LINUX PLUMBERS CONFERENCE September 20-24, 2021

gprofng - The Next Generation GNU Profiler

Ruud van der Pas, Vladimir Mezentsev
Compilers and Toolchain Team
Oracle Linux Engineering Organization

Outline of this Talk

This talk is about gprofng, the next generation GNU profiling tool

- After a brief introduction, we show several examples
- With these examples, we focus on the results
  - Due to time constraints we cannot explain much how we did this
  - The binutils repo contains a mini tutorial in Texinfo format
- We conclude with a brief overview of our short term direction

Submission to Binutils

August 11, 2021 - Submitted to binutils@sourceware.org for Review

[PATCH] gprofng: a new GNU profiler

Vladimir Mezentsev vladimir.mezentsev@oracle.com
Wed Aug 11 21:10:35 GMT 2021

[PATCH] gprofng: a new GNU profiler

Vladimir Mezentsev vladimir.mezentsev@oracle.com
Wed Aug 11 21:10:35 GMT 2021

- Previous message (by thread): [RISCV] RISC-V GNU Toolchain Biweekly Sync-up call (Aug 12, 2021)
- Next message (by thread): [PATCH] gprofng: a new GNU profiler
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hi people!

In this submission we are contributing a new profiler to the GNU binary
utilities, called gprofng (for GNU profiler, next generation).

Why a new profiler?
===================

The GNU profiler, gprof, works well enough in many cases. However, it
hasn't aged well and it is not that very well suited for profiling
modern-world applications. Examples of its limitations are lack of support
for profiling multithreaded programs, and shared objects. Both are
ubiquitous nowadays.

Main characteristics of gprofng
===============================

gprofng supports profiling C, C++ and Java programs. Unlike the old
gprof, it doesn't require to build annotated versions of the programs.
Profiling "production" binaries should work just fine.

Another distinguishing feature of gprofng is the support for various filters
that allow the user to easily drill deeper into an area of interest.

The profiler is commanded through a driver program called `gprofng'.
This driver supports the following sub-commands:

gpronfg collect app EXECUTABLE

This runs EXECUTABLE and collects application performance data.

gprofng display text EXPERIMENT

This runs a client command-line interface that provides access to the
collected performance data stored in the experiment directory.

gprofng display html EXPERIMENT

This generates an HTML report from the collected performance data.
stored in the experiment directory.

gprofng display src OBJECT-FILE
```

A Very Brief History of gprofng/1

The Oracle Developer Studio Performance Analyzer

- Was developed for 20+ years
- Many internal and external users with real-world applications
- Focus on the SPARC processor, Studio compilers, and Solaris operating system

This profiling tool served as a basis for gprofng

A Very Brief History of gprofng/2

The current gprofng project:

- Created a standalone version on Linux
- Adapted the source code to the GNU Coding Standards
- Adapted the build process to be compliant with other binutils components
- Added the port to Arm (aarch64)
- Fixed several bugs
- Completely redesigned the User Interface (UI)

About gprofng

Collects and displays application level performance data

- Languages supported: C, C++, Java, and Scala
- Full support for gcc compilers
- Currently supports various processors from Intel, AMD, and Arm
- No need to recompile the code
  - Works with production binaries
- Supports Multithreading
  - Posix Threads, OpenMP, and Java Threads

How Does gprofng Work?

Two step approach

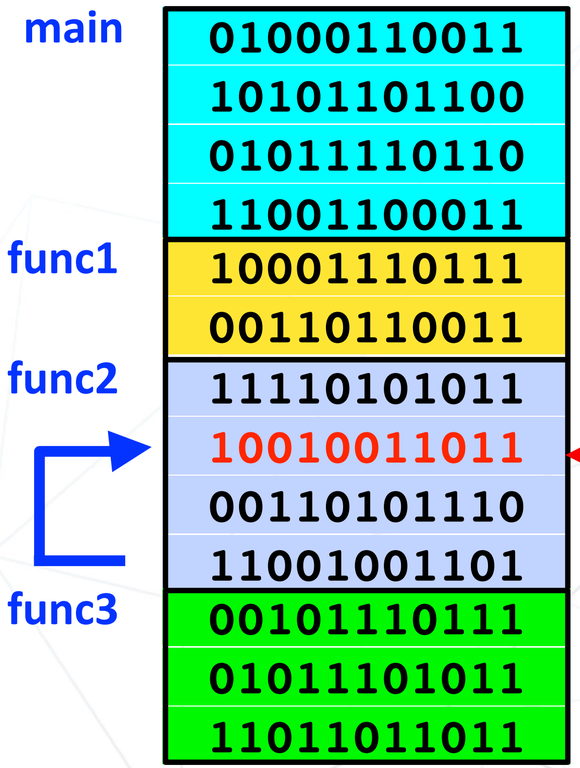
- First, collect the performance data on the target executable
- Next, display the data
  - Information is available at the function, source, and disassembly level
  - Multiple views into the data
  - Already a single run can provide a lot of insight
- **Scripting** support to generate and customize profiles in an automated way
- **Filters** help to zoom in on the data
- **Comparison** of profiles is supported

A Brief Comparison with gprof

gprof	gprofng
Uses tracing/instrumentation	Uses sampling
Requires a recompilation	Can use existing/production executables
Lacks support for modern features	Support for shared libraries and multithreading
Limited customization	Scripting commands supported
No support for filters	Various filters supported
Cannot compare profiles	Comparison of profiles is supported
No support for event counters	Event counter support*

**) Fully functional, but limited support for very recent processors (work in progress)*

Statistical Call Stack Sampling



1. *The program is stopped at regular intervals*

2. *The Program Counter (PC) and other information is recorded*

3. *An overview with the execution times is produced*

Execution time	Name
10	func2
5	func1
2	func3
1	main

The gprofng Command Structure

General syntax:

```
$ gprofng <functionality> [<qualifier>] [<options>]
```

Examples:

```
$ gprofng collect app
$ gprofng display text
$ gprofng archive
```

An Overview of the Commands

Command	Functionality
`$ gprofng collect app`	Collect the performance data
`$ gprofng display text`	Display the performance data in ASCII format
`$ gprofng archive`	Archive an experiment directory
`$ gprofng display src`	Display the source and disassembly of an object file
`$ gprofng display html*`	Generate an HTML structure to analyze the data in a browser

**) Currently limited functionality, but we intend a future patch to make a fully functional version available*

High Time for Examples!

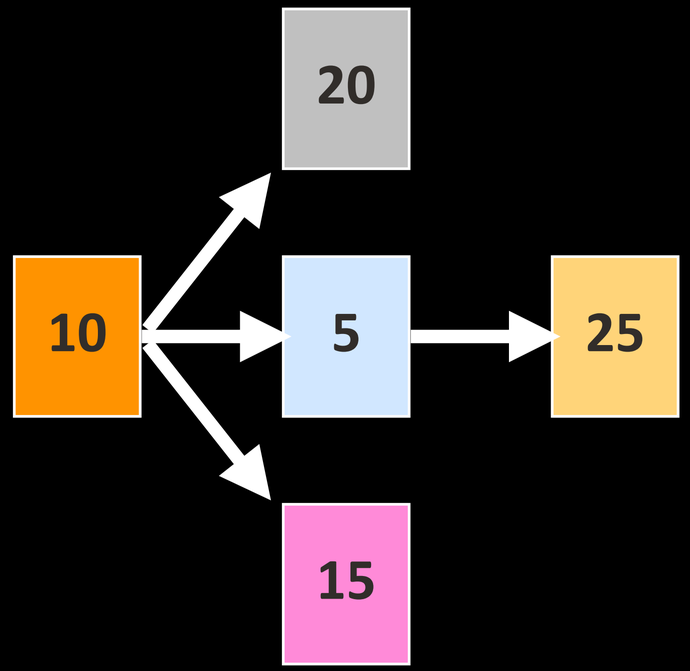
- How to get a basic profile
- Display the function, source, and disassembly information
- Customize the data collection and displays
- Scripting
- Support for multithreading
- Comparison of profiles

Inclusive and Exclusive Metrics

This is an important concept in gprofng

- The inclusive metric includes all callees underneath the caller
  - For example, all the CPU time accumulated when executing a function
- The exclusive metric excludes everything outside the caller
  - For example, the CPU time accumulated outside of calling other functions

An Example of Inclusive/Exclusive



Function	Inclusive time	Exclusive time
	75	10
	20	20
	15	15
	30	5
	25	25

Generate the Experiment Data

If this is how you normally run your program:

```
$ ./mxv-pthreads.exe -m 3000 -n 2000 -t 1
mxv: error check passed - rows = 3000 columns = 2000 threads = 1
$
```

The only difference is to run it under control of "gprofng collect app" now:

```
$ gprofng collect app ./mxv.pthreads.exe -m 3000 -n 2000 -t 1
Creating experiment database test.1.er (Process ID: 2416504) ...
mxv: error check passed - rows = 3000 columns = 2000 threads = 1
$
```

The Function Overview

```
$ gprofng display text -functions test.1.er
```

```
Functions sorted by metric: Exclusive Total CPU Time

Excl.       Incl.       Name
Total       Total
CPU sec.    CPU sec.
2.272       2.272       <Total>
2.160       2.160       mxv_core
0.047       0.103       init_data
0.030       0.043       erand48_r
0.013       0.013       __drand48_iterate
0.013       0.056       drand48
   … etc …
```

The Source Line Overview

```
$ gprofng display text -source mxv_core test.1.er
```

```
   Excl.     Incl.
   Total     Total
   CPU sec.  CPU sec.
                        <lines deleted>
                                <Function: mxv_core>
   0.        0.                 32. void __attribute__ ((noinline)) mxv_core(
                        <lines deleted>
   0.        0.                 33. {
   0.        0.                 34.    for (uint64_t i=row_index_start;
                                           i<=row_index_end; i++) {
   0.        0.                 35.        double row_sum = 0.0;
## 1.687     1.687              36.        for (int64_t j=0; j<n; j++)
   0.473     0.473              37.            row_sum += A[i][j] * b[j];
   0.        0.                 38.        c[i] = row_sum;
                                39.    }
   0.        0.                 40. }
```

The Disassembly Overview

```
$ gprofng display text -disasm mxv_core test.1.er
```

```
    Excl.     Incl.
   Total     Total
   CPU sec.  CPU sec.
                    <lines deleted>
                           35.         double row_sum = 0.0;
                           36.         for (int64_t j=0; j<n; j++)
                           37.             row_sum += A[i][j] * b[j];
                     <lines deleted>
   0.         0.                [35]   4021ce:  pxor    %xmm1,%xmm1
   0.002      0.002             [37]   4021d2:  movsd   (%rdx,%rax,8),%xmm0
   0.096      0.096             [37]   4021d7:  mulsd   (%r9,%rax,8),%xmm0
   0.375      0.375             [37]   4021dd:  addsd   %xmm0,%xmm1
## 1.683      1.683             [36]   4021e1:  add     $0x1,%rax
   0.004      0.004             [36]   4021e5:  cmp     %rax,%rcx
   0.         0.                [36]   4021e8:  jne     0xffffffffffffffea
              … etc …
```

Customize the Display

```
$ gprofng display text -limit 5 –metrics name:e.%totalcpu \
  -functions test.1.er
```

```
Current metrics: name:e.%totalcpu
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
Functions sorted by metric: Exclusive Total CPU Time

Name                  Excl. Total
                      CPU
                       sec.      %
 <Total>              2.272 100.00
 mxv_core             2.160  95.04
 init_data            0.047   2.06
 erand48_r            0.030   1.32
 __drand48_iterate    0.013   0.57
```

Scripting

```
$ gprofng display text -script my-script test.1.er
```

```
$ cat my-script

# This is my first gprofng script
# Set the metrics
metrics i.%totalcpu:e.%totalcpu:name
# Use the exclusive time to sort
sort e.totalcpu
# Limit the function list to 5 lines
limit 5
# Show the function list
functions
```

Scripting - The Output

```
Functions sorted by metric: Exclusive Total CPU Time

Incl. Total    Excl. Total     Name
CPU            CPU
 sec.      %    sec.      %
2.272 100.00   2.272 100.00    <Total>
2.160  95.04   2.160  95.04    mxv_core
0.103   4.52   0.047   2.06    init_data
0.043   1.89   0.030   1.32    erand48_r
0.013   0.57   0.013   0.57    __drand48_iterate
```

Customize the Data Collection

```
$ gprofng collect app -O mxv.thr.1.er ./mxv-pthreads.exe \
  -m 3000 -n 2000 -t 1
```

Additional options:

- Set the sampling granularity
- Add one or more labels to the experiment
- Archive sources, objects, etc in the experiment
- …

Support for Multithreading

```
$ gprofng collect app -O mxv.2.thr.er ./mxv-pthreads.exe \
  -m 3000 -n 2000 -t 2

$ gprofng display text -metrics e.%totalcpu -threads mxv.2.thr.er
```

```
Current metrics: e.%totalcpu:name
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
Objects sorted by metric: Exclusive Total CPU Time

Excl. Total     Name
CPU
 sec.       %
2.258 100.00    <Total>
1.075  47.59    Process 1, Thread 3
1.070  47.37    Process 1, Thread 2
0.114   5.03    Process 1, Thread 1
```

Multithreading and Filters

```
# Define the metrics
metrics e.%totalcpu
# Limit the output to 10 lines
limit 10
# Get the function overview for thread 1
thread_select 1
functions
# Get the function overview for thread 2
thread_select 2
functions
# Get the function overview for thread 3
thread_select 3
functions
```

The Profile for Thread 1

```
# Get the function overview for thread 1
Exp Sel Total
=== === =====
  1 1       3
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total    Name
CPU
 sec.       %
0.114 100.00   <Total>
0.051  44.74   init_data
0.028  24.56   erand48_r
0.017  14.91   __drand48_iterate
0.010   8.77   _int_malloc
0.008   7.02   drand48
… etc …
```

The Profiles for Threads 2-3

```
# Get the function overview for thread 2
Exp Sel Total
=== === =====
  1 2       3
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total     Name
CPU
 sec.       %
1.072 100.00   <Total>
1.072 100.00   mxv_core
0.      0.     collector_root
0.      0.     driver_mxv
```

```
# Get the function overview for thread 3
Exp Sel Total
=== === =====
  1 3       3
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total     Name
CPU
 sec.       %
1.076 100.00   <Total>
1.076 100.00   mxv_core
0.      0.     collector_root
0.      0.     driver_mxv
```

Generate the Two Experiments

```
$ gprofng collect app -O mxv.hwc.1.thr.er -h llm \
  ./mxv-pthreads.exe -m 3000 -n 2000 -t 1
Creating experiment database mxv.hwc.1.thr.er (Process ID: 23454) ...
mxv: error check passed - rows = 3000 columns = 2000 threads = 1

$ gprofng collect app -O mxv.hwc.2.thr.er -h llm \
./mxv-pthreads.exe -m 3000 -n 2000 -t 2
Creating experiment database mxv.hwc.2.thr.er (Process ID: 23462) ...
mxv: error check passed - rows = 3000 columns = 2000 threads = 2
```

Compare Absolute Numbers

```
$ gprofng display text -script comp1 mxv.hwc.*.thr.er
```

Name	mxv.hwc.comp.1.thr.er Excl. Last-Level Cache Misses	mxv.hwc.comp.2.thr.er Excl. Last-Level Cache Misses
<Total>	122709276	96696878
mxv_core	121796001	95793620
init_data	723064	763104
erand48_r	100111	50053
drand48	60065	70077

```
# Limit the output to 5 lines
limit 5
# Define the metrics
metrics name:e.llm
# Show absolute numbers
compare on
functions
```

Compare Ratios

```
$ gprofng display text -script comp2 mxv.hwc.*.thr.er
```

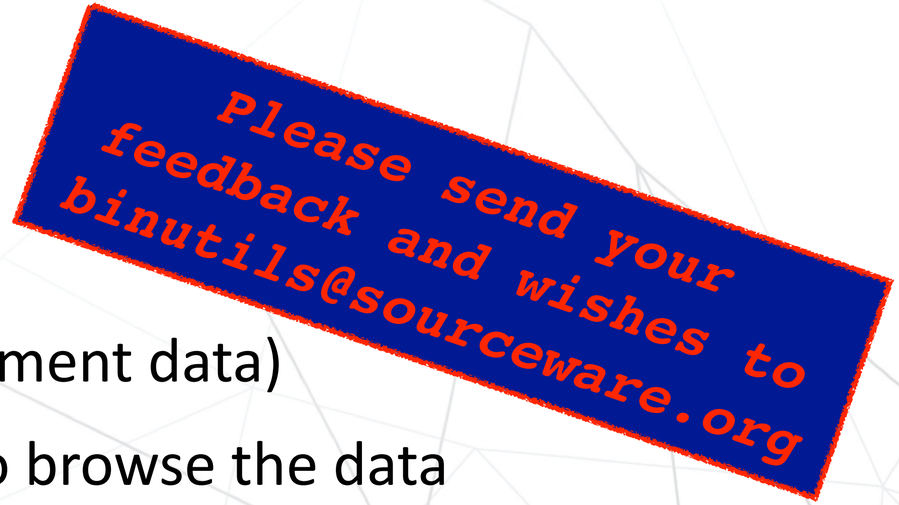
Name	mxv.hwc.comp.1.thr.er Excl. Last-Level Cache Misses		mxv.hwc.comp.2.thr.er Excl. Last-Level Cache Misses ratio
<Total>	122709276	x	0.788
mxv_core	121796001	x	0.787
init_data	723064	x	1.055
erand48_r	100111	x	0.500
drand48	60065	x	1.167

```
# Limit the output to 5 lines
limit 5
# Define the metrics
metrics name:e.llm
# Show the ratio current/ref
compare ratio
functions
```

More to Explore

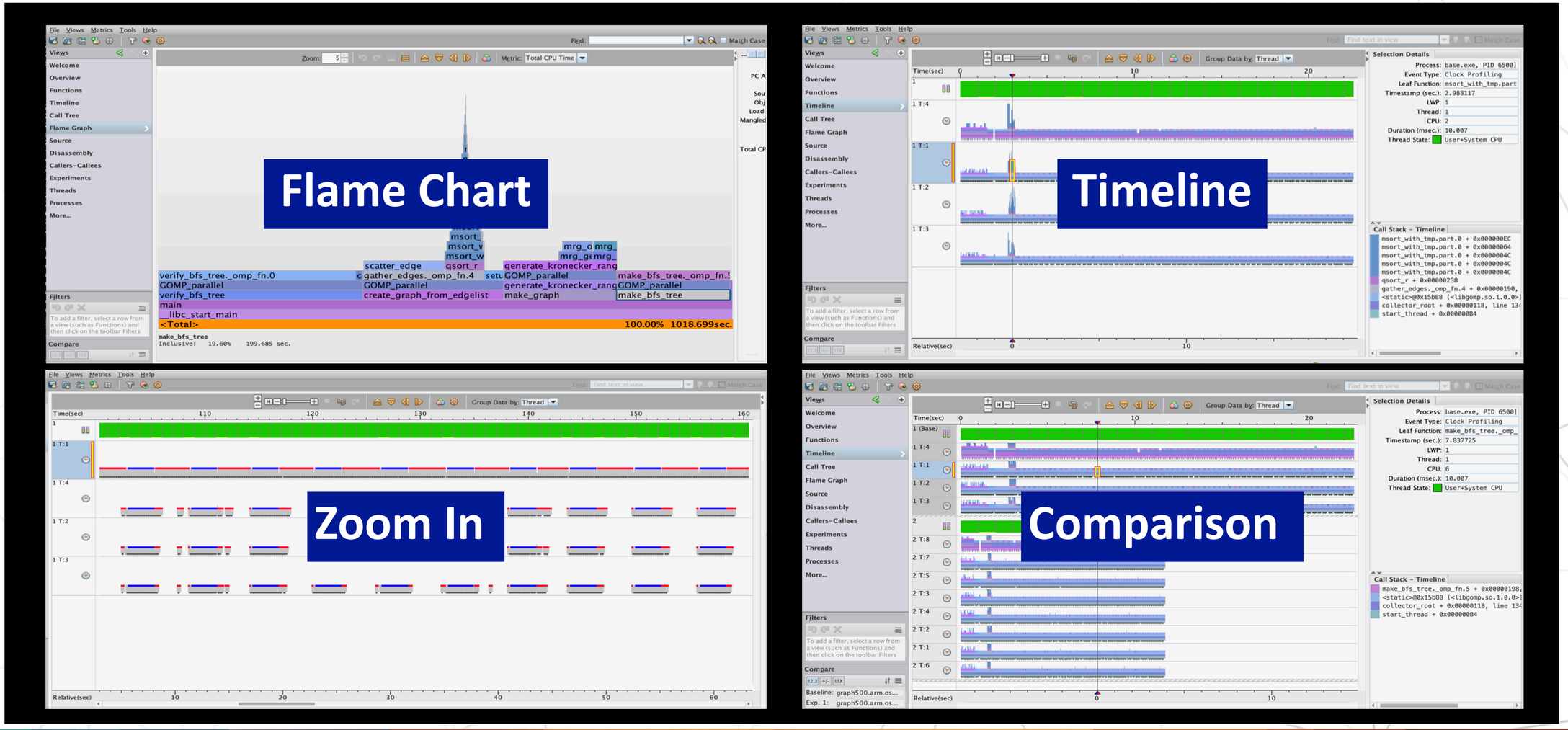
- The call tree
- Additional filters
- More information on the experiment(s)
- Additional customization
- Support for hardware event counters
- …

Future Directions



- **Fix bugs and help users to get started**
- **Top priorities for development**
  - Make a GUI available (to display and analyze the experiment data)
  - Finish the tool that generates an HTML based system to browse the data
- **Other topics under consideration**
  - Support for hardware event counters on more recent processors
  - Support additional metrics with call stack sampling
  - Attach to a running process
  - …

The gprofng GUI Sneak Preview



Thank you for attending!

Time for Q&A!