Transparent OS support for variable translation sizes

Stratos Psomadakis
National Technical University of Athens

Georgios Goumas
National Technical University of Athens

Research Problem

The address translation (AT) overhead has been widely studied in literature and the new 5-level paging is expected to make translation even costlier. Multiple solutions have been proposed to alleviate the issue either by reducing the number of TLB misses or by reducing their overhead. The solution widely adopted by industry involves extending the page sizes supported by the hardware and software, with the most common being 2MB and 1GB.

As the effectiveness of 2MB pages starts to diminish with the ever increasing memory footprint of applications [1], a natural solution would be to replace 2MB with 1GB pages. However, 1GB pages are more cumbersome to use. The alignment restrictions of larger pages (i.e., memory should be 1GB-aligned both physically and virtually) limits their usefulness, especially in high fragmentation scenarios [5]. In addition, the Linux memory subsystem imposes more drawbacks to their use. Since it faults-in the whole page, be it 4KB, 2MB or 1GB, the page fault tail latency and the memory bloat increase when the page size increases[3]. These are a few reasons why the OS support for 1GB pages remains limited (hugetlbfs), while for 2MB pages it is ubiquitous and in most cases transparent (THP). It seems that an intermediate page size could be more easily exploited.

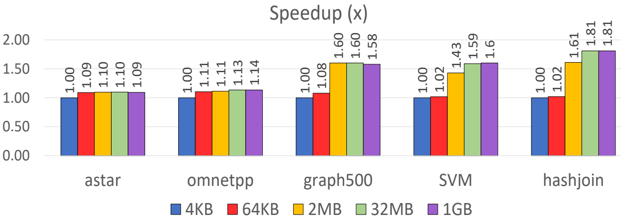
At the same time, in high fragmentation scenarios, even 2MB pages might become difficult to allocate [4]. In these cases, an intermediate page size between 4KB and 2MB could also be beneficial. r In contrast to x86, ARMv8-A and RISC-V provide architectural support for such an extended range of page sizes, in the form of either a configurable base page size or TLB support for OS-assisted coalescing, i.e., treating a group of OS-designated contiguous pages as a single translation entity. As these ISAs are gradually making their way to the datacenter, where the large memory footprints of the workloads stress the address translation hardware, we argue that these intermediate translation sizes can be exploited to address limitations exhibited by the prevalent 2MB / 1GB page model. We also consider their transparent OS support, that is currently missing, a key enabler to this direction.

Our contributions

Based on the above, we first evaluate the usefulness of these intermediate translation sizes, using memory-intensive workloads running on an ARMv8-A server. ARMv8-A paging structures include a (contig) bit which, if set in N consecutive page table entries, indicates that the mapped pages are contiguous and suitably aligned both physically and virtually. This allows the TLB to coalesce these entries to one, increasing the TLB reach and effectively forming intermediate translation sizes. The currently supported sizes are 64KB for 16 contiguous 4KB pages and 32MB for 16 contiguous 2MB pages [2].

Linux supports these intermediate sizes via the hugetlbfs interface, which requires memory pre-allocation. Despite this limitation, we show that running a series of benchmarks backed by 32MB Hugetlbfs pages reduces the AT overhead compared to 2MB pages, and provides similar performance gains to 1GB pages for big memory workloads, such as SVM and hashjoin. (Fig. 1). For smaller and irregular workloads, such as omnetpp and astar, 64KB pages eliminate the AT overhead, which can be especially useful in cases of high fragmentation.

Figure 1. Execution time normalized to 4KB



Based on these findings our research goal is to design a transparent OS mechanism that creates these intermediate translation sizes on demand, omitting the need for memory pre-allocation. To this end, we extend contiguity-aware paging [1] to allocate properly aligned contiguous pages across faults, thus lazily generate groups of contiguous pages. When such a group is created, i.e., when all the pages in the group are faulted-in, our mechanism promotes it transparently to the corresponding intermediate translation size. If for any reason the group is broken, our mechanism is responsible for demoting the mapping and keeping the TLBs coherent. Our preliminary results indicate performance comparable to hugetlbfs, while maintaining the flexibility and transparency of vanilla 4KB / 2MB paging.

Acknowledgements

This work is funded by the EU under the Horizon Europe grant 101092850 (project AERO).

References

[1] Chloe Alverti et al. “Enhancing and Exploiting Contiguity for Fast Memory Virtualization”. In: ISCA 2020.

[2] ARM LTD. *Arm Architecture Reference Manual for A-profile architecture (D.8.6.1 The Contiguous bit).*

[3] Youngjin Kwon et al. “Coordinated and Efficient Huge Page Management with Ingens”. In: OSDI 2016.

[4] Chang Hyun Park et al. “Every Walk’s a Hit: Making Page Walks Single-Access Cache Hits”. In: ASPLOS 2022.

[5] Zi Yan et al. “Translation Ranger: Operating System Support for Contiguity-Aware TLBs”. In: ISCA 2019.

Transparent OS Support for Variable Translation Sizes

Stratos Psomadakis, Georgios Goumas
National Technical University of Athens

Funded by the European Union

Summary

Problem: Address translation overheads exacerbated by 5-level and nested paging.

2MB pages (THP): Diminishing returns as:

- the working sets continue to grow.
- Fragmentation limits effective coverage.

1GB pages: Not as widely supported, harder to use, harder to allocate when memory gets fragmented.

Our proposal

- Exploit the **intermediate translation sizes** available on ARMv8-A and RISC-V.
- Enhance the OS memory manager to **transparently support** these variable translation sizes.

ARMv8-A Intermediate Translation Sizes

Contig Bit: L1 (PTE) and L2 (PMD) paging structures include a contig bit, which when set in 16 consecutive suitably aligned entries, allows the TLB to cache them as a single translation entry.

Supported Intermediate Sizes:
1. 16x L1 (PTE) 4KB entries, coalesced to a single 64KB translation,
2. 16x L2 (PMD) 2MB entries, coalesced to a single 32MB translation.

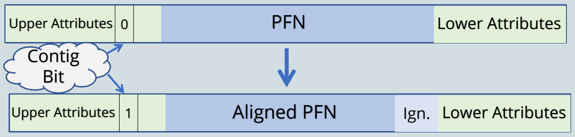
HW-supported Translations Sizes

Base	Large	Intermediate
4KB	2MB, 1GB	64KB, 32MB

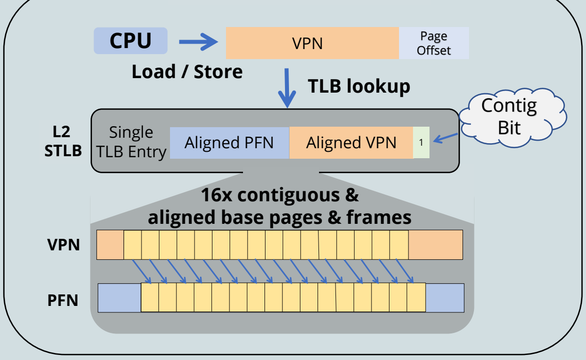
✓ Transparent OS Support (THP)

✗ Requires pre-allocation (Hugetlbfs)

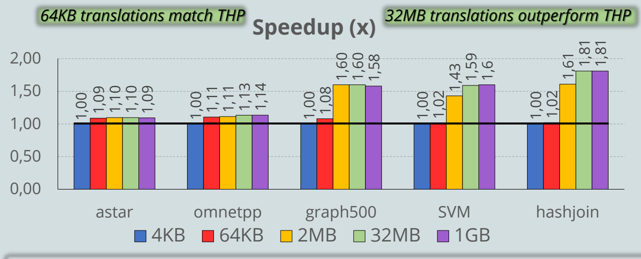
Contiguous Translation Descriptors (Page Table Entries) :



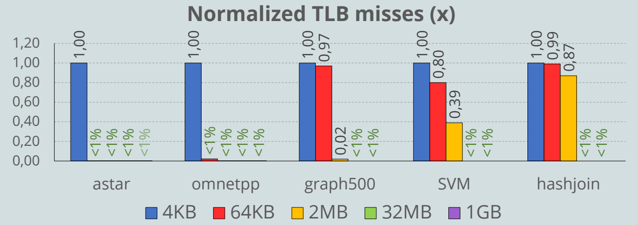
Coalesced TLB entries



Intermediate Translation Sizes: Potential and Challenges



Hugetlbfs performance normalized to 4KB,running on Ampere Altra (ARMv8.2-A)



Observations

Insight: Intermediate translation sizes provide *performance similar to larger sizes* while having *less* strict *alignment* and *fragmentation requirements*.

- ✓ *64KB translations* eliminate the AT overhead for irregular workloads with smaller footprints, like *astar* and *omnetp.*
- ✓ *32MB translations* provide up to *20%* better performance compared to THP, matching the performance of 1GB for big memory workloads. For *SVM,* they use *16%* less memory compared to 1GB (*37GiB* vs *44GiB*).

Under high memory fragmentation, we also expect a combination of 64KB and larger translations to be able to better utilize the available memory contiguity.

Limitations

Using intermediate sizes is currently impractical as they are supported on Linux only by Hugetlbfs, which requires:

1. system-wide memory pre-allocation,
   a. which incurs significant run-time overheads, e.g. 50% for hashjoin, for the 32MB configuration,
   b. makes reserved memory non-reclaimable by the OS.
2. application (or library) opt-in and non-standard userspace allocator configurations.

Barriers to transparent usage

- The OS memory manager does not support intermediate-sized on-demand faults.
- On-demand allocation of larger sizes via faults could increase tail latency and lead to memory bloat.
- On-demand faults for base pages are not contig bit–aware.

Transparent Variable Translation Sizes (TVTS)

Our Proposal

Transparently create intermediate-sized translations:

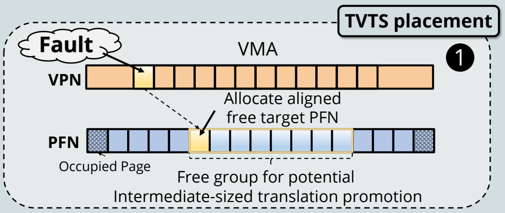
- Enhance CA-Paging [1] to create on demand suitably-aligned contiguous groups of pages.
- Transparently map them to the corresponding HW-supported intermediate translation sizes.

❶ First fault in the VMA. TVTS selects a intermediate-size aligned target PFN.

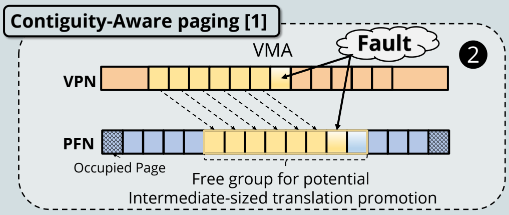
❷ CA-Paging [1] directs subsequent base page size faults to their corresponding aligned PFNs.

❸ All base pages have been allocated. TVTS promotes the group to an intermediate-sized translation, by setting the contig bit in the base pages' descriptors.

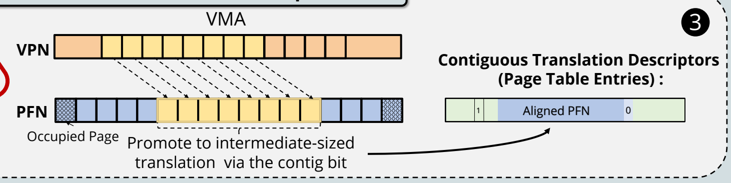
TVTS placement


Fault
VMA
1
VPN
Allocate aligned free target PFN
PFN
Occupied Page
Free group for potential Intermediate-sized translation promotion

Contiguity-Aware paging [1]


VMA
Fault
2
VPN
PFN
Occupied Page
Free group for potential Intermediate-sized translation promotion

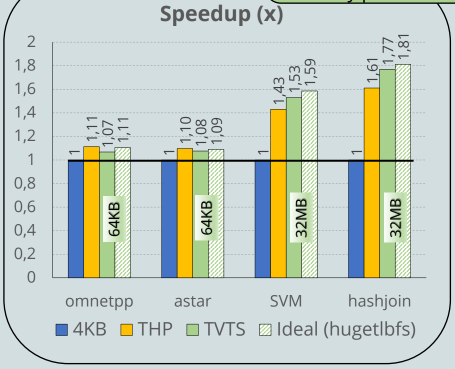
TVTS intermediate-sized translation promotion


VMA
3
VPN
Contiguous Translation Descriptors (Page Table Entries) :
1
Aligned PFN
0
PFN
Occupied Page
Promote to intermediate-sized translation via the contig bit

Implemented in Linux v5.18

Preliminary Results

Up to *16%* speedup on a real-world setup. Close to the ideal, without memory pre-allocation.



Work-in-progress

1. KVM support for contiguous translation descriptors.
2. Extend TVPS for virtualized execution.
3. Design an online page-size selection mechanism.
4. Study TVPS performance under fragmentation.
5. Add support for the RISC-V Svnapot extension.

[1] Chloe Alverti et al. "Enhancing and Exploiting Contiguity for Fast Memory Virtualization", ISCA'20