ACA - Lab4 - UCP

Rakesh Belagali (GTID: 903334434)

Implementation

- Refer umon.c and umon.h files.
- Set up one utility monitor (Umon) for each core.
- Utility monitors are analogous to pseudo L2 caches with entire ways for their respective cores.
- Each utility monitor has -
    - An Auxiliary Tag Directory (ATD) implemented as a 2D-array [32][16].
    - 16 hit counters for 16 ways - a 1D array of 16 elements.
    - partData - a structure to populate data needed for partitioning.
        - Number of ways for core0.
        - Total utility.
- Monitoring data -
    - One of every 32 sets of L2 cache is sampled and a total of 32 sets are monitored by Umon. This means on every L2 access if the set is one among the 32 that are sampled, it will also result in Umon access. (setIndex%32 == 0)
    - For each set, tags in ATD are arranged from MRU (most recently used) to LRU (least recently used) positions.
    - Whenever a tag in a set of ATD is a hit, the hit counter of that corresponding way is incremented and it is moved to MRU position. This will just result in reshuffling to move that tag to MRU, nothing is evicted.
    - But, if a tag is a miss, it is just inserted at MRU position of that set. NO hit counters are incremented. Tag in LRU position will be evicted.
- Making the partitioning decision -
    - Decision is taken every few million cycles.
    - Utility of a core for x ways = sum of first ‘x’ hit counters starting from MRU.
    - Total utility = utility of core0 for ‘x’ ways + utility of core1 for ‘16-x’ ways.
    - Total utility is calculated for all combinations from ‘1-15’ to ‘15-1’. (Note that at least one way is given to each core).
    - The combination that results in the highest total utility is chosen and partitioning is done according to that until the next decision event.
    - Hit counters are halved at the end of each decision.

Results

- It was run for 1000k and 10M decision intervals and both results are presented below.
- **‘Flip’** in tables means both the inputs of the test case were flipped. This is to make sure that UCP implementation treats both cores in a symmetric manner.
- Incase two UtilityTotals are maximum and equal then,

- Take sum of all hit counters for core 0.
- Take sum of all hit counters for core 1.
- If sumCore0 is greater than sumCore1 give more ways to core0 i.e. choose farther maximum. Else, vice versa.

1. Mix1

test	LRU	E1-4	E2-8	E3-12	10M-F	10M F-flip	1000k F	1000k F-flip
cycles	211035456	207230436	207089586	206985816	207271296	207270981	207216936	207216711
core0 cycles	124701140	133618685	124222535	119943645	117923930	207270980	118507875	207216710
core1 cycles	211035455	207230435	207089585	206985815	207271295	117923615	207216935	118509270
core0 ipc	0.802	0.748	0.805	0.834	0.848	0.482	0.844	0.483
core1 ipc	0.474	0.483	0.483	0.483	0.482	0.848	0.483	0.844
L2 read miss	1476565	1557001	1460186	1415008	1388952	1388952	1392327	1392327
L2 write miss	271	1215	200	157	159	159	835	835
L2 read miss %	71.641	75.544	70.846	68.654	67.39	67.39	67.554	67.554
L2 write miss %	0.046	0.206	0.034	0.027	0.027	0.027	0.141	0.141
DRAM read delay	90.906	91.558	89.216	87.882	87.256	87.256	87.159	87.159
DRAM write delay	145	145	145	145	145	145	145	145
Weighted IPC	1.915	1.8707	1.9373	1.9713	1.985	1.985	1.983	1.983

UCP gives -

- Fewer L2 read misses.
- Fewer L2 write misses.
- Better weighted-IPC.
- Part D (LRU) - application with fewer cache access will be affected by the application with larger cache access without taking into consideration who is benefited more by giving more cache space.
- Part E (static) - cache allocation is fixed. Will work well only for that specific combination of applications. But is not flexible and does not work well for multiple applications of varying cache hungryness in a real world scenario.
- Part F(UCP) - dynamically change amount of cache allocated to each application depending on how much it benefits from cache. Will require little bit of extra hardware for monitoring but gives a lot better performance.

2. Mix2

test	LRU	E1-4	E2-8	E3-12	10M F	10M F-flip	1000k F	1000k F-flip
cycles	166699051	162168181	170577421	177455311	177137071	177137161	175174171	175174171
core0 cycles	134689685	134453820	124785300	120565230	120670360	177137160	121376970	175174170
core1 cycles	166699050	162168180	170577420	177455310	177137070	120670270	175174170	121376970
core0 ipc	0.742	0.744	0.801	0.829	0.829	0.565	0.824	0.571
core1 ipc	0.6	0.617	0.586	0.564	0.565	0.829	0.571	0.824
L2 read miss	2286740	2299382	2202570	2157424	2158509	2158509	2162072	2162072
L2 write miss	584	1272	260	217	218	218	1170	1170
L2 read miss %	55.476	55.783	53.434	52.339	52.365	52.365	52.452	52.452
L2 write miss %	0.019	0.042	0.008	0.007	0.007	0.007	0.038	0.038
DRAM read delay	129.38	131.625	134.286	133.77	133.311	133.311	131.908	131.909
DRAM write delay	145	145	145	145	145	145	145	145
Weighted IPC	1.8355	1.8653	1.88	1.8808	1.8808	1.8808	1.8847	1.8847

UCP gives -

- Better results than 4 and 8 way static partitioning.
- Equal IPC as static-12 way.
- Similar but more by very little - L2 read/write miss %.

3. Mix3

test	LRU	E1-4	E2-8	E3-12	10M F	10M F-flip	1000k F	1000k F-flip
cycles	232025356	214048251	213973426	213461236	213769306	213771466	214293016	214298146
core0 cycles	167272575	178304595	171134835	162565395	177877005	213771465	178327725	214298145
core1 cycles	232025355	214048250	213973425	213461235	213769305	177876915	214293015	178327365
core0 ipc	0.598	0.561	0.584	0.615	0.562	0.468	0.561	0.467
core1 ipc	0.431	0.467	0.467	0.468	0.468	0.562	0.467	0.561
L2 read miss	3115856	3115894	3115856	3115856	3115856	3115856	3115856	3115856
L2 write miss	60	60	60	60	66	66	66	66
L2 read miss %	70.265	70.266	70.265	70.265	70.265	70.265	70.265	70.265
L2 write miss %	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
DRAM read delay	115.238	116.302	116.655	114.556	114.67	114.674	114.764	114.763
DRAM write delay	145	145	145	145	145	145	145	145
Weighted IPC	1.8531	1.8677	1.9048	1.9568	1.8714	1.8714	1.8677	1.8677

UCP gives -

- Similar performance to static way partitioning.
- It gives slightly better performance w.r.t LRU.