#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <immintrin.h> // 包含 SIMD 指令集（如 AVX）

#define BUFFER\_SIZE 8192

#define NUM\_THREADS 4

#define USE\_MULTITHREADING 1

typedef struct {

FILE \*input;

FILE \*output1;

FILE \*output2;

size\_t start;

size\_t end;

} split\_task;

void process\_with\_simd(unsigned char \*buffer, size\_t size, FILE \*output1, FILE \*output2) {

for (size\_t i = 0; i < size; i += 32) { // 每次处理 32 字节 (256 位)

\_\_m256i data = \_mm256\_loadu\_si256((\_\_m256i\*)&buffer[i]);

// 使用掩码分别提取偶数和奇数位置的字节

\_\_m256i even\_mask = \_mm256\_set1\_epi8(0xAA); // 10101010b

\_\_m256i odd\_mask = \_mm256\_set1\_epi8(0x55); // 01010101b

\_\_m256i even\_data = \_mm256\_and\_si256(data, even\_mask);

\_\_m256i odd\_data = \_mm256\_and\_si256(data, odd\_mask);

// 将结果写入到对应的输出文件中

\_mm256\_storeu\_si256((\_\_m256i\*)&buffer[i], even\_data);

fwrite(&buffer[i], 1, 32, output1);

\_mm256\_storeu\_si256((\_\_m256i\*)&buffer[i], odd\_data);

fwrite(&buffer[i], 1, 32, output2);

}

}

void \*split\_file\_task(void \*args) {

split\_task \*task = (split\_task \*)args;

fseek(task->input, task->start, SEEK\_SET);

unsigned char buffer[BUFFER\_SIZE];

size\_t bytes\_read;

while (ftell(task->input) < task->end && (bytes\_read = fread(buffer, 1, BUFFER\_SIZE, task->input)) > 0) {

process\_with\_simd(buffer, bytes\_read, task->output1, task->output2);

}

pthread\_exit(NULL);

}

void split\_file(const char \*input\_file, const char \*file1, const char \*file2) {

FILE \*input = fopen(input\_file, "rb");

FILE \*output1 = fopen(file1, "wb");

FILE \*output2 = fopen(file2, "wb");

if (!input || !output1 || !output2) {

perror("Error opening file");

exit(EXIT\_FAILURE);

}

fseek(input, 0, SEEK\_END);

size\_t file\_size = ftell(input);

size\_t chunk\_size = file\_size / NUM\_THREADS;

rewind(input);

if (USE\_MULTITHREADING) {

pthread\_t threads[NUM\_THREADS];

split\_task tasks[NUM\_THREADS];

for (int i = 0; i < NUM\_THREADS; i++) {

tasks[i].input = input;

tasks[i].output1 = output1;

tasks[i].output2 = output2;

tasks[i].start = i \* chunk\_size;

tasks[i].end = (i == NUM\_THREADS - 1) ? file\_size : (i + 1) \* chunk\_size;

if (pthread\_create(&threads[i], NULL, split\_file\_task, &tasks[i]) != 0) {

perror("Failed to create split thread");

exit(EXIT\_FAILURE);

}

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

} else {

unsigned char buffer[BUFFER\_SIZE];

size\_t bytes\_read;

while ((bytes\_read = fread(buffer, 1, BUFFER\_SIZE, input)) > 0) {

process\_with\_simd(buffer, bytes\_read, output1, output2);

}

}

fclose(input);

fclose(output1);

fclose(output2);

}

把这段代码手动向量化

我们可以通过在读取数据时手动执行字节的批量处理，使用位操作来实现分离偶数和奇数位置的字节。以下是改写后的代码：

实际的 SIMD 指令（如 AVX、SSE）可以同时在多个寄存器中并行处理数据，而手动向量化是通过软件层面的优化尽可能地并行处理数据，但它仍然是顺序执行的。这两者的性能差距较大，但在不支持 SIMD 的情况下，手动向量化是一个不错的替代方法。

#include "file\_split.h"

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define BUFFER\_SIZE 8192

void process\_buffer(unsigned char \*buffer, size\_t size, FILE \*output\_even, FILE \*output\_odd) {

for (size\_t i = 0; i < size; i++) {

if (i % 2 == 0) {

fputc(buffer[i], output\_even);

} else {

fputc(buffer[i], output\_odd);

}

}

}

void split\_file(const char \*input\_file, const char \*temp\_prefix) {

FILE \*input = fopen(input\_file, "rb");

if (!input) {

perror("Error opening input file");

exit(EXIT\_FAILURE);

}

char temp\_even\_file[FILENAME\_MAX];

char temp\_odd\_file[FILENAME\_MAX];

snprintf(temp\_even\_file, sizeof(temp\_even\_file), "%s\_even.tmp", temp\_prefix);

snprintf(temp\_odd\_file, sizeof(temp\_odd\_file), "%s\_odd.tmp", temp\_prefix);

FILE \*temp\_even = fopen(temp\_even\_file, "wb");

FILE \*temp\_odd = fopen(temp\_odd\_file, "wb");

if (!temp\_even || !temp\_odd) {

perror("Error opening temporary files");

exit(EXIT\_FAILURE);

}

unsigned char buffer[BUFFER\_SIZE];

size\_t bytes\_read;

size\_t position = 0;

while ((bytes\_read = fread(buffer, 1, BUFFER\_SIZE, input)) > 0) {

process\_buffer(buffer, bytes\_read, temp\_even, temp\_odd);

position += bytes\_read;

}

fclose(input);

fclose(temp\_even);

fclose(temp\_odd);

}

int main(int argc, char \*argv[]) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <input\_file> <temp\_prefix>\n", argv[0]);

return EXIT\_FAILURE;

}

split\_file(argv[1], argv[2]);

return EXIT\_SUCCESS;

}