Project 2 Report: Bank-level Parallelism and Memory Controller Design

Tai Duc Nguyen
ECEC 623: Advanced Topics in Computer Architecture

March 3, 2020

Abstract

In modern von-Neuman computing systems, the processing unit is often limited by the amount of information allowed to flow between it and the memory. Hence, in order to maximize this bandwidth resource, memory controller designs, like FR-FCFS, are created. While this legacy design works well for a single application situation, it can massively slow down programs in a system where multiple programs are executed in parallel by allowing a few memory hungry applications to dominate the bandwidth. For this reason, many researchers have introduced a lot of complexity to the design of FRFCFS to attack this problem of "unfair memory scheduling". Nonetheless, the paper *"BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling"* by Subramanian et. al demonstrated that the BLISS design can be fair while requiring minimal amount of complexity. The authors showed that BLISS achieves 25% better fairness than the best-performing memory scheduler for benchmarks from the SPEC CPU2006 suite, TPC-C, Matlab and the NAS parallel benchmark suite. However, this project will put BLISS to the test when fared against various heavy machine learning workloads. The results from this experiment only show that BLISS has a 1.7% (on average) fairness improvement when compared to FR-FCFS.

1 Experimental Setup

In our experimental setup, we use an approximated algorithm for FRFCFS whose only goal is to maximize the memory bandwidth. On every clock cycle, our FRFCFS implementation fills as many memory instructions into the banks as much as possible. The "fairness" of this greedy algorithm is compared against the modified BLISS algorithm, whose prioritizations are as follow (1 as the most important criteria):

1. Non-blacklisted application's requests
2. Requests hit to a free bank (FRFCFS)
3. Older requests.

The blacklisted status are calculated as follow:

1. If the last application ID is the same as the one being served, increase the counter for that application ID by 1
2. If such counter exceed the *blacklisting threshold*, mark that application ID as **blacklisted**
3. If the last application ID is different than the one being served, reset the counter for the last application ID to 0
4. The blacklisting status will be reset after N cycles.

In our setup, the *blacklisting threshold* is set to be 4 and the blacklisting reset is N = 10000 cycles.

Same as project 1, some assumptions used in the experiment are:

1. DRAM Memory timings are:
    (a) Reading takes 53 clock cycles
    (b) Writing takes 53 clock cycles
2. PCM Memory timings are:
    (a) Reading takes 57 clock cycles
    (b) Writing takes 162 clock cycles
3. Maximum number of requests in the waiting queue is: 64
4. The number of banks being tested are: 8, 16
5. FR-FCFS loop searching for new request when the first one has bank-conflict can be done in 1 clock cycle

The memory traces used are generated using a Hybrid-eDRAM-PCM computer simulator running different machine learning workloads. Their names and short-handed symbols are:

1. CNN (C)
2. Bi-directional RNN (BR)
3. Variational Autoencoder (VAE)
4. Autoencoder (AE)
5. Nearest neighbor (NN)
6. Word2vec (W2V)
7. RNN (R)
8. Random Forest (RF)
9. Kmeans (K)
10. Linear Regression (LIR)
11. DCgan (DCG)
12. GAN (G)
13. Logistic Regression (LOR)
14. Dynamic RNN (DR)
15. Neural Network (NRN)

From these ML application workloads, 5 traces are created where each consists of a unique combination of 8 different ML workloads above:

1. Trace 1: C-BR-VAE-AE-NN-W2V-R-RF
2. Trace 2: C-VAE-NRN-NN-BR-RF-W2V-R
3. Trace 3: K-LIR-DCG-BR-W2V-NRN-AE-LOR
4. Trace 4: NN-K-NRN-W2V-C-AE-LIR-VAE
5. Trace 5: R-AE-NN-W2V-DR-C-G-VAE

2 Results and Discussion

This section details the results from the simulation with graphs.

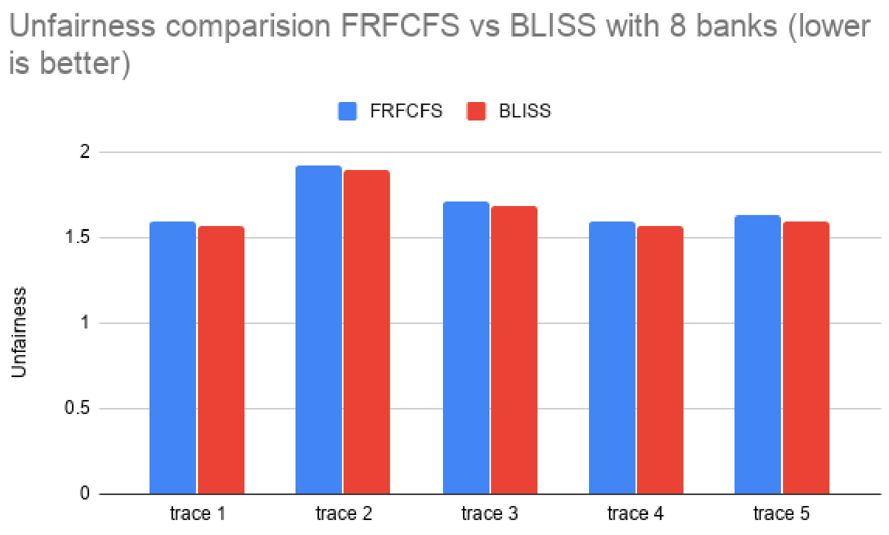

Figure 1: Unfairness metrics comparision between BLISS and FRFCFS on 5 ML traces and 8 banks

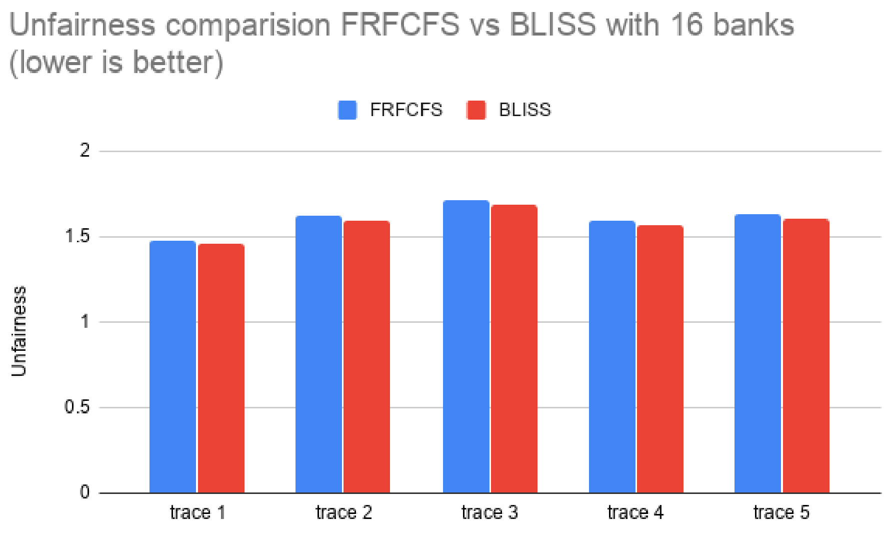

Figure 2: Unfairness metrics comparision between BLISS and FRFCFS on 5 ML traces and 16 banks

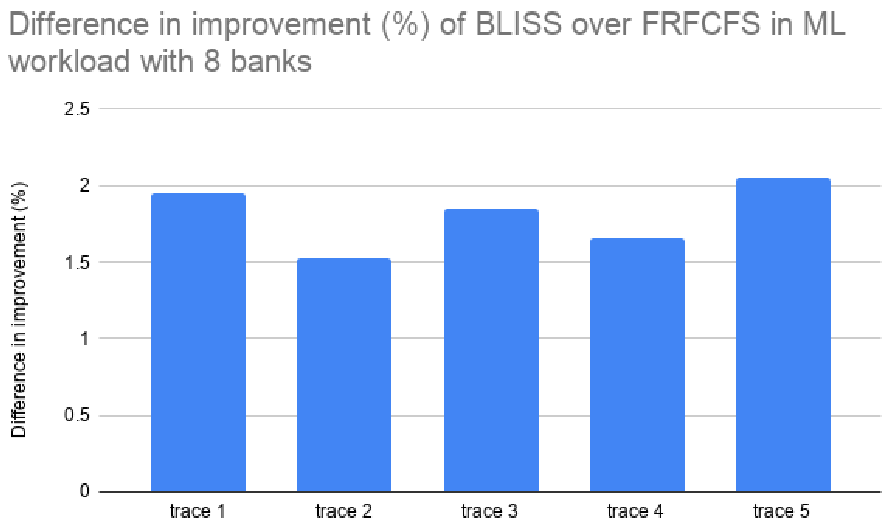

Figure 3: Improvement of BLISS over FRFCFS (in %) with 8 banks

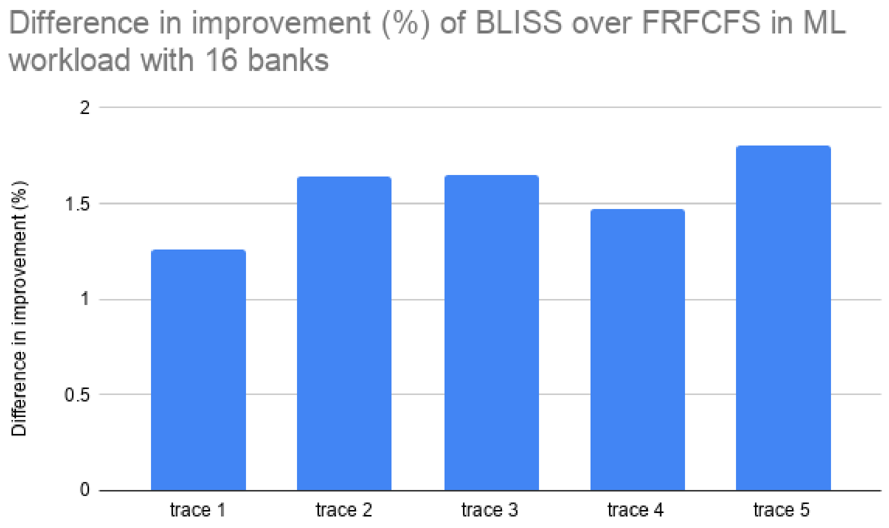

Figure 4: Improvement of BLISS over FRFCFS (in %) with 16 banks

From these results, it is apparent that BLISS is a more *fair* memory scheduler than FRFCFS; however, not by much. This low improvement (compare to the paper by Subramanian et. al) is likely due to the fact that the number of applications running simultaneously (on 8 banks) is 16, 24, and 32, which can induces a lot more interference. In addition, the effect of the modification of FRFCFS and BLISS algorithms is unknown and hard to predict.