Лабораторная работа №3	Б06	2023
Кэш	Прозорова Варвара Владиславовна	

Инструментарий и требования к работе: Python (3.11.5)

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-cache-stepashka-21

Результат работы написанной программы:

LRU: hit perc. 96.6571% time: 3894479

pLRU: hit perc. 96.6406% time: 3899428

Результат расчёта параметров системы:

MEM_SIZE	512 Кбайт = 2 ** 19 байт
ADDR_LEN	$\log_2$ (MEM_SIZE) = 19
Конфигурация кэша	look-through write-back
Политика вытеснения кэша	LRU и bit-pLRU
CACHE_WAY	CACHE_LINE_COUNT / CACHE_SETS_COUNT = 4
CACHE_TAG_LEN	10
CACHE_IDX_LEN	ADDR_LEN - CACHE_TAG_LEN - CACHE_OFFSET_LEN = 4
CACHE_OFFSET_LEN	$\log_2$ (CACHE_LINE_SIZE)= 5
CACHE_SIZE	CACHE_LINE_COUNT * CACHE_LINE_SIZE = 2048 байт
CACHE_LINE_SIZE	32 байт
CACHE_LINE_COUNT	64
CACHE_SETS_COUNT	2 ** CACHE_IDX_LEN = 16

Шина	Обозначение	Размерность
A1, A2	ADDR1_BUS_LEN, ADDR2_BUS_LEN	A1: 19 бит A2 — без offset: 14 бит
D1, D2	DATA1_BUS_LEN, DATA2_BUS_LEN	16 бит

C1, C2	`CTR1_BUS_LEN, CTR2_BUS_LEN`	`C1 – 7 команд: 3 бита` `C2 – 3 команды: 2 бита`

Описание работы написанного кода:

Я создала отдельный класс Cache, где реализуются все функции кэша. Сам кэш – список `cache` из `CACHE_SETS_COUNT` подсписков размера `CACHE_WAY`, в котором хранятся int'ы – тэги (по идее кэш-линии). Изначально в нем хранятся -1 (или ничего). Также параллельно с этим списком хранится список `time_for_all`, он такой же, как и `cache`. Он нужен для подсчета времени/числа раз использования конкретного элемента/тэга. Изначально в нем 0. Описания других констант, счетчиков есть в коде.

Описание функций:

read – на вход: индекс, тэг и политика –

LRU: циклом проходит по всем элементам `cache[ind_i]`, проверяет на равенство конкретному тэгу (то есть находится ли данный элемент в кэше). Если находится, то `ind_j` становится равным номеру найденного элемента и ко всем элеменетам в списке `time_for_all[ind_i]` прибавляется 1, иначе -1.

Иначе, *pLRU:* циклом проходит по всем элементам `cache[ind_i]`, проверяет на равенство конкретному тэгу (то есть находится ли данный элемент в кэше). Если находится, то `ind_j` становится равным номеру найденного элемента, иначе -1. Дальше: если все-таки элемент найден, в списке `time_for_all[ind_i]` сумма всех элементов равна 3, то есть если уже есть три единицы, и на месте найденного элемента 0, то нужно заменить все единицы на нули и нуль на единицу.

old_lru – на вход: индекс, для LRU – снова циклом проходит по всем элементам `time_for_all[ind_i]`, находит 0 или максимальное значение, то есть если есть пустое место, то можно просто записать данный тэг, или самый давний по использованию элемент, который нужно перезаписать на новый. Далее снова циклом проходит по всем элементам и прибавляет к каждому значению, если оно не 0, единицу (позднее на месте замененного элемента в `time_for_all[ind_i][ind_j]` будет 1). После того, как место в кэше заполняется в списке со временем на этом месте никогда не будет числа, меньшего 1.

old_plru – на вход: индекс, для pLRU – циклом проходит по всем элементам `time_for_all[ind_i]`, находит первый 0, если сумма элементов не 3 (а точнее меньше 3, так как 4 быть не может, это везде пресекается), то есть в этом списке не более двух единиц, тогда вот сюда будет записываться новый тэг, а иначе на месте нуля в `cache[ind_i]` будет записан данный тэг, а в списке времени н сам заменится на единицу, а все единицы станут нулями.

write – на вход: индекс сета, индекс конкретной линии и тэг, который нужно записать – просто записывает на данное место в `cache[ind_i][ind_j]` данный тэг.

mmul – на вход: политика вытеснения кэша – сначала присвоим каждой клетке матрицы свое значение, в `a` каждая клетка 1 байт, в `b` – 2 байта, в `c` – 4 байта. Создаем кэш с заданными в задаче значениями.

get_addr – функция для получения адреса (offset-ind-tag): индекс – 4 бита после 5, что идут на offset, тэг – последние 10 битов.

total_time – такты, **miss** – кол-во промахов, **query** – кол-во запросов

Цикл (y) по M, цикл (x) по N, цикл (k) по K, все как в задаче. Внутри:

для каждого элемента: `a[y][k]`, `b[k][x]`, `c[y][x]` все будет одинаково (кроме количества тактов), поэтому разберу только для `a[y][k]`. С помощью get_addr узнаем индекс и тэг элемента. Далее обращаемся к кэшу. Читаем в нужном сэте элементы (сделали запрос, query += 1).

Если политика LRU: если при чтении элемент не нашелся (случился промах, miss += 1), то есть read вернул индекс -1, находим давнее по использованию или пустое место, записываем новый тэг. Если нашелся, то ничего не делаем. В списке `time_for_all[ind_i]` на нужном месте записываем 1, так как элемент только что был использован, и неважно, перезапись это или нет.

Иначе, если политика pLRU: если при чтении элемент не нашелся (случился промах, miss += 1), находим первый 0 в списке времени, записываем туда новый тэг. Если нашелся, то ничего не делаем. В списке `time_for_all[ind_i]` на нужном месте записываем 1, так как элемент только что был использован, и неважно, перезапись это или нет.

hit perc.:

чтобы посчитать процент используется формула:

```
100 * (1 - miss / query)
```

Считаем такты:

total_time – подсчитывается общее количество тактов

Инициализация: изначально 2, в среднем цикле 3 (или одно присвоение).

Переход на новую итерацию цикла 1 такт + 1 такт x++ – внутри каждого цикла `total_time += cache.loop + cache.add`

Во внешнем цикле: два сложения (`pa += K; pc += N; == total_time += cache.add * 2`).

Во внутреннем цикле: умножение + сложение `(s += pa[k] * pb[x] == cache.mul + cache.add)` + сложение `(pb += N)`

Разберемся с функциями кэша. Реализация read занимает: 1 такт – запрос к кэшу в случае матриц a и b (`total_time += cache.A`), 2 такта – запрос к кэшу в случае матрицы c. Далее:

Если нужно записать в кэш: 1 такт на запрос к памяти (`cache.mem`), 100 тактов на ответ памяти (`cache.mem_answer`), (256/16) 16 тактов на отправку данных в кэш (`cache.D`), 4 такта на промах в кэше (`cache.cache_miss`).

Если прочитать: 6 тактов на попадание в кэше (`total_time += cache.cache_hit`).

Если запись в память: это есть в случае `(pc[x] = s)`. С помощью константы k из класса Cache будем фиксировать те случаи, когда есть перезапись: 16 тактов на отправку данных в память, 1 такт на запрос к памяти, 100 тактов на ответ памяти. Это все плюс еще к тому времени записи в кэш.