Compiling a Tuned BLAS for R

J. P. Olmsted

Last Updated April 6, 2011

Contents

1 Introduction

The purpose of this tutorial is to provide a walk-through of the steps necessary to begin using an optimized Basic Linear Algebra Subroutines (BLAS) bundle with R. This tutorial covers instructions for both *nix environments and the Windows environments.

Contact If you find any ambiguities, have any comments or questions, or find an error, please don't hesitate to contact me at jpolmsted@gmail.com.

Prerequisites In this presentation, I assume you are familiar with the statistical computing environment R and how to use it.[1] I also assume that you have access to my website which hosts some other documents to which I will refer.[2] I assume you have access to the website from which we will download the BLAS used.[3].

[1] See <http://cran.r-project.org/>.
[2] See <http://www.rochester.edu/college/gradstudents/jolmsted/>.
[3] See <http://www.tacc.utexas.edu/tacc-projects/gotoblas2/>.

2 Motivation

Why would you want to use an optimized BLAS? Most matrix operations done in R are actually passed off to a BLAS which is a distinct code-base. Instead of R's dedicated developers maintaining that software, they set up R so that it can use a BLAS as a building block. If you've ever installed R, it is possible to have never thought about this issue before and to have successfully completed the installation without worrying about pulling in other pieces of software. This is because R, by default, uses a reference implementation of a BLAS that is not optimized.[4] So, the answer to the question of *why use an optimized BLAS?* is that you can make your matrix operations in R run faster. Indeed, Revolution Analytics' version of R realizes performance increases for precisely this reason.[5]

Why not Revolution R? For sure, there is nothing wrong with Revolution R. Indeed, Revolution Analytics has contributed some incredibly useful packages to the R community. However, it may be the case that Revolution R is not for you.

- First, the BLAS implementation that they use may not be the fastest on your computer, although it is quite fast.
- Second, Revolution R (either the Community of Enterprise version) uses a version of R that lags behind current development. While version changes tend not to matter too much for most users, many packages require sufficiently new versions of R.
- Third, parts of Revolution R come with more restrictive licenses than R itself does.[6]
- Fourth, if you are choosing between CRAN R and Revolution R Enterprise, the IDE that they provide has a costly resource overhead which isn't necessary to gain the computation speed.

While it may seem strange to frame this discussion against the products provided by Revolution Analytics, their software has advanced performance in the R environment and the attention paid to performance by R users. It is, I think, incredibly important to emphasize that Revolution R is not right for everyone, but that it is certainly a great idea for many users. This tutorial is not to take away from the merits of Revolution Analytics, but rather mention and demonstrate an alternative.

3 Overview

There are three main steps to setting R up such that it uses an optimized BLAS implementation. First, download and compile the new BLAS. Second, prepare R such that it is ready to use the new BLAS. Third, instruct R to use the new BLAS. I provide step by step instructions for two different operating system environments.

[4] See <http://www.netlib.org/blas/>.

[5] See <http://www.revolutionanalytics.com/products/revolution-r.php>.

[6] We will use GotoBLAS as an alternative in this tutorial. Although it, itself, does not have the most open license, it is more permissive in the scope of academic uses than the Intel MKL BLAS provided by Revolution R is. If the GotoBLAS license is too restrictive for your uses, the ATLAS BLAS is a more permissive alternative (see <http://math-atlas.sourceforge.net/>). For me, the *Termination* clause is the most alarming, though my belief that this will ever be relevant is near 0.

4 *nix Environment

These instructions have been verified on Ubuntu 10.10, but should not be Ubuntu-specific. There are, indeed, multiple ways to accomplish our goal. I provide instructions to replicate the approach that I take. As is par for the course, your mileage may vary.

4.1 Downloading and Compiling the BLAS

I use GotoBLAS in this tutorial and on my personal machines. Its license is sufficient for me and my work (in the academic world), but worth reading.

1. Navigate to <`http://www.tacc.utexas.edu/tacc-projects/gotoblas2/`> and download the source code.
2. Ensure that your system has the requisite tools to compile software from source. On my Ubuntu machine the following command is sufficient for this step:

```
1 sudo apt-get install build-essential
```

3. Extract the source code archive with something like the following that depends on the version you have downloaded:

```
1 tar -xvf GotoBLAS2.tar.gz
```

4. Navigate to the directory created by extracting the source code.
5. You can briefly read some of the documentation which starts with 00, 01, 02, etc. I recommend reading the license.
6. Next, decide whether you would like to build GotoBLAS such that it uses more than one processor or core (if available). I've never found an appreciable difference in speed between the single-threaded and multi-threaded binaries but that could be dependent on my machine and the problems I've tested it on.[7] For the purpose of this tutorial, I will assume you use the default setting. The default setting is to use multi-threading only if the number of available processors/cores is larger than 2. Read `02QuickInstall.txt` if the default is not sufficient.
7. Once you are in the top-level GotoBLAS directory, simply type `make`. Assuming you are using some standard hardware and environment, the configuration should proceed without a problem.
8. Wait. Twenty minutes is not unreasonable for the compilation to finish. Five to ten minutes is probably more realistic on newer machines.
9. Once the compilation is done, you will notice two files (`libgoto2.a` and `libgoto2.so` in the current directory. These are symbolic links and not files, themselves. Determine the targets of these links with a command like `ls -l` and move the targets (not the links) to a sensible location for use in the future. I store them in ~`/lib/goto/` which is in my home directory (hence the ~).

[7] See my website for benchmarking comparisons across a wide range of configurations.

4.2 Preparing R

There are two ways to instruct R to use the newly compiled BLAS.[8] You can either build the software into R itself as you compile R or you can instruct R to look for the BLAS in a particular, distinct file on you computer. I prefer the latter because it separates the (distinct-to-me) tasks of keeping a BLAS up-to-date and keeping R up-to-date. Moreover, if R looks for the BLAS in an external file, then if you need to swap out your current BLAS for a different one, you need only move a file to a new location instead of re-compiling.[9] In R parlance, we will tell R to use a *shared BLAS*.

The next constraint we have on our R binary is that in my experience, though I can't understand why, R crashes (i.e. a `segfault`) when I use GotoBLAS as a shared BLAS once I do something in R that calls the `Tcl Tk` interface.[10] Forced to choose between faster calculations and an ugly GUI that I use once every so often which can be replaced by text, I choose speed. As such, we have to compile R such that it does not use the `Tcl Tk` interface.[11]

1. Navigate to CRAN (at <`http://cran.r-project.org/`>) and download a copy of the source code that suits you.
2. Extract the compressed directory hierarchy with something like the following:

```
tar -xvf R-2.12.tar.gz
```

3. Switch into the directory you just extracted with something like `cd R-2.12`.
4. Ensure that you have all of the necessary utilities and dependencies required to compile R. In Ubuntu, this step is as easy as `sudo apt-get build-dep r-base`. However, this will be different for different Linux distributions.
5. Configure the compilation instructions to build R such that it does not use the `Tcl Tk` interface and uses a shared BLAS file.[12]

```
    ./configure --enable-BLAS-shlib --enable-R-shlib --enable-memory-profiling --with-
        tcltk=no
```

6. Build the binary.

```
make
```

7. Install the binary. I use `checkinstall` which is for Debian-based systems (e.g. Ubuntu). You could use `make install`.

```
sudo checkinstall
```

[8] See <`http://cran.r-project.org/doc/manuals/R-admin.html`> for a full discussion.

[9] The developers of R mention that there may be some slowdown associated with using an external file and not compiling the BLAS into the R binary, but my benchmarking has shown no evidence of an appreciable difference.

[10] If you aren't familiar with this by that name, it is the interface that pops up too ask you from which mirror you want to download R packages and the one that actually lists R packages.

[11] It may be the case that you can avoid this step, but it is the way I proceed and it works.

[12] We also enable profiling here and the external libraries which provide R functionality inside other software programs (i.e. C++). This isn't necessary, but it can't hurt.

4.3 Having R Use the BLAS

The remaining part of the setup is to swap out the default reference BLAS that is contained in the shared file for the GotoBLAS library we compiled.

1. Find the precise location of R's BLAS file. You can use something like `locate` as I do below:

```
updatedb
locate libR
```

   On my system the path we care about is `/usr/local/lib/R/lib/libRblas.so`.

2. Backup the original reference BLAS:

```
sudo mv /usr/local/lib/R/lib/libRblas.so /usr/local/lib/R/lib/libRblas.so.reference
```

3. Copy the compiled BLAS to the location that R expects the shared BLAS to be. Assuming `~/lib/goto/libgoto2.so` is the compiled BLAS, use:

```
sudo cp ~/lib/goto/libgoto2.so /usr/local/lib/R/lib/libRblas.so
```

5 Notes

1. You'll need to make sure that the BLAS and R were compiled by the same version of compiler, otherwise you may run into some error messages.

2. If you use a one user system, it can be easier to use a symbolic link to the library in your home directory instead of copying the BLAS to the `/usr/` directory.

6 Windows Environment

These instructions have been verified on Windows XP, but should not be Windows XP-specific.

To be completed.