TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Cache Profiling with Callgrind

Linux/x86 Performance Practical, 17.06.2009

Zellescher Weg 12

Willers-Bau A106

Tel. +49 351 - 463 - 31945

Ulf Markwardt (ulf.markwardt@tu-dresden.de)

Matthias Lieber (matthias.lieber@tu-dresden.de)

ZIH

Center for Information Services & High Performance Computing

Valgrind

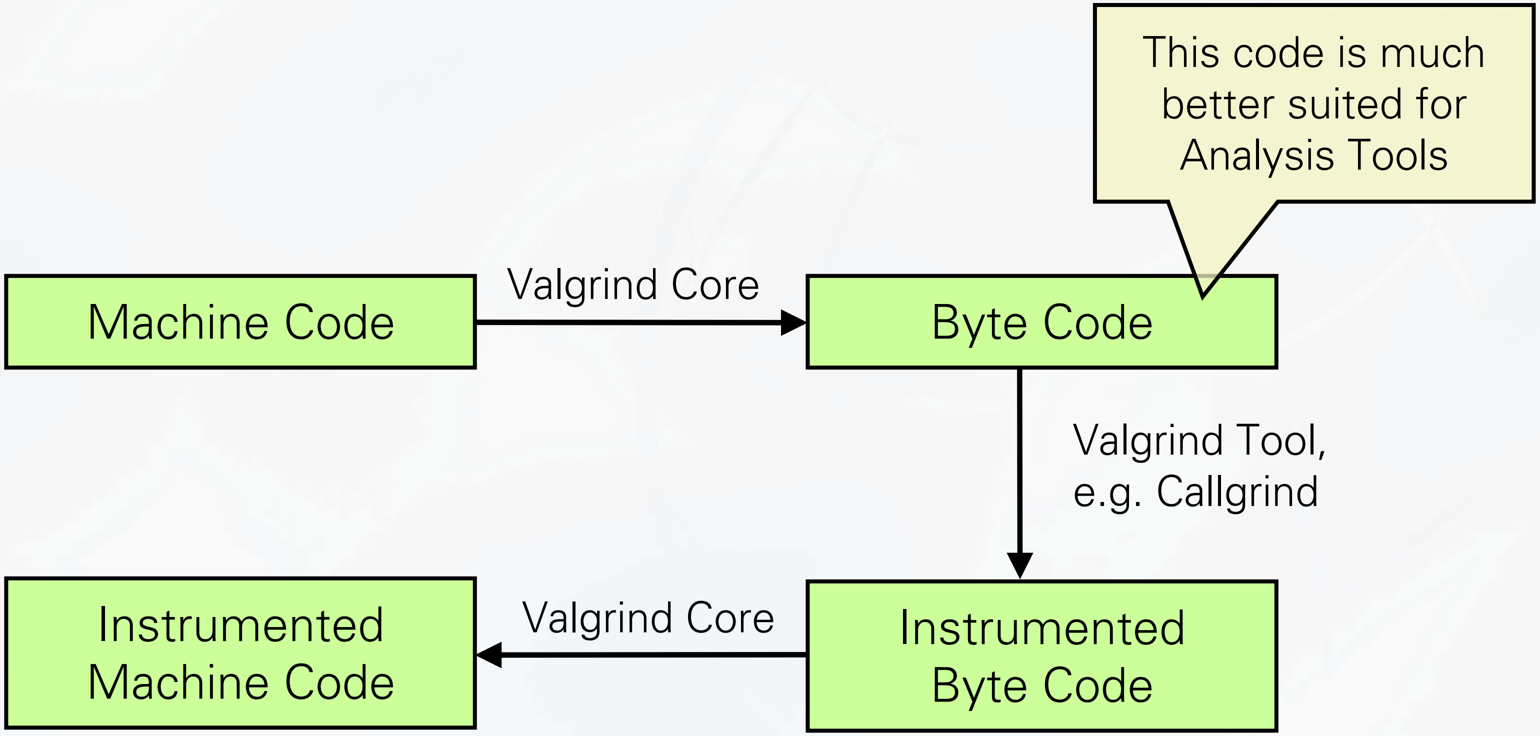
- Suite of simulation-based debugging and profiling tools
- Valgrind core simulates a CPU in software
- Tools implement various tasks by adding analysis code
- Available for Linux on x86 and PowerPC platforms (both 32/64 bit)
- Open source
- Standard Linux package
- Wide acceptance, e.g. Firefox, OpenOffice, KDE use Valgrind
- http://www.valgrind.org

Valgrind's Tool Suite

- Memcheck = "Valgrind"
  - Most prominent, detects memory management bugs
- Cachegrind
  - Cache profiler, finds source location of cache misses
- **Callgrind**
  - Cachegrind + function call graph information
- Massif
  - Heap profiler, where / how much memory allocations?
- Helgrind
  - Thread debugger

How does Valgrind work?

- Dynamic recompilation of program's binary at runtime
- Original code never runs directly on CPU
- **Large overhead**: 10 – 100 times slower, depending on tool



Cachegrind, Callgrind

- **Cachegrind** collects statistics about cache misses
- Simulates L1i, L1d, inclusive L2 cache
- Size of the caches can be specified, default is current machine’s cache
- Output:
  - Total program run hit/miss count and ratio
  - Per function hit/miss count
  - Per source code line hit/miss count

Very useful for performance tuning

- **Callgrind** is an extension of Cachegrind
  - Additional function call graph information

Callgrind Example: Program run under Callgrind

mlieber@phobos:~/loops

```
mlieber@phobos:~/loops> make
gcc -O2 -g -c loops-fast.c -o loops-fast.o
gcc -O2 -g -o loops-fast loops-fast.o
gcc -O2 -g -c loops-slow.c -o loops-slow.o
gcc -O2 -g -o loops-slow loops-slow.o
mlieber@phobos:~/loops> module load valgrind
Valgrind 3.4.1 loaded
Refer to http://valgrind.org/ for more info about the Valgrind tool suite.
mlieber@phobos:~/loops> valgrind --tool=callgrind --simulate-cache=yes ./loops-fast
==7827== Callgrind, a call-graph generating cache profiler.
==7827== Copyright (C) 2002-2008, and GNU GPL'd, by Josef Weidendorfer et al.
==7827== Using LibVEX rev 1884, a library for dynamic binary translation.
==7827== Copyright (C) 2004-2008, and GNU GPL'd, by OpenWorks LLP.
==7827== Using valgrind-3.4.1, a dynamic binary instrumentation framework.
==7827== Copyright (C) 2000-2008, and GNU GPL'd, by Julian Seward et al.
==7827== For more details, rerun with: -v
==7827==
==7827== For interactive control, run 'callgrind_control -h'.
sum =  10000.000
==7827==
==7827== Events    : Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
==7827== Collected : 8144314 1031520 1013097 783 126671 125566 781 126529 125546
==7827==
==7827== I   refs:      8,144,314
==7827== I1  misses:          783
==7827== L2i misses:          781
==7827== I1  miss rate:       0.0%
==7827== L2i miss rate:       0.0%
==7827==
==7827== D   refs:      2,044,617  (1,031,520 rd + 1,013,097 wr)
==7827== D1  misses:      252,237  (  126,671 rd +   125,566 wr)
==7827== L2d misses:      252,075  (  126,529 rd +   125,546 wr)
==7827== D1  miss rate:      12.3% (     12.2%   +      12.3%  )
==7827== L2d miss rate:      12.3% (     12.2%   +      12.3%  )
==7827==
==7827== L2 refs:         253,020  (  127,454 rd +   125,566 wr)
==7827== L2 misses:       252,856  (  127,310 rd +   125,546 wr)
==7827== L2 miss rate:        2.4% (      1.3%   +      12.3%  )
mlieber@phobos:~/loops> █
```

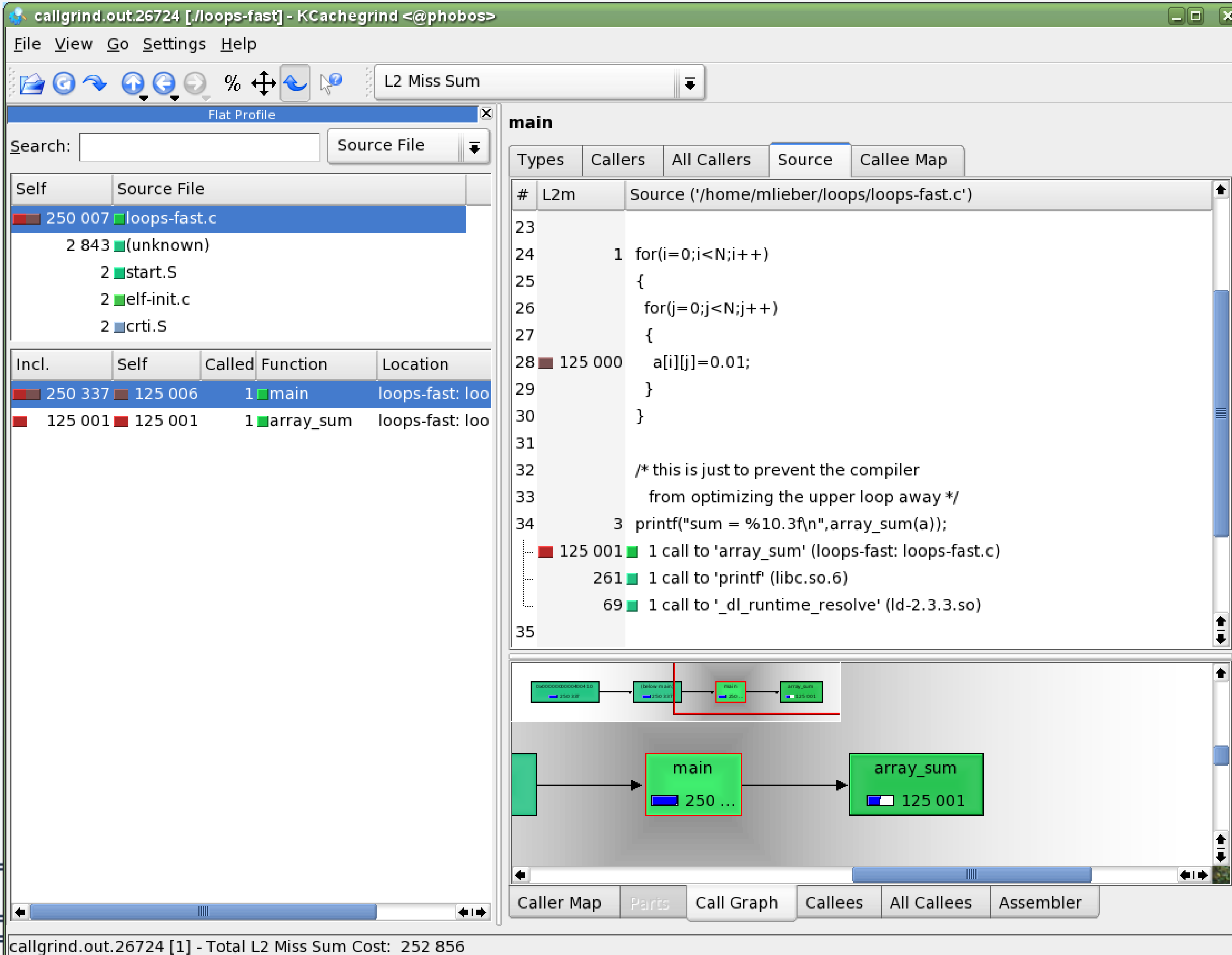
Callgrind Example: callgrind_annotate

```
mlieber@phobos:~/loops> ls -l
total 84
-rw-------  1 mlieber zih 22776 2009-06-08 18:11 callgrind.out.7827
-rwxr-xr-x  1 mlieber zih 14208 2009-06-08 18:10 loops-fast
-rw-r--r--  1 mlieber zih   559 2009-06-08 14:39 loops-fast.c
-rw-r--r--  1 mlieber zih  6952 2009-06-08 18:10 loops-fast.o
-rwxr-xr-x  1 mlieber zih 14208 2009-06-08 18:10 loops-slow
-rw-r--r--  1 mlieber zih   559 2009-06-08 14:39 loops-slow.c
-rw-r--r--  1 mlieber zih  6944 2009-06-08 18:10 loops-slow.o
-rw-r--r--  1 mlieber zih   275 2009-06-08 14:39 Makefile
mlieber@phobos:~/loops> callgrind_annotate callgrind.out.7827
--------------------------------------------------------------------------------
Profile data file 'callgrind.out.7827' (creator: callgrind-3.4.1)
--------------------------------------------------------------------------------
I1 cache: 65536 B, 64 B, 2-way associative
D1 cache: 65536 B, 64 B, 2-way associative
L2 cache: 1048576 B, 64 B, 8-way associative
Timerange: Basic block 0 - 2026375
Trigger: Program termination
Profiled target:  ./loops-fast (PID 7827, part 1)
Events recorded:  Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
Events shown:     Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
Event sort order: Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
       Ir        Dr        Dw I1mr    D1mr    D1mw I2mr    D2mr    D2mw
--------------------------------------------------------------------------------
8,144,317 1,031,520 1,013,097  783 126,671 125,566  781 126,529 125,546  PROGRAM TOTALS

--------------------------------------------------------------------------------
       Ir        Dr        Dw I1mr    D1mr    D1mw I2mr    D2mr    D2mw  file:function
--------------------------------------------------------------------------------
4,007,016         4 1,000,004    2       3 125,001    2       3 125,001  /home/mlieber/loops/loops-fast.c:main [/work/home/mlieber/loops/loops
-fast]
4,007,003 1,000,001         0    0 125,001       0    0 125,001       .  /home/mlieber/loops/loops-fast.c:array_sum [/work/home/mlieber/loops/
loops-fast]
   30,650    11,448     3,249   14     438       5   14     389       4  ???:do_lookup_x [/lib64/ld-2.3.3.so]
   20,318     5,109     2,436   33     666     264   33     628     254  ???:_dl_relocate_object [/lib64/ld-2.3.3.so]

mlieber@phobos:~/loops>
```

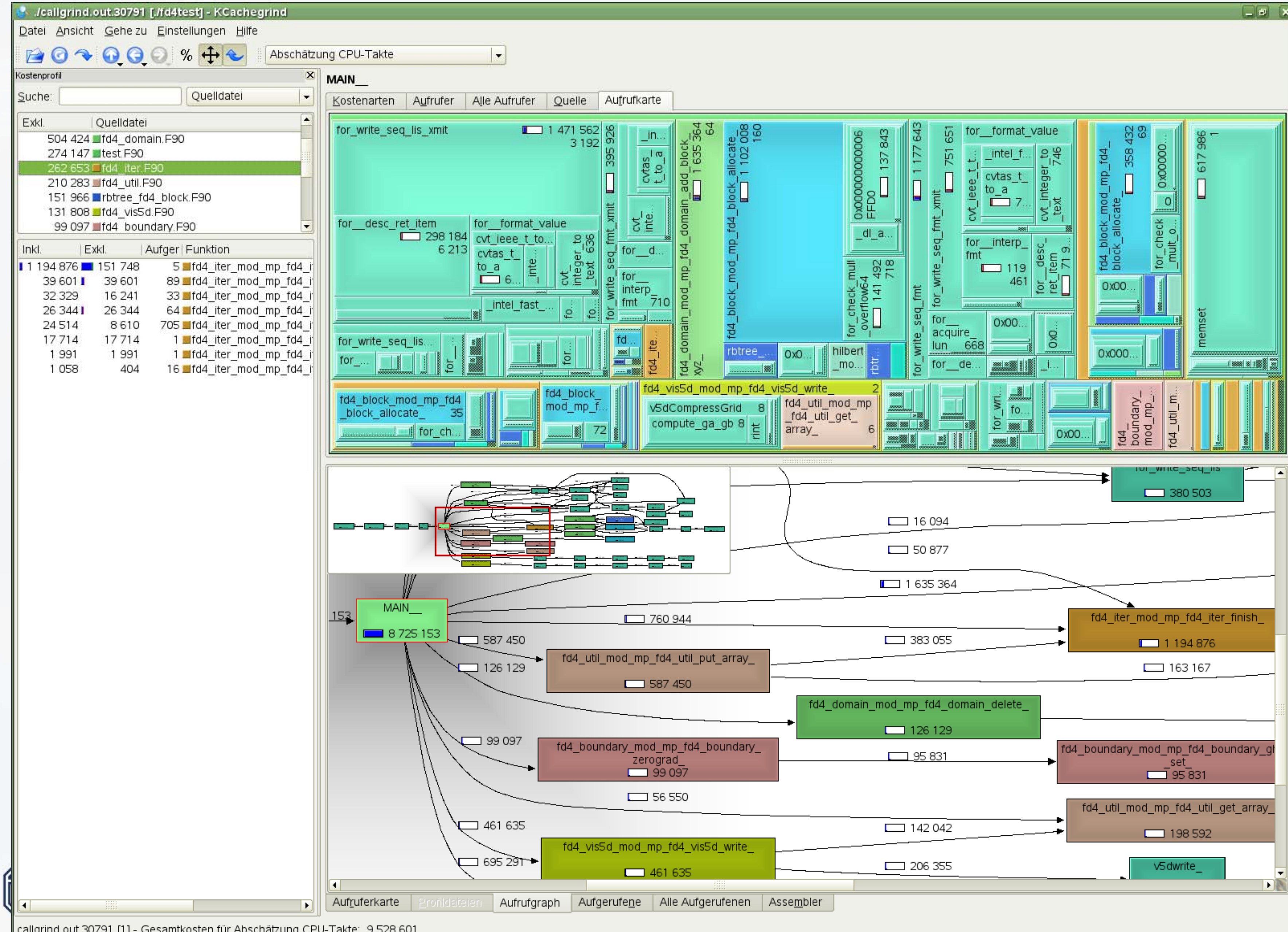
Callgrind Example: callgrind_annotate with Source

```
mlieber@phobos:~/loops
       Ir        Dr        Dw I1mr    D1mr    D1mw I2mr    D2mr    D2mw

-- line 3 ----------------------------------------
        .         .         .    .       .       .    .       .       .  #define N 1000
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .  double array_sum(double[N][N]);
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .  double array_sum(double a[N][N])
        .         .         .    .       .       .    .       .       .  {
        .         .         .    .       .       .    .       .       .    int i,j;
        .         .         .    .       .       .    .       .       .    double s;
        1         .         .    .       .       .    .       .       .    s=0;
    3,001         .         .    .       .       .    .       .       .    for(i=0;i<N;i++)
3,004,000         .         .    .       .       .    .       .       .      for(j=0;j<N;j++)
1,000,000 1,000,000         0    0 125,000       0    0 125,000       .        s += a[i][j];
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .    return s;
        1         1         0    0       1       0    0       1       .  }
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .  int main(int argc, char** argv)
        1         0         0    1       0       0    1       .       .  {
        .         .         .    .       .       .    .       .       .    double a[N][N];
        .         .         .    .       .       .    .       .       .    int i,j;
        .         .         .    .       .       .    .       .       .
    3,002         0         0    1       0       0    1       .       .    for(i=0;i<N;i++)
        .         .         .    .       .       .    .       .       .    {
3,004,000         .         .    .       .       .    .       .       .      for(j=0;j<N;j++)
        .         .         .    .       .       .    .       .       .      {
1,000,000         0 1,000,000    0       0 125,000    0       0 125,000        a[i][j]=0.01;
        .         .         .    .       .       .    .       .       .      }
        .         .         .    .       .       .    .       .       .    }
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .    /* this is just to prevent the compiler
        .         .         .    .       .       .    .       .       .       from optimizing the upper loop away */
       10         3         4    0       2       1    0       2       1    printf("sum = %10.3f\n",array_sum(a));
    4,771     1,181       688  198      42      21  198      42      21  => ???:printf (1x)
4,007,003 1,000,001         0    0 125,001       0    0 125,001       .  => /home/mlieber/loops/loops-fast.c:array_sum (1x)
    1,048       272       101    0      59      10    0      59      10  => ???:_dl_runtime_resolve (1x)
        .         .         .    .       .       .    .       .       .
        .         .         .    .       .       .    .       .       .    return 0;
        3         1         0    0       1       0    0       1       .  }

--------------------------------------------------------------------------------
Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
--------------------------------------------------------------------------------
98 97 99    0   99  100    0   99  100  percentage of events annotated

mlieber@phobos:~/loops> callgrind_annotate --auto=yes callgrind.out.7827
```

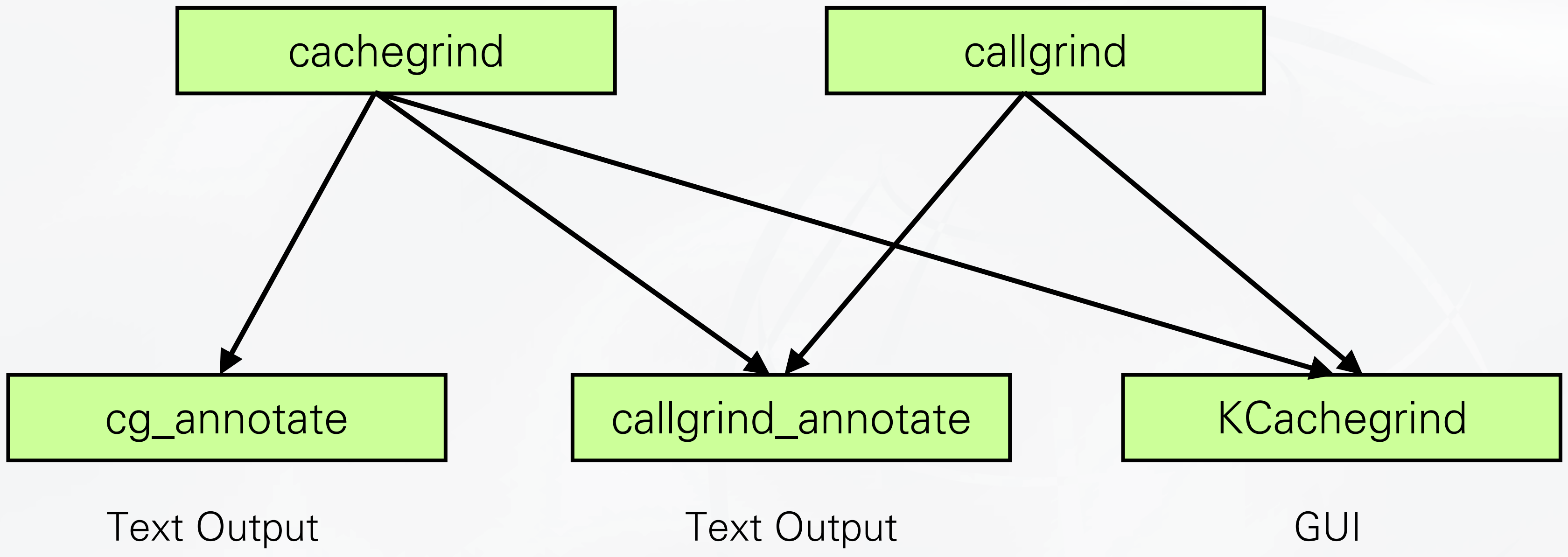
KCachegrind



KCachegrind



Tool Chains


cachegrind
callgrind
cg_annotate
callgrind_annotate
KCachegrind
Text Output
Text Output
GUI

Exercises

- `cd ~/callgrind`
- `make`

Exercise 1 - loops-fast

```c
#define N 1000

int main(int argc, char** argv)
{
  double a[N][N];
  int i,j;

  for(i=0;i<N;i++)
  {
    for(j=0;j<N;j++)
    {
      a[i][j]=0.01;
    }
  }

  printf("sum = %10.3f\n",array_sum(a));

  return 0;
}

double array_sum(double a[N][N])
{
  int i,j;
  double s;
  s=0;
  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      s += a[i][j];

  return s;
}
```

Fill 2D array

- Array size is 1000 x 1000 x 8 Byte = 8MB
- Phobos Cache:
  - 64kB L1i + 64kB L1d
  - 1MB L2
- Cache too small for array

Read the array

- If L2 cache was large enough (>8MB), no L2 miss would happen here!
- Because the upper loop loads the array into the cache

Exercise 1 - loops-fast

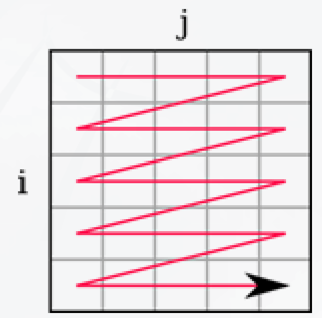
- Run the program with default cache settings:
  - `valgrind --tool=callgrind --simulate-cache=yes ./loops-fast`
  - 125.000 L1 and L2 misses in the write loop
  - 125.000 L1 and L2 misses in the read loop
- Run the program with custom cache settings, e.g. 16MB L2 cache:
  - `valgrind --tool=callgrind --simulate-cache=yes --L2=16777216,2,64 ./loops-fast`
  - Array fits in the L2 cache
  - No L2 misses in the read loop anymore
- View results with callgrind_annotate:
  - `callgrind_annotate --auto=yes ./callgrind.out.XXXX`
- View the reuslts with KCachegrind:
  - `module load kcachegrind`
  - `kcachegrind ./callgrind.out.XXXX`

Exercise 2 - loops-slow

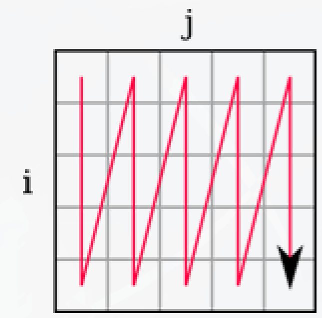
```c
#define N 1000

int main(int argc, char** argv)
{
  double a[N][N];
  int i,j;

  for(j=0;j<N;j++)
  {
    for(i=0;i<N;i++)
    {
      a[i][j]=0.01;
    }
  }

  printf("sum = %10.3f\n",array_sum(a));

  return 0;
}

double array_sum(double a[N][N])
{
  int i,j;
  double s;
  s=0;
  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      s += a[i][j];

  return s;
}
```

Fill 2D array, loop has “bad” order

C Storage:
row-major order



Access here:
column-major order



- Array size is 1000 x 1000 x 8 Byte = 8MB
- Phobos Cache:
  - 64kB L1i + 64kB L1d
  - 1MB L2
- Cache too small for array

Exercise 2 - loops-slow

```c
#define N 1000

int main(int argc, char** argv)
{
  double a[N][N];
  int i,j;

  for(j=0;j<N;j++)
  {
    for(i=0;i<N;i++)
    {
      a[i][j]=0.01;
    }
  }

  printf("sum = %10.3f\n",array_sum(a));

  return 0;
}

double array_sum(double a[N][N])
{
  int i,j;
  double s;
  s=0;
  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      s += a[i][j];

  return s;
}
```

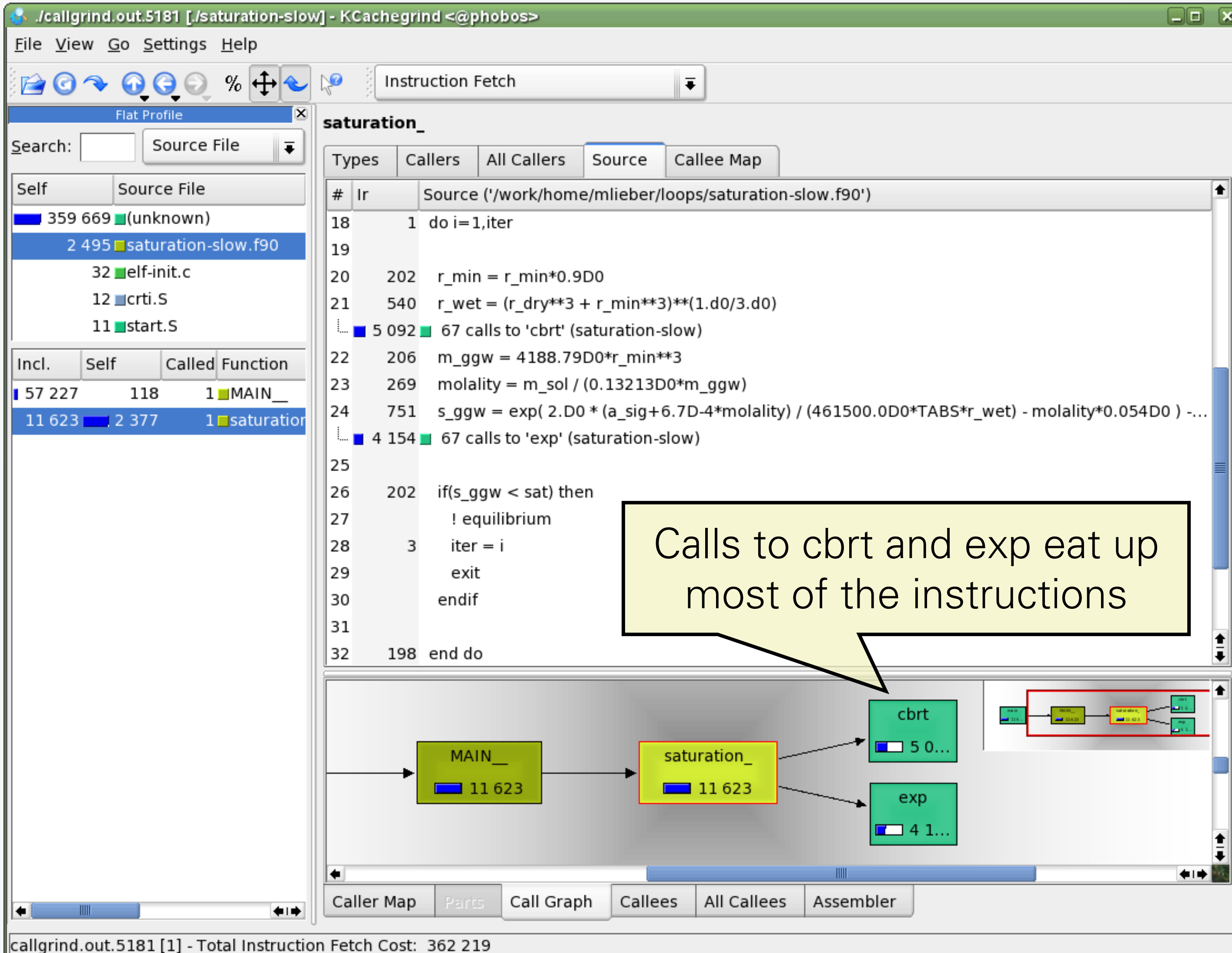
Fill 2D array, loop has “bad” order

- Cache misses will occur even if cache was large enough
  - Because array is not in cache
- But how many cache misses will occur?
  - Cache line size: 64 Byte = 8 double
  - 1.000.000 misses?
  - 125.000 misses?
- Try small L1 and L2 cache size (32768 Byte each) vs. default cache size setup
- Then try the example with “good” ordered loop with the small cache sizes and compare to “bad” ordered loop

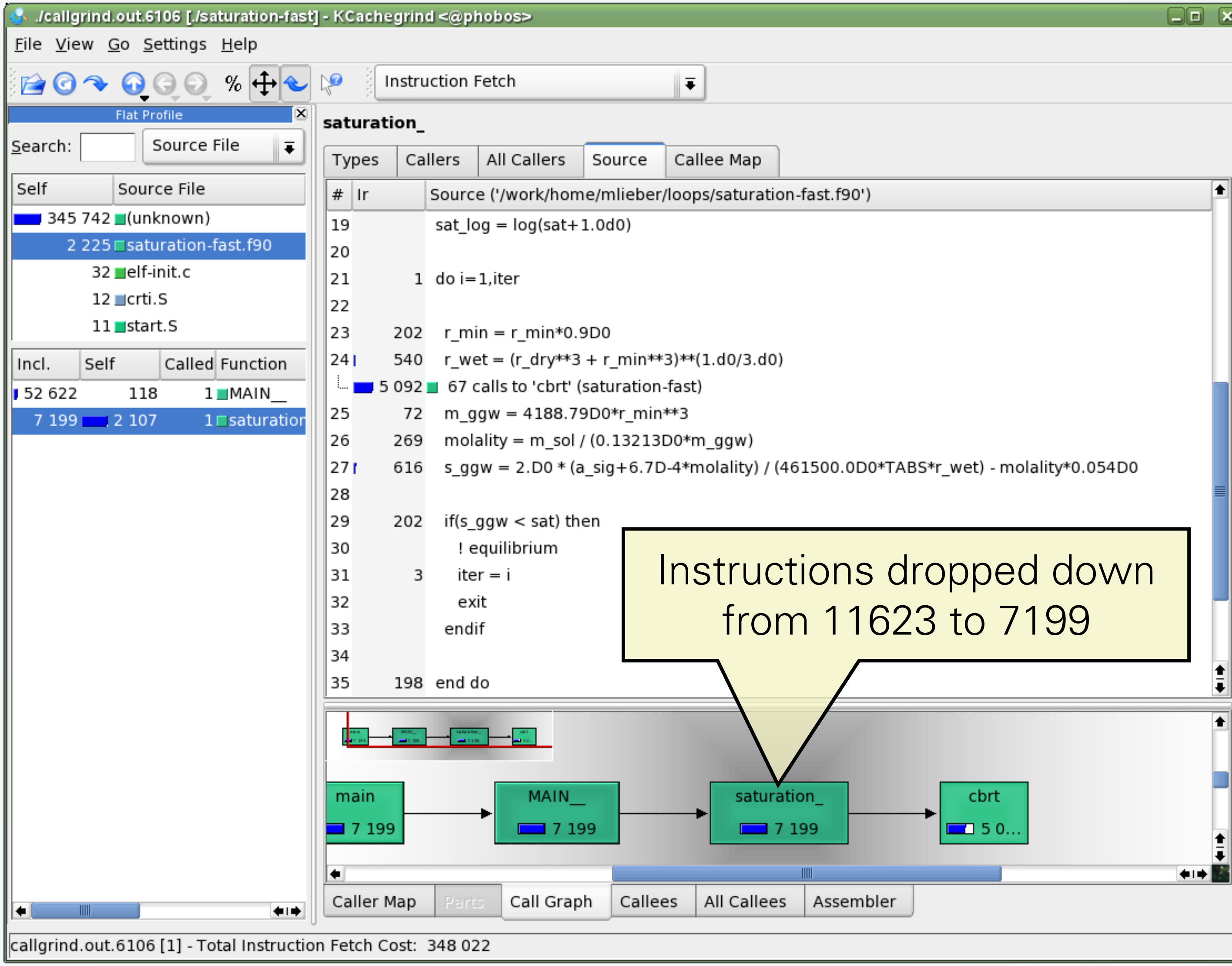
Beyond Cache Misses

- Callgrind can also be used to find performance problems which are not related to CPU cache
  - What code lines eat up most instructions (CPU cycles, time)
  - What system/math/library functions are called and what do they cost?
- Recorded instructions can be a measure of computational costs in cache-friendly code
- KCachegrind's cycle estimation allows incorporation of cache misses in this measure

Math function in KCachegrind: cbrt and exp are costly!



Math function in KCachegrind: optimized version – no exp!



Exercise 3 - saturation

- Compare the slow and the fast version of the saturation example
  - Don’t need to collect cache counters
  - **`valgrind --tool=callgrind ./saturation-fast`**
  - **`valgrind --tool=callgrind ./saturation-slow`**
- Display in KCachegrind

Selected Callgrind Command-Line Options

- **`--simulate-cache=[yes|no]`** – enable cache simulation
- **`--dump-instr=[yes|no]`** – collect information at per-instruction granularity, only useful for assembler view in KCachegrind
- **`--callgrind-out-file=<file>`** - output file
- **`--I1=<size>,<associativity>,<line size>`** - specifiy L1 instruction cache
- **`--D1=<size>,<associativity>,<line size>`** - specifiy L1 data cache
- **`--L2=<size>,<associativity>,<line size>`** - specifiy L2 cache

- More features and options:
  - User Manual: http://valgrind.org/docs/manual/cg-manual.html
  - **`valgrind --tool=callgrind --help`**

Summary

- Remember: Valgrind is based on simulation, no measurements!
  - Don’t trust the results to be absolutely accurate
  - Large Overhead
- Whenever using Cachegrind / Callgrind:
  - Reduce problem size, but should still be representative
  - Large application: extract computational kernel routines
- Easy to use
- But not available for IA64 (Altix)
- Callgrind_annotate is a good alternative when KCachegrind is not available (KCachegrind requires X11 and KDE libs)