第3部

第2章

画像フィルタの処理が2.5～5倍に…
高速化の一部始終を紹介

キャッシュを意識したコード最適化テクニック

高野 修一

● プログラムを改善してキャッシュの効果を確認

本稿ではC＋＋のプログラムの処理時間を実測することで，キャッシュの効果を確認します．基本的なキャッシュの効果に集中するため，マルチスレッド・プログラミングは扱わず，シングル・スレッドにおけるCPUの振る舞いのみを対象とします．

画像処理（ガウシアン・フィルタ）を題材とします．筆者は，MacBook Pro（2021，M1 Max，アップル）を用いて実験しましたが，キャッシュを搭載するどのCPUに関しても同じ考え方が通用するはずです．

最適化で検討すること

プログラムを最適化しようとしたとき，どのようなことを気にするべきでしょうか．最初に行うべきは，問題をより少ない計算量で解ける方法に置き換え，根本的に処理全体を軽くするようなアルゴリズム・レベルでの改善です．

同じアルゴリズムでもプログラムの書き方は複数あり，実行効率の良い方法を選びます．実装の効率化では，無駄な処理を省いたり，データの配置を工夫したりして，最小限の命令で実行できるように組み替えるのが基本です．しかし，それだけではうまくいかないこともあります．

● キャッシュを意識した最適化を行う

近年のコンピュータ・システムではメモリに関するボトルネックが大きく，それを隠蔽（いんぺい）するためのキャッシュ・システムがCPUに組み込まれています．キャッシュの性質を意識して組んだプログラムと，そうでないプログラムとの間では，数倍の性能差が出ることも珍しくありません．

● 最適化に集中するため画像はシンプルに扱う

▶画像データの定義

本稿では本質的な問題に集中するため，画像の各画素をfloatのモノクロ値とします．値の範囲は0～1とし，**リスト1**に示すような構造体で管理します．

リスト1　本稿で使う画像を定義したクラス

```
struct Image
{
  size_t w = 0, h = 0, stride = 0;
  std::vector<float, AlignedAllocator<float,
                                        64>> pix;
  // AlignedAllocator は別途定義されるアライン付き Allocator

  void alloc(size_t W, size_t H, size_t Stride = 0)
  {
    w = W; h = H; stride = Stride ? Stride : w;
    pix.assign(stride * h, 0.0f);
  }
};
```

画像は2次元のデータですが，メモリ上では行をつなげて1次元の連続した領域として保持するのが一般的です．この構造体では，pixに全ての画素値を1列に並べて格納し，座標(x, y)の画素には，pix[y * stride + x]としてアクセスします．

ここでwではなくstrideという別の変数を使っています．これにはx方向の画素数に特定の数を足した数値を格納してあります．末尾に余分な領域を追加してでも，画像の幅とは独立してメモリ上での1行の幅を指定できると便利なことがあるためです．

▶端数処理は簡易な方法で対処

この手の画像処理では画像の範囲を参照するため，端付近での境界処理が必要です．本稿で扱うプログラムは，入力画像の周囲に読み込み範囲分の余白をあらかじめ設けておくことで，その問題を無視できるようにします（**図1**）．

ガウシアン・フィルタの一般的な実装

● ガウシアン・フィルタの定義

比較的シンプルで実用的な画像処理の一例としてガウシアン・フィルタを取り上げます．ガウシアン・フィルタは，入力画像$I[x, y]$に対して，標準偏差σに関する2次元のガウス関数，

$$G(x, y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \cdots\cdots\cdots\cdots\cdots\cdots (1)$$