POSTER

Seamless Acceleration of Fortran Intrinsics via AMD AI Engines

NICK BROWN, The University of Edinburgh, Edinburgh, Scotland, U.K.

GABRIEL RODRIGUEZ-CANAL, The University of Edinburgh, Edinburgh, Scotland, U.K.

Published: 27 February 2025

Citation in BibTeX format

FPGA '25: The 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays
February 27 - March 1, 2025
CA, Monterey, USA

Conference Sponsors:
SIGDA

Seamless Acceleration of Fortran Intrinsics via AMD AI Engines

Nick Brown
n.brown@epcc.ed.ac.uk
EPCC at the University of Edinburgh
Edinburgh, UK

Gabriel Rodriguez-Canal
EPCC at the University of Edinburgh
Edinburgh, UK

Abstract

A major challenge that the HPC community faces is how to continue delivering the performance demanded by scientific programmers, whilst meeting an increased emphasis on sustainable operations. Specialised architectures, such as FPGAs and AMD's AI Engines (AIEs), have demonstrated significant energy efficiency advantages, however a major challenge is that to most effectively program these architectures requires significant expertise and investment of time which is a major blocker. Fortran in the lingua franca of scientific computing, and in this work we seamlessly accelerate Fortran intrinsics via the AIEs in AMD's Ryzen AI CPU.

CCS Concepts

• **Software and its engineering → Compilers**.

Keywords

AMD AI engines, Ryzen AI, Fortran, MLIR, xDSL

ACM Reference Format:
Nick Brown and Gabriel Rodriguez-Canal. 2025. Seamless Acceleration of Fortran Intrinsics via AMD AI Engines. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25), February 27-March 1, 2025, Monterey, CA, USA.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3706628.3708854

1 Our approach

LLVM provides reusable compiler and tool chain technologies that enable the development of compilers across different languages and hardware. LLVM-IR connects LLVM frontends and backends, however this is low level and MLIR addresses this by providing a range of IR abstractions. Contributing many IR dialects and transformations between them, a mix of higher level intermediate representations and transformations can be leveraged. MLIR is also a framework, enabling the development of new IR dialects and transformations.

Flang [3] is LLVM's Fortran frontend and built on-top of MLIR. Leveraging the open source Flang compiler and MLIR ecosystem, we focus on Fortran intrinsics, which are built in procedures defined by the Fortran standard to provide utility functionality. Given Fortran's lineage in scientific computing, a range of intrinsics are defined that undertake calculations and examples include the *matmul* procedure which undertakes matrix multiplication. Whilst the Flang compiler maps these directly to function calls in the Flang runtime library, the work undertaken in [2] instead maps these to the MLIR linear algebra, *linalg*, dialect using the xDSL [1] framework. Our approach illustrated in Figure 1 lowers operations in the MLIR *linalg* dialect to target the NPU in the Ryzen AI 7000 CPU. We primarily do this by providing a pre-generated library of generic IR for each intrinsic operation that will run on the AIEs of the NPU, with generics in the IR then concreted by the compiler.

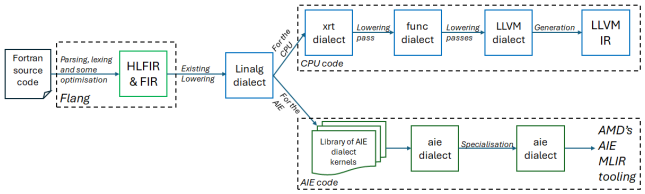

Figure 1: Our overarching compiler approach that offloads selected linear algebra operations to the AIE array

2 Results and Evaluation

Table 1 reports the performance of the matrix multiplication Fortran intrinsic for input input array sizes of 256x256 and 256x512, calculating a result array size of 256x512, across the CPU and NPU for different data types. It can be seen that the NPU outperforms the CPU for all configurations. The int32 and float32 calculations on the NPU are marked with an asterisk because this data type is used as the output, with the algorithm using the reduced precision counterpart (int16 and bfloat16 respectively) for inputs. It can also be seen that there is a performance difference between the first and subsequent runs on the NPU due to configuration overhead.

Data type	CPU runtime (us)	NPU first runtime (us)	NPU subsequent runtime (us)
int16	5473	2572	1353
int32	14032	2635*	1503*
bfloat16	815194	2626	1357
float32	17566	3901*	1471*

Table 1: Runtime (in microseconds) of *matmul* Fortran intrinsic with a problem size of 256x256x512 elements

References

[1] George Bisbas et al. 2024. A shared compilation stack for distributed-memory parallelism in stencil DSLs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3.* 38–56.
[2] Nick Brown. 2024. Fully integrating the Flang Fortran compiler with standard MLIR. *arXiv preprint arXiv:2409.18824* (2024).
[3] Flang 2024. *Flang Documentation.* Retrieved Aug 16, 2024 from https://flang.llvm.org/docs/

FPGA '25, February 27-March 1, 2025, Monterey, CA, USA

ACM ISBN 979-8-4007-1396-5/25/02
https://doi.org/10.1145/3706628.3708854