RCPP INTRODUCTION

Tomas Martinovic

Rcpp

- It is a package originally develop by Dirk Eddelbuettel and Romain François
- It aims to ease the extension of R with C++ code.
- It allows to load C++ code in an interactive session.
- It has framework to help when creating package with Rcpp.

Possible ways to interact

- You can create function directly in R code as a string put into `cppFunction()` call.
  - In this case Rcpp will do most of the heavy lifting for You. (headers, compilation, linking)
- You can write C++ function and source it by calling `souceCpp()`.
  - Adding `verbose = TRUE` will show the whole process.
- Creating a package with C++ files with `// [[Rcpp::export]]` attribute.

Some other Rcpp features

- Rcpp syntactic sugar makes rewrite from R code easier, thanks to possibility to sometimes use R like notation.
- Rcpp attributes allows not just easy C++ function exports, but also define dependencies, change function names, define initialization functions.
- Rcpp contains function to handle exception in the C++ code and to check for user interruption.
- There is whole ecosystem of Rcpp packages that further extends its capabilities.

Resources

- A Brief Introduction to Rcpp
- Rcpp Attributes
- Writing a package that uses Rcpp
- Rcpp syntactic sugar

Process

To use the Rcpp package in a C++ file and export a C++ function to R, you need to follow these steps:

- Install the Rcpp package if it is not already installed. You can do this from within R by running `install.packages("Rcpp")`.
- Create a new `.cpp` file with your C++ code.
- Add `Rcpp::export` attribute to functions that should be available for use from R.

CUDA specifics

- For CUDA we need to add specific `.cu` file containing the code for the GPU. Such functions are called kernel functions.
- To be able to call this kernel function in our C++ function, we will need to create `.h` file called header file, which describes the function we want to call.
- Additionally, it is necessary to configure makevars file which tells R which compiler to use

Files in package

- R function that calls a C++ function.
- C++ file
- cu file
- makevars
- header file
- DESCRIPTION
- NAMESPACE

Example of Mandelbrot set computation

Mandelbrot set is defined as $f_c(z) = z^2 + c$, where c is a complex number corresponding to the point coordinates.

```
mandelbrot <- function(c, max_iter = 100) {
  z <- c
  for (i in 1:max_iter - 1) {
    z <- z ^ 2 + c
    if (abs(z) > 2) {
      return(i)
    }
  }
  return(max_iter)
}
```

For loop R approach

```r
dc <- cmax - cmin
x <- y <- 1:resolution - 1
x <- Re(cmin) + (x / resolution * Re(dc))
y <- Im(cmin) + (y / resolution * Im(dc))
points <-
  outer(x, y, function(x, y)
    complex(real = x, imaginary = y))
result_for <- matrix(NA,
                     dim(points)[1], dim(points)[2])
for (x in 1:dim(points)[1]) {
  for (y in 1:dim(points)[2]) {
    result_for[x, y] <- mandelbrot(points[x, y],
                                   max_iter)
  }
}
```

Converting mandelbrot function to C

```
int Mandel(double real, double im,
           int max_iter = 100)
{
  std::complex<double> c(real, im);
  std::complex<double> z = c;
    for (int i=0; i< max_iter; i++){
      z = z * z + c;
      if (std::abs(z) > 2) {
        return i;
      }
    }
 return max_iter;
}
```

```
...
  std::complex<double> dc = cmax - cmin;
  IntegerMatrix out( resolution );
  for (int i=0; i < resolution; i++){
    for(int j=0; j < resolution; j++){
      double helper = static_cast<double>(i);
      double helper2 = static_cast<double>(j);
      double fx = helper / resolution * real(dc);
      double fy = helper2 / resolution * imag(dc);
      std::complex<double> c(real(cmin) + fx,
                              imag(cmin) + fy);
  ...
    }
  }
...
```