Performance Analysis and Optimization of C++ Standard Libraries

Sebastian Pop, Aditya Kumar

SARC: Samsung Austin R&D Center

October 26, 2017

Agenda

- C++ standard template libraries
- software performance analysis
- improvements to libc++ and libstdc++ performance

C++ Standard Template Libraries

- STL is easy to use
  - standard interface: portable
  - easy to change data types: list, vector, deque, map, etc.
  - easy to change algorithms: iterators
- complexity specified by standard
- performance left to implementation

Performance of STL implementations

- performance and memory usage depend on
  - implementation: libc++ vs. libstdc++ vs. MSVC, etc.
  - container type
  - inefficiencies in implementations
- always analyze software performance to validate your choice

Figure: Size in bytes of empty containers on x86_64

Container	libstdc++	libc++	MSVC
vector<int>	24	24	24
list<int>	24	24	16
deque<int>	80	48	40
set<int>	48	24	16
unordered_set<int>	56	40	64
map<int, int>	48	24	16
unordered_map<int, int>	56	40	64

Software Performance Analysis

- identify hot functions from execution profiles
- inspect hot path: unit-benchmarking
- identify resource utilization on hot path

Profiling: identify hot path

- linux-perf: cycles, instructions, HW counters
- valgrind: cachegrind (R/W/Instrs), callgrind
- oprofile

Unit-benchmarking: inspect hot path

unit-benchmarking is unit-testing for performance

- set-up data structures in memory
- call hot function
- execute hot function until performance measures stabilize [1]

check performance of a single hot operation: less noise, keep focus

[1] https://github.com/google/benchmark

Analyze resource utilization on hot path

Inspect:

- source code, compiler IR, assembly code
- CPU usage, instructions used and their latencies
- memory bus and caches: loads/stores, spills, cache misses

Improve Software Performance

- eliminate unnecessary work
  - call functionality from libc or libc++
  - reduce bus traffic: vectorize loads and stores
  - help compiler remove redundancies: attributes and inline
- analyze performance of different implementations
  - change data structures
  - change algorithms
  - change STL implementations
- analyze trade-offs of caching previous results
  - use more memory vs. less computation (and vice versa)

Our contributions to improve performance of libc++ and libstdc++

- string to int value parsing: xsgetn in libc++
  - replace byte by byte copy with call to libc memcpy
  - important speedup on proprietary benchmark
- std::string.find() in libc++ and libstdc++
  - replace byte by byte compare with call to memchr and memcmp
  - 12x speedup on std-benchmark [2]
- inline ctor/dtor
  - shared_ptr
  - basic_string
- add attribute noreturn to non-returning functions
  - __locale, vector, deque, future, regex, system_error, etc.
  - important for compiler optimizations
  - remove false positives in static analysis tools

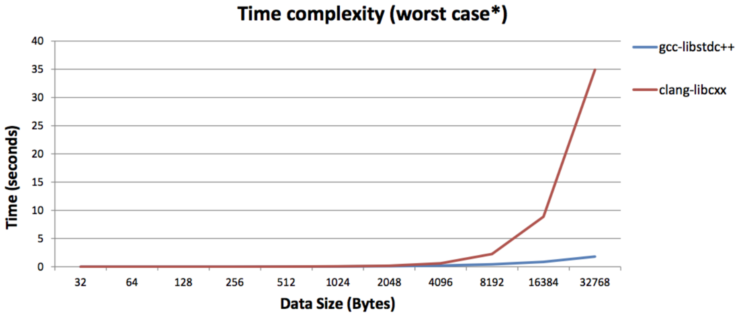
[2] https://github.com/hiraditya/std-benchmark

std::sort (libc++)

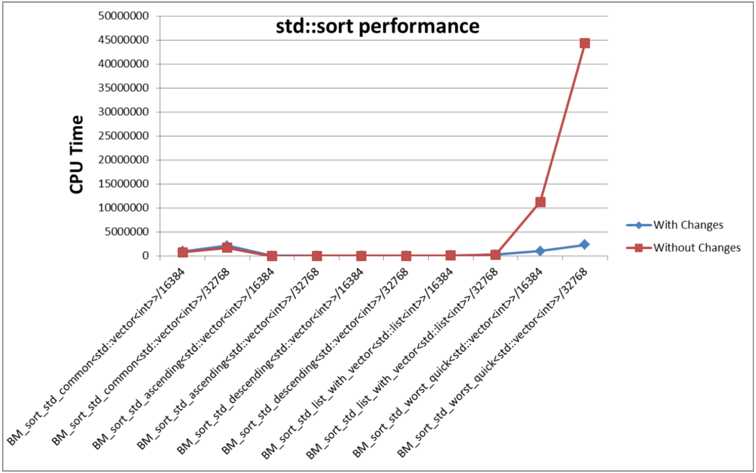
- Convert to introsort
- Sorting technique, which begins with quicksort and switches to heapsort after recursion reaches a threshold
- Worst case complexity of O(NlogN)
- Eliminate recursion
- Replaced memory intensive recursive calls with stack std::stack uses std::deque, which uses std::algorithm
- Improved worst case time complexity by a factor of 10 https://reviews.llvm.org/D36423
- quicksort with tail recursion elimination: quadratic worst case
- reimplemented as introsort: begin with quicksort, switch to heapsort when recursion depth goes beyond a threshold
- 16x speedup in the worst case (std-benchmark [3])

[3] https://github.com/hiraditya/std-benchmark

Issues with std::sort (libc++)



Sorting Results Plot (With std-benchmark)



Lessons learned (containers) optimizing destructor of string

```
#include<string>

int main() {
  std::string s("a");
  s+='a';
  return 0;
}
```

```
$ g++ -O3 t.cpp -S –fno-exceptions –std=c++11 -o - | grep _ZdlPv

$ clang++ -O3 t.cpp -S –fno-exceptions –std=c++11 -o - | grep _ZdlPv
        call    _ZdlPv
```

```
#include<string>
void foo();

int main() {
  const std::string s("a");
  foo();
  return 0;
}
```

```
$ g++ -O3 t.cpp -S –fno-exceptions –std=c++11 -o - | grep _ZdlPv
        call    _ZdlPv

$ clang++ -O3 t.cpp -S –fno-exceptions –std=c++11 -o - | grep _ZdlPv
```

Annex

Performance Analysis with Valgrind

- `valgrind [--tool=memcheck]`
- `valgrind --tool=cachegrind`
  - cache and branch simulator
  - count read, write, and branch instructions
- `valgrind --tool=callgrind`
  - execution call graph
  - visualization tool kcachegrind
- `valgrind --tool=massif`
  - how much heap and stack memory your program uses
  - which parts of program allocate the heap memory

Valgrind: Example – SQLite

```
$ valgrind --tool=cachegrind ./sqlite_llvm <test.sql >/dev/null
[...]
--------------------------------------------------------------------------------
           Ir       I1mr   ILmr           Dr       D1mr     DLmr           Dw       D1mw     DLmw
--------------------------------------------------------------------------------
1,278,771,731 29,231,219 35,783 359,414,267 6,707,514 528,920 197,515,528 2,594,262 171,968  PROGRAM TOTALS

--------------------------------------------------------------------------------
         Ir       I1mr  ILmr          Dr      D1mr    DLmr          Dw    D1mw   DLmw  file:function
--------------------------------------------------------------------------------
363,052,233 7,560,087 3,122 97,707,865 1,084,529  77,197 44,505,055 217,826 29,838  src/sqlite3.c:sqlite3VdbeExec
 95,048,357    80,721   111 33,248,107    59,086   7,273 20,173,275      91      7  src/sqlite3.c:vdbeRecordCompareWithSkip
 68,045,026   695,509 1,144 14,883,933   114,698   1,918  5,525,733 272,507 19,249  src/sqlite3.c:balance
 56,713,554 1,101,002   276 18,416,705   683,914  21,085  3,453,665   1,947     25  src/sqlite3.c:sqlite3BtreeMovetoUnpacked
 45,344,891    59,660    66 13,589,490    66,121  18,775 12,795,281  59,451     86  src/sqlite3.c:sqlite3VdbeRecordUnpack
 36,550,248    47,192    94  9,615,816   217,845  11,567          0       0      0  src/sqlite3.c:cellSizePtr
 35,156,491 1,031,905   859  7,810,853   489,509   1,936  6,546,085 175,469 26,159  /build/glibc-2.19/malloc/malloc.c:_int_malloc
 34,402,967   219,015    40 12,316,213    31,625   1,007          0       0      0  src/sqlite3.c:vdbeRecordCompareInt
 31,287,698   269,233   121 10,094,976   398,015  57,982 10,094,976 797,005 41,768  /build/glibc-2.19/string/../ports/sysdeps/aarch64/memcpy.S:mem
 30,895,222 1,055,479   718  3,990,072    45,246     157  3,247,672   1,200     58  src/sqlite3.c:sqlite3VXPrintf
 29,633,734        87    87  6,992,348   510,654 147,437  1,945,350     292     14  src/sqlite3.c:vdbeSorterSort
 28,301,654 1,222,726   236  7,685,792   129,350     101  4,693,862  15,480     91  src/sqlite3.c:sqlite3BtreeInsert
 27,452,670   605,975   428  7,719,336   275,711   3,045  6,130,240   1,247    180  /build/glibc-2.19/malloc/malloc.c:_int_free
 26,152,338    93,230    53  5,107,641    26,455      59  3,502,857   6,705      2  src/sqlite3.c:sqlite3VdbeSerialGet
 21,638,172   664,339   241  7,621,765   197,153   7,033  5,509,634  12,988     53  src/sqlite3.c:sqlite3PagerAcquire
 19,904,842   811,018   134  6,875,142    93,695     809  4,223,778   6,655     72  src/sqlite3.c:insertCell
 17,184,877   622,046   254  5,927,277   207,045     101  3,228,818  13,564     78  src/sqlite3.c:pager_write
 16,511,495   127,072    29  5,189,327     7,164   1,105  2,358,785       0      0  src/sqlite3.c:serialGet
 14,566,464   347,254   101  5,076,192    68,798   4,135  3,972,672 131,226  9,179  src/sqlite3.c:moveToChild
 14,089,915   528,334   433  3,522,612   169,118     295      1,089      24     22  ???:???
 13,516,049   315,369    75  3,660,565    70,941     104  2,252,728   2,740     20  /build/glibc-2.19/malloc/malloc.c:malloc
 13,444,711   370,614    60  3,136,255    74,755  57,116  3,757,149       0      0  src/sqlite3.c:btreeParseCellPtr
 11,814,468   620,489   364  3,444,231   109,318     159  1,401,768  11,253      7  src/sqlite3.c:sqlite3VdbeHalt
  9,867,819   655,851   130  3,350,976    68,237      46  1,820,276  62,050     70  src/sqlite3.c:moveToRoot
  9,023,249   615,625   175  2,774,458    27,649      72  1,719,012     578      1  src/sqlite3.c:sqlite3VdbeMemGrow
  9,015,155   136,420   114  2,528,161    33,460      40  1,808,361      12      7  /build/glibc-2.19/nptl/pthread_mutex_lock.c:pthread_mutex_lock
  8,932,696   193,491    71  1,956,326    55,921      22  1,411,634       2      0  /build/glibc-2.19/nptl/pthread_mutex_unlock.c:__pthread_mutex_
  8,916,165    82,925    47  2,092,310         0       0  1,933,573   1,583      3  src/sqlite3.c:memjrnlWrite
  8,869,488   284,528    72  4,276,902   299,688   8,315  1,834,026   6,712     17  src/sqlite3.c:pcache1Fetch
  8,120,421   171,173   145          0         0       0  4,459,287 446,962 23,788  /build/glibc-2.19/string/../ports/sysdeps/aarch64/memset.S:mem
  7,759,659   338,888    58  2,364,882    24,321   1,308  1,624,112 104,416  1,631  src/sqlite3.c:sqlite3PcacheRelease
  6,799,934    97,805   282  2,068,211    38,793     684  1,555,697   3,672     11  src/sqlite3.c:sqlite3BtreeNext
  6,674,044    88,515   123  1,706,065     4,244      43  1,094,451       7      0  src/sqlite3.c:freeSpace
  6,536,765   760,083   320  2,119,849   121,314      85  1,091,200       0      0  src/sqlite3.c:sqlite3_step
```

Performance Analysis with Linux Perf

Two modes of operation:

- sum up all counters: `perf stat`
- record events: `perf record`

Linux Perf: Example – SQLite

```
$ perf stat ./sqlite_llvm <test.sql >/dev/null

Performance counter stats for './sqlite_llvm':

      1045.856070      task-clock (msec)         #    1.000 CPUs utilized
                1      context-switches          #    0.001 K/sec
                0      cpu-migrations            #    0.000 K/sec
              809      page-faults               #    0.774 K/sec
    1,636,720,010      cycles                    #    1.565 GHz                    [83.16%]
      548,530,227      stalled-cycles-frontend   #   33.51% frontend cycles idle    [83.16%]
      218,991,051      stalled-cycles-backend    #   13.38% backend  cycles idle    [67.04%]
    3,385,841,295      instructions              #    2.07  insns per cycle
                                                 #    0.16  stalled cycles per insn [83.54%]
      709,436,490      branches                  #  678.331 M/sec                  [83.54%]
        2,586,354      branch-misses             #    0.36% of all branches        [83.17%]

      1.045918998 seconds time elapsed
```

Linux Perf: Example – 483.xalancbmk

```
$ perf record ./xalancbmk
$ perf report
  0.20 629a84:   ldr    w9, [x0,#24]
 18.71 629a88:   ldr    w8, [x1,#24]
 12.93 629a8c:   cmp    w9, w8
  2.74 629a90:   b.ne   629af8 <xalanc_1_8::XalanDOMString::equals
  2.00 629a94:   ldp    x8, x10, [x0]
  2.43 629a98:   cmp    x8, x10
  1.80 629a9c:   ldp    x10, x12, [x1]
  1.03 629aa0:   adrp   x11, 704000 <vtable for xalanc_1_8::ReusableArenaBlock+0x8>
  0.53 629aa4:   add    x11, x11, #0xb08
  0.03 629aa8:   csel   x8, x11, x8, eq
  1.33 629aac:   cmp    x10, x12
  0.34 629ab0:   csel   x10, x11, x10, eq
  1.78 629ab4:   cbz    w9, 629b00 <xalanc_1_8::XalanDOMString::equals
  0.02 629ab8:   ldrh   w11, [x8]
  4.02 629abc:   ldrh   w12, [x10]
  3.75 629ac0:   cmp    w11, w12
  1.03 629ac4:   b.ne   629b08 <xalanc_1_8::XalanDOMString::equals
  1.16 629ac8:   lsl    x9, x9, #1
       629acc:   add    x8, x8, #0x2
       629ad0:   add    x10, x10, #0x2
       629ad4:   sub    x9, x9, #0x2
 10.18 629ad8:   cbz    x9, 629b00 <xalanc_1_8::XalanDOMString::equals
  0.01 629adc:   ldrh   w11, [x8],#2
 18.79 629ae0:   sub    x9, x9, #0x2
  0.00 629ae4:   ldrh   w12, [x10],#2
  9.22 629ae8:   cmp    w11, w12
  5.11 629aec:   b.eq   629ad8 <xalanc_1_8::XalanDOMString::equals
       629af0:   mov    w0, wzr
       629af4:   ret
  0.69 629af8:   mov    w0, wzr
  0.09 629afc:   ret
       629b00:   orr    w0, wzr, #0x1
  0.10 629b04:   ret
       629b08:   mov    w0, wzr
       629b0c:   ret
```