LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-226769

Report on Challenges and Resolutions for the Purple Development Environment

W. S. Futral, J. C. Gyllenhaal, M. E. Wolfe, C. M. Chambreau

December 13, 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Purple L1 Milestone Review Panel

Report on Challenges and Resolutions for the Purple Development Environment

Scott Futral, John Gyllenhaal, Matthew Wolfe, Chris Chambreau
12/12/2006

Deliverable
This addresses a requirement for a report on problems encountered with the tools and environment, and the resolution or status.

Report
Previous AIX development environment experience with ASC White and Early Delivery systems UV and UM was leveraged to provide a smooth and robust transition to the Purple development environment. Still, there were three major changes that initially caused serious problems for Purple users. The first was making 64-bit builds of executables the default instead of 32-bit. The second was requiring all executables to use large page memory. The third was the phase-out of the popular, but now defunct, third-party C++ compiler KCC, which required the migration of many codes to IBM's xlC C++ compiler.

On Purple, the default build environment changed from 32-bit builds to 64-bit builds in order to enable executables to use the 4GB per processor (32GB per node) memory available, and in order for the MPI library to do collective optimizations that required the larger 64-bit address space. The 64-bit build environment was made default by setting the IBM environment variable OBJECT_MODE to 64 and wrapping third-party software (mainly the gnu compilers) in order to make them handle OBJECT_MODE properly. Because not all applications could port to 64-bit right away, (usually due to third-party constraints, such as python not supporting 64-bit AIX builds until very recently), 32-bit builds of the major common third-party libraries also had be supported. This combined 32/64 bit build support was accomplished fairly seamlessly using the AIX feature that allows both 32-bit and 64-bit versions of the code to appear in the same library file, and documentation with clear examples helped our library developers generate the required combined 32-bit and 64-bit libraries for Purple.

In general, the port to 64-bit AIX executables went smoothly. The most common problem encountered with 64-bit was that many C codes didn't prototype malloc everywhere, via '#include <stdlib.h>', which caused invalid pointers to be returned by unprototyped malloc calls. This was usually seen in old crusty C libraries, leading to segfault on first use of the invalid pointer. Users had not encountered this prototype issue on other 64-bit Operating Systems (Tru64 and SUN) because those vendors worked around this issue by "auto-prototyping" malloc for the user. IBM instead required a

compiler option to be thrown for autoprototyping. This issue was resolved with user education, and often a quick recognition of the symptoms by support personnel.

The migration to 64-bit executables also initially broke many tools on AIX. Most were eventually ported to 64-bit, but unfortunately, the popular 32-bit memory tool ZeroFault has been very slow (and reluctant) to support 64-bit executables, despite much encouragement to the vendor from LLNL. In order to guarantee availability of the needed memory tools, an effort was funded by ASC over the last several years for the development of new AIX memory tools from Etnus and the open source tool Valgrind. These are described in the report on tools additions and enhancements. ZeroFault may yet eventually deliver 64-bit support for AIX, which users have indicated would be desirable as an additional memory tool option.

The second major change for Purple's development environment was the requirement that applications indicate to the OS, either through linker flags or environment variables at run-time, that the application's data be placed in "large pages" of memory. On purple, most of the node memory (~24GB of the 32GB total) was placed in "large pages" in order to maximize the performance of Purple's Power5 prefetching logic by increasing effective memory bandwidth. Additionally, large pages benefit the MPI libraries with increased bandwidth through RMDA (Remote Direct Memory Access) optimizations, and Power5 TLB (Translation Lookaside Buffer) hardware by reducing page misses. These "large pages" on Purple are implemented as 16MB pinned memory pages and, unlike the normal "small pages" which are 4k in size. Large pages cannot be swapped out to disk and have to be specifically requested by the application. The use of large pages by all applications was not practical since the use of large pages significantly slowed down (3X-100X) shells, scripting languages, and compiler scripts. This slow down was primarily due to the high fork cost when using large pages. This particular situation created an user education issue on the need to run only their MPI applications with large pages, and everything else with small pages. Large pages also made some system processes memory hogs.

The use of "large pages" by user's MPI applications unfortunately is not optional on Purple. Since there is less than 8GB of small page memory on each Purple node, leaving effectively only ~4GB small pages free, large memory footprint MPI applications that do not use "large pages" quickly start swapping all the nodes on which they are running if they use more than .5GB of memory per processor. In addition to a swapping MPI application making very little progress, it actually can take down the parallel file system on Purple, effectively shutting down Purple. Heavily swapping AIX nodes often will ignore critical parallel file system communications for up to 30 minutes, which is long enough to cause the parallel file system to panic and shut down.

To help deal with the strict requirements to use large pages, the systems administrators monitor jobs, noting those that are not using large pages to identify users that need large page education, and to remove jobs that start to swap on Purple. Posting of news items, documentation in the login banner, and raising awareness of the issue to LC support staff are part of the solution to the user education issue. LLNL has also worked with IBM to

build a check into MPI_Init that warns the users when MPI applications are run without large pages, and to automatically kill the job. An environment variable can be set by the user to allow jobs to continue running if omission of large pages was intentional. This MPI check is in the pipeline as a product enhancement from IBM, but has not been delivered yet for use on Purple.

The use of large pages also exposed several AIX kernel bugs in the shared library loading logic. After providing progressively more complex non-export controlled test cases (finding non-export controlled test cases was the hard part), and installing a series of kernel fixes over a period of more than a year, it is believed IBM resolved all known large page shared library problems in early November 2006 on Purple. Most users had effective workarounds very early on and only one code had to use inconvenient workarounds during the long fix cycle.

The third major change for Purple's development environment was the migration of our large C++ applications from the popular, but now defunct, third-party C++ compiler KCC to IBM's xlC C++ compiler. KCC was a cross-platform C++ compiler in which some ASC code teams had invested significant effort over several years in making compatible with their heavily templated C++ codes. After initial porting attempts to IBM's xlC C++ compiler, it became clear that the xlC compiler had not undergone the same stress testing with large templated C++ codes. Many problems were present, including the compiler segfaulting when compiling many of the files. Because all the xlC compiler developers were foreign nationals, they could not have direct access to the failing codes, which were all export-controlled. In tight collaboration with IBM starting January 2005, an LLNL compiler support team member developed non-export controlled test cases and helped debug the xlC compiler. This required four months of effort to enable compilation of the ASC code Kull. At this point, the xlC compiled executable ran 2.5X slower than the KCC version and also took 2 hours instead of the previous 1 hour to compile. After another five months of intense collaboration with IBM, the executable performance and the compile time were both able to exceed the mark of KCC. This IBM collaboration later allowed rapid compiler fixes to be developed for LANL and Sandia codes that were being ported to Purple in 2006.

Additional items were addressed during the initial period of Purple integration, and continue to be addressed which were of a less critical nature. One typical example is the generation of lightweight corefiles for OpenMP applications. An OpenMP application would segfault, but the lightweight corefiles did not appear to be generated. The problem was reported to IBM in October, 2005. A PMR was generated by IBM, and a fix was made available at the end of February, 2006.

In another category of issues, LLNL has worked effectively with third party tool providers and IBM to address issues and improve the Purple user environment. For example, IBM added a 'loader preload' mechanism at the urging of LLNL that enables better memory debugging tools to be developed by Etnus Totalview and others. The partnership between LLNL and IBM support teams have proven to be very effective in addressing issues in a reasonable time frame, to the benefit of the Purple user community.