Wave Computing™

A Coarse Grain Reconfigurable Array (CGRA) for Statically Scheduled Data Flow Computing

Author: Dr. Chris Nicol, Chief Technology Officer, Wave Computing

Abstract

This paper argues a case for the use of coarse grained reconfigurable array (CGRA) architectures for the efficient acceleration of the data flow computations used in deep neural network training and inferencing. The paper discusses the problems with other parallel acceleration systems such as massively parallel processor arrays (MPPAs) and heterogeneous systems based on CUDA and OpenCL, and proposes that CGRAs with autonomous computing features deliver improved performance and computational efficiency. The machine learning compute appliance that Wave Computing is developing executes data flow graphs using multiple clock-less, CGRA-based System on Chips (SoCs) each containing 16,000 processing elements (PEs). This paper describes the tools needed for efficient compilation of data flow graphs to the CGRA architecture, and outlines Wave Computing's WaveFlow software (SW) framework for the online mapping of models from popular workflows like Tensorflow, MXNet and Caffe.

The Parallel Programming Problem

CPUs have not delivered a substantial improvement in the execution performance of a single-thread C-program for the last decade. A typical PC in 2007 contained a CPU running at 3.4GHz with 2GB DRAM. Today, a typical computer will have a CPU running at a similar clock frequency and about 8 GB DRAM. Once processors hit this "clock wall", multi-core (as well as many-core and MPPAs) are used to increase the compute performance in systems without increasing the clock speed. One problem with these systems is that they are difficult to program in such a way that linear speed up is achieved. The challenge of developing a compiler that exploits the concurrency in a C program and partitions it efficiently across MPPA architectures is non-trivial and remains an open problem. The programmer must rewrite the program using frameworks like OpenMP for shared memory or Message Passing Interface (MPI). Refactoring a C-program for efficient multi-threaded execution is a non-trivial exercise and requires techniques taught in graduate programming courses. These are suitable for a modest number of processor cores in a multi-core system, however for many applications, they are not easily scalable to 100 or 1,000 cores, or more. The author of this paper first outlined this problem in 2010 [1]. Of course, this doesn't prevent people from trying (consider Kalray, Epiphany V, Tilera and KnuEdge). Even if it were possible to map computations across a large number of homogeneous cores, the dynamic distribution of memory and communication between the processors eventually limits the scalability of any compiled solution.

Acceleration Using Heterogeneous Architectures

Heterogeneous systems promise increased performance from a simpler programming model that enables a sequential C program to make calls to coprocessors that provide parallel speed-up. A control program executes on a CPU that uses a runtime API to initiate parallel execution of threads in an accelerator to speed up certain tasks like Basic Linear Algebra Subprograms (BLAS). CUDA and OpenCL are two examples of widely-used runtime APIs that enable heterogeneous computing. OpenCL aims to span all types of accelerator architectures from GPUs to FPGAs whereas CUDA is only for Nvidia GPUs. Most of the current machine learning systems use the heterogeneous computing architecture just described. One assumption made is that the transfer of control and data between the CPU and the accelerator is relatively insignificant (i.e. it carries a low overhead relative to the computation speed-up provided by the accelerator) and for some high throughput applications like image processing this can be true.

Accelerators in a heterogeneous system either transfer blocks of data to and from the CPU main memory, or they use a shared memory model. These solutions are not scalable because the accelerator is always tethered to the control code and main memory on the CPU. Eventually the communication between the CPU and the accelerator (typically over PCIe) becomes the bottleneck that limits scalability. There are new chip-to-chip interconnect proposals to address this (like NVLink from Nvidia [2] and CCIX [3]). These also provide coherency enabling the sharing of memory between the CPU and

Copyright © Wave Computing, Inc. All rights reserved.

accelerator. Most systems seem to max out at 4-8 GPUs per multi-core CPU. A big data problem that exceeds the capacity of 4-8 GPUs must be partitioned across multiple heterogeneous subsystems. The communication between the accelerators may be limited by communication back to the network that connects the CPUs together. In a data center, this translates to excessive traffic through the data center network.

Data Flow Computing on a Coarse Grain Reconfigurable Array (CGRA)

The Wave Computing approach uses something completely different to heterogeneous computing. Wave uses a data flow computing on a hybrid coarse grain/fine grain reconfigurable array (CGRA) of processors in a Wave dataflow processing unit (DPU). In this model, data flows between software kernels, which are called data flow agents. Each agent is compiled and statically scheduled across a reconfigurable array of data flow processing elements. The entire data flow computation is managed autonomously by the agents without the need for the control or memory of a host CPU. This leads to an efficient utilization of the arithmetic units in the Wave DPU. The programmer expresses computation at a high abstraction that spans multiple processors without knowledge of the underlying hardware. This differentiates the DPU from other approaches and addresses the software scalability problem that many-core architectures suffer. Machine learning workflows like Tensorflow[4] use static data flow graphs of tensors and map directly to this dataflow model of computation. Wave's data flow computer therefore serves as an ideal target for machine learning frameworks.

MPPA (Many Core)	Heterogeneous Computing	Host-less Data Flow CGRA
Intel Xeon Phi	Intel CPU + GPU (nVidia)	Wave Computing
Adapteva Epiphany	Intel + Nervana	
Tilera (Mellanox)	AMD Heterogeneous System Architecture	
Kalray	ARM + Mali, Imagination, MediaTek, Qualcomm	
KnuEdge	Tensilica, Ceva	

Coarse Grain Reconfigurable Array (CGRA)

A CGRA is a class of reconfigurable architecture that provides word-level granularity in a reconfigurable array to overcome some of the disadvantages of FPGAs. For an overview of CGRA architectures, refer to RaPiD [5], ADRES[6] and Mosaic [7] (from The University of Washington). CGRA architectures are a mesh of locally connected processors and are statically mapped at compile time. Rather than a compiler mapping a C program to a single core, a CGRA tool flow will map a high-

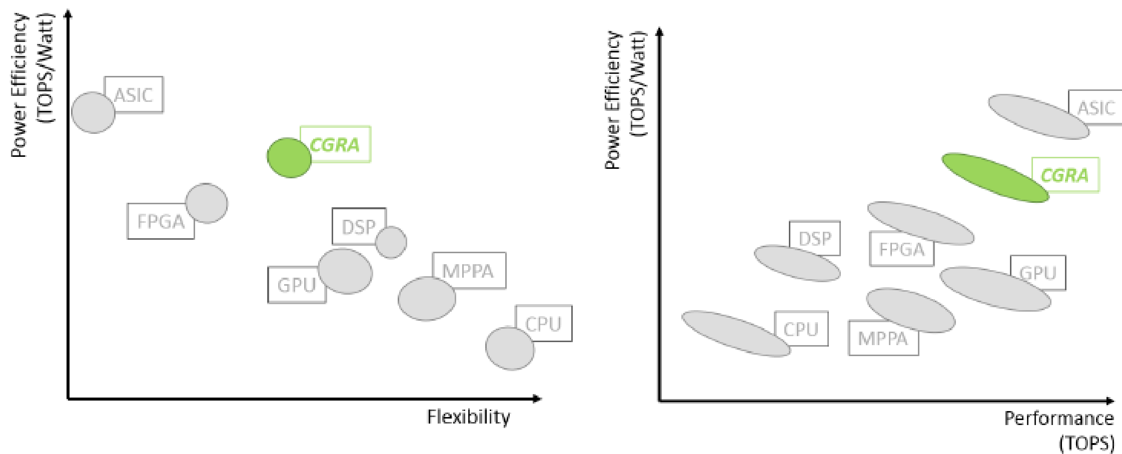

Fig 1. Power Efficiency vs Flexibility and Performance of CGRA vs CPU, DSP, MPPA, GPU, FPGA & ASIC

Copyright © Wave Computing, Inc. All rights reserved.

level program across multiple processing elements. A comparison is provided in Fig. 1 that compares the CGRA architecture against CPU, DSP, MPPA, GPU and FPGA. Power Efficiency is shown against flexibility and performance. Some things to note from these charts:

- DSPs have superior power efficiency to both CPU and GPU but lack scalable performance.
- MPPAs provide greater performance than CPU and have simpler cores, therefore greater efficiency than CPUs but inferior to GPUs.
- GPU power efficiency has surpassed CPUs and MPPAs over recent years (however GPUs still require multi-core CPUs to tell them what to do and when to do it).
- Modern FPGAs with DSP slices offer superior power efficiency to CPUs, MSPs, DSPs and GPUs, and provide strong performance (as expected from a fine grain reconfigurable architecture).
- CGRAs offer greater flexibility than FPGAs as they can be programmed efficiently in high level languages, and offer greater performance for machine learning applications due to the coarse grained nature of the calculations.
- Nothing beats an ASIC for computational efficiency, but everything beats an ASIC for flexibility.

The Wave CGRA architecture resembles the architecture in Fig. 2. This diagram shows what we call a “compute machine” that includes several CGRA clusters tethered to an AXI4 interface. Wave’s current chip has 24 of such machines configured as shown in Fig. 3.

CGRA architectures are sometimes confused with MPPAs, but they are quite different in terms of how they are programmed and the levels of efficiency they provide. As a rule, if each individual processor in the array is programmable using a C compiler, then it is a many core (MPPA) architecture and not a CGRA. Rather than operating from separate program

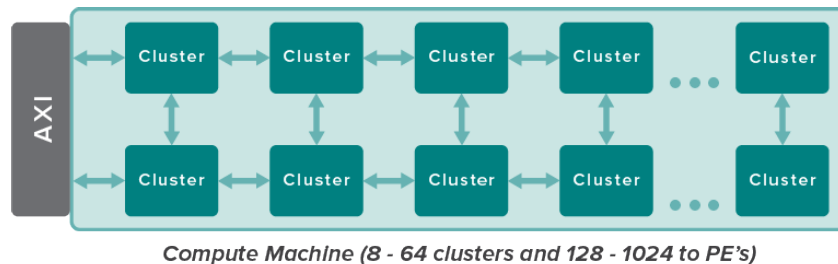

Fig. 2. Wave Computing CGRA structure. 2D mesh of clusters connected to AXI4 Interfaces.

counters, the PEs in a CGRA operate in lock-step and do not allow program branching. Each instruction memory is essentially a circular buffer, cycling through the same sequence of instructions. Rather than conditional execution via branching, a CGRA uses conditional execution of instruction streams.

CGRAs are not typically thought of as data flow processors. The work in [8] describes an excellent account of mapping multiple data flow kernels into CGRAs. We build upon the concepts described in this paper. Data flow computers are historically token-based systems, where the arrival of a token determined the computation that will be performed on it. Such systems are difficult to pipeline because instruction streams cannot be prefetched. The Wave CGRA uses a static schedule that enables the prefetching of instructions because the processor knows exactly what it will do when the next piece of data arrives. The data-flow aspect is applied at a level above the normal execution of the CGRA. If there is no data, the static schedule for a data flow agent is forgone entirely. Only when the data arrives does the static schedule for the data flow agent executes. At other times, the agent enters a low power sleep state. The system is therefore reactive at a level above the CGRA. The arrival of data automatically wakes up the agent (and several clusters used to execute it) from a sleep state (again without any control or intervention from a host CPU).

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

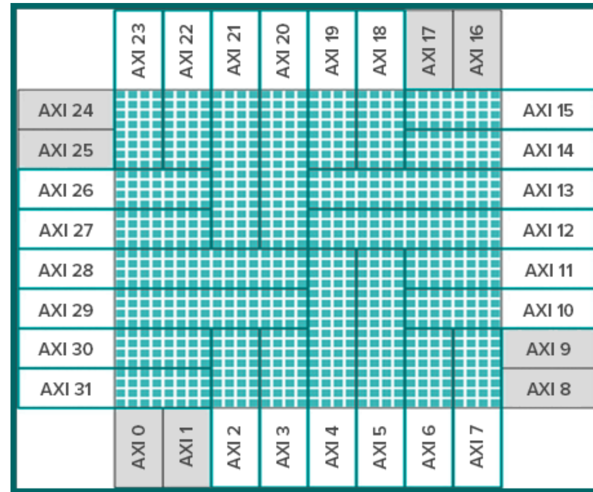

Fig. 3. A CGRA configured as a mesh of 32x32 clusters, and partitioned into 24 machines to provide multiple memory and data I/O ports to an SoC AXI4 NOC.

While there are several examples of CGRAs in literature, FPGAs (or fine grained reconfigurable architectures) are the only reconfigurable architecture that have achieved mainstream commercial success. The challenge with CGRAs is the development of a tool flow to schedule word-level coarse-grained computations into a mesh of PEs with a computational efficiency and functional density that surpasses FPGAs. Such tools should provide a way for the user to program the machine efficiently using popular high level programming languages like C/C++. FPGA architectures and tool flows have evolved to include word-level blocks like memories and DSP slices. Rather than starting with an FPGA architecture and enhancing it with time-multiplexed operations (like Tabula), the team at Wave Computing decided to create an efficient statically scheduled processing element that was an efficient target for a CGRA compilation flow.

CGRA PE Architecture

The architecture of a PE is shown in Fig. 4. The PE is completely different from a register-register RISC machine. Instead of drawing from and writing back to a multi-port register file, the PE uses a flow-through architecture where each result is forwarded to the immediate neighbors and can be used as an operand in the next cycle in one or more of the neighboring PEs. Each PE contains an accumulator to store its result. The contents of the accumulator can be fed back to the PE input as one of the operands for the next instruction. The accumulator microarchitecture, combined with the static schedule, provides the execute unit with two of the three inputs ahead of time. The PE takes advantage of this to perform pre-computation on the instruction and the accumulator value while the value from the neighboring PEs is in transit. When the 3rd input arrives, the PE evaluates very quickly. With this mechanism, the PE can compute and share data with its neighbors

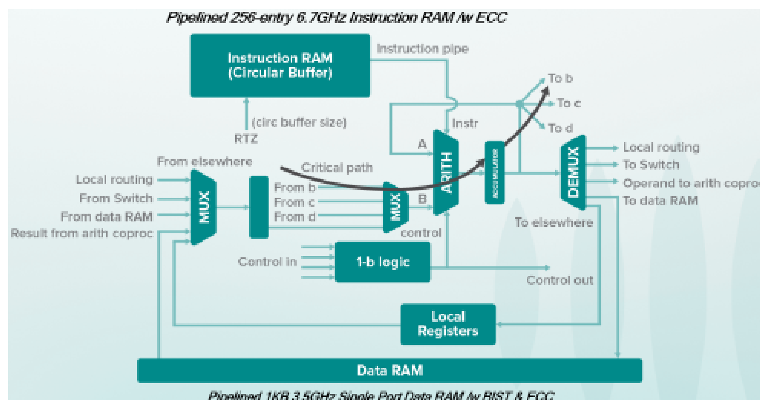

Fig. 4. Microarchitecture of a 6.7GHz Processing Element (PE)

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

at a sustained rate of 6.7GHz. Any combination of neighboring PEs can use the result as an operand in the very next cycle – making data flow operations with fan-out incredibly fast and efficient to execute. The circuitry is tuned to ensure that it is faster and more efficient to move data from one processor to the next, than it is to hold it in a local register for consumption at a later time. Arithmetic operations like multiply-accumulation are sent to arithmetic processors that behave as coprocessors, executing in parallel with the PEs. This means the PEs can perform other operations (like moving data to and from memory) while the MAC units are operating.

CGRA Cluster Architecture (of 16 PEs)

In Fig. 5, 16 Processing Elements are connected into a cluster, together with 8 arithmetic units and five switches for routing data across a 2-D mesh. The 16 PEs are configured as 4 quads, each containing 4 fully-interconnected PEs as described in Fig. 4. The quads can arbitrarily swap and shuffle data in each cycle making them ideal for switching, sorting and for implementing networks functions like trellis operations for channel decoding and neural networks. The quads are also fully

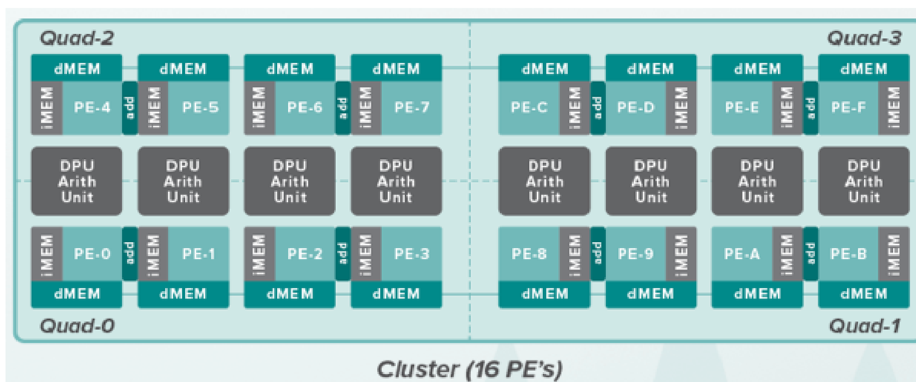

Fig. 5. CGRA Cluster of 16 PEs with 8 Arithmetic Units and 16KB data RAM

connected, so words of data can be swapped and shuffled at the maximum rate of the machine (albeit with a couple of cycles of latency). The memories are pipelined single port memories that enable both the PEs and the data switches to access the memories using static arbitration as scheduled by the compiler. The arithmetic units contain pipelined MAC units for BLAS operations needed by deep learning algorithms.

The arithmetic units are partitioned. They can perform 8-b operations in parallel (ideal for DNN inferencing) as well as 16-b and 32-b operations (or any combination of the above). Some 64-b operations are also available and these can be extended to arbitrary precision using software. The machine provides emulation of floating point with special functions like leading zero count.

Data Flow in a CGRA Computer

Programs in the CGRA are activated by the arrival of data (or control) for processing. When there is no data, the programs in the PEs place the cluster into a low power sleep state. In this state, the switching for the CGRA remains active – just the processors are placed to sleep. When data arrives, the cluster(s) of PEs are automatically brought out of one of two sleep states (deep or light) and re-aligned to the global static schedule such that they can consume the data as it arrives. There are multiple cycles of latency between the arrival of data and its consumption. This allows the circuits to be brought up to speed and the instruction streams to be filled. This latency is all managed by the tool flow that schedules both the arrival of data, and the instructions in the PEs to consume it. For data located in external memory (as is the case for large tensors), it is the arrival of control (presumably from another node in the data flow graph) that causes the clusters to come out of sleep. In which case, the control may cause the initiation of DMA transfers. The clusters may then go back to sleep until the transfers are complete. For the critical path of data flow computations, there is always data flowing and the system ensures a maximal overlap of communication (memory access) and computation using a flexible double buffering scheme for both operands and results.

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

Clockless CGRA Synchronization

The secret to achieving very high frequencies in a massively parallel array of PEs lies in the ability to overcome the cycle-to-cycle clock jitter that would otherwise constrain the period of a globally distributed clock. Wave decided NOT to distribute a global clock, but rather to use local synchronization circuits to safeguard local communication between clusters of PEs. Self-timed circuits ensure that each cluster of PEs is synchronized with its neighbors for the purposes of robust communication at the maximum rate of the machine. A 28nm test chip in 2012 demonstrated the feasibility of this approach. The chip contained a quad of 4 PEs that operated robustly across voltage and temperature variation at speeds exceeding 10 GHz. The processors computed results and shared them with neighbors such that they could each consume each other's results at the maximum rate of the machine. In the large-scale CGRA, data flows between clusters of PEs at a nominal frequency of 6.7 GHz without the need for FIFOs used in other Globally Asynchronous Locally Synchronous (GALS) schemes such as that described in the Kilocore MPPA project[9]. The CGRA has a bisection bandwidth of 8.6TB/s. The architecture of the self-timed synchronization (and the circuits used to implement it) are the topic of a publication to appear later this year. Wave will also reveal the mechanism for bootstrapping the synchronization network so that the PE all execute instruction 0 together.

Interfacing the CGRA to AXI4 SoC

Using self-timed circuits for synchronization means communication with the rest of an SoC requires extra care. Furthermore, the SoC uses a transaction based mechanism for communication, and the CGRA is statically scheduled. We use a patented architecture for providing multi-channel communication between AXI4 and the CGRA that avoids head of line blocking. It facilitates robust alignment of data to a static schedule as well as enables the CGRA to autonomously master transfers to and from external memories and devices in a 64-b address map. In all cases, flow control uses credit based mechanisms. Options are also available to use threshold-based flow control through the FIFOs. These circuits make extensive use of multi-channel FIFOs.

The Wave Computing Dataflow Processing Unit (DPU)

The Wave Computing DPU is an SoC that contains a 16,384 PEs, configured as a CGRA of 32x32 clusters. It includes four Hybrid Memory Cube (HMC) Gen 2 interfaces, two DDR4 interfaces, a PCIe Gen3 16-lane interface and an embedded 32-b RISC microcontroller for SoC resource management. The Wave DPU is designed to execute autonomously without a host CPU. In the Wave system, a small host is used for the purposes of initiating program loading, runtime reconfiguration, checkpointing and the like. The Wave Computing DPU is integrated into a dataflow computer that Wave aims to produce for enterprise and data center applications.

Programming the Wave CGRA Architecture

Instead of programming a single processor using a compiler, Wave Computing uses a tool flow to program several word-level processors configured as a 2-D Mesh connected CGRA. The input language is derived from any high level language that can generate an intermediate Low Level Virtual Machine (LLVM) representation. These include but are not limited to C/C++. Almost any programming language is compiled into a data flow graph and can therefore be used to program data flow agents into the DPU. The compiler identifies the dependencies between the computations to uncover opportunities for concurrency. Concurrent operations are automatically distributed across several processors. Sequential operations may be executed on the same, adjacent or distant processors, depending on the timing relationships and the critical path of the computation. The tools ensure that timing-critical computations are placed near each other and relax the placement of less critical computations. To do this effectively, the tools must manage computation, communication and memory allocation across a sea of distributed processors with distributed memories and a statically scheduled switch network.

The Wave Computing tool flow efficiently distributes the processing in the data flow graph across different processors and exploits all of the concurrency available in the input program, regardless of the source language. Once a program is mapped to a CGRA, the computations are very dense, very fast and very power efficient. The hard work is pushed to the tools at compile time so that the runtime system has the simpler task of placing and routing modules together to assemble a large data flow graph. Wave has already developed these CGRA tools that deliver exceptional functional density. Fig. 6 shows (in the background) a dataflow graph (WFG) generated by Wave's Compiler for a typical deep learning kernel. This is mapped across multiple processors using an advanced CGRA mapping tool flow. The instructions for a single cluster of 16 PEs are shown in the box to the left. Instruction slots (i.e. time) are shown vertically (with program counter = 0 at top)

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

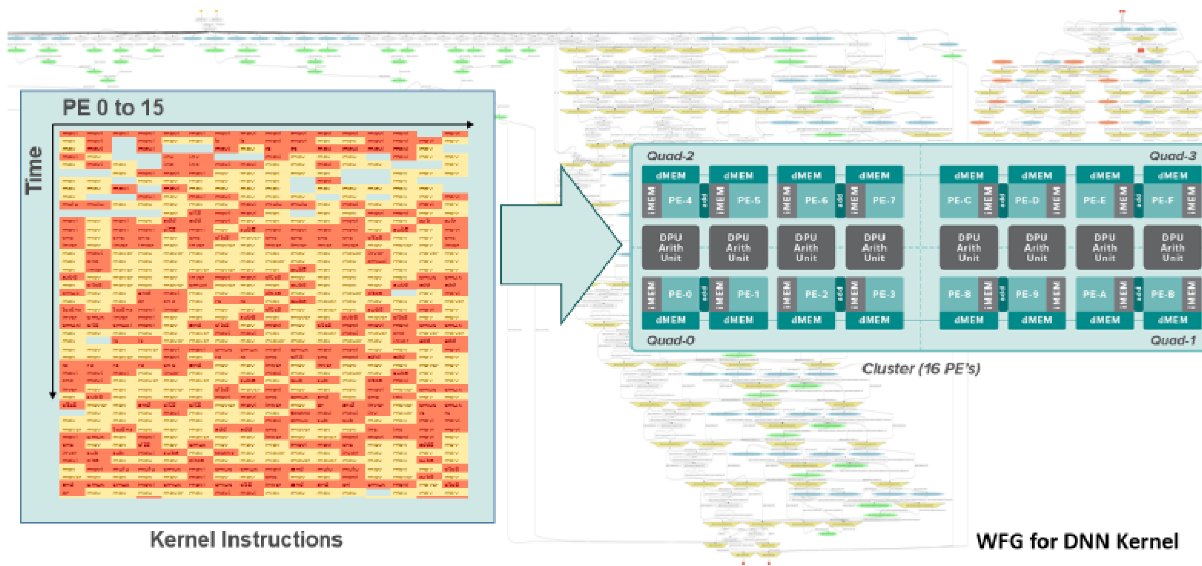

Fig. 6. Wave Computing CGRA tools map data flow graphs to instructions on multiple processors.

and the 16 PEs shown from left to right. The orange instructions indicate arithmetic and memory instructions, the yellow instructions are move instructions to move data between one or more processors. The remainder are NOP instructions.

Wave's "Ludicrous Mode" Kernel Mapping

Over a period of 5 years, Wave has developed these tools to the point where they out-perform the best assembler programmers using a method known as a Boolean Satisfiability.

For the kernel in Fig. 6, the SAT solver achieves a near-perfect 95% utilization of the available instruction slots. All kernels are compiled using relative addressing and are relocatable at run time to execute anywhere in the CGRA. To prepare the kernel for execution on the DPU chip, the instruction streams are fed through an assembler, compressed and encrypted. The runtime system in the DPU chip can load programs anywhere, and any number of times in the machine. Regions of external memory are assigned by the runtime and provided to the kernel via pointers. The Wave SAT solver is ideal for compiling the individual agents needed by our DNN agent library for machine learning. The Wave DNN Library team creates pre-compiled, relocatable kernels for common DNN functions used by workflows like TensorFlow. These can be assembled into Agents and instantiated into the machine to form a large data flow graph of tensors and DNN kernels.

The WaveFlow Framework

To facilitate the accelerated execution of data flow graphs from machine learning Workflows like TensorFlow, Wave is developing the WaveFlow SW framework shown in Fig. 7. Central to the WaveFlow framework is the WaveFlow agent library. This is a library of precompiled DNN operators that are used for training and inferencing of deep neural network models (examples include BLAS, Sigmoid, Apply Gradient Descent, Softmax, LSTM, etc). The WaveFlow session manager instantiates these into the DPU chips at runtime. An offline process is used to compile, link and verify agents for inclusion into the library. The WaveFlow SDK (including a modified C compiler, dataflow graph simulator and aforementioned CGRA SAT solver) is used to create the agents for the library offline. Wave intends to make this SDK available to customers to enable them to create their own agents for inclusion into the library.

To assemble an agent using the SDK, a DNN kernel is integrated with a software agent template that manages the flow of tensors of varying dimensions through (and between) the kernels. Agents can be linked together at runtime to form a large dataflow graphs. Agents support multiple inputs and outputs, each with flexible sized buffers that are located in high bandwidth memories. Agents can be linked across the CGAR in a single DPU chip, or across the CGRAs in multiple DPU chips. The communication between agents is asynchronous, double buffered, and managed through FIFO buffers. With this mechanism, robust processing of tensors is achieved without the need for a host CPU or runtime API like CUDA or OpenCL.

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

Wave Computing™

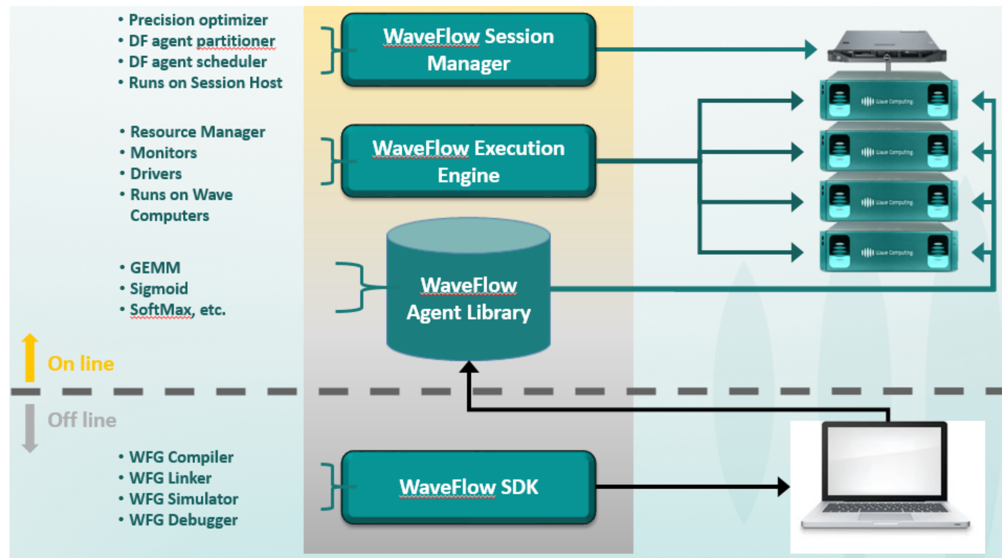

Fig. 7. WaveFlow SW Framework for mapping deep learning applications to dataflow computer.

Mapping Machine Learning Workflows to DPU Chips at Runtime

Wave is developing a session manager that interfaces with machine learning workflows like TensorFlow, CNTK, Caffe and MXNet as a worker process for both training and inferencing. These workflows provide data flow graphs of tensors to worker processes. At runtime, the Wave session manager analyzes data flow graphs and places the software agents into DPU chips and connects them together to form the data flow graphs. The software agents are assigned regions of global memory for input buffers and local storage. The static nature of the CGRA kernels and distributed memory architecture enables a performance model to accurately estimate agent latency. The session manager uses the performance model to insert FIFO buffers between the agents to facilitate the overlap of communication and computation in the DPUs. The variable agents support software pipelining of data flowing through the graph to further increase the concurrency and performance. The session manager monitors the performance of the data flow graph at runtime (by monitoring stalls, buffer underflow and/or overflow) and dynamically tunes the sizes of the FIFO buffers to maximize throughput. A distributed runtime management system in DPU-attached processors mounts and unmounts sections of the data flow graph at run time to balance computation and memory usage. This type of runtime reconfiguration of a data flow graph in a data flow computer is the first of its kind.

Advantages Over Heterogeneous-based Accelerators

The Wave Computing data flow computer is unique in that **NO CPU IS REQUIRED** to manage the flow of computation at runtime. This provides an enormous benefit over existing heterogeneous computing systems that must manage the synchronization of the sequential threads on a CPU with the parallel threads on an accelerator. Moreover, by removing the CPU, the system becomes truly scalable to execute data flow graphs of almost arbitrary size. Each 3U Wave data flow computer contains 2TB of DDR4 DRAM, 128GB high bandwidth HMC memory and several TB of SSD storage. These machine learning super-computers exploit the performance and computational efficiency of CGRA architectures to deliver world-class machine learning performance.

Copyright © Wave Computing, Inc. All rights reserved.
www.wavecomp.ai

Conclusion

This paper has argued that Coarse Grain Reconfigurable Array architectures are more efficient than other reconfigurable and programmable parallel computing architectures for machine learning applications. We have shown that CGRAs are particularly suited to the acceleration of data flow computations such as those used in machine learning workflows (like TensorFlow). We have described a CGRA with partitioned arithmetic co-processing units, distributed memories and high performance instruction-driven switches. We have described the WaveFlow Software development environment that enables offline compilation of CGRA agents which are assembled into tensor-based data flow graphs at runtime using a session manager. We have also shown how the session manager uses distributed FIFO buffers to balance the flow of tensors in the graph and thereby maximizes the utilization of the CGRA processing elements when executing the graph. These factors combine to create an efficient, high throughput data flow computer for machine learning. Furthermore, this data flow computer is not tethered to CPUs as per the heterogeneous computing used by other systems. The Wave Computing data flow computer is therefore scalable to support larger training problems.

About the Author

Dr. Chris Nicol is the Chief Technology Officer of the Wave Computing, and the lead architect of Wave's DPU architecture incorporated into Wave's Deep Learning Computer. In addition to his duties as CTO, Chris leads the systems team developing products, the runtime software environment and benchmarking applications using deep learning frameworks.

Prior to Wave, Chris was Chief Technology Officer, Embedded Systems, at NICTA, an Australian ICT R&D organization he helped to establish, where he supported over 20 R&D projects across 5 labs in hardware and software development for intelligent, secure, robust and power efficient embedded systems. Prior to NICTA, he founded Bell Labs Research in Australia and Agere Systems in Australia and he worked for AT&T Bell Labs, New Jersey.

Chris received his PhD degree from the University of New South Wales and his MBA from the Australian Graduate School of Management. He is an inventor on 21 U.S. patents. He has served on the TPC of IEEE International Solid State Circuits Conference ISSCC twice, and is serving his second term on the TPC of IEEE International Symposium on Low Power Electronics and Design.

References

[1] http://www.computerworld.com.au/article/354261/what_will_do_100_cores_/

[2] https://en.wikipedia.org/wiki/NVLink

[3] http://www.ccixconsortium.com/

[4] M. Abadi, et-al.,"Tensorflow: Large –Scale Machine Learning on Heterogeneous Distributed Systems" at download.tensorflow.org/paper/whitepaper2015.pdf

[5] C. Ebeling, D.C. Cronquist, & P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath", Int. Workshop on Field-Programmable Logic and Applications, 1996, pp. 126-135

[6] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, "ADRES: An architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix", in Int. Conf. Field Programmable Logic and Applications, 2003, pp. 61-70

[7] G.Theodoridis, D. Soudris, and S.Vassiliadis, "A survey of Coarse-Grain Reconfigurable Architectures and Cad Tools" in S. Vassiliadis & D. Soudris (eds.), "Fine and Coarse Grain Reconfigurable Computing", Springer 2007, pp. 89-148

[8] R. Panda, A. Wood, N. McVicar, C. Ebeling, S. Hauck, "Extending Coarse-Grained Reconfigurable Arrays with Multi-Kernel Dataflow", Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL 2012).

[9] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, B. Baas, "A 5.8 pJ/Op 115 Billion Ops/sec, to 1.78 Trillion Ops/sec 32nm 1000-Processor Array", in Proc. Symposia on VLSI Technology and Circuits, June, 2016

Copyright © Wave Computing, Inc. All rights reserved.