# CSC 548 Project: Memory Trace Compression using Extended PRSDs


## Abstract

Analyzing the memory traces of multithreaded programs is a cumbersome and expensive process due to large trace size, program complexity and long running times. Though many binary instrumentation tools generate memory traces, they either gather statistical information with loss of details or generate large trace files that are difficult to handle. The project aims to develop a tool that provides near constant size memory traces irrespective of the number of threads involved while preserving the memory access details along with order in which memory accesses are done. The proposed scheme also groups the memory accesses with in a loop to a single entity called Extended Power Regular Section Descriptor (EPRSD) which is an enhancement over Power Regular Section Descriptor (PRSD) concept. The compression scheme is based on repetitive memory access pattern within a thread and also across multiple threads.


## Description

Memory traces are generated using Intel's binary instrumentation tool called PIN. Generated trace consists of instruction-type (load/store), address, program counter, stack signature and thread identifier. Memory trace generator also performs stackwalk to compute signature and filters ir relevant instructions. Thus generated Memory traces of load/store instructions are fed into the compression tool. Memory traces are compressed based on two criteria. One is intra-task compression based on starting address and stride within a single thread. Other is inter-task compression where starting address varies depending on the thread identifier but length and stride are identical across multiple threads.

Intra-task compression is done by extending the concept of representing single loops using 'Regular Section Descriptors' (RSDs) to express load and store instructions nested in loops, in constant-size. An RSD is represented as <*start_address, stride, length, LD/ST*>, where length indicates the loop count, stride indicates the distance between the memory accesses of each iteration. EPRSD represents the loop dependencies by grouping the individual RSDs or EPRSDs together. An EPRSD is represented as <*start_value, thread_id_based_offset, stride, length, RSD1, RSD2, EPRSD1*> - 'start_value' and 'thread_id_based_offset' are only used for inter-task compression. Inter-task compression is done by identifying repetitive patterns of RSDs or EPRSDs across multiple threads. Another level of EPRSD is used to represent the task level dependency where the length, stride and base address are shown as a function of thread id.

## Milestones

| Task | Status | Estimated Completion Date |
|---|---|---|
| Generation of memory trace | Completed | |
| Generating RSDs | Completed | |
| Detecting loops and Generating EPRSDs for individual threads | In Progress | Nov 1 |
| Merging EPRSDs across threads | Pending | Nov 8 |
| Testing of Memory trace compression tool | Pending | Nov 15 |
| Final Report | Pending | Nov 22 |

## Project Web page

http://www4.ncsu.edu/~sbudanu/CSC548_Project/

## References


[1] P. Ratn, F. Mueller, Bronis R. de Supinski, Michael Noeth and M.Schulz, ScalaTrace: Scalable Compression and Replay of Communication Traces for High Performance Computing , Journal of Parallel and Distributed Computing, accepted Sep 2008, pages 1-14.

[2] Prasun Ratn, M.S. Thesis, Preserving Time in Large-Scale Communication Traces , North Carolina State University, Aug 2008.

[3] J. Marathe F. Mueller, T. Mohan, S. McKee, B. de Supinski, A.Yoo, METRIC: Memory Tracing via Dynamic Binary Rewriting to Identify Cache Inefficiencies, ACM Transactions on Programming Languages, Vol. 29, No. 2, Apr 2007, pages 1-36.

[4] M. Noeth and F. Mueller and M. Schulz and B. de Supinski Scalable Compression and Replay of Communication Traces in Massively Parallel Environments, P=ac2 Conference, IBM T.J. Watson, Oct 2006.

[5] J. Marathe, F. Mueller, T. Mohan, B. R. de Supinski, S. A. McKee and A. Yoo METRIC: Tracking Down Inefficiencies in the Memory Hierarchy via Binary Rewriting, International Symposium on Code Generation and Optimization, Mar 2003, pages 289-300.

[6] Pin binary instrumentation tool. http://www.pintool.org