

Identifying Software Usage at HPC Centers with the Automatic Library Tracking Database

Bilel Hadri, Mark Fahey and Nick Jones

National Institute for Computational Sciences

Oak Ridge National Laboratory,

Building 5100, P.O. Box 2008 MS6173,

Oak Ridge, TN 37831-6173

(+1) 865-241-5471

bhadri@utk.edu, mfahey@utk.edu, njones11@eecs.utk.edu

Abstract

A library tracking database has been developed to monitor software/library usage. This Automatic Library Tracking Database (ALTD) automatically and transparently stores, into a database, information about the libraries linked into an application at compilation time and also the executables launched in a batch job. Information gathered into the database can then be mined to provide reports. Analyzing the results from the data collected will help to identify, for example, the most frequently used and the least used libraries and codes, and those users that are using deprecated libraries or applications. We will illustrate the usage of libraries and executables on the Cray XT platforms hosted at the National Institute for Computational Sciences and the Oak Ridge Leadership Computing Facility (both located at Oak Ridge National Laboratory).

Categories and Subject Descriptors

K.6.3 [Management of Computing and Information Systems]: Software Management - Software maintenance

General Terms

Management

Keywords

Library, Tracking, Database, Cray XT, most/least used software.

1. INTRODUCTION and MOTIVATION

During the Third Workshop on HPC Best Practices organized by SciDAC [19] in September 2009, the scientific community reflected primarily on identifying best practices for maintaining reliable and sustainable software for use at High Performance Computing (HPC) centers and one issue of particular interest was “How do HPC centers monitor/measure software usage and forecast needs?”. Furthermore, national agencies such as the Department of Energy and the National Science Foundation occasionally request reports on library and application usage on HPC systems, with particular interest in software that was funded by one of their initiatives.

In an attempt to better understand library usage at the National Institute for Computational Sciences (NICS) and the Oak Ridge Leadership Computing Facility (OLCF) and address the needs above, an Automatic Library Tracking Database (ALTD) was

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

TeraGrid’10, August 2-5, 2010, Pittsburgh, PA, USA.

Copyright 2010 ACM 978-1-60558-818-6/10/08...\$10.00.

developed and put into production on some HPC systems located at Oak Ridge National Laboratory. ALTD transparently tracks linkage and execution information for applications that are compiled and executed on the NICS and OLCF HPC systems. Supercomputing centers like NICS and OLCF maintain a collection of program libraries and software packages to support HPC activities across diverse scientific disciplines and it behooves these centers to know which and how many users utilize these libraries and applications. With the costs associated in maintaining leadership computing systems, it is important to identify little-used software to save on support costs. A particularly good example of this situation is the two Cray XT5s located at NICS and the OLCF, Kraken and JaguarPF. On these systems, the staff supports more than a hundred software packages, each with multiple versions. Over time, upgrading and removing older versions of libraries is a necessary task. Without a database like ALTD, application support staff has to make these decisions based on surveys or personal knowledge. And inevitably, these decisions are based on incomplete data and therefore staff must be conservative when deprecating and/or changing default software versions.

In addition, ALTD stores information about compilation and execution into a database that can be mined to provide reports. For example, ALTD can generate data on the most or least used library with valuable details such as a version number. This database will help the application support staff in their decision process to upgrade, deprecate, or remove libraries and thusly provide higher quality service to their users. It will also provide the ability to identify users that are still linking against deprecated libraries or using libraries or compilers that are determined to have bugs. Tracking the usage of software allows for better quality and more efficient user support.

This paper is organized as follows. Section 2 gives a background and the related work on tracking software. Section 3 provides an overview on the design, the methodology and implementation of ALTD. Section 4 presents some results from early data mining efforts including the most used libraries on different Cray XT platforms and, finally, Section 5 summarizes the project and its future development.

2. RELATED WORK

ALTD transparently tracks libraries that are used at link time and execution time. To the best of our knowledge, no tool has been developed that addresses the objectives presented in the previous section and detailed in the Section 3. There are other approaches that can be considered, but all have major shortcomings and we briefly discuss a few alternative methods in this section.

One approach to mention is that of adding logging functionality to existing libraries. Since some libraries provide an “init” function (like in PETSc [3]), the init function could be modified to log information into a tracking database. Since the “init” function always has to be called, only this function would require modification. This would be fairly straightforward; however each version of the library would have to be modified similarly over time. This becomes a maintenance issue. Furthermore, there is no simple way to track libraries that don’t have an “init” function (like BLAS or LAPACK [1]) because code would have to be inserted into each and every routine just to know if LAPACK was used at all. This is not a reasonable solution even though it was considered as one option during the SciDAC meeting [20].

Another possibility is to use state of the art profiling and tracing tools such as CrayPAT [6], TAU [13], IPM [4] and Vampir [21] that perform analysis for all the function calls in an application. These tools could provide the desired information as a by-product (by adding additional infrastructure to identify what libraries a function is part of), but they are heavy-weight and introduce compile-time and runtime overheads that should not affect every user all the time.

Also PAPI [15], a well known performance tool, has a function `PAPI_get_shared_lib_info` which returns information such as the path, and name for the shared libraries used by the program. Nevertheless, this function is available only for programs written in C [27] and does not take in consideration the static libraries.

In Linux, thanks to commands such as `lastcomm` [23], which prints out previously executed commands; administrators can parse the output to retrieve summaries on software usage. This method is called process accounting, which records accounting information for the system resources used. However, turning on process accounting requires significant disk space due to the large amount of logs generated.

Some commercial or open source distributed resources manager tools such as TORQUE [22] can be used to monitor supercomputing systems, however they can only observe the number of time an executable has been run or how many CPU hours have been used by an application. Both TORQUE and process accounting commands report only the applications called in a batch environment and from job scripts respectively; however they cannot track library usage or even when executables are nested in scripts while ALTD can track every job launched and every library linked.

We mention the work on TOPAS [14], a tool that automatically monitors the usage and performance on the CRAY T3E by modifying the UNICOS/mk compiler wrapper script. It collects data like the number of operations, the L1 cache misses and unfortunately the usage of only two types of libraries (the programming language and the message passing library). Any other library would not be detected.

One other tracking mechanism that could be implemented is to log environment programming management loads and unloads. In this way, any time a library is loaded would imply that the library was linked into a user program. However, that implication is not necessarily true, and for instance, it is unknown into what executable the library was linked.

3. OVERVIEW ON ALTD

For the initial release of ALTD, the Cray XT architecture is the target machine for tracking library usage. Future releases will support additional HPC architectures. In the following, we present an overview on ALTD.

3.1 Key concepts

A primary objective of ALTD was a lightweight solution with essentially no overhead at compilation and runtime. Our solution is based on intercepting both the GNU linker (`ld`) [10] to get the linkage information and the job launcher (`aprun`) [2], since we only consider Cray machines. Wrapping the linker and the job launcher through scripts is a simple and efficient way to obtain the information automatically and transparently. ALTD is able to track static and shared libraries, however, libraries that are loaded and unloaded at runtime such as the dynamically linked libraries, are not tracked since ALTD retrieves information during the linking process. Nevertheless, we are considering the tracking of dynamically libraries for future development of ALTD.

The objective of ALTD is to track only libraries linked into the applications. Tracking function calls could easily be done using profiling technologies, but at the cost of significantly increased compile time and application runtime.

In addition, since most of scientific libraries are constantly improved and upgraded, getting the version number is highly desired, especially when several versions are installed in the same system. The version information is not a direct result of ALTD, but rather how libraries are installed and then made available to users. For example, NICS and OLCF use modulefiles to provide environment variables with paths to libraries and applications that then appear on the linkline, which ALTD then stores in the database. When a library is linked to an application, the whole path is intercepted that contains both the name of the library and the version.

3.2 Implementation

In the following, we describe the linker (`ld`) and the job launcher (`aprun`) wrappers.

3.2.1 Linker

Our custom wrapper for the linker (`ld`) intercepts the user linkline and parses the command line to capture the linkline. Because more libraries are included on the linkline than are actually used, we go through a 2-step process to identify the libraries actually linked into the executable while at the same time including an ELF section header in the user’s code. Thus, at a high level, two main steps are done.

The `ld` wrapper first generates a record in the `link_tags` table (see example in Figure1.b) with an auto-incremented `tag_id` during this step. As shown in the Figure1.b for the user2, if there is a failure in the compilation process, the table is filed with the exit code (-1) and the `linkline_id` is set to 0. Moreover, in case that the compilation line is the same as the previous one, the `linkline_id` is not incremented and it will refer to the `linkline_id` of a previous command. This is the case for the `tag_id` 91132, where the user user1 performed the same linking process consecutively.

During this phase, assembly code is generated, compiled and stored in the section header of the user’s executable. The assembly code contains four fields – version number of ALTD, build machine (*machine*), tag id (*tag_id*), the year (*year*). The build machine and tag id are two pieces of information that are necessary to be able to accurately track the executable in the jobs table back to the correct machine `link_tags` table. After generating the assembly code, a bash script is called again and checks the `tag_id`. If `tag_id` is positive, then all the files will be compiled and linking is performed, otherwise ALTD exits.

The second step consists in inserting or updating the *linkline*. A query on the linkline table with the *linkline* is performed. In case

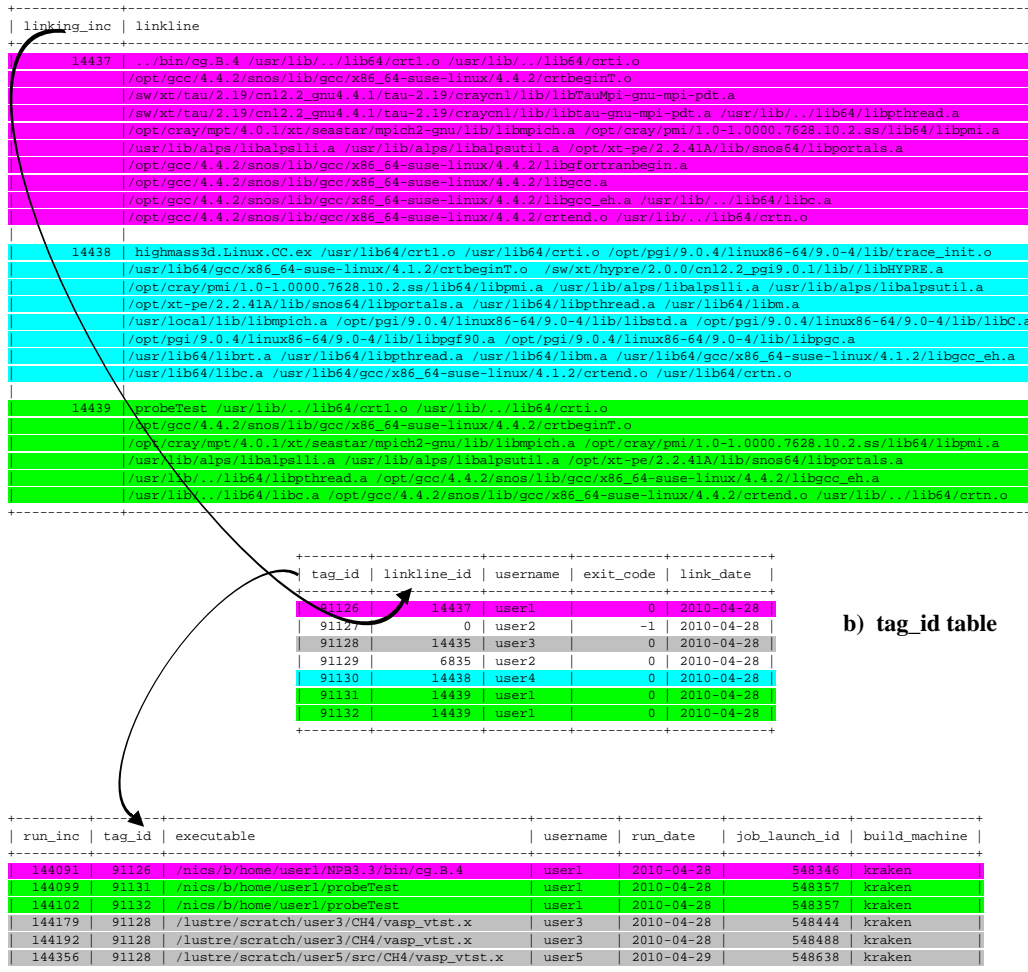


Figure 1. ALTD tables.

of positive answer, meaning that the linkline has been referenced before, the *linkline_id* is then updated with the first reference (as illustrated in Figure 1.b with the user user2 where the *linkline_id* is smaller than the previous compilation). Otherwise, we insert a new linkline and retrieve the *linking_inc* (which is the same as the query from linkline). Either way once the *linking_inc* has been retrieved, the *linkline_id* is updated in the tags table. Figure 1.a shows the linkline table obtained when loading the MySQL ALTD database. This link table corresponds to the *link_tags* table obtained in the Figure 1.b.

3.2.2 Job launcher

Launching parallel jobs on compute nodes is typically done through a parallel job launcher such as mpirun, mpiexec or aprun and often within a batch system (like PBS or LoadLeveler). Interactive support for parallel jobs is often limited if even available. We intercept the job launcher as a secondary measure of “library usage” by counting how many times an executable is run and thus in turn how many times the libraries are used by linking the jobs table data back to the linkline table. This is different than counting the number of times a library was used in a linkline. On the Cray XT systems, the job launcher is “aprun” and must be used within a PBS (TORQUE) job to run compiled

applications across one or more compute nodes. On both Kraken and JaguarPF, the aprun job launcher was already “wrapped” before ALTD was deployed. To work with the wrappers that were already in place, an “aprun-prologue” script was implemented that the aprun wrapper calls, which then does the ALTD tracking. The aprun-prologue extracts information on PBS environment variables such as the working directory (PBS_O_WORKDIR) and the job id (PBS_JOBID). Then, the command *objdump* is run on the executable to display the information that has been stored in the section header of the user’s executable during the linkage process. Finally, the extracted information is then inserted in the jobs database, and then control is passed back to the aprun wrapper that eventually calls the real aprun. Figure 1.c shows example final output in the jobs database. For more details on ALTD, we refer to the manual [7].

4. DATABASE ANALYSIS

As described in the previous section and shown in the Figure 1, the database gathers three tables, (two to store link information and one to store job information.). Moreover, Figure 1 shows the relationship between the two tables, as indicated by the arrows. In addition, the color of each row in the tables clearly illustrates the relationships. For example, in purple, user1 successfully compiled

his code as shown in the *tag_id* table (Figure 1.b) and the *tag_id* is assigned to 91126. In the *linkline* table, (Figure 1.a), in purple, the *linkline* of the user is shown, and notice (in the third row) that the user has used the TAU library. As explained in the previous section, we retrieved the whole path, which enables us to get the version of the software. In our example, user1 used version 2.19 of TAU. The *linking_inc* index is incremented to 14437 and in the *tag_id* table, the *linking_id* index is updated to the same value. In addition, the *linkline* table gathers the name of the executable (here *cg.B.4*) and in the *job_table* (Figure 1.c, row in purple), the executable was launched in a batch job and a *run_inc* index has been incremented and associated to the *tag_id* 91126.

4.1 Data Mining

ALTD is in production on several Cray XT systems, and in this paper, we show the benefits of having such a database installed on two Cray XT5 systems, JaguarPF and Kraken, ranked respectively 1st and 3rd in the TOP500 [24] in November 2009. A lot of data has been collected on these systems; and to transform this data into valuable information, it is essential to perform data mining on the usage of the libraries

4.1.1 How to generate the reports: Data mining

A primary task is to retrieve and parse the data from the *linkline* table (Figure 1.a). This table gathers the desired information; however, a *linkline* for a given code compiled is stored as a single line and it has several paths for the libraries linked into the code. Usually, a database contains only one data item per row and per variable (or column) and the mining operation is straightforward. To overcome this difficulty we fetch all the data from the *linkline* table for a given time frame with a python script, and we create a list of words separated by the whitespaces (this is how the path are separated in the database as shown in Figure 1.a).

/usr/lib64/libpthread.a	12249
/usr/lib64/gcc/x86_64-suse-linux/4.1.2/libgcc_eh.a	8337
/usr/lib64/crti.o	8207
/usr/lib64/crt1.o	8144
/usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtend.o	7850
/usr/lib64/libc.a	7548
/usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtbegin.o	7416
/usr/lib64/libm.a	6221
/usr/lib/./lib64/crti.o	5557
/opt/xt-pe/2.2.41a/lib/snos64/libportals.a	5463
/usr/lib/./lib64/crtn.o	5129
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/trace_init.o	4551
/opt/cray/pmi/1.0-1.0000.7628.10.2.ss/lib64/libpmi.a	4455
/usr/lib/alps/libalpslli.a	4445
/usr/lib/alps/libalpsutil.a	4444
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgc.a	4150
/usr/lib64/librt.a	4026
/usr/lib/./lib64/libpthread.a	3547
/usr/lib/./lib64/libc.a	3427
/opt/cray/mpt/4.0.1/xt/seatstar/mpich2-pgi/lib/libmpich.a	3324
/lib64/libc.so.6	3248
/lib64/ld-linux-x86-64.so.2	3248
/usr/lib64/libc_nonshared.a	2305
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgftnrtl.a	2087
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgf90.a	2051
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/libnspgc.a	1761
/opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgf90rtl.a	1671
/opt/xt-libsci/10.4.1/pgi/lib/libsci_istanbul.a	1508
/opt/xt-tools/craypat/5.0.0/cpatx/lib/lib_pat.a	1447

Figure 2: Frequency of the path

Then, we compute the frequency of each word in the list and, finally, the results of most frequent path are sorted and stored into a large file that can contains around forty thousand lines for three months of data. Figure 2 lists the name of the libraries found and their frequency. As expected, libraries associated with the parallel runtime (threads and portals), necessary objects files and compiler libraries are the most used libraries. This large report has too many details and does not really provide the packages that are used, but more specifically libraries and/or files from those packages.

/opt/xt-libsci/10.4.1/pgi/lib/libsci_istanbul.a	1508
/opt/xt-libsci/10.4.1/pgi/lib/libsci.a	192
/opt/xt-libsci/10.3.9/pgi/lib/libsci_istanbul.a	180
/opt/xt-libsci/10.3.8/pgi/lib/libsci.a	77
/opt/xt-libsci/10.4.1/gnu/lib/44/libsci_quadcore.a	50
/opt/xt-libsci/10.3.9/pgi/lib/libsci_quadcore.a	44
/opt/xt-libsci/10.4.1/gnu/lib/44/libsci_istanbul.a	34
/opt/xt-libsci/10.4.1/pgi/lib/libsci_quadcore.a	12
/opt/xt-libsci/10.3.9/pgi/lib/libsci.a	8
/opt/xt-libsci/10.4.1/gnu/lib/44/libsci_istanbul_mp.a	7
/opt/xt-libsci/10.4.1/pgi/lib/libsci_istanbul_mp.a	4
/opt/xt-libsci/10.3.8.1/pgi/lib/libsci_istanbul.a	3
/opt/xt-libsci/10.4.1/gnu/lib/44/libsci.so	1
/opt/xt-libsci/10.3.9/gnu/lib/44/libsci.so	1

Figure 3: Frequency Libsci variants on Kraken

A filtering process is needed. For example, if we extract the specific libraries associated with libsci (a collection of single-processor and parallel numerical routines tuned for performance on Cray), we obtain the results shown in Figure 3 for Kraken. We notice that several versions of the package are used (10.4.1, 10.3.9, 10.3.8 etc...). Our goal is to get the total number of uses of a library and the associated versions. Two reference lists have been produced containing respectively, only the name of the software and a more detailed one listing the software and its versions. Then, a search of the report produced previously is performed with the reference lists and the outputs are sorted by the frequency of usage.

4.2 Reports on libraries used during linking

The data collected for JaguarPF are since January 2009 (ALTD was in production with a C version) while the data on Kraken were gathered only since February 2010, when ALTD was ported in python.

In the following, we show the results obtained. For a briefer analysis, we do not report the usage of compilers and parallel runtime support libraries, which would appear the most.

4.2.1 Overall usage

Table 1 shows the library usage during linking for JaguarPF in 2009 and the first 4 months in 2010 (from January 1st to April 30th) and the results from Kraken in 2010 (from February 2010 until April 30th.) This table shows the top 10 libraries used in the systems considered. This report considers all libraries either provided by Cray (in /opt), installed locally by center staff, or installed by users (except for compiler and parallel runtime libraries.) For these top10 most used libraries, we notice that on both Kraken and JaguarPF, the software falls into three major categories; the numerical libraries shown in red, the performance analysis tools colored in blue and the I/O software in green.

Table 1. Library usage during linking on JaguarPF in 2009, in 2010 and on Kraken in 2010.

Rank	Jaguar 2009	usage	Jaguar 2010	usage	Kraken 2010	usage
1	hdf5	7744	fftw	6025	craypat	4002
2	craypat	6751	Hdf5	5370	libsci	1874
3	papi	5252	petsc	2654	charm	1172
4	petsc	5161	papi	2399	petsc	1066
5	libsci	3416	netcdf	2308	fftw	955
6	netcdf	2608	acml	2243	Hdf5	861
7	acml	2228	libsci	1748	netcdf	672
8	fftw	2041	craypat	1598	sprng	661
9	gromacs	1453	tau	1001	papi	463
10	trilinos	1353	trilinos	960	Hdf4	352

On JaguarPF during 2009, the HDF5 (Hierarchical Data Format) [9] package is the most used library, followed by two libraries that are part of performance analysis tools, CrayPAT (Cray’s Profiling and Analysis Tools) and PAPI. The next five most used packages are numerical libraries, PETSc, libsci, ACML (AMD Core Math Library, a set of numerical routines tuned specifically for AMD64 platform processors), FFTW (Fastest Fourier Transform in the West) [8] and Trilinos [25] (a suite of parallel object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems). In between the math libraries, we notice the presence of another I/O library NetCDF (Network Common Data Form) [18] ranked at the sixth position and GROMACS [26] (a versatile package to perform molecular dynamic) is the 9th most used package on JaguarPF during 2009.

In the first quarter of 2010 on JaguarPF, FFTW is the most used package and the ranking is again dominated by the numerical libraries since half of the ten most packages belongs to this category. This shows an extreme increase in usage of FFTW since it has been linked almost three more times than all of 2009 on the same system. If we look closer at this category, we notice that the libsci Cray optimized library is not the most used numerical software, it is exceeded not only by FFTW, but also PETSc and even ACML. This is quite interesting because libsci supports a multitude of routines tuned for the Cray XT systems, such as FFT, dense linear algebra function, sparse solvers. CrayPAT is the most used profiling tool in 2009, and it is closely followed by TAU. Overall, if the results are interpolated over the year, we get quite close results for the performance and profiling tools, however for the rest, it shows a larger utilization of libraries, especially the numerical variety. This can be explained by the increase of users on JaguarPF recently since more projects have been granted allocations to run on the world’s fastest computer with a theoretical peak performance of 2.33 PetaFLOPS.

Regarding Kraken, the usage in 2010 shows smaller numbers since ALTD was only in production since February (only three months of data have been collected) and the usage by staff is not considered. Still, CrayPAT is by far the most used package, taking the lead by almost two times more than libsci ranked in second place. In the same category, PAPI is also one of the most used, ranked in the 9th position. The numerical libraries do not dominate

the top 10, only three packages are present (libsci, PETSc and FFTW). At the third position, it is interesting to notice the presence of CHARM (an object oriented parallel language). As noticed in JaguarPF, the libraries associated to I/O such as HDF5, NetCDF and HDF4 are part of the top 10 ranking. Finally, we notice the high usage of, SPRNG [12] (Scalable Parallel Random Number Generators Library, a set of libraries for scalable and portable pseudorandom number generation). The usage of SPRNG on Kraken is entirely by one project doing 3D modeling of jet noise. The tracking database clearly shows them doing many compilations with and without profiling tools, an indication of a development cycle likely doing optimization/scalability work.

As shown in Figure 3, there are several versions of a package that the users link with. In the following, we analyze the results reporting the version of the package used in the linked and installed by the vendor in the directory /opt.

4.2.2 Usage of libraries installed by the vendor

Table 2 reports the number of libraries used during linking with version numbers for JaguarPF in 2009, 2010 and Kraken in 2010. Table 2 reports only the versions installed by the system vendor in the directory /opt. For JaguarPF in 2009, the high usage is dominated by the performance profiling tool CrayPAT with two versions highly used in 2009 and PAPI. Versions 4.4.0.4 and 4.1.1 cover almost 60% of the total usage of CrayPAT as reported in Table 1. The rest is divided among other version (older ones like 4.3.0 and 4.4.0; and a newer version 5.0, ranked outside the top 10). PAPI/3.6.2 is the second most used library from /opt in 2009, however, it accounts only for a third of the total usage of the PAPI as shown in Table 1. Like CrayPAT, several version of PAPI have been used, such as PAPI/3.6.2, but there is a noticeable difference since some users installed PAPI in their home directory. In addition, some libraries include PAPI, such as CrayPAT and TAU for getting information about hardware counters.

Table 2. Library usage (installed in opt/) during linking for JaguarPF in 2009, in 2010 and on Kraken in 2010

Rank	Jaguar 2009	usage	Jaguar 2010	usage	Kraken 2010	usage
1	craypat/ 4.4.0.4	2142	fftw/ 3.2.2.1	5452	craypat/ 5.0.0	3102
2	papi/ 3.6.2	1709	acml/ 4.3.0	1558	libsci/ 10.4.1	1674
3	craypat/ 4.1.1	1687	craypat/ 5.0.1	1096	petsc/ 3.0.0.9	319
4	acml/ 4.2.0	1076	libsci/ 10.4.1	890	fftw/ 3.2.2.1	253
5	hdf5/ 1.8.2.2	1042	petsc/ 3.0.0.8	859	hdf5/ 1.8.3.0	231
6	libsci/ 10.3.1	1030	papi/ 3.6.2.2	848	netcdf/ 3.6.2	214
7	petsc/ 3.0.0.4	912	hdf5- paralle /1.8.3.1	799	fftw/ 2.1.5	170
8	petsc/ 3.0.0	812	hdf5/ 1.8.3.1	696	netcdf/ 4.0.1.2	136
9	libsci/ 10.4.1	779	libsci/ 10.4.0	630	netcdf/ 4.0.1.0	136
10	libsci/ 10.3.8	777	petsc/ 3.0.0.4	433	hdf5/ 1.8.4.0	136

Besides the performance profiling libraries, six different versions of numerical libraries are part of the top 10. ACML/4.2.0 dominates the numerical libraries, (since it was the default version for a period of time and the rest of the total usage of ACML is divided between the versions 4.1 and 4.3) while the usage of

PETSc and libsci are split into several versions (up to thirteen different versions of libsci are available and only three version have been linked more than one hundred times, and seven different version have been used at most ten times). The same reason accounts for the drop of the I/O library HDF5, ranked only fifth with version 1.8.2.2, while it is the most used library as shown in Table 1.

In 2010, the numerical libraries dominate Table 2 by taking the two tops places. FFTW/3.2.2.1 has been heavily used during this four-month period and almost 90% of the total usage of the FFTW library is linked with the version 3.2.2.1. In second place, we have ACML/4.3.0 and its usage dominates largely the other mathematical libraries such as libsci and PETSc with two versions ranked in the top 10 (libsci/10.4.1 and 10.4.1; PETSc/3.0.0.8 and 3.0.0.4). For the rest, the I/O libraries are mostly used, namely HDF5/1.8.3.1 with two different implementations: one parallel and one sequential.

Comparing the data collected in 2009, and the first quarter of 2010, we observe that the newer versions are being extensively used and older versions disappearing from the top 10. For example, libsci 10.3.1 is not used anymore in 2010, while in 2009, it was highly used. This is explained by the availability of several new versions. libsci/10.3.8, ranked at the top 10 in 2009, is still being used by few users while 4 others new versions have been installed. This is a good example where application support staff can take action to depreciate older versions of libsci, and remove them from support. Regarding the usage on Kraken, CrayPAT/5.0.0 is the most used package. Then, the rest of the top 10 is dominated by numerical libraries followed by I/O libraries with the highest usage of both HDF5 and NetCDF with several versions. In addition, we notice that the libsci/10.4.1 is linked more on Kraken than on JaguarPF in 2010. Table 3 shows that ACML is generally more used than libsci on JaguarPF, while on Kraken, the use of ACML is not significant.

4.2.3 Usage of the libraries installed in /sw/xt

Table 3. Library usage (installed in /sw/xt*) during linking for JaguarPF in 2009, in 2010 and on Kraken in 2010

Rank	Jaguar 2009	usage	Jaguar 2010	usage	Kraken 2010	usage
1	szip/2.1	1214	tau/2.19	638	sprng/2.0b	661
2	hdf5/1.6.8	595	szip/2.1	591	petsc/2.3.3	586
3	trilinos/9.0.2	436	hdf5/1.8.1	497	iobuf/beta	340
4	pspline/1.0	370	hdf5/1.6.8	260	tau/2.19	55
5	netcdf/3.6.2	365	trilinos/1.0.4	225	szip/2.1	43
6	gromacs/4.0.5	344	vampirtrace/5.8	182	p-netcdf/1.1.1	40
7	parmetis/3.1	238	hdf5/1.6.7	160	ncl/5.0.0	30
8	petsc/3.0.0	238	Adios/1.1.0	124	atlas/3.8.3	25
9	hdf5/1.6.7	236	fpmpi/1.1	97	upc/2.8.0	24
10	hdf5/1.8.2	191	p-netcdf/1.0.3	93	hdf5/1.8.3	23

As previously mentioned Cray installs software in /opt. The application support staff needs to install and maintain a collection of program libraries and software packages to support HPC

activities across diverse scientific disciplines. These packages are installed in the directory /sw/xt5 for JaguarPF and /sw/xt for Kraken. Table 3 reports the results of the data mining for software in /sw/xt*. Overall, we observe that I/O libraries are the most used. Then, we detect new numerical libraries not shown in the previous reports (PSPLINE and PARMETIS) on JaguarPF in 2009. Also, in 2010, we identify recent performance profiling tools such as TAU, VAMPIRTrace and FPMPI, nevertheless, the usage is relatively small compared to CrayPAT and PAPI. Kraken has a very low usage, except three packages that are used more than 300 times, the rest of the top 10 libraries have been linked less than sixty times. Most of the packages on Kraken are used by a specific group of people working in the same project. We observe also on Kraken that some user develop their codes using specific tools such as UPC (Unified Parallel C an extension of the C programming language designed for high performance computing on large-scale parallel machines) and NCL (NCAR Command Language, an interpreted language designed specifically for scientific data processing and visualization.). In the same context, even though Table 1 stated that it was the third most used application; CHARM is not ranked among the software used from /sw/ because this package is mostly installed by the users in their home directories, and they do not link it with the versions provided by the application support staff. Also, the high usage listed in the first table comes from the fact that the developers named their projects with the same package name that they are developing. Looking for the versions and the location of libraries are paramount factors for an accurate report of their usage since performing a search with only its name can result in deceptive results like CHARM as the third most used library on Kraken.

Overall, Table 1, Table 2 and Table 3 show that the most used libraries at link time are the one installed by the vendor machine, in /opt directory. In the next section, we will focus on the most executed libraries.

4.3 Reports on executables launched

4.3.1 Overall executable usage

Table 4. Executable usage for JaguarPF in 2009, in 2010 and on Kraken in 2010

Rank	Jaguar 2009	usage	Jaguar 2010	usage	Kraken 2010	usage
1	nw_para	219933	nw_para	340926	interpo	60032
2	ior	46613	vbc1_7	32926	namd	8389
3	vbc1_4	29475	vasp	23134	chimera	4000
4	vbc1_3	13954	amber	6830	amber	3886
5	vasp	11478	namd	4201	mpiblast	2917
6	visit	2198	pltar	2925	enzo	2625
7	namd	710	chimera	1530	espresso	1018
8	espresso	639	espresso	350	lammps	886
9	spdep	578	visit	331	vasp	451
10	lammps	412	gromacs	173	gromacs	327

Table 4 shows the top 10 executables launched. The most executed code on Jaguar and Kraken is associated with climate

modeling (in dark blue). On JaguarPF, it is the same application both years, executed by only one user, that largely dominates the runs, more than 200,000 times in 2009 and more than 340,000 executions in the first quarter of 2010. The most executed code on Kraken is interpo, a tool used for interpolate grids in climate modeling applications. It presents a peculiarity of being compiled on a XT4 machine (Athena) while the runs were performed on the XT5 Kraken. We were able to easily detect this information since the job table gathers the information where the code was built. In 2009, the second most executed application is IOR (I/O benchmark). Since JaguarPF is connected to the largest file system in the world based on Lustre [28] (a massively parallel distributed file system), many I/O tests have been performed. A computational nuclear structure application, called vbc1_x (version 3 and 4 in 2009 and 7 in 2010), is one of the most executed applications in 2009 and 2010 on JaguarPF.

In orange, Molecular Dynamics (MD) applications are highlighted such as Espresso [11] (Extensible Simulation Package for REsearch on SOft matter, a versatile software package for the scientific Molecular Dynamics simulations and analysis), VASP (package for performing ab-initio quantum-mechanical molecular dynamics), NAMD [16], LAMMPS [17] (Large-scale Atomic/Molecular Massively Parallel Simulator), AMBER [5] and GROMACS. Half of the most used applications executed on JaguarPF and Kraken are MD applications.

In addition, we note the usage of utilities specifically developed for JaguarPF, SPDCP (parallel Lustre aware copy) and PLTAR (parallel Lustre aware tar).

4.3.2 Executable usage installed in /sw/xt

Table 5. Executable usage (installed in /sw/xt) for JaguarPF in 2009, in 2010 and Kraken in 2010

Rank	Jaguar 2009	usage	Jaguar 2010	usage	Kraken 2010	Usage
1	spdcp/ 0.3.6	493	vasp/ 4.6_r61	4180	namd/ 2.7b1	7866
2	namd/ 2.6	432	pltar/ 0.9.0	2925	amber/ 10	3730
3	vasp/ 4.6_r60	228	namd/ 2.6	1940	lamps/ mar09	276
4	cpmd/ 3.13.2	168	namd/ 2.7b1	296	gromacs/ 4.0.7	152
5	namd/ 2.7b1	130	spdcp/ 0.3.9	124	lamps/ oct09	137

Table 5 shows the top 5 executables launched that were installed in the /sw/xt directory. MD applications largely dominate the most executed applications installed by support staff. MD applications are the top 5 executed on Kraken. The most dominant are version 2.7 of NAMD and version 10 of AMBER. Besides the MD applications, the users intensively executed SPDCP and PLTAR only on JaguarPF, tools for archiving their data.

Comparing the results from the Table 4, we notice that some software does not show up in Table 5, such as Visit and Espresso. This is typically due to the users installing the software in their own directory. On JaguarPF, the version in /sw/xt is 1.11.1 and it has been used only 7 times while some users have installed more recent versions of Visit (2.0 and 1.12) in their home directories (version 2.0 has been executed 1170 times in 2009). A similar analysis with Espresso, installed on Kraken, where the version in /sw correspond to 2.1.1 while most of the usage correspond to version 4.1.3. This indicates that an upgrade of the software is needed.

4.4 Comparison with NICS's portal

Here the previous usage reports from ALTD are compared to the results from a different database tracking TORQUE usage data and job scripts on Kraken (not available on JaguarPF). This latter MySQL database stores the processed TORQUE accounting records for every job run on Kraken, and the corresponding job scripts. The executables are found from the job scripts using a set of heuristics that map patterns to application names and the results stored in a database. A web interface can then be used to generate metrics on the applications that have been run. These tools do not attempt to track library usage, rather just executables. All executables are tracked in this way with no ability yet to distinguish between users and staff.

Table 6. TORQUE most used executables on Kraken

Rank	Library	usage	Rank	Library	usage
1	arps	11844	6	sms	3383
2	amber	6789	7	sses	3153
3	namd	6450	8	vasp	3131
4	chimera	4473	9	mpiblast	2919
5	h3d	4270	10	gromacs	2234

Table 6 shows the top 10 applications based on absolute number of times an executable was found in job scripts by searching for a known list of executable names, for the same date range as Table 4 for Kraken. This method has a few drawbacks including false positive matches, inability to count executables more than once that appear in loops, and that some strings may match more than one application. For example, a user can name their directory mdrun for their own project, but mdrun is also the name of a parallel executable associated with GROMACS and as such can be double counted.

Table 5 shows that ALTD ranks interpo, namd, amber, chimera, and mpiblast in the top five. The data pulled from the Torque job scripts in Table 6 does not include interpo, due to the fact that the heuristic search has not been updated to look for it. Conversely, ALTD did not detect arps because it was run without being launched by aprun, and therefore can't be tracked by ALTD, a known limitation of ALTD.

The usage numbers do not correspond exactly. This is explained by several factors; for example, ALTD tracks only the executable run through aprun and does not catch interactively run codes like sequential runs for processing files such as grompp, a GROMACS executable to process files. Furthermore, the search method on the TORQUE job scripts produces some inconsistencies. For example, the "h3d" string is found in many job scripts, but often is not the executable in those scripts and thus the number reported is much higher than reality. ALTD reported the usage around 200 times (ranked 11th among all the applications) and the other 4000 detected by TORQUE on the scripts are irrelevant. NAMD is actually called many more times than reported in Table 6 because some scripts have the executable inside a loop, but it was only counted once. The usage of mpiblast essentially matches between the two tables. To be clear, there are fairly simple reasons why the results from ALTD differ from the Torque job scripts. In contrast, ALTD has an entry each time an executable is launched and therefore can provide an accurate count for those executables launched by aprun.

5. CONCLUSIONS

ALTD, the Automatic Library Trading Database, transparently records information on the usage of libraries used at link time and the usage of executables at job launch time. The data has been mined from two Cray XT5 systems (JaguarPF and Kraken) that indicate high usage of three main categories during the linking process. Performance and profiling tools such as CrayPAT and PAPI are the most used, followed by the numerical libraries like PETSc, libsci and ACML and the I/O libraries are the third most used category of libraries. Executables are largely dominated by climate modeling applications and the molecular dynamics applications such as NAMD and AMBER (installed by the application support staff.) Reporting the results collected at the link time and during the execution is imperative, because some packages are only used during compilation or at runtime and we should consider both case before making decisions on changing defaults or deprecating software.

The early results are quite interesting, especially the usage of numerical libraries such as ACML that is used more than libsci provided by Cray. This has spawned an interest in finding out why users are using ACML more than Libsci, for example.

In the near future, we envision generating more reports like determining the usage of libraries and executables by a project and doing similar rankings of "most used" executables but based on CPU hours used rather than absolute number of times launched. Also, we plan to develop a web-interface to dynamically and easily show different reports.

6. ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation. In addition, this research used resources at NICS supported by the National Science Foundation. This research was also partly sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy under contract number DE-AC05-00OR22725 with UT-Battelle, LLC. This research used resources of the OLCF at ORNL, which is supported by the Office of Science of the U.S. Department of Energy under contract number DE-AC05-00OR22725.

7. REFERENCES

- [1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., et al. (1999). LAPACK Users' Guide (Third ed.). *Society for Industrial and Applied Mathematics*
- [2] Aprun, Retrieved from <http://docs.cray.com/cgi-bin/craydoc.cgi?mode=Show;q=:f=man/alpsm/10/cat1/aprun.1.html>
- [3] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., et al. (2008). *PETSc Users Manual*. Argonne National Laboratory.
- [4] J. Borrill, J. C. Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms . *International Conference on Parallel Processing: ICPP, 2005*.
- [5] Case D.A. et al., The Amber biomolecular simulation programs. *J. Computat. Chem.* 26, 1668-1688, 2005
- [6] Cray. (n.d.). *Using Cray Performance Analysis Tools*. Retrieved from <http://docs.cray.com/books/S-2376-41/>
- [7] Fahey Mark, Jones Nick, Hitchcock Blake and Hadri Bilel, *ALTD Manual*. University of Tennessee, National Institute of Computational Sciences. in preparation 2010.

- [8] Frigo Matteo and Johnson Steven G., "The Design and Implementation of FFTW3," *Proceedings of the IEEE* 93 (2), 216–231 (2005). Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation
- [9] HDF5: <http://www.hdfgroup.org/HDF5/>
- [10] Levine, J. *Linkers and Loaders*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1999.
- [11] Limbach H., Arnold A., Mann B. and Holm C. "ESPreSo - An Extensible Simulation Package for Research on Soft Matter Systems". *Comput. Phys. Commun.* 174(9) (704-727), 2006
- [12] Mascagni M. and Srinivasan A. "Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation," *ACM Transactions on Mathematical Software*, 26: 436-461, 2000
- [13] Malony, S. S. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, SAGE Publications , 20(2):287-331, 2006
- [14] Mohr, B. TOPAS - Automatic Performance Statistics Collection on the CRAY T3E. *T3E, Proceedings of the 5th European SGI/Cray MPP Workshop, 1999*
- [15] Papi. Retrieved from <http://icl.cs.utk.edu/papi/>
- [16] Phillips J.C. et al., Scalable molecular dynamics with NAMD, *Journal of Computational Chemistry* 26, 16, p1781-1802, 2005
- [17] Plimpton S. J., Fast Parallel Algorithms for Short-Range Molecular Dynamics, *J Comp Phys*, 117, 1-19 (1995)
- [18] Rew, R. K. and G. P. Davis, NetCDF: An Interface for Scientific Data Access, *IEEE Computer Graphics and Applications*, Vol. 10, No. 4, pp. 76-82, July 1990.
- [19] SciDAC. Third Workshop on HPC Best Practices, September 2009 <http://outreach.scidac.gov/swbp/>
- [20] David Skinner(NERSC) and Chris Atwood (DOD), SciDAC notes <http://outreach.scidac.gov/swbp/report/Tools.pdf> .
- [21] Solchenbach, F. J. VAMPIR: Visualization and Analysis of MPI Resources, 1996.
- [22] Staples, G . TORQUE resource manager. *SC 06 Proceedings of the 2006 ACM/IEEE conference* (p. 8). Tampa, Florida: ACM
- [23] Tam, A. (2001). *Enabling Process Accounting on Linux HOWTO*
- [24] TOP500 <http://www.top500.org>
- [25] TRILINOS <http://trilinos.sandia.gov>.
- [26] Van Der Spoel, David and Lindahl, Erik and Hess, Berk and Groenhof, Gerrit and Mark, Alan E. and Berendsen, Herman J. C, GROMACS: Fast, flexible, and free, *Journal of Computational Chemistry*, 26, p1701-1718, 2005
- [27] Website on the function PAPI_get_shared_lib_info: http://geco.mines.edu/papi/html/papi_get_shared_lib_info.html
- [28] Wiki page information on LUSTRE project http://wiki.lustre.org/index.php/Main_Page