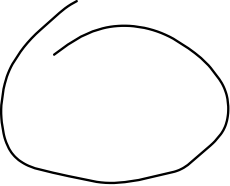
# 编程作业3.1：矩阵乘法的优化



**题目：**实现课件中四种矩阵乘法程序，并对比运行效率。

1. 串行算法
2. Catch优化
3. SSE/AVX版本
4. 分片策略

**要求：**提交实验报告（问题描述、算法设计（最好有复杂性分析）与实现、实验及结果分析）和源码（只将程序文件和工程文件提交，不要将编译出的目标文件和可执行文件也打包提交）。

一些细节问题：

1. 计时：
   1. 计时方法：参考课件示例代码中的计时方法，如QueryPerformance系列API；
   2. 计时精度：即使精度最高的计时机制，也可能测量不出太快的计算过程。如何解决？将计算过程重复多次，足以匹配计时精度，再计算平均时间。
2. 实验设计：
   1. 实验数据：矩阵大小可自定，也可测试不同矩阵规模，由小至大，随机生成矩阵元素值；
   2. 实验方法：为降低误差，每个测试应重复多次。
3. 报告撰写：
   1. 格式符合科技论文写作规范，参照第一次作业的格式。
   2. 有清晰的结果呈现，可以结合图表。

# 编程作业3.2：高斯消元法SSE/AVX并行化

**题目：**首先熟悉高斯消元法解线性方程组的过程（见**附录2**），然后实现SSE算法编程。过程中，请自行构造合适的线性方程组，并选取至少2个角度，讨论不同算法策略对性能的影响。

可选角度**包括但不限于**以下几种选项：

* 1. 相同算法对于不同问题规模的性能提升是否有影响，影响情况如何；
  2. 消元过程中采用向量编程的的性能提升情况如何；
  3. 回代过程可否向量化，有的话性能提升情况如何；
  4. 数据对齐与不对齐对计算性能有怎样的影响；
  5. SSE编程和AVX编程性能对比。

**要求：**

1. 提交实验报告（问题描述、算法设计（最好有复杂性分析）与实现、实验及结果分析）和源码（只将程序文件和工程文件提交，不要将编译出的目标文件和可执行文件也打包提交）。
2. 与编程作业3.1一起，写成一份报告。作业的命名格式，“姓名-学号-并行第2次作业”，如果是多个文件，需要打包为“.zip”格式并按如上方式命名。

**附录1：**

* 安装实验环境，建议windows系统，TDM-GCC+ Code::Blocks
  + TDM-GCC下载地址：<http://tdm-gcc.tdragon.net/download>
  + Code::Blocks下载地址： <http://www.codeblocks.org/downloads/>
  + 飞书备用下载：

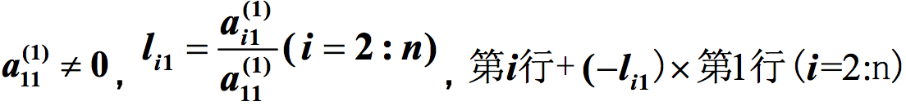
<https://nankai.feishu.cn/drive/folder/fldcn3iYQGMvoOkz67wok1jmODX>

* SSE函数集。
  + https://software.intel.com/sites/landingpage/IntrinsicsGuide/

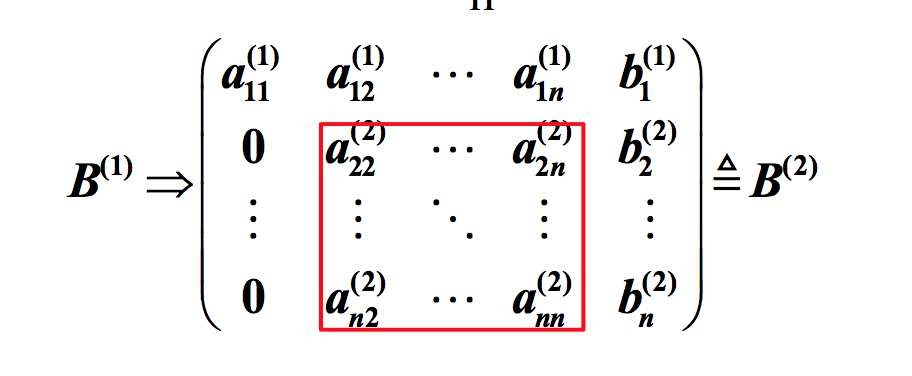
**附录2：高斯消元法的基本版本步骤示意：**

**（1）消元过程：**将Ax=b按照从上至下、从左至右的顺序化为上三角方程组，中间过程不对矩阵进行交换，主要步骤如下。

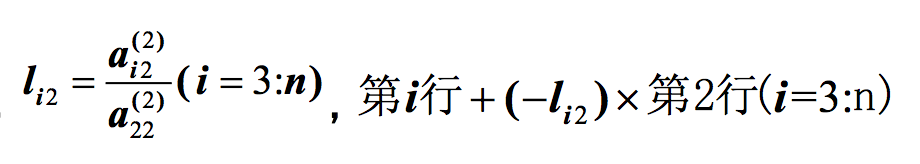
**Step1:** 将第2行至第n行，每行分别与第一行做运算，消掉每行第一个参数。公式如：



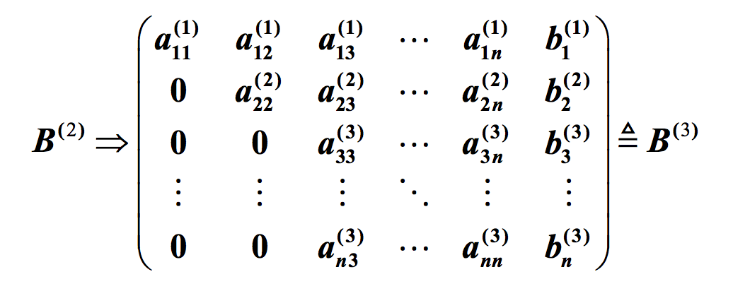
形成如下图所示新矩阵：



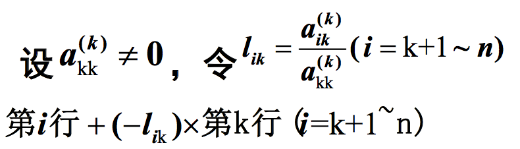
**Step2:** 从新矩阵的a22开始（a22不能为0），以第二行为基准，将第三行至第n行分别与第二行做运算，消掉每行第二个参数。公式如下：



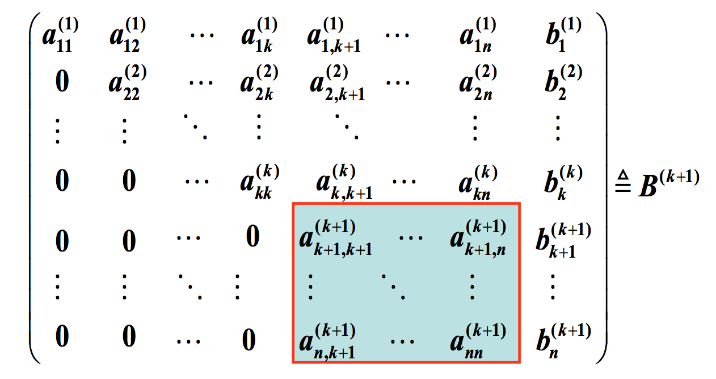
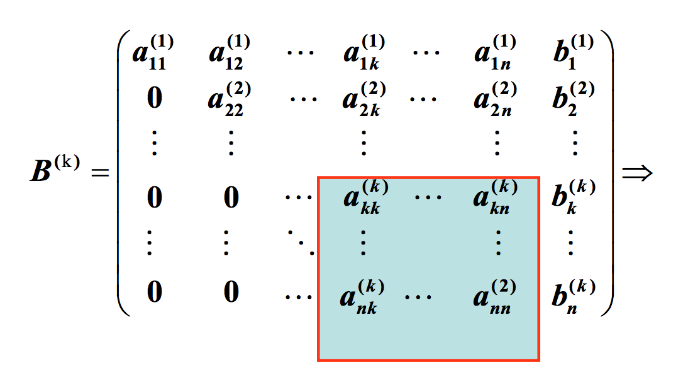
，形成如下图所示新矩阵：



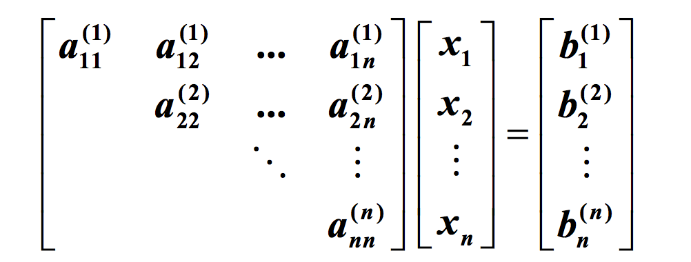
**Step K:** 按照上述方法，当第k步运算时，公式为：



运算前后的矩阵为：



**Step n-1:** 经过 n-1 步，方程组也就转化为了我们希望得到的上三角方程组，如下：



**消元过程的一种伪代码，供参考：**

1. **procedure** LU (*A*)

2. **begin**

3. **for** *k* := 1 **to** *n* **do**

4. **for** *j* := *k*+1**to** *n* **do**

5. *A*[*k, j*] := *A*[*k, j*]*/A*[*k, k*];

6 *A*[*k, k*] := 1.0;

7. **endfor**;

8. **for** *i* := *k* + 1 **to** *n* **do**

9. **for** *j* := *k* + 1 **to** *n* **do**

10. *A*[*i, j*] := *A*[*i, j*] - *A*[*i, k*] × *A*[*k, j* ];

11. **endfor**;

12. *A*[*i, k*] := 0;

13. **endfor**;

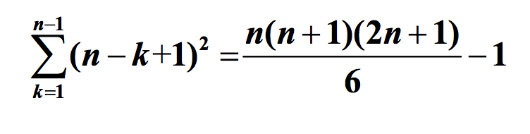
14. **endfor**;

15. **end** LU

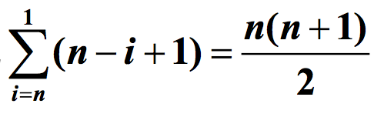
**(2)回代过程：**再从第n行开始，倒序回代前面的行中，即可求解 *x*1至 *x*n 的值。

***\*注：*顺序消去法计算量:**

消元过程：



回代过程：



总运算量：

