Early experiences with the Cray XK6 hybrid CPU and GPU MPP platform

Sadaf R Alam
Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland
e-mail: alam@cscs.ch

Jeffrey Poznanovic
Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland
e-mail: poznanovic@cscs.ch

Ugo Varetto
Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland
e-mail: uvaretto@cscs.ch

Nicola Bianchi
Swiss National Supercomputing Centre (CSCS)
Lugano, Switzerland
e-mail: nbianchi@cscs.ch

Antonio J. Peña
Universitat Politècnica de València (UPV)
Valencia, Spain
e-mail: apenya@gap.upv.es

Nina Suvanphim
Cray Inc.
Lugano, Switzerland
e-mail: nina@cray.com

***Abstract*— We report on our experiences of deploying, operating and benchmarking a Cray XK6 system, which is composed of hybrid AMD Interlagos and NVIDIA X2090 nodes and the Cray Gemini interconnect. Specifically we outline features and issues that are unique to this system in terms of system setup, configuration, programming environment and tools as compared to a Cray XE6 system, which is based also on AMD Interlagos (dual-socket) nodes and the Gemini interconnect. Micro-benchmarking results characterizing hybrid CPU and GPU performance and MPI communication between the GPU devices are presented to identify parameters that could influence the achievable node and parallel efficiencies on this hybrid platform.**

Keywords-MPP systems, Cray XK6, Cray XE6, accelerators, programming environment, performance characterization

I. INTRODUCTION

This manuscript provides an overview of the characteristics and features of the Cray XK6 system, which are critical for not only porting existing GPU accelerated applications but also fundamental for achieving high node and parallel efficiencies. The Cray XK6 is a hybrid CPU and GPU massively parallel processing (MPP) platform that incorporates the Cray Gemini interconnect, and NVIDIA X2090 accelerated compute nodes [12]. The system is considered as a first instance of a tightly integrated, accelerator based MPP platform as it offers not only an integrated node design but also an integrated programming and system management environment for efficient development and deployment of GPU accelerated applications. At the same time, the system is seamlessly integrated within the Swiss National Supercomputing Centre (CSCS) operational and job accounting environments alongside other Cray MPP production platforms.

Therefore, in addition to programming and performance characteristics, we provide deployment and operational considerations that are essential for managing a large-scale hybrid system. At CSCS, two Gemini interconnect based systems have been deployed recently: a 16-cabinet Cray XE6 system with dual-socket AMD Interlagos nodes [11] and a 2-cabinet Cray XK6 system with hybrid AMD Interlagos and NVIDIA X2090 GPU accelerator devices within a node. Both systems have 32 Gbytes per node memory, a Lustre scratch file system and have the SLURM resource manager. We identify unique features of the Cray XK6 system with respect to the Cray XE6 platform.

The outline of the report is as follows: an overview of the key architectural and programming features of the Cray XK6 platform is provided in section II. In section III, unique elements of Cray XK6 programming environment including code development tools, and operational considerations and issues are detailed. Benchmarking results that highlight the unique features of the Cray XK6 system are presented in section IV along with some a brief status update on applications development and porting efforts on the Cray XK6 platform. A summary of the paper and future plans for the Cray XK6 platform and its successor system are listed in section V.

II. ARCHITECTURAL AND PROGRAMMING ENVIRONMENT

In this section, we provide details of CSCS Cray XK6 and Cray XE6 platforms for architectural specifications, programming and operating environment, and management and operations of the system.

A. Processing Node

A Cray XK6 processing node is composed of an AMD Interlagos 6272 socket, which has 16 cores constituting 8 compute modules [9].

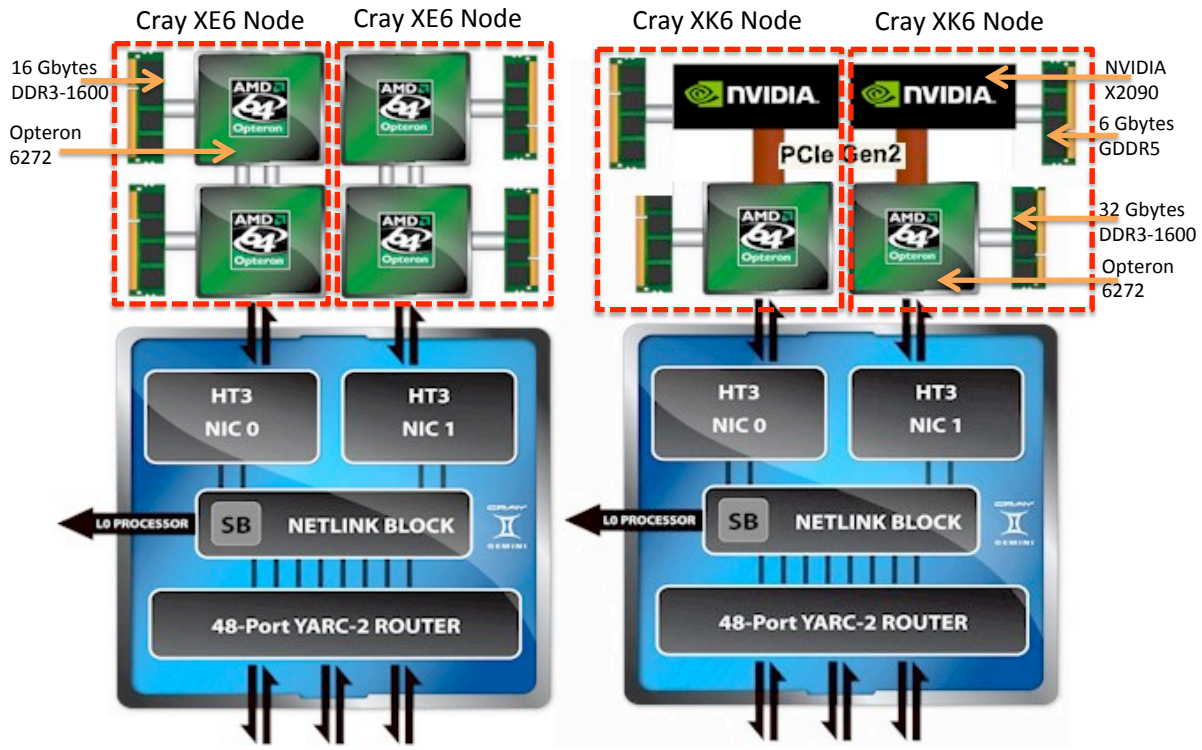

Figure 1. Layout of Cray XE6 and Cray XK6 nodes. Two nodes are connected to a Gemini interface.

The Cray XE6 has two AMD Opteron 6272 sockets per compute node. As an accelerator device, the Cray XK6 system has an NVIDIA Fermi X2090 GPU. Figure 1 shows two nodes of a Cray XE6 system and an XK6 system side by side. Both systems have 32 Gbytes of DDR3-1600 memory while the Cray XK6 system has an additional 6 Gbytes of GDDR5 GPU memory.

B. *Interconnect*

The Cray XE and XK series systems are based on the Cray Gemini interconnect. As shown in figure 1, two nodes are connected to a Gemini chip via a Hypertransport (HT3) interface. Both systems have a 3D torus topology: CSCS Cray XE6 system has Class 2 – 3D Torus and the XK6 has Class 0 – 3D Torus topology.

C. *Operating System*

Cray Linux Environment (CLE), which is a lightweight variant of Linux, has been installed on both systems. The systems are used in a cross-compile mode where typically code development has been done on a frontend system without CLE (using regular Linux) and GPU devices.

D. *Program Development and Execution Tools*

Code development utilities, for example, compilers, debuggers and performance measurement tools for the x86 microprocessors are similar on both platforms. For example, Cray, PGI, Intel, GNU and Pathscale compilers are available on both platforms. The Cray XK6 system has additional compilers and runtime drivers for GPU execution. Both systems have a similar job launching interface namely Application Level Placement Scheduler (ALPS), which allows users to specify mapping and control of parallel MPI and OpenMP jobs.

E. *Numerical libraries*

Tuned and optimized versions of math libraries including BLAS, Lapack, and FFTW are available as part of the Cray scientific libraries (libsci). A subset of GPU-accelerated APIs is available for the Cray XK6 platform.

F. *Communication libraies*

An optimized version of message-passing (MPI) communication for the Gemini interconnect has been provided as part of the Cray programming environment. In addition, compilers for PGAS languages (Coarray Fortran and UPC) are supported by the Cray compilation environment (CCE) [4].

G. *Job scheduler and resource accounting*

CSCS has deployed the SLURM resource management system on both platforms, which CSCS has developed for the Cray platforms [5]. Currently, the minimum allocation unit is a processing node of a Cray XK6 and XE6 node. Although users can specify core and memory requirements through the job scheduler script, there has been no mechanism in ALPS to identify accelerator resources.

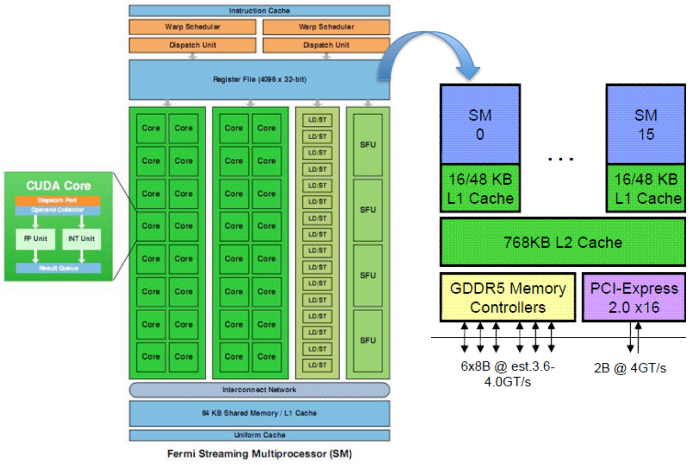

Figure 2. NVIDIA Fermi architecture. A streaming multi-processing (SM) unit has 32 cores and there are 16 such SMs in a Fermi X2090 device.

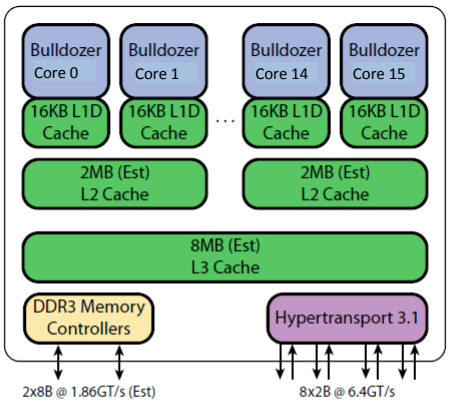

Figure 3. AMD Interlagos block diagram. Two Bulldozer cores form a compute module to share two 128-bit FMAC units.

Cray has extended system management and diagnostics tools for the GPU devices. Cray Linux Environment (CLE) provides a Node Knowledge and Recognition tool called NodeKARE, which performs node health checks [6]. For example, the node health checker takes into consideration the status of the GPU devices be checking whether the device is in a healthy state and that all the memory has been cleared and available to the next process. In other words, on a Cray XK6 system, additional tests are performed to verify nodes are healthy and ready to run jobs prior to usage

III. UNIQUE ELEMENTS OF CRAY XK6 SYSTEM

Previous section highlighted several common design and programming elements of the Cray XE and XK series MPP platforms, which could enable a smooth transition from a multi-core based node to an accelerator-based system. However, we (at CSCS) experienced a few issues as the first site to install such a system with an accelerator device, which is considered as an I/O component from the operating system viewpoint. In this section, background to the unique elements of the Cray XK6 platform is provided.

A. *Compute Units*

The characteristics feature of a Cray XK6 node is an accelerator device: NVIDIA X2090 Fermi GPU. Table I provides a comparison with AMD Interlagos 6272 Opteron processors.

TABLE I. COMPUTE UNIT SPECIFICATIONS OF GPU AND HOST CPU (OPTERON 6272) OF A CRAY XK6 NODE. TOTAL COMPUTE PERFORMANCE PER NODE IS 665+269 GFLOPS

	Fermi X2090	Opteron 6272
Cores	512	16
Clock frequency	1.15 GHz	2.1MHz
Floating point performance	665 GFlops (double-precision)	134.4 GFlops (double-precision)
Memory interface	GDDR5	DDR3 (1600)
Power envelope	225-250 W	90-115 Watts

B. *Memory Subsystems*

The two compute elements have distinct memory address spaces and offer distinct memory hierarchies in terms of the cache levels and sizes. Table II lists the characteristics of these compute unit hierarchies. Note that the minimum level of sharing for the NVIDIA devices is a streaming multiprocessor (SM), which is composed of 32 cores, and the L1 cache and shared memory are configurable by user software. On Fermi, 32,768 x 32-bit register files are also shared by the compute cores. Register files are dynamically allocated to threads executing on compute cores, this is the space where variables local to a thread are stored. Hence, these are considered "shared" (Table II) as there is no fixed configuration that assigns register files to specific cores.

TABLE II. CACHE AND MEMORY HIERARCHY OF THE NVIDIA FERMI X2090 AND OPTERON BULLDOZER 6272 PROCESSORS

	Fermi X2090	Opteron 6272
L1 cache (size)	16-48 KB	16 KB
L1 (sharing)	SM (32 cores)	Core
L2 cache (size)	768 KB	2028 KB
L2 (sharing)	All SMs	Module (2 cores)
L3 cache	--	8 MB
L3 (sharing)	--	Socket
Shared memory	16-48 KB per SM	--
Global memory	6 GB	32 GB

The AMD Opteron memory hierarchy also includes non-uniform memory access (NUMA) regions on socket as well as on a node. Figure 4 shows NUMA layout of a Cray XE6 node with 4 NUMA regions, 2 per socket. On the Cray XK6 platform, there are 2 NUMA regions, hence the impact of the memory placement is less significant as compared to a Cray XE6 node.

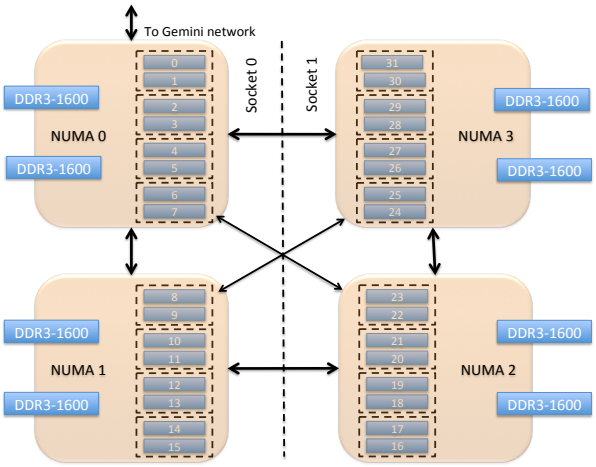

Figure 4. NUMA layout of a Cray XE6 node. There are 2 NUMA regions per AMD Opteron 6272 socket, therefore, a Cray XK6 node has 2 NUMA regions. Memory transfers take place over the Hypertransport links.

C. *Programming Languages (CUDA & OpenCL)*

CUDA is an interface for programming NVIDIA GPU devices that has been introduced by NVIDIA [3][13]. Both C and Fortran interfaces are available. An offload model has been proposed by CUDA where compute-intensive code blocks are executed on the GPU devices but the host does the control and data orchestration. CUDA proposes a data parallel programming model and a code developer can identify and develop parallel kernels using a few constructs. Likewise, there are constructs for specifying data transfer operations to and from host memory to the device memory. The CUDA programming model also describes a detailed memory model with register, local and global shared memories, which programmers can exploit in order to tune and optimize their codes on the device. In practice, most of tuning and optimization efforts using the CUDA model are centered on minimizing overheads of host and device transfers as well as localizing memory operations within the CUDA memory hierarchy. Recently, an x86 compiler has been introduced for CUDA codes by the PGI compiler. Hence, the codes written for the GPU devices can now be executed on CPUs [16]. The CUDA C compiler from NVIDIA (nvcc) and the CUDA Fortran compiler from PGI are available on the Cray XK6 platform.

Currently, an older version of CUDA, version 4.0 instead of version 4.1, is available on the CSCS Cray XK6 platform. There are a number of dependencies that have been delaying the update process. Earlier the issue was the availability of an appropriate driver version as the Cray platform installs a special version of the NVIDIA CUDA drivers. Recently, the driver has been updated to support CUDA 4.1 but the Cray PE has some other dependencies. Once resolved, version 4.1 will be available. It is expected this process will be streamlined for future releases of CUDA and OpenCL programming updates.

OpenCL is a set of open standards that have been introduced for developing programs for systems with heterogeneous compute units [18]. Hardware vendors provide the standard conformant drivers. Hence, OpenCL codes can be executed on both CPU and accelerators. The programming model allows for both data and task parallelism. Like CUDA, there is a concept of parallel programming for a device where concurrent tasks can be grouped into work-items. OpenCL memory model is also somewhat similar to the CUDA memory model (mainly due to the underlying architectural characteristics), where memory access options depend on how a data structure has been declared.

D. *GPU Drivers and Runtime (CUDA & OpenCL)*

On the Cray XK6 platform, in order to execute applications on the GPU devices, NVIDIA CUDA and OpenCL drivers and runtimes are required. For CUDA codes, the CUDA runtime and the driver API, both provided by NVIDIA, are available on the backend compute nodes of the system as the frontend or login nodes do not contain a GPU device. In the case of OpenCL everything (compiler and implementation) is in the driver, so there is basically one single driver for OpenCL and CUDA and two separate libraries. The OpenCL code is then just-in-time-compiled from C. The CUDA code has to be precompiled through nvcc to either binary or ptx.

An example of a unique issue we experienced soon after the system became operational was the OpenCL availability. A header file needed for OpenCL code was not installed on the XK6 compute node at the default location even though complete CUDA SDK was installed as part of the Cray PE. This issue was resolved by copying the file manually.

On the Cray XK6 platform, CSCS also installed an OpenCL driver for execution on the CPUs as this has not been made available as part of the Cray PE.

E. *Incremental GPU programmning using OpenACC*

Both CUDA and OpenCL programming approaches require fundamental changes to existing CPU-only code. In order to facilitate an incremental adoption of the accelerator devices, a few directives based standards have been introduced [1][17]. Cray compiler environment (CCE) provides support for the latest standard for accelerator programming called OpenACC. The OpenACC directives provide control for the following functionalities: regions of code to accelerate, data to be transferred to and from the device, and compiler hints for loop scheduling and cache usage. An example of an OpenACC accelerator region is below:

```
!$acc parallel loop vector_length(NTHREADS)
!$acc& private(x1,y1,i1,i2,i3,x2,y2) present(r,s)
      do  j3=2,m3j-1
        i3 = 2*j3-d3
        do  j2=2,m2j-1
          i2 = 2*j2-d2

          do j1=2,m1j
            i1 = 2*j1-d1
            x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >             + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
            y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >             + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
          enddo

          do  j1=2,m1j-1
            i1 = 2*j1-d1
            y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >         + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
            x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >         + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
            s(j1,j2,j3) =
     >             0.5D0 * r(i1,i2,i3)
     >           + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >           + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
     >           + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
          enddo

        enddo
      enddo
!$acc end parallel
```

The code can be compiled using an additional module (craype-accel-nvidia20) and can be executed using the aprun command without any additional flags for GPU execution. Hence, from a user point of view, there have been no changes in the execution model for a Cray XK6 node as compared to a Cray XE6 node. The only restriction is the number of MPI tasks. Only one MPI task can be executed per node when the code is GPU accelerated, as the NVIDIA driver setting on the Cray XK6 nodes has been configured to support a single processor accessing the GPU (exclusive access mode).

F. Accelerated libraries

Another critical component of the Cray integrated environment for GPU code development is availability of the accelerated numerical libraries. This includes a subset of BLAS and LAPACK functions [10][14]. Additionally, some functions have been optimized to run in hybrid host multi-threaded and GPU accelerated configurations. The library has been designed to work in different modes: it can work without any code modifications where data transfers to and from GPU are hidden from users; and it allows modifications to enable code developers to hide data transfer latencies. For example, a user can make the following call and it will be executed on the GPU if the matrix sizes are larger:

```
dgetrf (M, N, A, lda, ipiv, &info)
```

if instead of CPU, the GPU device pointers are being passed the code will execute on the device:

```
dgetrf (M, N, d_A, lda, ipiv, &info)
```

Data must be transferred to the GPU prior to the call to improve performance.

The libsci accelerator interface can also be invoked within the directives environment.

```
!$acc data copy(c), copyin(a,b)
!$acc host_data use_device(a,b,c)

call dgemm_acc('n','n',m,n,k,alpha,a,lda,
& b,ldb,beta,c,ldc)

!$acc end host_data
!$acc end data
```

G. Code Development Tools

The code development process involves bug fixing and tuning for optimization and performance. On Cray systems, there are third part debugging tools from TotalView and Allinea DDT for MPI and OpenMP programming [8][20]. Likewise, for performance measurement and tuning, Cray performance analysis toolset (perftools) are available for parallel MPI and hybrid MPI and OpenMP applications.

Both TotalView and Allinea DDT have introduced CUDA debugging features in their tools; however, DDT supports OpenACC officially.

> One of the most critical issues on the Cray XK6, right from the beginning, has been availability of a debugger for parallel, GPU-accelerated applications. Features that are unique to the Cray XK6 platform have been the root cause of some of the issues. Problems arise when MPI and debug processes try accessing the device simultaneously, which is setup as the exclusive access mode (as per NVIDIA recommendation). At the time of writing this paper, a fix has been made available and it is being verified.

Similar issues have been recorded and reported to both Cray and NVIDIA regarding seamless support of performance measurement tools. Cray performance tools can measure performance of the OpenACC code regions but it has not been straightforward to measure performance of MPI and CUDA applications without manually editing the source files. This issue has now been resolved.

> The GUI based performance tool from NVIDIA for performance profiling currently is not functional on the Cray XK6 system. This is because the data has been collected on the XK6 compute nodes that have the NVIDIA X2090 GPU devices, while the front-end and login nodes are without them. It is not possible to connect to the compute nodes through ssh like other commodity clusters.

```
> aprun -n 1 ./deviceQuery
[deviceQuery] starting...
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Found 1 CUDA Capable device(s)

Device 0: "Tesla X2090"
  CUDA Driver Version / Runtime Version          4.10 / 4.0
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 5375 MBytes (5636554752 bytes)
  (16) Multiprocessors x (32) CUDA Cores/MP:     512 CUDA Cores
  GPU Clock Speed:                               1.30 GHz
  Memory Clock rate:                             1848.00 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and execution:                 Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Concurrent kernel execution:                   Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support enabled:                Yes
  Device is using TCC driver mode:               No
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           2 / 0
  Compute Mode:
     < Exclusive Process (many threads in one process is able to use ::cudaSetDevice() with this device) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 4.10, CUDA Runtime Version = 4.0, NumDevs = 1,
Device = Tesla X2090
[deviceQuery] test results...
PASSED
```

Figure 5. Output on the device query test showing several aspects of the device setup. This data has been collected after the latest driver version update, which is in prerequisite for CUDA 4.1 installation.

IV. BENCHMARKING RESULTS

CUDA SDK provides a list of micro-benchmarks and examples for validating the device configuration and these can also serve as primitive diagnostic tools, which code developers can use to verify whether compiler and driver versions and other settings. Output from the device query test is shown in figure 4. The first piece of information it reports is the device is CUDA capable and it has been identified correctly by the runtime (X2090). Second, we observe that the driver version installed support the latest releases of CUDA compilers but later we note that the current runtime is 4.0. Other device settings, such as the support for features like pinned memory and unified virtual address have been listed in the output.

The amount of GDDR5 memory has been shown in figure 4 as well. Since the ECC is enabled, it is less than 6 Gbytes. The total number of CUDA cores is also reported together with the clock frequency. This confirms the layout shown in figure 2. Additionally, the memory hierarchy information is reported including the size of cache and memory bandwidth information. The maximum number of threads and memory per block are also indicated. The devices access mode (exclusive) is also mentioned in the output.

Another restriction of the Cray XK6 platform is that only a single MPI task can be assigned to a processing node for an accelerated application. This is due to non-exclusive device setting. Some applications, e.g. NAMD (www.ks.uiuc.edu/Research/namd), require support of multiple MPI tasks in order to achieve maximum efficiency per node.

CSCS has setup 8 nodes with non-exclusive mode where multiple MPI tasks can share the GPU device. On these devices the output is

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

```
> aprun -n 1 ./stream
[Double-Precision Device-Only STREAM Benchmark implementation in CUDA]
./stream Starting...

 Device 0: Tesla X2090
 Array size (double precision) = 8000000
 using 128 threads per block, 62500 blocks
 device STREAM_Copy:                  Pass
 device STREAM_Copy_Optimized:        Pass
 device STREAM_Scale:                 Pass
 device STREAM_Scale_Optimized:       Pass
 device STREAM_Add:                   Pass
 device STREAM_Add_Optimzied:         Pass
 device STREAM_Triad:                 Pass
 device STREAM_Triad_Optimized:             Pass
Function    Rate (MB/s)    Avg time      Min time      Max time
Copy:      133515.7922     0.000960     0.000959     0.000964
Copy Opt:  133779.2629     0.000958     0.000957     0.000961
Scale:     133457.8880     0.000960     0.000959     0.000962
Scale Opt: 133721.1137     0.000958     0.000957     0.000958
Add:       131538.5506     0.001460     0.001460     0.001461
Add Opt:   131532.7913     0.001461     0.001460     0.001462
Triad:     131478.0247     0.001461     0.001460     0.001462
Triad Opt: 131478.0247     0.001462     0.001460     0.001465

[streamBenchmark] - results:   PASSES

./stream Exiting...
```

Figure 6. GPU memory bandwidth results using CUDA version of the Stream benchmark. The benchmark has been developed by M. Fatice (NVIDIA). The peak bandwidth of the device is ~178 Gbytes/sec. Note that no data transfer between the host and device takes place for the benchmark.

The floating-point capabilities of the device can be measured by executing the libsci version of the DGEMM benchmark, which has a tuned and optimized version for:

- Multi-core host CPU (AMD Opteron 6272)
- NVIDIA Fermi X2090 GPU
- Hybrid Multi-core + GPU

The peak performance of the host processor and the device are listed in Table 1. Using rather large matrix sizes, for example, a 10,000 x 10,000 matrix, the following double-precision GFLOPS rates have been achieved:

- Host only = ~ 100 GFlops
- GPU only = ~ 360 GFlops
- Host + GPU = ~ 440 GFlops

Using even larger matrices, over 450 GFlops rates can be observed. For the memory bandwidth between host and device and the device and host, there are two configurations. The first one is called pageable, where the pages belonging to the GPU device can be swapped by the operating system. There is another option called the pinned memory where pages belonging to the devices are locked in the host memory. The impact on performance is obvious from the results in table III (CUDA SDK memory bandwidth test is used for bandwidth measurements):

TABLE III. MEMORY TRANSFER BANDWIDTH FOR TWO DIFFERENT MESSAGE SIZES, 1024 BYTES AND 32 MBYTES, FOR TWO DIFFERENT HOST MEMORY ALLOCATION SCHEMES (PAGEABLE AND PINNED)

	Pageable	Pinned
H->D (1024 Bytes)	33.6 MB/s	195.3 MB/s
D->H (1024 Bytes)	32.7 MB/s	276.2 MB/s
H->D (32 Mbytes)	2266.8 MB/s	5518.9 MB/s
D->H (32 Mbytes)	1975.7 MB/s	6273.2 MB/s

In order to measure the memory bandwidth on the device, we executed CUDA version of the stream benchmark [19]. This benchmark measures main memory bandwidth for simple, single-strided operations (Figure 6). Due to the NUMA layout of the microprocessor memory, depending on the access patterns, an application can exhibit sensitivity to the placement of MPI tasks and OpenMP threads on Cray XE6 and Cray XK6 nodes. Table IV performance of a multi-threaded version of the stream benchmark can yield a range of results, depending on the placement and binding of memory and threads.

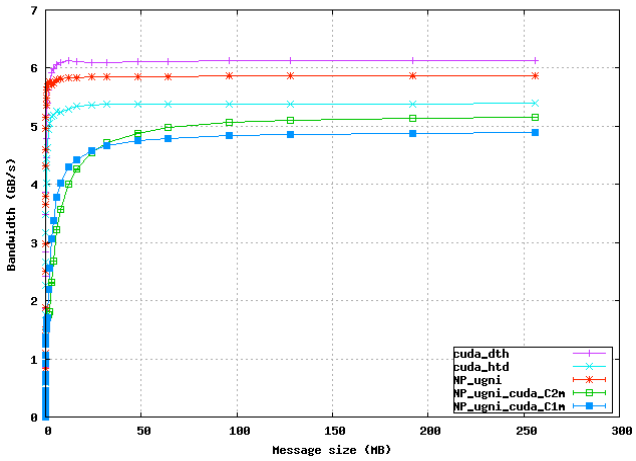

Figure 7. Characterization for GPU-GPU MPI transfers, bandwidth for host and device transfers and data transfers between two Cray XK6 nodes on the Gemini interconnect are presented.

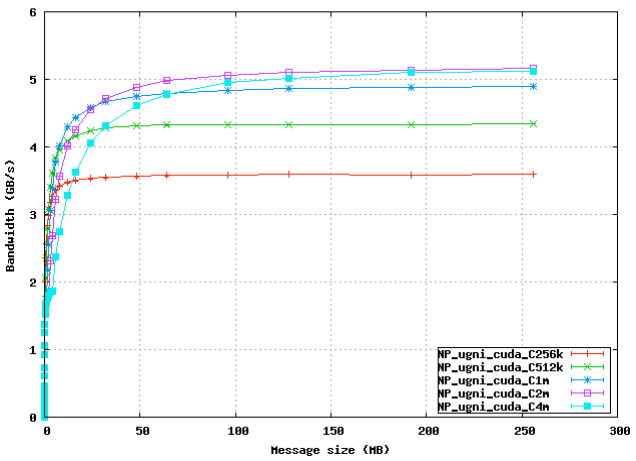

Figure 8. Impact of different block sizes are measured for using different blocking sizes using a NetPIPE benchmark that has been extended for GPU and uGNI.

TABLE IV. RESULTS OF THE STREAM MEMORY BANDWIDTH BENCHMARK (COPY OPERATION) WITH DIFFERENT NUMA MAPPINGS. THE THEORETICAL PEAK OF ONE NUMA MEMORY IS 25.6 GB/S. NOTE THAT UP TO THREE THREADS CAN SATURATE THE BANDWIDTH AND ALL RESULTS DEMONSTRATE HIGH SENSITIVITY TO THE MEMORY AFFINITY.

Number of threads	Thread placement	Memory binding	Bandwidth (MB/s)
1	Core 0	NUMA 0	12900.58
1	Core 0	NUMA 1	9013.58
1	Core 0	NUMA 2	6418.12
1	Core 0	NUMA 3	8894.06
3	Core 0-2	NUMA 0	15527.98
3	Core 0,2,4	NUMA 0	16832.34
4	Core 0-3	NUMA 0	16647.57
4	Core 0-3	NUMA 1	10451.38
4	Core 0,2,4,6	NUMA 0	16641.58
4	Core 0,2,4,6	NUMA 1	10512.12
8	Core 0-7	NUMA 0	15691.73
8	Core 0,2,4, 6,8,10,12,14	NUMA 0,1	33267.49
16	Core 0-15	NUMA 0,1	31403.30

Since the Cray XK6 system has been developed as a tightly integrated MPP platform with accelerators, we performed a number of tests for estimating GPU to GPU transfer rates over the Gemini interconnect. Currently, MPI calls cannot be made within the GPU kernels or pointers using the Cray MPI. Code developers are therefore responsible for copying data to and from the devices and MPI the host CPUs then invokes calls.

On the Cray XK6 platform, once CUDA module is loaded, the dynamically linked version of the MPI library is automatically linked instead of the static version. Some applications experienced significant slowdown in this mode and this issue has been under investigation. A temporary work around has been provided by Cray.

Latencies and bandwidth between the host and the CPU and between two hosts are highly sensitive to message sizes, and how the GPU memories are being declared (paged vs. non-paged). We have extended an interconnect micro benchmark called NetPIPE to measure the impact of these features [15]. The benchmark has been extended using the low-level Gemini network API called uGNI, and also inter-node GPU transfers using CUDA. On these transfers GPU and network transfers are pipelined, and different pipeline chunk sizes have been explored.

Figure 7 shows the results of experiments that have been performed to characterize data transfer latencies and bandwidths from a GPU on one node to another GPU over the Gemini interconnect.

This involves measuring the following paths independently:

- Device to host (cuda_dth)
- Host to device (cuda_htd)
- Between hosts over the interconnect (NP_ugni)

The effective bandwidth cannot be higher than the lowest bandwidth of the above three. In other words, the slowest path will limit the bandwidth. The results in figure 7 confirm this hypothesis. For large message sizes, cuda_dth have the highest bandwidths while cuda_htd is even lower than the point-to-point interconnect bandwidth (NP_ugni). Hence, overall transfer bandwidths, for a highly tuned and optimized application cannot exceed (cuda_htd). The measure bandwidths with different pipeline chunk sizes are shown as NP_ugni_cuda_C1m (1 MB) and NP_ugni_cuda_C2m (2 MB).

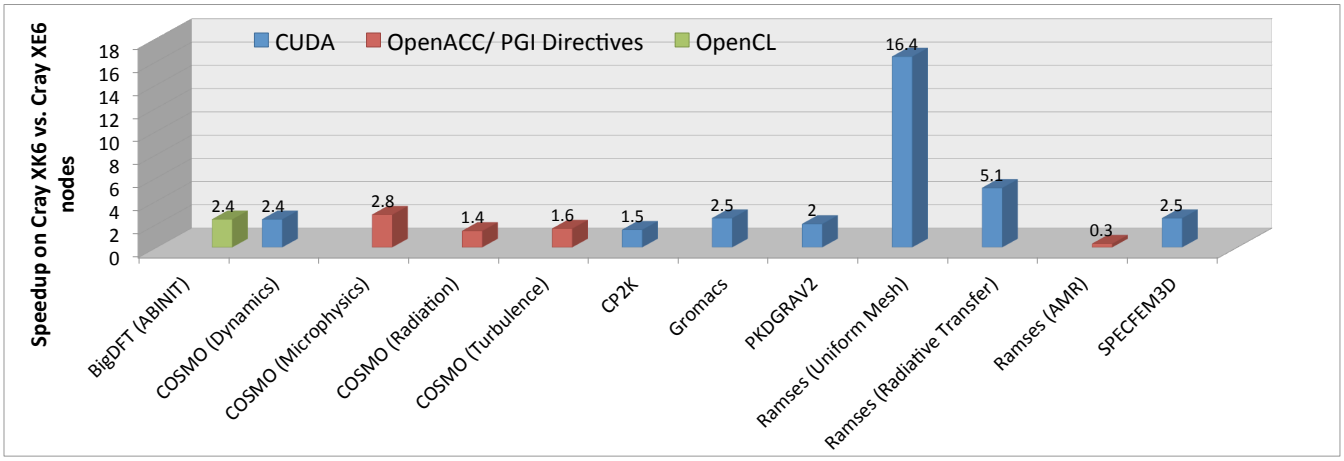

Figure 9. Summary of Cray XE6 and Cray XK6 node performance results. Note that a single Cray XE6 node has 2 AMD 6272 sockets while a Cray XK6 node has one AMD socket and an NVIDIA X2090 GPU device. These are preliminary results illustrating work in progress. For instance, speedup for applications accelerated with CUDA and OpenCL demonstrate higher speedup. We intend on reporting progress as the devices, software stack and algorithms improve for the Cray XK7 system.

In order to identify whether the pipeline sizes have a significant impact on the bandwidth between GPU to remote GPU transfers, additional tests were performed. Results are shown in figure 8 for pipeline sizes of 256 Kbytes to up to 4 Mbytes. The results demonstrate that the these transfer rates can be an important tuning parameter as the effective bandwidth could be improved by as much as 40% for transferring same message sizes between remote GPUs.

This evidence suggests that Cray could provide an integrated MPI library that is tuned and optimized for different message sizes for GPU to GPU transfers over the MPI interconnect as it provides for the non-accelerator, multi-core platforms such as Cray XE6, where the latencies are sensitive to NUMA regions. Note that there are 4 NUMA regions on a Cray XE6 dual-socket node vs. 2 on a Cray XK6 single-socket node. A similar solution has been proposed by the MVAPICH library developers and this could be implemented in the Cray MPI [7].

Since early 2010, CSCS has been engaging application development teams in preparation for the next-generation multi-petascale platforms as part of a project called High Performance High Productivity Computing (HP2C). Details of the project are available at: www.hp2c.ch. A number of these application groups have successfully been exploiting the Cray XK6 platform as early science users. These applications are listed in Table V. Most of these applications have been developed using CUDA and OpenCL on existing cluster platforms with accelerators, and the migration process to the Cray XK6 platform was straightforward.

Two applications, COSMO and RAMSES, are beginning to exploit the OpenACC incremental programming approach and these groups are making steady progress on the Cray XK6 platform using the Cray compiler environment. Comparison of Cray XK6 and Cray XE6 platform for different variants of applications that are listed in Table IV are shown in figure 9.

TABLE V. CACHE AND MEMORY HIERARCHY OF THE NVIDIA FERMI X2090 AND OPTERON BULLDOZER 6272 PROCESSORS

Application Names	Domains	Project & Development Teams Details
BigDFT	Materials science / Nanoscience	http://inac.cea.fr/L_Sim/BigDFT
COSMO	Regional climate / meteorology	http://www.clm-community.eu
CP2K	Chemical science / Nanoscience	http://www.cp2k.org
Gromacs	Life science	http://www.gromacs.org
PKDGRAV2	Cosmology	https://hpcforge.org/projects/pkdgrav2
RAMSES	Cosmology	http://irfu.cea.fr/Projets/COAST/software.htm
SPECFEM3D	Seismology	http://www.geodynamics.org/cig/software/specfem3d

Applications presented in figure 9 represent production-level simulations that have been awarded computation time on CSCS flagship Cray XE6 system. However, these are at different levels of maturity in terms of GPU accelerated components. For example, SPECFEM3D production level simulations are being performed using the GPU accelerated while in COSMO benefits of GPU acceleration has been explored using different strategies (CUDA and directives). Since the Cray XK6 is considered as a prototype platform for a multi-Petaflops Cray XK7 system that is expected to be installed later this year at the Oak Ridge National Laboratory (ORNL), some issues may only be resolved for a new hardware and programming environment [2]. We intend to collaborate with ORNL and Cray as we transition to the next-generation integrated accelerator based MPP platform.

V. SUMMARY AND FUTURE PLANS

Cray XK6 system is a tightly integrated MPP platform, which offers similar programming and execution environments as multi-core based Cray XT and XE series systems. However, due to a fundamental difference in the node design where a GPU device is available for code acceleration, the system has been equipped with some

unique programming, execution and operational design elements. As this report presents, a number of these unique features are available and functional within the integrated hardware and software platform. At the same time however, since installing the system in 2011, CSCS together with Cray have resolved a number of issues that emerged during the operational phase. As we have identified in the paper, some issues remain to be solved, for example, a parallel debugger for OpenCL code development, as well as high performance MPI for GPU programming.

ACKNOWLEDGEMENTS

We would like to acknowledge contributions of the following individuals: Kevin Peterson, Alistair Hart, Heidi Poxon, Luiz DeRose and Adrian Tate (Cray Inc), Tim Robinson and Gilles Fourestey (CSCS). Peter Messmer (NVIDIA).

REFENRENCES

[1] J. Beyer, E. Stotzer, A. Hart & B. De Supinski, "OpenMP for Accelerators," Proc. of the 7th International Workshop on OpenMP, 2011.

[2] A. S. Bland, R. L. Graham, O. E. Messer, II and J. C. Well, "Titan: Early experience with the Cray XK6 at Oak Ridge National Laboratory," Proceedings of the Cray User Group meeting, 2012.

[3] J. Nickolls, I. Buck, M. Garland, and K. Skadron. "Scalable Parallel Programming with CUDA." Queue 6, 2 (March 2008).

[4] H. Pritchard, I. Gorodetsky, and D. Buntinas. "A uGNI-based MPICH2 nemesis network module for the Cray XE," Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface (EuroMPI'11).

[5] G. Renker, N. Stringfellow, K. Howard, S. Alam and S. Trofinoff, "Employing SLURM on XT, XE, and Future Cray Systems," Proceedings of the Cray User Group meeting, 2011.

[6] J. Sollom, "Overview of Node Health Checker," Proceedings of the Cray User Group meeting, 2011.

[7] H. Wang, S. Potluri, M. Luo, A. K. Singh, S. Sur, and D. K. Panda. "MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters," Comput. Sci. 26, 3-4 (June 2011).

[8] Allinea parallel debugging tool (DDT): http://www.allinea.com/products/ddt/

[9] AMD 6200 series processors: http://www.amd.com/us/products/server/processors/6000-series-platform/6200/Pages/6200-series-processors.aspxCray XK6 platform: www.cray.com/Products/XK6/XK6.aspx

[10] BLAS (Basic Linear Algebra Subprograms): http://www.netlib.org/blas/

[11] Cray XE6 platform: www.cray.com/Products/XE/CrayXE6System.aspx

[12] Cray XK6 platform: www.cray.com/Products/XK6/XK6.aspx

[13] CUDA programming resources: http://developer.nvidia.com/category/zone/cuda-zone

[14] Lapack (Linear Algenra Package): http://www.netlib.org/lapack/

[15] NetPIPE (a network protocol independent performance evaluator): http://bitspjoule.org/netpipe/

[16] PGI Accelerator: http://www.pgroup.com/resources/accel.htm

[17] OpenACC directives for accelerators: www.openacc-standard.org

[18] OpenCL standard: www.khronos.org/opencl

[19] Stream memory bandwidth benchmark: http://www.cs.virginia.edu/stream/

[20] Totalview debugger: http://www.roguewave.com/products/totalview.aspx