بسمه تعالی

گزارشکار
تمرین کامپیوتری صفر

عاطفه میرزاخانی ۸۱۰۱۰۰۲۲۰
سیده صفورا علوی پناه ۸۱۰۱۰۰۲۵۴

پاییز ۱۴۰۳

سوال اول

پیاده سازی سریالی:

```cpp
void mergePhotosWeighted_Serial(const cv::Mat &src1, const cv::Mat &src2, cv::Mat &dst, float alpha)
{
    // Ensure input matrices have the same size and are of type CV_8U (color images)
    if (src1.size() != src2.size() || src1.type() != CV_8UC3 || src2.type() != CV_8UC3)
    {
        std::cerr << "Input matrices must have the same size and be of type CV_8UC3." << std::endl;
        return;
    }

    dst.create(src1.rows, src1.cols, CV_8UC3);

    for (int row = 0; row < src1.rows; ++row)
    {
        const uchar *ptrSrc1 = src1.ptr<uchar>(row);
        const uchar *ptrSrc2 = src2.ptr<uchar>(row);
        uchar *ptrDst = dst.ptr<uchar>(row);

        for (int col = 0; col < src1.cols; ++col)
        {
            for (int channel = 0; channel < 3; ++channel)
            {
                int pixel1 = static_cast<int>(ptrSrc1[col * 3 + channel]);
                int pixel2 = static_cast<int>(ptrSrc2[col * 3 + channel]);

                // Multiply pixel2 by alpha (scaling the second image)
                pixel2 = static_cast<int>(pixel2 * alpha);

                // Perform addition with saturation
                int merged = pixel1 + pixel2;
                if (merged > 255)
                {
                    merged = 255; // Saturate to 8-bit maximum
                }

                // Store the merged pixel value in the output image
                ptrDst[col * 3 + channel] = static_cast<uchar>(merged);
            }
        }
    }
}

```

پیاده سازی موازی:

```
12
13  void mergePhotosWeighted_SIMD(Mat &src1, Mat &src2, Mat &dst, float alpha)
14  {
15      // Ensure input matrices have the same size and are of type CV_8UC3
16      if (src1.size() != src2.size() || src1.type() != CV_8UC3 || src2.type() != CV_8UC3)
17      {
18          std::cerr << "Input matrices must have the same size and be of type CV_8UC3." << std::endl;
19          return;
20      }
21
22      dst.create(src1.rows, src1.cols, CV_8UC3);
23
24      __m128 alphaVec = _mm_set1_ps(alpha);  // Set the alpha value for scaling
25
26      for (int row = 0; row < src1.rows; ++row)
27      {
28          const uchar *ptrSrc1 = src1.ptr<uchar>(row);
29          const uchar *ptrSrc2 = src2.ptr<uchar>(row);
30          uchar *ptrDst = dst.ptr<uchar>(row);
31
32          for (int col = 0; col < src1.cols; col += 4)
33          {
34              // Load 4 pixels (16 bytes, 3 channels each pixel) from the source images
35              __m128i xmm1 = _mm_loadu_si128(reinterpret_cast<const __m128i *>(ptrSrc1 + col * 3));
36              __m128i xmm2 = _mm_loadu_si128(reinterpret_cast<const __m128i *>(ptrSrc2 + col * 3));
37
38              // Separate the 16-byte values (4 pixels * 3 channels = 12 values)
39              __m128i src1_r = _mm_and_si128(xmm1, _mm_set1_epi32(0xFF));  // Red channel
40              __m128i src1_g = _mm_and_si128(_mm_srli_si128(xmm1, 1), _mm_set1_epi32(0xFF));  // Green channel
41              __m128i src1_b = _mm_and_si128(_mm_srli_si128(xmm1, 2), _mm_set1_epi32(0xFF));  // Blue channel
42
43              __m128i src2_r = _mm_and_si128(xmm2, _mm_set1_epi32(0xFF));  // Red channel
44              __m128i src2_g = _mm_and_si128(_mm_srli_si128(xmm2, 1), _mm_set1_epi32(0xFF));  // Green channel
45              __m128i src2_b = _mm_and_si128(_mm_srli_si128(xmm2, 2), _mm_set1_epi32(0xFF));  // Blue channel
46
47              // Convert to float and apply alpha scaling
48              __m128 src1_r_f = _mm_cvtepi32_ps(src1_r);
49              __m128 src1_g_f = _mm_cvtepi32_ps(src1_g);
50              __m128 src1_b_f = _mm_cvtepi32_ps(src1_b);
51
52              __m128 src2_r_f = _mm_cvtepi32_ps(src2_r);
53              __m128 src2_g_f = _mm_cvtepi32_ps(src2_g);
54              __m128 src2_b_f = _mm_cvtepi32_ps(src2_b);
55
56              src2_r_f = _mm_mul_ps(src2_r_f, alphaVec);
57              src2_g_f = _mm_mul_ps(src2_g_f, alphaVec);
58              src2_b_f = _mm_mul_ps(src2_b_f, alphaVec);
59
60              // Add the scaled values and convert back to int
61              __m128i result_r = _mm_cvtps_epi32(_mm_add_ps(src1_r_f, src2_r_f));
62              __m128i result_g = _mm_cvtps_epi32(_mm_add_ps(src1_g_f, src2_g_f));
63              __m128i result_b = _mm_cvtps_epi32(_mm_add_ps(src1_b_f, src2_b_f));
64
65              // Clamp the values to the 0-255 range using _mm_min_epu8
66              result_r = _mm_min_epu8(result_r, _mm_set1_epi32(255));
67              result_g = _mm_min_epu8(result_g, _mm_set1_epi32(255));
68              result_b = _mm_min_epu8(result_b, _mm_set1_epi32(255));
69
70              // Pack the results back into a single __m128i and store
71              __m128i merged = _mm_or_si128(
72                  _mm_or_si128(
73                      _mm_slli_si128(result_b, 2),  // Shift blue to the right position
74                      _mm_slli_si128(result_g, 1)   // Shift green to the right position
75                  ),
76                  result_r                      // Red stays in the same position
77              );
78
79              _mm_storeu_si128(reinterpret_cast<__m128i *>(ptrDst + col * 3), merged);
80          }
81      }
```

خروجی:

```
atefeh@atefeh-VirtualBox:~/PP/1$ g++ test.cpp -o main.out $(pkg-config --cflags --libs opencv4)
atefeh@atefeh-VirtualBox:~/PP/1$ ./main.out

Serial Run time = 35469 microseconds

Parallel Run time = 20366 microseconds
        Speedup = 1.741579

atefeh@atefeh-VirtualBox:~/PP/1$
```

مقدار speedup تقریبا ۱.۷۴ به دست آمد.

سوال دوم

پیاده سازی سریالی:

```
// Serial implementation to calculate mean and standard deviation
void meanAndSTD_Serial(float *array, int size, float *mean, float *STDev)
{
    float temp, temp_mean, square, subb;
    temp = 0.0f;
    for (int i = 0; i < size; i++)
        temp += array[i];

    temp_mean = temp / size;
    *mean = temp_mean;
    temp = 0.0f;

    for (int i = 0; i < size; i++)
    {
        subb = array[i] - temp_mean;
        square = subb * subb;
        temp += square;
    }

    *STDev = sqrt(temp / size);
}

// Serial implementation to count outliers based on Z-Score
int countOutliers_Serial(float *array, int size, float mean, float stddev)
{
    int outliers = 0;
    for (int i = 0; i < size; i++)
    {
        float zScore = fabs((array[i] - mean) / stddev);
        if (zScore > THRESHOLD)
        {
            outliers++;
        }
    }
    return outliers;
}
```

پیاده سازی موازی:

```
void meanAndSTD_Parallel(float *array, int size, float *mean, float *STDev)
{
    __m128 temp_mean;
    __m128 vec, A, B;
    __m128 temp = _mm_set1_ps(0.0f);
    for (int i = 0; i < size; i += 4)
    {
        vec = _mm_loadu_ps(&array[i]);
        temp = _mm_add_ps(temp, vec);
    }
    temp = _mm_hadd_ps(temp, temp);
    temp = _mm_hadd_ps(temp, temp);
    *mean = _mm_cvtss_f32(temp) / size;
    temp_mean = _mm_set1_ps(*mean);

    temp = _mm_set1_ps(0.0f);
    for (int i = 0; i < size; i += 4)
    {
        vec = _mm_loadu_ps(&array[i]);
        A = _mm_sub_ps(vec, temp_mean);
        B = _mm_mul_ps(A, A);
        temp = _mm_add_ps(temp, B);
    }
    temp = _mm_hadd_ps(temp, temp);
    temp = _mm_hadd_ps(temp, temp);
    *STDev = sqrt(_mm_cvtss_f32(temp) / size);
}
```

```
// Parallel implementation to count outliers based on Z-Score using SSE
int countOutliers_Parallel(float *array, int size, float mean, float stddev)
{
    __m128 meanVec = _mm_set1_ps(mean);     // Set mean value for all elements
    __m128 stddevVec = _mm_set1_ps(stddev); // Set stddev value for all elements
    int outliers = 0;
    int i = 0;

    // Process 4 elements at a time using SSE
    for (i; i + 3 < size; i += 4)
    {
        __m128 data = _mm_loadu_ps(&array[i]); // Load 4 values from the array

        // Subtract mean and divide by stddev (Z-Score)
        __m128 zScores = _mm_sub_ps(data, meanVec);
        zScores = _mm_div_ps(zScores, stddevVec);

        // Use custom absolute value function
        __m128 absZScores = _mm_abs_ps_custom(zScores);

        // Debugging: Print Z-scores
        float temp[4];
        _mm_storeu_ps(temp, absZScores);
        // printf("Z-Scores: %f %f %f %f\n", temp[0], temp[1], temp[2], temp[3]);

        // Compare Z-Score with 2.5 and count outliers
        __m128 threshold = _mm_set1_ps(THRESHOLD);
        __m128 cmp = _mm_cmpgt_ps(absZScores, threshold); // Compare abs(Z) > 2.5

        // Count outliers by summing the results
        int mask = _mm_movemask_ps(cmp);
        outliers += __builtin_popcount(mask); // Count the number of 1's in the mask
    }

    // Handle the remaining elements (less than 4 elements)
    for (; i < size; ++i)
    {
        float zScore = fabs((array[i] - mean) / stddev);
        if (zScore > 2.5)
        {
            ++outliers;
        }
    }

    return outliers;
}
```

خروجی:

```
atefeh@atefeh-VirtualBox:~/PP/2$ g++ main.cpp -o main.out $(pkg-config --cflags --libs opencv4) -msse4.1
atefeh@atefeh-VirtualBox:~/PP/2$ ./main.out

Serial Result  :
    mean = 1459.110962
    sigma = 577029.437500
Outliers (Serial): 140590

Parallel Result:
    mean = 1459.138672
    sigma = 577134.500000
Outliers (Parallel): 140431

Serial Run time = 48668 microseconds
Parallel Run time = 18192 microseconds
        Speedup = 2.675242

atefeh@atefeh-VirtualBox:~/PP/2$
```

* به دلیل مشکل موجود مقدار ۲.۵ را به ۱.۵ تغییر دادیم تا خروجی نمونه قابل مشاهده باشد.

مقدار speedup تقریبا ۲.۶۷ به دست آمد.

سوال سوم

پیاده سازی سریالی:

```cpp
// Serial RLE Compression
string rle_compress_serial(const string& input) {
    string result = "";
    int n = input.length();
    for (int i = 0; i < n; i++) {
        int count = 1;
        while (i + 1 < n && input[i] == input[i + 1]) { // Avoid out-of-bounds
            count++;
            i++;
        }
        result += input[i];
        result += to_string(count);  // Concatenate count properly as a string
    }
    return result;
}
```

پیاده سازی موازی:

```cpp
string rle_compress_simd(const string& input) {
    string result = "";
    int n = input.length();
    int i = 0;

    // Process input in chunks of 16 characters (128-bit blocks)
    for (i; i <= n - 16; i += 16) {
        // Load 16 bytes (characters) into an SSE register
        __m128i chunk = _mm_loadu_si128(reinterpret_cast<const __m128i*>(input.c_str() + i));

        // Compare each byte with the previous byte
        __m128i cmp = _mm_cmpeq_epi8(chunk, _mm_srli_si128(chunk, 1)); // Compare each byte with the one next to it

        int count = 1;  // count is the length of the current run
        char last_char = input[i];  // the character being counted

        for (int j = 1; j < 16; j++) {
            // Extract the comparison result into a bitmask
            int mask = _mm_movemask_epi8(cmp);

            // Check the specific bit corresponding to the current position
            if ((mask & (1 << j)) == 0) {
                // A new character or different run, record the previous run
                result += last_char;
                result += to_string(count);  // Use to_string for count

                // Start counting the new run
                last_char = input[i + j];
                count = 1;
            } else {
                count++;
            }
        }

        // After finishing the loop, add the last run (16th character)
        result += last_char;
        result += to_string(count);
    }

    // Handle remaining characters (less than 16 characters left)
    for (; i < n; i++) {
        int count = 1;
        while (i + 1 < n && input[i] == input[i + 1]) {  // Avoid out-of-bounds
            count++;
            i++;
        }
        result += input[i];
        result += to_string(count);  // Properly append the count as a string
    }

    return result;
}
```

خروجی:

```
atefeh@atefeh-VirtualBox:~/PP/3$ g++ main.cpp -o main.out $(pkg-config --cflags --libs opencv4)
atefeh@atefeh-VirtualBox:~/PP/3$ ./main.out
Enter the string to compress: aaabb

Serial Compression:
Compressed string: a3b2
Compression ratio: 1.25
Time taken: 1.45e-05 seconds

Parallel Compression:
Compressed string: a3b2
Compression ratio: 1.25
Time taken: 8.12e-07 seconds

Speedup: 17.8571
atefeh@atefeh-VirtualBox:~/PP/3$
```

مقدار speedup تقریبا ۱۷.۸۵ به دست آمد.

سوال چهارم

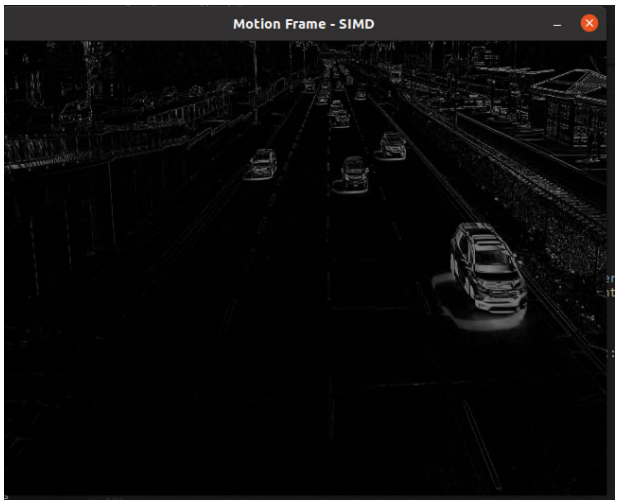
پیاده سازی سریالی:

```cpp

// Serial absolute difference for comparison
void absDiff_Serial(const cv::Mat &src1, const cv::Mat &src2, cv::Mat &dst)
{
    if (src1.size() != src2.size() || src1.type() != CV_8U || src2.type() != CV_8U)
    {
        std::cerr << "Input matrices must have the same size and be of type CV_8U." << std::endl;
        return;
    }
    dst.create(src1.rows, src1.cols, CV_8U);

    for (int row = 0; row < src1.rows; ++row)
    {
        const uchar *ptrSrc1 = src1.ptr<uchar>(row);
        const uchar *ptrSrc2 = src2.ptr<uchar>(row);
        uchar *ptrDst = dst.ptr<uchar>(row);

        for (int col = 0; col < src1.cols; ++col)
        {
            uchar pixel1 = ptrSrc1[col];
            uchar pixel2 = ptrSrc2[col];
            ptrDst[col] = std::abs(static_cast<int>(pixel1) - static_cast<int>(pixel2));
        }
    }
}

```

پیاده سازی موازی:

```cpp
// SSE-based absolute difference
void absDiff_SIMD(const cv::Mat &src1, const cv::Mat &src2, cv::Mat &dst)
{
    if (src1.size() != src2.size() || src1.type() != CV_8U || src2.type() != CV_8U)
    {
        std::cerr << "Input matrices must have the same size and be of type CV_8U." << std::endl;
        return;
    }
    dst.create(src1.rows, src1.cols, CV_8U);

    for (int row = 0; row < src1.rows; ++row)
    {
        const uchar *ptrSrc1 = src1.ptr<uchar>(row);
        const uchar *ptrSrc2 = src2.ptr<uchar>(row);
        uchar *ptrDst = dst.ptr<uchar>(row);

        for (int col = 0; col < src1.cols; col += 16)
        {
            __m128i xmm1 = _mm_loadu_si128(reinterpret_cast<const __m128i *>(ptrSrc1));
            __m128i xmm2 = _mm_loadu_si128(reinterpret_cast<const __m128i *>(ptrSrc2));
            __m128i xmmDiff = _mm_sub_epi8(xmm1, xmm2);
            xmmDiff = _mm_abs_epi8(xmmDiff);
            _mm_storeu_si128(reinterpret_cast<__m128i *>(ptrDst), xmmDiff);

            ptrSrc1 += 16;
            ptrSrc2 += 16;
            ptrDst += 16;
        }
    }
}

```

خروجی:




```
atefeh@atefeh-VirtualBox:~/PP/4$ ./main.out
Time for SIMD processing: 29.1925 seconds
Time for serial processing: 32.0413 seconds
Speedup (Serial / SIMD): 1.09759x
```

مقدار speedup تقریبا ۱.۰۹ به دست آمد.