Using Cache Decay for Managing Replacement in a Shared Cache

Gila Engel
Touro College

Professor Margaret Martonosi
Department of Electrical Engineering
Princeton University
mrm@ee.princeton.edu

1 Introduction

1.1 Problem Overview

In cache memory, there are usually multiple addresses in RAM that map to the same cache line. While this problem is alleviated slightly by the caches being an n-way set associative cache, it is not a complete fix. To accommodate all of the blocks of data that need to be brought into the cache, the processor manages replacement of the cache line. One of the primary methods used is to replace the least recently used (LRU) cache line. However, this is not always optimal. At times, the data that is evicted is more important than the evicting data.

Another issue arises when a processor uses simultaneous multithreading (SMT) [1]. With many different threads competing for cache memory at the same time, yet each running its own set of instructions, one thread that is more aggressive may end up grabbing a very large portion of the cache by evicting the data belonging to the other threads. If this aggressive thread is minimally important compared to the other threads running, the more important threads that should ideally have a larger, or at least equal, portion of the cache will be left with a really small one instead, or possibly evicted completely.

1.2 Previous Related Work

Cache Decay is the concept of a cache line "decaying", or becoming invalid for a specified reason, time related or otherwise. We will be focusing on the time related issue. Work has been done using cache decay to decrease static leakage. Many cache lines are accessed a few times as soon as they are brought in to the cache, and then remain inactive for a considerably larger amount of time than they were active, until they are evicted, and this wastes a lot of static power. By marking a line as decayed once it has not been active for a certain amount of time (determined by the decay interval), it becomes invalid and the power can be blocked from the decayed line, conserving all of that electrical energy. [2]

1.3 Goals

Cache Decay can be taken further, and implemented with regard to simultaneous multithreading. We propose to utilize cache decay to improve cache sharing policies for managing cache replacements in chip multiprocessors.

We envision a processor in which instead of evicting the cache line that was least recently used, a cache line that was decayed will be replaced instead. Each processor will have a different decay interval, and when the cache lines that had been brought in by that

To Top

processor have not been accessed for an amount of time equal to, or exceeding its interval, they decay. Using this, the more important processes can have a higher decay interval, which will prevent their cache lines from being evicted as frequently as others with a lower decay interval.

1.4 Contribution

Our research shows an average of a 3% performance improvement due to using cache decay for managing replacement of cache lines.

2. Methodology and Modeling

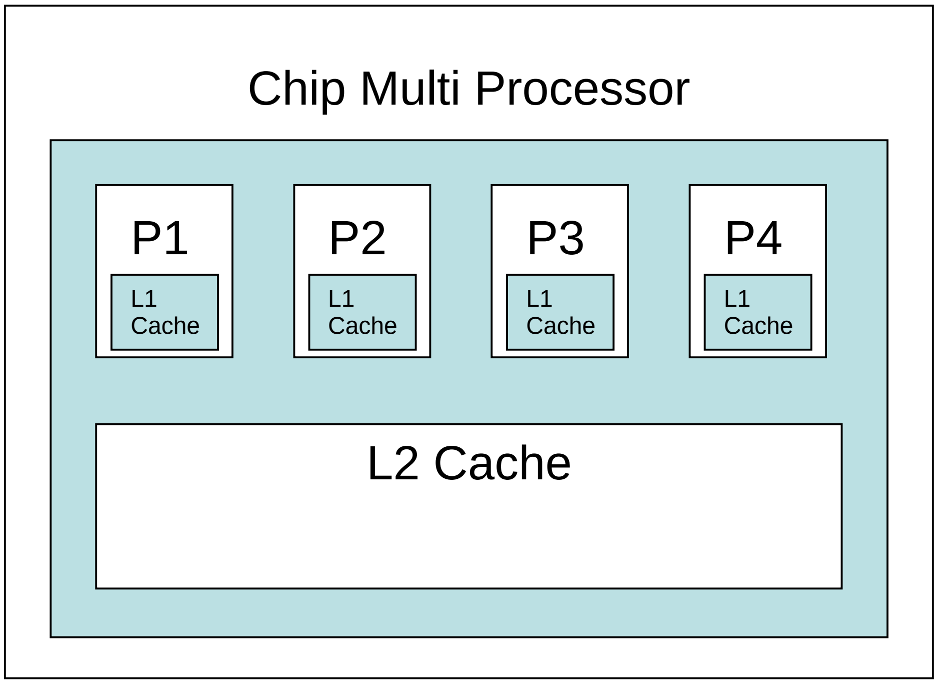


Figure 1. Diagram of 2-level cache hierarchy on a Chip Multi Processor.

2.1 Simulator

We used the simulator Parallel Turandot CMP (PTCMP) [3], which models the performance and power of a multi-core PowerPC™ processor, with a two way SMT core. It is programmed with POSIX threads to attain lightweight synchronization and parallel speedup. PTCMP can test various combinations of CMP and SMT configurations without limits on the number of cores.

We simulate a chip multiprocessor with 1 to 8 cores and a 2-level cache hierarchy. Both levels of cache reside on the chip. Each core has its own L1 cache, and they share the L2 cache. The L1 Cache is a 2-way set associative cache, with 256 indices, and a block size of 128 bytes. The L2 Cache is a 4-way set associative cache with 16,384 indices and a block size of 128 bytes. We will be focusing on managing the replacement in the L2 Cache. See Figure 1.

2.2 Benchmarks

We used benchmarks from the SPEC CPU2000 [4].

To Top

3 Preliminary Work

3.1 Overview

We began working with basic cache decay in the L2 cache to see how the performance improved by decaying cache lines with a general decay interval, regardless of which core had brought in the data.

3.2 General Cache Decay in the L2 Cache

As a processor processes the instructions of a program, it brings in blocks of data at once, placing it into the cache for easier access, which decreases the overhead of retrieving the consecutive bytes of data. The next time the processor looks for that data, it will already be on chip. However, if that data has not been accessed within a specified period of time which is determined by parameter files, it is considered decayed, and becomes invalid. The processor then deals with the failed retrieval as a regular miss, as if that data had never been brought into the cache.

3.3 Results

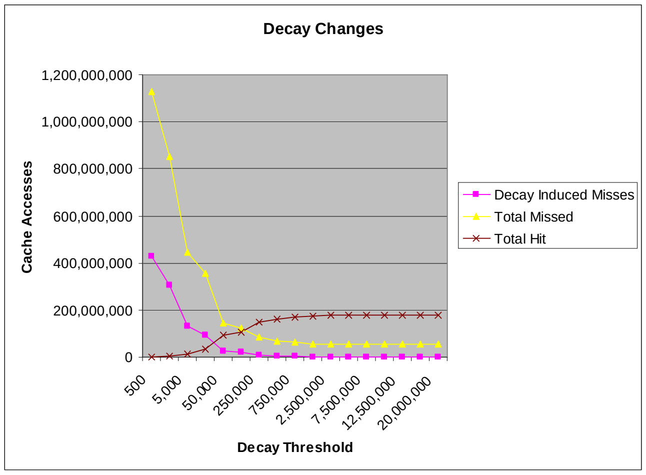


Figure 2. The effect on the amount of Decay Induced Misses, Total Misses, and Total Hits as the Decay Interval increases

3.3.1 Varying Decay Interval

We created a workload simulating an 8 core processor, each running 2 threads, with each thread running a minimum of 100,000,000 instructions. We ran the workload 9 times with a different threshold interval each time, between 500 and 20,000,000, to see the impact on the decay induced misses, total misses and total hits as the decay threshold increased. The conclusion was that as the decay interval increased, the amount of decay induced misses and total misses decreased, while the amount of hits increased. See Figure 2.

To Top

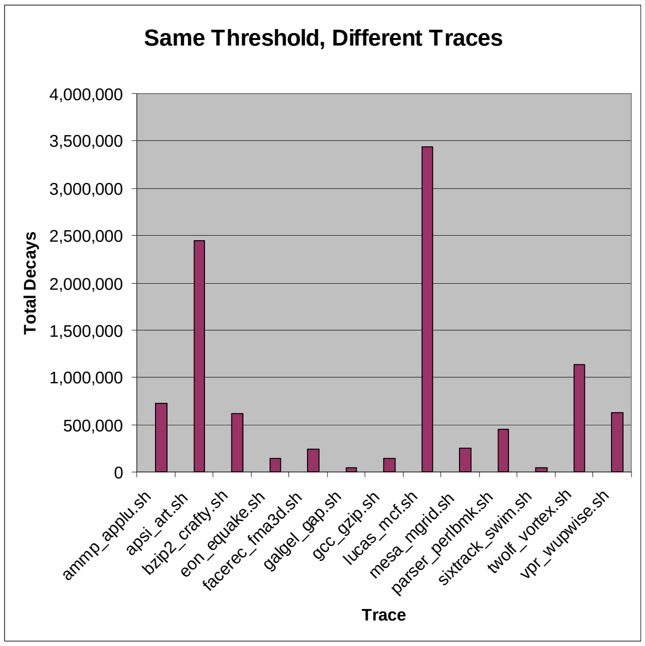


Figure 3. The comparison between different amounts of decay induced misses, depending on the different benchmarks used.

3.3.2 Varying Benchmarks

We created another 13 workloads, each simulating a 1 core processor, and running 2 threads, with each thread running a minimum of 70,000,000 instructions. Each workload ran using the same decay interval (an interval of 250,000), to see how varied the amount of decays could be, depending on the different benchmarks used. From these workloads, there was a large amount of diversity with regard to the amount of decays, from approximately 41,000 to 3,437,000. See Figure 3.

4 Managing Replacement with Cache Decay

4.1 Our Addition

We envision a processor in which cache decay is used to enhance managing replacement in the L2 cache. Previously, a popular method has been to replace the LRU. In our processor, we will try to replace only cache lines that have decayed. If there are multiple decayed lines, we use the LRU of those, and if there are no decayed lines, we replace the LRU for that index. Another enhancement of our processor is that each core is allotted its own decay interval. Each of the cache lines brought in by that core will be decayed, or become invalid, only after it has not been accessed within the amount of time delineated by that core's decay interval.

4.2 Workloads

We considered 15 different workloads, each using 3 cores, each core running 2 threads, with each thread running a minimum of 70,000,000 instructions. Each of the threads per workload ran the same benchmark. We ran each workload twice, once with decay and once without.

To Top

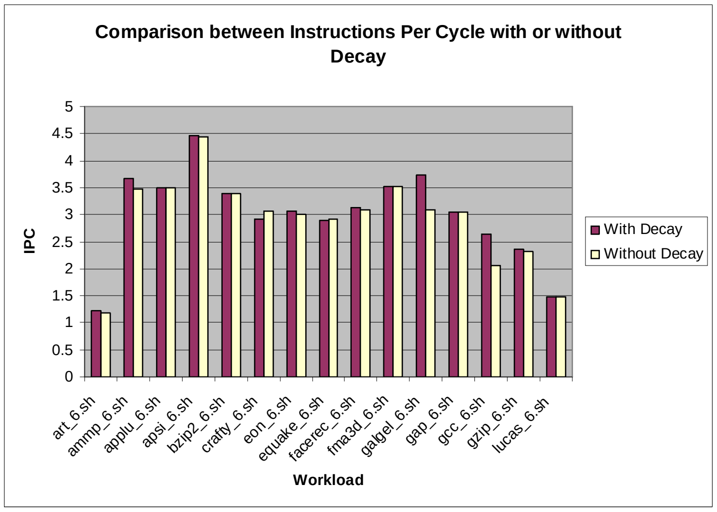


Figure 4. The comparisons of IPC for each workload.

4.3 Results

Upon comparing the IPC for each workload of those run with decay against those run without, we concluded that the effect of using cache decay for managing cache replacement, on average, improves performance slightly. Out of the 15 traces that we ran, 66% of them had a higher IPC when using cache decay than not, with an average of a 3.07% IPC increase. See Figure 4.

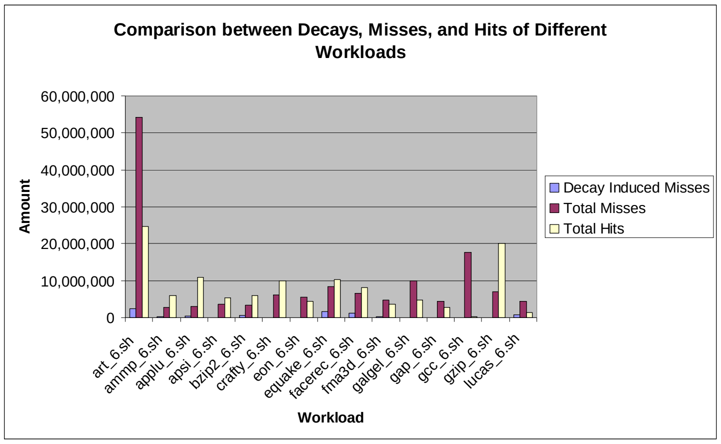


Figure 5. The effect of using different benchmarks on decay induced misses, total misses, and total hits.

The second comparison we looked at was how different workloads reacted differently with regard to decay induced misses, total misses, and total hits despite their having the same decay interval of 500,000. See Figure 5.

2.4.1 More Graphs

Refer to Appendix 1 for the graphs of all of the results of the workloads.

3 Conclusion

For our programs that we ran there was a modest performance improvement when using cache decay for managing replacement in shared caches. For much larger

To Top

programs, we imagine that there could be significant benefits from implementing cache decay to enhance managing cache replacement. That is something that is open to future scientific research.

References

[1] D. M. Tullsen, S. J. Eggers, H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In ISCA, June 1995

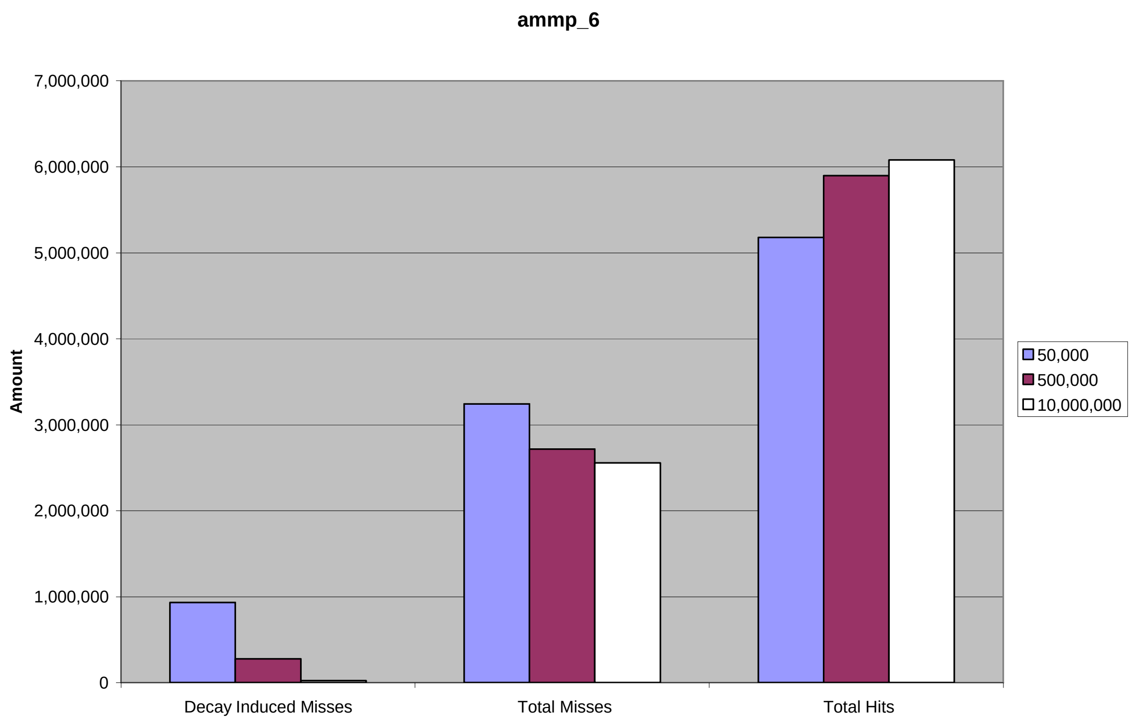
[2] S. Kaxiras, Z. Hu, M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In ISCA, 2001

[3] J. Donald, M. Martonosi. Power Efficiency for Variation-Tolerant MultiCore Processors. In ISPLED, 2006

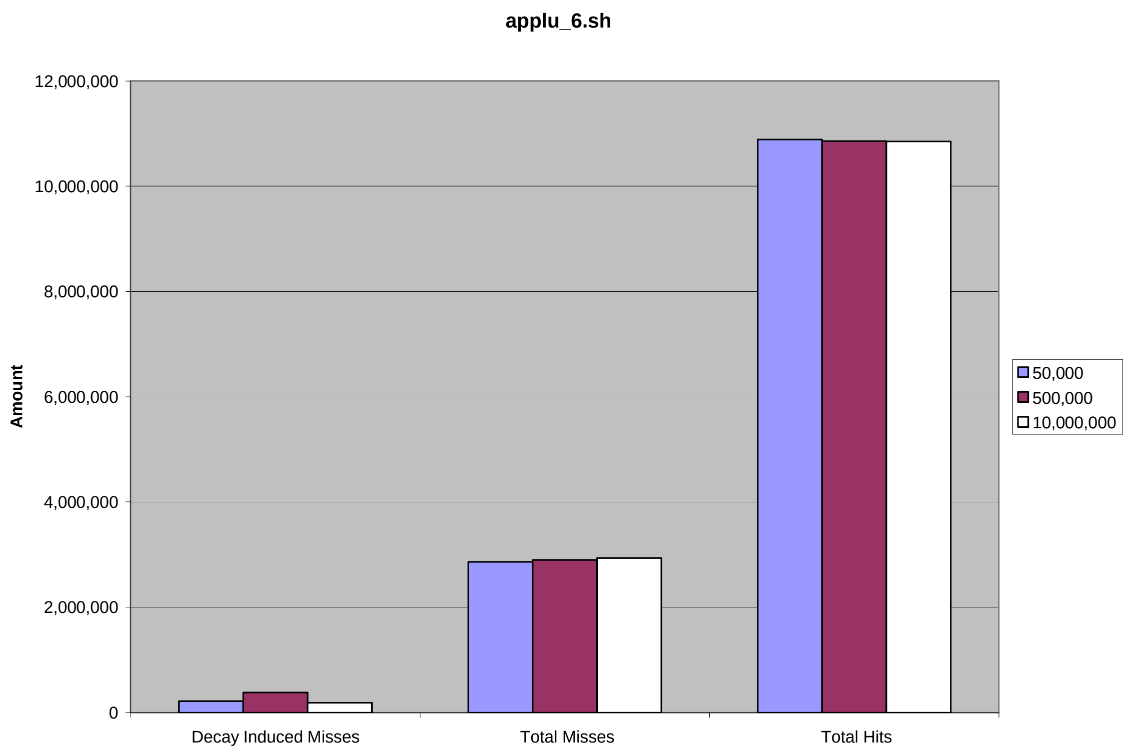
[4] The Standard Performance Evaluation Corporation. WWW Site. http://www.spec.org, Dec. 2000

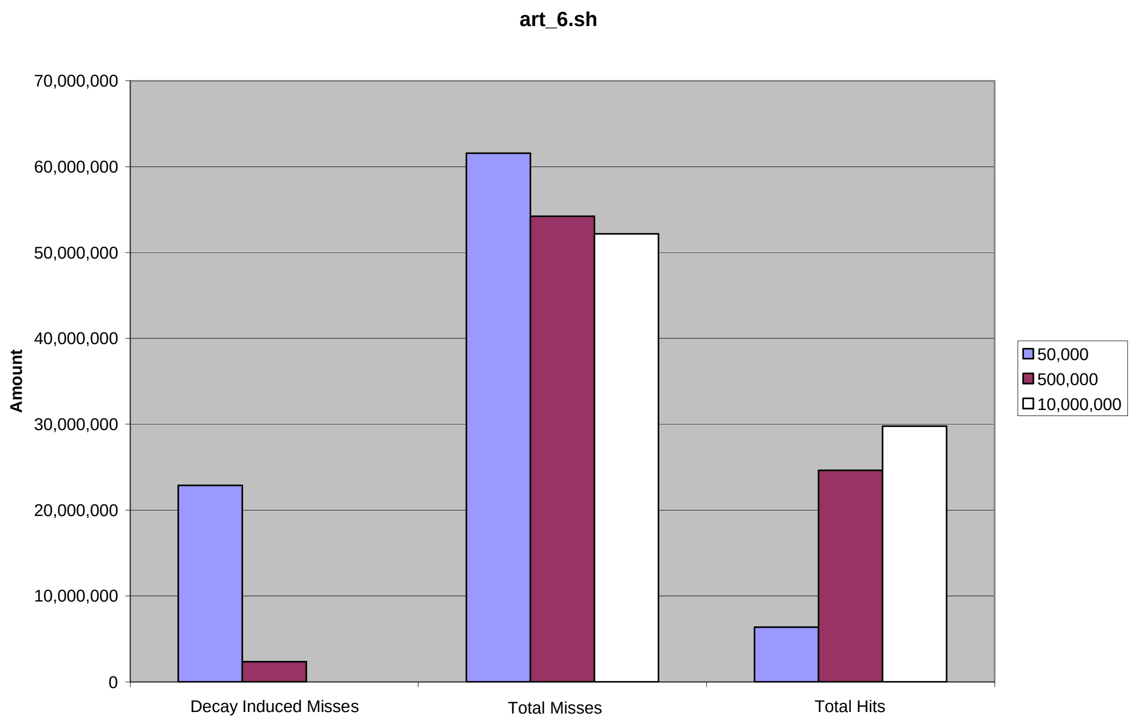
Appendix 1

Following are graphs, each depicting the results of a simulation of a 3 core processor, with each core running 2 threads. All of the threads per workload ran the same SPEC CPU2000 benchmark. Each core had a different decay threshold, one of 50,000, one of 500,000 and one of 10,000,000.

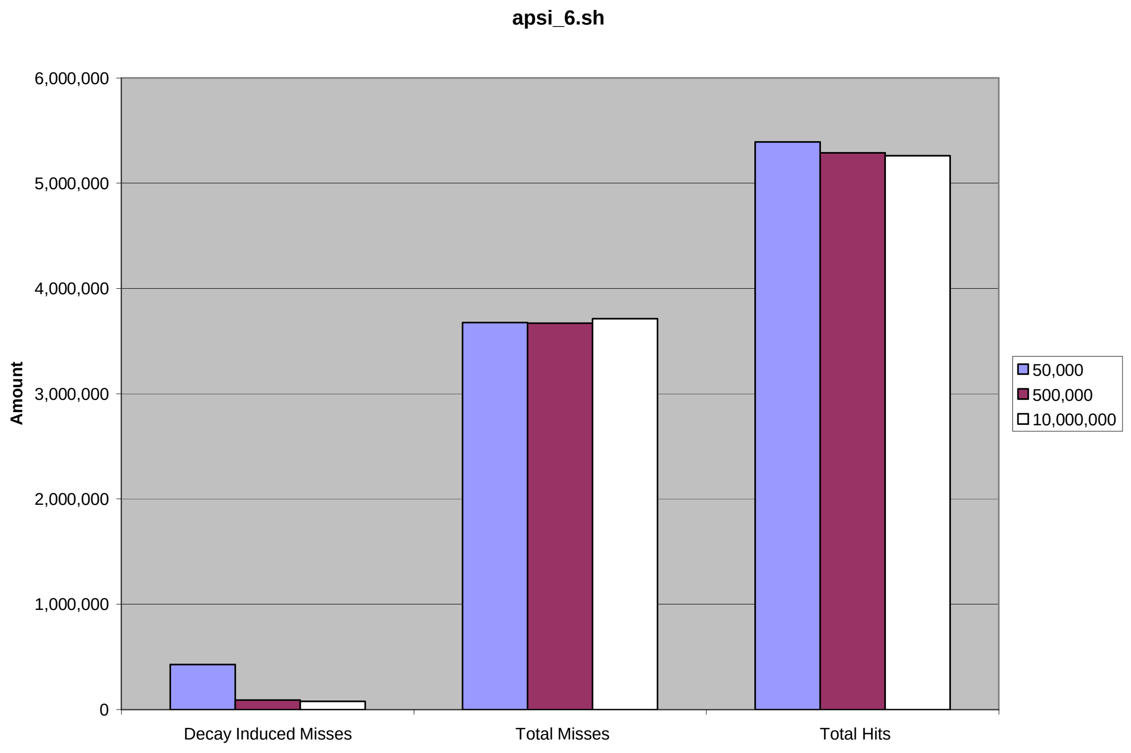


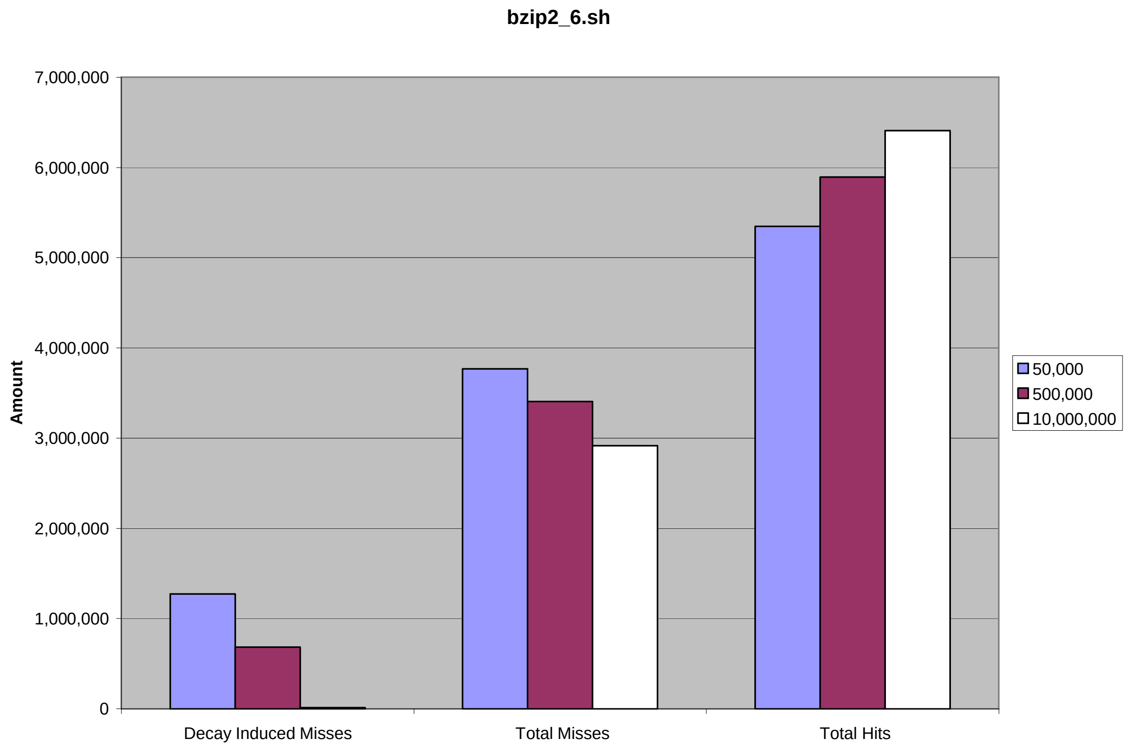
To Top


applu_6.sh
Amount
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

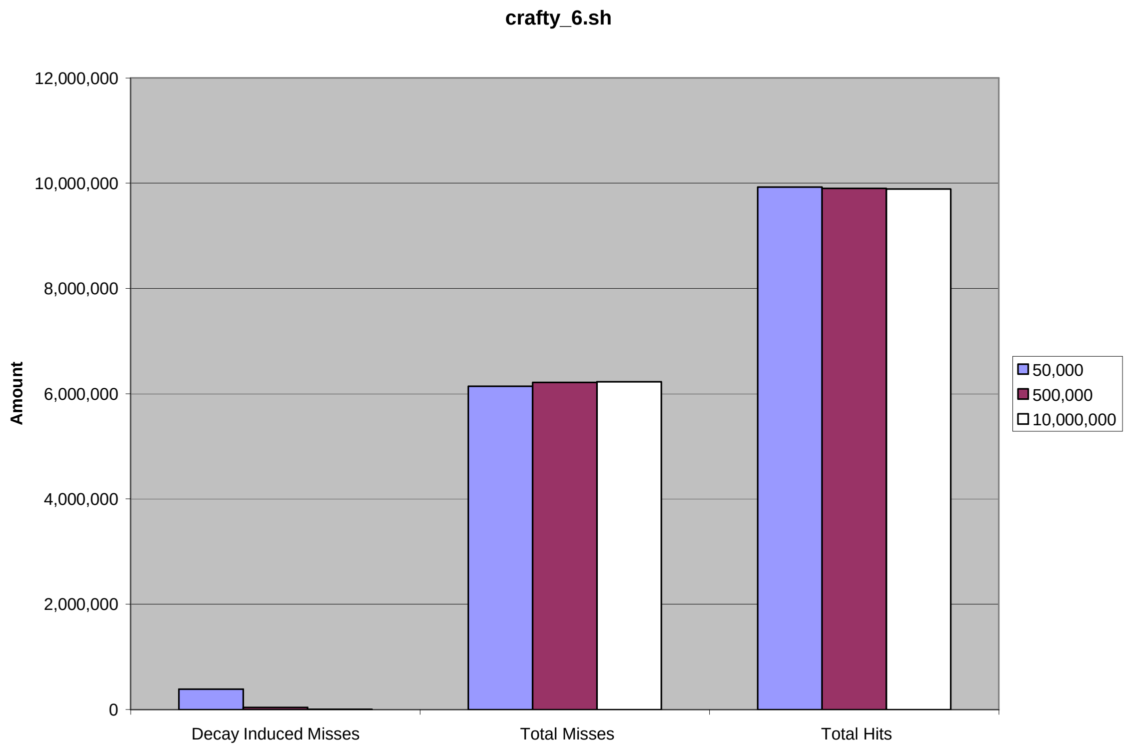

art_6.sh
Amount
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

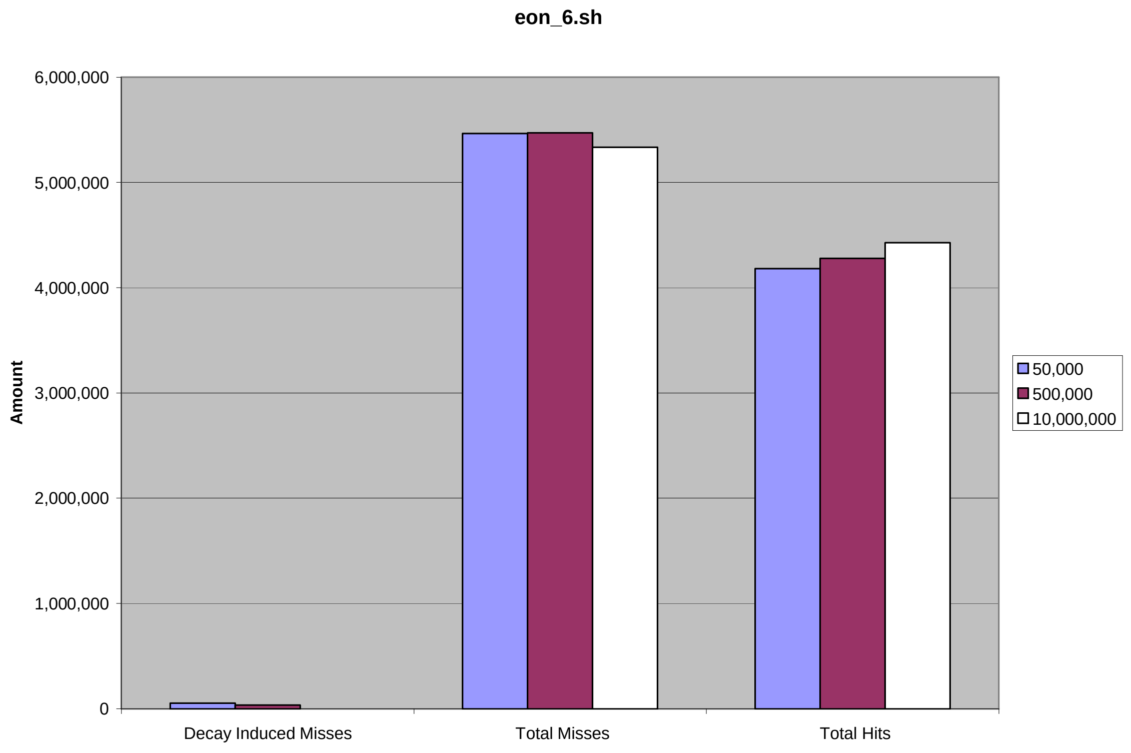
To Top


apsi_6.sh
Amount
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

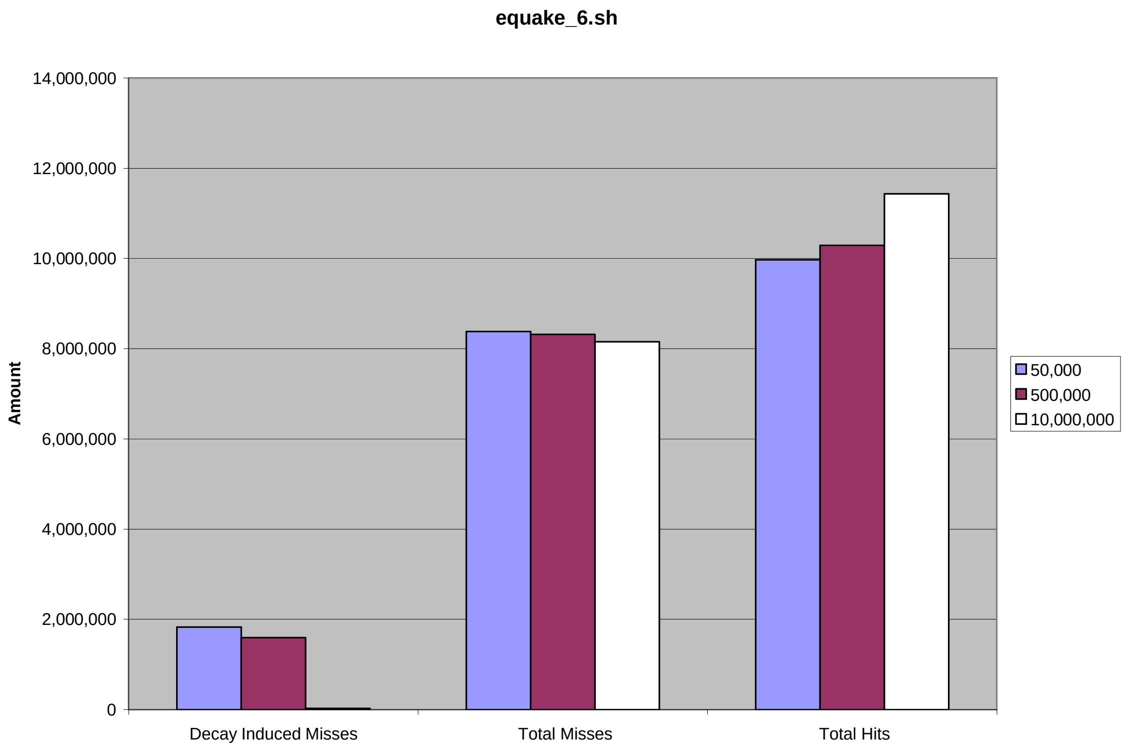

bzip2_6.sh
Amount
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

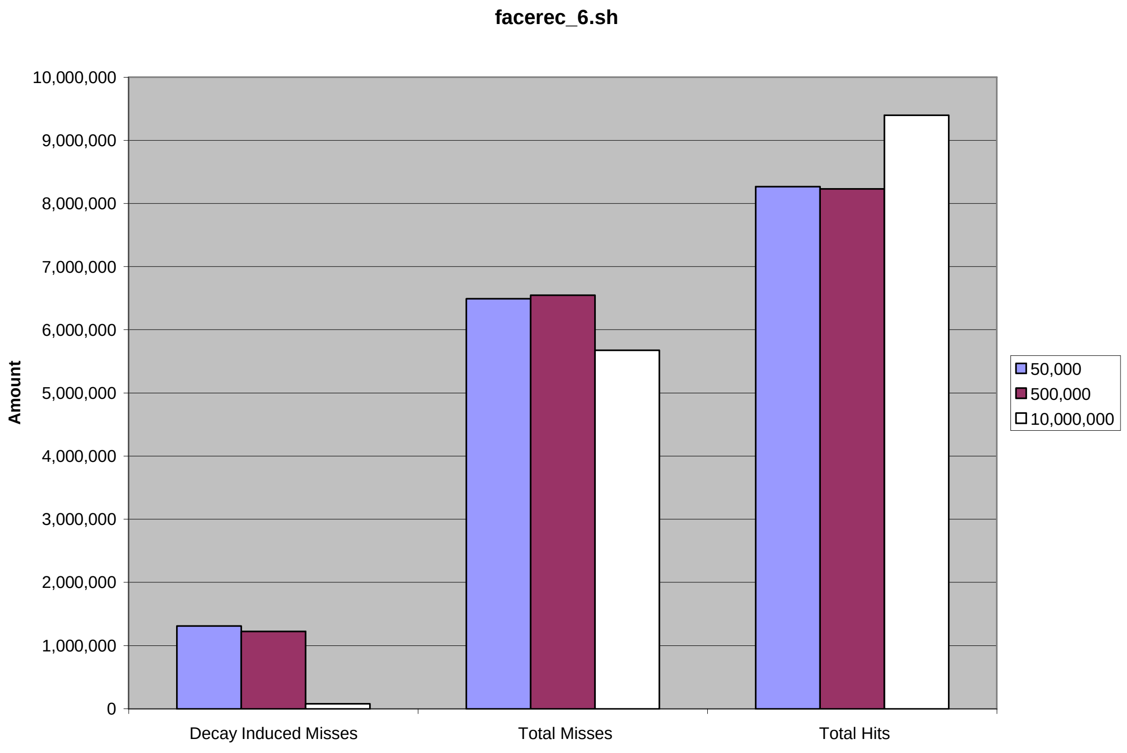
To Top


crafty_6.sh
Amount
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

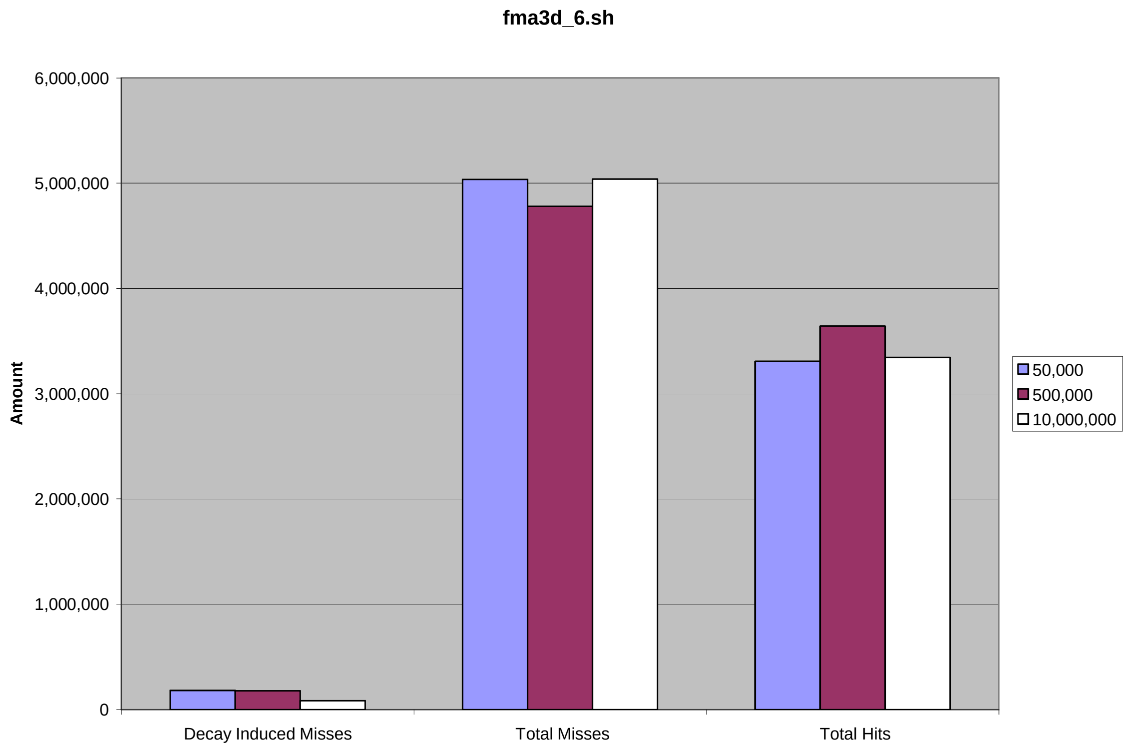

eon_6.sh
Amount
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

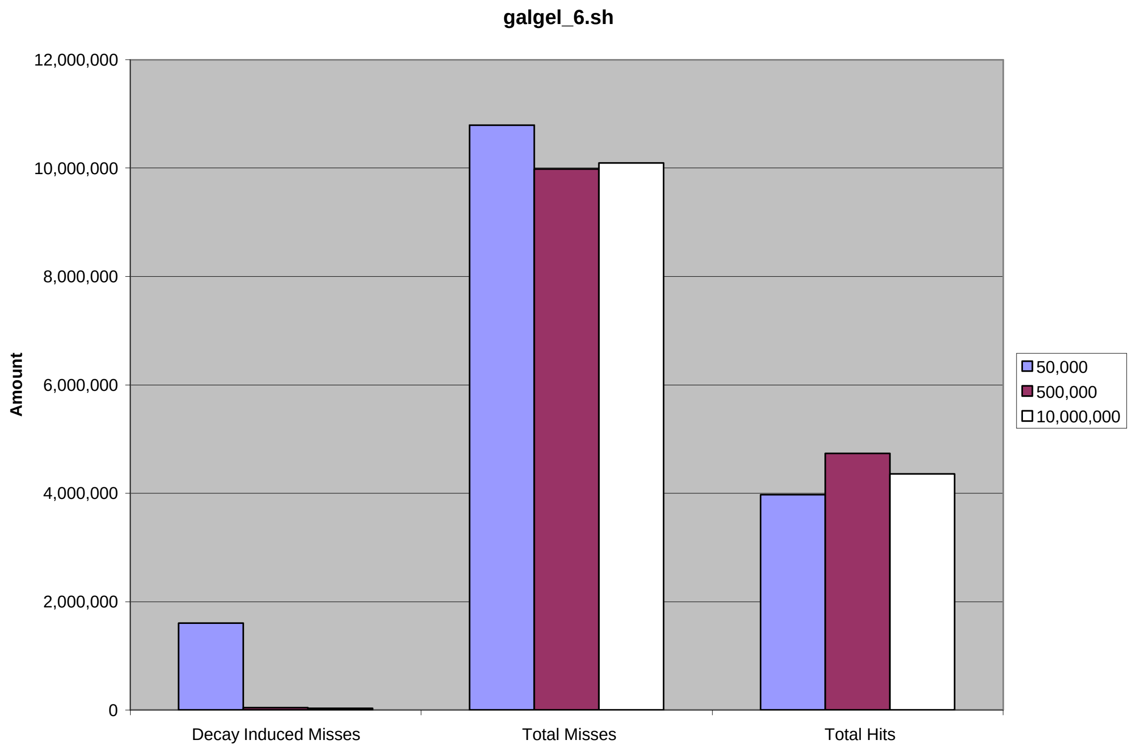
To Top


equake_6.sh
Amount
0
2,000,000
4,000,000
6,000,000
8,000,000
10,000,000
12,000,000
14,000,000
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

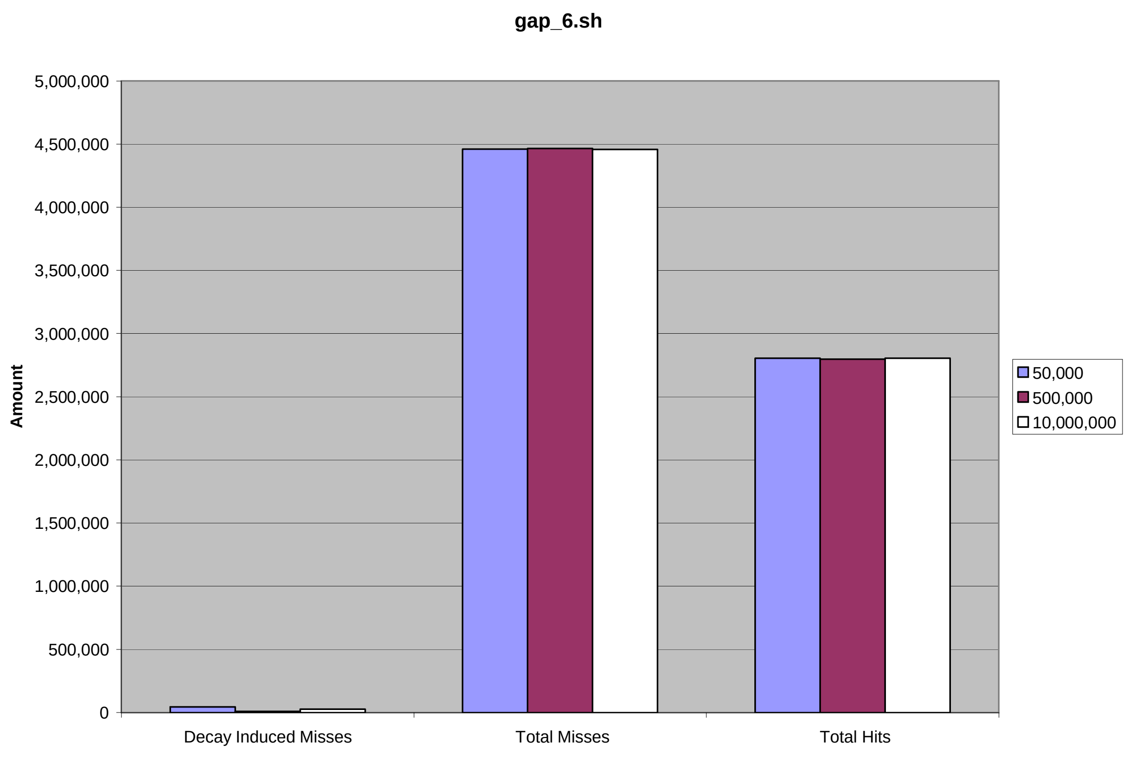

facerec_6.sh
Amount
0
1,000,000
2,000,000
3,000,000
4,000,000
5,000,000
6,000,000
7,000,000
8,000,000
9,000,000
10,000,000
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

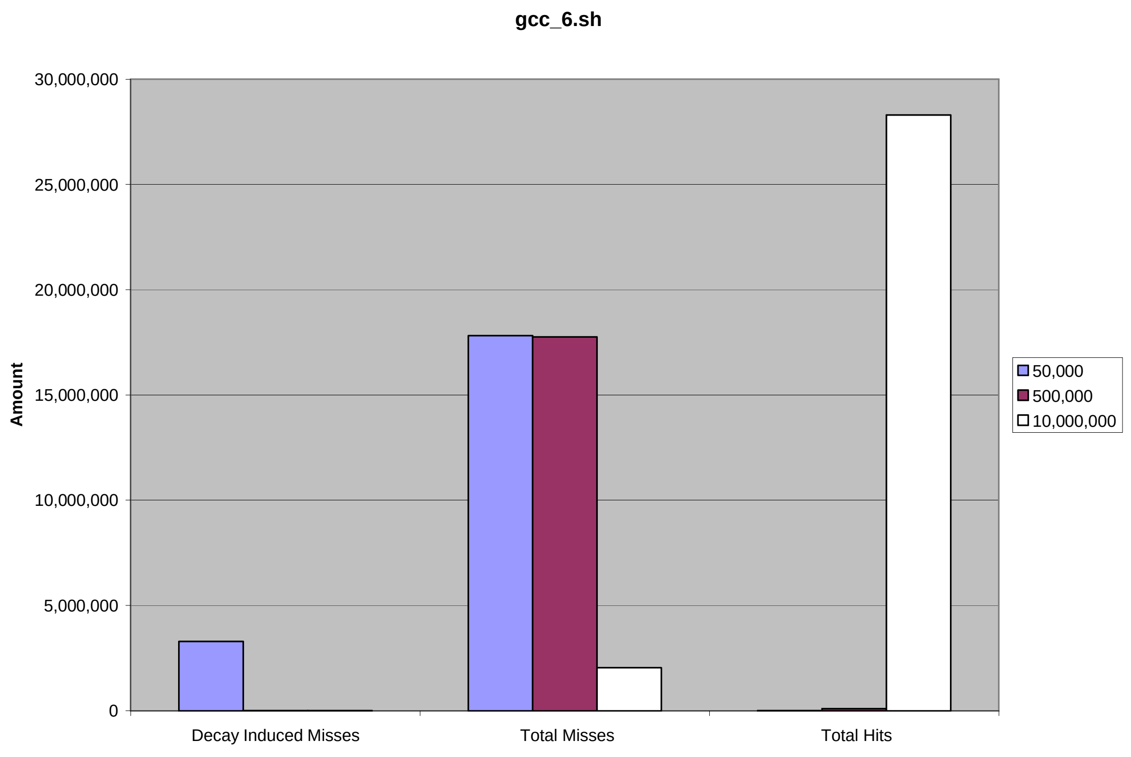
To Top


fma3d_6.sh
Amount
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

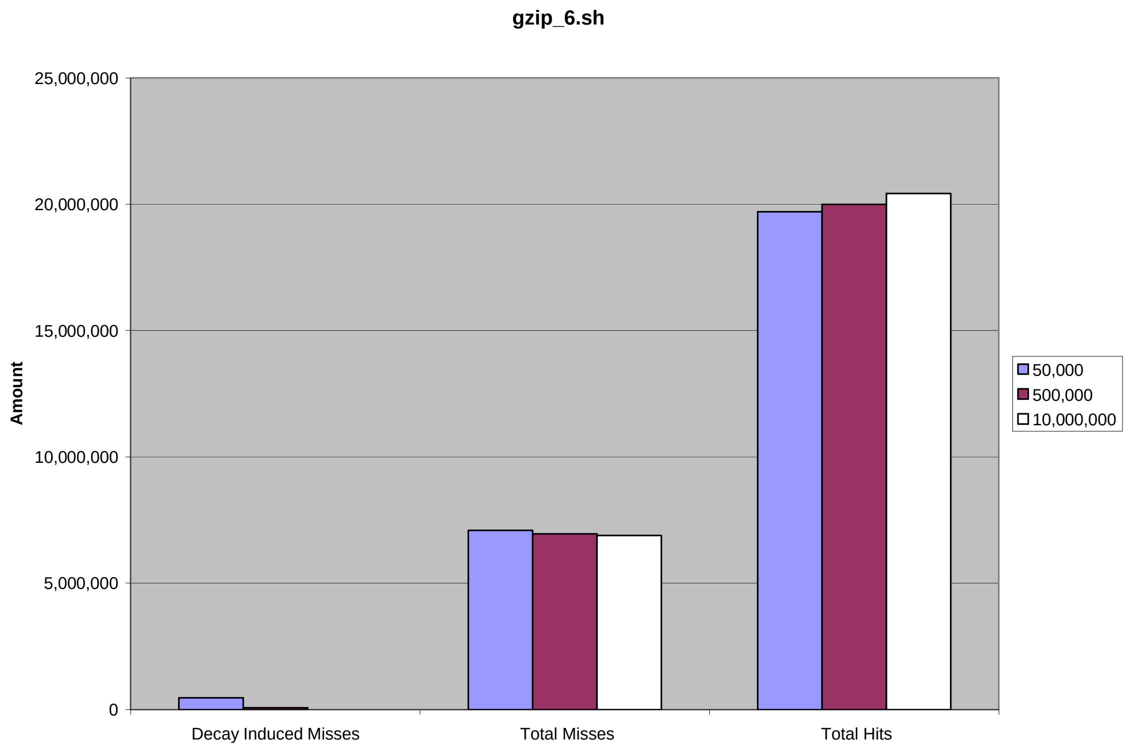

galgel_6.sh
Amount
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

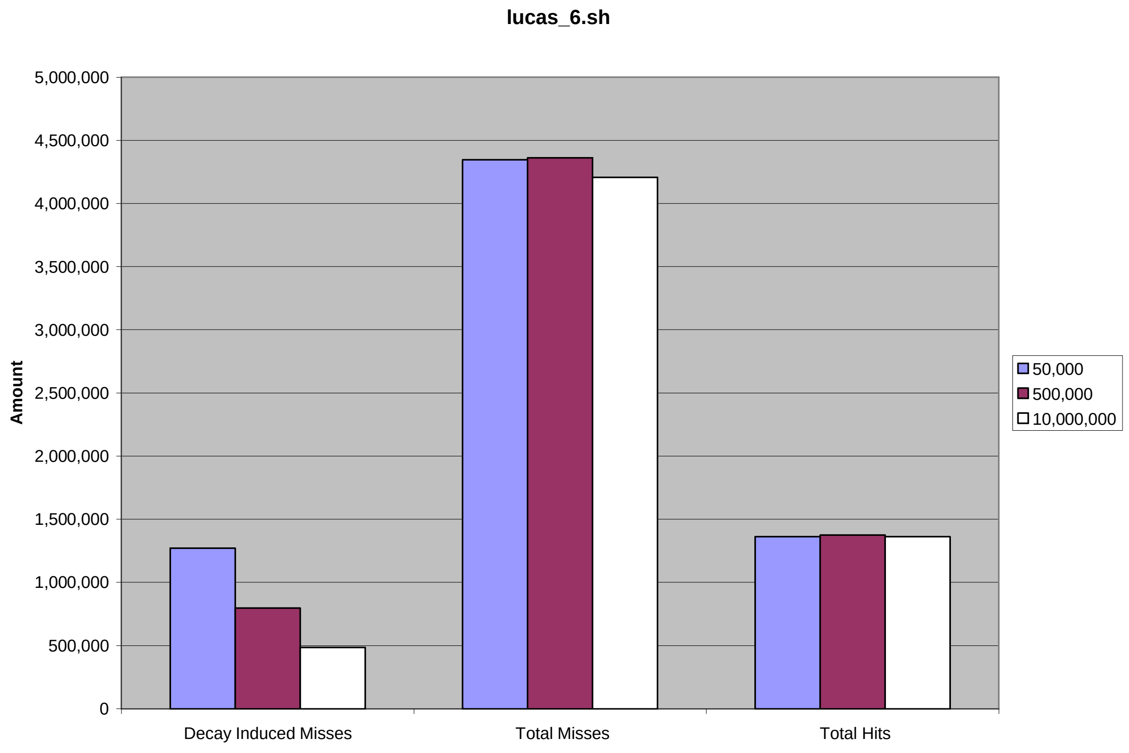
To Top


gap_6.sh
Amount
5,000,000
4,500,000
4,000,000
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000


gcc_6.sh
Amount
30,000,000
25,000,000
20,000,000
15,000,000
10,000,000
5,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

To Top


gzip_6.sh
Amount
25,000,000
20,000,000
15,000,000
10,000,000
5,000,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000


lucas_6.sh
Amount
5,000,000
4,500,000
4,000,000
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
Decay Induced Misses
Total Misses
Total Hits
50,000
500,000
10,000,000

To Top