# 動的計画法の並列化による省エネルギー列車運転曲線の計算高速化の検証

坂井　桂祐*　　大西　亘　　古関　隆章（東京大学）

## Verification of Fast Calculation of Energy-Saving Speed Profile by Parallel Dynamic Programming

Keisuke Sakai*, Wataru Ohnishi, Takafumi Koseki (The University of Tokyo)

It is important for electric railways to reduce running energy more to achieve carbon neutrality. Dynamic Programming is a common method to optimize speed profile in the driving strategy approach for energy-saving. However, it needs huge computational costs to improve the solution quality. The aim of this paper is to propose a parallel computing method to accelerate optimization with Dynamic Programming. We evaluate the calculation time and solution quality by changing the number of processors.

**キーワード**：電気鉄道，省エネルギー運転，運転曲線，最適化，動的計画法，並列計算
(electric railway, energy-saving driving, speed profile, optimization, dynamic programming, parallel computing)

## 1. はじめに

脱炭素社会の実現に向けて電気鉄道でも走行エネルギーをさらに削減することが求められる。排出される二酸化炭素の絶対量は，走行時に化石燃料を燃焼させるディーゼル車両よりも，運行本数の多い電車走行用の割合が多いためである[(1)]。

省エネルギー運転には 2 つのアプローチがある。1 つは電力変換半導体を改良する，あるいは電力貯蔵装置 (Energy Storage System) を設置するといったハードウェア的アプローチである。それぞれ駆動効率や，回生電力の再利用率の増加を通じて省エネルギーを達成できる。もう 1 つは運転操作を工夫するソフトウェア的アプローチである。古くからモータもブレーキも使わず惰性で走行する時間を長くして走行エネルギーを削減できることが知られており[(2)]，そのような運転操作を求める様々な方法が研究されてきた。

ソフトウェア的アプローチは駅間の列車運転操作を定める運転曲線の最適化問題として解かれてきた。代表的な手法として，動的計画法，変分法，遺伝的アルゴリズムがある。動的計画法は信号を含む[(3)] 速度制限や非線形の走行抵抗・モータ効率[(4)] といった複雑なモデルに対応している[(5)] が，解の精度と計算時間にトレードオフがあり，解の精度を高めるには計算量が多くなる課題がある。変分法は運転曲線を関数表現し最適化する手法で，厳密解を容易に得ることができるが，速度依存性を持つ引張力特性などの複雑なモデルに対応していない。遺伝的アルゴリズムは複雑なモデルに対応しており計算も高速であるが，解が大域最適である保証がない。

本研究は動的計画法に着目し，並列計算によりこの計算を高速化することを目的とする。コア数が増えている近年の計算機を最大限に活用するには並列計算が不可欠なためである。ムーアの法則に従って半導体を微細化することによるコアあたりの性能向上は限界を迎えつつあり，近年は中央処理装置 (CPU) のコア数を増やすことで全体の計算性能が高まっている。動的計画法による解法は 10 年以上前から用いられているが，当時とは計算機のコア数も変わっているため計算方法もアップデートすることが重要である。

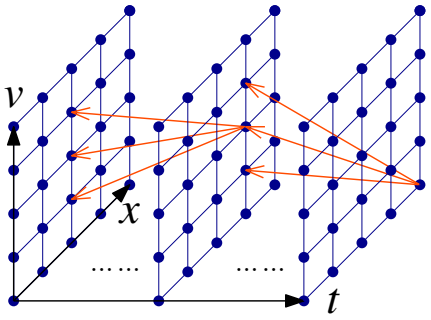


図 1　動的計画法の原理
Fig. 1.　Principle of Dynamic Programming

本稿では以下の 2 つの問題に取り組む。まず動的計画法による計算を並列化することで，並列化を行わない場合と比べて運転曲線最適化を高速化できることを検証する。次にその時短効果を限界まで高めるべくスーパーコンピュータを用いることで，市販 PC の場合と比べてさらに高速化できることを検証する。

## 2. 省エネルギー列車運転曲線の数理最適化

**〈2･1〉　動的計画法による運転曲線最適化の原理[(5)]**　省エネルギー列車運転曲線の最適化問題は，通常停車駅間ごとに消費エネルギー $E$ を最小化する軌道 $x(t)$ を求める問題として，以下の (1)–(4) のように定式化される。制約条件として，(2) の始端と終端における位置・速度，(3) の区間ごとの最高速度，(4) の加減速度の最大値などがある。

$$\text{minimize} \quad E \tag{1}$$

$$\text{subject to} \quad x(0) = 0, v(0) = 0, x(T) = L, v(T) = 0 \tag{2}$$

$$0 \le v(t) \le v_{max} \tag{3}$$

$$-\beta - R(x,v) \le \frac{dv}{dt} \le \alpha - R(x,v) \qquad (4)$$

ただし，$T$ は駅間走行時間，$L$ は駅間走行距離，$\alpha, \beta$ はそれぞれ車両の加速度と減速度の最大値，$R$ は走行抵抗である。

列車軌道 $x(t)$ は (5) の運動方程式に従い，消費エネルギー $E$ は (6) のように瞬時電力 $P$ を積分して求められる。

$$\frac{dv}{dt} = a(t) - r(v),\ \frac{dx}{dt} = v(t) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (5)$$

$$E = \int_0^T P(a,v)\mathrm{d}t \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (6)$$

ただし $a$ は制御入力である列車の加減速度である。

動的計画法では，この最適化問題を時間および空間的に離散化して近似的に解く。時間 $t$ 軸を離散化し，列車位置 $x$ と速度 $v$ からなる状態空間を格子状に分割して，図 1 に示すような $N$ 個の位相面を得る。各格子点は時刻 $T$ で終端状態へ到達するまでの累積消費エネルギーを評価値として持つ。その評価は，任意の時刻・状態から始まる最適軌道はそれより前の時間の軌道と無関係に定まる最適性原理を利用し，次の時刻の評価値と位相面間の消費エネルギー増分のみによって行う。具体的には時刻 $k\Delta t$ における格子点の位置・速度から，時間 $\Delta t$ の間制御入力 $a$ で加速し続けた次の時刻 $(k+1)\Delta t$ の格子点の評価値を参照する。これを制御入力 $a$ を変えて複数回行い，次の格子点の評価値と消費エネルギー増分の和が最小となるものが時刻 $k\Delta t$ での評価値となる。時刻 $T, T-\Delta t, \ldots$ の順に時間を遡って全ての格子点の評価を終えた後，始端状態から時刻 $0, \Delta t, \ldots$ の順に時間を進めて最適な制御入力 $a(k\Delta t)$ および列車軌道 $x(t)$ を得る。

動的計画法では終端状態の拘束条件を目的関数値に加算される (7) のようなペナルティとして扱う。このペナルティ値がステージ $N$ のエネルギー評価値となる。

$$\phi(x(T), v(T)) = c_x(x(T) - L)^2 + c_v v(T)^2 \quad \cdots\cdots\cdots\cdots\cdots (7)$$

また，微分方程式 (5)–(6) は台形積分などの数値積分によって解くことになる。

**〈2･2〉 並列計算の適用** 動的計画法による運転曲線最適化は並列計算に適する。時間方向のデータ依存性はあるが，空間方向のデータ依存性はないためである。すなわち，各格子点における計算では同時刻の他格子点のデータを必要としないため，同時刻の格子点は同時に計算できる。

この点を踏まえて，最適化計算プログラムは以下の 3 段階で構成できる。

始めに文献 (5) にならい式 (5) を離散化した差分方程式の解をメモ化する。具体的には格子点と制御入力の組ごとに探索点と補間計算に用いる係数をメモリに記憶する。差分方程式の求解は独立して行えることから，ここで並列化を行う。次の計算段階で大きく計算量を削減することができる。

次に時間軸逆方向に各格子点のエネルギー値を評価する。同じ時刻の格子点値は独立して評価できることから，ここでも並列化を行う。一般的にこの段階に最も計算時間を要する。

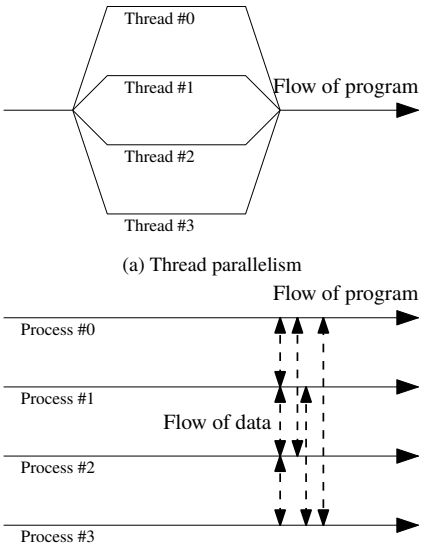


図 2 並列計算手法の概念図

Fig. 2. Concept of two parallel computing methods

最後に時間軸順方向に最適パスを探索する。この段階は全体に占める計算時間の割合が短いことから，並列化を省略する。

## 3. 並列計算の手法

鉄道分野でも用いられる並列計算の手法はいくつか存在するが[6]，本稿ではその中で OpenMP によるスレッド並列化と MPI によるプロセス並列化を採用する。

**〈3･1〉 スレッド並列 (OpenMP[7])** 実行の単位であるスレッドを同時に複数立ち上げる手法である。複数あるコアがそれぞれ別のスレッドを実行することで並列計算が可能となる。図 2a に示すようにプログラムの一部を並列化でき，スレッド間ではメモリが共有される。

スレッド並列を実現する API (Application Programming Interface) として OpenMP がある。OpenMP 仕様に対応したコンパイラがあれば数行の指示文を付け加えるだけで，自動的にループ処理をスレッド間で分担するなどの並列計算プログラムが出力され，簡単に並列計算を導入できる。

**〈3･2〉 プロセス並列 (MPI[8])** プログラムの単位であるプロセスを同時に複数立ち上げる手法である。プロセス間ではメモリが共有されず，各プロセスにデータが分散配置される。1 台のコンピュータには収まらないほど大きなデータを扱えるが，図 2b に示すようにプロセス間通信でデータを同期させなければ正しい結果を得られず，この通信時間がボトルネックとなることがある。

プロセス間通信の規格に MPI (Message Passing Interface) がある。プログラマが MPI 関数を呼び出し，プロセス間で計算データをやり取りする処理を挿入する必要がある。OpenMP と MPI を併用するハイブリッド並列化も可能である。

表 1　実験に使用した市販 PC の性能
Table 1.　PC performance

| | |
|---|---|
| CPU | AMD Ryzen 9 3900X (12 Cores 24 Threads, 3.8–4.6 GHz) |
| RAM | 64 GB |
| OS | Windows 10 21H2 |
| Compiler | Microsoft Visual C++ 2022 |

表 2　車両性能と路線条件
Table 2.　Train specifications and line conditions

| | |
|---|---|
| Maximum acceleration $\alpha$ | 3.3 km/h/s (0.917 m/s$^2$) |
| Maximum deceleration $\beta$ | 3.5 km/h/s (0.972 m/s$^2$) |
| Mass | 295 t |
| Resistance deceleration $r$ | $0.00001v^2 + 0.0002v + 0.056$ [km/h/s] |
| Traveling time $T$ | 100 s |
| Distance between stations $L$ | 1000 m |
| Maximum speed $v_{max}$ | 60 km/h |

表 3　シミュレーション条件と結果
Table 3.　Simulation cases and results

| | (1) Coarse | (2) Fine space | (3) Fine time |
|---|---|---|---|
| Phase interval $\Delta t$ | 0.5 s | 0.5 s | 0.25 s |
| Lattice interval $\Delta x$ | 0.125 m | 0.0625 m | 0.125 m |
| Lattice interval $\Delta v$ | 0.0625 km/h | 0.0625 km/h | 0.0625 km/h |
| Penalty coefficient $c_x$ | 8 | 8 | 8 |
| Penalty coefficient $c_v$ | 4 | 8 | 4 |
| The number of control inputs | 9 | 9 | 9 |
| Energy $E$ | 6.5267 kWh | 6.4249 kWh | 6.6432 kWh |
| Position error | 0.037 m | 0.050 m | 0.112 m |
| Speed error | 0.008 km/h | 0.000 km/h | 0.000 km/h |

## 4.　ケーススタディによる計算時間の評価

### 〈4･1〉　市販 PC を用いた計算時間高速化

**〈4･1･1〉　計算条件**　まず表 1 に示す市販 PC を使用して，運転曲線最適化問題の高速化効果を検証する。ここでは OpenMP によるスレッド並列化のみ行う。

車両性能と路線条件は表 2 に示す通りである。簡単のためモータ効率は 100%，回生効率は 0%で固定とし，き電回路も考慮せず加速に要する力学的パワーのみを積分して走行エネルギーとする。また，勾配や曲線はないものとする。ただし，これらはいずれも時間に依存しないため，考慮してもしなくても並列計算は妨げられない。

**〈4･1･2〉　結果と考察**　表 3 に示す時空間分割幅を変えた 3 つのケースに対して計算を実行した。ケース 1 を基準とし，ケース 2 では位置方向，ケース 3 では時間方向に分割幅を半分とした。

それぞれのケースにおいて並列数を変えた際の計算時間を図 3 に示す。計算時間は 5 回計測し最も短い時間をプロットした。スレッド数が 4 以下ではスレッド数にほぼ反比例して計算時間が短縮できている。しかしスレッド数を 5 以上に増

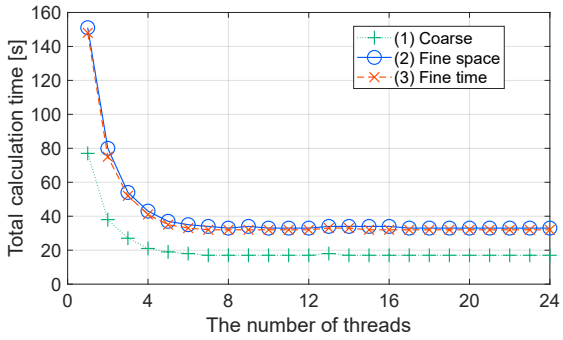


図 3　スレッド並列数を変えた PC の計算時間
Fig. 3.　Calculation time on the PC

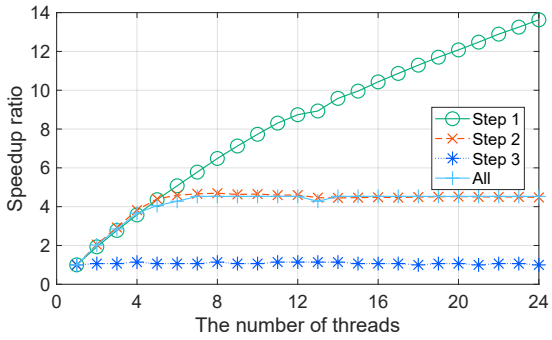


図 4　ケース 1(Coarse) の計算段階ごとの計算時間比
Fig. 4.　Calculation time ratio of (1) Coarse case for each calculation step

やしても計算時間の短縮効果があまりない。これは，並列化不可能な部分があると並列台数を増やしても時短効果が飽和する「アムダールの法則」[9] による限界と考えられる。

そこで，ボトルネックを特定するため 2.2 節で述べた 3 段階の処理それぞれの計算時間を取得した。その結果を 1 スレッド時の計算時間から何倍を短縮できたかを表す台数効果として図 4 に示す。第 1 段階「差分方程式のメモ化」は 24 スレッドまで時短効果が持続するが，第 2 段階「時間逆方向の評価」は 6 スレッドで飽和しており，これにより全体の処理時間が律速されている。第 3 段階「時間順方向の最適パス」は並列化していないため常にほぼ 1 倍であるが，全体に占める割合が小さいため合計時間に影響しない。

動的計画法の各時刻におけるエネルギー評価は，原理的に並列化可能であるにもかかわらず，実装上は並列化不可能な部分が生じる。その原因として，メモリアクセスの問題が考えられる。計算機は近いアドレスにあるメモリをキャッシュできるが，同じ格子点から参照される次の時刻の格子点は，それらのアドレスが離れたものとなり，キャッシュミスが多発することでメモリアクセスが渋滞する可能性がある。

最後にそれぞれのケースで得られた運転曲線を比較する。駅間走行エネルギーと終端状態の位置・速度誤差を表 3 に，軌跡を図 5 に示す。図 5 を見るといずれのケースも運転曲線の

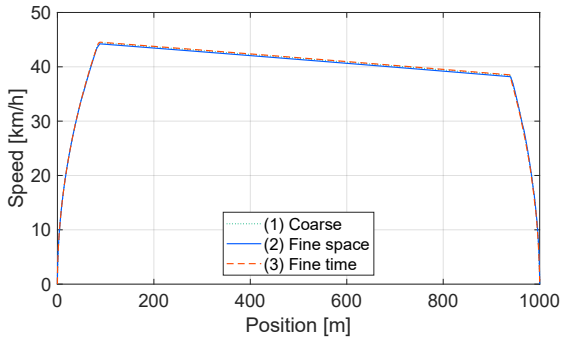


図 5　得られた最適運転曲線
Fig. 5.　Optimized speed profile

表 4　実験に使用したスーパーコンピュータの性能
Table 4.　Supercomputer performance

| | |
|---|---|
| CPU | Fujitsu A64FX (48 Cores, 2.2 GHz) |
| RAM | 32 GB per node |
| Compiler | Fujitsu Compiler |

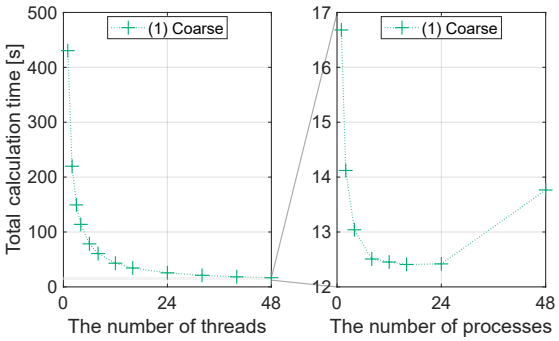


図 6　並列数を変えたスーパーコンピュータの計算時間
Fig. 6.　Calculation time on the supercomputer

概形は一致している。(2) Fine space は駅間走行エネルギーが (1) Coarse に比べ改善しており，解の精度と計算時間のトレードオフ関係に従っている。一方で (3) Fine time の駅間走行エネルギーは (1) Coarse に比べ増加しており，計算時間が長いにもかかわらず解の精度が悪化している。これは時間分割幅 $\Delta t$ に対する位置・速度の分割幅が大きくなったためと考えられる。

### 〈4･2〉　スーパーコンピュータを用いた高速化

**〈4･2･1〉　計算条件**　次に表 4 に示す性能を持つ東京大学情報基盤センターのスーパーコンピュータ Wisteria/BDEC-01 Odyssey を使用して，表 3 のケース (1) を計算した。OpenMP によるスレッド並列化のみの場合に加え、MPI によるプロセス並列化も併用する。併用時はプロセス数とスレッド数の積を 48 で一定とする。その他の条件は市販 PC と同様である。

**〈4･2･2〉　結果と考察**　計算時間の結果を図 6 に示す。左側はスレッド並列化のみ，右側はプロセス並列化の併用である。計算時間は 3 回計測し最も短い時間をプロットした。スレッド並列化のみの所要時間は市販 PC の場合よりも長いが，短縮効果は市販 PC と異なり飽和しなかった。その原因として，市販 PC と比べてスーパーコンピュータは CPU の動作周波数が低く，メモリ帯域幅が広いことが考えられる。また，プロセス並列化を併用することで市販 PC よりも計算時間を短縮できた。これらのことから，動的計画法は問題サイズを一定のまま並列度を上げても高速化効果を得られる問題であると言える。

## 5.　おわりに

運転曲線の省エネルギー最適化に用いられてきた動的計画法は，解の精度を高めるために計算量が多くなる課題があった。そこで本稿では動的計画法による運転曲線最適化を高速化する並列計算手法を提案した。市販 PC でスレッド並列化を行い，実行スレッド数を増やすと計算時間が短縮できることを検証した。また，スーパーコンピュータでハイブリッド並列化を行えば，時短効果が飽和することなく市販 PC よりも計算時間を短縮できることを確認した。今後はこの高速化効果を活用し，並列計算により大規模な運転曲線最適化問題を現実的な時間内に解けることを実証していきたい。

**謝辞**　本研究は東京大学情報基盤センターの FUJITSU Supercomputer PRIMEHPC FX1000 および FUJITSU Server PRIMERGY GX2570 (Wisteria/BDEC-01) を用いて実施した。

## 文　献

（1）　国土交通省鉄道局：鉄道分野のカーボンニュートラル加速化検討会 (第 1 回資料), `https://www.mlit.go.jp/tetudo/content/001474317.pdf` (2022).
（2）　木村幸男, 古賀澄夫 : 短区間を比較的高速で走行する通勤形電車の省エネルギー運転方法, 計測自動制御学会論文集, Vol. 20, No. 4, pp. 357–360 (1984).
（3）　大場直樹, 宮武昌史：固定閉塞式信号システムの影響を考慮した列車の省エネルギー運転曲線導出, 電気学会論文誌D（産業応用部門誌）, Vol. 138, No. 4, pp. 282–290 (2018).
（4）　渡邉翔一郎, 古関隆章, 野田慶親, 宮武昌史：リニア駆動鉄道の最適省エネルギー運転曲線, 電気学会論文誌D（産業応用部門誌）, Vol. 137, No. 1, pp. 44–52 (2017).
（5）　高英聖, 古関隆章, 宮武昌史：動的計画法を用いた列車運転曲線最適化問題の求解法, 電気学会論文誌D（産業応用部門誌）, Vol. 125, No. 12, pp. 1084–1092 (2005).
（6）　Wu, Q., Spiryagin, M., Cole, C. and McSweeney, T.: Parallel computing in railway research, *International Journal of Rail Transportation*, Vol. 8, pp. 111–134 (2020), doi: 10.1080/23248378.2018.1553115.
（7）　`https://www.openmp.org/`.
（8）　`https://www.mpi-forum.org/`.
（9）　Amdahl, G. M.: Validity of the single processor approach to achieving large scale computing capabilities, in *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference* (1967).