ORACLE

NHR FAU

gprofng: The Next Generation GNU Profiler

Ruud van der Pas, Vladimir Mezentsev

Compilers and Toolchain Team
Oracle Linux Engineering and Virtualization

NHR PerfLab Seminar
April 12, 2022

Outline

- 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 go into a lot of details
  - The gprofng repo contains a mini tutorial in Texinfo format
- We conclude with a brief overview of our short term direction

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
```

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
> url:
> branch:

Much better. I think that this b
the mainline.

Not being a git expert, I am going to ask for your advic
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

...

How to Get Your Copy

```
[gprofng]$
[gprofng]$ git clone git://sourceware.org/git/binutils-gdb.git
Cloning into 'binutils-gdb'...
remote: Enumerating objects: 59458, done.
remote: Counting objects: 100% (59458/59458), done.
remote: Compressing objects: 100% (47999/47999), done.
remote: Total 1103246 (delta 16844), reused 15307 (delta 11314), pack-reused 1043788
Receiving objects: 100% (1103246/1103246), 421.75 MiB | 13.24 MiB/s, done.
Resolving deltas: 100% (907652/907652), done.
Updating files: 100% (37562/37562), done.
[gprofng]$
```

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

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)

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?

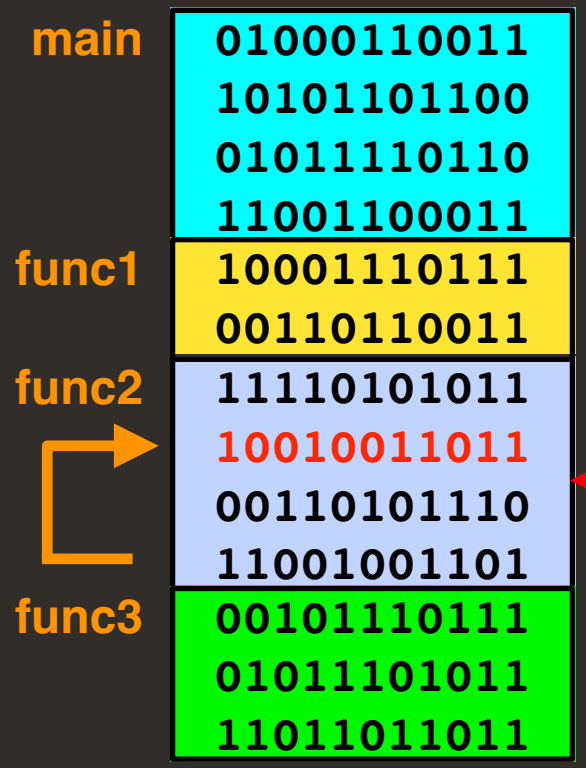
- Two step approach
  - First, collect the performance data on the target executable
  - Next, display the data
- Information is available at the function, source, and disassembly level
- Thanks to multiple views, already a single run can provide a lot of insight
- **Scripting** support to generate and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


An Overview of the Commands

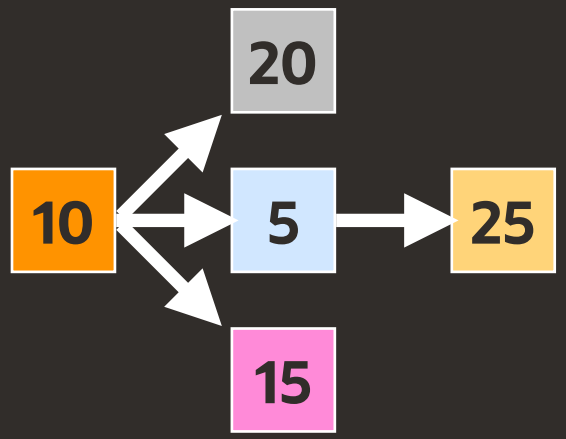
Command	Functionality
`$ gprofng collect app`	Collect the performance data
`$ gprofng display text`	Display the performance data in ASCII format
`$ gprofng archive`	Archive an experiment directory
`$ gprofng display src`	Display the source and disassembly of an object file
`$ gprofng display html*`	Generate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

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

Demo Time!

```
demo$ gprofng display text -limit 10 -functions test.1.er
Print limit set to 10
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
0.008     0.010      _int_malloc
0.001     0.001      brk
0.001     0.002      sysmalloc
0.        0.001      __default_morecore

demo$
```

More Examples

- How to get a basic profile
- Display the function, source, and disassembly information
- Customize the data collection and displays
- Scripting
- The call tree and callers-callees view
- Support for multithreading
- Basic filters
- Comparison of profiles

Create a Basic Profile - 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 directory 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 Most Time Consuming (“Hot”) Lines

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

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

Excl.      Incl.     Name
Total      Total
CPU sec.   CPU sec.
2.272      2.272     <Total>
1.687      1.687     mxv_core, line 36 in "mxv.c"
0.473      0.473     mxv_core, line 37 in "mxv.c"
0.032      0.088     init_data, line 72 in "manage_data.c"
0.030      0.043     <Function: erand48_r, instructions without line numbers>
0.013      0.013     <Function: __drand48_iterate, instructions without ...>
0.013      0.056     <Function: drand48, instructions without line numbers>
    … 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(
                               uint64_t row_index_start,
                               uint64_t row_index_end,
                               uint64_t m, uint64_t n, double **restrict A,
                               double *restrict b, double *restrict c)
   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>
   0.        0.                 [34]   4021c4:  ret
                            35.        double row_sum = 0.0;
                            36.        for (int64_t j=0; j<n; j++)
                            37.            row_sum += A[i][j] * b[j];
   0.        0.                 [37]   4021c5:  mov    (%r8,%rdi,8),%rdx
   0.        0.                 [36]   4021c9:  mov    $0x0,%eax
   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 Information Displayed

```
$ 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

```
$ cat my-script

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

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

The Output

Functions sorted by metric: Exclusive Total CPU Time

Name	Incl. Total CPU sec.	%	Excl. Total CPU sec.	%
<Total>	2.272	100.00	2.272	100.00
mxv_core	2.159	95.00	2.159	95.00
init_data	0.102	4.48	0.054	2.37
erand48_r	0.035	1.54	0.025	1.10
drand48	0.048	2.11	0.013	0.57

Customize the Data Collection - An Example

```
$ 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
- …

The Call Tree Command

```
$ gprofng collect app -O mxv.ct.1.thr.er ./mxv-pthreads.exe -m 3000 -n 2000 -t 1
Creating experiment directory mxv.ct.1.thr.er (Process ID: 3214927) ...
mxv: error check passed - rows = 3000 columns = 2000 threads = 1

$ gprofng display text -calltree mxv.ct.1.thr.er
```

This command shows the dynamic call tree for the experiment

The Call Tree - Output

```
Attr.       Name
Total
CPU sec.
1.877       +-<Total>
1.784         +-start_thread
1.784         |  +-collector_root
1.784         |    +-driver_mxv
1.784         |      +-mxv_core
0.094         +-__libc_start_main
0.094           +-main
0.086             +-init_data
0.048             |  +-drand48
0.033             |    +-erand48_r
0.012             |      +-__drand48_iterate
0.008             +-allocate_data
0.008               +-malloc
0.008                 +-_int_malloc
0.001                   +-sysmalloc
0.001                     +-__default_morecore
0.001                       +-sbrk
0.001                         +-brk
```

The Call Tree - A More Realistic Example

```
Functions Call Tree. Metric: Attributed Total CPU Time

Attr.      Attr. CPU  Attr.          Attr.          Name
Total      Cycles     Instructions   Last-Level
CPU sec.      sec.    Executed       Cache Misses
174.664    179.250    175838403203   1166209617     +-<Total>
174.309    179.188    175627135852   1166149390       +-GOMP_parallel
109.305    113.675     65537671505    695734585       |  +-verify_bfs_tree._omp_fn.0
  0.001      0.                 0            0       |  |  +-<static>@0x1dbb9 (<libgomp.so.1.0.0>)
 37.689     38.553     24468594304    416234675       |  +-make_bfs_tree._omp_fn.0
  0.553      0.498       723428864      7568862       |  |  +-int64_cas
  0.003      0.                 0            0       |  |  +-<static>@0x1dbb9 (<libgomp.so.1.0.0>)
 13.630     13.065     67810397138       250276       |  +-generate_kronecker_range._omp_fn.0
  9.771      9.296     45556891896       130140       |  |  +-make_one_edge
  6.901      6.680     35035143217        90100       |  |  |  +-generate_4way_bernoulli
  3.966      3.839     20051185809        40046       |  |  |  |  +-mrg_get_uint_orig
  3.430      3.346     17759255948        40046       |  |  |  |    +-mrg_orig_step
  1.333      1.236      6792560991        20021       |  |  |  |      +-mod_mac_y
  1.151      1.088      5652997430        10010       |  |  |  |      |  +-mod_mac
  1.135      1.178      5851463098        10011       |  |  |  |      +-mod_mul_x
  0.999      1.016      5057610084        10011       |  |  |  |        +-mod_mul
```

The Callers-Callees View - Output Fragment

```
$ gprofng display text -callers-callees mxv.1.thr.er
```

```
Attr.      Name
Total
CPU sec.
0.114      __libc_start_main
0.         *main
0.102      init_data
0.011      allocate_data
0.001      check_results

Attr.      Name
Total
CPU sec.
0.011      allocate_data
0.001      check_results
0.         *malloc
0.012      _int_malloc
```

Callers

Callees

Callers

Callees

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 and #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
```

Some More about Filters

- More filters are supported in “gprofng display text”
- The syntax is rather basic and not yet well documented
- *The upcoming GUI will have very easy support for filters*
- A C-style grammar for operands and operators is supported
- For example to select thread ID 1: THRID == 1

A Selection of the Commands and Operands related to Filters*

Commands

```
threads
cpus
samples
seconds
processes
experiment_ids
GCEvents
```

Operands

```
THRID
LWPID
CPUID
SAMPLE
EXPID
PID
```

*) This is a selection only

An Example using Filters - The Script

```
metrics name:i.%totalcpu:e.%totalcpu
sort e.totalcpu
# Show all the CPUs that have been used
cpus
# Show all the threads
threads
# Select a combination of CPU and Thread
filters "CPUID == 11 && THRID == 2"
# Get the function overview for this combination
functions
```

Note: Reset the filters by defining a filter called “1” (i.e. filters 1)

The Relevant Output/1

```
# Show all the CPUs that have been used
Objects sorted by metric: Exclusive Total CPU Time

Name      Excl. Total
          CPU
           sec.      %
 <Total> 2.261 100.00
 CPU 8   1.189  52.60
 CPU 11  1.072  47.40
```

```
# Show all the threads
Objects sorted by metric: Exclusive Total CPU Time

Name                    Excl. Total
                        CPU
                         sec.      %
 <Total>                2.261 100.00
 Process 1, Thread 3    1.076  47.57
 Process 1, Thread 2    1.072  47.40
 Process 1, Thread 1    0.114   5.03
```

The Relevant Output/2

```
# Select a combination of CPU and Thread
current filter setting: "CPUID == 11 && THRID == 2"
# Get the function overview for this combination
Functions sorted by metric: Exclusive Total CPU Time

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

Comparison of Profiles - Generate the Data

```
$ 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 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
```

More to Explore

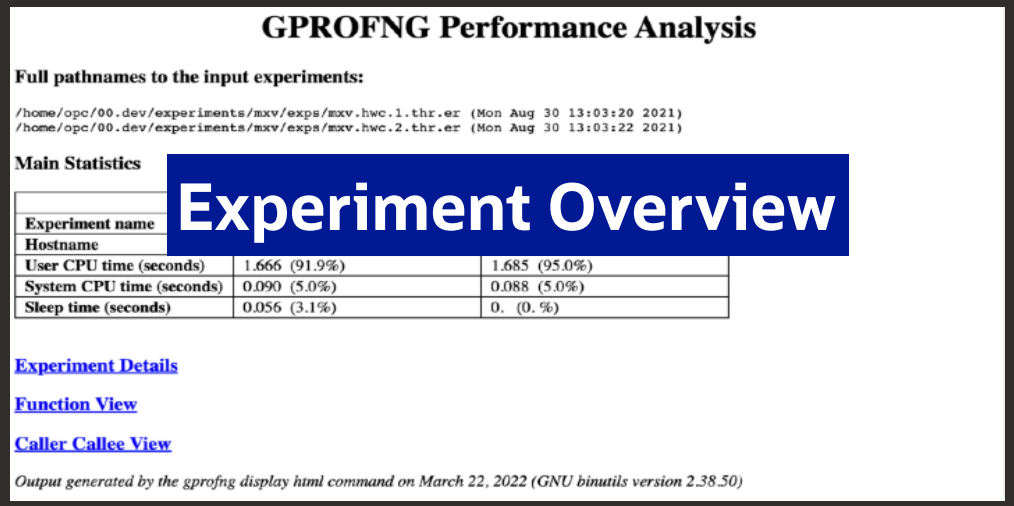
- Additional filters
- More information on the experiment(s)
- Additional customization
- Support for hardware event counters
- …

Future Directions

- Fix bugs
- Help users to get started
- Top priorities for development
  - Make a porting guide available
  - Make a GUI available (to display and analyze the experiment data)
  - Finish the tool that generates an HTML based system to browse the data
  - Expand the User Guide
- 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
  - ...

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

The gprofng HTML Based Output - A Sneak Preview



GPROFNG Performance Analysis

Full pathnames to the input experiments:

/home/opc/00.dev/experiments/mxv/exps/mxv.hwc.1.thr.er (Mon Aug 30 13:03:20 2021)
/home/opc/00.dev/experiments/mxv/exps/mxv.hwc.2.thr.er (Mon Aug 30 13:03:22 2021)

Main Statistics

Experiment name		
Hostname		
User CPU time (seconds)	1.666 (91.9%)	1.685 (95.0%)
System CPU time (seconds)	0.090 (5.0%)	0.088 (5.0%)
Sleep time (seconds)	0.056 (3.1%)	0. (0. %)

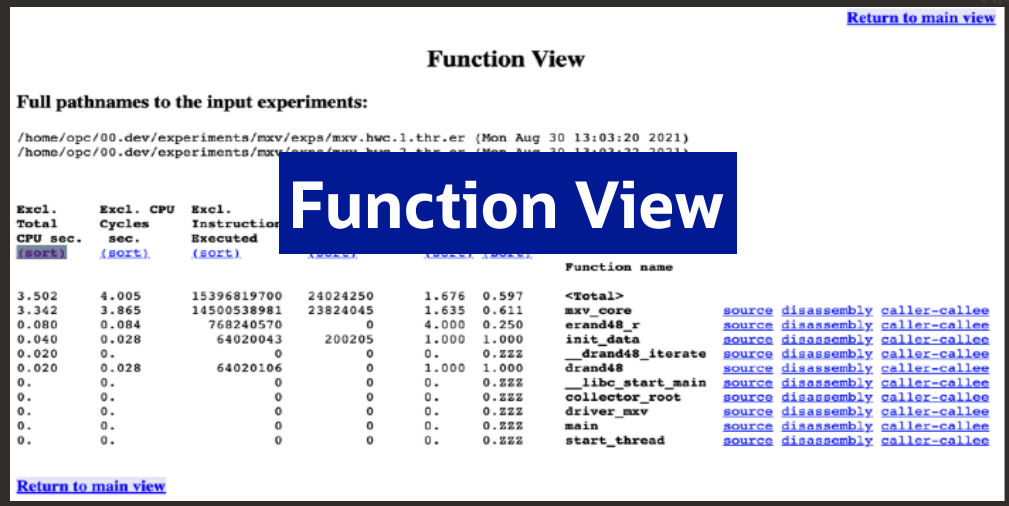
Experiment Details

Function View

Caller Callee View

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

Experiment Overview



Return to main view

Function View

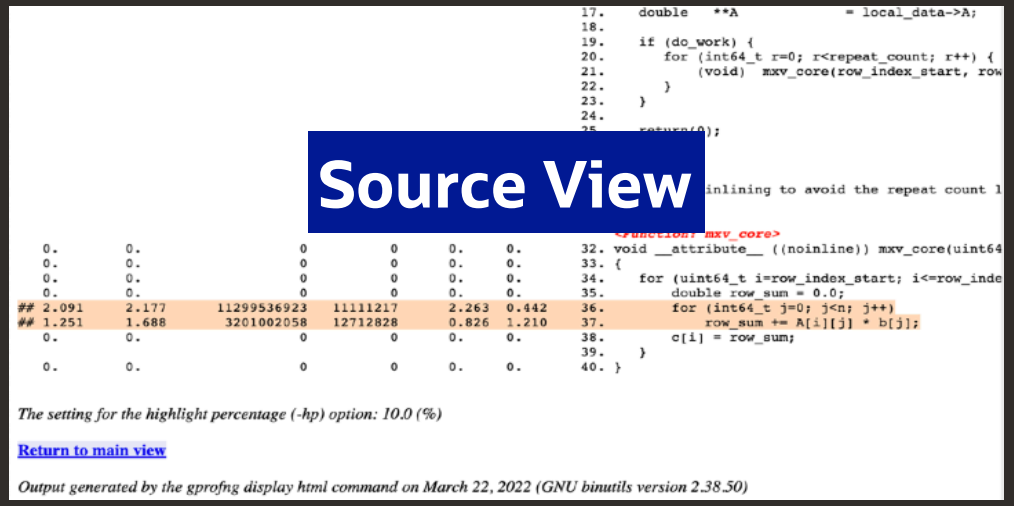
Full pathnames to the input experiments:

/home/opc/00.dev/experiments/mxv/exps/mxv.hwc.1.thr.er (Mon Aug 30 13:03:20 2021)

Excl. Total CPU sec. (sort)	Excl. CPU Cycles sec. (sort)	Excl. Instruction Executed (sort)				Function name	
3.502	4.005	15396819700	24024250	1.676	0.597	<Total>	
3.342	3.865	14500538981	23824045	1.635	0.611	mxv_core	source disassembly caller-callee
0.080	0.084	768240570	0	4.000	0.250	erand48_r	source disassembly caller-callee
0.040	0.028	64020043	200205	1.000	1.000	init_data	source disassembly caller-callee
0.020	0.	0	0	0.	0.???	__drand48_iterate	source disassembly caller-callee
0.020	0.028	64020106	0	1.000	1.000	drand48	source disassembly caller-callee
0.	0.	0	0	0.	0.???	__libc_start_main	source disassembly caller-callee
0.	0.	0	0	0.	0.???	collector_root	source disassembly caller-callee
0.	0.	0	0	0.	0.???	driver_mxv	source disassembly caller-callee
0.	0.	0	0	0.	0.???	main	source disassembly caller-callee
0.	0.	0	0	0.	0.???	start_thread	source disassembly caller-callee

Return to main view

Function View



```
17.    double   **A              = local_data->A;
18.
19.    if (do_work) {
20.      for (int64_t r=0; r<repeat_count; r++) {
21.        (void)  mxv_core(row_index_start, row
22.      }
23.    }
24.
25.    return(0);
```

inlining to avoid the repeat count l

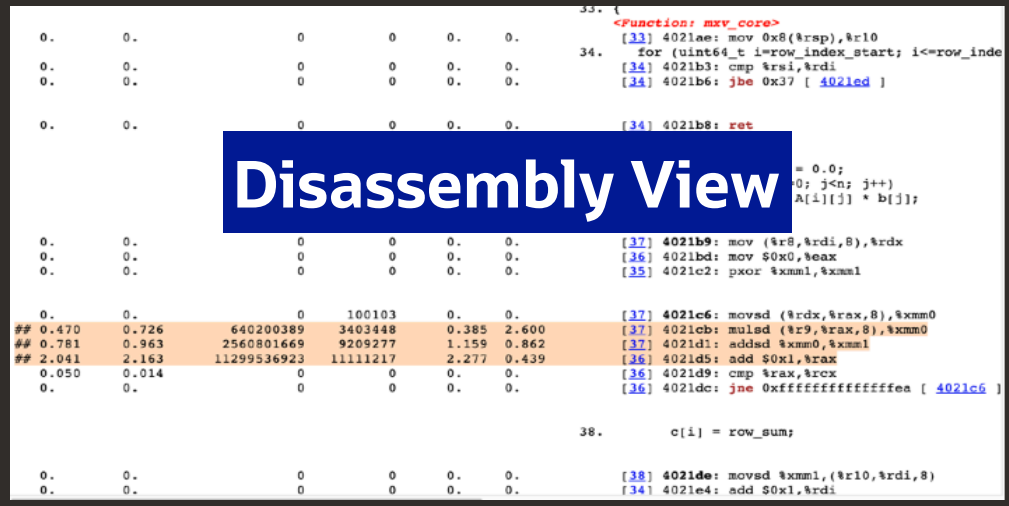
```
<Function: mxv_core>
0.  0.  0  0  0.  0.  32. void __attribute__ ((noinline)) mxv_core(uint64
0.  0.  0  0  0.  0.  33. {
0.  0.  0  0  0.  0.  34.   for (uint64_t i=row_index_start; i<=row_inde
0.  0.  0  0  0.  0.  35.     double row_sum = 0.0;
## 2.091  2.177  11299536923  11111217  2.263  0.442  36.     for (int64_t j=0; j<n; j++)
## 1.251  1.688  3201002058  12712828  0.826  1.210  37.       row_sum += A[i][j] * b[j];
0.  0.  0  0  0.  0.  38.     c[i] = row_sum;
0.  0.  0  0  0.  0.  39.   }
                     40. }
```

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)

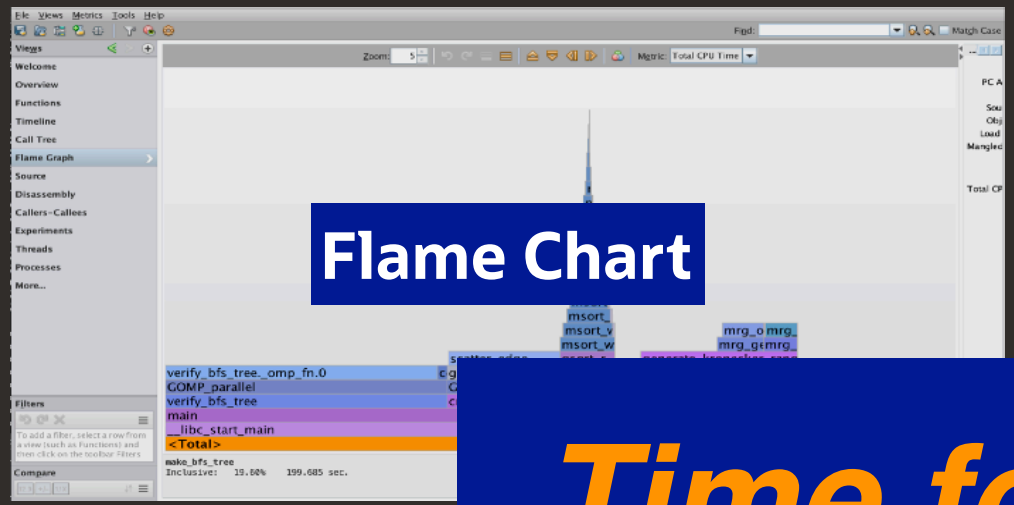
Source View

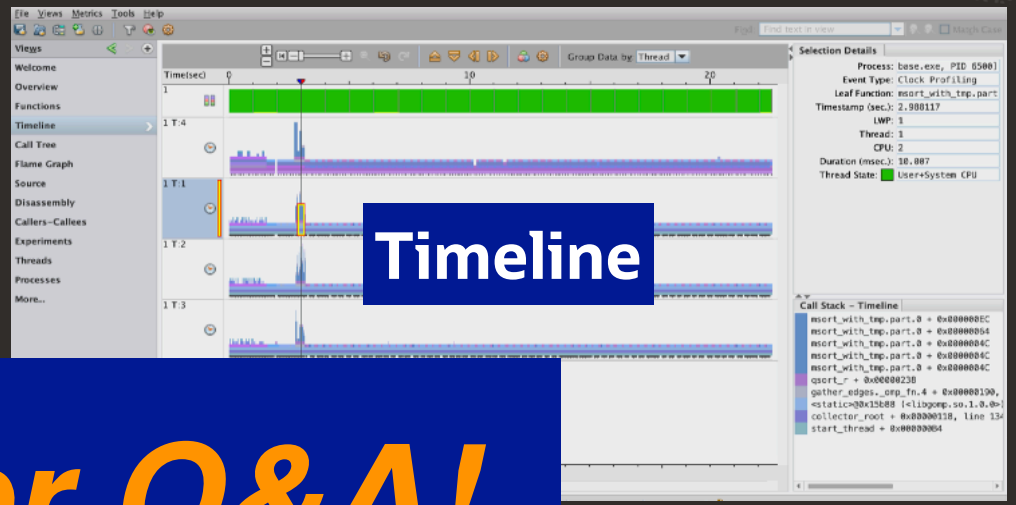


```
33. {
<Function: mxv_core>
0.  0.  0  0  0.  0.  [33] 4021ae: mov 0x8(%rsp),%r10
34.   for (uint64_t i=row_index_start; i<=row_inde
0.  0.  0  0  0.  0.  [34] 4021b3: cmp %rsi,%rdi
0.  0.  0  0  0.  0.  [34] 4021b6: jbe 0x37 [ 4021ed ]
0.  0.  0  0  0.  0.  [34] 4021b8: ret
                      = 0.0;
                     0; j<n; j++)
                     A[i][j] * b[j];
0.  0.  0  0  0.  0.  [37] 4021b9: mov (%r8,%rdi,8),%rdx
0.  0.  0  0  0.  0.  [36] 4021bd: mov $0x0,%eax
0.  0.  0  0  0.  0.  [35] 4021c2: pxor %xmm1,%xmm1
0.  0.  0  100103  0.  0.  [37] 4021c6: movsd (%rdx,%rax,8),%xmm0
## 0.470  0.726  640200389  3403448  0.385  2.600  [37] 4021cb: mulsd (%r9,%rax,8),%xmm0
## 0.781  0.963  2560801669  9209277  1.159  0.862  [37] 4021d1: addsd %xmm0,%xmm1
## 2.041  2.163  11299536923  11111217  2.277  0.439  [36] 4021d5: add $0x1,%rax
0.050  0.014  0  0  0.  0.  [36] 4021d9: cmp %rax,%rcx
0.  0.  0  0  0.  0.  [36] 4021dc: jne 0xfffffffffffffea [ 4021c6 ]
                     38.       c[i] = row_sum;
0.  0.  0  0  0.  0.  [38] 4021de: movsd %xmm1,(%r10,%rdi,8)
0.  0.  0  0  0.  0.  [34] 4021e4: add $0x1,%rdi
```

Disassembly View

The gprofng GUI - A Sneak Preview





Time for Q&A!