6.823: Lab 2 Questions

Nada Amin
namin@mit.edu

Due: 20 October 2008

1. The hit rate increases with higher associativity and higher capacity. The three cache models agree highly. The number of rows most dramatically affect the hit rate, while the associativity and block size affect it minorly.

   I am not surprised that the three cache models agree, because the difference is just in the translation, and shouldn't affect the hit rate much.

   It is obvious why the hit rate increases with increasing associativity and increasing number of rows as we are in effect using a bigger cache. The hit rate increases with increasing block sizes, because we don't have to fetch the same page multiple times.

   I further explore the more interesting trade-off between number of rows and block size at the same cache capacity. The sweet spot on the graph is with 2^6 number of rows and block sizes of 2^5 bytes.

2. The working set for a SPEC benchmark (used gap for INT and swim for FLOAT) is very large and doesn't fit in a cache. The working set is not supposed to fit in a cache, because then it wouldn't test all the relevant parts of the architecture like the replacement policy.

   To calculate the working set, I study the graph, looking for a value of $\log(r)$ after which the hit rate stagnates. Then the working set is $r \cdot b$ where b the block size (here, 4 bytes). I estimate r to be 24, because the FLOAT program has a sudden significant increase at the tail.

   The INT program seems to benefit more steadily from increasing the number of rows. For this reason, the FLOAT program might benefit more from a higher associativity. Thus, I would choose a cache configuration with more associativity in the FLOAT program than in the INT program.

3. The physically indexed, virtually tagged cache model doesn't make sense because, if we get the physical page number for indexing, we might as well use it later for tagging. Getting the physical page number takes time, and if we use it for indexing, this cost is incurred upfront. Once we get to the tagging phase, there is no advantage of using a virtual tag if we can use a physical tag.

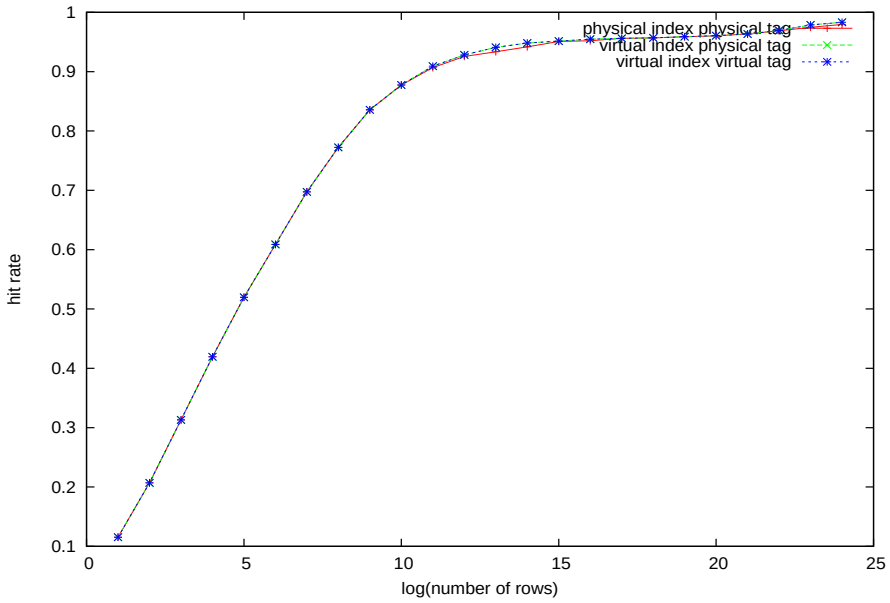

Figure 1: Varying the number of rows for SPEC gap. Block size is 4 bytes, associativity is 1.

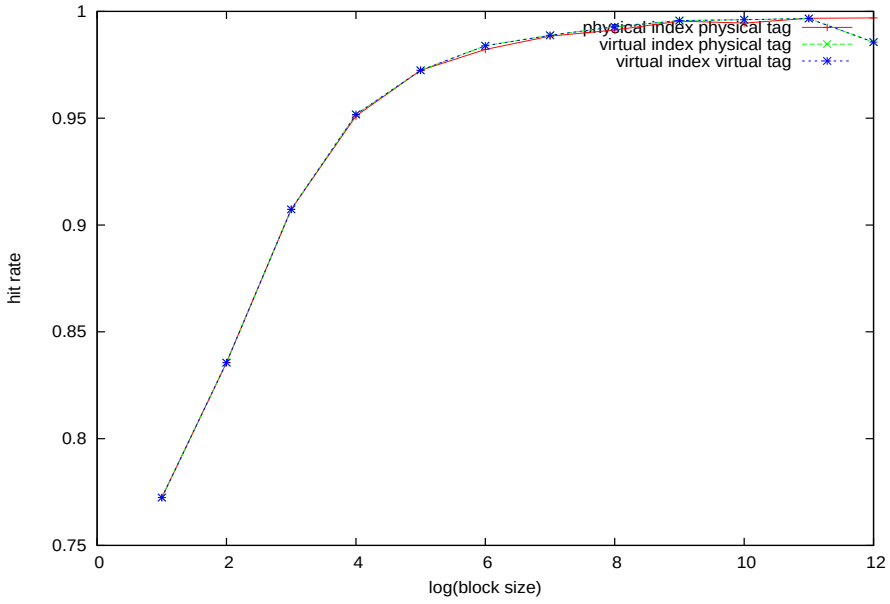

Figure 2: Varying the block size for SPEC gap. Number of rows is 512, associativity is 1.

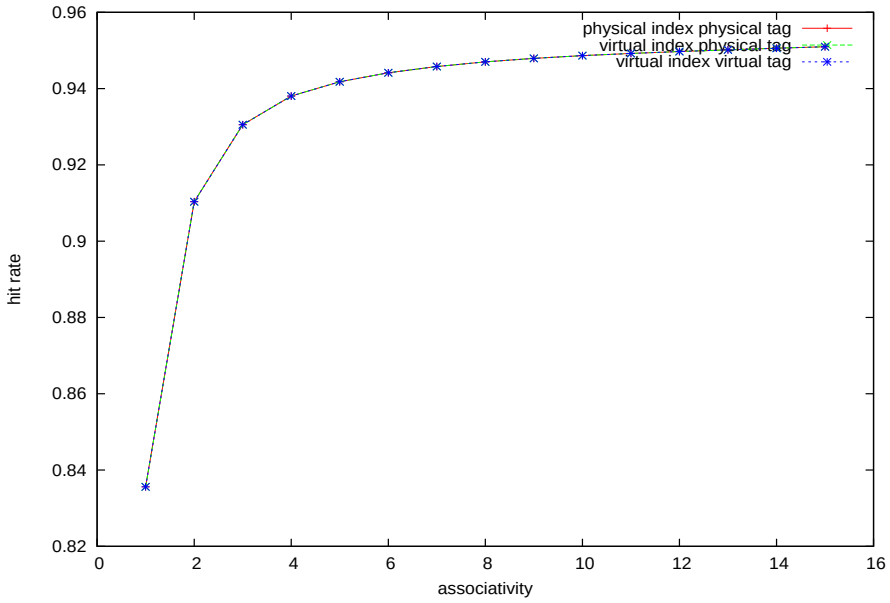

Figure 3: Varying the associativity for SPEC gap. Number of rows is 512, block size is 4 bytes.

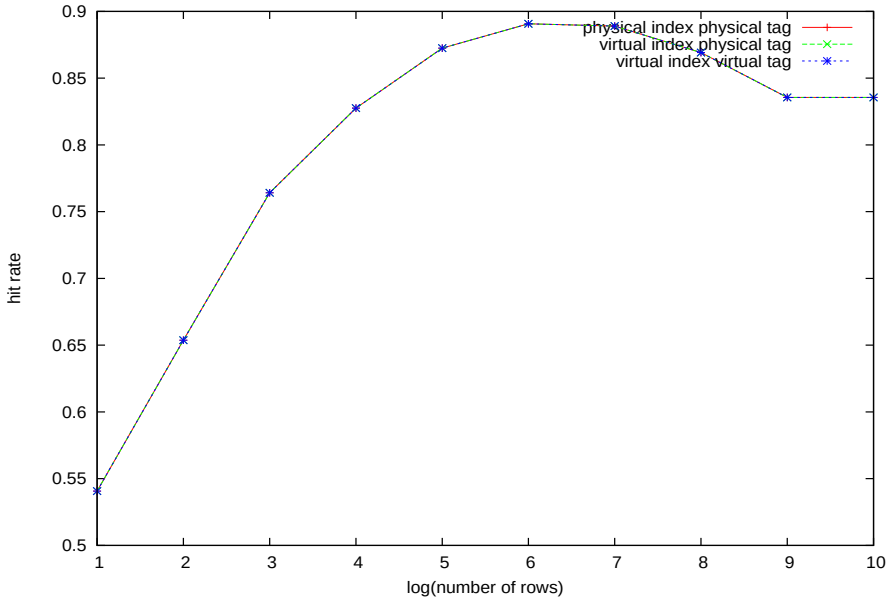

Figure 4: Varying the number of rows for SPEC gap while fixing the cache capacity. Cache capacity is 2^{11}. Associativity is 1.

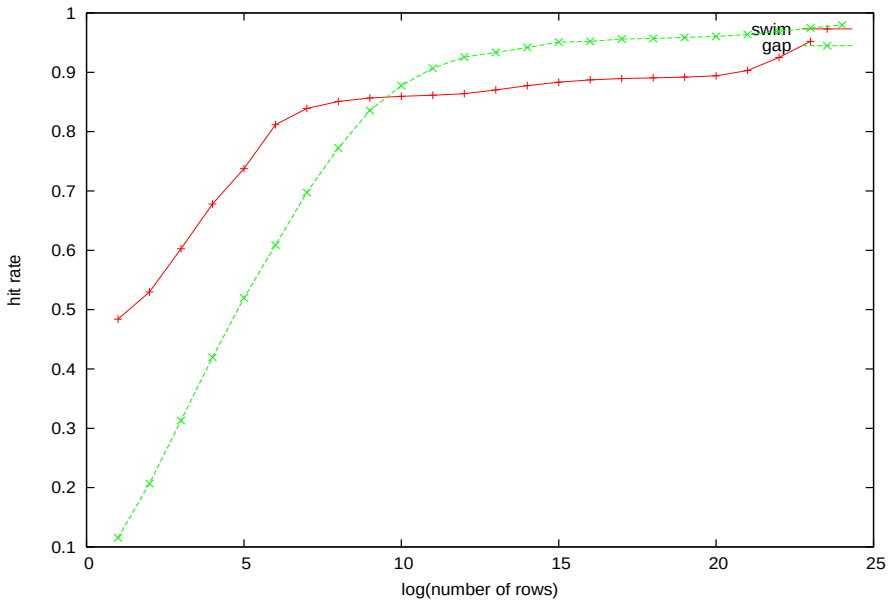

Figure 5: Comparing varying the number of rows for SPEC gap (INT) and swim (FLOAT). Block size is 4 bytes, associativity is 1.

4. Different processes will have different mappings from virtual to physical addresses. Thus, it is possible for the same virtual address to point to different physical addresses in different processes. Thus, using a virtual tag might lead us to hit the wrong data.

   For the physically indexed, physically tagged cache model, no modifications are necessary. For the virtually indexed, physically tagged cache model, we might have multiple copies of the same physical data lying around in the case where two different processes access the same physical data with two different virtual addresses. This is bad, because if one updates the data in its cache slot, the other might not see it. As explained above, the virtually tagged cache model is even worse, because we might hit the wrong data. One fool-proof albeit inefficient modification is to simply flush the cache when switching processes. In the virtually indexed, physically tagged cache model, we could instead have a physically indexed structure where we keep track of which process last wrote some physical data. Then, when reading some data, we check that the current process is the one who has written it last in order to return a hit.

   For the virtually indexed, virtually tagged cache model, given a page size of 256 bytes and a cache size of 512 bytes, some cache configurations work. The trick is for the index to be independent of the virtual page number. In addition, it is necessary to tag each cache entry with a process number along with the virtual tag. The cache configurations that work

in this model are those with $\log(r) + \log(b) \leq \log(p) = 8$, where r is the number of rows, b is the block size and p is the page size. Thus, in order to use the cache to the fullest, we need at least a 2-way cache, so that $abr = 512$, where a is the number of ways of the cache. For 2-way caches, we can vary $\log(r)$ from 1 to 7 and let $\log(b) = 8 - \log(r)$. More generally, for a-way caches, we can vary $\log(r)$ from 1 to $8 - \log(a)$, and let $\log(b) = 9 - \log(a) - \log(r)$.

5. I simply created a pin tool which counts the number of unaligned memory accesses and all memory accesses. The percentage of unaligned memory accesses on the benchmark vary from 0.2% to 15%. Our assumption makes sense, because most of the time, we will stay within a block. In the worst-case benchmark, the probability that we need to read more than one block of size 2^5 bytes for one memory access is $\frac{0.15}{2^5} = 0.4\%$.

   In any case, it makes sense that the frequency of unaligned memory accesses would be low, since the memory accesses are usually arranged by high-level programs to be word-aligned.