RAW
Reconfigurable Architectures Workshop

IPDPS 2018 VANCOUVER

POLITECNICO MILANO 1863
NECST laboratory

POLITECNICO MILANO 1863

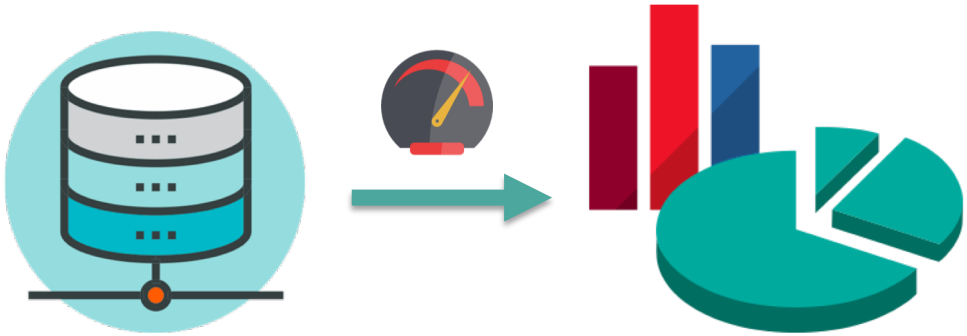
05/21/2018

FIDA: a framework to automatically integrate FPGA kernels within Data-Science applications

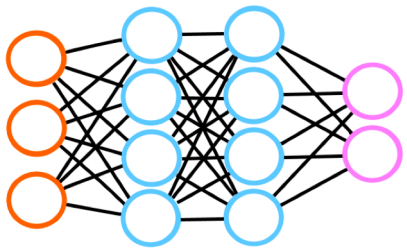
Luca Stornaiuolo
Dipartimento di Elettronica Informazione e Bioingegneria (DEIB)
luca.stornaiuolo@polimi.it

Alberto Parravicini, Donatella Sciuto and Marco Santambrogio

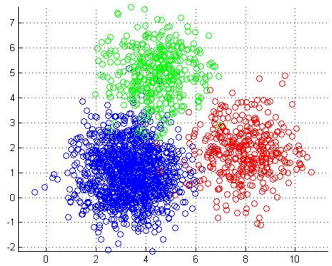
Context Definition



Huge amount of data that need to be processed to get aggregated information



Machine Learning
(e.g. Neural Network)



Data Mining
(e.g. K-means)

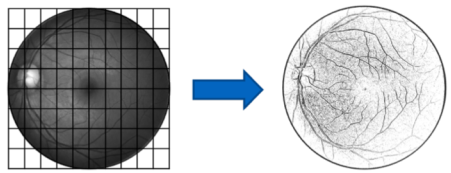
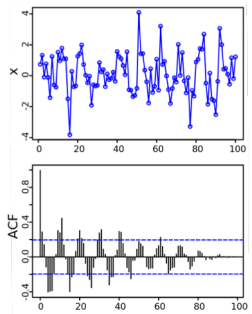


Image Analysis
(e.g. Vessel segmentation)



Signal Processing
(e.g. Autocorrelation)

Problem Definition

RAW
Reconfigurable Architectures Workshop

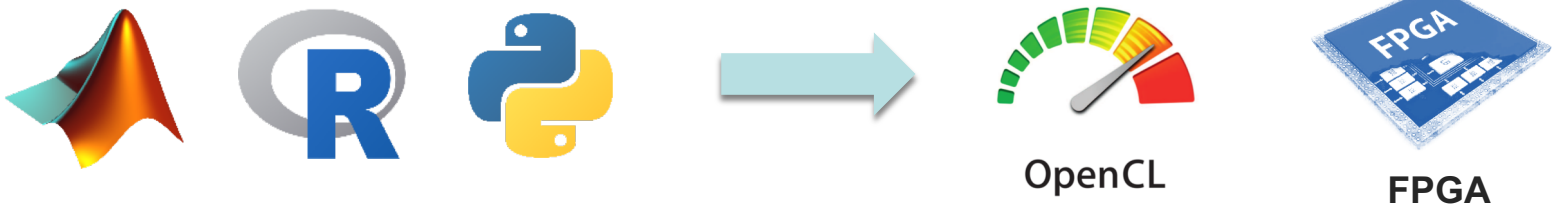
- Most computational pipelines rely on **high-level languages** like Python, R and MATLAB.
- These pipelines can be accelerated with **OpenCL** and **FPGAs**.
- Integrating OpenCL kernels with high-level languages is slow and error-prone.

OpenCL

FPGA

Proposed Solution

- Integrating OpenCL kernels with high-level languages is slow and error-prone.
- **FIDA automatize this process!**
  From a **simple description of the kernel**, we create interfaces for Python, R and MATLAB.



From this…

RAW
Reconfigurable Architectures Workshop

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <CL/opencl.h>
#include <time.h>
#include <sys/stat.h>
#include <fcntl.h>    /* For O_RDWR, O_WRONLY */
#include <unistd.h>   /* For open(), read(), ... */
#include <sys/time.h>
#include <iostream>
#include <ctime>
#include <fstream>


////////////////////////////////////////////////////////////////////////////////

int load_file_to_memory(const char *filename, char **result) {
    size_t size = 0;
    FILE *f = fopen(filename, "rb");
    if (f == NULL) {
        *result = NULL;
        return -1; // -1 means file opening fail
    }
    fseek(f, 0, SEEK_END);
    size = ftell(f);
    fseek(f, 0, SEEK_SET);
    *result = (char *) malloc(size + 1);
    if (size != fread(*result, sizeof(char), size, f)) {
        free(*result);
        return -2; // -2 means file reading fail
    }
    fclose(f);
    (*result)[size] = 0;
    return size;
}

////////////////////////////////////////////////////////////////////////////////

int main(int argc, char** argv)
{
#if defined(SDA_PLATFORM) && !defined(TARGET_DEVICE)
#define STR_VALUE(arg)      #arg
#define GET_STRING(name) STR_VALUE(name)
#define TARGET_DEVICE GET_STRING(SDA_PLATFORM)
#endif

    // Used to measure execution time;
    timespec ts_start;
    timespec ts_end;
    clock_gettime(CLOCK_REALTIME, &ts_start);

    char *TARGET_DEVICES[] = {__INSERT_TARGET_DEVICES__};
    char *XCLBIN_FILES[] = {__INSERT_XCLBIN_FILES__};
    int NUM_SUPPORTED_DEVICES = sizeof(TARGET_DEVICES) / sizeof(char *);
    char *xclbin;

    int err;                            // error code returned from api calls

    // Inputs;
    __INSERT_INPUTS__
    // Outputs;
    __INSERT_OUTPUTS__

    cl_platform_id platforms[16];       // platform id
    cl_platform_id platform_id;         // platform id
    cl_uint platform_count;
    cl_device_id device_id;             // compute device id
    cl_context context;                 // compute context
    cl_command_queue commands;          // compute command queue
    cl_program program;                 // compute program
    cl_kernel kernel;                   // compute kernel

    char cl_platform_vendor[1001];


    // Get all platforms and then select Xilinx platform
    err = clGetPlatformIDs(16, platforms, &platform_count);
    if (err != CL_SUCCESS)
    {
        printf("Error: Failed to find an OpenCL platform!\n");
        printf("Test failed with err: %d\n", err);
        return EXIT_FAILURE;
    }
    printf("INFO: Found %d platforms\n", platform_count);
```

```
// Find Xilinx Platform
int platform_found = 0;
for (unsigned int iplat = 0; iplat < platform_count; iplat++)
{
    err = clGetPlatformInfo(platforms[iplat], CL_PLATFORM_VENDOR, 1000, (void *) cl_platform_vendor, NULL);

    if (err != CL_SUCCESS)
    {
        printf("Error: clGetPlatformInfo(CL_PLATFORM_VENDOR) failed!\n");
        return EXIT_FAILURE;
    }
    if (strcmp(cl_platform_vendor, "Xilinx") == 0)
    {
        printf("INFO: Selected platform %d from %s\n", iplat, cl_platform_vendor);
        platform_id = platforms[iplat];
        platform_found = 1;
        break;
    }
}
if (!platform_found)
{
    printf("ERROR: Platform Xilinx not found. Exit.\n");
    return EXIT_FAILURE;
}


// Connect to a compute device.
// Find all devices and then select the target device
cl_device_id devices[16];  // compute device id
cl_uint device_count;
unsigned int device_found = 0;
char cl_device_name[1001];
err = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_ACCELERATOR, 16, devices, &device_count);
if (err != CL_SUCCESS)
{
    printf("Error: Failed to create a device group!\n");
    return EXIT_FAILURE;
}


// Iterate all devices to select the target device.
for (uint i = 0; i < device_count; i++)
{
    err = clGetDeviceInfo(devices[i], CL_DEVICE_NAME, 1024, cl_device_name, 0);

    if (err != CL_SUCCESS)
    {
        printf("Error: Failed to get device name for device %d!\n", i);
        return EXIT_FAILURE;
    }
    printf("INFO: Analyzing device: %s\n", cl_device_name);
    for (int t = 0; t < NUM_SUPPORTED_DEVICES; t++)
    {
        if (strcmp(cl_device_name, TARGET_DEVICES[t]) == 0)
        {
            device_id = devices[i];
            device_found = 1;
            xclbin = XCLBIN_FILES[t];
            printf("INFO: Selected %s as the target device\n", cl_device_name);
            break;
        }
    }
}
if (!device_found) {
    printf("ERROR: Target device not found. Exit.\n");
    return EXIT_FAILURE;
}


// Create OpenCL context
context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
if (!context) {
    printf("Error: Failed to create a compute context!\n");
    return EXIT_FAILURE;
}


// Create Command Queue
commands = clCreateCommandQueue(context, device_id, 0, &err);
if (!commands)
{
    printf("Error: Failed to create a command commands!\n");
```

```
// Create Program Objects
int status;

// Load binary from disk
unsigned char *kernelbinary;
printf("INFO: Loading %s\n", xclbin);
int n_i = load_file_to_memory(xclbin, (char **) &kernelbinary);
if (n_i < 0)
{
    printf("Failed to load kernel from xclbin: %s\n", xclbin);
    return EXIT_FAILURE;
}

size_t n = n_i;
// Create the compute program from offline
program = clCreateProgramWithBinary(context, 1, &device_id, &n, (const unsigned char **) &kernelbinary, &status, &err);
if ((!program) || (err != CL_SUCCESS))
{
    printf("Error: Failed to create compute program from binary %d!\n", err);
    return EXIT_FAILURE;
}


// Create the compute kernel in the program we wish to run
kernel = clCreateKernel(program, __INSERT_KERNEL_NAME__, &err);
if (!kernel || err != CL_SUCCESS)
{
    printf("Error: Failed to create compute kernel!\n");
    return EXIT_FAILURE;
}


// FIFOs in the /tmp/ folder;
const char *myfifo_in = "/tmp/myfifo_in";
const char *myfifo_out = "/tmp/myfifo_out";

// Creating the FIFOs;
printf("Receiver - Creating FIFOs...\n");
mkfifo(myfifo_in, 0666);
mkfifo(myfifo_out, 0666);

// Opening connections to the FIFOs;
printf("Receiver - Opening FIFOs...\n");
int fifo_in = open(myfifo_in, O_RDWR);
int fifo_out = open(myfifo_out, O_RDWR);

clock_gettime(CLOCK_REALTIME, &ts_end);
double exec_time = (double)(ts_end.tv_nsec - ts_start.tv_nsec) / 1000000000;
std::cout << "--> FPGA SETUP TIME=" << exec_time << std::endl;

// Store the execution time;
std::ofstream output_file;
output_file.open("../../../data/host.csv", std::ios_base::app);
output_file << "__INSERT_KERNEL_NAME_T__, , board_setup, " << exec_time  << std::endl;
output_file.close();

// Start the main kernel computation;
__INSERT_LOOP_START__

// Read values from the FIFOs;
printf("--> Receiver - Reading input data...\n");
ssize_t res_trans;

// Kernel inputs;
__INSERT_CL_INPUTS__
// Kernel outputs;
__INSERT_CL_OUTPUTS__


// Allocate inputs and read them from the FIFOs;
__INSERT_FIFO_INPUT__
// Allocate output arrays, if present;
__INSERT_FIFO_OUTPUT_ALLOC__

// Measure memory transfer to the FPGA;
clock_gettime(CLOCK_REALTIME, &ts_start);

// Create the inputs and outputs in device memory for our calculation;
__INSERT_CL_BUFFERS__
```

IPDPS 2018 VANCOUVER

POLITECNICO MILANO 1863 NECST laboratory

POLITECNICO MILANO 1863

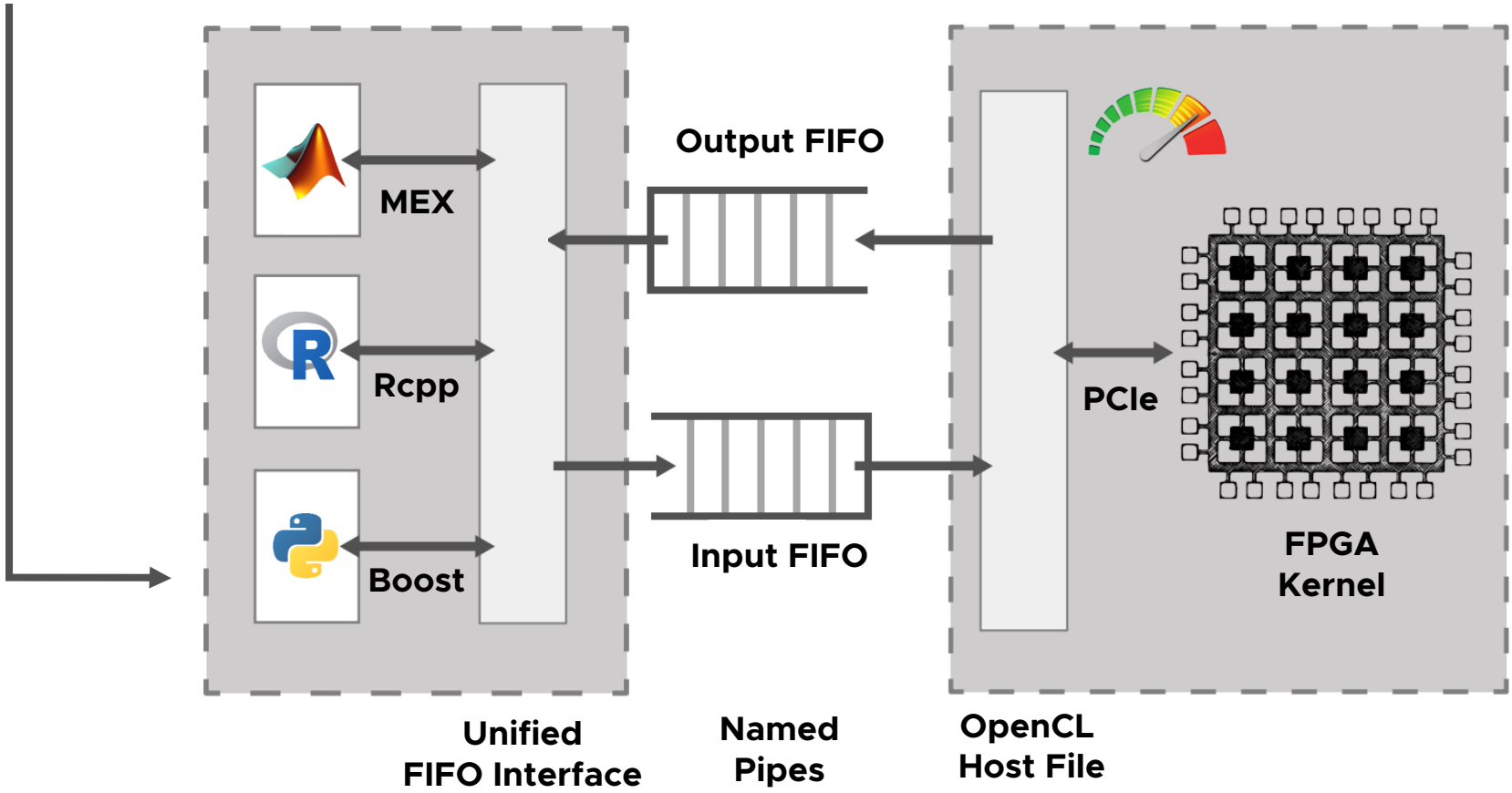
…to this!

```
{
  "kernel_name": "mmult",
  "board": ["xilinx_adm-pcie-7v3_1ddr_3_0"],
  "xclbin": ["kernel_7v3.xclbin"],
  "num_iterations": 3,
  "inputs": [

      {"type": "array",
      "name": "a",
      "length": 256,
      "class": "int",
      "position": 0},

      {"type": "array",
      "name": "b",
      "length":256,
      "class": "int",
      "position": 1}],

  "outputs": [

      {"type": "array",
      "name": "c",
      "length":256,
      "class": "int",
      "position": 2}]
}
```

Architecture

RAW
Reconfigurable Architectures Workshop

```
var = python_interface.var_fpga(x, len(x))
```



Experimental Results

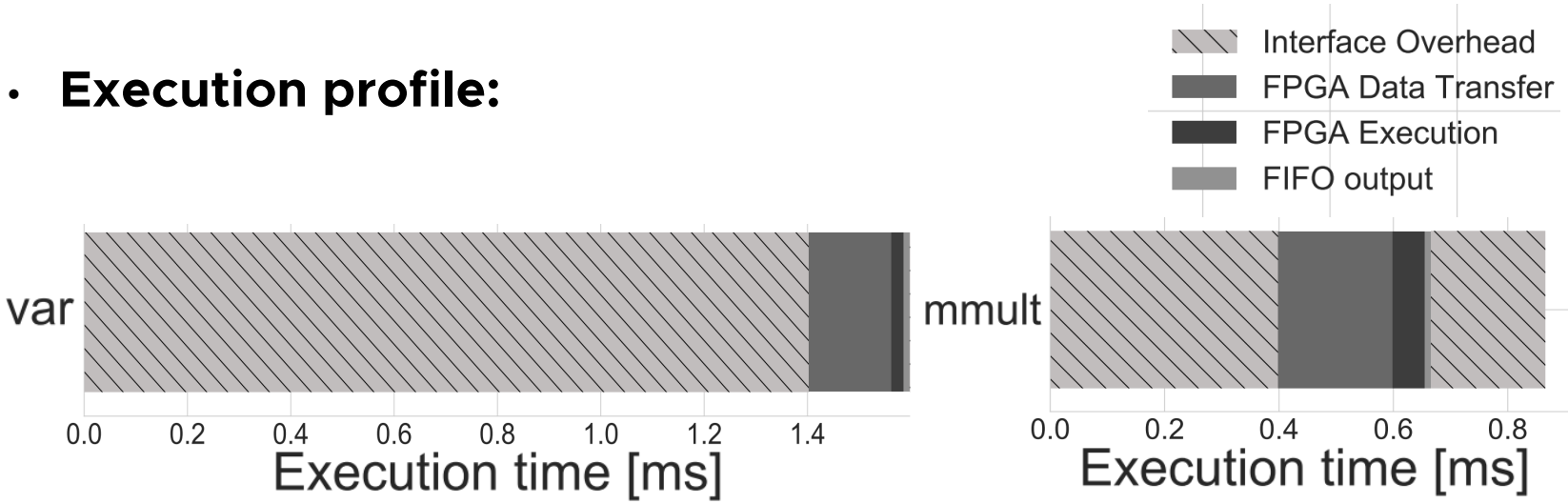
RAW
Reconfigurable Architectures Workshop

- We tested our architecture on different computational kernels.
- We profiled executions to find where bottlenecks are.
- We analyzed introduced execution time overheads.

Experimental Results

RAW
Reconfigurable Architectures Workshop

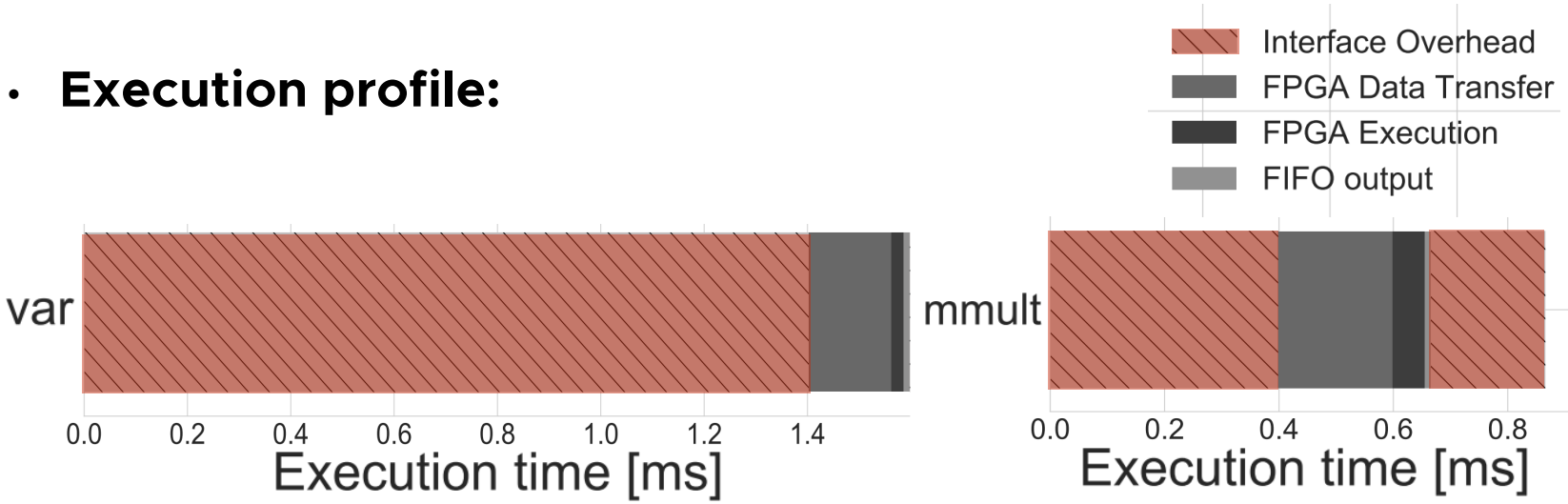
- **Execution profile:**



Experimental Results

RAW
Reconfigurable Architectures Workshop

- **Execution profile:**



- Most of the time is spent in interprocess communication:
  shared memory could drastically improve performance!

RAW
Reconfigurable Architectures Workshop

IPDPS 2018 VANCOUVER

POLITECNICO MILANO 1863
NECST laboratory

POLITECNICO MILANO 1863

05/21/2018

FIDA: a framework to automatically integrate FPGA kernels within Data-Science applications

https://necst.it/
https://www.slideshare.net/necstlab
https://www.facebook.com/groups/ReconfigurableArchitecturesWorkshop/

Luca Stornaiuolo
Dipartimento di Elettronica Informazione e Bioingegneria (DEIB)
luca.stornaiuolo@polimi.it

Alberto Parravicini, Donatella Sciuto and Marco Santambrogio