一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2021-58,DC2021-92(2022-03)

ORB-SLAM3 のローカルマッピングの並列化とコア割り当て手法の提案

山本　一貴† 長ヶ部拓吾† 小池穂乃花† 川角　冬馬† 藤田　一輝†
北村　俊明† 川島　慧大†† 納富　昭†† 木村　貞弘††† 木村　啓二†
笠原　博徳†

† 早稲田大学
†† オスカーテクノロジー株式会社
††† 株式会社 エヌエスアイテクス
E-mail: †kyamamoto@kasahara.cs.waseda.ac.jp

あらまし　ロボットや自動車の自律制御のために，周囲の情報から環境地図を作成し自己位置を推定する SLAM（Simultaneous Localization and Mapping）処理が必要とされる．SLAM は主にトラッキング，ローカルマッピング，ループグロージングの 3 つの処理から構成される．このうちローカルマッピングは，実行コストが大きい一方，疎行列やグラフ構造などを扱うことから効率の良い並列化が難しい．そのため本稿では，オープンソースの SLAM 実装である ORB-SLAM3 を対象として，データ構造の特徴を生かした並列化方法を提案する．またトラッキングの特徴点抽出処理も合わせて並列化した場合，発生する多くのスレッドができるだけ同一コアで競合しないような割り当て手法を報告する．これらの提案手法を適用して両処理を並列化したところ，Intel サーバ上ではトラッキングで 1.40 倍，ローカルマッピングで 1.35 倍の速度向上が同時に得られた．また，NVIDIA Jetson 上でも，トラッキングで 1.22 倍，ローカルマッピングで 1.26 倍の速度向上が得られ，提案手法がコア数の少ない組込み環境にも有効であることが確認できた．
キーワード　SLAM（Simultaneous Localization and Mapping），並列化，OpenMP

LocalMapping Parallelization and CPU Allocation Method on ORB-SLAM3

Kazuki YAMAMOTO†, Takugo OSAKABE†, Honoka KOIKE†, Tohma KAWASUMI†, Kazuki FUJITA†, Toshiaki KITAMURA†, Akihiro KAWASHIMA††, Akira NODOMI††, Sadahiro KIMURA†††, Keiji KIMURA†, and Hironori KASAHARA†

† Waseda University
†† Oscar Technology Corporation
††† NSITEXE,Inc.
E-mail: †kyamamoto@kasahara.cs.waseda.ac.jp

1. はじめに

ロボットや自動車の自律制御では，未知な環境においても目的地までの経路計画や障害物の回避を適切に行うために，SLAM と呼ばれる走行時に地図を作成しながら自己位置を推定する技術が用いられる．SLAM の中でも特に，Visual SLAM と呼ばれるカメラセンサーを使った SLAM は，比較的安価にシステムを構築できる利点があり広く用いられている．SLAM はその使用環境から組み込みシステムでの処理が必要であるが，その一方で画像処理やグラフ処理などの計算負荷の高い処理を扱うため，その高速化が求められている．

オープンソースの Visual SLAM の実装の一つに ORB-SLAM3 [1] がある．ORB-SLAM3 は，3 つのスレッドが並列に動作し，各スレッドでトラッキング，ローカルマッピング，ループクロージングと呼ばれる処理が行われる．トラッキングでは，カメラ画像から特徴点を抽出した上で，マップ上での自己位置を推定する．また，ローカルマッピングではトラッキングから挿入されたフレーム（キーフレーム）を用いて，マップの作成を行う．

以上の 2 つの処理は，走行中の多くの時間を費やして処理を行う一方，ループクロージングは同一地点を再度訪れた時のみ，ループとなるように軌道を接続し，マップを補正する役割を持つため，この主要な処理の実行頻度は小さい．そのため，安定したシステムの動作を実現するには，トラッキング及びローカルマッピングの，1 フレームあたりの処理時間の削減が重要となり、本稿ではこれらの処理の並列化による高速化を提案する．

トラッキングでは，特徴点を抽出する処理や特徴点記述子の作成にかかるコストが大きく，本処理が並列化の対象となる．

ローカルマッピングでは，トラッキングで抽出された特徴点とフレームの位置を元に作られる地図上のポイント生成や，地図上の 3 次元の点群やカメラ位置を補正するローカルバンドル調整がボトルネックとなる．これらの処理は，フレーム間や特徴点間に並列性があるものの，以下の特徴から一部処理を逐次で行う必要があり，並列化性能を引き出すのが難しい．

- 地図上の点群や点群からできるグラフの頂点・エッジが，複数のクラス・ライブラリから参照可能な共有領域に置かれる．
- グラフの探索や疎行列計算が行われる (グラフ最適化).

これらの特徴をふまえ，本稿では OpenMP による並列化とリダクションループを併用したローカルマッピングの並列化方法を提案する．

また，ORB-SLAM3 ではトラッキング・ローカルマッピング・ループクロージングの 3 つの処理を並列に動作させる上，各処理の内部を並列化すると多くのスレッドが生成される．そのため，OS によるスレッドのコア割り当てでは，トラッキング・ローカルマッピング両方を並列化した場合には並列化性能を十分に引き出せないという問題がある．そこで本稿では，OpenMP ランタイムを拡張し，トラッキング・ローカルマッピングで生成されるスレッドごとに実行コアを指定できるようにすることで，各処理の並列化性能を引き出すことを可能とした．

以下，第 2 節では ORB-SLAM3 の概要と高速化の関連研究について，第 3 節ではトラッキング処理の並列化手法，第 4 節ではローカルマッピング処理の並列化手法について説明する．また，第 5 節ではトラッキング・ローカルマッピング内部の各並列化スレッドを指定のコアに割り当てるための OpenMP ランタイムの実装の拡張と，適切なコア割り当てについて説明する．そして第 6 節では，第 3,4 節で述べる並列化によるトラッキング・ローカルマッピングの高速化の評価と第 5 節で述べるコア割り当て手法による効果について，最後に第 7 節でまとめる．

2. 関 連 研 究

2.1 ORB-SLAM [2] の概要

ORB-SLAM の構成を図 1 に示す．トラッキングでは，まずフレーム画像から FAST(Features from Accelerated Segment Test) [3] と呼ばれる手法を用いて，地図作成のために重要な物体の輪郭や端点など（特徴点）を検出する．次に，特徴点間のマッチング作業を高速に行うため，特徴量を 2 値ベクトルに圧縮した BRIEF(Binary Robust Independent Elementary Features) [4] と呼ばれる特徴点記述子を生成する．そして，前フレームとの比較によるカメラ位置の推定や，地図内でのカメラの位置の補正

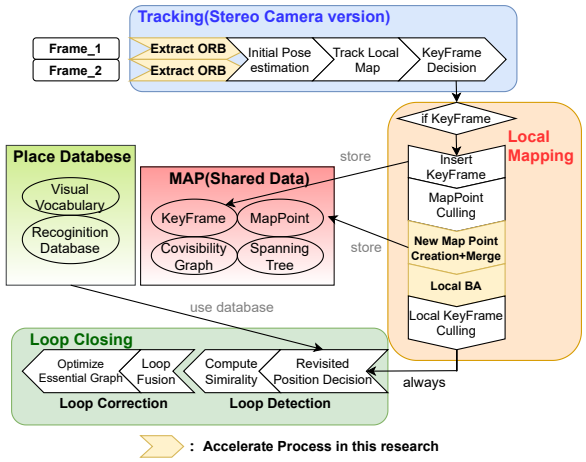


図 1: ORB-SLAM のシステム構成 [2]

を行い，最終的にフレーム内に地図に登録済みの特徴点がそれほど多くない場合は，キーフレームとしてローカルマッピングに挿入し，地図の作成に利用する．

ローカルマッピングでは，カメラの位置とフレーム内の特徴点からなる地図を作成する．断続的にトラッキングスレッドからキーフレームが挿入された時のみ行われ，キーフレームの位置と内部の特徴点からなるマップポイントの生成・統合，およびローカルバンドル調整と呼ばれる地図上の点群を補正する最適化を行う．最適化には，グラフ最適化のライブラリ g2o [5] を使っており，また内部の行列演算には C++ の線形代数演算のライブラリ Eigen [6] を使用している．

ループクロージングでは，マップ内の軌道がループになるかどうかを検出する．そして，ループが検出された場合には，同じフレーム位置の軌道を繋いでループにした上で，バンドル調整により地図上の全てのカメラ位置を補正する．

2.2 ORB-SLAM3 [1]

ORB-SLAM3 は，最初に公開された単眼カメラ版の ORB-SLAM を拡張したもので，ステレオカメラや RGB-D カメラに加えて，IMU センサーを用いた SLAM 処理が可能である．また，OpenCV の静的行列の利用などにより，トラッキングやローカルマッピングの 1 フレームあたりの時間を減らすためのソースコードの最適化も行われている．

2.3 SLAM の高速化

ORB-SLAM のローカルマッピング処理自体を高速化の対象としている先行研究はほとんどないものの，一般的なバンドル調整処理を高速化している研究はある．バンドル調整内の g2o ライブラリは，公開されている実装でも OpenMP 並列化が行われているが，本稿ではロック処理を排除する並列化により，さらなる速度向上を実現している（6.3 節）．また，Adaskit team 等 [7] や Jie 等 [8] は，GPU を用いた高速化を提案している．

ORB-SLAM のトラッキング処理の高速化を行っている研究として，GPU を使った Chen らの研究 [9]，Li らの研究 [10] や Aldegheri らの研究 [11] があげられる．これらの研究は，本研究と同じ特徴点抽出や記述子生成を高速化対象として，GPU によ

る高速化を行っている上，特に Li らの研究では，画像ピラミッドの各層の並列性を生かした並列化に着目している．本研究では，CPU を用いて同様の層間の並列性を利用して，OpenMP 並列化を行った．また，Mamri らの研究 [12] では，GPU に加えて FPGA を用いて並列化を行っているが，高速化対象が特徴点抽出後のカメラ位置推定である点で異なる．

一方，各処理を並列化する手法は提案されているが，並列化した両処理を SLAM 処理に統合して総合的に評価した例はない．

3. トラッキングの並列化

本節では，ORB-SLAM3 のトラッキング処理の OpenMP による並列化について述べる．トラッキング処理のボトルネックとなる特徴点抽出と特徴点記述子生成を対象に並列化を行った．

3.1 特徴点抽出の並列化

ORB-SLAM3 では，できるだけ高い精度で特徴点を抽出するため，入力データは異なるサイズ・解像度の画像を積み重ねた 8 段のピラミッド型に変形される．具体的には，OpenCV に実装されている `resize()` 関数を用いて，前の段のデータを幾何学変換により縮小しながら次の段のデータを作成している．

次に，画像ピラミッドの各段のデータに対して `FAST()` を適用し，周辺領域に対して中心領域の画素値が大きい画素を，特徴点として抽出する．この抽出処理において，各段が他の段のデータや抽出された特徴点を参照することはない．そのため，各段での特徴点の抽出操作に対して並列化を行った．

3.2 特徴点記述子生成の並列化

抽出された特徴点の情報は特徴点記述子生成によって特徴点記述子に格納される．この特徴点記述子は，独立した各段の特徴点を元に計算され，他の段の特徴点を参照することはない．そのため，各段での特徴点記述子の計算に対して並列化を行った．ただし，生成された特徴点記述子はピラミッド画像全体で管理し，全ての段で同じ 1 つのデータ構造に格納されることから，リダクションを用いて追加処理の逐次化を行った．

4. ローカルマッピングの高速化

本節では，ORB-SLAM3 のローカルマッピングの実装に対する OpenMP による並列化手法を提案する．並列化は実行時間のボトルネックである，地図上のマップポイント生成と近傍のポイント統合，およびローカルバンドル調整を対象に行った．

4.1 地図上のマップポイント生成・統合部分の並列化

マップポイントの生成は、主に次の手順により行う．これは，`CreateNewMapPoints()` 関数の処理に該当する．

（1） 直近に挿入されたキーフレームの近傍のキーフレーム群を探索し，フレーム内から地図に未登録の特徴点を取得

（2） 取得した特徴点と直近のキーフレーム内の特徴点同士で比較し，一致する点を「マップポイント」として地図に追加

ここで，書き込み先の地図上の新しいポイントに対して，近傍の他のキーフレームが参照することはない．そのため，近傍にある複数のキーフレーム間の処理に対して並列化を行った．ただし，地図自体や直近のキーフレームのデータ構造への書き込み処理は，並列化により複数のフレームから同時に書き込まれないように，critical section を指定して逐次化を行っている．

```
<変更前・OpenMP 並列化不可>
for(list<MapPoint*>::iterator lit=lLocalMapPoints.begin(),
  lend=lLocalMapPoints.end(); lit!=lend; lit++)
    MapPoint* mp = *lit;
<変更後>
for(list<MapPoint*>::iterator lit=lLocalMapPoints.begin(),
  lend=lLocalMapPoints.end(); lit!=lend; lit++)
    {array_mp_iterator[i] = lit; i++;}
#pragma omp parallel for
for(i=0;i< lLocalMapPoints.size();i++)
    MapPoint* mp = *array_mp_iterator[i];
```

図 2: List・set イテレータループの並列化

また近傍のポイント統合では，再度近傍のキーフレームを探索し，その中の特徴点と直近のフレーム上の新しいポイントの距離が近い 2 点を対応づける．この処理は，`SearchInNeighbors()` 関数に該当する．近傍のキーフレーム間の並列性を利用することで，ポイント生成と同様に探索ループの並列化を行った．

4.2 ローカルバンドル調整の高速化

ローカルバンドル調整では，直近とその近傍のキーフレーム，およびその中の特徴点を対象に，地図上の 3 次元座標を補正する．位置の補正では，「カメラ上からみた特徴点と他のフレームの 2 次元座標」と「地図上の 3 次元の特徴点を 2 次元に投影した際の座標」の誤差を最小化する．これを再投影誤差とよぶ．そして，誤差の最小化のために，フレームと特徴点を頂点，各フレームに対する特徴点・他のフレームとの再投影誤差をエッジとするグラフを構築し，非線形の最小二乗法を用いてグラフの最適化を行う [13]．この処理の高速化手法を，以下の第 4.2.1 節〜第 4.2.3 節に示す．

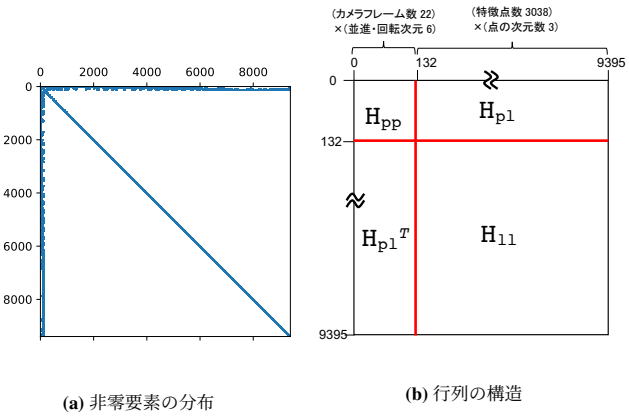
4.2.1 C++ の List・set 型のイテレータループの並列化

ORB-SLAM3 では，近傍のキーフレーム内のマップポイントを C++ の List 型のデータ構造で管理している．また，複数のフレームに含まれる同じマップポイントを重複して登録しないように，グラフの頂点を C++ の set で管理している．しかし，これらのイテレータはランダムアクセスができないため，イテレータを使ったマップポイントやグラフの探索ループが並列化可能であっても，そのままでは OpenMP による並列化ができない．そこで，図 2 のようにイテレータを配列に代入した後，配列アクセスのループに変更することで，並列化を可能とした．

4.2.2 リダクションを用いたループの並列化

グラフの頂点・エッジは，複数のライブラリから参照可能な領域に置かれるため，追加処理を逐次で行う必要がある．そこで，各スレッドで別々の配列に追加して並列化した上で，リダクションで 1 つのデータ構造へ集約することで高速化を行った．

また，グラフの最適化で行われる疎行列積計算やグラフの探索についてもリダクションを用いて並列化を行った．疎行列積計算は，誤差関数を最小化するために必要な線形方程式の係数行列を生成する処理で行われる．グラフ最適化では，まずグラフの頂点とエッジからヘッセ行列とよばれる行列を生成する．



(a) 非零要素の分布　　**(b)** 行列の構造

図 3: ローカルマッピングのローカルバンドル調整で生成されるヘッセ行列の例 (KITTI0 番のデータセット [14]・ステレオカメラ版を使用)

小行列が多数集まったブロック行列で，行・列ブロック数はカメラフレームと特徴点数の合計（頂点数）に等しい．小行列の 1 辺サイズは，カメラフレームのパラメータの場合は並進と回転座標の次元の合計，特徴点パラメータの場合は 3 である (3 次元のため)．

図 3 は，ORB-SLAM3 のローカルバンドル調整で生成されるヘッセ行列の例である．このとき，(a) のように行列はサイズが大きい上スパース度が高いことから，行列全体を線形ソルバーに通すと計算時間が増大してしまう．そのため g2o では，(b) のような H_{pp}，H_{pl}，H_{ll} の各行列を対象に疎行列積を計算し，行列サイズを H_{pp} まで抑えた行列（シューア補行列）を線形ソルバーに通すことで，演算時間を短縮している [5].

疎行列積計算では，行列サイズが H_{pl} と等しい横長の行列 (A とする) と，サイズが H_{pl}^T と等しい縦長の行列 (B とする) の行列積（結果の行列を C とする）を計算する．このとき，g2o の実装では，まず行列 A の j 列目に含まれる先頭の非零ブロック要素 $A_{i,j}$ と行列 B の j 行目の非零ブロック要素 $B_{j,0\ldots N-1}$ で乗算を行い，結果 C の i 行目の途中結果を生成する（$partC_{i,0\ldots N-1}$）．次に行列 A の j 列目の全ての非零ブロックに対して乗算を繰り返すことで，行列積全体の途中結果を生成する．そして，行列 A の全ての列の非零要素に対してこの処理を繰り返し，結果を足すことで最終的な行列積を生成する．ここで，公開されている g2o の OpenMP 実装では，行列 A の列方向で並列化しているが，複数のスレッドから同時に結果 C の要素に書き込まないように，ロックをかけた上で値を更新していたため，並列化時に速度が鈍化していた．そのため本研究では，H_{pp} のサイズが小さい SLAM の特徴を生かして，各スレッドで別々に途中結果の行列を作成して並列化した上で，リダクションにより最終的な結果行列 C を生成させることでロック処理を排除した．

グラフの探索では，異なるエッジが同じ頂点につながっている場合，各エッジから並列に同じ頂点の値を変更することができない．公開されている g2o の実装では，ロックを用いることで頂点の値の更新を逐次化していたが，各スレッドで異なる配列に頂点の値を書き込むことで並列化を行った．そして，リダクションを用いることで，頂点に最終結果を代入している．

4.2.3 密行列用のコレスキーソルバーを用いた高速化

第 4.2.2 節の通り，ソルバー処理に通す行列サイズは極めて小さい上，スパース度が低く非零要素の割合が大きい．そのため，密行列用のコレスキーソルバーを利用するよう変更した．

5. ORB-SLAM3 の各処理のコア割り当て手法

トラッキング及びローカルマッピング内部の処理を並列化すると，同時に多くのスレッドが生成される．そのため，できるだけ複数スレッドの処理が同一コアで競合しないようなコア割り当てが重要である．本節では，ステレオカメラのバージョンに対して，両処理を並列化した場合の各スレッドの理想的なコアの割り当て方について述べる．そして，それを実現する際に発生した現状の OpenMP のコア割り当て（アフィニティ）設定の問題と，解決のための新たな割り当て手法を提案する．

5.1 並列化時の理想的なコア割り当て方法

ステレオカメラ版では，左右のカメラ画像に対して pthread によるスレッドを 2 つ立ち上げて並列に特徴点を抽出する．そのため，各抽出処理をスレッド数 T で並列化したとすると，トラッキングでは $2T$ スレッド生成される．一方，ローカルマッピングでは OpenMP 並列化のスレッド数（L とする），ループクロージングではメインの 1 スレッド分生成される．なお，各処理から呼び出される g2o のグラフ最適化の並列化は，本研究ではローカルマッピングからの呼び出し時のみ行うように指定している．このような各処理のスレッドを実行するコアをできるだけ分散させるためには，次のような割り当てが有効である．なお，N はハードウェアに搭載されているコア数である．

(1) 最も実行時間が長いトラッキングとローカルマッピングのメインスレッドを異なるコアに割り当てる．

(2-1) ($2T + L + 1 \leq N$) 全スレッドを別のコアに割り当てる．

(2-2) ($2T + L + 1 > N$) まずローカルマッピングの L スレッド，および残りのコアにトラッキングのスレッドを連続に割り当てる．次に，未割り当てのスレッドを，実行頻度が小さいローカルマッピングの並列化部分のみを実行するコアに割り当てる．

表 1 は，10，6 コア環境での本割り当ての例である．なお各処理の先頭コアではメインスレッドの処理が行われる．

表 1: 10, 6 コア環境での，理想的な各処理の割り当てコア番号

スレッド数	10 コア		6 コア	
(T, L)	(2, 4)	(3, 4)	(2, 4)	
Tracking	0,1 and 2,3	0~2 and 3~5	0,1 and 3,4	0,1 and 4,5
LocalMapping	4~7	6~9	2~5	2~5
LoopClosing	8 or 9	7 or 8 or 9	5	3

5.2 OpenMP のコア割り当て機能の拡張

OpenMP では，並列化部分のコア割り当てに対して，環境変数 OMP_PLACES や OMP_PROC_BIND を用いた指定方法を提供している．OMP_PLACES は，並列化部分の実行コアをリストで指定したコアに限定するために使用し，OMP_PROC_BIND は限定した領域の中でのスレッドの分散度合いを制御する [15]．しかし，これらの既存のコア集合の指定では，同一の並列化階層にある 2 つのループは常に同じコア集合に割り当てられてしまう．そのため，トラッキングとローカルマッピングの並列化

表 2: ORB-SLAM3 の実行 CPU の仕様

項目	Intel Xeon	NVIDIA Jetson Xavier NX
コアの種類	Intel(R) Xeon(R) W-2155	ARM v8 Processor
コア数	10	6
最大周波数	3.30GHz	1.91GHz
L1d Cache	320KB	64KB
L2 Cache	10MB	2048KB
L3 Cache	13.8MB	4096KB

ループが，同じコア集合に割り当てられてしまい，コア領域を適切に分割することができない．

そこで，連続するコアに割り当てる場合には，各呼び出し元のマスタースレッドのコア番号を起点とした連続領域に割り当てるように，gcc の OpenMP ランタイム（libgomp）を拡張した．これにより，例えば表 1 にある 6 コアへの割り当ては，プログラム開始時に手動で各処理のメインスレッド，およびトラッキングのもう一方の pthread のコア番号を指定すれば，実行時に自動的に表にある通りに割り当てられるようになった．

6. 評　　価

6.1 評 価 環 境

本研究で評価に使用した 10 コア搭載の Intel Xeon サーバ，及び 6 コア搭載組込みボード NVIDIA Jetson Xavier NX の CPU 仕様を表 2 に示す．また評価には，安定した動作で軌道位置の推定精度が高いステレオカメラ版，および車上のカメラフレーム画像から構成される KITTI データセット [14] を使用している．さらに，特にローカルマッピングのローカルバンドル調整では，キーフレームが挿入されるたびに近傍の特徴点やカメラ位置に合わせてグラフや疎行列を構築し直すため，動的なメモリ確保が頻繁に行われるという特徴がある．そのため，マルチスレッド向けにメモリ確保処理が最適化されたメモリアロケータである，Jemalloc [16] を使用した．これにより，ローカルマッピングの実行時間は，約 1.17 倍速度向上している．

6.2 トラッキング単体を並列化させた場合の評価結果

ステレオカメラの左右画像に対して，T スレッドずつ並列化した場合の 1 フレームあたりの平均トラッキング時間を，表 3 に示す．Intel サーバ上では $T = 3$ の時に最大 1.26 倍，組込み Jetson 環境では $T = 2$ のときに最大 1.25 倍の速度向上が得られた．一方，Jetson では $T = 3$ の時は大きく鈍化しているが，これは合計で物理コア数を上回る 8 スレッドが生成され，同一コアでの他処理との競合が頻繁に発生したためである．

表 3: トラッキング単体並列化時の 1 フレーム平均 Tracking 時間 [ms]

スレッド数 T	10 コア Intel サーバ		6 コア組込み向け Jetson	
	アフィニティ有無		アフィニティ有無	
	なし	あり	なし	あり
並列化なし	45.03	—	72.56	—
$T = 2$	39.00	36.33	57.86	59.21
$T = 3$	39.43	35.72	64.30	78.51
$T = 4$	39.36	45.17	—	—

6.3 ローカルマッピング単体を並列化させた場合の評価結果

ローカルマッピングの処理について，並列化前，および公開されている g2o の OpenMP 並列化のみを適用した場合と提案手法を用いた並列化を行った場合の実行時間を表 4 に示す．また，組込み向け Jetson 実行時の処理時間の内訳を表 5 に示す．公開されている並列化実装に対して，提案手法を用いることで並列化前と比べて Intel サーバでは最大 1.37 倍，組込み向け Jetson では最大 1.34 倍の速度向上が得られた．また，Jetson のようなコア数の少ない環境では，提案手法の並列化性能をさらに引き出すためにアフィニティ設定が有効であることが確認できた．

表 4: ローカルマッピング単体並列化時の実行時間
(L は LocalMapping スレッドの並列化スレッド数)

L	1 フレームあたりの LocalMapping 平均処理時間 [ms]					
	10 コア Intel サーバ			6 コア組込み Jetson		
	オリジナル	提案手法 アフィニティ有無		オリジナル	提案手法 アフィニティ有無	
		なし	あり		なし	あり
並列化前	125.35	—	—	204.31	—	—
2	123.55	106.96	104.51	216.18	177.84	161.83
3	121.40	97.98	96.60	222.75	167.73	153.99
4	120.96	94.51	91.25	233.89	168.95	152.27

表 5: 6 コア組込み向け Jetson 上での並列化前後のローカルマッピング処理時間の内訳 [ms] (LBA: ローカルバンドル調整)

処理内容	並列化前	4 スレッド		
		オリジナル	提案手法 アフィニティ設定有無	
			なし	あり
KF insersion	14.47	13.52	13.51	14.86
MP Culling	0.93	0.97	0.94	0.95
MP Creation	46.87	45.33	29.00	25.43
LBA	138.60	171.32	120.76	106.32
KF Culling	5.06	4.48	5.30	5.21
全体時間	204.31	233.89	168.95	152.27

6.4 トラッキング・ローカルマッピング両方を並列化させた場合の評価結果

上記のように両処理ともに並列化を行った場合の速度向上を表 6 に示す．トラッキングは，ステレオの左右画像で T スレッドずつ，ローカルマッピングでは L スレッド分で並列化している．また割り当ての有無は，提案するコア割り当て手法を適用した場合と OS のスケジューリングに任せた場合を表している．

提案手法のコア割り当てを行うことで，Intel サーバでは $(T, L) = (3, 4)$ の時にトラッキング時間をさらに 13.7% 削減し，並列化前と比べて 1.40 倍の速度向上が得られている．また，組込み Jetson 上では $(T, L) = (2, 4)$ の時にローカルマッピング時間をさらに 6.2% 削減し，1.26 倍速度向上している．また両環境ともにもう一方の処理も速度向上していることから，提案手法による並列化，およびマルチスレッドアプリケーション向けの

表 6: トラッキング・ローカルマッピングとも並列化した際の速度向上

(T, L)	コア割り当て	1 フレームあたりの平均処理時間 [ms]			
		10 コア Intel サーバ		6 コア組込み Jetson	
		Tracking	LocalMapping	Tracking	LocalMapping
並列化前	なし	45.03	125.35	72.56	204.31
(2, 4)	なし	37.11	92.53	58.08	173.01
	あり	34.29	92.79	59.56	162.34
(3, 4)	なし	37.38	92.82	—	—
	あり	32.26	92.78	—	—

コア割り当て手法は，トラッキング・ローカルマッピング両スレッドを高速化させたい場合に有効であることを示している．

6.5 並列化した際の ORB-SLAM3 の軌道位置推定精度

SLAM では，できるだけ実空間の位置に近い精度で地図を生成しカメラ位置を推定することが重要であり，性能評価の指標には ATE と呼ばれるカメラ位置の推定データと正解データの絶対位置の誤差値が使われる [17]．表 7 の値は，並列化前の場合と，第 6.4 節で示したコア割り当てを行った上で両処理を並列化した場合の ATE の平均二乗誤差平方根 (RMSE) を表している [18]．本結果の並列化前後での約 10% の RMSE 値の増加は，データセットやカメラの種類は異なるものの，先行研究の 1 つである Li 等 [10] の高速化時の増加割合と同程度であるため，精度に与える影響は小さいと考えられる．

表 7: 並列化前・およびトラッキング・ローカルマッピング共に並列化を行った場合の ATE RMSE 値 [m]

並列化スレッド数 (T, L)	Intel サーバ	組込み NVIDIA Jetson
並列化前	1.23	1.23
(2, 4)	1.34	1.37

7. ま と め

本稿では，ORB-SLAM3 の処理で特に容易に並列性を引き出すことが難しいローカルマッピングを対象に，SLAM で生成されるデータ構造の特徴を生かした並列化とリダクションの方法を提案した．これにより，近傍にある特徴点・フレーム間の並列性に着目して，地図上の点群生成とローカルバンドル調整の並列化が可能となった．また，ローカルマッピングに加えてトラッキングの特徴点抽出を合わせて並列化した際に，各処理の並列化スレッドが使用するコア領域を適切に分割できるように，OpenMP ランタイムのコア割り当て機能を拡張した．これにより両処理を並列化した際にも，10 コアの Intel サーバの場合トラッキングで 1.40 倍，ローカルマッピングで 1.35 倍，またよりコアが少ない 6 コアの組込み NVIDIA Jetson 上でもトラッキングで 1.22 倍，ローカルマッピングで 1.26 倍の速度向上を同時に得ることができ，両処理の並列性を引き出すことが可能となった．特に本稿で提案している新しい OpenMP のコア割り当て手法は，ORB-SLAM3 に限らず様々なマルチスレッドアプリケーションにおいて，各スレッド内部の並列性を適切に引き出す際に有効であると考えられる．

謝　　辞

本研究の成果の一部は国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務 JPNP16007 の結果得られたものです．また本研究の一部は，JST 次世代研究者挑戦的研究プログラム JPMJSP2128 の支援を受けたものです．

文　　献

[1] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M.M Montiel, and Juan D. Tardos. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, Vol. 37, No. 6, p. 1874–1890, 2021.

[2] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, Vol. 31, No. 5, p. 1147–1163, Oct 2015.

[3] OpenCV: FAST Algorithm for Corner Detection. `https://docs.opencv.org/4.5.2/df/d0c/tutorial_py_fast.html`. (Accessed on 01/07/2022).

[4] BRIEF (Binary Robust Independent Elementary Features). `http://labs.eecs.tottori-u.ac.jp/sd/Member/oyamada/OpenCV/html/py_tutorials/py_feature2d/py_brief/py_brief.html`. (Accessed on 01/17/2022).

[5] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, 2011.

[6] Benoît Jacob Gaël Guennebaud, et al. Eigen v3. `http://eigen.tuxfamily.org/index.php?title=3.0`, 2010.

[7] adaskit Team. fixstars/cuda-bundle-adjustment: A CUDA implementation of Bundle Adjustment. `https://github.com/fixstars/cuda-bundle-adjustment`. (Accessed on 01/07/2022).

[8] Jie Ren, Wenteng Liang, Ran Yan, Luo Mai, Shiwen Liu, and Xiao Liu. MegBA: A High-Performance and Distributed Library for Large-Scale Bundle Adjustment, 2021.

[9] Zhaowei H, Yunzhi C, and Yiyou J. ORB-SLAM2 GPU Optimization. `https://yunchih.github.io/ORB-SLAM2-GPU2016-final/`.

[10] Jincheng Li, Guoqing Deng, Wen Zhang, Chaofan Zhang, Fan Wang, and Yong Liu. Realization of CUDA-based real-time multi-camera visual SLAM in embedded systems. *Journal of Real-Time Image Processing*, Vol. 17, pp. 713–727, 2019.

[11] Stefano Aldegheri, Nicola Bombieri, Domenico D. Bloisi, and Alessandro Farinelli. Data Flow ORB-SLAM for Real-time Performance on Embedded GPU Boards. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5370–5375, 2019.

[12] Ayoub Mamri, Mohamed Abouzahir, Mustapha Ramzi, and Rachid Latif. ORB-SLAM accelerated on heterogeneous parallel architectures. In *E3S Web of Conferences*, Vol. 229, p. 01055. EDP Sciences, 2021.

[13] 岡谷貴之．バンドルアジャストメント．研究報告コンピュータビジョンとイメージメディア (CVIM), Vol. 2009, No. 37, pp. 1–6, 2009.

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, Vol. 32, No. 11, pp. 1231–1237, 2013.

[15] GNU libgomp. OpenMP Environment Variables. `https://gcc.gnu.org/onlinedocs/libgomp/Environment-Variables.html#Environment-Variables`. (Accessed on 01/11/2022).

[16] Jason Evans. A scalable concurrent malloc (3) implementation for FreeBSD. In *Proc. of the bsdcan conference, ottawa, canada*, 2006.

[17] Amey Kasar. Benchmarking and comparing popular visual SLAM algorithms. *arXiv preprint arXiv:1811.09895*, p. 4, 2018.

[18] Michael, Grupp. evo: Python package for the evaluation of odometry and SLAM. `https://github.com/MichaelGrupp/evo`. (Accessed on 01/13/2022).