1-global

June 3, 2024

1 Measuring global time and memory, with the operating system commands

Whatever we want to monitor, we can start with the tools provided by the operating system, and get a global view.

1.1 Bash `time` command

- In the Linux world, information about the overall execution time of an application can be obtained by simply preceding the application name with the `time` command.
- Some shells such as `bash` have some built-in `time` command.

[19]:

```
%%file tmp.fibo.cpp

#include <iostream>

constexpr int fibonacci( int n ) {
  if (n>1) return fibonacci(n-1) + fibonacci(n-2) ;
  else return n ;
}

int main() {
  constexpr int res { fibonacci(36) } ;
  std::cout<<res<<std::endl ;
  return 0 ;
}
```

```
Overwriting tmp.fibo.cpp
```

[2]:

```
!rm -f tmp.fibo.exe
```

For a demonstration in thsi notebook, I prepare below a script, which I will run later with `bash -l`. This is a way to ensure we use the bash built-in time.

[3]:

```
%%file tmp.time.bash

echo $*
time $*
```

```
Overwriting tmp.time.bash
```

[4]:

```
!bash -l ./tmp.time.bash g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
```

```
g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe

real	0m2.921s
user	0m2.893s
sys	0m0.025s
```

Details of the **time** display: - **real** : the elapsed time seen in real life. - **user** : the cpu time spent in the user code. - **sys** : the cpu time spent in system calls.

[5]:

```
!bash -l ./tmp.time.bash ./tmp.fibo.exe
```

```
./tmp.fibo.exe
14930352

real	0m0.001s
user	0m0.001s
sys	0m0.000s
```

1.2 GNU time command

- One can also use GNU time, if installed, which has useful options for formatting.
- If you want to use the GNU flavor in a **bash**, you should backslash your call so to avoid the built-in command.
- GNU time can also monitor the memory you use.

[6]:

```
%%file tmp.time.bash

echo $*
\time -f "%U s, %M kBytes." $*
```

```
Overwriting tmp.time.bash
```

[7]:

```
!bash -l ./tmp.time.bash g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
```

```
g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
2.95 s, 64108 kBytes.
```

[8]:

```
!bash -l ./tmp.time.bash ./tmp.fibo.exe
```

```
./tmp.fibo.exe
14930352
0.00 s, 3500 kBytes.
```

[25]:

```
%%file tmp.fibo.cpp

#include <iostream>

int fibonacci( int n ) {
  if (n>1) return fibonacci(n-1) + fibonacci(n-2) ;
  else return n ;
}

int main() {
  int res { fibonacci(36) } ;
  std::cout<<res<<std::endl ;
  return 0 ;
}
```

```
Overwriting tmp.fibo.cpp
```

[26]:

```
!bash -l ./tmp.time.bash g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
```

```
g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
0.23 s, 64916 kBytes.
```

[27]:

```
!bash -l ./tmp.time.bash ./tmp.fibo.exe
```

```
./tmp.fibo.exe
14930352
0.09 s, 3452 kBytes.
```

1.3 Repeat

When monitoring a command execution, especially a fast one, and especially when running on a non-reserved dedicated machine : - **run your program several times** and compute the mean, - **ensure each single run is long enough** so that the processor pipelines get filled and you go well beyond the initial computing latency.

With the script below, we run the command once, so to check the result. Then we run it 10 times, measuring the time and memory with GNU time, and redirect the results into a python script, which will finally compute the means.

[28]:

```
%%file tmp.repeat.bash

echo $*
$*

rm -f tmp.repeat.py
echo "t = 0 ; m = 0" >> tmp.repeat.py
for i in 0 1 2 3 4 5 6 7 8 9
do \time -f "t += %U ; m += %M" -a -o ./tmp.repeat.py $* >> /dev/null
done
echo "print('(~ {:.3f} s)'.format(t/10.))" >> tmp.repeat.py
echo "print('(~ {:.0f} kBytes)'.format(m/10.))" >> tmp.repeat.py
python3 tmp.repeat.py
```

```
Overwriting tmp.repeat.bash
```

[29]:

```
!bash -l tmp.repeat.bash g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
```

```
g++ -std=c++17 tmp.fibo.cpp -o tmp.fibo.exe
(~ 0.240 s)
(~ 64906 kBytes)
```

```
[30]: !bash -l tmp.repeat.bash ./tmp.fibo.exe
```

```
./tmp.fibo.exe
14930352
(~ 0.089 s)
(~ 3508 kBytes)
```

If installed, you can also try hyperfine:

```
[31]: !hyperfine --warmup 3 "./tmp.fibo.exe"
```

```
Benchmark 1: ./tmp.fibo.exe              ETA
00:00:00
  Time (mean ± ):      94.8 ms ±   1.0

ms    [User: 94.3 ms, System: 0.5 ms]
  Range (min … max):    93.2 ms …  98.3 ms
31 runs
```

2 Questions ?

3 Resources

- hyperfine.