Cray® XC50™ CLE Port to ARM: Functionality, Performance, and Lessons Learned

Jeffrey J. Schutkoske
Compute Products R&D
Cray, Inc.
Bloomington, MN, USA
jjs@cray.com

***Abstract*— The Cray Linux Environment (CLE) supports the Cavium ThunderX2 ARM processor in the Cray® XC50™ system. This is the first ARM 64 processor that is supported by CLE. The port of CLE to ARM provides the same level of functionality, performance, and scalability that is available on other Cray XC systems. This paper describes the functionality, performance, and lessons learned from the port of CLE to ARM.**

Keywords: Cray® XC50™ system, Cray Linux Environment, CLE, system management, operating systems, programming environment

I. INTRODUCTION

The Cray XC50™ system includes a new Cray Processor Daughter Card (PDC) which supports the Cavium ThunderX2 CN99xx ARM processor. The Cray Linux Environment (CLE) is enhanced to support the ARM processor.

The Cray Linux Environment (CLE) software stack provides robust services for the Cray XC50™ system components including the compute and service nodes. CLE includes the base operating system for the computational complexes, networking software stack, I/O software, and other specialized services.

The Cray XC™ CLE architecture is organized around service nodes and compute nodes. Service nodes perform the functions needed to support users, administrators, and applications running on compute nodes. Above the operating system level are specialized daemons and applications that perform functions unique to each service node.

CLE starts with a subset of a Linux distribution and Cray adds features for scalability and usability, as well as reliability, availability, and serviceability (RAS). By tailoring Linux to provide only those services required for applications, CLE greatly reduces the operating system footprint without sacrificing application functionality. CLE is designed to address scaling issues by minimizing operating system induced noise or jitter.

The port of CLE to ARM provides the same level of functionality, performance, and scalability that is available on other processors in Cray XC™ systems. The process included porting existing functionality to the aarch64 architecture, replacing functionality with new aarch64 functionality, and in some cases no changes were required. This paper describes the functionality, performance, and lessons learned from the port of CLE to ARM. Section II describes the basic hardware configuration. BIOS initialization is covered in Section III. The compute node Linux (CNL) is described in Section IV followed by the managed services and network stack descriptions in Section V and VI respectively. Reliability, availability, and serviceability (RAS) features are described in Section VII. Workload managers are discussed in Section VIII. Section IX describes the changes required to support the aarch64 architecture in Cray system management. Power management changes are described in Section X. The Cray programming environment is described in Section XI. Initial performance is discussed in Section XII followed by lessons learned in Section XIII. A summary is provided in Section XIV.

II. BASIC HARDWARE CONFIGURATION

The Cray PDC supports a four node, dual socket per node configuration. Each compute blade supports two Cray PDCs. The two sockets that comprise a single node are connected together via the interchip coherency interface (ICI). Each node supports a PCIe Gen 3 x16 channel to the Cray Aries interconnect. Each node also supports up to 8 DDR4 memory channels. Each ARM socket contains 32 cores with four threads per core. So, a dual socket node contains 256 (2*32*4) logical CPUs.

III. BIOS INITIALIZATION

The BIOS is based on the ARMv8 UEFI BIOS from the Linaro open source [1]. On node power up, the M3 micro-core ROM enables low level functionality needed by ARMv8, while the ARMv8 cores are being held in reset. The ARMv8 Boot firmware performs DDR initialization as well as other board specific initialization. It also performs ICI linkup and slave node discovery in the case of multi-node configuration. When the node is running, M3 firmware performs monitoring and power management function.

Cray added additional support for the Cray Aries interconnect initialization, high speed network boot support, and specific hardware error handling and reporting through the out-of-band (OOB) Cray Hardware Supervisory System (HSS).

The initial BIOS execution on the first boot after a flash of the BIOS takes approximately fifteen minutes. The subsequent boots require only 5 minutes because the memory training parameters are stored and reused. When BIOS is flashed these values are cleared.

IV. Compute Node Linux (CNL)

CNL has many of the positive attributes of a lightweight kernel (LWK), however it does not restrict services to the extent that many LWKs do. Instead, it provides low operating system overhead, while exposing standard Linux services and interfaces to applications. CNL not only provides standard page-based memory management for locally accessed memory, but also provides support for PGAS programming models. There are a number of areas that involved additional work to support Cray XC50™ system including CPU performance counters, core specialization, and huge pages.

A. Performance Counters

With the Cray XC50™ system, CNL supports SLES 12 SP3, kernel version 4.4 for the initial aarch64 release of CLE. As part of the development effort, Cray updated the kernel to version 4.4.92-6.18.1, which added support for the power management unit (PMU) counters. A number of these counters are utilized by the Cray Programming Environment (Cray PE) through the performance API (PAPI) [2].

B. Core Specialization

Cray supports core specialization on the Cray XC50™ system. Core specialization binds sets of Linux kernel-space processes and daemons to one or more cores within a Cray compute node. This enables the software application to fully utilize the remaining cores within its *cpuset*. All possible overhead processing is restricted to the specialized cores within the reservation. This has been shown to improve overall application performance. There is a user level API that is supported in the libjob library. When core specialization is desired for a job, *job_set_corespec()* must be called between *job_create()* and *job_set_affinity()*. The core specialization requested here takes effect upon the successful *job_set_affinity()* call. The only aspect that takes effect immediately is that after this call, *job_create()* calls will fail until this job is destroyed. Core specialization is supported on a per application launch basis, i.e. using the Cray application level placement scheduler (ALPS) *aprun* or Slurm *srun* command. A single job can be made up of multiple application launches, both in parallel and sequentially. CLE core specialization support in the Cray XC50™ system with ARM processors is on par with the x86-64 Cray XC™ systems.

C. Huge Pages

A translation lookaside buffer (TLB) is a cache of virtual-to-physical translations. Typically, this is a very scarce resource on a processor. Operating systems try to make best use of a limited number of TLB resources. This optimization is more critical now as bigger and bigger physical memories (several GBs) are more readily available.

Huge pages (also known as large pages) in simplest terms are blocks of memory. The huge page support in the Linux kernel is built on top of the multiple page size support that is provided by most modern architectures. For example, x86 CPUs normally support 4K and 2M (1G if architecturally supported) page sizes. CLE supports dynamic or on-demand huge page allocation. In some cases (e.g. "*HUGETLB_DEFAULT_PAGE_SIZE=64M aprun -m256h*") the kernel may have pre-allocated a number of physical huge pages as requested by *aprun* and will assign one of those huge pages to the user application at the requested user address.

But if there are not any pre-allocated huge pages of the desired size, the kernel will attempt to allocate a huge page of the desired size via the kernel's "buddy allocator". In both cases (pre-allocated via *aprun* at the start of the job, allocated at the time of a page fault during the job), the huge page is assigned to the job at the time of the page fault.

The kernel has limited knowledge of "jobs" and their lifetimes, so from the kernel's point-of-view almost all huge page allocations are "dynamic". The main distinction from the kernel's point-of-view is between "boot-time" allocation (which bypasses the kernel's "buddy allocator") and "runtime" allocation (all other cases that Cray has implemented). Although the community and SLES kernels support "boot-time" allocation, this is not used in practice with CLE, so huge pages are always allocated "dynamically" at "runtime".

CLE enhances Linux to support any huge page size (power of 2) from 4KiB to 2GiB by combining hardware natively sized pages. CLE huge pages can be allocated dynamically at runtime. CLE huge pages provide significant enhancements over the community and SLES kernel which only support 4KiB, 2MiB, and 1GiB huge pages. CLE huge pages support in the Cray XC50™ system is on par with the x86 Cray XC™ systems.

V. Managed Services

CLE managed services execute on the Cray XC50™ system service nodes. Service nodes run a version of Linux that has more features enabled than typically are on the compute nodes. The features enabled may depend on the services being provided and are configurable. The managed services execute on the standard x86-64 nodes within the Cray XC50™ system. There are a number of managed services supported in the Cray XC50™ system as follows:

- System Database
- Boot Service
- Netroot Service
- Data Virtualization Service
- DataWarp Service

- Gateway Service
- Lustre® Network Service
- Login Service

A. System Database

The system database (SDB), which is a MySQL database, contains the CLE system database. The CLE database contains both persistent and non-persistent tables. The processor and service processor tables are non-persistent and are created from the HSS data at boot time. The CLE database tables track system configuration information. The SDB makes the system configuration information available to the application level placement scheduler (ALPS), which interacts with individual compute nodes running CNL. There were a number of changes required to support the new CPU type and core counts.

B. Boot Service

The Cray XC™ system supports the ability to efficiently bootstrap large numbers of nodes by leveraging the high-speed network infrastructure to distribute image content to compute and service nodes. The boot node is PXE booted from the system management workstation (SMW). The SMW boot manager process, *bootmanager*, recognizes that the boot node is being booted and is responsible for installing the appropriate files, creating the appropriate PXE boot configuration files and placing them into their proper locations, taking into consideration which partition and which boot node is being booted.

The boot node daemon (BND) supports the high-speed boot of the compute and service nodes. BND writes the kernel and the *initramfs* (in-memory root file system) images into each node's memory. BND takes into consideration the identity (i.e. CPU family) of target nodes.

The average boot time for CLE is approximately 200 seconds with CLE 6.0 UP06 for a single node. The Cray boot process executes the boot on multiple nodes in parallel so that as the system size increases the overall boot time does not increase.

C. Netroot Service

The Netroot service provides a scalable mechanism for the efficient projection of shared root file system images, as well as other file system images for binding in content, such as the Cray PE or Cray online diagnostics. Netroot provides a mechanism that allows images to be booted on XC compute and/or service nodes without incurring the memory overhead by constructing a copy-on-write (COW) root file-system from a scalable read-only network file system and tmpfs file system utilizing the OverlayFS support present in the SLES 12 and upstream kernels. There were no changes required to support the Cray XC50™ system with ARM processors.

D. Data Virtualization Service

Cray's I/O forwarding as well as file projection solution is provided by the Cray data virtualization service (DVS). DVS is a distributed network service that provides applications running on compute nodes transparent access to file systems residing on DVS service nodes and remote servers in the data center. Cray DVS provides features to aid in the distribution of large I/O requests.

DVS is a core service in CLE, used for Netroot fanout, serving the programming environment, and DataWarp (discussed next). DVS currently uses Lustre® networking (LNET) and Lustre® network driver (LND) as its primary transport mechanism. There were no changes required to support the Cray XC50™ system with ARM processors.

E. DataWarp Service

DataWarp is an I/O accelerator technology that allocates storage dynamically in either private (dedicated) or shared modes. Storage performance quality of service can be provided to individual applications, based on the user's policies.

DataWarp can be used as a global storage cache for parallel file systems (PFS) such as Lustre® [3] or General Parallel File System (GPFS™) [4]. However, Cray currently only tests and supports Lustre with DataWarp. In these scenarios, the applications I/O accelerator capabilities drive up the overall utilization of the parallel file system by buffering performance across this new tier. DataWarp improves overall application performance by decoupling application I/O from the corresponding PFS I/O, while also improving the performance and resiliency of conventional disk-based solutions [5]. There were no changes required to support the Cray XC50™ system with ARM processors.

F. Gateway Service

Cray XC™ systems must be easily integrated into customer site networks using gateway nodes. Cray XC™ systems use a RFC 1918 "Private" IP network (e.g. 10.x.x.x) internally, which often overlaps with customer site usage, and so a network address translation (NAT) mechanism is required. This is provided by the traditional Linux NAT running on the network gateway nodes, or by utilizing RSIP (Realm Specific IP) which provides a similar function with different performance characteristics. These gateway nodes are then statically allocated amongst the compute nodes. There were no changes required to support the Cray XC50™ system with ARM processors.

G. Lustre® Network Service

In a Cray XC™ system with a Lustre® file system, the high-speed network (HSN) connecting the Lustre® clients on the compute nodes with the Lustre® servers is implemented using Lustre® networking (LNet), which provides the communication infrastructure required by the Lustre® file system. Disk storage is connected to the Lustre® MDS and OSS server nodes using direct attached storage or

traditional storage area network (SAN) technologies. There were no changes required to support the Cray XC50™ system with ARM processors.

H. Login Service

Login Services provide a user environment on nodes selected for use with application development and initiation of workloads within the Cray XC50™ system. These services typically include providing access to the Cray PE, support for customer user account authentication such as through lightweight directory access protocol (LDAP), and other tools consistent with supporting a richer user environment than is found on compute nodes.

Users access an Ethernet network server connection to the login nodes. Logins are distributed among the login nodes by a load-leveling service through the domain name service (DNS) that directs them to the least loaded login node.

For Cray XC50™ system, the service nodes are all still x86-64 based nodes. Customers require the ability to compile their code natively on ARM based nodes. To support this requirement, Cray uses a repurposed compute node, which is ARM based, as a login node. Instead of logging into the x86-64 login node and then the ARM login node, users are automatically routed to the ARM login node. There is a one-to-one mapping from the x86 login node to the repurposed compute login node.

VI. Network Stack

The Cray Aries network software stack includes three kernel drivers and two user level libraries as follows:

- Generic Hardware Abstraction Layer driver (GHAL)
- Generic Network Interface (GNI) driver
- User level Generic Network Interface (uGNI) library
- Distributed Shared Memory Application (DMAPP)
- IP over Generic Fabric (IPoGIF)

The Cray XC™ system software stack is depicted in Figure 1. The GNI API includes two sets of function calls. User-level high-performance applications use uGNI functions while kernel-level drivers use kGNI functions.

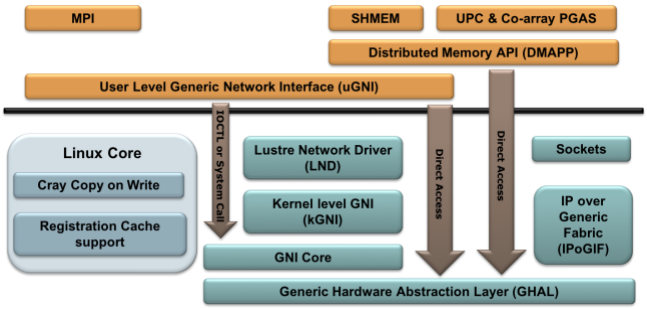

Figure 1: Cray XC Communications Stack

kGNI is a kernel module that presents to kernel-space code an API similar to that of uGNI. The GNI core provides low-level services to both uGNI and kGNI. kGNI and GNI core are both in the kGNI module. uGNI also provides additional functionality important for the communication clients such as MPI which is described in Section XI. The generic Hardware abstraction layer (GHAL) isolates all software from the hardware specifics of the Cray Aries interconnect.

Distributed Shared Memory Application (DMAPP) is a communication library which supports a logically shared, distributed memory (DM) programming model. DMAPP provides remote memory access (RMA) between processes within a job in a one-sided manner. One-sided remote memory access requests require no active participation by the process at the remote node; synchronization functions may be used to determine when side-effects of locally initiated requests are available. DMAPP usage in the Cray Programming environment is described in Section XI.

A. Aries PCIe Initialization

During the initial bring up of the ARM processors, the Aries driver failed to load. The problem was due to the fact that no resources were being assigned to the Aries device over PCIe. It was determined that the Aries device had a class of zero in its config space and the kernel does not assign resources to an "unclassified" device. The problem was resolved by setting the Aries device as PCI_CLASS_NETWORK_OTHER.

There were a number of additional changes required in the network software stack centered on enabling the correct kernel configuration options for the aarch64 architecture.

B. Aries Write Combining

All of the memory spaces associated with the fast memory access (FMA) are configured within an x86-64 processor to be a write combining space. ARM defines three properties as follows:

- Gathering or non Gathering (G/nG): This property determines whether multiple accesses can be merged into a single bus transaction
- Re-ordering or non Re-ordering (R/nR): This determines whether accesses to the same device can be re-ordered with respect to each other
- Early Write Acknowledgement or non Early Write Acknowledgement (E/nE): This determines whether an intermediate write buffer between the processor and the slave device being accessed is allowed to send an acknowledgement of a write completion

The combination of these three, GRE, which is the least restrictive, provides the equivalent of the x86-64 write combining in the aarch64 architecture [6].

This is done to improve performance in that it allows multiple store instructions performed by software to be

combined into a single request packet between the processor and Aries. Through write combining, multiple stores to Aries FMA descriptor space may become combined into a single request to Aries. Cray defined a *ioremap_wc()* method for the aarch64 architecture to align with the x86-64 method.

C. IP Over Generic Fabric

Cray provides a reliable, redundant path, high speed IP network over the Aries network. It is configured on the system as an IP over generic fabric (IPoGIF) interface. The socket buffer (SKB) is the fundamental data structure used in the Linux networking code. Every packet sent or received is handled using this data structure. When initializing the SKBs, the ipogif driver is called. Cray increased the size of the kernel configuration parameter, CONFIG_FORCE_MAX_ZONEORDER, from 13 to 16, which allows an allocation of up to 512MiB.

VII. Reliability, Availability, and Serviceability

Cray supports a number of features that enable reliability, availability, and serviceability (RAS) as follows:

- System Dump
- Node Heartbeat
- Node Health Checker
- ACPI Platform Error Interface (APEI)
- CoreSight
- Diagnostics

These features are described in the following sections.

A. System Dump

The *cdump* and *crash* utilities may be used to analyze the memory on any Cray service node or CNL compute node. The *cdump* command is used to dump node memory to a file. After *cdump* completes, the *crash* utility can be used on the dump file generated by *cdump*.

The *cdump* command executes on the SMW, which is an x86-64 processor. The *kexec-tools* are updated to version 2.0.15 to enable aarch64 support. The *fuse_vmcore script* fuse mounts remote node memory on the SMW and runs *kexec-tool* and *makedumpfile* to create dump on the SMW. Cray modified the *kexec-tools* and *makedumpfile* to run the aarch64 specific code on the x86-64 SMW.

The *crash* utility was also updated to handle both machine architectures automatically on the SMW. Cray builds and installs two binaries for each machine architecture: *crash_arm64* and *crash_x86_64*. The *crash* utility also automatically selects the correct machine type and sets up the crash environment for the user.

B. Node Heartbeat

CLE updates an Aries MMR per node approximately once per second to indicate that that node is alive, i.e. node heartbeat. The HSS firmware executing on the blade controller monitors this heartbeat. When the HSS firmware detects that a heartbeat has stopped it notifies the SMW.

Cray's original implementation used x86-64 specific code instead of a more generic implementation. Cray modified this code to be architecture agnostic.

C. Node Health Checker

The node health checker (NHC), sometimes referred to as NodeKARE, is used under three circumstances as follows:

1. Immediately after applications within a reservation have terminated
2. Immediately after a reservation has terminated
3. When a node boots

NHC can also be configured to run only on abnormal termination.

To support running NHC at boot time and after applications and reservations complete, NHC uses two separate and independent configuration files, which enable NHC to be configured differently for these situations.

The NHC is invoked automatically upon the termination of an application or the end of a batch system reservation. Cleanup is therefore a two-stage process:

1. Application cleanup is performed following an application exit.
2. Reservation-level cleanup is performed following the termination of a batch system reservation.

System resources cannot be freed for reallocation until both the application cleanup and reservation cleanup have completed successfully. There were no changes required to support the Cray XC50™ system with ARM processors.

D. ACPI Platform Error Interface (APEI)

With introduction of the Cray XC50™ system compute blade and its ARMv8 nodes, the Cray XC ™ system RAS infrastructure was modified to support a node type for which the x86-64 machine check architecture (MCA) is replaced by the aarch64 ACPI Platform Error Interface (APEI) [7].

A comprehensive description of APEI is out of scope for this paper. However, a brief background is in order. Per APEI, firmware on conformant, supporting platforms must publish a hardware error source table (HEST) in host memory. The HEST enumerates a platform's particular error sources that are exposed to CLE by firmware through the APEI interface [8].

Cray supports the in-band notification of errors to CLE. CLE reports the error information via the out-of-band (OOB) channel to HSS firmware executing on the blade controller, which is forwarded to the SMW.

Cray also supports the ability for HSS firmware executing on the blade controller to read uncorrectable errors directly from the node firmware via the OOB channel, thus providing a reliable reporting path even when the node hangs or takes a double fault.

All hardware errors are aggregated to the SMW in the hardware error log, *hwerrlog*, and can be viewed via the SMW *xthwerrlogd* command.

E. CoreSight

The embedded CoreSight interface, eCSI, is used to communicate with an aarch64 processor for system level monitoring and debugging. The eSCI replaces the embedded ITP interface for Intel processors. CoreSight defines the internal debug topology and mechanism for OOB debugging on ARM processors [9]. Cray leverages the Open On-Chip Debugger, OpenOCD, [10] to interface with CoreSight. OpenOCD provides several low-level JTAG interfaces for a variety of debug devices, and it also provides a command-line interface that offers a rich set of primitives, such as reading and writing memory or registers. Cray provides scaling and performance software on the SMW that utilizes the OpenOCD interfaces on the blade controller. Cray also wrote the JTAG driver that communicates with the ARMv8 processor on the blade controller.

The SMW eCSI command, *xtcsi*, is used to execute scripts on the designated node. There are a number of scripts available. Some of the basic scripts are as follows:

- csitest: tests the basic eCSI interface
- dump-memory-errors: gathers ECC memory error information and counts
- memory-read: reads a block of memory
- rungdb: runs an instance of the *gdb* debugger [11] on the target node

F. Diagnostics

Cray provides a number of online diagnostics that validate the nodes functionality and performance [12] [13]. These diagnostic tests execute under CLE and are defined as follows:

- Node NUMA Test (*xtnumatest*): A set of tests that exercise and test the NUMA capability of an SMP node.
- Node Memory Test (*xtmemtester*): A set of tests that exercise and test all of node memory.
- Node Performance Test (*xtcpuperf*): Targets the performance of the Processor utilizing DGEMM, which is a double precision floating-point matrix multiplication application and computational performance test for the compute node processors. At the end of an iteration, the performance is measured in GFlops.

Cray also provides the workload test suite (WTS) utility, *xtsystest*, that is comprised of a main top-level script named *test_suite.py*, a default configuration file named *test_suite.ini*, and a set of standard test modules. The test modules are pre-compiled benchmarks and diagnostics binaries that are delivered are part of CLE. The utility sequentially executes the set of benchmarks, diagnostics, and/or applications that are defined in the default configuration file, or the set of tests defined in a custom, user-defined configuration file. The tool provides a report about the individual results of each test, as well as an overall summary of all the tests that have been executed.

The Cray developed Aries network diagnostics are supported on the Cray XC50™ system. The Aries FMA and block transfer engine (BTE) concurrent test, *xtfbc*, is designed to test the dedicated FMA and BTE logic blocks concurrently, while stressing the shared hardware such as the Processor Interface (PI), Network Interface (NICs), Netlink (NL), network tiles, and high-speed links. The FMA and BTE threads exchange data between like thread types.

The Aries all-to-all performance test, *xta2a*, is used to measure the performance on all-to-all communication for sets of nodes corresponding to the physical structure of an Cray XC50™ system: blades, chassis, groups, and the whole system. The test is designed to run on as many nodes as are available, reporting variation in performance over sets of nodes of a given size. For example, one can run 512 instances of a blade level test on 2048 nodes and report variation between them.

VIII. WORKLOAD MANAGERS

The application level placement scheduler (ALPS) is a Cray developed and supported mechanism for placing and launching applications on compute nodes. ALPS provides application placement, launch, and management functionality and cooperates closely with third-party batch systems for application scheduling across Cray systems. The third-party batch systems make policy and scheduling decisions, while ALPS provides a mechanism to place and launch the applications contained within batch jobs.

ALPS application placement and launch functionality is only for applications executing on compute nodes. ALPS does not provide placement or launch functionality on service nodes.

In addition to ALPS, Cray supports customer choice for workload managers (WLMs). The following workload managers are supported:

- SchedMD®: Slurm® [14]
- Altair: PBSPro [15]
- Adaptive Computing: Moab/Torque [16]

Cray supports an homogeneous environment (matched x86-64 login/compute or aarch64 login/compute) for job launch. Multiple program multiple data (MPMD) is not supported at this time. If a WLM script executes anything on the login node, it must be the same architecture type (x86-64 or aarch64) as the intended compute nodes. A site with a mixed system should set up different queues based on the architecture type (x86-64 and aarch64).

IX. System Management

The Cray system management services (CSMS) provides system management tools and open source components to support the deployment and configuration of Cray's traditional HPC software stack. The current process used by Cray for creation of CLE images is based on the Image Management and Provisioning System (IMPS). The toolset supports the use of recipes, which ultimately define both a set of RPMs to install to create the desired image, as well as an optional set of additional commands to be executed within the created root file system (via *chroot*).

The IMPS tools including *imgbuilder*, *image*, *recipe*, *pkgcoll*, and *repo* collectively are used to describe image recipes and build those recipes into boot images. The IMPS image tools run on the SMW, which is x86-64 based.

To support the ability to create aarch64 images on an x86-64 based SMW, Cray included the quick emulator (QEMU) package from openSUSE on the SMW. Cray then extended a number of the IMPS tools (e.g. *recipe*, *pkgcoll*, and *image*) to understand an architecture type and to use QEMU to correctly build the image for the chosen architecture.

To create the full compute node image including Slurm for CLE 6.0 SP6 takes a little more than nine minutes. This is expected due to the high processing load during the emulated portions of the image builds.

X. Power Management

Power management for the Cray XC50™ system includes monitoring power and energy in a variety of ways and power capping (limiting the allowable power consumption of a system, or a part of a system). It includes specific features within HSS and CLE as well as a power management interface between CLE and HSS.

Cavium supports OOB maximum frequency limit in the M3 firmware. Cavium also support socket level power capping as compared to Intel which only supports node level power capping.

XI. Cray Programming Environment

The Cray XC50™ system with ARM processors includes a fully integrated Cray programming environment (CPE) with tools designed to maximize programmer productivity, application scalability, and performance. This feature-rich, easy-to-use programming environment facilitates the development of scalable applications.

The Cray XC50™ system with ARM processors provides for all of the common HPC programming models. It has full support for porting and developing distributed memory applications using MPI and Cray SHMEM. The Cray XC50™ system with ARM processors also supports the partitioned global address space (PGAS) programming models, Unified Parallel C (UPC), Coarray C++, and Fortran 2008 with coarrays, and shared memory programming models such as OpenMP and Pthreads within a node.

Cray MPI is derived from MPICH, an implementation of MPI-3 by the Argonne National Laboratory Group, which is highly optimized for the Aries interconnect. It includes full MPI-3.1 (or later version) support. MPI 3.0 introduced non-blocking collectives and RMA-3 one-sided communication that has been optimized for the Aries Network. The Cray MPI library is highly optimized for low latency and high bandwidth, both on-node and off node, for point-to-point and collective communication. Cray MPI has optimizations for the Aries network optional accelerators, and many core processors.

The DMAPP communication library is initially defined as part of the CLE Network software stack in section VI. DMAPP is typically not used directly within user application software. The DMAPP API allows one-sided communication libraries such as Cray SHMEM, and PGAS compilers such as Coarray Fortran/C++ and UPC, implemented on top of DMAPP, to realize much of the hardware performance of the Aries interconnection network while being reasonably portable to its successors.

Partitioned global address space (PGAS) is a parallel programming model. It assumes a global memory address space that is logically partitioned and a portion of it is local to each process, thread, or processing element. The PGAS model is the basis of Coarray Fortran/C++, Global Arrays, SHMEM, and Chapel.

The process management interface (PMI) [17] provides an interface between ALPS or native Slurm and Cray PE. PMI provides the basic interaction between the process and the resource manager. It also provides the mapping of the of ranks to nodes and cores.

The Cray XC50™ system with ARM processors Cray compiling environment (CCE) is the cornerstone innovation of the Cray adaptive computing paradigm. This compiler builds on a well-developed and sophisticated Cray technology base that identifies regions of computation that are either sequential scalar, vector parallel, or highly multithreaded. Programs can be statically or dynamically linked against performance-tuned scientific and runtime libraries.

Support for the 64-bit ARMv8 Application Binary Interface (ABI). There is full automatic vectorization support. For the 64-bit and 128-bit ARMv8 vector widths with a focus on 32-bit and 64-bit data. There is also vectorization of reductions, conditional code, and other idioms. Support for the commonly used ARMv8 intrinsics is also included.

The Cray XC50™ system with ARM processors includes a distribution of the GNU compiler collection (GCC) C, C++, and Fortran compilers. The GNU development tool chain and utilities that are provided as part of the standard SUSE distribution are also available.

Cray PE typically supports building libraries for three compilers per processor vendor. For Intel based systems

Cray PE supports CCE, GCC, and Intel. For aarch64 architectures Cray PE supports CCE, GCC, and ARM Clang/Flang [18] [19]. Note that support for third party compilers in this context means building Cray developed libraries MPT (Message Passing Toolkit) and LibSci, some third-party libraries including HDF (Hierarchical Data Format) [20], NetCDF (Network Common Data Form) [21], and FFTW (Fast Fourier Transform in the West) [22], and interoperability with the Cray Performance Analysis Tool, CrayPat, and Cray Apprentice2.

Cray compared 158 codes from standard benchmarks such as SPEP, NPB, and other specific benchmarks that are relevant to our customers. The comparison was between Cray CCE and the latest ARM v18.1 (LLVM) and gcc v7.3 looking at C, C++, and Fortran programming languages. Over 67% of the benchmarks executed faster using CCE when compared to LLVM and over 60% of the benchmarks executed faster when compared to gcc.

XII. Performance

There have been initial performance numbers for various system components outlined in various sections.

The initial performance numbers have been previously made available by Professor Simon McIntosh-Smith at Super Computing 2017 [23]. These numbers showed very positive results on the alpha hardware and software.

XIII. Lessons Learned

There have been various changes and lessons learned described throughout this paper. However, there are a few general suggestions for engineers when porting applications to the Cray XC50™ system with ARM processors system.

The first lesson learned is that the ARMv8 processor is less forgiving on alignment even with checking disabled. When modifying existing code, engineers were instructed to enable alignment checking. By default, Linux has alignment checking disabled. There were a number of changes in structures to change the declaration to a *long* instead of *int* which aligned the structure on a 64-bit boundary.

The second lesson learned was between the C and C++ *char* type defaults. The x86 *char* defaults to a *signed char* while the ARMv8 *char* defaults to *unsigned char*. Though the difference seems minor, it has proven to be a significant issue when porting applications from x86 to ARMv8. The command line option "*–h signedchars*" can be used to force ARMv8 to the x86 behavior.

Another lesson learned for some users was that some user applications used Intel's timing instruction directly through an *ASM()* intrinsic. This was replaced by either *gettimeofday()*, *MPI_wtime()*, or *omp_get_wtime()*. In the some of the Cray specific kernel features, Cray needed to use a generic timer instead of *gettime()*.

Finally, it is important to note that ARM is not sequentially (or processor) consistent while x86 is. But this is only an issue if the application is doing synchronization without using any hardware provided mechanisms (i.e. just using something like Dekker's or Peterson's algorithm which use shared memory for synchronization won't work).

XIV. Summary

Cray has successfully ported CLE to the new Cavium ThunderX2 ARM processor in the Cray® XC50™ system. The focus was on creating a system that was feature complete with comparable functionality to the Cray® XC™ systems. The port uncovered a number of areas that were specific to x86 architectures that needed to be updated to support multiple architectures.

This paper has stepped through the various components of CLE and described the functionality. It described the required changes to support the new aarch64 architecture whether it was a port or a replacement of specific functionality. In some cases, no changes were required. It also described the lessons learned along the way.

Cray has provided an ARM based system that provides equivalent functionality with other Cray® XC50™ systems in both usability, performance, and scalability.

References

1. Cavium ThunderX2 CN99XX Reference Platform Firmware Overview, Cavium, Revision 1.4, December 2017
2. Performance API, [Online], http://icl.cs.utk.edu/papi/index.html
3. Lustre filesystem, [Online], http://lustre.org
4. General Parallel File System (GPFS), [Online], https://www.ibm.com/support/knowledgecenter/en/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/bi_gpfs_overview.html
5. DataWarp User Guide. Cray Inc. [Online]. http://docs.cray.com/PDF/XC_Series_DataWarp_User_Guide_CLE60UP06_S-2558.pdf
6. ARM Cortex-A Series Programmer's Guide for ARMv8-A, Arm, [Online]. http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/index.html
7. Cavium ThunderX2 CN99XX – Reliability, Accessibility, and Serviceability Solution User Guide, Cavium, Revision 1.0, August 2017
8. Advanced Configuration and Power Interface Specification, Hewlett-Packard Corporation, Intel Corporations, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, Revision 4.0a, April 5, 2010
9. ARM CoreSight, [Online], https://www.arm.com/products/system-ip/coresight-debug-trace
10. Open On-Chip Debugger (OpenOCD), [Online], http://openocd.org/
11. GDB: The GNU Project Debugger, [Online], https://www.gnu.org/software/gdb/
12. J. Schutkoske, "Cray XC System Level Diagnosability: Commands, Utilities and Diagnostic Tools for the Next Generation of HPC Systems ", *Proceedings of the Cray User Group (CUG)*, 2014, https://cug.org/proceedings/cug2014_proceedings/includes/files/pap120.pdf
13. J. Schutkoske, "Cray XC System Node Level Diagnosability", *Proceedings of the Cray User Group (CUG)*, 2015, https://cug.org/proceedings/cug2015_proceedings/includes/files/pap130.pdf
14. SchedMD Slurm, [Online], https://www.schedmd.com/

15. Altair PBS Professional, [Online]. https://pbsworks.com/PBSProduct.aspx?n=PBS-Professional&c=Overview-and-Capabilities
16. Adaptive Computing MOAB/Torque, [Online], http://www.adaptivecomputing.com/products/open-source/torque/
17. PMI v2 API, Argonne National Laboratory, [Online], https://wiki.mpich.org/mpich/index.php/PMI_v2_API
18. Arm C/C++ Compiler, [Online], https://developer.arm.com/products/software-development-tools/hpc/arm-cpp-compiler
19. Arm Fortran Compiler, [Online], https://developer.arm.com/products/software-development-tools/hpc/arm-fortran-compiler
20. HDF5 Software Documentation, [Online], https://support.hdfgroup.org/HDF5/doc/index.html
21. Network Common Data Form (NetCDF), [Online]. https://www.unidata.ucar.edu/software/netcdf/
22. Fast Fourier Transform in the West (FFTW), [Online], http://www.fftw.org/
23. Professor Simon McIntosh-Smith, University of Bristol, UK, [Online]. http://www.goingarm.com/slides/2017/SC17/GoingArm_SC17_Bristol_Isambard.pdf