5A-02

リアルタイム予測制御の処理高速化を目的とした車載 ECU への実装方法

坂田昂亮[†1]　川邊武俊[†2]　湯野剛史[†3]
日立オートモティブシステムズ株式会社[†1]　九州大学[†2]　九州大学[†3]

1. はじめに

近年、自動車機器業界は自動運転車の普及に向けた開発に力を入れており、自動運転車のシステム構成や次世代 ECU(電子制御ユニット)の検討が盛んに行われている。車両が自動で走行するためにはドライバに代わり車両の将来の行動を計画する機能が必須であり、それは数 km 規模の長距離行動計画と、数秒規模の短距離行動計画に大別できる。短距離行動計画は時々刻々と変化する外界環境に応じて常に安全を担保するために、比較的高頻度に計算を更新する必要があり、実使用範囲の車速や近年の外界センサの性能を考慮すると 50～100[msec]の更新周期が妥当だと考えられる。このように行動計画は安全性の観点からレイテンシに厳しい制約があるが ECU にはセンシングや車両制御などその他のアプリも共存しているため、全体として要求の更新周期を満たすよう、それぞれのスループットを短縮することが課題となる。

他方で、近年の運転支援向け車載 ECU ではソフトウェアの大規模化に対応するため、マルチコア CPU を採用することが増えている。 アプリを同時並列に処理することで、全体のスループットを向上させられるため、今後はより一層マルチコア CPU や GPU、FPGA などを活用した並列化 ECU 実装技術に対するニーズが高まっていくと予想される。

本報告では行動計画を実現する手法の一つとして文献[1]などで報告されている、モデル予測制御をターゲットとし、当該手法をマルチコア CPU を用いて高速に処理する分散化実装方法について検討した結果を報告する。

2. 従来のモデル予測制御の課題点

モデル予測制御には種々の実現方法があるが、本報告では一例として目的関数を最適化することで入力系列を得る方法に注目した。本報告では紙面都合により説明を割愛するが文献[1]ではこの方法によってモデル予測型クルーズコントロールを構成する実現方法が示されている。

目的関数の最適化を行うための数値計算手法としては GMRES が比較的計算負荷が軽くリアルタイム実装に適していることが知られているが[2]、全体の計算量はベクトル空間の二乗に比例して増加し、また、ベクトル空間のサイズは予測する仮想時間空間を離散化する細かさに比例して増加する。車両制御に利用するための十分な分解能の軌道を得るためには問題サイズが肥大化し、オリジナルの実装では車載 ECU の限られた計算資源でのリアルタイム実行は困難である。今回、GMRES の計算量の内訳について分析を行った結果、処理時間の 97%が Gram-Schmidt 正規直交化によるものだということが分かった。当該のアルゴリズムを以下に示す。

Algorithm 1 : Classic Gram-Schmidt

```
1  do j = 1,n
2  |  q_j = a_j
3  |  do i = 1,j − 1
4  |  |  q_j = q_j − (q_i,a_j)q_i
5  |  end do
6  |  q_j = q_j / ‖q_j‖
7  end do
```

n はベクトル空間の次元数、a は線形独立なベクトル、q は正規直交基底をそれぞれ意味する。アルゴリズム内の特に内側の i 方向ループで行われるベクトルの内積、積、減算の処理が負荷の大部分を占めている。

そこで、これらのベクトル演算を効率化することがスループット向上に最も効果が高いと判断し、当該処理を対象としてマルチコアへの分散実装を検討することとした。Gram-Schmidt 直交化の並列化に関しては幾つかの報告がすでになされているが、本報告ではソフトウェア並列化に伴う処理の複雑化によって、開発工数が増大するという普遍的課題に対し、工数への影響を低減させる目的で、出来るだけシンプルな実装形態を目指して検討した。

3. 計算負荷軽減実装方法

上述したベクトル演算はループのイタレータ i に対して計算順序の依存関係がないため、処理を分散化して非同期に実行することができる。例えばループ i で処理されるタスクを前半と後半

Efficient implementation of real-time predictive control
†1 KOSUKE SAKATA , Hitachi Automotive Systems, Ltd.
†2 TAKETOSHI KAWABE , Kyushu University.
†3 TSUYOSHI YUNO , Kyushu University.

に 2 分割した場合の計算空間のモデルを図 1 上段に示す。図中の直角三角形の縦軸はループ *j* を示し、横軸はループ *i* を示している。また、三角形の面積が計算の全体量を示し、色を分けて示している計算空間の *i* 方向の演算は並列に計算することが可能なことを示している。以上を踏まえて、分割された計算空間を並列に処理する方法として図 1 下段に示す実装方法を提案する。

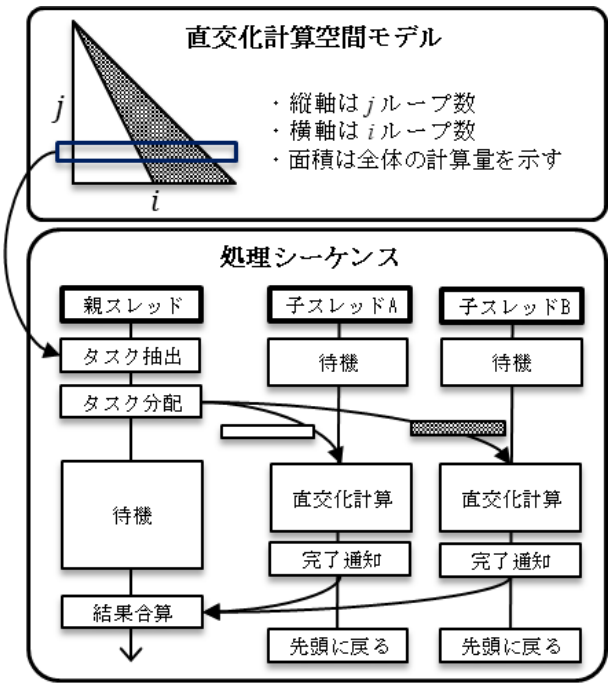

図 1　分散化マルチスレッド実装形態

処理は主体となる親スレッドと、分割されたベクトル演算を行う複数の子スレッドで構成されている。親スレッドは計算空間モデルから処理すべきタスクを順に抽出し、*i* 方向で分割した処理タスクを子スレッドに割り当てる。子スレッドはスレッドプールとしてプロセス内に常駐しており、タスクが割当たり次第処理を実行し、処理の完了を親スレッドに対して通知する。通知を受けた親スレッドは子スレッドの処理結果を統合して直交化されたベクトルを得る。その後は必要に応じて正規化計算を行い、*j* 方向ループを次に進めていく。本実装形態によれば利用できるハードウェア資源に応じて子スレッドを増やして並列数を増やすことで、同様の形態のままスケーラブルな設計とすることが出来る。

4. テスト及び結果

提案手法によるモデル予測アプリをマルチコア CPU へ実装しテストを行った。表 1 にテスト条件を示す。テストでは Gram-Schmidt 直交化の処理時間を計測しオリジナル実装に対する高速化効果について検証を行った。

表 1　テスト条件

CPU	Corei7- 3740QM 2.7GHz 4Core
サイズ	n=100～1000 で試行
並列数	2 並列、4 並列
言語	C++, pthreads

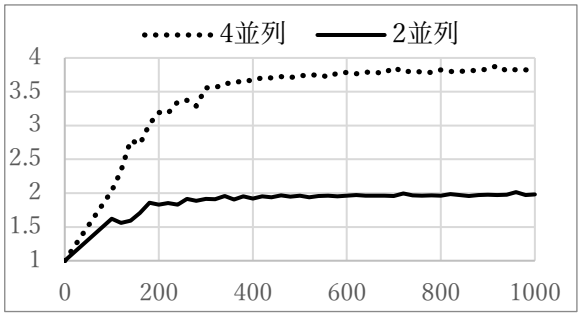

図 2 速度改善効率(横軸:サイズ 縦軸:効率[倍])

結果として得られた問題サイズと速度高速化効率の関係を図 2 に示す。2 並列、4 並列ともに期待通りに高速化している。問題サイズが大きいほど効率が高く、収束値は 2 並列が約 1.98 倍、4 並列が約 3.80 倍となった。

5. 結果の考察

2 並列の高速化効率は期待値(2 倍)に対して 99%の割合に収束している、4 並列については期待値(4 倍)に対して 95%となった。収束割合に差がある原因は、分散化に伴うオーバーヘッドが 4 並列の方が多く発生することが原因と考えられる。
また 2 並列は *n*=300 付近で効率が収束しているが、4 並列は *n*=600 付近であり、小サイズの問題に対して効率が比較的得にくい結果となった。

6. 結論と残課題

モデル予測制御の計算負荷が高い箇所を明らかにし、提案した分散化実装によって高速化が可能であることを示した。本手法は計算空間モデルの頂点付近では分割できるループ数が小さいため、高速化効果が低いと考えられる。

適切な分割サイズや損益分岐点を明らかにすれば、より小サイズの問題での効率改善が期待できる。

参考文献

[1]草富義也ら：信号機切り替わりの将来情報を利用したモデル予測エコ・ドライビング制御 第59回自動制御連合講演会(2016)
[2]大塚敏之ら：実時間最適化による制御の実応用(2015)