به نام خدا

گزارش تمرین کامپیوتری

شماره چهار

پردازش تصویر بی درنگ با Pthread

سید علیرضا میرشفیعی – 810101532

محمد صدرا عباسی – 810101469

کلاس ThreadSafeQueue

هدف، پیاده سازی یک صف ایمن میان ترد هاست، چرا که قرار است بخش انتقال اطلاعات میان pipeline ها را برعهده بگیرد. در نتیجه باید از Race Condition و کرش کردن برنامه بدلیل دسترسی غیرمجاز جلوگیری کند.

این کلاس در اصل یک wrapper حول صف است، که به ما امکان مدیریت دسترسی همزمان ترد ها به حافظه را میدهد. در نتیجه نیاز است که از قفل برای لاک کردن دسترسی همزمان استفاده کرد، در اینجا از mutex استفاده شده.

حال به سراغ جزئیات پیاده سازی هر بخش میرویم.

Push

این متد با استفاده از lock_gaurd میوتکس را قفل میکند، و داده جدید را درون صف پوش میکند.
مزیت lock_gaurd این است که پس از خارج شدن از بلوک کد بصورت خودکار قفل را آزاد میکند.

```cpp
template <typename T>
void ThreadSafeQueue<T>::push(T value) {
    // lock_guard is used for simple critical sections (RAII)
    std::lock_guard<std::mutex> lock(mtx);

    // If shutdown has started, we generally shouldn't accept new items,
    // though this depends on specific requirements. Here we ignore push if shutdown.
    if (shutdown_flag) return;

    queue.push(std::move(value));

    // Notify one waiting thread that data is ready
    cv.notify_one();
}
```

جزئیات دیگر پیاده سازی این بخش در بخش های دیگر توضیح داده میشود..

Pop

این متد پس از صدا شدن اول قفل را برای خود لاک میکند، سپس با استفاده از condition_variable منتظر میماند تا زمانی که صف خالی شود، سپس یک مقدار از سر صف pop میکند.

cv.wait تا زمانی منتظر میماند صف حداقل یک مقدار بگیرد، و بجای هربار چک کردن صف منتظر یک سیگنال میماند .این سیگنال در زمان push اتفاق میفتد، چرا که تنها جاییس که یک مقدار جدید به صف اضافه میشود، درنتیجه در انتها متد push یکی از ترد ها که در cv.wait منتظر بوده بیدار میشود.

```cpp
template <typename T>
T ThreadSafeQueue<T>::pop() {
    // unique_lock is required for condition_variable::wait
    std::unique_lock<std::mutex> lock(mtx);

    // Wait until queue is not empty OR shutdown is triggered.
    // The lambda handles the spurious wakeups.
    cv.wait(lock, [this] {
        return !queue.empty() || shutdown_flag;
    });

    // If the queue is empty and shutdown is flagged, return nullopt (terminate signal)
    if (queue.empty() && shutdown_flag) {
        return T();
    }

    // Process the item
    T value = std::move(queue.front());
    queue.pop();

    return value;
}
```

Shutdown

در صورت اتمام برنامه، دیستراکتور کلاس متد shutdown صدا زده میشود و boolian اتومیک shutdown_flag را true میکند، این کار باعث میشود تا شرط ترد هایی که در cv.wait در انتظار بودند true شوند.

در نهایت با cv.notify_all این ترد ها را بیدار میکند.

```
template <typename T>
void ThreadSafeQueue<T>::shutdown() {
    {
        // Scope block to minimize lock duration
        std::lock_guard<std::mutex> lock(mtx);
        shutdown_flag = true;
    }

    // Notify ALL threads to wake up and check the shutdown_flag
    cv.notify_all();
}
```

Serial

Input_Worker

این ترد به وبکم متصل میشود و FPS ورودی را برای استفاده در بخش خروجی ذخیره میکند.

سپس در یک loop فریم هارا کپی میکند و درون ThreadSafeQueue خروجی میدهد. همچنین برای توقف برنامه از ورودی کیبورد منتظر 2 ورودی q یا esc میماند تا برنامه را پایان دهد. برای پایان دادن برنامه تحت یک قرار داد یک فریم خالی به عنوان فریم نهایی را درون صف ارسال میکند، با این کار worker های بعدی نیز فریم خالی را به جلو درون صف raw_queue ارسال میکنند تا اتمام برنامه اتفاق بیفتد.

```
while (true) {
    // 3. Read a new frame from video capture
    cap >> frame;

    // 4. Check if we succeeded (if frame is empty, stream might have ended)
    if (frame.empty()) {
        std::cerr << "[Input] Error: Blank frame grabbed or camera disconnected." << std::endl;
        // send a signal to worker using empty frame
        my_args->raw_queue->push(cv::Mat());
        break;
    }

    // 5. Push the frame to the thread-safe queue
    // Note: frame.clone() is often safer in threading to ensure
    // a deep copy of image data is passed, preventing data races
    // if OpenCV reuses the memory of 'frame' in the next iteration.
    my_args->raw_queue->push(frame.clone());

    // Optional: Log every X frames to avoid console spam
    // std::cout << "[Input] Pushed frame to queue." << std::endl;

    if (_kbhit()) {
        // _getch() reads the key without waiting for Enter
        int key = _getch();

        // 113 is ASCII for 'q', 27 is ASCII for 'ESC'
        if (key == 'q' || key == 113 || key == 27) {
            cout << "Stopping recording..." << endl;
            // send a signal to worker using empty frame
            my_args->raw_queue->push(cv::Mat());
            break;
        }
    }
}
```

Process_Worker

این ترد وظایف محاسباتی را بر عهده دارد، از درون صف raw_input هر فریم را pop میکند، محاسبات پردازش تصویر الگوریتم sobel و سپس push کردن نتیجه درون result_queue.

همچنین وجود سیگنال قرار دادی (فریم خالی) را چک میکند تا درصورت اتمام برنامه از حلقه بیرون آید و سیگنال اتمام را به pipeline بعدی ارسال کند.

```cpp
// 2. Process Worker
void* process_worker(void* args) {
    ProcessArgs* my_args = static_cast<ProcessArgs*>(args);

    std::cout << "[Process] Worker started (Sobel Algo)." << std::endl;

    while (true) {
        // 1. Receive frame from queue
        cv::Mat frame = my_args->raw_queue->pop();

        // Check for stop signal (Poison Pill)
        if (frame.empty()) {
            std::cout << "[Process] Stop signal received." << std::endl;
            // Forward the empty frame to the next queue to stop the output worker
            my_args->result_queue->push(frame);
            break;
        }

        // 2. Convert to Grayscale
        cv::Mat gray;
        cv::cvtColor(frame, gray, cv::COLOR_BGR2GRAY);

        // 3. Apply Sobel Edge Detection
        cv::Mat grad_x, grad_y;
        cv::Mat abs_grad_x, abs_grad_y;
        cv::Mat detected_edges;

        // Gradient X
        // ddepth = CV_16S to avoid overflow/negative values issues
        cv::Sobel(gray, grad_x, CV_16S, 1, 0, 3, 1, 0, cv::BORDER_DEFAULT);

        // Gradient Y
        cv::Sobel(gray, grad_y, CV_16S, 0, 1, 3, 1, 0, cv::BORDER_DEFAULT);

        // Converting back to CV_8U (Absolute value)
        cv::convertScaleAbs(grad_x, abs_grad_x);
        cv::convertScaleAbs(grad_y, abs_grad_y);

        // 4. Combine X and Y edges
        // Total Gradient (approximate) = 0.5 * |grad_x| + 0.5 * |grad_y|
        cv::addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, detected_edges);

        // 5. Send to result queue
        my_args->result_queue->push(detected_edges);
    }

    std::cout << "[Process] Worker stopped." << std::endl;
    return NULL;
}
```

Output_worker

این ترد اخرین عضو pipeline است، فریم هارا از درون result_queue pop میکند و درون فایل خروجی مینویسد. درصورت رسیدن سیگنال اتمام نیز از حلقه خارج میشود.

```cpp
// 3. Output Worker
void* output_worker(void* args) {
    OutputArgs* my_args = static_cast<OutputArgs*>(args);

    cv::VideoWriter writer;
    bool is_writer_initialized = false;
    long frame_count = 0;

    // Timer variables
    auto start_time = std::chrono::steady_clock::now();

    std::cout << "[Output] Worker started. Waiting for frames..." << std::endl;

    while (true) {
        // 1. Receive frame from queue
        cv::Mat frame = my_args->result_queue->pop();

        // Check for stop signal
        if (frame.empty()) {
            std::cout << "[Output] Stop signal received. Finalizing..." << std::endl;
            break;
        }

        // 2. Initialize VideoWriter with the first frame
        if (!is_writer_initialized) {
            // Using MJPG codec
            int fourcc = cv::VideoWriter::fourcc('M', 'J', 'P', 'G');

            double fps = *(my_args->shared_fps);
            if (fps <= 0) fps = 30.0;

            cv::Size frame_size = frame.size();

            // IMPORTANT: isColor = false because Sobel output is Grayscale
            bool isColor = false;

            writer.open(my_args->filename, fourcc, fps, frame_size, isColor);

            if (!writer.isOpened()) {
                std::cerr << "[Output] Error: Could not open video writer!" << std::endl;
                break; // Exit if we can't save
            }

            // Start the timer when the first frame arrives
            start_time = std::chrono::steady_clock::now();
            is_writer_initialized = true;
            std::cout << "[Output] VideoWriter initialized. Recording..." << std::endl;
        }

        // 3. Save processed frame
        writer.write(frame);
        frame_count++;

        // Optional: Show progress every 30 frames
        if (frame_count % 30 == 0) {
            std::cout << "." << std::flush;
        }
    }
```

همچنین در نهایت نتایج کلی مانند fps و تعداد همه فریم هارا نمایش میدهد

```cpp
// 4. Calculate Average FPS
auto end_time = std::chrono::steady_clock::now();
std::chrono::duration<double> elapsed_seconds = end_time - start_time;

if (elapsed_seconds.count() > 0) {
    double avg_fps = frame_count / elapsed_seconds.count();
    std::cout << "\n[Output] Processing Finished." << std::endl;
    std::cout << "[Output] Total Frames: " << frame_count << std::endl;
    std::cout << "[Output] Average FPS:  " << avg_fps << std::endl;
}
```

Pthread

مدیریت هر یک از این ترد ها با Pthread انجام میشود، از آنجایی که ساختار Pthread_create تنها اجازه ارسال یک آرگومان وروذی برای فراخوانی یک تابع را میدهد. داده های ورودی هر pipeline بصورت یک struct مشخص طراحی شده اند که در بخش های قبل توضیح داده شده اند.

```cpp
struct InputArgs {
    ThreadSafeQueue<cv::Mat>* raw_queue;
    int camera_id; // Added camera ID to make it flexible
    double* shared_fps;
};

struct ProcessArgs {
    ThreadSafeQueue<cv::Mat>* raw_queue;
    ThreadSafeQueue<cv::Mat>* result_queue;
};

struct OutputArgs {
    ThreadSafeQueue<cv::Mat>* result_queue;
    std::string filename;
    double* shared_fps;
};
```

در نهایت درون main برای هر یک از این توابع یک thread ایجاد میشود. و در نهایت نیز main به ترتیب ایجاد هر یک از توابع منتظر join شدن تمامی اشان میماند.

Parallel

در این بخش ما برای ترد محاسباتی که همان Process_worker میباشد موازی سازی با openMP انجام میدهیم.

این موازی سازی در دو بخش انجام میشود:

گرادیان

محاسبه گرادیان عمودی و افقی، که هردو از آنجایی که مستقل از یک دیگر هستند بصورت موازی انجام میشود. که به عبارتی تسک ها موازی میشوند.

```
// --- OPENMP SECTION 1: Task Parallelism ---
// Calculate Sobel X and Sobel Y at the same time using different threads
#pragma omp parallel sections
{
    #pragma omp section
    {
        // Calculate Gradient X
        cv::Sobel(gray, grad_x, CV_16S, 1, 0, 3, 1, 0, cv::BORDER_DEFAULT);
    }

    #pragma omp section
    {
        // Calculate Gradient Y
        cv::Sobel(gray, grad_y, CV_16S, 0, 1, 3, 1, 0, cv::BORDER_DEFAULT);
    }
}
```

ترکیب لبه ها

در مرحله نهایی الگوریتم، ترکیب قدر مطلق گرادیان ها برای ساخت تصویر نهاییست، که در بخش serial با استفاده از تابع آماده سریال addWeighted هندل میشد. اما در این بخش ما با موازی سازی داده ها در یک حلقه ردیف هارا بین ترد ها تقسیم میکنیم.

```
// Parallelize the loop over rows
#pragma omp parallel for
for (int i = 0; i < rows; ++i) {
    // Get row pointers for fast access
    short* row_x = grad_x.ptr<short>(i);
    short* row_y = grad_y.ptr<short>(i);
    uchar* row_out = detected_edges.ptr<uchar>(i);

    for (int j = 0; j < cols; ++j) {
        // Calculate absolute values manually
        short val_x = std::abs(row_x[j]);
        short val_y = std::abs(row_y[j]);

        // Weighted sum: 0.5 * x + 0.5 * y
        // We perform saturation cast to ensure it fits in uchar (0-255)
        int sum = (val_x + val_y) / 2;

        // Clamp/Saturate the result
        row_out[j] = cv::saturate_cast<uchar>(sum);
    }
}
```

Output

برای مقایسه این دو برنامه از یک ویدیو یکسان به عنوان ورودی استفاده شده است.

Serial

```
[Output] Processing Finished.
[Output] Total Frames: 901
[Output] Time:  13.6845
[Output] Average FPS:  65.8409
[Main] Output thread joined.
[Main] Application finished successfully.
```

Parallel

```
[Output] Processing Finished.
[Output] Total Frames: 901
[Output] Time:  12.2221
[Output] Average FPS:  73.719
[Main] Output thread joined.
[Main] Application finished successfully.
```

با این حال بدلیل گلوگاه اصلی این برنامه و محدود بودن سرعت به بخش های I/O با موازی سازی بخش process_worker speedup زیادی حاصل نمیشود.