Auto-tuning for The Era of Relatively High Bandwidth Memory Architectures

-A Discussion Based on an FDM Application-

Takahiro Katagiri

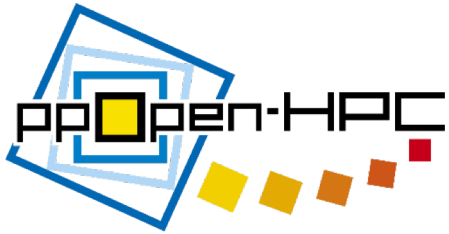
(Information Technology Center, Nagoya University)

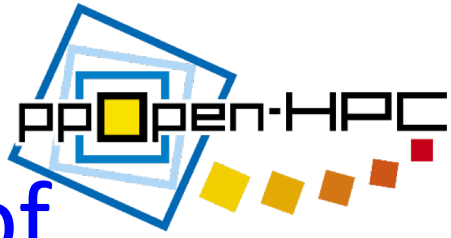
Collaborative work with

Masaharu Matsumoto and Satoshi Ohshima

(Information Technology Center, The University of Tokyo)

The Thirteenth International Workshop on Automatic Performance Tuning (iWAPT2018), May 25, 2018, JW Marriott Parq Vancouver, Vancouver, British Columbia CANADA, AT Techniques, 11:30-12:00, May 25, 2018


ppOpen-HPC



Auto-tuning for The Era of Relatively High Bandwidth Memory Architectures

-A Discussion Based on an FDM Application-

Takahiro Katagiri

(Information Technology Center, Nagoya University)

Collaborative work with

Masaharu Matsumoto and Satoshi Ohshima

(Information Technology Center, The University of Tokyo)

The Thirteenth International Workshop on Automatic Performance Tuning (iWAPT2018), May 25, 2018, JW Marriott Parq Vancouver, Vancouver, British Columbia CANADA, AT Techniques, 11:30-12:00, May 25, 2018

Outline

- Background
- An Auto-tuning (AT) Language: *ppOpen-AT* and Adapting AT to an FDM code
- An Evidence: Changes from FLOPS to BYTES
- Conclusion

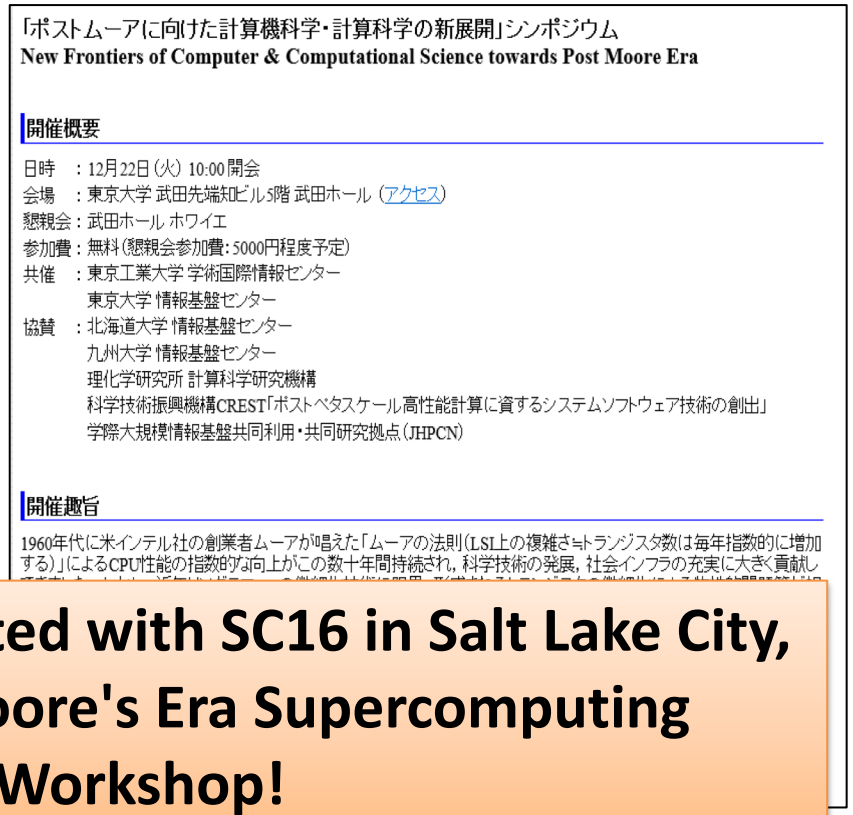
Outline

- Background
- An Auto-tuning (AT) Language: *ppOpen-AT* and Adapting AT to an FDM code
- An Evidence: Changes from FLOPS to BYTES
- 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

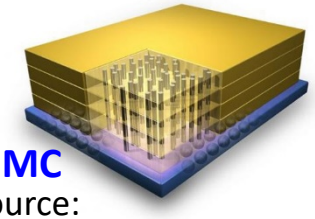
1960年代に米インテル社の創業者ムーアが唱えた「ムーアの法則（LSI上の複雑さ≒トランジスタ数は毎年指数的に増加する）」によるCPU性能の指数的な向上がこの数十年間持続され，科学技術の発展，社会インフラの充実に大きく貢献し

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

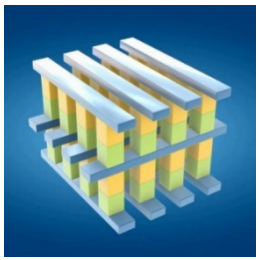
AT Technologies in Post Moore's Era

- 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

Increase of Number of Cores, Programming Models, and Code Optimizations (Architecture Kinds)

Compile and Run

Code Generation

```
do i=1, n, 2
  do j=1, n
    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. Phase of Optimization

Analyzing of Results

2. Phase of Programming

```
!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
```

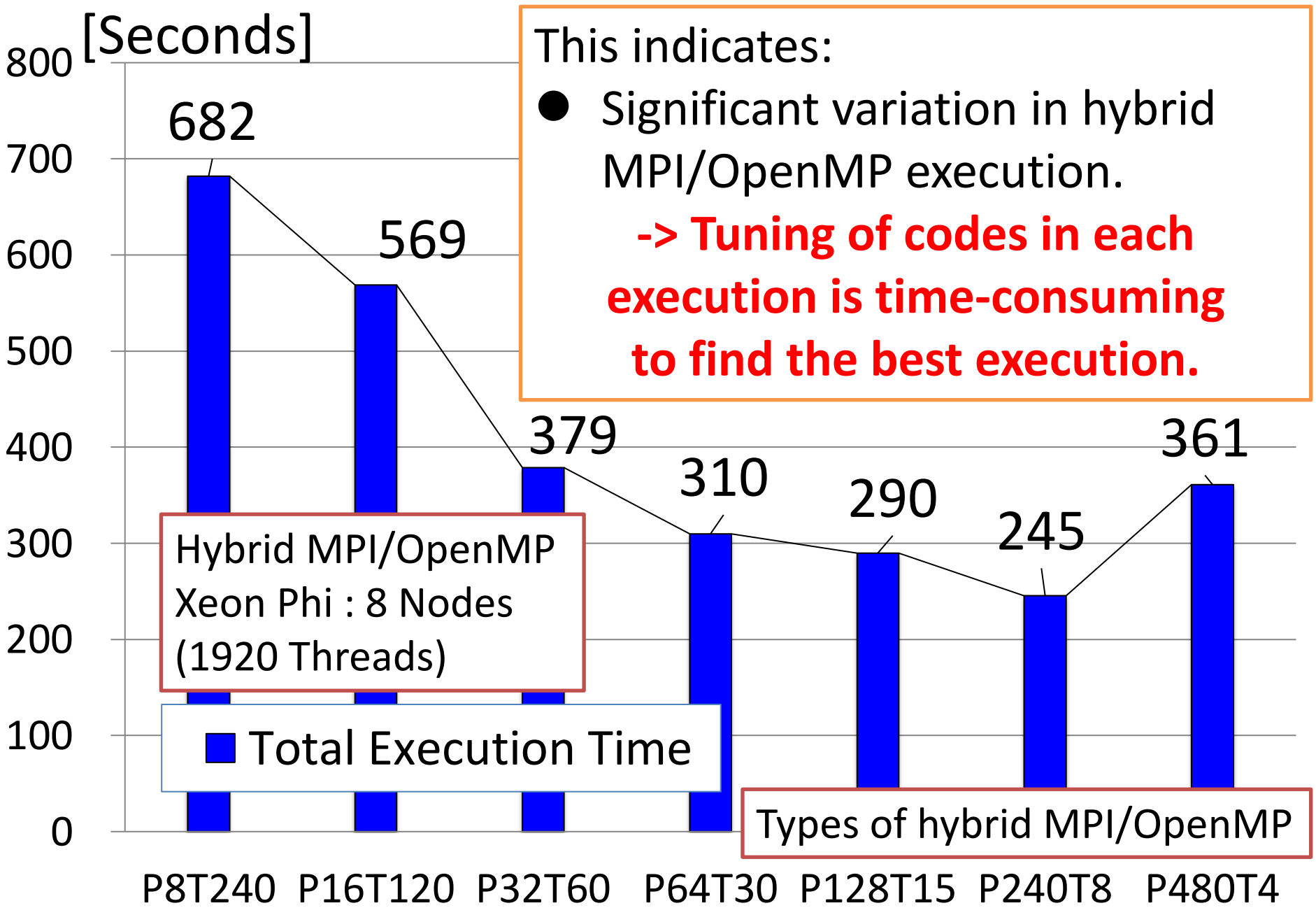
Target Computers

4. Phase of Database and Knowledge of Discovery for Tuning

Database for Tuning Knowledge

1. Phase of Specification

A Motivating Example (An simulation based on FDM)



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

Library Developer

User Knowledge ①

*ppOpen-APPL /**

Before Release-time

ppOpen-AT Directives

Automatic Code Generation

②

*ppOpen-APPL / **

Selection

Candidate *n*

3

2

1

⑤

ppOpen-AT Auto-Tuner

Run-time

This user benefited from AT.

Auto-tuned Kernel Execution ⑥

③ Library Call

④ Execution Time

Library User

:Target Computers

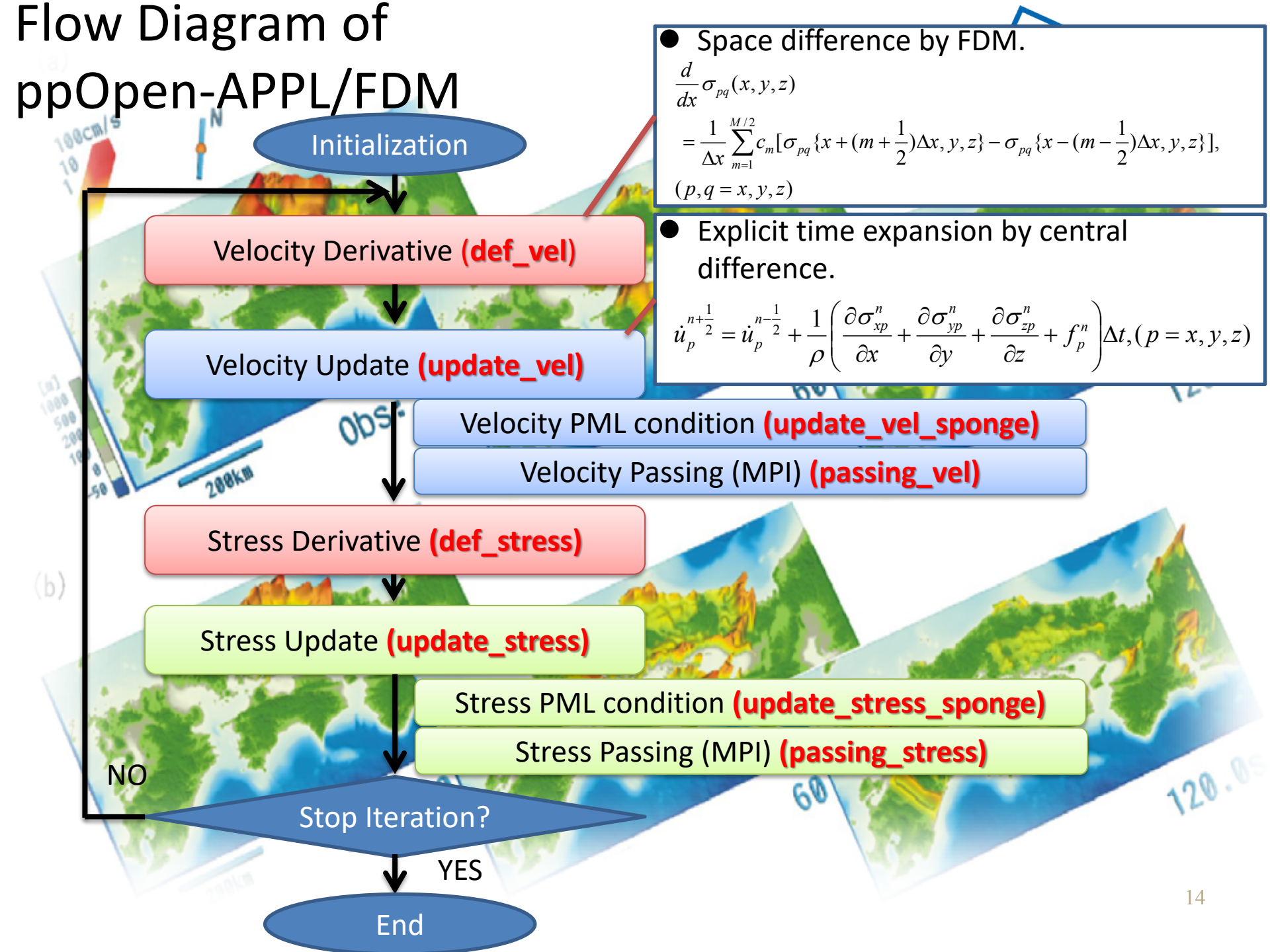
Outline

- 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
```

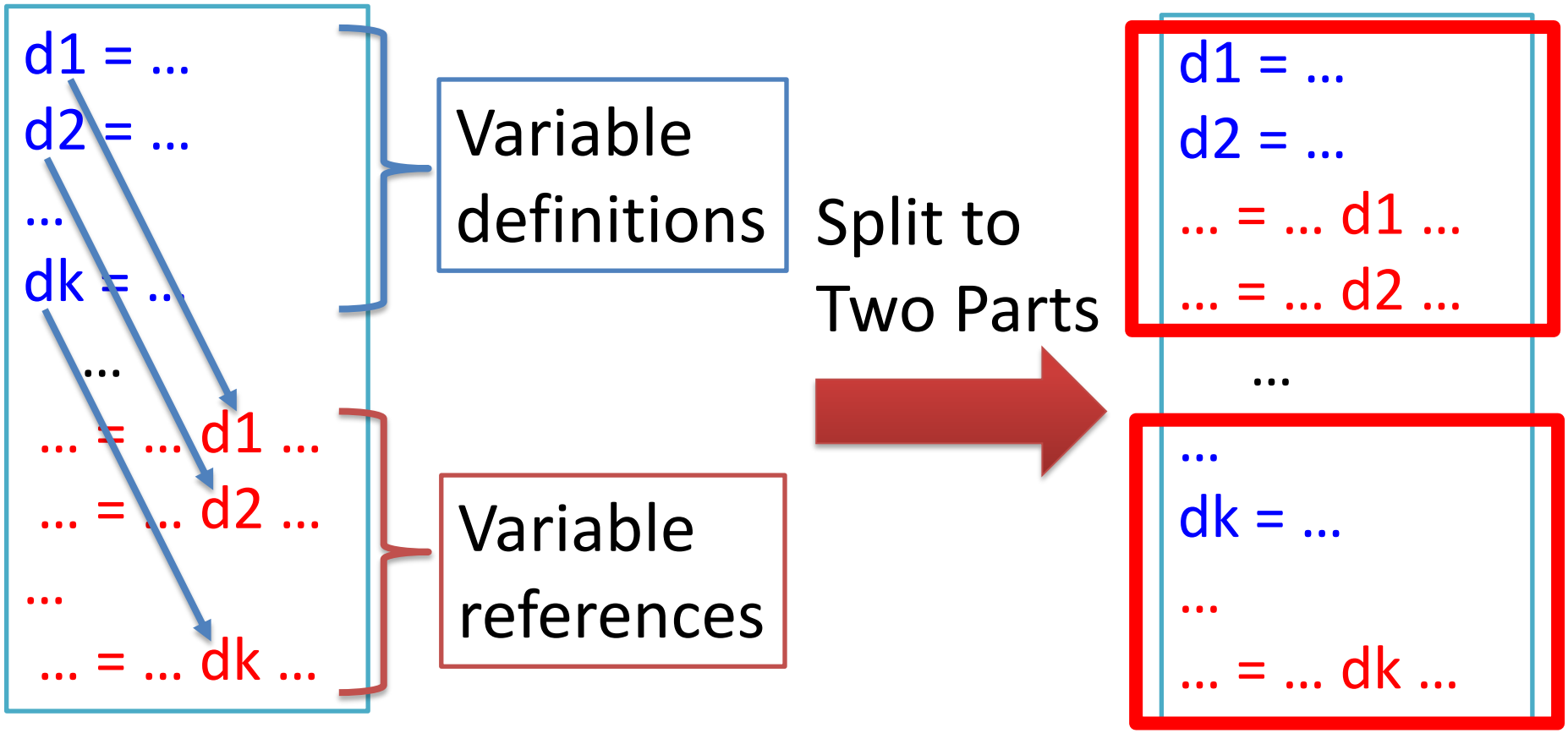
OpenMP directive to the outer loop (Z-axis)

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.

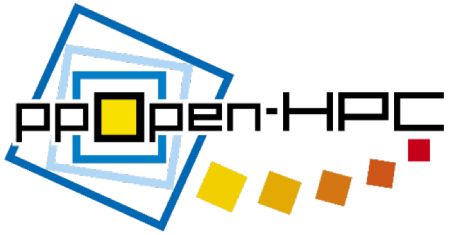


Original Code

```
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

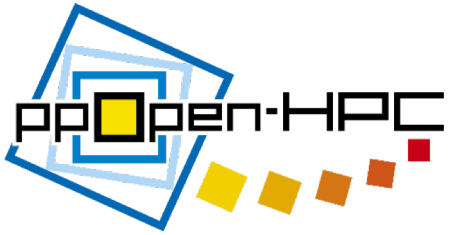
Loop Collapse – One dimensional




```
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
```

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

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

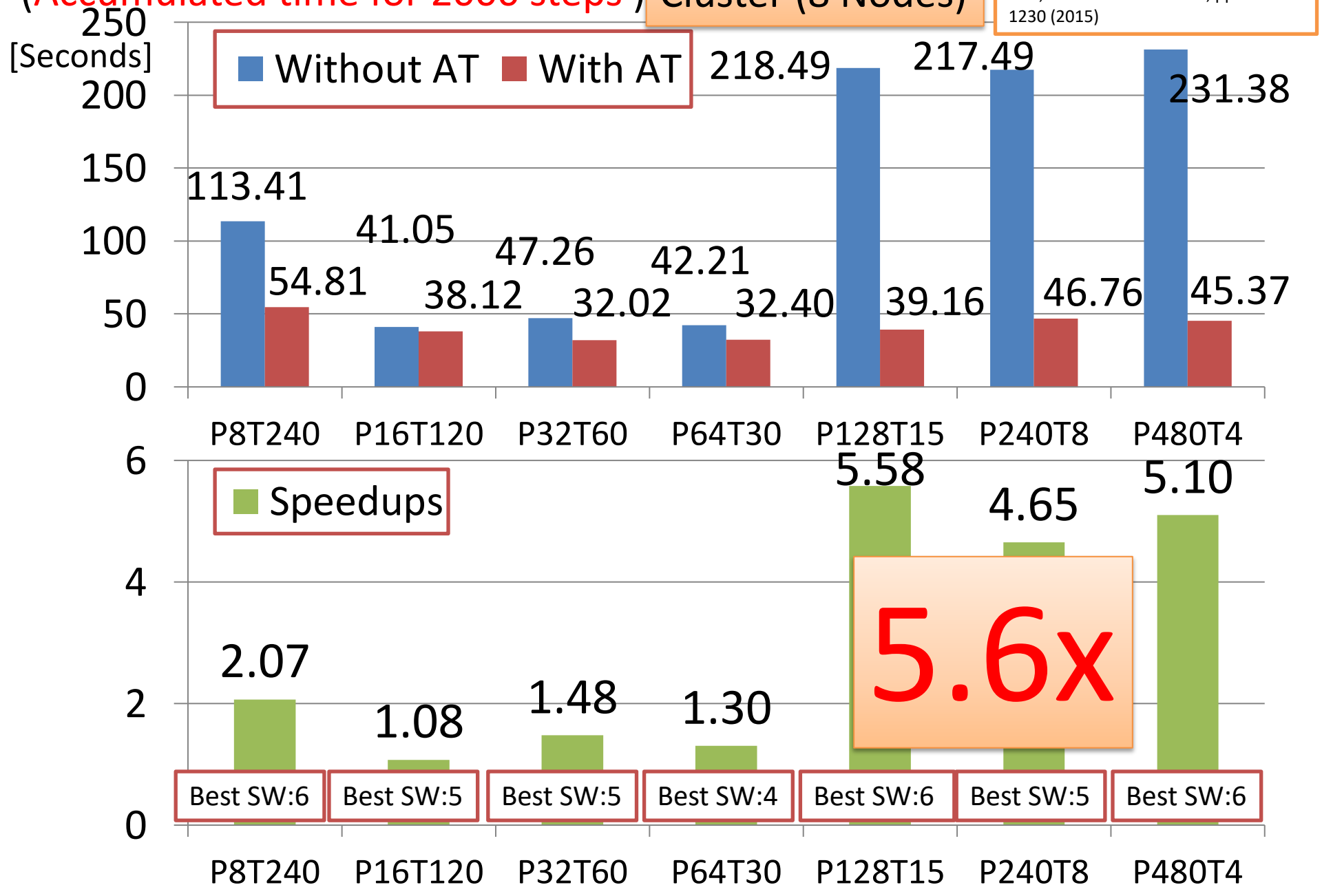
Using the re-calculation is defined.

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_ver
! 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

Fourth-order accurate central-difference scheme for velocity.

Explicit time expansion by leap-frog scheme.

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

☺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
```

```
el-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
nd do
d do
do
!$omp end parallel do
```

Fourth-order accurate central-difference scheme for velocity

Explicit time expansion by leap-frog scheme.

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
  .....
....
```

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
- An Evidence: Changes from FLOPS to BYTES
- Conclusion

FX10 AND 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)

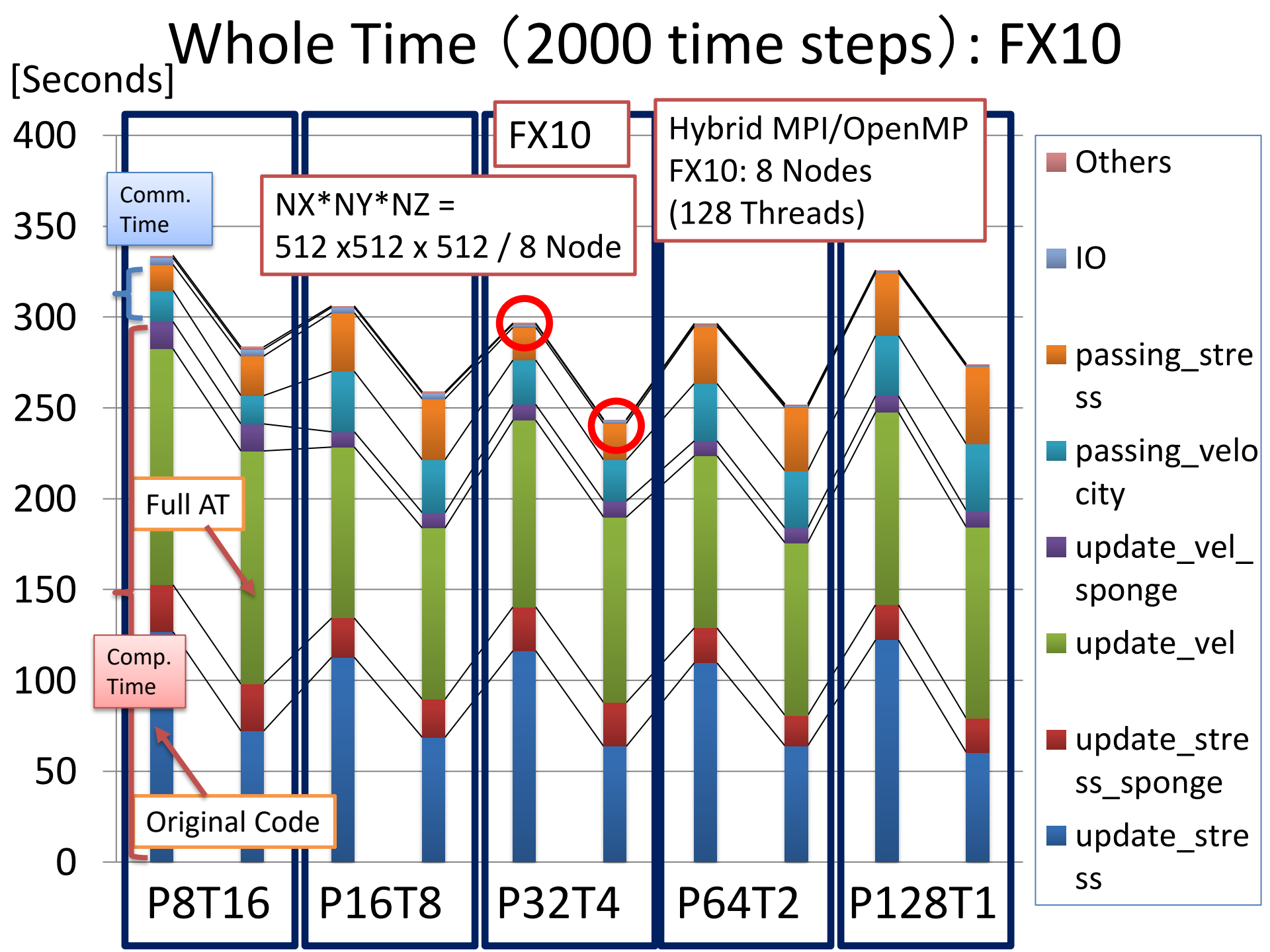
Contents		Specifications
Node	Theoretical Peak Performance	1 TFLOPS (double precision)
	#Processors (#Cores)	32 + 2 (assistant cores)
	Main Memory Amounts	32 GB
Processor	Processor Name	SPARC64 XI-fx
	Frequency	2.2 GHz
	Theoretical Peak Performance (Core)	31.25 GFLOPS

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

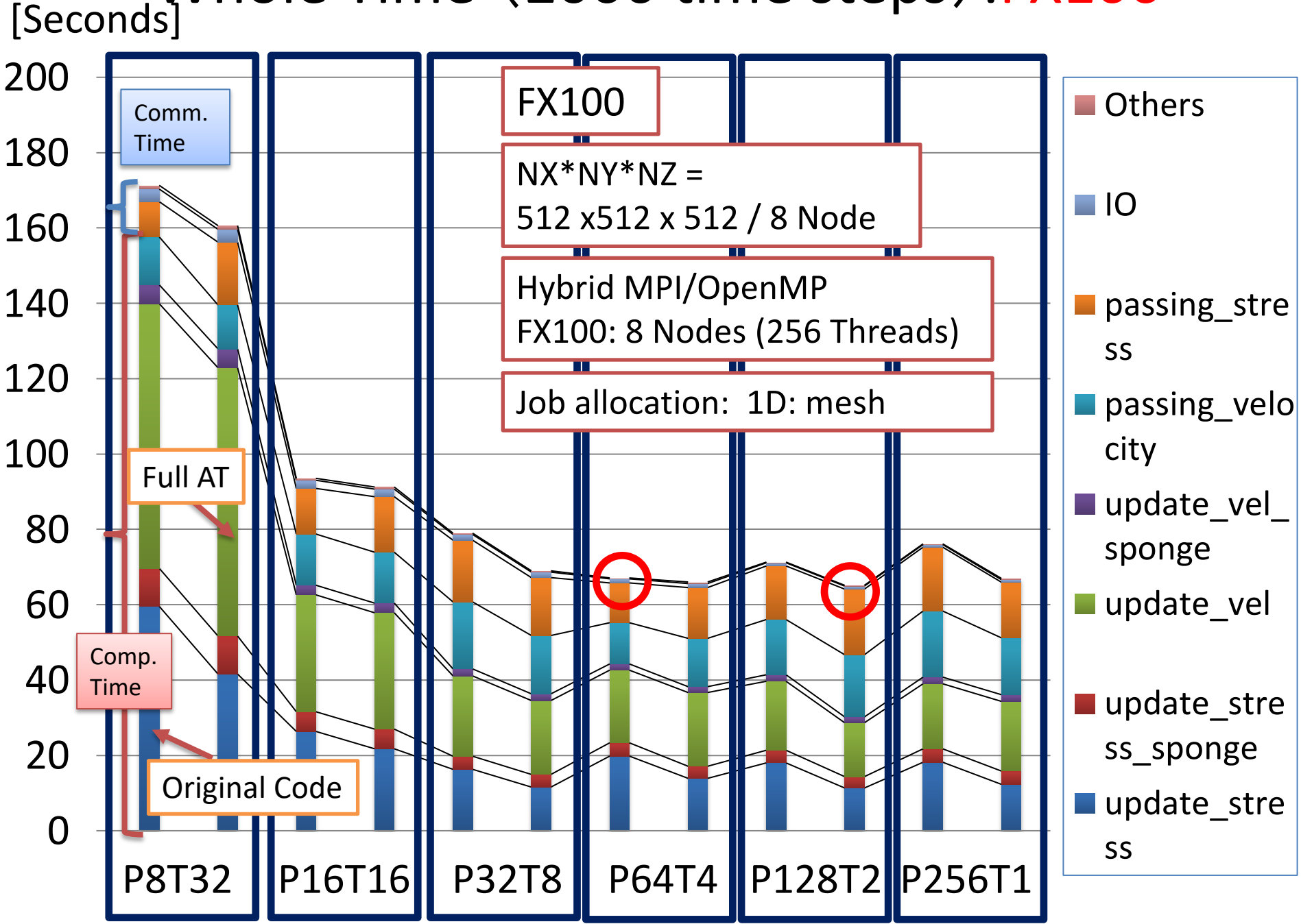
	FX10	FX100	Ratios (FX100/FX10)
Node FLOPS	236.5 GFLOPS (double precision)	1 TFLOPS (double precision)	4.22x
Memory Bandwidth	85 GB/S	480 GB/S	5.64x
Networks	5 GB/S x2	12.5 GB/S x2	2.5x
B/F	0.35	0.48	-

COMPARISON OF EXECUTION TIME BETWEEN THE FX10 AND THE FX100

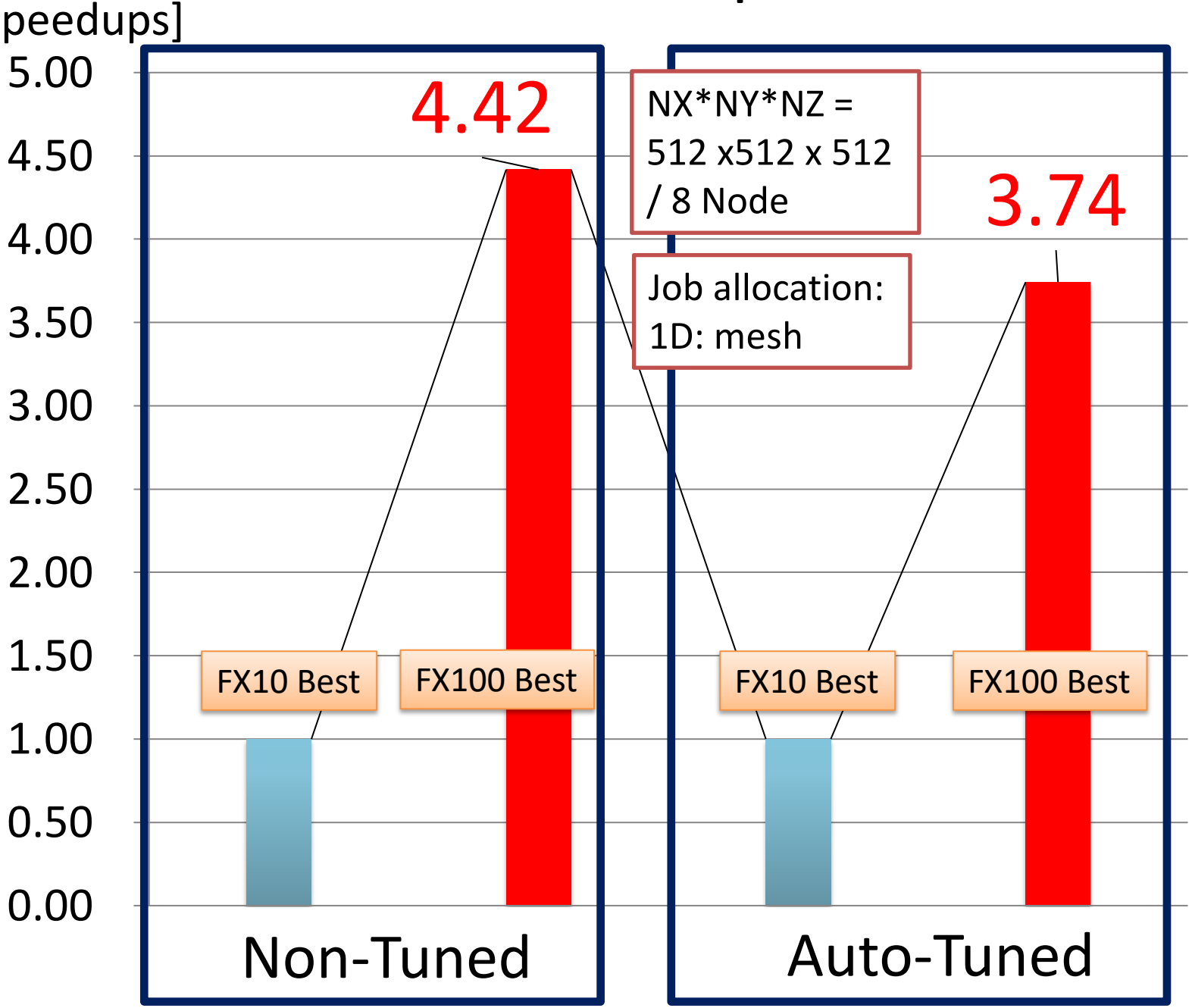
Whole Time（2000 time steps）: FX10



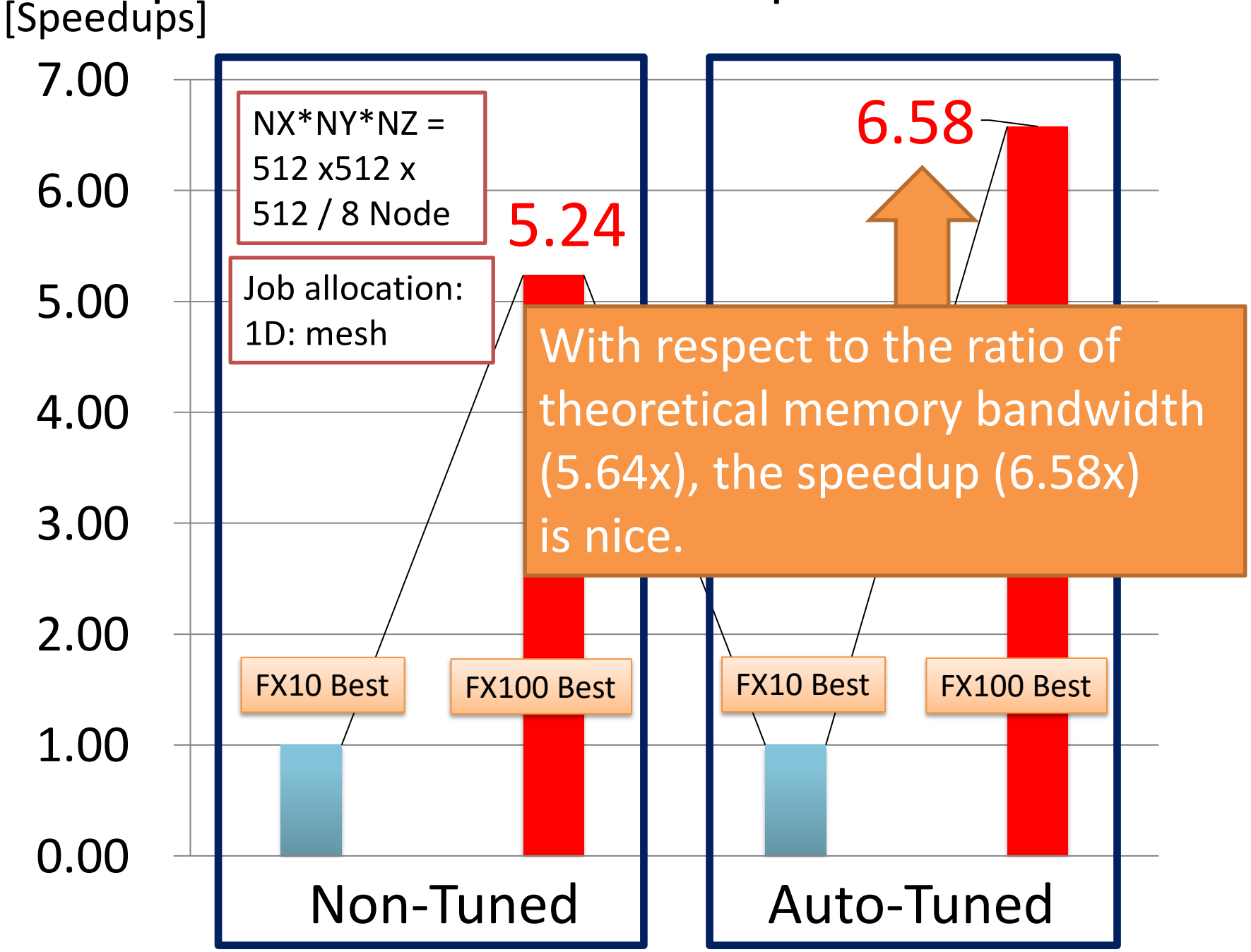
Whole Time（2000 time steps）:FX100



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



Comp. Kernels（2000 time steps）:FX10 vs. FX100



CHANGE B/F RATIOS WITH THE FX100

Changing B/F ratio by Hardware Viewpoint

- In the FX100, it has a NUMA with two sockets :
  - Each socket has:
    - 16 cores with 1011 GFLOPS (single) and 505.5 GFLOPS (double)
    - 16GB memory with maximum 240 GB/sec
    - There are 2 lines to access socket of HMC per core, which has 15 Gbit per line.
    - Up to 8 core, each core can use 2 lines.
  - Hence total bandwidths are:
    - 30 GB/sec per core (#cores <= 8)
    - 240 GB (#core > 8)

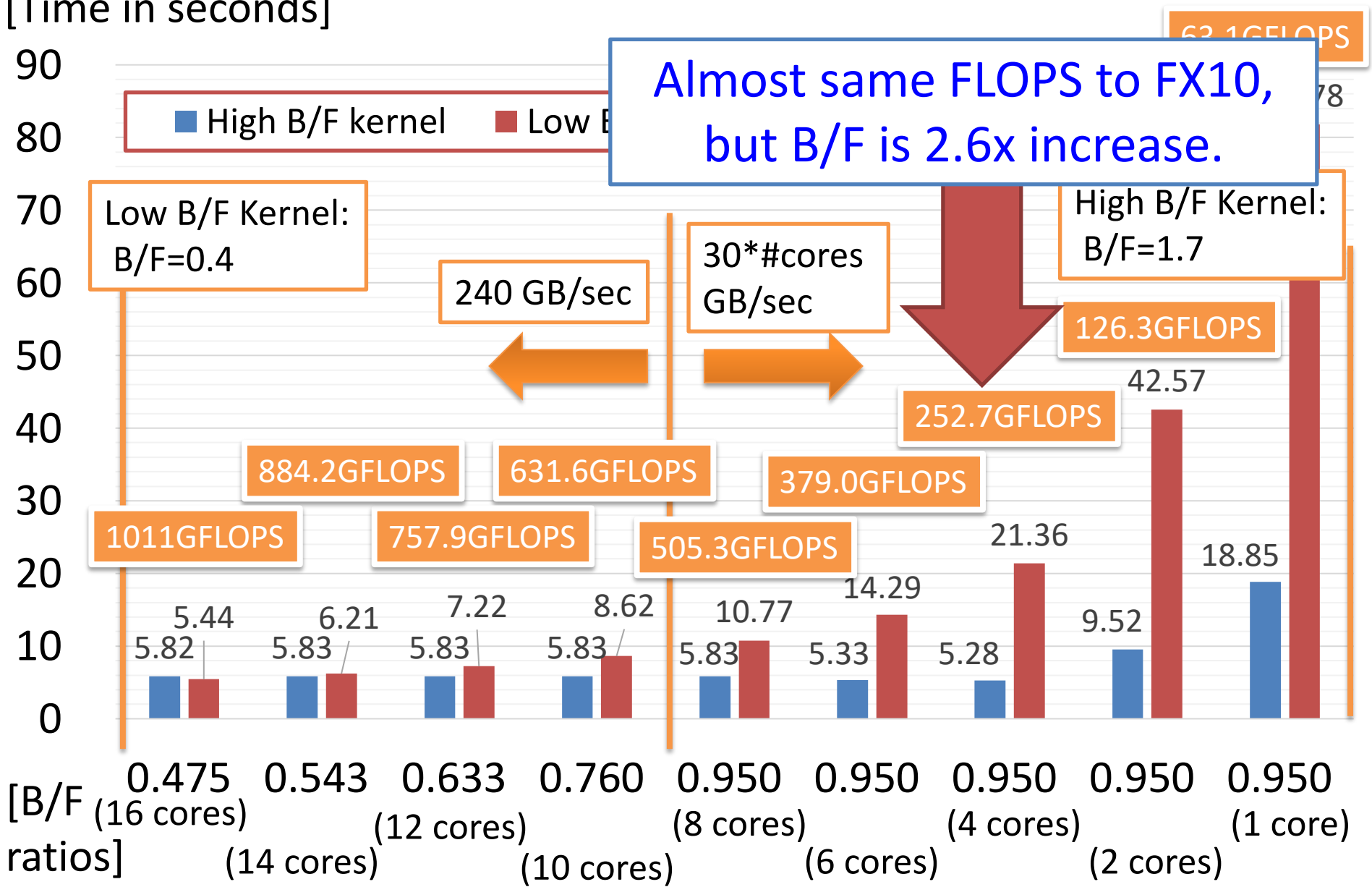
Changing B/F ratio by Hardware Viewpoint

- If we use a double precision computation, we have the following B/F:
  - 16 cores: B/F = 240 / 505.5 = 0.47
  - 14 cores: B/F = 240 / (505.5/16 *14) = 0.543
  - 12 cores: B/F = 240 / (505.5/16 *12) = 0.633
  - 10 cores: B/F = 240 / (505.5/16 *10) = 0.760
  - From 8 cores to 1 core:
    B/F = 240 / (505.5/16*8) = 0.950
- Hence we can test B/F from 0.47 to 0.950 by using the FX100.

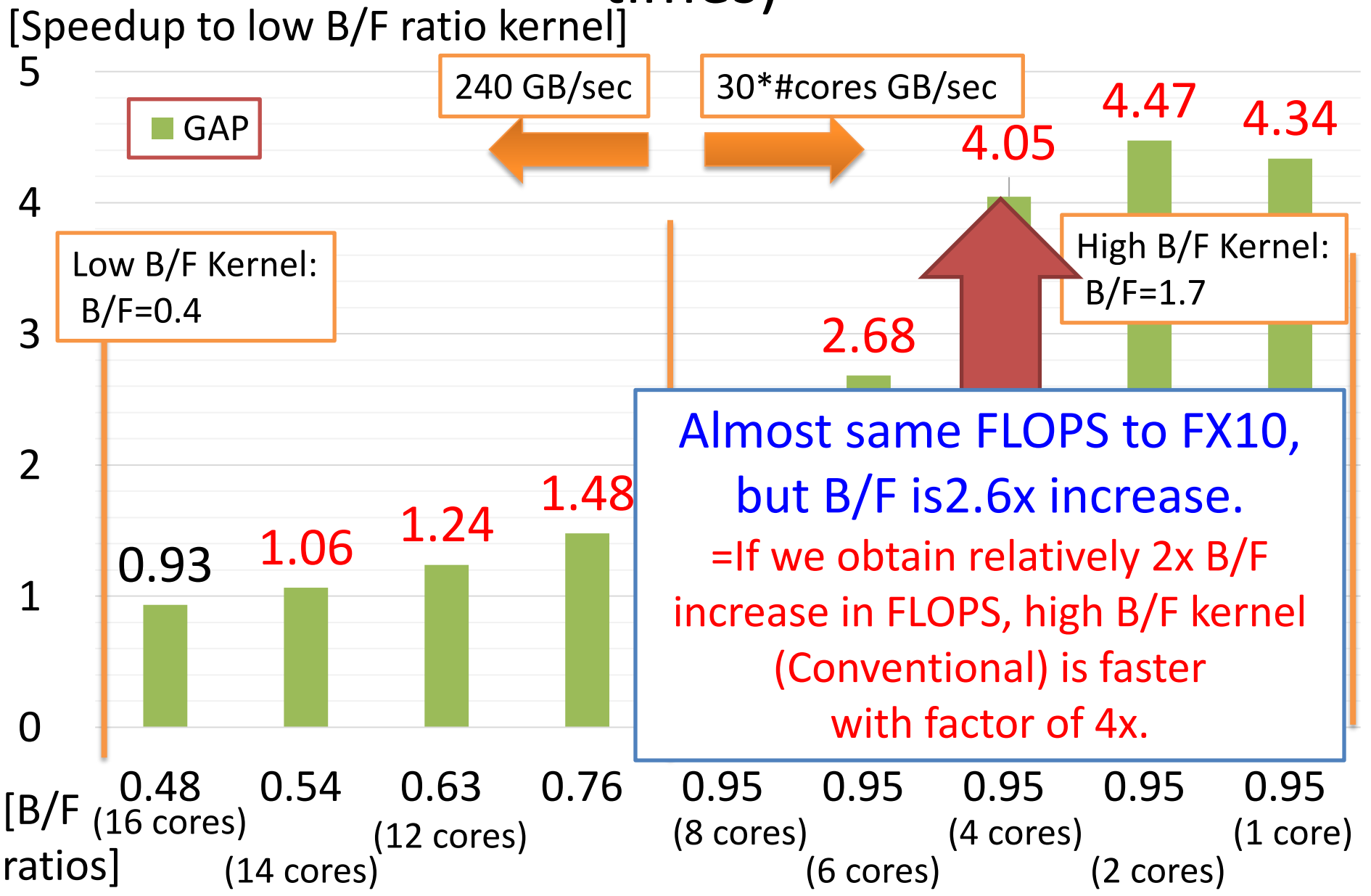
Experimental Condition

- We choose different computation kernels from seism_3D in *update_stress* kernels:
  - High B/F kernel
    - Original Kernel (for vector machine)
    - B/F = 1.7
  - Low B/F kernel
    - Variant2 (IF-free, but there is IF-statements inside loop for process of model boundary.)
    - B/F = 0.4
- Our estimation:
  - According to hardware B/F increases, the low B/F kernel is getting faster to the high B/F kernel.
  - -> If B/F ratio is increase in Post Moore's Era, algorithm trend changes, like using Low B/F kernel to High B/F Kernel.

Results in Time (update_stress, 100 times)



Results in Speedup (update_stress, 100 times)

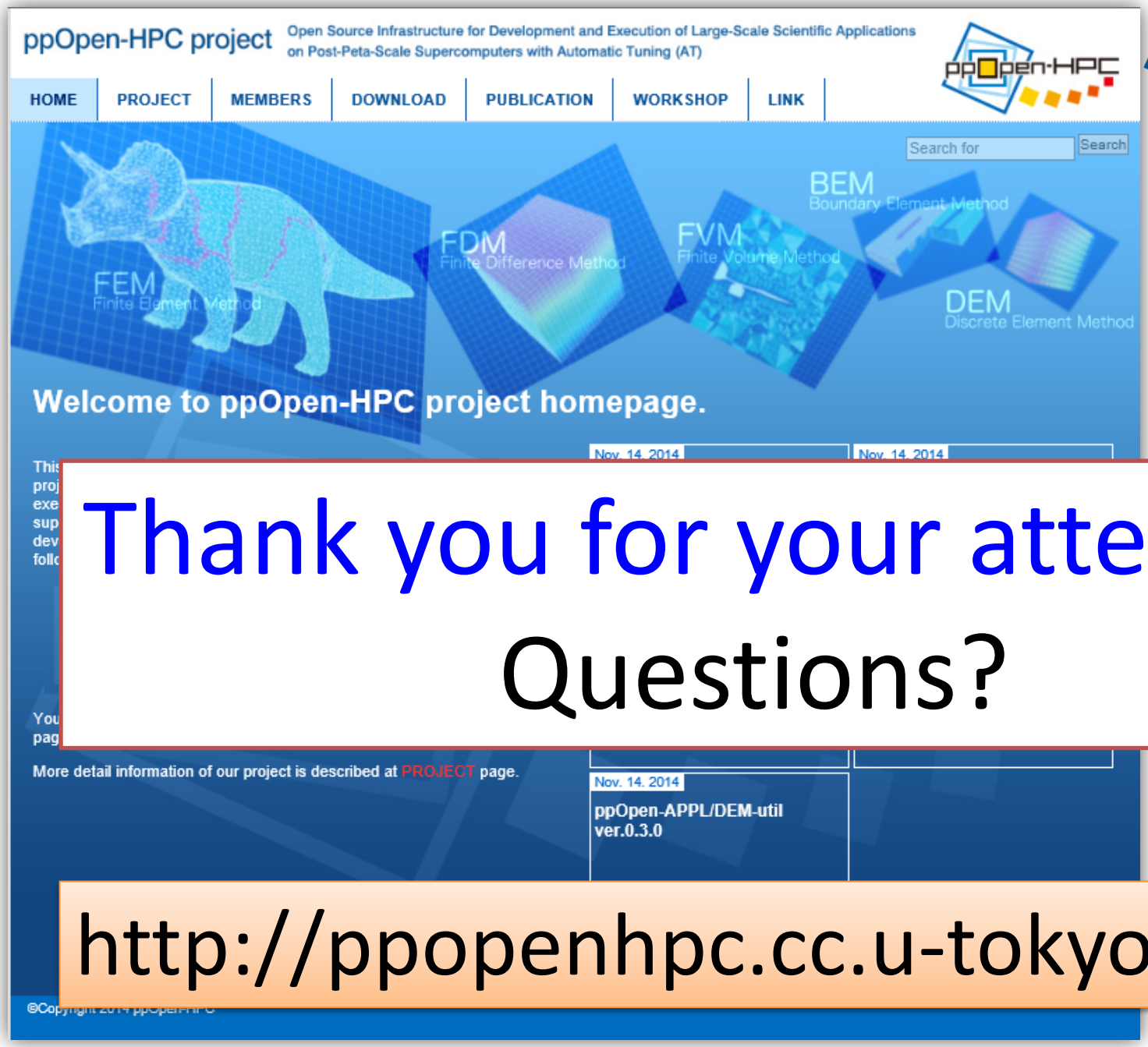


Outline

- Background
- An Auto-tuning (AT) Language: *ppOpen-AT* and Adapting AT to an FDM code
- An Evidence: Changes from FLOPS to BYTES
- Conclusion

Conclusion

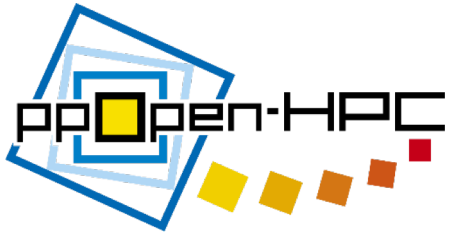
- In Post Moore's Era, we think that ability of BYTES is relatively growing with compared to ability of FLOPS.
- IF so, we need to change paradigm from FLOPS to BYTES for numerical algorithms, in particular, computations inside node.
- The evidence indicates that the best implementation changes according to increase of B/F ratios.



Thank you for your attention!

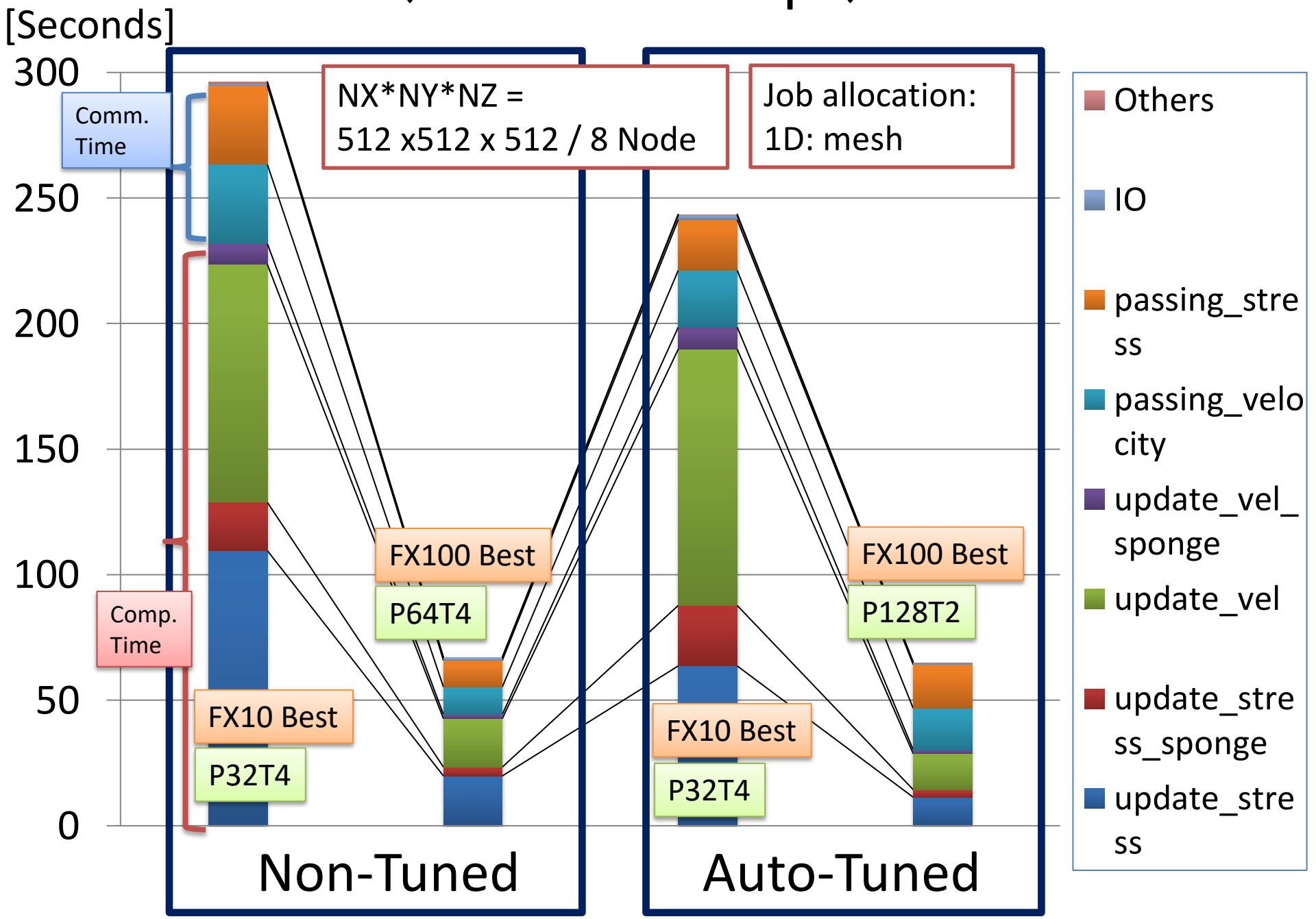
Questions?

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

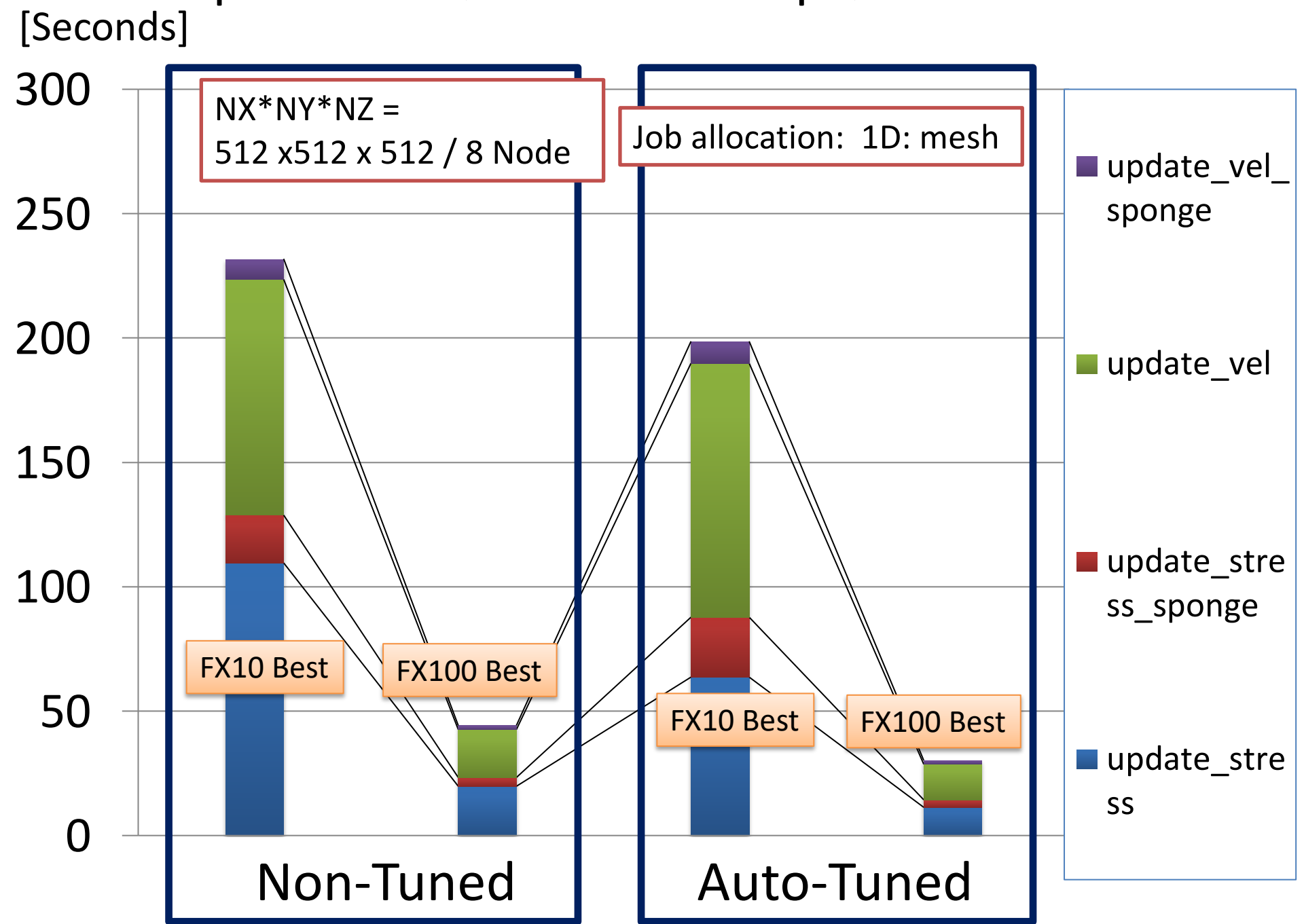

ppOpen-HPC

BACK UP

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



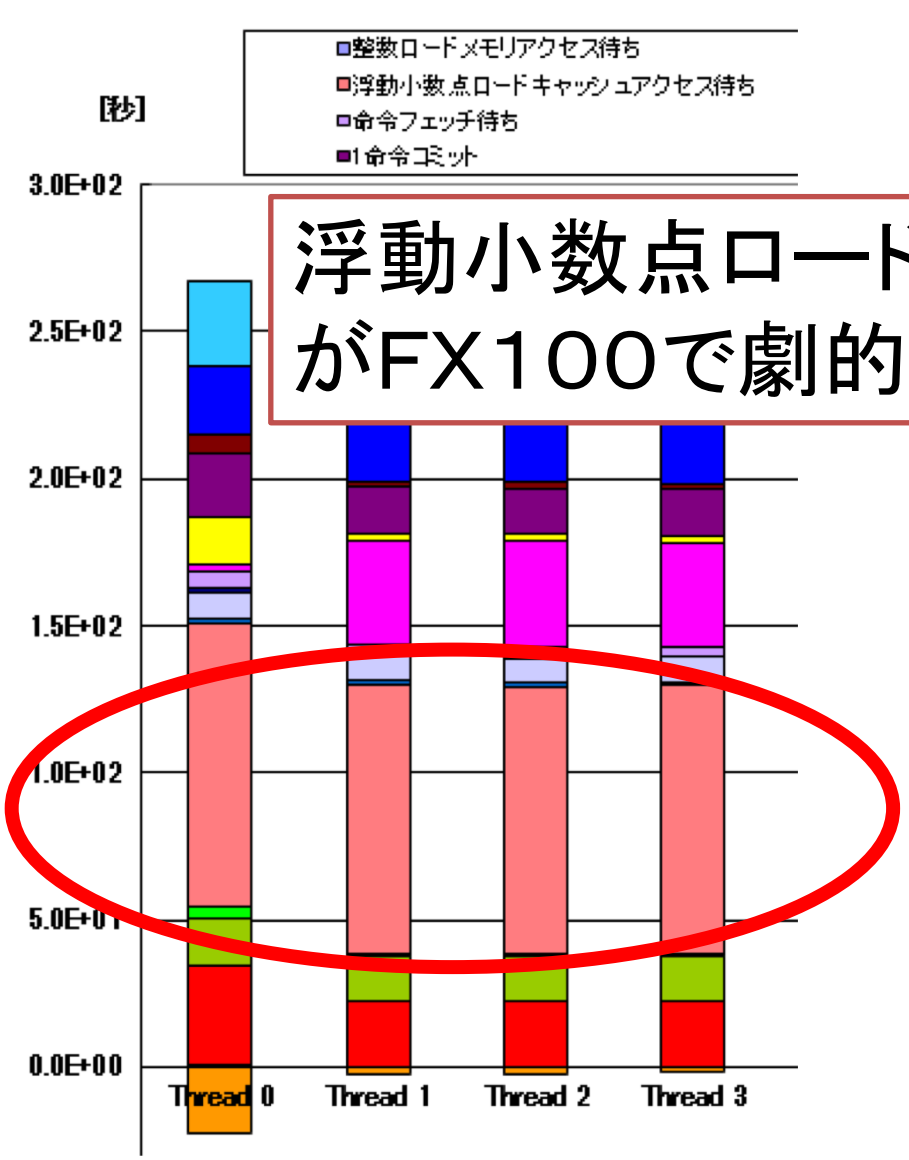
Comp. Kernels（2000 time steps）:FX10 vs. FX100



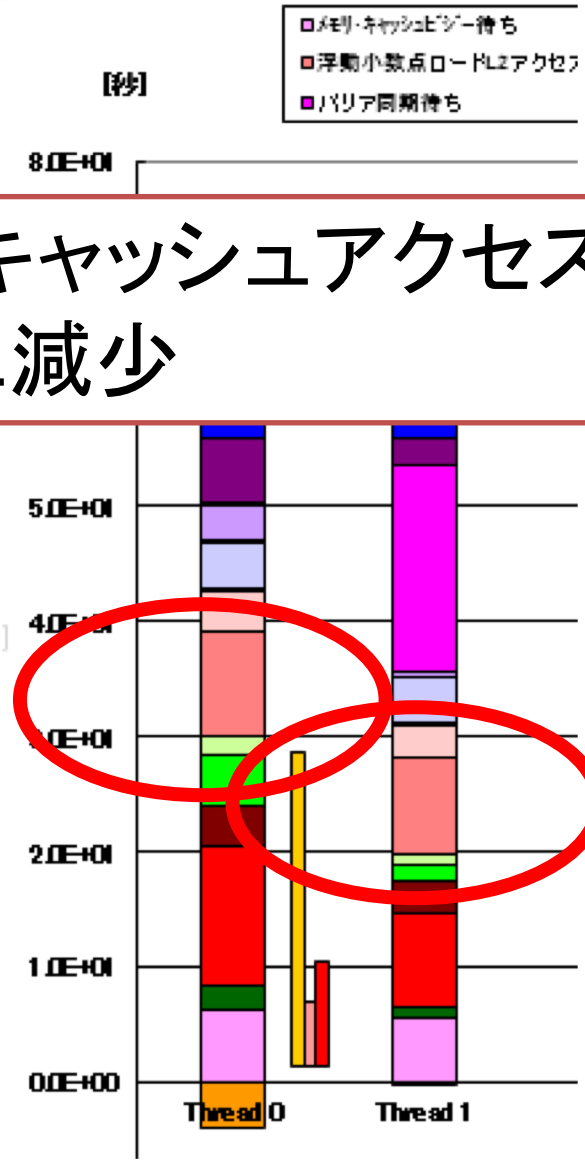
RESULTS OF PERFORMANCE PROFILING

Profile Results (Tuned)

浮動小数点ロードキャッシュアクセス待ちがFX100で劇的に減少



（a）FX10 (P32T4)



（b）FX100 (P128T2)

Profile Results (Tuned)

(a) FX10 (P32T4)

時間グラフ内訳割合 (%)	整数ロードメモリアクセス待ち	浮動小数点ロードメモリアクセス待ち	ストア待ち	整数ロードキャッシュアクセス待ち	浮動小数点ロードキャッシュアクセス待ち	整数演算待ち	浮動小数点演算待ち	分岐命令待ち	命令フェッチ待ち	バリア同期待ち	uOPコミット	その他の待ち	1命令コミット	整数レジスタ書き込み制約	2/3命令コミット	4命令コミット	合計
Thread 0	0.32%	13.78%	6.71%	1.5%	39.31%	0.80%	3.52%	0.60%	2.43%	0.77%	6.71%	−9.30%	8.87%	2.48%	9.78%	11.70%	100.00%
Thread 1	0.12%	8.95%	6.28%	0.8%	37.55%	0.52%	3.57%	0.10%	1.29%	14.50%	0.98%	−0.99%	6.45%	0.88%	8.43%	11.10%	100.00%
Thread 2	0.12%	8.96%	6.27%	0.8%	37.18%	0.51%	3.56%	0.10%	1.29%	14.87%	0.98%	−0.85%	6.39%	0.87%	8.37%	11.09%	100.00%
Thread 3	0.14%	9.09%	6.26%	0.7%	37.27%	0.51%	3.57%	0.10%	1.29%	14.46%	0.98%	−0.66%	6.39%	0.87%	8.37%	11.09%	100.00%

(b) FX100 (P128T2)

時間グラフ内訳割合 (%)	メモリ・キャッシュビジー待ち	整数ロードメモリアクセス待ち	浮動小数点ロードメモリアクセス待ち	ストア待ち	整数ロードL2アクセス待ち	整数ロードL1Dアクセス待ち	浮動小数点ロードL2アクセス待ち	浮動小数点ロードL1Dアクセス待ち	整数演算待ち	浮動小数点演算待ち	分岐命令待ち	命令フェッチ待ち	バリア同期待ち	その他の待ち	1命令コミット	2/3命令コミット	4命令コミット	合計
Thread 0	10.01%	3.26%	19.42%	5.79%	7.26%	1.41%	14.67%	5.36%	0.60%	6.21%	0.49%	4.97%	0.34%	−6.52%	8.86%	8.03%	8.86%	100.00%
Thread 1	8.68%	1.40%	12.99%	4.15%	2.28%	1.46%	13.14%	4.47%	0.36%	6.02%	0.07%	0.92%	27.93%	−0.19%	3.75%	4.26%	8.32%	100.00%

FX10：浮動小数点ロードキャッシュアクセス待ち：37～39%

FX100：浮動小数点ロードL2キャッシュアクセス待ち：13～14%

FX10：L2ミス数： 4.8 ~ 5.0E+9

FX100：L2ミス数：1.5 ~ 1.9E+9

最大で3.3倍ほどミス数が削減

Profile Results (Tuned)

浮動小数点ロードの

FX10：キャッシュ待ち時間：
90.9 ~ 96.1 [s]

FX10：メモリアクセス待ち時間：
21.9 ~ 33.6 [s]

FX100：L2アクセス待ち時間：
8.3 ~ 9.1 [s]

FX100：メモリアクセス待ち時間：
8.2 ~ 12.1 [s]

最大で
11.5倍ほど高速化

最大で
4.0倍ほど高速化

FX100ではL2アクセス
待ち時間が効率化

ノード当たりのL2キャッシュ
容量の増加が高速化に貢献

PARAMETER DISTRIBUTION

Parameter Distribution (update_stress_select)