SCIMITAR
A Scalable I/O and Memory Pinning and Translation Architecture

Muli Ben-Yehuda[1], Zorik Machulsky[1], Leah Shalev[1], Julian Satran[1]
Ben-Ami Yassour[1], Orit Wasserman[1], Michael Vasiliev[1]
Yaron Weinsberg[2], Emil Elazar[3]

[1]{muli,machuslk,leah,satran,benami,oritw,michaelv}@il.ibm.com
IBM Haifa Research Lab
[2]yaron.weinsberg@microsoft.com
Microsoft Corporation
[3]emil.elazar@mail.huji.ac.il
Hebrew University of Jerusalem

It is hard to imagine general purpose computer systems without support for virtual memory translation. Virtual memory has many virtues, but most importantly, it makes it possible to efficiently share an extremely sparse resource—physical memory—among multiple consumers (such as operating-system processes or virtual machines), without requiring any knowledge or involvement of those consumers. It was a great invention, simultaneously making the consumers lives simpler and the system as a whole more efficient.

There is one central aspect of our computer systems, however, which was left behind when virtual memory was invented: the I/O peripheral subsystem. Although in today's systems most I/O operations are still initiated and controlled by a privileged software intermediary (e.g., operating-system kernel or hypervisor) there are systems where I/O devices are accessed directly by consumers, bypassing the privileged software intermediary. In such systems low latency and CPU utilization, as well as high throughput, are crucial, and "direct I/O" (also known as "OS-bypass I/O") is the means usually employed to achieve them. Examples of such systems include high-performance Infiniband clusters and servers running I/O-intensive workloads in virtual machines.

Although in theory there should be no fundamental difference between a "move to memory" instruction executed by a consumer on a CPU and a "DMA to memory" I/O operation executed by a device on behalf of the same consumer. However, in practice, a "move to memory" instruction goes through virtual memory translation, while a "DMA to memory" operation usually accesses physical memory directly. In order to synchronize these physical memory accesses between CPUs and devices, consumers need to take extra care when initiating and completing I/O operations. First, a consumer must "register" or "pin" the portion of memory it is about to access. Pinning locks a shared resource–physical memory–for exclusive access by that consumer, for an unbounded amount of time. Second, once the consumer is done using that portion of physical memory, the consumers must "unregister" or "unpin" the memory and release it back to the system. Pinning wastes both CPU cycles and memory, the former since it requires context-switching into privileged software (such as an OS or a hypervisor), and the former since once a portion of physical memory is pinned it is no longer available for the rest of the system. In addition to being wasteful, it is also complex to use and places an additional burden on the consumer's developers.

The SCIMITAR project is a research exploration of what happens when theory and practice meet and we eliminate the differences between CPU and I/O memory accesses. What happens if every memory access by an I/O adapter went through the same virtual memory translation mechanisms as memory accesses done by the CPU?

The SCIMITAR project began with the observation that the majority of memory pages pinned for I/O are likely to be already present in the page tables of the consumer initiating the I/O, because they have been accessed recently. This implies that pinning them explicitly, as is done today, is wasteful in the majority of cases. This up-front pinning of memory is done because there is no way for I/O devices to handle the rare case where those memory pages are **not** present. This makes the common case–memory pages present–slow, but is needed because today's I/O devices cannot handle the rare case of memory pages not present. Therefore, SCIMITAR dispenses with memory registration completely, making the common case fast. A consumer initiates an I/O operation at the device **without** going through the privileged software first, using the consumer's virtual memory address instead of a physical address. The device, either directly or via privileged software, walks the consumer's page tables in order to find whether the pages are present and to perform the necessary virtual to physical translation. Assuming they are present, it goes ahead performs the DMA operation.

This radical approach immediately raises several questions, such as what happens if the pages which are target for the I/O are not present? In order to deal with this rare occurrence, SCIMITAR introduces the notion of an *I/O page fault.* Similar to a CPU page fault which is raised when a consumer tries to access a not-present virtual memory page, an I/O page fault is raised when an I/O device tries to access a non-present memory page on behalf of a consumer. Privileged software is again responsible for handling the page fault and restarting the I/O access.

Once the notions of "I/O page fault" and "I/O through virtual memory addresses" are introduced, the I/O device can be considered as just another computational core running code which sometimes accesses memory. This implies that the I/O device needs to participate in the host's TLB synchronization protocols, despite being relatively far from the other cores and not running any of the privileged software's code. How to do this safely and efficiently is one of the key questions SCIMITAR is exploring.

With the introduction of isolation-capable IOMMUs on commodity servers (e.g., Intel's VT-d and AMD's IOV), the industry has taken a step toward virtual memory for I/O. However, these IOMMUs only support a per-device address space, whereas SCIMITAR takes things a step further by unifying the CPU and I/O address spaces. SCIMITAR uses either the consumer's page tables or an exact, synchronized copy of them for I/O. In addition, we note that neither Intel's VT-d nor AMD's IOV solve the pinning problem, since they do not have an I/O page fault mechanism and cannot recover from an attempt to access a not-present page.

In order to explore the questions posed by SCIMITAR and validate our initial assumptions, we are implementing a prototype system. The prototype system includes I/O devices capable of page table walks and raising I/O page faults and privileged software capable of handling I/O page faults and synchronizing page table updates with the I/O devices. The prototype system is nearing completion and we expect initial results shortly.

Some of the questions posed by SCIMITAR include:

(a) How does one build an extremely asymmetric system where some of the cores are near and some are far from memory and from each other??

(b) What sort of temporal and/or spatial locality is exhibited by the I/O operations of different workloads?

(c) What sort of operating system changes are needed to support I/O page faults? What is the correct split of responsibility between devices and privileged software, e.g., who does page table walks?

(d) How does one build I/O devices and protocols when some memory accesses will take orders of magnitude longer than other operations, or may even fail out-right?

(e) What is the right way to build a device IOTLB?

In summary, SCIMITAR is a research exploration of virtual memory for I/O devices. SCIMITAR opens up a host of new research directions. It is simultaneously a radical departure from today's system architectures and a logical conclusion of several current trends. It poses many interesting questions that we look forward to answering together with the larger research community.