Porting R to Darwin/X11 and Mac OS X

by Jan de Leeuw

Mac OS X

Earlier this year Apple officially released OS X, its new operating system. OS X now comes preinstalled on all Macs, although by default you still boot into MacOS 9.x. But soon OS X will be the default.

OS X is not an incremental upgrade, it is a completely new operating system. It has a layered architecture. The lowest layer is Darwin, which consists of the Mach 3.0 kernel and a version of BSD 4.4. Thus OS X is, among other things, a certified and POSIX compliant Unix. Darwin is (certified) Open Source, and it can be downloaded from the Apple servers. One surprising consequence of the above is that soon Apple will be the largest distributor of Unix, and that soon OS X will be the most popular Unix on the planet, although most users will be blissfully unaware of this fact.

On top of Darwin there is a lot of proprietary software, used to generate the user interface components. The main libraries are Apple's version of OpenGL for 3D, QuickTime for multimedia, and Quartz for printing and screen drawing. Quartz replaces Display Postscript in earlier versions of the system, using PDF as its native format.

Application environments

On top of the three graphics engines are no less than five application environments that developers can use.

Classic For the foreseeable future it will remain possible to boot into OS 9.x, and to run older Macintosh programs in the Classic environment with OS X, which emulates an older Macintosh with OS 9.x. Some of the more powerful programs for the Mac, such as Office and Photoshop and SPSS, still have to run in Classic, although Carbon versions have been announced.

Carbon The classical Mac Toolbox API has been cleaned up and extended. This now makes it possible to write applications that run natively on both OS 9.x and OS X. Netscape, MSIE, R, Stata, AppleWorks have all been carbonized. It is of some interest, however, that there are two types of Carbon applications. Those that can run on OS 9.x are under the control of the Code Fragment Manager and use the PEF executable format. If run on OS X, they run on top of a layer that translates CFM/PEF to dyld/Mach-O. Mach-O is the native format for OS X, and program control is exercised by the dynamic linker dyld. The other type of Carbon application is dyld/Mach-O, which means it does not run on OS 9.x.

Cocoa This is the native OS X API, inherited from its NeXTStep and Rhapsody parents and grandparents. Applications using these interfaces use optimally the capacities of the OS. Cocoa applications are still comparatively rare, because they have to be written from scratch, either in Objective-C or in Java. But there are already fine browsers, spreadsheets, editors, graphic tools, and TeX systems in Cocoa.

Java The JDK (including runtime, JIT compiler, AWT, and Swing) is integrated with OS X, and Java libraries are available to write Cocoa applications. Swing, of course, has the native OS X look-and-feel. Of course anything you write in Java on OS X is (at least in principle) completely portable.

BSD Darwin comes with optimized Apple versions of the GNU tools. Since the application environment for Darwin is FreeBSD, porting of Unix programs is a breeze. It can be made even easier by using Fink (see below). In particular, it is trivial to install an X server, in fact a complete X11R6, using Xfree86, and a large number of different window managers. There are ports to Darwin of all of gnome, including the Gimp and Guppi, of various Matlab like programs such as octave, scilab, yorick, and of all the standard X graphic tools such as xfig, tgif, xpdf, xdvi, xv, ghostview, gnuplot, grace, xgobi.

User experience

The Mac OS X user, of course, will not notice any of these under-the-hood changes. The obvious change is Aqua, the new look-and-feel, often described as "lickable". Windows and the menu bar look different, there is a "dock", and so on. The Aqua interface is automatic for all Cocoa and Carbon applications that use Quartz to draw to the screen.

The user will notice greatly increased stability of the OS. This is mostly provided by the Mach kernel, which provides protected and advanced virtual memory, as well as preemptive and cooperative multitasking and threading. OS X will run forever without crashing, and applications that crash don't take down the OS with them any more. The need to reboot has virtually disappeared.

Moreover, OS X promises speed, although not in the early versions. The OS is written to take full

advantage of multiprocessing, and multiprocessor Macs are becoming more and more common. Rumor has it that the G5 will even be multicore. Many graphics programs, including Quartz and OpenGL, are optimized for the Altivec vector processor on the G4 chip. Recent builds of the OS show great speed.

Finally, remember that OS X is first and foremost a Unix, i.e. a multitasking and multiuser OS. You have to login, you can allow others to login, and people can login remotely. Although you can use the system as a dedicated single-person desktop OS, that is only one of its uses. There are many people who log into the Cube in my office.

Porting problems

Darwin/X11 programmers must take into account some important differences with the more usual ELF based Unix systems. Most of those are due to the Mach heritage. All these peculiarities had to be taken into account in building R, and in modifying the `autoconf` configure files.

In the first place, Darwin maintains a strict distinction between two types of shared libraries. There are bundles, which can be loaded at runtime into an application using the appropriate dynamic loading interface. Also, there are dynamic libraries, that are used at link time when building applications or other libraries. Different compiler and linker switches are needed to build the two different types of libraries. For ELF systems the two types coincide. Building R as a shared (dynamic) library, which can be linked against other application programs, will be available in R-1.4.0 and does not work yet in R-1.3.1. The modules and packages which use bundles of object code that are loaded at runtime work fine.

Second, the Darwin dynamic linker `dyld` is very intolerant, and does not allow multiply defined symbols at all. The static linker is much more tolerant. Thus one must make sure not to include a file with definitions more than once, and so on.

Third, the API for dynamic loading is very different from the more usual `dlopen()` interface in ELF systems.

And finally, some of the necessary components needed for building R (X11R6, a Fortran compiler) are missing from the current version of Darwin.

Fink

The task of porting BSD and X11 software has been made easy by the existence of Fink (see `http://fink.sourceforge.net`). This is a package management system for Darwin setup by Christoph Pfisterer, and maintained by a group of volunteers. There are now more than 300 packages in Fink, and you can say `fink install foo` to download, configure, compile, and install package foo, and then `fink update foo` to update the package when it has changed in the Fink central location. Of course such package management systems exist for Linux, Debian, FreeBSD (and actually for R and Stata), but it is good to have one for Darwin as well.

What do you need from Fink for building a Darwin version of R? In the first place `Xfree86`. The Darwin version has been modified with a Cocoa front end called XDarwin that let's you choose between full-screen and rootless mode, where in rootless mode the X11 windows exist on the same desktop as the Aqua windows of the OS X Finder. Second, you can install all of gnome, which can be used for the (experimental and unsupported) gnome module in R. Third, Fink has ATLAS, an optimized BLAS library for OS X. Fourth, there is `dlcompat`. This wraps the dyld API for dynamic loading in the familiar ELF `dlopen` API, so you can continue to use the standard calls in the R sources. Fifth, there is `tcl/tk`, for the `tcltk` package in R. And finally there are various other libraries, which are either not in Darwin or are more recent versions. Examples are `libjpeg`, `libpng`, `libz`, and `libreadline`. There is also a `g77` in Fink, but it does not work with the configure scripts in R, so all our builds so far use `f2c`.

In fact, R-1.3.1 base and recommended are both in Fink. The info scripts and patch files are maintained by Jeffrey Whitaker (`jsw@cdc.noaa.gov`). This provides you with yet another way to install R on your Mac.

R

Combining all this new knowledge makes it possible to describe what we have on CRAN and what we still need. We have a CFM/PEF Carbon version of R, made by Stefano Iacus, and described in the first issue of R-News. It uses a Carbon version of the Macintosh QuickDraw driver. We also have a Darwin/X11 version, with support for Tcl/Tk, GNOME, and ATLAS, maintained by Jan de Leeuw (me).

The Carbon version runs on both OS 9.x and OS X, but we have seen that it needs a dyld/Mach-O layer to run on OS X, so it's not really native. There is no support in the Carbon version for Tcl/Tk, and the internet-based R package update and install system is not available. There are no free tools to build this version in OS X; you have to build it in OS 9.x, or buy an IDE from Metrowerks or Absoft.

The Darwin/X11 version is dyld/Mach-O, and is consequently native in that sense, but it does not use the native Quartz library and Cocoa interfaces at all. If you run the X server in full-screen mode, your Mac looks just like a Linux or Solaris machine. This is somewhat disappointing for Mac people.

There are various ways in which the current situation can be improved. Stefano is working on a Quartz driver for the graphics. It would be useful

to have a dyld/Mach-O Carbon version, truly native to OS X. The Quartz driver also brings us closer to a Cocoa version of R, which could be implemented initially as a Cocoa shell around the Darwin version of R.

Much will depend on the reception of OS X, and on how many Mac users will switch from 9.x to X. If your hardware supports OS X, I think switching is a no-brainer, especially if you program, develop, or compute. As I have indicated above, the possibilities are endless.

Jan de Leeuw
University of California at Los Angeles
deleeuw@stat.ucla.edu

RPVM: Cluster Statistical Computing in R

by Michael Na Li and A.J. Rossini

rpvm is a wrapper for the *Parallel Virtual Machine* (PVM) API. PVM (Geist et al., 1994) is one of the original APIs for extending an application over a set of processors in a parallel computer or over machines in a local area cluster. We discuss the PVM API, how it is implemented in R, and provide examples for its use. **rpvm** provides a quick means for prototyping parallel statistical applications as well as for providing a front-end for data analysis from legacy PVM applications.

Introduction

PVM was developed at Oak Ridge National Laboratories and the University of Tennessee starting in 1989. It is a *de facto* standard for distributed computing designed especially for heterogeneous networks of computers. The notion of "virtual machine" makes the network appear logically to the user as a single large parallel computer. It provides a mechanism for specifying the allocation of tasks to specific processors or machines, both at the start of the program as well as dynamically during runtime. There are routines for the two main types of intertask communication: point-to-point communication between tasks (including broadcasting) and collective communication within a group of tasks.

The primary message passing library competitor to PVM is MPI (*Message Passing Interface*). The biggest advantage of PVM over MPI is its flexibility (Geist et al., 1996). PVM can be run on an existing network consisting of different platforms (almost all platforms are supported, including Microsoft Windows 98/NT/2000 systems). Tasks can be dynamically spawned, which is not supported in MPI-1 upon which most MPI implementations are based. Hosts can be dynamically added or deleted from the virtual machine, providing fault tolerance. There are also a visualization tool, `xpvm`, and numerous debugging systems. MPI has advantages of speed as well as being an actual standard. However, for prototyping and research, it isn't clear that either of these are critical features.

PVM has been successfully applied to many applications, such as molecular dynamics, semiconductor device simulation, linear algebra (ScaLAPACK, NAG PVM library), etc. It also has great potential in statistical computing, including optimization (expensive or large number of function evaluations; likelihood computations), simulations (resampling, including bootstrap, jackknife, and MCMC algorithms; integration), enumeration (permutation and network algorithms), solution of systems of equations (linear, PDE, finite-element, CFD).

This article presents a new R package, **rpvm**, that provides an interface to PVM from one of the most powerful and flexible statistical programming environments. With **rpvm**, the R user can invoke either executable programs written in compiled language such as C, C++ or FORTRAN as child tasks or spawn separate R processes. It is also possible to spawn R processes from other programs such as Python, C, FORTRAN, or C++. Therefore **rpvm** is ideal for prototyping parallel statistical algorithms and for splitting up large memory problems. Using **rpvm**, statisticians will be able to prototype difficult statistical computations easily in parallel. The rest of the article which follows looks at installation, features, a programming example, and concludes with issues for on-going development.

Installation

PVM source code can be downloaded from http://www.netlib.org/pvm3/pvm3.4.3.tgz. Binary distributions exist for many Linux distributions (see individual distributions) as well as for Microsoft Windows NT/2000/XP. However, the Windows implementation of **rpvm** is untried (it is possible to communicate with C or FORTRAN processes running under Microsoft Windows). The following procedures refer to UNIX-like environments.