به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

برنامه نویسی موازی

گزارش کار پروژه‌ی اول

نام و نام خانوادگی	امیرحسین ثمودی – آرمین قاسمی
شماره دانشجویی	810100198 – 810100108
تاریخ ارسال گزارش	1403/8/20

فهرست گزارش سوالات

سوال اول: Image Blending 3

بخش سریال: 3

بخش موازی: 4

بدست آوردن SpeedUp: 5

سوال دوم: 5

بخش سریال: 5

بخش موازی: 7

بدست آوردن SpeedUp: 8

سوال سوم: 9

بخش سریال: 9

بخش موازی: 9

بدست آوردن SpeedUp و سایر نتایج: 10

سوال چهارم: تشخیص حرکت (Motion Detection) 11

بخش سریال: 11

بخش موازی: 12

بدست آوردن SpeedUp: 13

سوال اول: Image Blending

در این سوال قصد داریم عکس لوگوی دانشگاه را با یک تصویر پس زمینه با فرمول ذکر شده در صورت پروژه ترکیب نماییم.

بخش سریال:

برای انجام بهینه بخش سریال به این نکته توجه داریم که ضریبی که در Logo ضرب میشود (0.625) را میتوان به صورت جمع یک بار شیفت یافته (0.5) و 3 بار شیفت یافته (0.125) محاسبه نمود. با این کار دیگر نیازی به ضرب Float نیست و زمان اجرای برنامه سریال بهبود میابد.

تابعی که برای این بخش نوشته ایم به صورت زیر میباشد:

```
void blendSerial( Mat& front, Mat& logo, Mat& output) {
    const int logoRows = logo.rows;
    const int logoCols = logo.cols;

    uchar* frontRow;
    uchar* logoRow;
    int blendedLogo;

    for (int i = 0; i < logoRows; i++) {
        frontRow = output.ptr<uchar>(i);
        logoRow = logo.ptr<uchar>(i);

        for (int j = 0; j < logoCols * 3; j++) {
            // Approximate blend factor by shifting
            blendedLogo = (logoRow[j] >> 1) + (logoRow[j] >> 3);
            frontRow[j] = cv::saturate_cast<uchar>(frontRow[j] + blendedLogo);
        }
    }
}
```

کد سریال سوال1

همچنین توجه داریم که به دلیل رنگی بودن تصویر در هر پیکسل 3 کانال RGB داریم برای همین تعداد تکرار های حلقه در هر سطر 3 برابر تعداد ستون ها میباشد.

خروجی برنامه برای اجرای سریال به صورت زیر میباشد:

خروجی برنامه (اجرای سری)

بخش موازی:

برای انجام این عملیات به صورت موازی داده های دو تصویر را به صورت پک های 16 تایی از بایت ها میخوانیم. برای محاسبه 0.5*Logo ابتدا لوگو را یک بار با دستور _mm_srli_epi16 شیفت میدهیم (دستور SIMD ئی برای شیفت داده های 8 بیت نداریم). سپس برای اینکه اثر بیت های شیفت یافته بایت مجاور که بیت MSB بیت سمت راستش میشود با دستور _mm_and_si128 آن را در اعداد 8 بیتی 0X7F که معادل باینری 01111111 است and مینماییم. حال همینکار را برای ضریب 0.125 نیز انجام داده با این تفاوت که 3 بیت شیفت به راست داده و بایت ها را با 0X1F که معادل باینری 00011111 است and میکنیم. در نهایت این دو مقدار شیفت یافته لوگو و تصویر پس زمینه را با دستور جمع SIMD جمع مینماییم و در تصویر خروجی مینویسیم. کد این بخش به صورت زیر میباشد:

```
// Parallel SIMD blending function
void blendParallel( Mat& front, Mat& logo, Mat& output) {
    const int logoRows = logo.rows;
    const int logoCols = logo.cols;

    __m128i frontPixels, logoPixels, result;
    __m128i logo_shift_1, logo_shift_3;
    __m128i MSB1Zero = _mm_set1_epi8(0x7F);
    __m128i MSB3Zero = _mm_set1_epi8(0x1F);

    uchar* frontRow;
    uchar* logoRow;

    for (int i = 0; i < logoRows; i++) {
        frontRow = output.ptr<uchar>(i);
        logoRow = logo.ptr<uchar>(i);

        for (int j = 0; j <= (logoCols * 3) - 16; j += 16) {
            // Load 16 consecutive bytes
            frontPixels = _mm_loadu_si128((__m128i*)(frontRow + j));
            logoPixels = _mm_loadu_si128((__m128i*)(logoRow + j));

            // Approximate blend factor by shifting
            logo_shift_1 = _mm_srli_epi16(logoPixels, 1);
            logo_shift_1 = _mm_and_si128(MSB1Zero, logo_shift_1);

            logo_shift_3 = _mm_srli_epi16(logoPixels, 3);
            logo_shift_3 = _mm_and_si128(MSB3Zero, logo_shift_3);

            result = _mm_adds_epu8(frontPixels, _mm_adds_epu8(logo_shift_1, logo_shift_3));

            // Store result
            _mm_storeu_si128((__m128i*)(frontRow + j), result);
        }
    }
```

خروجی برنامه برای اجرای موازی نیز به صورت زیر میباشد:

خروجی برنامه (اجرای موازی)

بدست آوردن SpeedUp:

برای محاسبه SpeedUp از کتابخانه ipp استفاده میکنیم و تعداد کلاک سایکل های هر اجرا را بدست می اوریم. جدول زیر مقادیر SpeedUp را برای 8 اجرای برنامه را نشان میدهد.

اجرا	1	2	3	4	5	6	7	8	میانگین
SpeedUp	4.224	13.185	7.080	4.515	6.396	7.413	5.313	5.905	6.754

مشاهده میکنیم که SpeedUp ما در حالت موازی نسبت به صورت میانگین 6.754 میباشد.

سوال دوم:

پس از تولید 2^{20} عدد float به صورت تصادفی، بخش های سریال و موازی را به این شکل مینویسیم:

بخش سریال:

```
start = __rdtsc();
float s_sum = 0;
float sum_1 = 0;
float sum_2 = 0;
float sum_3 = 0;
float sum_4 = 0;
```

```
for (int i = 0; i < numRandomFloats ; i += 4) {
    sum_1 += randomFloats[i];
}
for (int i = 1; i < numRandomFloats ; i += 4) {
    sum_2 += randomFloats[i];
}
for (int i = 2; i < numRandomFloats ;  i += 4) {
    sum_3 += randomFloats[i];
}
for (int i = 3; i < numRandomFloats ; i+=4) {
    sum_4 += randomFloats[i];
}
s_sum = sum_1 + sum_2 + sum_3 + sum_4;
float s_m = s_sum / numRandomFloats;

float s_sigma2 = 0;
float sigma1 = 0;
float sigma2 = 0;
float sigma3 = 0;
float sigma4 = 0;


for (int i = 0; i < numRandomFloats; i+=4) {
    sigma1 += (randomFloats[i] - s_m) * (randomFloats[i] - s_m);
}
for (int i = 1; i < numRandomFloats ; i += 4) {
    sigma2 += (randomFloats[i] - s_m) * (randomFloats[i] - s_m);
}
for (int i = 2; i < numRandomFloats ; i += 4) {
    sigma3 += (randomFloats[i] - s_m) * (randomFloats[i] - s_m);
}
for (int i = 3; i < numRandomFloats ; i += 4) {
    sigma4 += (randomFloats[i] - s_m) * (randomFloats[i] - s_m);
}

s_sigma2 = sigma1 + sigma2;
s_sigma2 += (sigma3 + sigma4);
float s_v = s_sigma2 / numRandomFloats;
float s_s = sqrtf(s_v);

int s_o_count = 0;
for (int i = 0; i < numRandomFloats; i++) {
    if ((randomFloats[i] - s_m) / s_s > 2.5)
    {
        s_o_count += 1;
    }
}
```

```
    end = __rdtsc();
    s_t = end - start;
```

با توجه به اینکه اعداد ما float هستند و اعداد float تنها بخشی از اعداد حقیقی را پوشش میدهند، ترتیب جمع اعداد برای ما اهمیت دارد. بنابراین آنها را به همان ترتیب که در بخش موازی جمع می کنیم باهم جمع میکنیم. سایر عملیات ها هم به همین شکل است. صرفا عملیات آخر که صرفا شمارش اعداد پرت است، می تواند به صورت عادی انجام شود.

بخش موازی:

```
start = __rdtsc();
    __m128 sum_vec = _mm_setzero_ps();

    for (int i = 0; i < numRandomFloats; i += 4) {
        __m128 vec = _mm_loadu_ps(&randomFloats[i]);
        sum_vec = _mm_add_ps(sum_vec, vec);
    }

    sum_vec = _mm_hadd_ps(sum_vec, sum_vec);
    sum_vec = _mm_hadd_ps(sum_vec, sum_vec);
    float p_sum = _mm_cvtss_f32(sum_vec);
    float p_m = p_sum / numRandomFloats;


    __m128 variance_vec = _mm_setzero_ps();
    __m128 mean_vec = _mm_set1_ps(p_m);

    for (int i = 0; i < numRandomFloats; i += 4) {
        __m128 vec = _mm_loadu_ps(&randomFloats[i]);
        __m128 diff = _mm_sub_ps(vec, mean_vec);
        __m128 diff_sq = _mm_mul_ps(diff, diff);
        variance_vec = _mm_add_ps(variance_vec, diff_sq);
    }

    variance_vec = _mm_hadd_ps(variance_vec, variance_vec);
    variance_vec = _mm_hadd_ps(variance_vec, variance_vec);
    float p_v = _mm_cvtss_f32(variance_vec);
    p_v = p_v / numRandomFloats;
    float p_s = sqrtf(p_v);

    __m128 threshold_vec = _mm_set1_ps(2.5f);
    __m128 stddev_vec = _mm_set1_ps(p_s);

    int p_o_count = 0;

    for (int i = 0; i < numRandomFloats; i += 4) {
```

```
        __m128 vec = _mm_loadu_ps(&randomFloats[i]);
        __m128 diff = _mm_sub_ps(vec, mean_vec);
        __m128 normalized = _mm_div_ps(diff, stddev_vec);
        __m128 mask = _mm_cmpgt_ps(normalized, threshold_vec);


        int mask_int = _mm_movemask_ps(mask);
        p_o_count += popcount(mask_int);
    }
    end = __rdtsc();
    p_t = end - start;
```

در این بخش همان الگوریتم قبلی را به صورت برداری انجام میدهیم. به این صورت که در هر بار تکرار حلقه 4 عدد float را می خوانیم و باهم جمع میکنیم. نهایتا هم از _mm_hadd_ps برای جمع سطری اعضای وکتور استفاده میکنیم. عملیات های تفریق و توان و نهایتا محاسبه واریانس و انحراف معیار را نیز به همین صورت انجام میدهیم.

برای شمارش تعداد داده های پرت، دوباره چهار تا چهار تا از داده های خود می خوانیم، میانگین را از آنها کم میکنیم و تقسیم بر واریانس میکنیم. نهایتا این اعداد را با برداری که در هر خانه آن 2.5 نوشته شده است مقایسه می کنیم. پس از این کار می توانیم از تابع _mm_movemask_ps استفاده کنیم که به ازای هر مقایسه یک بیت صفر یا یک دریافت کنیم و بعد با تابع popcount تعداد یک ها را میشماریم و به p_o_count اضافه میکنیم.

بدست آوردن SpeedUp:

```
Speed-Up = 2.73013
Number of outliers (Paralell): 0
Number of outliers (Serial): 0
Clock cycles (Paralell): 5.62795e+07
Clock cycles (Serial): 1.5365e+08
mean (Paralell): 0.499668
mean (Serial): 0.499668
variance (Paralell): 0.0832845
variance (Serial): 0.0832845
```

```
Speed-Up = 2.54662
Number of outliers (Paralell): 0
Number of outliers (Serial): 0
Clock cycles (Paralell): 6.20609e+07
Clock cycles (Serial): 1.58045e+08
mean (Paralell): 0.499585
mean (Serial): 0.499585
variance (Paralell): 0.0833507
variance (Serial): 0.0833507
```

```
Speed-Up = 2.80072
Number of outliers (Paralell): 0
Number of outliers (Serial): 0
Clock cycles (Paralell): 5.65511e+07
Clock cycles (Serial): 1.58384e+08
mean (Paralell): 0.500013
mean (Serial): 0.500013
variance (Paralell): 0.0834545
variance (Serial): 0.0834545
```

```
Speed-Up = 2.06626
Number of outliers (Paralell): 0
Number of outliers (Serial): 0
Clock cycles (Paralell): 7.1898e+07
Clock cycles (Serial): 1.4856e+08
mean (Paralell): 0.49971
mean (Serial): 0.49971
variance (Paralell): 0.0832323
variance (Serial): 0.0832323
```

مشاهده میکنیم که SpeedUp به طور میانگین 2.53 میباشد.

سوال سوم:

پس از تولید 100000 کاراکتر به صورت تصادفی، بخش های سریال و موازی را به این شکل مینویسیم:

بخش سریال:

```
    double t_start = __rdtsc();
    vector<pair<char, int>> standardCompressed;
    char currentChar = data[0];
    int count = 1;
    for (size_t i = 1; i < data.size(); ++i) {
        if (data[i] == currentChar) {
            ++count;
        }
        else {
            standardCompressed.push_back({ currentChar, count });
            currentChar = data[i];
            count = 1; }
    }
    standardCompressed.push_back({ currentChar, count });
    double t_end = __rdtsc();
    double ser = double(t_end - t_start);
    cout << "serial time " << ser << " cc" << endl;
```

در این بخش رشته کاراکتر ها را طی میکنیم و خروجی را میسازیم. currentChar را نگه میداریم و با count تعداد آن را میشماریم. درصورتی که این کاراکتر با کاراکتربعدی برابر باشد count را زیاد می کنیم و در غیر این صورت currentChar را آپدیت و count را به 1 ریست می کنیم.

بخش موازی:

```
t_start = __rdtsc();
    vector<pair<char, int>> simdCompressed;
    int i = 0;
    __m128i chunk;
    char prevChar;
    int cnt;
    while (i + 15 < n) {
        chunk = _mm_loadu_si128(reinterpret_cast<const
__m128i*>(&data[i]));
        i += 16;
        prevChar = data[i - 16];
        cnt = 1;
```

```
        char curChar;
        for (int j = 1; j < 16 && i - 16 + j < data.size(); ++j) {
            curChar = reinterpret_cast<char*>(&chunk)[j];
            if (curChar == prevChar) {
                ++cnt;
            }
            else {
                simdCompressed.push_back({ prevChar, cnt });
                prevChar = curChar;
                cnt = 1;
            }
        }
        simdCompressed.push_back({ prevChar, cnt });
    }
    char curChar;

    while (i < data.size()) {
        curChar = data[i];
        int cnt = 1;
        ++i;
        while (i < data.size() && data[i] == curChar) {
            ++cnt;
            ++i;
        }
        simdCompressed.push_back({ curChar, cnt });
    }
    t_end = __rdtsc();
    double par = double(t_end - t_start);
```

در بخش موازی بلوک های 16 کاراکتری را لود می کنیم و باهم پردازش میکنیم و روال قبل را طی میکنیم همچنین پس از پردازش بلوک‌های 16 کاراکتری، کاراکترهای باقی‌مانده به روش عادی پردازش می‌شوند.

بدست آوردن SpeedUp و سایر نتایج:

```
serial time 305160 cc
paralell time253142 cc
Verification (Standard RLE):Success
Verification (SIMD RLE): Success
speedup 1.20549
compression ratio serial0.95
```

```
serial time 5.0053e+06 cc
paralell time3.18405e+06 cc
Verification (Standard RLE):Success
Verification (SIMD RLE): Success
speedup 1.57199
compression ratio serial0.958496
```

```
serial time 1.99272e+07 cc
paralell time8.14953e+06 cc
Verification (Standard RLE):Success
Verification (SIMD RLE): Success
speedup 2.44519
compression ratio serial0.960693
```

```
serial time 3.75778e+07 cc
paralell time1.22916e+07 cc
Verification (Standard RLE):Success
Verification (SIMD RLE): Success
speedup 3.05718
compression ratio serial0.958862
```

0

مشاهده میکنیم که SpeedUp به طور میانگین 2.265 میباشد.

سوال چهارم: تشخیص حرکت (Motion Detection)

در این سوال میخواهیم با پردازش یک ویدیو حرکت های موجود در ویدیو را دیتکت کنیم. روش این کار به این صورت است که هر دو فریم متوالی را از هم کم کرده و قدر مطلق میگیریم.

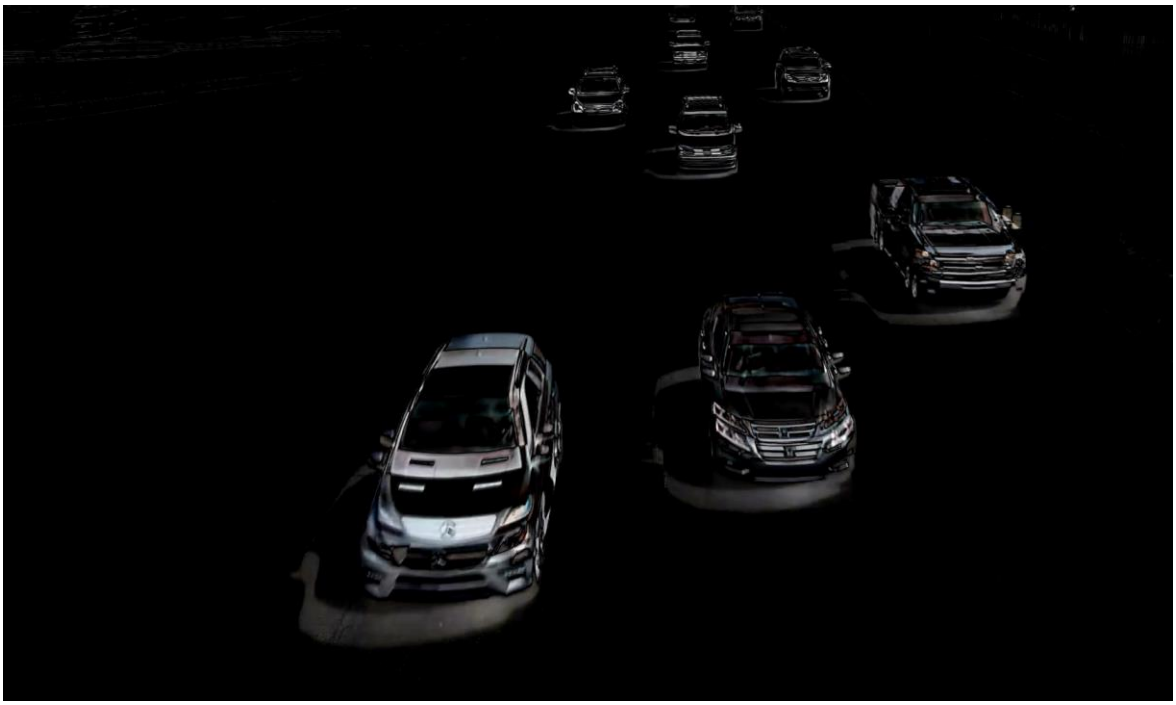
بخش سریال:

در این بخش هر دو فریم متوالی را میخوانیم و سپس بر روی همه پیکسل ها حلقه زده و با فرمول داده شده پیکسل خروجی را محاسبه میکنیم.

```
void serialMotionDetection(cv::VideoCapture& cap, cv::Mat& previous_frame, cv::Mat& motion_frame, cv::VideoWriter& output) {
    cv::Mat current_frame;
    int diff;

    while (cap.read(current_frame)) {
        for (int i = 0; i < current_frame.rows; i++) {
            uchar* prevRow = previous_frame.ptr<uchar>(i);
            uchar* currRow = current_frame.ptr<uchar>(i);
            uchar* motionRow = motion_frame.ptr<uchar>(i);

            for (int j = 0; j < current_frame.cols * 3; j++) {
                diff = currRow[j] - prevRow[j];
                motionRow[j] = static_cast<uchar>(std::abs(diff));
            }
        }
        output.write(motion_frame);
        previous_frame = current_frame.clone();
    }
}
```

کد سریال سوال 4

یک فریم از ویدیو خروجی به صورت زیر میباشد:



خروجی برنامه سریال

بخش موازی:

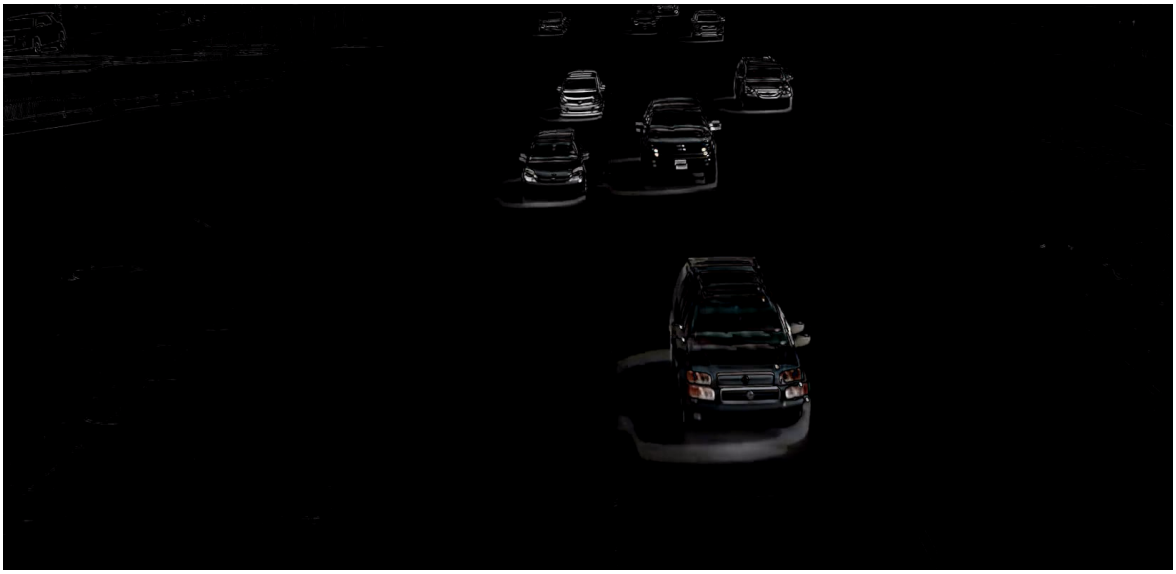
برای اجرای بخش موازی هر 16 بایت متوالی از دو فریم متوالی را میخوانیم و با دستور_mm_subs_epu8 از هم کم میکنیم. نکته ای که وجود دارد بدلیل اینکه دستور SIMD قدر مطلق برای اعداد بدون علامتی که ما با آن سر و کار داریم وجود ندارد ابتدا با دستور _mm_max_epu8(prevPixels, currPixels) بیشینه مقادیر را بدست اورده و با دستور _mm_min_epu8(prevPixels, currPixels) کمینه را بدست اورده و سپس از دستور _mm_subs_epu8 استفاده میکنیم تا مطمئن شویم خروجی 8 بیتی ما نیز عددی منفی نمیشود.

بقیه مراحل کار نیز مانند همان کد سریال میباشد. کد موازی به صورت زیر میباشد:

```
void parallelMotionDetection(cv::VideoCapture& cap, cv::Mat& previous_frame, cv::Mat& motion_frame, cv::VideoWriter& output) {
    cv::Mat current_frame;
    __m128i prevPixels, currPixels, diff;
    uchar* prevRow,* currRow,* motionRow;

    while (cap.read(current_frame)) {
        for (int i = 0; i < current_frame.rows; i++) {
            prevRow = previous_frame.ptr<uchar>(i);
            currRow = current_frame.ptr<uchar>(i);
            motionRow = motion_frame.ptr<uchar>(i);

            for (int j = 0; j <= (current_frame.cols * 3) - 16; j += 16) {
                prevPixels = _mm_loadu_si128((__m128i*)(prevRow + j));
                currPixels = _mm_loadu_si128((__m128i*)(currRow + j));

                diff = _mm_subs_epu8(_mm_max_epu8(prevPixels, currPixels), _mm_min_epu8(prevPixels, currPixels));
                _mm_storeu_si128((__m128i*)(motionRow + j), diff);
            }
            /*for (; j < current_frame.cols * 3; j++) {
                int diff = currRow[j] - prevRow[j];
                motionRow[j] = static_cast<uchar>(std::abs(diff));
            }*/
        }
        output.write(motion_frame);
        previous_frame = current_frame.clone();
    }
}
```

کد موازی سوال4

یک فریم از خروجی تولید شده به صورت زیر میباشد:



خروجی برنامه موازی

بدست آوردن SpeedUp:

برای محاسبه SpeedUp از کتابخانه ipp استفاده میکنیم و تعداد کلاک سایکل های هر اجرا را بدست می اوریم. جدول زیر مقادیر SpeedUp را برای 8 اجرای برنامه را نشان میدهد.

اجرا	1	2	3	4	میانگین
SpeedUp	2.02	2.11	2.33	2.31	2.19

مشاهده میکنیم که با موازی کردن عملیات پردازش تصویر speedup مان به طور میانگین به 2.19 میرسد.