week-13

JX-Ma

2023/12/11

1 实验环境

- 系统: Ubantu 22.01
- gcc version : 9.5.0
- 优化选项: -O2
- cpu:AMD Ryzen 7 6800H 3.20GHz

2 实验准备

2.1 数据

- 存储类：Tensor1D
- inputTensor: 1,3,227,227
- filterTensor: 96,3,12,12
- outputTensor: 1,96,216,216
- stride : 1

2.2 测试代码

```
#include "../include/optimization/gemm2.hpp"
#include "../include/util/Timer.h"
#include<fstream>
#include <filesystem>

#define GIGA 1e9
#define LOGPATH "../../../log"
#define LOGFILE "../../../log/log.txt"
#define DATAPATH "../../../data/gemm/"
```

```
using namespace std;
namespace fs = filesystem;



bool dataPath(string path,string c){
    // 判断文件夹是否存在
    if(!fs::exists(DATAPATH)){
        fs::create_directory(DATAPATH);
    }
    // 判断文件夹是否存在
    if(!fs::exists(DATAPATH + path)){
        fs::create_directory(DATAPATH + path);
    }
    // 判断文件夹是否存在
    if(!fs::exists(DATAPATH + path + "/" + c)){
        return fs::create_directory(DATAPATH + path + "/" + c);
    }
    return true;
}


int main(int argc,char *argv[])
{

    string path = string(argv[1]);

    const int count = stoi(string(argv[2]));
    cout<<path<<": count "<< count<<"..."<<endl;

    ofstream ofs; //打开一个写的文件流
    dataPath(path,"");
    path = DATAPATH + path;
    string str_txt = path+"/times.txt";
    ofs.open(str_txt, ios::app);
    Tensor1D<double> A(1,3,227,227);
    Tensor1D<double> B(96,3,12,12);
    Tensor1D<double> C(1,96,216,216);
    A.randomAssign(10);
```

```
B.randomAssign(10);
Timer T;
double elapsed;
// opt none
// T.start();
// directConvolution_tensor(A,B,C,1);
// T.stop();

// // opt 1
// T.start();
// directConvolution_tensor1(A,B,C,1);
// T.stop();

// opt 2
// T.start();
// directConvolution_tensor2(A,B,C,1);
// T.stop();

// opt 1x4_1
// T.start();
// directConvolution_tensor1x4_1(A,B,C,1);
// T.stop();

// // opt 1x4_2
// T.start();
// directConvolution_tensor1x4_2(A,B,C,1);
// T.stop();

 // opt 1x4_3
// T.start();
// directConvolution_tensor1x4_3(A,B,C,1);
// T.stop();

//  // opt 1x4_4
// T.start();
// directConvolution_tensor1x4_4(A,B,C,1);
// T.stop();

//  // opt 1x4_5
```

```
// T.start();
// directConvolution_tensor1x4_5(A,B,C,1);
// T.stop();

//  // opt 1x4_6
// T.start();
// directConvolution_tensor1x4_6(A,B,C,1);
// T.stop();

//  // opt 1x4_7
// T.start();
// directConvolution_tensor1x4_7(A,B,C,1);
// T.stop();

//  // opt 1x4_5_A
// T.start();
// directConvolution_tensor1x4_5_A(A,B,C,1);
// T.stop();

//  // opt 1x4_7_A
// T.start();
// directConvolution_tensor1x4_7_A(A,B,C,1);
// T.stop();



// opt 4x4_1
// T.start();
// directConvolution_tensor4x4_1(A,B,C,1);
// T.stop();

// // opt 4x4_2
// T.start();
// directConvolution_tensor4x4_2(A,B,C,1);
// T.stop();

 // opt 4x4_3
// T.start();
// directConvolution_tensor4x4_3(A,B,C,1);
```

```
        // T.stop();

        //  // opt 4x4_4
        // T.start();
        // directConvolution_tensor4x4_4(A,B,C,1);
        // T.stop();

        //  // opt 4x4_5
        // T.start();
        // directConvolution_tensor4x4_5(A,B,C,1);
        // T.stop();

        //  // opt 4x4_6
        // T.start();
        // directConvolution_tensor4x4_6(A,B,C,1);
        // T.stop();

        //  // opt 4x4_7
        T.start();
        directConvolution_tensor4x4_7(A,B,C,1);
        T.stop();

        //  // opt 4x4_5_A
        // T.start();
        // directConvolution_tensor4x4_5_A(A,B,C,1);
        // T.stop();

        //  // opt 4x4_7_A
        // T.start();
        // directConvolution_tensor4x4_7_A(A,B,C,1);
        // T.stop();


        ofs<< T.elapsed()<<" ";
        ofs.close();
    }
```

2.3 算法正确性验证

```
#include "../include/optimization/gemm2.hpp"
```

```
#include "../include/util/Timer.h"
using namespace std;

int main()
{

    Tensor1D<double> A(1, 1, 11, 11);
    Tensor1D<double> B(1, 1, 4, 4);
    Tensor1D<double> C(1, 1, 8, 8);
    A.randomAssign(10);
    B.randomAssign(10);
    cout << "A:xx" << endl;
    A.print();
    cout << "Bxx" << endl;
    B.print();
    Timer T;
    T.start();
    directConvolution_tensor(A, B, C, 1);
    T.stop();
    cout << "直接卷积" << endl;
    C.print();

    Tensor1D<double> C1(1, 1, 8, 8);
    directConvolution_tensor1(A, B, C1, 1);
    cout << "优化卷积" << endl;
    C1.print();

    Tensor1D<double> C2(1, 1, 8, 8);
    directConvolution_tensor2(A, B, C2, 1);
    cout << "优化卷积" << endl;
    C2.print();

    Tensor1D<double> C3(1, 1, 8, 8);
    directConvolution_tensor1x4_1(A, B, C3, 1);
    cout << "1x4_1" << endl;
    C3.print();

    Tensor1D<double> C4(1, 1, 8, 8);
    directConvolution_tensor1x4_2(A, B, C4, 1);
```

```
cout << "1x4_2" << endl;
C4.print();

Tensor1D<double> C5(1, 1, 8, 8);
directConvolution_tensor1x4_3(A, B, C5, 1);
cout << "1x4_3" << endl;
C5.print();

Tensor1D<double> C6(1, 1, 8, 8);
directConvolution_tensor1x4_4(A, B, C6, 1);
cout << "1x4_4" << endl;
C6.print();


Tensor1D<double> C7(1, 1, 8, 8);
directConvolution_tensor1x4_5(A, B, C7, 1);
cout << "1x4_5" << endl;
C7.print();

Tensor1D<double> C8(1, 1, 8, 8);
directConvolution_tensor1x4_6(A, B, C8, 1);
cout << "1x4_6" << endl;
C8.print();

Tensor1D<double> C9(1, 1, 8, 8);
directConvolution_tensor1x4_7(A, B, C9, 1);
cout << "1x4_7" << endl;
C9.print();



Tensor1D<double> C11(1, 1, 8, 8);
directConvolution_tensor4x4_1(A, B, C11, 1);
cout << "4x4_1" << endl;
C11.print();

Tensor1D<double> C12(1, 1, 8, 8);
directConvolution_tensor4x4_2(A, B, C12, 1);
cout << "4x4_2" << endl;
```

```
C12.print();

Tensor1D<double> C13(1, 1, 8, 8);
directConvolution_tensor4x4_3(A, B, C13, 1);
cout << "4x4_3" << endl;
C13.print();

Tensor1D<double> C14(1, 1, 8, 8);
directConvolution_tensor4x4_4(A, B, C14, 1);
cout << "4x4_4" << endl;
C14.print();


Tensor1D<double> C15(1, 1, 8, 8);
directConvolution_tensor4x4_5(A, B, C15, 1);
cout << "4x4_5" << endl;
C15.print();

Tensor1D<double> C16(1, 1, 8, 8);
directConvolution_tensor4x4_6(A, B, C16, 1);
cout << "4x4_6" << endl;
C16.print();

Tensor1D<double> C17(1, 1, 8, 8);
directConvolution_tensor4x4_7(A, B, C17, 1);
cout << "4x4_7" << endl;
C17.print();

Tensor1D<double> C21(1, 1, 8, 8);
directConvolution_tensor1x4_5_A(A, B, C21, 1);
cout << "1x4_5_A" << endl;
C21.print();

Tensor1D<double> C22(1, 1, 8, 8);
directConvolution_tensor1x4_7_A(A, B, C22, 1);
cout << "1x4_7_A" << endl;
C22.print();

Tensor1D<double> C23(1, 1, 8, 8);
```

```
    directConvolution_tensor4x4_5_A(A, B, C23, 1);
    cout << "4x4_5_A" << endl;
    C23.print();

    Tensor1D<double> C24(1, 1, 8, 8);
    directConvolution_tensor4x4_7_A(A, B, C24, 1);
    cout << "4x4_7_A" << endl;
    C24.print();

}
```

3 实验步骤和结果

3.1 验证算法正确性

输出每个优化和无优化的结果来对比，如果数据不一样则说明算法错误。
实验结果保存在 data/verify.txt(手动保存)

3.2 实验过程

将每个算法跑 100 遍，将每次的结果放在 data/gemm/opt-Name/time.txt
最后将 100 次结果数据取平均值，算出 gflops.
绘图

3.3 实验结果

表 1: 数据表

类型	opt−none	opt−1	opt−2	ax4_1	ax4_2	ax4_3	ax4_4
1x4time	8.01351	1.61098	1.19589	0.839375	0.808708	0.697931	0.534112
1x4gflops	0.482327	2.40216	3.23594	4.61038	4.78521	5.54473	7.24537
4x4time	8.01351	1.61098	1.19589	0.673429	0.795129	0.782666	0.390342
4x4gflops	0.482327	2.40216	3.23594	5.74646	4.86693	4.94443	9.91396

表 2: 数据续表

类型	ax4_5	ax4_6	ax4_7	ax4_5_A	ax4_7_A
1x4time	0.564337	0.419986	0.425889	0.392032	0.346673
1x4gflops	6.85731	9.21419	9.08648	9.87122	11.1628
4x4time	0.733458	0.360699	0.350656	0.181922	0.166473
4x4gflops	5.27615	10.7287	11.036	21.2719	23.2461

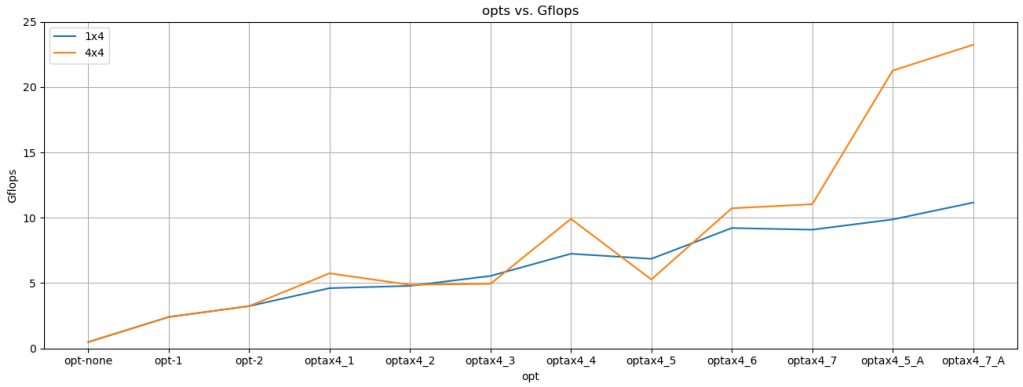

图 1: gflops

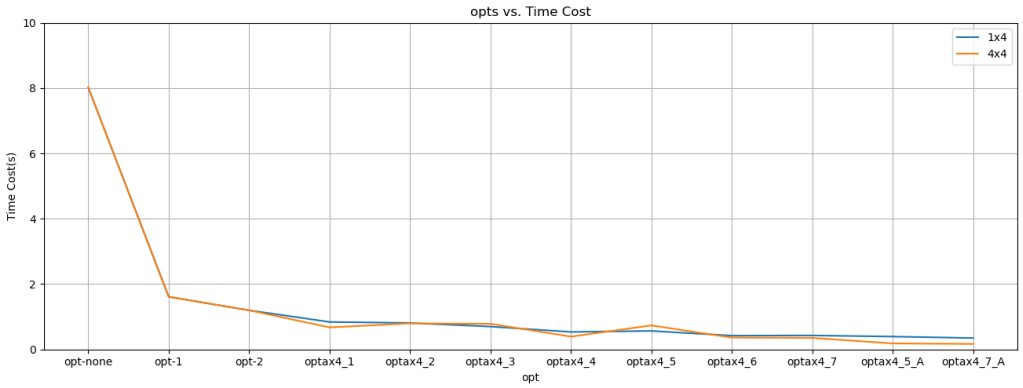

图 2: times

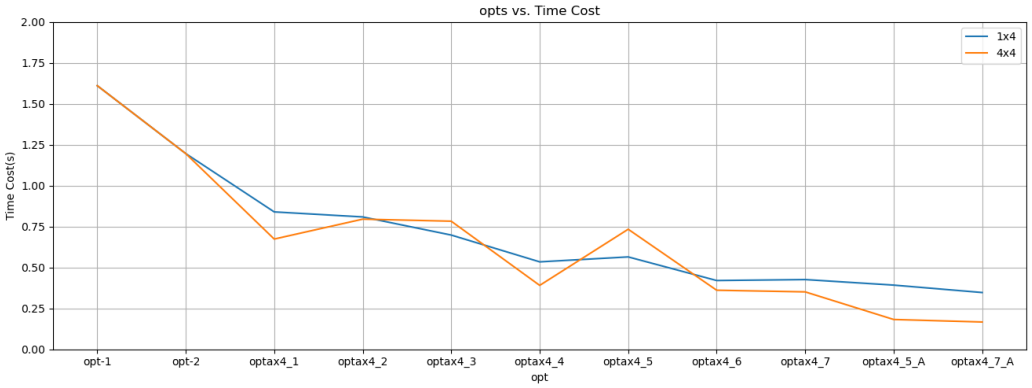
opts vs. Time Cost
Time Cost(s)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
opt-1
opt-2
optax4_1
optax4_2
optax4_3
optax4_4
optax4_5
optax4_6
optax4_7
optax4_5_A
optax4_7_A
opt
1x4
4x4

图 3: times