CS 5341

Oscar Galindo

**Modeling Cache Performance Beyond LRU**

This paper begins by offering some background on high-performance replacement policies among them LRU. The intention for this background information is to specify important aspects about cache, among them that cache at “high” levels (i.e., closer to the chip) like L1 and L2 do not utilize as “sophisticated” cache policies as the LLC (i.e., Last-Level Cache). Furthermore, the paper mentions that “Traditional replacement policies (e.g., LRU, pseudo-LRU, least-frequently used, or random) all perform poorly at the LLC because private caches filter out most locality”. The paper then refers to three principles that rule the development of replacement policies, these principles are recency (prioritizing recently used lines over old), protection (offering protection against the deletion of parts of working-set memory from cache), and classification (branding lines based on utilization like “reused” or “not reused”). Policies that follow the previously mentioned principles are DIP, DRRIP, SHiP, SDBP, etc.

As the paper mentions, the objective of this work is to propose a new methodology which allows to map reuse distances distributions to miss rates. The work shown in this paper, proposed to achieve the goal of mapping reuse distances distributions to miss rates by constructing a so-called “model”/cache model. In summary the algorithm takes as input the “cache architecture (its size, associativity, and replacement policy) and a concise description of the access stream [described as reuse distance distribution]”. With the input taken the algorithm then produces cache’s hit and eviction distributions. Internally, though, the algorithm utilizes the hit/eviction distributions and the age distribution of cached lines. By outputting the distributions for hit and eviction of the cache, the model allows to predict, indirectly, the performance of cache to then adapt the policies of cache to respond better to the system’s needs.

The paper then proceeds to demonstrate how the model can be constructed to predict LRU replacements. Under this “simple” model it is explained the model must solve three equations – age, hit, and eviction - for every possible “age” of cache line to obtain the distributions that model LRU. Among the three distributions (i.e., age, hit, and eviction) the hardest to obtain is the eviction distribution. The paper then proceeds to define the needed changes to support other, more complex replacement policies. The paper offers the modification of eviction equations that construct the eviction distribution by including raking functions (i.e., functions that rank eviction priority based on age) and by generalizing the eviction distributions.

Then, the paper offers the implementation of the needed code to solve the system of equations needed to construct the model. Since the model implementation relies on convergence between simulations, the paper describes the possibility of sectioning the range of possible ages in detailed or more general simulations to obtain probabilities for hit, eviction, and age. In addition, the paper offers details of implementation like overheads and application profiling.

Lastly, the paper presents the results of applying the cache model to different benchmarks, in general the results of this experimentations is that the size of sampling distance (N) is quite relevant and there seems to be a sweet spot of 128. In addition, it seems that the model can outperform other replacement policies with a low degree of complexity in the computations while keeping the errors “low”. This suggests that the model is successful at controlling the replacement policy of the LLC. The paper also shows improvement on the LLC partioning over state-of-the-art cache partioning policies.

In general, I believe the paper proposes a very ingenious mechanism to control the replacement policy at the LLC. But as in any other algorithm that relies on random sampling, partitioning and statistical analysis there is always room for fatal error. I believe the methodology could be helpful to speed up architectures with low amounts of cache, where probably consistency is not a super main concern, but for very extreme execution patterns like those of a supercomputer this methodology would be highly unsafe.