# CS 6354: Memory Hierarchy II

31 August 2016



# Memory Hierarchy

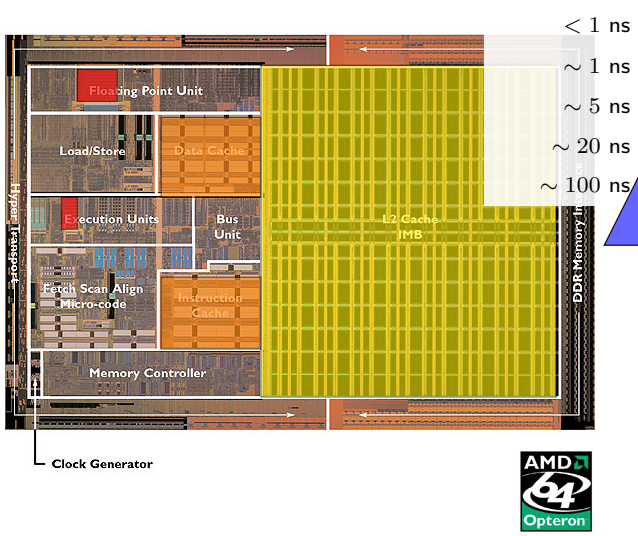



Image: approx 2004 AMD press image of Opteron die



# Last time

Smith, "Cache memories"

- Trace-based simulation of lots of cache parameters
- Overlap virtual to physical lookup and cache lookup

Bernstein, "Cache timing attacks on AES"

- Fighting for constant-time (with respect to secrets)
- Suggestions for architects
- Suggestions for crypto implementors



# Last time: Cache optimizations

| Technique | Improves Hit time | Miss penalty | Hit rate | Band-width |
|---|---|---|---|---|
| Increase block size | | N | Y | |
| Increase cache size | N | | Y | |
| Increase associavity | N | | Y | |
| Multilevel caches | | Y | | |
| Prioritize reads over writes | | Y | | |
| Virtual-index = Physical | Y | | | |
| Pipelined cache accesses | | | | Y |
| Non-blocking caches | | Y | | Y |
| Prefetching | | Y | Y | |
| Way-prediction | Y | | | |

\+ complexity costs

(adapted from tables in H&P B and H&P 2.2)

# Homework 1

Checkpoint due 12 September

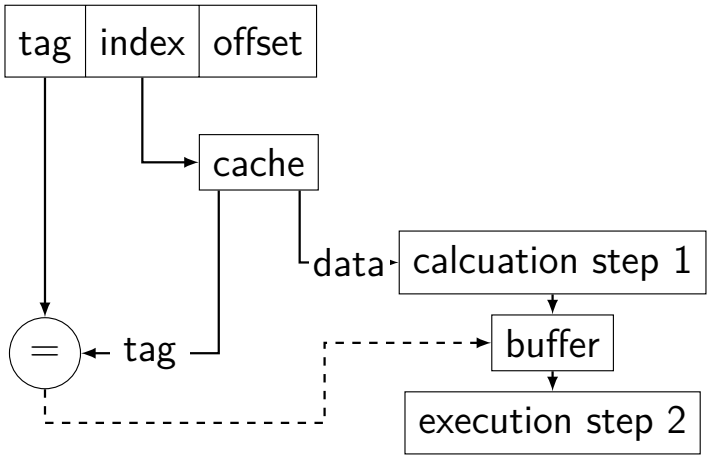
Intuition: 32KB much faster than 34KB, then 32KB cache

Required for checkpoint:

* For each data or unified (data and instruction) cache:
  - The size of that cache
  - The size of blocks (AKA lines) in that cache
* For each data or unified TLB:
  - The size (number of entries) of that TLB
* The single-core sequential throughput (read and write) of main memory
* The single-core random throughput (read and write) of main memory

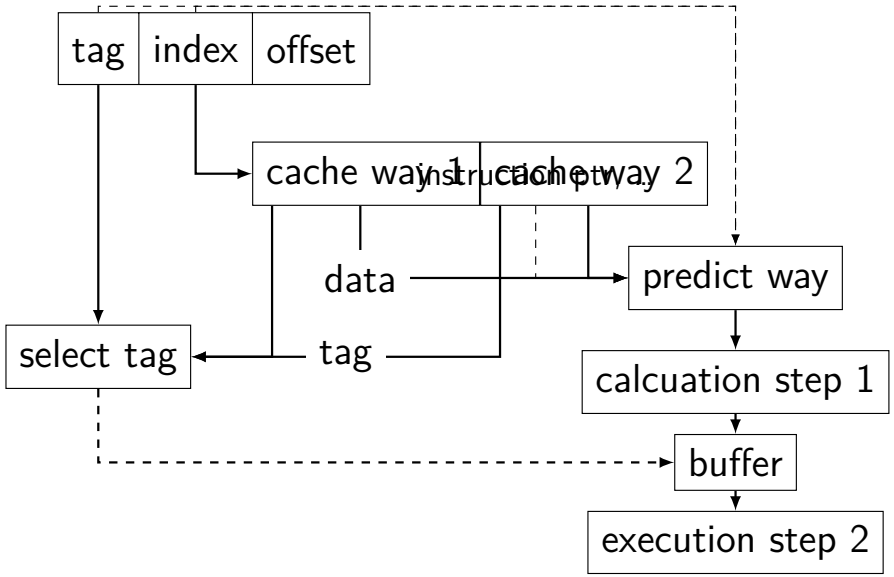


# Avoiding associativity



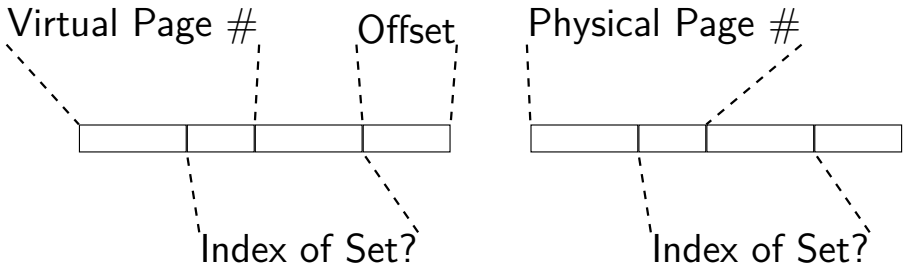



# Way prediction






# Why not direct-mapped?

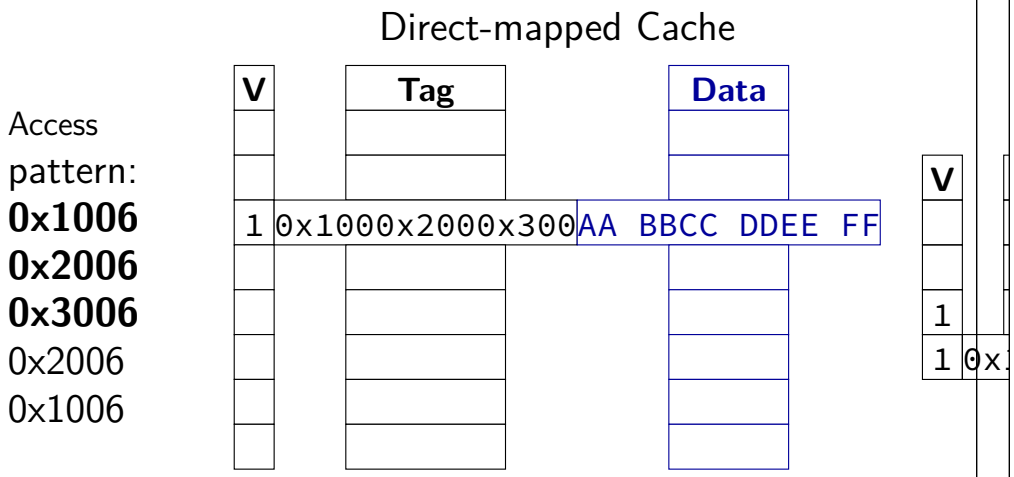



Avoid virtual caches

Mitigation: way prediction

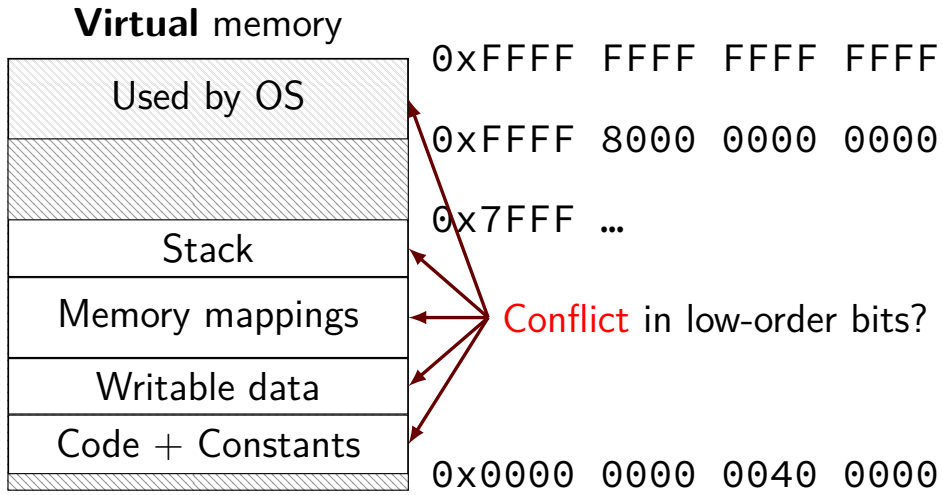
Different HW speed tradeoffs today?

# Victim Caches






# Different kinds of memory






# Old prefetch strategies

Prefetch always

Fetch next on miss

Tagged prefetch — next on non-prefetch use

Common goal: sequential access patterns



# Sequential access patterns

Examples?

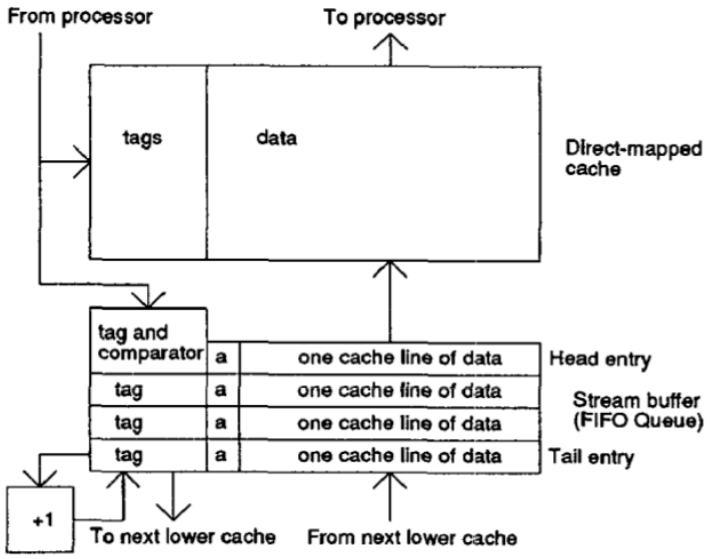
Instructions

Dense matrix/array math

String processing

Some database operations

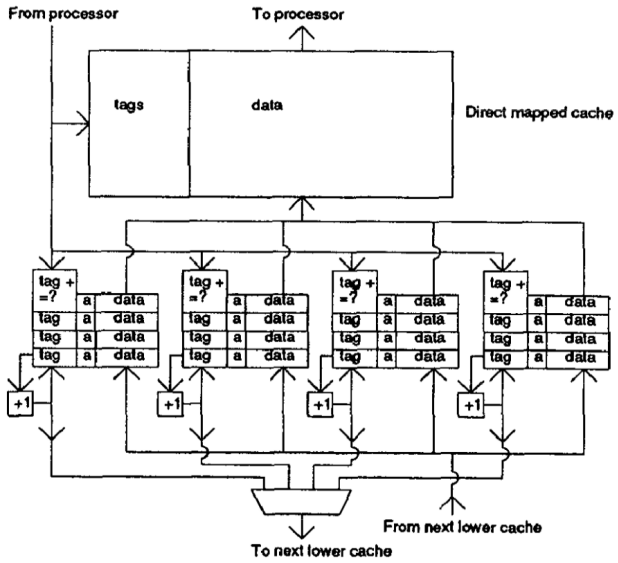
# Stream buffers




hit: shift up
miss: clear



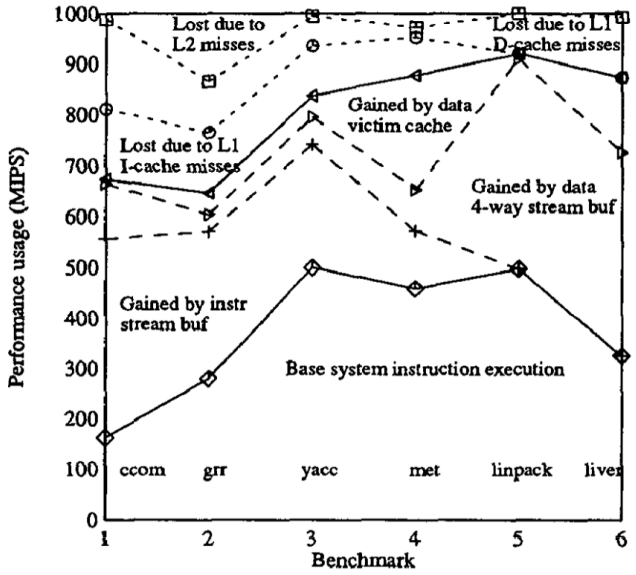
# Multi-way stream buffers




hit: shift up
miss: clear LRU



# Performance Results






# Prefetching on recent Intel (1)

From the Intel Optimization Manual on Sandy Bridge:

Two hardware prefetchers load data to the L1 DCache:

- **Data cache unit (DCU) prefetcher**. This prefetcher ... is triggered by an ascending access to very recently loaded data.

...

- **Instruction pointer (IP)-based stride prefetcher**. This prefetcher keeps track of individual load instructions. If a load instruction is detected to have a regular stride, then a prefetch is sent to the next address which is the sum of the current address and the stride. ...

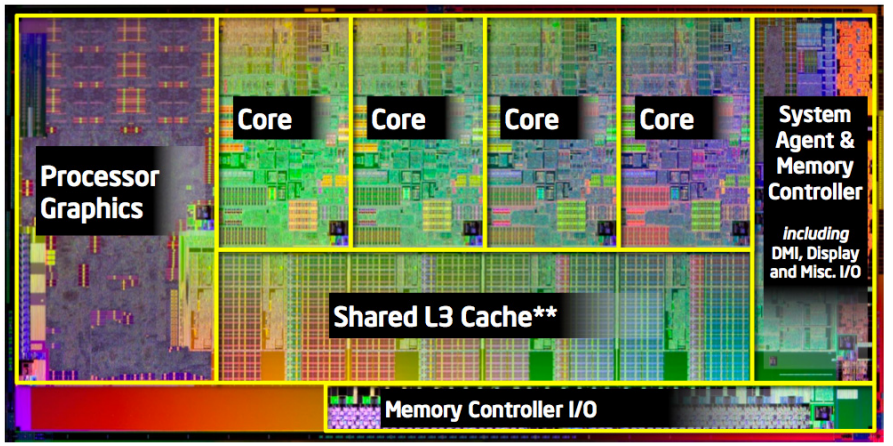
# Prefetching on recent Intel (2)

From the Intel Optimization Manual on Sandy Bridge:

> The following two hardware prefetchers fetched data from memory to the L2 cache and last level cache:
> **Spatial Prefetcher**: This prefetcher strives to complete every cache line fetched to the L2 cache with the pair line that completes it to a 128-byte aligned chunk.
> **Streamer**: This prefetcher monitors read requests from the L1 cache for ascending and descending sequences of addresses. Monitored read requests include ... load and store operations and ... the [L1] hardware prefetchers, and ... code fetch.



# Sandy Bridge die




via anandtech (original is Intel press photo??)



# Within each core

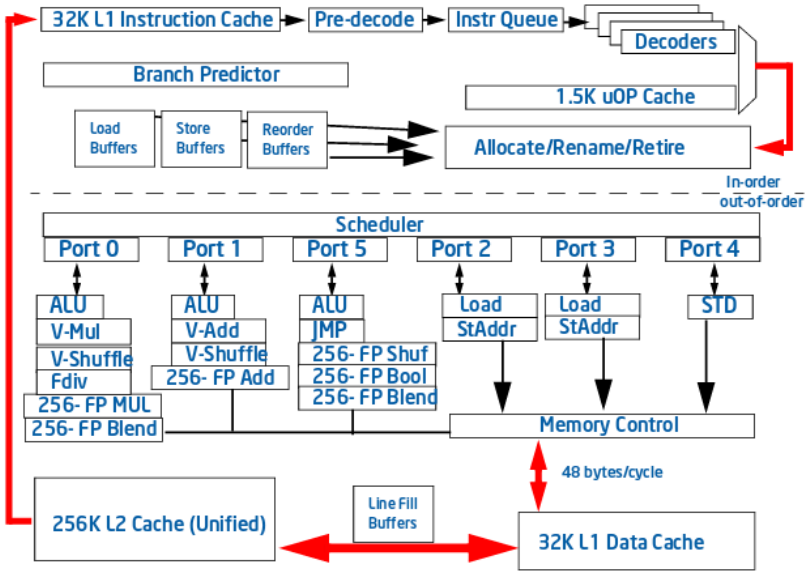


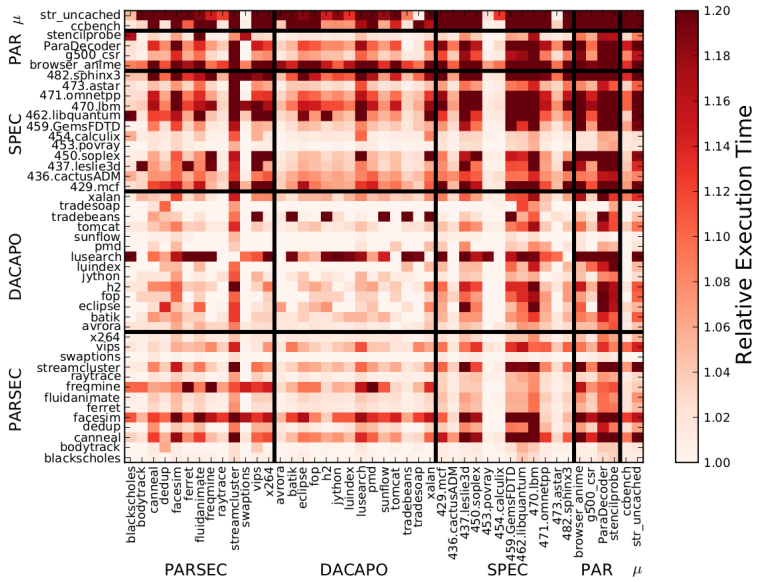

Image: Intel's Optimization Reference Manual



# Cook's Benchmark Categorization

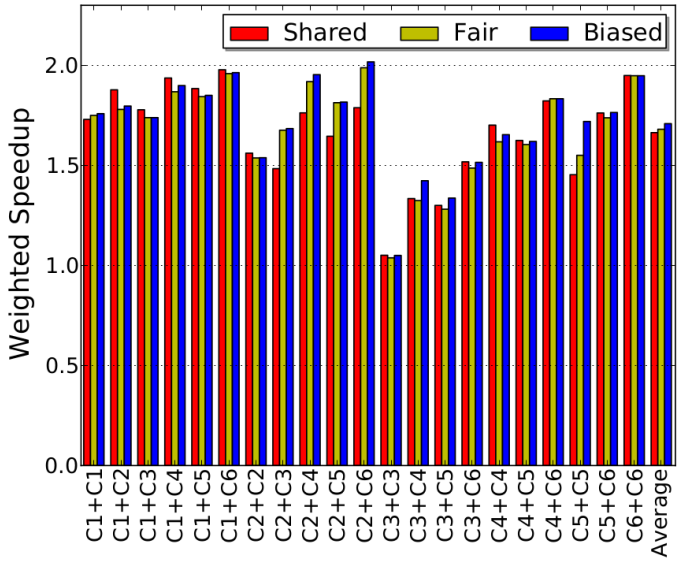
- Number of threads
- Last level cache size
- Prefetchers
- Memory bandwidth

# Interference between programs






# Why a shared last-level cache?



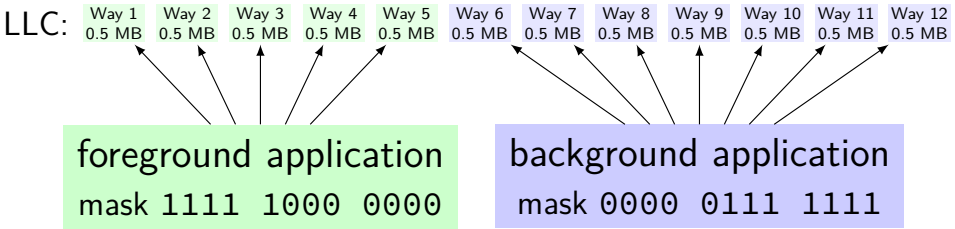



# Sandy Bridge's cache partitioning

12-way cache

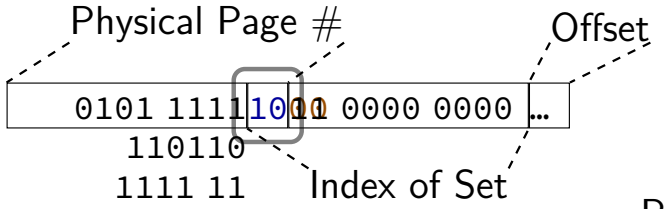
way — 'column' of set-associative cache
each way is like a direct-mapped cache

Mask for which ways are used to store things on miss

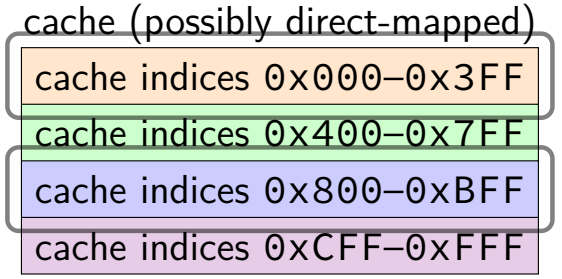




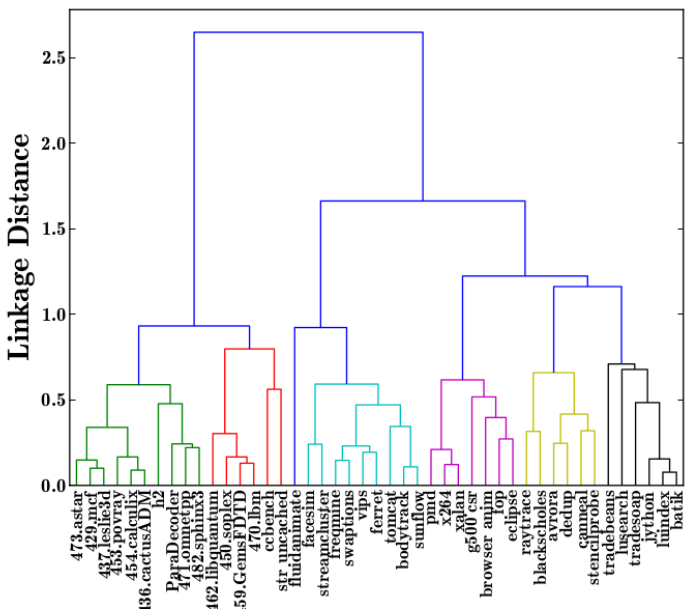

# Page coloring




Page colors:
00, 01, 10, 11

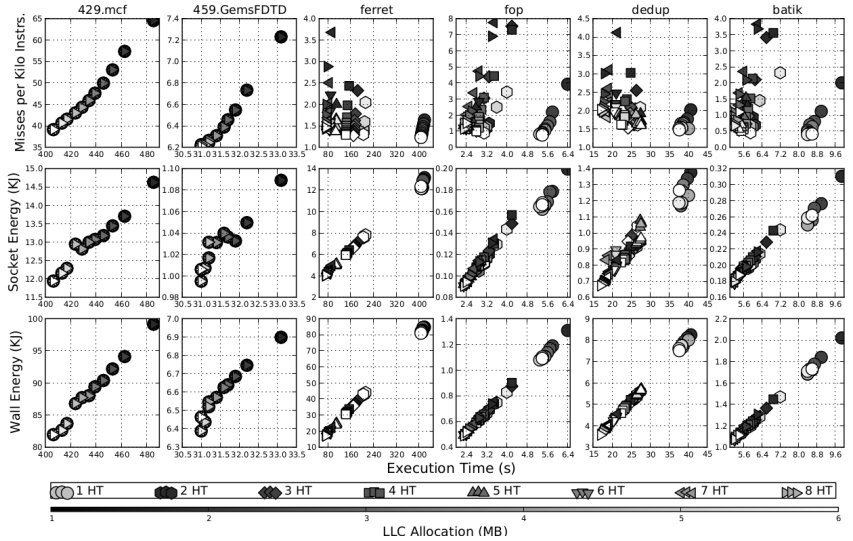


# Experiment Design






# Energy: Race-to-Halt






# Phases

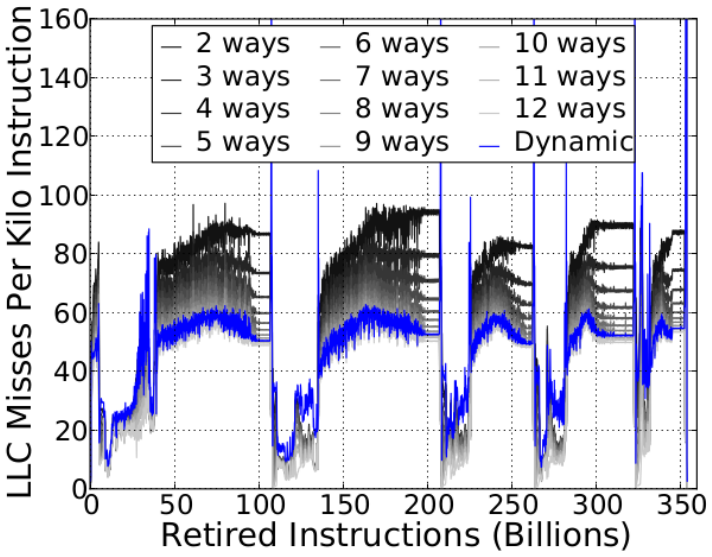



Figure 12: `429.mcf` LLC MPKI phase changes with different



# Dynamic partitioning

Dynamic partitioning inputs:

LLC misses over 100 ms, every 100 ms

Thresholds for detecting changes

Increase to max allocation — then decrease slowly

# Reproducibility