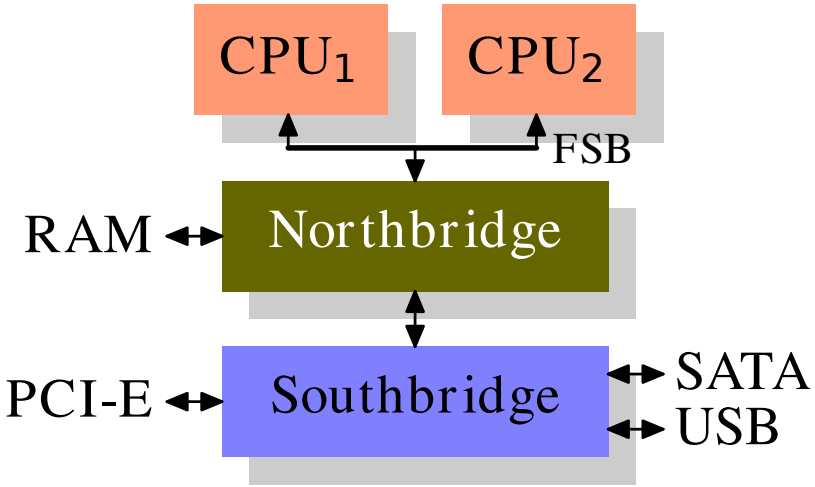
Ulrich Drepper: What Every Programmer Should Know About Memory

Motto

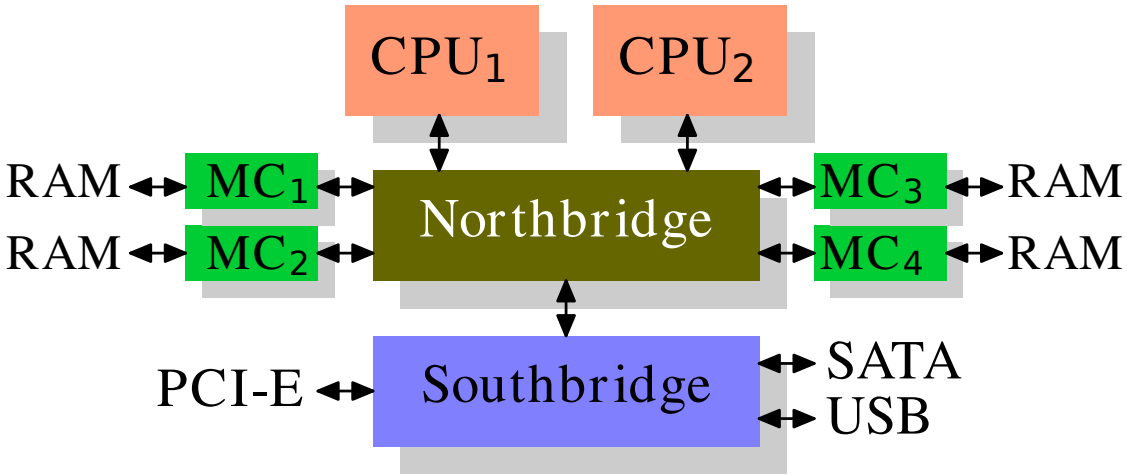
memory access = the limiting factor for most programs

Table of contents

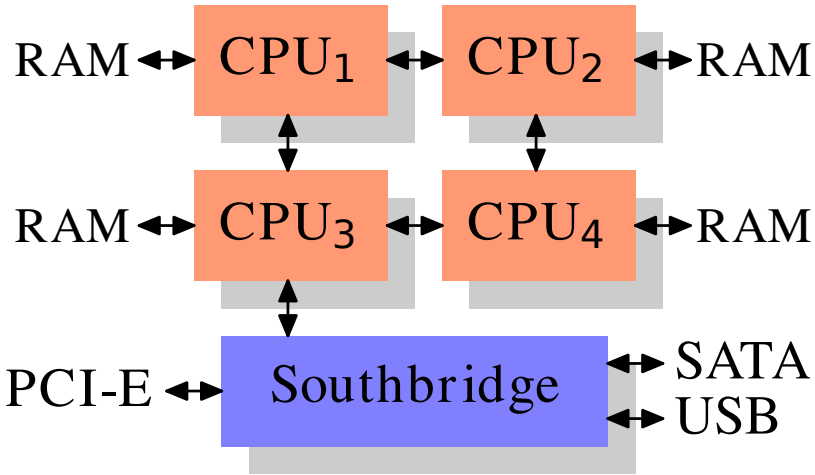
Chipset: Northbridge and Southbridge



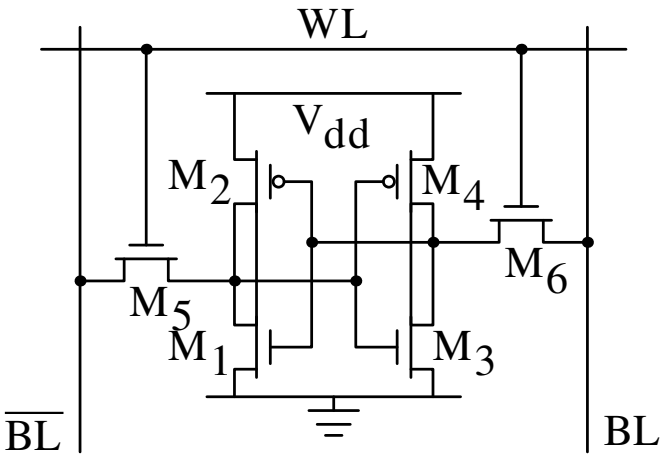
Chipset: Northbridge with External Controllers

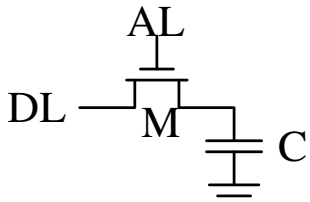


Chipset: Integrated Memory Controller

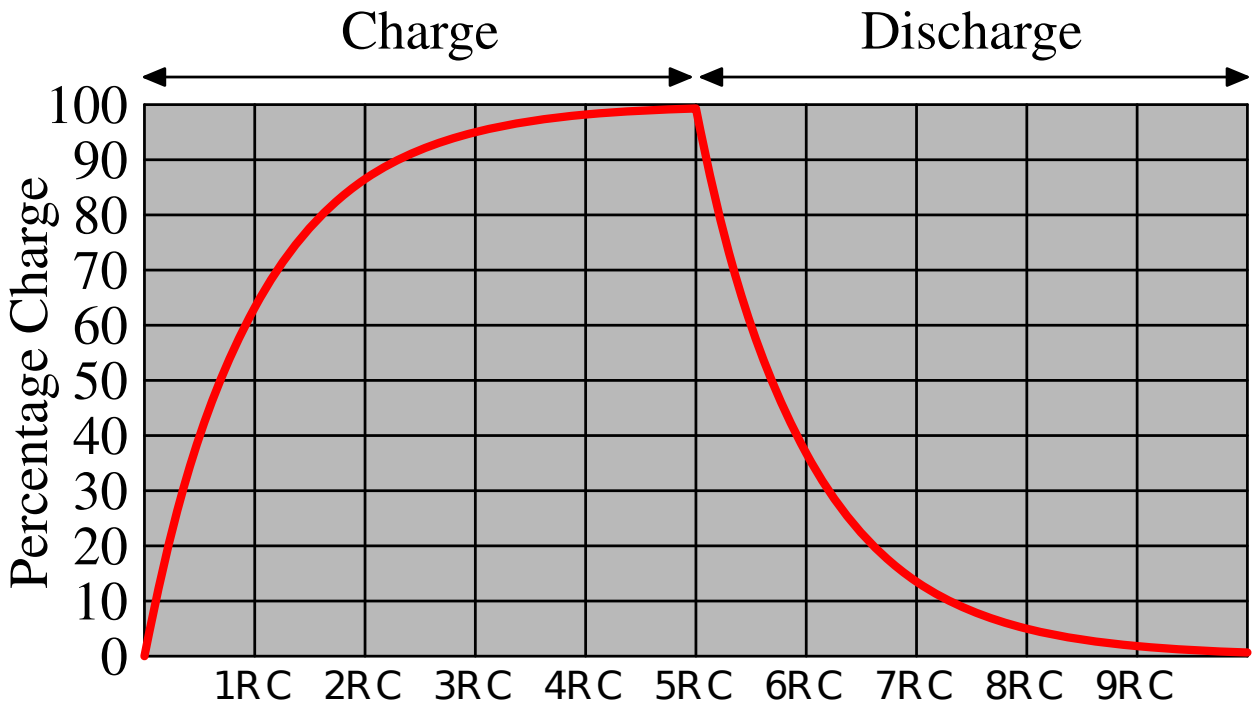


RAM Types: SRAM, DRAM

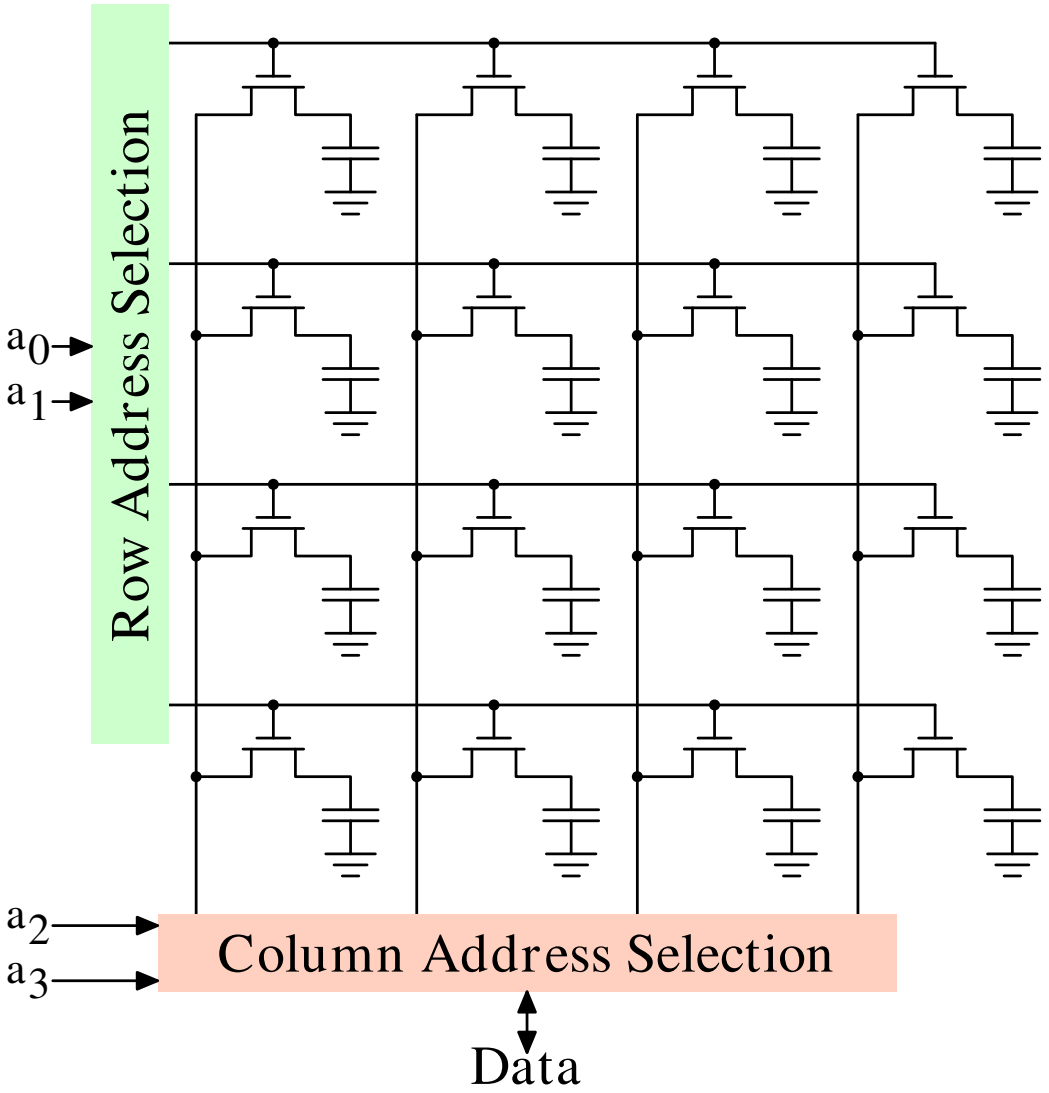




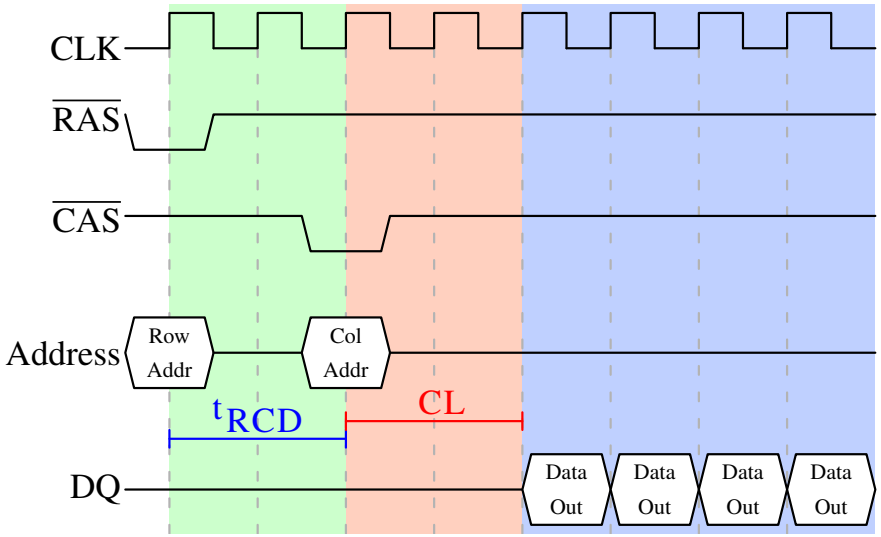
DRAM: Capacitor Charge and Discharge Timing

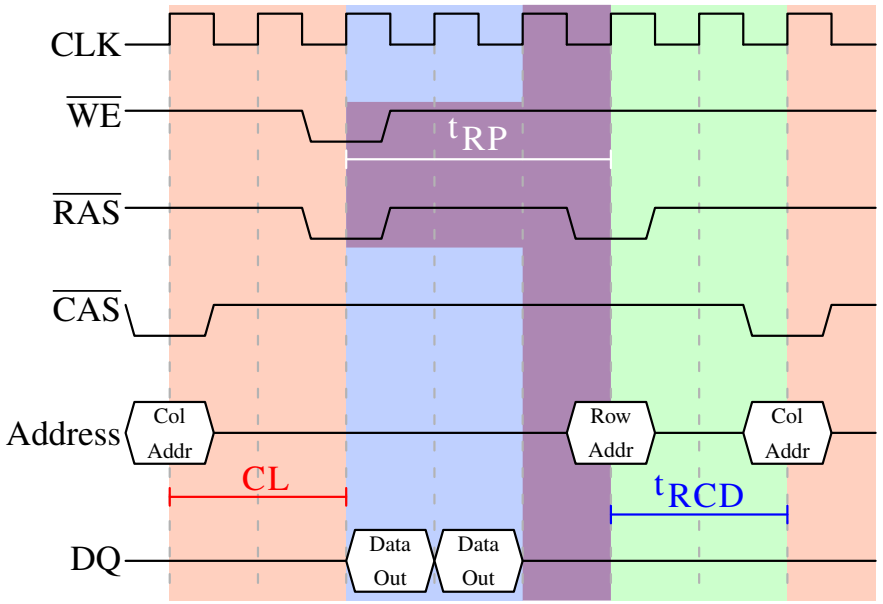


DRAM: Schematic, RAS, CAS



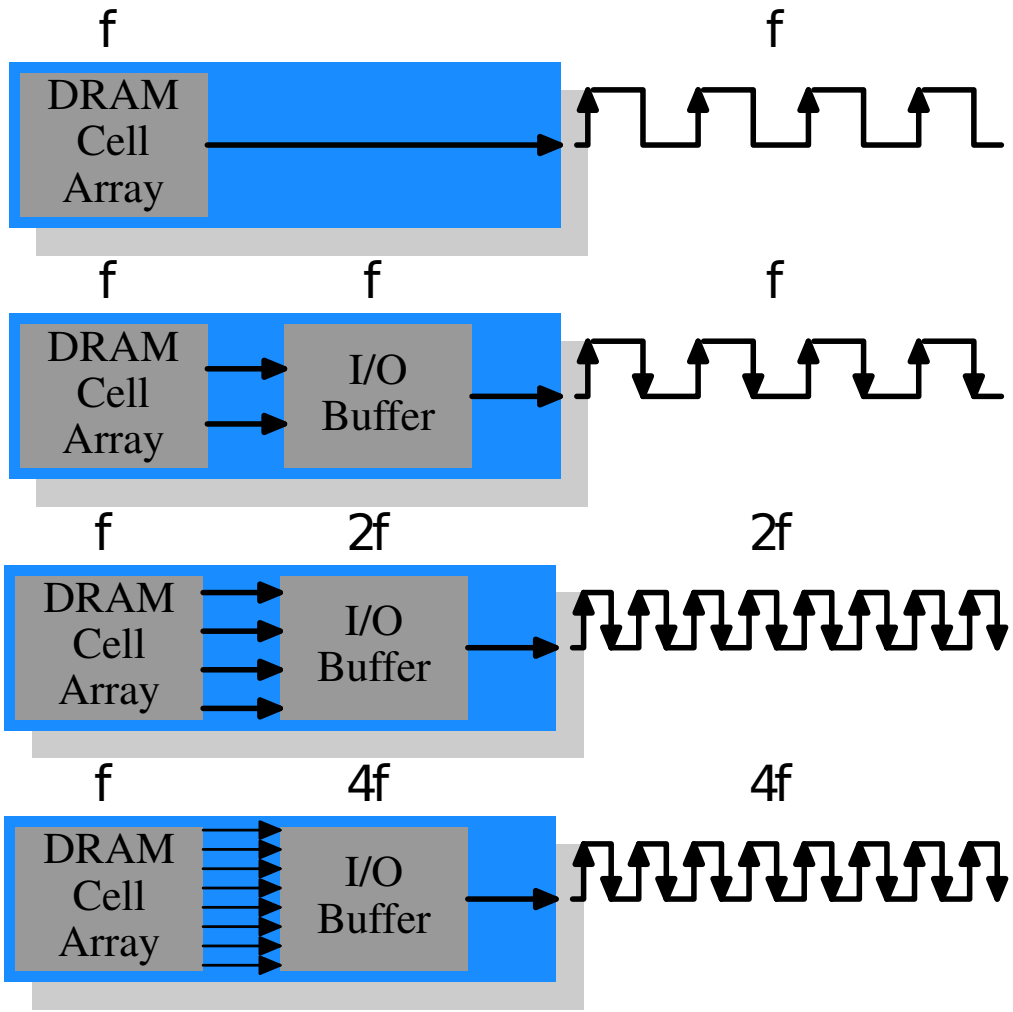
SDRAM: Read Access Timing, Precharge and Activation



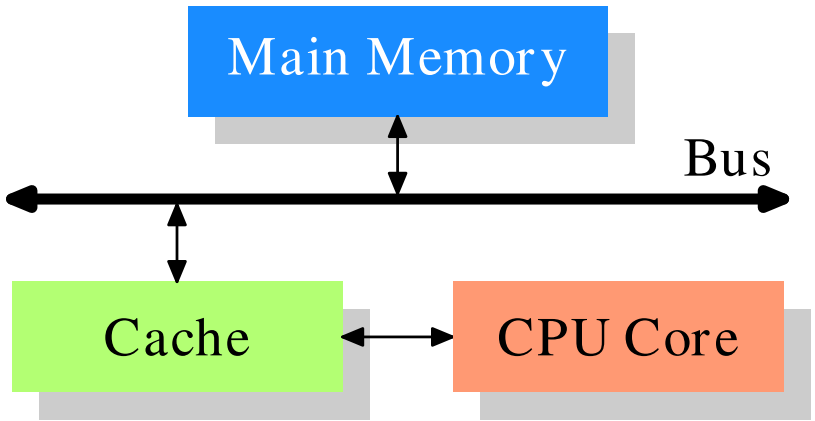


burst mode

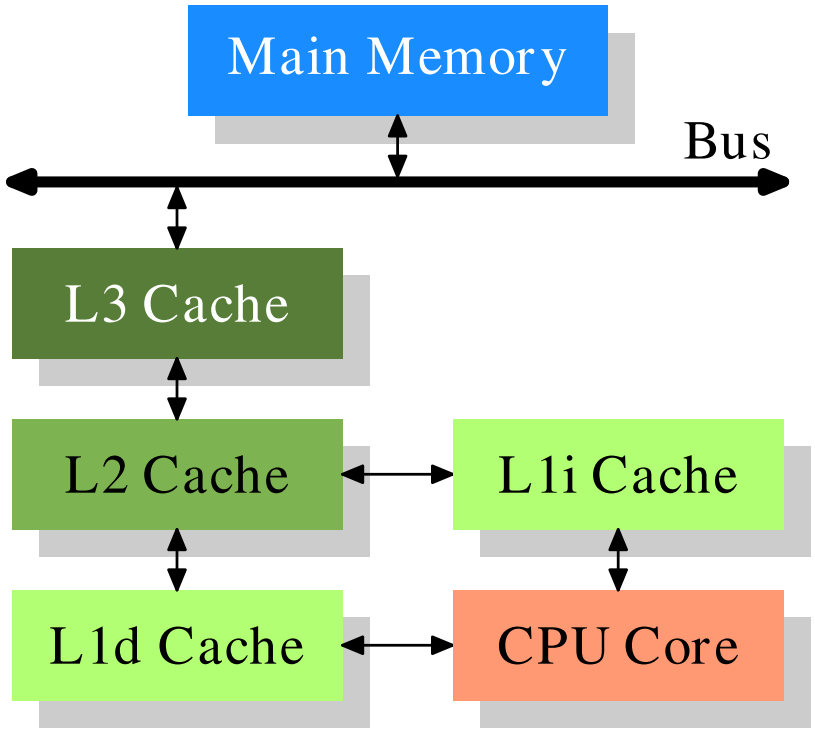
SDRAM: SDR, DDR1, DDR2, DDR3



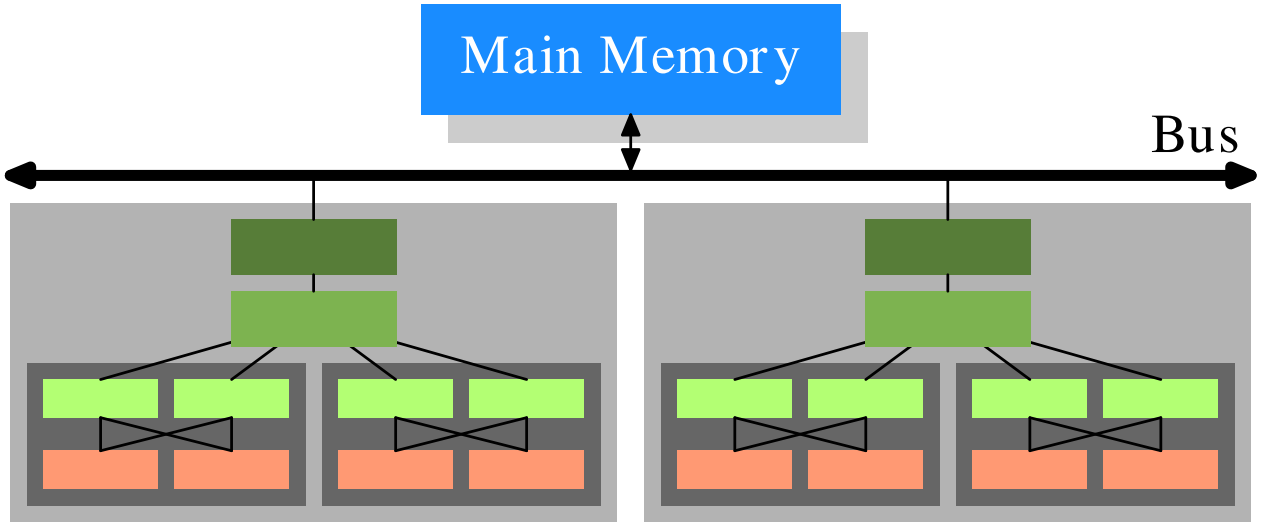
Minimum Cache Configuration



Processor with Level 3 Cache



Multi processor, multi-core, multi-thread



Cache

caches

- L1d, L1i (decoded), L2, L3
  L1 (virt.), L2, L3 (phys.)
- TLB

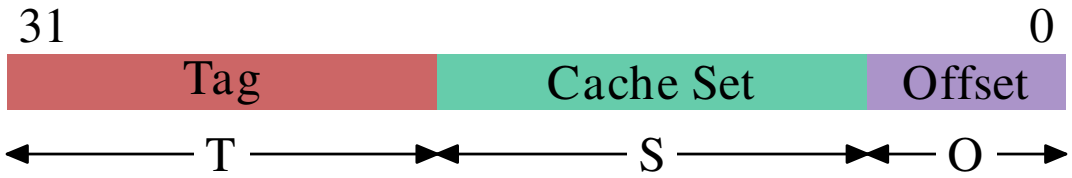
policies

- write-through
- write-back
- write-combining
- uncacheable

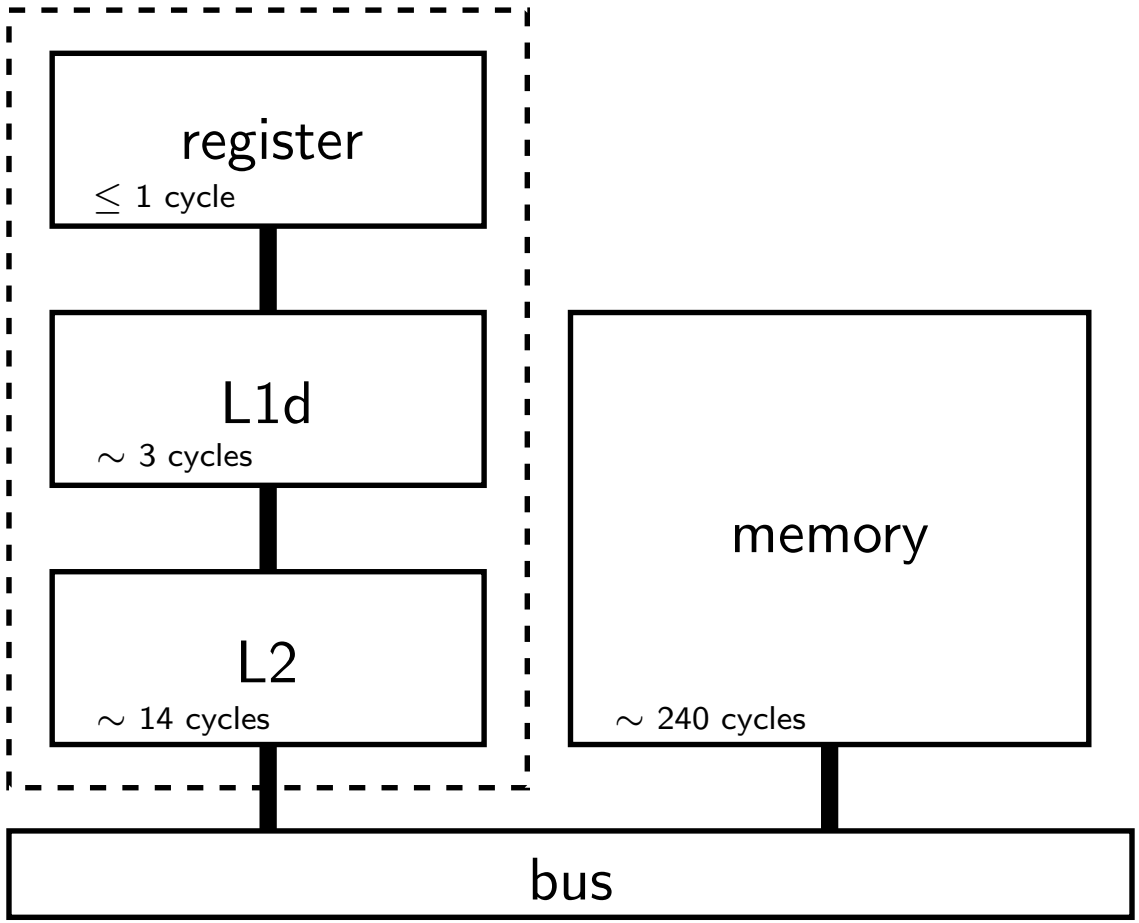
multi-level caches

- exclusive
- inclusive

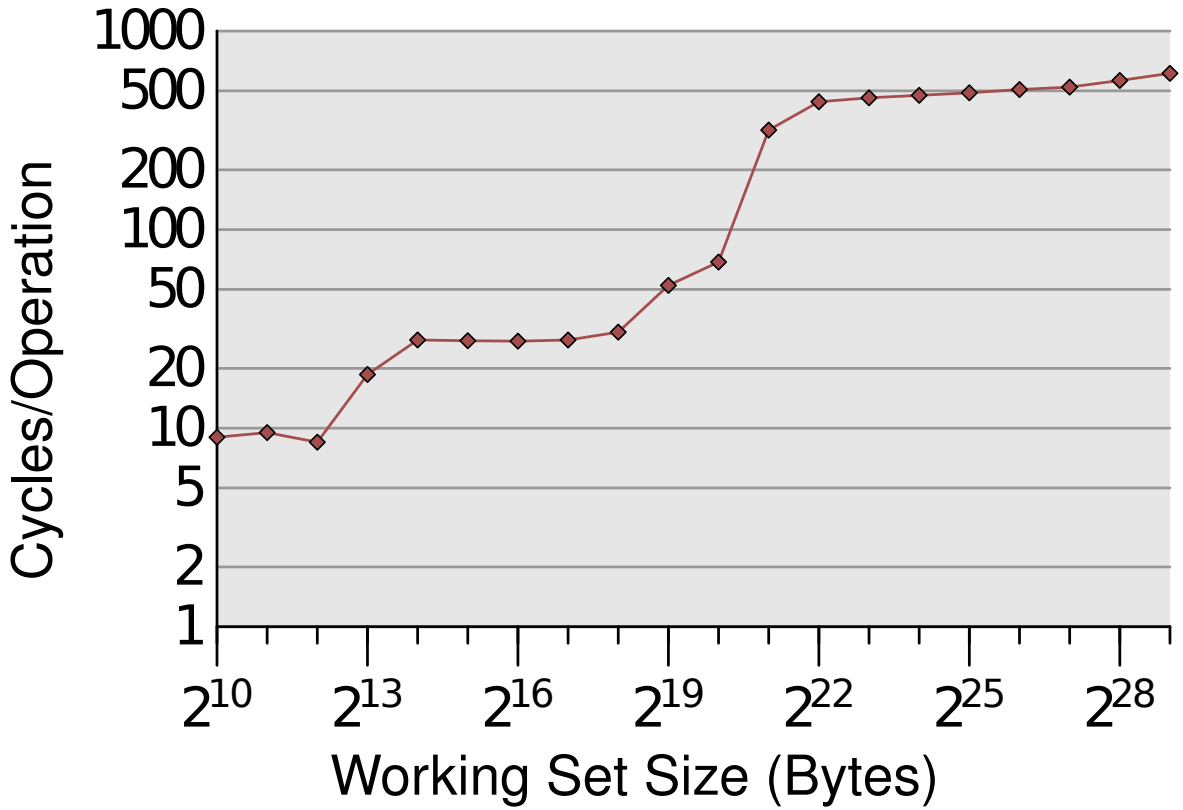
Cache: 32-bit address



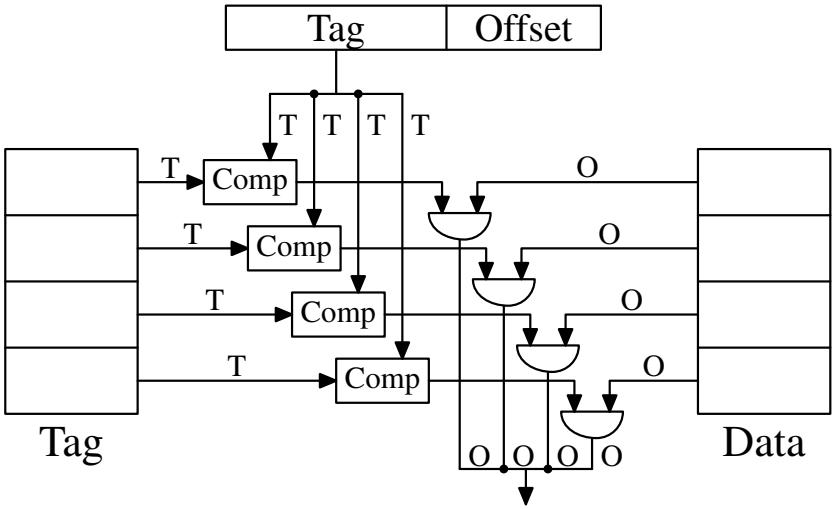
Cache: access times

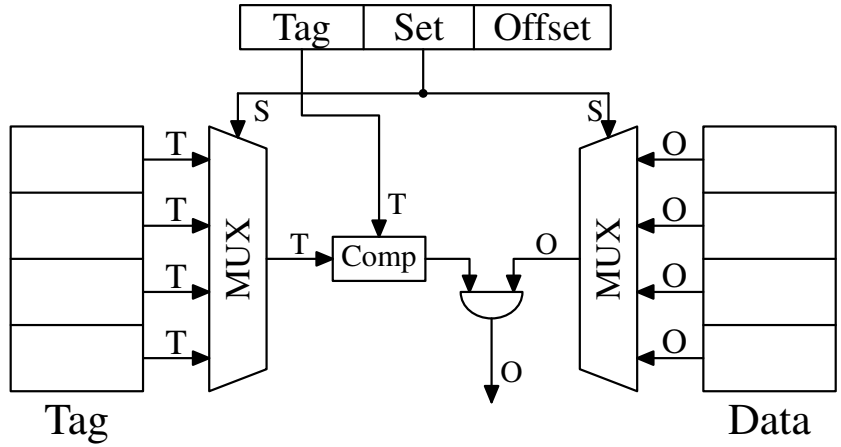


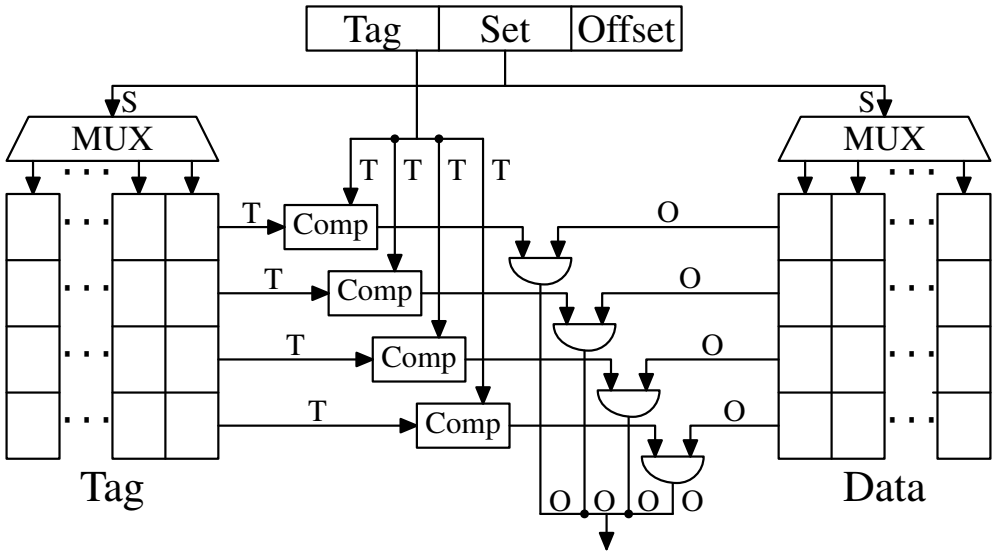
Cache: Access Times for Random Writes



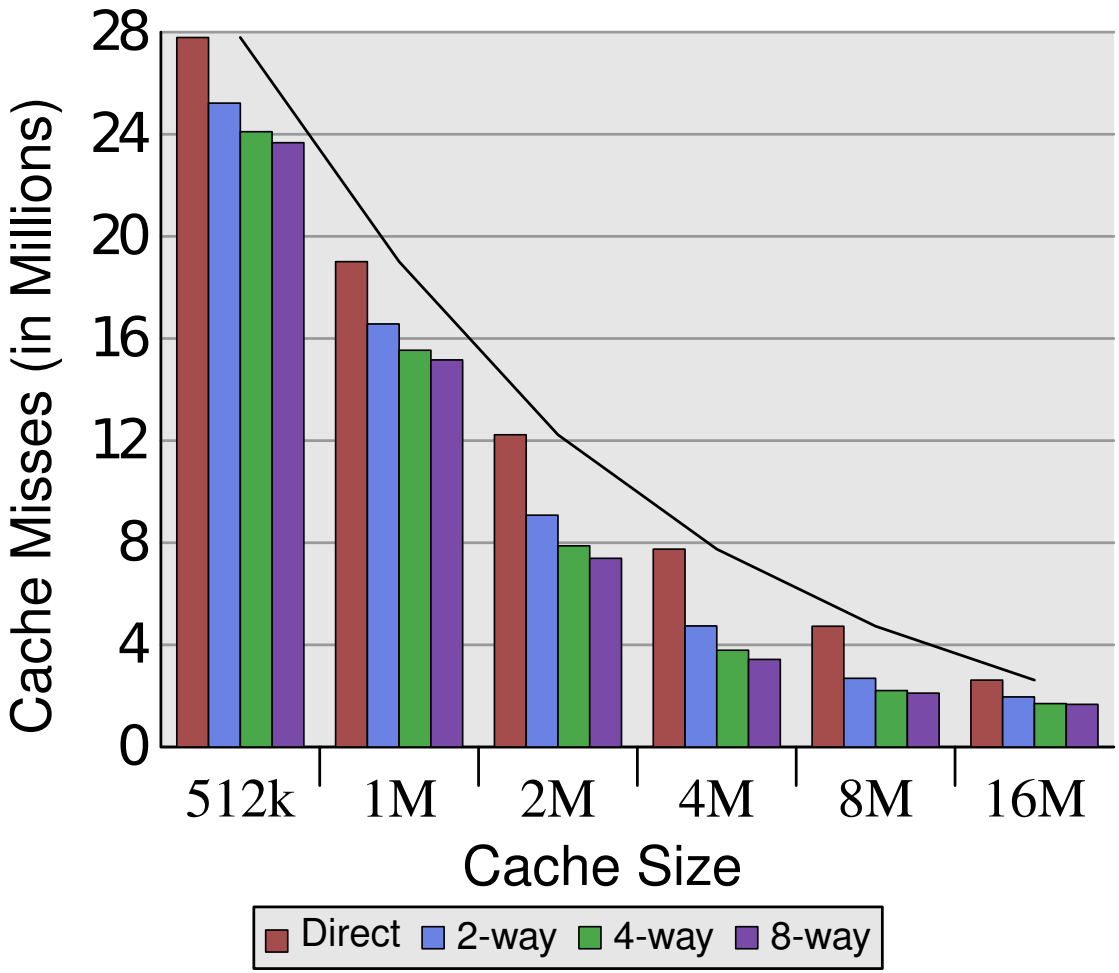
Cache: Fully Associative, Direct-Mapped, Set-Associative





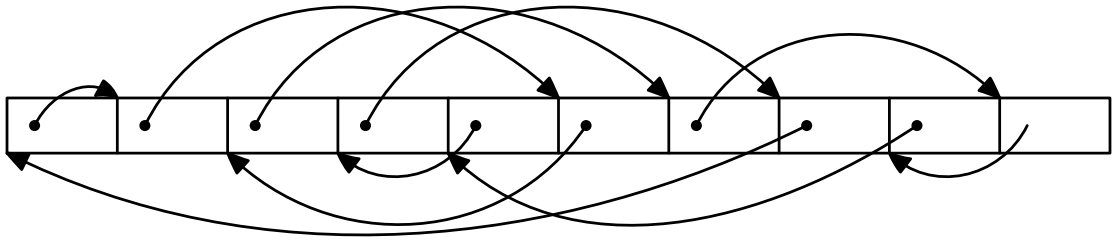


Cache: Size vs Associativity

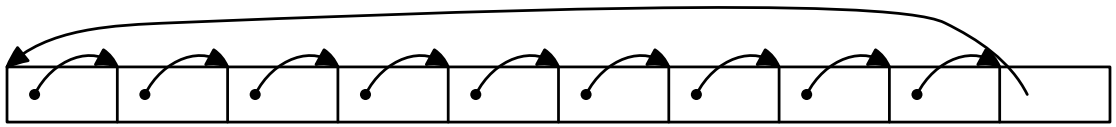


Cache: Test Memory Layouts

```
struct l {
  struct l *n;
  long int pad[NPAD];
};
```

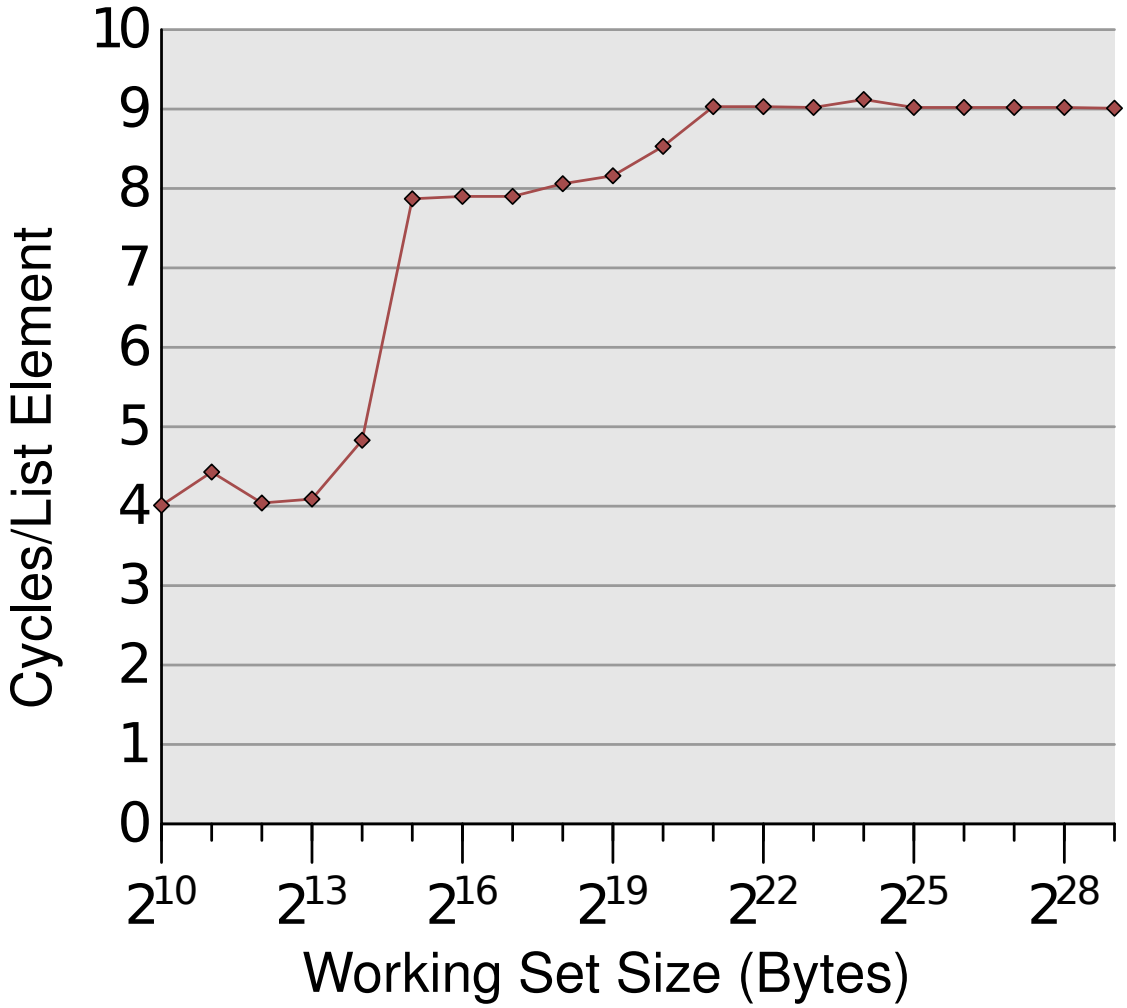


Random

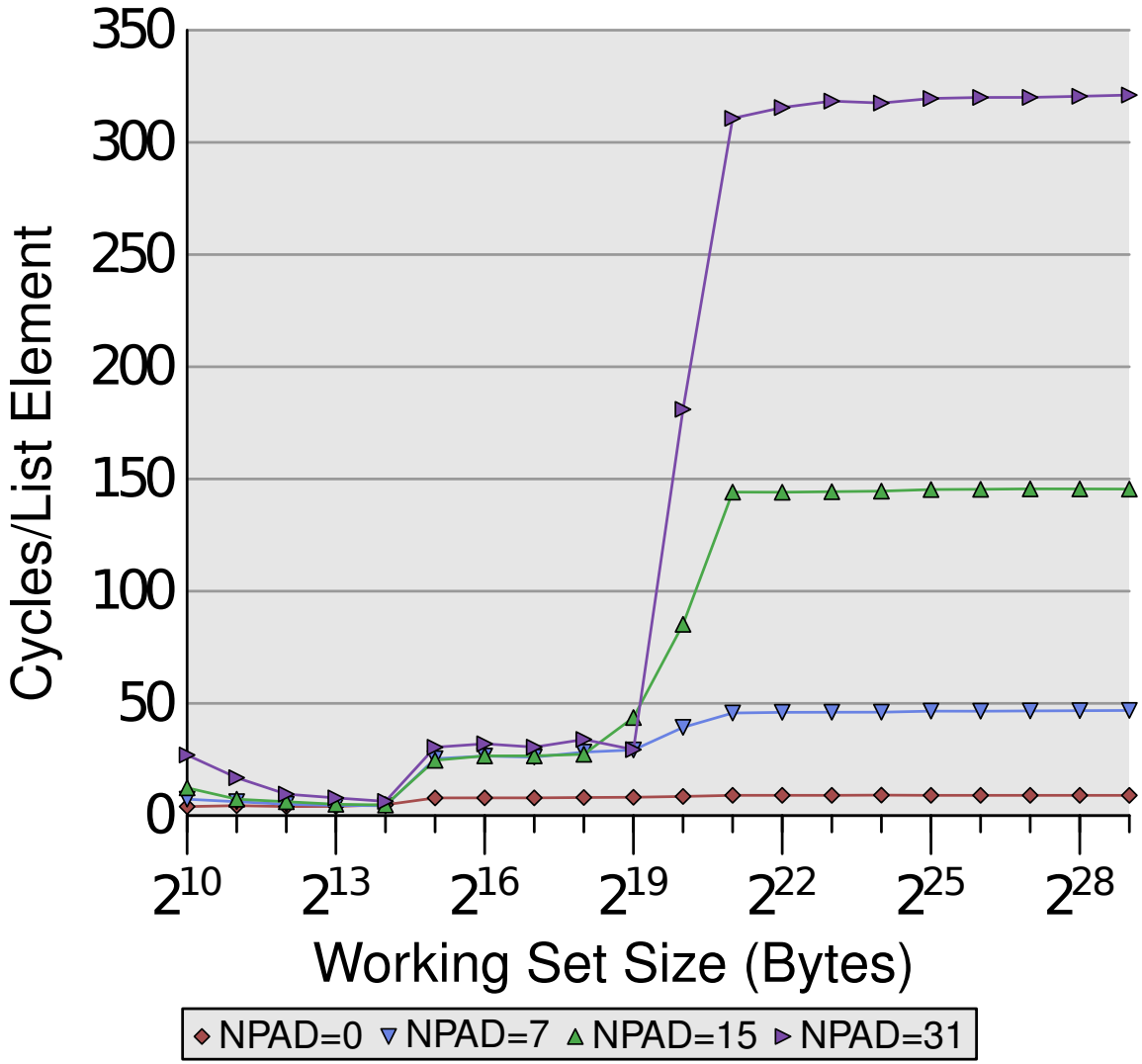


Sequential

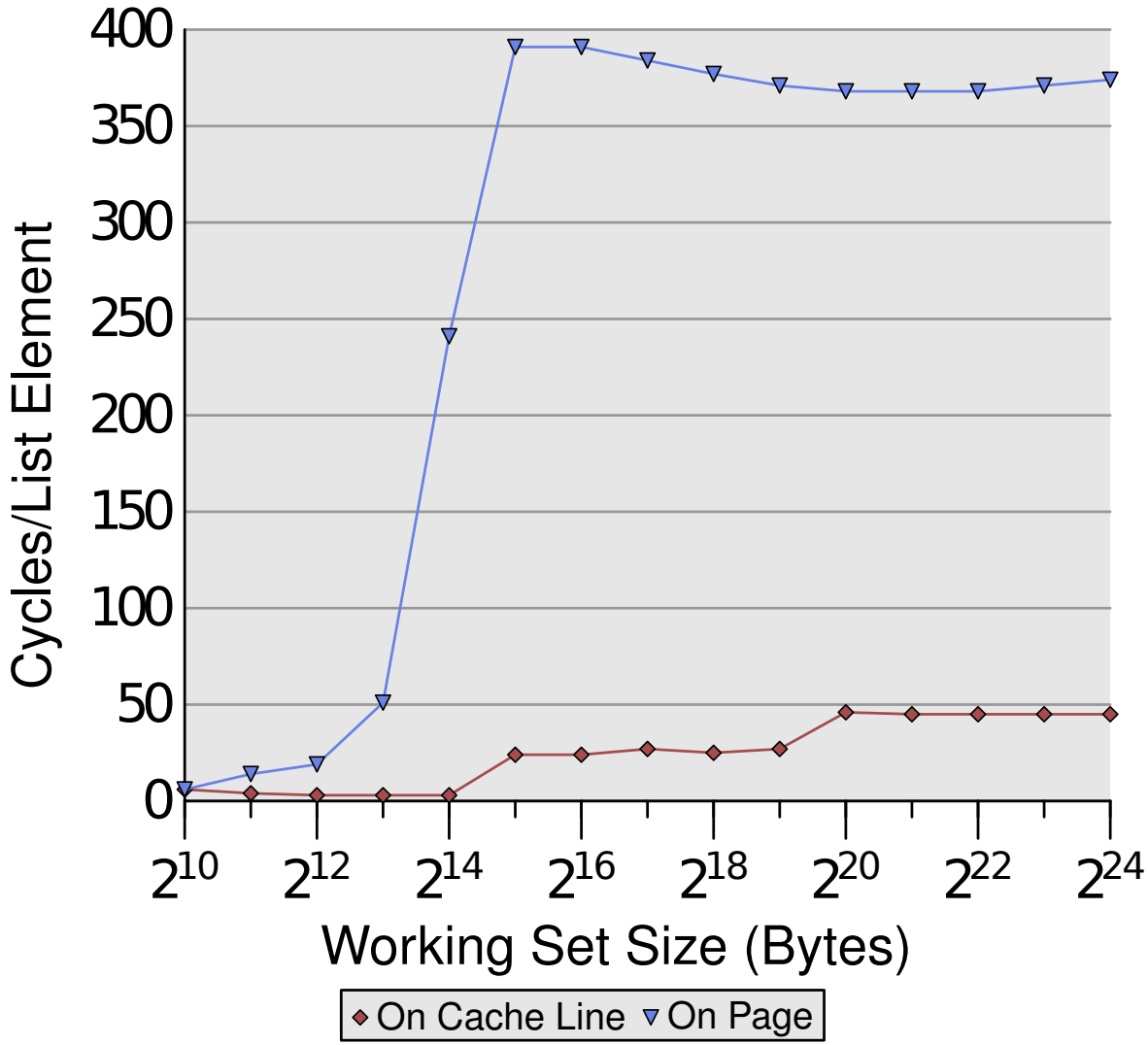
Cache: Sequential Read Access



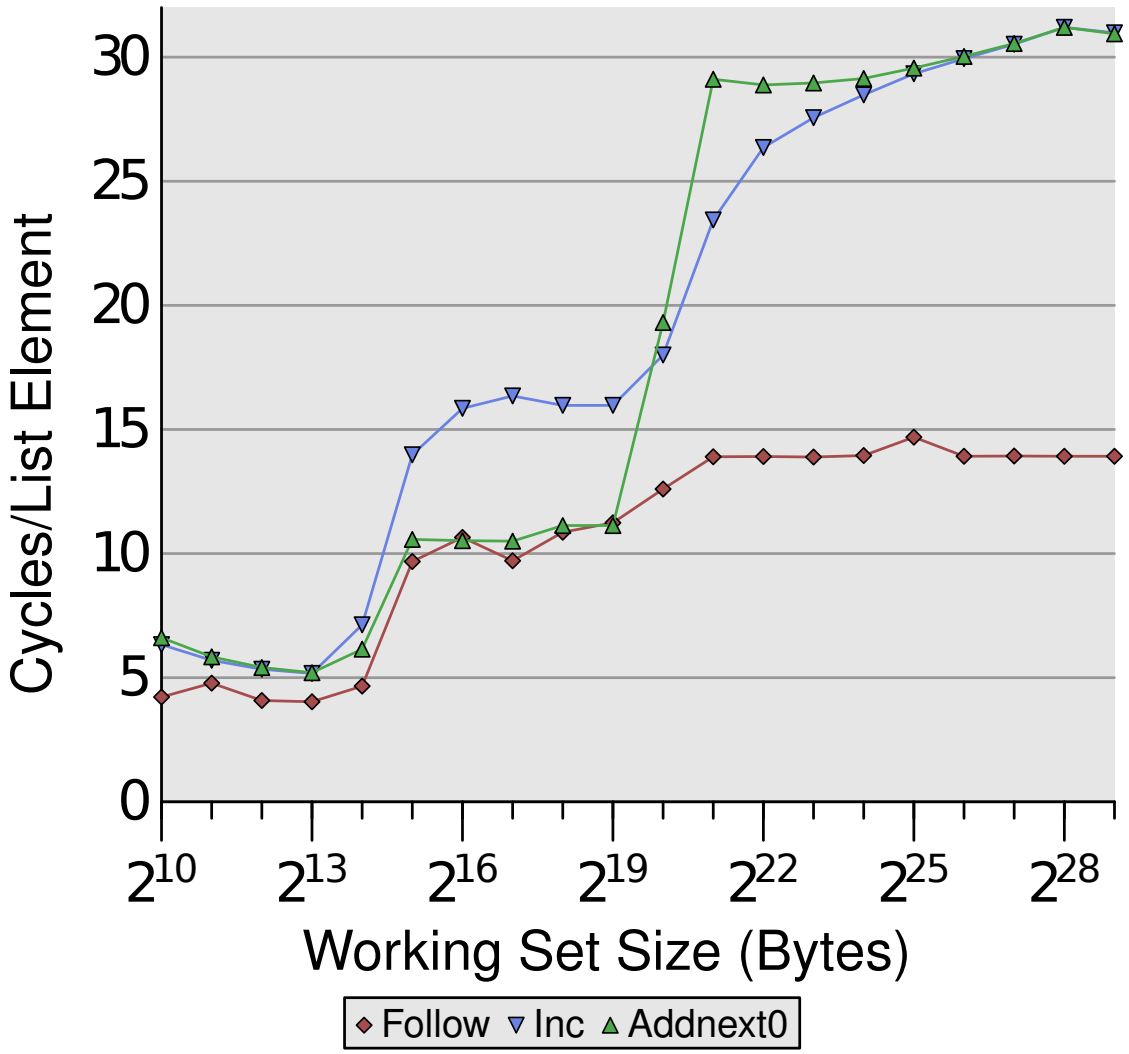
Cache: Sequential Read for Several Sizes



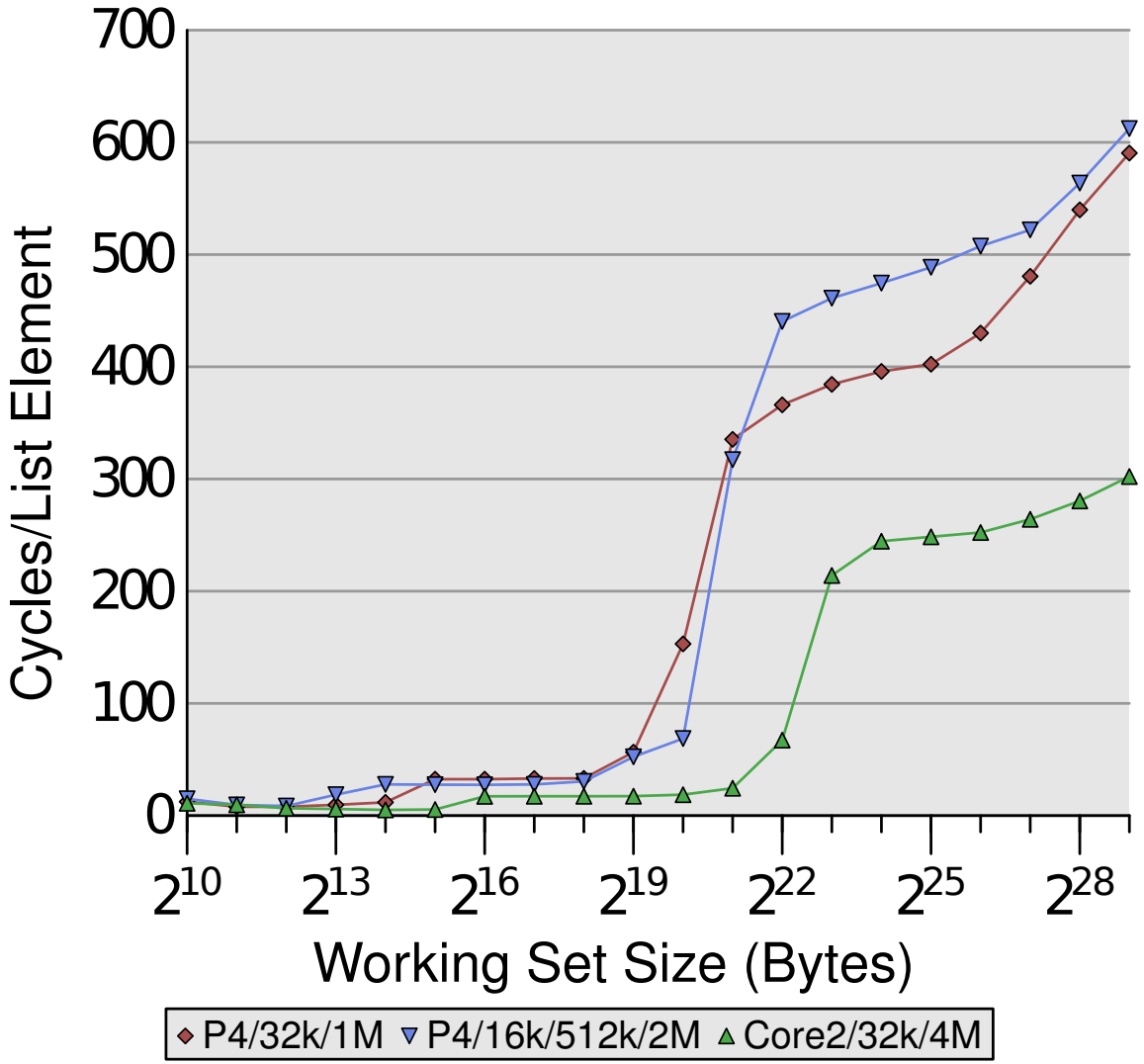
Cache: TLB Influence for Sequential Read



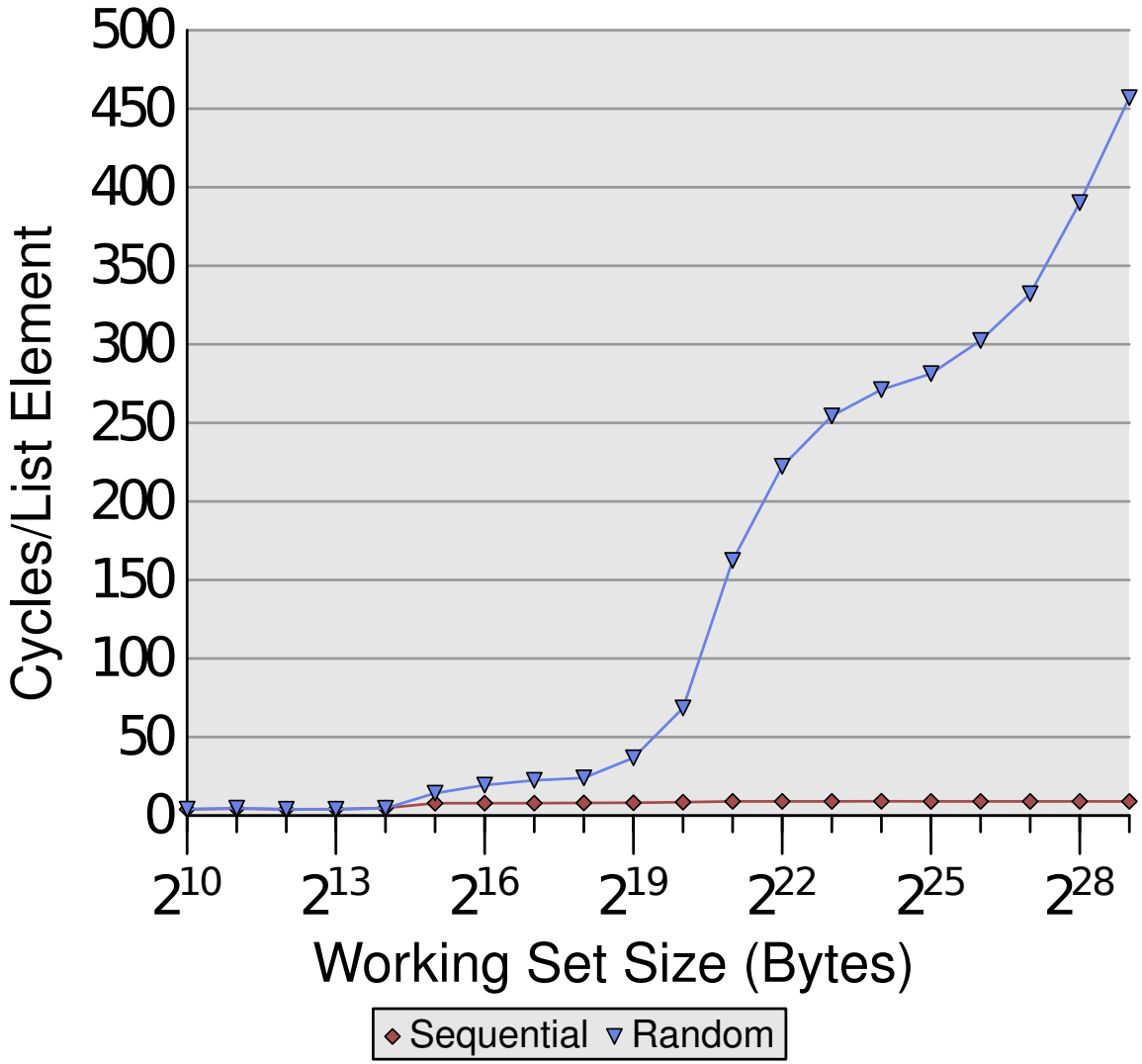
Cache: Sequential Read and Write



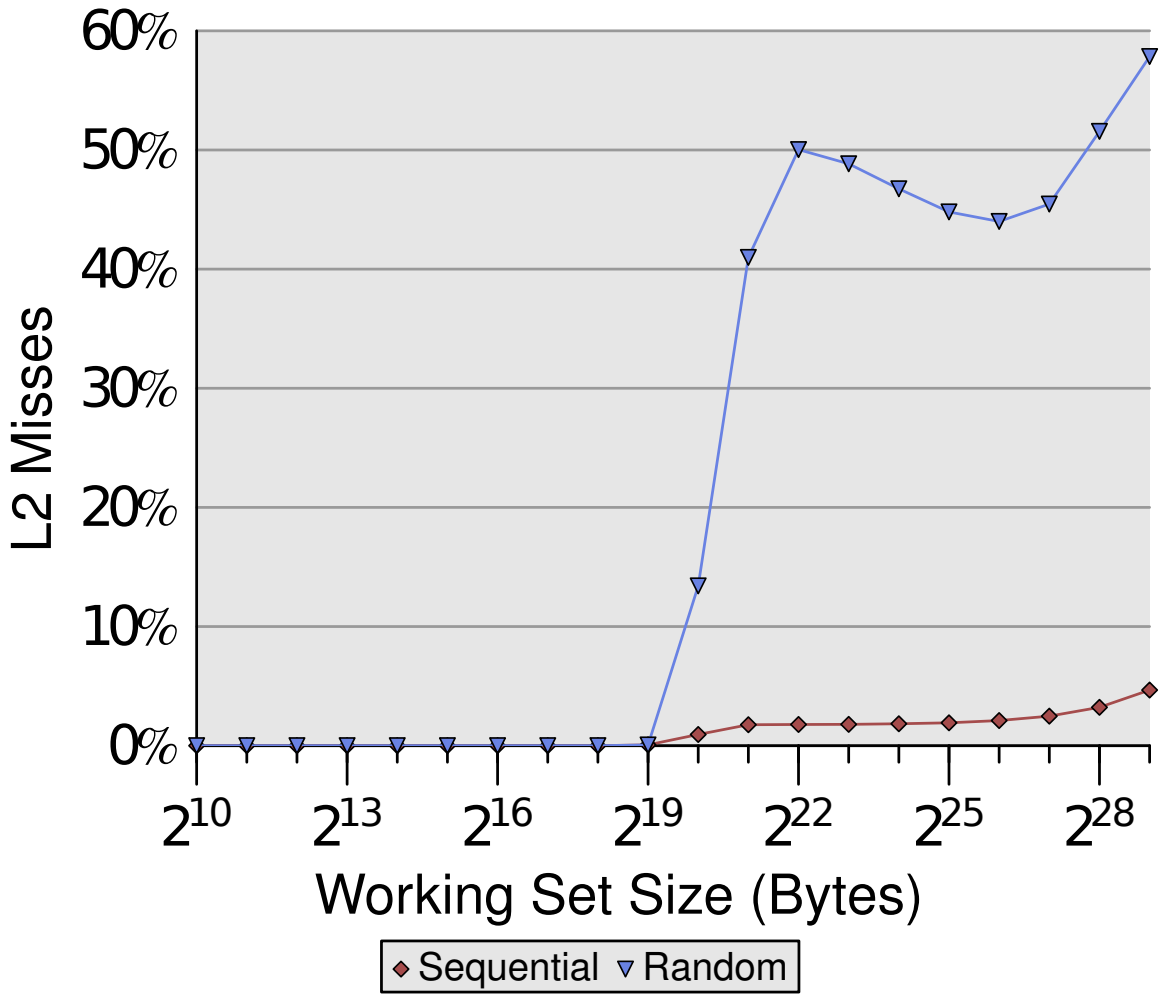
Cache: Advantage of Larger L2/L3 Caches



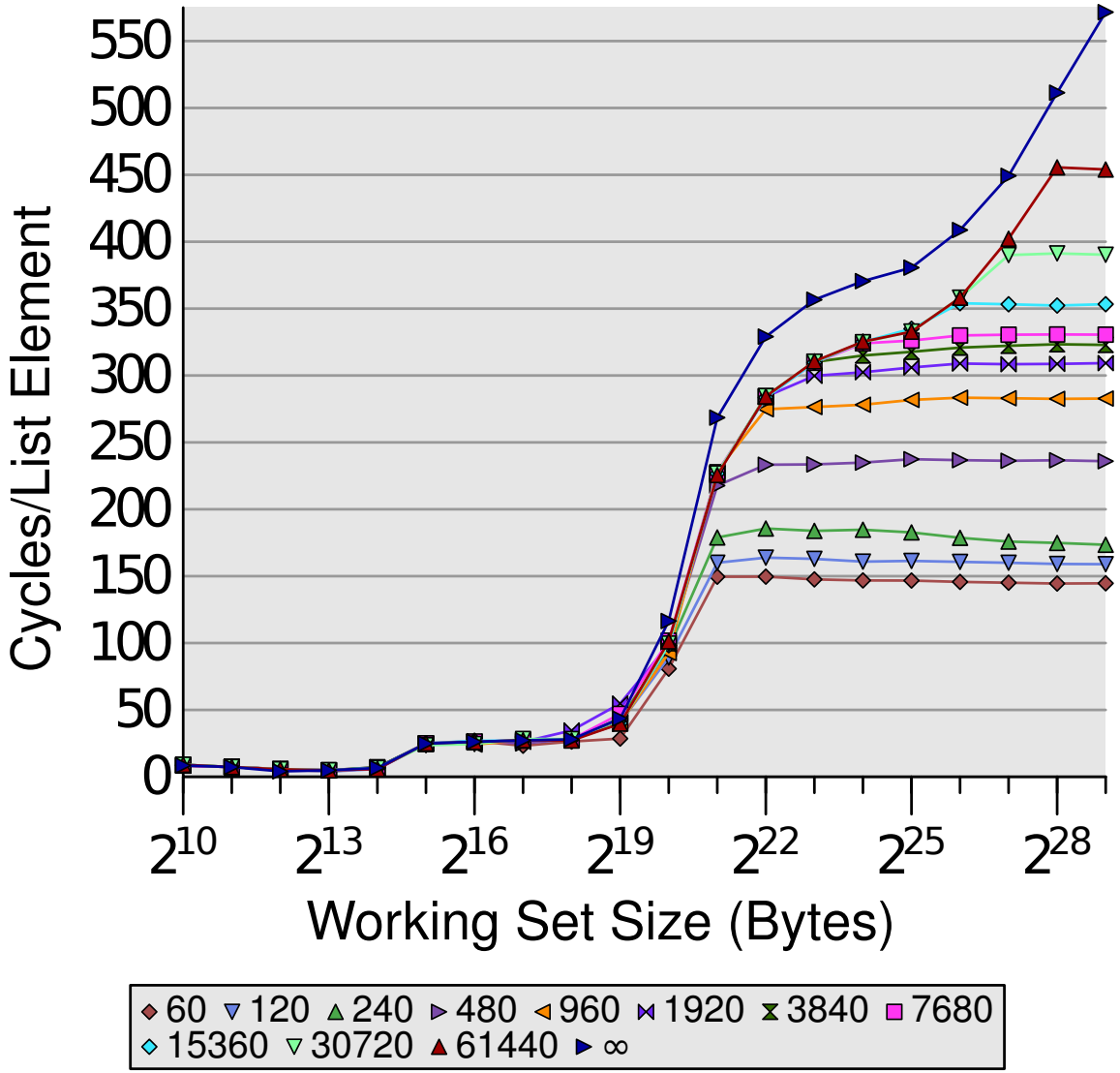
Cache: Sequential vs Random Read



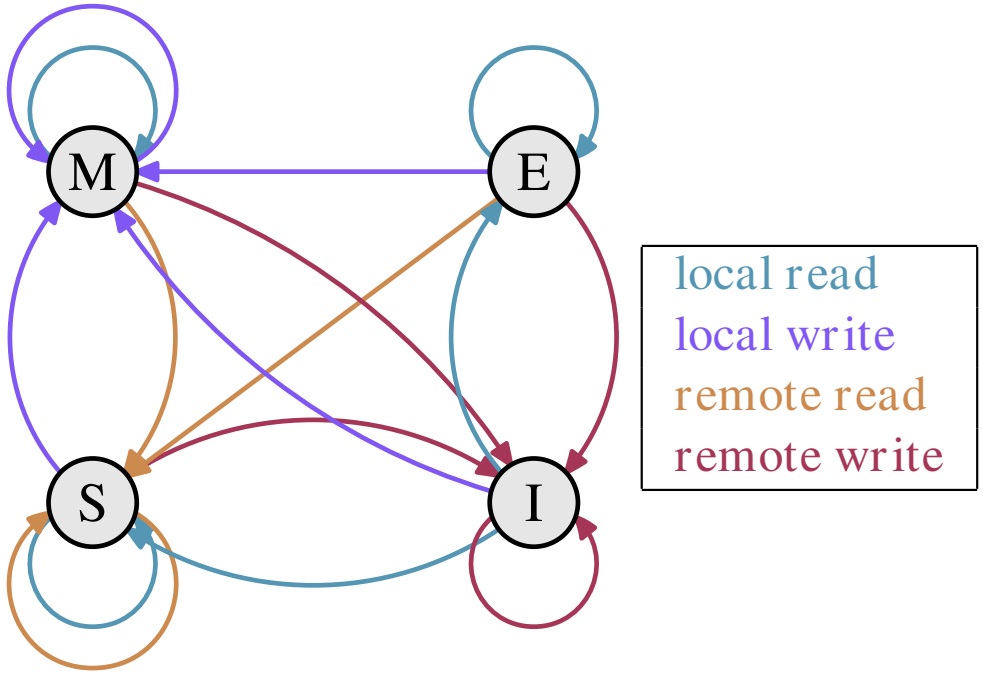
Cache: L2d Miss



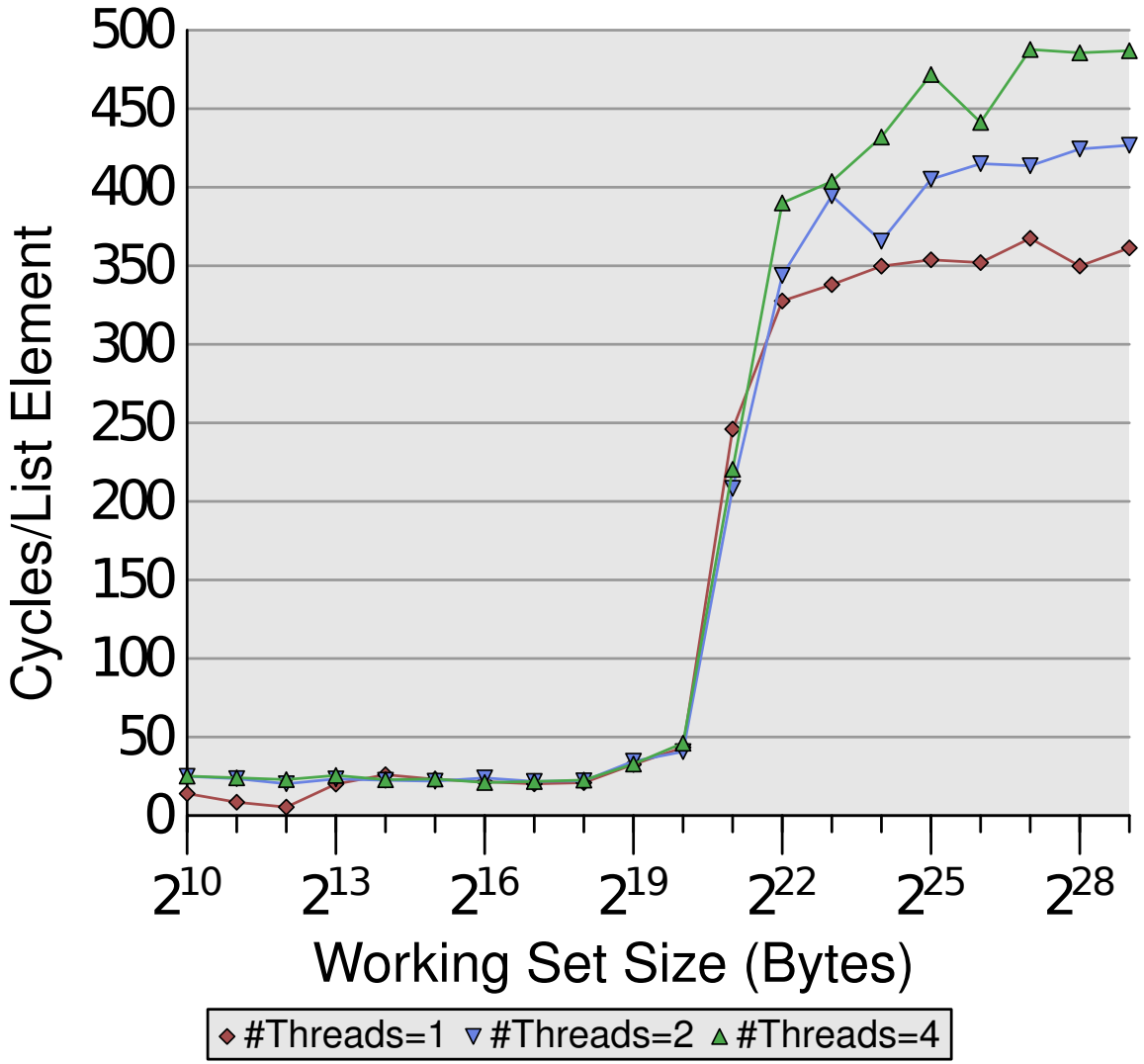
Cache: Page-Wise Randomization



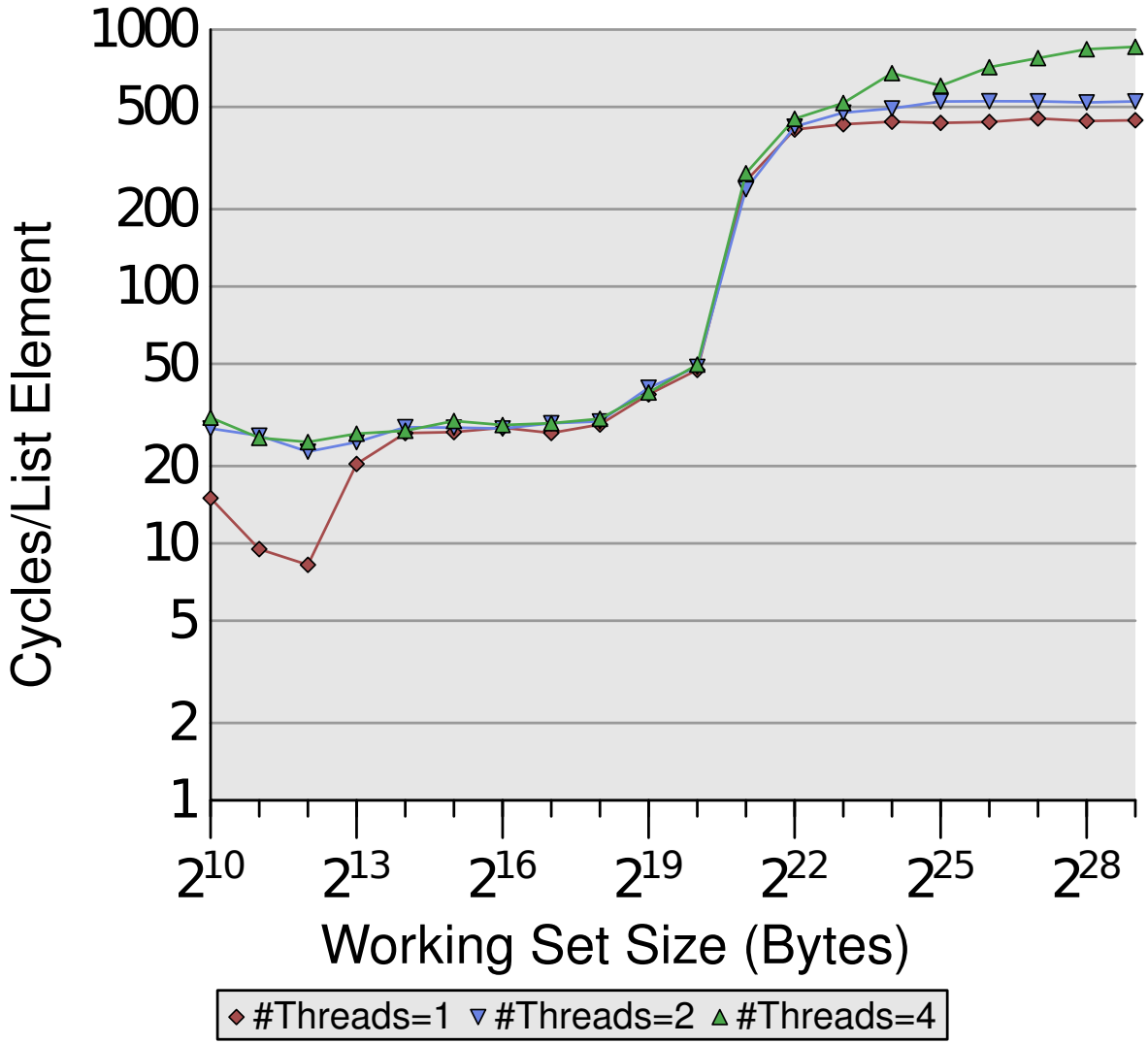
Cache: MESI Protocol Transitions



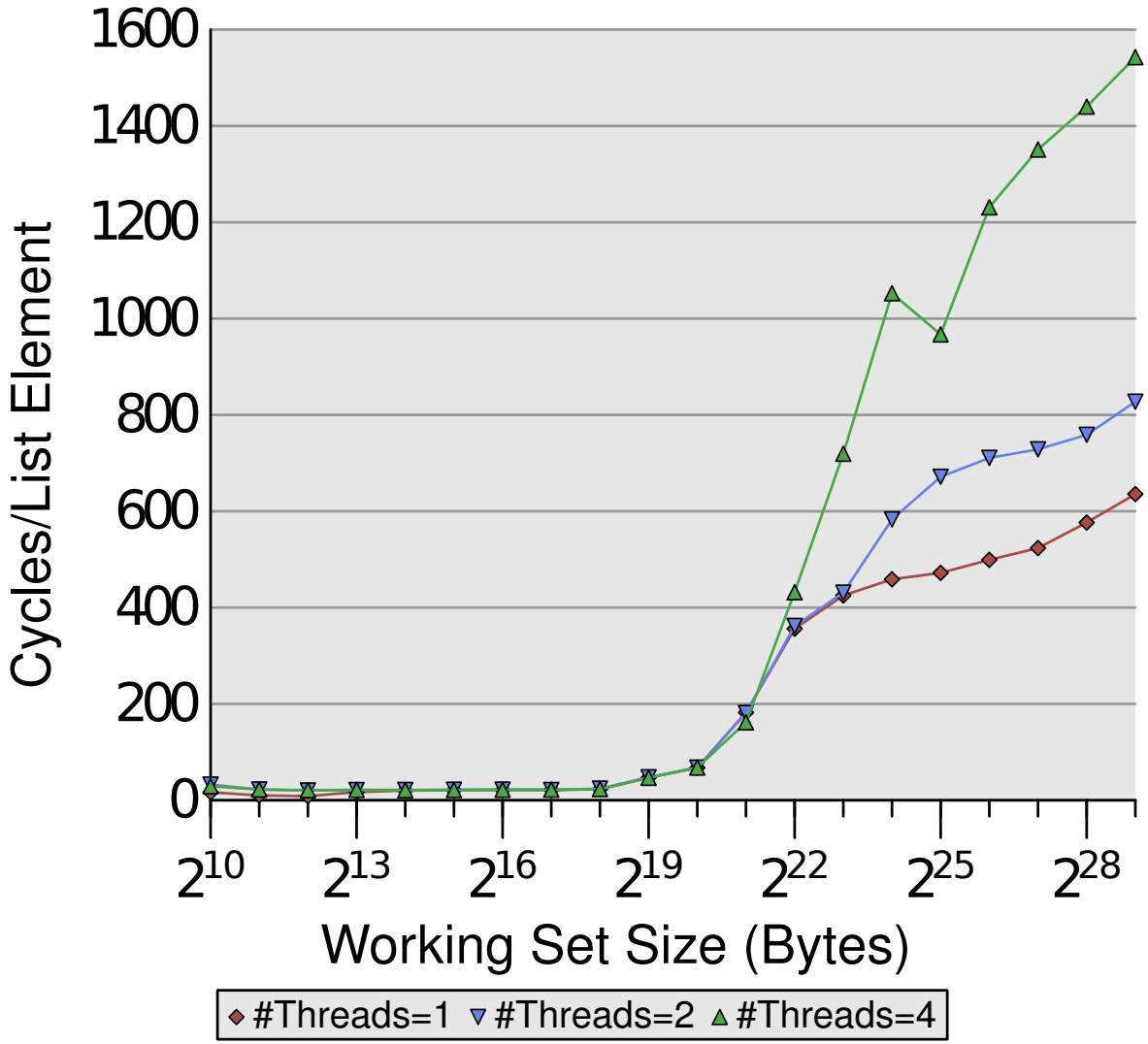
Cache: Sequential Read Access, Multiple Threads



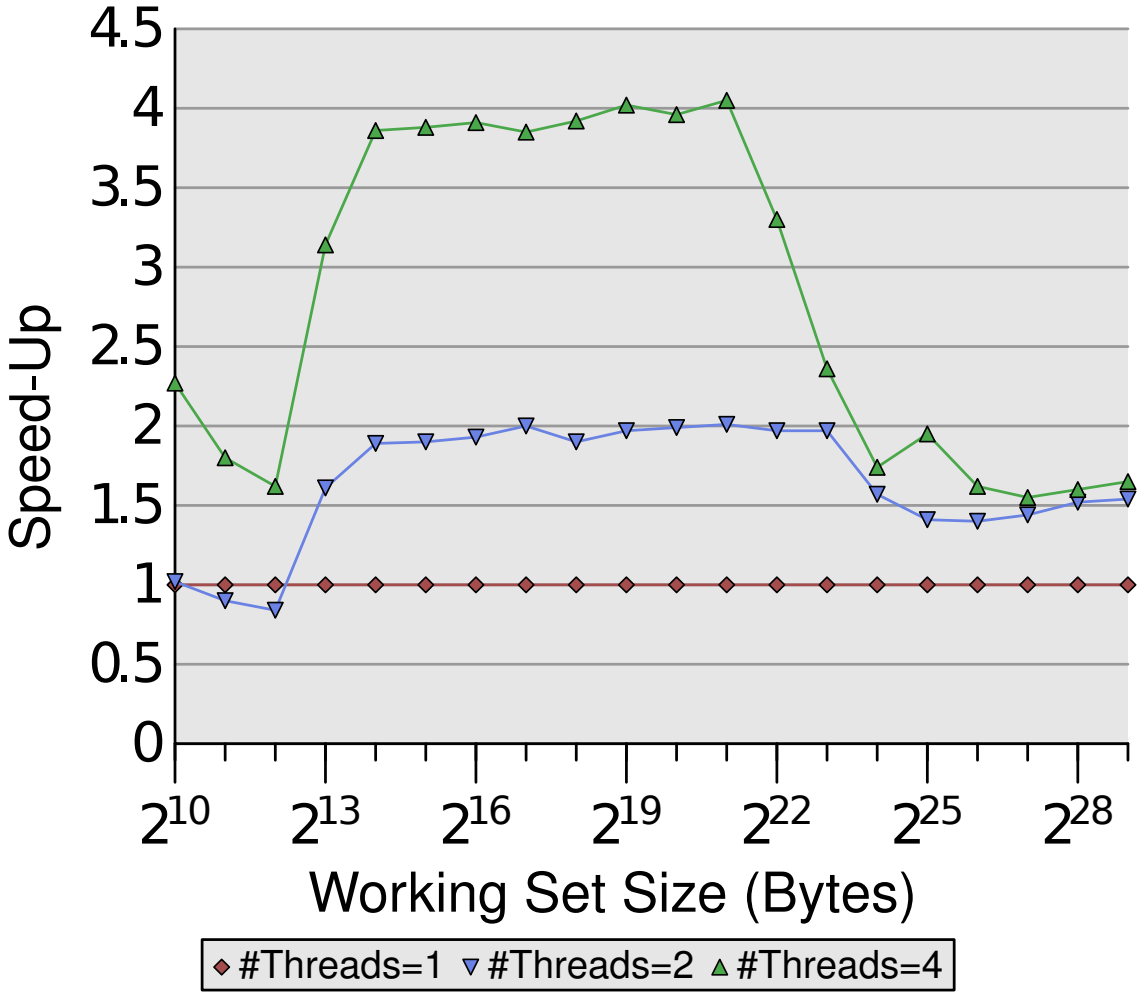
Cache: Sequential Increment, Multiple Threads



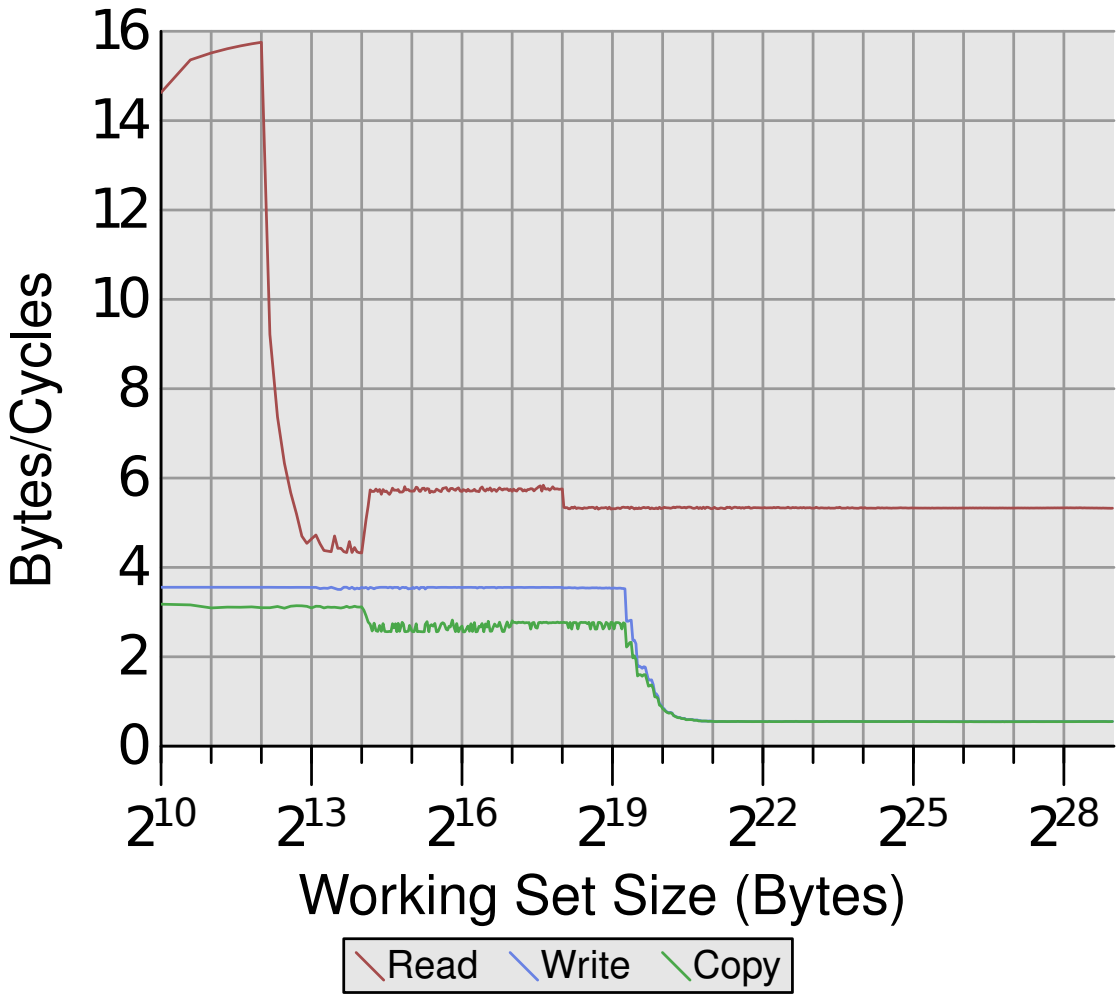
Cache: Random Addnextlast, Multiple Threads



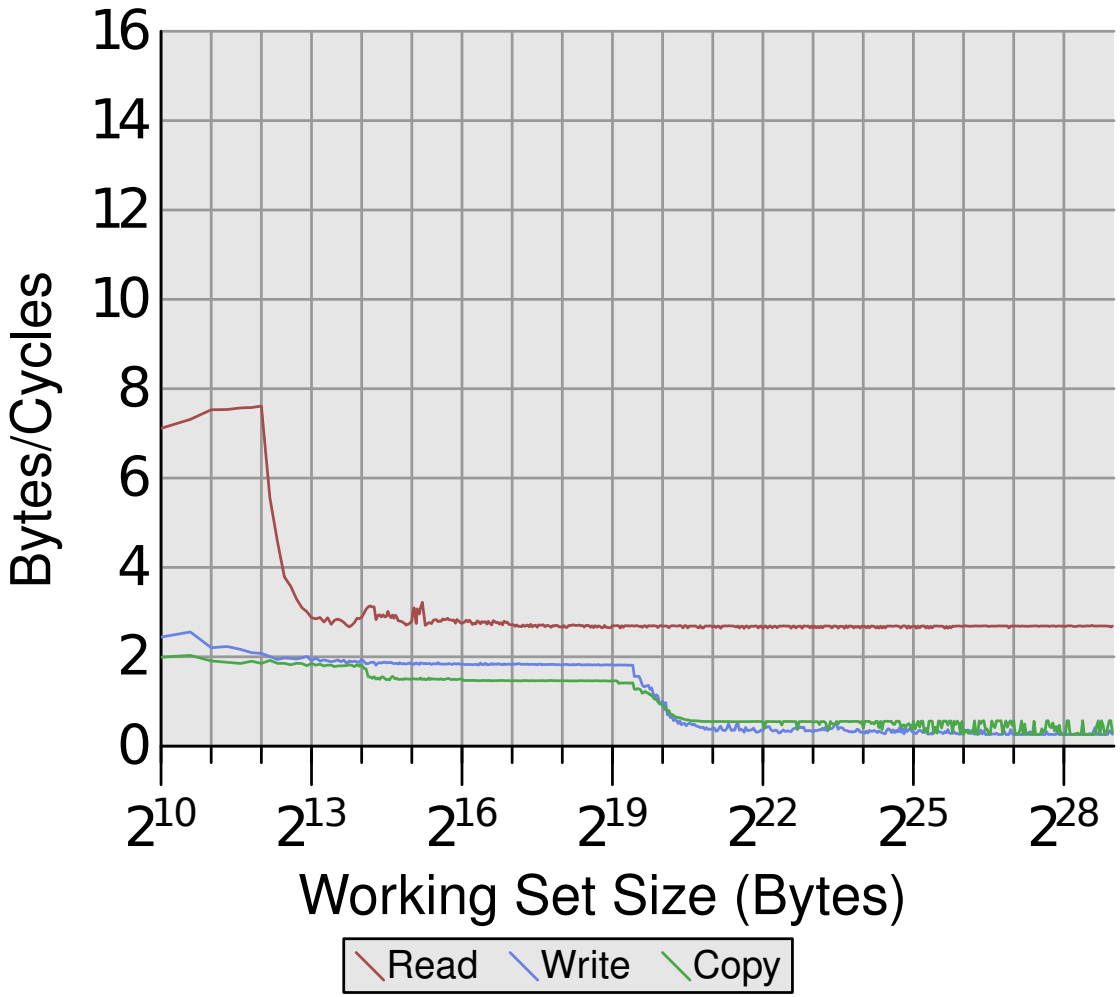
Cache: Speed-Up Through Parallelism



Cache: Pentium 4 Bandwidth

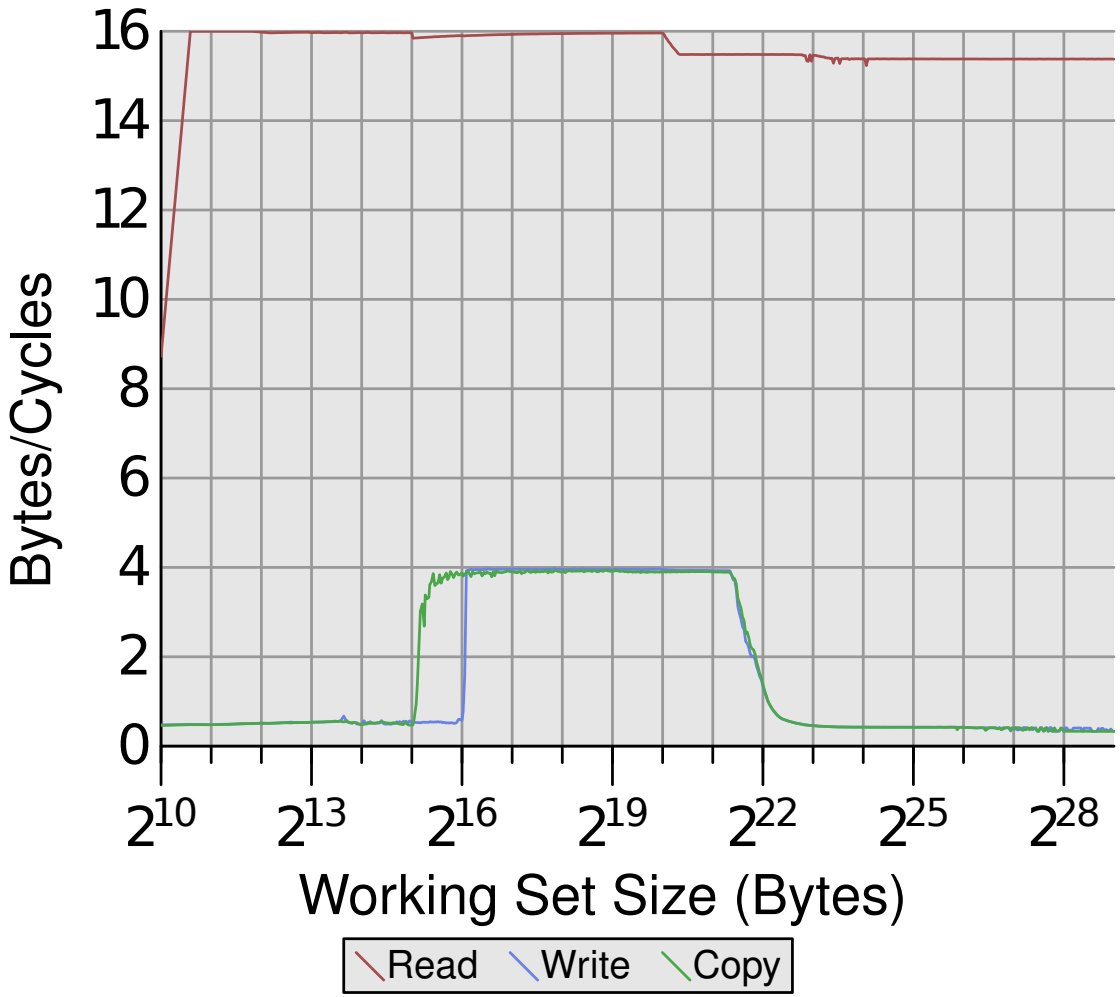


Cache: P4 Bandwidth with 2 Hyper-Threads

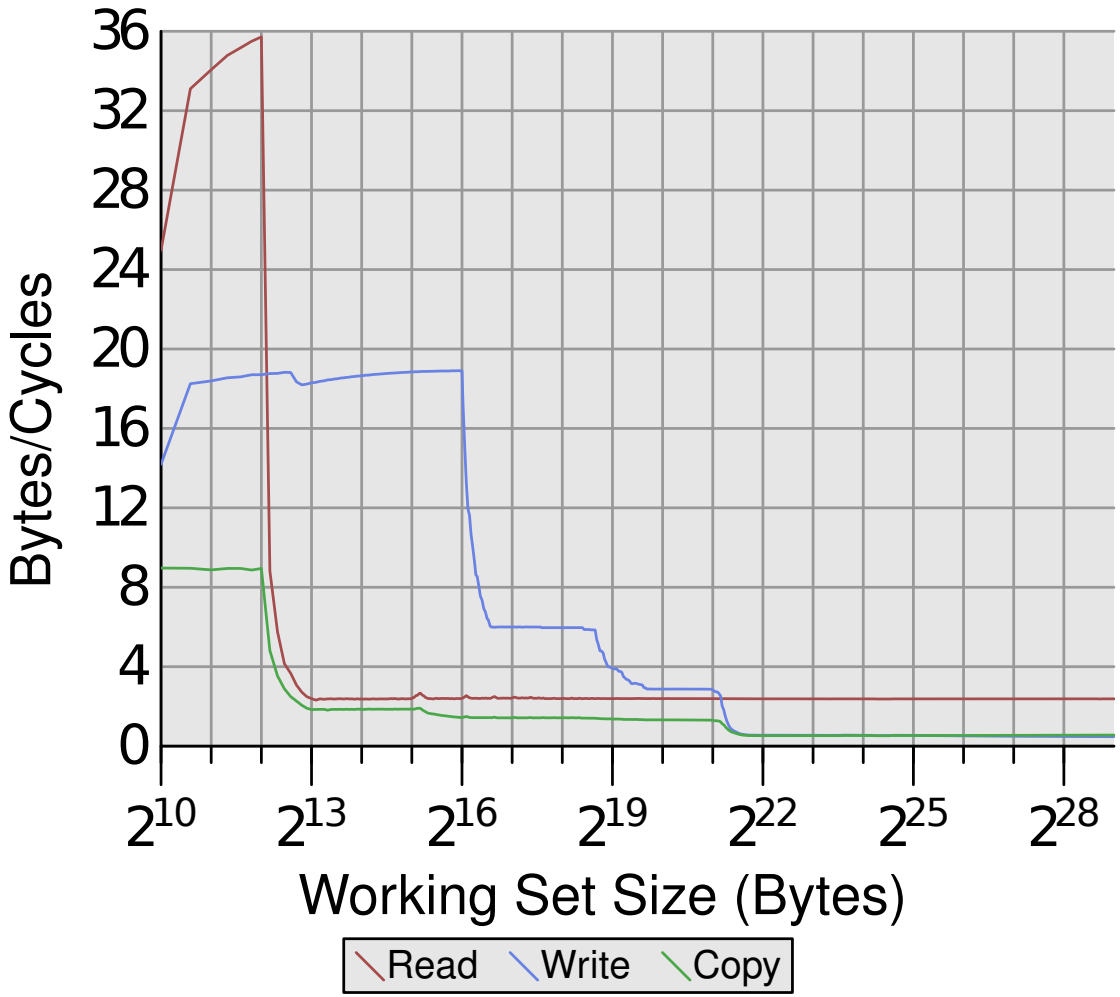


Cache: Core 2 Bandwidth

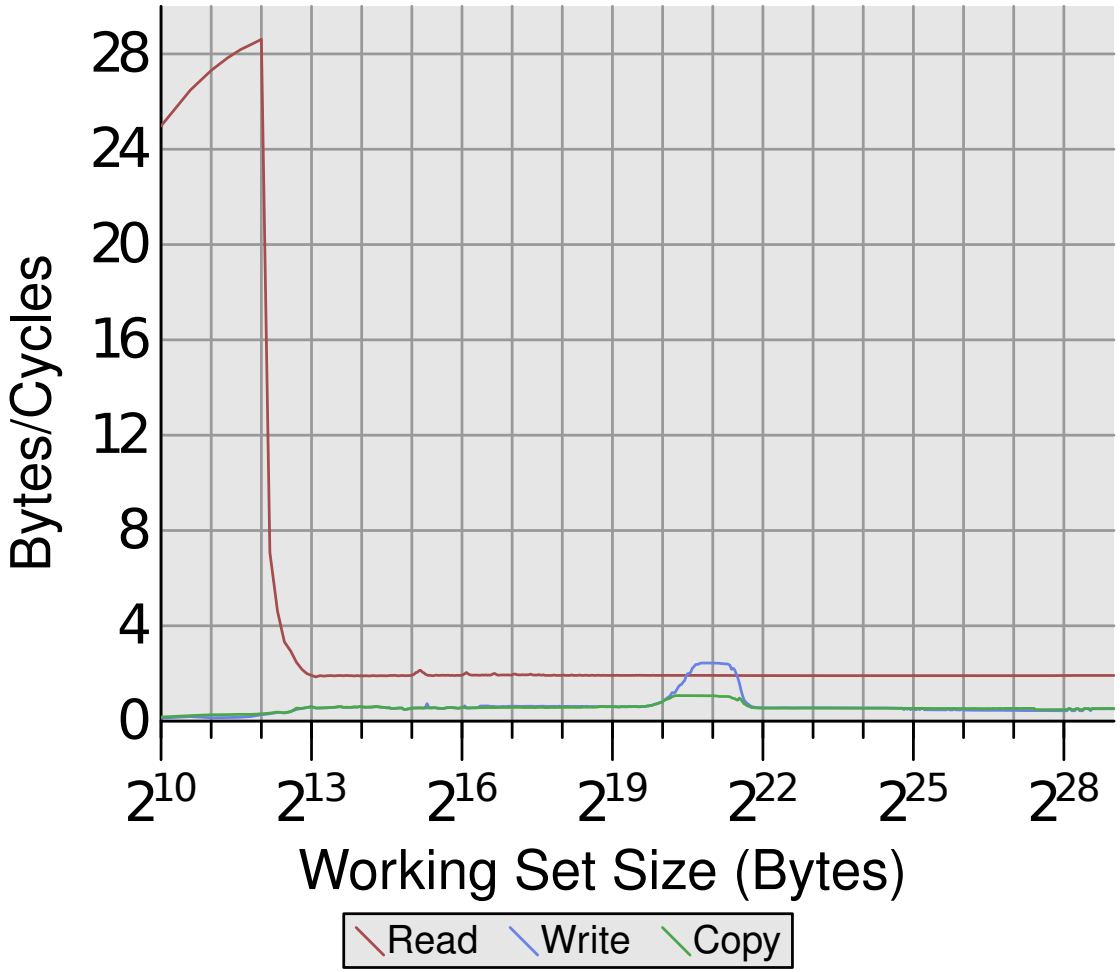
Cache: Core 2 Bandwidth with 2 Threads



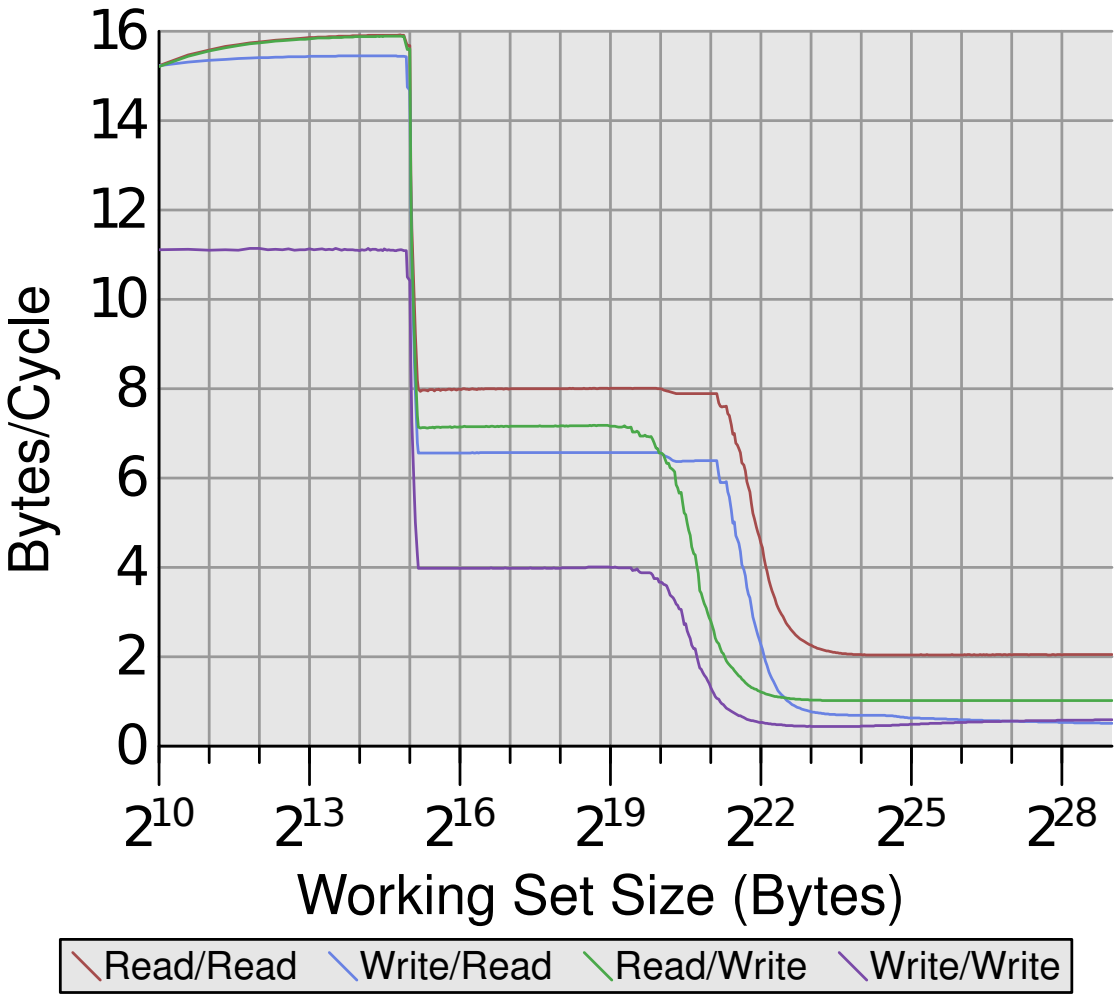
Cache: AMD Family 10h Opteron Bandwidth



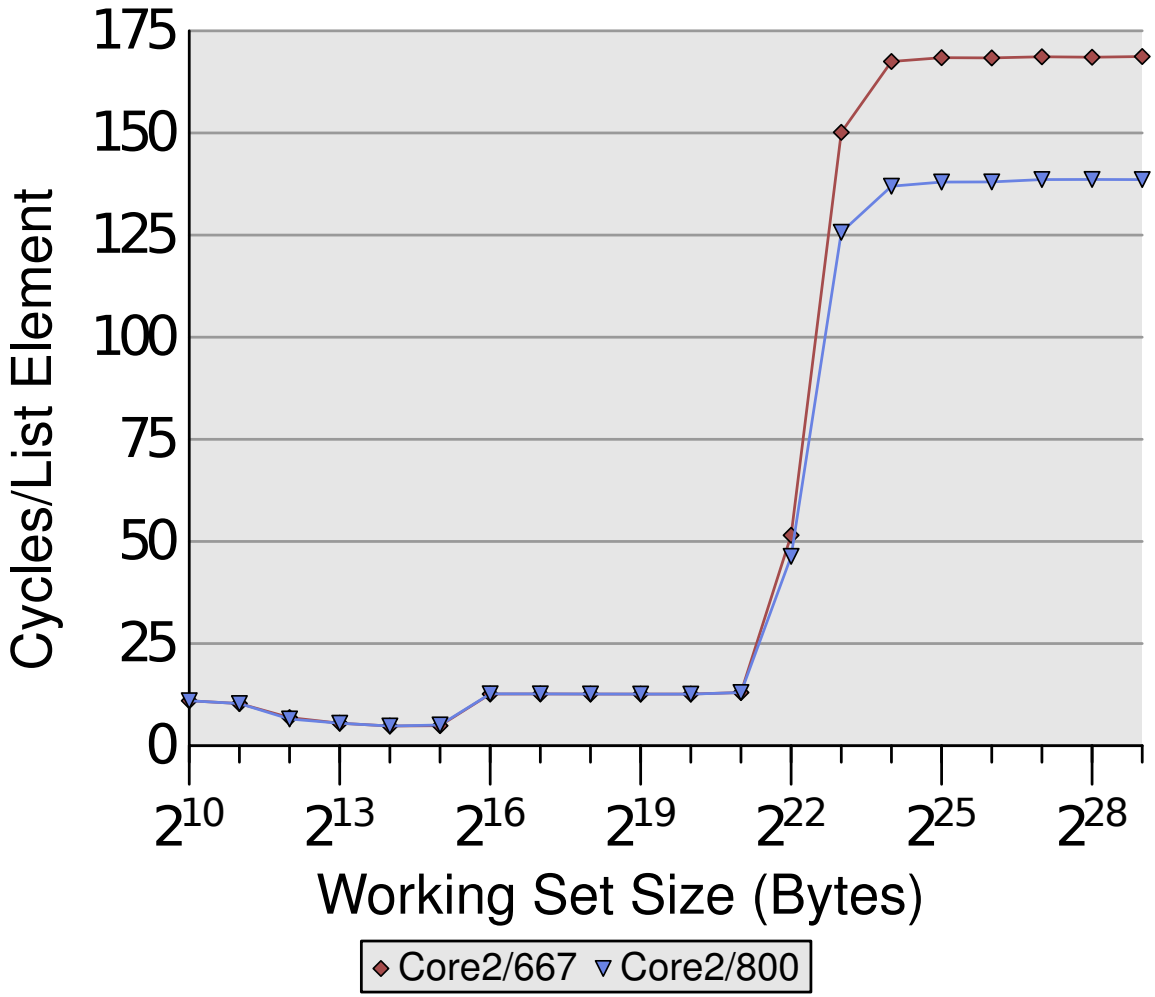
Cache: AMD Fam 10h Bandwidth with 2 Threads



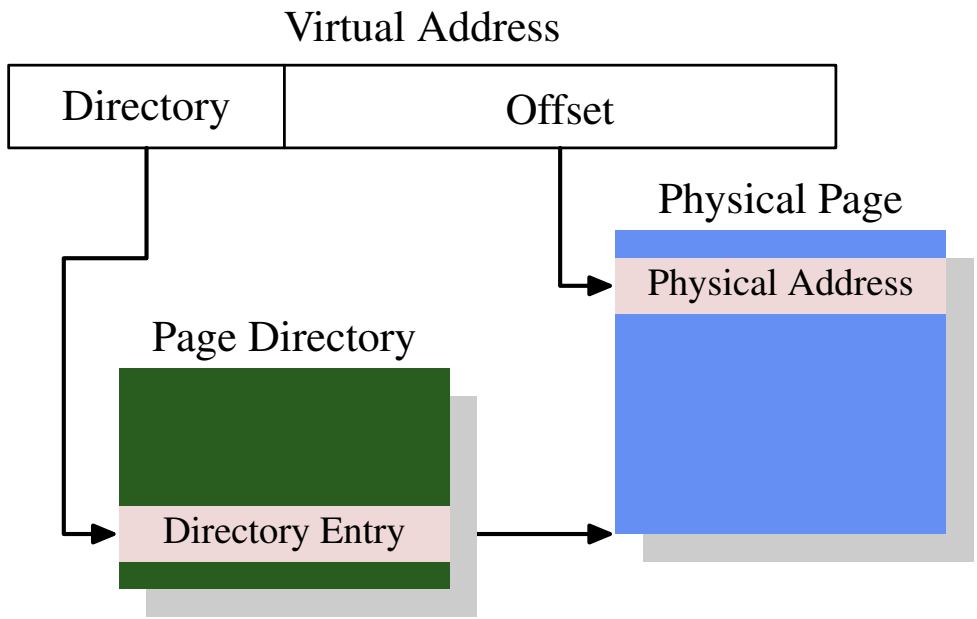
Cache: Bandwidth with two Processes



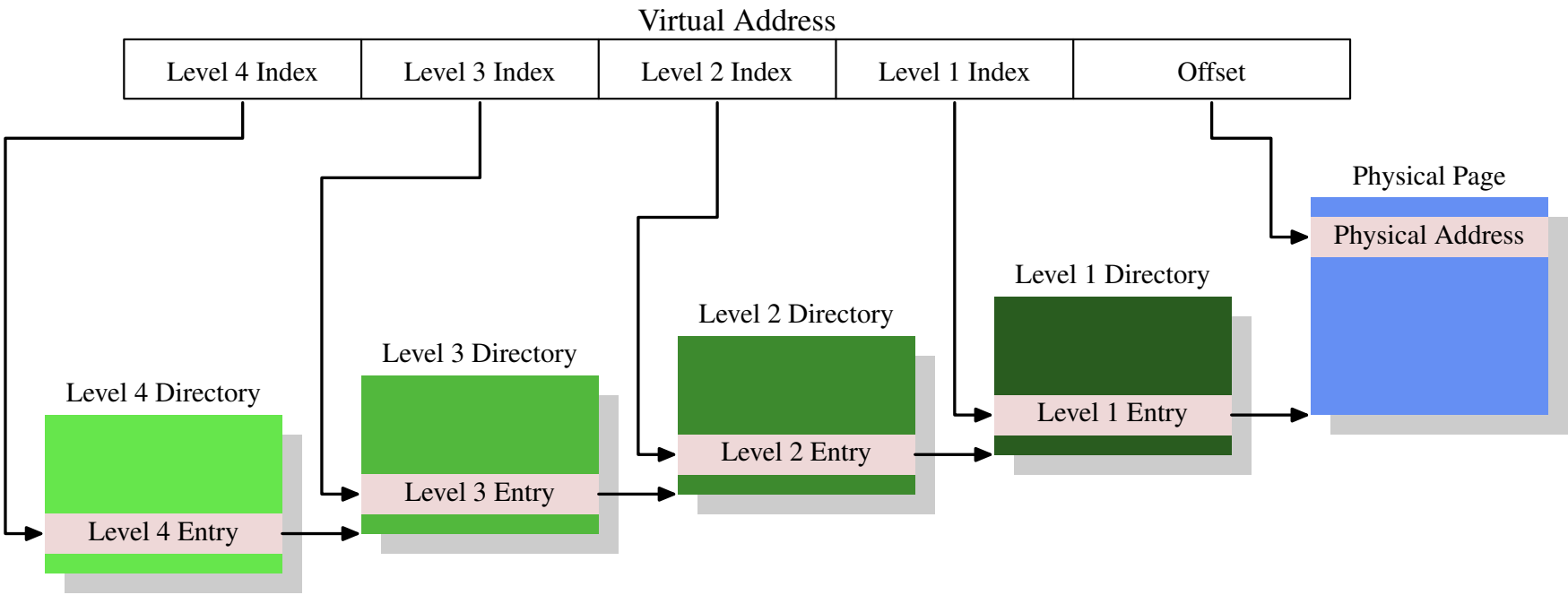
Cache: Influence of FSB Speed



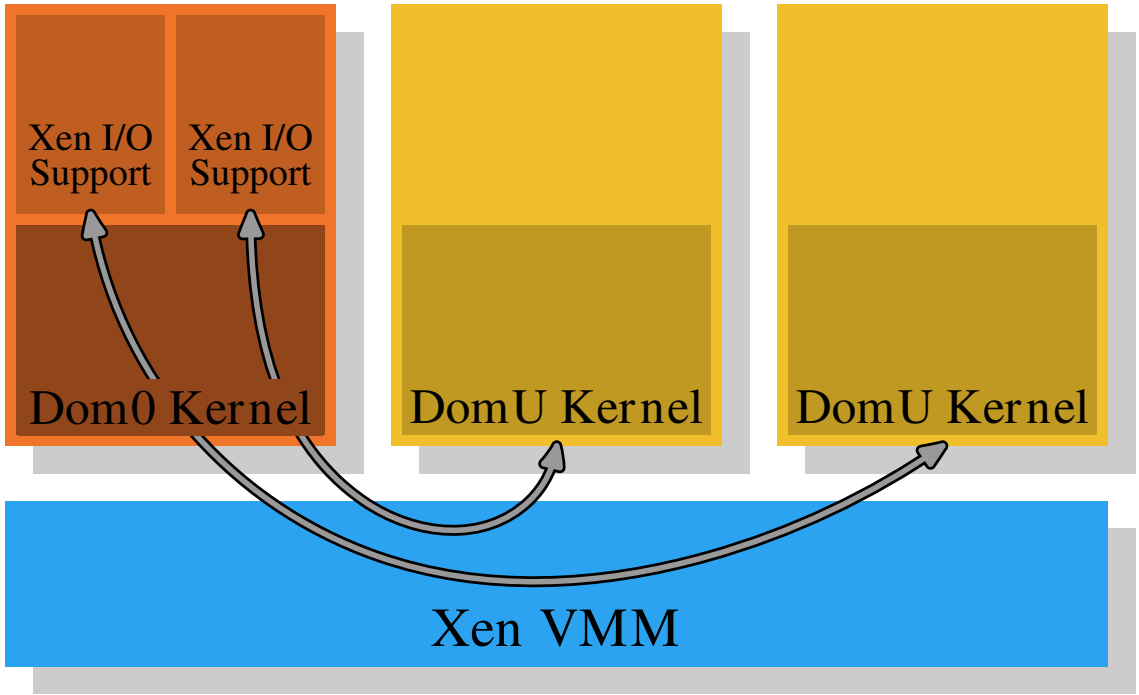
VM: 1-Level Address Translation



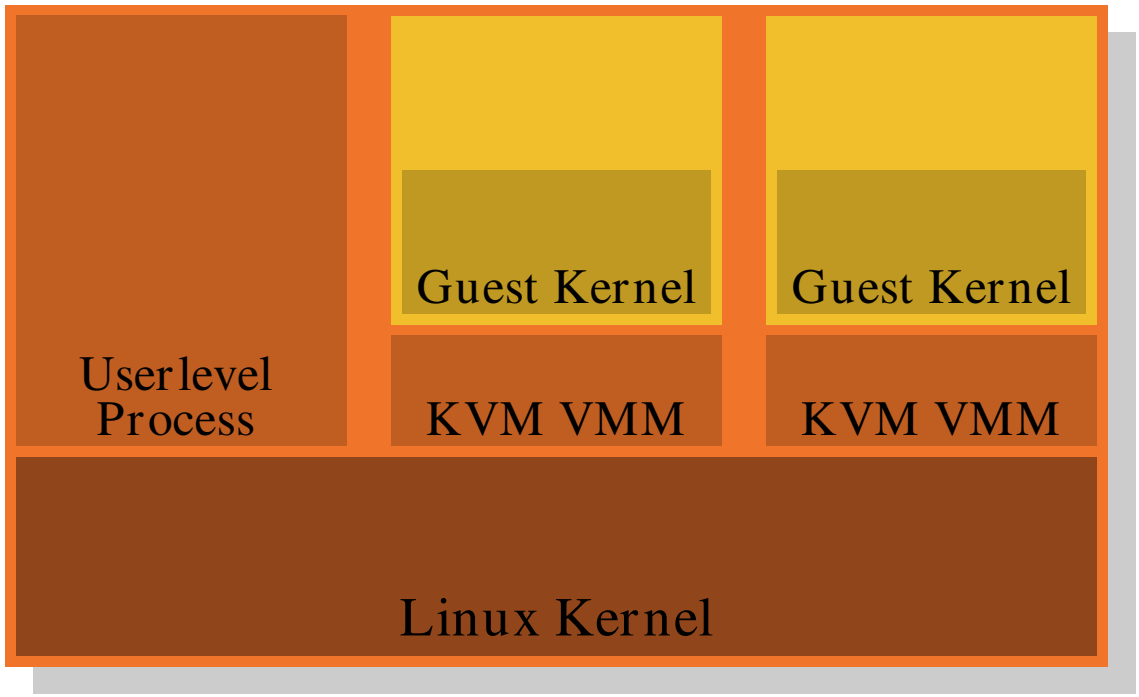
VM: 4-Level Address Translation



VM: Xen Virtualization Model

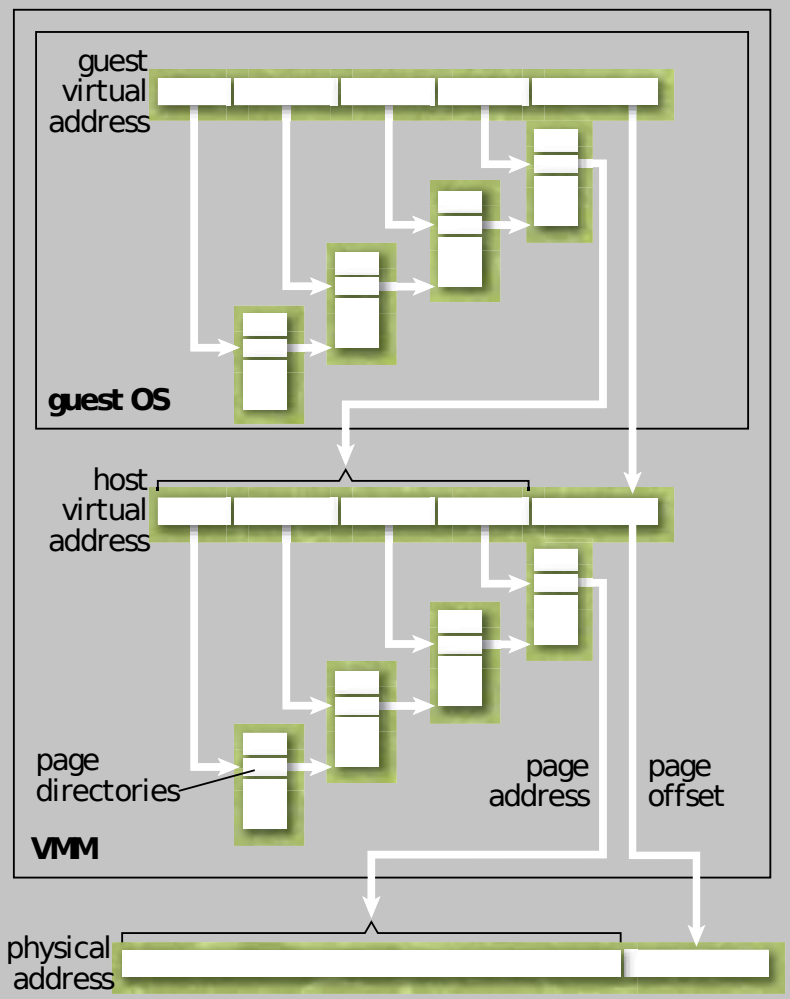


VM: KVM Virtualization Model

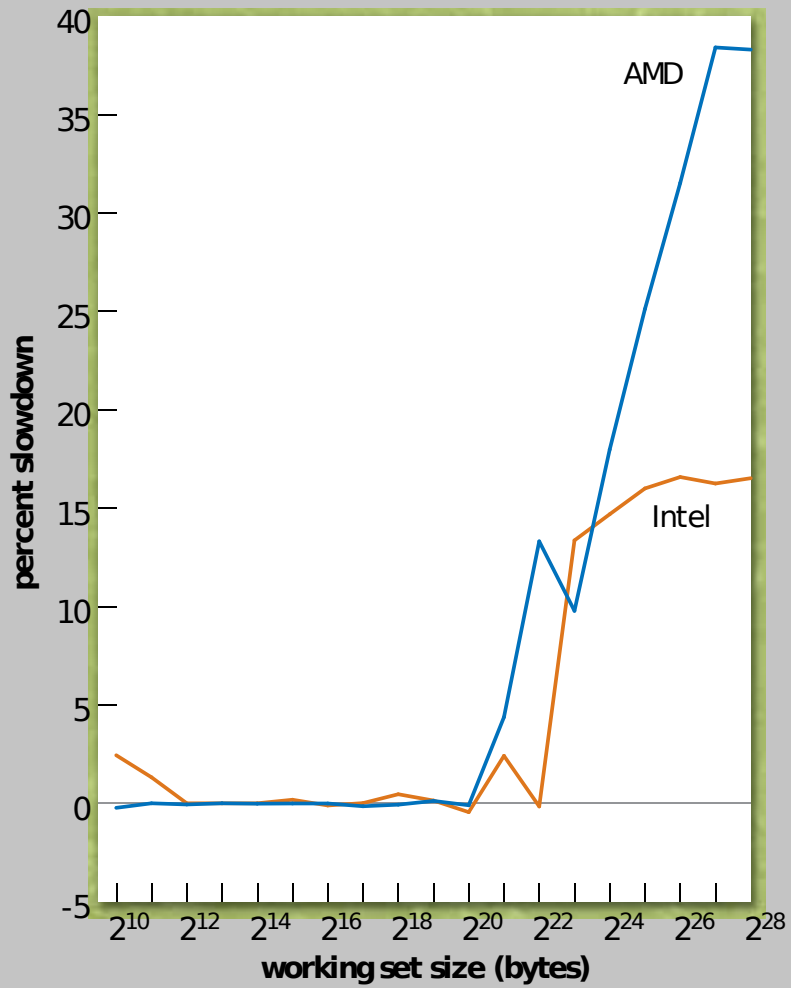


The Cost of Virtualization

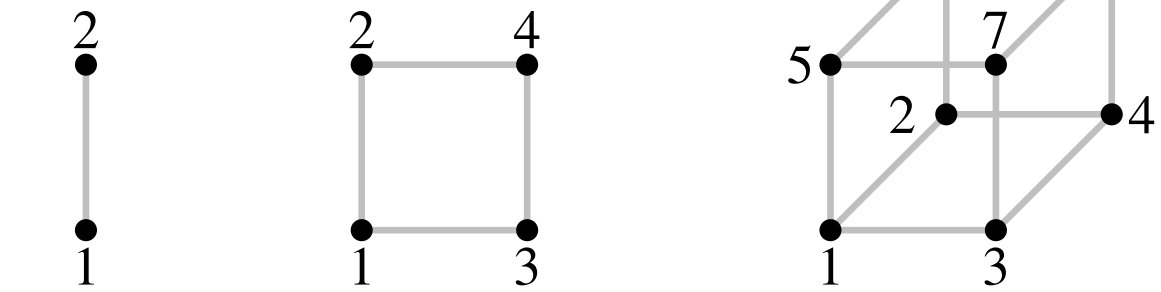
Guest Address Translation with Extended Page Table Trees



Slowdown of Program Running in Virtual Guest Domain

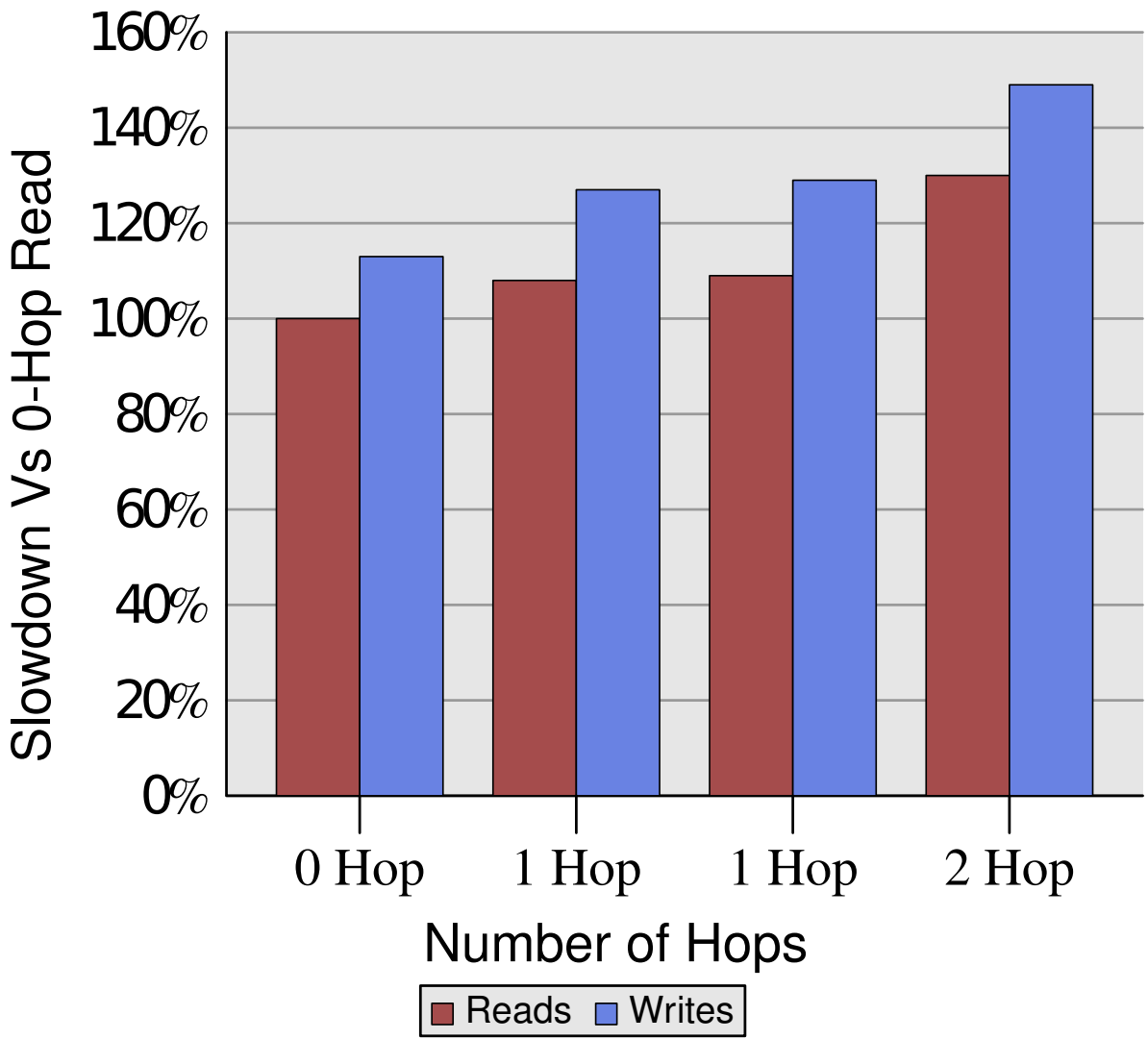


NUMA: Hypercubes



C = 1 C = 2 C = 3

NUMA: Read/Write Performance with Multiple Nodes



NUMA: Operating on Remote Memory

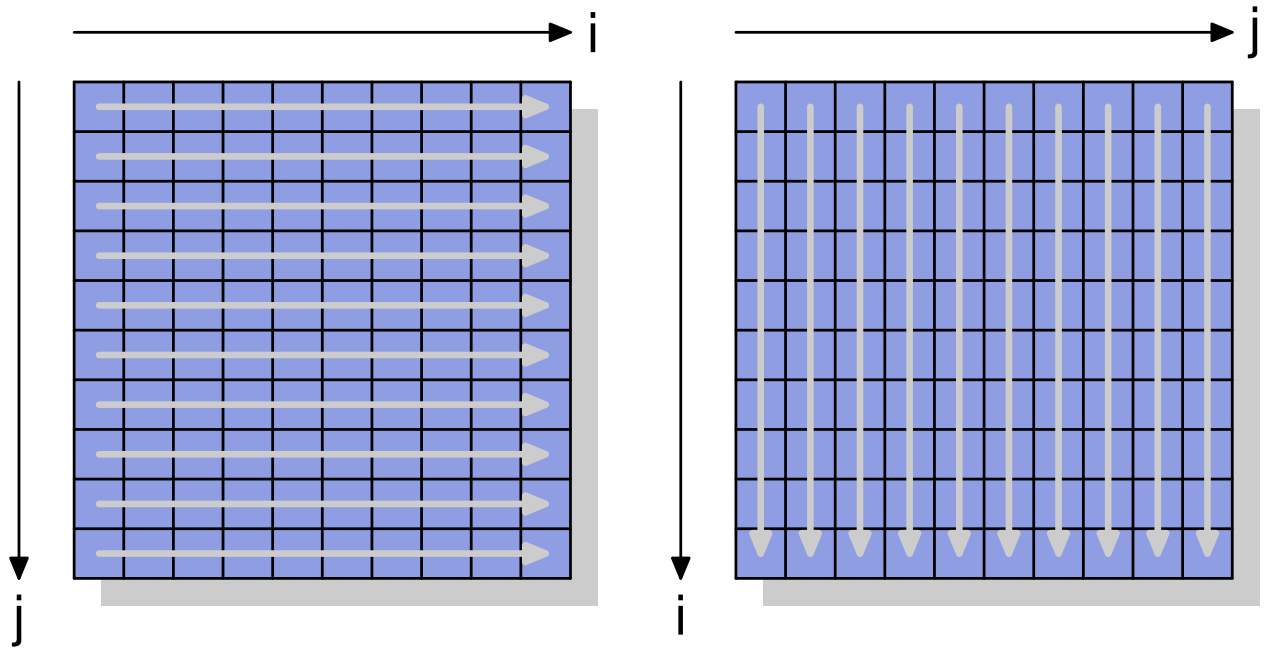
Non-temporal write operations

```
#include <emmintrin.h>
void _mm_stream_si32(int *p, int a);
void _mm_stream_si128(int *p, __m128i a);
void _mm_stream_pd(double *p, __m128d a);

#include <xmmintrin.h>
void _mm_stream_pi(__m64 *p, __m64 a);
void _mm_stream_ps(float *p, __m128 a);

#include <ammintrin.h>
void _mm_stream_sd(double *p, __m128d a);
void _mm_stream_ss(float *p, __m128 a);
```

Write-combining

```
#include <emmintrin.h>
void setbytes(char *p, int c)
{
  __m128i i = _mm_set_epi8(c, c, c, c,
                           c, c, c, c,
                           c, c, c, c,
                           c, c, c, c);
  _mm_stream_si128((__m128i *)&p[0], i);
  _mm_stream_si128((__m128i *)&p[16], i);
  _mm_stream_si128((__m128i *)&p[32], i);
  _mm_stream_si128((__m128i *)&p[48], i);
}
```

Matrix Access Pattern



Timing Matrix Initialization

	Inner Loop Increment	
	Row	Column
Normal	0.048s	0.127s
Non-Temporal	0.048s	0.160s

Matrix multiplication

$$(AB)_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{kj}$$

Matrix multiplication

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    for (k = 0; k < N; ++k)
      res[i][j] += mul1[i][k] * mul2[k][j];
```

Matrix multiplication: transpose

$$(AB)_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{jk}^{T}$$

Matrix multiplication: transpose

```
double tmp[N][N];

for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    tmp[i][j] = mul2[j][i];

for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    for (k = 0; k < N; ++k)
      res[i][j] += mul1[i][k] * tmp[j][k];
```

Matrix multiplication: transpose

	Original	Transposed
Cycles	16,765,297,870	3,922,373,010
Relative	100%	23.4%

Matrix multiplication: two iterations of the middle loop

```
gcc -DCLS=$(getconf LEVEL1_DCACHE_LINESIZE) ...

#define SM (CLS / sizeof (double))

for (i = 0; i < N; i += SM)
  for (j = 0; j < N; j += SM)
    for (k = 0; k < N; k += SM)
      for (i2 = 0, rres = &res[i][j],
           rmul1 = &mul1[i][k]; i2 < SM;
           ++i2, rres += N, rmul1 += N)
        for (k2 = 0, rmul2 = &mul2[k][j];
             k2 < SM; ++k2, rmul2 += N)
          for (j2 = 0; j2 < SM; ++j2)
            rres[j2] += rmul1[k2] * rmul2[j2];
```

Matrix multiplication

	Original	Transposed	Sub-Matrix	Vectorized
Cycles	16,765,297,870	3,922,373,010	2,895,041,480	1,588,711,750
Relative	100%	23.4%	17.3%	9.47%

Spreading Over Multiple Cache Lines

Layout of a data structure

```
struct foo {
  int a;
  long fill[7];
  int b;
};
```

Output of pahole Run

```
struct foo {
        int a; /* 0 4 */
        /* XXX 4 bytes hole, try to pack */
        long int fill[7]; /* 8 56 */
        /* --- cacheline 1 boundary (64 bytes) --- */
        int b; /* 64 4 */
}; /* size: 72, cachelines: 2 */
   /* sum members: 64, holes: 1, sum holes: 4 */
   /* padding: 4 */
   /* last cacheline: 8 bytes */
```

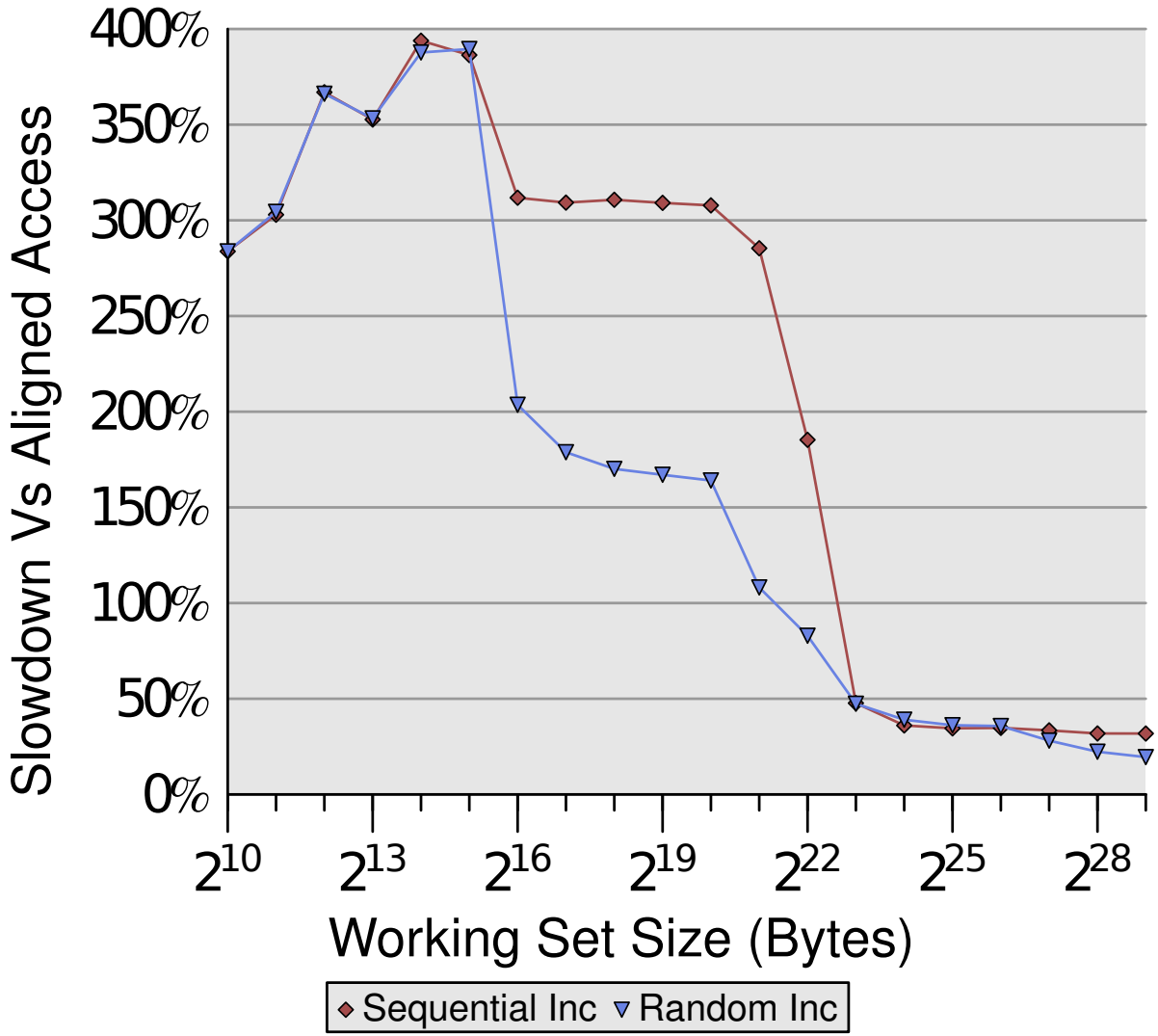
Alignment

```
#include <stdlib.h>

int posix_memalign(void **memptr,
                   size_t align,
                   size_t size);

struct strtype variable
   __attribute((aligned(64)));
```

does not work for arrays

```
struct strtype {
...members...
} __attribute((aligned(64)));
```

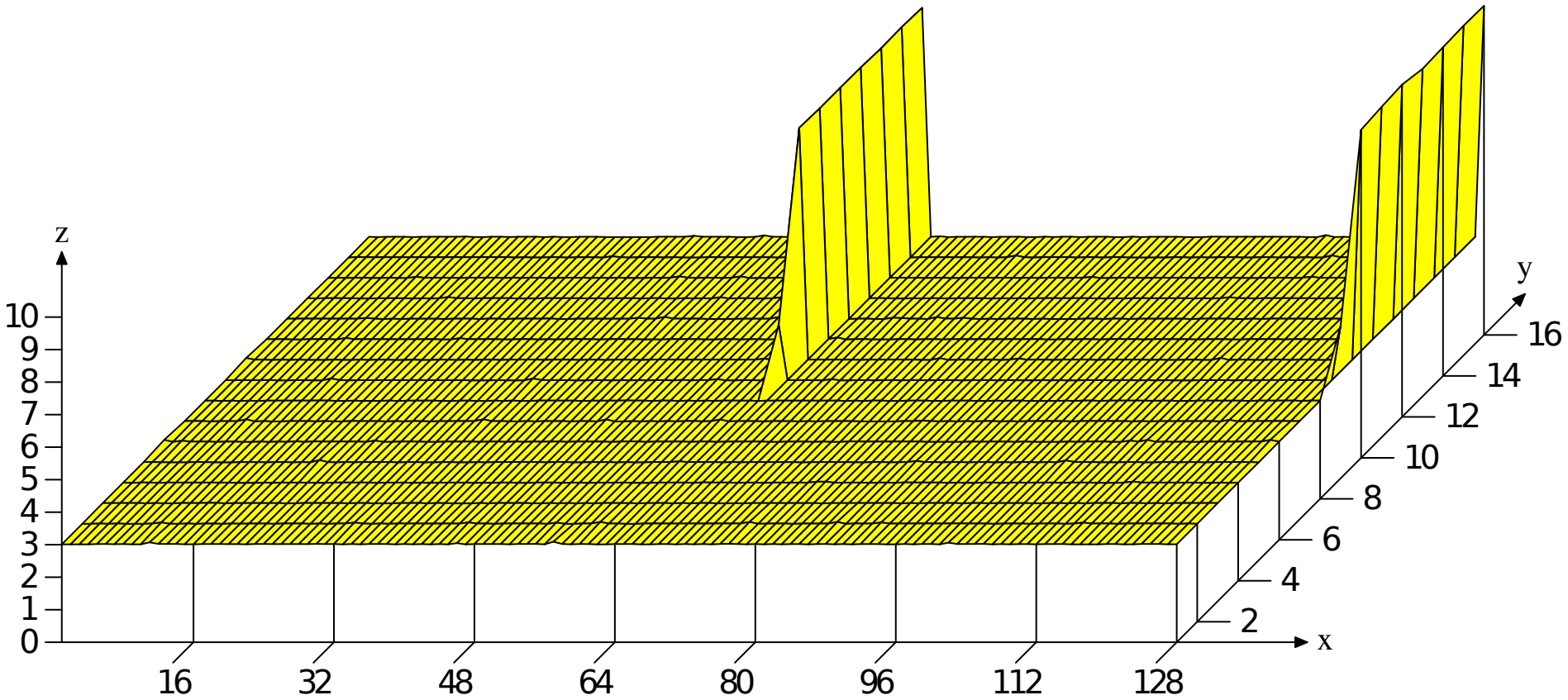
Overhead of Unaligned Accesses



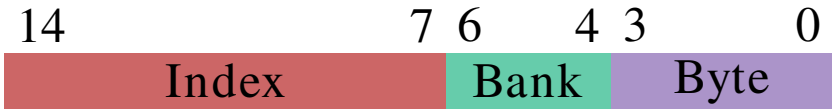
Rearrange data structures

```
struct order {
  double price;
  bool paid;
  const char *buyer[5];
  long buyer_id;
};
```

Cache Associativity Effects



Bank Address of L1d on AMD



AMD: L1d can receive two data words per cycle

Optimizing L1i Cache Access

- reduce the code footprint as much as possible. This has to be balanced with optimizations like loop unrolling and inlining.
- code execution should be linear without bubbles.
- aligning code when it makes sense.

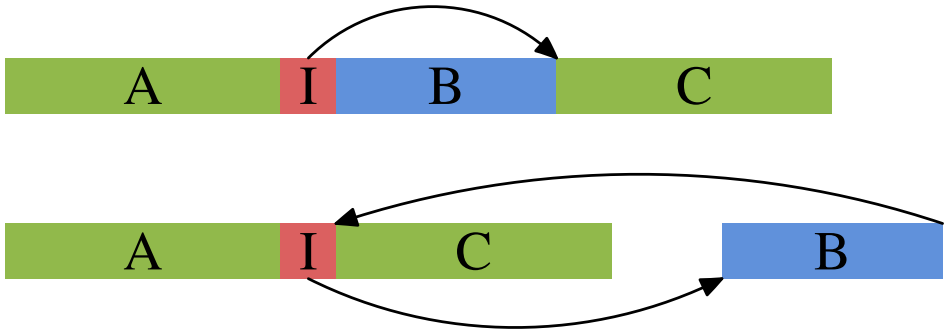
Inlining vs. Not

```
start f1
  code f1
  inlined inlcand
  more code f1
end f1

start f2
  code f2
  inlined inlcand
  more code f2
end f1
```

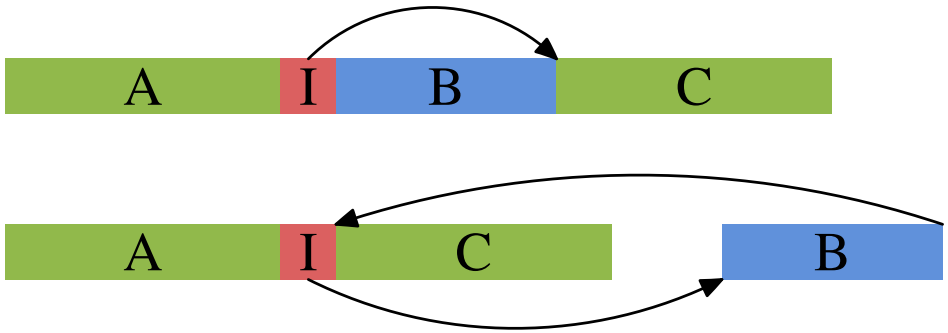
```
start inlcand
  code inlcand
end inlcand

start f1
  code f1
end f1

start f2
  code f2
end f1
```

Conditional jumps

```
void fct(void) {
    ... code block A ...
  if (condition)
    inlfct()
  ... code block C ...
}
```



gcc: conditional jumps




```
long __builtin_expect(long EXP, long C);

#define unlikely(expr) __builtin_expect(!!(expr), 0)
#define likely(expr) __builtin_expect(!!(expr), 1)

if (likely(a > 1))

-freorder-blocks
```

Alignment of code

- at the beginning of functions;
- at the beginning of basic blocks which are reached only through jumps;
- to some extent, at the beginning of loops

`-falign-functions=N`

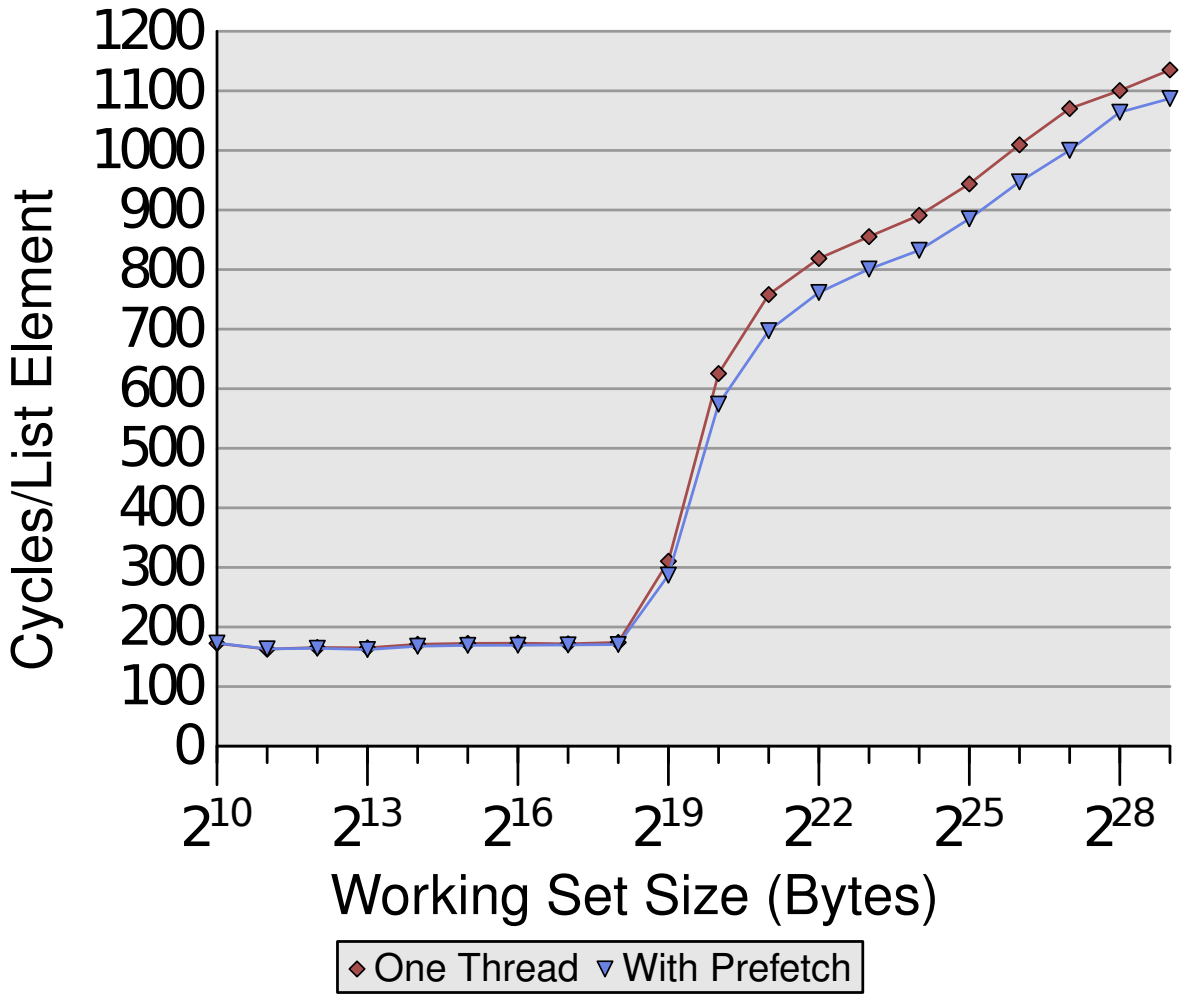
`-falign-jumps=N`

`-falign-loops=N`

Software Prefetching

```
#include <xmmintrin.h>

enum _mm_hint
{
  _MM_HINT_T0 = 3,
  _MM_HINT_T1 = 2,
  _MM_HINT_T2 = 1,
  _MM_HINT_NTA = 0
};

void _mm_prefetch(void *p,
                  enum _mm_hint h);
```

Average with Prefetch



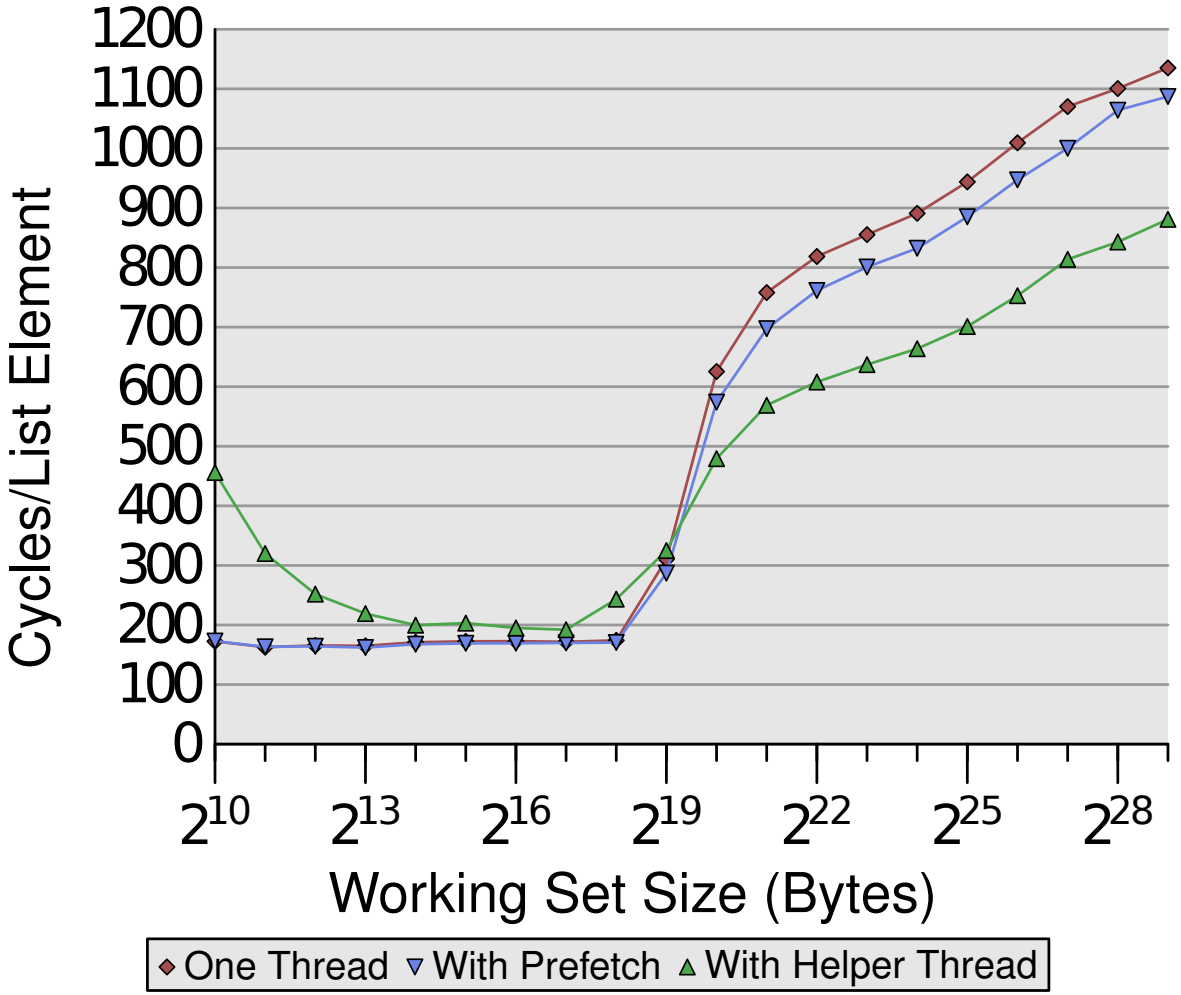
gcc: loop is iterating over an array

`-fprefetch-loop-arrays`

Prefetch: Speculation (IA-64)

```
st8 [r4] = 12
ld8 r6 = [r8];;
add r5 = r6, r7;;
st8 [r18] = r5

ld8.a r6 = [r8];;
[... other instructions ...]
st8 [r4] = 12
ld8.c.clr r6 = [r8];;
add r5 = r6, r7;;
st8 [r18] = r5
```

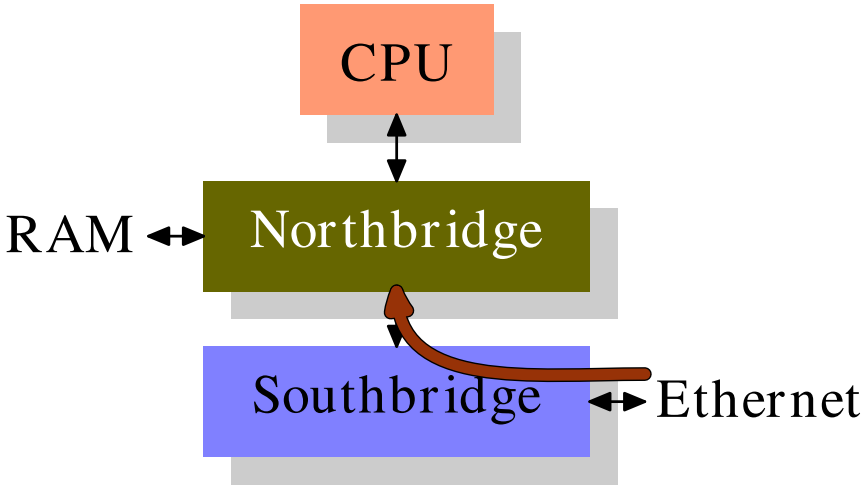
Helper Threads

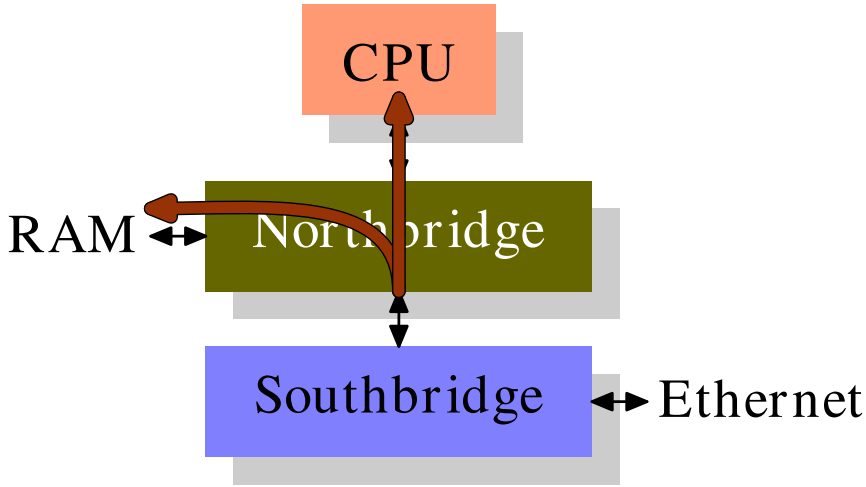


libNUMA: hyper-threads

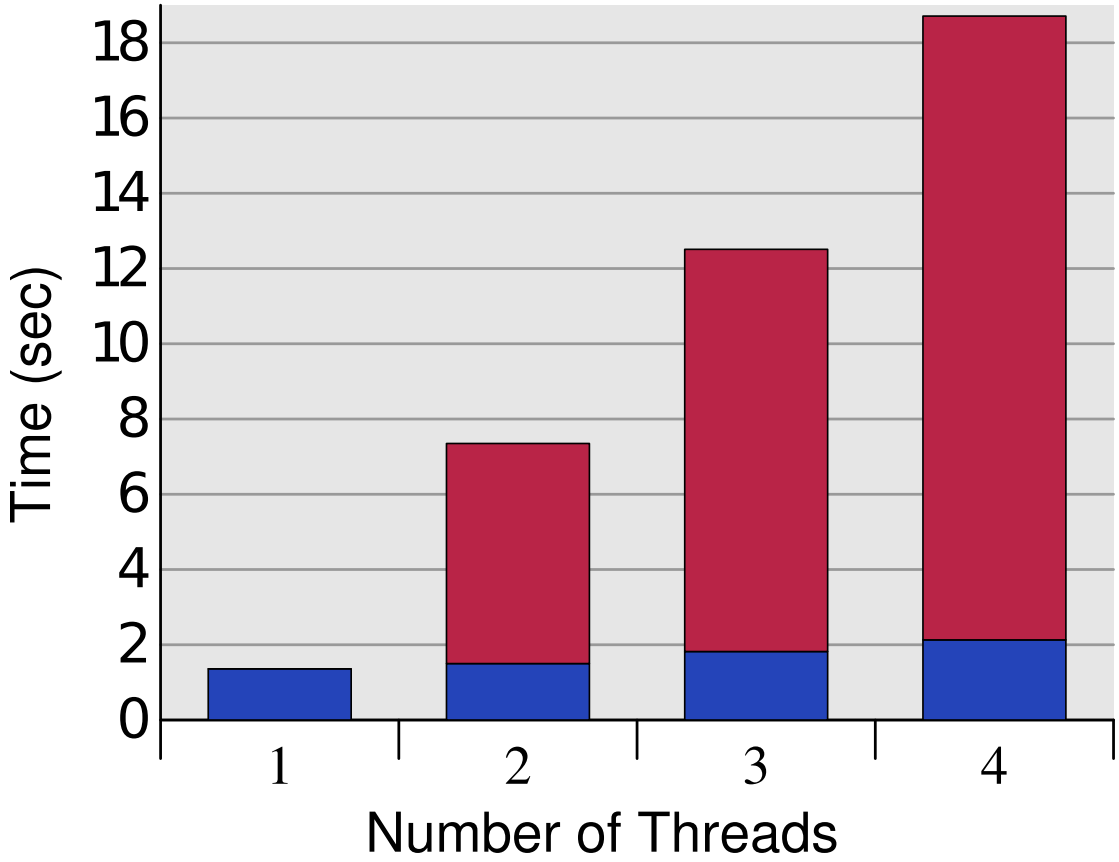
```
#include <libNUMA.h>

ssize_t NUMA_cpu_level_mask(size_t destsize,
                            cpu_set_t *dest,
                            size_t srcsize,
                            const cpu_set_t*src,
                            unsigned int level);

cpu_set_t self;
NUMA_cpu_self_current_mask(sizeof(self),
                           &self);
cpu_set_t hts;
NUMA_cpu_level_mask(sizeof(hts), &hts,
                    sizeof(self), &self, 1);
CPU_XOR(&hts, &hts, &self);
```

Direct Cache Access

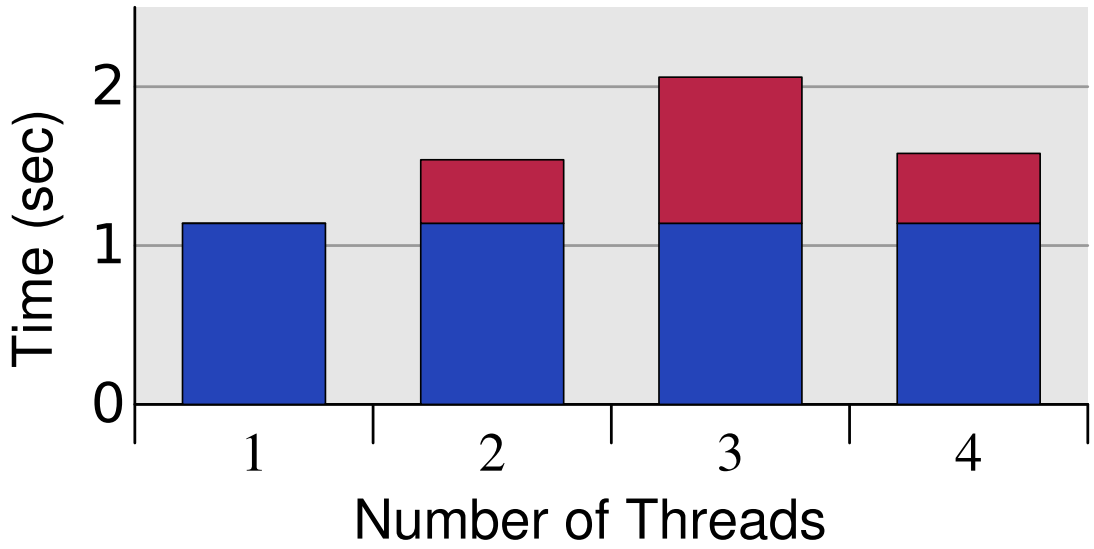




Concurrent Cache Line Access Overhead



Overhead, Quad Core



Advices: Concurrency Optimizations

- Separate at least read-only (after initialization) and read-write variables.
- Group read-write variables which are used together into a structure.
- Move read-write variables which are often written to by different threads onto their own cache line.
- If a variable is used by multiple threads, but every use is independent, move the variable into TLS.

Atomicity Optimizations

- Bit Test
- Load Lock/Store Conditional (LL/SC)
- Compare-and-Swap (CAS)
- Atomic Arithmetic

Atomicity Optimizations

```
int curval;
int newval;
do {
  curval = var;
  newval = curval + addend;
} while (CAS(&var, curval, newval));
```

```
int curval;
int newval;
do {
  curval = LL(var);
  newval = curval + addend;
} while (SC(var, newval));
```

Atomic Increment in a Loop

```
for (i = 0; i < N; ++i)
  __sync_add_and_fetch(&var,1);
```

1. Add and Read Result

```
for (i = 0; i < N; ++i)
  __sync_fetch_and_add(&var,1);
```

2. Add and Return Old Value

```
for (i = 0; i < N; ++i) {
  long v, n;
  do {
    v = var;
    n = v + 1;
  } while (!__sync_bool_compare_and_swap(&var,
                                         v,n));
}
```

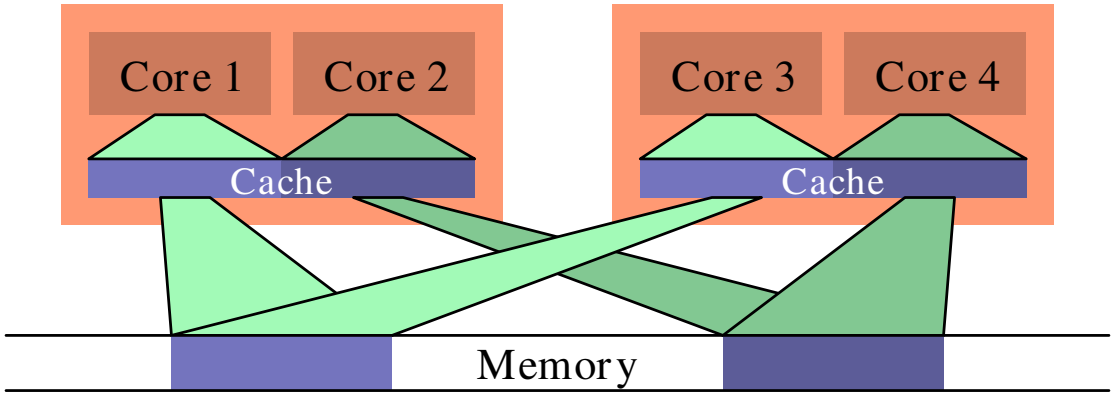
3. Atomic Replace with New Value

1. Exchange Add	2. Add Fetch	3. CAS
0.23s	0.21s	0.73s

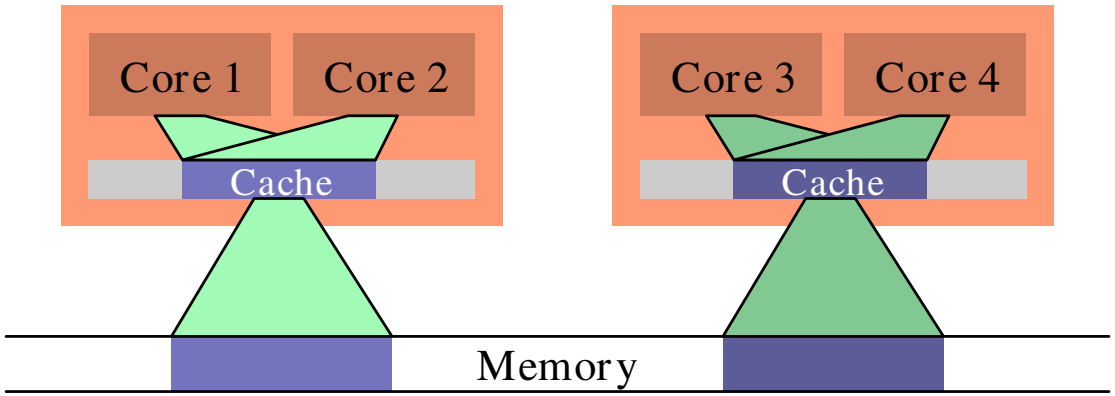
Atomic Arithmetic

```
cmpl $0, multiple_threads
je 1f
lock
1: add $1, some_var
```

Inefficient Scheduling



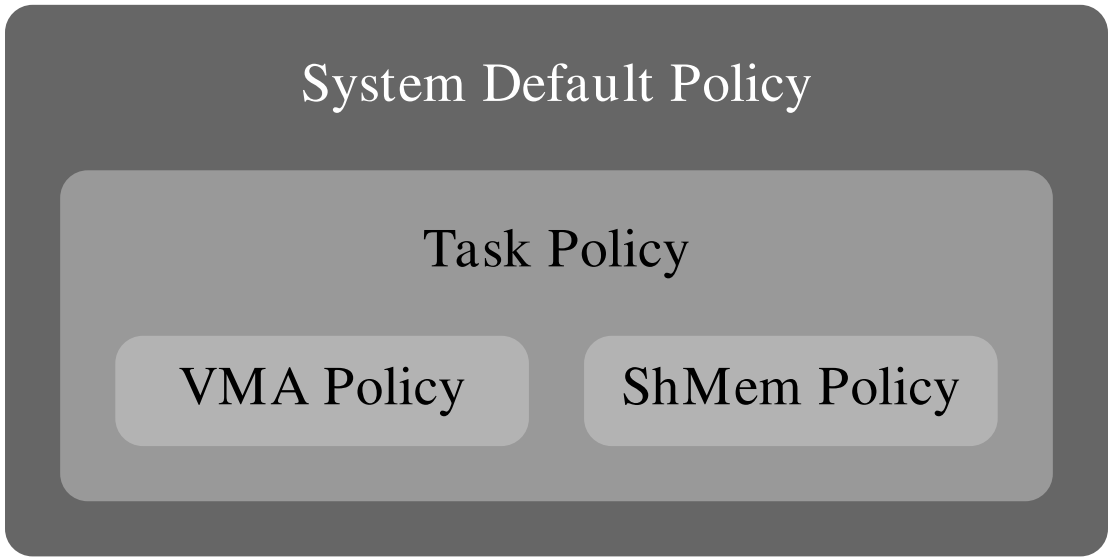
Efficient Scheduling



Affinity of a thread

```
#include <sched.h>

int sched_setaffinity(pid_t pid, size_t size,
                      const cpu_set_t *cpuset);
int sched_getaffinity(pid_t pid, size_t size,
                      cpu_set_t *cpuset);

#include <pthread.h>

int pthread_setaffinity_np(pthread_t th,
                           size_t size,
                           const cpu_set_t *cpuset);
int pthread_getaffinity_np(pthread_t th,
                           size_t size,
                           cpu_set_t *cpuset);
```

Memory Policy Hierarchy



Linux system calls

`<numaif.h>` (libnuma, numactl package)

mbind
: Select binding of specified memory pages.

set_mempolicy
: Set the default memory binding policy.

get_mempolicy
: Get the default memory binding policy.

migrate_pages
: Migrate all pages of a process on a given set of nodes to a different set of nodes.

move_pages
: Move selected pages to given node or request node information about pages.

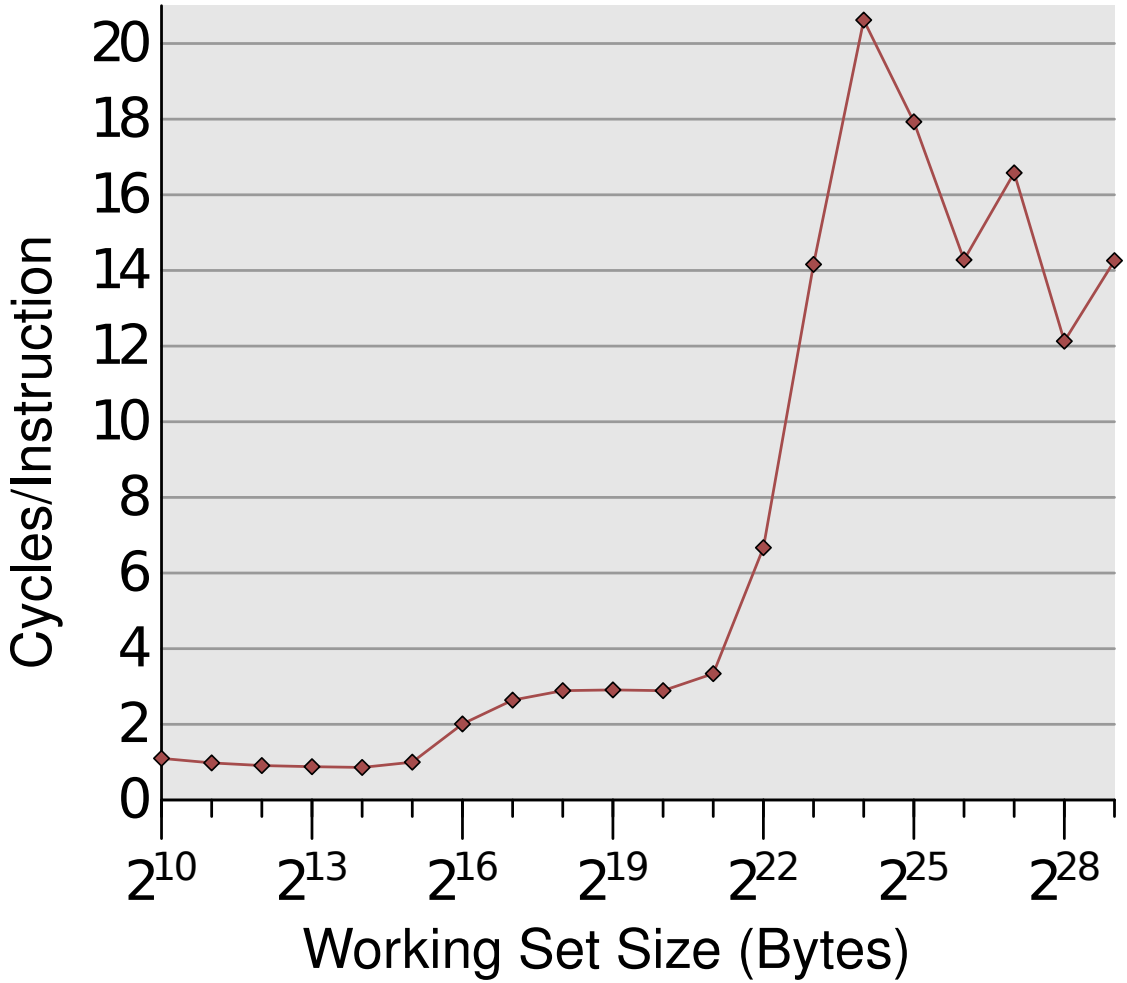
VMA Policy

```
#include <numaif.h>

long mbind(void *start, unsigned long len,
           int mode,
           unsigned long *nodemask,
           unsigned long maxnode,
           unsigned flags);
```

Tools

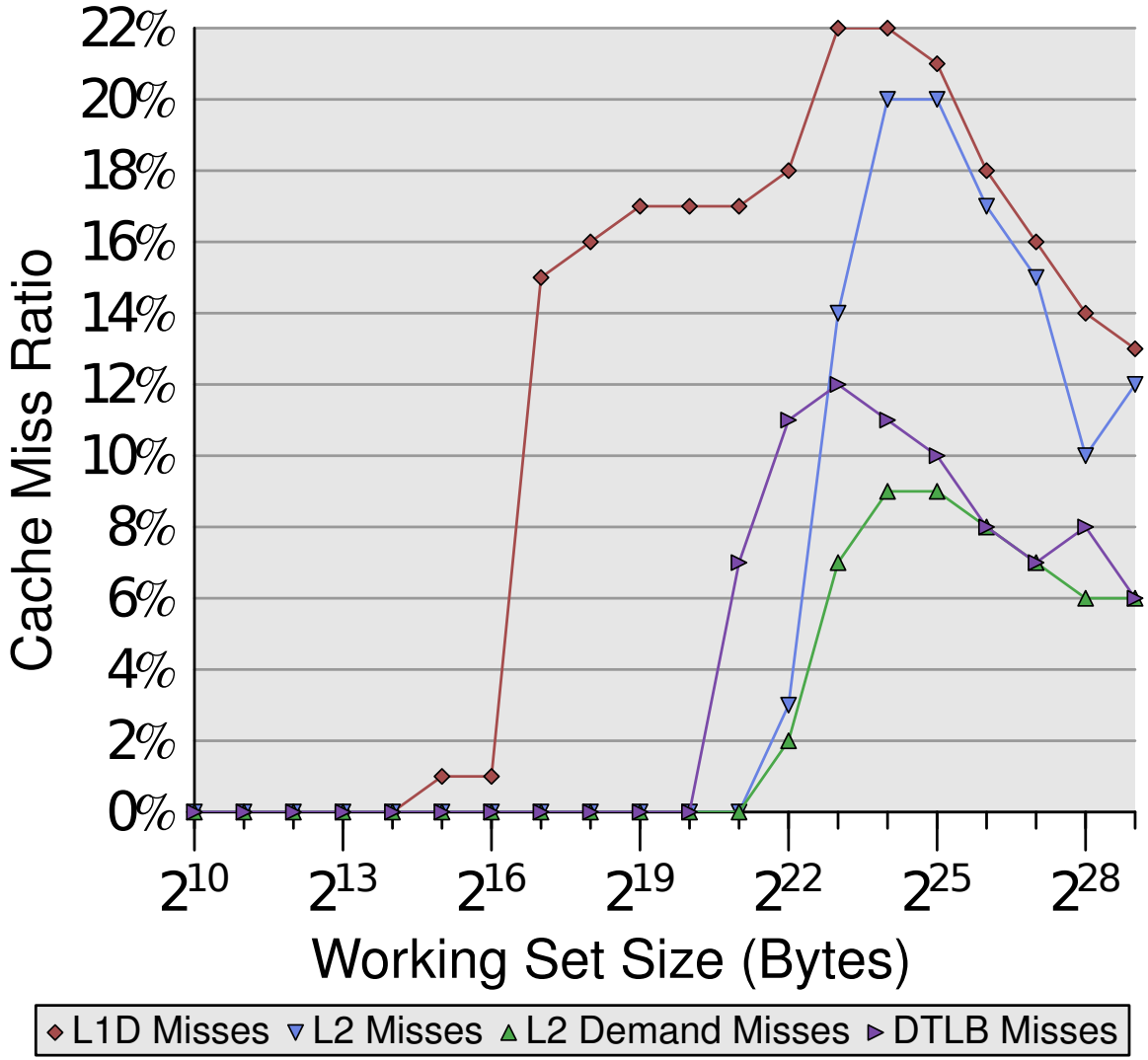
- oprofile
- pfmon
- time
- cachegrind
- massif
- memusage
- gcc
- pagein

Cycles per Instruction (Follow Random)



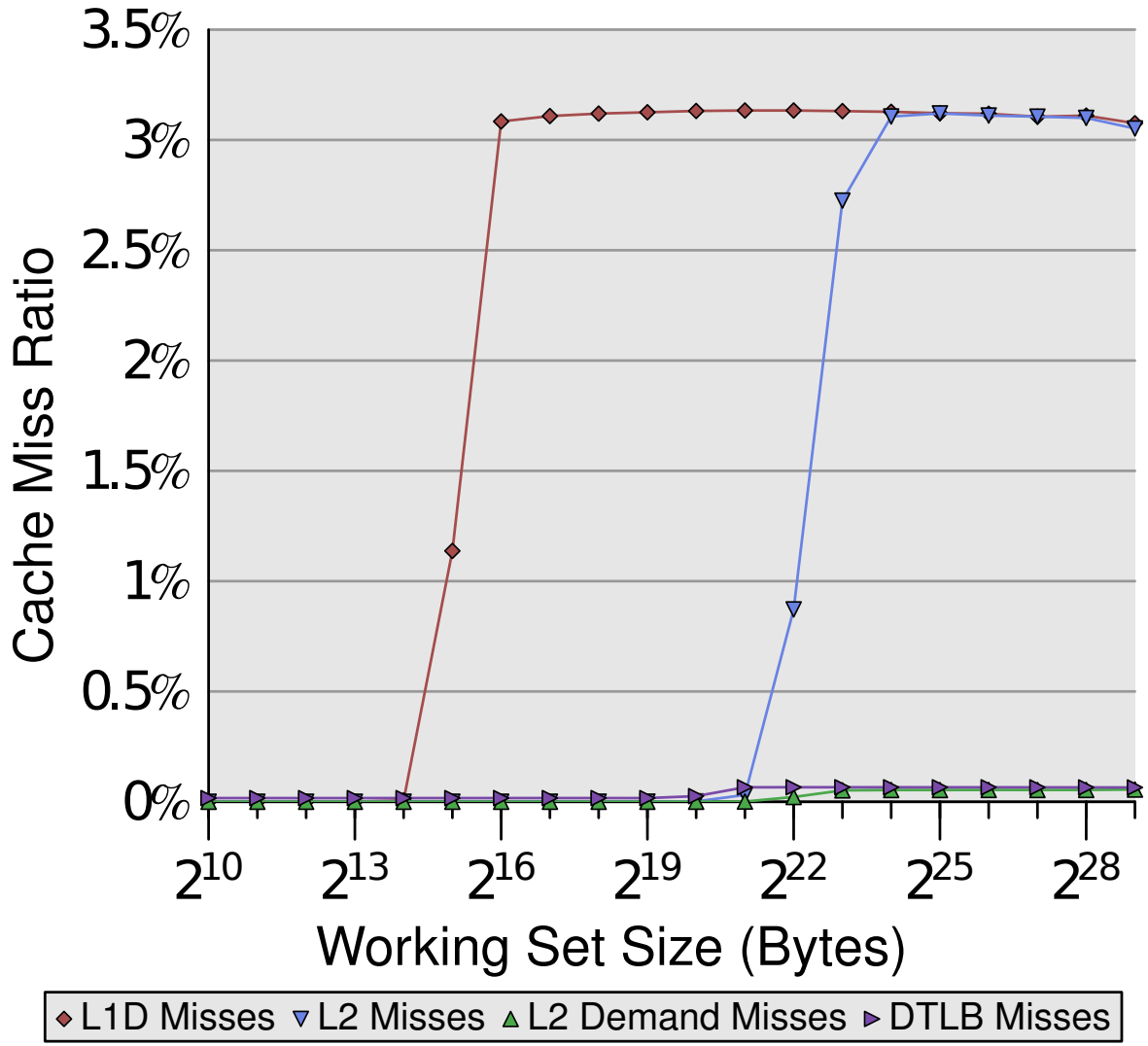
oprofile: `CPU_CLK_UNHALTED` / `INST_RETIRED`

Measured Cache Misses (Follow Random)



oprofile: `L1D_REPL`, `DTLB_MISSES`, `L2_LINES_IN`

Measured Cache Misses (Follow Sequential)



oprofile: `L1D_REPL`, `DTLB_MISSES`, `L2_LINES_IN`

Output of the time utility

```
$ \time ls /etc
[...]
0.00user 0.00system 0:00.02elapsed
  17%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (1major+335minor)pagefaults 0swaps
```

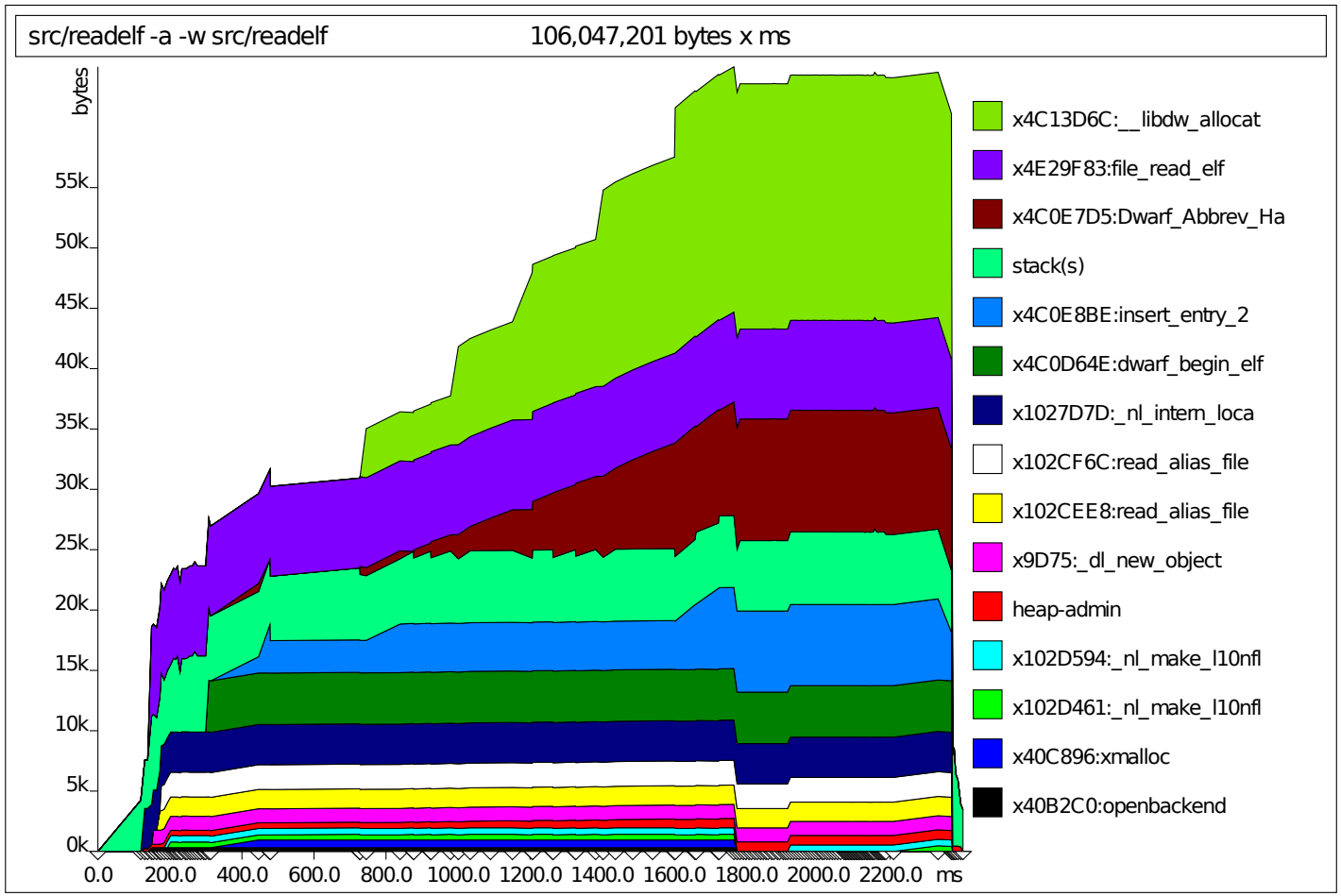
Cachegrind Summary Output

```
valgrind --tool=cachegrind command arg
```

```
==19645== I   refs:      152,653,497
==19645== I1  misses:         25,833
==19645== L2i misses:          2,475
==19645== I1  miss rate:        0.01%
==19645== L2i miss rate:        0.00%
==19645==
==19645== D   refs:       56,857,129  (35,838,721 rd + 21,018,408 wr)
==19645== D1  misses:         14,187  (    12,451 rd +      1,736 wr)
==19645== L2d misses:          7,701  (     6,325 rd +      1,376 wr)
==19645== D1  miss rate:         0.0% (       0.0%   +        0.0%  )
==19645== L2d miss rate:         0.0% (       0.0%   +        0.0%  )
==19645==
==19645== L2 refs:            40,020  (    38,284 rd +      1,736 wr)
==19645== L2 misses:          10,176  (     8,800 rd +      1,376 wr)
==19645== L2 miss rate:          0.0% (       0.0%   +        0.0%  )
```

Massif Output

```
valgrind --tool=massif command arg
```



malloc

gcc: Improving Branch Prediction

- `__builtin_expect` (`likely`, `unlikely`)
- profile-guided optimization (PGO)
  `-fprofile-generate`
  `-fprofile-use`

Output of the pagein Tool

```
0 0x3000000000 C           0 0x3000000B50: (within /lib64/ld-2.5.so)
1 0x 7FF000000 D        3320 0x3000000B53: (within /lib64/ld-2.5.so)
2 0x3000001000 C       58270 0x3000001080: _dl_start (in /lib64/ld-2.5.so)
3 0x3000219000 D      128020 0x30000010AE: _dl_start (in /lib64/ld-2.5.so)
4 0x300021A000 D      132170 0x30000010B5: _dl_start (in /lib64/ld-2.5.so)
5 0x3000008000 C    10489930 0x3000008B20: _dl_setup_hash (in /lib64/ld-2.5.so)
6 0x3000012000 C    13880830 0x3000012CC0: _dl_sysdep_start (in /lib64/ld-2.5.so)
7 0x3000013000 C    18091130 0x3000013440: brk (in /lib64/ld-2.5.so)
8 0x3000014000 C    19123850 0x3000014020: strlen (in /lib64/ld-2.5.so)
9 0x3000002000 C    23772480 0x3000002450: dl_main (in /lib64/ld-2.5.so)
```

Huge page

- System V shared memory interface
- filesystem `hugetlbfs`; `mmap`

```
key_t k = ftok("/some/key/file", 42);
int id = shmget(k, LENGTH,
                SHM_HUGETLB|IPC_CREAT
                |SHM_R|SHM_W);
void *a = shmat(id, NULL, 0);
```

```
int fd = open("/dev/hugetlb/file1",
              O_RDWR|O_CREAT, 0700);
void *a = mmap(NULL, LENGTH,
               PROT_READ|PROT_WRITE,
               fd, 0);
```

Follow with Huge Pages