2019 Spring NCTU Computer Organization

Lab5 Report

0616015 劉姿利、0616092 粘捷

1 Memory Stall Cycles Calculations

Notions	Operations	Delay (cycle)
t_{sa}	Send the address	1
t_{al}	Access single cache content	2
t_{al1}	Access L1 cache content	1
t_{al2}	Access L2 cache content	10
t_{am}	Access memory content	100
t_{sw}	Send a word of data	1

1.1 One-word-wide memory organization

$$\begin{aligned} hit\_cycles &= t_{sa} + t_{al} + t_{sw} \\ &= 1 + 2 + 1 = 4 \\ miss\_cycles &= t_{sa} + 8 \times (t_{sa} + t_{am} + t_{sw} + t_{al}) + t_{al} + t_{sw} \\ &= 1 + 8 \times (1 + 100 + 1 + 2) + 2 + 1 = 836 \end{aligned}$$

1.2 Wider memory organization

$$\begin{aligned} hit\_cycles &= t_{sa} + t_{al} + t_{sw} \\ &= 1 + 2 + 1 = 4 \\ miss\_cycles &= t_{sa} + (t_{sa} + t_{am} + t_{sw} + t_{al}) + t_{al} + t_{sw} \\ &= 1 + (1 + 100 + 1 + 2) + 2 + 1 = 108 \end{aligned}$$

1.3 Two-level memory organization

$$\begin{aligned} hit\_cycles &= t_{sa} + t_{al1} + t_{sw} \\ &= 1 + 1 + 1 = 3 \\ L1\_miss\_cycles &= t_{sa} + (t_{sa} + t_{al2} + t_{sw} + t_{al1}) + t_{al1} + t_{sw} \\ &= 1 + 4 \times (1 + 10 + 1 + 1) + 1 + 1 = 55 \\ global\_miss\_cycles &= t_{sa} + 32 \times (t_{sa} + t_{am} + t_{sw} + t_{al2}) + 4 \times (t_{sa} + t_{al2} + t_{sw} + t_{al1}) + t_{al1} + t_{sw} \\ &= 1 + 32 \times (1 + 100 + 1 + 10) + 4 \times (1 + 10 + 1 + 1) + 1 + 1 = 3639 \end{aligned}$$

2 Differences among Memory Organizations

We can see that increasing bandwidth indeed reduce the miss penalty (from 836 to 108). While with using L-2 cache, the L-2 miss penalty is much higher, so we need to focus on reducing miss rate to avoid total memory stall cycle higher then original memory organizations (Fig1.(a)).

3 Bonus

3.1 matmul.txt C++ style code

```
for (i = 0; i < m; i ++) {
    for (j = 0; j < p; j ++) {
        for (k = 0; k < n; k ++) {
            load C[i][j];
            load A[i][k];
            load B[k][j];
            do C[i][j] += A[i][k] * B[k][j];
            store C[i][j];
        }
    }
}
```

3.2 bonus_matmul.txt C++ style code

```
for (i = 0; i < m; i ++) {
    for (j = 0; j < n; j ++) {
        load A[i][j];
        for (k = 0; k < p; k ++) {
            load C[i][k];
            load B[j][k];
            do C[i][k] += A[i][j] * B[j][k];
            store C[i][k];
        }
    }
}
```

3.3 Explanation

我們將load A[i][j] 提到第二層迴圈，這樣就可以將原來要load A[i][j] $m \times n \times p$ 次減少到$m \times n$ 次，減少require data 的次數。

3.4 Test and Delay Cycles Calculations for Bonus

Compiling: make bonus
Execution: ./simulate_caches [input_filename] [output_filename]
the output format is exactly the same as the original one.