Mitosis: Transparently Self-Replicating Page Tables for Big Memory Machines

Reto Achermann (student)[1,3], Jayneel Gandhi[1], Abhishek Bhattacharjee[2], Timothy Roscoe[3]
acreto@inf.ethz.ch, gandhij@vmware.com, abhib@cs.rutgers.edu, troscoe@inf.ethz.ch
[1]VMware Research, [2]Rutgers University, [3]ETH Zurich

Big memory machines consists of multiple processors to increase the total amount of bandwidth to and capacity of main memory. This multi-socket machines typically feature non-uniform memory access (NUMA) characteristics: accessing memory attached to the local processor has different performance characteristics than accessing memory attached to a remote processor. The observed latencies and bandwidth to remote memory can differ up to 2-3x depending on the machine topology. Optimizing workloads and data placement strategies for such NUMA machines has been an open research area e.g. back-box concurrent data structures [1], Carrefour [2], large pages in NUMA systems [3] or Shoal [4]. Commercial OSes implement NUMA page migration that tracks accesses to pages and migrates them when they've been frequently accessed by a remote node. For example, Linux implements a feature called AutoNUMA for migrating pages between NUMA nodes

None of the prior work investigated the effects of page-table placement in the context of NUMA machines. Data pages are allocated on the NUMA node that first touches the page (or interleaved across all nodes) and the data pages can be easily migrated in case of a mis-placement. Whereas the page table pages cannot be migrated easily since they use physical address-based pointers. In addition, on x86, a single page table has 512 entries and using an interleave policy implies that $N - 1/N$ entries will point to a remote data page (with N the number of NUMA nodes).

The page table placement has become more important since the TLB reach has been diminishing: the fraction of physical memory that can be mapped by all the TLB entries is tiny. For example, on an Intel Haswell processor the TLB reach is less than 1% assuming 1TB of main memory. Therefore, in big-memory machines, the pressure on the TLB has increased requiring up to 4 memory accesses (or up to 24 memory accesses in the virtualized case) by the page walker to resolve a single virtual-to-physical mapping. In the context of parallel workloads, this in turn means that on average $N - 1/N$ of these memory accesses will be served by a remote node and hence the page walker will suffer from the additional NUMA latency when the page table is allocated on a remote node. This is mainly caused by the fact that there is a single page table for the process used by all cores and secondly a single page table page has multiple entries which may point to pages on different NUMA nodes. Through our experiments, we found that even with local allocation policy we saw up to 80% of the page tables pointing to a remote node.

In this work, we investigate the performance implications of page table placement in big-memory machines. Our microbenchmarks show that on a 4 socket Intel Haswell machine, the performance degradation due to remote placement of page tables can be up to 3.4x for the HPCC RandomAccess benchmark workload. While this workload is designed to have a high cache-miss rate, similar effects can be seen in key-value stores like Redis, for example, where the performance degradation can be up to 2x in the worst case.

To reduce the overheads of remote accesses on a page table walk on a NUMA machine, we design *Mitosis*: a transparently self-replicating page table for big-memory machines. In our design, we fully replicate page tables on NUMA nodes by changing page table allocation and management subsystem. We can enable page table replication on a per-process basis which will create and maintain a replica for each NUMA node the process runs on. When the process is scheduled to run on a core, it writes the core's page table pointer (CR3 register on x86 processors) with the physical address of the page table replica of the local NUMA node. Each time the OS modifies the page tables, we ensure that the updates are propagated to all replica page tables efficiently and reads return consistent values based on all replicas (including the hardware updated dirty and access bits).

In addition, we would like to extend the design of *Mitosis* to support virtualized environments. In a virtual environment with hardware support (extended page tables), handling a TLB miss has a higher overhead than in the native case due to the required 2D page table walk introducing at most 24 memory accesses to resolve a single TLB miss. We envision to modify the hypervisor to do the replication transparently underneath the guest enabling unmodified guest OSes to run in a VM.

Currently, we implement our design of *Mitosis* in Linux and only supports native execution. Our experiments shows that we can fully mitigate the 3.4x slowdown of HPCC RandomAccess and improve other single threaded workloads by 30% and multi threaded workloads by 15% respectively. We would extend our *Mitosis* design in Linux with KVM to improve performance of applications running in a virtualized system.

[1] Calciu, I., Sen, S., Balakrishnan, M., and Aguilera, M. K. Black-box Concurrent Data Structures for NUMA Architectures. ASPLOS '17.

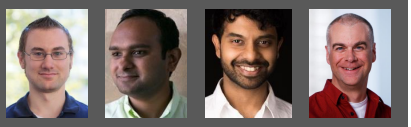
[2] Dashti, M., Fedorova, A., Funston, J., Gaud, F., Lachaize, R., Lepers, B., Quema, V., and Roth, M. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. ASPLOS '13.

[3] Gaud, F., Lepers, B., Decouchant, J., Funston, J., Fedorova, A., and Quéma, V. Large Pages May Be Harmful on NUMA Systems. USENIX ATC'14.

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

Mitosis: Transparently Self-Replicating Page Tables for Big Memory Machines

Mitigating NUMA effects on TLB misses with smart page table allocation policies



Reto Achermann, Jayneel Gandhi, Abhishek Bhattacharjee, Timothy Roscoe

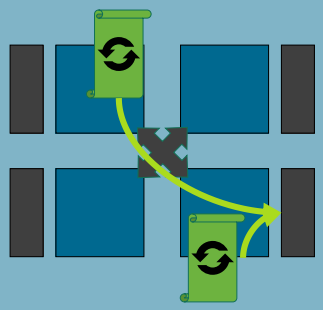
Problem

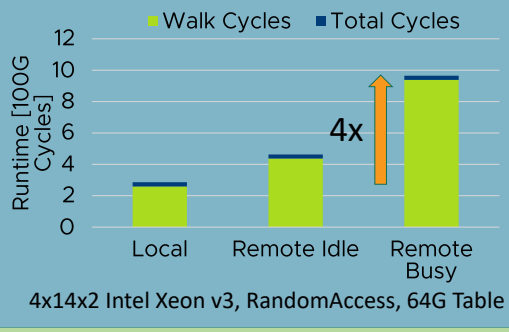
Multi-socket servers:
- widely used in data centers and cloud appliances
- shared machines
- heterogeneous workloads

Bad memory allocation harm performance:
- Additional latencies
- Contention on memory controllers / interconnect

TLB miss are expensive:
- page walks may miss CPU caches
- require up to 24 additional memory accesses

Multi threaded programs: page-tables will be remote (N-1)/N times



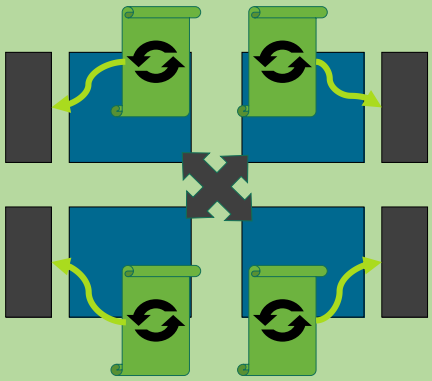


4x14x2 Intel Xeon v3, RandomAccess, 64G Table

Approach

Smart allocation policies for page tables in large NUMA machines

- Model the page table accesses
- Analyze runtime of program
- Adapt placement of page tables



State of Work

Initial Measurements	Analysis	OS Kernel Implementation	Hypervisor Implementation
- Workloads - Explore Configurations - Gather Data	- K-Factor Analysis - Performance Model	- Linux Kernel modifications	- Hypervisor modifications (ESXi / KVM)

vmware RESEARCH