FFTX-IRIS: A Dynamic Execution System for Heterogeneous Platforms

Sanil Rao*, Mohammad Alaul Haque Monil†, Het Mankad*, Jeffrey Vetter†, Franz Franchetti*
*Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
†Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA
*{hmankad, sanilr, franzf}@andrew.cmu.edu, †{monilm, vetter}@ornl.gov

***Abstract*—FFTX-IRIS is a dynamic system to efficiently utilize novel heterogeneous platforms. This system links two next generation frameworks, FFTX and IRIS, to navigate the complexity of different hardware architectures. FFTX provides a runtime code generation framework for high performance kernels. IRIS provides a heterogeneous runtime environment allowing computation on any available compute resource. Together, FFTX-IRIS enables seamless portability and performance without user involvement. We show the design of the FFTX-IRIS system as well as a simple example of a common Fast Fourier Transform(FFT).**

I. Introduction

Next generation computation systems are becoming increasing heterogeneous to satisfy modern application's computational needs. This heterogeneity comes at the cost of increased complexity to modern application developers. Developers need to focus on portability of applications across a wide variety of target hardware platforms as well as performance on each platform. This new focus distracts developers from the application itself, instead spending time on tedious porting and specifics of computer architecture.

To combat this issue we propose FFTX-IRIS, a dynamic system to address the complexities of next generation systems. FFTX-IRIS handles the performance and portability of modern applications in the backend while providing developers a familiar library style API. In the backend, FFTX-IRIS uses a combination of code generation, Just-In-Time compilation, and dynamic runtime scheduling to enable efficient use of all available computing platforms.

II. Background

FFTX-IRIS consists of two critical software frameworks, FFTX, and IRIS. We provide a brief overview of each framework before showing by example, the new FFTX-IRIS design.

FFTX. The FFTX project is a high performance library designed to provide common FFT transforms for new hardware architectures. Unlike traditional libraries, FFTX uses a combination of a code generator, SPIRAL [1], and Just-In-Time compilation as its library backend. This is in contrast to hand written routines implementing library function calls. This approach enables increased optimization and portability of applications across different hardware platforms.

IRIS. IRIS [2] is a programming system for extremely heterogeneous architectures. IRIS enables application developers to write portable applications across diverse heterogeneous programming platforms including CUDA, HIP, Level Zero, OpenCL, and OpenMP. It orchestrates multiple programming platforms in a system into a single execution/programming environment by providing portable tasks and shared virtual device memory.

III. FFTX-IRIS design: Multi-Dimensional FFT Example

We describe the design of FFTX-IRIS by going through a simple Multi-Dimensional FFT(MDDFT) example end to end.

```
#include <iris/iris.hpp>
#include <iris/iris_openmp.h>
#include <include/interface.hpp>
#include <include/mddftlib.hpp>
#include <stdio.h>
#include <iostream>
#include <vector>

int main(int argc, char** argv) {
  int n,m,k;
  n = 8;
  m = 8;
  k = 8;
  std::vector<int> sizes{n,m,k};
  double *Y, *X, *sym;
  X = new double[n*m*k*2];
  Y = new double[n*m*k*2];
  sym = new double[n*m*k*2];
  generateInputBuffer(X, sizes);
  std::vector<void*> args{Y,X,sym};
  MDDFTProblem mdp(args,sizes,"mddft");
  mdp.transform();
  for(int i = 0; i < n*m*k; i++) {
    std::cout << Y[i] << std::endl;
  }
  return 0;
}
```

Fig. 1. MMDFT Application using the FFTX-IRIS system

Figure 1 shows a user written MDDFT application with traditional C++ style. The user declares some sizes and creates some buffers for the input and output of the FFT. They then create and instantiate the FFTX problem object (the library API of FFTX) and call transform to perform the computation after which they print the result to verify correctness. FFTX-IRIS follows a header only design, allowing for minimal if any changes to the source application. If our example application had already been written using a different FFT API one would simply have to replace the library calls with FFTX while keeping the rest the same.

To compile the application one has to go through the traditional compilation steps using a standard C++ compiler making sure to include the directories of FFTX and IRIS. The user must also specify all target platforms they would want the application to run on by setting the `IRIS_ARCHS` environment variable. For this example we will say that we are running on a compute node that has an NVIDIA GPU as an target platform. After the `MDDFTProblem` is executed the backend FFTX system is invoked. `MDDFTProblem` is a derived class of the general `FFTXProblem` [3] which contains the functions for all the transforms supported by FFTX. In the `FFTXProblem` a transform script is generated which describes computation being performed, using runtime information like the sizes the user wrote, called `semantics`. `Semantics` is the input to the SPIRAL code generation system which generates an efficient implementation for our NVIDIA GPU target at the user specified size. This code is written to disk so that it can be seen by the IRIS runtime system. Writing to disk also acts as a caching mechanism, allowing reuse in future runs of the application.

After the generated code has been written to disk the IRIS backend is initialized. In this phase the first step takes the generated code and compiles to the appropriate hardware assembly; `PTX` in our specific example for NVIDIA platforms. Then the IRIS setup phase is invoked to create the IRIS runtime task. This task is setup in the following manner. First a simple parser is used to parse the metadata of the generated code. This provides information such as kernel names, kernel launch parameters, and internal device arrays. Next iris memory objects are created which mirror the memory objects created by the user. These memory objects are then populated with the data of the mirrored user objects using the general iris host-to-device call. Finally the task is created which captures all the above information and includes the kernel names and kernel launch parameters. These are sent to the iris scheduler to be scheduled and executed on the target platform, the NVIDIA GPU.

Once the execution of all the kernels is complete a data transfer from the GPU is initialized to move the data back to the user memory object. This allows for termination of the IRIS system and a move back to the original application. In the user application the user prints the output data as a result of performing the transform, which will contain the output of the code run on the GPU. This entire process was done transparently to the user as it was all hidden in the backend, while the frontend just used the high level FFTX library API.

IV. FFTX-IRIS Overhead

Introducing a runtime system like FFTX-IRIS comes at the cost of additional overhead. Code-generation and runtime scheduling are happening at runtime when they weren't happening before. Figure 2 shows that overhead of the example 8 by 8 by 8 MDDFT application using FFTX-IRIS. Given that this run is a small size the overhead is significant but reduced by an order of magnitude thanks to caching. We expect larger sizes to be less impacted due to increased computation of the original application and optimization of the generated code.

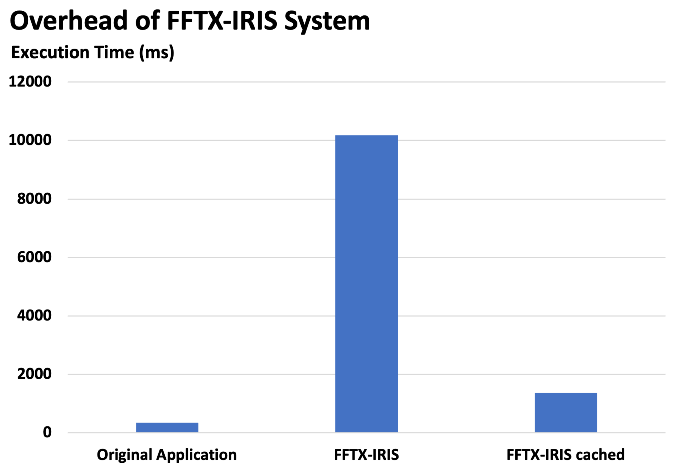

Fig. 2. Overhead of FFTX-IRIS System with and without caching for 8x8x8 MDDFT [4]

V. Conclusion

FFTX-IRIS is a novel software framework for accelerating applications on heterogeneous platforms. It takes two independent frameworks, FFTX and IRIS, and integrates them to provide increased performance on many new hardware platforms. FFTX is the user facing kernel framework, providing optimized computation kernels. IRIS is the backend heterogeneous runtime system executing FFTX kernels on any and all available target platforms. FFTX-IRIS is transparent to the user, obfuscating complicated application porting. Moving forward, FFTX-IRIS will be able to supported increased heterogeneity by running on multiple platforms at the same time.

VI. Acknowledgements

This research used resources of the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] F. Franchetti, T.-M. Low, T. Popovici, R. Veras, D. G. Spampinato, J. Johnson, M. Püschel, J. C. Hoe, and J. M. F. Moura, "SPIRAL: Extreme performance portability," *Proceedings of the IEEE, special issue on "From High Level Specification to High Performance Code"*, vol. 106, no. 11, 2018.

[2] J. Kim, S. Lee, B. Johnston, and J. S. Vetter, "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems," in *2021 IEEE High Performance Extreme Computing Conference, HPEC 2021, Waltham, MA, USA, September 20-24, 2021*, pp. 1–8, IEEE, 2021.

[3] S. Rao, A. Kutuluru, P. Brouwer, S. McMillan, and F. Franchetti, "GBTLX: A First Look," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2020.

[4] "CuFFT." Available at https://docs.nvidia.com/cuda/cufft/index.html.