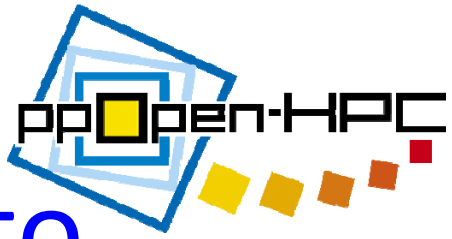


Auto-tuning Towards to Post Moore's Era: Adapting a new concept from FLOPS to BYTES

Takahiro Katagiri

(Information Technology Center, Nagoya University)

Collaborative work with

Masaharu Matsumoto and Satoshi Ohshima

(Information Technology Center, The University of Tokyo)

First International Workshop on Deepening Performance Models for Automatic Tuning (DPMAT)
Date: September 7th (Wed), 2016, 14:00-14:30
Place: Room IB-014, IB Building (Integrated Building), Higashiyama Campus, Nagoya University

Outline

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

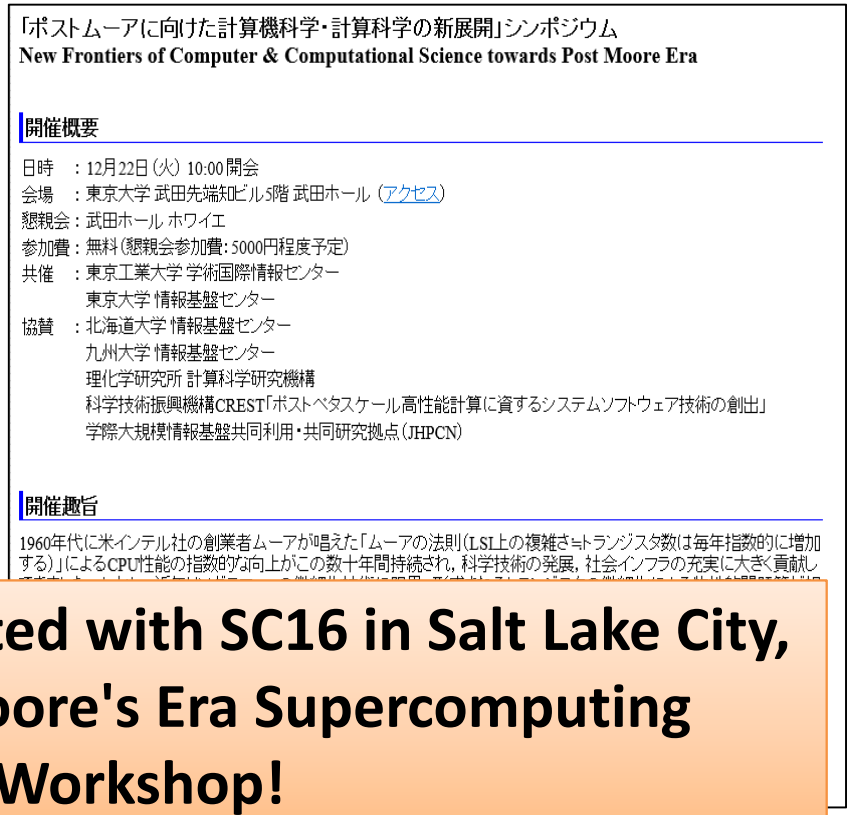
Outline

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

Symposium on “New Frontiers of Computer & Computational Science towards Post Moore Era ”

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

- Planned by:
  Prof. Satoshi Matsuoka @TITECH
- December 22th, 2015@U.Tokyo
  - Hosted by:
    ITC, U.Tokyo and GSIC, TITECH
  - Co-hosted by:
    ITC, Hokkaido U., ITC, Kyusyu U.,
    AICS, RIKEN, JST CREST, JHPCN
- Targets:
  - Hardware
  - System Software
  - Algorithm & Applications
    (PI: kengo Nakajima@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
　　理化学研究所 計算科学研究機構
　　科学技術振興機構CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」
　　学際大規模情報基盤共同利用・共同研究拠点（JHPCN）

開催趣旨

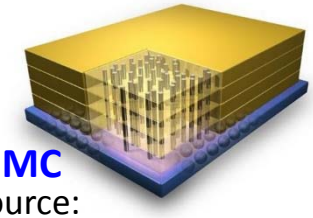
1960年代に米インテル社の創業者ムーアが唱えた「ムーアの法則（LSI上の複雑さ≒トランジスタ数は毎年指数的に増加する）」によるCPU性能の指数的な向上がこの数十年間持続され，科学技術の発展，社会インフラの充実に大きく貢献し

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

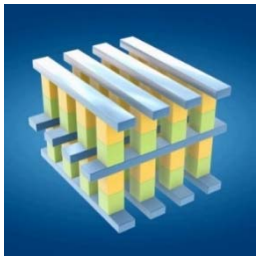
AT Technologies in Post Moore's Era

- It is expected that Moore's Low is broken around end of 2020.
  - End of "One-time Speedup": Many Cores, Wiring miniaturization to reduce power.

  → It cannot increase FLOPS inside node.
- However, memory bandwidth inside memory can increase by using "3D Stacking Memory" Technologies.
- 3D Stacking Memory:
  - It can increase bandwidth for Z-Direction (Stacking distance) , and keeping access latency (→ High performance)
  - It can be low access latency for X-Y directions.
- Access latency between nodes goes down, but bandwidth can be increased by optical interconnection technology.
- We need to take care of new algorithms with respect to ability of data movements.



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



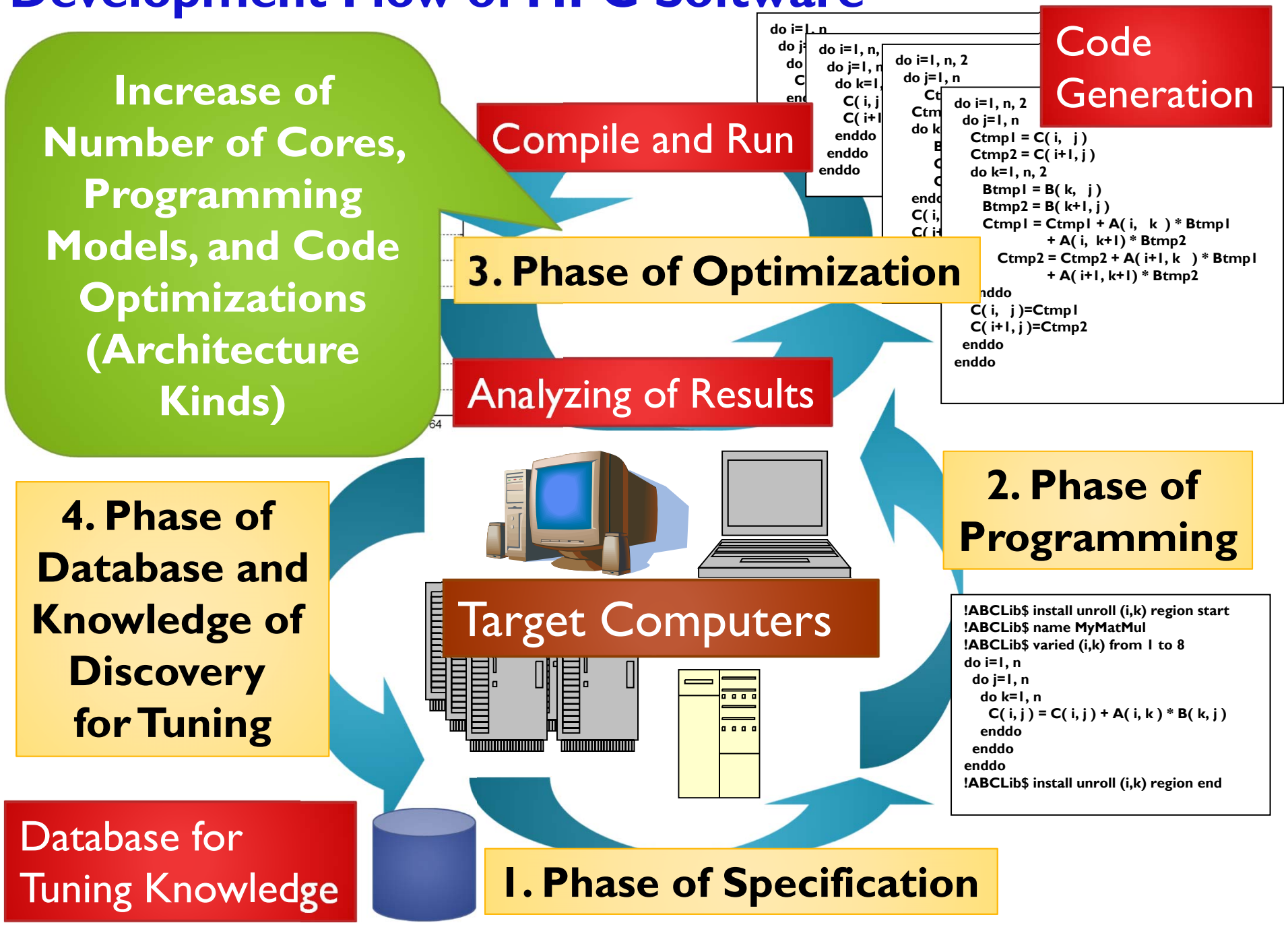
Intel 3D Xpoint
Source: http://www.theregister.co.uk/2015/07/28/intel_micron_3d_xpoint/

Reconstruction of algorithms w.r.t.
- Increase memory bandwidth
- Increase local memory amounts (caches)
- Non-Uniform memory accesses latency

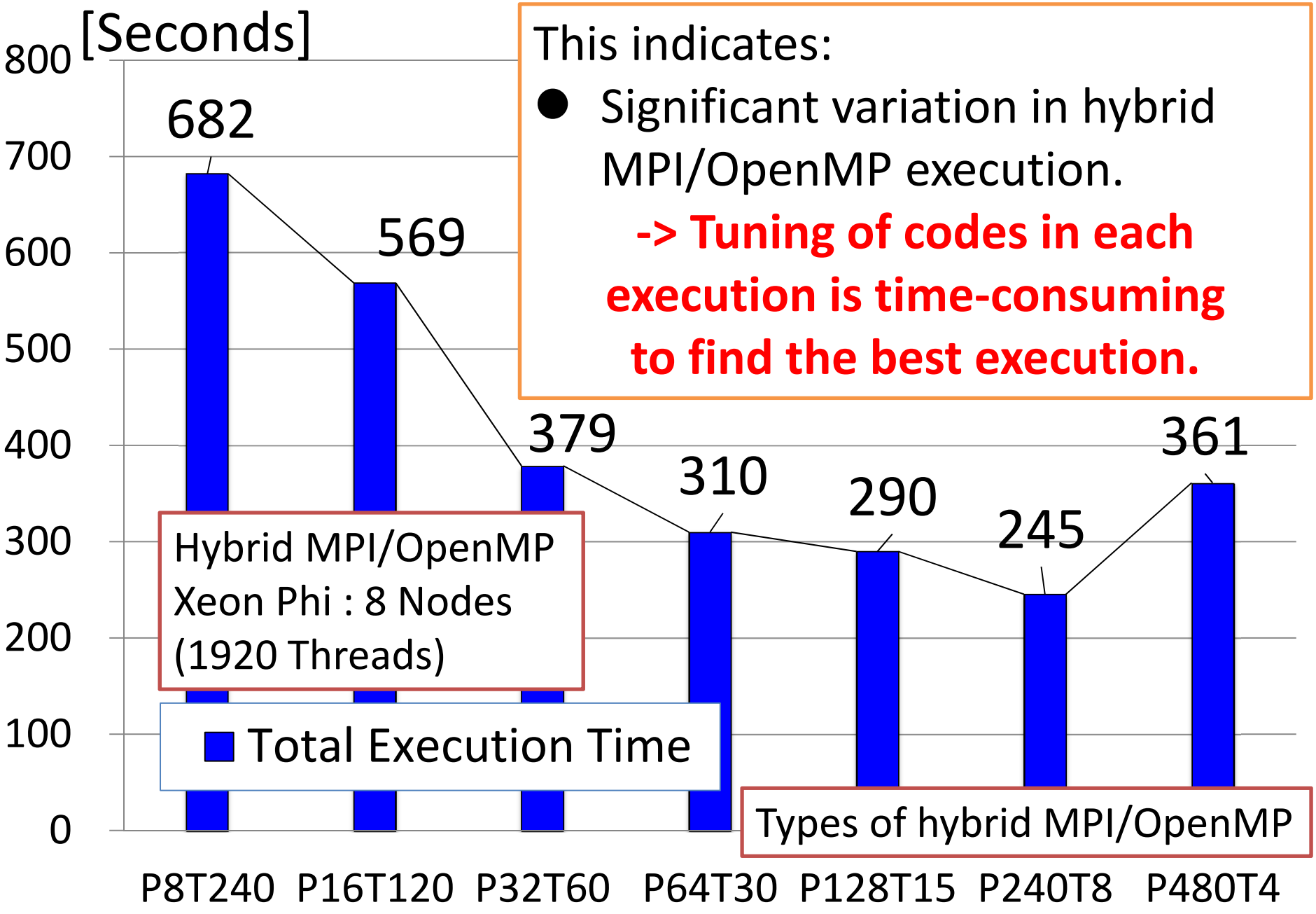
Issues in AT Technologies.

- Hierarchical AT Methodology
- Reducing Communication Algorithm.
- New algorithms and code (algorithm) selection utilizing high bandwidth ability.
  - **Rethink classical algorithms.**
  - **Non-Blocking Algorithms.**
  - **From explicit method to implicit method.**
  - **Out of Core algorithms (Out of main memory)**

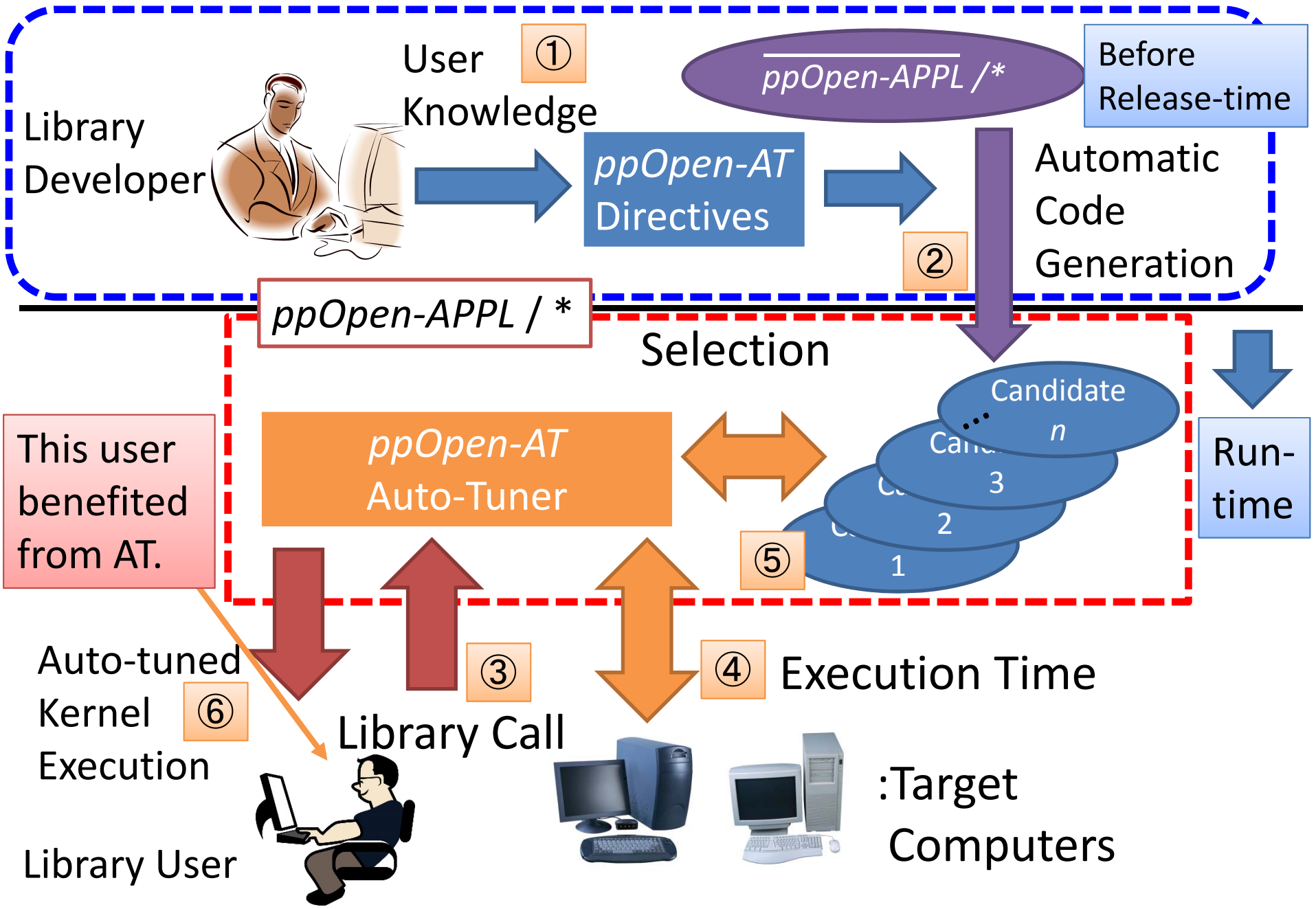
Development Flow of HPC Software



A Motivating Example (An simulation based on FDM)



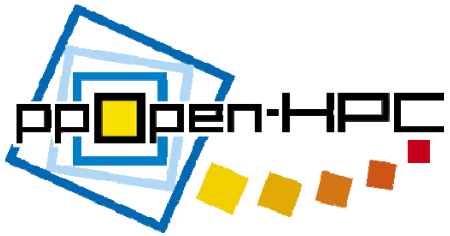
ppOpen-AT System (Based on FIBER [2),3),4),5)])



Outline

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

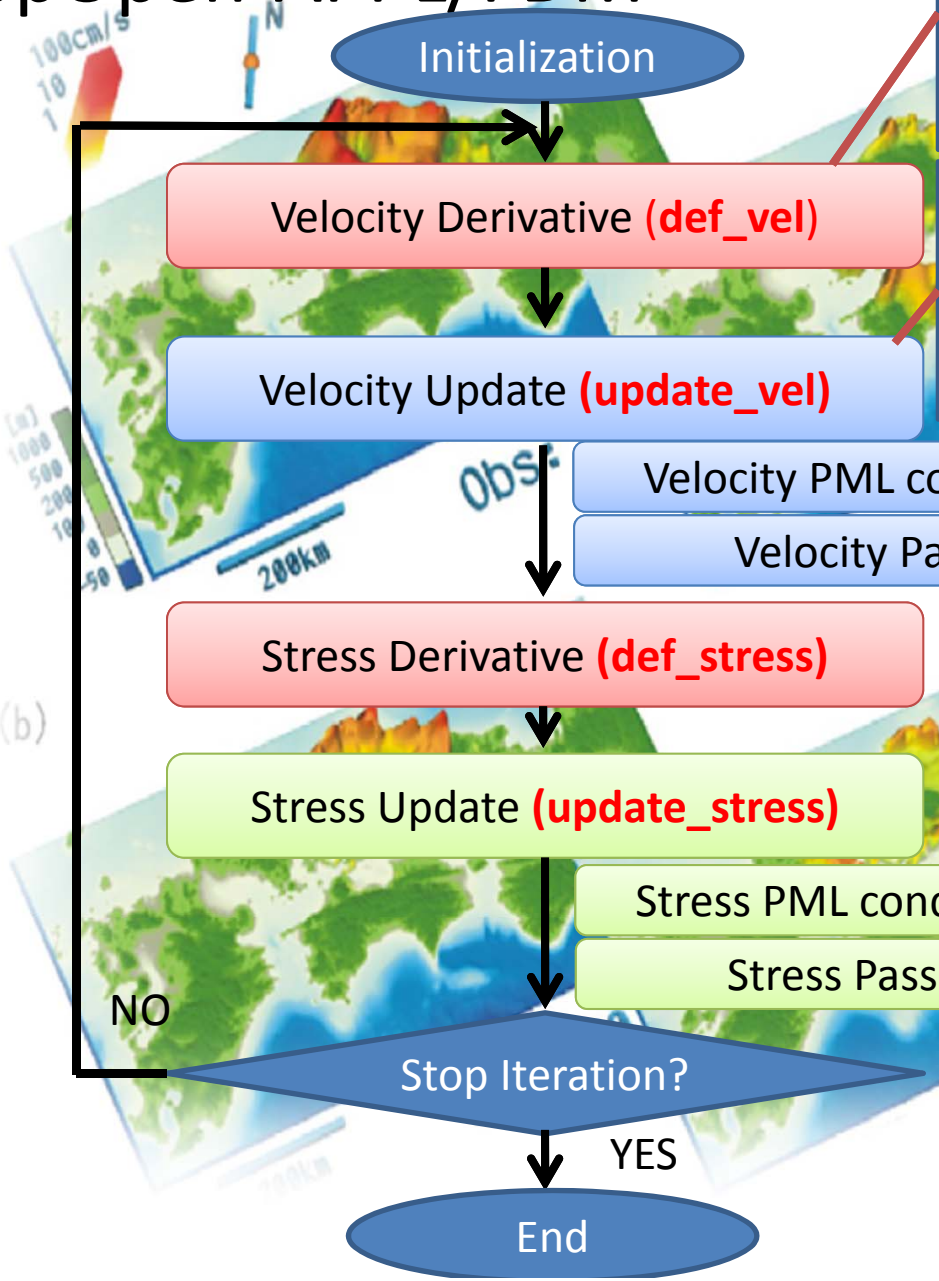
Target Application



- **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

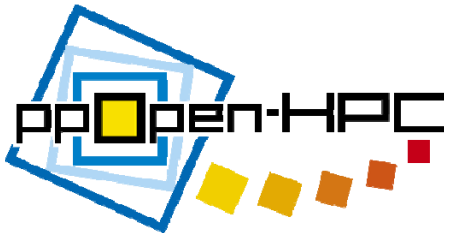


- Space difference by FDM.

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

- Explicit time expansion by central difference.

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



AT WITH LOOP TRANSFORMATION

Target Loop Characteristics

- Triple-nested loops

```
!$omp parallel do
do k = NZ00, NZ01
  do j = NY00, NY01
    do i = NX00, NX01
      <Codes from FDM>
    end do
  end do
end do
!$omp end parallel do
```

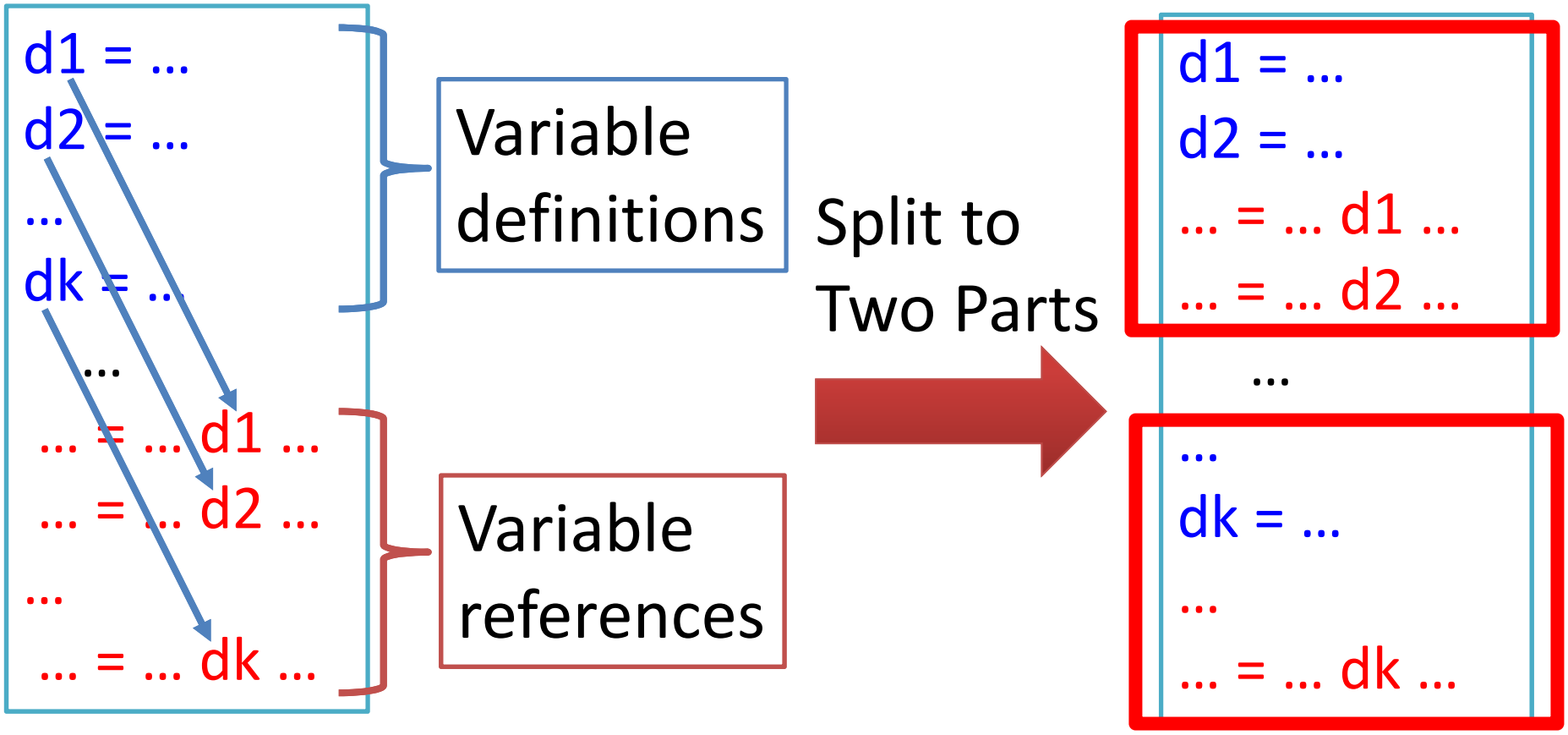
OpenMP directive to the outer loop (Z-axis)

Loop lengths are varied according to problem size, the number of MPI processes and OpenMP threads.

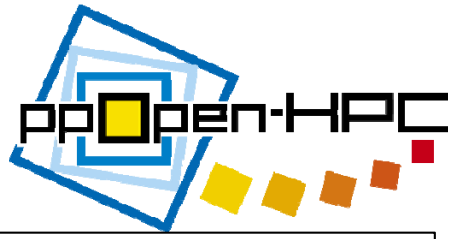
The codes can be separable by loop split.

What is separable codes?

- Variable definitions and references are separated.
- There is a flow-dependency, but no data dependency between each other.



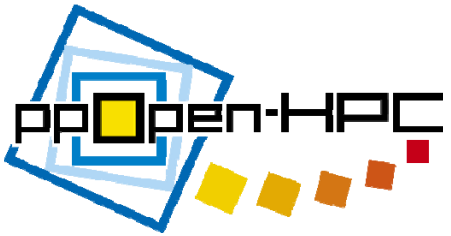
Original Code




```
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
```

A Flow Dependency

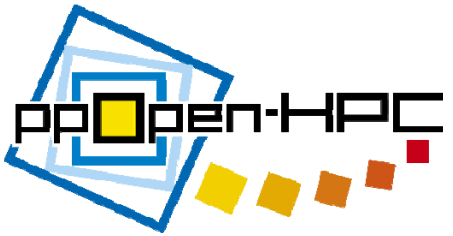
Loop Collapse – One dimensional



Merit: Loop length is huge.
This is good for OpenMP thread parallelism.

```
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
```

Loop Collapse – Two dimensional



Merit: Loop length is huge.
This is good for OpenMP thread parallelism.

This I-loop enables us an opportunity of pre-fetching

```
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
```

Loop Split with Re-Computation

```
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+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
```

Re-computation is needed.
⇒Compilers do not apply it without directive.

Perfect Splitting

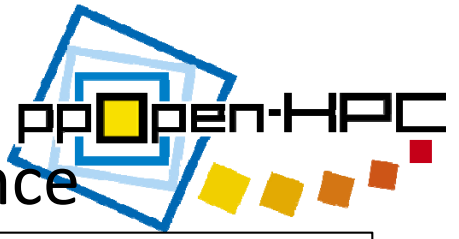
```
DO K = 1, NZ
DO J = 1, NY
DO I = 1, NX
   RL  = LAM (I,J,K)
   RM  = RIG (I,J,K)
   RM2 = RM + RM
   RLTHETA  = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
   QG  = ABSX(I)*ABSY(J)*ABSZ(K)*Q(I,J,K)
   SXX (I,J,K) = ( SXX (I,J,K)  + (RLTHETA + RM2*DXVX(I,J,K))*DT )*QG
   SYY (I,J,K) = ( SYY (I,J,K) 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

Perfect Splitting

```
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;
```

ppOpen-AT Directives

: Loop Split & Collapse with data-flow dependence




```
!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
```

Specify Loop Split and Loop Fusion

Re-calculation is defined.

Loop Split Point

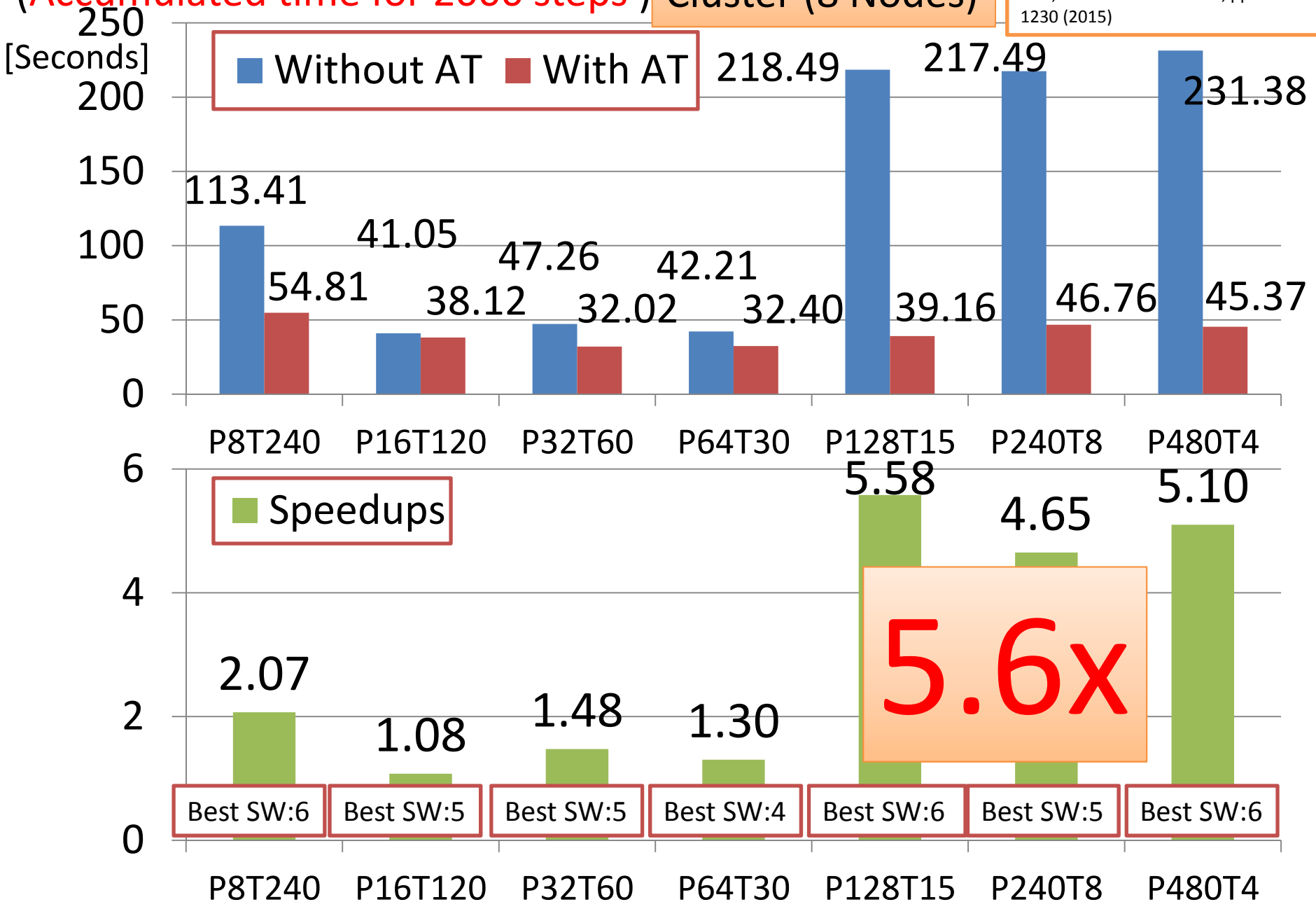
Using the re-calculation is defined.

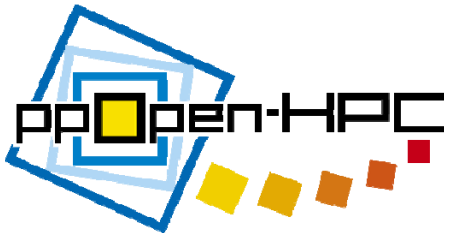
AT Effect (update_stress)

(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)





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

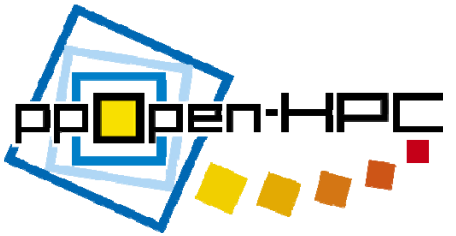
```
    DXVX1 = DXVX0;  DYVY1 = DYVY0
    DZVZ1 = DZVZ0;  D3V3  = DXVX1 + DYVY1 + DZVZ1
     SXX (I,J,K) = SXX (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
     SYY (I,J,K) = SYY (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
     SZZ (I,J,K) = SZZ (I,J,K) &
       + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1)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

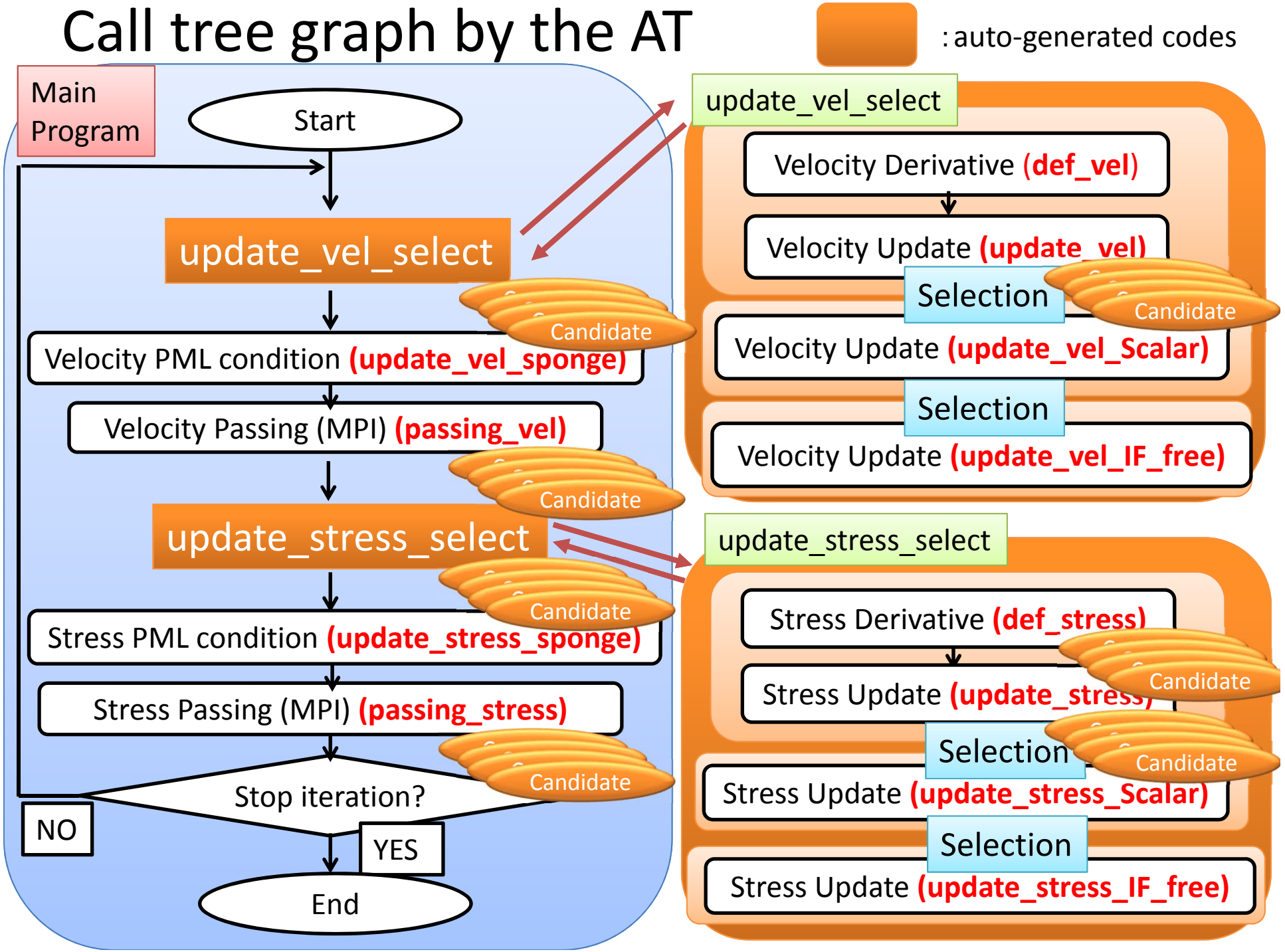
Lower Code

```
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
```

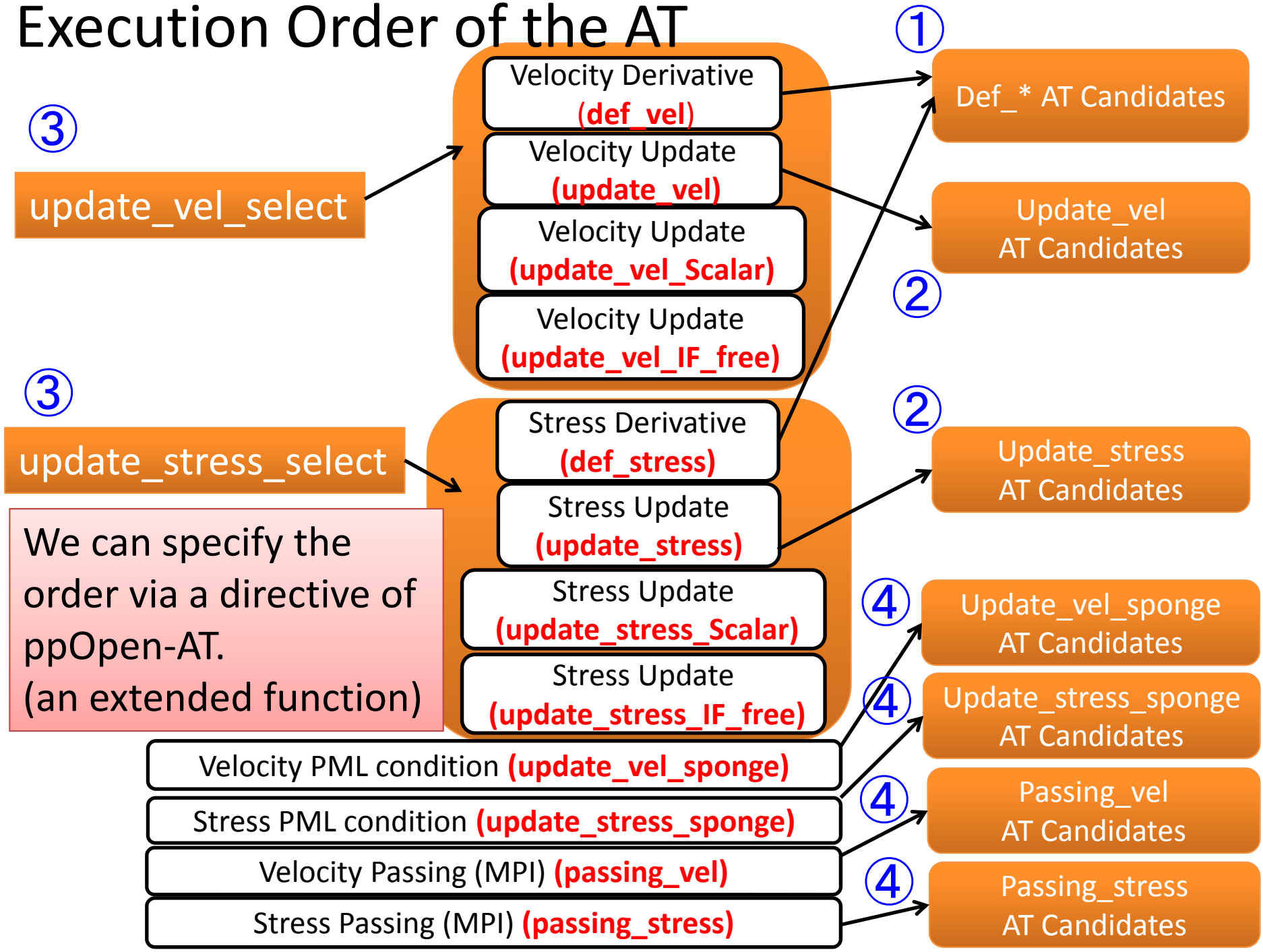
```
subroutine ppohFDM_pdiffx3_p4(....)
....
!OAT$ install LoopFusion region start
....
```

```
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.)

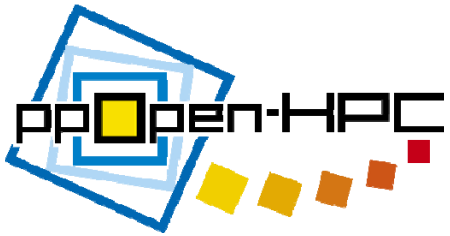
	FX10	FX100	Ratios (FX100/FX10)
Node FLOPS	236.5 GFLOPS	2 TFLOPS (single precision)	8.44x
Memory Bandwidth	85 GB/S	480 GB/S	5.64x
Networks	5 GB/S x2	12.5 GB/S x2	2.5x

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



RELATED WORK

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 Reci	Recipe	✔					✔	ChiLL
PO								POET translator, ROSE
X lang								X Translation, 'C and tcc
SP								A Script Language
ADA								Polaris Compiler Infrastructure, Remote Procedure Call (RPC)
Atun								A Monitoring Daemon
PEPPHER	(interface)							PEPPHER task graph and run-time
Xevolver	Directive Extension (Recipe Descriptions)		(✔) (Users need to define rules.)	(✔)		(✔)		ROSE, XSLT Translator

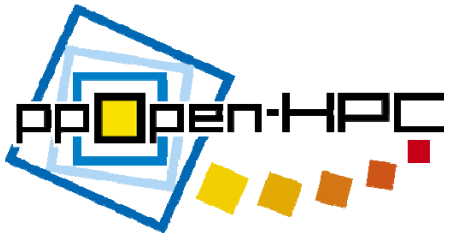
The **explicit** function of loop transformations (loop splits and collapses) and code selection **at simultaneously** is unique!

#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.**

Outline

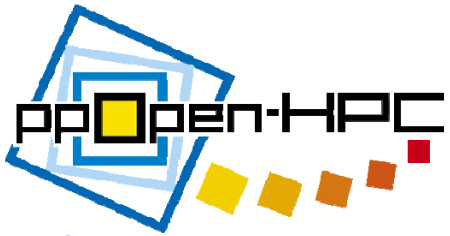
- Background
- An Auto-tuning (AT) Language: *ppOpen-AT* and Adapting AT to an FDM code
- Performance Evaluation with the FX100
- Conclusion

Conclusion



- Evaluation performance between the FX10 with a conventional memory and the FX100 with a 3D stacking memory.
- We observe effective execution in the FX100 with respect to hardware ability for memory transfer performance to that of the FX10.
- According to results of profiler, loading time for floating point data from cache is dramatically reduced in the FX100 compared to the FX10.

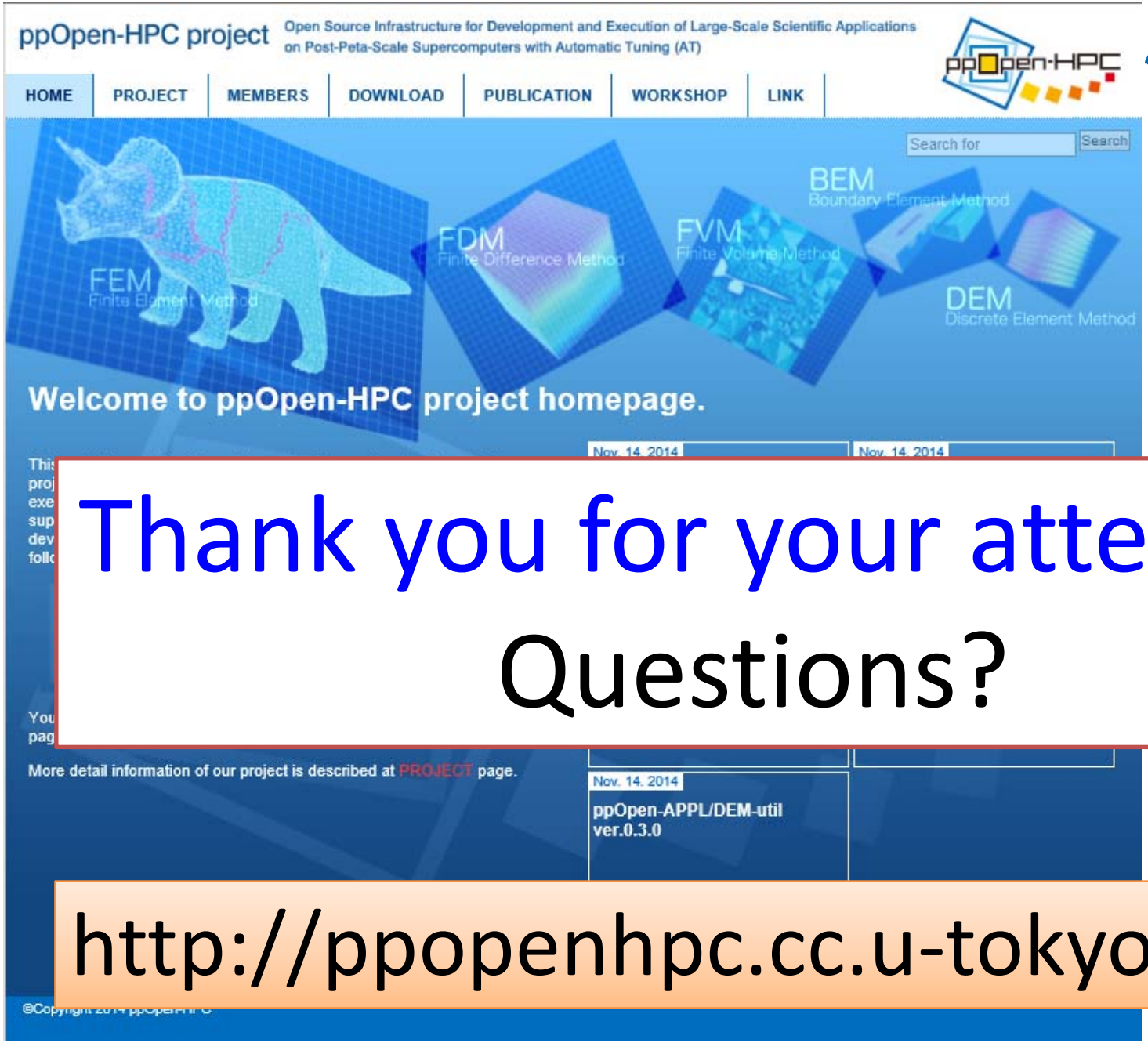
Future work



- Auto-tuning for dynamic scheduling and its chunk size.
  - **schedule(dynamic, chunk_size)** in OpenMP
  - There is a case to speedup by using it in the FX10.
    - Code selection for description with **dynamic** or **static**, and the size of **chunk_size**.
- OpenACC optimization
  - Code selection for description of “Gang”, “Vector”, “Worker” and its size of resources.

S. Ohshima, T. Katagiri, M. Matsumoto: "Utilization and Expansion of ppOpen-AT for OpenACC", Proc. of IPDPSW2016, pp.1496 - 1505 (2016)

- Performance Modeling
  - **Performance model (Black-Box model) for hierarchical AT.**
  - **D-spline model (Tanaka, Fujii, et.al@Kogakuin U.)**
  - **Surrogate model (Wang, et.al@National Taiwan U.)**



Thank you for your attention!

Questions?

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