2019年5月11日
情報学研究科 知能情報学専攻
M2 神浦 晃行

逐次計算の高速化に関する課題

$$c_{ij} = \sum_{k=0}^{2047} \frac{a_{ik} b_{kj}}{3}, \quad i, j = 0,1,\cdots,2047.$$

```
for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
      c[i][j] = ZERO;
      for (k = 0; k < N; k++){
            c[i][j] = c[i][j] + a[i][k]*b[k][j]/THREE;
      }
    }
}
```

time = 176.21466

プログラムはそれぞれ, mat_1.c, mat_2.c, mat_3.c, mat_4.c と名付けておく.
mat_1.c - https://github.com/AkiyukiKamiura/computationalscience/blob/master/mat_1.c
mat_2.c - https://github.com/AkiyukiKamiura/computationalscience/blob/master/mat_2.c
mat_3.c - https://github.com/AkiyukiKamiura/computationalscience/blob/master/mat_3.c
mat_4.c - https://github.com/AkiyukiKamiura/computationalscience/blob/master/mat_4.c

問題 1 配列のアクセス方向の変更による高速化を行う.

3重ループの順番を入れ替えるとき, $3! = 6$ 通りのループの組み合わせがある。C言語では配列の同じ行の隣り合う成分がメモリ上の隣に格納されるため、最初の mat.c のプログラムにおいては、b[k][j]の成分が列ごとに読み込まれることとなり、メモリアクセスの効率が悪いと考えられる.
変更したプログラムを以下に示す。まず、kとjのループを入れ替えることにより、配列b[k][j]の読み込まれる順番が、完全にループの順番と一致することになる。また、配列a[i][k]も最後のループjで毎回読み込む必要はないため、変数tmpにループの外で読み込んでおくことにする。そして、配列cの初期化の場所を変更し、プログラムの完成とする。

```
static double tmp;

for (i = 0; i < N; i++){
  for (j = 0; j < N; j++){
    c[i][j] = ZERO;
  }

  for (k = 0; k < N; k++){
    tmp = a[i][k];
    for (j = 0; j < N; j++){
      c[i][j] = c[i][j] +tmp*b[k][j]/THREE;
    }
  }
}
```

結果は time = 73.36652 となり、約59%の時間削減ができたことになる。

問題 2 (1)演算の回数を減らす (2)演算の種類を変更する 事による高速化を行う.

演算の回数を減らすことでの高速化に加え、乗算は高速化する方法が一般化されているのに対して、除算は分けて計算する方法がないため、乗算の方が高速なため、除算を逆数での乗算に変更することにより高速化を行った。具体的には、

(1)演算の回数を減らす => jループの直前に、a[i][j]を3で割ること

(2)演算の種類の変更 => [÷3]を[x(1/3)]とすること

以上により、実装を行った。

```
static double tmp;
static double one_third_part = 1.0/3.0;

for (i = 0; i < N; i++){
  for (j = 0; j < N; j++){
    c[i][j] = ZERO;
  }

  for (k = 0; k < N; k++){
    tmp = a[i][j]*one_third_part;
    for (j = 0; j < N; j++){
      c[i][j] = c[i][j] + tmp*b[k][j];
    }
  }
}
```

結果は time = 66.19242 となり、約62%の時間が削減されたことになる。

問題 3 dgemm.f を使用して高速化を行う.

倍精度の行列積に関する計算を行うdgemmは、dgemm(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)のように定義される。想定されている行列積の数式は、

$$C = \alpha A * B + \beta C$$

である。以下では、dgemmの引数を羅列して説明する。

- ALPHA, BETA - 上記の数式の数値.
- A, B, C - 上記の数式の行列.
- TRANSA, TRANSB - AとBを共役転置するかどうかを示す. (共役転置しないので 'N')
- M, N, K - AはM行K列, BはK行N列, CはM行N列.
- LDA, LDB, LDC - 行列のリーディングディメンジョン(ある列から次の列へのストライド)

以上のことと, fortranとC言語の配列のメモリ配置は逆になるため, aとbが逆になることを踏まえて,
ALPHA = (1/3), BETA = 0, TRANSA = TRANSB = 'N', A = b, B = a, C = c
LDA = LDB = LDC = N とおいた.

```
static double alpha = 1.0/3.0;
static double beta = 0.0;
static int n = N;

dgemm_("N", "N", &n, &n, &n, &alpha, b, &n, a, &n, &beta, c, &n);
```

結果は time = 0.86297 であった. 0.45%になるまでの時間削減が行われた.

問題 4 dgemm.f の内容を参考に高速化を行う.

dgemm.f の内容を参考にする. TRANSA = 'N', TRANSB = 'N' の場合を以下に示す. mat_2.cに加えて、配列の要素が0であった場合に3つめのループをスキップする工夫がなされていることがわかるので、mat_4.cではその工夫を取り入れた.
結果は time = 65.98376 となり, 約62%で問題2の場合よりもわずかに削減されたことがわかる.
この場合分けは行列が疎であればあるほど高速化されるため, 今回のように乱数で初期化された値では0が多くならないと考えられるため, それほど高速化が行われなかった.

```
  DO 90 J = 1,N
     IF (BETA.EQ.ZERO) THEN
       DO 50 I = 1,M
         C(I,J) = ZERO
50       CONTINUE
     ELSE IF (BETA.NE.ONE) THEN
       DO 60 I = 1,M
         C(I,J) = BETA*C(I,J)
60       CONTINUE
     END IF
     DO 80 L = 1,K
       IF (B(L,J).NE.ZERO) THEN
         TEMP = ALPHA*B(L,J)
         DO 70 I = 1,M
           C(I,J) = C(I,J) + TEMP*A(I,L)
70         CONTINUE
       END IF
80     CONTINUE
90   CONTINUE
```

```
static double one_third_part = 1.0/3.0;
static double tmp;

for (i = 0; i < N; i++){
  for (j = 0; j < N; j++){
    c[i][j] = ZERO;
  }

  for (k = 0; k < N; k++){
    tmp = a[i][k];
    if (tmp != ZERO) {
      tmp = one_third_part*tmp;
      for (j = 0; j < N; j++){
        c[i][j] = c[i][j] + tmp*b[k][j];
      }
    }
  }
}
```