EEL 5764
Due 12/6/19

Performance Effects of TLB Inclusion for Address Translation

TLB: Team Let's Ball
Christopher Brant, Quan Pham, and Nick Poindexter

A translation lookaside buffer is a hardware module that stores recent translations of virtual to physical addresses. Typically, these are utilized to reduce the overhead related to accessing the same locations in virtual memory over and over again, the same way that adding a cache decreases performance overhead when spatial and temporal locality is very high. Since CPUs access virtual memory via both instruction fetches, and memory loads and stores, implementing a translation lookaside buffer (TLB) for both instructions and data can be extremely helpful to overall memory access time. The purpose of this project was to be able to observe the impact that utilizing a TLB and having different configurations for TLBs will have upon a memory system.

Experimental Description

The experiment utilizes SimpleScalar micro-architecture simulator to explore the effects of TLB inclusion. Since the cross-compiling capability of SimpleScalar was set up unsuccessfully, SimpleScalar simulation feature was used instead of the emulation. However, even with this restriction, viable experiments were still able to be performed. The experiments that were performed are described in full detail below, along with descriptions and rationalizations for the metrics that were chosen as well.

The metrics to evaluate architecture with TLB and architecture without TLB are as follows. AMAT (average memory access time): we want to observe whether or not changing having TLBs would change the overall memory access time. Theoretically, TLB should not have any effect on AMAT since AMAT is primarily a cache performance metric. Therefore, our expectation is AMAT is the same throughout all experiments. TLB utilization (hit rate): there is no concrete formulation of utilization as a metric. However, we consider utilization is the number of TLB hits with respect to the number of TLB access (i.e. hit rate). If no instruction TLB is present, utilization of instruction TLB is default 0. Similarly, if no data TLB is present, utilization of data TLB is default 0. As mentioned in the introduction, TLBs are widely used to increase performance. Therefore, our expectation is that the utilization would be high once added. However, each different program has different amounts of virtual memory access. Consequently, utilization should slightly vary from one binary file to another. CPI (cycles per instruction): TLBs greatly enhance performance and especially time to access virtual memory. As a result, our expectation is that CPI would decrease as TLBs are added.

It is important to note the options for different types of TLBs that are available and those that were chosen to be used in our different configurations. There are 2 basic types of TLB, one for instructions (iTLB), one for data (dTLB). SimpleScalar provides capability to configure both

TLBs in terms of miss latency, page size, associativity (number of sets/ways), and replacement policy. For the effects of TLB inclusion to be observable through metrics mentioned above, unless otherwise specified, the miss latency is set at 100 cycles to exaggerate the impact of TLBs. All other configurable properties (e.g. cache size, etc) are kept at default. Since the effect of TLB inclusion is the primary objective, the simulated TLB configurations are as follows:

- No TLB for instructions nor for data.
- Instruction TLB and no data TLB: iTLB: page size 4096, 2 sets, 4 ways, least recently used (LRU) replacement policy.
- Data TLB and no instruction TLB: dTLB: page size of 4096, 4 sets, 4 ways, LRU replacement policy
- Both instruction TLB and data TLB: configuration exactly the same as above.
- Both TLBs with full associativity and random replacement policy.
- Both TLB with low miss latency: miss latency 1 cycle.

SimpleScalar includes many benchmarks files, some can be cross-compiled, some are already pre-compiled into binary files. 5 pre-compiled benchmark binary files that were simulated are as follows:

- fmath: floating point math
- llong: long long math
- lswlr: print hello world
- math: integer math
- printf: print extended output to terminal

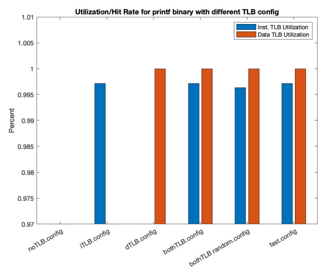
Every combination of TLB configuration and binary file were simulated to produce a total of 30 unique sets of results.

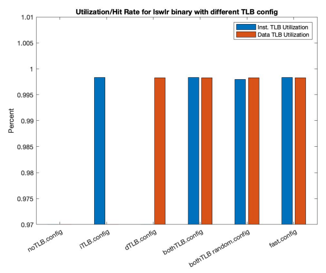
The observed performance statistics that sim-outorder provided are L1 miss rate (both instruction and data), L2 miss rate (both instruction and data), number of iTLB accesses, number of iTLB hits, number of dTLB accesses, number of dTLB hits, simulation average CPI. Based on these statistics, the formulas for metrics mentioned above are as follows:

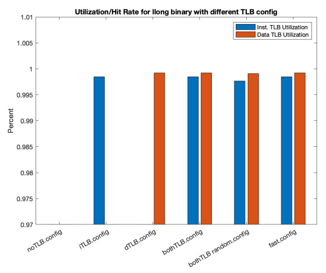
- *Data AMAT = (Data Cache L1 Hit Latency) +(Data Cache Level 1 Miss Rate)*(Data Cache L2 Hit Latency + (Data Cache L2 Miss Rate *Memory Latency)*
- *Instruction AMAT = (Instruction Cache L1 Hit Latency) +(Instruction Cache Level 1 Miss Rate)*(Instruction Cache L2 Hit Latency + (Instruction Cache L2 Miss Rate *Memory Latency)*
- $Data\ TLB\ Utilization\ Rate\ = 100 * (Data\ TLB\ Hits/Data\ TLB\ Accesses)$
- $Instruction\ TLB\ Utilization\ Rate\ = 100 * (Instruction\ TLB\ Hits/Instruction\ TLB\ Accesses)$
- SimpleScalar produces a CPI using its simulation algorithm. This CPI number was used in our analysis of results.

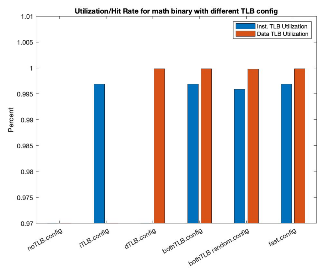
Experimental Results

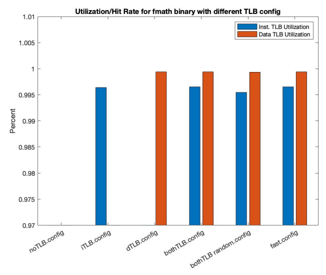
I. Utilization





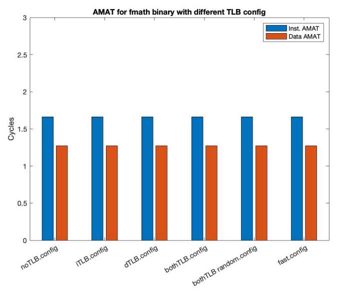


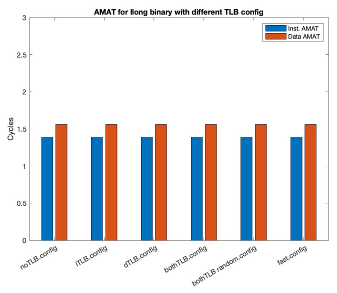


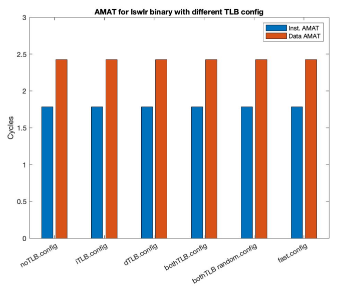


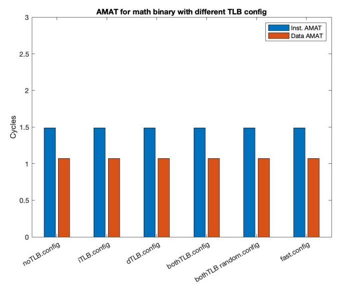
As seen in the graphs above, there is a low variance in utilization between tests. However, it is also seen that as the complexity of the test performed increases, the utilization decreases.

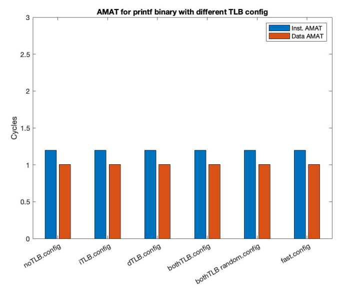
II. AMAT





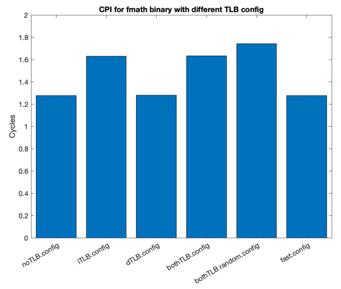


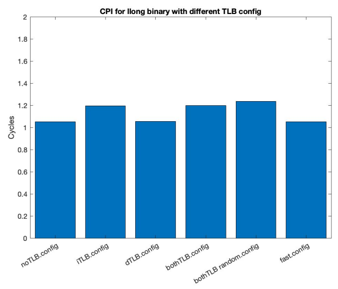


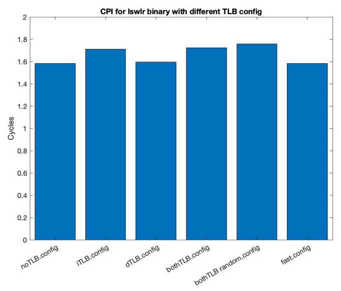


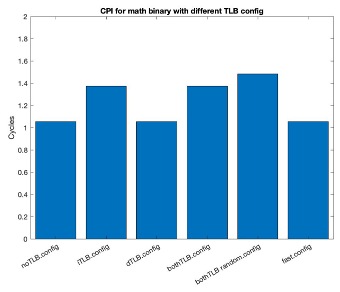
As seen in the graphs above, AMAT is unaffected by the inclusion of either a data TLB or an instruction TLB. However, there is variance between binary files that is caused by the differences in complexity of the programs.

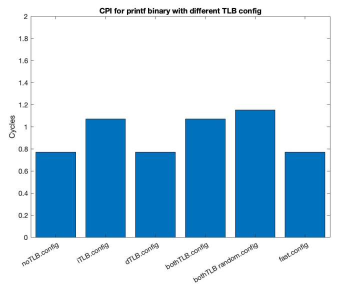
III. CPI











As previously mentioned, the TLB miss latency was dramatically increased in order to exaggerate the effect of TLB hit rate on performance. This effect is shown in the high variance of CPI dependent on the inclusion of the data and instruction TLBs. Since CPI is dependent on TLB hit rate and TLB miss latency, as the TLB hit rate changes, so does the CPI.

Discussion of Results and Conclusions

Based on observations of our experimental results, it can be concluded that including a TLB will increase the complexity of the design but will also cause a possible decrease in CPI, in our particular experimental cases. This is likely because the simulator will assume a translation time of 1 cycle if TLB is not configured, and therefore we can only observe poor TLB performance vs better TLB performance. It can also be observed that the CPI increases in the case that the TLB replacement policy is changed from LRU to random. In general, the differences in performance between the different TLB configurations was relatively small, even when we changed the associativity and replacement policy for our final configuration. It can be concluded that introducing a TLB for fast memory address translation does have an impact on system performance, as our results were able to illustrate with our high TLB miss latency. When that latency was reduced, we see that performance (in this particular simulator) is similar to not having a TLB at all, and this is likely because of the aforementioned simulator constraints. Just as well, it can be concluded that changing the TLB replacement policy to be random replacement instead of LRU will reduce performance of the overall system.