# Chapter 7

## Keeneland: Computational Science Using Heterogeneous GPU Computing

**Jeffrey S. Vetter**

*Oak Ridge National Laboratory and Georgia Institute of Technology*

**Richard Glassbrook, Karsten Schwan, Sudha Yalamanchili, Mitch Horton, Ada Gavrilovska, and Magda Slawinska**

*Georgia Institute of Technology*

**Jack Dongarra**

*University of Tennessee and Oak Ridge National Laboratory*

**Jeremy Meredith, Philip C. Roth, and Kyle Spafford**

*Oak Ridge National Laboratory*

**Stanimire Tomov**

*University of Tennessee*

**John Wynkoop**

*National Institute for Computational Sciences*

## 7.1 Overview

The Keeneland Project[VGD+11] is a five-year Track 2D grant awarded by the National Science Foundation (NSF) under solicitation NSF 08-573 in August 2009 for the development and deployment of an innovative high performance computing system. The Keeneland project is led by the Georgia Institute of Technology (Georgia Tech) in collaboration with the University of Tennessee at Knoxville, National Institute of Computational Sciences, and Oak Ridge National Laboratory.

**NSF 08-573: High Performance Computing System Acquisition - Towards a Petascale Computing Environment for Science and Engineering**

*An experimental high-performance computing system of innovative design.* Proposals are sought for the development and deployment of a system with an architectural design that is outside the mainstream of what is routinely available from computer vendors. Such a project may be for a duration of up to five years and for a total award size of up to $12,000,000. It is not necessary that the system be deployed early in the project; for example, a lengthy development phase might be included. Proposals should explain why such a resource will expand the range of research projects that scientists and engineers can tackle and include some examples of science and engineering questions to which the system will be applied. It is not necessary that the design of the proposed system be useful for all classes of computational science and engineering problems. When finally deployed, the system should be integrated into the TeraGrid. It is anticipated that the system, once deployed, will be an experimental TeraGrid resource, used by a smaller number of researchers than is typical for a large TeraGrid resource. (Up to 5 years' duration. Up to $12,000,000 in total budget to include development and/or acquisition, operations and maintenance, including user support. First-year budget not to exceed $4,000,000.)

### 7.1.1 The Case for Graphics Processors (GPUs)

The Keeneland team originally assessed numerous technologies to propose in 2008. The team's conclusion was that heterogeneous architectures using GPUs provided the right balance of performance on scientific applications, productivity, energy-efficiency, and overall system cost.

Recently, heterogeneous architectures have emerged as a response to the limits of performance on traditional commodity processor cores, due to power and thermal constraints. Currently, most commodity multi-core architectures use a small number of replicated, general purpose cores that use aggressive techniques to minimize single thread performance using techniques like out-of-order instruction execution, caching, and a variety of speculative execution techniques. While these approaches continue to sustain performance, they can also carry high costs in terms of energy efficiency.

Simultaneously, other approaches, like graphics processors, have explored design strategies that constitute different design points: large numbers of simple cores, latency hiding by switching among a large number of threads quickly in hardware, staging data in very low latency cache or scratchpad memory, and, wider vector (or SIMD) units. For GPUs, in particular, these techniques were originally intended to support a fixed pipeline of graphics operations (e.g., rasterization).

A number of early adopters recognized that these design points offered benefits to their scientific applications, and they began using GPUs for general purpose computation [AAD+, OLG+05a] (so called 'GPGPU'). Shortly thereafter, the GPU ecosystem started to include programming systems, such as Cg [MGAK03], CUDA [NB07], and OpenCL [SGS10], to make GPUs available to an even wider non-graphics audience.

Eventually, GPUs have added critical features that have made them much more applicable to a wider array of scientific applications and large-scale HPC systems. For example, NVIDIA's Fermi [NVI09a], was the first GPU to add much improved performance on IEEE double precision arithmetic (only 2 times slower than single precision), and error correction and detection, which makes these devices more reliable in a large-scale system. These new capabilities, when combined with the original niche of GPUs, provide a competitive platform for numerous types of computing, such as media processing, gaming, and scientific computing, in terms of raw performance (665 GF/s per Fermi), cost, and energy efficiency.

Accordingly, these trends have garnered the attention of HPC researchers, vendors, and agencies. Beyond the Keeneland project, a significant number of large GPU-based systems have already been deployed. Examples include China's Tianhe-1A (cf. §19.1), Nebulae at the National Supercomputing Centre in Shenzhen (NSCS), Tokyo Tech's TSUBAME2.0 (cf. §20.1), Lawrence Berkeley National Laboratory's Dirac cluster, FORGE at the National Center for Supercomputing Applications, and EDGE at Lawrence Livermore National Laboratory. Notably, the Chinese Tianhe-1A system at the National Supercomputer Center in Tianjin achieved a performance of 2.57 pf/s on the TOP500 LINPACK benchmark (http://www.top500.org), which placed it at #1 on the list in November 2010. All of these examples are scalable heterogeneous architectures that leverage mostly commodity components: scalable node architectures with a high performance interconnection network, where each node contains memory, network ports, and multiple types of (heterogeneous) processing devices. Most experts expect this trend for heterogeneity to continue into the foreseeable future, given current technology projections and constraints.

### 7.1.2 Timeline

The Keeneland project is a five-year effort, and it is organized into two primary deployment phases. The initial delivery system was deployed in the fall of 2010 and the full-scale system was deployed in the summer of 2012. The first phase provides a moderately-sized, initial delivery system for the development of software for GPU computing, and for preparing and optimizing applications to exploit GPUs. The Keeneland Initial Delivery (KID) system was not available as an official production resource to NSF users, but it was made available to over 200 users across 90 projects.

The second phase of Keeneland was deployed during the summer and fall of 2012 and it provides a full-scale system for production use by computational scientists as allocated by the National Science Foundation. The Keeneland Full Scale (KFS) system is similar to the KID system in terms of hardware and software (see Table 7.1). The KFS system is an XSEDE resource available to a broad set of users. Although there now appears to be a large migration of the HPC community to these heterogeneous GPU architectures, a critical component of the Keeneland Project is the development of software to allow users to exploit these architectures, and to reach out to applications teams that have applications that may map well to this architecture, in order to encourage them to port their applications to architectures like Keeneland.

## 7.2 Keeneland Systems

### 7.2.1 Keeneland Initial Delivery System

The KID system has been installed, and operating since November 2010; it is primarily used for the development of software tools and preparation of applications to use this innovative architecture. In addition, KIDS served the scientific community with over 100 projects and 200 users through discretionary accounts in order to allow scientists to evaluate, port, and run on a scalable GPU system.

As of June 2012, the KID system configuration (cf. Table 7.1) is rooted in the scalable node architecture of the HP Proliant SL-390G7, shown in Figure 7.1. Each node has two Intel Westmere host CPUs, three NVIDIA M2090 6GB Fermi GPUs, 24GB of main memory, and a Mellanox Quad Data Rate (QDR) InfiniBand Host Channel Adapter (HCA). Overall, the system has 120 nodes with 240 CPUs and 360 GPUs; the installed system has a peak performance of 255 TFLOPS in 7 racks (or 90 sq ft including the service area).

More specifically, in the SL390, memory is directly attached to the CPU sockets, which are connected to each other and the Tylersburg I/O hubs via Intel's Quick Path Interconnect (QPI). GPUs are attached to the node's two I/O hubs using PCI Express (PCIe). The theoretical peak for unidirectional bandwidth of QPI is 12.8 GB/s and for PCIe x16 is 8.0 GB/s. In particular, with these two I/O hubs, each node can supply a full x16 PCIe link bandwidth to three GPUs, and x8 PCIe link bandwidth to the integrated InfiniBand QDR HCA. This design avoids contention and offers advantages in aggregate node bandwidth when the three GPUs and HCA are used concurrently, as they are in a scalable system. In contrast, previous architectures used a PCIe-switch-based approach, and the switch quickly became a performance bottleneck. Using this PCIe-switch-based approach, vendors are currently offering systems with up to 8 GPUs per node.

The node architecture exemplifies the architectural trends described earlier, and has one of the highest number of GPUs counts per node in the Top500 list. The SL390 design has significant benefits over the previous generation architecture, but also exhibits multiple levels of non-uniformity [MRSV11]. In addition to traditional NUMA effects across the two Westmere's integrated memory controllers, the dual I/O hub design introduces non-uniform

**TABLE 7.1**: Keeneland hardware configurations.

| **Feature** | **KIDS (July 2012)** | **KFS System** |
|---|---|---|
| Node Architecture | HP Proliant SL390 G7 | HP Proliant SL250s G8 |
| CPU | Intel Xeon X5660 | Intel Xeon E5-2670 |
| CPU Microarchitecture | Westmere | Sandy Bridge |
| CPU Frequency (GHz) | 2.8 | 2.6 |
| CPU Count per Node | 2 | 2 |
| Node Memory Capacity (GB) | 24 | 32 |
| Node PCIe | Gen 2 | Gen 3 |
| GPU | NVIDIA Tesla M2090 | NVIDIA Tesla M2090 |
| GPU Count per Node | 3 | 3 |
| GPU Memory Capacity (GB) | 6 | 6 |
| Interconnection Network | InfiniBand QDR | InfiniBand FDR |
| Network Ports per Node | 1 IB QDR HCA | 1 IB FDR HCA |
| Compute Racks | 5 | 11 |
| Total Number of Nodes | 120 | 264 |
| Peak FLOP Rate (TF) | 201 | 615 |

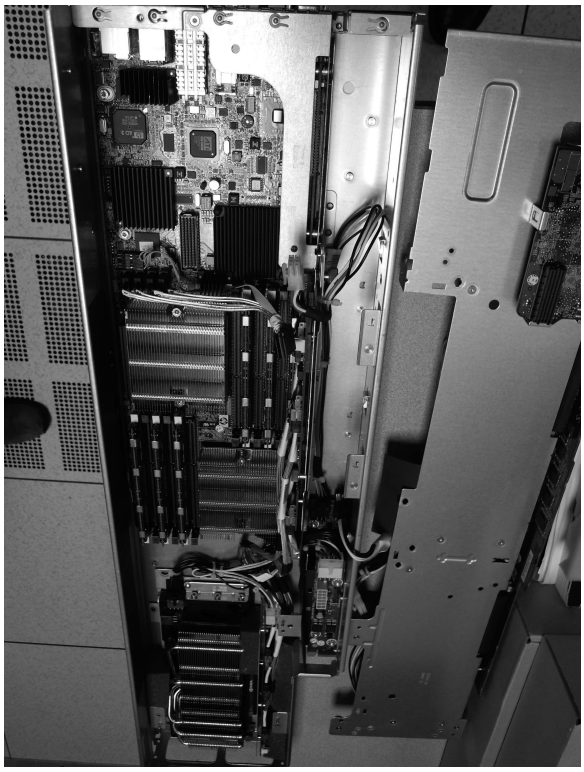

**FIGURE 7.1**: HP Proliant SL390 node.

characteristics for data transfers between host memory and GPU memory. These transfers will perform better if the data only traverses one QPI link (such as a transfer between data in the memory attached to CPU socket 0 and GPU 0) than if it traverses two QPI links (such as a transfer between data in the memory attached to CPU socket 0 and GPU 1 or GPU 2).

In addition, KIDS's GPUs include other features that can greatly affect performance and contribute to non-uniformity. For instance, each GPU contains Error Correcting Code (ECC) memory. ECC memory is desirable in a system designed for scalable scientific computing. Enabling ECC gives some assurance against these transient errors, but results in a performance penalty and adds yet another complexity to the GPU memory hierarchy.

### 7.2.2 Keeneland Full Scale System

The KFS system has been installed and operating since October 2012; it is an XSEDE production resource, which is allocated quarterly by XSEDE's XRAC allocations committee.

As of July 2012, the KFS system configuration (cf. Table 7.1) is very similar to the KID system architecture, but with upgraded components in most dimensions. In particular, each node is a HP Proliant SL250G8 server with two Intel Sandy Bridge host CPUs, three NVIDIA M2090 6GB Fermi GPUs, 32GB of DDR3 main memory, and a Mellanox Fourteen Data Rate (FDR) InfiniBand HCA. Overall, the system has 264 nodes with 528 CPUs and 792 GPUs; the installed system has a peak performance of 615 TFLOPS in 11 compute racks.

A major difference between the KID and KFS systems is the host processor and node configuration. First, the KFS system uses Intel's new Sandy Bridge architecture. This change

has several benefits including an increase from six to eight cores per socket, 40 lanes of integrated PCIe Gen3, and new AVX instructions. This integrated PCIe Gen3 eliminates the need for a separate Tylersburg I/O hub, as was the case for KIDS's Westmere architecture. Second, the Proliant SL250G8 node adapts to the new Sandy Bridge architecture by expanding memory, eliminating I/O chips, while retaining the capacity for 2 CPUs, 3 GPUs, and an FDR IB port, all at full PCIe Gen3 bandwidth.

## 7.3 Keeneland Software

The system software used on the Keeneland systems reflects the perspective that the systems are Linux x86 clusters with GPUs as compute accelerators. Both the KID and KFS systems use CentOS, a clone of Red Hat Enterprise Linux, as the fundamental Linux distribution. This distribution provides the Linux kernel, plus a large collection of user-level programs, libraries, and tools. The KFS system was deployed with CentOS 6.2. KIDS was deployed using CentOS version 5.5, but was upgraded to version 6.2 when the KFS system was deployed. In addition to the stock CentOS distribution, the NVIDIA GPU driver and the NVIDIA Compute Unified Device Architecture (CUDA) Software Development Kit are installed on each compute node to allow programs to use the system's GPUs. Likewise, Mellanox's variant of the Open Fabric Enterprise Distribution (OFED) is installed on each system node to support the use of the InfiniBand interconnection networks. The InfiniBand network is routed using the Open Subnet Manager (OpenSM) on KIDS and Mellanox's Unified Fabric Manager (UFM) on KFS.

**TABLE 7.2**: Keeneland software configurations.

| **Feature** | **KIDS** | **KFS System** |
|---|---|---|
| Login Node OS | CentOS 5.5 | CentOS 6.2 |
| Compute Node OS | CentOS 5.5 | CentOS 6.2 |
| Parallel Filesystem | Lustre 1.8 | |
| Compilers | Intel 12 | |
| | PGI 12 with support for compiler directives | |
| | CAPS HMPP 2.4.4 | |
| | GNU 4.1 | GNU 4.4 |
| | NVIDIA Toolkit 4.1 | NVIDIA Toolkit 4.2 |
| MPI | OpenMPI (default) | |
| | MVAPICH | |
| Notable Libraries | HDF5 | |
| | netcdf/pNetCDF | |
| | Intel Math Kernel Library | |
| | Thrust | |
| | Boost | |
| | FFTW | |
| Job Scheduler | Moab | |
| Resource Manager | Torque | |
| Debugging Tools | Allinea DDT | |
| | NVIDIA cuda-gdb | |
| Performance Tools | TAU | |
| | HPCToolkit | |
| | NVIDIA Visual Profiler | |

### 7.3.1 Filesystems

Two primary filesystems are used on the Keeneland systems. The Network File System (NFS) version 3 is used for filesystems that are not expected to need the high I/O performance provided by a parallel filesystem, such as home directories and installations of shared software (i.e., commonly-used numerical libraries and tools). For programs that require high I/O performance, such as parallel climate simulations that write large checkpoint files, a parallel file system is available. These file systems are part of the NICS center-wide storage systems that serve not only the Keeneland systems but also the other systems operated by NICS personnel. Initially, the NICS parallel filesystem accessible to KIDS users was IBM's General Parallel File System (GPFS), but that filesystem was replaced in 2011 by a Lustre parallel file system. In 2012, NICS' center-wide Lustre filesystem was further expanded. This expanded filesystem gives the Keeneland systems access to over 4 Petabytes of scratch space with a maximum throughput of over 70 Gigabytes per second. The shared Lustre filesystem is accessed using a center-wide QDR InfiniBand network. Unlike some high performance computing systems, both the home directories and the parallel file system are accessible to programs running on Keeneland system compute nodes.

### 7.3.2 System Management

The KID system was initially deployed using a combination of Preboot Execution Environment (PXE) to boot the nodes via the network and Kixstart, a scripted installer that is included with CentOS, to perform the OS installation. The open source configuration management tool puppet was used to maintain the configuration of the KID system. The KID system was modified to use a shared NFS root model in fall 2012 to better mirror the configuration of KFS. The system uses the nfsroot package developed by Lawrence Livermore National Laboratory to provide the shared root implementation. The KFS system is managed using HP's Cluster Management Utility (CMU). KFS also utilizes a shared root filesystem. However, the management is done via the CMU tool. Job scheduling on both systems is provided via the Moab batch environment with Torque deployed as the resource manager. To facilitate remote power control and remote console access the HP Integrated Lights Out (iLO) controllers and HP Advanced Power Manager (APM) management systems are deployed in both systems. KIDS uses HP's iLO 3, while the KFS system uses the iLO 4.

---

## 7.4 Programming Environment

The Keeneland programming environment is a blend of standard HPC software, such as MPI, augmented by GPU-enabled programming environments. Keeneland offers a variety of approaches for exploiting GPUs including GPU-enabled libraries, writing CUDA or OpenCL directly, or using directive-based compilation. Keeneland also offers a set of development tools for correctness and performance investigations.

In addition to commercial and research tools available from the community, the Keeneland project team is developing several tools for GPU-enabled systems. These tools include GPU-enabled scientific libraries (MAGMA), productivity tools (Ocelot), and virtualization support.

### 7.4.1 Programming Models

Developing software for the KID and KFS systems involves a process similar to that used when developing for a traditional Linux cluster. However, to make use of the systems' GPUs, the traditional process must be augmented with development tools that can produce and debug code that runs on the GPUs.

**MPI for Scalable Distributed Memory Programming.** As described in Section 7.2, both KID and KFS are distributed memory systems, and a message passing programming model is the primary model programs used for communication and synchronization between processes running on different compute nodes. Several implementations of the Message Passing Interface (MPI) are available on the systems; OpenMPI is the default. These implementatons are built to take advantage of the systems' high performance InfiniBand interconnection networks. Although the KID system is capable of using the NVIDIA GPUDirect inter-node communication optimization across Mellanox InfiniBand networks, it has not yet been enabled due to challenges in deploying it with the CentOS 5.5 kernel used on that system. KFS, which uses a newer CentOS kernel, uses the GPUDirect inter-node optimization.

**CUDA and OpenCL.** In contrast to the near-ubiquity of MPI for inter-node communication and synchronization, we observe much more variety in the approaches used to make use of the systems' GPUs and multi-core processors. With respect to using the GPUs, NVIDIA's Compute Unified Device Architecture (CUDA) is currently the most common approach, but some programs running on the Keeneland systems use OpenCL. On the Keeneland systems, support for developing CUDA and OpenCL programs is provided by development software freely available from NVIDIA. The NVIDIA CUDA compiler, `nvcc`, is part of the CUDA Toolkit, and the NVIDIA GPU Computing Software Development Kit (SDK) is available for developers that use the utility software from that SDK. The CUDA Toolkit also provides libraries needed to develop OpenCL programs that use NVIDIA GPUs.

**Directive-Based Compilation.** Using CUDA or OpenCL can provide excellent performance on systems like KID and KFS, but some developers feel that these approaches require programming at a level of abstraction that is too low. Developers seeking not only high performance but also high productivity are often drawn to the idea of using *compiler directives*. Such directives are pragmas (in C or C++ programs) or comments (in Fortran programs) embedded in the program's source code that indicate to the compiler the parts of the code that should be executed on a GPU. In most cases, if a program containing compiler directives is processed by a compiler without support for the directives or such support is disabled, the compiler still produces a valid, single-threaded program executable that executes only on the system's CPU.

Several compilers supporting compiler directives are provided on the Keeneland systems. For programs using GPUs, the PGI Fortran, C, and C++ compilers are available with support for both OpenACC and PGI Accelerate directives in the Fortran and C compilers. We also make available the CAPS HMPP compiler supporting the OpenHMPP compiler directives. The GNU and Intel compilers are also available, and although they do not provide any particular support for developing programs that use GPUs, they do support OpenMP compiler directives for producing multi-threaded programs for the CPUs in Keeneland system compute nodes.

**GPU-Enabled Libraries.** Multi-threaded and GPU-enabled libraries provide another high-productivity approach for developers targeting systems like KID and KFS. For instance, NVIDIA's CUDA Toolkit provides several libraries containing GPU-accelerated implementations of common operations such as the Basic Linear Algebra Subroutines (BLAS) and Fast Fourier Transform (FFT). The Intel compilers include the Intel Math Kernel

Library that provides OpenMP-enabled BLAS and FFT implementations for targeting the Keeneland systems' multi-core CPUs.

To make best use of the computational hardware available in KID and KFS compute nodes, a multi-level hybrid programming model is possible that combines MPI tasks, OpenMP, and one of the previously mentioned GPU programming models. In such a model, a program places a small number of MPI tasks on each compute node, each of which uses CUDA function calls to make use of one of the node's GPUs, and is multi-threaded using OpenMP to make use of the node's CPU cores. An interesting question for the application user is whether to use two MPI tasks to match the number of processors in each node, or three MPI tasks to match the number of GPUs.

**Development Tools for Performance and Correctness.** Converting source code into executable code is only part of the software development task. Finding and fixing functional and performance problems in programs are also important software development tasks. For functional problems, the Allinea Distributed Debugging Tool (DDT) is available on the Keeneland systems. DDT is a debugger that implements the traditional breakpoint and single-step debugging models for parallel programs. It has support for programs that use MPI and OpenMP. Most importantly for a scalable heterogeneous computing system like the Keeneland systems, DDT supports setting breakpoints and single-stepping through CUDA kernel code, and observing data held in GPU memory. In addition to DDT, we also provide the NVIDIA debugger cuda-gdb, though this debugger is more suitable for use with single-process programs than the multi-node parallel programs that are the target workload for the Keeneland systems.

With respect to tools for identifying the source of performance problems, we rely on both NVIDIA and third-party tools. As part of the CUDA Toolkit, we make available NVIDIA's Compute Profiler that collects performance data about a program's GPU use, analyzes that data, and then makes recommendations about how to improve the program's use of the system's GPUs. As with cuda-gdb, this tool is targeted mainly at single-process programs. For programs that use processes running on multiple compute nodes, third-party tools are available such as the Tuning and Analysis Utilities [SM06a] (TAU) from the University of Oregon and HPCToolkit [ABF+10] from Rice University. In addition to support for collecting performance data regarding a program's MPI and OpenMP behavior, TAU now supports the collection of performance data about CUDA and OpenCL kernels. HPCToolkit has been highly useful in collecting performance profiles for full applications at scale, though the versions we have used do not have support for collecting profiles of code running on the system's GPUs.

### 7.4.2 Keeneland Developed Productivity Software

**Libraries and Frameworks.** An integral part of the Keeneland project is the development of fundamental linear algebra algorithms and numerical libraries for hybrid GPU-CPU architectures. The goal is to enable the efficient use of the KID and KFS systems, as well as to ease the porting of key applications to them. Existing GPU-only libraries, that implement the most basic algorithms, capturing main patterns of computation and communication, are available on the Keeneland systems. In particular, for dense linear algebra (DLA) this is the NVIDIA CUBLAS library [NVI12a], for sparse linear algebra the NVIDIA CUSPARSE [NVI12c], for spectral methods the NVIDIA CUFFT [NVI12b], etc. The NVIDIA Thrust library [HB10], providing a C++ template GPU functionality with an interface similar to the C++ Standard Template Library (STL), is also available. The CPU equivalents provided from libraries such as MKL from Intel, ACML from AMD, GotoBLAS, and Atlas, are installed as well.

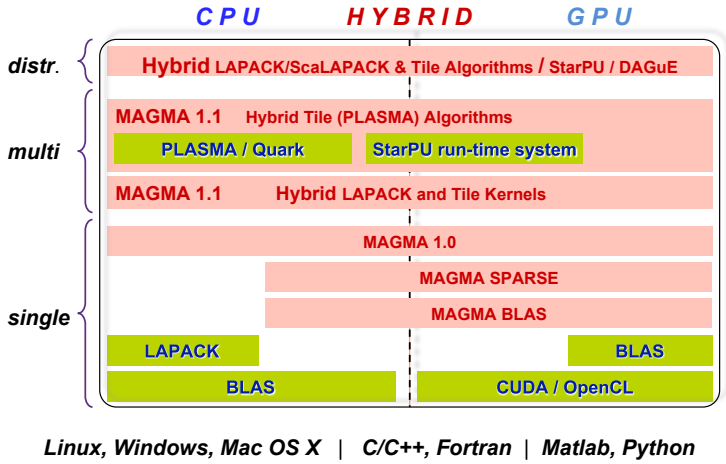


**FIGURE 7.2**: MAGMA software stack.

Developing higher level algorithms, as well as applications in general, for hybrid GPU-CPU systems by simply replacing computational routines with their equivalents from the GPU- or CPU-centric libraries is possible, but will lead to inefficient use of the hardware due for example to all synchronizations of the fork-join parallelism at every parallel routine call, and GPU-CPU communications in preparing the data. The goal of the Keeneland project, as related to libraries, has been to overcome challenges like these and to develop numerical libraries for hybrid GPU-CPU architectures.

To this end, in what follows, we outline the main development challenges to numerical libraries for hybrid GPU-CPU systems and how to overcome them. Illustrations are given using the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library [TDB10] that we develop. Its newest release, MAGMA 1.2 [mag12], targeting DLA algorithms is installed on Keeneland. The Trilinos [HBH+05] and PETSc [MSK12] libraries, targeting specifically sparse linear algebra computations, are also available on the Keeneland systems.

The MAGMA project aims to develop the next generation of LAPACK and ScaLAPACK-compliant linear algebra libraries for hybrid GPU-CPU architectures. MAGMA is built on the GPU- or CPU-centric libraries mentioned above, which is also illustrated in Figure 7.2, giving the software stack for MAGMA. The currently released software is for shared memory multicore CPUs with single GPU or multiple GPUs (see Figure 7.3 for more details). Software for distributed memory systems has also been developed [STD12] and will be added to subsequent releases after further development.

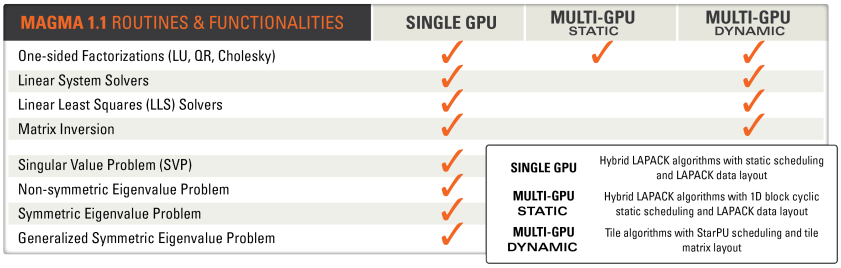


**FIGURE 7.3**: MAGMA 1.1 supported functionality.

There are a number of challenges in the development of libraries for hybrid GPU-CPU systems, and numerical libraries in general. Most notably, just to mention a few, these are:

- **Synchronization**, as related to parallelism and how to break the fork-join parallel model
- **Communication**, and in particular, the design of algorithms that minimize data transfers to increase the computational intensity of the algorithms
- **Mixed precision methods**, exploiting faster lower precision arithmetic to accelerate higher precision algorithms without loss of accuracy
- **Autotuning**, as related to building "smarts" into software to automatically adapt to the hardware

Synchronization in highly parallel systems is a major bottleneck for performance. Figure 7.4 quantifies this statement for the case of the LU factorization using the fork-join LAPACK implementation (with parallel high-performance BLAS from the MKL library) *vs.* MAGMA 1.1. Note that the systems compared have the same theoretical peaks and the expectation is that the performances will be comparable. Because of the fork-join synchronizations, LAPACK on this 48 core system is about $4X$ slower than MAGMA on a system using a single Fermi GPU (and a four core CPU host).

To overcome this bottleneck MAGMA employs a Directed Acyclic Graph (DAG) approach. The DAG approach is to represent algorithms as DAGs in which nodes represent subtasks and edges represent the dependencies among them, and subsequently schedule the execution on the available hardware components. Whatever the execution order of the subtasks, the result will be correct as long as these dependencies are not violated. Figure 7.5 illustrates a schematic DAG representation for algorithms for multicore on the left and for hybrid systems on the right [ADD+09]. The difference with hybrid systems is that the GPU tasks must be suitable and large enough for efficient data-parallel execution on the GPU.

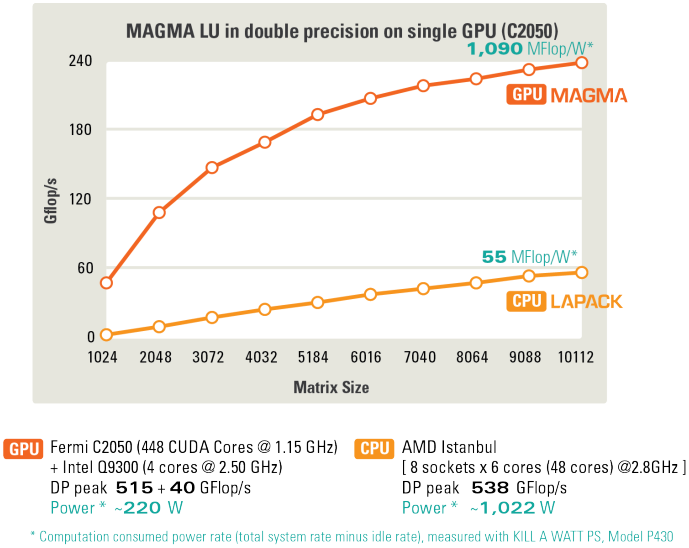


**FIGURE 7.4**: Performance of LU – LAPACK on multicore vs MAGMA on hybrid GPU-CPU system.

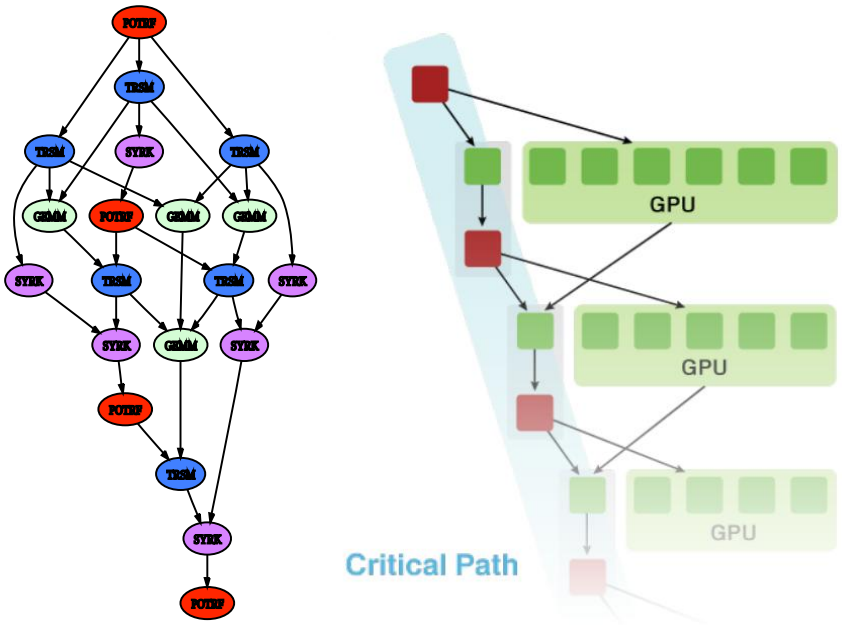


**FIGURE 7.5**: Algorithms as DAGs for multicore (left) and hybrid systems (right).

To account for this we have extended the DAG approach to a hybridization methodology where the critical path of the algorithm is determined and in general scheduled for execution on the multicore CPUs. This approach has been applied to the main DLA algorithms for solving linear systems of equations and eigenvalue problems [TNLD10, TND10], and extended to multiple GPUs in MAGMA 1.1 [YTD12]. Figure 7.6 illustrates the performance obtained for the LU factorizations using all GPUs on a Keeneland node. Performance scales

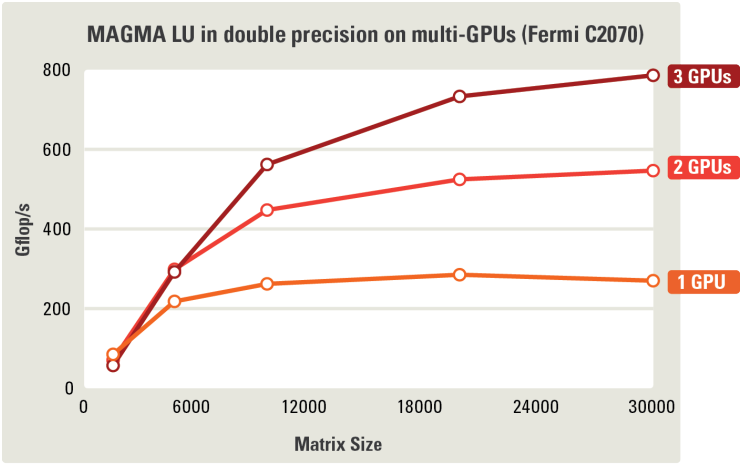


**FIGURE 7.6**: LU factorization on a multiGPU–multicore Keeneland system node.

as we increase the number of GPUs. The algorithms involve static 1-D block cyclic data distribution with look-ahead, where the panels are factored on the CPU [YTD12]. Similarly to the case of one GPU, the algorithm reduces GPU-CPU communications by placing the entire matrix on the GPUs, and communicating only the panels. The cases where the matrix does not fit on the GPU memories is handled as well by implementing left-looking versions that are known to reduce communications [YTD12].

If the execution is statically scheduled, look-ahead techniques have been successfully applied to matrix factorizations [TNLD10, YTD12] to remedy the problem of barrier synchronizations introduced by the non-parallelizable tasks in a fork-join parallelization. Current efforts are concentrated on **dynamic scheduling** where an out-of-order execution order is determined at run-time in a fully dynamic fashion. For further information and examples on hybrid scheduling (using the QUARK scheduler on the multicore host and static on the GPU) see [HTD11], and for fully dynamic scheduling using StarPU see [AAD$^+$].

Finally, to build the DAG approach into efficient frameworks, the scheduling should allow building applications by combining components – available in libraries as DAGs – without synchronization between them. We call this feature **DAG composition**, illustrated in Figure 7.7 with the execution traces on a 48 core system with and without DAG composition. This feature is becoming available in the DAG schedulers that we use such as QUARK [YKD11], StarPU [ATNW10], and DAGuE [BBD$^+$12].

**Productivity Tools: Correctness and Performance Debugging.** The Keeneland system is host to a set of productivity tools developed for CUDA applications and distributed as part of the Ocelot dynamic execution infrastructure [DKYC10]. Ocelot was originally conceived to facilitate and accelerate research in GPGPU computing using CUDA, and has evolved into an infrastructure to support research endeavors across a broad spectrum of hardware and software challenges for GPGPU computing. One of the main challenges has been software productivity faced by the designers of GPGPU architectures and systems with integrated GPGPUs. Major challenges to software productivity are seen to be i) execution portability, ii) performance portability, and iii) introspection, e.g., performance tuning and debugging tools.

At its core, Ocelot is a dynamic compiler that translates compute kernels for execution on NVIDIA GPUs. Ocelot's internal representation is based on NVIDIA's parallel thread

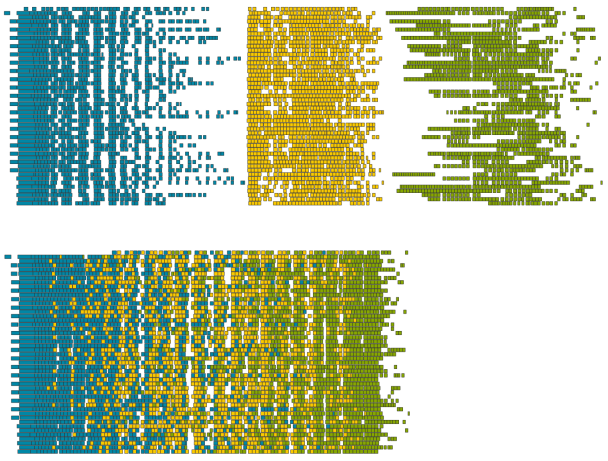

**FIGURE 7.7**: Composition of DAGs – execution traces on a 48 core system with synchronization (top) *vs.* with DAG composition (bottom).

execution (PTX) low level virtual instruction set architecture (ISA). Ocelot implements a just-in-time (JIT) compiler by translating kernel PTX to the intermediate representation (IR) of the LLVM compiler infrastructure and using LLVM's back-end code generators [LA04]. Back-ends have been built and tested for i) multicore x86, ii) Intel SSE, iii) NVIDA GPUs, iv) AMD GPUs [DSK11], and v) execution on GPUs attached to remote nodes. Ocelot includes a re-implementation of the CUDA runtime to support these back-end devices, and existing CUDA applications can be executed by simply linking with the Ocelot runtime.

Within this infrastructure we have two main additions that support software productivity tools. The first is an additional back-end device that is a functionally accurate emulator for the PTX ISA [AAD+]. The emulator is instrumented for trace generation. Event trace analyzers coupled with the emulator can be used for correctness checks, workload characterization, and performance debugging. The second addition is an interface for the specification and dynamic instrumentation of PTX kernels. This latter capability does not require any modification to the CUDA source. Such dynamic instrumentation can host a number of correctness checks and debugging support substantially several orders of magnitude faster than the emulator. The functionality provided by these two infrastructures is described in the following.

**The Ocelot Emulation Environment.** A key information gathering infrastructure in Ocelot is the trace generation capability coupled with event trace analyzers. These provide for correctness checking functionality such as memory alignment checks as well as performance analysis support such as the assessment of control-flow uniformity and data sharing patterns. These trace generators and event trace analyzers can be selectively attached to the application when executing on the emulator. When executing a PTX kernel, the emulator records detailed state immediately prior to and immediately after the execution of each PTX instruction, e.g., PC, memory addresses referenced, and thread ID, producing a stream of event object containing this state information. These event object streams are analyzed by individual event trace analyzers which are of two types: correctness checking and performance analysis. Correctness checking trace generators check for illegal behavior in the application and throw an exception if one is detected. Performance analysis examines the trace of an application and presents key information on its behavior patterns.

For example, a memory checker trace analyzer can detect alignment and out-of-bounds access errors in memory operations (load, store, and texture sampling instructions). Bounds checking compares every memory access with a list of valid memory allocations created at runtime. Alignment errors occur when a data is accessed at an address not a multiple of its data size. Instructions that result in an error trigger a runtime exception showing the thread ID, address, and PC. For example, listing 7.1, 7.2, and 7.3 demonstrate an unaligned memory access in CUDA form, PTX form, and Ocelot's output. Since memory is byte addressable and an integer data type is four bytes wide, memory references to an integer must be divisible by four. The example introduces an offset of 1 to the parameter pointer to create an unaligned access.

**Listing 7.1**: Example of unaligned memory access.

```
__global__ void badRef(int *A)
{
  char *b = reinterpret_cast<char *>(a);
  b += 1;
  a = reinterpret_cast<int *>(b);
  a[0] = 0;                    // faulting store
}
```

**Listing 7.2**: Same example in PTX form.

```
mov.s32 %r0, 0
ld.param.u32 %r1, [__cudaparm__Z12badRefPi___val_parama]
st.global.s32 [%r1 + 1], %r0            //offset of one
exit
```

**Listing 7.3**: Memory checker output.

```
==Ocelot== Ocelot PTX Emulator failed to run kernel "_Z12badRefPi"
with exception:
==Ocelot== [PC 2] [thread 0] [cta 0] st.global.s32 [%r1 + 1], %r0 -
Memory access 0x8151541 is not aligned to the access size ( 4 bytes )
==Ocelot== Near tracegen.cu:19:0
==Ocelot==
terminate called after throwing an instance of 'hydrazine::Exception'
what(): [PC 2] [thread 0] [cta 0] st.global.s32 [%r1 + 1], %r0 -
Memory access 0x8151541 is not aligned to the access size ( 4 bytes )
Near tracegen.cu:19:0
```

Ocelot includes many such event trace analyzers. For example, a branch trace generator and analyzer records each branch event and the number of divergent threads and generates branch statistics and branch divergence behavior such as the percentage of active threads and the number of branches taken. This can be beneficial for finding areas of high divergence and eliminating unnecessary branch divergence to speed up execution by re-factoring or otherwise modifying the application code. Another example is where the analysis of memory references can track inter-thread data flow. For example, Figure 7.8 shows the amount of inter-thread communication in many benchmark applications as a percentage of loads to shared memory, and also as a fraction of the total dynamic instruction count. This requires support within the emulator to track producer and consumer threads. Such insights are useful when tuning the memory behavior and sharing patterns between threads to maximize performance.

**Dynamic Instrumentation.** While the emulator provides significant functional fidelity at the PTX instruction set level, software emulation can be time consuming. A second tool chain developed for Ocelot is *Lynx* — an infrastructure for dynamic editing of PTX kernels to provide real-time introspection into platform behavior for both performance debugging and correctness checks [FKE+12a]. Lynx is a dynamic instrumentation infrastructure for constructing customizable program analysis tools for GPU-based, parallel architectures. It provides an extensible set of C-based language constructs to build program analysis tools that target the data-parallel programming paradigm used in GPUs. Lynx provides the capability to write instrumentation routines that are (1) *selective*, instrumenting only what is needed, (2) *transparent*, without changes to the applications' source code, (3) *customizable*,

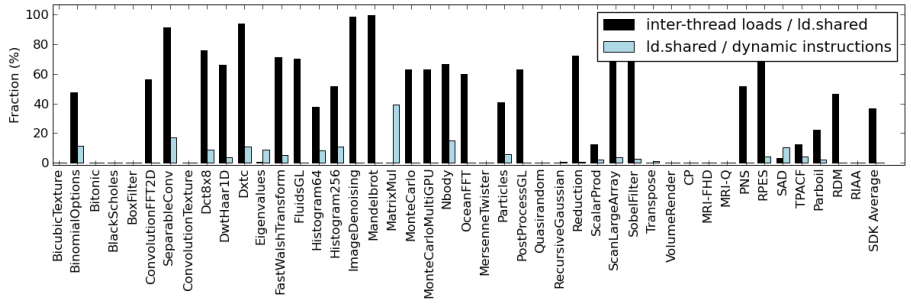


**FIGURE 7.8**: Measuring inter-thread data sharing.

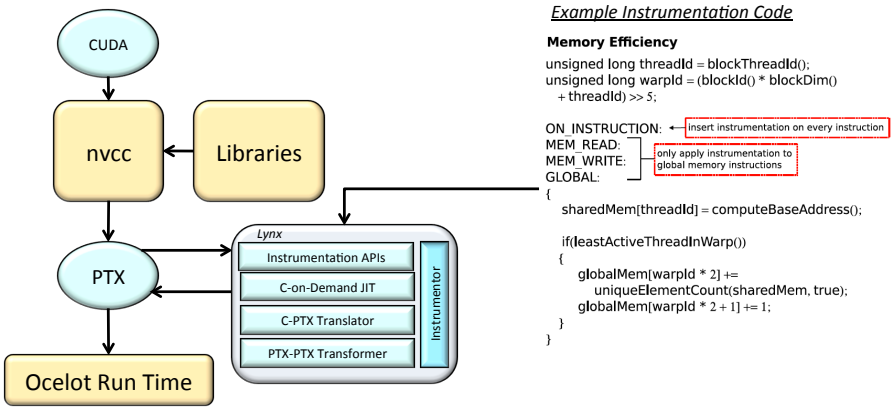


**FIGURE 7.9**: Lynx: a dynamic editing tool for PTX.

and (4) *efficient*. Lynx is embedded into the broader GPU Ocelot system, which provides run-time code generation.

An overview of Lynx is shown in Figure 7.9. Instrumentation code is specified in a C-based instrumentation language that is JIT compiled into PTX. The figure shows an example of instrumentation code that computes memory efficiency at run-time. The compiled PTX instrumentation code is then patched into the kernel using a Ocelot compiler pass over the PTX code. This pass modifies the PTX kernel to correctly place instrumentation code as well as supporting code to manage instrumentation data. Run-time support further enables the transfer of this instrumentation data to the host.

The major utility of Lynx is the ability to create user customizable code that can measure and quantify behaviors that cannot be captured by the vendor supplied tools [NVI11a, NVI11b]. In general, vendors cannot anticipate all of the functionality that their customers will need and that their instrumentation tools should provide. Lynx provides the capability to address such needs. However, it should be kept in mind that dynamic instrumentation perturbs the application code and changes its runtime behavior, e.g., cache behaviors. Lynx is best suited for measurements that are not affected by such perturbations.

In summary, Ocelot provides an open source dynamic compilation infrastructure that includes fully functional PTX emulator as the anchor for an ecosystem of productivity tools to support introspection primarily for the purpose of correctness checking and performance debugging. These open source tools fill a gap left by vendors to permit user customizable behaviors that can augment traditional GPU vendor tools. Research explorations continue to investigate the productivity tools requirements for Keeneland class machines and pursue prototypes that can enhance developer productivity and application performance.

**Virtualization.** The addition of accelerators like GPUs [NVI09b] to general purpose computational machines has considerably boosted their processing capacities. We argue that, with increasingly heterogeneous compute nodes and new usage models, it is important to move beyond current node-based allocations for next generation infrastructure. We base this argument on the following. With node-based allocations, to efficiently use heterogeneous physical nodes, application developers have to laboriously tune their codes so as to best leverage both the CPU and GPU resources present on each node. This means that codes must be re-configured for each hardware platform on which they run. For instance, for the Keeneland machine, where nodes are comprised of two Intel Xeon X5560 processors coupled with three Fermi GPUs, to efficiently use this configuration, an application must not only

accelerate a substantial portion of its code, but must do so in ways that utilize each node's twelve CPU and over one thousand GPU cores. If this is not the case, then (1) end users may be charged for node resources they do not use, and (2) machine providers may see low or inadequate levels of utilization. This implies wasted opportunities to service other jobs waiting for machine resources and higher operational costs due to potentially wasted energy in underutilized nodes.

To address these challenges, we provide software that permits applications to acquire and use exactly the cluster resources they need, rather than having to deal with coarse-grained node-based allocations. Specifically, we provide logical—*virtual*—rather than physical sets of CPU/GPU nodes (Figure 7.10), which are constructed using an abstraction termed *GPU assemblies*, where each assembly is comprised of a 'slice' of the machine containing some number of CPUs and GPUs (along with proportional use of memory as well as network resources). One slice can be an assembly consisting mainly of nodes' CPUs for running a CPU-intensive application, but then those nodes' GPUs (since the locally running CPU-intensive programs do not need them) can be made available to other, GPU-intensive programs running at the same time (i.e., running on other cluster nodes). Such sharing of GPUs can reduce the need for tuning applications to specific hardware, make it easier to fully exploit the accelerator capabilities determined by processing requirements rather than static machine configurations, and offer levels of flexibility in job scheduling not available on current systems. Furthermore, by allowing applications to specify, and use, virtual cluster

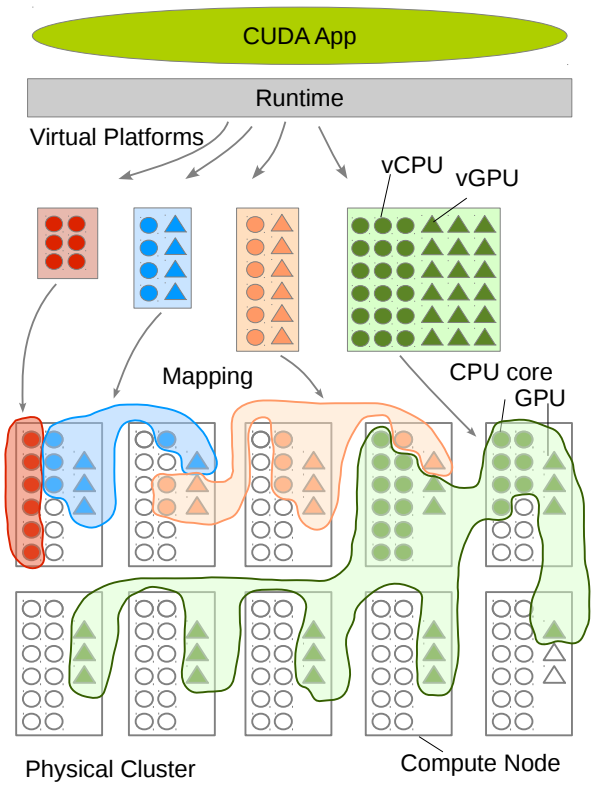


**FIGURE 7.10**: Shadowfax: the assemblies concept.

configurations, distinct from what the underlying hardware offers, we achieve an additional benefit of improved application *portability*, by obviating the need to extensively tune each code to the physical configuration of underlying hardware.

In order to evaluate the feasibility and utility of creating virtual cluster instances and of enabling fine grain sharing of machine resources, as argued above, we develop a system-level runtime—Shadowfax—which permits assembly construction and management. Shadowfax offers support for characterizing the applications that can benefit from assembly use, and it addresses the challenges in sharing physical hardware (i.e., node and network) resources across multiple assemblies. This creates new opportunities for both end users and machine providers, because when using assemblies, a job scheduler simply maps an application to a "logical" (i.e., virtual) cluster machine, i.e., a GPU assembly, and Shadowfax then tracks the physical resources available, monitors their consequent levels of utilization (and/or other performance metrics), and informs end users about the effectiveness of their choices (Figure 7.11). Specifically, Shadowfax captures and uses monitoring data about both application performance and machine utilization, and it provides such data to end users to better understand the performance of their high-end codes. It also ensures resource availability before sharing node and machine assets, by actively monitoring the use of both the CPU and GPU resources present on cluster nodes. Monitoring at system-level is used to allocate appropriate distributed resources to different assemblies, and monitoring GPU invocations at application-level provides the feedback needed to make appropriate decisions about using local vs. remote accelerators. For instance, for throughput-intensive codes, remote accelerators can be equally effective as local ones, whereas latency-sensitive codes will benefit from or require locally accessible GPUs. An additional benefit of Shadowfax instrumentation is that it can be used to isolate applications from each other, so as to prevent one application from interfering with the execution of another, e.g., by starving it or by causing undue levels of noise [CH11]. As a result, with Shadowfax, capacity computing is assisted by making it possible to map user applications to hardware resources already used by other codes, and is enhanced by providing codes that can use more GPUs than those physically present on each node with the additional resources they require.

The implementation of Shadowfax builds on the (i) interception and (ii) remoting support for GPU access and sharing, described in [MGV+11, GGS+09, GST+11], but extends it in several ways.

The interception of applications' GPU accesses is key for decoupling the application-perceived GPU context from the physical one supported by the underlying hardware. This permits both sharing of GPU resources across multiple applications, and transparent redirection of GPU operations to remote resources. GPU accesses are intercepted via a lightweight interposing library, and are redirected to a designated management process, one per node, as illustrated in Figure 7.11. The management entity communicates via RPC to other management entities, to redirect GPU operations to remote physical GPUs, and to create and maintain the required virtual platform properties. The interposing library in Shadowfax supports (a subset of) the NVIDIA CUDA API, but extensions are easily made and similar solutions can be provided for OpenCL or others. This approach permits application software to remain unchanged to take advantage of the virtual platforms provided by Shadowfax. All that is needed for applications to use Shadowfax, is to run 'runtime main' on a designated node, to run 'runtime minion ip_addr_to_main' on the remote side(s), and on the local side, we preload the interposer library with the app executable: 'LD_PRELOAD=libipser.so matrixMul...' if the code matrixMul is being run. It is a bit more involved to also add instrumentation support for detailed GPU measurement via Lynx, requiring the app to be recompiled and linked with libocelot.so rather than with libcuda.so. We note that such instrumentation support is only per specific user requests, meaning that un-instrumented applications can be run without recompilation.

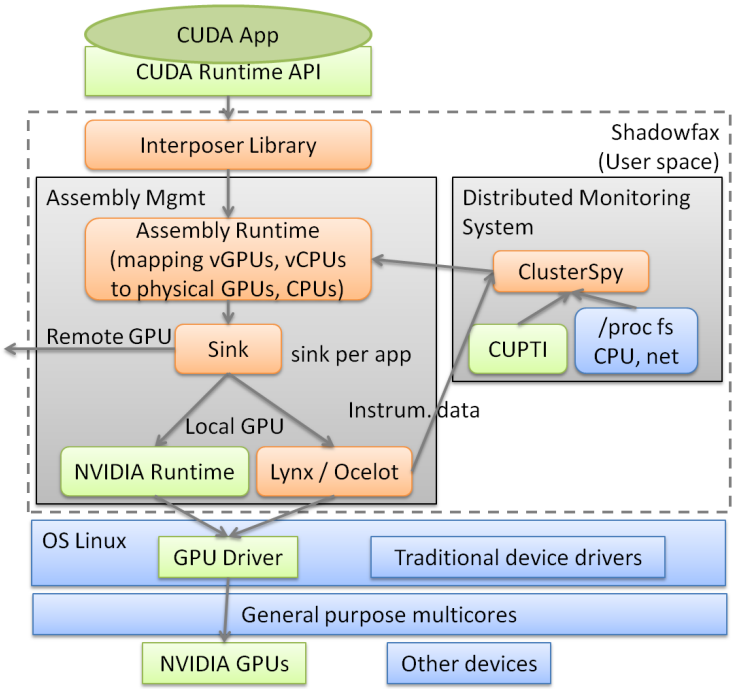


**FIGURE 7.11**: Shadowfax: the components of the system.

Additional features of the assembly middleware include (1) efficient implementation of remoting software on InfiniBand-based cluster interconnects, (2) maintainance of per-node and cluster-wide profile information about system, GPU, and assembly resources, including monitoring data about how applications exploit GPUs, and (3) use of performance models derived from profile and runtime data to evaluate a system's capacity to dynamically host workloads, considering both a node's remaining capacity as well as a workload's sensitivity to remoting. Such data includes utilization information on the CPU, in- and out-bound network traffic on the InfiniBand fabric, node memory capacity, and NVIDIA CUDA PTX instrumentation, the latter made possible through Lynx [FKE+12b]. Combined with offline analysis of workloads' runtime characteristics of GPU use (e.g., using CUDA API traces), such information enables the intelligent mapping and the scheduling strategies needed to scale applications, increase cluster utilization, and reduce interference.

In summary, with GPGPU assemblies, by multiplexing multiple virtual platforms and thereby sharing the aggregate cluster resources, we leverage *diversity* in how applications use underlying CPU, GPU, and network hardware, to better schedule the resources of heterogeneous cluster-based systems like Keeneland. Making it easy for applications to use local or remote GPUs helps obtain high application performance and enables scaling beyond physical hardware configurations, while at the same time, improving system utilization. For instance, on a 32-node hardware configuration, two instances of the LAMMPS molecular simulator—one implemented for CPUs and one for GPUs—were provided different assemblies representing individual hardware needs to achieve an increase in 74% system throughput over the pure GPU-based code alone, due to an abundance of under-utilized CPU cores, seen in Figure 7.12. Additional experimental results demonstrate low overheads for remote GPU use, particularly for applications using asynchronous CUDA calls that

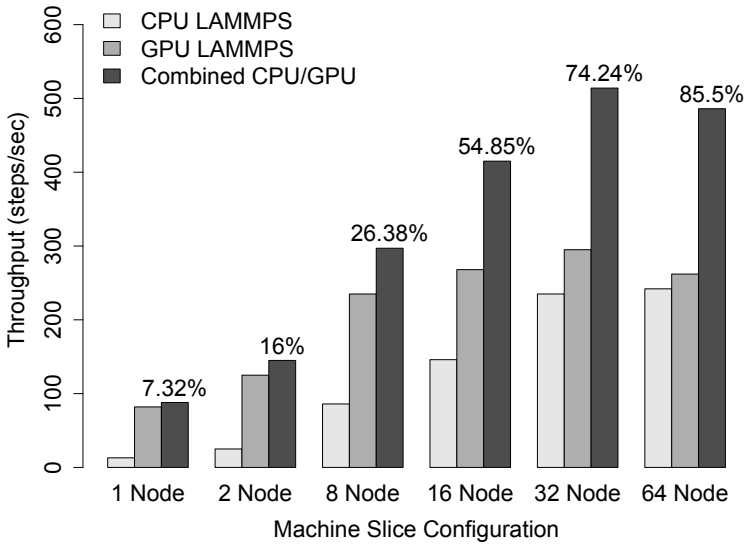


**FIGURE 7.12**: Shadowfax: cluster throughput with various hardware slices using LAMMPS.

can be batched to better utilize the interconnect and offer flexibility in exactly when some CUDA call must be run on a GPU.

Finally, and going beyond the current Keeneland based implementation of the virtual cluster functionality provided by Shadowfax, we are also exploring the utility of leveraging system-level virtualization technology to achieve assembly-like functionality without necessitating any changes at all to applications. To do so, we leverage the fact that virtualization solutions like Xen [BDF+03] can be supported with low overheads on modern hardware. This is also true for I/O virtualization of high performance fabrics such as the InfiniBand interconnect in Keeneland [LHAP06, RKGS08], and our group has made significant progress in providing low-overhead virtualization and sharing of GPU devices [GGS+09]. In fact, the initial prototype of the Shadowfax solution was implemented in a virtualized Xen-based environment [MGV+11], in part in order to directly leverage hypervisor support for isolation and scheduling on heterogeneous nodes such as those present in Keeneland [GST+11]. For the HPC domain addressed by the current realization of Shadowfax, we have chosen to build a lightweight, user-level implementation of these concepts to be able to operate on large-scale cluster machines running standard Linux operating systems and up-to-date device drivers. We are continuing, however, to explore the suitability of leveraging existing virtualization support provided by open-source hypervisors like Xen for use in large-scale systems, and new open source CUDA software available for the latest NVIDIA GPUs [KMMB12].

## 7.5 Applications and Workloads

As the Keeneland Initial Delivery System is primarily intended to help applications scientists prepare their codes for scalable heterogeneous systems, the dominant workloads are heavily represented by applications under active development. System usage is spread among a variety of science domains. For example, in the biology domain we see peptide folding on

surfaces using AMBER [she12], simulating blood flow with FMM, protein-DNA docking via a structure-based approach [hon12], and biomolecular simulations with NAMD [PBW+05]. Materials science applications include the LAMMPS molecular dynamics simulator, Quantum Monte Carlo for studying correlated electronic systems using QMCPACK [qmc12] and high temperature superconductor studies using DCA++ [ASM+08, MAM+09], a 2009 Gordon Bell Prize winner. We also see applications in high energy physics (hadron polarizability in lattice QCD) [had12], combustion (turbulence, compressible flow) and some codes with application to national security. We also see pure computer science applications, such as development and testing of numerical linear algebra libraries (MAGMA) [TDB10], new programming models like Sequoia/Legion [seq12, leg12], and improvements to existing programming models like MPI [SOHL+98].

The codes occupying the greatest proportion of node hours are as follows:

- AMBER is typically the most heavily used software package, ranging from 34% to 59% of total machine use. AMBER is a package of molecular simulation programs with full GPU support. The CPU version has been in use for more than 30 years.
- LAMMPS usage is consistently around 11% of the system. LAMMPS is 175K+ lines of classical molecular dynamics code that can be run in parallel on distributed processors with GPU support for many code features. LAMMPS can model systems with millions or billions of particles.
- MCSim usage fluctuates from 5% to 32%, depending on the month. MCSim is Monte Carlo Markov chain simulation software. MCSim is versatile; it is not tailored for a specific domain.
- TeraChem usage can be as high as 19%. TeraChem is the first computational chemistry software program written entirely from scratch to benefit from GPUs. It is used for molecular dynamics.

Since the full-scale Keeneland system focuses primarily on production science, less time is allocated to application development and computer science research. The initial allocation requests apportion 50% of the KFS system usage for testing universality in diblock copolymers (driven by materials science); 25% for understanding ion solvation at the air/water interface from adaptive QM/MM molecular dynamics (driven by chemistry); 13% for molecular dynamics of biological and nanoscale systems over microseconds, and salt effect in peptides and nucleic acids (both driven by biology); and 2% for simulation of relativistic astrophysical systems (driven by computational astrophysics).

### 7.5.1 Highlights of Main Applications

**MoBo**. During the early KIDS acceptance testing, we ported and successfully ran the main kernel (Fast Multipole Method) for a blood simulation application; the results were presented at the 2010 International Conference on High Performance Computing, Networking, Storage, and Analysis, where this paper was awarded the SC10 Gordon Bell prize [RLV+10]. The application is a fast, petaflop-scalable algorithm for Stokesian particulate flows. The goal is the direct simulation of blood, a challenging multiscale, multiphysics problem. The method has been implemented in the software library MoBo (for "Moving Boundaries"). MoBo supports parallelism at all levels, inter-node distributed memory parallelism, intra-node shared memory parallelism, data parallelism (vectorization), and fine-grained multithreading for GPUs. MoBo has performed simulations with up to 260 million *deformable* RBCs (90 billion unknowns in space). The previous largest simulation at the

same physical fidelity involved O(10,000) RBCs. MoBo achieved 0.7 PF/s of sustained performance on NCCS/Jaguar.

**AMBER**. ("Assisted Model Building with Energy Refinement") refers to two things: a set of molecular mechanical force fields for the simulation of biomolecules (which are in the public domain, and are used in a variety of simulation programs); and a package of molecular simulation programs which includes source code and demos. This package evolved from a program that was constructed in the 1970s, and now contains a group of programs embodying a number of powerful tools of modern computational chemistry, focused on molecular dynamics and free energy calculations of proteins, nucleic acids, and carbohydrates. Molecular dynamics simulations of proteins, which began about 25 years ago, are now widely used as tools to investigate structure and dynamics under a variety of conditions; these range from studies of ligand binding and enzyme reaction mechanisms to problems of denaturation and protein refolding to analysis of experimental data and refinement of structures. AMBER is the collective name for a suite of programs that allows users to carry out and analyze molecular dynamics simulations, particularly for proteins, nucleic acids, and carbohydrates. None of the individual programs carries this name, but the various parts work reasonably well together, providing a powerful framework for many common calculations. It should be recognized, however, that the code and force fields are separate; several other computer packages have implemented the AMBER force fields, and other force fields can be used within the AMBER programs. For the past 16 years, new versions of AMBER have been released on a two-year schedule. Under way are continued improvements in code cleanup, with an eye toward maintainability, portability, and efficiency. Amber is a code that is heavily used by its developers, and reflects their interests, but attempts are being made to lower the learning curve for scientists new to the simulation field. For more information, see http://ambermd.org/.

**LAMMPS**. ("Large-scale Atomic/Molecular Massively Parallel Simulator") is a molecular dynamics application from Sandia National Laboratories. LAMMPS makes use of MPI for parallel communication and is free open source code. LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and course grained systems using a variety of force fields and boundary conditions. LAMMPS is freely available source code and is designed to be easy to modify or extend with new capabilities, such as force fields, atom types, boundary conditions, or diagnostics. The 1995 reference paper *Fast parallel algorithms for short-range molecular dynamics* [Pli95] has more than 2,500 citations.

Figure 7.13 shows results on Keeneland from a 250k particle simulation of nanodroplet formation using the Gay-Berne potential in LAMMPS. This figure shows the speedup from using all three GPUs in each of up to 32 nodes of KIDS (with varying numbers of processes per node) relative to using all 12 CPU cores on each node.

### 7.5.2 Benchmarks

To understand the usefulness of a system for running a particular workload, it is critical to measure the performance of real-world applications from that workload running on the system. However, because real-world applications can be highly complex and because they tend to exercise many facets of a system, it can be very useful to focus on the performance of each facet individually using benchmark programs. To better understand the strengths of the Keeneland systems, we used the TOP500 High-Performance Linpack (HPL) and the Scalable HeterOgeneous Computing (SHOC) benchmark suites.

**TOP500 HPL.** High-Performance Linpack is a commonly used reference point for supercomputers involving solutions to dense linear systems in double precision. On KIDS,

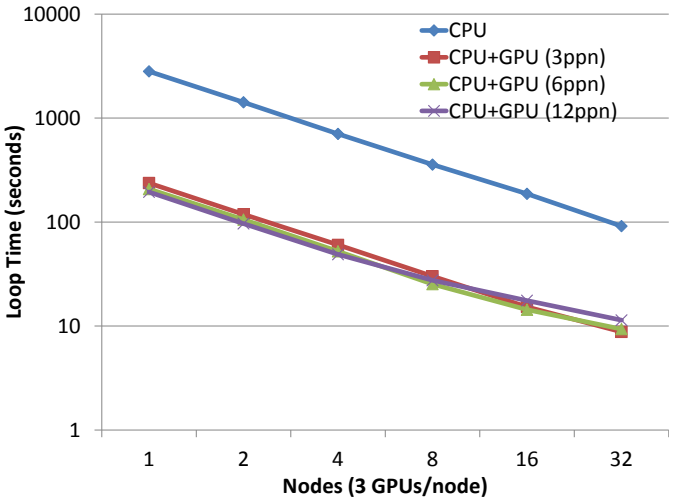


**FIGURE 7.13**: CPU and GPU results from a 250k particle Gay-Berne LAMMPS simulation on KIDS.

HPL achieved 106.30 TFLOPS, placing Keeneland at position 111 in November 2011's TOP500 list and rating its power efficiency at 901 MFLOPS/W.

Initially, our experiments on HPL only a few days after KIDS was deployed in November of 2011, reached 63.92 TFLOPS; this early version of HPL did not stream data to the GPU, and hence it had a relatively low efficiency for floating-point rate. Even with this inefficiency, KIDS ranked at position 117 on the November 2010 TOP500 list, and given its low power usage of 94.4 kW during the run, placed the system as the 9th most power-efficient supercomputer in the world at 677 MFLOPS/W on the Green500 list.

**SHOC.** Early on, our Keeneland team could not find a set of scalable heterogeneous GPU benchmarks for testing the reliability and performance of these types of systems, so we designed the Scalable Heterogeneous Computing (SHOC) benchmark suite [DMM+10]. SHOC plays an integral part of not only the KIDS and KFS acceptance tests but also health checks on the deployed systems. For more information about SHOC and performance results on KIDS, see Section 7.8.

## 7.6 Data Center and Facility

The Keeneland systems are co-located at Oak Ridge National Laboratory's Leadership Computing Facility (OLCF), along with other HPC systems from DOE, NSF, NOAA, and other customers. Power and cooling are provided as part of the facility co-location fees.

The KID system was originally located in a traditionally designed datacenter that utilizes a raised floor design. Computer Room Air Conditioners (CRACs) located around the perimeter of the room pull air from near the ceiling, chill the air, and then duct it below a 36" floor to provide positive cold air pressure under the raised floor. Perforated tiles are then placed in front of the equipment racks to direct chilled air to the equipment. Hot air is exhausted from the rear of the equipment rack into the datacenter. Power is provided via

monitored circuits fed from the facility's line side. Due to the large power draw of such a system, it was not cost effective to connect the KID system to the facility UPS.

In conjunction with the deployment of the KFS system, KIDS was relocated to a new datacenter. Both the KID and KFS systems are installed in a newly remodeled datacenter designed to take advantage of cold isle containment techniques and in-row chilled water air handlers. The new datacenter is designed with an 18" raised floor. The systems are deployed in 20 rack sections, nicknamed pods, with the racks deployed in rows with the fronts of the equipment racks opposing each other. Due to the limited under-floor space, In-Row air handlers were installed between equipment racks to pull hot air from the rear of the equipment racks (the hot isle), remove the heat from the air, and deposit it in the enclosed area in the front of the racks (the cold isle). This design makes use of containment techniques, such as blanking panels in the racks and panels over covering the rows, to ensure that the hot air from the exhaust does not infiltrate the cold isle. By reducing the mixing of hot and cold air at the equipment intake, it is possible to ensure a constant inlet temperature, thus providing more effective cooling. Both systems are powered via dedicated transformers located at the end of the pod. Using dedicated transformers reduces the impact of maintenance performed on other systems to the Keeneland systems.

## 7.7 System Statistics

The KID system has been in operation for almost two years, and during this time, we have monitored its operation closely. The KFS system has only been operational for a few months, and as such, its statistics have been dominated by transients. Hence, we discuss only our KID system statistics here.

KIDS has been used for software and application development activities; for education, outreach, and training; and by many groups to test codes in a GPU-accelerated environment. At a high level, the research areas utilizing KIDS include computer science and computational research, astronomical sciences, atmospheric sciences, behavioral and neural sciences, biological and critical systems, chemistry, design and manufacturing systems, Earth sciences, materials research, mathematical sciences, mechanical and structural systems, molecular biosciences, physics, cross-disciplinary activities, and education/training. Both the number of users and the breadth of science fields is well beyond what was initially envisioned for KIDS. Unlike production resources, the majority of KIDS usage is discretionary, rather than allocated by a review committee. KIDS has an architecture very similar to the KFS system, so it can be used for production workload as necessary, and applications that have been successfully tuned for the KIDS architecture should run well on the KFS system architecture. Although the number of accounts has been reduced as we prepare to go into production, KIDS had 83 project accounts and 244 users in May 2012 with almost half of them active in any given month.

KIDS utilization has fluctuated around 60% during the past year with the peak utilization of 84.5% in September 2011, before we changed our usage policy to encourage more development jobs and larger scale jobs. May 2012 is an example where the utilization was down by design as we added more capability periods to allow users to run scaling tests for SC12 papers that were due, and we needed blocks of time for Keeneland staff to test the system after the upgrade from NVIDIA M2070 GPUs to M2090 GPU (see Figure 7.14). The draining of KIDS for these jobs resulted in lower utilization, but provided an opportunity for paper authors to complete their work in time for submission and our preparation for

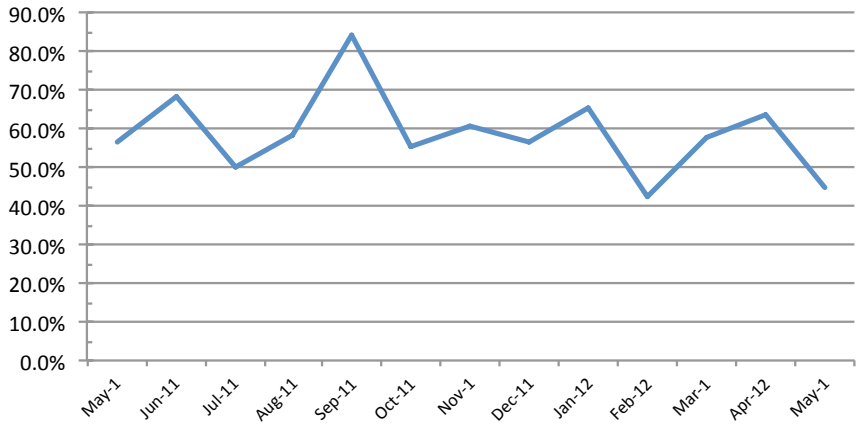


**FIGURE 7.14**: KIDS percent utilization/month.

XSEDE production. The number of large jobs run in May 2012 was more than the prior two months combined.

As shown by Figure 7.15, there is significant variability of the workload distribution from month to month. There has usually been a large portion of the workload for small node-count jobs, and there has been a general shift toward mid-sized jobs. Workload for the large jobs tends to be smaller because these jobs need to be scheduled once per week and tend to be scaling runs that, unlike production runs, tend to be shorter in duration.

KIDS shows a fairly common usage pattern: the majority of jobs are small single-node jobs, but the majority of the workload is comprised of single-node jobs and multimode jobs in the range of 9-32 nodes. KIDS is being used in a hybrid mode, where part of the system is reserved for development work during prime hours of the day while pre-production (capacity) work is also being accomplished. The entire machine is reserved once per week for capability jobs that need more than 72 nodes for a single job. This capability reservation

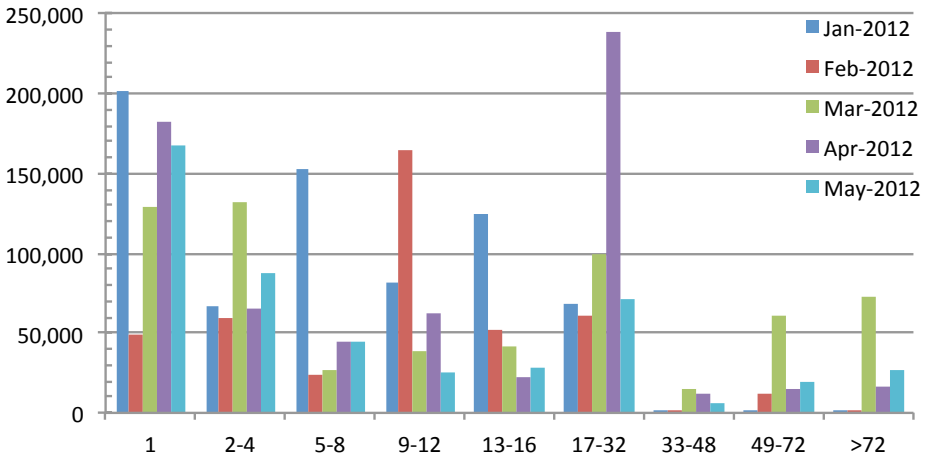


**FIGURE 7.15**: KIDS workload distribution by numbers of nodes/job.

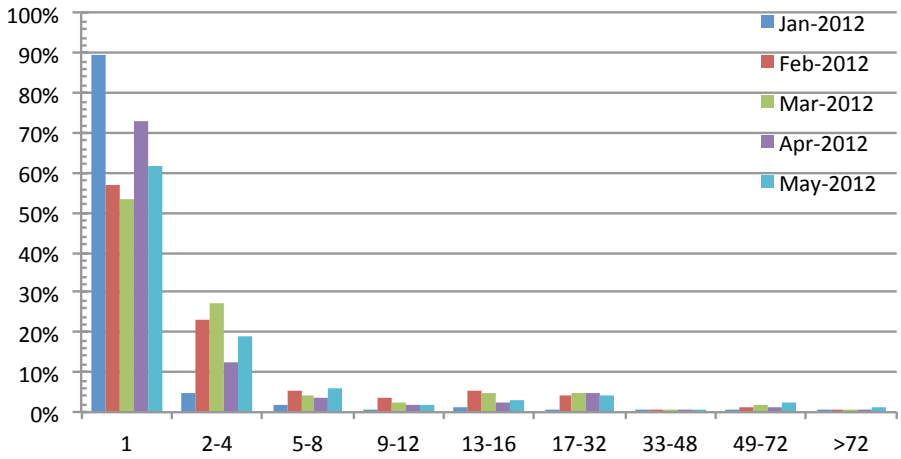


**FIGURE 7.16**: KIDS workload distribution by percentage of jobs at number of nodes.

allows users to perform scaling studies and benchmark codes in an environment that is not generally available (see Figures 7.16 and 7.17).

The Keeneland user community represents a wide variety of scientific disciplines. However, the majority of the workload comes from molecular biosciences and physics, followed by chemistry and materials research. Table 7.3 shows the dominance of those areas and the fluctuation in the workload for those research areas over the six-month period from December 2011 through May 2012. The workload is presented as node-hours of usage.

The Keeneland Initial Delivery System is used in a hybrid mode with preference given to development activities during prime hours but with production workloads allowed after hours or when the system is not fully utilized. Looking at only the aggregate usage masks some of the detail of how the machine is used by various projects. Even within a particular research area, there are differences in how the system is used. Table 7.4 shows that the production workload for the highest usage projects in node-hours/month fluctuates from

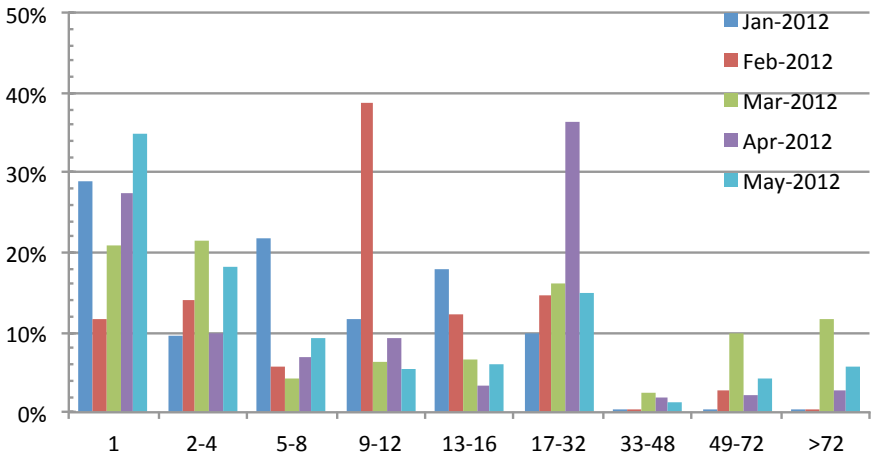


**FIGURE 7.17**: KIDS workload distribution by percentage of machine at number of nodes.

**TABLE 7.3**: KIDS node-hours by scientific discipline.

| Research Area | Dec11 | Jan12 | Feb12 | Mar12 | Apr12 | May12 |
|---|---|---|---|---|---|---|
| Molecular Biosciences | 12,353 | 11,643 | 16,868 | 16,684 | 32,317 | 13,946 |
| Physics | 6,782 | 27,609 | 10,894 | 18,298 | 9,946 | 5,794 |
| Chemistry | 15,957 | 11,926 | 646 | 5,157 | 900 | 3,721 |
| Materials Research | 10,902 | 1,197 | 469 | 3,570 | 2,764 | 1,868 |
| Scientific Computing | 0 | 0 | 86 | 371 | 2,708 | 7,024 |
| Cross-Disciplinary | 1,605 | 17,987 | 195 | 929 | 336 | 523 |
| Computation Research | 1,873 | 741 | 504 | 1,995 | 547 | 3 |
| Bio./Critical Systems | 0 | 1,876 | 2,062 | 0 | 0 | 0 |

month to month but remains at high utilization. This represents more of a production workload, while other projects farther down the table demonstrate a start-up/development usage that starts low, peaks, and then drops back down. Each of the projects used enough resources in a given month to be one of the top ten users in at least one month, but over the six-month period, most actually represent a relatively low utilization. A few projects have multiple cycles of peak utilization followed by low utilization.

During the six-month period discussed above, the top five usage projects per month averaged over 80% of the KIDS workload. As Keeneland moves into a stable XSEDE production environment, the workload on the KFS system is expected to come from a limited number of users who either have jobs that require a large scalable system to complete in a reasonable time frame or who have an equivalent aggregate workload. The development workload is expected to continue to run on KIDS with overflow from the KFS system using the remaining cycles. As the XSEDE workload transitions to the KFS system, the KFS system is exhibiting longer queue backlogs and a higher utilization, while KIDS continues to exhibit the same or a higher degree of bimodal usage. Considering that a single project requested (but did not receive) half of the available allocation on the KFS system for a

**TABLE 7.4**: KIDS workload.

| Research Area | Dec11 | Jan12 | Feb12 | Mar12 | Apr12 | May12 |
|---|---|---|---|---|---|---|
| Molecular Biosciences 1 | 10,427 | 6,577 | 7,397 | 16,683 | 32,317 | 12,738 |
| Physics 1 | 4,887 | 12,937 | 8,676 | 4,851 | 3,925 | 3,791 |
| Chemistry 1 | 14,490 | 11,060 | 642 | 5,157 | 900 | 3,721 |
| Materials Research 1 | 10,853 | 1,197 | 304 | 2,266 | 2,699 | 1,868 |
| Physics 2 | 1,871 | 11,191 | 646 | 946 | 3,016 | 446 |
| Physics 3 | 0 | 712 | 292 | 12,501 | 3,005 | 1,558 |
| Molecular Biosciences 2 | 561 | 5,066 | 9,471 | 1 | 0 | 1,209 |
| Scientific Computing 1 | 0 | 0 | 86 | 255 | 1,872 | 6,285 |
| Molecular Biosciences 3 | 67 | 4 | 54 | 1,555 | 3,254 | 1,993 |
| Cross-Disciplinary 1 | 1,569 | 2,995 | 0 | 0 | 0 | 0 |
| Physics 4 | 24 | 2,769 | 1,280 | 0 | 0 | 0 |
| Bio./Critical Systems 1 | 0 | 1,876 | 2,062 | 0 | 0 | 0 |
| Computation Research 1 | 1,873 | 0 | 460 | 1,238 | 0 | 0 |
| Computation Research 2 | 0 | 741 | 44 | 757 | 547 | 3 |
| Materials Research 2 | 49 | 0 | 164 | 1,304 | 66 | 0 |
| Molecular Biosciences 4 | 1,395 | 0 | 0 | 0 | 0 | 0 |
| Chemistry 2 | 967 | 228 | 4 | 0 | 0 | 0 |
| Cross-Disciplinary 2 | 0 | 0 | 195 | 857 | 67 | 0 |
| Scientific Computing 2 | 0 | 0 | 0 | 116 | 811 | 27 |
| Cross-Disciplinary 3 | 36 | 15 | 0 | 71 | 268 | 523 |
| Scientific Computing 3 | 0 | 0 | 0 | 0 | 25 | 711 |
| Chemistry 3 | 0 | 638 | 0 | 0 | 0 | 0 |

year, it is clear that there is a significant demand for this type of system and that system utilization will be high over its lifecycle.

## 7.8 Scalable Heterogeneous Computing (SHOC) Benchmark Suite

As systems like Keeneland become more common, it is important to be able to compare and contrast architectural designs and programming systems in a fair and open forum. To this end, the Keeneland project has supported the development of the Scalable HeterOgeneous Computing benchmark suite (SHOC)[1].

The SHOC benchmark suite was designed to provide a standardized way to measure the performance and stability of non-traditional high performance computing architectures. The SHOC benchmarks are distributed using MPI and effectively scale from a single device (GPU or CPU) to a large cluster.

The SHOC benchmarks are divided into two primary categories: stress tests and performance tests. The stress tests use computationally demanding kernels to identify OpenCL devices with bad memory, insufficient cooling, or other component defects. The other tests measure many aspects of system performance on several synthetic kernels as well as common parallel operations and algorithms. The performance tests are further subdivided according to their complexity and the nature of the device capability they exercise.

In addition to OpenCL-based benchmarks, SHOC also includes a Compute Unified Device Architecture (CUDA) version of its benchmarks for comparison with the OpenCL version. As both languages support similar constructs, kernels have been written with the same optimizations in each language.

**Level Zero: "Speeds and Feeds."** SHOC's level zero tests are designed to measure low-level hardware characteristics (the so-called "feeds and speeds"). All level zero tests use artificial kernels, and results from these benchmarks represent an empirical upper bound on realized performance. As these are designed for consistency, they can be used not just as a comparative performance measure, but can also detect a variety of issues, such as lower than expected peak performance, chipsets with only eight PCI-Express (PCIe) lanes, or systems with large variations in kernel queueing delays.

**Level One: Parallel Algorithms**. Level One benchmarks measure basic parallel algorithms, such as the Fast Fourier Transform (FFT) or the parallel prefix sum (a.k.a. scan). These algorithms represent common tasks in parallel processing and are commonly found in a significant portion of the kernels of real applications.

These algorithms vary significantly in performance characteristics, and stress different components of a device's memory subsystem and functional units. Several of the benchmarks are highly configurable and can span a range of the spectrum based on problem size or other input parameters.

**Level Two: Application Kernels**. Level two kernels are extracted routines from production applications:

**SHOC on Keeneland**. SHOC was a major component in the acceptance test for the KID system; indeed, the KIDS acceptance test was a primary motivation for the initial development of SHOC. SHOC was also used in the acceptance test for the KFS system.

Figure 7.18 shows the performance of the SHOC Stencil2D benchmark program on KIDS. Stencil2D implements a nine-point stencil operation over the values in a two-dimensional matrix. In its MPI version, Stencil2D splits the matrix across all the available MPI tasks and

[1] http://j.mp/shocmarks

**TABLE 7.5**: SHOC components.

| Component | Description |
|---|---|
| | **Level Zero** |
| Bus Speed Download and Readback | Measures the bandwidth of the interconnection bus between the host processor and the OpenCL device (typically the PCIe bus) by repeatedly transferring data of various sizes to and from the device. |
| Device Memory Bandwidth | Measures bandwidth for all device memory address spaces, including global, local, constant, and image memories. The global address space is benchmarked using both coalesced and uncoalesced memory accesses. |
| Kernel Compilation | OpenCL kernels are compiled at runtime, and this benchmark measures average compilation speed and overheads for kernels of varying complexity. |
| Peak FLOPS | Measures peak floating point (single or double precision) operations per second using a synthetic workload designed to fully exercise device functional units. |
| Queueing Delay | Measures the overhead of launching a kernel in OpenCL's queueing system. |
| Resource Contention | Measures contention on the PCIe bus between OpenCL data transfers and MPI message passing. |
| | **Level One** |
| FFT | Measures the performance of a two-dimensional Fast Fourier Transform. The benchmark computes multiple FFTs of size 512 in parallel. |
| MD | Measures the speed of a simple pairwise calculation of the Lennard-Jones potential from molecular dynamics using neighbor lists. |
| Reduction | Measures the performance of a sum reduction operation using floating point data. |
| Scan | Measures the performance of the parallel prefix sum algorithm (also known as Scan) on a large array of floating point data. |
| GEMM | This benchmark measures device performance on an OpenCL version of the general matrix multiply (GEMM) BLAS routine. |
| Sort | Measures device performance for a very fast radix sort algorithm [SHG09] which sorts key-value pairs of single precision floating point data. |
| Stencil2D | Measures performance for a standard two-dimensional nine point stencil calculation. |
| Triad | An OpenCL version of the STREAM Triad benchmark[DL05]. |
| | **Level Two** |
| S3D | Measures the performance of the S3D's computationally intensive getrates kernel, which calculates the rate of chemical reactions for the 22 species of the ethylene-air chemistry model. |
| QTC | Measures the speed of a complex quality threshold clustering operation. QTC clustering is conceptually similar to the more well-known $k$-means clustering, but requires no prior knowledge of the appropriate value of $k$. |

a halo exchange is used between tasks that are neighbors with respect to the program's two-dimensional Cartesian task organization. The program uses a weak-scaling model, where the total size of the matrix is proportional to the number of tasks available. The figure shows the program's performance as the number of GPUs used was varied, for both CUDA and OpenCL versions of the program. For this experiment, we used a pre-release version of SHOC version 1.1.4 with CUDA version 4.1. We placed three MPI tasks per compute node so that each task controlled a GPU and no GPUs were left idle. We did not use any process affinity control for this experiment. The algorithms used by the CUDA and OpenCL versions of Stencil2D are the same. Unlike the case with many of the SHOC benchmark programs, the MPI+OpenCL version of this program slightly outperforms the

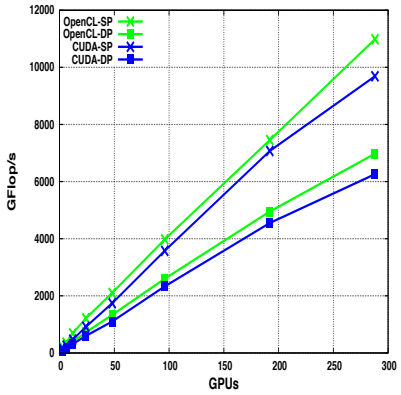


**FIGURE 7.18**: Performance of SHOC Stencil2D benchmark on KIDS, CUDA and OpenCL versions.

MPI+CUDA version. We hypothesize that this is a result of using GPU work distributions that, by default, are slightly more amenable to the OpenCL version. Also, the GPU Direct software was not enabled on KIDS when these data were collected. Because of the frequent halo exchanges needed in this program, we expect that the GPU Direct optimization would provide a substantial performance boost to the MPI+CUDA version that would allow it to outperform the OpenCL version.

---

## Acknowledgments

Keeneland is funded by the National Science Foundation's Office of Cyberinfrastructure under award #0910735. The Keeneland team includes Georgia Institute of Technology, Oak Ridge National Laboratory, and the University of Tennessee at Knoxville.