

Performance analysis and optimization

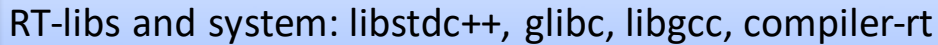
Aditya Kumar

ACM Distinguished Speaker

Twitter: @_hiraditya_

Code:

Compiler: GCC, LLVM, ICC, MSVC



Hardware: AArch64, Intel, RISC-V



Ways to improve performance

Improve runtime of algorithms

- Removing recursion, redundancies
- Micro optimizations tend to become less relevant with new compilers/runtime/hardware

Change algorithms

- Bubble sort to quick sort

Using right data structures

- list to vector, map to hash_map
- Encoding data intelligently

Switching to recent compiler versions

- GCC 12

Using improved language features, or optimize existing language semantics

- C++17/20 (RVO, compile time evaluations, noexcept)

Changing the programming language

- Java to C++
- Customizing the compiler to modify semantics of the language within certain constraints

Profile based optimization

- Hand optimization, AutoFDO, PGO

Analyzing system performance

- Performance analysis pitfalls

Improve runtime of algorithms

Suboptimal basic_streambuf::xsgetn (libc++)

```
template <class _CharT, class _Traits>
streamsize
xsgetn(char_type* __s, streamsize __n) {
    streamsize __i = 0;
    for (; __i < __n; ++__i, ++__s) {
        if (__ninp_ < __einp_)
            *__s = *__ninp_++;
        else
            break;
    }
    return __i;
}
```

Optimized basic_streambuf::xsgetn

// Before

```
template <class _CharT, class _Traits>
streamsize
xsgetn(char_type* __s, streamsize __n) {
    streamsize __i = 0;
    for (; __i < __n; ++__i, ++__s) {
        if (__ninp_ < __einp_)
            * __s = * __ninp_ ++;
        else
            break;
    }
    return __i;
}
```

// After

```
template <class _CharT, class _Traits>
streamsize xsgetn(char_type* __s, streamsize __n)
{
    streamsize __i = 0;
    while (__i < __n) {
        if ( __ninp_ < __einp_ ) {
            const streamsize __len = std::min(__einp_ - __ninp_, __n - __i);
            traits_type::copy( __s, __ninp_, __len);
            __s += __len;
            __i += __len;
            this->gbump(__len);
        }
        else
            break;
    }
    return __i;
}
```

Suboptimal string::find algorithm

```
// 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;
            }
        } // End While
```

} Match rest of the string

```
    } // End While
```

```
}
```

Optimized string::find algorithm

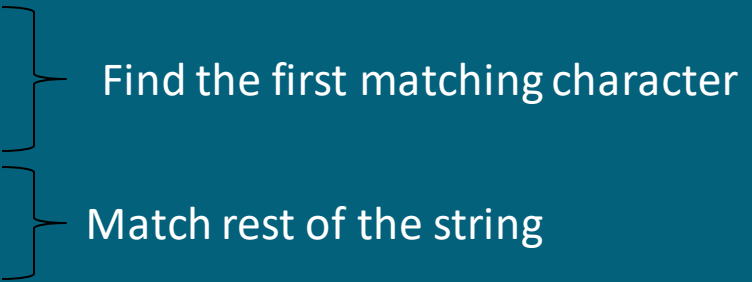
```
const _CharT *
__search_substring(const _CharT *__first1, const _CharT *__last1, const _CharT *__first2, const _CharT *__last2)
{
    // First element of __first2 is loop invariant.
    _CharT __f2 = *__first2;

    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.
    } // End While
}
```



Find the first matching character

Match rest of the string

Performance improvements

Benchmark	Without patch	With patch	Gain
Test1/32768	28157 ns	2203 ns	12.8x
Test2/32768	28161 ns	2204 ns	12.8x

Missing inlining opportunities in `basic_string` (libc++)

Important functions not inlined

- `basic_string::__init(const value_type* __s, size_type __sz)`
- `basic_string::~~basic_string()`

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:  
    __vector_base_common() {}  
    __attribute__((__noreturn__)) void __throw_length_error() const;  
    __attribute__((__noreturn__)) void __throw_out_of_range() const;  
};
```

Issues with number parsing in locale (libc++)

Uses `std::string` to store the parsed numbers

- Results in (unnecessary) calls to `memset`

Possible characters for all kinds of numbers (octal, hex, decimal) stored in one string

- `__atoms = "0123456789abcdefABCDEFxX+-pPiInN"`

Makes unnecessary copies of `'__atoms'` string which are not modified in common case

compiler vs. programmer vs. hand-optimized

Relative performance w.r.t. g++ (Lower is better)			
Data: 32KB	programmer	compiler	C-memcpy
MSVC	11	11	1.04
clang++	1	1	1.3
g++	1	1.3	1.3

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

```
// Helping compiler with user annotation
const char*
assign_res(const char * __restrict beg,
           const char * __restrict end,
           char * __restrict dest)
{
    while (beg != end)
        *dest++ = *beg++;
    return beg;
}
```

Change algorithms

Bernstein Hash

```
uint32_t ComputeHash(const ZipString& name)
{
    uint32_t hash = 0;
    uint16_t len = name.name_length;
    const uint8_t* str = name.name;
    while (len--) {
        hash = hash * 31 + *str++;
    }
    return hash;
}
```

Improved Bernstein Hash

// After

```
uint32_t ComputeHash(const ZipString& name)
```

```
{
```

```
    uint32_t hash = 0;
```

```
    uint16_t len = name.name_length;
```

```
    const uint8_t* str = name.name;
```

```
    unsigned chunk;
```

```
    const unsigned sz = sizeof(chunk);
```

```
    // Hash sz bytes at a time.
```

```
    while (len > sz) {
```

```
        // Why not plain typecast??
```

```
        __builtin_memcpy(&chunk, str, sz);
```

```
        hash = hash * 31 + chunk;
```

```
        len -= sz;
```

```
        str += sz;
```

```
    }
```

```
    // Hash the left-over bytes.
```

```
    while (len--) {
```

```
        hash = hash * 31 + *str++;
```

```
    }
```

```
    return hash;
```

```
}
```

// Before

```
uint32_t ComputeHash(const ZipString& name)
```

```
{
```

```
    uint32_t hash = 0;
```

```
    uint16_t len = name.name_length;
```

```
    const uint8_t* str = name.name;
```

```
    while (len--) {
```

```
        hash = hash * 31 + *str++;
```

```
    }
```

```
    return hash;
```

```
}
```


Comment: standard library algorithms

Iterator based algorithms can lose information and hence, can result in suboptimal performance

- `std::rotate` on doubly linked list. Same algorithms can be written optimally for ranges

No optimized algorithms for non-char arrays

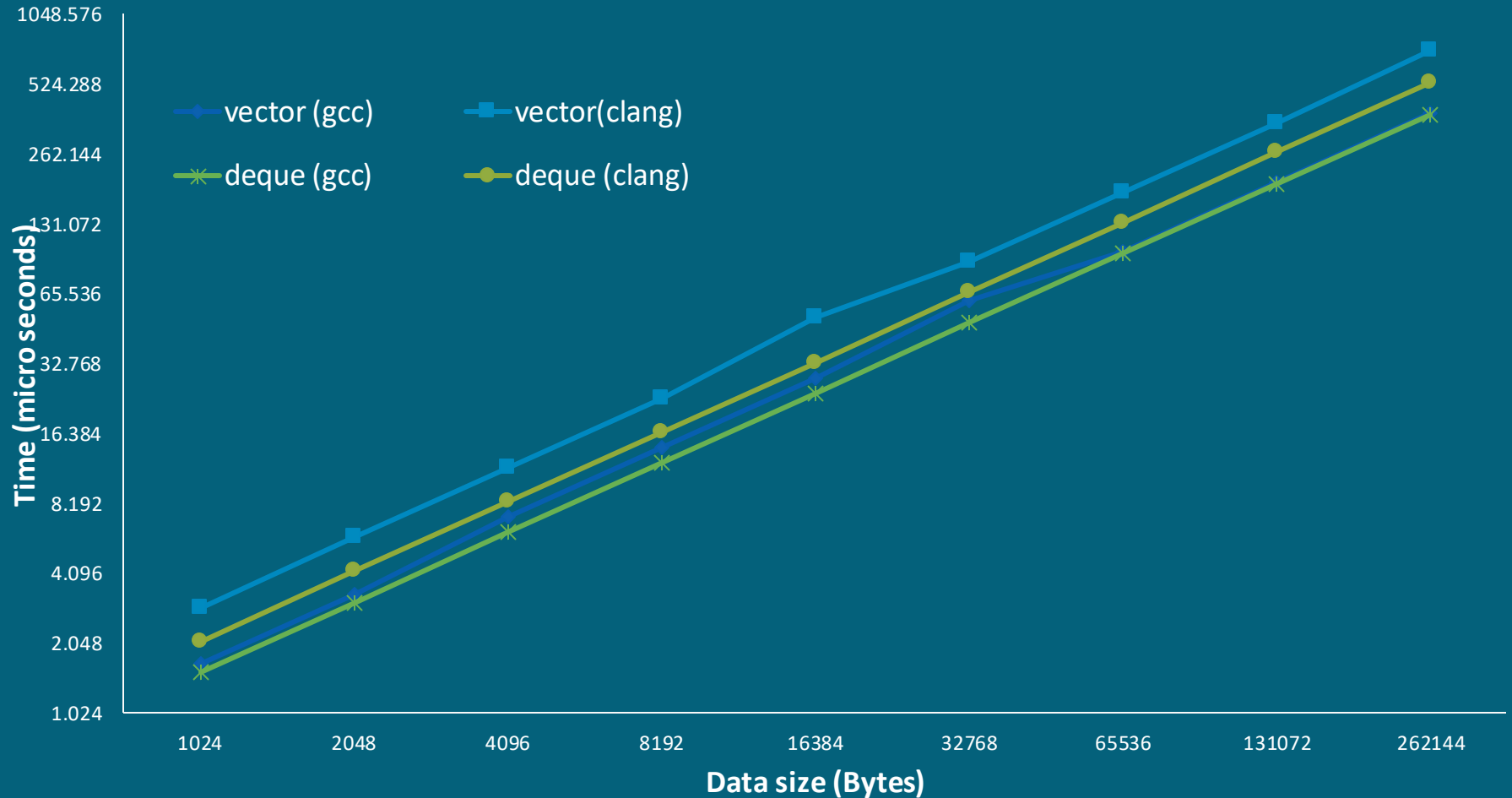
- Copying an array of pairs

`std::find` may not always be the right choice

- `substr`

Using the right data structure

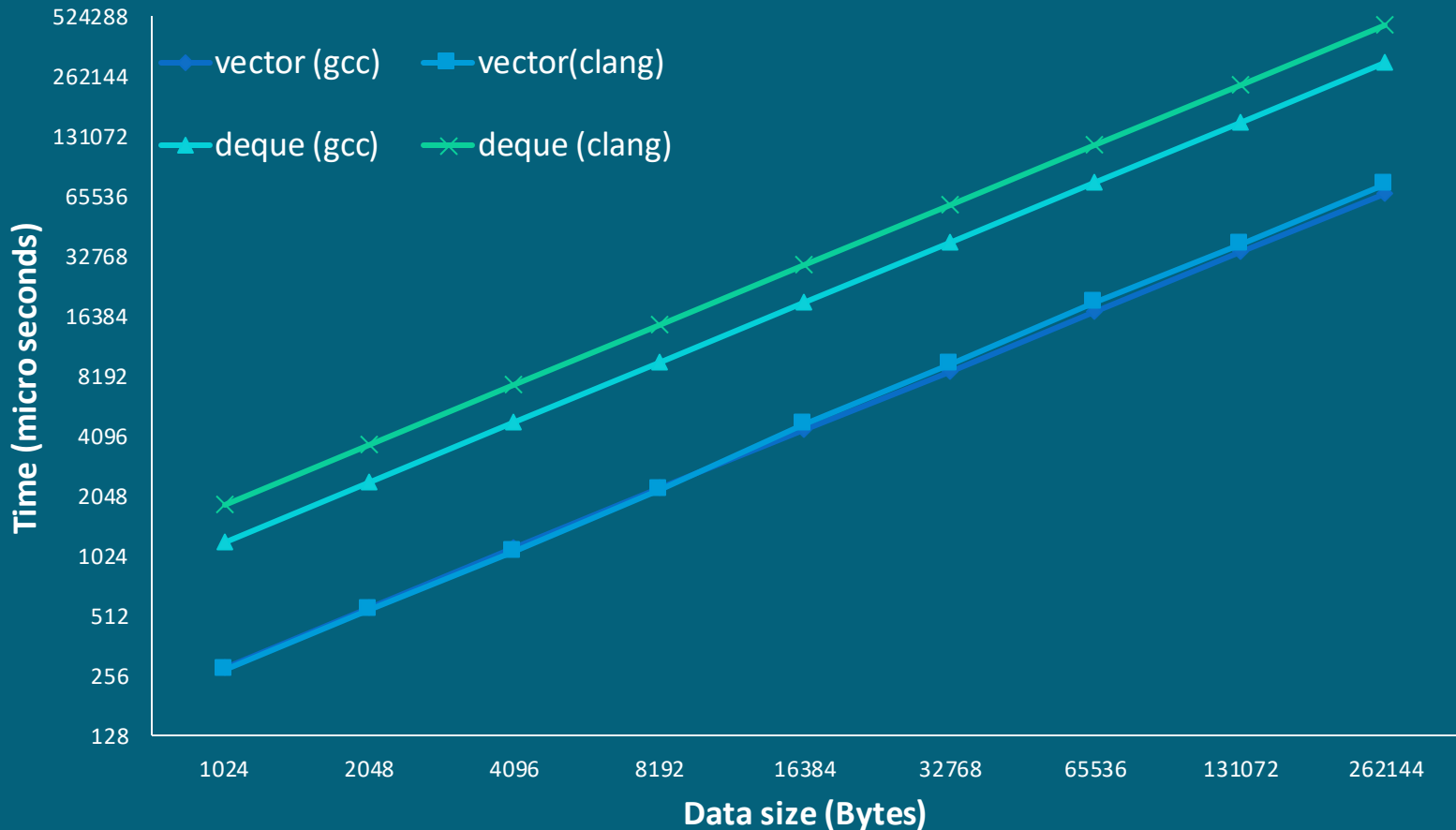
vector vs. deque (push_back)



* [push_back N elements]

* Lower is better.

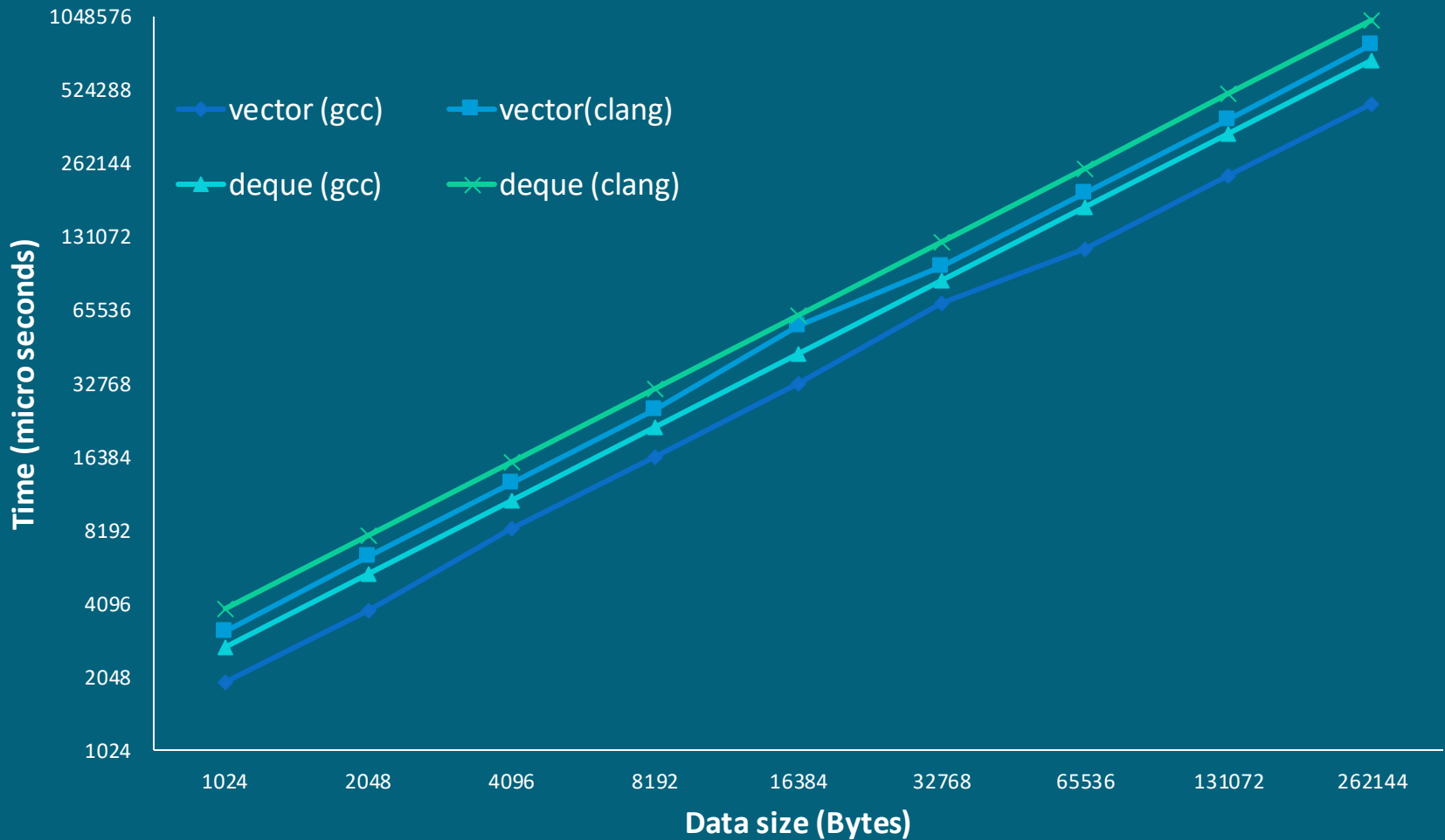
vector vs. deque (access)



* [access N elements in sequence]

* Lower is better.

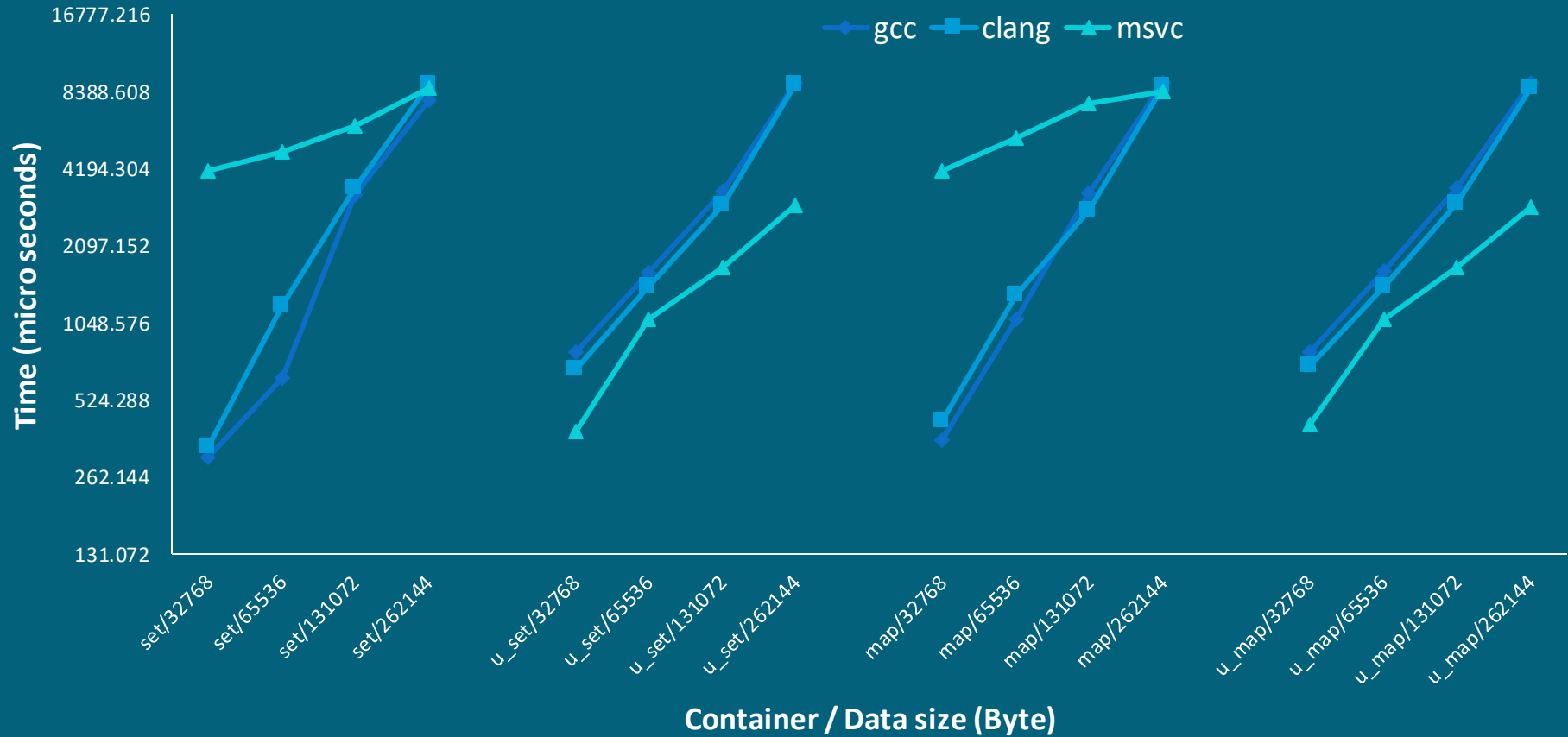
vector vs. deque (push_back + access)



* [push_back N elements + access N elements in sequence]

* Lower is better

Associative vs Hashed Associative (Finding random integers)



* Lower is better

Comment: standard library containers

Consider total cost

- Take ratio of reads/writes to decide
- vector causes memory fragmentation ($\sim 2N$ allocations for N elements)
- if reads < writes, deque can be a better choice

'resize' initializes the memory

Encoding data intelligently

Find two numbers such that sum is zero

```
#include<algorithm>
#include<array>
#include<iostream>

template<typename T>
// Returns the positions in [b, e) which sum to zero.
std::pair<T, T> find_complement(T b, T e) {
    T p1 = b;
    T p2 = e;
    --p2; // Point to the last element of the range.
    while (p1 != p2) {
        int sum = *p1 + *p2;
        if (sum == 0)
            return {p1, p2}; // Preserve the information computed
        else if (sum < 0)
            ++p1;
        else // sum > 0
            --p2;
    }
    return {nullptr, nullptr};
}

int main() {
    std::array<int, 9> a{ -4, -4, -1, 0, 1, 2, 3, 4, 4 };
    std::pair<int*, int*> v = find_complement(a.begin(), a.end());
    if (v.first != nullptr) {
        std::cout << "\nFound: " << *v.first << ", " << *v.second;
    }
    return 0;
}
```


Encoding data intelligently

Find all pairs such that sum is zero

```
#include<algorithm>
#include<array>
#include<iostream>

template<typename T>
// Returns the positions in [b, e) which sum to zero.
std::pair<T, T> find_complement(T b, T e) {
    T p1 = b;
    T p2 = e;
    --p2; // Point to the last element of the range.
    while (p1 != p2) {
        int sum = *p1 + *p2;
        if (sum == 0)
            return {p1, p2}; // Preserve the information computed
        else if (sum < 0)
            ++p1;
        else // sum > 0
            --p2;
    }
    return {nullptr, nullptr};
}

int main() {
    std::array<int, 9> a{ -4, -4, -1, 0, 1, 2, 3, 4, 4 };
    std::pair<int*, int*> v = find_complement(a.begin(), a.end());
    while (v.first != nullptr) {
        std::cout << "\nFound: " << *v.first << ", " << *v.second;
        v = find_complement(++v.first, v.second);
    }
    return 0;
}
```

Switching compiler/language/libraries

Switching language version

```
#include<string>
```

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

```
$ g++ -O3 t.cpp -S -std=c++20 -o -
```

```
main:  
xor eax, eax  
ret
```

```
$ g++ -O3 t.cpp -S -std=c++17 -o -
```

```
main:  
push rbp  
mov esi,97  
push rbx  
sub rsp,40  
movzx eax,WORD PTR .LC0[rip]  
lea rbx,[rsp+16]  
mov rdi,rsp  
mov QWORD PTR [rsp+8],1  
mov QWORD PTR [rsp],rbx  
mov WORD PTR [rsp+16],ax  
call _ZNSt7__cxx1112basic_stringlcSt11char_traitslcESalcEE9push_backEc  
mov rdi,QWORD PTR [rsp]  
cmp rdi,rbx  
je .L6  
mov rax,QWORD PTR [rsp+16]  
lea rsi,[rax+1]  
call _ZdlPvm  
.L6:  
add rsp,40  
xor eax,eax  
pop rbx  
pop rbp  
ret  
mov rbp,rax  
jmp .L3  
main.cold:  
.LC0:  
.byte 97  
.byte 0
```

Switching library

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

Performance analysis tools

Valgrind

Linux Perf

Visual studio performance tool

Intel Vtune

XCode Instruments

Performance Analysis with Valgrind

```
valgrind [--tool=memcheck]
```

- valgrind mostly known for its memory leak checker

```
valgrind --tool=cachegrind
```

- cache and branch simulator
- count read, write, and branch instructions

```
valgrind --tool=callgrind
```

- execution call graph
- visualization tool kcachegrind

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

Perf stat: 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
```


Perf record: Example – 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:    ldrrh   w11, [x8]
4.02 629abc:    ldrrh   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
```

AutoFDO: Feedback Directed Optimization

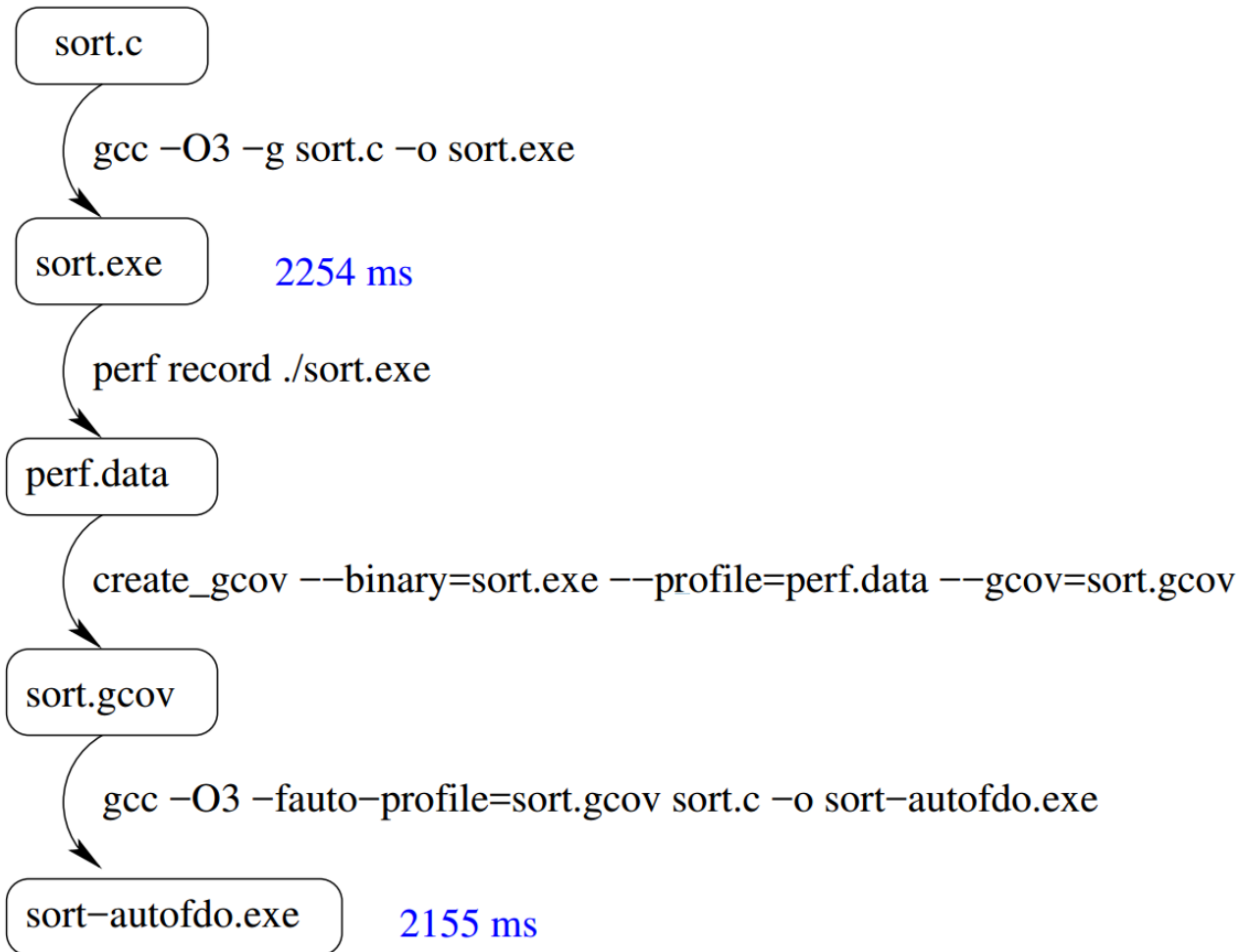
Linux-perf extracts profiles of running systems

little overhead

coverage (basic block frequencies) from dynamic profiles

continuous profiling and tuning of optimizations

AutoFDO: Example



Analyzing System Performance

Vary one Component of the System at a time

- Measure impact of one component on the System
- Run multiple times

Disable frequency scaling

- `cpufrequtils`

Performance metrics

- Dynamic profiles, compiler logs

Systematic performance analysis

- Monitor performance regression over time
- Time series: track performance of system over time
- Git bisect performance changes

Performance analysis pitfalls

- Central tendency: the median instead of the mean
- Use the quantile values instead of a single median value.
- Outlier detection
- Weighted samples for combining historical data

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>

Performance analysis pitfalls

Central tendencies

- The median instead of the mean

Use the quantile values instead of a single median value

- Helps with prioritization

Outlier detection

- Filter outliers

Weighted samples for combining historical data

- Recent data more important than the previous data.

Optimization opportunities!

```
#include<string>
```

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

```
$ g++ -O3 t.cpp -S -fno-exceptions -std=c++20 -o - | grep _ZdlPv  
# Optimized away!
```

```
$ clang++ -O3 t.cpp -S -fno-exceptions -std=c++20 -o - | grep _ZdlPv  
# Optimized away!
```

```
#include<string>
```

```
void foo();
```

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

```
$ g++ -O3 t.cpp -S -fno-exceptions -std=c++20 -o - | grep _ZdlPv  
call    _ZdlPvm
```

```
$ clang++ -O3 t.cpp -S -fno-exceptions -std=c++20 -o - | grep _ZdlPv  
call    _ZdlPv@PLT
```

Optimization opportunities!

```
#include <string>
#include <iostream>
#include <sstream>

static void construct_istream(std::istream& input) {
    std::string str;
    for (int i = 0; i < 10000; ++i)
        str += "asdkljadsflkjasd;flkjasdlfjdasfj7\n";
    input.str(str);
}

static void GetlineBench(benchmark::State& state) {
    // read file line by line
    std::istream input;
    construct_istream(input);
    for (auto _ : state) {
        std::string line;
        std::getline(input, line); // gcc 3x faster than clang.
    }
}

BENCHMARK(GetlineBench);
```

gcc: <https://quick-bench.com/q/S2LGUrHKnYSn1y8WR-AyKh8jrF4>

clang: <https://quick-bench.com/q/nuv3RspJoDe6Eh2ZMtSq8DLKHls>

llvm issue tracker: <https://github.com/llvm/llvm-project/issues/21566>

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)
- https://github.com/hiraditya/hiraditya/tree/master/_posts
- <https://github.com/google/benchmark>
- <https://github.com/hiraditya/std-benchmark>

References and Acknowledgements

Acknowledgements:

- Initial version in collaboration with Sebastian Pop (Samsung Austin R&D Center)

Measurement Pitfalls:

- <https://stats.stackexchange.com/a/156787>
- <https://www.longevitas.co.uk/site/informationmatrix/quantilesandpercentiles.html>
- <https://aakinshin.net/posts/statistics-for-performance/>
- Linux perf: <http://www.brendangregg.com/perf.html>
- Performance Analysis: <https://github.com/AndreyAkinshin/perfolizer>
- Microbenchmarking: <https://www.bfilipek.com/2016/01/micro-benchmarking-libraries-for-c.html>
- C++ benchmark tool: <https://github.com/ivafanas/sltbench>

AutoFDO Resources

- <https://gcc.gnu.org/wiki/AutoFDO/Tutorial>
- <https://github.com/google/autofdo>
- <https://gcc.gnu.org/lists.html>

References and Acknowledgements

Patches:

- <https://android-review.googlesource.com/c/platform/system/core/+369092>