The efficient implementation of sparse linear solvers is crucial to enabling many large-scale simulations. The new generation of high performance computers with multi-/many core nodes and million-way parallelism presents new challenges for the linear solvers in the FASTMath software tools, PETSc, SuperLU, Trilinos and hypre. To better utilize the new hardware resources, we have focused our efforts on multi-threading key matrix kernels, hiding communication in Krylov solvers, the introduction of new scheduling strategies and lightweight OpenMP threads in sparse matrix factorizations, and the reduction of communication in algebraic multigrid methods.

Multi-threaded Matrix Kernels

Objectives
- Good sparse linear solver performance requires highly efficient sparse matrix kernels
- An abstract task based model for shared memory computation is developed, which is independent of the threading runtime.

Required Abstractions
- Dynamic dependencies, as dependences are not a priori known
- Recursive parallelism to adapt the implementations to the hierarchical architectures
- Composable constructs: need parallel for inside a task, and ability for tasks to create other tasks.

Generation of Tasks within a Kernel
- Using static analysis of the input graph for matrix.
- Static analysis possible using graph ordering methods.
- Currently using a custom version of Scotch graph partitioner to find tasks corresponding to UMA regions.
- Number of tasks equal the number of UMA regions.
- Number of leaves equals the number of threads.

Sparse Matrix Vector Multiplication – Performance Results
- Uses Intel TBB with automatic partitioner within the parallel_for of the UMA regions.
- The work within each UMA region is completely parallel.
- Requires more code than just using a parallel_for
- May lead to more overhead than using a parallel_for
- Task stealing is limited to threads within the UMA regions

Pipe-lined Krylov Solvers

Objectives
- The use of global all-to-all communication to orthogonalize and normalize Krylov vectors in each iteration of GMRES is becoming a bottleneck on modern HPC systems.
- The development of a non-blocking version of GMRES is crucial for future good performance.

Pipe-lined GMRES
- Changing the order of the operations within GMRES as described below can hide communication latency, system noise and load imbalance.

Conventional
- norm and projection start before applying operator, not needed until afterward
- result of projection used to correct $A_{i+1}$

Pipe-lined
- 2D Bratu problem solved with a Newton-Krylov method preconditioned by block Jacobi/ILU(0), GMRES restart parameter: 30.
- $\phi_0 = 0$.
- $\phi_{i+1} = A \phi_i + r$.
- Orthogonalize
- $\phi_0 = \phi_0 - A \phi_1$.
- $\phi_1 = \phi_1 - A \phi_2$.
- $A \phi_{i+1} = r_{i+1}$.
- Normalize
- $\phi_0 = \phi_0 - A \phi_1$.
- $\phi_1 = \phi_1 - A \phi_2$.
- $\phi_{i+1} = \| \phi_i \| / \| \phi_{i+1} \|$

Pipe-lined GMRES – Performance Results
- 2D Bratu problem solved with a fixation-Krylov method preconditioned by block Jacobi/ILU(0), GMRES restart parameter: 30.
- $\phi_0 = 0$.
- $\phi_{i+1} = A \phi_i + r$.
- Orthogonalize
- $\phi_0 = \phi_0 - A \phi_1$.
- $\phi_1 = \phi_1 - A \phi_2$.
- $\phi_{i+1} = \phi_{i+1} - A \phi_{i+2}$.
- Normalize
- $\phi_0 = \phi_0 - A \phi_1$.
- $\phi_1 = \phi_1 - A \phi_2$.
- $\phi_{i+1} = \| \phi_i \| / \| \phi_{i+1} \|$

Scheduling Strategies and Threading for Sparse Matrix Factorizations

Objectives
- Reduce overheads due to communication
- Develop scalable sparse direct linear solvers that are essential in the simulations of the numerically challenging problems, e.g., accelerator, fusion, quantum chemistry, and fluid mechanics.
- New performance challenges arise due to memory and bandwidth constraints on modern HPC systems with many nodes and many cores per node.

New Scheduling Strategies
- Implemented new static scheduling and flexible task-assignment algorithms that reduces the processors idle time and shortened the length of the critical path.
- The parallel factorization achieved nearly 3x speedup on thousands of cores.

Factorization time improvement of Version 3.5 over previous Version 1.5.
- Hopper at NERSC, Cray XE6.

Accelerator Omega3P, dimension 2.7 M
- Fusion M3D-C1, dimension 801 K

Redundant Coarse Grid Solve
- Parallel coarsening is stopped earlier on a finer level, all data of the system on level $k$ is distributed to all processes still containing grid points, and sequential AMG is used to solve the system on level $k$.
- Achieved up to 2x speedups of AMG using redundant coarse grid solves (table reports total speedup for AMG/CG for two problems on three different architectures with different fail-time networks).

Non-Galerkin AMG
- Choose non-Galerkin coarse grid for parallel efficiency by sparsifying $\Delta_{k+1} = \pi^T \Delta_{k}$ to yield a new coarse-grid operator $A_{k+1}$.
- Raises critical issues related to AMG theory:
  - Requires grid spectral equivalence: $\Delta_{k+1}/A_{k+1} = \pi^T A_{k+1}$
  - Algorithm heuristic: $\| x - A_{k}\|/\| x - A_{k+1}\| \leq \epsilon$
  - Provably implies AMG convergence

Communication Reducing Algebraic Multigrid Methods

Objectives
- Algebraic multigrid (AMG) methods have shown excellent weak scalability on distributed-memory architectures, however the increasing communication complexities on coarser levels have led to decreased performance on modern multicomputer architectures.
- The development of new methods with reduced communication is essential.

More Information: http://www.fastmath-scidac.org or contact Ulrike Yang, LLNL, yang11@llnl.gov