Visualizing Reference Traces

How do we capture program behavior?

A.N.M. Imroz Choudhury[1], Steven G. Parker[1,2]

[1] *Scientific Computing and Imaging Institute, University of Utah*
[2] *NVIDIA*

Introduction

Visualization of program behavior is an important approach to helping construct high-performance software, as needed in fields such as scientific computing, where large problem sizes and scarce supercomputer time demand the highest possible performance from client applications. Existing performance analysis tools, such as code profilers, can indicate the places in a program that lead to runtime slowdowns, but they often offer little or no help in analyzing the underlying causes of such slowdowns. Our approach has been to identify one contributor to performance slowdowns—namely, the memory subsystem—and to develop methods and tools for (1) extracting viable data sources related to memory and (2) transforming and visualizing these sources in order to help develop insight about performance problems that may arise from poor use of the memory subsystem.

The centerpiece of this effort is a visualization system, the Memory Trace Visualizer (MTV)[1], with a modular design inspired by the SCIRun system[2]. So far, this system handles memory reference traces generated by CPU programs by casting the references in terms of pre-defined arrays in memory containing data the user wishes to focus on, allowing for the visualization of access patterns as they appear in memory itself.

Currently, we are investigating an extension of this idea, namely, using application-specific information to re-frame the same reference trace data in a new way. An example showing a first attempt at this approach on a particle simulation appears below. We also have plans to extend this system to handle GPU programs. Finally, because such traces are very large, yet contain relatively little information (due to the spatiotemporal locality that is the hallmark of memory references in typical programs) we are also looking into methods for capturing such traces in a sparse manner, without overly degrading their quality (i.e., accuracy), with the goal of both fast and accurate trace collection.

```
MTR::Record(Read, 0x7fffbb01be88)
MTR::Record(Write, 0xb052d8)
MTR::Record(Read, 0x7fffbb01be90)
MTR::Record(Read, 0x7fffbb01be98)
MTR::Record(Write, 0x7fffbb01be98)
MTR::Record(Read, 0x7f78cca24b68)
MTR::Record(Write, 0xb05350)
MTR::Record(Write, 0xb05358)
MTR::Record(Read, 0x7f78cca23a50)
MTR::Record(Write, 0x7fffbb01be90)
MTR::Record(Read, 0x7f78cca23570)
MTR::Record(Write, 0xb052e0)
MTR::Record(Write, 0xb05348)
MTR::Record(Write, 0xb0534c)
MTR::Record(Read, 0x7f78cc7b2c9c)
MTR::Record(Write, 0xb052f0)
MTR::Record(Write, 0xb052e8)
MTR::Record(Write, 0xb05300)
MTR::Record(Write, 0xb052f8)
MTR::Record(Write, 0xb05308)
MTR::Record(Write, 0xb05310)
MTR::Record(Write, 0xb05318)
MTR::Record(Write, 0xb05320)
MTR::Record(Write, 0xb05328)
```

Reference Traces

It can be arranged for a running program to record its full interaction with memory. The result is a list of Read/Write codes associated with memory addresses: a large amount of data that is difficult to understand by direct inspection. Though it represents the full and complete interaction with the memory subsystem, it lacks both **framing** and **context**. The standard way to reduce the amount of data to a manageable form is to run the trace through a cache simulator, which then reports aggregate numbers of hits and misses, for a summarizing, monolithic view of program performance.

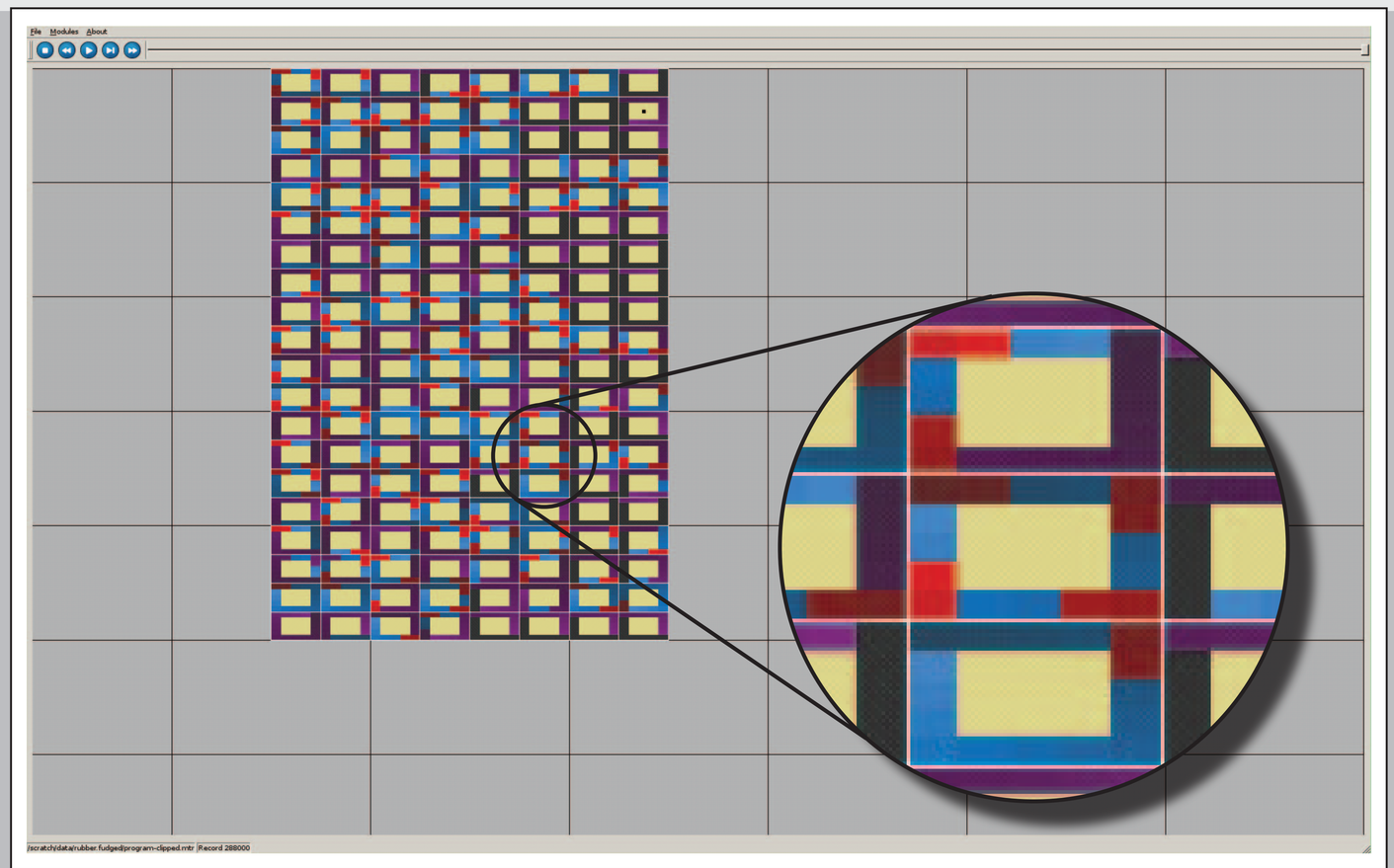
Array View

The first MTV system provides the *array view*, in which the user selects some regions of memory to watch; the program registers to disk the base and limit addresses, and the size of the datatype, for each of these regions. This view performs cache simulation with all the trace records, and displays the reads and writes that occur within the user-selected regions, along with the "cache result"—hit or miss—encoded as blue-to-red lights under each region's data representation.

This example shows a trace for a particle simulation code; currently, the particles' velocity gradient values are being updated, as evidenced by the reads followed by writes to that region. This view gives some temporal cues to help the user understand the recent history of the trace: note the fading trails left by older references as they age. This view provides a good way to **frame** the reference trace, but the only context provided is very general, in terms of the individual effects the references have in a simulated cache.

Physical View

Our very current work involves using the output of the program in the visualization of its memory behavior. This is the same reference trace data once more, but displayed in terms of the particles being physically simulated. Here we see an elastic bar hanging from a ceiling, discretized into several material particles, and loaded by gravity. The references are still simulated through a cache, and now each particle shows a history of the cache effects its accesses have caused, flowing around the edge of its glyph, with brighter colors indicating more recent accesses. A marker also shows which particle was the last to be accessed. By correlating events in the trace with the end of a particular simulation timestep, MTV is able to load new simulation output data from disk at the appropriate time, providing a new application-specific **context** for the reference trace data. This approach essentially attempts to marry together traditional scientific visualization with a new source of data. By examining the resulting patterns, our hope is that the memory performance of data structures and computation approaches can be "debugged".



[1] A. I. Choudhury, K. C. Potter, and S. G. Parker. Interactive visualization for memory reference traces. Computer Graphics Forum, 27(3):815–822, May 2008.
[2] D. Weinstein, S. Parker, J. Simpson, K. Zimmerman, and G. Jones. Visualization in the scirun problem-solving environment. In C. Hansen and C. Johnson, editors, The Visualization Handbook, pages 615–632. Elsevier, 2005.