Tales from the Trenches: Whipping Chapel Performance into Shape

Ben Harshbarger, Elliot Ronaghan, Greg Titus
Chapel team, Cray Inc., Seattle WA, USA
{bharshbarg, eronagha, gbt}@cray.com

This talk will detail performance improvements made to Chapel in recent releases and will showcase several benchmarks that now perform as well as hand-coded C, OpenMP, and MPI reference versions. Historically, poor performance has been a stumbling block to Chapel's adoption, but we believe these performance advances demonstrate that Chapel is sufficiently fast for early adopters and show that a productive programing language like Chapel can still achieve high performance without sacrificing elegance or readability.

As a result of improving array access times, reducing task-spawning overheads, and improving NUMA affinity, Chapel's performance for the Livermore Compiler Analysis Loop Suite (LCALS) is now on par with the reference C and OpenMP implementations for most serial and parallel kernels (Figure 1). LCALS is a collection of loop kernels that emphasize floating-point operations, dense array manipulation, and other operations commonly found at the heart of many HPC applications.

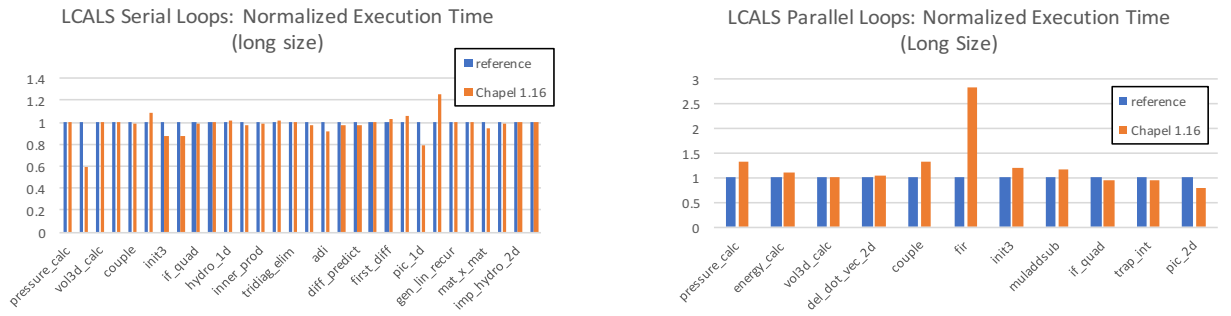

Figure 1: Execution Times for Chapel version 1.16 (in orange) on the LCALS serial and parallel loop kernels, normalized to the reference C+OpenMP times (in blue).

As another point of comparison for single-locale applications, Chapel's entries in the Computer Language Benchmarks Game tend to be as concise as the entries in scripting languages like Python, Ruby, and Javascript while resulting in performance that competes with or beats C, C++, and Fortran, as well as more modern languages like Go, Rust, and Swift (see Figure 2). For our entries, we strove to submit elegant and non-heroic implementations to demonstrate that productive and concise code can still perform well.

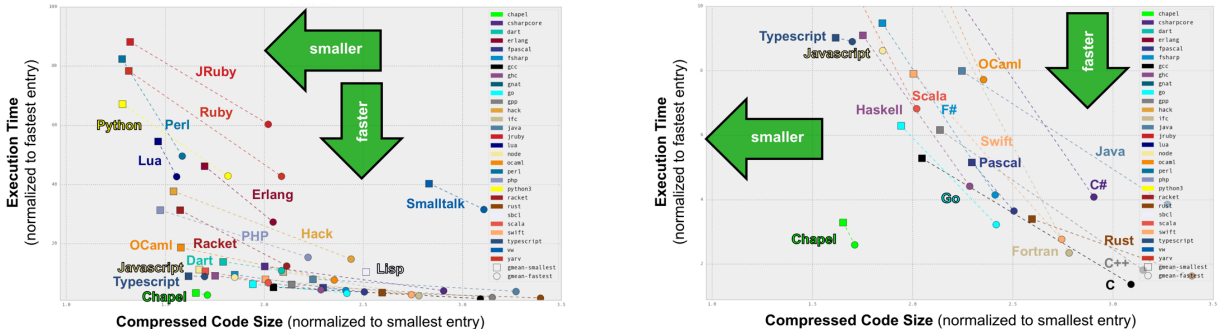

Figure 2: Scatter plots of the geometric means of the fastest and smallest Computer Language Benchmarks Game entries in dozens of languages, normalized to the fastest and smallest programs in any language. Being closer to the x-axis means that a language's programs are faster. Being closer to the y-axis suggests that they are more compact. Note Chapel's unique position down and to the left relative to other languages, indicating a unique combination of performance and conciseness.

For multi-locale applications, Chapel implementations of Stream Triad, Random Access (RA), the Intel Parallel Research Kernels (PRK) Stencil benchmark, and ISx (a bucket-exchange sort/histogram proxy application) perform on par with reference versions that are written in C+MPI or C+SHMEM (Figure 3).

Performance parity for these benchmarks was achieved through a variety of optimizations including remote task-spawning improvements, bulk-communication enhancements, dynamic memory registration, runtime optimizations, and array locality tuning.

The graphs in Figure 3 show results up to 256 nodes (36 cores per node) on a Cray XC. For the talk, we plan to gather results at even higher node counts on a Cray XC at NERSC (between 512 and 2056 nodes depending on machine availability.)

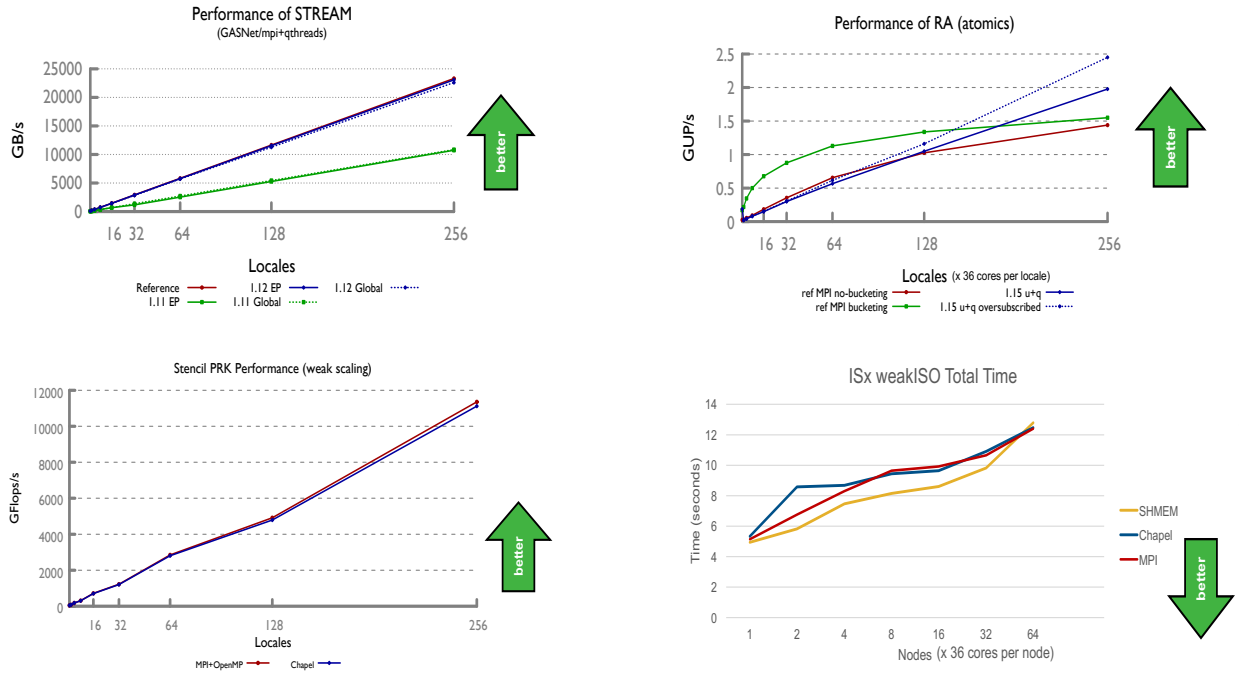

Figure 3: Performance scalability graphs for HPCC STREAM Triad, HPCC Random Access, PRK Stencil, and ISx as compared to C + MPI/SHMEM [+OpenMP].