Managing Distributed, Shared L2 Caches through OS-Level Page Allocation

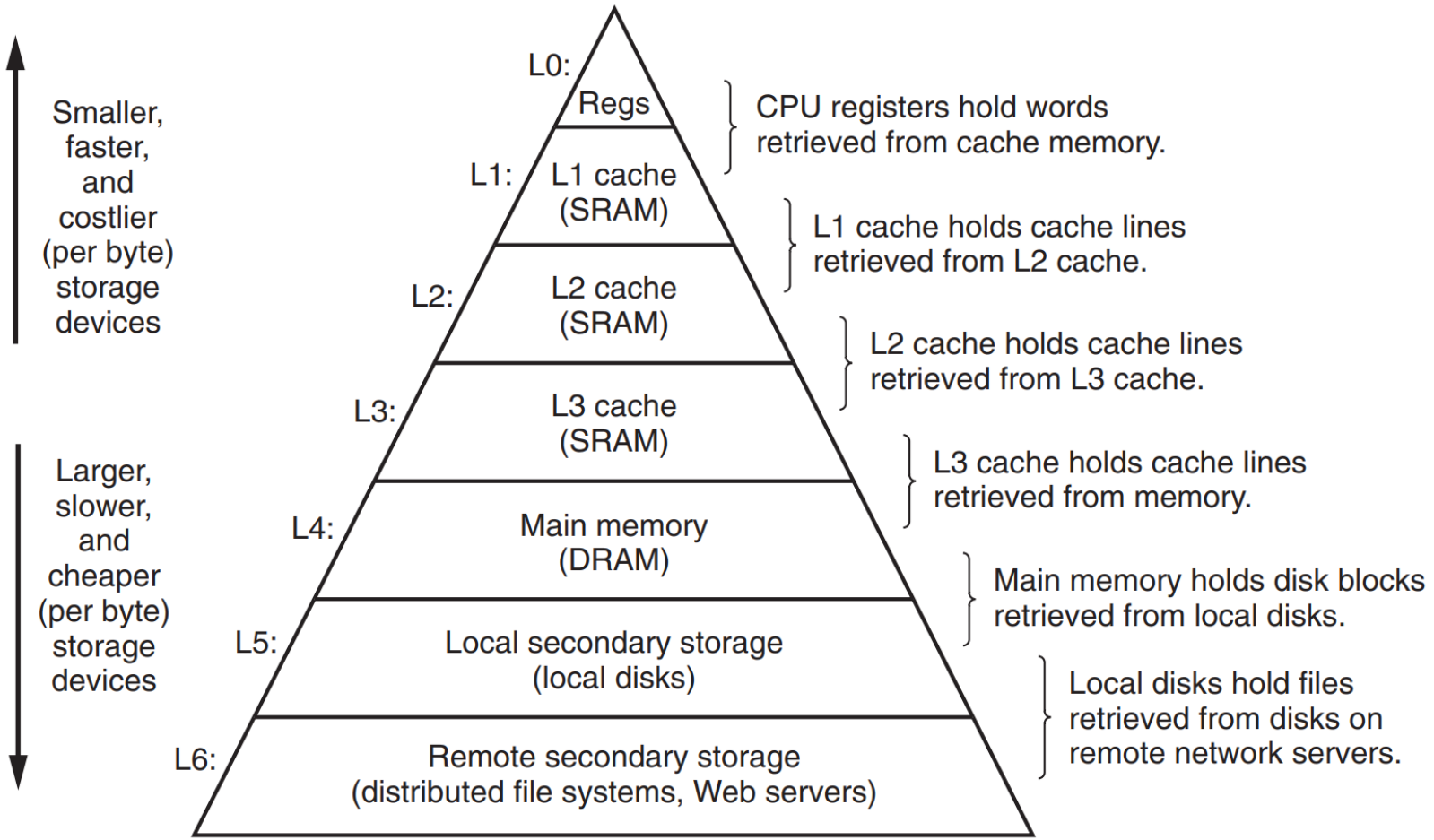
January 21, 2020

In the last episode...

Problem: CPUs are cheap and fast.
Memories are cheap, fast, capacious (choose two out of three).
Fast CPU with slow memory doesn't make sense.

Idea: Put fast memory between CPU and main memory.
Fast – so small.

CPU looks for data in the cache; if it finds it, it gets it from there and continues to work; if it doesn't find it, it looks for it in main memory.


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
CPU registers hold words retrieved from cache memory.
L1:
L1 cache (SRAM)
L1 cache holds cache lines retrieved from L2 cache.
L2:
L2 cache (SRAM)
L2 cache holds cache lines retrieved from L3 cache.
L3:
L3 cache (SRAM)
L3 cache holds cache lines retrieved from memory.
L4:
Main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L5:
Local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers.
L6:
Remote secondary storage (distributed file systems, Web servers)

memory	latency	size
L1	4 cycles	32KiB
L2	10 cycles	256KiB
L3	40-75 cycles	8MiB
DRAM	600 cycles	8GiB
HDD	∞	∞


INTEL®©'01 PHILIPPINES
1133/256/133/1.475
A4484T 01 S2
7130B208-0273
PENTIUM III SL5GQ

It was working

It was working; but now...


AMD
EPYC

How to live?

How to live?

Private cache?

How to live?

Private cache? Shared cache?

Private cache

Private cache

Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency
Cons: capacity misses

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency
Cons: capacity misses

Shared cache

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency
Cons: capacity misses

Shared cache
Single cache where each cache slice accepts only an exclusive subset of all memory blocks.

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency
Cons: capacity misses

Shared cache
Single cache where each cache slice accepts only an exclusive subset of all memory blocks.

Pros: better overall utilization of on-chip caching capacity, enforcing cache co-herence becomes simpler

Private cache
Each cache slice is associated with a specific processor core and replicates data freely as the processor accesses them.

Pros: low hit latency
Cons: capacity misses

Shared cache
Single cache where each cache slice accepts only an exclusive subset of all memory blocks.

Pros: better overall utilization of on-chip caching capacity, enforcing cache co-herence becomes simpler
Cons: cache hit latency will be longer

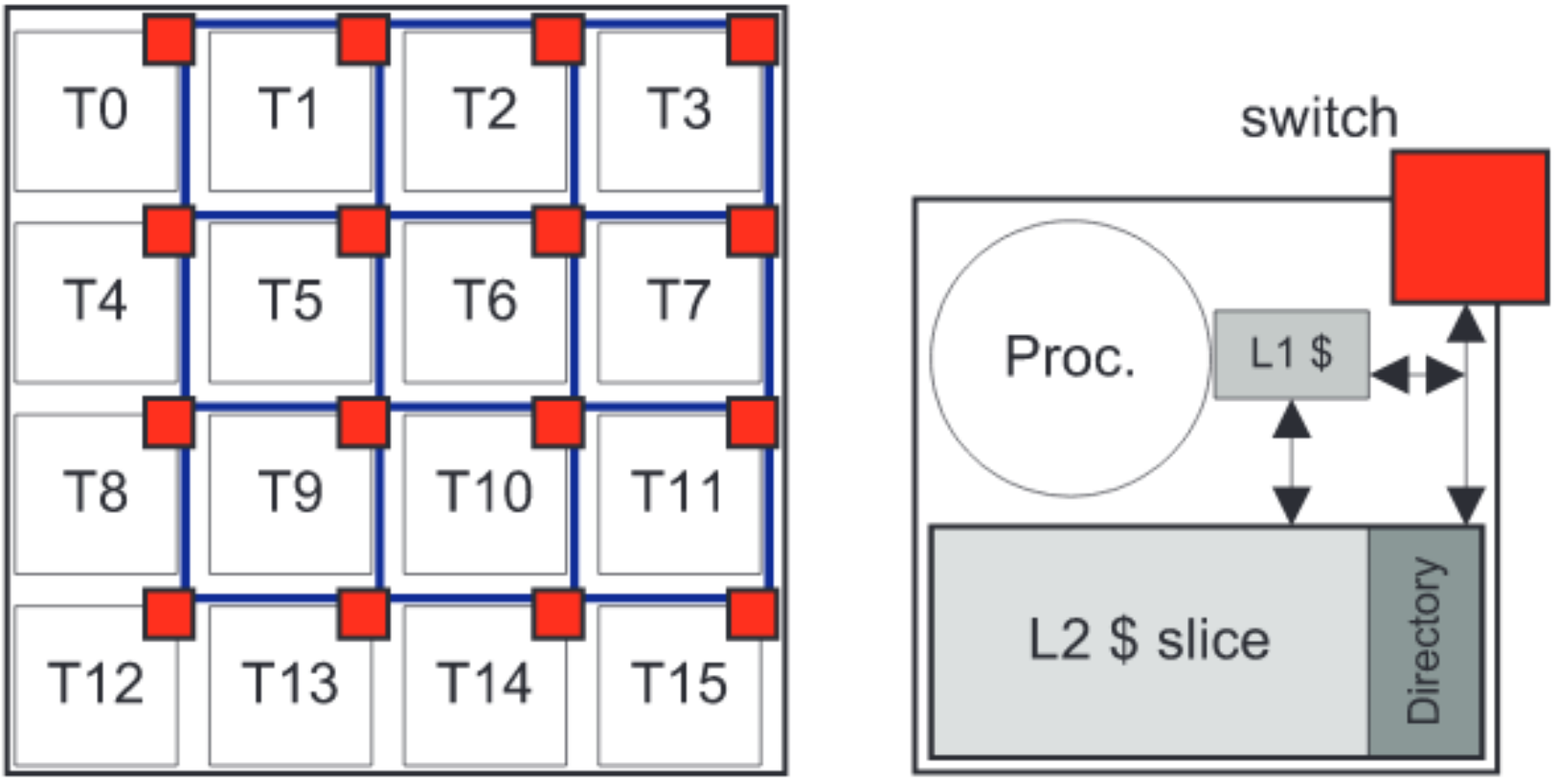
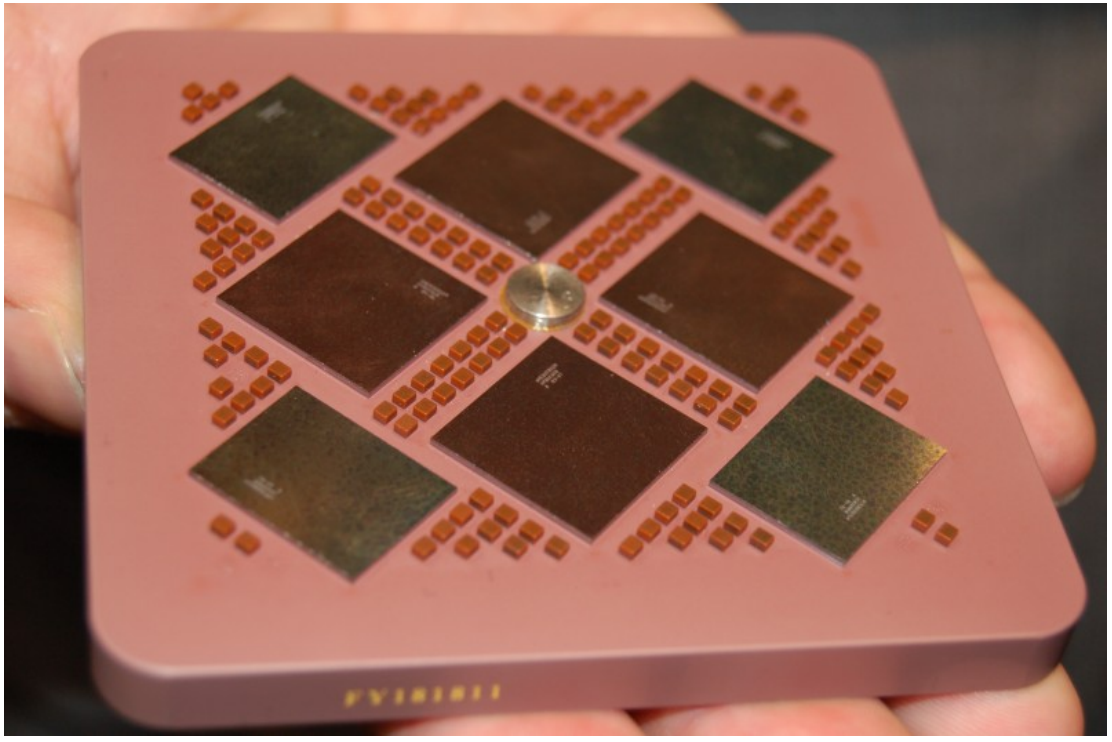


Figure 1. An example 16-core tiled processor chip and its tile (core).

Idea: give OS control over cache.

Idea: give OS control over cache.
Using simple shared cache hardware we can implement a private caching policy, a shared cache policy, or a hybrid of the two

Idea: give OS control over cache.
Using simple shared cache hardware we can implement a private caching policy, a shared cache policy, or a hybrid of the two **without any hardware support**.



IBM Power 5!

$$S = A \bmod N$$

$$S = A \bmod N$$

Where S stands for the cache slice number, A for the memory block address, and N for the number of cache slices.

Problem: contiguous memory blocks are hosted by different cache slices

Problem: contiguous memory blocks are hosted by different cache slices
Solution: replacing A with physical page number (PPN)

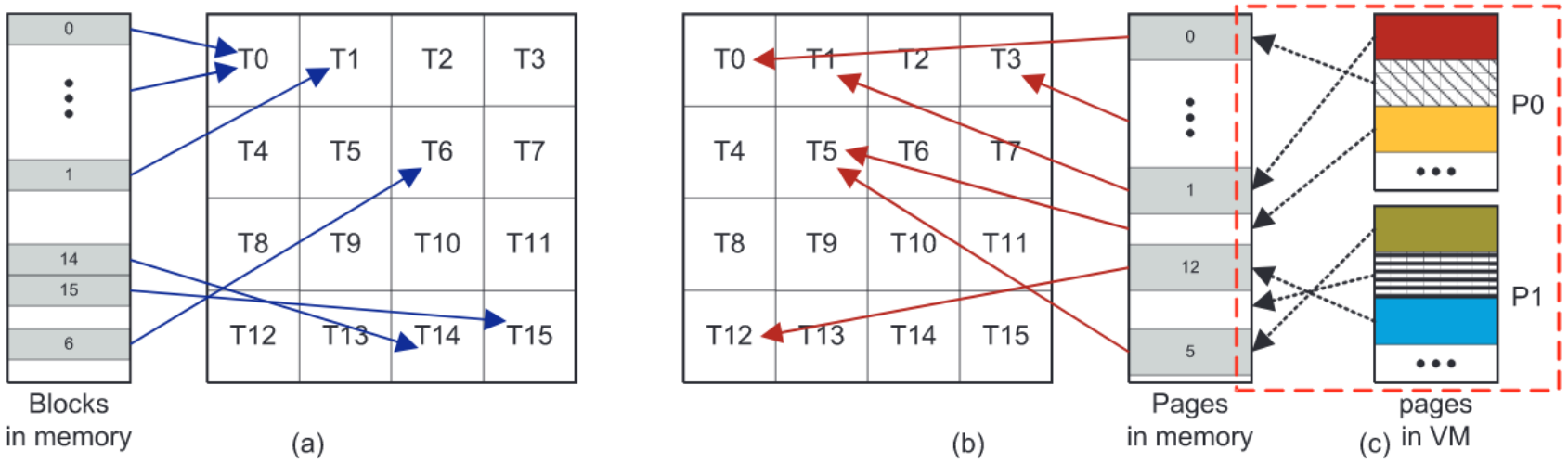


Figure 2. (a) Physical memory partitioning and mapping to cache slice at the cache line granularity. (b) Physical memory partitioning and mapping at the memory page granularity. (c) Virtual to physical page mapping (P0, P1: process 0 and 1, VM: virtual memory).

CG_i $(0 < i < N - 1)$ – congruence group
$CG_i = \{$ physical page $(\text{PPN} = j) \mid \text{pmap}(j) = i\}$

OS has control on pmap.

CG_i $(0 < i < N - 1)$ – congruence group
$CG_i = \{$ physical page $(\text{PPN} = j) \mid \text{pmap}(j) = i\}$

OS has control on pmap.

Private caching: for a page requested by a program running on core i, allocate a free page from CG_i

CG_i $(0 < i < N - 1)$ – congruence group
$CG_i = \{$ physical page $(\text{PPN} = j) \mid \text{pmap}(j) = i\}$

OS has control on pmap.

Private caching: for a page requested by a program running on core i, allocate a free page from CG_i

Shared caching: for a requested page, allocate a free page from all the congruence groups using random selection, round-robin etc...

CG_i $(0 < i < N - 1)$ – congruence group
$CG_i = \{$ physical page $(\text{PPN} = j) \mid \text{pmap}(j) = i\}$

OS has control on pmap.

Private caching: for a page requested by a program running on core i, allocate a free page from CG_i

Shared caching: for a requested page, allocate a free page from all the congruence groups using random selection, round-robin etc...

Hybrid caching: partition $\{CG_i\}$ into K groups $(K < N)$; then define a mapping from a core to a group; for page requested by program running on core i, allocate a free page from the group that core i maps to

Figure 3. Program ("P") and data locations determine the minimum distance to bridge them.

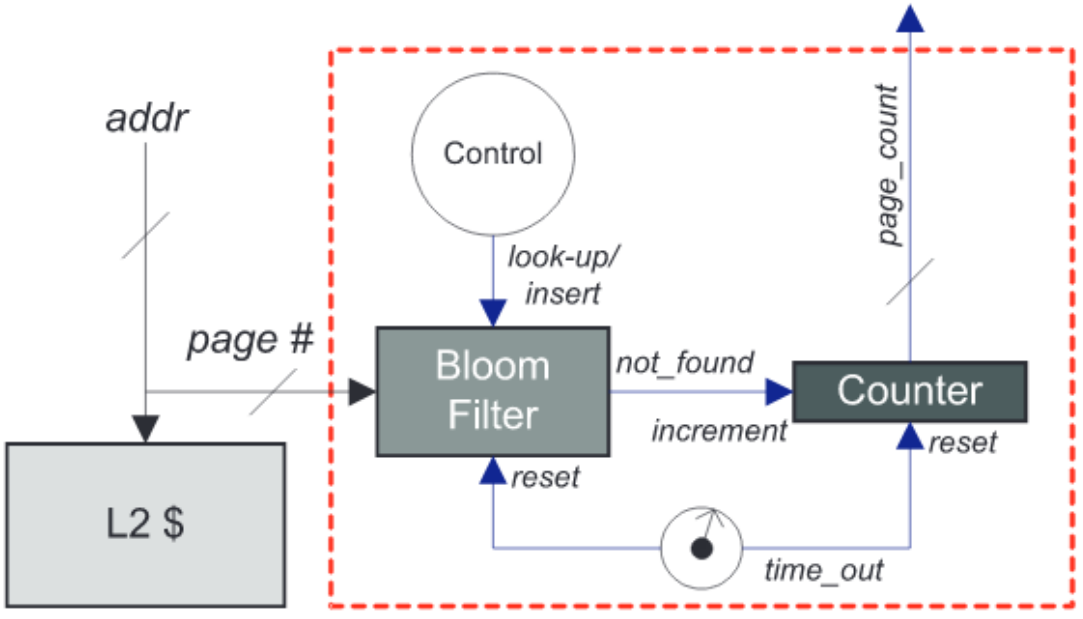


Figure 4. A Bloom filter based monitor mechanism to count actively accessed pages.

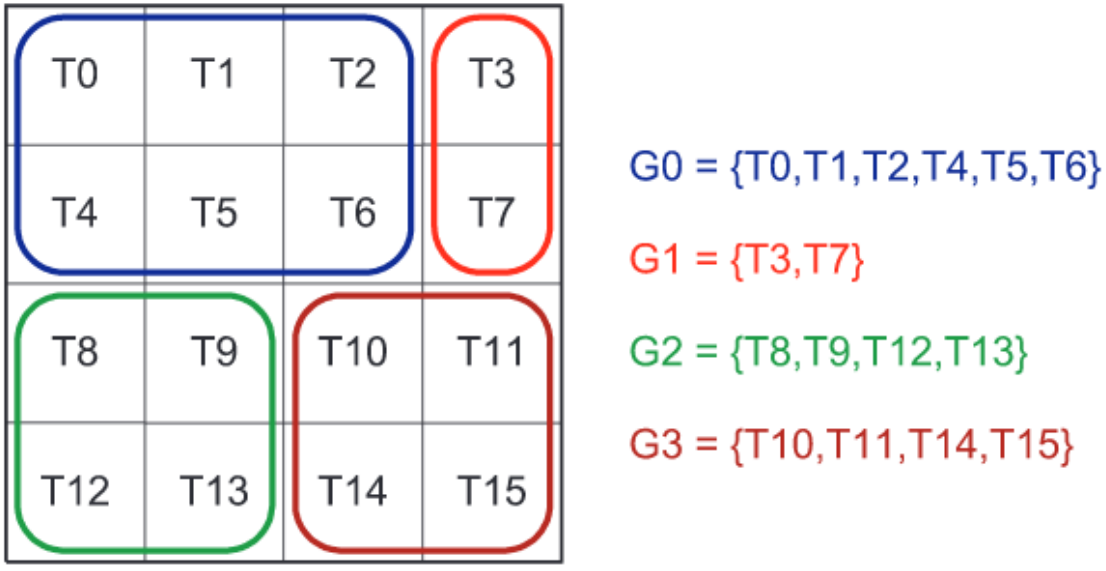


Figure 5. A virtual multicore (VM) example.

	PRIVATE	SHARED (w/ line interleaving)	OS-BASED
Hardware (tile)	Similar to a conventional uniprocessor core with two-level caches; coherence mechanism (*e.g.*, directory) to cover L1 and L2 caches	Coherence enforcement is simpler and is mainly for L1 because L2 is shared by all cores	(Same as SHARED); simple hardware-based performance monitoring mechanism will help reduce monitoring overhead
Software	(NA)	(NA)	OS-level support, esp. in the page allocation algorithm
Data Proximity	Data items are attracted to local cache slices through active replication; limited caching space can result in performance degradation due to capacity misses	Fine-grained cache line interleaving results in non-optimal data distribution; there is no explicit control over data mapping	Judicious data mapping though page allocation can improve data proximity
Network Traffic	High coherence traffic (*e.g.*, directory look-up and invalidation) due to data replication [23]; increased off-chip traffic due to high on-chip miss rate	High inter-tile data traffic due to remote L2 cache accesses, often 2× to 10× higher than PRIVATE [30]; lower off-chip traffic due to larger caching capacity	Low off-chip traffic like SHARED; improved program-data proximity through page allocation and process scheduling leads to lower inter-tile traffic than SHARED

Table 1. Comparing private caching, shared caching, and OS-based cache management approaches.

Test setup

CPU: 16 tiles (4x4); 16kB L1 I/D caches, 512kB L2 cache slice

L1 caches are foure-way set associative with a 32-byte line size

Each 8-cycle L2 cache slice is 8-way set associative with 128-byte lines, two-cycle latency per each hop

2-GB off-chip main memory with 300 cycles latency

NAME	DESCRIPTION	INPUT
gcc	gcc compiler	reference (`integrate.i`)
parser	English parser	reference
eon	probabilistic ray tracer	reference (chair)
twolf	place & route simulator	reference
wupwise	quantum chromodynamics solver	reference
galgel	computational fluid dynamics	reference
ammp	ODE solver for molecular dynamics	reference
sixtrack	particle tracking for accelerator design	reference
fft	fast Fourier transform	4M complex numbers
lu	dense matrix factorization	512×512 matrix
radix	parallel radix sort	3M integers
ocean	ocean simulator	258×258 grid

Table 2. Benchmark programs.

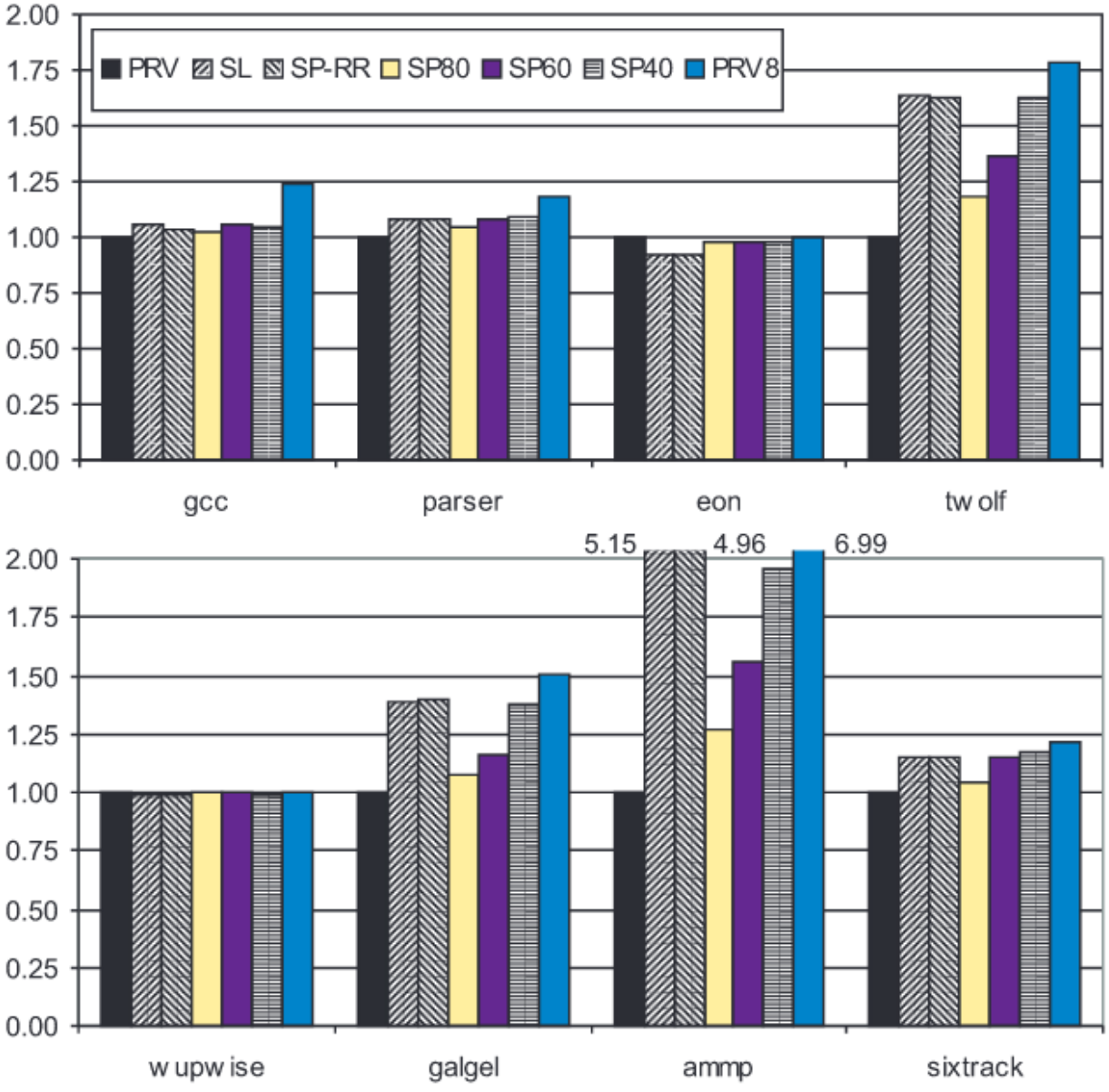


Figure 6. Single program performance ($time^{-1}$) of different policies, relative to PRV.

		PRV	*SL*	*SP-RR*	*SP40*	*SP60*	*SP80*	*PRV8*
load miss rate (%)	*gcc*	2.9	0.1	0.5	1.8	2.1	2.8	0.1
	parser	6.6	0.5	0.6	2.6	3.7	5.8	0.4
	eon	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	twolf	16.3	0.1	0.1	1.6	7.3	13.1	0.0
	wupwise	25.0	25.0	25.0	25.0	25.0	25.0	25.0
	galgel	6.3	0.1	0.1	0.9	3.4	5.0	0.1
	ammp	46.6	0.1	0.4	18.9	26.4	34.9	0.1
	sixtrack	13.5	0.5	0.5	1.4	3.2	10.4	0.5
on-chip network traffic	*gcc*	10.8	270.4	261.7	135.9	76.0	55.6	0.4
	parser	8.7	96.8	96.5	40.4	18.9	18.2	0.5
	eon	0.0	86.9	90.2	23.7	20.4	17.7	0.0
	twolf	35.0	138.2	150.1	67.8	48.4	37.8	0.1
	wupwise	35.1	39.4	39.9	20.3	15.6	10.1	0.1
	galgel	38.0	412.0	406.6	185.8	132.3	76.2	0.6
	ammp	441.7	810.9	803.4	424.6	361.9	306.9	0.5
	sixtrack	9.6	57.2	60.9	22.0	18.9	15.8	0.4

Table 3. L2 cache load miss rate and on-chip network traffic (message-hops) per 1k instructions.

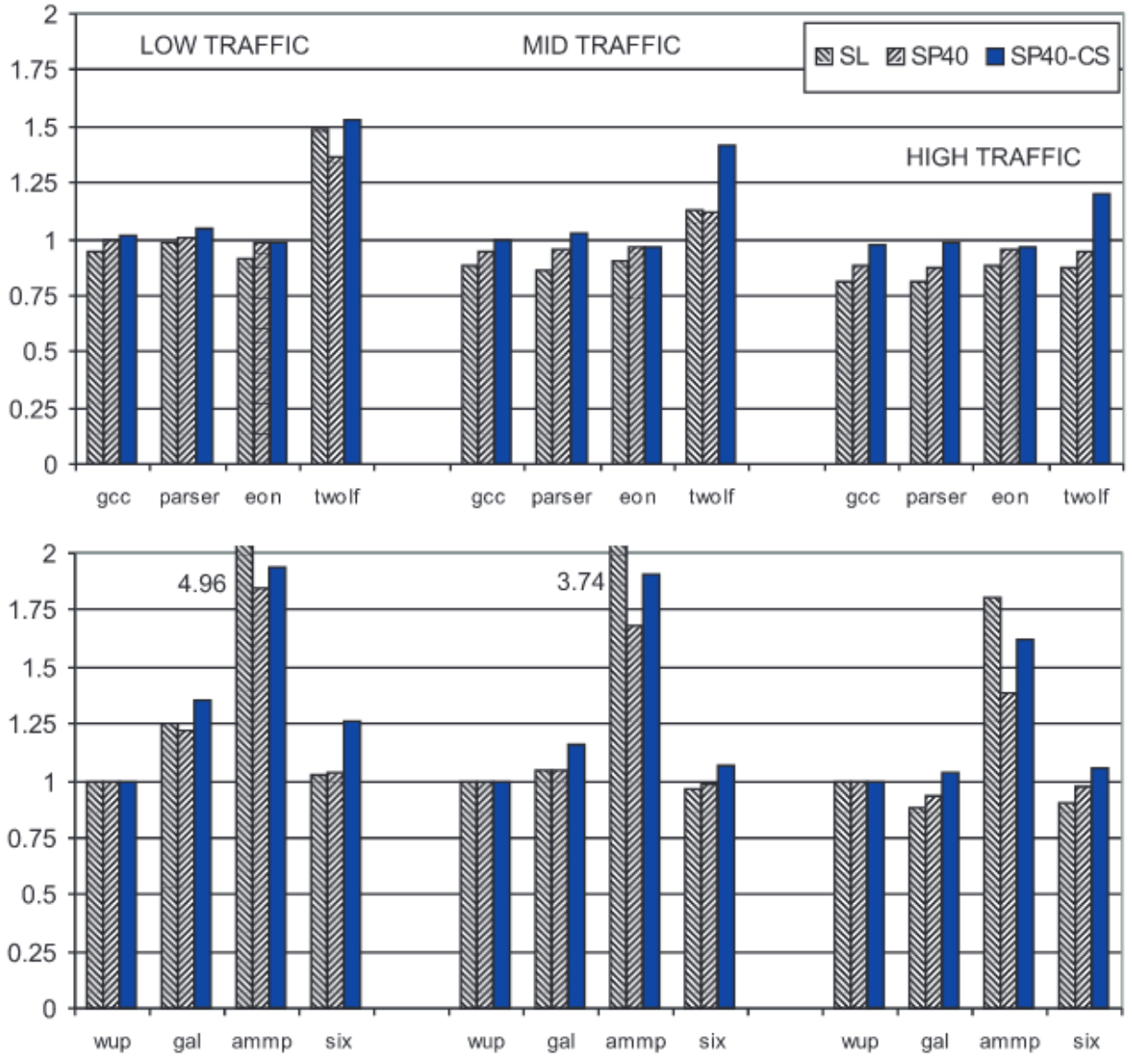


Figure 7. Performance sensitivity to network traffic. Performance relative to PRV, no traffic case.

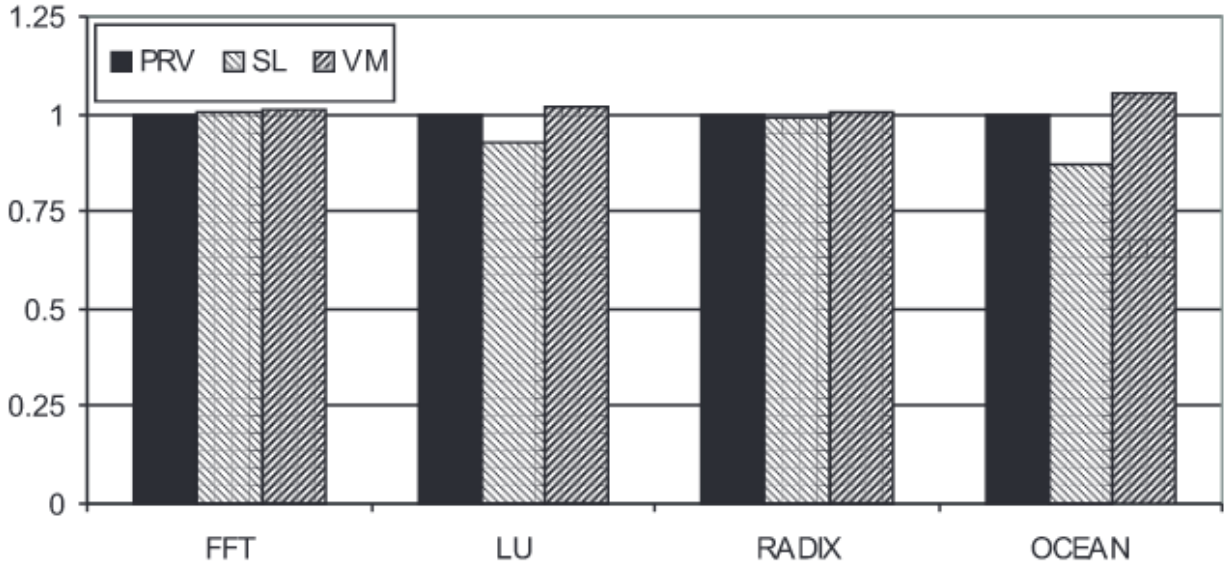


Figure 8. Performance of parallel workloads, relative to PRV.

(standing ovation)