Vol.2016-HPC-155 No.38
2016/8/10

3 次元積層技術による高メモリバンド幅時代の自動チューニング ～FDM コードを例にして～

片桐 孝洋†, 松本 正晴††, 大島 聡史††

本報告では，3 次元積層メモリ技術の導入により，従来のコード最適化方式，および自動チューニング（AT）方式がどのように影響するかを検証する．従来メモリ構造をもつ Fujitsu PRIMEHPC FX10 と，3 次元積層技術によるメモリをもつ Fujitsu PRIMEHPC FX100 を利用し，有限差分法(FDM)コードである ppOpen-APPL/FDM を用いて性能評価を行った．その結果，FX10 から FX100 に移行したときの速度向上率は，全体時間で AT なしの実行時間で 4.42 倍，AT ありの実行時間で 3.74 倍，および，演算カーネル時間のみの速度向上率で，AT なしの実行時間で 5.24 倍，AT ありの実行時間で 6.58 倍と，メモリアクセスに関するハードウェア性能比率に対し妥当である結果を得た．また性能プロファイル結果によると，浮動小数点ロードキャッシュアクセス待ち時間の占める割合が，FX10 では 37.18%～39.31%，FX100 では 13.14%～14.61%と削減されていることが確認できた．一方で，FX100 では FX10 よりもバリア同期待ち時間の占める割合の大幅な増加が確認された．この原因は解析中であるが，3 次元積層メモリ技術などにより相対的にメモリアクセス性能が向上した場合，従来とは異なる観点でのコード最適化を考慮しなくてはいけないことを示唆している可能性がある．

1. はじめに

近年，ムーアの法則の崩壊により，計算機の演算性能が向上しなくなると言われている．一方，3 次元積層技術などメモリ内のデータ転送能力の向上は相対的に達成されると予測する研究者もいる．この仮定に基づくと，従来の演算性能（FLOPS）の向上に基づく計算方式から，データ移動性能（BYTES）に基づく計算方式に再構築し，従来のソフトウェアスタックの再利用や再構築をすべき，という主張がなされている[1].

そこで本報告では，上記の主張である **FLOPS から BYTES への転換**を前提とし，数値計算アルゴリズムの再構築の可能性について調査する．そこで，現在普及しつつある 3 次元積層技術による高バンド幅なメモリをもつ計算機に移行するとき，従来の最適化方式がどのように変化するかを検証する．この際，従来技術である自動チューニング（Auto-tuning, 以降 AT）は，将来に渡り高性能計算のための必要な技術の 1 つになっていると著者は予想している．この観点から, AT による最適化結果についても予備評価する．

対象となるアプリケーションは現時点では限定されてはいるが，数値シミュレーションにおける主要な計算手法の 1 つであり，かつ，メモリに対するアクセスの多い特性を持つ，有限差分法（Finite Difference Method, FDM）による実アプリケーションを対象とする．

本報告の構成は以下のとおりである．2 章で関連研究について述べる．3 章は，本報告で利用する AT 専用言語である ppOpen-AT[2]-[5]の説明と，評価対象となる FDM を用いたアプリケーションにどのように AT 方式を実装したかを説明する．4 章は，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，本報告で得られた知見について述べる．

2. 関連研究

AT 研究では，FFT や基本線形代数副プログラム集（BLAS）に代表される数値計算処理の特定アプリケーションを対象にした AT 方式[6]-[8]や，疎行列-ベクトル積や疎行列反復解法に特化した AT 方式[9]-[14]が研究されてきた．また，計算機システムやプログラムを対象にした汎用方式[15]-[24]まで，幅広いアプリケーションと計算機環境においても研究が継続されている．

本研究では，ポストムーア時代に想定される，3 次元積層メモリ技術などによるデータ移動性能の相対的な向上による影響を取り扱う．また，改装メモリ型計算機に向く AT 方式の影響も取り扱う. この 2 つの観点からの AT 研究は，著者の知る限りほとんど行われていない.

3. ppOpen-AT と ppOpen-APPL/FDM への階層型 AT 処理の実装

3.1 概要

ppOpen-AT は，AT 機能付数値計算ミドルウェア ppOpen-HPC [25]のための AT 機能を提供するための計算機言語として開発された．ppOpen-AT の主要 AT 技術は，ABCLibScript [15]から引き継いでいるが，ループ消滅，ループ分割, および, 式並び替え機能[2]-[4]が追加されている．

一方，著者らは既存研究[26]において，AT 候補を全て静的に取り込むとこで AT 機能付ソフトウェアを構築するソフトウェア構築方法論 SCG-AT を示した．この SCG-AT を採用し，FDM の実アプリケーションに AT 機能を適用することで，実用的なコード量の増加で AT 機能が実装できることを示してきた[2]-[5].

† 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
†† 東京大学 情報基盤センター スーパーコンピューティング研究部門

3.2 階層型 AT 処理

既存研究[5][26]において「階層型 AT 処理」の概念を示した．ここで階層型 AT 処理とは，ある箇所（A）に AT が記述されている場合に，その箇所の内部（B），および，その箇所の内部から呼び出される手続き内（C）にも AT が記述されている AT の処理である．一般的に，B および C 内にも AT が記述されている処理も，階層型 AT となる．

例えば，ある主ループに AT が記述されており，その主ループ内から呼び出される手続き内にも AT が記述されている場合（図 1）である．

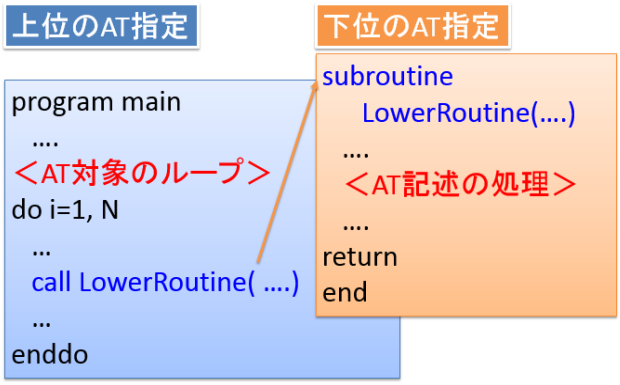


図 1　階層型 AT 処理の例（2 階層）

図 1 の階層型 AT 処理は，その構造から，多階層メモリ向け最適化の AT に必須な AT の実現方式と考えられる．また，上位の AT 指定でアルゴリズム選択（コード選択）を行い，下位の AT 指定で，選択されたアルゴリズムに向く実装方式を選択すると，異なる計算機アーキテクチャ間でも最適となるコードを選択できる可能性がある．

このように，階層型 AT 処理では，現在の高 FLOPS 指向の CPU 向き実装と，将来の高 BYTES 指向の CPU 向き実装を自動的に切り分ける仕組みを提供できる可能性がある．

3.3 ppOpen-APPL/FDM

3.3.1 処理の流れ図

我々は既に，階層型 AT 処理を ppOpen-HPC [25]で提供している地震波の FDM シミュレーションコード（アプリケーション）Seism3D [27]を構成するライブラリ ppOpen-APPL/FDM に対し AT 適用を行い，さらに AT の有効性の評価を行った[5][26]. ppOpen-APPL/FDM の処理の流れ図を図 2 に示す．

図 2 の計算方式は陽解法である．そのため，演算を行う箇所（演算カーネル）が分散されて配置されており，AT 対象である演算カーネルは複数存在する．手動で性能チューニングを行う場合は，複数の演算カーネルについて，それぞれ行う必要があるため，煩雑であり時間的コストがかかる．そのため，AT を導入することで，チューニングのためのコストを減少させるには好適な事例となる．

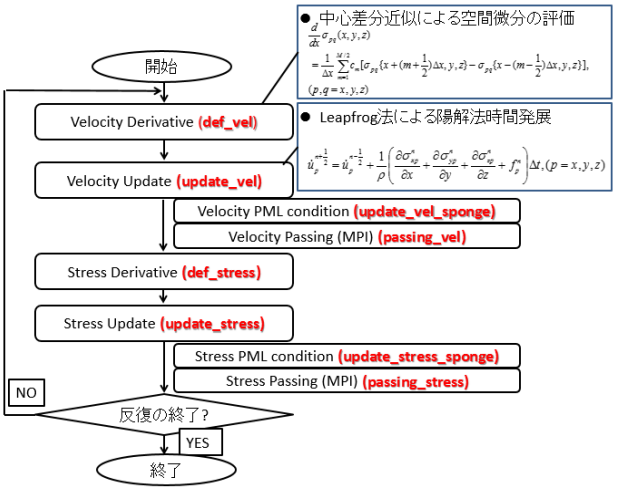


図 2　ppOpen-APPL/FDM の処理の流れ図

3.3.2 ベクトル向き実装とスカラ計算機向き実装

我々は ppOpen-APPL/FDM において，従来用いていたベクトル計算機向きコードを基にした AT 実装[2]-[4]に対し，スカラ計算機向きコードを開発した[26]. さらに，このスカラ計算向きコードにおいて，最内ループ中の IF 文を削減し高速化する，IF-free コード[5]を開発した．

3.3.3 ppOpen-APPL/FDM における階層型 AT 処理

3.3.2 節の，(1)ベクトル計算機向きコード，(2)スカラ計算機向きコード，および，(3)スカラ計算機向きコード(IF-free)の 3 種のコード（アルゴリズム選択）を上位の AT 方式として実装した．

また，下位の AT として，ループ消滅とループ分割を中心としたコード変換の AT を実装した．両者を合わせた階層型 AT 処理としての参照実装とその評価を，文献[5]で行った．ここでは，この文献[5]の AT 方式による実装を対象とする．

表 1 に，ppOpen-APPL/FDM ver.1.0.0 におけるカーネル名と AT 候補をまとめる．

表 1　カーネル名と AT 候補

カーネル名	AT対象	候補数
1. update_stress	・ループ消滅とループ分割：8種 ・コード選択: 3種	10
2. update_vel	・ループ消滅、ループ分割、および式並び替え：6種 ・コード選択：3種	8
3. update_stress_sponge	・ループ消滅：3種	3
4. update_vel_sponge	・ループ消滅：3種	3
5. ppohFDM_pdiffx3_p4	カーネル名：def_update、def_vel ・それぞれ、ループ消滅：3種	3
6. ppohFDM_pdiffx3_m4		3
7. ppohFDM_pdiffy3_p4		3
8. ppohFDM_pdiffy3_m4		3
9. ppohFDM_pdiffz3_p4		3
10.ppohFDM_pdiffz3_m4		3
11. ppohFDM_ps_pack	データパックとデータアンパッキング ・ループ消滅：3種	3
12. ppohFDM_ps_unpack		3
13. ppohFDM_pv_pack		3
14. ppohFDM_pv_unpack		3

表 1 から，現在の実行では 14 カーネルがある．また，AT 候補数は 54 種類となる．ハイブリッド MPI/OpenMP の実行毎に，表 1 の組み合わせ候補がある．

いま，ハイブリッド MPI/OpenMP の実行数を 7 組とすると，全ての候補は 54×7＝378 種にものぼる．そのため，ハイブリッド MPI/OpenMP 実行においては，人手ではチューニングがより一層困難となってきており，AT 技術の適用が望まれている．

表 1 では，上位の AT として，update_stress および update_vel カーネルにおけるコード選択 3 種がある．この 3 種は，3.3.3 節で説明した(1)ベクトル計算機向きコード，(2)スカラ計算機向きコード，および，(3)スカラ計算機向きコード(IF-free)である．一方，下位の AT として，ベクトル計算機向きコードで使われている def_update および def_vel があり，これらの実体はカーネル 5 番から 10 番である．これらのコードは，典型的な 4 次の差分コードである．

3.4 ベンチマーク問題設定

本評価で利用する問題領域の大きさは，文献[5]で用いたベンチマークと同一の NX × NY × NZ＝512 × 512 × 512 に固定した．

各プロセスのデータ分割は，MPI プロセス数に依存するが，X 軸，Y 軸，および Z 軸の 3 次元に分割する．

3.5 想定する AT 方式

本実験で仮定する AT の実行形式は，実行起動前時 AT である．したがって，ユーザが問題サイズを確定したときに AT を実行し，その結果を用いて何回も対象となるアプリケーションの実行をする形態を想定する．

具体的には，実行前に一度 AT によるカーネル測定を行い，最適な実装情報を得る．その後，最適化した実装情報を取得し，最適化した実装のみで本計算を行う．

ここでは，最適化した実装で本計算を行ったときの実行時間を「AT を行った場合の実行時間」とする．つまり，AT 実行時間は含んでいない．

3.6 その他の設定について

AT 起動時の各対象の計測回数，および，ppOpen-APPL/FDM での時間ステップ回数を以下とする．

- AT のための演算カーネルの反復回数: 100 回
- 時間ステップの数：2000 時間ステップ

4. 性能評価

4.1 対象計算機の構成

ppOpen-APPL/FDM ver.1.0.0 のコードを利用し，スカラ向き演算ルーチンを追加した．また，ppOpen-AT ver.1.0.0 に本 AT の機能は実装されている．

以下の計算機を利用した．

1. Fujitsu PRIMEHPC FX10 (FX10)
   - 東京大学情報基盤センター設置
   - CPU：SPARC64 IXfx，1.848 GHz，16 コア
   - 記憶容量：32 GB
   - 理論ピーク性能(ノード)：236.5 GFLOPS
   - キャッシュ構成
     - L1:32KB（命令／データ分離，コア毎），L2:12MB（共有）
     - 2 ウェイ
   - 富士通 MPI
   - コンパイラ：富士通 Fotran90 コンパイラ version 1.2.1 P-id: T01641-04 (Jul 10 2014 14:29:18)
   - コンパイラオプション：-O3 -Kopenmp
   - メモリアクセス性能（node あたり）：85 GB/秒
   - Stream 性能(Triad)： 64.7 GB/秒 [28]
2. 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：-O3 -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 [29]

ppOpen-APPL/FDM は単精度演算である．FX10 と FX100 との演算性能とメモリアクセス性能（Triad）の比率は，以下になる．

- 演算性能比：9.5x
- メモリアクセス性能比：4.9x

4.2 ハイブリッド MPI/OpenMP の表記法

対象となるハイブリッド MPI/OpenMP 実行について，以下の表記で記載する．

- 表記の P*X* T*Y*は，*X* MPI プロセスと，プロセス当たり *Y* スレッドでの実行を意味する．

8 ノード実行に固定すると，以下の組み合わせがある．

- FX10
  - P8T16, P16T8, P32T4, P64T2, P128T1(ピュア MPI)
- FX100
  - P8T32, P16T16, P32T8, P64T4, P128T2, P256T1(ピュア MPI)

4.3 階層型 AT 処理の AT 結果

3.3.3 節で説明した階層型 AT 処理の結果は，FX10 と FX100 ともに，(3)スカラ計算機向きコード(IF-free)が選択された．

4.4 FX100 における NUMA の影響

FX100 は，16 コアを 1 ソケットとし，2 ソケットを 1 ノードとする NUMA 構成となっている．そのため，MPI プロセスのメモリ確保と物理コアの割り当て（NUMA affinity）の影響を受ける．また，FX100 用の専用の NUMA affinity の設定方法が提供されている．そこで，NUMA affinity を設定した場合(NUMA affinity)と設定しない場合(Default)で性能差を調べた．図 3 に，NUMA affinity を設定した場合の全体時間に対する，Default の速度向上率を示す．

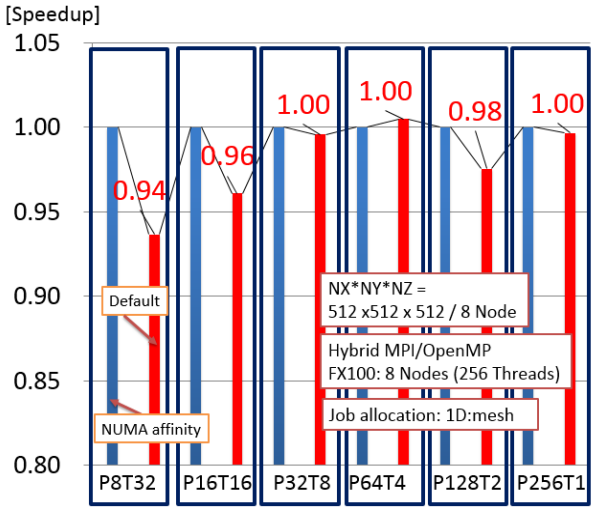


図 3　NUMA affinity 設定の影響（全体時間）

図 3 では，P8T32 と P16T16 のときに NUMA affinity 設定の効果が大きい．これは，FX100 では 1 ノード 2 ソケットのため，1 ソケットあたり 1MPI プロセスの割り当てを明示的に行うほうが効果的であるからと考えられる．

4.5 全体時間の比較

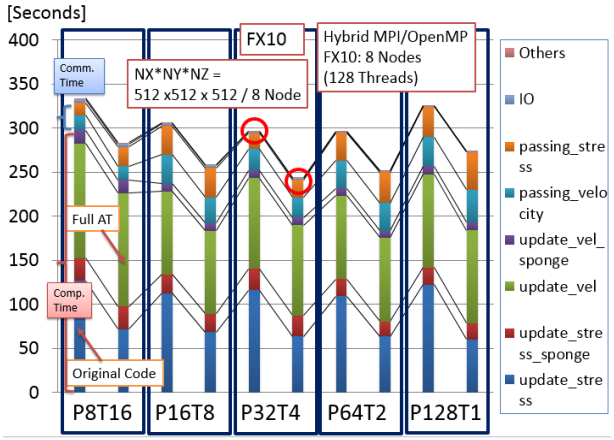
図 4 に FX10 と FX100 において，全体の実行時間（2000 時間ステップ）における AT の効果を載せる．なお FX100 においては，NUMA affinity を指定し，ノード割り当てを連続（mesh で 1 次元割り当て）に固定した．

図 4(a)では，FX10 ではハイブリッド MPI/OpenMP の実行の形態に依存せず実行時間が変動していないが，図 4(b)から FX100 では，ハイブリッド MPI/OpenMP の実行形態により実行時間が大きな影響を受ける．特に P8T32 は，FX100 の NUMA 構成実行で，1 ノードあたり 1MPI プロセス実行となる．そのため，NUMA 構成のメモリを跨いだアクセスが生じる．このことから，他の実行形態に対して実行時間が増加したと推察される．

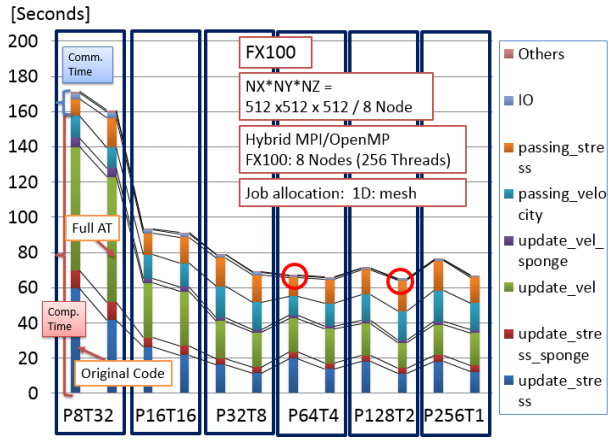
図 4(a)から，FX10 では，AT ある／なし双方とも，最速となる実行形態は P32T4 である．一方 FX100 では，図 4(b)から，AT なしでは P64T4 が最速であるが，AT ありでは P128T2 が最速となった．

4.6 速度向上率の比較

図 5 に，FX10 の実行時間を 1 としたときの FX100 の速度向上率を示す．図 5(a)から，FX100 に移行したときの速度向上率は，全体時間で AT なしの実行時間で 4.42 倍，AT ありの実行時間で 3.74 倍である．また図 5 (b)から，演算カーネル時間のみの速度向上率は，AT なしの実行時間で 5.24 倍，AT ありの実行時間で 6.58 倍である．

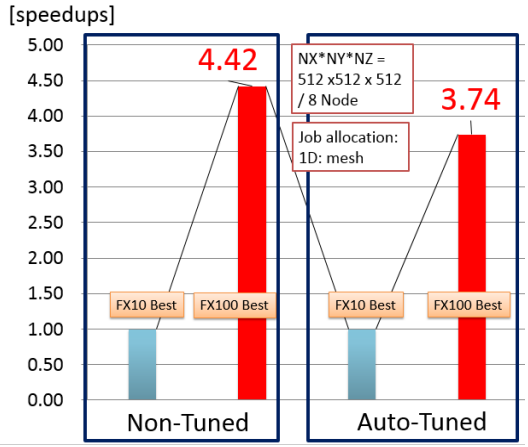


(a) FX10 の全体時間

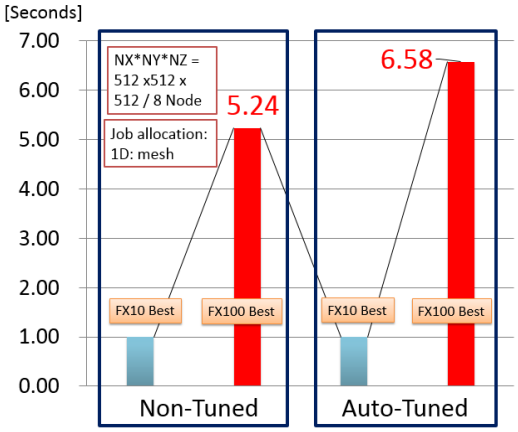


(b) FX100 の全体時間

図 4　全体の実行時間（2000 時間ステップ）における AT の効果



(a) FX10 と FX100 の速度向上率（全体時間）



(b) FX10 と FX100 の速度向上
（演算カーネルのみ）

図 5　FX10 の実行時間を 1 としたときの
FX100 の速度向上率

4.7 プロファイル結果

FX10 と FX100 での性能差の原因を見るため，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 2 に，AT ありの場合における FX10 と FX100 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 2 から，実行時間に占める割合において，FX10 での浮動小数点ロードキャッシュアクセス待ち時間の占める割合が 37.18%～39.31%と多くを占めるが，FX100 では、浮動小数点ロード L2 アクセス待ち時間の占める割合が 13.14%～14.61%と削減されている．

4.8 考察

4.8.1 ハード性能比からみた速度向上

図 5(a)(b)から，FX100 に移行したときの速度向上率は，全体時間で 3.74 倍～4.42 倍，演算カーネル時間のみの速度向上率は，5.24 倍～6.58 倍であった．

ppOpen-APPL/FDM は陽解法コードであり，基本的にはメモリバウンドな演算であること，および，FX10 と FX100 の Stream 性能比である 4.9 倍を考慮すると，全体時間の速度向上率はハード性能比の観点で妥当である．

一方，演算カーネルのみの速度向上は，ハードウェア性能比以上の速度向上率を達成している．特に，AT 済みのコードに対する速度向上率が 6.58 倍と大きい．これは演算カーネルで最も時間を占める update_stress カーネルについて，AT で選択されるコードである(3)スカラ計算機向きコード(IF-free)の B/F 値が 0.4 であるが，ベースラインのオリジナルコードの B/F 値である 1.7 よりも，とても小さいことが挙げられる．そのため，FX10 と FX100 との演算性能比である 9.5 倍に近づく速度向上が達成できたと考えられる．

4.8.2 性能プロファイル結果からの考察

表 2(a)(b)より，実行時間に占める割合において，FX10 での浮動小数点ロードキャッシュアクセス待ち時間の占める割合が 37.18%～39.31%から，FX100 では 13.14%～14.61%と削減されている．このことから，FX100 では，L2 キャッシュ容量の増加によるキャッシュヒット率の向上により，結果としてメモリアクセス性能が向上したと推察される．そのため FX100 では，データアクセス時間が FX10 に対し短縮され，結果として，FX100 での高速化に貢献したと考えている．実際，性能プロファイル結果によると，FX10 でのコア当たりの L2 スループットは 3.29～3.51GB/秒であったが，FX100 では 10.84～12.68GB/秒と，最大で 3.8 倍ほど高性能化していた．このことから，本事例は 3 次元積層メモリ技術によるアクセス性能向上が原因でないとしても，メモリアクセス性能が相対的に向上した場合に生じる，性能的な問題を示唆していると解釈できる．

一方，表 2(b)より FX100 では，バリア同期待ち時間の全体に占める割合が 27.93%と大きくなっている．なぜバリア同期待ちが増加しているかは不明であるが，1 つの理由として，各コアが担当する演算負荷が均一でないことが推定される．そのため，動的負荷分散技術により速度向上が達成できる可能性がある．

FX100 のバリア同期待ち時間増加の原因を追究する必要があるが，データアクセス時間が短縮すると，演算時間の占める割合が増加し，結果として負荷バランスの不均衡を生み，バリア同期待ち時間が増加した可能性がある．この仮定が正しい場合，3 次元積層メモリ技術などの導入で FLOPS から BYTES へ技術トレンドが移る場合，従来とは異なる観点でコード最適化を考えていく必要がある．

5. おわりに

本研究は，将来来るべき 3 次元積層メモリ技術などによるデータアクセス性能向上率が演算性能向上率に対して相対的に高くなることを想定した FLOPS から BYTES への性能転換が実現される場合に生じるであろう，数値計算アルゴリズムと実装技術の変革の可能性を検証することにある．

従来型のメモリを持つ FX10 と，3 次元積層技術によるメモリをもつ FX100 で性能を比べたところ，ハードウェアによるデータ転送能力に対して妥当な速度向上を得た．また，プロファイラによる実行時間の内訳を解析したところ，浮動小数点演算のキャッシュからのデータアクセス時間の割合が FX100 において，FX10 よりも大幅に減少していた．そのため，3 次元積層技術の導入による効果ではないとしても，将来的にデータアクセス性能が相対的に向上する場合に生じる実行性能の問題を模倣していると推察された．

この一方で，FX100 では FX10 に対しバリア同期時間が増加していた．この 1 つの理由として，データアクセス時間が減少したことにより，演算負荷のばらつきが顕著になったことが考えられる．この仮定が正しい場合，3 次元積層技術の導入により，従来とは異なる方針のコード最適化（および AT）をしなければならないことを示唆しているが，原因については今後も性能分析を継続していく．

ppOpen-APPL/FDM に ppOpen-AT による d-spline 法を用いた逐次追加型の性能モデルによる AT 方式[30][31]を追加し，AT 適用の局面を広げることは，重要な今後の課題の 1 つである．

謝辞

本機能の開発に関し，ppOpen-APPL/FDM の利用でご協力頂いた東京大学の古村孝志教授に感謝します．

本研究の一部は，科学技術研究費補助金，基盤研究(B)，「複合的・階層的な自動チューニングを実現する数理基盤手法の研究とライブラリの開発」（課題番号：15H02708），および，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）の支援による．

参考文献

1) S. Matsuoka, H. Amano, K. Nakajima, K. Inoue, T. Kudoh, N. Maruyama, K. Taura, T. Iwashita, T. Katagiri, T. Hanawa, T. Endo, “From FLOPS to BYTES: Disruptive Change in High-performance Computing Towards the Post-moore Era, Proceedings of the ACM International Conference on Computing Frontiers (CF’16), pp.274-281 (2016)
2) T. Katagiri, S. Ito, S. Ohshima, “Early experiences for adaptation of auto-tuning by ppOpen-AT to an explicit method,” Special Session: Auto-Tuning for Multicore and GPU (ATMG) (In Conjunction with the IEEE MCSoC-13), Proceedings of MCSoC-13, pp.153-158 (2013)
3) T. Katagiri, S. Ohshima, M. Matsumoto, “Auto-tuning of computation kernels from an FDM Code with ppOpen-AT,” Special Session: Auto-Tuning for Multicore and GPU (ATMG) (In Conjunction with the IEEE MCSoC-14), Proceedings of MCSoC-14, pp.253-260 (2014)
4) T. Katagiri, S. Ohshima, M. Matsumoto, “Directive-based Auto-tuning for the Finite Difference Method on the Xeon Phi,” International Workshop on Automatic Performance Tuning (iWAPT2015), Proceedings of IPDPSW2015, pp.1221-1230 (2015)
5) T. Katagiri, M. Matsumoto, S. Ohshima, “Auto-tuning of Hybrid MPI/OpenMP Execution with Code Selection by ppOpen-AT,” International Workshop on Automatic Performance Tuning (iWAPT2016), Proceedings of IPDPSW2016, (2016)
6) M. Frigo, S.G. Johnson, “FFTW: An adaptive software architecture for the FFT,” in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 3, IEEE Press, Los Alamitos, CA, pp. 1381–1384 (1998)
7) R.C. Whaley, A. Petitet, J.J. Dongarra, “Automated empirical optimizations of software and the ATLAS project,” Parallel Computing, Vol. 27, Issue 1-2, pp. 3-35, (2001)
8) J.Cámara, J. Cuenca, D. Giménez, L. P. García, A. M. Vidal, “Empirical Installation of Linear Algebra Shared-Memory Subroutines for Auto-Tuning”, International Journal of Parallel Programming, Vol. 42, Issue 3, pp. 408-434 (2013)
9) H. Kuroda, T. Katagiri, M. Kudoh, Y. Kanada, “ILIB_GMRES: An auto-tuning parallel iterative solver for linear equations,” SC2001 (2001) (A Poster.)
10) E-J. Im, K. Yelick, R. Vuduc, “SPARSITY: Optimization framework for sparse matrix kernels,” International Journal of High Performance Computing Applications (IJHPCA), Vol. 18, No. 1, pp. 135–158 (2004)
11) R. Vuduc, J.W. Demmel, K.A. Yelick, “OSKI: A library of automatically tuned sparse matrix kernels,” In Proceedings of SciDAC, Journal of Physics: Conference Series, Vol. 16, pp. 521–530 (2005)
12) T. Katagiri, K. Kise, H. Honda, T. Yuba, “ABCLib_DRSSED: A parallel eigensolver with an auto-tuning facility,” Parallel Computing, Vol. 32, Issue 3, pp. 231–250 (2006)
13) T. Sakurai, T. Katagiri, H. Kuroda, K. Naono, M. Igai, S. Ohshima, “Sparse Matrix Library with Automatic Selection of Iterative Solvers and Preconditioners,” Proceedings of the International Conference on Computational Science, ICCS 2013,Vol. 18, pp.1332‑1341 (2013)
14) T. Katagiri, P.-Y. Aquilanti, and S. Petiton, “A Smart Tuning Strategy for Restart Frequency of GMRES(m) with Hierarchical Cache Sizes,” Selected Papers of 10th International Meeting on High-Performance Computing for Computational Science (VECPAR'2012) , Springer Lecture Notes in Computer Science, Vol. 7851, pp.314-328 (2013)
15) T. Katagiri, K. Kise, H. Honda, T. Yuba, “ABCLibScript: A directive to support specification of an auto-tuning facility for numerical software,” Parallel Computing, Vol. 32, Issue 1, pp. 92–112 (2006)
16) Q. Yi, K. Seymour, H. You, R. Vuduc, D. Quinlan, “POET: Parameterized optimizations for empirical tuning,” Proceedings of Parallel and Distributed Processing Symposium (IPDPS 2007), pp. 1–8 (2007)
17) S. Donadio, J. Brodman, T. Roeder, K. Yotov, D. Barthou, A. Cohen, M.J. Garzar´an, D. Padua, K. Pingali, “A language for the compact representation of multiple program versions,” Proceedings of Languages and Compilers for Parallel Computers (LCPC’05), Lecture Notes in Computer Science 4339, pp.136–151 (2007)
18) C. Chen, J. Chame, M. Hall, J. Shin, G. Rudy, “Loop transformation recipes for code generation and auto-tuning,”

Proceedings of the 22nd International Workshop on Languages and Compilers for Parallel Computing (2009)

19) S. Ramalingam, M. Hall, C. Chen, “Improving high-performance sparse libraries using compiler-assisted specialization: A PETSc case study,” Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 487–496 (2012)
20) C. Schaefer, V. Pankratius, W.F. Tichy, “Atune-IL: An instrumentation language for auto-tuning parallel applications,” Proceedings of the 15th International Euro-Par Conference on Parallel Processing (Euro-Par09), pp. 9–20 (2009)
21) S. Benkner, S. Pllana, J.L. Traff, P. Tsigas, U. Dolinsky, C. Augonnet, B. Bachmayer, C. Kessler, D. Moloney, V. Osipov, “PEPPHER: Efficient and productive usage of hybrid computing systems,” IEEE Micro Vol. 31, No. 5 pp. 28–41 (2011)
22) H. Takizawa, S. Hirasawa, Y. Hayashi, R. Egawa, H. Kobayashi, “Xevolver: An XML-based Code Translation Framework for Supporting HPC Application Migration,” Proceedings of IEEE International Conference on High Performance Computing (HiPC) (2014)
23) M.J. Voss, R. Eigenmann, “High-level adaptive program optimization with ADAPT,”, ACM SIGPLAN Notices 2001, pp. 93–102 (2001)
24) T. Katagiri, K. Kise, H. Honda, T. Yuba, “FIBER: A general framework for auto-tuning software,” The Fifth International Symposium on High Performance Computing (ISHPC-V), Springer LNCS 2858, pp. 146–159 (2003)
25) K. Nakajima, M. Satoh, T. Furumura, H. Okuda, T. Iwashita, H. Sakaguchi, T. Katagiri, M. Matsumoto, S. Ohshima, H. Jitsumoto, T. Arakawa, F. Mori, T. Kitayama, A. Ida and M. Y. Matsuo, “ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT),” Optimization in the Real World, Mathematics for Industry 13, K. Fujisawa et al. (eds.), Springer, pp.15-35 (2016)
26) 片桐孝洋，松本正晴，大島聡史：“SCG-AT：静的コード生成のみによる自動チューニング実現方式”，Vol. 2015-HPC-150，No. 32 (2015)
27) T. Furumura, L. Chen, “Parallel simulation of strong ground motions during recent and historical damaging earthquakes in Tokyo, Japan,” Parallel Computing, Vol. 31, pp. 149–165 (2005)
28) 大島聡史， 實本英之， 鴨志田良和， 片桐孝洋， 田浦健次朗， 中島 研吾: “大規模超並列スーパーコンピューターシステム Oakleaf-FX(Fujitsu PRIMEHPC FX10)の性能評価”, 情報処理学会研究報告， Vol. 2012-HPC-135, No. 43 (2012)
29) 千葉修一, “PRIMEHPC FX100 の特徴と概要” (2015) http://accc.riken.jp/wp-content/uploads/2015/06/chiba.pdf
30) T. Tanaka, R. Otsuka, A. Fujii, T. Katagiri, T. Imamura, “Implementation of d-Spline-based incremental performance parameter estimation method with ppOpen-AT,” Scientific Programming, IOS Press, Vol. 22, No. 4, pp. 299-307 (2014)
31) R. Murata, J. Irie, A. Fujii, T. Tanaka, T. Katagiri, “Enhancement of Incremental Performance Parameter Estimation on ppOpen-AT,” Special Session: Auto-Tuning for Multicore and GPU (ATMG-15), Proceedings of IEEE MCSoC2015, pp.203-210 (2015)

表 2 性能プロファイル結果（AT あり）

(a) FX10 (P32T4)

時間グラフ内訳割合 (%)	整数ロード・メモリアクセス待ち	浮動小数点ロード・メモリアクセス待ち	ストア待ち	整数ロード・キャッシュアクセス待ち	浮動小数点ロード・キャッシュアクセス待ち	整数演算待ち	浮動小数点演算待ち	分岐命令待ち	命令フェッチ待ち	バリア同期待ち	UOPコミット	その他の待ち	1命令コミット	整数レジスタ書き込み制限り	2/3命令コミット	4命令コミット	合計
Thread 0	0.32%	13.78%	6.71%	1.53%	39.31%	0.80%	3.52%	0.60%	2.43%	0.77%	6.71%	−9.30%	8.87%	2.48%	9.73%	11.70%	100.00%
Thread 1	0.12%	8.95%	6.28%	0.28%	37.55%	0.52%	3.57%	0.10%	1.29%	14.50%	0.98%	−0.99%	6.45%	0.88%	8.43%	11.10%	100.00%
Thread 2	0.12%	8.96%	6.27%	0.28%	37.18%	0.51%	3.56%	0.10%	1.29%	14.87%	0.98%	−0.85%	6.39%	0.87%	8.37%	11.09%	100.00%
Thread 3	0.14%	9.09%	6.26%	0.27%	37.27%	0.51%	3.57%	0.10%	1.29%	14.46%	0.98%	−0.66%	6.39%	0.87%	8.37%	11.09%	100.00%

(b) FX100 (P128T2)

時間グラフ内訳割合 (%)	メモリ・キャッシュヒンジー待ち	整数ロード・メモリアクセス待ち	浮動小数点ロード・メモリアクセス待ち	ストア待ち	整数ロード・L2アクセス待ち	整数ロード・L1Dアクセス待ち	浮動小数点ロード・L2アクセス待ち	浮動小数点ロード・L1Dアクセス待ち	整数演算待ち	浮動小数点演算待ち	分岐命令待ち	命令フェッチ待ち	バリア同期待ち	その他の待ち	1命令コミット	2/3命令コミット	4命令コミット	合計
Thread 0	10.01%	3.26%	19.42%	5.79%	7.26%	2.41%	14.67%	5.36%	0.60%	6.21%	0.49%	4.97%	0.34%	−6.52%	8.86%	8.03%	8.86%	100.00%
Thread 1	8.68%	1.40%	12.99%	4.15%	2.28%	1.46%	13.14%	4.47%	0.36%	6.02%	0.07%	0.92%	27.93%	−0.19%	3.75%	4.26%	8.32%	100.00%