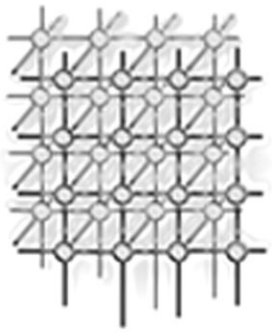
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2002; **14**:1445–1456 (DOI: 10.1002/cpe.670)

# NetBuild: transparent cross-platform access to computational software libraries



Keith Moore*,† and Jack Dongarra

*Innovative Computing Laboratory, University of Tennessee, 1122 Volunteer Blvd., Suite 413, Knoxville, TN 37996-3450, U.S.A.*

**SUMMARY**

**NetBuild is a suite of tools which automate the process of selecting, locating, downloading, configuring, and installing computational software libraries from over the Internet, and which aid in the construction and cataloging of such libraries. Unlike many other tools, NetBuild is designed to work across a wide variety of computing platforms, and perform fine-grained matching to find the most suitable version of a library for a given target platform. We describe the architecture of NetBuild and its initial implementation. Copyright © 2002 John Wiley & Sons, Ltd.**

KEY WORDS: computational software; software libraries; automatic library selection

## 1. OVERVIEW

### 1.1. Description and goals

As computationally intensive modeling and simulation become increasingly important staples of scientific life across nearly every domain and discipline, the challenge of deploying the software that encodes the underlying computational science has become increasingly complex and pressing. The ubiquity of the Internet and the establishment of large software repositories, such as Netlib [1], have made it relatively easy to provide basic access to such software. However, a rapidly growing and more diverse user community still has to cope with the process of collecting, compiling, installing, configuring, and managing the software on a constantly changing and increasingly elaborate computing infrastructure.

For all of its promise, the development of Grid-enabled computing [2] may actually exacerbate this problem. A program may require that certain software libraries be installed on large numbers of diverse

*Correspondence to: Keith Moore, Innovative Computing Laboratory, University of Tennessee, 1122 Volunteer Blvd., Suite 413, Knoxville, TN 37996-3450, U.S.A.
†E-mail: moore@cs.utk.edu



*Received 1 July 2001*
*Revised 6 January 2002*

computing platforms. These platforms are administered by a variety of concerns, and therefore have wide variation in their library support. It can be difficult merely to keep track of which libraries are installed on each platform, and even more difficult to arrange for any missing (or obsolete) libraries to be installed (or updated).

In addition, an increased acceptance of software reuse has enabled the creation of more and more sophisticated software packages, built out of numerous components obtained from diverse sources. This in turn has created a configuration management problem for users and system administrators, who must devote valuable time to obtaining and installing those components, tracking changes (including enhancements, bug fixes, and sometimes security fixes) to those components. Version conflicts, where one software package requires version X of a library while another package on the same platform requires version Y, are common.

Packages such as ATLAS [3,4] can increase the effective performance of many platforms by optimizing the parameters of the computation for that specific platform. However, for the same reason, these packages can be sensitive to specific platform characteristics (CPU features, cache sizes, etc.). These packages must therefore be configured on a per-platform basis, using fairly fine-grained attributes of each platform to identify the proper version of the library.

To enable scientists and engineers to spend more of their time thinking about their research and less time managing the quirks and intricacies of their computing tools, new approaches must be found to automate these tasks. The NetBuild tools attempt to address the problem of automatic library identification, configuration, and installation. Though these tools can be used independently of the Grid, we believe that they are well-positioned to address problems of large-scale complex distributed software systems of which the Grid is an example.

At the same time, NetBuild's goals are modest. In particular:

- NetBuild does not attempt to incorporate any knowledge about the characteristics of the data being used into the library selection process. Thus it remains the programmer's responsibility to explicitly choose between (for example) a routine that expects a dense matrix and a similar routine that expects a sparse matrix.
- NetBuild does not attempt to adapt between dissimilar calling conventions. So, for instance, NetBuild cannot generate 'glue' code to allow a Fortran program to call a C library.
- NetBuild cannot resolve fundamental incompatibilities between different libraries. However, given sufficiently detailed information about those libraries it may be able to avoid such incompatibilities, if there is a set of libraries for the target platform which are mutually compatible.

NetBuild is best understood as a set of tools to ease configuration management, by removing the need for a user to explicitly arrange that the libraries needed to perform a particular computation are installed and up-to-date.

### 1.2. Services provided

The NetBuild suite of tools provides the following services:

- automatic tagging (at compilation time) of computational software libraries with catalog information, including an indication of the target platform characteristics for which the library is suitable;

- replication of libraries to multiple servers from which the library can be downloaded;
- indexing of libraries according to targeted platform characteristics, and indexing of locations of each instance of a library;
- automated searching of libraries for candidates meeting target platform criteria;
- matching of candidate libraries against fine-grained characteristics of a target platform;
- downloading of a selected candidate library from an appropriate location;
- cryptographic verification of the authenticity and integrity of a library;
- linking of the downloaded library into an executable image; and
- run-time dynamic loading of a downloaded library into a running program.

From the perspective of the author of a scientific computing program, NetBuild provides the ability to incorporate standardized computational software libraries, at link-time or run-time, without imposing the requirement that they be pre-installed on a target platform.

### 1.3. Systems/sites/users served

We initially intend to make the most popular of the mathematical software libraries on Netlib available for use from NetBuild, with libraries pre-built for the most popular computing platforms. These libraries will be available on all Netlib mirrors. There are efforts underway to equip Netlib with facilities to allow computational software developers to compile and test their libraries on a wide range of target platforms.

Once NetBuild is more stable, we intend to make the entire suite of tools available to others, as free software, via the Internet. This will allow other developers to build NetBuild-compatible libraries, and other repositories to establish NetBuild-compatible servers.

### 1.4. Status

NetBuild currently exists as a prototype, with limited distribution to alpha testers.

## 2. ARCHITECTURE

The following tools comprise the principal components of NetBuild.

- The `netcompile` tool assists the library developer in constructing libraries that can be used by NetBuild. It performs several functions, including:
  - invoking the compiling system's compilers, linkers, and library managers with appropriate options to produce either code which is portable across a variety of target environments, or code which is optimal for a particular target environment, according to the developer's preference;
  - concurrently with compilation of an object module, producing metadata which precisely describes both the compilation environment and the target platform(s) for which the object module is presumed to be appropriate;
  - cryptographically signing the library and metadata, to allow users to verify these components for authenticity and integrity;

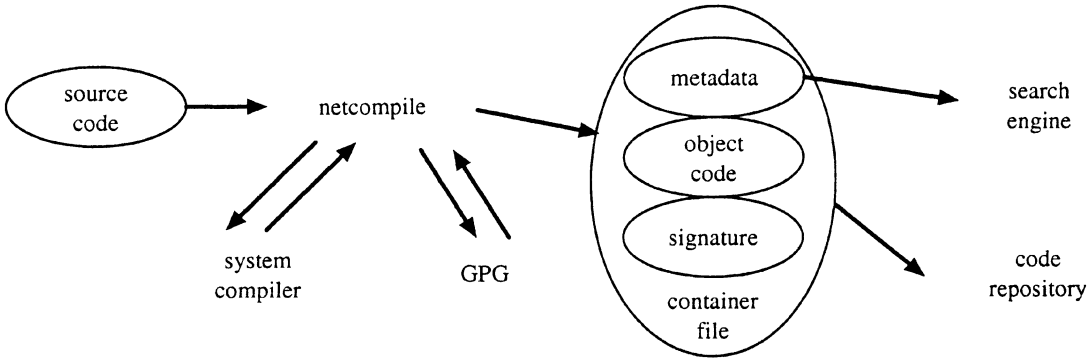



Figure 1. Operation of the `netcompile` tool.

- incorporating the results into a package from which individual components can be extracted, and which is usable by NetBuild.

Operation of `netcompile` is illustrated in Figure 1. The `netcompile` tool accepts source code from the library author, invokes the system compiler and/or linker (with appropriate options) to compile the code and produce a library, and uses GNU Privacy Guard (GPG) to generate a signature for that library. The result (along with automatically- and optional manually-generated metadata) is packaged into a container file which is placed in a (distributed) code repository. The metadata is also sent to a search engine for use by `netbuild` and NetLoader.

- The `netbuild` tool assists the computational scientist in incorporating NetBuild libraries into his or her programs. Its functions include:

  - intercepting calls to the build platform's compilers and linkers;
  - identifying libraries that are needed by the software and which are not installed on the build platform;
  - searching for instances of the missing libraries which are compatible with the build platform and appropriate for the target platform;
  - downloading those libraries to the build platform;
  - verifying their signatures for authenticity and integrity;
  - invoking the build platform's compilers and linkers with appropriate options to allow the downloaded libraries to be linked into the compiled program.

  Operation of `netbuild` is shown in Figure 2. The `netbuild` tool interprets the command-line supplied by the user (or the Makefile or compilation scripts), consults a search engine to determine locations of needed libraries which are suitable for the target platform, downloads those libraries from a code repository, verifies their signatures using GPG, and if successful, feeds those libraries to the local compiler and/or linker, which combines those with local source code and libraries to produce an executable program.

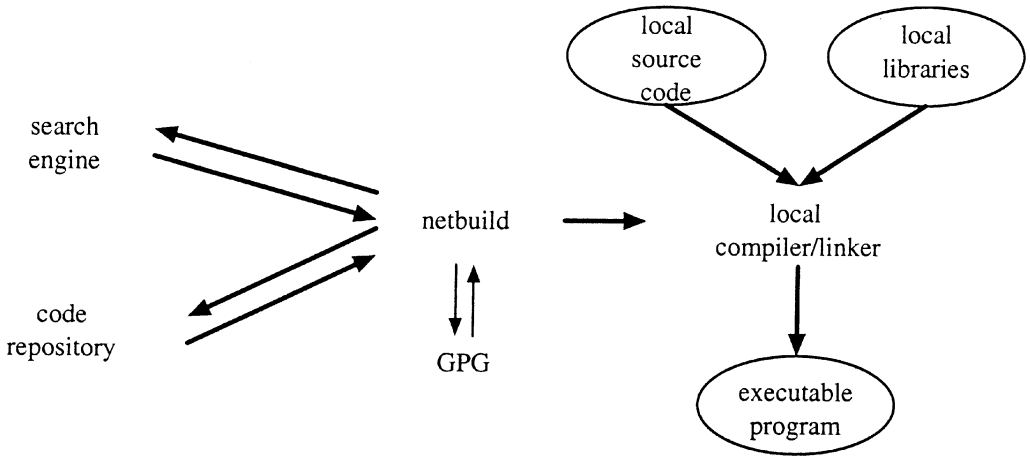



Figure 2. Operation of the `netbuild` tool.

- NetLoader is a set of library routines which are analogous to the `netbuild` tool, but which operate at run-time. Specifically, when asked to load a particular library, NetLoader:
  - searches for instances of that particular library which are appropriate for the target platform;
  - downloads the library to the target platform;
  - verifies the signature on that library for authenticity and integrity; and
  - incorporates that library into the running program.

In addition the NetBuild system includes a server which indexes object files according to their characteristics, and which answers queries from NetBuild tools and NetLoader library calls for locations of libraries matching particular characteristics. The libraries are provided using ordinary HTTP servers.

## 3. IMPLEMENTATION

### 3.1. `netbuild` tool

The `netbuild` tool may be invoked in either of two ways—explicitly, by use of the `netbuild` command, or implicitly. In the latter case the user or program invokes a compiler or linker as it normally would, but the `PATH` environment variable has been modified to contain a directory of *shims* that intercept calls to the compiler and linker. Users do not normally include the shim directory in their `PATH`s; this is done only when the compiler is some descendant subprocess of an explicit invocation of `netbuild`. This allows compilers and linkers called by `make` and other tools to be invoked in the

```
PATH=/usr/ccs/bin:/usr/local/bin:/usr/bin:/bin

/usr/ccs/bin:  /usr/local/bin:   /usr/bin:  /bin:

 as             g++
 cc             gcc
 f77            g77
 ld             netbuild
 make
```

```
PATH=/usr/local/netbuild/bin:/usr/ccs/bin:/usr/local/bin:/usr/bin:/bin

/usr/local/netbuild/bin:            /usr/ccs/bin:  /usr/local/bin:   /usr/bin:   /bin:

 as -> /usr/local/bin/netbuild       as             g++
 cc -> /usr/local/bin/netbuild       cc             gcc
 f77 -> /usr/local/bin/netbuild      f77            g77
 g++ -> /usr/local/bin/netbuild      ld             netbuild
 gcc -> /usr/local/bin/netbuild      make
 g77 -> /usr/local/bin/netbuild
 ld -> /usr/local/bin/netbuild
 make -> /usr/local/bin/netbuild
```

Figure 3. Illustration of the shim directory.

`netbuild` environment simply by typing `netbuild make`, without changes to the Makefiles or other compilation scripts.

The effect of the shims is shown in Figure 3. The top half of the figure shows a user's `PATH` and the contents of the directories in the `PATH`. (For simplicity, only the compilation tools are shown.) If, for example, the user types 'netbuild make program' the user's shell will search the `PATH`, find the `netbuild` executable in `/usr/local/bin`, and invoke it.

Since `netbuild` was invoked by the name 'netbuild', it modifies its `PATH` to look like that in the lower half of Figure 1, and then treats the remainder of the line as a command to be run with the modified `PATH`. In this example the command 'make program' would be run with the modified `PATH` variable.

`make` would then invoke the commands (specified in the Makefile or as system defaults) required to compile 'program'. However, it would search for those commands using the modified `PATH`. Since all of the program-building tools have 'shims' in `/usr/local/netbuild/bin`, any attempt to invoke a compiler or linker would actually cause `netbuild` to be invoked with the name of that compiler or linker.

`netbuild` would then parse the command-line according to the rules for that compiler or linker, identify any missing libraries, download them to the local system, and (after verification) invoke the 'real' system compiler or linker (found using the original `PATH`) to compiler and/or link the user's program.

### 3.1.1. General

`netbuild` is a C program that runs on several UNIX-derived and UNIX-like platforms. It works as follows.

- The program checks the name by which it was invoked. If that name ends in `netbuild`, it adds the directory containing `netbuild`'s shims to the `PATH` environment variable, and treats the remainder of the command line as a command to be invoked with the modified `PATH`.
- Otherwise, `netbuild` parses the command-line arguments as if it were the compiler or linker, identifying options that specify libraries to be linked.
- For each of these libraries, `netbuild` determines whether those libraries are already installed on the local system.
- For each of the libraries that are not installed, `netbuild` consults one or more network servers in an attempt to find libraries which match the characteristics of the target platform. When it finds such a library it will download it to the local system. Previously downloaded libraries are cached so they are not downloaded again if they have not changed.
- The authenticity and integrity of the libraries is verified, and if valid, the libraries are installed in local directories which are private to `netbuild`.
- The system compiler or linker is then invoked with extra options to cause the newly-downloaded libraries to be linked in along with the resident ones.

### 3.1.2. Option parsing

Since `netbuild` is invoked as if it were the normal system compiler or linker, it needs to be able to understand options that vary from one compiler or linker to another. `netbuild` therefore has a configurable parser for command-line options. The parser can be configures on a per-host, per-platform, and per-compiler basis.

`netbuild` need not understand the full syntax and semantics of each option, but it does need to know which options require additional arguments (so that subsequent arguments beginning with a hyphen are not treated as separate options), which options specify libraries to be linked, and which options specify local directories which should be searched. In the future, other extensions may be necessary. For instance, it may also need to be aware of options which specify variants of the compiler's target platform, so that it can use the correct libraries if the specified target is different from the default one.

### 3.1.3. Searching for local libraries

`netbuild` must search local directories to determine whether some of the requested libraries are already resident. Since these directories vary from one target platform to another and from one compiler to another on the same platform, the list of directories which `netbuild` consults is configurable. In addition, any directories specified on the command line are also consulted. Finally, since naming conventions vary from one platform to another, `netbuild` can be configured to understand the file naming conventions for libraries on the local platform. For instance, library 'xyz' might be matched by any of `libxyz.a`, `libxyz.so`, or `libzyx.so.1.2`.

### 3.1.4. *Identifying suitable remote libraries*

In order to search for suitable remote libraries, `netbuild` queries a search engine for libraries matching the target CPU type and operating system. A list of descriptions of candidate libraries is returned. The descriptions include a 'constraint expression', a 'preference expression', and a URI. The constraint expression is a Boolean expression, written in terms of characteristics of the target platform, which is evaluated to determine whether the candidate library can be used on the target platform. The NetBuild client may impose additional constraints, based on the characteristics of the candidate library.

In many cases (especially where performance is not critical) this is sufficient to find an appropriate match. However, if multiple libraries satisfy the constraint expressions, the preference expression of each library is evaluated (higher values are preferred) to select a library from the candidates.

The set of attributes which can be used in computing these expressions is platform-specific and extensible, but may include:

- CPU architecture family (e.g. Intel IA32);
- instruction set version (where later versions are supersets of earlier ones);
- instruction set extensions (e.g. MMX, 3DNow!);
- sizes of various caches;
- CPU chip vendor;
- CPU chip version;
- number of processing elements;
- primary memory required;
- working set size required;
- disk space required;
- operating system;
- operating system version;
- operating system features/extensions;
- compiler;
- compiler version;
- compiler ABI or calling sequence;
- source language;
- object file format (e.g. ELF, COFF, a.out).

A concrete example is shown in Table I, which illustrates constraint expressions for various versions of ATLAS libraries which were compiled for Windows NT or 2000.

### 3.1.5. *Search engine interface*

`netbuild` currently consults an external HTTP server to identify which libraries might be available for a particular platform. In order to find which versions of library 'xyz' are available, `netbuild` accesses a URL which is constructed from the library name, the target platform name, and the target operating system. Accessing that URL causes a 'common gateway interface' (CGI) program to be invoked, which returns a list of candidate libraries, including their constraint expressions, preference expressions, and URIs.

Table I. Constraint expressions for Windows ATLAS binaries.

| | |
|---|---|
| Windows NT/2000 Athlon 'classic' with 512K off-chip L2 cache | `target.arch == "ia32" && target.ia32.vendor == "AuthenticAMD" && target.ia32.family = 6 && (target.ia32.processor == "AMD-K7(tm) Processor" \|\| target.ia32.processor == AMD Athlon(tm) Processor") && target.ia32.12.size = 512*1024 && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |
| Windows NT/2000 Athlon 'enhanced' with 256K on-chip L2 cache | `target.arch=="ia32" && target.ia32.vendor=="AuthenticAMD" && target.ia32.family = 6 && (target.ia32.processor == "AMD-K7(tm) Processor" \|\| target.ia32.processor == "AMD Athlon(tm) Processor") && target.ia32.12.size = 256*1024 && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |
| Windows NT/2000 Intel PII with 512K L2 cache | `target.arch=="ia32" && target.ia32.vendor=="GenuineIntel" && target.ia32.family = 6 && target.ia32.model == 3 && target.ia32.model <= 5 && target.ia32.12.size = 512*1024 && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |
| Windows NT/2000 Intel PIII with 256K on-chip L2 cache | `target.arch=="ia32" && target.ia32.vendor=="GenuineIntel" && target.ia32.family = 6 && target.ia32.model == 7 && target.ia32.size = 256*1024 && target.ia.32.sse && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |
| Windows NT/2000 Intel Pentium Pro with 256K on-chip L2 cache | `target.arch=="ia32" && target.ia32.vendor=="GenuineIntel" && target.ia32.family == 6 && target.ia32.model == 1 && target.ia32.size = 256*1024 && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |
| Windows NT/2000 Intel P4 with 256K on-chip L2 cache, using SSE instructions | `target.arch=="ia32" && target.ia32.vendor=="GenuineIntel" && target.ia32.family = 15 && target.ia32.12.size = 256*1024 && target.ia.32.sse2 && (compiler=="gcc" \|\| compiler=="ms.vc++" \|\| compiler=="compaq.vf")` |

We used HTTP and CGI both for ease of prototyping and because most sites allow access to external HTTP servers and through their firewalls. However, the search engine is not inherently centralized, and the NetBuild client can choose from multiple search engines if they are advertised either in the domain name system or via HTTP redirects.

### *3.1.6. Library container file format*

`netbuild` expects downloaded libraries to be in a NetBuild-specific container format with multiple components. The actual library archive is one component, the metadata is another, and the signature on the library and metadata are yet another component. On some platforms, the archive may also contain

other components. After downloading the container file the archive is extracted, renamed as necessary, verified against its signature, and copied to the cache directory.

#### *3.1.7. Caching*

`netbuild` caches files that are downloaded from the network so that they are not downloaded again unless necessary. The cache is currently maintained on a per-user basis, in each user's private filestore, due to security concerns associated with maintaining a shared cache. Libraries downloaded from servers are stored in a directory whose name is derived from a hash of the (canonicalized) URL from which the library was obtained: a separate metadata file contains the last-modified data of that URL. Subsequent attempt to download that file use the HTTP 'if-modified-since' directive which causes the file to be downloaded only if it has been changed. Note that the last change date of the container file which is downloaded may be different than the last change date of the actual library; thus it is possible for the metadata (and signature) to be updated even though the object library is unchanged. We can use this feature, for example, to change the constraints on use of a library to reflect experience with the library after its publication.

#### *3.1.8. Authenticity and integrity verification*

We currently use GNU Privacy Guard (GPG) [5] to verify digital signatures on `netbuild` libraries. Compatible signatures can be created with GPG or any of several PGP variants. Because the trust model for `netbuild` libraries is different from that of normal PGP signatures (just because a signature on a library is trusted to be authentic does not mean that it is safe to execute code from that library on a computer), the signatures used by `netbuild` are kept on a separate key ring in a separate directory.

GPG is used in `netbuild` prototypes because it is easy to interface to, portable, readily available, and presumably free of patent issues. However, the current implementation requires that GPG be installed in addition to `netbuild`. To make it easier for the user to install `netbuild`, it would be preferable for the signature verification code to be incorporated directly. This would allow `netbuild` to support additional signature formats, and additional certificate formats such as X.509v3.

### 3.2. `netcompile`

The operation of `netcompile` is similar to `netbuild`, but the function is different. Instead of supplying missing libraries, `netcompile`'s job is to invoke the native compilers and library managers in such a way that:

- the compiler is optimally tuned for the assumed target environment, or to produce portable code;
- the characteristics of the compiler and the resulting object files are recorded;
- the characteristics of the assumed target are recorded;
- the resulting library and metadata are signed; and
- the result is packaged for easy downloading by `netbuild`.

Whereas `netbuild` tries to transparently preserve the native interface to each compiler or linker, `netcompile` tries to provide a platform- and compiler-independent interface to compilers, linkers, and library managers.

### 3.3. NetLoader

NetLoader consists of an application programmer's interface to the portions of `netbuild` which download and verify code, along with a dynamic library loader. However, the matching algorithms are subtly different than those used by `netbuild`. For instance, NetLoader can only use dynamically-loadable libraries, while `netbuild` can also use static libraries. Another difference is due to target platforms that support emulation of the operating environment for non-native images—for instance, a NetBSD platform can often run an executable image compiled for Linux. In this case it is necessary to load a library that is compatible with the emulated environment, rather than one which is comparable with the target's native environment. On the other hand, if `netbuild` is asked to install a particular executable image, it might substitute one intended for a different platform if the target platform supports emulation of the other platform and no native executable is available.

#### *3.3.1. Distribution and location of libraries*

The distribution system takes advantage of the fact that Netlib is already extensively mirrored. Each distinct version of a library is assigned a unique (and cryptic) filename which will never be reused, so if a file with the same name appears on a mirror site, it is very likely to be an identical copy of that library. The `netbuild` servers will maintain a list of mirror sites and the directories on which they store Netlib files, to use in determining the probable location of the library on the mirror. The servers initially will use heuristics (such as the last-changed-date of the library) to guess whether a particular mirror site is likely to have a copy of a particular library. (If the client does not find a copy of the library at that mirror, or if the library is truncated or altered, the client with try to obtain the library from another mirror.)

We plan to periodically check Netlib mirror sites for integrity and currency of mirroring, and update our location database accordingly.

#### *3.3.2. Related work*

NetBuild bears a resemblance to several other works.

The NetLink project [6] has similar goals to those of NetBuild, in that it is also focused on easing the burden of maintaining software libraries used in scientific computing. Their stated objective is to 'identify a data distribution architecture [. . . ] that can help to centralize the library maintenance and tuning'. Their work appears to be focused on identifying an appropriate data distribution architecture, whereas our work to date has focused on making NetBuild as transparent as possible, and effective matching of object files with target platform characteristics. We intend to actively cooperate with the NetLink project as part of our continued work on NetBuild.

Libtool [7] is a package which eases the burden of development of software for multiple computing environments, by providing a uniform interface to a variety of compilers, linkers, and library archivers. Libtool is similar to `netbuild` and `netcompile` in that both sets of tools run the existing compilers, linkers, etc. within a wrapper. Libtool allows developers to write scripts and makefiles which invoke compilers and linkers via the `libtool` command; these scripts and makefiles will be portable to every platform supported by Libtool. Libtool also includes the ability to dynamically load libraries

created by Libtool. Libtool has been ported to a wide variety of platforms. Our initial implementation of `netcompile` uses Libtool as a platform-independent means of generating shared libraries.

The NetBSD operating system [8] contains a 'package' facility [9] which automates the process of downloading source code for a particular program or library, verifying its integrity, configuring it to run on NetBSD, compiling it, installing it, and cataloguing it for configuration management purposes. The package facility also handles dependencies, so that any components needed by the component to be installed are automatically compiled and installed also. The facility supports both source-code packages and pre-compiled binary packages, and has successfully been used to ease installation of over two thousand different tools and libraries. The package facility has been ported to each of the several hardware platforms supported by NetBSD, as well as to the Solaris operating system. The NetBSD package system is itself descended from a similar 'ports' system incorporated into FreeBSD; similar but independently-derived features exist in Debian and RedHat Linux.

## 4. STATUS AND FUTURE PLANS

The NetBuild tools currently exist as prototypes. We are currently experimenting with the data model for platform and compiler characteristics, and are working on a more automatic derivation of these characteristics at library build time. Our next step will be to construct a 'compile and test zoo' for the purpose of automatically building and testing large numbers of libraries on various platforms.

The current implementation is intended as a proof-of-concept and a testbed for new features rather than a code base for use by ordinary users. As such, it is designed to be flexible and configurable and easily implemented, rather than (say) secure and robust. The publicly-distributed version of NetBuild will need to pay much more attention to security and robustness issues.

### REFERENCES

1. Browne S, Dongarra J, Grosse E, Rowan T. The Netlib Mathematical Software Repository. *D-Lib Magazine*. http://www.dlib.org/dlib/september95/netlib/09browne/html [September 1995].
2. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman: San Francisco, 1999.
3. Whaley RC, Dongarra J. Automatically tuned linear algebra software. *Proceedings Supercomputing 1998 Conference*. http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Whaley814/INDEX.HTM [1998].
4. Whaley R, Petitet A, Dongarra J. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing* 2001; **27**(1–2):3–25.
5. Gnu Privacy Guard. http://www.gnupg.org/.
6. Holmqvist I, Lindström E. NetLink: A modern data distribution approach applied to transparent access of high performance software libraries. *Applied Parallel Computing: Large Scale Scientific and Industrial Problems* (*Lecture Notes in Computer Science*, vol. 1541), Kågström *et al*. (eds.). Springer: Berlin, 1998; 248–254.
7. GNU Libtool Web site and documentation. Free Software Foundation. http://www.gnu.org/software/libtool/libtool.html [2001].
8. NetBSD. http://www.netbsd.org/.
9. Feyrer H, Crooks A. Documentation on the NetBSD package system. ftp://ftp.netbsd.org/pub/NetBSD/package/pkgsrc/Packages.txt.