How Elegant Code Evolves with Hardware:

The Case of Gaussian Elimination

报告人：樊玉伟

Author： Jack Dongarra
Piotr Luszczek

作者简介

Jack Dongarra：

- 田纳西大学计算机科学系的计算机科学杰出教授
- 美国橡树岭国家实验室（ORNL）杰出研究人员
- 曼彻斯特大学计算机科学与数学学院的Turing Fellow
- 美国莱斯大学计算机科学系的副教授
- AAAS，ACM和IEEE的成员
- 美国工程院的院士
- 研究领域：线性代数中的数值算法，并行计算，高级计算机架构的应用，程序设计方法学，用于并行计算机的工具
- 在ISPACK, LINPACK, the BLAS, LAPACK, ScaLAPACK, Netlib, PVM, MPI, NetSolve, Top500, ATLAS, 和 PAPI上做出过贡献

Piotr Luszczek：

- 田纳西大学诺克斯维尔分校 研究教授
- 研究领域：并行的核外（out-of-core）库
  大型超级计算机安装的标准化评价
- 开发了一个自适应的软件库，能够自动选择最优的算法来有效地利用现有硬件以及有选择地处理输入数据
- 对高性能编程语言的设计和实现感兴趣

- 数学上等价的公式在计算上不等价

- “计算上等价”的算法在实现上不等价

真正的适应性 I

- 应用程序层面上

- 软件层面上

性能

可移植性

代码可理解性

真正的适应性 II

真正的适应性应试算法具有很好的可移植性

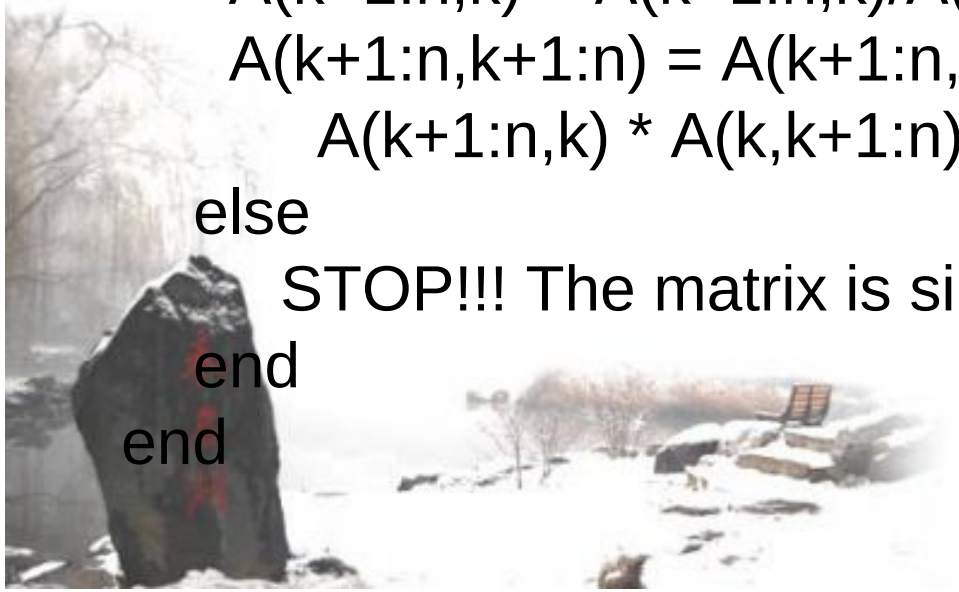
可移植性 = 可迁移性 + 模块化 + 高效的软件标准

预备知识

- 稀疏矩阵：绝大多数元素为0的矩阵
- 稠密矩阵
- Gauss消元法
- 计算机体系结构

Gauss消元法算法

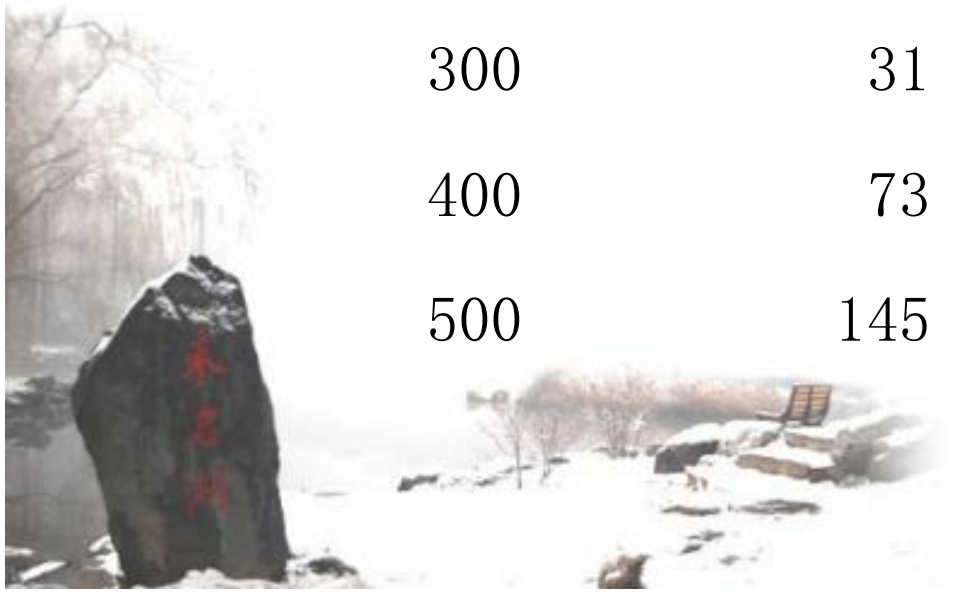
```
for k = 1:n-1
    find p (k <= p <= n) s.t.
          |A(p,k)| = max{|A(i,k)|, i = k:n }
    A(k, 1:n) <--> A(p, 1:n)
    u(k) = p
    if A(k,k) =/= 0
      A(k+1:n,k) = A(k+1:n,k)/A(k,k)
      A(k+1:n,k+1:n) = A(k+1:n,k+1:n) -
           A(k+1:n,k) * A(k,k+1:n)
    else
        STOP!!! The matrix is singular!
    end
end
```



程序实现与运行结果

- MATLAB内部实现
- MATLAB代码实现
- C++实现

n	MATLAB内部	MATLAB代码	C++代码
300	31	435	63
400	73	1420	143
500	145	3052	234



原因分析

- MATLAB是按列存储数据的→修改内层循环顺序，可节省30%时间

- 数据传输占用过多时间

BLAS-1

- 70年代中期，向量架构用于科学计算
- 70年代末，向量计算机出现
- 向量运算标准化→定义一套简单、频繁调用的操作，并在多种系统上实现

- 向量-向量运算

LINPACK

- 基于BLAS-1
- 由Fortran语言实现
- 采用面向列算法→可保持引用局部性
- 提供了基本代数运算的子程序
- 使得程序可以独立于计算机

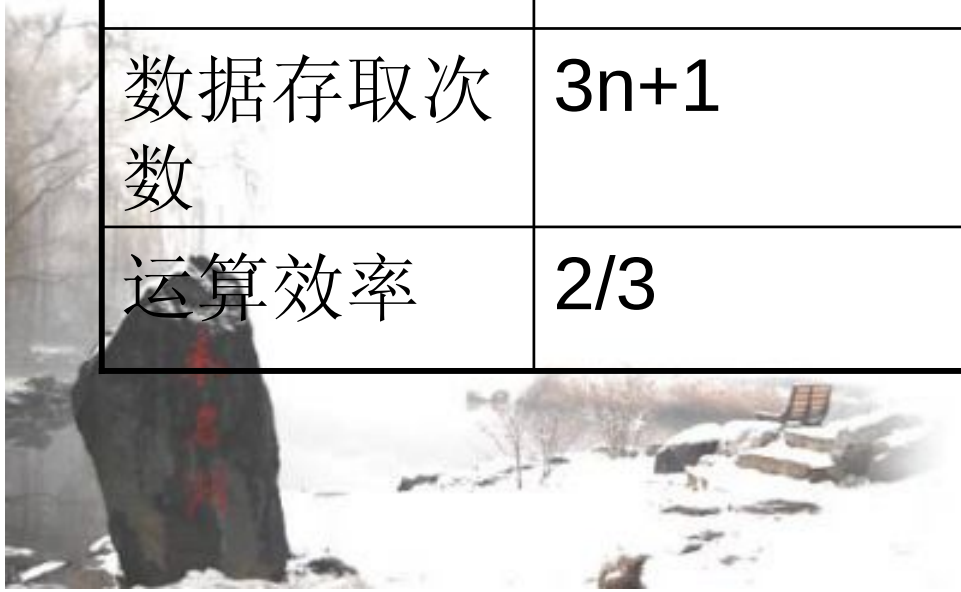
- dgefa程序

- 不足：向量机存储器与寄存器间带宽较窄
  在移植到RISC计算机上时未考虑多层存储结构

- 解决方法：减少数据移动

BLAS对比

	BLAS-1	BLAS-2	BLAS-3
特点	向量-向量	矩阵-向量	矩阵-矩阵
代表操作	$\vec{y} \leftarrow \alpha \vec{x} + \vec{y}$	$\vec{y} \leftarrow A\vec{x} + \vec{y}$	$C \leftarrow AB + C$
运算次数	2n	2n^2	2n^3
数据存取次数	3n+1	n^2+3n	4n^2
运算效率	2/3	2	n/2



LAPACK

- 为适应RISC缓存系统和并行计算
- 基于BLAS-3
- 可用于并行处理器
- 内层循环使用分块矩阵运算

可迁移性 +

可移植性 = 模块化 +

高效的软件标准

LAPACK的可移植性基础：每种机器上实现高度优化的分块矩阵运算

LAPACK 中LU分解

- 算法思路
- dgetrf程序

LAPACK 中LU分解 II

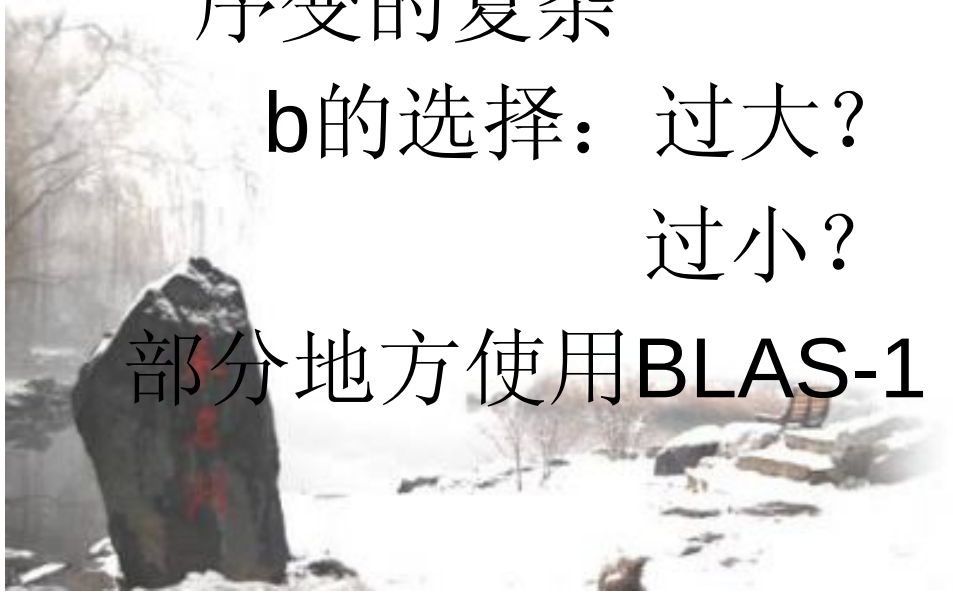
优势：更大程度的数据重用，降低内存读写频率

不足：向量表示更加自然，矩阵表示使得程序变的复杂

b的选择：过大？

过小？

部分地方使用BLAS-1



LINPACK V.S LAPACK

	1 processor	8 processor
Maximum speed	330 Mflops	2640 Mflops
LINPACK(Cholesky,n=500)	72 Mflops	72 Mflops
LAPACK(Cholesky,n=500)	290 Mflops	1414 Mflops
LAPACK(Cholesky,n=1000)	301 Mflops	2115 Mflops
该试验在巨型机Cray YMP上进行的。Mflops为10^6次浮点运算		

递归LU

- 算法思路
- 1.如果矩阵为一列，则按支点缩放返回
- 2.将矩阵A(m*n阶)按列等分成两个长方阵
- 3.对左侧阵进行LU
- 4.利用左侧分解对右侧阵变换
- 5.对右侧阵进行LU

- rdgetrf程序

- 优点：不用考虑b的选取
  BLAS-1的调用被降到最少
  程序的层次结构变得清晰

- 不足：使用递归

PBLAS引论

- 类似于BLAS，提供了一个分布式内存标准
- 用于并行软件开发

并行与多核系统的多线程设计

- Amdahl's Law: the observation that the time taken by the sequential portion of a computation provides the minimum bound for the entire execution time, and therefore limits the gains achievable from parallel processing.

各种LU对比

- 所有算法的稳定性都满足：

$$\|r\|/\|A\| \le \|e\| \le \|A^{-1}\| * \|r\|$$

- 其中 $e = x - y$ ，$r = Ay - b$

- x为Ax=b精确解，y为数值解

- 所有算法复杂度都是 2/3 n^3+O(n^2)

各种LU对比 II

- 运行速度
- MATLAB实现 << LINPACK Fortran实现 < LAPACK Fortran 实现 < 递归LU实现

- 计算上等价的方法实现上未必等价