Offload now, or your CPU dies!

Tony King-Smith

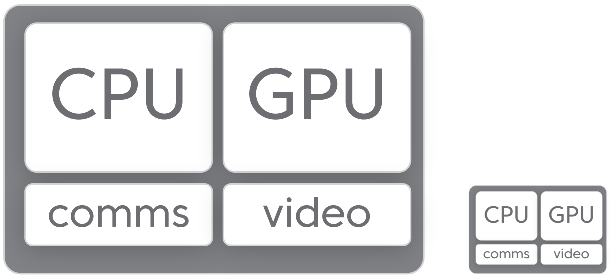
AI is increasingly being recognized as one of the most fundamental transformations of computing since the invention of the CPU. Since most algorithms start as sequential software code on CPUs, later accelerated by GPUs for some NN workloads, many believe that the next hardware step is to add NN accelerators to the CPU+GPU clusters. However, that could strangle the performance of the SoC, and might not deliver anything near the performance or efficiency expected There are other ways, explains Tony King-Smith, Executive Advisor for aiWare.

Keywords: host offload, neural network acceleration, aiWare hardware IP core, SOC, NOC, memory

Introduction: the rise of SoCs

The rise of the smartphone saw many tremendous strides forward in technology, especially computing hardware. The stringent demands for high performance at low power consumption and thermal envelope led to a step function improvement in our understanding of how to design high performance systems on chip (SoC) that could deliver desktop-class performance for a power budget of just a few Watts.

Figure 1: Mobile SOCs have shrunken in size while providing enourmous performance gains.



SoC	iPhone 3GS	iPhone XS
Date Launched	June 2009	September 2018
CPU	Single core Arm 32-bit @ 600MHz	6-core Arm 64-bit @ 2.5GHz
GPU	PowerVR SGX535 1.6GFLOPs	Custom 4-core 1 TFLOPs (est)
Display	640x480	2436x1125
Video	640x480@30fps	3840x2160@60fps

As SoCs for mobile developed, so too did the complexity of functions on-chip. Growing from a relatively modest 16-bit MCU in the early days of 2G mobiles, the mobile SoC grew to become one of the most complex compute platforms ever created. It contains everything on a single chip: software execution (CPU + caches), graphics acceleration (GPU), communications (Wi-Fi, 3G/4G modem, Bluetooth), video compression and decompression, camera pipeline (ISP), as well as a host of low-speed peripherals for everything from the display controller and touchscreen to sensor interfaces, power management, thermal management and more. Today's mobile SOC processors, featuring 64-bit CPU clusters of 6 or more CPUs, plus GPUs with hundreds or thousands of ALU cores, deliver more compute power than desktop computers 3–4 years earlier.

Given that we've now mastered the design of such advanced SoCs, surely the right thing to do is just keep doing more of what we've been doing, and add in the NN accelerator? Well for 100 GOPS or so, that's probably a great idea. However, with automotive multi-sensor systems demanding 50–100 TOPs and more processing power, that's a much tougher task. Just cranking up the clock speed and performance is easier said than done, as everything is already working hard. When NN accelerators starts demanding tens of GBytes/s or more of additional bandwidth, sharing the central memory resources of the CPUs and GPUs, it gets tough.

On-chip bandwidth is complicated

SoC performance means nothing if each processor cannot get the data it needs to perform each and every task when it needs it. That's why today's mobile SoCs have advanced on-chip memory caches and sophisticated high-bandwidth networks-on-chip (NOCs) to connect the various processors and accelerators to memory, peripherals and each other.

However, today's SoCs have many different functional blocks all connected to the NOC (or often network of NOCs). So, for every data movement, decisions must be made as to who gets priority. That ends up being a pretty complex task and gets harder as the clock speed goes up. Every cycle lost to prioritization decisions amounts to lost bandwidth. And every cycle where one block is waiting to get access to the NOC is efficiency lost.

As more advanced silicon processes are developed, one major dividend is that we are able to put significantly more memory on-chip, usually SRAM (static RAM). The closer memory is to the function block needing it, the faster it is, and the less power it takes to read or write data. SRAM is very easy to use, as you can read or write any unit of data randomly at any time, and it is extremely fast. That's why it is used in many applications, from on-chip cache memories to FIFOs and buffers, through to tightly-coupled local memory.

However, because there is more and more memory on-chip, there are more situations where data has to be moved between these blocks of SRAM. This creates a significant demand on the SOC's NOC too. The more memory we put on-chip, the more extreme the demands on the overall chip architecture become.

One of the greatest challenges facing any high performance SoC designer is how to manage and prioritize the hugely complex data flow, under a wide range of workloads and operating conditions.

External memory demands also high

Designers of CPUs, GPUs or any other accelerator or functionblock in an SoC try to keep as much data as possible on-chip. Indeed, the more data can be kept within the block itself, the less it needs to request data transfers from the NOC. However, on-chip memory is expensive – the largest cache for a CPU core is rarely more than 1MByte.

When processing GBytes of data coming in from sensors, and applications often demanding GBytes of working memory, it isn't long before the SoC needs to talk to external memory.

External memory is usually based on DRAM (dynamic RAM), which is 5–6 times denser than SRAM. However, to benefit from this greatly improved density, you need to access the memories by trying to request blocks of data, not just individual bytes. This contrasts with on-chip SRAM, where it is no problem to reward any location you wish at random. This need to access external DRAM in a structured way leads to a whole new level of complication: How to organize data requests to make best use of the peak data rate of memory. Failing to do so can easily rob you of 50% or more of your memory bandwidth.

The numbers are big. Consider a single 64-bit CPU clocking at 2GHz. Fetching one instruction, plus one data read or write per cycle results in a raw data rate of 16 GBytes/s. Even if we assume 75% of that traffic is contained within the on-chip L1 and L2 caches, each CPU is demanding up to 4 GBytes/s bandwidth. With an 8 CPU cluster, assuming 50% utilization, that's 16 GBytes/s. And that's just for the CPU cluster!

Since the central processor is the heart of most SoCs, the CPU and GPU clusters have to be supplied with plenty of memory bandwidth. Otherwise they can easily stall, throwing away the majority of their theoretical peak processing power. So, unless the CPUs are spending most of their time accessing in-cache data, external memory bandwidth quickly becomes a big issue.

GPU's can easily consume more bandwidth than the CPUs, as they are parallel processors doing lots more work per clock cycle, because they are arrays of programmable ALUs working in parallel.

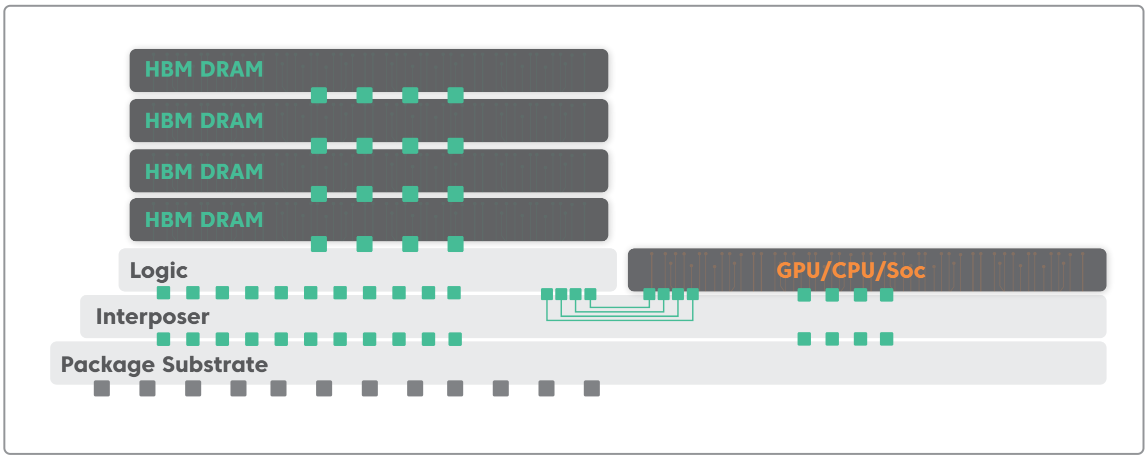


Figure 2: HBM2 memory stacks are high-performance solutions, however their extreme power consumption renders them useless for automotive and other embedded platforms.

The latest memory technologies such as HBM2 can deliver staggering bandwidths of up to 256 GByte/s. But these high-power technologies are designed for datacenters, not embedded automotive or mobile applications. We need to use more realistic low power memory technologies – usually LPDDR (low power dynamic RAM), which has a fraction of the available bandwidth.

The most common memory used in today's embedded or mobile devices is LPDDR4, which delivers 4.2 Gbps per pin. Just feeding a small CPU cluster can easily consume this bandwidth, before anything else (e.g. the GPU, modem or video) can use it.

Next generation LPDDR5 is just around the corner, capable of delivering an impressive 8 Gbps per pin. But this will come at the expense of consuming significantly more power and generating more heat. Thus, it will be several years before we see widespread use of LPDDR5 at maximum bandwidth in automotive systems.

Using caches vs external data

Another area that needs to be considered is the difference in workloads between a general purpose mobile platform capable of playing millions of different applications, compared to embedded automotive applications that are very tightly constrained in what they do. This means that the profile of workloads can be far better optimized for an automotive application.

For a demanding mobile app such as a game, a significant part of the data is being held locally. All sorts of caching schemes can be used to optimize how much data is being held locally on-chip to minimize external memory traffic. However, for a real-time system such as autonomous vehicles, new data is continuously being received from sensors at hundreds of Mbytes/s or more. Since that data is coming from outside the application, it must pass through the external memory subsystem at least once. Indeed, it will probably pass in and out of external memory multiple times as applications read and modify the data, then write it out again. This is especially true if low level functions are being performed on the data such as filtering or NN functions.

If the host CPUs are pre-processing that data, even the simplest loops rapidly cause GBytes of additional bandwidth load on the external memory subsystem. If the GPU is then used to process that data, it will also need to read the data written out by the CPUs, process it, and write out the results. It all adds up.

The task of managing data on modern embedded SoCs is already one of the biggest challenges facing SoC designers. So, when we want to add another high-performance, data-heavy accelerator such as a NN accelerator, the challenges become potentially insurmountable.

Why is NN processing so demanding?

Processing NNs is a massively parallel processing task. Until recently, it was not practical to create any significant NN accelerator, as silicon technology was not up to the task of integrating the huge number of computation elements needed.

However, it has been recognised by many in the field that NN computation is actually a data-driven, not computation-driven, problem. The performance of any NN accelerator will be determined by how data is moved between the thousands of computation engines during network evaluation. And since it is data-intensive, how best to use on-chip vs off-chip memory will be a crucial architectural issue. But one thing is for certain: when processing multiple HD sensor inputs in real-time, the volume of datamoving continuously through the system will be huge.

Also, NNs are getting wider and deeper as our knowledge of algorithms continues to improve. One of AImotive's concerns has been that because of our extensive use of multiple HD cameras, we need our NNs to accommodate as many as eight or more 3x1920x1024 camera inputs at 30fps or higher. That's almost 200 Mbytes/s per camera, or nearly 1.6 GBytes/s in total – just for the incoming data!

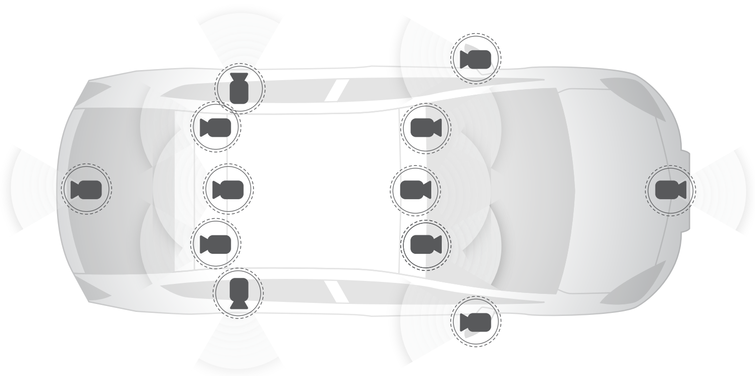


Figure 3: A self-driving vehicle must a 360° view to have a full understanding of their environment. Several high-resolution sensors must be used to a achieve this, each of which place further demands to the processing platform.

The first layers performing multiple convolutions will need to write outputs of similar sizes, probably multiple times, before the later operations start working on smaller sizes of input data. See how quickly we end up with 10's of GBytes/s of bandwidth?

That's why AImotive believes that in order to accommodate the extreme bandwidths required for multi-camera (or any LIDAR, Radar or any precision sensor fusion system), the NN computation platform must be carefully considered in the context the overall system architecture. Managing the system bandwidths needed to perform every function needed and, keeping every functional processing block operating as efficiently as possible, is key.

It's all about the data, not the OPS

Following extensive analysis of many NN workloads developed by AImotive's research engineers creating the aiDrive software stack, engineers concluded that the computation is not exclusively dominated by how many MACs were executing at the same time. It's in fact all down to data movement: how each intermediate result moves through the accelerator from one calculation to the next. What this showed was that the more that weights and intermediate results were stored as closely as possible to the relevant execution units, the more efficient the processing was. Thus, eliminating intermediate caches or central shared memory during execution is essential to achieving efficient performance.

That is why aiWare was created: to implement this deep knowledge of real workload-based data flow into a hardware engine that would be far more efficient than conventional approaches. We wanted to create an engine that could deliver sustained, extremely high efficiency NN computation while managing the data bandwidths both within the accelerator, and between the NN accelerator and other parts of the hardware platform, especially the CPU-based SoCs.

Many recent implementations of NN accelerators have found, just like the aiWare team, that their design relies on how much local memory is used. For aiWare, around 80%-85% of the total area is taken up with local SRAM blocks. It is the usage of memory and how data flows into and through it that really matters. That's why AImotive's key aiWare patents are all about data flow, not computation.

This is about real-time inference, not training

Many NN accelerators have been designed over the past few years – almost every week a new line appears! However, most of these target training workloads for datacenter applications. Some smaller engines are also targeting mobile applications, but these are only for low-bandwidth tasks such as speech recognition and image classification. As a result, while we see many big numbers for NN accelerators, their performance is based on delivering results for a different problem to those faced by designers of real-time inference engines for automotive and other power-constrained, high-reliability embedded applications.

For continuous operation in embedded applications such as autonomous vehicles, it is essential that data moves efficiently frame after frame for many hours at a time. There's no room for any system that needs to periodically clean up buffers, do garbage collection, or otherwise stop for maintenance. These are all facilities available to datacenter hosts, or indeed non-real time applications for mobile.

For automotive embedded inference, the memory strategies used must be robust under all conditions, and work continuously without fail.

Why don't we just make everything faster?

SoC's are getting faster every year, and as we continue to shrink process nodes. So surely, we'll be able to just make existing chips so fast and capable that this isn't a problem?

There are a few limitations that mean this approach is starting to break down:

1. Distance: although distances might shrink with each more advanced process node, the distance they need to communicate with higher integration is longer. The longer the distance you need to send a signal, the more energy you need. The faster you need to send that signal, the more energy you need. And the smaller the geometry, the less energy is available for each signal unless you amplify it – which increases power consumption. So even though silicon process geometries continue to shrink, we can't simply add more logic and expect it to all go faster.

2. Cost: each new advanced silicon node costs anything up to 4-times the previous one to make the masks for the chip. And while the cost per transistor reduces, it doesn't reduce as much as its used to thanks to the sheer complexity of the manufacturing processes. So advanced process nodes cost much more for the design, and for the final device

3. Noise: as process nodes get smaller, voltages drop and fewer electrons move for each bit. That makes every signal more susceptibleto noise. That's bad enough when a chip has billions of wires transmitting every clock cycle; but that's compounded by the electrical noise in a car. Will these systems be robust as we continue to shrink them?

4. Moore’s law: the infamous law that logic density doubles every 18 months has served us well for many years. But it simple isn’t working any more: logic density is only increasing 1.2x–1.4-times per generation. However, memory does continue to track process density. So, for future process nodes, the more memory dominates the design, the better it will scale

5. Memory: external memory bandwidth is not scaling at the same speed as on-chip memory. That’s because moving data on and off chip demands huge speeds – up to 2Gbps per pin for HBM2. That takes a lot of power, and because its focused in a small area also causes big thermal issues. Doubling external memory bandwidth isn’t easy – especially when trying to stay within tough power and thermal envelopes such as an embedded automotive application.

The central processor is, well, central!

Many industry stakeholders believe that future autonomous vehicles will rely on a powerful central processing unit. Indeed, a growing number of automotive electronics engineers are considering future vehicles becoming “datacenters on wheels”, just as the smartphone was a “PC in your pocket”. It does make sense: the flexibility of a central processing-based approach means we don’t need all the answers when building our processor hardware platform. This will, for example, allow upgradeability, using modern datacenter approaches of adding and replacing blades of computing and virtualizing all resources.

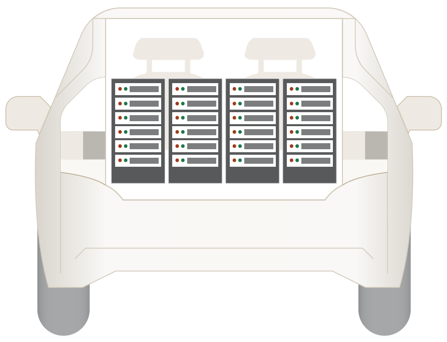


Figure 4: A datacenter on wheels may not be the best solution for automotive OEMs and end users. Automotive hardware platforms must be small, power efficienct and high-performance.

The challenge is that a datacenter is not designed to be a real-time system. It can sometimes stop and reconfigure itself; reboot some processors while moving compute loads to others; continuously change its workloads when it sees bottlenecks. It also consumes a lot of power to achieve that flexibility, which is why the biggest challenge for modern datacenters is how much power they can get to them. Does that sound like the ideal approach for a car?

This approach will be used for many non-critical functions in future cars. However, for real-time, life-critical functions such as the essential control systems for L4/L5 vehicles, the flexibility of a datacenter-like approach must be traded off against reliability and robustness in all situations, as well as meeting challenging power, thermal and electrical noise constraints.

That means we must ensure the central processor is surrounded by increasingly intelligent subsystems, that can manage the available bandwidth, processing and power constraints in a more distributed manner. Moving some of the NN processing out to each sensor, for example, could mean the central NN acceleration resource can be smaller, and thus, more manageable, dedicated to critical tasks such as trajectory scenario planning or high-level object tracking.

Industry stakeholders will pour significant resources into finding the best options for both centralized and distributed solutions. In fact, the hardware platforms and solutions OEMs and Tier1s chose to implement their autonomous systems could become major market differentiators in the coming years.

Offloading the central processor

When looking at a vision-first system, AImotive realized that it will be many years before the automotive industry converges on the agreed best hardware architecture. However, to get the processing power needed for AI-based software solutions like AImotive's aiDrive into a small box, the NN acelerator has to have the flexibility to be used in all types of hardware architectures as new solutions are developed. These can range from a central processing resource to a distributed, edge processing approach – that's what aiWare was designed for.

One key factor is how to manage system bandwidths. Each camera generates high bandwidth data streams. The NN computation for front-end perception and classification needs to be extremely fast and 100% reliable. The aiWare memory architecture was designed to offer the option for system designers to stream pre-processed camera (or other HD sensor) data directly into the aiWare external memory. This offers the potential for all the bandwidth relating to the gathering and pre-processing of sensor data to be offloaded entirely from the host processor SoCs. The aiWare subsystem can then deliver much lower bandwidth results into the central processor for later functions such as trajectory planning and control.

This approach enables much better scalability, as it is far easier to design a scalable dedicated sensor fusion front-end system based on low-level hardware, than to try to scale up complex processors and deal with all the complexity related to managing the huge bandwidths and complex memory traffic.

This also means that the flexibility of the central processor can be much better utilized for what its best at: flexible, complex heuristics and handling a broad range of applications. For example, much of the later stages of an AI processing pipeline rely on a mix of conventional computer vision and other algorithms that are far better executed on a general-purpose platform.

So, by providing modest NN acceleration within the central processor, augmented by high performance dedicated external NN processing, the ideal mix of scalable performance at lowest possible power consumption can be achieved. And by using a more distributed approach, it will also be far easier to implement redundancy schemes that deliver the required levels of always-on safety.

Conclusions

The future is an exciting place with autonomous vehicles. However, to be accepted by people around their world, we must deliver previously unseen levels of performance at challenging power budgets, cost budgets in demanding operational environments. Simply doing things the way we've done them before won't be an option.

AImotive uses a holistic approach to develop a comprehensive portfolio of technologies to solve the challenges: the aiDrive software stack; aiSim simulation and aiWare hardware IP. Each product is optimized thanks to the in-depth knowledge supplied by the other teams, creating a unique development environment where software engineers can anticipate hardware platforms when making crucial design decisions.

Despite the amazing progress that hardware technologies have made over the past 10–20 years, in both the high-performance datacenter as well as highly constrained mobile environments, it is wrong to assume that performance will ever be cheap. Automotive will always be especially demanding: extreme operating conditions, highly constrained power and thermal resources, tough cost constraints, long development timescales and even longer product lifetimes.

Hardware innovation will help solve these challenges not only by using the most advanced technologies to integrate more functions and make things faster. Innovation will also be maximized by creating new hardware architectures capable of leveraging existing mature technologies in new ways, enabling us to evolve automotive and other embedded hardware platforms. By combining these two sources of innovation, we will be best positioned to tackle the future needs of these exciting new markets.