パターンマッチングアクセラレータのソフトウェア高速化手法の研究

愛媛大学 工学部工学科コンピュータ科学コース
計算機・ソフトウェアシステム研究室
2520245K　髙橋 佳汰

概要

- 研究背景
- Set Operating Processor（SOP）
- 機能検証ソフトの処理フロー
- 研究目的・目標
- ソフトウェア全体構造
- 処理時間計測結果（改良前コード）
- ボトルネック特定・原因
- ボトルネック改良
- 処理時間計測結果（コード改良後）
- 機能選定
- 処理時間計測結果（コード改良・機能選定後）
- 考察
- まとめ・今後の課題

研究背景

- パターンマッチングが幅広い分野で利用
  - 自動運転や工場検査など

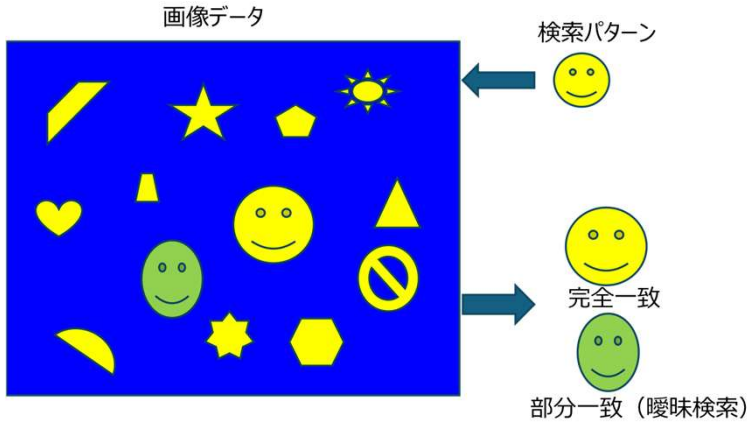

図1. パターンマッチングの例

- エッジAIデバイスではリアルタイム性と低消費電力の両立が必要
  - CPUではフォンノイマンボトルネックが顕在化，両立困難
- Set Operating Processor（SOP）は低消費電力で高速なパターンマッチングが可能

Set Operating Processor (SOP)

- 高速画像パターンマッチングアクセラレータ
  - ニアメモリ型アーキテクチャ
- ピクセルモジュールを行列状に配置
  - 画像サイズ分用意
  - 並列動作により高速パターンマッチングが可能
- 画素の色データと基準点からの相対位置情報に基づくマッチング

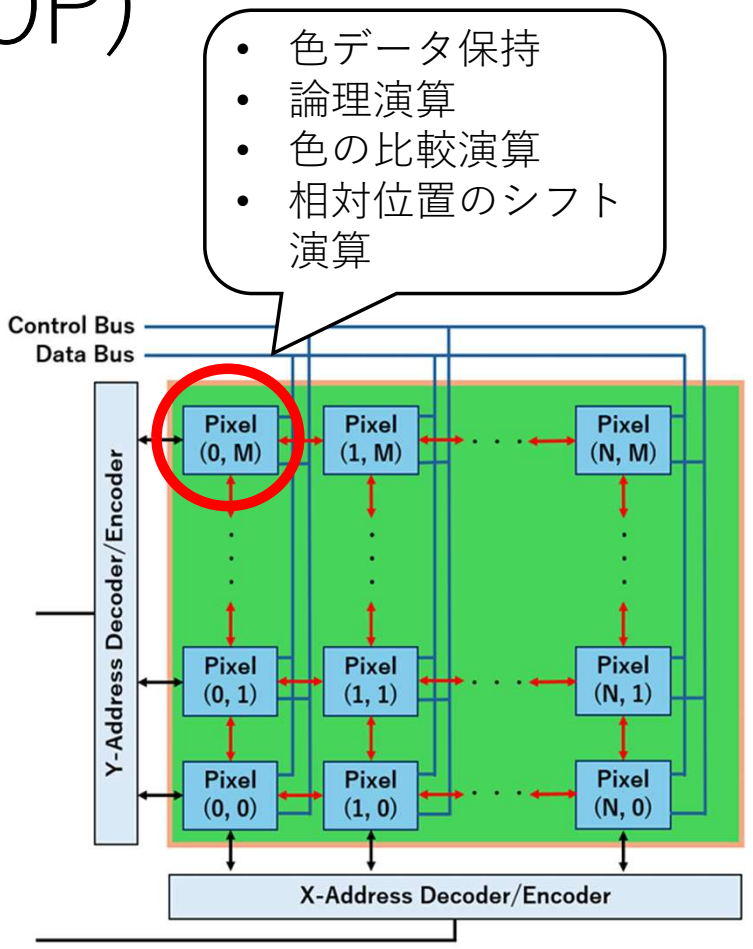

図2. SOPの構造

機能検証ソフトの処理フロー

- 画像は180×180にリサイズ
- 全ピクセルのRGB値を取得し画像データ生成
- 特徴量ファイルから特徴設定

- SOPの機能検証ソフトが存在（Python）
  - SOP制御・命令列生成・結果取得
- しかしパターンマッチング処理が遅い課題が存在
  - SOPの高速性が引き出せない
  - 実運用が困難

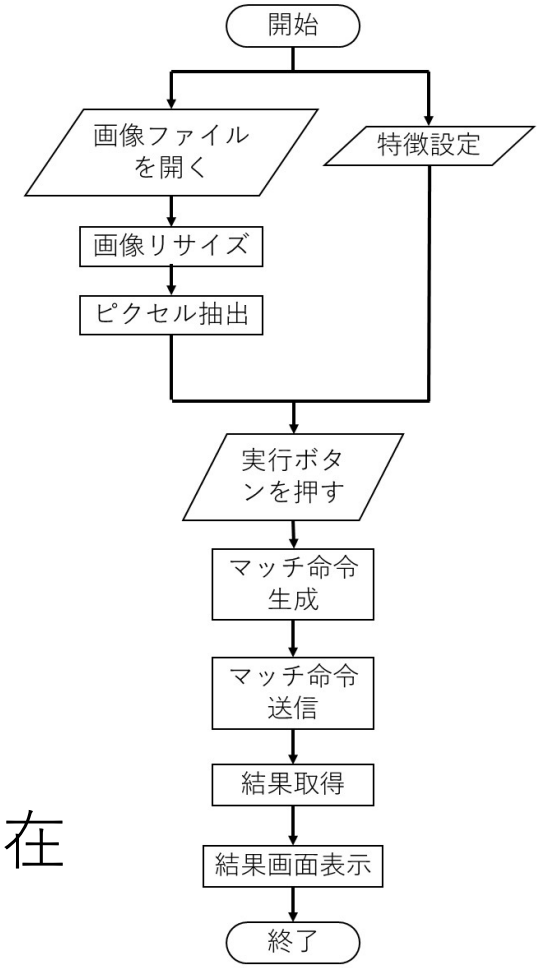

図3. 機能検証ソフトによるパターンマッチング処理のフローチャート

研究目的・目標

目的：SOPのパターンマッチングシステムの高速化
目標：機能検証ソフトのボトルネックを特定・改良

進め方：
step1:ソフトウェアの構造把握，ボトルネック特定
step2:ボトルネックに対する高速化手法の提案
step3:改良法の実装，ソフトウェア再作成
step4:処理時間計測

ソフトウェア全体構造

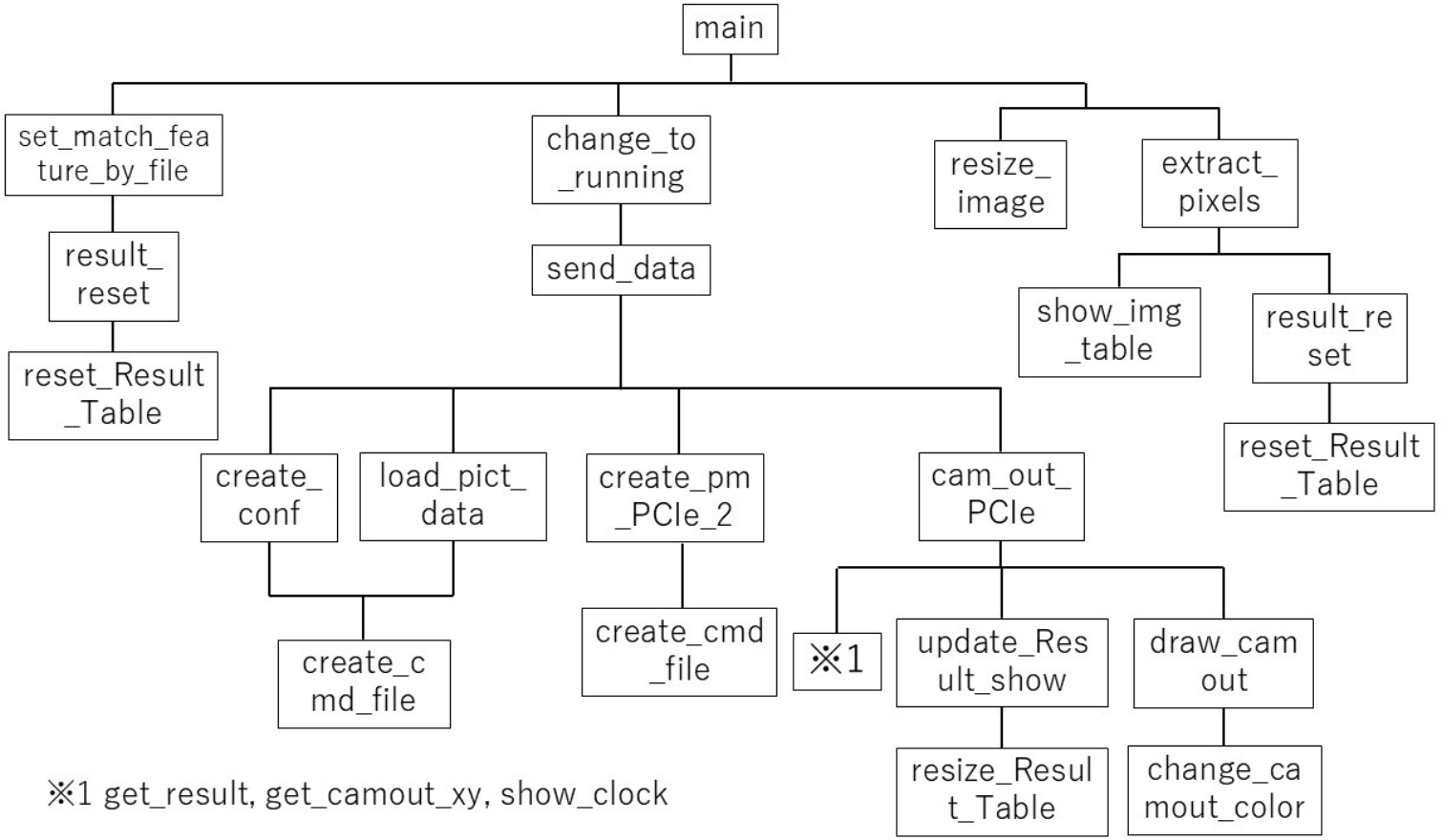

図4. 機能検証ソフト内の関数の階層構造

処理時間計測結果（改良前コード）

各関数に対して処理時間計測

表1. 改良前の各関数の処理時間

関数名	平均値[ms]	最悪値[ms]	p95[ms]
resize_image	0.384	0.484	0.450
extract_pixels （GUI描画なし）	214.268	266.496	221.391
set_match_feature_by_file （GUI描画なし）	18.618	61.089	18.777
create_conf	0.222	0.368	0.307
load_pict_data	0.254	0.462	0.344
create_pm_PCIe_2	2.317	3.362	2.796
get_result	81.274	109.813	89.638

ボトルネック特定

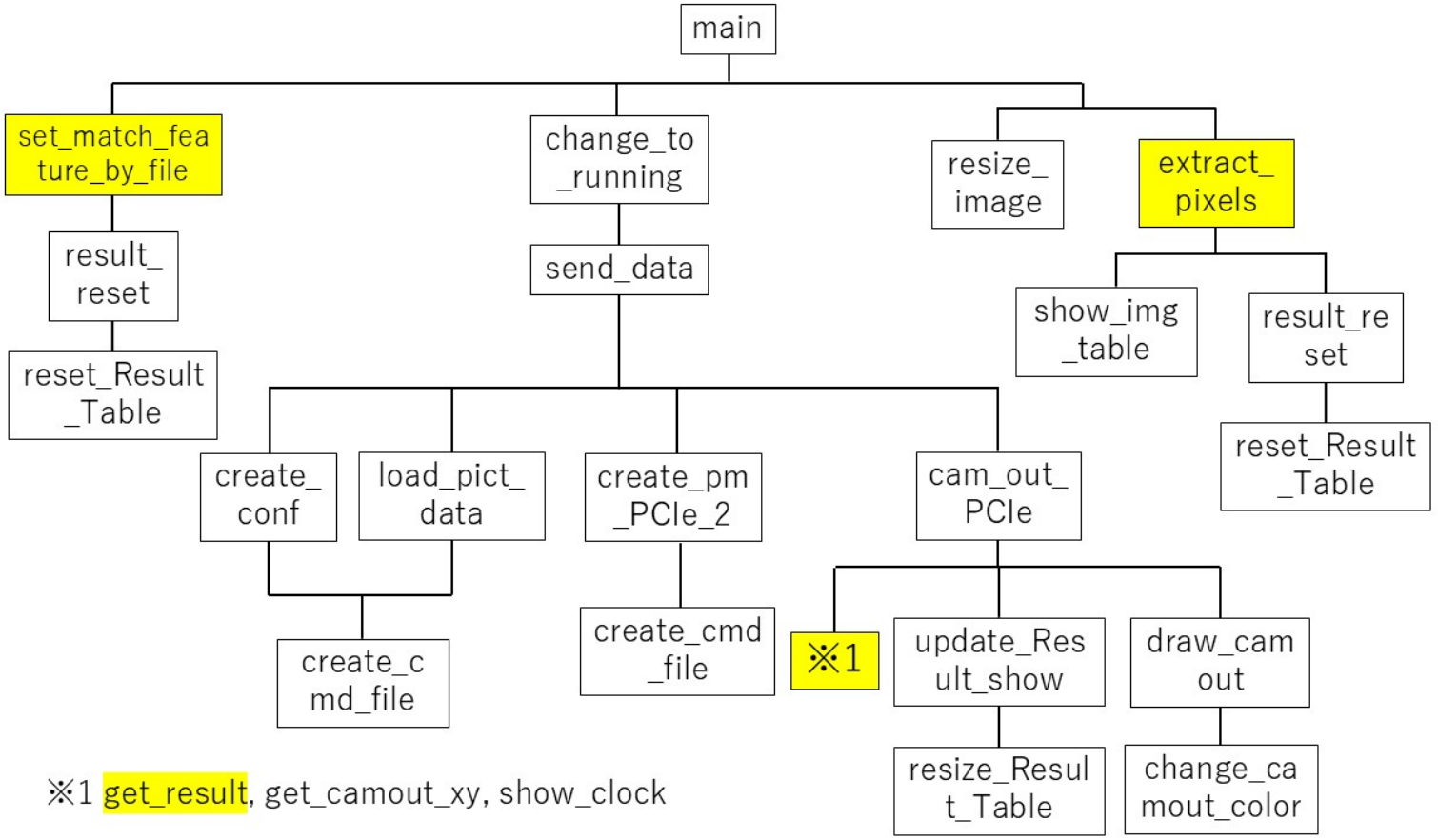

図5. 機能検証ソフト内の関数の階層構造（ボトルネック強調）

ボトルネックの原因

- 画素抽出処理（extract_pixels）
  - for文により1画素ずつ抽出
- 特徴量ファイル（Excel）読み込み（set_match_feature_by_file）
  - read_excelにより各シート（3枚）を読み込む
- SOPからの結果取得処理（get_result）
  - シェルスクリプトの読み込み・実行に時間がかかる

ボトルネック改良

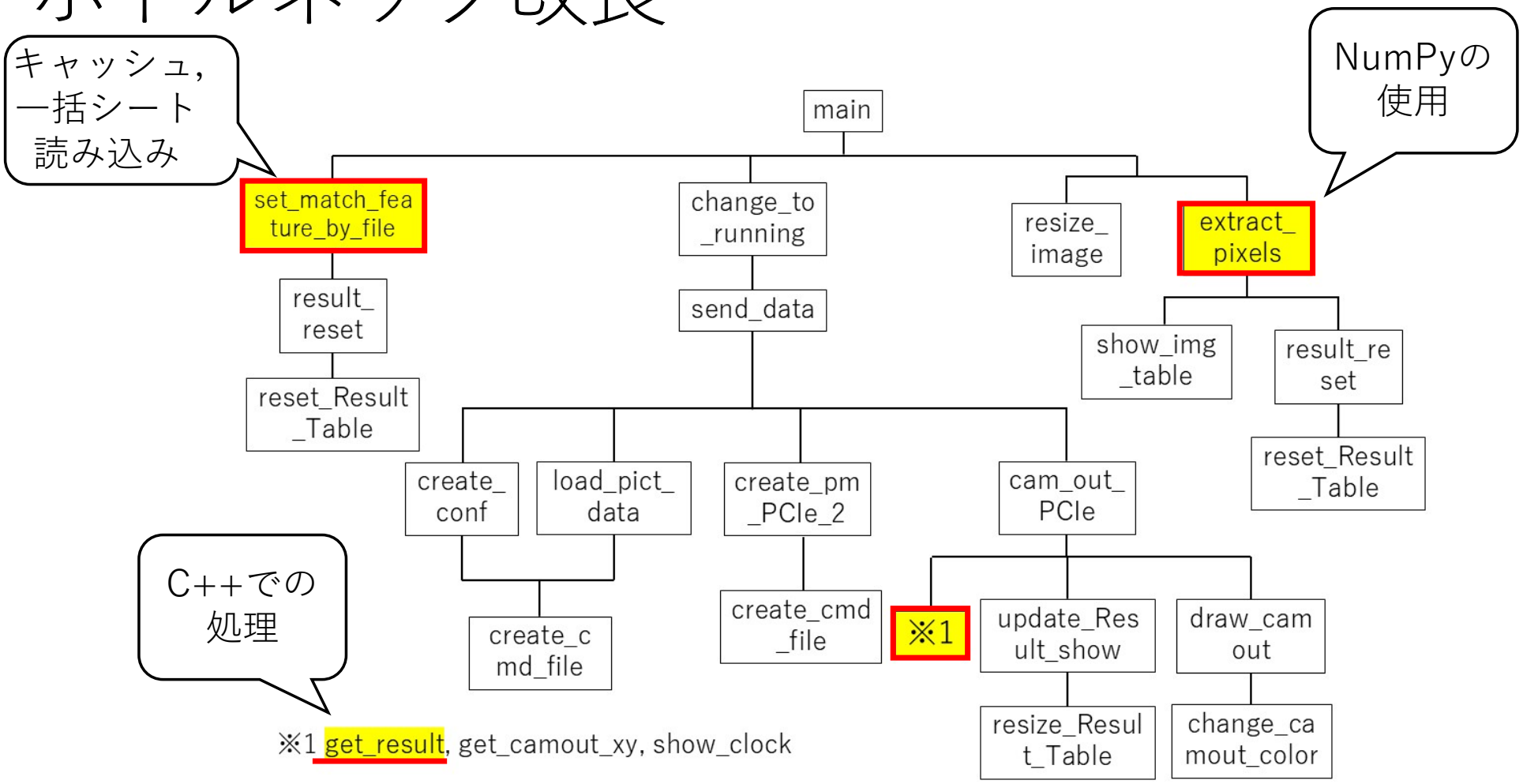

図6. 機能検証ソフト内の関数の階層構造（ボトルネック改良案）

処理時間計測結果（コード改良後）

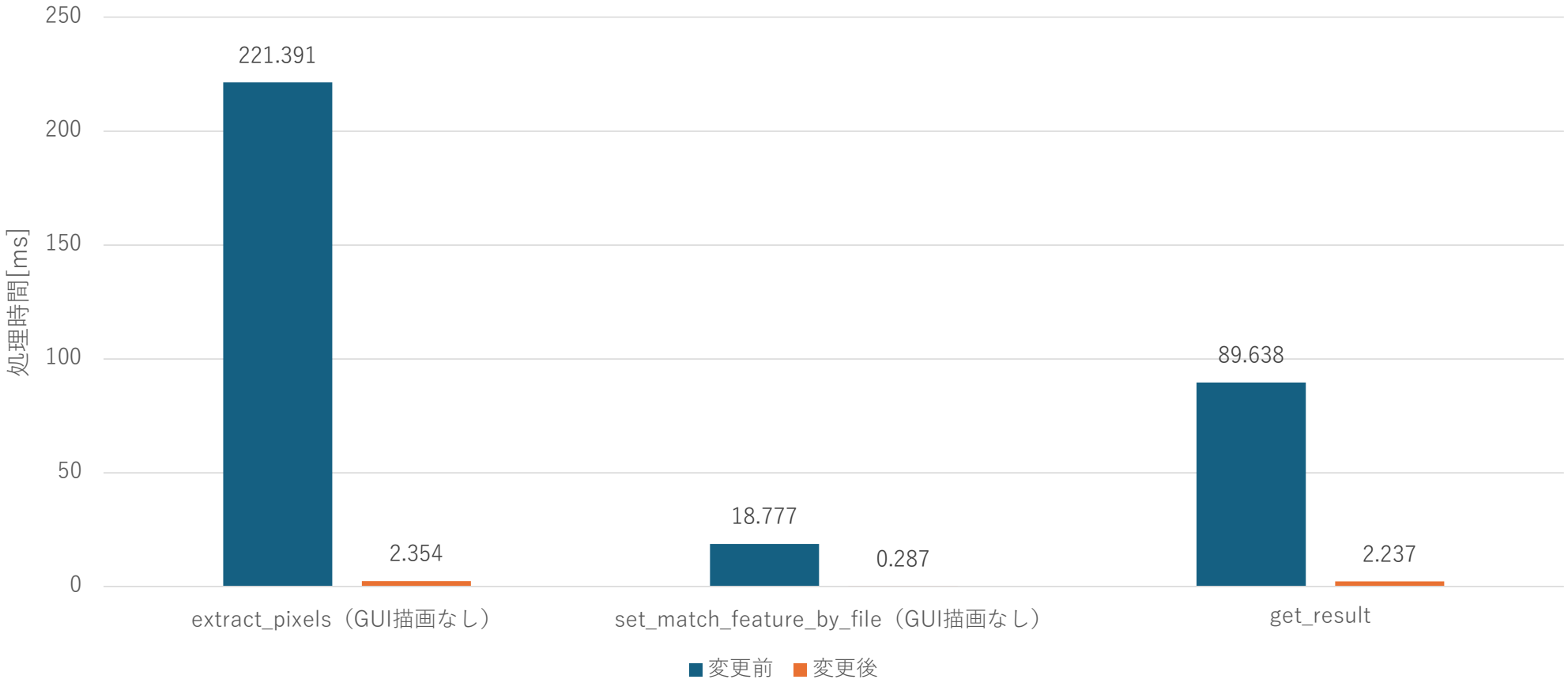

図7. ボトルネック関数の改良前と改良後の処理時間（p95）

機能選定

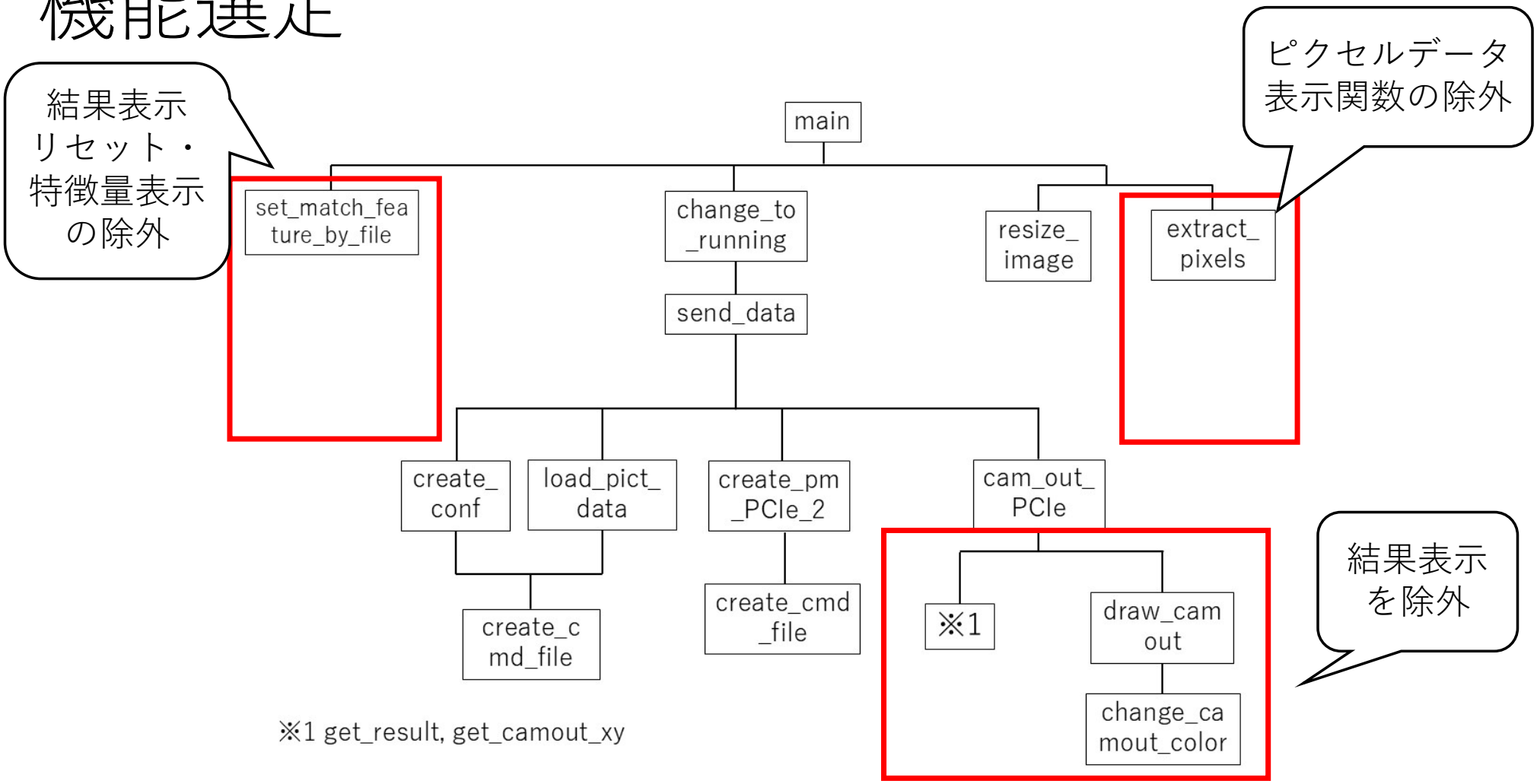

図8. 機能選定後のソフト内の関数の階層構造

処理時間計測結果（コード改良・機能選定後）

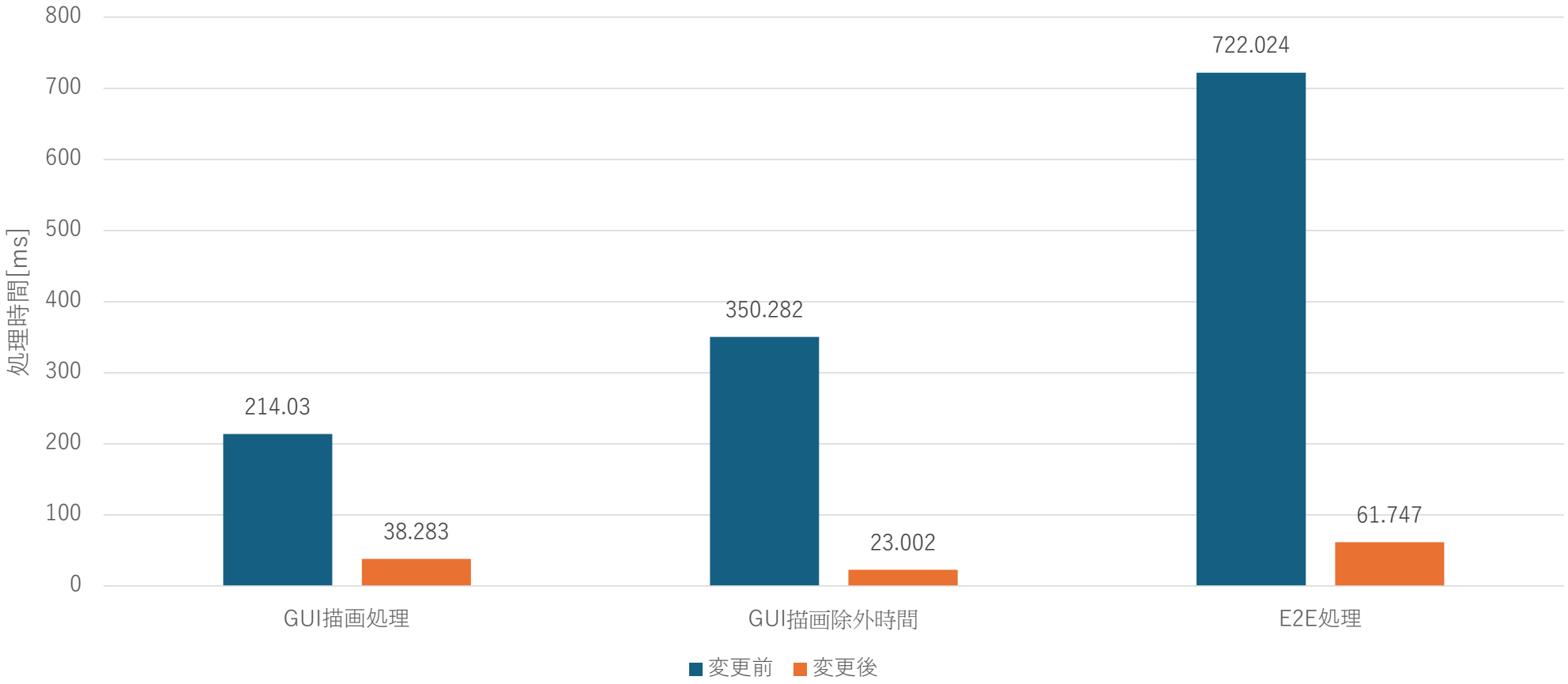

図9. 改良前とコード改良及び機能選定後の全体処理時間（p95）

考察

- 命令列送信までの処理に高速化の余地あり
  - パターンマッチング中に次回の画像リサイズから命令列生成までを行う（マルチスレッド）
  - get_resultと同様にC++によるAPI化
- GUI描画処理では結果描画処理が新たなボトルネックになりうる
  - 結果ごとに描画処理を呼び出す

まとめ・今後の課題

- ソフトウェア構造の理解と処理時間計測からボトルネック特定
- コード改良及び機能選定により処理時間を削減した
  - GUI描画除外時間（p95）は350.282 msから23.002 ms
  - E2E処理時間（p95）は722.024 msから61.747 ms
- 今後はマルチスレッドと命令列送信処理のAPI化により高速化
- GUI描画処理のボトルネックである結果描画処理の高速化