Address Trace Compression Through Loop Detection and Reduction

E.N. Elnozahy
IBM Austin Research Lab
11400 Burnet Rd.
Austin, TX 78758
512-823-6738
mootaz@us.ibm.com

An Extended Abstract

ABSTRACT

This paper introduces a new technique for compressing memory address traces. The technique relies on the simple observation that most programs spend their time executing loops, and therefore the trace will follow the structures of such loops. We adapt classic control flow analysis to detect the loops within an address trace, then analyze them to identify constant and loop-varying memory references. These references are efficiently encoded to reduce the size of the trace, often resulting in an order of magnitude reduction in size compared to the most compact trace format known to date.

Keywords

Compression, control flow analysis, address traces, traces.

1. INTRODUCTION

The use of memory address traces is an established technique for simulation-based studies of computer systems [10]. The focus in this paper is on lossless compression for reducing the size of an existing trace after it has been gathered. Lossless compression preserves the information in a trace and introduces no errors of its own during simulation [4][5][9][11]. This is in contrast with techniques that use sampling or exploit the nature of the cache hierarchy to reduce the trace size [1].

There are several motivating factors for reducing the size of an address trace:

- Processor speeds continue to increase as predicted, resulting in a proportional increase in the address trace size within a given period of real time.
- Long traces are needed to give good estimates for the system performance.
- Disk speeds are not improving. A performance study often becomes I/O-bound during the simulation phase.

Furthermore, memory address traces have low information entropy with ample opportunity for compression [2]. It may be argued that storage costs and capacities are decreasing at rates comparable or exceeding those of the processor speeds. While this may be true, managing terabytes of trace volumes is not likely to be a simple undertaking.

This paper introduces a new technique, which relies on the simple observation that most programs spend the bulk of their processing time executing in loops. Thus, if these loops could be detected within the trace, then one could eliminate the data address references that are constant or change by a constant value between consecutive loop iterations.

2. Loop Detection and Reduction (LD&R)

Consider the following code fragment:

```
for(j = 0; j < n; j++)
        a[j] = b[j] * c[j];
```

The assembly code for this fragment typically loads the addresses of the vectors *a*, *b* and *c* in some processor registers, then uses a register to index through the loop. The corresponding trace records show repeated execution of the same basic block with the data addresses differing by a constant offset between consecutive iterations. By detecting such a loop in the trace file, one can replace all the occurrences with some encoding like:

```
loop: starting addresses of a, b, and c
offsets +4, +4, +4
n times
```

This is an extreme case and shows the gist of the technique used in this paper. The trace reduction occurs in two steps. In the first, we adapt classic control flow analysis techniques to detect loops within a trace. The second step identifies three types of address references within each loop detected in the first step:

- Constant address references. These do not change from one loop iteration to the next. Constant address references occur for example when a stack variable is repeatedly read into a register.
- Loop varying address references. These references change by a constant offset between consecutive loop iterations. The addresses generated for a[j], b[j] and c[j] in the example above are loop varying references.

- Chaotic address references. These references change between consecutive loop iterations without following any pattern.

Constant and loop varying address references are encoded once in the loop body, while chaotic address references have to be included in the sequence in which they appear within the trace. Depending on the trace, and how uniform the loops are, loop detection and reduction can yield moderate to substantial savings.

Some complex loops nevertheless may contain jumps to functions, or may contain complex intra-loop branches that will change the structure of the trace records from one loop iteration to the next. In such situations, it becomes very difficult to detect patterns, and the effectiveness of the technique is greatly hampered. As an example, consider the following code fragment.

```
For(j = 0; j < n; j++)
        a[j] = d[bsearch(a[j])]
```

The addresses generated for the vector *d* depend on the results of executing a complex binary search function. In situations like this, it is possible to identify the references to the vector *a* as loop varying references, but otherwise the references to vector *d* will be chaotic address references. Nonetheless, even with such restrictions, the simple cases seem to manifest themselves often enough to generate very good compression.

3. Implementation and Performance

We have implemented this technique in the context of the MTRACE trace tool and its associated format [8]. MTRACE is a sophisticated tool that captures traces of multi-programmed workloads including the effects of context switching, operating system, and shared library code. It uses a combination of hardware assist [8] and software instrumentation similar to other techniques [3][6][7].

Most importantly, the MTRACE format does not store the instruction address references. Instead, it stores in the trace the address of the leading instruction in each basic block followed by the data address references within that basic block. This technique results in considerable compression, since non load-store instructions do not contribute to the size of the trace unless they occur at the beginnings of basic blocks. Interestingly, the idea of storing the trace in the form of basic blocks was later developed independently by Fox and Grün, although the trace formats differ considerably [4]. The actual instruction references can be constructed using the leading instructions within each basic block along with an auxiliary file describing the basic blocks within the workload being traced. To accommodate context switches, the tool records in the trace the instruction address at which a basic block is interrupted or resumed, such that the interleaved trace could be reconstructed during the simulation phase.

We conducted an experimental study using seven traces from different benchmarks, ranging from commercial transaction processing systems to scientific loads. The results are very encouraging and show that LD&R yields compression ratios that are about an order of magnitude better than the efficient MTRACE format.

4. ACKNOWLEDGMENTS

Tom Keller and Anne Marie Maynard provided the traces used in this experiment. A pending U.S. patent covers the technique introduced in this paper.

5. REFERENCES

[1] Agrawal, A., and Huffman, M. Blocking: Exploiting space locality for trace compaction. Proceedings of the 1990 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1990.

[2] Becker, J. and Park, A. An analysis of the information content of address and data reference streams. In Proceedings of the 1993 SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, May 1993.

[3] Eggers, S., Keppel, D., Koldinger, E., and Levy, H. Techniques for efficient inline tracing on a shared-memory multiprocessor. Proceedings of the 1990 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1990.

[4] Fox, A., and Grün, T. Compression of address traces for cache simulations. Proceedings of the International Data Compression Conference, February 1997.

[5] Johnson, E.E., and Ha, J. PDATS: Lossless address trace compression for reducing file size and access time. Proceedings of the International Phoenix Conference on Computers and Communications, March 1994.

[6] Larus, J.R. Efficient program tracing. IEEE Computer, May 1993.

[7] Larus, J.R. Abstract execution: A technique for efficiently tracing programs. Software: Practice and Experience, December 1998.

[8] Levine, F., Twichell, B., and Welborn, E. Hardware mechanism for instruction/data address tracing. U.S. Patent No. US5446876, August 1995.

[9] Samples, A.D. Mache: No-loss trace compaction. Proceedings of the 1989 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1989.

[10] Uhlig, R., and Mudge, T. Trace-driven memory simulation. A survey. ACM Computing Surveys, Vol. 29, No. 2, June 1997.

[11] Ziv, L., and Lempel, A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, August 1977.