Entering the Fray:
Chapel's Computer Language Benchmark Game Entry

Bradford L. Chamberlain* Ben Albrecht Lydia Duncan Ben Harshbarger,
Elliot Ronaghan Preston Sahabu Laura Delaney Michael Noakes
Cray Inc., Seattle WA
chapel_info@cray.com

The Computer Language Benchmark Game (CLBG)[1] is a website that hosts implementations of thirteen toy benchmark programs written in ~30 languages, where a language can provide multiple implementations of each benchmark. The site permits users to browse the programs' source code in order to qualitatively compare between languages and implementations. For each benchmark, the quantitative execution time, memory use, source code size, and CPU load is listed, and the implementations of a given benchmark can be sorted by these metrics. Languages can also be compared pair-wise across the suite of programs. The program metrics are measured on a quad-core 64-bit Ubuntu workstation via scripts provided and run by the site's maintainer.

Like most benchmark suites or contests, the conclusions that can be drawn from the CLBG are inherently limited. The front page of the website admits as much, stating: "We want easy answers, but easy answers are often incomplete or wrong." The programs are modest in size to lower the barrier to entry and acknowledged as such. Conclusions drawn from the implementations of a benchmark on the site may not extend to other implementations that *could* be written, while conclusions about the languages themselves based on the benchmarks may not extend to other programs written in the languages.

Despite these limitations, the CLBG generates a large amount of online discussion and certainly seems to influence the community's perception of languages. In this regard, it could be considered similar to the TOP500: it may not capture everything that actual users require in practice, but it serves as a straightforward and established way to make some comparisons. This is also arguably the CLBG's strength: rather than being paralyzed by the impossibility of creating a comparison that is 100% fair, accurate, or complete, it simply strives to do something approachable, tangible, and open (relatively speaking).

For several years, the Chapel team has been working on creating ports of the CLBG benchmarks as a background activity. Our interest in the CLBG may seem surprising given that Chapel's traditional focus has been on large-scale parallel computing while the CLBG benchmarks are often serial by nature, or at best shared-memory parallel. Yet despite our focus on large-scale systems, Chapel is also intended for productive programming on workstations. Given our interest in productivity, the CLBG serves as an interesting setting for comparing performance alongside code size and qualitative coding decisions.

In addition, success on the largest system scales depends on node-level issues: If local processors are not used efficiently, program performance and scalability will often suffer. Moreover, several of the key features exercised by the CLBG—such as lightweight synchronization, arbitrary precision arithmetic, file I/O, memory management, strings, and vectorization—are ones that Chapel users have expressed interest in. Finally, who doesn't enjoy a contest?

In early 2016, as Chapel's string support finally began to mature, we felt that we'd reached the point where we could assemble a CLBG entry that we'd be happy with. We inquired about the possibility of submitting an entry in late February and learned that the site would accept Chapel versions of the benchmarks in late April. From there, we strove to create polished versions of the benchmarks that would perform well without sacrificing too much in terms of elegance or clarity. Put another way, we tried to avoid creating heroic

---

* presenting author

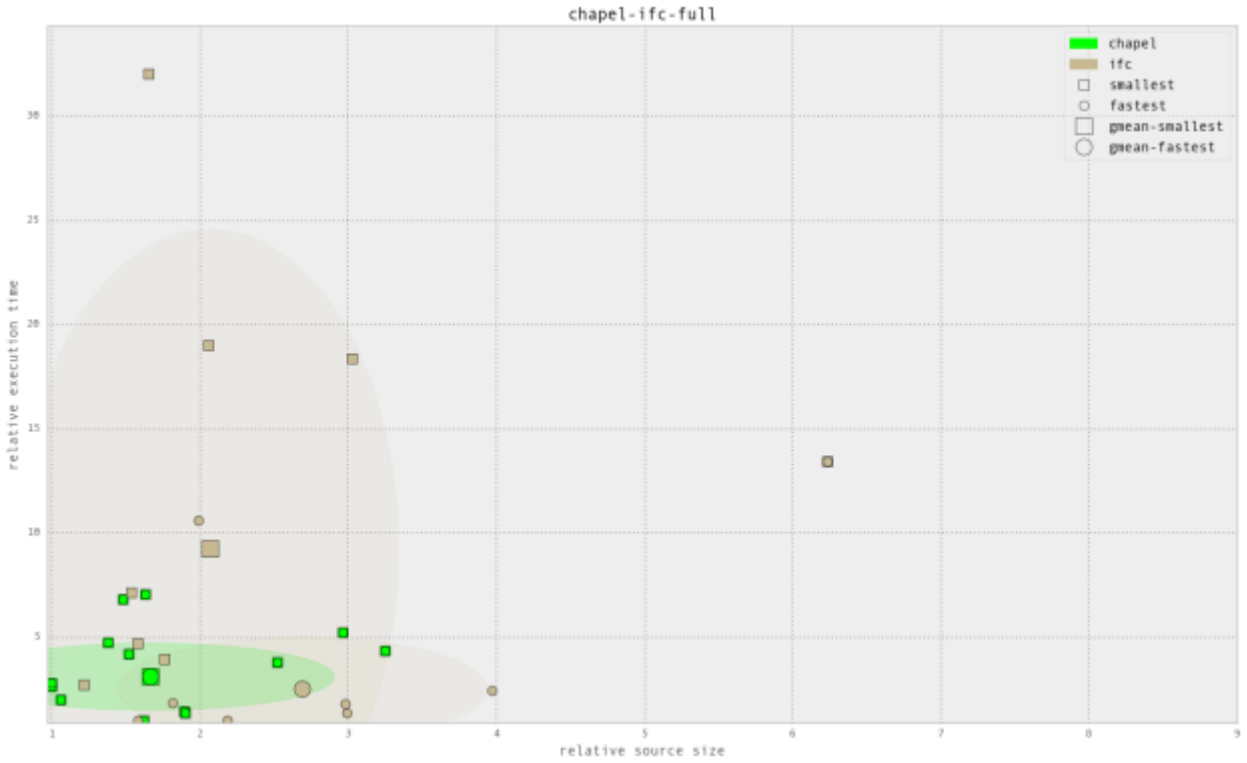
[1] Gouy, Isaac, http://benchmarksgame.alioth.debian.org/

versions of the benchmarks while striving to write code that others might consider useful versions to learn from as they implemented the benchmarks themselves. Our first program was accepted in late May and our final one in early September. At that point, Chapel was added to the front page of the CLBG site.

In this talk, we'll give a brief introduction to the Chapel versions of the CLBG benchmarks. In doing so, we'll highlight features of Chapel that benefit the entries in terms of performance and/or elegance. We'll also call out improvements that have been made since our original submissions, and aspects of the language or implementation that require attention to further improve our entries.

We'll also summarize how Chapel is doing compared to other languages, both according to the CLBG's metrics (where it's currently ranked as the 8th fastest language, just after Fortran) as well as via a series of scatter plots that we've developed to consider performance and code size simultaneously. As an example, the following plot compares our best Chapel programs (in green) with Fortran's fastest (tan circles) and most compact (tan squares). Each point is normalized to the fastest/most compact version of the benchmark in *any* language using the CLBG's measurements. The translucent ovals show one geometric standard deviation for the set of benchmarks, centered at the geometric mean. Thus, this graph shows that the Chapel versions are nearly as fast as the fastest Fortran codes, but tend to be about 30% smaller. Meanwhile, Fortran's most compact versions are about 20% larger than Chapel's while performing about 3x worse.

We'll conclude the talk by advocating for the creation of an HPC-oriented contest similar to the CLBG.



Acknowledgements: The authors would like to thank several former members of the Chapel team who contributed to the initial Chapel implementations of our CLBG entries: Casey Battaglino, Kyle Brady, Sung-Eun Choi, Hannah Hemmaplardh, Tom Hildebrandt, Jacob Nelson, Albert Sidelnik, and Thomas Van Doren. We'd also like to thank Isaac Gouy for hosting and maintaining the benchmarks game and for including Chapel.