



矩阵向量乘积 并行算法 (OpenMP)

目录页

Contents

1

并行计算预备知识

2

矩阵向量并行乘积

1

预备知识

- 加速比与并行效率
- 性能优化
- 一些记号和设定

并行算法基础知识

- 加速比

$$S_p(q) = \frac{T_s}{T_p(q)}$$

其中 T_s 串程序执行时间, $T_p(q)$ 为 q 个进程 (线程) 的执行时间

- 并行效率

$$E_p(q) = \frac{S_p(q)}{q}$$

程序性能优化

■ 串行程序性能优化 — 并行程序性能优化的基础

- 调用高性能库。如：BLAS、LAPACK、FFTW
- 选择编译器优化选项：-O2、-O3
- 合理定义数组维数
- 注意嵌套循环次数：数据访问的空间局部性和时间局部性
- 循环展开
- 数据分块

程序性能优化

■ 并行程序性能优化

- 负载平衡，减少进程（线程）的空闲时间
- 减少数据竞争
- 设计好的并行算法和通信模式
- 减少通信次数、提高通信粒度
- 多进程通信时尽量使用高效率的聚合通信算法
- 通信与计算的重叠
- 通过引入重复计算来减少通信

2

矩阵向量并行乘积

- 串行算法
- 自动并行
- 手动并行

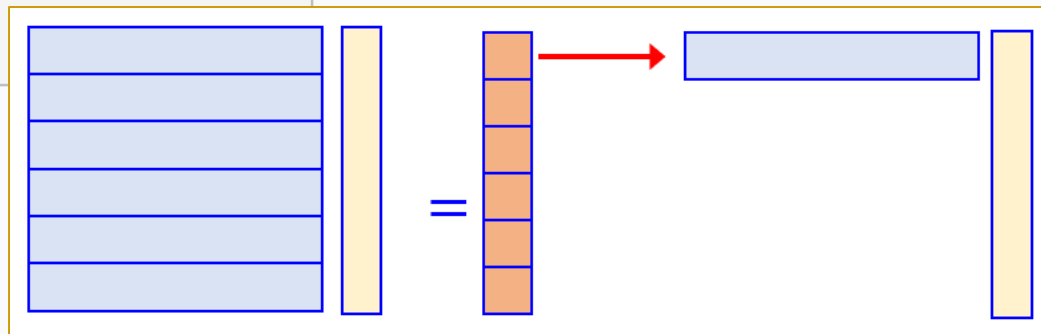
矩阵向量乘积

$$\boxed{y = Ax} \quad A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^n$$

■ 串行算法

- 实现方法一：i-j 循环 —— 行方式

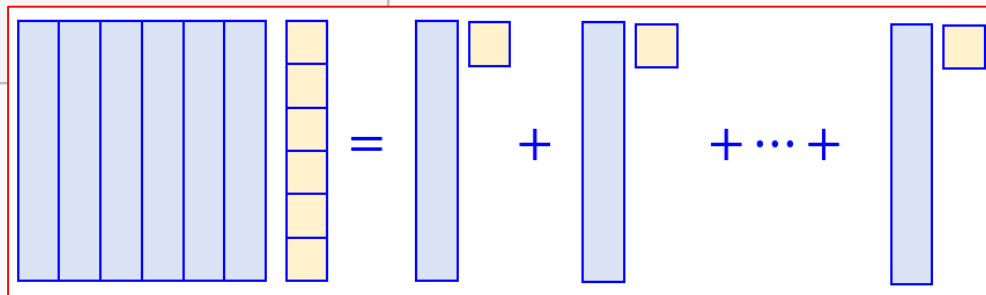
```
for i=1 to m
  y[i]=0.0
  for j=1 to n
    y[i]=y[i]+A[i][j]*x[j]
  end for
end for
```



矩阵向量乘积

- 实现方法二：j-i 循环 — 列方式

```
// 需先赋 y 初值 0.0  
for j=1 to n  
  for i=1 to m  
    y[i]=y[i]+A[i][j]*x[j]  
  end for  
end for
```



C_MatVec.c

例：分别用两种方式计算 $y = Ax$ ，其中 A 和 x 的元素为 $[0,1]$ 中的随机小数， m, n 取 8192，输出两种方法所需的时间。

矩阵向量并行乘积

■ 并行算法一

- 自动并行: **for** 循环共享

OMP_MatVec_for.c

```
double A[m][n], x[n], y[m];  
... ..  
  
#pragma omp parallel for shared(A,x,y2) private(j)  
for(i=0; i<m; i++)  
{  
    y2[i] = 0.0;  
    for(j=0; j<n; j++)  
        y2[i] += A[i][j] * x[j];  
}
```

矩阵向量并行乘积

■ 并行算法二

- 手工划分：按行划分

$$Ax = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{p-1} \end{bmatrix} x = \begin{bmatrix} A_0 x \\ A_1 x \\ \vdots \\ A_{p-1} x \end{bmatrix}$$

每个线程计算 $A_i x$ 。

† 划分时需考虑负载平衡，可采用卷帘方式。

矩阵向量并行乘积

```
#pragma omp parallel shared(A,x,y2) private(i,j,tid,nthreads)
{
    tid = omp_get_thread_num();
    nthreads = omp_get_num_threads();
    for(i=tid; i<m; i+=nthreads)
    {
        y2[i] = 0.0;
        for(j=0; j<n; j++)
            y2[i] += A[i][j] * x[j];
    }
}
```

OMP_MatVec_row.c

矩阵向量并行乘积

■ 并行算法三

- 手工划分：按 **列** 划分

将矩阵 **A** 按列划分，并对 **x** 也做相应的划分，即

$$Ax = [A_0, A_1, \dots, A_{p-1}] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{p-1} \end{bmatrix} = A_0 x_0 + A_1 x_1 + \dots + A_{p-1} x_{p-1}$$

每个线程计算 $A_i x_i$ ，然后求和。