

**Faculty of engineering - Shoubra**

**Benha University**

**Research Article / Research Project / Literature Review**

in fulfillment of the requirements of

|  |  |
| --- | --- |
| **Department** | **Electrical Engineering** |
| **Division** | **Computer Engineering** |
| **Academic Year** | **2019-2020** |
| **Course name** | **Computer Architecture** |
| **Course code** | **ECE322C** |

**Title: -**

**Domain specific architectures**

By:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Edu mail | B.N |
| 1 | **Khaled Mahmoud** | **khaled160314@feng.bu.edu.eg** | **343022** |

**Approved by:**

|  |  |
| --- | --- |
| Examiners committee | Signature |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Research objectives**

Refresh some information and law about pioneers of computer architecture, same some respect to compiler writers.

Define and have more details about what we mean by specific domain architectures and when should we be turning out interesting point to that DSAs.

Illustrate some examples with diagrams and knowledge of how it works to achieve what is the DSAs and what its applications.

**Abstract**

Now these days some dependable principals and law’s came to dead end with a damn whole package of questions now in out paper will have s brief of Moore’s law and why it ends in last decade and Dennard scaling problems and approaches.

IBM have problem with instruction set it was difficult have compiler bugs and different OS needs different architecture, so it unified the who different things in only one instruction set and bet on the whole company and that a door opening and closed, who could think it will open again.

Now compared to multicore processors and interrupts branches and multi-level caches have his own achievements and abnormal progress for general purpose processors, so in small compute-intensive potion we focus on small tasks not for whole domain but in a specific portion, not just make it small but make it done very well.

We will replace such caches with memories, and make instructions set more simple usage and the parallelism easier and let benchmark program take the rest and tell us how performance we achieved to what so far and it was so impressive.

**Table of contents**

|  |  |
| --- | --- |
| **Subject / section** | **Page** |
| **Introduction** | **6** |
| **Review** | **8** |
| **Guidelines for DSAs** | **9** |
| **TPU (tensor processing unit)** | **10** |
| **Result** | **14** |
| **Consolation** | **15** |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure I.D** | **Description** | **Page** |
| **1** | diagram for Tensor processing unite accept instruction form host CPU | **10** |
| **2** | Figure 2 control and data flow parallelism for systolic array | **12** |

**Introduction**

Moore does not only predict growth of transistors but also predict its demise years later. Now these days there are a slowing development for new processors and have many transistors per chip. Moore’s law does a revulsion to make transistor more efficient, higher performance.

The resources for a five-stage pipeline, 32-bit RISC processor, which need specific number of transistors later grew by factor of 100,000 to enable features that accelerated general purpose code on general processors. ROM was faster than RAM so control for instruction set wat ROM but with Moore’s law they were the same they have same semiconductor and RAM is more flexible and writable control store. Instead on running stand instruction set you can modify it to be suitable for your program.

Architectures targeted million-line programs written in efficient language like C++. Architects treat code such as black box, without understand internal structure or what even that code try to do. Benchmark programs were just modified and accelerated.

Compiler writers are hardware-software interface, they have limited understanding of high-level application behavior. Dennard scaling ended much earlier than Moore’s law, because he wanted lower energy per operation but more power performance per second in small transistor, but power budget is the key constrains and the translation from power to energy depend on how efficient we use transistors and it was not that efficient.

Existing of core get us 10% of improvements, improvements of number of arithmetic operations per second, but if we want order of magnitude improvements and programmability, we need to increase operation per instruction from one to hundred.

And for achieve that level of efficiency we need a root of change in architecture from general purpose core or general processors to domain-specific architectures (DSAs).

We are moving from CISC to RICS, we get caches to hold frequent used data, and what in RAM depend on program, keep the ISA very simple no need for interpreters, with Moore’s law could 32-bit data-path and small cache in one chip, RICS is 4X faster than CISC. VLIW have much long code design to make it fast and easy to execute, epic, ITANIANM.

**Literature Review**

Domain specific processors does only a narrow range of tasks, but they do them extremely well then computer will be more heterogenous than the homogeneous multicore chips of the past. In terms of energy usage so resources from Moore’s law could be reused, Caches are perfect and most beautiful innovation for general processors but in miss state it takes energy time to access memory and table, it could be not necessary for DSAs; for applications with predictable memory access patterns or huge data sets multilevel caches will be overkill.

DSAs is not created for large program like compliers, but for small intensive kernels of large system, face recognition, autocorrect, speech understanding. DSAs focus on subset not the entire program. Changing code or benchmarks now valid not breaking any rules to speed up DSAs.

Example from the past **John Cocker** and IBM take the mainframe IBM instruction set and use it but not the whole but the important one like load/store register-resister to build his ECL server. The performance went ×3 faster by using only a subset.

Vax 11/180 CPI = 10, 20% of VAX ISA => 60% is not used just microcode, so all bugs were in microcode not the complicated made things. Challenges for domain-specific architects is to find area justify demand of silicon on an **SOC** or a custom chip. But for **NRE** it is not economic or may be on FPGAs arrays have lower NRE and could be reused and reconfigurable hardware to amortize its const. the gain of FPGAs is quite better, unlike the less efficiency of hardware.

The software challenge it to find language that good for programmer and good for execution, it like how to make python run like C will be awesome.

***Guidelines for DSAs***

There guidelines not only leads to increase area and energy efficiency, the also lead to simple designs which reduce the cost of NRE of DSAs, and for user applications, accelerators that follow these principle are a better match to the 99th percentile response-time deadlines than traditional optimizations of traditional processors.

Use dedicated memories to minimize the distance over which data is moved.

Caches had a great deal of area and energy to deliver data optimally to the program. The compiler writers and programmer if DSAs know their domain, so no need for hardware to try to move data for them, instead data movement is replaced with software-controlled memories that are dedicated to and tailored for specific function within domain.

Invest resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories.

Turning understanding of the execution of programs in these rower domains, these resources must spend on more processing unites or larger on-chip memory.

Use the easiest form of parallelism that matches the domain.

Target domain for DSAs inherent parallelism. We will take advantage of it and expose it to domain, like using SIMD in domain is easier for programmer and complier writer than MISD, if VLIW can express the instruction-level

parallelism for the domain, the design can be smaller and more energy-efficiency than out-of-order execution.

Reduce data size and type to the simplest needed for the domain.

Application are memory-bound, so increasing the effective memory bandwidth and on-chip memory by using narrower data types, which make you pack more arithmetic units into same chip area.

Use domain-specific programming language to port code to the DSA.

Assume that programmer will rewrite their code just for your hardware, architects switch their desire to DSAs, example like TensorFlow for DDNs, made porting for your applications more feasible, and as we mentioned before only compute-intensive portion of application needs to run in DSAs in some domains, which also simplify porting.

Tensor Processing Unit (TPU) is the first custom ASCI DSA for WSCs. Its domain is the interface phase of DDNs hardware part, and programmed by TensorFlow framework, which were designed for DDNs.

The heart of TPU is an 8-bit ALU Matrix Multiply unit and a large software-managed on-chip memory, 700 MHz clock rate, 92T operation/second, 24 Mi-B of an on-chip unified Buffer (activation memory), 8 Gi-B off-chip weight DRAM memory.

Google engineers have goal after people used speech recognition DDNs, to improve performance 10X over GPU to handle computation demands. They design a custom ASIC for interface. To reduce delay of deployment, the TPU was designed to be coprocessor on the PCIe I/O bus, connected to exist server, the host will send

instruction over the PCIe bus directly to TPU for it to execute, rather than having fetch TPU we put in close to float-point unit which fetch instruction from its memory.

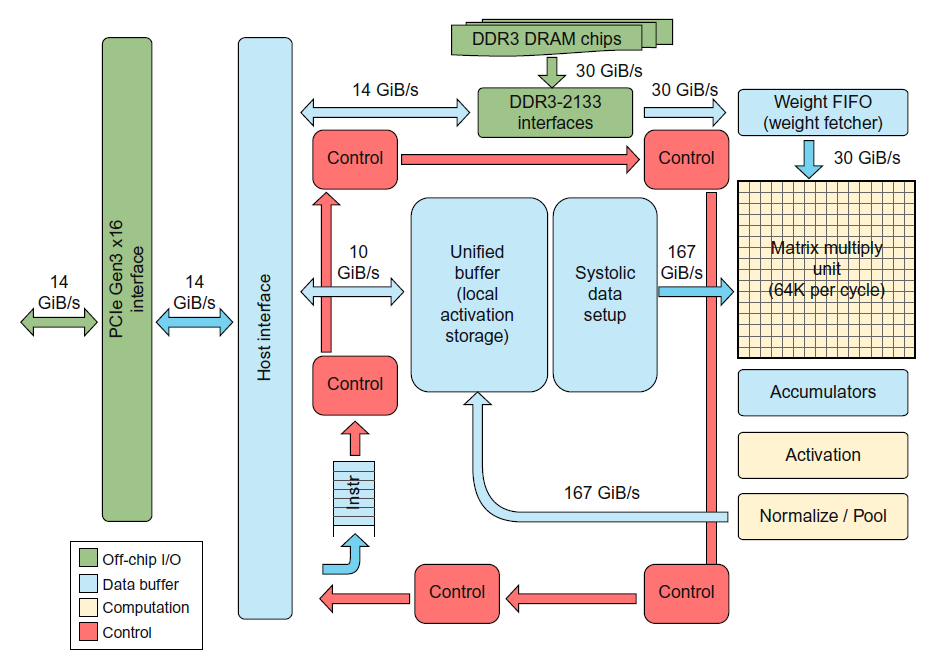


Figure diagram for Tensor processing unite accept instruction form host CPU

Figure above shows diagram of the TPU. The host CPU sends TPU instruction over PCIe bus into buffer (output and input for matrix), matrix multiply unit is the heat of TPU it contains 256 x 256 ALUs that can perform 8-bit multiply and adds on signed or unsigned integers. The 16-bit output collected in 4 Mi-B of 32-bit Accumulators, matrix unit could write and read 256 values per clock cycle and convolution or multiply. The non-linear functions are calculated by *Activation hardware.* Weight FIFO that reads from off-chip DRAM called weight memory (support many active models and it all about 8-Gi-B). the result in held in 24 Mi-B on-chip unified

buffer, which can be served as input to matrix combined with weight FIFO selecting model and put result into the next layer processing. DMA controller transfer data to or from CPU Host memory and the Unified Buffer.

TPU instructions follow the CISC tradition, include repeat field, it does not have program counter, and has no branch instructions, which sent from host CPU, have 10-20 CPI. It has dozens of instructions, but these are the key ones:

[1] Read\_Host\_Memory reads data from host CPU to unified buffer through 2048-bits paths.

[2] Read\_Weights reads weights from Weight Memory into the Weight FIFO as input to Matrix Unit,

[3] MatrixMultiply/Convolve cause the Matrix Multiply to perform operation, a vector-matrix multiply, or convolution form buffer to accumulator.

[4] Activate performs the nonlinear function of the artificial neuron, and its inputs are accumulators, and the outputs is unified buffer.

[5] Write\_Host\_Memory write data from unified buffer into CPU hos memory.

MatrixMultiply instruction is 12 bytes, 3 are Unified Buffer, 2 Accumulator, 4 are length, and the rest are opcode and flags.

The goal to run whole models is to reduce interactions with Host and achieve DNNs need. We desire to keep the Matrix Multiply Unit busy, hide execution of the other instructions by overlapping their execution with the matrix instructions.

Each have separate execution hardware (with read and write host memory combined into the same unit). We will increase parallelism further so we need input form buffer supported with weight form Weight FIFO, there is a case when no-ready signal for Matrix is will stall, then we can achieve more progress before addressing the result, so

making two dimension array called *systolic array* it is compute partial result from precomputed one, because the date flowed in advanced way it have interval where they are all combined, from array we have read once at input and write once at output causing more saving in power than accessing SRAM buffer.

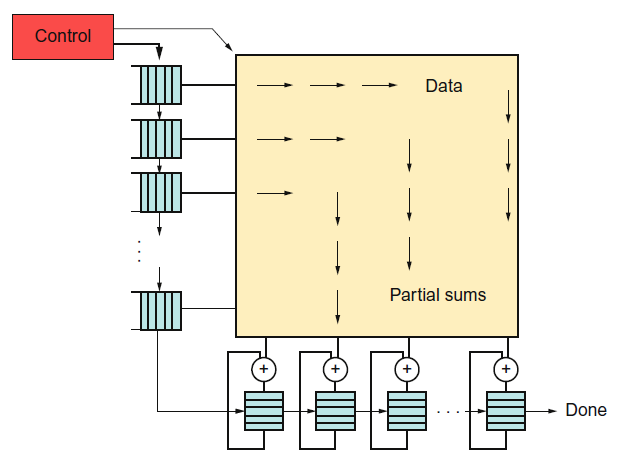
****

Figure control and data flow parallelism for systolic array

When array rotated then the data flows from the left into array and weight loaded from the top, the 256 elements move in diagonal front wave. Control and data are pipelined to virtually the 256-elements read at once, and after feed delay, they update one location of each of 256 accumulator memories. Software doesn’t worry about that array, but it does worry about latency of the unit.

**Results and discussion**

Use dedicated memories to minimize the distance over which data is moved.

The TPU has the 24 Mi-B Unified Buffer that holds the intermediate matrices and vectors of MLPs and LSTMs and the feature maps of CNNs. It is optimized for accesses of 256 bytes at a time.

Invest the resources saved from dropping advanced microarchitectural optimizations

into more arithmetic units or bigger memories.

The TPU offers 28 Mi-B of dedicated memory and 65,536 8-bit ALUs, which

means it has about 60% of the memory and 250 times as many ALUs as a

server-class CPU. Compared to a server-class GPU, the TPU has 3.5 times the on-chip memory and 25times as many ALUs.

Use the easiest form of parallelism that matches the domain.

The TPU delivers its performance via a two-dimensional SIMD parallelism

with its 256x256 Matrix Multiply Unit, which is internally pipelined with a

systolic organization, plus a simple overlapped execution pipeline of its

instructions.

Reduce data size and type to the simplest needed for the domain.

The TPU computes primarily on 8-bit integers, although it supports 16-bit

integers and accumulates in 32-bit integers. CPUs and GPUs also support

64-bit integers and 32-bit and 64-bit floating point.

Use a domain-specific programming language to port code to the DSA.

The TPU is programmed using the TensorFlow programming framework,

whereas GPUs rely on CUDA and OpenCL and CPUs must run virtually everything.

**Conclusions**

Having its own domain-specific architecture like DDNs or Tensor for the 2013 it have goal to update datacenter to double it with size to have that king of information which been collected from billions of people and doing its own business but when they found if people use speech recognition about 4 or 5 minute per day we will need to change the whole plane and made a specific hardware developed in only 15 month to achieve more performance than GPUs for fetching data computer operation and handle specific function on the specific domain so made a piece of custom ASIC interface.

Rather using and improve general processors the face coming to an important direction give us a very well performance over specific tasks and programed for only your hardware architecture like neural networks for machine learning and tensor for DDNs and other things more.

For specific domain we need specific language and made compile writer more relieved about their code it uses SIMD for simple and easy instruction set.

**References**

[1] Agarwal, A., 1987. Analysis of Cache Performance for Operating Systems and Multiprogramming(Ph.D. thesis). Tech. Rep. No. CSL-TR-87-332. Stanford University, PaloAlto, CA.

[2] Computer Architecture: a quantitative approach by john l. Hennessy, David a. Patterson

[3] Nielsen, M., 2016. Neural Networks and Deep Learning. http://neuralnetwork

sanddeeplearning.com/.