AFRL-RY-WP-TR-2023-0261

REAL-TIME MACHINE LEARNING (RTML) COMPILING HARDWARE NEURAL-NET ACCELERATORS (CHANNA)

Krste Asanovic, Borivoje Nikolic, Sophia Shao, and Jonathan Ragan-Kelley
University of California-Berkeley

June 2024
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2023-0261 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//Signature//

_____

CHRISTOPHER A. BOZADA
Program Manager
Aerospace Components and Subsystems Division

//Signature//

_____

JOHN S. CETNAR (Acting)
Deputy Chief, Aerospace Components &
Subsystems Technology Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
June 2024	Final	3 January 2020	3 January 2023

4. TITLE AND SUBTITLE

REAL-TIME MACHINE LEARNING (RTML)
COMPILING HARDWARE NEURAL-NET ACCELERATORS (CHANNA)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8650-20-2-7006	N/A	61101E

5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER
N/A	N/A	Y219

6. AUTHOR(S)

Krste Asanovic, Borivoje Nikokic, Sophia Shao, and Jonathan Ragan-Kelley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of California-Berkeley
University Avenue and, Oxford St,
Berkeley, CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Forces

Defense Advanced Research Projects Agency
DARPA/MTO
675 North Randolph Street
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYD

11. SPONSOR/MONITOR'SREPORT NUMBER(S)

AFRL-RY-WP-TR-2023-0261

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-20-2-7006. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), or the U.S. Government. Report contains color.

14. ABSTRACT

The CHANNA project developed an open-source machine-learning accelerator generator (named *Gemmini*) that supports integration into a full system-on-a-chip design. Additionally, the team completed evaluations of generated accelerators running various machine learning (ML) workloads; demonstrated that the generator can produce competitive accelerators for a wide range of ML tasks; and developed a novel accelerator virtualization mechanism, AuRORA, to enable virtualized and disaggregated accelerator integration for many-accelerator-many-application systems.

15. SUBJECT TERMS

machine learning, system-on-a-chip design, custom hardware accelerators, deep neural networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	22

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Christopher Bozada	N/A

Table of Contents

Section **Page**

List of Figures

Section **Page**

1 CHANNA (COMPILING HARDWARE NEURAL-NET ACCELERATORS) OVERVIEW

The goal of the CHANNA project was to develop an open-source machine-learning accelerator generator (named *Gemmini*) that supports integration into a full SoC design, and which includes full software support. The following list summarizes the key accomplishments during the RTML program:

- We have successfully released several versions of the Gemmini framework to the community, and the framework is now being widely used by other groups. The Gemmini paper was published at DAC'2021 and won the "Best Paper Award" of that year.

- We completed evaluations of generated accelerators running various ML workloads, including silicon implementations of several variants (two taped out during the program, and one will be taped out in November 2023).

- We have demonstrated that the generator can produce competitive accelerators for a wide range of ML tasks, including recent transformer-based ML tasks that became popular after Gemmini was initially developed.

- We have developed a software flow, Exo, to automate library development for Gemmini. Exo was published at PLDI'2022. It has been open-sourced and used by industry collaborators.

- We have used Gemmini to evaluate many-accelerator systems and have developed a novel mechanism, called MoCA, for dynamic memory-resource management, published at HPCA'2023. The hardware and software implementations of MoCA are publicly available.

- We developed a novel accelerator virtualization mechanism, called AuRORA, to enable virtualized and disaggregated accelerator integration for many-accelerator-many-application systems. AuRORA is published at MICRO'2023 and is publicly available.

2 GEMMINI OVERVIEW

Gemmini, is an open-source, full-stack DNN accelerator generator for DNN workloads, enabling end-to-end, full-stack implementation and evaluation of custom hardware accelerator systems for rapidly evolving DNN workloads. Gemmini's hardware template and parameterization allows users to tune the hardware design options across a broad spectrum spanning performance, efficiency, and extensibility. Unlike existing DNN accelerator generators that focus on standalone accelerators, Gemmini also provides a complete solution spanning both the hardware and software stack, and a complete SoC integration that is compatible with the RISC-V ecosystem. In addition, Gemmini implements a multi-level software stack with an easy-to-use programming interface to support different programming requirements, as well as tight integration with Linux-capable SoCs which enable the execution of any arbitrary software. The open-source repository is made available at: **`github.com/ucb-bar/gemmini`**. The Gemmini paper was published at the Design Automation Conference (DAC) in 2021 and won the "Best Paper Award" for that year.

2.1 Gemmini Hardware Generation

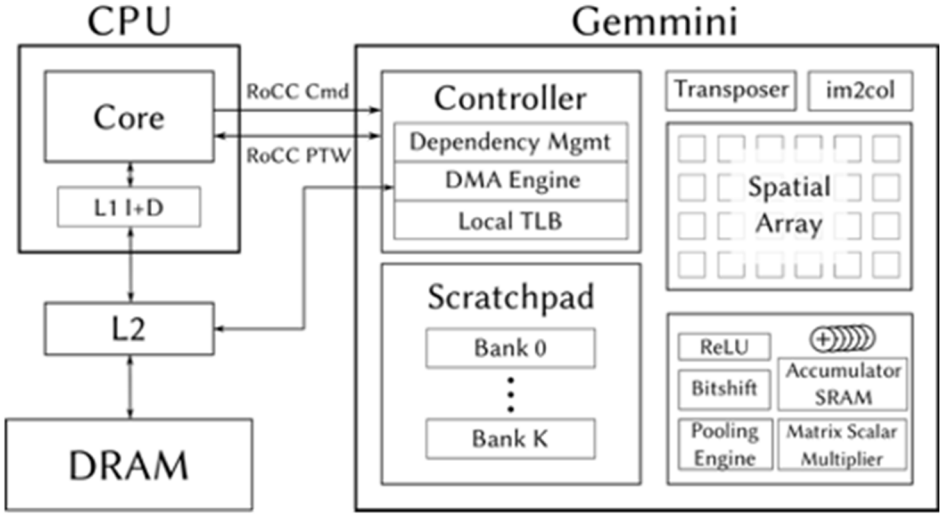

Figure 1: Gemmini Hardware Architectural Template Overview

Figure 1 illustrates Gemmini's architectural template. The central unit in Gemmini's architectural template is a spatial architecture with spatially distributed processing elements (PEs), each of which performs dot products and accumulations. The spatial array reads data from a local, explicitly managed scratchpad of banked SRAMs, while it writes results to a local accumulator storage with a higher bitwidth than the inputs.

Gemmini also supports other commonly-used DNN kernels, e.g., pooling, non-linear activations (ReLU or ReLU6), and matrix-scalar multiplications, through a set of configurable, peripheral circuitry. Gemmini-generated accelerators can also be integrated with a RISC-V host CPU to program and configure accelerators.

We design Gemmini's spatial array with a two-level hierarchy to provide a flexible template for different microarchitecture structures, as demonstrated in Figure 2. The spatial array is first composed of tiles, where tiles are connected via explicit pipeline registers. Each of the individual tiles can be further broken down into an array of PEs, where PEs in the same tile are connected combinationally without pipeline registers. Each PE performs a single multiply-accumulate (MAC) operation every cycle, using either a weight- or output-stationary dataflow. The tiles are composed of rectangular arrays of PEs, where PEs in the same tile are connected combinationally with no pipeline registers inbetween them. The spatial array, likewise, is composed of a rectangular array of tiles, but each tile does have pipeline registers between it and its neighbors. Every PE and every tile shares inputs and outputs only with its adjacent neighbors.

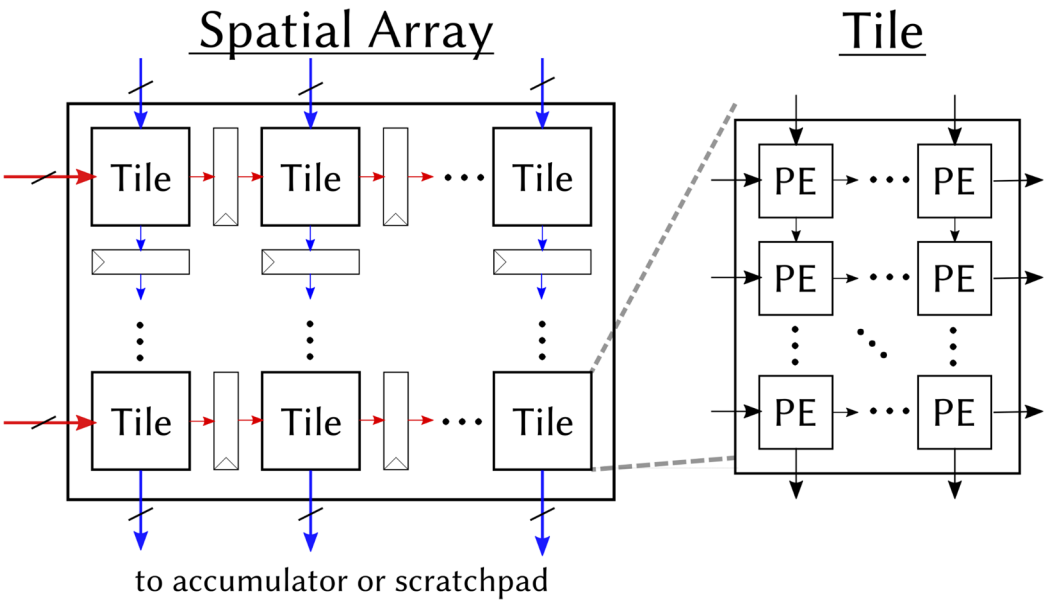

Figure 2: Microarchitecture of Gemmini's Two-level Spatial Array

2.2 Gemmini Software Stack

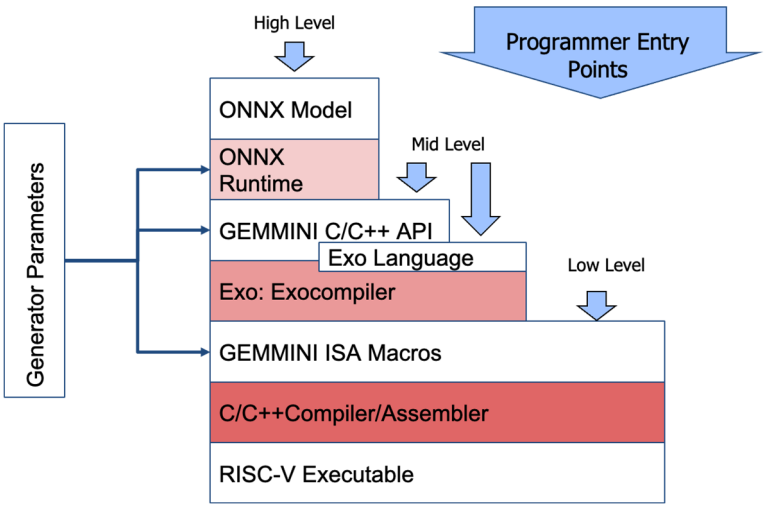

Figure 3: Overview of the Gemmini Software Stack with Multi-level Interfaces

The Gemmini generator produces not just a hardware stack, but also a tuned software stack, boosting developers' productivity as they explore different hardware instantiations. Specifically, Gemmini provides a multi-level software flow to support different programming scenarios. At the high level, Gemmini contains a push-button software flow which reads DNN descriptions in the ONNX file format and generates software binaries that will run them, mapping as many kernels as possible onto the Gemmini-generated accelerator. Alternatively, at the low level, the generated accelerator can also be programmed through C/C++ APIs, with tuned functions for common DNN kernels. These functions must be tuned differently for different hardware instantiations in order to achieve high performance, based on scratchpad sizes and other parameters. Therefore, every time a new accelerator is produced, Gemmini also generates an accompanying header file containing various parameters, e.g. the dimensions of the spatial array, the dataflows supported, and the compute blocks that are included (such as pooling, im2col, or transposition blocks).

2.3 Gemmini Impact

Gemmini-generated accelerators have been successfully fabricated in both TSMC 16nm FinFET and Intel 22nm FinFET Low Power (22FFL) process technologies, demonstrating that they can be physically realized with high quality of results. In addition, our evaluation shows that Gemmini-generated accelerators deliver comparable performance to a state-of-the-art, commercial DNN accelerator with a similar set of hardware configurations and achieve up to 2,670x speedup with respect to a baseline CPU. Gemmini's fully-integrated, full-stack flow enables users to co-design the accelerator, application, and system all at once, opening up new research opportunities for future DL SoC integration. Specifically, in our Gemmini-enabled case studies, we demonstrate how designers can use Gemmini to optimize virtual-address translation mechanisms for DNN accelerator workloads, and to partition memory resources to balance the different compute requirements of diverse layer types within a DNN.

In brief, on the DNN acceleration front we have accomplished the following:

1) We built Gemmini, an open-source, full-stack DNN accelerator design infrastructure to enable systematic evaluation of deep-learning architectures. Specifically, Gemmini provides a flexible hardware template, a multi-layered software stack, and an integrated SoC environment.

2) We performed rigorous evaluation of Gemmini-generated accelerators by using FPGA-based performance measurement and commercial ASIC synthesis flows for performance and efficiency analysis. Our evaluation demonstrates that Gemmini-generated accelerators deliver comparable performance compared to state-of-the-art, commercial DNN accelerators.

3) We demonstrate that the Gemmini infrastructure enables system-accelerator co-design of SoCs running DNN workloads, including the design of efficient virtual-address translation schemes for DNN accelerators and the provisioning of memory resources in a shared cache hierarchy.

2.4 Exo Compiler

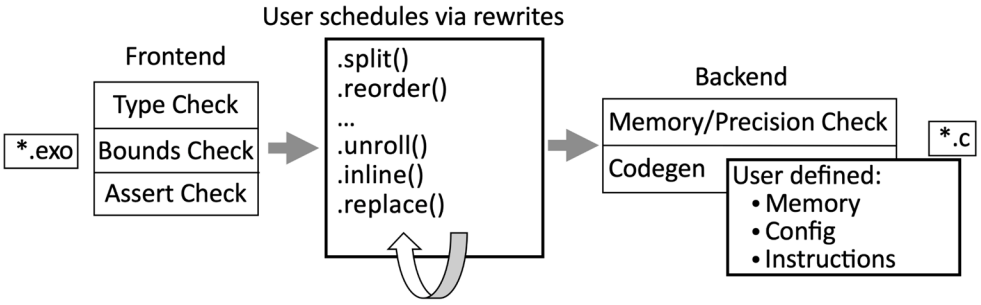

Figure 4: Exo Compilation Flow for Gemmini

Exo is a new programming language designed to accelerate the development of optimized high-performance kernel libraries for specialized hardware like Gemmini. Exo is designed around the novel principle of *exocompilation*: externalizing target-specific code generation support and optimization policies to user-level code. It allows custom hardware instructions, specialized memories, and accelerator configuration state to be defined and targeted entirely from user code, rather than requiring expert compiler engineers to modify the compiler core and write all-new backends. It builds on the idea of user scheduling, popularized by languages like Halide, to externalize hardware mapping and optimization decisions: programmers start with a simple, concise specification of *what* they want to compute, and then a schedule controls *how* that computation should be performed on the underlying hardware.

In contrast to optimization by manually rewriting low-level code, scheduling transformations are concise and safe. They elide many details like array and loop re-indexing (which can be automatically inferred), while guaranteeing both functional equivalence and memory safety. Different schedules best optimize the same library function for different hardware, or even for different parameter values, and specialized versions for each case can be generated from a single source algorithm. Arbitrary program fragments can be replaced during scheduling with equivalent user-defined accelerator instructions, or specialized subroutines, using a *unification* procedure that automates the transformation of essential arguments and array indexing. But schedules are guaranteed not to break the program or change what is computed, eliminating a

large class of bugs performance engineers commonly struggle with when hand-optimizing code in traditional performance-oriented languages like C/C++.

Exo goes beyond prior user-schedulable languages like Halide and TVM by defining schedules as rewrites within the language, rather than applied as a one-time lowering from high- to low-level code. Rewrites are naturally composable and extensible, allowing the rapid definition of new scheduling patterns as new optimizations or hardware features are developed. We developed a set of effect analyses which guarantee program equivalence and memory safety through these rewrites.

Using Exo's inherent extensibility, we have implemented support for targeting the essential computational primitives and specialized memories of various generations and instantiations of the Gemmini architecture. Leveraging Exo's unique exocompilation paradigm, we've been able to natively express Gemmini-specific hardware instructions and configurations directly from user-level code. This accelerates the development process as the hardware architecture evolves, and alleviates the bottleneck of requiring specialized compiler engineering expertise for each change to the hardware. We've particularly focused on two key machine learning kernels, namely General Matrix Multiply (GEMM) and Convolution layers, which are fundamental to a broad range of machine learning applications. We have used Exo to implement and optimize these kernels on Gemmini hardware, achieving higher performance with significantly less effort and shorter code than the alternative of hand-written C and assembly.

3 FULL-STACK OPTIMIZATION OF TRANSFORMER

A goal of the CHANNA project was to develop an AI/ML accelerator generator that could be quickly retargeted based on advances in new AI/ML algorithms. Recent advances in state-of-the-art neural network architecture design have been moving toward Transformer models. These models achieve superior accuracy across a wide range of applications in computer vision, natural language processing, and speech recognition. This trend has been consistent over the past several years since Transformer models were originally introduced. However, the amount of compute and bandwidth required for inference of recent Transformer models is growing at a significant rate, and this has made their deployment in latency-sensitive applications challenging. As such, there has been an increased focus on making Transformer models more efficient, with methods that range from changing the architecture design, all the way to developing dedicated domain-specific accelerators.

As a case study, we leverage Gemmini's flexibility to explore hardware-software co-design for transformer inference. Gemmini was originally designed before Transformers became popular and focused on the then-popular CNN algorithms, so there was initially no support for non-linear normalization operations such as LayerNorm or Softmax. Neither was there support for GELU, which is a relatively expensive non-linear activation function often implemented with costly lookup tables. When we first run BERT on the default Gemmini, it achieves less than 1% utilization of its functional units when performing BERT inferences. Although individual matmuls achieve 74% utilization, operations that the accelerator doesn't support natively, such as LayerNorm, significantly reduce performance as they must be performed by the CPU instead. In fact, we found that 96% of execution is spent on non-matmul operations. Note that over 99% of FLOPs in our Transformer inference are MACs for matmuls, so the time consumed by each operation in the baseline accelerator's run is far from the theoretical ideal, unless further optimizations are made.

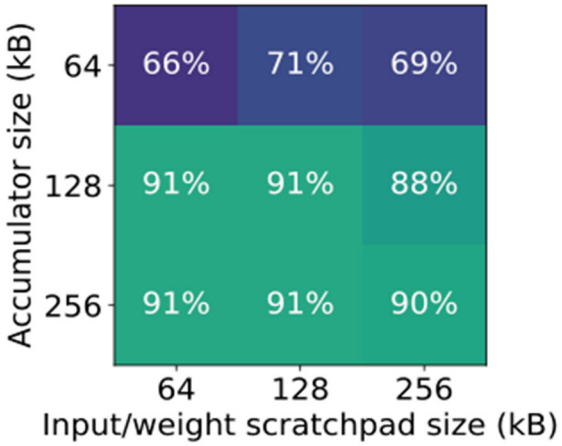

Figure 5: Accumulator and Scratchpad Size Sweep for Running BERT on Gemmini

Transformer matmuls (in particular, the act-to-act matmuls) often have very different shapes and arithmetic intensities than the convolutional layers in CNNs. As illustrated in Figure 5, leveraging Gemmini's ability to quickly generate different hardware parameters, simply adjusting the sizes of the input/weight scratchpad and 32-bit partial accumulator significantly

improves the performance of BERT's matmul operations. Larger accumulators enable higher output-reuse, which is more suited for several of the matmuls in Transformers. Given this observation, we reduce the size of our baseline accelerator's shared input/weight scratchpad to 64 kB from 256kB, and we increase the size of the partial-sum accumulator to 256 kB from 64kB. This involves no increase in the total SRAM capacity and virtually no change to the total area of our accelerator. However, these changes yield a substantial 36% reduction in total matmul latency.

To alleviate the overhead of runtime quantization and dequantization, we switched our baseline Transformer workload from a naive BERT implementation, where only matmuls are quantized, to an integer-only BERT variant called I-BERT. The main idea of I-BERT is to replace floating-point nonlinear operations such as GELU and Softmax with integer polynomial approximations such that they are both faster and cheaper to implement in specialized hardware accelerators.

To incorporate I-BERT, we added new integer implementations of I-BERT's GELU, LayerNorm, and Softmax variants to Gemmini. The 32-bit matmul results resident in the accumulator are fed into a newly added normalization unit which computes sums, sums-of-squares, maxes, and other reductions which are used by LayerNorm and Softmax. Multiple passes of accumulator-reads are required to compute all the reductions in these operations. For example, a sum is computed first before a variance is computed using that sum. Afterwards, the matmul results in the accumulators are read one final time to be fed into a set of 16 activation units which compute I-BERT's GELU, LayerNorm, or Softmax variants in parallel.

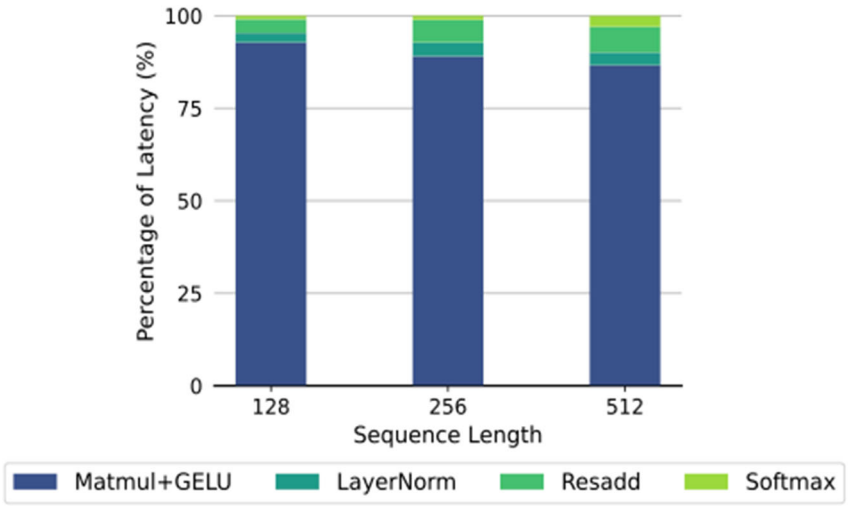

Figure 6: Time Breakdown After Extending Gemmini for BERT

With these new features, overall end-to-end BERT inference performance improved by 39.6X over the baseline accelerator's initial performance. As Figure 6 illustrates, the computational bottleneck once again became the matmuls rather than normalization or activation functions; and this trend persists across different sequence lengths. Quantization and dequantization no longer become necessary, since the non-linear floating-point operations are replaced with I-BERT's integer polynomial approximations. Also note that GELU operations can now be trivially fused with the preceding matmuls, so that they become one pipelined operation. When synthesized

with the ASAP7 PDK, the new hardware units increased the total area consumption of the accelerator by only 14%, and the GELU, LayerNorm, and Softmax operations increased the power consumption of a BERT inference by only 9.3%.

4 VLSI EVALUATIONS

We have successfully fabricated two Gemmini-generated accelerators integrated in complete SoCs, in three different process technologies: TSMC 16nm finFET, GlobalFoundries 12nm finFET and Intel 22nm finFET low power (22FFL) process technologies, with a third tapeout planned for November 2023 in Intel 16.

In the phase 3 of the DARPA CRAFT program a large SoC, EagleX, was designed and taped out in TSMC 16nm FFC process. This SoC included ten clusters with two RISC-V Rocket cores in each tile, a system-management tile, a three-level cache system and high-speed serial links. It also included a small, 16x16 int8 Gemmini tile that was accumulating to 32b outputs. The tile was controlled by a separate Rocket core and had 256 KiB of scratchpad for inputs and weights, and 64 KiB scratchpad for the output accumulator.

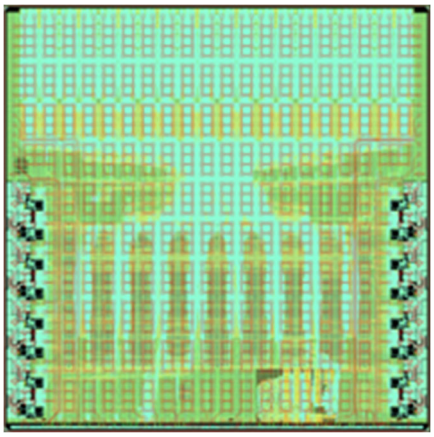

Figure 7: EagleX, 16nm FFC TSMC

EagleX is fully functional, and each of the 20 cores is able to boot Linux. Schmoo plot shows operation at 950MHz at the nominal supply voltage and up to 1.5GHz at 1.15V. The Gemmini-generated tile is also functional, operating at 500MHz, and provides more than 1000x acceleration over the Rocket cores for matrix-multiplication workloads.

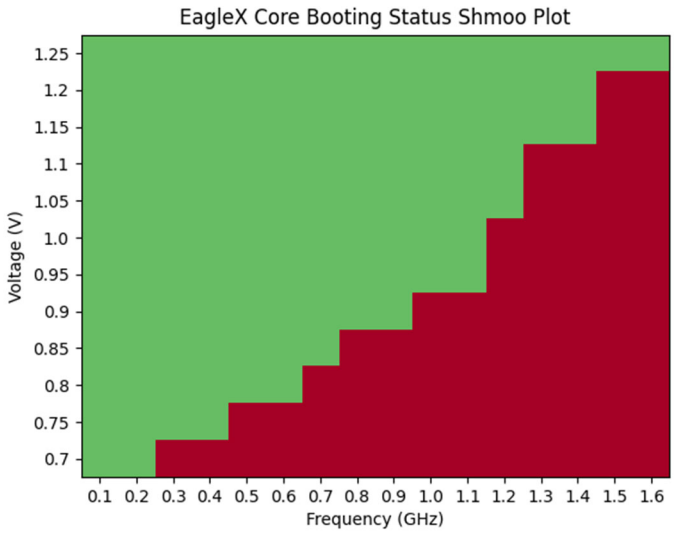

Figure 8: EagleX Schmoo Plot

```
performing 2 C = AB + D operations with CPU and Gemmini...
cpu time taken: 5537 ms,         gemmini time taken: 4 ms
performing 2 C = AB + D operations with CPU and Gemmini...
cpu time taken: 5542 ms,         gemmini time taken: 4 ms
performing 2 C = AB + D operations with CPU and Gemmini...
cpu time taken: 5540 ms,         gemmini time taken: 4 ms
performing 2 C = AB + D operations with CPU and Gemmini...
cpu time taken: 5544 ms,         gemmini time taken: 4 ms
performing 2 C = AB + D operations with CPU and Gemmini...
```

Figure 9: EagleX Gemmini Acceleration Results from Silicon

The second preliminary instance of Gemmini was fabricated in the Intel 22FFL process. The Beagle chip included general-purpose and machine-learning domains. The machine-learning domain consisted of a Rocket in-order processor connected to a Gemmini systolic-array DNN accelerator. The Rocket instance was configured as a small Linux-capable 5-stage in-order RV64GC core, which included a 16KiB data and instruction cache as well as a small 2-level branch predictor with a 512-entry branch history table and 14 entry branch target buffer. Tightly coupled to the Rocket core over its RoCC interface is the Gemmini systolic array DNN accelerator. This Gemmini instance is configured to contain a 16x16 systolic array of 8-bit multiply-accumulate processing elements supporting runtime-programmable weight stationary (WS) and output stationary (OS) dataflows. This instance was somewhat more complete, as it included some DNN functional units, such as ReLU, ReLU6, and accumulation quantization. Operand matrices are stored in 256KiB of scratchpad memory and 64KiB of accumulator memory.

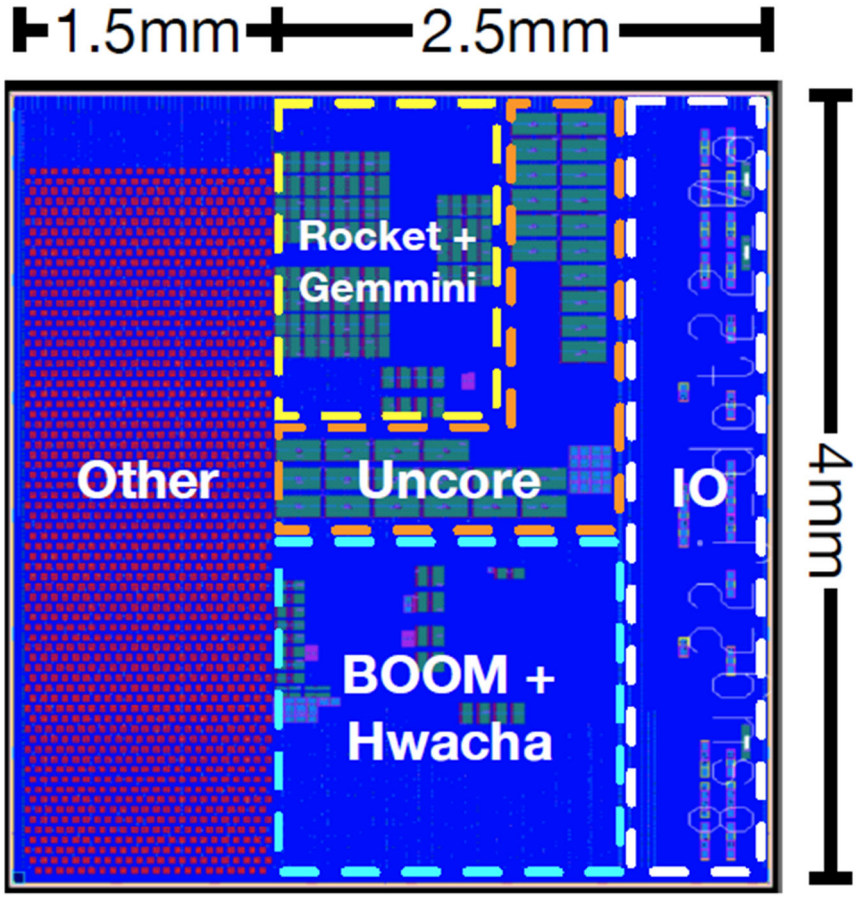

Figure 10: Beagle SoC Layout

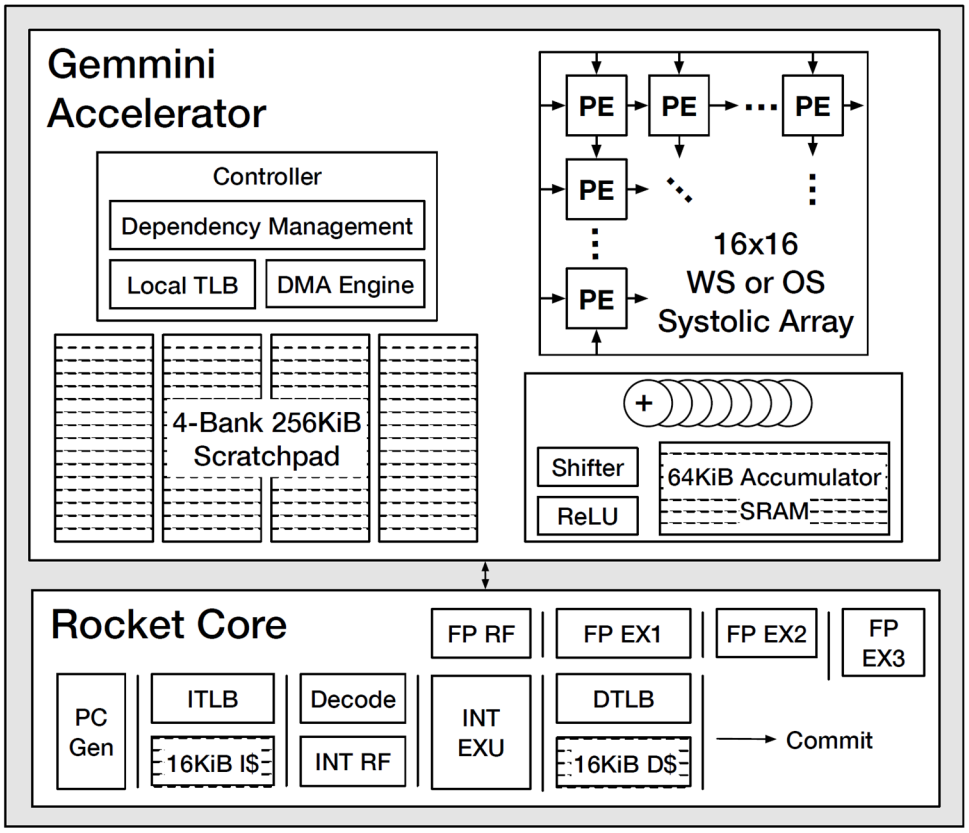

Figure 11: Beagle SoC Block Diagram

Figure 12 below demonstrates the heterogeneous nature of the Beagle SoC and the ability to run an application across all cores and accelerators with varying levels of efficiency. Specifically, an example general matrix multiplication (GEMM) benchmark, a common kernel of DNN and linear algebra workloads, of increasing size can be executed across the following compute: the BOOM out-of-order core with speculation enabled (full BOOM) and performance gated (non-speculative BOOM), the Rocket in-order core, the Hwacha vector accelerator, and the Gemmini DNN accelerator using a weight-stationary (WS) and output-stationary (OS) dataflow. When compared to the baseline non-speculative BOOM core performance, the Rocket core obtains a 4.0-4.9x speedup while the full BOOM core obtains 7.1-16.6x speedup. As expected, the data-parallel Hwacha and Gemmini accelerators obtain orders of magnitude greater speedups. Hwacha, coupled to the BOOM core, achieves up to 677x while Gemmini, coupled to the Rocket core, leads in performance with an improvement of 1297.2x by explicitly accelerating GEMMs. The performance gains are echoed in efficiency with the Gemmini WS and OS modes reaching up to 106.1 GOPS/W. Following are Hwacha and Rocket which reach up to 9.0 GOPS/W and 410.2 MOPS/W, respectively. Both the non-speculative BOOM and full BOOM modes are the least efficient at 10.8 MOPS/W and 176.6 MOPS/W, respectively. Finally, measurements were obtained with all cores and accelerators running at 210 MHz at 0.85 V to obtain relative results.

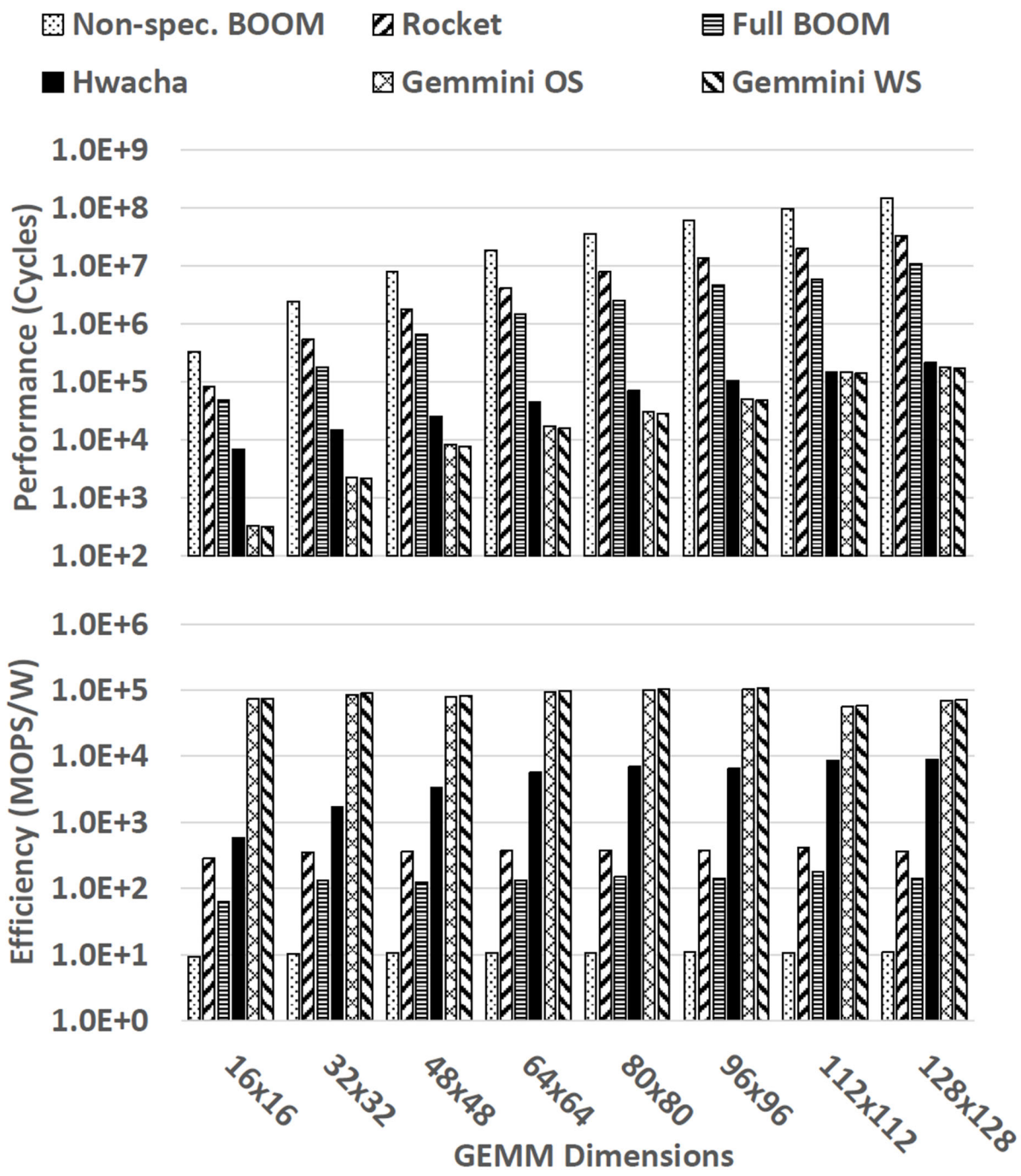

Figure 12: Comparative Performance of Heterogeneous Processors on Beagle

Figure 13 below shows the peak frequency and efficiency of a 16x16 GEMM benchmark across the operating voltages of the ML domain. Peak efficiency for Rocket and the Gemmini WS mode occurs at 0.75 V at 385.9 MOPS/W and 73.3 GOPS/W, respectively. Additionally, peak frequencies roughly match across the domain with a range of 675-961 MHz. While the uncore and ML domains were signed off at 500 MHz and the general-application domain was signed-off at 700MHz, the general-application domain contained an unconstrained asynchronous reset path preventing peak frequencies past 210 MHz at any voltage.

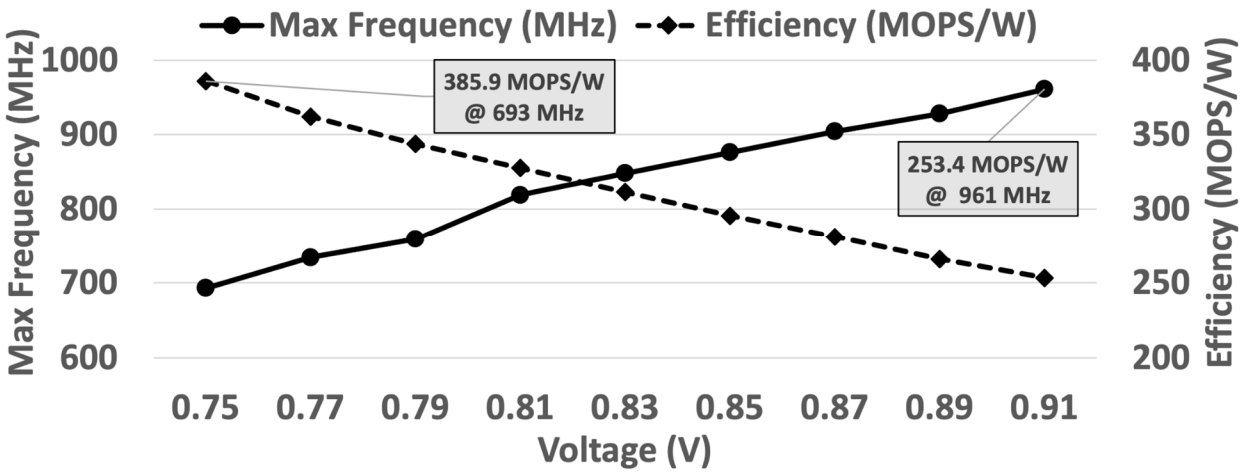

Figure 13: Beagle Measured Energy-efficiency versus Frequency Plots for GEMM

Finally, fully-featured instances of Gemmini were included in the Argo test chip, fabricated in Globalfoundries 12nm process technology. Argo included four Gemmini tiles, where each tile contained both a 16x16 int8 -> int32 Gemmini matrix multiplier and a 4x4 fp16 -> fp32 Gemmini matrix multiplier.

The Argo chip has been received from fabrication, and packaged and assembled after a 1.5-year delay due to pandemic-related semiconductor supply-chain issues. All cores are functional, and full performance evaluation is underway.

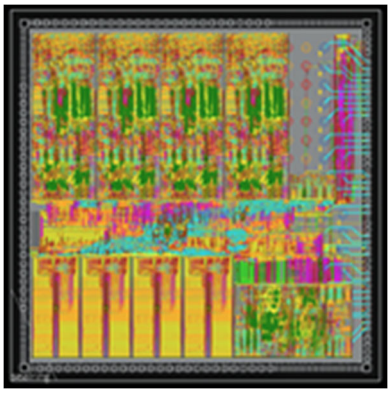

Figure 14: Argo Chip Layout

5 PUBLICATIONS

The following publications have been produced based on work undertaken under this award:

- Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao, **"Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration"**, Design Automation Conference (DAC-2021), San Francisco, CA, December 2021.
  Winner, Best Paper Award
- Hasan Genc, Seah Kim, Vadim Vadimovich Nikiforov, Simon Zirui Guo, Borivoje Nikolić, Krste Asanović and Yakun Sophia Shao, **"Gemmini: An Open-Source, Full-System DNN Accelerator Design and Evaluation Platform"**, Open-Source Computer Architecture Research Workshop (OSCAR-2022) at the 49th ACM/IEEE International Symposium on Computer Architecture (ISCA-2022), New York City, NY, June 2022.
- Yuka Ikarashi, Gilbert Louis Bernstein, Alex Reinking, Hasan Genc, and Jonathan Ragan-Kelley, **"Exocompilation for productive programming of hardware accelerators"**, Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022), San Deigo, CA, June 2022.
- Alon Amid, Hasan Genc, Jerry Zhao, Krste Asanović, Yakun Sophia Shao, and Borivoje Nikolić,**"Accelerating General-Purpose Linear Algebra on DNN Accelerators"**, 1st Workshop on Democratizing Domain-Specific Accelerators (WDDSA-2022) at the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO-2022), Chicago, Illinois, October 2022.
- Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao, **"MoCA: Memory-Centric, Adaptive Execution for Multi-Tenant Deep Neural Networks"**, 29th International Symposium on High-Performance Computer Architecture (HPCA-2023), Montreal, QC, Canada, February 2023.
- Seah Kim, Jerry Zhao, Krste Asanović, Borivoje Nikolić, and Sophia Shao, **"AuRORA: Virtualized Accelerator Orchestration for Multi-Tenant Workloads"**, 56th IEEE/ACM International Symposium on Microarchitecture (MICRO-2023), Toronto, Canada, October 2023.

5.1 Source Code Repositories and Documentation

The following source code repositories have been released:

- Gemmini: Berkeley's Systolic Array Generator:
  - https://github.com/ucb-bar/gemmini
- Exo: language & compiler for productive accelerator programming:
  - https://github.com/exo-lang/exo

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

ACRONYM	DESCRIPTION
CHANNA	Compiling Hardware Neural-Net Accelerators
DAC	Design Automation Conference
GMM	General Matrix Multiply
OS	Output Stationary
PE	Processing Elements
WS	Weight Stationary