Vol.2018-HPC-167 No.2
2018/12/17

医用画像処理における LDDMM の並列化とコード最適化

中島　大地†，田村 友輝††，物部 峻太郎††，本谷 秀堅††，片桐　孝洋†††，永井　亨†††，荻野 正雄†††

医用画像処理では，データ容量と計算量が多い 3 次元画像を取り扱う．医用画像処理で用いられる大変形微分同相写像（Large Deformation Diffeomorphic Metric Mapping, LDDMM)の計算部分は演算量が多いため，コード最適化および並列化の適用が期待されている．そこで本研究では，LDDMM による医用画像処理プログラムにおいて，コード最適化と並列化を施すことで実行時間の短縮を目指す．並列化では MPI(Message Passing Interface)と OpenMP を利用し，コード最適化ではループ融合手法と OpenMP のスケジューリング方式を活用した．性能評価環境として名古屋大学設置の HPC である FX100 スーパコンピュータシステムを利用して，提案手法による性能評価を行った．性能評価の結果，オリジナルコードに対して，8 ノード利用時に 10.7 倍の速度向上を達成した。

1. はじめに

医用画像[1]はレントゲン写真から始まり，1970 年代には X 線 CT や MRI，PET などが登場し，近年では分子画像や３次元 CT，４次元 CT などの実用化といった発展がなされてきた．このような医用画像の発展と共に，人が処理することが可能な画像量の限界が来ている．そこで我々はコンピュータを用いた医用画像処理の技術を向上させることで，この難局面への対応を試みる．

医用画像処理では，3 次元の画像を利用する場合がある．3 次元の医用画像の利用のためには，癌や臓器といった，対象物の統計モデルが必要となる．しかし現状では，CT などによる症例データの不足から，統計モデルを作成するための十分な教師データが存在しない．

そこで臓器領域間の微分同相写像を行うことで，限られた数の教師データを適切に内挿させ，教師データを生成する [2]．また異なる患者の臓器間の写像を生成することで，それらの患者の画像が教師データであるならば、新たに写像によって生成した画像も教師データとして利用可能とする．この様な教師データの生成手法は，統計モデルの構築において有効となる．よって我々は教師データを生成可能なプログラムとして，臓器領域間の大変形微分同相写像を求めるプログラム LDDMM コードを作成した．コードは，LDDMM (Large Deformation Diffeomorphic Metric Mapping) 法に基づいた変分問題を最急降下法により求めるプログラムである．

LDDMM コードでは，3 次元の医用画像を取り扱う性質上計算量が大きくなり，教師データの組み合わせの数だけ内挿・補間するには効率的なコードが必要となる．そこで本研究では，医用画像処理を行うプログラムである LDDMM コードの並列化とコード最適化を行い，実行時間の短縮を目指す．LDDMM コードにより生成された画像を図 1 に示す．

本研究では，LDDMM コードにおいて実行時間の大きな割合を占める，反復計算部分である LDDMM の演算に対する並列化とコード最適化を実装した．並列化の実装には，MPI (Message Passing Interface)および OpenMP を用いた．コード最適化では，ループ融合，スケジューリングを用いて性能向上を行った．LDDMM コードの並列化とコード最適化の効果を検証するため，名古屋大学設置のスーパコンピュータ Fujitsu PRIMEHPC FX100 を用いて性能評価を行った．

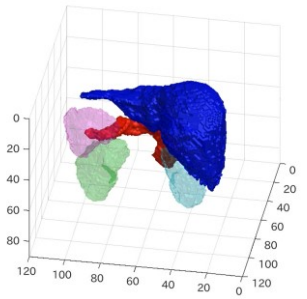

図 1　LDDMM コードによる生成画像

本報告は以下の構成とする．2 章では，LDDMM コードについて，3 章では並列化及び最適化についての説明を行う．4 章では，FX100 を用いて性能評価を行った．最後にまとめを記述する．

2. LDDMM

2.1 概要

本研究で利用する LDDMM コードは，LDDMM 法[2]に基づいたアルゴリズムと，最急降下法による最適なベクトル場の推定が利用されている．この章では LDDMM 法と最急降下法の説明及び，その活用について記載する．

2.2 LDDMM とは

LDDMM では，2 枚の入力画像I_0, I_1に対して，I_0からI_1へ滑らかでかつ全単射な写像を生成する．画像が定義される領域は$\Omega \in R^n$(n は領域の次元数)で定義され，写像 φ は φ: $\Omega \rightarrow \Omega$ の式が成り立つ．この時，画像 I の輝度は $I: \Omega \rightarrow R^d$(d=1 はグレースケール，d=3 はカラー画像，d=5 は拡散

†　名古屋大学 大学院情報学研究科情報システム学専攻
††　名古屋工業大学大学院 情報工学専攻
†††　名古屋大学 情報基盤センター

テンソル画像)によって示される．

写像は微分可能でかつ連続であり，I_0からI_1に写像を行う．よって，I_0をテンプレート画像，I_tを時間 t 後の変位画像として，それぞれのある座標をx_0, x_t，時間 t 後の写像をϕ_tと表すと，式(1)が成り立つ．

$$x_t = \varphi_t(x_0) \quad \cdots(1)$$

式(1)が示すように，2 枚の入力画像からの写像によって，新たな画像の生成が可能となる．LDDMM コードでは，このアルゴリズムを利用して，画像の生成を行う．例としてI_0，I_1を入力画像，$I_{0.25}$，$I_{0.5}$，$I_{0.75}$を成画像とした図を図 2 に示す．

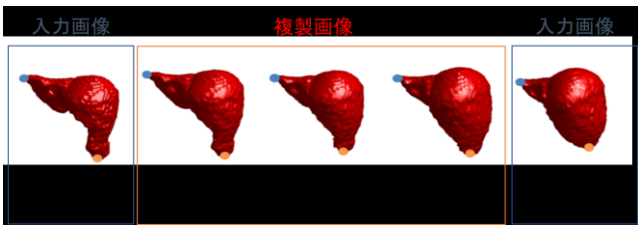

図 2　肝臓平均形状から肝臓症例への LDDMM

また写像は，時間変化するベクトル場を$v: \Omega \to R^n$として利用することで，式(2)のようにあらわすことができる．なおv_tはφ_tの時間微分であり，φ_0は恒等写像，$t \in [0,1]$である．

$$\varphi_t(x) = \int_0^t v_t \, dt + \varphi_0 \quad \cdots(2)$$

式(2)が示すように，写像ϕはベクトル場 v を求めることで，導出することが可能となる．

2.3 単位変形ベクトル場

変形単位ベクトル$\hat{v}$は式(3)で示される．

$$\hat{v} = \underset{v}{argmin} \left(\int_0^1 ||v_t||_V^2 \, dt + \frac{1}{\sigma^2} ||I_0 \circ \varphi_1^{-1} - I_1||_{L^2}^2 \right) \quad \cdots(3)$$

右辺第 1 項$||v_t||_V$は滑らかなベクトル場を生成するための正則項であり，第 2 項$\| I_0 \circ \varphi_1^{-1} - I_1 \|_{L^2}$は 2 つの画像の二乗誤差である．写像が微分同相な写像とするためには，ベクトル場が滑らかなものを選択しなければならない．

画像I_0からI_1への変形単位ベクトルを，式(4)に示すエネルギー関数$E(v)$として抽出する

$$\hat{v} = \underset{v}{arginf} \, E(v) \quad \cdots(4)$$

式(4)の最小化問題は，式(5)に示す Euler-Lagrange 方程式を満たす．

$$2\hat{v}_t - K\left(\frac{2}{\sigma^2} \left| D\varphi_{t,1}^{\hat{v}} \right| \nabla J_t^0 (J_t^0 - J_t^1) \right) = 0 \quad \cdots(5)$$

このとき$J_t^0 \doteq I_0 \circ \phi_{t,0}$，$J_t^1 \doteq I_t \circ \phi_{t,1}$，$\phi_{s,t} = \varphi_t \circ \left(\phi_s \right)^{-1}$である．

また K は $K: K^2(\Omega, R^d) \to V$の条件を満たすコンパクト自己随伴作用素であり，一意に$< a, b >_{L^2} = < Ka, b >_V$と定義される．

変形ベクトル$v \in L^2([0,1], V)$を$h \in L^2([0,1], V)$方向に向けてε摂動させる．エネルギー関数であるガトー微分$\partial E(v)$はフレシェ微分$\nabla_v E$を用いることで表すことができる．

$$\partial_h E(v) = \lim_{\varepsilon \to 0} \frac{E(v + \varepsilon h) - E(v)}{\varepsilon} = \int_0^1 < \nabla_v E_t, h_t >_V \, dt \quad \cdots(6)$$

式(3)の第 1 項，第 2 項をそれぞれエネルギー関数E_1，E_2と表す．エネルギー関数$E_1(v) = \int_0^1 ||v_t||_V^2 \, dt$の微分はフレシェ微分より，式(7)と表される

$$\partial_h E_1(v) = 2 \int_0^1 < v_t, h_t >_V \, dt \quad \cdots(7)$$

同様にエネルギー関数$E_2(v) = \frac{1}{\sigma^2} ||I_0 \circ \varphi_{1,0}^{-1} - I_1||_{L^2}^2$も，式(8)と表される．

$$\partial_h E_2(v) = \frac{1}{\sigma^2} < I_0 \circ \varphi_{1,0}^v - I_1, DI_0 \circ \varphi_{1,0}^v \, \partial_h \varphi_{1,0}^v >_{L^2}$$

$$A) = \frac{2}{\sigma^2} < I_0 \circ \varphi_{1,0}^v - I_1, DI_0 \circ \varphi_{1,0}^v \times (-D\varphi_{1,0}^v \int_0^1 \left(D\varphi_{1,t}^v \right)^{-1} h_t \circ \varphi_{1,t}^v dt) >_{L^2}$$

$$B) = \frac{-2}{\sigma^2} \int_0^1 < (I_0 \circ \varphi_{1,0}^v - I_1, D(I_0 \circ \varphi_{1,0}^v) \times \left(D\varphi_{1,t}^v \right)^{-1} h_t \circ \varphi_{1,t}^v) >_{L^2} dt \quad \cdots(8)$$

式(8)A)では$\partial_h \varphi_{s,t}^v = D\varphi_{s,t}^v \int_s^t \left(D\varphi_{s,u}^v \right)^{-1} h_t \circ \varphi_{s,u}^v du$，式(8)B)では$D(I_0 \circ \varphi_{1,0}^v) = DI_0 \circ \varphi_{1,0}^v D\varphi_{1,0}^v$を用いる．

また$\phi_{1,t}^v(y) = x, \phi_{1,t}^v(x) = y$を用いて，ヤコビアン変数変換$\left| D\phi_{t,1}^v(x) \right| dx = dy$を得る．$\varphi_{1.0}^v \to \varphi_{1.0}^v \circ \varphi_{t,1}^v = \varphi_{t,0}^v$　とヤコビアン変数変換より，式(8)は式(9)で表される．

$$\partial_h E_2(v) = \frac{-2}{\sigma^2} \int_0^1 < |D\varphi_{t,1}^v| (I_0 \circ \varphi_{t,0}^v - I_1 \circ \varphi_{t,1}^v, D(I_0 \circ \varphi_{t,0}^v) h_t >_{L^2} dt$$

$$= \frac{-2}{\sigma^2} \int_0^1 < |D\varphi_{t,1}^v| (j_t^0 - j_t^1) \nabla j_t^0, h_t >_{L^2} dt$$

$$= \int_0^1 < K\left(\frac{2}{\sigma^2} \left| D\varphi_{t,1}^v \right| \nabla J_t^0 (J_t^0 - J_t^1) \right), h_t >_V dt \quad \cdots(9)$$

よってエネルギー関数E_1，E_2よりよりエネルギー関数の勾配は以下のように求められる．

$$(\nabla_v E_t)_v = 2v_t - K\left(\frac{2}{\sigma^2} \left| D\varphi_{t,1}^v \right| \nabla J_t^0 (J_t^0 - J_t^1) \right) \quad \cdots(10)$$

$(\nabla_v E_t)_v$における V は空間 V における勾配である．よって最適なベクトル場は Euler-Lagrange 方程式を満たす．

$$\partial_h E(\hat{v}) = \int_0^1 < 2\hat{v}_t - K\left(\frac{2}{\sigma^2} \left| D\varphi_{t,1}^{\hat{v}} \right| \nabla J_t^0 (J_t^0 - J_t^1) \right), h_t >_V dt = 0 \quad \cdots(11)$$

h は$L^2([0,1], V)$を満たす．この式（11）を用いることで式に最適なベクトル場が計算される．

2.4 最急降下法

最適なベクトル場を計算するために，最急降下法を用いる．$t_j \in [0,T]$，$j \in [0,N]$，$T=N\times\delta t$とし，時刻t_jは時間間隔δtをj回繰り返した時刻とする．ベクトル場$v_{t_j}^k(y)$及び写像$\varphi_{t_j}^k(y)$は最急降下法をk回適応させ，時刻t_jで得られたものとする．最急降下法は式(12)で示される．

$$v^{k+1} = v^k - \varepsilon\nabla_{v_{t_j}^k} E \qquad \ldots(12)$$

また式(10)を離散化した式を式(13)で示す．

$$\nabla_{v_{t_j}^k} E_{t_j}^k(y) = 2v_{t_j}^k(y) - \frac{2}{\sigma^2}K(\left|D\varphi_{t_j,T}^k(y)\right| DJ_{t_j}^0(y) * (J_{t_j}^0(y) - J_{t_j}^T(y))) \qquad \ldots(13)$$

2.5 LDDMM コードへの適用

LDDMM コードでは LDDMM 法，最急降下法を用いることで写像を導き出している．コードは以下の 4 ステップで構成される．

I. ヤコビアンの演算
II. Backward Integration
III. 変形ベクトル場 v の更新
IV. φ と位置情報の更新

ステップ I～III の各ステップと式(12)と式(13)の対応を，図 3 に示す．

図 3 I~III のステップとの対応

LDDMM コードはφを導出し，画像の生成を行うために，各ステップにおいて，N(位置情報の数)×T(時間間隔)の計算と，4 ステップ全体に対しての反復計算が必要となる．そのため，LDDMM コードでは N×T×G(反復回数)の多大な計算が必要となる．またプロファイルの結果，上記の 4 ステップのうち，ステップⅢは実行時間の約 60%を占めていることが判明している．次節で説明する並列化は，これら 4 ステップに対して実装されており，コード最適化は，実行時間に占める割合が大きいステップⅢに実装した．

3. 並列化およびコード最適化

3.1 概要

本研究では，医用画像を生成する LDDMM コードの並列化を実装するとともに，様々な最適化手法を適用させることで実行時間の短縮を目指す．この章では並列化と最適化についての説明を行う．

3.2 並列化の詳細

LDDMM コードには並列化手法として，MPI と OpenMP による Hybrid MPI/OpenMP 並列化手法を実装した．

3.2.1 MPI による並列化

MPI による並列化として，以下の手順で実装を行った．

I. MPI の送受信をするためのメモリ確保
II. MPI プロセスごとに，ループの長さを調整し計算を実施
III. 計算したデータを送信バッファに格納
IV. 送信するデータを，MPI_ALLGATHER を用いて送受信
V. 送信したデータを受信バッファに格納し利用

以上の手順を行うことで，容易に異なるプロセス間での通信を行い並列化が可能となる．

以下の図４，図５において，LDDMM コードの並列化を説明する．

```
//2  :MPI プロセスごとに，ループ処理の長さを調整し，
     計算を実施
for (int n=nhead; n<=ntail; n++) {
  unsigned int c = t*9*(ntail-nhead+1)+(n-nhead)*9;
  float  w1x  =LINT::linterp3(L[t][n].x[0]+1,L[t][n].x[1],
            L[t][n].x[2],B[t].v[0],x,y,z);
    ・・・・・・・・・
  float  w6z  =LINT::linterp3(L[t][n].x[0],L[t][n].x[1],
            L[t][n].x[2]-1,B[t].v[2],x,y,z);
//3: 計算したデータを送信バッファに格納
  nsend[c++] = (w1x-w4x)/2;
    ・・・・・・・・・
  nsend[c++] = (w3z-w6z)/2;
}
```

図 4 LDDMM コードにおける MPI の利用(1/2)

```
//4: 送信するデータを，MPI_ALLGATHER を用いて
     送受信する
MPI_Allgather(nsend,itmp[2],MPI::FLOAT,nrecv,itmp[2],
     MPI::FLOAT,MPI_COMM_WORLD);
//5: 送信したデータを受信バッファに格納し，利用する
#pragma omp parallel for private(c) firstprivate(cc)
for(int i=0; i<pe*2; i=i+2){
  cc = i/2;   c = itmp[2]*cc;
  for(unsigned int t=0;t<M;t++){
    for(int n=order[i];n<=order[i+1];n++) {
      L[t][n].Dv[0][0] = nrecv[c++];
        ・・・・・・・・・
      L[t][n].Dv[2][2] = nrecv[c++];
}}}
```

図 5 LDDMM コードにおける MPI の利用(2/2)

図 4，5 における nhead，ntail は各プロセスのループ開始インデックス，終了インデックスを示している．このインデックスの計算を図 6 に示す．ここで，p は MPI プロセス数，i は MPI のランク数，x はループ長である．

```
if ( x % p - i >= 0 )
   nhead = ( x / p ) * ( i – 1 ) + 1;
else
   nhead = ( x / p ) * ( i – 1 ) ;
if ( i != p – i )
   ntail = ( x / p ) * i;
else
   ntail = x;
```

図 6 nhead，ntail の定義

3.2.2 OpenMP

OpenMP による並列化として，for ループに parallel 構文の適用を行った．以上の実装により，異なるスレッド間でのタスクを分散させることで，タスクの並列化が可能となる．図 7 に LDDMM コードにおける実装例を示す．

```
#pragma omp parallel for private(j,k)//i についてスレッド並列化する
for (int i=0; i<(int)x; i++) {
   for (unsigned int j=0; j<y; j++) {
      for (unsigned int k=0; k<z; k++) {
         B[t+1].phi[0][i][j][k] = B[t].phi[0][i][j][k] +
               Delta*V0[0][i][j][k];
            ・・・・・・・・・
         B[t+1].phi[2][i][j][k] = B[t].phi[2][i][j][k] +
               Delta*V0[2][i][j][k];
} } }
```

図 7 LDDMM コードにおける OpenMP の利用

図 7 から，LDDMM コードでは，画像サイズに関するループ i, j, k についてデータ依存がないため，どのループでも OpenMP の parallel for 構文が適用できる．図 7 では，ループ i に適用している．

3.3 コード最適化

前節までの手順により実装された MPI を用いた並列化は，ノード内のコード最適化の観点で十分ではない．そこでループ融合とスケジューリングのコード最適化手法を適用し，演算効率の向上を目指す．

3.3.1 ループ融合

ループ融合は，複数のループを一まとまりにすることで，ループ長を増加させ，スレッド並列化効率やデータプリフェッチ効率を高めることで高速化するコード最適化手法の一つである．融合対象が 3 重ループである場合，1 重ループ化，および，2 重ループ化によるループ融合が実装できる．ループ融合は，以下の手順で取り扱う．

I. ループのループ長を，融合させる長さ分増加
II. 融合したループを消滅
III. 消滅したループ内において，元のインデックスを復元するための変数を定義

図 8 は実装前の 3 重ループである．図 8 を 2 重ループ融合（i，j ループの融合），3 重ループ融合（i，j，k ループの融合）させたものをそれぞれ，図 9 および図 10 で示す．

```
for (unsigned int i=0; i<x; i++) {
   for (unsigned int j=0; j<y; j++) {
      for (unsigned int k=0; k<z; k++) {
         //演算処理 }}}
```

図 8 ループ融合実装前コード

```
for (unsigned int ij=0; ij<x*y; ij++){
   unsigned int i = ij / y;
   unsigned int j = ij % y;
   for(int k=0; k<z; k++){
      //演算処理    }}
```

図 9　2 重ループ融合コード

```
for (unsigned int kk=0; kk<x*y*z; kk++) {
   unsigned int i = kk / y*z;
   unsigned int j = (kk / z) % y;
   unsigned int k = kk % z;
    //演算処理 }
```

図 10　3 重ループ融合コード

ループ融合によりループ長が伸びることで，高スレッド実行時の負荷分散の均等化と，コンパイラによる最内ループのプリフェッチ処理の増進といった，速度向上に資するメリットが得られる．

3.3.2　スケジューリング

OpenMP の parallel 構文では，対象ループの範囲をスレッド個数分に分割して，並列処理を行っている．スケジューリングは，最適な割り当て間隔(チャンクサイズ)を明示的に指示することで負荷分散を改善させるコード最適化手法の一つである．実装は補助指定文である schedule(dynamic) を用いて行われる．実装による変化を表 1 にまとめる．

LDDMM コードにおいては，並列化部分の一部において if 文が用いられている．これは if 文の分岐によってスレッドごとに計算量が異なるため，負荷の均等化を妨げる要因となる．よってこの並列化部分のスケジューリングを行うことで，コードの最適化を図る．ただし動的なスケジューリングはシステムのオーバヘッドが存在するため、実行時

のチャンクサイズのチューニングが必須となる.

表 1 スケジューリングの実装による変化点

	実装前	実装後
形式	静的 static	動的 dynamic
チャンクサイズ	ループ長/スレッド数（デフォルト値）	指定する

4. 性能評価

4.1 問題設定

本実験では問題サイズ（3 次元画像サイズ）XxYxZ を 100x100x100 に固定し，実験を行った.

4.2 実験環境

本実験で使用した計算機 FX100 についての構成を表 2 に示す.

表 2 FX100 のスペック

ハードウェア構成	
CPU	
プロセッサ	Fujitsu SPARC64 XIfx
動作周波数	2.2GHz
コア数/ノード	32(+2 アシスタントコア)
演算性能/ノード	1.126 TFLOPS
全体性能	
総ノード数	2880
総理論演算	3.2FLOPS
ソフトウェア構成	
開発環境	C/C++
MPI 通信ライブラリ	富士通 MPI
コンパイラ環境	
mpiFCCpx -std=c++11 -SCALAPACK -SSL2 -Kfast -Kopenmp -g -o 実行ファイル コード -lm	

4.3 実験設定(ループ融合)

ループ融合では，従来法(コード最適化なし)，提案法 1(2 重ループ融合)，および，提案法 2(3 重ループ融合)，の 3 つの実装方式を実装し，スレッド数比較，ピュア MPI における MPI プロセス数比較，ハイブリット実行の比較による性能評価を行った.

スレッド数比較においては 1 ノードにおいてスレッド数を 1-32 に変化させることで性能評価を行った．ピュア MPI における MPI プロセス数比較においては、1 ノードのピュア MPI においてプロセス数を 1-32 に変化させることで性能評価を行った．ハイブリット実行の比較では，複数の MPI プロセスとスレッドを用いた評価である．設定は 8 ノードに固定化し，(MPI プロセス数 P×スレッド数 T）と表記するとき，(8Px32T), (16Px16T) ，(32Px8T) ，(64Px4T) ，(128Px2T)，((256Px1T):ピュア MPI 実行)とした場合と，スレッド数を 32 に固定して MPI プロセス数を 1-32 に変化させた場合の 2 種類で行った．また反復計算部分の反復回数は，スレッド数比較，ピュア MPI における MPI プロセス数比較では 50 回, ハイブリット実行の比較では 200 回とした.

4.4 実験結果(ループ融合)

4.4.1 スレッド数比較

図 11 にスレッド数の増加による台数効果を示す．図 11 において，ループ融合なし，2 重ループ融合，3 重ループ融合においてそれぞれ，7.5 倍(29 スレッド) ，6.6 倍(31 スレッド)，7.8 倍(32 スレッド)の速度向上を確認した.

図 12 に，スレッド数の増加による効果を示す．ループ融合の最適化とループ融合がない場合と比較して，2 重ループ融合は常に性能の悪化を示し，3 重ループ融合は性能の向上と悪化の両方を示した.

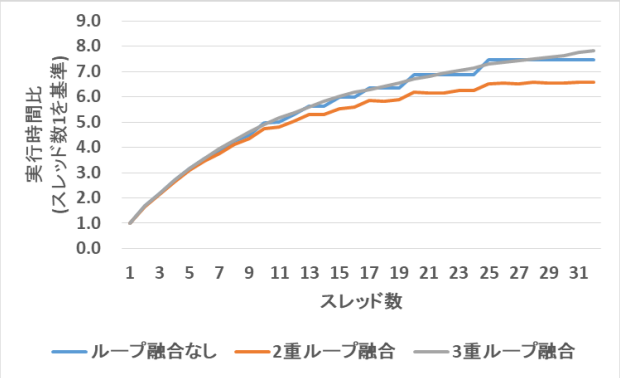

図 11 スレッド数の増加による効果(台数効果)

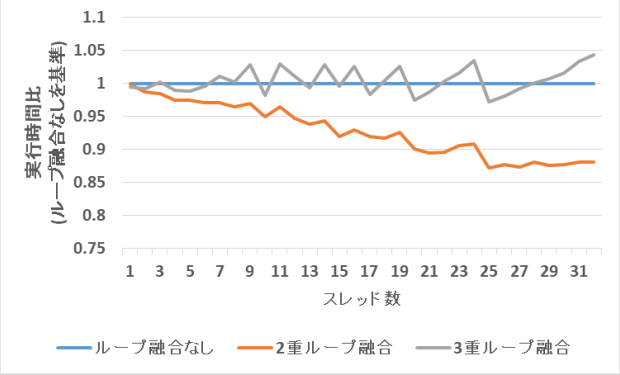

図 12 スレッド数の増加による効果(速度向上率)

4.4.2 ピュア MPI における MPI プロセス数比較

ピュア MPI におけるプロセス数の増加による台数効果を図 13 に示す．ループ融合なし，2 重ループ融合，3 重ループ融合においてそれぞれ，6.5 倍(25 プロセス)，6.8 倍(32 プロセス)，6.6 倍(32 プロセス)の高速化を確認した.

図 14 に，ピュア MPI 実行における MPI プロセス数増加による効果を示す．ループ融合がない場合と比較して，2 重ループ融合，3 重ループ融合ともに性能の向上と悪化の両方を示した．速度向上率はそれぞれ最大で，1.09，1.07 倍を示した．

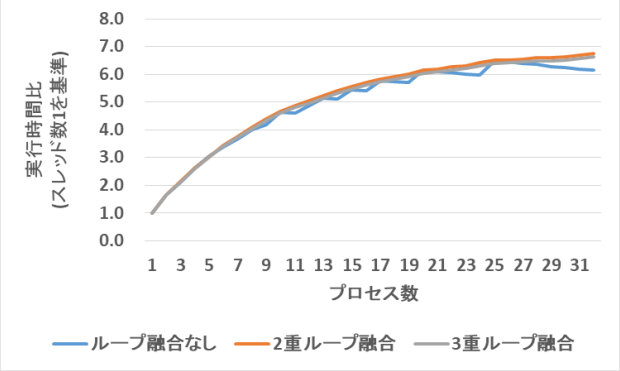

図 13 ピュア MPI 実行における MPI プロセス数の増加による効果 (台数効果)

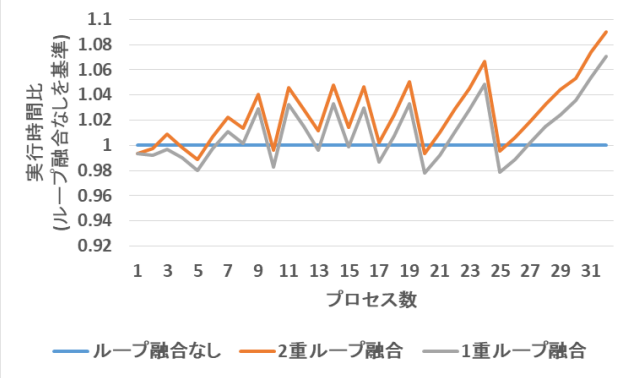

図 14 ピュア MPI 実行における MPI プロセス数の増加による効果 (速度向上率)

4.4.3 ハイブリット MPI 実行の比較(ノード数固定)

図 15 に，ハイブリット MPI 実行のプロセス数とスレッド数の組み合わせによる効果(ノード数固定)を示す．図 15 から，ハイブリット MPI 実行時(ノード数固定)では，最も実行時間が速いものは 3 重ループ融合において(16Px16T)の場合の 600[秒]であった．2 重ループ融合の時も同様に，(16Px16T)の場合において 615[秒]，ループ融合無しの場合は，(8Px32T) の場合において 633[秒]の最速実行時間を示した．また図 15 に示すように，ピュア MPI 実行に近づくほど，ループ融合の効果が大きいことが示された．

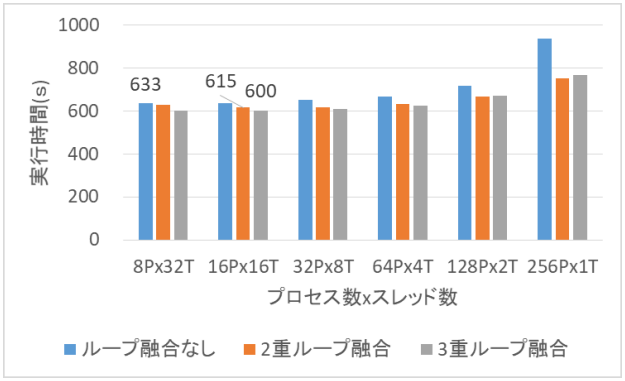

図 15 ハイブリット MPI 実行のプロセス数とスレッド数の組み合わせによる効果(ノード数固定)

4.4.4 ハイブリット MPI 実行の比較(スレッド数固定)

図 16 に，ハイブリット MPI 実行の比較における MPI プロセス数の増加による効果 (スレッド数固定)を示す．図 16 では，2 重ループ融合 MPI プロセス数 9 未満では性能の悪化を示し，9 以上では性能の向上を示した．3 重ループ融合は常に性能の向上を示した．それぞれ MPI プロセス数 20 の場合に最速である 1.15 倍，1.18 倍の性能向上を示した．

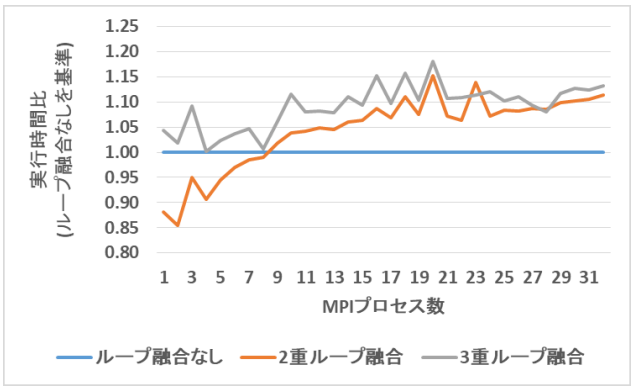

図 16 ハイブリット MPI 実行の比較における MPI プロセス数の増加による効果 (スレッド数固定)

4.5 実験設定(スケジューリング)

スケジューリングでは，様々なチャンクサイズ(1,10,20,…,100,200,…,1000,1500,2000,3000)に対して，動的なスケジューリングを行い，スケジューリングの実装を行わなかったものと比較して性能評価を行った．ただし両コードに対して，前節で最も実行時間が早かった 3 重ループ融合を施したものを対象とした．利用したノード数は 8 ノードとし，最短の実行時間を記録した MPI プロセス数 16，スレッド数 16 による性能評価を行った．反復計算部分の反復回数は，200 回とした．

4.6 実験結果(スケジューリング)

図 17 に，チャンクサイズを変化させることによる動的スケジューリングの効果を示す．図 17 が示すように，動

的スケジューリングを用いることで，チャンクサイズ 10-1500 間では実装を施さない場合（静的スケジューリング，チャンクサイズはデフォルト値）と比較して，高速化が確認された．最も実行時間が速かったものは，チャンクサイズ 70 の場合であり，実行時間は 588[s]であり 1.03 倍の性能向上を確認した．

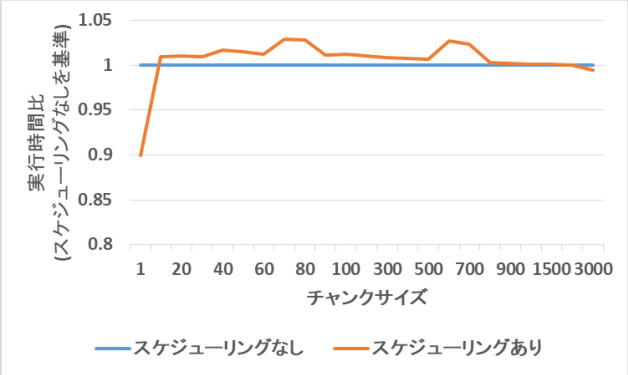

図 17　チャンクサイズを変化させることによる動的スケジューリングの効果

5. おわりに

本研究では，医療画像処理コードに、コード最適化技法，および MPI 並列化を適用した実装方式を提案した．提案手法であるコード最適化(ループ融合・スケジューリング)は有効であるといえる．

性能評価により，2 重ループ融合はプロセス数の増加が，3 重ループ融合はスレッド数の増加が，性能に大きく影響することが判明した．ハイブリット MPI を用いる場合，一定スレッド(8 スレッド)以上の実行が望ましい．またハイブリット MPI を用いることで，2 重ループ融合，3 重ループ融合では，最高で 1.15 倍，1.18 倍の高速化が可能となった．一方，動的スケジューリングを用いることで，1.03 倍の高速化が達成できた．

1 ノード 1 プロセス 1 スレッドの場合と，最適化手法を施した(3 重ループ融合･スケジューリング(チャンクサイズ 70))8 ノード 16 プロセス 16 スレッドの場合の比較を行った．反復計算回数の 1 回当たりの実行時間はそれぞれ．31[s], 2.9[s]を観測した．これは並列化と最適化手法により 10.7 倍の高速化が実現した．オリジナルの LDDMM コードを実行した場合は約 18 日の実行時間が必要なのに対して，最適化したコードは約 40 時間となることを意味しており，実行時間の劇的な短縮が可能となる．

本実装では，ループ融合などのコード生成に加えて，チャンクサイズなど，性能に関連するパラメタが存在する．これらコード生成と性能パラメタの調整を行い，幅広い計算機環境でも高性能を達成する技術が，自動チューニング（AT）技術[4]である．また，チューニングのためのコードを自動生成できる AT 言語として ppOpen-AT[5]が知られている．本稿で取り扱った高性能化実装方式について，AT 技術の適用、および AT 言語の ppOpen-AT の適用することは重要な今後の課題である．

謝辞

本研究の一部は，科学研究費補助金新学術領域研究（研究領域提案型）（課題番号：17H05290），科学研究費補助金基盤研究（B）（課題番号：18H03262），および，学際大規模情報基盤共同利用・共同研究拠点、および、革新的ハイパフォーマンス・コンピューティング・インフラの支援による(課題番号: jh180027-DAJ).

参考文献

[1]日本生体医工学会，画像情報処理(Ⅰ)-解析・認識編-，コロナ社(2005)

[2] Grenander, Ulf and Miller, Michael I and others ， Pattern theory: from representation to inference， Oxford university press(2007)

[3] Mirza Faisal Beg，Alain Trouvé，Michael Miller，Laurent Younes，Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms，International Journal of Computer Vision，https://www.researchgate.net (2005)

[4] T. Katagiri and D. Takahashi, Japanese Auto-tuning Research: Auto-tuning Languages and FFT, Proceedings of the IEEE, Vol. 106 , Issue 11, pp. 2056-2067 (2018)

[5] T. Katagiri, S. Ohshima, M. Matsumoto, Auto-tuning on NUMA and Many-core Environments with an FDM code, Proceedings of IEEE IPDPSW2017, pp.1399-1407 (2017)