# CS-311 Assignment6

# *K.S.N Manikanta(210010050), M.Suresh(210010030)*

# Configurations Of Cache

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cache Size** | **16B** | **128B** | **512B** | **1KB** |
| **Latency** | 1 cycle | 2 cycles | 3 cycles | 4 cycles |
| **Line Size** | 4B | | | |
| **Associativity** | 2 | | | |
| **Write Policy** | Write Through | | | |

Table 1: Results and Observations

# The Analyses that need to be Done

* + First fix the size of the L1d-cache at 1KB. Vary the size of the L1i-cache from 16B to 1KB (remember to change latency accordingly), and study the performance (instructions per cycle). Plot your results for the different benchmarks.
  + Now fix the size of the L1i-cache at 1KB. Vary the size of the L1d-cache from 16B to 1KB, and plot your results for the different benchmarks.
  + In this report, we correlated the nature of the benchmark to the observed plot. In this assignment, we simulate the pipelined processor with data and instructions read from caches and main memory with corresponding latenices. We present the throughput in terms of Instructions per cycle (IPC) vs cache size for the benchmark programs.
  + Create a toy-benchmark that shows significant performance improvement when the L1i-cache is increased from 16B to 128B. Assume favorable L1d cache size.
  + Create a toy-benchmark that shows significant performance improvement when the L1d-cache is increased from 16B to 128B. Assume favorable L1i-cache size.

# IPC(Throughput) for different values of L1d and L1i cache

**Analysis of the IPC without caches**

|  |  |
| --- | --- |
| **Assembly File** | **IPC (without cache)** |
| **evenorodd.out** | 0.023166023 |
| **descending.out** | 0.023516428 |
| **prime.out** | 0.023789993 |
| **palindrome.out** | 0.023763336 |
| **fibonacci.out** | 0.021781625 |

**When the data cache size is constant (L1d = 1kB)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assembly File** | **L1i=16B** | **L1i=128B** | **L1i=512B** | **L1i=1KB** |
| evenorodd.out | 0.02238806 | 0.021978023 | 0.021582734 | 0.019543974 |
| descending.out | 0.025002256 | 0.11878216 | 0.10448887 | 0.07014434 |
| prime.out | 0.03243848 | 0.045383412 | 0.04347826 | 0.035758324 |
| palindrome.out | 0.030265596 | 0.07174231 | 0.063885264 | 0.048466865 |
| fibonacci.out | 0.022807017 | 0.053682037 | 0.04797048 | 0.037249282 |

**When the instruction cache size is constant (L1i = 1kB)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assembly File** | **L1d=16B** | **L1d=128B** | **L1d=512B** | **L1d=1KB** |
| evenorodd.out | 0.021428572 | 0.021352313 | 0.021276595 | 0.019543974 |
| descending.out | 0.0800578 | 0.10386202 | 0.100580975 | 0.07014434 |
| prime.out | 0.041907515 | 0.041786745 | 0.041786745 | 0.035758324 |
| palindrome.out | 0.060344826 | 0.060270604 | 0.06019656 | 0.048466865 |
| fibonacci.out | 0.048811015 | 0.047852762 | 0.04693141 | 0.037249282 |

# Graphs

# 

# 

1. **Observation and Analysis of the Graphs**

First, we vary the size of the L1i cache (instruction cache) while keeping the size of the L1d cache(data cache) constant at 1KB. The first graph displays how the IPC varies as we implement the above. When the size of the L1i is increased, we infer that the latency and hit rate also increase likewise. We also infer that the size of the L1i cache will be optimal when the IPC is the highest for a particular benchmark. We notice that the IPC decreases as the cache size increases due to the fact that search time in each cache line is increasing.

Second, we vary the size of the L1d cache (data cache) while keeping the size of the L1i cache(instruction cache) constant at 1KB. The second graph displays how the IPC varies as we implement the above. We notice that as the L1d cache size increases, the latency and hit rate increase too. We infer that the L1d cache size is optimal when the IPC is at its highest. We can see that after this point, the IPC is not changing very significantly.

* For fixed L1i and varying L1D we see that for evenodd , prime and palindrome benchmarks throughput remains the same because there are very few memory read/write involved.
* However in descending and palindrome , extensively memories are accessed so there is varied throughput.
* In even odd program , since there is only 1 memory read instruction so irrespective of cache size we can fit the instructions and the data so increasing cache size simply decreases the throughput.
* In descending we see the advantage of temporal locality and since the numbers are very less all these numbers can fit in the cache , when the cache size is increased , latency is increased so IPC decreases.
* We also observe that except for even odd all benchmarks have higher throughput when L1i cache size is 128B.

# Benchmark program for L1i Cache

This assembly code shows the performance improvement when L1i cache size is increased from 16B to 128B keeping L1D as 1 kilobytes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | . data | | | | |
| 2 a: |  | | | | |
| 3 | 10 | | | | |
| 4 | . text | | | | |
| 5 **ma** | | in: | | | |
| 6 | sub % x19 , % x19 , % x19 | | | | |
| 7 | addi % x19 , 10 , % x19 | | | | |
| 8 lo | op: | |  |  |  |
| 9 | addi | | % x23 , | 1 , | % x23 |
| 10 | addi | | % x23 , | 2 , | % x23 |
| 11 | addi | | % x23 , | 3 , | % x23 |
| 12 | addi | | % x23 , | 4 , | % x23 |
| 13 | addi | | % x23 , | 3 , | % x23 |
| 14 | addi | | % x23 , | 4 , | % x23 |
| 15 | addi | | % x23 , | 5 , | % x23 |
| 16 | addi | | % x23 , | 6 , | % x23 |
| 17 | subi | | % x19 , | 1 , | % x19 |

18 bgt % x19 , % x0 , loop

19 end

### Comparison

|  |  |  |
| --- | --- | --- |
| Parameters | L1i Cache Size | |
| 16B | 128B |
| No. of Instructions | 103 | 103 |
| No. of Cycles | 4678 | 911 |
| IPC | 0.022 | 0.113 |

Table 1: Performance Metrics

# Benchmark program for L1d Cache

This assembly code shows the performance improvement when the L1d cache size is increased from 16B to 128B keeping L1i as 1 kilobyte.

1 . data

2 a:

3 1

4 2

5 3

6 4

7 5

8 6

9 7

10 . text

11 main :

12 sub % x12 , % x12 , % x12

13 sub % x3 , % x3 , % x3

14 addi % x3 , 10 , % x3

15 loop:

16 load % x12 , $a , % x12

17 load % x12 , $a , % x12

18 load % x12 , $a , % x12

19 load % x12 , $a , % x12

20 load % x12 , $a , % x12

21 load % x12 , $a , % x12

22 load % x12 , $a , % x12

23 sub % x12 , % x12 , % x12

24 subi % x3 , 1 , % x3

25 bgt % x3 , % x0 , loop

26 end

### Comparison

|  |  |  |
| --- | --- | --- |
| Parameters | L1D Cache Size | |
| 16B | 128B |
| No. of Instructions | 104 | 104 |
| No. of Cycles | 3645 | 1341 |
| IPC | 0.028 | 0.07 |

Table 2: Performance Metrics