simpleloop

Alg : mem	Hit rate	Hit count	Miss count	Overall eviction count	Clean Eviction Count	Dirty Eviction Count
Rand: 50	70.937	7298	2990	2940	235	2705
Rand: 100	72.9879	7509	2779	2679	60	2619
Rand: 150	73.6198	7574	2714	2564	14	2550
Rand: 200	73.5809	7570	2718	2518	11	2507
FIFO: 50	70.9953	7304	2984	2934	218	2716
FIFO: 100	73.1921	7530	2758	2658	45	2613
FIFO: 150	73.5809	7570	2718	2568	16	2552
FIFO: 200	73.6586	7578	2710	2510	12	2498
LRU: 50	72.9491	7505	2783	2733	89	2644
LRU: 100	73.8919	7602	2686	2586	2	2584
LRU: 150	73.9114	7604	2684	2534	0	2534
LRU: 200	73.9114	7604	2684	2484	0	2484
CLOCK: 50	72.7547	7485	2803	2753	105	2648
CLOCK: 100	73.8725	7600	2688	2588	4	2584
CLOCK: 150	73.8822	7601	2687	2537	0	2537
CLOCK: 200	73.9016	7603	2685	2485	0	2485
OPT: 50	74.0572	7619	2669	2619	19	2600
OPT: 100	74.3002	7644	2644	2544	0	2544
OPT: 150	74.3002	7644	2644	2494	0	2494
OPT: 200	74.3002	7644	2644	2444	0	2444

matmul

Alg : mem	Hit rate	Hit count	Miss count	Overall eviction count	Clean Eviction Count	Dirty Eviction Count
Rand: 50	65.5179	1892140	995836	995786	956117	39669
Rand: 100	88.8157	2564976	323000	322900	315446	7454
Rand: 150	96.663	2791603	96373	96223	93846	2377
Rand: 200	98.0381	2831316	56660	56460	54826	1634
FIFO: 50	60.9665	1760699	1127277	1127227	1083237	43990
FIFO: 100	62.4808	1804430	1083546	1083446	1061224	22222
FIFO: 150	98.8085	2853566	34410	34260	32943	1317
FIFO: 200	98.8265	2854087	33889	33689	32433	1256

LRU: 50	63.9462	1846752	1041224	1041174	1040066	1108
LRU: 100	65.1501	1881520	1006456	1006356	1005275	1081
LRU: 150	98.8612	2855089	32887	32737	31656	1081
LRU: 200	98.8616	2855100	32876	32676	31595	1081
CLOCK: 50	63.9452	1846721	1041255	1041205	1040091	1114
CLOCK: 100	63.9503	1846868	1041108	1041008	1039926	1082
CLOCK: 150	98.85	2854764	33212	33062	31978	1084
CLOCK: 200	98.8607	2855073	32903	32703	31621	1082
OPT: 50	79.6586	2300520	587456	587406	586319	1087
OPT: 100	96.7867	2795178	92798	92698	91611	1087
OPT: 150	99.0784	2861361	26615	26465	25378	1087
OPT: 200	99.3329	2868711	19265	19065	17978	1087

blocked

Alg : mem	Hit rate	Hit count	Miss count	Overall eviction count	Clean Eviction Count	Dirty Eviction Count
Rand: 50	99.6479	2409637	8515	8465	5909	2556
Rand: 100	99.7813	2412864	5288	5188	3432	1756
Rand: 150	99.8214	2413834	4318	4168	2713	1455
Rand: 200	99.8407	2414300	3852	3652	2323	1329
FIFO: 50	99.7307	2411641	6511	6461	4186	2275
FIFO: 100	99.8206	2413814	4338	4238	2759	1479
FIFO: 150	99.8252	2413926	4226	4076	2652	1424
FIFO: 200	99.8687	2414978	3174	2974	1875	1099
LRU: 50	99.7843	2412936	5216	5166	2815	2351
LRU: 100	99.8434	2414366	3786	3686	2605	1081
LRU: 150	99.8441	2414383	3769	3619	2558	1061
LRU: 200	99.8472	2414456	3696	3496	2435	1061
CLOCK: 50	99.7824	2412891	5261	5211	2880	2331
CLOCK: 100	99.8336	2414129	4023	3923	2616	1307
CLOCK: 150	99.8373	2414217	3935	3785	2574	1211
CLOCK: 200	99.8681	2414963	3189	2989	1927	1062
OPT: 50	99.8467	2414444	3708	3658	2573	1085
OPT: 100	99.8755	2415142	3010	2910	1837	1073
OPT: 150	99.8955	2415625	2527	2377	1300	1077
OPT: 200	99.9058	2415875	2277	2077	1010	1067

my_prog

Alg : mem	Hit rate	Hit count	Miss count	Overall eviction count	Clean Eviction Count	Dirty Eviction Count
Rand: 50	80.506	12823	3105	3055	2373	682
Rand: 100	85.2712	13582	2346	2246	1595	651
Rand: 150	87.7574	13978	1950	1800	1170	630
Rand: 200	89.1135	14194	1734	1534	900	634
FIFO: 50	80.8011	12870	3058	3008	2318	690
FIFO: 100	85.4219	13606	2322	2222	1587	635
FIFO: 150	88.9126	14162	1766	1616	987	629
FIFO: 200	88.9879	14174	1754	1554	927	627
LRU: 50	82.2639	13103	2825	2775	2133	642
LRU: 100	85.9053	13683	2245	2145	1525	620
LRU: 150	89.1512	14200	1728	1578	958	620
LRU: 200	89.1512	14200	1728	1528	908	620
CLOCK: 50	82.2137	13095	2833	2783	2137	646
CLOCK: 100	85.9116	13684	2244	2144	1522	622
CLOCK: 150	89.1512	14200	1728	1578	958	620
CLOCK: 200	89.1575	14201	1727	1527	906	621
OPT: 50	87.0417	13864	2064	2014	1377	637
OPT: 100	90.1683	14362	1566	1466	829	637
OPT: 150	91.9011	14638	1290	1140	503	637
OPT: 200	92.8428	14788	1140	940	303	637

My program

The program I choose is a recursive algorithm to find the majority element of a list. I thought it would be interesting to see the effect the recursive calls and growing call stack have on the hit rate of the replacement algorithms. Given the foreknowledge that a global variable (list/array) is accessed frequently, I was interested to see how well each algorithm does at keeping the page(s) with the array in physical memory.

Comparison of algorithms

Random page replacement seldom does better than the other algorithms, and when it's better it's generally insignificantly better (with one exception: table matmul, row rand 100). For a given program and memory size, LRU and CLOCK give pretty similar results, likely because they both follow a similar philosophy that pages that haven't been accessed recently won't likely be accessed again soon. FIFO is

pretty similar to LRU/CLOCK, but often does worse (especially with memory = 50). This is likely because there is insufficient memory for all the instruction accesses, and LRU/CLOCK favours the instruction pages (those pages being accessed frequently often causes them to be some of the “most recently accessed” pages). Whereas FIFO more likely evicts the instruction pages not knowing it will likely be accessed again soon. There doesn’t appear to be Belady’s anomaly with FIFO and these programs, according to the tables. As expected, OPT is always the best.

LRU analysis

Hit rate for LRU always increases as memory increases. This is guaranteed, because LRU has the “stack property”, where the increasing the size of the memory can only improve the hit rate. The other algorithms also have increased hit rates correlated with increased memory (even though rand and FIFO don’t have the stack property), likely because the extra memory means more pages can be held at once. For LRU (and CLOCK/FIFO), the algorithm does particularly poorly with smaller memory on a large trace like matmul. The problem is that there isn’t enough memory to store all the numbers needed for the matrix multiplication, so some get booted to the swap, but they are needed later to figure out other entries in the product matrix. This is problematic for LRU because there’s no guarantee that the least recently used number evicted from memory won’t be used some time in the near future.