Summer Research Report
Benchmarks on the VSoC Simulator

August, 2013

Advised by: Tali Moreshet

Callen Rain & Peng Zhao

Contents

1 VSoC

1.1 Installation

1.2 Structure

1.2.1 TCDM

1.3 Benchmark Compilation

1.4 Benchmark Execution

1.5 Increase TCDM Size

In VSoC, the default TCDM size is defined to be 0x40,000 Bytes (256 KB). Unfortunately, there are some particularly "hungary" applications that require more memory to run. If we run these applications with the default TCDM size, VSoC will run out of shared memory, and the program will crash.

To solve this problem, we will need to increase the size of TCDM. Here is the procedure:

1). Open vsoc-beta/src/core/config.h , and change CL_TCDM_SIZE (around line 60) to the desired value.

- Skiplist on a 8-core system needs 0x00080000 Bytes (512KB) to run.
- Skiplist on a 16-core system needs 0x000b0000 Bytes (768KB).
- RedBlack on a 16-core system needs 0x00080000 Bytes (512KB).
- Full version of Patricia needs 0x00400000 Bytes (4MB).

See Sections 3.5, 3.6, and 3.7 for more details.

2). Open vsoc-beta/apps/support/simulator/vsoc.ld

Change the value of STACK_TOP (line 6) to 0x08000000 + CL_TCDM_SIZE - 0x1004.

Change the value of ORIGIN (line 10) to 0x08000000 + CL_TCDM_SIZE - 0x1000.

For example, if CL_TCDM_SIZE is set to be 0x00080000, then STACK_TOP should be 0x0807effc, and ORIGIN should be 0x0807f000.

The variables STACK_TOP and ORIGIN define the start of the stack. They need to change together with TCDM size to make sure that the stack is always located at the end of shared memory. See Section 2.1 for more details.

3). Go to the vsoc-beta directory, and run the following command:

student@ubuntu: /vsoc-beta$ source SOURCEME

4). Go to the vosc-beta/scripts directory, and run:

student@ubuntu: /vsoc-beta/scripts$ vsoc_clean

student@ubuntu: /vsoc-beta/scripts$ vsoc_build -a

5). Clean and recompile the benchmark (to make changes in vsoc.ld effective), and run it on the simulator.

2 Application Support

2.1 Shared Memory Allocation

2.2 Initialization Flags

2.3 Global Pointers

2.4 Shared Memory Allocation

2.5 Barriers

2.6 Locks

3 Benchmarks

3.1 Hello World

This benchmark simply prints out a "Hello World" statement to the screen. It was included in the version of VSoC that we downloaded and consisted of one line of code that printed a line of code to the screen.

In the summer of 2013, Peng and Callen (Swarthmore College) made a few changes to the benchmark to test two of the application support functions that we made.

3.2 Count

Count is a very basic benchmark that was obtained from Iris Bahar's Low Power VLSI System Design class at Brown University. The benchmark is included as an example in a shared memory synchronization lab for the class. It originally ran on the MPARM hardware simulator. Students are shown the unparallelized and the parallelized versions of the benchmark and then asked to perform similar modifications to another application.

In the summer of 2013, we decided to port the benchmark from this lab application on MPARM to the VSoC system we were working on. It was useful as a simple test of the locks implemented in VSoC (it turned out that these locks were not completely functional).

The functionality of the benchmark is simple. The cores take turns acquiring a lock for a shared counter. They increase it by one and release the lock. The final value of the counter is correct if none of the cores have interfered with each other's atomic increases.

3.3 Matrix Multiplication

Along with hello world, matrix mult is a benchmark that came with the VSoC system from University of Bologna.

The operation of the benchmark is the multiplication of two arrays. These arrays are defined in matrix.h. Originally they were generated with MatLab and are sized 16x16.

Matrix multiplication isn't terribly useful for synchronization tests because the application can simply break up the matrix into pieces and assign each core to have their own chunk. They can store a local copy of the arrays and only operate on the part they are assigned to. Since this arrangement doesn't use locks, the matrix multiplication benchmark is more useful as a test of the VSoC installation but not in the testing of transactional memory.

3.4 C5

The C benchmark was developed as a microbenchmark was developed by the team from Brown University, Swarthmore College, and University of Bologna in transactional memory research. It utilizes basic synchronization on a scale much smaller than that of the STAMP benchmarks or even a benchmark like patricia.

The benchmark functions by performing operations on shared and local arrays. These operations are simple integer manipulation, and have no practical relevance.

These operations are split into 4 parts in the C benchmark. Each core has a local array and there is also

an array shared by all the cores. The local array just simulates work that doesn't require synchronization functionality. The shared array contains several vectors that overlap on each other. Constants such as the size of the vectors, number of overlap elements, and number of iterations performed are defined at the beginning of the testbench file.

The locks required for the C benchmark are specifically assigned to certain areas of the array. The layout for the lock setup is shown in the main source file.

3.5 Patricia

The original patricia benchmark was written by Matt Smart from The University of Michigan. His description of the functionality of the program is provided below.

* This code is an example of how to use the Patricia trie library for * doing longest-prefix matching. We begin by adding a default * route/default node as the head of the Patricia trie. This will become * an initialization functin (pat_init) in the future. We then read in a * set of IPv4 addresses and network masks from "pat_test.txt" and insert * them into the Patricia trie. I haven't yet added example of searching * and removing nodes.

This version of Patricia was then ported to the MPARM system simulator used at the Univeristy of Bologna in research of Transactional Memory Systems. This benchmark is useful because it contains several critical sections of code. Researchers can test different memory configurations in muti-core and many-core systems using the patricia benchmark because all the cores will have to share a single data structure.

Finally, in the summer of 2013, Peng and Callen (Swarthmore College) ported thisbenchmark to the Virtual System on Chip (VSoC) simulator used by Brown University and Swarthmore College in transactional memory research for many-core clustered systems.

3.6 Skiplist

The original skiplist benchmark was published on http://epaperpress.com as a sorting and searching example. The descriptions can be found here:

http://epaperpress.com/sortsearch/skl.html

This version of skiplist was then ported to the MPARM system simulator used at the Univeristy of Bologna in research of Transactional Memory Systems. This benchmark is useful because it contains several critical sections of code. Researchers can test different memory configurations in muti-core and many-core systems using the patricia benchmark because all the cores will have to share a single data structure.

Finally, in the summer of 2013, Peng and Callen (Swarthmore College) ported thisbenchmark to the Virtual System on Chip (VSoC) simulator used by Brown University and Swarthmore College in transactional memory research for many-core clustered systems.

3.7 Redblack

The original redblack benchmark was published on http://epaperpress.com as a sorting and searching example. The descriptions can be found here:

http://epaperpress.com/sortsearch/skl.html

This version of redblack was then ported to the MPARM system simulator used at the Univeristy of Bologna

in research of Transactional Memory Systems. This benchmark is useful because it contains several critical sections of code. Researchers can test different memory configurations in muti-core and many-core systems using the patricia benchmark because all the cores will have to share a single data structure.

3.8 K-Means

K-means was originally included in Stanford's STAMP benchmark suite for multiprocessor systems. It was then ported to the MPARM simulator, and this file documents the changes necessary to run it on the VSoC simulator developed by researchers from the University of Bologna.

K-means is a clustering algorithms that can cluster a set of vectors into a certain number of groups. For example, the main input to the benchmark contains 64 vectors with 8 elements each. The system clusters them into 8 groups. The test input, found in "goldinput.h", uses 12 objects each with 2 attributes, and clusters them into 2 groups.

The algorithm works in two stages. First, random vecors are chosen from the set as the initial cluster centroids. Then, each vector is assigned to the centroid that is closest to it. The centroid is then recomputed as the point which minimized the sum of squares distance between the centroid and each of the vectors. Then the vectors are reassigned again. These two steps are iterated until no vectors switch centroids and the distance each centroid moves when it is recalculated is reduced to zero.

3.9 Vacation

Vacation was originally released as part of the STAMP benchmark suite. It was ported to MPARM in the summer of 2008 by Trilok Acharya.

From the original README,

"This benchmark implements a travel reservation system powered by a non-distributed database. The workload consists of several client threads interacting with the database via the system's transaction manager.

The database is consists of four tables: cars, rooms, flights, and customers. The first three have relations with fields representing a unique ID number, reserved quantity, total available quantity, and price. The table of customers tracks the reservations made by each customer and the total price of the reservations they made. The tables are implemented as Red-Black trees."

It would be wise to read Trilok's README, found in the app directory, for details on how he ported the original STAMP benchmark to MPARM. In this README, I will only discuss changes that were made by Peng and I to get the benchmark to run on VSoC. The initial port to MPARM was a much, much larger project.

3.10 Genome

The original genome benchmark was written by Chi Cao Minh from Stanford University. His description of the functionality of the program is provided below.

* This benchmark implements a gene sequencing program that reconstructs the * gene sequence from segments of a larger gene. * * For example, given the segments TCGG, GCAG, ATCG, CAGC, and GATC, the * program will try to construct the shortest gene that can be made from them. * * For example, if we slide around the above segments we can get: * * TCGG * GCAG * ATCG * CAGC * GATC * ========== * CAGCAGATCGG * * * This gives a final sequence of length 11. Another possible

solution is: * * TCGG * GCAG * ATCG * CAGC * GATC * ========== * GATCGGCAGC * * This solution has length 10. Both are consistent with the segments provided, * but the second is the optimal solution since it is shorter. * * The algorithm used to sequence the gene has three phases: * * 1) Remove duplicate segments by using hash-set * 2) Match segments using Rabin-Karp string search algorithm [3] * - Cycles are prevented by tracking starts/ends of matched chains * 3) Build sequence * * The first two steps make up the bulk of the execution time and are * parallelized. * * * References * ——— * * [1] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford * Transactional Applications for Multi-processing. In IISWC '08: * Proceedings of The IEEE International Symposium on Workload * Characterization, September 2008. * * [2] C. Cao Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, * C. Kozyrakis, and K. Olukotun. An Effective Hybrid Transactional Memory * System with Strong Isolation Guarantees. In Proceedings of the 34th * Annual International Symposium on Computer Architecture, 2007. * * [3] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching * algorithms. IBM Journal of Research and Development, 1987.

3.11 Labyrinth

The original labyrinth benchmark was written by Chi Cao Minh from Stanford University. His description of the functionality of the program is provided below.

* * Given a maze, this benchmark finds the shortest-distance paths between pairs * of starting and ending points. The routing algorithm used is Lee's algorithm * [2]. * * In this algorithm, the maze is represented as a grid, where each grid point * can contain connections to adjacent, non-diagonal grid points. The algorithm * searches for a shortest path between the start and end points of a * connection by performing a breadth-first search and labeling each grid point * with its distance from the start. This expansion phase will eventually reach * the end point if a connection is possible. A second traceback phase then * forms the connection by following any path with a decreasing distance. This * algorithm is guaranteed to find the shortest path between a start and end * point; however, when multiple paths are made, one path may block another. * * * When creating the transactional version of this program, the techniques * described in [3] were used. When using this benchmark, please cite [1]. * * * References * ——— * * [1] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford * Transactional Applications for Multi-processing. In IISWC '08: * Proceedings of The IEEE International Symposium on Workload * Characterization, September 2008. * * [2] C. Lee. An algorithm for path connections and its applications. IRE * Trans. On Electronic Computers, 1961. * * [3] I. Watson, C. Kirkham, and M. Lujan. A Study of a Transactional Parallel * Routing Algorithm. Proceedings of the 16th International Conference on * Parallel Architectures and Compilation Techniques, 2007. *

3.12 ScalParC