Designing a Memory Benchmark

by

Chris Bell, Scott Clark & Ryan Radcliff

Deopli Corporation

Executive Summary

Analyzing the performance of a memory subsystem, as a component of a full system, can be difficult. Most benchmarks are a scaled down version of some real tool, leading to difficulties isolating the performance of the memory from the rest of the server, such as CPU, chipsets, and storage. In addition, the memory's footprint is often a pitfall. Some tests from many years ago are still utilized, when gigabytes-worth of memory were only a dream, and only exercise a small percentage of RAM. However, modern systems have large caches with multiple levels. and benchmarks need to use a LOT of memory to avoid skewing results in the event of a fast on-chip cache.

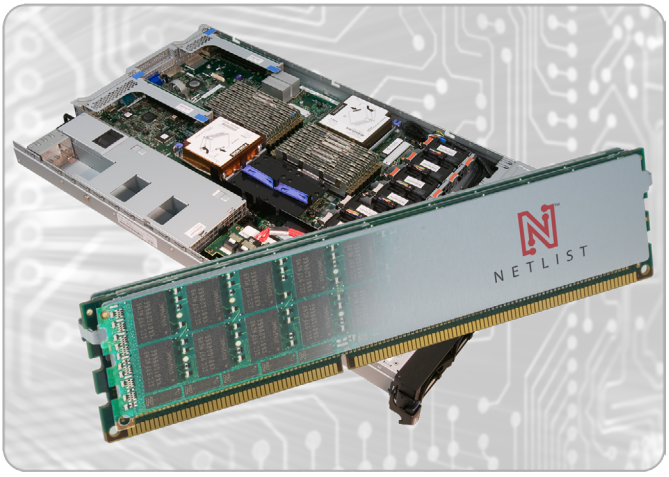
Modern systems involved in tests have two CPU sockets, with memory banks tied to each socket. With modern operating systems, predicting which CPU will execute a given job (or time slice of a job) has grown difficult. Therefore, it is wise to run multiple concurrent jobs, averaging the results in an effort to cancel out any bias from accessing memory, either directly or through the alternate CPU.

In summary, the tool must:

- Fit in memory and not rely on the data on disk or network
- Avoid using a lot of processing time between memory accesses
- Use a large enough memory footprint to negate the accelerated caches
- Easily use multiple processors/cores
- Be written as a benchmark that carefully measures and reports performance
- Be written as a benchmark with running parameters easily adjusted to compensate for various test scenarios (e.g., 144 or 288 GB system).

Although not a hard requirement, an open source tool is preferable - as the code can be inspected, reviewed, and modified, if necessary.

To meet these requirements, the open-source tool, Ramspeed, was selected. Ramspeed is specifically written to overcome the concerns listed above, as well as test the speed of memory reads and writes with as little interference as possible from the other components of a system.



DRAFT - NOT For Release

Installing Ramspeed/SMP

The following installation procedure assumes a Linux operating system (such as CentOS, Redhat, Ubuntu) with a working GCC build environment. The examples below are presented in command-line format and a familiarity with Unix shell commands and scripts is assumed.

1. Get source code and documentation from the following: http://alasir.com/software/ramspeed/
2. Use the SMP flavor, version 3.5.0: http://www.alasir.com/software/ramspeed/ramsmp-3.5.0.tar.gz
3. Unpack the archive:
   `tar zxvf ramsmp-3.5.0.tar.gz`
4. Change directory to the newly created directory:
   `cd ramsmp-3.5.0`
5. Build the tool. The proper OS should be automatically detected and binary "ramsmp" produced.
   `./build.sh`

Performing Memory Tests

The first choice was the test number, which is related to the mathematical operations used. A simple set was chosen, excluding the fancier MMX or SSE instructions, that simply adds and multiplies. The graph below displays the block size (in kilobytes) on the X-axis, and the total memory speed (in Megabytes/second) on the Y-axis. The three plots are the number of concurrent processes (2, 4 and 8). The following three runs are displayed:

```
../ramsmp -b 1 -m 8192
../ramsmp -b 1 -m 4096 -p 4
../ramsmp -b 1 -m 2048 -p 8
```

Note: Different block sizes are intended to limit the test to a maximum total memory size - so that tests without a full 288 GB Ram could be run using the same script.

As illustrated, the affect of caches can be observed by varying the block size of the memory tests (as shown on the X-axis, in Kb/block). This particular CPU contains three levels of on-chip cache, obvserved by drops near 64Kb, 512Kb, and 4Mb. In addition, the interaction between multiple cores can is observed, as the relative speeds converge when accessing main memory.

The L1 (32K data) and L2 (256K) cache is dedicated to each core, and as the number of processes increases, the total memory speed increases linearly (on the left-hand side of the graph). The L3 cache (12MB) is shared over the 6 cores in a CPU chip, and the third plateau does not have as much benefit from the 8 cores, since they are all sharing the cache. Above the 16MB block size (16384 on the graph), the effects of cache are overwhelmed as all three lines settle to a fairly consistent level. This action represents the speed of the memory system. Therefore, as high memory use is the intent of the test, a point larger than 16MB is chosen. This selection ensures no cache was involved, while large memory utilization is represented by selecting 1024K (which is 1048576 on the graph) for the block size. In addition, the highest concurrency (i.e., 8 processes) is chosen to utilize both CPU chips.

The published results were generated using the following run:

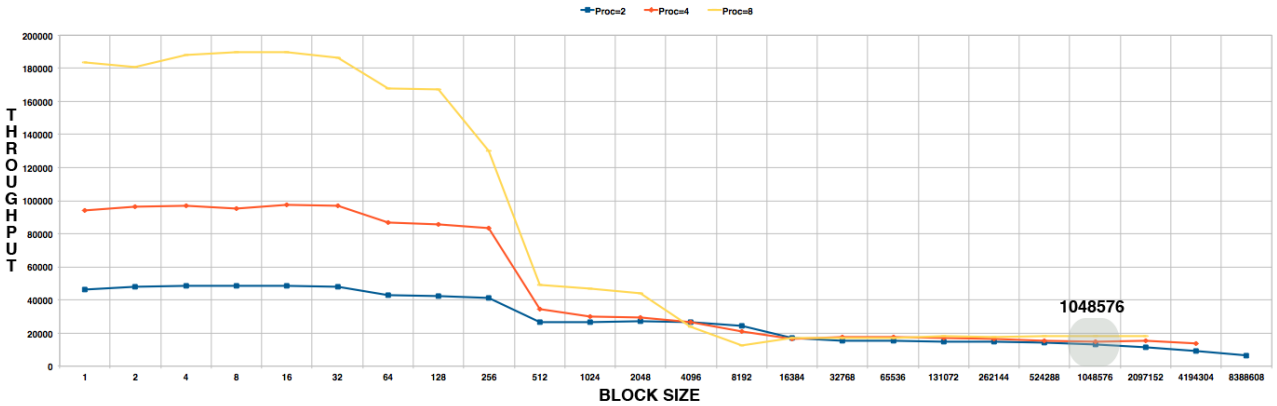
```
./ramsmp -b 3 -m 1024 -p 8
```

The output of this test (test=3) displays as the following:

```
./ramsmp -b 3
8Gb per pass mode, 2 processes
INTEGER   Copy:      2614.68 MB/s
INTEGER   Scale:     2568.34 MB/s
INTEGER   Add:       2804.88 MB/s
INTEGER   Triad:     2783.94 MB/s
INTEGER   AVERAGE:   2692.96 MB/s
```

The output of the incremental tests (like test=1 or 2) displays as the following:

```
./ramsmp -b 1
8Gb per pass mode, 2 processes
INTEGER & WRITING    1 Kb block: 76996.27 MB/s
INTEGER & WRITING    2 Kb block: 80227.28 MB/s
INTEGER & WRITING    4 Kb block: 80828.76 MB/s
INTEGER & WRITING    8 Kb block: 81128.01 MB/s
```



HyperCloud DIMM Test Results. The 1048576 block size is used as the basis for additional test results.

```
INTEGER & WRITING   16 Kb block: 76981.78 MB/s
INTEGER & WRITING   32 Kb block: 80577.03 MB/s
INTEGER & WRITING   64 Kb block: 79105.38 MB/s
INTEGER & WRITING  128 Kb block: 20166.38 MB/s
INTEGER & WRITING  256 Kb block: 20161.37 MB/s
INTEGER & WRITING  512 Kb block: 20330.83 MB/s
INTEGER & WRITING  1024 Kb block: 4409.01 MB/s
INTEGER & WRITING  2048 Kb block: 1934.89 MB/s
INTEGER & WRITING  4096 Kb block: 1916.94 MB/s
INTEGER & WRITING  8192 Kb block: 1919.43 MB/s
INTEGER & WRITING 16384 Kb block: 1921.77 MB/s
INTEGER & WRITING 32768 Kb block: 1929.74 MB/s
```

The actual script run under the various test conditions ran a bunch of tests to provide a sanity check, and then the whole script was run multiple times and the results averaged to reduce any anomalous glitches. Here is the full script:

```
#!/bin/sh -x

run="run4"

echo =====

echo =====

echo 3-marray-256-proc-4

../ramsmp -b 3 -m 256 -p 4 | tee ${run}-3-
marray-256-proc-4.log

echo =====

echo 3-marray-512-proc-4

../ramsmp -b 3 -m 512 -p 4 | tee ${run}-3-
marray-512-proc-4.log

echo =====

echo 3-marray-1024-proc-4

../ramsmp -b 3 -m 1024 -p 4 | tee ${run}-3-
marray-1024-proc-4.log

echo =====

echo 3-marray-2048-proc-4

../ramsmp -b 3 -m 2048 -p 4 | tee ${run}-3-
marray-2048-proc-4.log

echo =====

echo 3-marray-4096-proc-4

../ramsmp -b 3 -m 4096 -p 4 | tee ${run}-3-
marray-4096-proc-4.log

echo =====

echo =====

echo 3-marray-256-proc-8

../ramsmp -b 3 -m 256 -p 8 | tee ${run}-3-
marray-256-proc-8.log

echo =====

echo 3-marray-512-proc-8

../ramsmp -b 3 -m 512 -p 8 | tee ${run}-3-
marray-512-proc-8.log

echo =====

echo 3-marray-1024-proc-8

../ramsmp -b 3 -m 1024 -p 8 | tee ${run}-3-
marray-1024-proc-8.log

echo =====

echo 3-marray-2048-proc-8

../ramsmp -b 3 -m 2048 -p 8 | tee ${run}-3-
marray-2048-proc-8.log

echo =====

echo =====

echo 1-marray-8192-proc-2

../ramsmp -b 1 -m 8192 | tee ${run}-1-
marray-8192.log

echo =====

echo 1-marray-4096-proc-4

../ramsmp -b 1 -m 4096 -p 4 | tee ${run}-1-
marray-4096-proc-4.log

echo =====

echo 1-marray-2048-proc-8

../ramsmp -b 1 -m 2048 -p 8 | tee ${run}-1-
marray-2048-proc-8.log

echo =====

echo =====

echo 2-marray-8192-proc-2

../ramsmp -b 2 -m 8192 | tee ${run}-2-
marray-8192.log

echo =====

echo 2-marray-4096-proc-4

../ramsmp -b 2 -m 4096 -p 4 | tee ${run}-2-
marray-4096-proc-4.log

echo =====

echo 2-marray-2048-proc-8

../ramsmp -b 2 -m 2048 -p 8 | tee ${run}-2-
marray-2048-proc-8.log

echo =====

echo =====

echo 7-marray-8192-proc-2

../ramsmp -b 7 -m 8192 | tee ${run}-7-
marray-8192.log

echo =====

echo 7-marray-4096-proc-4

../ramsmp -b 7 -m 4096 -p 4 | tee ${run}-7-
marray-4096-proc-4.log

echo =====

echo 7-marray-2048-proc-8

../ramsmp -b 7 -m 2048 -p 8 | tee ${run}-7-
marray-2048-proc-8.log

echo =====
```

```
echo =====

echo 10-marray-8192-proc-2

../ramsmp -b 10 -m 8192 | tee ${run}-10-
marray-8192.log

echo =====

echo 10-marray-4096-proc-4

../ramsmp -b 10 -m 4096 -p 4 | tee ${run}-10-
marray-4096-proc-4.log

echo =====

echo 10-marray-2048-proc-8

../ramsmp -b 10 -m 2048 -p 8 | tee ${run}-10-
marray-204
```

Interpreting Results

The Ramspeed tool prints the results as a total bandwidth the group of processes achieved. The results are highly dependent on various factors within the design of the system, as well as the current configuration. As well, care must be taken to ensure that the exact same option settings are chosen before comparing different runs. Memory speeds, interleaving, CPU clock speeds, CAS/RAS timings and block sizes all contribute to the resulting equation - proving an easy formula does not exist for computing <xx MB/s> to <yyy memory clock speed>. However, the described tests still provide an indication of the kind of performance a real application can reach, and the results are useful as a relative scale (similar to old SPECint numbers).

References

Ramspeed/SMP [http://alasir.com/software/ramspeed/] •

- Intel's Tick-Tock Model [http://www.intel.com/ technology/tick-tock/ index.htm] •
- Intel Xeon Processor 5500 Series Datasheet, Volume 2 (April 2009) [http://www.intel.com/Assets/en_US/PDF/ datasheet/321322. pdf] \
- Intel Xeon Processor 5600 Series Datasheet, Volume 2 (March 2010) [http://www.intel.com/Assets/en_US/PDF/ datasheet/323370. pdf]