Understanding Portability of a High-Level Programming Model on Diverse HPC Architectures

Jeffrey S. Vetter
Seyong Lee, Joel Denny, Jungwon Kim, et al.

Presented to COEPP
Glendale, AZ

19 Apr 2016
Executive Summary

• Architectures are growing more complex
  – This will get worse; not better

• Programming systems must provide performance portability (in addition to functional portability)!!

• Diverse heterogeneous systems including FPGAs

• Programming NVM systems is the next major challenge
# Current ASCR Computing At a Glance

<table>
<thead>
<tr>
<th>System attributes</th>
<th>NERSC Now</th>
<th>OLCF Now</th>
<th>ALCF Now</th>
<th>NERSC Upgrade</th>
<th>OLCF Upgrade</th>
<th>ALCF Upgrades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Installation</td>
<td>Edison</td>
<td>TITAN</td>
<td>MIRA</td>
<td>Cori 2016</td>
<td>Summit 2017-2018</td>
<td>Theta 2016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Aurora 2018-2019</td>
</tr>
<tr>
<td>System peak (PF)</td>
<td>2.6</td>
<td>27</td>
<td>10</td>
<td>&gt; 30</td>
<td>150</td>
<td>&gt; 8.5</td>
</tr>
<tr>
<td>Peak Power (MW)</td>
<td>2</td>
<td>9</td>
<td>4.8</td>
<td>&lt; 3.7</td>
<td>10</td>
<td>1.7</td>
</tr>
<tr>
<td>Total system memory</td>
<td>357 TB</td>
<td>710TB</td>
<td>768TB</td>
<td>~1 PB DDR4 + High Bandwidth Memory (HBM) + 1.5 PB persistent memory</td>
<td>&gt; 1.74 PB DDR4 + HBM + 2.8 PB persistent memory</td>
<td>&gt;480 TB DDR4 + High Bandwidth Memory (HBM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 7 PB High Bandwidth On-Package Memory Local Memory and Persistent Memory</td>
</tr>
<tr>
<td>Node performance (TF)</td>
<td>0.460</td>
<td>1.452</td>
<td>0.204</td>
<td>&gt; 0</td>
<td>&gt; 40</td>
<td>&gt; 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 17 times Mira</td>
</tr>
<tr>
<td>Node processors</td>
<td>Intel Ivy Bridge</td>
<td>AMD Opteron</td>
<td>Nvidia Kepler</td>
<td>64-bit PowerPC A2</td>
<td>Intel Knights Landing many core CPUs</td>
<td>Intel Knights Landing Xeon Phi many core CPUs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Multiple IBM Power9 CPUs &amp; multiple Nvidia Volta GPUs</td>
<td>Knights Hill Xeon Phi many core CPUs</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>5,600 nodes</td>
<td>18,688 nodes</td>
<td>49,152</td>
<td>9,300 nodes</td>
<td>~3,500 nodes</td>
<td>&gt;2,500 nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,900 nodes in data partition</td>
<td></td>
<td>&gt;50,000 nodes</td>
</tr>
<tr>
<td>System Interconnect</td>
<td>Aries</td>
<td>Gemini</td>
<td>5D Torus</td>
<td>Aries</td>
<td>Dual Rail EDR-IB</td>
<td>Aries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd Generation Intel Omni-Path Architecture</td>
</tr>
<tr>
<td>File System</td>
<td>7.6 PB 168 GB/s, Lustre®</td>
<td>32 PB 1 TB/s, Lustre®</td>
<td>26 PB 300 GB/s GPFS™</td>
<td>28 PB 744 GB/s Lustre®</td>
<td>120 PB 1 TB/s GPFS™</td>
<td>10PB, 210 GB/s Lustre initial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150 PB 1 TB/s Lustre®</td>
</tr>
</tbody>
</table>
Core, Processor Architectures

- **LT v. TO Cores**
  - GPUs (discrete, integrated)
  - FPGAs
- **SIMD/short vector**
- **SMT, threading models**
- **DVFS (incl Turboboost)**
- **Special Purpose**
  - RNGs
  - AES, video engines
  - Transactional memory
  - Virtualization support
- **Reconfigurable computing**
- **etc**
Integration, M&A Dominate Discussion
Complex Programming Models

System: MPI, Legion, HPX, Charm++, etc

Node: OpenMP, Pthreads, U-threads, etc

Cores: OpenACC, CUDA, OpenCL, OpenMP4, …

- Low overhead
- Resource contention
- Locality
- SIMD
- NUMA, HBM
- Memory use, coalescing
- Data orchestration
- Fine grained parallelism
- Hardware features
Programming
Heterogeneous Systems
Vertically integrated toolchain for programming systems
- ARES is not trying to build a complete toolchain, but rather leverage other software

Define an open-source, extensible, universal High-Level Intermediate Representation (HLIR) leveraging the widely adopted LLVM infrastructure

HLIR Analysis and optimization passes can be applied to any Frontend

HLIR enables higher level analysis and transformation than low level IRs

Lowered to LLVM or native support (e.g., CUDA)
Understanding Performance Portability of High-level Programming Models for Heterogeneous Systems

• Problem
  Directive-based, high-level accelerator programming models such as OpenACC provide code portability. But how does it fare on performance portability? And what architectural features/compiler optimizations affect the performance portability? And how much?

• Solution
  – Proposed a high-level, architecture-independent intermediate language (HeteroIR) to map high-level programming models (e.g., OpenACC) to diverse heterogeneous devices while maintaining portability.
  – Using HeteroIR, port and measure the performance portability of various OpenACC applications on diverse architectures.

• Results
  – Using HeteroIR, OpenARC ported 12 OpenACC applications to diverse architectures (NVIDIA CUDA, AMD GCN, and Intel MIC), and measured the performance portability achieved across all applications.
  – HeteroIR abstracts out the common architecture functionalities, which makes it easy for OpenARC (and other compilers) to support diverse heterogeneous architectures.
  – HeteroIR, combined with rich OpenARC directives and built-in tuning tools, allows OpenARC to be used for various tuning studies on diverse architectures.

<table>
<thead>
<tr>
<th></th>
<th>CUDA</th>
<th>GCN</th>
<th>MIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Program version of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUDA</td>
<td>100</td>
<td>84</td>
<td>65</td>
</tr>
<tr>
<td>GCN</td>
<td>91</td>
<td>100</td>
<td>67</td>
</tr>
<tr>
<td>MIC</td>
<td>58</td>
<td>68</td>
<td>100</td>
</tr>
</tbody>
</table>

Overall Performance Portability

- Better perf. portability among GPUs
- Lesser across GPUs and MIC
- Main reasons
  - Parallelism arrangement
  - Compiler optimizations: e.g. device-specific memories, unrolling etc.

Intelligent selection of optimizations based on target architecture

Figure 5: Memory Coalescing Benefits on Different Architectures: MIC is impacted the least by the non-coalesced accesses.

Figure 7: Impact of Tiling Transformation: MATMUL shows higher benefits than JACOBI owing to more contiguous accesses.

Figure 9: Effects of Loop Unrolling - MIC shows benefits on unrolling.

Fig. 11: Comparison of hand-written CUDA/OpenCL programs against auto-tuned OpenARC code versions: Tuned OpenACC programs perform reasonably well against hand-written codes.
OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing

• **Problem**
  – Reconfigurable computers, such as FPGAs, offer more performance and energy efficiency for specific workloads than other heterogeneous systems, but their programming complexities and low portability have limited their deployment in large scale HPC systems.

• **Solution**
  – Proposed an OpenACC-to-FPGA translation framework, which performs source-to-source translation of the input OpenACC program into an output OpenCL code, which is further compiled to an FPGA program by the underlying backend Altera OpenCL compiler.

• **Recent Results**
  – Proposed several FPGA-specific OpenACC compiler optimizations and pragma extensions to achieve higher throughput.
  – Evaluated the framework using eight OpenACC benchmarks, and measured performance variations on diverse architectures (Altera FPGA, NVIDIA/AMD GPUs, and Intel Xeon Phi).

• **Impact**
  – Proposed translation framework is the first work to use a standard and portable, directive-based, high-level programming system for FPGAs.
  – Preliminary evaluation of eight OpenACC benchmarks on an FPGA and comparison study on other accelerators identified that the unique capabilities of an FPGA offer new performance tuning opportunities different from other accelerators.

Reconfigurable Computing Tests Performance Portability in a New Dimension

Listing 4: Altera OpenCL (AOCL) Channel Example

```c
#pragma acc data copyout(a[0:N]) create(b[0:N]) \ copyin(c[0:N])
{
#pragma acc kernels loop gang worker present(b, c)
    for (i=0; i<N; i++) b[i] = c[i] + c[i];
#pragma acc kernels loop gang worker present(a, b)
    for (i=0; i<N; i++) a[i] = b[i];

    (a) Input OpenACC code

#pragma acc data copyout(a[0:N]) pipe(b[0:N]) \ copyin(c[0:N])
{
#pragma acc kernels loop gang worker pipeout(b) present(c)
    for (i=0; i<N; i++) b[i] = c[i] + c[i];
#pragma acc kernels loop gang worker pipeln(b) present(a)
    for (i=0; i<N; i++) a[i] = b[i];
}

(b) Modified OpenACC code for kernel-pipelining

#pragma OPENCL EXTENSION cl_alterx_channels : enable
channel float pipe_b;
__kernel void kernel0(__global float * c)
{
    int i = get_global_id(0);
    write_channel_alterx(pipe_b, (c[i]+c[i]));
}
__kernel void kernel1(__global float * a)
{
    int i = get_global_id(0);
    a[i] = read_channel_alterx(pipe_b);
}

(c) Output OpenCL code with channels
```

Figure 2: FPGA OpenCL Architecture

Figure 3: Difference in Global Memory Access Pattern as a Result of Channels Implementation

Emerging Non-volatile Memory Systems
Attendees envisioned two possible architectural swim lanes:
1. Homogeneous many-core thin-node system
2. Heterogeneous (accelerator + CPU) fat-node system

<table>
<thead>
<tr>
<th>System attributes</th>
<th>2009</th>
<th>“Pre-Exascale”</th>
<th>“Exascale”</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 PF</td>
<td>100-200 PF/s</td>
<td>1 Exaflop/s</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>15 MW</td>
<td>20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>5 PB</td>
<td>32–64 PB</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>150 PB</td>
<td>500 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>0.5 TF</td>
<td>7 TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>0.1 TB/s</td>
<td>1 TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(100)</td>
<td>O(1,000)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>500,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Node interconnect BW</td>
<td>1.5 GB/s</td>
<td>150 GB/s</td>
<td>1 TB/s</td>
</tr>
<tr>
<td>IO Bandwidth</td>
<td>0.2 TB/s</td>
<td>10 TB/s</td>
<td></td>
</tr>
<tr>
<td>MTTI</td>
<td>day</td>
<td>O(1 day)</td>
<td>O(0.1 day)</td>
</tr>
</tbody>
</table>
Memory Systems are Diversifying

- HMC, HBM/2/3, LPDDR4, GDDR5X, WIDEIO2, etc
- 2.5D, 3D Stacking
- New devices (ReRAM, PCRAM, STT-MRAM, Xpoint)
- Configuration diversity
  - Fused, shared memory
  - Scratchpads
  - Write through, write back, etc
  - Consistency and coherence protocols
  - Virtual v. Physical, paging strategies

![Memory Systems Diagram](https://www.micron.com/~media/track-2/images/content/images/content_image_hmc.jpg?la=en)

---


NVVRAM Technology Continues to Improve – Driven by Market Forces
# Comparison of Emerging Memory Technologies

<table>
<thead>
<tr>
<th></th>
<th>SRAM</th>
<th>DRAM</th>
<th>eDRAM</th>
<th>2D NAND Flash</th>
<th>3D NAND Flash</th>
<th>PCRAM</th>
<th>STTRAM</th>
<th>2D ReRAM</th>
<th>3D ReRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Retention</strong></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Cell Size (F²)</strong></td>
<td>50-200</td>
<td>4-6</td>
<td>19-26</td>
<td>2-5</td>
<td>&lt;1</td>
<td>4-10</td>
<td>8-40</td>
<td>4</td>
<td>&lt;1</td>
</tr>
<tr>
<td><strong>Minimum F demonstrated (nm)</strong></td>
<td>14</td>
<td>25</td>
<td>22</td>
<td>16</td>
<td>64</td>
<td>20</td>
<td>28</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td><strong>Read Time (ns)</strong></td>
<td>&lt; 1</td>
<td>30</td>
<td>5</td>
<td>10⁴</td>
<td>10⁴</td>
<td>10-50</td>
<td>3-10</td>
<td>10-50</td>
<td>10-50</td>
</tr>
<tr>
<td><strong>Write Time (ns)</strong></td>
<td>&lt; 1</td>
<td>50</td>
<td>5</td>
<td>10⁵</td>
<td>10⁵</td>
<td>100-300</td>
<td>3-10</td>
<td>10-50</td>
<td>10-50</td>
</tr>
<tr>
<td><strong>Number of Rewrites</strong></td>
<td>10¹⁶</td>
<td>10¹⁶</td>
<td>10¹⁶</td>
<td>10⁴-10⁵</td>
<td>10⁴-10⁵</td>
<td>10⁸-10¹⁰</td>
<td>10¹⁵</td>
<td>10⁸-10¹²</td>
<td>10⁸-10¹²</td>
</tr>
<tr>
<td><strong>Read Power</strong></td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Write Power</strong></td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Power (other than R/W)</strong></td>
<td>Leakage</td>
<td>Refresh</td>
<td>Refresh</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Sneak</td>
<td>Sneak</td>
</tr>
<tr>
<td><strong>Maturity</strong></td>
<td><a href="http://ft.ornl.gov/trac/blackcomb">http://ft.ornl.gov/trac/blackcomb</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As NVM improves, it is working its way toward the processor core

- **Newer technologies improve**
  - density,
  - power usage,
  - durability
  - r/w performance

- **In scalable systems, a variety of architectures exist**
  - NVM in the SAN
  - NVM nodes in system
  - NVM in each node

Diagram:
- Caches
- Main Memory
- I/O Device
- HDD
## Current ASCR Computing At a Glance

<table>
<thead>
<tr>
<th>System attributes</th>
<th>NERSC Now</th>
<th>OLCF Now</th>
<th>ALCF Now</th>
<th>NERSC Upgrade</th>
<th>OLCF Upgrade</th>
<th>ALCF Upgrades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned Installation</td>
<td>Edison</td>
<td>TITAN</td>
<td>MIRA</td>
<td>Cori 2016</td>
<td>Summit 2017-2018</td>
<td>Theta 2016</td>
</tr>
<tr>
<td>System peak (PF)</td>
<td>2.6</td>
<td>27</td>
<td>10</td>
<td>&gt; 30</td>
<td>150</td>
<td>&gt;8.5</td>
</tr>
<tr>
<td>Peak Power (MW)</td>
<td>2</td>
<td>9</td>
<td>4.8</td>
<td>&lt; 3.7</td>
<td>10</td>
<td>1.7</td>
</tr>
<tr>
<td>Total system memory</td>
<td>357 TB</td>
<td>710 TB</td>
<td>768 TB</td>
<td>~1 PB DDR4 + High Bandwidth Memory (HBM)+1.5PB persistent memory</td>
<td>&gt; 1.74 PB DDR4 + HBM + 2.8 PB persistent memory</td>
<td>&gt;480 TB DDR4 + High Bandwidth Memory (HBM)</td>
</tr>
<tr>
<td>Node performance (TF)</td>
<td>0.460</td>
<td>1.452</td>
<td>0.204</td>
<td>&gt; 3</td>
<td>&gt; 40</td>
<td>&gt; 3</td>
</tr>
<tr>
<td>Node processors</td>
<td>Intel Ivy Bridge</td>
<td>AMD Opteron</td>
<td>Nvidia Kepler</td>
<td>64-bit PowerPC A2</td>
<td>Intel Knights Landing many core CPUs Intel Haswell CPU in data partition</td>
<td>Multiple IBM Power9 CPUs &amp; multiple Nvidia Volts GPUS</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>5,600 nodes</td>
<td>18,688 nodes</td>
<td>49,152</td>
<td>9,300 nodes</td>
<td>1,900 nodes in data partition</td>
<td>~3,500 nodes</td>
</tr>
<tr>
<td>System Interconnect</td>
<td>Aries</td>
<td>Gemini</td>
<td>5D Torus</td>
<td>Aries</td>
<td>Dual Rail EDR-IB</td>
<td>Aries</td>
</tr>
<tr>
<td>File System</td>
<td>7.6 PB 168 GB/s, Lustre®</td>
<td>32 PB 1 TB/s, Lustre®</td>
<td>26 PB 300 GB/s GPFS™</td>
<td>28 PB 744 GB/s Lustre®</td>
<td>120 PB 1 TB/s GPFS™</td>
<td>10PB, 210 GB/s Lustre initial</td>
</tr>
</tbody>
</table>

**Complexity α T**
Opportunities for NVM in Emerging Systems

- Burst Buffers, C/R
- In-situ visualization

- In-mem tables


http://ft.ornl.gov/eavl
Programming NVM Systems
Design Goals for NVM Programming System

- **Active area of research**
  - See survey

- **Architectures will vary dramatically**
  - How should we design the node?
  - Portable across various NVM architectures

- **Performance for HPC scenarios**
  - Allow user or compiler/runtime/os to exploit NVM
  - Asymmetric R/W
  - Remote/Local

- **Security**

- **Assume lower power costs under normal usage**

- **Correctness and durability**
  - Enhanced ECC for NVM devices
  - A crash or erroneous program could corrupt the NVM data structures
  - Programming system needs to provide support for this model

- **ACID**
  - Atomicity: A transaction is “all or nothing”
  - Consistency: Takes data from one consistent state to another
  - Isolation: Concurrent transactions appear to be one after another
  - Durability: Changes to data will remain across system boots

---

MPI and OpenMP do not solve this problem.
NVL-C: Portable Programming for NVMM

- **Impact**
  - Minimal, familiar, programming interface:
    - Minimal C language extensions.
    - App can still use DRAM.
  - Pointer safety:
    - Persistence creates new categories of pointer bugs.
    - Best to enforce pointer safety constraints at compile time rather than run time.
  - Transactions:
    - Prevent corruption of persistent memory in case of application or system failure.
  - Language extensions enable:
    - Compile-time safety constraints.
    - NVM-related compiler analyses and optimizations.
  - LLVM-based:
    - Core of compiler can be reused for other front ends and languages.
    - Can take advantage of LLVM ecosystem.

```c
#include <nvl.h>
struct list {
    int value;
    nvl struct list *next;
};
void remove(int k) {
    nvl_heap_t *heap = nvl_open("foo.nvl");
    nvl struct list *a = nvl_get_root(heap, struct list);
    #pragma nvl atomic
    while (a->next != NULL) {
        if (a->next->value == k)
            a->next = a->next->next;
        else
            a = a->next;
    }
    nvl_close(heap);
}
```

Preliminary Results

- Applications extended with NVL-C
- Compiled with NVL-C
- Executed on Fusion ioScale
- Compared to DRAM
- Various levels of optimization

**LULESH**

**XSBENCH**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtMem or ExM</td>
<td>Use persistent storage as if extended DRAM</td>
</tr>
<tr>
<td>No Durability or ND</td>
<td>Skip runtime operations for durability</td>
</tr>
<tr>
<td>Base or B</td>
<td>Basic NVL-C version w/o Safety, RefCnt, and transaction (TX0, TX1, ...)</td>
</tr>
<tr>
<td>Safety or S</td>
<td>Automatic pointer-safety checking</td>
</tr>
<tr>
<td>RefCnt or R</td>
<td>Automatic reference counting</td>
</tr>
<tr>
<td>TX0</td>
<td>B+S+R + Enforce only durability of each NVM write</td>
</tr>
<tr>
<td>TX1</td>
<td>B+S+R + Enforce ACID properties of each transaction</td>
</tr>
<tr>
<td>TX2</td>
<td>TX1 + aggregated transaction using backup clauses</td>
</tr>
<tr>
<td>TX3</td>
<td>TX2 + skipping unnecessary backup using clobber clauses</td>
</tr>
<tr>
<td>TX4</td>
<td>TX3 at the granularity of each loop</td>
</tr>
<tr>
<td>CLFlush</td>
<td>Flush cache line to memory</td>
</tr>
<tr>
<td>MSync</td>
<td>Synchronize memory map with persistent storage</td>
</tr>
</tbody>
</table>
Summary

• Recent trends in extreme-scale HPC paint an ambiguous future
  – Contemporary systems provide evidence that power constraints are driving architectures to change rapidly
  – Multiple architectural dimensions are being (dramatically) redesigned: Processors, node design, memory systems, I/O
  – Complexity is our main challenge

• Applications and software systems are all reaching a state of crisis
  – Applications will not be functionally or performance portable across architectures
  – Programming and operating systems need major redesign to address these architectural changes
  – Procurements, acceptance testing, and operations of today’s new platforms depend on performance prediction and benchmarking.

• We need performance portable programming models now more than ever!
• Programming systems must provide performance portability (in addition to functional portability)!!
  – New memory hierarchies with NVM everywhere
  – Heterogeneous systems
Acknowledgements

• **Contributors and Sponsors**
  – US Department of Energy Office of Science
    • DOE Vancouver Project: https://ft.ornl.gov/trac/vancouver
    • DOE Blackcomb Project: https://ft.ornl.gov/trac/blackcomb
    • DOE ExMatEx Codesign Center: http://codesign.lanl.gov
    • DOE Cesar Codesign Center: http://cesar.mcs.anl.gov/
    • DOE Exascale Efforts: http://science.energy.gov/ascr/research/computer-science/
  – US National Science Foundation Keeneland Project: http://keeneland.gatech.edu
  – US DARPA
  – NVIDIA CUDA Center of Excellence
Bonus Material