Performance analysis and optimization of C++ standard libraries

Aditya Kumar[1] and Sebastian Pop[2]

[1]Samsung Austin R&D Center
[2]Samsung Austin R&D Center
{*aditya.k7, s.pop*}@samsung.com

Summary

C++ programs are widely used in performance critical applications. The standard libraries of C++, hence, are expected to be very efficient. However, experimental results show opportunities for improvements in some of the most commonly used data structures and algorithms.

We will present the performance analysis work on libc++ and libstdc++ and the changes we did to these libraries and to the LLVM compiler to optimize the code using them. This includes our contributions to standard library algorithms like string::find, libc++::basic_streambuf::xsgetn, and libc++::locale. We improved these suboptimal algorithms, particularly string::find which improved by more than 10x. Similarly, we enabled the inlining of constructor and destructor of libc++::string. We will present a systematic analysis of function attributes and the places where we added missing attributes. We will present a comparative analysis of libc++ vs. libstdc++ vs. Microsoft's C++ standard library on commonly used data structures and algorithms based on our std-benchmark (https://github.com/hiraditya/std-benchmark), that we started developing to help analyze standard C++ libraries. We will discuss the performance issues with libc++::stringstream and libc++::sort that we are currently working on. We will also present the lessons learned as a result of analyzing C++ standard libraries, for example:

1. Iterator based algorithms can lose information and hence, can result in suboptimal performance. This is exemplified in the implementation of std::rotate where we can just exchange few pointers should the underlying container is a doubly linked list e.g., std::list.

2. The C++ programming language has a limitation that the constructor and destructor cannot be const qualified which could have facilitated useful compiler optimizations like removing the destructor of a const std::string when the string is small enough to be kept on the stack.

Keywords: C++, performance analysis, benchmarking libraries, compiler optimization, LLVM, libstc++, libc++

Reference to previous talks: `http://sched.co/A8J7`, `http://sched.co/8Yzk`