2-compilation

June 3, 2024

1 Profiling compilation

Because the C++ compiler wants to do more at compile time, and less at run time, the compilation can be long and requires much memory. It makes sense to profile it.

1.1 Globally, for a single-file code

If you want to measure the compilation time/memory of a given code, and this code fits in a single file, you may get it with GNU time or hyperfine, as described in the previous notebook.

1.1.1 Exercice

With GNU time or hyperfine, compare the compilation time of those three implementations of fibonacci:

[1]:
```
%%file tmp.fibo1.cpp

#include <iostream>

template<int N>
struct fibonacci {
  enum { value = fibonacci<N-1>::value + fibonacci<N-2>::value } ;
} ;

template<>
struct fibonacci<1> {
  enum { value = 1 } ;
} ;

template<>
struct fibonacci<0> {
  enum { value = 0 } ;
} ;

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

```
Writing tmp.fibo1.cpp
```

[10]:
```
!rm -f tmp.fibo1.exe
```

[11]:
```
!g++ -O2 -std=c++17 tmp.fibo1.cpp -o tmp.fibo1.exe
```

[12]:
```
!./tmp.fibo1.exe
```

```
14930352
```

[4]:
```
%%file tmp.fibo2.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 ;
}
```

```
Writing tmp.fibo2.cpp
```

[5]:
```
!rm -f tmp.fibo2.exe
```

[5]:
```
!g++ -O2 -std=c++17 tmp.fibo2.cpp -o tmp.fibo2.exe
```

[6]:
```
!./tmp.fibo2.exe
```

```
14930352
```

[7]:
```
%%file tmp.fibo3.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 ;
}
```

```
Writing tmp.fibo3.cpp
```

[8]:
```
!rm -f tmp.fibo3.exe && g++ -O2 -std=c++17 tmp.fibo3.cpp -o tmp.fibo3.exe
```

[9]:
```
!./tmp.fibo3.exe
```

```
14930352
```

Warning : the use of memory reported bu GNU time may be not so relevant for the compilation, because the g++ frontend process may spawn subprocesses which are not taken into account.

1.2 Comparing alternative implementations

If you do not care about the absolute compilation time, but want to compare two (or more) alternative small implementations, you can try BuildBench.

1.2.1 Exercice

With BuildBench, compare those implementations, for compilation time and compilation memory. Are the results consistent when you play with different options ? (compiler, C++ version, optimization level, libc++).

Beware: depending on the load of this web service when you use it, when requiring high values such as 36 for the program, the compilation of the `constexpr` implementation may be too long and trigger a time out.

```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 ;
}
```

```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 ;
}
```

1.3 Within a file

The compilers have dedicated options, which helps you to know what goes inside: - `g++ -ftime-report` let you known the time spent in different phases like preprocessing, compilation, assembly, and linking, - `clang++ -ftime-trace` goes further and produces a json flamegraph.

[32]:
```
!rm -f tmp.fibo2.exe
```

[33]:
```
!g++ -ftime-report -std=c++17 tmp.fibo2.cpp -o tmp.fibo2.exe
```

```
Time variable                                   usr           sys          wall
GGC
 phase setup                    :   0.00 (  0%)   0.00 (  0%)   0.01 (  3%)
1752k (  4%)
 phase parsing                  :   0.23 ( 88%)   0.10 ( 91%)   0.34 ( 89%)
37M ( 88%)
 phase lang. deferred           :   0.03 ( 12%)   0.01 (  9%)   0.03 (  8%)
3283k (  7%)
 |name lookup                   :   0.03 ( 12%)   0.01 (  9%)   0.09 ( 24%)
1707k (  4%)
 |overload resolution           :   0.03 ( 12%)   0.00 (  0%)   0.02 (  5%)
2048k (  5%)
 preprocessing                  :   0.04 ( 15%)   0.01 (  9%)   0.06 ( 16%)
1670k (  4%)
 parser (global)                :   0.02 (  8%)   0.05 ( 45%)   0.10 ( 26%)
13M ( 31%)
 parser struct body             :   0.05 ( 19%)   0.02 ( 18%)   0.06 ( 16%)
8780k ( 20%)
 parser enumerator list         :   0.01 (  4%)   0.00 (  0%)   0.00 (  0%)
108k (  0%)
 parser function body           :   0.01 (  4%)   0.00 (  0%)   0.02 (  5%)
2296k (  5%)
 parser inl. func. body         :   0.03 ( 12%)   0.00 (  0%)   0.02 (  5%)
1512k (  3%)
 parser inl. meth. body         :   0.01 (  4%)   0.00 (  0%)   0.02 (  5%)
3270k (  7%)
 template instantiation         :   0.08 ( 31%)   0.03 ( 27%)   0.07 ( 18%)
10M ( 24%)
 constant expression evaluation :   0.01 (  4%)   0.00 (  0%)   0.02 (  5%)
62k (  0%)
 TOTAL                          :   0.26          0.11          0.38
42M
```

[34]:
```
!clang++ -ftime-trace -std=c++17 tmp.fibo2.cpp -o tmp.fibo2.exe
```

```
Time trace json-file dumped to /tmp/tmp-3f185d.json
Use chrome://tracing or Speedscope App (https://www.speedscope.app) for
flamegraph visualization
```

If this seems a little obscure, Crofiler may help.

1.4 When there are many files

As soon as the program is splitted in several modules, if you do not want to handle yourself many numbers, you will need the help of your build system.

2 Questions ?

3 Resources

- BuildBench.
- Crofiler