Vol.2016-HPC-157 No.17
2016/12/22

アジョイント法における Forward model への階層ブロッキング適用による高性能化

池田 朋哉†, 伊藤 伸一††, 長尾 大道†3, 片桐 孝洋†4, 永井 亨†4, 荻野 正雄†4

本報告では，非逐次型データ同化手法であるアジョイント法の Forward model の計算に対して時空間ブロッキングを適用し，さらに複数 Forward model の計算に対してブロッキングを適用する 2 段の階層ブロッキングの効果を検証した．Fujitsu PRIMEHPC FX100 を利用し，2 次元格子点の初期状態を推定する双子問題を用いて階層ブロッキングの性能評価を行った．時空間ブロッキング適用により，ブロッキング適用前の Forward model の計算における実行時間を 1 とした時に 1.47 倍の速度向上を達成した．また複数 Forward model を計算するブロッキングの適用により，上限値として 1.78 倍の速度向上を達成し，さらに階層ブロッキングを適用することで，階層ブロッキング適用前の Backward model の計算を除くアジョイント法の実行時間を 1 とした時に 1.46 倍の速度向上を達成した．また性能プロファイル結果によると，時空間ブロッキング適用により L1 キャッシュミス数は 30%, L2 キャッシュミス数は 32%削減されメモリアクセスが減少し，浮動小数点演算ピーク比は階層ブロッキング適用前の 4.89 %から 7.18 %と性能向上する結果を得られた．

1. はじめに

大規模数値シミュレーションと大容量観測データを融合するための計算技術として「データ同化」が注目されており，特に現代の気象予報においては，データ同化は必要欠くべからざるものとなっている[1-6].

アジョイント法はデータ同化手法の中でも非逐次型に分類される手法で，時系列データ全体を評価し，実測データとシミュレーションモデルとの乖離度を表す評価関数を最小化することにより，初期状態とパラメータを同時に推定する手法である[7-10].

その応用先の一つに、フェーズフィールドモデルを用いた材料内部の構造の状態・パラメータ推定がある[2,11]．フェーズフィールドモデルを用いた状態・パラメータ推定では，連続場における数値計算を行うため， 問題の大規模化にともない計算時間が増大する．そのため， 高性能化なしでは現実的な時間で解くことができない．この計算時間が増大する要因の一つに，シミュレーションの際に必要となるデータサイズがシミュレーションを行う計算機のキャッシュ容量よりもはるかに大きくなることが挙げられる．例えばアジョイント法の Forward model の計算では，フェーズフィールドモデルの拡散項の計算に差分法を用いる．差分法は近傍の格子点値を計算に必要とするステンシル計算を用いて解を求める方法であるため，大規模で自由度が高いモデルでは，この近傍の格子点値のアクセス範囲がキャッシュ容量を超えてしまう．その結果，メモリを読み書きする回数が非常に多くなるので，メモリアクセスが増大し演算効率の劇的低下を生じる．このようにアジョイント法では頻繁にメモリアクセスするメモリインテンシティブな処理のため，メモリ帯域の上限が計算速度の制約となり十分な性能を得られない．したがって，メモリアクセスを減少させるためにキャッシュにあるデータを出来る限り再利用して演算を行うブロッキングによる高性能化が必要になる．

しかしこういった高性能化が必須である一方で、現状では、アジョイント法におけるブロッキングを用いた高性能化手法について十分な検討がなされているとはいえない．そこで本論文では，アジョイント法の Forward model におけるステンシル計算に時空間ブロッキング[12-17]を適用し，さらに複数の Forward model の計算をブロッキングして同時に計算する **2 段の階層ブロッキング手法**を提案する．さらに提案手法にスレッド並列を適用し，2 次元格子点の初期状態を推定する双子問題を設定して性能評価を行い，提案手法の有効性を検証した．

本論文の構成は以下のとおりである． 2 章では，アジョイント法について説明する．3 章では，提案手法である Forward model への階層ブロッキングを提案する．4 章では，名古屋大学情報基盤センターに設置されているスーパーコンピュータである富士通 PRIMEHPC FX100 を用いた性能評価を行う．最後に得られた知見についてまとめを述べる．

2. データ同化

2.1 概要

データ同化は，実測データと計算機シミュレーションをベイズ統計学の枠組みで融合する手法である[1]．元々は気象学や海洋学の分野において用いられていた手法であるが，現在では地震学や材料工学といった様々な分野に応用されている[2-6]．データ同化の目的の一つに数値モデルの最適化があり，実測データと数値シミュレーションとの乖離度を評価関数として，この評価関数を最小化することにより尤もらしいモデルに近づける．

† 名古屋大学 大学院情報科学研究科
†† 東京大学 地震研究所
†3 東京大学 地震研究所/東京大学 大学院情報理工学系研究科
†4 名古屋大学 情報基盤センター 大規模計算支援環境研究部門

ⓒ2016 Information Processing Society of Japan

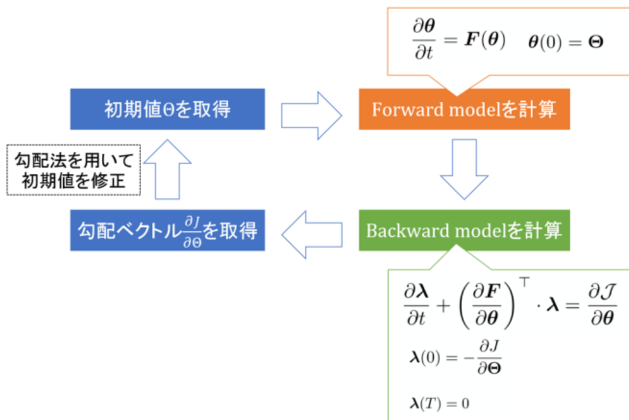

図 1　アジョイント法の計算手順

データ同化には，逐次型データ同化と非逐次型データ同化がある．逐次型データ同化は，時系列データを時間ステップ毎に評価することにより，システムに含まれる状態を修正して適切なものに収束させるもので，代表的な手法として，カルマンフィルタ(KF) やアンサンブルカルマンフィルタ(EnKF) ，粒子フィルタ(PF) といった手法がある.これに対して，非逐次型データ同化では時系列データ全体を評価し，状態ベクトルにモデルパラメータを含める「自己組織化」を行うことにより，初期状態と同時にパラメータの推定を行う．逐次型と非逐次型の大きな違いの一つは，状態・パラメータ推定における計算コストである.例えば，アンサンブル近似に基づく逐次型データ同化は，状態・パラメータ空間全体を探索することによって，原理的には各時間ステップにおける状態の推定値とその不確実性を自然に計算することが可能であるため，計算コストは推定する格子点数とパラメータ数の和である自由度の指数オーダとなる．一方で，本稿で取り扱うアジョイント法に基づく非逐次型データ同化は，算出された評価関数の状態ベクトルに関する勾配に基づき，勾配法によって事後分布が最大になる状態・パラメータ空間の点を探索するため，計算コストは自由度の線形オーダとなる．そのため，所与のシミュレーションモデルの自由度が大きい場合には，もっぱら非逐次型データ同化が用いられる[7-10]．材料工学分野においては，材料内部における組織の成長を表現するためにフェーズフィールドモデル[11]が用いられるが，パラメータ推定だけでなく，その不確実性評価を実施する必要性に迫られている．そのため，大規模シミュレーションモデルに基づくデータ同化においても不確実性評価が可能となるようにするために，2nd-order-adjoint 法を応用した新しいアジョイント法が提案されている[2]．本論文では，まずはアジョイント法について高性能化を検討した．

2.2 アジョイント法

アジョイント法全体の計算手順について説明する（図 1）． まず適当な初期値を設定し，設定した初期値を用い

```
subroutine forward_model_naive()
 do it = 1, nda
  store values in halo region
   do y = 1, ny
    do x = 1, nx
     update A(x,y,it) by using A(x-1,y,it-1),
            A(x+1,y,it-1), A(x,y-1,it-1),
            A(x,y+1,it-1) and A(x,,y,it-1)
    end do
   end do
 end do
end
```

図 2　Naïve な実装における
5 点ステンシル計算カーネル

て Forward model の計算を行う. Forward model では，初期値の情報は時間が進む方向へ伝搬する．次に，Forward model から変分原理によって導出される Backward model に従った計算を行う．Backward model では，実測データとシミュレーションモデルとの差が，時間を遡る方向に逆伝搬する. アジョイント法では，この実測データとシミュレーションモデルとの乖離度を表す評価関数を設定し，この評価関数を最小化する．評価関数を最小化するため，Backward model の計算によって得られた勾配ベクトルを用いて勾配法を適用し，初期値を更新する．アジョイント法において，計算のボトルネックになっているのが Forward model と Backward model の計算である．本稿では、このうち Forward model の計算に着目する．

Forward model の計算において，フェーズフィールドモデルの拡散項は差分法を用いて計算される．差分法は，近傍の格子点値を計算に必要とするステンシル計算を用いて解を求める方法である．Naive な実装における Forward model の 5 点ステンシル計算の計算カーネルを (図 2) に示す. Naive な実装では，このように問題領域の格子点値を逐一更新している．そのため，大規模自由度系において次の時間ステップにおける格子点値を計算するために必要な格子点値がキャッシュ容量を超えてしまい，キャッシュミスを引き起こす可能性がある．このように現状では，アジョイント法の Forward model におけるステンシル計算へのキャッシュブロッキングを適用した高性能化手法について十分な検討がなされているとはいえない．そこで本論文では，この Forward model におけるステンシル計算に対して時空間ブロッキングを適用し，さらに複数 Forward model の計算をブロッキングする 2 段の階層ブロッキング手法を提案する．

ⓒ2016 Information Processing Society of Japan

3. 提案手法

3.1 概要

時空間ブロッキングは，空間方向だけでなく時間方向に対してもキャッシュブロッキングを適用することによって，メモリアクセスを減少させキャッシュにあるデータの再利用性を高める最適化手法である[12-17].

今回の問題では，5 点ステンシル計算によって推定値を計算する．この 5 点ステンシル計算では，次の時間ステップにおける格子点値を計算するために，現在の時間ステップにおける格子点値の近傍 5 つが必要となる（図 3）．図 3 は x 軸と y 軸の空間 2 次元と時間 1 次元の合計 3 次元における 5 点ステンシル計算を表している．

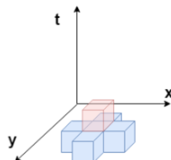

図 3　5 点ステンシル計算

3.2 Forward model の時空間ブロッキング

大規模で自由度が高いモデルでは格子点数が非常に膨大であるため，一般には計算に必要な格子点値のアクセス範囲がキャッシュ容量を超えてしまう．大規模な問題において時空間ブロッキングを適用しない場合，次の時間ステップの格子点値を計算する際に前の時間ステップにおける格子点値がキャッシュに残っておらず，メモリを参照しなければならない．これに対して，時空間ブロッキングを適用し時間方向に対して一気に計算をすることによって，次の時間ステップにおける格子点値を計算する際にキャッシュ上に前の時間ステップにおける格子点値を残す．したがって，キャッシュに前の時間ステップにおける格子点値が残っているのであればメモリを参照する必要がなくなり，その結果メモリアクセスを減少させることができる．

ブロッキング適用後の全格子点値の計算手順は次のとおりである．

STEP 1. 時間ブロッキングサイズ iblt をパラメータとして与え，ブロッキングサイズ iblt まで先の格子点値をピラミッド型に計算する．

STEP 2. 計算していない残りの格子点値をブロッキングサイズ先まで計算する．

ここでは，袖領域の格子点値を更新した後に残りの格子点値を計算する．

STEP 3. 指定されたループ回数だけ時間ステップ後の全ての格子点値を計算し終えたら Forward model の計算を終了し，そうでなければ STEP 1.に戻る．

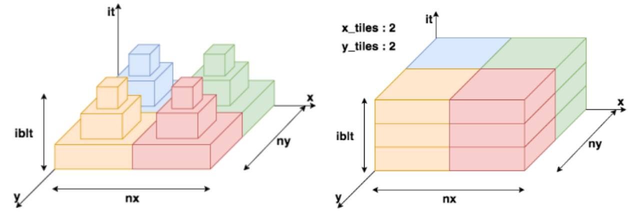

図 4 時空間ブロッキング適用後の
2 x 2 タイルにおける格子点の計算イメージ

2 次元ステンシル計算におけるブロッキング適用後の計算イメージを 図 4 に示す． STEP 1. のピラミッド型の格子点計算について説明する．ここでは，計算は行われないが隣接する格子点の計算のために使われる領域を**袖領域(Halo)**と呼ぶ．次の時間ステップにおいて問題領域の端となる格子点値を計算するためには，この袖領域にある格子点値が必要である．しかし STEP 1.において格子点値を計算する際に，この袖領域にある格子点値を持っていないため，次の時間ステップにおいて端となる格子点値を計算することができない．したがって，時間方向にブロッキングする場合には一度に全ての格子点値を計算することは出来ない．その結果，ある時間ステップにおける全格子点の値を用いて連続的に計算できる格子点は図 4 のようなピラミッド型になる[18].

3.3 複数の Forward model 計算のブロッキング

アジョイント法全体の計算手順 (図 1) を 1 サイクルとする．アジョイント法では，初期値を用いて Forward model, Backward model と計算することにより勾配ベクトルを求める．得られた勾配ベクトルを用いて勾配法を適用し，初期値を更新する．実装では，このサイクルをいくつか設定された収束条件の内の一つを満たすまで繰り返す（図 5）．収束条件は，勾配法における Step 幅を見つけられなかった場合や，十分な回数のループを繰り返した場合などが該当する．

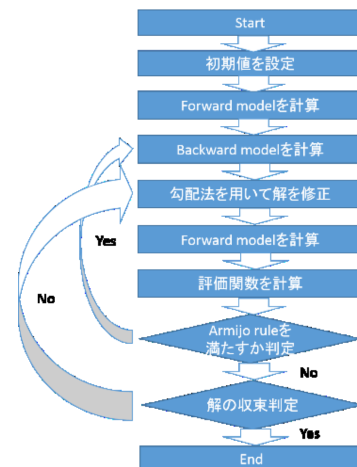

図 5　ブロッキング適用前の計算順序
および収束条件

ⓒ2016 Information Processing Society of Japan

このサイクルの初期値を更新するフェーズにおいて，変数変換後に初期値を更新し逆変換を行うことによって新たな初期値の推定値を得る．次の Forward model の計算フェーズにおいては，得られた全格子点の初期値の推定値を用いて Forward model を計算し，Forward model の計算結果を用いて評価関数を計算する．評価関数の計算後，計算した評価関数の値が Wolfe condition の Armijo rule を満たしているかどうかを逐一判定する．Armijo rule は，勾配法の Step 幅が効果的に評価関数を降下させることの保証である．この Armijo rule の判定結果によって Forward model の計算回数は変化するので, Forward model の計算回数はアジョイント法のサイクル毎に異なる．例えば，初回の収束条件の判定において．評価関数の値が Armijo rule を満たしている場合は, Forward model の計算回数は合計 1 回である．一方で，評価関数の値が Armijo rule を満たさない場合は再度繰り返すため，Forward model の計算回数は合計 2 回以上となる．特に，勾配法の Step 幅が見つからない条件に収束するまでには非常に多い回数の Forward model の計算を行うことが実験的に分かった．そこで本論文では，同時に複数の Forward model を計算するために複数の Forward model にブロッキングを適用する (図 6) . Forward model の計算フェーズにおいて，現在の時間ステップと次の時間ステップにおける Forward model の計算間に依存関係はないため，あらかじめ各時間ステップにおける初期値を更新することで，それぞれの Forward model の計算を同時に行うことができる．このブロッキングによる効果として，Armijo rule を満たすかどうかを逐一判定する回数が減り，同時に Forward model と評価関数を計算することができるため，高性能化が期待できる．

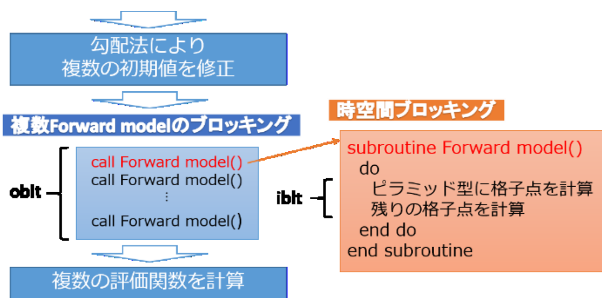

図 6　階層ブロッキング適用後の
イメージ図

3.4 ベンチマーク問題設定

本評価では，2 次元格子点の初期状態を推定する双子問題をベンチマークとして利用した．問題領域は, NX x NY = 1600 x 1600 とし，時間方向のステップ数は nda = 128 とした．また，各格子点の値の initial_guess は全て 0.2 とし，真値は[0,1]の区間における乱数を設定した．

3.5 実験設定

各スレッドが計算する空間方向のブロッキングサイズを NX x (NY/スレッド数) とした．また，時間方向のブロッキングサイズ iblt は 1~50 の値を設定し，複数 Forward model の計算のブロッキングサイズ oblt は 1~3 の値を設定した．各ブロッキングサイズの増加とともにメモリ使用量が増大する. FX 100 の 1 ノードあたりのメモリ容量は 32GB であるため，容量を超えない範囲で設定可能なブロッキングサイズを採用した．ただし, 25 スレッドと 32 スレッドの時には各スレッドの計算領域における y 軸方向の格子点数がそれぞれ 64 と 50 となるため，最大となる時間方向のブロッキングサイズ iblt = 32 と 25 までの結果となっている．

FX 100 は，1 ソケットあたり 16 コア，1 ノードあたり 2 ソケットの NUMA 構成となっている．そのため 1 ノードには 2 つの 16GB メモリがあり，ジョブを発行する際にメモリへの割り当てのポリシを明示的に指定することが可能である．メモリ割り当てポリシには, 2 つのメモリに対して交互に割り当てる interleave_local や，片方のメモリに対して優先的に割り当てる localalloc があるが，今回のベンチマーク問題においてはメモリ割り当てポリシを変更した時の実行時間に殆ど差が無かったので，デフォルトの localalloc を設定した．

これらの設定のもとで，時空間ブロッキング適用による効果と複数 Forward model の計算のブロッキング適用による効果，および階層ブロッキング適用による効果を検証するため, 3 つの実験を行った．

4. 性能評価

4.1 対象計算機の構成

以下の計算機を利用した．

1. Fujitsu PRIMEHPC FX100（FX100）
   - 名古屋大学情報基盤センター設置
   - CPU：SPARC64 XIfx, 2.2 GHz 32(+2)コア
   - 記憶容量：32 GB
   - 理論ピーク性能（ノード）：1.1264 TFLOPS(倍精度)，2.2528 TFLOPS（単精度）
   - キャッシュ構成
     - L1:64KB（命令／データ分離，コア毎），L2:24MB（共有）
     - 4 ウェイ
   - 1 ソケット当たり 16 コア，ノードあたり 2 ソケットの NUMA 構成
   - 富士通 MPI
   - コンパイラ:富士通 Fotran90 コンパイラ version 2.0.0 P-id: T01760-01 (Oct 28 2015 10:14:24)
   - コンパイラオプション: -Kfast –Kopenmp
   - メモリアクセス性能（node あたり）：240 GB/秒 (入力／出力ごと)
   - Stream 性能(Triad)：約 320 GB/秒 [19]

ⓒ2016 Information Processing Society of Japan

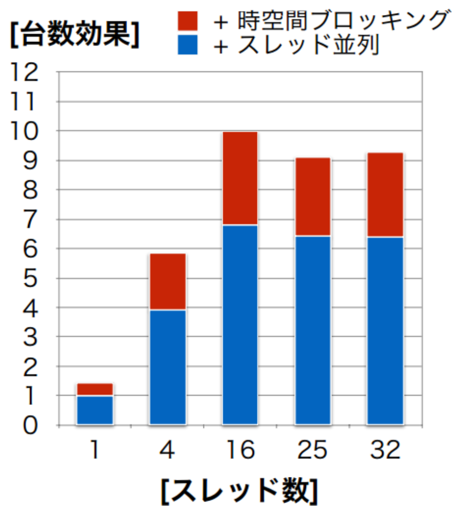

図 7　スレッド並列および時空間ブロッキング
適用による台数効果

4.2　台数効果の比較

図 7 に，Naïve なコードに対してスレッド並列および時空間ブロッキングを適用した時の最速の実行形態における台数効果を載せる．これは 5 点ステンシル計算の計算カーネルだけでなく，配列への値のストア等の操作を含むアプリケーションとしての評価である．図 7 では，16 スレッドの時が最も性能が向上した．スレッド並列により 6.8 倍，さらに時空間ブロッキングを適用することにより 10 倍の性能向上が得られた．一方で 25 スレッドと 32 スレッドの時は，16 スレッドと比べ性能が低下した．

4.3　速度向上率の比較

Forward model の計算において，時間ブロッキングサイズ iblt = 1 の実行時間を 1 とした時に，時空間ブロッキングのみを適用した後の速度向上率を図 8 に示す．図 8 では，横軸が時間ブロッキングサイズ iblt，縦軸が速度向上率を表している．16 スレッドでブロッキングサイズ iblt = 16 の時に最も性能が向上し，速度向上率は 1.47 倍となった．ただし 25 スレッドと 32 スレッドの時は各スレッドの計算領域における y 軸方向の格子点数がそれぞれ 64 と 50 となるため，最大の時間方向のブロッキングサイズ iblt = 32 と 25 までの結果となっている．

図 9 は，複数 Forward model の計算ブロッキング適用前の実行時間を 1 とした時におけるブロッキング適用後の速度向上率で，最速の実行形態である 16 スレッドの時の効果である．横軸は Forward model の計算回数，縦軸は速度向上率を表している．複数 Forward model の計算のブロッキングを適用した場合，実際の Forward model の計算回数が 1 回の時は余分に Forward model を計算するため，ブロッキングサイズ oblt が大きいほど性能は低下する．一方で，実際の Forward model の計算回数が 2 回以上の時は性能が向上し，計算回数が 2 回の時に 1.49 倍，計算回数が 3 回の時に 1.78 倍の速度向上を達成した．

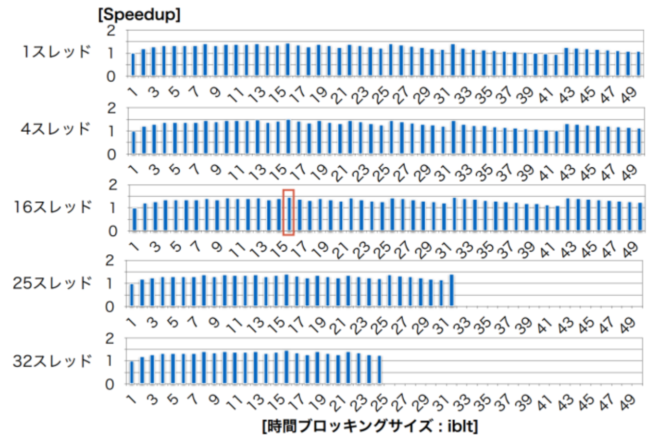

図 8　時空間ブロッキング適用による効果

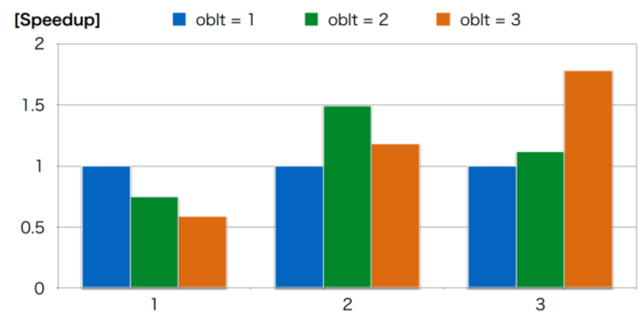

図 9　複数 Forward model の計算のブロッキング
適用による効果（16 スレッドの時）

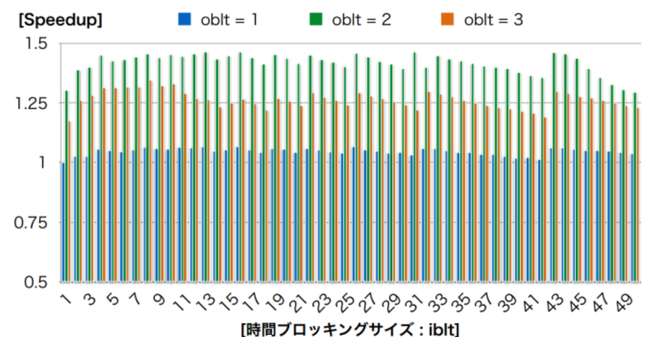

図 10　階層ブロッキング適用による効果
(16 スレッドの時)

図 10 に，Backward model の計算を除くアジョイント法 500 サイクルの実行時間において，2 段の階層ブロッキング適用前の実行時間を 1 とした時の速度向上率を載せる．これは，最速の実行形態である 16 スレッドの時の効果である．横軸は時間ブロッキングサイズ iblt，縦軸は速度向上率を表している．利用したベンチマーク問題においてブロッキングサイズ iblt = 16, oblt = 2 の時に最大となる 1.46 倍の速度向上を達成した．

4.4　プロファイル結果

時空間ブロッキング適用前と適用後での性能差の原因を調査するため，富士通社の詳細プロファイラ（精密 PA 可視化）を利用した．表 1 に，時空間ブロッキング適用前と適

ⓒ2016 Information Processing Society of Japan

用後の最速の実行形態における性能プロファイル結果を載せる.

表 1 (a) (b) より, 時空間ブロッキング適用前のキャッシュミス数と比べ, 時間ブロッキングサイズ iblt = 16 の時の L1 キャッシュミス数は 30%, L2 キャッシュミス数は 32%削減され, それぞれのキャッシュミス率は減少した. また, 浮動小数点演算ピーク比においても時空間ブロッキング前の 4.89%と比べ, 適用後には 7.18%と 2.29 ポイント増加した.

4.5 考察

4.5.1 台数効果

図 7 の台数効果では, 16 スレッドの時のスレッド並列により 6.8 倍, さらに時空間ブロッキング適用により 10 倍の性能向上を得られた. 一方で, 25 スレッドと 32 スレッドでは 16 スレッドの時と比べ性能が低下した. 原因の一つとして, 今回のベンチマークの問題設定では, 問題領域全体としての問題サイズが 16GB よりも小さく, 別ソケットのコアによるメモリアクセスにより性能が低下したことが考えられる.

4.5.2 各ブロッキング適用による速度向上

図 8 より, 時空間ブロッキング適用前の実行時間を 1 とした時のブロッキング適用後の速度向上率は, 時間ブロッキングサイズ iblt = 16 の時に最大となる 1.47 倍を達成した. 特に 2 の冪の時間ブロッキングサイズにおいては, 他のブロッキングサイズと比べ性能が向上した. この理由としては, 今回の実験では Forward model の計算 1 回あたりの時間ステップ数 nda = 128 と設定しており, 無駄な時間ステップ分の計算をすることなく Forward model の計算が可能であることが挙げられる.

図 9 より, 複数 Forward model の計算のブロッキング適用前の実行時間を 1 とした時のブロッキング適用後の向上率は, 実際の Forward model の計算回数が 1 回の時には, ブロッキング適用により性能が低下した一方で, Forward model の計算回数が 2 回の時には 1.49 倍, 計算回数が 3 回の時には 1.78 倍の速度向上を達成した. 実際のアジョイント法では, 1 サイクル毎に Forward model の計算回数が異なるため, この結果は理想的な場合における速度向上率となっている. 最大の速度向上を得られるのは複数 Forward model の計算のブロッキングサイズと実際の Forward model の計算回数が一致する時であるが, 計算回数が 2 回の時のブロッキングサイズ oblt = 3 のような Forward model の計算を余分に実行している場合においても, 複数 Forward model の計算のブロッキング適用前と比べ速度が向上する結果となった.

図 10 は, Backward model の計算を除くアジョイント法 500 サイクルにおいて, 各ブロッキング適用前の実行時間を 1 とした時の 2 段の階層ブロッキング適用後の結果である. 時間ブロッキングサイズ iblt = 16 で, 複数 Forward model の計算のブロッキングサイズ oblt = 2 の時に最大

	L1Dミス率 (/ロード・ストア数)	L1Dミス数	L2ミス率 (/ロード・ストア数)	L2ミス数	実行時間 (sec)	浮動小数点演算ピーク比	MFLOPS
Thread 0	2.20%	4.08E+06	2.11%	3.90E+06	0.15	4.90%	1725
Thread 1	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.91%	1728
Thread 2	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.91%	1729
Thread 3	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1726
Thread 4	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1725
Thread 5	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1724
Thread 6	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.89%	1722
Thread 7	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.89%	1722
Thread 8	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1725
Thread 9	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1725
Thread 10	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1724
Thread 11	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1724
Thread 12	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1724
Thread 13	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.91%	1727
Thread 14	2.20%	4.08E+06	2.10%	3.89E+06	0.15	4.90%	1725
Thread 15	2.20%	4.08E+06	2.11%	3.89E+06	0.15	4.90%	1726
Process	2.20%	6.52E+07	2.10%	6.23E+07	0.15	4.89%	27548

(a) 時空間ブロッキング適用前

	L1Dミス率 (/ロード・ストア数)	L1Dミス数	L2ミス率 (/ロード・ストア数)	L2ミス数	実行時間 (sec)	浮動小数点演算ピーク比	MFLOPS
Thread 0	1.93%	2.89E+06	1.77%	2.65E+06	0.11	7.19%	2530
Thread 1	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2531
Thread 2	1.93%	2.89E+06	1.77%	2.66E+06	0.11	7.19%	2532
Thread 3	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2533
Thread 4	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.20%	2534
Thread 5	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2530
Thread 6	1.93%	2.89E+06	1.78%	2.66E+06	0.10	7.21%	2536
Thread 7	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.20%	2536
Thread 8	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.20%	2535
Thread 9	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.20%	2533
Thread 10	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2531
Thread 11	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2539
Thread 12	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.19%	2532
Thread 13	1.93%	2.89E+06	1.78%	2.66E+06	0.11	7.20%	2534
Thread 14	1.93%	2.89E+06	1.77%	2.66E+06	0.11	7.18%	2538
Thread 15	1.93%	2.89E+06	1.78%	2.67E+06	0.11	7.19%	2532
Process	1.93%	4.63E+07	1.78%	4.26E+07	0.11	7.18%	40450

(b) 時空間ブロッキング適用後

表 1　16 スレッドの時の性能プロファイル結果

1.46 倍の速度向上を達成した. 調査した結果, 今回利用したベンチマーク問題では Forward model の計算回数が 1 サイクルあたり平均 2 回であるため, 複数 Forward model の計算のブロッキングサイズ oblt = 2 の時が 1 と 3 の時に比べ性能が向上したと考えられる.

4.5.3 性能プロファイル結果からの考察

表 1 (a) (b) より, 16 スレッドの時の時間ブロッキングサイズ iblt = 16 におけるキャッシュミス数は, 時空間ブロッキング適用前と比べ, L1 キャッシュミス数が 30%, L2 キャッシュミス数が 32%と僅かに削減され, キャッシュミス率が減少した. キャッシュミス数の減少した割合が低い原因を調査する必要があるが, 利用しているベンチマーク問題の問題サイズが小さいため, 元々の Naïve な実装における Forward model の計算自体のキャッシュヒット率が高いことが原因として考えられる.

このキャッシュミス数の削減によってデータの再利用性が高まり, 浮動小数点演算ピーク比は時空間ブロッキング適用前の 4.89%と比べ, ブロッキング適用後には 7.18% と 2.29 ポイント向上した. これは, より多くのキャッシュ上にあるデータを利用した計算が可能になったことでメモ

ⓒ2016 Information Processing Society of Japan

リアクセスが減少し，結果として実行時間が短くなったためと推測される．

5. おわりに

本研究では，アジョイント法の Forward model の計算に時空間ブロッキングを適用し，さらに複数 Forward model の計算にブロックングを適用する 2 段の階層ブロッキングを提案し，各ブロッキングとブロッキングを組み合わせた後の効果を検証した. FX 100 を用いてブロッキングの効果を調べたところ，ブロッキング適用前の Forward model の計算の実行時間を 1 とした時に，時空間ブロッキング適用によって 1.47 倍の速度向上を達成し，複数 Forward model の計算のブロッキング適用によって Forward model の計算回数が 3 回の時に上限値となる 1.78 倍の速度向上を達成した．さらに 2 段の階層ブロッキングの適用により，階層ブロッキング適用前の Backward model の計算を除くアジョイント法の実行時間を 1 とした時に，階層ブロッキング適用によって 1.46 倍の速度向上を達成した．また，プロファイラにおけるキャッシュミス数とキャッシュミス率を解析したところ，時空間ブロッキング適用前と比べ，適用後では L1 キャッシュミス数が 30%, L2 キャッシュミス数が 32%削減され，キャッシュミス率が減少した．これによりメモリアクセスが減少し，浮動小数点演算ピーク比は時空間ブロッキング適用前の 4.89%と比べ 7.18%となり 2.29 ポイント向上した．

今回はベンチマークの問題サイズを固定して 2 段の階層ブロッキングの効果を検証したが，提案手法の有効性を調査するため，異なる問題サイズにおいても同様に検証する必要がある．また，アジョイント法の Backward model の計算も同様にボトルネックとなっているため，この Backward model の計算への時空間ブロッキングの適用と，これらの時空間ブロッキングサイズおよび複数 Forward model の計算のブロッキングサイズに対する自動チューニング[12]の適用は，今後の課題である．

謝辞

本研究の一部は，科学技術研究費補助金、基盤研究（B)、「通信回避・削減アルゴリズムのための自動チューニング技術の新展開」(課題番号:16H02823), および,・ポスト「京」萌芽的課題アプリケーション開発、萌芽的課題 1 基礎科学のフロンティアー極限への挑戦､「極限の探究に資する精度保証付き数値計算学の展開と超高性能計算環境の創成」による．

参考文献

1) 樋口知之 編著, , データ同化入門: 次世代のシミュレーション技術. 朝倉書店 (2011).
2) S. Ito, H. Nagao, A. Yamanaka, Y. Tsukada, T. Koyama, M. Kano, and J. Inoue, Data assimilation for massive autonomous systems based on a second-order adjoint method, Physical Review E 94, 043307 (2016).
3) 長尾大道，樋口知之，地震音波データ同化システムの開発，統計数理，第 61 巻，第 2 号, 257-270 (2013).
4) E. Kalnay, Atmospheric Modeling, Data Assimilation and Predictability (Cambridge University Press, Cambridge, 2003).
5) T. Tsuyuki and T. Miyoshi, Recent Progress of Data Assimilation Methods in Meteorology, J. Meteorol. Soc. Jpn. Ser. II 85B, pp. 331-361 (2007).
6) M. Ghil and P. Malanotte-Rizzoli, Advances in Geophysics (Elsevier, New York, 1991), Vol. 33, pp. 141–266.
7) 淡路敏之，蒲地政文，池田元美，石川洋一 編著，データ同化: 観測・実験とモデルを融合するイノベーション，京都大学学術出版会, (2009)
8) J. M. Lewis and J. C. Derber, The use of adjoint equations to solve a variational adjustment problem with advective constraints, Tellus A 37A, 309-322 (1985).
9) F.-X. Le Dimet and O. Talagrand, Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects, Tellus A 38A, 97-110 (1986).
10) M. Iri and K. Kubota, The Japan Society for Industrial and Applied Mathematics 1, 17 (1991) (in Japanese).
11) 小山敏幸，高木知弘，フェーズフィールド法入門，丸善出版, (2013)
12) Datta, Kaushik, et al. "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures." Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008.
13) 須田礼仁，一般化菱形行列冪カーネルのための領域分割アルゴリズム，研究報告ハイパフォーマンスコンピューティング (HPC), Vol.2016-HPC-155, No.43, pp.1-9 (2016).
14) 金光浩，遠藤敏夫，松岡聡，GPU メモリ容量を超える問題規模に対応する高性能ステンシル計算法，研究報告ハイパフォーマンスコンピューティング (HPC), Vol.2012-HPC-137, No.31, pp. 1-6, (2012).
15) 南武志，高橋康人，岩下武史，中島浩，キャッシュメモリを考慮した 3 次元 FDTD カーネルの性能改善，情報処理学会論文誌コンピューティングシステム (ACS), Vol.4, No. 2, pp. 70-83, (2011).
16) Maruyama, Naoya, and Takayuki Aoki. "Optimizing stencil computations for NVIDIA Kepler GPUs." Proceedings of the 1st International Workshop on High-Performance Stencil Computations, Vienna (2014).
17) Meng, Jiayuan, and Kevin Skadron. "Performance

modeling and automatic ghost zone optimization for iterative stencil loops on GPUs." Proceedings of the 23rd international conference on Supercomputing. ACM (2009).

18) 深谷猛, 岩下武史, 反復型ステンシル計算のマルチコア・メニーコア向け実装に関する考察, 日本応用数理学会「行列・固有値問題の解法とその応用」研究部会, 第21回研究会, 2016年並列/分散/協調処理に関する『松本』サマー・ワークショップ (SWoPP2016), (2016) (口頭発表)
19) 千葉修一, "PRIMEHPC FX100 の特徴と概要" (2015) http://accc.riken.jp/wp-content/uploads/2015/06/chiba.pdf