# IRISX: A Dynamic Trade-off System for Harnessing Heterogeneity for Performance Portability

Sanil Rao*, Mohammad Alaul Haque Monil†, Het Mankad†,
Narasinga Rao Miniskar†, Keita Teranishi†, Jeffrey Vetter†, Franz Franchetti*
*Carnegie Mellon University, Pittsburgh, PA. {sanilr, franzf}@andrew.cmu.edu
†Oak Ridge National Laboratory, Knoxville, TN. {monilm, mankadhy, miniskarnr, teranishik, vetter}@ornl.gov

***Abstract*—Contemporary high performance computing, cloud, and embedded systems are equipped with various combinations of heterogeneous processors. This has led to the development of portable abstractions to target these heterogeneous systems. However, these abstractions fail to deliver performance portability as they cannot easily modify key parameters such as the amount of concurrency, the representation of the computation graph, the kernel implementation, and data transfers for varying problem sizes and hardware architectures. To address this wide parameter space, this work presents IRISX, a framework that dynamically finds the performance portable solution based on application characteristics and underlying hardware. By changing the representation of the computation at the kernel level, task level, and graph level, IRISX finds the appropriate set of heterogeneous processors that provides the best performance while ensuring the portability to different heterogeneous systems.**

### A. Motivation

In-node heterogeneity is ubiquitous in contemporary high performance computing (HPC), cloud, and embedded systems. Although it facilitates the increase in computational power, it also introduces two main portability challenges [41]. The first is functional portability, where the software stack is made portable through various abstractions for higher programming productivity, not ensuring the best performance. The second challenge is performance portability that depends on optimized kernels, as well as seamless scalability and utilization of heterogeneous processors. To harness heterogeneity for obtaining the best performance, an ideal combination of different factors from application and hardware needs to be considered. These include architecture-optimized kernels, the right granularity of the computation (kernels), efficient representation of the computation that exposes concurrency to be utilized, the number of devices to be used, and seamless orchestration of the heterogeneous devices with efficient scheduling that reduces unnecessary data movement.

This work presents IRISX, a dynamic trade-off system for harnessing multi-accelerator heterogeneity that strives towards providing the ideal solution mentioned above (motivated by the findings in previous explorations [1], [2]). IRISX exposes architecture-agnostic high-level interfaces to applications which provide functional portability, and at runtime, it establishes an active interaction between the SPIRAL [3] code generation engine that generates architecture-optimized kernels and the heterogeneous runtime IRIS [4] to efficiently orchestrate computation to ensure performance. IRISX goes beyond the state-of-the-art efforts by employing dynamic adaptation through reorganizing the computation at the kernel and task graph levels to provide efficient execution using various underlying heterogeneous processors.

### B. IRISX Design

For a given application, the IRISX design flow consists of three main components shown in Fig.1. ① The code generation system, SPIRAL [5], which generates architecture-specific optimized kernels for different scientific applications such as spectral methods [6], graph algorithms [7], partial differential equations (PDE) [8], and cryptography [9], using a mathematical declarative language called Operator Language (OL) [10]. ② The intelligent runtime system (IRIS) reads the optimized kernels generated by SPIRAL and generates a directed acyclic graph (DAG) of tasks that are run on various available CPU-GPU architectures. A task in IRIS can contain multiple application kernels.

In SPIRAL, a given OL expression generates a set of computation kernels that can provide varying levels of concurrency through kernel fusion. These kernels are then passed to IRIS to generate a computational task graph (DAG) with three representations. First, each input DAG can be executed serially (DAG serial) with concurrency opportunities only at the kernel level. Second, each input DAG can be merged into a single task graph (DAG Fusion), with each sub-DAG running concurrently. Finally, all IRIS tasks in a given sub-DAG can be combined into a single task (Task Fusion), with each task executing concurrently. ③ The model-seeded tuning takes into account an analytical memory model that runs through all viable kernel combinations from SPIRAL and task graph representations from IRIS and dynamically finds the one that works best for the given size and architecture.

### C. Performance Variability

To demonstrate the performance variability and the efficacy of IRISX, we used a structured grid problem (3D Euler equation) implemented using the Proto library [11], which is designed for optimized numerical approximations to various PDE-based model problems. To showcase the interplay between various optimizations in IRISX and a variety of hardware architectures, multi-GPU nodes from Frontier (eight AMD MI250X GPUs), Aurora (six Intel Max series GPUs), and Equinox (four NVIDIA V100 GPUs) were considered. Considering various input sizes and number of GPUs, we ran 84 combinations (# of GPU configurations × # of machines

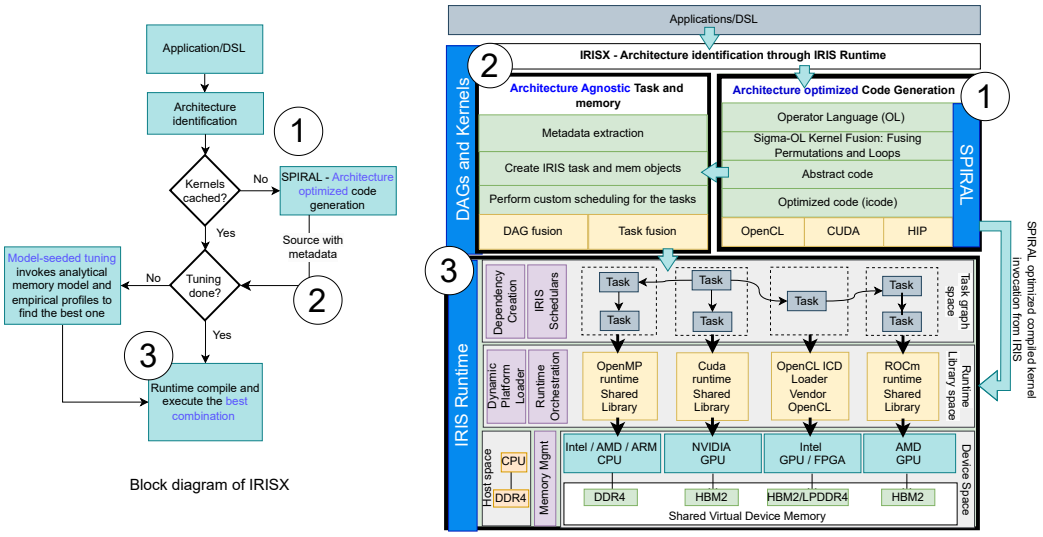
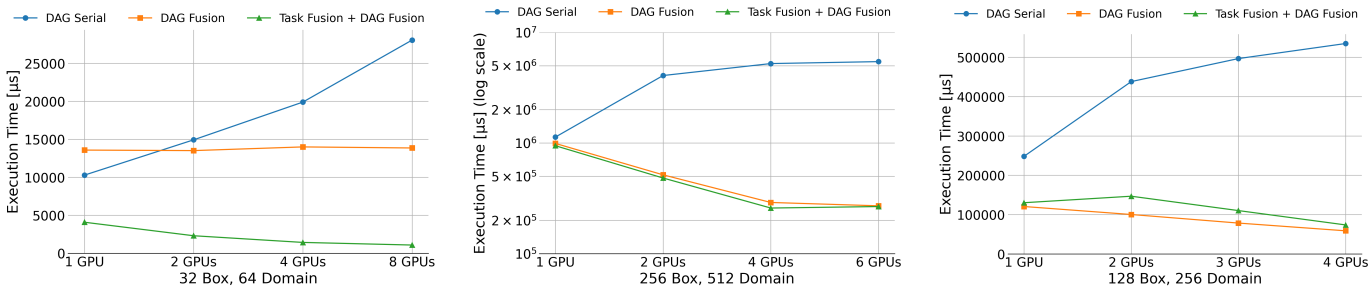



Fig. 1: IRISX Design. Left: The three main components and the flow of the IRISX system are shown. Right: The detailed software stack of the IRISX system is shown.




(a) Frontier-32 (b) Aurora-256 (c) Equinox-128

Fig. 2: Results for various box and domain sizes across Frontier, Aurora and Equinox. These charts show the dynamic relationship between the problem size and the underlying hardware configurations.

× # of Box and Domain combinations) for each optimization of IRISX. While the scalability results in a multi-GPU environment exhibited consistent trends from one to the maximum number of GPUs, a wide variety of observations were found when the problem size changed depending on the architecture, demonstrating the need for model-seeded tuning of IRISX.

Three cases of performance variability are shown in Fig. 2 where partial results from 84 combinations are considered. Here, the blue line represents the performance without any optimization of IRISX, which consistently shows the worst results in all three cases because of the serial execution of each box. In this scenario, each box (sub-DAG) utilizes multiple GPUs, leading to increased data transfers. However, the orange (DAG Fusion) and green (Task Fusion + DAG Fusion) lines in Fig. 2 show different results and trends. More fusion (Task + DAG Fusion) provided better results in Frontier (in Fig. 2a) for smaller box sizes because of the reduction of task management overhead through fusing tasks. On the other hand, Aurora showed invariance with or without task fusion for larger box sizes, indicating a proper balance between task overhead and concurrency. Interestingly, the Equinox results in Fig. 2c show an opposite trend to Frontier, with DAG fusion providing the best result, which can be correlated with good utilization of multi-stream execution. Moreover, kernel fusion also provides such variations. Although performance varies widely, IRISX is capable of finding the best-performing configurations through its model-seeded tuning.

### *D. Conclusion*

IRISX provides a portable solution for multi-accelerator systems by integrating the SPIRAL code generation engine and the IRIS Runtime. It goes beyond contemporary portable abstractions by dynamically optimizing the shape of the DAG and kernels. IRISX demonstrates automatic scalability through a structured grid application across various hardware platforms. We believe that a system like IRISX is critical for addressing the challenges of performance portability in scientific computing.

## I. Acknowledgment

This work is funded, in part, by Bluestone, an X-Stack project in the DOE Advanced Scientific Computing Office with program manager Hal Finkel.

## REFERENCES


[1] S. Rao, M. A. H. Monil, H. Mankad, J. Vetter, and F. Franchetti, "FFTX-IRIS: Towards Performance Portability and Heterogeneity for SPIRAL Generated Code," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1635–1641. [Online]. Available: https://doi.org/10.1145/3624062.3624242

[2] H. Mankad, M. A. H. Monil, S. Rao, P. Colella, B. Van Straalen, F. Franchetti, and J. Vetter, "A Performance-Portable MultiGPU Implementation of 3D Euler Equations using ProtoX and IRIS," in *SC '24 workshop, ScalAH '24: The 15th Workshop on Latest Advances in Scalable Agorithms for Large Scale Heterogeneous Systems*, 2024, accepted for publication.

[3] 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.

[4] N. R. Miniskar, M. A. H. Monil, P. Valero-Lara, F. Y. Liu, and J. S. Vetter, "Iris-dmem: Efficient memory management for heterogeneous computing," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2023, pp. 1–7.

[5] "SPIRAL," https://spiral.net/.

[6] J. Xiong, J. Johnson, R. Johnson, and D. Padua, "SPL: A language and compiler for DSP algorithms," in *Proc. Programming Language Design and Implementation (PLDI)*, 2001, pp. 298–308.

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

[8] H. Mankad, S. Rao, P. Colella, B. Van Straalen, and F. Franchetti, "Protox: A first look," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–6.

[9] N. Zhang, A. Ebel, N. Neda, P. Brinich, B. Reynwar, A. G. Schmidt, M. Franusich, J. Johnson, B. Reagen, and F. Franchetti, "Generating high-performance number theoretic transform implementations for vector architectures," in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–7.

[10] F. Franchetti, F. de Mesmay, D. McFarlin, and M. Püschel, "Operator language: A program generation framework for fast kernels," in *Domain-Specific Languages*, W. M. Taha, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 385–409.

[11] "Proto Github," https://github.com/applied-numerical-algorithms-group-lbnl/proto.