Лабораторная работа №3	22Б05	2024
Кэш	Алькаев Равиль Юрисович	

Инструментарий и требования к работе: VS Code, C++

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-cache-alkaev/

Результат работы на тестовых данных:

Реализованы три политики вытеснения: LRU, bit-pLRU и Round-robin. Ниже предоставлены ответы, которые я получил

LRU: hit perc. 99.125% time: 3157550

pLRU: hit perc. 99.4046% time: 3083066

RR: hit perc. 98.9627% time: 3200582

MEM_SIZE	2^(ADDR_LEN) = 64Кбайт
ADDR_LEN	16 бит
Конфигурация кэша	look-through write-back
Политики вытеснения кэша	LRU, bit-pLRU, Round-robin
CACHE_WAY	4
CACHE_TAG_LEN	ADDR_LEN - CACHE_IDX_LEN – CACHE_OFFSET_LEN= = 16 – 5 – 5 = 6 бит
CACHE_IDX_LEN	LOG_2(CACHE_SETS_COUNT) = LOG_2(32) = 5 бит
CACHE_OFFSET_LEN	LOG_2(CACHE_LINE_SIZE) = LOG_2(32) = 5 бит
CACHE_SIZE	CACHE_LINE_COUNT * CACHE_LINE_SIZE = 128 * 32 = 4096 байт
CACHE_LINE_SIZE	32 байт
CACHE_LINE_COUNT	CACHE_SETS_COUNT * CACHE_WAY = 32 * 4 = 128
CACHE_SETS_COUNT	32

Размерность шин (ППА)

Шина	Обозначение	Размерность
A1	ADDR1_BUS_LEN	ADDR_LEN = 16
A2	ADDR2_BUS_LEN	CACHE_TAG_LEN + CACHE_IDX_LEN = 5 + 6 = = 11
D1	DATA1_BUS_LEN	16 бит

D2	DATA2_BUS_LEN	32 бит
C1	CTR1_BUS_LEN	LOG_2(кол-во команд между процессором и кэшем) = LOG_2(7) = 3
C2	CTR2_BUS_LEN	LOG_2(кол-во команда между памятью и кэшем) = = LOG_2(3) = 2

Разберемся с тиками

Переписывание кэш линий 16 тиков из кэша в MEM и обратно, MEM отвечает за 100 тиков, еще 4 тика для запроса кэша при промахе и для ответа кэшу нужно еще 6. Тогда получим такие значения.

int Cache_to_Mem_Ticks = 101;(MEM отвечает за 100 тиков + 1 тик для передачи по C2_RESPONSE)

int Cache_Find_Ticks = 6; (кэш попадание 6 тиков)

int Cache_Eror_Ticks = 113; (4 тика кэш мисс, 100 тиков тратит MEM, тик на передачу по C2_RESPONSE размер кэш линии 32 байта, а по D2 передается 4 байта, значит это еще 8 тиков (4+100+1 +8 = 113))

Разберемся с программой.

int Ask = 0; кол-во запросов

int Miss = 0; кол-во миссов

int Ticks = 0; кол-во тиков

Tags массивой наших тегов, Times здесь храним время, Modified – надобность записи в память.

```
Cache_Settings(std::string Politics) : Politics(Politics) {
        Tags = std::vector<std::vector<int>>(CACHE_SETS_COUNT, std::vector<int>(CACHE_WAY, -1));
```

```
            Times = std::vector<std::vector<int>>(CACHE_SETS_COUNT, std::vector<int>(CACHE_WAY, 0));
            Modified = std::vector<std::vector<int>>(CACHE_SETS_COUNT, std::vector<int>(CACHE_WAY, 0));

        if (Politics == "RR") {
            for (int i = 0; i < CACHE_SETS_COUNT; i++) {
                Times[i][0] = 1;
            }
        }
    }
```

Для начала разберем функцию cache_request она симулировала работу кэша.

```
void cache_request(int Address, std::string Command) {
```

Сейчас мы вычислим тег и индекс по адресу.

```
        int tag = Address >> (CACHE_OFFSET_LEN + CACHE_IDX_LEN);
            int Index = (Address >> CACHE_OFFSET_LEN) % CACHE_SETS_COUNT;
        Ask += 1; // увеличиваем кол-во запросов на 1
```

Далее разбираем случай, когда наш тег уже лежит в массиве, то есть случилось кэш попадание.

```
        auto it = std::find(Tags[Index].begin(), Tags[Index].end(), tag);
        if (it != Tags[Index].end()) {
            Ticks += Cache_Find_Ticks; // кэш попадание
            int Second_index = std::distance(Tags[Index].begin(), it);
```

Случай когда политика вытеснения LRU, тогда нужно у всех ячеек увеличить время на 1, кроме нашего кэш попадания, его мы зануляем.

```
if (Politics == "LRU") {
for (int i = 0; i < CACHE_WAY; i++) {
    Times[Index][i] += 1;
}
Times[Index][Second_index] = 0;
```

Теперь политика bit-pLRU, проверяем есть ли свободное место, если есть то записываем и меняем это значение на 1, иначе обнуляем все и записываем на это значение 1. (В этой политике на нужно любое свободное место)

```
} else if (Politics == "pLRU") {
    Times[Index][Second_index] = 1;
    int Full = 1;
    for (int i = 0; i < CACHE_WAY; i++)
        if (Times[Index][i] == 0)
            Full = 0;
    if (Full == 1) {
        Times[Index] = std::vector<int>(CACHE_WAY, 0);
        Times[Index][Second_index] = 1;
    }
}
```

Если write то отмечаем, что сохранили тег, чтобы при вытеснение мы его записали в память

```
if (Command == "Write") {
    Modified[Index][Second_index] = 1;
}
```

Мы не разбирали политику RR так, как у наш кэш попадание и мы не должны двигать указатель.

Теперь разбираем кэш мисс

```
} else {
    Miss += 1;
    Ticks += Cache_Eror_Ticks; // добавление тиков при миссе
    int Second_index = 0;
```

При политике LRU находим индекс элемента с максимальным временем

```
    if (Politics == "LRU") {
                int max_time = *std::max_element(Times[Index].begin(), Times[Index].end());
                    Second_index = std::distance(Times[Index].begin(), std::find(Times[Index].begin(), Times[Index].end(), max_time));
```

При bit-pLRU нужно просто найти нулевой элемент.

```
    } else if (Politics == "pLRU") {
                    Second_index = std::distance(Times[Index].begin(), std::find(Times[Index].begin(), Times[Index].end(), 0));
```

При RR нам нужно найти указатель (три элемента массива нули, оставшийся один это указатель, равный единице)

```
    } else if (Politics == "RR") {
                    Second_index = std::distance(Times[Index].begin(), std::find(Times[Index].begin(), Times[Index].end(), 1));
    }
```

Если нужно сохранить в память, сохраняем и добавляем тики.

```
    if (Modified[Index][Second_index] == 1) {
        Ticks += Cache_to_Mem_Ticks;
    }
```

Сохраняем тег и если запись, то еще не забываем про то, что нужно будет еще сохранить в память.

```
    Tags[Index][Second_index] = tag;
```

```
        if (Command == "Read"){
            Modified[Index][Second_index] = 0;
        }
        else {
            Modified[Index][Second_index] = 1;
        }
        if (Politics == "LRU") {
        for (int i = 0; i < CACHE_WAY; i++) {
            Times[Index][i] += 1;
        }
        Times[Index][Second_index] = 0;
    } else if (Politics == "pLRU") {
        Times[Index][Second_index] = 1;
        int Full = 1;
        for (int i = 0; i < CACHE_WAY; i++)
            if (Times[Index][i] == 0)
                Full = 0;
        if (Full == 1) {
            Times[Index] = std::vector<int>(CACHE_WAY, 0);
            Times[Index][Second_index] = 1;
        }
```

Разберем только, случай RR так, как остальные были выше. Нужно просто найти указатель и увеличить его на 1.

```
    } else if (Politics == "RR" ) {
        for (int i = 0; i < CACHE_WAY; i++) {
            if (Times[Index][i] == 1) {
                Times[Index][i] = 0;
                Times[Index][(i + 1) % CACHE_WAY] = 1;
                break;
```

```
            }
          }
        }
        }
    }};
```

Функция cache_implentation реализация перемножения матриц, на ней мы и запустим нашу функцию cache_request.