Configuration and Administration of a Cray CS 400 Heterogeneous Cluster with Bright Cluster Manager

Omar A. Morris
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
omar.morris@jsums.edu

Khalid H. Abed
Dept. of Electrical & Computer Engineering
Jackson State University
Jackson, MS. 39217 USA
khalid.h.abed@jsums.edu

Abstract—*This paper presents configuring and administering a Cray CS 400 heterogeneous cluster using Bright Cluster Management software. It details the architecture of the computational environment and the steps taken from initial assembly of the hardware to the installation of the various software layers that comprise the OS and system management tools. The Bright Cluster Manager is an enterprise level software suite that provides bare metal to fully functioning system management of High Performance Computing clusters and Big Data systems.*

1 INTRODUCTION

As our body of knowledge has increased so has our realization that there are ever more discoveries to be made. The digital tools used to make these discoveries and expand our understanding have evolved from analog to digital and are becoming increasingly complex. The growing demands on these computational systems are driving the traditional CPU to its physical limits. To address this problem, which is the interconnected relationship between the need for more powerful hardware and the software with the ability to exploit it to handle the growing complexity and vast amounts of data in modern applications and computational models, High Performance Computing (HPC) has emerged. HPC is the aggregation of computational elements in parallel to implement large-scale mathematically intensive applications in the areas of engineering, science, mathematics and business It is geared toward modeling complex physical phenomena such as climate change models, the 3-D mapping of proteins, military applications, and academic research.

The industry has turned to the use of accelerators or coprocessing units to increase the power and efficiency of standalone traditional CPUs. Intel developed a coprocessor capable of executing highly parallel numerically intensive applications quickly and efficiently; the massively parallel Xeon Phi Many Integrated Core (MIC), based on the x86 Intel P5 processors. The Xeon Phi environment utilizes all standard programing platforms and paradigms such as OpenMP, POSIX threads and MPI as well as high level languages like C++ and Fortran [1]. This allows the developer to work with familiar tools sets and languages and does not require massive rewrites of exiting code. That is not to say that achieving the desired speed up of application execution does not require effort and specialized knowledge. The developer must understand the underlying architecture of the Xeon Phi coprocessor to exploit its strengths and to determine if the platform is the correct one to port their code onto.

2 Environment

The system used in this research is a Cray CS 400 comprised of a single head node and six compute nodes. This section gives a description of the hardware specifications of the two types of nodes and the cluster's network topology.

2.1 The Head Node

The head node has an Intel Xeon E5-2650 v3 CPU, a Connect-IB (InfiniBand) Single Port QSFP, Fourteen Data Rate (FDR) adapter card and a ConnectX-3 EN10GbE Dual-Port SFP+ PCIe3.0 x8 8GT/s Network Interface Card. It also has an Intel RS2BL08D 8-port SAS RAID controller.

2.2 The Compute Nodes

Each compute node, named node001 – node006, has the Intel Xeon E5-2698 2.3GHz and dual Intel Xeon Phi 7120 passively cooled 1.25 GHz coprocessors. The Xeon Phi 7120 has 61 in-order 64-bit cores able to execute two instructions per cycle with 16GB of GDDR memory. The cores are fully functional computational units which allow the coprocessor to run its own Linux based operating system called uOS.

Each core has a 32KB L1 instruction cache, a 32KB L1 data cache, a 512KB L2 cache and a vector processing unit (VPU) that contains 32 512-bit registers capable of processing 16 single-precision or 8 double-precision floating-point arithmetic operations or 32-bit integers in parallel [2]. Each core is capable of multithreading and contains four hardware threads. Thus each of our compute nodes is capable of running 488 threads. The cores communicate with each other and maintain L2 cache coherency through the ring interconnect

with distributed tag directories. The bi-directional ring interconnect also allows the cores to access data and instructions from main memory via the memory controller [3]. Figure 1 illustrates the main components of the Xeon Phi's architecture.

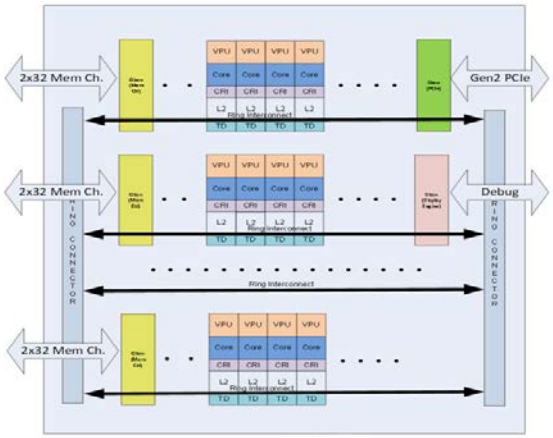

Figure 1: Xeon Phi Architecture [2].

The Xeon Phi communicates with its host processor via a Peripheral Component Interconnect Express (PCIe) bus. Because the card does not have any input and output options, all data travels through the PCIe interface and thus the PCIe bus is a source of data transfer overhead. Also, the Xeon Phi coprocessors run at approximately one third the rate of the Xeon host processors.

These factors: the large VPUs, the bi-directional ring interconnect, the PCIe bus and the 512-bit SIMD capability make it necessary to determine when and how to utilize the Xeon Phi MIC architecture.

2.3 Network Topology

The Jackson State University (JSU) cluster is connected via a10Gb Ethernet network and FDR InfinBand as shown in Figure 2.

The InfiniBand core fabric connects the compute nodes and enables high-speed data transfer between them. While the 10Gb and 1 Gb switches facilitate network management and are outward facing to the data centers switches and enable SSH communication remotely.

3 Software Installation

HPHC clusters require management software. Our software of choice is Bright Cluster Manager from Bright Computing. The recommended method of installing Bright Cluster Manager is on a bare metal system. A bare metal system is less prone to installation errors because it has no previous configuration and the OS is loaded on the machine during this process [4]. The Bright Computing management software is a robust and powerful set of tools that can seem to have a formidable learning curve for the complete novice to cluster computing and managing cluster environments in general and Bright Management software specifically. Once a new user familiarizes themselves with the various manuals and performs hands-on work with the software its power and ease of use becomes evident.

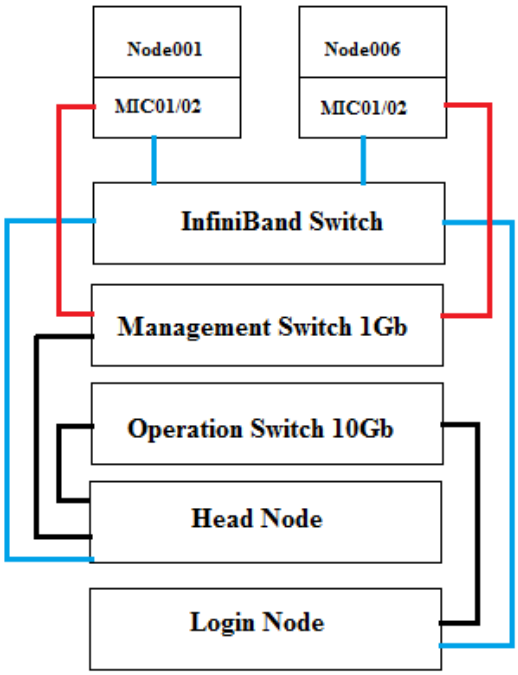

Figure 2: JSU CS 400 Cray HPHC Cluster.

We followed the add-on installation method because after receipt of our system we installed the OS in order to configure it for integration into the university's network. We installed Redhat Enterprise Linux Server version 6.5 (Santiago) kernel 2.6.32-431.el6.x86_64.

The add-on process initially lead to several mistakes and was a definite learning by trial and error experience. It required the removal of an improper install of Bright Management and starting over completely. This paper will not go into the many wrong paths we took but will emphasize the most valuable lessons learned in proceeding in an incorrect manner and will illustrate the final successful installation of Bright Cluster Manager. The most important lessons gleaned from the frustrating initial incorrect install is that Bright Cluster Manager must be installed via a package installer when the add-on method is chosen and that Bright Cluster Manager must be allowed to override any conflict with anything that is currently running on the system [4].

When installing Bright Cluster Manager, the system must have repository access to the distribution of the OS. In our case this required registering with Redhat Network (RHN) and obtaining a subscription to Redhat Subscription Manager. From the command line we entered:

```
# rhn_register
```

This command initiates the registration process and requires an account sign-in to the RHN portal after which the hardware and necessary package profiles are registered and transferred to RHN. Our system required the additional configuration of the *rhel-x86_64-server-6* and the *rhel-x86_64-server-optional-6* RHN channels.

After the registration is reviewed and accepted, the system requires a reboot and the Redhat daemon *rhnsd* starts. This allows the system to synchronize with Redhat and enables the *yum-rhn-plugin* which in turn makes it possible to use yum for repository access for updates and package installations.

To ensure registration completely successfully and that the system has full access to the repositories, we ran the command:

```
# yum repolist
```

This command generates a response that confirms the system's registration to Redhat Subscription Manager and that it has access to the base repositories needed to package downloads and updates.

Upon completion of registration Bright Cluster Manager, installation can begin. The Bright Cluster Manager package can be installed directly via DVD or ISO without the package installer but this will lead to errors that make the target system prone to less than complete or useful functionality. The *bright-installer 7.1* can be pulled from the DVD and then installed with rpm as root on the head node with the following command:

```
# rpm –ivh bright-installer-bright-129_cmbright.\
```

We were prompted to install several packages, which we did with yum and then began the install with:

```
#install-bright –n
```

This initiates an install with an internet connection. The installer will then alert to any software conflicts and take whatever action the user decides and then requests the user's license key. After inputting the key, the configuration stage is reached. The installer has default vales which we changed to the correct values for our network. After entering the interface values and addresses the Bright Cluster Manager packages are installed. After a successful install the user is told that the install is complete and they have successfully configured their cluster's head node.

3.1 Compute Node Image Creation

The compute nodes receive a software image from the image directory on the head node. Our image was the standard default image customized with the particular software packages we installed for our requirements. We created our custom image with this command:

```
#cm-create-image
```

We gave the custom image a relevant name and placed it with the default image in the images directory on the head node. Then, we set our new custom image as the default image to be provisioned to the compute nodes upon booting.

After powering up, the nodes receive the bootloader from the image distributed by the head node which loads the Linux kernel and allows the compute nodes to complete the boot process. The nodes boot from the network via the 10 GB Ethernet connection.

3.2 Many Platform Software Stack (MPSS)

The MPSS is the software that is required for the Xeon Phi to function. Installation is performed through the Bright software manager and was completed using YUM. The entire set of packages for our OS was installed using the following command:

```
#yum install intel-mic-*-rhel6.5 –
installroot=/cm/images/michost-image
```

This command was based on the version of the cluster's OS and the path and name of the custom compute node image. After the system installs MPSS it is then necessary to add the MIC environment variables with this command:

```
#module add intel/mic/runtime
```

After this executing this command, the MPSS is ready to configure the Xeon Phi coprocessors. This is done by using Bright's MIC configuration tool. Entering the following command invokes the tool:

```
#cm-mic-setup
```

The configuration tool can also be accessed through the Bright *Create MICs* GUI wizard. The JSU MICs were configured via the command line in interactive mode. During this process, the tool requested several pieces of information, including the addresses and various settings of the hosts and coprocessors that form the cluster. After the nodes are configured and integrated into the cluster, provisioning roles were assigned. In the case of the JSU cluster the head node kept its default role of boot and provisioning node. The size of the cluster did not require multiple boot nodes or nodes that need to handle

provisioning and image distribution to ordinary computational nodes. After the roles were assigned and images are set in the correct directories, the cluster was ready to be rebooted and set up for its first set of computation tasks.

3.3 Module Environment

After logging into the head node via SSH, it is necessary to complete several steps to prepare the cluster for program execution. Programs are stored centrally on the head node and are accessed by using the *modules environment* software package. This third party piece of software was installed with the Bright Cluster Manager install and works seamlessly with the Bright software. From the command line the list of available programs that can be shared across the nodes is accessed via the following command:

module avail

This command will list the available packages that the user can load once logged into a compute node, also accessed via SSH. To obtain the full set of modules this command is executed:

module load shared

This makes all programs that have been loaded onto the head node available. It is possible for the system administrator to create persistent profiles for users or to write shell scripts that specify modules to be loaded by default when a particular user completes a bash login.

3.4 Intel Compiler

The Intel compilers rely on environment variables to function properly. First, we needed to execute the setup script to configure the Linux runtime environment. We used ***compilervars.sh*** for BASH shell. We have to source this with the following command on the head node:

#source /opt/intel/compilers_and_libraries_2016/linux/bin/compilervars.sh intel64 or source opt/intel/bin/compilervars.sh.

We then checked to make sure that environment is properly set up by running:

#icc –V

Which if the compiler environment is active, it returns the version and build number.

Next it was necessary to start the flex license service on the head node to enable usage of the Intel compiler. This was accomplished by locating folder which holds the Flexlm license file which was: ***opt/intel/licenses***. In this folder, the lmgrd process was started with the following command:

#: /lmgrd – c

After executing this command, it is now possible to compile and execute software on the head node. However, the head node does not have MICs attached to it so our application execution was performed on node001. Several additional steps were required to allow this. From the head node's command prompt, we used SSH to login into node001.

No additional authentication is necessary once a user with sufficient privileges is logged into the system. Next the previously described command ***module load shared*** was run making available the necessary software packages. After the modules were made available, the following command was executed:

#module load intel/compiler/64/16.0.1/2016.1.150

This command loads the Intel Complier suite from tiger onto the node. The finals steps consisted of copying the Flexlm license file, creating both a directory for the Flexlm license file and an environment variable that points to the file location. The directory that was created matches the Flexlm directory on tiger and is also: ***opt/intel/licenses.*** A copy of the Flexlm license file was transferred to this directory from tiger with the secure copy command:

#scp flexlm.lic node001: opt/intel/licenses

After logging back into node001, the environment variable was set to point to the location of the Flexlm license with the following command:

#export INTEL_LICENSE_FILE=opt/intel/licenses

Upon completion of the aforementioned steps node001 was ready for program compilation and execution.

4 Research

An important step in evaluating new approaches in high performance computing hardware is selecting the correct testing applications. As it is often difficult to port full-scale production quality science and engineering applications to

new platforms, benchmarking tools are normally used. The benchmark tools (or kernels) are greatly reduced small code fragments that only contain the performance intensive computations of the full-scale application [5].

The full-scale applications are, of course, the best performance indicators, but they are usually too large or complex for use in the beginning stages of hardware development and may require a well-trained and experienced software developer to port them to a new system. For this reason, they are usually not implemented until the major design decisions are undertaken and the system is nearing production levels of maturity. However, this presents a problem. Crucial design designs must be made in the beginning stages rather than at the end, and computational kernels and benchmarking tools are becoming more inadequate as accurate measurers of system performance as the platforms and the data sets grow more complex. Developers have noticed an increasing gap between benchmark results and actual application performance on production-level HPC platforms [6].

Factors that affect performance should be defined and understood as early as possible in the development of new hardware systems. The necessity to address the need for software between the two poles of inadequate benchmarking tools and full-scale applications that will enable developers to test their designs and make informed decisions has led to the exploration and development of scaled down versions of production level applications.

4.1 The Mantevo Project

These smaller versions contain the performance-intensive computational kernels of the full-scale applications, like benchmarks, but they also contain the context of the computational components of the full application and include libraries wrapped in a test driver providing representative inputs [4]. These compact self-contained proxies, called mini-apps (proxy application), were developed under the auspices of the Mantevo Project led by Michael Heroux and Richard Barrett of Sandia National Laboratories. The Mantevo Project has several collaborators. Among them are: Livermore computational physicists David Richards and James Belak, various researchers from Los Alamos and Sandia national laboratories, NVIDIA Corporation, and three British institutions—the University of Bristol, the University of Warwick, and the Atomic Weapons Establishment [5]. Mantevo mini-apps are reliable and accurate predictors of application and system performance. The mini-apps average 5,000 lines of code as opposed to many full-scale applications which can easily contain over a million lines of source code. As such, they are much easier to understand, deploy, change or rewrite and are becoming widely used in HPC design because their implementation requires a fraction of the time, effort and training that porting a full-scale application demands.

4.2 MiniMD

MiniMD is a simplified, miniature version of the popular Sandia National Laboratories Large-scale Atomic Molecular Massively Parallel Simulator (LAMMPS) program. MiniMD's source code is less than 3,000 lines as compared to the full-scale size of LAMMPS which is over 130,000 line of C++ [4]. Similar to LAMMPS MiniMD uses spatial decomposition MD. The user can also specify several parameters including problem size, temperature, atom density and number, timestep size and number of timestep, potential cut-off, skin distance and frequency of Neighbor List rebuilds. These parameters are defined in an input file prior to compilation [7].

Unlike LAMMPS, MiniMD only supports the Lennard-Jones (LJ) inter-atomic potential. No other pair interaction features such as long-range electrostatics or molecular force field calculators are available. Combining these features would have made MiniMD unnecessarily complex and too unwieldy for a novice to MD and would have also resulted in a program difficult to port to new hardware for testing purposes. Despite MiniMD's reduced size, it performs its role as well as, and in many cases better than, much larger and more complex programs.

4.3 Porting MiniMD to the Intel Xeon Phi™

The primary reason for obtaining the Cray CS 400 cluster was to perform research on HPHC by analyzing and comparing the performance of the miniMD mini application on a conventional multicore Xeon CPU with its execution on the Intel Xeon Phi architecture. The objective is to achieve and demonstrate a significant increase in speed and efficiency in runtime on the MIC hardware over the stand-alone CPU.

The research consisted of executing the algorithm in several configurations: entirely on a conventional multi-core Xeon processor; in native mode where computation takes place entirely on the Xeon Phi coprocessor; in offload execution or heterogeneous programming mode in which the host CPU offloads all or part of the data from one or several host threads to the Xeon Phi coprocessor. Computation starts on the host and as it moves forward the host can send the data to the Xeon Phi coprocessor where the coprocessor and the host can work on it in parallel.

5 Conclusion

The scope of this paper is not to discuss the performance of the cluster but rather to give an overview of the process and software used to configure and manage it. What is relevant to the focus of this paper is that Bright Cluster Manager worked seamlessly with Intel's software and drivers. Future work will detail actual cluster performance in addition to improving the install and administration process.

6 Acknowledgements

This work was supported in part by the U.S. Department of Defense High Performance Computing Modernization Program under the U.S. Army Engineer Research and Development Center (ERDC) contract number W912HZ-15-2-0001 entitled: "*Strategic Cyber Science, Warfare, Security, Application Development and High Performance Computing Research and Development,*" and the research project is entitled: "*Investigating High Performance Heterogeneous Computing with Advanced Architectures*," and in part by Army Research Office HBCU/MSI contract number W911NF-13-1-0133 entitled: "*Exploring High Performance Heterogeneous Computing via Hardware/Software Co-Design.*"

7 REFERENCES

[1] T. Cramer, D. Schmidl, M. Klemm and D. an Mey, "OpenMP Programming on Intel® Xeon Phi™ Coprocessors: An Early Performance Comparison," Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University , pp. 38-44, November 2012.

[2] Intel. *Intel Xeon Phi Coprocessor System Software Developers Guide.* [Online]. Available: http://www.intel.com/design/literature.htm, March 2014.

[3] R. Rahman *Intel Xeon Phi Coprocessor Architechture and Tools.* New York: Apress, 2013.

[4] Bright Computing, *Bright Cluster Manager 7.1. Installion Manual.* [Online] Available: www.brightcomputing.com: http://support.brightcomputing.com/manuals/7.1/installation-manual.pdf. December, 2015.

[5] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. *Mantevo Overview.* [Online] Avaialable: Mantevo: https://mantevo.org/MantevoOverview.pdf. September, 2009.

[6] R. Hansen, *Mini-apps Accelerate Hardware and Software Development.* Science and Technology Review. October 2013.

[7] S. J. Pennycook, C. J. Hughesy, M. Smelyanskiyy and S. A. Jarvis "Exploring SIMD for Molecular Dynamics, Using Intel® Xeon Processors and Intel® Xeon Phi™ Coprocessors." 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, pp. 1085-1097, 2013.