A Cache Conflict Analysis Tool

Vincent Weaver, Cornell University

CASC, Martin Schulz

Lawrence Livermore National Laboratory

11 August 2005

Abstract

The Cache Conflict Analysis Tool takes memory access traces from an instrumented program and runs them through a cache simulator, tracking which data structures conflict with each other. We will use the results of the tool to perform automatic cache conscious data placement, resulting in increased program performance.

Background

- The cache location of a data structure is dependent on its location in main memory.
- The layout of data (both static and dynamic) in main memory can dramatically affect cache performance due to conflicts, and can adversely affect performance of an application.
- Compilers typically do not take cache conflict behavior into account when creating an executable, nor does the memory subsystem or operating system at time of execution.
- By analyzing cache behavior with this tool, hopefully better data placements can be achieved.

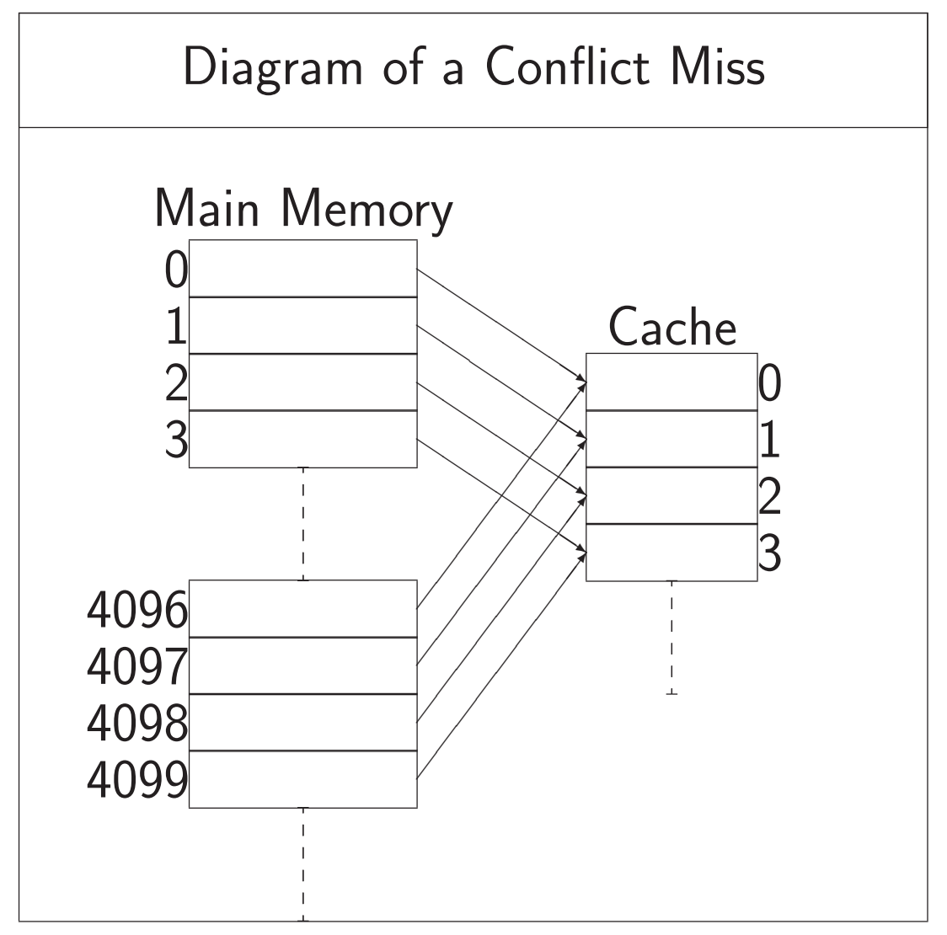
Below is an example of how in a direct mapped 4k cache with a 32 byte blocksize, every 4096th byte in main memory will map to the same area of cache. So if you have two data structures in use at the same time that are 4096 bytes apart, both will not fit in the cache at the same time and performance will be adversely affected.

Pseudo-code of a Conflict Miss

```
Allocate A[512] (allocated at offset 0)
Allocate B[512] (allocated at offset 4096)

Loop i=0..N
    Loop j=0..511
        A[j]=A[j]+B[j]
```

Without conflicts, A and B should be in the cache after the first access. Due to conflicts, each access causes a miss.



Results

Below are some results from running different benchmarks through the tool, with a simulated cache configuration similar to that of a Pentium 4 (8K 4-way 64byte L1, 512K 8-way 64byte L2)

equake from the spec2k benchmarks

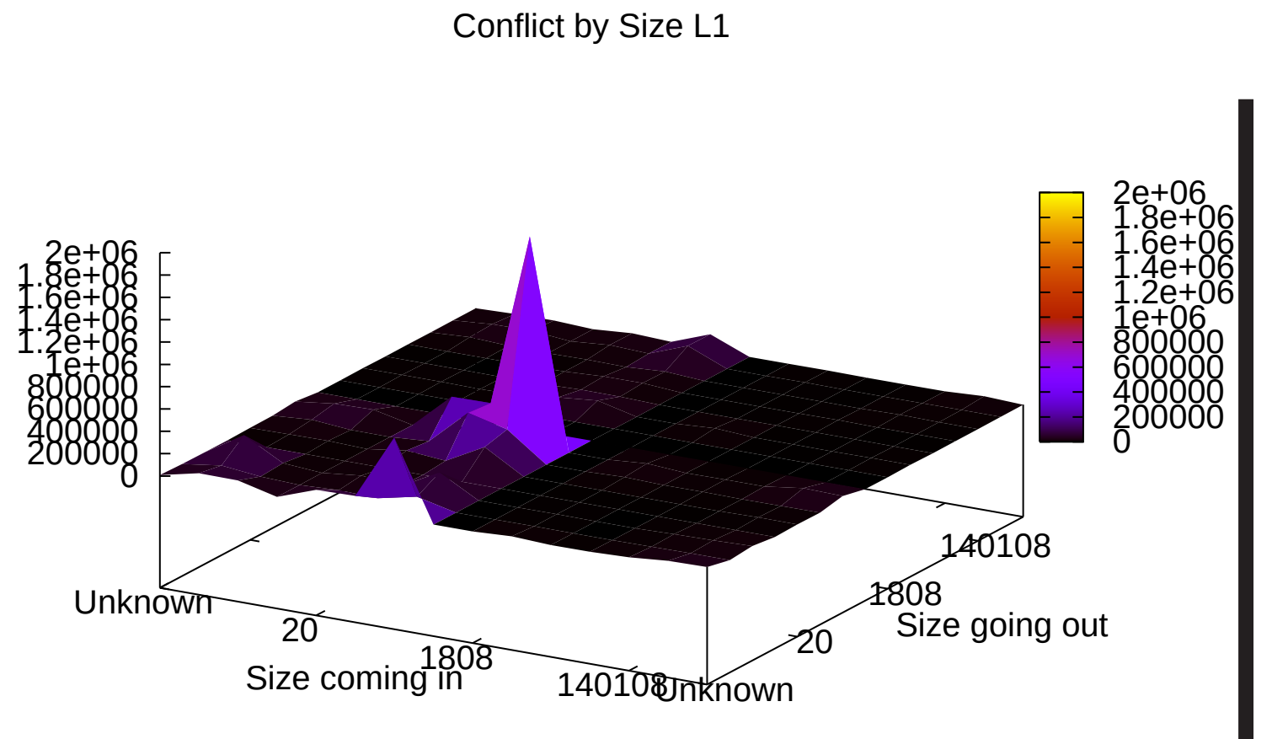


FIGURE 1: Conflicts by size in the L1 Cache. The biggest peak is that of 24-byte allocations (corresponding to an array of 3 doubles) kicking each other out of the cache.

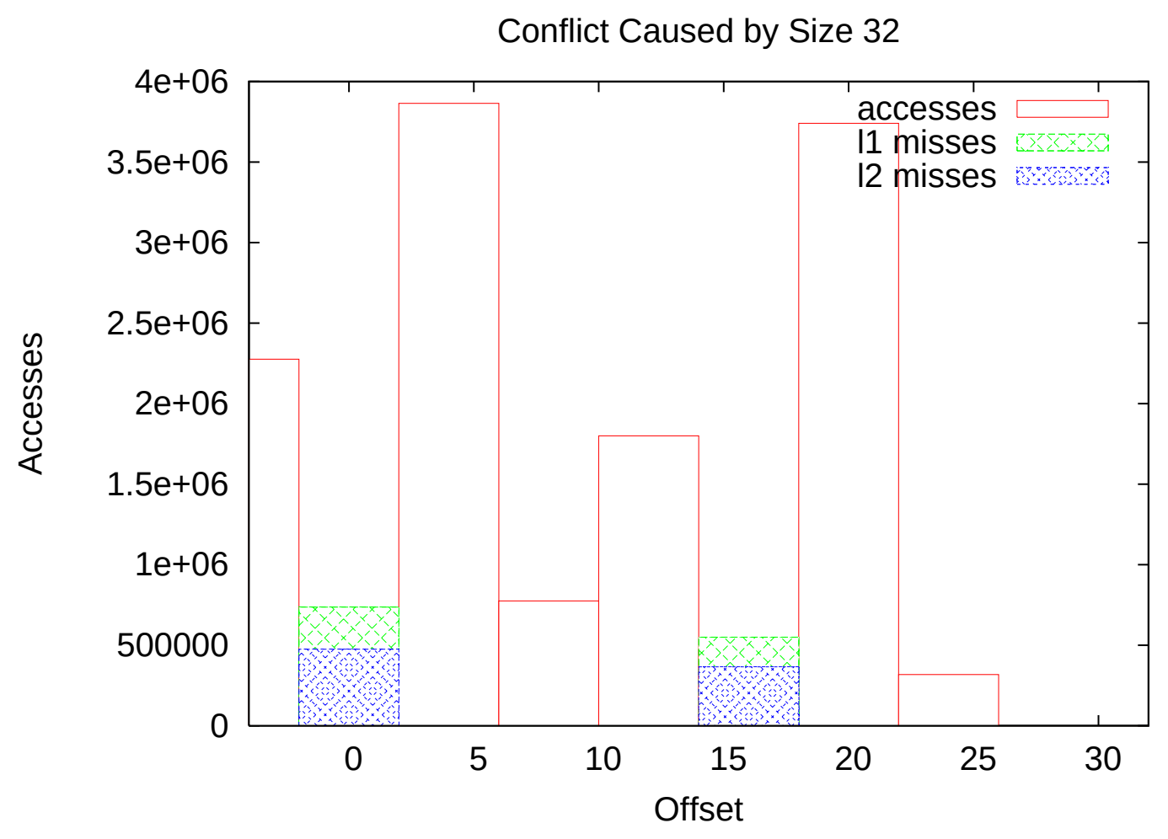


FIGURE 2: Accesses, both total and those causing misses, for offsets in the 24-byte allocations (plus 8 bytes of padding added by the system's *malloc()* routine).

smg2k from the ASCI Purple Benchmarks

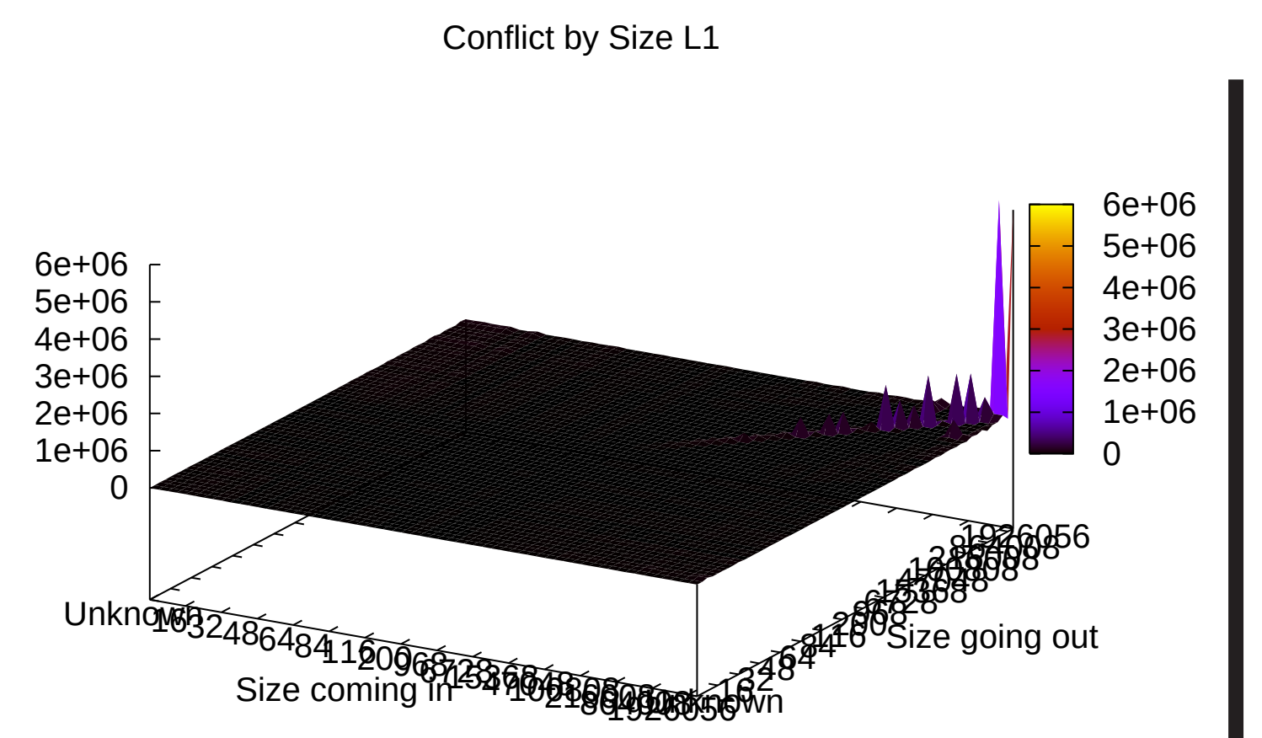


FIGURE 3: Conflicts by size in the L1 Cache. The biggest peaks here are from large allocations (bigger than the cache size) causing capacity misses.

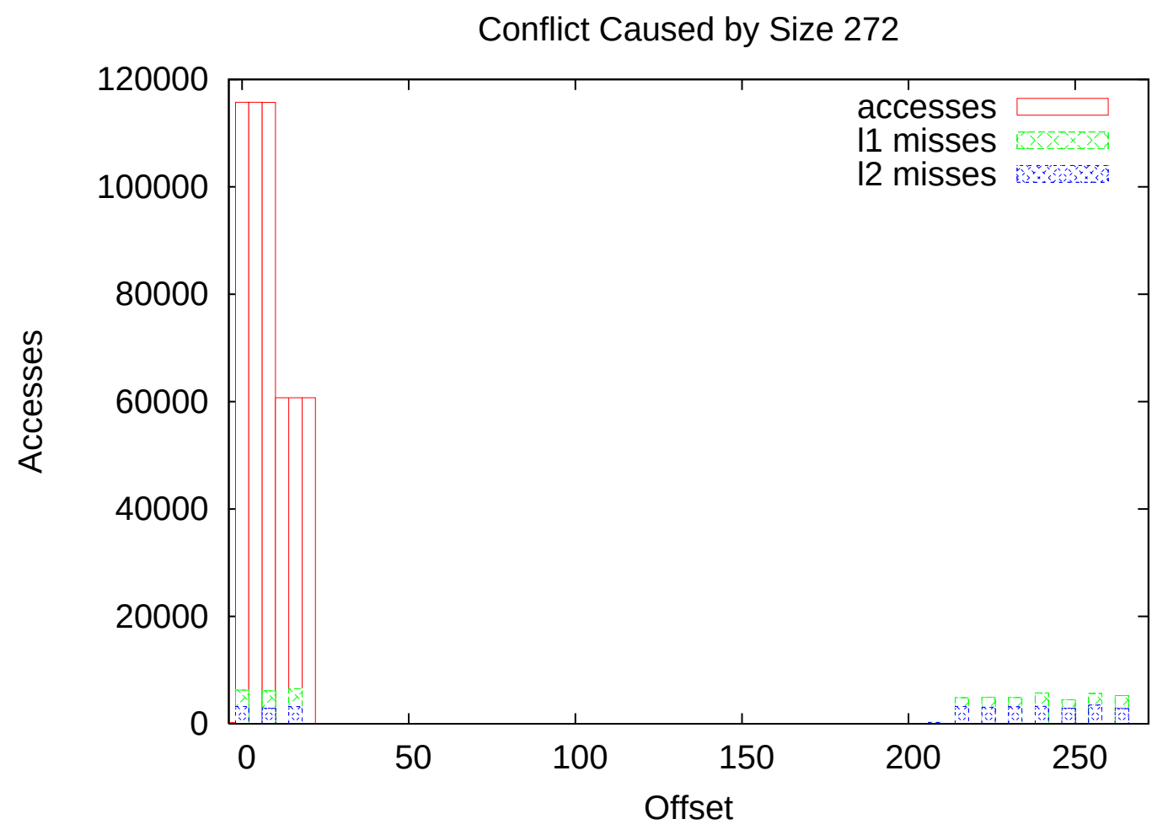
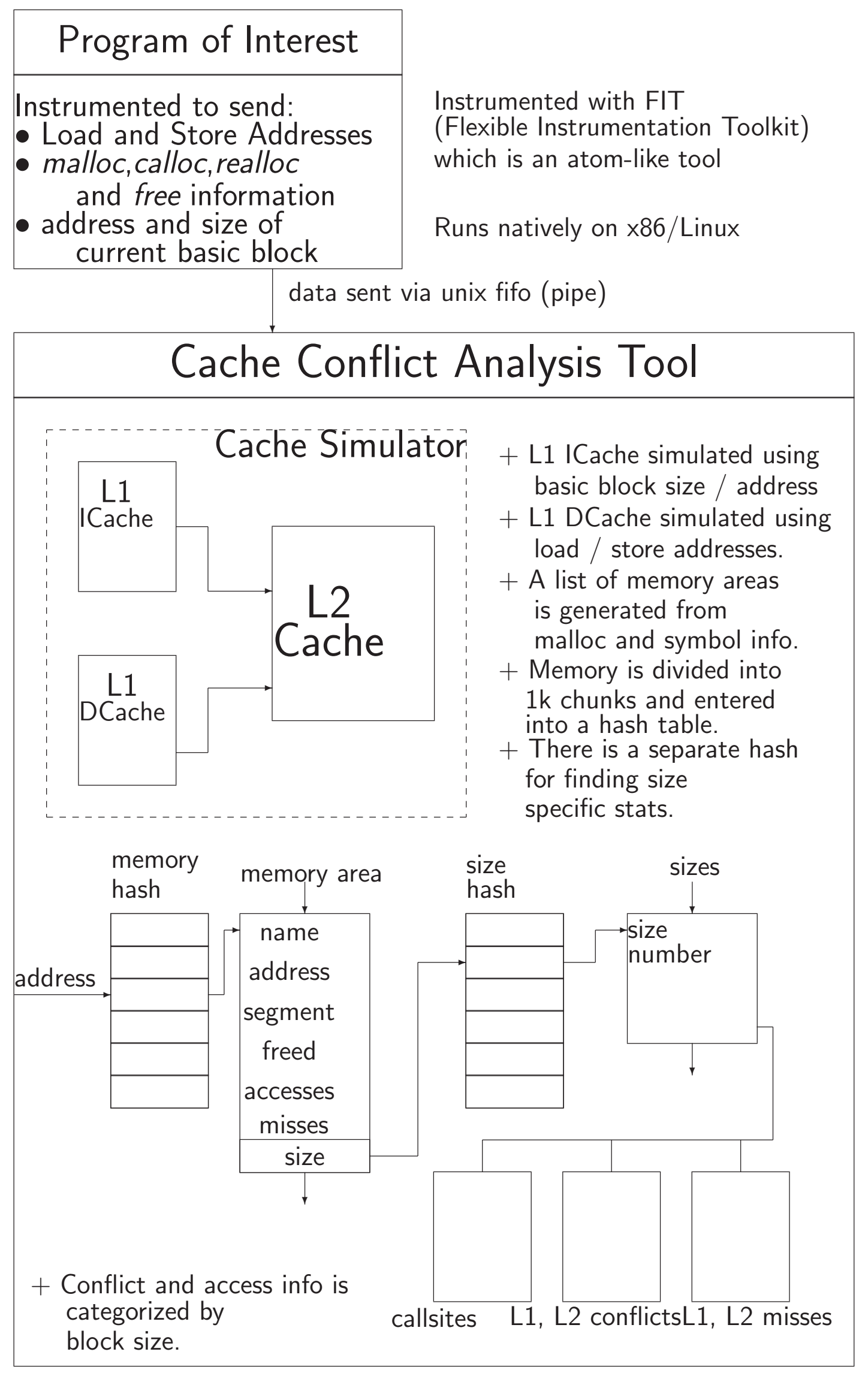


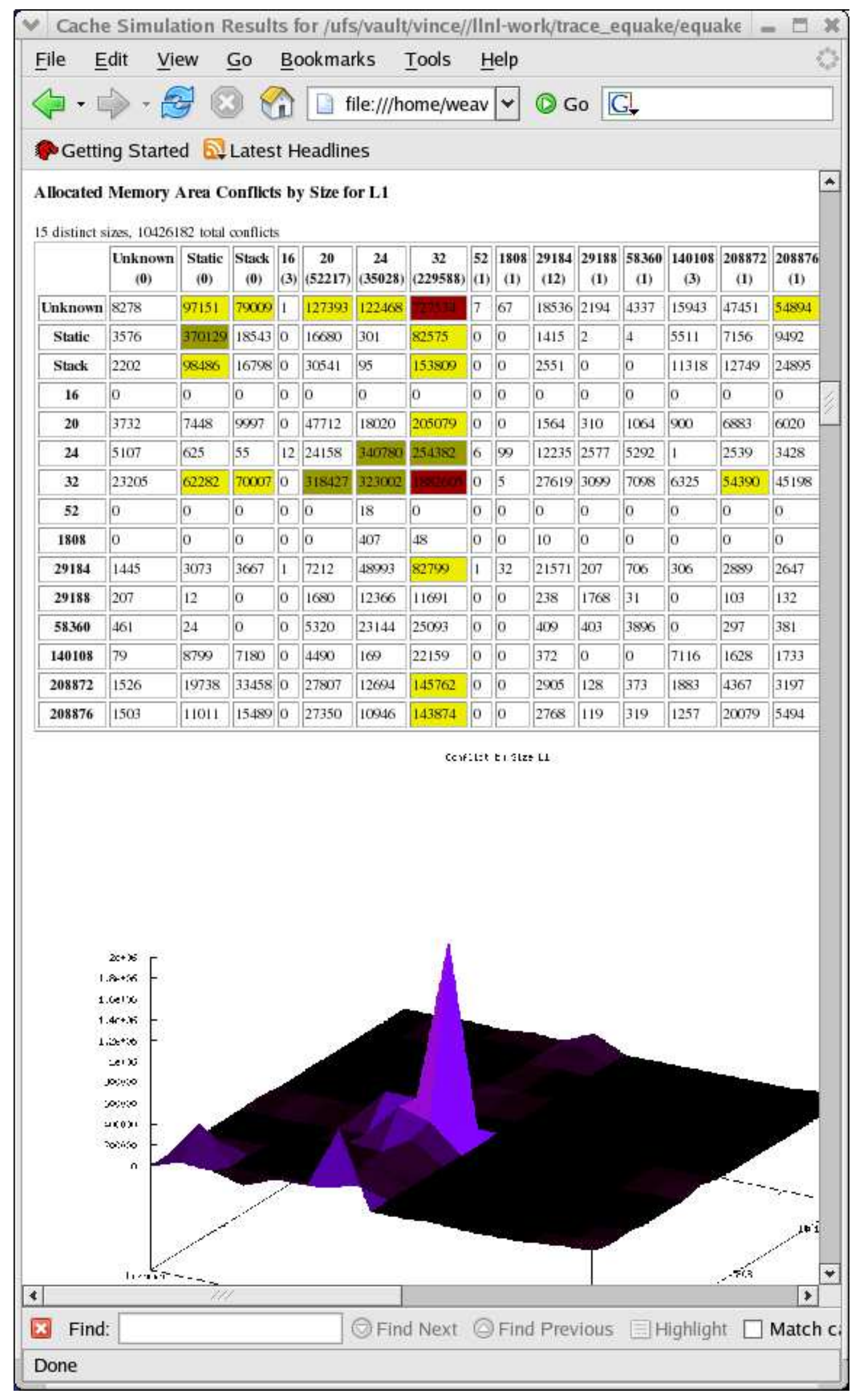
FIGURE 4: Accesses, both total and those causing misses, for offsets in the 272-byte allocations. The large gap of unused space in the middle might open opportunities for better cache placement.

Implementation



Program of Interest
Instrumented to send:
• Load and Store Addresses
• malloc,calloc,realloc and free information
• address and size of current basic block
Instrumented with FIT (Flexible Instrumentation Toolkit) which is an atom-like tool
Runs natively on x86/Linux
data sent via unix fifo (pipe)
Cache Conflict Analysis Tool
Cache Simulator
L1 ICache
L1 DCache
L2 Cache
+ L1 ICache simulated using basic block size / address
+ L1 DCache simulated using load / store addresses.
+ A list of memory areas is generated from malloc and symbol info.
+ Memory is divided into 1k chunks and entered into a hash table.
+ There is a separate hash for finding size specific stats.
memory hash
address
memory area
name
address
segment
freed
accesses
misses
size
size hash
sizes
size number
callsites
L1, L2 conflicts
L1, L2 misses
+ Conflict and access info is categorized by block size.

Memory access info is collected from the program on the fly, run through a cache simulator, and various statistics are recorded.



Allocated Memory Area Conflicts by Size for L1

15 distinct sizes, 10426182 total conflicts

	Unknown (0)	Static (0)	Stack (0)	16 (3)	20 (52217)	24 (35028)	32 (229588)	52 (1)	1808 (1)	29184 (12)	29188 (1)	58360 (1)	140108 (3)	208872 (1)	208876 (1)
Unknown	8278	97151	79009	1	127393	122468	722534	7	67	18536	2194	4337	15943	47451	54894
Static	3576	370129	18543	0	16680	301	82575	0	0	1415	2	4	5511	7156	9492
Stack	2202	98486	16798	0	30541	95	153809	0	0	2551	0	0	11318	12749	24895
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	3732	7448	9997	0	47712	18020	205079	0	0	1564	310	1064	900	6883	6020
24	5107	625	55	12	24158	340780	254382	6	99	12235	2577	5292	1	2539	3428
32	23205	62282	70007	0	318427	323002	1882609	0	5	27619	3099	7098	6325	54390	45198
52	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0
1808	0	0	0	0	0	407	48	0	0	10	0	0	0	0	0
29184	1445	3073	3667	1	7212	48993	82799	1	32	21571	207	706	306	2889	2647
29188	207	12	0	0	1680	12366	11691	0	0	238	1768	31	0	103	132
58360	461	24	0	0	5320	23144	25093	0	0	409	403	3896	0	297	381
140108	79	8799	7180	0	4490	169	22159	0	0	372	0	0	7116	1628	1733
208872	1526	19738	33458	0	27807	12694	145762	0	0	2905	128	373	1883	4367	3197
208876	1503	11011	15489	0	27350	10946	143874	0	0	2768	119	319	1257	20079	5494

Results are presented as HTML with graphics.

Future Work

- Implement automatic cache-conscious data placement:
  - + by hand through code organization
  - + automatically at runtime via customized memory management
  - + using hardware remapping mechanisms (such as IMPULSE)
- Enable instrumentation by atom or valgrind in addition to FIT.
- Instrument more benchmarks.

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48

UCRL-POST-214300