Assignment 2

Task 1

`simpleloop` analysis:

```
$ ruby analyze_trace.rb /u/csc369h/fall/pub/a2-traces/simpleloop 0x7ff0009ee 0x7ff0009ef
Unique code pages (I):       :                    64
Unique data pages (S + L + M):                  2612
                            S:                  2524
                            L:                    74
                            M:                    14
Memory accesses before
and after the markers:                        210337
Memory accesses
in algorithm component:                       172652
```

`simpleloop` accesses memory by inserting the integers 0 to 9999 into an array in ascending order, thereby referencing memory addresses of the array in ascending order. These addresses are used mostly within a certain time interval (i.e. high temporal locality).

`matmul` analysis:

```
$ ruby analyze_trace.rb /u/csc369h/fall/pub/a2-traces/matmul-100 0x7ff0009ee 0x7ff0009ef
Unique code pages (I):       :                    74
Unique data pages (S + L + M):                  1994
                            S:                   963
                            L:                  1017
                            M:                    14
Memory accesses before
and after the markers:                        211726
Memory accesses
in algorithm component:                     62973521
```

`matmul` reads in each matrix value into memory, does the computation, then stores the result in memory. `matmul` references the same memory addresses repeatedly, scattered throughout execution, i.e. low temporal locality.

`blocked` analysis:

```
$ ruby analyze_trace.rb /u/csc369h/fall/pub/a2-traces/blocked-100-25 0x7ff0009de 0x7ff0009df
Unique code pages (I):       :                    77
Unique data pages (S + L + M):                  1995
                            S:                   963
                            L:                  1018
                            M:                    14
Memory accesses before
and after the markers:                        212190
Memory accesses
in algorithm component:                     67664112
```

Similar to `matmul`, `blocked` reads in each matrix value into memory, does the computation, then stores the result in memory. `blocked` references the same memory addresses repeatedly,scattered throughout execution, i.e. low temporal locality.

`make` analysis:

```
$ ruby analyze_trace.rb make.trace 0 0
Unique code pages (I):       :                   144
Unique data pages (S + L + M):                   276
                            S:                    71
                            L:                   162
                            M:                    43
Memory accesses before
and after the markers:                       5734245
Memory accesses
in algorithm component:                            0
```

`make` has a significant amount of memory references (5734245) compared with the above programs (e.g. 212190), yet it optimized to use significantly less memory pages.

`pwd` analysis:

```
$ ruby analyze_trace.rb pwd.trace 0 0
Unique code pages (I):       :                    90
Unique data pages (S + L + M):                   152
                            S:                    30
                            L:                   105
                            M:                    17
Memory accesses before
and after the markers:                        750473
Memory accesses
in algorithm component:                            0
```

Like `make`, `pwd` has a significant amount of memory references compared with the previous programs, yet it optimized to use significantly less memory pages.

Task 2

fifo

Trace	m	Hits	Misses	Total references	Hit rate	Miss rate
simpleloop	50	380013	2977	382990	99.2227	0.7773
simpleloop	100	380240	2750	382990	99.2820	0.7180
simpleloop	150	380280	2710	382990	99.2924	0.7076
simpleloop	200	380288	2702	382990	99.2945	0.7055
matmul-100	50	62055545	1129703	63185248	98.2121	1.7879
matmul-100	100	62101279	1083969	63185248	98.2845	1.7155
matmul-100	150	63150843	34405	63185248	99.9455	0.0545
matmul-100	200	63151365	33883	63185248	99.9464	0.0536
blocked-100-25	50	67869846	6458	67876304	99.9905	0.0095
blocked-100-25	100	67871968	4336	67876304	99.9936	0.0064
blocked-100-25	150	67872062	4242	67876304	99.9938	0.0062
blocked-100-25	200	67873126	3178	67876304	99.9953	0.0047

lru

Trace	m	Hits	Misses	Total references	Hit rate	Miss rate
simpleloop	50	380213	2777	382990	99.2749	0.7251
simpleloop	100	380312	2678	382990	99.3008	0.6992
simpleloop	150	380314	2676	382990	99.3013	0.6987
simpleloop	200	380314	2676	382990	99.3013	0.6987
matmul-100	50	62144028	1041220	63185248	98.3521	1.6479
matmul-100	100	62178803	1006445	63185248	98.4072	1.5928
matmul-100	150	63152370	32878	63185248	99.9480	0.0520
matmul-100	200	63152381	32867	63185248	99.9480	0.0520
blocked-100-25	50	67871123	5181	67876304	99.9924	0.0076
blocked-100-25	100	67872494	3810	67876304	99.9944	0.0056
blocked-100-25	150	67872509	3795	67876304	99.9944	0.0056
blocked-100-25	200	67872612	3692	67876304	99.9946	0.0054

clock

Trace	m	Hits	Misses	Total references	Hit rate	Miss rate
simpleloop	50	380013	2977	382990	99.2227	0.7773
simpleloop	100	380240	2750	382990	99.2820	0.7180
simpleloop	150	380280	2710	382990	99.2924	0.7076
simpleloop	200	380288	2702	382990	99.2945	0.7055
matmul-100	50	62055545	1129703	63185248	98.2121	1.7879
matmul-100	100	62101279	1083969	63185248	98.2845	1.7155
matmul-100	150	63150843	34405	63185248	99.9455	0.0545
matmul-100	200	63151365	33883	63185248	99.9464	0.0536
blocked-100-25	50	67869846	6458	67876304	99.9905	0.0095
blocked-100-25	100	67871968	4336	67876304	99.9936	0.0064
blocked-100-25	150	67872062	4242	67876304	99.9938	0.0062
blocked-100-25	200	67873126	3178	67876304	99.9953	0.0047

opt

Trace	m	Hits	Misses	Total references	Hit rate	Miss rate
simpleloop	50	380323	2667	382990	99.3036	0.6964
simpleloop	100	380357	2633	382990	99.3125	0.6875
simpleloop	150	380357	2633	382990	99.3125	0.6875
simpleloop	200	380357	2633	382990	99.3125	0.6875
matmul-100	50	62597795	587453	63185248	99.0703	0.9297
matmul-100	100	63092458	92790	63185248	99.8531	0.1469
matmul-100	150	63158641	26607	63185248	99.9579	0.0421
matmul-100	200	63165991	19257	63185248	99.9695	0.0305
blocked-100-25	50	67872583	3721	67876304	99.9945	0.0055
blocked-100-25	100	67873287	3017	67876304	99.9956	0.0044
blocked-100-25	150	67873776	2528	67876304	99.9963	0.0037
blocked-100-25	200	67874026	2278	67876304	99.9966	0.0034

Performance and implementation comparison

The first-in-first-out strategy for page replacement is fast and the simplest to implement (by simply cycling through each frame in `coremap`) and provides a good cache hit rate. However since it evicts the oldest page (instead of a page that will not be used for a while), it does not work well in cases where a memory address is referenced soon after eviction. The exact least-recently-used (LRU) algorithm evicts the page that was referenced longest ago, which is intended to predict which pages will not be referenced in the future. It's slightly more complex to implement than FIFO and requires more memory (adding the timestamp parameter to each page frame), but provides a slightly better cache hit rate. The clock algorithm implementation is similar in complexity to LRU, but requires less memory since only an extra ref bit needs to be stored with each frame, instead of a large timestamp. It performs almost as well as LRU, and in one case even outperformed LRU (99.9953 hit rate compared with 99.9946). Belady's algorithm (i.e. opt) has the most complex implementation since it scans ahead through the entire memory trace to determine which page will not be used for the longest time. However, it has a prohibitively long run time due to having to scan ahead, and is impractical since most of the time you will not be able to know how memory will be referenced ahead of time.

LRU

As the memory size increases with the least-recently-used (LRU) strategy, increases slightly. In the case of the `matmul-100` trace, the hit rate increased dramatically when the memory size was raised from 100 to 150: from 98.4072 to 99.9480. As the memory size approaches the number of unique pages (1994 + 74 = 2068 in the case of `matmul`), LRU becomes more effective since there is a lower chance that the least-recently-used page frame will be used sooner.

Task 3

The trace in `beladys_anomaly.txt` illustrates Belady's anomaly:

```
$ ./sim -f beladys_anomaly.txt -m 20 -a fifo

Hit count:         4
Miss count:        43
Total references:  47
Hit rate:          8.5106
Miss rate:         91.4894

$ ./sim -f beladys_anomaly.txt -m 21 -a fifo

Hit count:         3
Miss count:        44
Total references:  47
Hit rate:          6.3830
Miss rate:         93.6170
```

Notice that running `./sim` with a memory size of 21 results in a lower cache hit rate of 6.3830, compared with the hit rate of 8.5106 for a memory size of 20.

Compiling this README

Run:

```
pandoc --from-markdown+grid_tables --to-latex -V geometry:margin-1in -o README.pdf README.md
```