Version 7/10/18

Computer Architecture

Lab 2

Fernando.Rincón@uclm.es
Serafin.Benito@uclm.es

Contents

- The Qt framework
- Convert to gray example
- Histogram

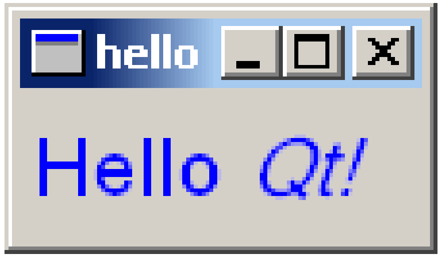
Qt

- Developed by Trolltech (www.trolltech.com)
- multiplatform C++ GUI application framework
- Fully object-oriented and component-based
- Basis of the KDE linux Desktop environment

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
  QApplication app( argc, argv );
  QLabel hello( "<font color=blue>Hello <i>Qt!</i>“,
                "</font>", 0 );

  app.setMainWidget( &hello );
  hello.show();
  return app.exec();
}
```



Hello *Qt!*

Qt

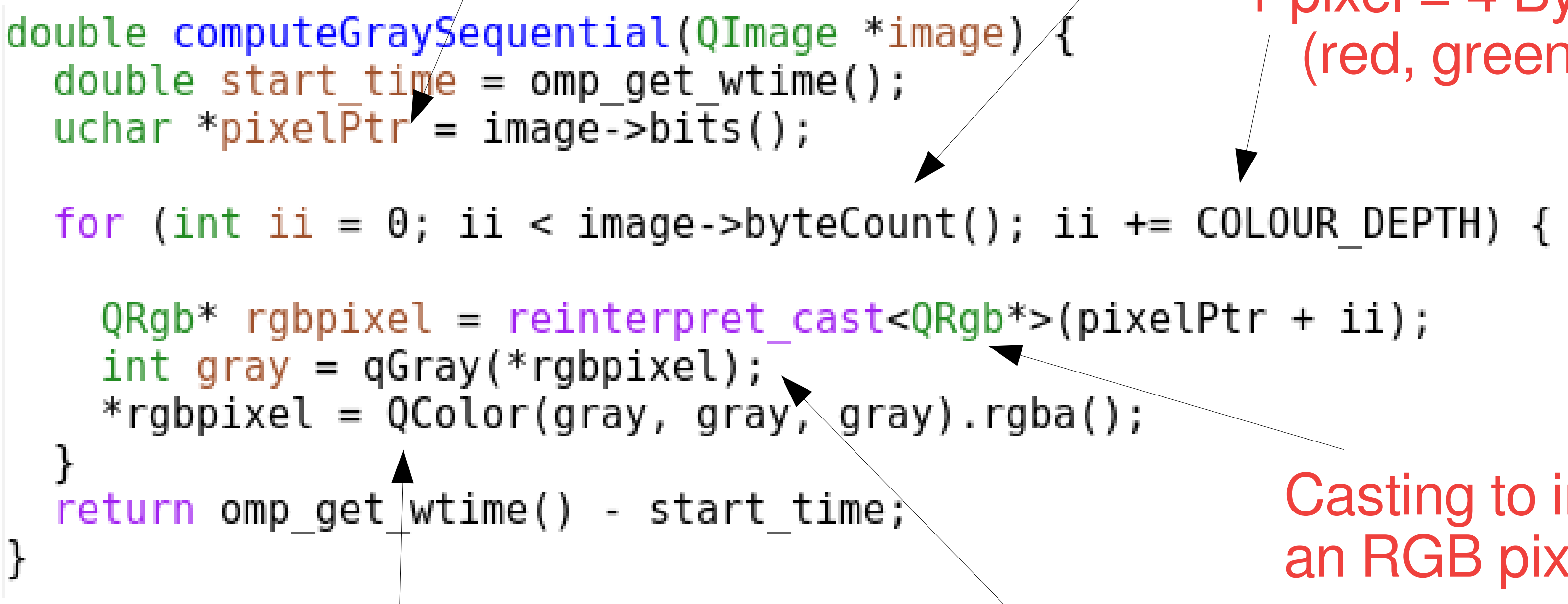
- Development procedure
  - `mydir$ qmake -project`
    - builds the project file **mydir.pro** that will be the source to create the Makefile
    - You can include flags, libraries, …
      - To use OpenMP you should add with an editor the following two lines to the **.pro** file
        - `QMAKE_CXXFLAGS = -fopenmp`
        - `LIBS += -fopenmp`
  - `$ qmake` // builds the Makefile
  - `$ make` // the executable
  - `$ ./myapp` // runs the executable

Color to Gray example

Pointer to image raw data

Total number of bytes in the image

1 pixel = 4 Bytes
(red, green, blue, alpha)



```
double computeGraySequential(QImage *image) {
  double start_time = omp_get_wtime();
  uchar *pixelPtr = image->bits();

  for (int ii = 0; ii < image->byteCount(); ii += COLOUR_DEPTH) {

    QRgb* rgbpixel = reinterpret_cast<QRgb*>(pixelPtr + ii);
    int gray = qGray(*rgbpixel);
    *rgbpixel = QColor(gray, gray, gray).rgba();
  }
  return omp_get_wtime() - start_time;
}
```

Casting to interpret data as an RGB pixel

Generate a gray level from the pixel

Create a new pixel with all components as grey

Color to Gray example

- Function `computeGrayParallel` has been obtained from the sequential one just parallelizing the for loop
- Function `computeGrayScanline` is sequential, but uses two nested loops for the computation: one for rows, the other for columns
- The main program:
  - Reads the image in the Qt way
  - Computes both the parallel and sequential versions
  - And shows the time spent in all case
- Use it as a guide

Color to Gray example

- First compile & test the code:
  - `cd gray`
  - `qmake -project`: to generate the project file (.pro)
  - Edit the resulting `gray.pro` file and add the following two lines:
    - `QMAKE_CXXFLAGS = -fopenmp`
    - `LIBS += -fopenmp`
  - `qmake`: to generate the final Makefile
  - `make`
  - `./gray`

Task 1: program `graya.cpp`

- Add a new function `computeGrayScanlineParallel` which should be the parallel version of `computeGrayScanline`:
  - Parallelize the first loop
  - Choose which variables must be private and which shared
  - Protect the following instruction with a critical section:
    - `scan = image->scanLine(ii);`
- Add the corresponding function call to computeGrayScanlineParallel to the main program.
  - Also measure the time elapsed and test the result is the same in all case

Task 2: Histogram

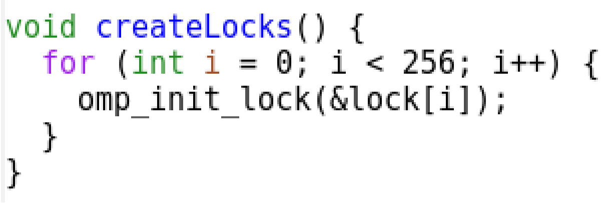
- Purpose of the Lab:
  - Compute the histogram of the image
- The histogram:
  - `int histogram[256];`
    - Since we have 256 levels of gray
  - All values are set to ‘0’ (`memset` function is recommended for such purpose)
  - For each pixel, get the gray value and increment the corresponding entry in the array.
    - Once computed, for each value of `level` (`0≤level<256`), **`histgr[level]`** must contain **the number of pixels of the image whose gray level is `level`**

Histogram

- Versions to implement:
  - Sequential
  - Parallel using the following alternatives for the histogram modification (which must be protected since it's shared between all threads)
    - Critical section
    - Atomic
    - Using low level locks
    - With reduction over the histogram variable
    - Manually paralellized
- All versions inside the same program in a file called `histogram.cpp`

Histogram

- Locks:
  - Low level synchronization mechanism
  - Usage:
    - First initialize the locks



```
void createLocks() {
  for (int i = 0; i < 256; i++) {
    omp_init_lock(&lock[i]);
  }
}
```

    - Before accessing the variable get the lock
      - `omp_set_lock(&lock[gray]);`
    - Then access the variable
    - An unset it afterwards
      - `omp_unset_lock(&lock[gray]);`

Histogram

- Manual parallelization
  - Divide the single loop into a nested one, with the outer performing as many passes as N cores in your PC, each over a Nth part of the array
  - Use a **parallel for** to compute the partial histograms
  - Use a **sequential for** to combine all histograms into the final one

Histogram

- All versions in a single source code file
- Should include the timing and the code to test the validity of the solution
- Include your conclusions about the results as comments in the source code, specially if they where unexpected
- Upload the source code before next lab to Campus Virtual
  - Include a first line comment with your name
- You can work in pairs
  - The only one student should upload the code
  - Remember to put both names in the first line comment
- **Tasks to upload:**
  - File graya.cpp: at the end of the lab session
  - File histogram.cpp: before the next lab session

And remember that **plagiarize is a really bad idea!**

No matter if you are a Minister of the Goverment

All your uploads will be compared