Supplemental Information for "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rates in GPGPU"

Imran S. Haque[1]
Vijay S. Pande[1,2]
Stanford University, Departments of [1]Computer Science and [2]Chemistry

December 4, 2009

1 Design and Validation of MemtestG80

1.1 Offload and Parallelization Scheme

Several design parameters affect the sensitivity and speed of a software-based GPU memory tester. Specifically, the three components of the memory tester — pattern generation, memory access (writing and reading patterns to/from memory), and pattern verification — can be performed either by the CPU or the GPU itself; and if performed on the GPU, can be performed either serially or in parallel across the multiple GPU cores. The decisions made in MemtestG80 are informed both by the assumptions we make about the relative error rates of various system components and by responsiveness requirements dictated by operation on donated distributed-computing resources.

To improve the speed and responsiveness of the memory tester, all pattern generation, memory access, and verification is done in parallel on the GPU. We assume that the memory error rate is sufficiently low that the on-GPU code (which occupies a much smaller footprint than the tested region) will not be corrupted during the test execution. Performing verification on the GPU leaves the tester vulnerable to GPU logic errors. We therefore implicitly assume that the GPU logic error rate is lower than the GPU memory error rate; however, we verify this assumption by also running a custom logic test that should report errors on architectural paths similar to those used in other parts of the tester.

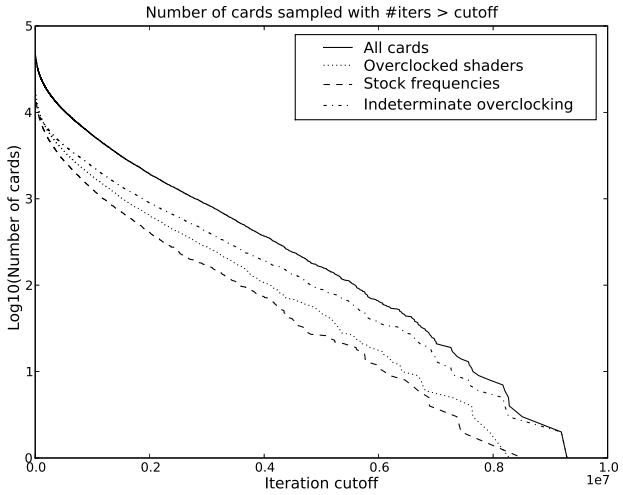
1.2 Tests implemented in MemtestG80

- Moving inversions (ones and zeros) (**MI10**) — Writes constant pattern of all zeros (0x0) or ones (0xFFFFFFFF) to all words in memory. Verifies all words.
- Moving inversions (random) (**MIR**) — Writes host-provided random word to all words in memory. Verifies all words.
- Memtest86 variant of walking 8-bit (1-byte) pattern (**1WM**) — Writes Memtest86 variant of walking 8-bit pattern to memory at each of 8 possible bit-shift-rotations. Verifies all words. Also tests complement of given pattern.
- True walking 8-bit (1-byte) pattern (zeros/ones) (**1W0, 1W1**) — Writes true walking-zeros or -ones pattern to memory with 8-bit width, at all 8 shifts. Verifies all words.
- True walking 32-bit (4-byte) pattern (zeros/ones) (**4W0, 4W1**) — As 1W0 and 1W1, but with 32-bit width and all 32 possible shifts.
- Random blocks (**RB**) — Generates and writes a different pseudorandom word to each word in memory. Verifies all words. Pseudorandom numbers generated using an on-GPU multi-core, multi-threaded leapfrogged Park-Miller Minimal Standard pseudorandom number generator [2].
- Modulo-20 (**M20**) — The modulo-20 test proceeds in 20 rounds. In round i, a 32-bit pattern is written to each memory location whose offset from the start of tested memory is equal to i modulo 20; the bitwise complement of the pattern is then written twice to every other memory location. Subsequently, the offsets equal to i modulo 20 are read back and verified.
- Logic test, one or four iterations through LCG cycle (**L, L4**) — Logic test, described in following section. LCG state kept in registers.
- Logic test in shared memory, one or four iterations through LCG cycle (**LS, LS4**) — Logic test, described in following section. LCG state kept in shared memory.

1.3 Logic Testing

Because results from the GPU can be passed back to the host CPU only by a copy from the GPU main memory, detection of GPU logic errors under the assumption that memory errors are more frequent than logic errors is nontrivial — an error in a computed result may be caused by an error in logic or memory. To overcome this problem, a test can be designed which produces the same expected result after varying amounts of logic operations. The same test can be run twice with (for example) four times the number of logic operations in the second execution. Since both tests write the same data to memory, the expected rate of errors due to memory faults will be equal between the two executions; since the latter test runs more logic operations, errors from logic faults should scale with the number of operations.

The design of our logic test, unique to MemtestG80, is based on the preceding principle. For the core calculation, we use a linear congruential random number generator (LCG) with a short period k starting from zero. Such a generator, when started from zero, will return to zero after a fixed number of iterations k. Because the generator only reaches k states, of 2^{32} possible (in the 32-bit generator), assuming a uniform probability of error over bits, most logic errors will cause the generator to transition to a state outside the normal operation cycle. Such a state is unlikely to return to zero in the correct number of steps, and therefore whether the generator returns to zero is a good indication of whether a logic error occurred. Our logic test starts the generator from zero and runs it for k or $4k$ cycles, each time writing the results out to memory, reading it back, and verifying



Supplementary Figure 2: Number of cards tested as a function of minimum number of iterations completed. Breakdowns by overclocking status.

Card Family	# cards $\geq$ 300,000 iter.
Consumer graphics cards	*17648 total*
GeForce GTX	5520
GeForce 8800	5478
GeForce 9800/GTS	4923
GeForce 9600	1516
Other Desktop GeForce	181
Mobile GeForce	30
Professional graphics cards	*89 total*
Quadro FX	83
Quadroplex 2200	6
Dedicated GPGPU cards	*37 total*
Tesla T10	27
Tesla C1060	10

Supplementary Table 1: Distribution of cards tested on FAH that ran at least 300,000 test iterations

that it contains only zeros. Any nonzero values indicate either the presence of a logic or a memory error. Scaling of the number of nonzero values with the number of LCG iterations indicates logic, rather than memory, errors. The use of constant zero as the test pattern further increases the sensitivity of the test to logic rather than memory errors; as we show in Section **??**, the constant zero pattern is insensitive to faulty memory.

1.4 Validation

Supplementary Figure 1 shows word-error-rate in 20 iterations of MemtestG80 as a function of memory clock frequency (normal memory clock rate = 400 MHz).

2 Methodology

Supplementary Table 1 shows a breakdown of cards that completed at least 300,000 iterations of MemtestG80 on Folding@home, separated by card family.

Supplementary Figure 2 plots the distribution of the number of cards we tested, as a function of the minimum number of iterations each card completed. It also plots the same trace broken down into cards which were (logic) overclocked, at or below rated frequencies, or had an unknown overclocking status.

3 Analysis

3.1 Hypothesis testing by information gain

To test our hypotheses we apply the information-theoretic measure known as *information gain*, which is broadly used in data mining as a heuristic criterion for building decision tree models of data [3]. The hypothesis testing problem is formulated as follows: given a labeled dataset D, we partition D according to an indicator variable V into multiple subsets $D_1, D_2, \cdots, D_{|V|}$. We would like to know how good V is at explaining the variability in D.

We measure the "variability" of D and each of its subsets by their respective Shannon entropies, $H(D)$, defined as

$$H(D) = -\sum_{x \in D} p(x) \log_2 (p(x))$$

The information gain on D from V, $I(D;V)$ (also known as the mutual information between D and V), is defined as:

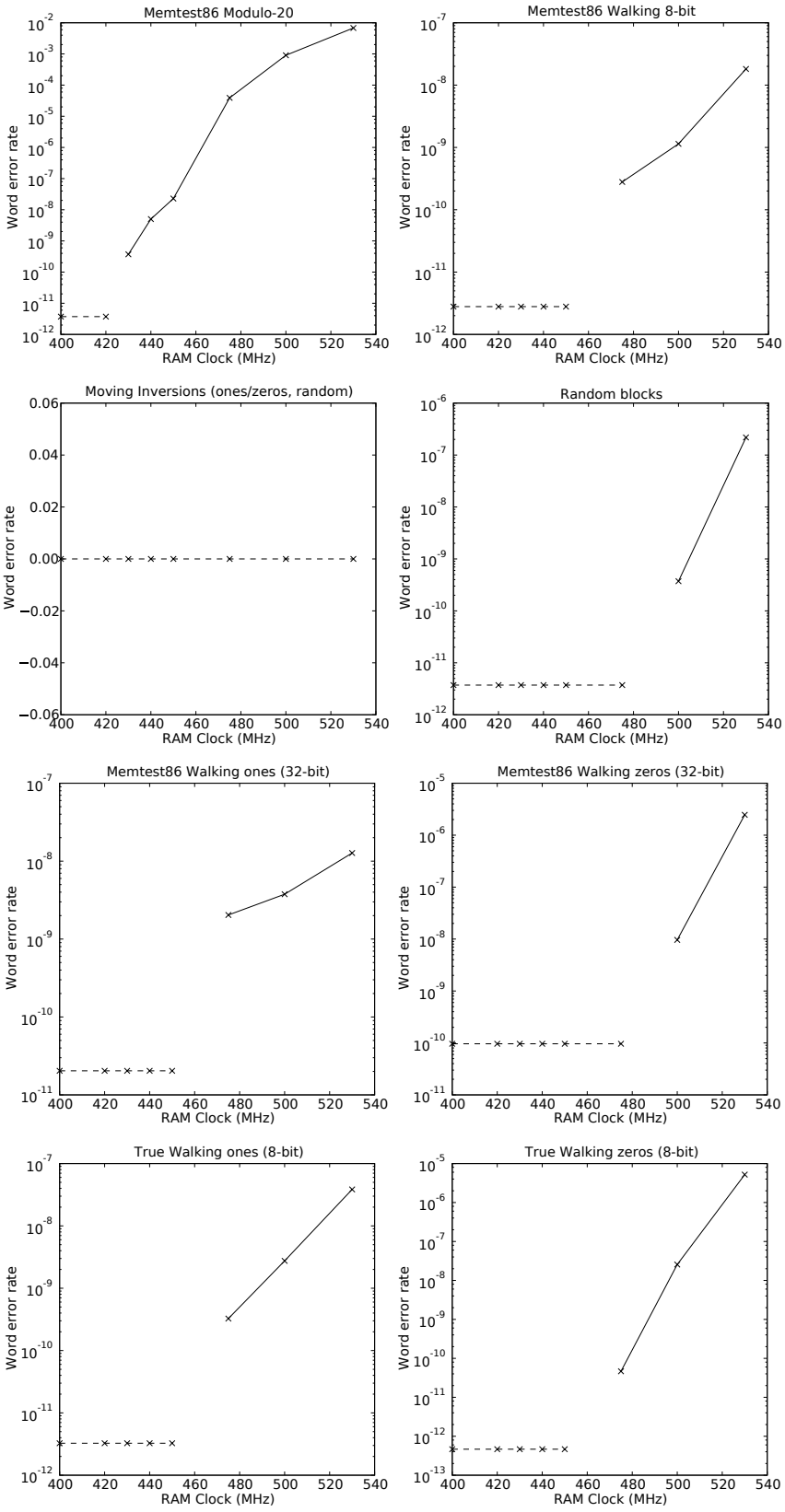
$$\begin{aligned} I(D;V) &= H(D) - H(D|V) \\ I(D;V) &= H(D) - \sum_{v \in V} H(D_v) P(V = v) \end{aligned}$$

If $I(D;V)$ is large compared to $H(D)$, then V explains a significant portion of the distribution of D.

To estimate probability distributions D in our hypothesis testing, we histogrammed failure probabilities on a per-card basis as was done for each distribution in Figure **??**, but across the entire range of probabilities from 0 to 1. Although this resolution is too high for the low number of counts at higher probabilities, most of these bins will be zero-valued and will not affect the entropy calculations.

3.2 Card identification

For analyses requiring identification of individual card models (the overclocking, architecture, and consumer/professional board splits), we excluded any Folding@home client IDs which corresponded to more



Supplementary Figure 1: Positive control: word error rate versus frequency on sample of MemtestG80 tests. Dashed lines represent zero errors found.

than one returned board name in our dataset. This was done to reduce the probability that multiple boards were conflated into one. For some work units, no board name was returned. Client IDs with missing data were also excluded, as it was impossible to identify what kind of board ran that test.

There was one exception to the above procedure - if a client ID mapped only to either a "Tesla C1060" or a "Tesla T10 Processor" on different work units, we did not exclude that client ID, as these are likely to be the same board, possibly with a revised driver.

3.3 Failure modes of tests

By examining the mutual information between the results of each individual test comprising a MemtestG80 iteration, it is possible to better understand the mechanisms triggering failures under various conditions. For each test, we construct a list in which each element corresponds to a single execution of MemtestG80, and the value of each element is the number of failures on that test for that execution. Corresponding elements in each vector map to the same MemtestG80 execution. Each list of failure counts was then histogrammed into 10 bins and normalized to build an empirical probability mass function for the number of failures in that test on a given execution of MemtestG80. Using these probability distributions for tests X and Y we calculated the entropies $H(X)$ and $H(Y)$ according to the formulas in Supplementary Section 3.1; in this case we use an alternative (equivalent) formulation for the mutual information $I(X;Y)$:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

The entropy $H(X)$ can be interpreted as the uncertainty in X, as measured by the number of bits required by an optimal code to specify a value from the distribution $p_X(x)$. The mutual information $I(X;Y)$ can be interpreted as the reduction in uncertainty in X caused by knowledge of the value of Y, or vice versa (mutual information is symmetric) [1]. Supplementary Figure 3 shows the ratio of $I(X;Y)$ to $H(X)$ for all tests X and Y used in MemtestG80; this ratio is the fraction of the uncertainty in X explained by knowledge of Y. In Supplementary Figure 3, the Y (the "explaining" distributions) are along the rows; the X (the "explained" distributions) are along the columns. We use the codes defined in Supplementary Section 1.2.

Several interesting trends emerge from this data:

1. **The Modulo-20 test stands on its own**
   Both the M20 column and the M20 row have small values across their lengths, indicating the Modulo-20 test covaried strongly with no other test. This is likely due to the Modulo-20 test's increased sensitivity relative to other tests and reinforces the notion that it probes a different failure mechanism than do other tests.

2. **The Random Blocks test is a good logic test**
   Although it was not intended as a logic test, the large values in the RB row for the columns corresponding to the LCG-based logic tests indicate that RB does a good job of capturing the errors measured by the LCG tests. Conversely, the small values in the RB column for the LCG tests demonstrate that RB is measuring a superset of errors relative to the LCG tests. This result is reasonable in retrospect: the RB test is very shader-logic intensive. We have designed it around a multithreaded, multicore Park-Miller Minimal Standard pseudorandom number generator [2], which in the course of generating a new random number for each memory location performs many more logic operations than any other MemtestG80 test.

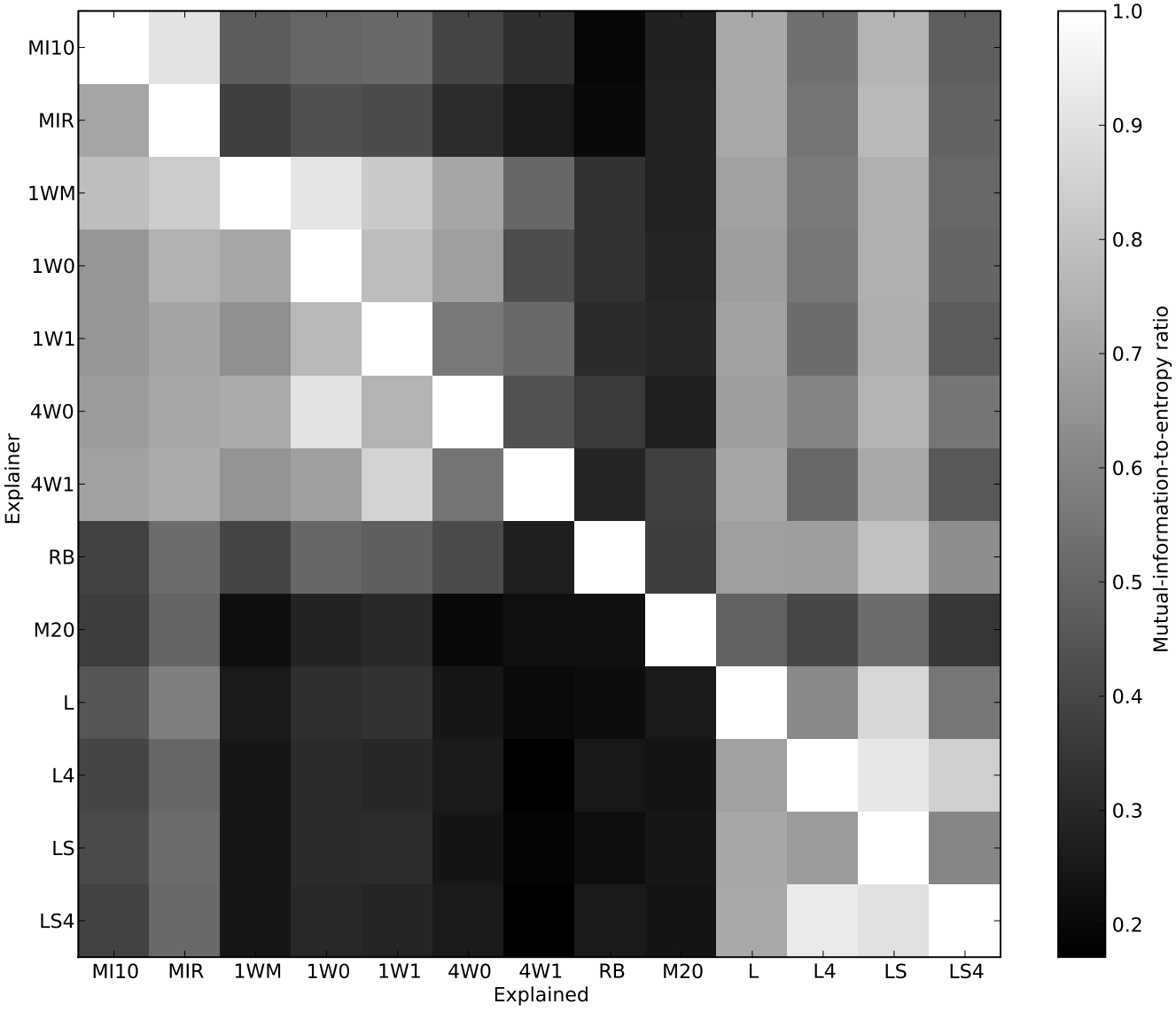
3. **The logic tests measure a distinct failure mode from most memory tests**
   The four-iteration variants of the logic test (L4 and LS4) are poorly explained by most memory tests, and in particular, are less-well-explained by the memory tests than are their one-iteration counterparts (L and LS). This is to be expected, as the one-iteration variants are more influenced by memory errors. However, the bright block in the bottom-right of the mutual information plot shows that the logic tests covary strongly among themselves. Furthermore, memory tests have higher mutual information to the L4 test than the LS4 test, indicating that the use of shared memory in the logic test is a significant variable. Together, these results show that the logic tests detect a failure mode distinct from that tested in the memory tests, and that apparent logic errors can be triggered by soft errors in the on-GPU shared memory.

References

[1] T. M. Cover and J. A. Thomas. *Elements of Information Theory, 2nd Edition*. Wiley-Interscience, July 2006.

[2] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, October 1988.

[3] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition*. Morgan Kaufmann, June 2005.



Supplementary Figure 3: Mutual information-to-entropy ratios for each test pair. Each entry is the fraction of the entropy of the test in that column explained by the test in that row. Brighter squares indicate that more of the variance of the explained test is explained by the explainer test. Test codes defined in Section 3.3.