Vol.2015-HPC-150 No.32
2015/8/6

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

片桐 孝洋†, 松本 正晴†, 大島 聡史†

本報告では，自動チューニング（AT）を実行するに当たり，コード最適化時に動的なコード生成とコンパイルを行わず，実行前に静的に生成したコードのみを利用する AT ソフトウェア構成方式の Static Code Generation Auto-tuning (SCG-AT)を提案する．SCG-AT による AT を評価するにあたり「階層型 AT 処理」を実装した．差分法による地震波シミュレーション ppOpen-APPL/FDM において，従来のベクトル計算機向けコードと新規開発したスカラ計算機向けコードのコード選択処理を実装した．Xeon Phi，Ivy Bridge，および FX10 の 3 種の全く異なる計算機で SSG-AT によるコード選択の AT を評価した．評価の結果，Xeon Phi と Ivy Bridge においてはスカラ計算機向けコードの選択により，従来行われていた AT 方式では達成できない速度向上が達成できることを明らかにした．

1. はじめに

本報告は，静的コード生成のみによる自動チューニング(Auto-tuning, AT)実現方式である Static Code Generation Auto-tuning (SCG-AT)の提案を行う．

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，本報告で利用する自動チューニング（Auto-tuning, AT）専用言語である ppOpen-AT[1]-[3]を説明する．また，SCG-AT を実現するにあたり，「階層型 AT」の処理を ppOpen-AT に実装する場合の既存方法における問題について説明する．4 章は，現在普及しているメニーコア CPU の Intel Xeon Phi，マルチコア CPU の Ivy Bridge，および富士通 PRIMEHPC FX10 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. 関連研究とオリジナリティ

2.1 関連研究

AT 研究は，数値計算処理の特定アプリケーションを対象にした方式[4]-[12]から，計算機システムやプログラムを対象にした汎用方式[13]-[24]まで，幅広いアプリケーションと計算機環境において研究がされている．現在まで継続的に研究開発がされているアクティブな研究分野の 1 つである．本報告では，汎用 AT 方式の手法に関連する．

汎用 AT 方式のフレームワークでは，主に実行時情報の取得を基にした最適化がなされている. AT を適用する対象としては, OS レベルを対象とする AT 研究がある[13]-[15]. これら OS レベルを対象とする AT の汎用性は高いが, 一方で OS カーネルの改変や AT 専用のデーモンの起動が必要となるため, AT を適用できる計算機環境が限定される．たとえば，スパコンセンターにおける運用中のスパコンに OS レベルの AT を適用することは，保守契約の都合から困難である．実際，運用レベルのスパコンに適用され，ユーザに公開されて運用されている OS レベルの AT の実例は, 著者の知る限り存在していない．

汎用な AT 方式のフレームワークでは，実行時に判明するループ長などの有用な情報を入手して最適化を行う．そのため，コンパイラレベルを対象とする AT 研究が多数提案されている[16]-[24]．このコンパイラレベルを対象とする AT 研究はコンパイラのコード最適化技術を基にしており，伝統的に行われてきたコンパイラコード最適化技術[25]-[29]を活用している．

コンパイラのコード最適化技術と AT 技術の違いについて述べたい．コンパイラ最適化の研究は，特定の計算機環境で効果がある方法や性能評価をすることが多い．一方，AT 研究では複数の計算機環境で効果がある方式を目的にしている．AT 技術の目的は，単一コードでも複数の計算機環境で高性能を実現する**性能可搬性**（**Performance Portability**）の実現にある．

AT 技術は実コードと実計算機システムに対する機能開発が目的になることが多い．そのため，リサーチコンパイラのみ開発できればよいわけではなく，現在使われているコンパイラへの適用を強く指向しなくてはならない．この点においても，伝統的なコンパイラのコード最適化研究との違いがあると著者は考えている．

2.2 本提案のオリジナリティ

2.2.1 従来法と SCG-AT

本提案は，コンパイラレベルを対象とする AT に属するが，以下の観点で異なるアプローチである．

図 1 に従来の AT 方式の流れ図を示す．図 2 に，提案する AT 方式の流れ図を示す。

図 1 の従来法のように，動的にコード生成とコンパイルを繰り返す方式による AT ソフトウェアの構成方式を **Dynamic Code Generation Auto-tuning (DCG-AT)**とよぶ．一方，図 2 に示す静的コード生成方式のみによる AT ソフトウェアの構成方式を **Static Code Generation Auto-tuning (SCG-AT)** と呼ぶ．

† 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

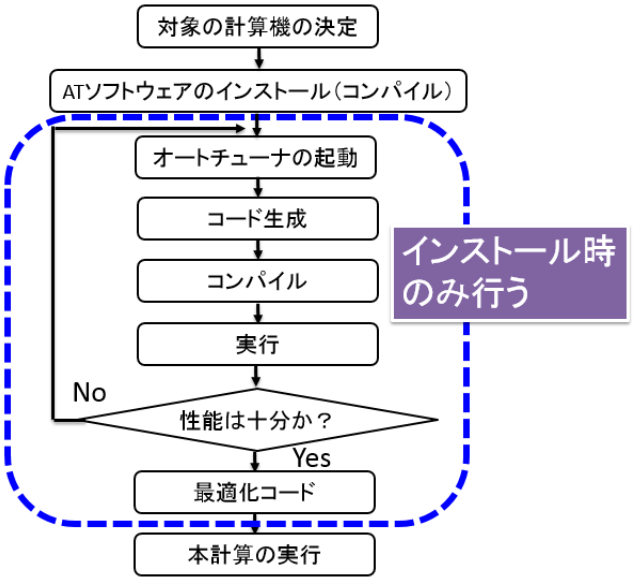


図 1　従来の AT 方式の流れ図

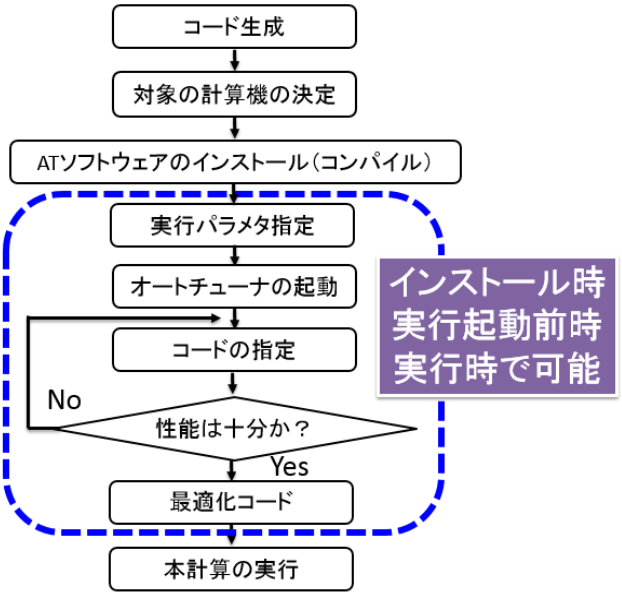


図 2　提案する AT 方式の流れ図

2.2.2 SCG-AT の利点と欠点

DCG-AT に対する SCG-AT の利点と欠点をまとめる．

- 利点
  - コードの再コンパイルが不要．そのため：
    1. （コンパイルに費やす）AT のための時間が削減される．
    2. コード自動生成後のコンパイラ起動がない．AT を実行する計算機上（例えば，ログインノード）に余分な負荷を与えない．
  - 実行時のコード自動生成が不要．そのため：
    1. コード自動生成のためのソフトウェアスタック（スクリプト言語など）が不要．
    2. 直前の実行からコード生成を連結する処理が不要．特にバッチジョブシステムを利用する場合は，AT システム内でのバッチジョブスクリプトの書き直しが不要のため利便性が高い．
    3. コード生成を対象計算機と異なる環境で行う場合（たとえばスパコンを使う場合に，コード生成をスパコンセンター外の計算機で行う場合）の処理の自動化で必要なデーモン起動・常駐などが不要．
  - AT で効果的となるアルゴリズム選択（コード選択）の実装が容易．
- 欠点
  - AT 候補のコードをプログラム中に持つため，コード量が増大する．実用的なコード量にするためには，対象処理についての高度な知識が必要．
  - 最適化済み情報を得るための処理（ファイルアクセスなど）が追加されるため，AT のための処理時間（AT のオーバーヘッド）が増える．
  - AT 候補を選択する処理の挿入により，オリジナルコードの実行に対し，キャッシュなどの振る舞いが異なることがある．そのため，AT 効果が十分に得られないことがある．

2.2.3 階層型 AT とコード選択の観点でのオリジナリティ

SCG-AT の概念は，著者らが開発した FIBER 方式[30]で提案されている．しかし既発表論文[30]では，SCG-AT の長所について明確に主張がなされていない．そこで本稿において，明確な主張を行う．

すでに説明したように SCG-AT では複数のコード選択をする AT について，対象となる AT コード（選択対象となるコード）を明示的に所有する．そのため，AT が実装しやすく，かつ AT 実行を行い易い．このような複数のコードを所持して動的にコード選択をする AT 手法は，たとえば論文[29]で提案されているが，いずれも DCG-AT の枠組みである．SCG-AT によるコード選択の実現はチャレンジングな課題であり，かつ，オリジナリティを有する．

本研究では，SCG-AT に基づく以下の実装評価を行う点においてオリジナリティがある：

- 階層型 AT 処理の適用とその実装方法の検討
- 階層型 AT 処理の性能評価

3. ppOpen-AT における実装

3.1 既存研究における位置づけ

著者らは既存研究において，SCG-AT を Finite Difference Method (FDM)の実アプリケーションに適用し，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[1]-[3]．ここでは，これらの既存研究において実装されていない「**階層型 AT 処理**」に対して，SCG-AT を適用する場合の問題点を検討する．

3.2 階層型 AT 処理

階層型 AT 処理とは，少なくとも，ある箇所に AT が記述されている場合に，その箇所外部から呼び出す場所にも AT が記述されている場合をいう．

例えば，あるループに AT が記述されており，そのループ内からコールされる手続き内にも AT が記述されているような場合である．図 3 にその例を示す．

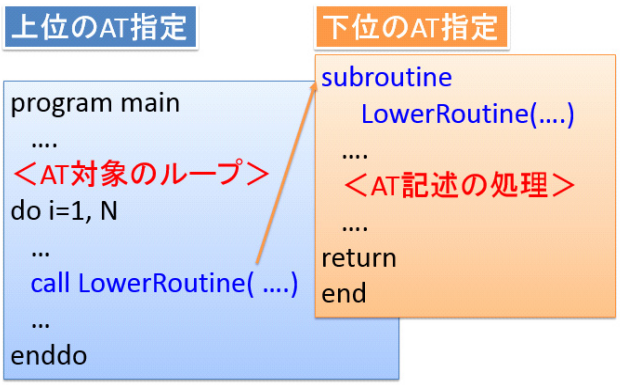


図 3　階層型 AT 処理の例（2 階層）

図 3 では，AT を行うにあたり以下の任意性がある：

1. ①下位の AT を行う．②下位の AT で最適化したパラメタを利用した上で，上位の AT を行う．
2. ①（下位の AT のパラメタを固定した上で）上位の AT を行う．②上位の AT で最適化したパラメタを利用した上で，下位の AT を行う．
3. その他の方法．たとえば，上位，下位のパラメタをそれぞれ固定した上で，同時に AT を行う．

通常どのように AT を行うかは，対象となる処理のデータ依存関係から判断できる．そのため，ユーザ知識を利用し，ユーザが AT 実行の順番を指定できる仕組みが必要である．

3.3 ppOpen-APPL/FDM における階層型 AT 処理

3.3.1 処理の流れ図

ここでは階層型 AT 処理の実例として，ppOpen-HPC[31]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 を構成するためのライブラリ ppOpen-APPL/FDM に対して，AT 適用を行った例を取り上げる．ppOpen-APPL/FDM の処理の流れ図を図 4 に示す．

図 4 では陽解法を用いているが，演算を行う箇所（演算カーネル）が分散されて配置されている．そのため，AT 対象である演算カーネルは複数存在する．

3.3.2 ベクトル向き実装とスカラ計算機向き実装

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 実装に対して，スカラ計算機向きコード（Intel 向きコード）を開発した．対象の計算機環境に応じて，適するコード選択を行う AT の実装について説明する．まず図 5 に，主要な演算カーネルの 1 つである update_vel カーネルの従来実装を示す．

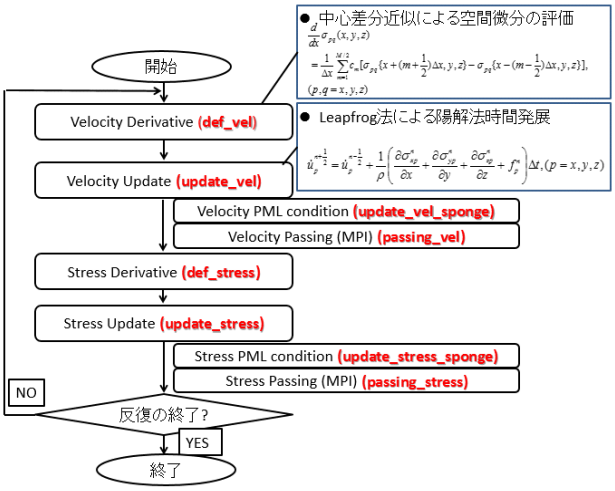


図 4　ppOpen-APPL/FDM の処理の流れ図

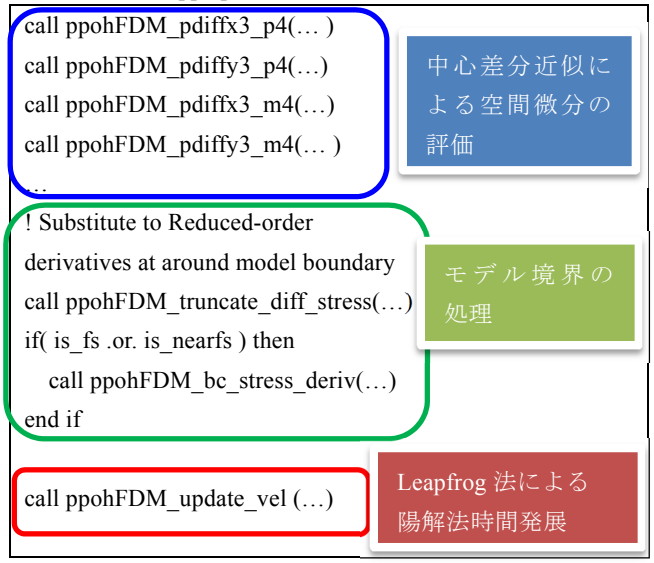


図 5　update_vel カーネルの従来実装
（ベクトル計算機向き）

図 5 では，コードの構成が（1）中心差分近似による空間微分の評価，（2）モデル境界の処理，（3）Leapfrog 法による陽解法時間発展，の 3 つに分かれている．そのため，それぞれの部分ごとに対応した手続き群がコールされる．特に（1）の中心差分近似による空間微分の評価では，3 次元空間に対して 3 変数の差分近似があるため，合計 9 個の手続きがある．そのため，データの読出し／書き戻し回数が多い実装になっている．結果として，手続き全体での Byte per FLOPS (B/F) 値を大きくする要因となっている．

一方，図 5 と同様の処理を行う場合において，（3）Leapfrog 法による陽解法時間発展ループ内に，（1）中心差分近似による空間微分の評価，（2）モデル境界の処理，を入れ込む実装も考えられる．この実装では，図 5 のコードに対して，データの読出し／書き込み量を減らすことができるため，スカラ計算機向きになる可能性がある．update_vel カーネルについてこの実装を行ったプログラムの概要を図 6 に示す．

```
!$omp parallel do private (i,j,k,RL1,RM1,RM2,RLRM2,…)
do k_j=1, (NZ01-NZ00+1)*(NY01-NY00+1)
  k=(k_j-1)/(NY01-NY00+1)+NZ00
  j=mod((k_j-1),(NY01-NY00+1))+NY00
  do i = NX00, NX01
    RL1   = LAM (I,J,K);  RM1   = RIG (I,J,K)
    RM2   = RM1 + RM1;  RLRM2 = RL1+RM2
! 4th order diff
    DXVX0 = (VX(I,J,K)  -VX(I-1,J,K))*C40/dx &
              - (VX(I+1,J,K)-VX(I-2,J,K))*C41/dx
    DYVY0 = (VY(I,J,K)  -VY(I,J-1,K))*C40/dy &
              - (VY(I,J+1,K)-VY(I,J-2,K))*C41/dy
    DZVZ0 = (VZ(I,J,K)  -VZ(I,J,K-1))*C40/dz &
              - (VZ(I,J,K+1)-VZ(I,J,K-2))*C41/dz
! truncate_diff_vel
    ! X dir
    if (idx==0) then
      if (i==1)then
        DXVX0 = ( VX(1,J,K) - 0.0_PN    )/ DX
      end if
      if (i==2) then
        DXVX0 = ( VX(2,J,K) - VX(1,J,K) )/ DX
      end if
    end if
    if( idx == IP-1 ) then
      if (i==NXP)then
        DXVX0 = ( VX(NXP,J,K) - VX(NXP-1,J,K) ) /  DX
      end if
    end if
    ! Y dir
    if( idy == 0 ) then ! Shallowmost
      if (j==1)then
        DYVY0 = ( VY(I,1,K) - 0.0_PN    )/ DY
      end if
      if (j==2)then
        DYVY0 = ( VY(I,2,K) - VY(I,1,K) ) / DY
      end if
    end if
    if( idy == JP-1 ) then
      if (j==NYP)then
        DYVY0 = ( VY(I,NYP,K) - VY(I,NYP-1,K) )/ DY
      end if
    end if
    ! Z dir
    if( idz == 0 ) then ! Shallowmost
      if (k==1)then
        DZVZ0 = ( VZ(I,J,1) - 0.0_PN    ) / DZ
      end if
      if (k==2)then
        DZVZ0 = ( VZ(I,J,2) - VZ(I,J,1) ) / DZ
      end if
    end if
    if( idz == KP-1 ) then
      if (k==NZP)then
        DZVZ0 = ( VZ(I,J,NZP) - VZ(I,J,NZP-1) )/ DZ
      end if
    end if

    DXVX1 = DXVX0;  DYVY1 = DYVY0
    DZVZ1 = DZVZ0;  D3V3  = DXVX1 + DYVY1 + DZVZ1
    SXX (I,J,K) = SXX (I,J,K) &
     + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
    SYY (I,J,K) = SYY (I,J,K) &
     + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
    SZZ (I,J,K) = SZZ (I,J,K) &
     + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1) ) * DT
  end do
end do
!$omp end parallel do
```

中心差分近似による空間微分の評価

モデル境界の処理

Leapfrog 法による陽解法時間発展

図 6　update_vel カーネルのスカラ計算機向き実装

図 6 では，（2）モデル境界の処理を実現するため IF 文をループの最内部に実装しなくてはならない．そのため，データプリフェッチなどのコンパイラ最適化を阻害する．結果として，ハードウェアに依存して性能が発揮できなくなる．この観点から，AT による**コード選択**の実装が必要となる．

3.3.3 Select 節を用いたコード自動選択

ppOpen-AT において，3.3.2 節のベクトル計算機向き実装とスカラ計算機向き実装のコード選択を実装する機能として，ABCLibScript から引き次いでいる Select 節を用いた実装ができる．図 7 に，update_vel カーネルでの実例を示す．

図 7 より，下位の AT 実装である ppohFDM_pdiffx3_p4 と ppohFDM_update_vel 中にも AT 指定があるため，階層型 AT 処理になっていることがわかる．図 7 の Select 節を利用した AT 記述に対して，プリプロセッサを通すと図 8 のようなコードが自動生成される．

図 8 では，従来のベクトル向けコードであった一連の処理と，スカラ向けコードであった update_vel_Intel 手続きとを選択する手続き（例えば，update_vel_select）が自動生成され，その手続き内部に選択処理が自動実装される．選択処理だけではなく，選択処理中，およびそれ以外にある手続き内で AT 処理が指定されている箇所については，AT の処理（たとえば，ループ融合やループ分割）に応じた自動生成コード（AT 候補）が自動生成される．

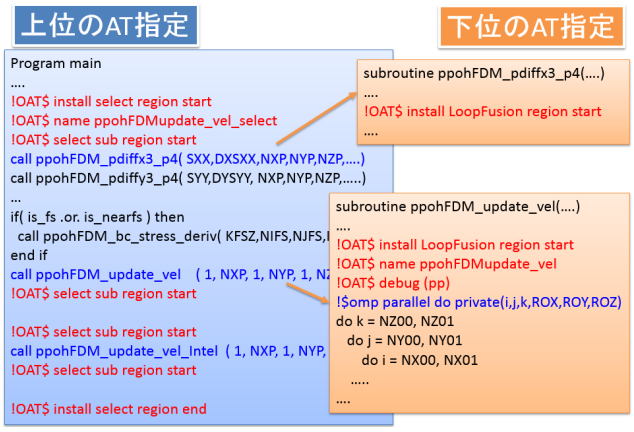


図 7　Select 節でのコード選択の実装.
update_vel_Intel 手続きがスカラ計算機向け実装.

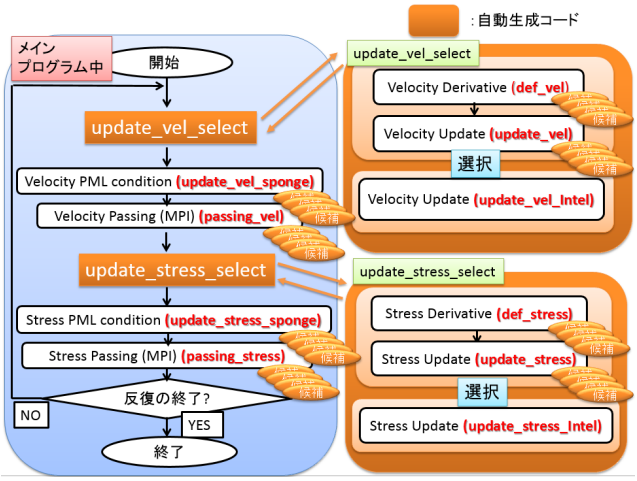


図 8　Select 節でのコード選択を利用した自動生成コードの例（ppOpen-APPL/FDM 全体）.

図 8 のコードでは，どの手続きから AT を実行するか任意性がある．しかし図 4 の流れ図を考慮すると，図 9 の順番で AT 実行すべきであるといえる.

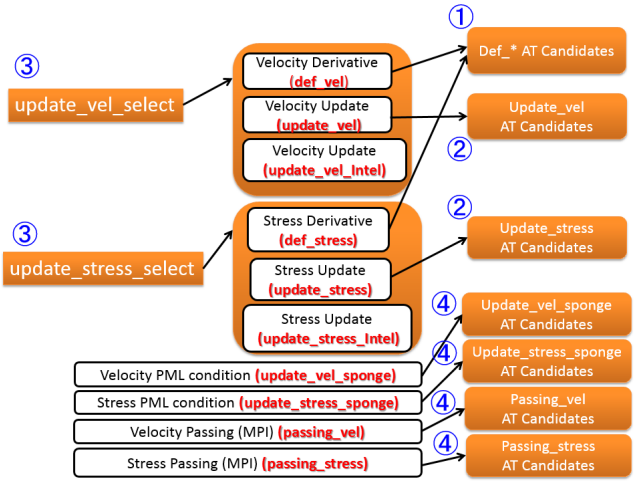


図 9　AT 実行の順序．○内の番号が順番.

図 9 では，まず上位の update_vel_select の AT 実行を考える．このとき利用する手続きのうち，中心差分近似による空間微分の評価に相当する手続き群「def_vel」に AT 候補の手続き群 def_* があるため，まず第 1 位の優先度でこの def_*の AT を行う必要がある．第 2 位の優先度では，update_vel の AT 候補について AT を行う必要がある．この第 1 位，第 2 位の優先度の AT を終了すると，最適パラメタが def_*，および update_vel 手続き内で指定されるため，第 3 位の優先度として，update_vel_select 手続き内のコード選択処理を実行できる．この際，AT により最適化された従来のベクトル計算機向き実装と，スカラ計算機向き実装が比較されるため，妥当なコード選択がなされる.

3.3.4 AT 順序の指定機能拡張

前節の 3.3.3 節で説明したように，階層型 AT 処理を行う際には，対象となる場所の依存関係を考慮した上で，妥当な AT 実行順序が定まる．この際，ppOpen-AT において AT 実行の順番を指定できる機能が無いことが問題となるa.

そこで機能拡張を行い，ユーザ自身が AT 実行の順番を，AT 領域における節で記載できるようにした．そのために，Exec_order 節を新たに設けた．Exec_order 節の記述法は以下である.

- C 言語
  #pragma exec_order <*x*>
- Fortran 言語
  !OAT$ exec_order <*x*>

以下に，記述例を載せる。

- 利用例(Fortran 言語)：

```
subroutine ppohFDM_update_vel(….)
…
!OAT$ install LoopFusion region start
!OAT$ name ppohFDMupdate_vel
!OAT$ exec_order 2
!$omp parallel do private(i,j,k,ROX,ROY,ROZ)
do k = NZ00, NZ01
  do j = NY00, NY01
    do i = NX00, NX01
    …..
```

この Exec_order 節で指定できる順番は「静的な」順番である．現仕様においては，実行時に，動的に AT 呼び出しの順番を決める機能は考慮されていない.

3.3.5 ppOpen-AT における Module の相互参照問題と解決法

ところで，階層型 AT 処理を実装する時，正しい自動生成コードが生成できるかは重要である．図 8 の Select 節を含む階層型 AT 処理のコード自動生成を行うにあたり，ppOpen-AT ver. 0.2 の Fortran 言語版の実装で問題が生じたので，ここで簡単に説明を行う.

本問題は ppOpen-AT の実装方式固有の問題であり，本質的な問題ではない．上位の AT 指定内に含まれる，下位の

a ppOpen-AT でのデフォルトは，プリプロセッサが AT の指定箇所（AT 領域）を見つけた順番で，AT 実行がなされる.

AT 指定がある場合に生じる．つまり，階層的な AT をする場合に生じる．前提として，以下の ppOpen-AT の自動生成コード中の Module 構成，およびファイル構成に起因する．

- ppOpen-AT による自動生成コードのファイル名
  - OAT_Control_Routines.f
    (OAT_Control_Routines モジュール)
    - AT の制御に関する手続き群を含むモジュール
  - OAT_Instrall_Routines.f
    (OAT_Install_Routines モジュール)
    - インストール時最適化を行う AT 候補コードを含むモジュール

AT が指定された場所は，以下のコードが自動生成される：(1) OAT_Control_Routines.f 中にある AT パラメタ設定手続きの OAT_SetParm を呼び出す．(2) その後，対象コードを手続き化した手続きが呼ばれる．(2) の対象コードを手続き化した手続きは，インストール時最適化の場合，OAT_Instrall_Routines.f 中に手続きが自動生成される．

例えば，以下のような AT 指定がある場合：

```
subroutine ppohFDM_update_vel(…)
…
!OAT$ install LoopFusion region start
!OAT$ name ppohFDMupdate_vel
!$omp parallel do private (i,j,k,ROX,ROY,ROZ)
      do k = NZ00, NZ01
        do j = NY00, NY01
          do i = NX00, NX01
            …
!OAT$ install LoopFusion region end
```

以下のようなコードが自動生成される．

```
subroutine ppohFDM_update_vel(…)
…
!!OAT$ install LoopFusion region start
!!OAT$ name ppohFDMupdate_vel
ctmp = "ppohFDMupdate_vel"
! AT パラメタ設定手続き
call OAT_SetParm
   (1, ctmp, NZ01, iusw1_ppohFDMupdate_vel)

!該当コードを含む手続き
call OAT_InstallppohFDMupdate_vel
  (NZ00,…,iusw1_ppohFDMupdate_vel)
…
!!OAT$ install LoopFusion region end
```

このとき，OAT_Control_Routines モジュールでは，オートチューナーの起動のため，モジュール OAT_Install_Routines を use する必要がある．この条件で，以下の 2 通りを考える．

1. **上位の AT 指定内に含まれる下位の AT 指定が無い場合**
   OAT_Install_Routines モジュール内では，自動生成される AT 候補は AT 制御ルーチンを呼ばない．
2. **上位の AT 指定内に含まれる、下位の AT 指定がある場合（階層型 AT 処理の場合）**
   OAT_Install_Routines モジュール内で，先程説明した AT 制御コード呼び出しの理由により，自動生成される AT 候補内で AT パラメタ設定ルーチン（OAT_SetParm）が呼ばれる．そのため，OAT_Control_Routines モジュールの use が必要となる．

以上の 2 では，OAT_Control_Routines モジュール内で“use OAT_Install_Routines”され，かつ，OAT_Install_Routines モジュール内で“use OAT_Control_Routines”されるため，循環依存が生じ，コンパイルできない自動生成コードが出力される．これを，**ppOpen-AT における Module の相互参照問題**とよぶ．

ppOpen-AT における Module の相互参照問題の解決法は，以下の 2 つがある．

i. OAT_Control_Routines.f (OAT_Control_Routines モジュール) と OAT_Instrall_Routines.f (OAT_Install_Routines モジュール)を結合し，1 つのモジュールとする．

ii. OAT_Instrall_Routines.f (OAT_Install_Routines モジュール) 中で必要となる OAT_Control_Routines.f (OAT_Control_Routines モジュール)の手続きを，OAT_Instrall_Routines.f (OAT_Install_Routines モジュール)中にコピーし，依存関係を断ち切る．

ここでは，ii で問題を回避した．理由は，(1)モジュール構成は ppOpen-AT の設計では本質であるので大きな変更をすべきでないこと，(2)自動生成コードであるため人的なコストが少ないこと，である．ただし自動生成コードの量の増大，および冗長手続きの生成による自動生成コードの複雑化といった問題が残る．

4. 予備評価

4.1 問題設定

4.1.1 ベンチマーク問題

本評価で利用する問題領域の大きさは，Seism3D[32]で計算された 2000 年西部鳥取沖地震の大きさ[33]を考慮して決めた．解析領域は，820km × 410km × 128 km で，0.4km 間隔で離散化しており，X 方向，Y 方向，Z 方向の格子数は NX × NY × NZ = 2050 × 1025 × 320 となるため，X 方

向：Y 方向：Z 方向の比率を 6.4 ： 3.2 ： 1 で定めた.

本評価では，利用する計算機のメモリ制約から，NX × NY × NZ = 1536 × 768 × 240 とした.

4.1.2 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を実行する.

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,最適化した実装のみ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.

4.1.3 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に定めた.

- 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

ベクトル計算機向きコードとスカラ計算機向きコードの切り替えは，update_stress カーネル，および update_vel カーネルの 2 カ所に実装している.

4.2 対象計算機の構成

ppOpen-APPL/FDM ver.0.2 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.0.2 による自動生成コードの出力を考慮し，機能拡張部分のコードを手動で追加した.

以下の 3 つの計算機環境（全て 8 ノード）を利用した.

1. Intel Xeon Phi コプロセッサ クラスタ
   - CPU：Xeon Phi 5110P, 1.053 GHz, 60 コア
   - 記憶容量：8 GB
   - 理論ピーク性能：1 TFLOPS（ = 1.053 GHz x 16 FLOPS x 60 core）
   - Intel MPI Version 5.0 Update 3 Build 20150128
   - コンパイラ：Intel Fortran version 15.0.3 20150407
   - コンパイラオプション: -ipo20 -O3 -warn all -openmp -mcmodel=medium -shared-intel -mmic -align array64byte
   - KMP_AFFINITY=granularity=fine, balanced（スレッドをソケット内に均等に配置）
   - Native mode で実行
   - ノード当たり Xeon Phi が 1 枚

2. Intel Xeon クラスタ(Ivy Bridge)
   - CPU：Intel Xeon E5-2670 V2 @ 2.50GHz, 2 ソケット×10 コア
   - ハイパースレッディング：オン (HT2)
   - 1 ノード理論性能：400 GFLOPS
   - 1 ノード記憶容量：64 GB
   - MPI: MVAPICH2 2.0
   - コンパイラ：Intel Fortran version 15.0.3 20150407
   - コンパイラオプション：-ipo20 -O3 -warn all -openmp -mcmodel=medium -shared-intel
   - KMP_AFFINITY=granularity=fine, compact（スレッドをソケット内に配置）

3. FX10 スーパーコンピュータシステム（FX10）
   - CPU：Sparc64 IX-fx, 1.848 GHz, 16 コア
   - 記憶容量：32 GB
   - 理論ピーク性能：236.5 GFLOPS
   - 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

Xeon Phi と Ivy Bridge のネットワーク環境は以下である.

- InfiniBand FDR x 2 Ports
  - Mellanox Connect-IB
  - PCI-E Gen3 x16
  - 56Gbps x 2
  - 理論ピークバンド幅：13.6 GB/s
  - フルバイセクション

4.3 評価結果

4.3.1 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.

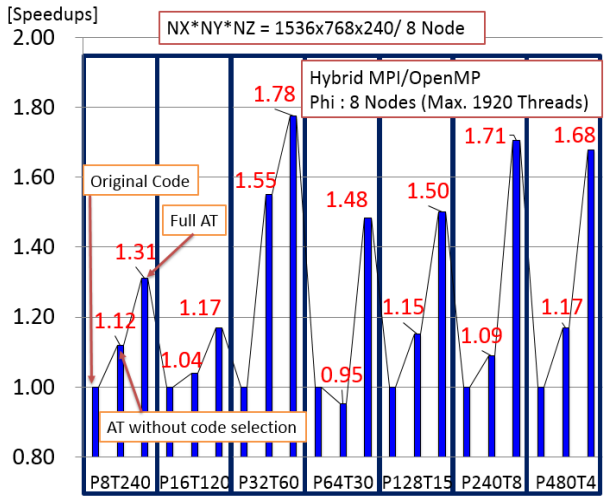
- 表記の PXTYは，X MPI プロセスと，プロセス当たり Y スレッドでの実行を意味する.

4.3.2 AT の結果

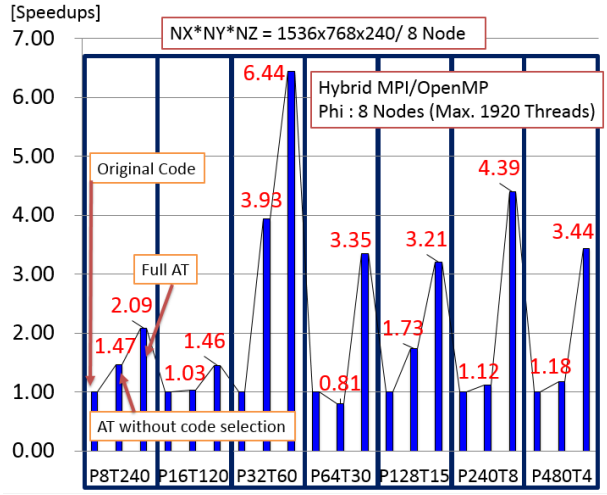
Xeon Phi および Ivy Bridge においては，スカラ向けコードが AT で選択された．一方，FX10 においては，ベクトル向けコードが AT で選択された.

4.3.3 Xeon Phi での結果と考察

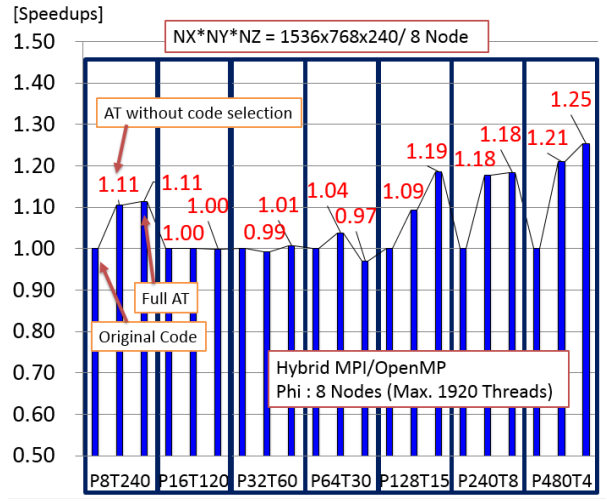
図 10 に Xeon Phi での AT の効果を示す.ここで,Original Code は AT なしでの実行時間を意味し，この AT なしでの実行時間を 1.00 として，AT の効果を示している．また，図中の「AT without code selection」とは，コード選択無しで従来の AT をすべて適用した結果である．ここでのコード選択無しの場合は，ベクトル計算機向きコードが選択されている．また図中の「Full AT」とは，コード選択を含む全ての AT を行った場合の結果である．すべての時間は，ランク 4 番について,2000 回の時間ステップにおける該当箇所の実行時間の累積値を基に計算している.



(a)全体時間における AT 効果



(b)update_stress カーネルにおける AT 効果



(c)update_vel カーネルにおける AT 効果

図 10　Xeon Phi での AT 効果

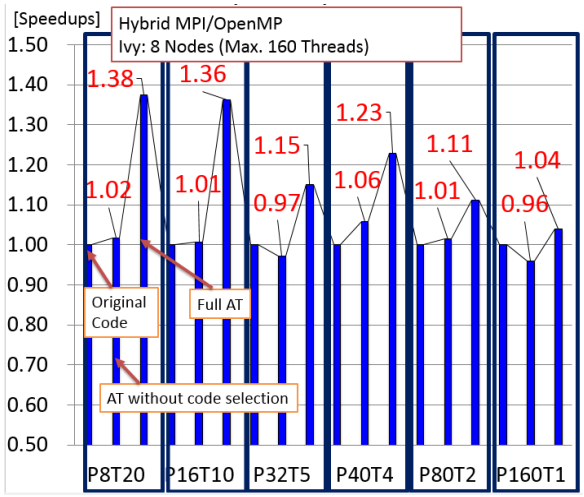
図 10(a)より，全体時間に関して，従来 AT では最大で 1.55 倍ほどの高速化(P32T60)が上限であったが，コード選択の AT を導入することで最大で 1.78 倍(P32T60)の速度向上が得られる．また従来 AT では効果が無い P64T30 の実行形態においても，コード選択を行うことで 1.48 倍の速度向上が得られている．

図 10(b)から，update_stress カーネルにおける AT 効果は特に大きい．従来の AT では，最大で 3.93 倍の速度向上(P32T60)であったが，コード選択の AT を導入することで，最大で 6.44 倍の AT による速度向上（P32T60）が得られている．

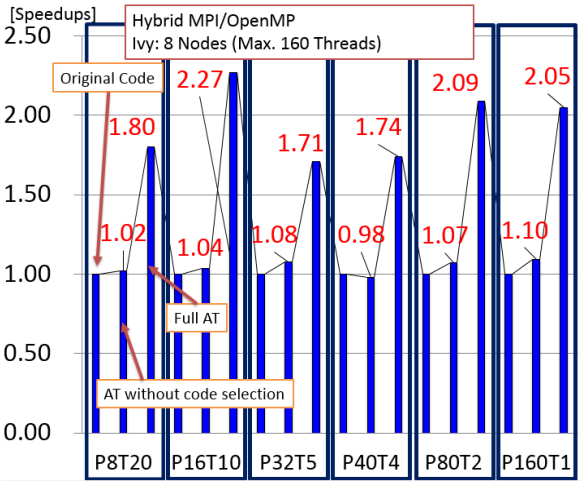
図 10 (c) では，update_vel カーネルにおける AT 効果は，update_stress カーネルにおける AT 効果より全般に少ない．しかしながら，従来の AT に対してコード選択の AT を入れると，より AT の効果が得られる例が多い（7 例中 4 例）．

4.3.4 Ivy Bride での結果と考察

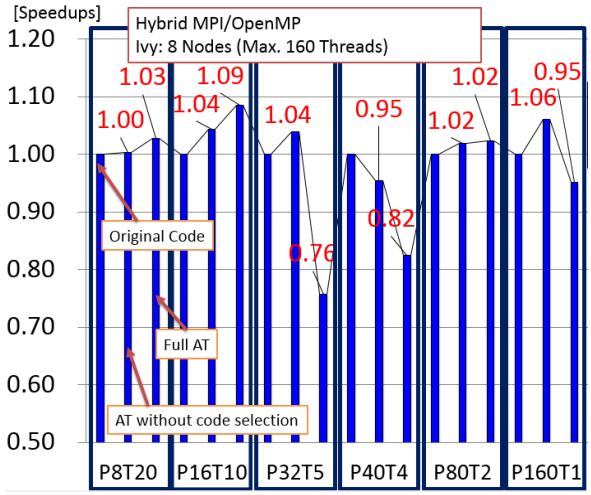
図 11 に Ivy Bridge での AT 効果を載せる．



(a)全体時間における AT 効果



(b)update_stress カーネルにおける AT 効果



(c)update_vel カーネルにおける AT 効果

図 11　Ivy Bridge での AT 効果

図 11 (a) より，全体時間に対して従来 AT では最大で 1.02 倍 (P8T20) の速度向上に留まっていたが，コード選択の AT を行うことで最大で 1.38 倍 (P8T20) の速度向上を得ることができた．

図 11 (b) より，update_stress カーネルにおいても，従来 AT では最大で 1.10 倍(P160T1)であったが，コード選択の AT をすることで最大で 2.27 倍(P16T10)に達する．

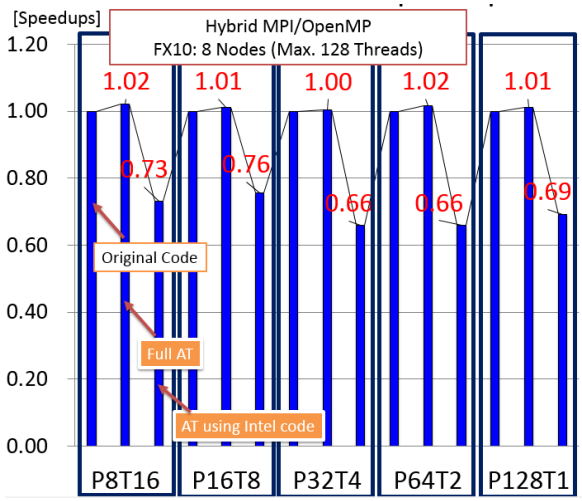
図 11(c)より，update_vel カーネルでは，従来 AT よりもコード選択をすると遅くなる場合がある．この場合においても，実行起動前時 AT でのカーネルの時間測定では，スカラ計算機向けコードが高速と判断されている．そのため，スカラ計算機向け実装が選択されているが，実際の実行時間で遅くなる理由については不明であり，詳細解析する必要がある．

4.3.5 FX10での結果と考察

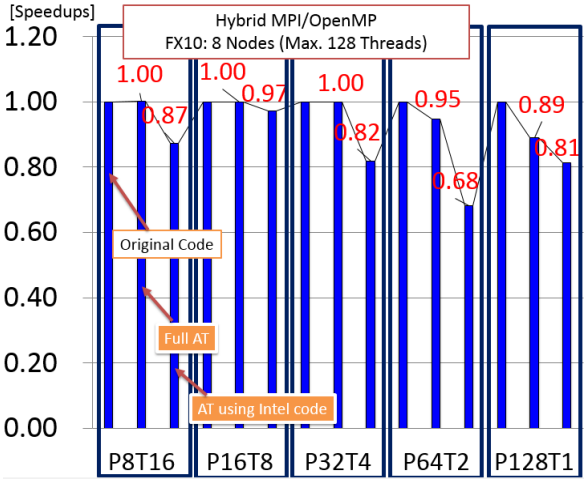
図 11 に，FX10 での AT の効果を載せる．ここで注意は，FX10 では従来から使われているベクトル計算機向きコードが選択されるため，図中の「Full AT」は従来の AT と同じになる．そこで，強制的にスカラ計算機向けコードを選択した場合の実行を，図中の「AT using Intel code」に示した．

図 12(a)から，コードをスカラ計算機向けコードにした場合では，FX10 では全体時間で約 0.42 倍の速度向上 (P128T1)になり，大幅な速度低下になる．このことから，単一のコードのみだけでは，多様な計算機環境で高性能を達成できない．本提案によるコード選択の AT により，性能可搬性を達成できる．

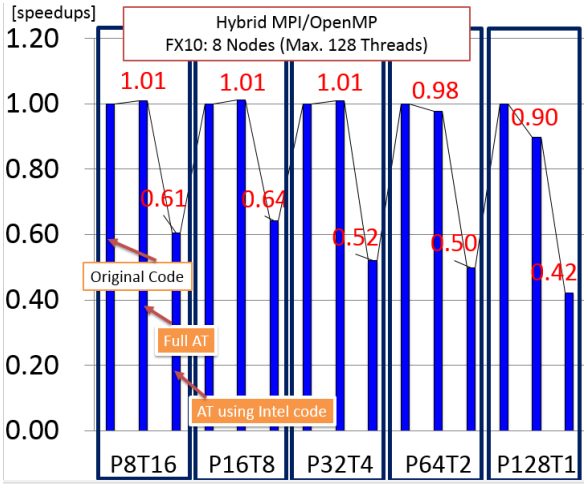
一方，図 12 では，Full AT による効果が，最大でも 1.01 倍と極めて低い．これは，FX10 におけるこの問題サイズの実行では，効率的でない最適化方式が AT で実装されているからといえる．



(a)全体時間における AT 効果



(b)update_stress カーネルにおける AT 効果



(c)update_vel カーネルにおける AT 効果

図 12　FX10 での AT 効果

FX10 においては，スカラ計算機向けコード中の IF 文がコンパイラ最適化を妨げるため，高性能化に寄与しない．スカラ計算機向けコードにおいて IF 文を除去する実装が FX10 では効果的あると予想される．このコードを加えたコード選択機能をもつ AT の評価は，今後の課題である．

5. おわりに

本報告では，AT を実行するに当たり，コード最適化に動的なコード生成とコンパイルを行わず，実行前に静的に生成したコードのみを利用する AT ソフトウェアの構成方式の Static Code Generation Auto-tuning (SCG-AT)を提案した．

SCG-AT による AT を行うに当たり，最も効率的であると思われる階層型 AT 処理を実装した．階層型 AT 処理とは，上位の AT 指定に含まれる手続き等の内部にも AT 指定がある AT 処理である．階層型 AT 処理における典型事例として，本報告ではコード選択の事例を紹介した．さらに，差分法による地震波シミュレーション ppOpen-APPL/FDM において，従来のベクトル計算機向けコードと新規開発したスカラ計算機向けコードのコード選択処理を実装した．

Xeon Phi，Ivy Bridge，FX10 の 3 種の全く異なる計算機で SSG-AT によるコード選択の AT を評価した．評価の結果，Xeon Phi と Ivy Bridge においてはスカラ計算機向けコードの選択により，従来行われていた AT 方式では達成できない速度向上を達成できた．この一方で FX10 においては，従来用いているベクトル計算機向けコードが選択され，新規開発のスカラ計算機向けコードを強制利用すると性能が悪化する．そのため，AT を用いないと性能可搬性が達成できないことを示した．

今後の課題として，FX10 向けのコード選択を行う AT を追加することで，さらに効果的な AT 方式を実現することがあげられる．

謝辞

本研究は，JST CREST 領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」, H23 年度採択課題「自動チューニング機構を有するアプリケーション開発・実行環境」（代表：中島研吾 東大教授）の支援による．日頃ご議論いただく ppOpen-HPC プロジェクトの諸氏に感謝します．また本機能の開発に関し，ppOpen-APPL/FDM の利用でご協力いただいた，東京大学の古村孝志 教授，および，森太志 博士に感謝します．

また本研究の一部は，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）の支援による．

参考文献

1) 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)
2) 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)
3) 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)
4) 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)
5) 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)
6) H. Kuroda, T. Katagiri, M. Kudoh, Y. Kanada, “ILIB_GMRES: An auto-tuning parallel iterative solver for linear equations,” SC2001 (2001) (A Poster.)
7) 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)
8) 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)
9) 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)
10) T. Sakurai, T. Katagiri, H. Kuroda, K. Naono, M. Igai, S. Ohshima, "A 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)
11) 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)
12) 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)

13) R.L. Ribler, J.S. Vetter, H. Simitci, D.A. Reed, “Autopilot: Adaptive control of distributed applications,” In HPDC '98: Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, pp. 172 (1998)
14) C. Cascaval, E. Duesterwald, P.F. Sweeney, R.W. Wisniewski, “Multiple page size modeling and optimization,” Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT 2005), pp. 339–349 (2005)
15) A. Tiwari, J.K. Hollingsworth, “Online adaptive code generation and tuning,” Parallel and Distributed Processing Symposium (IPDPS), pp. 879–892 (2011)
16) J. Xiong, J. Johnson, R. Johnson, D. Padua, “SPL: A language and compiler for DSP algorithms,” Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation (PLDI ’01), pp. 298–308 (2001)
17) 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)
18) 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)
19) 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)
20) 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)
21) 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)
22) 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)
23) 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)
24) 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)
25) D. A. Padua, M.J. Wolfe, “Advanced compiler optimizations for supercomputers,” Communications of the ACM, Vol. 29, pp.1184–1201 (1986)
26) S. Leung, J. Zahorjan, “Improving the performance of runtime parallelization,” Proceedings of PPoPP93, pp. 83–91 (1993)
27) L. Rauchwerger, and D. Padua, “The LRPD Test: Speculative Run-time Parallelization of Loops with Privatization and Reduction Parallelization,” IEEE Transaction on Parallel and Distributed Systems, Vol. 10, No. 2, pp. 160-180 (1999)
28) F. Dang, H. Yu, L. Rauchwerger, “The R-LRPD test: Speculative parallelization of partially parallel loops,” Parallel and Distributed Processing Symposium (IPDPS) 2002, pp. 20–29 (2002)
29) M.J. Voss, R. Eigenmann, “High-level adaptive program optimization with ADAPT,”, ACM SIGPLAN Notices 2001, pp. 93–102 (2001)
30) 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)
31) ppOpen-HPC: http://ppopenhpc.cc.u-tokyo.ac.jp/
32) 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)
33) T. Furumura, “Large-scale parallel FDM simulation for seismic waves and strong shaking,” Supercomputing News, Information Technology Center, The University of Tokyo, Vol. 11, Special Edition 1 (2009) (In Japanese.)