電子情報学専攻　専門
平成 22 年　解答･解説

diohabara

2021 年 8 月 16 日

第 1 問 電気･電子回路

第 2 問 計算機アーキテクチャ

(1)

初期参照ミス

キャッシュラインを最初にアクセスしたときに起こるミス

競合性ミス

同じインデックスを持つ異なるキャッシュラインにアクセスすることで起こるミス

容量性ミス

キャッシュしたいライン数がキャッシュの容量を上回ることで起こるミス

(2)

競合性ミス (フルアソシアティブキャッシュではインデックスを用いず、参照されるごとにすべてのラインのタグとメモリアドレス内のタグを比較するため。)

(3)

1 つのインデックスに対応するキャッシュラインの集合、セットの大きさを大きくする。つまり、連想度を大きくする。

(4)

- キャッシュは新しく領域を確保する
- メモリからデータを取ってきて領域に格納する

(5)

TLB、Translation Lookaside Buffer とは仮想アドレスと物理アドレスの変換表であるページテーブル専用のフルアソシアティブ方式のキャッシュである。仮想ページアドレスをタグとし、対応する物理ページアドレスを格納している。

(6)

ページサイズが 4096 バイトなので、$\log 4096 = 12$ ビットのページ内オフセットが必要である。
仮想アドレスが 32 ビットなので仮想ページアドレスは $32 - 12 = 20$ ビットとなる。これが TLB のタグとなる。
同様に物理アドレスが 31 なので物理ページアドレスは $31 - 12 = 19$ ビットとなる。となる。これが TLB のキャッシュラインの大きさである。
更に有効ビット 1 ビットを考慮して、TLB は 64 エントリなので、求める TLB の大きさは

$$(1 + 20 + 19) \times 64 = 2560[\text{bit}] = 320[\text{byte}]$$

(7)

1. ページテーブルを参照する
2. TLB に空いているエントリがある場合、現在参照している仮想ページアドレスに対応する物理ページアドレスが入れられる
3. TLB が空いていない場合、LRU ラインなエントリを使って入れる

(8)

ページフォールト

(9)

- CPU の処理を一時中断する
- 例外を起こして処理を OS に移し、OS の特権的命令を用いる
- テーブルのエントリに入っている二次記憶上のアドレスから主記憶の空いている場所にページをコピーする
- ページテーブルのエントリに物理ページアドレスを書き込み、有効ビットを 1 にする
- 主記憶に空いた場所がなければどれか 1 つのページを二次記憶上に追い出して、空いた場所に必要とされるページを読み込む

(10)

$\log 4096 = 12$ より、ページ内オフセットは 12 ビットである。これだけがキャッシュアクセスに使える。
キャッシュラインが 64 バイトで 1 ワードが 32 ビットなので、$(64 \times 8)/32 = 16$ より 1 ラインに 16 ワード入るから、ワードを特定するために $\log 16 = 4$ ビットが必要。
更に 1 ワードの中に 4 バイトが入るから、バイトを特定するために $\log 4 = 2$ ビットが必要。
更にキャッシュアクセスのインデックスとして使われるのは $12 - 4 - 2 = 6$ ビットとなる。
2 ウェイセットアソシアティブであり、1 ラインのデータ語が 64 バイトなので、求めるキャッシュの最大容量 (データ語) は

$$2^6 \times 2 \times 64 = 8192[\text{byte}]$$

(11)

TLB を参照する時間を待ってからメモリ語にアクセスするので、メモリ語にアクセスされるまでの時間が多くかかる。

(12)

名称

エイリアス

どのようなものか

別の 2 つの仮想アドレスが同じ物理アドレスを指している場合、片方の仮想アドレスによるキャッシュの更新がもう片方の仮想アドレスを利用する側に伝わらないという問題が発生する。

第 3 問 アルゴリズムとデータ構造

(1)

- if 文の中で (単語、出現頻度) を出力したあとに変数 num を 0 で初期化するように修正する。修正以前は単語ごとの出現頻度でなく、今まで登場したすべての単語への出現頻度が出力されていた。
- 処理の最後に output_pair(word, num) を足すように修正する。修正以前は単語リストの最後にある単語の (単語、出現頻度) ペアが出力されない。

(2)

```python
def merge_two_lists(
    pair1: list[tuple[str, int]], pair2: list[tuple[str, int]]
) -> list[tuple[str, int]]:
    ptr1, ptr2 = 0, 0
    result = []
    while ptr1 < len(pair1) and ptr2 < len(pair2):
        if pair1[ptr1][0] < pair2[ptr2][0]:
            result.append(pair1[ptr1])
            ptr1 += 1
        elif pair1[ptr1][0] == pair2[ptr2][0]:
            result.append(pair1[ptr1][0], pair1[ptr1][1] + pair2[ptr2][1])
            ptr1 += 1
            ptr2 += 1
        else:
            result.append(pair2[ptr2])
            ptr2 += 1
    if ptr1 < len(pair1):
        result += pair1[ptr1:]
    else:
        result += pair2[ptr2:]
    return result
```

(3)

N 個から 2 つのリストを選んで、(2) で定義した関数を適用して 1 つのリストにマージする。そして、出来上がったリストと残りの N-2 個のリストを順番に (2) の関数を使ってマージさせる。

(4)

文字を ascii コードなどに変換してそれらの数字を足し合わせたものを N で割ってマッピングする。(ただし、マッピング関数は分配が一様になる方が N 台のマシンに処理が均等に分配でき処理効率がよい。)

(5)

マシンが N 台あるとする。
それぞれのマシンで持てる単語の数に上限を定めて、それを超えた場合は別のマシンに単語を分配する。用いるマッピング関数は (4) と同じ。

第 4 問 情報通信

第 5 問 信号処理

(1)

片側 z 変換の定義は $X(z) = \sum_{n=0}^{\infty} x(n)z^{-n}$ なのでこれを適用する。また、$n - m \to l$ の置き換えと、$x(n) = 0(n < 0)$ を用いる。
$x(n-m)$ を z 変換したものを $X'(z)$ として

$$\begin{aligned} X'(z) &= \sum_{n=0}^{\infty} x(n-m)z^{-n} \\ &= \sum_{l=-m}^{\infty} x(l)z^{-l-m} \\ &= z^{-m} \sum_{l=-m}^{\infty} x(l)z^{-l} \\ &= z^{-m}X(z) \end{aligned}$$

(2)

畳み込みの定義は以下の通り。

$$x_1(n) * x_2(n) = \sum_{k=0}^{\infty} x_1(n-k)x_2(k)$$

(3)

$X_1(z)X_2(z)$ の z^{-n} の係数に注目すると、この部分は $\sum_{k=0}^{n} x_1(n-k)x_2(k)$ になっているので、(2) で答えた離散信号の Z 変換になっていることがわかる。

(4)

$\oplus$ に入力される信号は $x(n) + ay(n-1)$ であり、この b 倍が $y(n)$ となるので

$$y(n) = b\{x(n) + ay(n-1)\}$$

また、Z 変換を施すと $Y(z) = b\left\{X(z) + a^{-1}Y(z)\right\} \to (1 - abz^{-1})Y(z) = bX(z)$ となるので求める $H(z)$ は

$$H(z) = \frac{b}{1 - abz^{-1}}$$

となる。

(5)

Z 変換において

$$x(n) = a^n \cdot u(n) \to X(z) = \frac{1}{1 - az^{-1}}$$

という式が成り立つ。ただし、$u(n)$ はステップ関数。
この公式で $a = ab$ としたものに b をかけると (4) の結果となる。よって、$h(n) = a^n b^{n+1}$。

よって、$y(n) = (x * h)(n)$ の関係より

$$\begin{aligned} y(n) &= \sum_{k=0}^{n} x(n-k)h(k) \\ &= \sum_{k=0}^{n} x(n-k)a^k b^{k+1} \end{aligned}$$