A comprehensive analysis of superpage management mechanisms and policies

Weixi Zhu, Alan L. Cox, Scott Rixner
{wxzhu, alc, rixner}@rice.edu
Department of Computer Science, Rice University

Superpages benefit large-memory Applications' performance

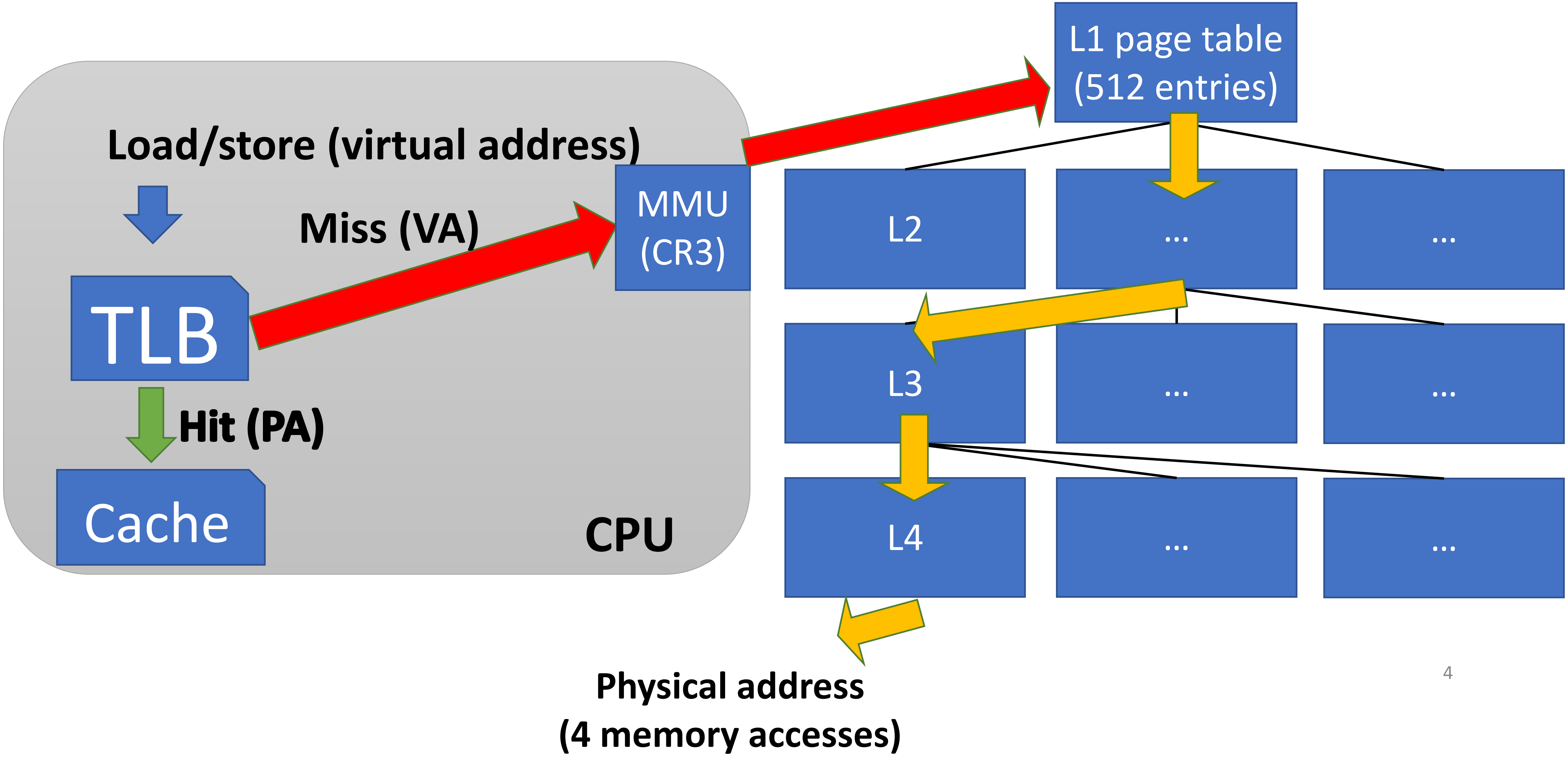
- Large memory applications have high address translation overhead
- Using superpages can reduce address translation overhead
- Many challenges in implementing transparent superpage support in the operating system – can cause performance regression

Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver

X86-64 4KB-page address translation



Translation Look-aside Buffers (TLBs)

- Caches 4KB/2MB page mappings
- Typical capacity: 1536 entries in Intel Skylake STLB
- Fewer TLB misses -> fewer page walks -> better performance

Benefits of Superpages (2MB)

Address translation benefits

- Cheaper page walk cost: 4 -> 3 memory accesses
- Significantly increased TLB coverage: 6MB -> 3GB
  - Intel Skylake STLB: 1536 * (~~4KB~~ 2MB) = ~~6MB~~ 3GB
  - Reduced # TLB misses (page walks) -> better performance

OS-level benefits

- Reduced number and average cost of page faults

Drawbacks of Superpages (2MB)

- Underutilization
  - Waste free memory, causing memory bloat
  - Waste CPU time preparing unused memory
- Allocation is easier to fail under fragmentation
  - Require 2MB-aligned free contiguous physical memory
- Latency spikes
  - Preparing a 2MB page (e.g. zeroing or disk-reading) is much more costly

Contributions of this paper

1. **Developed a comprehensive scheme for describing the design space**
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver

Five decoupled events of superpage lifetime

-- To help understand OS superpage management

Event	Definition
Physical allocation	Acquisition of a free physical superpage
Physical preparation	Incremental or full preparation of the initial data for an allocated physical superpage
Mapping creation	Creation of a virtual superpage in a process's address space and mapping it to a fully prepared physical superpage
Mapping destruction	Destruction of a virtual superpage mapping
Physical deallocation	Partial or full deallocation of an allocated physical superpage

Implementation choices

- Sync vs. Async allocation
  - During page fault time
  - When scanning page tables
- Incremental vs. full preparation
  - 4KB at a time
  - 2MB all at once
- In-place vs. out-of-place mapping (4KB->2MB promotion)
  - In-place promotion requires tracking allocated physical superpage
  - Out-of-place promotion involves migrating used pages to a different allocated physical superpage

Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. **Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye**
3. Proposed Quicksilver based on FreeBSD, driven by our novel insights

https://github.com/rice-systems/quicksilver

Existing designs in 5-event design space

Events	Linux	Ingens (Linux-based)	HawkEye (Linux-based)	FreeBSD
Allocation	Sync upon first page fault, or async for regions with utilization > 0. Defragmenting if necessary	Only async for regions with utilization > 90%, round-robin among processes	Only async for regions with utilization > 0, with fine-grained order	Upon first page fault (tracked by reservation system)
Preparation	Coupled with allocation, sync or async, full	Coupled with allocation, only async, full	Same as left	Incrementally prepares in-place 4KB pages on page faults
Mapping	Coupled with preparation, sync or async	Coupled with preparation. Async and out-of-place	Same as left	After the last page preparation. Sync and in-place
Unmapping	Upon freeing, partial or full mapping change	Same as left	Same as left	Same as left
Deallocation	Upon superpage unmapping	Same as left	Same as left	Deferred as long as possible

Observation #1: coupling physical allocation, preparation and mapping creation brings more drawbacks

System: Linux

Benefit: Immediate address translation benefits and fewer page faults

-- Best performance on freshly booted machine

Multiple Drawbacks:

- Easy to create underutilized superpages and bloat memory
- Fail to create superpages for growing heap, e.g. 602.gcc_s in SpecCPU-2017
  - Allocations will fail when the 2MB virtual region is not covered.
- Cannot easily choose between 2 superpage sizes, e.g. 64KB and 2MB in ARM
- Cannot extend to 1GB superpages or file-backed superpages (higher full preparation cost)

Observation #2: asynchronous out-of-place promotion delays superpage mapping creation

Systems: Ingens (Linux-based), HawkEye (Linux-based)

Benefit: Alleviate latency spikes from costly page faults

Drawbacks:

- Preparation involves costly page migrations (the asynchronously allocated superpage is out-of-place)
- Superpage mapping creation is delayed – much slower than in-place promotion (FreeBSD)

Speedups	Linux	Ingens	HawkEye	FreeBSD
GraphChi: PageRank	1	0.58	0.53	0.77
BlockSVM: classification	1	0.81	0.73	0.96

Observation #3: Reservation-based policies enables speculative physical allocation, multiple page sizes and in-place promotion

System: FreeBSD

Requirement: A reservation system that tracks allocated physical superpages

Benefits:

- Decoupled allocation and preparation – enables speculative allocation for growing heaps (602.gcc_s), incremental preparation and in-place promotion
- Obviating need of async out-of-place promotion – can allocate physical superpages for growing heaps
- Supporting multiple page sizes

Observation #4: Reservations and delaying partial deallocation fight fragmentation

System: FreeBSD

Benefit:

- Less memory fragmentation from delayed partial deallocation – individual 4KB pages are less likely reallocated for other purpose
- No latency spikes – Linux's memory compaction during page faults result in latency spikes in server workloads.

Observation #5: Bulk zeroing is consistently more efficient on modern processors

Typical zeroing: 512 calls of zeroing assembly code with size of 4KB

Bulk zeroing: Fewer calls of zeroing assembly code with bulk size > 4KB

Latency (us) of 2MB zeroing: drops consistently with larger bulk sizes

CPU (GHz)	temporal			Non-temporal		
Bulk Size	4KB	32KB	2MB	4KB	32KB	2MB
E3-1231v3 (3.4)	92	88	87	114	99	97
E3-1245v6 (3.7)	84	67	65	92	74	71
E5-2640v3 (2.6)	355	287	280	154	112	106
E5-2640v4 (2.4)	409	334	325	163	113	106
R7-2700X (4.3)	185	183	159	99	60	53

Contributions of this paper

1. Developed a comprehensive scheme for describing the design space
2. Presented novel insights from existing systems – Linux, FreeBSD, Ingens and HawkEye
3. **Proposed Quicksilver based on FreeBSD, driven by our novel insights**

https://github.com/rice-systems/quicksilver

Quicksilver – guided by novel observations

- Allocation: allocates a reservation speculatively upon first page fault
- Preparation: incrementally prepares 4KB on demand, performs a synchronous full preparation upon a utilization threshold (Sync-1, Sync-64) – match or beat Linux's performance
- Mapping: Relaxed for more file-backed mappings
- Unmapping: same as FreeBSD
- Deallocation: delayed until the superpage is inactive, then asynchronously evicts 4KB pages to perform a whole deallocation

Evaluation of Quicksilver

- Performance of a wide variety of workloads
  - on a freshly booted machine
  - on a heavily fragmented machine
- Throughput and tail latency of server workloads
- A parallel compilation task with many small jobs

Quicksilver Beats Linux on a freshly-booted machine

Frag-0	GUPS	Graphchi-PR	BlockSVM	XSBench	ANN	Canneal	Freqmine	Gcc	mcf	Dsjeng	XZ
Linux	1	1	1	1	1	1	1	1	1	1	1
Ingens	0.87	0.58	0.81	0.98	1	0.95	0.99	1	0.99	0.99	0.96
HawkEye	0.28	0.53	0.73	0.88	1	0.95	0.99	0.99	0.94	0.86	0.9
FreeBSD	0.96	0.77	0.96	0.99	0.98	1.14	1	1.05	0.99	1	0.99
Sync-1	0.99	**1.07**	1	1	**1.07**	**1.14**	0.99	**1.05**	1	1	1
Sync-64	0.98	**1.05**	1	1	**1.08**	**1.14**	0.99	**1.05**	1	1	1

Linux is no longer the best on a freshly-booted machine!

Quicksilver outperforms every other systems under severe memory fragmentation

Frag-100	GUPS	Graphchi-PR	BlockSVM	XSBench	ANN	Canneal	Freqmine	Gcc	mcf	Dsjeng	XZ
Linux	1	1	1	1	1	1	1	1	1	1	1
Ingens	1.02	1.13	0.86	1.04	1	1	1	1	1.01	1.01	1.02
HawkEye	0.97	1.11	0.85	1.03	1	1.01	1	1	0.99	0.97	1.02
FreeBSD	0.96	1.1	0.85	1.04	0.98	1.05	1	1	1	1.04	1.02
Sync-1	**2.35**	**2.18**	**1.12**	**1.07**	**1.04**	**1.12**	1	**1.05**	**1.02**	**1.1**	**1.14**
Sync-64	**2.29**	**2.11**	**1.13**	**1.07**	**1.01**	**1.12**	1	**1.05**	**1.05**	**1.11**	**1.14**

2.18x speedup on PageRank task!

Quicksilver obtains high throughput without latency spikes

Throughput (GBps) and 95th tail latency (ms) of Redis workloads

Cold-start	Linux-4KB	Linux	Ingens	HawkEye	FreeBSD	Sync-1	Sync-64
Frag-0	1.04 (5.6)	**1.34 (4.1)**	1.00 (5.9)	1.00 (5.9)	1.11 (6.1)	1.26 (4.5)	1.20 (4.8)
Frag-50	1.04 (5.7)	0.92 (10.2)	0.95 (5.9)	1.02 (5.9)	1.04 (6.2)	**1.25 (4.5)**	**1.27 (4.7)**
Frag-100	1.07 (5.6)	0.81 (9.9)	0.94 (6.1)	1.00 (5.8)	0.98 (6.5)	**1.31 (4.5)**	**1.26 (4.6)**

Warm-start	Linux-4KB	Linux	Ingens	HawkEye	FreeBSD	Sync-1	Sync-64
Frag-0	1.06 (6.5)	**1.32 (5.2)**	1.23 (5.7)	1.03 (6.7)	**1.30 (5.6)**	**1.32 (5.5)**	**1.31 (5.5)**
Frag-50	1.07 (6.5)	1.17 (5.9)	1.09 (6.4)	1.03 (6.7)	1.18 (6.1)	**1.32 (5.5)**	**1.32 (5.5)**
Frag-100	1.07 (6.5)	1.16 (5.9)	1.01 (6.9)	1.05 (6.6)	1.10 (6.6)	**1.33 (5.4)**	**1.34 (5.5)**

Also observe low memory bloat on Quicksilver

Quicksilver (Sync-64) avoids creating underutilized superpages

FreeBSD Kernel compilation task (make buildkernel –j9):

Buildkernel	real	user	sys	# superpages	# page faults
Sync-1	197.7	1409.4	89.4	200.5 K	5.3 M
Sync-64	196.9	1408.8	78.5	99.6 K	10.3 M
FreeBSD	203.7	1436.7	98	36.9 K	30.2 M

Sync-1 creates 100.9 K underutilized superpages with average utilization < 50 4KB pages
-- Sync-64 is as competitive as Sync-1, but also avoids underutilized superpages

Takeaways from this paper

- Our comprehensive scheme allow comparing and contrasting superpage management policies
- Our novel insights motivated Quicksilver's innovative design
- Quicksilver obtains benefits of aggressive superpage allocation, with mitigated memory bloat and fragmentation issues that arise from underutilized superpages
- Sync-64 and Sync-1 can both match or beat existing systems in both lightly and heavily fragmented scenarios, in terms of application performance, tail latency and memory bloat
- Sync-64 avoids creating underutilized superpages and is preferable for long-running servers

Thank you

For more details, please check our ATC-2020 paper.

Quicksilver source code: https://github.com/rice-systems/quicksilver