Python 對 Linux perf 分析器的支援

發🄵 *3.12.3*

Guido van Rossum and the Python development team

5 月 02, 2024

Python Software Foundation
Email: docs@python.org

Contents

作者
Pablo Galindo

Linux 性能分析器 (Linux perf profiler) 是一個非常🄵大的工具，可讓你分析🄵獲取有關應用程式的性能資訊。perf 還擁有一個非常活躍的工具生態系統，有助於分析其生成的資料。

在 Python 應用程式中使用 perf 分析器的主要問題是 perf 僅獲取有關原生符號的資訊，即用 C 編寫的函式和程式的名稱。這表示程式碼中的 Python 函式名稱和檔案名稱不會出現在 perf 的輸出中。

從 Python 3.12 開始，直譯器可以在特殊模式下執行，該模式允許 Python 函式出現在 perf 分析器的輸出中。🄵用此模式後，直譯器將在執行每個 Python 函式之前插入 (interpose) 一小段動態編譯的程式碼，🄵使用 perf map 檔案來告訴 perf 這段程式碼與相關聯的 Python 函式間的關🄵。

備🄵： 目前對 perf 分析器的支援僅適用於 Linux 的特定架構上。檢查 configure 建構步驟的輸出或檢查 python -m sysconfig | grep HAVE_PERF_TRAMPOLINE 的輸出來查看你的系統是否支援。

例如，參考以下🄵本：

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

def bar(n):
    foo(n)
```

(繼續下一頁)

(繼續上一頁)

```
def baz(n):
    bar(n)

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

我們可以執行 `perf` 以 9999 赫🄵🄵樣 CPU 堆🄵追🄵 (stack trace)：

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

然後我們可以使用 `perf report` 來分析資料：

```
$ 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
                        |          |          |                     |          |     ␣
↪     |
```

(繼續下一頁)

(繼續上一頁)

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

如你所見，Python 函式未顯示在輸出中，僅顯示 `_Py_Eval_EvalFrameDefault`（🄵 Python 位元組碼 (bytecode) 求值的函式）。不幸的是，這不是很有用，因🄵所有 Python 函式都使用相同的 C 函式來替位元組碼求值，因此我們無法知道哪個 Python 函式是對應於哪個位元組碼計算函式。

作🄵替代，如果我們在🄵用 `perf` 支援的情🄵下執行相同的實驗，我們會得到：

```
$ 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%--_
↪PyEval_EvalFrameDefault
```

(繼續下一頁)

(繼續上一頁)

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

1 如何🄵用 perf 分析支援

要🄵用 perf 分析支援，可以在一開始就使用環境變數 PYTHONPERFSUPPORT 或使用 -X perf 選項，也可以使用 sys.activate_stack_trampoline() 和 sys.deactivate_stack_trampoline() 來動態🄵用。

sys 函式優先於 -X 選項、-X 選項優先於環境變數。

例如，使用環境變數：

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

例如，使用 -X 選項：

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

例如，在 example.py 檔案中使用 sys API：

```
import sys

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

non_profiled_stuff()
```

... 然後：

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

2 如何獲得最佳結果

🄵了獲得最佳結果，應使用 CFLAGS="-fno-omit-frame-pointer -mno-omit-leaf-frame-pointer" 來進行 Python 編譯，因🄵這能允許分析器僅使用 frame 指標而不是 DWARF 除錯資訊來解析 (unwind)。這是因🄵，由於插入以允許 perf 支援的程式碼是動態生成的，因此它🄵有任何可用的 DWARF 除錯資訊。

你可以透過執行以下指令來檢查你的系統是否已使用此旗標進行編譯：

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

如果你F有看到任何輸出，則表示你的直譯器尚未使用 frame 指標進行編譯，因此它可能無法在 `perf` 的輸出中顯示 Python 函式。

索引

非依字母順序

P