Self-Learning, Predictive Computer Systems

ICRI-CI Highlights Report, Jun. 2013

Project: **Memory Prefetching for Task-Based Programming Models**
Faculty: Prof. Yoav Etsion, Prof. Shie Mannor and Prof. Uri Weiser
Student: Adi Fuchs

Description: Existing memory prefetchers infer access patterns by tracking individual memory accesses. However, emerging task-based programming models decompose applications into encapsulated sequential tasks that execute recurring code fragments as individual work units. Task-based models thus coarsen the granularity of a semantic work-unit – from a single instruction to an entire task. This coarsening provides an opportunity for higher-level learning of memory access patterns, where patterns can be inferred by tracking entire tasks rather than single memory accesses.

Status: We are currently studying task-centric memory access patterns for several task-based parallel benchmarks and are identifying interestingly predictable memory access patterns.

Our current study records heap-based memory traces for entire tasks. One striking pattern is that of *task differentials vectors*. We define a memory trace of task A as the sequence of heap-based memory addresses:

$$T_A = \{a_1, a_2 ... a_n\}$$

We then define a *task differential vector* for two consecutive tasks A and B as the aligned sequence of memory addresses differences across both traces:

$$\Delta_{AB} = \{\delta_i \mid \delta_i = b_i - a_i \ \ for\ each\ i\}$$

We then examine the correlation between the total number of ***unique*** task differential vectors versus the total number of ***instances*** of task differential vectors. The latter correlates to the total number of task instances, as every two consecutive tasks A and B generate an instance of a (possibly recurring) task differential vector.

Figure 1 plots the correlation between the distribution of unique task differential vectors (horizontal axis) and the total number of vector (or task) instances. The extremely non-uniform correlation suggests that a tiny fraction of the unique vector values can be used to predict the

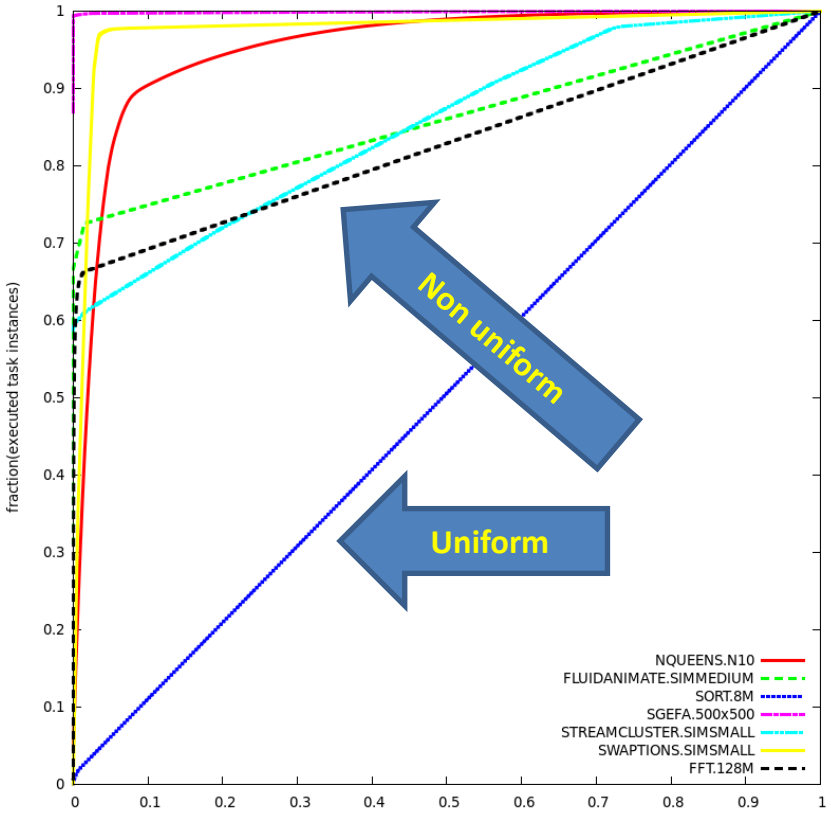

Figure 1. The correlation between the distribution of task references streams and that of the differential vectors. The non-uniform distribution, observed in most benchmarks, suggest that the vast majority of memory references initiated by tasks can be predicted using only a tiny fraction of differential vectors.

memory access for most tasks. For example, the figure shows that in the case of the FFT benchmark (dashed black line in the figure), which is part of the Barcelona OpenMP Task Suite (BOTS), roughly 2% of the unique differential vectors can predict the entire memory access traces for 65% of the tasks.

Task differential vectors, therefore, provide a promising vehicle for predicting memory access traces for entire tasks and, in general, demonstrate the importance of predicting memory accesses across tasks. We are currently engaged in developing a task-based hardware prefetcher based on the task-differential scheme.

Project: **Rethinking the IOMMU**

Faculty: Prof. Dan Tsafrir
Students: Moshe Malka, Nadav Amit, Muli Ben-Yehuda

Description: The IOMMU (I/O memory management unit) is to an I/O device, what MMU is to a process. Namely, both MMU and IOMMU provide a level of indirection: the MMU for when the process accesses the memory, and the IOMMU for when the device performs a DMA. Both MMU and IOMMU are implemented similarly, using a radix-tree-like hierarchical structure of page tables that the operating system maintains and the hardware "walks" upon an (IO)TLB miss.

The goal of this project is to explore whether the decision to design the IOMMU similarly to the MMU makes sense and, in the case it does not, to come up with an alternative, better IOMMU design.

Status: We have constructed an experimental environment of the Intel virtualization technology for direct I/O (VT-d) on top Linux/KVM/QEMU. The experimental setup will allow us to log, profile, analyze, and study the DMA activity that the IOMMU services. We have extended QEMU to support IOMMU emulation, and our modified QEMU version now provides Intel's VT-d functionality. Our experimental environment is now functional and operational.

We have identified several canonical I/O-intensive benchmarks with which we conduct our study: (1) the Netperf network profiler, (2) the memcached distributed object caching system, and (3) the Apache web server. We have executed these workloads on top of our experimental setup and have produced logs that characterize all activity experienced by the IOMMU. Extensive study of the logs reveals a usage pattern that is fundamentally different than that which is typical to MMUs. Importantly, and somewhat surprisingly, we have learned that the IOMMU activity is (nearly) completely predictable, but only when taking into account the manner by which I/O device drivers typically function. In particular, we attribute the root cause of the predictability to the pervasive use of "ring buffers" for driving I/O devices (see Figure 2), and to the fact that these ring buffers are utilized in a cyclic manner. Our next step is to exploit these understandings and to designan IOMMU that makes use of this predictability.

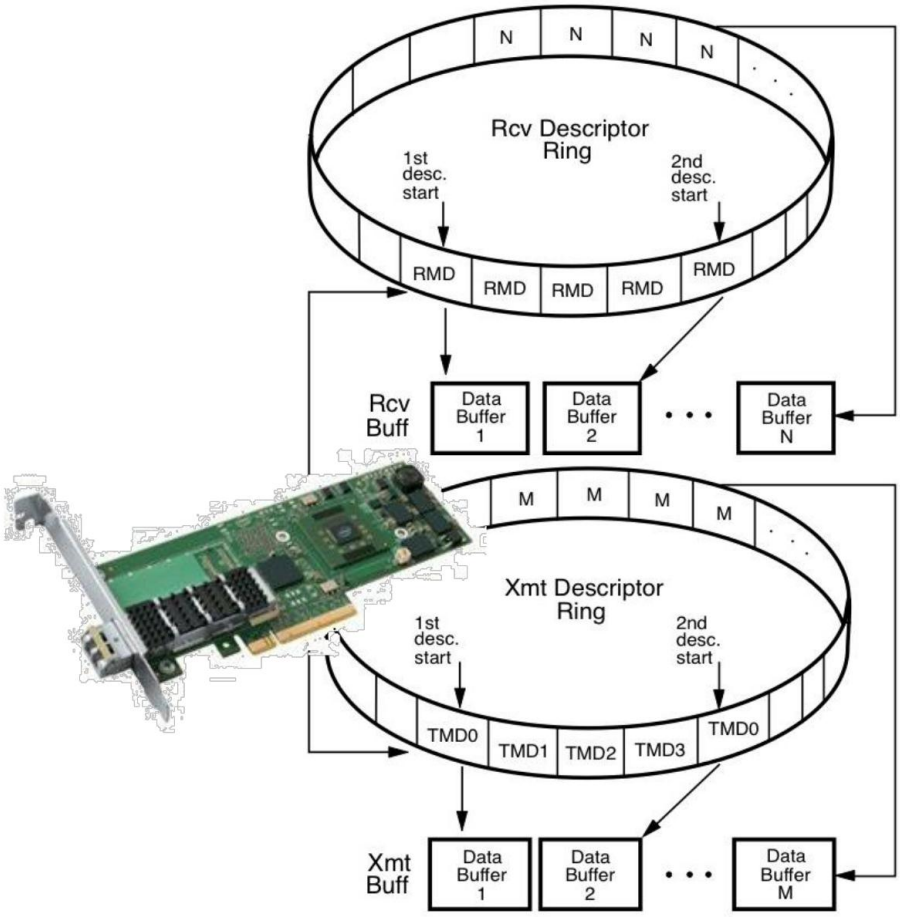

Figure 2. Device drivers typically drive the activity of their associated I/O devices through ring buffers; if the semantics of these buffers are known, then the access order becomes predictable.