第3章 SIMDやレジスタをうまく活用して CPU比29倍の高速化!

テンプレート・マッチングでGPUとCPUの速度を比較する

近藤 鯛貴，竹田 大将，佐藤 裕幸

$$ZNCC = \frac{MN\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} I(i,j)T(i,j) - \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} I(i,j) \times \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} T(i,j)}{\sqrt{(MN\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} I(i,j)^2 - (\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} I(i,j))^2)(MN\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} T(i,j)^2 - (\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} T(i,j))^2)}}$$

図1　テンプレート・マッチングZNCCの式

本章ではGPUによる処理の応用としてテンプレート・マッチングを実装します．CPUだけで実行した場合とGPUだけで実行した場合との計算時間を比較します．

比較する処理は テンプレート・マッチング

● 製造現場の品質管理やロボットの位置合わせに使われる処理

テンプレート・マッチングとは入力画像からテンプレートと呼ばれる小さな画像と類似する位置を探索するアルゴリズムです．古くから製造における傷や欠品などを確認する品質管理やロボットの位置合わせに用いられています．さまざまな手法がありますが，本稿では代表的な手法のZNCC（Zero-means Normalized Cross-Correlation）を実装します．数学的な理論は割愛し，GPUの特性とアルゴリズムの計算特性からどのようにすれば処理が速くなるかにフォーカスして解説します．

GPU実装については，それぞれ最適化の度合いが異なる3ステップに分けて説明します．ステップ1ではGPUによるマルチスレッド化を行います．ステップ2ではSIMDによる高速化を試します．ステップ3ではレジスタを積極的に使う最適化を行います．

ZNCCの式を**図1**に示します．$I(i, j)$は入力画像の画素値，$T(i, j)$はテンプレート画像の画素値です．M，Nがテンプレート画像サイズです．ZNCCはアルゴリズムの改良などさまざまな高速化の研究がなされています．今回はもともとの計算式から共分散の公式で簡略化したものを実装します[1]．演算結果として得られる各配列要素には－1.0～＋1.0のデータが格納され，1に最も近い数値（配列の中の最大値）が格納されている位置がテンプレート画像と最も類似している座標点となります．

● 筆者提供のサンプルで試す

本章でも第1章で使ったリポジトリのデータを使います．リポジトリのtemplate_matching/にコードが入っています．makeを実行するだけで全てのコードがコンパイルされます．src/host_l0.cppがホスト側のコードです．GPUのコードはsrc/step1～src/step3のディレクトリに入っています．各ステップでホスト側のコードは同じものを使っています．ホスト側ではスレッド数しか違わないので，コンパイル時のマクロでスレッド数を決定しています．

試しに最も処理が速いステップ3を実行してみます．実行時にコマンドラインの引数によって入力画像とテンプレート画像を指定します．リポジトリのimagesに大小2パターンの猫の画像が入っています．どちらもテンプレート画像は猫の左目を切り出しています．小さい方のサイズは，入力画像が128×128（mini_cat.jpg），テンプレート画像が32×32（mini_cat_eye.jpg）です．大きい方のサイズは入力画像が1920×1080（cat.jpg），テンプレート画像が320×320（cat_eye.jpg）です．1920×1080の方は，非常に計算時間が長いです．試してみたい方には小さい画像をおすすめします．

```
./step3.l0.skl ../images/mini_cat.
    jpg ../images/mini_cat_eye.jpg↵
```

結果画像としてtemplate_matching/にout.jpgが生成されます．入力画像を**図2**に，テンプレート画像を**図3**に示します．