Benchmarking BOF

Kristof Beyls

This is a summary of what was discussed at the Performance Tracking and Benchmarking Infrastructure BoF session last week at the LLVM dev meeting.

At the same time it contains a proposal on a few next steps to improve the setup and use of buildbots to track performance changes in code generated by LLVM.

The buildbots currently are very valuable in detecting correctness regressions, and getting the community to quickly rectify those regressions. However, performance regressions are hardly noted and it seems as a community, we don't really keep track of those well.

The goal for the BoF was to try and find a number of actions that could take us closer to the point where as a community, we would at least notice some of the performance regressions and take action to fix the regressions. Given that this has been discussed already quite a few times at previous BoF sessions at multiple developer meetings, we thought we should aim for a small, incremental, but sure improvement over the current status. Ideally, we should initially aim for getting to the point where at least some of the performance regressions are detected and acted upon.

We already have a central database that stores benchmarking numbers, produced for 2 boards, see http://llvm.org/perf/db_default/v4/nts/recent_activity#machines. However, it seems no-one monitors the produced results, nor is it easy to derive from those numbers if a particular patch really introduced a significant regression.

At the BoF, we identified the following issues blocking us from being able to detect significant regressions more easily:

- A lot of the Execution Time and Compile Time results are very noisy, because the individual programs don't run long enough and don't take long enough to compile (e.g. less than 0.1 seconds).
- The proposed actions to improve the execution time is, for the programs under the Benchmarks sub-directories in the test-suite, to:
    - Increase the run time of the benchmark so it runs long enough to avoid noisy results. "Long enough" probably means roughly 10 seconds. We'd probably need a number of different settings, or a parameter that can be set per program, so

that the running time on individual boards can be tuned. E.g. on a faster board, more iterations would be run than on a slower board.
    - Evaluate if the main running time of the benchmark is caused by running code compiled or by something else, e.g. file IO. Programs dominated by file IO shouldn't be used to track performance changes over time. The proposal to resolve this is to create a way to run the test suite in 'benchmark mode', which includes only a subset of the test suite useful for benchmarking. Hal Finkel volunteered to start this work.
- The identified action to improve the compile time measurements is to just add up the compilation time for all benchmarks and measure that, instead of the compile times of the individual benchmarks. It seems this could be implemented by simply changing or adding a view in the web interface, showing the trend of the compilation time for all benchmarks over time, rather than trend graphs for individual programs.
- Furthermore, on each individual board, the noise introduced by the board itself should be minimized. Each board should have a maintainer, who ensures the board doesn't produce a significant level of noise. If the board starts producing a high level of noise, and the maintainer doesn't fix it quickly, the performance numbers coming from the board will be ignored. It's not clear what the best way would be to mark a board as being ignored. The suggestion was made that board maintainers could get a script to run before each benchmarking run, to check whether the board seems in a reasonable state, e.g. by checking the load on the board is near zero; "dd" executes as fast as expected; .... It's expected that the checks in the script might be somewhat dependent on the operating system the board runs.
- To reduce the noise levels further, it would be nice if the execution time of individual benchmarks could be averaged out over a number (e.g. 5) consecutive runs. That way, each individual benchmark run remains relatively fast, by having to run each program just once; while at the same time the averaging should reduce some of the insignificant noise in the individual runs.

I'd appreciate any feedback on the above proposals. We're also looking for more volunteers to implement the above improvements; so if you're interested in working on some of the above, please let us know.