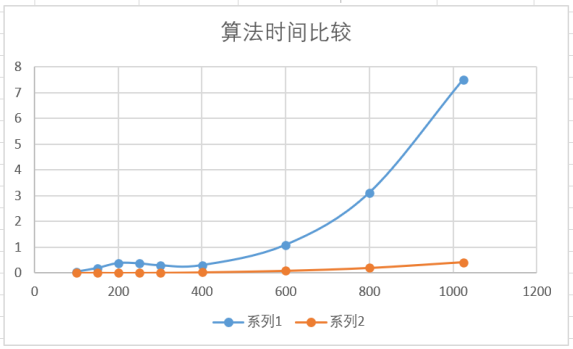
使用不同方法实现矩阵乘法：

矩阵规模	100	150	200	250	300	400	600	800	1024
naïve用时	0.043256	0.183104	0.377961	0.366873	0.293286	0.296668	1.084156	3.109983	7.505269
OpenBlas用时	0.003842	0.01185	0.004546	0.008141	0.01292	0.02853	0.086405	0.192298	0.40402



可以看出 openBlas 的算法明显优于 naïve 算法，

Naïve 算法耗时随输入规模增大增长很明显（在 200-400 区间内有小幅下降），

使用 02 优化后，naïve 算法的性能也能得到较大提升，1024*1024 大小的矩阵，耗时从 7.505269 减少到了 1.752625，可见编译器的优化是很有用的

gcc:

gcc 与 g++ 分别是 gnu 的 c & c++ 编译器 gcc/g++ 在执行编译工作的时候，总共需要 4 步：

- 1、预处理,生成 .i 的文件[预处理器 cpp]
- 2、将预处理后的文件转换成汇编语言，生成文件 .s [编译器 egcs]
- 3、有汇编变为目标代码(机器代码)生成 .o 的文件[汇编器 as]
- 4、连接目标代码，生成可执行程序 [链接器 ld]
-

gcc 的一些命令

-c

只激活预处理,编译,和汇编,也就是他只把程序做成 obj 文件

例子用法:

```
gcc -c hello.c
```

他将生成 .o 的 obj 文件

-S

只激活预处理和编译，就是指把文件编译成为汇编代码。

例子用法:

```
gcc -S hello.c
```

他将生成 .s 的汇编代码，你可以用文本编辑器察看。

-E

只激活预处理,这个不生成文件，你需要把它重定向到一个输出文件里面。

例子用法:

```
gcc -E hello.c > pianoapan.txt
gcc -E hello.c | more
```

慢慢看吧，一个 hello word 也要与处理成 800 行的代码。

-o

制定目标名称，默认的时候，gcc 编译出来的文件是 a.out，很难听，如果你和我有同感，

改掉它，哈哈。

例子用法:

```
gcc -o hello.exe hello.c (哦,windows 用习惯了)
    gcc -o hello.asm -S hello.c
```

-O0 、-O1 、-O2 、-O3

编译器的优化选项的 4 个级别，-O0 表示没有优化，-O1 为默认值，-O3 优化级别最高

GDB 是什么

GDB 全称“GNU symbolic debugger”，从名称上不难看出，它诞生于 GNU 计划（同时诞生的还有 GCC、Emacs 等），是 Linux 下常用的程序调试器。发展至今，GDB 已经迭代了诸多个版本，当下的 GDB 支持调试多种编程语言编写的程序，包括 C、C++、Go、Objective-C、OpenCL、Ada 等。实际场景中，GDB 更常用来调试 C 和 C++ 程序。

Windows 操作系统中，人们更习惯使用一些已经集成好的开发环境（IDE），如 VS、VC、Dev-C++ 等，它们的内部已经嵌套了相应的调试器。

图 1 GDB 的吉祥物：弓箭鱼

总的来说，借助 GDB 调试器可以实现以下几个功能：

1. 程序启动时，可以按照我们自定义的要求运行程序，例如设置参数和环境变量；
2. 可使被调试程序在指定代码处暂停运行，并查看当前程序的运行状态（例如当前变量的值，函数的执行结果等），即支持断点调试；
3. 程序执行过程中，可以改变某个变量的值，还可以改变代码的执行顺序，从而尝试修改程序中出现的逻辑错误

GDB 有以下常用方面的命令：

1. 运行 GDB
2. 运行命令
3. 设置断点
4. 设置观察点

(不常用,但疑难问题可能用得着)

5. 断点维护命令
6. 搜集任务信息
7. 其它命令