

#### Cache Performance Analysis with Callgrind and KCachegrind

#### Josef Weidendorfer<sup>1</sup>

40<sup>th</sup> VI-HPS Tuning Workshop

June 2021, Garching

<sup>1</sup> Leibniz Computing Centre (LRZ)



#### 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

### VI-HPS

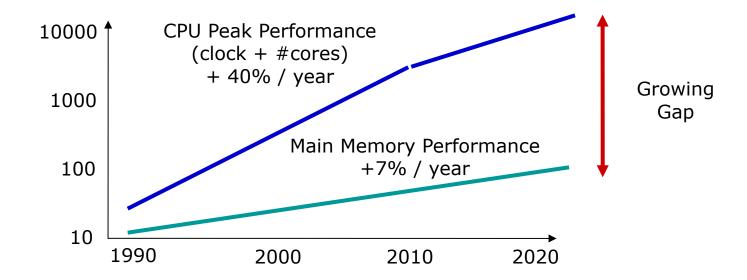
### Outline

- Background
- Callgrind and {Q,K}Cachegrind
  - Measurement
  - Visualization
- Hands-On
  - Example: Matrix Multiplication

VIRTUAL INSTITUTE HIGH PRODUCTIVITY SUPERCOMPUTING

#### Single Node CPU Performance: 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

#### Single Node CPU Performance: Cache Exploitation is Important

This will mostly be true also in the future

- Latency of main memory access does not improve
- Bandwidth to typical main memory (DDRx) increases slower than compute power
  - Lots of integrated cores requesting access to memory (multi-core, accelerators)
  - Improvements GDDRx / HBMx: better bandwidth, but fixed capacity, higher power (Examples: GPUs / A64FX with HBM)

Power consumption [Keynote Dongarra, PPAM 2011]

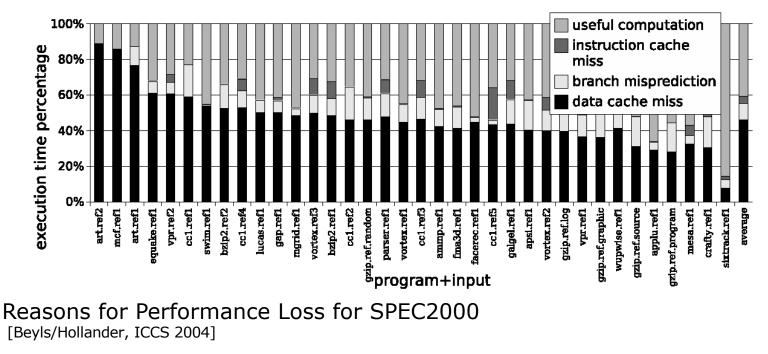
- DP FMADD: 100 pJ (2011) → 10 pJ (2018)
  DP Read DRAM: 4800 pJ (2011) → 1920 pJ (2018)
- "Power Wall": drives large system designs (lots of compute but low memory capacity  $\rightarrow$  NVM)
- "Affinity Wall": pressure towards shorter transfer distances makes programming more complex (increased number of NUMA domains, NUCA & PIM designs)

#### **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"



#### How to do Cache Optimization on Parallel Code

- Analyze sequential code phases
  - Optimization of sequential phases always improve 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  $\rightarrow$  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

#### **Back to the Memory Wall: Improvements**

#### 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)

#### **Cache Optimization (1): Reduce Number of Accesses**

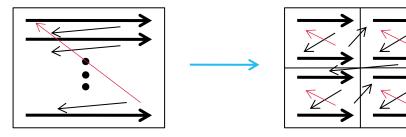
- Use large data types (may be done by compiler)
  - Vectors instead of bytes
- I cache line = 1 access: use full cache lines
  - Alignment: crossing cache line gives two accesses
- (redundant) Calculation instead of memory access
- Avoid unneeded writes
  - Check if a variable already has given value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

### **Cache 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



#### **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



#### **Callgrind: Basic Features**

Based on Valgrind

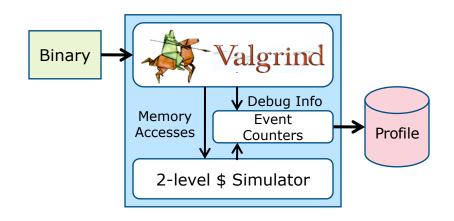
- Runtime instrumentation infrastructure (no recompilation needed)
- Dynamic binary translation of user-level processes
- Linux/AIX/OS X on x86, x86-64, PPC32/64, ARM/ARM64, MIPS
- Open source (GPL), www.valgrind.org
- Includes correctness checking & profiling tools
  - "memcheck": accessibility/validity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

V VIRTUAL INSTITUTE - HIGH PRODUCTIVITY SUPERCOMPUTING

#### **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



#### **Pro & Contra (i.e. 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

- 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

IvyMUC node: 2x Intel Xeon (IvyBridge, each 16 cores, 20 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

#### **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)

## **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

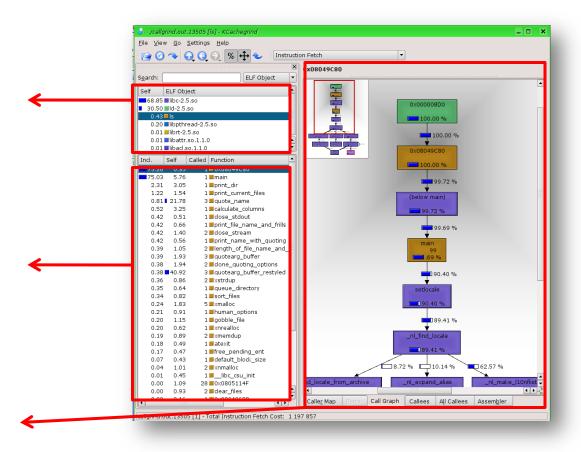
- Call relationship of functions (callers, callees, call graph)
- Exclusive/Inclusive cost metrics of functions
  - Grouping according to ELF object / source file / C++ class
- Source/assembly annotation: costs + CFG
- Arbitrary events counts + specification of derived events

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



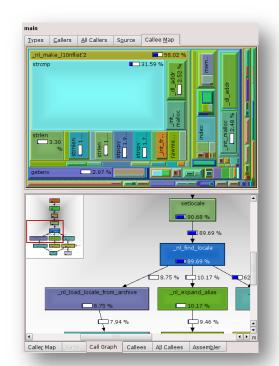
Right: visualization panes

#### **Visualization panes for selected function**

- List of event typesList of callers/callees
- Treemap visualization
- Call Graph

Source annotationAssemly annotation

<u>T</u> ypes	<u>C</u> allers	<u>A</u> ll Caller	s S <u>o</u> u	urce	Ca	llee <u>M</u> ap				
Event T	ype	Incl.	Self	Short	Τ	Formula				
Instructi	on Fetch	75.03	0.02	I	r					
Data Re	ad Access	72.31	0.02	D	r					
Data Wr	ite Access	73.02	0.07	Dv	v					
	Fetch Miss		2.43	I1m	r					
	Read Miss		0.22	D1m	r					
	Write Miss			D1mv						
	Fetch Miss		2.53							
	Read Miss		0.00							
	Write Miss			D2mv						
L1 Miss		52.65				I1mr + D1				
L2 Miss : Cvde Es		44.93				I2mr + D2 Ir + 10 L1				
Cycle Es	unacon	67.05	0.50	CES		n + 10 L1		100 L2III	'	
Ir	Count	ممالد								
Ir	Count C		(libe 2.5		tlor					
90.6	8 1	setlocale			tloc	cale.c)				
	8 1 8 1		ls: ls.c)		tloc	cale.c)				
90.6 3.0	8 1 8 1 5 1	setlocale print_dir (	ls: ls.c) 2.5.so: (	exit.c)						
90.6 3.0 1.9	8 1 8 1 5 1 8 8	setlocale  print_dir ( lexit (libc-:  _dl_runtir	ls: ls.c) 2.5.so: e ne_resc	exit.c) blve (Id-	-2.5		:)			
90.6 3.0 1.9 1.7	8 1 8 1 5 1 8 8 1 2	setlocale  print_dir ( lexit (libc-:  _dl_runtir	ls: ls.c) 2.5.so: « me_resc pting_op	exit.c) blve (Id- ptions (	-2.5 ls:	i.so) quotearg.c	:)			
90.6 3.0 1.9 1.7 0.5	8 1 8 1 5 1 8 8 1 2 7 5	lsetlocale  print_dir ( lexit (libc-2  _dl_runtir  clone_qu	ls: ls.c) 2.5.so: e me_resc oting_op bc-2.5.s	exit.c) blve (Id- ptions ( co: gete	-2.5 ls: anv.	i.so) quotearg.c	;)			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2	8 1 8 1 5 1 8 8 1 2 7 5 6 1 8 1	setlocale lprint_dir ( lexit (libc-: _dl_runtir lclone_qua lgetenv (li lqueue_di lhuman_o	ls: ls.c) 2.5.so: ( ne_resc oting_op bc-2.5.s rectory ptions (	exit.c) blve (Id- ptions ( co: gete (Is: Is.c	-2.5 ls: env.	i.so) quotearg.c .c)	:)			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2	8 1 8 1 8 8 1 2 7 5 6 1 8 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_resc oting_op bc-2.5.s rectory ptions (	exit.c) blve (Id- ptions ( co: gete (Is: Is.c Is: hum	-2.5 ls: anv. ) ian.	i.so) quotearg.c .c)	)			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2	8 1 8 1 5 1 8 8 1 2 7 5 6 1 8 1 5 1 3 1	setlocale print_dir ( lexit (libc-3 l_dl_runtir ldone_qua lgetenv (li lqueue_di human_o latexit (ls) free_pena	ls: Is.c) 2.5.so: ( me_resc oting_op bc-2.5.s rectory ptions ( ding_en	exit.c) olve (Id- ptions ( co: gete (Is: Is.c Is: hum t (Is: Is	-2.5 ls: anv. an. .c)	i.so) quotearg.c .c) c)	)			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.2	8 1 8 1 5 1 8 8 1 2 7 5 6 1 8 1 5 1 3 1 1 1	setlocale print_dir ( lexit (libc-3 _dl_runtir clone_que getenv (li queue_di human_o latexit (ls) lfree_pene getopt_lo	ls: Is.c) 2.5.so: e me_resc bting_op bc-2.5.s rectory ptions ( ding_en ng (libc	exit.c) olve (Id- ptions ( co: gete (Is: Is.c Is: hum t (Is: Is -2.5.so	-2.5 ls: anv. ) ian. .c) : ge	i.so) quotearg.c .c) c) etopt1.c)				
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.2 0.1 0.0	8 1 8 1 8 8 1 2 7 5 6 1 8 1 5 1 3 1 1 1 6 1	setlocale print_dir ( lexit (libc-3 _dl_runtir clone_que getenv (li queue_di human_o latexit (ls) lfree_pene getopt_lo bindtextd	ls: ls.c) 2.5.so: e ne_resc bting_op bc-2.5.s rectory ptions ( ding_en ng (libc omain (	exit.c) olve (Id- ptions ( :o: gete (Is: Is.c Is: hum t (Is: Is -2.5.so libc-2.5	-2.5 ls: anv. ) an. .c) : ge	i.so) quotearg.c c) c) topt1.c) : bindtextc	om.e			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.1 0.0 0.0	8 1 8 1 8 8 1 2 7 5 6 1 8 1 5 1 1 1 1 1 6 1	setlocale print_dir ( lexit (libc- done_qua getenv (li queue_di lhuman_o latexit (ls) free_pena getopt_lo bindtextd	ls: ls.c) 2.5.so: ( ne_resc bting_op bc-2.5.s rectory ptions ( ding_en ng (libc- omain ( in (libc-	exit.c) olve (Id- ptions ( :o: gete (Is: Is.c Is: hum t (Is: Is -2.5.so libc-2.5 2.5.so:	-2.5 ls: anv. ) an. .c) : ge	i.so) quotearg.c .c) c) etopt1.c)	om.e			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.1 0.0 0.0 0.0	8 1 8 1 5 1 8 8 1 2 7 5 6 1 8 1 5 1 3 1 1 1 6 1 5 1 4 1	setlocale print_dir ( lexit (libc- _dl_runtir ldone_qua getenv (li queue_di lhuman_o latexit (ls) firee_pena getopt_lo bindtextd textdoma xnmalloc i	ls: ls.c) 2.5.so: ( ne_resc oting_op bc-2.5.s rectory ptions ( ding_en ng (libc- omain ( in (libc- (ls: xma	exit.c) olve (Id- ptions ( :o: gete (Is: Is.c Is: hum t (Is: Is -2.5.so libc-2.5 2.5.so: Iloc.c)	-2.5 ls: anv. ) an. .c) : ge .so tex	i.so) quotearg.c c) c) topt1.c) : bindtextc	om.e			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.2 0.1 0.0 0.0 0.0 0.0	8         1           8         1           5         1           8         8           7         5           6         1           9         1           10         1           11         1           12         1           13         1           14         1           14         1	setlocale print_dir ( lexit (libc-1 _dl_runtir ldone_qu lgetenv (li queue_di human_o latexit (ls) ffree_penv lgetopt_lo bindtextd textdoma xnmalloc 1 lisatty (lib	Is: Is.c) 2.5.so: ( me_resc bting_op bc-2.5.s rectory ptions ( ding_en ng (libc- omain ( in (libc-3 (Is: xma c-2.5.so	exit.c) olve (Id- ptions ( io: gete (Is: Is.c Is: hum t (Is: Is -2.5.so libc-2.5 2.5.so: Iloc.c) : isatty	-2.5 ls: anv. ) .c) .so tex .c)	i.so) quotearg.c c) c) topt1.c) : bindtextc tdomain.c)	om.e			
90.6 3.0 1.9 1.7 0.5 0.4 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.0 0.0 0.0 0.0 0.0	8     1       8     1       5     1       8     8       1     2       7     5       6     1       13     1       14     1       14     1       10     1	setlocale print_dir ( lexit (libc-1 _dl_runtir idone_qu getenv (li lqueue_di lqueue_di lqueue_di lqueue_di tops inter_pend getopt_lo bindtextd textdoma ixnmalloc isatty (libi set_char_	Is: Is.c) 2.5.so: ( me_resc bting_op bc-2.5.s rectory ptions ( ding_en ng (libc- omain ( in (libc-2 (ls: xma c-2.5.so _quoting_	exit.c) blve (ld- ptions ( co: gete (ls: ls.c ls: hum t (ls: ls -2.5.so libc-2.5 2.5.so: 2.5.so: lloc.c) : isatty g (ls: qu	-2.5 ls: anv. ) .c) .so tex .c)	i.so) quotearg.c c) c) topt1.c) : bindtextc tdomain.c)	om.e			
90.6 3.0 1.9 1.7 0.5 0.4 0.4 0.2 0.2 0.2 0.2 0.2 0.1 0.0 0.0 0.0 0.0	8         1           8         1           5         1           8         8           1         2           7         5           1         1           8         1           5         1           8         1           1         1           6         1           5         1           4         1           1         1           0         1	setlocale print_dir ( lexit (libc-1 _dl_runtir ldone_qu lgetenv (li queue_di human_o latexit (ls) ffree_penv lgetopt_lo bindtextd textdoma xnmalloc 1 lisatty (lib	Is: Is.c) 2.5.so: ( me_resc bting_op bc-2.5.s rectory ptions ( ding_en omain ( ibc- omain ( ibc- omain ( ibc- comain ( ibc- coma	exit.c) olve (ld- ptions ( co: gete (ls: ls.c ls: hum t (ls: ls -2.5.so libc-2.5 2.5.so: lloc.c) : isatty g (ls: qu c)	-2.5 ls: anv. ) an. .c) : ge .so tex .c)	i.so) quotearg.c c) xtopt1.c) : bindtextc tdomain.c) earg.c)	om.e			



<u>T</u> ypes	<u>C</u> allers	<u>A</u> ll Callers	S <u>o</u> urce	Callee <u>M</u> ap				
# Ir		Source (	 '/usr/src/de	bug/coreutils-6	.4/src/ls.c')	<b></b>		
1118						100		
1120			NOCLDSTO			222		
1121		bool caught_sig[nsigs];						
1122		#endif						
1123				rgc, &argv);				
1124								
1125	2		n_name = a e (LC ALL,					
1126	8							
_	814 979 2 155			e' (libc-2.5.so: :				
1127	2 155			time_resolve' (I ACKAGE, LOCAL				
12/	-							
	2 263 1 call to '_dl_runtime_resolve' (ld-2.5.so) 565 1 call to 'bindtextdomain' (libc-2.5.so: bindtextdom.c)							
128	7				.5.50. Diffuce/cubi	11.0)		
	1 935	textdomain (PACKAGE); 1 call to ' dl runtime resolve' (Id-2.5.so)						
	456	1 call to 'textdomain' (libc-2.5.so; textdomain.c)						
1129					eoneaonnainno,			
1130		initialize	evit failu	re (LS FATLURE	Ъ.	•		
#	Ir			Assemble	sao_soc_ra@picr			
304 DF8E		1	su					
304 DF91		1	je	804e5d8	<acl_set_fd@plt+< td=""><td>0x4968&gt;</td></acl_set_fd@plt+<>	0x4968>		
				Jump 1 o	f 1 times to 0x80	4E5D8		
304 DF97	·		ca	l 8049650	<abort@plt></abort@plt>			
304 DF90	2		mo	ovl \$0x2,0x8	05e328			
304 DFA3								
304 DFA6			mo	ovi \$0x4,0x4	(%esp)			
304 DFAE								
304 DFAE			ma					
304 DFB5 304 DFB4			ca		<acl_set_fd@plt+< td=""><td>0xaac0&gt;</td></acl_set_fd@plt+<>	0xaac0>		
304 DFBA 304 DFC1		1	→ mo	ovi \$0x0,0x8	U5e320			
304 DFC1 304 DFC4		1	ma	ovi \$0x0.0x8	05-220			
304 DFC4		-	me		006330			
304 DFCE		1	ma	ovb \$0x0,0x8	056334			
304 DFD5		1		vb \$0x0,0x8				
		1		wh #0.40 0.40		•		
(								
Caller Ma	p Part	Call Gra	ph Calle	es All Callee	s Assembler			



## Hands-on



## Getting started (see /lrz/sys/courses/vihps/material/kcg)

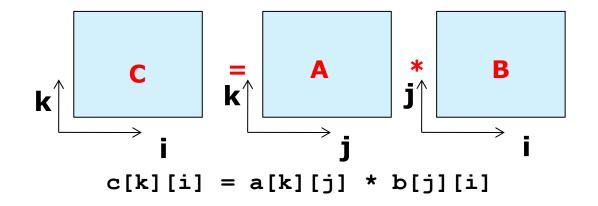
#### Setup

- "ssh -X hpckursXX@lxlogin10.lrz.de"
- "cp -r /lrz/sys/courses/vihps/material/kcg ."
- module use /lrz/sys/courses/vihps/modulefiles
- "module load qcachegrind"
- Valgrind is already in path (in system image)
- 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"

"source kcg/kcg.env"

### **Detailed analysis of matrix multiplication**

- Kernel for C = A \* B
  - Side length N → N3 multiplications + N3 additions

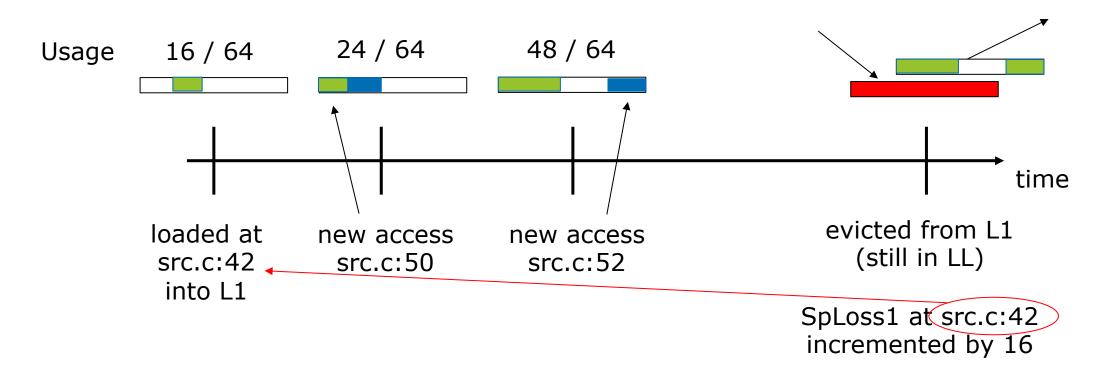


- 3 nested loops (i,j,k): Best index order?
- Optimization for large matrixes: Blocking

Callgrind

VIRTUAL INSTITUTE - HIGH PRODUCTIVITY SUPERCOMPUTING

#### **Cacheline Usage**



Why maintain "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

### **Detailed analysis of matrix multiplication**

#### • To try out...

- "cd ~/kcg; make"
- timing of orderings (e.g. size 512): "./mm 800"
- cache behavior for small matrix (fits into cache): "valgrind --tool=callgrind --cache-sim=yes ./mm 300"
- How good is L1/L2 exploitation of the MM versions?
- Warning: Login node has 20MB LLC (also used in simulation)

Cache Line Usage

- "valgrind --tool=callgrind --cacheuse=yes ./mm"
- 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"

Other example: 2d Jacobi solver: jc / jc.c

#### How to run with MPI (here: interactive session)

- Optional: reduce iterations in BT\_MZ / use class A
  - sys/setparams.c, write\_bt\_info, class A: set niter = 5
  - "make clean; make bt-mz CLASS=A NPROCS=2"
- module load salloc\_conf/ivymuc
- "salloc --nodes=1 --reservation=hhps1s21\_workshop"
- "module load valgrind"
- "export OMP\_NUM\_THREADS=2"
- "mpiexec -n 2 valgrind --tool=callgrind --cache-sim=yes \

--separate-threads=yes bin/bt-mz\_A.2"

(verification with fail as it assumes 200 iterations)

- Ioad all profile dumps at once:
  - in directory you started mpiexec: "qcachegrind callgrind.out.\*"

VIRTUAL INSTITUTE - HIGH PRODUCTIVITY SUPERCOMPUTING

Josef Weidendorfer LRZ weidendo@lrz.de