gemm optimization

JX-Ma

2023/11/24

1 实验环境

- 系统:ubantu20.4
- gcc version:13.1.0
- 优化选项:-O2
- CPU:AMD Ryzen 7 6800H 3.20GHz

2 优化函数

2.1 无优化

```
void directConvolution_tensor(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s)
{
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); ++m){
for (int64_t n = 0; n < C.num_width(); ++n){
for (int64_t r = 0; r < B.num_channel(); ++r){
for (int64_t u = 0; u < B.num_height(); ++u){
for (int64_t v = 0; v < B.num_width(); ++v) {
    // AI=2/8*3=1/12
   C(i, j, m, n) += A(i, r, m * s + u, n * s + v) * B(j, r, u, v);
    }
}
}
}
}
}
}
}
```

2.2 优化 1

主要优化部分: 利用指针来进行地址偏移寻找数据

```
template<typename T>
void directConvolution_tensor1(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s)
{
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); ++m){
for (int64_t n = 0; n < C.num_width(); ++n){
for (int64_t r = 0; r < B.num_channel(); ++r){
for (int64_t u = 0; u < B.num_height(); ++u){
    // AI = 6/7*8=3/28
    AddDot(B.num_width(),&A(i,r,m*s+u,n*s),1,&B(j, r, u, 0),&C(i, j, m, n));
    //C.setTensors(i, j, m, n, (A(i, r, m + u, n + v) * B(j, r, u, v)));
    }
}
}
}
}
}
}
void AddDot(int64_t k, double * A,int intcx,double *B,double *C)
{
    //2*3/8*7
    for(int p = 0; p < k; ++p){
        *C += *A++ * *B++;
    }
}
```

2.3 优化 2-分块:1*3

```
template<typename T>
void directConvolution_tensor2(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s)
{
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); ++m){
for (int64_t n = 0; n < C.num_width(); n+=3){
for (int64_t r = 0; r < B.num_channel(); r++){
```

```
for (int64_t u = 0; u < B.num_height(); ++u){
    // AI = 6*3/8*(7+6+6)=9/76 //地址偏移次数4*3次
    AddDot(B.num_width(),&A(i,r,m*s+u,n*s),1,&B(j, r, u, 0),&C(i, j, m, n));
    AddDot(B.num_width(),&A(i,r,m*s+u,n*s+1),1,&B(j, r, u, 0),&C(i, j, m, n+1));
    AddDot(B.num_width(),&A(i,r,m*s+u,n*s+2),1,&B(j, r, u, 0),&C(i, j, m, n+2));
    }
}
}
}
}
}
}
```

2.4 优化 2-分块:3*3

```
template<typename T>
void directConvolution_tensor7(Tensor<T>& A, Tensor<T>& B, Tensor<T>& C, int64_t s) {
for (int64_t i = 0; i < C.num_batch(); ++i) {
for (int64_t j = 0; j < C.num_channel(); ++j) {
for (int64_t m = 0; m < C.num_height(); m+=3) {
for (int64_t n = 0; n < C.num_width(); n += 3) {
for (int64_t r = 0; r < B.num_channel(); r++) {
for (int64_t u = 0; u < B.num_height(); ++u) {
    AddDot(B.num_width(), &A(i, r, m * s + u, n * s), 1,
    &B(j, r, u, 0), &C(i, j, m, n));
    AddDot(B.num_width(), &A(i, r, m * s + u, n * s + 1), 1,
    &B(j, r, u, 0), &C(i, j, m, n + 1));
    AddDot(B.num_width(), &A(i, r, m * s + u, n * s + 2), 1,
    &B(j, r, u, 0), &C(i, j, m, n + 2));

    AddDot(B.num_width(), &A(i, r, m * s + u + 1, n * s), 1,
    &B(j, r, u, 0), &C(i, j, m+1, n));
    AddDot(B.num_width(), &A(i, r, m * s + u + 1, n * s + 1), 1,
    &B(j, r, u, 0), &C(i, j, m+1, n + 1));
    AddDot(B.num_width(), &A(i, r, m * s + u + 1, n * s + 2), 1,
    &B(j, r, u, 0), &C(i, j, m+1, n + 2));

    AddDot(B.num_width(), &A(i, r, m * s + u + 2, n * s), 1,
    &B(j, r, u, 0), &C(i, j, m+2, n));
    AddDot(B.num_width(), &A(i, r, m * s + u + 2, n * s + 1), 1,
    &B(j, r, u, 0), &C(i, j, m+2, n + 1));
    AddDot(B.num_width(), &A(i, r, m * s + u + 2, n * s + 2), 1,
    &B(j, r, u, 0), &C(i, j, m+2, n + 2));
        }
}
}
}
}
}
}
```

2.5 优化 3-分块 1*3

主要优化部分: 充分利用指针偏移，

```
template<typename T>
void directConvolution_tensor3(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s){
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); ++m){
for (int64_t n = 0; n < C.num_width(); n+=3){
for (int64_t r = 0; r < B.num_channel(); r++){
for (int64_t u = 0; u < B.num_height(); ++u){
    AddDot1x3(B.num_width(),&A(i,r,m*s+u,n*s),1,&B(j, r, u, 0),&C(i, j, m, n));
    }
}
}
}
}
}
}


    void AddDot1x3(int64_t k, double * A,int intcx,double *B,double *C)
{
    // A = &A(i,r,m*s+u,n*s) B = &B(j, r, u, 0)
    // AI = 3*6/19*8=9/
    AddDot(k,&(*A++),1,&(*B),&(*C++));
    AddDot(k,&(*A++),1,&(*B),&(*C++));
    AddDot(k,&(*A),1,&(*B),&(*C));
}
```

2.6 优化 3-分块 3*3

```
template<typename T>
void directConvolution_tensor8(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s){
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); m+=3){
for (int64_t n = 0; n < C.num_width(); n+=3){
for (int64_t r = 0; r < B.num_channel(); r++){
for (int64_t u = 0; u < B.num_height(); ++u){
    AddDot1x3(B.num_width(), &A(i, r, m * s + u, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m, n));
        AddDot1x3(B.num_width(), &A(i, r, m * s + u+1, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m+1, n));
        AddDot1x3(B.num_width(), &A(i, r, m * s + u+2, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m+2, n));
    }
}
}
}
}
}
}
```

2.7 优化 4-分块 3*1

```
    // Optimization4 unroll = 3
template<typename T>
void directConvolution_tensor4(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s){
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); ++m){
for (int64_t n = 0; n < C.num_width(); n+=3){
for (int64_t r = 0; r < B.num_channel(); r++){
for (int64_t u = 0; u < B.num_height(); ++u){
    AddDot1x3_2(B.num_width(),&A(i,r,m*s+u,n*s),1,&B(j, r, u, 0),&C(i, j, m, n));
    }
}
}
}
}
}
}
    void AddDot1x3_2(int64_t k, double * A,int intcx ,double *B,double *C)
{
    // AddDot(k,&(*A++),1,&(*B),&(*C++));
    int p = 0;
    double* A1=A;
    double* B1=B;
    for(p = 0; p < k; ++p){
        *C += *A++ * *B++;
    }
    //AddDot(k,&(*A++),1,&(*B),&(*C++));
    C = C+1; A = A1+1; B = B1;
     for(p = 0; p < k; ++p){
        *C += *A++ * *B++;
    }
   C = C+1; A = A1+1; B = B1;
    //AddDot(k,&(*A),1,&(*B),&(*C));
     for(p = 0; p < k; ++p){
        *C += *A++ * *B++;
    }
}
```

2.8 优化 4-分块 3*3

```
    // Optimization4 unroll = 3
template<typename T>
void directConvolution_tensor4(Tensor<T>& A,Tensor<T>& B,Tensor<T>& C,int64_t s){
for (int64_t i = 0; i < C.num_batch(); ++i){
for (int64_t j = 0; j < C.num_channel(); ++j){
for (int64_t m = 0; m < C.num_height(); m+=3){
for (int64_t n = 0; n < C.num_width(); n+=3){
for (int64_t r = 0; r < B.num_channel(); r++){
for (int64_t u = 0; u < B.num_height(); ++u){
    AddDot1x3_2(B.num_width(), &A(i, r, m * s + u, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m, n));
        AddDot1x3_2(B.num_width(), &A(i, r, m * s + u + 1, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m + 1, n));
        AddDot1x3_2(B.num_width(), &A(i, r, m * s + u + 2, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m + 2, n));
    }
}
}
}
}
}
}
```

2.9 优化 5-分块 1*3

```
// Optimization5 unroll = 3
template<typename T>
void directConvolution_tensor5(Tensor<T>& A, Tensor<T>& B, Tensor<T>& C, int64_t s) {
for (int64_t i = 0; i < C.num_batch(); ++i) {
for (int64_t j = 0; j < C.num_channel(); ++j) {
for (int64_t m = 0; m < C.num_height(); ++m) {
for (int64_t n = 0; n < C.num_width(); n += 3) {
for (int64_t r = 0; r < B.num_channel(); r++) {
for (int64_t u = 0; u < B.num_height(); ++u) {
    // AI = 6*3/8*(7+6+6)=9/76
    AddDot1x3_3(B.num_width(), &A(i, r, m * s + u, n * s), 1, &B(j, r, u, 0), &C(i, j,
    }
}
}
}
}
}
}
void AddDot1x3_3(int64_t k,double* A, int intcx,double* B, double* C)
{
        // AddDot(k,&(*A++),1,&(*B),&(*C++));
        int p = 0;
        double* C1 = C + 1;
        double* C2 = C1 + 1;
        double* B1 = B;
        double* B2 = B;
        double* A1 = A + 1;
        double* A2 = A1 + 1;
        for (p = 0; p < k; ++p) {
```

```
                    *C += *A++ * *B++;
                    *C1 += *A1++ * *B1++;
                    *C2 += *A2++ * *B2++;
          }
}
```

2.10 优化 5-分块 3*3

```
// Optimization5 unroll = 3
template<typename T>
void directConvolution_tensor10(Tensor<T>& A, Tensor<T>& B, Tensor<T>& C, int64_t s) {
for (int64_t i = 0; i < C.num_batch(); ++i) {
for (int64_t j = 0; j < C.num_channel(); ++j) {
for (int64_t m = 0; m < C.num_height(); m+=3) {
for (int64_t n = 0; n < C.num_width(); n += 3) {
for (int64_t r = 0; r < B.num_channel(); r++) {
for (int64_t u = 0; u < B.num_height(); ++u) {
     AddDot1x3_3(B.num_width(), &A(i, r, m * s + u, n * s), 1,
      &B(j, r, u, 0), &C(i, j, m, n));
          AddDot1x3_3(B.num_width(), &A(i, r, m * s + u + 1, n * s), 1,
      &B(j, r, u, 0), &C(i, j, m + 1, n));
          AddDot1x3_3(B.num_width(), &A(i, r, m * s + u + 2, n * s), 1,
      &B(j, r, u, 0), &C(i, j, m + 2, n));
     }
}
}
}
}
}


\subsection{优化6−分块1*3}
\lstset{language=C++}
\begin{lstlisting}
// Optimization6 unroll = 3
template<typename T>
void directConvolution_tensor6(Tensor<T>& A, Tensor<T>& B, Tensor<T>& C, int64_t s) {
for (int64_t i = 0; i < C.num_batch(); ++i) {
for (int64_t j = 0; j < C.num_channel(); ++j) {
```

```
for (int64_t m = 0; m < C.num_height(); ++m) {
for (int64_t n = 0; n < C.num_width(); n += 3) {
for (int64_t r = 0; r < B.num_channel(); r++) {
for (int64_t u = 0; u < B.num_height(); ++u) {
    AddDot1x3_4(B.num_width(), &A(i, r, m * s + u, n * s), 1, &B(j, r, u, 0), &C(i, j,
    }
}
}
}
}
}
}
void AddDot1x3_4(int64_t k, double* A, int intcx, double* B, double* C)
{
        // AddDot(k,&(*A++),1,&(*B),&(*C++));
        int p = 0;
        double t1, t2, t3;
        t1 = 0.0;
        t2 = 0.0;
        t3 = 0.0;
        double* B1 = B;
        double* B2 = B;
        double* A1 = A + 1;
        double* A2 = A1 + 1;
        for (p = 0; p < k; ++p) {
                t1 += *A++ * *B++;
                t2 += *A1++ * *B1++;
                t3 += *A2++ * *B2++;
        }
        *C++ += t1;
        *C++ += t2;
        *C += t3;
}
```

2.11 优化 6-分块 3*3

```
// Optimization6 unroll = 3
template<typename T>
void directConvolution_tensor11(Tensor<T>& A, Tensor<T>& B, Tensor<T>& C, int64_t s) {
for (int64_t i = 0; i < C.num_batch(); ++i) {
```

```
for (int64_t j = 0; j < C.num_channel(); ++j) {
for (int64_t m = 0; m < C.num_height(); m+=3) {
for (int64_t n = 0; n < C.num_width(); n += 3) {
for (int64_t r = 0; r < B.num_channel(); r++) {
for (int64_t u = 0; u < B.num_height(); ++u) {
    AddDot1x3_4(B.num_width(), &A(i, r, m * s + u, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m, n));
    AddDot1x3_4(B.num_width(), &A(i, r, m * s + u + 1, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m + 1, n));
    AddDot1x3_4(B.num_width(), &A(i, r, m * s + u + 2, n * s), 1,
     &B(j, r, u, 0), &C(i, j, m + 2, n));
    }
}
}
}
}
}
}
```

3 实验结果

数据

- **input:1x512x8x8**
- **F: 512x512x3x3**
- **output:1*512*6*6**
- **s : 1**
- **padding: no**
- **dateType: double**

本次实验虽说是 11 种优化算法，实际上只有 6 种，后面五种优化算法分别在分块为 1x3，和分块为 3*3 做相同的优化

表 1: Gfloat 表

数据	无优化	优化 1	优化 2	优化 3	优化 4	优化 5	优化 6
分块 1x3	0.427695	1.25999	1.49834	2.2886	2.2566	2.31094	2.97743
分块 3x3	0.427695	1.25999	1.55918	2.45546	2.50008	2.55816	3.25192

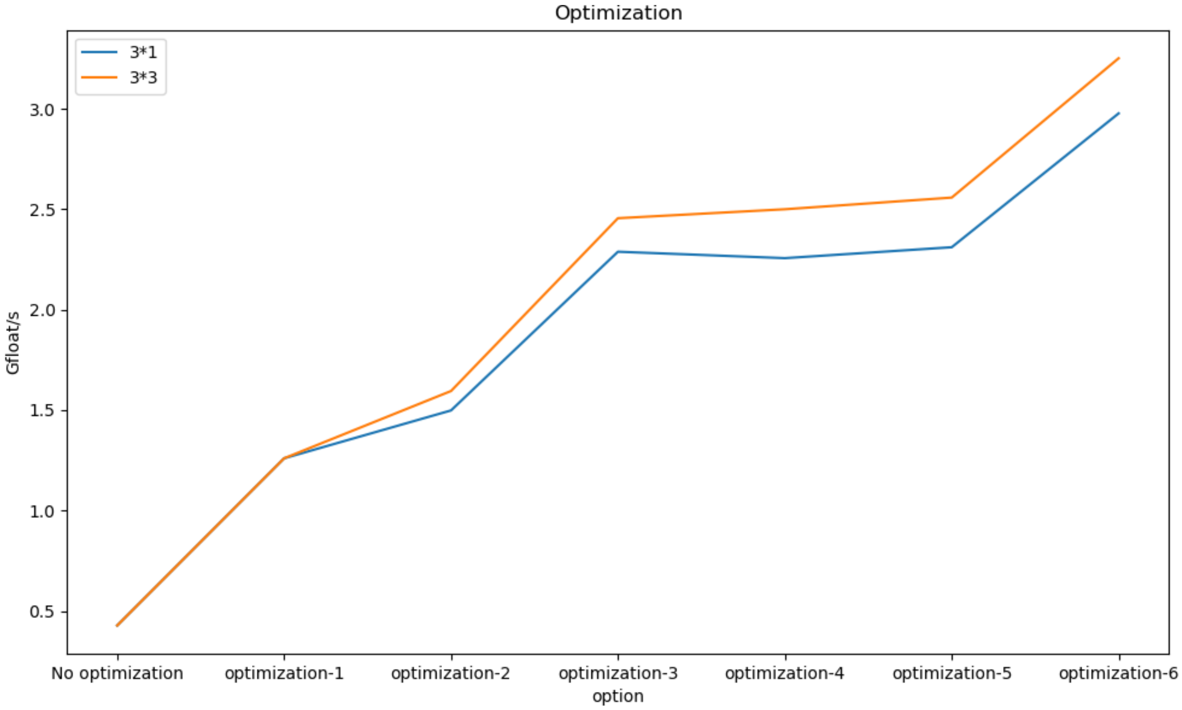
Optimization
3*1
3*3
Gfloat/s
0.5
1.0
1.5
2.0
2.5
3.0
No optimization
optimization-1
optimization-2
optimization-3
optimization-4
optimization-5
optimization-6
option

图 1: Gfloat