1) Names and contact information (email) of all members of the group

Joshua Linge: Jlinge@knights.ucf.edu

Anjana Mishra: anjana.mishra0@knights.ucf.edu

Basundhara Dey: dey.basundhara@knights.ucf.edu

2) Project Title

Increasing Cache Efficiency by Dead Block Prediction and Elimination

3) Project description.

- What are you investigating?

We are going to investigate on improving Cache efficiency by predicting and eliminating dead blocks after a **Cache Burst**. A Cache burst begins when a block in cache become most recently used (MRU) and ends up when the same becomes non-MRU. As a formal definition-"A cache burst is the contiguous group of cache accesses a block receives while it is in the MRU position of its cache set with no intervening references to any other block in the same set."

Due to the presence of dead blocks in the cache for a long period of time with no use, before the eviction, the cache efficiency decreases. So, other than increasing the capacity of cache, we can increase the number of live blocks instead of dead ones for better performance.

So in this project, we will work on **dead block predictors (DBP)**. The earlier we identify a dead block, there will be more opportunity to improve the cache efficiency.

With addition to that, we will also keep an eye on best tradeoff between **timeliness and prediction accuracy/coverage**, by making dead block predictions at different points during the dead time of a block.

Moreover, Cache bursts only work well at the **L1 cache**. For the **L2 cache**, counting-based predictors work best. We also consider **RefCount+** (helps in handling confidence bit and current history store) predictor.

- How will you be investigating it?

  Upon the last access to a block with the LRU replacement, multiple replacements to that set must occur before the dead block is evicted. This may occur after several accesses of cycles, or immediately upon the first time the block is loaded into the cache.

  Considering this fact, a dead block should be identified between the last access to the block and final eviction from the cache. This identification (better called prediction) can be done both using software and hardware. But, in software, there will be less accuracy as the hints will be come through profiling or compiler analysis to the hardware. Moreover, PC based approaches will require much lower storage overhead.

  Cache efficiency measures the portion of the cache that actually holds useful data; the remaining portion holds useless data and can be vacated to store useful data. The reason cache efficiency is low is that the time a block stays alive in the cache is usually much shorter than the time that it is dead. The interval between the last access to a block and its eviction from the cache is called the dead time of the block. By identifying dead blocks early with the best dead-block predictors we evaluated for the L1 and L2 caches and replacing them with new blocks through prefetching.

  We came to know three approach for dead block prediction: trace –based (RefTrace), counting-based (RefCount) and time-based (Timekepping/TK). RefTrace records the sequence of instructions that have referenced a block by hashing the PCs of these instructions together. In RefCount, each block in the cache is augmented with a counter that records both how many times the block has been referenced and the PC of the instruction that first missed on the block. Lastly, TK dynamically learns the number of cycles a block stays alive and if the block is not accessed in more than twice this number of cycles, it is predicted dead.

  But, all prior dead-block predictors try to find regular patterns in the individual reference history of each block. For example, control flow irregularity and data alignment variation can cause irregular reference history. Based on the

observation, we found some new class of DBPs: Burst counting predictor and Burst trace predictor.

- Why is it an interesting architectural idea?

  Cache efficiency can be improved if more live blocks are stored in the cache without increasing its capacity. Improved cache efficiency reduces cache-miss rate and improves system performance.
  There are several distinct ways to use dead-block prediction to improve performance. We evaluate three ways to increase cache efficiency by eliminating dead blocks early: replacement optimization, bypassing, and prefetching. The most effective approach, prefetching into dead blocks, increases the average L1 efficiency from 8% to 17% and the L2 efficiency from 17% to 27%. This increased cache efficiency translates into higher overall performance: prefetching into dead blocks outperforms the same prefetching scheme without dead-block prediction by 12% at the L1 and by 13% at the L2. Triggering prefetches on dead-block predictions improves the timeliness of prefetching compared to triggering prefetches on cache misses. A conservative approach, including replacement optimization and bypassing, only evicts dead blocks early to give other blocks more opportunities to get reused. A more aggressive approach prefetches new blocks into dead blocks to reduce future demand misses.
  Some other cache optimizations including power reduction, and coherence protocol optimizations. Dead block prediction has also been used to reduce leakage by turning off dead blocks.

4) Related Work. Cite at least one paper that relates to your project.

Haiming Liu, Michael Ferdman, Jaehyuk Huh and Doug Burger. Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency.

5) Metric to be used.

We are going for higher cache efficiency.

6) Simulation Infrastructure. What simulator will you be using for your study? What benchmarks will you be using?

Simulator used will be Simple scalar. Benchmarks used are GCC, Bzip2, and Parser.