SQuadS: Self-Serve System Services for new Hardware-Software Cooperation

Nazerke Turtayeva
UC Santa Barbara
Santa Barbara, CA, USA

Guillem López-Paradís
BSC & UPC
Barcelona, Spain

Jonathan Balkind
UC Santa Barbara
Santa Barbara, CA, USA

ABSTRACT

Overarching computational demand and the well-known end of Moore's law are pushing architects to conceive new solutions resulting in increasing heterogeneity and complexity of hardware. The result is that there is too much for host CPUs to juggle in between accelerators and complex software.

In the face of these persistent trends, we argue that it is time to re-imagine the current Hardware-Software (HW-SW) interface between the Hardware, OS and provided System Services. We argue that traditional HW-SW models no longer serve today's dynamically changing hardware environments. As a new paradigm, we propose Self-Serve System Services (SQuadS) for an efficient and flexible interface. Specifically, this comprises moving the System Services of choice down to HW alongside existing computing units. This enables accelerators to choose between hardware implementations of System Services to Self-Serve or to call the OS from host CPUs as usual. We further argue that circumstances are ideal for such a move to be made in the near future.

1 INTRODUCTION

While accelerators became prevalent as the death knell sounded for Moore's Law, system services have also grown complex. To address the increasing complexity, recent work in our community has focused on building hardware accelerators for system services like Networking [2, 16, 17], Encryption [1, 4], Garbage Collection [13], Memory Allocation [9] and Page Fault Serving [10]. The following are descriptions of some of those works. We highlight in particular the interfaces adopted for these services and accelerators.

Network stack. Traditionally, network stacks operated on host CPUs and smart NICs [2, 5, 12]. With the recent popularity and benefits of hardware acceleration, network stack accelerators have begun to emerge [16, 17]. While applications across the stack can talk via common Network on Chip messages, the interface between the uppermost TCP and user space applications has been mostly effectively linked via shared memory queues [21].

Garbage Collection. There are interesting Garbage Collection implementations both on GPGPUs and as hardware accelerators [3, 8, 13, 14]. With GPGPUs, hosts can offload path-traversal-heavy part of the algorithm to the device, while with accelerators, the entire GC logic can be offloaded. Notably, some GC accelerators use shared memory queues via user space APIs to control heap space [13].

Memory Allocation. Most hardware for Memory Allocation is based on tightly coupled accelerators or microarchitectural changes [9]. What's more, these hardware implementations enable the offloading of only the slow path of the malloc logic because it is hard to beat highly optimized software implementations of the fast path logic. The software versions often use shared memory queues too [11].

1.1 New Opportunities

These days, software stacks are migrating down toward the hardware and contributing to increasing architectural heterogeneity. Accordingly, it is getting harder to manage these systems. It is true that some bespoke HW-SW interfaces exist today. However, there is no well-discussed, documented and universal interface, and we strongly believe that now more than ever, it is crucial to think about novel HW-SW services!

Interesting work to this end has already begun based on microkernels [6, 7, 15, 19] and libOS [20, 21]. They are built as compact control planes on top of accelerators to serve light tasks or to redirect complex requests to host CPUs. The trouble is that today's accelerators mostly operate in the physical address domain, do not have a full view of the system, and are designed with different I/O interfaces that make them harder to integrate into systems-on-chip. Hence, connecting our accelerators to the OS as another standalone CPU-like unit is not possible and would not be beneficial at the moment. That is why microkernels and I/O kernel bypass OSes are very important.

However, these recent innovations of the control plane still preserve the "call-your-host-first" model to serve the system software which is not fully transformative. We argue that adopting microkernels is not sufficient to provide a complete solution and that a new phase of HW-SW cooperation should push forward with a flexible and elastic interface to keep up with innovation in the hardware world.

2 PROPOSED IMPLEMENTATION

We propose Self-Serve System Services (SQuadS) as a hybrid approach to access system services both from hardware

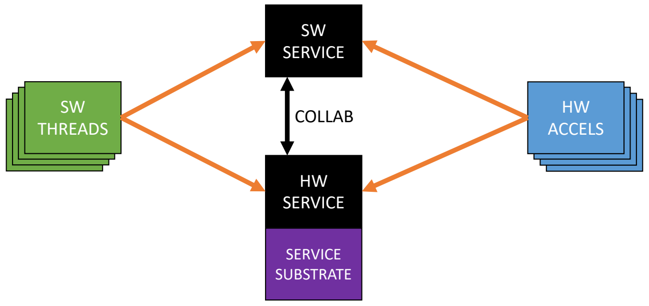

Figure 1: HW-SW co-design with SQuadS

and host software for flexibility, redundancy, and efficiency. To realize this, we argue that it is important to have 1) a common substrate between all System Services, (2) hardware implementations of the System Services and (3) a clear communication mechanism to access those System Services from both applications running on host CPUs or accelerators. Substrates, in turn, should serve as a common block that every System Service accelerator can be based off. Services should also be accessible from applications running both in hardware and in software. The SQuadS model is illustrated in Figure 1. With these features and flexibility in place, we believe that SQuadS will be useful in a variety of contexts, from the edge to the cloud.

As a communication mechanism, we propose to use a classical yet elegant queue-based shared memory protocol between CPUs, datapath accelerators and System Service Substrates. As a widespread paradigm, shared memory queues support efficient programming practices, enabling flexibility in API design for different services' data structures. They can also enable extensive performance benefits in terms of low overhead and efficient use of CPU time for general-purpose cores feeding accelerators, as demonstrated by our recent work, Cohort [18]. Shared memory protocols are also known for scaling well, enabling SQuadS to be useful in a wide range of applications.

Thus, our contributions are (1) a call to the community to discuss a new HW-SW interface for emerging heterogeneous SoCs; (2) a new HW-SW interface itself via SQuadS and (3) an approach to use shared memory protocols as a communication API between mainstream System Services.

Requirements on Accelerators. Our recent work on Cohort [18] demonstrates a couple of aspects needed in a Service Substrate to enable system service acceleration. In order to support communication via shared memory queues, accelerators must have access to applications' virtual memory, but this is not possible with an average accelerator. However, with the addition of a light wrapper (like that provided by Cohort) on top of accelerators, we can enable Virtual Memory translation. Further, for efficient hardware implementation, accelerators will be provided with simple hardware queue interfaces, like latency-insensitive valid/ready streams. These features enable us to generalize the heterogeneous accelerator environment without putting pressure on accelerator designers, making development of Self-Serve System Services more attractive.

SQuadS can also work with both tightly and loosely coupled accelerators. The only constraint SQuadS requires is that accelerators operate on streams of data, utilising input and output streams to communicate with each other. SQuadS also provide necessary interfaces for accessing memory and system services via the substrate.

Requirements on a Substrate. The substrate should contain the addresses of different accelerators and system services and these should be configured via the driver. Then, to manage the accelerator's requests it needs a Cohort-like structure. Also, it is important to have a memory engine for fetching and storing data structures via pointers, as is common for many network and memory system services. We believe that these baseline blocks are sufficient as a substrate, but encourage community discussion on this interesting point. Our ultimate view is that, with a careful choice of feature set, SQuadS will become a new, flexible and useful architectural component to enable tightly integrated HW-SW co-design.

Accordingly, with SQuadS, accelerators may request services with lower contention and context switch overhead than traditional software-based services. This open doors for greater parallelism and overall system efficiency. Therefore, SQuadS is important not only to enable a universal and flexible interface between the processors, accelerators, and system services, but also to push to the next level of performance and efficiency. Thorough evaluation and discussion will be developed in our future work.

3 CONCLUSION

Ultimately, with this work, we are aiming for a future where HW-SW interfaces fully catch up with the recent, immense growth in hardware heterogeneity. To solve this problem and to foster new discussion in the direction of HW-SW co-design, we propose Self Serve System Services (SQuadS) which can migrate and co-exist across both accelerators and CPUs. With this idea, we can transform how we view operating systems and hardware today to come up with better performing, more efficient, scalable, and cheaper designs.

REFERENCES

[1] Salah Amr. 2013. AES-128 Encryption Core. OpenCores AES-128 Pipelined Encryption Implementation in Verilog , https://opencores.org/projects/aes-128_pipelined_encryption.

[2] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. 2022. HXDP: Efficient Software Packet Processing on FPGA NICs. *Commun. ACM* 65, 8 (jul 2022), 92–100. https://doi.org/10.1145/3543668

[3] Chen-Yong Cher and Michael Gschwind. 2008. Cell GC: Using the Cell Synergistic Processor as a Garbage Collection Coprocessor. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Seattle, WA, USA) *(VEE '08)*. Association for Computing Machinery, New York, NY, USA, 141–150. https://doi.org/10.1145/1346256.1346276

[4] Jonny Doin. 2016. SHA256 Hash Core. OpenCores SHA256 implementation in VHDL, https://opencores.org/projects/sha256_hash_core.

[5] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation* (Renton, WA, USA) *(NSDI'18)*. USENIX Association, USA, 51–64.

[6] Matthias Hille, Nils Asmussen, Hermann Härtig, and Pramod Bhatotia. 2020. A Heterogeneous Microkernel OS for Rack-Scale Systems *(APSys '20)*. Association for Computing Machinery, New York, NY, USA, 50–58. https://doi.org/10.1145/3409963.3410487

[7] Matthias Hille, Nils Asmussen, Hermann Härtig, and Pramod Bhatotia. 2020. A Heterogeneous Microkernel OS for Rack-Scale Systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems* (Tsukuba, Japan) *(APSys '20)*. Association for Computing Machinery, New York, NY, USA, 50–58. https://doi.org/10.1145/3409963.3410487

[8] José A. Joao, Onur Mutlu, and Yale N. Patt. 2009. Flexible Reference-Counting-Based Hardware Acceleration for Garbage Collection. *SIGARCH Comput. Archit. News* 37, 3 (jun 2009), 418–428. https://doi.org/10.1145/1555815.1555806

[9] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. 2017. Mallacc: Accelerating Memory Allocation. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 33–45. https://doi.org/10.1145/3037697.3037736

[10] Gyusun Lee, Wenjing Jin, Wonsuk Song, Jeonghun Gong, Jonghyun Bae, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2020. A Case for Hardware-Based Demand Paging. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 1103–1116. https://doi.org/10.1109/ISCA45697.2020.00093

[11] Paul Liétar, Theodore Butler, Sylvan Clebsch, Sophia Drossopoulou, Juliana Franco, Matthew J. Parkinson, Alex Shamis, Christoph M. Wintersteiger, and David Chisnall. 2019. Snmalloc: A Message Passing Allocator. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on Memory Management* (Phoenix, AZ, USA) *(ISMM 2019)*. Association for Computing Machinery, New York, NY, USA, 122–135. https://doi.org/10.1145/3315573.3329980

[12] Jiaxin Lin, Kiran Patel, Brent Stephens, Anirudh Sivaraman, and Aditya Akella. 2019. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. USENIX Association, 243–259.

[13] Martin Maas, Krste Asanovic, and John Kubiatowicz. 2019. A Hardware Accelerator for Tracing Garbage Collection. *IEEE Micro* 39, 3 (2019), 38–46. https://doi.org/10.1109/MM.2019.2910509

[14] Martin Maas, Philip Reames, Jeffrey Morlan, Krste Asanović, Anthony D. Joseph, and John Kubiatowicz. 2012. GPUs as an Opportunity for Offloading Garbage Collection. *SIGPLAN Not.* 47, 11 (jun 2012), 25–36. https://doi.org/10.1145/2426642.2259002

[15] Changwoo Min, Woonhak Kang, Mohan Kumar, Sanidhya Kashyap, Steffen Maass, Heeseung Jo, and Taesoo Kim. 2018. Solros: A Data-Centric Operating System Architecture for Heterogeneous Computing. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 36, 15 pages. https://doi.org/10.1145/3190508.3190523

[16] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA) *(ISCA '14)*. IEEE Press, 13–24.

[17] Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso, and Sergio López-Buedo. 2019. Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack. https://doi.org/10.1109/FPL.2019.00053

[18] Tianrui Wei, Nazerke Turtayeva, Marcelo Orenes-Vera, Omkar Lomkar, and Jonathan Balkind. 2023. Cohort: Software-Oriented Acceleration for Heterogeneous SoCs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '23)*. Association for Computing Machinery, Vancouver, Canada, 6 pages. https://doi.org/xxxx

[19] Gerd Zellweger, Simon Gerber, Kornilios Kourtis, and Timothy Roscoe. 2014. Decoupling cores, kernels, and operating systems. USENIX Association, 17–31. https://doi.org/10.5555/2685048.2685051

[20] Irene Zhang, Jing Liu, Amanda Austin, Michael Lowell Roberts, and Anirudh Badam. 2019. I'm Not Dead Yet! The Role of the Operating System in a Kernel-Bypass Era. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 73–80. https://doi.org/10.1145/3317550.3321422

[21] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. 2021. The Demikernel Datapath OS Architecture for Microsecond-Scale Datacenter Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 195–211. https://doi.org/10.1145/3477132.3483569