EPFL

CS-473: System programming for Systems on Chip

Practical work 4

Caches

Version:
1.0

Contents

1 Introduction **1**
1.1 Prerequisites 1
1.2 Objectives 1

2 Exercises **2**
2.1 Structure Memory Layout and Caches 2
2.2 Programming Tasks 4

1 Introduction

1.1 Prerequisites

On moodle you find the file `pw4.zip`; it contains the complete source code template for this practical work. Download this file, and unzip it in the directory `programs`.

1.2 Objectives

In the scope of this practical work, you will learn about:

1. memory layouts of structures in C,
2. optimizing the memory layout for caches,
3. designing cache-friendly algorithms,
4. the interaction between I/O devices and caches.

2 Exercises

2.1 Structure Memory Layout and Caches

Please read "The Lost Art of Structure Packing" by Eric S. Raymond to answer the questions in this section. It is available at: `http://www.catb.org/esr/structure-packing/`.

Project `cache_sweep` under `cache/sweep/` executes a C program summarized in Figure 2.1. The program contains a struct item_t consisting of an ID and data, an array of item_ts, and function items_find that finds an item_t with a given ID. The full source code of the program is available under `cache/sweep/src/datapoint/datapoint.c`. The executable contains multiple versions of this program for different configurations described by the following parameters:

1. PARAM_PACKED: item_t has the __packed modifier or not.
2. PARAM_DATALEN: length of field data of struct item_t.
3. PARAM_COUNT: length of the array.

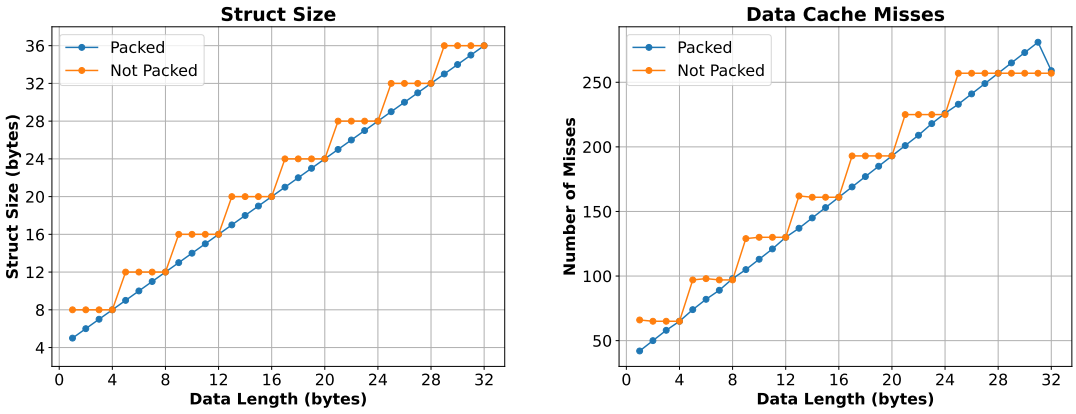
We aim to analyze how these parameters affect (1) the code generated by the compiler and (2) the number of data cache misses. Both plots in Figure 2.2 sweep PARAM_DATALEN. Figure 2.2a plots sizeof(item_t). Figure 2.2b plots the number of data cache misses for function items_find. For all the data points in Figure 2.2b, function items_find accesses all the array elements.

Questions:

1. What does the __packed modifier do? Note that __packed is defined in Figure 2.1. Online resources and the compiler manual might help you with this part of the practical work.
2. Explain the memory layout of item_t with and without __packed and for PARAM_DATALEN $\in \{1, 2, 3, 4\}$. Specify (1) the ordering of the fields, (2) the size of each struct field, and (3) padding bytes (the processor has a 32-bit alignment). You do not need to make a table; however, be clear and concise!
3. Repeat the previous question for the memory layout of the array. Consider the first two array elements.
4. How does __packed affect the number of cache misses? Why?
5. Consider the data series in Figure 2.2 that do not use __packed (i.e., the orange lines). What is the cache line size for the data cache? Explain your approach.
6. How does __packed affect the generated assembly code size? Why? What does the compiler do differently? Generated assembly file for each data point is available under `cache/sweep/build-release-or1300/src/datapoint/datapoint_<COUNT>_<LENGTH>_<PACKED>.s`. **Important:** to have these files available you have to add `-save-temps` to line 7 in your makefile, it should like: `CFLAGS ?= -save-temps`

```c
/** @note defined in `defs.h`. */
#define __packed __attribute__((packed))

/** @brief Item struct. Relates an `id` to a piece of `data`. */
typedef struct __packed /* or nothing, depending on the configuration */ {
    uint32_t id;
    char data[PARAM_DATALEN];
} item_t;

/** @brief An array of items. */
static item_t items[PARAM_COUNT];

/**
 * @brief Searches for an item matching the `id`.
 *
 * @param id
 * @return item_t* Pointer to the found item.
 */
item_t* items_find(uint32_t id) {
    for (size_t i = 0; i < PARAM_COUNT; ++i) {
        if (items[i].id == id)
            return &items[i];
    }
    return NULL;
}

```

Figure 2.1: A simplified version of the measured program. Each data point consists of PARAM_DATALEN , PARAM_COUNT, and PARAM_PACKED. The program is compiled and executed for each data point. PARAM_PACKED determines if the struct has __packed modifier. For each data point, we record sizeof(item_t) and the number of data cache misses.



(a) Structure size for varying data length.

(b) Number of data cache misses for varying data length.

Figure 2.2: Both figures vary the data length (PARAM_DATALEN). Figure (b) is drawn for PARAM_COUNT=256. The number of memory accesses is same as PARAM_COUNT.

2.2 Programming Tasks

Please read Section 4 of "Cache-Conscious Data Structures" from Microsoft Research to answer the questions in this section. It is available at: `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/ccds.pdf`.

In this part of the assignment, you will modify the source of project `cache_tasks` under `cache/tasks/`. Take a look at the directory structure and understand it. As a first step, please check your setup:

1. Compile the project as usual.
2. Upload the `.cmem` file and execute the program. Note that uploading might take up to 10 seconds.

The expected program output is the following (cache miss numbers might be slightly different):

```
task1_main
sizeof(node_t) = 64:
struct layout for node_t:
member          -> offset
node.id         -> 0x000
node.data       -> 0x004
node.next       -> 0x038
node.prev       -> 0x03c
#0 -> #11 -> #6 -> #12 -> #9 -> #2 -> #4 -> #8 -> #1 -> #3 -> #7 -> #15 -> #14 ->
    #13 -> #10 -> #5 -> Done.
Task 1: dcache misses:          42
task2_main
Item ID = 15, data = random data: 11
Task 2: dcache misses:          16
task3_main
Task 3: dcache misses:       65893
out_vector verification successful!
task4_main
original address: 0x50000800
new address: 0x08000800
```

Task 1: Optimize `node_t`

Consider the following source files:

- `cache/tasks/src/task1.c`
- `cache/tasks/src/node.c`
- `cache/tasks/include/node.h`

struct node_t is defined in `node.h` and function node_count is defined in `node.c`:

```c
#define NODE_DATALEN 52

typedef struct node_t node_t;

/**
 * @brief Defines a node.
 * @note **Do not remove** any fields from this structure.
 */
struct node_t {
    /** @brief Node ID. */
    unsigned id;

    /** @brief Node data. */
    char data[NODE_DATALEN];

    /** @brief The next node. */
    node_t* next;

    /** @brief The previous node. */
    node_t* prev;
};

uint32_t node_count(node_t* node) {
    // YOU ARE NOT SUPPOSED TO MODIFY THIS.

    uint32_t result = 0;

    while (node) {
        printf("#%u -> ", node->id);
        result++;
        node = node->next;
    }

    printf("Done.\n");

    return result;
}
```

Questions:

7. What accesses in function node_count cause cache misses? Why?
8. Propose a strategy to decrease the number data cache misses caused by function node_count. Test your approach on the Gecko board. What is the new number of cache misses? Do not modify the functions or remove fields from struct node_t.

Note that node_count is unlikely to access node_ts that are adjacent in memory.

Task 2: Optimize `item_t`

Consider the following source files:

- `cache/tasks/src/task2.c`
- `cache/tasks/src/item.c`
- `cache/tasks/include/item.h`

struct item_t is defined in `item.h`; functions items_find and item_init are defined in `item.c`:

```c
#define ITEM_DATALEN 32

typedef struct item_t item_t;

/**
 * @brief An item connects an `id` to `data`.
 *
 */
struct item_t {
    /** @brief Item ID. */
    unsigned id;

    /** @brief Item data. */
    char data[ITEM_DATALEN];
};

item_t* items_find(item_t* items, size_t log2n, unsigned id) {
    // YOU ARE NOT SUPPOSED TO MODIFY THIS.
    for (size_t i = 0; i < (1 << log2n); ++i) {
        if (items[i].id == id)
            return &items[i];
    }
    return NULL;
}

void item_init(item_t* item, uint32_t id, const char* data) {
    // YOU CAN MODIFY THIS.
    item->id = id;

    if (data != NULL)
        memcpy(item->data, data, ITEM_DATALEN);
    else
        memset(item->data, 0, ITEM_DATALEN);
}
```

Questions:

9. What is the source of data cache misses in function items_find? Explain.
10. Propose a strategy to decrease the number data cache misses caused by function items_find. Test your approach on the Gecko board. What is the new number of cache misses? You can modify only function items_init. You can also modify struct items_t as long as it holds the ID-data relationship.

Notice that function items_find accesses items_t objects that are adjacent in memory.

Task 3: Optimize Matrix-Vector Multiplication

Consider `cache/tasks/src/task3.c`. In this file, function multiply is defined as:

```
/**
 * @brief Multiplies the matrix with the input vector.
 * Writes to the output vector.
 */
static void multiply() {
    // YOU CAN MODIFY THIS.

    for (int j = 0; j < MATRIX_N; ++j) {
        for (int i = 0; i < MATRIX_N; ++i) {
            out_vector[i] += matrix[i][j] * in_vector[j];
        }
    }
}
```

Questions:

11. Explain the terms row-major and column-major order. Which approach is used in C?
12. What accesses cause cache misses in function multiply?
13. Propose a strategy to decrease the number data cache misses caused by function multiply.

Task 4: Bouncing Ball Example

Consider `cache/tasks/src/task4.c`. In this file, functions bouncing_ball and init_dcache are defined as:

```
// old base address
#define LEDS_OLD_BASE 0x50000800ull

// new base address
#define LEDS_NEW_BASE 0x08000800ull

// pixel-based control of LEDs
#define LEDS_LEDS_OFFSET 0x400ull

// the base address is configurable.
// defines the offset of the base address register.
#define LEDS_BASEADDR_OFFSET 0x7FCull

void init_dcache() {
    // YOU CAN MODIFY THIS.
    dcache_enable(0);
    dcache_write_cfg(CACHE_FOUR_WAY | CACHE_SIZE_4K | CACHE_REPLACE_LRU |
    CACHE_WRITE_BACK);
    dcache_enable(1);
}

void bouncing_ball() {
    // YOU CAN MODIFY THIS.
    int xdir, ydir, xpos, ypos, index;
    volatile unsigned int* leds = (unsigned int*)(LEDS_NEW_BASE + LEDS_LEDS_OFFSET)
    ;

    xdir = ydir = 1;
    xpos = ypos = 5;
    while (1) {
        index = ypos * 12 + xpos;
        leds[index] = 0;
        if (ypos == 8)
            ydir = -1;
        if (ypos == 0)
            ydir = 1;
        if (xpos == 11)
            xdir = -1;
        if (xpos == 0)
            xdir = 1;
        ypos += ydir;
        xpos += xdir;
        index = ypos * 12 + xpos;
        leds[index] = swap_u32(2);
        for (volatile long i = 0; i < 100000; i++)
            ;
    }
}
```

Questions:

14. Why does not the bouncing ball work as expected? You should be observing no change in the LEDs. The expected behavior is a single pixel moving and reflects as it bounces at the edges.
15. Propose two approaches to fix the behavior.