Parrallel Computing with OpenMP

C++ for R Applications Working Group

Adam Wang

Introduction

OpenMP (Open Multi-Processing) is a library that allows you to parallelize your code. It is a set of compiler directives that tell the compiler to run certain parts of your code in parallel.

Why? Because it is faster.

We have only one goal today: run `nicemp.cpp` successfully.

There two functions in the script:

1. `rmvn_loop`: a simple loop that generates random numbers from multivariate normal.
2. `rmvn_par`: same functionality, but parallelized with OpenMP.

The only difference between the two functions is this line before the loop:

```
#pragma omp parallel for
```

This tells the compiler to parallelize the following `for` loop.

If successful, the parallelized function should run about **2X** faster.

Read this if you want to learn more.

Installation

Windows

Just add this line in `nicemp.cpp`:

```
// [[Rcpp::plugins(openmp)]]
```

Mac

Long story short, Apple's clang does not support OpenMP. So we need to manually install a few things:

1. Install Homebrew from the pkg file. Homebrew is a free and open-source software package management system that simplifies the installation of software…
2. Open Terminal and run `brew install gcc`, then `brew install libomp`. This will install the GNU Compiler Collection and the OpenMP library. This can take a while.
3. In terminal, run `brew install vim`, this will install the Vim text editor.
4. Still in terminal, run `vim ~/.R/Makevars`
5. Add the following lines by pressing `i` to enter insert mode:

```
CXXFLAGS += -Xclang -fopenmp
LDFLAGS += -lomp
```

6. Press `esc` to exit insert mode, then type `:wq` to write and quit. Done!

The two mysterious lines tell R to use OpenMP when compiling C++ code.

Test

Open the script `nicemp.cpp` in Rstudio and run it.

Caveats

Some loops are not parallelizable. For example, MCMC loops are not parallelizable because each iteration depends on the previous one.

Parralelization is not always faster. It depends on the size of the problem and the number of cores available. If you have a small problem, the overhead of parallelization can make it slower.