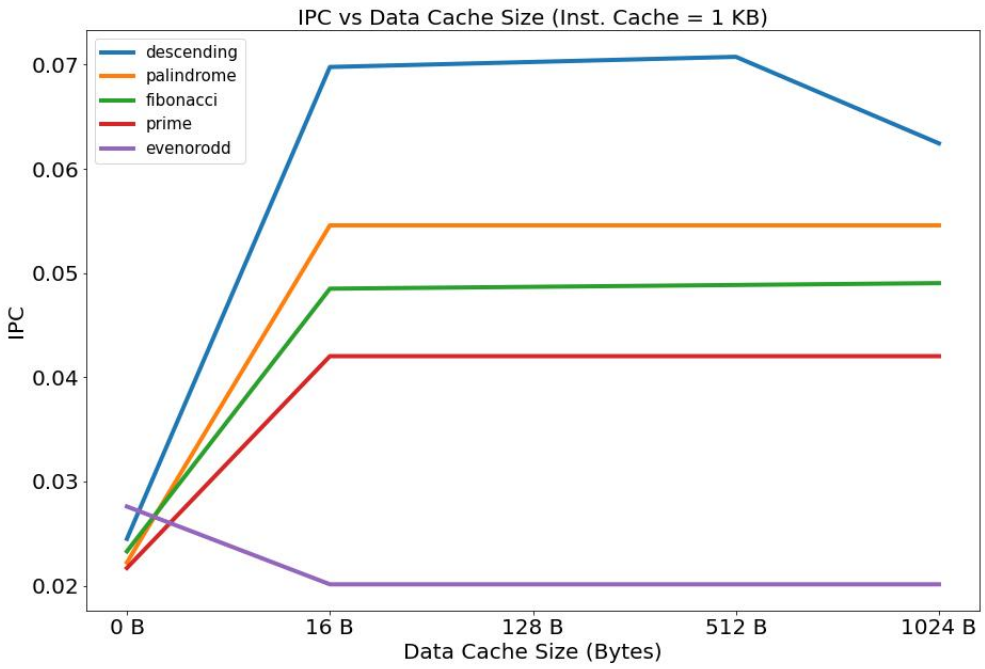
CS 311: Assignment 6

Harrithha (200010018), Shashank P (200010048)

September 2022

1 Data Cache

In this experiment, we have set the instruction cache to have a fixed size of $1KB$. On varying the size of the data cache from $16B$ to $1KB$, we obtain the following plots for **IPC**.



From the above plot, we can infer the following

1. The IPC increases for almost all the benchmarks on adding a data cache as the latency for data access is lower.

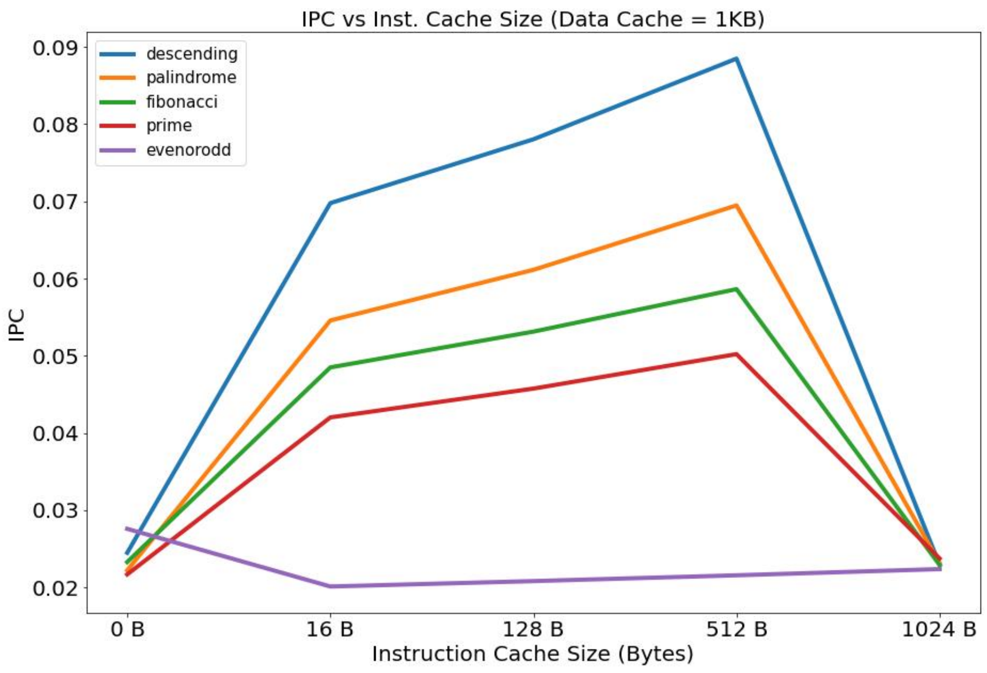
2. But for **evenorodd**, the latency decreases as there is only 1 load operation which will be a cache miss increasing latency. Here the cache does not play a major role.

3. The IPC increases the most for descending as there are a lot of reads and writes on an array. Once the first element is fetched, almost the entire array is also fetched. Therefore all the further load operations fetch it directly from the data cache.

4. We can see that after a point adding extra cache either has no effect on IPC or decreases the IPC as even though there is a lot of space, it is not being used by the program. Hence there is a sort of optimal balance between the size and latency of the cache, which is around $16B$

2 Instruction Cache

In this experiment, we have set the data cache to have a fixed size of $1KB$. On varying the size of the instruction cache from $16B$ to $1KB$, we obtain the following plots for **IPC**.



From the above plot, we can infer the following

1. The IPC increases for almost all the benchmarks on adding an instruction cache as the latency for data access is less.
2. Due to reasons mentioned in the previous section **evenorodd** has a reduced IPC.
3. The IPC increases the most for descending as the same set of instructions are executed in a loop multiple times. There is a very high chance that as the size of the cache increases, all the instructions are stored in the cache.
4. We can observe that benchmarks such as **palindrome** and **fibonacci** also have a higher IPC up to a point due to the presence of loops. There is an optimal size cache where there is a balance between the size and latency of the cache, which is around $512B$

3 Instruction Cache Benchmark

We have developed the following benchmark that has a significance **IPC** boost when the instruction cache size is increased from $16B$ to $128B$, keeping the data cache at $1KB$.

```
    .data
n:
    50
        .text
main:
    load %x0, $n, %x4
    addi %x0, 1, %x5
loop:
    beq %x5, %x4, return
    addi %x0, 1, %x3
    addi %x3, 2, %x3
    addi %x3, 3, %x3
    addi %x3, 4, %x3
    addi %x3, 5, %x3
    addi %x3, 6, %x3
    addi %x3, 7, %x3
    addi %x3, 8, %x3
    addi %x3, 9, %x3
    addi %x3, 10, %x3
    addi %x3, 11, %x3
    addi %x3, 12, %x3
    addi %x3, 13, %x3
    addi %x3, 14, %x3
    addi %x3, 15, %x3
```

```
     addi %x3, 16, %x3
     addi %x3, 17, %x3
     addi %x3, 18, %x3
     addi %x3, 19, %x3
     addi %x3, 20, %x3
     addi %x5, 1, %x5
     jmp loop
return:
     end
```

Here 17 instructions are repeatedly executed 50 times, while the $128B$ cache can easily store all the instructions, the $16B$ cache constantly replaces the instruction causing lower IPC. On experimentation, we obtained $IPC = 0.02327$ for $16B$ cache and $IPC = 0.20369$ for $128B$ cache, which is a significant improvement.

4 Data Cache Benchmark

We have developed the following benchmark that has a significance **IPC** boost when the data cache size is increased from $16B$ to $128B$, keeping the instruction cache at size $1KB$.

```
     .data
a:
     18
     50
     3
     4
     5
     6
     7
     8
     9
     10
     11
     12
     13
     14
     15
     16
     17
     18
          .text
main:
     load %x0, $a, %x3
     addi %x0, 1, %x8
```

```
    load %x8, $a, %x4
    addi %x0, 0, %x5
    addi %x0, 0, %x6
loop:
    beq %x5, %x3, inc
    load %x5, $a, %x7
    addi %x5, 1, %x5
    jmp loop
inc:
    beq %x6, %x4, return
    addi %x0, 0, %x5
    addi %x6, 1, %x6
    jmp loop
return:
    end
```

Here an array of size 18 is repeatedly loaded 50 times, while the $128B$ data cache can easily store all the elements in the array, the $16B$ cache constantly replaces the array data causing lower IPC. On experimentation, we obtained $IPC = 0.06601$ for $16B$ cache and $IPC = 0.12132$ for $128B$ cache, which is almost a 200% increment.