Python 对 Linux perf 性能分析器的支持

发行版本 *3.12.3*

Guido van Rossum and the Python development team

五月 16, 2024

Python Software Foundation
Email: docs@python.org

Contents

作者
Pablo Galindo

Linux perf 性能分析器 是一个非常强大的工具，它允许你分析并获取有关你的应用程序运行性能的信息。perf 还拥有一个非常活跃的工具生态系统可以帮助分析它所产生的数据。

将 perf 性能分析器与 Python 应用程序配合使用的主要问题在于 perf 只能获取原生符号的信息，即以 C 编写的函数和过程的名称。这意味着在你的代码中的 Python 函数名称和文件名称将不会出现在 perf 输出中。

从 Python 3.12 开始，解释器可以运行于一个允许 perf 性能分析器的输出中显示 Python 函数的特殊模式下。当启用此模式时，解释器将在每个 Python 函数执行之前插入一小段即时编译的代码，它将使用 perf 映射文件来告知 perf 这段代码与相关联的 Python 函数之间的关系。

备注： 对 perf 性能分析器的支持目前仅在特定架构的 Linux 上可用。请检查 configure 构建步骤的输出或检查 python -m sysconfig | grep HAVE_PERF_TRAMPOLINE 的输出来确定你的系统是否受到支持。

例如，考虑以下脚本:

```
def foo(n):
    result = 0
    for _ in range(n):
        result += 1
    return result

def bar(n):
```

(续下页)

(接上页)

```
    foo(n)

def baz(n):
    bar(n)

if __name__ == "__main__":
    baz(1000000)
```

我们可以运行 perf 以 9999 赫兹的频率来对 CPU 栈追踪信息进行采样:

```
$ perf record -F 9999 -g -o perf.data python my_script.py
```

然后我们可以使用 perf report 来分析数据:

```
$ perf report --stdio -n -g

# Children      Self       Samples  Command     Shared Object       Symbol
# ........  ........  ............  ..........  ..................  ..................
↪.......................
#
    91.08%     0.00%             0  python.exe  python.exe          [.] _start
            |
            ---_start
            |
                --90.71%--__libc_start_main
                        Py_BytesMain
                        |
                        |--56.88%--pymain_run_python.constprop.0
                        |          |
                        |          |--56.13%--_PyRun_AnyFileObject
                        |          |          _PyRun_SimpleFileObject
                        |          |          |
                        |          |          |--55.02%--run_mod
                        |          |          |          |
                        |          |          |           --54.65%--PyEval_EvalCode
                        |          |          |                     _PyEval_
↪EvalFrameDefault
                        |          |          |                     PyObject_
↪Vectorcall
                        |          |          |                     _PyEval_Vector
                        |          |          |                     _PyEval_
↪EvalFrameDefault
                        |          |          |                     PyObject_
↪Vectorcall
                        |          |          |                     _PyEval_Vector
                        |          |          |                     _PyEval_
↪EvalFrameDefault
                        |          |          |                     PyObject_
↪Vectorcall
                        |          |          |                     _PyEval_Vector
                        |          |          |                     |
                        |          |          |                     |--51.67%--_
↪PyEval_EvalFrameDefault
                        |          |          |                     |          |
                        |          |          |                     |          |--11.
↪52%--_PyLong_Add
                        |          |          |                     |          |
```

(续下页)

(接上页)

```
↪    |
                        |          |          |                    |          |    ␣
↪    |--2.97%--_PyObject_Malloc
...
```

如你所见，Python 函数不会显示在输出中，只有 `_PyEval_EvalFrameDefault` (评估 Python 字节码的函数) 会显示出来。不幸的是那没有什么用处因为所有 Python 函数都使用相同的 C 函数来评估字节码所以我们无法知道哪个 Python 函数与哪个字节码评估函数相对应。

相反，如果我们在启用 `perf` 支持的情况下运行相同的实验代码我们将获得:

```
$ perf report --stdio -n -g

# Children      Self       Samples  Command     Shared Object       Symbol
# ........  ........  ............  ..........  ..................  ...................
↪.............................................
#
    90.58%     0.36%             1  python.exe  python.exe          [.] _start
            |
            ---_start
            |
                --89.86%--__libc_start_main
                        Py_BytesMain
                        |
                        |--55.43%--pymain_run_python.constprop.0
                        |           |
                        |           |--54.71%--_PyRun_AnyFileObject
                        |           |           _PyRun_SimpleFileObject
                        |           |           |
                        |           |           |--53.62%--run_mod
                        |           |           |           |
                        |           |           |            --53.26%--PyEval_EvalCode
                        |           |           |                      py::<module>:/src/
↪script.py
                        |           |           |                      _PyEval_
↪EvalFrameDefault
                        |           |           |                      PyObject_
↪Vectorcall
                        |           |           |                      _PyEval_Vector
                        |           |           |                      py::baz:/src/
↪script.py
                        |           |           |                      _PyEval_
↪EvalFrameDefault
                        |           |           |                      PyObject_
↪Vectorcall
                        |           |           |                      _PyEval_Vector
                        |           |           |                      py::bar:/src/
↪script.py
                        |           |           |                      _PyEval_
↪EvalFrameDefault
                        |           |           |                      PyObject_
↪Vectorcall
                        |           |           |                      _PyEval_Vector
                        |           |           |                      py::foo:/src/
↪script.py
                        |           |           |                      |
                        |           |           |                      |--51.81%--_
```

(续下页)

(接上页)

```
↪PyEval_EvalFrameDefault
                        |           |           |                       |           |
                        |           |           |                       |           |--13.
↪77%--_PyLong_Add
                        |           |           |                       |           |     ␣
↪    |
                        |           |           |                       |           |     ␣
↪    |--3.26%--_PyObject_Malloc
```

1 如何启用 perf 性能分析支持

要启动 perf 性能分析支持可以通过使用环境变量 PYTHONPERFSUPPORT 或 -X perf 选项，或者动态地使用 sys.activate_stack_trampoline() 和 sys.deactivate_stack_trampoline() 来运行。

sys 函数的优先级高于 -X 选项，-X 选项的优先级高于环境变量。

示例，使用环境变量:

```
$ PYTHONPERFSUPPORT=1 python script.py
$ perf report -g -i perf.data
```

示例，使用 -X 选项:

```
$ python -X perf script.py
$ perf report -g -i perf.data
```

示例，在文件 example.py 中使用 sys API:

```
import sys

sys.activate_stack_trampoline("perf")
do_profiled_stuff()
sys.deactivate_stack_trampoline()

non_profiled_stuff()
```

... 然后:

```
$ python ./example.py
$ perf report -g -i perf.data
```

2 如何获取最佳结果

要获取最佳结果，Python 应当使用 CFLAGS="-fno-omit-frame-pointer -mno-omit-leaf-frame-pointer" 来编译因为这将允许性能分析器仅使用帧指针而不是基于 DWARF 调试信息进行展开。这是因为被插入以允许 perf 支持的代码是动态生成的所以它没有任何 DWARF 调试信息可用。

你可以通过运行以下代码来检查你的系统是否为附带此旗标来编译的:

```
$ python -m sysconfig | grep 'no-omit-frame-pointer'
```

如果你没有看到任何输出则意味着你的解释器没有附带帧指针来编译因而它将无法在 `perf` 的输出中显示 Python 函数。

索引

非字母

P