Benchmark FFT using GPU and OpenCL

In this example we will create a random NxN matrix using uniform distribution and find the time needed to calculate a 2D FFT of that matrix. The calculation will be done using GPU card and OpenCL with a group of MatDeck functions that incorporate ArrayFire functionalities.

First, we will set the environment to use the GPU for calculations. Using the function, afp_supported_backends, a list of all supported backends that can be used for calculations will be produced. In our case, calculations can be made on the CPU, using OpenCL or CUDA framework.

$$\text{afp\_supported\_backends}() = \begin{bmatrix} \text{"cpu"} \\ \text{"opencl"} \\ \text{"cuda"} \end{bmatrix}$$

Default environment for calculations is the CPU, we can change the current environment with the function, afp_set_backend, and check which environment is currently in use with the afp_backend function.

$$\text{afp\_set\_backend}(\text{"opencl"}) = \text{true}$$

$$\text{afp\_backend}() = \text{"opencl"}$$

In each environment, there can be several devices which support calculations within it. To check the number of devices which supports calculations in the current environment, use the function, afp_get_device_count, and the functions afp_get_device and afp_set_device to check/change current device.

$$\text{afp\_get\_device\_count}() = 3$$

$$\text{afp\_get\_device}() = 1$$

$$\text{afp\_set\_device}(1) = \text{true}$$

To display information about currently selected devices, use the function afp_device_info

$$\text{afp\_device\_info}() = \begin{bmatrix} \text{"Intel(R)\_HD\_Graphics\_620"} \\ \text{"OpenCL"} \\ \text{"Intel(R) OpenCL"} \\ \text{"2.1"} \end{bmatrix}$$

Finally, we have set the OpenCL as a calculation backend and set the device with number 1 - integrated Intel graphic card as a device on which we will do all calculations.

Six iterations will be done to create a uniformly random NxN matrix with real values, calculate the 2D FFT calculation time and Gigaflops benchmark in each iteration. Each iteration will have a different input matrix size and the summary of the calculation will be displayed in the console window.

In the following code, we will create a function bench() that will do all the calculations that we have described.

```
bench()
{
  print("Benchmar N x N 2D FFT:\n")
  for(M := 7; M <= 12; M += 1)
  {
    N := 1 << M
    print(to_string(N) + " x " + to_string(N) + "input matrix size")
    A := afp_randu(N, N, "real")
    a := timenow()
    afp_fft2(A, 1, N, N)
    b := timenow()
    time := b - a
    gflops := 10 * N * N * M / (time * 1000000000)
    print(" - Time: " + to_string(time))
    print(" - Gflops: " + to_string(gflops) + "\n")
  }
}
```

Now, when the benchmark function is ready, all we have to do is to call the bench() function and analyze the printed console results.

```
bench()
```