ppOpen-ATの概要、インストールと実行

片桐 孝洋[†]

[†]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

第20回AT研究会オープンアカデミックセッション(ATOS20) ppOpen-AT講習会
日時：2018年8月31日（金）13:10-14:50
会場：工学院大学新宿キャンパス 0712教室

話の流れ

第1部:背景

1. ソフトウェア自動チューニングとは
2. 自動チューニング記述言語
3. 性能モデル化とAT
4. 適用事例:有限差分法コード
5. 性能評価
6. 今後の展望

第2部:インストールと実行

1. ppOpen-ATのインストール
2. コンパイルと実行
3. 質疑応答

話の流れ

第1部：背景

1. ソフトウェア自動チューニングとは
2. 自動チューニング記述言語
3. 性能モデル化とAT
4. 適用事例：有限差分法コード
5. 性能評価
6. 今後の展望

第2部：インストールと実行

1. ppOpen-ATのインストール
2. コンパイルと実行
3. 質疑応答

ソフトウェア
自動チューニングとは

ソフトウェア自動チューニング（AT）とは

- 広義
  - 「自動的に性能をチューニングすること、あるいは、性能チューニングの自動化」
  - ＜性能＞は計算機（対象マシン）の計算速度に限定しない
    - ネットワークを対象とすれば通信速度
    - 計算と通信の両者を含んだシステム性能として応答速度
    - 記憶量、消費電力量、信頼性など、性能基準が決まるもの全て
    - ＜自動＞は機械学習を意味するが、人間の介在を否定しない
- 狭義
  - 適用分野ごとに異なる
  - 数値計算分野においては、「応用プログラム（アプリケーション）の性能向上をもたらす自動的な性能チューニング」
    - 異なる対象マシンに対しても高性能が得られるように、与えられた応用プログラムをチューニングする過程を自動化すること
    - 原義：人手による性能チューニング作業の自動化

引用：自動チューニング研究会HP:　http://atrg.jp/ja/

ソフトウェア自動チューニング（AT）とは

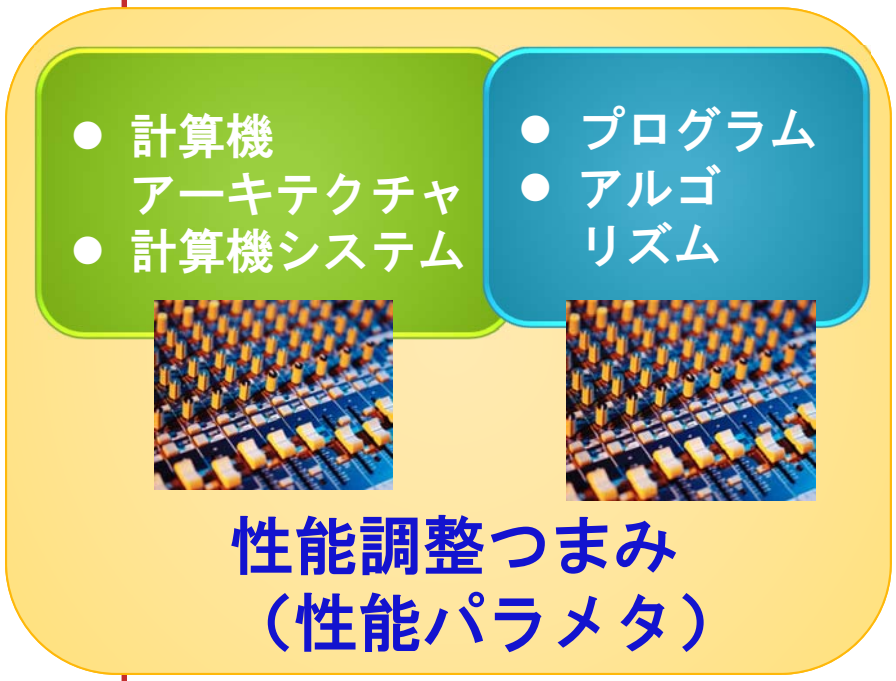
i. プログラム上の対象個所を決定

ii. 対象個所中で、性能に影響を及ぼす
パラメタ（性能調整つまみ）
集合X を抽出

iii. チューニングする評価基準の
関数F をパラメタ集合X で定義

iv. 関数F ： X →Y を最小化する
パラメタ集合X の値をプログラムを
実行しながら見つける
（解パラメタ集合X^* ）

v. 解パラメタ集合X^* でプログラムを実行



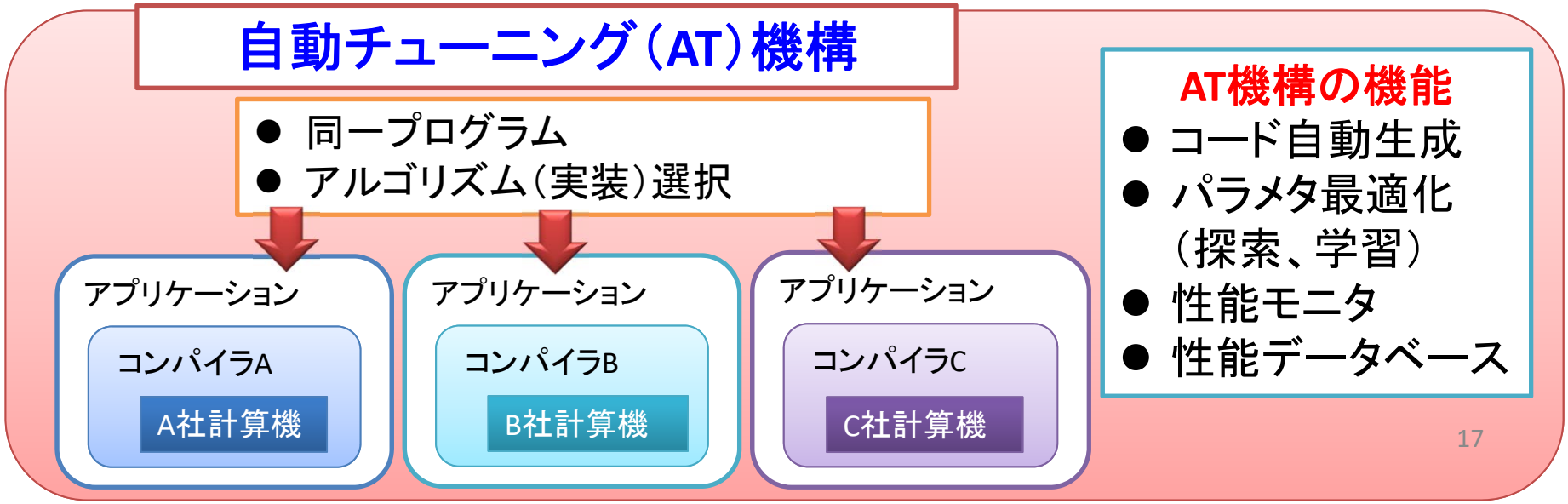
- i.～iv.が全自動になっていることが望ましいが、
一部のみ自動（半自動）でも、自動チューニングと呼ぶ。
- **性能に影響を及ぼすパラメタの抽出には、**
専用言語を利用する（ABCLibScript、ppOpen-AT）

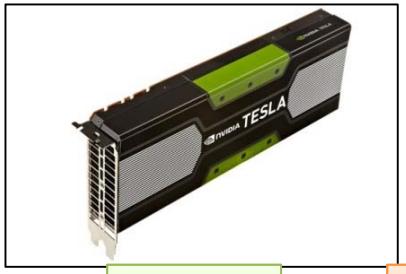
AT機構とは


•計算機アーキテクチャ
•計算機システム
•プログラム
•アルゴリズム
性能調整つまみ
（性能パラメタ）
調整機構
•最適化
•パラメタ探索
•学習／自己適応
•プログラム
•アルゴリズム
性能モニタ
機構
つまみ
自動生成
機構
AT性能データベース
自動チューニング機構

性能可搬性(Performance Portability)とは？

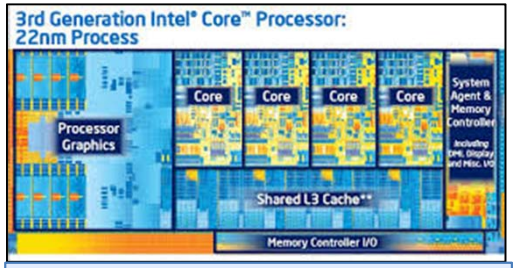
- 複数計算機環境での最適化を提供するパラダイム
  （HPCI 技術ロードマップ白書、数値計算ライブラリのための自動チューニング2012 年3 月）
  - 2000年頃から日本で使われている技術用語
  - 同一プログラムで計算機環境が変わっても高性能を維持





GPU

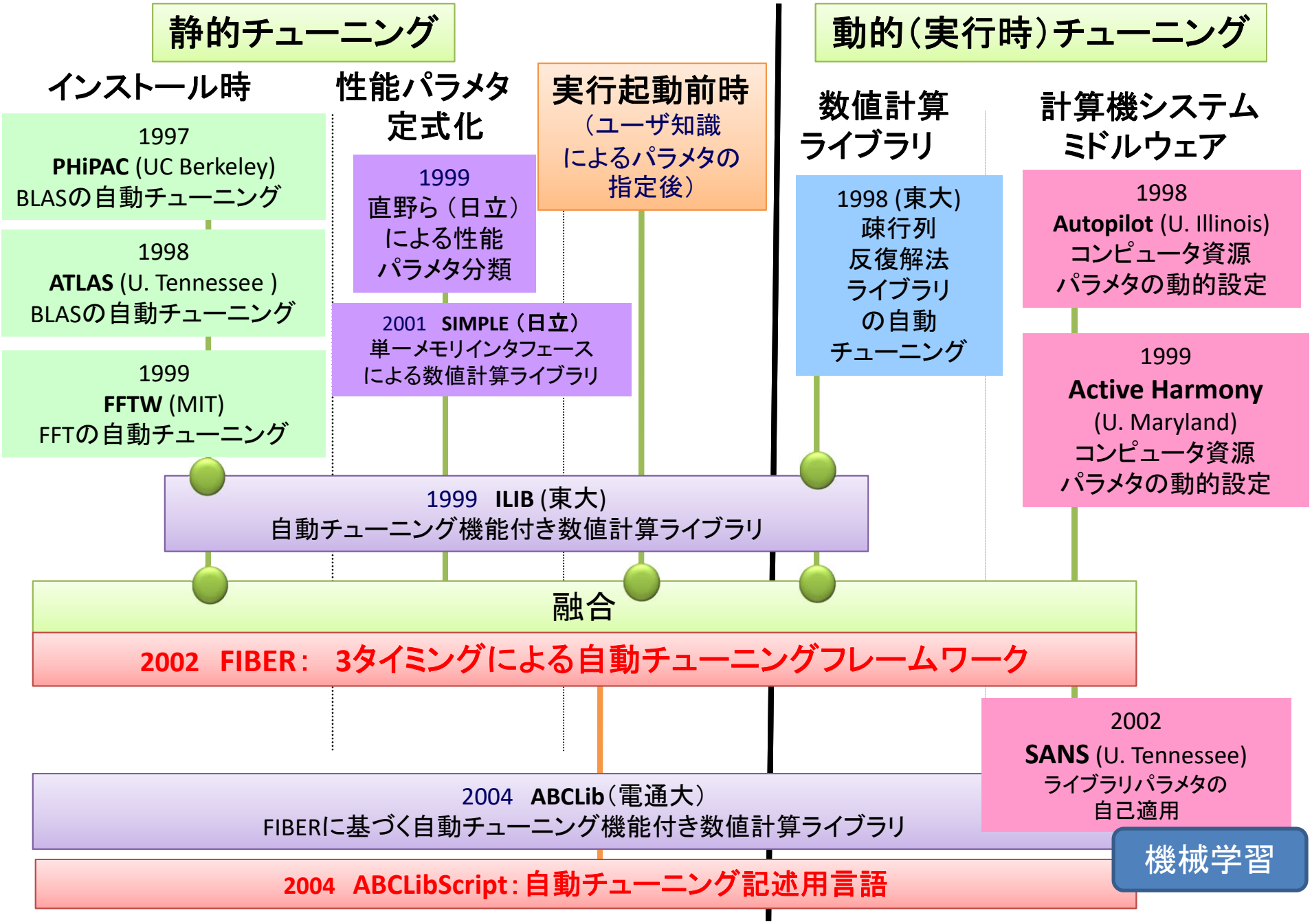
メニーコアCPU



マルチコアCPU

エクサに向けた
多様な
ハードウェア
環境

自動チューニング研究の分類

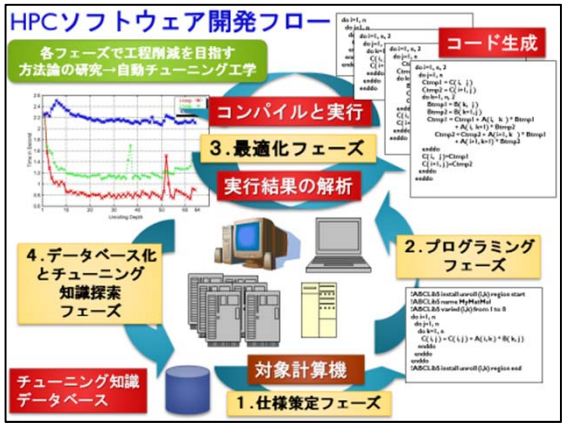


ATシステムと機械学習（数値計算）

- 最適パラメタ（アルゴリズム）の自動分類による、最適パラメタ推定
  - 性能モデルによる性能値の推定ではない
- AutoPilot [2001, D. Reed, U. Illinois]
  - Fuzzy Rule
- Sans and Salsa [2002, J. Dongarra, U. Tennessee]
  - Self-Adapting Numerical Software (SANS)
  - Self-Adapting Large-scale Solver Architecture (SALSA)
  - Alternating Decision Trees (AdaBoost)
- ATMathCoreLib [2011, R. Suda, U. Tokyo]
  - Bayesian Inference
- Compiler Optimization [2012, J. Cavazos, U. Delaware]
  - Artificial Neural Network
  - Neuro-Evolution for Augmenting Topologies
- Nitro [2015, M. Hall, U. Utah]
  - Support Vector Machine
  - Radial-Basics Function

自動チューニング記述言語

AT専用言語ppOpen-ATによるソフトウェア開発手順



ソフトウェア開発者

- ●計算機環境に非依存
- ●コード、計算機資源、電力、精度、の最適化方式を記述

↓

ppOpen-ATによる自動チューニング記述

↓

専用言語処理系（プリプロセッサ）の起動

↓

AT機構が付加されたプログラム

↓

最適化候補とAT機構が付加された実行可能コード

```
#pragma OAT install unroll (i,j,k) region start
#pragma OAT varied (i,j,k) from 1 to 8
for(i = 0 ; i < n ; i++){
 for(j = 0 ; j < n ; j++){
  for(k = 0 ; k < n ; k++){
   A[i][j]=A[i][j]+B[i][k]*C[k][j]; }}}
#pragma OAT install unroll (i,j,k) region end
```

■自動生成される機構
- ●最適化候補
- ●性能モニタ
- ●パラメタ探索
- ●性能モデル化

コンパイラではできない最適化

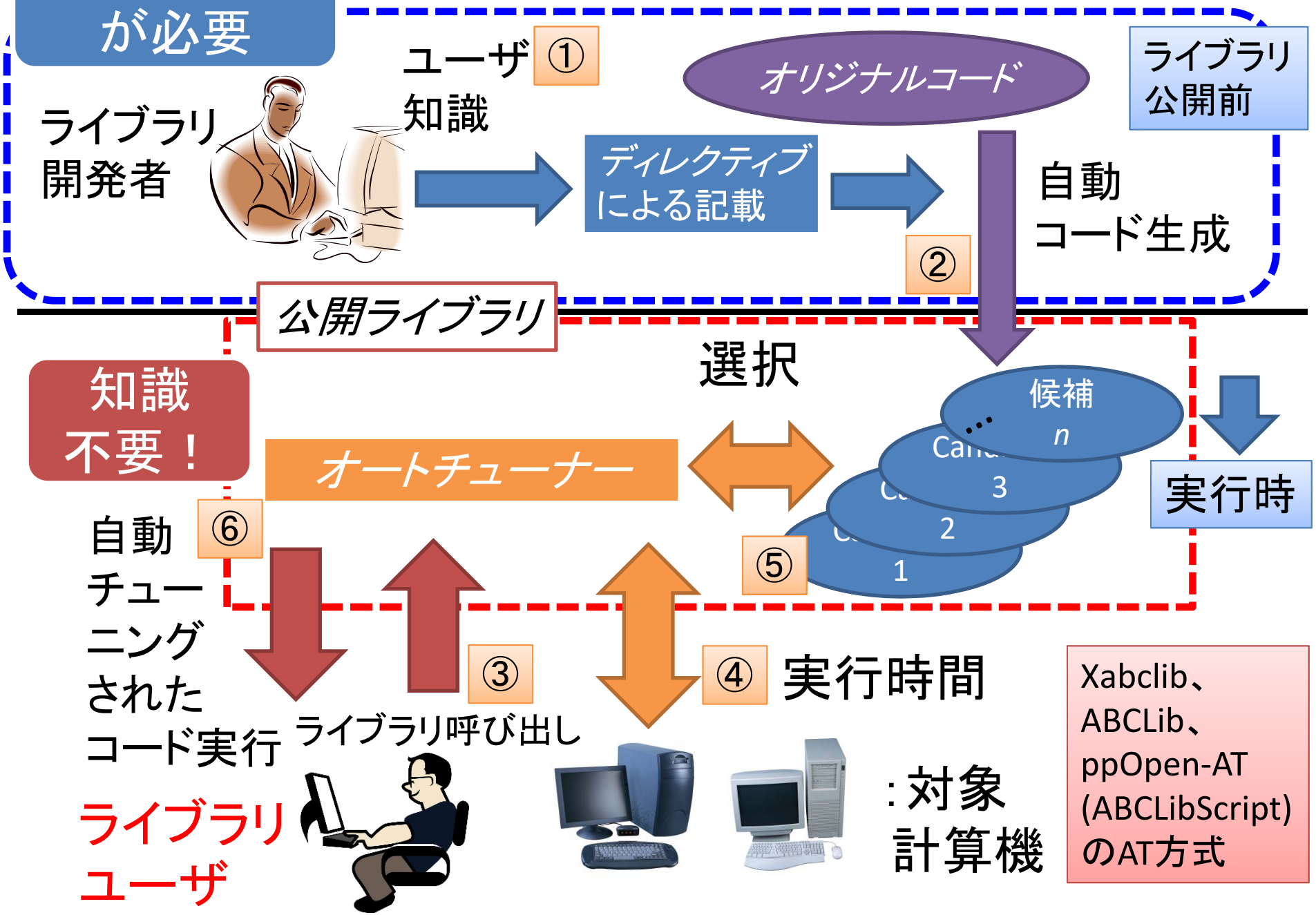
ループアンローリングの例（行列-行列積、Fortran言語）

- i-ループ および j-ループ 2段展開 (nが２で割り切れる場合)

```
do i= 1 , n,  2
  do j= 1 , n,  2
    do k= 1 ,  n
      C(i      , j      ) = C(i      ,  j     ) +A(i      , k) * B(k, j      )
      C(i      , j+ 1 ) = C(i      , j+ 1 ) +A(i      , k) * B(k, j+ 1  )
      C(i+ 1 , j      ) = C(i+ 1 , j      ) +A(i+ 1 , k) * B(k, j      )
      C(i+ 1 , j+ 1 ) = C(i+ 1 , j+ 1 ) +A(i+ 1 , k) * B(k, j + 1 )
 enddo;  enddo;  enddo;
```

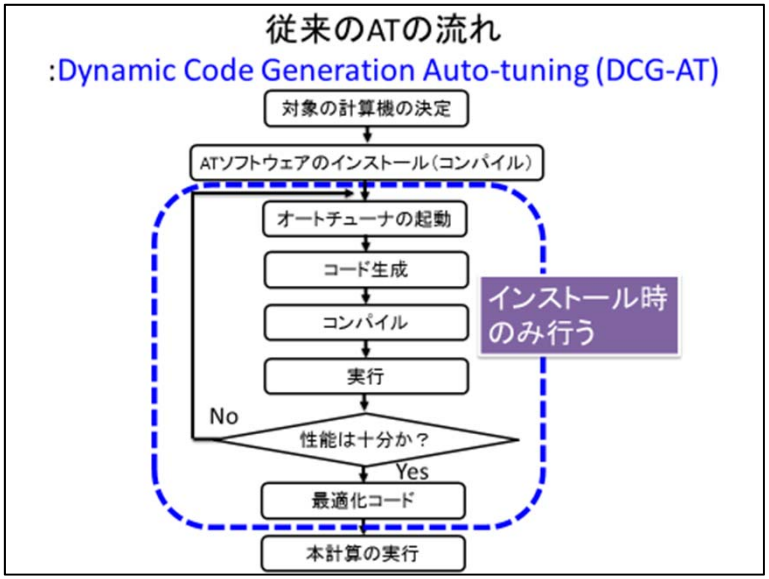
- A(i, j)、A(i+ 1 , k)、B(k, j)、B(k, j+ 1)をレジスタに置き高速化
- 何段が最適かは計算機依存（段数が＜性能パラメタ＞）

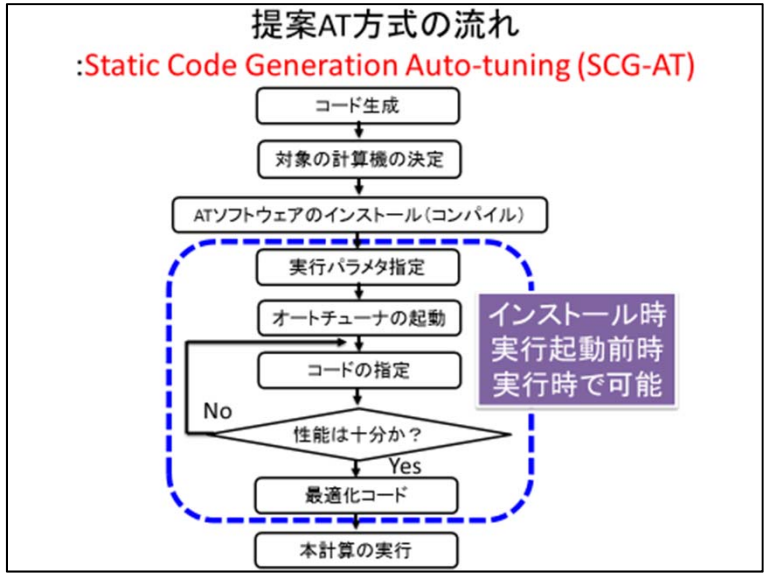
FIBERフレームワーク[Katagiri et.al., 2003]



FIBERフレームワークの特徴
(ABCLibScript、ppOpen-AT)

- AT候補を含む、ATに関係するコードを静的に1度だけ生成する(SCG-AT方式)
  - 既存の全てのATソフトは、コードは動的生成(DCG-AT方式)
- ☹SCG-AT方式ではコード量増大を招くため、適切なAT指示が必要(ユーザ知識を利用)
- ☺コンパイル1回のみ、コード生成のスプリクト言語は不要
  - 超軽量な実行時AT機構を実装可能
  - スパコン環境(バッチジョブ環境、OSカーネルは改変できない)環境で、すべてユーザ権限で、簡単にAT実行可能
    →高い利便性、適用範囲が広い





性能モデル化とAT

AT時間削減のための性能モデル化

- ユーザ知識が足りない、対象の問題が複雑な場合、性能パラメタの組合わせが増加

  →性能パラメタの組合わせが増加すると、探索空間が増加

  →AT時間が増加

- 性能をモデル化し、効率よく最適パラメタを探索する必要

ATのための性能モデル化

- 一部の性能パラメタでの測定結果をもとに、未計測性能パラメタでの実行性能値を予測
- ホワイトボックスモデル
  - 対象となるプログラムコードの構成（ループ構造や演算特徴）を見て、性能モデル化する
- ブラックボックスモデル
  - 対象となる箇所の出力（実行時間、電力、・・・）を見て、性能モデル化する
- ブラックボックスモデルのほうが、汎用性が高い
  - ホワイトボックスモデルはアプリ特化（例：ステンシル演算用の性能モデル）

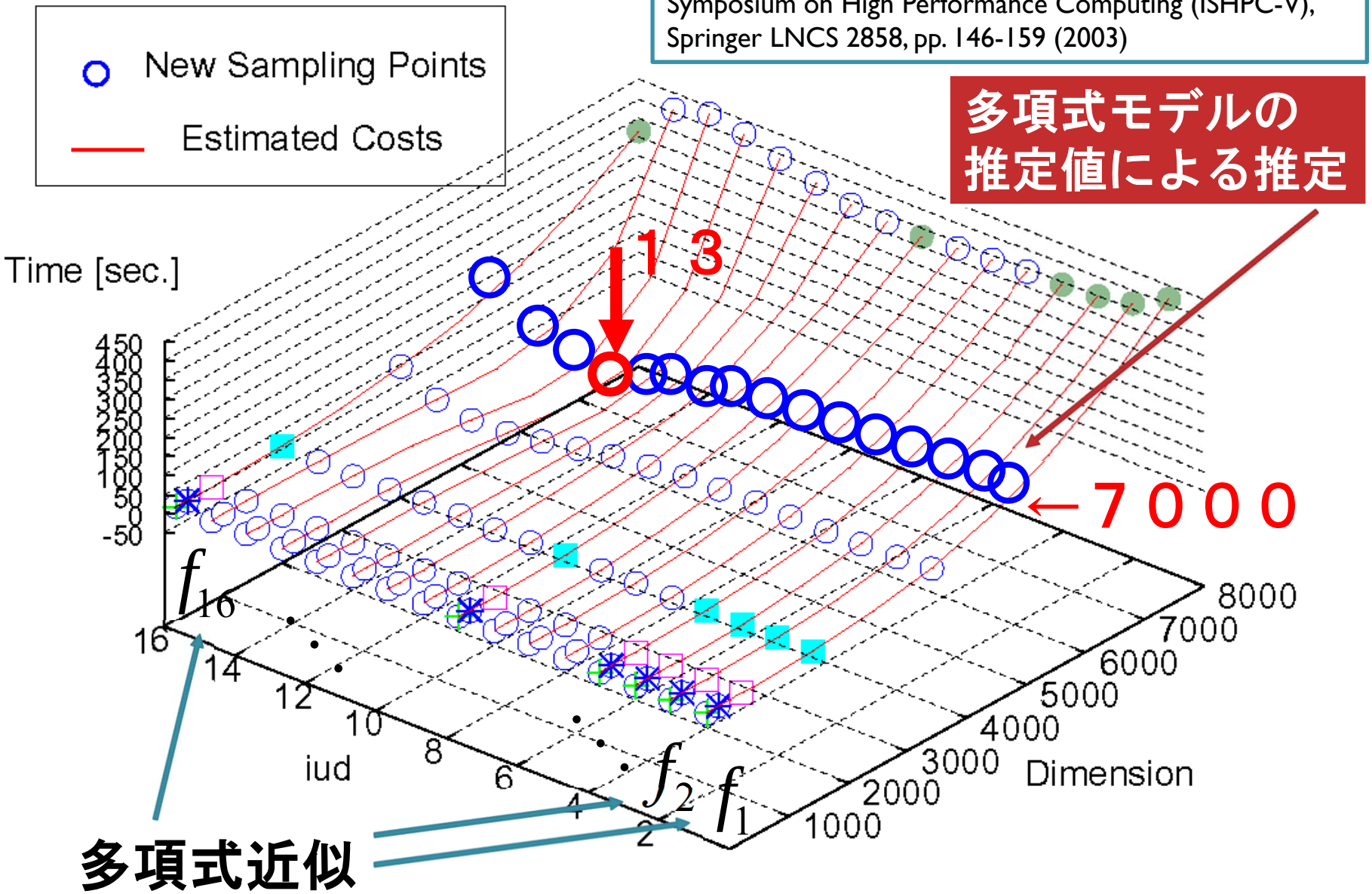
性能モデルと既存研究

ホワイトボックスモデル			ブラックボックスモデル	
ATソフトウェア名	対象演算	アルゴリズム	ATソフトウェア名	アルゴリズム
ATLAS	BLAS（行列-行列積）	キャッシュブロック化考慮の経験則	Active Harmony	Simplex法による最適化
OSKI	疎行列-ベクトル積	キャッシュサイズ、非零位置情報、考慮の経験則	**ABCLibScript (FIBER方式)**	多項式係数を最小二乗法で最適化
Physis	ステンシル演算	ループ融合を考慮した経験則	**ppOpen-AT**	D-Splineによる補間、および、IPPE方式による最適パラメタ予測法
ー（論文のみ）	BLAS(行列-行列積) ステンシル演算	・DGEMM最適化の経験則モデル提案は多数 ・ループ構成から多項式近似	ー（論文のみ）	・遺伝的アルゴリズム（GA） **・サロゲーションモデル**
			ATMathCoreLib	測定誤差の正規分布仮定とベイズ推定

FIBER方式による性能推定

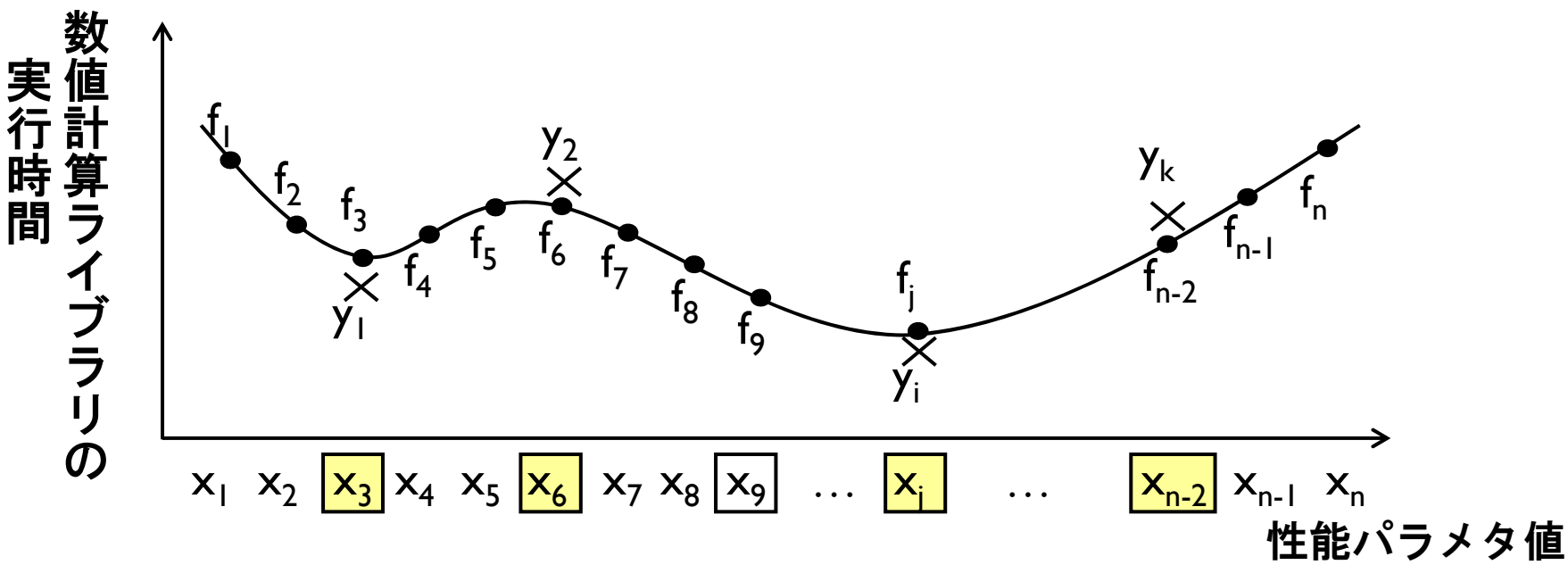
(HITACHI SR8000/MPP)

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)



コスト定義関数 *d-Spline*

コスト定義関数 f(x) を $f=(f_1,f_2,f_3,\ldots,f_j,\ldots,f_n)^t$ で表現



□：性能パラメタの取り得る値（内，■：標本点）

y_i：標本点に対する数値計算ライブラリの実測値（$1 \leq i \leq k,\ k < n$）

田中輝雄、片桐孝洋、弓場敏嗣：ソフトウェア自動チューニングにおける標本点追加型性能パラメタ推定法、電子情報通信学会論文誌 A、Vol.J90-A、No.4、pp .281-291 (2007)

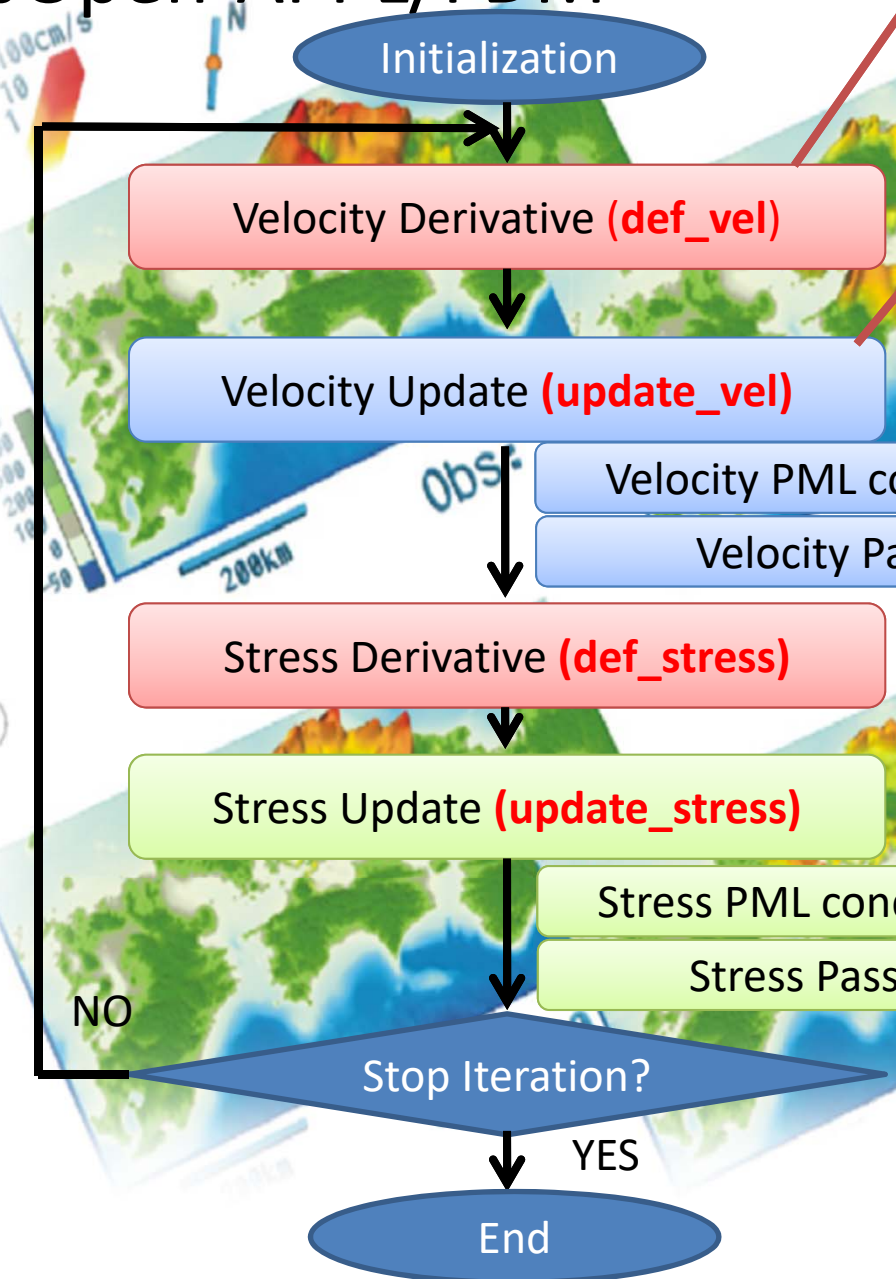
適用事例
:有限差分法コード

共同研究者:
松本正晴、大島聡史

Target Application

- **Seism_3D**:
  Simulation for seismic wave analysis.
- Developed by Professor Furumura at the University of Tokyo.
  - The code is re-constructed as ppOpen-APPL/FDM.
- Finite Differential Method (FDM)
- 3D simulation
  - 3D arrays are allocated.
- Data type: Single Precision (real*4)

Flow Diagram of ppOpen-APPL/FDM



- Space difference by FDM.

$$\frac{d}{dx}\sigma_{pq}(x,y,z) = \frac{1}{\Delta x}\sum_{m=1}^{M/2} c_m[\sigma_{pq}\{x+(m+\frac{1}{2})\Delta x, y, z\} - \sigma_{pq}\{x-(m-\frac{1}{2})\Delta x, y, z\}],$$

$$(p,q = x,y,z)$$

- Explicit time expansion by central difference.

$$\dot{u}_p^{n+\frac{1}{2}} = \dot{u}_p^{n-\frac{1}{2}} + \frac{1}{\rho}\left(\frac{\partial\sigma_{xp}^n}{\partial x} + \frac{\partial\sigma_{yp}^n}{\partial y} + \frac{\partial\sigma_{zp}^n}{\partial z} + f_p^n\right)\Delta t, (p = x,y,z)$$

ppOpen-APPL/FDMへの適用シナリオ

[FIBER方式、2003]
実行起動前時AT

ライブラリ
利用者

↓

行列サイズ、
MPIプロセス数、
OpenMPスレッド数の指定

↓

ATパラメタの指定と
ライブラリの実行
（OAT_AT_EXEC=1）

↓

ATパラメタの指定と
ライブラリの実行
（OAT_AT_EXEC=0）

↓

AT無しで、最適化された
カーネルの実行

■オートチューナの実行
- ●固定ループ長での実行
  （行列サイズ、MPIプロセス数、
  OpenMPスレッド数の固定による）
- ●対象カーネルの実行時間の計測
- ●最適パラメタ情報の収納

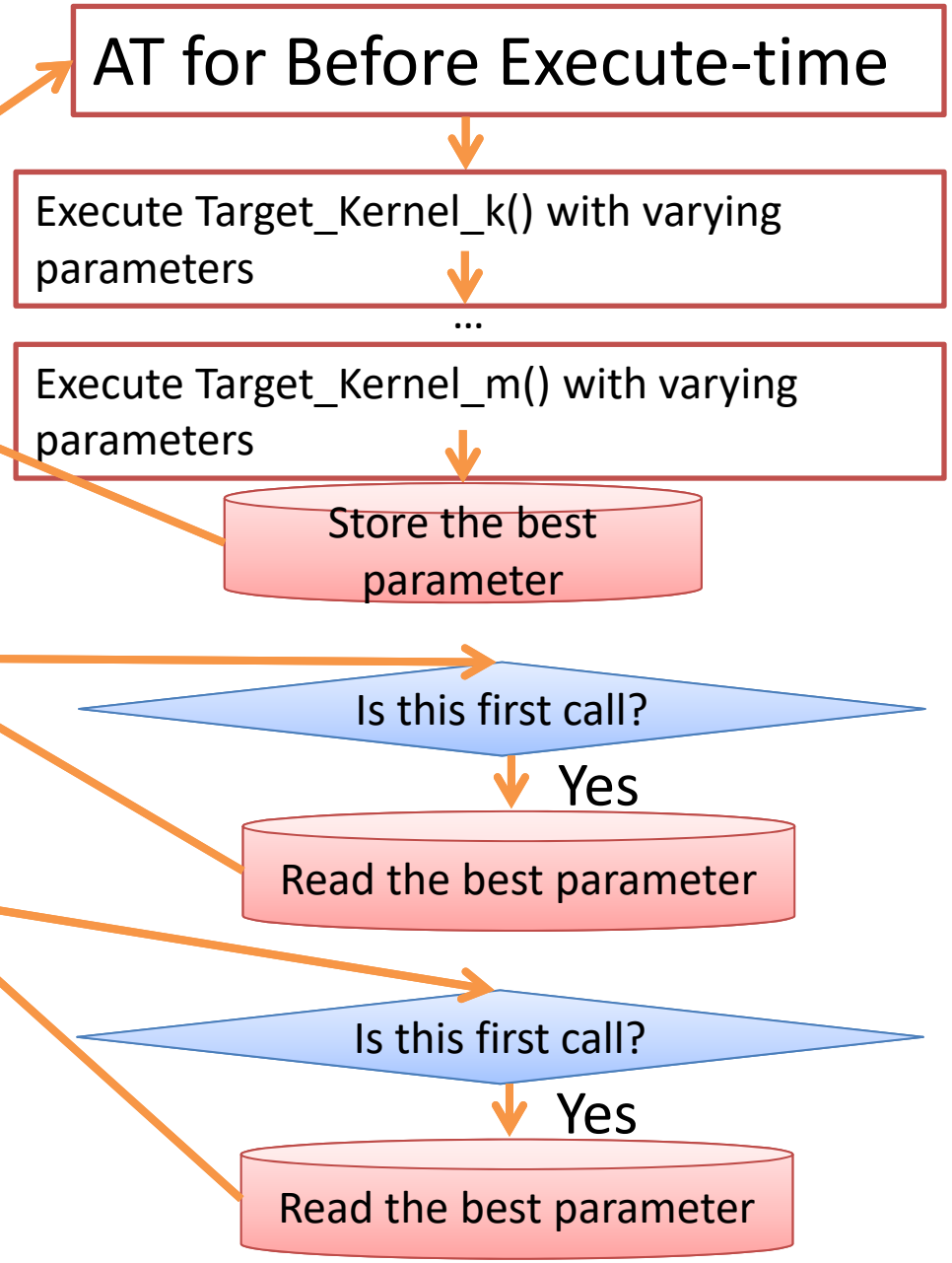
最速カーネル情報の
収納

ATなしで最速カーネルを使う
（行列サイズ、MPIプロセス数、
OpenMPスレッド数を変えない限り有効）

AT Timings of *ppOpen-AT* (FIBER Framework)

One time execution (except for varying problem size and number of MPI processes)

```
OAT_ATexec()
...
do i=1, MAX_ITER
   Target_kernel_k()
   ...
   Target_kernel_m()
   ...
enddo
```



ループ分割 (Loop Split)

```
DO K = 1, NZ
DO J = 1, NY
DO I = 1, NX
   RL(I)  = LAM (I,J,K)
   RM(I)  = RIG (I,J,K)
   RM2(I) = RM(I) + RM(I)
   RMAXY(I) = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J+1,K) + 1.0/RIG(I+1,J+1,K))
   RMAXZ(I) = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I+1,J,K+1))
   RMAYZ(I) = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I,J+1,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I,J+1,K+1))
   RLTHETA(I) = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL(I)
   QG(I) = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
END DO
```

分割点

```
DO I = 1, NX
   SXX (I,J,K) = ( SXX (I,J,K)  + (RLTHETA(I) + RM2(I)*DXVX(I,J,K))*DT )*QG(I)
   SYY (I,J,K) = ( SYY (I,J,K)  + (RLTHETA(I) + RM2(I)*DYVY(I,J,K))*DT )*QG(I)
   SZZ (I,J,K) = ( SZZ (I,J,K)  + (RLTHETA(I) + RM2(I)*DZVZ(I,J,K))*DT )*QG(I)
   SXY (I,J,K) = ( SXY (I,J,K)  + (RMAXY(I)*(DXVY(I,J,K)+DYVX(I,J,K)))*DT )*QG(I)
   SXZ (I,J,K) = ( SXZ (I,J,K)  + (RMAXZ(I)*(DXVZ(I,J,K)+DZVX(I,J,K)))*DT )*QG(I)
   SYZ (I,J,K) = ( SYZ (I,J,K)  + (RMAYZ(I)*(DYVZ(I,J,K)+DZVY(I,J,K)))*DT )*QG(I)
END DO
END DO
END DO
```

K, J, I-ループに対する ループ融合(Loop Collapse)

```
DO KK = 1, NZ * NY * NX
  K = (KK-1)/(NY*NX) + 1
  J = mod((KK-1)/NX,NY) + 1
  I = mod(KK-1,NX) + 1
  RL  = LAM (I,J,K)
  RM  = RIG (I,J,K)
  RM2 = RM + RM
  RMAXY = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J+1,K) + 1.0/RIG(I+1,J+1,K))
  RMAXZ = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I+1,J,K+1))
  RMAYZ = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I,J+1,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I,J+1,K+1))
  RLTHETA = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
  QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
  SXX (I,J,K) = ( SXX (I,J,K)  + (RLTHETA + RM2*DXVX(I,J,K))*DT )*QG
  SYY (I,J,K) = ( SYY (I,J,K)  + (RLTHETA + RM2*DYVY(I,J,K))*DT )*QG
  SZZ (I,J,K) = ( SZZ (I,J,K)  + (RLTHETA + RM2*DZVZ(I,J,K))*DT )*QG
  SXY (I,J,K) = ( SXY (I,J,K)  + (RMAXY*(DXVY(I,J,K)+DYVX(I,J,K)))*DT )*QG
  SXZ (I,J,K) = ( SXZ (I,J,K)  + (RMAXZ*(DXVZ(I,J,K)+DZVX(I,J,K)))*DT )*QG
  SYZ (I,J,K) = ( SYZ (I,J,K)  + (RMAYZ*(DYVZ(I,J,K)+DZVY(I,J,K)))*DT )*QG
END DO
```

OpenMPで並列実行するとき高並列性能を実現

K, J-ループに対するループ融合

OpenMPで並列実行するとき高並列性能を実現

連続アクセスを残し、コンパイラ最適化を有効に

```
DO KK = 1, NZ * NY
  K = (KK-1)/NY +  1
  J = mod(KK-1,NY) + 1
  DO I = 1, NX
   RL  = LAM (I,J,K)
   RM  = RIG (I,J,K)
   RM2 = RM + RM
   RMAXY = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J+1,K) + 1.0/RIG(I+1,J+1,K))
   RMAXZ = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I+1,J,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I+1,J,K+1))
   RMAYZ = 4.0/(1.0/RIG(I,J,K) + 1.0/RIG(I,J+1,K)  + 1.0/RIG(I,J,K+1) + 1.0/RIG(I,J+1,K+1))
   RLTHETA = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
   QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
   SXX (I,J,K) = ( SXX (I,J,K)  + (RLTHETA + RM2*DXVX(I,J,K))*DT )*QG
   SYY (I,J,K) = ( SYY (I,J,K)  + (RLTHETA + RM2*DYVY(I,J,K))*DT )*QG
   SZZ (I,J,K) = ( SZZ (I,J,K)  + (RLTHETA + RM2*DZVZ(I,J,K))*DT )*QG
   SXY (I,J,K) = ( SXY (I,J,K)  + (RMAXY*(DXVY(I,J,K)+DYVX(I,J,K)))*DT )*QG
   SXZ (I,J,K) = ( SXZ (I,J,K)  + (RMAXZ*(DXVZ(I,J,K)+DZVX(I,J,K)))*DT )*QG
   SYZ (I,J,K) = ( SYZ (I,J,K)  + (RMAYZ*(DYVZ(I,J,K)+DZVY(I,J,K)))*DT )*QG
  ENDDO
END DO
```

ppOpen-ATディレクティブ
: ループ分割とループ融合の全組合わせ指定

```
!oat$ install LoopFusionSplit region start
!$omp parallel do private(k,j,i,STMP1,STMP2,STMP3,STMP4,RL,RM,RM2,RMAXY,RMAXZ,RMAYZ,RLTHETA,QG)
    DO K = 1, NZ
    DO J = 1, NY
    DO I = 1, NX
        RL  = LAM (I,J,K);   RM  = RIG (I,J,K);   RM2 = RM + RM
        RLTHETA  = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
!oat$ SplitPointCopyDef  region  start
        QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
!oat$ SplitPointCopyDef  region  end
        SXX (I,J,K) = ( SXX (I,J,K) + (RLTHETA + RM2*DXVX(I,J,K))*DT )*QG
        SYY (I,J,K) = ( SYY (I,J,K) + (RLTHETA + RM2*DYVY(I,J,K))*DT )*QG
        SZZ (I,J,K) = ( SZZ (I,J,K) + (RLTHETA + RM2*DZVZ(I,J,K))*DT )*QG
!oat$ SplitPoint  (K, J, I)
        STMP1 = 1.0/RIG(I,J,K);  STMP2 = 1.0/RIG(I+1,J,K);  STMP4 = 1.0/RIG(I,J,K+1)
        STMP3 = STMP1 + STMP2
        RMAXY = 4.0/(STMP3 + 1.0/RIG(I,J+1,K) + 1.0/RIG(I+1,J+1,K))
        RMAXZ = 4.0/(STMP3 + STMP4 + 1.0/RIG(I+1,J,K+1))
        RMAYZ = 4.0/(STMP3 + STMP4 + 1.0/RIG(I,J+1,K+1))
!oat$ SplitPointCopyInsert
        SXY (I,J,K) = ( SXY (I,J,K) + (RMAXY*(DXVY(I,J,K)+DYVX(I,J,K)))*DT )*QG
        SXZ (I,J,K) = ( SXZ (I,J,K) + (RMAXZ*(DXVZ(I,J,K)+DZVX(I,J,K)))*DT )*QG
        SYZ (I,J,K) = ( SYZ (I,J,K) + (RMAYZ*(DYVZ(I,J,K)+DZVY(I,J,K)))*DT )*QG
    END DO;  END DO;  END DO
!$omp end parallel do
!oat$ install LoopFusionSplit region end
```

ループ分割とループ融合の全組合わせ指定

ループ分割時の再計算式宣言

ループ分割点の指定

ループ分割時の再計算場所の指定

コード選択のAT
（階層型AT）

階層型AT処理の例（2階層）

上位のAT指定

```
program main
  ....
＜AT対象のループ＞
do i=1, N
  ...
  call LowerRoutine( ....)
  ...
enddo
```

下位のAT指定

```
subroutine
    LowerRoutine(....)
  ....
  ＜AT記述の処理＞
  ....
return
end
```

Original Implementation (For Vector Machines)

Fourth-order accurate central-difference scheme for velocity. (**def_stress**)

```
call ppohFDM_pdiffx3_m4_OAT( VX,DXVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
call ppohFDM_pdiffy3_p4_OAT( VX,DYVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
call ppohFDM_pdiffz3_p4_OAT( VX,DZVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
call ppohFDM_pdiffy3_m4_OAT( VY,DYVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,... )
call ppohFDM_pdiffx3_p4_OAT( VY,DXVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,... )
call ppohFDM_pdiffz3_p4_OAT( VY,DZVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
call ppohFDM_pdiffx3_p4_OAT( VZ,DXVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
call ppohFDM_pdiffy3_p4_OAT( VZ,DYVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,... )
call ppohFDM_pdiffz3_m4_OAT( VZ,DZVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,...)
```

```
 if( is_fs .or. is_nearfs ) then
      call ppohFDM_bc_vel_deriv( KFSZ,NIFS,NJFS,IFSX,IFSY,IFSZ,JFSX,JFSY,JFSZ )
end if
```

Process of model boundary.

```
call ppohFDM_update_stress(1, NXP, 1, NYP, 1, NZP)
```

Explicit time expansion by leap-frog scheme. (update_stress)

Original Implementation (For Vector Machines)

```
subroutine OAT_InstallppohFDMupdate_stress(..)
!$omp parallel do private(i,j,k,RL1,RM1,RM2,RLRM2,DXVX1,DYVY1,DZVZ1,…)
   do k = NZ00, NZ01
      do j = NY00, NY01
         do i = NX00, NX01
            RL1   = LAM (I,J,K); RM1   = RIG (I,J,K);  RM2   = RM1 + RM1; RLRM2 = RL1+RM2
            DXVX1 = DXVX(I,J,K);  DYVY1 = DYVY(I,J,K);  DZVZ1 = DZVZ(I,J,K)
            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
            DXVYDYVX1 = DXVY(I,J,K)+DYVX(I,J,K);  DXVZDZVX1 = DXVZ(I,J,K)+DZVX(I,J,K)
            DYVZDZVY1 = DYVZ(I,J,K)+DZVY(I,J,K)
            SXY (I,J,K) = SXY (I,J,K) + RM1 * DXVYDYVX1 * DT
            SXZ (I,J,K) = SXZ (I,J,K) + RM1 * DXVZDZVX1 * DT
            SYZ (I,J,K) = SYZ (I,J,K) + RM1 * DYVZDZVY1 * DT
         end do
      end do
   end do
retuen
end
```

Input and output for arrays in each call -> Increase of B/F ratio: ~1.7

Explicit time expansion by leap-frog scheme. (update_stress)

Implementation Strategy for Code Variants

- Change of computation order based on mathematics.
- With experimental knowledge, we implement the following codes for the variants.
- For time expansion of stress tensor, the loop can be separated to two loops with respect to nature of isotropic elastic body (Loop Spilit):
  - Loop1 : *Sxx*, *Syy*, *Szz*, and
  - Loop2: *Sxy*, Sxz, *Syz*,

  where the stress tenser **S** is defined by:
  S = [(*Sxx*, *Sxy*, *Sxz*), (*Sxy*, *Syy*, *Syz*), (*Sxz*, *Syz*, *Szz*)]
- The three nested loop for the above two loops is collapsed with the outer two loops.

The Code Variants (For Scalar Machines)

- Variant1 (IF-statements inside)
  - The followings include inside loop:
  1. Fourth-order accurate central-difference scheme for velocity.
  2. Process of model boundary.
  3. Explicit time expansion by leap-frog scheme.
- Variant2 (IF-free, but there is IF-statements inside loop for process of model boundary.)
  - To remove IF sentences from the variant1, the loops are reconstructed.
  - The order of computations is changed, but the result without round-off errors is same.
  - **[Main Loop]**
  1. Fourth-order accurate central-difference scheme for velocity.
  2. Explicit time expansion by leap-frog scheme.
  - **[Loop for process of model boundary]**
  1. Fourth-order accurate central-difference scheme for velocity.
  2. Process of model boundary.
  3. Explicit time expansion by leap-frog scheme.

Variant1 (For Scalar Machines)

Stress tensor of Sxx, Syy, Szz

```
!$omp parallel do   private
(i,j,k,RL1,RM1,RM2,RLRM2,DXVX, …)
do k_j=1, (NZ01-NZ00+1)*(NY01-NY00+1)
  k=(k_j-
  j=mod(
  do i = N
    RL1
    RM2
! 4th ord
    DXV
        - (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
```

Fourth-order accurate central-difference scheme for velocity.

```
! Y dir
    if( idy == 0 ) then ! Shallowmost
      if (j==1)then
        DYVY0 = ( VY(I,1,K) - 0.0_
      end if
      if (j==2)then
        DYVY0 = ( VY(I,2,K) - VY(
      end if
    end if
    if( idy == JP-1 ) then
      if (j==NYP)then
        DYVY0 = ( VY(I,NYP,K) - V
      end if
    end if
! Z dir
    if( idz == 0 ) then ! Shallowmost
      if (k==1)then
        DZVZ0 = ( VZ(I,J,1) - 0.0_
      end if
      if (k==2)then
        DZVZ0 = ( VZ(I,J,2) - VZ(
      end if
    end if
    if( idz == KP-1 ) then
      if (k==NZP)then
        DZVZ0 = ( VZ(I,J,NZP) - V
      end if
    end if
```

Explicit time expansion by leap-frog scheme

```
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
```

☺B/F ratio is reduced to 0.4

☹IF sentences inside – it is difficult to optimize code by compiler.

Process of model boundary.

Variant2 (IF-free)

Stress tensor of Sxx, Syy, Szz

```
!$omp parallel do   private(i,j,k,RL1,RM1,・・・)
do k_j=1, (NZ01-NZ00+1)*(NY01-NY00+1)
  k=(k_j-1)/(NY01-NY00+1)+NZ00
  i=mod((k_j-1),(NY01-NY00+1))+NY00
  NX00, NX01
    = LAM (I,J,K); RM1   = RIG (I,J,K);
  2   = RM1 + RM1; RLRM2 = RL1+RM2
  rder diff (DXVX,DYVY,DZVZ)
  VX0 = (VX(I,J,K)  -VX(I-1,J,K))*C40/dx -  (VX(I+1,J,K)-VX(I-2,J,K))*C41/dx
  VY0 = (VY(I,J,K)  -VY(I,J-1,K))*C40/dy  - (VY(I,J+1,K)-VY(I,J-2,K))*C41/dy
  DZVZ0 = (VZ(I,J,K)  -VZ(I,J,K-1))*C40/dz  - (VZ(I,J,K+1)-VZ(I,J,K-2))*C41/dz
  DXVX1 = DXVX0; DYVY1 = DYVY0;
  DZVZ1 = DZVZ0;
  D3V3  = DXVX1 + DYVY1 + DZVZ1;
  SXX (I,J,K) = SXX (I,J,K) + (RLRM2*(D3V3)-RM2* (DZVZ1+DYVY1) ) * DT
  SYY (I,J,K) = SYY (I,J,K) + (RLRM2*(D3V3)-RM2* (DXVX1+DZVZ1) ) * DT
  SZZ (I,J,K) = SZZ (I,J,K)  + (RLRM2*(D3V3)-RM2* (DXVX1+DYVY1) ) * DT
  nd do
nd do
!$omp end parallel do
```

Fourth-order accurate central-difference scheme for velocity.

Explicit time expansion by leap-frog scheme.

☺Win-win between B/F ratio and optimization by compiler.

Variant2 (IF-free)

Loop for process of model boundary

```
! 2nd replace
if( is_fs .or. is_nearfs ) then
!$omp parallel do private(i,j,k,RL1,RM
do i=NX00,NX01
   do j=NY00,NY01
      do k = KFSZ(i,j)-1, KFSZ(i,j)+1, 2
         RL1   = LAM (I,J,K); RM1   = RIG
         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
         DYVX0 = (VX(I,J+1,K)-VX(I,J,K)  )*C40/dy &
             - (VX(I,J+2,K)-VX(I,J-1,K))*C41/dy
         DXVY0 = (VY(I+1,J,K)-VY(I  ,J,K))*C40/dx &
             - (VY(I+2,J,K)-VY(I-1,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
         DXVZ0 = (VZ(I+1,J,K)-VZ(I  ,J,K))*C40/dx &
             - (VZ(I+2,J,K)-VZ(I-1,J,K))*C41/dx
         DYVZ0 = (VZ(I,J+1,K)-VZ(I,J,K)  )*C40/dy &
             - (VZ(I,J+2,K)-VZ(I,J-1,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
```

Process of model boundary.

Fourth-order accurate central-difference scheme for velocity

Explicit time expansion by leap-frog scheme.

```
vel derive
 if (K==KFSZ(I,J)+1) then
    DZVX0 = ( VX(I,J,KFSZ(I,J)+2)-VX(I,J,KFSZ(I,J)+1) )/ DZ
    DZVY0 = ( VY(I,J,KFSZ(I,J)+2)-VY(I,J,KFSZ(I,J)+1) )/ DZ
    else if (K==KFSZ(I,J)-1) then
       DZVX0 = ( VX(I,J,KFSZ(I,J)  )-VX(I,J,KFSZ(I,J)-1) )/ DZ
       DZVY0 = ( VY(I,J,KFSZ(I,J)  )-VY(I,J,KFSZ(I,J)-1) )/ DZ
    end if
    DXVX1 = DXVX0
    DYVY1 = DYVY0
    DZVZ1 = DZVZ0
    D3V3  = DXVX1 + DYVY1 + DZVZ1
    DXVYDYVX1 = DXVY0+DYVX0
    DXVZDZVX1 = DXVZ0+DZVX0
    DYVZDZVY1 = DYVZ0+DZVY0
    if (K==KFSZ(I,J)+1)then
       KK=2
    else
       KK=1
    end if
    SXX (I,J,K) = SSXX (I,J,KK) &
        + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
    SYY (I,J,K) = SSYY (I,J,KK) &
        + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
    SZZ (I,J,K) = SSZZ (I,J,KK) &
        + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1) ) * DT
    SXY (I,J,K) = SSXY (I,J,KK) + RM1 * DXVYDYVX1 * DT
    SXZ (I,J,K) = SSXZ (I,J,KK) + RM1 * DXVZDZVX1 * DT
    SYZ (I,J,K) = SSYZ (I,J,KK) + RM1 * DYVZDZVY1 * DT
end do
d do
do
!$omp end parallel do
```

Select節によるコード選択と階層的なAT指定

上位のAT指定

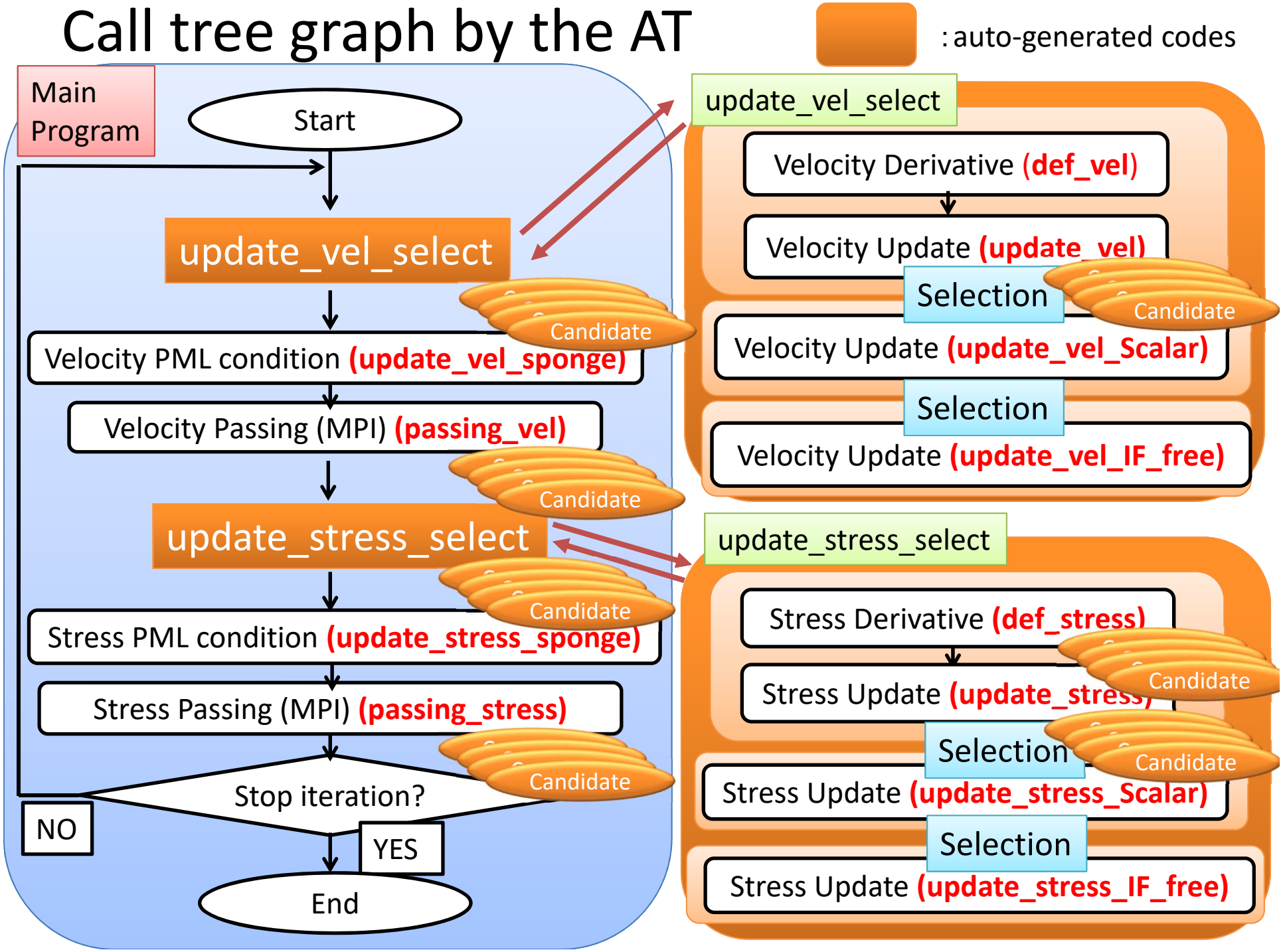
```
Program main
....
!OAT$ install select region start
!OAT$ name ppohFDMupdate_vel_select
!OAT$ select sub region start
call ppohFDM_pdiffx3_p4( SXX,DXSXX,NXP,NYP,NZP,....)
call ppohFDM_pdiffy3_p4( SYY,DYSYY, NXP,NYP,NZP,.....)
...
if( is_fs .or. is_nearfs ) then
  call ppohFDM_bc_stress_deriv( KFSZ,NIFS,NJFS,
end if
call ppohFDM_update_vel    ( 1, NXP, 1, NYP, 1, N
!OAT$ select sub region start
!OAT$ select sub region start
call ppohFDM_update_vel_Intel  ( 1, NXP, 1, NYP,
!OAT$ select sub region start
!OAT$ install select region end
```

下位のAT指定

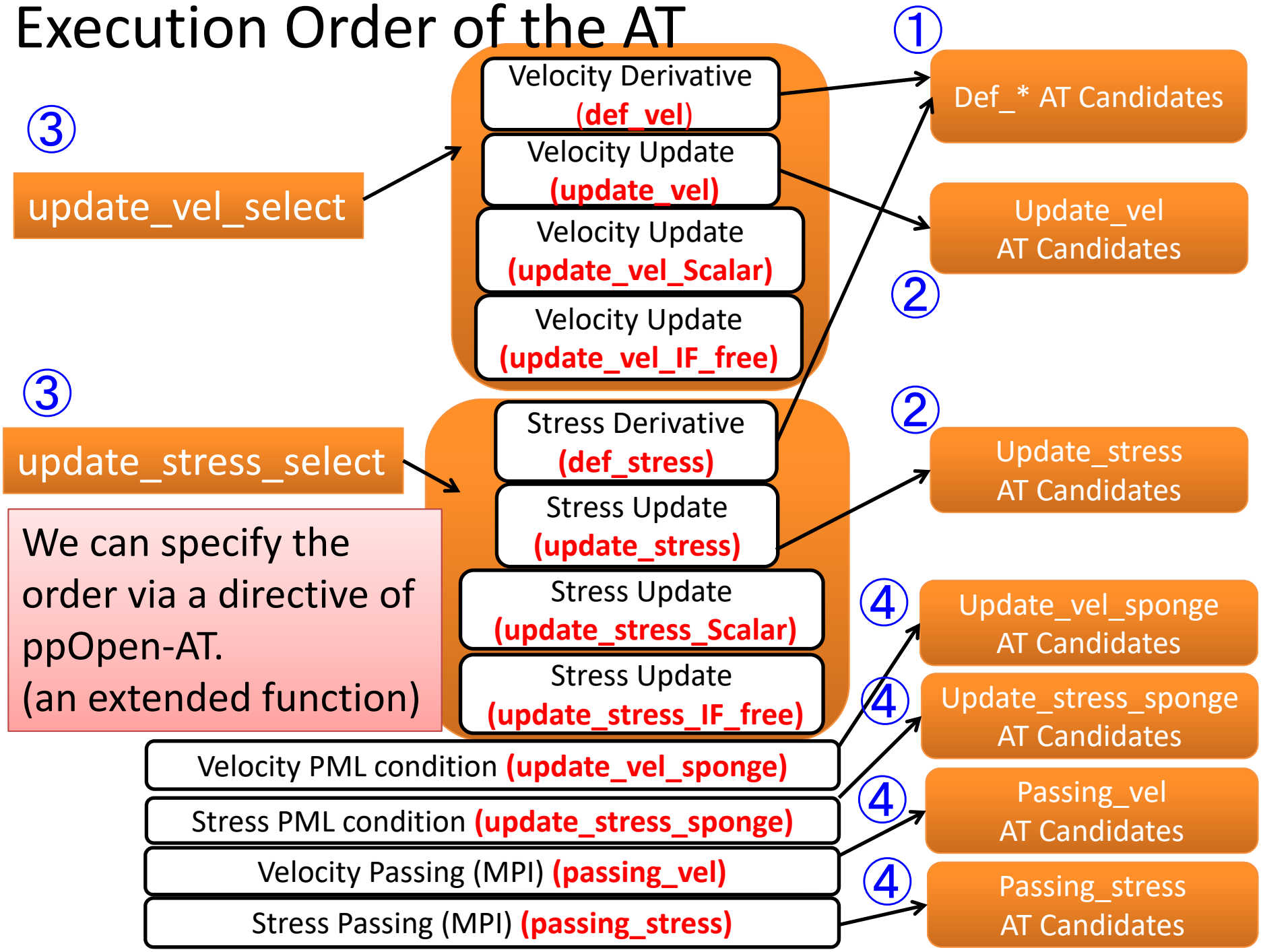
```
subroutine ppohFDM_pdiffx3_p4(....)
....
!OAT$ install LoopFusion region start
....
```

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

Call tree graph by the AT



Execution Order of the AT



We can specify the order via a directive of ppOpen-AT.
(an extended function)

AT順序指定のためのディレクティブ拡張

- 補助指定子
- C言語
  #pragma exec order <*x*>
- Fortran言語
  !OAT$ exec order <*x*>
- 例

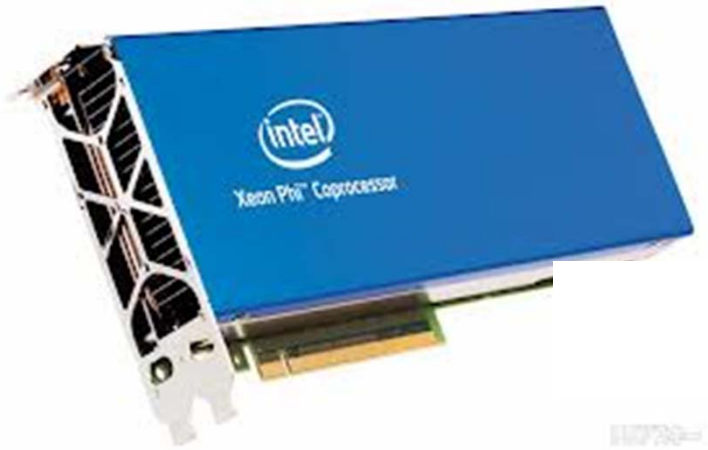
```
subroutine ppohFDM_update_vel(....)
....
!OAT$ install LoopFusion region start
!OAT$ name ppohFDMupdate_vel
!OAT$ exec order 2
!$omp parallel do private(i,j,k,ROX,ROY,ROZ)
do k = NZ00, NZ01
  do j = NY00, NY01
    do i = NX00, NX01
  .....
```

同一順序内での
AT実行順番は任意
（デフォルトでは、
プリプロセッサが
見つけた順番）

性能評価

Machine Environment
(8 nodes of the Xeon Phi)

- The Intel Xeon Phi
  - Xeon Phi 5110P (1.053 GHz), 60 cores
  - Memory Amount : 8 GB (GDDR5)
  - Theoretical Peak Performance : 1.01 TFLOPS
  - One board per node of the Xeon phi cluster
  - InfiniBand FDR x 2 Ports
    - Mellanox Connect-IB
    - PCI-E Gen3 x16
    - 56Gbps x 2
    - Theoretical Peak bandwidth 13.6GB/s
    - Full-Bisection
  - Intel MPI
    - Based on MPICH2, MVAPICH2
    - Version 5.0 Update 3 Build 20150128
  - Compiler : Intel Fortran version 15.0.3 20150407
  - Compiler Options :
    -ipo20 -O3 -warn all -openmp -mcmodel=medium -shared-intel –mmic
    -align array64byte
  - KMP_AFFINITY=granularity=fine, balanced (Uniform Distribution of threads between sockets)

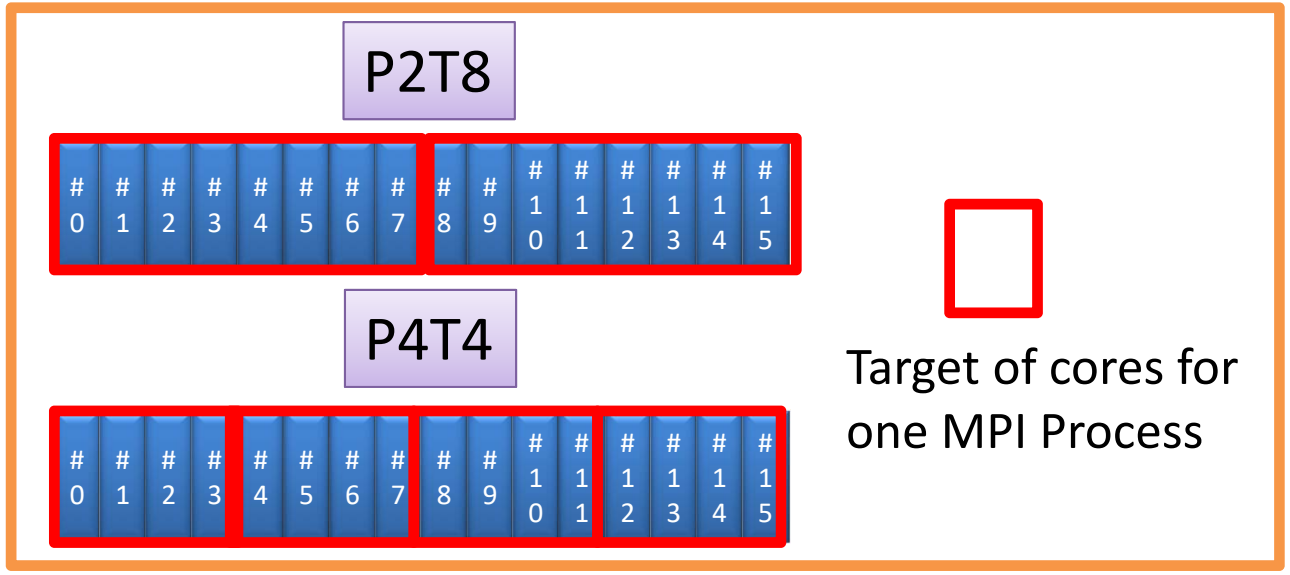


Execution Details

- ppOpen-APPL/FDM ver.0.2
- ppOpen-AT ver.0.2
- The number of time step: 2000 steps
- The number of nodes: 8 node
- Native Mode Execution
- Target Problem Size
  (Almost maximum size with 8 GB/node)
  - NX * NY * NZ = 512 x 512 x 512 / 8 Node
  - NX * NY * NZ = 256 * 256 * 256 / node
    (!= per MPI Process)
- The number of iterations for kernels
  to do auto-tuning: 100

Execution Details of Hybrid MPI/OpenMP

- Target MPI Processes and OMP Threads on the Xeon Phi
  - The Xeon Phi with 4 HT (Hyper Threading)
  - P*X* T*Y*: *X* MPI Processes and *Y* Threads per process
  - P8T240 : Minimum Hybrid MPI/OpenMP execution for ppOpen-APPL/FDM, since it needs minimum 8 MPI Processes.
  - P16T120
  - P32T60
  - P64T30
  - P128T15
  - P240T8
  - P480T4



  - Less than P960T2 cause an MPI error in this environment.

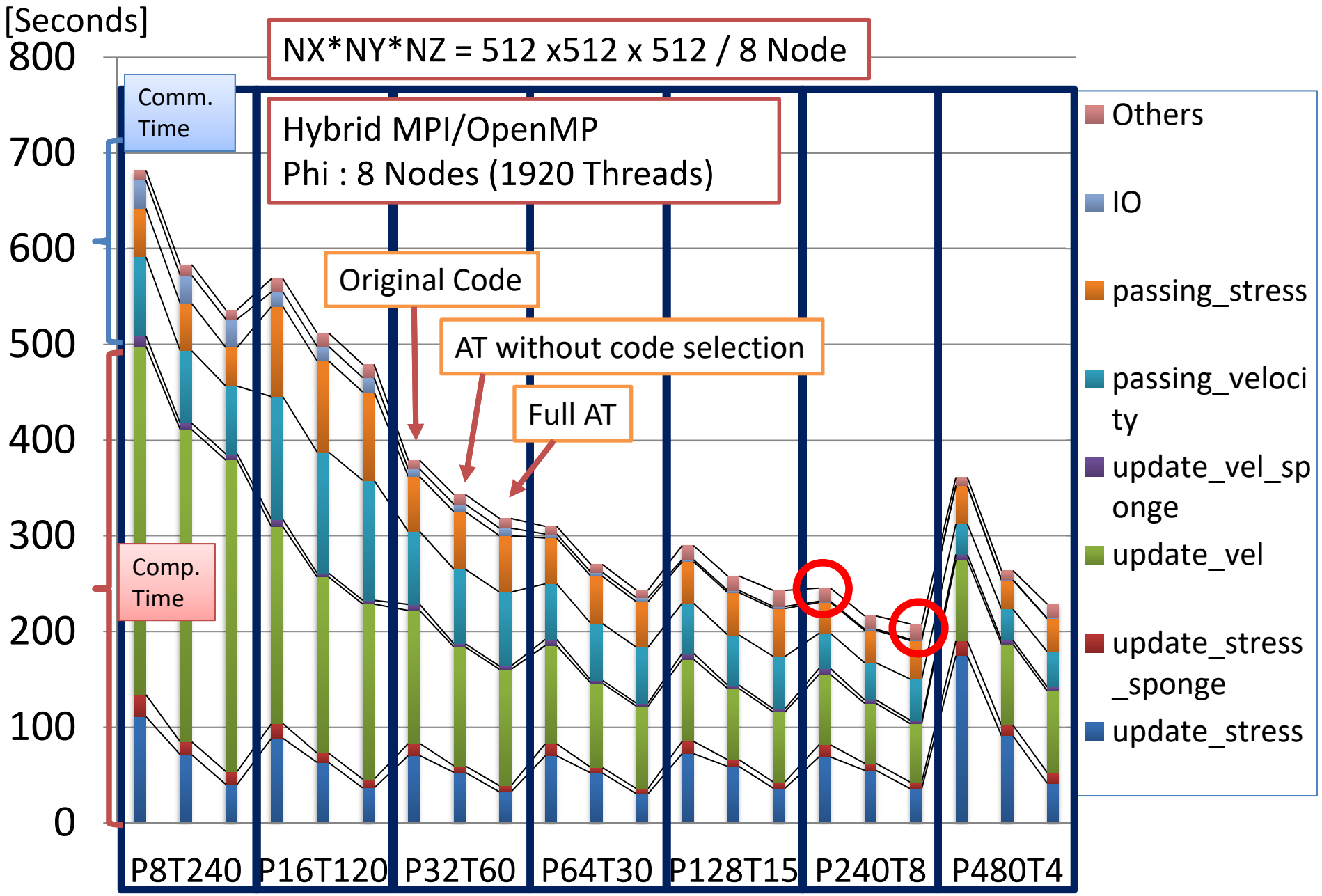
AT候補の組合わせ数(ppOpen-APPL/FDM)

カーネル名	対象	組合わせ数
1. update_stress	・ループ融合とループ分割：8種 ・コード選択2種	**10**
2. update_vel	・ループ融合、ループ分割、 演算順序変更：6種 ・コード選択2種	**8**
3. update_stress_sponge	・ループ融合：3種	**3**
4. update_vel_sponge	・ループ融合：3種	**3**
5. ppohFDM_pdiffx3_p4	カーネル：def_update、def_vel ・ループ融合：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	データバッキング、アンパッキング ・ループ融合：3種	**3**
12. ppohFDM_ps_unpack		**3**
13. ppohFDM_pv_pack		**3**
14. ppohFDM_pv_unpack		**3**

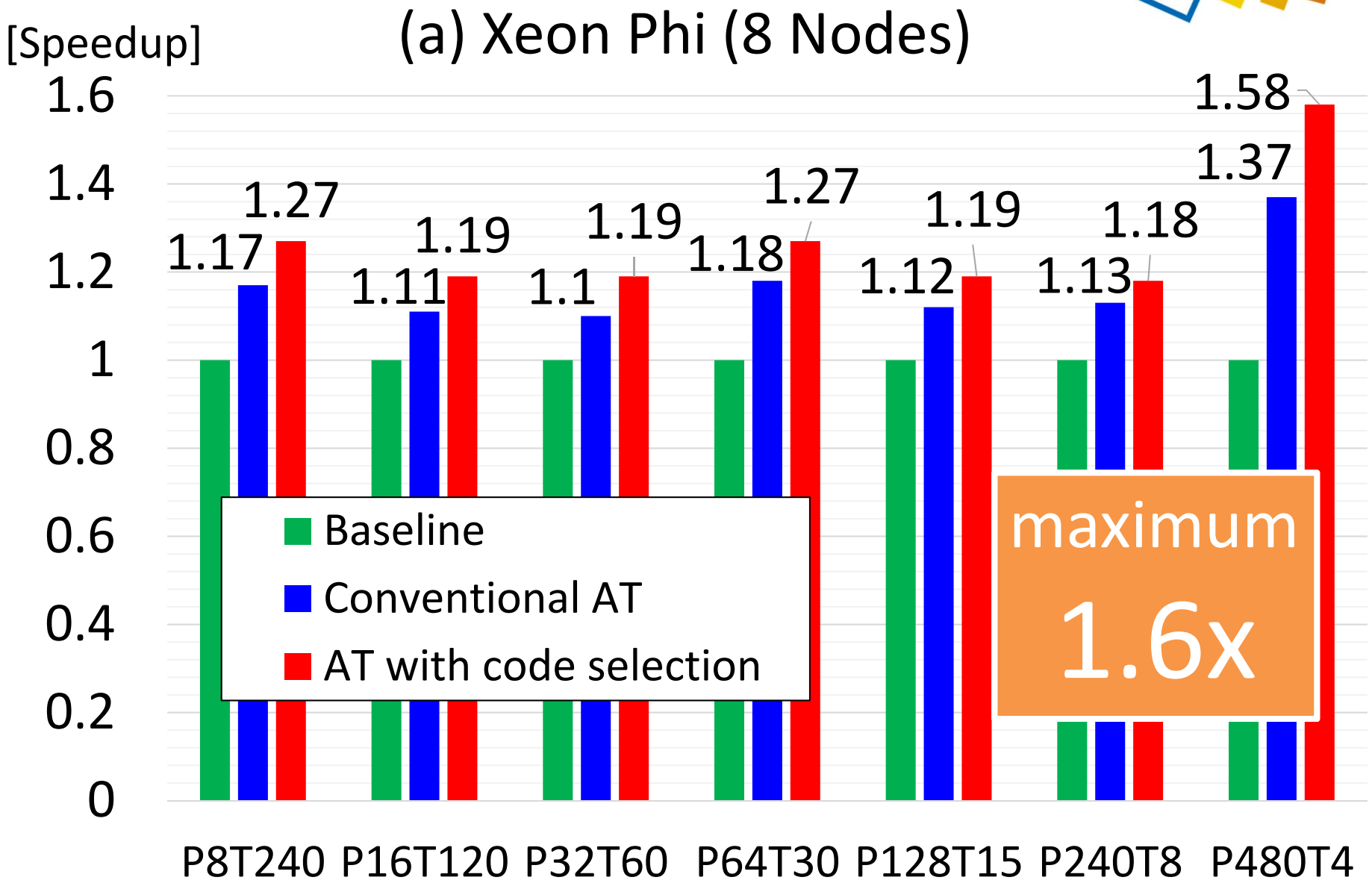
- 合計：54種
- ハイブリッド MPI/OpenMP 実行:7種
- 54 × 7 = 378種

BREAK DOWN OF TIMINGS

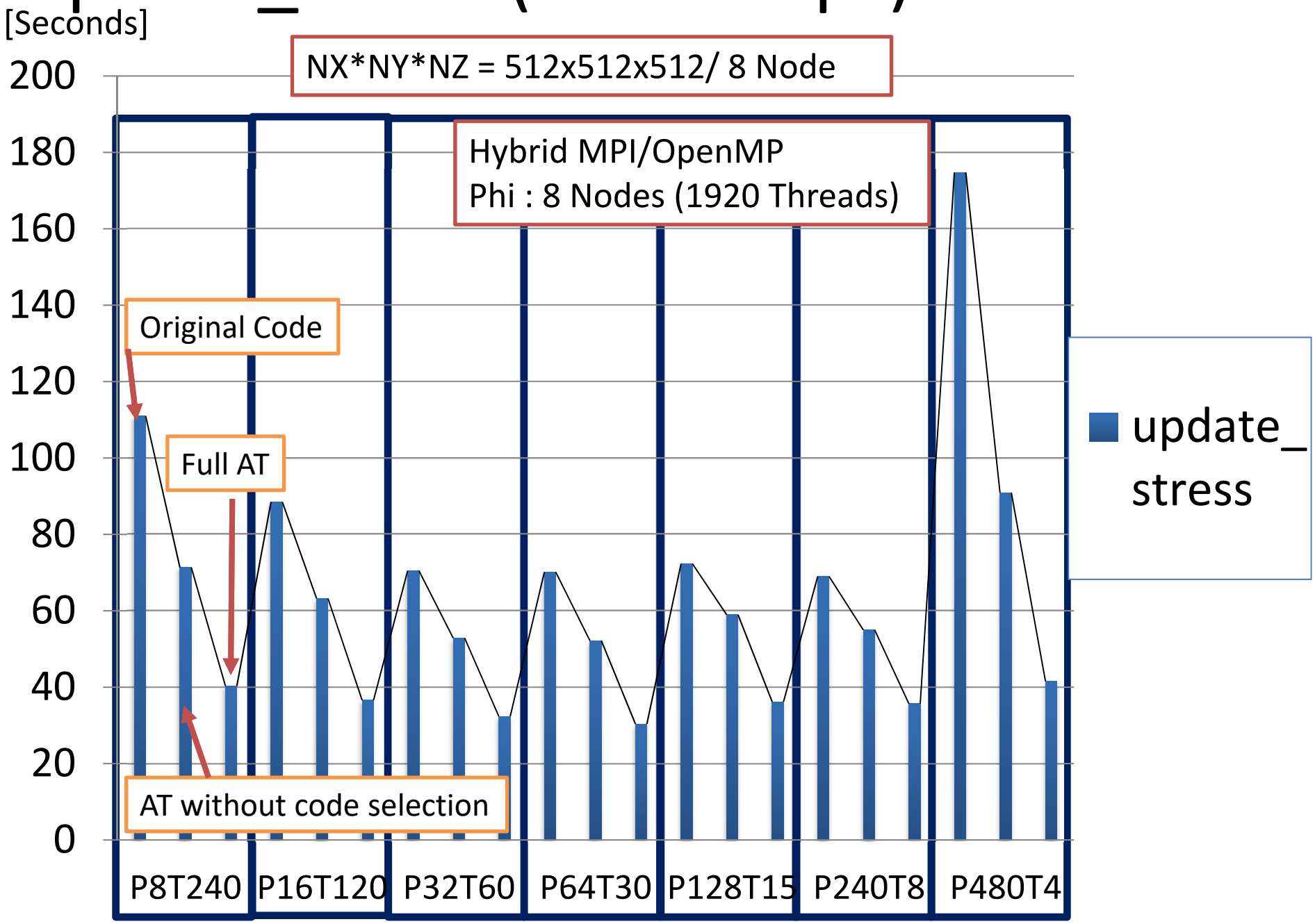
Whole Time (2000 time steps)



Speedups (Whole Time)

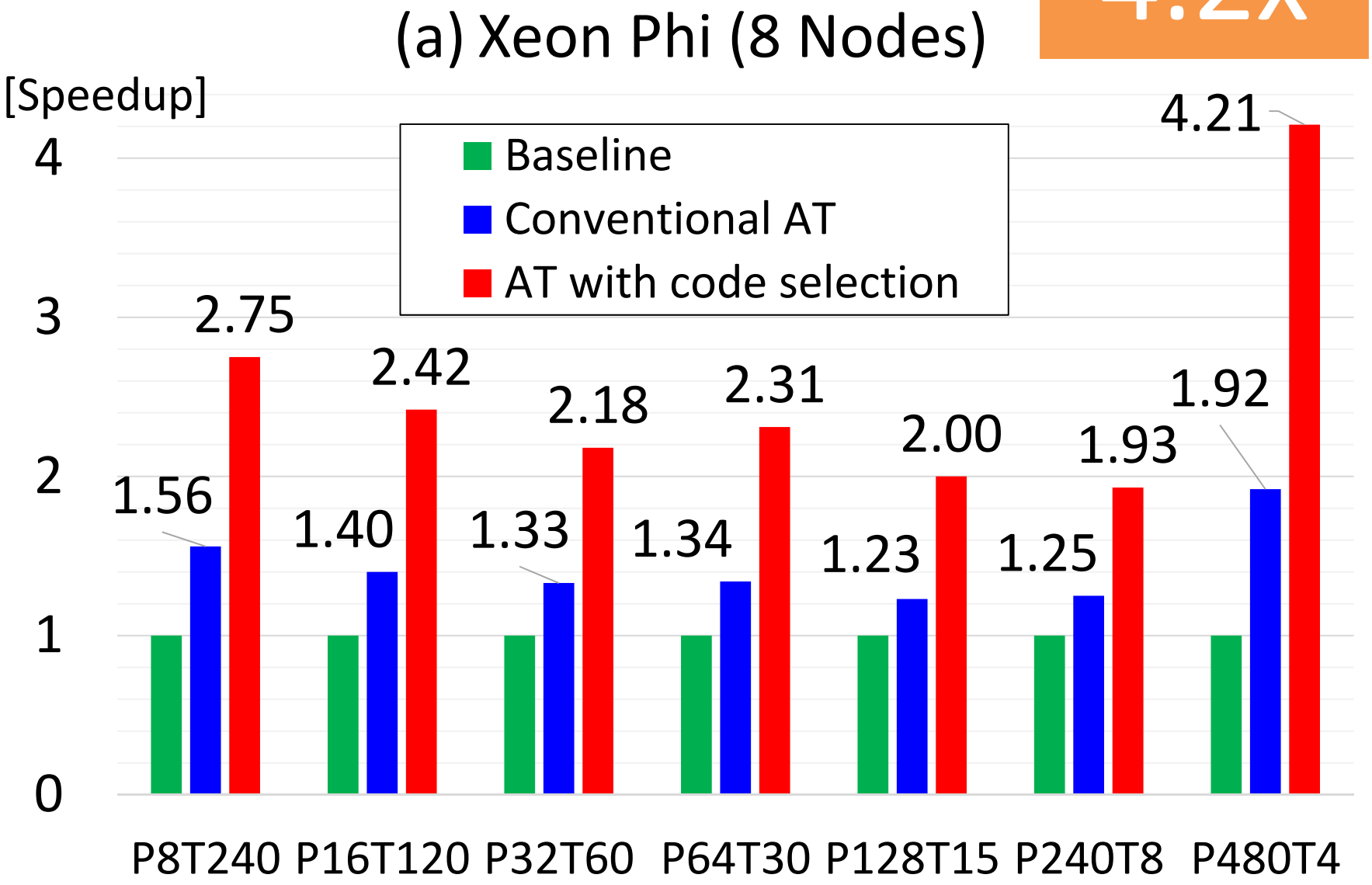


update_stress (2000 steps)



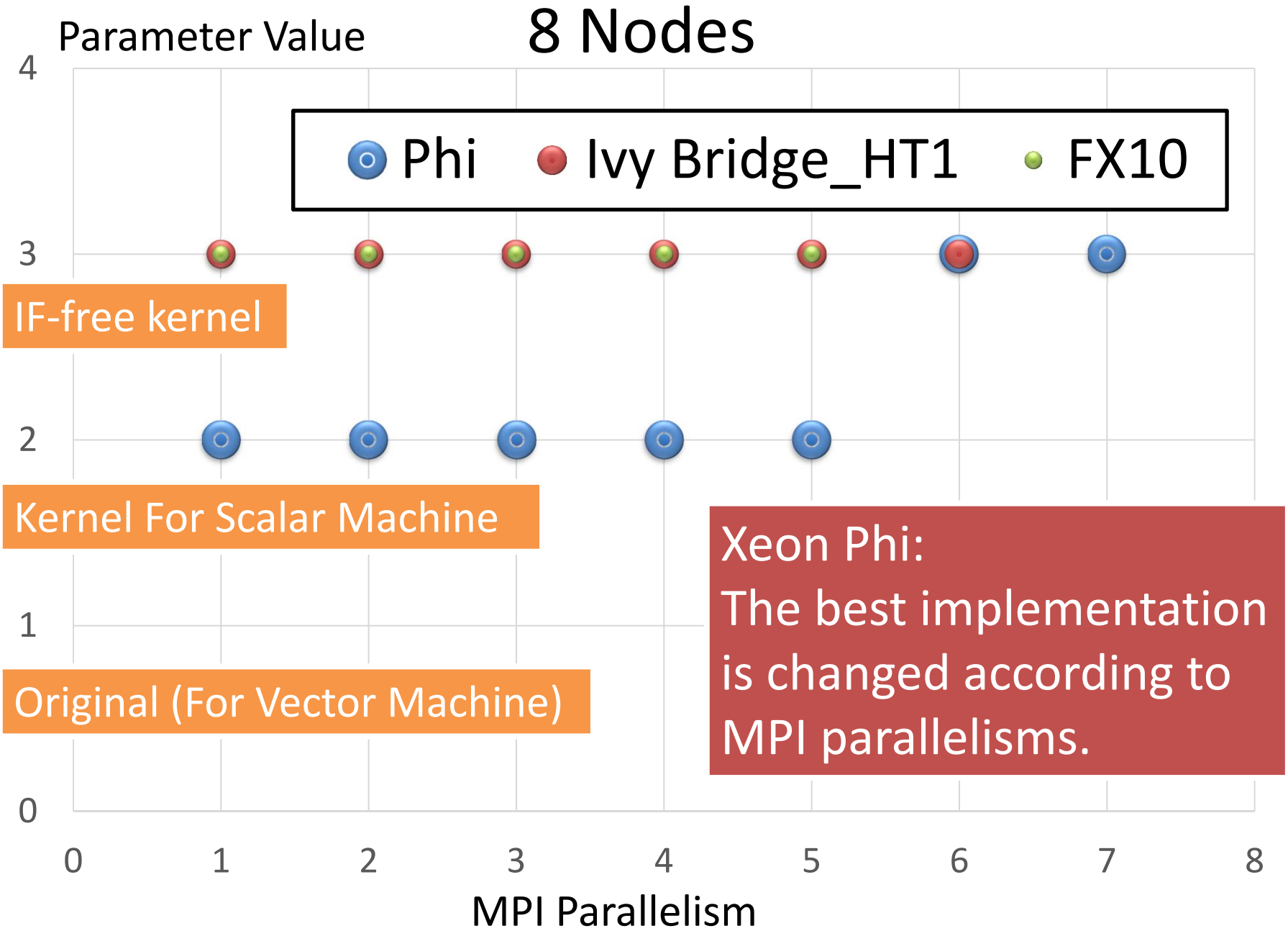
Speedups（update_stress）

maximum
4.2x



複数の計算機環境での評価

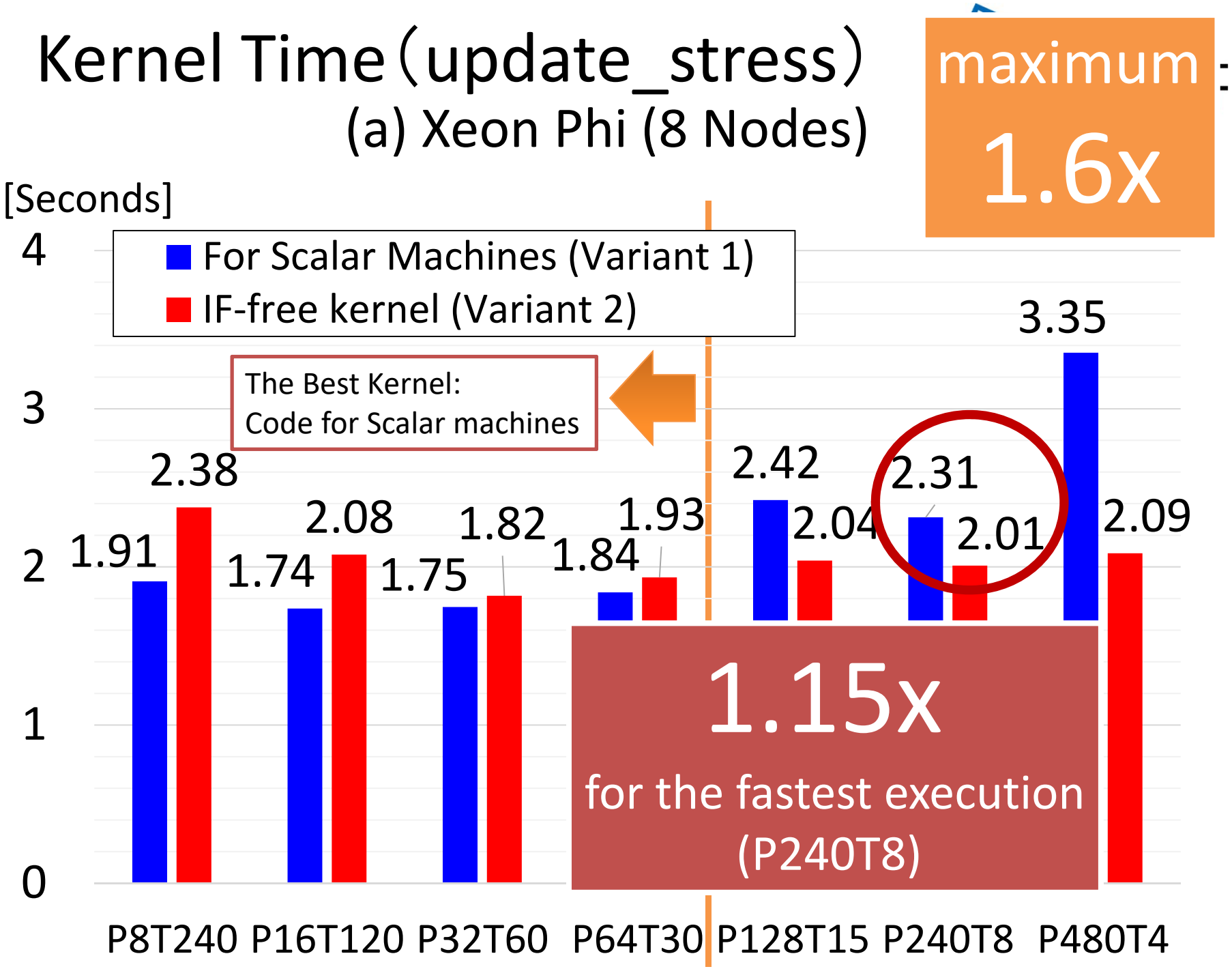
Parameter Distribution (update_stress_select)



Xeon Phi:
The best implementation is changed according to MPI parallelisms.

Kernel Time（update_stress）

(a) Xeon Phi (8 Nodes)



今後の展望

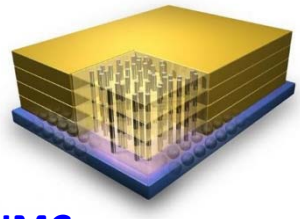
ポストムーア時代のソフトウェア自動チューニングに向けて

:参考文献[V]

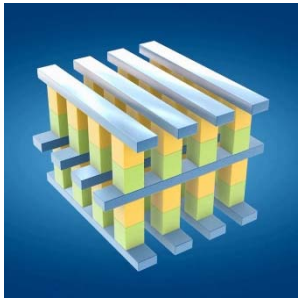
ポスト・ムーア時代のAT

- 2020年頃のエクサスケールスパコンを境に
  ムーアの法則での速度向上が終焉
  - “One-time Speedup”のメニーコア化、
    配線微細化、による低電力化の終わり
  - →ノード内の計算性能（FLOPS）は増加しなくなる
- チップ内のメモリ量やメモリ帯域は、
  「3次元積層技術」により向上
- 3次元積層メモリ：
  - Z方向（積層方向）のバンド幅は増加、
    低いアクセスレイテンシの達成（→高性能化）
  - X-Y方向は、アクセスレイテンシは維持
- ノード間は、アクセスレイテンシが悪化、だが
  光配線技術などでバンド幅は増加できる
- 演算量ではなくデータ移動量の増加を主体に
  （アルゴリズムの変更が求められる！）
- 以上は2020年後半（→ポスト・ムーア）

・局所メモリ（キャッシュ量）増加
・メモリアクセスレイテンシが非均一な、「非均質・階層メモリ」に対するアルゴリズムやコード最適化技術



HMC
Source:
http://www.engadget.com/2011/12/06/the-big-memory-cube-gamble-ibm-and-micron-stack-their-chips/



Intel 3D Xpoint
Source:
http://www.theregister.co.uk/2015/07/28/intel_micron_3d_xpoint/

AT技術の対応

- 階層型AT方式
- 通信削減アルゴリズム
- 大きいキャッシュ活用のアルゴリズム
  - 古典アルゴリズムの復活？
  - キャッシュブロック化なし数値アルゴリズム
  - 陽解法から陰解法（連立一次方程式解法）へ
  - アウト・オブ・コア・アルゴリズム（主メモリ外）

機械学習の適用

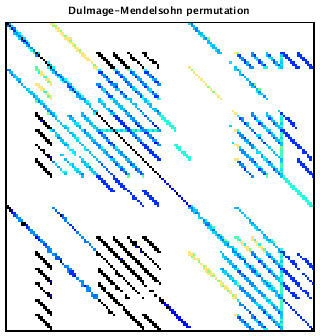
国内の研究動向

- 根拠に基づく性能チューニングの支援
  - 理研AICS 橋本政朋 研究員
  - 戦略アプリのチューニングログを利用
  - 機械学習で、チューニングパターンを予測
    - 「このカーネルコード」の「ここをループ分割」すると、「京コンピュータ」で速くなる確率は60%
  - 以下の課題があるが注目される研究の１つ：
    1. チューニングデータベースの充実
       - 通常、失敗したチューニングのコード履歴は残さない
       - 網羅的なチューニングパターンがいる（自動化が必要）
    2. 計算機環境データの充実
       - CPU種別、コア数、問題サイズなど、多種の実行条件のデータが必要

AT言語で複数のAT候補の実行時間を自動測定可能。
自動でデータベース化し、機械学習への展開に期待

数値計算分野で求められる機械学習（疎行列ソルバーの場合）参考文献[IV]

(Matrices Source: https://www.cise.ufl.edu/research/sparse/matrices/)



未知の行列

予測
最適候補

疎行列形状

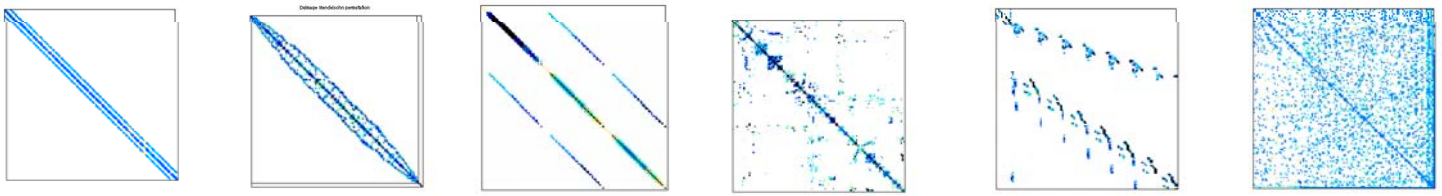
前処理

数値解法

実装／データ構造

学習セット

疎行列形状



前処理	None	Jacobi	SSOR	ILU(0)	ILUT	ILUT
数値解法	IDR	GCR	BiCGStab	BiCGStab	GMRES	GMRES
実装／データ構造	DIAG	CRS	ELL	Hyb. CRS	COO	CRS

話の流れ

第1部：背景

1. ソフトウェア自動チューニングとは
2. 自動チューニング記述言語
3. 性能モデル化とAT
4. 適用事例：有限差分法コード
5. 性能評価
6. 今後の展望

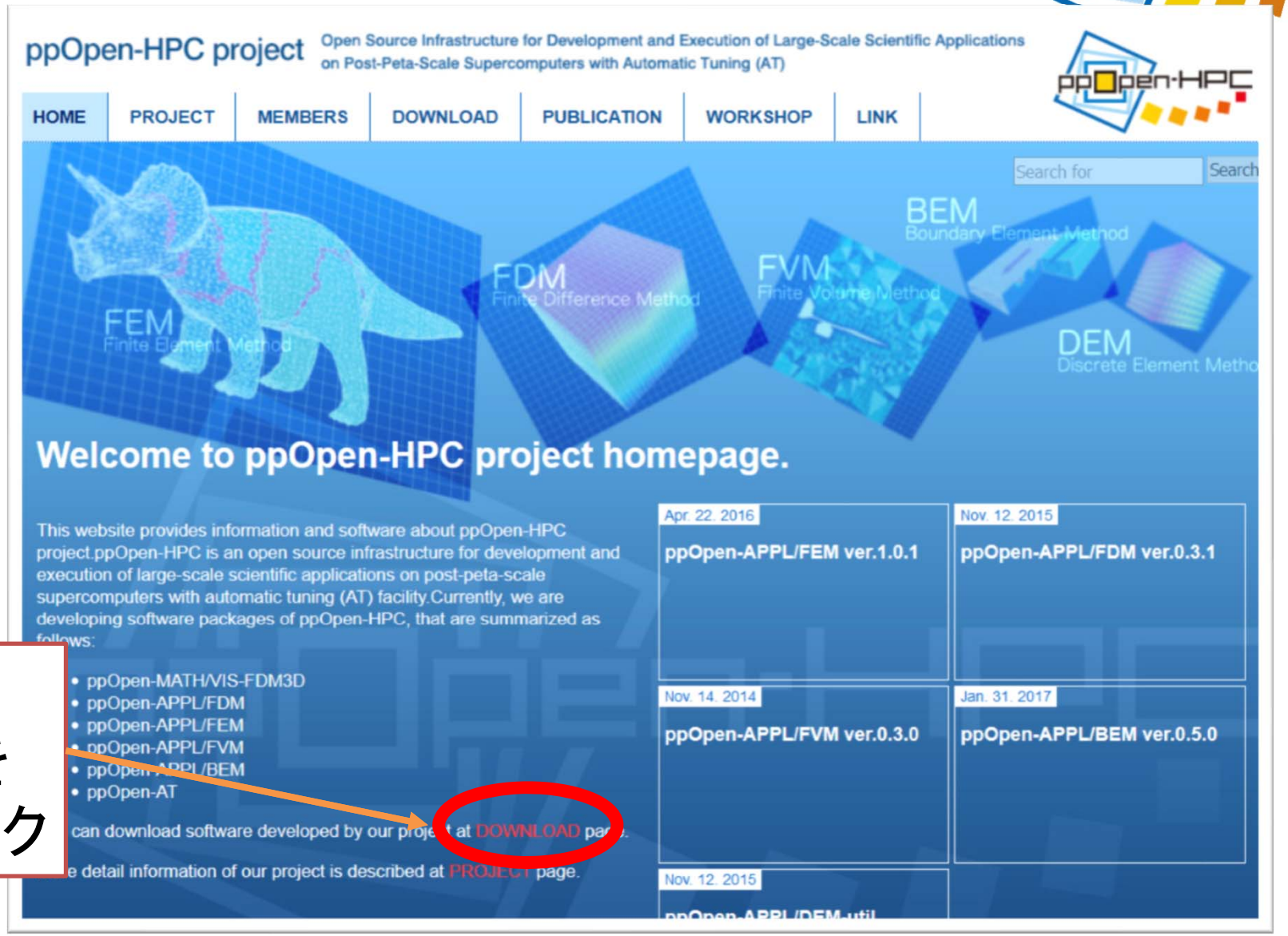
第2部：インストールと実行

1. ppOpen-ATのインストール
2. コンパイルと実行
3. 質疑応答

ppOpen-ATのインストール（1/5）

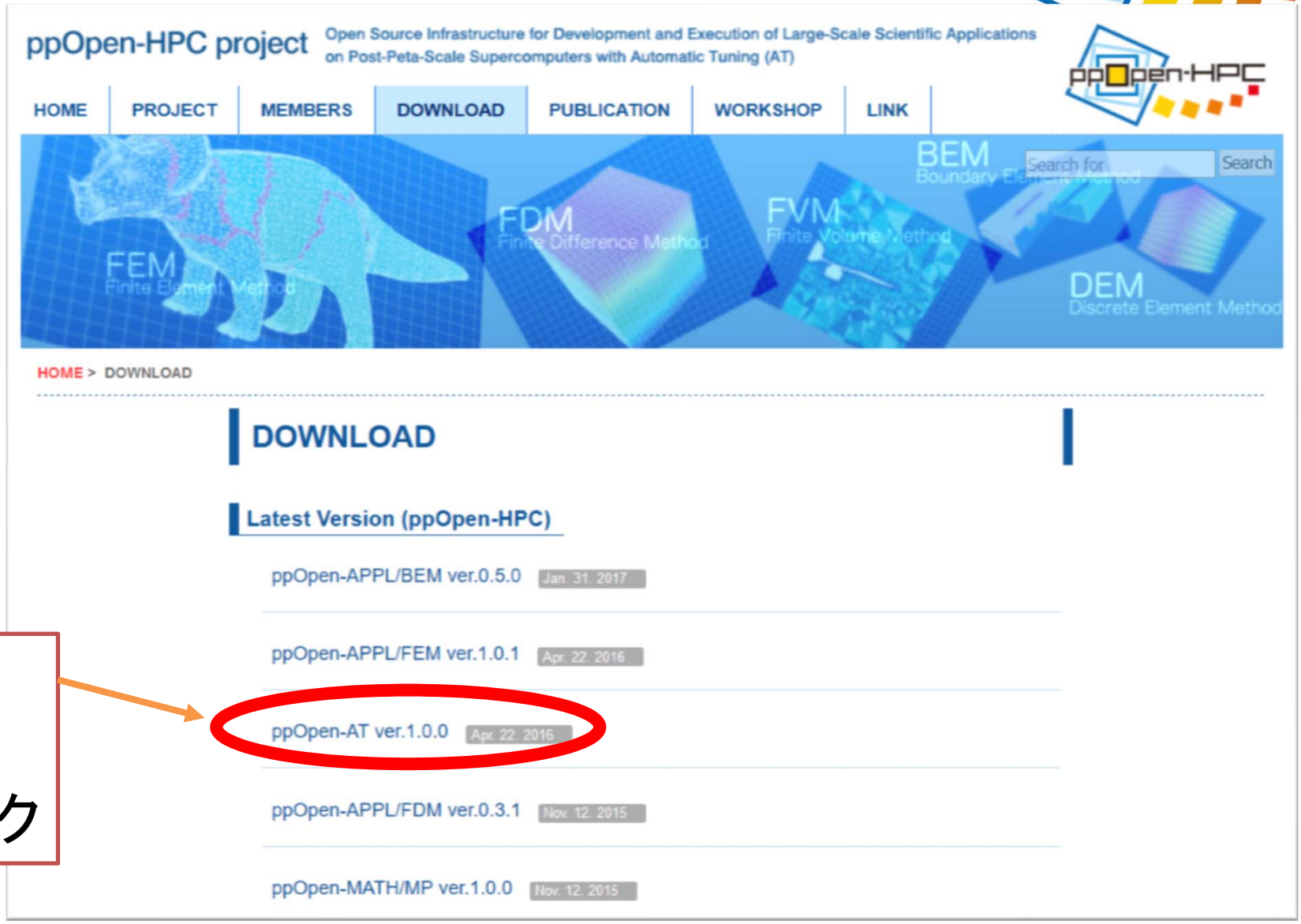
1. ppOpen-HPCのサイトに行く
   http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/
2. DOWNLOADリンクをクリック
3. Latest Version (ppOpen-HPC)
   ppOpen-AT ver.1.0.0 (Apr. 22. 2016)
   をクリック
4. 必要事項を入力
5. ppOpen-ATのリンクをクリックしてダウンロード

ppOpen-ATのインストール（2/5）



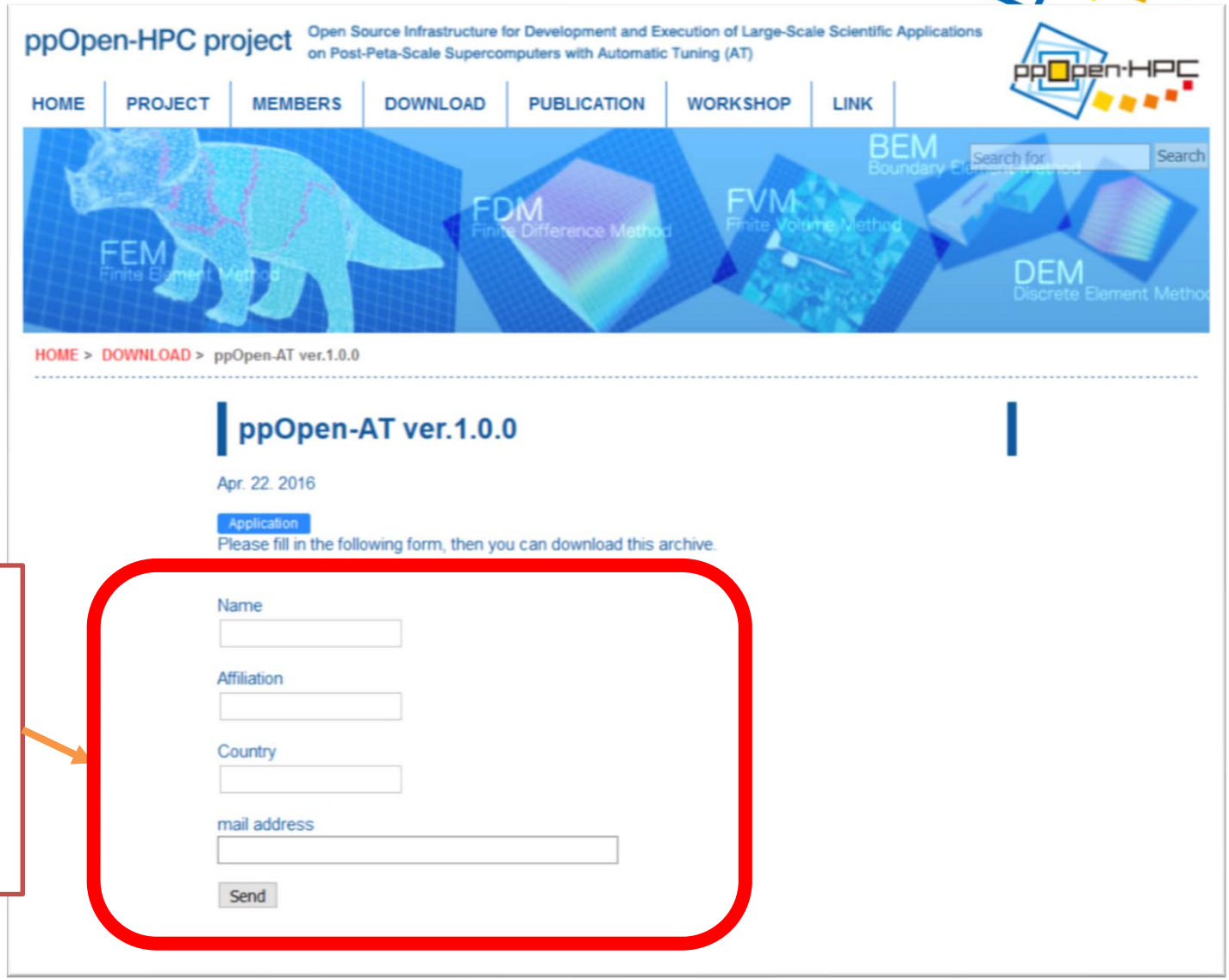
②
ここを
クリック

ppOpen-ATのインストール（3/5）



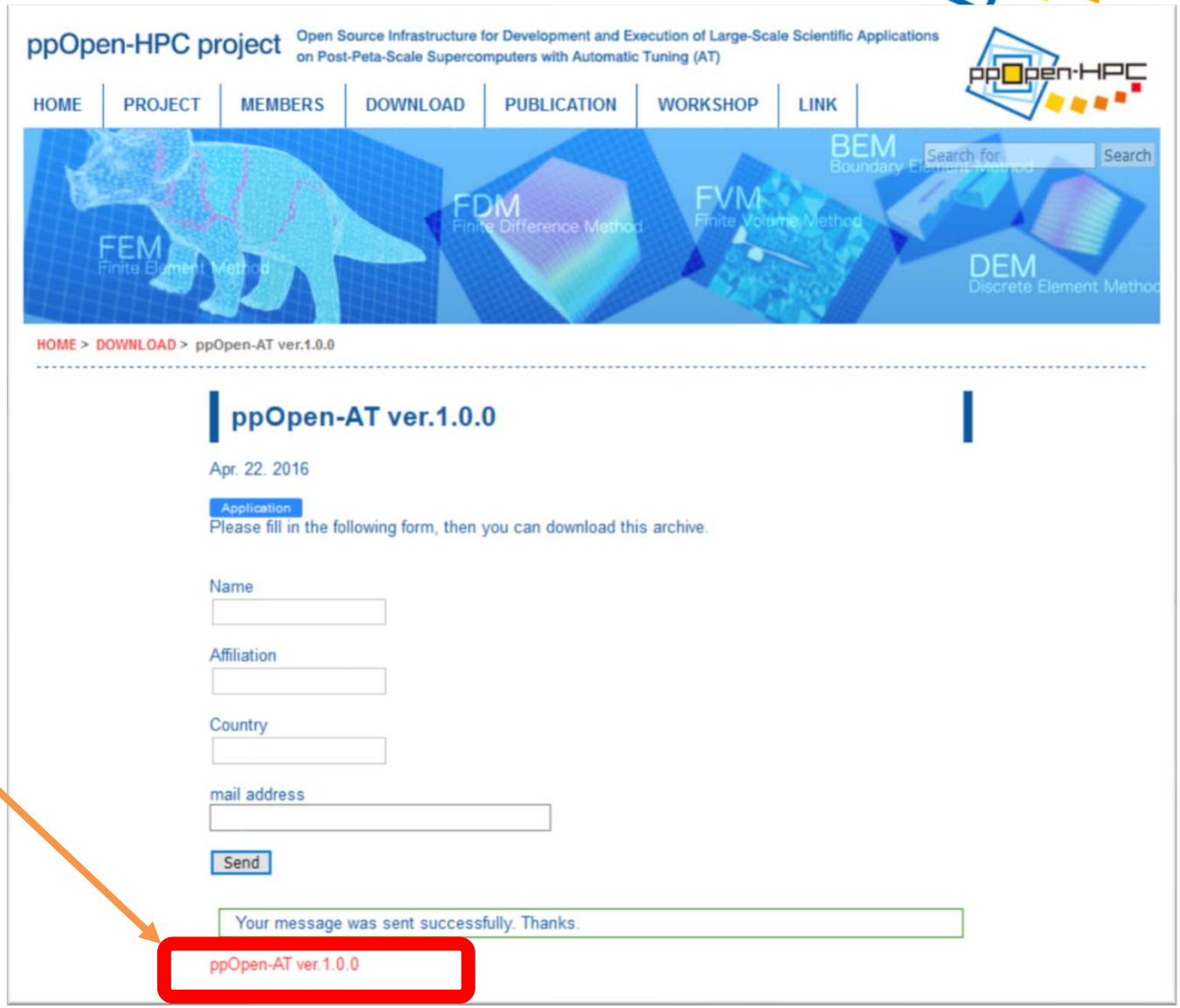
③
ここを
クリック

ppOpen-ATのインストール（4/5）



④
必要事項を
入力して
Send

ppOpen-ATのインストール（5/5）



⑤
クリックして
ダウンロード

コードの展開とコンパイル

1. ppohAT_1.0.0.tar.gz
   を、カレントディレクトリに置く
2. tarコマンドで展開
   $ tar xvf ppohAT_1.0.0.tar.gz
3. ppohAT_1.0.0ディレクトリに移動
   $ cd ppohAT_1.0.0
4. Makefileを自分の環境に書きなおす
5. makeを実行する
   $make

コンパイルログ

```
(cd src ; make )
make[1]: ディレクトリ '.../ppohAT_1.0.0/src' に入ります
gcc -I../include  -lm -lstdc++ -c main.cpp pass1.cpp pass2.cpp
pass3.cpp pass4.cpp pass5.cpp TuneRegion.cpp VCL_Lib.cpp
Visual.cpp
echo "Linking oat ..."
Linking oat ...
gcc -I../include  main.o pass1.o pass2.o pass3.o pass4.o pass5.o
TuneRegion.o VCL_Lib.o Visual.o -lm -lstdc++ -o oat
echo "done"
done
cp -f oat ../bin/
make[1]: ディレクトリ '.../ppohAT_1.0.0/src' から出ます
```

ディレクトリ構造

ppohAT_1.0.0/

- bin/　実行可能コードを入れる場所
- doc/　ppohAT ユーザマニュアルの場所
- etc/　その他のファイルの集積場所
  （例えばMakefile.in.*）
- examples/　コード自動生成のサンプルプログラムの集積場所
- include/　外部利用のためのヘッダファイルの集積場所
- lib/　外部利用のためのlibファイルの集積場所
  （将来拡張用）
- src/　ppohATのソースコード集積場所
- tools/　ppohATツールの集積場所（将来拡張用）

梱包されているMakefileの雛形

- ./etc/Makefile.in.hitachi
  : HITACHI C++ コンパイラ用
- ./etc/Makefile.in.fujitsu
  ： Fujitsu C++ コンパイラ用
- ./etc/Makefile.in.intel
  : Intel C++ コンパイラ用
- ./etc/Makefile.in.pgi
  : PGI C++ コンパイラ用

デフォルトは gcc

Make のコマンド

- all - 全てのパッケージを作る
- src - example と tools以外のパッケージを作る
- clean - ビルドに必要なファイルを消す
- distclean – ビルドに必要なファイルと
  実行オブジェクトを消す
- install - 実行可能ファイル、ヘッダファイル、
  およびライブラリを、ディレクトリ <PREFIX> に
  コピーする
- uninstall – すべてのインストールファイルとディレクトリを消す
- 例）

$ make all

Exampleフォルダの中身: C （1/2）

- C フォルダ内 （C言語用サンプルプログラム）
- matunroll
  :行列-行列積のループアンローリング
- matunrolldyn
  :行列-行列積のループアンローリング（実行時AT）
- Test_i_1_to64
  :行列-行列積のループアンローリング
  （1段～64段までのコードを自動生成）

Exampleフォルダの中身: C (2/2)

- C フォルダ内 （C言語用サンプルプログラム）
- matunroll
  : 行列-行列積のループアンローリング例
- matunrolldyn
  : 行列-行列積のループアンローリング例（実行時AT）
- Test_i_1_to64
  : 行列-行列積のループアンローリング（1段～64段までのコードを自動生成）

Exampleフォルダの中身: F90 (1/2)

- F90フォルダ内 （F90言語用サンプルプログラム）
- SetNum, SetDebug :MPIコード
  : MPIプロセス指定の例、debug指定子の例
- MGSker
  : 修正グラムシュミット法(Modified Gram-Schmidt法)の計算カーネルにループアンローリングを付加する例
- TDRker
  :Householder三重対角化の更新部の計算カーネルにループアンローリングを付加する例（MPIコード）
- TrdMatVec

  :Hoseholder三重対角化の行列-ベクトル積の計算カーネルにループアンローリングを付加する例（MPIコード）
- Fit_Sample1, Fit_Sample2

  : 行列-行列積のコードにループアンローリングを付加する場合において、性能モデルにd-Spline法を適用する例（MPIコード）

Exampleフォルダの中身: F90 (2/2)

- F90フォルダ内 （F90言語用サンプルプログラム）
- Dynamic_Set1, Dynamic_Set2
  : 行列-行列積、および疎行列-ベクトル積のループアンローリング例（実行時AT）（MPIコード）
- Test_Select_1, Test_Select_2, Test_Select_3
  : Select指定子の例（MPIコード）
- Test_i_1_to64, Test_k_1_to64, Test_j_1_to64, Test_ijk_1_to_4
  : 行列-行列積のループアンローリング例（MPIコード）

コード自動生成のデモ

行列-行列積のアンローリングの処理例（C言語）

1. サンプルプログラムの場所へ移動
   $ cd ~/ppohAT_1.0.0/examples/C/Test_i_1_to64
2. ファイルを確認する
   matmul.c : 対象のCコード
   OAT.h　:ppOpen-ATが生成するコードの実行に必要なヘッダファイルの雛形（必須）。必ず、対象のコードのフォルダ内に置くこと。
3. ppOpen-ATプリプロセッサを起動して、コードを自動生成させる

   $ ../../../bin/oat.exe matmul.c

行列-行列積のアンローリングの処理例（C言語）：実行ログ

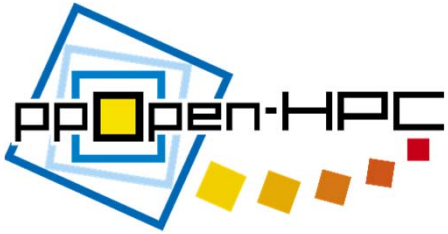
```
--- ppOpenAT version 1.0.0 ---
OAT_PATH = ""
CommandLine = ../../../bin/oat matmul.c
**** Start Code Generation ****
SrcFile ="matmul.c"
Pass1 Start
Pass2 Start
------ Val List ---------
0: (MIdx=0) Int iauto (RW=6,3) DefPos=16
1: (MIdx=0) Int n (RW=4,1) DefPos=20
2: (MIdx=0) Real A(0:0)(439888:440032) (RW=4,0) DefPos=24
3: (MIdx=0) Real N(0:0) (RW=6,0) DefPos=26
4: (MIdx=0) Real B(0:0)(444352:-2144040648) (RW=4,0) DefPos=32
5: (MIdx=0) Real C(0:0)(96:95) (RW=4,0) DefPos=40
6: (MIdx=1) Real A(6:12)(439752:887088) (RW=2,1)(Arg) DefPos=212
7: (MIdx=1) Real N(6:12) (RW=6,0)(Arg) DefPos=214
8: (MIdx=1) Real B(6:12)(441608:883408) (RW=2,0)(Arg) DefPos=221
9: (MIdx=1) Real C(0:0)(0:0) (RW=2,0)(Arg) DefPos=230
10: (MIdx=1) Int n (RW=4,0)(Arg) DefPos=239
11: (MIdx=1) Real da1 (RW=3,2) DefPos=245
12: (MIdx=1) Real da2 (RW=1,0) DefPos=247
13: (MIdx=1) Real dc (RW=2,1) DefPos=251
14: (MIdx=1) Int i (RW=5,1) DefPos=255
15: (MIdx=1) Int j (RW=5,1) DefPos=257
16: (MIdx=1) Int k (RW=5,1) DefPos=259
```

```
17: (MIdx=2) Real A(0:87)(441088:441232) (RW=1,1)(Arg) DefPos=435
18: (MIdx=2) Real N(23041:23041) (RW=6,0)(Arg) DefPos=437
19: (MIdx=2) Real B(0:0)(-1:4) (RW=1,1)(Arg) DefPos=444
20: (MIdx=2) Real C(0:6145)(3:435) (RW=1,1)(Arg) DefPos=453
21: (MIdx=2) Int n (RW=5,0)(Arg) DefPos=462
22: (MIdx=2) Int i (RW=7,2) DefPos=468
23: (MIdx=2) Int j (RW=7,2) DefPos=470
24: (MIdx=3) Real A(0:0)(67:134) (RW=2,0)(Arg) DefPos=605
25: (MIdx=3) Real N(0:0) (RW=6,0)(Arg) DefPos=607
26: (MIdx=3) Real B(26723:26723)(1936:1935) (RW=1,0)(Arg) DefPos=614
27: (MIdx=3) Real C(0:0)(-1:0) (RW=1,0)(Arg) DefPos=623
28: (MIdx=3) Int n (RW=3,0)(Arg) DefPos=632
29: (MIdx=3) Int i (RW=4,1) DefPos=643
30: (MIdx=3) Int j (RW=4,1) DefPos=645
Pass3 Start
Pass4 Start
OAT variable:      OAT_DEBUG = 1
OAT variable:      OAT_NUMPROCS = 4
OAT variable:      OAT_STARTTUNESIZE = 100
OAT variable:      OAT_ENDTUNESIZE = 500
OAT variable:      OAT_SAMPDIST = 100
Base Paramater (BPset): n
------ Tuning Region -- Count = 1
Tuning Region : 0
Install Unroll MyMatMul
OAT_InstallMyMatMul(,n,,,A,,,C,B)
   varied i (0,i<n,) from 1.000000 to 64.000000 step 1.000000
  RefValStr = ( i,da1, )
  Case Count = 64
Pass5 Start
TuneRegion = 1/1
tune 0
**** End Code Generation **** OK !
```

行列-行列積のアンローリングの処理例（C言語）：自動生成コード

- ppOpen-ATでは、実行フォルダにOATというファイルが自動生成され、その下に自動生成コードが入る
- このサンプルでは、以下が自動生成される
  - OAT.h　:自動生成された実行用ヘッダファイル（必須）
  - OAT_Routines.h　:ループアンローリングなど、ppOpen-ATにより自動生成されたコードを収納するファイル
  - OAT_ControlRoutines.c：自動チューニング制御コードを収納するファイル
  - OAT_InstallRoutines.c：インストール時自動チューニングのサブルーチンを収納するファイル
  - OAT_StaticRoutines.c　:実行起動前時自動チューニングのサブルーチンを収納するファイル
  - OAT_DynamicRoutines.c：実行時自動チューニングのサブルーチンを収納するファイル
  - OAT_matmul.c　:対象コードに自動チューニング機能を入れ込んだコード
  - OATLog.txt　: ppOpen-ATプリプロセッサのログを収納するファイル

行列-行列積のアンローリングの処理例（C言語）：コードの説明（1/3）




```
main()
{
        int iauto;
        int n;
        double A[N][N], B[N][N], C[N][N];

#pragma OAT call OAT_ATset(OAT_ALL, OAT_AllRoutines)
#pragma OAT call OAT_ATset(OAT_INSTALL,
OAT_InstallRoutines)
#pragma OAT call OAT_ATset(OAT_STATIC, OAT_StaticRoutines)
#pragma OAT call OAT_ATset(OAT_DYNAMIC,
OAT_DynamicRoutines)
        myid = 0;
        iauto = 1;
        n = 10;
#pragma OAT OAT_DEBUG = 1
```

初期化の指定子（必ずこのように記載する）

デバックモードの指定（しないときは、0を代入）

行列-行列積のアンローリングの処理例（C言語）：コードの説明（2/3）

```
 if (iauto == 1){
#pragma OAT OAT_NUMPROCS = 4
#pragma OAT OAT_STARTTUNESIZE = 100
#pragma OAT OAT_ENDTUNESIZE = 500
#pragma OAT OAT_SAMPDIST = 100
#pragma OAT call OAT_ATexec(OAT_INSTALL,
OAT_InstallRoutines,n,A,B,C)
        }
        InitData(A, B, C, n);
        MatMul(A, B, C, n);
        OutputData(A, B, C, n);
        return 0;
}
```

MPI数の指定（MPIではないときは無視）

自動チューニング開始時の変数の値

自動チューニング終了時の変数の値

自動チューニング時の変数増加のストライド

自動チューニング開始場所の指定
（この場所で自動チューニングが開始される）

行列-行列積のアンローリングの処理例（C言語）：コードの説明（3/3）

```
#pragma OAT call OAT_BPset("n")
#pragma OAT install unroll (i) region start
#pragma OAT name MyMatMul
#pragma OAT varied (i) from 1 to 64
#pragma OAT debug (pp)
#pragma OAT allocate (auto)
#pragma omp parallel for
    for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
            da1 =  A[i][j];
            for(k = 0 ; k < n ; k++){
                dc = C[k][j];
                da1 = da1 + B[i][k] * dc;
            }
            A[i][j] = da1;
        }
    }
#pragma OAT install unroll (i) region end
```

自動チューニング変数指定（n）

インストール時AT、アンローリング、対象ループ変数（i）を指定

AT領域名称の定義

アンローリング段数の指定
1段～64段まで

デバック表示の指定

CPUとGPUとの実行をする場合、nに応じて、高速なもので実行する自動チューニングを指定
（GPU実行を行うオプションを指定しないと無効）

行列-行列積のアンローリングの処理例（C言語）：自動生成コードのコンパイル

1. OATディレクトリに入る
   $ cd OAT
2. 自動生成コードをコンパイルする
   gccの場合は、以下
   $ gcc.exe -O3 -Wl,--stack,10485760 OAT_matmul.c
3. a.exeを実行する
   $ ./a.exe

行列-行列積のアンローリングの処理例（C言語）: 自動チューニング実行ログ

```
$ ./a.exe
N=100 iusw1=1 0.000000
N=100 iusw1=2 0.000000
N=100 iusw1=3 0.000000
N=100 iusw1=4 0.000000
N=100 iusw1=5 0.000000
N=100 iusw1=6 0.000000
N=100 iusw1=7 0.000000
N=100 iusw1=8 0.000000
N=100 iusw1=9 0.000000
N=100 iusw1=10 0.000000
N=100 iusw1=11 0.000000
N=100 iusw1=12 0.000000
N=100 iusw1=13 0.000000
....
```

```
N=500 iusw1=54 0.062000
N=500 iusw1=55 0.079000
N=500 iusw1=56 0.078000
N=500 iusw1=57 0.078000
N=500 iusw1=58 0.062000
N=500 iusw1=59 0.094000
N=500 iusw1=60 0.063000
N=500 iusw1=61 0.093000
N=500 iusw1=62 0.063000
N=500 iusw1=63 0.078000
N=500 iusw1=64 0.078000
N=500 BestSw=17
myid: 0
Install Routine: MyMatMul
iusw1: 1
```

行列-行列積のアンローリングの処理例（C言語）: 最適パラメタの情報

- 自動チューニング終了後、以下のファイルが自動生成される
  - OAT_InstallMyMatMulParam.dat
- 上記ファイルの中に、最適なパラメタ情報が残る

行列-行列積のアンローリングの処理例（C言語）: 最適パラメタの情報

```
$ cat OAT_InstallMyMatMulParam.dat
(MyMatMul
  (OAT_NUMPROCS 4)
  (OAT_SAMPDIST 100)
  (OAT_PROBSIZE 100
    (MyMatMul_I 1)
  )
  (OAT_PROBSIZE 200
    (MyMatMul_I 1)
  )
  (OAT_PROBSIZE 300
    (MyMatMul_I 2)
  )
  (OAT_PROBSIZE 400
    (MyMatMul_I 20)
  )
  (OAT_PROBSIZE 500
    (MyMatMul_I 17)
  )
)
```

行列-行列積のアンローリングの処理例（C言語）：最適パラメタでの実行

1. Emacsで iauto=0に変更。
   適当な行列サイズ（n=500など）に変更。
   $ emacs -nw OAT_matmul.c
2. 自動生成コード再コンパイルする
   gccの場合は、以下
   $ gcc.exe -O3 -Wl,--stack,10485760 OAT_matmul.c
3. a.exeを実行する
   $ ./a.exe

行列-行列積のアンローリングの処理例（C言語）：最適パラメタでの実行ログ

```
$ ./a.exe
myid: 0
Install Routine: MyMatMul
iusw1: 17
```

インストール時ATの結果
である、iループのアンローリング段数
の17段での実行を確認

質問タイム

ppOpen-HPC

より深く知っていただくために

ATフレームワーク

- インストール時ATを、GMRES法などの反復解法に初めて適用した[1)]。世界的に見ても、反復解法をATする提案は当時はなく、AT研究として初期の研究の1つである。
- FIBER方式[2),3)]では、インストール時、実行起動前時、実行時のATタイミングを規定し、AT機能の規格化（標準化）を行った。また、汎用的な性能モデル（ブラックボックスモデル）を採用し、多項式近似による性能モデルを最小二乗法で最適化し、パラメタ探索に適用する汎用的なATフレームワークを示した。
- AutoPilitを用いた実行時のパラメタ最適化を、疎行列ダイレクトソルバに適用したATフレームワークを提案[4)]。疎行列ダイレクトソルバの実行時AT適用としては、最も古い適用事例といえる。
- FIBER方式による世界初の自動チューニング言語ABCLibScriptを提案し、AT言語の機能を規格化した[5)]。汎用ATフレームワークを有するAT言語として世界初の研究開発事例といえる。
- 疎行列反復解法による数値計算ライブラリのAT機能を規定するOpenATLibを提案[6)]した。疎行列ライブラリ用の汎用AT機構のライブラリ化として初めての提案である。
- ABCLibScriptの適用アプリを広げ（有限差分法など）、さらにループ変換の機能を追加し規格化を行ったAT言語ppOpen-ATを提案[7),8),9)]。

数値計算ライブラリ適用(その1)

- ATにおいて、最も歴史があるのが、数値計算ライブラリへのAT機能の実装である（例：ATLAS）。
- 世界で初めて、疎行列反復解法にAT機能の適用可能性のフィージビリティスタディをした研究が報告10)である。
- 三重対角化、GMRES法、および密行列と帯行列LUにインストール時ATと実行時ATを入れたAT機能付き数値計算ライブラリILIB[11),12),13),14),15),16)]が開発された。
- 再帰BLASにATが適用できる方式(AutoTuned-RB)の提案[17)]。
- 対称密行列固有値ソルバABCLib_DRSSEDにおけるAT方式（実行起動前AT、および、FIBER方式による実装）の提案[18)]をした。
  - 非BLASの演算に適用し、有効であることを示した。
  - ヘテロ環境を想定し、計算負荷バランスの事前調整機能の実装提案を行っている。
  - さらに、FIBER方式によるATモデル（多項式による近似）がうまくいかない例として、実行時間変動が激しい例を提示した。この実例を克服するため、後のD-Spline法によるAT提案へとつながった。

数値計算ライブラリ適用(その2)

- 三重対角行列用固有値アルゴリズムMRRRにおいて、最も計算負荷の高い二分法について、複数の固有値と複数の探索点を導入することによる並列性の高い方式MMEを提案し、MMEの性能パラメタをATする方式[19)]を提案した。このAT方式は、動的計画法に基づく方式である。
- 疎行列反復解法GMRES（m）におけるリスタート周期、疎行列ベクトル積実装など、ほぼ全てのパラメタをチューニングする実行時AT方式[20)]を提案した。
- 疎行列データ形式について、フリーの数値流体ソフトウェアOpenFOAMへの適用を指向したAT方式[21)]を提案した。
- 任意の行列に対する並列実行時の負荷分散が可能なSegmented Scan方式において、スカラ計算機向きにしたBSS方式の実行時AT、および対称行列用の高スレッド実装を選択するAT方式[22)]を提案した。
- 疎行列反復解法の前処理方式を、実行時の収束情報をもとに決定するAT方式[23)]を提案した。
- 疎行列反復解法GMRES（m）法のリスタート周期のATの効率的なAT方式を、フランスの研究チームとともに提案評価した[24)]。この手法は、階層キャッシュの容量情報を利用し、パラメタ探索を効率的に行う手法である。このようにハードウェアなどの情報を利用し、効率よくATする方針を、スマートチューニングという。
- 数値計算ミドルウェアppOpen-HPCに適用できるAT方式を提案し、AT言語ppOpen-ATに参照実装した[25)]。

AT用性能モデル(D-Spline法によるIPPE)

- 工学院大の田中輝雄教授により開発されたD-Splineによる性能モデル[26)]は、離散スプライン近似により、任意のコードの実行時間のモデル化が可能である。
- D-SplineによるATでは、得られた性能モデルによる関数の各地点での勾配を計算し、最少と予想されるパラメタを探索候補にする（IPPE方式[27),28)]）。複数回同じ探索候補が選ばれた点で、終了する。
  - IPPEでは性能モデル化のための最適化計算に、Givens法によるQR分解を利用した逐次追加型の最適化方式を提案。低い演算コストで推定可能。そのため、実行時ATに採用可能。
  - D-Splineの採用により、関数に数学的な特異点があっても推定可能。この特性が、実用上とても大きい。
- 適用対象を、疎行列ベクトル積など、密行列以外の主要な数値計算演算にも拡大[28),30)]。
- 当初は1パラメタ推定のみ推定可能であったが、2パラメタの同時推定にも拡張[31)]。

Originality (AT Languages)

AT Language / Items	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8
ppOpen-AT	**OAT Directives**	✔	✔	✔	✔	✔		**None**
Vendor Compilers	Out of Target			Limited				-
Transformation Recipes	Recipe Descriptions	✔					✔	ChiLL
POET	Xform Description	✔					✔	POET translator, ROSE
X language	Xlang Pragmas	✔					✔	X Translation, 'C and tcc
SPL	SPL Expressions	✔				✔	✔	A Script Language
ADAPT	ADAPT Language	✔					✔	Polaris Compiler Infrastructure, Remote Procedure Call (RPC)
Atune-IL	Atune Pragmas					✔		A Monitoring Daemon
PEPPHER	PEPPHER Pragmas (interface)	✔				✔	✔	PEPPHER task graph and run-time
Xevolver	Directive Extension (Recipe Descriptions)		(✔) (Users need to define rules.)	(✔)		(✔)		ROSE, XSLT Translator

#1: Method for supporting multi-computer environments. #2: Obtaining loop length in run-time.
#3: Loop split with increase of computations[8] ,and loop collapses to the split loops[7),8),9)] .
#4: Re-ordering of inner-loop sentences[8] . #5:Code selection with loop transformations (Hierarchical AT descriptions*) *This is originality in current researches of AT as of 2015. #6:Algorithm selection.
#7: Code generation with execution feedback. **#8: Software requirement.**

References (related to the authors)

自動チューニングフレームワーク関連

1) H. Kuroda, T. Katagiri, Y. Kanada: Knowledge discovery in auto-tuning parallel numerical library, Progress in Discovery Science", Springer LNAI, Vol. 2281, pp.628-639 (2001)
2) 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)
3) T. Katagiri, K. Kise, H. Honda, T. Yuba: Effect of auto-tuning with user's knowledge for numerical software, Proceedings of the ACM 1st conference on Computing frontiers (CF2004), pp.12-25, (2004)
4) T. Katagiri, Y. Ishii, H. Honda: RAO-SS: A prototype of run-time auto-tuning facility for sparse direct solvers, Technical Report of UEC-IS-2005-2, pp. 1-10 (2005)
5) T. Katagiri, K. Kise, H. Honda, T. Yuba: ABCLibScript: A directive to support specification of an auto-tuning facility for numerical software, Parallel Computing ,32 (1), pp.92-112 (2006)
6) 櫻井隆雄、直野健、片桐孝洋、中島研吾、黒田久泰: OpenATLib：数値計算ライブラリ向け自動チューニングインターフェース、情報処理学会論文誌：ACS, Vol.3, No.2, pp.39-47 (2010)
7) 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 (2013)
8) 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 (2014)
9) T.Katagiri, S.Ohshima, M. Matsumoto: Directive-based Auto-tuning for the Finite Difference Method on the Xeon Phi, The Tenth International Workshop on Automatic Performance Tuning (iWAPT2015) (In Conjunction with the IEEE IPDPS2015), Proceedings of IPDPSW2015, pp.1221-1230 (2015)

References (related to the authors)

自動チューニング機能付き数値計算ライブラリ: Part I of II

10) 黒田久泰，片桐孝洋，佃良生，金田康正：自動チューニング機能付き並列数値計算ライブラリ構築の試み --- 対称疎行列用の連立一次方程式ソルバを例にして ---, 情報処理学会第57回全国大会講演論文集(1), pp.1-10 - 1-11 (1998)

11) T. Katagiri, H. Kuroda, and Y. Kanada: A Methodology for Automatically Tuned Parallel Tri-diagonalization on Distributed Memory Parallel Machines, Proceedings of VecPar2000, pp.265 -277 (2000)

12) H. Kuroda, T. Katagiri, Y. Kanada: Performance of Automatically Tuned Parallel GMRES(m) Method on Distributed Memory Machines, Proceedings of VecPar2000, pp.251-264, (2000)

13) 片桐孝洋，黒田久泰，大澤清，金田康正: ILIB: 自動チューニング機能付き並列数値計算ライブラリとその性能評価, 2000年 並列処理シンポジウム(JSPP'2000)，JSPP'2000 論文集，pp. 27-34 (2000)

14) 片桐孝洋，黒田久泰，大澤清，工藤誠，金田康正: 自動チューニング機構が並列数値計算ライブラリに及ぼす効果, 情報処理学会論文誌: ハイパフォーマンスコンピューティングシステム, Vol.42, No.12, pp.60-76 (2001)

15) T. Katagiri: A study on large scale eigensolvers for distributed memory parallel machines, Ph. D Thesis, the Department of Information Science, the University of Tokyo (2001)

16) H. Kuroda, T. Katagiri, M. Kudoh, Y. Kanada: ILIB_GMRES: An auto-tuning parallel iterative solver for linear equations, Proceedings of SC2001 (2001) (Poster)

17) 木下靖夫，片桐孝洋，弓場敏嗣: AutoTuned-RB: 再帰BLASライブラリの自動チューニング方式, 2005年 ハイパフォーマンスコンピューティングと計算科学シンポジウム（HPCS2005），HPCS2005 論文集，pp. 33-40 (2005)

References (related to the authors)

自動チューニング機能付き数値計算ライブラリ: Part II of II

18) 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)

19) T. Katagiri, C. Vömel, JW Demmel: Automatic performance tuning for the multi-section with multiple eigenvalues method for symmetric tridiagonal eigenproblems, Proceedings of Applied Parallel Computing, State of the Art in Scientific Computing (PARA2006), Springer LNCS, Vol. 4699, pp.938-948 (2007)

20) K. Naono, T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, S. Itoh and K. Nakajima: A Fully Run-time Auto-tuned Sparse Iterative Solver with OpenATLib, The 4th International Conference on Intelligent and Advanced Systems (ICIAS2012), Proceedings of ICIAS2012, pp. 143-148 (2012)

21) S. Ito, S. Ohshima, T. Katagiri: SSG-AT: An Auto-tuning Method of Sparse Matrix-vector Multiplicataion for Semi-Structured Grids - An Adaptation to OpenFOAM -, Special Session of Special Session: Auto-Tuning for Multicore and GPU (ATMG) , Proceedings of MCSoC-2012, pp.191-197 (2012)

22) T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, K. Naono and K. Nakajima: Control Formats for Unsymmetric and Symmetric Sparse Matrix-vector Multiplications, Selected Papers of 10th International Meeting on High-Performance Computing for Computational Science (VECPAR'2012) , Springer Lecture Notes in Computer Science, Volume 7851, pp.236-248 (2013)

23) T. Sakurai, T. Katagiri, H. Kuroda, K. Naono, M. Igai, S. Ohshima: A Sparse Matrix Library with Automatic Selection of Iterative Solvers and Preconditioners, Eighth international Workshop on Automatic Performance Tuning (iWAPT2013) (In conjunction workshop with International Conference on Computational Science, ICCS2013), Proceedings of the International Conference on Computational Science, ICCS 2013,Vol. 18, pp.1332-1341 (2013)

24) T. Katagiri, P.Y. Aquilanti, S. Petiton: A smart tuning strategy for restart frequency of GMRES (m) with hierarchical cache sizes, Proceedings of High Performance Computing for Computational Science-VECPAR 2012, Springer LNCS, Vol.7851, pp.314-328 (2013)

25) Kengo Nakajima, Masaki Satoh, Takashi Furumura, Hiroshi Okuda, Takeshi Iwashita, Hide Sakaguchi, Takahiro Katagiri, Masaharu Matsumoto, Satoshi Ohshima, Hideyuki Jitsumoto, Takashi Arakawa, Futoshi Mori, Takeshi Kitayama, Akihiro Ida and Miki 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) DOI:10.1007/978-4-431-55420-2_2

References (related to the authors)

自動チューニングのための性能モデル

26) T. Tanaka, T. Katagiri, T. Yuba: d-Spline based incremental parameter estimation in automatic performance tuning, Proceedings of Applied Parallel Computing, State of the Art in Scientific Computing (PARA2006), Springer LNCS, Vol. 4699, pp.986-995 (2007)

27) 田中輝雄，片桐孝洋，弓場敏嗣: ソフトウェア自動チューニングにおける標本点追加型性能パラメタ推定法，電子情報通信学会論文誌 A，Vol.J90-A, No.4, pp .281--291 (2007)

28) 田中輝雄，片桐孝洋，弓場敏嗣:ソフトウェア自動チューニングにおける標本点逐次追加型性能パラメタ推定法の疎行列計算への適用, 情報処理学会論文誌：コンピューティングシステム，Vol.48, No. SIG13 (ACS 19), pp. 223-234 (2007)

29) R. Suda, L. Cheng, T. Katagiri: A Mathematical Method for Online Autotuning of Power and Energy Consumption with Corrected Temperature Effects, Eighth international Workshop on Automatic Performance Tuning (iWAPT2013) (In conjunction workshop with International Conference on Computational Science, ICCS2013), Proceedings of the International Conference on Computational Science, ICCS 2013,Vol. 18, pp.1302–1311 (2013)

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, Issue 4, pp. 299-307 (2014)

31) Riku Murata, Jun Irie, Akihiro Fujii, Teruo Tanaka, Takahiro Katagiri: Enhancement of Incremental Performance Parameter Estimation on ppOpen-AT, Special Session: Auto-Tuning for Multicore and GPU (ATMG-15) (In Conjunction with the IEEE MCSoC-15), Proceedings of MCSoC2015, Politecnico di Torino, Turin, Italy, September 23-25, 2015, (2015)

References (related to the authors)

その他：最近の出版物 (2016~)

I. Takahiro Katagiri, Masaharu Matsumoto, Satoshi Ohshima: Auto-tuning of Hybrid MPI/OpenMP Execution with Code Selection by ppOpen-AT, The Eleventh International Workshop on Automatic Performance Tuning (iWAPT2016) (In Conjunction with the IEEE IPDPS2016), Proceedings of IPDPSW2016, pp. 1488 - 1495, May 27th, 2016, Chicago Hyatt Regency, Chicago, Illinois USA, (2016) DOI: 10.1109/IPDPSW.2016.49

II. Satoshi Ohshima, Takahiro Katagiri, Masaharu Matsumoto: Utilization and Expansion of ppOpen-AT for OpenACC, The Eleventh International Workshop on Automatic Performance Tuning (iWAPT2016) (In Conjunction with the IEEE IPDPS2016), Proceedings of IPDPSW2016, pp.1496 - 1505, May 27th, 2016, Chicago Hyatt Regency, Chicago, Illinois USA (2016) DOI: 10.1109/IPDPSW.2016.123

III. Takahiro Katagiri, Satoshi Ohshima, Masaharu Matsumoto: Auto-tuning on NUMA and Many-core Environments with an FDM code, The Twelfth International Workshop on Automatic Performance Tuning (iWAPT2017) (In Conjunction with the IEEE IPDPS2017), Proceedings of IPDPSW2017, pp. 1399-1407, June 2nd, 2017, Buena Vista Palace Hotel, Orlando, Florida, USA (2017) DOI: 10.1109/IPDPSW.2017.27

IV. G. Fan, A. Fujii, T. Tanaka, T. Katagiri: D-Spline Performance Tuning Method Flexibly Responsive to Execution Time Perturbation, 12th International Conference on Parallel Processing and Applied Mathematics (PPAM2017), Lublin, Poland, September 10-13 (2017)

V. Takahiro Katagiri: Auto-tuning for The Era of Relatively High Bandwidth Memory Architectures: A Discussion Based on an FDM Application , The Thirteenth International Workshop on Automatic Performance Tuning (iWAPT2018) (In Conjunction with the IEEE IPDPS2018), Proceedings of IPDPSW2018, pp. , May 21 – May 25, 2018, JW Marriott Parq Vancouver, Vancouver, British Columbia CANADA , (2018) DOI: 10.1109/IPDPSW.2018.00167

VI. 山田賢也、片桐孝洋、永井亨、荻野 正雄：疎行列形状のカラー画像を入力としたディープラーニングによる数値計算ライブラリの自動チューニング方式、第162回ハイパフォーマンスコンピューティング研究発表会、くまもと県民交流館パレア、2017年12月19日（火）、情報処理学会研究報告162-HPC-2017, pp.1-10, (2017)

VII. 櫻井 刀麻, 片桐 孝洋, 永井 亨, 荻野 正雄：OpenMPの対象ループを変更する自動チューニング手法の評価、2018年並列／分散／協調処理に関する『熊本』サマー・ワークショップ (SWoPP2018)、2018年7月30日(月) - 8月1日(水)、情報処理学会研究報告HPC-165-2018, pp.1-7 (2018)