Introduction to OpenCL

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

CINECA

Heterogeneous High Performance Computing

Green500 list

Top of the list is dominated by heterogeneous supercomputers. In particular:

- **The number one Eurora, which is located at CINECA, wins with more than 3 gigaflops/watt**.
  - different heterogeneous architectures: CPUs, GPUs, AMD FirePro, **Intel MIC**….

Heterogeneous High Performance Programming framework

- http://www.hpcwire.com/hpcwire/2012-02-28/opencl_gains_ground_on_cuda.html

HPC wire

"As the two major programming frameworks for GPU computing, OpenCL and CUDA have been competing for mindshare in the developer community for the past few years. Until recently, CUDA has attracted most of the attention from developers, especially in the high performance computing realm. But OpenCL software has now matured to the point where HPC practitioners are taking a second look.

Both OpenCL and CUDA provide a general-purpose model for data parallelism as well as low-level access to hardware, but only OpenCL provides an open, industry-standard framework. As such, it has garnered support from nearly all processor manufacturers including AMD, Intel, and NVIDIA, as well as others that serve the mobile and embedded computing markets. As a result, applications developed in OpenCL are now portable across a variety of GPUs and CPUs."

Processor Parallelism

CPUs
Multiple cores driving performance increases

Emerging Intersection

OpenCL

GPUs
Increasingly general purpose data-parallel computing

Multi-processor programming – e.g. OpenMP

Heterogeneous Computing

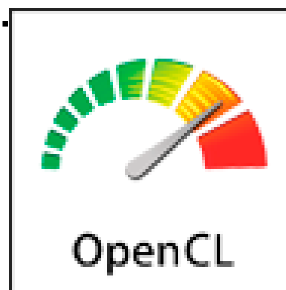
Graphics APIs and Shading Languages

OpenCL is a programming framework for heterogeneous compute resources

© Copyright Khronos Group, 2010 - Page 3

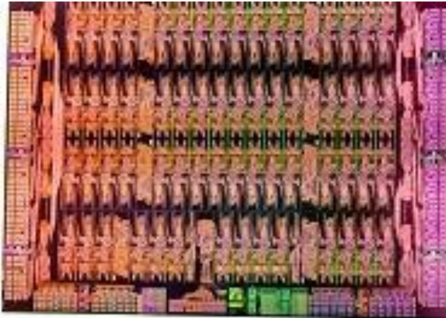
OpenCL

- *O*pen *C*ompute *L*anguage
- For heterogeneous parallel-computing systems
- Cross-platform
  - Implementations for
    - ATI GPUs
    - NVIDIA GPUs
    - **Intel MIC**
    - 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 Mac OS 10.6 (Snow Leopard)
- Most recent: OpenCL 1.2 Nov 2011
- Release for multicore ARM CPUs Nov 2012
- Developed by the Khronos Group

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

© Copyright Khronos Group, 2010 - Page 4

OpenCL

- API very close to OpenGL
- Based on the C language (addition to the language for parallelism, built-in functions..)
- 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

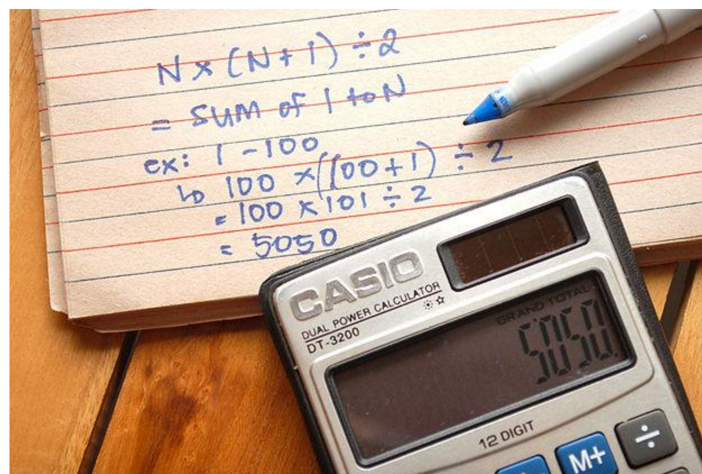
➢ for example, ***cl::memory*** maps to OpenCL type ***cl_mem***

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

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

❑ Libraries and Tools (ViennaCL,…)

OpenCL simple example



Just to illustrate basic OpenCL commands

Sum 0+1+…ARRAY_SIZE-1

array transferred to device

Result returned to host (CPU)

OpenCL code

```
#define PROGRAM_FILE "add_numbers.cl"
#define KERNEL_FUNC "add_numbers"
#define ARRAY_SIZE 1048576
#define G_ARRAY_SIZE (ARRAY_SIZE/8)
#define L_ARRAY_SIZE (ARRAY_SIZE/32)

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#ifdef MAC
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif

/* Find a device associated with the first available platform */
cl_device_id create_device() {

 cl_platform_id platform;
 cl_device_id dev[10],device;


 cl_uint number,i;
   int err;

   /* Identify a platform */
   err = clGetPlatformIDs(1, &platform, &number);
   if(err < 0) {
      perror("Couldn't identify a platform");
      exit(1);
   }
   char buffer[10240];
   cl_uint buf_uint,idims;
   cl_ulong buf_ulong;
   size_t wgsize,wisize[3];
```

```
 for(i=0; i<number; i++) {
      clGetPlatformInfo(platform,CL_PLATFORM_PROFILE,10240,buffer,NULL);
      printf("PROFILE=%s\n",buffer);
      clGetPlatformInfo(platform,CL_PLATFORM_VERSION,10240,buffer,NULL);
      printf("VERSION=%s\n",buffer);
      clGetPlatformInfo(platform,CL_PLATFORM_NAME,10240,buffer,NULL);
      printf("NAME=%s\n",buffer);
      clGetPlatformInfo(platform,CL_PLATFORM_VENDOR,10240,buffer,NULL);
      printf("VENDOR=%s\n",buffer);
      clGetPlatformInfo(platform,CL_PLATFORM_EXTENSIONS,10240,buffer,NULL);
      printf("EXTENSIONS=%s\n",buffer);
   }

   /* Access a device */
   err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 10, dev, &number);
   if(err == CL_DEVICE_NOT_FOUND) {
      perror("Couldn't access any devices");
      exit(1);
   }
 for(i=0; i<number; i++) {
      printf("--%d--\n",i);
      clGetDeviceInfo(dev[i],CL_DEVICE_NAME,10240,buffer,NULL);
      printf("DEVICE NAME=%s\n",buffer);
      clGetDeviceInfo(dev[i],CL_DEVICE_VENDOR,10240,buffer,NULL);
      printf("DEVICE VENDOR=%s\n",buffer);
      clGetDeviceInfo(dev[i],CL_DEVICE_VERSION,10240,buffer,NULL);
      printf("DEVICE VERSION=%s\n",buffer);
      clGetDeviceInfo(dev[i],CL_DEVICE_MAX_COMPUTE_UNITS,sizeof(buf_uint),&buf_uint,NULL);
      printf("DEVICE_MAX_COMPUTE_UNITS=%u\n",buf_uint);
      clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_GROUP_SIZE,sizeof(size_t),&wgsize,NULL);
      printf("DEVICE_MAX_WORK_GROUP_SIZE=%i\n",(int)wgsize);

clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,sizeof(cl_uint),&idims,NULL);
      printf("DEVICE_MAX_WORK_ITEM_DIMENSIONS=%u\n",idims);
      clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_SIZES,sizeof(size_t[3]),&wisize,NULL);
      printf("DEVICE_MAX_WORK_ITEM_SIZES=%lu %lu %lu\n",wisize[0],wisize[1],wisize[2]);
      clGetDeviceInfo(dev[i],CL_DEVICE_GLOBAL_MEM_SIZE,sizeof(cl_ulong),&buf_ulong,NULL);
 }
   device=dev[0];
   return device;
}
```

OpenCL code again

Summer School on PARALLEL COMPUTING

```
/* Create program from a file and compile it */
cl_program build_program(cl_context ctx, cl_device_id dev, const char*
filename) {

   cl_program program;
   FILE *program_handle;
   char *program_buffer, *program_log;
   size_t program_size, log_size;
   int err;

   /* Read program file and place content into buffer */
   program_handle = fopen(filename, "r");
   if(program_handle == NULL) {
      perror("Couldn't find the program file");
      exit(1);
   }
   fseek(program_handle, 0, SEEK_END);
   program_size = ftell(program_handle);
   rewind(program_handle);
   program_buffer = (char*)malloc(program_size + 1);
   program_buffer[program_size] = '\0';
   fread(program_buffer, sizeof(char), program_size, program_handle);
 fclose(program_handle);

   /* Create program from file */
   program = clCreateProgramWithSource(ctx, 1,
      (const char**)&program_buffer, &program_size, &err);
   if(err < 0) {
      perror("Couldn't create the program");
      exit(1);
   }
   free(program_buffer);

   /* Build program */
   err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

```
if(err < 0) {

      /* Find size of log and print to std output */
      clGetProgramBuildInfo(program, dev, CL_PROGRAM_BUILD_LOG,
            0, NULL, &log_size);
      program_log = (char*) malloc(log_size + 1);
      program_log[log_size] = '\0';
      clGetProgramBuildInfo(program, dev, CL_PROGRAM_BUILD_LOG,
            log_size + 1, program_log, NULL);
      printf("%s\n", program_log);
      free(program_log);
      exit(1);
   }

   return program;
}
int main() {

   /* OpenCL structures */
   cl_device_id device;
   cl_context context;
   cl_program program;
   cl_kernel kernel;
   cl_command_queue queue;
   cl_int i, j, err,counter;
   size_t local_size, global_size;

   /* Data and buffers */
   float data[ARRAY_SIZE];
   float sum[L_ARRAY_SIZE], total, actual_sum;
   cl_mem input_buffer, sum_buffer;
   cl_int num_groups;

   /* Initialize data */
   for(i=0; i<ARRAY_SIZE; i++) {
      data[i] = 1.0f*i;
   }
```

OpenCL code again again

```
/* Create device and context */
   device = create_device();
   context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
   if(err < 0) {
      perror("Couldn't create a context");
      exit(1);
   }

   /* Build program */
   program = build_program(context, device, PROGRAM_FILE);

 /* Create data buffer */
   global_size = G_ARRAY_SIZE;
   local_size = 4;
   num_groups = global_size/local_size;
   input_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY |
         CL_MEM_COPY_HOST_PTR, ARRAY_SIZE * sizeof(float), data,
&err);
   sum_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE |
         CL_MEM_COPY_HOST_PTR, num_groups * sizeof(float), sum,
&err);
   if(err < 0) {
      perror("Couldn't create a buffer");
      exit(1);
   };

   /* Create a command queue */
   queue = clCreateCommandQueue(context, device, 0, &err);
   if(err < 0) {
      perror("Couldn't create a command queue");
      exit(1);
   };
```

```
 /* Create a kernel */
   kernel = clCreateKernel(program, KERNEL_FUNC, &err);
   if(err < 0) {
      perror("Couldn't create a kernel");
      exit(1);
   };

   /* Create kernel arguments */
   err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input_buffer);
   err |= clSetKernelArg(kernel, 1, local_size * sizeof(float), NULL);
   err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &sum_buffer);
   if(err < 0) {
      perror("Couldn't create a kernel argument");
      exit(1);
   }
   for(counter=0; counter<1000; counter++) {
   /* Enqueue kernel */
   err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global_size,
         &local_size, 0, NULL, NULL);
   if(err < 0) {
      perror("Couldn't enqueue the kernel");
      exit(1);
}
   }

   /* Read the kernel's output */
   err = clEnqueueReadBuffer(queue, sum_buffer, CL_TRUE, 0,
         sizeof(sum), sum, 0, NULL, NULL);
   if(err < 0) {
      perror("Couldn't read the buffer");
      exit(1);
   }

   /* Check result */
   total = 0.0f;
   for(j=0; j<num_groups; j++) {
      total += sum[j];
   }
   actual_sum = 1.0f * ARRAY_SIZE/2*(ARRAY_SIZE-1);
   printf("Computed sum = %.1f.\n", total);
```

OpenCL code again again and again

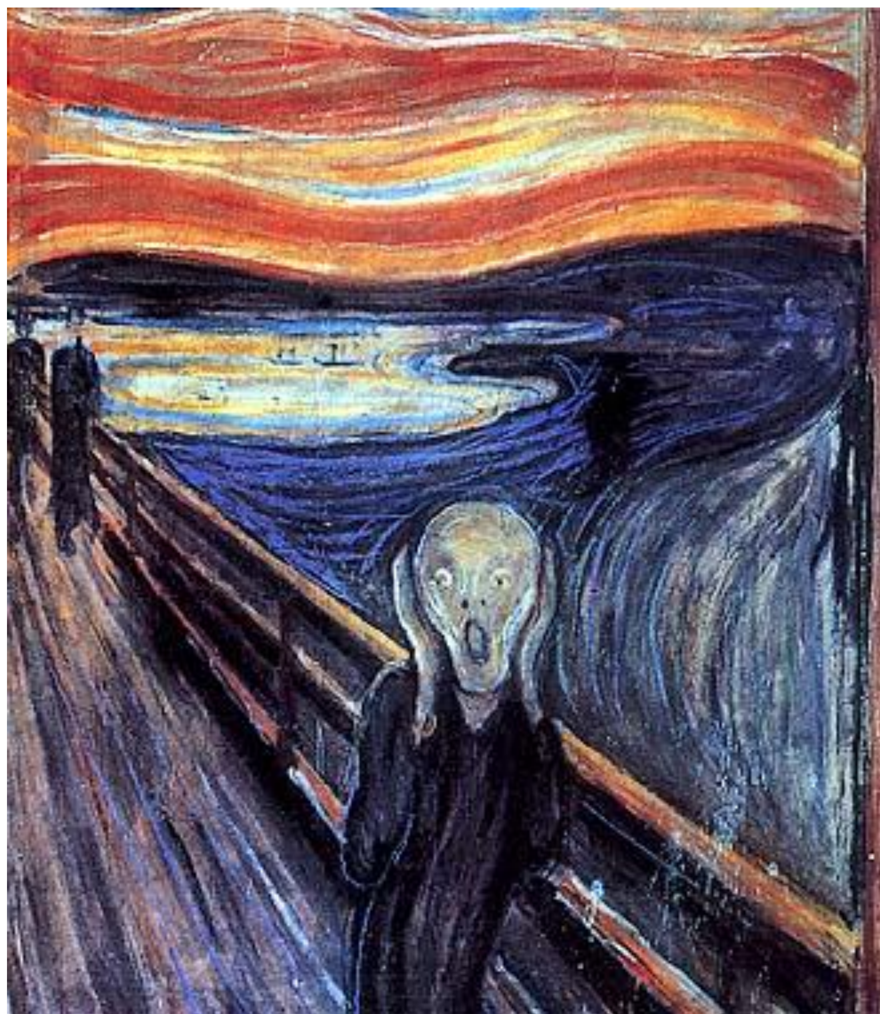
```
if(fabs(total - actual_sum) > 0.001*fabs(actual_sum))
    printf("Check failed.\n");
  else
    printf("Check passed.\n");

  /* Deallocate resources */
  clReleaseKernel(kernel);
  clReleaseMemObject(sum_buffer);
  clReleaseMemObject(input_buffer);
  clReleaseCommandQueue(queue);
  clReleaseProgram(program);
  clReleaseContext(context);
  return 0;
}
```

```
__kernel void add_numbers(__global float4* data,
      __local float* local_result, __global float* group_result) {

   float sum;
   float4 input1, input2, sum_vector;
   uint global_addr, local_addr;

   global_addr = get_global_id(0) * 2;
   input1 = data[global_addr];
   input2 = data[global_addr+1];
   sum_vector = input1 + input2;

   local_addr = get_local_id(0);
   local_result[local_addr] = sum_vector.s0 + sum_vector.s1 +
                              sum_vector.s2 + sum_vector.s3;
   barrier(CLK_LOCAL_MEM_FENCE);

if(get_local_id(0) == 0) {
      sum = 0.0f;
      for(int i=0; i<get_local_size(0); i++) {
         sum += local_result[i];
      }
      group_result[get_group_id(0)] = sum;
   }
}
```

The Kernel!

OpenCL

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



CINECA

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 defined in OpenCL API

```
/* Find a device associated with the first available platform */
cl_device_id create_device() {

 cl_platform_id platform;
 cl_device_id dev[10],device;


 cl_uint number,i;
  int err;

  /* Identify a platform */
  err = clGetPlatformIDs(1, &platform, &number);
  if(err < 0) {
     perror("Couldn't identify a platform");
     exit(1);
  }
  char buffer[10240];
  cl_uint buf_uint,idims;
  cl_ulong buf_ulong;
  size_t wgsize,wisize[3];
```

```
 for(i=0; i<number; i++) {
       clGetPlatformInfo(platform,CL_PLATFORM_PROFILE,10240,buffer,NULL);
       printf("PROFILE=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_VERSION,10240,buffer,NULL);
       printf("VERSION=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_NAME,10240,buffer,NULL);
       printf("NAME=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_VENDOR,10240,buffer,NULL);
       printf("VENDOR=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_EXTENSIONS,10240,buffer,NULL);
       printf("EXTENSIONS=%s\n",buffer);
  }

  /* Access a device */
  err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 10, dev, &number);
  if(err == CL_DEVICE_NOT_FOUND) {
     perror("Couldn't access any devices");
     exit(1);
  }
 for(i=0; i<number; i++) {
       printf("--%d--\n",i);
       clGetDeviceInfo(dev[i],CL_DEVICE_NAME,10240,buffer,NULL);
       printf("DEVICE NAME=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_VENDOR,10240,buffer,NULL);
       printf("DEVICE VENDOR=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_VERSION,10240,buffer,NULL);
       printf("DEVICE VERSION=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_COMPUTE_UNITS,sizeof(buf_uint),&buf_uint,NULL);
       printf("DEVICE_MAX_COMPUTE_UNITS=%u\n",buf_uint);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_GROUP_SIZE,sizeof(size_t),&wgsize,NULL);
       printf("DEVICE_MAX_WORK_GROUP_SIZE=%i\n",(int)wgsize);

clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,sizeof(cl_uint),&idims,NULL);
       printf("DEVICE_MAX_WORK_ITEM_DIMENSIONS=%u\n",idims);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_SIZES,sizeof(size_t[3]),&wisize,NULL);
       printf("DEVICE_MAX_WORK_ITEM_SIZES=%lu %lu %lu\n",wisize[0],wisize[1],wisize[2]);
       clGetDeviceInfo(dev[i],CL_DEVICE_GLOBAL_MEM_SIZE,sizeof(cl_ulong),&buf_ulong,NULL);
 }
  device=dev[0];
  return device;
}
```

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

CL_DEVICE_TYPE_ALL is equivalent to all OpenCL devices in the system

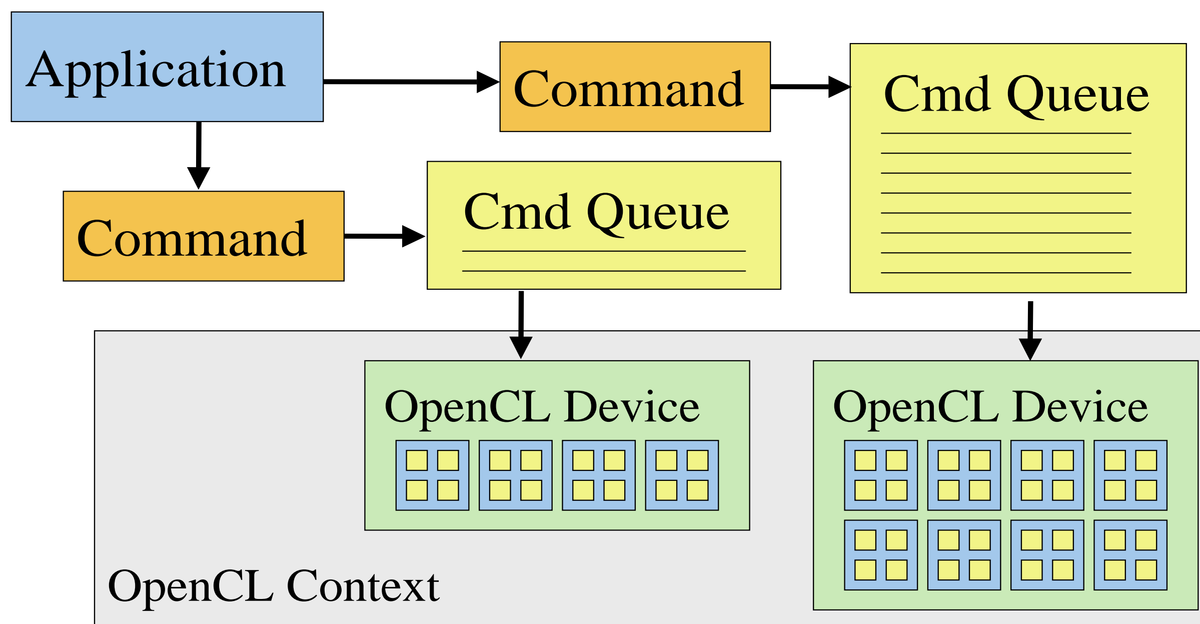
```
/* Find a device associated with the first available platform */
cl_device_id create_device() {

 cl_platform_id platform;
 cl_device_id dev[10],device;


 cl_uint number,i;
   int err;

   /* Identify a platform */
   err = clGetPlatformIDs(1, &platform, &number);
   if(err < 0) {
      perror("Couldn't identify a platform");
      exit(1);
   }
   char buffer[10240];
   cl_uint buf_uint,idims;
   cl_ulong buf_ulong;
   size_t wgsize,wisize[3];
```

```
 for(i=0; i<number; i++) {
       clGetPlatformInfo(platform,CL_PLATFORM_PROFILE,10240,buffer,NULL);
       printf("PROFILE=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_VERSION,10240,buffer,NULL);
       printf("VERSION=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_NAME,10240,buffer,NULL);
       printf("NAME=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_VENDOR,10240,buffer,NULL);
       printf("VENDOR=%s\n",buffer);
       clGetPlatformInfo(platform,CL_PLATFORM_EXTENSIONS,10240,buffer,NULL);
       printf("EXTENSIONS=%s\n",buffer);
   }

   /* Access a device */
   err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 10, dev, &number);
   if(err == CL_DEVICE_NOT_FOUND) {
      perror("Couldn't access any devices");
      exit(1);
   }
 for(i=0; i<number; i++) {
       printf("--%d--\n",i);
       clGetDeviceInfo(dev[i],CL_DEVICE_NAME,10240,buffer,NULL);
       printf("DEVICE NAME=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_VENDOR,10240,buffer,NULL);
       printf("DEVICE VENDOR=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_VERSION,10240,buffer,NULL);
       printf("DEVICE VERSION=%s\n",buffer);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_COMPUTE_UNITS,sizeof(buf_uint),&buf_uint,NULL);
       printf("DEVICE_MAX_COMPUTE_UNITS=%u\n",buf_uint);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_GROUP_SIZE,sizeof(size_t),&wgsize,NULL);
       printf("DEVICE_MAX_WORK_GROUP_SIZE=%i\n",(int)wgsize);

clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,sizeof(cl_uint),&idims,NULL);
       printf("DEVICE_MAX_WORK_ITEM_DIMENSIONS=%u\n",idims);
       clGetDeviceInfo(dev[i],CL_DEVICE_MAX_WORK_ITEM_SIZES,sizeof(size_t[3]),&wisize,NULL);
       printf("DEVICE_MAX_WORK_ITEM_SIZES=%lu %lu %lu\n",wisize[0],wisize[1],wisize[2]);
       clGetDeviceInfo(dev[i],CL_DEVICE_GLOBAL_MEM_SIZE,sizeof(cl_ulong),&buf_ulong,NULL);
 }
   device=dev[0];
   return device;
}
```

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()

```
/* Create device and context */
   device = create_device();
   context = clCreateContext(NULL, 1, &device, NULL, NULL, &err);
   if(err < 0) {
      perror("Couldn't create a context");
      exit(1);
   }
```

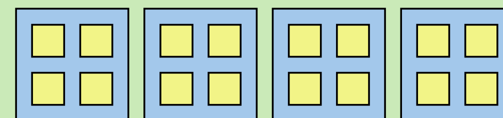
```
int main() {

   /* OpenCL structures */
   cl_device_id device;
   cl_context context;
   cl_program program;
   cl_kernel kernel;
   cl_command_queue queue;
   cl_int i, j, err,counter;
   size_t local_size, global_size;

   /* Data and buffers */
   float data[ARRAY_SIZE];
   float sum[L_ARRAY_SIZE], total, actual_sum;
   cl_mem input_buffer, sum_buffer;
   cl_int num_groups;

   /* Initialize data */
   for(i=0; i<ARRAY_SIZE; i++) {
      data[i] = 1.0f*i;
   }
```

Context again

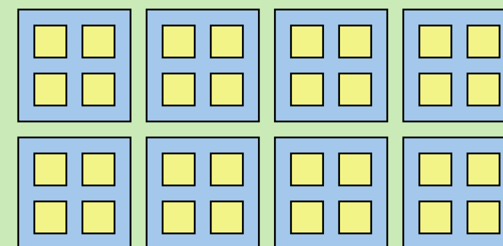
- Contains one or more devices
- OpenCL memory objects associated with a context, not a specific device
- clCreateBuffer() is the main data object allocation function
  - error if an allocation is too large for any device in the context
- Each device needs its own work / command queue(s)
- Memory transfers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

OpenCL Context

OpenCL Device



OpenCL 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()

```
/* Create a command queue */
   queue = clCreateCommandQueue(context, device, 0, &err);
   if(err < 0) {
      perror("Couldn't create a command queue");
      exit(1);
   };
```

```
int main() {

   /* OpenCL structures */
   cl_device_id device;
   cl_context context;
   cl_program program;
   cl_kernel kernel;
   cl_command_queue queue;
   cl_int i, j, err,counter;
   size_t local_size, global_size;

   /* Data and buffers */
   float data[ARRAY_SIZE];
   float sum[L_ARRAY_SIZE], total, actual_sum;
   cl_mem input_buffer, sum_buffer;
   cl_int num_groups;

   /* Initialize data */
   for(i=0; i<ARRAY_SIZE; i++) {
      data[i] = 1.0f*i;
   }
```

Allocating memory on device

clCreateBuffer() allocation function

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 object and copy data from memory referenced by host_ptr pointer

```
/* Create data buffer */
   global_size = G_ARRAY_SIZE;
   local_size = 4;
   num_groups = global_size/local_size;
   input_buffer = clCreateBuffer(context, CL_MEM_READ_ONLY |
         CL_MEM_COPY_HOST_PTR, ARRAY_SIZE * sizeof(float), data,
&err);
   sum_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE |
         CL_MEM_COPY_HOST_PTR, num_groups * sizeof(float), sum,
&err);
   if(err < 0) {
      perror("Couldn't create a buffer");
      exit(1);
   };
```

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 add_numbers(__global float4* data,
      __local float* local_result, __global float* group_result) {

   float sum;
   float4 input1, input2, sum_vector;
   uint global_addr, local_addr;

   global_addr = get_global_id(0) * 2;
   input1 = data[global_addr];
   input2 = data[global_addr+1];
   sum_vector = input1 + input2;

   local_addr = get_local_id(0);
   local_result[local_addr] = sum_vector.s0 + sum_vector.s1 +
                              sum_vector.s2 + sum_vector.s3;
   barrier(CLK_LOCAL_MEM_FENCE);

if(get_local_id(0) == 0) {
      sum = 0.0f;
      for(int i=0; i<get_local_size(0); i++) {
         sum += local_result[i];
      }
      group_result[get_group_id(0)] = sum;
   }
}
```

CINECA

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

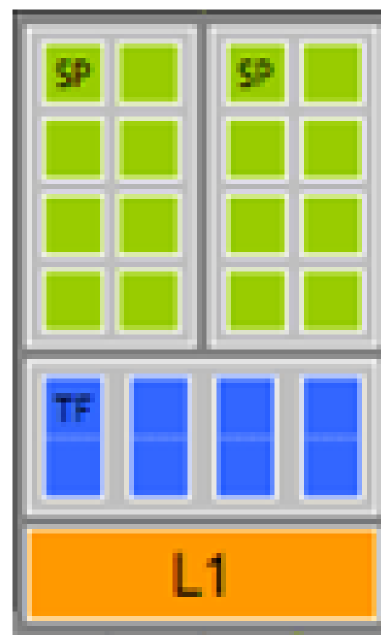
OpenCL NDRange Configuration

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()

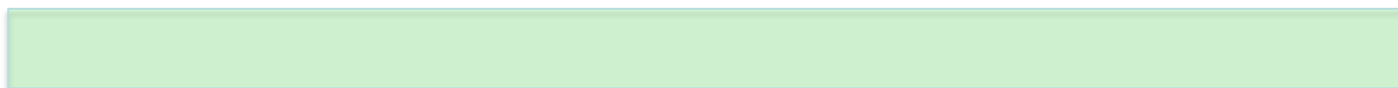
```
__kernel void SAXPY (__global float* x, __global  float*
y, float a)
{
                const int i = get_global_id (0);

                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

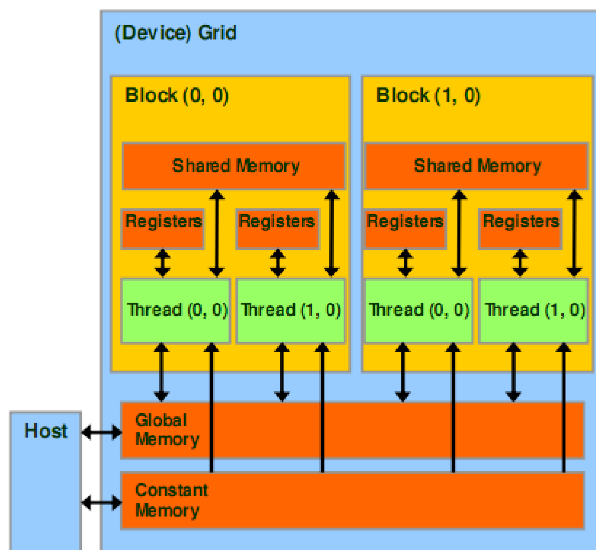
- __global – large, long latency
- __private – on-chip device registers
- __local – memory accessible from multiple PEs or work items
  - May be SRAM or DRAM, must query...
- __constant – read-only constant cache
- Programmer manages device memory explicitly

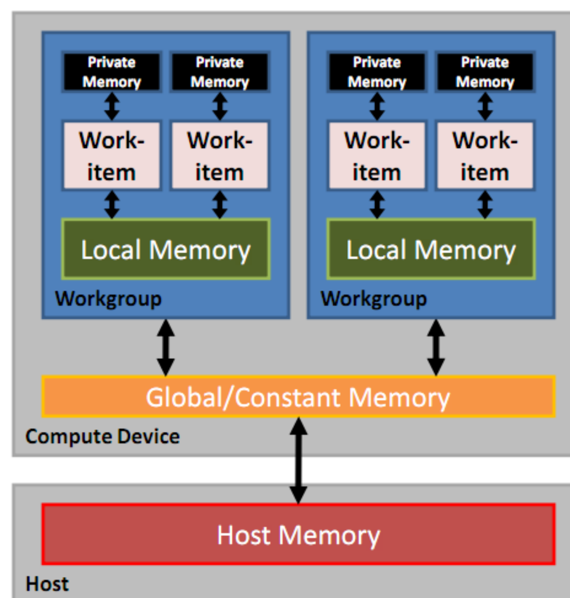


OpenCL Memory Types	CUDA Equivalent
global memory	global memory
constant memory	constant memory
local memory	shared memory
private memory	Local memory

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 from a file and compile it */
cl_program build_program(cl_context ctx, cl_device_id dev, const char*
filename) {

   cl_program program;
   FILE *program_handle;
   char *program_buffer, *program_log;
   size_t program_size, log_size;
   int err;

   /* Read program file and place content into buffer */
   program_handle = fopen(filename, "r");
   if(program_handle == NULL) {
      perror("Couldn't find the program file");
      exit(1);
   }
   fseek(program_handle, 0, SEEK_END);
   program_size = ftell(program_handle);
   rewind(program_handle);
   program_buffer = (char*)malloc(program_size + 1);
   program_buffer[program_size] = '\0';
   fread(program_buffer, sizeof(char), program_size, program_handle);
 fclose(program_handle);

   /* Create program from file */
   program = clCreateProgramWithSource(ctx, 1,
      (const char**)&program_buffer, &program_size, &err);
   if(err < 0) {
      perror("Couldn't create the program");
      exit(1);
   }
   free(program_buffer);

   /* Build program */
   err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

```
if(err < 0) {

      /* Find size of log and print to std output */
      clGetProgramBuildInfo(program, dev, CL_PROGRAM_BUILD_LOG,
            0, NULL, &log_size);
      program_log = (char*) malloc(log_size + 1);
      program_log[log_size] = '\0';
      clGetProgramBuildInfo(program, dev, CL_PROGRAM_BUILD_LOG,
            log_size + 1, program_log, NULL);
      printf("%s\n", program_log);
      free(program_log);
      exit(1);
   }

   return program;
}
int main() {

   /* OpenCL structures */
   cl_device_id device;
   cl_context context;
   cl_program program;
   cl_kernel kernel;
   cl_command_queue queue;
   cl_int i, j, err,counter;
   size_t local_size, global_size;

   /* Data and buffers */
   float data[ARRAY_SIZE];
   float sum[L_ARRAY_SIZE], total, actual_sum;
   cl_mem input_buffer, sum_buffer;
   cl_int num_groups;

   /* Initialize data */
   for(i=0; i<ARRAY_SIZE; i++) {
      data[i] = 1.0f*i;
   }
```

Create OpenCL Kernel Object(s)

Build (compiles and link) kernel program

```
/* Build program */
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

Preprocessor, OpenCL extensions and optimization options here

Creating kernel objects

Set kernel arguments

```
/* Create a kernel */
  kernel = clCreateKernel(program, KERNEL_FUNC, &err);
  if(err < 0) {
     perror("Couldn't create a kernel");
     exit(1);
  };

  /* Create kernel arguments */
  err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input_buffer);
  err |= clSetKernelArg(kernel, 1, local_size * sizeof(float), NULL);
  err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &sum_buffer);
  if(err < 0) {
     perror("Couldn't create a kernel argument");
     exit(1);
  }
);
```

CINECA

Enqueue and Copy back

Enqueue a command to execute kernel on device

```
for(counter=0; counter<1000; counter++) {
  /* Enqueue kernel */
  err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global_size,
        &local_size, 0, NULL, NULL);
  if(err < 0) {
    perror("Couldn't enqueue the kernel");
    exit(1);
}
  }

  /* Read the kernel's output */
  err = clEnqueueReadBuffer(queue, sum_buffer, CL_TRUE, 0,
        sizeof(sum), sum, 0, NULL, NULL);
  if(err < 0) {
    perror("Couldn't read the buffer");
    exit(1);
  }

  /* Check result */
  total = 0.0f;
  for(j=0; j<num_groups; j++) {
    total += sum[j];
  }
  actual_sum = 1.0f * ARRAY_SIZE/2*(ARRAY_SIZE-1);
  printf("Computed sum = %.1f.\n", total);
```

Copy results from buffer object to host memory

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

CL_TRUE imply blocking read

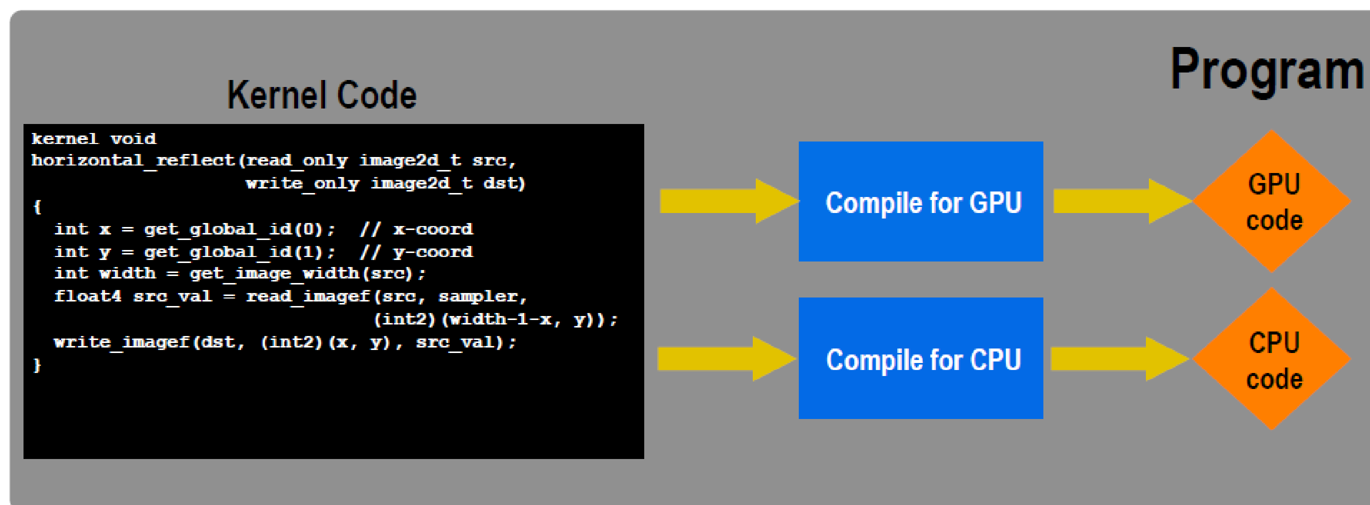
Clean-up

```
/* Deallocate resources */
   clReleaseKernel(kernel);
   clReleaseMemObject(sum_buffer);
   clReleaseMemObject(input_buffer);
   clReleaseCommandQueue(queue);
   clReleaseProgram(program);
   clReleaseContext(context);
   return 0;
}
```

Running OpenCL code on Multiple Devices

Executing Code

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



© Copyright Khronos Group, 2011 - Page 36

Compiling for Intel MIC device

Need OpenCL header:

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

and link to the OpenCL library.

Compile OpenCL host program add_numbers.c using icc compiler,

```
icc   -o add_numbers add_numbers.c -I. -I/opt/intel/opencl/include/  -L/opt/intel/opencl/libmic -lOpenCL
```

Compiling for NVIDIA device

Need OpenCL header:

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

and link to the OpenCL library.

Compile OpenCL host program add_numbers.c using gcc compiler,

```
gcc -std=c99  -o add_numbers add_numbers.c -I. -I/cineca/prod/compilers/cuda/5.0.35/none/include  -lOpenCL
```

Running environment

NVIDIA Tesla K20

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

Intel MIC Xeon Phi

- 236 compute units
- 8 GB of global memory
- CPU clock rate 1052 MHz

Running on Intel

```
PROFILE=FULL_PROFILE
VERSION=OpenCL 1.2 LINUX
NAME=Intel(R) OpenCL
VENDOR=Intel(R) Corporation
EXTENSIONS=cl_khr_fp64 cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store
--0--
DEVICE NAME=     Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
DEVICE VENDOR=Intel(R) Corporation
DEVICE VERSION=OpenCL 1.2 (Build 67279)
DEVICE_MAX_COMPUTE_UNITS=16
DEVICE_MAX_WORK_GROUP_SIZE=1024
DEVICE_MAX_WORK_ITEM_DIMENSIONS=3
DEVICE_MAX_WORK_ITEM_SIZES=1024 1024 1024
DEVICE_GLOBAL_MEM_SIZE=16685436928
--1--
DEVICE NAME=Intel(R) Many Integrated Core Acceleration Card
DEVICE VENDOR=Intel(R) Corporation
DEVICE VERSION=OpenCL 1.2 (Build 67279)
DEVICE_MAX_COMPUTE_UNITS=236
DEVICE_MAX_WORK_GROUP_SIZE=1024
DEVICE_MAX_WORK_ITEM_DIMENSIONS=3
DEVICE_MAX_WORK_ITEM_SIZES=1024 1024 1024
DEVICE_GLOBAL_MEM_SIZE=6053646336
--2--
DEVICE NAME=Intel(R) Many Integrated Core Acceleration Card
DEVICE VENDOR=Intel(R) Corporation
DEVICE VERSION=OpenCL 1.2 (Build 67279)
DEVICE_MAX_COMPUTE_UNITS=236
DEVICE_MAX_WORK_GROUP_SIZE=1024
DEVICE_MAX_WORK_ITEM_DIMENSIONS=3
DEVICE_MAX_WORK_ITEM_SIZES=1024 1024 1024
DEVICE_GLOBAL_MEM_SIZE=6053646336
Computed sum = 549754961920.0.
Check passed.
```

Intel OpenCL platform found and 3 devices (cpu and Intel MIC card)

Intel MIC device was selected

Results are OK no matter what performances

- Targets a broader rang of CPU-l and GPU-like devices th n CUDA
  - Targets endors
  - Man 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,…)
- Simple OpenCL codes have a proven impressive performance on given hardware (i.e. Himeno Benchmark, update kernel)

26 GFLOPs

on one K20

```
__kernel void update(const float OMEGA, const int msizex,const int
msizey,const int msizez, const int sizex,const int sizey,const int sizez,__global
float* a1, __global float* a2,__global float* a3, __global float* a4, __global float*
b1, __global float* b2, __global float* b3, __global float* c1, __global float* c2,
__global float* c3, __global float* wrk1, __global float* wrk2, __global float*
bnd, __global float* p, __global float* gosa)
{
// (x,y,z)-> (x+sizex*(y+sizey*z))
#define idxz(I,J,K) ((I)+msizex*((J)+msizey*(K)))

   int I=get_global_id(0);
   int J=get_global_id(1);
   int K=get_global_id(2);
   const int sizex1=sizex-1;
   const int sizey1=sizey-1;
   const int sizez1=sizez-1;

   gosa[get_group_id(0)]=0.0;
   if(I>=1 && I<sizex1 &&J>=1 && J<sizey1 &&K>=1 && K<sizez1)
{
   float s0=a1[idxz(I,J,K)]*p[idxz(I+1,J,K)]+ a2[idxz(I,J,K)]*p[idxz(I,J+1,K)]+
a3[idxz(I,J,K)]*p[idxz(I,J,K+1)]+ b1[idxz(I,J,K)]*(p[idxz(I+1,J+1,K)]-p[idxz(I+1,J-
1,K)]-p[idxz(I-1,J+1,K)]+p[idxz(I-1,J-1,K)])+ b2[idxz(I,J,K)]*(p[idxz(I,J+1,K+1)]-
p[idxz(I,J-1,K+1)]-p[idxz(I,J+1,K-1)]+p[idxz(I,J-1,K-1)])+
b3[idxz(I,J,K)]*(p[idxz(I+1,J,K+1)]-p[idxz(I-1,J,K+1)]-p[idxz(I+1,J,K-1)]+p[idxz(I-
1,J,K-1)])+ c1[idxz(I,J,K)]*p[idxz(I-1,J,K)]+ c2[idxz(I,J,K)]*p[idxz(I,J-1,K)]+
c3[idxz(I,J,K)]*p[idxz(I,J,K-1)]+wrk1[idxz(I,J,K)];
   float ss=(s0*a4[idxz(I,J,K)]-p[idxz(I,J,K)])*bnd[idxz(I,J,K)];
   wrk2[idxz(I,J,K)]=p[idxz(I,J,K)]+OMEGA*ss;
 }
}
```

OpenCL Performance

- Tremendous amount of computing power available

1170 GFLOPs peak

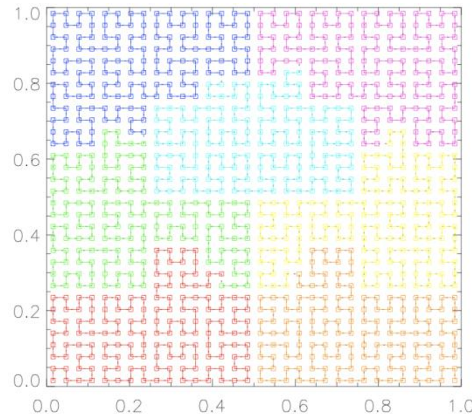
1070 GFLOPs peak

- OpenCL promises and potential: 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 across completely different hardware?

The Hydro benchmark

Romain Dolbeau, Francois Bodin, Guillame Colin de Verdière

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 at CINECA

- We use hydro benchmark at CINECA in order to understand:
- ❑ readability of a medium size OpenCL source code
- ❑ OpenCL portability across different platform and devices
- ❑ OpenCL performances across different platform and devices
- ❑ …some other things….

The Hydro OpenCL benchmark at CINECA

readability of a medium size OpenCL source code

- 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 varying local and global work group size on different hardware)

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;
}
```

The Hydro OpenCL benchmark at CINECA

- We use hydro benchmark at CINECA in order to understand:
- ❑ readability of a medium size OpenCL source code
- ❑ **OpenCL portability across different platform and devices**
- ❑ **OpenCL performances across different platform and devices**
- ❑ …some other things….

Optimizing OpenCL apps on Intel Xeon Phi

- OpenCL Hydro code is running on Intel and NVIDIA Platform. The same code runs on both platforms (portability of OpenCL).
- **Performances are strictly depending on hardware. For Intel MIC:**

- ❑ At initialization time OpenCL driver create 240 SW threads and pins them to HW threads. *clEnqueueNDRange* schedules work groups (WG) on the 240 threads.
- ❑ A WG is the smallest task on thread. So, with less than 240 WGs, leaves the coprocessor underutilized. Use more than 240 WGs (1000 or more are good).
- ❑ Implicit vectorization over dimension zero of NDRange. No reason to vectorize manually
- ❑ Avoid local memory and barriers (are emulated by software). Intel MIC doesn’t distinguish between local and global memory
- ❑ Data prefetching is crucial for performances. New OpenCL *clBuildProgram* switch specific to Intel MIC Xeon Phi
- ❑ **Different guidelines for NVIDIA platform**

Coming Next: Hydro live@Eurora@CINECA

Hydro live@Eurora

- Eurora CINECA-Eurotech prototype
- 1 rack
- Two Intel SandyBridge and
- two NVIDIA K20 cards per node
- **Two Intel MIC card per node**
- Hot water cooling
- Energy efficiency record (up to 3210 MFLOPs/w)
- 100 TFLOPs sustained

Hydro OpenCL comparison

Performances of NVIDIA platform are very good. Intel OpenCL for MIC performs better than CPU OpenCL

OpenCL run, 4091x4091 domain,

100 iterations

Device/version	Elapsed time (sec.) without initialization	EfficiencyLoss (with respect to the best timing)
Intel OpenCL for CPU	154.3	2.66
OpenCL K20	42.09	0
Intel OpenCL for MIC (-auto-prefetch-level=3)	111.21	1.64

Intel MIC preliminary run on Eurora. There is room for improvement. Using prefetching at the higher level gives boost of 15%

Hydro OpenCL tuning

OpenCL run, single Intel MIC card,4091x4091 domain,100 iterations

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

Work Group size	Elapsed time (sec.) without initialization	EfficiencyLoss (with respect to the best timing)
16	297.68	1.67
64	117.69	0.05
128	113.92	0.02
256	115.91	0.04
512	111.21	0
1024	111.76	0.004

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

Hydro OpenCL tuning

OpenCL run, single NVIDIA K20, 4091x4091 domain,100 iterations

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

Work Group size	Elapsed time (sec.) without initialization	EfficiencyLoss (with respect to the best timing)
16	82.88	0.96
64	43.10	0.02
128	43.19	0.02
256	43.90	0.04
512	42.09	0
1024	45.71	0.08

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

Hydro run comparison

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

OpenAcc run it fails using Pgi compiler

More than 16 Intel Xeon SandyBridge cores are needed to compare OpenCL 1 K20 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 K20	52.37	0.24
OpenCL K20	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**

References

Optimizing OpenCL applications on Intel Xeon Phi
http://iwocl.org/wp-content/uploads/2013/06/Optimizing-OpenCL-Applications-on-Intel-Xeon-Phi-IWOCL.pdf

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