# 实验6、性能优化实验

注意！！如果发现评分波动较大，那是你的处理器在运行程序过程中自动降频所致。可以到BIOS设置中，关闭SpeedStep优化、低功耗优化、超线程优化等。

特别提醒：实验验收，是将你的poly.c和rowcol.c，复制到老爷机上，在老爷机上编译、运行。以最后在老爷机上运行的得分为准。**并不是在你自己的机器上的得分！**

本实验，有两个主要任务。任务一，优化一个多项式计算函数，达到最高性能，主要是考查算术运算优化能力；任务二，优化一个矩阵求和函数，达到最高性能，主要考查存储器访问优化能力。所有评分，以运行时间为衡量，运行越迅速，得分越高。

**任务一（多项式计算函数优化）：**

**原理提示：**乘法运算是“昂贵而缓慢”的运算，加法/减法要快得多，移位运算也非常快。循环的优化，请参考教材5.4、5.5、5.6、5.8、5.9。

1. 首先不要修改任何文件，然后运行程序，程序会提示你，你需要实现的常系数多项式计算函数具体是什么，据此可以修改poly.c的const\_poly\_eval函数。
2. 请仔细阅读poly.c，并对文件进行修改，书写自己的poly\_eval、const\_poly\_eval函数实现，力争达到最大性能。你需要至少书写三个函数实现，一个用于计算常系数多项式，一个用于任意的高阶多项式计算，另一个用于固定的10阶多项式计算。书写函数，可以使用汇编语言书写。

参考样例中的poly\_eval函数，实现了下面的计算：

1. 利用修改好的poly.c，生成可执行程序。程序会自动评分。
2. 生成可执行文件，运行可执行文件，程序将会给出自动评分。你的目标就是通过不断的优化const\_poly\_eval、poly\_eval等函数使得评分尽可能高！



**任务二（矩阵计算优化）：**

**原理提示：**存储器访问是“非常昂贵而缓慢”的操作。地址连续的存储器访问，要远远快于地址分散的多次存储器访问。存储访问的优化，请参考教材5.12、6.5、6.6。

1. 请仔细阅读rowcol.c，并对文件进行修改，书写自己的c\_sum、rc\_sum函数实现，力争达到最大性能。你需要至少书写两个函数实现，一个用于计算矩阵中的每一列的和（c\_sum），另一个用于计算矩阵中的每一行、每一列的和（rc\_sum）。书写函数，可以使用汇编语言书写。
2. 利用修改好的rowcol.c，生成可执行程序。程序会自动评分。
3. 生成可执行文件，运行可执行文件，程序将会给出自动评分。你的目标就是通过不断的优化c\_sum、rc\_sum等函数使得评分尽可能高！

