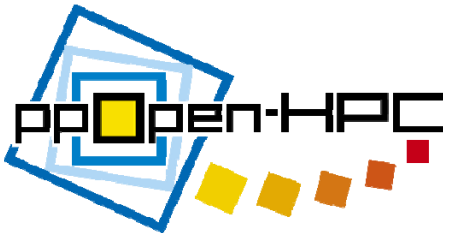


ポストムーア時代の数値計算アルゴリズム開発に向けて

片桐孝洋

(名古屋大学情報基盤センター)

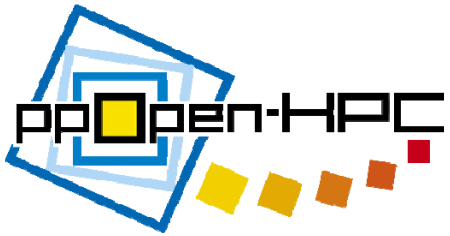
共同研究者

松本正晴、大島聡史

(東京大学情報基盤センター)

研究会「計算物質科学における時空間アップスケーリングと数理手法」
日時：2016年11月28日(月)午後-29日(火)、場所：電気通信大学　西９号館3階　AVホール
2016年11月29日（火）15:00-15:35 (35min)

話の流れ



- 背景
- ポストムーア時代の課題例
- 事例：AT言語 *ppOpen-AT* とFDMコードへの適用
  - –FX100を用いた性能評価
- おわりに

話の流れ

- 背景
- ポストムーア時代の課題例
- 事例：AT言語 *ppOpen-AT* とFDMコードへの適用
  - –FX100を用いた性能評価
- おわりに

「ポストムーアに向けた計算機科学・計算科学の新展開」シンポジウム

http://www.cspp.cc.u-tokyo.ac.jp/p-moore-201512/

- 東工大　松岡聡　教授
- 2015年12月22日＠東大
  - Hosted by:
    ITC, U.Tokyo and GSIC, TITECH
  - Co-hosted by:
    ITC, Hokkaido U., ITC, Kyusyu U.,
    AICS, RIKEN, JST CREST, JH
- 対象:
  - ハードウェア
  - システムソフトウェア
  - アルゴリズム & アプリケーション
    (PI: 中島研吾教授@ITC, U.Tokyo)


「ポストムーアに向けた計算機科学・計算科学の新展開」シンポジウム
New Frontiers of Computer & Computational Science towards Post Moore Era

開催概要

日時　：12月22日（火）10:00開会
会場　：東京大学 武田先端知ビル5階 武田ホール（アクセス）
懇親会：武田ホール ホワイエ
参加費：無料（懇親会参加費：5000円程度予定）
共催　：東京工業大学 学術国際情報センター
　　東京大学 情報基盤センター
協賛　：北海道大学 情報基盤センター
　　九州大学 情報基盤センター

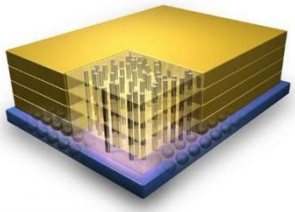
計算性能（FLOPS）の伸びが停止することを意味しています。そのようなポスト・ムーア時代において高性能計算技術の持続的発展を図るためには，幅広い分野の専門家の緊密な協力の下に，現状のスパコンに代表される計算性能（FLOPS）中心の性能向上技術からデータ移動（BYTES）中心に転換する新たな体系を築くことが必要です。
本シンポジウムでは，国内外の3名の招待講演者も含めて，デバイス，アーキテクチャ，システムソフトウェア，アルゴリズム，アプリケーションの各分野の専門家が一堂に会し，ポスト・ムーア時代における様々な技術的問題点について議論することを目的とします。本シンポジウムが契機となって，ポスト・ムーア時代に向けた分野間協力，国際協力の進展とともに，高性能計算の一般的原理を目指す，計算機科学・計算科学を融合した新たな学術領域の形成につながることを強く期待するものです。

Co-located with SC16 in Salt Lake City, Post-Moore's Era Supercomputing (PMES) Workshop!

ポストムーア時代のAT！？

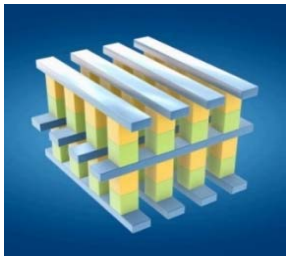
- 2020年頃のエクサスケールスパコンを境に
  ムーアの法則での速度向上が終焉
  - “One-time Speedup”のメニーコア化、
    配線微細化、による低電力化の終わり

  →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、
    アクセスレイテンシは維持（→高性能化）
  - X-Y方向は、低アクセスレイテンシ
- ノード間は、アクセスレイテンシは（相対的に）
  悪化、だが光配線技術などでバンド幅は増加
  できる
- 以上は2020年後半（→ポストムーア）



HMC

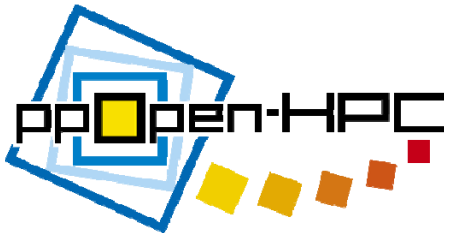
Source:
http://www.engadget.com/2011/12/06/the-big-memory-cube-gamble-ibm-and-micron-stack-their-chips/



Intel 3D Xpoint

Source:
http://www.theregister.co.uk/2015/07/28/intel_micron_3d_xpoint/

ポストムーア時代のAT！？



- 演算量（FLOPS）増加ではなく
  <データ移動量増加> の
  アルゴリズム変更
  が求められる！

- メモリバンド幅増加
- 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非均一

を考慮したアルゴリズム再構築

AT技術の課題

- 階層型AT方式
- 通信削減アルゴリズム
- 高バンド幅活用の
  <新アルゴリズム探求>
  と<自動切替>
  （アルゴリズム選択）
  - 古典アルゴリズム復活？
  - キャッシュブロック化なしアルゴリズム
  - 陽解法から陰解法へ
  - アウト・オブ・コア(OoC)アルゴリズム（主メモリ外）

話の流れ

- 背景
- ポストムーア時代の課題例
- 事例：AT言語 *ppOpen-AT* とFDMコードへの適用
  - –FX100を用いた性能評価
- おわりに

事例(1)

- ブロック化アルゴリズム再考
  - 現状:アルゴリズムを変更してBLAS3利用
  - ⇒実装が大変　⇒場合により、演算量の増加
    (例:固有値計算)
  - ポストムーア:非ブロック計算の復帰
  - ⇒BLASで定義できない複雑演算を直接実装
  - B/F＞4以上のハードウェアでBLASインタフェース再考
    - DGEMMが不要
    - BLASで定義できない演算: $A^{(k)} - (x_k - t_k v_k) v_k^T - v_k y_k^T$
      を分離せず演算すると高速
      →BLASをつかなわない方が高速
  - メモリに直接演算器を付けDAXPY演算ができれば:
    - 低レイテンシー化により、小規模行列演算が高速化
  - ⇒通信レイテンシ隠蔽のための通信ブロック化は必要

事例(2)

- 精度保証
  - 現状:大規模計算の演算精度が未検証
  - ⇒連立一次方程式や固有値問題の
    真の解からのずれを検証する手段がない
  - ⇒HPLの精度:倍精度1000万次元で1～2桁精度
  - ⇒何を計算しているかわからない
  - ポストムーア:精度保証技術の導入?
  - 高B/Fの環境で密行列演算が高速化できれば:
  - ⇒真の解からのずれを知ることができる、反復改良
    で (なしの場合に対し低いコストで)演算精度を改良
  - ⇒高性能実装のための実装技法、ライブラリの研究開発が未整備

ポストムーア課題（1/4）

- 超高バンド幅アーキテクチャ向きアルゴリズム再構築
- データ移動が多い古典的アルゴリズム（非ブロック化アルゴリズム）
  - B/F = 1程度で効果あり（HITACHI SR2201、現在でもL1キャッシュ内データのB/F値）
  - B/F > 4程度で、おそらくブロック化は要らなくなる
  - 密行列固有値解法一般（ブロック化すると演算量が増加）
    - 非ブロック化Householder三重対角化
      - 三重対角化は、ブロック化すると演算量が増える！
    - ブロックヤコビ法（B/F値大、演算量10倍、強スケーリング可）
  - 非ブロック化／多階層メモリ向け FFTアルゴリズム

ポストムーア課題（2/4）

- 超高バンド幅アーキテクチャ向き
  アルゴリズム再構築
- 高精度だが、高B/Fなアルゴリズムの
  積極活用
  - 直交化処理
    - 修正G-S（安定、ブロック化困難） vs.
      古典G-S（不安定、ブロック化可）
  - 連立一次方程式の反復解法前処理全般
    （安定な汎用アルゴリズムはB/F値が高い）

ポストムーア課題(3/4)

- 特定部分をFPGA化
  - 固有値計算
    - スツルムカウント
    - ある値以下の場合数をカウントする
    - 必ずIF文が最内側
      - 先進CPU/GPUで実行効率が悪いため、FPGA化による高速化に期待
      - 最外側ループ(固有値存在区間)の並列性はある(多分法と併用)

```
NEG=0;
S=0.0;
for (i=0; i<m; i++) {
   T = S – SIGMA;
   DPLAS = A[i] + T;
   S = T * B[i] / DPLAS;
   if (DPLAS < 0.0) NEG ++
}
```

- 陰解法の復活
  - MHDコード
    - 従来使っている陽解法から <陰解法> へ
    - 陽解法ではそもそも収束しない問題がある:クーラン条件
    - 陰解法:GMRES、BiCGStab法の利用
      (データ移動の多い<疎行列-ベクトル積>ベースの演算)
    - データ移動の多い前処理手法の適用

ポストムーア課題（4/4）

- アウト・オブ・コア（OoC）アルゴリズム
  - キャッシュ＋3D積層メモリ＋新技術（MCDRAM + Intel 3DXpoint）の技術展開を想定した、主記憶外
    超高バンド幅、超多階層メモリ向けOoCアルゴリズム
    - OoC LU分解、OoC QR分解
- 新技術の適用
  - 人工知能技術適用（特にAT技術との融合）
    - 数値アルゴリズムのパラメタ自動設定
      - 前処理、0と見なせる値、解法選択、・・・
    - 最適データ構造（AoS, SoA、・・・）の予測
    - 数値計算アルゴリズム自動導出
    - ・・・などなど
  - 量子コンピュータ・・・

分野間協調(コ・デザイン)

- 数値ライブラリ⇔アプリ
  (ベクトル時代に戻るのではない)
  - 超高バンド幅、超並列アルゴリズム開発
  - 問題レベル、実行時情報を利用した超並列性の抽出
    - 行列リオーダリング、領域分割・・・
    - ループ変換(ループ長を長くして並列性を抽出するコード最適(Collapse)など)
  - 通信回避アルゴリズム
- アプリ⇔計算機システム・計算機アーキテクチャ
  - 電力最適化
  - 高性能実装、精度(4倍精度、SIMD、・・・)
  - メモリ直接接続演算器、・・・・

話の流れ

- 背景
- ポストムーア時代の課題例
- 事例：AT言語 *ppOpen-AT* とFDMコードへの適用
  - –FX100を用いた性能評価
- おわりに

HPCソフトウェア開発流れ図

コア数の増大、
プログラミング
モデルおよびコード最
適化の複雑化
(多様な計算機
アーキテクチャ)

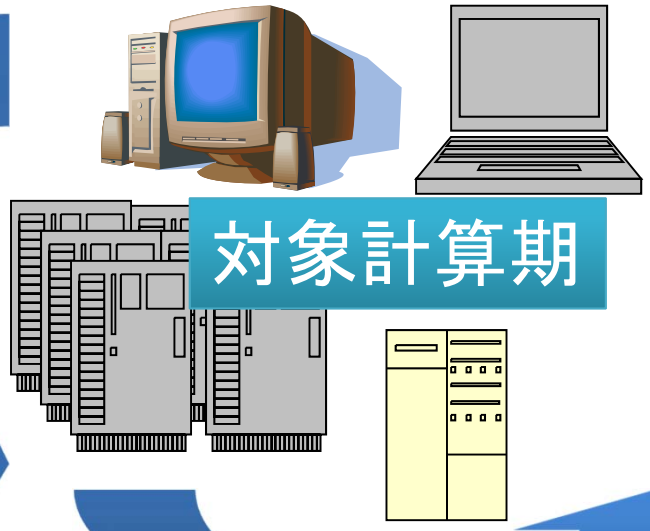
コンパイルと実行

コード
生成

```
do i=1, n, 2
  do j=1, n, 2
    Ctmp1 = C( i,   j )
    Ctmp2 = C( i+1, j )
    do k=1, n, 2
      Btmp1 = B( k,   j )
      Btmp2 = B( k+1, j )
      Ctmp1 = Ctmp1 + A( i,   k   ) * Btmp1
                    + A( i,   k+1 ) * Btmp2
      Ctmp2 = Ctmp2 + A( i+1, k   ) * Btmp1
                    + A( i+1, k+1 ) * Btmp2
    enddo
    C( i,   j )=Ctmp1
    C( i+1, j )=Ctmp2
  enddo
enddo
```

3. 最適化フェーズ

結果の解析



対象計算期

2. プログラミング
フェーズ

```
!ABCLib$ install unroll (i,k) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i,k) from 1 to 8
do i=1, n
  do j=1, n
    do k=1, n
      C( i, j ) = C( i, j ) + A( i, k ) * B( k, j )
    enddo
  enddo
enddo
!ABCLib$ install unroll (i,k) region end
```

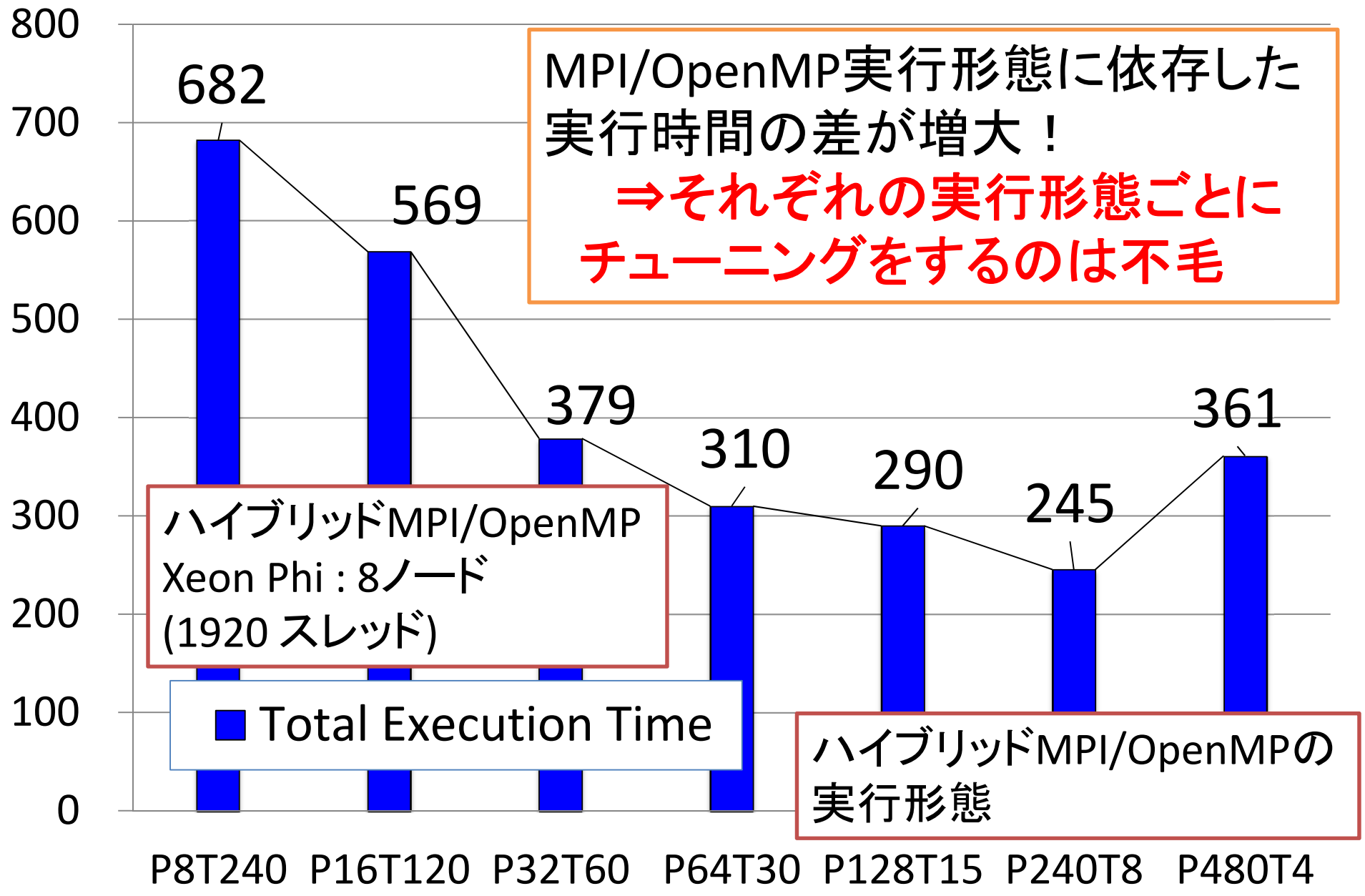
4. チューニングに
関する知識探索と
データベース化
のフェーズ

チューニング知識
データベース

1. 仕様策定フェーズ

ハイブリッドMPI/OpenMP実行の実例

(ある有限差分法のアプリ)



FIBERフレームワーク[Katagiri et.al., 2003]

高度な知識が必要

ライブラリ開発者

ユーザ知識 ①

ディレクティブによる記載

オリジナルコード

ライブラリ公開前

自動コード生成

②

公開ライブラリ

知識不要！

選択

候補 *n*

3

2

1

⑤

実行時

オートチューナー

自動チューニングされたコード実行 ⑥

③ ライブラリ呼び出し

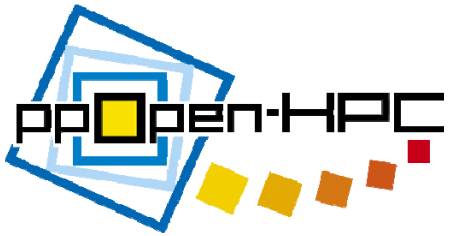
④ 実行時間

ライブラリユーザ

: 対象計算機

Xabclib、ABCLib、ppOpen-AT (ABCLibScript) のAT方式

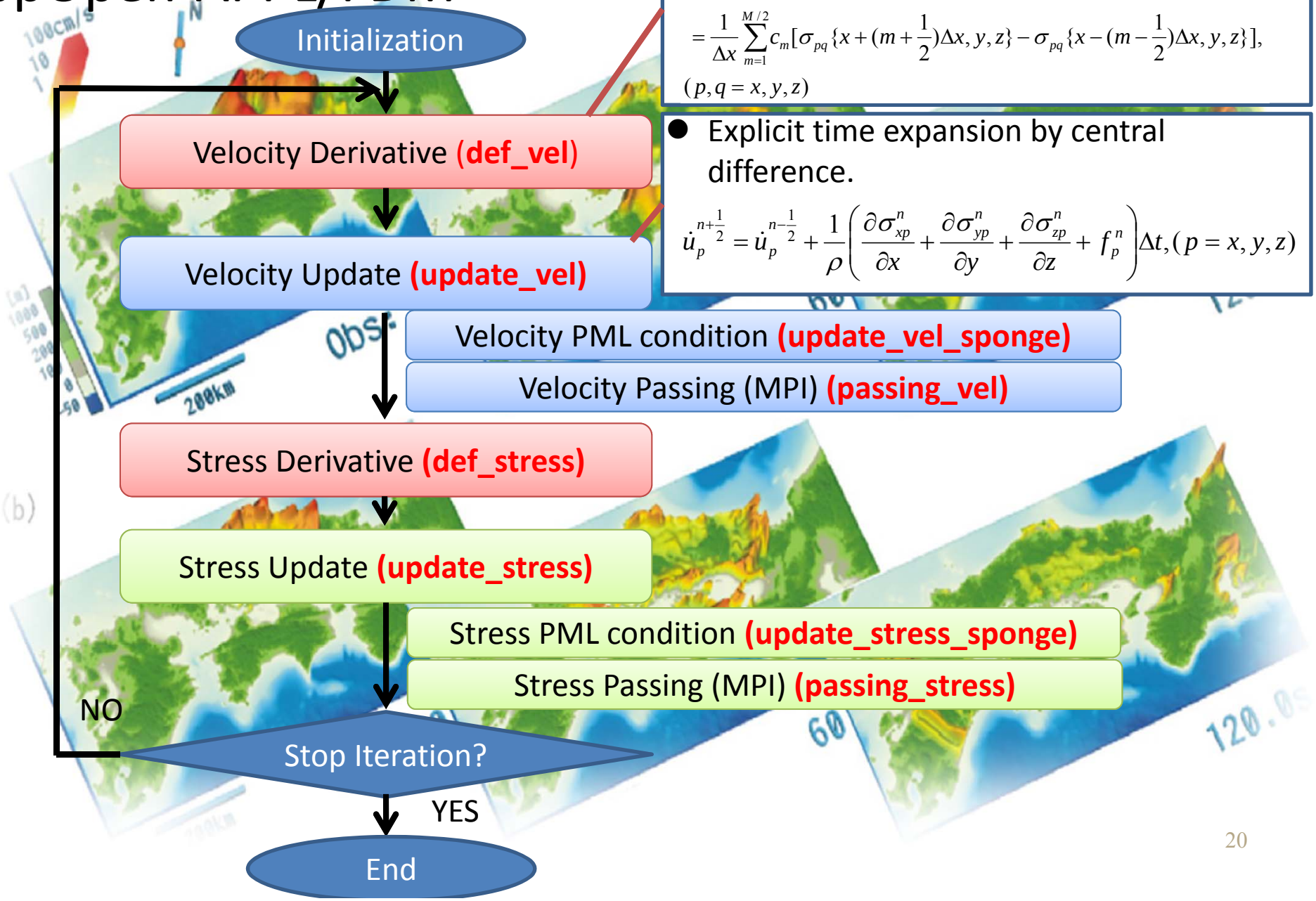
Target Application



- **Seism3D**:

  Simulation for seismic wave analysis.
- Developed by Professor T.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.

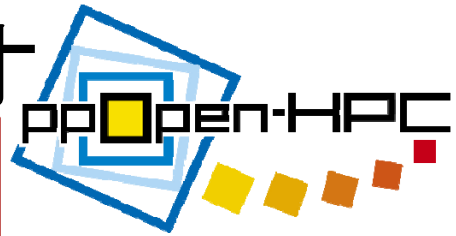
$$\frac{d}{dx}\sigma_{pq}(x,y,z) = \frac{1}{\Delta x}\sum_{m=1}^{M/2} c_m[\sigma_{pq}\{x+(m+\frac{1}{2})\Delta x,y,z\} - \sigma_{pq}\{x-(m-\frac{1}{2})\Delta x,y,z\}],$$

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

- 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

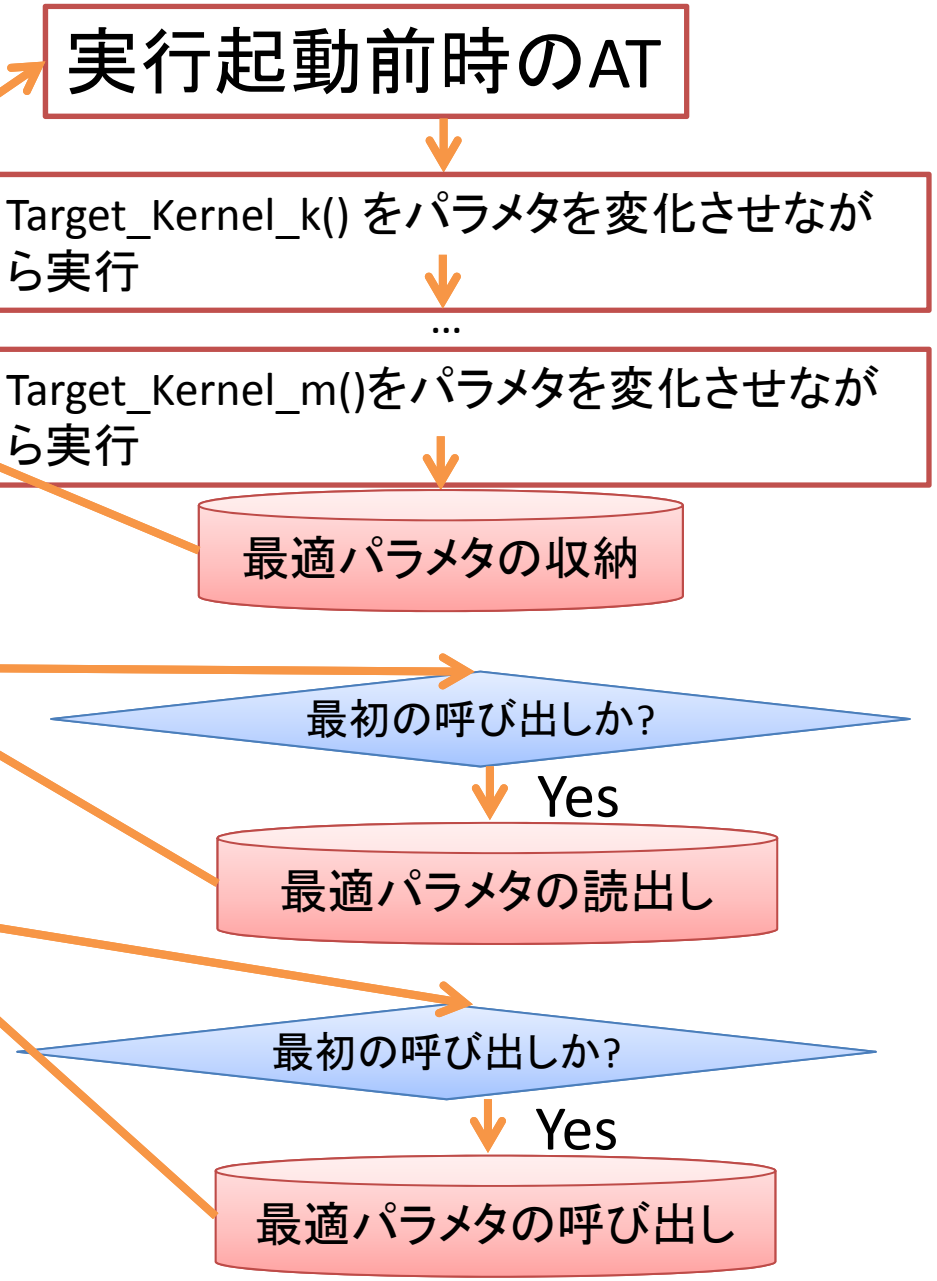
最速カーネル情報の
収納

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）

ppOpen-AT のATタイミング(FIBERフレームワーク)

一度のみ実行(問題サイズやMPI数を変えたりする場合を除く)

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



対象のループ特性

- ３重ループ

```
!$omp parallel do
do k = NZ00, NZ01
  do j = NY00, NY01
    do i = NX00, NX01
      <FDMによる導出式>
    end do
  end do
end do
!$omp end parallel do
```

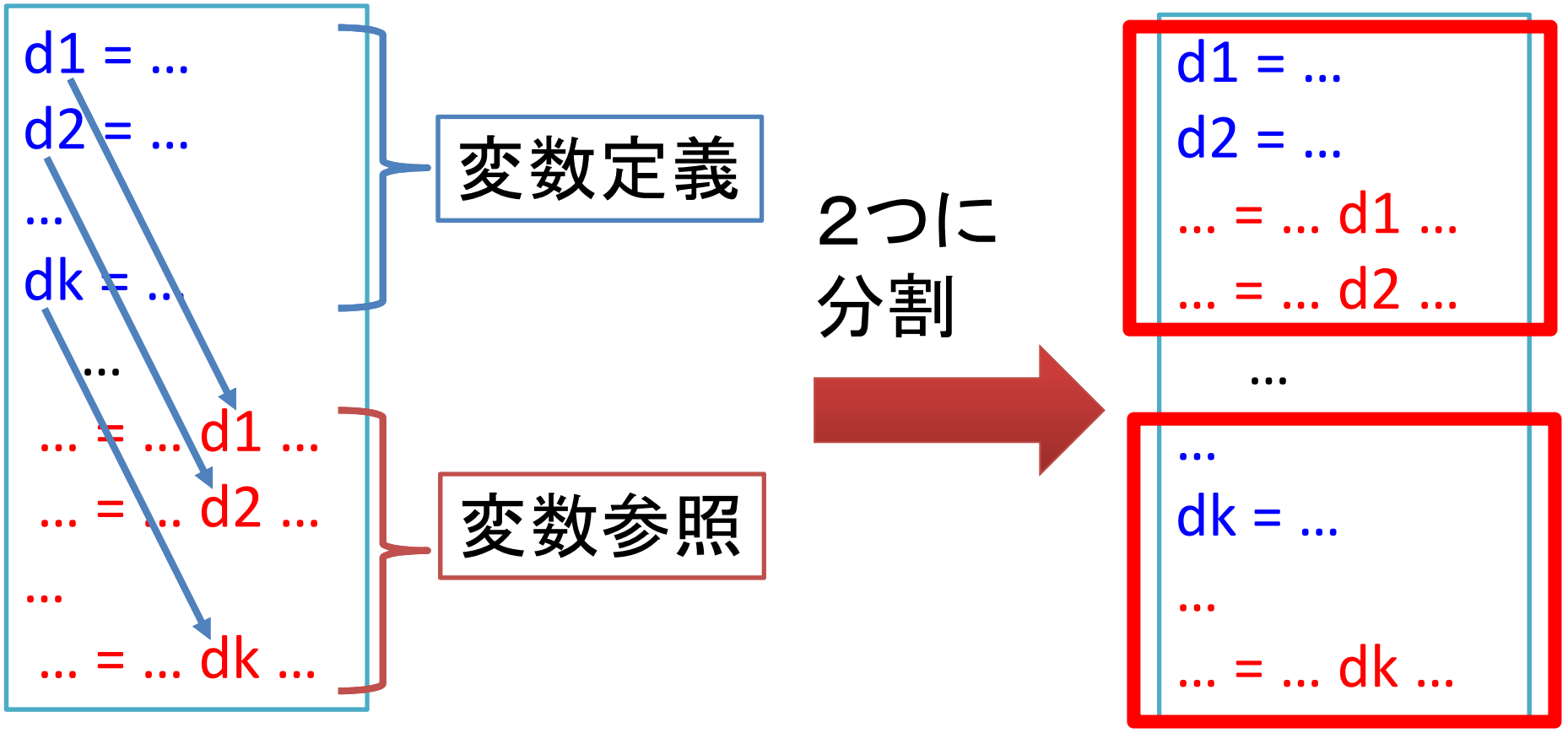
最外側ループのOpenMP記述 (Z-軸)

ループ長は、問題サイズ、MPIプロセス数、OpenMPスレッド数、などで変化する

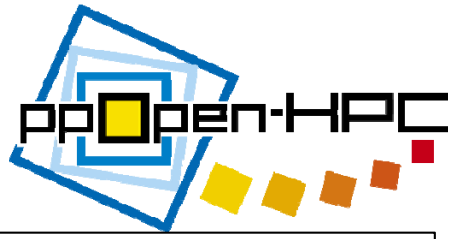
コードは、ループ分割により分割可能

分割可能なコードとは?

- 変数定義と参照が分かれている
- 流れ依存があるが、変数間にデータ依存はない



元のコード




```
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
  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
  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))
  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
```

流れ依存

ループ分割（Loop Collapse）

```
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
   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
ENDDO
DO I = 1, NX
   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+
   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))
   QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
   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
```

分割点

再計算が必要
⇒計算量が増すので、
コンパイラによる自動化が困難

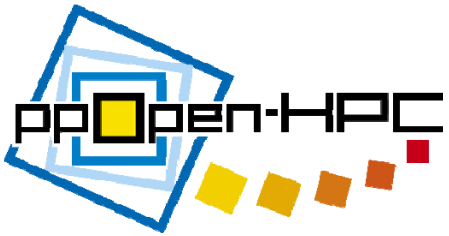
ループ分割（別ループ化）

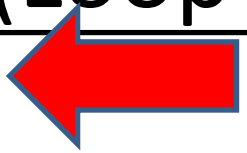
```
DO K = 1, NZ
DO J = 1, NY
DO I = 1, NX
   RL  = LAM (I,J,K)
   RM  = RIG (I,J,K)
   RM2 = RM + RM
   RLTHETA  = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
   QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
   SXX (I,J,K) = ( SXX (I,J,K)  + (RLTHETA + RM2*DXVX(I,J,K))*DT )*QG
   SYY (I,J,K) = ( SYY (I,J,K)  + (RLTHETA + RM2*DYVY(I,J,K))*DT )*QG
   SZZ (I,J,K) = ( SZZ (I,J,K)  + (RLTHETA + RM2*DZVZ(I,J,K))*DT )*QG
ENDDO;  ENDDO; ENDDO
```

完全分割

```
DO K = 1, NZ
DO J = 1, NY
DO I = 1, NX
   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))
   QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
   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;
```

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

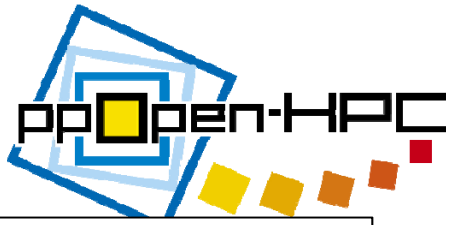
K, J-ループに対するループ融合

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

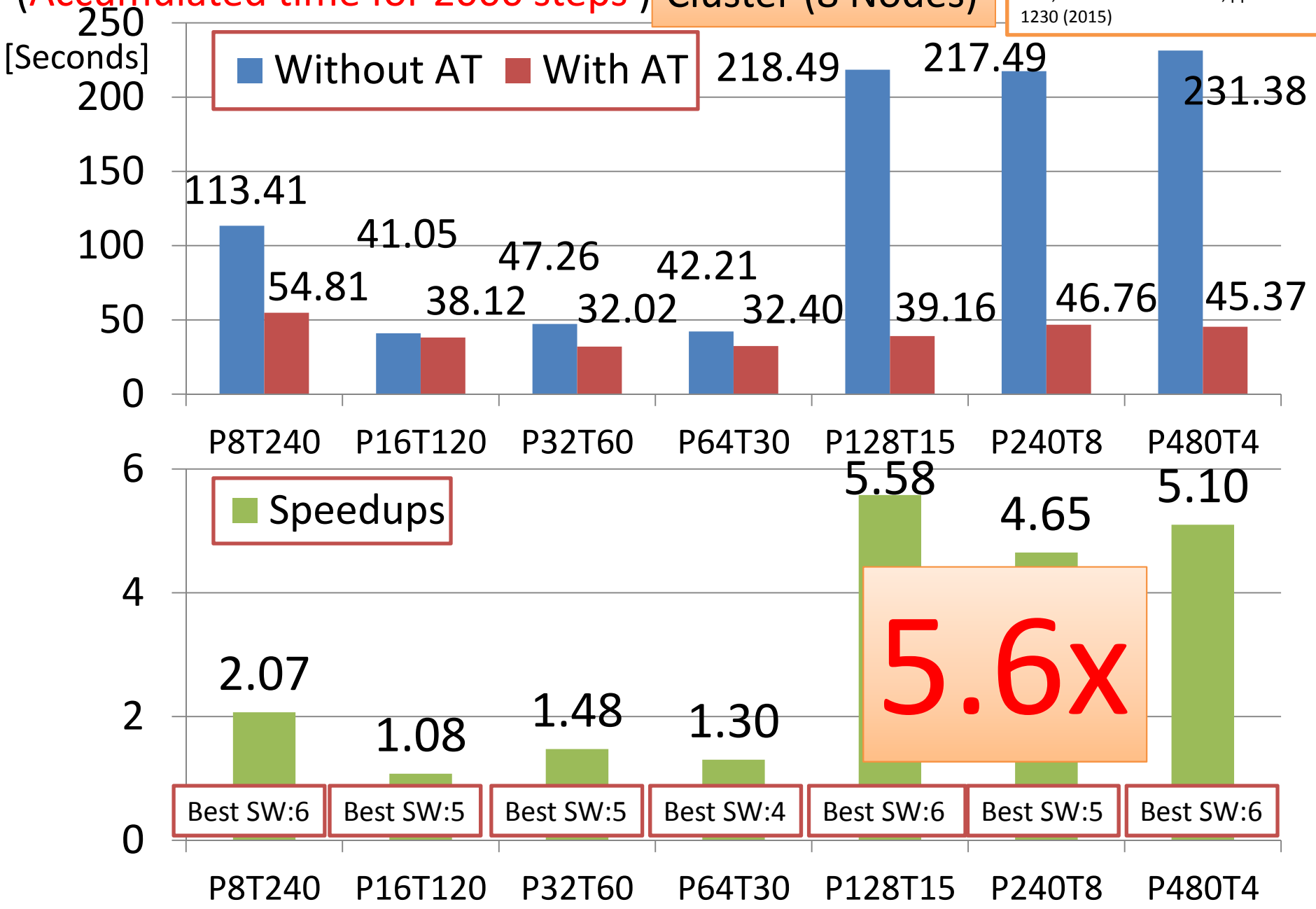
ループ分割時の再計算場所の指定

AT Effect (update_stress)

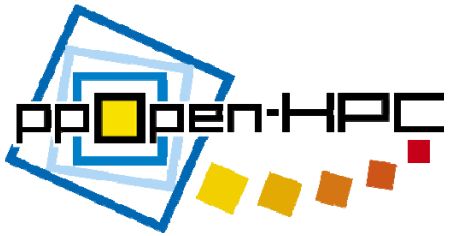
(Accumulated time for 2000 steps)

Xeon Phi (KNC)
Cluster (8 Nodes)

T. Katagiri, S. Ohshima, M. Matsumoto: "Directive-based Auto-tuning for the Finite Difference Method on the Xeon Phi", Proc. of IPDPSW2015, pp.1221-1230 (2015)



コード選択のAT
（階層型AT）



AT WITH CODE SELECTION

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)

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_P
        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

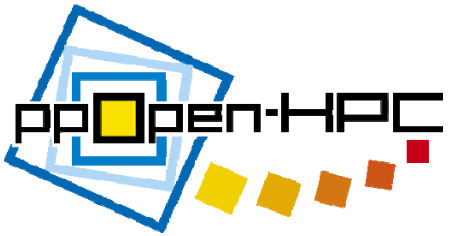
```
    DXVX1 = DXVX0;  DYVY1 = DYVY0
    DZVZ1 = DZVZ0;  D3V3  = DXVX1 + DYVY1 + DZVZ1
     SXX (I,J,K) = SXX (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
     SYY (I,J,K) = SYY (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
     SZZ (I,J,K) = SZZ (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1)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

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

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
```

Fourth-order accurate central-difference scheme for velocity

Explicit time expansion by leap-frog scheme.

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

Code selection by ppOpen-AT and hierarchical AT

Upper Code

With Select clause, code selection can be specified.

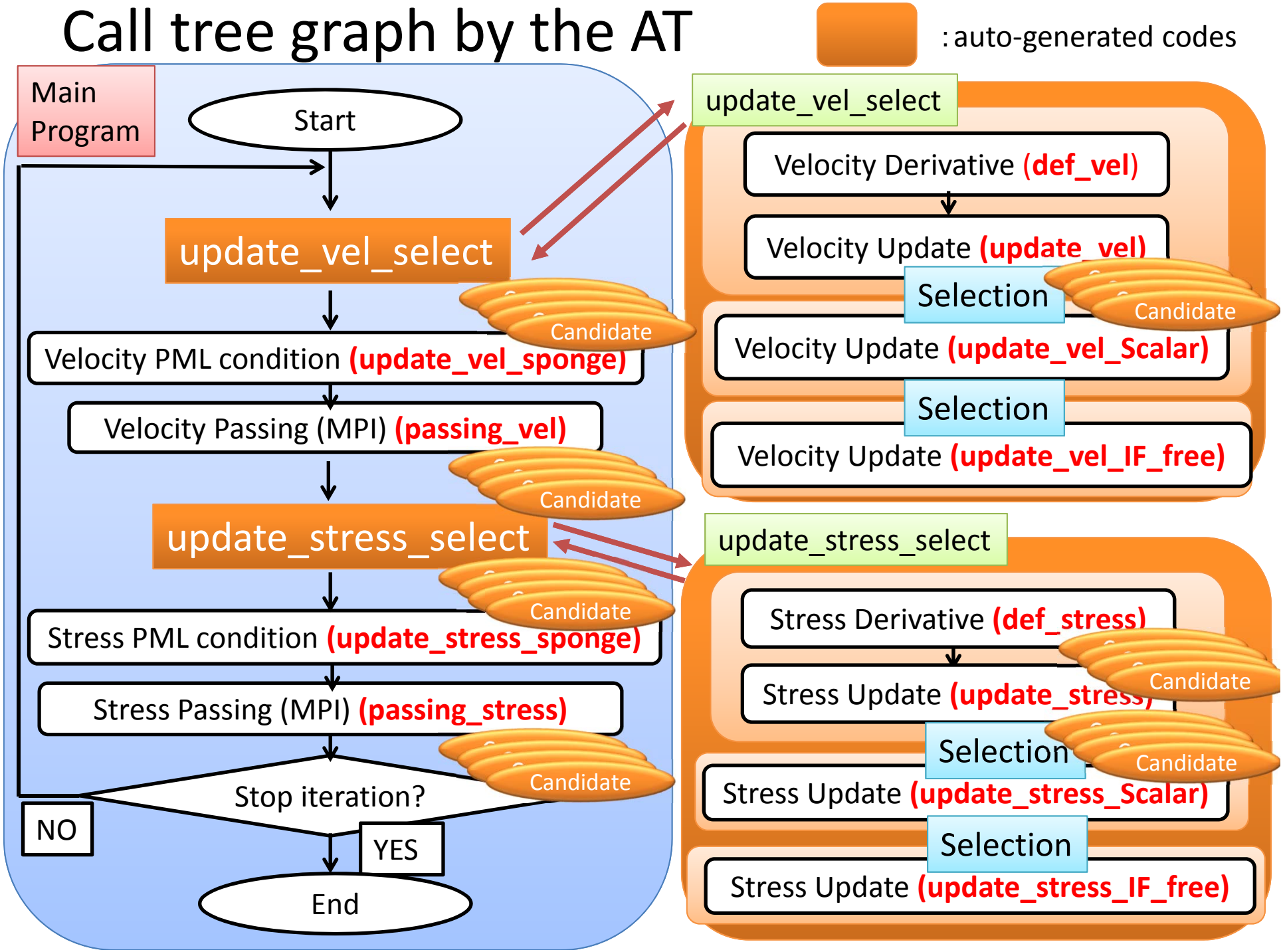
Lower Code

```
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,II
end if
call ppohFDM_update_vel    ( 1, NXP, 1, NYP, 1, NZ
!OAT$ select sub region end
!OAT$ select sub region start
Call ppohFDM_update_vel_Intel  ( 1, NXP, 1, NYP,
!OAT$ select sub region end
!OAT$ install select region end
```

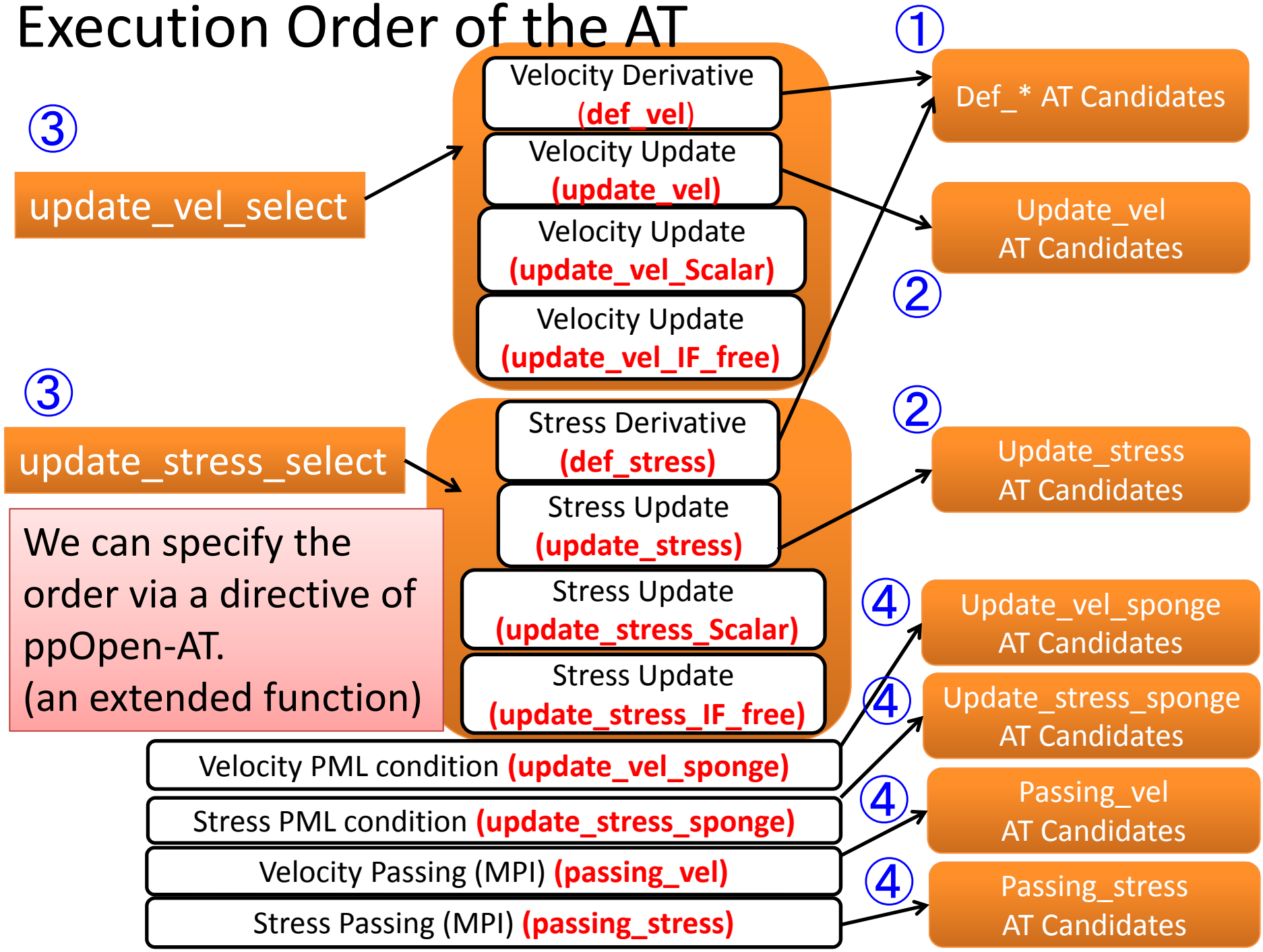
```
subroutine ppohFDM_pdiffx3_p4(....)
....
!OAT$ install LoopFusion region start
....
```

```
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



The Number of AT Candidates (ppOpen-APPL/FDM)

Kernel Names	AT Objects	The Number of Candidates
1. update_stress	・Loop Collapses and Splits ：8 Kinds ・Code Selections : 2 Kinds	**10**
2. update_vel	・Loop Collapses, Splits, and re-ordering of statements: ：6 Kinds ・Code Selections: 2 Kinds	**8**
3. update_stress_sponge	・Loop Collapses：3 Kinds	**3**
4. update_vel_sponge	・Loop Collapses：3 Kinds	**3**
5. ppohFDM_pdiffx3_p4	Kernel Names：def_update、def_vel ・Loop Collapses：3 Kinds	**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	Data packing and unpacking ・Loop Collapses： 3 Kinds	**3**
12. ppohFDM_ps_unpack		**3**
13. ppohFDM_pv_pack		**3**
14. ppohFDM_pv_unpack		**3**

- Total：54 Kinds
- Hybrid MPI/OpenMP: 7 Kinds
- 54 × 7 = 378 Kinds

Outline

- Background
- An Auto-tuning (AT) Language: *ppOpen-AT* and Adapting AT to an FDM code
- Performance Evaluation with the FX100
- Conclusion

FX100

FX100(ITC, Nagoya U.), The Fujitsu PRIMEHPC FX100

Contents		Specifications
Whole System	Total Performance	**3.2 PFLOPS**
	Total Memory Amounts	90 TiB
	Total #nodes	2,880
	Inter Connection	The TOFU2 (6 Dimension Mesh / Torus)
	Local File System Amounts	6.0 PB

2880 Nodes (92,160 Cores)

Contents		Specifications
Node	Theoretical Peak Performance	1 TFLOPS (double precision)
	#Processors (#Cores)	32 + 2 (assistant cores)
	Main Memory Amounts	32 GB
Processor	Processor Name	SPARC64 XI-fx
	Frequency	2.2 GHz
	Theoretical Peak Performance (Core)	31.25 GFLOPS

Comparison with the FX10 (ITC, U. Tokyo) and FX100(ITC, Nagoya U.)

	FX10	FX100	Ratios (FX100/FX10)
Node FLOPS	236.5 GFLOPS	2 TFLOPS (single precision)	8.44x
Memory Bandwidth	85 GB/S	480 GB/S	5.64x
Networks	5 GB/S x2	12.5 GB/S x2	2.5x

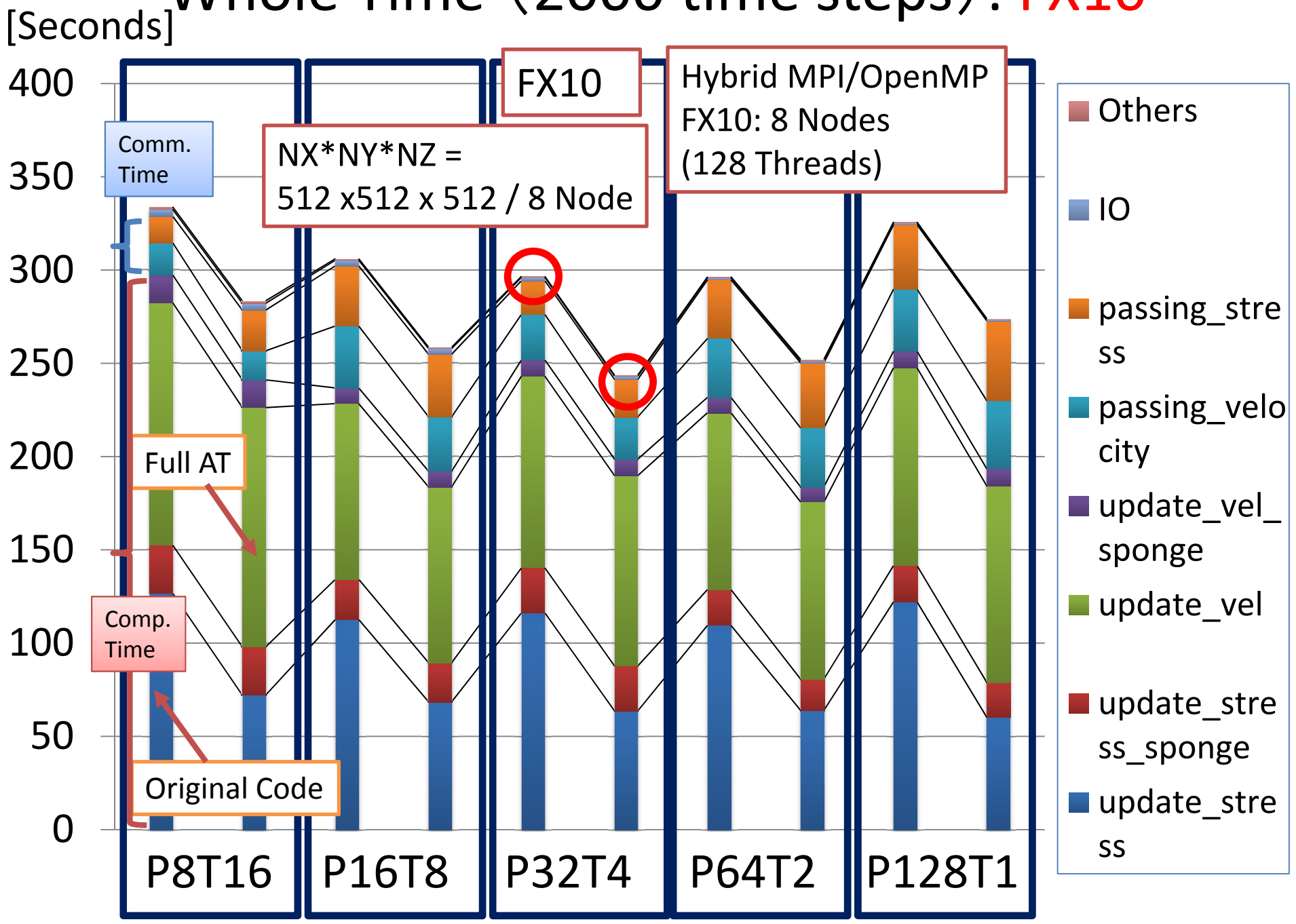
Execution Details

- ppOpen-APPL/FDM ver.0.2
- ppOpen-AT ver.0.2
- The number of time step: 2000 steps
- The number of nodes: 8 node
- 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)
- Target MPI Processes and Threads on the Xeon Phi
  - 1 node of the Ivy Bridge with 2 HT (Hyper Threading)
  - P*X*T*Y*: *X* MPI Processes and *Y* Threads per process
  - P8T32 : Minimum Hybrid MPI-OpenMP execution for ppOpen-APPL/FDM, since it needs minimum 8 MPI Processes.
  - P16T16
  - P32T8
  - P64T4
  - P128T2
  - P256T1: pure MPI
- The number of iterations for the kernels: 100

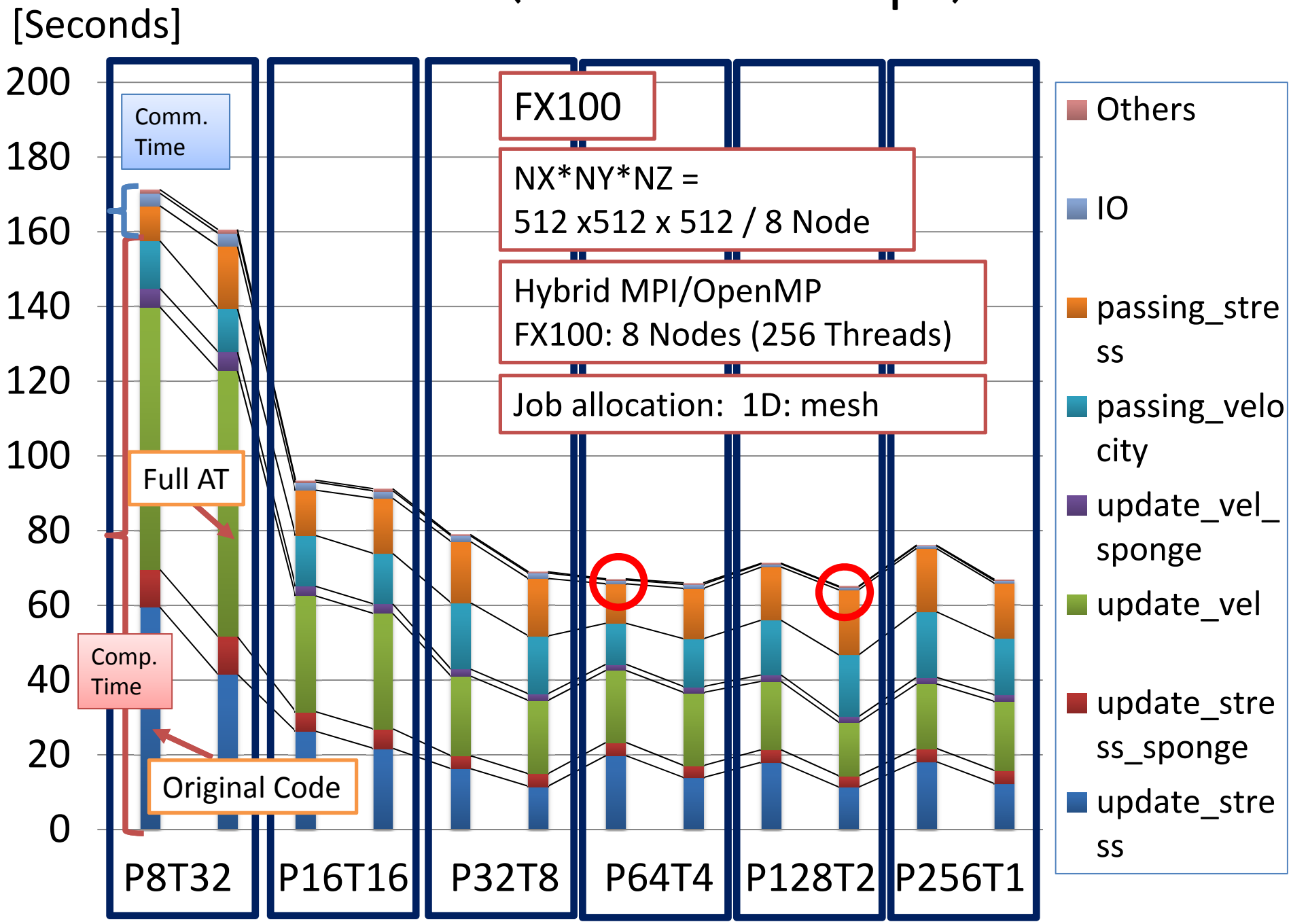
NUMA affinity

- Sparc64 XI-fx is a NUMA.
- 2 sockets: 16 cores + 16 cores
- NUMA affinity
  - Memory allocation
    - “Local allocation” is used.
    - plm_ple_memory_allocation_policy=localalloc
  - CPU allocation
    - P8 and P16:
      plm_ple_numanode_assign_policy=simplex
    - More than P32 :
      plm_ple_numanode_assign_policy=share_band

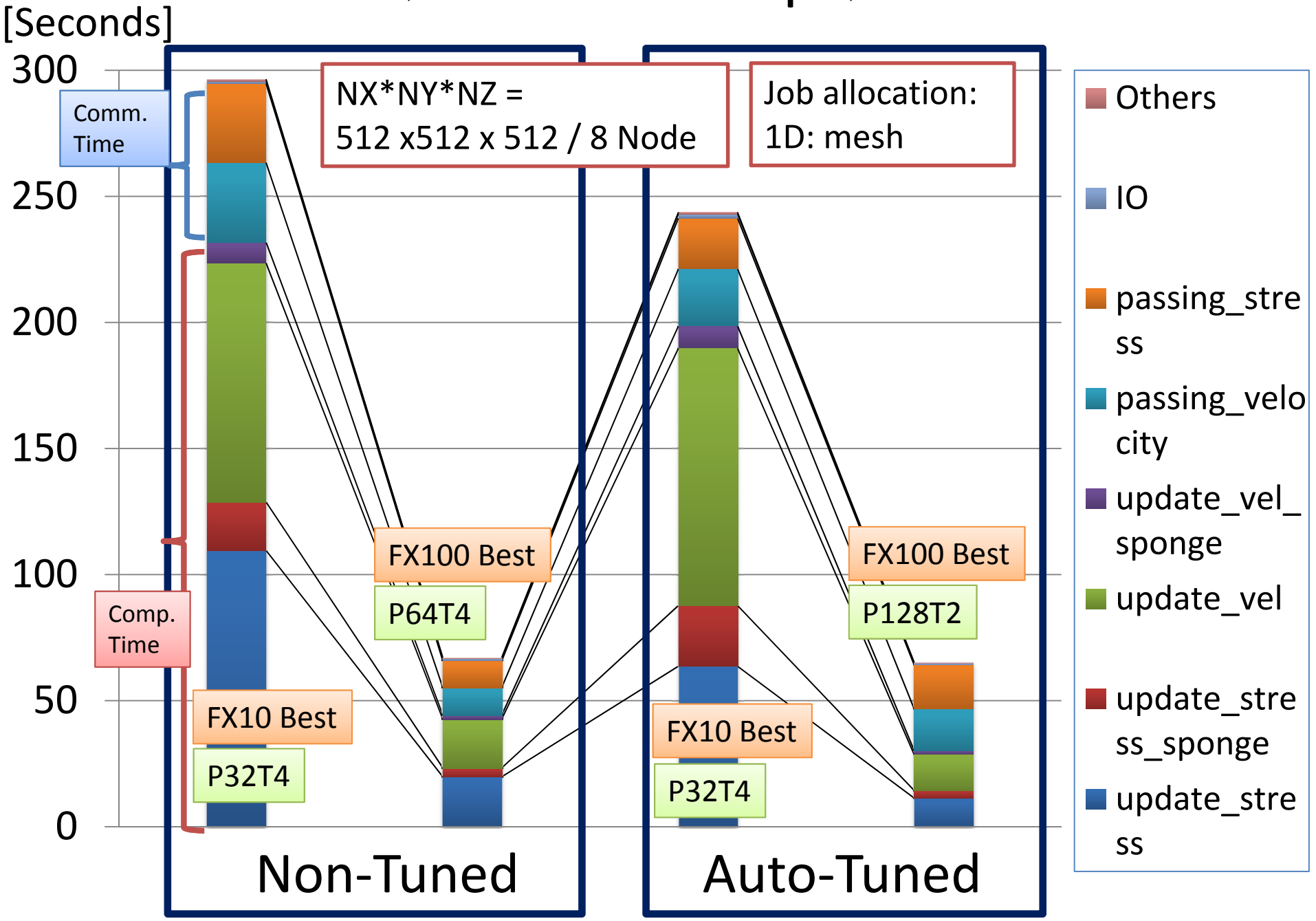
Whole Time（2000 time steps）: FX10



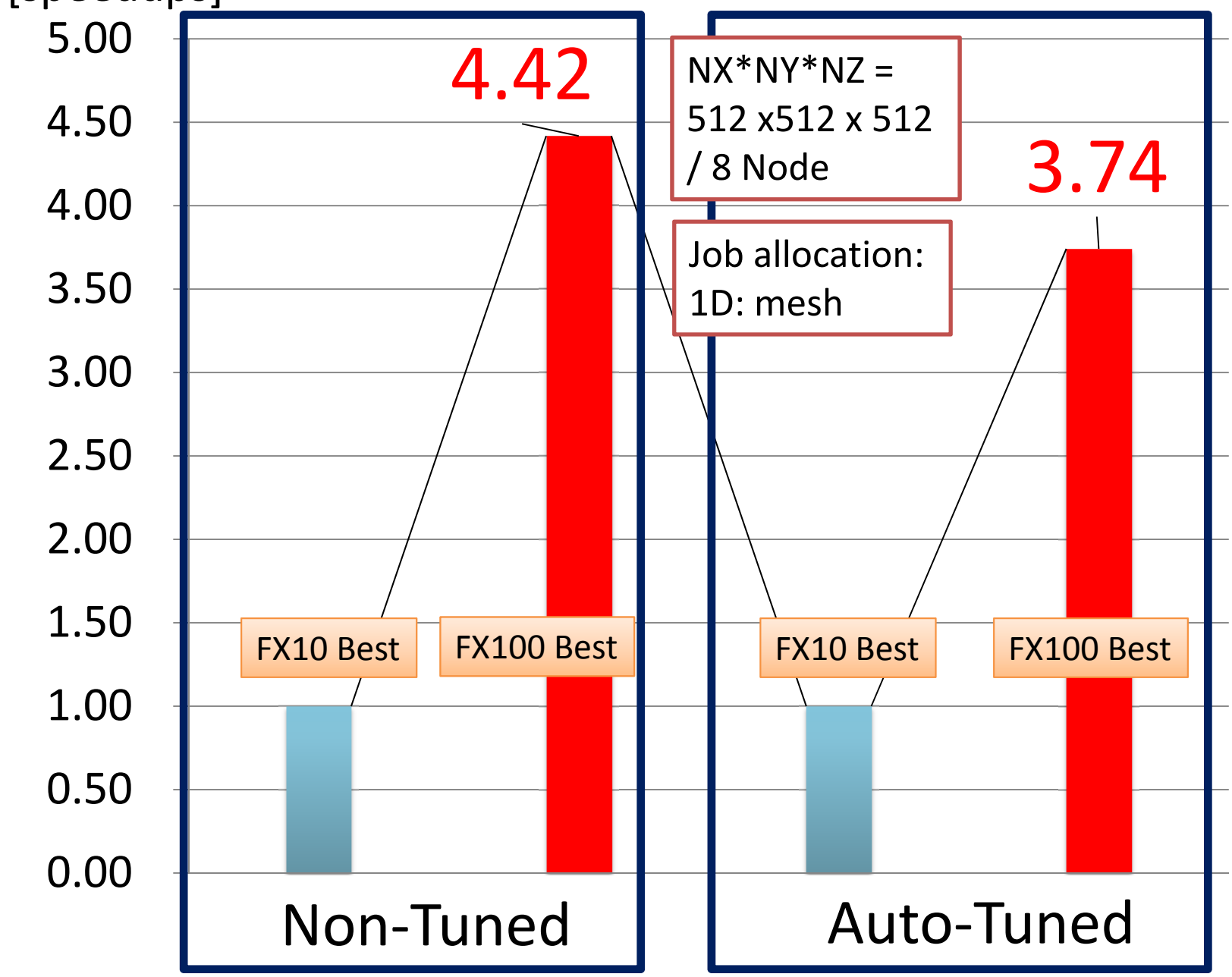
Whole Time（2000 time steps）:FX100

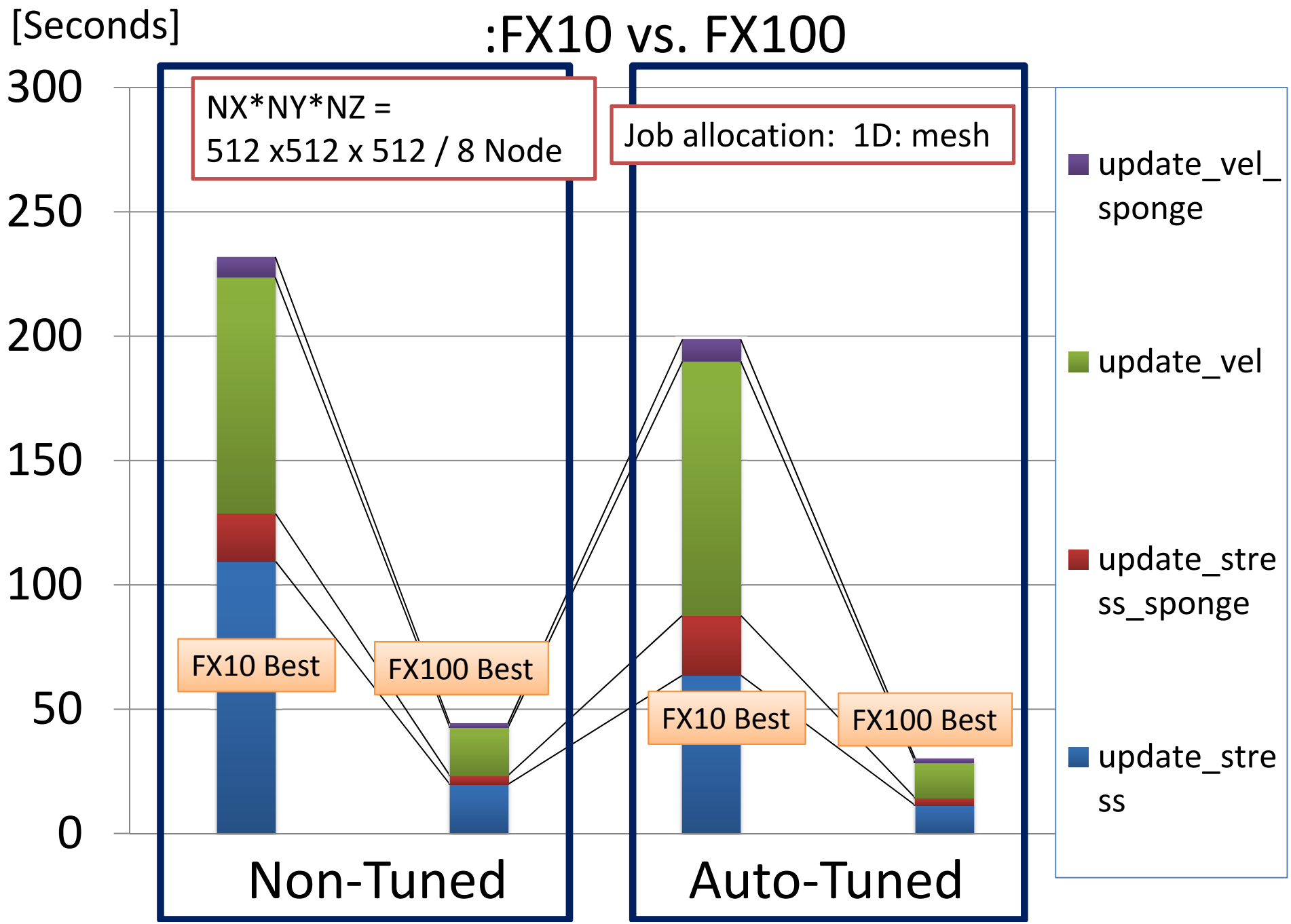


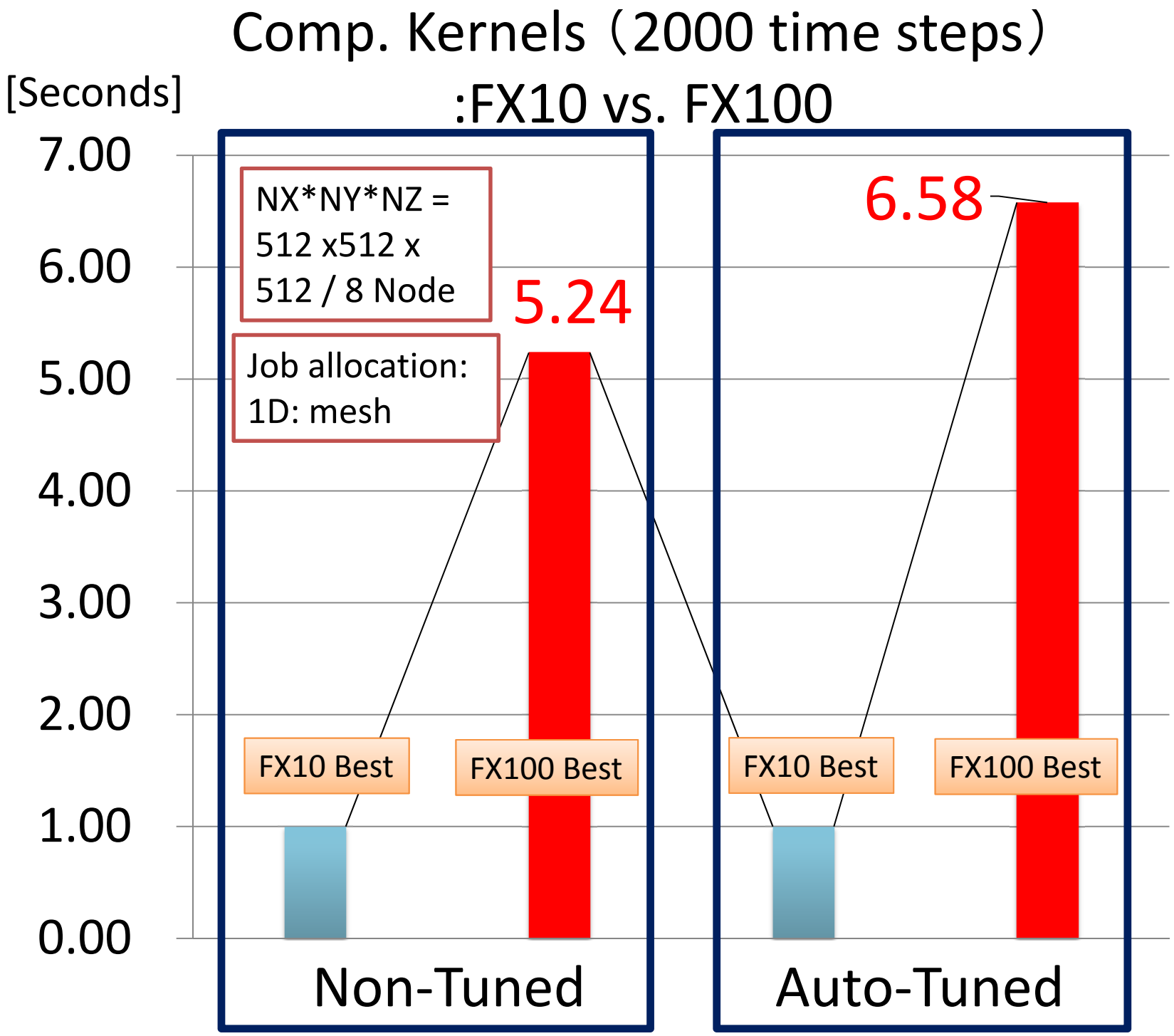
Whole Time（2000 time steps）:FX10 vs. FX100



Whole Time（2000 time steps）:FX10 vs. FX100




Comp. Kernels（2000 time steps）
:FX10 vs. FX100
[Seconds]
300
250
200
150
100
50
0
NX*NY*NZ =
512 x512 x 512 / 8 Node
Job allocation: 1D: mesh
FX10 Best
FX100 Best
FX10 Best
FX100 Best
Non-Tuned
Auto-Tuned
update_vel_sponge
update_vel
update_stress_sponge
update_stress


Comp. Kernels（2000 time steps）
:FX10 vs. FX100
[Seconds]
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
NX*NY*NZ =
512 x512 x
512 / 8 Node
Job allocation:
1D: mesh
5.24
6.58
FX10 Best
FX100 Best
FX10 Best
FX100 Best
Non-Tuned
Auto-Tuned

Originality (AT Languages)

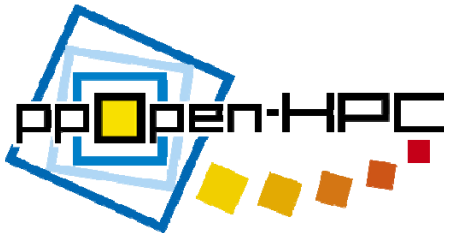
AT Language / Items	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8
ppOpen-AT	**OAT Directives**	✔	✔	✔	✔	✔		**None**
Vendor Compilers	Out of Target			Limited				-
Transformation Recipes	Recipe Descriptions	✔					✔	ChiLL
POET	Xform Description	✔					✔	POET translator, ROSE
X language	Xlang Pragmas	✔					✔	X Translation, '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[6] ,and loop collapses to the split loops[6),7),8)] .
#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.**

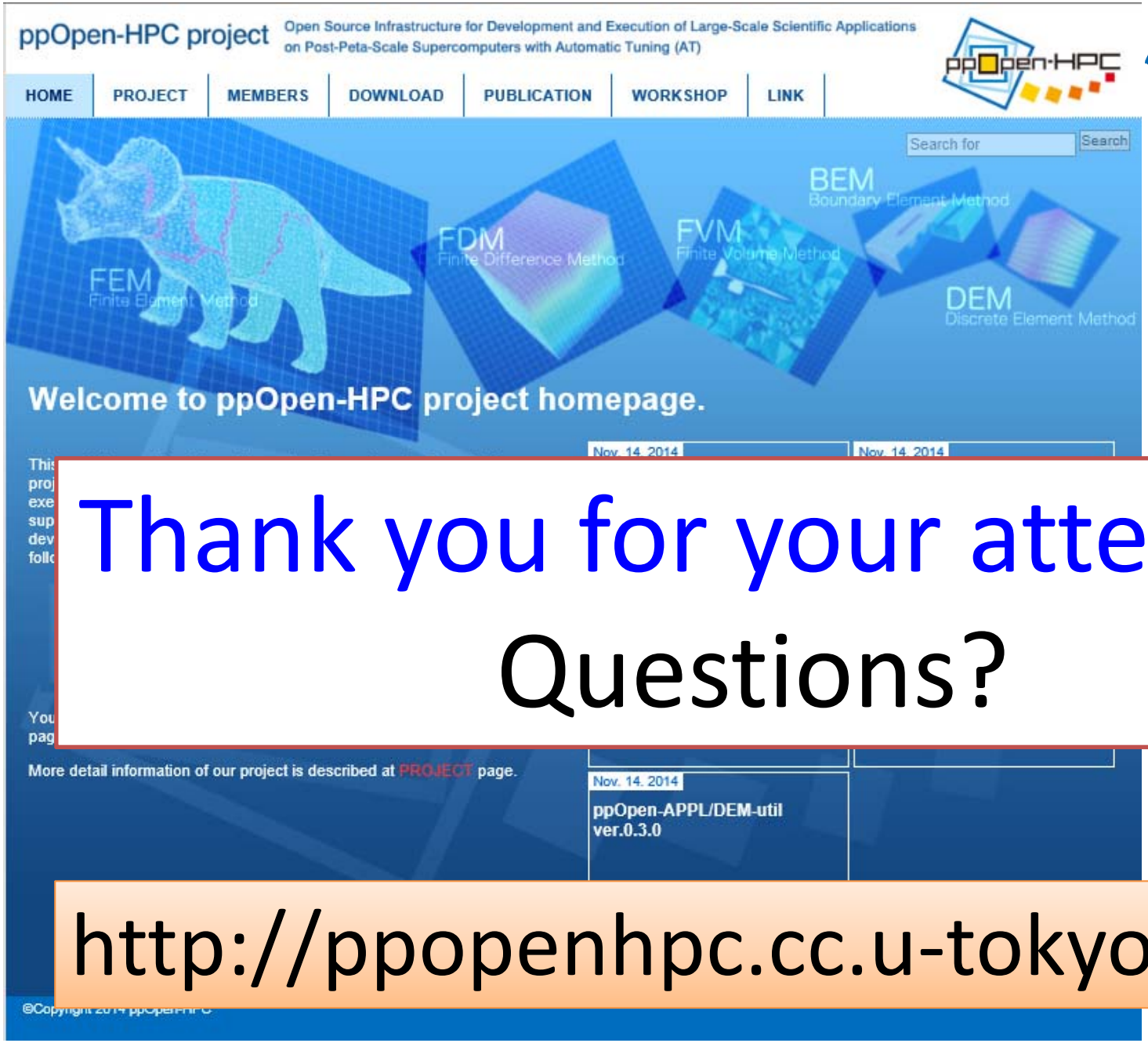
話の流れ

- 背景
- ポストムーア時代の課題例
- 事例：AT言語 *ppOpen-AT* とFDMコードへの適用
  - –FX100を用いた性能評価
- おわりに

おわりに



- ポストムーア時代には、FLOPSの増加よりも、相対的にBYTESが増加すると予想
- もしもそうなら、数値計算アルゴリズム、特に、ノード内のアルゴリズムに対して、FLOPS から BYTESへのパラダイムの転換が必要
- B/F値の変化に応じて、最適な実装が変わることを示し、この根拠とした。



Thank you for your attention!

Questions?

http://ppopenhpc.cc.u-tokyo.ac.jp/