Published in Towards Data Science

Shashank Prasanna Follow
Aug 24 · 18 min read · Listen

AI Accelerators and Machine Learning Algorithms: Co-Design and Evolution

Efficient algorithms and methods in machine learning for AI accelerators — NVIDIA GPUs, Intel Habana Gaudi and AWS Training and Inferentia

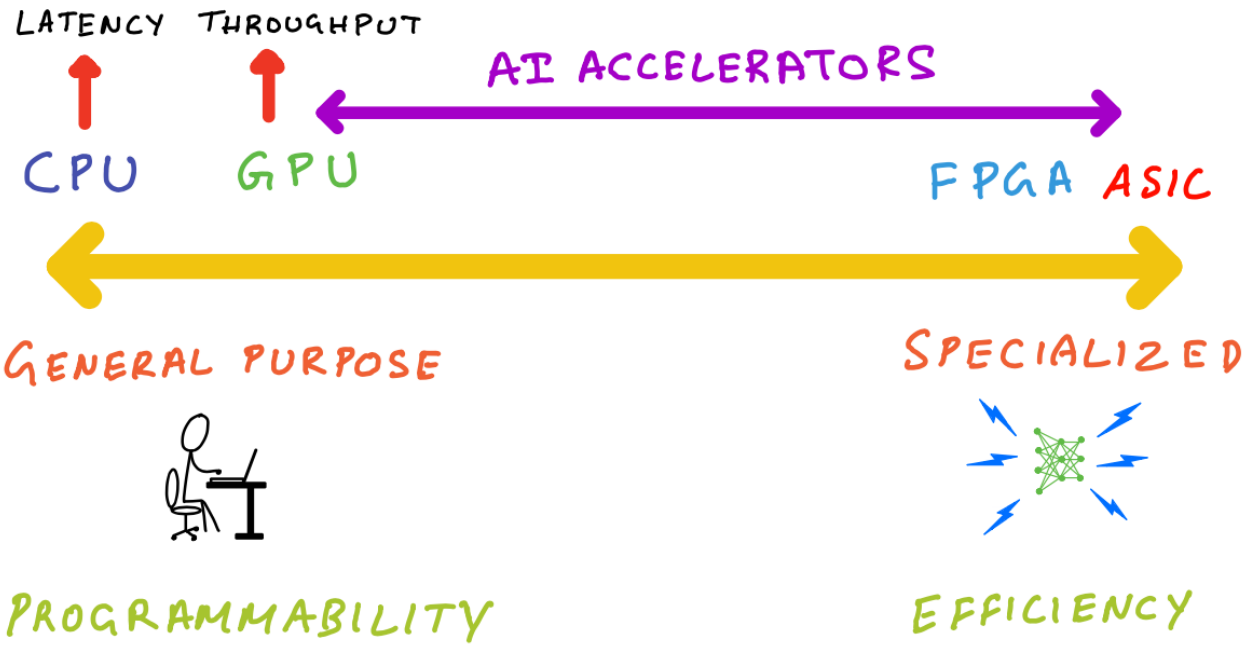


illustration by author

Chances are you're reading this blog post on a modern computer that's either on your lap or on the palm of your hand. This modern computer has a central processing unit (CPU) and other dedicated chips for specialized tasks such as graphics, audio

We've been pairing CPUs with a specialized processor since the early days of computing. The early 8-bit and 16-bit CPUs of the 70s were slow at performing floating-point calculation as they relied on software to emulate floating-point instructions. As applications like computer aided design (CAD), engineering simulations required fast floating-point operations, the CPU was paired with a math coprocessor to which the CPU could offload all floating point calculations to. The math coprocessor was a specialized processor designed to do a specific task much more quickly and efficiently than the CPU.

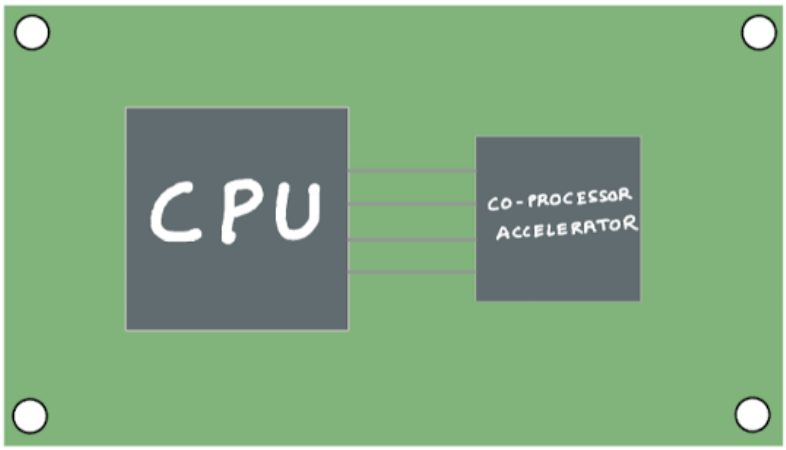


illustration by author

If you've been following recent advancements in AI or the semiconductor industry, you've probably heard of dedicated machine learning processors (AI accelerators). While the most popular accelerators are NVIDIA GPUs, there are others including Intel Habana Gaudi, Graphcore Bow, Google TPUs, AWS Trainium and Inferentia. Why are there so many AI accelerator options today? How are they different from CPUs? How are algorithms changing to support these hardware? How is hardware evolving to support the latest algorithms? Let's find out. In this blog post I'll cover:

- Why do we need specialized AI accelerators?
- Categorizing machine learning hardware — CPUs, GPUs, AI accelerators, FPGAs and ASICs
- "Hardware-aware" algorithms and "algorithms-aware" hardware
- Co-evolution of AI accelerators and efficient machine learning algorithms

- Future of AI accelerators

Why do we need specialized AI accelerators?

The three most important reasons for building dedicated processors for machine learning are **(1) energy efficiency (2) faster performance and (3) model size and complexity**. Recent trends to improve model accuracy, have been to introduce larger models with more parameters and train them on larger data sets. We're seeing this trend across all applications including computer vision, natural language and recommendation systems.

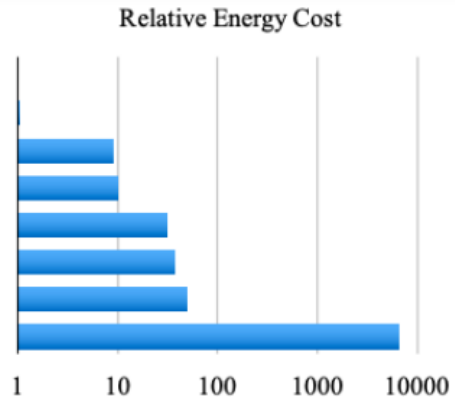
Language models such as GPT-3 have 175 billion parameters, which was considered extreme just a couple of years ago. Since then we've seen models like GLaM with 1.2 trillion parameters and NVIDIA's MT-NLG with 530 billion parameters. If history is any indication, model sizes will continue to get larger, and current processors won't be able to deliver the processing power needed to train or run inference on these models under tight time-to-train and inference latency requirements.

However, the single most important reason we need specialized AI accelerators and to make the economics of developing a custom chip work is the need for **energy efficiency.**

Why we need energy-efficient processors?

The larger the machine learning model the more memory access operations you need to perform. Computation operations like matrix-matrix and matrix-vector computations are far more energy efficient operations than memory access. If you look at how much energy is consumed by read access from memory vs. how much is consumed by addition and multiplication operations [source], the memory access operation is takes a couple of orders of magnitude more energy than computation operations. Since large networks do not fit in on-chip storage, there are a lot more energy expensive DRAM accesses.

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



source: https://arxiv.org/pdf/1506.02626v3.pdf

When applications like deep neural networks run on general purpose processors, even small improvements in performance by scaling come at huge energy consumption and equipment costs.

> *General purpose processors like CPUs trade-off energy efficiency for versatility and special purpose processors (AI accelerators) trade off versatility for energy efficiency.*

AI accelerators on the other hand can be designed with features to minimize memory access, offer larger on-chip cache and include dedicated hardware features to accelerate matrix-matrix computations. Since AI accelerators are purpose built devices it is "aware" of the algorithms that it runs on and its dedicated features will run it more efficiently than a general purpose processor.

Categorizing machine learning hardware — CPUs, GPUs, AI accelerators, FPGAs and ASICs

Let's now take a look at different types of processors and where they fall on a spectrum covering general purpose and specialized processors.

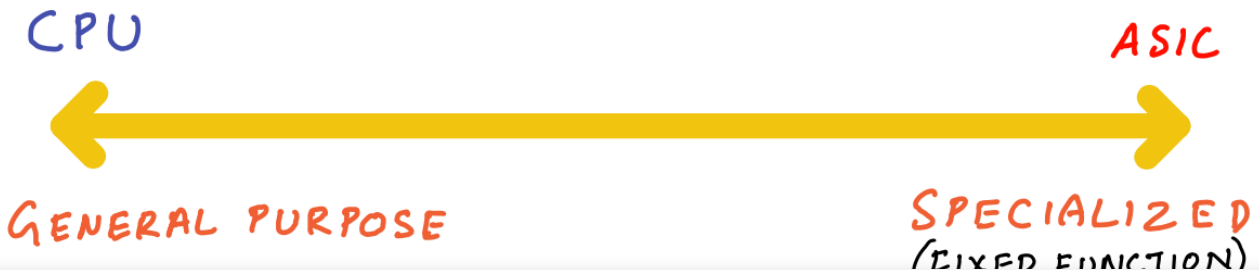


illustration by author

One the left end of the spectrum are CPUs, these are truly general purpose, as you can run arbitrary code on them. You can use them to perform any task that a dedicated chip does from graphics, audio processing, machine learning and others. As discussed earlier you'll be giving up on performance and energy efficiency.

On the other end of the spectrum are Application Specific Integrated Circuits (ASICs). These are also called fixed function silicon, because they exist to do just one or few things, and are not normally programmable and don't expose developer focused APIs. An example of this is the noise canceling processor in your headphones. It needs to be highly energy efficient to extend battery life of your headphones, and just powerful enough to reach the target latency so you don't experience delays or frustrating out-of-sync issues when using them to watch your favorite TV show.

The left end of the spectrum is about general purpose and programmability, and the right end of the spectrum is about special purpose and efficiency.

Where do processors like GPUs, FPGAs and AI accelerators fall on this spectrum?

Answer: Somewhere between these two extremes.

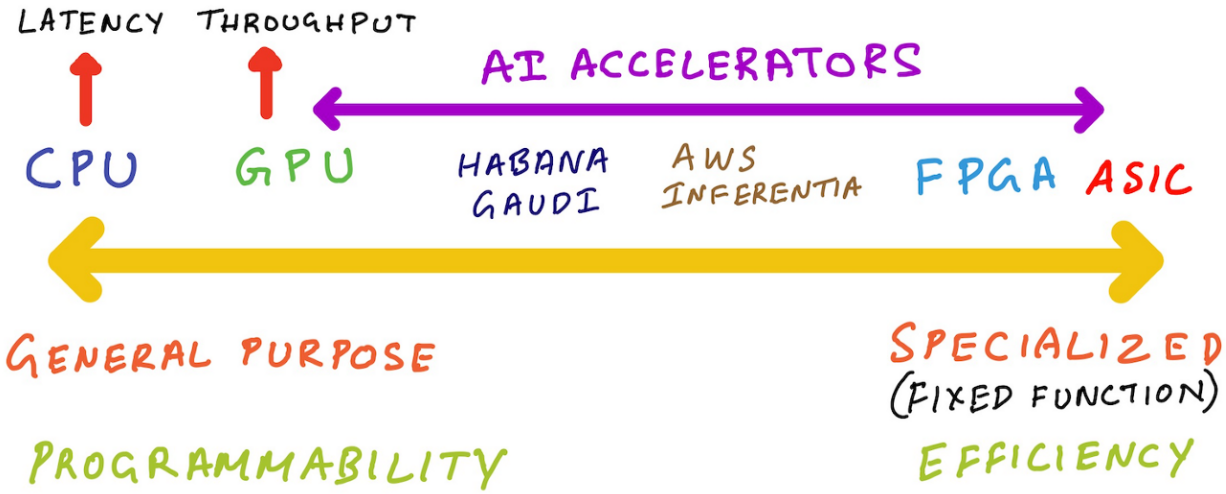


illustration by author

words, it's too low-level and close to the silicon that software developers won't have the skills and tools to program them.

To the right of the CPU end of the spectrum are GPUs. GPUs unlike CPUs are special purpose processors that excel at parallel workloads like graphics shader computations and matrix multiplications. CPUs are better suited for latency sensitivity applications, whereas GPUs are better suited for applications that need high-throughput. GPUs are like CPUs in the sense that they are programmable and using languages like NVIDIA CUDA and OpenCL these parallel processors can run a smaller set of arbitrary functions compared to a CPU, but really excel at code that exploits parallelism.

AI accelerators such as Intel Habana Gaudi, AWS Trainium and AWS Inferentia fall somewhere to the right of GPUs. Habana Gaudi offers programmability, but is less versatile than a GPU so we can put it closer to GPUs. AWS Inferentia is not programmable, but offers a list of supported operations it can accelerate, if your machine learning model doesn't support those operations then AWS Inferentia implements CPU fall back mode for those operations. So, we can put AWS inferentia further right on that spectrum.

"Hardware-aware" algorithms and "algorithms-aware" hardware

Now that we have a mental model for how to think about different types of processors, let's discuss how these processors interact with software that runs on them.

illustration by author

about where it's running. And when hardware features are designed, they're designed to support the widest range of software.

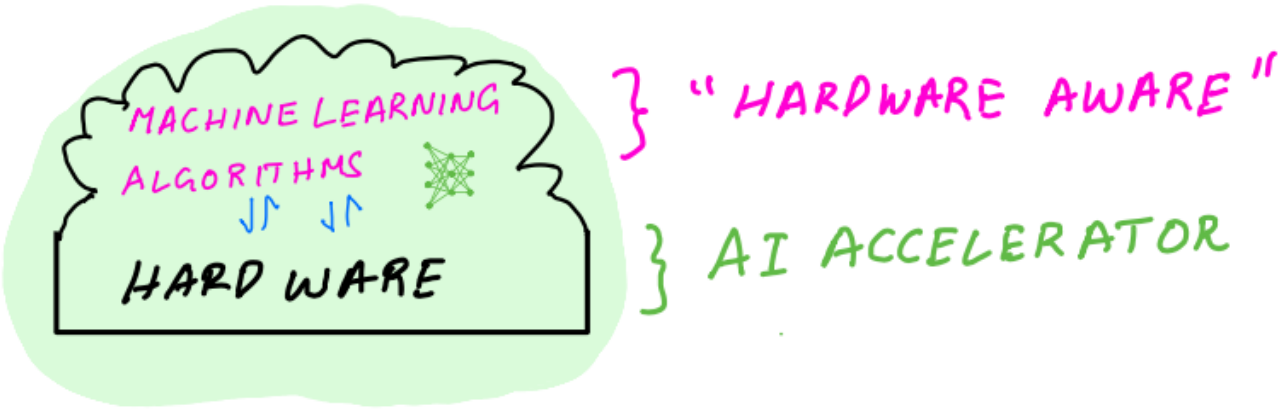


illustration by author

This model starts to evolve for applications like deep learning which demands higher performance and energy efficiency by bridging the gap between algorithm design and hardware design. The modern machine learning computing model also includes two components (1) machine learning algorithms and software frameworks (2) a general purpose processor paired with an AI accelerator.

Unlike in the general purpose computing model, researchers develop machine learning algorithms which are "hardware-aware" i.e. the code they write take advantage of hardware specific features such as multiple precision types (INT8, FP16, BF16, FP32), and target specific silicon features (mixed-precision, structured sparsity). As a user you can take advantage of these features via popular machine learning software frameworks. Similarly, hardware designers build AI accelerators that are "algorithms-aware" i.e. they design dedicated silicon features that that can accelerate machine learning matrix computations (e.g. Tensor Cores).

This "hardware-algorithms awareness" is possible because AI accelerator hardware and machine learning algorithms are co-evolving. Hardware designers are adding features to AI accelerators that are leveraged by machine learning algorithms, and machine learning researchers are creating new algorithms and approaches that can take

The co-evolution of AI accelerators and efficient machine learning algorithms

AI accelerators fall into two categories — **(1) AI accelerators for training (2) AI accelerators for inference**. Since the goals of training and inference are different and AI accelerators are dedicated processors for specialized workloads, it makes sense to design separate processors for each type of workload.

The goal of the training accelerator is to reduce time to train and training accelerators are designed with hardware features that training algorithms can take advantage of. This usually means training accelerators are higher powered processors, with more memory enabling higher throughput (data processed per second). Since training accelerators are throughput focused devices, its energy cost/throughput will be lower if you maximize throughput or utilization. Training accelerators also include support for mixed precision training using dedicated hardware features where computation is performed faster in lower-precision and results are accumulated in high-precision, making them even more energy efficient compared to general purpose processors. We'll discuss mixed-precision training in further detail in the AI accelerators for training section.

The goal of inference accelerators on the other hand is to reduce prediction latency for large numbers of independent data batches. An inference accelerator, therefore, needs to be energy efficient and have a low energy cost/prediction. You can use a training accelerator for inference (the forward pass of training is essentially an inference task), however the energy cost/inference on a training accelerator will be much higher since it will be under utilized for small batch inference requests.

A training accelerator is like a public transportation bus that's only energy efficient if the bus is full all the time so that the fuel-cost/passenger is lower. If you use the bus to transport only 1 person each trip the fuel-cost/passenger is higher. An inference accelerator on the other hand is like a sports coupe that's faster than a bus and more energy efficient than a bus for a single passenger (since less fuel-cost/1 passenger is lower). But if you want to transport 50 people in a sports coupe, it'll be highly inefficient and slower (not to mention illegal).

In the next section we'll take a closer look at training and inference workflows separately. For each we'll take a look at AI accelerators and software features that deliver the best performance and energy efficiency for their task.

AI accelerators and efficient algorithms for inference

We know that machine learning inference involves applying a trained model to new data to get model prediction results. In this section we'll discuss efficient algorithms that run on AI accelerators to make inference more performant and energy efficient.

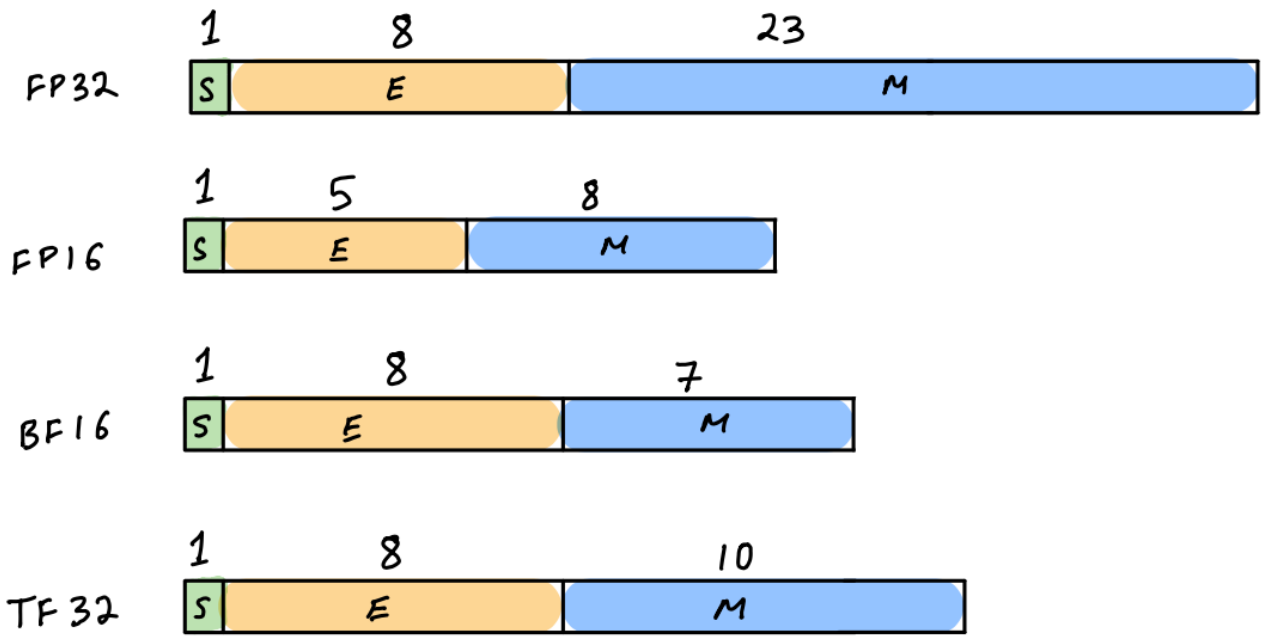


illustration by author

The most important hardware and software trick that can make machine learning

Machine learning algorithms commonly store and process numbers that are in single precision (FP32) based on the IEEE 754 standard. IEEE 754 specifies additional floating-point types such as half-precision (FP16) and double precision (FP64) typically supported in AI accelerators. Modern AI accelerators also support other non-IEEE 754 standard representations such as bfloat16 (introduced by Google Brain, supported by NVIDIA Ampere GPUs, AWS Inferential, AWS Tranium, Intel Habana Gaudi, Google TPUs) and TF32 (in NVIDIA Ampere architecture and AWS Tranium accelerators). Inference accelerators also typically support integer precisions such as INT8 and INT4.

Benefits of quantization for inference

For inference tasks, model weights and activations can be quantized, i.e. converted from FP32 (typical training precision) to lower-precision representation such as FP16, bfloat16 or INT8. Lower-precision models means better performance and energy efficiency. When you go from FP32 bit to FP16 operations, there is 2 times data size reduction and about four times reduction in energy use and about four times reduction in the silicon area for FP16 operations.

From an Inference-only hardware design perspective it makes sense to build smaller inference-only accelerators that only support lower precision and are therefore more energy efficient. Moving from FP32 to INT8 results will result in even lower energy utilization due to 4 times data size reduction.

This huge performance efficiency gain often comes at the cost of model prediction accuracy. Moving to lower-precision representation is a type of compression, which means you'll lose some data when you quantize. FP32 has a large dynamic range shown below, compared to FP16 and INT8. Therefore the goal of quantization for inference is to retain only the "signal" when quantizing to lower precision and throw out the "noise" and there are a few ways of doing this.

Quantization with NVIDIA GPUs

Newer generation of NVIDIA GPUs based on the NVIDIA Ampere and NVIDIA Turing architecture support a range of precision types just for inference. While FP16 precision type was first introduced in NVIDIA Pascal architecture way back in 2016, the latest

Choosing the right GPU for deep learning on AWS

How to choose the right Amazon EC2 GPU instance for deep learning training and inference — from best performance to the…

towardsdatascience.com

In this section I'll focus on both hardware and software support for quantization on GPUs.

Let's take the example of NVIDIA Ampere GPU architecture. On AWS you can access Ampere based NVIDIA A100 by launching the p4d Amazon EC2 instance and NVIDIA A10G by launching GPUs from the G5 instance type. Both these NVIDIA Ampere based GPUs support FP64, FP32, FP16, INT8, BF16, TF32 precision types and also include dedicated silicon for mixed-precision arithmetic that NVIDIA calls Tensor Cores. For inference, the precision types that we only care about are FP16 and INT8, and we'll revisit the other precision types when we discuss training in the next section.

Since most deep learning frameworks train models in FP32 on NVIDIA GPUs, to run inference more efficiently NVIDIA provides TensorRT, a compiler that can quantize model weights and activations to FP16 or INT8. To perform quantization, TensorRT will determine a scaling factor to map the FP32 tensor's dynamic range to FP16 or INT8's dynamic range. This can be particularly challenging for INT8 because its dynamic range is dramatically smaller than FP32's dynamic range. INT8 can only represent 256 different values whereas FP32 can represent approximately a whooping 4.2×10^9 values!

The ML research and hardware community has come up with two approaches to take advantage of quantization without losing accuracy:

1. Post-training quantization (PTQ): You start with a trained model in FP32 and derive scaling factors to map FP32 -> INT8. TensorRT does this by measuring the distribution of activations for each layer and finding a scaling factor that minimizes information loss (KL divergence) between reference distribution and the quantized distribution.




Let's take a moment to appreciate this dynamic between hardware and algorithms. We can see that the hardware has evolved to offer more efficient silicon features like reduced precision, now algorithms have to evolve to take advantage of the silicon features!

For code examples on how quantization works with NVIDIA TensorRT on GPUs read my blog post:

A complete guide to AI accelerators for deep learning inference — GPUs, AWS Inferentia and Amazon…

Learn about CPUs, GPUs, AWS Inferentia, and Amazon Elastic Inference and how to choose the right AI accelerator for…

towardsdatascience.com

Quantization with AWS Inferentia

While NVIDIA GPUs were initially created to accelerate graphics and are evolving to become powerful AI accelerators, AWS Inferentia was created with the single purpose of accelerating machine learning inference. Each AWS Inferentia chip has 4 NeuronCores, and each NeuronCore is a systolic-array matrix-multiply engine with two stage memory hierarchy and a very large on-chip cache. It supports FP16, BF16 and INT8 data types and doesn't support higher precision formats because you don't need it for inference — it is after all a specialized processor. Just like NVIDIA's TensorRT compiler for GPUs, AWS Neuron SDK and Compiler that supports quantization and optimization for efficient inference.

At the time of this writing, even though AWS Inferentia hardware supports INT8, AWS Neuron compiler only supports FP16 and bfloat16 as quantization targets. A model that's trained in FP32 will automatically be converted to bfloat16 during the compilation process. If you quantize the weights manually from FP32 to FP16 before you use the AWS Neuron compiler, then it'll retain the FP16 precision for inference.

Compared to GPUs, AWS Inferentia accelerators are not programmable and therefore falls to the right of GPUs closer to ASICs in the hardware spectrum we discussed earlier. If the model has operations that are fully supported by AWS Inferentia, it can be

additional data movement between CPU and the accelerator reducing its performance and efficiency.

Training AI accelerators and efficient algorithms

Machine learning training involves optimizing parameters of a model to improve model's prediction accuracy using training data. In this section we'll discuss efficient algorithms that run on AI accelerators to make inference more performant and energy efficient.

Let's revisit precision, but this time for training workflows. As we discussed earlier during training weights and activations are stored in FP32 representation which is an IEEE 754 floating-point standard which predates deep learning. ML researchers chose this as the default floating-point representation because during training FP16 couldn't hold enough information and FP64 was too large and we don't actually need all that magnitude and precision. What they needed was something in-between but there was no such thing in hardware that was available at that time.

In other words, the existing hardware was not aware of algorithm needs or "algorithm-aware" as we discussed earlier.

If ML researchers had the ability to choose any floating-point representation that worked great for machine learning, then they would choose a representation that would look different from FP32 or use a combination of precision to improve performance and efficiency. This is precisely the direction AI accelerators took with mixed-precision training and it required co-design of hardware and algorithms to make it work.



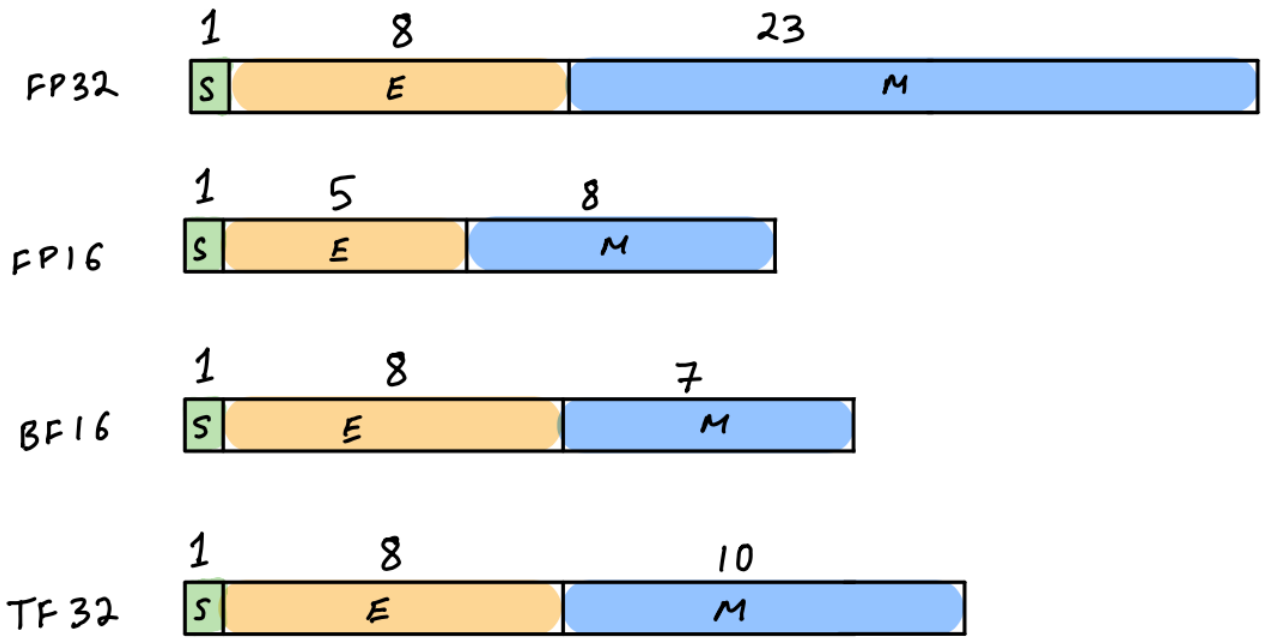



illustration by author

Mixed-precision training to improve performance and efficiency

Matrix multiplication operations are the bread and butter of neural network training and inference and an AI accelerator spends most of its time multiplying large matrices of input data and weights in various layers. The idea behind mixed-precision training is that matrix-multiplication during training happens in lower-precision representation (FP16, bfloat16, TF32) so that they're faster and more energy efficient and results are accumulated in FP32 to minimize information loss, making the training more efficient and faster.

Mixed-precision training with NVIDIA GPUs

In 2017 NVIDIA announced Volta architecture which featured dedicated machine learning specific features in its silicon called NVIDIA Tensor Cores. Tensor Cores enabled mixed-precision training using FP16 math with FP32 accumulation. Newer generations of NVIDIA GPU architectures have extended support beyond FP16 to other reduced precision formats (bfloat16, TF32). At the silicon level, Tensor Core performs reduced-precision fused multiply-add (FMA) with FP32 accumulation.

NVIDIA architecture improvements over each generation is a great example of

- NVIDIA Turing (2018) extended Tensor Cores to support INT8 and INT4 reduced-precisions (which primarily benefits inference, not training)
- NVIDIA Ampere (2020) extended Tensor Cores to support bfloat16, TF32, which means it can perform FP16, bfloat16 or TF32 math and accumulate in FP32 for mixed-precision

One of the challenges with mixed-precision training is that from a software point of view it didn't "just work". The user had to perform additional steps during training, such as casting weights to FP16 data types when possible, but also keeping a copy of the FP32 weights and loss scaling. While NVIDIA enabled deep learning frameworks to do these steps under the hood with minimal code changes, it's conceptually demanding from an end user and not as straightforward as training in FP32.

With NVIDIA Ampere the hardware has evolved to support the new TF32 format that significantly addresses this user experience drawback. What's exciting about TF32 is that it has the range of FP32 and the precision of FP16, which means deep learning frameworks can support it out of the box without needing to perform additional steps like casting and bookkeeping. While TF32 delivers improved performance over FP32 with no developer overhead, NVIDIA still recommends using FP16 or bfloat16 based mixed-precision training for the fastest training performance.

Mixed-precision training with other AI accelerators

Intel Habana Gaudi

Habana Gaudi accelerators support mixed-precision in a similar way that NVIDIA supports it — with the help of an additional tool that you can use with the deep learning framework that can perform casting and book-keeping for you. On AWS you can access Intel Habana Gaudi AI accelerators by launching Amazon EC2 DL1 instances which will give you access to 8 x Gaudi accelerators.

AWS Tranium

AWS announced AWS Tranium at re:invent 2021, an AI accelerator that was built by AWS's Annapurna labs. At the time of this writing AWS Tranium is not yet generally available. In this announcement talk AWS shared that AWS Tranium at launch will




We're living in exciting times for both machine learning algorithm research and machine learning hardware design. AI accelerators will continue to evolve to deliver higher-performance, better energy efficiency and become as seamless to use as general purpose processors.

Modern AI accelerators already include anticipatory hardware features such as support INT1 and INT4 precision types which aren't used for training or inference tasks today, but may someday enable new machine learning algorithms. Networking for AI accelerators is another important area that's seeing a transformation. As model sizes continue to grow we'll need larger computing clusters with many AI accelerators connected to each other to support larger workloads. NVIDIA offers high-bandwidth inter-GPU interconnect with NVLink and NVSwitch and Intel Habana Gaudi has integrated on-chip and Ethernet-based RoCE RDMA. AI accelerators are here to stay and will become a staple in modern computing environments as more and more applications integrate AI based solutions.

An area I expect to see more progress is user/developer experience. Today's heterogeneous computing model which involves working with multiple CPUs, AI accelerators and their networking and storage setups are very challenging for most data scientists and developers. While managed services in the cloud like Amazon SageMaker removes all the need to manage infrastructure making it easy to scale machine learning, open-source frameworks still expect the user to be knowledgeable about underlying hardware, precision types, compiler options, and networking primitives etc.

In the future, a developer will log into a remote IDE and run code using an open-source machine learning framework, and won't give a second thought to where and how it's running. They will only specify cost vs. speed trade-offs — faster results cost more, slower results saves cost — that will be their only cognitive burden. I'm an optimistic person and I think we're close.

Hello there! Thank you friend, for reading to the end!

If you found this article interesting, consider giving this an applause and following me on medium. Please also check out my other blog posts on medium or follow me on