The Past, Present, and Future of AI Engines

Alexander Redding
Dept. of Computer Science and Engineering
University of California San Diego
alredding@ucsd.edu

ABSTRACT

TODO: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi augue libero, ullamcorper sed scelerisque id, tincidunt eget ligula. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent ut sapien non lectus porta pulvinar. Aenean dui nisl, maximus at tortor a, aliquet consequat erat. Proin tempus blandit quam sit amet tristique. Etiam fermentum urna vitae tristique consectetur. In efficitur eros lacus, et aliquam ligula commodo vitae. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis eleifend sit amet lectus sit amet accumsan. Fusce eget pharetra ligula. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Pellentesque finibus elementum metus, quis lobortis dolor laoreet ac. In hac habitasse platea dictumst. Curabitur feugiat libero id eros ultricies, quis convallis ipsum hendrerit. Phasellus ut diam volutpat, venenatis arcu at, pharetra libero. Donec maximus in dolor at suscipit. Nullam gravida quam non nibh sodales, aliquam lacinia elit tincidunt. Aliquam faucibus sapien vel massa fermentum scelerisque. Vestibulum ut euismod erat. Cras non erat arcu. Vestibulum risus ante, malesuada ut imperdiet eu, tincidunt varius augue. Etiam fringilla lorem aliquam malesuada lacinia. Nunc eget ex eu magna dictum mattis eu a sapien. Duis rhoncus justo sed ligula suscipit dictum.

1 INTRODUCTION

The introduction of AlexNet in 2012 marked a pivotal moment which showcased the effectiveness of hardware acceleration for deep neural network (DNN) training on a graphics processing unit (GPU) [25]. AlexNet achieved a 2D convolution speedup and pipeline parallelism between two NVIDIA GTX 580s, reducing the training time of their convolutional neural network (CNN) on the ImageNet dataset to less than a week. General purpose GPUs (GPGPUs) have since been the preferred hardware choice for machine learning acceleration across datacenters, both for training and inference. Emerging software techniques and hardware additions have enabled increasing speedups through data and tensor parallelism, and optimizations for tensor operations.

While GPGPUs are effective for increasing ML workload throughput, researchers quickly realized the limitations of streaming multiprocessor (SMP) architectures. Parallel efficiency significantly decreases when threads diverge, control logic contributes to a non-negligible amount of die space and power consumption, and non-deterministic cache hierarchies make certain realtime applications impossible [12, 23]. At the same time AlexNet was published, adjacent research was spawned with the focus of designing dedicated compute architectures for accelerating ML operations.

The first dedicated architecture for machine learning acceleration was the original Neural Processing Unit (NPU) in 2012 [7, 12]. This NPU consisted of a network of eight identical processing engines, each with dedicated multiply-add and sigmoid activation hardware, as well as data buffers and a statically configured controller. Applications such as Sobel edge detection and Fast Fourier Transforms were accelerated, with many whole-applications achieving an average 2.3x speedup and an energy savings of 3x. Since the first NPU, "neural processing unit" has become a generic term for ML accelerators.

Shortly after the development of the NPU, Google announced their first version of their tensor processing unit (TPU) which was originally deployed in 2015 for inference workloads in datacenters [23, 40]. The first version of the TPU contained a 256x256 systolic array of 8-bit integer multiply-accumulators (MACs) and a large 24 MiB local buffer. Control logic occupied a minimal amount of die space, as the TPU immediately executed CISC instructions sent from a host machine. Compared to contemporary datacenter CPUs and GPUs, the TPU was many times faster and more energy efficient.

Beyond GPUs, ML accelerator architectures have become increasingly heterogeneous and specialized. As Moore's law declines, there is an increasing motivation to explore novel architectures to overcome the limits of technology scaling. Recently, the rise of large multi-gigabyte machine learning models has exacerbated the memory wall issue for ML accelerators, and has inspired further optimizations for on-chip data management.

1.1 The Memory Wall

Since the introduction of the transformer architecture in 2018 [47], the size of notable SoTA machine learning models has grown exponentially [49]. Large language models (LLMs) are (typically) encoder-decoder transformers trained for natural language processing (NLP) tasks, and the primary contributor to this growth in model parameters. While LLMs have exploded in popularity for both research and industrial applications, their underlying mechanisms are largely a "block box" and an active area of research. However, one property remains clear: LLMs trained with more parameters exhibit emergent abilities and yield increasingly better performance. That is, larger models trained with more parameters acquire unique abilities that aren't present in smaller models [50, 55].

Since at least the mid-90s, researchers have observed that computing resources are not scaling proportionately with off-chip memory bandwidth [32]; and the rise of large multi-gigabyte ML models has exacerbated the memory wall issue for the compute accelerators that execute them [55, 56]. This bottleneck can be reasoned about by using a workload's arithmetic intensity; or the ratio between the compute operations necessary to execute a unit of work, and the number of bytes that must be accessed [30, 55]. For example,

assume we have an NVIDIA A10 GPU with an ideal arithmetic intensity of 208.3 OPs/byte [35]. The standard implementation of the attention layer in Llama 2 7B [43] with float-16 weights and a sequence length of 4096 tokens has an arithmetic intensity of 62 OPs/byte [10, 46], far below our maximum compute capacity (about 30%).

The memory wall is also an issue for models with poor arithmetic intensity, not because they are large, but because of their high throughput inference requirements. Specifically, an increasing number of scientific fields are relying on high data-rate, low-latency neural networks (NNs) to process data recorded by sensors with extremely high sampling rates [52]. For example, take the autoencoder for lossy data compression of calorimeter readings in the CERN Large Hadron Collider [18]. This model has so few parameters that it uses <1% of our compute capacity with an arithmetic intensity of about .98 (assuming the model has float-16 weights for the sake of comparison). The challenge lies in inferencing data within the few nanoseconds between sensor readings. In practice, GPUs are not typically used for similar scientific applications as they usually have strict power constraints.

The memory wall has created a new interest in utilizing reconfigurable computing as the vehicle for ML acceleration [37]. Specifically, coarse-grained reconfigurable arrays (CGRAs) are seeing new applications as compute accelerators for machine learning inferences for the following reasons:

Flexibility. CGRAs offer the flexibility of creating specialized logic for different emerging neural network architectures and operations that application-specific integrated circuits (ASICs) simply cannot, and at a much lower price point. While field-programmable gate arrays (FPGAs) offer more flexibility than CGRAs, CGRAs trade this flexibility for increased clock frequencies and reduced compilation times [37].

Performance. When compared to datacenter GPUs, CGRA architectures tend to exhibit superior performance-per-area, even with inferior transistor node technology. Additionally, CGRAs allow for the creation of application specific optimizations such as customized pipelines for minimal latency [5].

Parallelism. CGRAs offer more opportunities to exploit parallelism as they allows for the creation of custom memory hierarchies and compute units. Additionally, they are able to concurrently process data from multiple external sensors through high-speed I/O directly interfaced with the fabric [5].

In 2020, AMD/Xilinx introduced their AI Engine (AIE), a CGRA architecture which consists of a 2D mesh of general purpose VLIW processors [3]. This paper is a survey of the AIE architecture, specifically looking at:

- How the AIE architecture compares with prior and contemporary CGRA designs, as well as current ML accelerators.
- Existing AIE toolchains and runtimes, both official and third party, across industry and academia.
- Applications deployed on AIEs.

2 BACKGROUND

Although largely dominated by GPGPUs, deep learning (DL) has become the latest focus for reconfigurable computing [28, 37]. While field-programmable gate arrays (FPGAs) and coarse-grained reconfigurable arrays (CGRAs) have traditionally been utilized for applications such as digital signal processing (DSP), both DSP and DL workloads share a common compute-intensive and highly parallel operation: matrix multiplication (MM). As of 2017, Google reported that 90% of their neural network (NN) inference workloads consisted of dense matrix multiplies [24, 56]. Predictably, this has only been exacerbated since the rise of large transformer models in the following years.

The follow section will provide a background on what coarse-grained reconfigurable arrays are, and how they came to be used for accelerating machine learning tasks. We will also look at the history of 2D processor mesh architectures as they lead up to the introduction of the AI Engine.

2.1 Coarse-Grained Reconfigurable Arrays

To understand what coarse-grained reconfigurable arrays are, it is essential to understand the reconfigurable architecture they descended from: field-programmable gate arrays. Originating in the mid-80s, FPGAs were developed to facilitate the simulation and testing of ASICs [37, 44]. Even today, taping out an ASIC comes with a high cost of error and a lengthy turnaround time of weeks to months. FPGAs were designed to simulate any digital logic design with a fine-grained reconfigurable fabric of logic cells known as look-up tables (LUTs). LUTs consist of programmable static random-access memory (SRAM) and a number of input and output wires. The SRAM defines a look-up table which describes a mapping between its input and output signals. FPGAs also contain many programmable interconnects to route connections between LUTs.

While FPGAs were originally developed as a simulation tool, they began to be used as general-purpose compute devices in their own right by the early 90s [37]. FPGAs saw use in telecommunication, military, and automotive applications that required custom, small-scale, high-performance logic that didn't require the efficiency of an ASIC. However, FPGAs have a number of limitations that reduce their effectiveness for general purpose compute. Fine-grained programmability comes at the cost of reduced clock speeds, with many designs running between one to two orders of magnitude slower than an ASIC counterpart [37]. Finding the ideal synthesis, placement, and routing of logic elements means that the electronic design automation (EDA) flow can take hours to days for large designs. About 90% of FPGA fabric consists of interconnects which largely increases the routing problem space, and dedicates a significant amount of power and area away from compute [14]. Additionally, essential arithmetic logic did not map well to the FPGA architecture as blocks such as a simple integer MAC unit could consume a large fraction of an FPGA's resources [37].

Coarse-Grained Reconfigurable Arrays were developed in order to address the limitations of FPGAs by trading some reconfigurability to coarsen certain computing elements in order to make better us its silicon, improve clock speeds, and reduce compilation times. Early CGRAs provided a small amount of coarsening over FPGAs, with architectures like Garp [19] being programmable at a 2-bit

granularity, instead of 1-bit like an FPGA [37]. Following CGRAs like CHESS [31], REMARC [34], and MATRIX [33] added higher bit-width programmability, contained hard compute elements such as programmable ALUs and register files, and introduced new network topologies which enabled communication between neighboring compute elements.

It should be noted that many early CGRAs were tightly-coupled accelerators (TCAs): co-processors which "consist of one or more specialized hardware functional units which can accelerate critical portions of an application kernel" [9]. TCAs are embedded into, or closely with the host processing core, and are typically accessed via instructions extended onto the host instruction set architecture (ISA). The host core shares resources with the TCA, such as register files, memory-management logic, and caches; which means that the host core can stall during TCA execution.

In 2000, MorphoSys [41] and PipeRench [16] added new features now common in modern CGRAs, including the addition of a hardware integer multiplier in the ALU, and new network topologies and virtualization techniques [37]. These were some of the first loosely-coupled accelerators (LCAs): co-processors which are located outside of the host processing core and are capable of direct, independent access to global memory [9]. This allows for more complex accelerator designs than with TCAs, since LCAs are not at risk of degrading the host processor's performance. The hardware additions and network structures introduced by MorphoSys and PipeRench provided the basis for many modern CGRAs [37].

2.2 2D Processor Meshes

The 2D mesh topology is the most commonly used network topology between compute units in CGRAs [37]. Compared to other network topologies, meshes allow for dynamically reconfigurable dataflows which can be scheduled by a compiler, time-sharing or context-switching of resources, and a larger degree of scalability. Mesh topologies are used to network reconfigurable cells of varying complexity, from integer ALUs with a register file [41], to general purpose processors.

This section will focus on 2D meshes of general purpose processors, which can also be considered as distributed memory systems. Distributed memory systems try to optimize access to global resources, such as I/O and global memory, by using multiple processing nodes which are interconnected via a high-speed network [54]. Nodes each have some amount of local memory and communicate with each other via message passing or shared memory.

TODO: Add generic distributed memory diagram

Figure 1 shows the subsection of a traditional multiprocessor GPGPU and its memory hierarchy (using the NVIDIA naming convention). While this figure depicts a GPU, its memory hierarchy is similar to other general purpose multiprocessor systems. Compared to memory hierarchy shown in Figure 5, which demonstrates the interconnects between AI Engine tiles (as will be explored later in Section 3), distributed memory systems can avoid or reduce contention for global memory. Memory behavior is more deterministic, as traditional cache flushes will not occur. This comes at the cost of added complexity: data movement must be managed manually; kernel programming must adapt new communication models as processor nodes may not share any address space, or may lack

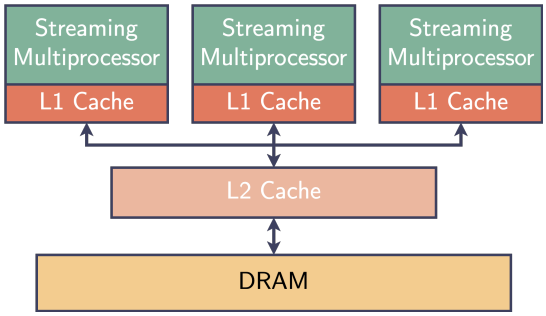

Figure 1: Traditional GPGPU Memory Hierarchy

shared memory all together; and local memories may be smaller than a traditional L1 cache.

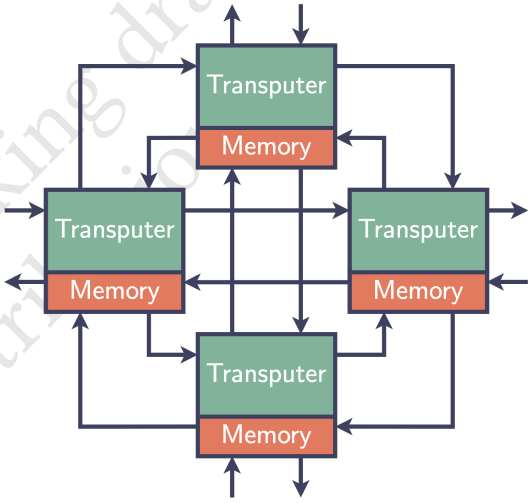

Figure 2: Transputer Network

2.2.1 Transputers. Developed in the early 1980s, Transputers were one of the first processors explicitly designed for parallel, distributed memory computing [8, 26]. A Transputer was a discrete processor which consisted of a stack machine, some local memory, and communication links [53]. Multiple nodes were connected together to form large Transputer networks which could be easily scaled, as seen in Figure 2.

Transputers were designed around the Occam programming language [22, 36], one of the first programming languages built on the concepts of communicating sequential processes [21]. Occam introduced parallel programming concepts for reasoning about and managing concurrency, such as channels [39], which mapped well onto the Transputer hardware. Many of these concepts are now found in modern networking languages, such as Go [6, 17].

Transputers saw use in early supercomputing [8, 20], as well as reconfigurable computing research. By 1995, Transputers were being paired with FPGAs for parallel application acceleration [27]. While Transputers weren't CGRAs, the idea of combining a 2D mesh of hard processor cores, along with some programmable logic

to supplement the processors hardware, is still in use today and can be seen in modern designs such as the AI Engine [3].

2.2.2 CGRAs. One of the first CGRA designs to feature a 2D mesh of general purpose processors was the 167-processor[37, 45], designed in 2009 to accelerate DSP workloads. The 167-processor contains 164 homogeneous processor nodes and introduced new multi-core clocking techniques to reduce power consumption. The processor nodes have relatively simple 16-bit architectures, containing only a fixed-point ALU and multiplier; a 40-bit accumulator; a 128x35-bit instruction memory and 128x16-bit data memory; and two 64x16-bit FIFOs for interprocess communication (IPC) [45]. All processors share three on-chip 16KB global memories. Similar to the Transputer [8], 167-processor nodes each have 4 bidirectional asynchronous communication links between neighbors.

The RHyME/REDEFINE [11, 29, 37] architecture was introduced in the mid-2010s to accelerate MM-based high-performance computing (HPC) applications, and consists of a 6x4 2D mesh of HyperCells. HyperCells are processing nodes with a dataflow architecture [48] designed for executing dataflow graphs. Each HyperCell has an 8x64-bit register file and 16KB of configuration memory [11]. A multi-core HyperCell system has 768 KB of global memory and features a honeycomb network topology which is managed by the RECONNECT network-on-chip (NoC) architecture [15].

Plasticine [37, 38] is an accelerator for general parallel applications which was published in 2017. A Plasticine mesh consists of two kinds of "pattern units": pattern compute units and pattern memory units. A pattern compute unit contains an ALU with floating-point support, as well as single-instruction-multiple-data (SIMD) vector support; control logic; and communication FIFOs. Pattern memory units contain scratchpad memory for the pattern compute units, and address generation units (AGUs) for global memory access. The Delite Hardware Definition Language [42] provides parallel pattern constructs for users to map their workloads to and is used to statically configure Plasticine. Plasticine is one of the first CGRAs to be used for machine learning acceleration, and one of the few to be usable for accelerating training (stochastic gradient descent).

2.3 CGRAs for Machine Learning

Machine learning training typically requires model parameters to be represented with high-precision, floating-point data types. Inference, however, does not require as much precision and can operate on quantized model weights. Quantization is the process of reducing the precision of a floating-point value by projecting it onto a lower bit-width representation. Many models are able to be quantized to integers of 8 bits or less [51]. Integer quantization also has the effect of significantly reducing arithmetic logic (integer ALU instead of floating-point), making weights easier and faster to compute with.

While GPGPUs are typically the dominant hardware choice for ML training, CGRAs are being explored as the vehicle to exploit the efficiency of lower dimensional datatypes for inference tasks. This is a natural extension of previous CGRA work, as CGRA architectures have historically been optimized for integer MAC-heavy workloads, and for matrix multiplication. Additionally, GPGPUs are typically designed to achieve the absolute best compute performance at the cost of high power consumption, whereas CGRA architectures are typically optimized for size and power efficiency [37]. For edge deployments which only require inference tasks, CGRAs may be the preferable choice for acceleration.

DT-CGRA was one of the first CGRAs explicitly designed for accelerating machine learning inference [13, 37].

TODO: Finish DT-CGRA (2016) notes
TODO: Add Eyeriss + Eyeriss v2 (2016, 2018) notes
TODO: Add NVDLA notes
TODO: Mention Cerebras Wafer Scale Engine
TODO: Transition to AIE Architecture section

3 AI ENGINE ARCHITECTURE

In 2018, AMD/Xilinx announced the first generation of the AI Engine architecture as part of their new Versal Adaptive Compute Acceleration Platform (ACAP) [2]. The Versal ACAP is a line of FPGA system-on-chips (SoCs) which contain an Arm Cortex-A72 application processor, programmable logic, NoC, and an AI Engine array. An AI Engine array contains a 2D mesh of AI Engine tiles. Each tile contains a 6-way VLIW processor with a 32-bit scalar RISC processor, 512-bit fixed-point vector unit; 512-bit floating-point vector unit; two vector load units; and one vector store unit [4]. AMD/Xilinx has not publicly released the instruction set for the AI Engine, but the instruction format can be seen in Figure 4. A diagram of the AI Engine functional units can be seen in Figure 6. In addition to a VLIW processor, an AI Engine tile also contains a 1024x128-bit (total of 16KB) read-only program memory; data memory interface; cascade stream interface; memory-mapped AXI4 debug interface; and an AXI4-Stream interface consisting of two 32-bit input and two 32-bit output ports.

TODO: Include initial ISA reverse engineering work (table and/or GitHub)

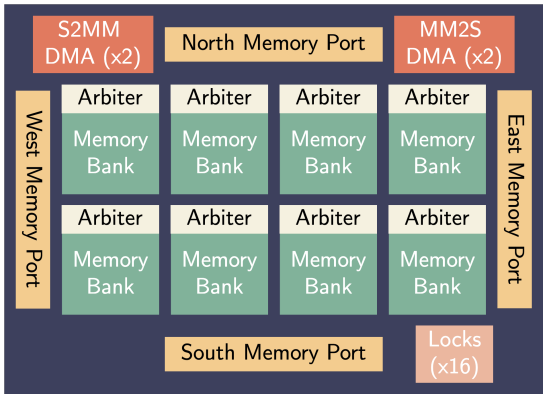

Figure 3: AI Engine Memory Module

3.1 Intra-AIE Communication

AI Engine tiles can communicate with each other through three ways:

(1) Shared memory
(2) Cascade stream
(3) AXI4-Stream

3.1.1 Shared memory. In-between all AI Engine tiles are AI Engine memory modules [4], which can be seen in Figure 3. In the original AI Engine architecture, each memory module contains eight banks of 256x128-bit (total of 32KB) single port memory. The first two of these banks have error-correcting code (ECC) protection, while the rest have a parity check. The ECC protection can detect and correct 1-bit errors, and detect 2-bit errors per each 32-bit word. Memory tiles contain two input AXI4-Stream to memory-map DMA interfaces, and two output memory-map DMA to AXI4-Stream interfaces. Additionally, memory modules have synchronization logic including 16 hardware locks and a round-robin request arbitrator to satisfy one new request per cycle.

While AMD/Xilinx literature depicts memory modules as being integrated with AI Engine tiles (as seen in Figure 5), they can be thought of as separate tiles whose memory is shared between neighboring AI Engine tiles on the North, South, East, and West.

Because the memory modules are shared between neighboring AI Engine tiles, they can communicate through shared memory via their data memory interfaces. This means that under ideal scheduling, a maximum of four local AI Engine processors can each store two and load one 256-bit vector(s) into a shared buffer. Shared memory can be used to create pipeline and graph dataflow kernel execution models.

If an AIE tile wishes to access a non-neighboring memory module, it can use a memory-mapped AXI4 to AXI4-Stream interface (located on a local memory module) to do this, at the cost of additional latency since all stream ports have a 32-bit width (memory modules are 128-bits wide). DMA can be used in this way to synchronize ping-pong style memory buffers between AI Engine tiles.

TODO: Add table of communication methods and memory bandwidths

3.1.2 Cascade stream. **TODO: Add accumulator stream notes**

3.1.3 AXI4-Stream. **TODO: Add AXI4-Stream interconnect notes**

3.2 External Array Communication

TODO: Add PL interface notes, (not present on Ryzen AIE)
TODO: Add Configuration interface notes, explain how AIEs are programmed
TODO: Add NoC interface notes, explain how AIEs access DRAM

3.3 Variants

TODO: Briefly explain other AIE variants (AIE-ML, AIE-ML v2, AIE-ML in Ryzen)
TODO: Add table showing differences (vector datatypes, clock speeds, power)

3.4 Motivation

In a 2020 white paper titled "Architecture Apocalypse", AMD/Xilinx discusses the motivation behind their development of the Versal ACAP [1]. The author explains that the Versal AI Engine array is a multi-core architecture that essentially lies somewhere in-between a GPU, and a dedicated ML accelerator ASIC like the Google TPU.

TODO: Finish adding AIE motivation notes (not that good of a paper...)

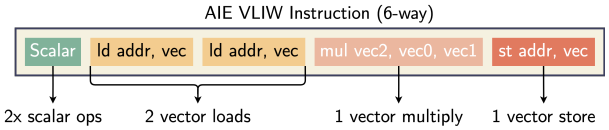

Figure 4: AI Engine Instruction Format

Table 1: AI Engine Variants

	Release	test	test
AIE	2020	test	test
AIE-ML	2022	test	test
AIE-ML v2	2024	test	test

TODO: Show connection to global memory, or make generic distributed memory diagram

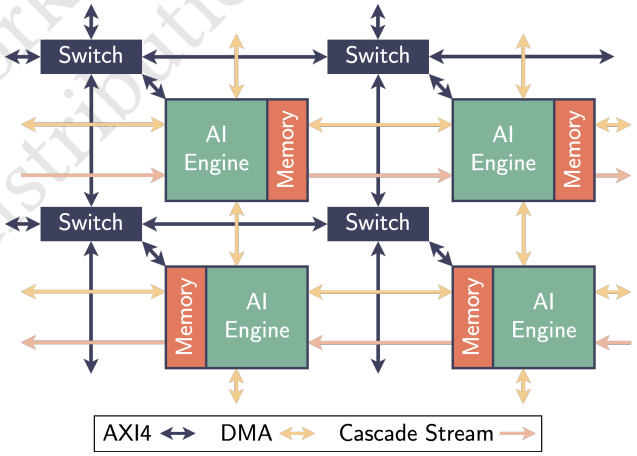

Figure 5: AI Engine Interconnects

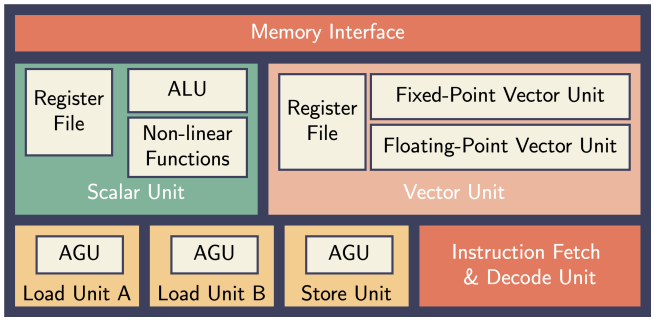

Figure 6: AI Engine Processor

4 AI ENGINE DEVELOPMENT

4.1 Closed Source

TODO: Add official Xilinx tools

4.2 Open Source

TODO: Add Graphtoy notes
TODO: Add MLIR notes, although kind of half open source. Maybe separate subsection
TODO: Add GEMMs here, or keep in runtime section. Large overlap...
TODO: Add CHARM (2023) notes
TODO: Add CHARM-2 (2023) notes
TODO: Add MaxEVA (2023) notes

5 AI ENGINE RUNTIMES

TODO: Add Vyasa (2020) notes

6 AI ENGINE APPLICATIONS

6.1 Graph Neural Network Acceleration

TODO: Add H-GCN (2020) notes
TODO: Add "Exploiting" (2023) notes

6.2 Transformers

TODO: Add SSR (2024) notes
TODO: Add feasibility analysis comparing NVIDIA A100
TODO: Mention BERT example in AIE-MLIR

6.3 Finance

TODO: Add price discovery (2024) notes

6.4 Space

TODO: Add radar notes TODO: Summarize space related works since not actually deployed

7 DISCUSSION

TODO: Add power notes TODO: Add programming challenges notes

REFERENCES

[1] Gupta Alok. 2020. Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core. *Embedded World* (2020).

[2] AMD/Xilinx. 2018. Xilinx AI Engines and Their Applications (WP506). https://docs.amd.com/v/u/en-US/wp506-ai-engine.

[3] AMD/Xilinx. 2020. Versal: The First Adaptive Compute Acceleration Platform (ACAP). https://docs.amd.com/v/u/en-US/wp505-versal-acap.

[4] AMD/Xilinx. 2023. Versal Adaptive SoC AI Engine Architecture Manual (AM009). https://docs.amd.com/r/en-US/am009-versal-ai-engine.

[5] Andrew Boutros, Aman Arora, and Vaughn Betz. 2024. Field-Programmable Gate Array Architecture for Deep Learning: Survey & Future Directions. arXiv:2404.10076 [cs.AR]

[6] Matilde Broløs, Carl-Johannes Johnsen, and Kenneth Skovhede. 2021. Occam to Go translator. In *2021 IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference (COPA)*. 1–8. https://doi.org/10.1109/COPA51043.2021.9541431

[7] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. 2020. A Survey of Accelerator Architectures for Deep Neural Networks. *Engineering* 6, 3 (2020), 264–274.

[8] Cambridge Cherry Hinton and Ruth Ivimey. [n. d.]. Legacy of the transputer. *emergence* 80186 ([n. d.]), 19.

[9] Emilio G Cota, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca P Carloni. 2015. An analysis of accelerator coupling in heterogeneous architectures. In *Proceedings of the 52Nd Annual Design Automation Conference*. 1–6.

[10] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2024. FLASHATTENTION: fast and memory-efficient exact attention with IO-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA.

[11] Saptarsi Das, Nalesh Sivanandan, Kavitha T. Madhu, Soumitra K. Nandy, and Ranjani Narayan. 2016. RHyMe: REDEFINE Hyper Cell Multicore for Accelerating HPC Kernels. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. 601–602. https://doi.org/10.1109/VLSID.2016.29

[12] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 449–460. https://doi.org/10.1109/MICRO.2012.48

[13] Xitian Fan, Huimin Li, Wei Cao, and Lingli Wang. 2016. DT-CGRA: Dual-track coarse-grained reconfigurable architecture for stream applications. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 1–9. https://doi.org/10.1109/FPL.2016.7577309

[14] Umer Farooq, Zied Marrakchi, and Habib Mehrez. 2012. *FPGA Architectures: An Overview*. Springer New York, New York, NY, 7–48.

[15] Alexander Fell, Prasenjit Biswas, Jugantor Chetia, S.K. Nandy, and Ranjani Narayan. 2009. Generic routing rules and a scalable access enhancement for the Network-on-Chip RECONNECT. In *2009 IEEE International SOC Conference (SOCC)*. 251–254. https://doi.org/10.1109/SOCCON.2009.5398048

[16] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matthew Moe, and R Reed Taylor. 2000. PipeRench: A reconfigurable architecture and compiler. *Computer* 33, 4 (2000), 70–77.

[17] Google. [n. d.]. Go. https://go.dev.

[18] Giuseppe Di Guglielmo, Farah Fahim, Christian Herwig, Manuel Blanco Valentin, Javier Duarte, Cristian Gingu, Philip Harris, James Hirschauer, Martin Kwok, Vladimir Loncar, Yingyi Luo, Llovizna Miranda, Jennifer Ngadiuba, Daniel Noonan, Seda Ogrenci-Memik, Maurizio Pierini, Sioni Summers, and Nhan Tran. 2021. A Reconfigurable Neural Network ASIC for Detector Front-End Data Compression at the HL-LHC. *IEEE Transactions on Nuclear Science* 68, 8 (2021), 2179–2186. https://doi.org/10.1109/TNS.2021.3087100

[19] John R Hauser and John Wawrzynek. 1997. Garp: A MIPS processor with a reconfigurable coprocessor. In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No. 97TB100186)*. IEEE, 12–21.

[20] Anthony J. G. Hey. 1990. Supercomputing with transputers—past, present and future. *SIGARCH Comput. Archit. News* 18, 3b (jun 1990), 479–489. https://doi.org/10.1145/255129.255192

[21] C. A. R. Hoare. 1978. Communicating sequential processes. *Commun. ACM* 21, 8 (aug 1978), 666–677. https://doi.org/10.1145/359576.359585

[22] Christian L Jacobsen and Matthew C Jadud. 2004. The transterpreter: a transputer interpreter. *Communicating Process Architectures 2004* (2004), 99–106.

[23] Norman Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro* 38, 3 (2018), 10–19. https://doi.org/10.1109/MM.2018.032271057

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[26] Robert Kuhn and David Padua. 2021. *Parallel Hardware*. Springer International Publishing, Cham, 11–54.

[27] Adrian Lawrence, Andrew Kay, Wayne Luk, Toshio Nomura, and Ian Page. 1995. Using reconfigurable hardware to speed up product development and performance. In *Field-Programmable Logic and Applications*, Will Moore and Wayne Luk (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–118.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[29] Kavitha T. Madhu, Saptarsi Das, Nalesh S., S. K. Nandy, and Ranjani Narayan. 2015. Compiling HPC Kernels for the REDEFINE CGRA. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 405–410. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.139

[30] Dan C. Marinescu. 2018. Chapter 4 - Parallel and Distributed Systems. In *Cloud Computing (Second Edition)* (second edition ed.), Dan C. Marinescu (Ed.). Morgan Kaufmann, 113–150.

[31] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, and Brad Hutchings. 1999. A reconfigurable arithmetic array for multimedia applications. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*. 135–143.

[32] Sally A. McKee and Robert W. Wisniewski. 2011. *Memory Wall*. Springer US, Boston, MA, 1110–1116.

[33] Mirsky and DeHon. 1996. MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, 157–166.

[34] Takashi Miyamori and Kunle Olukotun. 1999. REMARC: Reconfigurable multimedia array coprocessor. *IEICE Transactions on information and systems* 82, 2 (1999), 389–397.

[35] NVIDIA. [n. d.]. NVIDIA A10 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/products/a10-gpu/.

[36] Conor O'Neill. [n. d.]. occam-2 language implementation manual. ([n. d.]).

[37] Artur Podobas, Kentaro Sano, and Satoshi Matsuoka. 2020. A survey on coarse-grained reconfigurable architectures from a performance perspective. *IEEE Access* 8 (2020), 146719–146743.

[38] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A Reconfigurable Architecture For Parallel Paterns. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 389–402.

[39] Anthony Roscoe. 1998. The theory and practice of concurrency. (1998).

[40] Cristina Silvano, Daniele Ielmini, Fabrizio Ferrandi, Leandro Fiorin, Serena Curzel, Luca Benini, Francesco Conti, Angelo Garofalo, Cristian Zambelli, Enrico Calore, Sebastiano Fabio Schifano, Maurizio Palesi, Giuseppe Ascia, Davide Patti, Stefania Perri, Nicola Petra, Davide De Caro, Luciano Lavagno, Teodoro Urso, Valeria Cardellini, Gian Carlo Cardarilli, and Robert Birke. 2023. A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. arXiv:2306.15552 [cs.AR]

[41] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J Kurdahi, Nader Bagherzadeh, and Eliseu M Chaves Filho. 2000. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE transactions on computers* 49, 5 (2000), 465–481.

[42] Arvind K Sujeeth, Kevin J Brown, Hyoukjoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2014. Delite: A compiler architecture for performance-oriented embedded domain-specific languages. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 4s (2014), 1–25.

[43] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[44] Stephen M. Trimberger. 2015. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proc. IEEE* 103, 3 (2015), 318–331. https://doi.org/10.1109/JPROC.2015.2392104

[45] Dean N. Truong, Wayne H. Cheng, Tinoosh Mohsenin, Zhiyi Yu, Anthony T. Jacobson, Gouri Landge, Michael J. Meeuwsen, Christine Watnik, Anh T. Tran, Zhibin Xiao, Eric W. Work, Jeremy W. Webb, Paul V. Mejia, and Bevan M. Baas. 2009. A 167-Processor Computational Platform in 65 nm CMOS. *IEEE Journal of Solid-State Circuits* 44, 4 (2009), 1130–1144. https://doi.org/10.1109/JSSC.2009.2013772

[46] Philip Kiely Varun Shenoy. [n. d.]. A guide to LLM inference and performance. https://www.baseten.co/blog/llm-transformer-inference-guide/.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[48] Arthur H Veen. 1986. Dataflow machine architecture. *ACM Computing Surveys (CSUR)* 18, 4 (1986), 365–396.

[49] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. 2022. Machine Learning Model Sizes and the Parameter Gap. arXiv:2207.02852 [cs.LG]

[50] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. arXiv:2206.07682 [cs.CL]

[51] Olivia Weng. 2023. Neural Network Quantization for Efficient Inference: A Survey. arXiv:2112.06126 [cs.LG]

[52] Olivia Weng, Alexander Redding, Nhan Tran, Javier Mauricio Duarte, and Ryan Kastner. 2024. Architectural Implications of Neural Network Inference for High Data-Rate, Low-Latency Scientific Applications. arXiv:2403.08980 [cs.LG]

[53] Colin Whitby-Strevens. 1985. The transputer. *ACM SIGARCH Computer Architecture News* 13, 3 (1985), 292–300.

[54] Peng Zhang. 2010. CHAPTER 17 - Distributed operating systems. In *Advanced Industrial Control Technology*, Peng Zhang (Ed.). William Andrew Publishing, Oxford, 685–732.

[55] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[56] Jinming Zhuang, Jason Lau, Hanchen Ye, Zhuoping Yang, Yubo Du, Jack Lo, Kristof Denolf, Stephen Neuendorffer, Alex Jones, Jingtong Hu, Deming Chen, Jason Cong, and Peipei Zhou. 2023. CHARM: Composing Heterogeneous AcceleRators for Matrix Multiply on Versal ACAP Architecture. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '23)*. Association for Computing Machinery, New York, NY, USA, 153–164.