Artifact Description: Hardware Transactional Persistent Memory

Ellis R. Giles
Rice University
Ph.D. Candidate
erg@rice.edu

Kshitij Doshi
Intel Corporation
Industry Advisor
kshitij.a.doshi@intel.com

Peter Varman
Rice University
University Advisor
pjv@rice.edu

APPENDIX A
ARTIFACT DESCRIPTION APPENDIX: [HARDWARE TRANSACTIONAL PERSISTENT MEMORY]

A. Abstract

This description contains the information needed to launch some experiments of the "Hardware Transactional Persistent Memory" poster. We describe how to compile and run the cc-HTM solutions, simulator for hardware, STAMP benchmarks, and experiments. A system containing many CPU cores and Hardware Transactional Memory HTM support with Intel TSX is required.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** HTPM memory controller using a simulator supplied, cc-HTM via lock-free queues and persistent storage techniques using aliasing similar to SoftWrAP.
- **Program:** C binary, C library, C++ system, STAMP benchmarks
- **Compilation:** gcc
- **Data set:** Supplied STAMP benchmark data and random generated data
- **Run-time environment:** HTM based system
- **Hardware:** Intel based multi-core CPU equipped with version 4 or TSX-I support
- **Execution:** Execution of the supplied test scripts
- **Output:** Tab separated text files
- **Experiment workflow:** Download and install Intel Pin. Download and install STAMP benchmark suite. Download and install HTPM. HTPM distribution contains McSimA+ modifications, STAMP benchmark modifications, TL2 modifications, and core software implementation based on cc-HTM and SoftWrAP. Compile set of solutions, run example test suite.
- **Experiment customization:** The included benchmarks for micro-benchmarks may be customized via size, number of threads, amount of work, transaction set, etc. The test scripts sweep much of the system space. The STAMP benchmark suite may be customized per documentations.
- **Publicly available?:** Yes via request. Pending approval for GNU Public License open source.

2) How delivered: HTPM may be cloned from BitBucket via request.

```
git clone https://egiles@bitbucket.org/egiles/htpm.git
```

3) Hardware dependencies: Multi-core Intel TSX-I supported processor. Note that the processor must support HTM (Hardware Transactional Memory) transactions using the Restricted Transactional Memory (RTM) implementation at this time.

Ideally, a system with Persistent Memory should be used. For evaluation purposes, DRAM may be substituted for PM and access speeds may be scaled.

Included tests in the downloaded benchmarks can simulate the reduction in write speeds.

4) Software dependencies: Linux, gcc, and g++ are required.

Specifically:

```
Thread model: posix
gcc version 7.3.1 20180303
```

Additionally, Intel Pin and STAMP Benchmarks are required. Pin is a Binary Instrumentation Tool. Pin for Linux is required for the memory controller optional support.

Download PIN from:

```
https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads
```

The STAMP or The Stanford Transactional Applications for Multi-Processing; a benchmark suite for transactional memory research is required.

Download STAMP from:

```
git clone https://github.com/kozyraki/stamp.git
```

HTPM contains a modified version of TL2 or Transactional Locking 2. We modified TL2-x86 implementation to add support for Persistent Memory.

The original TL2-x86 source may be obtained from:

```
git clone https://github.com/ccaominh/tl2-x86.git
```

PMDK. Persistent Memory Development Kit.

For full PM development and evaluation on a PM enabled system, download and install PMDK.

```
https://pmem.io/pmdk/
```

5) Datasets: The included distribution for STAMP and HTPM have all the included datasets. Random data is generated for micro-benchmarks including inserts into Red Black trees and arrays. Random access to large memory arrays is also performed. Random data is seeded to generate repeatable results. STAMP benchmarks have all data sets included.

C. Installation

```
[user@system htpm]$ cd src
% Build TL2-x86 Modifications
[user@system src]$ cd tl2-x86-mp/
[user@system tl2-x86-mp]$ make
...
[user@system tl2-x86-mp]$ cd ..
% Build HTPM libraries and micro-benchmarks
[user@system stamp-0.9.10]$ cd ..
[user@system src]$ make
...
% Build STAMP Benchmarks Suite with HTM and TL2-x86 Linked Binaries
[user@system src]$ cd stamp-0.9.10/
[user@system stamp-0.9.10]$ make
...
```

D. Experiment workflow

Simply execute the exp script which runs 20 experiments numbered with exp01 and up.

The exp script generates an output file for each experiment in a tab delimited format suitable for import into any graphing software. The output files are named with the dot out extension, e.g. exp01.out.

```
[user@system src]$ ./exp
```

E. Evaluation and expected result

All results for each experiment will be in the exp01.out file. Outputs in the space separated format similar to:

```
threads htmonly hw hwstrict software ptl2
1 9.29E+05 1.52E+05 1.53E+05 1.15E+05 2.77E+04
2 1.17E+06 1.94E+05 2.00E+05 1.55E+05 4.29E+04
3 1.69E+06 2.93E+05 2.68E+05 2.34E+05 5.61E+04
4 2.09E+06 3.96E+05 3.36E+05 2.76E+05 7.22E+04
5 2.37E+06 4.91E+05 4.12E+05 3.31E+05 7.24E+04
...
```

F. Experiment customization

All experiments may be customized for size, cache-size, HTM, number of threads, etc.

HTM implementations may vary.

The fall-back lock for HTM may be tuned appropriately for optimized performance.

HTM tuning is provided in the tune directory. Simply run the tune scripts to optimize the fall-back lock policy, e.g. after how many failed HTM transactions should a global fallback lock be taken.

The $tune4$ script will optimize the HTM implementation fallback path.

```
[user@system src]$ cd tune
[user@system tune]$ ./tune4
```

G. Notes

For evaluation, we employed Intel(R) Xeon(R) E5-2697 v4 series processors, 18 cores per processor, running at 2.30 GHz, with Red Hat Enterprise Linux 7. We used *numactl* to restrict all threads to a single processor (socket) for repeatable results and reducing interference from other background activities during measurements. HTM transactions were implemented with Intel Transactional Synchronization Extensions (TSX) using a global fallback lock. A set of runtime environment variables allowed for the easy configuration of Alias Table sizes, persistence methods such as cc-HTM or no persistence, and maximum allowable lag. We built our software using g++ 4.8.2. Each measurement reflects an average over twenty repeats with small variation among the repeats.

Using micro-benchmarks and SSCA2 and *vacation*, from the STAMP benchmark suite, we compared the following methods:

- **HTM Only:** Hardware Transactional Memory with Intel TSX, without any logging or persistence. This method provides a baseline for transaction performance in cache memory without any persistence guarantees. If a power failure occurs after a transaction, writes to memory locations may be left in the cache, or written back to memory in an out-of-order subset of the transactional updates.
- **Software:**HTM for Persistent Memory. This is our full implementation using the Persistence Management Thread, Queue, Logging, and Alias Table.
- **WrAP:** Our memory controller supported method. The volatile delay buffer in the controller is assumed to be able to keep up with back pressure from the cache, as shown in Section 3. We therefore perform all other aspects of the protocol such as logging, reading timestamps, HTM and fall-back locking, etc.
- **WrAP-Strict:** Same as above, but we implement the software strict durability method as described in Section 4. Threads wait until all prior-open transactions have closed before proceeding.
- **PTL-Eager:** (Persistent Transactional Locking). In this method, we added persistence to Transactional Locking (TL-Eager) by persisting the undo log at the time that a TL transaction performs its sequence of writes. The undo-log entries are written with write-through stores and SFENCEs, and once the transaction commits and the new data values are flushed into PM, the undo-log entries are removed.