We had 64-bit, yes. What about second 64-bit?

Mathieu Bacou[1], Adam Chader[1], Chandana Deshpande[2], Christian Fabre[3], César Fuguet[3], Pierre Michaud[4], Arthur Perais[2], Frédéric Pétrot[2], Gaël Thomas[1], Eduardo Tomasi[2,3]*

[1]Télécom SudParis [2]Univ. Grenoble Alpes, CNRS, Grenoble INP†, TIMA [3]Univ. Grenoble Alpes, CEA, List [4]Inria, Univ Rennes, CNRS, IRISA

Abstract

High-performance architectures are increasingly heterogeneous and incorporate often specialized hardware. We have first seen the generalization of GPUs in the most powerful machines, followed by FPGAs, and now by many other accelerators such as Tensor Processor Units (TPUs) for Deep Neural Networks, or variable precision FPUs. Recent hardware manufacturing trends make it very likely that specialization will not only persist, but increase. Manually managing this heterogeneity is complex and not maintainable. We therefore propose to revisit how we design both hardware and OS in order to better hide the heterogeneity. To ensure long term viability of our proposal, we propose to entertain the use of 128-bit addressing.

1 Introduction & Motivation

Cloud and high-performance infrastructures have evolved to become distributed and heterogeneous systems: one machine now includes several CPU sockets interconnected to multiple memory banks through complex Non-Uniform Memory Access (NUMA) networks; and each socket is itself a complex system with an integrated network-on-chip interconnecting the cores and many levels of caches. In addition, the storage system is a complex stack of layers with variable performance characteristics and features. To top it all off, each computer contains heterogeneous accelerators dedicated to specific workloads: GPUs, FPGAs, SmartNICs, TPUs, Variable Precision (VRP) FPUs, etc.

Meanwhile, the software system part remains a stack of competing layers, each based on tweaked generic heuristics and concepts, that try to manage the same hardware resources and system objects. For example, data-centers use a virtualization stack made of a Linux kernel-based virtual machine (VM), on top of a QEMU/KVM (Linux kernel)-based hypervisor, all in an attempt to handle the distribution and the heterogeneity. To handle the hardware and software diversity, Linux has grown to 20 millions lines of code ; and despite this large code base, the kernel does not provide adequate abstractions for modern hardware. As a result, many libraries turn to bypassing the operating system (OS), for performance reasons and to provide more adequate abstractions to the users. We came to the conclusion that the situation is not sustainable in the long term, so we propose to explore the redesign of the OS and of its hardware support in light of modern devices and concepts.

Our redesign is best illustrated by the following example: in the context of a Deep Neural Network (DNN) application, a learning process must: 1. load training data from disk to central memory, directed by the CPU; 2. load training data from central memory to GPU memory for model learning, directed by the CPU; 3. load learned model from GPU memory to central memory, directed by the CPU; 4. load trained model from central memory to network card queues for model distribution, directed by the CPU.

In this scenario, the CPU is always involved to control the data movements, despite never being useful to the actual process taking place; and there are two indirect data movements. This is explained by the heterogeneity of the devices and accelerators and by their non integration into a disaggregated system with a unified memory space.

By contrast, we want to achieve the following scenario: 1. load training data from disk to GPU memory for model learning, directed by the GPU; 2. load learned model from GPU memory to network card queues for model distribution, directed by the GPU. The first step is to unify the hardware, and the second is to revisit how it is exposed by the OS.

The unification of the hardware stands on two pillars: a common minimal instruction set, and a unified address space. The common minimal instruction set is made possible by RISC-V and its ability to support different extensions in different cores. The unified address space is a straightforward idea: to expose any device (from general purpose processors to specialized processing units, through memory banks and NICs, etc.) in the virtual address space of the process. Towards that endeavor, we propose the use of 128-bit virtual addresses associated with machine-wide cache-coherency protocols such as CXL [1].

Based on the unification of the hardware, we propose to revisit the design of the OS to handle the distribution and heterogeneity of the machine to expose its resources to the processes. A multi-kernel design allows to distributed the OS, where devices run satellite kernels collaborating with each other to execute processes. Then, we want to hide the heterogeneity of the devices' interfaces by generalizing the kernel bypass approach. By leveraging the unified 128-bit address space, we install the OS as a controller that grants access to hardware resources to its processes by exposing the devices directly in their address spaces.

To achieve our collaborative hardware-software redesign through 128-bit addressing, we will tackle the following challenges: i) Tame the performance penalty induced by

*Alphabetical order. This work is funded by ANR project *Maplurinum*.
Corresponding author: arthur.perais@univ-grenoble-alpes.fr
†Institute of Engineering Univ. Grenoble Alpes

the transition to 128-bit; ii) Implement efficient, scalable hardware communication primitives through the unified address space; iii) Build the satellite multi-kernel as an efficient control plane; iv) Design kernel-to-process interfaces to provide a performing data plane.

2 Hardware Perspective

Unifying the address space is a good opportunity to also consider 128-bit addresses, as it is bound to put additional pressure on address availability. However, the impacts of doing so on hardware must be considered.

Efficient 128-bit Microarchitecture Naively doubling the width of datapaths, registers and functional units will incur power, area and latency overheads. Combined to the doubled footprint of pointers in memory, the transition to 128-bit is likely to undermine performance rather than improve it, at least from the processor perspective. Our intuition is that most 128-bit computation will be address calculation rather than "regular" integer arithmetic. Therefore, we envision a clustered microarchitecture with a 64-bit cluster where 32- and 64-bit integer arithmetic will be performed, and a 128-bit cluster where address calculation and 128-bit arithmetic will be performed. This divide & conquer approach will allow to expose full 128-bit support to software while paying for the wider datapath only in part of the processor backend. A key difficulty will be to steer 32- and 64-bit instructions to the 128-bit cluster if they do participate in address calculation, as inter-cluster communication generally incurs latency [2].

Memory Hierarchy Dealing with the NUMA effects is one of the most important challenges with 128-bit addressable space. Indeed, highly heterogeneous supercomputers and datacenters will implement multi-level cache-hierarchies with multiple distributed memory banks at the chip level to increase memory bandwidth. These memory banks are physically distributed but logically shared, that is, any device in the system can access any memory bank. However, the access latency depends on the physical distance between the device and the target memory bank, which is exacerbated with multi-socket and multi-board systems. A possible solution for reducing NUMA effects is to place code and data near the device that uses them. However, correctly placing memory objects is hard when code and data memory objects are shared by multiple devices that are physically far from each other. The multi-kernel OS can replicate code and data segments and put copies near the devices. This leads to yet another issue: the management of the copies. Data coherency and consistency must be guaranteed in case of multi-write and multi-read data. For example, CXL protocols ensure cache coherency between a machine's devices, enabling extremely efficient direct accesses to memory-mapped registers. We envision OS-driven hardware mechanisms to ease both the replication and management of copies, for example by allowing the OS to dynamically define the cache coherency between devices.

3 Software & System Perspective

The operating system needs a redesign to handle the challenges of heterogeneity and disaggregation of the rack.

We aim at solving this challenge by redesigning the OS as a multi-kernel, centered around a unified address space. The redesign is supported first by RISC-V as a common minimal instruction set, and second but most importantly by a 128-bit unified address space [3].

Multi-kernel A multi-kernel is a distributed system: each processing unit runs a kernel, and they collaborate to execute processes that run on the different units. Multi-kernels have already been proposed [4, 5], but they only took into account plain homogeneous CPUs. They also limited themselves to one communication paradigm via message passing.

To counter this, we propose to revisit the concept of satellites: each heterogeneous device and accelerator is equipped with an optimized CPU solely dedicated to the operations of the control plane. By doing so, devices and accelerators can be seen as homogeneous processing units: they become active in the disaggregated system. The common minimal instruction set and hardware virtualization will help in implementing satellite kernels that communicate efficiently via a unified 128-bits address space.

Unified address space In the context of a disaggregated rack, the multi-kernel is a controller that grants access to different hardware resources to user tasks. We want to revisit the classic kernel interfaces and abstractions around the idea of direct access to the data plane, i.e., the memory. Indeed, all satellites will give out grants for other satellites to directly map some control structure and data buffers into their memory. This design is in line with the bypassing of the control plane in favor of optimizing data plane-related operations. This is also a way to homogenize the low-level interfaces of the kernel, all through a unified address space.

References

[1] CXL Consortium. *Compute Express Link: The Breakthrough CPU-to-Device Interconnect*. `https://www.computeexpresslink.org/` (visited: 2023-03-17). 2020.

[2] Richard E Kessler. "The alpha 21264 microprocessor". In: *IEEE micro* 19.2 (1999), pp. 24–36.

[3] Andrew Waterman et al. *The RISC-V instruction set manual, volume I: User-level ISA, Version 2.0*. Tech. rep. `https://github.com/riscv/riscv-isa-manual/releases/tag/isa-449cd0c`. 2023, pp. 49–50.

[4] Andrew Baumann et al. "The Multikernel: A New OS Architecture for Scalable Multicore Systems". In: *Proc. of the Symp. on Operating Systems Principles*, pp. 29–44.

[5] Edmund B. Nightingale et al. "Helios: Heterogeneous Multiprocessing with Satellite Kernels". In: *Proc. of the Symp. on Operating Systems Principles*, pp. 221–234.