Medellin Applied Research Concepts

*Hyper-converged Infrastructure Workload Code*
*Individual Workload Code Descriptions*
*v.1 -- August 2017*

## Overview

This document describes the first version of a set of programs that simulate workloads. The code is written in C and is portable on many environments. The objective of these products are to analyze various hyper-converged infrastructures for behavior in large scale usage of a particular resource (Compute, Memory or Network). This first generation only includes the Compute and Memory components.

## Key Characteristics

The programs are written in one of two main processing characteristics:

- Single thread, single process
- Multi-thread, single process

Different coding patterns are used within each one of those processing characteristics. A main objective of this variation is to test interrupt handling within the hypervisors or container managers that are under test, there are differences.

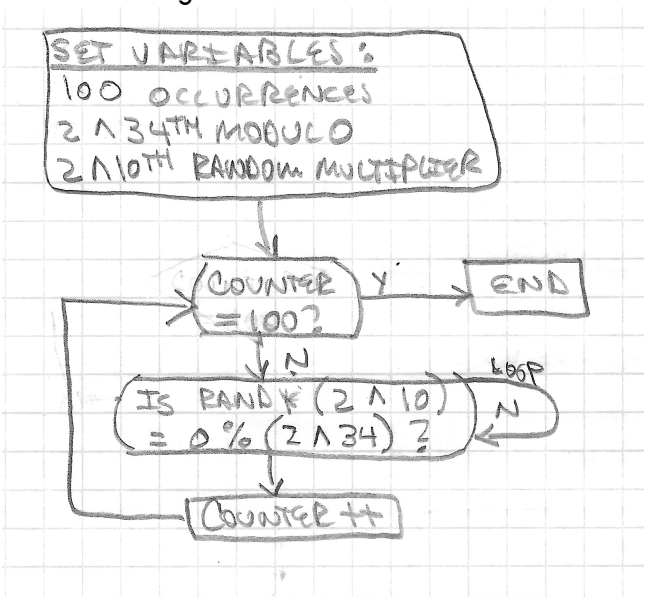
## The Compute Workload Programs

The Compute Workload Programs are cpu-intensive algorithms that do just that, exercise the CPU continuously. The algorithm used is to take a random number generated, multiply it by 2^Nth (the default is 2^10) and see if it is a modulo 0 of a 2^Xth (default is 2^34) and keep regenerating until a number of occurrences (default is 100) is found. Each of the 3 parameters can be changed, however, most C compilers will not support powers that are greater than 2^64th.

There are 2 programs:

- SMUProgram1a.c: single threaded
- SMUProgram1bthreads: employs several threads at a time

Here is the program flow for SMUProgram1a.c:



There is no need for a make file, the compile command for Linux is given on the first few lines of the code. The only difference between the single threaded version and the multi-threaded version is:

1. Threads and mutexes are instantiated.
2. Mutex locks the critical region (the counter) when an occurrence of modulo 0 happens and increases the counter.
3. Mutexes are used for synchronization.

Preliminary tests indicate that the multi-threaded version runs slower on bare metal but has a significant speedup in cloud environments. This is perhaps for the hypervisor's ability to handle interrupt processing more effectively that native linux itself.

**The Memory Workload Programs**

The memory workload programs exercise the memory hierarchy (register→cache→RAM→Secondary) in different ways. They all are based on a binary search algorithm and take as their input a 486,000 row table with two elements:

1. The value to find
2. A list of all the values to find sorted in ascending order

Processing essentially loads the two arrays into memory from a file provided in the tool kit and the sequence begins with the first value to be found. The second array is searched using a binary-partition algorithm until the value to be found matches the value in the second array. At that point, the row number is either written to a third array or is written to the output file.

Programs SMUProgram3a.c and SMUProgram3b.c are single threaded algorithms. Their compilation instruction is also found near the top of the code. Program 3a executes the algorithm as described above and 3b is just a straight record by record copy of the input file into an output file. BTW, all input files are the same name and all output files are the same name (output files are overwritten every time the program runs).

Program SMUProgram3c.c is a multithreaded version of program 3a. It defaults to 4 threads to perform the search and basically each thread looks for ¼ of the values. The number of threads can be varied (most C compiler directives will typically not allow for more than 300 threads to be open concurrently so watch out). The method used for synchronization is thread-specific data which is an advanced method of the posix standard (see IEEE). No interrupts are used here except when the threads are created and there are no mutexes or critical regions. The space is partitioned based on the number of threads and the threads look for separate occurrences from each other. The file is written at the very end once a third array with row numbers have been found for all the 486,000 unique numbers in the input file.

Because of the structural nature and design pattern of program 3c it is SIGNIFICANTLY faster than it's close sibling 3a in all the environments tested.

Next page has a program flow of 3c.

Here is the program flow for SMUProgram3c.c: