Programming Environments for Exascale

The Department of Energy’s exascale software stack program (X-Stack) is exploring novel ideas for programming exascale machines in the 2023 time frame. Driven by power constraints and diminishing returns on traditional uniprocessor scaling the architectural landscape is undergoing a fairly radical transformation relative to the networks of single core and simple multicore systems of the past decade. The goal of the X-Stack program is to enable scientific discovery and the solution of critical mission problems on future exascale systems. This implies both performance and productivity goals, driven by the needs of the application community on the one hand and constrained by the available hardware approaches on the other. There is uncertainty on both sides, i.e., what future hardware will look like and what future applications and algorithms will be used and how they will be structured. This makes the design of an effective, high performance, portable programming model especially challenging. The X-Stack program is exploring a number of approaches in languages, compilers, runtime systems and tools to express, manage and automate various aspects of parallelism, locality, communication, scheduling, and variability. The purpose of this report is not to give a complete survey of the X-Stack activities, but to highlight some of the key ideas by looking at three projects: D-TEC, DEGAS, and Traleika Glacier.

In addition to summarizing the technical approaches, we will look at existing and planned engagement activities with the applications community, which is the next step in evaluating these ideas on real applications at scale for problems of interest to DOE. These include partnering with the Co-Design Centers and others in the applications community, developing implementations of proxy applications or larger problems, and collaborating on the design of language features and APIs. Each of the three projects (and others in the X-Stack program) has developed software that is, or can be used by the application community today. It is, however, important to emphasize that the current X-Stack program was designed to explore various ideas, and not to build hardened, tested and supported production-quality software. While impressive results have already been obtained on some application problems or simplified versions of them, and some components of these systems are used in production settings, one should not underestimate the investment required to deploy these ideas in a production environment.

1. Hardware Context for Exascale

The energy drivers for exascale are leading to architectural disruptions that will necessitate changes in the way software is written for the machines. There are also microarchitectural changes that, while they may not change the programming interface, could dramatically change the relative performance costs within the machine and therefore also require new algorithms and software models. The Abstract Machine Models (AMM) report summarizes the most likely changes foreseeable at exascale and points out that some of these may be ignored by software, while others are hidden from higher-level software, and still others may be exposed directly or through some type of abstraction that hides details of the feature to the application level software. The programming model similarly can ignore, hide, abstract, or expose things to the application programmer. New hardware features such a local scratchpad memory may be ignored and therefore unused by software; it may also be hidden by the programming model implementation (compiler and runtime system) by having data movement in and out of the scratchpad automated; it may be abstracted into higher level programming concepts that describe locality more generally, and aid the programming model implementation in managing
data movement; and finally, it can simply be provided as a feature of the programming model “as is” with data movement constructs.

Like today’s petascale systems, an exascale system will be built as a network of computing nodes, where most of the changes will happen within the node. The AMM report describes a canonical exascale node as shown in Figure 1

Figure 1: Abstract Node Architecture for Exascale

Anticipated features of the exascale node that will affect programming fall into the following categories:

1) **Lightweight cores**: A large number (hundreds to thousands) of simple, low power, processor cores that may either form a homogeneous manycore node or be mixed with a smaller number of more powerful (and less energy efficient) cores. Each node may also contain some number of more specialized accelerators, which may accelerate either computation or data movement operations. Both accelerators and general-purpose cores are likely to have some level of fine-grained data parallel support, e.g., SIMD or vector instructions. The node may either be a single integrated “system on a chip” which can facilitate higher bandwidth communication between cores or may be separate chips that communicated through a more limited interface. While the space of possible architectures is well understood, the uncertainty of the specific processor options is a major challenge for the design of exascale programming models and their implementations. Essentially, these programming models need to be retargetable to ensure good performance across all of the likely architectural targets.

2) **Software-managed scratchpad memory**: New technologies are emerging for high bandwidth, low capacity memory, either on chip or on as silicon substrate on the die. If these are treated as additional levels of cache (hardware management data movement in and out) then they can affect tuning of codes, but are conceptually the same abstraction that programmers deal with today, namely optimizations for spatial and temporal locality within a given memory size. Existing techniques tiling loops may be increasingly
important but should continue to be effective. More disruptive would be the introduction of software-managed memory hierarchy levels. These are already appearing in systems like the Intel Knights Landing processor to appear in major DOE systems within the next few years. Software-managed memory is one of those features that the programming model may ignore (the features can often be “turned off” in favor of caching), expose to the programmer, or attempt to automatically hide it through compiler or runtime support.

3) **Hierarchical communication:** Cores may communicate through a number of different mechanisms: cache-coherent shared memory; a symmetric global address space mechanism in which a core can read or write another core’s memory, but does not cache it; some type of Direct Memory Access (DMA) offload or message queuing protocol which is asymmetric, e.g., a CPU offloads to an accelerator but not the reverse. A number of studies have shown performance problems for cache-coherence at scale, which argues against it as a model across a full exascale node. At the same time, the convenience of shared memory seems to be superseding asymmetric “offload” models. A hierarchical model seems likely in which there are coherence domains on the chip and communication between those domains is done either with either a separate uncached data region or data that is cached only by the “owner” core.

4) **On-chip interconnect:** Integration of the Network Interconnect Controller (NIC) on the processor chip is likely in the exascale timeframe. This means that lower overhead communication should be supported although the specifics of how that works with the on-chip communication mechanism is less clear. Today’s petascale programming often suffers from performance problems that arise between the node and global programming systems, e.g., designating only one core for communication or the use of locking to protect access to shared message queues.

5) **Performance heterogeneity:** The AMM report describes a number of potential sources of performance heterogeneity, including algorithmic adaptivity, dynamic clock speed scaling, and resilience mechanisms that may hide actual faults but introduce significant performance variability. Programming models that rely on bulk-synchronous executions with static assignments of work to cores may be impractical on exascale systems.

2. **Application Context for Exascale**

The potential workload for exascale computers is large and diverse, ranging from materials modeling and climate change applications, to data analytics problems in genomics and cosmology. For the purpose of studying exascale programming models, applications may be roughly categorized based on their computational patterns. From the perspective of node-level code generation and optimization, and many of the on-chip memory hierarchy and parallelism issues, it is helpful to consider the computational “motifs” such as the original seven dwarfs, articulated by Phil Colella: dense matrices, sparse matrices, structured grids, unstructured grids, particle methods, spectral methods, Monte Carlo techniques. While these categories are neither exhaustive nor entirely separable, they help to describe the kinds of memory access patterns and fine-grained parallelism that a programming model should support. Alternatively, at a coarser level, issues of scheduling, load balancing and communication become important: 1) **static computations** are highly predictable, such that the volume of data, amount of work associated with each data item, and the dependencies and communication are all known in advance of program execution; 2) **dynamic computations** may have task dependence structures, task costs,
As we approach exascale, features like network congestion, dynamic voltage scaling, and component failures will lead to systems with highly unpredictable performance. This may make even the most regular computations appear irregular. The cost of simultaneously optimizing for communication and load balance remains inherently expensive. Dynamic approaches that ignore locality cannot be effective at scale and may not even be effective even within a node if the cost of data movement on chip is high enough. A set of open questions for exascale hardware is how significant the variability will be and at what scale it will happen. If individual cores on a chip will vary significantly even when running the same computation, then dynamic load balancing within a node may be important. However, if the variability is significant between nodes, dynamic load balancing may not be practical due to the growing cost of communication.

3. Three Programming Environment Approaches for Exascale

The DOE X-Stack program involves 12 research projects exploring a variety of ideas for how to express parallelism, translate it to code that is optimized for anticipated architectures, and manage the communication, computation and memory resources during program execution. This report is not intended to be a comprehensive summary of the X-Stack program, but instead highlights the approaches being used in three of the major projects, namely D-TEC (DSL Technology for Exascale Computing), DEGAS (Dynamic Exascale Global Address Space), and Traleika Glacier. The X-Stack program includes complementary efforts in compilers (X-TUNE and Vancouver, which also has higher level programming models work); runtime systems, (XPRESS and DynAX); correctness and semantic analysis tools (CORVETTE and SLEEC); performance tools (PIPER); and resilience support (GVR). The three projects discussed here are representatives of the overall program and each of the three have developed software that can be, and in some cases is used, by the applications community. The complete X-Stack program is described with more detail on all of the projects on the X-Stack website: https://xstackwiki.modelado.org/Extreme_Scale_Software_Stack. A future execution plan document would describe a more comprehensive picture of all the X-Stack projects and the flow of the research ideas and software into the production software stack.

3.1. D-TEC: Domain Specific Languages Technology

The D-TEC project is defining how to build Domain Specific Language (DSL) support within the X-Stack program and in collaboration with the DOE Co-Design centers. The value of DSL support is in the encapsulation of both application, algorithmic, and target machine specific details to define as simple a programming model as practical for the development of Exascale software. A specific focus of D-TEC work has been the high level support to stencil applications, which are
common to HPC and the Co-Design centers. DSL support has includes how to map applications to the unique aspects of expected Exascale architectures. Working with the SNL SST simulator we have demonstrated how to support the automated code generation for software managed caches and there use across multiple levels of memory hierarchy. The level of complexity of this generated code is well beyond where we expect users to want to write code by hand. This work complements other work in D-TEC showing how to generate GPU specific code from the same high level DSL implementation of target algorithms and applications. More than developing specific DSL the goal of D-TEC is to demonstrate how to develop DSLs more generally and to lower the barrier to their development and use.

The development of DSLs is a significant step in technology to how one might simplify writing future applications. In this context we have an incremental approach for applications to use D-TEC capabilities. New mechanisms permit the automated localized rewriting of software to use DSLs to automate the generation of alternative code. (e.g. generation of DSL code to support migration of existing code). Additionally, where the development of a DSL has typically paralleled the complexity of defining compilers for small languages, D-TEC makes it simple to build DSLs using general purpose languages such as C, C++, and FORTRAN as a base language and defining only high level abstractions and the custom code generation expected from the use of those abstractions mixed into existing programs. In many cases the high level abstractions can come from existing libraries; thus defining custom compiler support for such library abstractions. As a result D-TEC delivers custom analysis and transformation capabilities on the applications that DOE code groups develop and in the common languages with which they are already familiar.

Figure 2 shows the use of specific capabilities within D-TEC from the user’s perspective. The approach leverages custom compiler support to define code generation based on high-level abstractions to simplify the development of applications (this phase in architecture independent). This approach defines DSLs and the code generation from the DSLs to use either the Vendor compiler or LLVM directly. Code generation details are tailored to the target architecture for performance portability. Tool interfaces support an API that allow tools to report feedback to user in terms of the high level abstractions or DSLs. A migration approach provides high productivity in managing how legacy code can be fit into our D-TEC vision.
D-TEC is working with the different X-Stack teams to fit their work into the code generated from the DSLs. Where the DSL is appropriate is addressing more than the node level of the target machine we will generate code that is specific to the OCR. We anticipate that XPRESS will have an API layer implementing OCR so that we can develop transformations and code generation targeting a single runtime layer; alternatively we may target OCR and/or XPRESS’s XPI more directly.

Code transformations can target the complexity of future exascale architectures. Transformations can assist the multi-level memory simulation from three aspects: 1) preparing codes to be suitable for a multi-level memory, 2) enabling codes to trigger the APIs required in the simulator, and 3) other optimizations to increase performance. ROSE compiler provides strength in code transformation to assist users accomplishing these tasks as part of specific optimizations directly or in defining translations from DSLs. The development will first provide supports in the first two tasks. The third task will depend on details in the simulator design and require evaluation and discussion to finalize the task.

Preparation to multi-level memory: Loop tiling will be the key optimization that can transform a stencil code to be suitable to the multi-level memory. The tiled code is expected to perform parallel execution for multi-core, and a pipelined execution to hide the memory movement between different levels of memory. ROSE has an integrated polyhedral optimization package that can perform sophisticated loop transformations. The polyhedral tiling can tile three loops in a triple-nested loop in one transformation. This will be ideal for the tiling support in the...
multi-level memory simulation. There are ongoing developments in extending the tiling transformation to manage data in different levels of memory.

Preparation for simulator: With ROSE’s support, users can use attributes of variables or directives to designate an array or variable to a specific memory level. A transformation will loop up the information stored in the symbol tables and insert appropriate APIs and parameters to the APIs for the memory operations.

Other optimizations for multi-level memory system: Several transformations are optional but with potential to boost performance for codes running with a multi-level memory system. First, multi-buffered data can allow concurrent executions for data reading, computation and data writing. This optimization can work well if prefetching or concurrent read and write. Second, optimization for asynchronous memory moving could reduce the overhead in memory movement. In current API design, a wait has to be inserted to complete a memory movement. The wait can be inserted to a better location with assistance from program analysis. Third, an adaptive tiling that considers memory latency, bandwidth, and computation/communication ratio could deliver an optimal tile size for a given multi-level memory system.

Optimization of high-order stencils: High-order stencils arise in high-accuracy numerical solution approaches for PDEs in various domains. For example, Chombo and Overture applications can make use of high-order stencils, but current implementations result in a considerably lower rate of stencil applications per second with use of high order stencils. A new domain-specific optimization, described in a recent PLDI 2014 paper, has been developed that enables significantly enhanced performance for high-order stencils, on multi-core processors. A customized processor-specific code addressing effective vectorization and data locality is automatically synthesized from a high-level DSL description of the stencil computation, delivering a high level of programmability and performance. Other platform-specific automatic optimizing code synthesizers for stencil DSLs have also been developed for GPUs and FPGAs. These developments imply that more accurate solution schemes using high-order stencils can now be used, achieving virtually the same rate of execution in stencil applications per second as with lower order stencils of lower accuracy that are currently used in many applications.

D-TEC is also working on novel techniques based on software synthesis that will both help in mapping the high-level DSL code to low-level code. For example, in recent work published in SC 2014, we have shown that synthesis can help map from a high-level implementation of a kernel like SPMV to one that scales to tens of thousands of cores by automatically deriving the details of communication routines. The same technology can also help existing codes leverage the optimization opportunities afforded by DSLs by automatically mapping low-level imperative code to a DSL. In a recent set of experiments, for example, we show that we can automatically synthesize DSL implementations for most of the stencils in NAS MG and the Cloverleaf MiniApp. By doing this mapping, the DSL can apply aggressive optimizations that achieve up to 12x performance improvements over what a traditional compiler can achieve.

D-TEC is developing the OpenTuner autotuning system that uses advanced machine learning techniques. A source of major complexity and a large chunk of code in modern compilers focus on optimization selection. As the overall performance depends on the correct composition of wide range of optimizations, from the high-level domain-specific, mid-level data-flow type, and low-level scalable parallelization, partitioning for the multi-level memory hierarchy and
vectorization transforms making these local optimization decisions with a global impact is exceedingly difficult. Thus, a DSL developer adding a domain specific transformation needs to understand the full compiler flow and the supported architectures. OpenTuner will reduce the burden from the developers and will facilitate making the code more portable from current petascale systems to future exascale systems.

3.2. DEGAS: Dynamic Exascale Global Address Space

The Dynamic Exascale Global Address Space (DEGAS) project is addressing all aspects of the hardware changes, with a focus on locality control through hierarchical parallelism and data layouts, communication minimization through explicit software managed locality domains and provably optimal data movement within algorithms and compiler-generated code. A major theme in DEGAS is also to minimize the cost of communication when it happens, and to provide a data movement model that is scalable within the node, at the interface of the nodes to the interconnect, and across the scale of the system. The project is building on the idea of partitioned global address space (PGAS) languages, but with several new features to support new systems and applications. DEGAS views both performance heterogeneity and resilience as important challenges that need to be managed using an intelligent, tailored approach, so that a given application can select the desired level of adaptivity and resilience to match the application and system demands. To tackle the problem of unknown processor architectures (heterogeneous, SIMD, etc.) DEGAS is developing domain-specific code generation technology, which isolates hardware-specific optimizations in the code generator and performance autotuning (search) on a set of possible implementations.

The original goal of the DEGAS project was to develop and demonstrate language features that could be embedded in multiple languages (C, C++, Python and FORTRAN). As the project transitions into a mode of supporting software for the user community, we are directing most of our implementation work on UPC++. UPC++ is implemented using overloaded operators and templates, rather than requiring a full C++ compiler. It can be used as “global” language with OpenMP, CUDA, or other programming models in the node; in this way it is an alternative to MPI and also a good comparison point for MPI-3’s one sided support. UPC++ can also be used within the node to take advantage of the locality support that reflects the on node hierarchy. UPC++ allows for simple pointer dereference which translates into a DMA operation at the hardware level. Currently the SEJITS work is being done as a separate code generator, the results of which are included into a full application, although we are working on ways to incorporate the code generation step directly into an application build system.
Lightweight cores: The code generation technology in DEGAS is based on the SEJITS project, and the idea is similar to that of domain-specific languages, although the focus is on the intermediate format and code generation, rather than the language itself. The DEGAS project built a new version of SEJITS called CTree (now under development by the ASPIRE project), which uses LLVM for the backend code generation and has been demonstrated on some small numerical kernels. We are currently working on a version of HPGMG to demonstrate performance portability, i.e., the ability to generate high performance code from a single high-level description of the kernel computations within the benchmark. The focus at the current time is on stencil computations, because these are not easy to package in a library (there are too many) and are critical to so many DOE applications. We expect this to be done later this fiscal year, but not in time for the March meeting.

Software-managed scratchpad memory and hierarchical communication: Software managed memory spaces, whether they are part of a single core hierarchy or partitioned across cores, is an ideal fit to the PGAS model, which already has mechanisms to describe data layouts, separate allocation pools, and direct access across the memory regions. Because these features are still not common in existing hardware, demonstration of these ideas is still underway. We plan to work with the CAL team to use a simulator to demonstrate these ideas. The project is also exploring communication-avoiding algorithms, and in particular, compiler support for automatically generated communication-optimal code from high level descriptions. This work is basic research, with a high payoff if it succeeds, but not something on the critical path to a successful exascale programming system. Hierarchy of control and synchronization is a major theme within DEGAS, and one that has lead to more general notions of hierarchical teams of processors as well as very flexible and scalable synchronization primitives (e.g., phasers).
**On-chip interconnect:** The DEGAS team is working closely with the vendor community to understand how to best use proposed features of the interconnect, and also to give them feedback on the needs of scalable global address space support. One of the advantages of the unified PGAS model is allow all threads to communicate directly to other threads, whether on on or off and many current hybrid applications see a small fraction of available hardware bandwidth due to poor interchange between the different programming models.

**Performance heterogeneity:** To raise the level of abstraction in writing parallel applications, DEGAS is developing a set of distributed data structures, which are well supported by the type system in UPC++. There are two types of data structures: those use to hold/lookup state (e.g., a distributed hash table used for certain graph representations) and those used for scheduling (load balancing work queues, etc.). In this way, the various distributed data structures for DAG scheduling, randomize load balancing, and other locality-aware load balancing give application programmers the necessary abstractions for dealing with load imbalance in the algorithms and systems. The containment domain research is exploring in a similar spirit specialized support for resilience. The hierarchical approach in DEGAS also makes it possible to mix static and dynamic runtime approaches together as appropriate. For example, Habanero (dynamic DAG-based runtime) can be used within a node in a UPC++/Habanero program. Interoperability, both with pre-existing programming models and some others under development has been a top priority in DEGAS. The containment domain research is exploring in a similar spirit specialized support for resilience.

### 3.3. Traleika Glacier

The Traleika Glacier program develops X-Stack software components in close collaboration with application specialists at the DOE Co-Design centers and with the best available knowledge of the exascale systems we anticipate will be available in 2022 (see above “Hardware Context for Exascale”). We have built a straw man hardware platform that embodies potential technology solutions to well-understood challenges. This straw man is implemented in the form of a simulator that is being used as a tool to test software components under investigation by the Traleika Glacier team members. Co-design is achieved by developing representative application components that stress software components and platform technologies and then using these stress tests to refine platform and software elements iteratively to an optimum solution. All software and simulator components are being developed in open source, facilitating cross-team collaboration. The software environment includes Habanero-C - a portable programming system for heterogeneous processors, Concurrent Collections (CnC) - a template library for developing parallel applications, Hierarchically Tiled Arrays (HTA) - a natural extension of the array type and R-Stream - a source-to-source compiler for sequential C to parallel C. The interface between the software components and the simulator is being built to facilitate backend replacement with current production architectures (MIC and Xeon) providing a broadly available software development vehicle and facilitating the integration of new tools and compilers conceived and developed under this proposal with existing environments like MPI, OpenMP, and OpenCL.

While the Traleika Glacier program does not directly support application development, our researchers work closely with members of the three DOE Co-Design centers. Team members at each Co-Design center are responsible for producing codes that represent parts of applications that stress particular concept platform technologies. These codes are being used to develop well-founded guidance of their impact to future product plans. Our plan is to collaborate with runtime teams on DOE X-Stack and OSR programs including XPRESS, D-TEC, and DEGAS to
facilitate interchangeable programming environments. Further, these interchangeable programming environments will allow our Co-Design partners to quickly deploy new technologies into key DOE applications. But most important to the future of the market, the open tools developed under this proposal will reduce barriers to cross-software stack innovation and stimulate a period of rapid innovation for the high performance computing industry as a whole.

Figure 4: Traleika Glacier Software Stack

Traleika Glacier programming system research focuses on the challenges of extreme parallelism necessary for achieving efficient performance by establishing clear boundaries between the responsibilities of the domain specialist, algorithm specialist, tuning specialist and the runtime.

**Domain Specialist:** responsible for expressing the functions and data necessary to complete the simulation and the relationship between those functions and data.

**Algorithm Specialist:** responsible for breaking down the functions into tasks, and data into chunks, setting task priorities and execution patterns by using the basic task-oriented API as well as an application-centric tuning language.

**Tuning Specialist:** responsible for providing scheduling hints and priorities based on the system configuration and past runtime knowledge by using a tuning language.

**Runtime:** responsible for taking tuning language input and the hardware mapping hints and mapping tasks to nodes and data to memory, monitoring the progress of those tasks and re-scheduling them when failures or opportunities occur.

Through this separation of concerns neither the Domain nor the Algorithm Specialist need know anything about the underlying platform. Furthermore, the optimization hints that the algorithm specialist supplies are entirely at the application level and do not refer to a specific nodes. This approach provides portability and interoperability between the evolutionary, revolutionary and partner programming environments.
This provides a gentle slope programming model where a domain specialist can use existing functions developed with either evolutionary, revolutionary or partner tools and quickly build a simulation, albeit with poor performance. Over time and with use the algorithm specialist can increase parallelism with revolutionary tools and the tuning specialist improve performance and efficiency with a tuning language.

The program’s execution model and system software addresses the rapidly growing proportion of energy used for communication versus the energy used for computation. This imbalance increases the benefit of moving code to where data resides. To allow this to happen, the programming system will generate tiny tasks, called event driven tasks (EDTs), which are scheduled dynamically and asynchronously.

To achieve high efficiency and reliability, the execution model will manage node variability and power use across a very large machine. This heterogeneity will necessitate a level of “self-awareness” in scheduling and introspective multi-variable optimization within the system software. This research will be demonstrated and disseminated by developing a proof-of-concept Open Community Runtime (OCR) system.

We are developing modules that plug into the open source QEMU simulator and enable the straw man technology stress tests outlined above. This system level simulator will be available to all of our partners, and during the period of this program will be tuned and extended to meet the needs of the research team.

4. Application and Demonstrations of Technology

4.1. Demonstrations of D-TEC

In this strawman we plan to demonstrate the feasibility of using the DSL technology being invented in the D-TEC project for the future exascale systems by applying the existing technology to a single application domain. The application domain we have selected is stencil computation. With the existing technologies we can demonstrate how DSL technology can benefit realistic stencil applications. As the DSL technology matures, we will be able to expand such benefits to other application classes.

With the current DSL technologies, we are planning to demonstrate how the application community can benefit from DSLs using the following seven criteria.

1. Demonstrate ease of programming to hero level performance.
   a. We have released multiple DSLs specific to the specification of stencils. The Shift Calculus DSL is an implementation of novel approaches to how stencils for can be specified for AMR applications. The Maple DSL is a specification of the mathematical operator and from that the derivation of the lower level code using symbolic analysis and simplification using Maple.

   b. We plan to use the Halide DSL for stencil code generation to show that Halide programs written in a few hours, able to obtain performance up to 5× faster than hand-tuned C, intrinsics, and CUDA implementations written by experts over weeks or months.
2. **Demonstrate best of class performance on current hardware.**
   a. We have demonstrated the generation of code from Maple to support the 4th and 6th order stencil discretization of the wave equation for both Cartesian and curvilinear coordinate grids. Where this was compared to hand coded version it was 95% efficient on Cartesian grids, but could also be automatically generated for curvilinear coordinate grids where it was considered not tractable to write the 6th order stencil case by hand.
   b. We plan to use Halide DSL to show that for multiple image processing algorithms, the Halide generated code outperforms the best known hand-optimized code available.
   c. We plan to demonstrate the high performance achievable with high-order stencils using a new approach to associative reordering that reduces register pressure which otherwise severely limits performance of high-order stencil computations on multi-core processors.

3. **Demonstrate performance portability across multiple radically different existing platforms.**
   a. Within both the Shift Calculus and Maple DSLs code generation can be specified for conventional architectures, GPU architectures, and the Exascale simulator (SST) using software managed caches.
   b. We will show that the same Halide program without any modification can give best-of-class performance in both a NUMA CPU and a GPU environment.

4. **Demonstrate how to overcome exascale challenges for a few identified exascale challenges.**
   a. In recent work, the D-TEC project has shown how synthesis can help manage data distribution and communication for distributed memory implementations. In a recent paper in SC 2014 by Xu, Kamil and Solar-Lezama, for example, it was shown how synthesis can help bridge the gap between a simple sequential description of a task and a distributed implementation. The work used 3D transpose, MultiGrid and SPMV as examples, and showed how synthesis could be used to discover the details of data packing and communication routines. The code generated in this way was shown to be competitive with hand-crafted FORTRAN code even when scaled up to 16K MPI processes. The combination of synthesis, together with a set of high-level primitives to express bulk-synchronous computation made it easier to tackle the challenge of writing distributed code and have some confidence in its correctness.
   b. Using the HPC Challenge Benchmarks and 4 additional kernel benchmarks (K-Means, Smith-Waterman, Betweenness Centrality, and Unbalanced Tree Search), the X10 language and runtime system has been demonstrated to achieve scalable performance on a system consisting of 55,680 Power7 cores. We have also shown the scalability of the X10 implementation of UTS and Betweenness
Centrality on 16,384 cores of a BlueGene/Q system and 8,192 nodes of the K computer. This demonstrated the ability of the X10 system and its Asynchronous Partitioned Global Address Space programming model to achieve peak performance and scalability on these eight kernels that was comparable to the best implementations of the kernels using the traditional MPI-based programming model. As part of our X-Stack work in the D-TEC project, we have fully implemented three Co-Design center proxy applications in X10: Lulesh, MCCK, and CoMD. To date, we have demonstrated that the X10 versions of these three proxy applications achieve comparable performance and scalability results at relatively small scale: running up to 512 cores on a Power7 based system similar to the one used for the earlier large-scale X10 performance results. Our current work focuses on the evaluation of Lulesh on larger scale systems, including non-IBM systems at NERSC.

c. D-TEC work has also demonstrated how to efficiently manage exascale specific hardware features such as software managed caches by automating the generation of such code where it would otherwise be a huge burden to the application developer. The SST simulator has been central to the evaluation of these capabilities and to the evaluation of expected hardware designs, memory performance envelopes, and expected power constraints. We expect that this work will in the future help shape algorithm selection, design, and the placement of critical data.

5. **Demonstrate how to extend the system to handle non-trivial application evolution.**
a. Our work on stencil DSLs is being done in the context of AMR applications. Thus the target application of our work has both dynamic and static parallelism and the intense computational requirements to drive the research specific to defining automated transformations for proposed future exascale architectures. Subsets of the AMR target application permit focused work specific to leveraging hardware simulators to explore automated technique to express the complexity of code required to exploit software-managed caches within architectures with many levels of memory.

6. **Demonstrate a path from legacy applications to the new programming environment.**
a. We plan to show that stencil kernels in legacy software can be extracted using two systems. The Helium system uses unsound simple dynamic program analysis to identify stencil kernels and lift them to Halide. The Sting sound synthesis based techniques to extract loop invariants to identify stencil computations and then map them to Halide. These will be demonstrated using cloverleaf and miniGMG miniapps.

b. Additional work has automated the transformation of existing codes using MPI to define task-based models. In this work the MPI code is rewritten to support overlapped communication and computation and spread the computational work across many tasks connected via dependences. This work defines how D-TEC is
addressing the migration of legacy code to leverage novel task-based runtime systems being proposed for exascale architectures.

7. Demonstrate verification for correctness.
   a. Domain Specific Languages (DSLs) define simple high-level abstractions for users, but also introduce the requirement for correctness in their low-level generated code. Compilers have shown great promises in delivering high-performance for a variety of targets from a single input source (for example to map affine stencil computations for CPUs, GPUs and FPGAs), where each target requires its dedicated set of program transformations. D-TEC is developing the verification support to evaluate correctness of HPC transformations from DSLs to support Exascale architectures.

   b. On the one hand, deductive verification techniques based on Coq can be used to verify the rewrite rules used as part of the DSL to guarantee they can be safely used under any allowed context. On the other hand, D-TEC has also experimented with new exhaustive bounded checking techniques to help establish the correctness of manual or semi-automated transformations performed by the user as part of the lowering process from DSL to low-level code.

4.2. Demonstrations of DEGAS

The DEGAS project involves several component technologies that can be used together with existing programming systems or as a single unified stack. Interoperability has been a first class goal of the DEGAS project from its inception, because the team sees this a critical to adoption of the technology. Our application partnerships are intended to show how the DEGAS software can enable use of novel architecture and application features. At this point, most of the work is directed at UPC++ and related libraries and runtime features, although some applications use components of the software (GASNet-EX, SEJITS/CTree or Habanero), or older languages (UPC) with new data structures and runtime features that are transitioning into the DEGAS libraries.

4.2.1 Demonstration of productivity and scalability of large random access data structures and domain-specific runtime optimizations

We collaborated with Dan Rokhsar and Jarrod Chapman from JGI and Aydin Buluc (ASCR-funded on other projects) on the use of PGAS and to write a new version of the Meraculous genome assembler. It uses UPC and takes advantage of remote atomics and novel domain-specific runtime techniques being developed by DEGAS. The key data structure (which will be part of DEGAS UPC++ library) is a distributed hash table that uses remote atomics aggregation and software caching. Different phases of the computation require different runtime techniques, depending on the usage pattern for the hash table. The previous code relied on hardware shared memory for the hash table in one phase of the pipeline and was therefore non-scalable. The new code is 7000x faster (on more hardware and with a faster serial language as well as optimized I/O). This enabled one of the first de novo assemblies of the wheat genome.

4.2.2 Demonstration of data fusion and transition path for legacy software
We collaborated with Scott French and Barbara Romanowicz of UCB on a seismic simulation and analysis problem. This application, which does Full-Waveform Seismic Imaging, incorporated observational data into a simulation to improve both the quality and speed of the simulation. We see this scenario as an increasingly common one across DOE; it involves a “fusion” of data that is unaligned with the simulation data structures and is therefore quite expensive. Because the thread receiving data may have no information about its existence, it is also awkward in a two-side model. Using UPC++ with asynchronous remote task execution and one-sided communication, one can update distributed non-contiguous data structures efficiently. The code demonstrated 90% parallel efficiency on 12,000 cores and is 6x faster than using MPI’s new one-sided features (at least on the current Edison installation). With UPC++, the code can solve problems that were previously impossible due to memory size constraints on the shared memory implementation of the matrix assembly code, which enabled French and Romanowicz to extend their earlier upper-mantle tomographic imaging work (French et al., 2013, Science) and compute the first ever whole-mantle global tomographic model using numerical seismic wavefield computations (French & Romanowicz, 2014, GJI). The code also demonstrates the interoperability of UPC++ with legacy codes, as the rest of the application, which uses MPI, OpenMP and calls to the ScALAPACK library.

4.2.3 Demonstration of domain-specific schedulers for event-driven DAG-based scheduling in UPC++

We are working with Esmond Ng and Mathias Jacquelin (LBNL / FASTMATH) on an implementation of Sparse Cholesky Factorization using UPC++ asynchronous communication for an event-driven task execution model for better load balancing and latency hiding. The team has also implemented minimum degree ordering algorithms in UPC++, demonstrating the value of the global address space memory model, which enables users to parallelize important algorithms with dynamic irregular communication patterns that would be otherwise very difficult to be expressed in a message-passing memory model.

4.2.4 Demonstration of domain-specific code generators for performance portable code

We had developed a block structured grid implementation similar to the “Shift Calculus” described above based on the multidimensional array abstraction from the Titanium language (co-designed by Phil Colella). We have demonstrated this on the miniGMG benchmark and are working on an HPGMG implementation. The array library has first class notions of multidimensional indexes (Points) and index sets (Domains) over which there is a calculus of operations for intersecting, slicing, and translating. Arrays are built on domains and can be “viewed” through different index perspectives without copying data. These are useful for uniformed structured mesh codes (represented by miniGMG) and also for adaptive block structured codes (for which HPGMG is an initial proxy). We have done a quantitative study of three communication paradigms in miniGMG (bulk, fine-grained and multidimensional array abstraction) and demonstrated that one programming system (UPC++) can enable different programming styles with different performance and productivity trade-offs. The high-level multidimensional arrays abstraction in UPC++ enhances productivity while providing portable performance. In the ongoing UPC++ HPGMG work, we show that PGAS can enable applications to naturally express communication and thus enhances productivity significantly (reducing ~1000 lines of very challenging communication code in the case of HPGMG).
We have developed domain-specific runtime optimizations for the communication between arrays and are working on using SEJITS technology to generate optimized GPU, manycore and multicore code for the local stencil computations. (We developed the CTree implementation of SEJITS and have used it on smaller problems, such as sparse matrix-vector multiply to date.)

4.2.5 Demonstration of PGAS on a Chip

This work is relatively new due to the lack of hardware for testing, but we plan to work with the CAL project and various block-structured grid computation projects (ExaCT and FastMath) to explore the use of the PGAS model on a manycore chip with NUMA (non-uniform memory access) effects and to compare hardware designs, e.g., with and without cache-coherence across the chip. This is particular relevant to the incoming Cori and Trinity machines and our initial results using Babbage (a Xeon Phi as an accelerator) show promising scalability numbers, but without the NUMA effects so far.

4.2.6 Demonstration of communication-avoiding libraries and compilers

We have demonstrated some of the novel communication-avoiding ideas outside of the original domain of linear algebra, namely in n-body and most recently k-body (k at a time interaction) methods. We are working on an optimized SParse Matrix Dense Matrix Multiply (SPMDMM), which was a problem suggested by the ExMatEx team, but are currently looking at matrices from Buluc’s Graph algorithms project. If successful we plan to put this in a library. We continue to develop the theory of how to put these ideas into a compiler, and have demonstrated the distributed memory version (on dense matrix multiply) using a previously developed HPF compiler infrastructure.

4.2.7 Demonstration of communication scalability within and between nodes

A major issue with manycore systems arises from bottlenecks at the network interface. By using a single programing model (UPC++) rather than hybrid (e.g., MPI + OpenMP) we can use the fine-grained parallelism on the chip to speed up communication and avoid serial bottlenecks that often arise in practice. We are using the HPGMG benchmarks and a proxy for the Meraculous contig generation to demonstrate these ideas and plan to work with ExaCT, FastMath and NERSC and Intel (through the NESAP effort on Meraculous) to demonstrate these ideas.

We have also built a version of the Global Arrays abstraction in the NWChem application that uses DEGAS’s GASNet-EX infrastructure. Ongoing collaborations with Bert de Jong (LBNL, SciDAC) will look at evaluating the scalability of this on the current Babbage cluster at NERSC and future manycore systems as hardware becomes available.

4.2.8 Demonstration of DEGAS resilience model

To demonstrate the use of containment domains for resilience, we are working with the UPC++ version of Lulesh and (in collaboration with the TOORSES project) Strompack (HSS), which will also use BLCR. We have also used the UPC version of the NAS Parallel Bechmarks and LULESH for evaluation of Affinity-Aware BLCR and the NAMD code written in CHARM++. Current work is also looking at a Flexible GMRES algorithm, an iterative solver.
4.3. Demonstrations of Traleika Glacier

To fully explore the architectural changes exposed by the Traleika Glacier simulator key parts of DOE applications must be rewritten in new programming models. This can only be done with the full cooperation of the DOE application community and particularly the domain experts for each application.

The goal of these collaborations is two-fold:

- To learn from the experts and explore re-factoring DOE codes using asynchronous task models
- To build up a library of code and kernels that will be given back to the DOE using these new asynchronous programming models for bootstrapping the transformation of other programs

Frequent guidance from the experts of each application area is needed to guide how problems are re-built to the expected environment that future machines will provide. This will return direct benefits for system design teams and DOE experts, as they learn the re-factoring of the application without having to do the actual coding effort personally.

Concurrently, we will build evolutionary software support on top of the simulator and runtime infrastructure. While full benefits will only come through use of the new programming models and tools, we will endeavor to make sure that existing DOE applications identified by our partners in the DOE will run (after recompiling) on the strawman architecture.

This model of co-design and close interaction presupposes the availability of key DOE personnel and experts on a routine, monthly basis. It also presupposes that space and resources will be made available to Intel application engineers that travel to DOE sites.

During 2014-Q4, we worked with the DOE to specify the proxy applications we will use for program studies. We selected several proxy applications and are exploring one larger application. These workloads and their status are presented in a matrix format below. During our semi-annual applications workshop in April 2015, we will be concentrating upon two current implementations in this matrix, CoMD and HPGMG, as well as the selection of kernels.

<table>
<thead>
<tr>
<th>Notes</th>
<th>HPGMG</th>
<th>SNAP</th>
<th>miniAMR</th>
<th>XSbench/RSbench</th>
<th>Real app</th>
<th>Kernels</th>
<th>LULESH and SAR, UTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CnC</td>
<td>yes</td>
<td>In work</td>
<td></td>
<td>Tempest model</td>
<td></td>
<td>Cholesky, CG, FFT, Fibonacci</td>
<td>LL Well studied</td>
</tr>
<tr>
<td>Habanero-C</td>
<td></td>
<td></td>
<td></td>
<td>Several</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
In evaluating OCR with these kernels and the proxy application, we are interested in evaluating and understanding 1) the inherent scaling of OCR and 2) the improvements to be realized by exploiting the synergies between the OCR design and the strawman architecture.

OCR is designed to present a single common API to the application and be able to target multiple platforms. The two main platforms we are exploring are a traditional cluster based machine and the revolutionary architecture embodied by the strawman architecture. For the cluster based architecture, our implementation currently relies on services provided by a traditional Linux OS (threads, memory allocation, etc.) as well as asynchronous MPI calls to communicate between the nodes although this can be readily replaced by other communication protocols such as GasNET. For the strawman architecture, OCR directly manages the hardware through a very slim kernel and runs on FSim, the Traleika Glacier functional simulator.

This design of OCR will allow us to work on advanced policies for placement, energy management, and explore resiliency, etc. on both current platforms and the proposed TG architecture. Our implementation on the TG architecture, which we expect to scale to around 1000 nodes to properly explore configurations that implement several instantiations of the highest levels the memory hierarchy, will also allow us to study the performance and energy improvements that the TG architecture is expected to provide over current architectures.