Putting the OS in Control of DRAM with Mapping Aliases

Marius Hillenbrand * Frank Bellosa

Karlsruhe Institute of Technology

os@itec.kit.edu

On multicore CPUs, concurrent processes compete for shared resources such as caches and the memory subsystem. Sharing DRAM resources, such as channels and banks, can increase request latencies and thereby slow down applications [3]. As an illustration, we measure the slowdown of the Parsec suite with several memory-intensive background jobs. We observe that slowdowns depend on the combination of workloads and can vary widely for a given benchmark.

These slowdowns result from how DRAM operates, as read and write requests fall into two categories: (1) fast *row hits* address the row currently loaded in a DRAM bank's row buffer whereas (2) *row misses* first have to load another row into the buffer and suffer a $\sim 3x$ higher latency. When the request streams of competing processes hit the same bank, locality is lower than from each process alone, resulting in a higher row miss rate and thus higher memory latency. Consequently, DRAM partitioning is effective for reducing the slowdown from memory interference because it avoids sharing: Partitioning dedicates DRAM resources, such as channels and banks, via page allocation [1, 2]. Effectively, the OS performs long-term scheduling on DRAM, treating it as a processing resource for read and write requests.

While partitioning can mitigate interference, it introduces overhead: we need to disable interleaving to give the OS control of how pages map to DRAM resources. Interleaving improves application performance by mapping channel and bank indices to low-order address bits within the page offset, thereby spreading requests over parallel DRAM resources. In contrast, partitioning reduces DRAM parallelism because channel and bank indices need to be controlled by high-order address bits within the page frame number chosen by the OS.

In a conventional system, we have to choose either DRAM partitioning or interleaving at boot time, when the BIOS configures a fixed mapping of physical addresses to DRAM. At that moment, we typically do not know yet which jobs will later run on a system and whether their interference will cause slowdowns. When our choice turns out to be wrong later, we pay a performance penalty: We might have chosen interleaving, yet find at run time that we need partitioning to reduce slowdowns. Vice versa, we might boot a system with partitioning and run only jobs that would benefit from interleaving and do not suffer from interference.

We propose to overcome this limitation by introducing *DRAM Mapping Aliases*: We map DRAM into the physical address space multiple times, each time with another mapping scheme, by configuring the memory controller. Typically, one of these mappings, which we call *aliases*, would be interleaved, while another one would partition channels and banks. By choosing page frames from the right *alias* when allocating memory, the OS can choose between interleaving and sharing or partitioning DRAM channels and banks at run time. Mapping aliases break the common assumption that physical addresses uniquely identify memory. Thus, we have to be careful not to allocate the same underlying DRAM twice.

We implement mapping aliases with small changes to the Linux kernel. For partitioning and choosing aliases, we create *fake* NUMA nodes, one for each interleaved alias and each partition. Then, we can decide to isolate processes to separate channels and/or banks or let them share an interleaved alias by binding them to the respective fake nodes using existing NUMA APIs. To ensure that each piece of DRAM is allocated at most once, we utilize memory hotplugging to have only one mapping of every piece of DRAM visible to Linux at all times. For reconfiguring, we first migrate the available physical memory by taking blocks *offline* in the old memory and then setting the corresponding blocks *online* in the new alias. Then, we migrate processes' pages and page tables to the new alias.

Our prototype runs on an *off-the-shelf* AMD Athlon X4 880K CPU [1] and provides aliases for channel interleaving and partitioning. In our experiments, we find that migrating processes between both schemes at run time mitigates most of the performance penalty induced by conventional systems' fixed mapping.

References

[1] L. Liu et al. BPM/BPM+: Software-based dynamic memory partitioning mechanisms for mitigating dram bank-/channel-level interferences in multicore systems. *ACM Trans. Archit. Code Optim.*, Feb. 2014. .

[2] S. P. Muralidhara et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *IEEE/ACM MICRO-44*, pages 374–385, New York, NY, USA, 2011. ACM. .

[3] O. Mutlu et al. Research problems and opportunities in memory systems. *Supercomputing Frontiers and Innovations: an Intern. Journal*, 1 (3):19–55, Oct. 2014. .

* PhD student, will be presenting.

[1] For Intel CPUs, the necessary documentation is not publicly available.

Karlsruhe Institute of Technology

Operating Systems Group
http://os.itec.kit.edu

Putting the OS in Control of DRAM with Mapping Aliases

Optimistically use Interleaving or Switch to DRAM Partitioning at Runtime

Marius Hillenbrand, Frank Bellosa

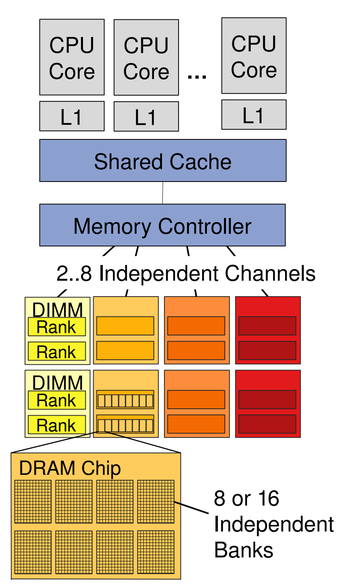
Motivation: DRAM Interference

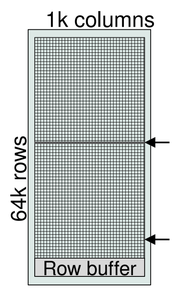
- In a dynamic system, we do not know in advance if interference will be a problem
- Experimental Setup
  - Parsec suite + background job
  - 4-core AMD Athlon X4 880K
  - Partition cores and caches
- Slowdown depends on workload mix

<= 2% for 4 foreground jobs (freqmine bodytrack swaptions blackscholes)	up to 9% for 5 (raytrace x264 fluidanimate ferret dedup)	up to 24% for 2 (streamcluster facesim)

- 10-core Intel Xeon E5 v4: Up to 120% slowdown

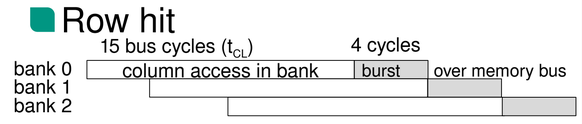
Classic Mitigation: DRAM Partitioning



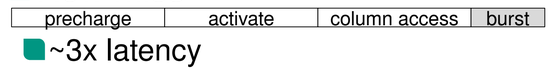


Two categories of requests

- Row hit

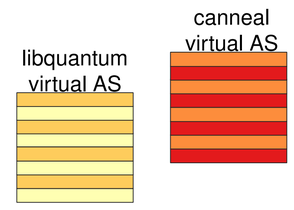


- Row miss



  - ~3x latency

- Sharing DRAM banks overlaps request streams and reduces row locality
- Partitioning improves row locality yet reduces bank/channel parallelism

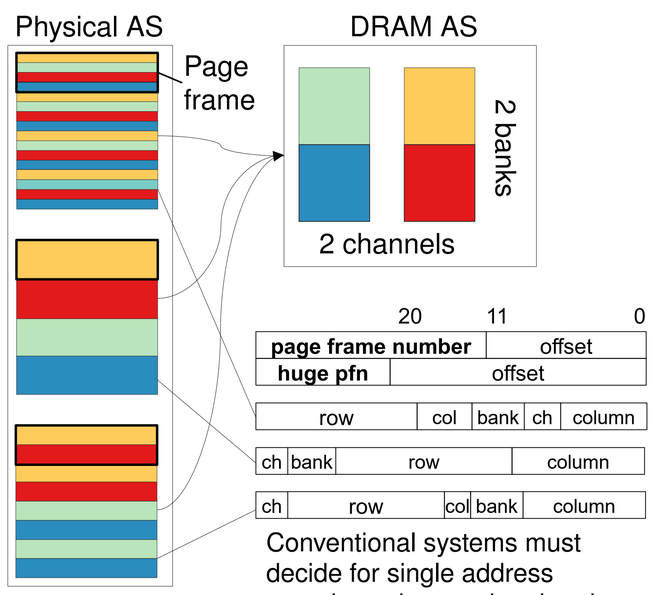


- Dedicate channels or banks via page allocation
- Reduced bandwidth over interleaving
- Must decide when booting

Approach: Mapping Aliases – Choose Interleaving or Partitioning at Runtime

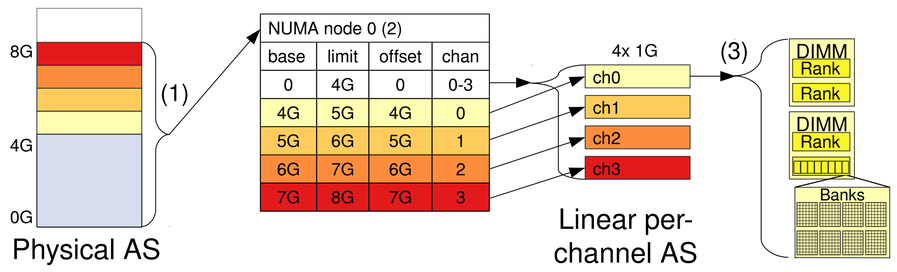
- Map DRAM multiple times, for example
  1) Bank + channel interleaving
     - Max parallelism
     - No isolation
  2) Linear
     - Channel and bank partitioning
     - Minimum parallelism
  3) Bank interleaving
     - Channel partitioning
     - Bank parallelism

(Bank aliases not supported on current hardware)



Conventional systems must decide for single address mapping scheme when booting

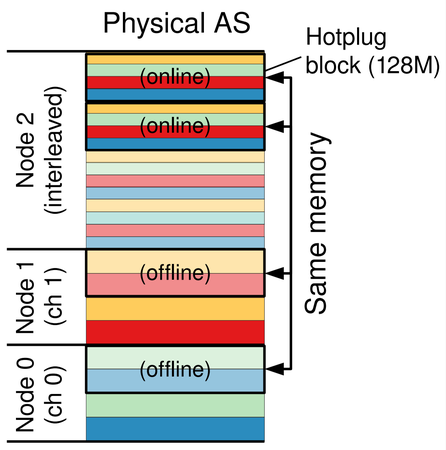
- Reconfigure existing physical addressing scheme
  - (1) Source Address Decoder (CPU core/LLC)
  - (2) Target Address Decoder (memory controller on NUMA node)
  - (3) Per-channel address decoder



- AMD Desktop/Server: Prototype on Steamroller
  fully documented: [BIOS and Kernel Developer's Guide for AMD Family 15h Models 30h-3Fh Processors]
- Intel Server: Plausible, config regs scarcely documented
  [Intel Xeon Processor 7500 Series Datasheet, Volume 2]
  [W. Song et al. Plkit: A New Kernel-Independent Processor-Interconnect Rootkit. 25th Usenix Security Symposium, 2016]

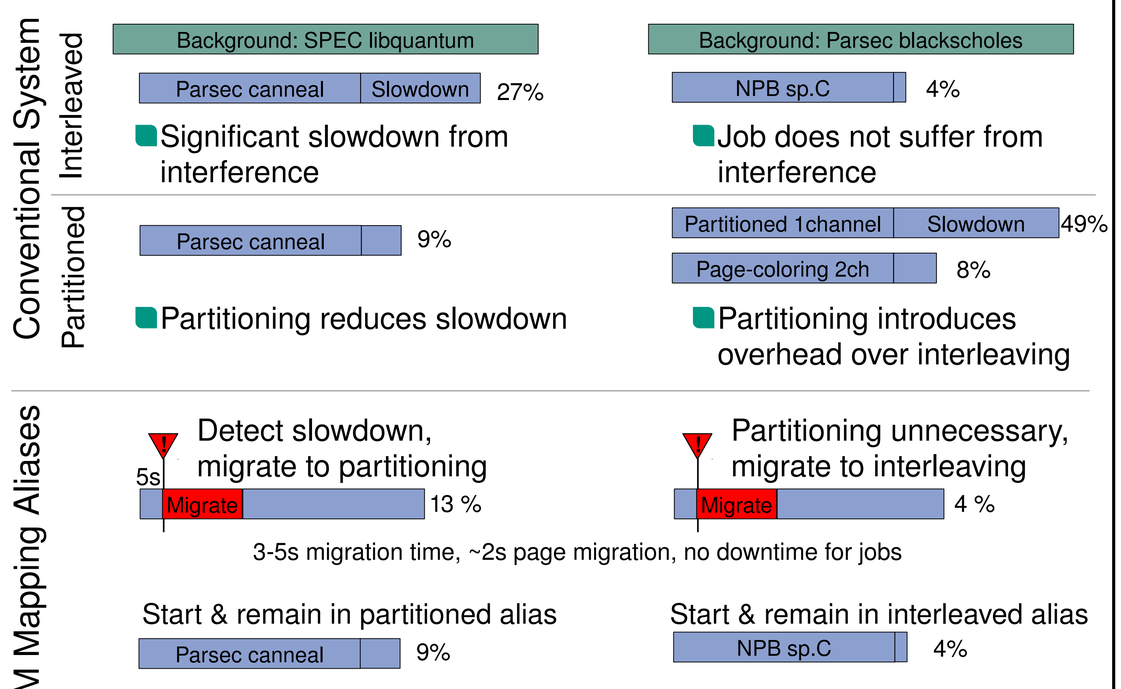
Implementation: Alias Allocation in Linux

- Partitioning and Allocation
  - Reuse existing NUMA support
  - Custom assignment of phys. AS to pseudo-NUMA nodes
- Avoiding Conflicts
  - Utilize memory hotplugging support
  - Aliases define conflicts between blocks
  - Never have conflicting blocks online at the same time
- Migration Between Aliases at Runtime
  - Existing NUMA migration for processes
  - Also migrate page tables
  - Offline/online blocks to migrate physical memory to requested mapping mode
  - Caches unaware of aliases: must invalidate caches during migration



Evaluation

- Parsec canneal + SPEC libquantum
- NPB sp.C + blackscholes



- Significant slowdown from interference
- Job does not suffer from interference
- Partitioning reduces slowdown
- Partitioning introduces overhead over interleaving
- Similar behavior for other benchmarks
- Prototype: AMD Athlon X4 880K (4x4 GHz, 2-channel 32 GiB), modified Linux 4.4.36, transparent hugepages enabled