Developing a BLAS library for the AMD AI Engine

Tristan Laan
Vrije Universiteit Amsterdam
t.laan2@student.vu.nl

Tiziano De Matteis
Vrije Universiteit Amsterdam
t.de.matteis@vu.nl

Abstract—Spatial (dataflow) computer architectures can mitigate the control and performance overhead of classical von Neumann architectures such as traditional CPUs. Driven by the popularity of Machine Learning (ML) workload, spatial devices are being marketed as ML inference accelerators. Despite providing a rich software ecosystem for ML practitioners, their adoption in other scientific domains is hindered by the steep learning curve and lack of reusable software, which makes them inaccessible to non-experts. We present our ongoing project AIEBLAS, an open-source, expandable implementation of Basic Linear Algebra Routines (BLAS) for the AMD AI Engine. Numerical routines are designed to be easily reusable, customized, and composed in dataflow programs, leveraging the characteristics of the targeted device without requiring the user to deeply understand the underlying hardware and programming model.

I. Introduction

The end of Dennard scaling and Moore's law, along with the growing computational needs of application domains such as machine learning, are compelling industry and researchers to rethink classical computer organization. One promising alternative is constituted by Spatial Architectures, which, moving away from the von Neumann model, aim to make more efficient use of the available transistors and chip space.

Spatial architectures are today marketed mainly as Machine Learning (ML) workload accelerators (e.g., AMD/Xilinx's ACAP platform [1], the Sambanova Reconfigurable Dataflow Architecture [2], and Cerebras Wafer Scale Engine [3]). These devices have tens to thousands of Processing Elements organized in a 2D grid, communicating using a fast, reconfigurable Network-On-Chip (NoC). Common to all is their amenability to being programmed with a dataflow programming model to favor on-chip data movement and reduce control overheads. In line with their primary audience, manufacturers of spatial architectures offer integration with popular high-level ML frameworks (e.g., PyTorch and TensorFlow [4]–[6]) and release predefined models [7], [8], allowing the users to target these devices for inference tasks conveniently.

Despite the promise of massive parallelism and high performance, the scientific community has yet to fully explore the use of such devices in areas other than ML, such as computational science or graph processing. In such cases, programmers have to rely on lower-level APIs (e.g., AMD ADF [9], Cerebras CSL [10]) to fully utilize the devices' capabilities. However, the steep learning curve of these APIs and the lack of reusable libraries make it hard for non-experts to explore and leverage these new devices.
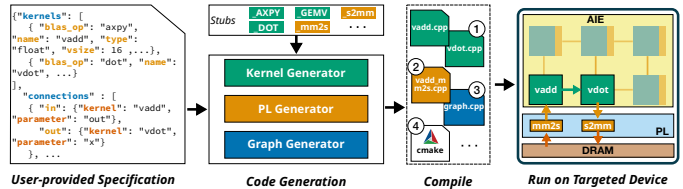

Fig. 1. AIEBLAS development workflow.

In this talk, we present AIEBLAS, our ongoing effort to design and develop an open-source[1] implementation of Basic Linear Algebra Routines (BLAS) for the AMD AI Engine (AIE), a spatial architecture currently being offered in commodity CPUs [11] and data center accelerators [1]. Our goals for the AIEBLAS library are *1)* to provide ready-to-use numerical routines that can be customized and integrated with other code without requiring the user to write lengthy and complicated lower-level code; *2)* to be easily expandable with new functionalities and optimizations; *3)* to naturally favor on-chip communications using a dataflow approach.

Although our focus in this work is on the AIE architecture, we believe similar design principles and reasoning can also be applied to other spatial architectures.

II. Background

AIEBLAS targets Versal Adaptive Compute Acceleration Platform (ACAP) devices. Figure 2 shows the high-level architecture of the VCK5000 development board [12]. The AIE array is organized in an 8×50 grid of 400 AIEs. Each AIE contains a Very Long Instruction Word vector processor and 32KB of local memory. It can share data with the adjacent AIEs by reading/writing directly from/to their local memory. Non-local communications are implemented via AXI4 Streams. The Programmable Logic (PL) component comprises logic blocks, memory, and DSPs that can be used to implement custom logic in hardware. The AIE array and PL communicate via multiple AXI interfaces (312 PL $\rightarrow$ AIEs, and 234 AIEs $\rightarrow$ PL), each operating at 4 GB/s.

The AIEs can be programmed using the Adaptive Dataflow (ADF) API [9]. The application is represented by a dataflow graph of kernels scheduled one the AIEs. Kernels exchange data by blocks (*windows*) or element by element (*streams*), using the underlying NoC and neighbor interfaces. The PL can be programmed using High-Level Synthesis (HLS) or RTL.
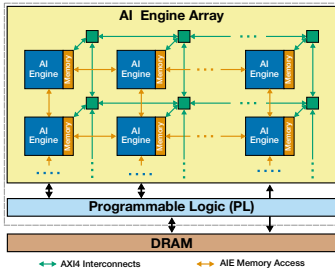
---

[1]The library is available at: https://github.com/atlarge-research/AIE-BLAS

Fig. 2. Overview of the AMD Versal ACAP Architecture.

Fig. 3. AIEBLAS evaluation results.

III. DESIGN AND IMPLEMENTATION

Figure 1 shows the general development flow with AIEBLAS. Starting from a JSON high-level specification, AIEBLAS generates a design consisting of ① the AIE kernels that implement the required BLAS routines, ② the PL kernels to send and receive data from the device DRAM, ③ a dataflow graph to execute the program and, if applicable, connect the AIE kernels as specified, and ④ a CMake project to build the design. Different template-based code generators are in charge of producing the code for the various design components. All can be conveniently extended to implement new functionalities (e.g., a new routine) or improve existing ones (e.g., an optimized implementation of a given routine).

AIEBLAS routines accept and produce scalar data using streams. For vectors and matrices, we let routines accept and produce *windows*. This approach has several benefits. First, windows are stored on the local memory, and can be accessed using a wider datapath compared to AXI4 streaming interconnects, allowing us to fully leverage the AIE vector processor. Second, they allow decoupling communications between two communicating AIEs, which is useful for on-chip communications. Kernel code is vectorized to fully utilize the computing capabilities of the AIEs. The user can set the vector width, which defaults to the maximum supported (512 bits).

Numerical computations can be composed of two or more routines that share data. For instance, the example of Figure 1 computes an axpydot ($\beta = z^T u$ with $z = w - \alpha v$, where w, v, and u are vectors, and α and β are scalars, [13]). This can be implemented by first performing a vector addition (axpy), and then using its output as the input of the subsequent dot product. Rather than exchanging data via off-chip memory, we want to favor on-chip communications, composing the routines in a dataflow graph. In this way, we reduce the amount of expensive off-chip accesses and allow for the pipeline executions of multiple routines. AIEBLAS gives users the option to specify connections between BLAS routines, and the code generator will produce the corresponding dataflow graph definition. If a routine input/output is not connected to another routine, AIEBLAS will create a PL kernel to load/store the data from off-chip memory.

IV. INITIAL RESULTS

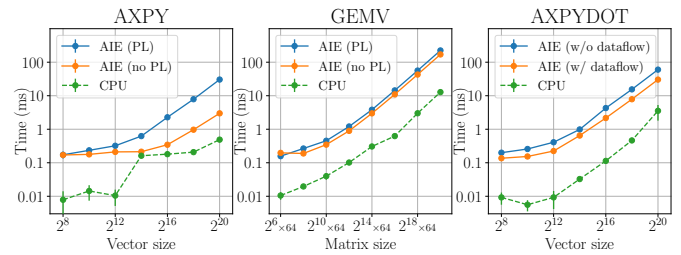We evaluated the current implementation of AIEBLAS on an AMD VCK5000. The code has been compiled with Xilinx Vitis v2022.2 and GCC 11.4.0. The host has a 10 cores Intel Xeon Silver 4210R operating at 2.4GHz, and 256 GB of DDR4 memory. For the CPU benchmarks we use OpenBLAS 0.3.27.

We considered the vector addition (axpy) and matrix-vector multiplication routines (gemv) as well as the composed axpydot. Figure 3 reports the averaged execution times.

For the single routines, we tested an implementation that uses PL kernels to read/write data from DRAM and an implementation where the data is generated directly on-chip. The latter results in reduced running time, stressing how off-chip access can impact performance and emphasizing the importance of carefully managing PL data movers to leverage the multiple AIE-PL interfaces. For the axpydot routine, we evaluated both dataflow and no-dataflow implementations. As expected, the dataflow approach doubled the performance, indicating that pipelined execution offers significantly better performance. Finally, CPU performance is generally better (up 10x) than AIE implementations for all scenarios considered. This is due to OpenBLAS's optimized multicore implementation, and suggests that further optimizations are needed for AIE to achieve competitive performance.

V. CONCLUSION AND FUTURE WORK

In this talk, we presented the design of AIEBLAS, a BLAS library for the AMD AIE spatial architecture. The library uses automatic code generation to produce architecture-specific code based on the user's higher-level specification, significantly increasing user productivity. Initial performance results demonstrate how dataflow composition is necessary to favor on-chip communications and enable the pipelined execution of multiple routines. The results also highlight the need for more spatial parallelism. Indeed, it is crucial to exploit more parallelism in the PL, to leverage the multiple PL-AIE interfaces and saturate available DDR bandwidth, and via multi-AIE routine implementations, to improve compute performance even for communication-bound routines.

We intend to continue developing and extending AIEBLAS features. First, we want to systematically support multi-AIEs routine implementations, to increase compute performance, and tiling to further reduce off-chip accesses. Second, we want to increase BLAS coverage by implementing more routines, also by considering state of the art implementations [14]–[16]. Finally, by publicly releasing AIEBLAS, we want to engage the community in developing this and other software libraries for current and future spatial architectures.

REFERENCES

[1] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versaltm architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[2] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, and A. Sujeeth, "Accelerating scientific applications with sambanova reconfigurable dataflow architecture," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 114–119, 2021.

[3] S. Lie, "Multi-million core, multi-wafer ai cluster," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–41.

[4] "AMD Vitis AI," https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[5] Sambanova, "Accelerated Computing with a Reconfigurable Dataflow Architecture," https://sambanova.ai/hubfs/23945802/SambaNova_Accelerated-Computing-with-a-Reconfigurable-Dataflow-Architecture_Whitepaper_English-1.pdf, 2022.

[6] "Supporting PyTorch on the Cerebras Wafer Scale Engine," https://www.cerebras.net/blog/supporting-pytorch-on-the-cerebras-wafer-scale-engine/.

[7] "Vitis AI Model Zoo," https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo.

[8] "Cerebras Model Zoo," https://github.com/Cerebras/modelzoo/.

[9] *AI Engine Kernel and Graph Programming Guide (UG1079)*, Advanced Micro Devices, Inc. [Online]. Available: https://docs.amd.com/r/2022.2-English/ug1079-ai-engine-kernel-coding

[10] J. Selig, "The Cerebras Software Development Kit: A Technical Overview ," Cerebras, Tech. Rep., 2022.

[11] A. Rico, S. Pareek, J. Cabezas, D. Clarke, B. Ozgul, F. Barat, Y. Fu, S. Münz, D. Stuart, P. Schlangen, P. Duarte, S. Date, I. Paul, J. Weng, S. Santan, V. Kathail, A. Sirasao, and J. Noguera, "Amd xdna™ npu in ryzen™ ai processors," *IEEE Micro*, pp. 1–10, 2024.

[12] *VCK5000 Versal Development Card*, Advanced Micro Devices, Inc. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vck5000.html

[13] S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and C. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Trans. Math. Softw.*, vol. 28, no. 2, pp. 135–151, Jun. 2002.

[14] E. Taka, D. Gourounas, A. Gerstlauer, D. Marculescu, and A. Arora, "Efficient approaches for gemm acceleration on leading ai-optimized fpgas," 2024. [Online]. Available: https://arxiv.org/abs/2404.11066

[15] Z. Wu, M. Gokhale, S. Lloyd, and H. Patel, "Sccl: An open-source systemc to rtl translator," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 23–33.

[16] J. Zhuang, Z. Yang, and P. Zhou, "High performance, low power matrix multiply design on acap: from architecture, design challenges and dse perspectives," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.