CS341: Computer Architecture Lab

Lab 4
Report

Pradipta Parag Bora (190050089)

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
2021-2022

Contents

Getting Things Ready

How easy was it for you to install Vtune and get it up and running ? What were the challenges that you faced during the installation process and how much time did it take in total?

Answer:

Initially it appeared to be simple to install and use VTunes on Ubuntu and so I installed it on my system. However I was facing numerous issues here. Firstly VTunes took a lot of time to start and secondly the sampling driver was not working for performance analysis. I built the driver again from scratch and installed it but it was not getting loaded. This meant the first part of the lab was not doable on my Ubuntu system.

So I switched to Windows and installed it there. Everything was working fine except that hotspot analysis of quicksort wouldn't work on Windows. So I had to switch to Ubuntu to get the hotspot analysis of Ubuntu done while the rest were done on Windows. Thus I was able to do the lab due to having two operating systems and the constant switching from one OS to another was quite taxing.

1. Profiling with VTune

1.1 Performance Snapshot

1.1.1 `bfs.cpp`

1. The observed IPC was 1.753
2. Logical Core utilisation was 32% (1.281 out of 4) and Physical Core utilisation was also 32% (1.281 out of 4).
3. 23.3% of Pipeline Slots are Memory Bound.

1.1.2 `matrix_multi.cpp`

1. The observed IPC was 1.043
2. Logical Core utilisation was 33.1% (1.326 out of 4) and Physical Core utilisation was 33.1% (1.326 out of 4).
3. 22.9% of Pipeline Slots are Memory Bound.

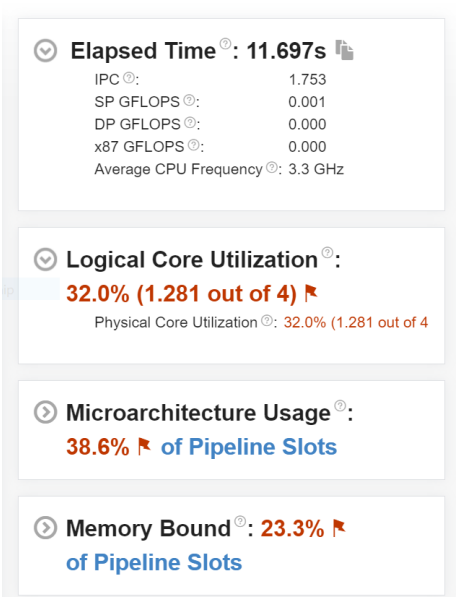
1.1.3 `matrix_multi2.cpp`

1. The observed IPC was 1.116
2. Logical Core utilisation was 33.2% (1.329 out of 4) and Physical Core utilisation was 33.2% (1.329 out of 4).
3. 20.7% of Pipeline Slots are Memory Bound.

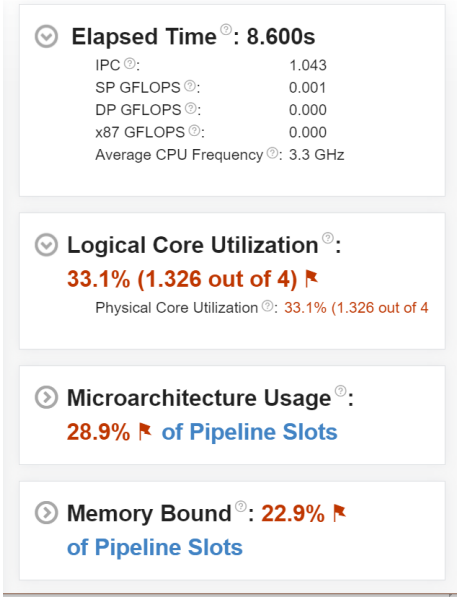
1.1.4 `quicksort.cpp`

1. The observed IPC was 1.529
2. Logical Core utilisation was 61.3% (2.454 out of 4) and Physical Core utilisation was 61.3% (2.454 out of 4).
3. 20.5% of Pipeline Slots are Memory Bound.

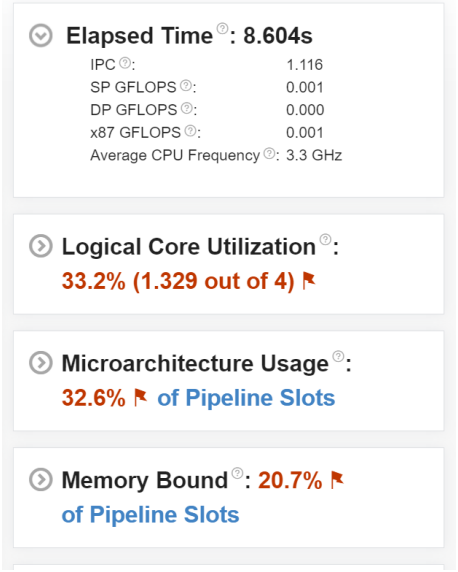
Screenshots of results of Performance Snapshots:



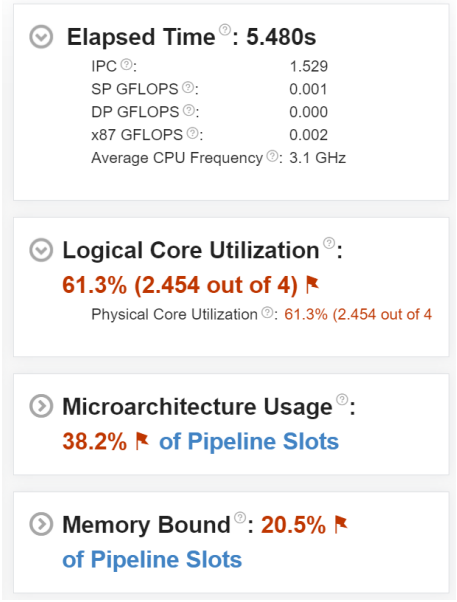
(a) `bfs.cpp`



(b) `matrix_mult.cpp`



(a) `matrix_mult2.cpp`



(b) `quicksort.cpp`

1.2 Hotspots

1.2.1 `bfs.cpp`

1. The top hotspots are:

Function	Module	CPU Time	% of CPU Time
bfs	bfs.exe	5.845s	51.2%
main	bfs.exe	2.411s	21.1%
malloc	msvcrt.dll	1.566s	13.7%
free	msvcrt.dll	1.019s	8.9%
__gnu_cxx::new_allocator<Node*>::construct	bfs.exe	0.234s	2.0%

2. The statements in decreasing order of usage are:

Line No	Line Code	Utilisation
96	`left_child = curr_node->left`	31 %
120	`bfs(root)`	21.1 %
97	`right_child = curr_node->right`	7.1 %
92	`for(int i=0; i<q_size; i++)`	7.0 %
100	`if(right_child) node_Q.push(right_child)`	4.1 %
99	`if(left_child) node_Q.push(left_child)`	1.7 %

Screenshot of Output

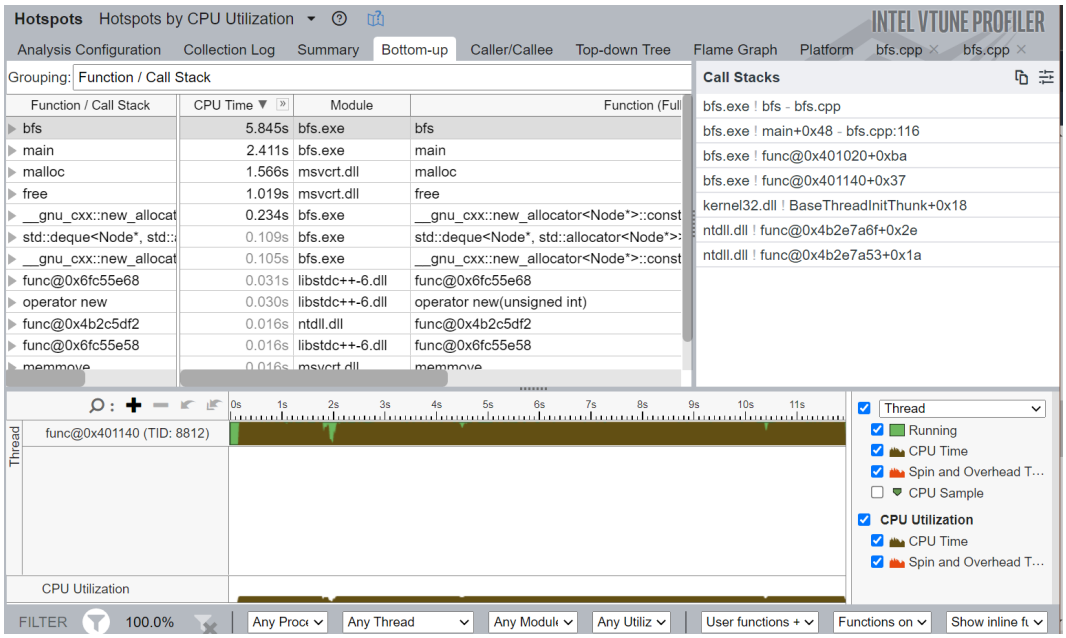


Figure 1.3: Hotspots in BFS

1.2.2 `matrix_multi.cpp`

1. The top hotspots are:

Function	Module	CPU Time	% of CPU Time
matrix_product	matrix.exe	8.702s	100.0%

2. The statements in decreasing order of usage are:

Line No	Line Code	Utilisation
32	`C[i][j] += A[i][k] * B[k][j];`	99.8 %

Screenshot of Output

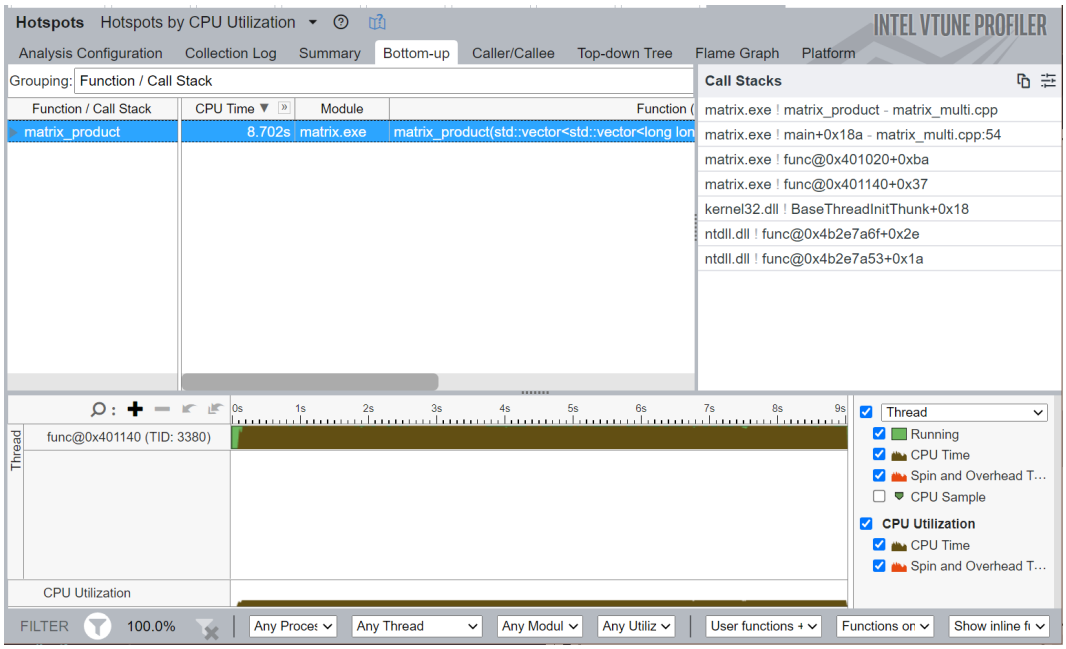


Figure 1.4: Hotspots in `matrix_mult`

1.2.3 `matrix_multi2.cpp`

1. The top hotspots are:

Function	Module	CPU Time	% of CPU Time
matrix_product	matrix2.exe	7.830s	100.0%

2. The statements in decreasing order of usage are:

Line No	Line Code	Utilisation
32	`C[i][j] += A[i][k] * B[k][j];`	98.6 %

Screenshot of Output

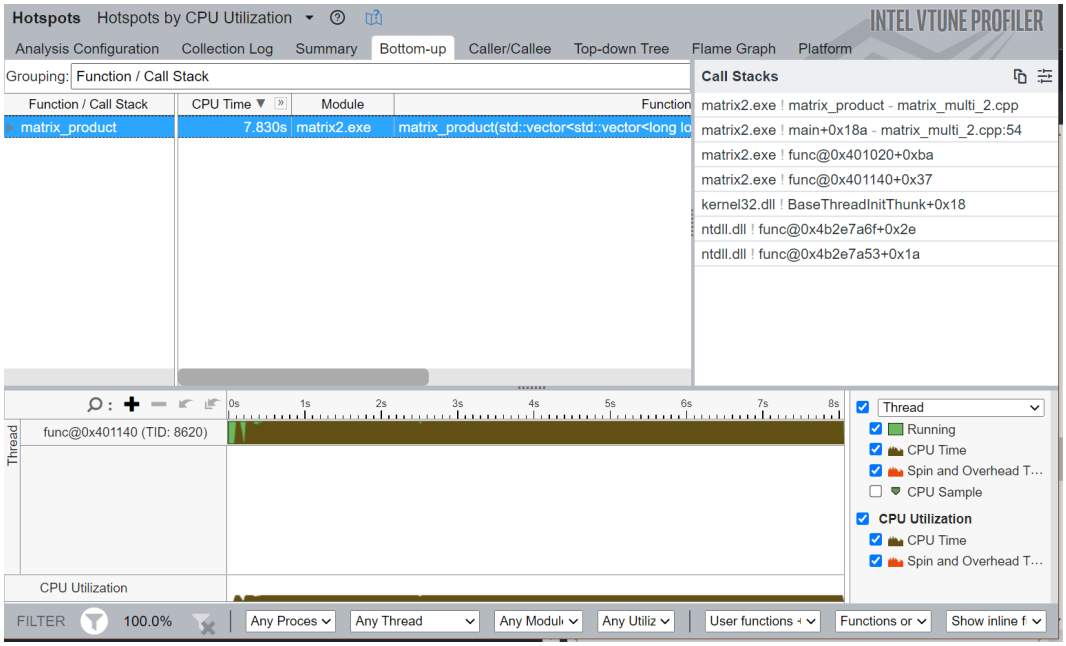


Figure 1.5: Hotspots in `matrix_multi2`

1.2.4 `quicksort.cpp`

1. The top hotspots are:

Function	Module	CPU Time
quicksort	quicksort	2.718s
operator new	libstdc++.so.6	0.280s
_int_free	libc.so.6	0.220s
partition	quicksort	0.114s
swap	quicksort	0.068s

*N/A is applied to non-summable metrics.

2. The statements in decreasing order of usage are:

Line No	Line Code	Utilisation
45	`quicksort(nums, lo, p-1)`	79.9 %
44	`long p = partition(nums, lo, hi)`	57.2 %
46	`quicksort(nums, lo, p-1)`	19.8%
31	`if(nums[i] < pivot) {`	1.2 %

Screenshot of Output

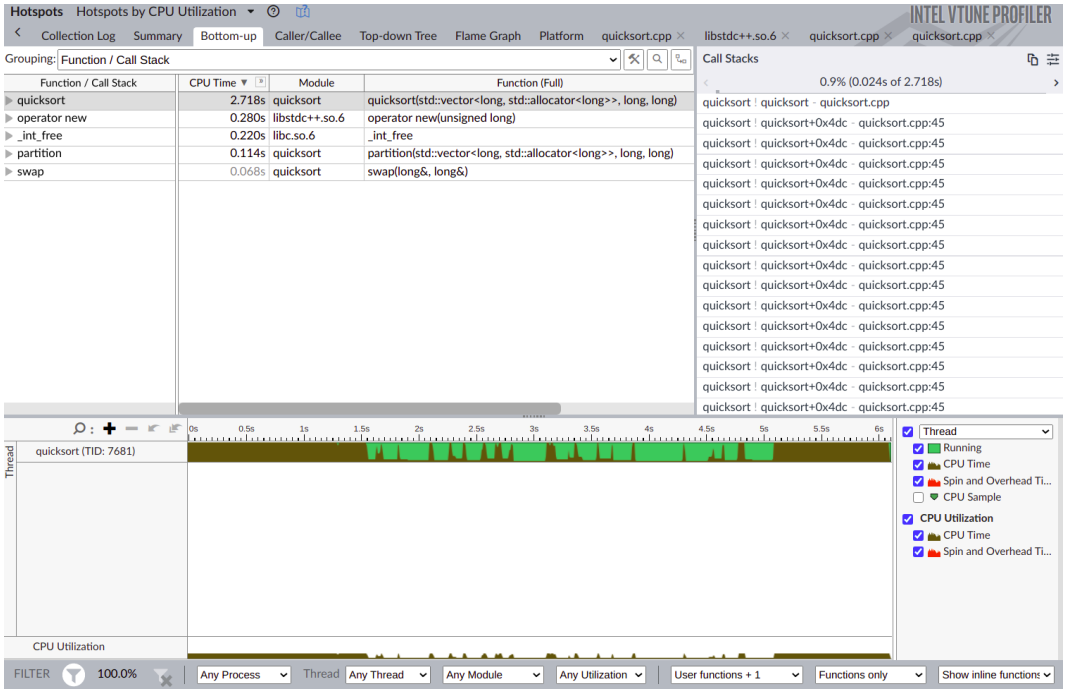


Figure 1.6: Hotspots in `quicksort`

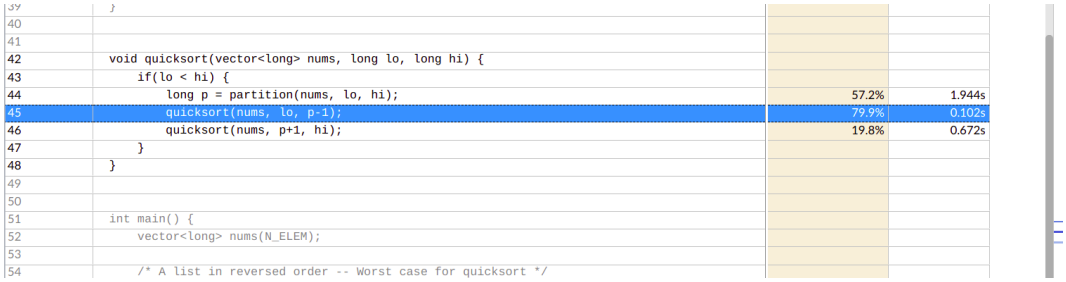


Figure 1.7: Utilisation By Percentage

2. Simulating with ChampSim

2.1 Effect of using Direct-Mapped Cache at all levels

In general here we should expect the MPKI (and hence number of misses) to increase which results in the IPC decreasing. The latency is expected to decrease.

The reason for this is that going to a direct mapped cached will increase the number of misses because two entries with same index but different tag cant be there in the cache at the same time. This will result in more MPKI and lower IPC. More associativity results in higher latency since more logic elements like MUXes are needed to select as per the tag line in more associative cache and thus in direct mapped cache (which as lower associativity) the latency should be lower. This can be seen in the observations:

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.311	0.426
LID MPKI	119.1159	6.2137
L1D Avg Miss Latency (cycles)	11.96	37.07
L1I MPKI	0.0105	0.0002
L1I Avg Miss Latency (cycles)	89.22	215
L2 MPKI	1.6489	1.2304
L2 Avg Miss Latency (cycles)	120.04	111.527
LLC MPKI	1.8314	1.2297
LLC Avg Miss Latency (cycles)	81.2704	81.5882

   The IPC decreased which is not an improvement and the MPKI increased while the average miss latency decreased mostly (for L2 cache it increased). The anomaly in L2 could be attributed to more number of misses outweighing the decrease in number of MUXes.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.560493	0.563902
LID MPKI	3.2374	0.7711
L1D Avg Miss Latency (cycles)	40.04	116.248
L1I MPKI	0.7812	0.0001
L1I Avg Miss Latency (cycles)	14.4108	44
L2 MPKI	0.9193	0.7608
L2 Avg Miss Latency (cycles)	90.06	102.623
LLC MPKI	0.7836	0.7551
LLC Avg Miss Latency (cycles)	75.3054	73.1708

   Here also the IPC decreased (although not by much) and the MPKI increased while the average miss latency decreased

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.560932	0.563837
LID MPKI	2.3797	0.7711
L1D Avg Miss Latency (cycles)	54.8854	116.5
L1I MPKI	0.644	0.0001
L1I Avg Miss Latency (cycles)	14.9034	44
L2 MPKI	1.3718	0.7604
L2 Avg Miss Latency (cycles)	70.0546	102.933
LLC MPKI	1.2284	0.7551
LLC Avg Miss Latency (cycles)	56.4319	73.4444

   Again here also the IPC decreased (although not by much) and the MPKI increased while the average miss latency decreased.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.840712	0.843171
LID MPKI	3.0039	2.0646
L1D Avg Miss Latency (cycles)	70.7696	93.7904
L1I MPKI	13.9303	0.0001
L1I Avg Miss Latency (cycles)	14.0231	215
L2 MPKI	2.2395	2.0139
L2 Avg Miss Latency (cycles)	75.0532	80.7692
LLC MPKI	1.3643	1.311
LLC Avg Miss Latency (cycles)	79.7805	77.9893

   Finally also the IPC decreased (again not by much) and the MPKI increased while the average miss latency decreased except in LLC although the values were close enough. This difference could be attributed to more amount of misses outweighing the benefit due to lower number of MUXes.

2.2 Effect of using Fully-Associative Cache at all levels

We normally expect the latency to go up with associativity. In general we expect the missrate to go down with associativity, however the difference is marginal when moving from the baseline associativity to a fully associative cache as the existing associativity could be good enough to handle the working set of the tasks. As such the MPKI and IPC values (and as a result the miss latency) are expected to be somewhat close. This is seen in the results where there is not much difference in the MPKI values and as a result the IPC values are also very close. The miss latency also is very close and we can say they are roughly the same in the results.

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.426	0.426
LID MPKI	6.2138	6.2137
L1D Avg Miss Latency (cycles)	37.2035	37.07
L1I MPKI	0.0002	0.0002
L1I Avg Miss Latency (cycles)	129.5	215
L2 MPKI	1.2305	1.2304
L2 Avg Miss Latency (cycles)	112.132	111.527
LLC MPKI	1.2296	1.2297
LLC Avg Miss Latency (cycles)	82.1925	81.5882

The values are very close, within the margin of error except L1I.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563902	0.563902
LID MPKI	0.7712	0.7711
L1D Avg Miss Latency (cycles)	116.09	116.248
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7604	0.7608
L2 Avg Miss Latency (cycles)	102.529	102.623
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	73.0384	73.1708

   The values are very close, within the margin of error.

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563837	0.563837
LID MPKI	0.7712	0.7711
L1D Avg Miss Latency (cycles)	116.487	116.5
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7604	0.7604
L2 Avg Miss Latency (cycles)	102.933	102.933
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	73.4444	73.4444

   The values are very close, within the margin of error.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.842857	0.843171
LID MPKI	2.0646	2.0646
L1D Avg Miss Latency (cycles)	93.6113	93.7904
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	215
L2 MPKI	2.0404	2.0139
L2 Avg Miss Latency (cycles)	79.5304	80.7692
LLC MPKI	1.311	1.311
LLC Avg Miss Latency (cycles)	77.0875	77.9893

The values are very close, within the margin of error.

2.3 Effect of halving the size of the caches at all levels

In general here we expect due to lower number of sets, the number of misses will go up. This should result in higher MPKI and due to more misses, more number of cycles would be missed.

Looking at the data in the programs where the working set is small enough to fit in the reduced cache there is not much difference in the MPKI. But in memory intensive tasks like matrix multiplication reducing the size of the cache will result in higher misses and hence higher MPKI and this is seen in the data.

CACTI was used to find the latency of the cache here, the change in L1I and L1D was negligible (old value of L1I was 2.29369 ns and new was 2.18023 ns access time) and similarly close values where seen in L1D which meant there was no change in the latency value here.

For L2 we observed a decrease in access time (2.77869 ns vs original value of 3.13704 ns) which resulted in changing the value of latency from 10 to 9 here.

Similarly for LLC we saw a reduction from 6.19609 ns to 5.4195 ns which resulted in reducing the latency value from 20 to 18 here.

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.425654	0.426027
LID MPKI	6.2137	6.2137
L1D Avg Miss Latency (cycles)	53.3023	37.07
L1I MPKI	0.0002	0.0002
L1I Avg Miss Latency (cycles)	209	215
L2 MPKI	1.4625	1.2304
L2 Avg Miss Latency (cycles)	167	111.527
LLC MPKI	1.2301	1.2297
LLC Avg Miss Latency (cycles)	166.457	81.5882

   As expected there is a increase in MPKI in L2 and in miss latency while the value of IPC does not change virtually at all. This is in line with our explanation that the MPKI should increase, the same value of IPC could be attributed to lower latency of the cache which compensates the increases in misses. Higher miss latency is due to increase in misses. Here the MPKI has gone up in L2 which could be attributed to the memory intensive nature of the task and hence the new cache could not fit in the working set at L2.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563825	0.563902
LID MPKI	1.1737	0.7711
L1D Avg Miss Latency (cycles)	85.765	116.248
L1I MPKI	0.0232	0.0001
L1I Avg Miss Latency (cycles)	20.375	44
L2 MPKI	0.76867	0.7608
L2 Avg Miss Latency (cycles)	107.285	102.623
LLC MPKI	0.7555	0.7551
LLC Avg Miss Latency (cycles)	84.2008	73.1708

Similarly there is a decent increase in MPKI and while the value of IPC does not change virtually at all. This being a memory intensive task we see a big jump in MPKI values at L1 which is expected as the new cache is not large enough at the L1 level. The L2 and LLC caches are large enough and so there is no difference in MPKI values there.

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563776	0.563837
LID MPKI	1.0824	0.7711
L1D Avg Miss Latency (cycles)	92.5371	116.5
L1I MPKI	0.0232	0.0001
L1I Avg Miss Latency (cycles)	19.8017	44
L2 MPKI	0.7863	0.7604
L2 Avg Miss Latency (cycles)	108.312	102.933
LLC MPKI	0.7553	0.7551
LLC Avg Miss Latency (cycles)	85.246	73.4444

   The same trends and explanation as the previous part follow here as well.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.83483	0.843171
LID MPKI	2.0655	2.0646
L1D Avg Miss Latency (cycles)	131.896	93.7904
L1I MPKI	0.001	0.0001
L1I Avg Miss Latency (cycles)	86.3	215
L2 MPKI	2.0566	2.0139
L2 Avg Miss Latency (cycles)	118.422	80.7692
LLC MPKI	1.4241	1.311
LLC Avg Miss Latency (cycles)	132.025	77.9893

   Similarly here the MPKI values have gone up due to the working set not fitting in the smaller cache while the other values are roughly similar as the increase in MPKI is compensated by lowering latency to give same IPC.

2.4 Effect of doubling the size of the caches at all levels

Here due to more number of sets, the misses would be lower which should result in lower MPKI and lower IPC. The miss latency should also be lower as a result. However in reality the difference is not

much and we see that the values are mostly close in the data. This could be because the size of the working set of the programs is smaller than the cache size and so doubling the size is not making much difference.

We again used CACTI here to find the new latency values to give in the `cache.h` file. The value for L1I and L1D remained the same while that of L2 increased from 10 to 13 (new access time 3.95873 ns vs old access time of 3.13704 ns) while of LLC increased from 18 to 24 (7.3387 ns is the new access time while the old access time was 6.19609 ns).

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.424007	0.426027
LID MPKI	6.2137	6.2137
L1D Avg Miss Latency (cycles)	40.4949	37.07
L1I MPKI	0.0002	0.0002
L1I Avg Miss Latency (cycles)	54	215
L2 MPKI	1.23	1.2304
L2 Avg Miss Latency (cycles)	113.622	111.527
LLC MPKI	1.229	1.2297
LLC Avg Miss Latency (cycles)	76.6843	81.5882

   As explained previously, no difference in IPC/MPKI due to working set fitting in smaller baseline cache. The increase in miss latency at L1D and L2 could be due to the higher latency of the larger cache.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563908	0.563902
LID MPKI	0.7703	0.7711
L1D Avg Miss Latency (cycles)	129.698	116.248
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	54	44
L2 MPKI	0.7552	0.7608
L2 Avg Miss Latency (cycles)	113.936	102.623
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	76.9462	73.1708

   Same logic as previous example, no difference in IPC/MPKI due to working set fitting in smaller baseline cache. The increase in miss latency at all levels is due to the higher latency of the larger cache.

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563841	0.563837
LID MPKI	0.7701	0.7711
L1D Avg Miss Latency (cycles)	130.13	116.5
L1I MPKI	0	0.0001
L1I Avg Miss Latency (cycles)	undefined	44
L2 MPKI	0.7552	0.7604
L2 Avg Miss Latency (cycles)	114.342	102.933
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	77.3523	73.4444

Due to increasing the size we reduced the miss from 1 to 0 in L1I resulting in undefined average miss latency. There is an increase in miss latency due to higher latency of the larger cache. However everything else is roughly the same as per the same logic in the previous part.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.849648	0.843171
LID MPKI	2.0632	2.0646
L1D Avg Miss Latency (cycles)	91.7816	93.7904
L1I MPKI	0.001	0.0001
L1I Avg Miss Latency (cycles)	54	215
L2 MPKI	1.3829	2.0139
L2 Avg Miss Latency (cycles)	110.065	80.7692
LLC MPKI	1.3109	1.311
LLC Avg Miss Latency (cycles)	77.0779	77.9893

Same as other parts, no change in IPC/MPKI. There is variation seen in miss latency though due to changing size of the cache, the latencies have changed resulting in different values.

2.5 Effect of doubling the number of MSHRs at all levels

Changing the MSHR we do not generally expect to see much change in IPC/MPKI as MSHR does not affect the properties of the cache and so accesses and misses should be similar. This is seen in the data. We may however see higher miss latency which could be due to the cache taking more time for its MSHR to fill when doubled and hence the cache will wait for more time before fetching entries from lower levels.

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.426027	0.426027
LID MPKI	6.2137	6.2137
L1D Avg Miss Latency (cycles)	37.0795	37.07
L1I MPKI	0.0002	0.0002
L1I Avg Miss Latency (cycles)	215	215
L2 MPKI	1.2304	1.2304
L2 Avg Miss Latency (cycles)	111.527	111.527
LLC MPKI	1.2297	1.2297
LLC Avg Miss Latency (cycles)	81.5882	81.5882

As expected the entries are exactly the same.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563901	0.563902
LID MPKI	0.7711	0.7711
L1D Avg Miss Latency (cycles)	117.115	116.248
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7608	0.7608
L2 Avg Miss Latency (cycles)	103.501	102.623
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	74.056	73.1708

   Again the values are very close. We only see see an increase in miss latency and this is explained by higher MSHR, the cache will wait longer until MSHR is full to fetch entries from lower levels which results in higher miss latency.

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563835	0.563837
LID MPKI	0.7711	0.7711
L1D Avg Miss Latency (cycles)	117.333	116.5
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7604	0.7604
L2 Avg Miss Latency (cycles)	103.777	102.933
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	74.2947	73.4444

   Same explanation as the previous part.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.843177	0.843171
LID MPKI	2.0646	2.0646
L1D Avg Miss Latency (cycles)	93.8293	93.7904
L1I MPKI	0.001	0.0001
L1I Avg Miss Latency (cycles)	215	215
L2 MPKI	2.0139	2.0139
L2 Avg Miss Latency (cycles)	80.8081	80.7692
LLC MPKI	1.311	1.311
LLC Avg Miss Latency (cycles)	78.049	77.9893

There is no difference at all except a slight increase in miss latency which follows from the same logic given in matrix multiplication.

2.6 Effect of halving the number of MSHRs at all levels

Similarly here we do not expect to see changes in IPC/MPKI as MSHR does not affect the properties of the cache and so accesses and misses should be similar. However we do expect the miss latency to go down as now MSHR will get filled earlier and so fetching from lower levels would be done more quicker.

1. `quicksort`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.426027	0.426027
LID MPKI	6.2137	6.2137
L1D Avg Miss Latency (cycles)	37.0795	37.07
L1I MPKI	0.0002	0.0002
L1I Avg Miss Latency (cycles)	215	215
L2 MPKI	1.2304	1.2304
L2 Avg Miss Latency (cycles)	111.527	111.527
LLC MPKI	1.2297	1.2297
LLC Avg Miss Latency (cycles)	81.5882	81.5882

   As expected no change in IPC/MPKI. There is also no change in miss latency which could be due to low misses and hence the MSHR getting filled is not a factor here.

2. `matrix_multi`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563901	0.563902
LID MPKI	0.7711	0.7711
L1D Avg Miss Latency (cycles)	116.04	116.248
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7608	0.7608
L2 Avg Miss Latency (cycles)	102.412	102.623
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	72.9589	73.1708

   Again as expected no change in IPC/MPKI. But there is a slight decrease in miss latency which follows from our explanation above.

3. `matrix_multi2`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.563836	0.563837
LID MPKI	0.7711	0.7711
L1D Avg Miss Latency (cycles)	116.313	116.5
L1I MPKI	0.0001	0.0001
L1I Avg Miss Latency (cycles)	44	44
L2 MPKI	0.7604	0.7604
L2 Avg Miss Latency (cycles)	102.742	102.933
LLC MPKI	0.7551	0.7551
LLC Avg Miss Latency (cycles)	73.2527	73.4444

Similarly as expected no change in IPC/MPKI. But there is a slight decrease in miss latency which follows from our explanation above.

4. `bfs`:

   The following data was observed:

Metric	Current Value	Baseline Value
IPC	0.843142	0.843171
LID MPKI	2.0646	2.0646
L1D Avg Miss Latency (cycles)	93.7666	93.7904
L1I MPKI	0.001	0.0001
L1I Avg Miss Latency (cycles)	215	215
L2 MPKI	2.0139	2.0139
L2 Avg Miss Latency (cycles)	80.7452	80.7692
LLC MPKI	1.311	1.311
LLC Avg Miss Latency (cycles)	77.9524	77.9893

   Same as the above two, there is no change in IPC/MPKI. There is a slight decrease in miss latency which follows from our explanation above.

Please move to the next page for plots and results.

3. Results

Here we have the plots of IPC Speedup and MPKI improvement. The IPC speedup is given in normalised values as IPC/Baseline IPC. A value of normalised IPC value greater than 1 means that the performance is better than baseline and is therefore good while a lower than 1 value means performance has degraded.

MPI improvement is the percentage change in MPKI so a high value is a degradation actually in performance while a lower value means misses have reduced.

We have 5 plots, one for IPC speedup and 4 for MPKI improvement of each cache.

At the end we have 4 additional plots that simply plot MPKI for a better comparision (no percentage). The percentage plots have some very high values that make it difficult to see other values that are smaller.

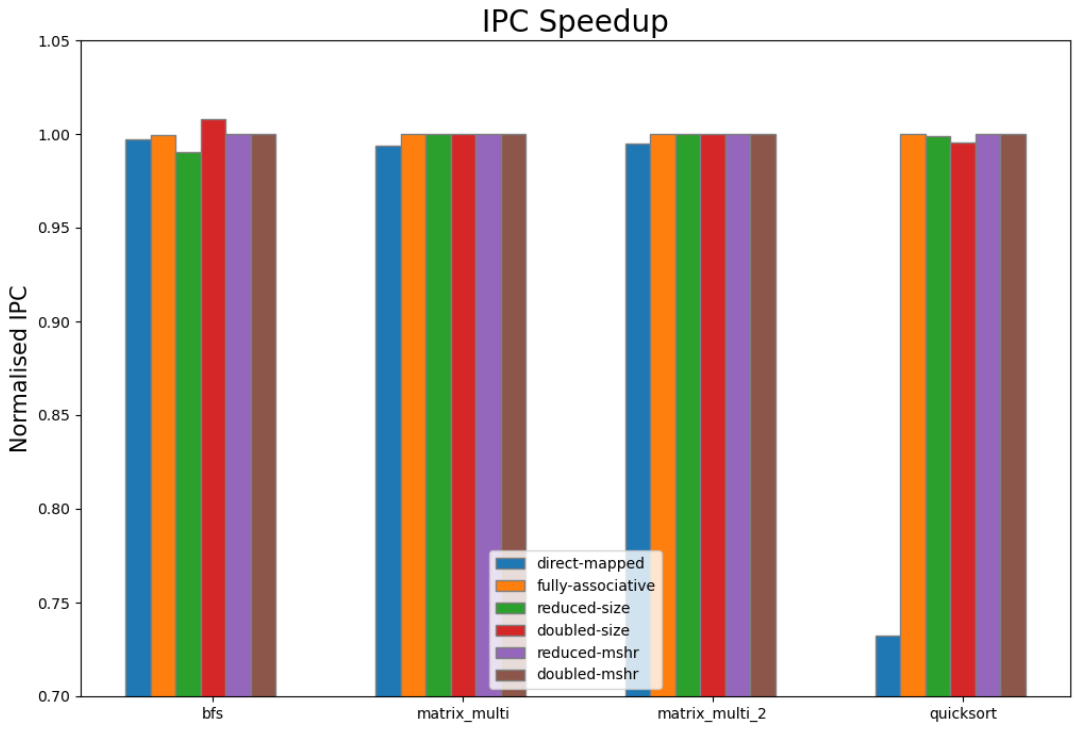


Figure 3.1: IPC Speedup Plot

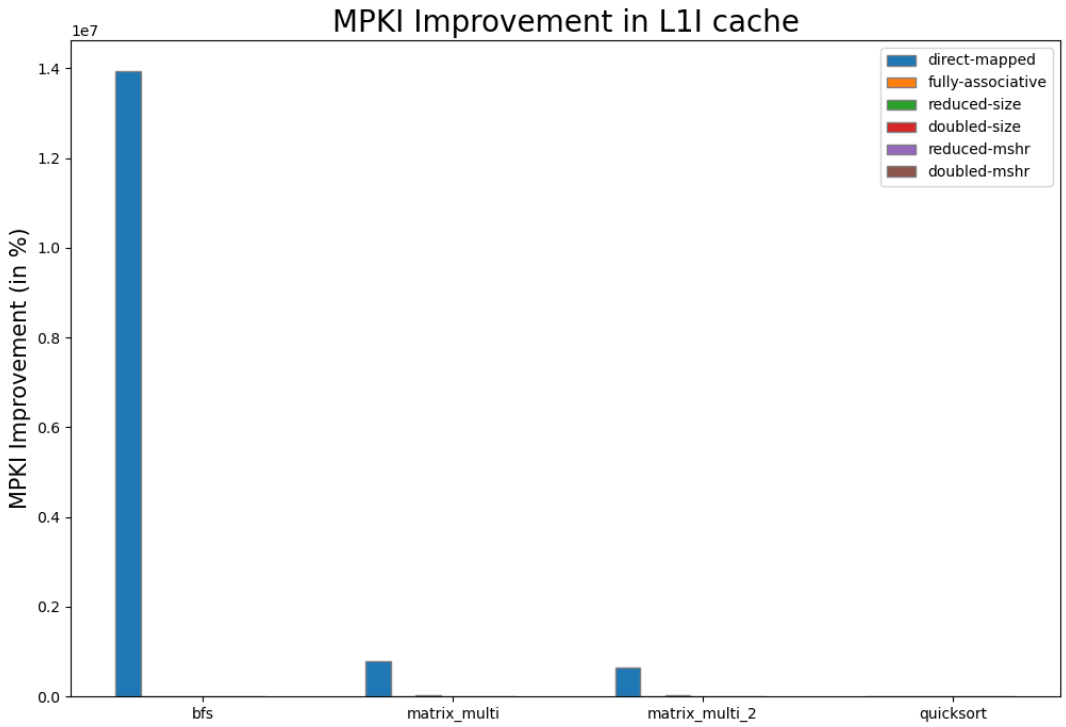


Figure 3.2: MPKI Improvement in L1I

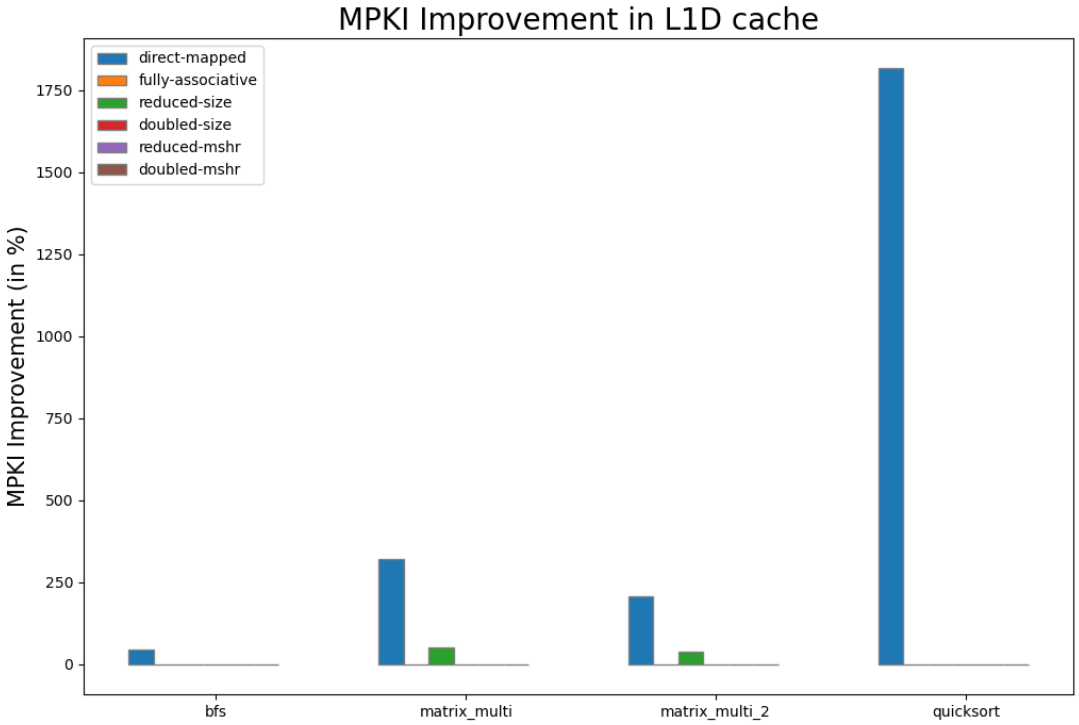


Figure 3.3: MPKI Improvement in L1D

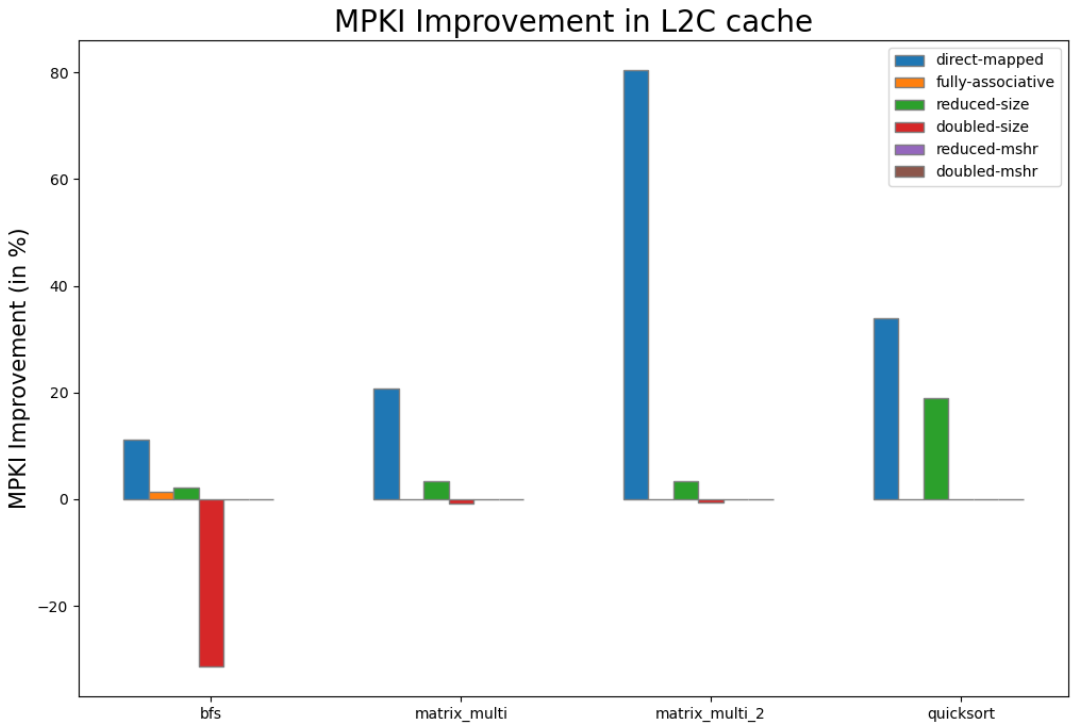


Figure 3.4: MPKI Improvement in L2

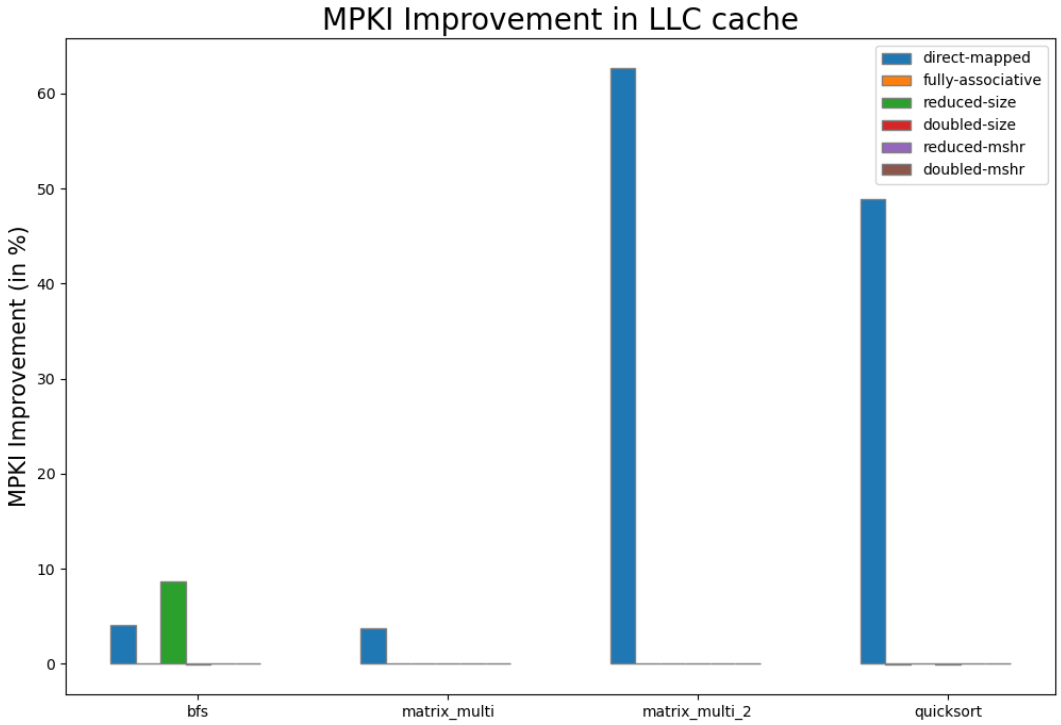


Figure 3.5: MPKI Improvement in LLC

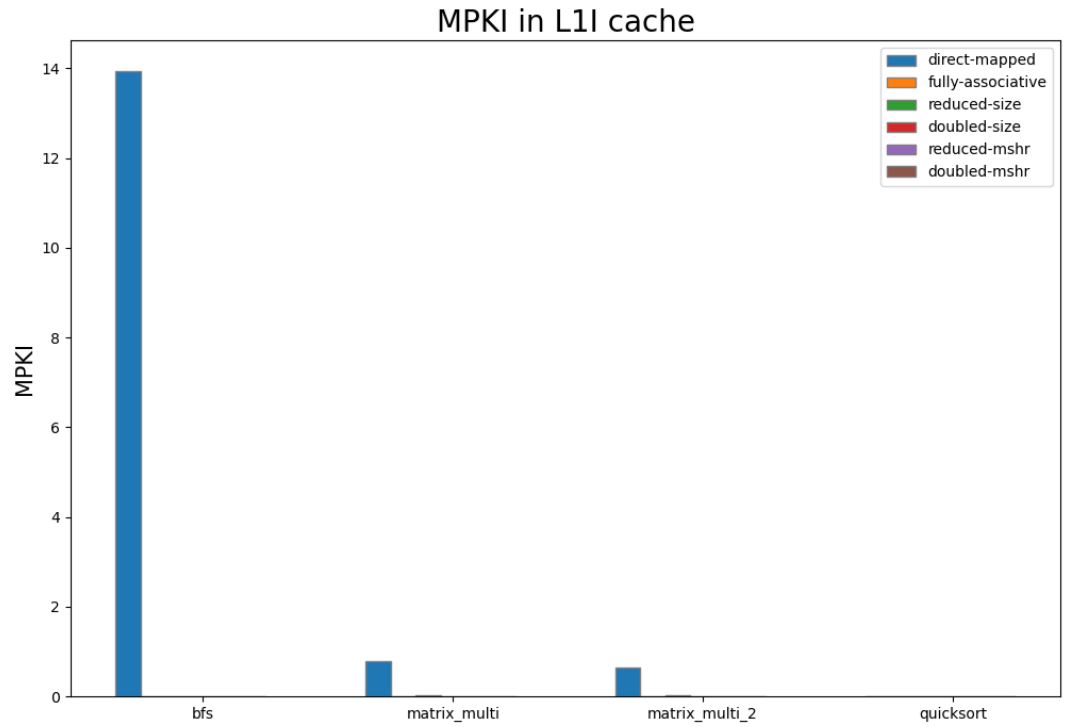


Figure 3.6: MPKI in L1I

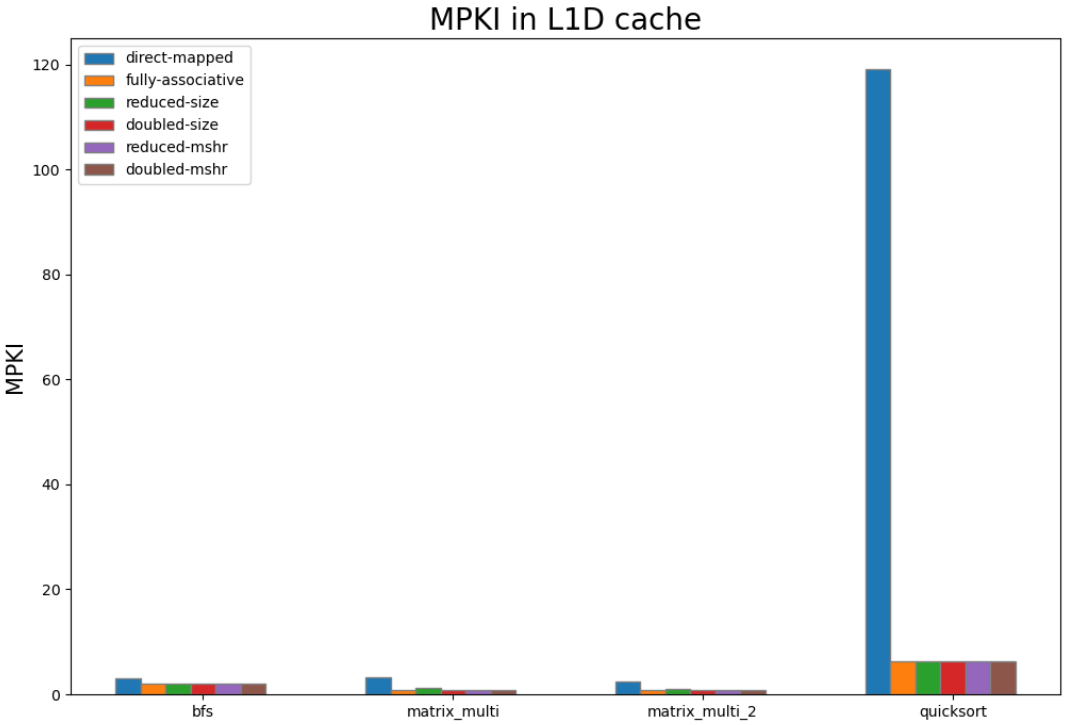


Figure 3.7: MPKI in L1D

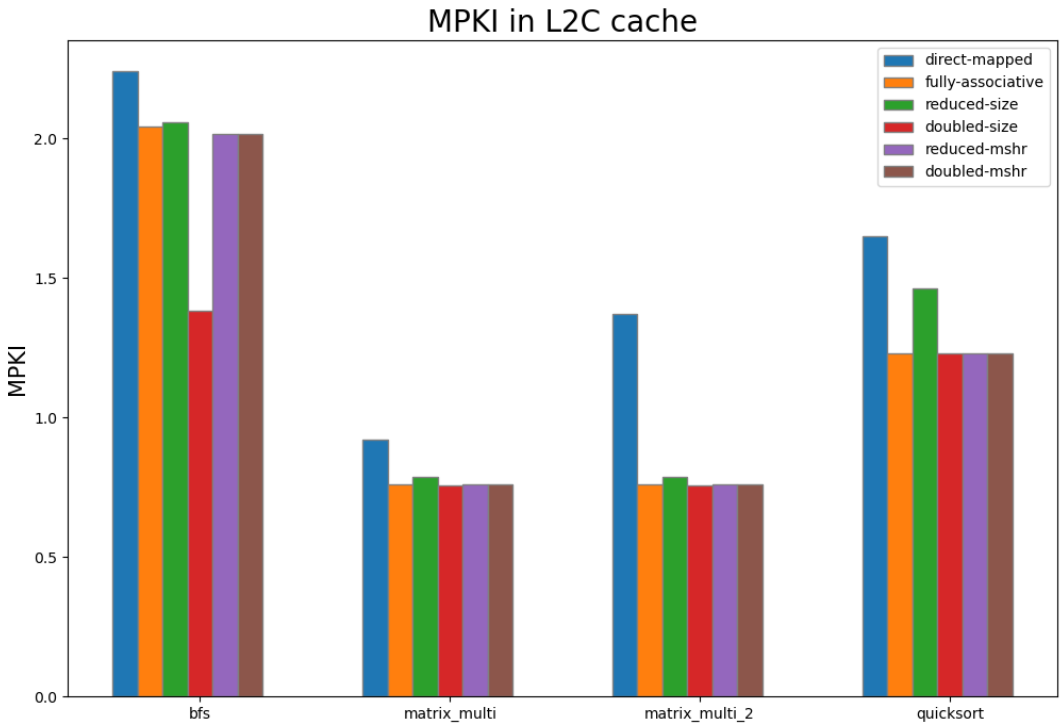


Figure 3.8: MPKI in L2

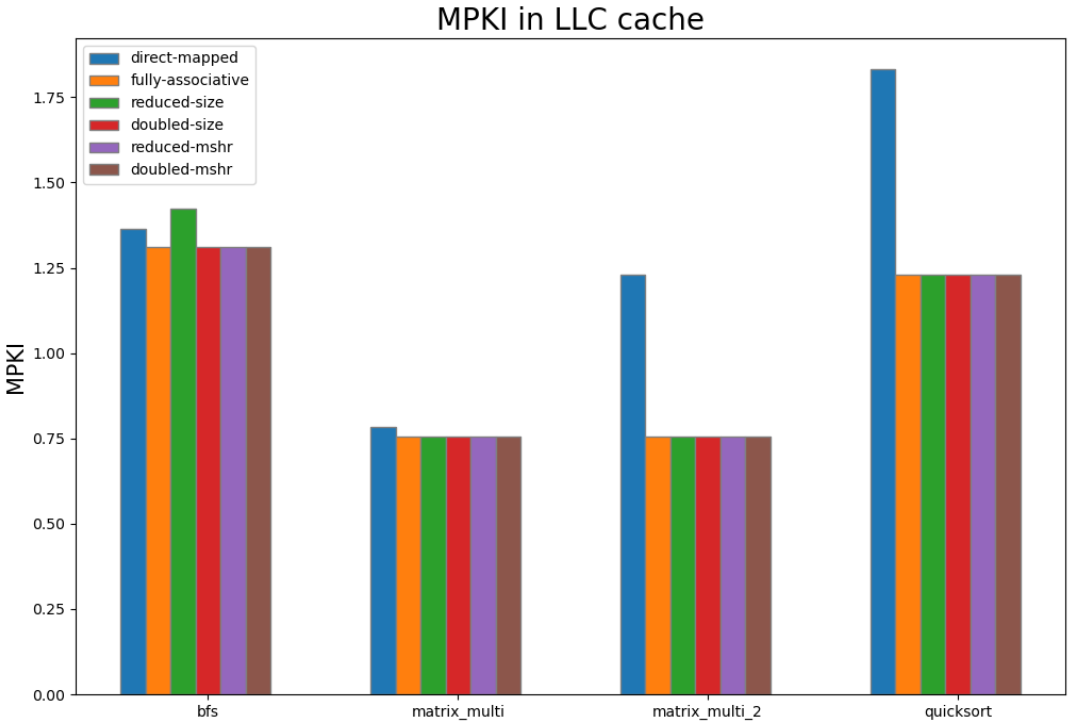


Figure 3.9: MPKI in LLC