Vol.2014-HPC-145 No.7
2014/7/28

実アプリケーションにおけるウェーブレット変換に基づくチェックポイントデータの非可逆圧縮手法

佐々木 尚人[1]　佐藤 賢斗[1]　遠藤 敏夫[1]　松岡 聡[1]

概要：近年，HPC システムやスーパーコンピュータの規模は急速に拡大しつつあり，それに伴いシステムの平均故障間隔が短縮してしまう傾向にある．また，多くのシステムでは耐故障機能としてチェックポインティングが採用されているが，将来的にチェックポイント時間が平均故障間隔を上回ってしまう可能性があることが問題視されている．そこで，我々はチェックポイント時間を短縮するため，チェックポイントデータの非可逆圧縮手法を提案する．具体的には，チェックポイントデータに対してウェーブレット変換，量子化，符号化に加えてスタンダードな圧縮手法を適用することで非可逆圧縮を行う．本研究ではこの提案手法を気象アプリケーション NICAM のチェックポイント対象データに適用し，発生する誤差，圧縮率，圧縮時間について測定，評価を行った．その結果，特定の条件下で，相対誤差の最大が 5%以内で，チェックポイント時間を約 70%短縮できることを確認した．

1. はじめに

近年、HPC システムやスーパーコンピュータの規模は急速に拡大しつつあり、それに伴い HPC システムの演算性能は飛躍的に進歩している．その性能を利用することで様々なアプリケーションでは計算時間の短縮、計算の精度の向上を達成することができている．

しかし、その一方で大規模化によって生じる問題もあり、それは計算ノード数の増加、またノード内のコンポーネントの増加による故障率の増大である [1], [2], [3]．特に、2020 年頃までに運用が開始が目指されているエクサスケールスーパーコンピュータでは MTBF(平均故障間隔) が 30 数分との予測がされており、これは対処しなければならない問題である．

そのため、HPC システムでは耐故障機能を備える必要があり、その多くが耐故障機能としてチェックポインティングを採用している [4], [5]．しかし、MTBF の短縮によりチェックポイント時間が MTBF を上回ってしまい、アプリケーションの計算が進行しなくなってしまう可能性がある．

そこで、我々はチェックポイント時間を短縮するため、ウェーブレット変換 [6] に基づくチェックポイントデータの非可逆圧縮を提案する．具体的には、チェックポイント対象となるデータに対して、ウェーブレット変換、量子化、符号化に加えて gzip を適用することでチェックポイントデータの圧縮を行う．本研究では、提案手法を気象アプリケーション NICAM[7] のチェックポイント対象データに適用し、発生する誤差、圧縮率、圧縮時間に関して測定を行い評価した．その結果、特定の条件下で、相対誤差の最大が 5%以内で、チェックポイント時間を約 70%削減できることを確認した．

[1] 東京工業大学

2. 背景

2.1 チェックポイントにおける圧縮

チェックポイント時間削減のため，我々はチェックポイントに対して圧縮を適用する．その際，削減したチェックポイントサイズの大きさがチェックポイント時間に影響を与えることになるのでチェックポイントの圧縮率が高いことが望まれる．

しかし，それだけを考慮すれば良いということではでなく，チェックポイントデータの圧縮では圧縮時間についても考慮しなければならない．例えば，圧縮を行わなかった場合のチェックポイントの I/O 時間を N，圧縮を行った場合のチェックポイントの I/O 時間を T，圧縮時間を C としたとき，

$$N > T + C \tag{1}$$

となっていなければならない．

2.2 非可逆圧縮へのモチベーション

圧縮には可逆圧縮と非可逆圧縮が存在している [8]．非

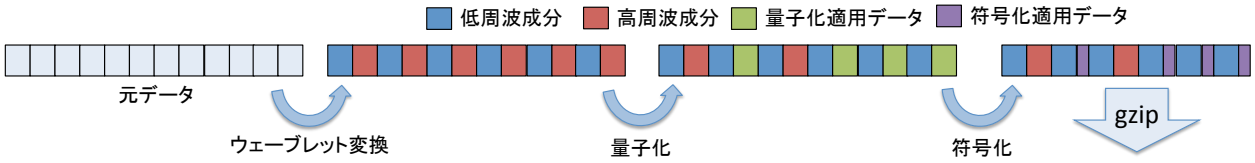

図 1 提案手法の流れ

可逆圧縮では可逆圧縮と異なり，復元の際にデータが欠損してしまうが，比較的高い圧縮率を実現することができることが特徴である．

アプリケーションによっては必ずしもデータの全ビットを保存する必要性がないものがある．例えば，気象予測のシミュレーションで格子の粒度を変更しても，その計算結果はほぼ等しいものが得られたという事例が存在し，これはデータのを完全に保存する必要性がないことを示している．

そのようなアプリケーションを対象とした場合，チェックポイントデータに非可逆圧縮を適用することで，シミュレーションの計算結果の精度が落ちる一方で，高い圧縮率と短いチェックポイント時間を実現することができる．故に，本研究ではチェックポイントデータの非可逆圧縮を研究対象としている．

2.3 ウェーブレット変換へのモチベーション

ウェーブレット変換は周波数解析の技術であるが，その特徴である多重解像度解析を用いることでデータ圧縮に有効であるとされている．実際に，JPEG2000 の圧縮アルゴリズムにもウェーブレット変換が採用されており，それまで JPEG が用いていた離散コサイン変換 [9] と比べて優れた圧縮効果を実現している．

また，ウェーブレット変換を用いた圧縮手法では対象となるデータがなめらかであること，つまり圧縮対象となるデータの近傍の値の差が小さいことを利用している．そのため，本手法に向いているのはそのようななめらかな物理量を配列として持つようなシミュレーション等のアプリケーションである．以上のような理由から，本研究ではウェーブレット変換を採用する．

3. ウェーブレット変換に基づく非可逆圧縮手法の提案

チェックポイント時間を短縮するため，我々はウェーブレット変換，量子化，符号化を適用し，後述する出力フォーマットで保存したデータに gzip を用いてチェックポイントデータの非可逆圧縮を行う（図 1）．これらの手法のアルゴリズムは二重ループ等をなくし，チェックポイントサイズに対して $O(n)$ となるように実装した．

この手法を適用することにより誤差が発生するため，ポインタのような誤差を許さないようなデータにこの手法を適用することはできない．その一方で，アプリケーションの特性によっては誤差を許容するデータも存在し，本手法ではそのようなデータ，特に本研究では float 型，double 型の 1,2,3 次元配列を今回の圧縮対象とする．

以上のことから，本手法の適用対象となるデータの範囲はユーザーが指定するものと仮定し，以下ではその具体的なアルゴリズムについて説明していく．

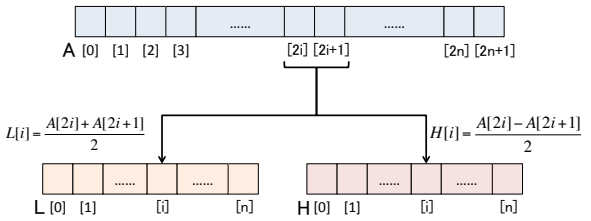

図 2 1 次元のウェーブレット変換

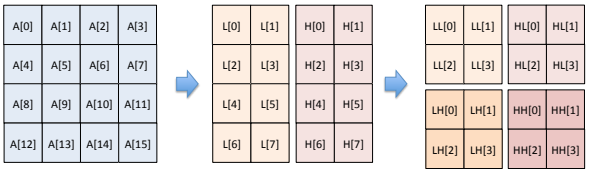

図 3 2 次元のウェーブレット変換

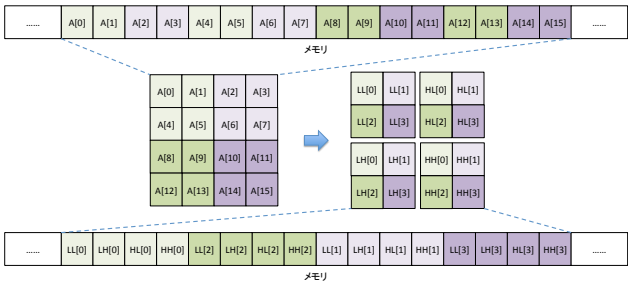

図 4 ウェーブレット変換時のメモリ配置の工夫

3.1 ウェーブレット変換

本手法では，まず圧縮対象となるデータに対してウェーブレット変換を適用する．ウェーブレット変換では対象となる配列の値を 2 つに分割する．まずは 1 次元のウェーブレット変換について具体的な変換方法を説明する（図 2）．ウェーブレット変換には様々な変換アルゴリズムが存在しているが，本手法のウェーブレット変換では，隣接する値の平均値と差の半分を利用していて，各隣接した値をそれぞれ，その平均値と差の半分に変換する．ここで求めた平

均値のことを低周波成分，差の半分を高周波成分と呼ぶこととする．

2,3 次元のウェーブレット変換は，1 次元のウェーブレット変換を各次元に対して適用することで実現している（図 3）．このとき，変換に使用したメモリ配置と同様な配置で変換後の値を保存するのではなく，変換後の値を連続するメモリに順に保存するような形を取る（図 4）．

2 次元のウェーブレット変換では図 3 の LH, HL, HH の部分が高周波成分にあたり，隣接した 4 つのデータから 1 つの低周波成分と 3 つの高周波成分が計算される．同様に，3 次元のウェーブレット変換では 8 つの隣接するデータから 1 つの低周波成分と 7 つの高周波成分が計算されることとなる．

3.2 量子化

本手法の適用対象はなめらかなデータであることを想定しているので，近傍との差が 0 に近い．そのため、ウェーブレット変換適用後のデータの高周波成分は 0 に近い値が多く含まれる．そこで量子化では，その利点を活用して，図 5 のようにウェーブレット変換を行ったデータの一部または全ての高周波成分に対して変換を行う．

3.2.1 単純な量子化手法

まず高周波成分の最大値，最小値を求め，図 5 の上段のように，全ての分割の値の幅が均等になるように，高周波成分を n 個に分割する．ここではこの n のことを分割数と呼ぶこととする（図 5 の例では $n = 4$）．次に，各分割内の平均値を求め，その分割内の全ての値を求めた平均値で置き換える．これにより，高周波成分の全てのデータは n 種類の平均値に置き換えられたことになる．このように，量子化では，高周波成分の値を各分割毎の平均値で置換することで，誤差が発生する．

3.2.2 誤差軽減のための提案量子化手法

本手法適用対象となる物理量は比較的滑らかであることが期待され，ウェーブレット変換による高周波成分は 0 に近い値を多く含むこととなり，高周波成分のヒストグラムは図 5 に示されているように 0 の近傍に「山」のような形を作る傾向にある．高周波成分の大部分が「山」を成しているので，「山」の外は圧縮しなくとも，単純な量子化と比較して圧縮率への影響が少ないことが期待される．その上，「山」の外は比較的大きな値であるので単純な量子化の際に誤差が大きく出やすいと考えられる．そこで，新しい量子化手法ではその「山」のみを対象に量子化することを考える．

まず「山」を発見するため，高周波成分の最大値，最小値を求め，全ての分割の値の幅が均等になるように，高周波成分を適当な個数 (以下，この数を d とする) に分割する．図 5 の例では $d = 10$ となる。また高周波成分の要素数を N_{high}，各分割に含まれる値の個数を N_{div} としたとき，

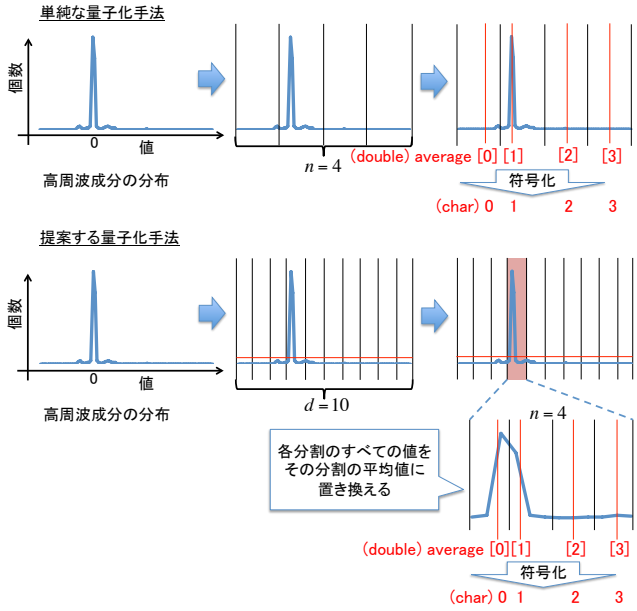

図 5 量子化から符号化までの流れ

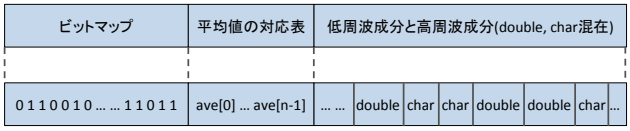

図 6 出力フォーマット

$$N_{div} \geq \frac{N_{high}}{d} \tag{2}$$

を満たす分割のみを対象に最大値，最小値を求める．その後，その最小値から最大値までの区間に対して，従来の量子化手法と同様に n 分割し，量子化を行う．

3.3 符号化

量子化適用後のデータは高周波成分の量子化適用部分が n 種類の double(float) 型の値となっている．符号化では，この n 種類のデータを対応する char 型に変換する（図 5）．具体的には，i 番目の分割の平均値を char 型の i に変換する．

これにより float 型ならデータ量を 1/4 に，double 型なら 1/8 にすることができ，今回は簡単のため char 型を用いて符号化を行ったが，最低限の bit 数 k で符号化を実現すれば，データ量は float 型なら $k/32$，double 型なら $k/64$ のデータ量に圧縮することができる．

3.4 出力フォーマット

圧縮後のデータを復元に必要なデータを考えることで出力フォーマットについて考える．まず，もちろんであるが double(float) 型の低周波成分と，double(float) 型と char 型の混在する高周波成分のデータが必要となる．また，後にデータを復元可能とするために，高周波成分のどの部分を char 型に変換したかをビットマップとして記憶する．その

表 1 使用マシンのスペック

CPU	Intel Core i7-3930K 6 cores 3.20GHz
メモリサイズ	16GB
ファイルシステム	Network File System (NFS) v3 1.5TB
ネットワークカード	Broadcom bnx2
RAID	Dell PERC H700 (RAID6)
ディスク	Western Digital WD (model:WD2002FAEX)

他にも，どのような double(float) 型のデータをどのような char 型に変換したかを対応表として記憶する．

故に，我々は圧縮後の出力フォーマットとして図 6 のようなフォーマットを使用する．ここで，それぞれが要するデータ量を考えると以下のようになる．

- ビットマップ
  高周波成分の要素数分だけの bit 数
- 対応表
  double 型なら（分割数*64）bit，float 型なら（分割数*32）bit
- 低周波，高周波成分
  どの程度 char 型に変化するかにより bit 長が変化

4. 評価

本研究では，気象アプリケーション NICAM[7] のチェックポイント対象データに対して本手法を適用し，その際，発生する誤差，圧縮率，圧縮時間について測定，評価を行った．

4.1 評価環境

本研究では，表 1 のようなスペックを持つクラスタを使用して実験を行った．実験には気象アプリケーション NICAM を用い，その中でも圧力，温度，速度（x,y,z 方向）を表す 3 次元配列に対して本手法を適用した．この 3 次元配列はいずれも配列サイズが $1156 \times 82 \times 2$ で倍精度の変数である．本論文に掲載しているグラフは温度配列について測定したものである．チェックポイントデータは NFS による共有ディスクに格納される．

今回は量子化を行う際の分割数を 2^0~2^7（1~128）まで変化させ，その際発生する誤差，圧縮率，圧縮時間について測定，評価を行い，2 つの量子化手法についても発生する誤差，圧縮率の観点で比較を行う．また，チェックポイントに非可逆圧縮を適用するため，誤差を持ったチェックポイントデータを使用してアプリケーションの計算が進行することになるので，計算の進行に伴う誤差の影響についても評価を行う．

ここでの圧縮率（cr）とは，未圧縮時のチェックポイントサイズを cs_{orig}，本手法適用時のチェックポイントサイズを cs_{comp} としたとき，

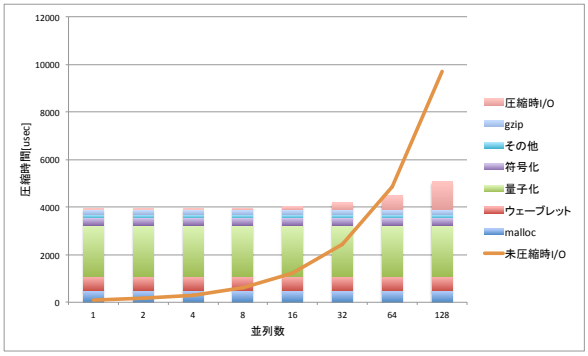

図 7 並列数の変化と圧縮時間

$$cr = \frac{cs_{comp}}{cs_{orig}} \times 100 \tag{3}$$

という式で表され，圧縮率の測定にはファイルサイズの多くを占める低周波成分と高周波成分の値のみを圧縮したものを使用している．相対誤差（re_i）とは，元のチェックポイントデータを $X = \{x_i\}$，非可逆圧縮，復元を行ったチェックポイントデータを $\tilde{X} = \{\tilde{x}_i\}$ としたとき

$$re_i = \frac{|x_i - \tilde{x}_i|}{\max_j\{x_j\} - \min_j\{x_j\}} \tag{4}$$

という式で表されるものとする．また，図 5 で表される d は 64 に固定して評価を行っている．

4.2 分割数，量子化手法によるトレードオフ

量子化の際の分割数を変化させ，評価を行った．図 8，図 10 からわかるように，分割数の増加に伴い，圧縮率は増加してしまう傾向にあるが，その分誤差を軽減できること，単純な量子化手法と比べると，提案手法の圧縮率が増加し，その分誤差を軽減できることが確認できた．これにより，圧縮率と誤差はトレードオフの関係にあることがわかり，許容される誤差等に応じて分割数のなどを決定する必要がある．

圧縮時間は分割数による変化はほとんどなく，量子化手法により多少の変化がある．また，問題サイズに対して比例して圧縮時間が増加してくことが期待される．以下では圧縮時間，圧縮率，相対誤差の詳細な評価を述べていく．

4.3 圧縮時間に関する評価

本研究では，提案手法の圧縮時間を未圧縮時と比較することで評価を行う．ここで，提案手法を適用した際の圧縮時間と圧縮されたチェックポイントの I/O 時間の合算が，圧縮されていないチェックポイントの I/O 時間よりも短くなっていれば，提案手法の意義がある．

図 7 は並列数の増加に対する圧縮時間と I/O 時間を示している．この図ではどの並列数でも各プロセスが約 1.5MB のチェックポイントデータを持っている弱スケーリングであること，I/O のスループットが 20GB/s であることの 2 つを仮定して I/O 時間を算出している．その他の処理時間

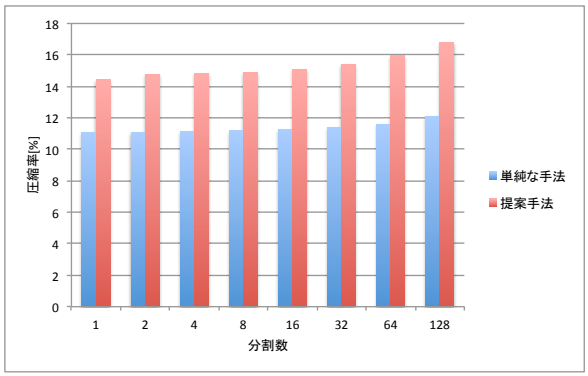

図 8 量子化手法，分割数の変化と圧縮率

は 1 プロセス時の実測に基づく．I/O のスループットが低いほど，ファイル I/O の占める時間が多くなるので本手法はより有効な手法となる．この仮定より，図 7 が示すように各プロセスが行う圧縮時間は並列数によらず変化せず，チェックポイントサイズは約（1.5× 並列数）MB となるのでチェックポイントの並列ファイルシステム等への I/O 時間は並列数に比例して増加していくことになる．

低い並列数では I/O 時間に比べて圧縮時間が大きくなり，本手法による時簡短縮は達成できなくなるが，高い並列数では圧縮時間が無視できるようになり，圧縮率に関しては後述するが，本手法による圧縮率が最大でも約 29%であるので，並列数が増加すると時間削減率が最低でも約 70%に近づいていき．この仮定の元では並列数が増加するほどチェックポイント時間の削減効率が向上する．

また第 3 章で説明したように，このアルゴリズムは I/O 時間と同じくチェックポイントサイズに対して $O(n)$ の時間であるので，1 プロセスが扱うチェックポイントサイズが増加しても．図 7 で見られた時間に関するの優位性は保たれる．

4.4 圧縮率に関する評価

ここでは量子化手法の違い，分割数の増減による圧縮率の変化について評価する．図 8 は量子化手法，分割数を変化させたときの温度配列の圧縮率を示している．圧縮率は分割数の増加と共に増加していく傾向にあり，単純な手法では分割数が 1 のとき 11.06%，128 のとき 12.10%，提案手法では分割数が 1 のとき 14.43%，128 のとき 16.75%となる．また単純な手法に比べ，提案手法では圧縮率が約 30%大きくなってしまうが，その分誤差が軽減することが期待される．誤差の評価については以下で説明を行う．なお，他の配列についても評価したところ，単純な手法での圧縮率は 11~13%であるが，提案手法では 13~29%と配列毎で圧縮率のばらつきが見られた．

また，図 9 は提案手法を使用しない，または一部のみを使用した際の圧縮率を測定したものである．その各種測定方法を以下に示す．

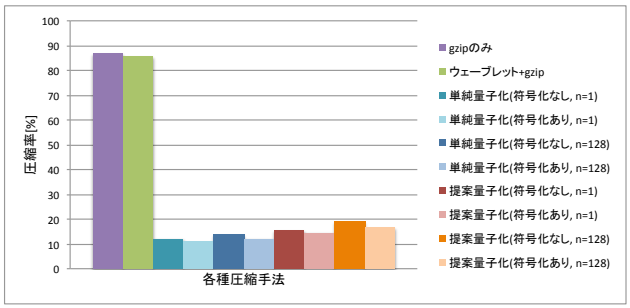

図 9 提案手法未適用時または一部適用時との比較

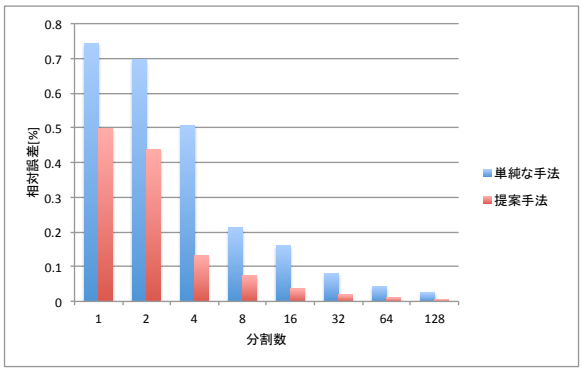

図 10 量子化手法，分割数の変化と相対誤差

- 元データに gzip のみを適用
- ウェーブレット変換適用データに gzip を適用
- 単純な量子化（分割数 $n = 1$）で符号化を行わず gzip を適用
- 単純な量子化（分割数 $n = 1$）を用いた提案手法
- 単純な量子化（分割数 $n = 128$）で符号化を行わず gzip を適用
- 単純な量子化（分割数 $n = 128$）を用いた提案手法
- 提案量子化（分割数 $n = 1$）で符号化を行わず gzip を適用
- 提案量子化（分割数 $n = 1$）を用いた提案手法
- 提案量子化（分割数 $n = 128$）で符号化を行わず gzip を適用
- 提案量子化（分割数 $n = 128$）を用いた提案手法

gzip のみを適用した圧縮では圧縮率が 86.78%，ウェーブレット変換と gzip では 85.56%であるのに対して，量子化を行うことで最大でも圧縮率が 18.92% になっていることから量子化が圧縮効率に大きな効果を与えていることがわかる．さらに，符号化を行うことで符号化を行わなかったときと比べて約 10%ほどチェックポイントサイズを削減することができていることが確認できる．

4.5 誤差に関する評価

ここでは量子化手法の違い，分割数の増減による平均の相対誤差の変化について評価する．図 10 では量子化手法，分割数を変化させたときの相対誤差を示している．相対誤差は分割数が増加することにより軽減していく傾向があり，単純な手法では分割数が 1 のとき 0.74%，128 のとき 0.025%，提案手法では分割数が 1 のとき 0.49%，128 のとき 0.0056%となる．単純な手法に比べると，提案手法では最大で約 67%誤差が軽減していることを確認できた．

単純な手法での相対誤差の最大値が 0.048~56.84%，

誤差の平均値が 0.0053~14.56%，提案手法では最大値が 0.0022~5.94%，平均値が 0.0004~1.19%であり，圧縮率と同様に配列毎でのばらつきが見られ，量子化手法の変化による誤差の軽減が確認された．

また本手法はチェックポイントデータに対して適用することを想定しているため，誤差を持ったまま計算が進行することになる．

図 11 は誤差を持ったチェックポイントデータを使用し，1500 ステップ（約 20 日分のシミュレーション）の計算を行ったときの相対誤差の平均値の推移を示している．単純な手法では，計算の始めは誤差が振動しながら上昇しているが，830 ステップあたりから少しずつ誤差が減少している．提案手法では，誤差の振動が少なく 1500 ステップ通して誤差が上昇傾向にある．どの配列でも，大まかにこのような傾向が見られるが，配列や計算手法の違いにより誤差がどのように変化するかに違いが現れると予想される．

また，誤差の大きさだけでなく誤差の分布や，アプリケーション，変数の特性によりシミュレーションデータとしてシミュレーションの結果に影響を与えるかどうかは異なる．よって今後は発生する誤差についてシミュレーションデータとして許容される誤差であるということを示すような評価をしていく必要がある [10].

5. 関連研究

関連研究としてチェックポイントに関する研究 [11], [12] と，チェックポイントサイズ削減 [13], [14], [15], [16] に関する研究，非可逆圧縮手法を用いた研究について述べる．

チェックポイントに関する研究ではチェックポイント間隔の最適化が研究されている [11]．この研究では障害を含めたアプリケーションの進行を 3 状態のマルコフ連鎖 [17], [18] でモデル化し，チェックポインティングにかかるオーバーヘッドやレイテンシ，システムの故障率等のパラメータから定間隔にチェックポイントを取る際の最適なチェックポイント間隔を算出する式を提案している．本手法を用いて圧縮したチェックポイントを定間隔に保存するときには，本手法の圧縮にかかる時間を考慮しつつ，この研究の成果を利用することができる．

その他に，チェックポイント時間の最適化の研究もされている [12]．この研究ではマルチコンピュータ上でのチェックポイント時間の最適化について研究しており，3 種類の一貫性を持つチェックポイントアルゴリズム [19] と 2 種類のチェックポイント時間の最適化手法を提案している．この研究は，チェックポイントの取り方に関する工夫を行っているので，我々の手法で圧縮を行うことでさらにチェックポイント時間の短縮が期待される．

チェックポイントサイズを削減する研究の 1 つとしてインクリメンタルチェックポイントという技術がある [13], [14], [15]．インクリメンタルチェックポイントとは，チェックポイントを取る際に，一つ前のチェックポイントとの差分があるデータのみを保存するという手法で，差分のあるデータが半分あればチェックポイントサイズも半分に削減することができる．しかし，リカバリーの際に数世代のチェックポイントを読まなければならなくなるので，リカバリーのコストは通常のチェックポイントに比べ増大する．特に，インクリメンタルチェックポイントに関するオーバーヘッドとリカバリーコスト，双方を含めたコストの最適化の研究も行われている [13].

インクリメンタルチェックポイントでは，チェックポイントの圧縮は行っていないので，我々の手法と組み合わせることは可能であるが，本手法が利用しているデータがなめらかであるという点が利用できなくなってしまう．

また，可逆圧縮でのチェックポイントの圧縮率を高める研究もされている [16]．この研究では分散しているチェックポイントを変数毎にマージしてチェックポイントの数を減らし，変数毎に効率の良い圧縮手法を選ぶことでチェックポイントサイズの削減を行う．ここでの圧縮は可逆圧縮であるである点が我々の手法とは異なるが，チェックポイントのマージを行うことでチェックポイントのオーバーヘッドを 87%，リスタートのオーバーヘッドを 62%削減している．可逆圧縮ではあるが，チェックポイントのマージを行うことで I/O 時間の削減，圧縮効率の向上を達成することができている．

また，様々な非可逆圧縮手法 [20], [21] を用いて，その際に発生する誤差を気象シミュレーションのデータとして，欠損があるかを調査する研究もされている [10]．この研究では一定間隔で出力されるシミュレーションデータを圧縮することを対象としており，チェックポイントデータに非可逆圧縮手法を適用することは考えていない点，つまり誤差を持ったままアプリケーションの計算が進行することを考えていない点が我々の手法と異なる．誤差を持ったままアプリケーションの計算が進行することを想定してはいないが，気象シミュレーションとしての誤差の評価というでは非常に興味深い研究である．

6. まとめと今後の課題

本研究では，チェックポイント時間を短縮するため，ウェーブレット変換に基づくチェックポイントデータの非可逆圧縮手法について提案し，気象アプリケーション NICAM を用いて，圧縮時間，圧縮率，相対誤差について評価を行った．

その結果，特定の条件下で，相対誤差の最大が 5%以内で，チェックポイント時間を約 70%短縮できることを確認し，チェックポイント時間短縮に有効な手法であることを実証した．また，圧縮率と誤差がトレードオフの関係にあることも確認した．

今後の課題として，圧縮アルゴリズムの改善による圧縮

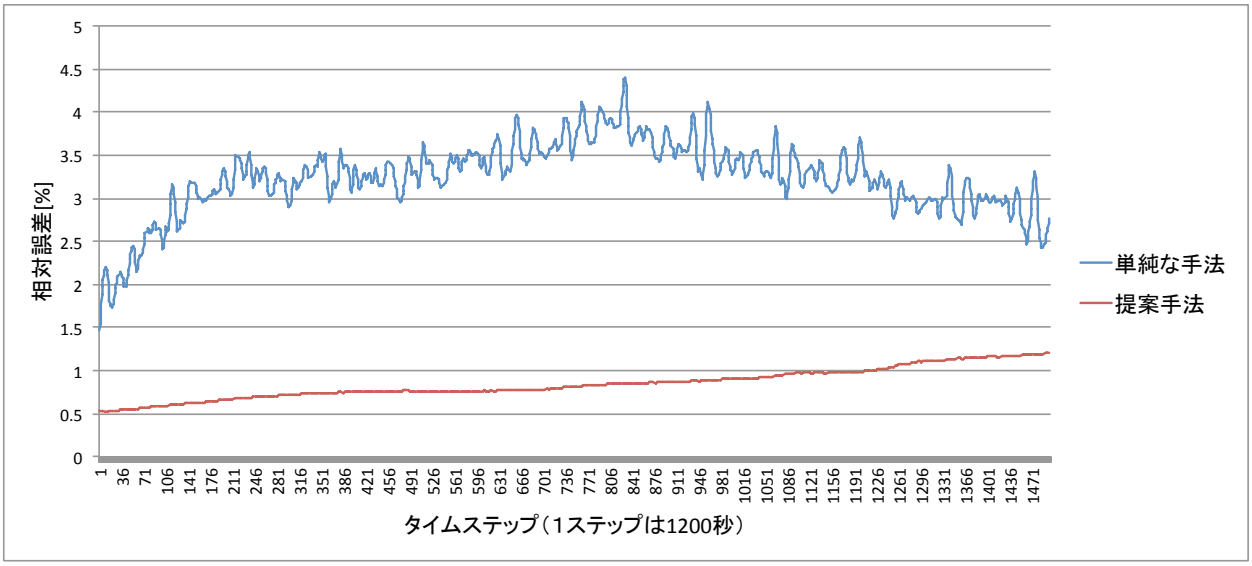

図 11 アプリケーションの進行による相対誤差の変化

率，誤差の改善やアプリケーション，変数配列毎に許容される誤差は異なるので，発生する誤差やアプリケーションの進行による誤差の変化の予測とそれに基づく誤差のコントロール，許容誤差内での圧縮率の最適化等が挙げられる．

謝辞 本研究は科学研究費助成事業（基盤研究（S）23220003）と JST-CREST の研究課題「ポストペタスケール時代のメモリ階層の深化に対応するソフトウェア技術」の支援によります．

参考文献

[1] Liang, Y., Zhang, Y., Jette, M., Sivasubramaniam, A. and Sahoo, R.: BlueGene/L Failure Analysis and Prediction Models, *International Conference on Dependable Systems and Networks(DSN 2006)*, pp. 425–434 (online), DOI: 10.1109/DSN.2006.18 (2006).

[2] Sato, K., Maruyama, N., Mohror, K., Moody, A., Gamblin, T., de Supinski, B. R. and Matsuoka, S.: Design and Modeling of a Non-blocking Checkpointing System, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Salt Lake City, UT, USA, IEEE Computer Society Press, pp. 19:1–19:10 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2388996.2389022⟩ (2012).

[3] Moody, A., Bronevetsky, G., Mohror, K. and De Supinski, B.: Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System, *2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11 (online), DOI: 10.1109/SC.2010.18 (2010).

[4] Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N. and Matsuoka, S.: FTI: High Performance Fault Tolerance Interface for Hybrid Systems, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, ACM, pp. 32:1–32:32 (online), DOI: 10.1145/2063384.2063427 (2011).

[5] Nagarajan, A. B., Mueller, F., Engelmann, C. and Scott, S. L.: Proactive Fault Tolerance for HPC with Xen Virtualization, *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, New York, NY, USA, ACM, pp. 23–32 (online), DOI: 10.1145/1274971.1274978 (2007).

[6] Graps, A.: An introduction to wavelets, *Computational Science Engineering, IEEE*, Vol. 2, No. 2, pp. 50–61 (online), DOI: 10.1109/99.388960 (1995).

[7] Satoh, M., Matsuno, T., Tomita, H., Miura, H., Nasuno, T. and Iga, S.: Nonhydrostatic icosahedral atmospheric model (NICAM) for global cloud resolving simulations, *Journal of Computational Physics*, Vol. 227, No. 7, pp. 3486 – 3514 (online), DOI: http://dx.doi.org/10.1016/j.jcp.2007.02.006 (2008).

[8] Said, A. and Pearlman, W.: An image multiresolution representation for lossless and lossy compression, *IEEE Transactions on Image Processing*, Vol. 5, No. 9, pp. 1303–1310 (online), DOI: 10.1109/83.535842 (1996).

[9] Ahmed, N., Natarajan, T. and Rao, K.: Discrete Cosine Transform, *IEEE Transactions on Computers*, Vol. C-23, No. 1, pp. 90–93 (online), DOI: 10.1109/T-C.1974.223784 (1974).

[10] Baker, A. H., Xu, H., Dennis, J. M., Levy, M. N., Nychka, D., Mickelson, S. A., Edwards, J., Vertenstein, M. and Wegener, A.: A Methodology for Evaluating the Impact of Data Compression on Climate Simulation Data, *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, New York, NY, USA, ACM, pp. 203–214 (online), DOI: 10.1145/2600212.2600217 (2014).

[11] Vaidya, N.: On Checkpoint Latency, *In Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems*, pp. 60–65 (1995).

[12] Plank, J. and Li, K.: ickp: a consistent checkpointer for multicomputers, *Parallel Distributed Technology: Systems Applications, IEEE*, Vol. 2, No. 2, pp. 62–67 (online), DOI: 10.1109/88.311574 (1994).

[13] Naksinehaboon, N., Liu, Y., Leangsuksun, C., Nassar, R., Paun, M. and Scott, S.: Reliability-Aware Approach: An Incremental Checkpoint/Restart Model in HPC Environments, *8th IEEE International Sympo-*

sium on Cluster Computing and the Grid, 2008. CCGRID '08, pp. 783–788 (online), DOI: 10.1109/CCGRID.2008.109 (2008).

[14] Plank, J. S., Xu, J. and Netzer, R. H. B.: Compressed Differences: An Algorithm for Fast Incremental Checkpointing, Technical Report CS-95-302, University of Tennessee (1995).

[15] Sancho, J., Petrini, F., Johnson, G. and Frachtenberg, E.: On the feasibility of incremental checkpointing for scientific computing, *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings*, pp. 58– (online), DOI: 10.1109/IPDPS.2004.1302982 (2004).

[16] Islam, T., Mohror, K., Bagchi, S., Moody, A., De Supinski, B. and Eigenmann, R.: MCREngine: A scalable checkpointing system using data-aware aggregation and compression, *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pp. 1–11 (online), DOI: 10.1109/SC.2012.77 (2012).

[17] Trivedi, K. S.: *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, John Wiley and Sons Ltd., Chichester, UK, 2nd edition edition (2002).

[18] Ziv, A. and Bruck, J.: Analysis of Checkpointing Schemes for Multiprocessor Systems, *Tech. Rep. RJ 9593, IBM Almaden Research Center*, pp. 52–61 (1993).

[19] Chandy, K. M. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Trans. Comput. Syst.*, Vol. 3, No. 1, pp. 63–75 (online), DOI: 10.1145/214451.214456 (1985).

[20] Woodring, J., Mniszewski, S., Brislawn, C., DeMarle, D. and Ahrens, J.: Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision, *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 31–38 (online), DOI: 10.1109/LDAV.2011.6092314 (2011).

[21] Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R. and Samatova, N. F.: Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data, *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par'11, Berlin, Heidelberg, Springer-Verlag, pp. 366–379 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2033345.2033384⟩ (2011).