FINAL PROJECT

Design

Juan José Betancourt
Nicolás Delgado
Camila Paladines

Parallel Programming

Professor: Roger Alfonso Gómez Nieto

Jun 2, 2021

Contents

1 The Code to Parallelize

1.1 Algorithm Code

In the following subsections you can see the main functions of the application to be worked on.

1.1.1 Make Segments

```
vector<QrSegment> QrSegment::makeSegments(const char *text) {
  // Select the most efficient segment encoding automatically
  vector<QrSegment> result;
  if (*text == '\0');  // Leave result empty
  else if (isNumeric(text))
    result.push_back(makeNumeric(text));
  else if (isAlphanumeric(text))
    result.push_back(makeAlphanumeric(text));
  else {
    vector<uint8_t> bytes;
    for (; *text != '\0'; text++)
      bytes.push_back(static_cast<uint8_t>(*text));
    result.push_back(makeBytes(bytes));
  }
  return result;
}
```

1.1.2 Encode Text

```
QrCode QrCode::encodeText(const char *text, Ecc ecl) {
  vector<QrSegment> segs = QrSegment::makeSegments(text);
  return encodeSegments(segs, ecl);
}


QrCode QrCode::encodeBinary(const vector<uint8_t> &data, Ecc ecl) {
  vector<QrSegment> segs{QrSegment::makeBytes(data)};
  return encodeSegments(segs, ecl);
}
```

The encode method that the before functions are calling is the following:

```
1
2  QrCode QrCode::encodeSegments(const vector<QrSegment> &segs, Ecc ecl,
3      int minVersion, int maxVersion, int mask, bool boostEcl) {
4    if (!(MIN_VERSION <= minVersion && minVersion <= maxVersion && maxVersion
       <= MAX_VERSION) || mask < -1 || mask > 7)
5      throw std::invalid_argument("Invalid value");
6
7    //Find the minimal version number to use
8    int version, dataUsedBits;
9    for (version = minVersion; ; version++) {
10     int dataCapacityBits = getNumDataCodewords(version, ecl) * 8;  // Number of data
         bits available
11     dataUsedBits = QrSegment::getTotalBits(segs, version);
12     if (dataUsedBits != -1 && dataUsedBits <= dataCapacityBits)
13       break;   // This version number is found to be suitable
14     if (version >= maxVersion) {  // All versions in the range could not fit the given
         data
15       std::ostringstream sb;
16       if (dataUsedBits == -1)
17         sb << "Segment too long";
18       else {
19         sb << "Data length = " << dataUsedBits << " bits, ";
20         sb << "Max capacity = " << dataCapacityBits << " bits";
21       }
22       throw data_too_long(sb.str());
23     }
24   }
25   if (dataUsedBits == -1)
26     throw std::logic_error("Assertion error");
27
28   //Increase the error correction level while the data still fits in the current version
       number
29   for (Ecc newEcl : vector<Ecc>{Ecc::MEDIUM, Ecc::QUARTILE, Ecc::HIGH}) {  //
       From low to high
30     if (boostEcl && dataUsedBits <= getNumDataCodewords(version, newEcl) * 8)
31       ecl = newEcl;
32   }
33
34   // Concatenate all segments to create the data bit string
35   BitBuffer bb;
36   for (const QrSegment &seg : segs) {
37     bb.appendBits(static_cast<uint32_t>(seg.getMode().getModeBits()), 4);
38     bb.appendBits(static_cast<uint32_t>(seg.getNumChars()), seg.getMode().
         numCharCountBits(version));
39     bb.insert(bb.end(), seg.getData().begin(), seg.getData().end());
40   }
41   if (bb.size() != static_cast<unsigned int>(dataUsedBits))
42     throw std::logic_error("Assertion error");
43
44   // Add terminator and pad up to a byte if applicable
45   size_t dataCapacityBits = static_cast<size_t>(getNumDataCodewords(version, ecl)) * 8;
46   if (bb.size() > dataCapacityBits)
47     throw std::logic_error("Assertion error");
48   bb.appendBits(0, std::min(4, static_cast<int>(dataCapacityBits - bb.size())));
49   bb.appendBits(0, (8 - static_cast<int>(bb.size() % 8)) % 8);
50   if (bb.size() % 8 != 0)
51     throw std::logic_error("Assertion error");
```

```
52
53    // Pad with alternating bytes until data capacity is reached
54    for (uint8_t padByte = 0xEC; bb.size() < dataCapacityBits; padByte ^= 0xEC ^ 0x11)
55      bb.appendBits(padByte, 8);
56
57    // Pack bits into bytes in big endian
58    vector<uint8_t> dataCodewords(bb.size() / 8);
59    for (size_t i = 0; i < bb.size(); i++)
60      dataCodewords[i >> 3] |= (bb.at(i) ? 1 : 0) << (7 - (i & 7));
61
62    // Create the QR Code object
63    return QrCode(version, ecl, dataCodewords, mask);
64  }
```

1.1.3 Create QR

```
QrCode::QrCode(int ver, Ecc ecl, const vector<uint8_t> &dataCodewords, int msk) :
    // Initialize fields and check arguments
    version(ver),
    errorCorrectionLevel(ecl) {
  if (ver < MIN_VERSION || ver > MAX_VERSION)
    throw std::domain_error("Version value out of range");
  if (msk < -1 || msk > 7)
    throw std::domain_error("Mask value out of range");
  size = ver * 4 + 17;
  size_t sz = static_cast<size_t>(size);
  modules = vector<vector<bool> >(sz, vector<bool>(sz));  // Initially all white
  isFunction = vector<vector<bool> >(sz, vector<bool>(sz));

  // Compute ECC, draw modules
  drawFunctionPatterns();
  const vector<uint8_t> allCodewords = addEccAndInterleave(dataCodewords);
  drawCodewords(allCodewords);

  // Do masking
  if (msk == -1) {  // Automatically choose best mask
    long minPenalty = LONG_MAX;
    for (int i = 0; i < 8; i++) {
      applyMask(i);
      drawFormatBits(i);
      long penalty = getPenaltyScore();
      if (penalty < minPenalty) {
        msk = i;
        minPenalty = penalty;
      }
      applyMask(i);  // Undoes the mask due to XOR
    }
  }
  if (msk < 0 || msk > 7)
    throw std::logic_error("Assertion error");
  this->mask = msk;
  applyMask(msk);  // Apply the final choice of mask
  drawFormatBits(msk);  // Overwrite old format bits

  isFunction.clear();
  isFunction.shrink_to_fit();
}
```

1.2 Algorithm

The project is a QR code generator, in different languages, such as: C, C++, Python, Rust, and JavaScript. It has different ways to generate the QR code, either by using numeric characters, or by creating in characters the code to create a SVG/XML (less in the C code). In the Parallel Programming project, the focus will be on the code made in C++, where a namespace is created, to work with the QR creator, where we could find different parts that can potentially be parallelizable.

Input:

- ***text:*** is the text string that will be encoded for the QR code. Its maximum length must be 2600 characters, since for larger values a Segmentation Fault error occurs.
- ***n:*** is the length of the `text` or the number of characters.
- ***errCorLvl:***

Output:

- ***svgString:*** is the text string describing the SVG image of the QR code.

Description:

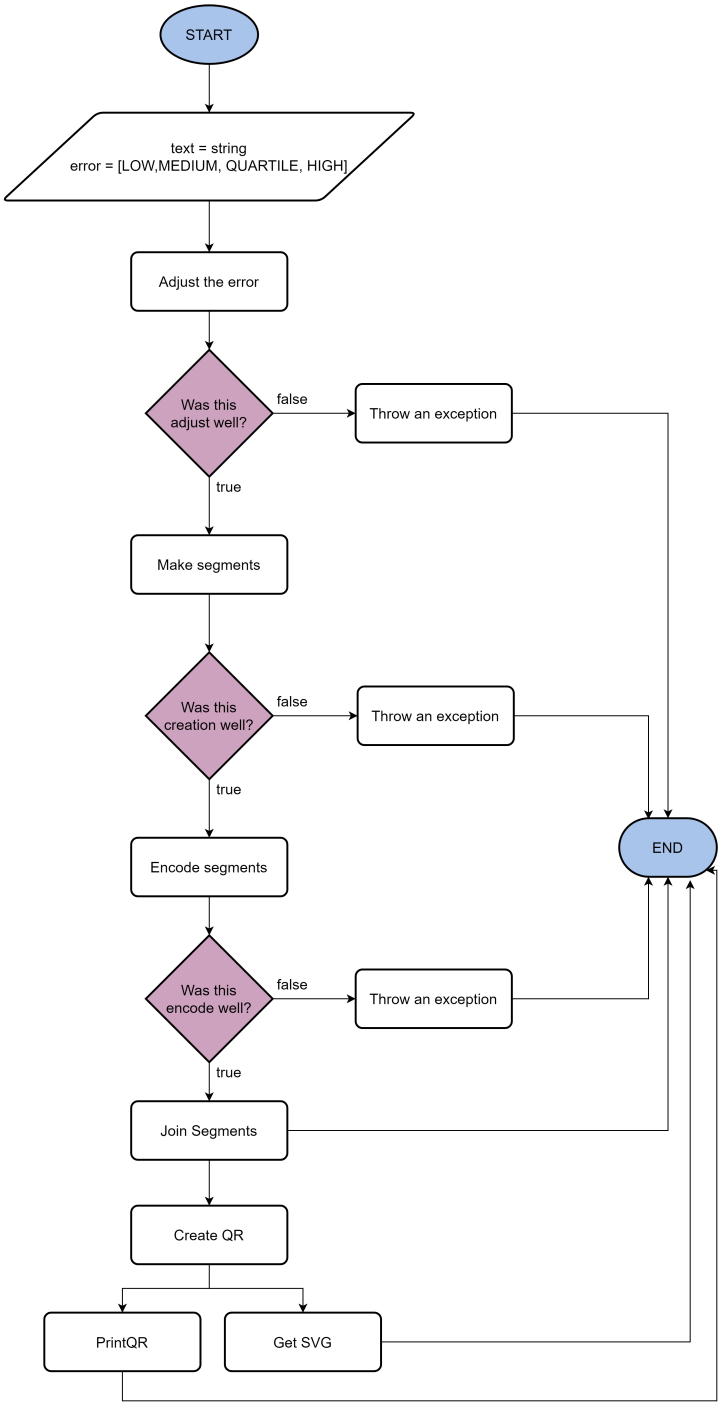
The algorithm basically what it does is, based on a size for the error (which can be corrected as it increases) and a text, to see if the text fits in the respective error, and adjusts it. After this, it goes to the encoding process where the segments are created according to the type of values that were entered according to whether it has alphanumeric or numeric values and the size of the text that was sent. Once it has been transformed, each segment is encoded, where it is checked if the size is adequate and each segment is joined together.

If there is no error when joining the segments it creates the QR code and after that you can use the method to print the QR code found in the demo or we can also create a file in SVG format with functional QR code.

This creation of the QR code in an SVG file is done as follows: it is sent to have 4 edges, the format of an SVG/XML file is copied and in the path we proceed to go transforming module by module to what goes in the path of the SVG files and so on until there are no more modules, this could be sent as an output to a single file with .svg format, to obtain the desired file or simply printed in console what goes inside the SVG file. There is also the option to print in console a representation of the QR code with # where black cells would go in the SVG.

Flowchart:

The operation of the algorithm can be better understood in the following flowchart:


START
text = string
error = [LOW,MEDIUM, QUARTILE, HIGH]
Adjust the error
Was this adjust well?
false
Throw an exception
true
Make segments
Was this creation well?
false
Throw an exception
true
Encode segments
Was this encode well?
false
Throw an exception
true
Join Segments
Create QR
PrintQR
Get SVG
END

1.3 Complexity

1.4 Time Analysis

The experimental environment where the tests were taken are as follows:

Configuration	Value
System Version	Ubuntu 18.04.5
Compiler	GCC 7.5.0
CPU	Intel Core i5-7200U 2.50GHz x4
GPU	Intel HD Graphics 620
Memory Capacity	11,5 GiB
OpenMP Version	OpenMP 4.5
MPI Version	MPICH 3.4.1

Configuration (CPU)	Value
Model Name	Intel(R) Core(TM) i5-7200U @ 2.50GHz
Architecture	x86_64
CPU(s)	4
Thread(s) per Core	2
Core(s) per Socket	2
Socket(s)	1
CPU Max MHz	3100,0000
CPU Min MHz	400,0000
BogoMIPS	5399.81
Virtualization	VT-x
L1d Cache	32K
L1i Cache	32K
L2 Cache	256K
L3 Cache	3072K

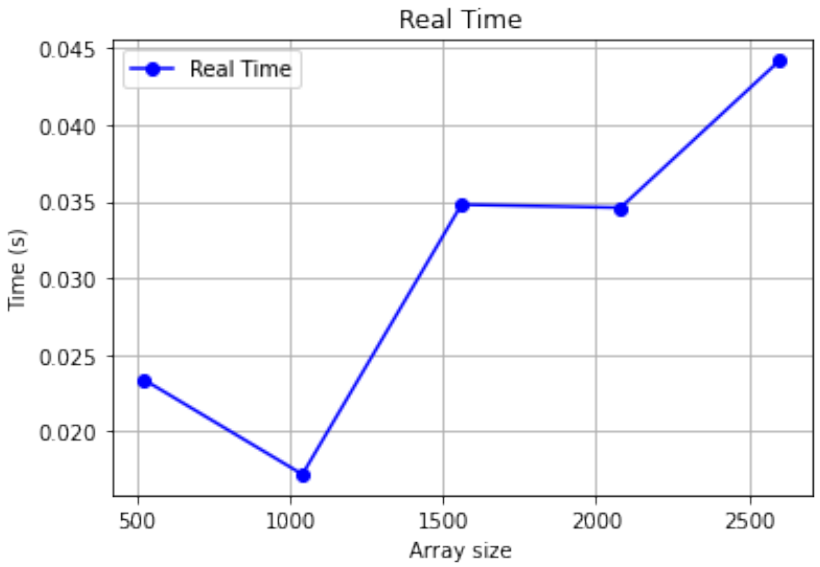
Configuration (RAM)	Value (RAM 1)	Value (RAM 2)
Total Width	64 bits	64 bits
Data Width	64 bits	64 bits
Size	4GB	8GB
Type	DDR4	DDR4
Type Detail	Sync. Unbuffered	Sync. Unbuffered
Speed	2133 MT/s	2133 MT/s
Manufacturer	Samsung	Kingston
Configured Clock Speed	2133 MT/s	2133 MT/s

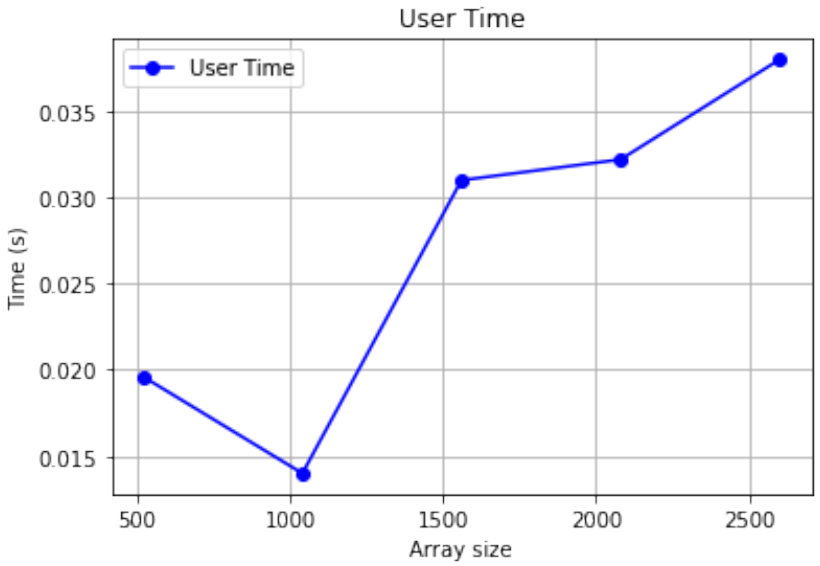
Configuration (Disk)	Value
Model Name	ST2000LM007
Capacity	2 TB
Width	32 bits
Speed	66MHz
Buffer size	128 MB
Average Seek Time	13 ms
Data Transfer Rate	600 MBps
Internal Data Rate	100 MBps

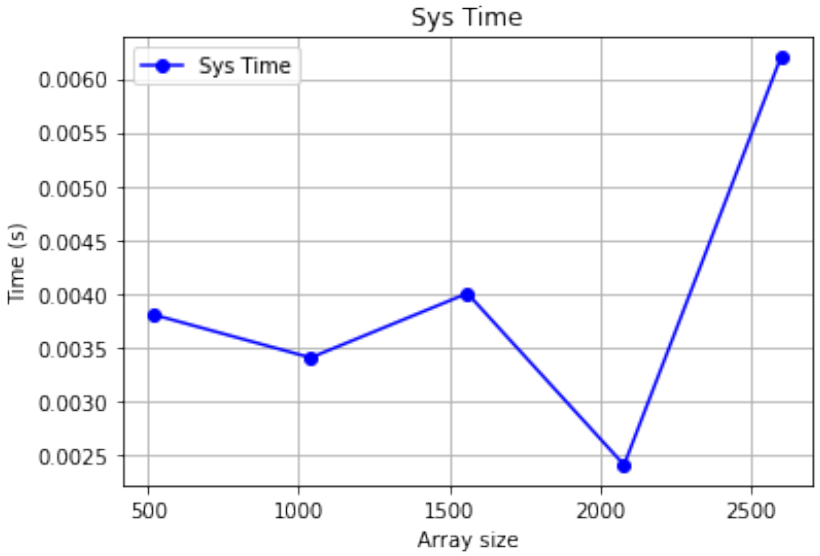
Measuring the execution time with the `time` tool for different values of `n`, the following result was obtained:

n	Real $\bar{e}$	Real σ_e	User $\bar{e}$	User σ_e	Sys $\bar{e}$	Sys σ_e
520	0.023	0.007	0.020	0.005	0.004	0.002
1040	0.017	0.009	0.014	0.007	0.003	0.003
1560	0.035	0.011	0.031	0.011	0.004	0.003
2080	0.035	0.010	0.032	0.009	0.002	0.002
2600	0.044	0.008	0.038	0.006	0.006	0.006

The time averages can best be seen in the following graphs:



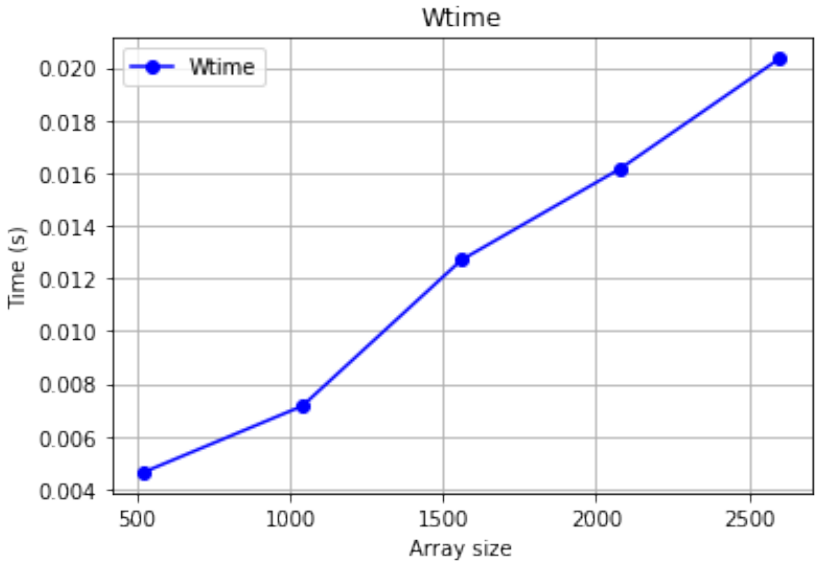

User Time
User Time
0.035
0.030
0.025
0.020
0.015
Time (s)
500
1000
1500
2000
2500
Array size


Sys Time
Sys Time
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
Time (s)
500
1000
1500
2000
2500
Array size

Using the `wtime` tool to measure the execution time of the `doBasicDemo()` function at the same input sizes, the following results are obtained:

n	**Wtime** $\bar{e}$	**Wtime** σ_e
520	0.00463	0.00040
1040	0.00715	0.00157
1560	0.01270	0.00126
2080	0.01617	0.00236
2600	0.02035	0.00156

The average times can best be seen in the following graph:



The application was analyzed using the tool `gprof`. Because the result was too long to fit in the document, only a part of it is shown, its full version can be seen in the file `gprof_analysis.txt` in the folder where this report is located.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 50.01      0.01     0.01    81990     0.00     0.00  qrcodegen::QrCode::reedSolomonMultipl
 50.01      0.02     0.01        8     1.25     1.25  qrcodegen::QrCode::getPenaltyScore()
  0.00      0.02     0.00  1337072     0.00     0.00  qrcodegen::QrCode::module(int, int) c
  0.00      0.02     0.00   237738     0.00     0.00  qrcodegen::QrCode::finderPenaltyAddHi
  0.00      0.02     0.00   120221     0.00     0.00  qrcodegen::QrCode::finderPenaltyCount
  0.00      0.02     0.00    28561     0.00     0.00  qrcodegen::QrCode::getModule(int, int
  0.00      0.02     0.00     5307     0.00     0.00  void std::vector<unsigned char, std::
  0.00      0.02     0.00     2704     0.00     0.00  qrcodegen::QrCode::finderPenaltyTermi
  0.00      0.02     0.00     2703     0.00     0.00  qrcodegen::BitBuffer::appendBits(unsi
  0.00      0.02     0.00     2026     0.00     0.00  qrcodegen::QrCode::setFunctionModule(
  0.00      0.02     0.00       46     0.00     0.00  qrcodegen::QrCode::drawAlignmentPatte
```

2 How to Parallelize the Algorithm

2.1 Proposal for Parallelization

2.2 Data Independence

3 References

- Course material, available on Blackboard
- `Nayuki`. 2020. Taken from this Github Repo: `cpp-QR-Code-generator`