# Lab Group 38 Final Report

*By Ismail Ahmed and Joey Perrello*

The inspiration for our project is the paper “SWQUE: A Mode Switching Issue Queue with Priority-Correcting Circular Queue” by Dr. Hideki Ando from Nagoya University, which revolves around improving the issue queue (IQ) performance. The SWQUE mechanism they implemented dynamically switches between two modes: a Circular Queue with Priority Correction/Round-Robin (CIRC-PC/Round-Robin) and an Age-based Random Queue matrix (AGE). The SWQUE approach balances the need for correct instruction prioritization and capacity efficiency by dynamically selecting between these modes based on workload conditions. This balance is essential for dealing with the end of Dennard Scaling, which previously allowed improvements in single-thread performance through increased clock frequencies. SWQUE achieves an average performance improvement of around 9.7% for integer programs and 2.9% for floating-point programs compared to CIRC-PC, as demonstrated in the evaluations using the SPEC2017 benchmarks run on the Intel Sunny Cove processor simulation software of Scarab. The SWQUE design enhances overall processor performance without significantly increasing complexity, making it an effective solution for the challenges of modern single-thread performance.

To implement the SWQUE technique, we started by exploring the source code of Scarab, specifically the “memory/memory.c” file, which controls memory operations in the processor’s pipeline. We thoroughly studied the “memory.c” file to understand its functionality. We then identified and isolated the functions that needed modification, marking these locations with comments to indicate the points of acknowledged changes. We discovered that Scarab, by default, uses the CIRC-PC method, which meant that we only had to implement AGE and allow

Scarab to switch between both methods automatically, based on performance characteristics, to implement SWQUE. We then implemented the AGE method as a separate IQ insertion logic within Scarab, which included the priority-correcting logic to manage the circular buffer effectively.

Our approach to AGE involves the following steps. We kept the default wrap-around round-robin queue and the original CIRC-PC code. However, we wrote additional AGE code for a random age matrix to randomly select from the older half of instructions and insert them into the IQ. We also implemented a mutually exclusive toggle between the CIRC-PC and AGE modes within an existing function, activating CIRC-PC for younger instructions in the younger half of instructions while all others utilize AGE. That way, younger institutions would use CIRC-PC, and older institutions would use AGE, which follows the recommendations from the paper.

After successfully modifying the code, we moved on to our implementation's testing and debugging phases. This involved two main testing rounds: one to identify and eliminate all remaining errors, memory issues, and warnings and another to gather actual performance data, like previous lab assignments. The first testing round tested a single benchmark, the Perl interpreter, and the second testing round tested all the SPEC-2017 benchmarks.

Following debugging, we generated two plots using Python scripts (Figures 1-2). The two plots represent the performance impact of our modified SWQUE implementation compared to the original CIRC-PC implementation. The first graph shows the D-cache miss ratio across different benchmarks, while the second shows the I-cache miss ratio.

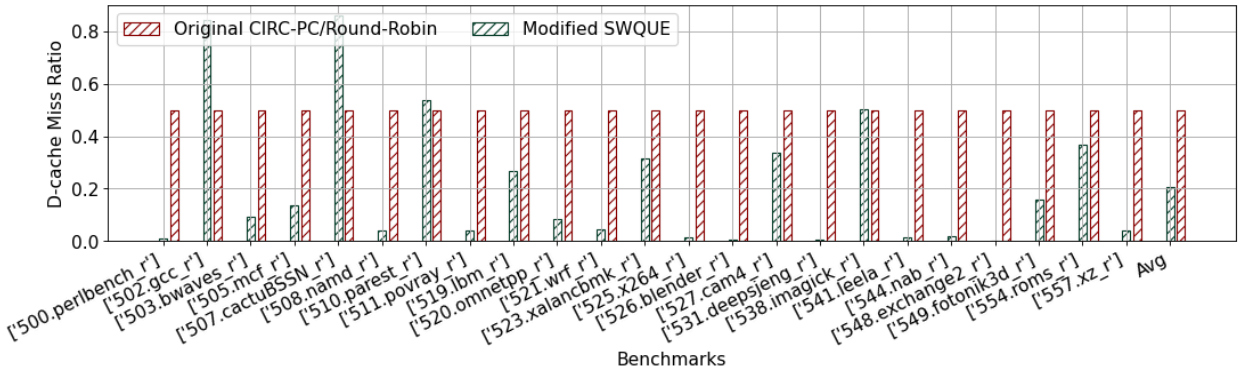


*Figure 1: A bar plot comparing the D-cache miss ratios of the original CIRC-PC/Round-Robin issue queue and the modified SQWUE issue queue.*

The modified SWQUE implementation consistently performed better than the original CIRC-PC implementation for the D-cache miss ratio, resulting in lower miss ratios across nearly all benchmarks. In Figure 1, benchmarks such as “503.bwaves_r” and “510.parest_r” show significantly lower miss ratios for SWQUE than CIRC-PC. This could be due to the nature of these benchmarks, which involve biomedical imaging and fluid dynamics, which involve more predictable memory access patterns that allow SWQUE's prioritization mechanisms to be more effective. In contrast, benchmarks like “523.xalancbmk_r” exhibit a minor difference between SWQUE and CIRC-PC, possibly indicating that our modifications have less influence over its unpredictable memory access behavior since it parses XML data. Overall, this improvement suggests a better prioritization of instructions, allowing for more efficient use of the data cache.

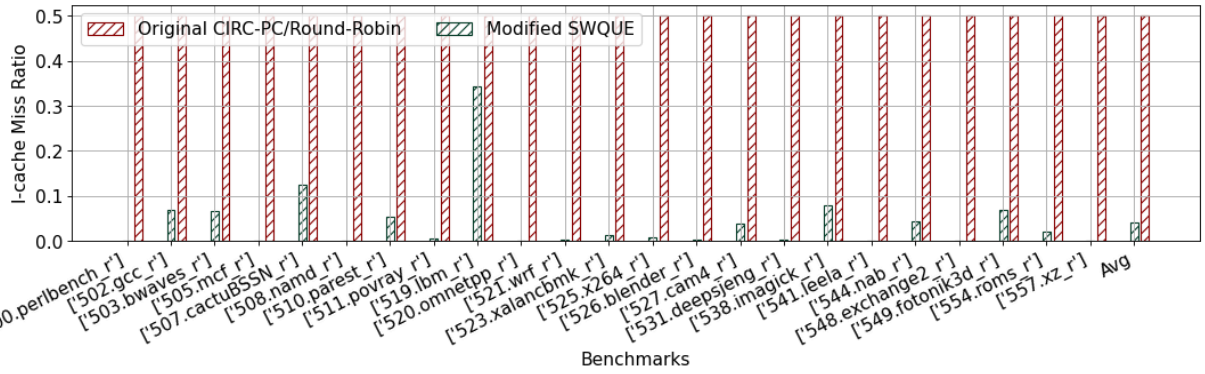


*Figure 2: A bar plot comparing the I-cache miss ratios of the original CIRC-PC/Round-Robin issue queue and the modified SQWUE issue queue.*

The I-cache miss ratio also showed a significant improvement with the SWQUE implementation compared to the original CIRC-PC implementation. Figure 2 demonstrates that CIRC-PC was worse than SWQUE in nearly all cases. Benchmarks such as “505.mcf_r” and “544.nab_r” showed substantial improvements, suggesting that SWQUE's ability to prioritize instructions correctly allowed for efficient instruction cache utilization for scheduling and anomaly detection, which have predictable memory accesses. These observations indicate that the benefits of SWQUE are particularly pronounced in benchmarks with predictable memory and instruction access, where better prioritization can reduce cache misses more effectively.

The improvements observed in both D-cache and I-cache miss ratios demonstrate the effectiveness and potential of the SWQUE technique. Our modifications have shown the potential to reduce cache miss rates, thereby enhancing processor efficiency, which directly supports the findings from the original paper. In summary, our implementation of SWQUE has successfully achieved lower miss ratios than the baseline, indicating that our design modifications effectively enhanced the processor's ability to manage instruction prioritization dynamically, with the promise of more performance gains given further study and commitment.