C-P44

# デプスカメラを用いた屋内自己位置推定における GPU 並列処理の有用性評価

北海道大学　○畠山 龍，　金井 理，　伊達 宏昭

要旨

本研究では，地上設置型レーザースキャナにより得られた精密な 3 次元の屋内環境モデルを環境地図情報とし，デプスカメラから得られる距離画像を用いた 3 次元の自己位置推定を行うことを目的とする．本報では，自己位置推定で必要となるモンテカルロ自己位置推定(MCL)処理高速化のための GPU 並列処理の有用性を評価する．

## 1. はじめに

本研究では，地上型レーザースキャナで作られた詳細な 3 次元メッシュモデルを環境モデル(環境地図)として利用し，さらにセンサとしてデプスカメラを用いた 3 次元自己位置推定の実現を目的とする. 前報[1]ではモンテカルロ自己位置推定(Monte Carlo Localization:MCL)[2]手法を用い，メッシュモデルに対する 3DCG レンダリングを利用する GPU シーンシミュレーション[3]によりデプスカメラから得られる距離画像をシミュレーションし，実際に自己位置推定を行った．この実験結果では，レーザースキャナを用いて生成された高精度な 3 次元環境モデルを利用することで先行研究[3]より位置推定精度が改善されることを確認したが，MCL 処理の速度が十分ではなく，完全 3 次元自己位置推定(同時 6 状態変数推定)は実現できていなかった．

そこで本報では, 更なる MCL 処理高速化のために, GPGPU を導入し, 前報での MCL 内の尤度計算を GPU による並列処理で行う有用性を評価する予備実験を行ったので報告する．

## 2. モンテカルロ自己位置推定アルゴリズム

MCL は，推定対象の状態空間上の存在確率密度分布をパーティクルにより離散的に表現する手法である[4]. 前報で提案したパーティクルフィルタを用いた自己位置推定アルゴリズムを図 1 に示す．この手法では，デプスカメラの位置・姿勢を状態変数とし, 主に次の 3 ステップの処理を繰り返す. (A-1)現時点でのパーティクルの集合を予測に基づき伝播し予測パーティクル集合を生成する. (A-2)予測パーティクル集合に対して，図 2 に示す GPU シーンシミュレーションによりシミュレーション距離画像を生成し，デプスカメラから得られる実測の距離画像とこれを比較することで，パーティクルの状態が実際のデプスカメラの状態とどの程度一致しているかを表す尤度を計算し，重み付けを行う．(A-3)最後に重みに基づき予測パーティクル集合をリサンプリングすることで，パーティクルを更新する．以後はパーティクルを更新しながら，同様にこの 3 ステップの処理を繰り返す．状態の推定には, (A-2)で得られた予測パーティクル集合の重み付き和を用いている．

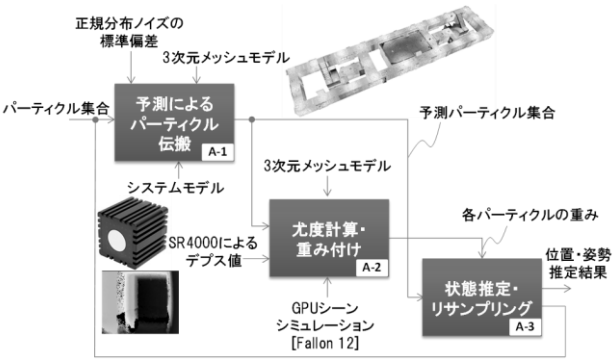


図 1　MCL を用いた自己位置推定アルゴリズム[1]

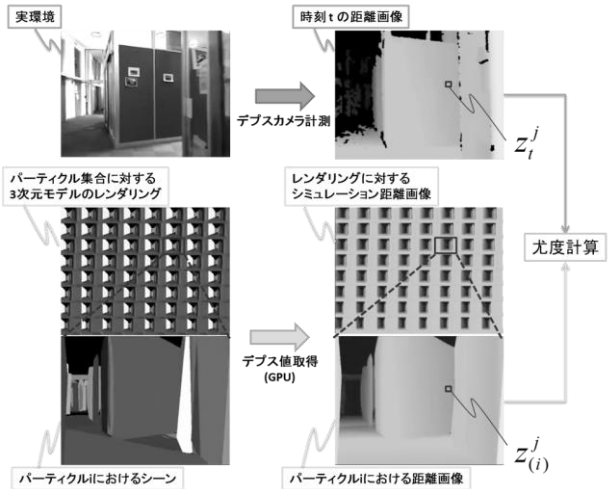


図 2　GPU シーンシミュレーション[3]概要

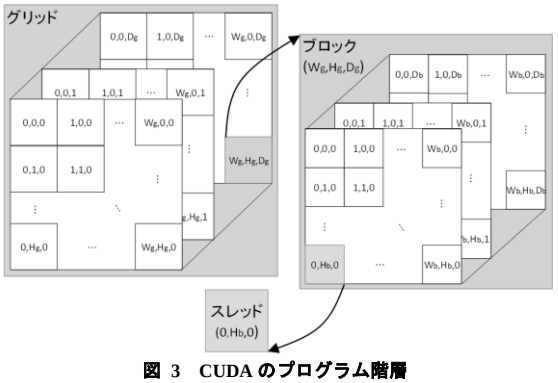


図 3　CUDA のプログラム階層

## 3. GPGPU を用いた自己位置推定の高速化手法

### 3.1 GPGPU

GPGPU は General Purpose computing on GPU(GPU による汎用計算)の略語であり，GPU の高い並列計算能力を用いて，従来 CPU で処理していた汎用計算を高速に実行させることを意味する．現在，CPU のプロセッサコアが数個～数十個であるのに対して GPU は数百から数千個という非常に多くのコアを積んでおり, 並列計算におけるメリットが多い. また, GPU 汎用計算には NVIDIA 社が統合開発環境として CUDA を提供しており, 本研究でもこの CUDA を用いて GPGPU による MCL の高速化を行った．

### 3.2 CUDA

CUDA は NVIDIA 社が提供するフレームワークまたは統合開発環境のことである．CUDA においてマザーボード，CPU，メモリなどで構成される PC 本体側をホストと呼び，グラフィックボード側をデバイスと呼ぶ．ホスト側のプログラムは，通常のようにホスト上の CPU で動作し，ホスト上のメモリを利用する．デバイス上で動作するプログラムはカーネルプログラムと呼ばれ，デバイス上で動作し，GPU が処理を行い，VRAM のメモリを利用する．

CUDA のプログラム階層を図 3 に示す．プログラム構造は，グリッド, ブロック, スレッドによる階層構造になっており，グリッドの中にブロックが，さらにブロックの中にスレッドがそれぞれ 3 次元で定義されている．カーネル関数の実行時にブロック，スレッドの利用範囲を指定し，関数内ではブロ

ック，スレッドそれぞれの 3 次元のインデックスを用いてデータを参照し，並列処理を行う．

### 3.3 GPGPU による高速化手法概要

前報[1]の実験では，2 節の処理(A-1)～(A-3)の中で最も負担がかかる計算処理が(A-2)であることが判った．(A-2)では OpenGL により，パーティクルに対応する位置・姿勢から 3 次元環境モデルのレンダリングを行い，Z バッファ値を取得することでシミュレーション距離画像を得る．この手法は，CPU 上でシミュレーション距離像を生成するよりも効率的ではあるものの，推定精度向上のためパーティクル数を増加させようとするとパーティクル毎に必要な Z バッファ値のデータ転送がボトルネックとなる問題があった．

完全 3 次元自己位置推定の実現にはパーティクル数の増加が必要不可欠であると考えられる．そこで本報では，GPGPU の導入による前述のボトルネック解消と処理高速化の可能性に関する予備実験を行った．

前述の処理(A-1)～(A-3)は全て CPU(ホスト側)により実行されていたが，本報では(A-2)を GPU(デバイス側)で実行することを考える．前報におけるホスト-デバイス間の処理とデータ転送，本報での GPGPU による高速化案を**図 4** に示す．従来の自己位置推定では(A-2)の尤度計算・重み付けの処理において，パーティクル毎にシミュレーション距離画像を生成し，そのデータを Z バッファからホスト側まで転送し(**図** 4(a)-②)，同じくホスト側に転送した計測データ(**図** 4(a)-③)と比較することで尤度計算・重み付けを行っていた．

これに対し，パーティクル毎の距離画像を生成した後に，そのデータをホスト側に転送することなく，CUDA を用いてデバイス側で計測距離画像と比較し，尤度計算・重み付けまでを行うことが可能となれば，結果のパーティクル毎の重みのみをホスト側に転送すればよい(**図** 4(b)-②)．

さらに，時刻フレーム$t$におけるパーティクル$i$の尤度$l_{(i)}$は下式(1)により計算される．ここで，$m$はピクセル数，$z_t^j$は計測距離画像の$j$番目ピクセル距離値，$z_{(i)}^j$はパーティクル$i$のシミュレーション距離画像の$j$番目ピクセル距離値である．

$$l_{(i)} = \sum_{j=1}^{m} \left( 1 - \frac{\left| z_t^j - z_{(i)}^j \right|}{0.5 + \left| z_t^j - z_{(i)}^j \right|} \right) \qquad (1)$$

パーティクル数が$m = 100$だとすると，従来では式(1)のピクセル$j$ごとの演算が約 250 万回行われていた．しかし式(1)のΣ内部のピクセル$j$の尤度の計算は逐次的なものではないため並列化が可能であり，GPGPU の導入により尤度計算自体の高速化が期待できる．

## 4. 実験結果

デプスカメラとパーティクルから得られる仮想的な距離画像をあらかじめ生成し，従来のように CPU のみを用いて式(1)を計算した場合と，CUDA を用いて GPU による並列処理で式(1)を計算した場合の処理時間を比較した．これにより MCL の尤度計算に対する GPGPU 導入の有用性を評価した．なお，距離画像の画素数，パーティクル数，使用機器は**表** 1 の通りである．

それぞれの実行結果の処理時間の 10 回平均値を**表** 2 に示す．ただし，GPU 処理では並列化数を 1（並列化なし），176×144(画素数), 176×144×100(画素数×パーティクル数)の 3 パターンに設定した．**表** 2 から GPU の並列化数を増やすと処理時間が短縮されることが確認できる．また CPU のみを用いた場合の処理時間が 10ms だったのに対して，最大限並列化を行った場合は処理時間が 1ms となっている．このことから尤度計算のみに関しては処理時間が従来の 10%に削減可能なことが判った．

## 5. おわりに

本報では，前報で提案した自己位置推定手法の更なる高速化に向けて，GPGPU の導入による処理高速化の可能性に関して予備実験により評価を行い，尤度計算において従来手法に対し大きな処理速度の向上を確認した．しかし，ホスト-デバイス間でのデータ転送量の削減に関しては，デバイス内での OpenGL のレンダリング結果の Z バッファ値を CUDA で読み取る関数が標準では未だサポートされておらず，距離画像データの高速な受け渡しの効果については評価できなかった．今後は，OpenGL のシェーダにより Z バッファ値をカラー情報としてレンダリングすることで，この問題を解決できる可能性があるため，その実装を行う予定である．

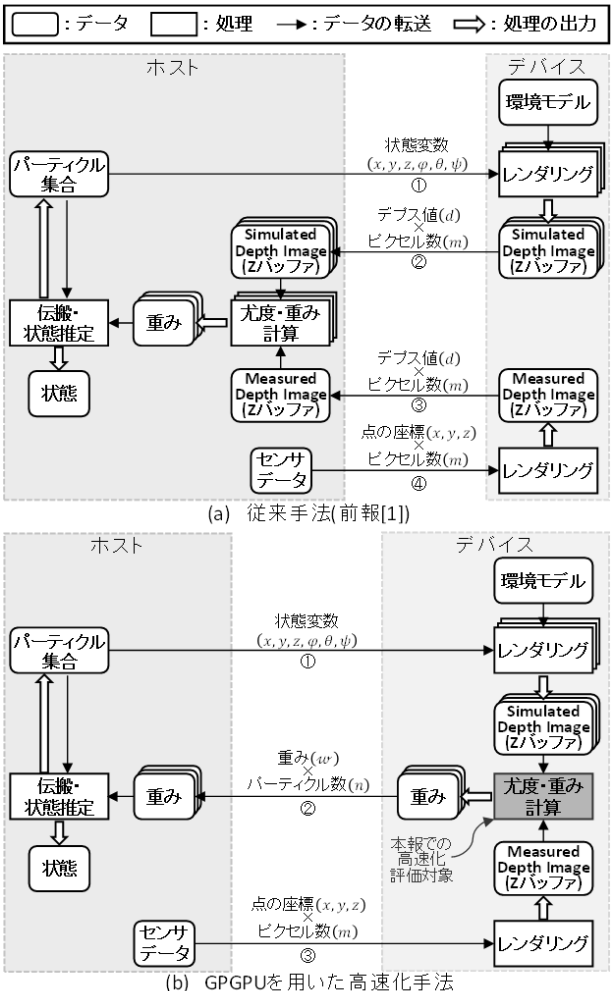


**図 4 ホストーデバイス間の処理，データ転送**

**表 1 パラメータ，実験環境**

| | |
|---|---|
| 距離画像画素数 | 176×144 |
| パーティクル数 | 100 |
| CPU | Intel Core i7-3770 (3.4GHz 8コア) |
| GPU | Geforce GTX770 (1.046GHz 1536コア) |

**表 2 実験結果(10 回平均値)**

| | 従来手法[1] | 提案手法（並列化数） | | |
|---|---|---|---|---|
| | | 1 | 176×144 | 176×144×100 |
| 処理時間 | 10ms | 460ms | 2ms | 1ms |

## 参考文献


[1] 畠山ほか，「レーザースキャン環境モデル内でのデプスカメラによるモンテカルロ自己位置推定」, 2013 年度精密工学会秋季大会学術講演会講演文集，pp.633-634，(2013)

[2] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun, “Monte Carlo Lolalization for Mobile Robots”, Proc.ICRA, Vol.2, pp.1322-1328, (1999)

[3] Maurice F. Fallon, Hordur Johannsson, and John J. Leonard, “Efficient Scene Simulation for Robust Monte Carlo Localization using an RGB-D Camera”, Proc.ICRA, pp.1663-1670, (2012)

[4] 樋口知之,『予測にいかす統計モデリングの基本ーベイズ統計入門から応用まで』，講談社，(2011)