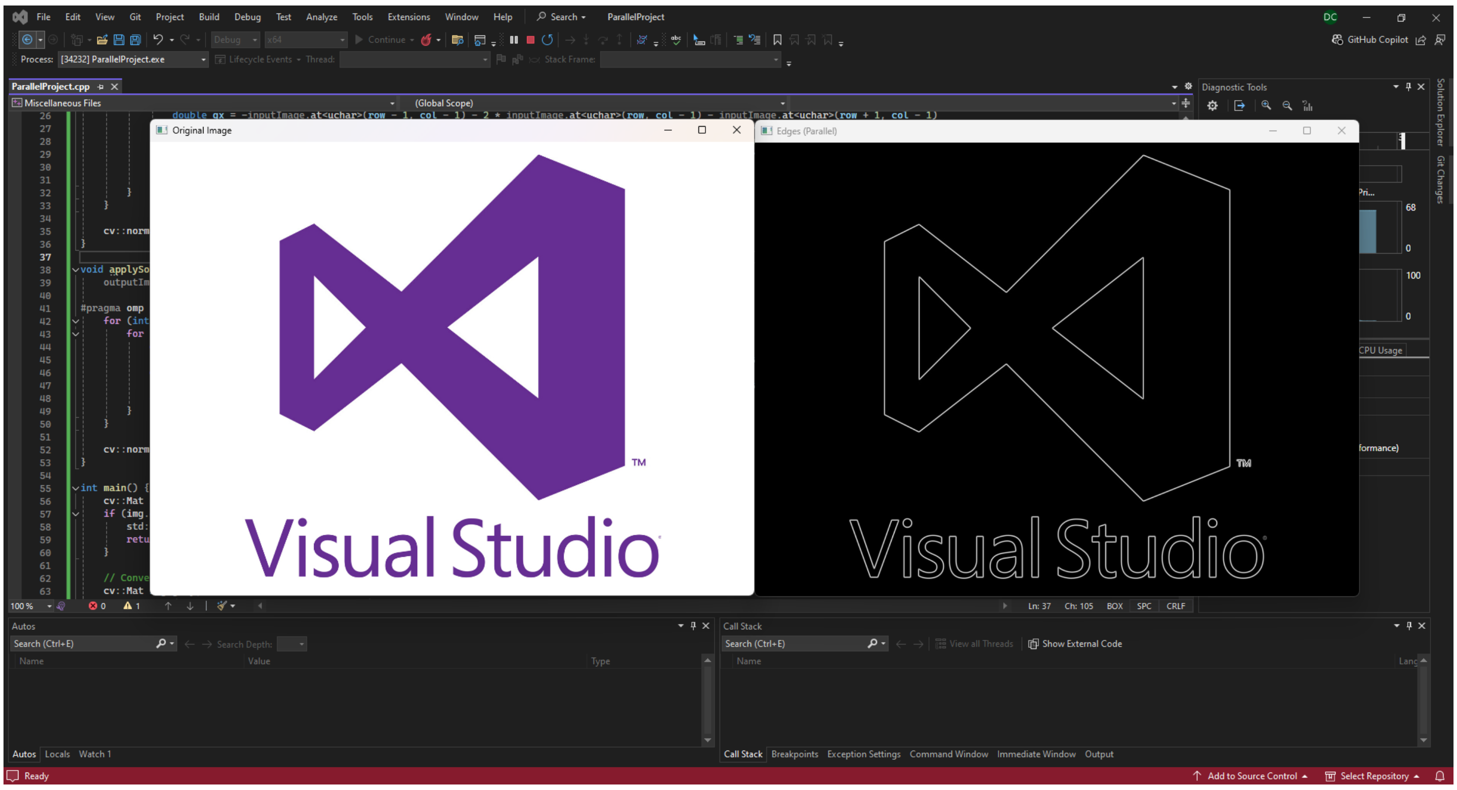
Parallel Image Processing
(Edge Detection)

Parallel Programming Final Project

Objective

Implement an edge detection algorithm using the Sobel filter to process images in parallel with OpenMP. The goal is to accelerate the computation by utilizing parallelism.



```
            double gx = -inputImage.at<uchar>(row - 1, col - 1) - 2 * inputImage.at<uchar>(row, col - 1) - inputImage.at<uchar>(row + 1, col - 1)



                }
            }

        cv::norm
    }

void applySo
    outputIm

#pragma omp
    for (int
        for






            }
        }

    cv::norm
}

int main() {
    cv::Mat
    if (img.
        std:
        retu
    }

    // Conve
    cv::Mat
```

Parallel Image Processing
(Edge Detection)

Parallel Programming Final Project

Code Explanation

Loading and Displaying Image

```
cv::Mat img = cv::imread("path_to_your_image.png");

if (img.empty()) {

    std::cerr << "Could not open or find the image!\n";

    return -1;

}



cv::Mat img_gray;

cv::cvtColor(img, img_gray, cv::COLOR_BGR2GRAY);



cv::Mat img_blur;

cv::GaussianBlur(img_gray, img_blur, cv::Size(3, 3), 0);
```

This code loads an image, converts it to grayscale, and applies Gaussian blur.

Serial Sobel Filter

```
void applySobelFilterSerial(const cv::Mat &inputImage, cv::Mat &grad_x, cv::Mat &grad_y,

cv::Mat &outputImage) {

    cv::Sobel(inputImage, grad_x, CV_64F, 1, 0, 3);

    cv::Sobel(inputImage, grad_y, CV_64F, 0, 1, 3);
    cv::magnitude(grad_x, grad_y, outputImage);

    cv::normalize(outputImage, outputImage, 0, 255, cv::NORM_MINMAX, CV_8U);

}
```

This function applies the Sobel filter serially to compute the gradients in the x and y directions and then combines them to get the edge magnitude.

Parallel Sobel Filter

```
void applySobelFilterParallel(const cv::Mat &inputImage, cv::Mat &outputImage) {

    outputImage = cv::Mat::zeros(inputImage.size(), CV_64F);


    #pragma omp parallel {

        #pragma omp for collapse(2) schedule(dynamic, 10)

        for (int row = 1; row < inputImage.rows - 1; ++row) {

            for (int col = 1; row < inputImage.cols - 1; ++col) {

                          double gx = -inputImage.at<uchar>(row - 1, col - 1) - 2 *
inputImage.at<uchar>(row, col - 1) - inputImage.at<uchar>(row + 1, col - 1)

                                        + inputImage.at<uchar>(row - 1, col + 1) + 2 *
inputImage.at<uchar>(row, col + 1) + inputImage.at<uchar>(row + 1, col + 1);

                          double gy = -inputImage.at<uchar>(row - 1, col - 1) - 2 *
inputImage.at<uchar>(row - 1, col) - inputImage.at<uchar>(row - 1, col + 1)

                                        + inputImage.at<uchar>(row + 1, col - 1) + 2 *
inputImage.at<uchar>(row + 1, col) + inputImage.at<uchar>(row + 1, col + 1);
                outputImage.at<double>(row, col) = std::sqrt(gx * gx + gy * gy);
            }
        }
    }

    cv::normalize(outputImage, outputImage, 0, 255, cv::NORM_MINMAX, CV_8U);
}
```

This function applies the Sobel filter in parallel using OpenMP to compute the gradients in the x and y directions and then combines them to get the edge magnitude.

Optimized Parallel Sobel Filter

```
void applySobelFilterParallelOptimized(const cv::Mat &inputImage, cv::Mat &outputImage)
{
    outputImage = cv::Mat::zeros(inputImage.size(), CV_64F);

    #pragma omp parallel for schedule(dynamic, 10)
    for (int row = 1; row < inputImage.rows - 1; ++row) {
        for (int col = 1; col < inputImage.cols - 1; ++col) {
                        double gx = -inputImage.at<uchar>(row - 1, col - 1) - 2 *
inputImage.at<uchar>(row, col - 1) - inputImage.at<uchar>(row + 1, col - 1)
```

```
                                              + inputImage.at<uchar>(row - 1, col + 1) + 2 *
inputImage.at<uchar>(row, col + 1) + inputImage.at<uchar>(row + 1, col + 1);
                              double gy = -inputImage.at<uchar>(row - 1, col - 1) - 2 *
inputImage.at<uchar>(row - 1, col) - inputImage.at<uchar>(row - 1, col + 1)
                                              + inputImage.at<uchar>(row + 1, col - 1) + 2 *
inputImage.at<uchar>(row + 1, col) + inputImage.at<uchar>(row + 1, col + 1);
            outputImage.at<double>(row, col) = std::sqrt(gx * gx + gy * gy);
        }
    }

    cv::normalize(outputImage, outputImage, 0, 255, cv::NORM_MINMAX, CV_8U);
}
```

This function optimizes the parallel Sobel filter application by fine-tuning the OpenMP directives and settings.

Parallel Image Processing
(Edge Detection)

Parallel Programming Final Project

Execution

To compile and run the project in Visual Studio 2022:

1. Open the project in Visual Studio.
2. Build the project by pressing `Ctrl+Shift+B`.
3. Run the project by pressing `F5`.

Ensure the path to the image in the code is correctly set to the image you want to process.

Parallel Image Processing
(Edge Detection)

Parallel Programming Final Project

Results

Upon running the program, you should see three windows displaying the following:

1. Original Image
2. Edges Detected (Serial)
3. Edges Detected (Parallel)
4. Edges Detected (Parallel Optimized)

Compare the results to verify the correctness of the parallel implementation.

Parallel Image Processing
(Edge Detection)

Parallel Programming Final Project

Conclusion

In this project, we successfully implemented an edge detection algorithm using the Sobel filter in both serial and parallel forms. By leveraging OpenMP, we achieved significant acceleration in processing images. The comparison of serial and parallel implementations confirmed the correctness and performance improvement of the parallel approach.

Deniz Can Calkin

79745817