「都市全域の地震等自然災害シミュレーションに関する研究（hp130015）」の高度化支援作業結果報告

Report of the Results of Program Tuning Support to "Research on Simulations of Earthquakes and Other Natural Disasters in the Entire City (hp130015)"

一般財団法人高度情報科学技術研究機構
志澤　由久、小林　　寛

RIST神戸センター利用支援部では、「京」利用支援の一つとして、東京大学地震研究所の課題「都市全域の地震等自然災害シミュレーションに関する研究（hp130015）」（HPCI戦略プログラム分野３の研究開発課題）の高度化支援を実施した。本支援を反映した研究成果は、2014年のゴードン・ベル賞ファイナリスト（最終候補）に選出された。本稿はその高度化支援結果について報告する。

１．はじめに

RIST神戸センター利用支援部では、「京」利用支援の一つとして、2013年度、東京大学地震研究所の課題「都市全域の地震等自然災害シミュレーションに関する研究(hp130015)」（HPCI戦略プログラム分野３の研究開発課題）の高度化支援を実施した。本支援を反映した研究成果［１］は、2014年のゴードン・ベル賞ファイナリスト（最終候補）に選出された。ゴードン・ベル賞はHPC分野で最も権威ある賞の一つであり、高い実行性能と計算科学技術への成果に対してACM（Association for Computing Machinery）が授与する賞である。本支援により実行性能の向上が認められ、そのことがファイナリスト選出に少なからず貢献できたと考えている。

以下に高度化支援作業結果の詳細について記述する。まず、プログラムの概要について説明する。次に、課題担当者から受けた支援依頼の内容を説明する。それに基づき作業目的と作業方針を設定し、それに沿って実施した作業内容の詳細を報告する。最後にまとめをする。

２．プログラム概要

プログラムで採用されているアルゴリズムの詳細は文献［１］に記述されている。ここではプログラムの概要のみを記す。

３次元の非線形波動場が計算対象である。変位ベースの四面体二次要素の有限要素法により解く。連立一次方程式のソルバーは共役勾配法を使用する。剛性マトリクスと変位ベクトルとの行列ベクトル積は、Element-by-Element法で計算する。ソースはFortran77（約5000行）で記述されている。ノード間はMPIを、ノード内はOpenMPを用いて、ハイブリッド並列化されている。

３．高度化支援の依頼内容

課題担当者から受けた支援依頼の内容は以下のとおりである。解きたい問題規模はフルノードで1000億自由度であり、必要な時間ステップ数は10万ステップである。現状では、昨年度、富士通殿が単体チューニング、及び64ノード位までのMPIチューニングを実施した。その結果では、1000ノード位まではスケールするが、4000ノード以降で不穏そうである。そこで、MPI及びI/O等のチューニン

グを行うことで、さらなる高速化と十分なスケーリングの実現を依頼された。

4．今回の作業の目的および作業方針

以上のような依頼に基づいて、今回の支援の目的、及び方針を、「並列性能の向上」と「単体性能の向上」に設定した。

並列性能の向上のために、通信、演算、I/Oの経過時間を処理ブロック毎に計測し、スケーラビリティを調査した。そして、阻害箇所があれば、原因を特定し、改善策を検討することとした。

単体性能の向上のために、まず性能解析ツールを用いてコスト分布を調査した。そして高コストな箇所を特定し、単体性能向上策を試行することとした。

5．プロファイラについて

「京」で提供されている幾つかの性能解析ツール（以下プロファイラ）を用い、性能分析を行った。各プロファイラの特徴を以下に記述する。

- 基本プロファイラ（fipp）
  - ➢アプリケーション全体、あるいは指定された区間の基本プロファイラ情報（アプリケーションのハードウェアモニタ情報やコスト情報等）を収集出来る。
  - ➢一定間隔でサンプリングを行い、その結果をコスト情報として出力する。
  - ➢比較的少ないオーバーヘッドで容易に基本プロファイラ情報を取得する事が可能。
  - ➢コスト分布を参照することで、負荷の高い箇所を特定可能。
- 精密PA（PA：Performance Analysis）
  - ➢指定された区間の詳細なハードウェアモニタ情報(キャッシュミス率や演算数等)を収集できる。
  - ➢ハードウェアカウンタ情報取得の都合、７回プログラムを実行しなければならない。
  - ➢問題箇所の原因を探るのに使用。

6．使用データ

プログラムを実行する際の入力データには、課題担当者より提供頂いたデータを使用した。使用したデータを表１に示す。上二つはストロングスケールを調べるためのデータで、下三つはウィークスケールを調べるためのデータである。

ストロングスケールのデータ（領域数：64，256）は、全体で１億自由度、64ノードのデータを基点として、同じ自由度で、分割数＝ノード数が４倍だけ違う関係にある。

ウィークスケールのデータ（領域数：256，4096，16384）は、１億自由度256ノードを基点として、自由度とノード数が同じ倍率を保って、16倍、64倍の関係にある。

表１　使用データ

	テストデータ名	要素タイプ	自由度の桁数	領域数＝実行ノード数＝実行時プロセス数	1ノード当りのコア数＝1ノード当りのスレッド数	タイマー挿入した区間での1ノード当り使用メモリ（ファイルから読込）
strong scaling	case2	四面体二次	1億	64	8	1595.6 MB
weak scaling	case2	四面体二次	1億	256	8	435.7 MB
	case3	四面体二次	16億	4096	8	459.5 MB
	case4	四面体二次	64億	16384	8	483.5 MB

「京」で実行する際のラージページの設定は、以後、全て4MBとした。基本プロファイラ使用によるオーバーヘッドは無視できること、及び計測したい箇所にタイマーを挿入したことによるオーバーヘッドは無視できることを確認した。時間ステップ数は100とした。

7．基本プロファイラによるコスト分布調査：asis版、256ノード

asis版（オリジナル版）について基本プロファイラを用いてコスト分布を調査し、高コストな箇所を特定する作業を実施した。以下にその結果を報告する。ノード数256のデータを使用し、コンパイルオプションは-Kfast，openmpとし、１ノード当り８コアを使用した。

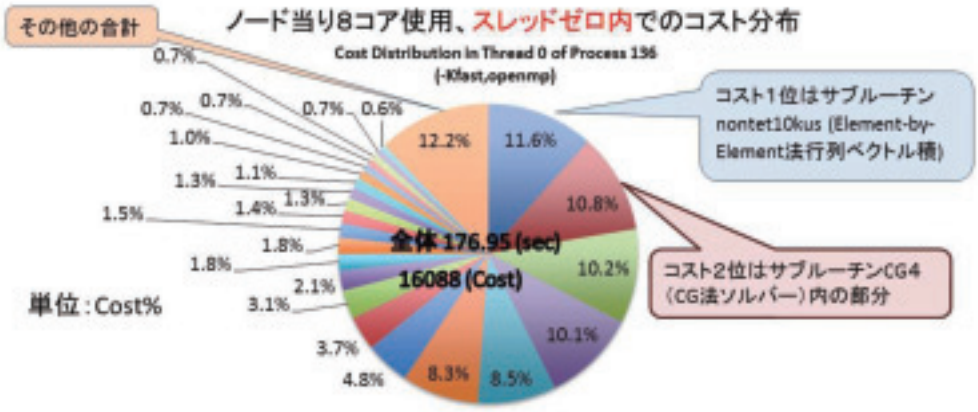

図１　基本プロファイラを用いたコスト分布調査結果。一つのプロセスにおける、スレッドゼロでのコスト分布。

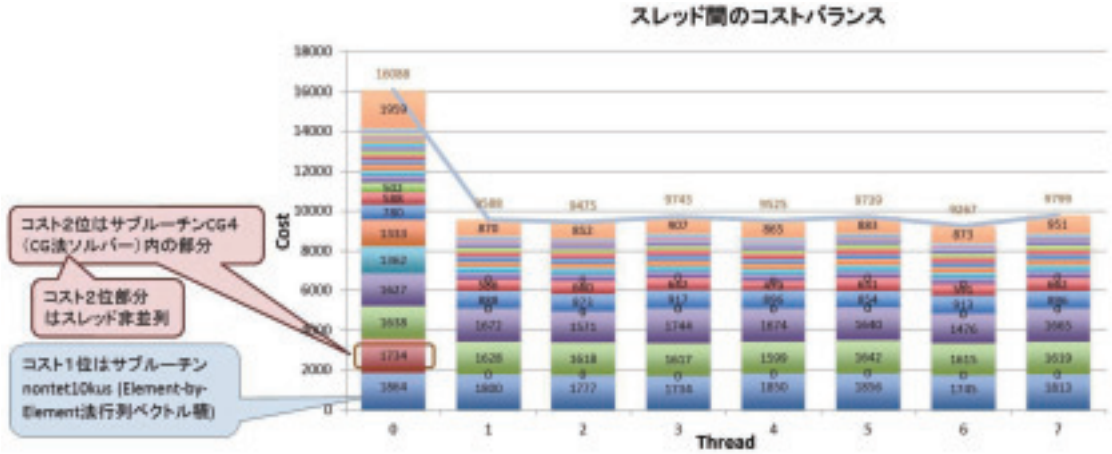

図２　基本プロファイラを用いたコスト分布調査結果。図１と同じプロセスにおける８個のスレッド間のコストバランス。コスト２位はスレッド非並列箇所であることが分かった。

図１の円グラフは、一つのプロセスにおける、スレッドゼロでのコスト分布である。コスト１位は11%を占めていて、それはnontet10kusという名前のサブルーチンであることが分かった。ソースコードを見ると、これはElement-by-Element法で行列ベクトル積を計算しているサブルーチンであることが分かった。コスト２位は10%を占めていて、それはCG法ソルバーのサブルーチン、CG4内の部分であることが分かった。これがどのような部分なのかを、図２の棒グラフで表した。この棒グラフは、図１と同じプロセスにおける８個のスレッド間のコストバランスを示している。図１の円グラフは、図２の棒グラフにおける、スレッドゼロの部分に対応している。棒グラフから、コスト２位の赤い部分はスレッドゼロにのみ存在していて、スレッド非並列箇所であることが分かった。

８．プログラムの構造の概要

これらの高コスト箇所がプログラムの中でどのような位置にあるのかを把握するためにプログラム構造を調査した。プログラム構造図を図３に示す。プログラムは、まずグリッドデータを読み込み、時間ステップループに入り、その中で計算を行い終了するという構造をしている。時間ステップループの中で、時間ステップごとの入力データを読み込み、CG法のソルバーで解いて、計算結果を時間ステップごとにファイルに出力する。入力は

ループの外で一度だけファイルオープンし、一つのプロセスは一つのファイルから次々と読んでいく。これに対して出力のほうは、ループ内でその都度ステップ毎の別ファイル名をオープンして書き出しクローズしている。

図３　プログラム構造図

CG法のソルバーは、図４のような計算で構成されている。これらのうち、各領域内でベクトルの内積計算する処理と、ベクトルのスカラー倍や、和・差の計算にスレッド非並列部分がある。これらの部分の合計がコスト２位箇所に対応している。

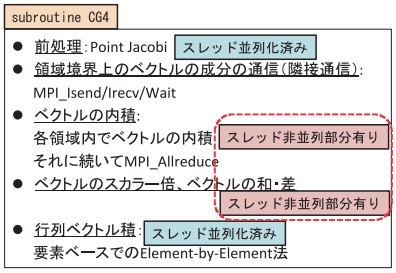

図４　CG法のソルバーを構成する処理の種類。赤い点線で囲った部分の合計はコスト２位箇所に対応する。

行列ベクトル積について、構造の概略を図５に示す。まずスレッドのループがあり、その内側に各スレッドで担当する四面体要素のループがある。コスト１位箇所のサブルーチンnontet10kusは、一つの四面体要素として取得したデータについて計算する処理に対応している。

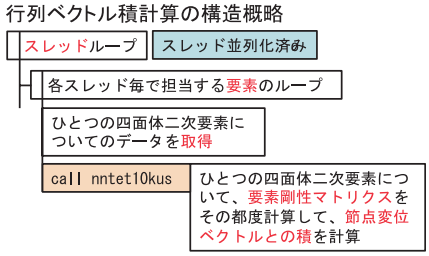

図５　行列ベクトル積の構造概略。サブルーチンnontet10kusはコスト１位箇所に対応する。

９．並列性能向上のための改善作業

改善作業結果の報告として、まず、並列性能向上のための改善作業の報告から入ることとする。

本プログラムの場合、基本プロファイラによる計測ではI/Oのコストは計測されない。I/Oの時間も把握するため、asis版の時間ステップループ内について、タイマーMPI_Wtimeを挿入し、通信、演算、及びI/Oの時間を個別に計測することとした。さらにその結果からスケーラビリティを調査した。阻害箇所があれば、それらを特定する作業を実施した。スケーラビリティの調査結果を図６に示す。横軸がノード数であり、64から256ノードがストロングスケール、256から１万6000ノードがウィークスケールデータである。縦軸が経過時間である。青の実線が時間ステップループ合計の経過時間で、点線がその内訳である。

赤の点線の演算部分について見てみると、64から256ノードで時間が約１/４になっており、ストロングスケーラビリティは良好であるといえる。256ノード以降で時間はほぼ一定であり、ウィークスケーラビリティも良好であるといえる。

対して、ファイル出力がスケーラビリティを阻害していることがみてとれる。

また、asis版には、もともとMPI_Barrier

が通信ルーチンの前後に挟まれており、１ステップ当り約1400回と数多くコールされていることが分かった。これらのバリアをコールする時間も計測対象としたところ、バリアもスケーラビリティを阻害していることが分かった。さらに、バリアは計算ロジックに何の影響も無く、削除できることを確認した。

以上の考察から、ファイル出力の改善とバリアの除去を行った。

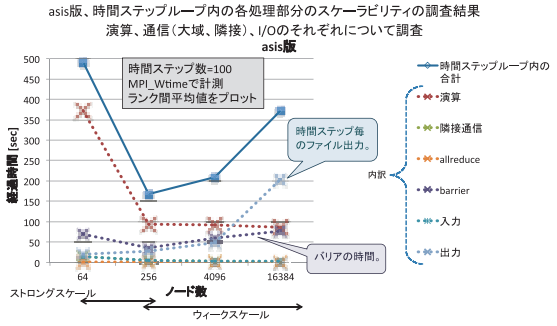

図６　スケーラビリティの調査結果

9.1　ファイル出力の改善

ファイル出力の改善効果の検討結果（ファイルサイズの観点）

まず、ファイル出力の改善作業結果から報告する。表２にファイル出力の改善に関する検討結果、及びasisからtune版への改善点を示す。

asis版では各ランクはステップ毎に別ファイルに出力するという特徴がある。従って、ステップ数が多くなればなるほど出力ファイル数も多くなる。この方法の弱点は、より長いステップ数だとステージアウトのファイル数の上限値を超えてしまうということである。

そこで改善方法として、各ランクは時間ステップを通して同一のファイルに出力するようにした(表２の上側の赤い点線部分)。さらに、テキストで書き出していたものを、tune版ではバイナリで書き出すように改めた（表２の下側の赤い点線部分)。これにより、ファイルサイズの削減が見込まれ、実際一回の書き出しでのサイズは46%に削減された。書き出しをバイナリにすることは、テキスト変換の処理が無くなることを意味しており、その効果による経過時間の短縮が期待できる。

表2　ファイル出力の改善に関する検討結果

		asis版	tune（ファイル出力改善）版
弱点		より長時間ステップだと、ステージアウトのファイル数の上限値：８００＊ノード数＋１０００を超えるので、現実的でない。	
特徴		各ランクはステップ毎に別ファイルに出力	各ランクは時間ステップを通して同一のファイルに出力
オープン・クローズの回数		25	1
100ステップでのファイル出力の回数		25（4ステップ毎）	25（4ステップ毎）
ファイル形式		テキスト	バイナリ
1ステップ内書き出しでの出力ファイルのサイズ（MB）ランク間平均値	4096ノード	0.361	0.165 （asisに比べて45.7%に減少）

テキストへの変換処理がなくなることを意味する。

ファイル出力の改善効果の計測結果

ファイル出力の時間を計測した結果を図７に示す。左のグラフが縦軸を経過時間としたグラフである。ファイル出力の時間は無視できる程度にまで削減された。右のグラフはスケーラビリティを見るためのグラフである。縦軸は256ノードでの時間を分母として時間を規格化したものである。スケーラビリティはストロング、ウィークともにほぼ理想値へと改善したことがみてとれる。

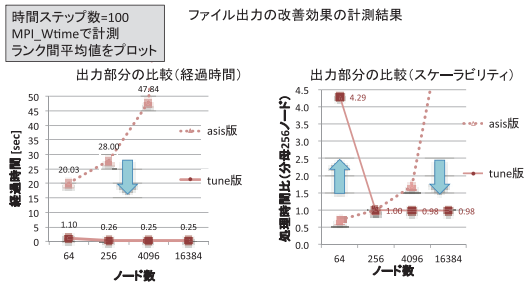

図７　ファイル出力の改善効果の計測結果（経過時間及びスケーラビリティ）

次にランク間のばらつきの計測結果を表３に示す。ばらつきの時間を「ランク間の最大ー最小」と定義すると、オープン・クローズやテキスト変換処理がなくなったことにより、特に4000ノード以上で、ばらつきの時間が大幅に減少した。

表３　ファイル出力の改善効果の計測結果
（出力のばらつき）

出力のばらつき＝ランク間最大値 ― ランク間最小値（単位：秒）

ノード数	asis版	tune版
64	0.70	0.21
256	2.74	0.05
4096	20.50	0.07
16384	75.95	0.08

時間ステップを通して同一のファイルに出力する改善手法、及びバイナリで書き出すようにする改善手法は、本アプリケーションプログラムに限らず、「京」で性能向上させるための必須の改善手法であるといえる。

9.2　バリアの除去

バリアの有無に関する注意点

バリアの除去結果について報告する前に、まず注意点にふれる。asis版では、バリアは100ステップで約14万回コールされている。asis版では、バリアは通信ルーチンの前後に挟まれている。それ故、計測されたバリアの時間の中には、バリアの純粋な時間に加えて、演算やI/Oのロードインバランスに起因する、同期待ち時間も計上されている。一方、バリア除去したtune版では、同期待ち時間は、allreduceや隣接通信の計測時間の中に計上されている。バリアを除去することで、少なくともバリアを100ステップで14万回コールする際に発生するバリアの純粋な時間の削減が期待できる。

バリア除去の効果の確認

効果を確認するため、バリアとallreduceと隣接通信の時間を合計した値を比較することとした。結果を図８に示す。グラフより、確かに削減効果が得られたことが確認できた。特に4096ノードの時23．74秒短縮である。

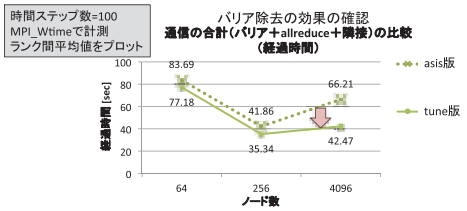

図８　バリア除去効果の確認結果

9.3　バリア除去＋ファイル出力改善による総合的な効果の確認

バリア除去とファイル出力改善による総合的な効果を時間ステップループについて確認する作業を実施した。結果を図９に示す。経過時間については、4096ノードの時、1.5倍の高速化を達成した。256ノードの時間を分母にして規格化し、スケーラビリティをみると、ストロング、ウィークともに改善した。

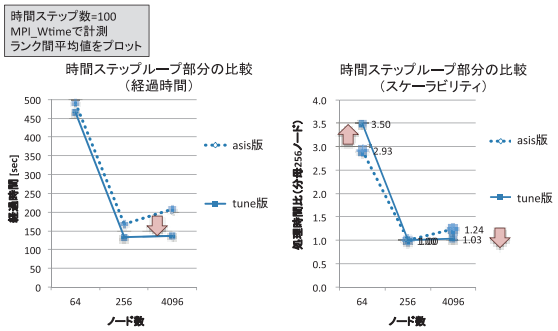

図９　バリア除去とファイル出力改善による総合的な効果の確認結果

10.　単体性能向上のための改善作業

次に、単体性能の改善作業の結果報告に移る。ここでは実際に行ったチューニングのうち、コスト１位：nontet10kusルーチンの高速化、及びコスト２位：CG4ルーチンのスレッド非並列部分のスレッド並列化について報告する。

10.1　コスト1位：nontet10kusルーチンの高速化（Element-by-Element法による行列ベクトル積計算部分）

まず、コスト１位ルーチンの高速化につい

て報告する。このサブルーチンは、ひとつの四面体二次要素（自由度30）について、要素剛性マトリクスを生成して、節点ベクトル（配列要素数30）に掛ける処理を行う、というものである。計算に用いる１個の四面体データは小さいのに対して、計算は複雑で多いため、要求B/Fは低い。要素数回コールされるため、高コストである。

nontet10kus asis版～nontet10kusを切り出したテストプログラムの性能分析～

サブルーチンを切り出したテストプログラムを作成（64ノードデータ使用）して、性能分析と改善策を試行する方法をとった。まず、asis版の精密PAの結果を図10に示す。このサブルーチンは１個の四面体に関する計算である。初回使用時にメモリから取得した計算に必要なデータは、その後L1キャシュ上に全て乗る問題である（メモリスループットの測定値はゼロである）。演算性能ピーク比は19.89%である。

nontet10kus asis版
～nontet10kusを切り出した、テストプログラムの性能分析（64ノードデータ使用）～

	経過時間[s]	浮動小数点演算ピーク比	MFLOPS	浮動小数点演算数
asis版	7.72	19.89%	3182	2.46E+10

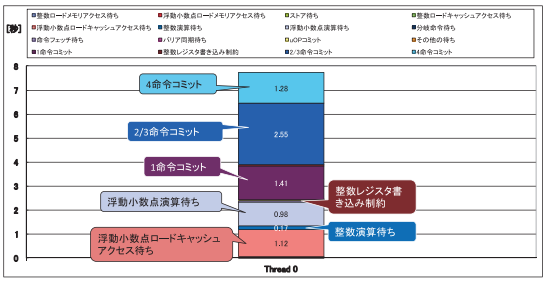

図10　nontet10kus asis版 を切り出したテストプログラムの性能分析（64ノードデータ使用）。nontet10kusのコール回数は64とした。１コール当りの経過時間は0.121秒である。

基本プロファイラによるnontet10kus asis版の性能分析

これ以上の高速化が可能かどうか検討するために、基本プロファイラでこのサブルーチンの高コスト箇所を調査した。各実行文について、コストの高い順に示したのが図11である。

一番コストが高いのは、計算結果を格納する配列BDBuのゼロクリアであり、全体の10%ぐらいを占めていることが分かった。配列BDBuのサイズは120byteと小さいが、他の積和演算はL1オンキャッシュのため、メモリアクセスが必要なこの部分が相対的に高コストである。

積和演算部分が２%前後で、その後に続いている。これは、積和演算はL1オンキャッシュだが、複雑な計算が多数存在しているためである。

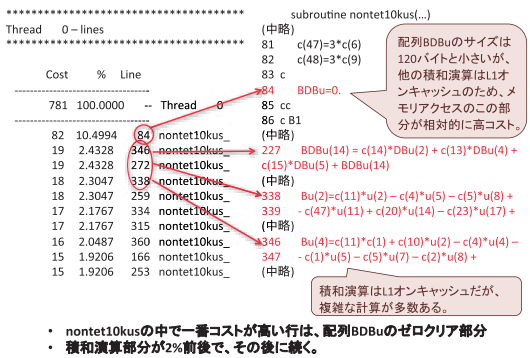

図11　nontet10kus asis版の基本プロファイラによる性能分析結果

nontet10kusチューニング：配列BDBuのゼロクリア削除

配列BDBuには計算結果を次々と足し込む処理が行われていて、ゼロクリアは不要であることが確認できた。そこで、ゼロクリアをコメントアウトし、BDBuへの一回目の計算値代入に関して、BDBuの加算処理を削除した（図12参照）。

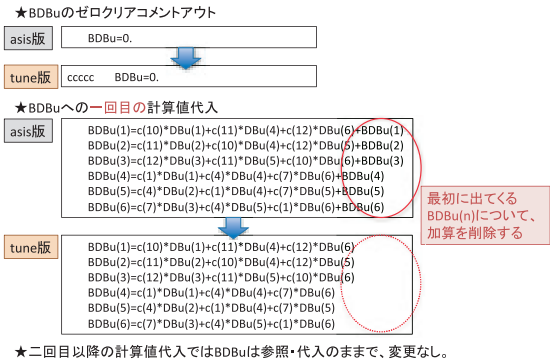

図12　nontet10kusチューニング例1：配列BDBuのゼロクリア削除

nontet10kusチューニング： 重複計算除去

何度も現れる計算パターンについて、一時変数に格納し、それを使いまわすことで演算数を削減した。その例を図13と図14に示す。図13一行目におけるc（20）*... という式は何度も現れるので、左辺の緑色の一時変数に格納して、図13の３行目の右辺で使うようにした。

さらに図13の３～４行目にかけての赤及び緑字で記した長い式も、図14の上側の四角内で赤字で記した様に同じパターンで何度も現れるので、図13の２行目の左辺のように、変数名は長いが一時変数に格納して、これらの計算式が現れる箇所で、図14の下側の四角内の赤字部分の様に置き換えて使いまわした。このようにすることで演算数を削減した。

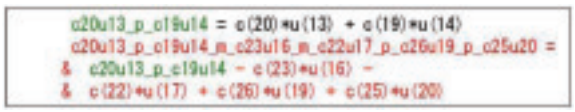

図13　nontet10kusチューニング例2：重複計算除去

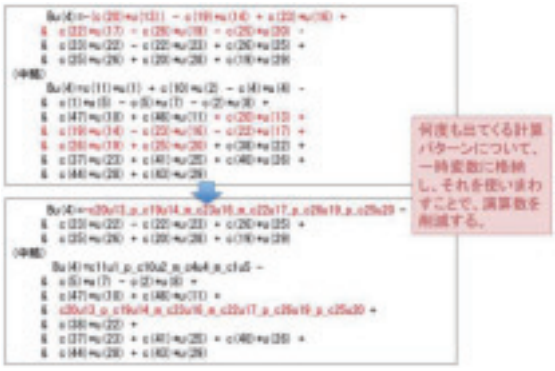

図14　nontet10kusチューニング例2：重複計算除去（続）

nontet10kus チューニング版～nontet10kusを切り出したテストプログラムの性能分析～

チューニング後のコードに対する精密PAを計測した結果を図15に示す。演算数は8.1%減少を確認した。演算性能ピーク比は21%に向上した。経過時間は15.8%短縮した。

チューニング後の状態で、命令コミット系の時間が減少しており、また、各命令コミットの占める割合に、若干の変化が見られる。このような現象が、演算数の変化率と経過時間の変化率が必ずしも一致しない原因の一つなのかもしれないと推察している。

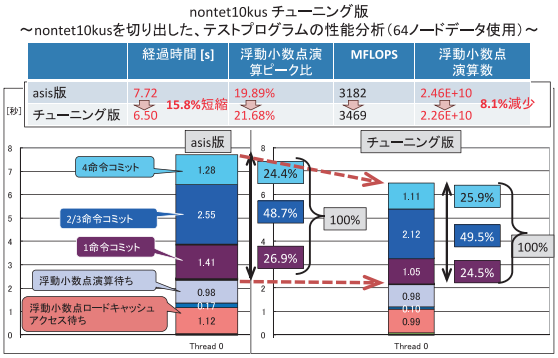

図15　チューニング前後のコードに対する精密PA計測結果の比較

10.2　コスト2位：CG4スレッド非並列部分のスレッド並列化及びその他チューニング（例：内積の計算）

次に、コスト２位のCG4スレッド非並列部分のOpenMPスレッド並列化チューニング

について報告する。ここでは、例として、二つの節点ベクトルの内積計算のコードを取り上げる。このコードには、節点内自由度のインデックス計算の除去、及び内積計算をスレッド並列化する際にOMP REDUCTIONを用いない手法も適用できるので、合わせて適用した結果を示す。

asis版の内積計算コードを図16に示す。asis版のコードでは、まず節点のループがあり、その内側に節点当り３つの自由度に関するループがある。節点ベクトルｚとｑを掛けて、さらに重みファクターｗを掛けて、zqに足し込んで、内積を求めている。その際、節点番号と節点内自由度の番号から配列ｚに対するインデックスを図16の赤い点線で囲まれた部分の引数の式のように計算している。

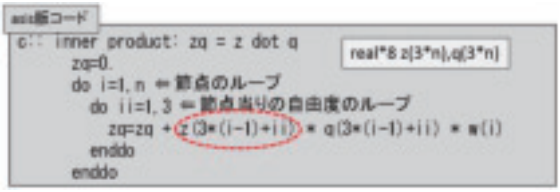

図16　内積計算のコード（asis版）

節点内自由度のインデックス計算の除去

チューニング版のコードを図17に示す。チューニング版では、赤い点線で囲まれた部分のように配列ｚを二次元配列に変更し、一次元目は節点内自由度のインデックス、二次元目は節点番号とすることで、余計なインデックス計算が不要となる。さらに、重みファクターｗは節点番号のみに依存することに着目して、節点当り三つの自由度のループを展開し、まとめて括ってからｗを掛けることで、掛け算の数を減らすことができる。

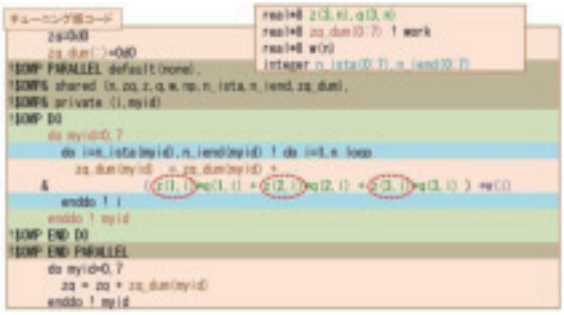

図17　内積計算のコード（チューニング版）

スレッド並列化、及びREDUCTIONの回避

次に、チューニング版のコード（図17）における、スレッド並列化及びREDUCTIONの回避の仕方について説明する。スレッド番号myidのループと、各スレッドでの担当範囲内での節点iのループ、というように二段階にする。外側のスレッドのループをOpenMPでスレッド並列化する。内側の節点ループについて、各スレッドが担当する範囲の下限・上限を明示的に前もって作成し、配列n_istaとn_iendに格納したものを参照して下限･上限を指定する。スレッドmyidでの部分和は、一旦、配列要素zq_dum（myid）に格納する。配列zq_dumの配列要素をスレッド番号順に和をとる。このようにすることでREDUCTION指示文を使用せずにスレッド並列化できる。この方法の利点は実行形式（逐次／スレッド並列）によらず常に同一な計算結果が得られることである。

参考として、スレッド番号順に和をとる別法として、REDUCTIONを使用し、かつ、コンパイルオプション-Kordered_omp_reductionを用いる方法もあるということに触れておく。

チューニングの効果の確認

処理内容と適用した手法との対応は表４のようになる。これらの処理内容の合計に相当する部分（通信含まず）について、チューニング前後で経過時間を比較した結果を図18に示す。64ノードで約1.8倍の高速化を達成した。理想値１/８に届かない理由は、ループ

内の演算が少なくスレッド並列化のオーバーヘッドが大きいためである。

表４　処理内容と適用した手法との対応

処理内容＼手法	スレッド非並列部分のOpenMP並列化	節点内の3つの自由度を参照するインデックス計算を除去	OMP REDUCTIONの回避
ベクトルの内積	*	*	*
ベクトルのスカラー倍、ベクトルの和・差	*	—	—

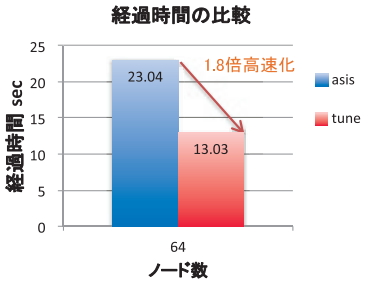

図18　表４の処理内容の合計部分（通信含まず）の経過時間の比較結果

11.　総合的な改善効果の確認

以上の単体性能と並列性能向上策の総合的な効果の確認のための計測を時間ステップループに対して行った。その計測結果として、経過時間、スケーラビリティ、演算性能、及びメモリスループットについて報告する。

11.1　最終チューニング版の計測結果の要約

最終チューニング版の時間ステップループ（I/Oと通信を含む）部分の各種計測結果を図19に示す。経過時間については、１万6000ノードの時、3.2倍の高速化を達成した。スケーラビリティについても、ストロング、ウィークともに改善した。チューニングにより、演算性能、メモリスループットともに向上した。特に、１万6000ノードの時の演算性能の向上率は2.9倍、メモリスループットの向上率は3.2倍となった。

12.　まとめ

並列性能の向上作業として、バリアの除去、及びファイル出力の改善を実施した。ファイル出力の時間は、ほぼ無視できる程度にまで削減され、スケーラビリティも改善した。また、時間ステップループのスケーラビリティも改善した。

単体性能の向上作業として、高コストルーチンのチューニング、及び各種チューニングを実施した。コスト１位nontet10kusについ

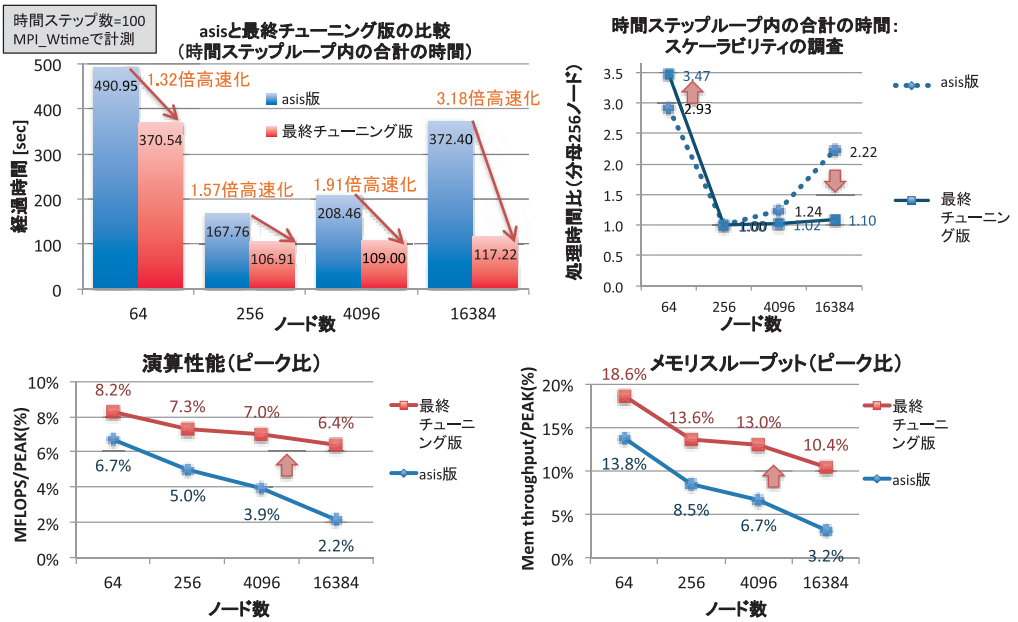

図19　最終チューニング版の計測結果の要約（時間ステップループ部分の性能）

ては、64ノードの時、1.19倍高速化し、演算性能ピーク比は21.68%に向上した。コスト２位CG4スレッド非並列部分は、64ノードの時、1.8倍高速化した。

並列性能と単体性能の向上策を併せて実施したことで、大幅な改善が達成された。１万6000ノードでのステップ数100の時間ステップループ（I/Oと通信を含む）部分について、asisと最終チューニング版の比較を表５に示す。経過時間は3.2倍の高速化を達成し、演算性能ピーク比は2.2%から6.4%に2.9倍向上し、メモリスループットピーク比は 3.2%から10.4%に3.2倍向上した。

表５　ノード数16384での時間ステップループ（I/Oと通信を含む）部分の性能比較

バージョン	経過時間［sec］	演算性能ピーク比	メモリスループットピーク比
asis版	372.40	2.2%	3.2%
最終チューニング版（向上率）	117.22（3.2倍）	6.4%（2.9倍）	10.4%（3.2倍）

なお、本高度化申請後のチューニングにより、本支援を反映した研究成果［１］では、270億自由度の問題に対して、京コンピュータ全系（82944ノード）で演算性能ピーク比6.93%（0.736 PFLOPS）の実行性能を出すに至っている。

参考文献

［１］Tsuyoshi Ichimura, Kohei Fujita, Seizo Tanaka, Muneo Hori, Maddegedara Lalith, Yoshihisa Shizawa, and Hiroshi Kobayashi: Physics-based urban earthquake simulation enhanced by 10.7 BlnDOF x 30 K time-step unstructured FE non-linear seismic wave simulation, Proceedings of the 2014 ACM/IEEE conference on Supercomputing (SC' 14), (ACM Gordon Bell Prize Finalist), 2014, in press.