General Matrix Multiplication Optimization

Apple Zhang

College of Computer Science and Software Engineering, Shenzhen University

December 5, 2022

Table of Contents

深圳大学 SHENZHEN UNIVERSITY

Table of Contents

Trivial Matrix Multiplication

$$C_{ij} = \sum_{r=1}^{n} A_{ir} B_{rj}. \tag{1}$$

```
void gemm_trivial(int m, int n, int k, double *a, int lda,
                                     double *b, int ldb,
                                     double *c, int ldc)
{
    // TRIVIAL MATRIX MULTIPLICATION
    for (int j = 0; j < n; j++) {
        for (int i = 0; i < m; i++) {
            for (int r = 0; r < k; r++) {
                C(i, j) += A(i, r) * B(r, j);
            }
        }
    }
}
```

Simple cache optimization

Change the iteration order: $jir \to jri$.

```
void gemm_cache(int m, int n, int k, double *a, int lda,
                                  double *b, int ldb,
                                  double *c, int ldc)
{
    // OPTIMIZE THE ORDER OF MATRIX MULTIPLICATION
    for (int j = 0; j < n; j++) {
        for (int r = 0; r < k; r++) {
            register double brj = B(r, j);
            for (int i = 0; i < m; i++){
                C(i, j) += A(i, r) * brj;
            }
        }
    }
}
```

Simple cache optimization

Scan the matrix in column order, and reduce the cache miss.

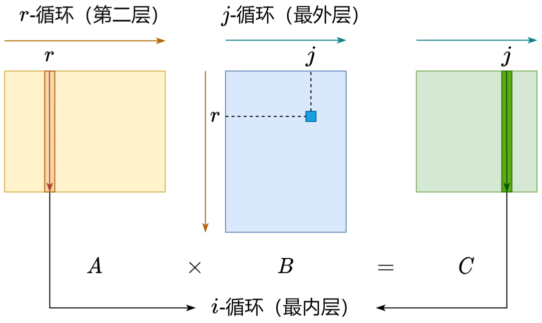

Figure: Simple cache opitmization

Compute 1x4 vector

Avoid reading the same elements from matrix A repeatedly.

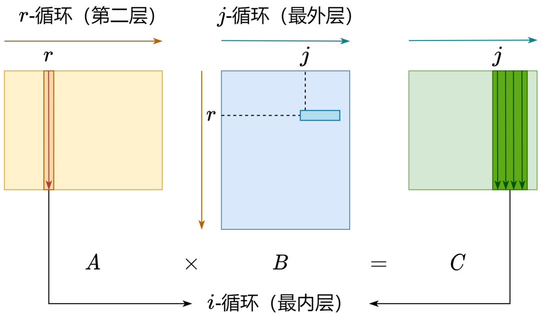

Figure: Compute four elements in each iteration.

Compute 4x4 block

Scan four rows of A and four columns of B.

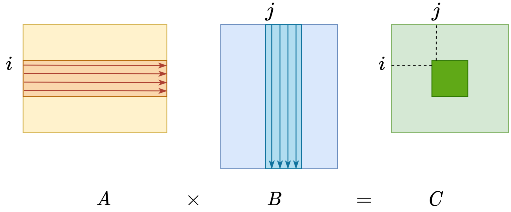

Figure: Compute a 4x4 matrix in each iteration.

Compute 4x4 block with AVX2

Consider SIMD, use AVX2 instructions for acceleration.

```
for (int r = 0; r < k; r++) {
    ar_reg.v = _mm256_load_pd(&A(0, r));

    b0_reg.v = _mm256_broadcast_sd(br0_ptr++);
    b1_reg.v = _mm256_broadcast_sd(br1_ptr++);
    b2_reg.v = _mm256_broadcast_sd(br2_ptr++);
    b3_reg.v = _mm256_broadcast_sd(br3_ptr++);

    c0_reg.v = _mm256_add_pd(c0_reg.v,
                  _mm256_mul_pd(ar_reg.v, b0_reg.v));
    c1_reg.v = _mm256_add_pd(c1_reg.v,
                  _mm256_mul_pd(ar_reg.v, b1_reg.v));
    c2_reg.v = _mm256_add_pd(c2_reg.v,
                  _mm256_mul_pd(ar_reg.v, b2_reg.v));
    c3_reg.v = _mm256_add_pd(c3_reg.v,
                  _mm256_mul_pd(ar_reg.v, b3_reg.v));
}
```

Block matrices optimization

Since for matrix multiplication, we have

$$C = AB = \begin{bmatrix} A_1 & A_2 & \cdots & A_K \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_K \end{bmatrix} \tag{2}$$

we can partition the matrices for better **spatial locality**.

Block matrices optimization

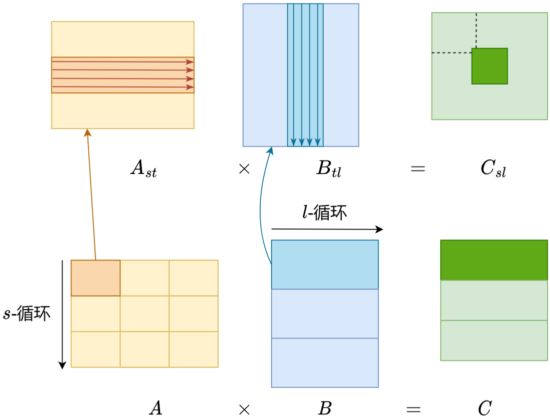

Figure: Block matrix multiplication

Reorganize the matrices

Rearrange the elements of the matrices at beginning for better spatial locality.

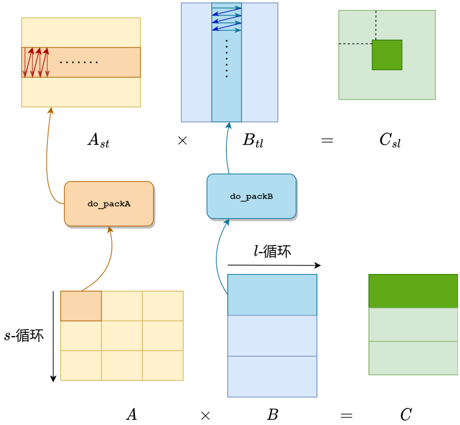

Figure: Rearrange the elements of matrices

Multi-thread with OpenMP

Use OpenMP to implement multi-thread.

```c
void gemm_pack_memory(int m, int n, int k, double *a, int lda,
                                 double *b, int ldb,
                                 double *c, int ldc)
{
    for (int l = 0; l < k; l += ikk) {
        int lb = min(k - l, ikk);
#ifdef _OPENMP // MULTI-THREAD VIA OPENMP
        #pragma omp parallel for schedule(dynamic)
#endif
        for (int s = 0; s < m; s += ikm) {
            int sb = min(m - s, ikm);
            inner_kernel_packAB(sb, n, lb, &A(s, l), lda, &B(l, 0),
                ldb, &C(s, 0), ldc);
        }
    }
}
```

SHENZHEN UNIVERSITY

Table of Contents

Evaluation

- OS: Linux (WSL)
- CPU: Intel i5-11400H

```
(base) apple@LAPTOP-INDIFA6M:~/codes/csexp1-gemm$ ./gemm_test 1024 6
#thread -> 6/12
Trivial:        3.006579 sec
Cache:          0.328501 sec
1x4:            0.229344 sec
4x4:            0.510782 sec
4x4-avx:        0.327548 sec
Block:          0.120619 sec
packOpenMP:     0.032284 sec

max elem-wise error = 1.63425e-13
CORRECT
final speedup: 93.129073x
(base) apple@LAPTOP-INDIFA6M:~/codes/csexp1-gemm$
```

Figure: The results of the program.

Evaluation

$$\text{GFLOPS} = \frac{2 \times 10^{-9} mnk}{\text{time}} \tag{3}$$

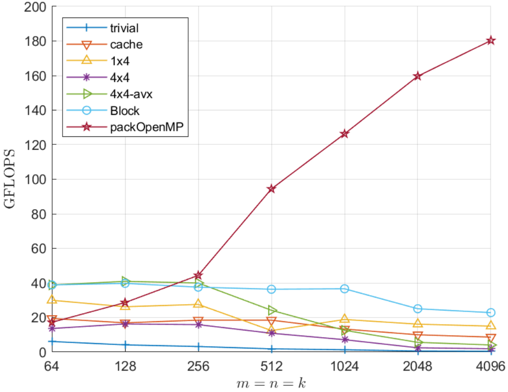

Figure: GFLOPS of each optimization.

Evaluation

But compared with Matlab (MKL)...

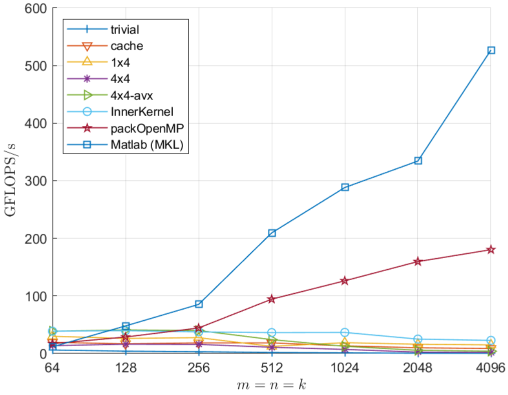

Figure: GFLOPS of each optimization.

Thanks!

My full codes can be viewed in
`https://github.com/Apple-Zhang/optimize-gemm`