FOX

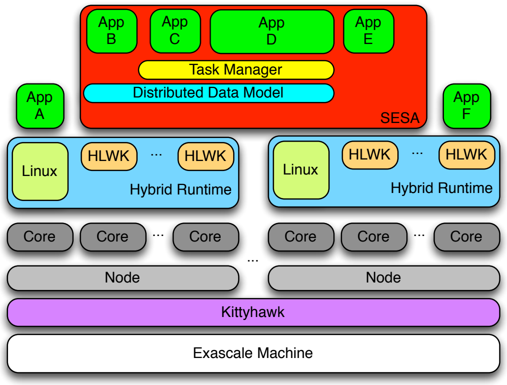
A Fault-Oblivious Extreme-Scale Execution Environment

Exascale computing systems will provide a thousand-fold increase in parallelism and a proportional increase in failure rate relative to today's machines. Systems software for exascale machines must provide the infrastructure to support existing applications while simultaneously enabling efficient execution of new programming models that naturally express dynamic, adaptive, irregular computation; coupled simulations; and massive data analysis in an unreliable hardware environment with billions of threads of execution. The FOX project is developing systems software and runtime support for a new approach to the data and work distribution for fault oblivious application execution. Our OS work includes adaptive, application tailored OS services optimized for multi → many core processors.

Operating Systems

Bell Labs and Sandia National Labs, in collaboration with Google and University of Rey Juan Carlos, created a new role-based many-core OS called NxM, which derives from Plan 9. NxM designates some cores to run full OS services while others run dedicated application processes, passing control to the OS core(s) to service system calls. NxM also provides convenient access to large pages. NxM researchers continue to explore the design space in more detail, examining alternatives for I/O, scheduling, hybrid system APIs and efficient inter-core communication choices.

FOX members from IBM Research have made improvements to the FusedOS version of Linux on BG/P to improve compatibility with mainline Linux. FusedOS combines Linux on a distinguished core with CNK-managed applications on the remaining cores of a Blue Gene node.



The Boston University team has developed a generalized model for developing future HPC applications that are hybrids of customized runtimes and general-purpose commodity operating systems. The model targets the development of libraries and applications that exploit distributed data structures and associated communication optimizations in the face of dynamic changes to the set of nodes. We have actively been defining this model and developing it in the context of constructing a prototype runtime for supporting hash-table software. We have developed a "bare metal" prototype of a key-value store that uses the hash-table. An outgrowth of this work has been a runtime model that uses HPC systems approaches along with distributed systems research techniques to yield an environment that combines commodity software stacks with reusable high performance software that reacts to dynamic changes. This runtime model was presented at the Exascale OS/Runtime Workshop.

Load Balancing

Dynamic load balancing is a promising technique to adapt to variations in the execution environment such as irregular computation, faults, system noise, and energy constraints. We have developed distributed memory load balancing algorithms based on work stealing and demonstrated scalability to hundreds of thousands of processor cores. Our techniques are shown to incur low overheads (space and time) to ensure fault tolerance, with the overheads decreasing with per-process work at scale. We demonstrated consistently high efficiencies on ALCF Intrepid, NERSC Hopper, and OLCF Titan for the Hartree-Fock and Tensor Contraction benchmarks on up to 140K processor cores.

Characterizing the behavior of work stealing schedulers and tracking task execution is complicated by the dynamic nature of mapping tasks to processors. We have developed an approach to efficiently trace async-finish parallel programs scheduled using work stealing with low time and space overheads. We demonstrated the broader applicability of this work, in addition to replay-based performance analysis, through two very different use cases: the optimization of correctness tools that detect data races in async-finish programs; the design of load balancing algorithms that exploit past load balance information to incremental adapt to changes.

Fault Tolerance

Checkpoint-restart approaches to fault tolerance typically roll back all the processes to the previous checkpoint in the event of a failure, a heavyweight solution that will not scale to exascale. We developed novel data-driven resilience algorithms for work stealing schedulers that minimize both the overhead in the absence of faults and the performance penalty incurred by a fault. We presented three recovery schemes that present distinct trade-offs — lazy recovery with potentially increased re-execution cost, immediate collective recovery with associated synchronization overheads, and non-collective recovery enabled by additional communication. We demonstrated that the overheads (space and time) of the fault tolerance mechanism are low, the costs incurred due to failures are small, and the overheads decrease with per-process work at scale

Tuple Space

Over the last few months, we have been developing a derivative of the Linda programming model which generalizes the key/value store to arbitrarily-type tuples and even wildcard matching. A flexible C++ template library allows dependencies between tuples and tasks to be easily expressed. Arbitrary event listeners can be attached to tuple operations, allowing tasks to immediately respond as new data becomes available. Although the framework most naturally suggests a Task-DAG model, the tuple space framework is designed to be as “expressive” as possible, allowing multiple parallel models to be expressed.

A major reason for choosing a Linda-like approach is a large body of work on fault-tolerant Linda from the 1990s. In particular, Linda can be extended naturally with a a set of fault-tolerant transactions for resilient computation. In the shorter-term, our resilience experiments are focusing on “slow nodes” rather than totally failed nodes. This should demonstrate clearly the flexibility of the Linda runtime that derives from processes being decoupled in space and time.

Applications

Computing elementary reaction rates for macroscopic flame models requires a detailed description of the reactant molecule's electronic structure. For the required accuracy, an extensive mathematical treatment must be given via the coupled-cluster formalism (most commonly denoted CCSD). The electronic structure is expressed as set of equations can be largely reduced to block-sparse matrix multiplications. The involved matrices, however, are both large and irregular, demanding tools for load balancing, fault tolerance, and data-driven task flow. Despite the CCSD model's complexity and steep scaling [$O(N^6)$ work with $O(N^4)$ storage in the number of electrons], our current implementations can routinely model molecules containing 100 electrons or more.