Lab 1 Using R to visualize the runtime of Fibonacci algorithms

August 20, 2019

In this lab, we will evaluate and visualize empirical runtime of Fibonacci algorithms. Then we will examine if the runtime growth is consistent with theoretical analysis.

1 Setup the environment

We will setup the R environment to run C++ programs conveniently to take advantage of the powerful plotting functions in R.

1.1 Install R, Rstudio, and R package Rcpp

All software packages are free to download. You can skip the following steps if you use CS Linux computers, as they are already installed on CS computers.

For your home computers, you need to install them:

1. Install R from www.r-project.org
2. Install Rstudio from www.rstudio.com
3. Install package Rcpp within R

1.2 Configuration of the search path to R

Configure the search path to include the R directory on CS Linux computers. Insert the following R configuration line to the end of the `.cshrc` file in your home directory:

```
source /home/unsupported/linux64/config/cshrc.R
```

2 Evaluate and visualize the runtime of a C++ program in R

The purpose of running C++ program within R is to utilize the powerful visualization and timing functions in R to conduct quick performance evaluation on the C++ program with minimal amount of programming.

The following are required steps to integrate R part into the C++ program and run it in R or Rstudio. You can do all the steps within the Rstudio integrated development environment:

1. Develop your C++ program (”`fib.cpp`”) as normal
2. Test and debug your C++ code
3. Insert a comment line right before the function you want to run in R:

   ```
   // [[Rcpp::export]]
   int fib1(int n)
   {
     ...
   }
   ```

4. Insert R code to measure the runtime of your C++ code
   - Use R function `system.time()` to accurately count time charged to your program, not other concurrent processes on the same computer.
5. Insert R code to visualize the result of your C++ code
   - Use R function `plot()` to visualize the result as a curve.
6. Invoke R or Rstudio
7. If your code uses C++11 features, type in the following command in R:

   ```
   R command to enable c++11 features
   > Sys.setenv(”PKG_CXXFLAGS”=”-std=c++11”)
   ```

If you copy and paste from the above line, the double quote may have to be re-entered as this document code them differently from R.

8. Run the R code portion within your C++ code using

R command to run the R portion of fib.cpp

```
> Rcpp::sourceCpp(”fib.cpp”)
```

or equivalently, you can push the “source” button within Rstudio without typing in the command.

3 Example: computing Fibonacci numbers

See example `fib.cpp`

fib.cpp

```
#include <iostream>
using namespace std;

// [[Rcpp::export]]
int fib1(int n)
{
  if(n == 0) return 0;
  if(n == 1) return 1;

  return fib1(n-1) + fib1(n-2);
}

// [[Rcpp::export]]
bool test_fib1()
{
  bool passed = true;
  int ns[] = {0, 1, 2, 3, 4, 5, 6};
  int Fs[] = {0, 1, 1, 2, 3, 5, 8};
  for(int i=0; i<7; i++) {
    if(fib1(ns[i]) != Fs[i]) {
      passed = false;
      break;
    }
  }
  if(passed) {
    cout << ”test_fib1() passed. Congratulations!” << endl;
  } else {
    cout << ”test_fib1() failed!” << endl;
  }
  return passed;
}

int main()
{
  test_fib1();
  return 0;
}

// Above is regular C++ portion
// ---------------------------------------------------------
// Below is the R portion to visualize the performance

// You can include R code blocks in C++ files processed
// with sourceCpp (useful for testing and development).
// The R code will be automatically run after C/C++
// compilation.

/*** R

k <- 10
ns <- 34+(1:k)
runtime <- vector(length=k)

for(i in 1:k) {
  n <- ns[i]
  runtime[i] <- system.time(fib1(n))[”user.self”]
}

plot(ns, runtime, type=”b”, xlab=”n”,
     ylab=”runtime (second)”)
grid(col=”blue”)
*/
```

4 Implement and evaluate linear time Fibonacci algorithm

1. Implement the linear time Fibonacci algorithm `int fib2(int)` from the lecture notes in C++. Name the C++ file `fib-linear.cpp`.
2. Write a test function in C++ to check the results when $n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
3. Create a main function in the C++ file to call the test function you just developed. Make sure the test is passed.
4. Develop R code to visualize the runtime in the same C++ source code file `fib-linear.cpp`.
5. Does the runtime grow linearly as input number n increases?
6. How many times is the linear algorithm faster than the exponential algorithm in `fib1.cpp` on the same input?
7. Write a lab report that describes your work done in the following sections:
   (a) Introduction (define the background and the problem),
   (b) Methods (provide the solutions),
   (c) Results (show the numbers and figures),
   (d) Discussions (implications and issues),
   (e) Conclusions (summarize the lab and point to a future direction).
8. Submit the program and your report online.