FAST LOCAL PAGE TABLES FOR NUMA SERVERS WITH MITOSIS

Reto Achermann, University of British Columbia
MITOSIS: MITIGATING NUMA EFFECTS ON PAGE TABLE WALKS

Up to 3.2x speed up for single threaded applications.

Up to 1.6x speed up on multi-threaded applications

No modifications of the application workloads

Linux (native) and KVM-based virtual machines
NUMA EFFECTS ON LARGE MULTI-SOCKET MACHINES

Bandwidth & capacity limited per processor socket

More bandwidth & capacity needed
→ Multi-socket machine

Non-uniform memory access (NUMA) machine
→ Memory accesses experience different performance characteristics

Measured on 4x12 Intel Xeon E7-4850 v3, with 512GB RAM (4x128GB)
KEEPING PROCESS THREADS AND DATA CLOSE

NUMA-aware scheduling

The page-table stayed where it was...

NUMA-aware data migration
RECAP: VIRTUAL TO PHYSICAL TRANSLATION

Processes deal with virtual addresses

Defined by the page table.

Multi-level radix-tree.

Translation (MMU)

On every memory access

Processes deal with virtual addresses

Memory knows physical addresses
# TLB Misses Trigger Page Table Walks

**Virtual address** 0x40000930

**TLB Miss (slow path)**

<table>
<thead>
<tr>
<th>Page Table Root Pointer</th>
</tr>
</thead>
</table>

**TLB Hit (fast path)**

- **VA**
- **PA**
- **Perms**

**Page Table Walk**

- **PML4**
- **PDPT**
- **PDIR**
- **PT**

**Frame**

- Up to 4 additional memory accesses
Modern processors support two-stage translation schemes.
TWO STAGE TRANSLATIONS OF VIRTUAL MACHINES

Translation is **not** sequential!

Page table walk for gVA -> gPA translation uses guest physical addresses. → guest page table walk needs translation

Up to 24 memory accesses to perform gVA -> hPA translation
THE PROBLEM IN SUMMARY

1. More DRAM capacity & stagnating TLB size → decreasing TLB coverage and more frequent TLB misses.

2. These TLB misses require multiple memory accesses


4. Contention may evict page tables from caches → page walker accesses end up in DRAM

5. Accessing DRAM may be subject to NUMA effects
NUMA effects on page table walks – a systematic analysis.
REMOTE PAGE TABLE WALKS ON NUMA SYSTEMS

How often is the page table remote and what’s the resulting slowdown?

Systematic analysis with different scenarios:

| Multi-Threaded Scenario | Workload-Migration Scenario |
Multi-Threaded Scenario

MANY THREADS – ONE PAGE TABLE

Capture the page table
Analyze NUMA affinity
of page table entries

Virtual machine
NUMA visible
configuration

Physical Machine

Virtual machine
NUMA oblivious
configuration
WHERE DO PAGE TABLE ENTRIES POINT TO?

Majority of TLB misses require remote memory accesses.

Virtual machine NUMA visible configuration

Virtual machine NUMA oblivious configuration

Workload: Graph500
REMOTE PAGE TABLE WALKS ON NUMA SYSTEMS

How often is the page table remote and what’s the resulting slowdown?

Systematic analysis with different scenarios:

Multi-Threaded Scenario

Majority of TLB misses require remote memory accesses

Workload-Migration Scenario
REMOTE PAGE TABLE WALKS ON NUMA SYSTEMS

Pinning page tables, data pages, threads to specific cores

Configurations:

Data: Local/Remote
Page Table: Local/Remote
EPT: Local/Remote
+ Interference on PT node

Virtual machine
NUMA visible configuration

Physical Machine
Workload-Migration Scenario

REMOTE PAGE TABLE WALKS ON NUMA SYSTEMS

Remote page walks during TLB misses have significant impact on application performance.

Virtual machine NUMA visible configuration

More configurations and workloads in the paper

Physical Machine
SUMMARY: MEMORY ACCESES FROM PAGE WALKER EXPERIENCE NUMA EFFECTS

TLB misses result in additional memory accesses

Systematic analysis with different scenarios:

- **Multi-Threaded Scenario**
  
  Majority of TLB misses require remote memory accesses

- **Workload-Migration Scenario**
  
  Remote page walks during TLB misses have significant impact on application performance
Mitosis: Keep Page Tables Local using Replication and Migration.

Design and Implementation on Linux/KVM / x86_64

Available on GitHub
https://github.com/mitosis-project
MITOSIS+VMITOSIS: MAKING PAGE TABLES LOCAL

Multi-Threaded Scenario

Mitosis replicates the page table

Workload-Migration Scenario

Mitosis migrates the page table
REPLICATION OF PAGE TABLES

Allocation
(Performance)

Consistency
(Correctness)

Utilization
(Performance)
MITOSIS EAGERLY ALLOCATES REPLICA PAGE TABLES

We need to efficiently find the new replica’s parent page table.

allocate all replica page tables from the corresponding NUMA node.
EFFICIENT PROPAGATION OF UPDATES

Added pointers for circular linked list

Metadata for page 1

Metadata for page 2

Metadata for page 3

Metadata for page 4

Replica page 1
PTE

Replica page 2
PTE

Replica page 3
PTE

Replica page 4
PTE

Primary
SELECTING THE RIGHT REPLICA

Scheduler picks thread to run

Context switch

Select local replica

Write translation base register

Extend Linux’ mm_struct with an array of page table roots
Indexed by NUMA node

mm->root[local_node()] (KVM similar)
MIGRATING PAGE TABLES: REPLICATE + FREE

- Create a new replica
- Free old replica
- Switch to local replica

Linux already migrates data pages, can we hook in there?
MIGRATION OF PAGE TABLES

Locality Invariant: Placing page-table pages with majority of their children

Ideal placement: socket-1
MECHANISM: AUTONUMA FOR PAGE TABLES

AutoNUMA migrates data page
USING MITOSIS

No application modifications
numactl -r <sockets> <workload>

*) Requires Mitosis enabled kernel and libnuma

Available on GitHub
https://github.com/mitosis-project
REPLICATION WITHOUT NUMA TOPOLOGY INFORMATION

NUMA visible case is easy:
1) get NUMA node of CPU
2) select local replica for gPT and ePT

Problem: NUMA topology is not always available
(e.g., hidden by the hypervisor)

Requirements:
- vCPU -> NUMA node
- memory page -> NUMA node
RECONSTRUCTING NUMA TOPOLOGY

Measure **cache-line transfer latency** between any pair of cores.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>272</td>
<td>590</td>
<td>587</td>
</tr>
<tr>
<td>1</td>
<td>265</td>
<td>-</td>
<td>593</td>
<td>585</td>
</tr>
<tr>
<td>2</td>
<td>588</td>
<td>585</td>
<td>-</td>
<td>269</td>
</tr>
<tr>
<td>3</td>
<td>590</td>
<td>589</td>
<td>271</td>
<td>-</td>
</tr>
</tbody>
</table>

**vNUMA 0**: 0, 1, ...
**vNUMA 1**: 2, 3, ...
**vNUMA 2**: 4, 5, ...
**vNUMA 3**: 6, 7, ...

Measured on 4x12 Intel Xeon E7-4850 v3, with 512GB RAM (4x128GB)

- ~279 cycles ~11 GB/s
- ~28 GB/s
- ~583 cycles
ALLOCATING LOCAL PAGES

Exploit the first-touch / local allocation policy

Chose vCPU to access the page.

Hypervisor handles ePT violation, and maps local page
Results
EVALUATION

Multi-Threaded Scenario
What is the resulting speedup on multi-threaded workloads with Mitosis?

Workload-Migration Scenario
Can Mitosis prevent the significant slowdown?

Mitosis Overheads
What is the overhead of Mitosis?

Linux and Linux+KVM

4x26 Intel Xeon Gold 6252 with 1.5TiB RAM (4x384 GiB)

4x12 Intel Xeon E7-4850 v3, with 512GB RAM (4x128GB)
Mitosis replicates the page table.
MITOSIS ON KVM VIRTUAL MACHINE

Multi-Threaded Scenario

NUMA VISIBLE

<table>
<thead>
<tr>
<th></th>
<th>Memcached</th>
<th>XSBench</th>
<th>Graph500</th>
<th>Canneal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/KVM</td>
<td>1.1x</td>
<td>1.6x</td>
<td>1.3x</td>
<td>1.7x</td>
</tr>
<tr>
<td>vMitosis</td>
<td>1.1x</td>
<td>1.4x</td>
<td>1.5x</td>
<td>1.3x</td>
</tr>
</tbody>
</table>

NUMA OBLIVIOUS

<table>
<thead>
<tr>
<th></th>
<th>Memcached</th>
<th>XSBench</th>
<th>Graph500</th>
<th>Canneal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/KVM</td>
<td>1.2x</td>
<td>1.25x</td>
<td>1.4x</td>
<td>1.2x</td>
</tr>
<tr>
<td>vMitosis</td>
<td>1.1x</td>
<td>1.1x</td>
<td>1.2x</td>
<td>1.1x</td>
</tr>
</tbody>
</table>
## Multi-Threaded Scenario

### MITOSIS WITH TRANSPARENT HUGE PAGES (THP)

<table>
<thead>
<tr>
<th>NUMA VISIBLE</th>
<th>NUMA OBLIVIOUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normalized runtime</strong></td>
<td><strong>Normalized runtime</strong></td>
</tr>
<tr>
<td>Memcached</td>
<td>Memcached</td>
</tr>
<tr>
<td>XSBench</td>
<td>XSBench</td>
</tr>
<tr>
<td>Graph500</td>
<td>Graph500</td>
</tr>
<tr>
<td>Canneal</td>
<td>Canneal</td>
</tr>
<tr>
<td><strong>Normalized runtime</strong></td>
<td><strong>Normalized runtime</strong></td>
</tr>
<tr>
<td>Linux/KVM</td>
<td>Linux/KVM</td>
</tr>
<tr>
<td>vMitosis</td>
<td>vMitosis</td>
</tr>
</tbody>
</table>

- **NUMA VISIBLE**
  - **Linux/KVM**
  - **vMitosis**
  - **Normalized runtime**
  - **Out-of-memory**

- **NUMA OBLIVIOUS**
  - **Linux/KVM**
  - **vMitosis**
  - **Normalized runtime**
  - **Out-of-memory**

- **vMitosis** shows a 1.12x improvement over **Linux/KVM** in **out-of-memory** scenario.
MITOSIS ON NATIVE LINUX

Workload-Migration Scenario

Mitosis migrates the page table

Execution Cycles

0 1 2 3 4

Local PT Remote PT Mitosis Local PT Remote PT Mitosis

GUPS Redis

Mitosis

Xeon E7 v3

128 GB DDR

Xeon E7 v3

128 GB DDR

Xeon E7 v3

128 GB DDR

Xeon E7 v3

128 GB DDR

Process

L2

L1

Local PT Remote PT

Mitosis

1.8x

3.2x
MITOSIS ON KVM VIRTUAL MACHINE

**Workload-Migration Scenario**

### Page size: 4KiB

<table>
<thead>
<tr>
<th>Workload</th>
<th>Local</th>
<th>Linux/KVM</th>
<th>vMitosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUPS</td>
<td>4.0</td>
<td>3.1x</td>
<td>1.02x</td>
</tr>
<tr>
<td>Redis</td>
<td>3.5</td>
<td>2x</td>
<td>1.47x</td>
</tr>
<tr>
<td>Memcached</td>
<td>3.0</td>
<td>2.4x</td>
<td>out-of-memory</td>
</tr>
<tr>
<td>Canneal</td>
<td>2.5</td>
<td>2.3x</td>
<td>1.35x</td>
</tr>
</tbody>
</table>

### Page size: 2MiB (THP)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Local</th>
<th>Linux/KVM</th>
<th>vMitosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUPS</td>
<td>3.9</td>
<td>3.1x</td>
<td>1.02x</td>
</tr>
<tr>
<td>Redis</td>
<td>3.4</td>
<td>2x</td>
<td>1.47x</td>
</tr>
<tr>
<td>Memcached</td>
<td>2.9</td>
<td>2.3x</td>
<td>out-of-memory</td>
</tr>
<tr>
<td>Canneal</td>
<td>2.4</td>
<td>2.2x</td>
<td>1.35x</td>
</tr>
</tbody>
</table>
OVERHEAD OF MMAP AND FRIENDS

Effects of **eager** updates to 4 replicas – fixable with deeper page fault handler integration

Space Overhead: 0.6% of additional memory for 4-way replication
ACKNOWLEDGEMENTS

Ashish Panwar
Arkaprava Basu
Kanchi Gopinath
Abhishek Bhattacharjee
Timothy Roscoe
Jayneel Gandhi
# MITOSIS: ELIMINATE NUMA EFFECTS ON PAGE TABLE WALKS

## Observations

- Large memory server use **multi-socket architecture**.
- Existing NUMA optimizations **ignore kernel objects**.
- NUMA effects on **page table walks** are inevitable.

## Contributions

- **Systematic study** showing NUMA effects on page table walks.
- Mechanisms and policy for **replication and migration** of page tables.
- **Linux + KVM** implementation

## Results

- Speedup for big-memory workloads **without application modifications**
  - 1.06 - 1.6x speed up for multi-socket workloads
  - 1.8 - 3.1x speed up for single-socket workloads

Contact: achreto@cs.ubc.ca

Available on GitHub
https://github.com/mitosis-project