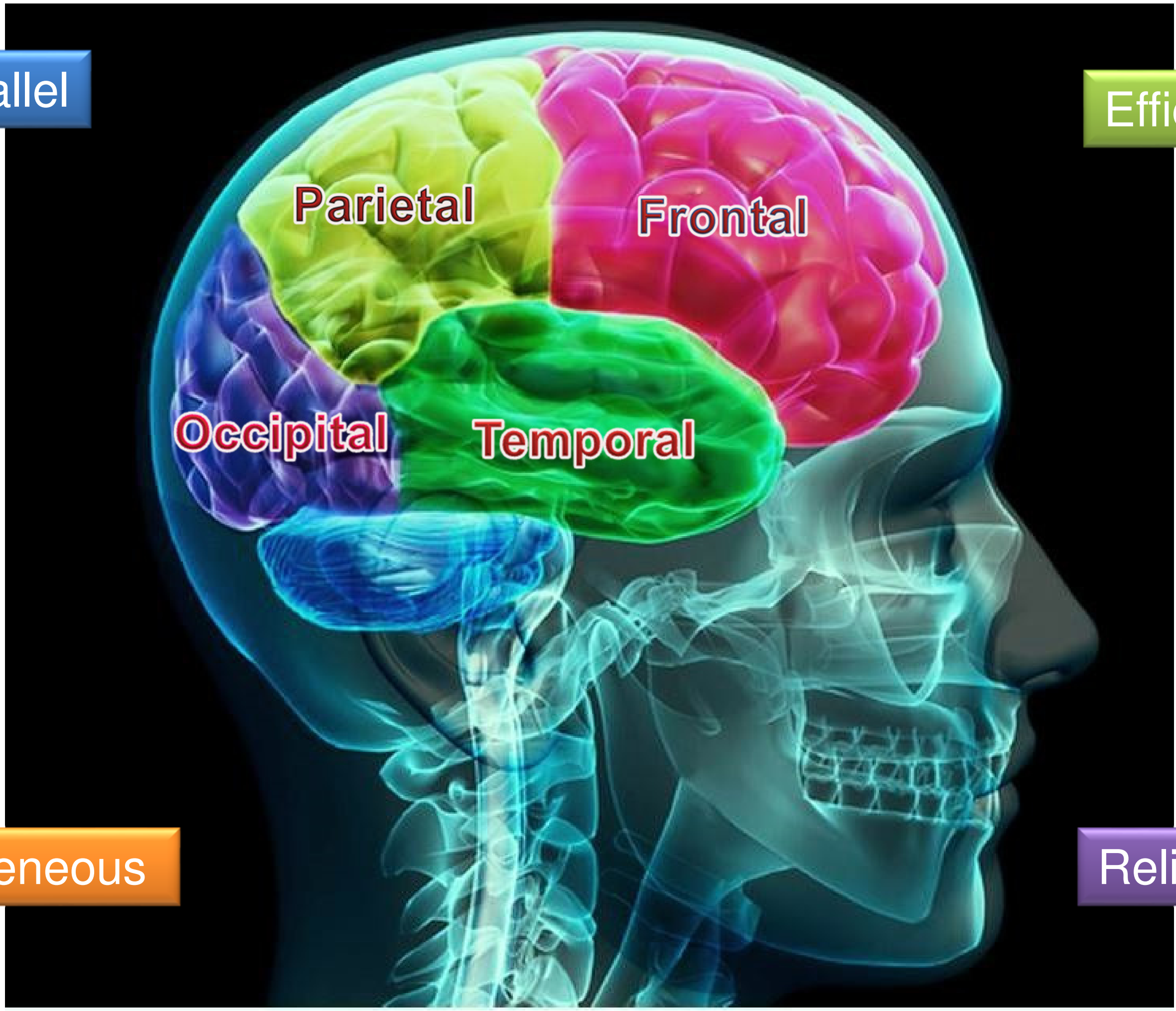
Opportunities

- Build efficient computing architectures
  - More transistors per area,
  - Make a good use of these transistors
    - Parallelization: creating more identical processing units.
    - Specialization: building domain-specific hardware accelerators.
- Heterogeneous parallel computing
  - The concept of combining these two ideas leads to, heterogeneous computing combining many-cores CPUs, GPUs, FPGAs together with various accelerators.

Nature is massively parallel

Parallel

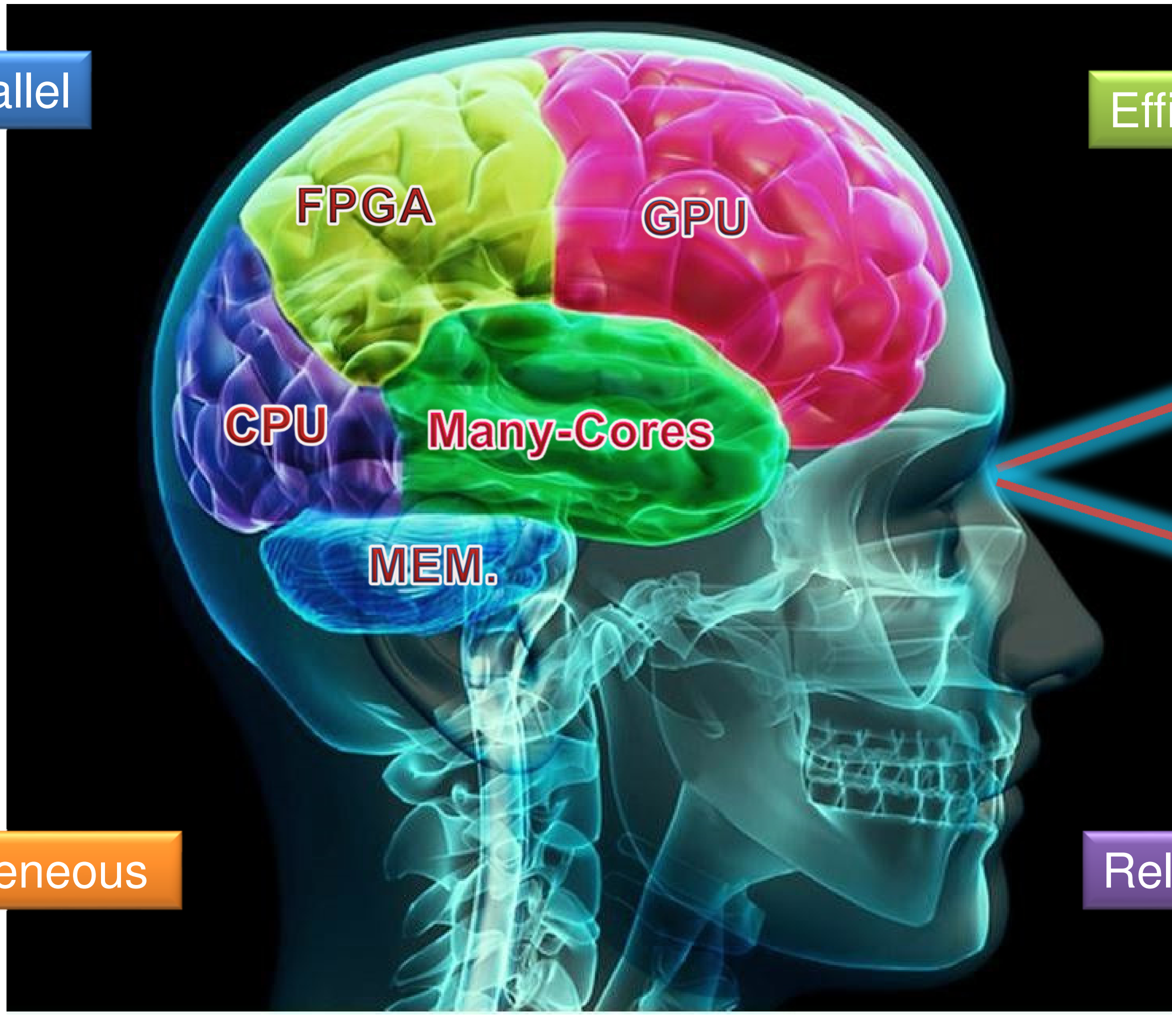
Efficient



Heterogeneous

Reliable

Heterogeneous systems



Parallel

Efficient

Heterogeneous

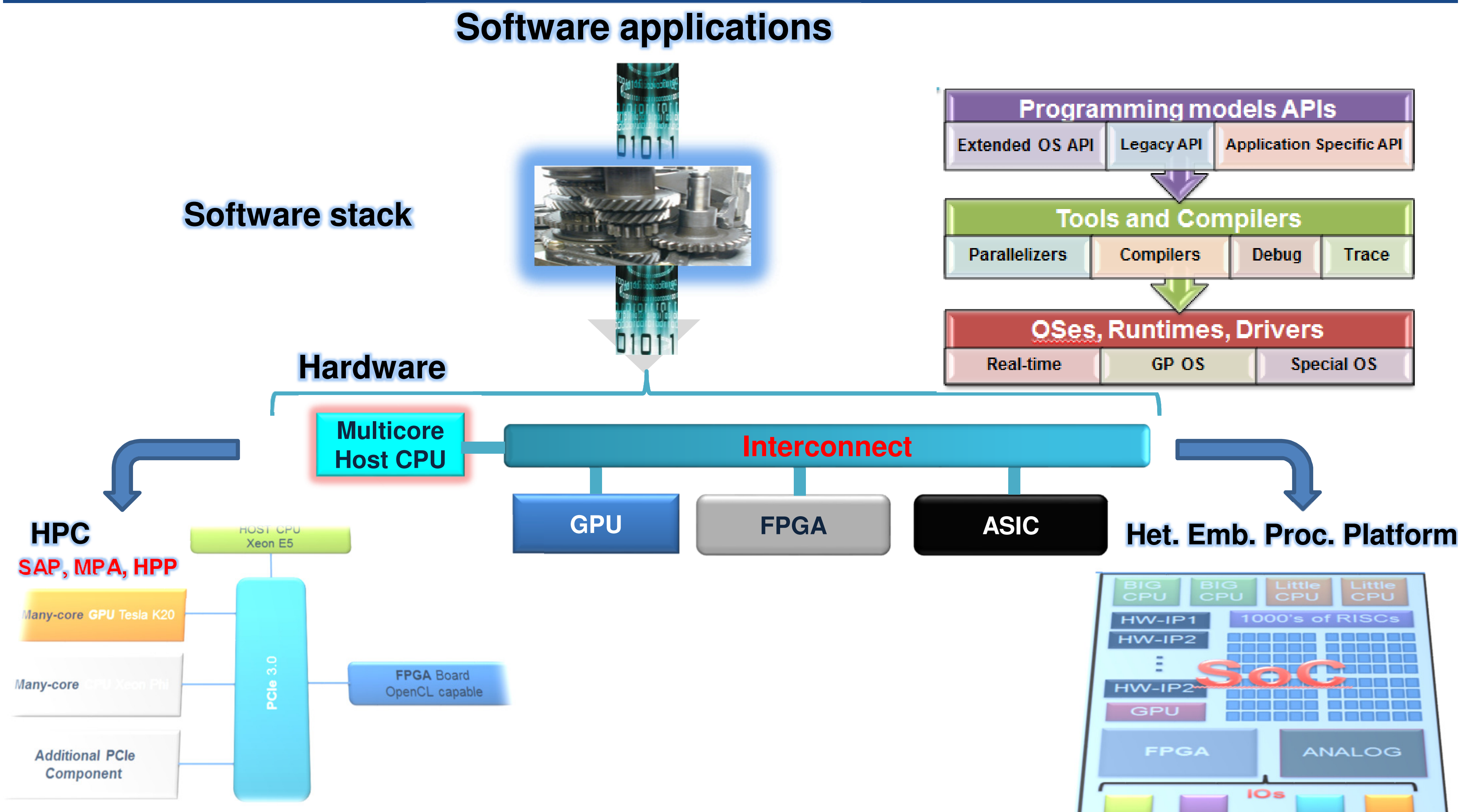
Reliable

Data…
Context…

Computer Vision and Heterogeneous Computing

- Many high-level tasks consist of both parallel and serial subtasks:
  - **Parallel subtasks**: “embarrassingly parallel,” because they are so easy to parallelize
    - **Ex. rendering or filtering pixels.**
  - **Serial subtasks**: do not parallelize easily, as they contain serial segments where the results of the later stages depend on the results of earlier stages.
    - **Ex. Many iterative numerical optimization algorithms, stack-based tree-search algorithms**
- Heterogeneous approach for computer vision applications
  - Run “embarrassingly parallel,” tasks on the GPU or FPGA
  - Run sequential tasks on the multi-core CPU or FPGA
- Key challenges (Research direction)
  - Synchronization between dependent tasks
  - The overhead of moving the data between the different processing units

Heterogeneous Systems Architecture



OpenCV

- Open source library for computer vision, image processing and machine learning
- Permissible BSD license
- Freely available (www.opencv.org)

- **Portability**
- Real-time computer vision (x86 MMX/SSE, ARM NEON, CUDA)
- C (11 years), now C++ (3 years since v2.0), Python and Java
- Windows, OS X, Linux, Android and iOS

Usage

- Usage
  - >6 million downloads, > 47,000 user group
  - Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota

- Applications
  - Street view image stitching
  - Automated inspection and surveillance
  - Robot and driver-less car navigation and control
  - Medical image analysis
  - Video/image search and retrieval
  - Movies - 3D structure from motion
  - Interactive art installations

Functionality

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA and OpenCL GPU acceleration (gpu, ocl)

HPP: GPU
OPENCV + CUDA

Installation of OpenCV on the HPP

- Create a directory for OPENCV
  - mkdir OPENCV
  - cd OPENCV/
- Getting the Cutting-edge OpenCV from the Git Repository
  - git clone https://github.com/Itseez/opencv_contrib.git
  - cd opencv
- Create a temporary directory where you want to put the generated Makefiles, project files as well the object files and output binaries:
  - mkdir release
  - cd release/
- Configuration:
  - cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
- Build:
  - make
  - sudo make install
- Add OpenCV CUDA libraries
  - cmake -D WITH_CUDA=ON -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPEN_GL=ON -D WITH_vtk=ON ..
  - sudo make
  - sudo make install

OpenCV GPU Module

- The GPU module is designed as a host-level API
- Set of classes and functions to achieve the best performance with GPUs
- Implemented using NVIDIA* CUDA* Runtime API and supports only NVIDIA GPUs
- Maintain conceptual consistency with the current CPU functionality
- The OpenCV GPU module includes:
  - Utility functions and Low-level vision primitives:
    - Provide a powerful infrastructure for developing fast vision algorithms taking advantage of GPU
  - High-level algorithms:
    - State-of-the-art algorithms (such as stereo correspondence, face and people detectors, and others).

OpenCV CPU example

```
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{
Mat src, dst;
src = imread("test.jpg", 0);
bilateralFilter(src, dst, -1, 50, 7);
Canny(dst, dst, 0, 30, 3);
imwrite("out.png", dst);
}
```

←OpenCV header files
←OpenCV C++ namespace

←Allocate a temp

←Load an image file as grayscale
←Filter the image
←Find the edges, drawn as white pixels
←Store to an image file

OpenCV CPU example

```
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{
Mat src, dst;
src = imread(“test.jpg”, 0);
bilateralFilter(src, dst, -1, 50, 7);
Canny(dst, dst, 0, 30, 3);
imwrite(“out.png”, dst);
}
```

OpenCV CUDA example

```
#include <opencv2/opencv.hpp>   ←OpenCV GPU header file
#include <opencv2/gpu/gpu.hpp>
using namespace cv;
int main() {
Mat src = imread("test.jpg", 0);
if (!src.data) exit(1);
gpu::GpuMat d_src(src); ←Upload image from CPU to GPU memory
gpu::GpuMat d_dst; ←Allocate a temp output image on the GPU
gpu::bilateralFilter(d_src, d_dst, -1, 50, 7); ←Process images on the GPU
gpu::Canny(d_dst, d_dst, 0, 30, 3); ←Process images on the GPU
Mat dst(d_dst); ←Download image from GPU to CPU mem
imwrite("out.png", dst);
return 0;
}
```

OpenCV CUDA example

```
#include <opencv2/opencv.hpp>
#include <opencv2/gpu/gpu.hpp>
using namespace cv;
int main() {
Mat src = imread("car1080.jpg", 0);
if (!src.data) exit(1);
gpu::GpuMat d_src(src);
gpu::GpuMat d_dst;
gpu::bilateralFilter(d_src, d_dst, -1, 50, 7);
gpu::Canny(d_dst, d_dst, 35, 200, 3);
Mat dst(d_dst);
imwrite("out.png", dst);
return 0;
}
```

GPU-accelerated Computer Vision

http://docs.opencv.org/2.4/modules/gpu/doc/gpu.html

- GPU Module Introduction
- Initalization and Information
- Data Structures
- Operations on Matrices
- Per-element Operations
- Image Processing
- Matrix Reductions
- Object Detection
- Feature Detection and Description
- Image Filtering
- Camera Calibration and 3D Reconstruction
- Video Analysis

Building and running OpenCV CUDA example

- **Source Codes are available in: opencv/samples/gpu**
- **Building and running the hog.cpp**
  1. **Save the original CMakeLists.txt to CMakeLists_original.txt**
  2. **Edit CMakeLists.txt**
  3. **Compile, link and build**
  4. **Run**

```
cmake_minimum_required(VERSION 2.8)
project(HogCode)
find_package(OpenCV REQUIRED)
add_executable(HogCode hog.cpp)
target_link_libraries(HogCode ${OpenCV_LIBS})
```

```
[root@HPPPrototype gpu]# cmake .
-- The C compiler identification is GNU 4.4.7
-- The CXX compiler identification is GNU 4.4.7
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found CUDA: /usr/local/cuda-7.0 (found suitable exact version "7.0")
-- Configuring done
-- Generating done
-- Build files have been written to: /root/OPENCV/opencv/samples/gpu
[root@HPPPrototype gpu]# make
Scanning dependencies of target HogCode
[100%] Building CXX object CMakeFiles/HogCode.dir/hog.cpp.o
Linking CXX executable HogCode
```

```
[root@HPPPrototype gpu]# ./HogCode road.png --camera 0
Device 0:  "Tesla K20c"  4800Mb, sm_35, 2496 cores, Driver/Runtime
ver.7.50/7.0

Controls:
        ESC - exit
        m - change mode GPU <-> CPU
        g - convert image to gray or not
        1/q - increase/decrease HOG scale
        2/w - increase/decrease levels count
        3/e - increase/decrease HOG group threshold
        4/r - increase/decrease hit threshold
```

Live Demo

HPP: GPU

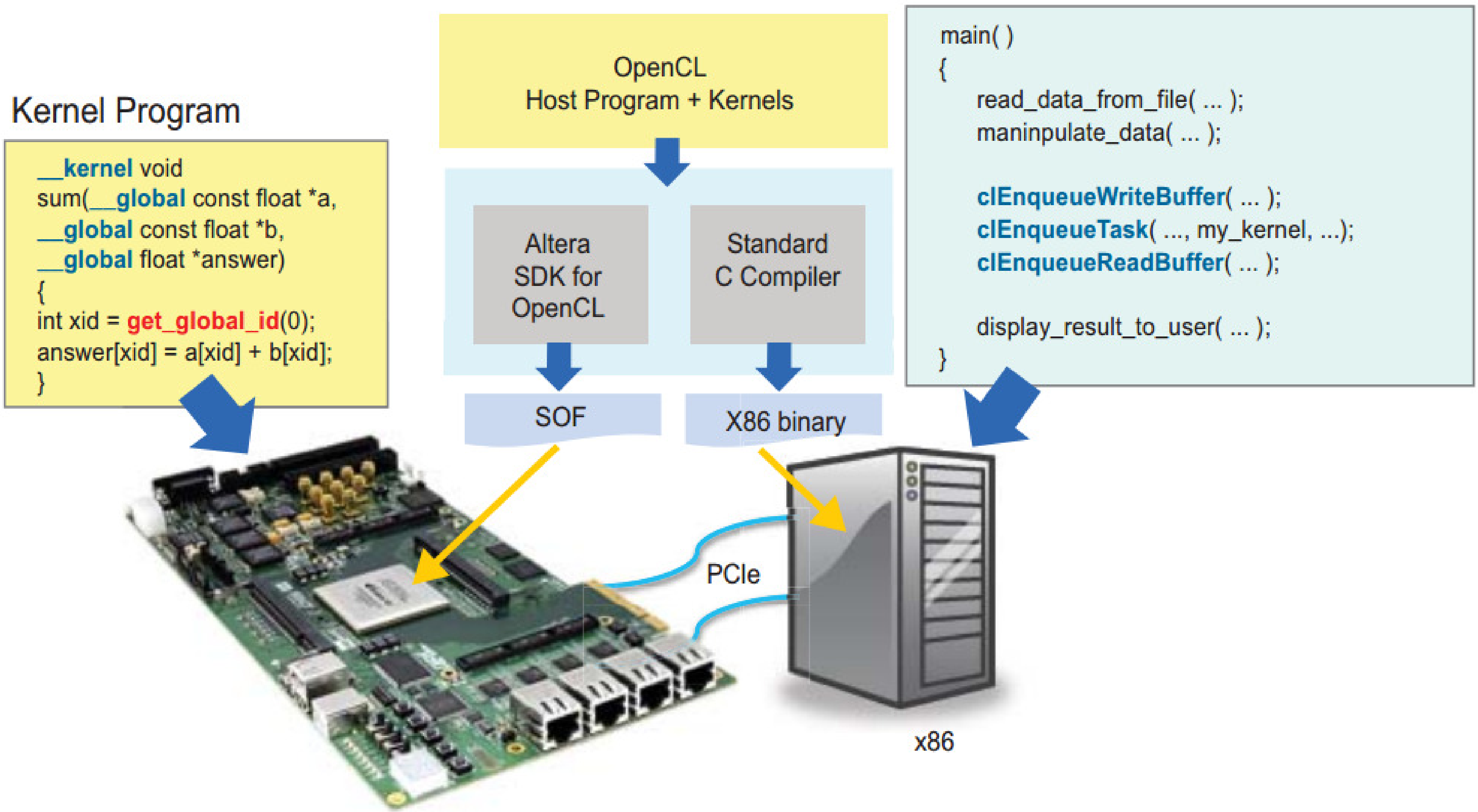
OPENCV + CUDA

HPP: FPGA OPENCL+OPENGL

OpenCL vs. OpenGL

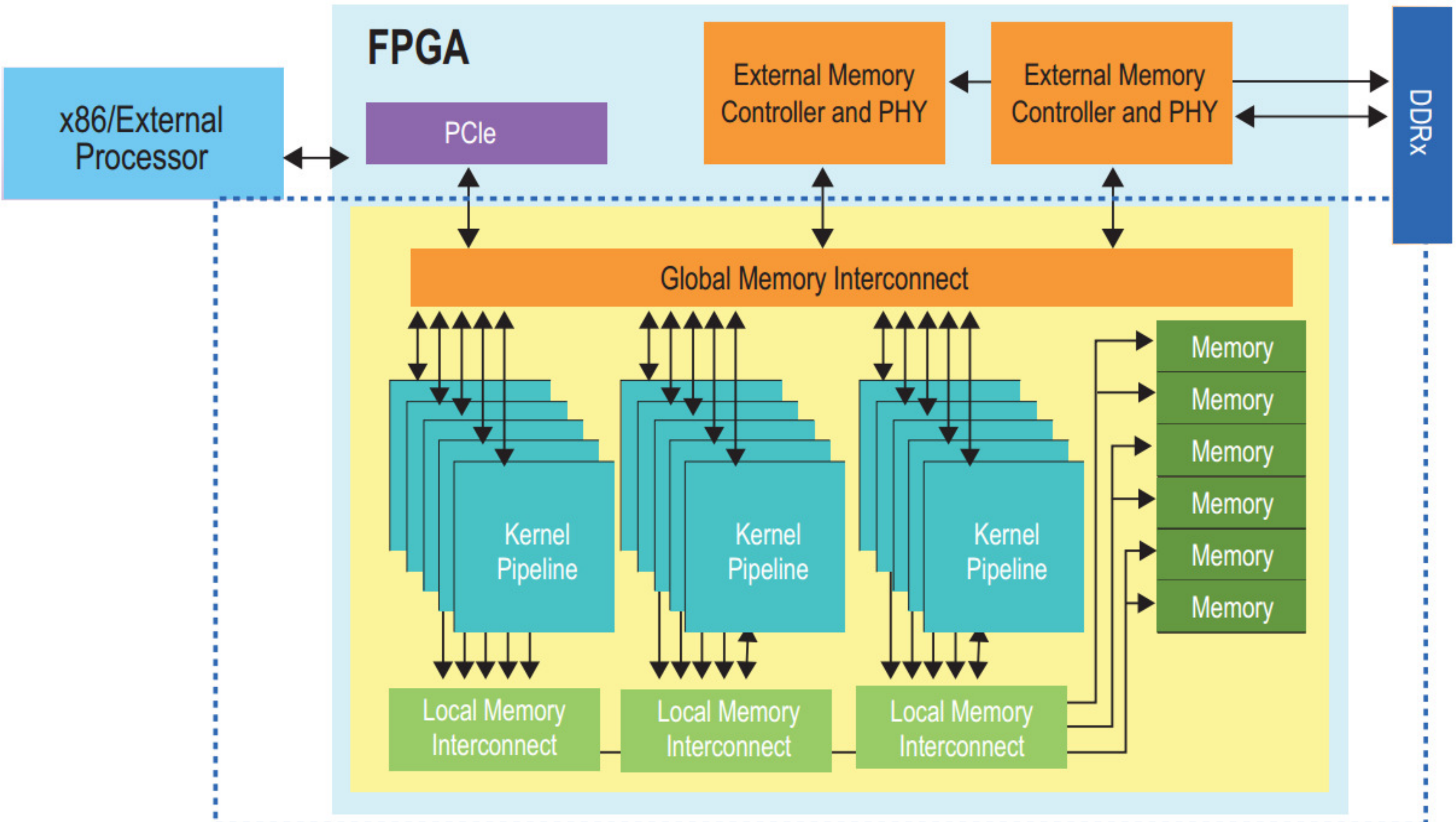
- Unified programming model
  - More accessible to the software developers
  - Host CPU-FPGA communication
  - C based programming language
  - Memory Hierarchy auto generated
  - Potential to integrate existing VHDL/Verilog IP
- Open Graphics Library (OpenGL)
  - Cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.
  - The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.

OpenCL for FPGA



White Paper: Implementing FPGA Design with the OpenCL Standard (Figure 4)

OpenCL Overview



White Paper: Implementing FPGA Design with the OpenCL Standard (Page: 7)

AOCL design flow steps

1. **Intermediate compilation (aoc –c [-g] *<your_kernel_filename>*.cl)**
   - Checks for syntatic errors
   - Generates a **.aoco** file without building the hardware configuration file
   - Generate estimated resource usage summary ***<your_kernel_filename>*.log**
2. **Emulation (aoc -g <your_kernel_filename>.cl)**
   - The AOCL Emulator generates a **.aocx** file that executes on x86-64 Windows or Linux host
   - Assess the functionality of your OpenCL kernel
3. **Profiling (aoc --profile <your_kernel_filename>.cl)**
   - aocl report *<your_kernel_filename>*.aocx profile.mon
   - Instruct the Altera Offline Compiler to instrument performance counters in the Verilog code in the **.aocx** file
   - During execution, the performance counters collect performance information which you can then review in the Profiler GUI.
4. **Full deployment**
   - Execute the .aocx file on the FPGA

Sobel Filter Design Example

```
int main(int argc, char **argv)
{
    imageFilename = (argc > 1) ? argv[1] : "butterflies.ppm";
    initGL(argc, argv);
    initCL();
    input = (cl_uint*)alignedMalloc(sizeof(unsigned int) * rows * cols);
    output = (cl_uint*)alignedMalloc(sizeof(unsigned int) * rows * cols);
   // Read the image
    if (!parse_ppm(imageFilename.c_str(), cols, rows, (unsigned char *)input)) {
        std::cerr << "Error: could not load " << argv[1] << std::endl;
        teardown();
    }
    std::cout << "Commands:" << std::endl;
    std::cout << " <space>  Toggle filter on or off" << std::endl;
    std::cout << " -" << std::endl          << "    Reduce filter threshold" << std::endl;
    std::cout << " +" << std::endl          << "    Increase filter threshold" << std::endl;
    std::cout << " =" << std::endl          << "    Reset filter threshold to default" << std::endl;
    std::cout << " q/<enter>/<esc>" << std::endl          << "    Quit the program" << std::endl;
    glutMainLoop();
    teardown(0);
}
```

Initialize OpenGL

Initialize OpenCL

Allocate Buffers on the HOST

Parsing the image

How to control de app.

GLUT event processing loop

Free the resources

initGL

```
void keyboard(unsigned char key, int, int)
{    switch (key) {
        case ' ':
            useFilter = !useFilter;
            break;
        case '=':
            thresh = 128;
            break;
        case '-':
            thresh = std::max(thresh - 10, 16u);
            break; ***  }}
```

```
void initGL(int argc, char **argv)
{
   ***
   glutWindowHandle = glutCreateWindow("Filter");
***
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutIdleFunc(idle);
   glutTimerFunc(REFRESH_DELAY, timerEvent, 0);
***
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   if (useFilter) {
      filter(output);
      glDrawPixels(cols, rows, GL_RGBA, GL_UNSIGNED_BYTE, output
   } else {
      glDrawPixels(cols, rows, GL_RGBA, GL_UNSIGNED_BYTE, input);
   }

 ***
}
```

initCL

```
void initCL()
{
    platform = findPlatform("Altera");
    status = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, NULL);
    clGetDeviceInfo(device, CL_DEVICE_NAME, sizeof(info), info, NULL);
    context = clCreateContext(0, num_devices, &device, NULL, NULL, &status);
    queue = clCreateCommandQueue(context, device, 0, &status);
    std::string binary_file = getBoardBinaryFile("sobel", device);
    std::cout << "Using AOCX: " << binary_file << "\n";
    program = createProgramFromBinary(context, binary_file.c_str(), &device, 1);
    status = clBuildProgram(program, num_devices, &device, "", NULL, NULL);
    kernel = clCreateKernel(program, "sobel", &status);
    in_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY, sizeof(unsigned int) * rows * cols, NU
    out_buffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(unsigned int) * rows * cols, NULL, &status);
    int pixels = cols * rows;
    status  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &in_buffer);
    status |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &out_buffer);
    status |= clSetKernelArg(kernel, 2, sizeof(int), &pixels);
    checkError(status, "Error: could not set sobel args");
}
```

Initialize OpenCL

Create Command Queue

Create Kernel

Create Buffers

Setup Kernel Args.

filter

```
void filter(unsigned int *output)
{
    ***
    status = clEnqueueWriteBuffer(queue, in_buffer, CL_FALSE, 0, sizeof(unsigned int) * rows * cols, input, 0, NULL, NULL);
    ***
    status = clSetKernelArg(kernel, 3, sizeof(unsigned int), &thresh);
    ***
    status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &sobelSize, &sobelSize, 0, NULL, &event);
    ***
    status  = clFinish(queue);
    ***
clReleaseEvent(event);
    status = clEnqueueReadBuffer(queue, out_buffer, CL_FALSE, 0, sizeof(unsigned int) * rows * cols, output, 0, NULL, NULL);
    checkError(status, "Error: could not copy data from device");
    status = clFinish(queue);
    checkError(status, "Error: could not successfully finish copy");
}
```

copy data into device

set sobel threshold

enqueue sobel filter

Blocks until queued OpenCL commands are issued and have completed.

copy data from device

Live Demo

HPP: FPGA
OPENCL+OPENGL

Q&A

Yassine Hariri
Hariri@cmc.ca