計数工学プログラミング演習最終レポート

計数工学科システム情報学コース 3 年
03-190615
工藤龍

2019 年 6 月 8 日

1　課題内容

疎行列の 2 乗を様々な手法で計算し，実行時間を測定した．具体的には，行列の保持方法が二次元配列の場合と隣接リストの場合でアルゴリズムを分け，また，計算方法の部分でもいくつかの種類を考えた．

2　手法

今回の実験に用いたのは，以下の 4 つのアルゴリズムである．

- dense ijk
- dense ikj
- sparce transpose
- sparce access

以下，上記の四つの説明をする．dense ijk は，二次元配列の形で行列を保持するアルゴリズムである．次のような形で積を計算する．

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        x = 0;
        for (int k = 0; k < n; k++) {
            x += A[i][k] * A[k][j];
        }
        M[i][j] = x;
    }
}
```

dense ikj は，同じように二次元配列の形で行列を保持するが，積の計算の順序がやや異なる．具体的には以下のようになっている．

```
for (int i = 0; i < n; i++) {
    for (int k = 0; k < n; k++) {
        for (int j = 0; j < m; j++) {
            M[i][j] += A[i][k] * A[k][j];
        }
    }
}
```

sparce transpose は，隣接リストの形で行列を保持するものである．入力した行列と，その転置行列を考えることで計算する方針をとっている．

sparce access は，transpose と同様に隣接リストで行列を保持するが，access 関数を用いることで，転置を考えずに直接積を計算している．

入力した行列は，matrixmarket の行列である．実行時間の計測は，time コマンドの user の値を利用することとした．

3 実験結果

それぞれのアルゴリズムに関し，計算を 5 回繰り返して平均をとった．結果は次の表のようになった．

表 1 アルゴリズムごとの行列の大きさと計算時間 [s]

行列の大きさ	dense ijk	dense ikj	sparce transpose	sparce access
39	0.007	0.007	0.007	0.008
49	0.007	0.007	0.007	0.009
118	0.010	0.008	0.011	0.029
274	0.040	0.020	0.017	0.203
443	0.165	0.045	0.018	0.548
1454	8.835	1.695	0.057	16.914
1612	16.052	2.317	0.066	22.695
1624	17.645	2.355	0.072	23.359
1723	22.589	2.754	0.127	27.605
5300	3291.228	95.127	1.385	879.153

その結果のグラフが次のものである．

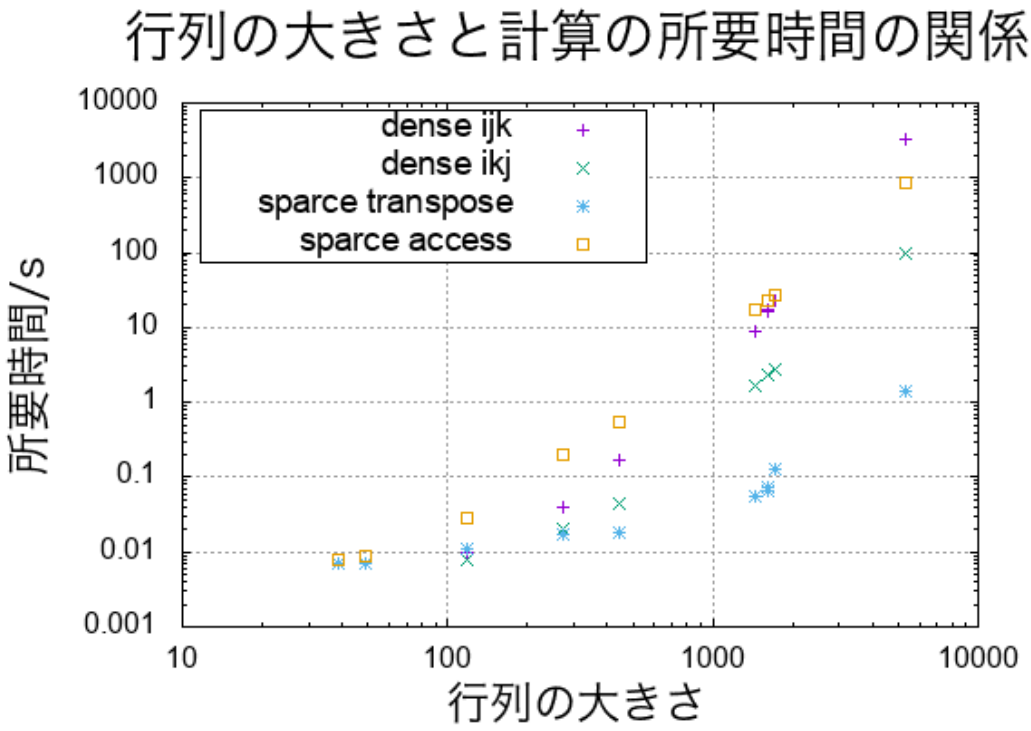

図 1　行列の大きさと計算の所要時間の関係

ここからわかることとしては，全体として計算の速さが sparce transpose，dense ijk，dense ikj，sparce access の順だということである．また，dense ikj と sparce access は最後の行列でだけ順番が逆転している．

4　考察

まず，sparce access が一番遅かったことに関してであるが，これは納得できることだ．何故ならば，access 関数のアルゴリズムを見てみると，一つの値を探すのに，最大でその値が存在するであろう行全てを探索することになるからである．これは，二次元配列で行列を保持している場合よりも大きくなる可能性がある．一方，一行探索しなくても良い場合もあるので，行列の形によっては二次元配列よりも早くなることもあるであろう．sparce access が dence ijk を最後の行列でだけ抜いているのは，行列の形によってたまたまそうなったものと考えられる．

また，特徴的なのは dense ijk と dense ikj の大きな差である．両者の違いは積の計算の順番だけであるのに，このように顕著な差が現れるのは興味深い．これは，以下のように理解できる．まず，ijk の順であると A[i][k] と A[k][j] の両方が一番内側のループ内で毎回変わっていく．一方，ikj の順であると，A[k][j] は一番内側のループで毎回変わるが，A[i][k] は変わらない．計算に使用

する値を毎回変更する必要がなくなるのが，これだけの差が生まれた原因だと考えられる．

また，sparce transpose は，二次元配列ではなく隣接リスト表現で値を保持し，かつ転置してから計算するため，ベクトル同士の積として計算することができる．ゆえに先の ikj よりも負担の少ない形で計算できるため，もっとも実行時間の短いアルゴリズムとなっていると考えられる．

今後の展望としては，二次元配列のアルゴリズムについて今回は ikj と ijk のみ扱ったが，ijk 三つのならびに関して 6 通りの場合が考えられる．それらの違いに関して実験し，考察してみるということが考えられる．