



**EUROPEAN  
SPALLATION  
SOURCE**



# Code Performance Analysis and Visualisation

A blaze through performance tools on macOS

PRESENTED BY MORTEN HILKER-SKAANING

2020-09-13



# Agenda



- 1 macOS Instruments: CPU time sampler
- 2 macOS Instruments: Viewing transient allocations
- 3 Mention Callgrind + QCachegrind
- 4 Understanding machine code using Ghidra
- 5 Fast code with ISPC language + compiler
- 6 Viewing large C++ projects with SourceTrail
- 7 Viewing C++ compilation time profiles

1

# macOS Instruments: CPU time sampler





# macOS Instruments: CPU time sampler

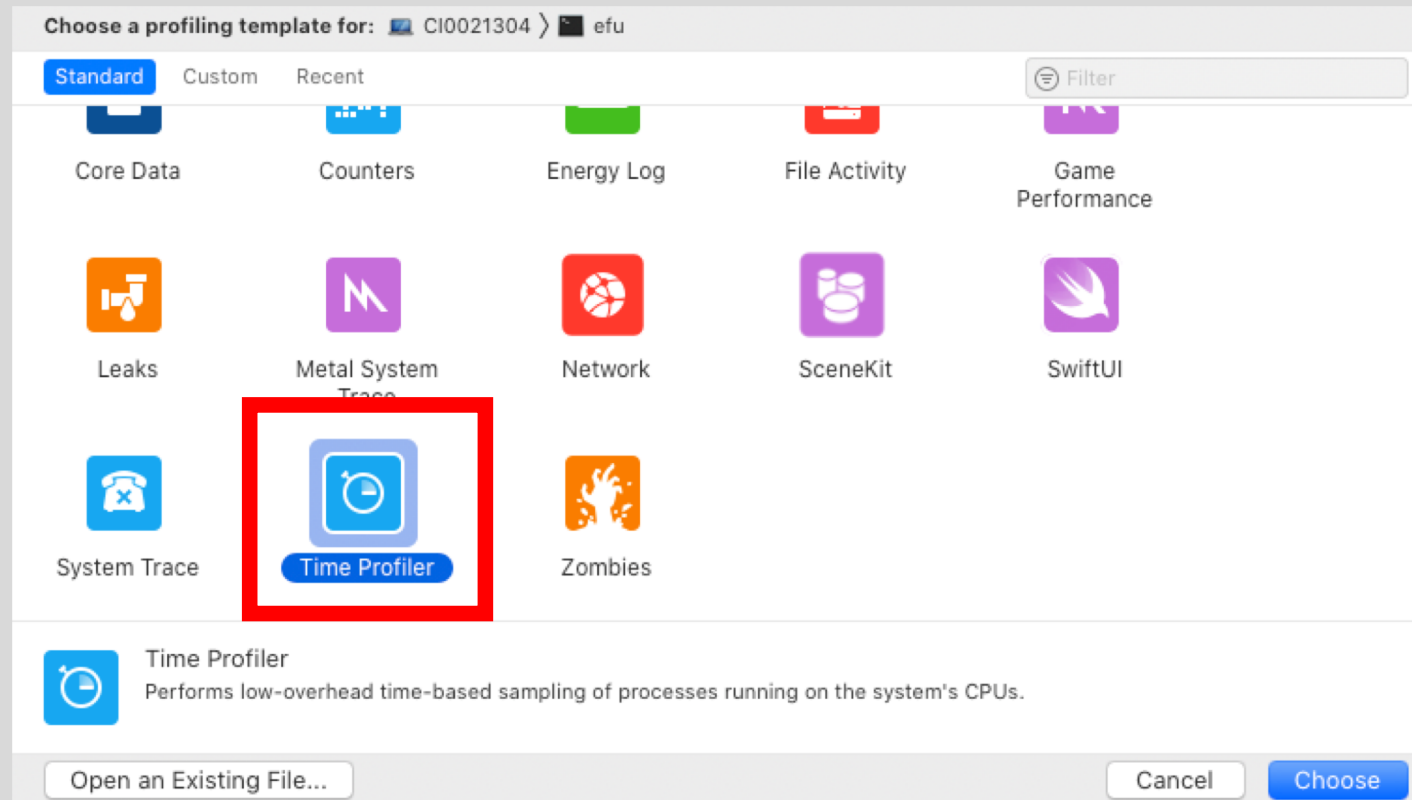
## Install

Easier than Cachegrind+QCachegrind

Bundled with Xcode, worth the download

Case: Allocations in NMX benchmark

# macOS Instruments: CPU time sampler



# macOS Instruments: CPU time sampler



**Select target to run or process to attach to**

Weight	Self Weight	Symbol Name
1.57 s	100.0%	0 s
		►NMXBenchmarkTest (79215)

Heaviest Stack Trace	
15716	NMXBenchmarkTest (79215)
15716	Main Thread 0x1ead23
15346	start
15340	main
15340	benchmark::RunSpecifiedBench
15330	benchmark::internal::(anonymous)
15330	benchmark::Fixture::Run(benchmark::
15327	NmxBenchmarkTest_Dummy_Benchmark
15327	void BenchmarkLoop<NmxBenchmarkTest
15326	NmxBenchmarkTest_Dummy_Benchmark
8053	void BenchmarkLoopPaused<NmxBenchmark
7775	NmxBenchmarkTest_Dummy_Benchmark
5812	gem::HitGenerator::randomHitsFor
5402	gem::HitGenerator::makeHitsFor
4235	std::_1::vector<Hit, std::_1::allocator
4049	void std::_1::vector<Hit, std::_1::allocator
2340	std::_1::_split_buffer<Hit, std::_1::allocator
2340	std::_1::_split_buffer<Hit, std::_1::allocator
2296	std::_1::allocator_traits<std::_1::allocator
2296	std::_1::allocator<Hit>::allocate
2291	std::_1::_libcxx_allocate(unsigned long)
2280	operator new(unsigned long)
2195	malloc
2130	malloc_zone_malloc
1969	szone_malloc_should_clear
1834	tiny_malloc_should_clear

# macOS Instruments: CPU time sampler



**Choose Target:**

Executables  
CI0021304  
Recents  
Favorites  
Volumes

Launchd  
Agents  
Daemons  
Recents

- ESSGeometryBenchmarkTest
- MBDataParserBenchmarkTest
- MBEventBuilderBenchmarkTest
- NMXBenchmarkTest
- NMXClustererBenchmarkTest

Name NMXBenchmarkTest  
Size 821,18 KiB  
Created Saturday, 12 September 2020 at  
Modified Saturday, 12 September 2020 at

Environment Variable	Value

+ - ⚙

Show Hidden Files  Traverse Packages

--benchmark\_filter=Dummy

/Users/mortenhilkerskaaning/dev/event-formation-unit/build

Cancel Choose

# macOS Instruments: CPU time sampler



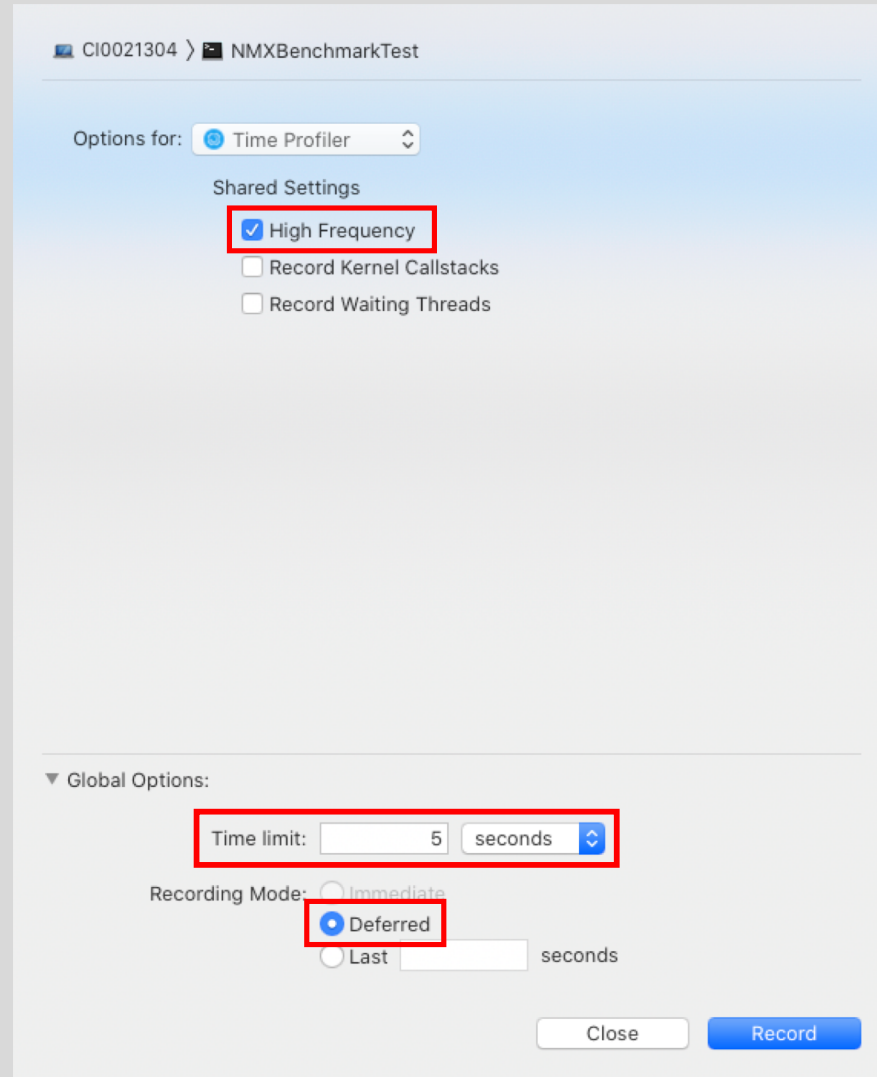
Hold → "Recording Options"

Weight	Self Weight	Symbol Name
1.57 s	100.0%	0 s
		►NMxBenchmarkTest (79215)

**Heaviest Stack Trace**

- 15716 NMxBenchmarkTest (79215)
- 15716 Main Thread 0x1ead23
- 15346 start
- 15340 main
- 15340 benchmark::RunSpecifiedBenc
- 15330 benchmark::internal::(anonymo
- 15330 benchmark::Fixture::Run(benc
- 15327 NmxBenchmarkTest\_Dummy\_Br
- 15327 void BenchmarkLoop<NmxBen
- 15326 NmxBenchmarkTest\_Dummy\_Br
- 8053 void BenchmarkLoopPaused<Nm
- 7775 NmxBenchmarkTest\_Dummy\_Ber
- 5812 Gem::HitGenerator::randomHits(
- 5402 Gem::HitGenerator::makeHitsFor
- 4235 std::\_1::vector<Hit, std::\_1::alk
- 4049 void std::\_1::vector<Hit, std::\_1
- 2340 std::\_1::\_split\_buffer<Hit, std;:
- 2340 std::\_1::\_split\_buffer<Hit, std;:
- 2296 std::\_1::allocator\_traits<std::\_1
- 2296 std::\_1::allocator<Hit>::allocate
- 2291 std::\_1::\_libcxx\_allocate(unsign
- 2280 operator new(unsigned long)
- 2195 malloc
- 2130 malloc\_zone\_malloc
- 1969 szone\_malloc\_should\_clear
- 1834 tiny\_malloc\_should\_clear

# macOS Instruments: CPU time sampler







# macOS Instruments: CPU time sampler

Time Profiler > Profile > Root

Weight	Self Weight	Symbol Name	
823.30 ms	100.0%	0 s	▼NMXBenchmarkTest (79255) +
823.30 ms	100.0%	0 s	▼Main Thread 0x1ebd9d
789.70 ms	95.9%	0 s	▼start libdyld.dylib
789.70 ms	95.9%	0 s	▼main NMXBenchmarkTest
789.70 ms	95.9%	0 s	▼benchmark::RunSpecifiedBenchmarks(benchmark::BenchmarkReporter*, benchmark::BenchmarkReporter*) NMXBenchmarkTest
788.70 ms	95.7%	0 s	▼benchmark::internal::(anonymous namespace)::RunInThread(benchmark::internal::Benchmark::Instance const*, unsigned long, int, benchr
788.70 ms	95.7%	100.00 μs	▼benchmark::Fixture::Run(benchmark::State&) NMXBenchmarkTest
788.40 ms	95.7%	0 s	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&) NMXBenchmarkTest
788.40 ms	95.7%	0 s	▼void BenchmarkLoop<NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&):\$_0>(benchmark::State&, Nm
788.30 ms	95.7%	500.00 μs	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&):\$_0::operator()() const NMXBenchmarkTest
417.90 ms	50.7%	0 s	▶void BenchmarkLoopPaused<NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&):\$_0::operator()() c
279.90 ms	33.9%	0 s	▶_cluster_plane(std::_1::vector<Hit, std::_1::allocator<Hit> >&, std::_1::shared_ptr<AbstractClusterer>, std::_1::shared_ptr<Ab
79.80 ms	9.6%	2.00 ms	▶CenterMatcher::match(bool) NMXBenchmarkTest
7.20 ms	0.8%	1.30 ms	▶EventAnalyzer::analyze(Event&) const NMXBenchmarkTest
1.30 ms	0.1%	0 s	▶<Unknown Address>
900.00 μs	0.1%	400.00 μs	▶Event::both_planes() const NMXBenchmarkTest
500.00 μs	0.0%	500.00 μs	szone_free_definite_size libsystem_malloc.dylib
300.00 μs	0.0%	300.00 μs	DYLD-STUB\$\$operator delete(void*) NMXBenchmarkTest
100.00 μs	0.0%	0 s	▶benchmark::State::end() NMXBenchmarkTest
100.00 μs	0.0%	100.00 μs	szone_free_definite_size libsystem_malloc.dylib
100.00 μs	0.0%	0 s	▶<Unknown Address>
400.00 μs	0.0%	0 s	▶benchmark::ConsoleReporter::ReportContext(benchmark::BenchmarkReporter::Context const&) NMXBenchmarkTest
400.00 μs	0.0%	0 s	▶benchmark::internal::FindBenchmarksInternal(std::_1::basic_string<char, std::_1::char_traits<char>, std::_1::allocator<char> > const&
100.00 μs	0.0%	0 s	▶benchmark::BenchmarkReporter::Context::Context() NMXBenchmarkTest
100.00 μs	0.0%	0 s	▶0x106c574e7 NMXBenchmarkTest
33.60 ms	4.0%	0 s	▶_dyld_start dyld

Input Filter  [Call Tree](#) Call Tree Constraints Data Mining



# macOS Instruments: CPU time sampler

Time Profiler > Profile > Root

Weight	Self Weight	Symbol Name	
823.30 ms	100.0%	0 s	▼NMXBenchmarkTest (79255)
823.30 ms	100.0%	0 s	▼Main Thread 0x1ebd9d
789.70 ms	95.9%	0 s	▼start libdyld.dylib
789.70 ms	95.9%	0 s	▼main NMXBenchmarkTest
789.70 ms	95.9%	0 s	▼benchmark::RunSpecifiedBenchmarks(benchmark::BenchmarkReporter*, benchmark::BenchmarkReporter*) NMXBenchmarkTest
788.70 ms	95.7%	0 s	▼benchmark::internal::(anonymous namespace)::RunInThread(benchmark::internal::Benchmark::Instance const*, unsigned long, int, benchr
788.70 ms	95.7%	100.00 µs	▼benchmark::Fixture::Run(benchmark::State&) NMXBenchmarkTest
788.40 ms	95.7%	0 s	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&) NMXBenchmarkTest
788.40 ms	95.7%	0 s	▼void BenchmarkLoop<NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0>(benchmark::State&, Nm
788.30 ms	95.7%	500.00 µs	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const NMXBenchmarkTest
417.90 ms	50.7%	0 s	▶void BenchmarkLoopPaused<NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const NMXBenchmarkTest
279.90 ms	33.9%	0 s	▶_cluster_plane(std::__1::vector<Hit, std::__1::allocator<Hit>>&, Hit*) NMXBenchmarkTest
79.80 ms	9.6%	2.00 ms	▶CenterMatcher::match(bool) NMXBenchmarkTest
7.20 ms	0.8%	1.30 ms	▶EventAnalyzer::analyze(Event&) const NMXBenchmarkTest
1.30 ms	0.1%	0 s	▶<Unknown Address>
900.00 µs	0.1%	400.00 µs	▶Event::both_planes() const NMXBenchmarkTest
500.00 µs	0.0%	500.00 µs	szone_free_definite_size libsystem_malloc.dylib
300.00 µs	0.0%	300.00 µs	DYLD-STUB\$\$\$operator delete(void*) NMXBenchmarkTest
100.00 µs	0.0%	0 s	▶benchmark::State::end() NMXBenchmarkTest
100.00 µs	0.0%	100.00 µs	szone_free_definite_size libsystem_malloc.dylib
100.00 µs	0.0%	0 s	▶<Unknown Address>
400.00 µs	0.0%	0 s	▶benchmark::ConsoleReporter::ReportContext(benchmark::BenchmarkReporter::Context const&) NMXBenchmarkTest
400.00 µs	0.0%	0 s	▶benchmark::internal::FindBenchmarksInternal(std::__1::basic_string_view<char>, benchmark::internal::Benchmark::Instance const*
100.00 µs	0.0%	0 s	▶benchmark::BenchmarkReporter::Context::Context() NMXBenchmarkTest
100.00 µs	0.0%	0 s	▶0x106c574e7 NMXBenchmarkTest
33.60 ms	4.0%	0 s	▶_dyld_start dyld

Input Filter:   Call Tree Constraints Data Mining

Context menu for 'void BenchmarkLoopPaused<NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const':

- Prune 'void BenchmarkLoopPaused<NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const' (highlighted)
- Change 'void BenchmarkLoopPaused<NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const' to callers
- Flatten 'NMXBenchmarkTest' to boundary frames
- Focus on subtree
- Focus on calls made by 'void BenchmarkLoopPaused<NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const'
- Focus on callers of 'void BenchmarkLoopPaused<NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const'
- Focus on calls made by 'NMXBenchmarkTest'
- Focus on callers of 'NMXBenchmarkTest'
- Reveal in Xcode
- Locate dSYM...
- Download dSYM
- Load dSYM



# macOS Instruments: CPU time sampler

Time Profiler > Profile > Root

Weight	Self Weight	Symbol Name
405.40 ms	100.0%	0 s ▼NMXBenchmarkTest (79255)
405.40 ms	100.0%	0 s ▼Main Thread 0x1ebd9d
371.80 ms	91.7%	0 s ▼start libdyld.dylib
371.80 ms	91.7%	0 s ▼main NMXBenchmarkTest
371.80 ms	91.7%	0 s ▼benchmark::RunSpecifiedBenchmarks(benchmark::BenchmarkReporter*, benchmark::BenchmarkReporter*) NMXBenchmarkTest
370.80 ms	91.4%	0 s ▼benchmark::internal::(anonymous namespace)::RunInThread(benchmark::internal::Benchmark::Instance const*, unsigned long, int, benchr
370.80 ms	91.4%	100.00 μs ▼benchmark::Fixture::Run(benchmark::State&) NMXBenchmarkTest
370.50 ms	91.3%	0 s ▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&) NMXBenchmarkTest
370.50 ms	91.3%	0 s ▼void BenchmarkLoop<NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0>(benchmark::State&, Nm
370.40 ms	91.3%	500.00 μs ▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator() const NMXBenchmarkTest
279.90 ms	69.0%	0 s ▼_cluster_plane(std::_1::vector
272.70 ms	67.2%	1.00 ms ►GapClusterer::cluster(std::
3.60 ms	0.8%	0 s ►GapClusterer::flush() NMXB
3.20 ms	0.7%	0 s ►sort_chronologically(std::_1
300.00 μs	0.0%	0 s ►<Unknown Address>
100.00 μs	0.0%	100.00 μs szone_free_definite_size
79.80 ms	19.6%	2.00 ms ►CenterMatcher::match(bool)
7.20 ms	1.7%	1.30 ms ►EventAnalyzer::analyze(Event&
1.30 ms	0.3%	0 s ►<Unknown Address>
900.00 μs	0.2%	400.00 μs ►Event::both_planes() const N
500.00 μs	0.1%	500.00 μs szone_free_definite_size libsy
300.00 μs	0.0%	300.00 μs DYLD-STUB\$\$\$operator delete
100.00 μs	0.0%	0 s ►benchmark::State::end() NMXB
100.00 μs	0.0%	100.00 μs szone_free_definite_size libsystem
100.00 μs	0.0%	0 s ►<Unknown Address>
400.00 μs	0.0%	0 s ►benchmark::ConsoleReporter::ReportContext(benchmark::BenchmarkReporter::Context const&) NMXBenchmarkTest
400.00 μs	0.0%	0 s ►benchmark::internal::FindBenchmarksInternal(std::_1::basic_string<char, std::_1::char_traits<char>, std::_1::allocator<char> > const&

Input Filter   Call Tree Constraints

- Charge 'NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator() const NMXBenchmarkTest
- Prune 'NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator() const NMXBenchmarkTest
- Charge 'NMXBenchmarkTest' to callers
- Flatten 'NMXBenchmarkTest' to boundary frames
- Focus on subtree**
- Focus on calls made by 'NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator() const NMXBenchmarkTest
- Focus on callers of 'NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator() const NMXBenchmarkTest
- Focus on calls made by 'NMXBenchmarkTest'
- Focus on callers of 'NMXBenchmarkTest'
- Reveal in Xcode
- Locate dSYM...
- Download dSYM
- Load dSYM





# macOS Instruments: CPU time sampler

Time Profiler > Profile > Root > NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const

Weight	Self Weight	Symbol Name
370.40 ms 100.0%	500.00 μs	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$0::operator>()() const NMXBenchmarkTest
279.90 ms 75.5%	0 s	▶_cluster_plane(std::__1::vector<Hit, std::__1::allocator<Hit> >&, std::__1::shared_ptr<AbstractClusterer>, std::__1::shared_ptr<AbstractMatcher>
272.70 ms 73.6%	1.00 ms	▶GapClusterer::cluster(std::__1::vector<Hit, std::__1::allocator<Hit> > const&) NMXBenchmarkTest
3.60 ms 0.9%	0 s	▶GapClusterer::flush() NMXBenchmarkTest
3.20 ms 0.8%	0 s	▶sort_chronologically(std::__1::vector<Hit, std::__1::allocator<Hit> >&) NMXBenchmarkTest
300.00 μs 0.0%	0 s	▶<Unknown Address>
100.00 μs 0.0%	100.00 μs	szone_free_definite_size libsystem_malloc.dylib
79.80 ms 21.5%	2.00 ms	▶CenterMatcher::match(bool) NMXBenchmarkTest
7.20 ms 1.9%	1.30 ms	▶EventAnalyzer::analyze(Event&) const NMXBenchmarkTest
1.30 ms 0.3%	0 s	▶<Unknown Address>
900.00 μs 0.2%	400.00 μs	▶Event::both_planes() const NMXBenchmarkTest
500.00 μs 0.1%	500.00 μs	szone_free_definite_size libsystem_malloc.dylib
300.00 μs 0.0%	300.00 μs	DYLD-STUB\$\$operator delete(void*) NMXBenchmarkTest

Input Filter  [Call Tree](#) Call Tree Constraints [Data Mining](#)

# macOS Instruments: CPU time sampler



Time Profiler > Profile > Root > NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&)::\$\_0::operator>() const

Weight	Self Weight	Symbol Name
370.40 ms 100.0%	500.00 μs	▼NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&)::\$_0::operator>() const NMXBenchmarkTest
279.90 ms 75.5%	0 s	▼_cluster_plane(std::_1::vector<Hit, std::_1::allocator<Hit> >&, std::_1::shared_ptr<AbstractClusterer>, std::_1::shared_ptr<AbstractMatcher>
272.70 ms 73.6%	1.00 ms	▼GapClusterer::cluster(std::_1::vector<Hit, std::_1::allocator<Hit> > const&) NMXBenchmarkTest
270.40 ms 73.0%	6.00 ms	▼GapClusterer::insert(Hit const&) NMXBenchmarkTest
261.10 ms 70.4%	500.00 μs	▼GapClusterer::flush() NMXBenchmarkTest
259.20 ms 69.9%	5.20 ms	▼GapClusterer::cluster_by_coordinate() NMXBenchmarkTest
187.80 ms 50.7%	18.00 ms	▼Cluster::insert(Hit const&) NMXBenchmarkTest
168.00 ms 45.3%	1.20 ms	▼std::_1::vector<Hit, std::_1::allocator<Hit> >::push_back(Hit const&) NMXBenchmarkTest
166.50 ms 44.9%	400.00 μs	▼void std::_1::vector<Hit, std::_1::allocator<Hit> >::_push_back_slow_path<Hit const&>(Hit const&) NMXBenchmarkTest
98.90 ms 26.7%	0 s	▼std::_1::_split_buffer<Hit, std::_1::allocator<Hit>&>::_split_buffer(unsigned long, unsigned long, std::_1::allocator<Hit>&) NM
98.90 ms 26.7%	1.10 ms	▼std::_1::_split_buffer<Hit, std::_1::allocator<Hit>&>::_split_buffer(unsigned long, unsigned long, std::_1::allocator<Hit>&) NI
97.80 ms 26.4%	0 s	▼std::_1::allocator_traits<std::_1::allocator<Hit> >::allocate(std::_1::allocator<Hit>&, unsigned long) NMXBenchmarkTest
97.80 ms 26.4%	400.00 μs	▼std::_1::allocator<Hit>::allocate(unsigned long, void const*) NMXBenchmarkTest
97.40 ms 26.2%	300.00 μs	▼std::_1::libcpp_allocate(unsigned long, unsigned long) NMXBenchmarkTest
96.90 ms 26.1%	3.30 ms	▶operator new(unsigned long) libc++abi.dylib
200.00 μs 0.0%	200.00 μs	DYLD-STUB\$\$malloc libc++abi.dylib
59.70 ms 16.1%	0 s	▶std::_1::_split_buffer<Hit, std::_1::allocator<Hit>&>::~~_split_buffer() NMXBenchmarkTest
4.40 ms 1.1%	0 s	▶std::_1::vector<Hit, std::_1::allocator<Hit> >::_swap_out_circular_buffer(std::_1::_split_buffer<Hit, std::_1::allocator<Hit>&>&)
1.40 ms 0.3%	1.00 ms	▶std::_1::vector<Hit, std::_1::allocator<Hit> >::_recommend(unsigned long) const NMXBenchmarkTest
1.20 ms 0.3%	0 s	▶void std::_1::allocator_traits<std::_1::allocator<Hit> >::construct<Hit, Hit const&>(std::_1::allocator<Hit>&, Hit*, Hit const&) NM
500.00 μs 0.1%	500.00 μs	std::_1::vector<Hit, std::_1::allocator<Hit> >::size() const NMXBenchmarkTest
300.00 μs 0.0%	0 s	▶void std::_1::allocator_traits<std::_1::allocator<Hit> >::construct<Hit, Hit const&>(std::_1::allocator<Hit>&, Hit*, Hit const&) NM)
700.00 μs 0.1%	0 s	▶unsigned short const& std::_1::min<unsigned short>(unsigned short const&, unsigned short const&) NMXBenchmarkTest
600.00 μs 0.1%	600.00 μs	std::_1::vector<Hit, std::_1::allocator<Hit> >::size() const NMXBenchmarkTest
300.00 μs 0.0%	0 s	▶unsigned short const& std::_1::max<unsigned short>(unsigned short const&, unsigned short const&) NMXBenchmarkTest
200.00 μs 0.0%	200.00 μs	std::_1::vector<Hit, std::_1::allocator<Hit> >::empty() const NMXBenchmarkTest
45.30 ms 12.2%	1.00 ms	▶AbstractClusterer::stash_cluster(Cluster&) NMXBenchmarkTest

Input Filter: Involves Symbol Call Tree Call Tree Constraints Data Mining

# macOS Instruments: CPU time sampler



Time Profiler > Profile > Root > Cluster::insert(Hit const&)

NMXBenchmarkTest

```
43 DebugSplitOptimizer();
44
45 // If plane identities don't match, invalidate
46 if (plane_ != e.plane) {
47     plane_ = Hit::InvalidPlane;
48     // For now it's decided to not discard everything in this case
49     // clear();
50 }
51
52 DebugSplitOptimizer();
53
54 hits.push_back(e);| 89.46%
55
56 DebugSplitOptimizer();
57
58 // TODO can we avoid converting to doubles and stay in uint64?
59 double weight64 = static_cast<double>(e.weight); 0.37%
60 double coordinate64 = static_cast<double>(e.coordinate); 0...
61 double time64 = static_cast<double>(e.time); 1.06%
62
63 DebugSplitOptimizer();
64
65 double We = weight64;
66 double We2 = weight64 * weight64;
67 double WeCo = weight64 * coordinate64; 1.01%
```

153	+0x23f	popq	%r12	0.32%
154	+0x241	popq	%r13	0.05%
155	+0x243	popq	%r14	0.05%
156	+0x245	popq	%r15	0.21%
157	+0x247	popq	%rbp	0.11%
158	+0x248	retq		0.11%
159	+0x249	xorl	%r15d, %r15d	
160	+0x24c	xorl	%eax, %eax	
161	+0x24e	jmp	"std::__1::__split_buffer<Hit, std::__1::allocator<Hit>&::__split_buffer(unsigned long, unsigned long, std::__1::allocator<Hit>&)+0x28754"	
162	+0x253	movq	%rbx, %rdi	
163	+0x256	callq	"DYLD-STUB\$\$std::__1::__vector_base_common<true>::__throw_length_error() const"	
164				

Cluster.cpp, 1 Line 1680 Samples

Cluster::insert(Hit const&), 57 Lines, 0 Instructions, 1685 Samples

Input Filter  [Call Tree](#) Call Tree Constraints [Data Mining](#)

Needs cmake target RelWithDebInfo to find source



# macOS Instruments: CPU time sampler

## Debugging optimizations

Try prevent optimizer from reordering sections of code

Need to understand generated code from individual sections

```
__attribute__((noinline)) void DebugSplitOptimizer() {  
    asm volatile("" : : : "memory");  
}
```

Code taken from Google Benchmark: `::benchmark::ClobberMemory()`

[Article](#) by Mike Action

# macOS Instruments: CPU time sampler



Time Profiler > Profile > Root > NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&>::\$\_0::operator>() const

Weight	Self Weight	Symbol Name	
370.40 ms	100.0%	500.00 μs	▼ NmxBenchmarkTest_Dummy_Benchmark::BenchmarkCase(benchmark::State&>::\$_0::operator>() const NMXBenchmarkTest
279.90 ms	75.5%	0 s	▶ _cluster_plane(std::__1::vector<Hit, std::__1::allocator<Hit> >&, std::__1::shared_ptr<AbstractClusterer>, std::__1::shared_ptr<AbstractMatcher>
79.80 ms	21.5%	2.00 ms	▶ CenterMatcher::match(bool) NMXBenchmarkTest
7.20 ms	1.9%	1.30 ms	▶ EventAnalyzer::analyze(Event&) const NMXBenchmarkTest
1.30 ms	0.3%	0 s	▶ <Unknown Address>
900.00 μs	0.2%	400.00 μs	▶ Event::both_planes() const NMXBenchmarkTest
500.00 μs	0.1%	500.00 μs	szone_free_definite_size libsystem_malloc.dylib
300.00 μs	0.0%	300.00 μs	DYLD-STUB\$\$operator delete(void*) NMXBenchmarkTest

Separate by State  
 Separate by Thread  
 Invert Call Tree  
 Hide System Libraries  
 Flat  
 Top Functions

Input Filter  **Call Tree** Call Tree Constraints



# macOS Instruments: CPU time sampler



Time Profiler > Profile > Root > NmxBenchmarkTest\_Dummy\_Benchmark::BenchmarkCase(benchmark::State&)::\$\_0::operator>() const

Weight	Self Weight	Symbol Name
77.30 ms	20.8%	77.30 ms ▶tiny_malloc_should_clear libsystem_malloc.dylib
33.90 ms	9.1%	33.90 ms ▶free_tiny libsystem_malloc.dylib
23.80 ms	6.4%	23.80 ms ▶tiny_malloc_from_free_list libsystem_malloc.dylib
19.10 ms	5.1%	19.10 ms ▶Cluster::insert(Hit const&) NMXBenchmarkTest
17.30 ms	4.6%	17.30 ms ▶tiny_free_no_lock libsystem_malloc.dylib
16.50 ms	4.4%	16.50 ms ▶tiny_free_list_add_ptr libsystem_malloc.dylib
14.60 ms	3.9%	14.60 ms ▶tiny_size libsystem_malloc.dylib
9.00 ms	2.4%	9.00 ms ▶free libsystem_malloc.dylib
7.80 ms	2.1%	7.80 ms ▶set_tiny_meta_header_in_use libsystem_malloc.dylib
7.20 ms	1.9%	7.20 ms ▶malloc_zone_malloc libsystem_malloc.dylib
6.70 ms	1.8%	6.70 ms ▶Cluster::Cluster(Cluster&&) NMXBenchmarkTest
6.60 ms	1.7%	6.60 ms ▶szone_malloc_should_clear libsystem_malloc.dylib
6.30 ms	1.7%	6.30 ms ▶GapClusterer::insert(Hit const&) NMXBenchmarkTest
5.80 ms	1.5%	5.80 ms ▶default_zone_malloc libsystem_malloc.dylib
5.20 ms	1.4%	5.20 ms ▶GapClusterer::cluster_by_coordinate() NMXBenchmarkTest
5.10 ms	1.3%	5.10 ms ▶void std::_1::_sort<sort_by_increasing_coordinate(std::_1::vector<Hit, std::_1::allocator<Hit> >&):'lambda'(Hit const&, Hit const&)&, Hit*>(Hit*
4.90 ms	1.3%	4.90 ms ▶Cluster::operator=(Cluster&&) NMXBenchmarkTest
4.90 ms	1.3%	4.90 ms ▶std::_1::enable_if<(is_move_constructible<Hit>::value) && (is_move_assignable<Hit>::value), void>::type std::_1::swap<Hit>(Hit&, Hit&) NMXBe
4.60 ms	1.2%	4.60 ms ▼operator new(unsigned long) libc++abi.dylib
4.30 ms	1.1%	4.30 ms ▶std::_1::_libcxx_allocate(unsigned long, unsigned long) NMXBenchmarkTest
300.00 μs	0.0%	300.00 μs ▶<Unknown Address>
4.60 ms	1.2%	4.60 ms ▼malloc libsystem_malloc.dylib
4.60 ms	1.2%	4.60 ms ▼operator new(unsigned long) libc++abi.dylib
4.60 ms	1.2%	4.60 ms ▶std::_1::_libcxx_allocate(unsigned long, unsigned long) NMXBenchmarkTest
4.50 ms	1.2%	4.50 ms ▶std::_1::_list_iterator<Cluster, void*>::operator++() NMXBenchmarkTest
4.30 ms	1.1%	4.30 ms ▶Cluster::empty() const NMXBenchmarkTest
4.30 ms	1.1%	4.30 ms ▶Cluster::time_end() const NMXBenchmarkTest

Input Filter  [Call Tree](#) Call Tree Constraints [Data Mining](#)



# macOS Instruments: CPU time sampler

## Conclusion

Simple tool to setup and use for iterations

Wrote custom allocator for `std::vector` and `std::list` → 5x perf

2

## macOS Instruments: Viewing transient allocations



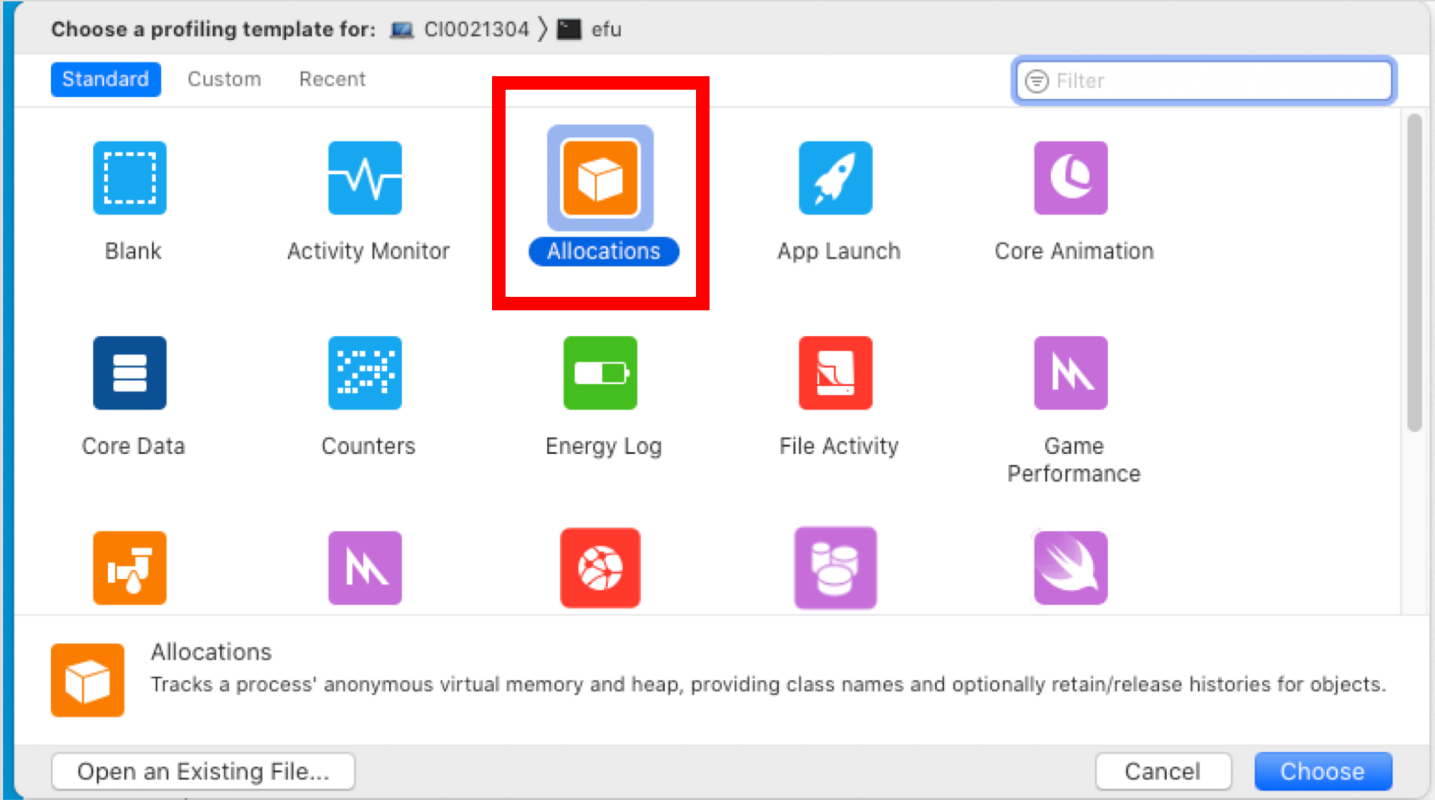


# macOS Instruments: Transient allocations

A simple way to view short-lived allocations

Useful when knowing you spend time on allocations

# macOS Instruments: Transient allocations



# macOS Instruments: Transient allocations



**Choose Target:**

Executables

- AdcFileStreamer
- AdcIntegrationTestConfig.ini
- AdcSimulator
- efu
- Executing artefacts.md
- mbgen\_hits
- mggen\_readouts
- nmxgen\_readouts
- udpgen\_hits
- udpgen\_jalousie
- udpgen\_jalousie2
- udpgen\_loki
- udpgen\_pcap

Name: efu  
Size: 22,55 MIB  
Created: Tuesday, 18 August 2020 at  
Modified: Tuesday, 18 August 2020 at

Environment Variable	Value

Show Hidden Files  Traverse Packages

```
-d AdcReadout --nohcheck -g dmsc-services01.cslab.ess.lu.se -r 10
```

/Users/mortenhilkerskaaning/dev/event-formation-unit/build

Cancel Choose

**Can't attach, must start target**

# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window. A red box highlights the recording options in the top-left corner. A black callout box with white text says "Hold → 'Recording Options'". The main area displays the "Allocations" instrument, which is currently set to "No Runs". Below this, the "Allocation Summary" table is visible, showing columns for Persistent, # Persistent, # Transient, Total Bytes, # Total, and Persistent/Total Bytes. The table is currently empty. The right side of the window shows "No Detail".

Graph	Category	Persistent	# Persistent	# Transient	Total Bytes	# Total	Persistent/Total Bytes



# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window with the 'Allocations' instrument selected. A configuration dialog is open, showing options for the instrument. The 'Global Options' section is highlighted with a red box, indicating the settings for transient allocations.

Options for: Allocations

All Allocations

- Discard unrecorded data upon stop
- Discard events for freed memory
- Only track VM allocations

Heap Allocations

- Record reference counts
- Identify virtual C++ objects
- Enable NSZombie detection

Recorded Types

Type String	Search	Action
<input checked="" type="checkbox"/> *	is Contain...	Record
<input type="checkbox"/> NS	is Prefix	Ignore
<input type="checkbox"/> CF	is Prefix	Ignore
<input type="checkbox"/> Malloc	is Prefix	Ignore

+ - Rules are evaluated top-to-bottom; later rules override earlier ones.

Global Options:

Time limit:  seconds

Recording Mode:

- Immediate
- Deferred
- Last  seconds

Close Record



# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window with the 'Allocations' instrument selected. The 'Allocation Summary' table is displayed, showing various memory allocation categories and their statistics. A red box highlights the 'Created & Persistent' filter option in the 'Mark Generation' dropdown menu.

Graph	Category	Persistent	# Persistent	# Transient	Total Bytes	# Total	Persistent/Total Bytes
<input type="checkbox"/>	All Heap Allocations	704,50 KiB	3,075	6,030	819,27 KiB	10,105	70%
<input type="checkbox"/>	All Anonymous VM	2,56 MiB	5	12	7,02 MiB	17	15%
<input type="checkbox"/>	VM: Stack	2,56 MiB	5	8	6,64 MiB	13	20%
<input type="checkbox"/>	Malloc 320 Bytes	99,38 KiB	318	30	108,75 KiB	348	31%
<input type="checkbox"/>	Malloc 52,00 KiB	52,00 KiB	1	0	52,00 KiB	1	100%
<input type="checkbox"/>	Malloc 64 Bytes	34,81 KiB	557	86	40,19 KiB	643	54%
<input type="checkbox"/>	Malloc 16,00 KiB	32,00 KiB	2	0	32,00 KiB	2	100%
<input type="checkbox"/>	Malloc 96 Bytes	30,09 KiB	321	54	35,16 KiB	375	80%
<input type="checkbox"/>	Malloc 32 Bytes	29,81 KiB	954	4,447	168,78 KiB	5,401	55%
<input type="checkbox"/>	Malloc 9,50 KiB	19,00 KiB	2	1	28,50 KiB	3	67%
<input type="checkbox"/>	Malloc 144 Bytes	15,89 KiB	113	286	56,11 KiB	399	29%
<input type="checkbox"/>	Malloc 112 Bytes	14,98 KiB	137	30	18,27 KiB	167	82%
<input type="checkbox"/>	Malloc 1,50 KiB	10,50 KiB	7	1	12,00 KiB	8	88%
<input type="checkbox"/>	UDPCClient	10,00 KiB	1	0	10,00 KiB	1	100%
<input type="checkbox"/>	Malloc 16 Bytes	9,72 KiB	622	22	10,06 KiB	644	96%
<input type="checkbox"/>	Malloc 288 Bytes	6,47 KiB	23	20	12,09 KiB	43	56%
<input type="checkbox"/>	Malloc 4,50 KiB	4,50 KiB	1	1	9,00 KiB	2	50%
<input type="checkbox"/>	Malloc 2,00 KiB	4,00 KiB	2	0	4,00 KiB	2	100%
<input type="checkbox"/>	Malloc 48 Bytes	3,94 KiB	84	343	20,02 KiB	427	20%
<input type="checkbox"/>	std::_1::_function::_fu...	3,28 KiB	105	0	3,28 KiB	105	100%
<input type="checkbox"/>	Malloc 832 Bytes	3,25 KiB	4	1	4,06 KiB	5	80%
<input type="checkbox"/>	Malloc 2,50 KiB	2,50 KiB	1	1	5,00 KiB	2	50%
<input type="checkbox"/>	Malloc 304 Bytes	2,38 KiB	8	8	4,75 KiB	16	50%
<input type="checkbox"/>	Malloc 1,00 KiB	2,00 KiB	2	0	2,00 KiB	2	100%
<input type="checkbox"/>	Malloc 80 Bytes	1,64 KiB	21	817	65,47 KiB	838	20%
<input type="checkbox"/>	std::_1::_shared_ptr_e...	1,50 KiB	1	0	1,50 KiB	1	100%
<input type="checkbox"/>	asio::detail::scheduler	1,06 KiB	4	0	1,06 KiB	4	100%
<input type="checkbox"/>	std::_1::_function::_fu...	1,06 KiB	34	0	1,06 KiB	34	100%
<input type="checkbox"/>	Log::GraylogConnection:...	992 Bytes	1	0	992 Bytes	1	100%
<input type="checkbox"/>	Malloc 256 Bytes	768 Bytes	3	1	1,00 KiB	4	75%
<input type="checkbox"/>	Log::ConsoleInterface	752 Bytes	1	0	752 Bytes	1	100%
<input type="checkbox"/>	Malloc 128 Bytes	640 Bytes	5	50	512 Bytes	55	93%
<input type="checkbox"/>	Malloc 272 Bytes	544 Bytes	2	3	768 Bytes	5	40%
<input type="checkbox"/>	Malloc 528 Bytes	528 Bytes	1	34	528 Bytes	35	3%

Select "Created & Destroyed"

# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window. The top track is 'All Tracks' with a time range from 00:00:000 to 00:04:000. Below it, the 'Allocations' track is selected, showing a blue bar representing heap allocations. A red box highlights the initial phase of the run. A black box with a red arrow points to this phase, containing the text 'Exclude startup phase as it's irrelevant'. Below the tracks, the 'Allocation Summary' table is visible.

Graph	Category	Persistent	# Persistent	# Transient	Total Bytes	# Total	Persistent/Total Bytes
<input checked="" type="checkbox"/>	All Heap & Anonymous VM	0 Bytes	0	28	1,44 KiB	28	
<input type="checkbox"/>	All Heap Allocations	0 Bytes	0	28	1,44 KiB	28	
<input type="checkbox"/>	Malloc 48 Bytes	0 Bytes	0	20	960 Bytes	20	
<input type="checkbox"/>	Malloc 64 Bytes	0 Bytes	0	8	512 Bytes	8	

# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window. At the top, it says "Start a deferred mode recording" and "Run 1 of 1 | 00:00:05". Below that is a "Track Filter" set to "All Tracks". The main area shows a timeline with a large blue bar representing memory allocations. On the left, there are instrument panels for "Allocations" and "VM Tracker". A red box highlights the "Call Trees" menu item in the "Allocations" panel, with a black callout box containing the text "Select 'Call Trees'". Below this, a table lists the call stack for the selected allocation:

Bytes	Count	Symbol Name
1.44 KB	100.0%	28 start libdyld.dylib
1.44 KB	100.0%	28 main efu
1.44 KB	100.0%	28 StatPublisher::publish(std::__1::shared_ptr<Detector>, Statistics&) efu
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > fmt::v6::format<char [10], std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > > fmt::v6::internal::vformat<char>
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > fmt::v6::to_string<char, 500ul
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > ::basic_string(char const*, un
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > ::basic_string(char const*, u
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > ::__init(char const*, unsi
1.44 KB	100.0%	28 operator new(unsigned long) libc++abi.dylib

On the right side, the "Heaviest Stack Trace" panel shows a list of stack frames, each with a size of 1.44 KB:

- start
- main
- StatPublisher::publish(std::\_\_1::shared\_ptr<Detector>, Statistics&)
- std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > fmt::v6::format<char [10], std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > > fmt::v6::internal::vformat<char>
- std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > fmt::v6::to\_string<char, 500ul
- std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > ::basic\_string(char const\*, un
- std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > ::basic\_string(char const\*, u
- std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > ::\_\_init(char const\*, unsi
- operator new(unsigned long)
- malloc
- malloc\_zone\_malloc

# macOS Instruments: Transient allocations



The screenshot shows the Instruments application window. At the top, the process is identified as 'efu' (CI0021304) and the run is 'Run 1 of 1' with a duration of '00:00:05'. The 'Allocations' instrument is selected, showing a large blue bar representing memory usage over time. The 'Call Trees' panel is expanded to show the 'Call Tree' view, which lists the following entries:

Bytes Used	Count	Symbol Name
1.44 KB	100.0%	28 start libdyld.dylib
1.44 KB	100.0%	28 main efu
1.44 KB	100.0%	28 StatPublisher::publish(std::__1::shared_ptr<Detector>, Statistics&) efu
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > fmt::v6::format<char [10], std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > fmt::v6::internal::vformat<char>
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > fmt::v6::to_string<char, 500ul
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >::basic_string(char const*, un
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >::basic_string(char const*, t
1.44 KB	100.0%	28 std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >::init(char const*, unsgi
1.44 KB	100.0%	28 operator new(unsigned long) libc++abi.dylib

The 'Heaviest Stack Trace' panel on the right shows the following entries:

- 1.44 KB start
- 1.44 KB main
- 1.44 KB StatPublisher::publish(std::\_\_1::shared\_ptr<Detector>,
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>.
- 1.44 KB operator new(unsigned long)
- 1.44 KB malloc
- 1.44 KB malloc\_zone\_malloc

# macOS Instruments: Transient allocations



Allocations > Call Trees > Call Tree > StatPublisher::publish(std::\_1::shared\_ptr<Detector>, Statistics&)

```
14 StatPublisher::StatPublisher(std::string IP, int Port)
15   : IPAddress(IP), TCPPort(Port) {
16   if (not Socket::isValidIp(IPAddress)) {
17     IPAddress = Socket::getHostByName(IPAddress);
18   }
19   StatDb.reset(new TCPTransmitter(IPAddress.c_str(), TCPPort));
20 }
21
22 ///
23 void StatPublisher::publish(std::shared_ptr<Detector> DetectorPtr,
24                             Statistics &OtherStats) {
25     int unixtime = (int)time(NULL);
26
27     if (StatDb->isValidSocket()) {
28         for (int i = 1; i <= DetectorPtr->statsize(); i++) {
29             auto StatString = fmt::format("{} {} {}\n", DetectorPtr->statname(i),
30                                           DetectorPtr->statvalue(i), unixtime);
31
32             StatDb->senddata(StatString.c_str(), StatString.size());
33         }
34         for (size_t i = 1; i <= OtherStats.size(); i++) {
35             auto StatString = fmt::format("{} {} {}\n", OtherStats.name(i),
36                                           OtherStats.value(i), unixtime);
37
38             StatDb->senddata(StatString.c_str(), StatString.size());
39         }
40     } else {
41         handleReconnect();
42     }
43 }
44
45 ///
```

Annotations

- 86.96% auto StatString = fmt::format("{} {} {}\n", DetectorPtr->statname(i), DetectorPtr->statvalue(i), unixtime);
- 13.04% auto StatString = fmt::format("{} {} {}\n", OtherStats.name(i), OtherStats.value(i), unixtime);

Needs cmake target RelWithDebInfo to find source



# macOS Instruments: Transient allocations

## Conclusion

StatPublisher not likely a bottleneck, try fixed-size buffer for formatting

👍 Easy to map allocations to source code



3

Mention Callgrind +  
QCachegrind





# Mention Callgrind + QCachegrind

- Mac + Linux
- Full install guide & walkthrough [on Confluence](#)

The screenshot displays the Callgrind and QCachegrind analysis tool interface. It is divided into several panes:

- Top Cost Call Stack:** A table showing the most expensive calls. The top entry is `GapClusterer::insert(Hit const&)` with a cost of 50.35 and 61,920 calls.
- Flat Profile:** A table showing the cost of various ELF objects. `libsystem_malloc.dylib` is the most expensive at 71.71.
- Source Code:** A view of the source code for `Cluster::insert(Hit const&)`, showing a function that returns a `std::basic_stringbuf` object.
- Call Graph:** A hierarchical diagram showing the call graph for `std::_1::basic_stringbuf`. The root node is `std::_1::basic_stringbuf` (23.02%), which calls `_platform_memmove` (0.68%), `operator delete` (11.44%), and `operator new` (10.20%).

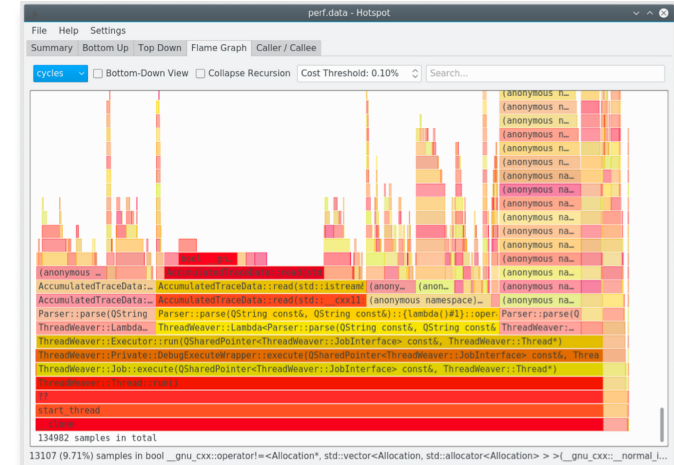
At the bottom, the status bar shows: `callgrind.out.90577 [1] - Total Cycle Estimation Cost: 135 335 578` and the date `2020-09-13`.



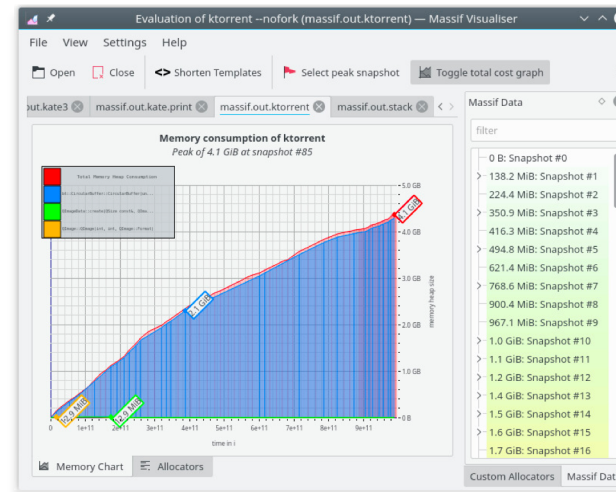
# Mention Callgrind + QCachegrind



- Callgrind is Mac+Linux, try dedicated Linux tools:
- [Hotspot](#) (perf)



- [Massif Visualizer](#) (mem)



- [Heaptrack](#) (mem)



4

# Understanding machine code using Ghidra





# Understanding machine code using Ghidra

Open-source binary reverse engineering tool by NSA

Tries to generate source code from machine code

Can be installed using brew, instructions [here](#) (also needs to install Java)

Create a project and import binary.

Names are mangled, but you can search using "\*" instead of ":", etc.

Case: generate() loop in AdcSimulator

# Understanding machine code using Ghidra



Instruments3

Run 3 of 4 | 00:00:05

Track Filter: All Tracks Instruments CPUs Threads

Time Profiler

Details > Profile > Root

Weight	Self Weight	Library Name	Symbol Name
4.88 s 100.0%	0 s		▼AdcSimulator (88024)
1.21 s 24.8%	1.21 s	libsystem_kernel.dylib	▶write
460.40 ms 9.4%	460.40 ms	libsystem_m.dylib	▶llround
379.10 ms 7.7%	379.10 ms	AdcSimulator	▶SampleRunGenerator::generate(double, TimeStamp) ↕
354.70 ms 7.2%	354.70 ms	libsystem_kernel.dylib	▶kevent
329.50 ms 6.7%	329.50 ms	libsystem_kernel.dylib	▶_sendmsg
284.80 ms 5.8%	284.80 ms	libsystem_kernel.dylib	▶read
161.60 ms 3.3%	161.60 ms	AdcSimulator	▶asio::executor::impl<asio::io_context::executor_type, std::_1::allocator<void> >::clone() const
160.40 ms 3.2%	160.40 ms	libsystem_thread.dylib	▶pthread_mutex_lock
160.20 ms 3.2%	160.20 ms	libsystem_thread.dylib	▶pthread_mutex_unlock
152.20 ms 3.1%	152.20 ms	libsystem_kernel.dylib	▶mach_absolute_time
133.40 ms 2.7%	133.40 ms	AdcSimulator	▶asio::executor::impl<asio::io_context::executor_type, std::_1::allocator<void> >::destroy()
84.50 ms 1.7%	84.50 ms	libsystem_platform.dylib	▶_platform_memmove\$VARIANT\$Haswell
64.40 ms 1.3%	64.40 ms	AdcSimulator	▶PoissonDelay::start()
55.30 ms 1.1%	55.30 ms	AdcSimulator	▼asio::detail::scheduler::do_run_one(asio::detail::conditionally_enabled_mutex::scoped_lock&, asio::detail::schedul

# Understanding machine code using Ghidra



Instruments3  
Run 3 of 4 | 00:00:05

Track Filter: All Tracks Instruments CPUs Threads

Time Profiler: Instrument

Details > Profile > Root > SampleRunGenerator::generate(double, TimeStamp)

```
75 std::for_each(
76     BeginIterator, EndIterator,
77     [&SampleCounter, BitPatternMultiplier, BitPattern](auto &Sample) {
78         std::uint16_t BitTester =
79             0x8000 >> (SampleCounter / BitPatternMultiplier);
80         if (BitTester & BitPattern) {
81             Sample = 0.99;
82         }
83         ++SampleCounter;
84     });
85 }
86
87 std::pair<void *, std::size_t>
88 SampleRunGenerator::generate(double Amplitude, TimeStamp const Time) {
89     HeaderPtr->TimeStamp = {Time.getSeconds(), Time.getSecondsFrac()};
90     HeaderPtr->TimeStamp.fixEndian();
91     for (auto i = 0u; i < NrOfSamples; ++i) {
92         SamplePtr[i] = htons(
93             std::lround(PeakBuffer[i] * Amplitude + BkgSlope * i + BkgOffset));
94     }
95     return std::make_pair(Buffer.get(), NrOfSamples * sizeof(std::uint16_t) +
96                             sizeof(DataHeader) + 4);
97 }
98
```

Address	Instruction	Comment	Percentage
30 +0x5d	movq	10(%r14), %r15	0.03%
31 +0x61	xorl	%ebx, %ebx	0.03%
32 +0x63	movsd	%xmm2, -64(%rbp)	0.08%
33 +0x68	nopl	(%rax,%rax)	
34 +0x70	movss	(%r15,%rbx,4), %xmm0	1.87%
35 +0x76	xorps	%xmm1, %xmm1	0.50%
36 +0x79	cvtss2sd	%xmm0, %xmm1	10.47%
37 +0x7d	movl	%ebx, %eax	3.53%
38 +0x7f	xorps	%xmm0, %xmm0	0.13%
39 +0x82	cvtsi2sdq	%rax, %xmm0	10.47%
40 +0x87	mulsd	%xmm2, %xmm1	3.56%
41 +0x8b	mulsd	-56(%rbp), %xmm0	1.00%
42 +0x90	addsd	%xmm1, %xmm0	10.71%
43 +0x94	addsd	-48(%rbp), %xmm0	6.41%
44 +0x99	callq	"DYLD-STUB\$\$lround"	13.80%
45 +0x9e	movsd	-64(%rbp), %xmm2	0.32%
46 +0xa3	rolw	\$8, %ax	11.55%
47 +0xa7	movw	%ax, (%r13,%rbx,2)	1.64%
48 +0xad	leal	1(%rbx), %ebx	19.05%
49 +0xb0	cmpq	%rbx, %r12	0.05%
50 +0xb3	ja	"SampleRunGenerator::generate(double, TimeStamp)+0x70"	
51 +0xb5	leaq	20(%r12,%r12), %rdx	0.13%
52 +0xba	jmp	"SampleRunGenerator::generate(double, TimeStamp)+0x70"	

# Understanding machine code using Ghidra



```
Listing: AdcSimulator - (69 addresses selected)
*AdcSimulator x

10004c53d 10 40 50 MOVSD qword ptr [RBP + local_40],param_1
          f2 0f 11 45 c8
10004c542 f2 41 0f MOVSD param_1,qword ptr [R14 + 0x38]
          10 46 38
10004c548 f2 0f 11 MOVSD qword ptr [RBP + local_38],param_1
          45 d0
10004c54d 4d 8b 6e 10 MOV R13,qword ptr [R14 + 0x10]
10004c551 31 db XOR EBX,EBX
10004c553 f2 0f 11 MOVSD qword ptr [RBP + local_48],XMM2
          55 c0
10004c558 0f 1f 84 NOP dword ptr [RAX + RAX*0x1]
          00 00 00
          00 00

LAB_10004c560 XREF[1]:
10004c560 f3 41 0f MOVSS param_1,dword ptr [R15 + RBX*0x4]
          10 04 9f
10004c566 0f 57 c9 XORPS XMM1,XMM1
10004c569 f3 0f 5a c8 CVTSS2SD XMM1,param_1
10004c56d 89 d8 MOV EAX,EBX
10004c56f 0f 57 c0 XORPS param_1,param_1
10004c572 f2 48 0f CVTSI2SD param_1,RAX
          2a c0
10004c577 f2 0f 59 ca MULSD XMM1,XMM2
10004c57b f2 0f 59 MULSD param_1,qword ptr [RBP + local_40]
          45 c8
10004c580 f2 0f 58 c1 ADDSD param_1,XMM1
10004c584 f2 0f 58 ADDSD param_1,qword ptr [RBP + local_38]
          45 d0
10004c589 e8 6c 11 CALL __stubs::_lround
          08 00
10004c58e f2 0f 10 MOVSD XMM2,qword ptr [RBP + local_48]
          55 c0
10004c593 66 c1 c0 08 ROL AX,0x8
10004c597 66 41 89 MOV word ptr [R13 + RBX*0x2],AX
          44 5d 00
10004c59d 8d 5b 01 LEA EBX,[RBX + 0x1]
10004c5a0 49 39 dc CMP R12,EBX
10004c5a3 77 bb JA LAB_10004c560
10004c5a5 4b 8d 54 LEA RDX,[R12 + R12*0x1 + 0x14]
          24 14
10004c5aa eb 05 JMP LAB_10004c5b1

LAB_10004c5ac XREF[1]:
```

```
Decompile: generate - (AdcSimulator)
1
2 /* SampleRunGenerator::generate(double, TimeStamp) */
3
4 undefined [16] __thiscall generate(SampleRunGenerator *this,double param_1,TimeStamp param_2)
5
6 {
7 double dVar1;
8 double dVar2;
9 uint uVar3;
10 uint uVar4;
11 ulong uVar5;
12 long lVar6;
13 ushort uVar7;
14 long lVar8;
15 ulong uVar9;
16 undefined7 in_register_00000031;
17
18 lVar8 = *(long *)(this + 8);
19 *(undefined4 *)(&lVar8 + 8) = (int)CONCAT71(in_register_00000031,param_2);
20 *(undefined4 *)(&lVar8 + 0xc) = (int)((uint7)in_register_00000031 >> 0x18);
21 lVar8 = *(long *)(this + 8);
22 uVar3 = *(uint *)(&lVar8 + 8);
23 uVar4 = *(uint *)(&lVar8 + 0xc);
24 *(uint *)(&lVar8 + 8) =
25     uVar3 >> 0x18 | (uVar3 & 0xff0000) >> 8 | (uVar3 & 0xff00) << 8 | uVar3 << 0x18;
26 *(uint *)(&lVar8 + 0xc) =
27     uVar4 >> 0x18 | (uVar4 & 0xff0000) >> 8 | (uVar4 & 0xff00) << 8 | uVar4 << 0x18;
28 uVar5 = *(ulong *)(&this + 0x18);
29 if (uVar5 == 0) {
30     lVar8 = 0x14;
31 }
32 else {
33     lVar8 = *(long *)(this + 0x48);
34     dVar1 = *(double *)(this + 0x30);
35     dVar2 = *(double *)(this + 0x38);
36     lVar6 = *(long *)(this + 0x10);
37     uVar9 = 0;
38     do {
39         uVar7 = _lround((double)uVar9 * dVar1 + (double)*(float *)(&lVar8 + uVar9 * 4) * param_1 +
40             dVar2);
41         *(ushort *)(&lVar6 + uVar9 * 2) = uVar7 << 8 | uVar7 >> 8;
42         uVar9 = (ulong)((int)uVar9 + 1);
43     } while (uVar9 < uVar5);
44     lVar8 = uVar5 * 2 + 0x14;
45 }
46 return CONCAT88(lVar8,*(undefined8 *)this);
```

Method call with 'this' parameter



# Understanding machine code using Ghidra



The image shows the Ghidra interface with two windows: 'Listing: AdcSimulator - (69 addresses selected)' and 'Decompile: generate - (AdcSimulator)'. The assembly window shows instructions from 10004c53d to 10004c5aa. The decompiled window shows the C++ code for the generate function, with a callout box for the `__m128d_mm_cvtsi32_sd` instruction.

**Assembly Listing (Selected Instructions):**

```
10004c53d f2 0f 11 MOVSD qword ptr [RBP + local_40], param_1
10004c542 f2 41 0f MOVSD param_1, qword ptr [R14 + 0x38]
10004c548 f2 0f 11 MOVSD qword ptr [RBP + local_38], param_1
10004c54d 4d 8b 6e 10 MOV R13, qword ptr [R14 + 0x10]
10004c551 31 db XOR EBX, EBX
10004c553 f2 0f 11 MOVSD qword ptr [RBP + local_48], XMM2
10004c558 0f 1f 84 NOP dword ptr [RAX + RAX*0x1]
10004c560 f3 41 0f MOVSS param_1, dword ptr [R15 + RBX*0x4]
10004c566 0f 57 c9 XORPS XMM1, XMM1
10004c569 f3 0f 5a c8 CVTSS2SD XMM1, param_1
10004c56d 89 d8 MOV EAX, EBX
10004c56f 0f 57 c0 XORPS param_1, param_1
10004c572 f2 48 0f CVTSI2SD param_1, RAX
10004c577 f2 0f 59 ca MULSD XMM1, XMM2
10004c57b f2 0f 59 MULSD param_1, qword ptr [RBP + local_40]
10004c580 f2 0f 58 c1 ADDSD param_1, XMM1
10004c584 f2 0f 58 ADDSD param_1, qword ptr [RBP + local_40]
10004c589 e8 6c 11 CALL __stubs::_round
10004c58e f2 0f 10 MOVSD XMM2, qword ptr [RBP + local_48]
10004c593 66 c1 c0 08 ROL AX, 0x8
10004c597 66 41 89 MOV word ptr [R13 + RBX*0x2], AX
10004c59d 8d 5b 01 LEA EBX, [RBX + 0x1]
10004c5a0 49 39 dc CMP R12, RBX
10004c5a3 77 bb JA LAB_10004c560
10004c5a5 4b 8d 54 LEA RDX, [R12 + R12*0x1 + 0x14]
10004c5aa eb 05 JMP LAB_10004c5b1
```

**Decompiled Code (Selected Snippets):**

```
/* SampleRunGenerator::generate(double, TimeStamp) */
undefined [16] __thiscall generate(SampleRunGenerator *this, double param_1, TimeStamp param_2)
{
    double dVar1;
    double dVar2;
    uint uVar3;
    uint uVar4;
    ulong uVar5;
    long lVar6;
    ushort uVar7;
    long lVar8;
    ulong uVar9;
    undefined7 in_regi...
    lVar8 = *(long *)...
    *(undefined4 *)...
    *(undefined4 *)...
    lVar8 = *(long *)...
    uVar3 = *(uint *)...
    uVar4 = *(uint *)...
    *(uint *)... =
        uVar3 >> 0x18 | (uVar3 & 0xff0000) >> 8 | (uVar3 & 0xff00) << 8 | uVar3 << 0x18;
    *(uint *)... =
        uVar4 >> 0x18 | (uVar4 & 0xff0000) >> 8 | (uVar4 & 0xff00) << 8 | uVar4 << 0x18;
    uVar5 = *(ulong *)...
    if (uVar5 == 0) {
        lVar8 = 0x14;
    }
    else {
        lVar8 = *(long *)...
        dVar1 = *(double *)...
        dVar2 = *(double *)...
        lVar6 = *(long *)...
        uVar9 = 0;
        do {
            uVar7 = _round((double)uVar9 * dVar1 + (double)*(float *)...
                dVar2);
            *(ushort *)... = uVar7 << 8 | uVar7 >> 8;
            uVar9 = (ulong)((int)uVar9 + 1);
        } while (uVar9 < uVar5);
        lVar8 = uVar5 * 2 + 0x14;
    }
    return CONCAT88(lVar8, *(undefined8 *)this);
}
```

**Callout Box:** Can't inline 😞

# Understanding machine code using Ghidra



## Conclusion

Let's disallow implicit numeric conversion

Can't predict what functions can be inlined

Implies we can't predict whether C++ compiler will vectorize code

Rewrote in ISPC to target 8 floats with AVX2 (fast enough)

5

Fast code with  
ISPC language +  
compiler





# Fast code with ISPC language + compiler

Explicit parallelism in an shader-like manner

Data-parallel, run N instances of the same function, on N sets of input arguments, using N SIMD lanes.

[Paper](#) (13 pages)

Example using 4 program instances

```
int f(int a, int b) {  
    if (a < 0)  
        a = 3;  
    else  
        a += b;  
    return a;  
}
```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```
int f(int a, int b) {  
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]  
  
    if (a < 0)  
        a = 3;  
  
    else  
        a += b;  
  
    return a;  
}
```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```
int f(int a, int b) {  
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]  
  
    if (a < 0)    m: [F, F, 0, 0]    ← a: [-2, -1, 1, 2]    <    [0, 0, 0, 0]  
  
        a = 3;  
  
    else  
  
        a += b;  
  
    return a;  
}
```



# Fast code with ISPC language + compiler



(Format) variable: [instance1, instance2, instance3, instance4]

```
int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0]    ← a: [-2, -1, 1, 2]    < [0, 0, 0, 0]

        a = 3;
        a: [-2, -1, 1, 2]    & m: [F, F, 0, 0]    = [3, 3, 3, 3]    & m: [F, F, 0, 0]

    else

        a += b;

    return a;
}
```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```

int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0]    ← a: [-2, -1, 1, 2]    <    [0, 0, 0, 0]

        a = 3;
            a: [-2, -1, 1, 2]    &    m: [F, F, 0, 0]    =    [3, 3, 3, 3]    &    m: [F, F, 0, 0]
            a: [3, 3, 1, 2]    ←            a: [-2, -1, _, _]            =            [3, 3, 0, 0]

    else

        a += b;

    return a;
}

```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```

int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0] ← a: [-2, -1, 1, 2] < [0, 0, 0, 0]

        a = 3;
            a: [-2, -1, 1, 2] & m: [F, F, 0, 0] = [3, 3, 3, 3] & m: [F, F, 0, 0]
            a: [3, 3, 1, 2] ← a: [-2, -1, _, _] = [3, 3, 0, 0]

    else
        m: [0, 0, F, F] ← ~ m: [F, F, 0, 0]

        a += b;

    return a;
}

```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```

int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0] ← a: [-2, -1, 1, 2] < [0, 0, 0, 0]

        a = 3;
        a: [-2, -1, 1, 2] & m: [F, F, 0, 0] = [3, 3, 3, 3] & m: [F, F, 0, 0]
        a: [3, 3, 1, 2] ← a: [-2, -1, _, _] = [3, 3, 0, 0]

    else
        m: [0, 0, F, F] ← ~ m: [F, F, 0, 0]

        a += b;
        a: [3, 3, 1, 2] & m: [0, 0, F, F] += b: [1, 2, 3, 4] & m: [0, 0, F, F]

    return a;
}

```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```

int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0] ← a: [-2, -1, 1, 2] < [0, 0, 0, 0]

        a = 3;
            a: [-2, -1, 1, 2] & m: [F, F, 0, 0] = [3, 3, 3, 3] & m: [F, F, 0, 0]
            a: [3, 3, 1, 2] ← a: [-2, -1, _, _] = [3, 3, 0, 0]

        else
            m: [0, 0, F, F] ← ~ m: [F, F, 0, 0]

            a += b;
                a: [3, 3, 1, 2] & m: [0, 0, F, F] += b: [1, 2, 3, 4] & m: [0, 0, F, F]
                a: [3, 3, 3, 4] ← a: [_, _, 1, 2] += [0, 0, 3, 4]

    return a;
}

```



# Fast code with ISPC language + compiler

(Format) variable: [instance1, instance2, instance3, instance4]

```

int f(int a, int b) {
    int a: [-2, -1, 0, 1]    int b: [1, 2, 3, 4]    int mask m: [_, _, _, _]

    if (a < 0)
        m: [F, F, 0, 0] ← a: [-2, -1, 1, 2] < [0, 0, 0, 0]

        a = 3;
            a: [-2, -1, 1, 2] & m: [F, F, 0, 0] = [3, 3, 3, 3] & m: [F, F, 0, 0]
            a: [3, 3, 1, 2] ← a: [-2, -1, _, _] = [3, 3, 0, 0]

        else
            m: [0, 0, F, F] ← ~ m: [F, F, 0, 0]

            a += b;
                a: [3, 3, 1, 2] & m: [0, 0, F, F] += b: [1, 2, 3, 4] & m: [0, 0, F, F]
                a: [3, 3, 3, 4] ← a: [_, _, 1, 2] += [0, 0, 3, 4]

    return a;
}

```





# Fast code with ISPC language + compiler

## Conclusion

- Example had 4 simultaneous instances, scales to 8, 16 .. N, GPU in future
- Useful for processing independent data “cells”
- Used for converting data when writing network packets (high speed)
- Hope to use to parse NMX packets, possibly process independent clusters?

6

# Viewing C++ projects with SourceTrail





# Viewing C++ projects with SourceTrail

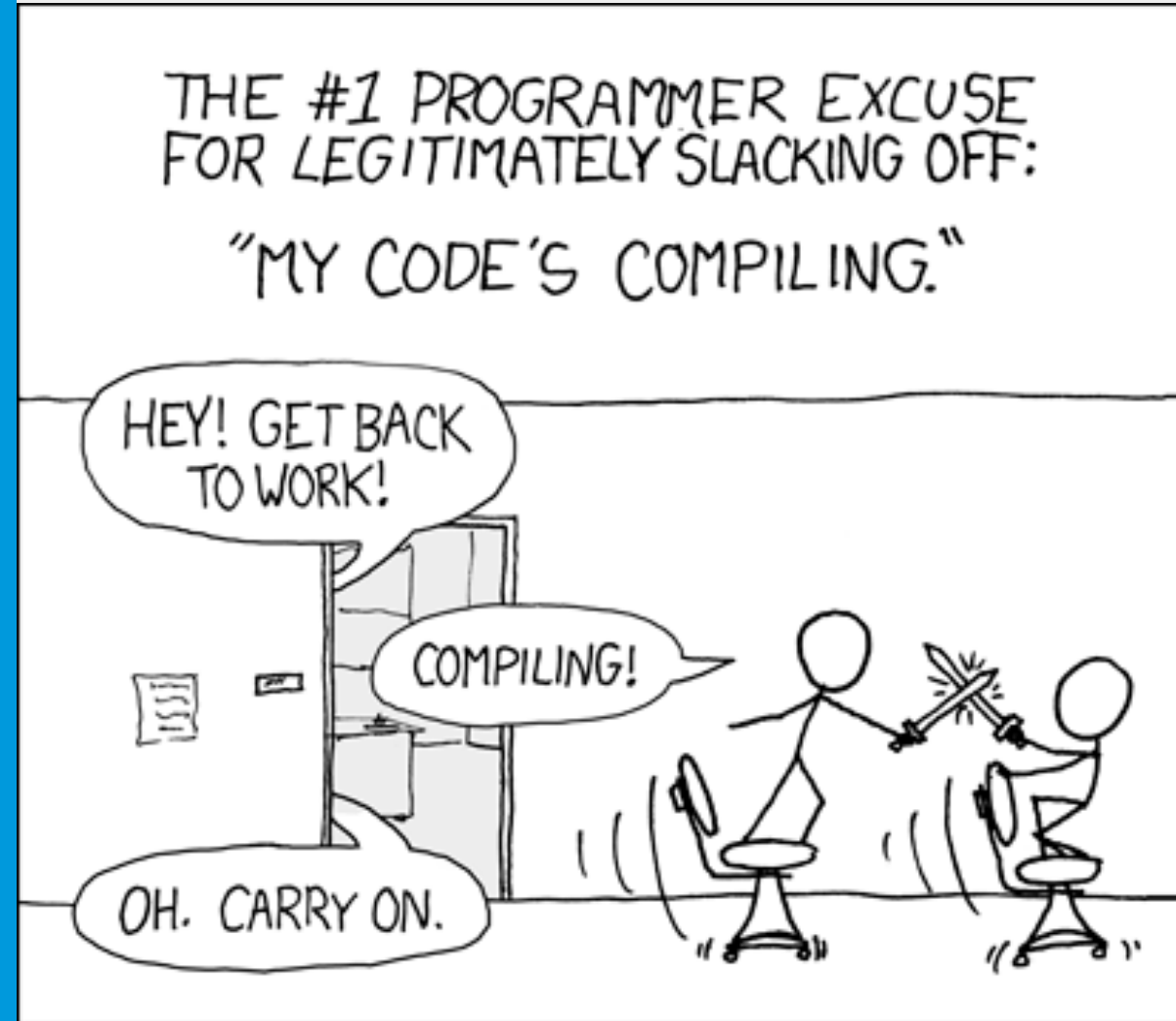
Open-source C++/Python code visualization tool [link](#)

Many ways of exploring symbols, implementations and references in a project

The screenshot displays the SourceTrail interface for a C++ project. On the left, a file tree shows the project structure with files like GdGem.cpp, GdGemBase.cpp, and GdGemBaseTest.cpp. The central pane shows a detailed view of GdGemBase.h, listing symbols such as MinNMXChannel, MaxNMXChannel, ZeroNMXOverlapSize, NMXSettings, and GdGemBase. On the right, a code editor shows the source code for GdGemBase.h, including copyright information, includes, and class definitions. The status bar at the bottom indicates 'No IDE connected' and '10 errors'.

7

# Viewing C++ compilation time





# Viewing C++ compilation time profiles

Speed up Jenkins turnaround time

[Guide to enable compile flags and run analysis](#)

Resulting data files can be visualized by <https://www.speedscope.app>

Results aggregated by [Clang Build Analyzer](#)

Case: EFUArgs.cpp, 255 lines, 22 sec to compile



# Viewing C++ compilation time profiles

Speed up Jenkins turnaround time

EFUArgs.cpp.json in <https://www.speedscope.app>



#include  
CLI.hpp  
1.5 sec

Instantiate templates for CLI and fmt  
7.0 sec

Optimize code for fmt  
9.7 sec





# Viewing C++ compilation time profiles

## Clang Builder Analyzer, highest scoring entries

Files **parse** (compiler frontend):

10466 ms: MultigridBase.cpp.o

Files **codegen** (compiler backend):

18433 ms: udpgen\_jalousie/generator.cpp.o

Templates **instantiate**:

15338 ms: fmt::v6::internal::vformat<char>

Templates sets **instantiate**:

171281 ms: std::\_\_1::function<\$>::function<\$>

Functions **compile**:

821 ms: CLI::App::\_parse\_arg(std::\_\_1::vector<...

Function sets **compile / optimize**:

115961 ms: void fmt::v6::internal::basic\_writer<\$>...

Expensive **headers**:

56290 ms: include/CLI/CLI.hpp (included 40 times, avg 1407 ms), included via ...



# Viewing C++ compilation time profiles

## Conclusion

Improve turnaround time by optimizing build time, it takes 15 min now, what in 3 years? 10?

Look into pre-defining “extern template” to recycle non-inlined code.

Look into Pimpl-like ways of hiding private members

Measure possible speed gain by creating mock macro definitions



# Finish presentation