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 each access causes a miss.

```
Pseudo-code of a Conflict Miss
Allocate A[512] (allocated at offset 0)
Allocate B[512] (allocated at offset 4096)
Loop j=0...511
    j = j + 0.511
```

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

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.

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
```

```
smq2k from the ASCI Purple Benchmarks
```

Implementation

```
```

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.