

No-ISA Is The Best ISA

Shreeyash Pandey, Rishik Ram

Vicharak, India @ vicharak.in

28th September, 2024

About us



Contents

- ① Chapter 1 - Motivations for our work
- ② Chapter 2 - Introduction to reconfigurable and heterogenous computing
- ③ Chapter 3 - Need for modern EDA compilers
- ④ Chapter 4 - Work Done Towards Implementation

Chapter 1 - Motivations for our work



Problems facing modern compute

Moores law is slowing down

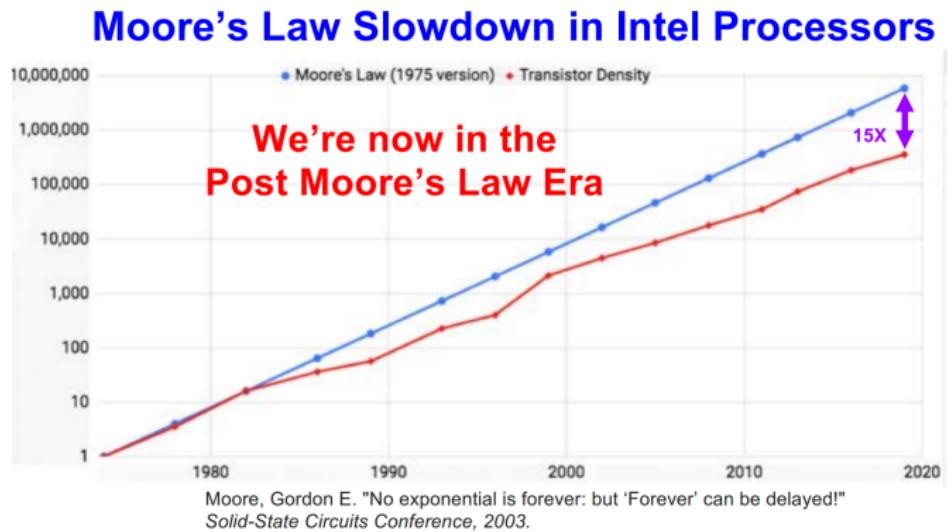
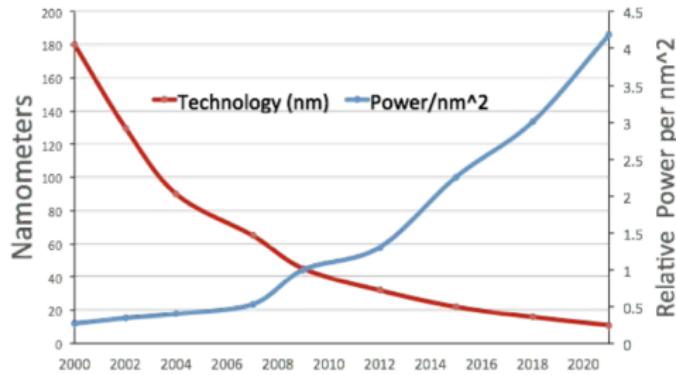


Figure: From "A Golden Age of Computers - David Patterson"

Problems facing modern compute

Dennard Scaling has stopped working

Technology & Power: Dennard Scaling



Energy scaling for fixed task is better, since more and faster transistors

Power consumption
based on models in
["Dark Silicon and the
End of Multicore
Scaling," Hadi
Esmaeilzadeh, ISCA,
2011](#)

14

Figure: From "A Golden Age of Computers - David Patterson"

Problems facing modern compute

- ① The free lunch afforded by hardware improvements over years is coming to an end
- ② Hardwares are designed first and complying softwares to support them after it.
- ③ New and creative architectures need to be designed along with the softwares abstractions to use them.

Overview of Modern Compute

- ① An average motherboard has a CPU and optionally a GPU
- ② In specialized domains, one may find ASICs being used (for eg, ML acceleration)
- ③ ASICs are pretty cool (and fast) and solve domain specific problems that CPU/GPUs may not be able to solve, but are they for everyone?
- ④ For starters, they are expensive to engineer and require a team of expert hardware engineers to be designed and fabricated
- ⑤ Once that's done, expert systems software engineers are required to make the ASIC usable/compatible with the existing operating systems.
- ⑥ A lot of hardwork, definitely not for everyone. As a result, ASICs are far and few
- ⑦ Should modern compute be restricted to CPUs/GPUs and a handful of ASICs?
- ⑧ What about the problems where none of existing compute suffices?

Hard-to-solve Problems for Modern Compute

Example 1

Problems involving many peripherals as well as compute

For example,

An embedded application that uses object detection to find objects in a line of sight and responds to it by driving many motors in real time needs heavy compute (for OD) and flexible IO to be able to drive all the motors reliably.

Existing solution would involve using a GPU for ML workload, and driving the motors from a CPU. A CPU may or may not have as many IOs as required, in which case an IO expander may have to be set up.

Hard-to-solve Problems for Modern Compute

Example 2

Unusual Representation of Numbers

Quantization is a technique of reducing precision of numbers at the loss of accuracy. Quantization is used extensively to speed up Neural Network inference. New techniques such as heterogenous quantization of layers (i.e. different bit-widths of numbers at layer granularity), odd-number quantization (such as 9-bit numbers), ternary computers etc. pose a significant challenge for existing fixed-bit-width computers.

Hard-to-solve Problems for Modern Compute

Example 2

New Architectures/Solutions for Old Problems

Kolmogorov-Arnold networks propose an alternative to MLPs (which is at the core of machine learning today). Quick and cheap inclusion of KANs into accelerators could help. TODO: write more

We can have our cake and eat it to

- ① Strengths of existing compute is known. We would like to have these strengths in our systems and bring reconfigurable heterogenous compute to tackle the weaknesses.
- ② We say, **It is possible to solve problems that are insufficiently solved by existing compute whilst keeping existing infrastructure (TODO: rephrase)**
- ③ The goal is to **assist** existing compute not **replace**.

Chapter 2 - Introduction to reconfigurable and heterogenous computing



Setting the stage

Two key ideas:

- ① Reconfiguration: The process through which a "reconfigurable processor" is re-programmed to implement a new circuit
- ② Heterogeneity: A system must include processors of different capacities/abilities well integrated together.

Reconfigurability: An Introduction to FPGAs

- ① FPGAs are a grid of cells that can be reprogrammed to implement any circuit.
- ② Digital circuits consist of gates (that implement logic) and connections (that connect gates to each other).
- ③ FPGAs, at the core, consist of SRAM cells (that implement the functionality of gates by storing their truth-tables in SRAM cells) and programmable interconnect (implemented via switch boxes) that allow connections
- ④ Circuits for FPGAs are described using Hardware Descriptions Languages (HDLs) such as Verilog, VHDL.
- ⑤ High level description of a circuit is compiled into real hardware (i.e. a representation that only uses FPGA primitives) by a "compiler"

Key problems with reconfigurable-heterogenous computing

- ① To implement a reconfigurable heterogenous computer with FPGAs, the problems are two-fold:
- ② Problem 1: Using FPGAs with traditional softwares are in-convenient.
- ③ Problem 2: Writing new hardwares for FPGAs, implementing custom solutions is tedious with a very steep learning curve, often times requiring domain expertise.

Problem 1: Programming model for FPGAs

- ① GPUs enjoy a concrete and abstract programming model
- ② No true industry grade programming model exists for FPGAs
- ③ There's OpenCL support for FPGAs. But that involves treating FPGAs like an ASIC.
- ④ A true programming model for FPGAs would heavily exploit reconfigurability

Comparison of a reconfigurable-heterogenous programming model with a von-neumann computer

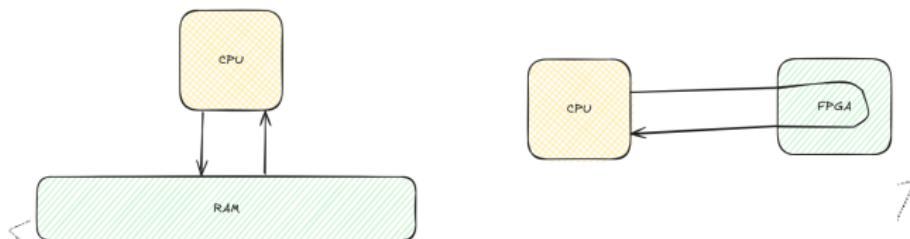


Figure: a) A von-neumann computer b) a flowing reconfigurable computer

Figure a) shows the difference b/w a von-neuman computer (which executes **instructions** on **data** over a **bus** resulting in back-and-forth of computation).

Figure b) is a flow computer where the hardware is configured to cause incoming data to be transformed in the way desired. There are **no instructions** as the hardware is configured to a desired operation. Data flows in and out of the chip transformed.

Flow architecture for JPEG encoding

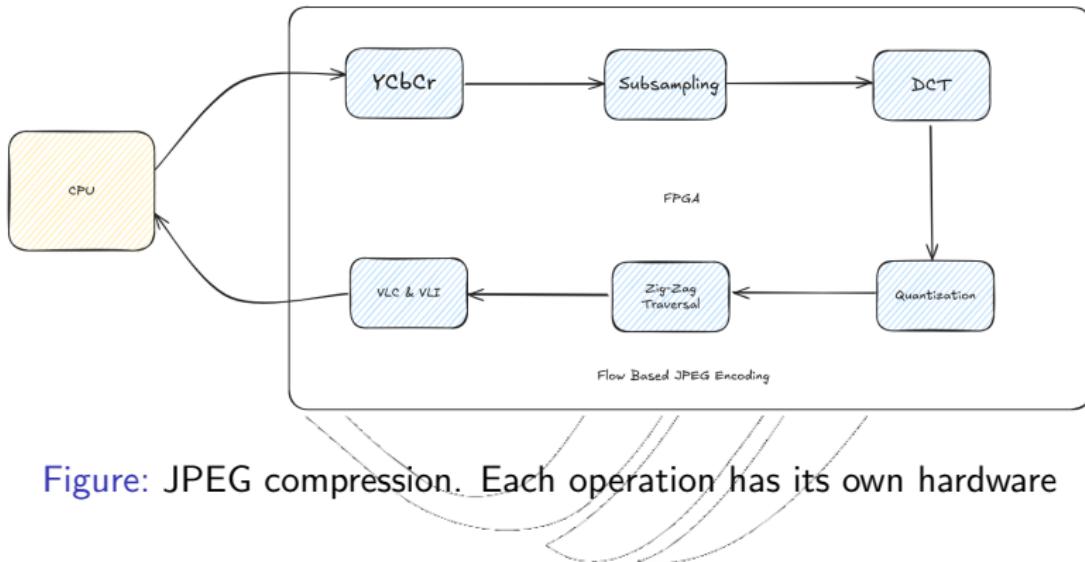


Figure: JPEG compression. Each operation has its own hardware

RAW images flow in, pass through the blocks, being encoded and the process and JPEG compressed images come out

Flow architecture for CNN inference

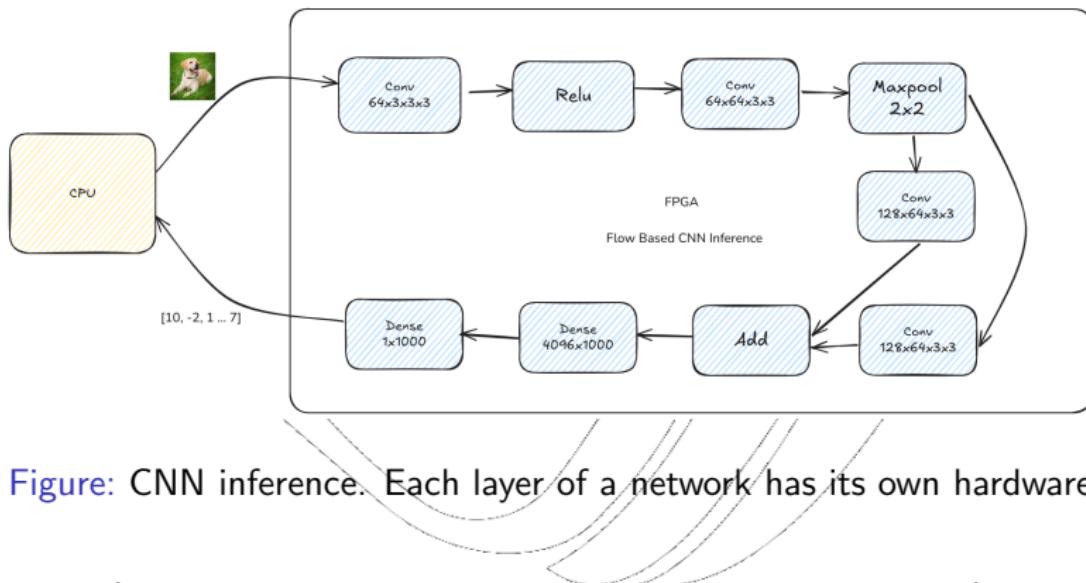


Figure: CNN inference. Each layer of a network has its own hardware

Images (according to the pre-process pipeline of a network) flow in, each layer manipulates and passes its computation to the hardware after it and end-results are returned by the last block.

Observation on flow-based computers

- ① Hardwares for a problem are generated by a "Compiler" from a high-level specification that describes connection of coarse functions.
- ② Any coarse hardware can be programmatically be plucked and placed in a different setting thanks to the compilers ability to reason with hardware connections.
- ③ Flow based computer exhibit a more functional approach towards computation
- ④ On a coarser scale, purity of computation is maintained as hardware blocks do not depend on a global state to execute

An exemplary DSL for reconfigurable compute architectures

Following is an example of a DSL that allows specification of coarse hardware. It provides an interface to define connections b/w hardware, control reconfiguration (through existing programming constructs (slide 3) and integrate it with existing codebases

An exemplary DSL for reconfigurable compute architectures

```
Base *input = new Tensor(std::vector<int>{1,2,3,4},  
                        "t1");  
Base *b = new MLEngineCore(input, "gc1");  
*b = input;  
Base *b1 = new PeripheralMultiplex(b, "AHB", "pp1");  
*b1 = b;  
for (int i = 0; i < 100; ++i) {  
    Base *b1 = new Sha256(b);  
    *b1 = b;  
}  
Model m1 = new Model(input, b1);
```

Describes an Mlaccelerator connected to a peripheral multiplexor which is connected to 100 Sha256 blocks, all through c++.

An exemplary DSL for reconfigurable compute architectures (2)

```
Base *cam_in = new CameraCore("MIPI0", "cam1");
Base *proc_one = new GaussianBlurCore(input, "gc1");
*proc_one = cam_in;
Base *display_out = new
    PeripheralMultiplex(b, "LVDS", "pp1");
*display_out = proc_one;
Model m2 = new Model(cam_in, display_out);
```

Describes an application that takes input from camera, implements gaussian blur on it and returns it to an LVDS interface on the FPGA.

An exemplary DSL for reconfigurable compute architectures (3)

```
m1->compute(input);
if (some_user_defined_condition(m1->output())) {
    m2->compute(m1->output());
} else {
    return m1->out();
}
```

model->compute is the function that triggers generation, flashing and computation on a hardware described by a Model.

Demonstrates conditional reconfiguration where based on m1->compute's result. If the result meets a user specified condition, m2's hardware is generated, flashed and computation begins for it.

Chapter 3 - Need for modern EDA compilers



Problem 2: Writing hardwares is hard

- ① Writing HDLs is a tedious task often requiring domain expertise
- ② EDA tools are proprietary and hard-to-work-with
- ③ The general problem of compilation of hardwares is NP-Complete but there are special cases that can be exploited.

Introduction to FPGA compilers

fpga toolflow infographic

Realizing this goal

- ① Realizing this goal requires designing and implementation from first principles
- ② To achieve this, we designed our own hardware: Vaaman
- ③ To understand the nature of applications (in the sense of what bottlenecks exist and whether or not a certain application would benefit from reconfigurable-heterogenous architecture), projects have been implemented
- ④ These include: Gati (an ML accelerator) and Periplex (a peripheral multiplexor)
- ⑤ Discussion on this work follows:

Optimization opportunity for EDA compilers

- ① Our DSL compiler connects hardwares together. The mapping phase of hardware generation can be completely by-passed if the compiler can be designed to operate on netlists directly instead of verilog.
- ② Mapping process involves among many steps a phase where it looks for a minimal boolean expression. In iterative write-compile-debug loops entire hardware may not change frequently so their resulting minimal boolean expressions can be cached and further sped up by performing a look up in this cache instead of searching all over again.
- ③ Routing can be re-designed to use GPUs (TODO: more)

Chapter 4 - Work Done Towards Implementation

Chapter 4 - Work Done Towards Implementation

The hardware (Vaaman)

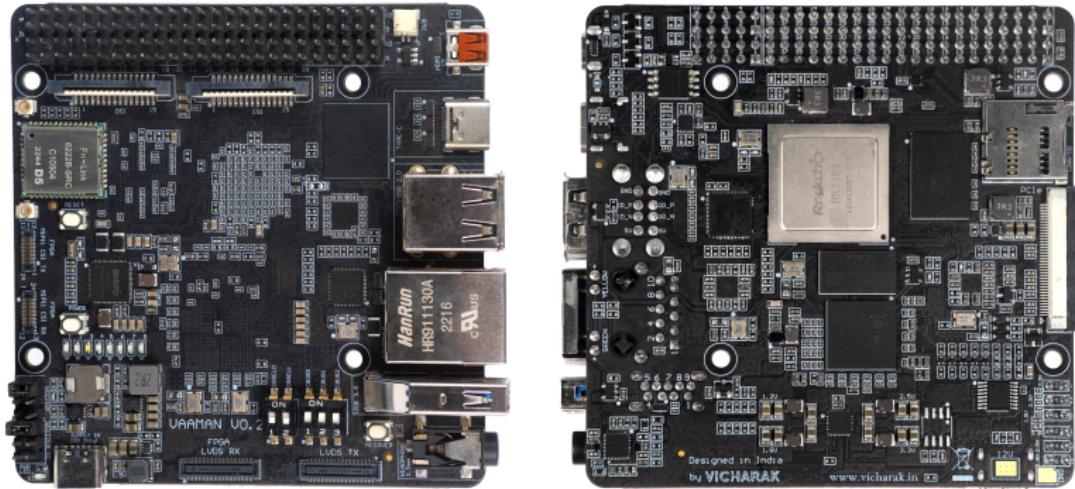


Figure: Vaaman: A heterogenous SBC

Implementation: ML Accelerator (Gati)

Gati is a set of hardware and software programs that perform CNN acceleration with FPGA as a co-processor.

- ① At the core of gati is a systolic array pipeline based MAC engine
- ② The Gati-ISA is macro (i.e. implements complex operations directly like Convolution) instead of breaking them down into primitives
- ③ The instructions have almost a one-to-one match with 'layers' from a neural network
- ④ Assisting this hardware is a Compiler/ Runtime.
- ⑤ The compiler does two primary things:
 - ① Parsing of input data and NN models (protobufs etc.), transpositions of kernels to allow contiguous memory access, and generation of a byte stream that can be fed to the FPGA
 - ② Generating custom hardware for every nn model
- ⑥ The runtime partitions a network into execute-on-host and execute-on-device, re-orders inputs, and offloads computation to the FPGA

Gati has an ISA? But you said ISAs are bad?

Gati is a testbed for modelling complex problems found in real world. At the moment it does and does not do many things that we eventually want from it. For example:

- ① Gati has a hardware generator. If this generator is generalized enough, we end up solving a part of problem 1.
- ② It still uses an ISA. But its possible to partition an ML model so that it can entirely fit into the FPGA hardwired to do only a part of the model followed by reconfiguration to execute later parts.

Implementation: Peripheral Multiplexor (Periplex)

- ➊ Periplex is a interface translator that allows communication between a set of inputs from a set of protocols to a set of outputs in another set of protocols
- ➋ Consider, an application where 2 inputs each from an SPI bus and an I2C bus need to drive 2 motors whose controller speaks CAN. Periplex can be used to easily make hardware and enable this communication.
- ➌ Periplex is, in a sense, the swiss army knife of embedded communication protocols.

Conclusion

- ① Reconfigurable architectures can provide a way to solve many problems that existing compute struggle with
- ② Reconfigurable architectures as demonstrated previously can help alleviate von-neumann bottleneck.