X-Stack Demos Descriptions

1. DEGAS: Leveraging HipMer Extreme Scale Genome Assembler via NERSC Web Portal

De novo assemblers are a key computational method to reconstruct an unknown genome, but are limited by slow runtimes and limited scalability. To address this challenge, a multi-disciplinary team of computer and domains scientists collaborated over several years to develop HipMer, the first end-to-end HPC parallelization of the high-accuracy JGI Meraculous assembler. HipMer is designed to scale to massive concurrencies via numerous algorithmic advanced by leveraging Unified Parallel C (UPC). This allows a fast, scalable, and portable implementation executable on both shared- and distributed-memory architectures without modifications. In this demo we will present a forthcoming NERSC web portal interface that will allow the external bioinformatics and computational research community to remotely leverage our efficient and scalable de novo assembly capability.

2. DEGAS: Containment Domains Resilience

Containment Domains (CDs) enable an application to express resilience concerns, error strategies, and fault tolerance. Fundamentally, a CD-enabled application is expressed as a logical tree of nested CDs using simple and consistent programming abstractions. This demonstration highlights the advantages of both localized resilience actions and localized recovery. We show how our PGAS and MPI runtime prototypes enable an application to tolerate errors and failures of differing severity and scope using localized recovery. Local recovery, in contrast to coordinated global recovery schemes, enables some nodes to make forward progress while others are recovering, resulting in significant savings in compute time and energy.

3. D-TEC: Halide DSL for stencil computations

We demonstrate the performance portability, performance scalability, programmer productivity and versatility of domain specific languages. The first demonstration uses the popular Halide DSL for stencil computations. We show a live demo of an image processing application written in Halide and running on three different architectures (parallel shared memory system, GPU and a NERSC supercomputer). The codes running on these architectures are generated automatically from the same algorithm (the user needs only to provide a schedule to specifying how the algorithm should be mapped to each architecture). The second demo uses Simit: a new domain specific language for computing on graphs using linear algebra. It allows the development of many physical simulations. It is easy to use (shorter than Matlab), provides high performance (better than existing simulation libraries) and portable (CPU, GPU). We will demonstrate an example of a physical simulation written in Simit.

4. D-TEC: MSL synthesis of distributed memory implementations

The demo will show how our system can generate distributed implementations of kernels such as Lulesh, multigrid, and 3D transpose kernels. We will demonstrate the functionality of MPI+Sketch language (MSL) with kernels from mini apps and other benchmarks.

5. D-TEC: Using the ROSE framework for code generation and optimizations

We will demonstrate code generation for multiple platforms and optimizations using the ROSE framework. Those include code generation/transformation for OpenMP 4.x on the GPU platform, MPI code generation for distributed memory, code generation for the SST hardware simulator for multi-level memory system, and transformations with Polyhedral transformation.

6. D-TEC: Stencil computation using embedded DSLs in the X10 programming language

The X10 programming language is a simple, clean, but powerful and practical programming language for scale out computation using the asynchronous partitioned global address space (APGAS) model. It supports a rich array sublanguage as well as control structure overloading for defining embedded domain specific languages (eDSLs). We will demonstrate how control structure overloading can be used to implement efficient parallel iteration, including tiling patterns for stencil computation. We will also present results for the LULESH hydrodynamics proxy application, comparing between the reference OpenMP/C++/MPI implementation and our X10 implementation. The X10 code is 40% shorter, and it is also significantly faster when run on 1-1024 nodes of the Edison supercomputer.

7. D-TEC: A CAFe (DSL) transformed program running on Titan

A Fortran compiler-based infrastructure has been created that allows Domain Specific Languages (DSLs) to be relatively easily created, providing a platform for experimentation with language extensions that reduce the complexity of scientific application development at exascale. CAFe is a DSL both utilizing and extending Fortran coarrays to support heterogeneous computing. It was designed to provide a unified programming model across distributed memory nodes, each containing multiple accelerator devices. CAFe subdivides the Partitioned Global Address Space (PGAS) memory model of Fortran hierarchically, thus allowing a programmer to directly allocate and transfer memory as well as run and coordinate tasks on multiple devices. CAFe takes advantage of the existing parallel constructs of Fortran, such as synchronization events, atomic operations, and collectives. We demonstrate a CAFe transformed program solving a shortest path algorithm on ORNL's Titan.

8. D-TEC: Tolerating Latent Errors and “Silent” Errors with Flexible, Application-controlled Resilience with GVR

We demonstrate efficient scaling and low-overhead of the Global View Resilience (GVR) library with three scientific applications -- OpenMC (Monte Carlo methods), ddcMD (molecular dynamics), and Chombo heat equation codes. GVR enables flexible application control of overhead and recovery. For more information, see http://gvr.cs.uchicago.edu/

1. GVR for resilience at scale: Three GVR-enabled applications at large scale (~4k ranks), showing error injection and efficient recovery. Applications exploit flexible, multi-version state capture and naming for sophisticated recovery.
2. GVR extreme efficiency on Burst Buffers. Using the Cori BB, we build on prior demonstrations with BGAS to show GVR versioning can be extremely cheap and powerful for resilience. (will be shown if Cori BB is available)
3. GVR flexible silent and latent error recovery: GVR+ OpenMC implements forward error recovery, enabling efficient recovery from latent errors. OpenMC on 1k ranks and inject errors, detected with latency. We show performance of roll-back and forward recovery.

9. Traleika: The Intel Open Community Runtime, tools and applications

The Intel XStack team will demonstrate the Open Community Runtime, tools, and applications both ported and unmodified on top of this ecosystem. Using a mixture of applications and kernels, including HPCG, CoMD, and 2D Stencils, we will demonstrate running these codes on ***up to 1,000 nodes of NERSC*** machines Cori and/or Edison. Changing the execution behavior by the optional use of additional semantic information, such as via user-specified hints, will also be part of the demonstration to illustrate opportunities for increasing metric-of-interest behavior (communications, computation, energy, etc.). Additional demos will be provided for tools to analyze the execution of applications and the behavior of the runtime/execution layer itself.

10. XPRESS: HPX-5 integrated APEX

This demo shows the performance scalability of HPX-5 integrated with the Autonomic Performance Environment for Exascale (APEX). The demo shows the LULESH application running on NERSC Cori, using the Photon integrated communication library. APEX Introspection observes the application, runtime, OS and hardware to maintain the APEX state, while its Policy Engine enforces policy rules to adapt, constrain or otherwise modify the application behavior. This application shows APEX adapting the runtime to turn hyper-threading down. Photon supports a tight coupling of the runtime system with the underlying network fabric that scales and remains performant in exascale environments.

11. XPRESS: Visualizations of the dynamic nature of HPX-5

This demo shows visualizations of the dynamic adaptive nature of the runtime. Using the fast multipole method (FMM) application as an example, the first visualization shows the difference between remote communication activity before and after dynamic rebalancing effected by the active global address space (AGAS) in HPX-5. The second visualization shows the benefit of asynchronous behavior from the over-decomposition in HPX-5 for LULESH application.

12. **Entire XPRESS software stack on KNL pre-release hardware used to run LULESH**

This demo shows the operation of the XPRESS LXK OS infrastructure running the HPX-5 version of LULESH on KNL (Knights Landing) pre-release hardware. The LXK kernel will boot, detect and initialize all cores and memory, then launch HPX-5 LULESH in user level. HPX-5 is configured to initially run on all CPUs in the system, creating a single POSIX thread on each. As HPX-5 LULESH is running, the RCR driver in the LXK kernel will continuously monitor the power usage of the system and publish this information via the RCR blackboard (a page of memory) mapped into user-space. The APEX policy engine running in user-space will periodically read the power usage information from the blackboard and determine how many HPX-5 threads should be activated, performing dynamic concurrency throttling (DCT) to meet a desired power usage target. As the HPX-5 LULESH run progresses, the demo will show the number of HPX-5 threads to observed power usage.

13. **XPRESS: Interactive, distributed Mandelbrot Renderer**

Writing scalable OpenCL applications often takes more effort than the average user is willing to spend. The Mandelbrot code demonstrates that HPXCL overcomes this obstacle. HPXCL is a scalable OpenCL API for distributed systems, on top of HPX, a scalable C++ runtime system. HPX exposes GPUs in the global address space and allows for asynchronous execution of GPU kernels. The results produced by the kernel are encapsulated in a future. In this way, an application can off-load a section of code to an OpenCL device and continue doing useful work until the off-loaded data is ready. The distributed Mandelbrot renderer demonstrates. The values of the set are offloaded to the GPUs and the results are visualized on the host cores. The demo runs on 32 GPUs in nodes of a standard Beowulf cluster at LSU.

14. **PIPER: Measuring and Attributing Performance of Applications that Employ Emerging Template-based Parallel Programming Models**

Measuring and attributing performance information has become increasingly difficult with the move to node-level programming models that employ C++ templates to implement abstractions for dispatching parallel work. This demonstration will show how HPCToolkit measures node-level performance of applications employing the RAJA and Kokkos template-based abstractions for parallel programming, uses the emerging OpenMP Tools API (OMPT) to translate implementation-level measurements back to intuitive user-level views, and uses binary analysis capabilities of Dyninst components to accurately attribute the performance of highly-optimized executables back to the application source code.

15. **PIPER: Putting Data in Context with Caliper: A Composite Performance Data Collection Approach**

We demonstrate cross-stack performance data collection and analysis with Caliper. In increasingly modular HPC software stacks, it is important understand interactions between software components, and between software and hardware. Caliper is a universal abstraction layer that provides performance data collection as a service to applications, runtime systems, libraries and tools. We demonstrate a performance analysis case study with LULESH, where we link memory performance bottlenecks to specific MPI/OpenMP runtime states and application-level context (e.g., cycles and computational kernels).

16. PIPER: Supporting Active Harmony's Autotuning with Application-level Information

The execution of high-performance computing applications can often classified into phases with distinct behavior. These periods of time are relatively short and create problems for many tools. The Caliper project attempts to improve understanding of these regions via run-time annotations. One type of tool that can be confused by short phases is auto-tuners. In particular, optimal parameters for one phase may not be optimal for another. This problem can be solved by using Caliper's run-time phase annotations when combined with an online auto-tuner like Active Harmony. This allows an auto-tuner to reassemble the disjoint (and possibly interleaved) execution phases of an application, and optimize each as if they were whole uninterrupted tuning targets. We demonstrate our Caliper/Active Harmony integration effort on LLNL's LULESH proxy kernel. We also demonstrate the overall capabilities of the Active Harmony system.

17. SLEEC: Using SemChace with Kokkos for handling data movement challenges

We demonstrate the use of SemCache integrated with Kokkos. Kokkos is one of the foundational packages of Trilinos — it provides distributed matrix and array data structures and mathematical operations on those data structures. Kokkos hides the implementation details of these operations, and hence is intended to allow programmers to offload computations to accelerators such as GPUs. However, in such a setting, Kokkos still requires programmer intervention to manage data movement to and from accelerators. We demonstrate the use of SemCache to take over the challenge of data movement, providing the desired accelerator-oblivious interface to Kokkos.

18. X-TUNE: Autotuning Compiler Demonstration: Geometric Multigrid and Tensor Contraction

We present an approach that separates a high-level sequential C/C++/FORTRAN implementation from architecture-specific implementation (OpenMP, CUDA, etc.), optimization, and tuning. The autotuning compiler and search framework, in conjunction with expert programmers and other tools, will transform the baseline code into a collection of highly-optimized implementations. Then autotuning is used to explore this search space and derive final implementations that are best-suited for a specific execution context, thus mitigating the need for extensive manual tuning. In this demonstration, we show how this approach has been applied to two different application domains: Geometric Multigrid and Tensor Contraction.