Software Usage on Cray Systems across Three Centers (NICS, ORNL and CSCS)

Bilel Hadri
National Institute for Computational Sciences (NICS)
University of Tennessee
Oak Ridge, TN, USA
e-mail: bhadri@utk.edu

Mark Fahey
National Institute for Computational Sciences (NICS)
Industrial and Information Engineering
University of Tennessee Knoxville
e-mail: mfahey@utk.edu

Timothy Robinson
Swiss National Supercomputing Centre (CSCS)
CH-6900 Lugano, Switzerland
e-mail: robinson@cscs.ch

William Renaud
Oak Ridge National Laboratory (ORNL)
Oak Ridge, TN, USA
e-mail: brenaud@ornl.gov

Abstract— **In an attempt to better understand library usage and address the need to measure and monitor software usage and forecast requests, an infrastructure named the Automatic Library Tracking Database (ALTD) was developed and put into production on Cray XT and XE systems at NICS, ORNL and CSCS. The ALTD infrastructure prototype automatically and transparently stores information about libraries linked into an application at compilation time and also tracks the executables launched in a batch job. With the data collected, we can generate an inventory of all libraries and third party software used during compilation and execution, whether they be installed by the vendor, the center's staff, or the users in their own directories. We will illustrate the usage of libraries and executables on several Cray XT and XE machines (namely Kraken, Jaguar and Rosa). We consider that an improved understanding of library usage could benefit the wider HPC community by helping to focus software development efforts toward the Exascale era.**

Keywords: **library tracking, Cray XT, Cray XE, NICS, ORNL, CSCS, numerical libraries, I/O, applications**

I. INTRODUCTION AND MOTIVATIONS

Supercomputing centers host HPC systems to enable the scientific discoveries of researchers worldwide. On these systems, the staff often supports hundreds of different software packages, each with multiple versions, and each version potentially built with multiple compilers; For example, at the National Institute for Computational Science (NICS), the staff support close to 150 software packages on Kraken [1] (funded by the National Science Foundation). With the costs associated in maintaining leadership computing systems, it is important to identify not only the most used libraries but also the least-used software in order to provide more efficient, targeted support. Without an in-depth knowledge of the actual usage of libraries at compilation and execution, application support staff are often required to make decisions about upgrading packages or removing older versions based to some degree on their own preferences or instincts. Unfortunately, because these decisions are based on incomplete data, staff must be conservative when deprecating and/or changing default software versions. Furthermore, national agencies, companies and research teams occasionally request reports on library and application usage on HPC systems, with particular interest placed on software that was funded or developed by one of their initiatives. Hence, it is essential to measure and monitor the software usage and forecast needs. The Automatic Library Tracking Database (ALTD) thus aims to better understand library usage on HPC systems.

The ALTD automatically and transparently stores information about libraries linked into an application at compilation time and also stores information about executions of such applications in batch jobs. The solution is based on intercepting the linker – to get information on libraries utilized – and intercepting the MPI job launcher to track parallel jobs executed. Wrapping the linker and the job launcher through scripts is a simple and efficient way to obtain the information automatically and transparently with no overhead. In addition, the ALTD stores information about compilation and execution in an SQL database, which can be mined to provide reports. For example, the ALTD can generate data on the most or least used libraries, and finer-grained detail such as specific version numbers. This database can assist application support staff in their decision process to upgrade, deprecate, or remove libraries, and thus provide a higher quality service to their users. It can also provide the ability to identify users that are still linking against deprecated libraries, or using libraries or compilers that are known to have bugs or performance issues. Tracking the usage of software thus allows for higher quality – and more efficient – user support.

The ALTD prototype has been installed on Cray XT/XEs at three different centers: NICS and Oak Ridge Leadership Computing Facility (OLCF), both located at Oak Ridge national Laboratory, and the Swiss National Supercomputing Centre (CSCS), in Lugano, Switzerland.

In this paper, we will present various reports on the usage of libraries and executables – for example, the number of instances that particular libraries have been used, and the total number of CPU hours consumed by third-party software applications. The data mining with ALTD on Cray systems managed by three different centers (NICS, ORNL

and CSCS) will help to obtain a detailed and accurate survey of the usage of software installed not only by the vendor, but also by the staff at different HPC centers and by the users in their home directory. For instance, since several compilers are provided on Cray systems, information about which compilers are used to build certain applications is beneficial not only for the centers, but also for the vendors and the developers of software applications and libraries.

Tracking library usage is part of a solution for improving the state of HPC software development and support as the computational science community moves towards the era of Exascale computing and beyond.

This paper is organized as follows: Section 2 provides an overview on ALTD. Section 3 presents results from data mining efforts including the most libraries and executables on Kraken, Jaguar and Rosa. Finally, section 4 summarizes the results of the software usage analysis and presents our plans for future improvements to the ATLD project.

II. OVERVIEW OF ALTD

A. Description of ALTD

In this section we describe briefly the objectives and implementation of the ALTD. We refer the reader to [2] for more details.

A primary objective of ALTD was to provide a lightweight solution with essentially no overhead at compilation or runtime. Our solution is based on intercepting the GNU linker (ld) [3] to get the linkage information, and the job launcher (aprun)[4] to get runtime information. Wrapping the linker and the job launcher through scripts is a simple and efficient way to obtain the required information automatically and transparently. ALTD is able to track both static and shared libraries; however, libraries that are loaded and unloaded at runtime such as dynamically linked libraries, are not tracked, since ALTD stores information during the linking process.

Our custom wrapper for the linker (ld) intercepts the user link line and parses the command line to capture the link line, which is then stored in the linkline table in the ALTD database (MySQL). Because of the fact that typically more libraries are included on a link line than are actually used, we employ a two-step process to identify the libraries actually linked into the executable. At the same time an ELF section header is included in the user's code, which is a marker that will be used to record any subsequent usage of this particular executable.

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 each library is used, by linking the jobs table back to the linkline table. This script extracts some job-specific environment variables from the batch system, such as job id (PBS_JOBID in the case of PBS, SLURM_JOBID in the case of SLURM). 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 linking process. Finally, the extracted information is inserted in the jobs table of the database, and control is passed back to the aprun wrapper that then calls the real aprun.

Towards the end of every month, Cray provides the latest release of its programming environment, which typically includes new versions of scientific libraries, message passing libraries, third-party libraries, and so on. It is generally the case, therefore, that many different versions of a given library (or compiler, or application) are present on the system at the same time. Hence it is important that not just the library name but also the version number is recorded. This is a fairly straightforward task, because of the way in which libraries are installed and made available to users. NICS, OLCF and CSCS all make their software available via modules, and the modulefiles set paths to libraries through setting environment variables. The paths contain version numbers according to known conventions, and thus the linkline recorded by ALTD contains version number information inherently. In other words, when a library is linked into an application, the complete library path is intercepted, and this path contains both the name of the library and its version number.

B. Installation

1) Various approaches

The ALTD framework can be put into production by implementing one of the following two methods:

a) using a modulefile: A modulefile can be used to make ALTD part of the default environment. This method gives the user the ability to bypass the ALTD wrappers by simply unloading the ALTD module (if it causes any unforeseen problem, for example). This method has the potential benefit of being scalable – if one has multiple linkers or job launchers with the same name in different locations, by loading the ALTD module appropriately, the wrappers are then "in front" of the various linkers/launchers.

b) linker and job launcher relocation: Another installation method is to rename the actual ld and aprun commands to, for example, ld.x and aprun.x, and then place ALTD's ld and aprun wrappers in /usr/bin. This method has the advantage that the ALTD framework is completely transparent to the user, and the user cannot turn off ALTD logging by unloading a module.

2) Machines installed

The ALTD has been in production on Cray XTs at three different centers: NICS, Oak Ridge Leadership Computing Facilty (OLCF), both located at Oak Ridge National Laboratory, and CSCS – the Swiss National Supercomputing Centre. A prototype was first put in production on the Cray XT Jaguar[5] at OLCF in 2009 using the linker and job relocation implementation. During the process of upgrading the Cray XT5 Jaguar to a Cray XK6 TITAN, the module approach has been deployed. ALTD has been in production on Kraken (using the module-based ATLD implementation) since February 2010. At CSCS, ALTD has been in production on Rosa [6] since June 2011, at which time the machine was an XT5. Rosa was upgraded to an XE6 in

December 2011. CSCS uses the linker and job launcher relocation implementation of ALTD.

3) Known Issues

There is a known issue between the Totalview [7] debugger and ALTD, which requires a workaround be put in place. The workaround allows Totalview to function correctly, but does not allow ALTD to track applications that are executed from within Totalview.

When testing ALTD with the Cray Programming Environment and the Cray Compiler (CCE), we noticed that CCE uses its own linker (rather than the GNU linker located in /usr/bin). A workaround with the altd modulefile is to set LINKER_X86_64 to the location of the ld from ALTD (/sw/altd/bin/ld) and the module ALTD has to be unloaded and loaded when PrgEnv-cray is used. After reporting this issue Cray created an environment variable ALT_LINKER for users who want to use an alternative linker path. This change was introduced with xt-asyncpe/5.05, whereby the workaround is no longer required.

III. Analysis Reports

ALTD stores information about every executable linked and every job executed, and this corresponds to hundreds of Mb of data gathered on each platform. It is clear, therefore, that data mining must be performed to extract any valuable information on the usage of libraries. For Jaguar and Kraken, we consider the data corresponding to a one-year period from 1 January to 31 December 2011. For Rosa, we consider the data gathered over a total of nine months: a six-month period (from 1 June to 30 November 2011) as a Cray XT5, and a three-month period (from 1 December 2011 to 29 February 2012) after the upgrade to XE6. In this report the usage by the staff involved in the installation of the software have not been included, to reduce the impact of high usage of a package during its installation. In total, ALTD has recorded the following data at each site:

- Kraken: 456,437 successful compilations by 860 users, and 1,434,972 application executions by 919 users.
- Jaguar: 1,024,793 successful compilations performed by 684 users and 1,325,538 application executions by 671 users.
- Rosa: 103,451 successful compilations by 254 users, and 501,102 application executions by 309 users.

It is interesting to observe that on both Kraken and Rosa the number of individual users compiling (linking) a code is significantly smaller than the total number of users running jobs. The percentage of active users who have never compiled a code is about 18% and 6%, on Rosa and Kraken, respectively. The presence of these "black-box" users, who are likely running applications installed either by the centers' staff or by their colleagues, needs to be taken into account when center staff are considering the installation and maintenance of third-party applications for their users.

A. Library usage during linking

In this section, we consider two different metrics for determining library usage: the total number of instances of linking a given library, and the number of unique users linking a given library. The reason we consider the latter metric is that the total number of instances can in some cases be artificially high – for example, in the case of autotuning experiments, which might involve a user performing many hundreds of compilations, generating executables that are never used for performing production science.

1) Compilers

Cray provides its users with support for several different compiler suites – open source and proprietary – including GNU, PGI, Intel, Pathscale and its own compiler, the Cray Compiler Environment (CCE). There is a not insignificant cost associated with supporting a compiler suite – programming environment infrastructure needs to be updated, and all associated libraries need to be continually rebuilt – so it is in the interest of Cray to have a good understanding of the use of compilers on its machines. Moreover, it is in the interest of HPC centers to know whether they need to be paying for particular proprietary compilers.

It is not, however, a completely trivial task to determine which compiler has actually been used in a given compilation. Determining the usage of the compiler by searching for compiler paths (like /opt/pgi, /opt/cray/cce, and so on) can produce false positives: most codes need some libraries from /opt/gcc, even when GNU's compiler was not used for the compilation. For this reason, we have restricted the mining of compiler usage to MPI codes, where we can determine the compiler definitively by searching for the "mpich2-<compiler>" string in the linkline.

Tables I and II show the number of compilations performed with, and the number of users of, the compilers available on the different systems. On both Kraken and Jaguar, PGI is the default compiler, and the results show that the majority of codes are built with PGI on these systems. The next most popular compilers are GNU and Intel, while the use of CCE and Pathscale – which is no longer officially supported by Cray – is much lower. It should be noted, however, that CCE has been available to Kraken users only from the middle of the 2011, and, moreover, the fix to track CCE usage described earlier was only implemented in the last quarter of 2011. The actual usage of CCE is therefore higher in early 2012, where the number of number of instances on Jaguar(TITAN), for example, has increased to close to 10% of total compilations and the number of users has increased to close to ten.

TABLE I. USAGE OF COMPILERS (NUMBER OF INSTANCES)

Compiler	*Kraken*	*Jaguar*	*Rosa*
GNU	26689	70854	9407
PGI	51154	132345	6116
Intel	6321	55182	1729
CCE	69	343	1415
Pathscale	14	1486	389

TABLE II. USAGE OF COMPILERS (NUMBER OF UNIQUE USERS)

Compiler	*Kraken*	*Jaguar*	*Rosa*
GNU	189	190	85
PGI	609	524	87

Compiler	*Kraken*	*Jaguar*	*Rosa*
Intel	146	64	41
CCE	3	3	39
Pathscale	3	38	20

Unlike the other centers, CSCS has chosen not to load a specific programming environment by default – users must load one explicitly themselves – and, perhaps because of this, the results are somewhat different to the other sites. At CSCS, GNU is the most popular compiler, followed by PGI. Like the other sites, Intel is the third most popular compiler, but CCE usage is much higher than at the other sites. Cray no longer officially supports Pathscale, and for this reason, CSCS encouraged its Pathscale users to change to a different compiler. Only two users have invoked the Pathscale compiler since the machine was upgraded to a Cray XE6 in December 2011.

2) Overall most used libraries

Besides the MPI library, the top 10 most used libraries fall into three major categories on all machines: numerical libraries (LibSci, FFTW, ACML, PETSc), I/O software (HDF5, NetCDF), and performance analysis tools (Craypat, PAPI, TAU).

In order to facilitate an improved understanding of library usage, we provide the rankings of libraries (installed by the vendor and the center staff) with the version number of the library used. This information can assist the staff, and vendor, in making decisions on deprecating and/or changing default software versions. Furthermore, this methodology also facilitates the detection of software installed by the users in their home directories.

3) Numerical Libraries

Numerical libraries are in general the most used libraries on the Cray systems and it is not surprising to observe that the Cray Scientific library (LibSci) [8] is the most linked library, as shown in Tables III to V (note that on Rosa, the MPI library is higher ranked with 19056 instances, as shown in Table II).

LibSci is loaded by default at all three sites and was linked in close to 10% of all codes compiled on the systems. It is used by about 50%, 43% and 33% of the users on Jaguar, Rosa and Kraken respectively. On Kraken it was determined, by examining the linklines recorded by ALTD, that a number of users were linking their applications to the native LAPACK/BLAS implementation from Netlib, rather than the optimized library from Cray. Such behavior would be nearly impossible to detect without the aid of a tool such as ALTD.

FFTW [9] is the second most popular numerical library and it is interesting to note that the 2.1.5 version of FFTW is still well used and, on Kraken, it represents about 15% of the total usage of FFTW. ACML [10] is in general the third most used library on the Cray systems. ACML is actually a dependency for the Cray compiler, however we find that most of its usage on Jaguar, for example, is in conjunction with the PGI and GNU compilers. The high usage of ACML might be explained by its increased performance for certain LAPACK routines such as QR factorization and eigenvalues, as shown in [11]. A variety of packages including PETSc [12], Trilinos, Sprng are next highest ranked. We note that the TPSL (Third Party Scientific Libraries) [8] module, which contains a collection of third-party mathematical libraries for solving problems in dense and sparse linear algebra, is ranked among the top 10 on both Kraken and Rosa. TPSL includes the Hypre, SuperLU, SuperLU_dist, MUMPs, and ParMetis libraries.

Remarkably, on Kraken, we notice that ATLAS is the second most linked package, albeit by only 8 users. This case is likely an example of an autotuning experiment, since ATLAS autotunes the BLAS for the systems through empirical compilations. The example of ATLAS shows that taking into consideration only the number of instances of linking of a given library can be in some cases give an unfair reflection of real usage.

On Jaguar and Kraken there are some heavily used numerical packages that were installed by the users; an example is fftpack [13]. By examining the linklines recorded by ALTD we were able to determine that the usage of fftpack is associated with the wrfv3 code, a Weather Research and Forecasting model [14].

TABLE III. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON KRAKEN

Library	instances	users	Library/ version	instances	users
Libsci	42271	291	libsci/10.5.02	29787	220
atlas	35954	8	fftw/3.2.2.1	15987	128
fftw	24494	235	xt-libsci/10.4.5	12167	169
acml	3537	59	fftw/2.1.5	3710	64
petsc	2460	20	acml/4.4.0/	3088	39
sprng	1745	13	sprng/2.0b/	1739	12
arpack	1721	11	petsc/3.1.05	1571	13
tspl	1517	14	arpack/2008	1543	1
gsl	1451	48	tpsl/1.0.0/	1517	14
fftpack	1317	35	gsl/1.14	1063	39

TABLE IV. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON JAGUAR

Library	instances	users	Library/ version	instances	users
libsci	74970	317	libsci/10.4.4/	47383	245
fftw	48728	163	fftw/3.2.2.1/	44779	109
acml	17198	58	libsci/10.5.0/	26303	208
trilinos	7518	25	acml/4.3.0/	9360	43
petsc	6008	58	acml/4.4.0/	7727	32
parmetis	1810	19	petsc/3.0.0.10/	1882	29
umfpack	1773	24	fftw/3.2.2/	1832	15
arpack	1166	12	parmetis/3.1 .1	1793	15
fftpack	1069	21	trilinos/10.4.0/	1786	10
pspline	1066	16	petsc/3.1.04/	1152	25

TABLE V. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON ROSA

Library	instances	users	Library/ version	instances	users
libsci	6240	109	fftw/3.2.2.1	4497	53
fftw	6042	84	libsci/11.0.01/	2806	33
acml	2020	48	libsci/10.5.02	1919	55
trilinos	1090	9	acml/4.4.0/	1123	44
tpsl	974	7	libsci/11.0.04/	974	33
parmetis	913	8	trilinos/10.6.0	731	1
umfpack	865	9	fftw/3.3.0.0	721	27
petsc	469	11	tpsl/1.0.0/	532	3
mkl	145	7	libsci/11.0.03/	512	24
Gsl	116	4	fftw/2.1.5	346	16

An examination of library versions reveals that users on Rosa are linking their codes with more recent versions compared to users on Kraken and Jaguar. This could be explained at least to some degree by the forced relink required when Rosa was upgraded from XT5 to XE6 in December 2011. It might also be explained by a higher regularity of changing of default modules on Rosa compared to Jaguar.

One of the most powerful aspects of ALTD is its ability to identify users running codes that were built with legacy versions of libraries (particularly ones that are known to provide suboptimal performance, or are known to contain bugs). As an illustrative example of how support staff might use the ALTD tool in this manner, we analysed which applications were run on a month-by-month basis, noting the version of LibSci that was linked in for each application. The results for Rosa are shown in Table VI. The time period covered corresponds to the XE6 incarnation of Rosa, which came online with LibSci version 11.0.03. Version 11.0.04 was released at the end of December, 11.0.05 at the end of February, and 11.0.06 at the end of March. The results show that the majority of codes that ran in March 2012 were built with the oldest version of LibSci available on the system (version 11.0.03, released in November 2011). Moreover, there are essentially no applications currently running on Rosa that were linked with either of the two most recent versions of LibSci. Similar analysis on Kraken revealed that nearly half of all users are still executing codes built with 10.5.X and even 10.4.X versions of LibSci. This shows that some users have not recompiled their codes since the upgrade from CLE2.2 to CLE3.1, as 10.4.5 is not available on the new OS.

TABLE VI. APPLICATIONS LINKED WITH LIBSCI ON ROSA

	Usage of LibSci versions			
Month	**11.0.3**	**11.0.4**	**11.0.5**	**11.0.6**
Dec 11	2326	18	0	0
Jan 12	4875	2387	0	0
Febr12	6388	4459	0	0
March 12	2280	1693	4	0
	Number of users using LibSci versions			
Dec 11	33	4	0	0
Jan 12	17	29	0	0
Febr12	12	30	0	0
March 12	12	26	2	0

TABLE VII. APPLICATIONS LINKED WITH LIBSCI ON KRAKEN

	Usage of LibSci versions					
Month	**10.4.5**	**10.5.0**	**10.5.02**	**11.0.01**	**11.0.4**	**11.0.6**
Dec 11	2977	0	18886	1930	0	0
Jan 12	4625	4	20370	621	77	0
Febr12	2794	17	14262	692	1	0
March 12	520	0	8582	979	8890	7
	Number of users using LibSci versions					
Dec 11	17	0	71	5	0	0
Jan 12	14	1	76	4	2	0
Febr12	17	1	95	7	1	0
March 12	11	0	47	7	75	2

4) I/O libraries

I/O libraries are the second most popular class of libraries used on the Cray systems, the most prevalent packages being HDF5 [15], NetCDF [16] and Szip [17], as shown in Tables VIII to X. We notice that the relative usage of these libraries differs significantly across the centers, however. On Kraken, hdf5 and iobuf [18] (installed by the staff) represent about 70% of I/O library usage. The usage of hdf5 is divided among several versions, and older versions such as 1.6.10 are used by a significant number of users. The Adaptable I/O System (ADIOS) [19] library, developed at OLCF, is used by only a few users on Kraken and Jaguar. Interestingly, the majority of use is of versions built by the users themselves. On Jaguar, NetCDF is the most used I/O package and most of the usage corresponds to a legacy version: 3.6.2. On Rosa, we find that users are linking their codes with only recent versions of HDF5 and NetCDF.

TABLE VIII. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON KRAKEN

Library	**instances**	**users**	**Library/ version**	**instances**	**users**
hdf5	6574	126	iobuf/beta	5763	16
Iobuf	5781	17	hdf5/1.8.4.1	2023	48
netcdf	3564	61	hdf5/1.8.5.0	1931	50
Adios	2804	3	hdf5-par/1.8.5.0	1811	23
Szip	1533	51	netcdf/3.6.2/	1600	25
hdf4	661	11	szip/2.1	1533	51
p-netcdf	320	6	hdf5-par/1.8.4.1	1467	17
Silo	62	5	netcdf/3.6.3/	868	5
			netcdf/4.0.1.3/	593	16
			hdf5/1.6.10	505	18

TABLE IX. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON JAGUAR

Library	**instances**	**users**	**Library/ version**	**instances**	**users**
netcdf	63172	159	netcdf/3.6.2	52164	98
hdf5	15750	167	p-netcdf/1.1.1	8577	21
p-netcdf	12448	36	hdf5/1.8.3.1	5713	42
szip	3427	55	netcdf/4.0.1.1	5089	30
adios	3427	30	hdf5-par/1.8.4.1	4875	50
silo	2339	20	p-netcdf/1.0.3	3863	15
liblut	273	5	szip/2.1	3427	55
			netcdf-hdf5par/4.0.1.3	3324	28
			hdf5/1.8.5.0/	2900	58
			netcdf/4.1.1.0/	2685	39

TABLE X. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON ROSA

Library	**instances**	**users**	**Library/ version**	**instances**	**users**
hdf5	4742	50	hdf5/1.8.5.0/	4375	47
netcdf	2769	43	netcdf/4.1.1.0/	2560	41
			hdf5-par/1.8.5.0/	2255	28
			hdf5-par/1.8.3.1/	1235	6
			netcdf-hdf5par/4.1.1.0	884	5

Library	instances	users	Library/ version	instances	users
			hdf5-par/1.8.4.1/	417	6
			hdf5/1.8.4.1	299	6
			hdf5-par/1.8.6/	234	10
			hdf5-par/1.8.2.3.1	157	3
			hdf5-par/1.8.7	45	5

5) Performance tools

The most commonly used performance tool on the Cray systems is Craypat [8], followed by PAPI[20] and TAU [21], as shown in Tables XI to XIII. As was highlighted in [2], Craypat was in fact found to be the most used package on Jaguar in 2009, and on Kraken in 2010, when ALTD was in its early production phase. The number of users making use of performance tools to analyze their code is relatively small compared to the other categories. This might be explained by the fact that most users are relatively familiar with the Cray systems and their codes, and that profiling and analysis might be employed during code development rather than during the production science project. Only on Jaguar do we observe a substantial use of Vampir [22] and the HPCToolkit [23].

TABLE XI. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON KRAKEN

Library	instances	users	Library/ version	instances	users
craypat	4628	33	perftools/5.2.0	3352	21
papi	999	63	craypat/5.1.3/	2950	30
tau	471	39	craypat/5.1.0/	1674	11
fpmpi	295	11	papi/4.1.2/	339	34
mpip	113	3	tau/2.20	315	38
ipm	32	6	fpmpi/1.1	295	11
			papi/4.1.0.0.2	229	17
			papi/3.6.2.2	200	8
			papi/3.7.2	133	19
			mpip/3.1.2	113	3

TABLE XII. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON JAGUAR

Library	instances	users	Library/ version	instances	users
papi	6091	144	papi/3.7.2	3787	121
craypat	3076	59	craypat/5.1.0/	1331	29
vampir	1854	20	vampirtrace/5.11	782	16
hpctoolkit	597	20	vampirtrace/5.12	780	4
tau	537	26	craypat/5.1.3	767	19
			papi/3.6.2.2	388	7
			papi/4.1.0.0.2	285	17
			craypat/5.0.2/	275	20
			vampirtrace/5.13	160	2
			hpctoolkit/5.1.0	143	8

TABLE XIII. LIBRARY USAGE RANKED BY NUMBER OF INSTANCES AND NUMBER OF USERS ON ROSA

Library	instances	users	Library/ version	instances	users
craypat	3465	32	craypat/5.1.3/	3462	31
papi	577	35	papi/4.1.0.0.2	295	28
scalasca	163	7	papi/4.1.4/	139	5
tau	27	2	papi/4.1.3/	95	3
			papi/4.1.0/	46	4
			tau/2.21	18	1

6) Programming Models/Language

Regarding programing models, the results show that on Rosa their use is limited to boost, which was used on 650 occasions and by just five users. We note that python use was detected for only one user, but this package is not generally used during the link phase, in any case. Indeed, for the others systems, the ALTD data show unreasonably low python usage.

TABLE XIV. NUMBER OF INSTANCES OF AND USERS USING DIFFERENT PROGRAMMING LANGUAGES

Programing Developments Packages	Machine			
	Kraken		*Jaguar*	
	instances	users	instances	users
Python	14	6	28	10
Chapel	6	2	635	10
Charm	210	10	814	10
Upc	13	1	10	1
globalarrays	214	6	78	5
boost	117	6	638	12

Otherwise, as shown in Table XIV, we notice a high usage of charm, globalarrays, and chapel, a parallel programming language whose development is led by Cray.

7) Software installed by users

On all the systems, many users choose to install their own versions of the libraries needed by their codes, even if they are installed centrally. Some popular packages detected are ATLAS, Trilinos, FFTW, Umfpack, as well as development projects such as ADIOS and NAMD[24].

B. Application usage during execution

This section will illustrate the usage of the most launched executables on the different systems. As described in the previous section, the left side of the table shows the overall usage of the executables from third party software, while the right side of the table presents only the results of the packages installed by the staff (this corresponds to the /sw/ tree on Jaguar and Kraken, and the /apps/ tree on Rosa). It should be noted that some users copy the centrally installed version of an application to their scratch space on Lustre before performing their simulations. This behavior would result in the use of centrally installed software being underestimated.[1]

It is a challenging task to create a complete reference list of executables and determine which of them correspond to a given third party software package. To search for a package such as Quantum Espresso[25], one would need to include a large number of different executable names (pw.x, ph.x, and so on). In the case of GROMACS [26] the executable is called mdrun; for Amber[27] it could be pmemd or sander, and lmp is generally associated with LAMMPS [28].

In the first part of this section we analyze application usage based on the number of individual executions, while

1 A better approach to identify applications would be to consider their "tag_id", which is assigned during link time and is stored in ALTD database. This method would identify categorically the packages installed by the staff at HPC centers.

in the second part we take in consideration the number of CPU hours consumed by the applications.

1) Application usage by the number of instances

Tables XV, XVI and XVII show the usage of the third party software and executables ranked by the number of instances on Kraken, Jaguar and Rosa, respectively. Overall, the data shows that the most used codes are dominated by classical and ab initio molecular dynamics packages (NAMD, Amber, Espresso, GROMACS, CPMD, CP2K, VASP). NAMD is by far the most used package both on Kraken and Rosa, while on Jaguar, the most executed application is IOR (an I/O benchmark). This is not as surprising as it seems when one considers that Jaguar is connected to the largest Lustre-based file system in the world [29], and we note also the use of utilities specifically developed for JaguarPF, such as SPDCP, a tool for archiving data. IOR is followed by LAMMPS, which is installed by the staff on Jaguar, however 97% of its utilization was for versions installed by the users. The same is true for the numerical weather prediction code ESMF (Earth System Modeling Framework) [30], Other climate modeling applications are also represented, including nwpara and ccsm.

TABLE XV. USAGE BY NUMBER OF INSTANCES AND USERS ON KRAKEN

Application	instances	users	Application / version	instances	users
namd	368349	109	namd/2.7/	294547	19
aprs	192749	20	namd/2.7b2	16237	10
amber	71261	18	namd/2.7b1-09	7834	4
hmc	51541	10	gromacs/4.5.3	3162	12
vasp	17884	33	namd/2.7b1/	2576	3
wrf	20141	19	amber/10	1830	7
espresso	14597	20	amber/11	1081	4
lammps	7035	40	namd/2.8	1052	10
gromacs	6345	28	cpmd/3.13.2	1047	6
cpmd	1773	6	q-espresso/4.2.1	950	1

TABLE XVI. USAGE BY NUMBER OF INSTANCES AND USERS ON JAGUAR

Application	instances	users	Application / version	instances	users
ior_bench	496352	2	vasp/4.6	16333	8
lammps	105345	31	lammps/9sep10	2899	6
esmf	86480	9	namd/2.7b1	1400	4
amber	64725	4	namd/2.6	692	1
vasp	45533	26	lammps/1jui11	477	4
nwpar	33484	1	spdcp/1.0.0	121	17
ccsm	25557	61	adios/1.3	46	4
espresso	19605	20	esmf/5.2.0	25	3
gromacs	8443	9	namd/2.7b4	14	1
namd	5143	14	gromacs/4.0.5/	12	1

TABLE XVII. USAGE BY NUMBER OF INSTANCES AND USERS ON ROSA

Application	instances	users	Application / version	instances	users
namd	74952	22	namd/2.8	29351	12
int2lm1	65274	8	namd/2.7b4	22160	6
parfe	52167	13	espresso/4.2.1	17296	7
cp2k	52010	37	cp2k/21.11.2011	11006	10
siba	37443	4	cp2k/17.08.2010	7838	5
gromacs	32573	13	vasp/5.2	2965	5
echam	28480	11	vasp/4.6	2053	3
espresso	21133	15	espresso/4.3.2	987	7
dlpoly	8535	4	Cpmd/3.13	680	4
vasp	5071	10	espresso/4.1	599	2

On Kraken, besides the molecular dynamic codes, we notice a high use of packages related to ARPS [31], a regional forecast system developed by the Center for Analysis and Prediction of Storms (CAPS). The use of this software on Kraken is mainly limited to pre- and post-processing of data from simulation runs executed on Athena, a Cray XT4 machine that was decommissioned in July 2011.

On Rosa, besides the molecular dynamics packages like NAMD and dlpoly[32], we observe a high usage of executables related to community development projects such as int2lm [33], a code developed by Climate Limited-area Modelling-Community. In the same science domain is ECHAM [34], a comprehensive general circulation model of the atmosphere. Two applications related to bone structures have particularly high usage: Parfe (a C++ code solving finite element problems arising from bone modeling)[35], and Siba (Simulate Bone Atrophy) [36].

2) Application usage by CPU hours consumed

The results presented in the previous section corresponded to the number of instances of execution of a given application. A better metric might perhaps be the number of CPU hours consumed by an application, a metric which is not available directly from ALTD, but which can be derived by retrieving the job_id associated with an executable (recorded in ALTD) and linking this to other databases such as batch system accounting [37] or user/project accounting systems.

Figures 1 and 2 show the top 10 most CPU-consuming applications on Kraken and Jaguar, respectively. Each application's position is plotted using the CPU hours consumed and the average cores per run, while the size of the circle corresponds to the total number of executions. The figures reveal that – for both Kraken and Jaguar – the applications that consumed the most CPU hours were not ranked among the top 10 most used applications.

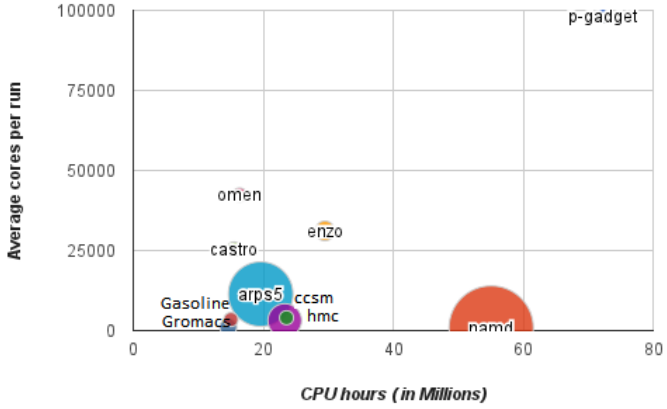

Figure 1. The top 10 most CPU consuming codes on Kraken in 2011, showing CPU hours consumed vs the average cores per run.

Indeed, on Kraken, the cosmological code p-gadget [38], used for the simulation of black hole formation, has used 72 million CPU hours (close to 10% of the total CPU hours consumed in 2011) in 332 executions using an average of 98304 cores. NAMD (the most executed application) is ranked in second position in terms of CPU usage, at 55 million CPU hours. Among the top 10, we find two further cosmological applications: enzo, with 12325 executions, and castro, which was run just 85 times.

On Jaguar the most CPU hours have been consumed by the S3D code[39], a massively parallel direct numerical solver (DNS) for the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry. We see some of the same applications on Jaguar as on Kraken, such as ccsm, HMC (Hybrid Monte Carlo simulations of lattice QCD) [40], and Omen. We note that HMC is typically executed on Jaguar using a very large number of cores (average of 55345 cores), while on Kraken the average is much lower (around 3150).

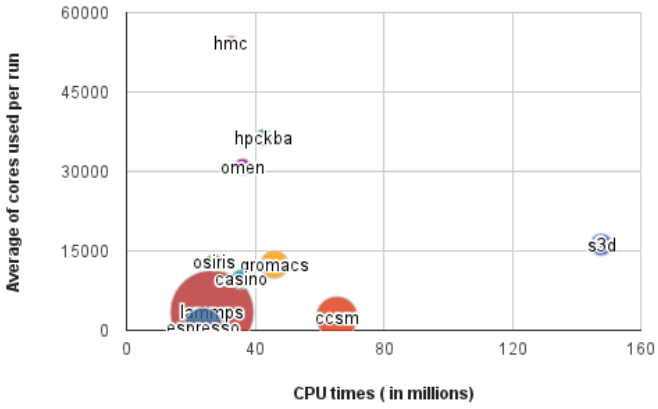

Figure 2. The top 10 most CPU consuming codes on Jaguar in 2011, showing CPU hours consumed vs the average cores per run.

IV. Conclusions and Future Work

ALTD, the Automatic Library Tracking Database, transparently records information about libraries used at link time and the usage of executables at job launch time. The data mined from three Cray systems (Kraken, Jaguar and Rosa) indicate high usage of three main categories of libraries during the linking process: numerical libraries (eg. LibSci, FFTW) are the most used, followed by I/O libraries (eg. HDF5, NetCDF), followed by performance and profiling tools (eg. CrayPAT and PAPI). The applications realm is largely dominated by molecular dynamics codes, such as NAMD, and climate modeling codes. In terms of the CPU hours consumed, however, cosmological simulations tend to dominate.

The results of our data mining have shown that a significant number of users are using their own executables even where there is a centrally installed application available. There could be a number of reasons for this: a user might require a non-standard version (they are contributing their own plugins or code modifications, for example), they might prefer to use a version built by them (perhaps built with exactly the same compiler and compiler options as other versions they use on different machines), or it may simply be that they are unaware that a centrally installed version exists.

The data mining has confirmed that there is extensive use made of numerical libraries (particularly LibSci, ACML, and PETSc). These packages provide BLAS, LAPACK, and FFTW routines as well sparse and iterative routines packed into single libraries. Since ALTD tracks only the name of library that was linked against (say LibSci or PETSc), the use of individual routines from such libraries is not currently identifiable. As future work, we envisage adding such functionality to track individual routines and correlate them back to "logical" libraries, which in turn could assist library develops and vendors (and centers) to tune the most used functions for current and upcoming architectures, including multicore and graphics accelerators.

At present the data mining of ALTD is a manual process, consisting of python scripts that generate a few simple SQL queries. In the future we hope to provide tools that automate the querying process: support staff would thus be alerted immediately to cases where, for example, a user is running a code that is linked against a deprecated or buggy library.

References

[1] http://nics.tennessee.edu/computing-resources/kraken

[2] M. Fahey, N. Jones, B. Hadri, The Automatic Library Tracking Database, Conference: Cray User Group 2010, Edinburgh, United Kingdom

[3] J. Levine, "Linkers and Loader," Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1999.

[4] aprun, "Retrieved from http://docs.cray.com/cgi-bin/craydoc.cgi?mode=Show;q=;f=man/alpsm/10/cat1/aprun.1.html,".

[5] http://www.olcf.ornl.gov/computing-resources/jaguar/

[6] http://user.cscs.ch/hardware/rosa_cray_xe6/index.html

[7] TotalView, "http://www.roguewave.com/products/totalview-family/totalview.aspx".

[8] Cray, "Cray Libraries," http://docs.cray.com.

[9] 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

[10] AMD Core Math Library, http://www.amd.com/acml

[11] B. Hadri and H. You A Performance Comparaison Framework for Numerical Libraries on Cray XT5 System., Proceedings of the 53rd Cray User Group (CUG11), Fairbanks, AK, May 2011

[12] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., et al. (2008). *PETSc Users Manual.* Argonne National Laboratory.

[13] P.N. Swarztrauber, Vectorizing the FFTs, in Parallel Computations (G. Rodrigue, ed.), Academic Press, 1982, pp. 51--83.

[14] http://www.wrf-model.org/index.php

[15] HDF5: http://www.hdfgroup.org/HDF5/

[16] 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.

[17] http://www.hdfgroup.org/doc_resource/SZIP/

[18] http://www.nics.tennessee.edu/computing-resources/kraken/software?software=iobuf

[19] ADIOS: http://www.olcf.ornl.gov/center-projects/adios/

[20] Papi. Retrieved from http://icl.cs.utk.edu/papi/

[21] Malony, S. S. The TAU Parallel Performance System. International Journal of High Performance Computing Applications, SAGE Publications , 20(2):287-331, 2006

[22] F. J. Solchenbach, "VAMPIR: Visualization and Analysis of MPI Resources," 1996.

[23] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R. Tallent. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010

[24] Phillips J.C. et al., Scalable molecular dynamics with NAMD, Journal of Computational Chemistry26,16,p1781-1802, 2005

[25] Limbach H., Arnold A., Mann B. and Holm C. "ESPResSo - An Extensible Simulation Package for Research on Soft Matter Systems". *Comput. Phys. Commun. 174(9)* (704-727), 2006

[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] Case D.A. et al., The Amber biomolecular simulation programs. *J. Computat. Chem.* 26**,** 1668-1688, 2005

[28] http://lammps.sandia.gov/doc/Manual.html

[29] http://www.olcf.ornl.gov/kb_articles/spider/

[30] ESMF http://www.earthsystemmodeling.org/

[31] Xue, M., K. K. Droegemeier, and V. Wong, 2000: The Advanced Regional Prediction System (ARPS) - A multiscale nonhydrostatic atmospheric simulation and prediction tool. Part I: Model dynamics and verification. Meteor. Atmos. Physics., 75, 161-193.

[32] http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx

[33] http://www.clm-community.eu/index.php?menuid=34&reporeid=53

[34] http://parfe.sourceforge.net/index.php

[35] http://www.mpimet.mpg.de/en/wissenschaft/modelle/echam.html

[36] http://www.utc.fr/esb/esb98/abs_htm/MUELLER/remodeling2.html

[37] "http://www.adaptivecomputing.com/resources/docs/gold/pdf/GoldUserGuide.pdf". Gold Database.

[38] Colin DeGraf, Tiziana Di Matteo, Nishikanta Khandai, Rupert Croft, Julio Lopez, Early Black Holes in Cosmological Simulations: Luminosity Functions and Clustering Behaviour, Volker Springel, Jul 2011

[39] Hawkes, E.R., Sankaran, R., Sutherland, J.C., Chen, J.H.: Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. Journal of Physics: Conference Series 16 (2005) 65,79

[40] M. A. Clark, Balint Joo, A. D. Kennedy, P. J. Silva , Better HMC integrators for dynamical simulations, PoS Lattice2010:323,2010