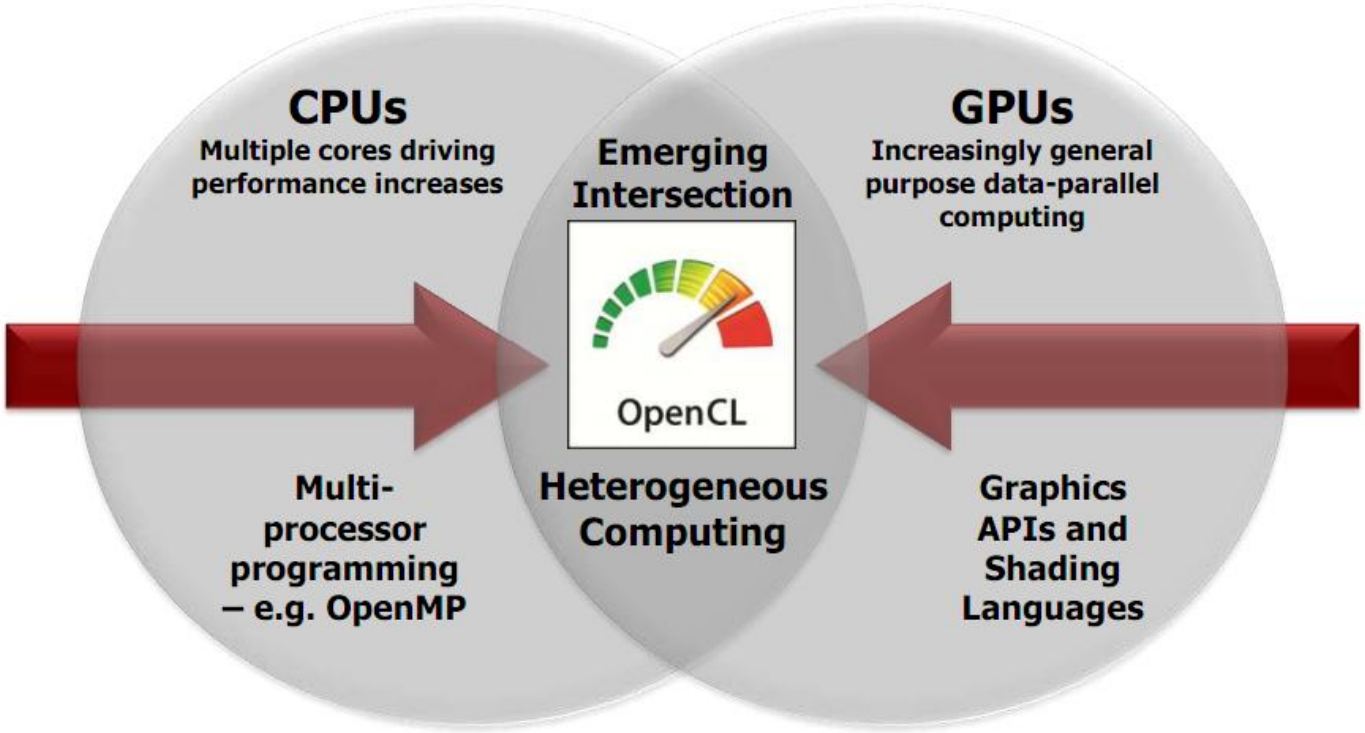
22nd Summer School on PARALLEL COMPUTING

Introduction to OpenCL

Piero Lanucara – p.lanucara@cineca.it
SuperComputing Applications and Innovation Department

CINECA

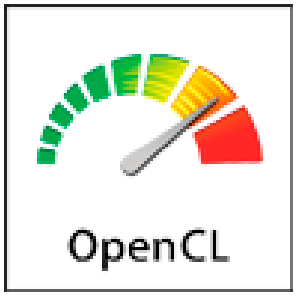
Processor Parallelism



OpenCL is a programming framework for heterogeneous compute resources

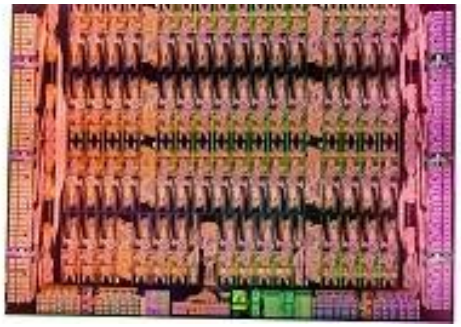
OpenCL

- *O*pen *C*ompute *L*anguage
- For heterogeneous parallel-computing systems
- Cross-platform
  - Implementations for
    - ATI GPUs
    - NVIDIA GPUs
    - x86 CPUs
    - Many others…..



More on Multi-Platform Targeting

- Targets a broader range of CPU-like and GPU-like devices than CUDA
  - Targets devices produced by multiple vendors



  - Many features of OpenCL are optional and may not be supported on all devices
- OpenCL codes must be prepared to deal with much greater hardware diversity
- A single OpenCL kernel will likely not achieve peak performance on all device types

OpenCL

- Standardized
- Initiated by Apple
- OpenCL 1.0 2008.
- Released with Max OS 10.6 (Snow Leopard)
- Most recent: OpenCL 1.2 Nov 2011
- Release for multicore ARM CPUs Nov 2012
- Developed by the Khronos Group

KHRONOS GROUP

CONNECTING SOFTWARE TO SILICON

OpenCL Working Group

- **Diverse industry participation**
  - Processor vendors, system OEMs, middleware vendors, application developers
- **Many industry-leading experts involved in OpenCL's design**
  - A healthy diversity of industry perspectives
- **Apple made initial proposal and is very active in the working group**
  - Serving as specification editor

OpenCL

- API very close to OpenGL
- Based on the C language
- Easy transition from CUDA
- Other languages:

❑ C++ wrapper API (is built on top of OpenCL C API)

➢ C++ API divided into a number of classes with corresponding OpenCL types

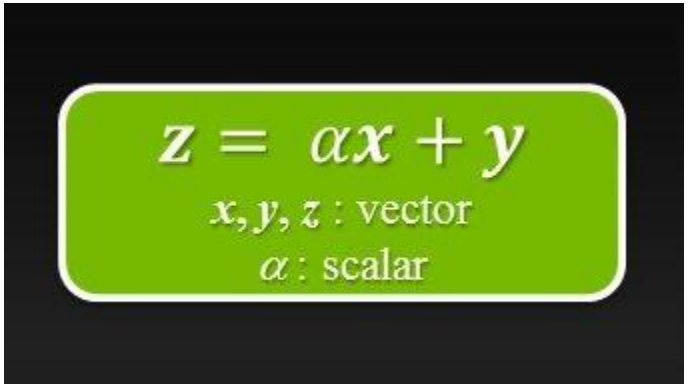
➢ for example, ***cl::memory*** maps to OpenCL type ***cl_mem***

➢ When possible, C++ inheritance and abstraction is given

❑ Fortran90 (FortranCL), Python (PyOpenCL) bindings, …

❑ Libraries and Tools (ViennaCL,…)

OpenCL implementation of BLAS SAXPY



Just to illustrate OpenCL commands

Add two vectors, x and y to produce z

x and y transferred to device (GPU)

Result, z, returned to host (CPU)

OpenCL code

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>

#ifdef __APPLE__
        #include "OpenCL/opencl.h"
#else
        #include "CL/cl.h"
#endif

std::string GetPlatformName (cl_platform_id id)
{
        size_t size = 0;
        clGetPlatformInfo (id, CL_PLATFORM_NAME, 0, NULL, &size);

        std::string result;
        result.resize (size);
        clGetPlatformInfo (id, CL_PLATFORM_NAME, size,
                        const_cast<char*> (result.data ()), NULL);

        return result;
}

std::string GetDeviceName (cl_device_id id)
{
        size_t size = 0;
        clGetDeviceInfo (id, CL_DEVICE_NAME, 0, NULL, &size);

        std::string result;
        result.resize (size);
        clGetDeviceInfo (id, CL_DEVICE_NAME, size,
                        const_cast<char*> (result.data ()), NULL);

        return result;
}
```

```
void CheckError (cl_int error)
{
        if (error != CL_SUCCESS) {
                std::cerr << "OpenCL call failed with error " << error <<
std::endl;


}

std::string LoadKernel (const char* name)
{
        std::ifstream in (name);
        std::string result (
                (std::istreambuf_iterator<char> (in)),
                std::istreambuf_iterator<char> ());
        return result;
}

cl_program CreateProgram (const std::string& source,
        cl_context context)
{
        size_t lengths [1] = { source.size () };
        const char* sources [1] = { source.data () };

        cl_int error = 0;
        cl_program program = clCreateProgramWithSource (context, 1, sources, lengths,
&error);
        CheckError (error);

        return program;
}
```

```
int main ()
{
                        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clGetPlatformIDs.html
                        cl_uint platformIdCount = 0;
                        clGetPlatformIDs (0, NULL, &platformIdCount);

                        if (platformIdCount == 0) {
                                        std::cerr << "No OpenCL platform found" <<
std::endl;
                                        return 1;
                        } else {
                                        std::cout << "Found " << platformIdCount << "
platform(s)" << std::endl;
                        }

                        std::vector<cl_platform_id> platformIds (platformIdCount);
                        clGetPlatformIDs (platformIdCount, platformIds.data(),NULL);

                        for (cl_uint i = 0; i < platformIdCount; ++i) {
                                        std::cout << "\t (" << (i+1) << ") : " <<
GetPlatformName (platformIds [i]) << std::endl;
                        }

                        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clGetDeviceIDs.html
                        cl_uint deviceIdCount = 0;
                        clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL, 0, NULL,
                                        &deviceIdCount);

                        if (deviceIdCount == 0) {
                                        std::cerr << "No OpenCL devices found" << std::endl;
                                        return 1;
                        } else {
                                        std::cout << "Found " << deviceIdCount << " device(s)"
<< std::endl;
                        }
```

```
std::vector<cl_device_id> deviceIds (deviceIdCount);
                        clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL,
deviceIdCount,
                                        deviceIds.data (), NULL);

                        for (cl_uint i = 0; i < deviceIdCount; ++i) {
                                        std::cout << "\t (" << (i+1) << ") : " <<
GetDeviceName (deviceIds [i]) << std::endl;
                        }

                        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clCreateContext.html
                        const cl_context_properties contextProperties [] =
                        {
                                        CL_CONTEXT_PLATFORM,
reinterpret_cast<cl_context_properties> (platformIds [0]),
                                        0, 0
                        };

                        cl_int error = CL_SUCCESS;
                        cl_context context = clCreateContext (contextProperties,
deviceIdCount,
                                        deviceIds.data (), NULL, NULL, &error);
                        CheckError (error);

                        std::cout << "Context created" << std::endl;
```

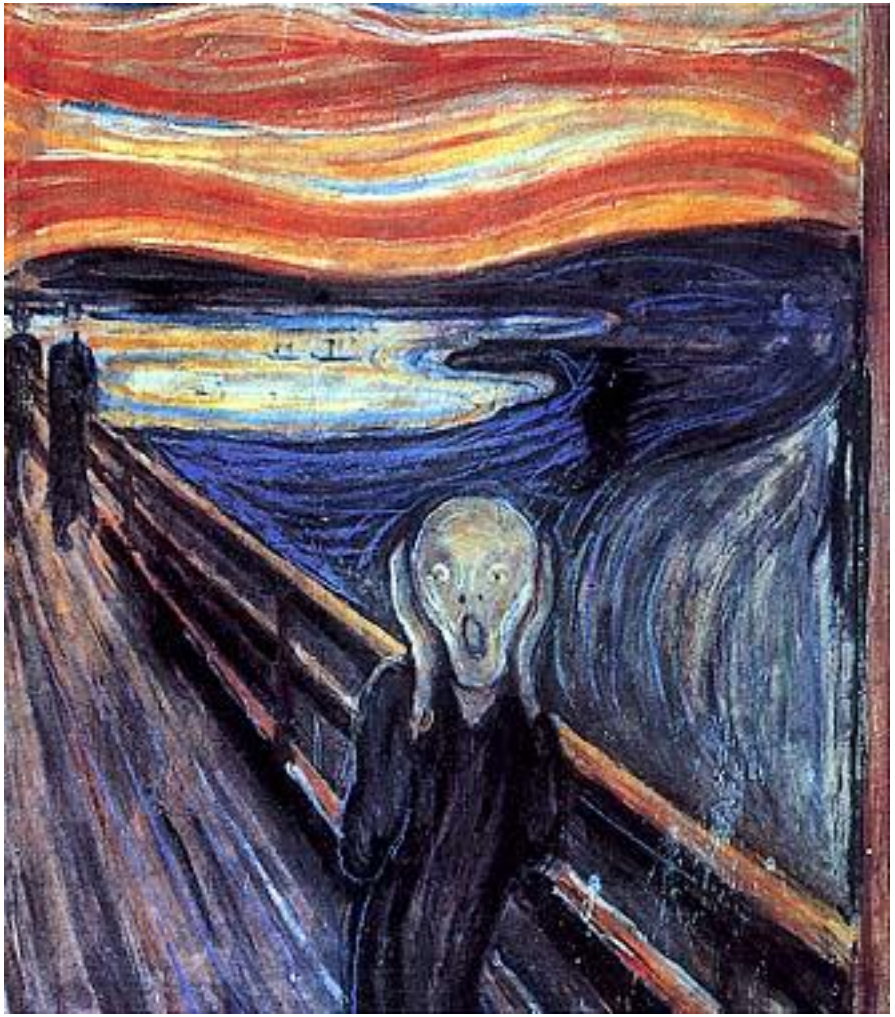
```
cl_program program = CreateProgram (LoadKernel ("kernels/saxpy.cl"),
                                    context);

        CheckError (clBuildProgram (program, deviceIdCount, deviceIds.data (),
NULL, NULL, NULL));

        cl_kernel kernel = clCreateKernel (program, "SAXPY", &error);
        CheckError (error);

        // Prepare some test data
        static const size_t testDataSize = 1 << 3;
        std::vector<float> a (testDataSize), b (testDataSize);
        for (int i = 0; i < testDataSize; ++i) {
                a [i] = static_cast<float> (23 + i);
                b [i] = static_cast<float> (42 + i);
        }

        cl_mem aBuffer = clCreateBuffer (context, CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR,
                sizeof (float) * (testDataSize),
                a.data (), &error);
        CheckError (error);

        cl_mem bBuffer = clCreateBuffer (context, CL_MEM_READ_WRITE |
CL_MEM_COPY_HOST_PTR,
                sizeof (float) * (testDataSize),
                b.data (), &error);
        CheckError (error);


        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clCreateCommandQueue.ht
ml
        cl_command_queue queue = clCreateCommandQueue (context, deviceIds
[1],
                0, &error);
        CheckError (error);
```

```
clSetKernelArg (kernel, 0, sizeof (cl_mem),  &aBuffer);
        clSetKernelArg (kernel, 1, sizeof (cl_mem),  &bBuffer);
        static const float two = 2.0f;
        clSetKernelArg (kernel, 2, sizeof (float), &two);

        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clEnqueueNDRangeKernel
.html
        const size_t globalWorkSize [] = { testDataSize, 0, 0 };
        CheckError (clEnqueueNDRangeKernel (queue, kernel, 1,
                NULL,
                globalWorkSize,
                NULL,
                0, NULL, NULL));

        // Get the results back to the host
        //
http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/clEnqueueReadBuffer.htm
l
        CheckError (clEnqueueReadBuffer (queue, bBuffer, CL_TRUE, 0,
                sizeof (float) * testDataSize,
                b.data (),
                0, NULL, NULL));
        // Check the results
        for (int i = 0; i < testDataSize; ++i) {
                std::cout<< b[i]<<std::endl;
        }

        clReleaseCommandQueue (queue);

        clReleaseMemObject (bBuffer);
        clReleaseMemObject (aBuffer);

        clReleaseKernel (kernel);
        clReleaseProgram (program);

        clReleaseContext (context);
```

```
__kernel void SAXPY (__global float* x, __global float* y, float a)
{
        const int i = get_global_id (0);

        y [i] += a * x [i];
}
```

OpenCL

- OpenCL relies on API which is rather verbose and cumbersome to use.
- Please, be patient….



Structure of OpenCL program

1. Get information about platform and devices available on system
2. Select devices to use - context
3. Create an OpenCL command queue
4. Create memory buffers on device
5. Transfer data from host to device memory buffers
6. Create kernel program object
7. Build (compile) kernel in-line (or load precompiled binary)
8. Create OpenCL kernel object
9. Set kernel arguments
10. Execute kernel
11. Read kernel memory and copy to host memory.

Platform and devices

"The host plus a collection of devices managed by the OpenCL framework that allow an application to share resources and execute kernels on devices in the platform."
Platforms represented by a *cl_platform_id* object, initialized with clGetPlatformIDs()

```
cl_uint platformIdCount = 0;
clGetPlatformIDs (0, NULL, &platformIdCount);

if (platformIdCount == 0) {
        std::cerr << "No OpenCL platform found" << std::endl;
        return 1;
} else {
        std::cout << "Found " << platformIdCount << "
platform(s)" << std::endl;
}

std::vector<cl_platform_id> platformIds (platformIdCount);
clGetPlatformIDs (platformIdCount, platformIds.data(),NULL);

for (cl_uint i = 0; i < platformIdCount; ++i) {
        std::cout << "\t (" << (i+1) << ") : " <<
GetPlatformName (platformIds [i]) << std::endl;
}
```

cl_uint defined in OpenCL API

Platform and devices

Device(s) represented by a *cl_device_id* object, initialized with clGetDeviceIDs()

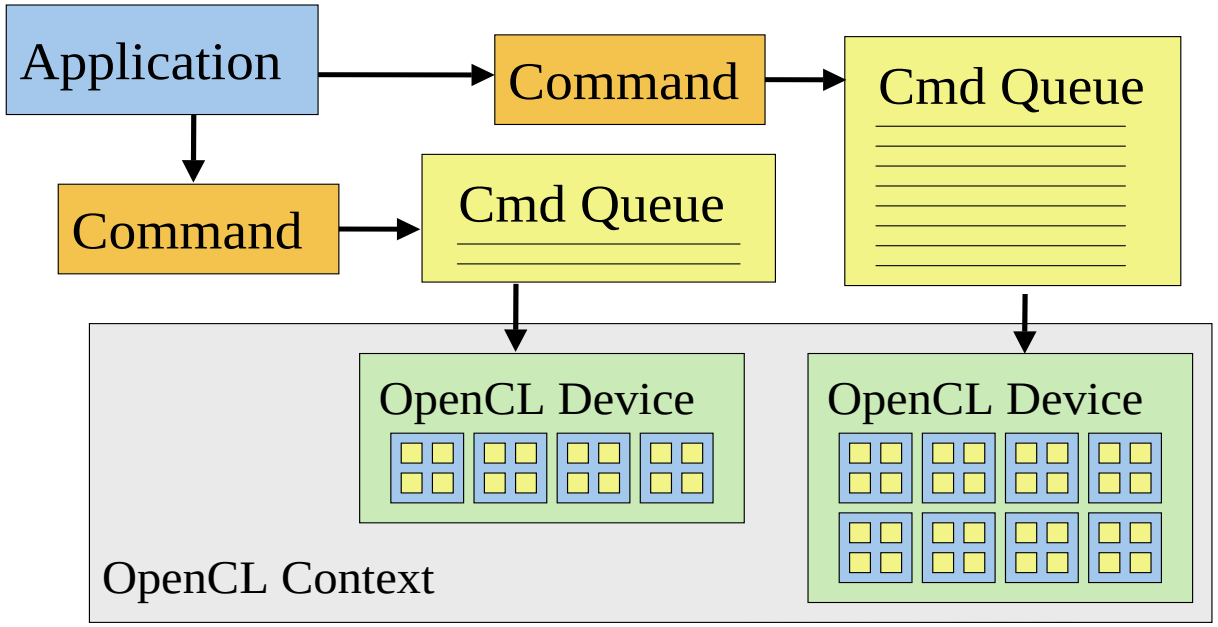
```
cl_uint deviceIdCount = 0;
        clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL, 0, NULL,
                &deviceIdCount);

        if (deviceIdCount == 0) {
                std::cerr << "No OpenCL devices found" << std::endl;
                return 1;
        } else {
                std::cout << "Found " << deviceIdCount << " device(s)" <<
std::endl;
        }

        std::vector<cl_device_id> deviceIds (deviceIdCount);
        clGetDeviceIDs (platformIds [0], CL_DEVICE_TYPE_ALL, deviceIdCount,
                deviceIds.data (), NULL);

        for (cl_uint i = 0; i < deviceIdCount; ++i) {
                std::cout << "\t (" << (i+1) << ") : " << GetDeviceName
(deviceIds [i]) << std::endl;
        }
```

CL_DEVICE_TYPE_ALL is equivalent to all OpenCL devices in the system

Context

"The environment within which the kernels execute and the domain in which synchronization and memory management is defined.

The context includes a set of devices, the memory accessible to those devices, the corresponding memory properties and one or more command-queues used to schedule execution of a kernel(s) or operations on memory objects."



Context

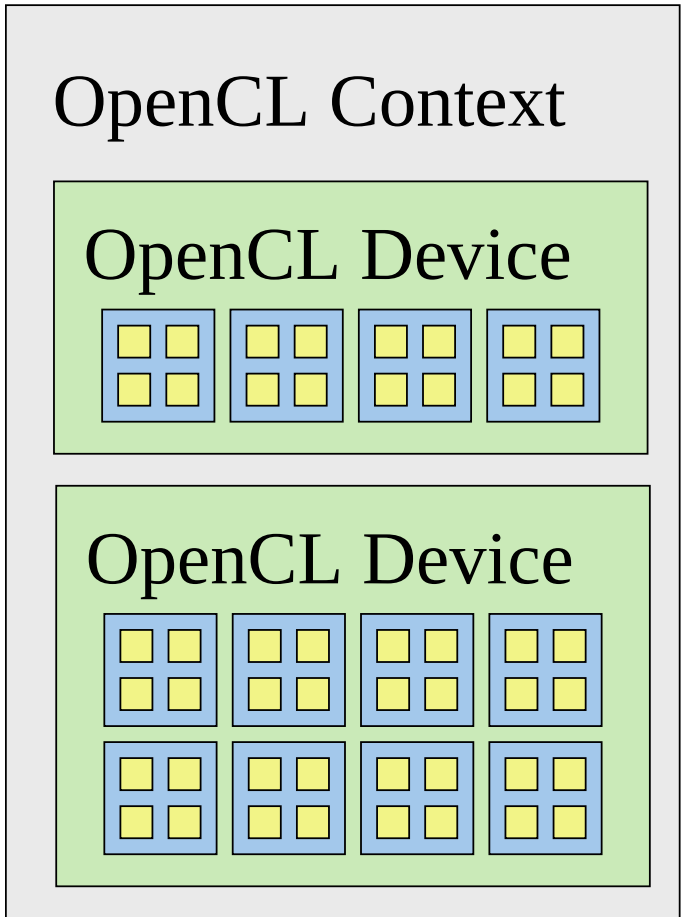
Context represented by a *cl_context* object, initialized with clCreateContext()

```
const cl_context_properties contextProperties [] =
                {
                                CL_CONTEXT_PLATFORM,
reinterpret_cast<cl_context_properties> (platformIds [0]),
                                0, 0
                };

                cl_int error = CL_SUCCESS;
                cl_context context = clCreateContext (contextProperties, deviceIdCount,
                                deviceIds.data (), NULL, NULL, &error);
                CheckError (error);

                std::cout << "Context created" << std::endl;
```

CL_CONTEXT_PLATFORM is a property setted in order to specify the platform to use.

Context again

- Contains one or more devices
- OpenCL memory objects associated with a context, not a specific device
- clCreateBuffer() is the main data object allocation function
  - error if an allocation is too large for any device in the context
- Each device needs its own work / command queue(s)
- Memory transfers are associated with a command queue (thus a specific device)



Command Queue

"An object that holds commands that will be executed on a specific device.

The command-queue is created on a specific device in a context.

Commands to a command-queue are queued in-order but may be executed in-order or out-of-order. ..."

Command queue represented by a *cl_command_queue* object, initialized with clCreateCommandQueue()

```
cl_command_queue queue = clCreateCommandQueue (context, deviceIds[1],
                0, &error);
CheckError (error);
```

Allocating memory on device

clCreateBuffer() allocation function

```
// Prepare some test data
static const size_t testDataSize = 1 << 3;
std::vector<float> a (testDataSize), b (testDataSize);
for (int i = 0; i < testDataSize; ++i) {
        a [i] = static_cast<float>  (23 + i);
        b [i] = static_cast<float>  (42 + i);
}

cl_mem aBuffer = clCreateBuffer (context, CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR,
        sizeof (float) * (testDataSize),
        a.data (), &error);
CheckError (error);

cl_mem bBuffer = clCreateBuffer (context, CL_MEM_READ_WRITE |
CL_MEM_COPY_HOST_PTR,
        sizeof (float) * (testDataSize),
        b.data (), &error);
CheckError (error);
```

CL_MEM_READ_ONLY *flag is used to specify a read only memory object within a kernel*

CL_MEM_COPY_HOST_PTR *flag is used to specify that the application wants to allocate memory for the memory oject and copy data from memory referenced by* host_ptr *pointer*

Kernel Program

Simple programs might be in the same file as the host code (in that case need to formed into strings in a character array).

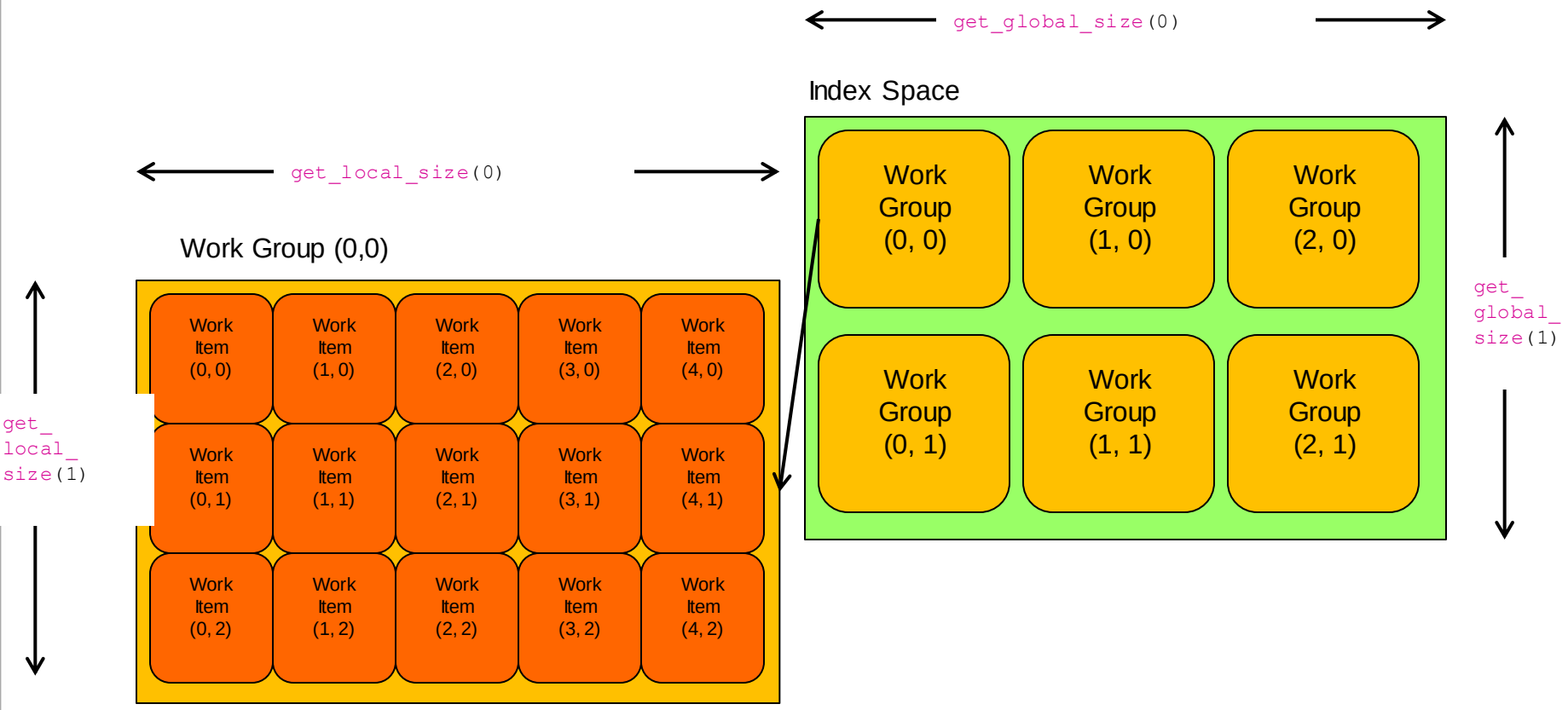
If in a separate file, can read that file into host program as a character string

```
__kernel void SAXPY (__global float* x, __global float* y, float
a)
{
        const int i = get_global_id (0);

        y [i] += a * x [i];
}
```

OpenCL Kernel

- Parallel work is submitted to devices by launching kernels
- Kernels run over global dimension index ranges (NDRange), broken up into "work groups", and "work items"
- Work items executing within the same work group can synchronize with each other using barriers or memory fences
- Work items in different work groups can only sync with each other by launching a new kernel

OpenCL NDRange Configuration

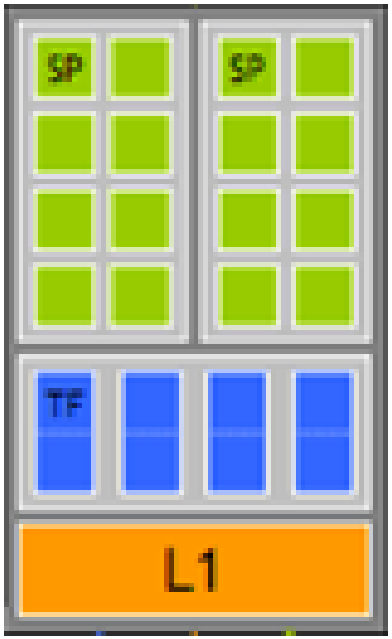


OpenCL and CUDA

- Many OpenCL features have a one to one mapping to CUDA features
- OpenCL
  - More complex platform and device management
  - More complex kernel launch
  - More lower level than CUDA

OpenCL and CUDA

- *Compute Unit* (CU) correspond to
  - CUDA streaming multiprocessors (SMs)
  - CPU core
  - etc.
- *Processing Element* correspond to
  - CUDA streaming processor (SP)
  - CPU ALU



OpenCL and CUDA

- *Work Item* (CUDA *thread*) – executes kernel code
- *Index Space* (CUDA *grid*) – defines work items and how data is mapped to them
- *Work Group* (CUDA *block*) – work items in a work group can synchronize

OpenCL and CUDA

- OpenCL: each thread has a unique global index
  - Retrieve with get_global_id()

```
__kernel void SAXPY (__global float* x, __global float* y, float
a)
{
        const int i = get_global_id (0);

        y [i] += a * x [i];
}
```

Mapping OpenCL indices

OpenCL API call	Explanation	CUDA equivalent
get_global_id(0);	Global index of the work item in the *x*-dimension	blockIdx.x×blockDim.x+threadIdx.x
get_local_id(0)	Local index of the work item within the work group in the *x*-dimension	threadIdx.x
get_global_size(0);	Size of NDRange in the *x*-dimension	gridDim.x ×blockDim.x
get_local_size(0);	Size of each work group in the *x*-dimension	blockDim.x

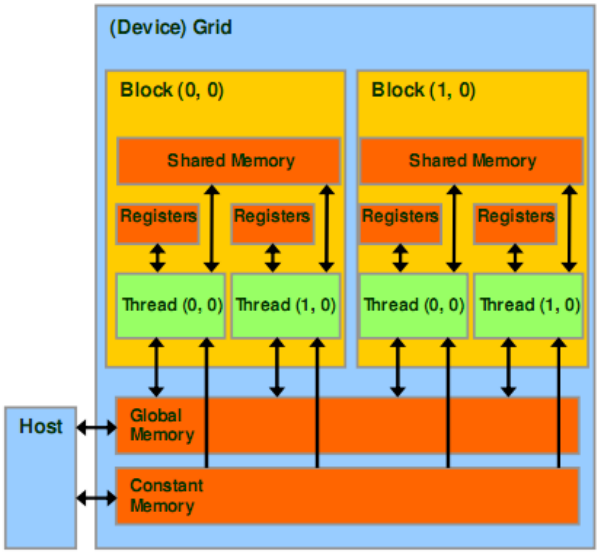
OpenCL Memory System

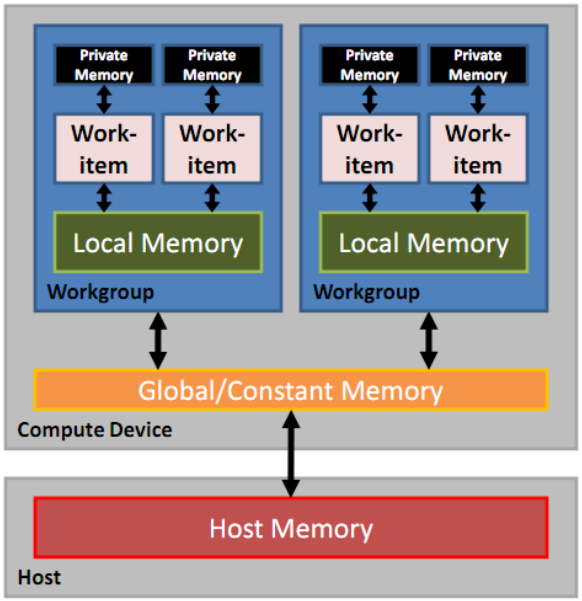
- __global – large, long latency
- __private – on-chip device registers
- __local – memory accessible from multiple PEs or work items
  - May be SRAM or DRAM, must query...
- __constant – read-only constant cache
- Programmer manages device memory explicitly

OpenCL Memory Types	CUDA Equivalent
global memory	global memory
constant memory	constant memory
local memory	shared memory
private memory	Local memory

OpenCL Memory System

CUDA	OpenCL
Global memory	Global memory
Constant memory	Constant memory
Shared memory	Local memory
Local memory	Private memory





Create OpenCL Program Object

Create program object

```
cl_program CreateProgram (const std::string& source,
                cl_context context)
{
                size_t lengths [1] = { source.size () };
                const char* sources [1] = { source.data  () };

                cl_int error = 0;
                cl_program program = clCreateProgramWithSource (context, 1, sources,
lengths, &error);
                CheckError (error);

                return program;
}

cl_program  program = CreateProgram (LoadKernel ("kernels/saxpy.cl"),
                                context);
```

clCreateProgramWithSource load the source code specified by the text string (saxpy.cl)

Create OpenCL Kernel Object(s)

Build (compiles and link) kernel program

```
CheckError (clBuildProgram (program, deviceIdCount, deviceIds.data (), NULL, NULL,
NULL));
```

Preprocessor, optimization options here

Creating kernel objects

```
cl_kernel kernel = clCreateKernel (program, "SAXPY", &error);
                   CheckError (error);
```

Set kernel arguments

```
clSetKernelArg (kernel, 0, sizeof (cl_mem), &aBuffer);
clSetKernelArg (kernel, 1, sizeof (cl_mem), &bBuffer);
static const float two = 2.0f;
clSetKernelArg (kernel, 2, sizeof (float), &two);
```

Enqueue and Copy back

Enqueue a command to execute kernel on device

```
const size_t globalWorkSize [] = { testDataSize, 0, 0 };
        CheckError (clEnqueueNDRangeKernel (queue, kernel, 1,
                        NULL,
                        globalWorkSize,
                        NULL,
                        0, NULL, NULL));
```

NDRange=1 is setted globalWorksize is equal to testDataSize setted in main program

Copy results from buffer object to host memory

```
// Get the results back to the host
        CheckError (clEnqueueReadBuffer (queue, bBuffer, CL_TRUE, 0,
                        sizeof (float) * testDataSize,
                        b.data (),
                        0, NULL, NULL));
        // Check the results
        for (int i = 0; i < testDataSize; ++i) {
                        std::cout<< b[i]<<std::endl;
        }
```

CL_TRUE imply blocking read

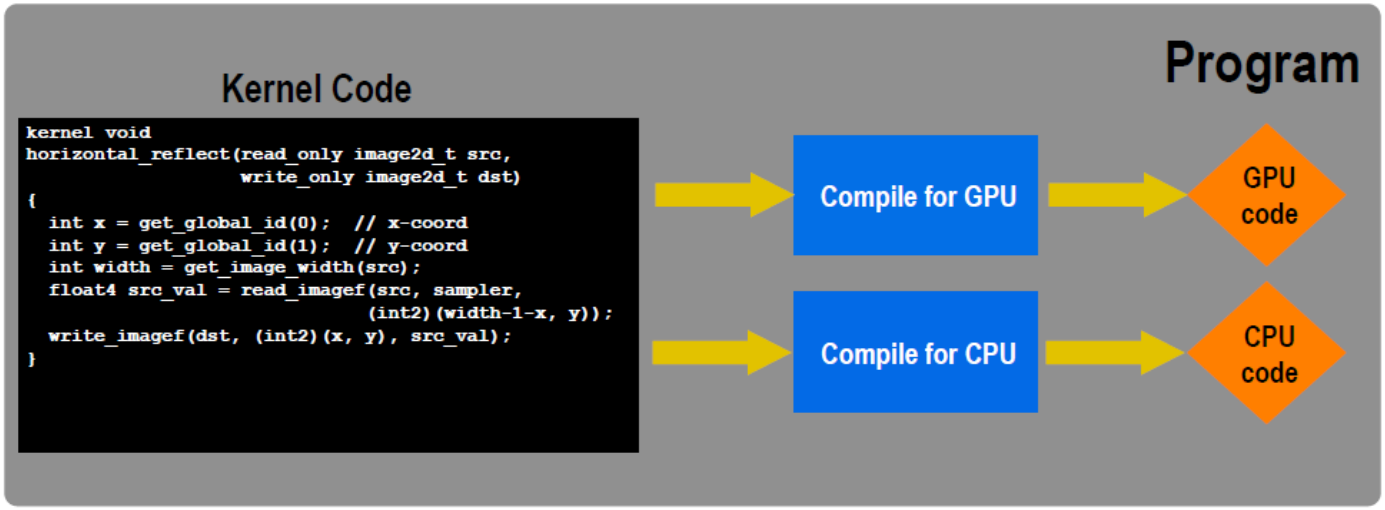
Clean-up

```
clReleaseCommandQueue (queue);

            clReleaseMemObject (bBuffer);
            clReleaseMemObject (aBuffer);

            clReleaseKernel (kernel);
            clReleaseProgram (program);

            clReleaseContext (context);
```

Running OpenCL code on Multiple Devices

Executing Code

- Programs build executable code for multiple devices
- Execute the same code on different devices



Compiling

Need OpenCL header:

```
#include <CL/cl.h>
```

(For mac: `#include <OpenCL/opencl.h>`)

and link to the OpenCL library.

Compile OpenCL host program main.cpp using g++,
two phases:

```
g++ -DNVIDIA –I/opt/cuda/include -c main.c -o main.o
g++ -DNVIDIA -L /usr/lib64/nvidia -l OpenCL main.o -o host
```

Running environment

NVIDIA Tesla K20C

- 13 Multiprocessors
- 2496 CUDA Cores
- 5 GB of global memory
- GPU clock rate 760MHz

NVIDIA GeForce GTS 450

- 4 Multiprocessors
- 192 CUDA Cores
- 1 GB of global memory
- GPU clock rate 800 MHz

Running

```
./main.exe
Found 1 platform(s)
            (1) : NVIDIA CUDA
Found 2 device(s)
            (1) : Tesla K20c
            (2) : GeForce GTS 450
Context created
88
91
94
97
100
103
106
109
```

NVIDIA CUDA platform found and 2 devices (K20C and GeForce)

GeForce device was selected

Results are OK no matter what performances

- Targets a broader rang of CPU- nd GPU-like devices th n CUDA
  - Targets endors
  - M ay not b
- OpenC th much s
- A sin t achieve peak p pes

Performance????

OpenCL Performance

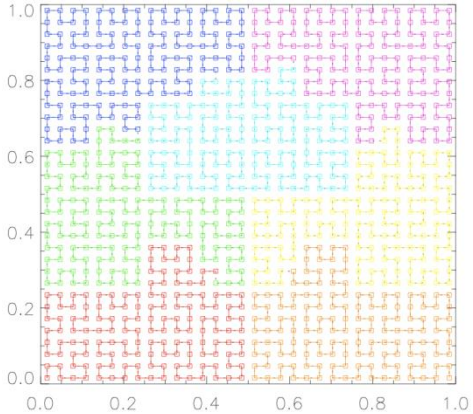
- OpenCL is a portable tool to a great many hardware (multicore CPUs, NVIDIA, AMD, Intel MIC, SoC,…)
- Tremendous amount of compute power

1170 GFLOPs peak

- One OpenCL code to rule them all?
- Performance portability is the real problem
- Q: how much performance do we lose by running an OpenCL application?

The Hydro benchmark

Hydro is a simplified version of RAMSES (CEA, France astrophysics code to study large scale structure and galaxy formation)



Hydro main features:

- regular cartesian mesh (no AMR)
- solves compressible Euler equations of hydrodynamics
- finite volume method, second order Godunov scheme
- it uses a Riemann solver numerical flux at the interfaces

The Hydro benchmark

Hydro is about 1K lines of code and has been ported to different programming environment and architectures, including accelerators. In particular:

- initial Fortran branch including OpenMP, MPI, hybrid MPI+OpenMP
- C branch for CUDA, OpenCL, OpenACC, UPC

The Hydro OpenCL benchmark

- Major problems in the Hydro OpenCL development:
  - ❑ how to write OpenCL kernels (**more or less trivial to implement**)
  - ❑ how to setting up OpenCL context and device(s), realize the necessary data movement operations, dispatching work to the device(s),… (**very difficult, error prone**)
- OpenCL rely on API which is verbose and cumbersome to use

The Hydro OpenCL benchmark

- OpenCL rely on API which is verbose and cumbersome to use
- **Solution: Hydro uses ad-hoc macro and functions to allow more readable and clear coding.**
- **Also useful for tuning (Hydro can potentially achieve even more OpenCL performances)**

Hydro OpenCL high level funcs and macros

```
double
oclLaunchKernel2D(cl_kernel k, cl_command_queue q, int
nbobjx, int nbobjy, int nbthread, const char *fname, const
int line)
{
  cl_int err = 0;
  dim3 gws, lws;
  cl_event event;
  double elapsk;
  int maxThreads = 0;
  cl_uint one = 1;
  cl_device_id dId = oclGetDeviceOfCQueue(q);
  size_t prefsz = 32;

  maxThreads = oclGetMaxWorkSize(k, dId);
  // printf("%d ", maxThreads);
  maxThreads = MIN(maxThreads, nbthread);
  // printf("%d ", nbthread);
```

```
oclLaunchKernel2D(ker[Loop1KcuRiemann], cqueue, Hnxyt,
slices, THREADSSZ, __FILE__, __LINE__);
```

```
gws[2] = lws[2] = 0;
  gws[1] = lws[1] = 1;
  gws[0] = lws[0] = 1;
  //
  lws[0] = maxThreads;
  // lws[0] /= 2; lws[1] *= 2;
  gws[0] = oclMultiple(nbobjx, lws[0]);
  gws[1] = oclMultiple(nbobjy, lws[1]);

  printf("Launch2D: %ld G:%ld %ld %ld L:%ld %ld %ld\n",
nbobjx * nbobjy, gws[0], gws[1], gws[2], lws[0], lws[1],
lws[2]);
  err = clEnqueueNDRangeKernel(q, k, NDR_2D, NULL, gws,
lws, 0, NULL, &event);
  oclCheckErrF(err, "clEnqueueNDRangeKernel", fname, line);
...
err = clWaitForEvents(one, &event);
  oclCheckErrF(err, "clWaitForEvents", fname, line);

  elapsk = oclChronoElaps(event);

  err = clReleaseEvent(event);
  oclCheckErrF(err, "clReleaseEvent", fname, line);

  return elapsk;
}
```

Coming Next: Hydro live@CINECA

Hydro live@Eurora

- Eurora CINECA-Eurotech prototype
- 1 rack
- Two Intel SandyBridge and two NVIDIA K20 cards per node
- Hot water cooling
- Energy efficiency record (up to 3150 MFLOPs/w sustained)

Hydro OpenCL tuning

OpenCL run, single NVIDIA Tesla K20 device,4091x4091

domain,100

iterations

performances are parameter independent. This fact may change using other devices

Size of the matrices	Elapsed time (sec.) without initialization	EfficiencyLoss (with respect to the best timing)
64	43.87	0.042
128	42.09	0
256	42.49	0.009
512	42.63	0.012
1024	43.79	0.040
2048	44.27	0.051

performances tuning with work group size (threads-per-block) parameter

Hydro run comparison

OpenAcc run it fails using Pgi compiler

performances of OpenCL code are very good (better than CUDA!)

More than 16 Intel Xeon SandyBridge cores are needed to compare OpenCL 1 K20S device

Intel MIC preliminary run on CINECA prototype. 240 threads, vectorized code,KMP_AFFINITY=balanced

Device/version	Elapsed time (sec.) without initialization	EfficiencyLoss (with respect to the best timing)
CUDA K20C	52.37	0.24
OpenCL K20C	42.09	0
MPI (1 process)	780.8	17.5
MPI+OpenMP (16 OpenMP threads)	109.7	1.60
MPI+OpenMP MIC (240 threads)	147.5	2.50
OpenACC (Pgi)	N.A.	N.A.

Hydro OpenCL scaling

performances are good. Scalability is limited by domain size

OpenCL+MPI run, varying the number of NVIDIA Tesla K20 device,4091x4091 domain,100 iterations

Number of K20 devices	Elapsed time (sec.) without initialization	Speed-Up
1	42.0	1.0
2	23.5	1.7
4	12.2	3.4
8	8.56	4.9
16	5.70	7.3

Conclusions and future perspectives

- OpenCL is an open standard, which is targeting for portability on heterogeneous devices (GPUs, AMD, Intel MIC,…)
- To reach its full potential, however, OpenCL needs to deliver Performance portability.
- OpenCL gives users the chance to efficiently use native SIMD engines (like vector units) of CPUs, accelerators, ….

❑ OpenCL will help you use existing and future devices increasingly provide tremendous computing power at a reduced energy-requirement and price

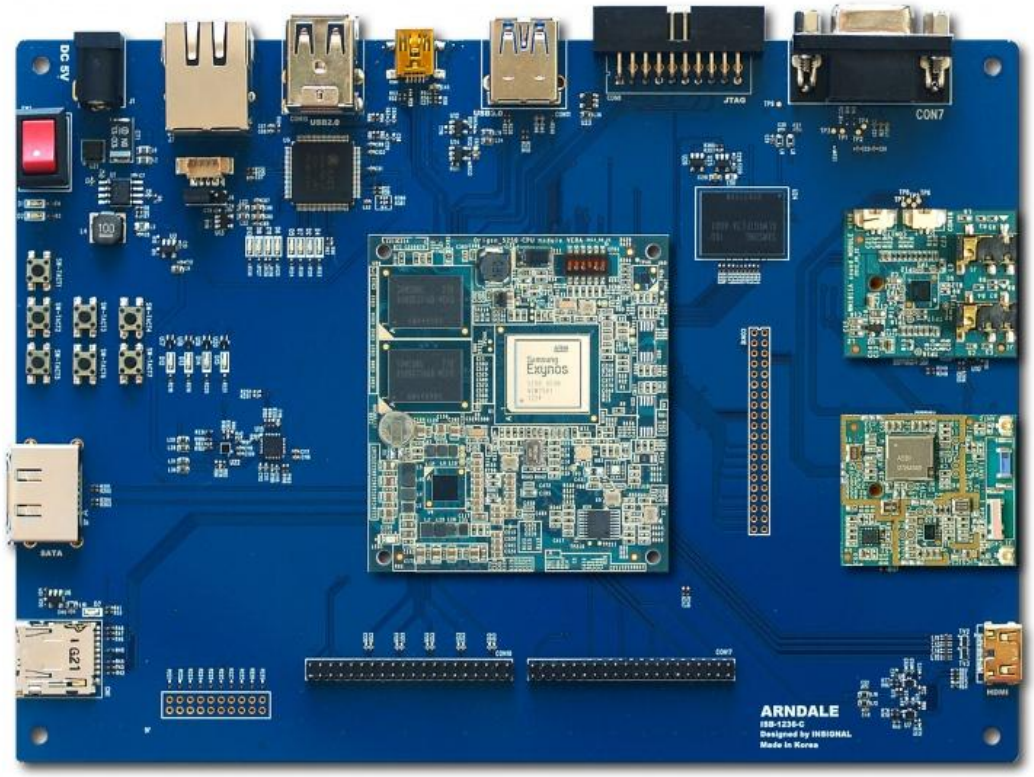
- **OpenCL can be considered as an emerging standard for the so-called Heterogeneous Computing needed to achieve Exascale sustained performance in the next years**

OpenCL for exascale

- Exynos 5 Dual
- ❑ 1.7 GHz Dual-core ARM Cortex A15 with Neon SIMD
- ❑ Integrated ARM Mali T604 GPU (4 cores)@533 MHz
- ❑ 72GFLOPs peak (DP)
- ❑ **GPGPU computing via OpenCL (OmpSs runtime)**

- European exascale project
- Leverage commodity and embedded power-efficient technology (**Cineca partner**)

System on Chip (SoC)



MONT-BLANC

References

OpenCL home page

http://www.khronos.org/opencl/

One OpenCL to Rule Them All

Romain Dolbeau, Francois Bodin, Guillame Colin de Verdière

http://www.caps-entreprise.com/wp-content/uploads/2012/08/One-OpenCL-to-rule-them-all.pdf

Ramses Project

Romain Teissyer, Pierre-Franois Lavallée, et al.

http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_sstechnique.php?id_ast=904