Optimizing Video Input for Embedded/Edge Devices

I specialize in video analytics solutions for **embedded/edge devices**, particularly in **security systems** and **robotics**. One of the most critical challenges in these domains is **performance optimization**.

Below, I'll demonstrate a conventional camera processing algorithm using **C/C++ pseudo-code**, followed by an optimized version that improves throughput by **at least 1.5×**.

Standard Approach (Using OpenCV)

The typical implementation (similar to OpenCV's samples: videocapture_starter.cpp, videocapture_camera.cpp) processes frames in a separate thread to avoid sync issues:

```
atomic<bool> ready = false;
cv::Mat image;

void thread_function() {
    while(true) {
        if (ready) {
            // Processing (e.g., object detection, streaming)
            ready = false;
        }
    }
}

void camera_loop() {
    cv::Mat img;
    cv::VideoCapture capture("rtsp://....");
    create_thread(thread_function);

    while(true) {
        capture >> img;
        if (!ready) {
            img.copyTo(image);  // Copy operation introduces overhead
            ready = true;
        }
    }
}
```

Drawback: `img.copyTo()` creates unnecessary memory copies, throttling performance.

Optimized Algorithm: Pointer Swapping + Double Buffering

By using **two frame buffers** and **pointer swapping**, we eliminate redundant copies:

```
atomic<bool> ready = false;
cv::Mat* pimage;

void thread_function() {
    while(true) {
        if (ready) {
            // Processing (e.g., object recognition, streaming)
            ready = false;
        }
    }
```

```
}

void camera_loop() {
    cv::Mat img[2];
    int i = 0;
    cv::VideoCapture capture("rtsp://....");
    create_thread(thread_function);

    while(true) {
        if (!ready && !img[i].empty()) {
            pimage = &img[i];  // Swap pointer (zero-copy)
            i = 1 - i;          // Toggle buffer index
            ready = true;
        }
        capture >> img[i];      // Async capture into inactive buffer
    }
}
```

Key Advantages

1. **Zero-Copy Optimization**: Pointer swaps replace memory-heavy `copyTo()`.
2. **Double Buffering**: Overlaps capture and processing (no pipeline stalls).
3. **Scalability**: Adaptable to other sensors (microphones, LiDAR, etc.).

Why This Matters

- **Edge Devices**: Critical for latency-sensitive applications (e.g., drones, surveillance).
- **Resource Efficiency**: Reduces CPU/GPU load, extending battery life in IoT systems.

Discussion: Could this be further optimized with triple buffering or lock-free queues? Share your thoughts!