Computer Architecture 2022 (July-Nov)

Assignment 1: Cache Profiling

ARUNKRISHNA AMS - EE19B001

Problem Statement:
Collect the cache statistics and analyze the cache performance on the following sorting algorithms

- Bubble Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Radix Sort

using Cachegrind simulator for different cache configurations and for arrays of different sizes. The IPC of the programs are also found.

System Configuration:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	46 bits physical, 48 bits virtual
CPU(s):	20
Model name:	12th Gen Intel(R) Core(TM) i7-12700K
CPU MHz:	3600.000
CPU max MHz:	5000.0000
CPU min MHz:	800.0000
L1d cache:	288 KiB
L1i cache:	192 KiB
L2 cache:	7.5 MiB

Cache details:

	Size	**Associativity**	**Line-Size/Block Size**
Level 1 ICache	32768	8	64
Level 1 DCache	49152	12	64
Level 2 Cache	1310720	10	64
Level 3 Cache	261214400	10	64
Level 4 Cache	0	-	-

```
lscpu
getconf -a | grep CACHE
```

The valgrind simulation simulates with machine with independent D1 and I1 (data and instruction) caches and a unified second level cache L2. The machine used in this assignment has L1, L2 and L3 caches - Cachegrind autodetects the cache configuration and simulates only the first and last level caches (L1 and L3). This is because the last level cache has the most influence on run time since it masks access to main memory, while there is a very high hit rate in L1 cache.

Similarly following properties are observed:

- Write-allocate: when a write miss occurs, the block written to is brought into the D1 cache.
- Neither strict inclusive nor exclusive cache: the LL cache typically replicates all the entries of the L1 caches, because fetching into L1 involves fetching into LL first (this does not guarantee strict inclusiveness, as lines evicted from LL still could reside in L1)
- If the machine's cache configuration can't be determined by Cachegrind through x86 CPUID instruction, then it follows with default configuration of model ¾ Athlon

For quite some reason, the specified LL cache is different from the simulated one. This is thrown up as an warning in stderr:

```
--20386-- warning: L3 cache found, using its data for the LL simulation.
--20386-- warning: specified LL cache: line_size 64  assoc 10  total_size 
26,214,400
--20386-- warning: simulated LL cache: line_size 64  assoc 13  total_size 
27,262,976
```

Execution of Code:

Enter the corresponding algorithm's folder:

```
cd bubblesort
```

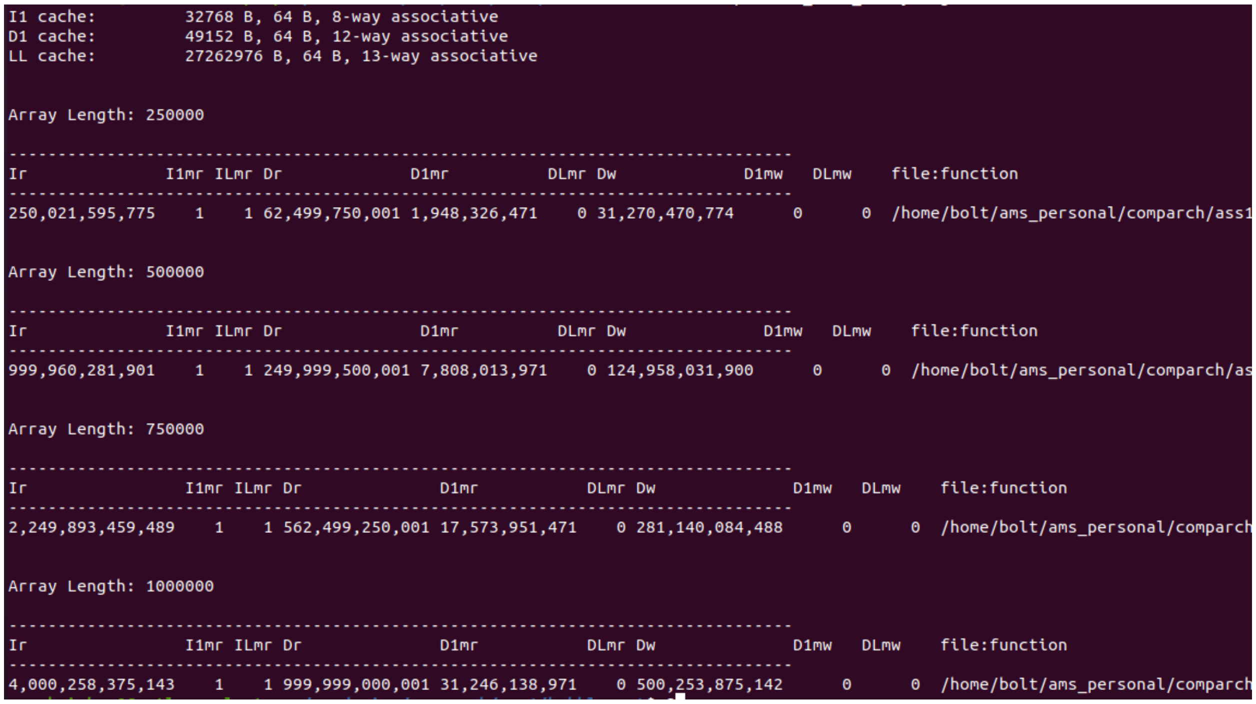
To analyze the cache performance for different cache configurations, run the following command

```
chmod +x ./bubblesort_cache.sh
./bubblesort_cache.sh
```

The output is displayed through standard output and is also stored in the file which can be viewed by

```
cat comparison_diff_config.txt
```

Following are the file contents:



```
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 12-way associative
LL cache:         27262976 B, 64 B, 13-way associative


Array Length: 250000

--------------------------------------------------------------------------------
Ir              I1mr ILmr Dr             D1mr          DLmr Dw             D1mw   DLmw    file:function
--------------------------------------------------------------------------------
250,021,595,775    1    1 62,499,750,001 1,948,326,471    0 31,270,470,774      0      0  /home/bolt/ams_personal/comparch/ass1


Array Length: 500000

--------------------------------------------------------------------------------
Ir              I1mr ILmr Dr              D1mr          DLmr Dw              D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,808,013,971    0 124,958,031,900      0      0  /home/bolt/ams_personal/comparch/as


Array Length: 750000

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr              D1mr           DLmr Dw              D1mw   DLmw    file:function
--------------------------------------------------------------------------------
2,249,893,459,489    1    1 562,499,250,001 17,573,951,471    0 281,140,084,488      0      0  /home/bolt/ams_personal/comparch


Array Length: 1000000

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr              D1mr           DLmr Dw              D1mw   DLmw    file:function
--------------------------------------------------------------------------------
4,000,258,375,143    1    1 999,999,000,001 31,246,138,971    0 500,253,875,142      0      0  /home/bolt/ams_personal/comparch
```

The first few lines mention the cache configuration used for the simulation purposes. For each of the array sizes: 250K, 500K, 750K and 1M, the simulation outputs are shown.

To analyze the cache performance for different sizes of array, run the following command

```
chmod +x ./bubblesort_array.sh
./bubblesort_array.sh
```

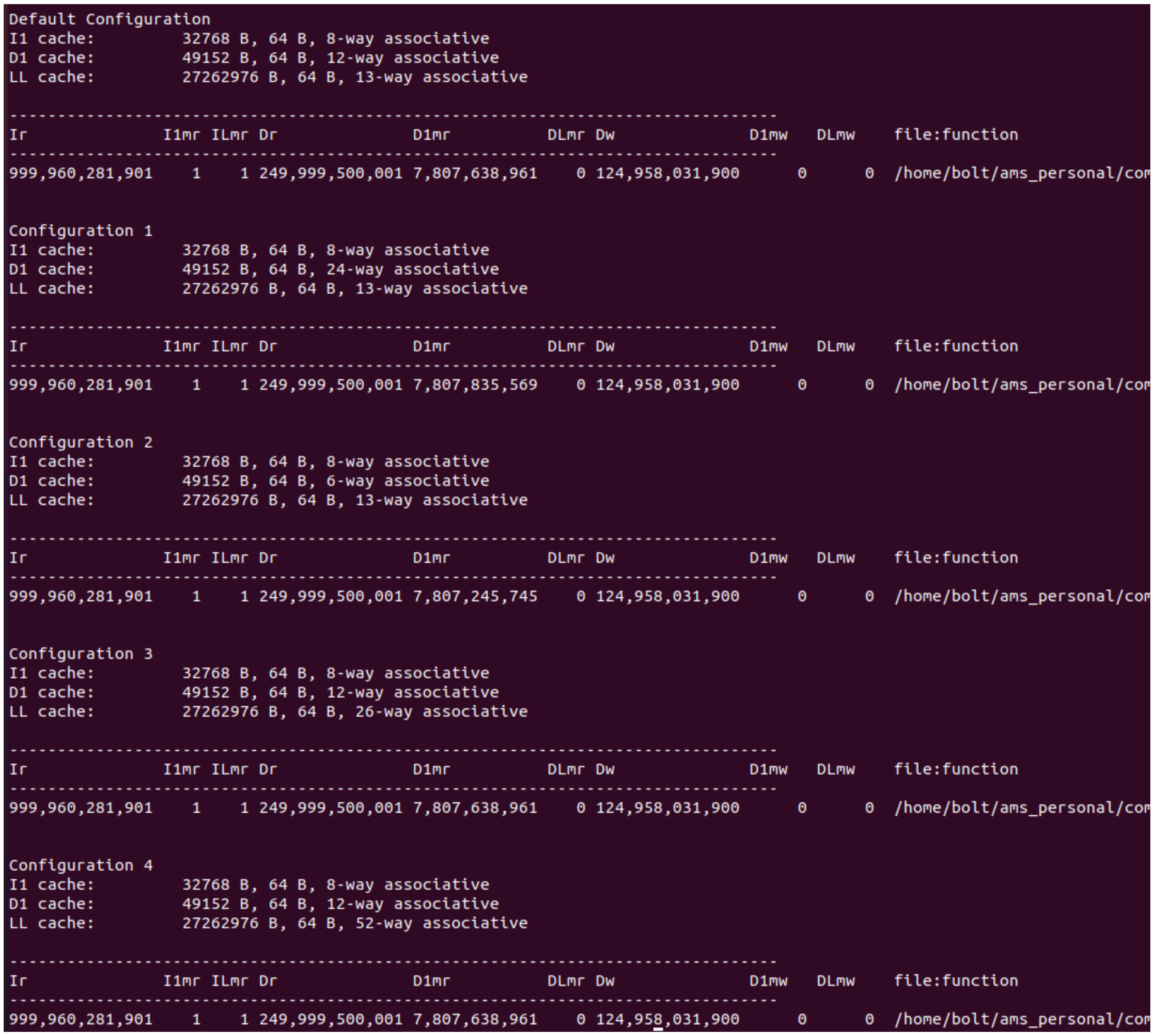
The output is displayed through standard output and is also stored in the file which can be viewed by

```
cat comparison_diff_arraylength.txt
```

For the array size 500K, the sorting algorithm is simulated with different cache configurations. Following configurations are considered:

- Default Configuration
- Increase Associativity of Data Cache from 12 to 24 while maintaining same total size
- Decrease Associativity of Data Cache from 12 to 6 while maintaining same total size
- Increase Associativity of LLC from 13 to 26 while maintaining same total size
- Increase Associativity of LLC from 13 to 52 while maintaining same total size.

Following are the file contents:



```
Default Configuration
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 12-way associative
LL cache:         27262976 B, 64 B, 13-way associative

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr                D1mr           DLmr Dw                D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,807,638,961    0 124,958,031,900      0      0  /home/bolt/ams_personal/com


Configuration 1
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 24-way associative
LL cache:         27262976 B, 64 B, 13-way associative

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr                D1mr           DLmr Dw                D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,807,835,569    0 124,958,031,900      0      0  /home/bolt/ams_personal/com


Configuration 2
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 6-way associative
LL cache:         27262976 B, 64 B, 13-way associative

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr                D1mr           DLmr Dw                D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,807,245,745    0 124,958,031,900      0      0  /home/bolt/ams_personal/com


Configuration 3
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 12-way associative
LL cache:         27262976 B, 64 B, 26-way associative

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr                D1mr           DLmr Dw                D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,807,638,961    0 124,958,031,900      0      0  /home/bolt/ams_personal/com


Configuration 4
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         49152 B, 64 B, 12-way associative
LL cache:         27262976 B, 64 B, 52-way associative

--------------------------------------------------------------------------------
Ir                I1mr ILmr Dr                D1mr           DLmr Dw                D1mw   DLmw    file:function
--------------------------------------------------------------------------------
999,960,281,901    1    1 249,999,500,001 7,807,638,961    0 124,958,031,900      0      0  /home/bolt/ams_personal/com
```

Abbreviations:
Ir: Instruction reads
I1mr : Instruction read misses in L1 cache
ILmr : Instruction read misses in LLC
Dr : Data reads
D1mr : Data read misses in L1 cache
DLmr : Data read misses in LLC
Dw : data writes
D1mw : Data write misses in L1 cache
DLmw : Data write misses in LLC

To simulate and profile the cache performance for a C-file - following commands are executed:

```
g++ "<file_name>.c" -O2 -g -o "<object_file_name>.out"

valgrind --tool=cachegrind
--cachegrind-out-file="<simulation_result_file>.out"
"<object_file_name>.out" 2> "<user_viewable_output_file1>.txt"

cg_annotate "<simulation_result_file>.out" >
"<user_viewable_output_file2_detailed>.txt"
```

File details:

```
bubblesort.c
```

Executes sorting for 500K array size. Used in profiling different cache configurations

```
bubblesort_250.c bubblesort_500.c bubblesort_750.c bubblesort_1000.c
```

Executes sorting for 250K, 500K, 750K, and 1000K array sizes.

```
./bubblesort_array.sh
```

Shell script that automatically profiles cache for different array sizes and shows the output

```
comparison_diff_arraylength.txt
```

Output of the previously mentioned shell script - with integrated results for easier analysis

```
./bubblesort_cache.sh
```

Shell script that automatically profiles cache for 500K array size for different cache configurations

```
comparison_diff_config.txt
```

Output of the previously mentioned shell script - with integrated results for easier analysis

```
bubblesort_diffarray
```

Folder containing detailed analysis of Profiling of cache for each of the different array sizes and their object files

```
bubblesort_diffconfig
```

Folder containing detailed analysis of Profiling of cache for each of the different configuration of cache and their object files

```
bubblesort_<250 or 500 or 750 or 1000>.out.cachegrind
```
Output of the simulation profile for the mentioned array size - non human readable

```
bubblesort_<250 or 500 or 750 or 1000>_output.txt
bubblesort_<250 or 500 or 750 or 1000>_annotate_output.txt
```
Output of the simulation profile for the mentioned array size in human readable format - the second text file is detailed and the output of cg_annotate

```
bubblesort_config<1 or 2 or 3 or 4 or default>.out.cachegrind
```
Output of the simulation profile for the mentioned cache configuration - non human readable

```
bubblesort_config<1 or 2 or 3 or 4 or default>_output.txt
bubblesort_config<1 or 2 or 3 or 4 or default>_annotate_output.txt
```
Output of the simulation profile for the mentioned cache configuration in human readable format - the second text file is detailed and the output of cg_annotate

RESULTS - DIFFERENT ARRAY SIZES:

Cache Configuration:

I1 cache:	32768 B, 64 B, 8-way associative
D1 cache:	49152 B, 64 B, 12-way associative
LL cache:	27262976 B, 64 B, 13-way associative

Bubble Sort:

Array Length	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
250K	250.038 B	922 (0.00%)	93.777 B	1.948 B (2.1%)	1.948 B	18141 (0.00%)
500K	999.993B	922 (0.00%)	374.971 B	7.808 B (2.1%)	7.808 B	33766 (0.00%)
750K	2249.94 B	922 (0.00%)	843. 659 B	17.574 B (2.1%)	17.574 B	49391 (0.00%)
1000K	4000.32 B	922 (0.00%)	1500.27 B	31.246 B (2.1%)	31.246 B	65016 (0.00%)

Selection Sort:

Array Length	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
250K	187.524 B	921 (0.00%)	31.257 B	1.948 B (6.2%)	1.948 B	18140 (0.00%)
500K	750.049 B	921 (0.00%)	125.014 B	7.808 B (6.2%)	7.808 B	33765 (0.00%)
750K	1687.57 B	921 (0.00%)	281.272 B	17.574 B (6.2%)	17.574 B	49390 (0.00%)
1000K	3000.09 B	921 (0.00%)	500.029 B	31.246 B (6.2%)	31.246 B	65015 (0.00%)

Merge Sort:

Array Length	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
250K	131.102 M	935 (0.01%)	35.419 M	900646 (2.5%)	901581	65056 (0.0 %)
500K	271.563 M	935 (0.01%)	73.294 M	2.050 M (2.8%)	2.051 M	127526 (0.3%)
750K	Segmentation Fault	-	-	-	-	-
1000K	Segmentation Fault	-	-	-	-	-

Quick Sort:

Array Length	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
250K	84.676 M	926 (0.00%)	23.122 M	398052 (1.7%)	398978	33788 (0.00%)
500K	179.565 M	926 (0.00%)	48.614 M	954880 (2.0%)	955806	65041 (0.00%)
750K	267.838 M	926 (0.00%)	71.926 M	145475 (2.0%)	1451401	96289 (0.00%)
1000K	363.086 M	926 (0.00%)	97.540 M	2.019 M (2.1%)	2.020 M	127541 (0.00%)

Radix Sort:

Array Length	I refs	I1 misses	D refs	D1 misses	LL refs	LL misses
250K	359.387 M	930 (0.00%)	69.054 M	2.436 M (3.5%)	2.437 M	33773 (0.00%)
500K	718.636 M	930 (0.00%)	138.062 M	4.897 M (3.5%)	4.898 M	65023 (0.00%)
750K	1077.88 M	930 (0.00%)	207.069 M	7.359 M (3.6%)	7.360 M	96273 (0.00%)
1000K	1437.13 M	930 (0.00%)	276.077 M	9.820 M (3.6%)	9.821 M	127523 (0.00%)

We can observe that the total number of instructions follows the following trends:
Quick Sort < Merge Sort < Radix Sort < Selection Sort < Bubble Sort. This is quite expected that the Bubble and Selection Sort operate at O(n^2) while Merge and QuickSort operate at O(nlogn).

We can also observe that the data accesses increases quadratically with 'n' as expected for bubble and selection - while almost linearly as 'n' for merge, radix and quick sort. Segmentation fault occurred for Merge Sort for larger array sizes - even though the code remained the same. I couldn't find the exact reason behind this.

While the size of instruction memory is independent of array size, the number of instructions executed are dependent on it. The number of instructions executed increases quadratically with 'n' as expected for bubble and selection - while almost linearly as 'n' for merge, radix and quick sort.

We can observe that the number of I1 (Instructions Level 1 Cache misses) are constant independent of array size for every sorting algorithm. The number of instructions misses are almost ~1000 for all the sorting algorithms indicating the I1 misses would most likely be Cold misses at the start of the program. From 'Instruction memory perspective', there is no 'very big jumps', thus most of the instructions reside in the cache itself.

The capacity of LL cache is quite large ~ 27 MB. The LLC misses are primarily due to cold misses. This can be asserted due to the following reasons:

- Since, the number of LL misses increases linearly with 'n' for all the sorting algorithms. If these were conflict misses - the number of misses should grow quadratically with 'n' for bubble and selection sort.
- We can also observe that the number of misses is almost same for bubble and selection sort and the number of LLC misses are lower compared to radix, merge and quick sorts. This is because - these sorts involve usage of temporary arrays - which would add more cold misses.

RESULTS - DIFFERENT CACHE CONFIGURATION

Cache Size:
I1 cache: 32768 B 8-way associative
D1 cache: 49152 B varying associativity
LL cache: 27262976 B varying associativity
Line/Block size: 64 B

Array Size: 500K

Bubble Sort:

Cache Configuration	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
D1: 12 way LL: 13 way	999.993 B	922 (0.00%)	374.971 B	7.807672 B (2.1%)	7.807673 B	33766 (0.00%)
D1: 24 way LL: 13 way	999.993 B	922 (0.00%)	374.971 B	7.807869 B (2.1%)	7.807870 B	33766 (0.00%)
D1: 6 way LL: 13 way	999.993 B	922 (0.00%)	374.971 B	7.807279 B (2.1%)	7.807280 B	33766 (0.00%)
D1: 12 way LL: 26 way	999.993 B	922 (0.00%)	374.971 B	7.807672 B (2.1%)	7.807673 B	33766 (0.00%)
D1: 12 way LL: 52 way	999.993 B	922 (0.00%)	374.971 B	7.807672 B (2.1%)	7.807673 B	33766 (0.00%)

Selection Sort:

Cache Configuration	**I refs**	**I1 misses**	**D refs**	**D1 misses**	**LL refs**	**LL misses**
D1: 12 way LL: 13 way	750.049 B	921 (0.00%)	125.014 B	7.808350 B (6.2 %)	7.808351 B	33766 (0.00%)
D1: 24 way LL: 13 way	750.049 B	921 (0.00%)	125.014 B	7.808546 B (6.2%)	7.808547 B	33766 (0.00%)
D1: 6 way LL: 13 way	750.049 B	921 (0.00%)	125.014 B	7.807957 B (6;2%)	7.807958 B	33766 (0.00%)
D1: 12 way LL: 26 way	750.049 B	921 (0.00%)	125.014 B	7.808350 B (6.2 %)	7.808351 B	33766 (0.00%)
D1: 12 way LL: 52 way	750.049 B	921 (0.00%)	125.014 B	7.808350 B (6.2 %)	7.808351 B	33766 (0.00%)

Merge Sort:

Cache Configuration	I refs	I1 misses	D refs	D1 misses	LL refs	LL misses
D1: 12 way LL: 13 way	271.563 M	935 (0.00%)	73.294 M	2050219 (2.8%)	2051154	127527 (0.0%)
D1: 24 way LL: 13 way	271.563 M	935 (0.00%)	73.294 M	2055298 (2.8%)	2056233	127527 (0.0%)
D1: 6 way LL: 13 way	271.563 M	935 (0.00%)	73.294 M	2033293 (2.8%)	2034238	127527 (0.0%)
D1: 12 way LL: 26 way	271.563 M	935 (0.00%)	73.294 M	2050219 (2.8%)	2051154	127527 (0.0%)
D1: 12 way LL: 52 way	271.563 M	935 (0.00%)	73.294 M	2050219 (2.8%)	2051154	127527 (0.0%)

Quick Sort:

Cache Configuration	I refs	I1 misses	D refs	D1 misses	LL refs	LL misses
D1: 12 way LL: 13 way	179.565 M	925 (0.00%)	48.614 M	954878 (2.0%)	955803	65039 (0.00%)
D1: 24 way LL: 13 way	179.565 M	925 (0.00%)	48.614 M	958040 (2.0%)	958965	65039 (0.00%)
D1: 6 way LL: 13 way	179.565 M	925 (0.00%)	48.614 M	947544 (2.0%)	948469	65039 (0.00%)
D1: 12 way LL: 26 way	179.565 M	925 (0.00%)	48.614 M	954878 (2.0%)	955803	65039 (0.00%)
D1: 12 way LL: 52 way	179.565 M	925 (0.00%)	48.614 M	954878 (2.0%)	955803	65039 (0.00%)

Radix Sort:

Cache Configuration	I refs	I1 misses	D refs	D1 misses	LL refs	LL misses
D1: 12 way LL: 13 way	252.115 M	930 (0.00%)	54.056 M	3255825 (6.0%)	3256755	127522 (0.00%)
D1: 24 way LL: 13 way	252.115 M	930 (0.00%)	54.056 M	3255740 (6.0%)	3256670	127522 (0.00%)
D1: 6 way	252.115 M	930	54.056 M	3255619	3256549	127522

LL: 13 way		(0.00%)		(6.0%)		(0.00%)
D1: 12 way LL: 26 way	252.115 M	930 (0.00%)	54.056 M	3255825 (6.0%)	3256755	127522 (0.00%)
D1: 12 way LL: 52 way	252.115 M	930 (0.00%)	54.056 M	3255825 (6.0%)	3256755	127522 (0.00%)

Since the number of instruction misses are very less, the magnitude of gain observed by varying associativity is negligible. We can also observe that for the bubble and selection sorts - decreasing associativity reduced the number of data cache misses - by a couple of millions. This effect is negligible for the faster sorting algorithms.

We can assume that this is because at lower associativity, the “tag comparison” operation takes up less amount of time - which might cumulatively reduce the number of clock cycles. The LLC misses remains the same even after varying the associativity.

The number of data reads, instruction reads and misses are independent of what cache architecture is being used.

IPC

The IPC of the programs are found using Perf Tools:

```
perf stat ./bubblesort.out
```

-) Bubble Sort: 2.12
-) Selection Sort: 1.73
-) Merge Sort: 1.89
-) Quick Sort: 1.43
-) Radix Sort: 2.43

Drive Link:
https://drive.google.com/drive/folders/1GfbFW43eyhRuz1DxRvjDqydlCq-sMgK-?usp=sharing