Incorporating Instruction-Based Sampling into AMD CodeAnalyst

Paul Drongowski, Lei Yu, Frank Swehosky, Suravee Suthikulpanit, Robert Richter
Advanced Micro Devices, Inc.

Software developers need objective, accurate information about program behavior to improve application performance. Program profiles are useful aids in performance analysis and tuning. A profile is a histogram-like chart that associates execution time or hardware-level events with specific instruction addresses or lines of source code.

Event-based profiles show where hardware-level microarchitectural events (such as cache misses) occur. Frequently occurring cache misses, data translation lookaside buffer (DTLB) misses, or branch mispredictions indicate the presence of a potential performance issue. The nature of an event often indicates a remedial action. For example, if memory accesses are missing in the data cache or DTLB, then the data layout or memory access pattern can be changed to be more compatible with the underlying memory subsystem.

Profiles can be collected through performance counter sampling (PCS). A hardware performance counter is configured to generate an interrupt after the expiration of a user-defined sampling period. The sampling period is the number of instances of a specific, selected microarchitectural event that must occur before a sample is taken by the profiler. PCS determines where the program was executing at the time of the interrupt (i.e., the instruction pointer value) and tallies the sample into the profile. Ideally, the sampled instruction address is the address of the instruction that actually caused the selected event to occur. Unfortunately, the culprit's address is not directly available and the instruction pointer value on the interrupt stack is used instead. This address is the memory location where the program restarts execution after returning from the interrupt, not the address of the culprit instruction. Thus, the sampled address rarely indicates the instruction causing the event that triggered the interrupt.

Further, a delay, called "skid", occurs between the triggering event and the sampling interrupt. Skid on machines with out-of-order execution varies and is not a fixed delay. Skid distributes the samples in the neighborhood near the true culprit instruction. This produces an inaccurate distribution of samples, and events are often attributed to the wrong instructions or lines of source code.

Instruction-Based Sampling (IBS) is a hardware mechanism that improves the accuracy of profiles. IBS is supported by AMD Family 10h processors [1]. The processing pipeline of an AMD Family 10h processor is separated into two loosely coupled phases: A front-end phase that fetches AMD64 instruction bytes and a back-end phase that execute "ops" which issue from decoded AMD64 instructions. An op is an internal, fixed-width instruction which is executed by the pipeline stages in the execution phase. More than one op may issue from an instruction. Due to the decoupling, IBS samples fetches and ops separately, i.e., there are two independent sampling mechanisms. We will concentrate on IBS op sampling in this discussion.

Given a user-defined sampling period, IBS selects and tags an op to be monitored. The address of the parent instruction is retained when the op is tagged. Microarchitectural events caused by the tagged op are recorded during its execution. Stalls and other delays are also measured. When the tagged op retires, the event flags and stall cycles are read by an interrupt service routine and a sample (consisting of the instruction address, event flags, and stall cycles) is written to a profile data file. The events and stall cycles can be attributed precisely to the culprit instruction since its address was retained.

Figure 1 is a simple C language implementation of matrix multiplication. C language allocates two-dimensional arrays in row major order. Index variable `k` changes the fastest. Access to the elements of `matrix_a` is sequential while access to `matrix_b` touches one element of each row with every iteration. Each row of a matrix occupies 4,000 bytes. With an underlying page size of 4,096 bytes, almost every access to `matrix_b` causes a DTLB miss.

Table 1 shows the PCS and IBS profiles for DTLB misses in the innermost loop of the matrix multiplication program. The DTLB miss samples are distributed across the body of the inner loop in the PCS profile and some instructions are falsely identifed as culprits. The IBS profile precisely identifies the floating point

```
float matrix_a[1000][1000] ;
float matrix_b[1000][1000] ;
float matrix_r[1000][1000] ;

for (int i = 0 ; i < 1000 ; i++) {
    for (int j = 0 ; j < 1000 ; j++) {
        float sum = 0.0 ;
        for (int k = 0 ; k < 1000 ; k++) {
            sum = sum +
                matrix_a[i][k] * matrix_b[k][j] ;
        }
        matrix_r[i][j] = sum ;
    }
}
```

Figure 1: Matrix multiplication (program excerpt)

multiply at address 4011c2 as the performance culprit. This instruction loads an element from `matrix_b` and causes a substantial number of DTLB misses.

IBS reports a wide spectrum of information. Data cache miss latency, data operand (effective) address, and locality flags are returned for ops that load data from memory. Figure 2 shows the distribution of data cache miss latencies for the read operation in a linked list pointer-chasing loop. In each case, the length of the linked list was chosen to fit into a specific level in the memory hierarchy. The distributions show where and how often the load operation hit in the memory hierarchy. Distributions such as these can be combined with the data operand address and locality information to guide data layout on a NUMA platform.

AMD CodeAnalyst™Performance Analyzer [2] is a suite of tools for program performance analysis. It supports IBS-based profiling and displays finished profiles in tabular and graphical form. IBS event counts are summarized and displayed in terms of IBS derived events. An IBS derived event [1] is either an abstract event defined in terms of one or more hardware-level IBS event flags, or a stall/latency cycle count. A common set of fifty IBS derived events were defined. Derived events are aggregated and displayed in the same manner as PCS events. Through this approach, we were able to quickly implement IBS support within the existing CodeAnalyst infrastructure.

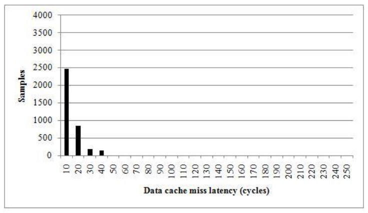
References

[1] Advanced Micro Devices, Inc., Software Optimization Guide for AMD Family 10h Processors, Publication 40546, May 2009.

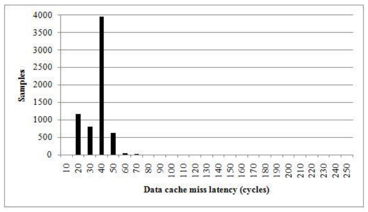
[2] P. J. Drongowski, An Introduction to analysis and optimization with AMD CodeAnalyst Performance Analyzer, `http://developer.amd.com`, September 2008.

Address	Instruction	PCS	IBS
401191	mov edx,[ebp-10h]	52	0
401194	add edx,1	111	0
401197	mov [ebp-10h],edx	0	0
40119a	cmp [ebp-10h],1000	0	0
4011a1	jnl 04011d1h	77	0
4011a3	mov eax,[ebp-04h]	0	0
4011a6	imul eax,eax,4000	0	0
4011ac	mov ecx,[ebp-10h]	31	0
4011af	imul ecx,ecx,4000	0	0
4011b5	mov edx,[ebp-10h]	308	0
4011b8	mov esi,[ebp-08h]	0	0
4011bb	fld dword [eax+edx*4+matrix_a]	0	18
4011c2	fmul dword [ecx+esi*4+matrix_b]	924	3337
4011c9	fadd dword [ebp-0ch]	979	0
4011cc	fstp dword [ebp-0ch]	484	0
4011cf	jmp 0401191h	153	0

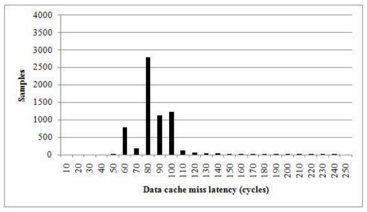
Table 1: DTLB miss events (PCS vs. IBS)



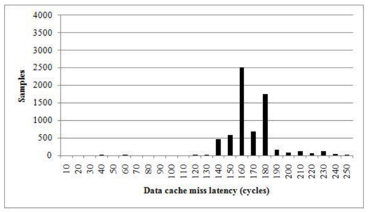
(a) L2 cache access



(b) L3 cache access



(c) Local DRAM access



(d) Remote DRAM access

Figure 2: IBS data cache miss latencies for linked list pointer chasing