

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.

```
afp_supported_backends( ) = [ "cpu"  
                             "opencl"  
                             "cuda" ]
```

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.

```
afp_set_backend("opencl") = true  
afp_backend( ) = "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.

```
afp_get_device_count( ) = 3  
afp_get_device( ) = 1  
afp_set_device(1) = true
```

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

```
afp_device_info( ) = [ "Intel(R)_HD_Graphics_620"  
                       "OpenCL"  
                       "Intel(R) OpenCL"  
                       "2.1" ]
```

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.

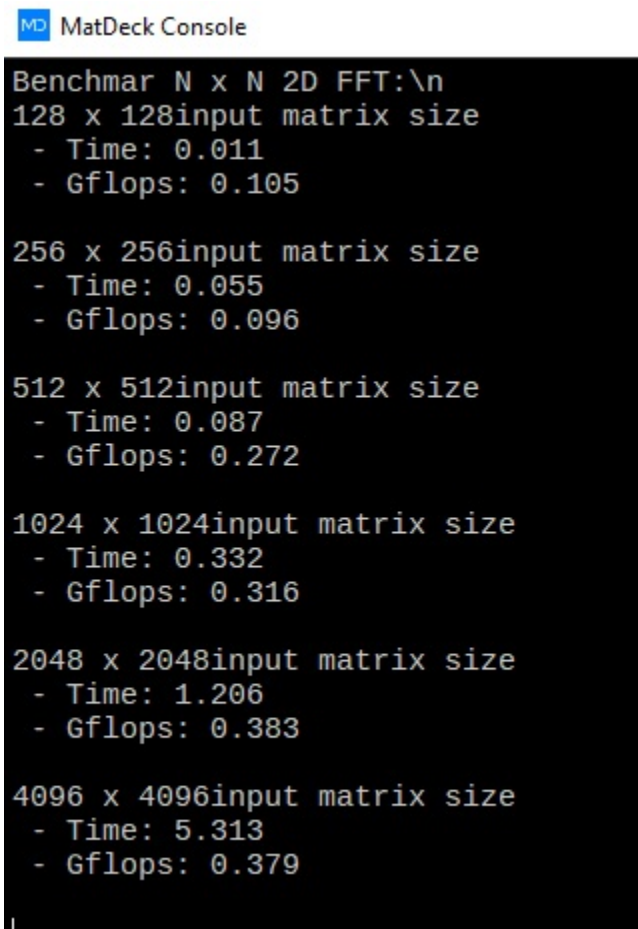
Six iterations will be done to create a uniformly random NxN matrix with real values, calculate the 2D FFT calculation time and GigaFlops benchmark in each iteration. Each iteration will have a different input matrix size and the summary of the calculation will be displayed in the console window.

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

```
1 bench()
2 {
3   print("Benchmar N x N 2D FFT:\n")
4   for(M := 7; M <= 12; M += 1)
5     {
6       N := 1 << M
7       print(to_string(N) + " x " + to_string(N) + "input matrix size")
8       A := afp_randu(N, N, "real")
9       a := timenow()
10      afp_fft2(A, 1, N, N)
11      b := timenow()
12      time := b - a
13      gflops := 10 * N * N * M / (time * 1000000000)
14      print(" - Time: " + to_string(time))
15      print(" - Gflops: " + to_string(gflops) + "\n")
16    }
17 }
```

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

```
18 bench()
```



MatDeck Console

```
Benchmar N x N 2D FFT:\n
128 x 128input matrix size
- Time: 0.011
- Gflops: 0.105

256 x 256input matrix size
- Time: 0.055
- Gflops: 0.096

512 x 512input matrix size
- Time: 0.087
- Gflops: 0.272

1024 x 1024input matrix size
- Time: 0.332
- Gflops: 0.316

2048 x 2048input matrix size
- Time: 1.206
- Gflops: 0.383

4096 x 4096input matrix size
- Time: 5.313
- Gflops: 0.379
```