DYNAMIC ADVANCE LOAD REMOVAL

Chad Rosier and Saurabh Sharma
{mcrosier,ssharma2}@ncsu.edu

April 11, 2005

1 Dynamic Binary Rewriting Implementation

1.1 Profiling

The first step to guiding the removal of a misspeculative advanced load is to identify such advance loads via dynamic instrumentation. In this project we will be using Intel's PIN tool. PIN enables a user to easily and dynamically instrument a binary at the instruction level. For each basic block we first identify any advance loads and their associated check instructions. A counter is inserted after the advanced load and at the entry to the recovery code. From this we can calculate the correctness of data speculation using the following function: $1 - \frac{times\_branched\_to\_recovery\_code}{times\_advance\_load\_executed}$. Once an advanced load is identified for removal this information is passed on to the *mutator process* via a shared buffer.

1.2 Mutator Process

Once all misspeculative advanced loads have been identified, information for modifying the target application is passed to the mutator process. This information includes the address of the advance load and associated check instruction as well as the address of the recovery code. After detaching PIN from the target application the mutator process attaches and begins to edit the process using the `ptrace()` system call. After the editing is complete the mutator process detaches and the optimized application continues to execute.

2 Current State

- *Verification of algorithm - C.Rosier* - To verify that our advance load removal algorithm is correct the Electron compiler was used to output an assembly file for a simple benchmark. The advance loads were then rewritten by hand to ensure the editing resulted in correct output.
- *Use of ptrace() - C.Rosier* - To test the functionality of the `ptrace()` system call a simple "Hello World" program was mutated. The target application prints "Hello World" and "Goodbye World" on a one second interval. `Objdump` was used to identify the address of the `br.call` instruction to the `printf(''Hello World'');` statement. The mutator application replaces this `br.call` with a `nop.b`. The resulting application then prints "Goodbye World" on a two second interval.
- *Identifying Bundle Template - S.Sharma and C. Rosier* - While PIN allows a user to easily instrument code at the instruction level it does need some assistance for identifying a bundle's template. Determining this is important because the check instruction (which is a memory operation) must be replaced with an unconditional branch (which is a branch operation). Therefore, in some cases the operations in the bundle must the reordered and the template redefined. To assist in the binary editing and to determine the bundle template the ELF library was used to read the .text region of the application into a buffer. Then PIN is used to calculate an index into the buffer which identifies the bundle's template.

- *Profiler - S.Sharma* - A profiler has been written to identify any advance load that results in a conflict in the Advance Load Address Table (ALAT).
- *Shared Buffer - C. Rosier* - To share information between the mutator process and PIN the `shm_open()` system call is used to create a POSIX shared memory object. This functionality has been implemented and tested independed of the dynamic rewriting system and will soon be integrated.

3 Future Work

- *Benchmarks - S.Sharma* - We are currently working with a set of kernels derived from a Texas Instrument DSP library. We hope to develop an additional set of synthetic benchmark to test the correctness of our dynamic rewriting system. At this time we are unable to compile the SPEC2K benchmark suite on an Itanium 2 processor. Due to the limited time remaining for the project we will most likely limit our benchmarks to simple kernel code and hand written assembly.
- *Transferring control between PIN and the Mutator Process - C.Rosier* - We are yet to address this issue. After PIN detaches we would like to be able to send a signal to the mutator process to begin editing the target application. Unfortunately, PIN does not return control after the detach function is called. We currently propose signaling the mutator process prior to the detach call and then have the mutator process sleep for a short period while PIN detaches. This presents a race condition and therefore will require additional consideration.
- *Increasing the amount of speculation - C.Rosier* - We are currently having a difficult time finding applications that include a large number of advance loads. GCC does not implement advance load speculation in any form. The Electron compiler does implement data speculation, but it is **very** cautious when inserting advance loads. One work indicated that the Electron compiler was 99.5% correct for the SPEC2K suite. We are investigating the use of the ORC compiler for increasing the amount of data speculation.

4 Targets

In the next two weeks we hope to have the basic system implemented. This includes:

- A PIN tool that collects profile information and determines when to undo an advance load. It should also pass this information to the mutator processes via the shared buffer.
- A Mutator process that forks the PIN process and sleeps until it is notified by PIN to begin editing the target application.

Our primary goal is to first implement a working system (not necessarily an efficient system). Then once we have verified the correctness of our ideas begin to whittle away the overheads associated with dynamic binary rewriting.