

短期連載 ▶ より速く、より少ないメモリで動く効率的なプログラムを書くための

CPU最適化プログラミング

第2回 SIMD 命令 (インテル AVX/AMX 命令) を使った行列演算の最適化

本橋 弘臣

リスト1 4×4行列乗算を行う dot_product() 関数

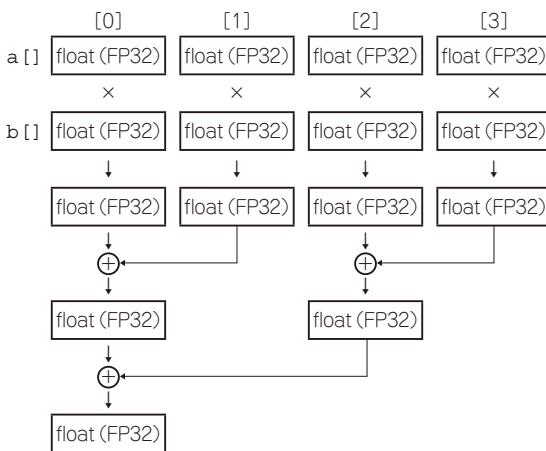
```
// 転置行列を得る関数
void transpose_matrix(const float *mat_b,
                     float *mat_b_trans)
{
    int x, y;

    for (y = 0; y < N; y++) {
        for (x = 0; x < N; x++) {
            mat_b_trans[y, x] = mat_b[x, y];
        }
    }
}

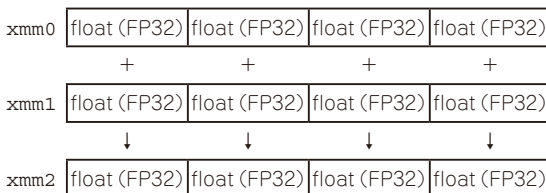
// 4x4 行列の掛け算を計算する関数: メモリ・アクセス改善版
void mat_mul_opt(const float *mat_a,
                 const float *mat_b_trans, float *mat_c)
{
    int x, y;
```

```
    for (y = 0; y < N; y++) {
        for (x = 0; x < N; x++) {
            mat_c[x, y] = dot_product(&mat_a[0, y],
                                      &mat_b_trans[0, x]);
        }
    }

    // floatx4 ベクトルの内積を求める下請け関数: メモリ・アクセス改善版
    INLINE float dot_product(const float *a, const float
                              *b)
    {
        return a[0] * b[0] +
               a[1] * b[1] +
               a[2] * b[2] +
               a[3] * b[3];
    }
}
```



(a) 前回行った方法



(b) インテル SSE 命令を使う方法

図1 dot_product() の加算処理

本連載では、CPUを意識してプログラムを最適化することにより、効率的なプログラムを書くことを目指します。効率的なプログラムは、より速く、より少ないメモリで動作します。これは製品の完成度を高め、ユーザの体験を向上させることに直結します。

第2回はSIMD命令を使った効率的なプログラムの書き方について解説します。(編集部)

4×4行列乗算プログラムをさらに最適化する

● SIMD 命令で最適化するにはデータの並びが重要

前回 (2025年12月号)、4×4行列乗算プログラムを最適化する方法について解説しました。この中で呼び出しているdot_product()関数(リスト1)をSIMD命令を有効活用して最適化すると、さらに速度を向上できます。

リスト1の最後にあるdot_product()関数をコンパイルすると、SSEなどのベクトル演算命令を使用せずに、FP32のデータ1個ずつに対して演算を行うスカラー命令を多数使用するコードが生成されました。このときの演算の流れを図示すると図1(a)のようになります。a[0] × b[0], ..., a[3] × b[3]の4

