

Optimizing Linux for Your Processor

or How to Double Your Computers Performance

by Frank Kruchio[New Zealand]
Konstantin Malakhanov[Germany]
Steve Martin[New Zealand]

Version: February 2, 2002

Copyright © 2000 Frank Kruchio and all the contributors to this tutorial. All rights reserved. This document is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this document; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Contents

1	Introduction	4
1.1	Getting Started	4
2	What do I need to get started ?	5
2.1	Installing autoconf and automake	5
3	Optimization for Athlon and Pentium	6
3.1	Using the gcc/g++ optimization flags	6
3.2	Remark on the rpmrc location	6
3.3	Important Notes	8
3.3.1	Can I install i386 RPMs later ?	8
3.3.2	What is BuildRoot ?	8
4	Optimizing QT for KDE	10
4.1	Getting the source RPM.	10
4.2	What if my compilation fails ?	10
5	Optimizing KDE	12
5.1	Optimizing KDE in the correct order	12
6	Optimizing XFree86	13
6.1	Faster GUI for gamers and all	13
7	Appendix	14
7.1	About the author	14
7.2	Feedback	14
7.3	Useful resources	14

1 Introduction

Before you read any further, please make sure you have the latest version of this tutorial. The latest version can be downloaded from:

<http://homepages.paradise.net.nz/frankkru/page2.html>
[my website](#).

1.1 Getting Started

Should you take this tutorial ? If you are interested in getting the maximum speed and efficiency out of your computer and do not mind learning new things you may find this tutorial useful.

The availability of free source code under Linux allows the end user to optimize software for a specific processor architecture. Once the exclusive domain of Linux gurus, today Linux optimization is easy for Linux users around the world. Linux is fast: but most Linux users I have come across almost never spend time on optimizing their Linux distribution of choice. Indeed the benefits are immediately noticeable and it provides a user experience and performance that no windows user can enjoy on the same hardware. While Linux distributions such as Slackware and Debian are common, this tutorial will focus only on the Red Hat Package Manager(RPM) based distributions such as SuSE, Red Hat and Mandrake... for the following reasons.

- People using Slackware and Debian are generally considered seasoned Linux users who know how optimizations are done.
- This tutorial is focusing on new Linux users: as such, they are more likely to use an RPM based distribution when they start out with Linux.

Thanks to RPM, users new to Linux can easily optimize Linux for their own processor architecture taking full advantage of their hardware. Distributions such as Red Hat are only compiled for i386, Suse i486 and even Mandrake only comes as i586. This leaves plenty of headroom to optimize your favorite Linux distribution for Duron, Athlon, Athlon XP, Athlon MP, Pentium II/III/IV and Celeron. While my example is focusing on optimizing workstation packages there is nothing holding you back to optimize a Linux server, firewall, router or any other Linux box.

2 What do I need to get started ?

2.1 Installing autoconf and automake

I will be using SuSE Linux 7.3 Professional on kernel 2.4.17 but all commands should work on any of the RPM based distributions out there. Before you begin, you will need to install two packages, one called *autoconf* and the other is *automake*. Check your Linux CD or download autoconf and automake from the ftp site/website of your distribution vendor in case they are not already installed on your hard drive. To install autoconf and automake, log in as **root** and find the directory where the autoconf.rpm and automake.rpm is located.

Now, run RPM as:

```
rpm -Uvh *.rpm
```

to install both packages (SuSE users can use yast → package management → install packages → directory).

After installation, you can confirm from a shell that installation was successful:

```
frank@babe:~> rpm -q autoconf automake
autoconf-2.52-44
automake-1.4_p5-44
```

If RPM reports the version number you successfully installed the required packages, now you are ready to move onto the next step.

NOTE:

If you would like to learn how the tools autoconf and automake work, you can visit the

http://sources.redhat.com/autobook/autobook/autobook_toc.html
[website](#) online book on the subject.

3 Optimization for Athlon and Pentium

3.1 Using the gcc/g++ optimization flags

RPM will consult a file in your home directory called *.rpmrc* during the optimization process. We need to create a new *~/.rpmrc* file in case you can not find one in your home directory. Simply open your favorite text editor and enter the following lines; feel free to copy and paste the rpmrc file below. Make sure the name of the file you save in your home directory is *.rpmrc*, note the dot. Place the flags in the *.rpmrc* file on one line where it says *optflags:*; RPM complains if you do not place them on one line or just use line breaks.

3.2 Remark on the rpmrc location

Instead of customizing *~/.rpmrc* file you can also put the same lines as root in */etc/rpmrc*. The difference is that */etc/rpmrc* will be used for all users, while *~/.rpmrc* values are only for you. One advantage of having a *~/.rpmrc* file is that you do not need to back it up if you will do a fresh install of your favorite Linux distribution. Assumed that you put your home directories and */etc* in the different partitions. your home directory will be untouched. The global RPM configuration file */etc/rpmrc* is deleted during a new installation.

```
# ----- cut here -----
# 2001/12/23, modified by
# Frank Kruchio [New Zealand]
# Konstantin Malakhanov [Germany]
# Any per-system configuration should be added to /etc/rpmrc
# while per-user configuration should be added to ~/.rpmrc
#
#####
# Values for the gcc/g++ compiler, Add these flags on 1 line !
# -mcpu=i686 is redundant
# add -malign-functions=4 -mpreferred-stack-boundary=2 for Athlon
#####
optflags: i686 -O3 -march=i686 -fomit-frame-pointer \
          -funroll-loops -fforce-mem -fforce-addr \
          -fexpensive-optimizations

#####
# Canonical arch names and numbers
```

```

arch_canon:      i686:  i686    1

#####
# Canonical OS names and numbers
os_canon:       Linux:  Linux   1

#####
# For a given uname().machine, the default build architecture
buildarchtranslate: i686: i686

#####
# Architecture compatibility
arch_compat:    i686: i686

buildarch_compat: i686: i686
# ----- cut here -----

```

If you have an Athlon, Duron, Pentium/II/III/IV or Celeron use the above file as it is.

For Athlons and Durons with GCC 3.0 or newer use:

```

optflags: athlon -O3 -march=athlon \
-funroll-loops -fomit-frame-pointer -fforce-mem \
-fforce-addr -finline-functions -malign-functions=4 \
-mpreferred-stack-boundary=2

```

and replace i686 with the word athlon everywhere in the file.

If you have an AMD K6-II or K6-III, replace i686 with the word i586 everywhere in the file. Both the K6-II and K6-III can execute MMX instructions. See:

```
cat /proc/cpuinfo
```

and see the entries under the CPU flags.

The `/etc/rpmrc` file above is just a short version of `/usr/lib/rpmrc` modified by me for my Athlon XP 1600+ processor. Here, you tell RPM what optimization flags you want to use for your processor. For detailed explanation on what they do, see *man gcc* OPTIMIZATION OPTIONS starting at line 960. The `-O3` flag is the highest gcc/g++ optimization flag, the compiler will accept higher settings such as `-O9` but it will only use `-O3`. There are

other optimization flags such as `-ffast-math` but I recommend you read the manual on `gcc` ¹ before applying them. In fact this is all I use on my Athlon XP 1600+ model 662 for maximum stability:

```
optflags: i686 -O2 -march=i686 -fomit-frame-pointer \  
-malign-functions=4 -mpreferred-stack-boundary=2
```

3.3 Important Notes

3.3.1 Can I install i386 RPMs later ?

Please note that after you created `~/rpmrc` you can still install lower say i386 optimized binary or source rpms but you may need to call `rpm` as:

```
rpm --target=i386 ...
```

3.3.2 What is BuildRoot ?

Beware of RPMs without BuildRoot specification! What exactly is that BuildRoot? Please visit ‘Maximum RPM’ [here](#) for more details.

“The build root is not the root directory under which the software is built. Rather, it is the root directory for the install phase of the build. When a build root is not specified, the software being packaged is installed relative to the build system’s root directory.”

Usually, well-specified RPMs always declare BuildRoot. A BuildRoot aware RPM can make two difficult situations much easier:

1. Performing a build without impacting the build system.
2. Allowing non-root users to build packages.

Let’s study the first situation in a bit more detail. Say, for example, that `sendmail` is to be packaged. In the course of creating a `sendmail` package, the software must be installed. This would mean that critical `sendmail` files, such as `sendmail.cf` and `aliases`, would be overwritten. Mail handling on the build system would almost certainly be disrupted. In other words, simply building a new version RPM without BuildRoot will silently overwrite files of the installed RPM.

¹Run the command `man gcc` in a shell.

The second situation is not so important for a stand-alone workstation. But if you care about security, you will try to do whatever is possible without using root account. But it is not enough for RPM to declare BuildRoot. The build specifications inside of RPM must be slightly modified to take BuildRoot into account. So some lazy RPM packagers don't care about BuildRoot and you will get all the trouble.

So you try to build an RPM and it fails, because it cannot write in */usr/bin* or something like that. Then you know that RPM does not use BuildRoot properly. How likely is that? Well, SuSE 6.4 had no BuildRoot for 1174 RPMs out of 1316 supplied; SuSE 7.2 has 459 of 1260 RPMs without BuildRoot. Things are getting better, but then, you still can catch one without BuildRoot. The best you can do is to back up all important files from the installed RPM, build the RPM as root and politely complain to RPM packager.

4 Optimizing QT for KDE

4.1 Getting the source RPM.

I would recommend you to start out optimizing with a small src.rpm file program and when you see how it works you will realize that it is very easy indeed. I will use here an example of qt for KDE but this can be applied to any other program in src.rpm or filename.spm package.

1. Copy from your CD or download to any directory of your choice the file qt-devel.rpm (a small 700kb file)
2. Install qt-devel.rpm (You may need to temporarily rename */etc/rpmrc* to */etc/rpmrcHOLD* to do this)
3. Download qt-2.3.2-10.src.rpm (Or the latest QT .src.rpm, .spm package)
4. Start optimizing; after making sure you have */etc/rpmrc* simply run the following command:

```
rpm --rebuild qt-2.3.2-10.src.rpm
```

5. If you have no errors once the compilation finished you will find all the optimized packages in one of the directories below, depending on what architecture you set in your rpmrc file.

```
/usr/src/packages/RPMS/i586/  
/usr/src/packages/RPMS/i686/  
/usr/src/packages/RPMS/athlon/
```

6. Just install your freshly optimized RPMs and enjoy the extra speed they bring to your server or desktop computers. Of course if you have more than one computer these optimized packages can be installed on any other with a similar setup.

4.2 What if my compilation fails ?

If you run into problems be patient and try the followings. Install your source rpm in this case qt-2.3.2-10.src.rpm and read the .spec file in */usr/src/packages/SPECS/*. The qt.spec file contains information on what needs to be installed for qt to compile correctly. You can also start optimized package building with:

```
rpm -bb /usr/src/packages/SPECS/qt.spec
```

The `-bb` option is invoked with **one** hyphen `-bb` only as opposed to two in `-rebuild`.

By far the easiest way to check if you have all the required packages installed is to open a shell and type `'rpm -q '`, next open the `qt.spec` file, copy and paste the required package names into the shell and press enter. Quick and easy to check even 50 package names this way.

5 Optimizing KDE

5.1 Optimizing KDE in the correct order

I found that upgrading to a newer optimized KDE package to be very simple given you are following these steps.

- Make sure you optimize and install QT first. See the Optimizing QT for KDE section.
- Before you can start optimizing kdelibs for your own cpu, you will need to download and install the kdelibs-devel.rpm package. Kdelibs-devel.rpm contains libraries and header files that are used in the optimization/compilation process. Optimize kdelibs. Just before installing the optimized KDE libraries, log out from KDE to a console then install the new libraries.
- Before rebuilding kdbase.src.rpm you will need to install kdbase-devel, lesstif, lesstif-devel, pam-devel and flex. Optimize kdbase. Install KDE base after you logged out from KDE.
- At this point you can optimize and install any of the other KDE packages in any order.

As an indication kdenetwork takes about 45 minutes to optimize, but kdbase approximately takes two hours to optimize and build your new .rpm package on a 400Mhz/128MB RAM PC. Linux is multitasking, therefore you can browse the net, download or do whatever else on your PC while the new packages are built for you. The end result is that KDE is much more **responsive and faster**.

6 Optimizing XFree86

6.1 Faster GUI for gamers and all

If you are feeling ambitious to rebuild XFree86 from the `xf86.src.rpm` (sometimes just called `xf86.spm`) install `x-devel`, `bison`, `libusb` and `glidesdk`. Download the rather large 45MB `xf86.spm` file containing the source code for XFree86 and start optimizing as:

```
rpm --rebuild xf86.spm
```

Compilation times depend on your CPU / RAM and optimization flags. Code optimization is a tradeoff between program speed and program size. Higher optimization flags will increase compile time and the executable size. XFree86 will approximately take two hours to optimize and build on a 400Mhz 128MB RAM PC. After installation of the new optimized XFree86 packages is complete, you can check if you use your own optimized version with the following command:

```
frank@babe:~ > rpm -qi xf86
Name           : xf86                      Relocations:
Version        : 4.1.0                  Vendor: (none)
Release        : 63                     Build Date: Mon 19 Nov 2001
Install date: Mon 19 Nov 2001          Build Host: babe.local
Group          : X11/XFree86            Source RPM: xf86-4.1.0-63
Size           : 30834111                License: 1994-2000
Summary        : Basic X11 package
Description    :
Contains XFree86 and some other programs from the ...
----- modified here for brevity -----
```

You can see from the output that the Build Host: `babe.local` (in my case) is the local computer. Not a difficult process as you can see and thanks to RPM you can run XFree86 at maximum speed now.

Note: If your build breaks, the rule applies here just like on earlier pages to check in the specifications file `/usr/src/packages/SPECS/xf86.spec` the dependencies to make sure you have all required components installed for a successful build.

7 Appendix

7.1 About the author

Frank Kruchio graduated in 2001 with a B.Sc. in Computer Science from Victoria University of Wellington, New Zealand. Most of his spare time is devoted learning about Linux and computers in general; but he also enjoys fly fishing. You can contact the author at `frank.kruchio@paradise.net.nz`.

7.2 Feedback

As always there is room for improvement and this tutorial is no exception. If you have any constructive ideas or comments you like to add or seen included in the next version of this tutorial please feel free to e-mail your ideas. I look forward to hearing from you!

7.3 Useful resources

- There is an *Optimizing linux for Athlon* tutorial at http://aboutlinux.com/art_k7opt1_a.html click [here](#) to visit the website.
- Intel's C/C++ compiler for Linux <http://www.intel.com/software/products/compilers/c50/linux/> click [here](#) to visit the Intel.
- Linux Performance tuning website <http://linuxperf.nl.linux.org/> click [here](#) to visit the website.

2

²Linux optimization February 2, 2002