Software-Defined Physical Memory: Putting The OS in Control of DRAM

Marius Hillenbrand

OS Group – Karlsruhe Institute of Technology

os@itec.kit.edu

Dynamic Random Access Memory (DRAM) is the ubiquitous technology for main memory today. While that name still suggests identical access times independent of location, the latencies and throughput observed in contemporary DDR3 and DDR4 DRAM memory systems depend on access patterns, however: Address locality, request ordering, and the relation of subsequent requests' addresses can change the achieved bandwidth by more than an order of magnitude.

On the one hand, achieving high single-thread performance requires (a) exploiting the parallelism in the DRAM structure by overlapping the processing of requests, and (b) avoiding bank conflicts, which can multiply the service time for individual requests by ˜3x. On the other hand, when concurrent tasks run on multicore systems, they compete for the shared resources in the common DRAM subsystem. Requests of different tasks then interfere because of rate limits, such as maximum command and data rates of memory busses, and penalties for conflicting accesses, such as high latencies for bank conflicts. This interference can both degrade system throughput and reduce individual tasks's performance unevenely (depending on application's characteristics).

Several techniques have been proposed to both improve single-thread performance and cope with interference in DRAM-based memory systems. Yet today, operating systems (OSs) are largely agnostic to DRAM structure. Page allocation is mostly concernced with *space*, considering all page frames in physical memory as equal. We argue that an OS should manage DRAM-based main memory as a *processing resource* in addition to its spatial character. Main memory subsystems form complex hierarchies of shared and parallel resources, of which each physical page frame covers a subset: The physical frame number defines which DRAM channels, ranks, and banks the physical addresses within that page frame can address. Thus, each page allocation decision is unavoidably an admission decision, in addition to a spatial allocation. As mediating access to shared resources forms a core task of an OS, we are convinced that operating system research should be aware of this additional facet in memory management. An OS should take explicit control of this admission process.

For informed placement decisions, the OS first requires knowledge about how physical addresses map to the underlying DRAM structure. To make such designs practical, we propose a standardized firmware interface that describes the system-specific mapping. Today, retrieving this information for the OS requires repeated development effort for each new platform, with varying amounts of reverse-engineering—a viable option only for research prototypes.

Previous designs for paging-based performance isolation required that the OS fully controls the mapping of virtual memory addresses to the physical entities of DRAM memory, such as channels and banks: Channel and bank addresses need to be controlled by high-order address bits, outside the page offset, so that the OS can define which page gets to reside on which channel by assigning appropriate physical frames. In contrast, *channel and bank interleaving* improves application performance by mapping channel and bank to low-order address bits. As a consequence, systems can either support OS-controlled DRAM performance isolation or provide high memory parallelism to applications, but not both at the same time.

We propose to overcome this limitation by introducing *mapping aliases*: A system's DRAM memory appears in the physical address space as *multiple regions*, or *aliases*. Each of these *aliases* employs a different scheme in mapping physical addresses to DRAM addresses (i.e., channel, rank, bank, row, and column). By choosing the right *alias* when allocating memory, the OS can vary how much control it keeps for page allocation (i.e., in high-order address bits) and how much memory parallelism it exposes to applications (i.e., in low-order bits) on a page by page basis, dynamically at run time. For example, it could provide one application with two-way channel interleaving and

dedicate another memory channel to a second application for performance isolation.

We present a prototype of *mapping aliases* on *off-the-shelf* hardware, which supports configuring aliases *out of the box*. Further, we argue that *mapping aliases* could be supported on a wide range of production hardware platforms with minimum modifications. Our approach is orthogonal to page-based virtual memory, because it applies to the translation from physical addresses to DRAM addresses. Thus, they can be applied independent of page size.

While providing new opportunities, our proposal raises new challenges for memory management because it invalidates two key assumptions of page-based virtual memory:

- The physical address space and *physical main memory* are no longer isomorph, so allocation can no longer rely on physical addresses as an identifier for space in memory.
- Physical page frames are not always contiguous: when allocated as contiguous page in one mapping alias, they will appear as a set of *page frame stripes* in each alias with different interleaving. Thus, careless allocation could disable an alias by fragmenting all page frames as seen from that alias.

In our talk, we motivate the need for DRAM-aware operating systems and discuss how informed memory management helps to cope with performance interference. Then, we present our prototype and sketch how to adapt page-based memory management for systems with flexible address mapping.