OpenCL Tutorial

CEPHIS

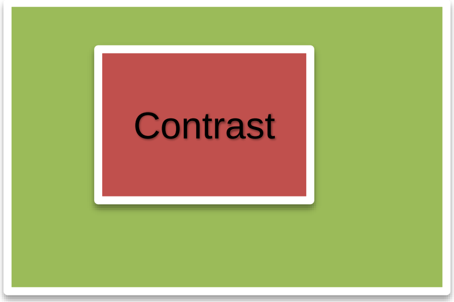
David Castells-Rufas
Microelectronics & Electronics Systems Department
Universitat Autònoma de Barcelona
david.castells@uab.cat

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

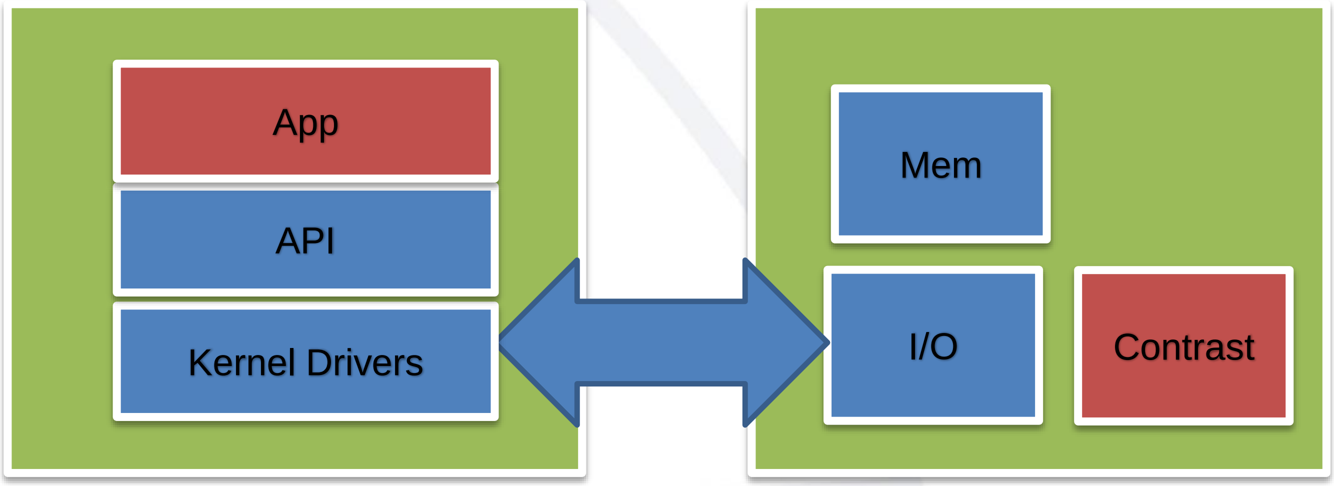
OpenCL for FPGAs

Accelerators

- We did



- But we need

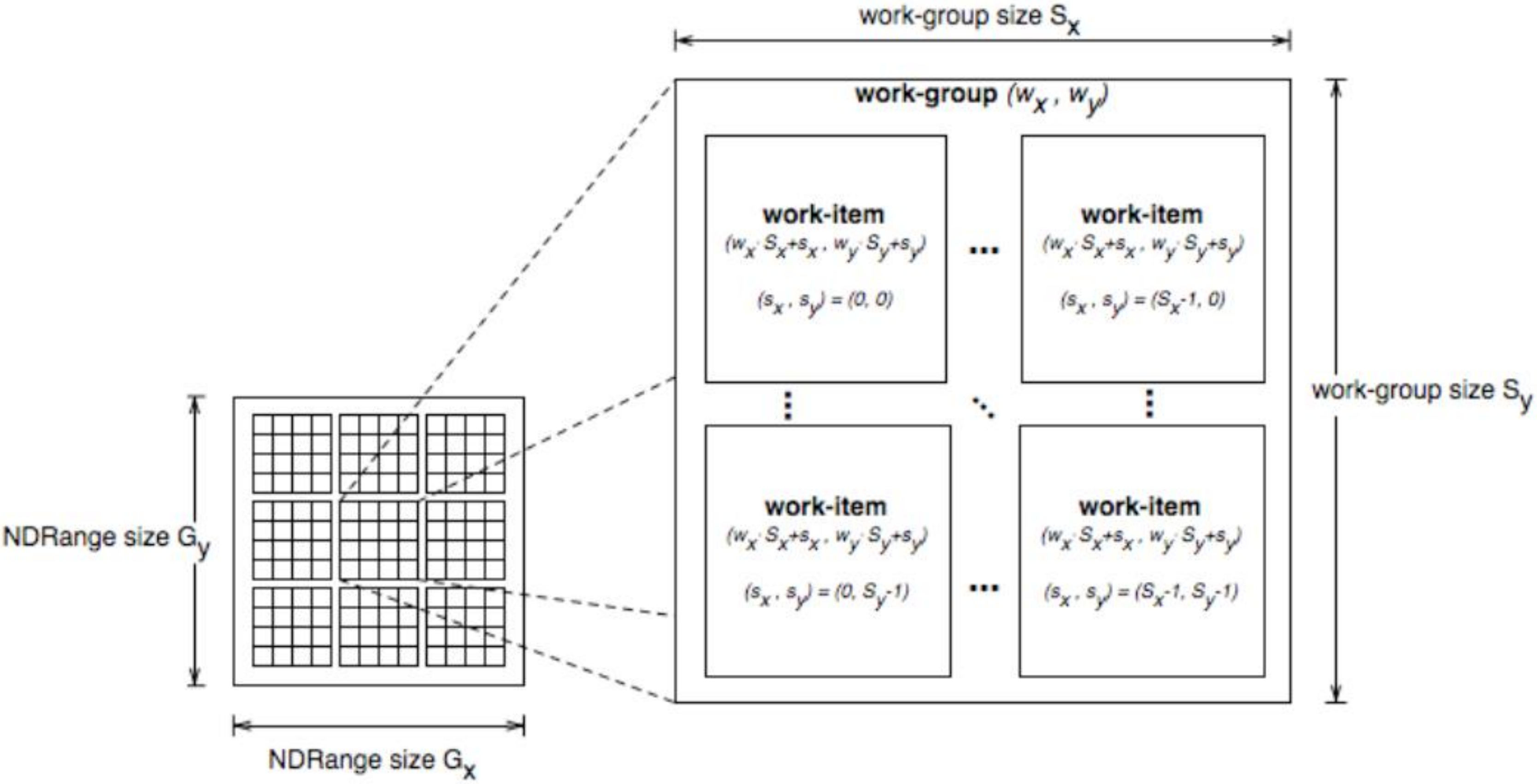


OpenCL

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

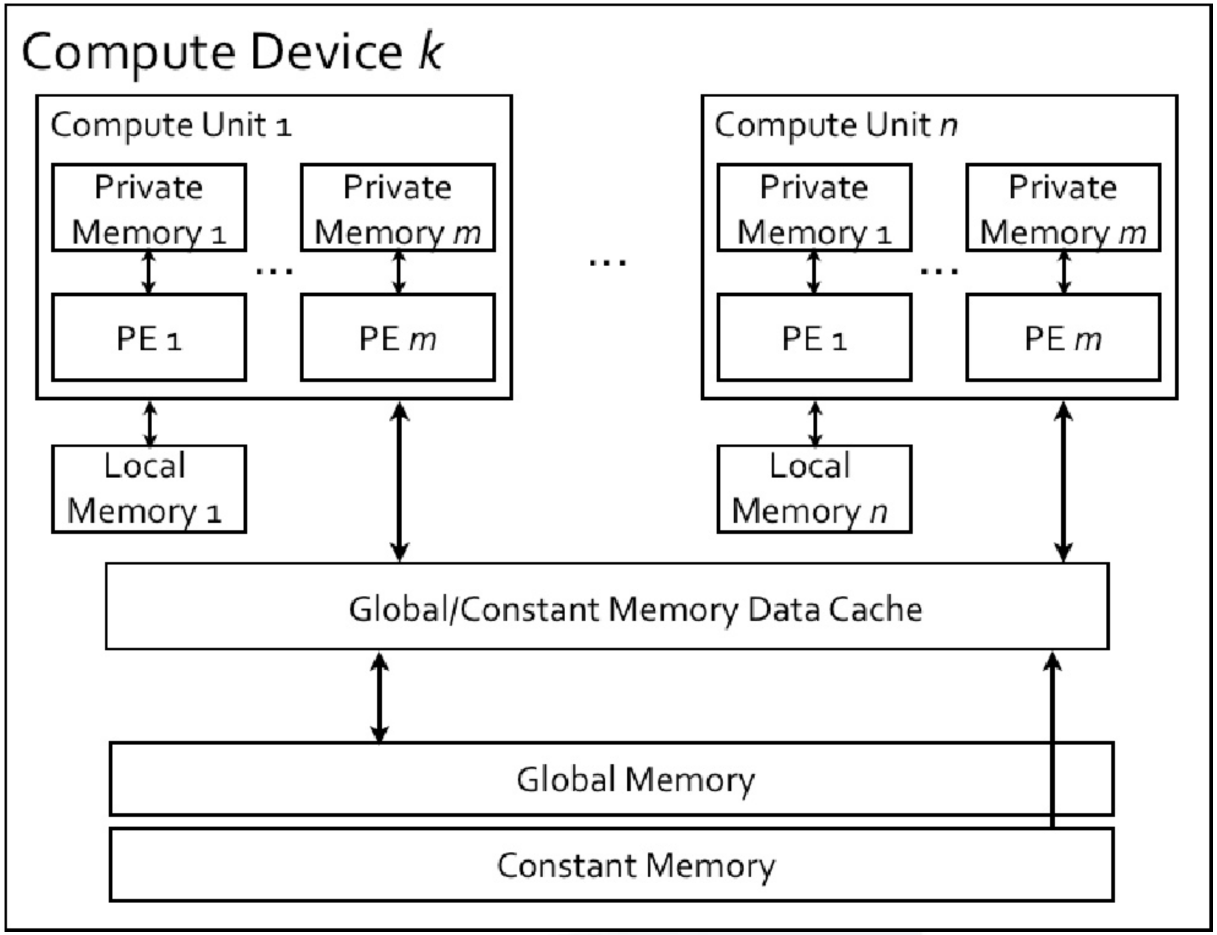
Logical organization of work

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


work-group size S_x
work-group (w_x, w_y)
work-item
$(w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
$(s_x, s_y) = (0, 0)$
...
work-item
$(w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
$(s_x, s_y) = (S_x - 1, 0)$
work-group size S_y
work-item
$(w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
$(s_x, s_y) = (0, S_y - 1)$
...
work-item
$(w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
$(s_x, s_y) = (S_x - 1, S_y - 1)$
NDRange size G_y
NDRange size G_x

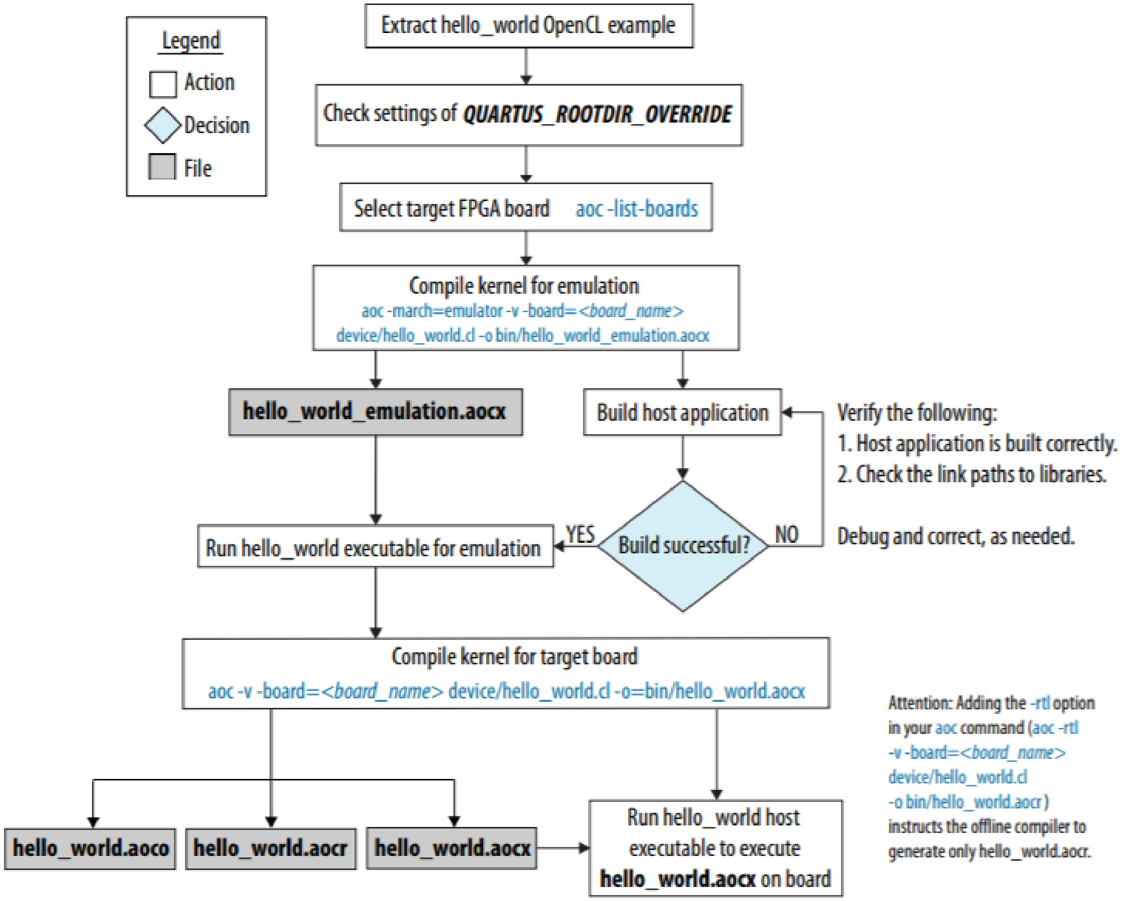
Conceptual OpenCL device architecture

- Host is not shown



The Intel FPGA OpenCL flow

- You need a Board Support Package



Legend
□ Action
◇ Decision
▨ File

Extract hello_world OpenCL example

Check settings of ***QUARTUS_ROOTDIR_OVERRIDE***

Select target FPGA board aoc -list-boards

Compile kernel for emulation
aoc -march=emulator -v -board=<*board_name*> device/hello_world.cl -o bin/hello_world_emulation.aocx

hello_world_emulation.aocx

Build host application

Build successful?

YES

NO

Verify the following:
1. Host application is built correctly.
2. Check the link paths to libraries.

Debug and correct, as needed.

Run hello_world executable for emulation

Compile kernel for target board
aoc -v -board=<*board_name*> device/hello_world.cl -o=bin/hello_world.aocx

hello_world.aoco **hello_world.aocr** **hello_world.aocx**

Run hello_world host executable to execute **hello_world.aocx** on board

Attention: Adding the -rtl option in your aoc command (aoc -rtl -v -board=<*board_name*> device/hello_world.cl -o bin/hello_world.aocr) instructs the offline compiler to generate only hello_world.aocr.

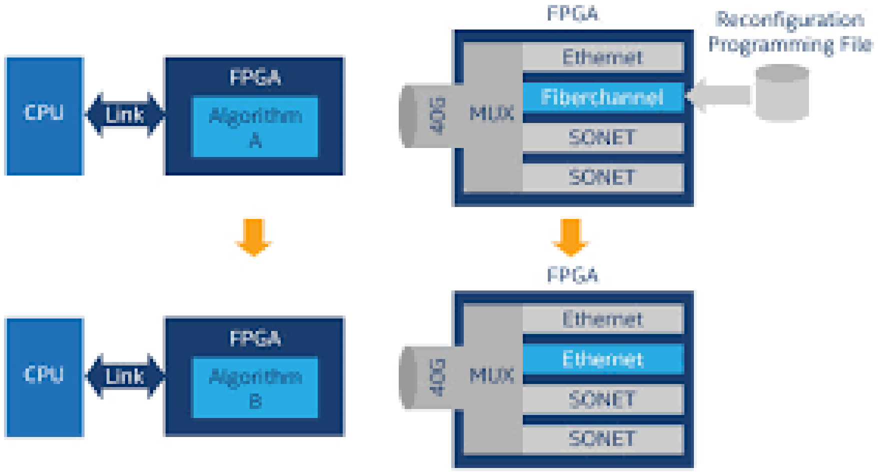
The Intel FPGA OpenCL Flow

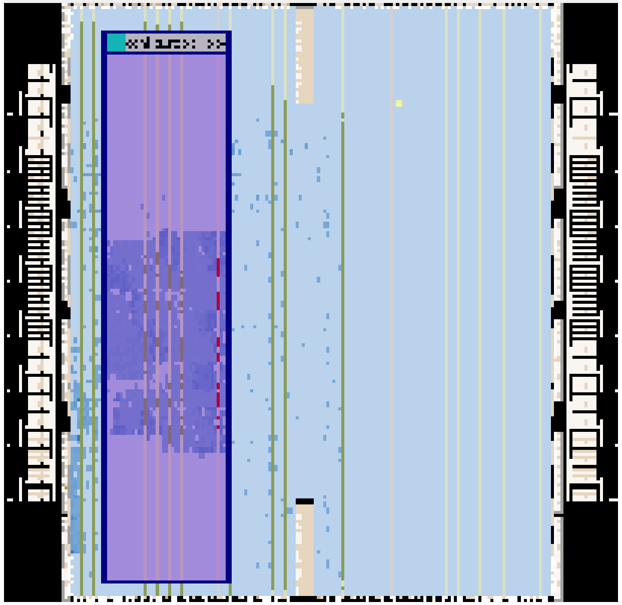
- You have **MUCH MORE** freedom when you design hardware
- OpenCL
  - Allows fast Design Space Exploration
  - Easier to code than HDL (Verilog, VHDL)

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

The Intel FPGA OpenCL Flow

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



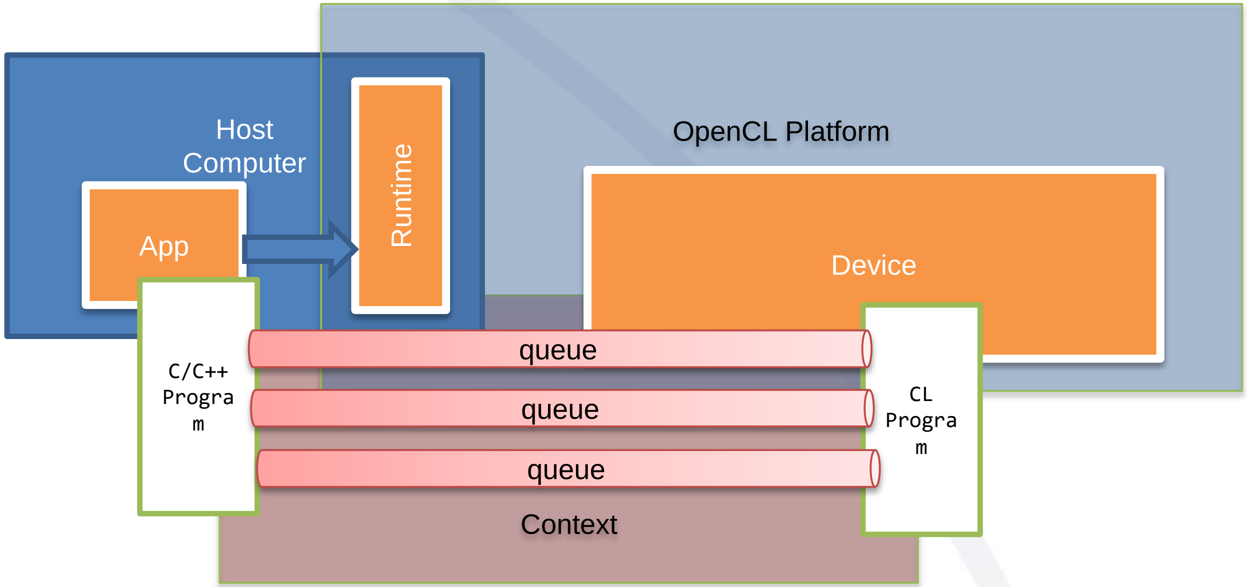


The Host API

Basics

- You start from C code
- There is a API
  - –some headers files (CL/cl.h)
  - –A dinyamic library (libOpenCL.so in Linux, OpenCL.dll in Windows)

- There is a user/provider architecture
  - –Several providers can implement the OpenCL API.
  - –Those provide something call ICD (Installable Client Driver)
    - •Intel has an ICD for (Intel OpenCL SDK)
    - •NVIDIA has an ICD for (CUDA based OpenCL)
    - •Cygwin has some POCL
  - –In Linux & Cygwin the providers are listed in /etc/OpenCL/vendors


Host
Computer
App
Runtime
OpenCL Platform
Device
C/C++
Progra
m
queue
queue
queue
CL
Progra
m
Context

Platforms

- The Platform Concept
  - –It is a “host” system connected to a number of devices
  - –It can be used to export different methods on the same physical machine
- Relevant functions
  - –clGetPlatformIDs
  - –clGetPlatformInfo

Example

```
void OpenCL_Interface::selectPlatform(cl_int m_platform_id)
{
    cl_int status;
    cl_uint num_platforms;
    int id_platform = m_platform_id;


    /* Get Platform Info */
    status = clGetPlatformIDs(0, nullptr, &num_platforms);
    checkError(status, "Error calling clGetPlatformIDs");
    printf("\n  - Platforms (%d):", num_platforms);

    cl_platform_id platforms[num_platforms];

    status = clGetPlatformIDs(num_platforms, platforms, nullptr);
    checkError(status, "Error calling clGetPlatformIDs");

    for (cl_uint i = 0; i < num_platforms; ++i)
    {
        // Get the length for the i-th platform name
        size_t platform_name_length = 0;
        status = clGetPlatformInfo(platforms[i], CL_PLATFORM_NAME, 0, 
nullptr, &platform_name_length);
        checkError(status, "Error calling clGetPlatformInfo");

        // Get the name itself for the i-th platform
        // use vector for automatic memory management
        char platform_name[platform_name_length];
        status = clGetPlatformInfo(platforms[i], CL_PLATFORM_NAME, 
platform_name_length, platform_name, nullptr);
        checkError(status, "Error calling clGetPlatformInfo");

        printf("\n\r    [%d] %s", i, platform_name);
        if (id_platform == i)
        {
            printf(" [Selected]");

            if (strstr(platform_name, "FPGA") != NULL)
                m_isFpga = true;

        }

        fflush(stdout);
    }
    printf("\n");
    m_platform = platforms[id_platform];


    char char_buffer[STRING_BUFFER_LEN];
    printf("Detected OpenCL platforms: %d\n", num_platforms);
    printf("Querying platform for info:\n");
    printf("%42s\n", "==========================================");
    clGetPlatformInfo(m_platform, CL_PLATFORM_NAME, 
STRING_BUFFER_LEN, char_buffer, NULL);
    printf("%-40s = %s\n", "CL_PLATFORM_NAME", char_buffer);
    clGetPlatformInfo(m_platform, CL_PLATFORM_VENDOR, 
STRING_BUFFER_LEN, char_buffer, NULL);
    printf("%-40s = %s\n", "CL_PLATFORM_VENDOR ", char_buffer);
    clGetPlatformInfo(m_platform, CL_PLATFORM_VERSION, 
STRING_BUFFER_LEN, char_buffer, NULL);
    printf("%-40s = %s\n", "CL_PLATFORM_VERSION ", char_buffer);

    printf("\n");
}
```

Devices

- The Device Concept
  - A particular device inside the platform

Example

```
void OpenCL_Interface::selectDevice()
{
    cl_int status;
    int id_device = 0;
    cl_uint num_devices;

    /* Get Device Info */
    status = clGetDeviceIDs(m_platform, CL_DEVICE_TYPE_ALL, 0, nullptr, &num_devices);
    checkError(status, "Error calling clGetDeviceIDs");
    printf("  - Devices (%d):",  num_devices);

    std::vector<cl_device_id> devices(num_devices);

    status = clGetDeviceIDs(m_platform, CL_DEVICE_TYPE_ALL, num_devices, &devices[0], nullptr);
    checkError(status, "Error calling clGetDeviceIDs");

    for (cl_uint i = 0; i < num_devices; ++i)
    {
        // Get the length for the i-th device name
        size_t device_name_length = 0;
        status = clGetDeviceInfo(devices[i], CL_DEVICE_NAME, 0, nullptr, &device_name_length);
        checkError(status, "Error calling clGetDeviceInfo");

        // Get the name itself for the i-th device
        // use vector for automatic memory management
        char device_name[device_name_length];
        status = clGetDeviceInfo(devices[i], CL_DEVICE_NAME, device_name_length, device_name, nullptr);
        checkError(status, "Error calling clGetDeviceInfo");

        printf("\n\r    [%d] %s", i, device_name);

        if (id_device == i)
            printf(" [Selected]");

        fflush(stdout);
    }
    printf("\n");
    m_device = devices[id_device];

}
```

Context

- It groups the objects to talk to a device (queues, memories, etc.)

Example

```
m_context = clCreateContext(NULL, 1, &m_device, NULL, NULL, &status);
checkError(status, "Failed to create Context");
```

Program

- This is the program that will be executed in the Device
- You have 2 options to create a program
  - –From Source Code

```
cl_program clCreateProgramWithSource (cl_context context,
    cl_uint count, const char **strings,
    const size_t *lengths, cl_int *errcode_ret)
```

    - •Standard method in GPUs
    - •We usually store the program in .cl files
  - –From Binary
    - **clCreateProgramWithBinary**
    - •Stardard method in FPGAs (because compilation is soooo slow)
  - –Programs must be compiled

Creating a program from Source

```
void OpenCL_Interface::createProgramFromSource()
{
    FILE *fp;
    char *source_str;
    size_t source_size;
    cl_int status;


    // Load the source code containing the kernel
    fp = fopen(m_kernel_file.c_str(), "r");
    if (!fp)
    {
        std::cerr << "Failed to load kernel!" << m_kernel_file.c_str() << std::endl;
        exit(EXIT_FAILURE);
    }

    source_str = (char*) malloc(MAX_SOURCE_SIZE);
    source_size = fread(source_str, 1, MAX_SOURCE_SIZE, fp);
    fclose(fp);

    // Create Kernel Program from the source
    m_program = clCreateProgramWithSource(gContext, 1, (const char **) &source_str, (const size_t *) &source_size,
&status);
    checkError(status, "Failed to create Program with Source");

    free(source_str); //malloc of source code
}
```

Kernels

- A kernel is a function of the Device program which is accessible from the host
  - –identified by __kernel word
- An example syntax

```
__kernel void add_kernel(__global int* in_array_a, __global int* in_array_b,
        __global int* in_array_r, int n)
{
        for (int i=0; i < n; i++)
                r[i] = a[i]+b[i];
}
```

- The Host program need to get the kernel identifiers with

```
cl_kernel clCreateKernel (cl_program  program,
        const char *kernel_name,            cl_int *errcode_ret);
```

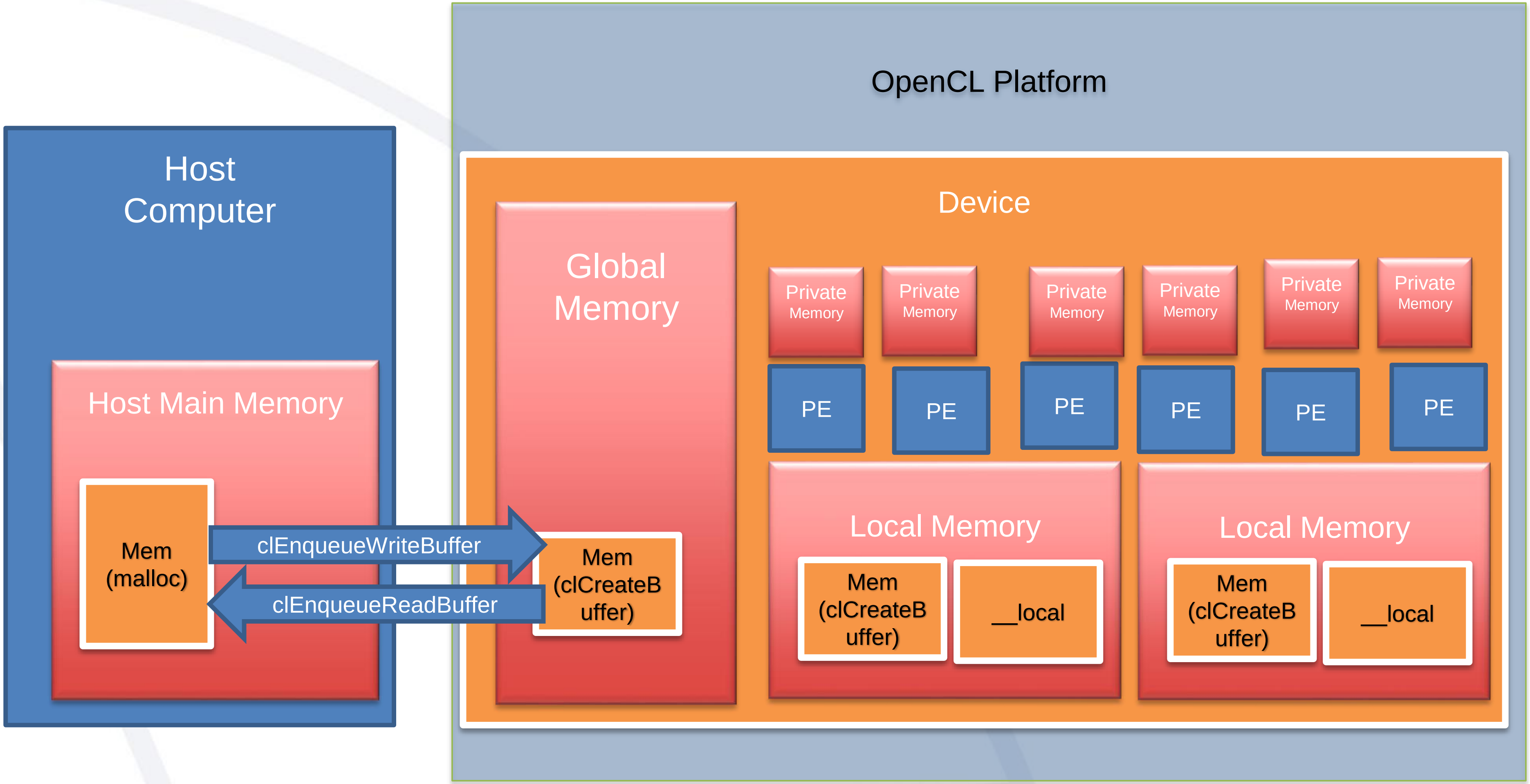
Example of creating program a

```
....

char* options = “”;

status = clBuildProgram(gProgram, 1, &gDevice, options, NULL, NULL);

if (status != CL_SUCCESS)
{
    size_t len;
    char buffer[4096];
    clGetProgramBuildInfo(gProgram, gDevice, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer, &len);
    std::cout << buffer << std::endl;
    fflush(stdout);
    checkError(status, "Failed to Build Program");
}


/* Create OpenCL Kernel/s */
string kernel_name = “add_kernel”;

gKernel = clCreateKernel(gProgram, m_kernel_name.c_str(), &status);

char msg[100];
sprintf(msg, "Failed to Create Kernel %s", kernel_name.c_str());
checkError(status && gKernel, msg);


printf("[ OK ]\n");
```

Queue

- a pipe to send/receive information to/from a kernel
- you can have several queues

Memory


OpenCL Platform
Host
Computer
Host Main Memory
Mem
(malloc)
clEnqueueWriteBuffer
clEnqueueReadBuffer
Device
Global
Memory
Mem
(clCreateB
uffer)
Private
Memory
Private
Memory
Private
Memory
Private
Memory
Private
Memory
Private
Memory
PE
PE
PE
PE
PE
PE
Local Memory
Mem
(clCreateB
uffer)
__local
Local Memory
Mem
(clCreateB
uffer)
__local

Global Memory Creation

- Creates memory into the device

```
cl_mem clCreateBuffer (      cl_context context,
      cl_mem_flags flags,
      size_t size,
      void *host_ptr,
      cl_int *errcode_ret)
```

Transferring Memory

- Transfer byte from host to device

```
cl_int clEnqueueWriteBuffer (      cl_command_queue command_queue,
           cl_mem buffer,
           cl_bool blocking_write,   size_t offset,      size_t cb,           const void
*ptr,
           cl_uint num_events_in_wait_list,   const cl_event *event_wait_list,
           cl_event *event)



cl_int clEnqueueReadBuffer (       cl_command_queue command_queue,
           cl_mem buffer,
           cl_bool blocking_read,    size_t offset,      size_t cb,           void *ptr,
           cl_uint num_events_in_wait_list,   const cl_event *event_wait_list,
           cl_event *event)
```

Kernel Invocation

- Argument Setting

- Invoking kernel execution

cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue,
cl_kernel kernel, cl_uint work_dim,
const size_t *global_work_offset, const size_t *global_work_size,
const size_t *local_work_size, cl_uint num_events_in_wait_list,
const cl_event *event_wait_list, cl_event *event)

Wort Items and Workgroups

- The work units in OpenCL are called Work-items
  - –They are equivalent to CUDA threads.
- Work-items can be grouped in Workgroups.
  - –Workgroups share local memory.
  - –They can be synchronized by barriers

- Work items are identified by IDs.
  - –global_id is unique at the global scale
  - –local_id is unique at the workgroup scale
- The workgroup size defined explicitelly with the local size, but the number of workgroups is inferred from global and local dimensions when calling clEnqueueNDRangeKernel

  - –Num Workgroups= GlobalSize / LocalSize

Wort Items and Workgroups

- get_global_id(dim) (gid)
  - returns the the global ID in the dim dimension (we can have up to 3 dimensions)
- get_local_id(dim) (lid)
  - returns the the local ID in the dim dimension (we can have up to 3 dimensions)
- get_group_id(dim) (wid)
  - returns the the work-group ID in the dim dimension (we can have up to 3 dimensions)

gid = wid * wsz + lid

- The workgroup size is not defined explicitelly but inferred by global and local dimensions when calling clEnqueueNDRangeKernel
  - Number of Workgroups= GlobalSize / LocalSize

The Kernel API

Kernel Code

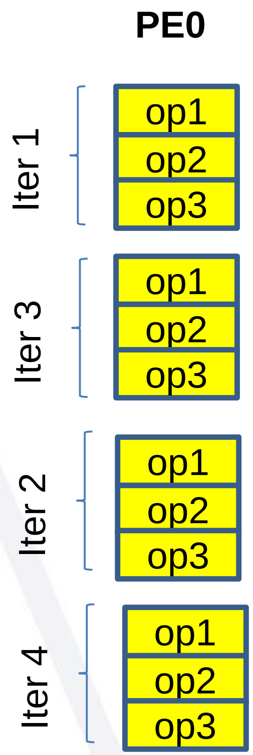
- Just regular C
- Functions
  - To be a kernel you just add __kernel before function declaration
  - Kernels functions are void (no return parameter)

- NO STACK -> ALWAYS INLINING

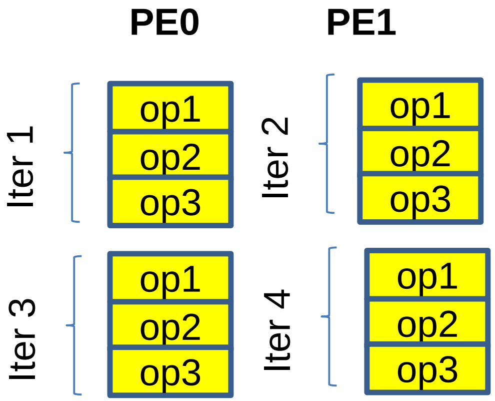
- Tricks are played with kernel APIs (get_global_id, barriers, etc) , #pragma and more...

Loop Unrolling and Loop pipelining

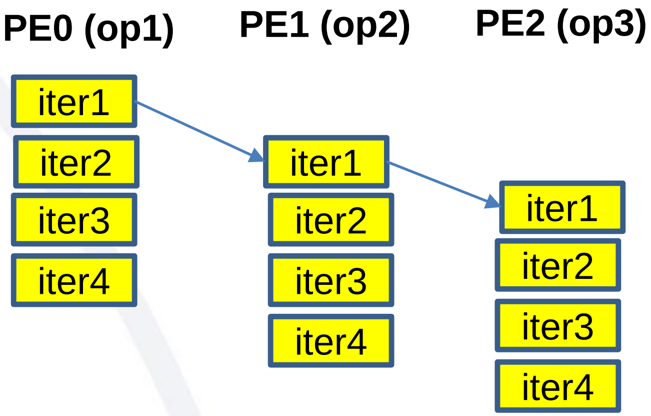
```
for each (iter ){
     op1;
     op2;
     op3;
}
```



- Unrolling



- Pipelining


PE0 (op1)
PE1 (op2)
PE2 (op3)
iter1
iter2
iter3
iter4
iter1
iter2
iter3
iter4
iter1
iter2
iter3
iter4