Cross-Language Microbenchmark Harness (CLAMH) Introduction

Accurate compute performance estimates are very important for system and application deployment. Typical uses of performance analysis include establishing target requirements/SLAs such as throughput and response time, determining the necessary compute resources to meet those SLAs, evaluating design alternatives, and improving application performance. Because users rarely have the hardware and software configurations that will be used in deployment, the standard technique for evaluating performance is to run software benchmarks on hardware platforms that are similar to those that will be deployed.

The Cross-Language Microbenchmark Harness (CLAMH) provides a unique environment for running software benchmarks. It is unique in that it allows comparison across different platforms and across different languages. For example, it allows the comparison of clang, gcc, llvm, and GraalVM Sulong on the same benchmark, and can also be used to compare the Java counterparts of the same benchmark running on any JVM. CLAMH allows users to verify vendor benchmark performance claims, baseline benchmark performance in their own compute environment, compare with other compute environments, and, by so doing, identify areas where performance can be improved.

CLAMH provides a set of benchmarks along with the test harness, and users can also run their own benchmarks with CLAMH. The initially supported languages are Java and C/C++. JavaScript will be available soon with additional languages to follow.

CLAMH Technology

Although accurate benchmarking and performance measurement are important, they can pose significant difficulties for programmers. There are numerous pitfalls that can trap the unwary and lead to wildly inaccurate results; these are well known to benchmark experts, but often not appreciated in the greater software development community.

Further complicating matters is the need to compare the performance of the same benchmark behavior across different languages and/or different frameworks. Successful and accurate performance comparison depends on the elimination of as many confounding variables as possible. One source of confounding variables is the variation in benchmark harness design, capability, and especially quality for different languages (or different platforms).

CLAMH provides a common benchmark harness that allows benchmark implementations in different languages to run in a test environment that is as consistent as possible, and thereby enable a more “apples to apples” comparison.

In addition, the benchmark harness environment that CLAMH provides assists benchmark developers and users in avoiding many of the pitfalls associated with benchmarking that can result in inaccurate or unreliable results. It generates a test harness tailored to a given benchmark that eliminates as many of these sources of error as possible.

Many of the pitfalls associated with benchmarking are related to avoiding undesirable compiler optimization that can obscure or skew the performance results of interest. They include:

- **Loops** Timer granularity is often too coarse to accurately measure a single operation, so the operation is often repeated in a loop. However, an aggressive compiler can defeat this attempt to measure the performance by combining optimizations across loop iterations. CLAMH generates the loop code for you in such a way that undesirable loop optimizations are suppressed while avoiding the introduction of unnecessary overhead that could skew the results in the other direction.
- **Inlining** Indiscriminant inlining by the compiler can skew performance results by placing copies of the code for warmup and measurement in separate code blocks, such that warmup may not apply to the actual code that is being measured. The test harness generated by CLAMH controls inlining of the code and ensures that exactly the same code is run during warmup and measurement.
- **Dead code elimination** A common source of inaccuracy in benchmarks is produced when an optimizing compiler eliminates code branches that produce results that are never used. However, benchmarking code often represents an abstraction that is intended to perform certain "work" and the results are unimportant. CLAMH provides low-to-zero-overhead methods to consume these results and guarantee that the compiler will see them as "used".
- **Constant folding** A smart compiler can often replace complex operations with simpler operations when the inputs are hard coded. However, if that input is, in the benchmark, standing in for a value that could be runtime dependent, this can result in artificially optimistic performance results. CLAMH generates code that "tricks" the compiler into thinking that inputs might be runtime dependent, but in such a way that it does not force unnecessary extraneous memory accesses at runtime (which would skew the measurements the other way).