



Cache Analysis with Callgrind

Code Optimization Workshop | 28 June 2022 | Josef Weidendorfer

Focus: CPU Cache Simulation using a Simple Machine Model



Why simulation? (in contrast to real measurement)

- Reproducability
- No influence of tool on results
- Allows to collect information not possible with real hardware
- No special permissions needed / cannot crash machine



Focus: CPU Cache Simulation using a Simple Machine Model



Why a simple machine model?

- easier to understand
- still captures most problems
- faster simulation

A sophisticated model includes

- All pipeline stages, Out-of-Order scheduling, speculation, instr. troughput & latency
- All cache layers, coherency protocol, replacement, memory parallelism, contention, hardware prefetching, exact interleaving of accesses from cores



Focus: CPU Cache Simulation using a Simple Machine Model



Why a simple cache model?

- Bottlenecks in the memory hierarchy often dominate anything else
 - You should first check this with real measurements
- Qualitative results still useful for cache optimizations



Outline



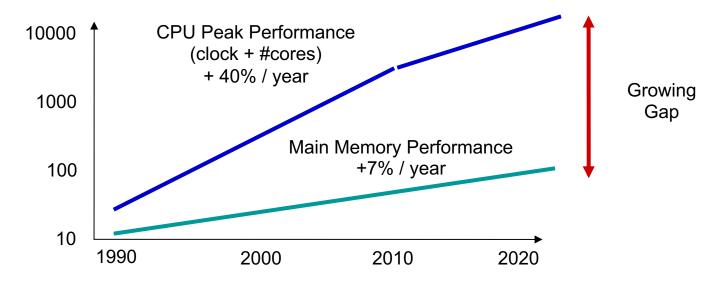
- Background
- Callgrind and {Q,K}Cachegrind
 - Measurement
 - Visualization
- Hands-On
 - Example: N-Body / Cache Use



Cache Exploitation is Important



"Memory Wall"



Access latencies to local memory on modern x86 processors ~ 200 cycles
→ AVX512 can do 200 * 8 (vector) * 4 (2 FMA units) = 6400 DP-FLOPs / access

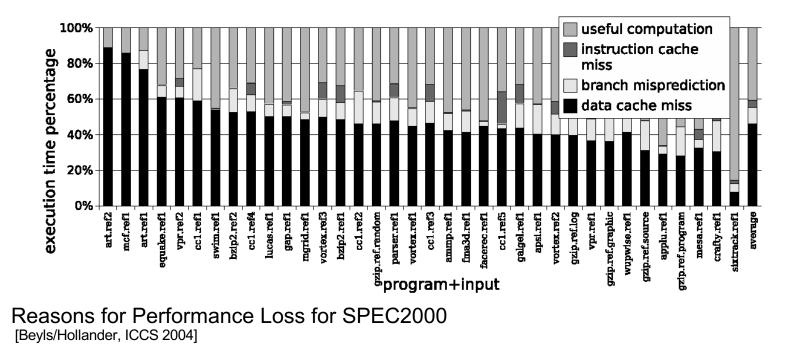
Caches do their Job transparently...



Caches work because programs expose access locality

• Temporal (hold recently used data) / Spatial (work on blocks of memory)

The "Principle of Locality" is not enough... → "Cache optimization"





Cache Optimization on Parallel Code



- Analyze sequential code phases
 - Optimization of sequential phases always improves runtime
 - No need to strip down to sequential program
- Influences of threads/tasks on cache exploitation
 - On multi-core: all cores share bandwidth to main memory
 - Use of shared caches: cores compete for space vs. cores prefetch for each other
 - Slowdown because of "false sharing"
 - not easy to measure with hardware performance counters



Going Sequential ...



- Sequential performance bottlenecks
 - Logical errors (unneeded/redundant function calls)
 - Bad algorithm (high complexity or huge "constant factor")
 - Bad exploitation of available resources (caches, vector units, pipelining,...)
- How to improve sequential performance
 - Use tuned libraries where available
 - Check for above obstacles → by use of analysis tools



(Sequential) Performance Analysis Tools



- Count occurrences of events
 - Resource exploitation is related to events
 - SW-related: function call, OS scheduling, ...
 - HW-related: FLOP executed, memory access, cache miss, time spent for an activity (like running an instruction)
- Relate events to source code
 - Find code regions where most time is spent
 - Check for improvement after changes
 - "Profile data": histogram of events happening at given code positions
 - Inclusive vs. Exclusive cost



How to measure Events



- Target: real hardware
 - Needs sensors for interesting events
 - For low overhead: hardware support for event counting
 - May be difficult to understand because of unknown micro-architecture, overlapping and asynchronous execution
- Target: machine model
 - Events generated by a simulation of a (simplified) hardware model
 - No measurement overhead: allows for sophisticated online processing
 - Simple models make it easier to understand the problem and to think about solutions
- Both methods (real vs. model) have advantages & disadvantages, but reality matters in the end





Latency

- Exploit (fast) cache: improve locality of data
- Allow hardware to prefetch data (use access patterns which are easy to predict)
- Memory controller on chip (standard today) be aware of NUMA

Bandwidth

- Share data in caches among cores
- Keep working set in cache (temporal locality)
- Use good data layout (spatial locality)
- If memory accesses are unavoidable
 - Predictable access pattern (stream/strided) → exploit HW prefetcher
 - Memory affinity
 - Avoid data dependencies (linked list traversals)

Optimization 1: Reduce Number of Accesses

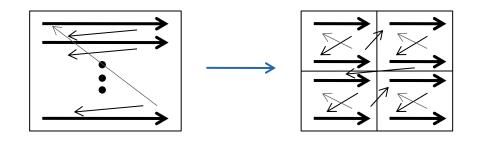


- Use large data types (may be done by compiler)
 - Vectors instead of bytes
- 1 cache line = 1 access: use full cache lines
 - Alignment: crossing cache line gives two accesses
- (redundant) Calculation instead of memory access
- Avoid unneeded writes
 - Check if a variable already has given value before writing
 - "Write-allocate" effect: higher bandwidth than expected



Optimization 2: Reorder Accesses

- If possible, do sequential accesses (in inner loop level)
 - Exploit full cache line
 - Trigger hardware prefetcher (small sequential accesses reduce accuracy of HW prefetcher)
- Blocking: reuse data as much as possible
 - Instead of multiple large sweeps over large buffer, split up into multiple small sweeps over buffer parts
 - Useful in 1d, 2d, 3d, ...
 - Recursive (multi-level) blocking: "cache-oblivious": best use of multiple cache levels at once!
 - Multi-core: consecutive iterations on cores with shared cache



Optimization 3: Improve Data Layout



- Group data with same access frequency and access type (read vs. write)
 - Use every byte of a fetched cache line (unused data is wasted space + bandwidth)
 - AoS-to-SoA
- Reorder data in memory according to traversal order in program
- Avoid power-of-2 strides: may produce conflict misses
 - By padding



Callgrind: Cache Simulation with Call-Graph Capturing







Based on Valgrind

- Runtime instrumentation infrastructure (no recompilation needed)
- Dynamic binary translation of user-level processes
- Linux/AIX/OS X on x86, x86-64, PPC32/64, ARM/ARM64, MIPS
- Open source (GPL), www.valgrind.org
- Includes correctness checking & profiling tools
 - "memcheck": accessibility/validity of memory accesses
 - "helgrind" / "drd": race detection on multithreaded code
 - "cachegrind"/"callgrind": cache & branch prediction simulation
 - "massif": memory profiling

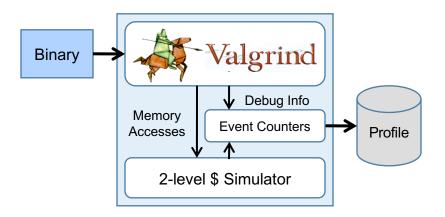


Callgrind: Basic Features



Part of Valgrind (Open Source, GPL)

- Callgrind vs. Cachegrind
 - Dynamic call graph
 - Simulator extensions
 - More control
- Measurement
 - Profiling via machine simulation (simple cache model)
 - Instruments memory accesses to feed cache simulator
 - · Hook into call/return instructions, thread switches, signal handlers
 - Instruments (conditional) jumps for CFG inside of functions
- Presentation of results: callgrind_annotate / {Q,K}Cachegrind





Simulation vs. Real Measurement



Usage of Valgrind

- Driven only by user-level instructions of one process
- Slowdown (call-graph tracing: 15-20x, + cache simulation: 40-60x)
 - "fast-forward mode": 2-3x
- Serializes threads
- Detailed observation
- Does not need root access / can not crash machine

Cache model

- "Not reality": synchronous 2-level inclusive cache hierarchy (size/associativity taken from real machine, always including LLC)
- Reproducible results independent on real machine load
- Derived optimizations applicable for most architectures

Callgrinds Cache Model vs. Xeon



Callgrind

- Parameters: size, line size, associativity
- L1 / LLC, inclusive, LRU, shared among threads
- Write back vs. write through does not matter for hit/miss counts
- Optional stream prefetcher

CoolMUC2 node: 2x Intel Xeon (Haswell, each 14 cores, 18 MB L3)

- private L1 (D/I a 32kB) + L2 (256 kB) per core
- L1/L2 strictly inclusive to L3, L3 shared

Callgrind only simulates 2 levels (L1+LLC) → LLC hit count higher

Assume all threads work on separate data: can specify LLC size = 18 / 14 MB



Callgrind: Advanced Features



- Interactive control (backtrace, dump command, ...)
- "Fast forward"-mode to quickly get at interesting code phases
- Application control via "client requests" (start/stop, dump)

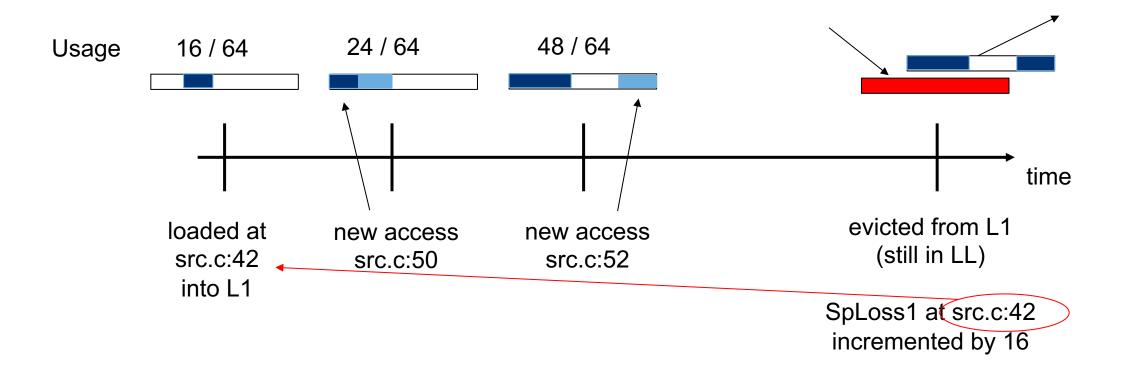
Optional

- Best-case simulation of simple stream prefetcher
- Byte-wise usage of cache lines before eviction
- Branch prediction
- Dynamic context in function names (call chain/recursion depth)
- Wallclock time spent in system calls (useful for MPI)



Byte-wise Cacheline Usage





- Why "Loss" events? Higher Numbers should point at larger bottlenecks (here: 16B lost)
- Why attribution to line loading the cacheline? No variable to attach "Loss" to, still understandable

Callgrind Cheat-Sheet



- "valgrind –tool=callgrind [callgrind options] <yourprogram> [args]"
- Cache simulator: "--cache-sim=yes"
- Specify cache sizes: "--L1/I1/LL=<size>,<assoc>,<linesize>"
- Branch prediction simulation: "--branch-sim=yes"
- Enable for machine code annotation: "--dump-instr=yes"
- Start in "fast-forward": "--instr-atstart=yes"
 - Switch on event collection: "callgrind_control –i on"
- Spontaneous dump: "callgrind_control –d [dump identification]"
- Current backtrace of threads (interactive): "callgrind_control –b"
- Separate output per thread: "--separate-threads=yes"
- Jump-profiling in functions (CFG): "--collect-jumps=yes"
- Time in system calls: "--collect-systime=yes"
- Byte-wise usage within cache lines: "--cacheuse=yes"



{Q,K}Cachegrind: Graphical Browser for Profile Visualization





Features



Open source, GPL, kcachegrind.github.io

- https://github.com/KDE/kcachegrind
- includes pure Qt version, able to run on Linux / OS-X / Windows

Visualization of

- Call relationship of functions (callers, callees, call graph)
- Exclusive/Inclusive cost metrics of functions
 - Grouping according to ELF object / source file / C++ class
- Source/assembly annotation: costs + CFG
- Arbitrary events counts + specification of derived events

Callgrind support: file format, events of cache model

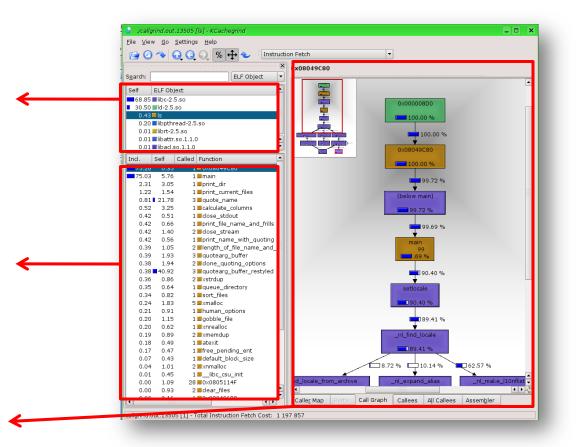


Usage



qcachegrind callgrind.out.<pid>

- Left: "Dockables"
 - list of function groups groups according to
 - library (ELF object)
 - source
 - class (C++)
 - list of functions with
 - inclusive
 - exclusive costs





Visualization panes for selected function



- List of event types
- List of callers/callees

| Types | <u>C</u> allers | <u>A</u> ll Caller | s S <u>o</u> | urce | Ca | llee <u>M</u> ap | | | |
|---|---|---|--|--|---|---|---------|-------|-------|
| Event Ty | pe | Incl. | Self | Short | | Formula | | | |
| Instructio | on Fetch | 75.03 | 0.02 | | Ir | | | | |
| Data Rea | nd Access | 72.31 | 0.02 | 2 C |)r | | | | |
| Data Wri | te Access | 73.02 | 0.07 | ' Di | w | | | | |
| L1 Instr. | Fetch Miss | 58.47 | 2.43 | I In | ۱r | | | | |
| | Read Miss | | 0.22 | 2 D1m | ٦r | | | | |
| | Write Miss | | 1.19 | 01m | w | | | | |
| L2 Instr. | Fetch Miss | 54.75 | 2.53 | I2m | ۱r | | | | |
| | Read Miss | | |) D2m | ۱r | | | | |
| | Write Miss | | | 2 D2m | | | | | |
| L1 Miss S | | 52.65 | | | | I1mr + D: | | | |
| L2 Miss S | | 44.93 | | | | I2mr + D2 | | | |
| Cycle Est | imation | 67.05 | 0.30 |) CEs | st = | Ir + 10 L1 | lm + 10 | 0 L2m | |
| | | | | | | | | | |
| Ir | Count C | allee | | | | | | | [|
| Ir 90.68 | | Callee | (libc-2. | 5.so: se | etloo | cale.c) | | | |
| | 3 1 | | | | etloc | cale.c) | | | |
| 90.68 | | setlocale | (ls: ls.c) | | etloc | cale.c) | | | |
| 90.68 | | setlocale print_dir | (ls: ls.c) 2.5.so: | exit.c) | | | | | |
| 90.68 3.08 1.95 1.78 0.53 | 3 1 3 1 5 1 8 8 | setlocale print_dir lexit (libc- _dl_runtii clone_qu | (ls: ls.c) 2.5.so: me_res oting_o | exit.c) olve (Id ptions | -2.5 (ls: | i.so) quotearg. | c) | | |
| 90.68 3.08 1.95 1.78 0.55 | 3 1 3 1 5 1 8 8 1 2 7 5 | lsetlocale print_dir exit (libc- _dl_runtir done_qu getenv (li | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5. | exit.c) olve (Id ptions so: get | -2.5 (ls: env. | i.so) quotearg. | c) | | |
| 90.68 3.08 1.95 1.78 0.53 0.47 0.47 | 3 1 3 1 5 1 3 8 4 2 7 5 5 1 | Isetlocale print_dir exit (libc- _dl_runtir done_qu getenv (li queue_di | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5.s rectory | exit.c) olve (Id ptions so: get (Is: Is.c | -2.5 (Is: env. c) | i.so) quotearg. .c) | c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.45 0.46 0.28 | 3 1 3 1 5 1 3 8 4 2 7 5 5 1 3 1 | setlocale print_dir exit (libc- _dl_runtir clone_qu getenv (li queue_di human_o | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5.: irectory ptions (| exit.c) olve (Id ptions so: get (Is: Is.c | -2.5 (Is: env. c) | i.so) quotearg. .c) | c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.45 0.45 0.46 0.28 0.28 | 8 1 8 1 5 1 8 8 1 2 7 5 5 1 8 1 5 1 1 5 1 1 5 1 | setlocale print_dir exit (libc- _dl_runtir done_qu getenv (li queue_di human_o atexit (ls) | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5.s rectory ptions (| exit.c) olve (ld ptions so: get (ls: ls.c (ls: hun | -2.5 (ls: env. c) nan. | i.so) quotearg. .c) | c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.45 0.46 0.28 0.25 0.25 | 8 1 3 1 5 1 3 8 4 2 5 1 3 1 5 1 3 1 5 1 3 1 5 1 3 1 | Isetlocale Iprint_dir I Iexit (libc- I_dl_runtir Idone_qu Igetenv (li Iqueue_di Ihuman_o Iatexit (Is) | (Is: Is.c) 2.5.so: me_res oting_o ibc-2.5.s irectory ptions (ding_er | exit.c) olve (ld ptions so: get (ls: ls.c (ls: hun nt (ls: ls | -2.5 (Is: env. c) nan. s.c) | i.so) quotearg. c) c) | c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.45 0.45 0.28 0.28 0.25 0.23 0.11 | 8 1 3 1 5 1 3 8 4 2 7 5 5 1 3 1 15 1 3 1 1 1 | setlocale print_dir i lexit (libc i_dl_runtir idone_qui igetenv (li iqueue_di human_o latexit (ls) ifree_pen- igetopt_lo | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5.s irectory ptions (ding_er ong (libc | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hun nt (Is: Is :-2.5.so | -2.5 (ls: env. c) nan. s.c) : ge | i.so) quotearg. .c) c) etopt1.c) | | | |
| 90.66 3.08 1.95 1.78 0.55 0.47 0.28 0.28 0.25 0.25 0.25 0.25 0.25 0.25 | 8 1 3 1 5 1 8 2 7 5 5 1 8 1 9 1 10 1 11 1 12 1 13 1 14 1 15 1 | setlocale print_dir lexit (libc-: _dl_runtin ldone_qui getenv (li queue_di human_o latexit (ls) lfree_pen- getopt_lo bindtextd | (ls: ls.c) 2.5.so: me_res oting_o bc-2.5.: irectory ptions (ding_er ong (libo lomain (| exit.c) olve (Id ptions so: get (Is: Is.c (Is: hun ht (Is: Is -2.5.so (libc-2.5 | -2.5 (ls: env. c) nan. s.c) c: ge 5.so | i.so) quotearg. c) c) topt1.c) : bindtexto | dom.c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.47 0.44 0.28 0.25 0.23 0.11 0.06 | 8 1 3 1 5 1 8 2 7 5 5 1 8 1 10 1 10 1 11 1 12 1 13 1 14 1 15 1 15 1 | setlocale print_dir lexit (libc-) ldone_que lgetenv (li lqueue_di lhuman_o latexit (ls) lfree_pen getopt_lc bindtextdd ltextdoma | (ls: ls.c) 2.5.so: me_res oting_o ibc-2.5.: irectory ptions (ding_er ong (libc- lomain (ibc- in (libc- | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hum t (Is: Is -2.5.so (libc-2.5 2.5.so: | -2.5 (ls: env. c) nan. s.c) c: ge 5.so | i.so) quotearg. .c) c) etopt1.c) | dom.c) | | |
| 90.68 3.08 1.95 1.78 0.55 0.42 0.25 0.25 0.23 0.11 0.06 0.05 0.04 | 3 1 3 1 5 1 3 8 4 2 5 1 3 1 5 1 3 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 | setlocale print_dir iexit (libc- i_dl_runtii ldone_qui getenv (li queue_di human_o latexit (ls) ffree_peni getopt_lc bindtextd textdoma xnmalloc | (ls: ls.c) 2.5.so: me_res oting_o bc-2.5.: irectory ptions (ding_er ong (libc lomain (in (libc- (ls: xma | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hur ht (Is: Is :-2.5.so libc-2.5 2.5.so: alloc.c) | -2.5 (ls: env. c) nan. s.c) : ge 5.so tex | i.so) quotearg. c) c) topt1.c) : bindtexto | dom.c) | | |
| 90.68 3.00 1.95 1.76 0.55 0.47 0.44 0.28 0.25 0.25 0.23 0.11 0.00 0.05 0.04 0.05 | 8 1 3 1 5 1 3 8 4 1 | setlocale print_dir lexit (libc- _d_runtii done_qui getenv (li queue_di human_o latexit (ls) ffree_pen- getopt_ld bindtextd textdoma xinmalloc lisatty (lib | (Is: Is.c) 2.5.so: me_res oting_o bc-2.5.s irectory ptions I ding_er ong (libc lomain I sin (libc- (Is: xma c-2.5.so | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hun t (Is: Is -2.5.so (Ibc-2.5 2.5.so: alloc.c) o: isatty | -2.5 (ls: env. c) nan. s.c) : ge 5.so tex (.c) | .so) quotearg. c) topt1.c) : bindtext tdomain.c | dom.c) | | |
| 90.66 3.00 1.95 1.76 0.47 0.46 0.25 0.25 0.23 0.11 0.06 0.00 0.00 0.00 | 3 1 3 1 5 1 3 8 2 7 5 1 3 1 5 1 3 1 5 1 3 1 5 1 5 1 5 1 5 1 5 1 5 1 5 1 6 1 1 1 1 1 1 1 | setlocale print_dir i lexit (libc- l_dl_runtii idone_qui getenv (li lqueue_di human_o latexit (ls) ifree_peni getopt_lc bindtextd textdoma ixnmalloc isatty (lib set_char_ | (Is: Is.c) 2.5.so: me_res oting_o bc-2.5.: irectory ptions I ding_er ong (libc- lomain I in (libc- (Is: xma c-2.5.so _quotin | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hun t (Is: Is -2.5.so (libc-2.5 2.5.so: 2.5.so: alloc.c) o: isatty g (Is: q | -2.5 (ls: env. c) nan. s.c) : ge 5.so tex (.c) | .so) quotearg. c) topt1.c) : bindtext tdomain.c | dom.c) | | |
| 90.68 3.00 1.95 1.76 0.55 0.47 0.44 0.28 0.25 0.25 0.23 0.11 0.00 0.05 0.04 0.05 | 8 1 3 1 5 1 8 8 4 1 5 1 5 1 8 1 5 1 8 1 5 1 5 1 5 1 6 1 1 1 5 1 4 1 0 1 | setlocale print_dir lexit (libc- _d_runtii done_qui getenv (li queue_di human_o latexit (ls) ffree_pen- getopt_ld bindtextd textdoma xinmalloc lisatty (lib | (Is: Is.c) 2.5.so: me_reso oting_o bc-2.5.s irectory ptions (ding_er ong (Ibc lomain (lioc- (Is: xma c-2.5.so _quotin s (Is: Is. | exit.c) olve (Id ptions so: get (Is: Is.c (Is: hur t (Is: Is -2.5.so (libc-2.5 2.5.so: 2.5.so: alloc.c) o: isatty g (Is: qu c) | -2.5 (ls: env. c) nan. s.c) tex f.so tex (.c) | i.so) quotearg. c) topt1.c) : bindtextd tdomain.c earg.c) | dom.c) | | |

- Treemap visualization
- Call Graph

| Types | <u>C</u> allers | <u>A</u> ll Callers | S <u>o</u> urce | Callee <u>M</u> ap | | | |
|-----------------|-----------------|---------------------|-----------------|--------------------|--------------|-------------|-------------|
| LnLm | ake_l10nfli | sť2 | | 58.0 | 2 % | | |
| strcm | p | | | 31.59 % | | % -dl_addr | |
| strier geten | strulen % | | stpcpy | | | t | |
| | | | | | setlocale | 6 | [|
| | ā. | | | | 89 | .69 % | |
| | | - | | - | nl_find_loca | | 6 |
| - | | | ~ | 68.75 % | 6 10. | 6 17 % | C 62 |
| | | d_locale_from | - | 68.75 % | 89.69 % | 6 17 % | 062 |

- Source annotation
- Assemly annotation

| Types | Callers | All Caller | s Source | Cal | lee Map | | | |
|--------------------|---------|------------|--|--------------|--|---------------|----------|--|
| # Ir | 20000 | | | | coreutils-6.4/ | arc/le_c') | | |
| 1119 | | 300100 | | iebug/ | corectila-oiy. | si c/isic / | | |
| 1120 | | #if ! SA | A_NOCLDST | OP | | | 10 | |
| 1121 | | bool | caught_sig[r | nsigs]; | | | | |
| 1122 | | #endif | | | | | | |
| 1123 | | | | | | | | |
| 1124 | | | ize_main (& | | | | | |
| 1125 | 2 | | am_name = | | D]; | | | |
| 1126 | 8 | | cale (LC_ALL | | | | | |
| | 814 979 | | | | c-2.5.so: setl | | | |
| 1107 | 2 155 | | | | resolve' (Id-2 | | | |
| 1127 | 8 | | | | GE, LOCALED | | | |
| 2 263 | | | 1 call to '_dl_runtime_resolve' (ld-2.5.so) | | | | | |
| 565 1128 7 | | | 1 call to 'bindtextdomain' (libc-2.5.so: bindtextdom.c) textdomain (DacKaCE); | | | | | |
| 1120 | 1 935 | | textdomain (PACKAGE); 1 call to ' dl runtime resolve' (Id-2.5.so) | | | | | |
| | 456 | | I call to 'textdomain' (libc-2.5.so; textdomain.c) | | | | | |
| 1129 | 150 | - 100 | an co concuo | intanii (| 100 2101001 00 | Accontantic) | - | |
| 1130 | | initial | ize evit fail | ure (I 🤇 | S FATLURE) | | | |
| | | | | | | | 1 | |
| # | Ir | | | | Assembler | rawpic | | |
| 804 DF8 | | 1 | | ub | \$0x1.%eax | | | |
| 804 DF9 | 1 | 1 | je | е | 804e5d8 <ao< td=""><td>l_set_fd@plt-</td><td>0x4968</td></ao<> | l_set_fd@plt- | 0x4968 | |
| | | | | | Jump 1 of 1 | times to 0x80 | 04E5D8 | |
| 804 DF9 | 7 | | c | all | 8049650 <ab< td=""><td>ort@plt></td><td></td></ab<> | ort@plt> | | |
| 804 DF9 | с | | m | novl | \$0x2,0x805e | 328 | | |
| 804 DFA | 3 | | | | | | | |
| 804 DFA | | | n | novl | \$0×4,0×4(%e | sp) | | |
| 804 DFA | | | | | | | | |
| 804 DFA | | | | lvor | \$0x0,(%esp) | | | |
| 804 DFB | - | | - | all | | l_set_fd@plt- | +0xa9c0> | |
| 804 DFB | | 1 | → n | novl | \$0x0,0x805e | 32c | | |
| 804 DFC | - | | | | to o o or- | | | |
| 804 DFC | | 1 | n | lvor | \$0x0,0x805e | 330 | | |
| 804 DFC | - | | _ | acub | +0-0 0-00 | 224 | | |
| 804 DFC 804 DFD | | 1 | | 10vb 10vb | \$0x0,0x805e \$0x0.0x805e | | | |
| 804 DFD | | 1 | | 10VD | \$0x0,0x805e | | | |









Cache Analysis | 28 June 2022 | J. Weidendorfer

Getting started



- Setup on CoolMUC2
 - "cp /lrz/sys/courses/hcow1s22/qcachegrind ~/bin"
 - put "~/bin" in your \$PATH
 - "module load valgrind"
- Test: What happens in "/bin/ls"?
 - run "valgrind --tool=callgrind ls /usr/bin"
 - run "qcachegrind"
 - function with highest instruction execution count? Purpose?
 - where is the main function?
 - run with cache simulation: "--cache-sim=yes"





Go into N-Body sources, directory "nbody/ver1"

- "cp /lrz/sys/courses/hcow1s22/compile-cg ."
- Run "./compile-cg" we compile with "-g" and compile Array-of-Struct variant
- See comments in compile-cg for how to run with callgrind

Cache Line Usage SoA vs AoS

- "valgrind --tool=callgrind --dump-instr=yes --cacheuse=yes \ ./\$BIN --run-sim=medium --steps=10" with BIN = "nbody-aos-g-ver1e" (AoS) and BIN = "nbody-g-ver1e" (SoA)
- Run "qcachegrind" (loads all callgrind.out.* files by default)
- Right-click in list on Types tab, "New Event Type", double click formula column: "64 L1m" = "How much data is loaded into L1"
- compare with "SpLoss1": "How much data was never accessed but loaded into L1"



Q&A

Josef Weidendorfer LRZ weidendo@lrz.de



