Provable Correct Memory Management

Reto Achermann, Timothy Roscoe
{reto.achermann, troscoe}@inf.ethz.ch
Systems Group, Dept. of Computer Science, ETH Zurich

March 16, 2017

Hardware forms an increasingly complex network that is diverse across different platforms. Memory accesses by a core or DMA engines traverse this network and undergo various translation and caching steps. Two cores see memory or devices at different addresses and the same address may refer to different things. This complexity needs to be correctly understood and configured by system software at runtime. We express the hardware configuration of a system in a formal model that allows us to reason about address spaces and resources for provable correct memory management and allocation.

1 Introduction

Modern systems are complex networks of cores, caches, memory, devices and translation units with virtualization support. Yet, textbooks present an overly simplified abstraction: A processor with MMU translating the virtual address to unique physical address or a fault. This representation is not only unsuitable as a basis for writing correct system software, in fact it's just plain wrong.

Moreover, a single physical address space is an illusion [4]: memory addresses are routed between multiple address spaces, often involving translating the address. To exemplify the challenge we address, consider the Texas Instruments OMAP4460 Multimedia SoC [8]: certain devices appear at different physical addresses, depending on which of the cores accesses the device – in particular the general purpose timer device has at least six different (physical) addresses it responds to. Programmable firewalls restrict subsystems access to other parts of the SoC. Based on this example, we can conclude that the view of the system is different from each core or device. This is not just limited to SoC, also desktop computers, servers and rack-scale systems show similar properties.

Writing correct system software is hard, it requires a thorough understanding of the system's hardware configuration. Memory management and resource allocation needs to deal with ambiguous physical addresses and limited reachability of resources. System software must *correctly* manage and program address translations on a broad range of system architectures.

We propose a formal framework for the interpretation of memory accesses (loads, stores, DMA operations, etc.) of a modern computer system which captures the relevant features of contemporary hardware [1]. The model gives an unambiguous interpretation of memory accesses, its applications include a foundation for system software verification, identifying problematic hardware designs, and generating correct-by-construction OS code. Our model is a decoding net, a directed graph consisting of nodes with two properties: *map* and *accept* where nodes represent hardware components (cores, memory controllers, MMUs, etc.). Each component either accepts a set of addresses (e.g. DRAM, device registers) or maps a set of addresses onto another node. We show that this abstraction is close to hardware by expressing various systems, from SoC to server cluster, in our model. We are able to proof the absence of loops and view equivalence of two representations using an implementation in the Isabelle/HOL theorem prover [1]. In contrast to other work [2, 3], our work focuses on the reachability of resources and decoding of addresses rather than memory models.

While developing the model, Humbel [5] discovered the applicability of the same model to the problem of interrupt routing and delivery by treating interrupt controllers as address spaces which forward (map) or are the destination of interrupts (accept).

2 Memory management

Our goal is to build a provable correct memory management system capable of dealing with a di-

verse set of hardware configurations. We want to guarantee a set of invariants such as a process can only access resources it has been granted access to or two cores do not interfere with each other (sharing memory controllers or cache pollution). We want to be able to make strong statements about isolation and performance characteristics. Currently, our model is able to express whether or not a resource can be accessed from a particular processor i.e. which names decode to it. To achieve this, we have to extend the model in the following four ways.

Refinement of the map property Hardware, in particular the (IO)MMU, distinguishes between reads and writes operations and further does not always allow translations or accesses at byte granularity. We therefore need to extend our model to express the direction of data flow between the nodes. Both properties, map and accept, of a node in the decoding net need to specify whether it can be decoded for a read and/or write access. The model should be able to decode reads and writes using the same name differently. In addition, constraints on the map and accept properties are required to enforce the translation and access granularities.

Performance characteristics Hardware has different access and performance characteristics depending on how and from where it is accessed. For instance, DRAM (memory controller) has a certain latency and a maximum bandwidth and memory access is faster locally compared to a remote socket. In addition, read and write performance may be different. Knowing the performance characteristics of hardware is required for smart allocations of memory [7] and building efficient message passing topologies [6]. To do smart resource allocation and topology construction, we plan to extend the model to include cost and capacity metrics for the map and accept properties of the nodes. We can express the read-only property of above by setting the latency to infinity and bandwidth to zero for the write part.

Isolation Properties In order to meet SLA requirements, two workloads (e.g. processes or virtual machines) must not interfere with each other or compete for shared resources. Assigning a core to a workload exclusively is not enough as memory controllers or interconnect links may still be accessed by the two workloads concurrently. We need to formulate and proof the isolation property on top of our model and show that a particular system configuration satisfies it.

Provable Correct Memory Management From the start of the system, there is a sequence of configuration states:

$$C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_t$$

Each state C_i must be correct with regard to memory allocations, protection and translations. At reset, the system starts at a well-known state C_0. We want to have the property that if the system is in a valid state C_t, the model gives us a transition function which moves the system into another valid configuration C_{t+1}. Ideally, we are able to generate system code that does this transition and formally verify its correctness.

We plan to implement this approach in the Barrelfish operating system. We think that the formal model is a natural match for Barrelfish's capability system that provides the basis for resource management and allows custom allocation policies.

References

[1] Achermann, R., Humbel, L., Cock, D., and Roscoe, T. Formalizing address spaces and interrupts. In Submission to 2nd Workshop on Models for Formal Analysis of Real Systems (MARS 2017), April 2017.

[2] Alglave, J., Maranget, L., Sarkar, S., and Sewell, P. Fences in Weak Memory Models. In *CAV'10* (2010), Springer-Verlag, pp. 258–272.

[3] Flur, S., Gray, K. E., Pulte, C., Sarkar, S., Sezgin, A., Maranget, L., Deacon, W., and Sewell, P. Modelling the ARMv8 Architecture, Operationally: Concurrency and ISA. In *POPL '16* (2016), pp. 608–621.

[4] Gerber, S., Zellweger, G., Achermann, R., Kourtis, K., Roscoe, T., and Milojicic, D. Not Your Parents' Physical Address Space. In *HOTOS'15* (Switzerland, 2015), USENIX Association, pp. 16–16.

[5] Humbel, L. Formalizing address spaces and interrupts. In Submission to11th EuroSys Doctoral Workshop (EuroDW 2017), April 2017.

[6] Kaestle, S., Achermann, R., Haecki, R., Hoffmann, M., Ramos, S., and Roscoe, T. Machine-aware atomic broadcast trees for multicores. In *OSDI'16* (2016), pp. 33–48.

[7] Kaestle, S., Achermann, R., Roscoe, T., and Harris, T. Shoal: Smart Allocation and Replication of Memory for Parallel Programs. In *USENIX ATC '15* (2015), pp. 263–276.

[8] Texas Instruments. *OMAP44xx Multimedia Device Technical Reference Manual*, April 2014. Version AB, `www.ti.com/lit/ug/swpu235ab/swpu235ab.pdf`.