CS 773: Assignment-1 Task-2

Approach

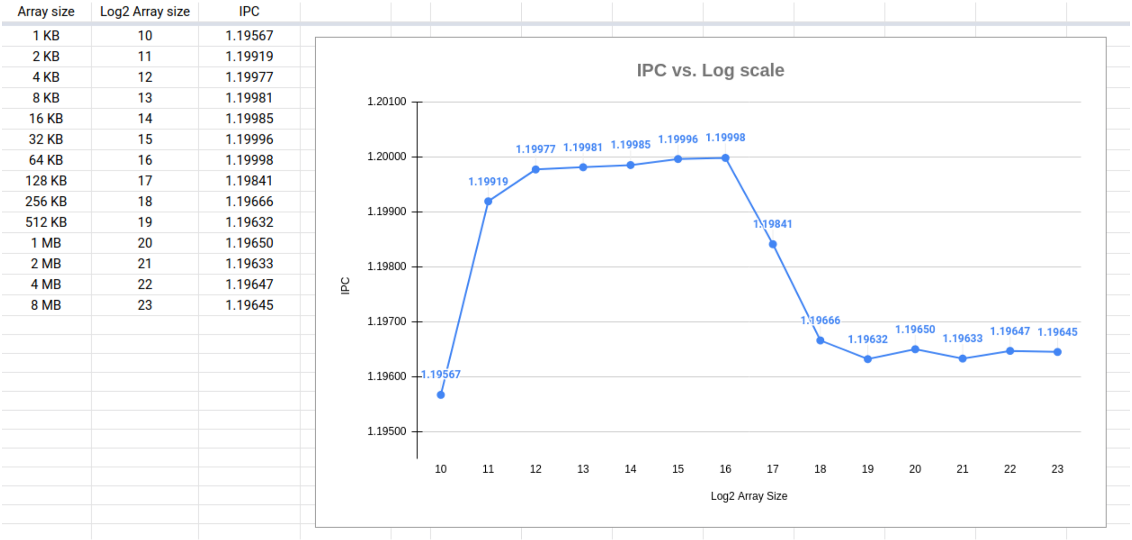
- Below is the code for this task:

```
// CS773: Assignment-1 Task-2

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define KB_BYTES 1024       // 1KiB (Kibibyte) or 1KB = 1024 bytes

void mem_access(int arr_size)
{
    int arr_len_bytes = arr_size * KB_BYTES;
    char arr[arr_len_bytes];        // array of size in multiple of 1024
bytes (1 KiB data)
    int k;

    // accessing the same array multiple times to increase the code size
    // (specifically for smaller arrays) and hence number of instructions
    // for trace file generation
    for ( int j = 0; j < 1000; j++ )
    {
        for ( int i = 0; i < arr_len_bytes; i++ )
        {
            k = arr[i];
        }
    }
}

int main(int argc, char const *argv[])
{
    int arr_size = pow(2, atoi( argv[1]) ) / 1024;

    if ( arr_size >= 1024 )
        printf("Array size input: %d MB or MiB\n", arr_size / 1024);
    else
        printf("Array size input: %d KB or KiB\n", arr_size);

    // execute the function mem_access to access the array of given size
input
    mem_access(arr_size);

    return 0;
}
```

- The **log2** value of various array sizes is provided as input to the code as command-line argument.
- The array size in unit of **KB** is passed to `mem_access` function.
- The function computes the array length (in bytes) required for given input of array size and creates a *char* array of that length.
- It then accesses these bytes one-by-one in a loop.

> **Note:**
>
> - For **smaller array sizes** (up till *1 MB*), the trace file generated by Pin tool (along with skipping of 10 M instructions and generate next 2 M instructions) had size of **0 bytes**.
> - I think this is due to the lesser number of instructions (LOAD, in this case) that get skipped.
> - Hence added another for loop to access the same array multiple times (just to increase the number of instructions).

- Trace files are generated using Pin tool and ran with ChampSim for different array sizes with input argument from **10 (1 KB)** to **23 (8 MB)**.
- IPC values are noted down and plotted against the log2 value of array sizes. Below is the plot for the same.



Array size	Log2 Array size	IPC
1 KB	10	1.19567
2 KB	11	1.19919
4 KB	12	1.19977
8 KB	13	1.19981
16 KB	14	1.19985
32 KB	15	1.19996
64 KB	16	1.19998
128 KB	17	1.19841
256 KB	18	1.19666
512 KB	19	1.19632
1 MB	20	1.19650
2 MB	21	1.19633
4 MB	22	1.19647
8 MB	23	1.19645

Findings

- With increasing array sizes, the instructions (LOAD, in this case) will increase and hence the increment is visible in IPC.
- Initially, there are no cache misses since the array size can easily fit into L1D cache.
- But at certain array size, the IPC drops and few cache misses are observed in L1D cache. This is due to the array can't be fit into the cache.
- Hence the array size before this drop is observed should be the actual cache size.
- Below is the actual cache size computed from the settings of ChampSim.
  - Block size found from `inc/champsim.h` file is **64**.

- Values in table are taken from `inc/cache.h` file.

Cache Level	L1D	L2	LLC
number of sets	64	1024	2048
number of ways	12	8	16
Actual Cache size = (# sets x # ways x block size)	**48 KB**	**512 KB**	**2048 KB = 2 MB**

- From the plot, the drops are observed at the below array sizes:
  - 1st drop
    - at 17 => 128 KB => **Measured L1D cache size = 64 KB**
  - 2nd drop
    - at 21 => 2 MB => **L2 cache size = 1 MB**
  - 3rd drop
    - at 23 => 8 MB => **LLC cache size = 4 MB**
- The measured cache size derived from the plot is **1 level more than** the actual cache size from ChampSim settings.

Thank you!