BLAS：基本线性代数子程序库

GitHub Copilot

2024 年 4 月 10 日

摘要

BLAS（基本线性代数子程序）是一组用于执行基本的线性代数操作的软件库，包括向量加法、向量和矩阵乘法、向量点积等。这些操作是许多科学计算应用的基础，包括线性代数、机器学习、物理模拟等。BLAS 库通常分为三级，分别对应向量-向量操作、矩阵-向量操作和矩阵-矩阵操作。BLAS 的一个重要特点是它的操作是高度优化的，许多 BLAS 库针对特定的硬件架构进行了优化，以提供最高的性能。这使得 BLAS 成为了许多高性能计算应用的基础。BLAS 最初是用 Fortran 编写的，但现在也有许多其他语言的接口，例如 C（CBLAS）、Python（NumPy、SciPy）等。

1 矩阵乘法算法的发展和优化

矩阵乘法是线性代数中的基本操作，其算法的发展和优化历程非常丰富。最初，矩阵乘法的算法是直接的，时间复杂度为 $O(n^3)$，其中 n 是矩阵的维数。这是因为对于两个 $n \times n$ 的矩阵，我们需要对每一对行和列进行点积运算，每次点积运算需要 $O(n)$ 的时间，总共需要进行 n^2 次点积运算，因此总的时间复杂度为 $O(n^3)$。

然而，随着计算机科学的发展，人们发现了许多更高效的矩阵乘法算法。例如，Strassen 在 1969 年提出了一种新的矩阵乘法算法，该算法的时间复杂度为 $O(n^{log_2 7})$，这是第一次将矩阵乘法的时间复杂度降低到 $O(n^3)$ 以下。Strassen 的算法利用了矩阵乘法的分治思想，将大矩阵分解成小矩阵，然后递归地进行矩阵乘法。

此后，许多更高效的矩阵乘法算法被提出。例如，Coppersmith 和 Winograd 在 1987 年提出了一种新的矩阵乘法算法，该算法的时间复杂度为 $O(n^{2.376})$，这是目前已知的最高效的矩阵乘法算法。

然而，尽管这些高效的矩阵乘法算法在理论上具有更低的时间复杂度，但在实际应用中，由于这些算法的常数因子较大，以及对矩阵大小的限制，直接的 $O(n^3)$ 矩阵乘法算法仍然是最常用的。为了提高直接矩阵乘法的效率，人们发展了许多优化技术，例如块状矩阵乘法、并行矩阵乘法等。这些优化技术使得矩阵乘法在现代计算机上能够高效地执行，成为了许多科学计算和工程应用的基础。

2 BLAS 介绍

BLAS（基本线性代数子程序）是一组用于执行基本的线性代数操作的软件库。这些操作包括向量加法、向量和矩阵乘法、向量点积等，是许多科学计算应用的基础，包括线性代数、机器学习、物理模拟等。

BLAS 库通常分为三级，分别对应向量-向量操作（Level 1 BLAS）、矩阵-向量操作（Level 2 BLAS）和矩阵-矩阵操作（Level 3 BLAS）。这种分级结构使得 BLAS 能够满足各种复杂度的线性代数运算需求。

BLAS 的一个重要特点是它的操作是高度优化的。许多 BLAS 库（例如 OpenBLAS、ATLAS 等）针对特定的硬件架构进行了优化，以提供最高的性能。这使得 BLAS 成为了许多高性能计算应用的基础。

BLAS 最初是用 Fortran 编写的，但现在也有许多其他语言的接口，例如 C（CBLAS）、Python（NumPy、SciPy）等。这些接口使得 BLAS 可以在各种编程环境中使用，极大地提高了其应用的灵活性和便利性。

3 BLAS 的级别

3.1 Level 1 BLAS

Level 1 BLAS 主要包括向量-向量操作，例如向量加法、向量点积、向量缩放等。这些操作在许多基础的线性代数运算中都有应用，例如在求解线性方程组、计算向量范数等问题中。

3.2 Level 2 BLAS

Level 2 BLAS 主要包括矩阵-向量操作，例如矩阵和向量的乘法、矩阵和向量的点积等。这些操作在许多复杂的线性代数运算中都有应用，例如在

求解线性方程组、计算矩阵范数等问题中。

3.3 Level 3 BLAS

Level 3 BLAS 主要包括矩阵-矩阵操作，例如矩阵乘法、矩阵转置等。这些操作在许多高级的线性代数运算中都有应用，例如在求解线性方程组、计算矩阵特征值等问题中。

4 BLAS 的实现

这部分介绍几种主要的 BLAS 实现，如 Netlib BLAS、OpenBLAS 和 ATLAS。

BLAS 有多种实现，这些实现在性能和功能上有所不同。以下是几种主要的 BLAS 实现：

4.1 Netlib BLAS

Netlib BLAS 是 BLAS 的原始实现，由 Fortran 编写。它提供了所有 BLAS 操作的基本实现，但没有针对特定硬件进行优化。因此，虽然 Netlib BLAS 在所有系统上都可以运行，但其性能可能不如其他实现。

4.2 OpenBLAS

OpenBLAS 是一个开源的 BLAS 实现，由 C 和 Fortran 编写。它针对许多常见的 CPU 架构进行了优化，包括 Intel、AMD、ARM 等。OpenBLAS 还提供了一些额外的功能，例如多线程支持。

4.3 ATLAS

ATLAS（自动调整线性代数软件）是另一个开源的 BLAS 实现。ATLAS 的特点是它会在安装时自动调整其性能，以适应特定的硬件。这使得 ATLAS 可以在各种不同的系统上提供良好的性能。

5 典型例子: cblas_dgemm

在这部分, 我们将展示如何使用 'cblas_dgemm'函数, 并将自己的实现与标准实现进行时间对比。

```
// cblas_dgemm的标准方法签名
void cblas_dgemm(const enum CBLAS_ORDER Order, const
    enum CBLAS_TRANSPOSE TransA,
                    const enum CBLAS_TRANSPOSE TransB,
                        const int M, const int N,
                    const int K, const double alpha,
                        const double *A,
                    const int lda, const double *B, const
                         int ldb,
                    const double beta, double *C, const
                        int ldc) {
    // 这里是乘法运算的逻辑
    // C := alpha*op( A )*op( B ) + beta*C

    // 参数说明:
    // Order: 矩阵存储顺序, 可以是CBLAS_ORDER::
        CblasRowMajor (行优先) 或CBLAS_ORDER::
        CblasColMajor (列优先)
    // TransA: 指定是否对矩阵A进行转置, 可以是
        CBLAS_TRANSPOSE::CblasNoTrans (不转置) 或
        CBLAS_TRANSPOSE::CblasTrans (转置)
    // TransB: 指定是否对矩阵B进行转置, 可以是
        CBLAS_TRANSPOSE::CblasNoTrans (不转置) 或
        CBLAS_TRANSPOSE::CblasTrans (转置)
    // M: 矩阵A的行数
    // N: 矩阵B的列数
    // K: 矩阵A的列数和矩阵B的行数
    // alpha: 系数, 用于乘以矩阵A和B的乘积
    // A: 指向矩阵A的指针
```

```
    // lda: 矩阵A的领先维度 (如果Order是CblasRowMajor
        , 则为A的列数; 如果Order是CblasColMajor, 则为A
        的行数)
    // B: 指向矩阵B的指针
    // ldb: 矩阵B的领先维度 (如果Order是CblasRowMajor
        , 则为B的列数; 如果Order是CblasColMajor, 则为B
        的行数)
    // beta: 系数, 用于乘以矩阵C
    // C: 指向矩阵C的指针
    // ldc: 矩阵C的领先维度 (如果Order是CblasRowMajor
        , 则为C的列数; 如果Order是CblasColMajor, 则为C
        的行数)
}

// 这里是GitHub Copilot的一个块矩阵乘法实现
void manual_dgemm2(int n, double* A, double* B, double
    * C) {
    int i, j, k, i1, j1, k1;
    // 外部三个循环遍历矩阵A和B的块
    for (i = 0; i < n; i += BLOCK_SIZE) {
        for (j = 0; j < n; j += BLOCK_SIZE) {
            for (k = 0; k < n; k += BLOCK_SIZE) {
                // 内部三个循环在每个块内进行矩阵乘法
                for (i1 = i; i1 < i + BLOCK_SIZE; ++i1
                    ) {
                    for (j1 = j; j1 < j + BLOCK_SIZE;
                        ++j1) {
                        double sum = 0.0;
                        for (k1 = k; k1 < k +
                            BLOCK_SIZE; ++k1) {
                            // 计算矩阵A的当前行与矩阵
                                B的当前列的点积
                            sum += A[i1*n + k1] * B[k1
                                *n + j1];
```

```
                        }
                        // 将结果加到矩阵C的相应位置
                        C[i1*n + j1] += sum;
                    }
                }
            }
        }
    }
}

// 这里是多重循环的实现
void manual_dgemm(int n, double* A, double* B, double*
    C) {
    int i, j, k;
    // 外部两个循环遍历矩阵A的行和矩阵B的列
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            double sum = 0.0;
            // 内部循环计算矩阵A的当前行与矩阵B的当前
                列的点积
            for (k = 0; k < n; k++) {
                sum += A[i*n + k] * B[k*n + j];
            }
            // 将结果存储在矩阵C的相应位置
            C[i*n + j] = sum;
        }
    }
}
```

6 实验结果

我们对 cblas_dgemm、manual_dgemm 和 manual_dgemm2 三种矩阵乘法方法进行了性能测试，矩阵的大小为 1600*1600。测试结果如下：

- cblas_dgemm 的运行时间为 0.528376 秒
- manual_dgemm 的运行时间为 12.630309 秒
- manual_dgemm2 的运行时间为 7.732212 秒

从结果可以看出，cblas_dgemm 的性能明显优于 manual_dgemm 和 manual_dgemm2。这主要是因为 cblas_dgemm 是使用 BLAS 库实现的，该库针对特定的硬件架构进行了优化，因此能够提供最高的性能。

7 结论

BLAS（基本线性代数子程序）是一组用于执行基本的线性代数操作的软件库，包括向量加法、向量和矩阵乘法、向量点积等。BLAS 库通常分为三级，分别对应向量-向量操作、矩阵-向量操作和矩阵-矩阵操作。许多 BLAS 库针对特定的硬件架构进行了优化，以提供最高的性能。

BLAS 的多种实现，如 Netlib BLAS、OpenBLAS 和 ATLAS，使得它可以在各种编程环境中使用，极大地提高了其应用的灵活性和便利性。然而，尽管 BLAS 已经有了很多优秀的实现，但仍有许多挑战和机会。例如，随着硬件的发展，如何进一步优化 BLAS 以利用新的硬件特性是一个重要的问题。此外，如何将 BLAS 更好地集成到其他科学计算库和应用中，以提供更高级的功能，也是一个值得研究的问题。

总的来说，BLAS 是科学计算的基础，它的发展将对许多领域产生深远影响。我们期待看到 BLAS 在未来的发展和创新。