

Performance Analysis

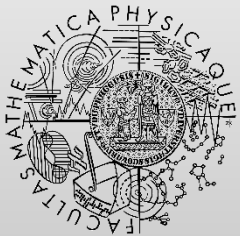
<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek

parizek@d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Performance analysis

- Find where the program spends most time
 - Identify code that you should optimize for speed
- Call graph: function names and spent time
- Important performance characteristics
 - throughput, latency, maximal load, average request processing time, ...
- Main approaches
 - **profiling**, benchmarking, load testing

Profiling

- Tools measuring frequency and duration of procedure calls during program execution
 - **GProf**, OProfile, Valgrind
- Basic principles (how it works)
 - Sampling: results not precise for short time periods
 - Recording program counter (PC) at regular intervals
 - Program instrumented with profiling-related code
 - Getting information from HW performance counters

- GNU Profiler
 - Distributed as a part of `binutils`

- Documentation
 - <https://sourceware.org/binutils/docs/gprof/>

How to use GProf

1) Build program with enabled profiling

```
gcc -g -pg -o program program.c
```

- Instrumentation: code that collects raw timing data added to the entry and exit points of each function

2) Execute the program normally

- Raw profile data written to the file `gmon.out`

3) Generate statistics (tables with results)

```
gprof <options> program [gmon.out]
```

- Output: flat profile, call graph

Flat profile

- How to get it

- `gprof -p program [gmon.out]`

- Excluding specific function

- `gprof -p -P<function_name> program`

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
35.29	0.06	0.06	3	20.00	43.33	compute
32.35	0.12	0.06	14000896	0.00	0.00	S_n
17.65	0.14	0.03	3	10.00	53.33	get_msg
5.88	0.15	0.01	5000320	0.00	0.00	F
5.88	0.17	0.01				main

Demo 1

- Basic features of GProf
 - Generating the flat profile
 - Excluding some functions
- Subject program
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/sha.tgz>
- Program has to run for a long time (at least few seconds) to get useful results
 - Measurement results are invalid otherwise

Flat profile: source code lines

- How to get it
 - `gprof -p -l program`

Each sample counts as 0.01 seconds.

```
% cumulative self
time  seconds  seconds  name
17.65   0.03   0.03   S_n (sha.c:18 @ 80485f9)
11.76   0.05   0.02   .... S_n (sha.c:17 @ 80485f0)
11.76   0.07   0.02   compute (sha.c:152 @ 804895b)
11.76   0.09   0.02   get_msg (sha.c:192 @ 8048baa)
```


Call graph

- How to get it
 - `gprof -q program`

```
index % time    self    children    called    name
-----
          0.03    0.13      3/3      main [1]
[2]    94.1    0.03    0.13      3    get_message_digest [2]
          0.06    0.07      3/3      compute_digest [3]
          0.00    0.00      3/3      get_padded_length [10]
          0.00    0.00      3/3      padd_message [11]
-----
          0.06    0.07      3/3      get_message_digest [2]
[3]    76.5    0.06    0.07      3    compute_digest [3]
          0.06    0.00 14000896/14000896  S_n [4]
          0.01    0.01 5000320/5000320   F [5]
-----
```

Demo 2

- GProf: other features
 - Flat profile for source code lines
 - Call graph (reading the output)

Performance analysis

- It is hard and tricky
 - Profiling results not 100% precise
 - Statistical approximation is used
 - Many things influence performance
 - Resource sharing (caches), garbage collection
 - Even harder for programs in JVM / .NET CLR
- Recommended practice
 - Use profilers only to identify parts of your program that are much slower than others

- GUI profiler for Java (heap, CPU)
- Documentation
 - <https://visualvm.github.io/>
- Important features
 - Heap dump
 - CPU sampling

Visual Studio profiling

- Available tools
 - CPU usage, memory usage, and many others
- Web (documentation, tutorials)
 - <https://learn.microsoft.com/en-us/visualstudio/profiling/?view=vs-2022>

Other profiling tools

- YourKit
 - Powerful profiler for Java and .NET
 - <http://yourkit.com/home/index.jsp>
 - Many advanced features (see web)
 - Handles also very large applications
- dotTrace
 - Target platform: C#, .NET applications
 - <https://www.jetbrains.com/profiler/>
- Valgrind
 - Supported tools: Cachegrind, Callgrind, Massif, DHAT, ...
 - Running: `--tool=<cachegrind | callgrind | massif>`
 - Inspecting results: `cg_annotate`, `callgrind_annotate`, `ms_print`
 - Demo 3: using tools on some program

Performance measurements – literature

- John Ousterhout. Always Measure One Level Deeper. Communications of the ACM, July 2018, Vol. 61 No. 7
 - <https://cacm.acm.org/magazines/2018/7/229031-always-measure-one-level-deeper/fulltext>
 - Describes common mistakes and suggestions how engineers should do measurements of software performance (methodology, infrastructure, hints)

Load testing

- Generating specific (heavy) load for server applications (WWW, email, database)
 - Target URL and payload
 - Number of threads (clients)
 - Frequency of requests
- JMeter (<http://jmeter.apache.org/>)
 - supports: GUI, command-line, distributed mode
- Netling (<https://github.com/hallatore/Netling>)

Coverage

- Metrics

- Statement coverage
- Branch coverage
- Control-flow paths

- Tools

- GCov (<https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>)
- Jcov (<https://wiki.openjdk.java.net/display/CodeTools/jcov>)
- JaCoCo (<http://www.jacoco.org/jacoco/>)

Measuring coverage with GCov

- Build program with special options
 - `gcc -fprofile-arcs -ftest-coverage -o program program.c`
- Execute the program normally
- Run the gcov tool on source code files
 - `gcov program.c`
- Open the file `program.c.gcov`

- With branch and block statistics
 - `gcov -b program.c`

Related courses

- NSWI131: Vyhodnocování výkonnosti počítačových systémů
 - Topics: benchmarking, experimental evaluation, statistical analysis, modeling, simulation
- NSWI126: Pokročilé nástroje pro vývoj a monitorování software
 - Topics: other profilers and performance analyzers
 - ZS 2024/2025

Homework

- Assignment
 - <http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly/>
- Deadline
 - 26.5.2024
- Homework targets GProf and programs in C
 - Alternative: you can use another profiler (e.g., for Java or C#) on the program of your choice
 - Report answers to performance-related questions very similar to those specified in tasks 5a-d, 5f and 7a-g
 - For questions inspired by tasks 7a-g, pick some functions in the program code instead of 'zip' and 'updcrc'