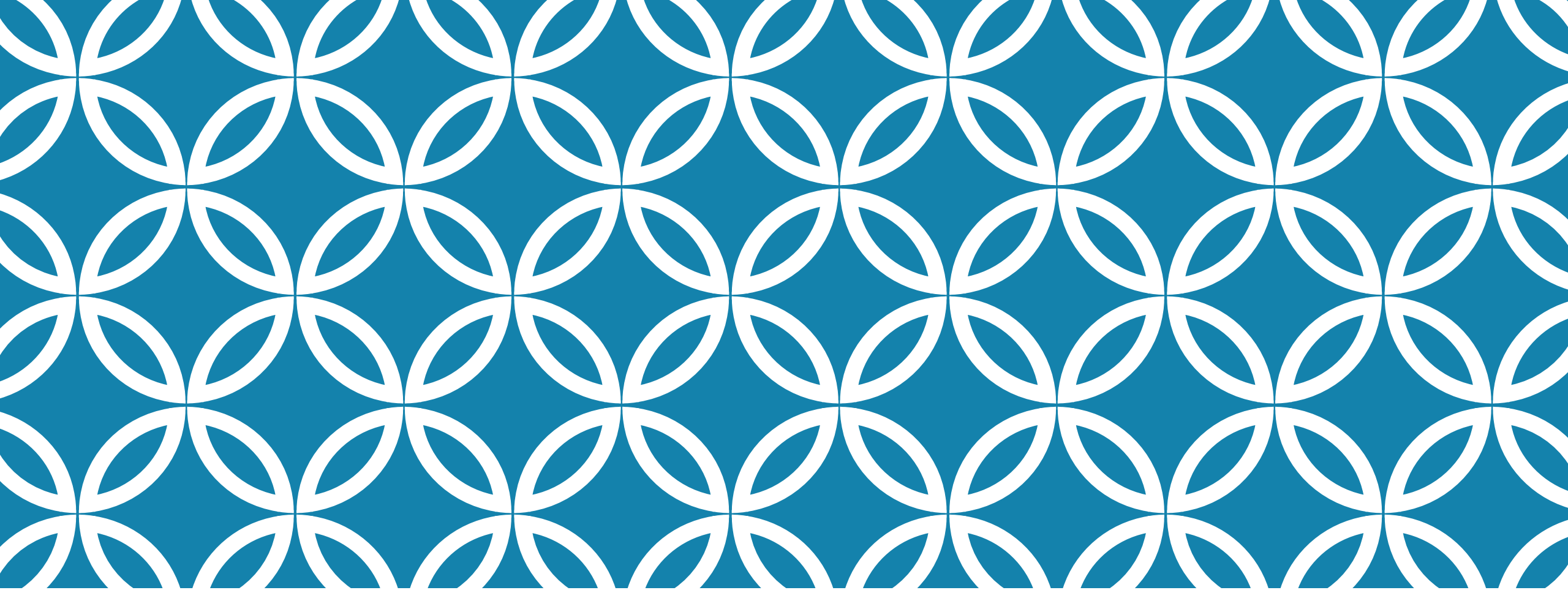


CS5412/LECTURE 26 HARDWARE ACCELERATORS

Ken Birman

CS5412 Fall 2022

TRADEOFFS

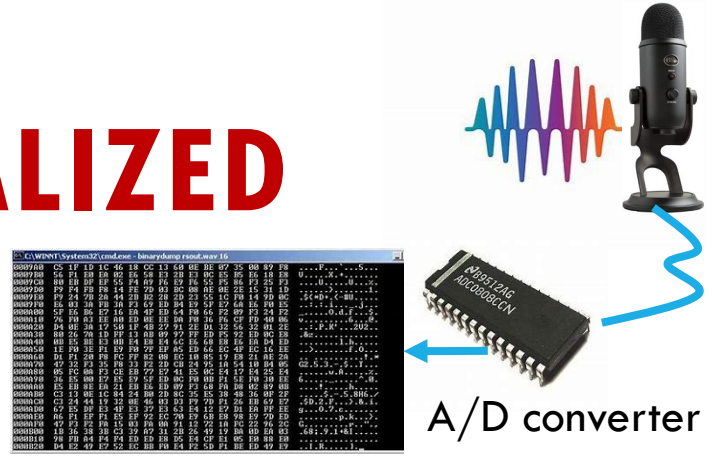
Amazon AWS server card

There has always been a tradeoff between *generality* and *efficiency*

A general purpose CPU has considerable advantages:

- Very cost-effective (high volume sales drive costs down)
- Highly performant (Moore's law, until ~2010. Multicore+hyperthreading since then), flexible (lots of languages, computing models), familiar.
- Virtualization (VMs and containers) easily support sharing, so cloud can pack jobs to keep machines busy.

BUT FOR CERTAIN TASKS, SPECIALIZED HARDWARE IS REALLY NEEDED



Basically, these are devices that can either do something in hardware that normal CPU instructions don't support (like direct operations on analog signals), or they can do parallel operations very efficiently.

The parallel computing opportunity is the most intriguing, today.

Someday, the analog dimension may get more attention.

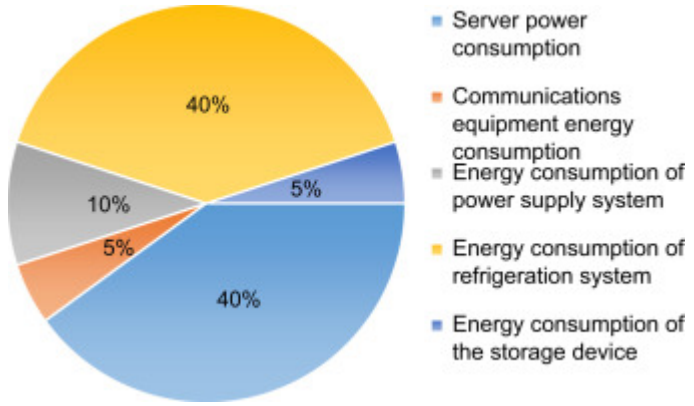
SUPPOSE OUR GOAL IS FAST, EFFICIENT CLOUD COMPUTING. HOW TO DO IT?

The general approach focuses on embarrassing parallelism at a very high level: we divide web page computations into a first-tier, and then run specialized μ-services that handle subtasks separately -- concurrently.

We also use batching to do lots of small tasks all at once. This can amortize overheads. It also introduces delays, but edge caching with weak consistency lets the cloud ride out those delays. Facebook goes even further with speculative edge updates

WHAT EXACTLY SHOULD WE OPTIMIZE?

Speed can often come at an energy cost! Cloud vendors are looking closely at their use of energy



An efficient cloud would fully utilize hardware but also minimize energy consumption. Those early steps were valuable and improved these metrics.

The early cloud was casual about energy, but…

- A lot of resources were "owned" but not fully used.
- Time and money and energy was being spent *waiting*.

TENSION: GENERALITY VS. EFFICIENCY

If we understand the workload deeply, we can often create extremely efficient specialized solutions, and could even create specialized chips that only include the exact hardware ideal for the task.

But because computing workloads evolve, the solution would only be ideal for a few years, at best. Then it would start to seem inflexible and inefficient!

Conversely, if we are overly general, we have this issue of copying data from place to place, and perhaps computing in less than ideal ways.

CAN WE HAVE IT ALL?

Modern datacenter hardware designers are asking:

- Can they create general purpose solutions in a normal way…
- … yet leverage specialized hardware where the benefits are large
- … in way that still can be upgraded periodically, or "repurposed"
- … and cut back on work done on the general purpose CPUs?

ACCELERATORS: THE SECRET TO CLOUD PERFORMANCE!

Each cloud vendor has begun to develop its own specialized accelerators

People who pretend the cloud is just a rent-a-server model lose access to the accelerators (the vendors all have security features that block you).

Accelerators have a huge impact. But you can't use them in the first tier, because they need a lot of management infrastructure. We find them in the μ-service layer, and in back-end batched computing frameworks.

HOW MUCH SPEEDUP CAN WE HOPE FOR?

This was a debated topic in the 1970's.

Some people imagined that there could be magic ways to speed computation up, and the people building the actual chips needed to find a way to limit these unrealistic expectations!

Eventually, Gene Amdahl found a way to explain the limits.

AMDAHL'S LAW

Consider a computational task. We can express the code in terms of actions that can occur in parallel, and actions that can only be done sequentially.

Measure the path-length of the sequential portion. This is performance-limiting for the whole computation!

If F is the fraction of a calculation that is sequential, and (1-F) is the fraction that can be parallelized, then the maximum speed-up that can be achieved by using P processors is 1/(F+(1-F)/P).

EXAMPLES

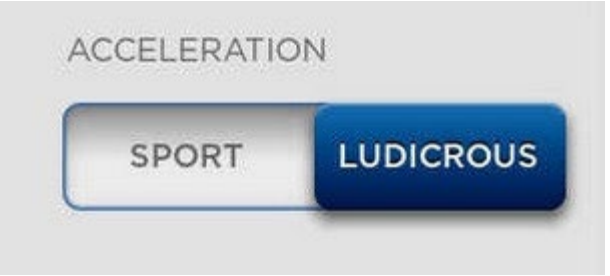
If 90% of a calculation can be parallelized then the maximum speed-up on 2 processors is 1/(0.1+(1-0.1)/2) or 1.8 (i.e. investing twice as much hardware speeds the calculation up by almost 2x)

… but with 10 processors, we only get a 5.2x speedup

… on 20 processors, our speedup is 6.9x: diminishing returns!

… on 1000 processors is 1/(0.1+(1-0.1)/1000) or 9.9x

HIGHWAY ANALOGY

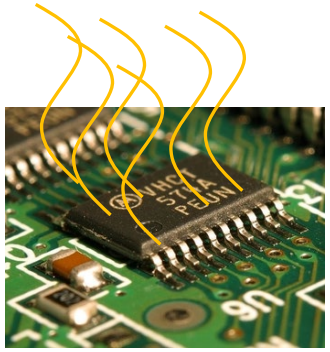


You buy a Tesla, take it out on California Route 101, and mash the "Ludicrous Acceleration" button.

It can instantly accelerate to the speed of light! But you won't get far…

Your commute will be limited by "stragglers".

THE OTHER LIMITING FACTOR: HEAT!



The clock rate might seem like a limiting factor, but a faster clock rate pumps more energy into the circuits and logic gates.

The heat dissipated will be proportional to the square of the clock rate.

In a parallel computing device, the whole surface might be active. So very fast clock rates make a chip run *very hot*.

BUT IF A DEVICE GETS *TOO* HOT…

Even a general purpose CPU is close to the heat-dissipation limits!

Operating systems like Linux run the clock as slowly as possible for less active computing elements, and even disable hardware components that are not currently in use. This helps.

But the clock rate on an accelerator might actually be lower than for a standard CPU! The (only) big win is parallelism.

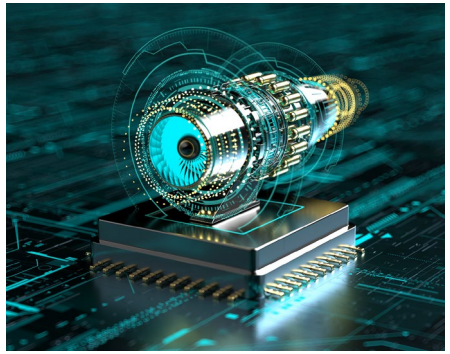
SO ACCELERATION OPTIONS ARE LIMITED TO HIGHLY PARALLEL TASKS OR "BUMP IN THE WIRE"

Hardware might be able to perform highly parallel steps rapidly.

We can also use hardware to reduce the work the host computer is doing.

And if host computers can't actually keep up with the network, we could perhaps wire the network directly to the hardware accelerator and if we're lucky, the device might keep up with the incoming data!

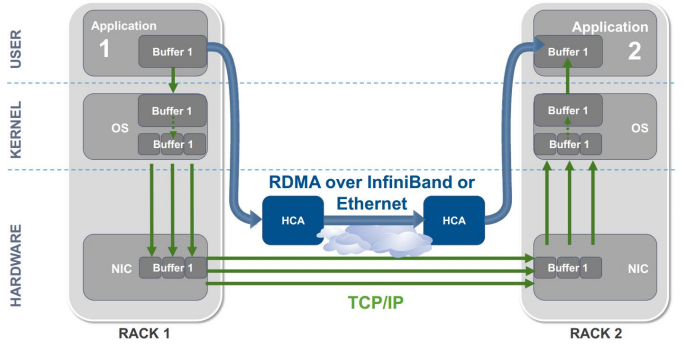
KEY CONCEPTS



One form of accelerator enables us to take methods (functions) that can be parallelized and create an optimized hardware version. You still call that function, but now it executes on the accelerator (often GPU or TPU)

A second form of accelerator focuses on applying the function to each message in a stream: the bump-in-the-wire model. Often implemented using FPGA hardware.

KEY CONCEPTS



Earlier in the semester we learned about RDMA network accelerators:

- RDMA runs a protocol similar to TCP network interface card, or NIC.
- The host is freed to use its CPU for other tasks.

But RDMA raises some security worries and is expensive to access in a cloud.

- This motivates interest in DPDK, which runs TCP in user-address space.
- DPDK bypasses the kernel, reducing delay. *No accelerator needed!*
- DPDK is slower than RDMA, but very secure. TCP is also "more standard"

ACRONYM CITY!

We are running into a staggering list of incomprehensible 4-letter terms.

You should memorize these to impress people.
But we wouldn't see them on exams!

Sort of a "survey of the options"

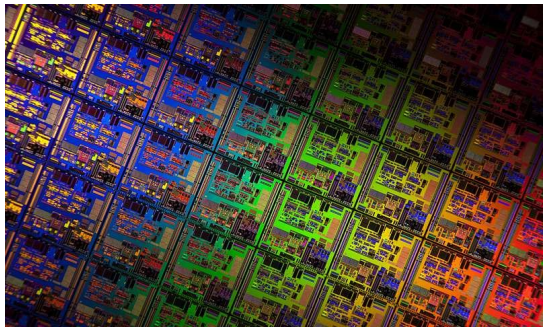
FIRST, STANDARD CPUS

As you know, prior to 2010 Moore's law was still "in control" and we had general purpose CPUs, with associated DRAM and caches, rotating disks.

Around 2010 rotating disks were displaced by flash memory drives. These are actually kind of slow, so they often have some DRAM as a buffer.

Simultaneously, chip designers invented branch prediction, data prefetching, speculative execution, hyperthreading, out-of-order exeuction

AFTER 2010 WE SAW NUMA



Today, a cloud computing data center server probably has 12 or more cores per CPU chip, with DRAM organized into clusters, perhaps 4 chunks of DRAM with 3 cores each. (More cores/server are likely in the future)

An on-board coherency protocol allows any core to access any memory, but the fastest data path is to the local DRAM.

Then with container virtualization, we can run lots of programs per server.

STORAGE DEVICES ARE IMPROVING TOO…

Disk I/O (even with flash SSD drives) often limits performance.

New "non-volatile memory" options like Intel's Optane NVMe are much faster. They use "phase change memory" technology. Today:

- NVMe is the new flash (somewhat expensive, but very fast)
- Flash is the new disk (slow, but cheaper and more capacity)
- Disk is the new tape (even slower, but massive capacity)

NETWORKS HAVE EVOLVED TOO

The Network Interface Card (NIC) on your server now has a small operating system in it, and runs programs in C! (Written by the vendor)

You can perform DMA transfers directly from machine to machine, not just from the network in and out of the machine as before. "Remote DMA" is like TCP (reliable, ordered, etc) but the hardware does all the work.

RDMA is way faster than TCP: we have RDMA at 200Gbps today, but the fastest TCP solutions are easily 4x or 6x slower.

RDMA FEATURES

With RDMA you can do some cool tricks

- Recall that with a NUMA machine, one core can access memory on any DRAM, so every machine shares the full memory pool.
- With RDMA, any core *in the data center* can potentially DMA transfer to memory anywhere else in the data center (but only if authorized).
- Moreover, RDMA allows direct access to variables or data structures hosted on a remote machine, too! (Again, only if authorized)

This is like having a normal computer, but a million times more memory…

GPU: A COMMON OPTION

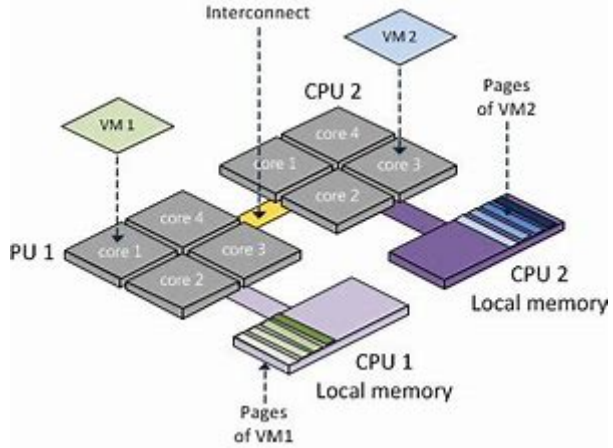
Titan GPU cluster

Desktop computers generally have a multicore general-purpose computing infrastructure, but servers have a GPU that the general machine controls.

The vendor creates a software library, so that a general purpose program can ask the GPU to perform a computation:

- DMA transfer to copy the data from general-purpose DRAM memory into the specialized GPU memory, which allows highly parallel access.
- GPU program executes to perform desired actions.
- Finally, the results are copied back to the general purpose host.

WHY "TRANSFER" THE DATA?



Normal host memory (DRAM) is in modules with cores clustered around them. The memory is "n-way" meaning it can handle n parallel requests

But a host would rarely have more than 20 cores, so n is generally < 20

The GPU has thousands of parallel compute cores. So even if we map DRAM where the GPU can "see it" (we often do!), GPU cache or GPU memory is often necessary to exploit the full potential of the accelerator

GPU PROGRAMMING IS HARD!

Example of CUDA code

```
//Vector size in elements
const int N = 1048576;
//Vector size in bytes
const int dataSize = N * sizeof(float);

//CPU memory allocation
float *h_A = (float *)malloc(dataSize);
float *h_B = (float *)malloc(dataSize);
float *h_C = (float *)malloc(dataSize);

//GPU memory allocation
float *d_A, *d_B, *d_C;
cudaMalloc((void **)&d_A, dataSize));
cudaMalloc((void **)&d_B, dataSize));
cudaMalloc((void **)&d_C, dataSize));

//Initialize h_A[], h_B[]…

//Copy input data to GPU for processing
cudaMemcpy(d_A, h_A, dataSize, cudaMemcpyHostToDevice) );
cudaMemcpy(d_B, h_B, dataSize, cudaMemcpyHostToDevice) );

//Run the core of N / 256 units, 256 streams each
//Assuming that N is multiple of 256
vectorAdd<<<N / 256, 256>>>(d_C, d_A, d_B);

//Read GPU results
cudaMemcpy(h_C, d_C, dataSize, cudaMemcpyDeviceToHost) );
```

There has been work on taking a general program coded in Java or C# and automatically finding "patterns" that can run on a GPU. Like a new compilation model in which the GPU offers special "instructions".

This gets to within 5x or 10x of hand-coded CUDA, but that isn't enough

OVERHEADS ALSO NEED TO BE CONSIDERED

Suppose we have a large matrix, 8k x 8k with doubles in it.

That will be 256MB, and if we have to move it into the GPU, then move the result back out into host memory, the data transfer time could be 2ms.

Even slower if the data is actually in a file somewhere.

… so it is important that we cache results in the GPU if we will reuse them, and be smart about multistep computations where we do a series of operations.

GPU OR HOST?

Another consideration is that the host itself has many cores, and each core might be hyperthreaded, and often they support MMX parallel instructions

As a result, the host itself is capable of doing some parallel computing!

If your computation is simple, like recoloring a photo, GPU kernel might not be worth the delay. But for matrix multiply or transpose, the GPU would be much faster – perhaps 50x faster or more!

EVEN USING PREBUILT GPU LIBRARIES IS AN ART

If our goal was just to recolor photos, it might be easier.

But graphics and vision algorithm often do very elaborate long sequences of matrix operations, and they may be designed with the specialized graphic display cards in mind (those cards can "see" data directly in memory, and can perform some operations on their own, like rescaling).

As a result, hard-core gaming or imaging companies hire specialists.

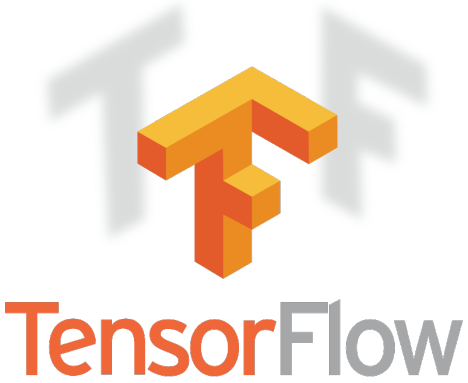
LIMITING ISSUES WITH GPU?

First, heat: on a GPU chip, we do a form of single-instruction, multiple data processing (like "multiply every pixel by this value"). Expends a lot of energy

But also copying: You also do a lot of copying from the general purpose host memory to the GPU memory, then back. (Hidden in GPU library, but costly).

The GPU has "extra logic" not really needed for machine learning. If we could just power those features down, we could reduce these costs.

"GPU FOR MACHINE LEARNING"?

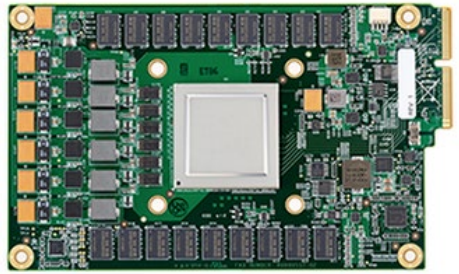


There has been more and more pressure to equip every computer in the cloud with a GPU, but this is very costly if those GPU units aren't all in use.

Still, many data centers take this approach.

Google is betting that GPUs just aren't cost-effective at scale and decided to strip the concept down to a minimum: TPUs == "Tensor Processing Units"

TPU IDEA



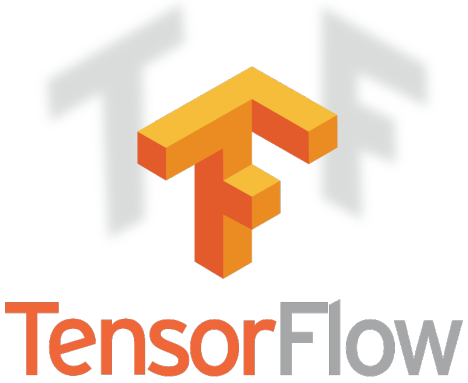
Google's first TPU unit

If the main demand for GPU is from machine learning, not full graphics code, we only need to support a subset of operations.

Google focuses on "tensor arithmetic" (a tensor is just a matrix, but with*d* dimensions rather than just 1 or 2).

Because less heat is produced by "unnessary circuitry" we can reduce energy costs, or even run the clock a bit faster.

TENSOR FLOW LANGUAGE?



Tensor flow is a version of Python extended to have a built-in concept of tensor objects and TPUs. Easy to express machine learning code this way.

The typical program is a kind of data-flow graph in which nodes compute and these tensors "flow" from input sources to outputs.

Mostly, tensor flow programs run on one NUMA machine, taking advantage of the attached TPU unit to accelerate the mathematical steps.

FIELD PROGRAMMABLE GATE ARRAYS (FPGA)

As you know, a CPU chip just maps instructions down to gate-level operations like AND, OR, XOR, NOT.

Xilinx invented a way to take a chip and "download" a wiring diagram and a logic diagram to it. So you can "configure" your chip to have, say, an ARM or i86 processor on it. Of course that would be silly.

But you can also design your own specialized chip, and in theory, it could do anything a GPU or TPU could do, or anything else, really.

FPGA PROGRAMMING IS HARD!

Chip designers use Verilog, and FPGA designers do too.

Normal chips are carefully debugged. If you try to create your own FPGA chip and it has bugs, you can cause the FPGA device to hang.

So FPGAs are often built up from libraries of logic blocks that are carefully tested, but this makes FPGA programming a specialized task.

EVEN SO…

FPGA turns out to be a very cost-effective option for some important cases seen in today's cloud!

Cryptography is one example: As a "bump in the wire", we can use FPGA chips to perform whatever cryptographic action is needed for various network security protocols (there are many).

You could create an "Application Specific Integrated Circuit" (ASIC) for each protocol, but an FPGA solution can be reconfigured as needed…

EVEN SO…

There is also very exciting work on mapping deep neural networks to clusters of FPGA chips.

This yields very rapid and cost-effect image classification solutions, or voice recognition ones. The future of vision and speech could easily depend on these FPGA clusters.

But these ideas depend on having a cloud full of data (the models used are massive, and there may be one model per "situation")

VISION: A SUPERCOMPUTER WITH MASSIVE NUMBERS OF GPUS OR FPGAS

Noctua supercomputer has millions of FPGAs but just 256 CPU nodes

The system is so dense that it requires high speed water cooling to pull excess heat away

Computes at 535 teraflops, one use is to control fusion reactor prototypes

MICROSOFT SOLUTION?

Equip every server in the data center with some GPU or FPGA accelerators, very close to the network card.

This way, one "host node" is really playing a dual role: general computing, but also care and feeding of one of the GPU or FPGA devices

The datacenter as a whole will now hold a massive accelerator array!

EACH CLOUD VENDOR HAS ITS OWN…

Microsoft has been creating FPGA arrays specifically for vision and voice recognition. (The NSA does this too… to hunt for cybercriminals/terrorists)

Google's biggest TPU deployments are revolutionizing their AI capabilities

Amazon is focused on robotics tasks. Facebook is emphasizing VR

Via hybrid cloud standards, some systems might even combine several accelerators in one use case!

NICS AND ROUTERS

We mentioned that NICs are able to do RDMA and run a kind of TCP on the card, in a dedicated processor.

But NICs and Routers are actually becoming programmable

Useful for many tasks: “smart routing” that looks into packets and directs packets based on content. Network virtualization. And there is even talk of running machine-learning tasks “directly” in the network itself.

AND THERE IS EVEN MORE!

I didn't even mention

- iSGX: Intel's hardware-based privacy and security model
- Bitcoin mining and BlockChain "proof of work" codes
- Special chips for performing tasks like FFTs and Sonar/Radar/Lidar
- Chips with analog components, or optical components
- Quantum computing accelerators …

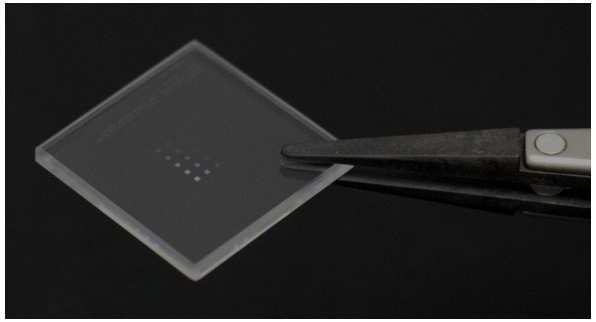
OPTICAL NETWORKS

Looking to the future, people are noticing that we can run multiple wavelengths on a typical strand of optical network fiber.

So why not have multiple side-by-side networks?

And if the application owns its own "wavelength", why not make that network specialized in various ways: a "software defined optical wavelength" just for the particular application!

HYPERSTORAGE

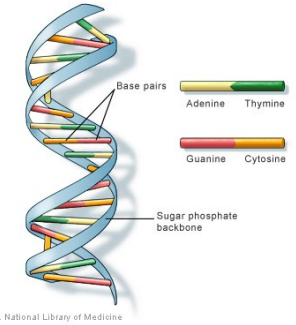


Microsoft Silica Zettabyte Storage Unit

Other cool ideas involve storage systems with insane capacity!

- A single cube of computing-quality silicon can "record" 7-bit "numbers" using a tricky laser "zap" system (in a "write once" model)
- Then the data will persist for as long as 100,000 years!
- One silica chip of this kind could hold all the movies ever made, plus the whole library of Congress, and would only be ~1/3 full.

There is another massive scale concept that uses DNA as a medium. DNA memory can potentially be read-write



DISAGGREGATION

Some people believe that eventually, data centers will evolve into what they call a “disaggregated” model.

In this we will have racks of identical components: one kind of rack specialized for CPU, another for memory, for storage, for TPU / GPU, etc.

Then the network wiring will evolve to let us “assemble” the ideal virtual processor on the fly, with exactly the hardware for the specific use case.

SO WHAT'S THE PROBLEM?

The cloud is becoming massively complex and specialized! And yet if you ignore all this stuff, your performance would be very poor.

The only possible answer is to learn to use the vendor-provided μ-services, because those take full advantage of this special hardware.

When you use their services and don't try to roll-your-own, you get cost-effective and scalable performance (and don't need to learn Verilog).

VENDORS HAVE A PROBLEM TOO: LINUX ISN'T UP TO THE NEW ROLE

You can bolt stuff to a Model T, but it still is a Model T underneath!

Employees at cloud companies *want* to write programs in a normal way!

(Even Microsoft has become a mostly Linux company.)

But Linux wasn't designed for specialized accelerators and cross-computer memory access and computation occurring at every level of every device…

LINUX ISSUES…

Its basic model is of a program with memory…

- But for most of this hardware the program actually controls some sort of device "outside" the computer, with its own memory, and maybe its own network connection.
- Even copying just once might be 10x or 100x slower than what the device could have done directly on the wire
- We also lack a security model for this kind of distributed sharing, and we don't understand how stable these new technologies will be

TODAY'S SITUATION?

As the owner of an infrastructure, a company like Google, Microsoft or Amazon can build μ-services that leverage all sorts of specialized hardware to accelerate important tasks.

But as a developer, you can only benefit if you use theirμ-service, not by trying to leverage these devices "directly".

Why? Because they worry about the risk of instabilities (very real).

TECH TALES FROM THE CRYPT

"They that live by the sword shall die by it…"

- *Bob forgot to verify his Verilog. The whole datacenter crashed.*
- *The new search engine… turned out to be racist*
- *AI controlled car confused by road-repair lane paint*
- *Attack of the zombie refrigerators*

Self-driving car suddenly turned left on freeway

BUT WHY ARE THOSE RELEVANT?

Massive, highly specialized accelerator solutions can be extremely hard to debug, and fixing issues may be even harder

As we grow complacent about the speed of machine intelligence, we depend more and more on these technologies.

Even success can be a cause of technology collapse – not everything scales!

SUMMARY?

Specialized hardware is a key to cost-effective modern cloud computing. Yet these innovations are also creating new "risks".

Accelerators will be even more critical if the future IoT edge needs a lot of support for very rapid computer vision, speech recognition, and intelligence. But these devices will need to live near the edge.

Accelerators can only be used in a μ-service model. The main exception is that end-users do have ways to access TPU and GPU accelerators from their code, via libraries of numerical methods.