

Performance analysis and optimization of C++ standard libraries

Aditya Kumar

Sebastian Pop

Samsung Austin R&D Center

Suboptimal basic_streambuf::xsgetn (libc++)

```
template <class _CharT, class _Traits>
streamsize
basic_streambuf<_CharT, _Traits>::xsgetn(char_type* __s, streamsize __n)
{
    const int_type __eof = traits_type::eof();
    int_type __c;
    streamsize __i = 0;
    for (; __i < __n; ++__i, ++__s) {
        if (__ninp_ < __einp_)
            *__s = *__ninp_++;
        else if ((__c = uflow()) != __eof)
            *__s = traits_type::to_char_type(__c);
        else
            break;
    }
    return __i;
}
```

Optimized basic_streambuf::xsgetn

```
template <class _CharT, class _Traits>
streamsize
basic_streambuf<_CharT, _Traits>::xsgetn(char_type* __s, streamsize __n) {
    const int_type __eof = traits_type::eof();
    int_type __c;
    streamsize __i = 0;
    while(__i < __n) {
        if (__ninp_ < __einp_) {
            const streamsize __len = _VSTD::min(__einp_ - __ninp_, __n - __i);
            traits_type::copy(__s, __ninp_, __len);
            __s += __len;
            __i += __len;
            this->gbump(__len);
        }
        else if ((__c = uflow()) != __eof) {
            *__s = traits_type::to_char_type(__c);
            ++__s;
            ++__i;
        }
        else
            break;
    }
    return __i;
}
```

```
template <class _CharT, class _Traits>
streamsize
basic_streambuf<_CharT, _Traits>::xsgetn(char_type* __s,
streamsize __n)
{
    const int_type __eof = traits_type::eof();
    int_type __c;
    streamsize __i = 0;
    for (; __i < __n; ++__i, ++__s) {
        if (__ninp_ < __einp_)
            *__s = *__ninp_++;
        else if ((__c = uflow()) != __eof)
            *__s = traits_type::to_char_type(__c);
        else
            break;
    }
    return __i;
}
```

Performance improvements

valgrind profile of a synthetic test case which only exercises xsgetn.

	Base compiler without patch	Base compiler with patch
Total no of instructions (valgrind)	1,378,842	1,359,235
basic_streambuf::xsgetn (char*, long)	20,015	0

```
struct test : public std::basic_streambuf<char> {  
    typedef std::basic_streambuf<char> base;  
    test() {}  
    void setg(char* gbeg, char* gnext, char* gend) {  
        base::setg(gbeg, gnext, gend);  
    }  
};
```

```
int foo(char* input, char *output, int N) {  
    test t;  
    t.setg(input, input, input+N);  
    char* pos = output;  
    pos += t.sgetn(pos, N);  
    return *pos;  
}
```

Suboptimal string::find algorithm (uses std::find)

b1, e1 iterators to the haystack string

b2, e2 iterators to the needle string

__search(b1, e1, b2, e2) {

...

while (true)

{

while (true)

{

if (__first1 == __s)

return make_pair(__last1, __last1);

if (__pred(*__first1, *__first2))

break;

++__first1;

}



Find the first matching character

_RandomAccessIterator1 __m1 = __first1;

_RandomAccessIterator2 __m2 = __first2;

while (true)

{

if (++__m2 == __last2)

return make_pair(__first1, __first1 + __len2);

++__m1;

if (!__pred(*__m1, *__m2))

{

++__first1;

break;

}

}

}



Match rest of the string

}

...

}


Optimized string::find algorithm

```
inline _LIBCPP_CONSTEXPR_AFTER_CXX11 const _CharT *
__search_substring(const _CharT *__first1, const _CharT *__last1, const _CharT *__first2, const _CharT *__last2) {
...
    // First element of __first2 is loop invariant.
    _CharT __f2 = *__first2;
    while (true) {
        __len1 = __last1 - __first1;
        // Check whether __first1 still has at least __len2 bytes.
        if (__len1 < __len2)
            return __last1;

        // Find __f2 the first byte matching in __first1.
        __first1 = _Traits::find(__first1, __len1 - __len2 + 1, __f2);
        if (__first1 == 0)
            return __last1;

        if (_Traits::compare(__first1, __first2, __len2) == 0)
            return __first1;

        ++__first1; // TODO: Boyer-Moore can be used.
    }
}
```



Find the first matching character

Match rest of the string

Performance improvements

Benchmark	Without patch	With patch	Gain
BM_StringFindMatch1/32768	28157 ns	2203 ns	12.8x
BM_StringFindMatch2/32768	28161 ns	2204 ns	12.8x

```
// Match somewhere towards the end
static void
BM_StringFindMatch1(benchmark::State &state)
{
    std::string s1(MAX_STRING_LEN / 2, '*');
    s1 += std::string(state.range(0), '-');

    std::string s2(state.range(0), '-');

    while (state.KeepRunning())
        benchmark::DoNotOptimize(s1.find(s2));
}
```

```
// Match somewhere from middle to the end.
static void
BM_StringFindMatch2(benchmark::State &state)
{
    std::string s1(MAX_STRING_LEN / 2, '*');
    s1 += std::string(state.range(0), '-');
    s1 += std::string(state.range(0), '*');

    std::string s2(state.range(0), '-');

    while (state.KeepRunning())
        benchmark::DoNotOptimize(s1.find(s2));
}
```

Missing inlining opportunities in basic_string

- Important functions not inlined.
 - `basic_string::__init(const value_type* __s, size_type __sz)`
 - `basic_string::~~basic_string()`
- Clang front end does not emit the definition of these functions (extern templates) in the IR
- Solution
 - Mark functions as inline

Missing function attributes (libc++)

- Missing `__attribute__((__noreturn__))` in important functions.
 - Prevents important compiler optimizations
 - Results in false positives in static analysis results
- `__throw.*` functions in `__locale`, `deque`, `future`, `regex`, `system_error`, `vector`

Example:

```
class __vector_base_common {  
protected:  
    _LIBCPP_ALWAYS_INLINE __vector_base_common() {}  
    void __throw_length_error() const _LIBCPP_NORETURN_ON_EXCEPTIONS;  
    void __throw_out_of_range() const _LIBCPP_NORETURN_ON_EXCEPTIONS;  
};
```

Issues with number parsing in locale (libc++)

- Uses `std::string` to store the parsed numbers
 - Results in multiple (unnecessary) calls to `memset`
- Possible characters for all kinds of numbers (octal, hex, decimal) are stored in one string
 - `__atoms = "0123456789abcdefABCDEFxX+-pPiInN"`
- Makes unnecessary copies of '`__atoms`' string which are never modified in common case

Issues with number parsing in locale (libc++)

- Avoiding copy of `__atoms` is hard because of ABI incompatibilities.
- Current workaround is to version the change with a macro

Benchmark	Without patch	With patch	Gain
BM_istream_numbers/32	8336 ns	7472 ns	11%

- Benchmark source:
 - `std-benchmark/cxx/stringstream.bench.cpp`
 - <https://reviews.llvm.org/D30268>

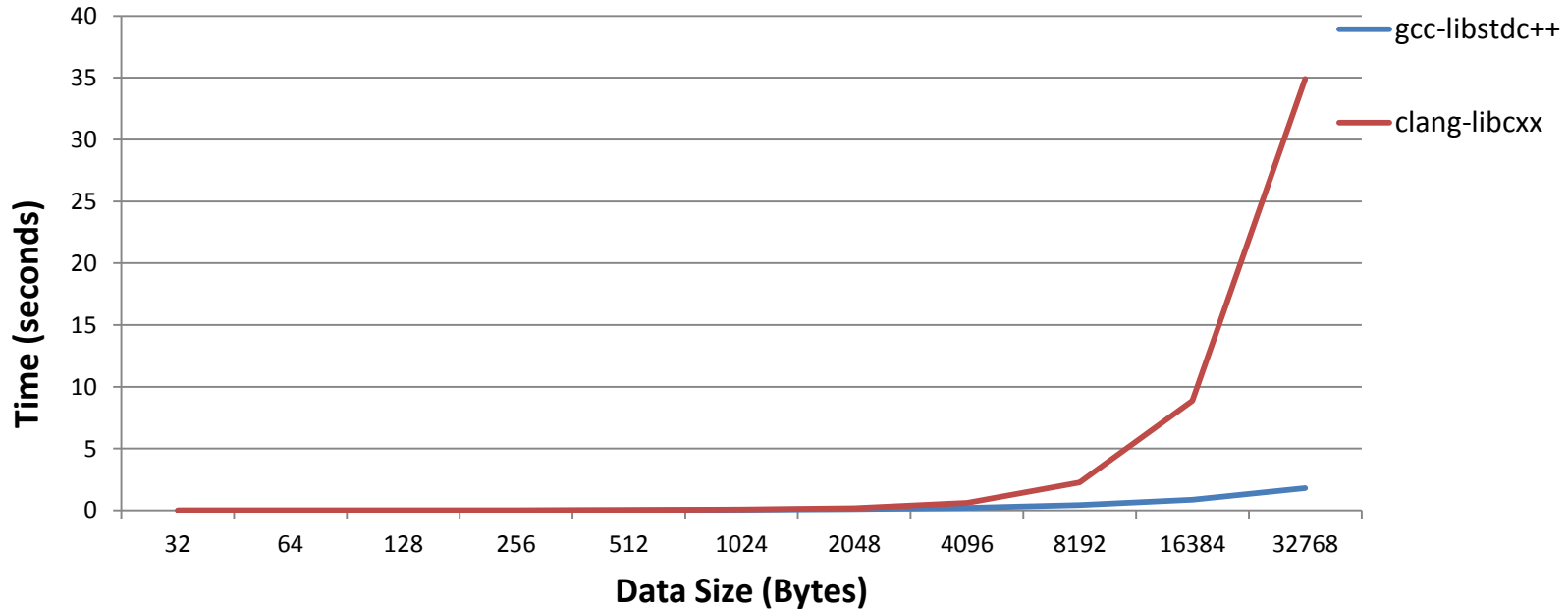
std-benchmark

- <https://github.com/hiraditya/std-benchmark>
 - WIP
 - Builds on Linux, Windows (thanks to cmake)
 - Performance numbers are very stable (based on google-benchmark)

Issues with sort (libc++)

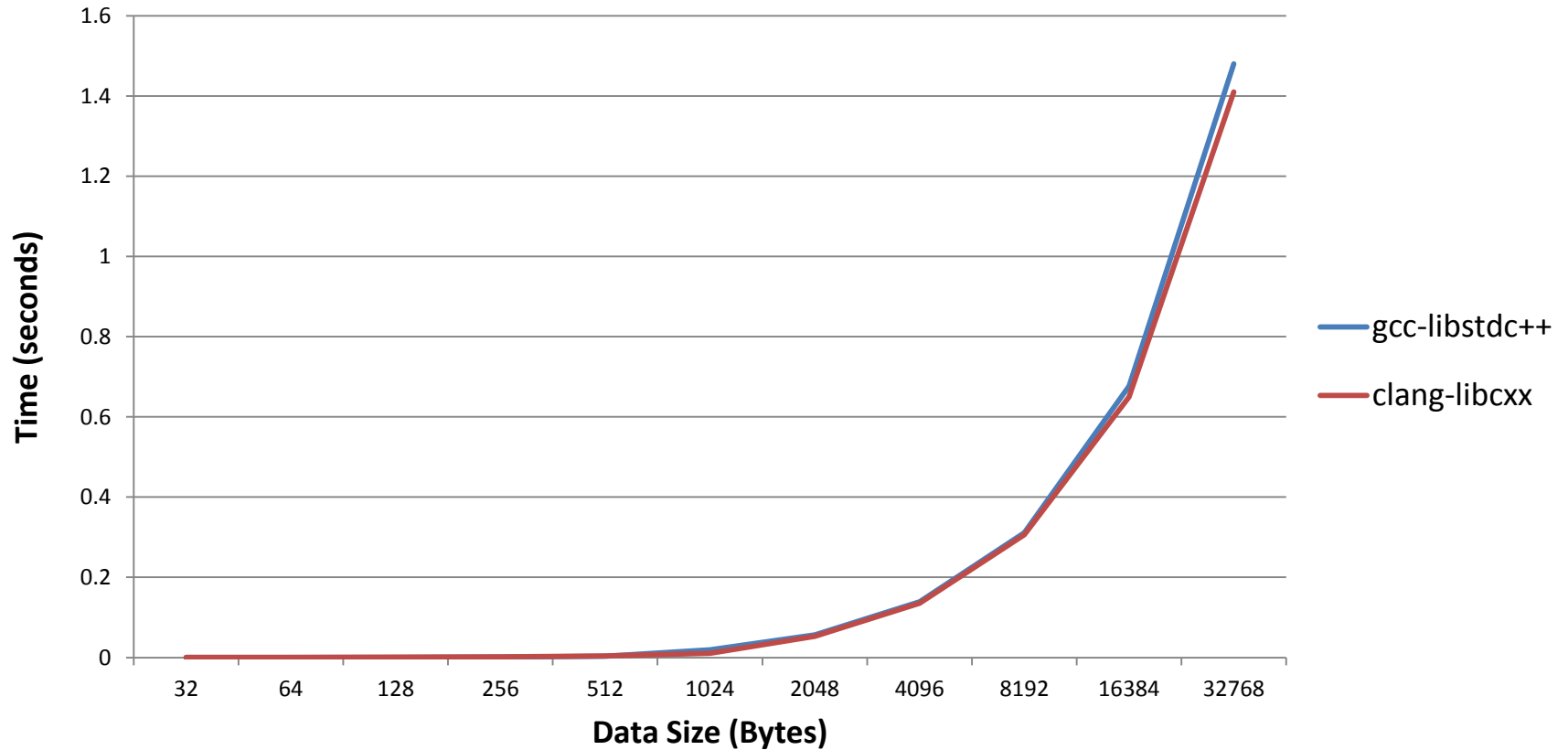
- Worst case $O(N^2)$ comparisons against gcc-libstdc++ $O(N \lg N)$
 - PR20837

Time complexity (worst case)



sort (Average case)

Time complexity (average case)



string::find

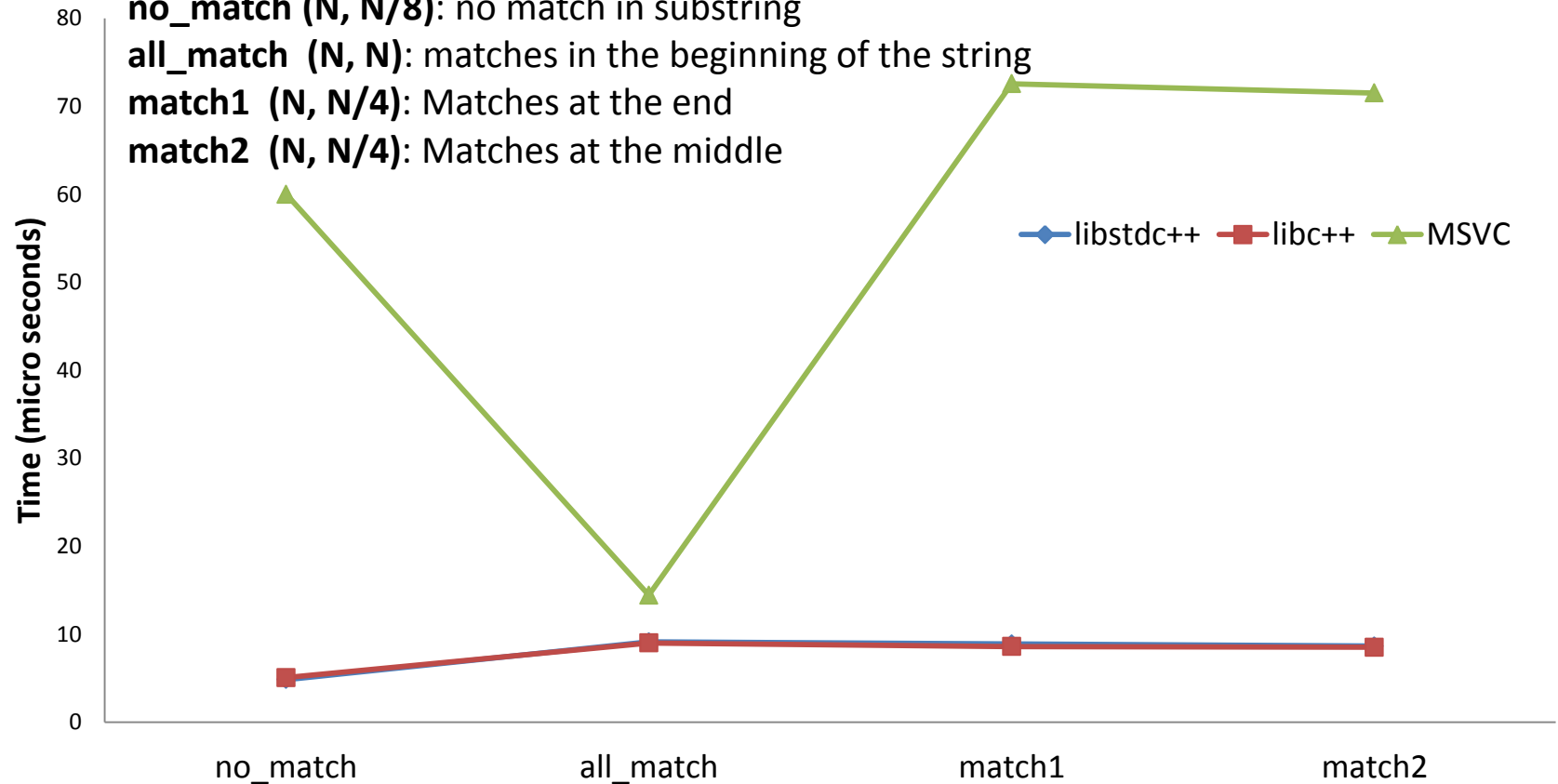
Data Size: 256KB

no_match (N, N/8): no match in substring

all_match (N, N): matches in the beginning of the string

match1 (N, N/4): Matches at the end

match2 (N, N/4): Matches at the middle



string::find vs. strstr

Data size: 32 KB

no_match (N, N/8): no match in substring

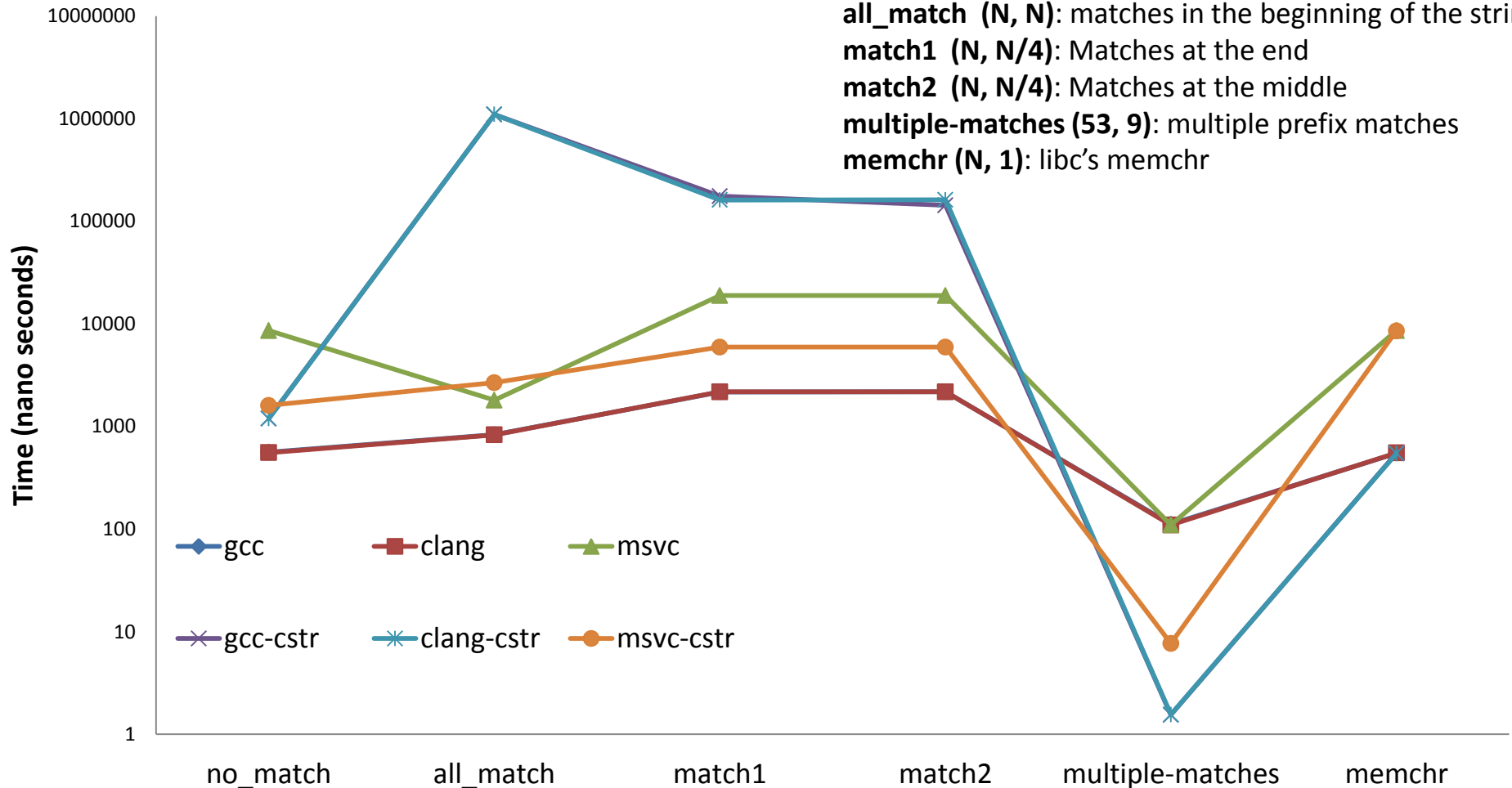
all_match (N, N): matches in the beginning of the string

match1 (N, N/4): Matches at the end

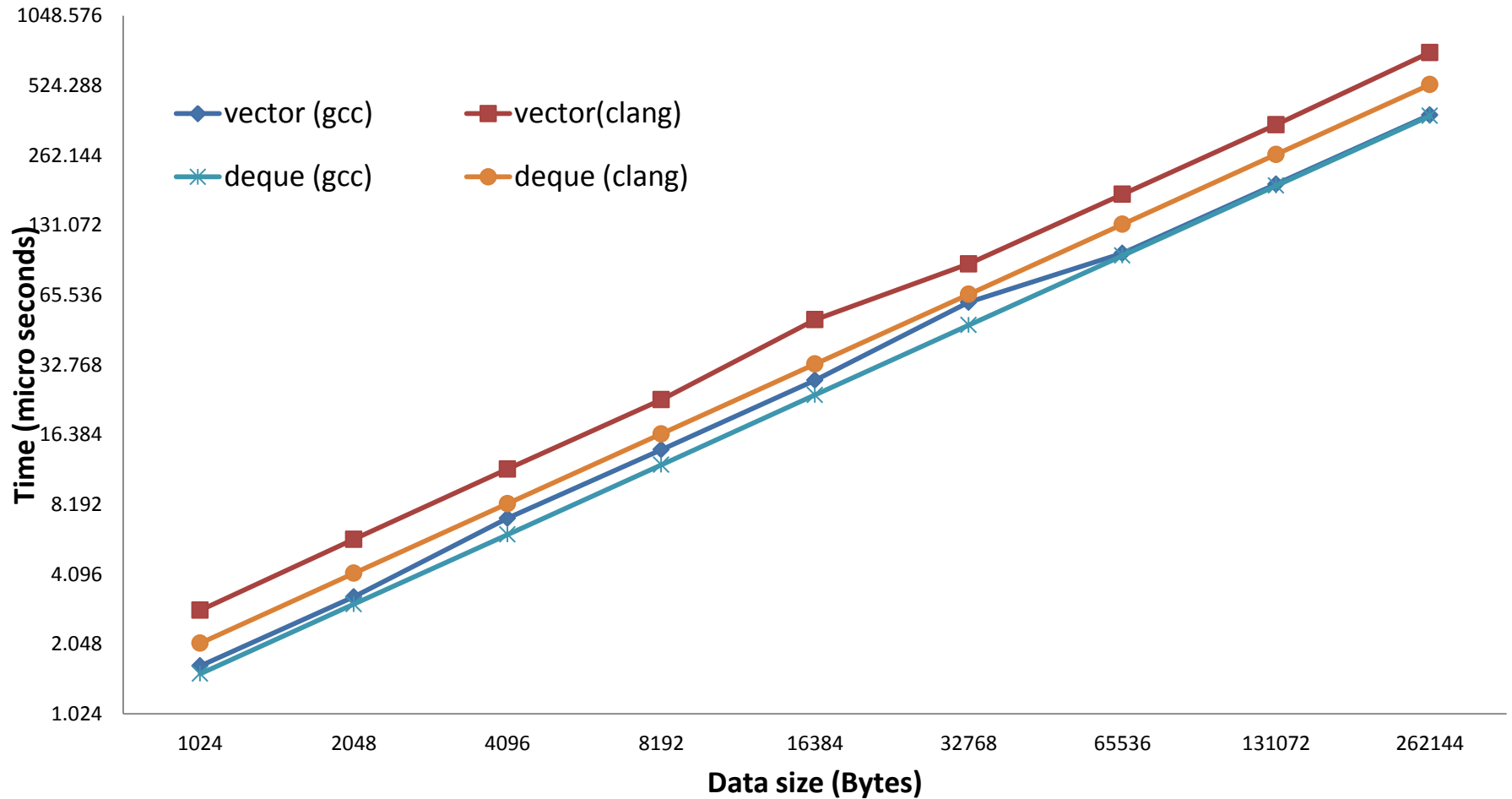
match2 (N, N/4): Matches at the middle

multiple-matches (53, 9): multiple prefix matches

memchr (N, 1): libc's memchr

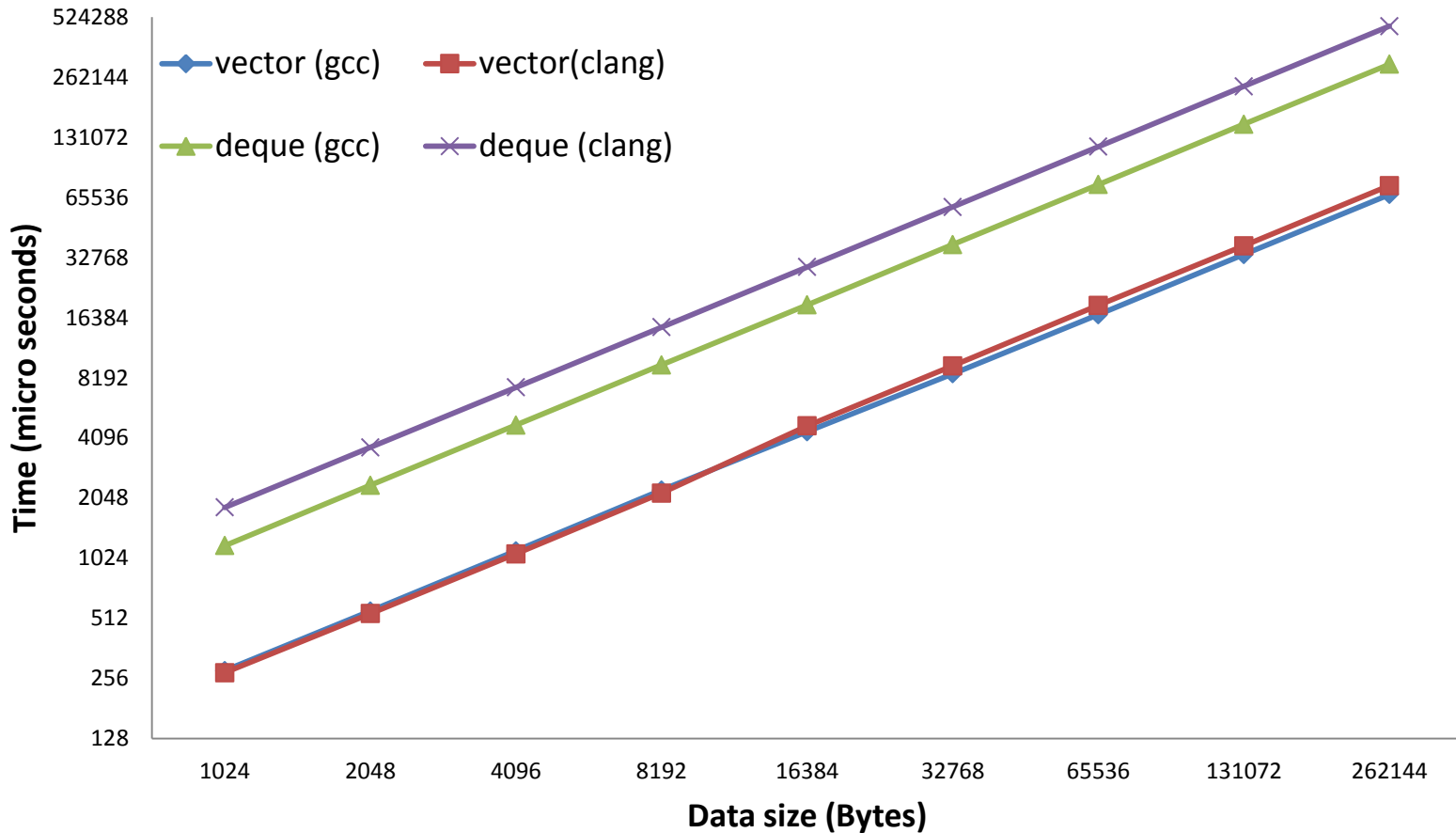


vector vs. deque (push_back)



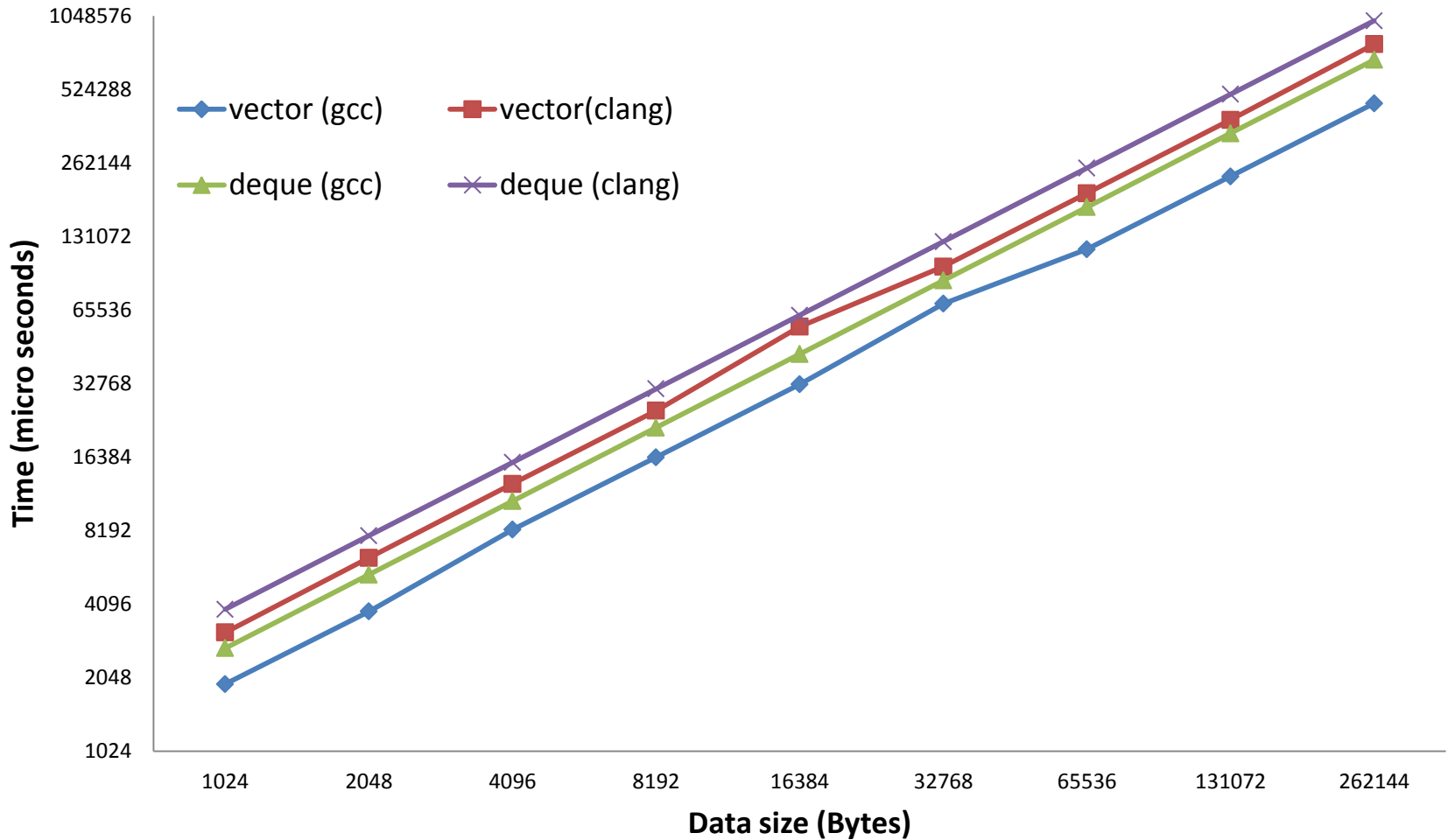
[push_back N elements]

vector vs. deque (access)



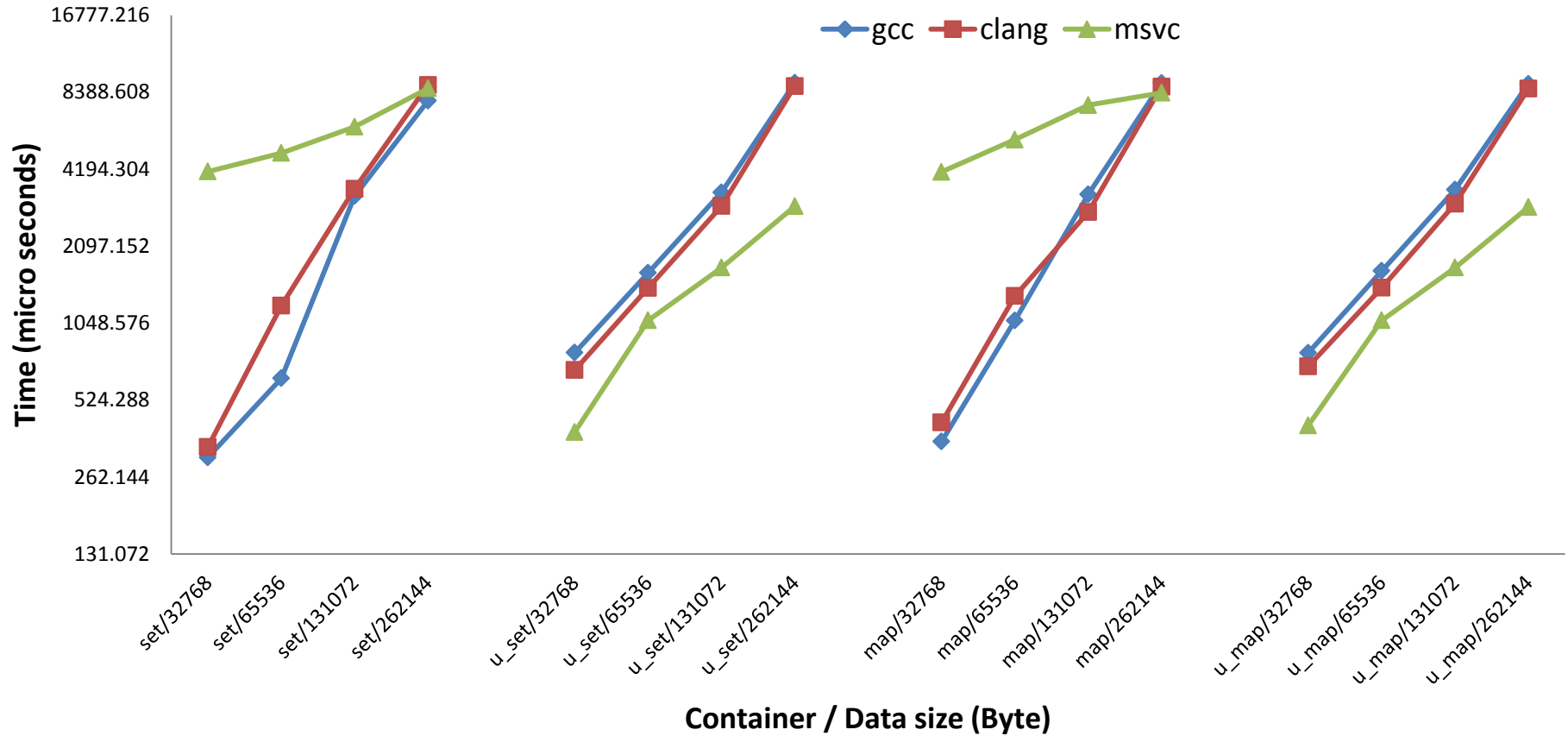
[access N elements in sequence]

vector vs. deque (push_back + access)



[push_back N elements + access N elements in sequence]

Associative vs Hashed Associative (Finding random integers)



compiler vs. programmer

Data: 32KB	Programmer	compiler	C-memcpy
MSVC	11,736ns	11,808ns	1,124ns
clang++	1083ns	1082ns	1478ns
g++	1084ns	1448ns	1460ns

```
const char*
assign(const char *beg,
       const char *end, char *dest) {
    while (beg != end)
        *dest++ = *beg++;
    return beg;
}
```

```
const char*
assign_res(const char * __restrict beg,
           const char * __restrict end,
           char * __restrict dest) {
    while (beg != end)
        *dest++ = *beg++;
    return beg;
}
```

Lessons learned (Algorithms)

- `std::find` may not always be the right choice
- Rotate but not `std::rotate` on linked lists

Lessons learned (containers)

- vector
 - Causes reallocation when enough space is not available ($\sim 2N$ space for N elements)
- If reads and writes are of the same order, `std::deque` may be a better choice (Find the ratio of reads/writes to decide)
- Consider using `unordered_map`/
`unordered_set` instead of `map`/`set`

Lessons learned (containers)

- string
 - calls memset when resized
 - destructor is difficult to optimize away

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();
```

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

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

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

Lessons learned (Language/Library)

- The constructor and destructor cannot be const qualified*
- Iterator based algorithms can lose information and hence, can result in suboptimal performance

(*) Kevlin Henney: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/1995/N0798.htm>

Size (in bytes) of empty containers

64 bit

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

Optimize for latency

Memory	Latency (cycles)
L1	4
L2	12
L3	36
RAM	36+57ns

Intel i7-4770 3.4GHz (Turbo Boost off) 22 nm. RAM: 32 GB (PC3-12800 cl11 cr2).

Source: <http://www.7-cpu.com/cpu/Haswell.html>

References

- <https://gcc.gnu.org/onlinedocs/libstdc++/index.html>
- http://clang-analyzer.llvm.org/annotations.html#attr_noreturn
- <https://reviews.llvm.org/D21103>
- <https://reviews.llvm.org/D22782>
- <https://reviews.llvm.org/D22834>
- <https://reviews.llvm.org/D21232>
- <https://reviews.llvm.org/D27068>
- <https://github.com/google/benchmark>
- <https://github.com/hiraditya/std-benchmark>