Факултет Техничких Наука

Трг Доситеја Обрадовића 6,

Нови Сад 106314

Предметни Пројекат из Паралелног Програмирања

Замена Боје На Слици

Студенти:

Душица Трбовић, SV 42/2021

Ана Попарић, SV 74/2021

Весна Васић, SV 78/2021

Нови Сад, мај 2023. год.

САДРЖАЈ

1. ОПИС ПРОЈЕКТНОГ ЗАДАТКА

Тема пројектног задатка била је замена боје на слици. Почетна идеја била је да се кориснику омогући унос слике, бирање боје коју жели да промени на учитаној слици, као и боја у коју жели да пиксели буду промењени. Како не живимо у перфектном свету схватили смо да би наша идеја била слабо применљива и ипак смо се одлучили да би боља опција била да проширимо опсег пиксела који ће се мењати. Па тако приликом избора боје коју корисник жели да промени, постоји праг толеранције који омогућава програму да замени пикселе који су веома сличне боје, под претпоставком да је циљ замена одређене целине која се налази на слици која је унета.

Главне функционалности које су морале бити имплементиране су:

- Интеракција са корисницима
  - Учитавање слике са рачунара
  - Избор боје пиксела који требају бити промењени
  - Избор боје у коју одређени пиксели требају бити промењени
- Проналажење 3 најчешће појављиване боје на слици
- Замена боје на слици
- Примена *blur* филтера

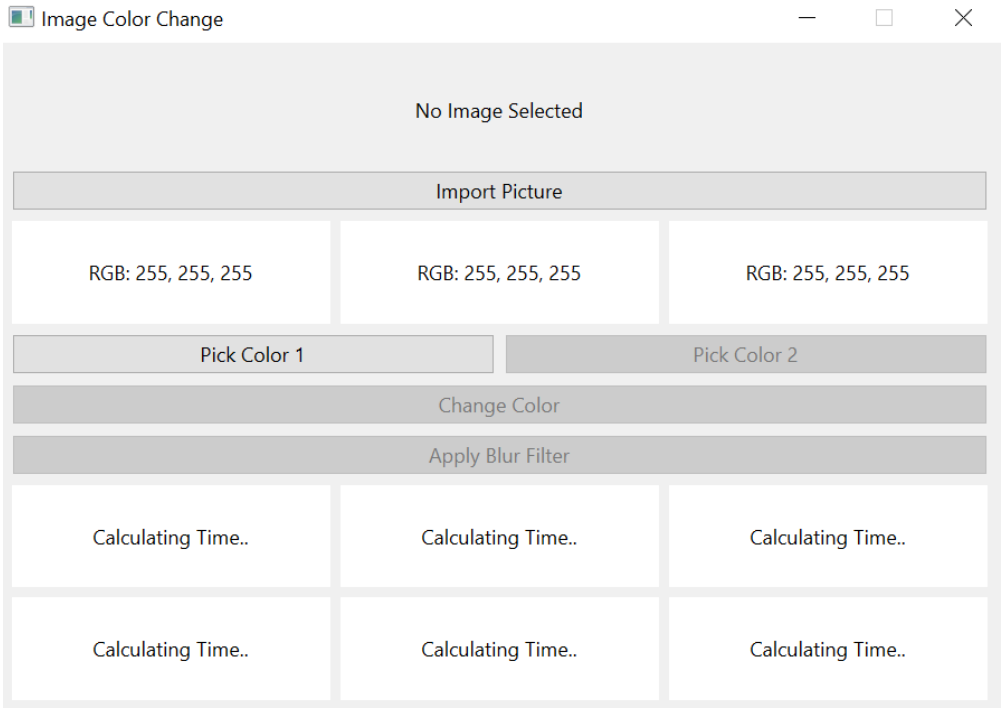
Интеракција са корисницима је омогућена коришћењем библиотеке *Qt.*

Последње три фнкционалности су подложне паралелизацији и на њих ћемо се фокусирати током овог пројекта. Идеја је била испитивање разлика у времену које је потребно за обављање сваке од ових функција понаособ, на различитим примерима, тј на сликама различитих резолуција.

2. РАД У/И ПОДСИСТЕМА

2.1. Изглед почетног прозора

Приликом покретања апликације, кориснику се отвара прозор са подразумеваним вредностима. Приказује се информација да слика још увек није учитана, као и дугме *Import Picture*, кликом на исто отвара се могућност да корисник изабере слику над којом жели да изврши промене.
Испод Дугмета *Import Picture* налази се простор где су приказане 3 најчешће појављиване боје са слике, подразумеване боје су 3 бела поља.
Након простора у ком се налазе боје, смештена су два дугмета *Pick Color 1* и *Pick Color 2* која кориснику омогућавају да изабере боју коју жели да промени као и боју у коју жели да изврши промену.



Слика 1 - Изглед Почетног Прозора

На стартном прозору је такође видљива опција 2 дугмета која ће омогућавати опције за промену боје на слици и примену филтера. Ова два дугмета су онемогућена за коришћење све док корисник не унесе слику над којом би наведене функције могле да се позову.
Времена су у процесу рачунања, тј функције за које се иста времена рачунају још увек нису покренуте, па ће тај изглед да се задржи све док се једна од функција не буде позвана и извршена.

2.2 Учитавање фотографије

Приликом клика на дугме *Import Picture* позива се функција *openImageDialog()* која уз помоћу путање коју корисник уноси учитава жељену слику. Извршава се провера да ли слика постоји на задатој путањи и ако је све у реду, слика се поставља на простор који је намењен за њу (горњи део стартног прозора). Дугмад која омогућавају примене функционалности нашег пројекта у овом моменту постају кликабилна.

```
void ImageColorChange::openImageDialog()
{
    QString imagePath = QFileDialog::getOpenFileName(this, "Select Image");
    if (!imagePath.isEmpty()) {
        QImage image(imagePath);
        imageLabel->setPixmap(QPixmap::fromImage(image).scaled(300, 300,
                                                               Qt::KeepAspectRatio));
        pickerButton1->setEnabled(true);
        changeColorButton->setEnabled(true);
        filterButton->setEnabled(true);

        timeLabel1->setText(QString("Calculating time.."));
        timeLabel2->setText(QString("Calculating time.."));
        timeLabel3->setText(QString("Calculating time.."));
        timeLabelParallel1->setText(QString("Calculating time.."));
        timeLabelParallel2->setText(QString("Calculating time.."));
        timeLabelParallel3->setText(QString("Calculating time.."));

        mostCommonColorsGUI();
    }
}
```

При самом крају функције позива се функција која служи за проналажење 3 најчешће појављиване боје на учитаној слици.

2.3 Избор боја

За избор боја задужене су две веома сличне функције, које за циљ имају имену два различита објекта, па су самим тим и одвојене.

```
void ImageColorChange::pickColor1()
{
    QColor color = QColorDialog::getColor(Qt::white, this, "Pick Color 1");
    if (color.isValid()) {
        pickerButton1->setStyleSheet(QString("background-color: %1")
                                                               .arg(color.name()));
        pickerButton1->setText(QString("RGB: %1, %2, %3")
                    .arg(color.red()).arg(color.green()).arg(color.blue()));
        pickerButton2->setEnabled(true);
    }
}
```

Функција *pickColor1()* отвара палету са бојама и омогућава кориснику да изабере боју коју жели да измени или у коју жели да промени већ изабране пикселе. Приликом избора, врши се провера да ли је боја валидна и након тога се мења позадина дугмета, као и сам текст, који након избора садржи *RGB* вредност изабране боје.

3. ФУНКЦИЈЕ ПОДЛОЖНЕ ПАРАЛЕЛИЗАЦИЈИ

Програм је подељен на три главне функције. Прва служи за Проналазак 3 најћешће појављиване боје на слици, друга Измену боја, и трећа за Примену *blur* филтера.

3.1. Функција „mostCommonColorsGUI()“

Функција је заправо задужена за контролу над подацима који се налазе на прозору који кроисник види, не и самим извршавањем логике. За то постоје две функције које се позивају унутар *mostCommonColorsGUI()*. Ова функција задужена је и за рачунање времена које је неопходно за ирвшавање једне од две подфункције које ће заправо извршавати саму логику.

```
QElapsedTimer timer;
timer.start();

QVector<QColor> sortedColors = findMostCommonColors(image);

qint64 elapsedTime = timer.elapsed();
timeLabel1->setText(QString("Most Common Serial: %1s").arg(QString::number
                                                (elapsedTime / 1000.0, 'f', 4)));
```

Након извршавања функције, израчунато време се поставља на одређену лабелу и приказује кориснику на прозору. У променљиву *sortedColors* се стављају резулати саме функције, тј *RGB* вредности 3 најчешће појављиване боје на слици. Након извршене и паралелизоване верзије ове функције резултати се постављају на лабеле које су намењене за приказ ових података.

```
if (!sortedColors.isEmpty()) {
      QColor color1 = sortedColors[0];
      colorLabel1->setStyleSheet(QString("background-color: %1").arg
                                                        (color1.name()));
      colorLabel1->setText(QString("RGB: %1, %2, %3").arg
                  (color1.red()).arg(color1.green()).arg(color1.blue()));
}
```

3.1.1. Фунцкија „findMostCommonColors(const QImage& image)“

При самом почетку пролазимо кроз сваки пиксел слике и бројимо понављања одређене боје. Како се подаци смештају у тип *Qhash*, уколико боја до тог тренутка није пронађена на слици креираће се нов пар типа боја-број понављања, док уколико је боја раније пронађена на слици се број понављања повећава за 1.

```
QHash<QColor, int> colorCounts;

for (int i = 0; i < image.height(); ++i) {
    for (int j = 0; j < image.width(); ++j) {
         QColor color = image.pixelColor(j, i);
         colorCounts[color]++;
    }
}
```

Након проласка кроз сваки пиксел слике и прикупљања информација о броју понављања одређене боје неопходно је извршити сортирање тих података.

```
QList<QColor> sortedColors = colorCounts.keys();
std::sort(sortedColors.begin(), sortedColors.end(), [&](const QColor& color1,
                        const QColor& color2) {
                              return colorCounts[color1] > colorCounts[color2];
                        });
```

Сортирани подаци се смештају у нову променљиву која ће након брисања непотребних елемената (сваки осим прва 3, уколико постоје), бити прослеђена као повратна вредност.

3.1.2. Паралелизована Фунцкија „findMostCommonColorsParallel(const QImage& image)“

Након уочавања могућности за паралелизацију претходне функције одређено је да је најбољи начин за извршавање исте подела слике на више мањих региона.

```
if (std::min(image.height(), image.width()) <= CUTOFF) {
        return findMostCommonColors(image);
}
```

Први корак је провера величине слике, тј провера да ли је унесена слика кандидат за паралелизацију. Уколико јесте извршава се подела на више региона.

```
QRect topLeftRect(0, 0, image.width() / 2, image.height() / 2);
QRect topRightRect(image.width() / 2, 0, image.width() / 2, image.height() / 2);
```

Како сваки регион мора бити обрађен, а циљ нам је паралелно извршавање, посао ћемо да поделимо на више *taskGroup*-ова. Сваки од њих ће позивати рекурзију, све док се не деси довољно велика подела да регион за обраду буде довољно мали и испуњава услов за серијско извршавање.

```
taskGroup.run([&] { partialResults[0] = findMostCommonColorsParallel
                                                (image.copy(topLeftRect));});
taskGroup.run([&] { partialResults[1] = findMostCommonColorsParallel
                                                (image.copy(topRightRect));};
```

Када је сваки *taskGroup* завршен прикупљени су сви непходни подаци који се прослеђују као повратна вредност функције.

3.2. Функција „colorChangingGUI ()“

На самом почетку функције прикупљамо информације које ће нам бити неопходне за позив серијске и паралелизоване подфункције. Прва јесте слика над којом се промена боје одвија, друга и трећа су боје (боја која се мења и циљна боја у коју је потребно изменити пикселе), последња инфромација која нам је неопходна јесте праг толеранције приликом измене боја, ово ће да нам омогући измену веома сличних боја на слици, у циљу лакшег рада над једном смисленом целином.

```
QImage image = imageLabel->pixmap().toImage();

QColor targetColor = pickerButton1->palette().button().color();
QColor replacementColor = pickerButton2->palette().button().color();

const int colorTolerance = 50;
```

Ова функција такође је задужена за мерење времена извршавања серијске и паралелизоване функције, као и ажурирање слике након промене боја.

3.2.1. Фунцкија „replaceColor(QImage& image, const QColor& targetColor, const QColor& replacementColor, int colorTolerance)“

Серијска верзија функције пролази кроз сваки пиксел слике и проверава да ли је у опсегу боја које су подобне за промену, уколико јесте, пиксел мења на циљну боју.

```
for (int y = 0; y < image.height(); ++y) {
    for (int x = 0; x < image.width(); ++x) {
        QColor pixelColor(image.pixel(x, y));
        if (isSimilarColor(pixelColor, targetColor, colorTolerance)) {
                image.setPixel(x, y, replacementColor.rgb());
        }
    }
}
```

Помоћна функција која постоји у овој функцији јесте *isSimilarColor* она враћа *bool* вредост која нам даје до знања да ли *RGB* вредност припада дозвољеном опсегу.

3.2.2. Фунцкија „replaceColorParallel(QImage& image, const QColor& targetColor, const QColor& replacementColor, int colorTolerance)“

Приликом писања паралелизоване верзије претходне фунције неопходно је било проверити да ли је слика кандидат за паралелизацију.

```
if (std::min(image.height(), image.width()) <= CUTOFF) {
     return replaceColor(image, targetColor, replacementColor, colorTolerance);
}
```

Уколико слика ипак није кандидат за исту, позива се серијска верзија функције. Ако је случај пак другачији, слика се дели на више региона и формирају се задаци тј *taskGroup*-ови. Свакоме ће се доделити одређени регион, уз позивање фунцкије *replaceColorParallel*, тј поновно прављење рекурзије, све док не наиђемо на случај довољно једноставан да је подложан решавању преко серијске функције.

```
taskGroup.run([&] { replaceColorParallel(bottomLeftImage, targetColor,
                                    replacementColor, colorTolerance); });
taskGroup.run([&] { replaceColorParallel(bottomRightImage, targetColor,
                                    replacementColor, colorTolerance); });

taskGroup.wait();
```

Након извршења сваког задатка понаособ, *taskGroup* синхронизује сва решења и добијамо крајњи резултат, тј слику са замењеним пикселима.

3.3. Функција „applyFilterGUI ()“

Слично као и предходне две функције, сама *GUI* функција задужена је за манипулисање подацима који су видљиви кориснику на прозору, као и мерење времена извршавања серијске и паралелизоване функције.

3.3.1. Фунцкија „applyFilter(QImage& sourceImage)“

Функција која примењује алгоритам за сортирање користи заправо *Gauss Blur* филтер. Да би ово било могуће неопходно је израчунати пар константи. Величина језгра заправо одређује колики ће вектор да се користи, док *sigma* представља стандардну девијацију Гаусове дистрибуције. Када је матрица креирана, постављамо вредност суме на 0 и пролазимо кроз сваки елемент вектора.

```
std::vector<double> kernel(kernelSize * kernelSize);
double sum = 0.0;
for (int y = -radius; y <= radius; ++y) {
   for (int x = -radius; x <= radius; ++x) {
       double value = std::exp(-(x * x + y * y) / (2 * sigma * sigma));
       kernel[(y + radius) * kernelSize + (x + radius)] = value;
       sum += value;
   }
}
```

За сваки елемент на позицији (x, y) у вектору кренела користимо формулу која одговара Гаусовој дводимензионалној расподели. Вредност коју смо израчунали остаје на позицији *(y + radius) * kernelSize + (x + radius*) у вектору али се и додаје на укупну суму, коју ћемо користити приликом нормализације.

```
for (double& value : kernel) {
   value /= sum;
}
```

Наредни део кода нормализује вредности у *kernel* вектору. Нормализација се обавља тако да се свака вредност дели са збиром свих вредности (претходно израчунатим у *sum* променљивој). Ово осигурава да сума свих вредности у *kernel* вектору постаје 1, што је неопходно за правилно примењивање Гаусовог филтера.

```
for (int y = 0; y < sourceImage.height(); ++y) {
   for (int x = 0; x < sourceImage.width(); ++x) {
   double red = 0.0, green = 0.0, blue = 0.0, alpha = 0.0;

     for (int dy = -radius; dy <= radius; ++dy) {
        for (int dx = -radius; dx <= radius; ++dx) {
           int sx = qBound(0, x + dx, sourceImage.width() - 1);
           int sy = qBound(0, y + dy, sourceImage.height() - 1);
           QColor pixelColor(sourceImage.pixel(sx, sy));
           double weight = kernel[(dy + radius) * kernelSize + (dx + radius)];
           red += pixelColor.redF() * weight;
           green += pixelColor.greenF() * weight;
           blue += pixelColor.blueF() * weight;
           alpha += pixelColor.alphaF() * weight;
        }
      }
   QColor blurredColor(qRound(red * 255.0), qRound(green * 255.0), qRound(blue *
                                                 255.0), qRound(alpha * 255.0));
   sourceImage.setPixel(x, y, blurredColor.rgba());
  }
}
```

Овај део кода примењује Гаусов филтер на слику. Приликом примене филтера, за сваки пиксел слике се пролази кроз вектор *kernel* и примењују се одговарајуће вредности kernela на суседне пикселе. Процес се понавља за сваки пиксел слике, што резултира применом Гаусовог замагљења на целу слику.

3.3.2. Фунцкија „applyFilterParallel(QImage& sourceImage)“

У паралелизованој функцији процес примене филтера је подељен на више задатака, тј региона слике. Дељење се одвија све док се не пронађе довољно мали регион подобан за извршавање серијске функције.

```
// Divide the image into smaller parts
QRect topLeftRect(0, 0, image.width() / 2, image.height() / 2);
QRect topRightRect(image.width() / 2, 0, image.width() / 2, image.height() / 2);

...

// Create a task group
tbb::task_group taskGroup;

// Run tasks for each image part
QImage topLeftImage = image.copy(topLeftRect);
QImage topRightImage = image.copy(topRightRect);

...

taskGroup.run([&] { applyFilterParallel(bottomLeftImage); });
taskGroup.run([&] { applyFilterParallel(bottomRightImage); });

// Wait for all tasks to complete
taskGroup.wait();
```

Сви задаци се додељују *taskGroup*-у, и на крају извршавања решења се синхронизују, и прослеђују као резултат примене *Gauss Blur* филтер-а.

Подела овог алгоритма на више мањих задатака показала је значајне напредке у времену извршавања, и то је управо и разлог одлуке за паралелизацију ове функције.

4. ТЕСТИРАЊА

За тестирање користио се лаптоп рачунар са следећим спецификацијама:

Device name DESKTOP-S6SUB98
Processor Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
Installed RAM8.00 GB (7.82 GB usable)
System type 64-bit operating system, x64-based processor

Приликом тестирања коришћене су слике различитих димензија, и величина. На овај начин смо прошли кроз велики број различитих случајева, и тестирали наш програм како на минималне проблеме тако и на оне веома велике.

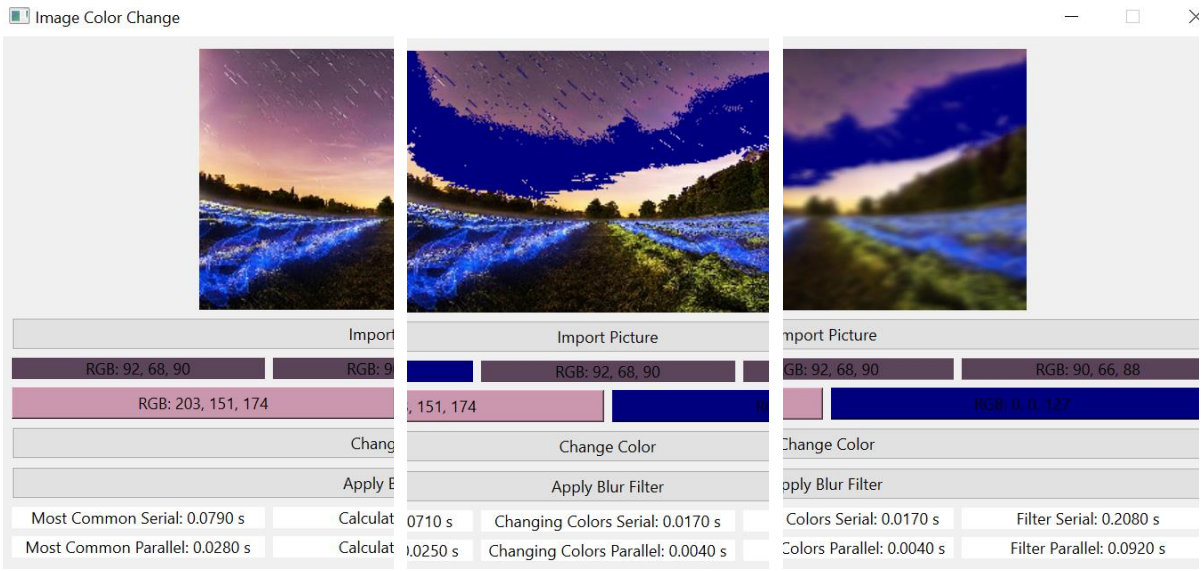
Следе неки примери рада рачунара у тест окружењу.

Тест 1:
Узорак:

- Слика – димензије 5184x3456 пиксела, величина 51.2 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.079 секунди
  - Паралелизација – 0.028 секунди
- Промена боје на слици
  - Серијализација – 0.017 секунди
  - Паралелизација – 0.004 секунди
- Примена филтера
  - Серијализација – 0.208 секунди
  - Паралелизација – 0.092 секунди



Слика 2 - Тест 1

Тест 2:
Узорак:

- Слика – димензије 1920x1080 пиксела, величина 1.12 КБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.018 секунди
  - Паралелизација – 0.006 секунди
- Промена боје на слици
  - Серијализација – 0.018 секунди
  - Паралелизација – 0.004 секунди
- Примена филтера
  - Серијализација – 0.189 секунди
  - Паралелизација – 0.074 секунди

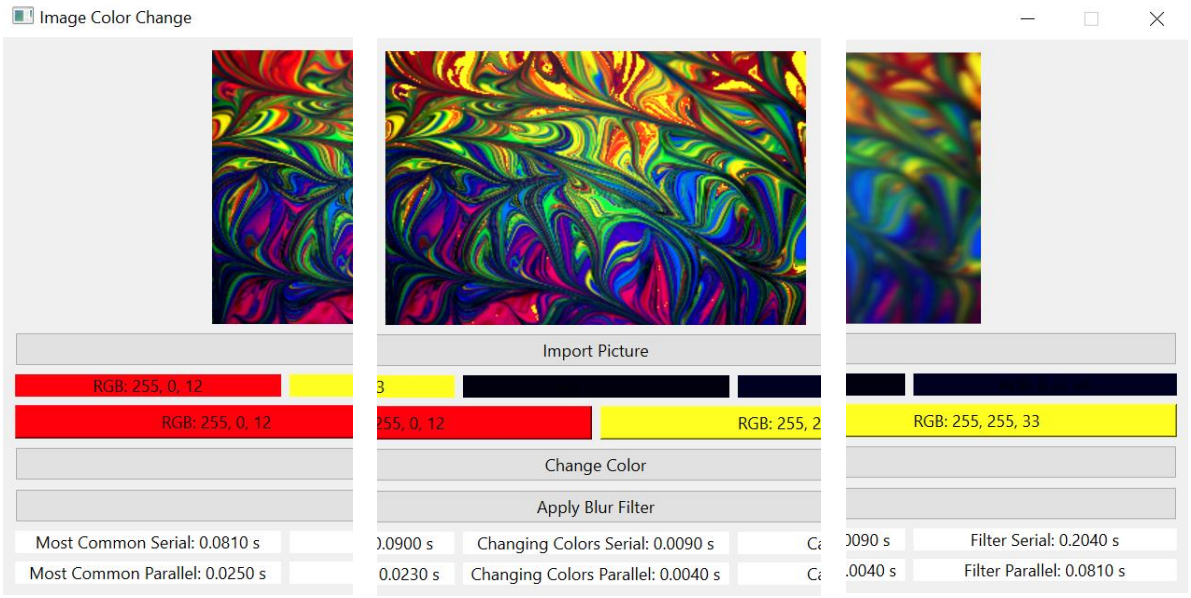
Слика 3 - Тест 2

Тест 3:
Узорак:

- Слика – димензије 5616x3744 пиксела, величина 2.94 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.081 секунди
  - Паралелизација – 0.025 секунди
- Промена боје на слици
  - Серијализација – 0.009 секунди
  - Паралелизација – 0.004 секунди
- Примена филтера
  - Серијализација – 0.204 секунди
  - Паралелизација – 0.081 секунди



Слика 4 - Тест 3

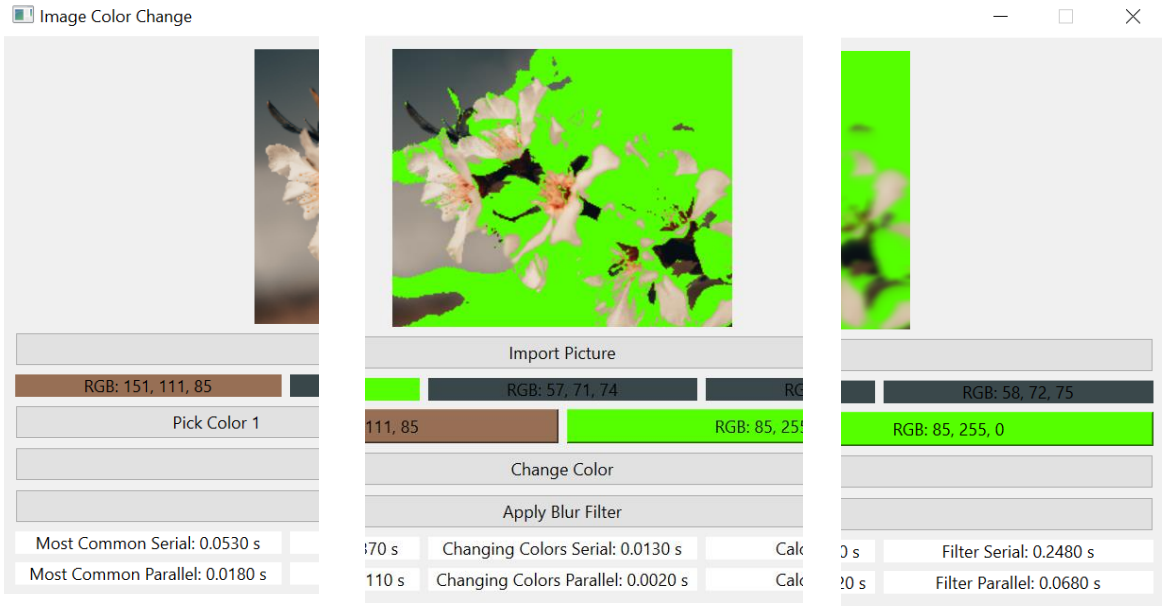
Тест 4:

Узорак:

- Слика – димензије 2160x2700 пиксела, величина 385 КБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.053 секунди
  - Паралелизација – 0.018 секунди
- Промена боје на слици
  - Серијализација – 0.013 секунди
  - Паралелизација – 0.002 секунди
- Примена филтера
  - Серијализација – 0.248 секунди
  - Паралелизација – 0.068 секунди



Слика 5 - Тест 4

Тест 5:
Узорак:

- Слика – димензије 1920x1080 пиксела, величина 3.26 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.046 секунди
  - Паралелизација – 0.015 секунди
- Промена боје на слици
  - Серијализација – 0.005 секунди
  - Паралелизација – 0.002 секунди
- Примена филтера
  - Серијализација – 0.181 секунди
  - Паралелизација – 0.081 секунди

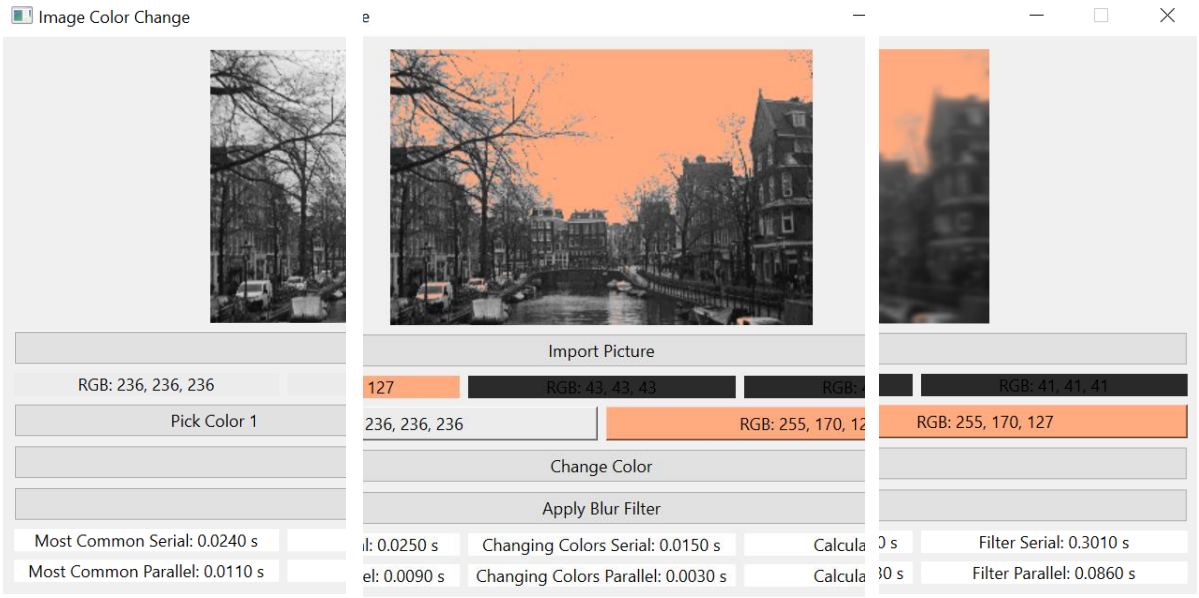
Слика 6 - Тест 5

Тест 6:
Узорак:

- Слика – димензије 3637x3637 пиксела, величина 2.09 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.024 секунди
  - Паралелизација – 0.011 секунди
- Промена боје на слици
  - Серијализација – 0.015 секунди
  - Паралелизација – 0.003 секунди
- Примена филтера
  - Серијализација – 0.301 секунди
  - Паралелизација – 0.086 секунди



Слика 7 - Тест 6

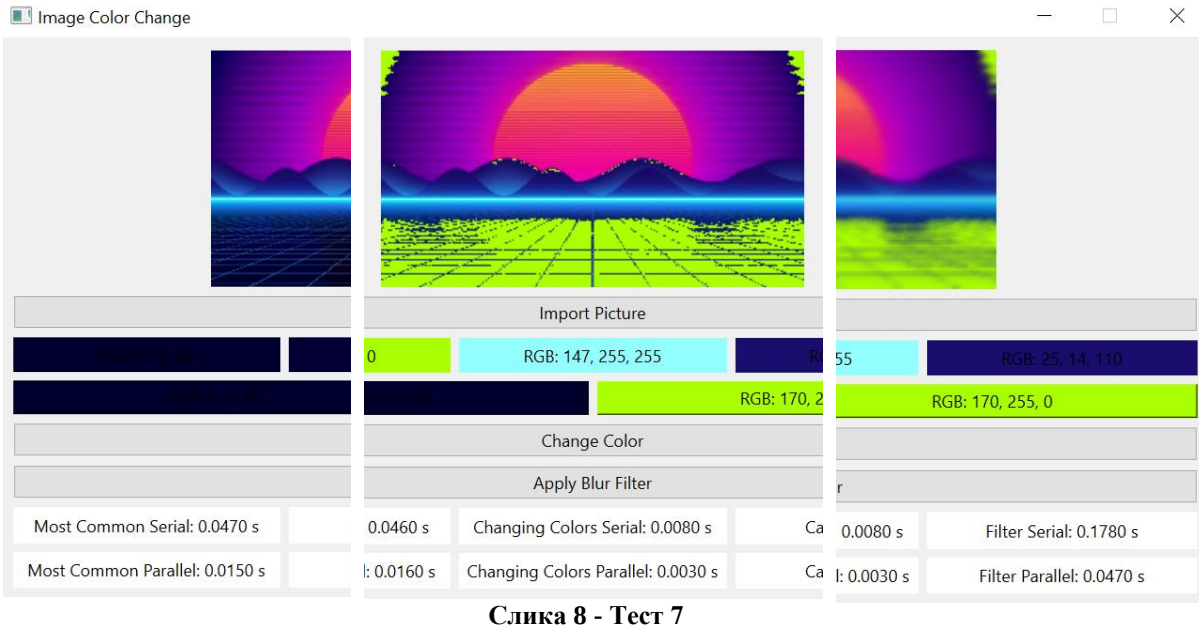
Тест 7:

Узорак:

- Слика – димензије 590x332 пиксела, величина 29.2 КБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.047 секунди
  - Паралелизација – 0.015 секунди
- Промена боје на слици
  - Серијализација – 0.008 секунди
  - Паралелизација – 0.003 секунди
- Примена филтера
  - Серијализација – 0.178 секунди
  - Паралелизација – 0.047 секунди



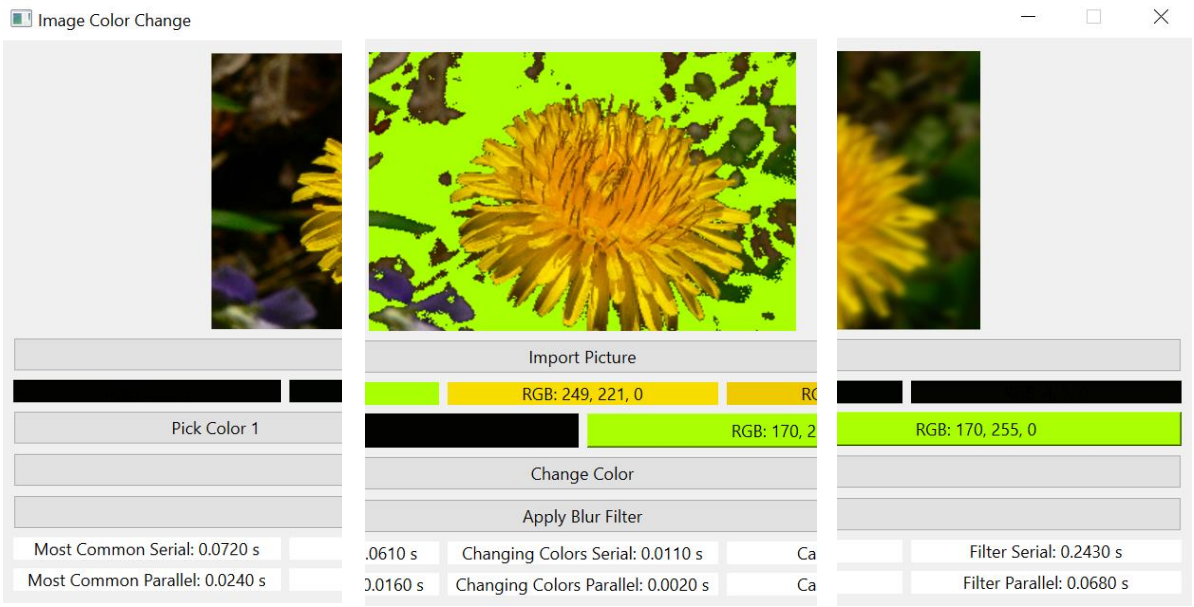
Слика 8 - Тест 7

Тест 8:
Узорак:

- Слика – димензије 1765x1413 пиксела, величина 7.13 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.072 секунди
  - Паралелизација – 0.024 секунди
- Промена боје на слици
  - Серијализација – 0.011 секунди
  - Паралелизација – 0.002 секунди
- Примена филтера
  - Серијализација – 0.243 секунди
  - Паралелизација – 0.068 секунди



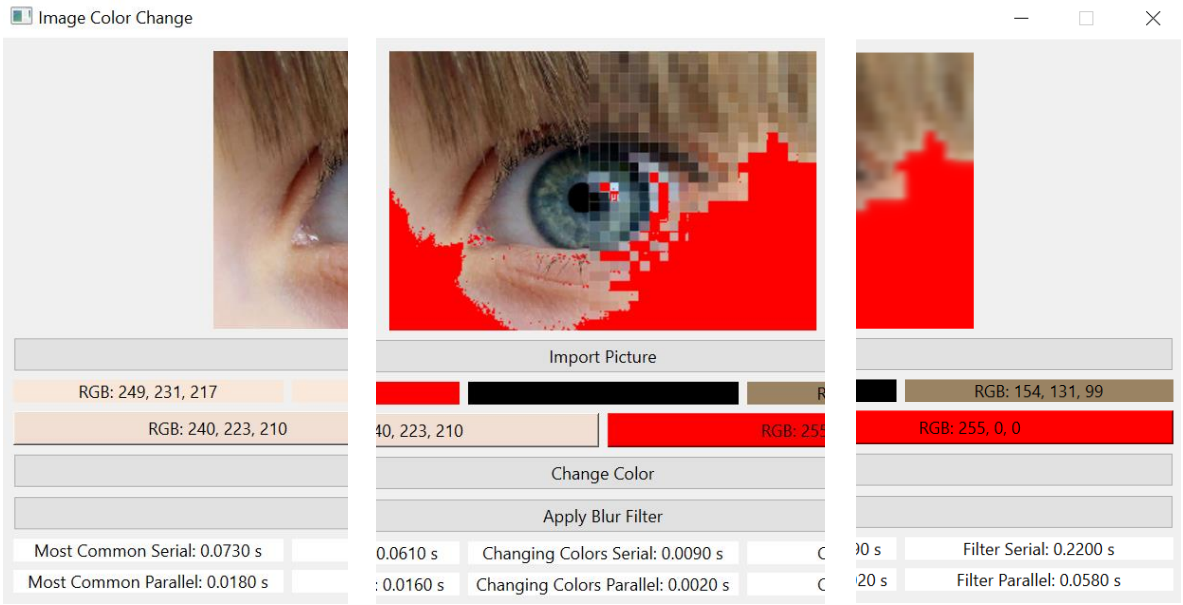
Слика 9 - Тест 8

Тест 9:
Узорак:

- Слика – димензије 669x468 пиксела, величина 49.5 КБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.073 секунди
  - Паралелизација – 0.018 секунди
- Промена боје на слици
  - Серијализација – 0.009 секунди
  - Паралелизација – 0.002 секунди
- Примена филтера
  - Серијализација – 0.220 секунди
  - Паралелизација – 0.058 секунди



Слика 10 - Тест 9

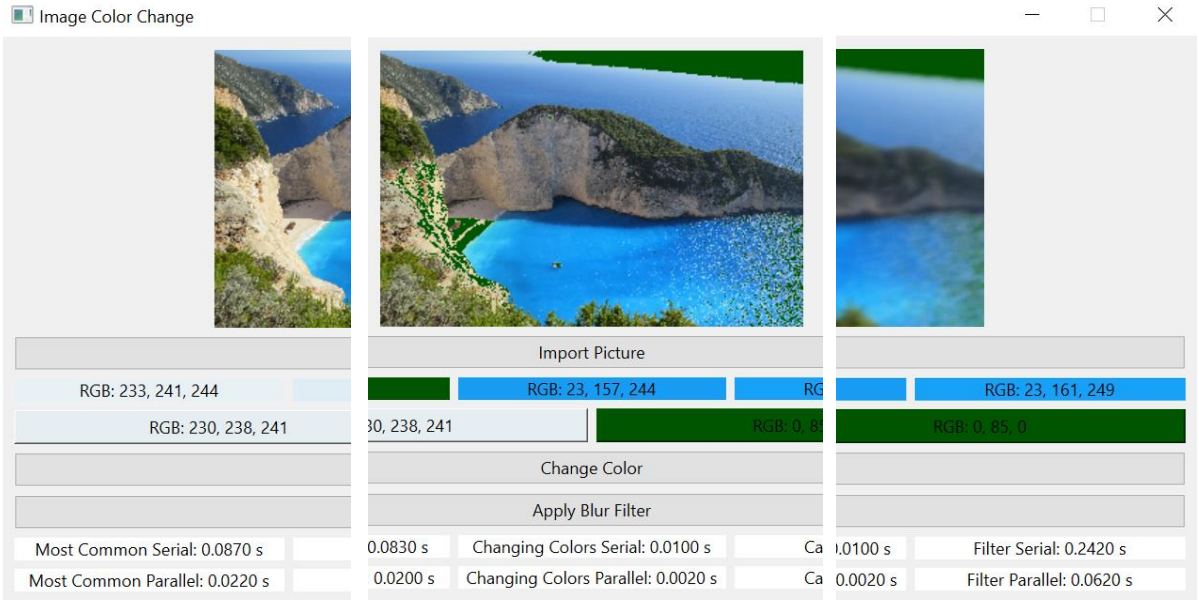
Тест 10:

Узорак:

- Слика – димензије 4000x3000 пиксела, величина 2.23 МБ

Резултати:

- Проналажење 3 најчешће боје:
  - Серијализација – 0.087 секунди
  - Паралелизација – 0.022 секунди
- Промена боје на слици
  - Серијализација – 0.010 секунди
  - Паралелизација – 0.002 секунди
- Примена филтера
  - Серијализација – 0.242 секунди
  - Паралелизација – 0.062 секунди

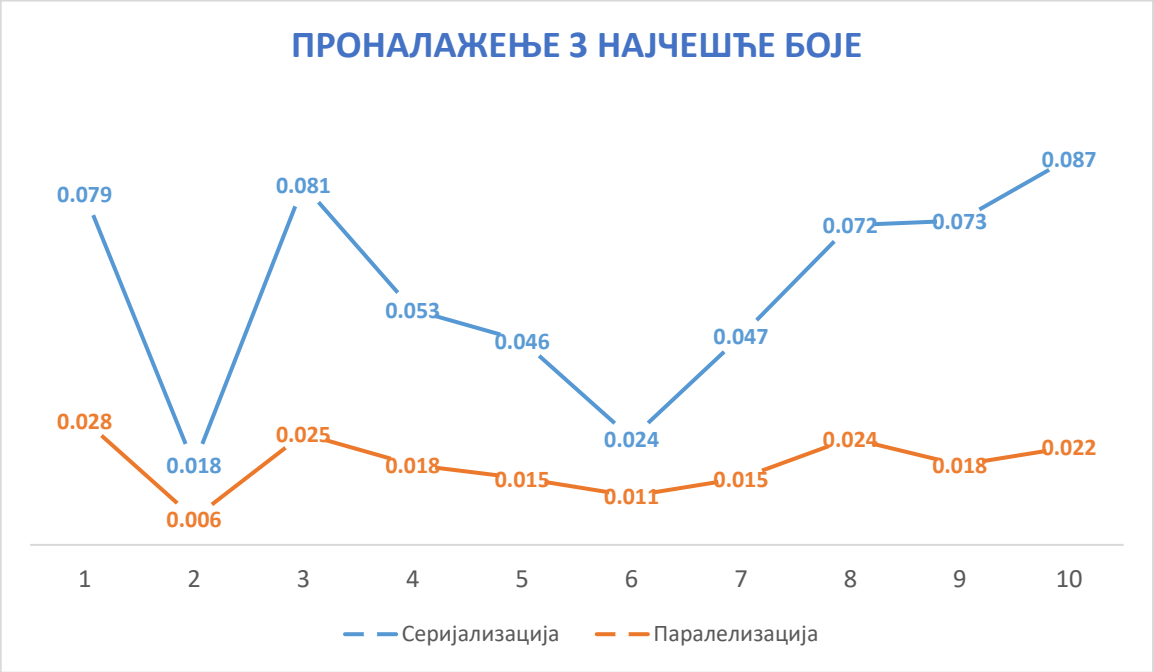


Слика 11 - Тест 10

4.1. Анализа Резултата за Функцију Проналажење 3 Најчешће Боје

Тест Број	Димензије Узорка	Величина Узорка	Серијализација	Паралелизација	Разлика
1	5184x3456	51.2 МБ	0.079	0.028	0.051
2	**1920x1080**	**1.12 КБ**	**0.018**	**0.006**	**0.012**
3	5616x3744	2.94 МБ	0.081	0.025	0.056
4	2160x2700	385 КБ	0.053	0.018	0.035
5	1920x1080	3.26 МБ	0.046	0.015	0.031
6	3637x3637	2.09 МБ	0.024	0.011	0.013
7	590x332	29.2 КБ	0.047	0.015	0.032
8	1765x1413	7.13 МБ	0.072	0.024	0.048
9	669x468	49.5 КБ	0.073	0.018	0.055
10	**4000x3000**	**2.23 МБ**	**0.087**	**0.022**	**0.065**

Табела 1 - Проналажење 3 најчешће боје
(времена су изражена у секундама)



Приликом тестирања функције за проналажење 3 најчешће боје можемо да приметимо да паралелизована функција у свим случајевима има боље време од серијализоване.

Најмању разлику можемо да приметимо у другом примеру, и то свега 0.012 секунди. Други пример је слика која садржи искључиво једну боју, па овакав резултат и није изненађујући.

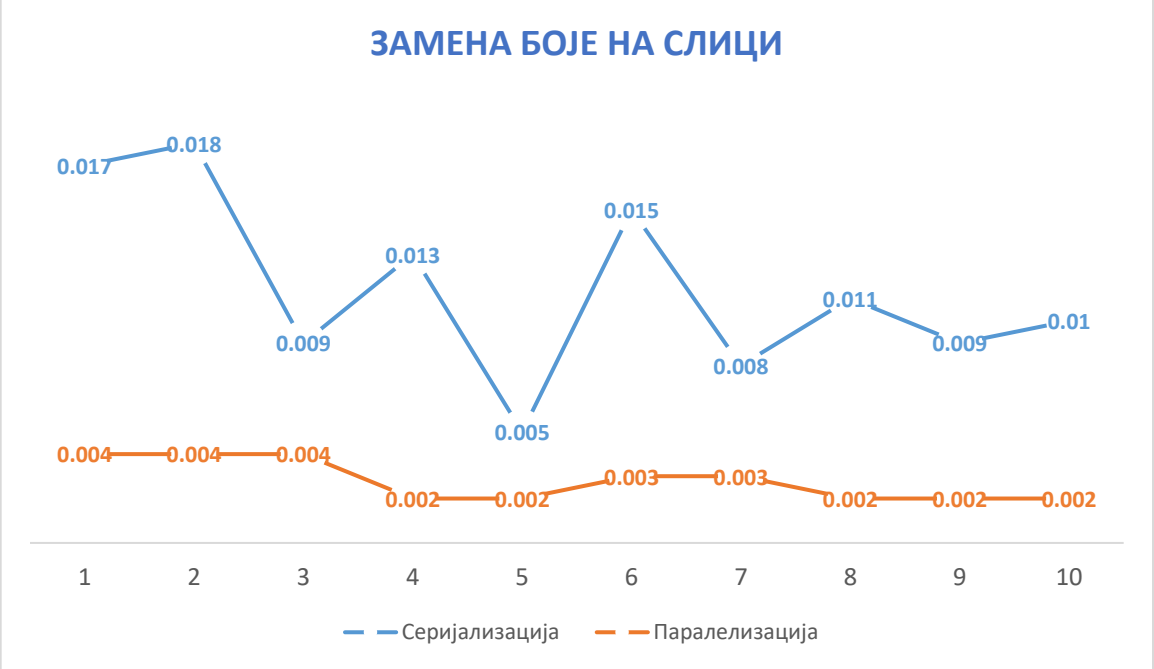
Са друге стране макимална разлика у временима се види у десетом примеру, и то 0.065 секунди, што је уједно побољшање за око 4 пута.

Још нека од великих разлика у времену можемо уочити у примерима 1, 3 и 9.

4.2. Анализа Резултата за Функцију Замена Боје на Слици

Тест Број	Димензије Узорка	Величина Узорка	Серијализација	Паралелизација	Разлика
1	5184x3456	51.2 МБ	0.017	0.004	0.013
2	**1920x1080**	**1.12 КБ**	**0.018**	**0.004**	**0.014**
3	5616x3744	2.94 МБ	0.009	0.004	0.005
4	2160x2700	385 КБ	0.013	0.002	0.011
5	**1920x1080**	**3.26 МБ**	**0.005**	**0.002**	**0.003**
6	3637x3637	2.09 МБ	0.015	0.003	0.012
7	590x332	29.2 КБ	0.008	0.003	0.005
8	1765x1413	7.13 МБ	0.011	0.002	0.009
9	669x468	49.5 КБ	0.009	0.002	0.007
10	4000x3000	2.23 МБ	0.010	0.002	0.008

Табела 2 - Замена Боје на Слици
(времена су изражена у секундама)



Приликом тестирања функције за замену боје на слици можемо да приметимо да паралелизована функција у свим случајевима има боље време од серијализоване.

Најмању разлику можемо да приметимо у петом примеру, и то свега 0.003 секунди.

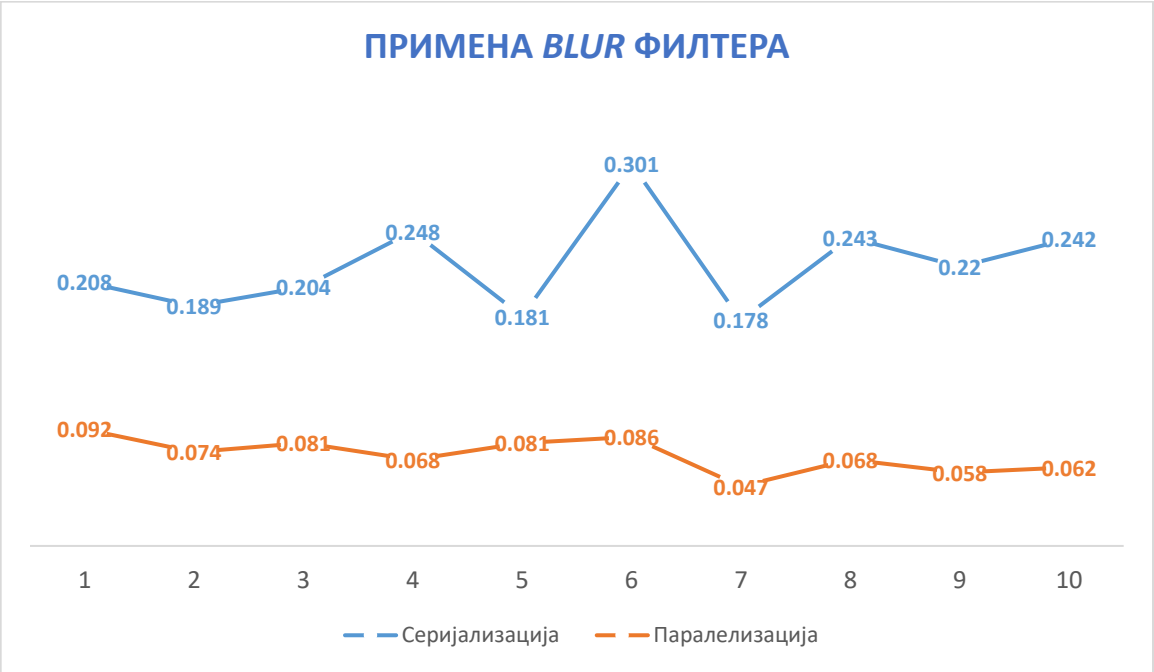
Са друге стране макимална разлика у временима се види у другом примеру, и то 0.014 секунди, што је уједно побољшање за око 4,5 пута.

Још нека од великих разлика у времену можемо уочити у примерима 1, 4 и 6.

4.3. Анализа Резултата за Функцију Примена blur Филтера

Тест Број	Димензије Узорка	Величина Узорка	Серијализација	Паралелизација	Разлика
1	5184x3456	51.2 МБ	0.208	0.092	0.116
2	1920x1080	1.12 КБ	0.189	0.074	0.115
3	5616x3744	2.94 МБ	0.204	0.081	0.123
4	2160x2700	385 КБ	0.248	0.068	0.18
5	**1920x1080**	**3.26 МБ**	**0.181**	**0.081**	**0.1**
6	**3637x3637**	**2.09 МБ**	**0.301**	**0.086**	**0.215**
7	590x332	29.2 КБ	0.178	0.047	0.131
8	1765x1413	7.13 МБ	0.243	0.068	0.175
9	669x468	49.5 КБ	0.220	0.058	0.162
10	4000x3000	2.23 МБ	0.242	0.062	0.18

Табела 3 - Примена *blur* Филтера
(времена су изражена у секундама)



Приликом тестирања функције за примену *blur* филтера можемо да приметимо да паралелизована функција у свим случајевима има боље време од серијализоване.

Најмању разлику можемо да приметимо у петом примеру, и то свега 0.1 секунди. Ипак, иако је ово најмања разлика, уколико упоредимо резултате серијске и паралелне функције, видећемо да је време побољшано и то чак за нешто више од 2 пута.

Са друге стране макимална разлика у временима се види у шестом примеру, и то 0.215 секунди, што је уједно побољшање за нешто мање од 4 пута.

Још нека од великих разлика у времену можемо уочити у примерима 8 и 10.

5. ЗАКЉУЧАК

Имали смо три главна задатка, а то су били:

- Проналажење 3 најчешће појављиване боје на слици
- Замена боје на слици
- Примена *blur* филтера

У примеру за проналажење 3 најчешће појављиване боје на слици могли смо да приметимо да је заједничко за све узорке јесте велика димензија слика, тако да можемо да закључимо да је за паралелизацију ове функције то био један од битних фактора приликом узимања узорака.

Резултат који смо добили приликом анализе паралелизације функције за замену боје на слици јесте изненађујући. Слика са само једном бојом, изузетно мале величине је најбољи узорак и показатељ ове паралелизације. Слике које су такође имале велику разлику у временима приликом покретања серијске и паралелне функције су 1, 4 и 6. Свака од њих је слика која је велике димензије, али не нужно и велике или мале величине. Ова чињеница нам указује на то како приликом паралелизације функције за замену боја на слици, сама величина није један од пресудних фактора за избор доброг узорка.

Из анализе резултата које смо добили за паралелизацију функције за примену *blur* филтера видимо да је најбољи резултат имао узорак који је био димензија квадрата, тј 3637x3637 пиксела. Узорци који су се такође издвојили су 8 и 10, ако погледамо димензије ових узорака можемо да приметимо како су њихове димензије веома близу квадратном облику, и можемо да закључимо како је то битан фактор који је утицао на резултате приликом паралелизације ове функције. Узорак који је имао најлошији резултат је уједно и најдаљи од облика квадрата, па и то иде у корист нашем закључку.

Након паралелизација и анализирања резултата у овим функцијама можемо да закључимо да напредак у временима засигурно постоји. Напредак је највидљивији у функцији која примењује филтер где је просечно време побољшања након паралелизације функције: 0.15 секунди. На другом месту се налази функција за проналажење 3 најчешће појављене боје на слици, са просечним временом побољшања: 0.04 секунде. Најлошији резултати након паралелизације се виде у функцији за мењање боје, где је просечно време побољшања: 0.009 секунди.

Резултати које смо добили иду у прилог томе што смо успели да пронађемо шаблон за слике које имају добро или лоше време побољшавања у функцијама за примењивање филтера и проналажење најчешћих боја, док у функцији за измену боја, шаблон није видљив.