Computer Architecture Lab
Assignment 6

Om Patil (200010036)
Hrishikesh Pable (200010037)

November 2022

1 Introduction

This assignment involved adding a set associative cache to our pipelined processor simulator. The following caches have been implemented,

- One cache between the IF stage and the main memory. We call this the "level 1 instruction cache" or the `L1i-cache`.
- One cache between the MA stage and the main memory. We call this the "level 1 data cache" or the `L1d-cache`.

The configurations of the caches are as follows,

Cache Size	16B	128B	512B	1kB
Latency	1 Cycles	2 Cycles	3 Cycles	4 Cycles
Line Size	4B			
Associativity	2			
Write Policy	Write Through			

2 Performance vs `L1i-cache` Size

We fix the size of `L1d-cache` at 1kB and vary the size of `L1i-cache` from 16B to 1kB. The performance (IPC) against the benchmark is plotted in figure 1.

From the figure it can be noted that programs which have more branches are the most benefited by the cache as they exhibit the most temporal locality. The programs only benefit from increase in `L1i-cache` size till a point post which the performance declines slowly. This can be attributed to the fact that the programs cannot utilize the additional cache size after a point, however the cache latency increases with the increase in size.

Programs such as `evenorodd` which have almost no branches and very little instructions do not benefit much from the cache as they do not exhibit much temporal locality.

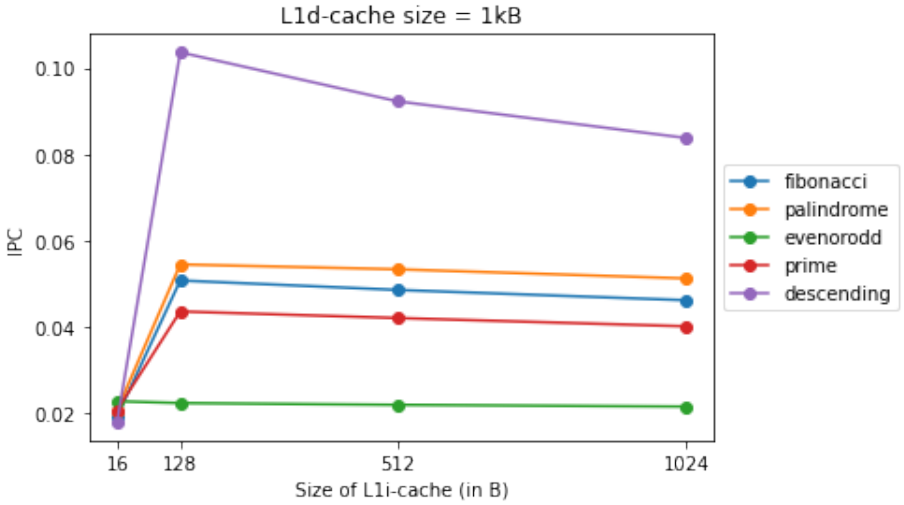

Figure 1: Performance vs `L1i-cache` Size

3 Performance vs `L1d-cache` Size

We fix the size of `L1i-cache` at 1kB and vary the size of `L1d-cache` from 16B to 1kB. The performance (IPC) against the benchmark is plotted in figure 2.

Programs which exhibit temporal locality in their memory calls benefit the most from the `L1d-cache`. The programs only benefit from increase in `L1i-cache` size till a point post which the performance declines slowly. This can be attributed to the fact that the programs cannot utilize the additional cache size after a point, however the cache latency increases with the increase in size.

Programs such as `palindrome`, `evenorodd`, `prime` and `fibonacci` exhibit little to no temporal locality in their data access calls and hence do not benefit much from the cache.

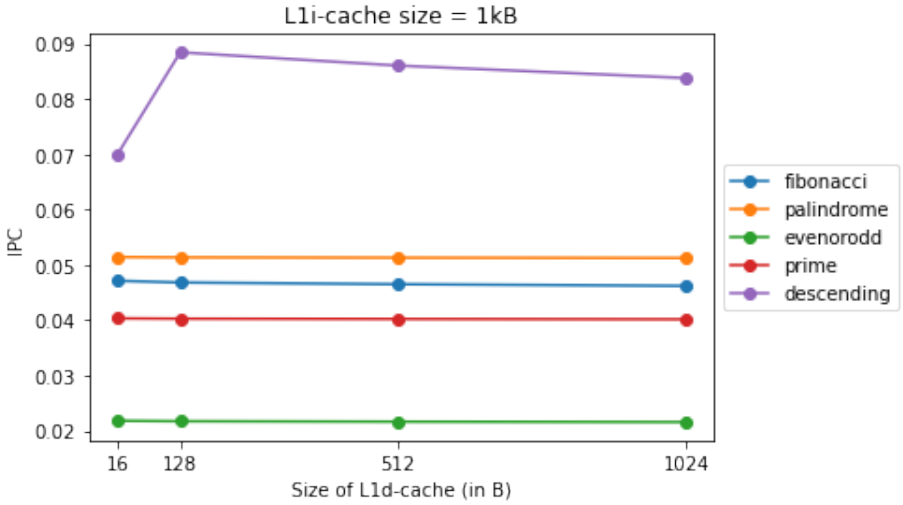

Figure 2: Performance vs `L1d-cache` Size

3.1 Toy Benchmark for L1i-Cache

```
 .data
a:
4567654
.text
main:
load %x0, $a, %x3
sub %x7, %x7, %x7
loop:  divi %x3, 10, %x4
addi %x31, 0, %x30
muli %x7, 10, %x7
add %x7, %x30, %x7
divi %x3, 10, %x3
bgt %x3, %x0, loop
load %x0, $a, %x5
beq %x5, %x7, palindrome
sub %x10, %x10, %x10
subi %x10, 1, %x10
end
palindrome:
sub %x10, %x10, %x10
addi %x10, 1, %x10
end
```

With `L1d-Cache` of size 1kB and increase in `L1i-Cache` size from 16B to 128B

the Toy benchmark performance (IPC) improves from 0.0207 to 0.0545.

3.2 Toy Benchmark for L1d-Cache

```
 .data
a:
40
20
50
60
80
30
10
70
n:
8
.text
main:
sub %x3, %x3, %x3
sub %x4, %x4, %x4
load %x0, $n, %x8
outerloop:
blt %x3, %x8, innerloop
end
addi %x3, 1, %x4
innerloop:
addi %x3, 1, %x4
innerloopz:
blt %x4, %x8, swap
addi %3, 1, %x3
jmp outerloop
swap:
load %x3, $a, %x5
load %x4, $a, %x6
blt %x5, %x6, exchange
addi %x4, 1, %x4
jmp innerloopz
exchange:
sub %x7, %x7, %x7
add %x0, %x5, %x7
store %x6, 0, %x3
store %x7, 0, %x4
addi %x4, 1, %x4
jmp innerloopz
```

With `L1i-Cache` of size 1kB and increase in `L1d-Cache` size from 16B to 128B

the Toy benchmark performance (IPC) improves from 0.0698 to 0.0884.