***Microsoft Confidential***

**MemoryPerf**

*A Windows Mobile Memory Performance Test*

July 17, 2009

Copyright © 2009 Microsoft Corporation

All Rights Reserved.

Table of Contents

[Introduction 3](#_Toc235603590)

[Exe Installation and Operation: 4](#_Toc235603591)

[Exe Command Line arguments 4](#_Toc235603592)

[TUX DLL Installation and Operation: 5](#_Toc235603593)

[TUX DLL Command Line arguments 5](#_Toc235603594)

[MemoryPerf Key Concepts and Techniques 6](#_Toc235603595)

[1. Access Techniques 6](#_Toc235603596)

[2. 1-D Patterns 6](#_Toc235603597)

[3. 2-D Patterns 7](#_Toc235603598)

[4. Latency Measurements 7](#_Toc235603599)

[5. Size Measurements 8](#_Toc235603600)

[6. CPU Speed 8](#_Toc235603601)

[7. Memory Copy 9](#_Toc235603602)

[8. Extra SDRAM Row Access Time 9](#_Toc235603603)

[9. Uncached Memory Testing 9](#_Toc235603604)

[10. DDraw Surface Testing 9](#_Toc235603605)

[11. Warnings 10](#_Toc235603606)

[12. Power Management 10](#_Toc235603607)

[13. Scoring 10](#_Toc235603608)

[Tests 11](#_Toc235603609)

[Page Walk and Translation Look-aside Buffer Measurements: 11](#_Toc235603610)

[L1 Read Measurements: 11](#_Toc235603611)

[L2 Read Measurements: 11](#_Toc235603612)

[Memory Bus Read Measurements: 12](#_Toc235603613)

[Write Measurements 12](#_Toc235603614)

[Cached MemCpy Measurements 12](#_Toc235603615)

[Overviews of Cached Memory Performance 12](#_Toc235603616)

[Uncached Memory Tests 12](#_Toc235603617)

[DDraw Memory Tests 13](#_Toc235603618)

[PerfScenario Output 13](#_Toc235603619)

[CSV Output 14](#_Toc235603620)

[Contact 16](#_Toc235603621)

# Introduction

MemoryPerf is a Windows Mobile memory performance tests that comes in two forms:

* MemoryPerfExe.exe is the executable form of the Windows Mobile Memory Performance Test.
* MemoryPerf.dll provides exactly the same functionality in a TUX DLL to facilitate automation plus it reports performance results to the PerfScenario database and can be integrated with the BSP Test Suite.

MemoryPerf measures many of the latencies, sizes and properties of the various stages of the memory hierarchy such as L1, L2, SDRAM, and TLB. It does this to illuminate the key memory component performance characteristics of a system. It also measures these memory characteristics when RAM is accessed in different modes (cached, uncached, DDraw surface). Then it combines these measurements into a final benchmark score which rates a system on how well its memory characteristics affect a key Windows Mobile End-User Scenario.

Project goals:

* An automated test to catch regressions in BSP configurations of a device’s memory subsystem.
* An automated test to measure and report memory performance within the BTS including a single benchmark score to represent how the low-level measurements combine to effect real-work WM7 performance.
* Automated warnings if a BSP has configured the memory subsystem in unusual ways (no-allocate-on-write-miss) or does not report correct info from the CeGetCacheInfo system function.
* A tool which can be used in manual mode to investigate and diagnose issues first observed by the automated mode.

# Exe Installation and Operation:

To use the executable in a stand-alone fashion:

1. Copy MemoryPerfExe.exe to the device’s \Windows directory using a tool like Windows Mobile Device Center (WMDC) or Platform Builder’s Remote Tools File Viewer.
2. Copy the four MemoryPerfExe\*.lnk files to the device’s “\Windows\Start Menu” directory
3. Press the start icon and then navigate to the one of the links
   1. MemoryPerfExe will run the program with no command line arguments. Output will be to a log file.
   2. MemoryPerfExe\_C will run the program with –c to create an excel .csv file as well as the log file.
   3. MemoryPerfExe\_CD will run the program with –c –d to tee the log info to the debug output stream.
   4. MemoryPerfExe\_CDO will run the program with –c –d -o to also output additional overview graphs.
4. Only press the appropriate MemoryPerfExe\* link once!
   1. You will get no visual feedback that anything is happening on the device as the program does not pop any UI. Running with no options takes about 3 minutes and –c –d –o takes about 10 minutes to complete.
   2. You can see debug output if you have a debug channel (like KITL) connected.
   3. If WMDC is connected to the device, you can navigate to “\Storage Card” and see the log file grow in size gradually to about 10K by clicking on refresh every once in a while.
5. MemoryPerfExe.log and MemoryPerfExe.csv files are written to the first directory in the following list where they can be accessed with WMDC.
   1. \Release (if KITL is connected)
   2. \Storage Card
   3. \My Documents
   4. \

Exe Command Line arguments:

MemoryPerf.exe [-d] [-f] [-l ddd] [-p [-p]] [-s ddd] [-u] [-v]

-c outputs csv formatted details for charting in excel

-d tees output to debug output

-f fills all allocated buffers with random data before the test

-h toggles flag which heats up the CPU before the test starts (default on)

-l ddd sets the priority level for the tests to ddd

(default is 152 which should allow the power manager to run,

consider 0 if power management is disabled for best accuracy)

-o toggles overview which shows additional CSV graphs (default off)

-p [-p] prints additional statistically diagnostic info to the log file

-s ddd sets sleep time at priority changes to ddd ms

(default is 0, consider 1 if power management is disabled)

-u toggles the uncached tests (normally enabled)

-v toggles the video (DDraw) surface tests (normally enabled)

-w toggles warnings (normally enabled)

-2 toggles testing for L2 (normally enabled). Disable when no L2 is present.

# TUX DLL Installation and Operation:

To use the tux dll with a KITL connection:

1. Copy MemoryPerf.dll to your Flat Release Directory (FRD).
2. In the FRD, copy ceperf\_module.dll ceperf.dll
3. Start program by one of the following typical commands or a variant:
   1. s tux -o -d MemoryPerf.dll -f\release\memoryperf.log
      1. Output will be to FRD\MemoryPerf.log
      2. This takes about 3 minutes to run on typical chassis 1 hardware.
   2. s tux -o -d MemoryPerf.dll –c”-c” -f\release\memoryperf.log
      1. Same log and Excel spreadsheet is created on \release\MemoryPerf.csv
   3. s tux -o -d MemoryPerf.dll –c”-c –o –p” –f\release\memoryperf.log
      1. additional detailed output to the CSV
      2. This test takes about 10 minutes to complete.
4. Add –c”-s 1 –l 0” when Power Management is disabled and expect the test to take a minute less to complete.
5. MemoryPerf.log and MemoryPerf.csv files are written to the first directory in the following list where they can be accessed with WMDC or File Viewer.
   1. \Release (if KITL is connected)
   2. \Storage Card
   3. \My Documents
   4. \

TUX DLL Command Line arguments:

s tux –o –d –d MemoryPerf.dll [–c”options (see below)”] –f\release\memoryperf.log

MemoryPerf options within the quotes of –c””:

-c outputs csv formatted details for charting in excel

-f fills all allocated buffers with random data before the test

-h toggles flag which heats up the CPU before the test starts (default on)

-l ddd sets the priority level for the tests to ddd

(default is 152 which should allow the power manager to run,

consider 0 if power management is disabled for best accuracy)

-o toggles overview which shows additional CSV graphs (default off)

-p prints additional statistically diagnostic info to the log/csv files

-s ddd sets sleep time at priority changes to ddd ms

(default is 0, consider 1 if power management is disabled)

-u toggles the uncached tests (normally enabled)

-v toggles the video (DDraw) surface tests (normally enabled)

-w toggles warnings (normally enabled)

-2 disables testing for L2 (normally enabled).

Additional TUX options:

-x1 run only test 1 – cached memory test

-x2 run only test 2 – uncached memory test

-x3 run only test 3 – DDraw Surface memory test

-x4 run only test 4 – Score Calculation (will run other tests to collect data)

# MemoryPerf Key Concepts and Techniques

MemoryPerf is based on a number of key techniques it uses to measure memory access latencies and to find patterns in sets of these latencies which represent underlying physical processes in the silicon and drivers which configure the silicon.

1. Access Techniques
   1. Memory is accessed in highly unrolled loops (unrolled up to 16-times) so timing measurements are dominated by the latency to read or write caches or memory rather than loop overhead.
   2. To minimize thread-switching which causes inaccurate measurements, the measured loop is run at very high priority (level 0 which is above device drivers) but that priority can be adjusted by command line argument.
   3. The test is repeated 16-times (see below) and the median of the 16 measurements is used to avoid allowing outliers from influencing the results.
   4. Each loop is unrolled to minimize loop overhead.
   5. Each access is a conventional LDR or STR of a 4-byte DWORD aligned modulo 4-bytes as this is the most frequent way memory is accessed in product code.
2. 1-D Patterns
   1. Memory is accessed in simple 1-D or 2-D patterns which step through memory in a simple and regular way reading or writing DWORD (4-byte) sized aligned quantities.
   2. The stride of the pattern varies from test to test. It may be 4-bytes (sequential), 32-bytes or 128 bytes (cache line size), 4096-bytes (page size) or something else to measure a particular memory access component and to avoid averaging different levels of the memory access hierarchy.
   3. The span of the pattern is the total range of accesses. So, for example, a pattern which accessed 512 DWORDS separated by 32-bytes would span 16KB. Span is typically used to control which level of the memory hierarchy is being accessed and measured.
   4. A 1-D Stride Pattern is described by (Y\_Strides,Y\_Stride).
   5. A 1-D read pattern is executed in the following way:

int iTotal = 0;

for (int iRpt = 0; iRpt < ciRepeats; iRpt++)

{

volatile const DWORD\* pdwT = pdwBase;

for ( int i = Y\_Strides; i > 0; i-- )

{

iTotal += \*pdwT;

pdwT += Y\_Stride;

}

}

* 1. The actual loop is special-cased for small Y\_Strides values and is unrolled up to 16-times to minimize loop overhead.
  2. Typical separation of LDR and the use of the register loaded varies between 3 and 6 instructions and averages 4.5 for most loops. There is only two pending loads even though the CPU’s typically can have three or more loads outstanding.

1. 2-D Patterns
   1. 2-D Memory Access patterns accesses a matrix or grid of DWORDS whose access is controlled by the X\_Stride size, the number of X\_Strides, the Y\_Stride size, and the number of Y\_Strides represented as (Y\_Strides,Y\_Stride,X\_Strides,X\_Stride).
   2. While the most efficient way to access such a grid is to set up the pattern to access DWORDS which are close together to take advantage of the locality designed into the silicon, the 2-D pattern is used within this test mainly in the opposite way to expose phenomena in more distant levels of the memory hierarchy. Many important algorithms used in graphics and multimedia (e.g. those that include a matrix transpose) must access their 2-D structures in the non-locality-based way.
   3. A 2-D read pattern is executed in the following way:

int iTotal = 0;

for (int iRpt = 0; iRpt < ciRepeats; iRpt++)

{

volatile const DWORD\* pdwY = pdwBase;

for( int iY = Y\_Strides; iY > 0; iY--, pdwY += Y\_Stride )

{

volatile const DWORD\* pdwX = pdwY;

for (int iX = X\_Strides; iX > 0; iX--, )

{

iTotal += \*pdwX;

pdwX += X\_Stride;

}

}

}

1. Latency Measurements
   1. Latency is measured by using QueryPerformanceCounter() which is a 32 KHz user accessible clock.
   2. The pattern is executed once right before the timing starts to warm-up the caches.
   3. The pattern is repeated N times over the same memory range in a tight loop so that the total duration is at least one millisecond to provide accuracy relative to the QPC time base.
   4. Each measurement is repeated 16-times starting at different offsets into a 27MB virtual memory buffer. This variation tests the pattern at different virtual-to-physical mappings. When a device has just been booted, the allocated 27MB buffer tends to have a fairly linear relationship between virtual and physical pages. But when device has done a lot of stuff since boot, the mapping can be skewed and some cache sets and ways can be under or over utilized since the caches are physically mapped. So some variations is expected.
   5. Latencies are always expressed in a per-access way so comparisons between patterns with different numbers of accesses can be made.
   6. The median of the 16-latencies are primarily reported in the log files although the mean and standard deviation is also reported in the log file as a sanity check. Additionally, the mean and standard-deviation are reported. In the CSV file, all 16-latencies are reported.
2. Size Measurements
   1. Sizes of caches, for example, are measured using collections of latency measurements which vary by stepping one of the pattern parameters (Y\_Strides, Y\_Stride, X\_Strides, X\_Stride) though a range. Any of the four parameters can be varied and that parameter can step using linear or power of two sequences.
   2. Two techniques are used to evaluate the set of results – one chooses the estimated size and the other evaluates the confidence that this estimated size is the actual physical size based on the consistency of the measurements.
   3. The size is determined by dividing the range of input variation into three sub-ranges (Low, Middle, High) which Low is the bottom of the input range which showed the lowest latency measurements and High is the range at the top of input range which show high latency and Middle is some possibly empty range which the latency transitions between low and high in a possibly irregular fashion. The input range is designed to transition across one or more levels of the memory hierarch.
      1. For example, to measure the size of L1, a set of 1-D patterns are used (128,64), (256,64), (512,64), (1024,64), (2048,64) where the first is a pattern which strides 64-bytes between DWORD reads and does so 128-times before repeating. This set of patterns spans more than the expected sizes of L1 from 8 KB to 128 KB. The stride of 64 is used because the program does not expect an L1 to have a larger cache line size than 64. The first three consistently yield low latency which the fourth and fifth yield much higher latencies. So the size of L1 is determined to be 512\*64 = 32 KB.
      2. For most size measurements, the top of the low range is the size of interest. But for some the bottom of the high range is the size of interest.
   4. The confidence is determined using a Mann-Whitney U-Test’s ρ-statistic normalized to a range of 0-1 by the formula Confidence = 2\*(ρ>½ ? ρ : 1-ρ) so the ρ-statistics near 0 or 1 yield high confidence. The U-Test is used because it is a non-parametric test of whether two independent sets of observations come from the same or different underlying distributions.
      1. The tests were designed to yield high confidence on a freshly booted system whose system design is within the expected range of Windows Mobile Seven processors.
      2. Confidences decrease when the virtual to physical map is not linear or when power management changes the frequency of the processor.
3. CPU Speed
   1. CPU speed when not accessing memory is evaluated using a loop composed of 16 ADD, 1 SUB and one BNE instructions to evaluate the raw speed of the CPU.
   2. While the test does not know the expected IPC of this loop, the CPU clock rate can be determined if some assumption is made on IPC.
   3. Measured CPU Speed can be compared to the BSP reported ProcessorInfo MHz value reported near the beginning of the log file.
   4. The test is run numerous times throughout the execution of all of these tests to detect if the CPU clock rate is changing since clock rate changes have an adverse effect on the accuracy and reliability of the tests.
4. Memory Copy
   1. MemCpy test measures the memcpy() function rather than one of the custom unidirectional read or write 1-D or 2-D patterns used elsewhere. Inherently it is a 1-D pattern with a stride of 4. It measures the combined latency of read and write taken together. This test is useful since many platforms have optimized versions of memcpy() which utilize multiple load and store instructions which may have better performance that single loads and stores instructions. However, the compiler does not generate LDM and STM except for saving and restoring registers or other specialized uses.
5. Extra SDRAM Row Access Time
   1. SDRAM Row Timing is measured in a number of ways. It is important because real-world end-user scenario’s memory accesses that miss both L1 and L2 are statistically random so they tend to change SDRAM rows frequently.
   2. Measurements are made using a 2-D pattern where the X stride is large to try to cross a ROW boundary and the Y stride is small to overfill the caches so memory is actually accessed.
   3. SDRAM Row timing can be hard to measure because there are many different configurations of SDRAM that can be used in a device and the CPU can access those SDRAMs by mapping address bits to them in many different ways. To be robust to different SDRAM configurations, the latency is measured for a collection of patterns just like those used to estimate the size of caches. What is reported is the maximum of the medians of all the patterns measured.
      1. The “maximum of medians” is described by the following example: Consider three patterns A, B, and C that vary in one of their parameters, so there are 16 A latency, 16 B latency and 16 C latency measurements. The value reported is the max( median(A1..A16), median(B1..B16), median(C1..C16) ).
      2. Max of Medians is used under the hypothesis that one or more of the patterns will actually access SDRAM and cross a row boundary on every access of its pattern.
6. Uncached Memory Testing
   1. Uncached Memory Tests test memory allocated with VirtualAlloc() with the PAGE\_NOCACHE flag set.
   2. Such memory is sometimes used by drivers and others although by design we do not expect extensive use of it.
   3. To allow it to be used to communicate between different processors or cores, Windows Mobile defines it to be Strongly Ordered.
7. DDraw Surface Testing
   1. DDraw Surface Tests test memory allocated as a DDraw off-screen surface such as a process would use to write pixels that will be composited by the GPU and displayed on the LCD.
   2. Writing such memory is a frequent activity of real-world end-user scenarios.
   3. Using Strongly-ordered memory is highly undesirable for most systems which should prefer uncached-buffered or cached memory with a flush on unlock.
   4. A DDraw surface can be opened as Read-Only (RO), Write-Only (WO), and Read-Write (RW). The Write-Only mode has a variant called Discard which asserts that the complete surface will be overwritten.
   5. Since in normal use a DDraw surface is typically about the screen size (480x800x4) or less, that is the size used and so the spans of many of these tests are quite a bit smaller than other tests.
8. Warnings
   1. Warnings are displayed in the log when a measurement or collection of measurements falls out of an expected range as well as when some things like the cache policy for Write-Through vs. Write-Back or Allocate-On-Write-Miss vs. No-Allocate-On-Write-Miss are unexpected.
   2. Individual systems may benefit from unexpected values, but each warning should be investigated since Microsoft has reason to expect typical systems will perform best when configured in particular ways.
9. Power Management
   1. Power management is an important and necessary technique to maximize battery life while providing good performance to the device when it is being actively used.
   2. However, changing CPU frequency can have an adverse effect on these latency measurements and lead to unpredictable results.
   3. The test warms the CPU up using a combination of memory and cpu intensive tasks before starting the test.
   4. The CPU speed is measured after warm-up and before the first test as well as several other times between tests. If the CPU frequency is detected to change, the warm-up routine is run again so these measurements will run at the highest CPU frequency possible.
   5. Warnings will be issued about changes in frequency as the reliability of the results will be adversely affected.
   6. The default options –s 0 –l 152 is recommended for devices with active power management. This does several things:
      1. –l 152 sets the thread priority to 152 (below power management) so the power manager gets to run even during the measured test loops. –l 0 will put the test above the power manager and everything else except interrupts for best accuracy, but the power manager will not be able to speed up the CPU to met the demands of the test
      2. –s 0 removes the 1 ms sleep time normally inserted right before a measurement and causes each measurements to be preceded by 10 milliseconds of warming activity so the power manager has a chance to ramp the voltage and CPU frequency up. This extra 10 ms before each duration measurement substantially increases the total runtime of the test by approximately 30 seconds when run with no options and about 7 minutes for –c –o. This time can be saved when Power Management is disabled by –s 1 –l 0.
10. Scoring
    1. Scoring is done by weighing the various measurements in a way so that the score represents the relative cost of the device’s memory performance compared to a idealized standard device. The weights represent the relative use of these modes in a trace of a representative real-world end-user scenario.
    2. Details to be evolved.

Tests (In logical order but also roughly in order of measurement and reporting):

## Page Walk and Translation Look-aside Buffer Measurements:

* Page Walk Latency strides 4160 (=4096+64) bytes 125 times spanning 523780 bytes (125,4160) so each access should cause a hardware page walk.
* TLB Explore Size measures (16,4224) to (4096,4224) stepping in powers of 2 to scope a wide range of possible TLB sizes.
* TLB Size measures (4,4224) to (128,4224) in steps of 4 although the 128 will vary depending on the results of TLB Explore.
* The TLB size is estimated as the top of the Low range.

## L1 Read Measurements:

* L1 Read Latency strides 64 bytes 256 times spanning 16KB (256,64) so each access should hit L1.
* Preliminary L1 Size measures (128,64) to (2048,64) stepping in powers of 2. It is preliminary because L1 Line size has not yet been measured.
* L1 Line Size measures (32768,12) to (32768,84) in steps of 8.
* L1 Revised Size is measured (256,32) to (4096,32) in power of 2 steps shown for an L1 Line Size having been measured as 32.
* L1 Replacement Policy is measured (128,32) to (2048,32) in power of 2 steps and the collection of results correlated to a cache model which varies replacement policy between { Global Round Robin, Per Set Round Robin, Per Set PLRU }. Typically if a Per Set policy is used, the two Per Set policies tie and Per Set Round Robin is assumed.

## L2 Read Measurements:

* Preliminary L2 Size measures (14336,32) to (3584,32) in steps of power of 2. Less strides than expected are used (14336 instead of 16384) to avoid noise created by non-linear virtual-to-physical mappings causing L2 to be under or over utilized.
* L2 Line Size measures (49152,32) to (49152,256) in power of 2 steps.
* L2 Revised Size measures (448,128) to (7168,128) in power of 2 steps where the 128 is the L2 Line size determined by the measurement above.
* L2 Replacement Policy measures (512,128) to (4096,128) in steps of 128. Typically if a Per Set policy is used, the two Per Set policies tie and the tie is not broken until a later test.
* L2 Read Latency is measured with (1024,128).
* A tie in L2 Per Set Replacement Policy is broken by a really weird pattern not a simple 1-D or 2-D pattern

## Memory Bus Read Measurements:

* Cached MBus Latency is measured with (8192,128) spanning 1 MB.
* Full Sequential MBus Latency is measured with (262144,4) which will utilize and average all levels of the memory hierarchy.
* RAS 2K^i Scan SDRAM Row measures (20,32,192,2048) to (20,32,192,32768) in power of 2 steps to determine the SDRAM Row size. The pattern is essentially a saw-tooth sitting on a shallow ramp. This measurement is very sensitive to the linearity of the virtual-to-physical mapping because it touches so much memory.
* Read 16K+Line Stride Memory Read Latency measures (129,128,8,16512) to (129,128,264,16512) and reports the maximum of the medians of the latencies (see ).
* Read 32K+Scan Stride Read Latency measures (257,128,8,32896) to (257,128,264,32896) for when SDRAM’s grow in size.

## Write Measurements

* L1 Write Latency is measured with (512,32).
* L1 Preloaded ps Write Latency is measured the same but after reading the memory once to bring it into L1 to detect No-Allocate-On-Write-Miss cache policies.
* L2 Write Latency measured with (512,128) spanning 64KB.
* L2 Preloaded Write Latency is measured the same but after reading the memory once to bring it into L2 to detect No-Allocate-On-Write-Miss cache policies.
* Cached Memory Bus Write Latency
* Cached 16K-Row Memory Bus Write Latency
* Cached 32K-Row Memory Bus Write Latency
* Cached Full Sequential Memory Bus Write Latency

## Cached MemCpy Measurements

* Cached L1-Span MemCpy Read-Write Latency
* Cached L2-Span MemCpy Read-Write Latency
* Cached Memory-Span MemCpy Read-Write Latency

## Overviews of Cached Memory Performance

Overviews are provided when the command line argument –o is used in order to generate graphs in the .csv file which show an overview of the memory performance of the device.

* LineSize Stride Overview Read Latency
* SDRAM Row Stride Overview Read Latency

## Uncached Memory Tests

* Uncached Small-Span DWORD-Sequential Memory Bus Read Latency
* Uncached Large-Span DWORD-Sequential Memory Bus Read Latency
* Uncached Large-Span LineSize-Strided Memory Bus Read Latency
* Uncached Small-Span DWORD-Sequential Memory Bus Write Latency
* Uncached Large-Span DWORD-Sequential Memory Bus Write Latency
* Uncached Large-Span LineSize-Strided Memory Bus Write Latency
* UnCached Memory-Span MemCpy Read-Write Latency
* Cached=>UnCached Memory-Span MemCpy Read-Write Latency
* UnCached=>Cached Memory-Span MemCpy Read-Write Latency

## DDraw Memory Tests

* DDraw-RO DWORD-Sequential Memory Bus Read Latency
* DDraw-RO LineSize-Strided Memory Bus Read Latency
* DDraw-RO to Cached MemCpy Read-Write Latency
* DDraw-WO DWORD-Sequential Memory Bus Write Latency
* DDraw-WO LineSize-Strided Memory Bus Write Latency
* DDraw-WO from Cached MemCpy Read-Write Latency
* DDraw-RW DWORD-Sequential Memory Bus Read Latency
* DDraw-RW DWORD-Sequential Memory Bus Write Latency
* DDraw-RW Memory-Span MemCpy Read-Write Latency
* DDraw-RW to Cached MemCpy Read-Write Latency
* DDraw-RW from Cached MemCpy Read-Write Latency

# PerfScenario Output

The Tux DLL version creates a file called MeoyryPerf\_test.xml. Key fields in that xml show measurements suitable for regression analysis.

* All latency values are reported in picoseconds where smaller is better.
* Sizes are reported in registers for the TLB or bytes for other sizes. Larger may be better but due to design trade-offs smaller may be better.
* Overall Memory Score is percentage of minimum target device performance where larger is better and the score should be greater than 100%.

|  |  |
| --- | --- |
| Statistic Name | ChangeAverage |
| RAW Instruction Latency ps | 700 |
| L1 Read Latency ps | 1,780 |
| Page Walk Latency ps | 21,080 |
| Effective TLB Size | 64 |
| Final L1 Cache Line Size | 32 |
| Final L1 Size | 32,768 |
| Final L2 Cache Line Size | 128 |
| Final L2 Size | 262,144 |
| L2 Read Latency ps | 10,070 |
| Cached MBus Read ps | 142,130 |
| Cached Full Seq MBus Read ps | 11,400 |
| SDRAM Row Extra Latency ps | 48,860 |
| L1 Write Latency ps | 20,230 |
| L1 Preload-Write Latency ps | 2,560 |
| L2 Write Latency ps | 20,500 |
| L2 Preload-Write Latency ps | 2,650 |
| Cached MBus Write Lat ps | 19,690 |
| SDRAM Row Extra WriteLatency ps | 44,960 |
| Cached Full Seq MBus Write ps | 5,000 |
| Cached L1-Span MemCpy ps | 5,030 |
| Cached L2-Span MemCpy ps | 6,970 |
| Cached Memory-Span MemCpy ps | 12,630 |
| Uncached DW SS Read Lat ps | 330,280 |
| Uncached DW LS Read Lat ps | 330,380 |
| Uncached Stride LS Read Lat ps | 329,630 |
| Uncached DW SS Write Lat ps | 242,390 |
| Uncached DW LS Write Lat ps | 242,540 |
| Uncached Stride LS Write Lat ps | 241,740 |
| UnCached Memory-Span MemCpy ps | 285,990 |
| Cached=>UnCached MemCpy ps | 123,160 |
| UnCached=>Cached MemCpy ps | 123,160 |
| DDraw-ROSuf Seq Read Lat ps | 330,440 |
| DDraw-ROSuf Strided Read Lat ps | 329,990 |
| DDraw-RO to Cached MemCpy ps | 253,510 |
| DDraw-WO Seq Write Lat ps | 242,630 |
| DDraw-WO Strided Write Lat ps | 242,680 |
| DDraw-WO from Cached MemCpy ps | 123,650 |
| DDraw-RW Seq Read Lat ps | 330,360 |
| DDraw-RW Seq Write Lat ps | 242,550 |
| DDraw-RW memory-Span MemCpy ps | 371,700 |
| DDraw-RW to Cached MemCpy ps | 253,510 |
| DDraw-RW from Cached MemCpy ps | 123,730 |
| Overall Memory Score (GT 100) | 56 |

# CSV Output

The –c option outputs a lot of internal data into an Excel Command Delimited file (.csv) which can be used to understand particular results in more detail.

1. All the information sent to the .log file is also sent to the .csv file
2. For latency measurements, each of the 16 measurements at different offsets are shown. The Latencies are output in column format to make is easy to graph, but most latencies are pretty consistent and create pretty uninteresting graphs.
3. For Size measurements, the data is presented in at least two ways.
   1. The first is used when understanding why the measurements where assigned to Low, Mid, and High groups and is organized to be suitable for an Excel “Scatter with only Markers” chart. Just select columns D, E and F of the range of rows of interested including the header row and click on Insert\_tab->Chart-Scatter-with-only-Markers. The blue dots show latency and the red dots show the Low, Mid or High groups that they were assigned to. For example, the TLB measurements might look like the chart below where the vertical axis is in nanoseconds and the horizontal axis is Input parameter which is a scaled version of the varying parameter.
   2. The same of latencies are presented a second time, this time organized to more clearly show trends of the latencies as they vary with the input variation as well as exposed which level of caching or memory is being accessed predicted by an idealized cache model. Select columns I..AB and the relevant rows but do not select the header row. Insert\_tab->Charts->Line\_with\_Markers. For collections with only a small number of rows, transpose the chart by Select\_Data\_Source->Switch\_Row/Column.
      1. The vertical axis is again nanoseconds and the horizontal one is the input parameter. In the diagram below, I have selected the Y\_Strides column for the horizontal axis to have more meaningful labeling
      2. Series 1 is L1 Hit Rate % predicted by the cache model (Dark Blue).
      3. Series 2 is L2 Hit Rate % predicted by the cache model (Red).
      4. Series 3 is MB Hit Rate % as predicted by the cache model (Green).
      5. Series 4 is the percent of the accesses which cause a page walk (violet).
      6. Series 5 is the percent of accesses which cross a Row boundary (light blue).
      7. Series 6 to 21 are the 16 Latency measurements sorted from smallest to largest.

The example shown below is from “LineSize Stride Overview” which measures (64,128) to (8192,128) in steps of 64. So the left-most latencies representing hitting L1, the intermediate ones represent hitting L2 and right hand ones represent hitting SDRAM with a gradual transition between L2 and SDRAM.

1. The final type of data included shows the component weights and latencies used to calculate the score for both the device being tested and the idealized target device. The scoring will evolve to more accurately represent the memory access characteristics of key scenarios needed by Windows Mobile 7 .

Contact:

Sil Sanders

Principle Software Development Engineer

Windows Mobile

Microsoft Corporation

[SilS@microsoft.com](mailto:SilS@microsoft.com)

425-703-8426 (Office)

360-708-4665 (Cell)