CS 3853 Computer Architecture

Project Report

Wednesday, Dec 4, 2019 7:30 p.m.

Team_9

Group Members

Christopher Coutinho

Phillip Jenkins

Tyler Mitchell

By signing this report I affirm that I know and agree with the contents.

report.docx
https://1drv.ms/w/s!AmXDBOGetBW8luIPRx4WRI2ztp6vZw?e=mnYygQ

Signatures:

Name #1: ____ [illegible] ____________________________

Name #2: ________________________________

Name #3: ________________________________

(50 pts) Milestone #3 – Analysis.
DUE: Wed Dec 4, 7:30 pm. Blackboard for softcopy. 7:31pm is LATE for the softcopy. ZERO!
Upload a .zip file with the following name to blackboard:

"2019_01_CS3853_Team_XX_M#3.zip"

TEN points deducted for incorrect name. XX is your team number.

The zip file should contain your final source code, all result output files used in the report, all softcopies of sources, and the final analysis report.

Objectives
A. Comprehend how a cache implementation works
B. Determine the performance benefits based on cache configuration
C. Learn to work in a small group

For each objective, briefly describe at least 3 specific things you learned by doing the project. This is a group answer so you do not need to list everything for everyone in the group.

Objective A (Cache Implementation) :

How to create any size cache with just the cache size, associativity and block size.

How to implement different replacement algorithms and the complexity of designing them around a cache that could be any size or associativity.

We learned how to handle address reads/writes that exceeded an index size and would rollover into the next index. We figured out ways to detect, handle and resolve these rollovers in a way that worked with out cache design.

Objective B (Performance Benefits) :

We learned that the cache enables faster CPU response times, faster data delivery than direct memory access, and that even though it costs more resources it can result in a significant speedup.

Objective C (Group work) :

We decided to use GitHub for our project and we learned a lot while using it. Managing a project with GitHub can have issues from time to time when there are collisions or progress needs to be rolled back. What was key to our success with using it was communicating our changes, small updates and giving a heads up on who was pushing at what time to reduce merge conflicts.

Coming up with ideas for the design and issues around the project was a team effort. Everyone had different solutions on how to solve issues and we needed to explain our reasoning, sometimes even demonstrating, why a given solution works or doesn't work. We all learned how to communicate our solutions better by giving good explanations for how they would work, and reasoning for why they work.

Staying motivated and not procrastinating is also a challenge when working in a group, especially when you don't know how long it could take. We learned to manage our time well by setting small goals and sporadically working on the project weeks or days before milestones were due. When the due date got really close we didn't have to procrastinate, just polish and fix things up before submission.

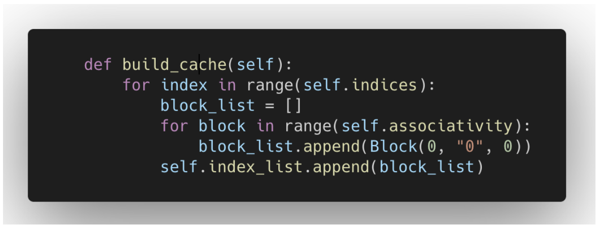
Algorithm

Briefly describe the algorithm used in implementation. How did you handle the different cache configurations – Arrays? Heap allocations? Linked lists?

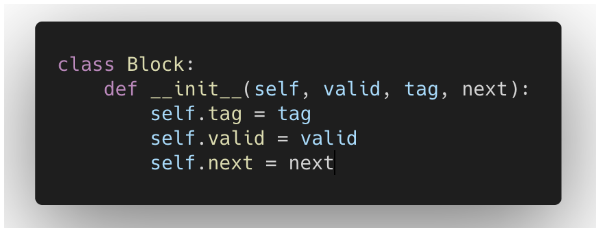
We built a 2D List/Array where the first layer represents the rows in the cache and the second layer held a list/array of Block objects to be the sets for each row.

When the cache is accessed the instruction address gets translated to an index, tag and offset. The index goes directly to the first layer of the 2D list (this represents accessing the row) then the tag gets compared to each Block in the second list until a hit is made. If no hit is made an empty block gets replaced. If there are no empty blocks the replacement algorithm will be performed and selects a block to replace.

Each Block object holds a tag and a replacement value used for Round Robin and LRU. This allowed us to simplify the complexity of doing the algorithm, essentially making each Block responsible for its own position or "turn".



```python
def build_cache(self):
    for index in range(self.indices):
        block_list = []
        for block in range(self.associativity):
            block_list.append(Block(0, "0", 0))
        self.index_list.append(block_list)
```

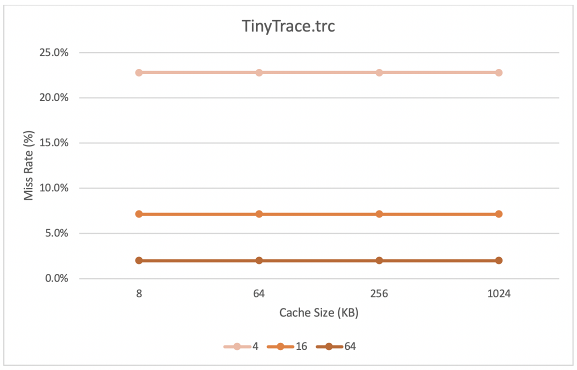


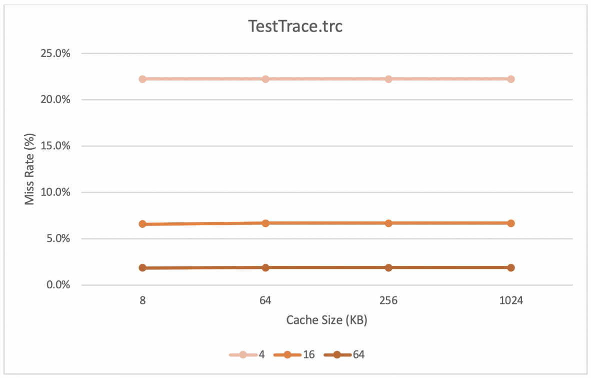
```python
class Block:
    def __init__(self, valid, tag, next):
        self.tag = tag
        self.valid = valid
        self.next = next
```

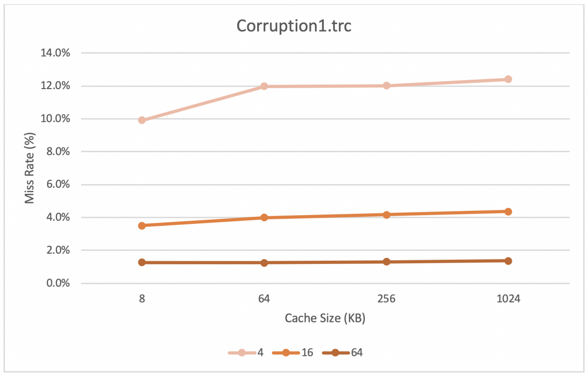
Analysis

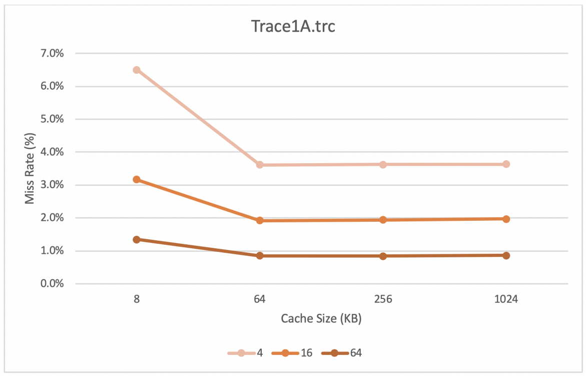
This should be the results of running your simulator over a broad range of cache configuration parameters – different sizes, different associativity, and/or different replacement algorithms. I'm not going to specify it exactly – experiment and decide which results would be good to illustrate the various configurations. Graphs are generally a good idea to visually summarize your results.

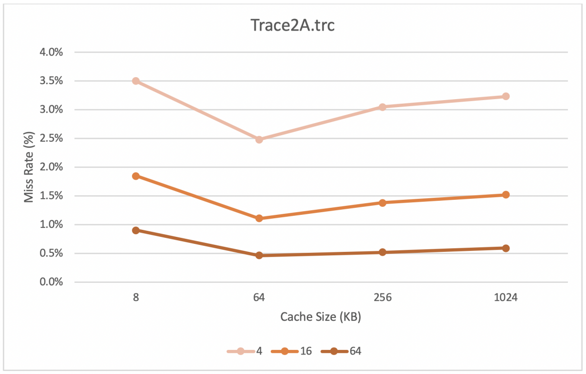
If you don't get the simulator completely working, you may use mine to do experiments and get results to report on. Just make sure you credit that.

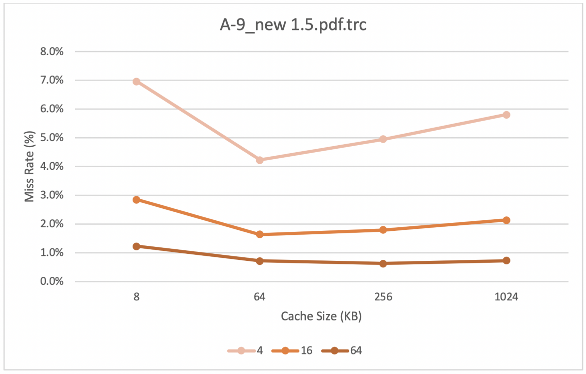




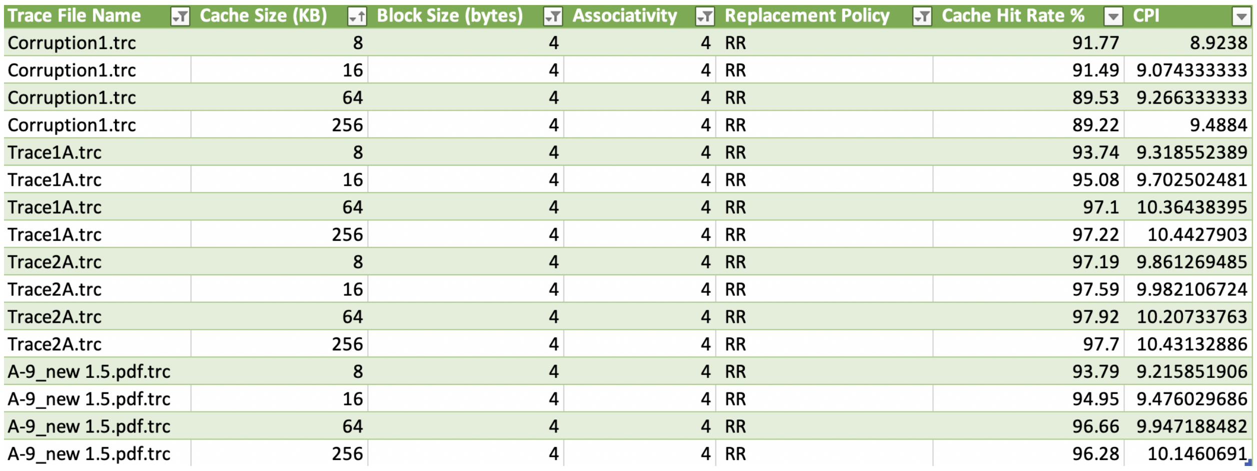




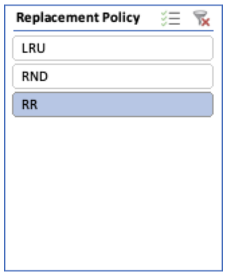




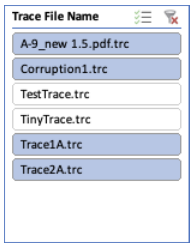
The color lines represent the different Block Sizes in KB.



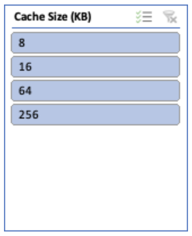
Trace File Name	Cache Size (KB)	Block Size (bytes)	Associativity	Replacement Policy	Cache Hit Rate %	CPI
Corruption1.trc	8	4	4	RR	91.77	8.9238
Corruption1.trc	16	4	4	RR	91.49	9.074333333
Corruption1.trc	64	4	4	RR	89.53	9.266333333
Corruption1.trc	256	4	4	RR	89.22	9.4884
Trace1A.trc	8	4	4	RR	93.74	9.318552389
Trace1A.trc	16	4	4	RR	95.08	9.702502481
Trace1A.trc	64	4	4	RR	97.1	10.36438395
Trace1A.trc	256	4	4	RR	97.22	10.4427903
Trace2A.trc	8	4	4	RR	97.19	9.861269485
Trace2A.trc	16	4	4	RR	97.59	9.982106724
Trace2A.trc	64	4	4	RR	97.92	10.20733763
Trace2A.trc	256	4	4	RR	97.7	10.43132886
A-9_new 1.5.pdf.trc	8	4	4	RR	93.79	9.215851906
A-9_new 1.5.pdf.trc	16	4	4	RR	94.95	9.476029686
A-9_new 1.5.pdf.trc	64	4	4	RR	96.66	9.947188482
A-9_new 1.5.pdf.trc	256	4	4	RR	96.28	10.1460691



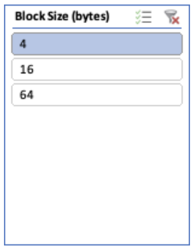
Replacement Policy
- LRU
- RND
- [✓ RR]



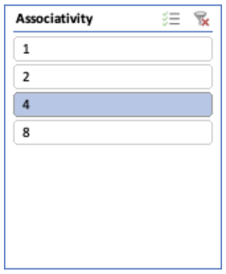
Trace File Name
- [✓ A-9_new 1.5.pdf.trc]
- [✓ Corruption1.trc]
- TestTrace.trc
- TinyTrace.trc
- [✓ Trace1A.trc]
- [✓ Trace2A.trc]



Cache Size (KB)
- [✓ 8]
- [✓ 16]
- [✓ 64]
- [✓ 256]



Block Size (bytes)
- [✓ 4]
- 16
- 64



Associativity
- 1
- 2
- [✓ 4]
- 8

Observations:

RND consistently has the same or higher hit rate than RR or LRU.

In some cases, LRU has a significantly worse hit rate.

The bigger the block size, the higher the hit rate.

The hit rate goes up the bigger the cache, especially noticeable with small block size and LRU.

had a hard time morphing the data into a graph-able format, see the results_ppt tab in charts.xlsx for interactive data view

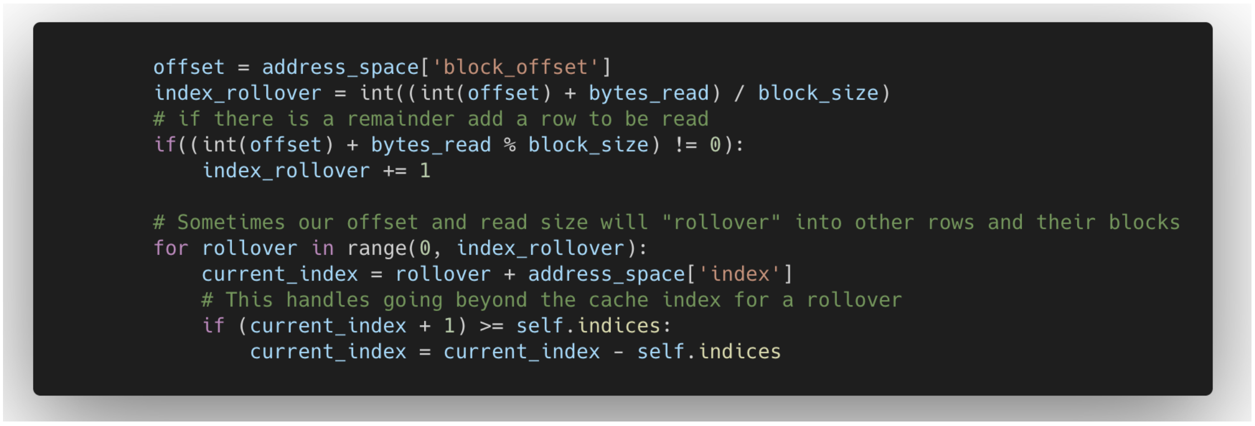
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4098b822-e9a6-4461-993c-d023ed44a26b/charts.docx

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8bd53327-a7b4-4d98-a52a-6942bec0246c/charts.xlsx

Technical Issues

Describe your most prominent technical issues and how you solved them. What was most difficult to figure out? You should have at least one and no more than three.

- How to handle the index rollover when the block offset and bytes to read cover multiple blocks



```
offset = address_space['block_offset']
index_rollover = int((int(offset) + bytes_read) / block_size)
# if there is a remainder add a row to be read
if((int(offset) + bytes_read % block_size) != 0):
    index_rollover += 1

# Sometimes our offset and read size will "rollover" into other rows and their blocks
for rollover in range(0, index_rollover):
    current_index = rollover + address_space['index']
    # This handles going beyond the cache index for a rollover
    if (current_index + 1) >= self.indices:
        current_index = current_index - self.indices
```

Group Member Contributions

Describe the contribution by each group member. This is NOT the place to complain about how one person didn't do enough or didn't work well with the group. Also, don't give me the line that everyone (in a group of 3) contributed 33.333% - not buying it! Contributions do not have to be equal but everyone should contribute something --- whether it is coding, testing and validation, or writing up the report.

Phillip Jenkins
Designing and creating cache simulation, functions, inserting addresses into cache, implementing replacement algorithms, parsing command line, debugging, testing.

Tyler Mitchell
Parsing trace files, contribution to cache calculations, organizing code, setting up the team with useful developer tools, Milestone 3 report

Christopher Coutinho
Cache math, small contributions to simulation, result automation and collection, processing result data, graphing data

Group Issues and Resolutions

None

Conclusion

Any parting thoughts on the project. Liked it? Didn’t like it? Suggestions for improvement?

The project overall helped with understanding the concepts behind CPU caching and its implementation.

Don't wait until the last minute to try to analyze the results of ~800 simulation runs. I discovered that turning the results into easily readable graphs is quite the headache.

This mainly stems from the fact that Excel keeps the original row numbers when sorting the data meaning you can't just change the selection criteria to get a new chart.

Also couldn't really find a convenient way to reorder the data the way I want it. I imagine some kind of script would be able to do it, but I'm not even sure where to start with that stuff. Maybe could've made some tiny table CSVs in python and then used that in Excel, but due to time constraints for the last step this isn't realistic.