522031910747+李若彬+hw0

实验复现

实验环境

test.py

```python
def f(n):
    if n == 1:
        return 1
    else:
        #print n
        return n * f(n-1)

print (f(100))
```

```
cdm@cdm-virtual-machine:~/Desktop/ADS$ time python test.py
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

real    0m0.038s
user    0m0.026s
sys     0m0.009s
```

test.cpp

```cpp
#include <iostream>
#include <gmpxx.h>
using namespace std;

mpz_class f(int n){
    if(n==1)
        return 1;
    else
        return n*f(n-1);
}

int main(){
    cout<<f(100)<<endl;
}
```

```
cdm@cdm-virtual-machine:~/Desktop/ADS$ time ./test
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

real    0m0.005s
user    0m0.000s
sys     0m0.005s
```

Q&A

Q1: 为什么使用c++写的程序会比使用python写的程序快?

> 因为 C++ 是一种编译型语言(low-level)，而 Python 是一种解释型语言(high-level)。在 C++ 中，由于是静态类型的，其变量的类型在编译时就是已知的，程序在编译阶段就会被直接翻译成机器代码还存在预编译，这样运行速度就比较快。而 Python 是动态类型语言，需要在运行时会调用解释器解释代码为可执行代码，这会导致程序运行速度相对较慢。

Q2: 去掉"test.cpp"中的#inlcude<gmpxx.h>，并将mpz_class改成int后，计算100的阶层的运行结果如何？为什么会出现这样的结果?

```
cdm@cdm-virtual-machine:~/Desktop/ADS$ time ./test
0

real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

> 由上图可知计算结果溢出，程序输出0，其时间都源于user（程序在用户态执行代码所花费的 CPU 时间。即程序运行过程中真正被程序使用的 CPU 时间）为0.002s少于调用GMP 库时的全部时间——sys（程序在内核态执行代码所花费的 CPU 时间。即程序运行过程中操作系统内核为程序提供服务所花费的 CPU 时间）。

溢出原因

原本的代码中使用了 GMP（GNU Multiple Precision Arithmetic Library）库中的 `mpz_class` 类型来处理大整数运算。`mpz_class` 类型可以表示任意精度的整数，因此能够准确地计算非常大的数值。

当将 `#include <gmpxx.h>` 和 `mpz_class` 类型都去掉，改用普通的 `int` 类型时，整数溢出问题就出现了。在计算 100 的阶乘时，结果远远超过了 `int` 类型所能表示的范围，导致溢出，最终结果为 0。

这是因为 `int` 类型通常只能表示有限范围的整数值（**2147483647**），而计算 100 的阶乘得到的结果远远超出了 `int` 类型的表示范围。因此，要正确处理这么大的数值，需要使用类似 GMP 这样支持任意精度整数运算的库或者数据类型。通过使用 `mpz_class` 类型，可以避免整数溢出问题，得到正确的计算结果。

时间成本

时间上的变化可能是由于两个不同的因素导致的。

首先，使用 GMP 库进行任意精度整数计算需要更多的计算资源和操作，因为它需要处理大整数的运算和存储。相比之下，使用普通的 `int` 类型进行计算只需要较少的资源和操作。因此，当使用 GMP 库时，程序运行的实际时间可能会更长。

其次，编译器在编译代码时会对不同的数据类型进行优化。在原本的代码中，当使用 `mpz_class` 类型时，编译器会根据 GMP 库的特性进行优化，以提高计算效率。在第一次运行中，实际时间几乎全部花费在了系统 CPU 时间上（`sys` 时间），这可能是因为程序运行过程中大量的计算或操作需要在内核态执行，导致系统 CPU 时间占用较多。而当使用普通的 `int` 类型时，编译器可能会进行更简单的优化，因为它可以利用 CPU 的原生整数运算指令。这种优化可能导致使用普通的 `int` 类型时计算的速度更快。在第二次运行中，实际时间几乎全部花费在了用户 CPU 时间上（`user` 时间），这可能是因为程序运行过程中大部分计算都在用户态执行，没有需要调用内核的操作，因此系统 CPU 时间占用较少。

综合以上因素，当使用 GMP 库进行任意精度整数计算时，程序可能需要更多的计算资源和操作，并且可能没有得到编译器针对普通整数类型的优化，因此运行时间较长。而当使用普通的 `int` 类型进行计算时，程序可能能够更好地利用编译器的优化，从而运行时间更短。

Q3: 除了编程语言外，影响程序运行快慢还可能有哪些因素?

- 算法复杂度：程序中使用的算法的复杂度会直接影响运行时间。某些算法可能需要更多的计算和处理时间，而其他算法可能更高效。选择合适的算法可以显著提高程序的运行速度。
- 数据规模：处理大规模数据集时，程序的运行时间可能会延长。操作大型数据集可能需要更多的计算和内存资源，因此程序的运行速度可能会变慢。优化数据处理方法和使用适当的数据结构可以减少数据规模对程序性能的影响。
- 硬件性能：计算机的硬件性能也会对程序的运行速度产生影响。例如，CPU的速度、内存容量和磁盘读写速度等都可以对程序的性能产生影响。
- 网络延迟：涉及网络通信的程序可能受到网络延迟的影响。如果程序需要从远程服务器获取数据或与其他系统进行交互，网络延迟可能导致程序运行变慢。
- 外部资源访问：程序在访问外部资源（如数据库、网络服务等）时可能会受到外部资源的响应时间限制。如果外部资源的访问速度较慢，程序的整体运行时间可能会延长。
- 并发性：程序是否涉及并发操作也会影响其运行速度。如果程序需要处理多个任务或并行计算，合理地利用多核处理器或并发编程技术可以加快程序的执行速度。

> - 编译器/解释器的优化：不同的编译器和解释器可以对代码进行不同程度的优化，从而影响程序的运行速度。一些编译器和解释器可能会对代码进行诸如代码优化、即时编译等操作，以提高程序的执行效率。