Name: **Kunal Chand**
UB IT Name: **kchand**
Email ID: **kchand@buffalo.edu**
UB Person Number: **50465175**

Project (CSE 490/590 – Summer 2023)

Assigned Parameters:
456.hmmer

L1 data cache size,
L1 instruction cache size,
L1 data associativity,
L2 associativity,
Block Size

1) Baseline Command: csh gemTest.csh 456.hmmer baseline **1kB 1kB** 1kB **1** 1 **1 16**
csh gemTest.csh <benchmark name> <output name> **<L1D size> <L1I size>** <L2 size> **<L1D assoc>** <L1I assoc> **<L2 assoc> <Block Size>**

Parameter Varied: ***L1 data cache size <L1D size>***

Baseline Command: csh gemTest.csh 456.hmmer baseline **1kB** 1kB 1kB 1 1 1 16
2) csh gemTest.csh 456.hmmer L1D_size_2kB **2kB** 1kB 1kB 1 1 1 16
3) csh gemTest.csh 456.hmmer L1D_size_4kB **4kB** 1kB 1kB 1 1 1 16
4) csh gemTest.csh 456.hmmer L1D_size_8kB **8kB** 1kB 1kB 1 1 1 16
5) csh gemTest.csh 456.hmmer L1D_size_16kB **16kB** 1kB 1kB 1 1 1 16

Parameter Varied: ***L1 instruction cache size <L1I size>***

Baseline Command: csh gemTest.csh 456.hmmer baseline 1kB **1kB** 1kB 1 1 1 16
6) csh gemTest.csh 456.hmmer L1I_size_2kB 1kB **2kB** 1kB 1 1 1 16
7) csh gemTest.csh 456.hmmer L1I_size_4kB 1kB **4kB** 1kB 1 1 1 16
8) csh gemTest.csh 456.hmmer L1I_size_8kB 1kB **8kB** 1kB 1 1 1 16
9) csh gemTest.csh 456.hmmer L1I_size_16kB 1kB **16kB** 1kB 1 1 1 16

Parameter Varied: ***L1 data associativity <L1D assoc>***

Baseline Command: csh gemTest.csh 456.hmmer baseline 1kB 1kB 1kB **1** 1 1 16
10) csh gemTest.csh 456.hmmer L1D_assoc_2 1kB 1kB 1kB **2** 1 1 16
11) csh gemTest.csh 456.hmmer L1D_assoc_4 1kB 1kB 1kB **4** 1 1 16
12) csh gemTest.csh 456.hmmer L1D_assoc_8 1kB 1kB 1kB **8** 1 1 16
13) csh gemTest.csh 456.hmmer L1D_assoc_16 1kB 1kB 1kB **16** 1 1 16

Parameter Varied: ***L2 associativity <L2 assoc>***

Baseline Command: csh gemTest.csh 456.hmmer baseline 1kB 1kB 1kB 1 1 **1** 16
14) csh gemTest.csh 456.hmmer L2_assoc_2 1kB 1kB 1kB 1 1 **2** 16
15) csh gemTest.csh 456.hmmer L2_assoc_4 1kB 1kB 1kB 1 1 **4** 16
16) csh gemTest.csh 456.hmmer L2_assoc_8 1kB 1kB 1kB 1 1 **8** 16
17) csh gemTest.csh 456.hmmer L2_assoc_16 1kB 1kB 1kB 1 1 **16** 16

Parameter Varied: ***Block Size <Block Size>***

Baseline Command: csh gemTest.csh 456.hmmer baseline 1kB 1kB 1kB 1 1 1 **16**
18) csh gemTest.csh 456.hmmer Block_Size_32 1kB 1kB 1kB 1 1 1 **32**
19) csh gemTest.csh 456.hmmer Block_Size_64 1kB 1kB 1kB 1 1 1 **64**

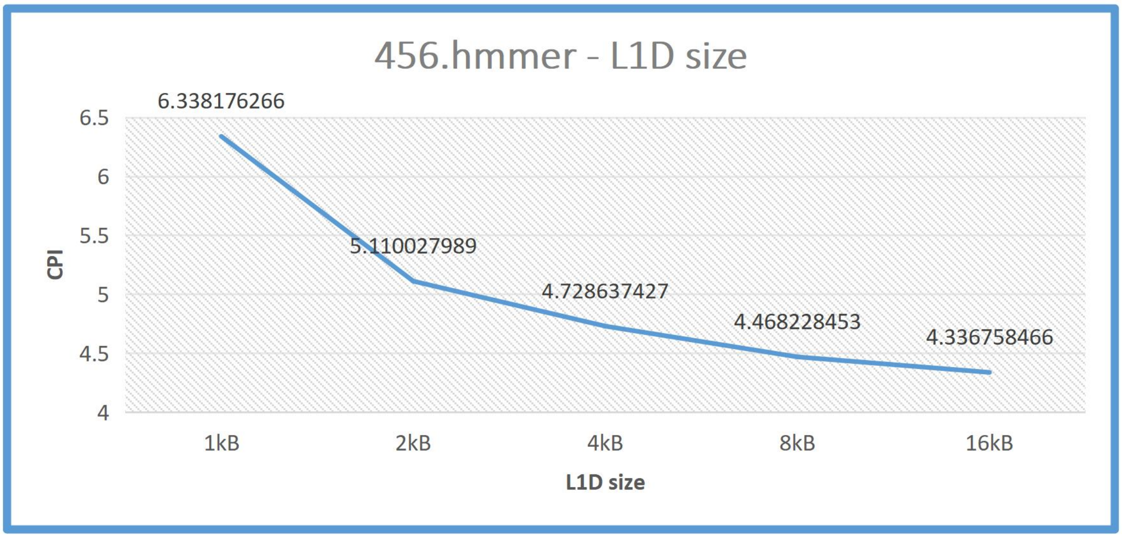
20) csh gemTest.csh 456.hmmer Block_Size_128 1kB 1kB 1kB 1 1 1 **128**
21) csh gemTest.csh 456.hmmer Block_Size_256 1kB 1kB 1kB 1 1 1 **256**

Following is the tabular result for all the executions:

benchmark	command #	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	1	baseline	NONE	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	2	L1D size	2kB	0.056503	0.081279	0.459249	290403	1128686	651715	10000001	0.943497	0.918721	0.540751	5.110027989
	3		4kB	0.040903	0.081279	0.436965	210227	1128686	585058	10000001	0.959097	0.918721	0.563035	4.728637427
	4		8kB	0.031036	0.081279	0.418462	159512	1128686	539062	10000001	0.968964	0.918721	0.581538	4.468228453
	5		16kB	0.025539	0.081279	0.409666	131262	1128686	516158	10000001	0.974461	0.918721	0.590334	4.336758466
	6	L1I size	2kB	0.109233	0.023824	0.568587	561417	330834	507322	10000001	0.890767	0.976176	0.431413	4.071960293
	7		4kB	0.109233	0.003046	0.661753	561417	42299	399511	10000001	0.890767	0.996954	0.338247	3.359784364
	8		8kB	0.109233	0.002035	0.661532	561417	28254	390086	10000001	0.890767	0.997965	0.338468	3.30423237
	9		16kB	0.109233	0.001165	0.665615	561417	16175	384454	10000001	0.890767	0.998835	0.334385	3.268824973
	10	L1D assoc	2	0.058152	0.081279	0.484643	298881	1128686	691860	10000001	0.941848	0.918721	0.515357	5.315839768
	11		4	0.043427	0.081279	0.480108	223198	1128686	649050	10000001	0.956573	0.918721	0.519892	5.056379994
	12		8	0.042986	0.081279	0.488971	220935	1128686	659926	10000001	0.957014	0.918721	0.511029	5.109402189
	13		16	0.044933	0.081279	0.49487	230939	1128686	672838	10000001	0.955067	0.918721	0.50513	5.179964582
	14	L2 assoc	2	0.109233	0.081279	0.512826	561417	1128686	866729	10000001	0.890767	0.918721	0.487174	6.347706265
	15		4	0.109233	0.081279	0.479984	561417	1128686	811222	10000001	0.890767	0.918721	0.520016	6.070171293
	16		8	0.109233	0.081279	0.45405	561417	1128686	767392	10000001	0.890767	0.918721	0.54595	5.851021315
	17		16	0.109233	0.081279	0.449907	561417	1128686	760390	10000001	0.890767	0.918721	0.550093	5.816011318
	18	Block Size	32	0.133374	0.050478	0.527596	681322	700969	729291	10000001	0.866626	0.949522	0.472404	5.475829152
	19		64	0.155224	0.028658	0.568678	792831	397956	677174	10000001	0.844776	0.971342	0.431322	5.10034179
	20		128	0.203618	0.019948	0.575491	1039811	277006	757816	10000001	0.796382	0.980052	0.424509	5.579169742
	21		256	0.240379	0.013474	0.610688	1227535	187108	863906	10000001	0.759621	0.986526	0.389312	6.168315283

<u>L1 data cache size <L1D size>:</u>

benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline L1D size	1kB	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	L1D size	2kB	0.056503	0.081279	0.459249	290403	1128686	651715	10000001	0.943497	0.918721	0.540751	5.110027989
		4kB	0.040903	0.081279	0.436965	210227	1128686	585058	10000001	0.959097	0.918721	0.563035	4.728637427
		8kB	0.031036	0.081279	0.418462	159512	1128686	539062	10000001	0.968964	0.918721	0.581538	4.468228453
		16kB	0.025539	0.081279	0.409666	131262	1128686	516158	10000001	0.974461	0.918721	0.590334	4.336758466



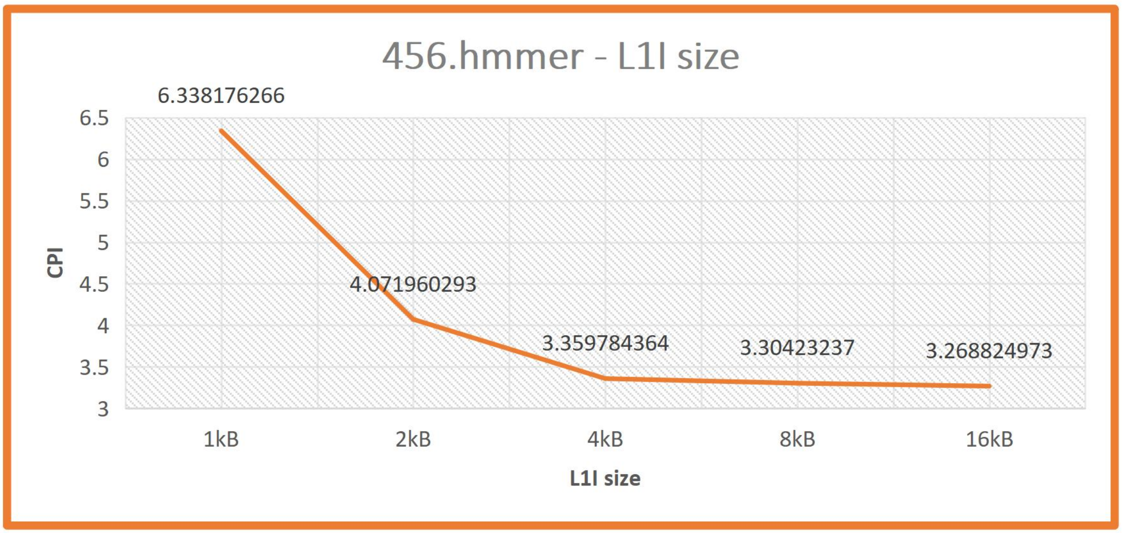
<u>Explanation (Observation):</u>
We can be observe that as the L1 data cache size increases, the CPI (cycles per instruction) decreases. This implies that larger L1 data cache sizes lead to improved performance.
Graphically, when plotting the L1 data cache size (in kilobytes) on the x-axis and the corresponding CPI values on the y-axis, the graph shows a decreasing trend. The graph would start with a higher CPI value at a smaller cache size (1kB) and gradually decrease as the cache size increases (2kB, 4kB, 8kB, and 16kB). The rate of decrease in CPI vary at different points, with larger cache size increments resulting in smaller performance gains.

<u>Explanation (Reasoning):</u>
The CPI (cycles per instruction) decreased as the L1 data cache size increased because the processor was able to find the data it needed more quickly in the larger cache. When the cache size was 1kB, the processor had to access main memory more often, which resulted in a higher CPI. As the cache

size increased, the processor was able to find the data it needed more often in the cache, which reduced the number of main memory accesses and resulted in a lower CPI.
The trend is consistent with the principle of locality of reference, which states that data that is accessed recently or frequently is likely to be accessed again in the near future. By increasing the L1 data cache size, the processor was able to store more of the data that was likely to be accessed, which reduced the number of main memory accesses and improved performance.

L1 instruction cache size <L1I size>:

benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline L1I size	1kB	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	L1I size	2kB	0.109233	0.023824	0.568587	561417	330834	507322	10000001	0.890767	0.976176	0.431413	4.071960293
		4kB	0.109233	0.003046	0.661753	561417	42299	399511	10000001	0.890767	0.996954	0.338247	3.359784364
		8kB	0.109233	0.002035	0.661532	561417	28254	390086	10000001	0.890767	0.997965	0.338468	3.30423237
		16kB	0.109233	0.001165	0.665615	561417	16175	384454	10000001	0.890767	0.998835	0.334385	3.268824973

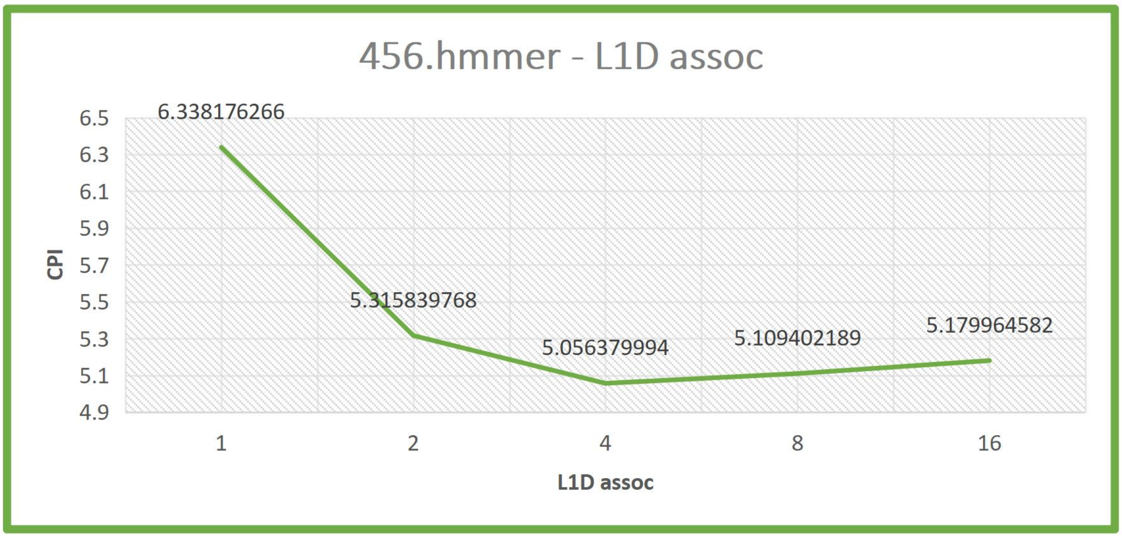


Explanation (Observation):
We can be observe that increasing the L1 instruction cache size leads to a significant improvement in performance, as indicated by the decreasing CPI values.
The graph represents the relationship between L1 instruction cache size (in kilobytes) and CPI exhibits a clear downward trend. As the cache size increases from 1kB to 16kB, the CPI consistently decreases, indicating better performance. The rate of improvement vary at different cache size increments, with larger cache size increases resulting in more significant performance gains initially and diminishing returns at larger cache sizes.

Explanation (Reasoning):
As the L1 instruction cache size increases, the CPI decreases. This is because the L1 instruction cache is responsible for storing the instructions that the processor is currently executing. A larger L1 instruction cache means more instructions can be stored and hence there is a higher chance that the instruction the processor needs is already in the cache, which can significantly reducing the frequency of instruction cache misses and subsequent stalls in the processor's pipeline. Consequently, the CPI decreases, indicating that instructions are fetched and executed more efficiently, leading to improved overall performance.

L1 data associativity <L1D assoc>:

benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline L1D assoc	1	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	L1D assoc	2	0.058152	0.081279	0.484643	298881	1128686	691860	10000001	0.941848	0.918721	0.515357	5.315839768
		4	0.043427	0.081279	0.480108	223198	1128686	649050	10000001	0.956573	0.918721	0.519892	5.056379994
		8	0.042986	0.081279	0.488971	220935	1128686	659926	10000001	0.957014	0.918721	0.511029	5.109402189
		16	0.044933	0.081279	0.49487	230939	1128686	672838	10000001	0.955067	0.918721	0.50513	5.179964582



Explanation (Observation):
Changing the L1 data cache associativity has a moderate impact on performance, as indicated by the slight variations in CPI values. Increasing the data cache associativity from 1 to 2 initially leads to a slight improvement in performance, as indicated by the decreased CPI. However, as the associativity further increases to 4, 8, and 16, the CPI values fluctuate within a relatively narrow range without a clear trend of improvement or degradation.

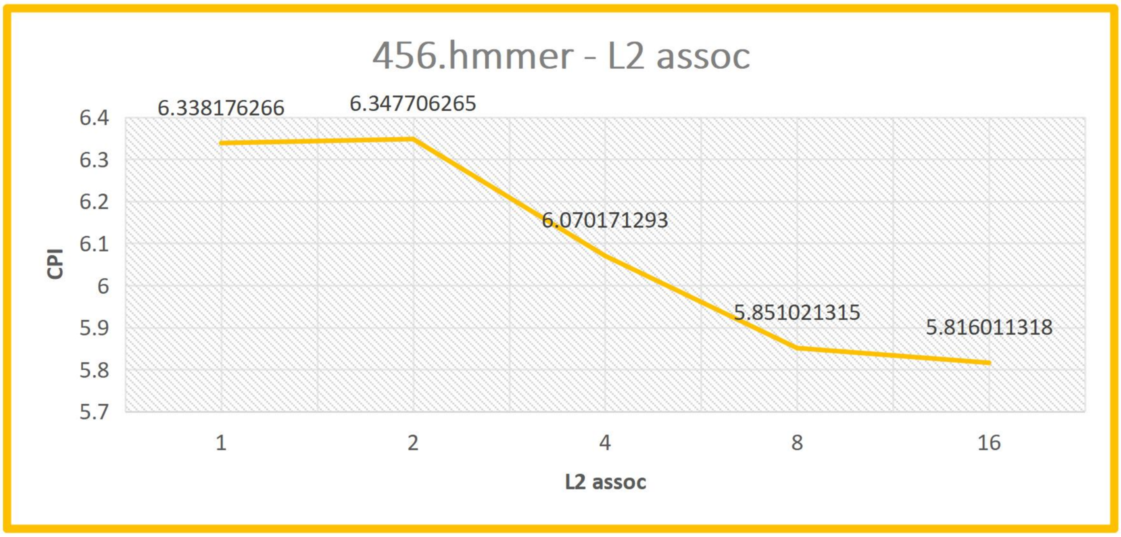
Explanation (Reasoning):
Data associativity refers to the number of cache sets that can store a particular data item. Higher associativity allows for a larger number of cache sets and reduces the likelihood of cache conflicts.

When the L1 data associativity is low, such as 1 or 2, the cache sets are limited, resulting in a higher probability of cache conflicts. Cache conflicts occur when multiple memory blocks compete for the same cache set, leading to evictions and frequent cache misses. As a result, the processor needs to access the main memory more frequently, incurring higher latency and increasing the number of cycles required to complete instructions, thus elevating the CPI. As the L1 data associativity increases to 4, 8, and 16, the cache sets become more numerous, reducing the likelihood of cache conflicts. With more cache sets available, data blocks have a higher chance of finding a vacant set, leading to fewer evictions and cache misses. Consequently, the processor can access data more efficiently from the cache, reducing memory latency and the number of cycles needed to complete instructions. This results in a decrease in CPI as the L1 data associativity increases.

Increasing the L1 data associativity reduces the likelihood of cache conflicts, because with more cache sets available, data blocks have a higher chance of finding a vacant set. However, there is a point of diminishing returns, where increasing the associativity further does not have a significant impact on the number of cache conflicts. This is because even with a high associativity, there will still be some cases where two data blocks will compete for the same cache set. Also, increasing the associativity also increases the complexity of the cache hardware, as the cache controller needs to keep track of which data blocks are stored in each cache set. So, as the associativity increases, the number of possible combinations of data blocks in each cache set increases, which makes it more difficult for the cache controller to track. This additional complexity can also lead to a small variation of CPI in the later stages.

L2 associativity <L2 assoc>:

benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline L2 assoc	1	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	L2 assoc	2	0.109233	0.081279	0.512826	561417	1128686	866729	10000001	0.890767	0.918721	0.487174	6.347706265
		4	0.109233	0.081279	0.479984	561417	1128686	811222	10000001	0.890767	0.918721	0.520016	6.070171293
		8	0.109233	0.081279	0.45405	561417	1128686	767392	10000001	0.890767	0.918721	0.54595	5.851021315
		16	0.109233	0.081279	0.449907	561417	1128686	760390	10000001	0.890767	0.918721	0.550093	5.816011318

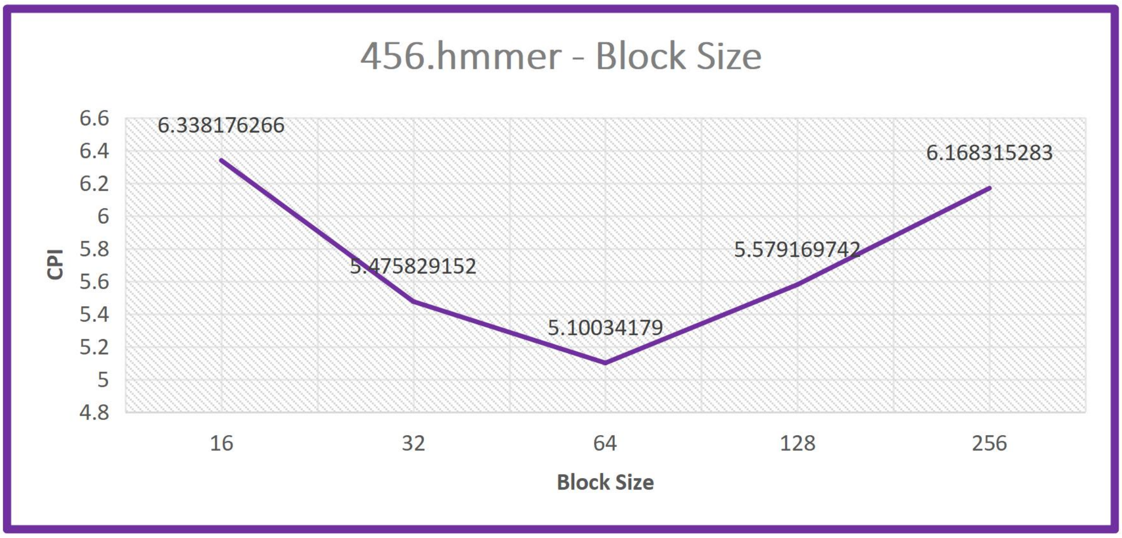


Explanation (Observation):
We can be observe that changing the L2 cache associativity has a limited impact on performance, as indicated by the slight variations in CPI values.
The graph representing the relationship between L2 cache associativity and CPI shows a relatively flat trend. The CPI values does show minor fluctuations as the associativity increases, but the changes are not significant enough.

Explanation (Reasoning):
L2 associativity refers to the number of cache lines or blocks that can be mapped to each set in the L2 cache. A higher associativity allows for a larger number of cache lines to be mapped to each set, reducing the likelihood of cache conflicts and improving cache hit rates. As the L2 associativity increases, cache conflicts decrease, resulting in improved cache performance. This, in turn, reduces the number of cycles required to fetch data from the L2 cache, thereby lowering the CPI.
With lower associativity, cache conflicts may occur more frequently, leading to longer access times and increased CPI. The decreasing trend in CPI as L2 associativity increases indicates that the x86 processor benefits from larger and more associative L2 caches, allowing for more efficient data retrieval and improved overall performance.
However, increasing associativity also introduces increased complexity and potential latency in cache access, which might counterbalance the benefits.

Block Size <Block Size>:

benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline Block Size	16	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	Block Size	32	0.133374	0.050478	0.527596	681322	700969	729291	10000001	0.866626	0.949522	0.472404	5.475829152
		64	0.155224	0.028658	0.568678	792831	397956	677174	10000001	0.844776	0.971342	0.431322	5.10034179
		128	0.203618	0.019948	0.575491	1039811	277006	757816	10000001	0.796382	0.980052	0.424509	5.579169742
		256	0.240379	0.013474	0.610688	1227535	187108	863906	10000001	0.759621	0.986526	0.389312	6.168315283

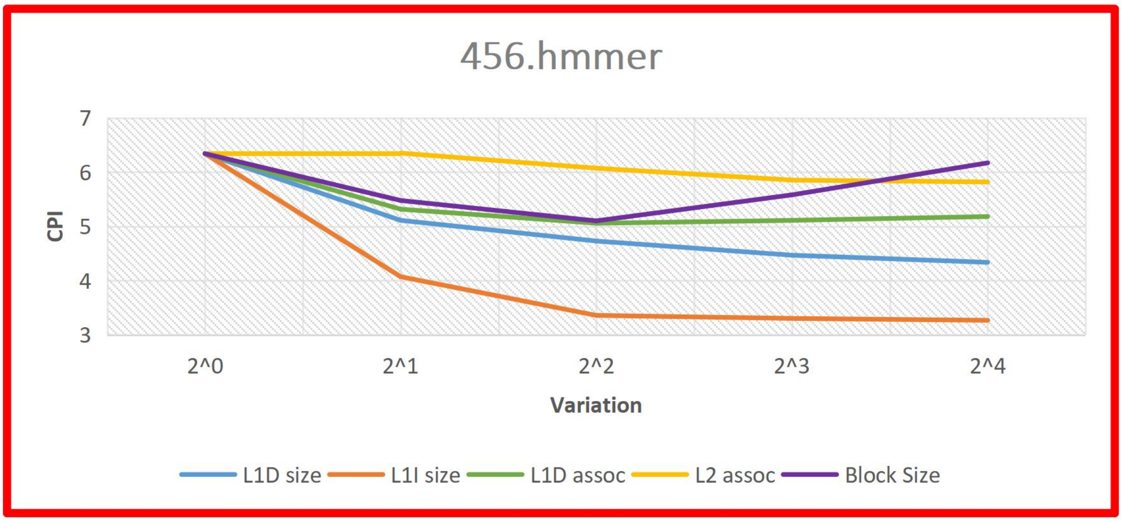


<u>Explanation (Observation):</u>
The trend observed in the CPI values as the block size changes indicate that there is an optimal block size (64) that minimizes cache misses and improves performance. Increasing the block size initially leads to reduced CPI as it allows more data to be fetched at once, reducing cache misses. However, if the block size becomes too large, it can lead to cache conflicts and contention, resulting in increased CPI and degraded performance.

<u>Explanation (Reasoning):</u>
The larger the block size, the less likely it is that a cache miss will occur. This is because a larger block size means that more data is stored in each cache line, which means that there is a smaller chance that the next data access will require a cache miss. As a result, the CPI decreases as the block size increases. However, there is a trade-off to consider. A larger block size also means that more memory is required to store the cache lines. This can lead to a decrease in the overall performance of the system if the memory bandwidth is not sufficient to keep the cache lines filled.
Spatial locality is an important factor to explain why the values are how they are. Spatial locality refers to the tendency of data that is accessed together to be located in close proximity in memory. This means that if a program accesses a certain piece of data, it is likely that it will also access other pieces of data that are located nearby. A larger block size allows the processor to take advantage of spatial locality by caching more data that is likely to be accessed together. This reduces the number of cache misses and improves the overall performance of the system.
While spatial locality plays a significant role in explaining the initial improvement in performance as the block size increases, it may not fully account for the subsequent increase in CPI after a certain point. Other factors come into play when the block size becomes excessively large. When the block size is increased beyond a certain threshold, the negative effects of a larger block size can start to outweigh the benefits of improved spatial locality. One of the potential reasons for the increasing CPI is the impact on temporal locality.
Temporal locality refers to the tendency of a program to access the same data repeatedly over a short period of time. With larger block sizes, a single cache line now holds more data elements. In such cases, a larger block size can lead to an increase in cache pollution, where cache lines are occupied by data that is not frequently reused. As a result, the cache may become filled with less relevant data, reducing the number of available cache lines for storing frequently accessed data. This can lead to an increase in cache misses and a subsequent increase in CPI.

Comparision:

Variation	L1D size CPI	L1I size CPI	L1D assoc CPI	L2 assoc CPI	Block Size CPI
2^0	6.338176266	6.338176266	6.338176266	6.338176266	6.338176266
2^1	5.110027989	4.071960293	5.315839768	6.347706265	5.475829152
2^2	4.728637427	3.359784364	5.056379994	6.070171293	5.10034179
2^3	4.468228453	3.30423237	5.109402189	5.851021315	5.579169742
2^4	4.336758466	3.268824973	5.179964582	5.816011318	6.168315283



Explanation (Observation):
Since the value for each parameter was different, the x-axis consists of relative increment of the respective values. And the y-axis represents the CPI values.
We can clearly see that modifying the **L1I size** parameter positively affects the CPI a lot more compared to other parameters.

Explanation (Reasoning):
The L1I cache size is a more important factor in CPI than the other parameters. This is because instructions are typically smaller than data, they are accessed more frequently, and the L1I cache is closer to the CPU core. Other parameters have less of an impact on CPI because they do not affect the performance of the CPU as directly as L1I size does.

Step-by-step explanation of how I completed this project:

Github link to all the files that I created and used to complete this project:
https://github.com/kunalchand/CSE590_Project/tree/dev

Step 1: I first wrote all the commands that I have to execute and varied the parameter one by one.

Assigned Parameters:
456.hmmer

- ***L1 data cache size,***
- ***L1 instruction cache size,***
- ***L1 data associativity,***
- ***L2 associativity,***
- ***Block Size***

1) Baseline Command: csh gemTest.csh 456.hmmer baseline **1kB 1kB** 1kB **1** 1 **1 16**
csh gemTest.csh <benchmark name> <output name> **<L1D size> <L1I size>** <L2 size> **<L1D assoc>** <L1I assoc> **<L2 assoc> <Block Size>**

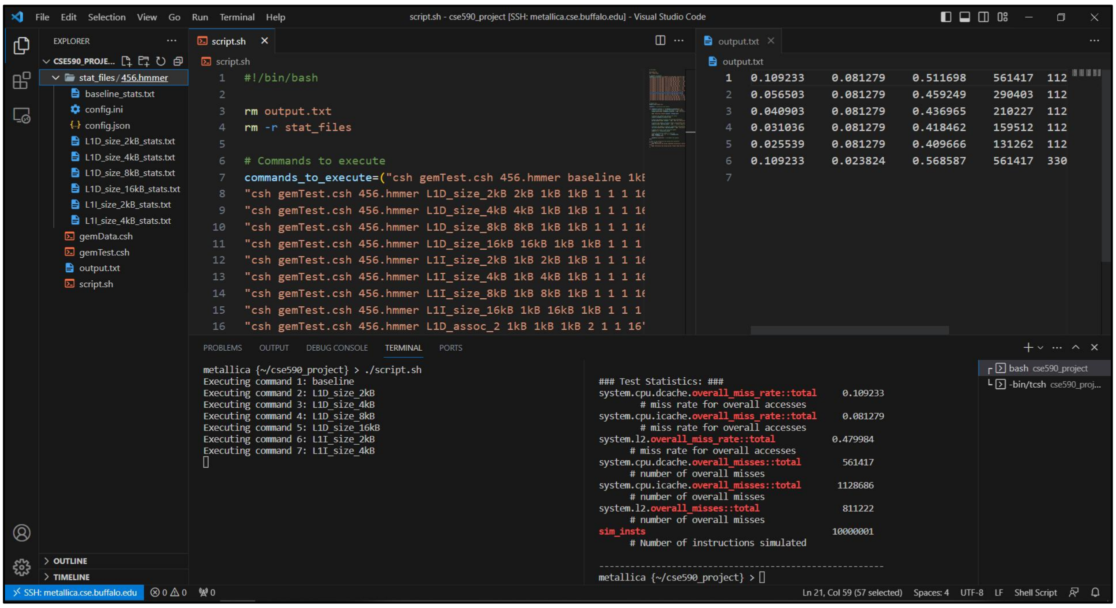
Parameter Varied: *L1 data cache size <L1D size>*

Baseline Command: csh gemTest.csh 456.hmmer baseline **1kB** 1kB 1kB 1 1 1 16
2) csh gemTest.csh 456.hmmer L1D_size_2kB **2kB** 1kB 1kB 1 1 1 16
3) csh gemTest.csh 456.hmmer L1D_size_4kB **4kB** 1kB 1kB 1 1 1 16
4) csh gemTest.csh 456.hmmer L1D_size_8kB **8kB** 1kB 1kB 1 1 1 16
5) csh gemTest.csh 456.hmmer L1D_size_16kB **16kB** 1kB 1kB 1 1 1 16

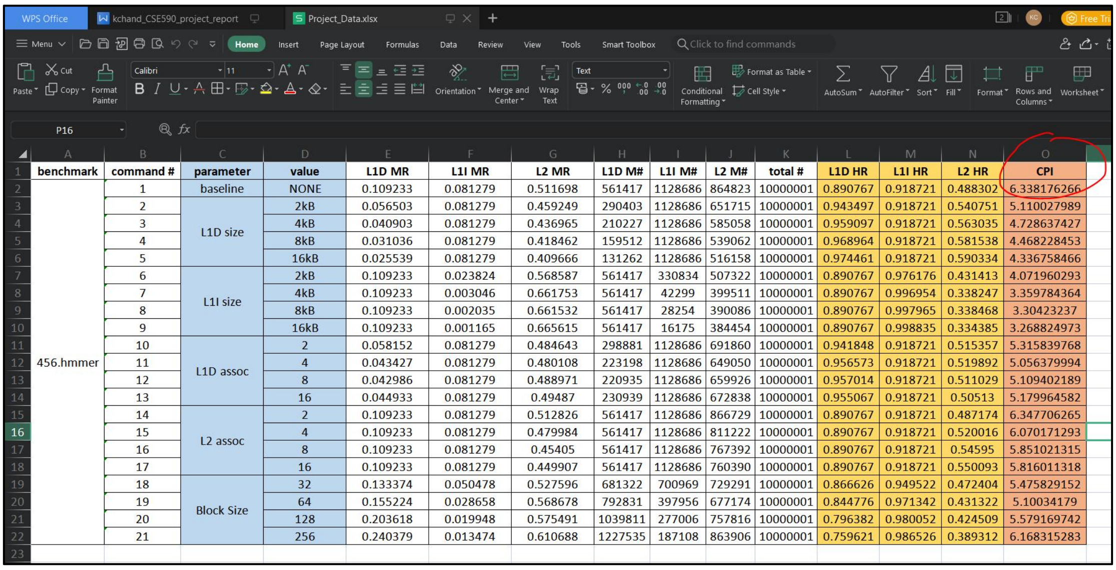
Parameter Varied: *L1 instruction cache size <L1I size>*

Baseline Command: csh gemTest.csh 456.hmmer baseline 1kB **1kB** 1kB 1 1 1 16
6) csh gemTest.csh 456.hmmer L1I_size_2kB 1kB **2kB** 1kB 1 1 1 16
7) csh gemTest.csh 456.hmmer L1I_size_4kB 1kB **4kB** 1kB 1 1 1 16

Step 2: I wanted to get rid of the manual job of copy pasting the terminal output in excel after executing every command one by one. So I created a script.sh file which executed all the 21 commands for me and dump the output values in output.txt file in one line after the other. In parallel, the stats files got generated too.

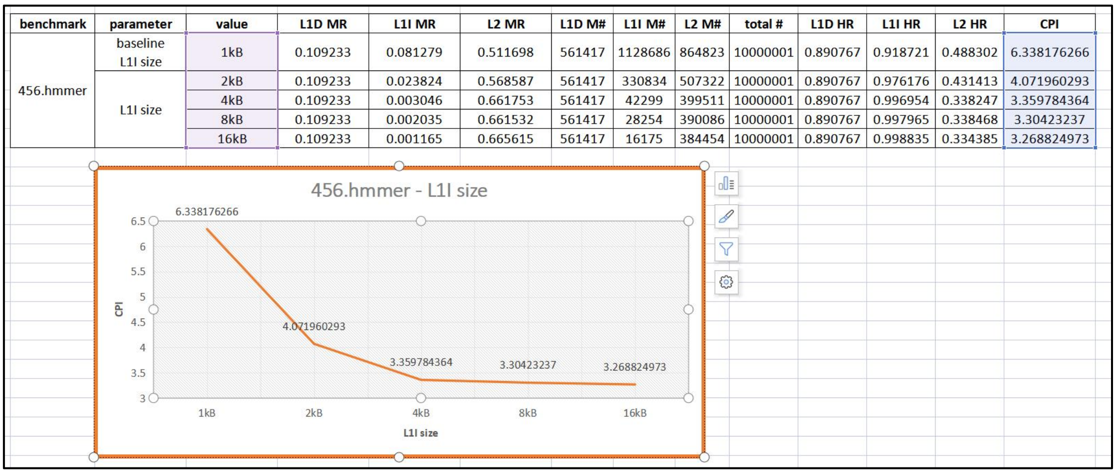


Step 3: Next I copy pasted the generated output values in the excel and used the formula to calculate the CPI value for each test case. I also calculated L1D HR, L1I HR and L2 HR, as they were part of the project requirement.



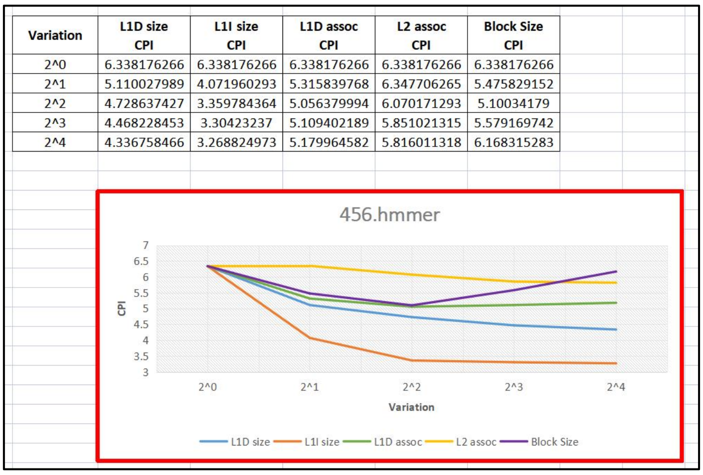
benchmark	command #	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	1	baseline	NONE	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	2	L1D size	2kB	0.056503	0.081279	0.459249	290403	1128686	651715	10000001	0.943497	0.918721	0.540751	5.110027989
	3		4kB	0.040903	0.081279	0.436965	210227	1128686	585058	10000001	0.959097	0.918721	0.563035	4.728637427
	4		8kB	0.031036	0.081279	0.418462	159512	1128686	539062	10000001	0.968964	0.918721	0.581538	4.468228453
	5		16kB	0.025539	0.081279	0.409666	131262	1128686	516158	10000001	0.974461	0.918721	0.590334	4.336758466
	6	L1I size	2kB	0.109233	0.023824	0.568587	561417	330834	507322	10000001	0.890767	0.976176	0.431413	4.071960293
	7		4kB	0.109233	0.003046	0.661753	561417	42299	399511	10000001	0.890767	0.996954	0.338247	3.359784364
	8		8kB	0.109233	0.002035	0.661532	561417	28254	390086	10000001	0.890767	0.997965	0.338468	3.30423237
	9		16kB	0.109233	0.001165	0.665615	561417	16175	384454	10000001	0.890767	0.998835	0.334385	3.268824973
	10	L1D assoc	2	0.058152	0.081279	0.484643	298881	1128686	691860	10000001	0.941848	0.918721	0.515357	5.315839768
	11		4	0.043427	0.081279	0.480108	223198	1128686	649050	10000001	0.956573	0.918721	0.519892	5.056379994
	12		8	0.042986	0.081279	0.488971	220935	1128686	659926	10000001	0.957014	0.918721	0.511029	5.109402189
	13		16	0.044933	0.081279	0.49487	230939	1128686	672838	10000001	0.955067	0.918721	0.50513	5.179964582
	14	L2 assoc	2	0.109233	0.081279	0.512826	561417	1128686	866729	10000001	0.890767	0.918721	0.487174	6.347706265
	15		4	0.109233	0.081279	0.479984	561417	1128686	811222	10000001	0.890767	0.918721	0.520016	6.070171293
	16		8	0.109233	0.081279	0.45405	561417	1128686	767392	10000001	0.890767	0.918721	0.54595	5.851021315
	17		16	0.109233	0.081279	0.449907	561417	1128686	760390	10000001	0.890767	0.918721	0.550093	5.816011318
	18	Block Size	32	0.133374	0.050478	0.527596	681322	700969	729291	10000001	0.866626	0.949522	0.472404	5.475829152
	19		64	0.155224	0.028658	0.568678	792831	397956	677174	10000001	0.844776	0.971342	0.431322	5.10034179
	20		128	0.203618	0.019948	0.575491	1039811	277006	757816	10000001	0.796382	0.980052	0.424509	5.579169742
	21		256	0.240379	0.013474	0.610688	1227535	187108	863906	10000001	0.759621	0.986526	0.389312	6.168315283

Step 4: Next I plotted graph of CPI vs Value of each parameter.



benchmark	parameter	value	L1D MR	L1I MR	L2 MR	L1D M#	L1I M#	L2 M#	total #	L1D HR	L1I HR	L2 HR	CPI
456.hmmer	baseline L1I size	1kB	0.109233	0.081279	0.511698	561417	1128686	864823	10000001	0.890767	0.918721	0.488302	6.338176266
	L1I size	2kB	0.109233	0.023824	0.568587	561417	330834	507322	10000001	0.890767	0.976176	0.431413	4.071960293
		4kB	0.109233	0.003046	0.661753	561417	42299	399511	10000001	0.890767	0.996954	0.338247	3.359784364
		8kB	0.109233	0.002035	0.661532	561417	28254	390086	10000001	0.890767	0.997965	0.338468	3.30423237
		16kB	0.109233	0.001165	0.665615	561417	16175	384454	10000001	0.890767	0.998835	0.334385	3.268824973

Step 5: At last, I combined all the graphs to compare and see which parameter variation was the most effective in improving the CPI.



Variation	L1D size CPI	L1I size CPI	L1D assoc CPI	L2 assoc CPI	Block Size CPI
2^0	6.338176266	6.338176266	6.338176266	6.338176266	6.338176266
2^1	5.110027989	4.071960293	5.315839768	6.347706265	5.475829152
2^2	4.728637427	3.359784364	5.056379994	6.070171293	5.10034179
2^3	4.468228453	3.30423237	5.109402189	5.851021315	5.579169742
2^4	4.336758466	3.268824973	5.179964582	5.816011318	6.168315283