NTNU – Trondheim
Norwegian University of
Science and Technology

Optimizing and Tools

Rune E. Jensen

PhD. Advisor: Anne C. Elster

HPC-Lab
Computer & Info. Science
Norwegian University of
Science and Technology
http://research.idi.ntnu.no/hpc-lab

Introduction

- Goal
  - Teach profiling and debugging
- Give hints about what matters
  - My opinion
  - Few details
- Look at some tools
  - Valgrind
  - Perf (short)
- Hands on
  - Next exercise

Outline

- **Optimization**
- Predictable performance
- Benchmarking
- Tools of the trade
- Valgrind - Howto
- Exercises :(

Optimization

- Strategies
  - Use -O3 -march=native
  - Rapid trial&error
  - Comment out key parts
  - Break correctness
  - Tools (valgrind, perf, vtune, Visual Profiler - CUDA)
- Problems
  - Multiple overlapping bottlenecks
  - Can become slower before speed-up
- 'Homework'
  - http://wiki.cs.utexas.edu/rvdg/HowToOptimizeGemm/
  - Excellent
  - No time today

Optimization

- Optimization
  - Faster is better :)
  - Program or programming?
- It is hard
  - But only after the basics are in place
  - Changes every HW generation
  - Too many factors to list
  - Practical experience needed

Optimization

- Basics
  - Work must match hardware capabilities (find them!!!)
- General overview
  - Cache and data locality in large data structures
  - Correct data type (changing float <--> integer can cost time)
  - Division, pow, exponential, square-root – needed every time??
- Branches
  - CPU pipelines operations, but branches mess up.
  - Prediction used to select a branch anyway, but might fail
  - if(a > b && speed == slow)
    - Write in a way that do not need branches?
  - else
    - Rewrite to help prediction?

Outline

- Optimization
- **Predictable performance**
- Benchmarking
- Tools of the trade
- Valgrind - Howto
- Exercises :(

Predictable Performance

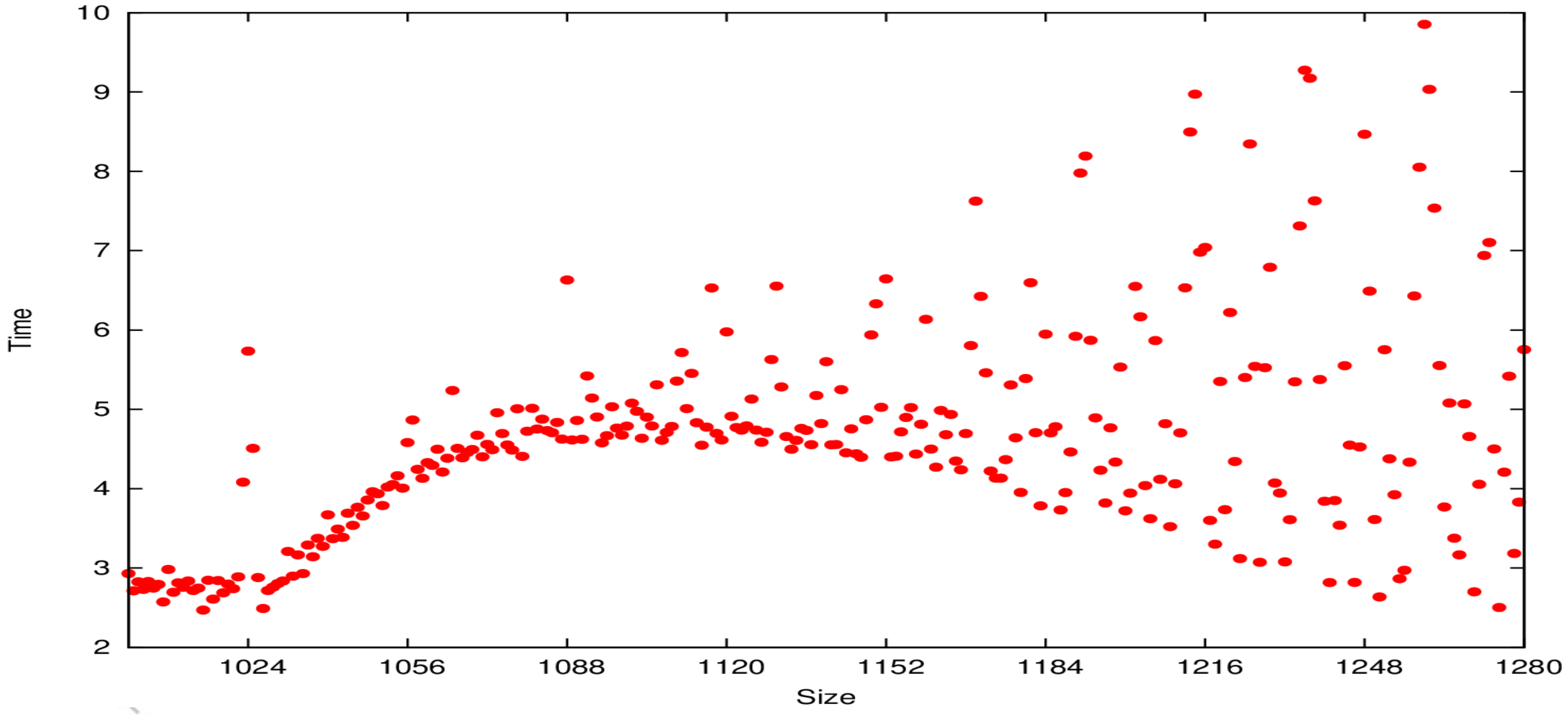
- Work proportional to time spent
- Same work – same time
- Correlation Needed
  - At least to some degree

Predictable Performance

- Issues
  - What makes things hard to predict?
- Algorithms
  - Sensitive to minor changes
  - Data alignment
  - Cache
- Compilers
  - A black magic box
- Bias
  - Random performance?

Predictable Performance

- 2D filter – identical work – different padding

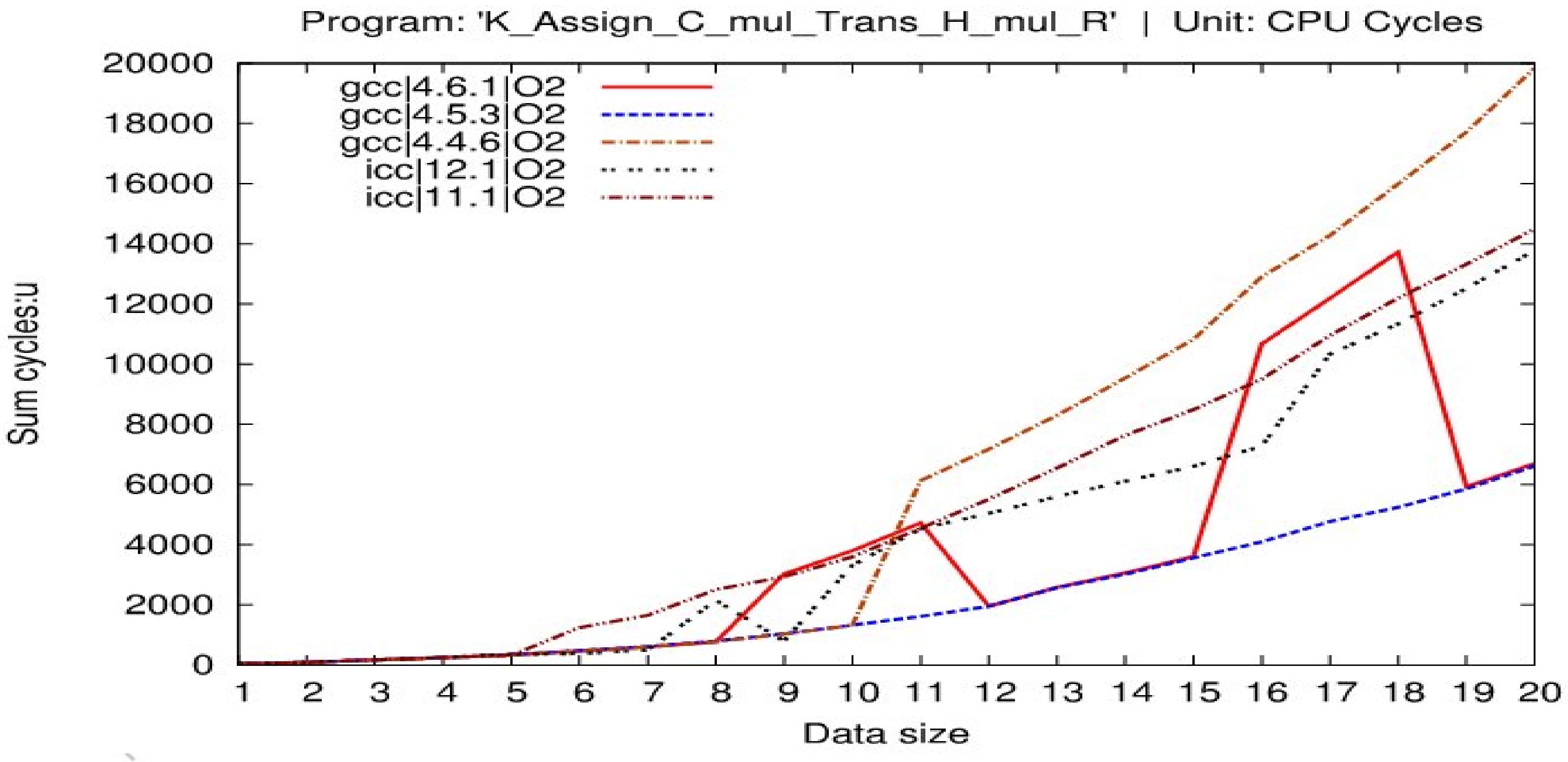


Compilers

- Translates high-level code to machine code
- Semi random code generation quality
- Every compiler version generates different code
- Newer versions not always better
- Complex optimization rules counter-productive*
  - The basics must work first!
- New processors not modeled efficiently

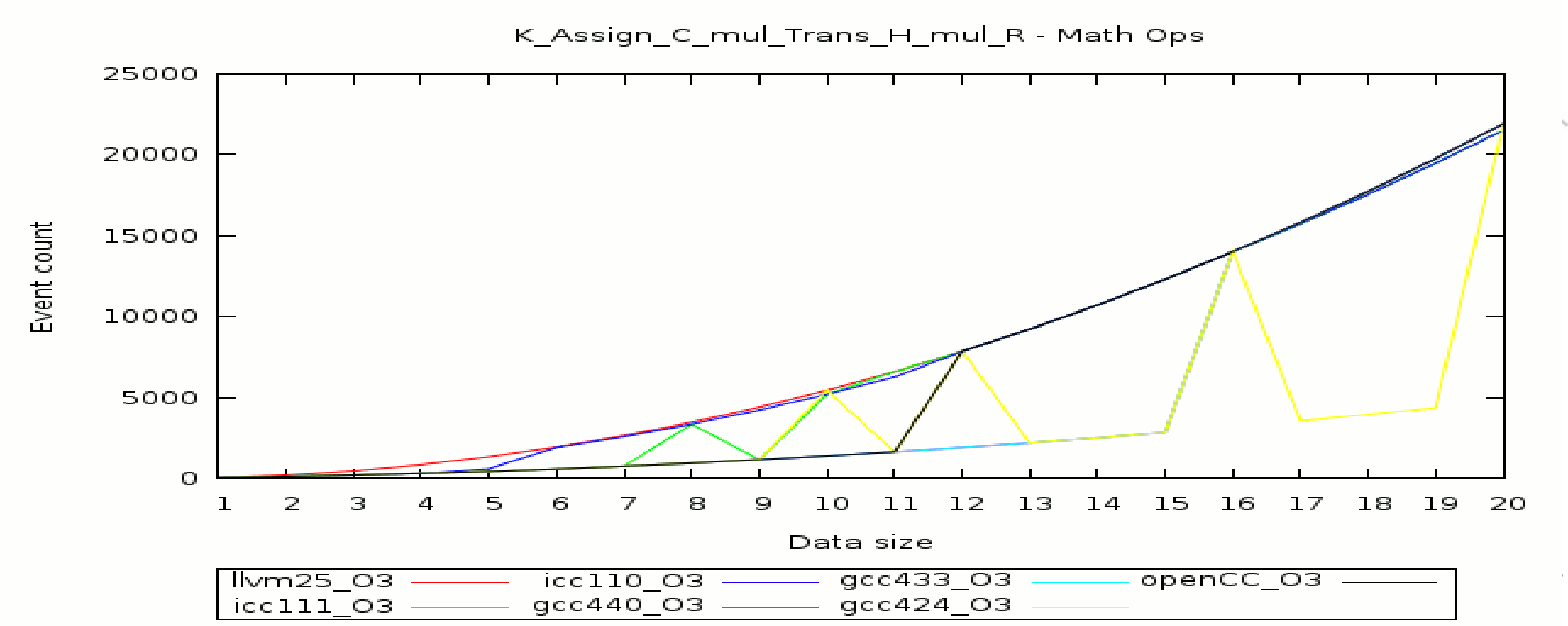
Compilers

- K = C ∗ H^T ∗ R (matrix multiplication)



Compilers

- Underlying analysis
  - Count math instructions with 'perf'

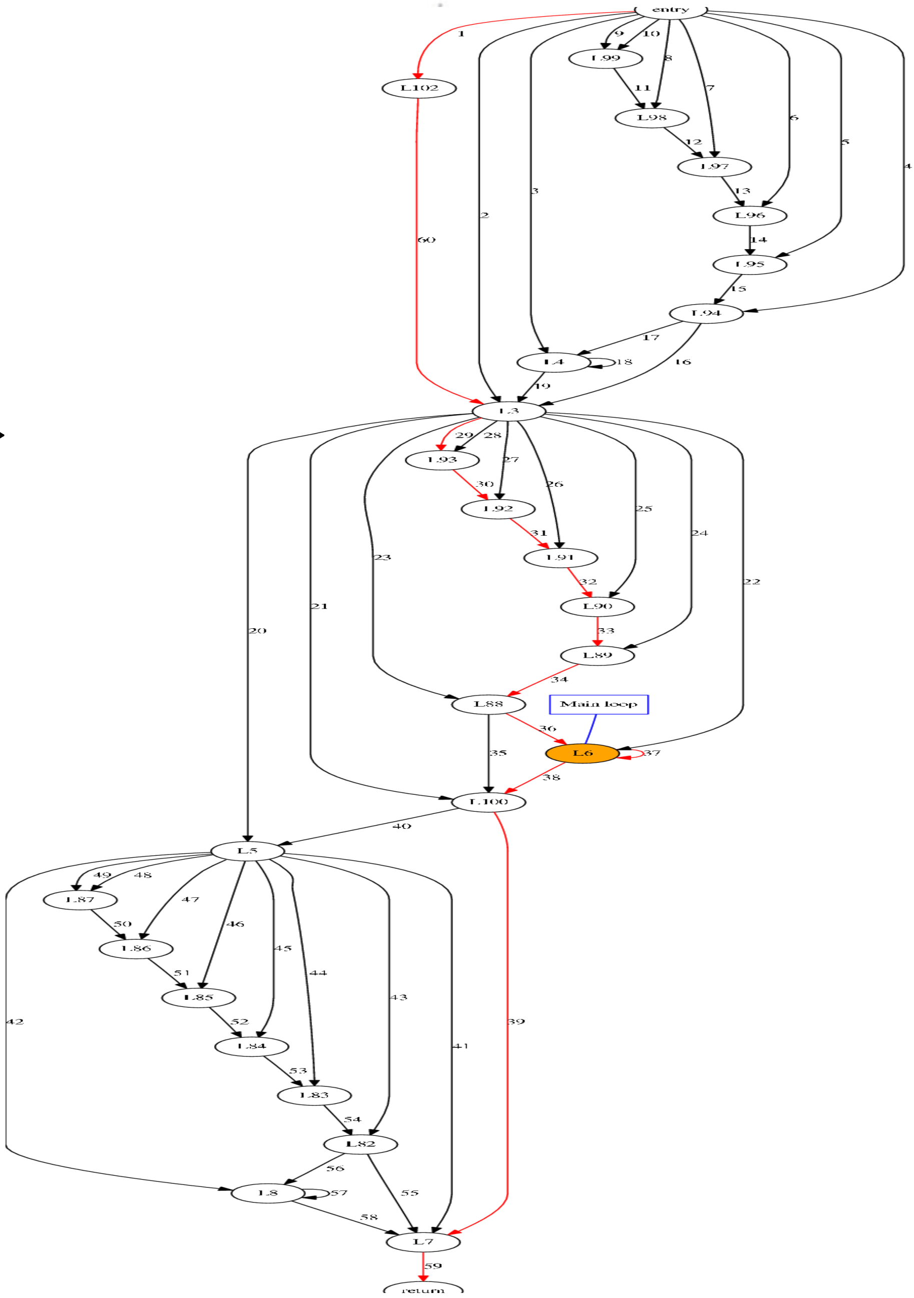


Compilers

- Assigning ”0” to an array →
  - 1024 Elements
  - Float data type

```
float data[1024];
for (int i=0; i<1024; i++)
  data[i] = 0;
```

- Found 17 Compiler issues!
- Can be performed easily
  - Some GCC versions do it
  - (4 Instructions needed)

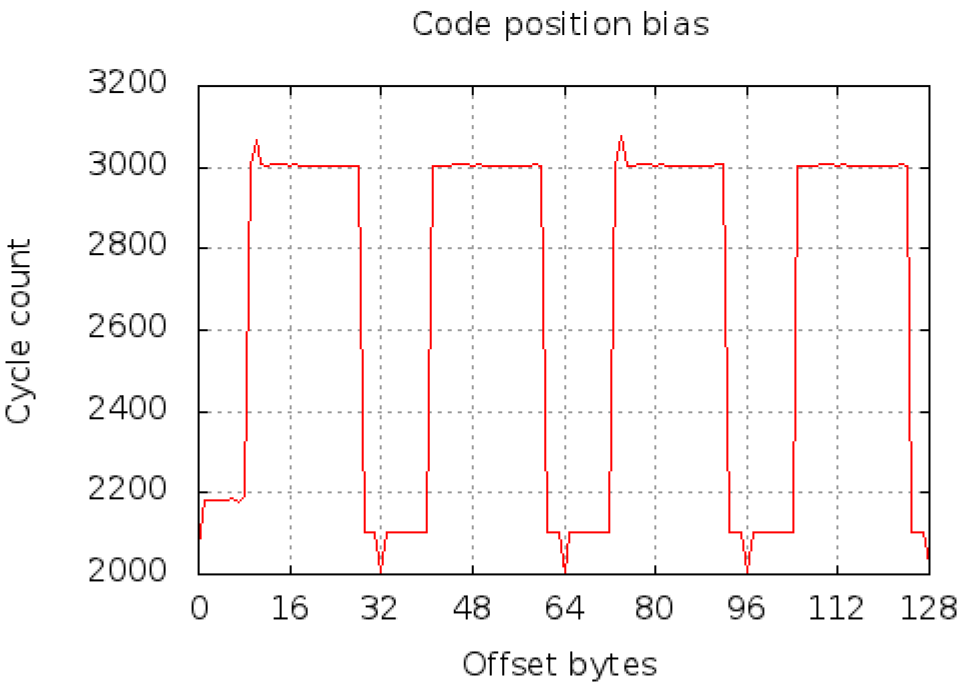


Bias and Randomness

- Bias
  - “Systematic errors are biases in measurement which lead to the situation where the mean of many separate measurements differs significantly from the actual value of the measured attribute.” -Wikipedia
- Compiler effects
  - Rule matching!
  - Tiny changes?
  - Code position?
  - Link order (the sequence of combining multi-file programs)
- Effect of system environment?
  - Environment variables (20% runtime bias)
  - Clustis3 (~50% runtime bias)
    - Two memory speed grades on same node

Bias and Randomness

- Memory randomization
  - Security feature
- Cache line offset
  - Forwarding/bank
- Code position
  - Instruction cache line offset
  - Instrumentation?
  - printf(...)
- Environmental variables
  - Modifies variable layout in memory (stack)
  - Your user name can affect performance



Bias!

- Bias
  - Literature survey of 133 recent papers [T. Mytkowicz *et. al.*]
  - None account for bias adequately :(
  - Bias > median speedup
- Timing is hard
  - 'Too many' error sources :(
  - But it averages out over longer periods

Outline

- Optimization
- Predictable performance
- **Benchmarking**
- Tools of the trade
- Valgrind - Howto
- Exercises :(

Benchmarking

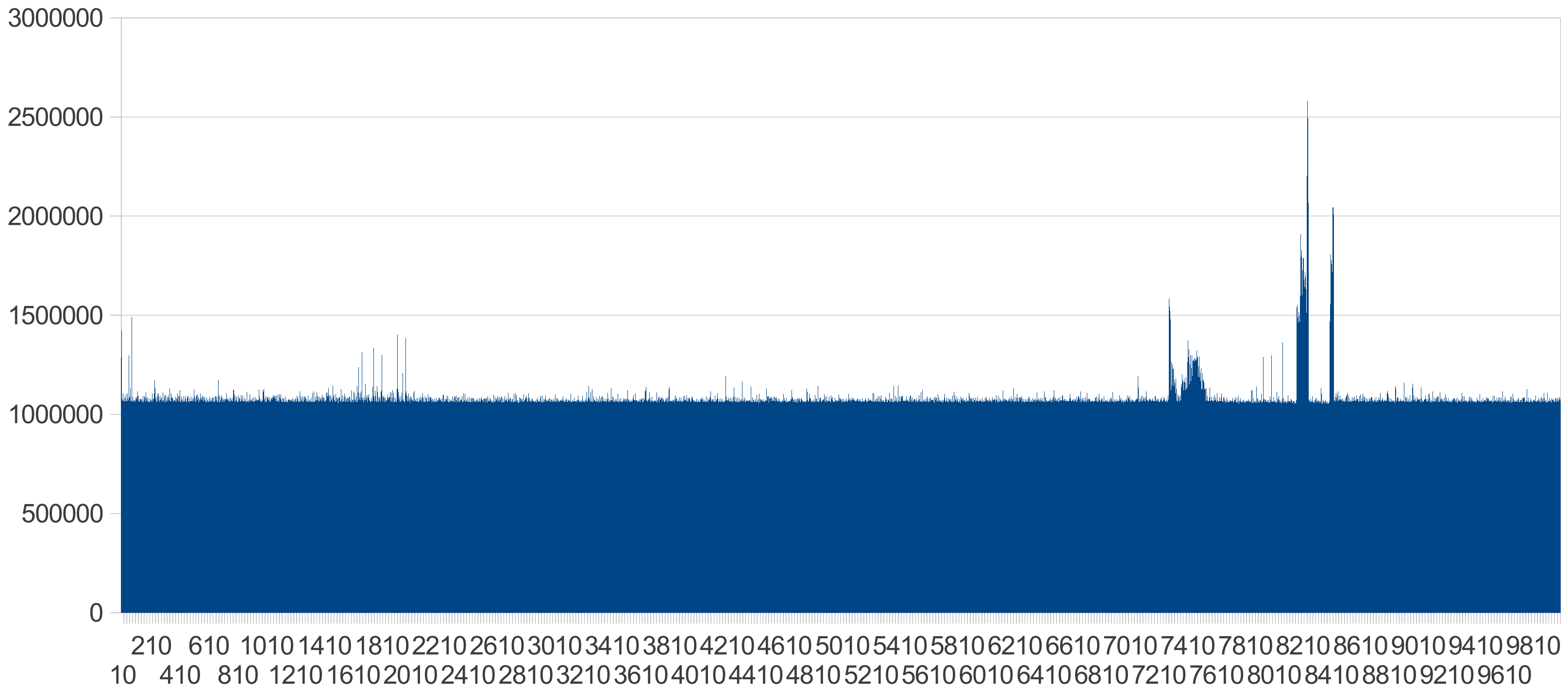
- Standard test to measure performance (eg. SPEC)
- How to benchmark
- Speed stepping & Turbo boost
  - Disable in BIOS/Software
- Address space layout randomization
  - sysctl -w kernel.randomize_va_space=0
  - setarch x86_64 -R ./bin/myprog
- Hyper threading
  - Disable in BIOS
- Affinity
  - taskset -c 0 ./bin/myprog
  - taskset -c 3-4 ./bin/myprog

Benchmarking

- Timers
  - time ./myprogram
  - gettimeofday() or RDTSC
- Precise Measurements
  - *Hardware Performance Counters*
  - Cycle exact
  - Rich metrics (~1000 different types)
  - No overhead or observer effect*
  - Only a few measured at a time (3 fixed + 4-8 generic)
- Valid Measurements
  - What about accuracy?
  - What is measured?
  - Bias & precise noise?

Runtime Test

- Time dependent noise?
  - Same program 10k times in a series



Outline

- Optimization
- Predictable performance
- Benchmarking
- **Tools of the trade**
- Valgrind - Howto
- Exercises :(

Tools of the trade

- How to debug and optimize
  - Less bugs + better understanding
  - =
  - More time for optimizing programs

Tools of the trade

- How to debug and optimize
  - Less bugs + better understanding
  - =
  - More time for optimizing programs
- High overhead instrumentation
  - Valgrind
- Medium overhead instrumentation
  - Pin
  - Sniper
- Low overhead instrumentation
  - Perf (performance bugs)

Tools of the trade

- Valgrind overview
  - Swiss army knife for programmers
  - Software CPU simulator
  - Multiple tools
  - Free
  - Linux based
- Cachegrind
  - A cache and branch-prediction profiler
- Callgrind
  - A call-graph generating cache and branch-prediction profiler
- Memcheck
  - A memory error detector

Tools of the trade

- Valgrind overview (2)
- Massif
  - A heap (malloc'ed memory) usage profiler
- DHAT
  - Another heap profiler
- Helgrind
  - Inconsistent Lock Ordering checker (pthreads)
- DRD
  - Another thread error detector
- SGCheck
  - An experimental stack and global array overrun detector

Tools of the trade

- PIN (native JIT compiler)
  - Instruction count (validation)
  - Ins. types w. count
  - Ins. register usage
  - Ins. lengths
  - Code coverage analyzer
  - Runtime load alignment tester
  - Test new instructions (SSE6, AVX3, my-own)
- Sniper (OoOE CPU simulator)
  - Cycle count
  - Energy metrics
  - ALU, ifetch, mem, icache, ..., dram
  - Adjustable architecture (Intel)

Tools of the trade

- Perf
  - Reads Hardware Performance Counters
  - Needs a new Linux kernel
  - Minimal overhead
  - Can filter out OS/kernel overhead
  - Follows on CPU migrations
  - Handles frequency scaling & turbo boost (better)
  - Precise
  - Rich metrics
  - Minimal observer effect*
- Easy to test/use
  - perf stat ./myprogram
  - perf stat -e instructions:u,cycles:u ./myprogram

Tools of the trade

- Some problems with 'perf'
  - Rich metrics
  - Only a few measured at a time (3 fixed + 4-8 generic)
  - Hard to understand
  - Hard to exploit
  - Bad documentation
  - CPU architecture specific
  - Might have strange bugs

Outline

- Optimization
- Predictable performance
- Benchmarking
- Tools of the trade
- **Valgrind - Howto**
- Exercises :(

Valgrind

- What can it do for you?
  - Help optimizing
  - Find bugs
  - Peace of mind

Valgrind

- What can it do for you?
  - Help optimizing
  - Find bugs
  - Peace of mind
- Alternative motivation
  - Exercises

Valgrind

- Memcheck: a memory error detector
  - The default tool
  - --tool=memcheck (not needed)
- Finds
  - Memory leaks
  - Using undefined values (variables without assigned value)
  - Accessing memory you shouldn't
    - Overrunning and underrunning heap (malloc'ed memory) blocks
    - Overrunning the top of the stack (~bad pointer access)
    - Accessing memory after it has been freed.
  - Incorrect freeing
- Usage
  - valgrind ./myprog

Valgrind

- 'valgrind ls'
  - ==24741== HEAP SUMMARY:
  - ==24741==   in use at exit: 27,512 bytes in 36 blocks
  - ==24741==   total heap usage: 71 allocs, 35 frees, 64,723 bytes allocated
  - ==24741== LEAK SUMMARY:
  - ==24741==   definitely lost: 0 bytes in 0 blocks
  - ==24741==   indirectly lost: 0 bytes in 0 blocks
  - ==24741==   possibly lost: 0 bytes in 0 blocks
  - ==24741==   still reachable: 27,512 bytes in 36 blocks
  - ==24741==   suppressed: 0 bytes in 0 blocks
  - ==24741== Rerun with --leak-check=full to see details of leaked memory

Valgrind

- Cachegrind: a cache and branch-prediction profiler
  - Performance evaluation tool
  - Profile I1, D1 and LL (last-level) caches
  - Cache effectiveness
  - Miss-predicted branches
  - Makes detailed output file: cachegrind.out.*pid*
    - *Pid* is a “random” number
  - Dedicated viewer*
- Usage
  - valgrind --tool=cachegrind ./myprog

Valgrind

- 'valgrind --tool=cachegrind ls'

```
– ==27980== I   refs:      565,805
– ==27980== I1  misses:      1,643
– ==27980== LLi misses:      1,526
– ==27980== I1  miss rate:    0.29%
– ==27980== LLi miss rate:    0.26%
– ==27980== D   refs:      203,926  (145,847 rd   + 58,079 wr)
– ==27980== D1  misses:      5,554  (  4,350 rd   +  1,204 wr)
– ==27980== LLd misses:      3,864  (  2,775 rd   +  1,089 wr)
– ==27980== D1  miss rate:     2.7% (    2.9%     +    2.0%  )
– ==27980== LLd miss rate:     1.8% (    1.9%     +    1.8%  )
– ==27980== LL refs:         7,197  (  5,993 rd   +  1,204 wr)
– ==27980== LL misses:       5,390  (  4,301 rd   +  1,089 wr)
– ==27980== LL miss rate:      0.7% (    0.6%     +    1.8%  )
```

Valgrind

- Callgrind: a call-graph generating cache and branch prediction profiler
  - Like cachegrind, only better
    - And slower
  - Excellent for understanding code written by others*
  - Makes detailed output file: callgrind.out.*pid*
  - Dedicated viewer*
- Usage
  - valgrind --tool=callgrind ./myprog
  - valgrind --tool=callgrind --branch-sim=yes --cache-sim=yes --simulate-hwpref=yes --dump-instr=yes --collect-jumps=yes --cacheuse=yes ./myprog

Valgrind

- Massif: Heap usage overview
  - Memory usage graph
  - Memory overhead cost
  - Overview tool
  - Text based only
  - Makes detailed output file: massid.out.*pid*
  - Pretty printer tool needed
- Usage
  - valgrind --tool=massif ./myprog
  - ms_print massif.out.12345 | less

Valgrind

- valgrind --tool=massif ./myprog
  - ms_print massif.out.12345 | less

```
KB
55.90^                         :#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#
   |                       :@:#     ::::::::::::::::::
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                       :@:#::::::::::   : @
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                       :@:#     :       : @
   |                     @:@::@:#      :       : @
  0 +--------------------------------------------------------------------------->ki
   0                                          531.6
```

Valgrind

- DHAT: Heap profiler
  - Finds inefficient memory usage
  - Profiles every malloc/new independently
  - Access (read/write) counts
  - Allocation lifetime (detailed)
  - Unused memory (unread/unwritten)
  - Counts accesses per offset (malloc's of 4KB or less)
  - Text based only
- Usage
  - valgrind --tool=exp-dhat ./myprog

Valgrind

- 'valgrind --tool=exp-dhat ls'
- ==6369== max-live: 32,808 in 1 blocks
- ==6369== tot-alloc: 32,808 in 1 blocks (avg size 32808.00)
- ==6369== deaths: 1, at avg age 16,290 (2.92% of prog lifetime)
- ==6369== **acc-ratios: 0.13 rd, 0.07 wr (4,343 b-read, 2,425 b-written)**
- ==6369==   at 0x4C2928F: malloc (vg_replace_malloc.c:270)
- ==6369==   by 0x55068C0: __alloc_dir (opendir.c:186)
- ==6369==   by 0x407F5E: ??? (in /bin/ls)
- ==6369==   by 0x547B30C: (below main) (libc-start.c:226)

Valgrind

- 'valgrind --tool=exp-dhat ls'
- Aggregated access counts by byte offset:
- ==6369==
- ==6369== [0] 81 81 54 54 54 54 54 50 50 50 49 49 46 43 35 35
- ==6369== [16] 8 7 6 5 4 4 4 4 4 4 4 4 4 4 4 2
- ==6369== [32] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- ==6369== [48] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- ==6369== [64] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- ==6369== [80] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- ==6369== [96] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- ==6369== [112] 0 0 0 0 0 0 0 0

Valgrind

- Helgrind & DRD
  - Improper use of the POSIX threads API.
  - Inconsistent Lock Orderings
  - Data races
    - data-race free if all conflicting memory accesses are ordered by synchronization operations.
  - Lock contention
  - pthreads only (POSIX)
  - OpenMP – GCC recompile needed :(
  - Text based only
- Usage
  - valgrind --tool=helgrind ./myprog
  - valgrind --tool=drd ./myprog

Valgrind

- SGCheck: an experimental stack and global array overrun detector
  - Experimental
  - More error checking and bug hunting
  - Listed for completeness
- Usage
  - valgrind --tool=exp-sgcheck ./myprog

Valgrind

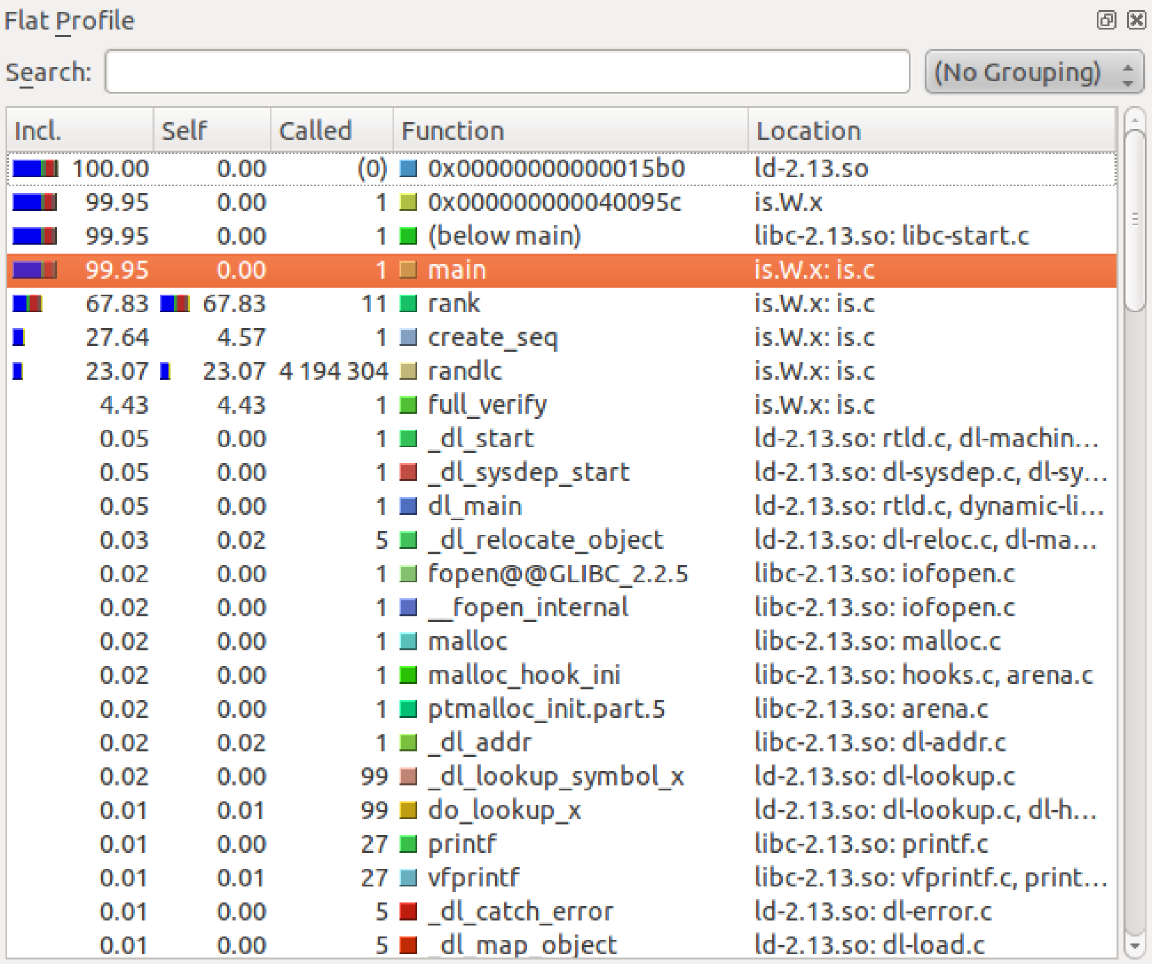
- Preparing your program
  - Some compiler flags **WILL** help Valgrind
  - -g – add debug information: Source code link and names
  - -O0 – Preserves structure, but less correct performance data
  - -O1 – Balance between structure and performane
    - -O2 and -O3 might remove many function calls
  - -fno-inline – turn off one code structure removal feature (with O2/3)
- Usage
  - gcc -g -O2 -fno-inline main.c -o myprogram main.c
    - = add descriptions, optimize and preserve structure

KCachegrind

- Valgrind visualization tool
  - Linux: Kcachegrind
  - Windows: Qcachegrind
    - Several non-working ports?
  - Makes nice graps and images

KCachegrind

- 3 windows
- Left: list of functions
  - (no grouping)
  - ELF object
  - Source file
  - Updates color group
- Sorts on metric
  - Ex. Instruction count



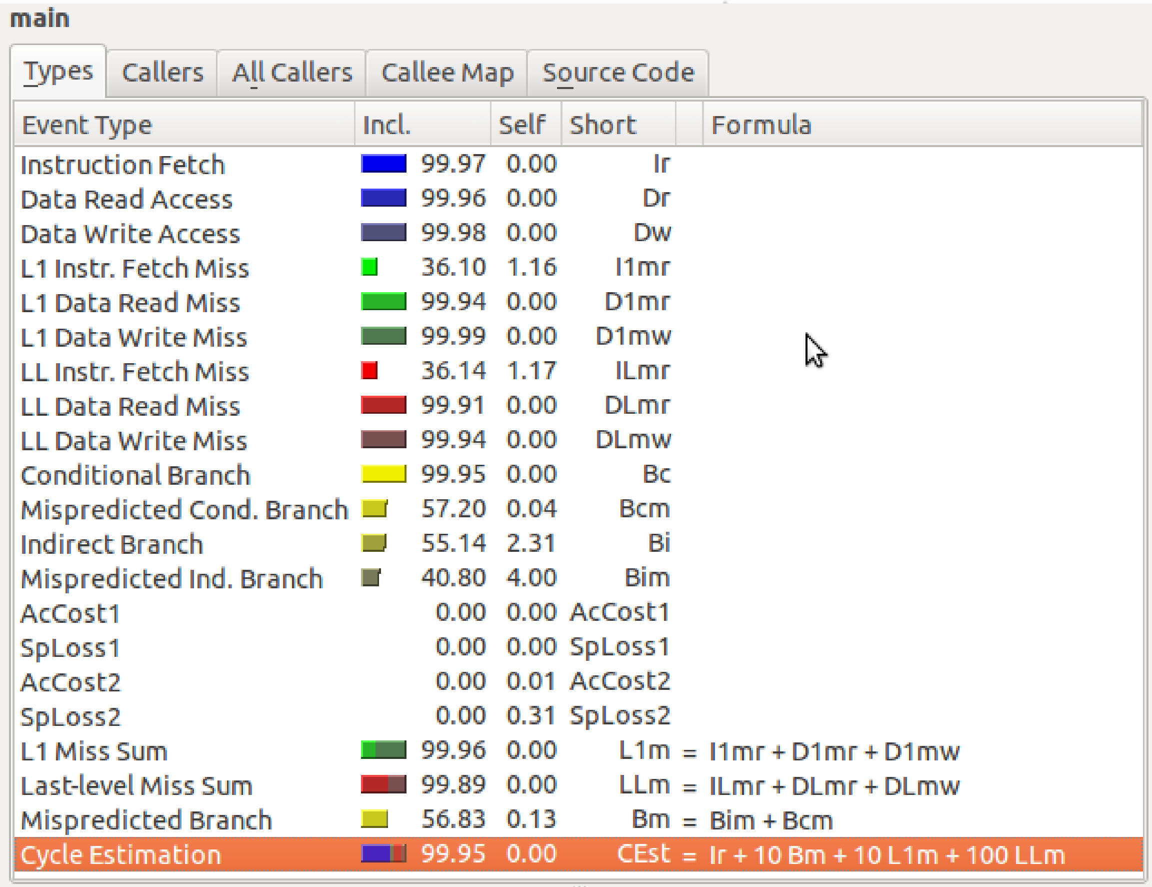
Search:

(No Grouping)

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000000000015b0	ld-2.13.so
99.95	0.00	1	0x000000000040095c	is.W.x
99.95	0.00	1	(below main)	libc-2.13.so: libc-start.c
99.95	0.00	1	main	is.W.x: is.c
67.83	67.83	11	rank	is.W.x: is.c
27.64	4.57	1	create_seq	is.W.x: is.c
23.07	23.07	4 194 304	randlc	is.W.x: is.c
4.43	4.43	1	full_verify	is.W.x: is.c
0.05	0.00	1	_dl_start	ld-2.13.so: rtld.c, dl-machin…
0.05	0.00	1	_dl_sysdep_start	ld-2.13.so: dl-sysdep.c, dl-sy…
0.05	0.00	1	dl_main	ld-2.13.so: rtld.c, dynamic-li…
0.03	0.02	5	_dl_relocate_object	ld-2.13.so: dl-reloc.c, dl-ma…
0.02	0.00	1	fopen@@GLIBC_2.2.5	libc-2.13.so: iofopen.c
0.02	0.00	1	__fopen_internal	libc-2.13.so: iofopen.c
0.02	0.00	1	malloc	libc-2.13.so: malloc.c
0.02	0.00	1	malloc_hook_ini	libc-2.13.so: hooks.c, arena.c
0.02	0.00	1	ptmalloc_init.part.5	libc-2.13.so: arena.c
0.02	0.02	1	_dl_addr	libc-2.13.so: dl-addr.c
0.02	0.00	99	_dl_lookup_symbol_x	ld-2.13.so: dl-lookup.c
0.01	0.01	99	do_lookup_x	ld-2.13.so: dl-lookup.c, dl-h…
0.01	0.00	27	printf	libc-2.13.so: printf.c
0.01	0.01	27	vfprintf	libc-2.13.so: vfprintf.c, print…
0.01	0.00	5	_dl_catch_error	ld-2.13.so: dl-error.c
0.01	0.00	5	_dl_map_object	ld-2.13.so: dl-load.c

KCachegrind

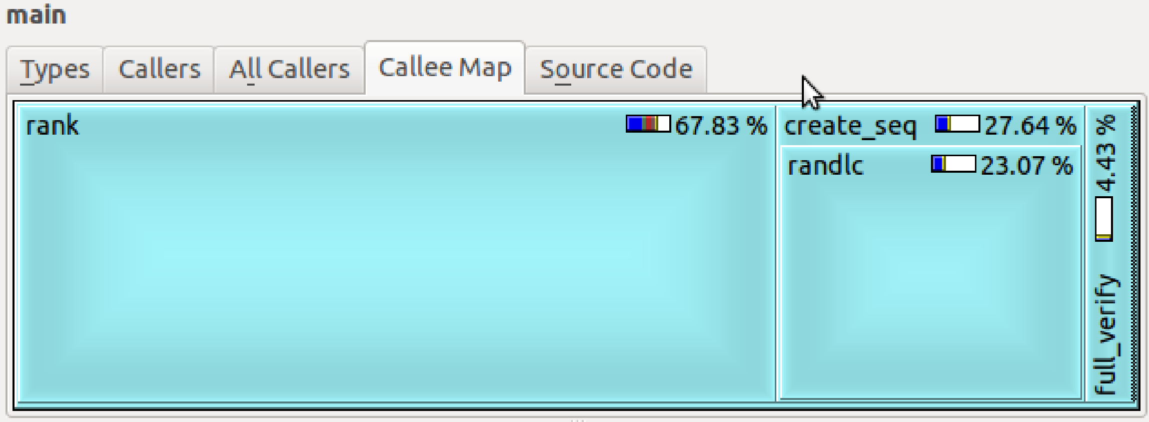
- Top right
  - Types
    - Cost Metrics
    - Instruction fetch
    - Data Read/Write
    - L1/LL cache
    - Branches
    - Sum events
    - Cycle estimation
  - Callee Map
    - Overview - optimize
  - Source Code
    - Shows source code



Event Type	Incl.	Self	Short	Formula
Instruction Fetch	99.97	0.00	Ir	
Data Read Access	99.96	0.00	Dr	
Data Write Access	99.98	0.00	Dw	
L1 Instr. Fetch Miss	36.10	1.16	I1mr	
L1 Data Read Miss	99.94	0.00	D1mr	
L1 Data Write Miss	99.99	0.00	D1mw	
LL Instr. Fetch Miss	36.14	1.17	ILmr	
LL Data Read Miss	99.91	0.00	DLmr	
LL Data Write Miss	99.94	0.00	DLmw	
Conditional Branch	99.95	0.00	Bc	
Mispredicted Cond. Branch	57.20	0.04	Bcm	
Indirect Branch	55.14	2.31	Bi	
Mispredicted Ind. Branch	40.80	4.00	Bim	
AcCost1	0.00	0.00	AcCost1	
SpLoss1	0.00	0.00	SpLoss1	
AcCost2	0.00	0.01	AcCost2	
SpLoss2	0.00	0.31	SpLoss2	
L1 Miss Sum	99.96	0.00	L1m	= I1mr + D1mr + D1mw
Last-level Miss Sum	99.89	0.00	LLm	= ILmr + DLmr + DLmw
Mispredicted Branch	56.83	0.13	Bm	= Bim + Bcm
Cycle Estimation	99.95	0.00	CEst	= Ir + 10 Bm + 10 L1m + 100 LLm

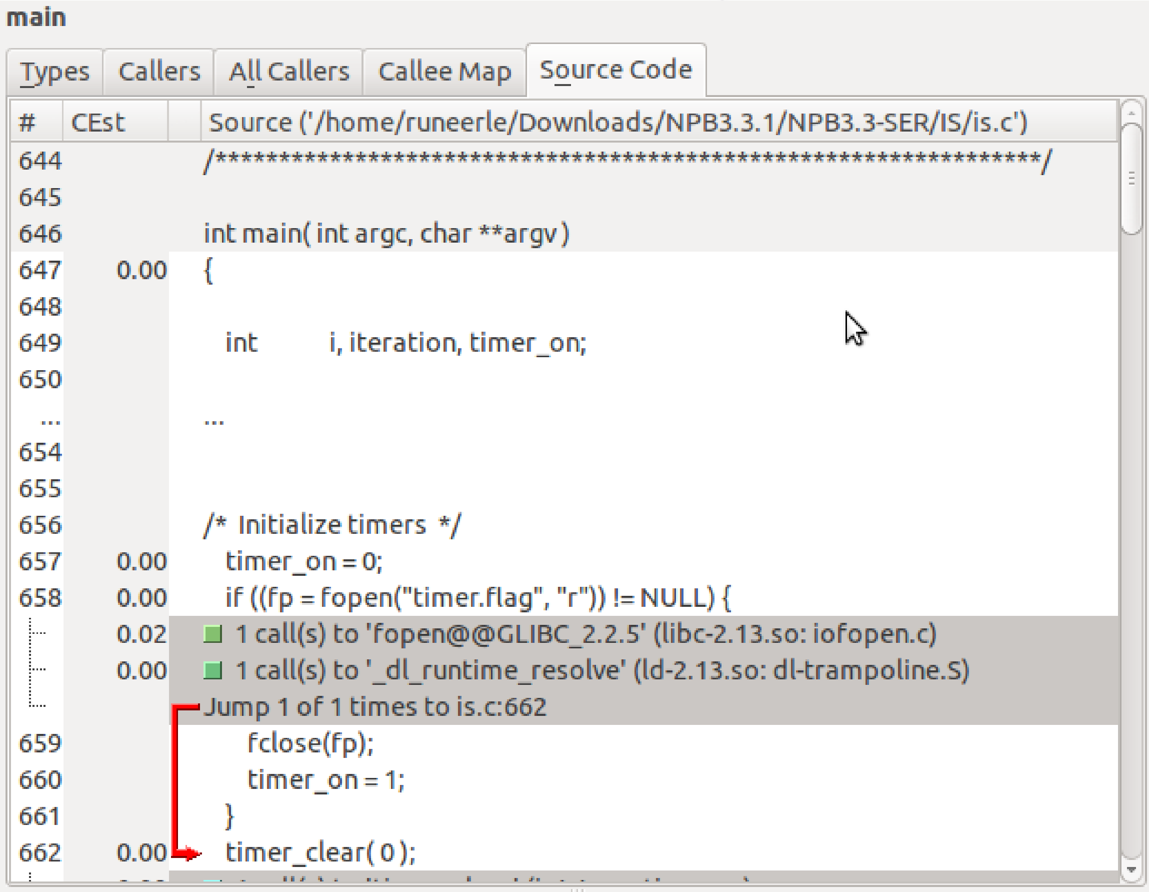
KCachegrind

- Top right
  - Types
    - Cost Metrics
    - Instruction fetch
    - Data Read/Write
    - L1/LL cache
    - Branches
    - Sum events
    - Cycle estimation
  - Callee Map
    - Overview - optimize
  - Source Code
    - Shows source code



KCachegrind

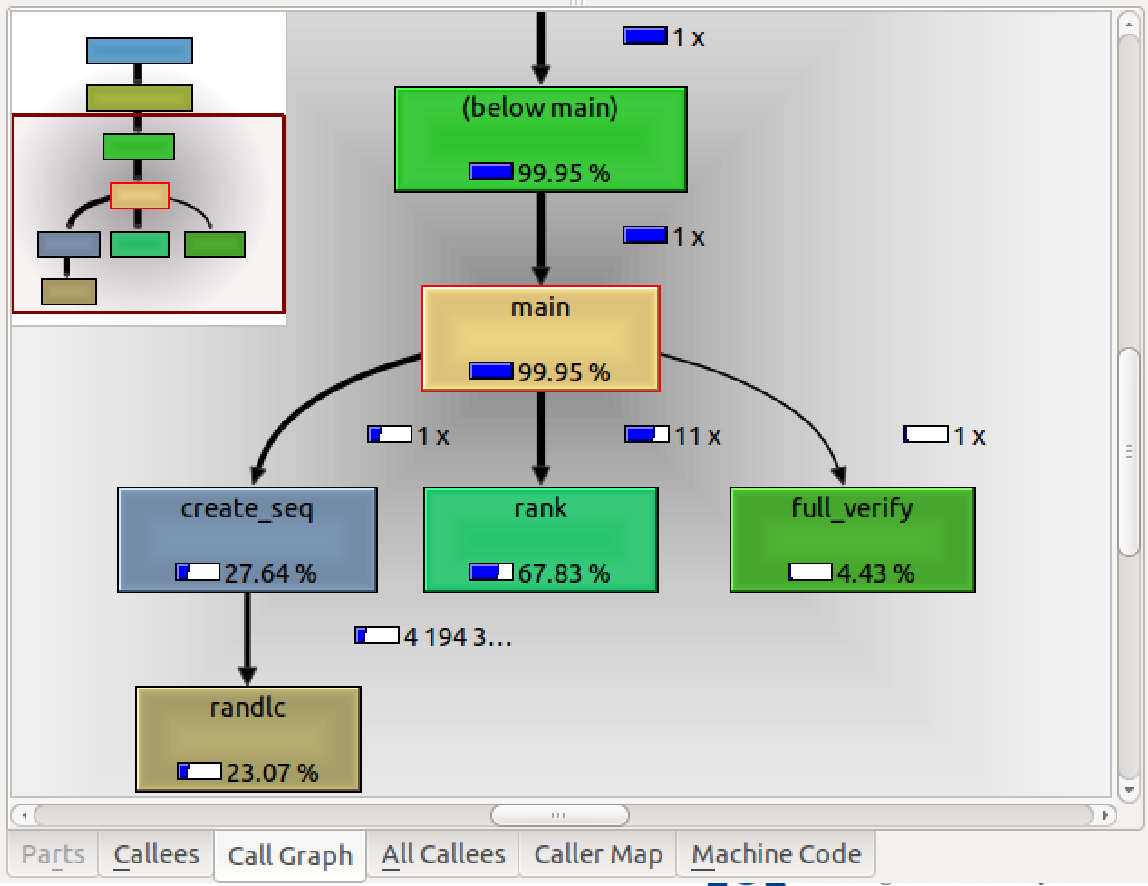
- Top right
  - Types
    - Cost Metrics
    - Instruction fetch
    - Data Read/Write
    - L1/LL cache
    - Branches
    - Sum events
    - Cycle estimation
  - Callee Map
    - Overview - optimize
  - Source Code
    - Shows source code



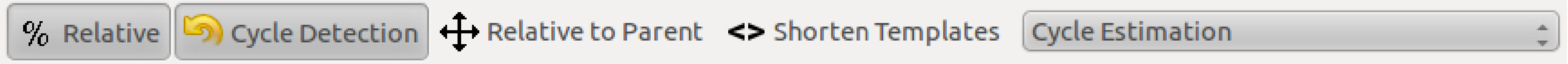
```
#     CEst    Source ('/home/runeerle/Downloads/NPB3.3.1/NPB3.3-SER/IS/is.c')
644           /*****************************************************************/
645
646           int main( int argc, char **argv )
647    0.00   {
648
649             int        i, iteration, timer_on;
650
...           ...
654
655
656           /* Initialize timers */
657    0.00     timer_on = 0;
658    0.00     if ((fp = fopen("timer.flag", "r")) != NULL) {
       0.02   🟩 1 call(s) to 'fopen@@GLIBC_2.2.5' (libc-2.13.so: iofopen.c)
       0.00   🟩 1 call(s) to '_dl_runtime_resolve' (ld-2.13.so: dl-trampoline.S)
              Jump 1 of 1 times to is.c:662
659               fclose(fp);
660               timer_on = 1;
661             }
662    0.00     timer_clear( 0 );
```

KCachegrind

- Bottom right
  - Call Graph
    - Options:
    - Compact/Normal
    - Depth
    - Min cost
  - Machine Code
    - --dump-instr=yes
- ***USE THIS***
  - Gives understanding
  - Code flow
  - Function call count



KCachegrind



- Controls
  - % Relative
    - Absolute counts or percentage of counts?
  - Cycle Detection
    - Call loops (never needed it)
  - Relative to Parent
    - When using % relative only
    - Make current selected node cost 100%, else use % of program total.
  - <> Shorten Templates
    - For C++
  - Cycle Estimation
    - Selected metric

Valgrind/Kcachegrind Q&A

- Questions?
- Comments?

Outline

- Optimization
- Predictable performance
- Benchmarking
- Tools of the trade
- Valgrind - Howto
- Exercises :(

Exercises

- Will be handed out later :|
- Use Valgrind for real
  - Bonus exercise =D
  - Look at the NAS Parallel Benchmarks with Valgrind
  - www.nas.nasa.gov/publications/npb.html

Exercises

- Learn to use Valgrind

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  char *mem = malloc(100);
  mem = "Hello\n";
  printf("%s", mem);
  return 0;
}
```

Q&A

- Questions?
- Comments?

NTNU – Trondheim
Norwegian University of
Science and Technology