The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

グラフ最適化を用いた多人数追跡手法における グリッドマップ生成の並列化

Parallel Implementation of Probability Occupancy Map Generation for Multiple Human Tracking Based on Graph Optimization

西川 由理 *1*2　佐藤 仁 *1　小澤 順 *1
Yuri Nishikawa　Hitoshi Sato　Jun Ozawa

*1産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

*2パナソニック株式会社
Panasonic Corporation

Multi-camera multiple object tracking methodology that uses probability occupancy map (POM) and graph optimization can efficiently track multiple people in spite of significant occlusion, even without appearance model and prerequisite knowledge about the number of people in the target area. However, the computation time of POM increases according to grid size and number of cameras, which limited latency and throughput. In this paper, we report the effect of hierarchically applying two parallelization techniques to enhance the performance of POM generation. As a result, we achieved more than 20 times higher throughput compared to sequential processing, and reduce latency by 66%.

1. はじめに

カメラ映像から複数の人物を追跡する多人数追跡技術は，セキュリティ，業務効率化，マーケティング，スポーツ戦略分析など，多岐に渡る応用がある．一方，オクルージョンの前後で人物の対応をとる人物同定 (association) は，今なお，コンピュータビジョン研究の重要なテーマの一つである．

複数カメラの多視点映像を統合する手法は，このこの課題に対する有効なアプローチの一つである．特に [1] は，各カメラの人物検出結果から，空間を格子状に分割したセル毎の占有確率である 2 次元のグリッドマップ (Probability Occupancy Map: POM) を生成し，世界座標系の人物位置を推定する手法を提案している．また，各時刻のグリッドマップから定められるネットワークフローグラフの最適化により，人物を追跡する手法が提案されている [2]．この追跡手法は，人物の顔や色といった画像特徴量が不要であり，作業服やユニフォーム等，複数人のアピアランスが類似するシーンに適用できる．また，人数や移動方向に関する前提知識も不要なため，追跡空間に人が出入りしたり，速度や向きが急に変化したりするシーンへの応用も可能である．

我々は，これらの利点に着目し，グラフ最適化アルゴリズムを用いた多人数追跡システムの開発と評価を進めている [3][4]．追跡精度と実行時間を評価した結果，カメラ数やグリッド数が増えるほど追跡精度が向上する反面，グリッドマップ (POM) 生成の処理時間が増加することが明らかになった．例えば，20cm 角に区切ったサッカー場をカメラ 6 台で撮影した動画 [5] では，1 フレーム分の POM 生成に Intel Ivy Bridge-EP CPU でも 10 秒前後を要する．さらに，グラフ最適化による追跡では，50〜100 フレーム程度の POM を用いてネットワークフローグラフを構築するため，POM を順次生成すると，数秒の動画長にもかかわらず 10 分以上を要していた．

本稿では，2 階層の並列化を適用することで，PC クラスタ上で POM 生成処理を高速化した結果を報告する．POM 生成はフレーム毎に独立して演算可能であり，GNU Parallel[6] を用い，各フレームを複数の計算ノードに分散する処理を行った．さらに並列プログラミング API である OpenMP[7] を用い，1 ノード内でカメラ毎の演算処理をマルチスレッド化し，各スレッドの結果を統合する改良を行った．これにより，同サッカーのフレームあたりの POM 生成処理が約 10 秒から約 3.5 秒と，約 66%短縮された．さらに GNU Parallel を用い，ノード数を 1, 2, 4, 8 と変化させながら並列分散処理した場合も，ノード数の増加によらず，一律に約 4 秒で処理が可能であり，20 倍以上のスループット向上を確認した．

連絡先: 〒 135-0064 東京都江東区青海 2-4-7
産業技術総合研究所臨海副都心センター別館 9 階
西川由理 (nishikawa.yuri@aist.go.jp)

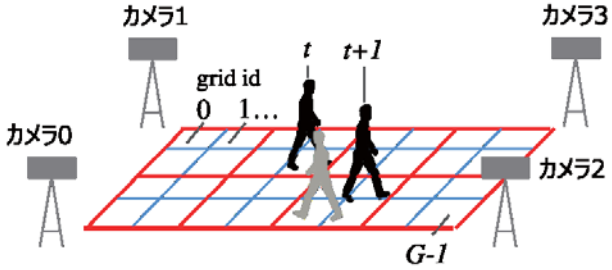


図 1: 追跡空間モデル

2. グラフ最適化を用いた多人数追跡

2.1 空間モデル

グラフ最適化を用いた多人数追跡では，図 1 に示すように，2 次元のグリッド（0,1，…，G-1）に分割された世界座標系の追跡空間において，各人物が連続フレーム間（t から t+1）にグリッドセルを移動すると想定する．また，各グリッドセルを占有するのが高々1 人であるという制約を置くことで，多項式時間で解くことが可能な最小コストフロー問題となる [2]．この制約を考慮すると，セルは高々一人が入る大きさにする必要がある．特に，人物の移動速度が速いスポーツなどでは，経験的に 20cm 角程度とするのが望ましいことが分かっている [4]．

2.2 多人数追跡システムの概要

多人数追跡システムの概要を図 2 に，またモジュール間のデータの依存関係を図 3 に示す．人物検出には深層学習ベースの一般物体認識手法 YOLO[8] を，グラフ最適化には K-Shortest

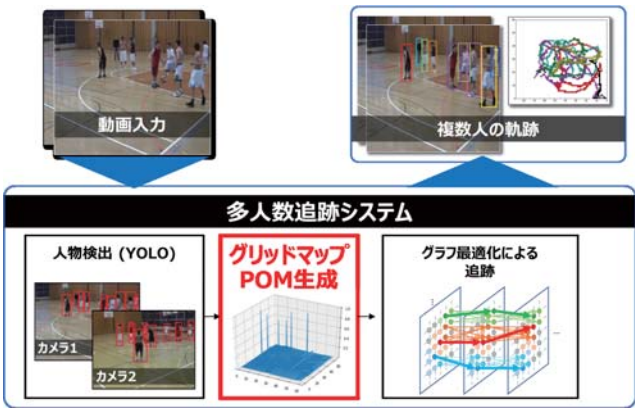


図 2: 多人数追跡システム

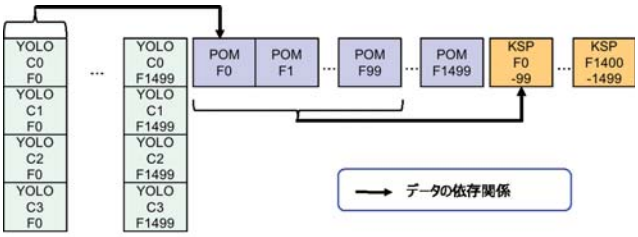


図 3: データの依存関係

Paths(KSP) を適用し，各々開発者が提供するソースコードを用いた．YOLO は 1 枚の画像を入力すると，人物検出領域のバウンディングボックス座標 (xmin, xmax, ymin, ymax) を出力する．POM 生成モジュールは，図 3 に示すように，複数カメラの出力を用い，グリッドマップを生成する．最後にグラフ最適化モジュールは，POM 生成モジュールが出力した複数フレームの POM からグラフを構築し，追跡を行う．

3. グリッドマップ POM の並列化

3.1 Probability Occupancy Map (POM) の概要

Probability occupancy map (POM) は，複数カメラで時刻 t に撮影された，画像中の人物のカメラ座標系における位置と大きさに基づき，格子状に区切られた空間の位置 i を人物が占有する確率 q_i^t を推定する手法である．基本となるのは，人物がある高さと直径の「円柱」であると想定し，人物が世界座標系の特定の位置に存在する場合に，その人物を表す円柱（四角形）を，カメラ画像中の適切な位置と大きさで描画する，生成モデルである．そして，複数カメラの画像が与えられた時，このモデルの下で，真の確率分布との KL(Kullback-Leibler) 情報量を最小化するモデル $Q^t = \{q_0^t, q_1^t, ..., q_{G-1}^t\}$ を推定することにより，各位置の占有確率を求める．なお，Q^t は，時刻毎に独立して演算が可能であるため，以後，必要がある場合を除き，添字 t を割愛して表記する．また，各変数の記法を表 1 に示す．

POM の具体的な生成手順を Algorithm1 の擬似コードに示す．また，2 つの画像 $\mathbf{A}$, $\mathbf{B}$ の類似度を表す「擬似距離」$\Psi(\mathbf{A}, \mathbf{B})$ を求める関数 get-pseudo-distance は，式 1 で定義される [*1]．また，各カメラの平均画像 $\mathbf{A_c}$ を求める関数 get-

*1 特に断りがない限り，太字の大文字の変数および「画像」という用語は，サイズ $W \times H$ の二次元配列を，次元変更した一次元ベクトルを表す．

Algorithm 1 POM の生成アルゴリズム

input: 時刻 t の人物検出結果 $\{\mathbf{B_0}, ..., \mathbf{B_{C-1}}\}$, 事前知識 $\{\mathbf{R_0^{\xi_0=1}}, ..., \mathbf{R_0^{\xi_{G-1}=1}}, ..., \mathbf{R_{C-1}^{\xi_{G-1}=1}}\}$
output: 時刻 t の POM $Q = \{q_0, ..., q_{G-1}\}$
$\{q_0, ..., q_{G-1}\} \leftarrow \{\epsilon_0, ..., \epsilon_{G-1}\}$
for $t \leftarrow 0$ to T **do**
  while $e < \varepsilon$ **do**
    for $c \leftarrow 0$ to カメラ数 C **do**
      $\mathbf{A_c} \leftarrow$ get-average-image $(\mathbf{A_c}, Q)$
      for $i \leftarrow 0$ to グリッドセル数 G **do**
        $\mathbf{S_c^{\xi_i=1}} \leftarrow \mathbf{A_c} \oplus \mathbf{R_c^{\xi_i=1}}$
        $\mathbf{S_c^{\xi_i=0}} \leftarrow \mathbf{A_c} \otimes \mathbf{R_c^{\xi_i=0}}$
        $\Psi_c^{\xi_i=1} \leftarrow$ get-pseudo-distance $(\mathbf{B_c}, \mathbf{S_c^{\xi_i=1}})$
        $\Psi_c^{\xi_i=0} \leftarrow$ get-pseudo-distance $(\mathbf{B_c}, \mathbf{S_c^{\xi_i=0}})$
        $sum_i += \Psi_c^{\xi_i=1} - \Psi_c^{\xi_i=0}$
      end for
    end for
    for $i \leftarrow 0$ to グリッドセル数 G **do**
      $tmp \leftarrow= 1/(1+\exp(\lambda_i + sum_i))$
      if $\mathrm{abs}(tmp - q_i) > e$ **then**
        $e \leftarrow \mathrm{abs}(tmp - q_i)$
      end if
      $q_i \leftarrow tmp$
    end for
  end while
end for

average-image は，式 2 で定義する．

$$\Psi(\mathbf{B}, \mathbf{S}) = \frac{1}{\sigma} \frac{|\mathbf{B} \otimes (\mathbf{I} - \mathbf{S})| + |(\mathbf{I} - \mathbf{B}) \otimes \mathbf{S}|}{|\mathbf{S}|} \quad (1)$$

$$\mathbf{A_c} = \mathbf{I} - \prod_i (\mathbf{I} - q_i \mathbf{R_c^{\xi_i=1}}) \quad (2)$$

Algorithm1 は，5 行目の誤差 e が収束するまで（多くの場合は 100epoch 以下）処理を繰り返す．直感的に述べると，ある epoch における $Q = \{q_0, q_1, ..., q_G\}$ から生成されるカメラ毎の生成画像 $\mathbf{S_c^{\xi_i}}$ と，実際のカメラ画像から YOLO 等の人物検出器が検出した人物位置に基づく画像 $\mathbf{B_c}$ の類似度 Ψ を式 1 により求める．誤差が収束し，2 枚の画像の類似度が十分近づいたとき，POM Q は，図 2 に示すように，ピークの尖った分布となる．

3.2 POM の並列化指針

並列化は，独立性の高い外側のループから順に行うのが基本指針である．ここで，Algorithm1 によれば，フレーム毎の演算を表す 4 行目の for ループは，独立して演算が可能である．一方，6 行目の for ループは，要素数 G の配列 sum にループ間の依存関係がある．しかし，配列 sum をループ毎にプライベートに確保し，14 行目の処理が完了した時点で要素毎にリダクション演算を行うことにすれば，6 行目のループでマルチスレッド化することで，独立に演算が可能であることが分かる．

以上を踏まえた，POM の並列化方針を図 4 に示す．まず，1 プログラム (コマンド) で 1 フレーム毎の POM 計算（4 行目のループ）を行うこととした．そして，GNU Parallel[6] というコマンドの並列実行を可能とするシェルツールを用い，複数の計算ノードを用いて分散処理することとした．一方，カメラ毎の並列化（6 行目のループ）は，14 行目終了時点で，都度

表 1: 変数の記法

C	カメラ数
G	グリッドセル数の合計
T	バッチ処理するフレーム数 (=50〜100 程度)
t	時刻 (フレーム)
i	位置
c	カメラ ID
ε	許容誤差
q_i	時刻 t における位置 i の占有確率
Q	時刻 t の POM $(=\{q_0, q_1, ..., q_G\})$
ξ_i	位置 i の真の占有確率
	$\xi_i = 0$:不在，$\xi_i = 1$:人が存在
ϵ_i	位置 i における事前確率
λ_i	ϵ_i のロジット $\log \frac{1-\epsilon_i}{\epsilon_i}$
$W \times H$	カメラ画像の解像度 (ピクセル)
$\mathbf{I}$	値が 1 の大きさ $W \times H$ の画像
$\lvert\mathbf{A}\rvert$	画像の合計（ベクトル要素同士の和）
$\mathbf{A} \otimes \mathbf{B}$	画像 $\mathbf{A}$ と $\mathbf{B}$ の要素毎の積
$\mathbf{A} \oplus \mathbf{B}$	画像 $\mathbf{A}$ と $\mathbf{B}$ の要素毎の和
$\Psi(\mathbf{A}, \mathbf{B})$	画像 $\mathbf{A}$ と $\mathbf{B}$ の擬似距離 (pseudo distance)
$\mathbf{A_c}$	カメラ c における平均画像
$\mathbf{B_c}$	人物検出器が検出した矩形内に対応する
	要素が 1，その他が 0 の画像
$\mathbf{R_c^{\xi_i=1}}$	カメラ c の位置 i における想定人物矩形内に
	対応する要素が 1，その他が 0 のシルエット画像
	$\mathbf{R_c^{\xi_i=0}} = \mathbf{I} - \mathbf{R_c^{\xi_i=1}}$
$\mathbf{S_c^{\xi_i}}$	画像 $\mathbf{A_c}$ と画像 $\mathbf{R_c^{\xi}}$ から生成される画像

表 2: 計算ノード仕様

項目	値
CPU	Intel Xeon E5-2680 v2 x 2
	(Ivy Bridge-EP)
CPU コア数	10 core/CPU
スレッド数	1 thread/core
主記憶	128GiB DDR3-1866
I/F	InfiniBand Mellanox ConnectX-3(FDR)
	Intel X520-DA2(10 GbE)
OS	CentOS 6.7

sum 配列の要素毎の和を求めるリダクション演算が発生する．CPU を用いるノード内並列化手法の候補としては，MPI[9] と OpenMP[7] が挙げられる．簡易評価の結果，プロセス毎に独立のメモリ空間を占有する MPI では，リダクション演算を行うたびにメッセージサイズが G の MPI 集団通信が発生し，性能低下が生じた．従って，共有メモリ型のスレッド並列化手法である OpenMP を用い，カメラ毎にプロセッサ内のコア（スレッド）を割り当てる，マルチスレッド化を行った．なお，グリッドセル単位（8 行目のループ）の並列化も，アルゴリズム上は可能だが，多くの同期の発生が見込まれ，今回は行っていない．

4. 評価

本評価では，産業技術総合研究所の PC クラスタ，AIST Super Green Cloud(ASGC) 上の計算ノード (表 2) を最大 8 台を用いた．実行時間の評価には，バスケットボール動画 [10]

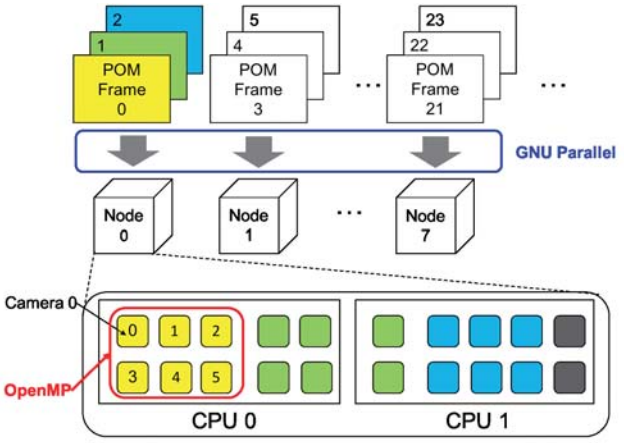


図 4: POM の並列化方針

表 3: 評価用動画の概要

	バスケ	サッカー 50	サッカー 20
空間の広さ	15m x 15m	120m x 90m	120m x 90m
グリッド	100 x 100	212 x 138	530 x 345
カメラ数	4	6	6
人数	13	22	22
fps	25	25	25

およびサッカー動画 [5] を用いた．動画の概要を表 3 に示す．前者は，半コート内で選手が競技する様子を，約 170cm の高さに設置した 4 台のカメラで撮影したものであり，多数のオクルージョンが含まれる．後者は競技を 6 台のカメラで撮影したものである．バスケットボールは追跡空間を約 20cm 角，サッカーは 50cm 角 (サッカー 50) と 20cm 角 (サッカー 20) のグリッドで離散化した．

3 種類の動画について，次の 3 点を評価した．

1. ノード内並列化による 1 フレームの処理高速化の効果 (OpenMP)
2. 1 ノードに 1 フレームのみ割り当てた時の，ノード数増加による処理高速化の効果 (GNU Parallel)
3. 1 ノードに複数フレーム（ジョブ）を割り当てた時の，ノード内ジョブ数の増加による処理時間への影響

1 の結果を，図 5 に示す．バスケットボール動画は 4 台のカメラの映像を用いるため 4 スレッド，サッカーは 6 スレッドで演算処理を行った．Epoch 毎に結果を統合するために発生した同期処理が OpenMP による並列化のオーバヘッドとなったが，バスケットボール動画で約 2.5 倍，サッカーで約 3 倍の高速化を確認した．特に OpenMP 適用前では「サッカー 20」

表 4: ノード数と処理時間の関係 (単位:秒)

ノード数	バスケ	サッカー 50	サッカー 20
1	0.73	2.12	4.07
2	0.75	2.18	4.05
4	0.75	2.24	4.16
8	0.78	2.19	4.11

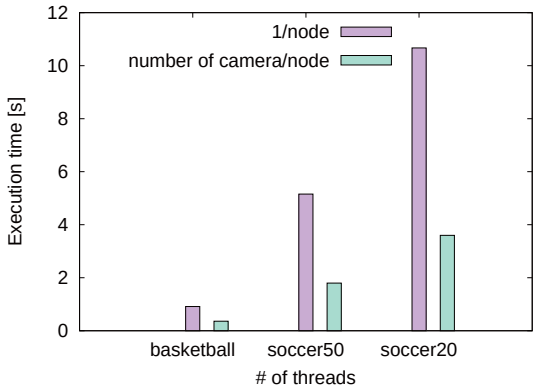


図 5: OpenMP によるカメラ毎並列化の効果

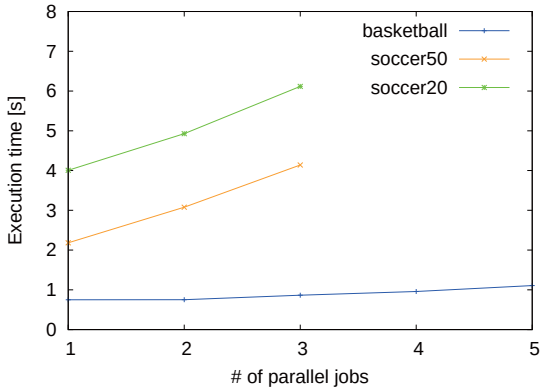


図 6: 1 ノード内のジョブ数と処理時間の関係

で約 10.7 秒を要した POM 生成処理が約 3.6 秒となり，実行時間が約 66%削減された．

次に，計算ノード数と処理時間の関係を，表 4 に示す．各ノードに割り当てるフレーム数を 1 に固定した上で，ノード数を 1, 2, 4, 8 台と変化させ，GNU Parallel で分散処理を行った時間を計測した．表に示すとおり，処理時間はノード数によらずほぼ一定であり，8 フレーム分の「サッカー 20」の結果が約 4.1 秒で得られた．OpenMP 適用前かつ 1 ノードでこれらの処理を行うには，$10.6 \times 8 = 84.8$ 秒を要したことから，分散処理により 20 倍以上の高速化が実現された．

最後に，GNU Parallel を用いて，1 ノード内に複数の OpenMP のジョブを投入したときの，ジョブ数と処理時間の関係を図 6 に示す．ノード数は 1 に固定 (20 スレッド/ノード) にした上で，6 スレッドを使用する OpenMP のジョブ数を 1〜3 (バスケットボールのみ 1〜5) と変化させた時の処理時間を計測した．図に示すとおり，処理時間は並列ジョブ数に従って増加し，サッカー 50 では約 1.9 倍となった．性能低下要因としては，GNU Parallel のオーバヘッドに加え，Disk I/O がボトルネックになっている可能性が挙げられる．今後より詳細に要因を分析し，性能改善を図る予定である．

5. まとめ

OpenMP と GNU Parallel を用いた 2 階層の並列化により，POM 生成処理の高速化を行った．まず OpenMP を用いたカメラ毎の演算処理のノード内並列化により，サッカー動画 1 フレームあたりの POM 生成処理に要する時間を約 66%削減した．さらに，GNU Parallel を用いて最大 8 台の計算ノードに 1 フレームずつ割り当てるノード間並列化を行ったところ，ノード数の増加によらず，ほぼ一定の遅延時間で，各ノードから出力が得られた．以上により，遅延時間の削減とスループットの向上を確認した．

一方で，1 ノード内に複数の POM 生成処理のジョブを投入すると，個々のジョブの処理時間が増える課題も明らかとなった．今後は性能要因を分析し，遅延時間増大の抑制を図ると共に，GPGPU 対応による処理時間の短縮を目指す．

参考文献

[1] F. Fleuret, J. Berclaz, R. Lengagne and P. Fua, *Multi-Camera People Tracking with a Probabilistic Occupancy Map*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **30**(2), 267/282, 2008.

[2] J. Berclaz, F. Fleuret, E. Türetken and P. Fua, *Multiple Object Tracking Using K-Shortest Paths Optimization*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **33**(9), 1806/1819, 2011.

[3] Y. Nishikawa, H. Sato and J. Ozawa, *Performance Evaluation of Multiple Sports Player Tracking System Based on Graph Optimization*, in IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications (BPOD2017) in IEEE BigData2017, 2903/2910, 2017.

[4] Y. Nishikawa, H. Sato and J. Ozawa, *Multiple Sports Player Tracking System Based on Graph Optimization Using Low-Cost Cameras*, in IEEE International Conference on Consumer Electronics (ICCE), 2018.

[5] T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, P. L. Mazzeo, *A Semi-Automatic System for Ground Truth Generation of Soccer Video Sequences*, in 6th IEEE International Conference on Advanced Video and Signal Surveillance, 2009.

[6] O. Tange, *GNU Parallel - The Command-Line Power Tool*, in ;login: The USENIX Magazine, **36**(1), 42/47, 2011.

[7] L. Dagum and R. Menon, *OpenMP: An industry standard API for shared-memory programming*, in IEEE Computational Science and Engineering, **5**(1), 46/55, 1998.

[8] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779/788, 2016.

[9] W. Gropp, E. Lusk, N. Doss and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing, **22**(6), 789/828, 1996.

[10] `https://cvlab.epfl.ch/data/pom`