The Role of Edge Offload for Hardware-Accelerated Mobile Devices

Mahadev Satyanarayanan, Nathan Beckmann, Grace A. Lewis, Brandon Lucia
satya@cs.cmu.edu, beckmann@cmu.edu, glewis@sei.cmu.edu, blucia@andrew.cmu.edu
Carnegie Mellon University

ABSTRACT

This position paper examines a spectrum of approaches to overcoming the limited computing power of mobile devices caused by their need to be small, lightweight and energy efficient. At one extreme is offloading of compute-intensive operations to a cloudlet nearby. At the other extreme is the use of fixed-function hardware accelerators on mobile devices. Between these endpoints lie various configurations of programmable hardware accelerators. We explore the strengths and weaknesses of these approaches and conclude that they are, in fact, complementary. Based on this insight, we advocate a software-hardware co-evolution path that combines their strengths.

CCS CONCEPTS

• **Hardware → Application specific processors**; • **Computer systems organization → Single instruction, multiple data**; **Multiple instruction, multiple data**; **System on a chip**; • **Networks → Wireless local area networks**; • **Human-centered computing → Mobile devices**; • **Computing methodologies → Artificial intelligence**; **Machine learning**.

KEYWORDS

edge computing, mobile computing, offloading, cyber foraging, 5G, ASIC, GPU, TCB, devops, stealth, disconnected operation, latency

ACM Reference Format:
Mahadev Satyanarayanan, Nathan Beckmann, Grace A. Lewis, Brandon Lucia. 2021. The Role of Edge Offload for Hardware-Accelerated Mobile Devices. In *The 22 nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21), February 24–26, 2021, Virtual, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3446382.3448360

1 Overcoming the Mobility Penalty

At the dawn of mobile computing, it was recognized that portable devices suffer a *mobility penalty* because of their need to be small, lightweight, and energy-efficient [51]:

> *Mobile elements are resource-poor relative to static elements.* Regardless of future technological advances, a mobile unit's weight, power, size, and ergonomics will always render it less computationally capable than its static counterpart. While mobile elements will undoubtedly improve in absolute ability, they will always be at a relative disadvantage.

This prediction has proved to be true over a period of nearly 30 years. Figure 1 illustrates how Moore's Law has differentially benefited mobile and server hardware since 1997. Each row compares a state-of-the-art mobile device and server at one point in time. Over the same period of time, user expectations of functionality, performance and productivity improvement from their mobile devices has grown. For example, even the smallest and lightest smartphone is expected to support speech recognition today. Bridging the large gap between user expectation and hardware reality is a perennial challenge.

	Typical Server		Typical Mobile Device	
Year	Processor	Speed	Device	Speed
1997	Pentium II	266 MHz	Palm Pilot	16 MHz
2002	Itanium	1 GHz	Blackberry 5810	133 MHz
2007	Intel Core 2	9.6 GHz (4 cores)	Apple iPhone	412 MHz
2011	Intel Xeon X5	32 GHz (2x6 cores)	Samsung Galaxy S2	2.4 GHz (2 cores)
2013	Intel Xeon E5-2697v2	64 GHz (2x12 cores)	Samsung Galaxy S4 Google Glass Explorer Edition	6.4 GHz (4 cores) 2.4 GHz (2 cores)
2016	Intel Xeon E5-2698v4	88.0 GHz (2x20 cores)	Samsung Galaxy S7 Microsoft HoloLens	7.5 GHz (4 cores) 4.16 GHz (4 cores)
2017	Intel Xeon Gold 6148	96.0 GHz (2x20 cores)	Pixel 2	17.4 GHz (8 cores)
2020	Intel Xeon Gold 6348H	110.4 GHz (2x24 cores)	Samsung Galaxy S20 Microsoft HoloLens 2 Google Glass Enterprise Ed. 2	18.66 GHz (8 Cores) 18.6 GHz (8 cores) 6.8 GHz (4 cores)

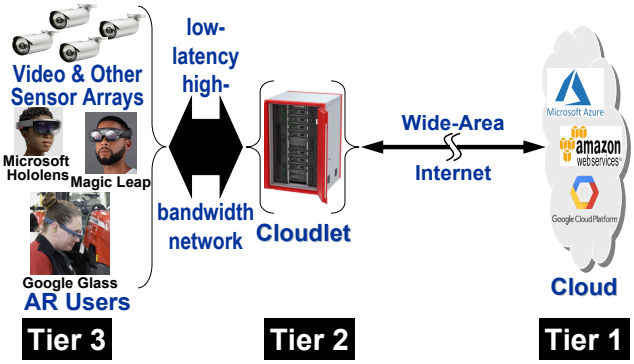
Source: Adapted by Roger Iyengar from Chen [7] and Flinn [20]. "Speed" = (number of processors × number of cores × per-core clock speed), ignoring GPUs/ASICs.

Figure 1: The Mobility Penalty Over Two Decades

One approach to overcoming the mobility penalty was first demonstrated in 1997 by Noble et al [44] for real-time speech recognition. Sensor data is captured on the device, transmitted to a nearby server for compute-intensive processing, and the result returned to the device. This approach, originally called "cyber foraging" [3, 52] and now "offloading", has been widely used by mobile services such as Apple speech recognition and Google language translation. A key motivation for the recent emergence of *edge computing* is the ability to perform offloading at much lower latency and higher bandwidth on a nearby *cloudlet* than on the distant cloud [53] (Figure 2).

More recently, a different approach to overcoming the mobility penalty has been demonstrated. This approach uses custom-designed on-device ASICs (application-specific integrated circuits) [13] for applications such as deep learning inference [36], super-resolution [37] and scene analysis [42]. Since this approach has been so successful, a question that is being actively debated in the mobile computing research community is whether the first approach is needed any longer. Is offloading a technique whose time come and gone? Was it merely a transitional strategy that is no longer needed? If it is still needed, how should designers choose between the two approaches? Are there circumstances in which offload is valuable in addition to ASICs? What are the tradeoffs between efficiency and generality in hardware accelerators, and how does one navigate them?

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HotMobile '21, February 24–26, 2021, Virtual, United Kingdom
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8323-3/21/02.
https://doi.org/10.1145/3446382.3448360



Offloading from device (Tier-3) to cloudlet (Tier-2) incurs much lower latency than offloading to the cloud (Tier-1).

Figure 2: Three-tier Model of Edge Computing

We address these questions in the rest of the paper. After examining the strengths and weaknesses of each approach in Sections 2 and 3, we explore in Section 4 how their strengths can be combined. The result (Sections 5 and 6) is an evolutionary path forward that is superior to either approach in the face of uncertainty. The primary contribution of this paper is to bring to the attention of mobile system designers relevant insights and experience that have been learned by computer architects and hardware engineers over many decades. A secondary contribution is to highlight the challenges of building mobile computing systems to the latter community.

2 Strengths of Edge Offload

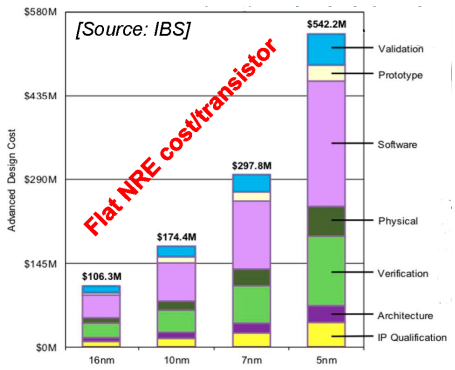
2.1 Speed, Flexibility and Cost of Development

Edge offload avoids hardware development. Being a software strategy, speed and cost of project execution are key advantages that it holds over all strategies involving creation or integration of custom hardware. Speed is particularly important in a highly competitive and rapidly evolving marketplace. The Google Play Store and Apple App Store are examples of ecosystems in which third parties can introduce new applications and rapidly evolve them. The ability to quickly introduce and evolve new *edge-native applications* [57], and to respond with agility to competitive threats are key advantages in this setting. A *DevOps model* that blurs the distinction between development and deployment can be used for extreme agility, as well as reduction in average release cycle time from weeks to days [6, 39].

These strengths of software over hardware are not new. They are enshrined in the concept of the microprocessor itself, which was invented by Intel in 1971 to avoid the development cost of creating slightly different hardware chip sets for customized realizations of a common underlying functionality [41]. The computing world has changed beyond recognition since 1971, but the inherent simplicity, speed and cost advantages of pure software development remain unchallenged by custom hardware. Figure 3 shows that the non-recurring engineering (NRE) costs of chip design remain very high even today. Though older manufacturing nodes can provide some reduction in NRE, these savings come with complex tradeoffs in performance and energy-efficiency [33].

2.2 Legacy Support

The ability to support *legacy devices* is a second major strength of edge offload. At each step of evolution, as a device with a new



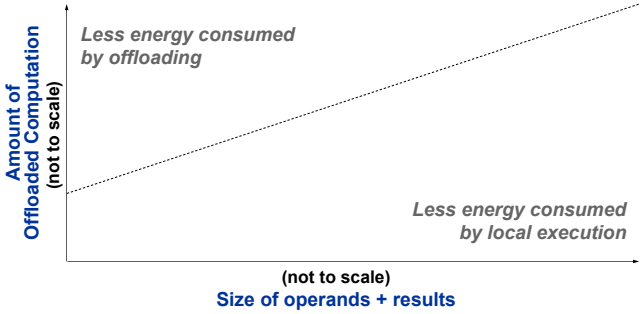
Adapted from Asanovic MICRO-25 Keynote [1]

Figure 3: Cost of Custom Chip Design

hardware accelerator is introduced, there will be a period of time when it coexists with one or more earlier generations of devices. An application that can run on both old and new devices can target a much larger market than one targeting only new devices. Edge offload can be used to provide speedup on older devices. Of course, upward compatibility is a double-edged sword: it reduces the incentive for a user to replace an old device. A more balanced view is to highlight attributes such as battery life as the primary benefit of the new device. With this evolutionary strategy, there is a sliding window of generations of devices that can run a particular application, but the newer devices have much longer battery life. The computer industry has learned repeatedly (dating back to IBM's System/360 architecture) that there is high value in helping users and application developers preserve their legacy investments [54].

2.3 Extending Battery Life

Under the right conditions, edge offload can reduce energy usage on a mobile device. Figure 4 qualitatively illustrates this tradeoff space. For the offloaded operation, the X axis shows the volume of data transmitted and received, while the Y axis shows the total amount of offloaded computation. The extreme points of this tradeoff space are easy to identify. When data transmission volume is negligible, but the amount of offloaded computation is large, offloading saves energy on the mobile device. This corresponds to the top left region of the figure. When these traits are reversed (bottom right), local execution uses less energy on the device than offloading. The specifics of this tradeoff space depend on many factors, such as the energy efficiency of the mobile device and network quality [20–22].



Conceptual figure only, not real measurements.

Figure 4: Energy Tradeoff Space

2.4 All-You-Can-Eat Scale-Up

Today, edge offload is typically used when a fixed computation must complete within a tight latency bound. For example, deeply immersive augmented reality on a wearable device has a latency bound of 16 milliseconds [18] for good user experience. For *wearable cognitive assistance (WCA)* applications, Chen et al [8] identify *loose* and *tight* latency bounds. Below the tight bound, the user does not perceive any further improvement. Offloading is superfluous if the relevant bound can be met using ASICs on a mobile device.

In contrast, unlimited scale-up may be valuable in the future. For example, imagine WCA for business, legal or diplomatic negotiations. These are adversarial settings in which even a subtle advantage in discerning an opponent's weakness can be valuable. Human communication involves many diverse inputs: the language content and deep semantics of the words, the tone in which they are spoken, the facial expressions and eye movements with which they are spoken, and the body language and gestures that accompany them. A cognitive assistant that can process all of these distinct channels of information in real time, and also combine them with additional knowledge from external sources will prove superior to one that is less capable because it does not use offloading. AI-based assistance for military applications ("Battlefield 2.0" [17]) also has an unlimited appetite for computation within a tight time budget. In such settings, edge offload will be valuable even with ASICs.

2.5 Aggregation

Crowd-sourced applications such as participatory sensing [5] and communal image collection [47] are inherently multi-device. A key decision is whether to perform processing on-device or offload it to a cloudlet or the cloud. Often processing only on-device is not practical: e.g., when aggregating data is fundamental to the application, as in a shared virtual environment. Even when on-device processing is possible, offloading can be attractive if the amount of data from other devices exceeds that generated locally.

3 Strengths of ASICs

3.1 Ubiquity

By eliminating the dependence on edge computing, ASICs enable a mobile device to operate anywhere. This will be especially valuable in the near future, when the rollout of edge computing is still young and coverage is sparse. There is a chicken-or-egg circularity to the likely speed of this rollout. If there is rapid creation and adoption of valuable edge-native applications, there will be high incentive for service providers to speed up deployment of edge infrastructure and 5G. On the other hand, the widespread deployment of edge infrastructure is itself an incentive for creation of new edge-native applications, while its sparseness will retard their creation. The use of custom ASICs eliminates the risk associated with the dependence on edge computing. However, it comes at the high NRE cost of custom chip design (Figure 3).

3.2 Disconnected Operation

Wireless connectivity (Wi-Fi or 4G LTE) is ubiquitous today. Offloading to the cloud is thus a fallback when edge infrastructure is unavailable. This will degrade the quality of user experience (QoE) for an edge-native application, because of increased end-to-end latency. The use of 5G rather than 4G LTE will not eliminate this problem because it only improves the final hop of a very long multi-hop network path from device to cloud. Application-aware graceful degradation is feasible [28, 44, 59], but it comes at the cost of implementation complexity since multiple levels of application-specific QoE have to be supported. The use of ASICs completely eliminates the need for fallback mechanisms. The device is able to provide full service even in locations without network connectivity.

3.3 Ultra Low Latency

Today, web-based interactive applications remain usable even at mean end-to-end latencies on the order of 100 ms, with very long tails. They achieve this latency and jitter tolerance through interaction metaphors such as hyperlinks that avoid immersivity. The much lower latency and jitter offered by edge computing will enable more deeply immersive interaction metaphors such as 3D collaboration [31]. Although the sub-millisecond *Tactile Internet* [49, 58] is a holy grail, it is hard to beat a purely on-device solution for the lowest latency and jitter. ASICs are thus at a strong advantage for the most deeply immersive applications.

3.4 Scalability

The average load placed on shared edge computing infrastructure by a typical mobile device determines the overall scalability of the system. Higher per-device loads translate to longer queueing delays, both for network access and for cloudlet scheduling. A system in which most of the marginal burden of a client is borne by the client itself is likely to be more scalable than alternative designs. This observation is as true today as it was when first articulated more than three decades ago [56]. The use of local computation to reduce or eliminate offloading thus contributes to the overall scalability of the system [59]. QoE is also improved because load-dependent queueing delays are avoided.

3.5 Battery Life & Intermittent Computing

As discussed in Section 2.3, avoiding offloading sometimes saves energy. Energy-efficiency increases the amount of data a device can process before depleting its battery, or exhausting a limited number of charge/discharge cycles. For energy-harvesting devices [11, 15, 43] that are used in intermittent computing [38], efficiency determines performance [16]. The time for a task is often dominated by energy collection [25]; more efficient use of energy minimizes long recharge periods, which minimizes makespan. In Figure 4, using an ASIC enlarges the region over which local compute wins: the separating line between the two regions is raised considerably.

3.6 Zero-Trust and Stealth Operation

A device that only computes locally need not trust any other computing resources. The use of ASICs can thus simplify the trust model of a system, and shrink its TCB by avoiding the need to vet or certify infrastructure used in offloading. Local computation's implicit trustworthiness may also be appealing to users who are especially concerned about the privacy of their data. In covert military operations, avoiding offload lowers observability.

Architecture	Efficiency (GFLOP / J)
CPU (Core i7)	1.14
FPGA (Xilinx LX760)	3.62
GPU (NVIDIA GTX285)	6.78
GPU (AMD R5870)	9.87
ASIC	50.73

Source: Table 4 in Chung et al [9]

Figure 5: Energy Efficiency of Matrix Multiplication Kernel

4 Merging Strengths: Considerations

Sections 2 and 3 suggest that the two approaches to overcoming the mobility penalty are complementary. Can we combine their strengths and neutralize their weaknesses? A hint on how to do this comes from the observation that creating an ASIC for a mobile device is similar in spirit to extending its instruction set with complex instructions. A wealth of insights has been gained in the multi-decade CISC versus RISC debate. Two observations from this large body of knowledge are of particular value to us. The first is by Saltzer et al on end-to-end arguments [50]:

> The arguments that are used in support of reduced instruction set computer (RISC) architecture are similar to end-to-end arguments. The RISC argument is that the client of the architecture will get better performance by implementing exactly the instructions needed from primitive tools; any attempt by the computer designer to anticipate the client's requirements for an esoteric feature will probably miss the target slightly and the client will end up reimplementing that feature anyway.

Trying to anticipate the functionality of an ASIC for a specific application is risky even when its algorithms are well understood. It is better to implement the application entirely in software, and easily modify it as the application evolves rapidly through actual usage. When the code stabilizes, it can be carefully reviewed to extract the interface for hardware acceleration. Such an empirical approach is less risky than premature specification of the ASIC.

The second observation is by John Cocke in his Turing Award lecture about the evolution of RISC technology [10]:

> Avoid complex instructions *whenever the same effects can be realized just as quickly by sequences of simple instructions*. For a given application, cache will become the equivalent of fast ROM for the macro-instructions most commonly used by that application.
>
> (Italics in the original)

This maxim reinforces the previous point about using empirical data to design the ASIC. If traces of the actual instructions executed during a typical run of an application are available, the empirical analysis suggested by Cocke is possible. Without such empirical analysis, there is high risk that a proposed ASIC may not provide the expected speedup. A designer's intuition is only a weak guide to ensuring that creation of an ASIC will be worth the effort.

These arguments apply equally well today, and suggest that there will always be a role for general-purpose processors that expose simple primitives as building blocks for larger computations. RISC CPUs lie at one extreme, but there is an increasingly rich landscape of *"programmable accelerator"* designs that strike a balance between the generality of CPUs and the efficiency of ASICs. FPGAs and GPUs are popular examples of programmable accelerators. Figure 5 shows that programmable accelerators significantly improve energy efficiency over CPUs. Though ASICs are far more efficient (Figure 5 and [9, 30]), recent research has narrowed this gap [26, 45, 46].

Since programmable accelerators improve efficiency by specializing hardware, the key tradeoff in their design is between programmability and efficiency. Hardware specialization increases efficiency primarily by eliminating overheads paid by the CPU: control overheads, such as repeated and unnecessary instruction fetch and decode operations, and data movement overheads, such as repeated and unnecessary reads and writes to a register file. At the extreme, the direct implementation of frequently executed application logic in specialized circuits can completely eliminate control overheads. However, this limits use of the device to that specific application. Instead, a programmable accelerator provides high efficiency for a broad *class of workloads* by partially, but not completely, specializing hardware to suit the class's common characteristics. This suggests a complex tradeoff between programmability and efficiency, ranging from the flexible, bit-level reconfigurability of an FPGA to the restricted but efficient operation of many machine-learning accelerators, with many design points in between.

Programmable accelerators often require non-trivial software engineering effort to port applications to specialized interfaces, such as CUDA for the GPU [35, 60], or Chisel for the FPGA [2]. But this effort is orders-of-magnitude less than the NRE cost to redesign an ASIC, especially as accelerator interfaces grow more CPU-like and compiler support improves. Significant progress has been made on streamlining programming interfaces for programmable accelerators, avoiding the need for hardware description languages or hand-coded assembly [26, 45, 46]. However, software development cost remains a challenge for programmable accelerators. Although unlikely to universally replace ASICs, programmable accelerators are often useful as stepping stones between the initial (software) offload version of applications and their eventual (hardware) ASIC implementations. Furthermore, for applications that are updated regularly or whose efficiency is close to an ASIC on a programmable accelerator, implementing an ASIC is probably not worth the cost. In such cases, programmable accelerators may remain fielded for long periods of time or, possibly, even indefinitely.

Programmable accelerators may play a particularly important role in Tier-2 (cloudlets) [55], where energy-efficiency requirements are less stringent. There is substantial experimental evidence from many edge-native applications [8, 29, 32] that offloading to CPU-only cloudlets does not provide sufficient speedup — it is necessary to use cloudlets equipped with programmable accelerators such as GPUs. Cloudlets may prefer the flexibility of a programmable accelerator to support a wider range of applications, whereas ASICs may be more attractive in a mobile device, where energy efficiency is paramount, or in the cloud, which has the scale to amortize NRE by keeping ASIC utilization high [33]. Ultra-low-power systems and energy harvesting devices [12, 24] are likely to benefit most from an ASIC. However, as applications for these emerging devices continue to mature [40, 48], programmability is likely to remain an important issue for a long time to come.

5 Merging Strengths: How to Do It

These considerations lead to the evolutionary path shown in Figure 6. In this figure, time flows from left to right. Since this is a logical diagram, it is not drawn to scale. What matters are the milestones shown, not the distance separating them. The evolution begins with a software-only release ($\mathbf{A}_1$) of a new edge-native application, **A**. In many cases, the offloaded code may include support ($\mathbf{B}_1$) from a programmable accelerator at Tier-2. Based on usage experience and response to competitive threats, **A** may go

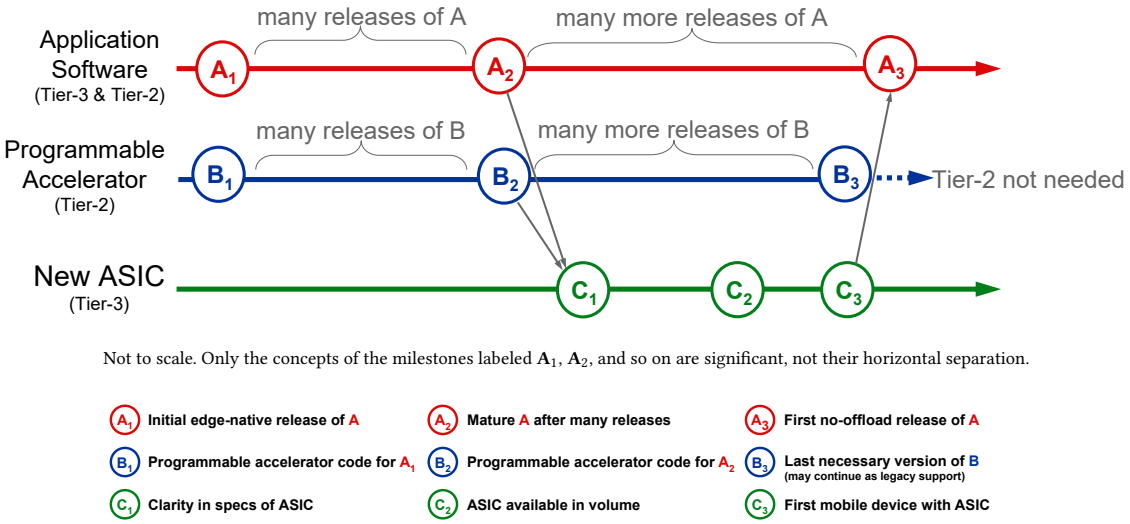


Figure 6: Logical Timeline: Combining Edge Offload & ASICs

through extensive improvement and refinement over a period of many months or possibly years. During this phase of extensive evolution, agility is paramount. Remaining software-only is advisable.

Eventually, the application stabilizes at a point $\mathbf{A}_2$, with programmable accelerator support $\mathbf{B}_2$. Although the application may continue to evolve, its rate of change is far lower than in the early stages. Since further change tends to be incremental, widely used software engineering practices such as modularity, separation of concerns, and use of design patterns can be leveraged to avoid major disruptions to the code base. It is at this point that the risk of mis-targeting highlighted by Saltzer et al [50] becomes low enough to consider creating an ASIC for this application. Prior to this point, such efforts run too high a risk of premature optimization, leading to an ASIC that rapidly becomes obsolete. Approaches such as aspect-oriented programming [34] can be used to minimize code changes for determining when this point has been reached.

As recommended by Cocke et al [10], empirical studies of the code sequences of $\mathbf{A}_2$ and $\mathbf{B}_2$ can now be performed to quantitatively establish the performance and energy benefits of a proposed ASIC. This is not an exercise in the abstract, because it is based on actual trace data from live executions of **A** in real use. This exploratory design work leads to the complete design and fabrication, $\mathbf{C}_1$, of a new ASIC. It is in the creation of $\mathbf{C}_1$ from empirical data available through $\mathbf{A}_2$ and $\mathbf{B}_2$ that the high NRE costs referenced by Asanovic [1] are incurred. Using early samples of $\mathbf{C}_1$, a no-offload version of **A** can be created, and the performance and energy wins can be validated. If needed, the design of $\mathbf{C}_1$ can be tweaked in response to shortcomings revealed by actual experiments.

At this point, the development of a validated ASIC for **A** is complete. Effort can then be focused on high-volume production and yield optimization, resulting in $\mathbf{C}_2$. Concurrently, the design of a new mobile device that leverages $\mathbf{C}_2$ can proceed. This leads to $\mathbf{C}_3$, where a new mobile device with the new ASIC is available in quantity. Upon release, it is able to run $\mathbf{A}_3$, the first no-offload release of **A**. There will typically be a period where both legacy devices and new devices are supported. Hence, both no-offload and offload versions of **A** will need to be maintained. Eventually, the offload version of **A** can be end-of-lifed.

The entire discussion above assumes that a new ASIC is created to support a single application, **A**. In practice, a new ASIC may support multiple new applications. This would modify Figure 6 to include dependence of $\mathbf{C}_1$ on quantitative insights from software-only implementations of multiple applications (including their programmable accelerator code). $\mathbf{C}_3$ would then enable no-offload versions of all those applications (e.g., $\mathbf{A}_3$) on new ASIC-equipped mobile devices.

6 Discussion

6.1 Variants of the Logical Timeline

Figure 6 depicts one plausible timeline for the lifecycle of an application in a world where hardware spans the full range from general-purpose CPUs and programmable accelerators to fixed-function ASICs. Other timelines are also possible, driven by technology changes, market pressures, and application requirements. Below, we discuss some possible variants of the canonical timeline.

Technology changes are always a major influence on the design timeline. For example, entering the era of "dark silicon" [19] left chip designers with ample chip real estate for a large number of miscellaneous ASICs. Taken to an extreme, this design strategy yields a collection of ASICs with some of the behavior of a programmable accelerator. This "garden of ASICs" model develops along a timeline with many new implementations, one for each new ASIC. Each ASIC has lower NRE than shown in Figure 3, due to (a) high reuse across designs and (b) very low configurability demanded by the rigidly-constrained functionality of each ASIC.

Market pressure could influence this timeline, providing an incentive to hardware developers to avoid programmability. Producing ASICs despite changing application requirements requires consumers to upgrade, lest they be unable to access a device's latest features. For a captive consumer market, a new device is the only

option. In this case, the cadence of the green circled points ($\mathbf{C}_i$) on the "new ASIC" timeline would be spaced out over time in bursts corresponding to a product cycle. This timeline is unlikely to include programmable acceleration as intermediate or terminal points (i.e., entirely omitting the timeline of blue circled points, $\mathbf{B}_i$). Such an ASIC-only approach carries substantial development risk, as it precludes incremental patching even in early-phase development.

Finally, application evolution and user demand are likely to be the most urgent and direct drivers of system design. In situations where application requirements do not stabilize in a few months to a few years, long-term flexibility with efficiency becomes crucial. This makes a strong case for permanent adoption of programmable accelerators rather than ASICs. The resulting timeline would look substantially different from Figure 6. A long period of high NRE cost would first produce a mature programmable accelerator for this genre of application. By leveraging (re-)programmability in the field and efficiency approaching ASIC levels, applications would avoid ever making the jump to a stand-alone ASIC. The programmable accelerator itself will accumulate optimizations over multiple generations of design and deployment, raising the bar for an ASIC to provide an efficiency advantage. This approach is low risk for the designer and provides high value to the consumer, but it depends upon the architect having the foresight to create an accelerator with just the right balance of generality and efficiency.

6.2 Enabling Tools and Practices

For the developer of a mobile application, it would be ideal to abstract where code executes: offloaded to Tier-2, or locally on an ASIC. In practice, the implementation of code offload requires more machinery than local execution. This includes code to manage the network connection to Tier-2, to serialize input and output parameters for network transmission, to validate results, and to deal with connectivity problems. This support can be provided in many ways: within libraries, within language-based frameworks such as Java RMI [27] and MAUI [14], or as plugins to integrated development environments (IDEs) such as Eclipse or Visual Studio Code. Tools for developers to control where a function executes can be helpful in the debugging and performance tuning aspects of the timeline shown in Figure 6. Also valuable will be tool suites (such as those of Balan et al [4]) that simplify the use of edge offload when porting legacy desktop or cloud applications to mobile devices.

6.3 Business Model Implications

For the evolution in Figure 6 to be self-sustaining, it must offer business viability for all stakeholders in the 3-tier model of Figure 2. At Tier-3, mobile devices are typically purchased by end users for their own use. The purchase price of a new device can include a premium for the new ASIC(s) needed to support application **A**. This premium will reflect the marginal cost of the ASIC(s) in the device, plus the amortization of a large fixed cost across millions of devices. That fixed cost includes the high NRE cost of ASIC development (Figure 3), as well as the multi-year application-specific research and development cost leading to that ASIC design (i.e., the cost of the entire green timeline in Figure 6). This approach is well established as a successful business model at Tier-3.

Paying for Tier-2 services (for points $\mathbf{B}_i$ on the blue timeline) is more challenging, especially in a fragmented ecosystem. Most end users are unaware of edge offload, so direct billing is unlikely to be successful. Pricing this cost into A may more successful, but it is difficult to align incentives for optimal low-latency offload between application creator and Tier-2 service provider. Operating Tier-2 at low utilization is conducive to low end-to-end latency, but increases the capital investment necessary. Aligning these incentives is easier in a monolithic ecosystem that is dominated by a single entity.

At the heart of these business challenges is the transparency of offloading. This is the secret to its technical success, but it also hides value from end users. In addition, operational costs tend to be higher at Tier-2 than at Tier-1. This reflects lower economies of scale in system management, arising from the dispersed nature of Tier-2 relative to an exascale cloud data center. These are some of the complications faced by the approach we advocate in this paper.

7 Closing Thoughts

One of the strongest forces in mobile computing today is the hunger for parallelism. Modern ASICs and programmable accelerators achieve one to two orders of magnitude higher degrees of parallelism than general purpose CPUs (10^3 to 10^4 GPU cores versus 10^2 CPU cores). Yet, both at Tier-2 and Tier-3, the versatility and open-endedness of CPU cores are also valuable. This tradeoff between scale and versatility in parallelism reflects the venerable SIMD/MIMD taxonomy of Flynn [23]. Today, the architecture community advocates exposing the inherent parallelism in an application through a higher-level, abstract representation, such as dataflow, and then using compile-time or runtime mapping to diverse SIMD/MIMD micro-architectures. One can view Figure 6 as conveying a similar message of delayed binding:

> Design your mobile application ignoring whether parallelism is achieved by an ASIC on Tier-3, or via edge offload to Tier-2. Optimize later.

Viewed as autonomous mobile computing systems with built-in sensing, processing, and persistent storage, humans achieve amazing feats of cognition and perception without offloading. The ASICs represented by our neural circuts are clearly the secret to this success. Their designs were honed over a billion years of evolution. Our chances of improving upon nature in a decade, or even a century, are negligible if we are bound by the same rules as biological evolution. Offloading is a way to "cheat." It allows us to leverage compute resources that are larger, heavier, more energy-hungry and more heat-dissipative than could ever be carried or worn by a human user. At least for the next decade, and likely well beyond that, we see edge offload as the enabler of new transformative applications on wearable devices. It is the key to accelerating by many orders of magnitude the discovery of ASIC designs that will become the building blocks of future cyber-human systems.

Acknowledgements

We thank Landon Cox, our shepherd, and the anonymous reviewers for their guidance in improving this paper. This research was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051 and by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Intel, Vodafone, Deutsche Telekom, Crown Castle, InterDigital, Seagate, Microsoft, VMware and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

REFERENCES

[1] K. Asanovic. Rejuvenating Computer Architecture Research (and the whole semiconductor industry) with Open-Source Hardware. Keynote Talk slides, MICRO-52, October 2019.

[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1216–1225, San Francisco, CA, 2012.

[3] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The Case for Cyber Foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop*, 2002.

[4] R. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying Cyber Foraging for Mobile Devices. In *Proceedings of the 5th International Conference on Mobile Systems Applications and Services*, San Juan, Puerto Rico, June 2007.

[5] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *In Proceedings of the Workshop on World-Sensor-Web (WSW'06)*, 2006.

[6] M. Callanan and A. Spillane. DevOps: Making It Easy to Do the Right Thing. *IEEE Software*, 33(3):53–59, May-June 2016.

[7] Z. Chen. *An Application Platform for Wearable Cognitive Assistance.* PhD thesis, Computer Science Dept., Carnegie Mellon Univ., 2018.

[8] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, Fremont, CA, October 2017.

[9] E. Chung, P. A. Milder, J. C. Hoe, and K. Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In *Proc. of the 43rd annual IEEE/ACM intl. symp. on Microarchitecture (MICRO-43)*, 2010.

[10] J. Cocke and V. Markstein. The Evolution of RISC Technology at IBM. *IBM Journal of Research and Development*, 34(1), January 1990.

[11] A. Colin, E. Ruppel, and B. Lucia. A Reconfigurable Energy Storage Architecture for Energy-Harvesting Devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2018.

[12] A. Colin, E. Ruppel, and B. Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, New York, NY, USA, 2018. ACM.

[13] L. Cox and L. Ao. Levelup: A thin-cloud approach to game livestreaming. In *Proceedings of the Fifth IEEE/ACM Symposium on Edge Computing (SEC2020)*, November 2020.

[14] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, June 2010.

[15] B. Denby and B. Lucia. Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[16] H. Desai and B. Lucia. A Power-Aware Heterogeneous Architecture Scaling Model for Energy-Harvesting Computers. *IEEE Computer Architecture Letters*, 19(1), 2020.

[17] Z. Doffman. Battlefield 2.0: How Edge Artificial Intelligence is Pitting Man Against Machine. *Forbes*, November 2018.

[18] S. R. Ellis, K. Mania, B. D. Adelstein, and M. I. Hill. Generalizeability of Latency Detection in a Variety of Virtual Environments. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 48, 2004.

[19] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *SIGARCH Comput. Archit. News*, 39(3):365£376, June 2011.

[20] J. Flinn. *Cyber Foraging: Bridging Mobile and Cloud Computing via Opportunistic Offload.* Morgan & Claypool Publishers, 2012.

[21] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-Tuned Remote Execution for Pervasive Computing. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, Schloss Elmau, Germany, May 2001.

[22] J. Flinn, S. Park, and M. Satyanarayanan. Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[23] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C-21(9), September 1972.

[24] G. Gobieski, N. Beckmann, and B. Lucia. Intelligence beyond the edge: Inference on intermittent embedded systems. *arXiv preprint arXiv:1810.07751*, 2018.

[25] G. Gobieski, B. Lucia, and N. Beckmann. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[26] G. Gobieski, A. Nagi, N. Serafin, M. M. Isgenc, N. Beckmann, and D. Sanchez. MANIC: A Vector-Dataflow Architecture for Ultra-Low-Power Embedded Systems. In *Proc. of the 52nd annual IEEE/ACM intl. symp. on Microarchitecture (MICRO-52)*, 2019.

[27] W. Grosso. *Java RMI.* O'Reilly Media, Inc., 2001.

[28] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of the Twelfth International Conference on Mobile Systems, Applications, and Services*, Bretton Woods, NH, June 2014.

[29] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. The Impact of Mobile Multimedia Applications on Data Center Consolidation. In *Proceedings of the IEEE International Conference on Cloud Engineering*, San Francisco, CA, March 2013.

[30] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proc. of the 37th Annual Intl. Symp. on Computer Architecture*, 2010.

[31] H. Hrimech, L. Alem, and F. Merienne. How 3D Interaction Metaphors Affect User Experience in Collaborative Virtual Environment. *Advances in Human-Computer Interaction*, January 2011.

[32] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan. Quantifying the Impact of Edge Computing on Mobile Applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2016)*, Hong Kong, China, 2016.

[33] M. Khazraee, L. Zhang, L. Vega, and M. B. Taylor. Moonwalk: NRE Optimization in ASIC Clouds or, accelerators will use old silicon. In *Proc. of the 22nd annual ACM intl. conf. on Architecture Support for Programming Languages and Operating Systems (ASPLOS-22)*, 2017.

[34] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European conference on object-oriented programming*, June 1997.

[35] D. Komatitsch, D. Michea, and G. Erlebacher. Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA. *Journal of Parallel and Distributed Computing*, 69(5):451–460, 2009.

[36] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE Intl. Conf. on Information Processing in Sensor Networks*, 2016.

[37] R. Lee, S. I. Venieris, L. Dudziak, S. Bhattacharya, and N. D. Lane. MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors. In *Proceedings of MobiCom 2019*, 2019.

[38] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel. Intermittent Computing: Challenges and Opportunities. In *Proceedings of the 2nd Summit on Advances in Programming Languages*, 2017.

[39] L. E. Lwakatare, T. Kilamo, T. Karvonen, T. Sauvola, V. Heikkila, J. Itkonen, P. Kuvaja, T. Mikkonen, M. Oivo, and C. Lassenius. DevOps in practice: A multiple case study of five companies. 114:217–230, October 2019.

[40] K. Maeng and B. Lucia. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019. ACM, 2019.

[41] M. S. Malone. *The Intel Trinity.* HarperCollins, New York, NY, 2014.

[42] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys2017)*, 2017.

[43] M. Nardello, H. Desai, D. Brunelli, and B. Lucia. Camaroptera: A Batteryless Long-Range Remote Visual Sensing System. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, 2019.

[44] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. of the 16th ACM Symp. on Operating Systems Principles*, 1997.

[45] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam. Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual Intl. Symp. on Computer Architecture*. IEEE, 2017.

[46] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun. Plasticine: A reconfigurable architecture for parallel patterns. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 389–402. IEEE, 2017.

[47] J. Rampton. Crowdsourcing Your Images: 4 Powerful Sites to Connect Directly with Your Audience, April 2015. https://www.forbes.com/sites/johnrampton/2015/04/22/crowdsourcing-your-images-4-powerful-sites-to-connect-directly-with-your-audience/#666536656e74.

[48] E. Ruppel and B. Lucia. Transactional concurrency for intermittent systems. In *Proceedings of the 40 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019. ACM, 2019.

[49] S. K. Sharma and I. Woungang and A. Anpalagan and S. Chatzinotas. Toward Tactile Internet in Beyond 5G Era: Recent Advances, Current Issues, and Future Directions. *IEEE Access*, 8, 2020.

[50] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Trans. on Computer Systems*, 2(4), Nov 1984.

[51] M. Satyanarayanan. Hot Topics: Mobile Computing. *IEEE Computer*, 26(9):81–82, 1993.

[52] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4), 2001.

[53] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), 2009.

[54] M. Satyanarayanan, T. Eiszler, J. Harkes, H. Turki, and Z. Feng. Edge Computing for Legacy Applications. *IEEE Pervasive Computing*, 19(4), October – December 2020.

[55] M. Satyanarayanan, W. Gao, and B. Lucia. The Computing Landscape of the 21st Century. In *Proc. of HotMobile 2019*, Feb 2019.

[56] M. Satyanarayanan, J. H. Howard, D. Nichols, R. N. Sidebotham, A. Z. Spector, and M. J. West. The ITC Distributed File System: Principles and Design. In *Proceedings of the 10th ACM Symposium on Operating System Principles*, December 1985.

[57] M. Satyanarayanan, G. Klas, M. Silva, and S. Mangiante. The Seminal Role of Edge-Native Applications. In *Proc. of the 2019 IEEE Intl. Conf. on Edge Computing*, Milan, Italy, July 2019.

[58] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis. 5G-Enabled Tactile Internet. *IEEE JSAC*, 34(3), 2016.

[59] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan. Towards Scalable Edge-Native Applications. In *Proc. of the Fourth IEEE/ACM Symp. on Edge Computing*, Nov. 2019.

[60] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. Hwu, and D. Chen. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* , November 2018.