ORACLE

gprofng: The Next Generation GNU Profiling Tool

Ruud van der Pas, Vladimir Mezentsev, Kurt Goebel

Oracle Linux Toolchain Team
Oracle Linux Engineering and Virtualization

PPoPP Conference - BID Workshop
January 25, 2023

Outline

- History and Status
- A Brief Overview of gprofng
- A Demo How to Make Your First Profile
- A Sneak Preview of the GUI
- Future Directions
- Q and A

A Very Brief History of gprofng/1

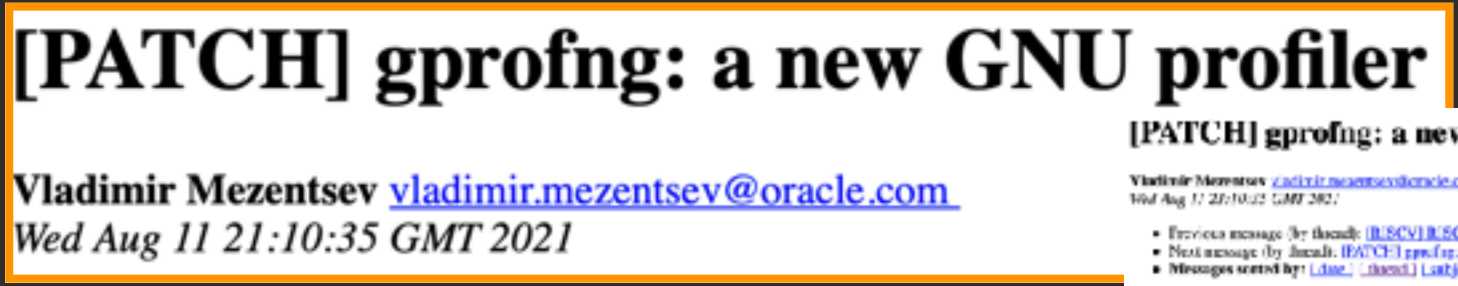
- The Oracle Developer Studio Performance Analyzer was developed for 20+ years
  - Many internal and external users with real-world applications
  - Focus on the SPARC processor, Studio compilers, and Solaris operating system
  - Support for x86 Linux for 10+ years
- This profiling tool served as a basis for gprofng

A Very Brief History of gprofng/2

The current gprofng project:

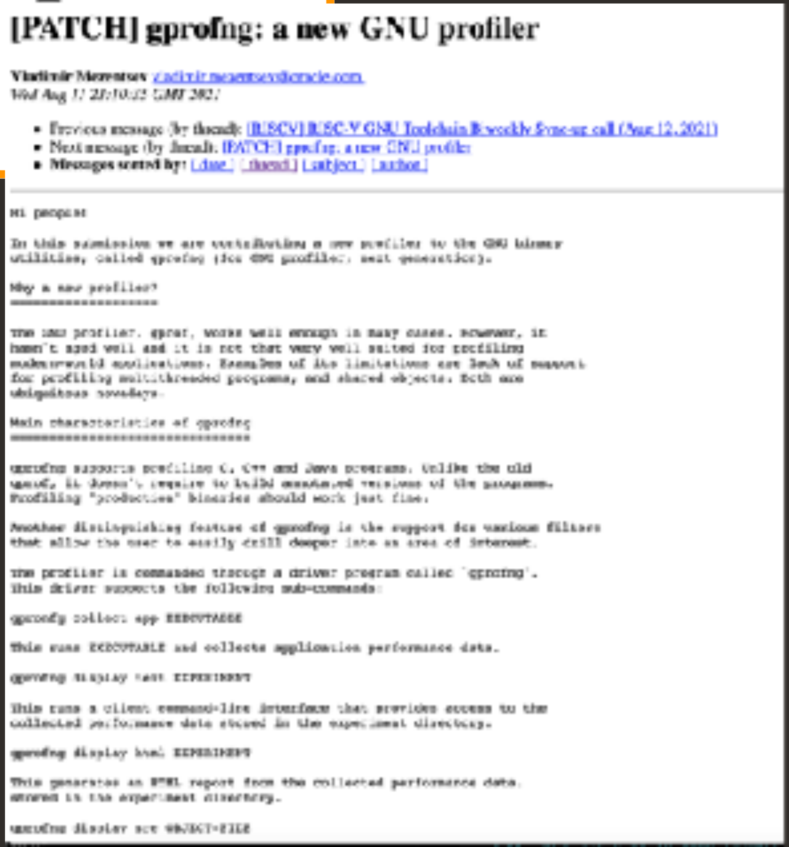
- Created a standalone version on Linux
- Adapted the source code to the GNU Coding Standards
- Adapted the build process to be compliant with other binutils components
- Added the port to Arm (aarch64)
- Fixed several bugs
- Completely redesigned the User Interface (UI)
- …

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



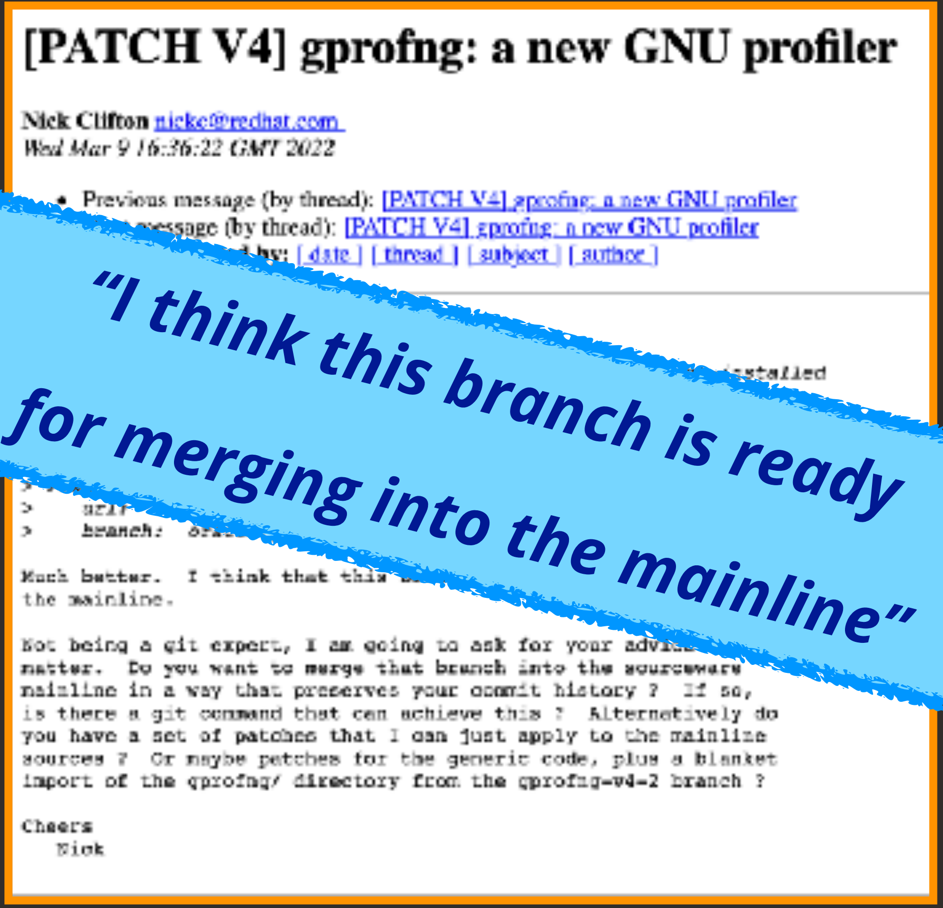
[PATCH] gprofng: a new GNU profiler

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 instrumented 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:

gprofng 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
```

March 9, 2022 - Approval to Merge into the Mainline!



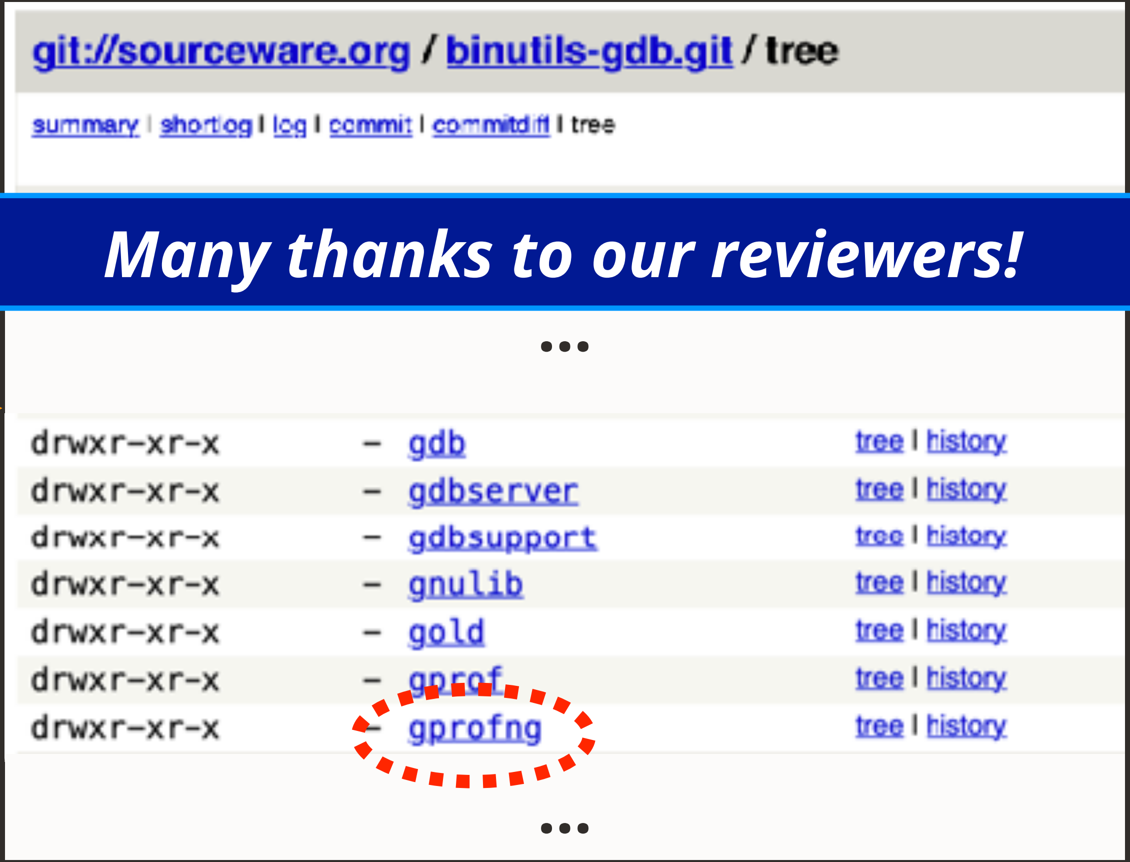
[PATCH V4] gprofng: a new GNU profiler

Nick Clifton nickc@redhat.com
Wed Mar 9 16:36:22 GMT 2022

- Previous message (by thread): [PATCH V4] gprofng: a new GNU profiler
- message (by thread): [PATCH V4] gprofng: a new GNU profiler
- by: [date] [thread] [subject] [author]

```
                                                  installed
>     
>     branch:
Much better.  I think that this
the mainline.

Not being a git expert, I am going to ask for your advice
matter.  Do you want to merge that branch into the sourceware
mainline in a way that preserves your commit history ?  If so,
is there a git command that can achieve this ?  Alternatively do
you have a set of patches that I can just apply to the mainline
sources ?  Or maybe patches for the generic code, plus a blanket
import of the gprofng/ directory from the gprofng-v4-2 branch ?

Cheers
  Nick
```

"I think this branch is ready for merging into the mainline"



git://sourceware.org / binutils-gdb.git / tree

summary | shortlog | log | commit | commitdiff | tree

Many thanks to our reviewers!

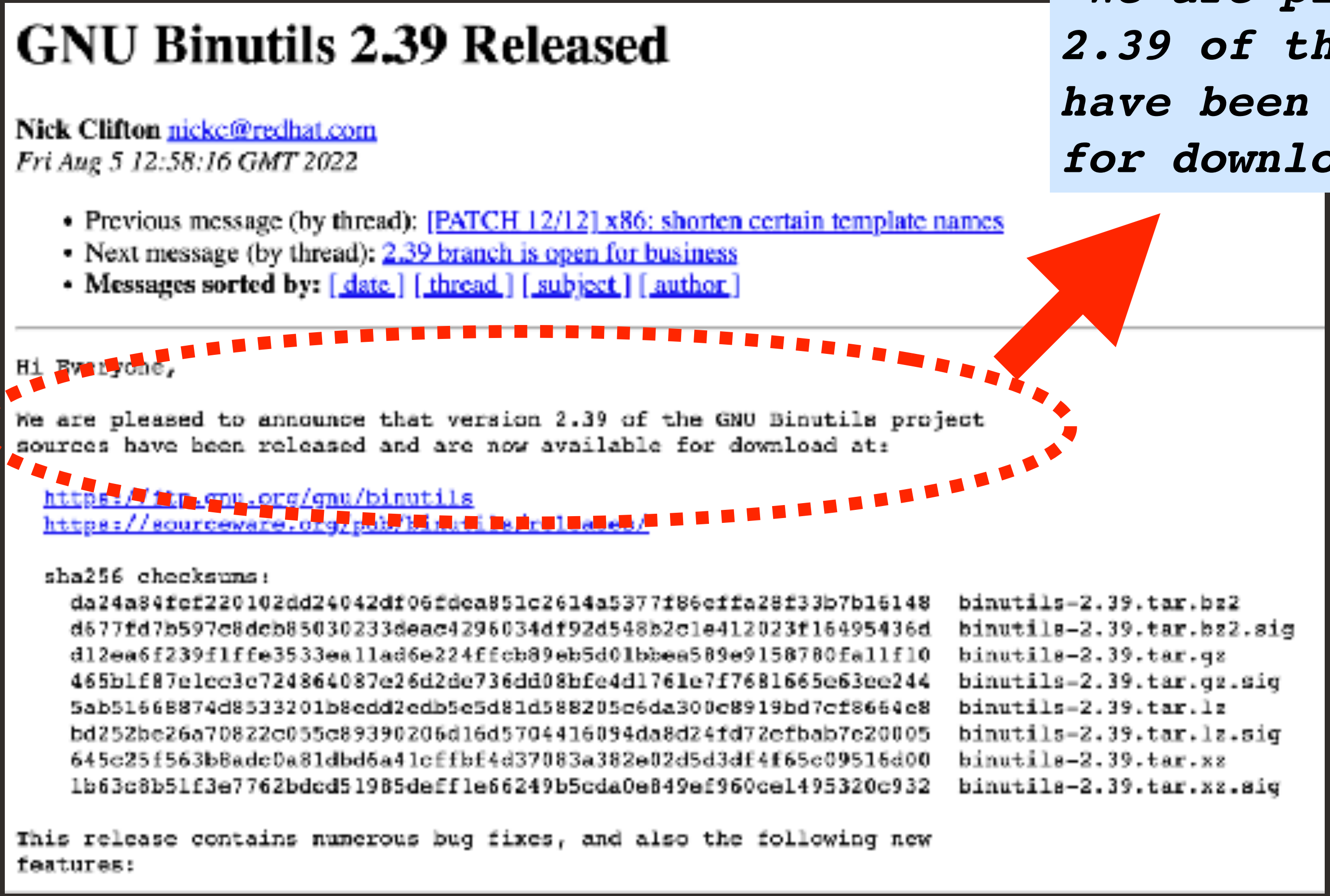
...

```
drwxr-xr-x   -  gdb          tree | history
drwxr-xr-x   -  gdbserver    tree | history
drwxr-xr-x   -  gdbsupport   tree | history
drwxr-xr-x   -  gnulib       tree | history
drwxr-xr-x   -  gold         tree | history
drwxr-xr-x   -  gprof        tree | history
drwxr-xr-x   -  gprofng      tree | history
```

...

August 5, 2022 - GNU binutils 2.39 has been released!

“We are pleased to announce that version 2.39 of the GNU Binutils project sources have been released and are now available for download at:”



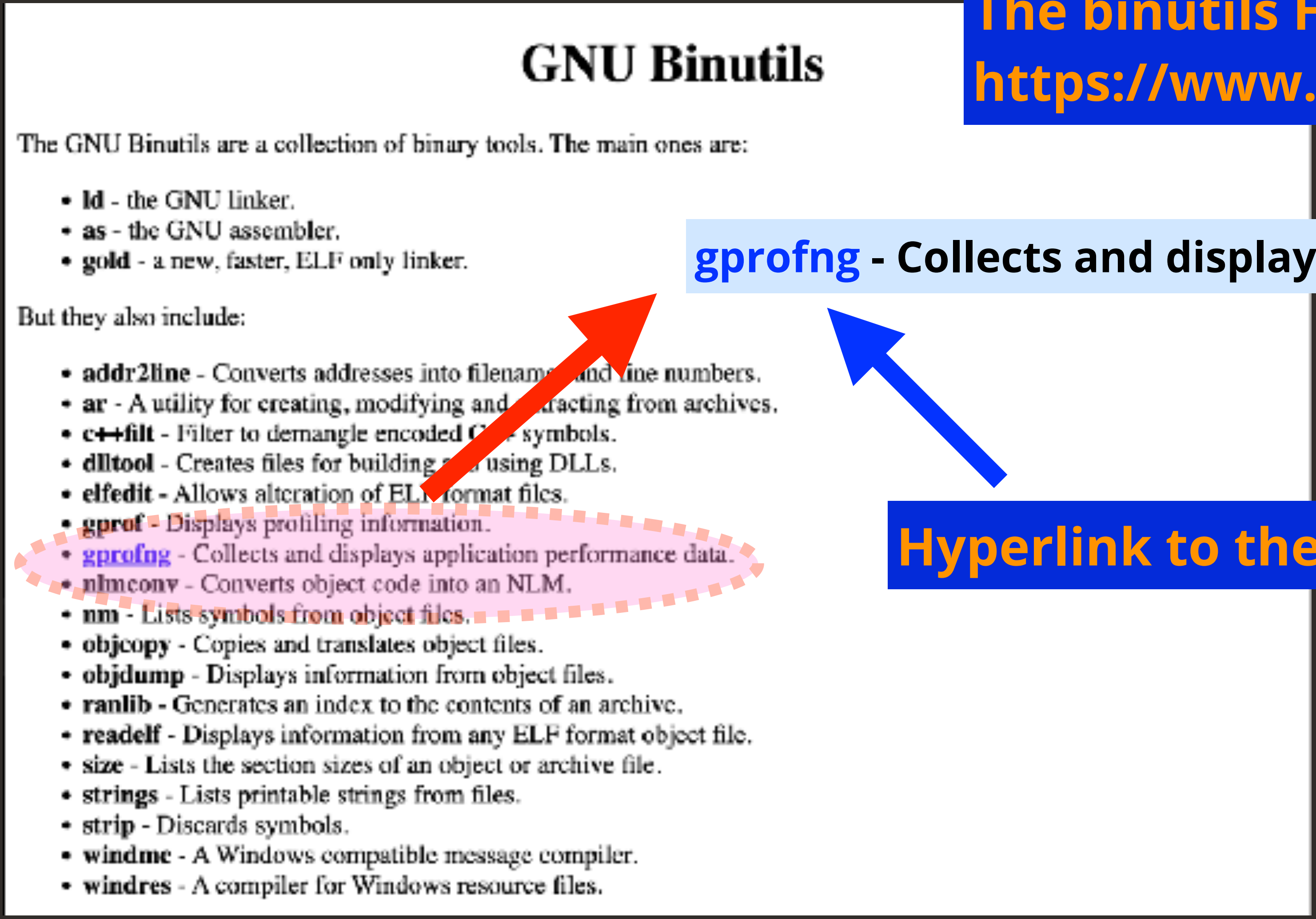
GNU Binutils 2.39 Released

Nick Clifton nickc@redhat.com
Fri Aug 5 12:58:16 GMT 2022

- Previous message (by thread): [PATCH 12/12] x86: shorten certain template names
- Next message (by thread): 2.39 branch is open for business
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hi Everyone,

We are pleased to announce that version 2.39 of the GNU Binutils project
sources have been released and are now available for download at:

   https://ftp.gnu.org/gnu/binutils
   https://sourceware.org/pub/binutils/releases/

   sha256 checksums:
     da24a84fef220102dd24042df06fdea851c2614a5377f86effa28f33b7b16148  binutils-2.39.tar.bz2
     d677fd7b597c8dcb85030233deac4296034df92d548b2c1e412023f16495436d  binutils-2.39.tar.bz2.sig
     d12ea6f239f1ffe3533ea11ad6e224ffcb89eb5d01bbea589e9158780fa11f10  binutils-2.39.tar.gz
     465b1f87e1ecc3c724864087e26d2de736dd08bfe4d1761e7f7681665e63ee244  binutils-2.39.tar.gz.sig
     5ab5166B874d8533201b8edd2edb5e5d81d588205c6da300c8919bd7cf8664c8  binutils-2.39.tar.lz
     bd252be26a70822c055c89390206d16d5704416094da8d24fd72efbab7c20005  binutils-2.39.tar.lz.sig
     645c25f563b8adc0a81dbd6a41cffbf4d37083a382e02d5d3df4f65c09516d00  binutils-2.39.tar.xz
     1b63c8b51f3e7762bdcd51985deff1e66249b5cda0e849ef960ce1495320c932  binutils-2.39.tar.xz.sig

This release contains numerous bug fixes, and also the following new
features:
```

And ... for the first time, gprofng was included!



GNU Binutils

The GNU Binutils are a collection of binary tools. The main ones are:

- **ld** - the GNU linker.
- **as** - the GNU assembler.
- **gold** - a new, faster, ELF only linker.

But they also include:

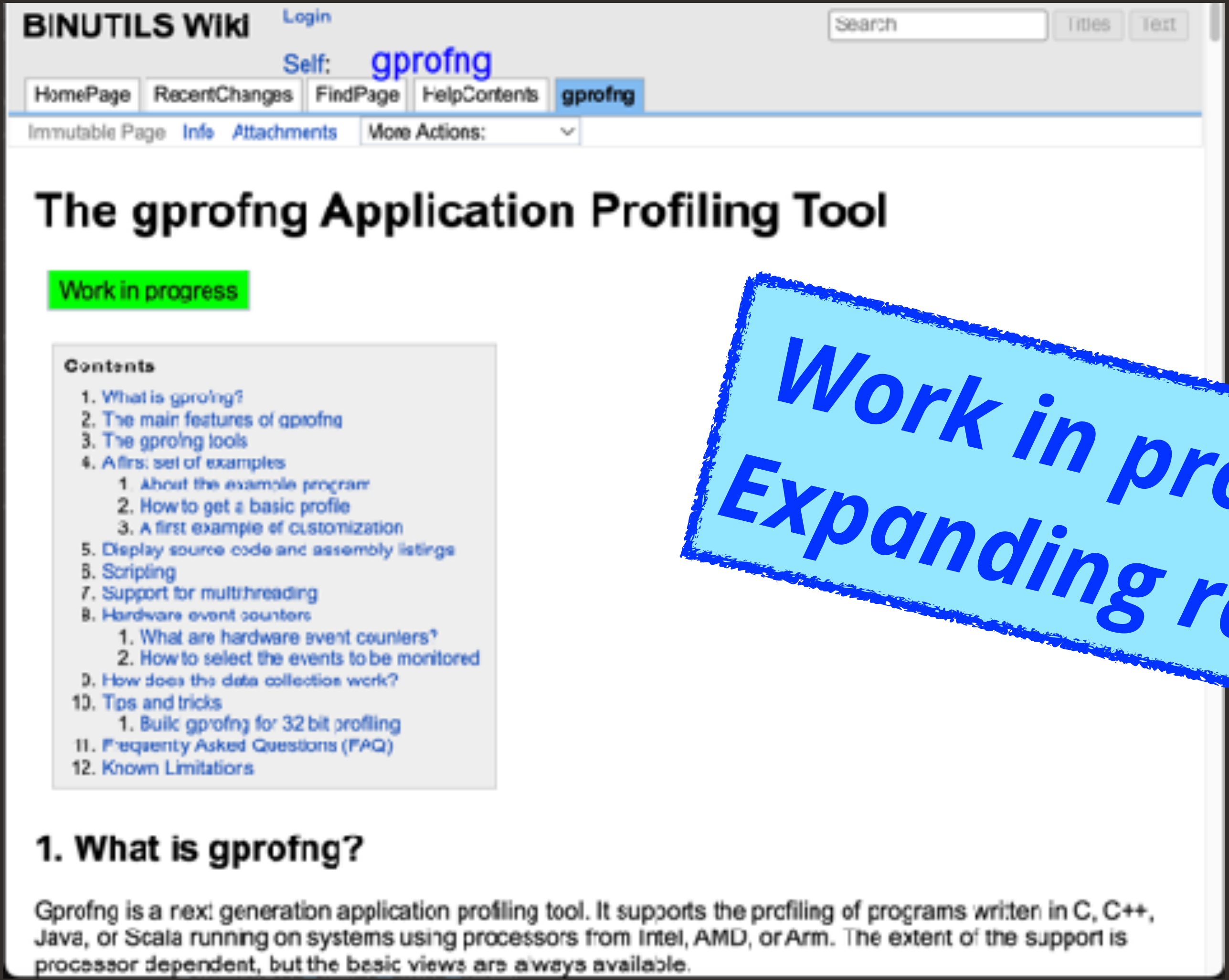
- **addr2line** - Converts addresses into filenames and line numbers.
- **ar** - A utility for creating, modifying and extracting from archives.
- **c++filt** - Filter to demangle encoded C++ symbols.
- **dlltool** - Creates files for building and using DLLs.
- **elfedit** - Allows alteration of ELF format files.
- **gprof** - Displays profiling information.
- **gprofng** - Collects and displays application performance data.
- **nlmconv** - Converts object code into an NLM.
- **nm** - Lists symbols from object files.
- **objcopy** - Copies and translates object files.
- **objdump** - Displays information from object files.
- **ranlib** - Generates an index to the contents of an archive.
- **readelf** - Displays information from any ELF format object file.
- **size** - Lists the section sizes of an object or archive file.
- **strings** - Lists printable strings from files.
- **strip** - Discards symbols.
- **windmc** - A Windows compatible message compiler.
- **windres** - A compiler for Windows resource files.

The binutils Home Page:
https://www.gnu.org/software/binutils/

gprofng - Collects and displays application performance data.

Hyperlink to the gprofng Wiki

The gprofng Wiki on sourceware.org



BINUTILS Wiki Login

Self: gprofng

Search | Titles | Text

HomePage | RecentChanges | FindPage | HelpContents | gprofng

Immutable Page | Info | Attachments | More Actions:

The gprofng Application Profiling Tool

Work in progress

Contents

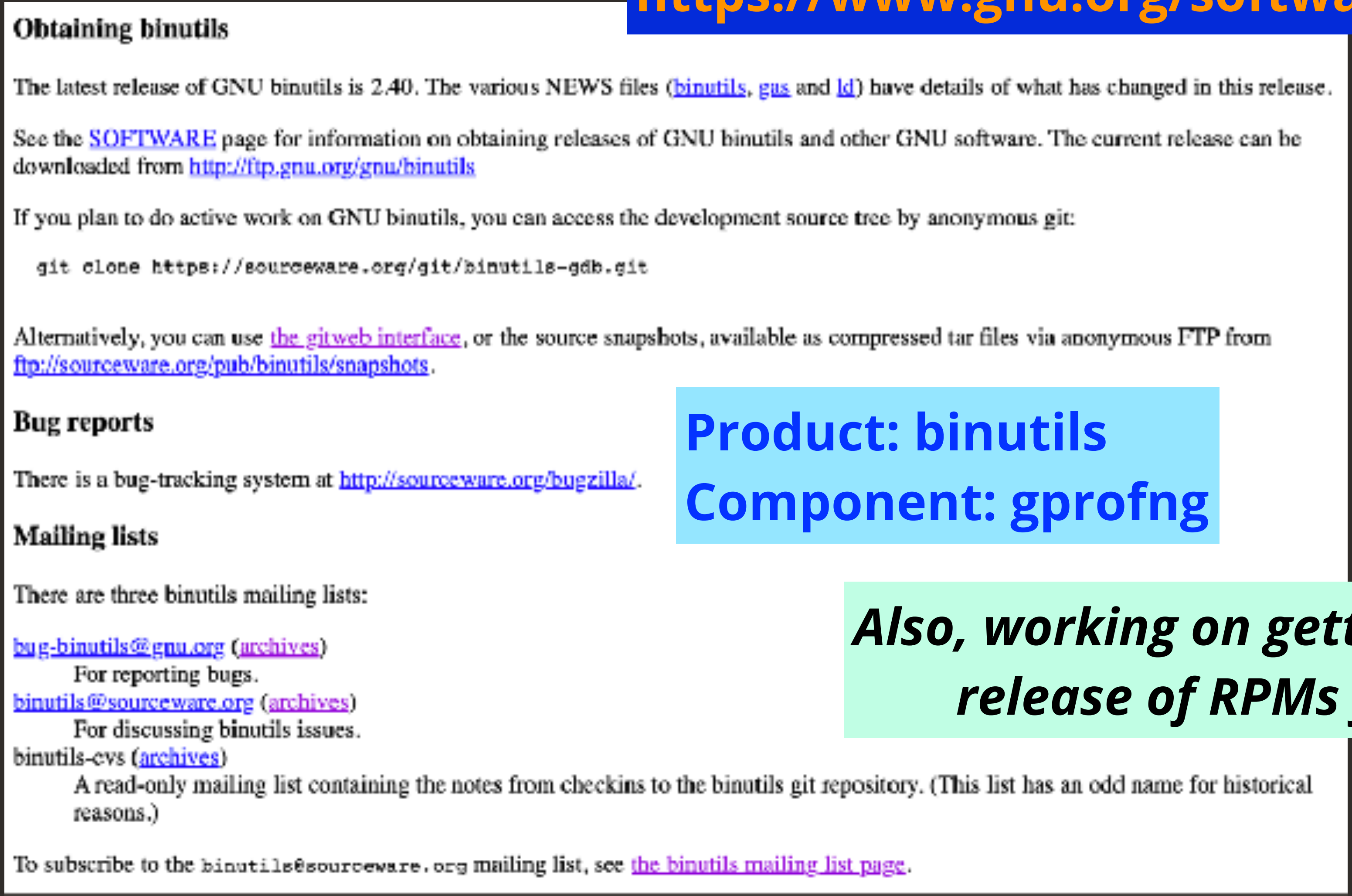
1. What is gprofng?

Gprofng is a next generation application profiling tool. It supports the profiling of programs written in C, C++, Java, or Scala running on systems using processors from Intel, AMD, or Arm. The extent of the support is processor dependent, but the basic views are always available.

Work in progress
Expanding rapidly

How to Get Your Copy

The binutils Home Page:
https://www.gnu.org/software/binutils/



Obtaining binutils

The latest release of GNU binutils is 2.40. The various NEWS files (binutils, gas and ld) have details of what has changed in this release.

See the SOFTWARE page for information on obtaining releases of GNU binutils and other GNU software. The current release can be downloaded from http://ftp.gnu.org/gnu/binutils

If you plan to do active work on GNU binutils, you can access the development source tree by anonymous git:

```
git clone https://sourceware.org/git/binutils-gdb.git
```

Alternatively, you can use the gitweb interface, or the source snapshots, available as compressed tar files via anonymous FTP from ftp://sourceware.org/pub/binutils/snapshots.

Bug reports

There is a bug-tracking system at http://sourceware.org/bugzilla/.

Mailing lists

There are three binutils mailing lists:

bug-binutils@gnu.org (archives)
For reporting bugs.
binutils@sourceware.org (archives)
For discussing binutils issues.
binutils-cvs (archives)
A read-only mailing list containing the notes from checkins to the binutils git repository. (This list has an odd name for historical reasons.)

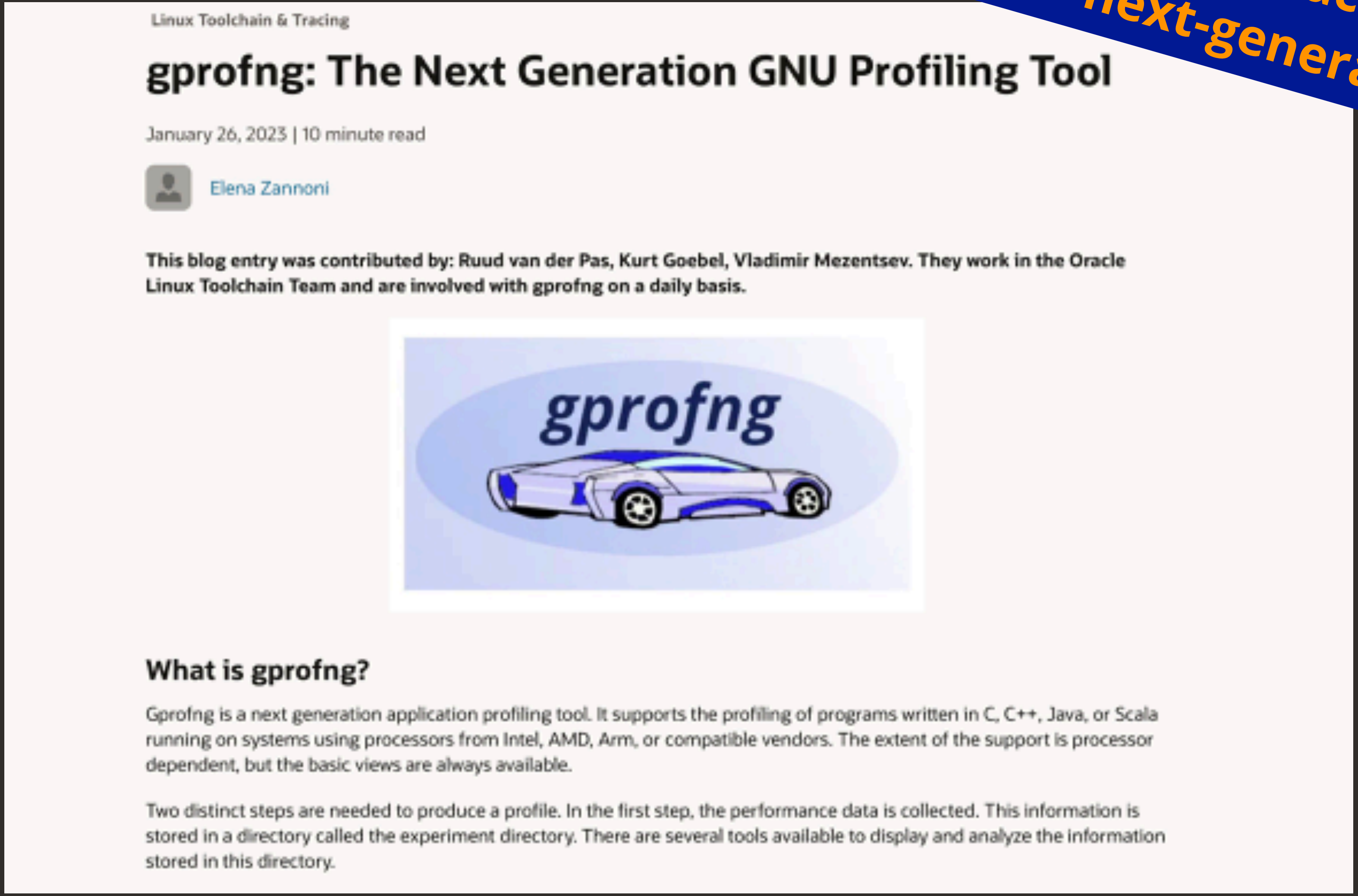
To subscribe to the binutils@sourceware.org mailing list, see the binutils mailing list page.

Product: binutils
Component: gprofng

Also, working on getting approval for the release of RPMs for OL8 and OL9

More Information on gprofng

https://blogs.oracle.com/linux/post/gprofng-the-next-generation-gnu-profiling-tool



Linux Toolchain & Tracing

gprofng: The Next Generation GNU Profiling Tool

January 26, 2023 | 10 minute read

Elena Zannoni

This blog entry was contributed by: Ruud van der Pas, Kurt Goebel, Vladimir Mezentsev. They work in the Oracle Linux Toolchain Team and are involved with gprofng on a daily basis.

What is gprofng?

Gprofng is a next generation application profiling tool. It supports the profiling of programs written in C, C++, Java, or Scala running on systems using processors from Intel, AMD, Arm, or compatible vendors. The extent of the support is processor dependent, but the basic views are always available.

Two distinct steps are needed to produce a profile. In the first step, the performance data is collected. This information is stored in a directory called the experiment directory. There are several tools available to display and analyze the information stored in this directory.

Gprofng - Collects and Displays Application Performance Data

- Languages supported: C, C++, Java, and Scala
- Full support for gcc compilers
- Fortran - full support for F77 and F95
  - Limited testing with gfortran v12 and -std=f2018 looks encouraging, but TBD
- 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?

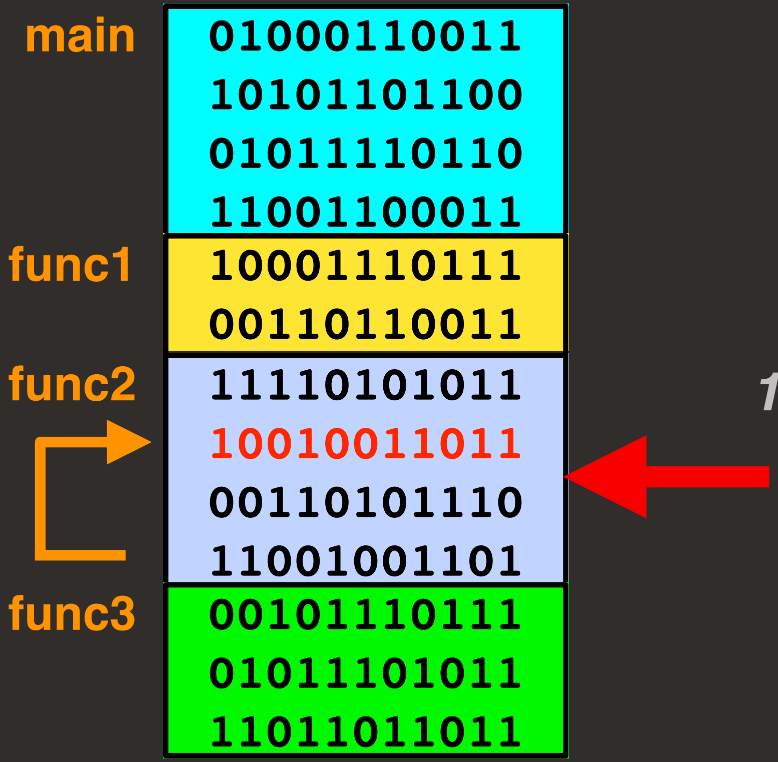
- A two step approach
  - First, collect the performance data on the target executable
  - Next, display the data
- Information is available at the function, source, and disassembly level
- Thanks to multiple views, already a single run can provide a lot of insight
- **Scripting** support to generate 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
Limited 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

Function	Time (s)	Percentage
<Total>	18	100.0%
func2	10	55.6%
func1	5	27.8%
func3	2	11.1%
main	1	5.6%

The gprofng Command Structure

General syntax:

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

Examples:

```
$ gprofng collect app   -O my-experiment.er
$ gprofng display text     my-experiment.er
$ gprofng archive          my-experiment.er
```

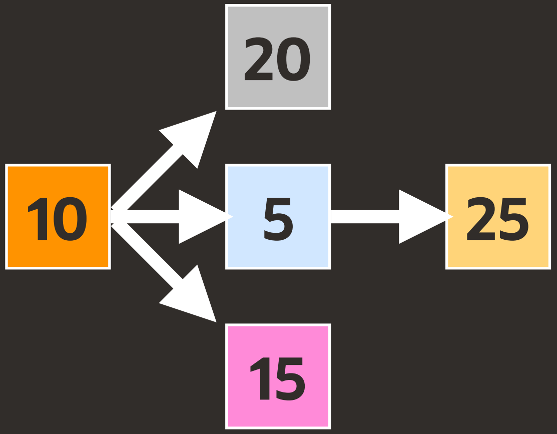
An Overview of the Commands

Command	Functionality
`$ gprofng collect app`	Collect the performance data
`$ gprofng display text`	Display the performance data in ASCII format
`$ gprofng display html`	Generate html structure and view in a browser (currently x86_64 only)
`$ gprofng display gui`	Launch the GUI (available soon)
`$ gprofng archive`	Archive an experiment directory
`$ gprofng display src`	Display the source and disassembly of an object file

Intermezzo - About Inclusive and Exclusive Metrics

This is an important concept in profiling tools

- 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



Function	Inclusive time	Exclusive time
A	75	10
B	20	20
C	30	5
D	15	15
E	25	25

Three Very Cool Features

Scripting - Produce ASCII profiles in “batch mode”

- May be used for automated QA testing for example

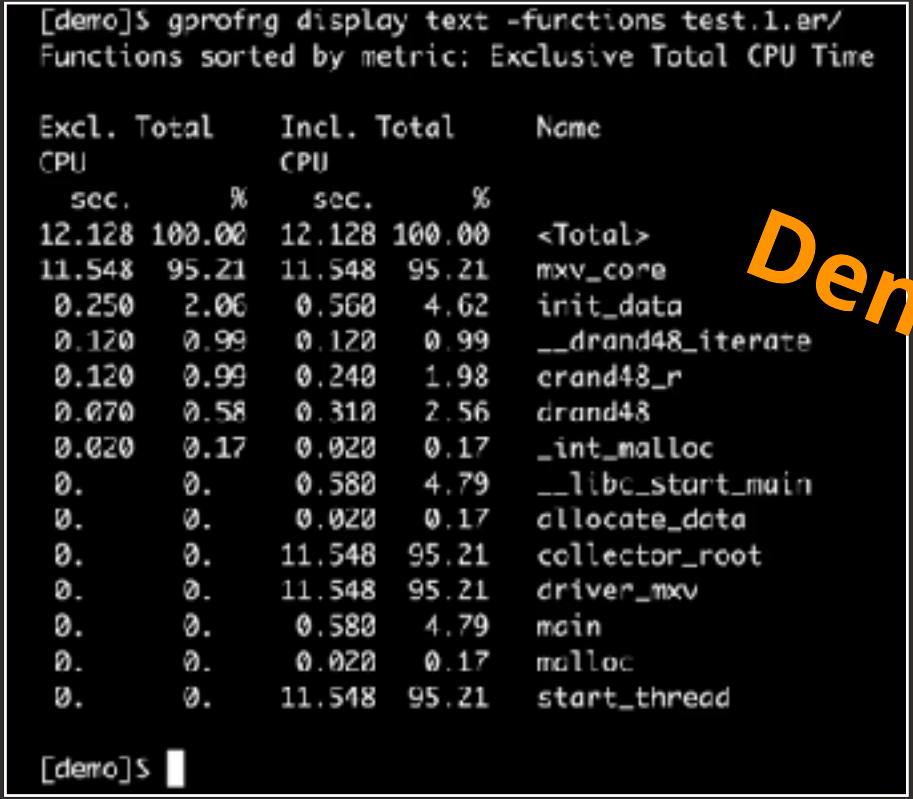
Comparison of Profiles - Compare profiling data

- A really cool feature! And very useful too ;-)
- Comparison of profiles is supported at different levels
- Supported in text mode and through the GUI

The Timeline [GUI] - A color coded view of the run time behaviour

- Provides immediate insight into the dynamics
- For example, gaps in the execution

Getting started



```
[demo]$ gprofng display text -functions test.1.er/
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total     Incl. Total      Name
CPU             CPU
  sec.       %    sec.       %
12.128 100.00  12.128 100.00    <Total>
11.548  95.21  11.548  95.21    mxv_core
 0.250   2.06   0.560   4.62    init_data
 0.120   0.99   0.120   0.99    __drand48_iterate
 0.120   0.99   0.240   1.98    erand48_r
 0.070   0.58   0.310   2.56    drand48
 0.020   0.17   0.020   0.17    _int_malloc
 0.      0.     0.580   4.79    __libc_start_main
 0.      0.     0.020   0.17    allocate_data
 0.      0.    11.548  95.21    collector_root
 0.      0.    11.548  95.21    driver_mxv
 0.      0.     0.580   4.79    main
 0.      0.     0.020   0.17    malloc
 0.      0.    11.548  95.21    start_thread

[demo]$
```

Demo Time!

Comparison of Profiles - Generate the Data

```
$ gprofng collect app -O mxv.hwc.1.thr.er -h llm \
  ./mxv-pthreads -m 3000 -n 2000 -t 1
Creating experiment directory 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 -m 3000 -n 2000 -t 2
Creating experiment directory mxv.hwc.2.thr.er (Process ID: 23462) ...
mxv: error check passed - rows = 3000 columns = 2000 threads = 2
```

Compare the Absolute Numbers

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

```
                  mxv.hwc.comp.1.thr.er  mxv.hwc.comp.2.thr.er
Name              Excl. Last-Level       Excl. Last-Level
                  Cache Misses           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
```

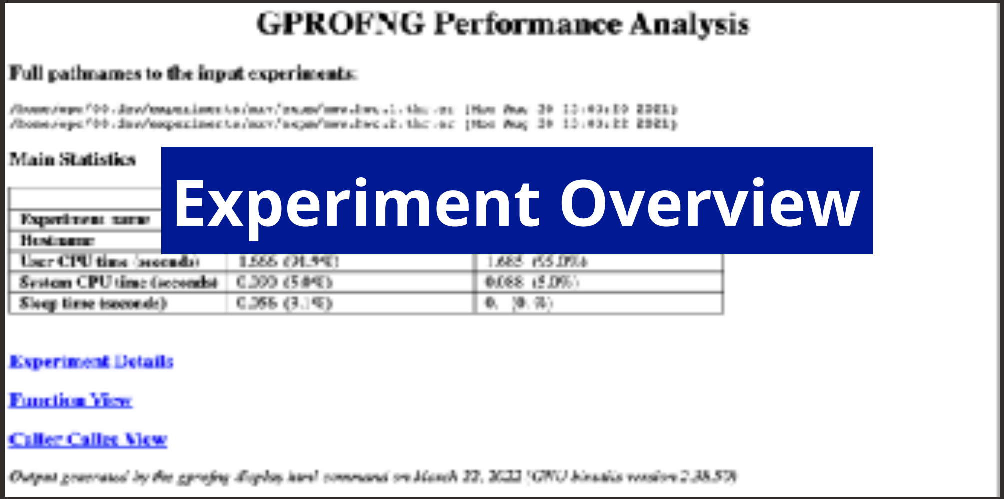
```
                   mxv.hwc.comp.1.thr.er  mxv.hwc.comp.2.thr.er
Name               Excl. Last-Level       Excl. Last-Level
                   Cache Misses           Cache Misses
                                              ratio
 <Total>           122709276              x   0.788
 mxv_core          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
```

Generate HTML

gprofng display html

The “gprofng display html” command creates an HTML structure



GPROFNG Performance Analysis

Full pathnames to the input experiments:

```
[illegible]
[illegible]
```

Main Statistics

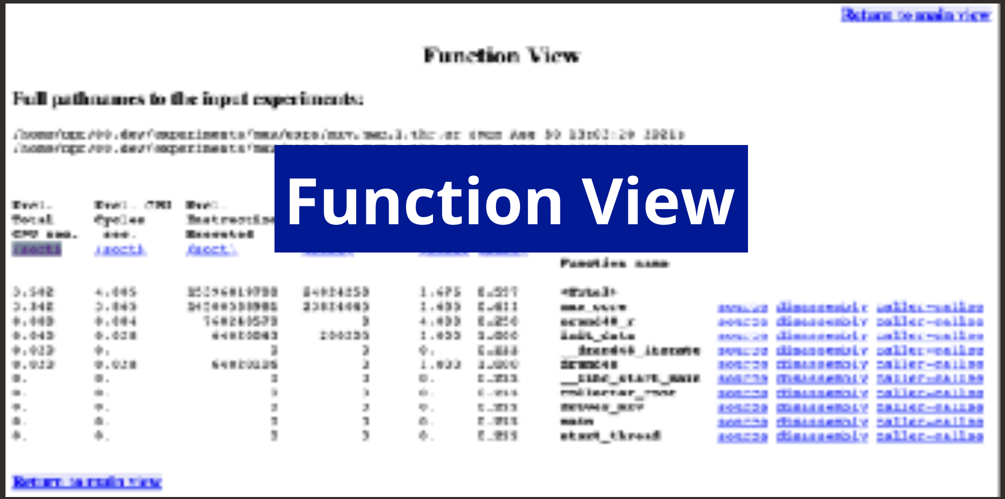
Experiment name	[illegible]	[illegible]
Hostname	[illegible]	[illegible]
User CPU time (seconds)	1.565 (91.9%)	1.685 (95.0%)
System CPU time (seconds)	0.200 (5.0%)	0.088 (5.0%)
Sleep time (seconds)	0.256 (3.1%)	0. (0.%)

Experiment Details

Function View

Caller Callee View

Output generated by the gprofng display html command on March 22, 2022 (GNU binutils version 2.38.50)



Return to main view

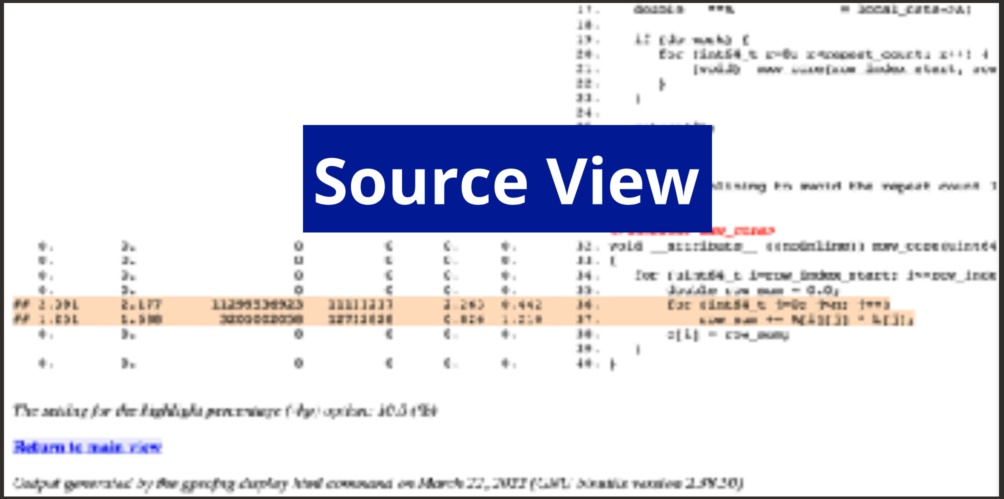
Function View

Full pathnames to the input experiments:

```
[illegible]
```

[illegible]

Return to main view

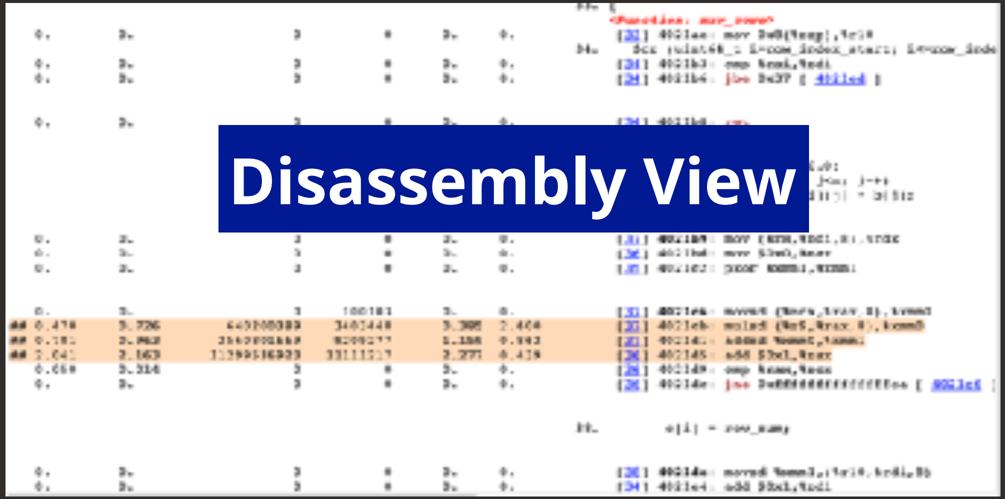


```
[illegible]
```

The setting for the highlight percentage (-hp) option: 10.0 (%)

Return to main view

Output generated by the gprofng display html command on March 22, 2022 (GNU binutils version 2.38.50)



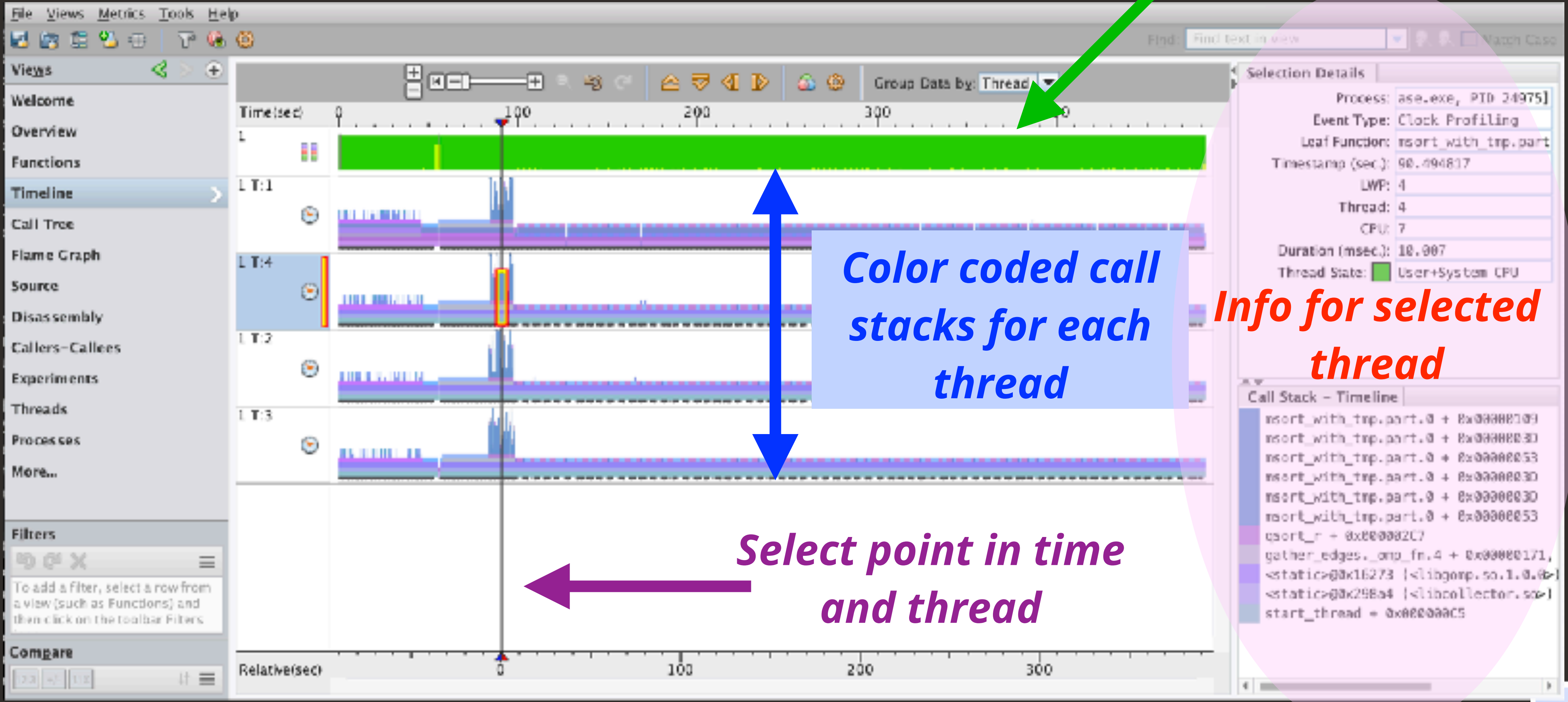
```
[illegible]
```

Sneak Preview

The gprofng GUI

The GUI Sneak Preview - The Timeline

Operating System State



To add a filter, select a row from a view (such as Functions) and then click on the toolbar Filters

Process:	ase.exe, PID 24975]
Event Type:	Clock Profiling
Leaf Function:	msort_with_tmp.part
Timestamp (sec.):	90.494817
LWP:	4
Thread:	4
CPU:	7
Duration (msec.):	10.007
Thread State: 🟩	User+System CPU

```
msort_with_tmp.part.0 + 0x00000109
msort_with_tmp.part.0 + 0x0000003D
msort_with_tmp.part.0 + 0x00000053
msort_with_tmp.part.0 + 0x0000003D
msort_with_tmp.part.0 + 0x0000003D
msort_with_tmp.part.0 + 0x00000053
qsort_r + 0x000002C7
gather_edges._omp_fn.4 + 0x00000171,
<static>@0x16273 (<libgomp.so.1.0.0>)
<static>@0x298a4 (<libcollector.so>)
start_thread + 0x000000C5
```

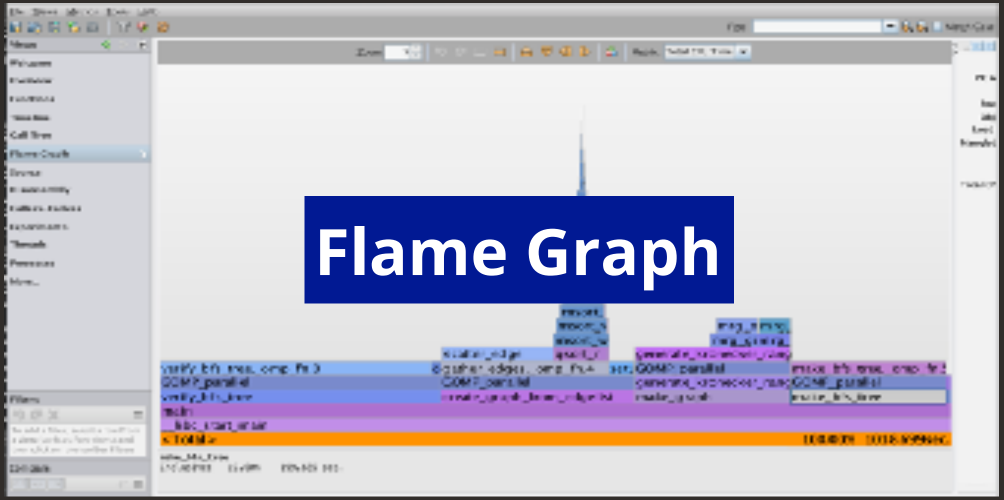
Color coded call stacks for each thread

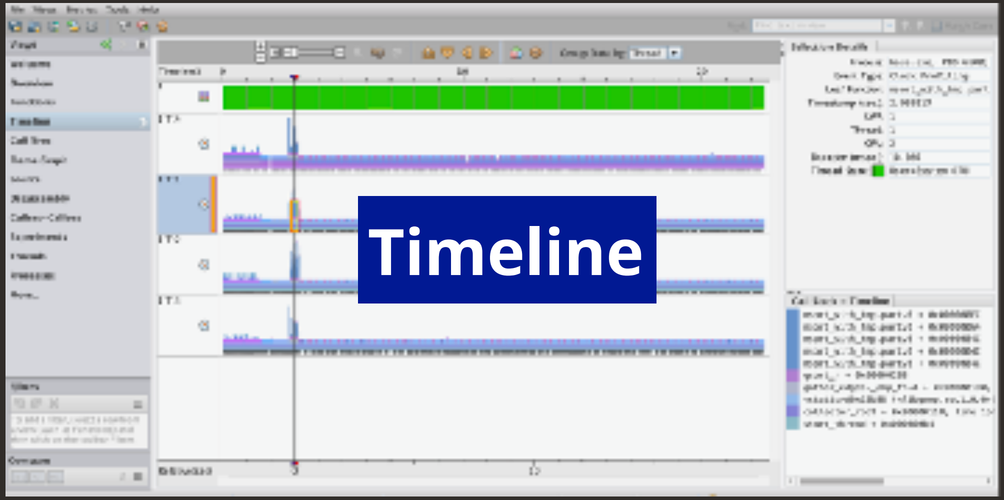
Info for selected thread

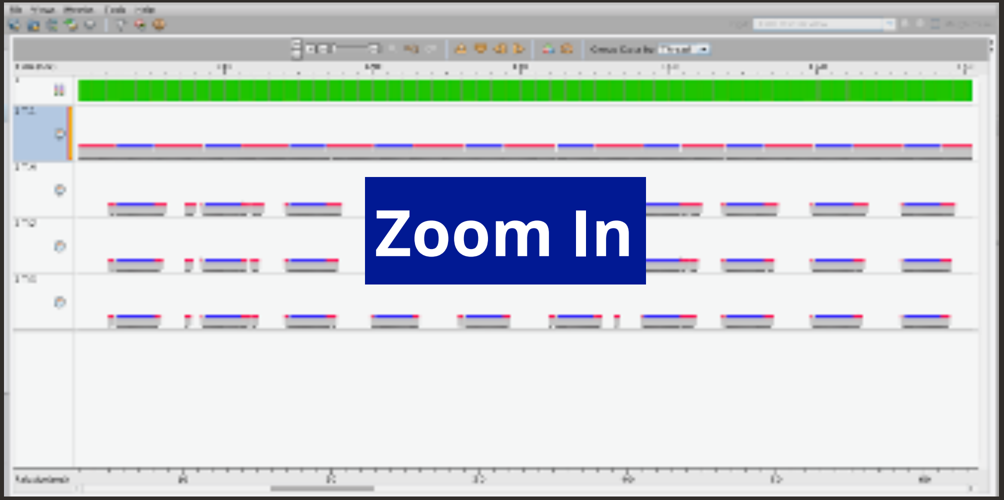
Select point in time and thread

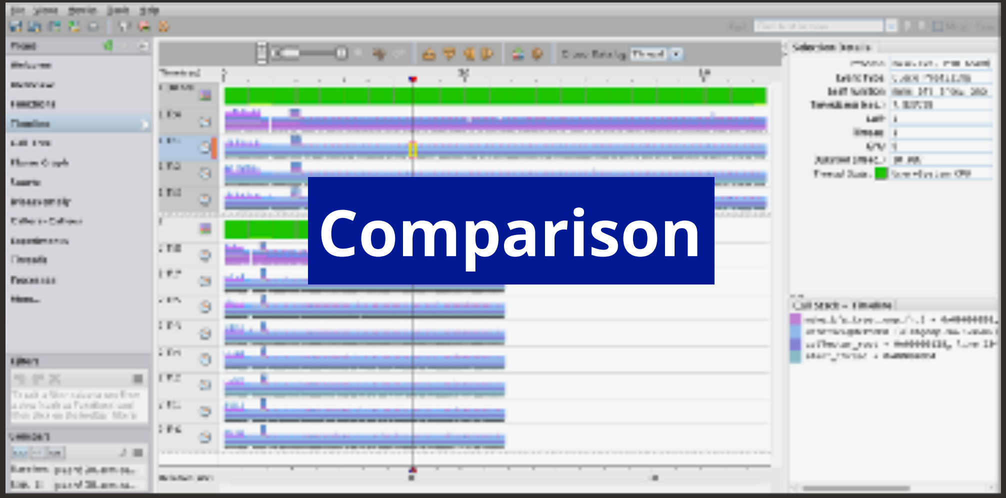
gprofng

The gprofng GUI Sneak Preview - Some Views









gprofng

Future Directions/1

- **Help users to get started**
  - Growing user base
- **Support users analyzing performance**
- **Main priorities for development**
  - Expand and update the Wiki and other documentation
  - Produce collaborative info for gprofng developers
  - Make RPMs for gprofng available for the RH universe
    - RPMs for Fedora (x86_64 and aarch64) on https://pkgs.org/download/gprofng
  - Support for aarch64 in “gprofng display html”
  - Support porting and distribution on other platforms
  - Make the GUI (to graphically display and analyze the experiment data) available
    - This will be a Savannah project

Future Directions/2

Other topics on the wish list

- Support for hardware event counters on more recent processors
- Provide additional metrics with call stack sampling
- Support remote analysis through a client-server set up
- Attach to a running process
- Further develop the “gprofng display html” functionality
- Write a porting guide (i.e. what does it take to port gprofng to other platforms)
- Investigate supporting AutoFDO
- …

Please send your feedback, or if you're interested to help, to binutils@sourceware.org

Thank You!

Time for Q&A!