ppOpen-HPC

Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications with Automatic Tuning (AT)

自動チューニング機構を有するアプリケーション開発・実行環境

Kengo Nakajima, Takahiro Katagiri

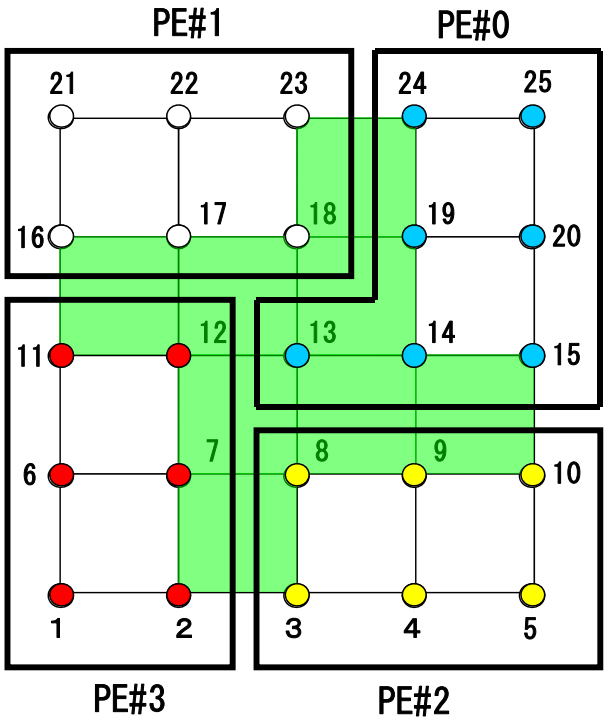
Information Technology Center

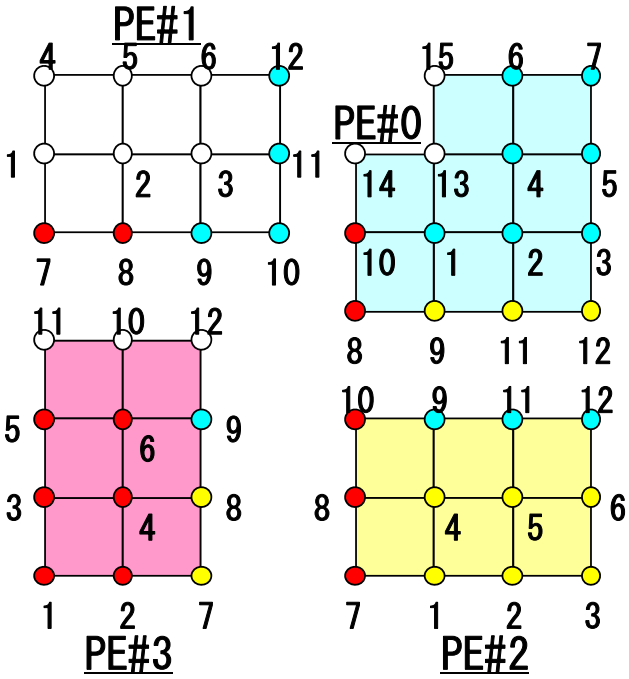
The University of Tokyo

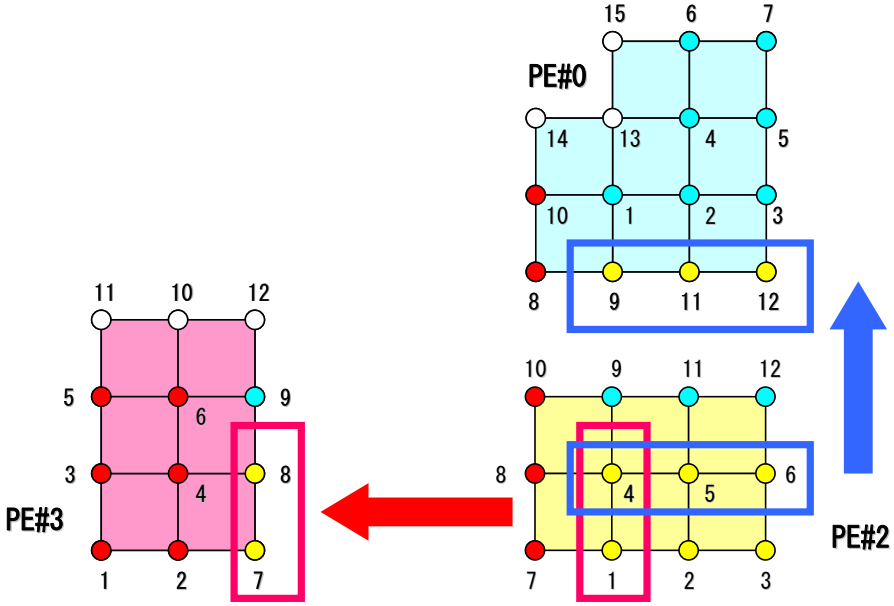
Lessons learned in the 20th Century

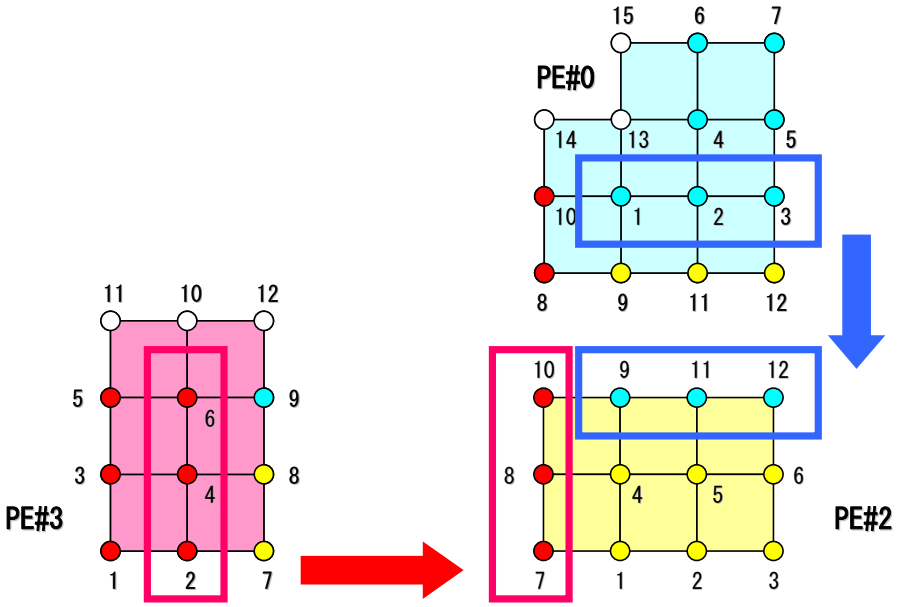
- Methods for scientific computing (e.g. FEM, FDM, BEM etc.) consists of typical data structures, and typical procedures.
  - Optimization of each procedure is possible and effective.

- Well-defined data structure can “hide” communication processes with MPI from code developer.
  - Code developers do not have to care about communications
  - Halo for parallel FEM


PE#1
PE#0
21 22 23 24 25
16 17 18 19 20
11 12 13 14 15
6 7 8 9 10
1 2 3 4 5
PE#3
PE#2


PE#1
4 5 6 12
1 2 3 11
7 8 9 10
PE#0
15 6 7
14 13 4 5
10 1 2 3
8 9 11 12
11 10 12
5 6 9
3 4 8
1 2 7
PE#3
10 9 11 12
8 4 5 6
7 1 2 3
PE#2

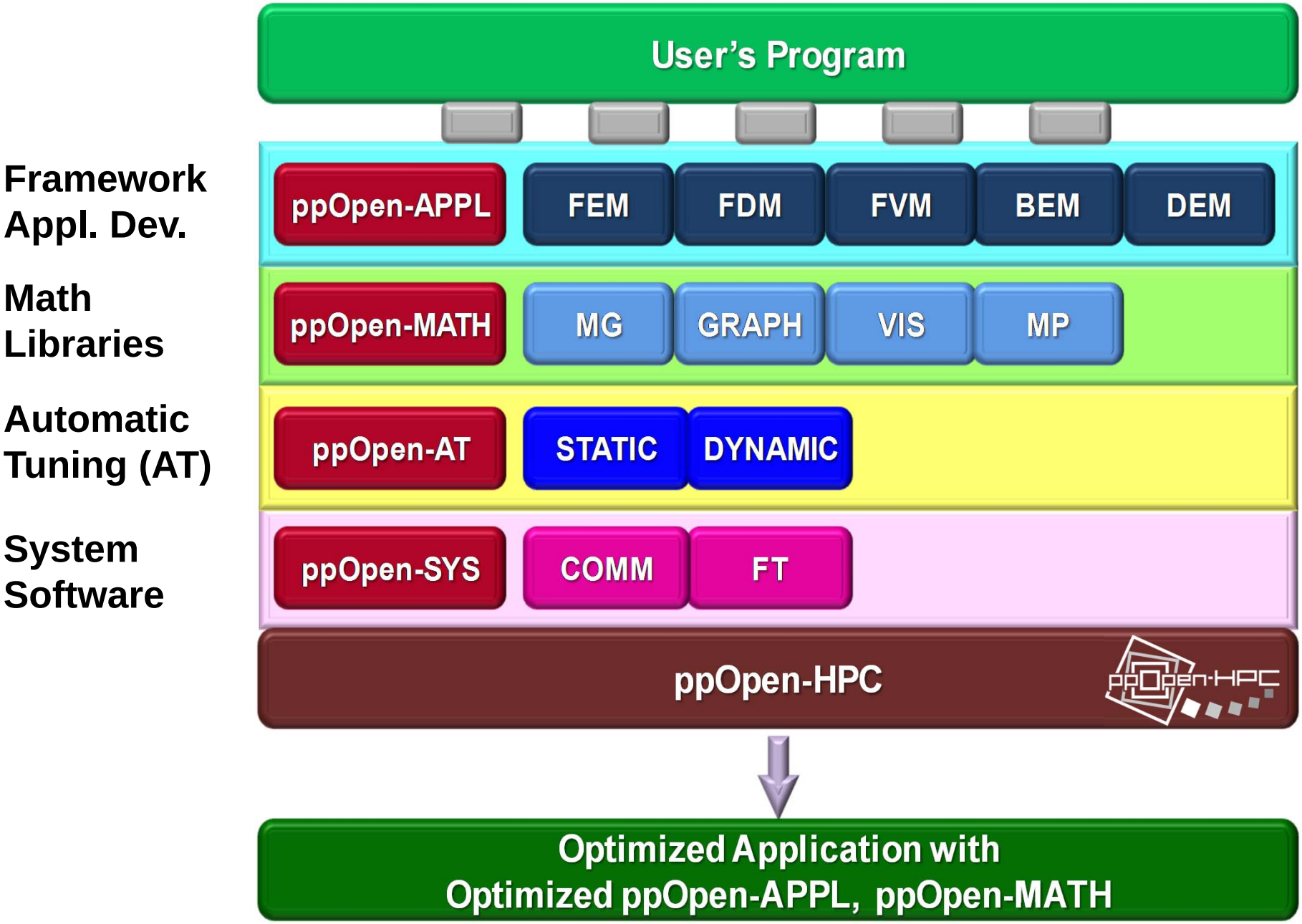

15 6 7
PE#0
14 13 4 5
10 1 2 3
8 9 11 12
10 9 11 12
8 4 5 6
7 1 2 3
PE#2
11 10 12
5 6 9
3 4 8
PE#3
1 2 7


15 6 7
PE#0
14 13 4 5
10 1 2 3
8 9 11 12
11 10 12
5 6 9
3 4 8
PE#3
1 2 7
10 9 11 12
8 4 5 6
7 1 2 3
PE#2

ppOpen-HPC: Overview

- Open Source Infrastructure for development and execution of large-scale scientific applications on post-peta-scale supercomputers with automatic tuning (AT)
  - “pp” : post-peta-scale
- Five-year project (FY.2011-2015) (since April 2011)
  - P.I.: Kengo Nakajima (ITC, The University of Tokyo)
  - Part of “Development of System Software Technologies for Post-Peta Scale High Performance Computing” funded by JST/CREST (Supervisor: Prof. Akinori Yonezawa, Co-Director, RIKEN AICS)
- Team with 7 institutes, >30 people (5 PDs) from various fields: Co-Desigin
  - ITC/U.Tokyo, AORI/U.Tokyo, ERI/U.Tokyo, FS/U.Tokyo
  - Hokkaido U., Kyoto U., JAMSTEC

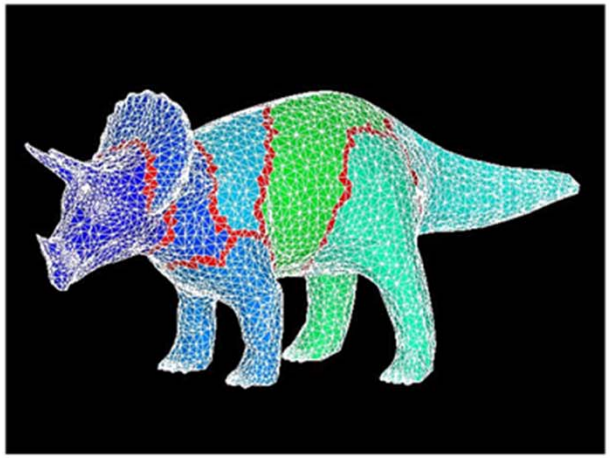
- Group Leaders
  - Masaki Satoh (AORI/U.Tokyo)
  - Takashi Furumura (ERI/U.Tokyo)
  - Hiroshi Okuda (GSFS/U.Tokyo)
  - Takeshi Iwashita (Kyoto U., ITC/Hokkaido U.)
  - Hide Sakaguchi (IFREE/JAMSTEC)
- Main Members
  - Takahiro Katagiri (ITC/U.Tokyo)
  - Masaharu Matsumoto (ITC/U.Tokyo)
  - Hideyuki Jitsumoto (ITC/U.Tokyo)
  - Satoshi Ohshima (ITC/U.Tokyo)
  - Hiroyasu Hasumi (AORI/U.Tokyo)
  - Takashi Arakawa (RIST)
  - Futoshi Mori (ERI/U.Tokyo)
  - Takeshi Kitayama (GSFS/U.Tokyo)
  - Akihiro Ida (ACCMS/Kyoto U.)
  - Miki Yamamoto (IFREE/JAMSTEC)
  - Daisuke Nishiura (IFREE/JAMSTEC)


User's Program
Framework
Appl. Dev.
ppOpen-APPL
FEM
FDM
FVM
BEM
DEM
Math
Libraries
ppOpen-MATH
MG
GRAPH
VIS
MP
Automatic
Tuning (AT)
ppOpen-AT
STATIC
DYNAMIC
System
Software
ppOpen-SYS
COMM
FT
ppOpen-HPC
Optimized Application with
Optimized ppOpen-APPL, ppOpen-MATH

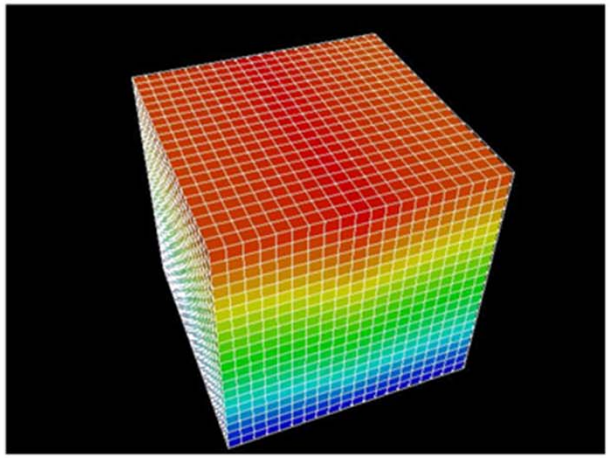
ppOpen-HPC: ppOpen-APPL

- ppOpen-HPC consists of various types of *optimized* libraries, which covers various types of procedures for scientific computations.
  - ppOpen-APPL/FEM, FDM, FVM, BEM, DEM
  - Linear Solvers, Mat. Assemble, AMR., Visualization etc.
  - written in Fortran 2003 (C interface is available soon)
- Source code developed on a PC with a single processor is linked with these libraries, and generated parallel code is optimized for post-peta scale system.
- Users don't have to worry about optimization tuning, parallelization etc.
  - Part of MPI, OpenMP, (OpenACC)

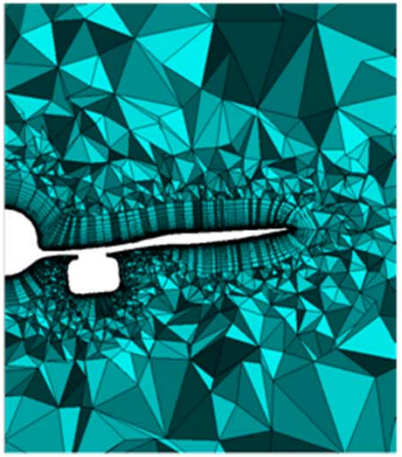
ppOpen-HPC covers …



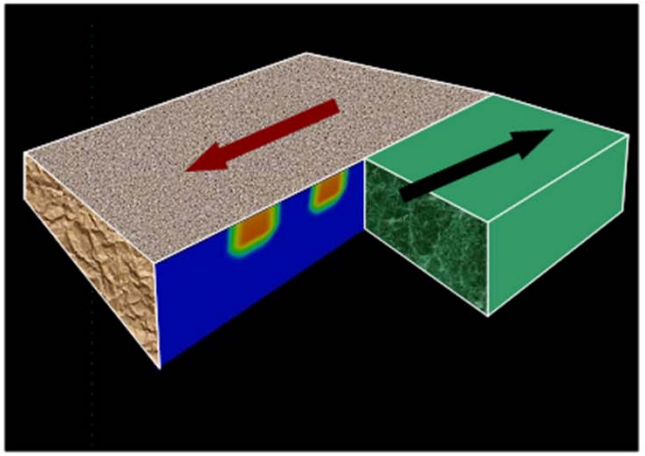
FEM
Finite Element Method



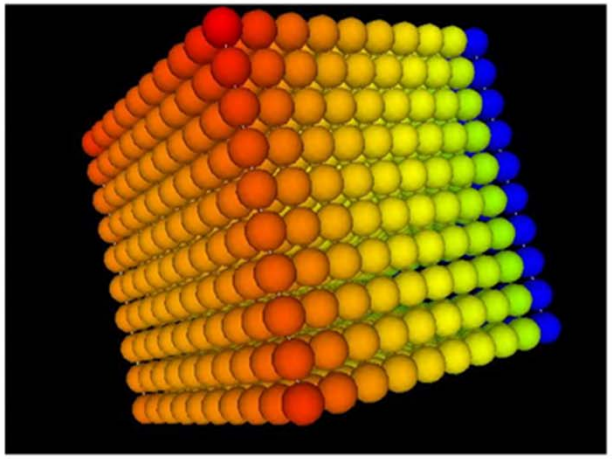
FDM
Finite Difference Method



FVM
Finite Volume Method



BEM
Boundary Element Method



DEM
Discrete Element Method

FEM Code on ppOpen-HPC

Optimization/parallelization could be hidden from application developers

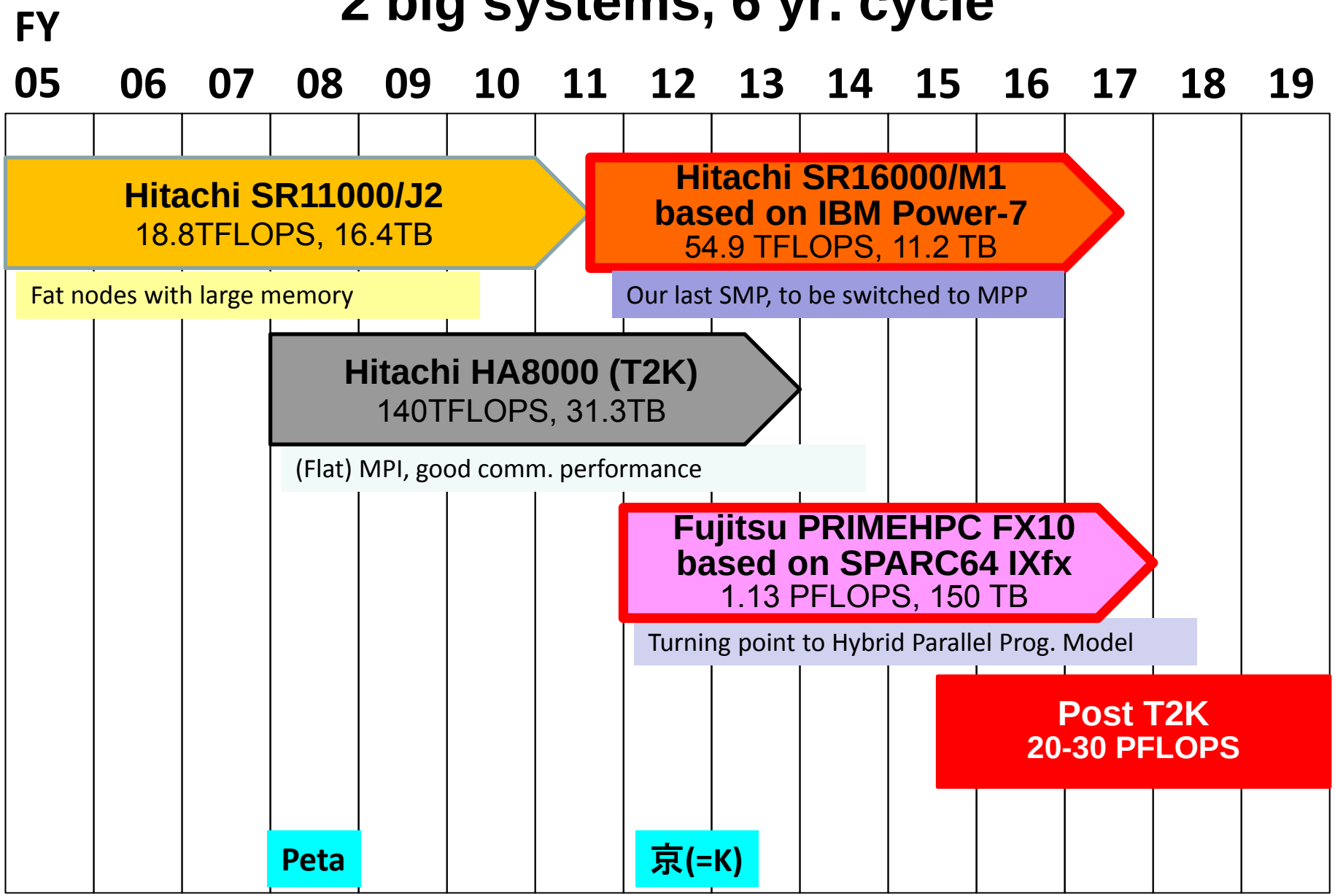
```
Program My_pFEM
use ppOpenFEM_util
use ppOpenFEM_solver

call ppOpenFEM_init
call ppOpenFEM_cntl
call ppOpenFEM_mesh
call ppOpenFEM_mat_init

do
  call Users_FEM_mat_ass
  call Users_FEM_mat_bc
  call ppOpenFEM_solve
  call ppOPenFEM_vis
  Time= Time + DT
enddo

call ppOpenFEM_finalize
stop
end
```

Target Applications

- Our goal is not development of applications, but we need some target appl. for evaluation of ppOpen-HPC.
- ppOpen-APPL/FEM
  - Incompressible Navier-Stokes
  - Heat Transfer, Solid Mechanics (Static, Dynamic)
- ppOpen-APPL/FDM
  - Incompressible Navier-Stokes
  - Transient Heat Transfer, Solid Mechanics (Dynamic)
- ppOpen-APPL/FVM
  - Compressible Navier-Stokes, Heat Transfer
- ppOpen-APPL/BEM
  - Electromagnetics, Solid Mechanics (Quasi Static) (Earthquake Generation Cycle)
- ppOpen-APPL/DEM
  - Incompressible Navier-Stokes, Solid Mechanics (Dynamic)

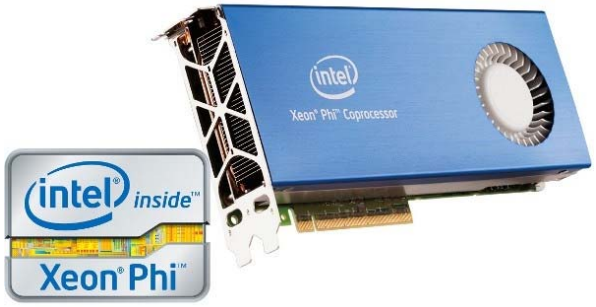
Supercomputers in U.Tokyo

2 big systems, 6 yr. cycle

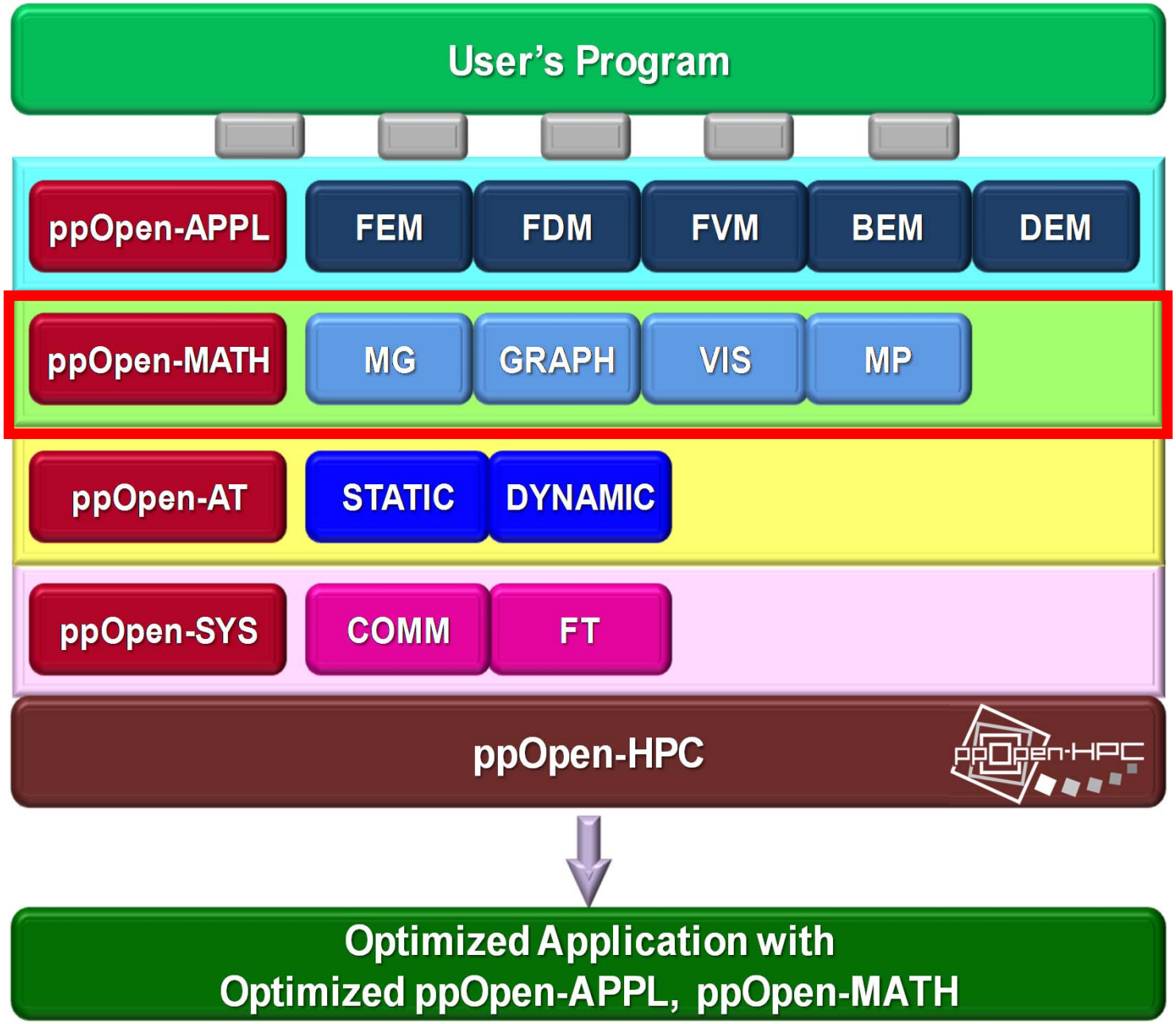


Target: Post T2K System

- Target system is Post T2K system
  - > 30 PFLOPS, FY.2015-2016
    - ✓ JCAHPC (Joint Center for Advanced High Performance Computing, 最先端共同HPC基盤施設): U. Tsukuba & U. Tokyo
    - ✓ http://jcahpc.jp/
  - Many-core based (e.g. Intel MIC/Xeon Phi)
    - ✓ MPI + OpenMP + X
  - ppOpen-HPC helps smooth transition of users (> 2,000) to new system
- K/FX10, Cray, Xeon clusters are also in scope



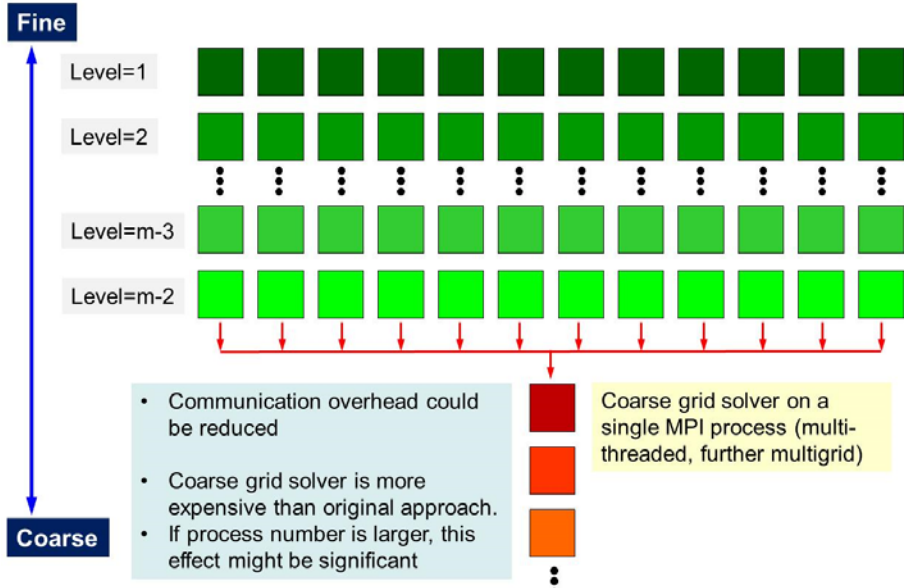
JCAHPC


User's Program
ppOpen-APPL
FEM
FDM
FVM
BEM
DEM
ppOpen-MATH
MG
GRAPH
VIS
MP
ppOpen-AT
STATIC
DYNAMIC
ppOpen-SYS
COMM
FT
ppOpen-HPC
Optimized Application with
Optimized ppOpen-APPL, ppOpen-MATH

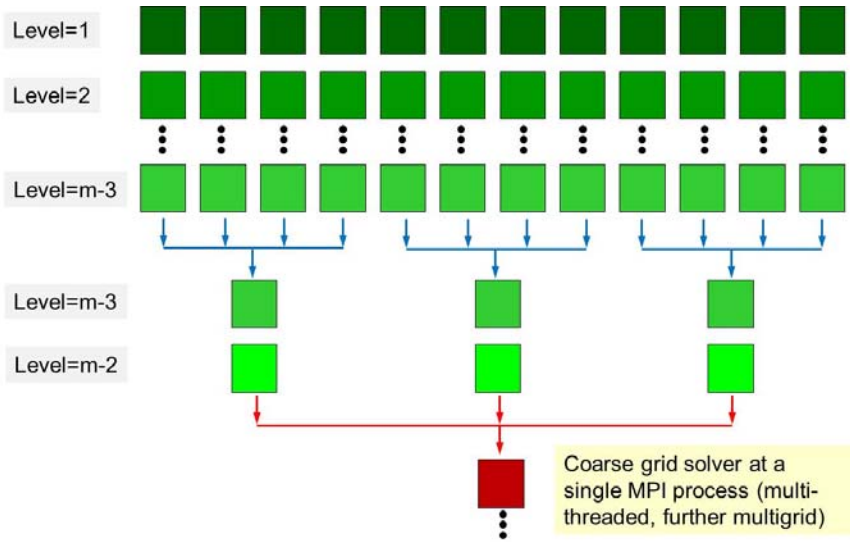
ppOpen-MATH

- A set of common numerical libraries
  - Multigrid solvers (ppOpen-MATH/MG) (Later)
  - Parallel graph libraries (ppOpen-MATH/GRAPH)
    - Multithreaded RCM for reordering (under development)
  - Parallel visualization (ppOpen-MATH/VIS)
  - Library for coupled multi-physics simulations (loose-coupling) (ppOpen-MATH/MP)
    - Originally developed as a coupler for NICAM (atmosphere, unstructured), and COCO (ocean, structured) in global climate simulations using K computer
      - Both codes are major codes on the K computer.
        - » Prof. Masaki Satoh (AORI/U.Tokyo): NICAM
        - » Prof. Hiroyasu Hasumi (AORI/U.Tokyo): COCO
    - Developed coupler is extended to more general use.
      - Coupled seismic simulations

ppOpen-MATH/MG (with CR/SR)



CGA (Coarse Grid Aggregation)

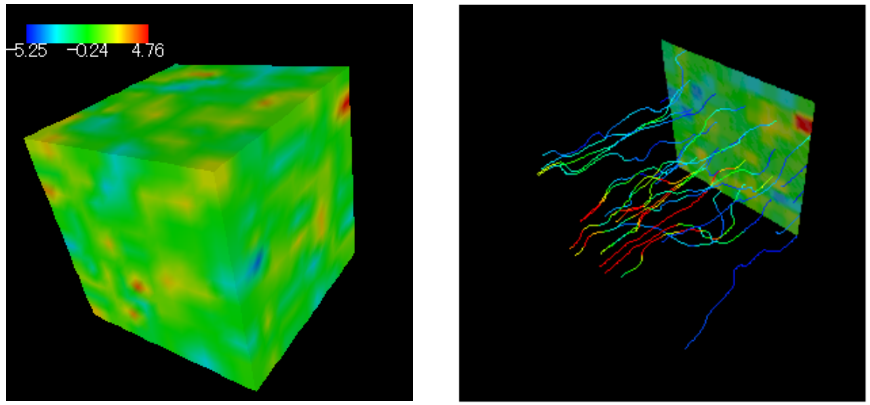


*h*CGA (Hierarchical CGA)

MGCG Solver with CGA/*h*CGA on 4,096 nodes (65,536 cores) of Fujitsu FX10 (Oakleaf-FX)

3D Groundwater Flow through Heterogeneous Porous Media

Nakajima, K. "Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method" (Best Paper Award, ICPADS 2014)

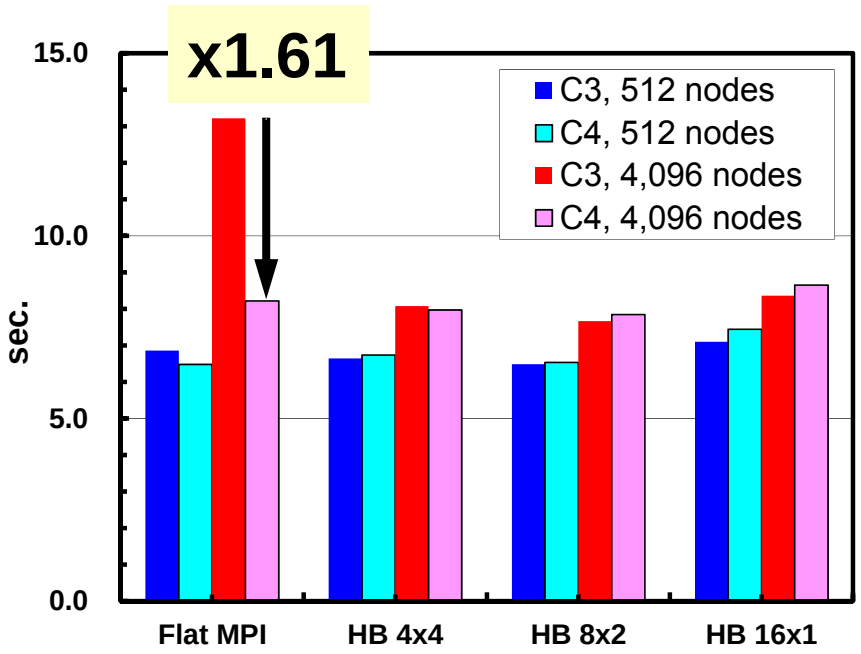


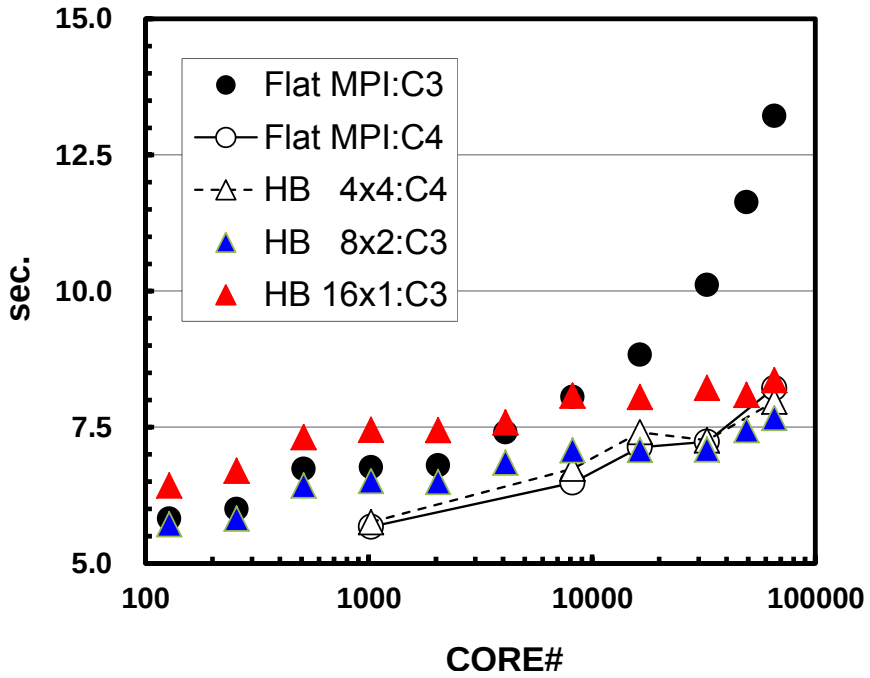
Weak Scaling up to 4,096 nodes

max. 17,179,869,184 meshes (64^3 meshes/core)

DOWN is GOOD

	Matrix	Coarse Grid
C0	CRS	Single Core
C1	ELL (org)	Single Core
C2	ELL (org)	CGA
C3	**ELL (sliced)**	**CGA**
C4	**ELL (sliced)**	***h*CGA**





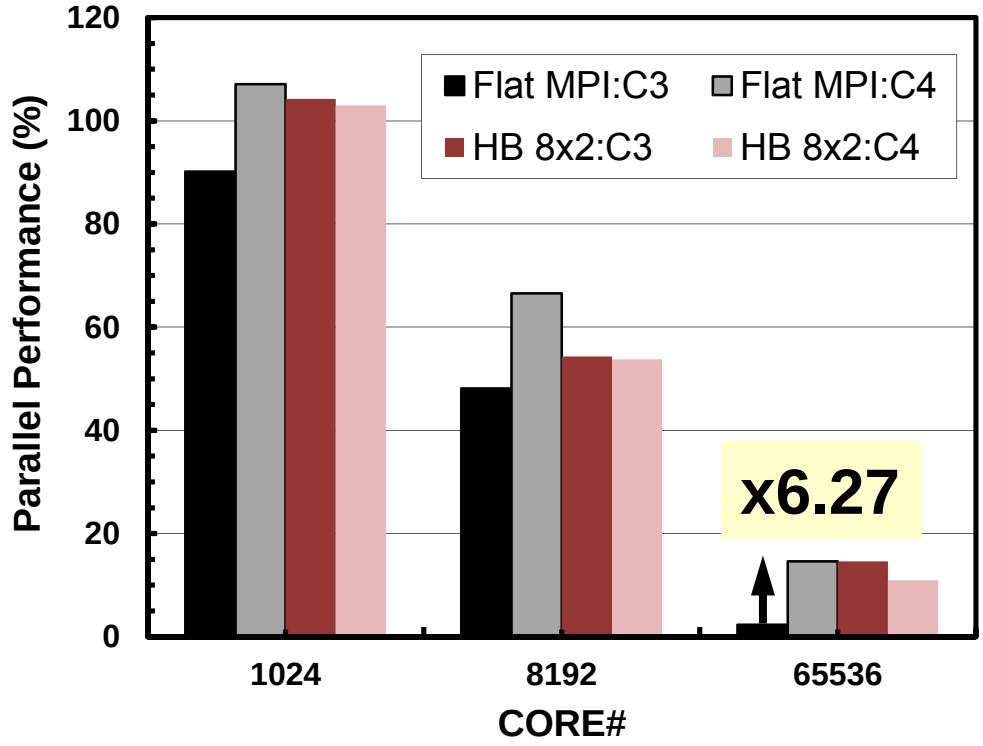
Strong Scaling up to 4,096 nodes

268,435,456 meshes, 16^3 meshes/core at 4,096 nodes

UP is GOOD

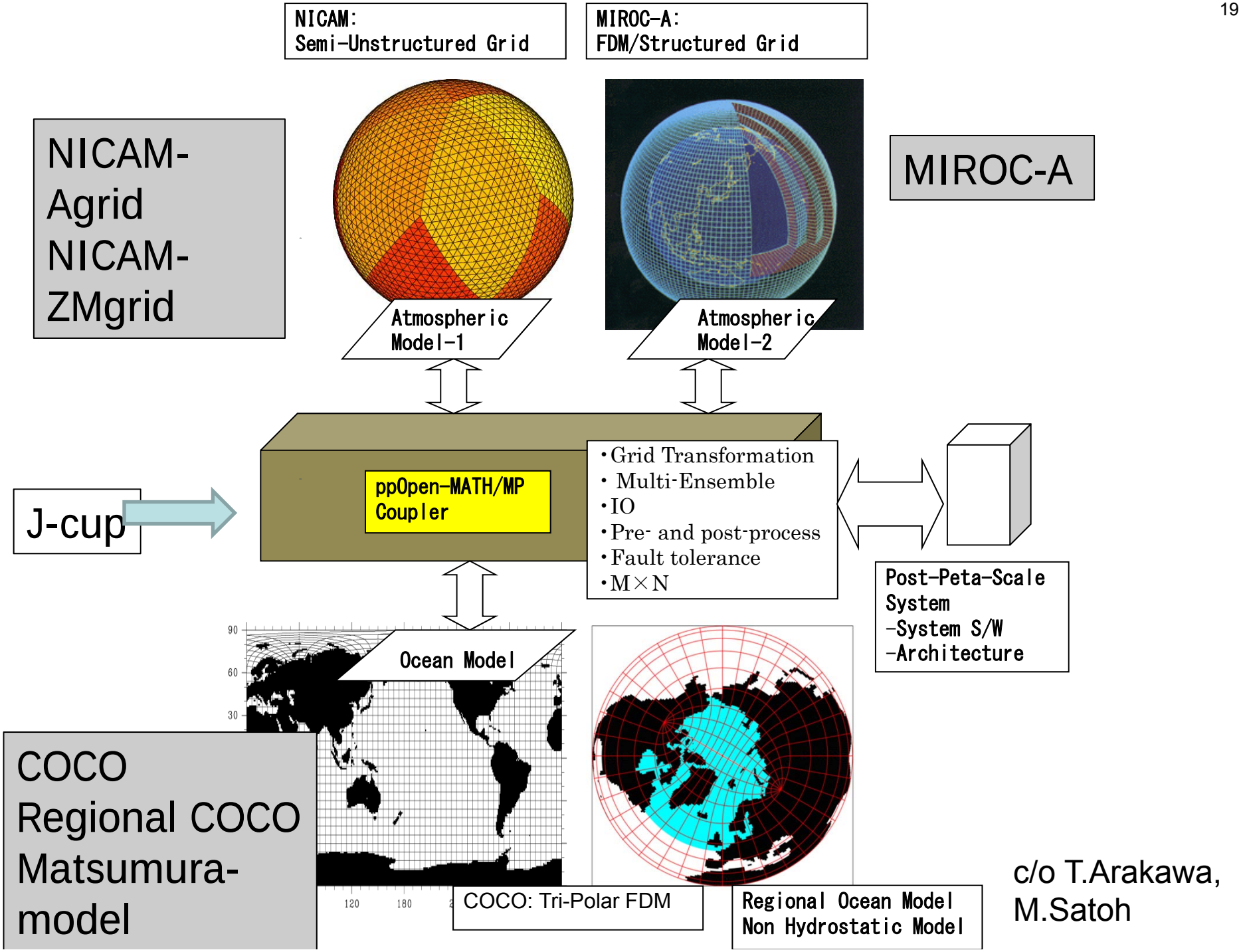
Flat MPI/ELL (C3),
8 nodes (128 cores) : 100%

	Matrix	Coarse Grid
C0	CRS	Single Core
C1	ELL (org)	Single Core
C2	ELL (org)	CGA
C3	**ELL (sliced)**	**CGA**
C4	**ELL (sliced)**	***h*CGA**



ppOpen-MATH

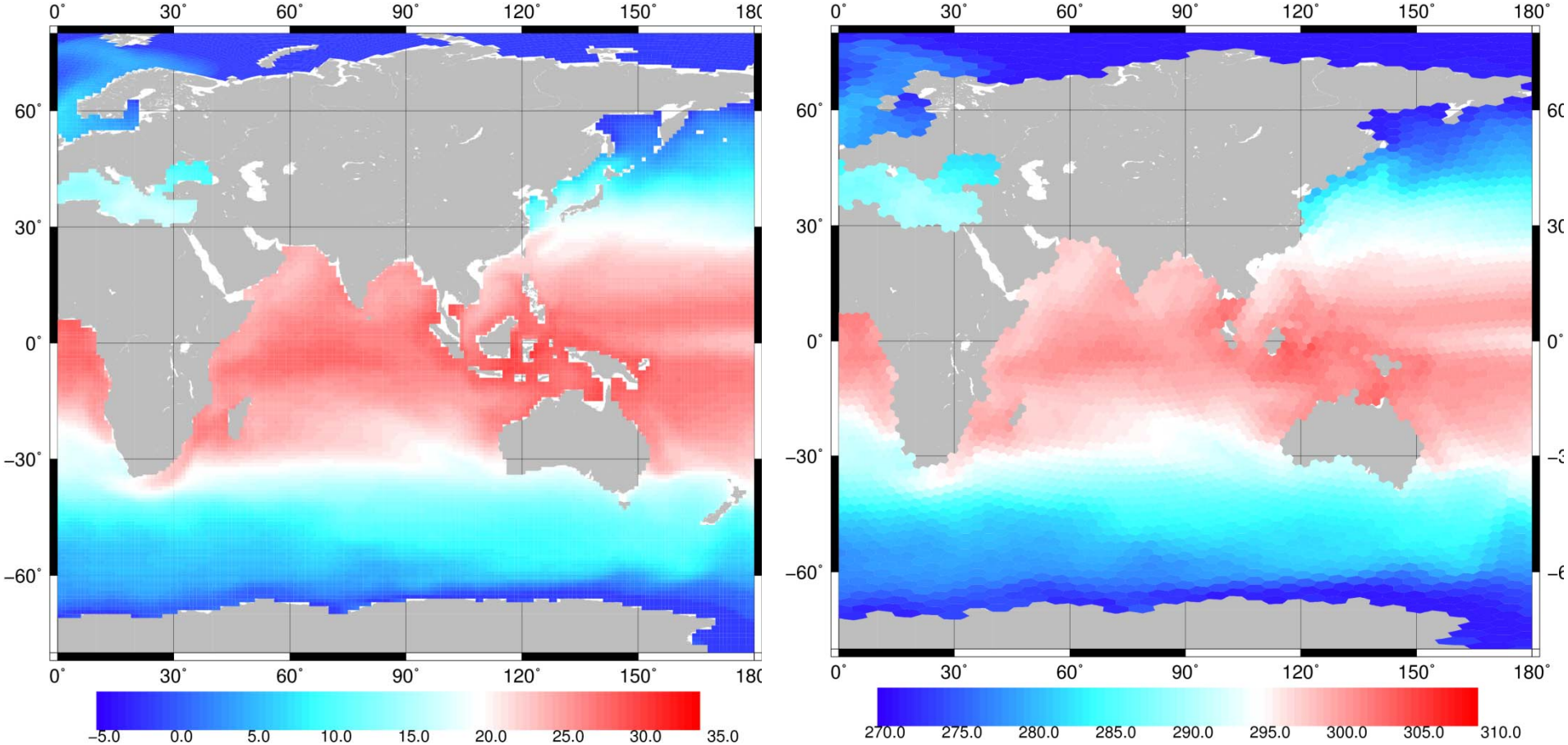
- A set of common numerical libraries
  - Multigrid solvers (ppOpen-MATH/MG)
  - Parallel graph libraries (ppOpen-MATH/GRAPH)
    - Multithreaded RCM for reordering (under development)
  - Parallel visualization (ppOpen-MATH/VIS)
  - Library for coupled multi-physics simulations (loose-coupling) (ppOpen-MATH/MP)
    - Originally developed as a coupler for NICAM (atmosphere, unstructured), and COCO (ocean, structured) in global climate simulations using K computer
      - Both codes are major codes on the K computer.
        - » Prof. Masaki Satoh (AORI/U.Tokyo): NICAM
        - » Prof. Hiroyasu Hasumi (AORI/U.Tokyo): COCO
    - Developed coupler is extended to more general use.
      - Coupled seismic simulations



c/o T.Arakawa, M.Satoh

c/o T.Arakawa, M.Satoh

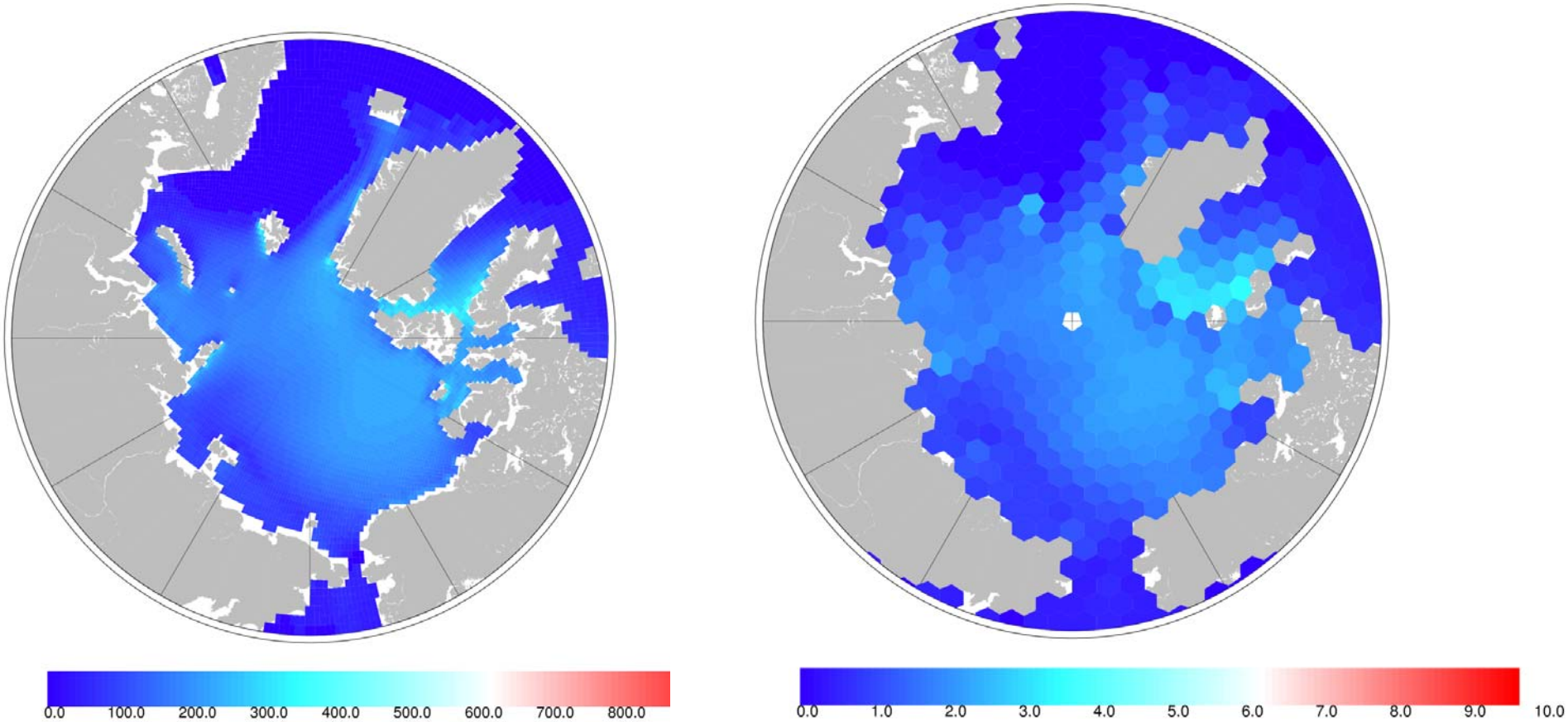
Sea surface temperature (OSST)



left : COCO (Ocean: Structured), right : NICAM (Atmospheric: Semi-Unst.)

c/o T.Arakawa, M.Satoh

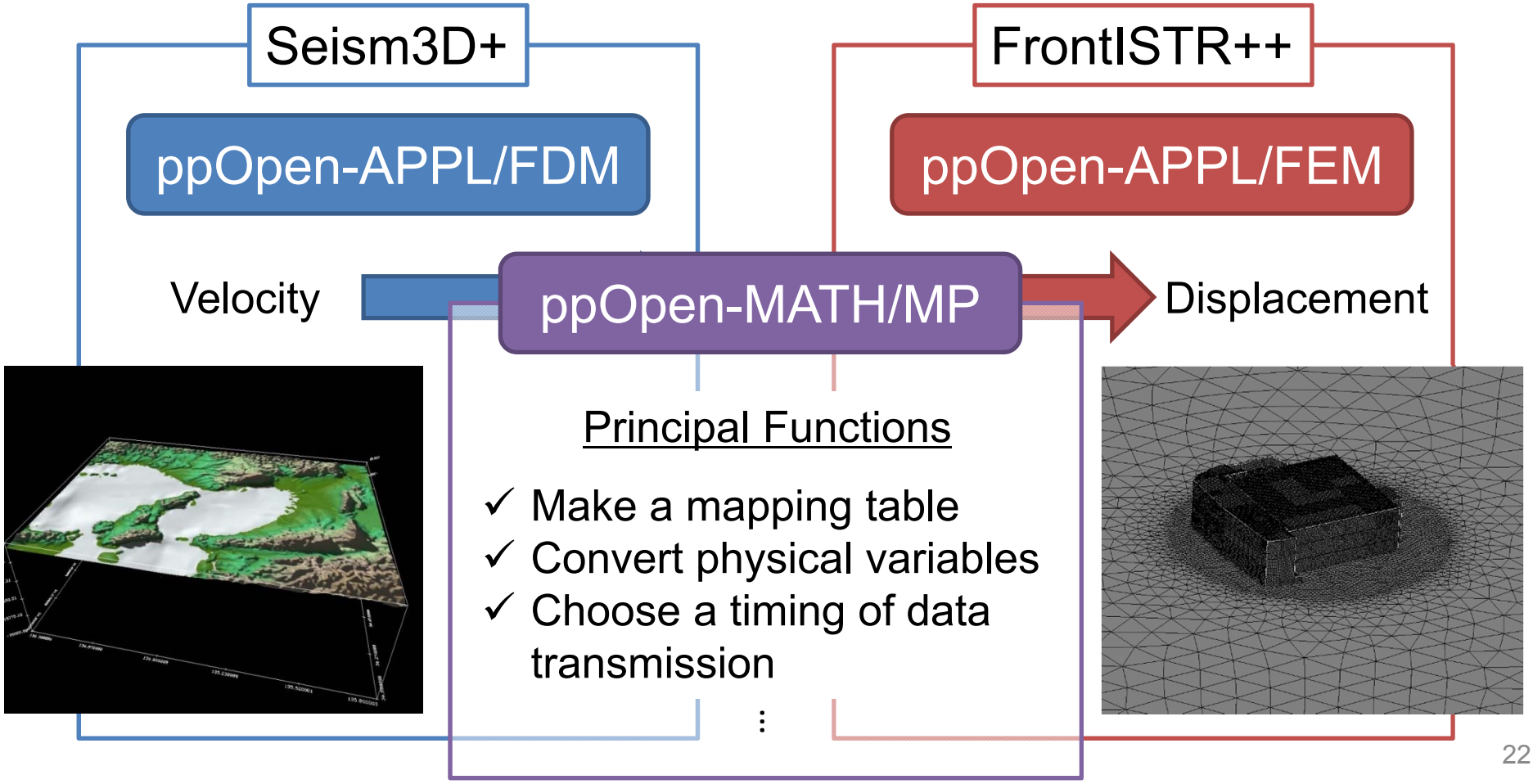
Thickness of Sea Ice (OHI)



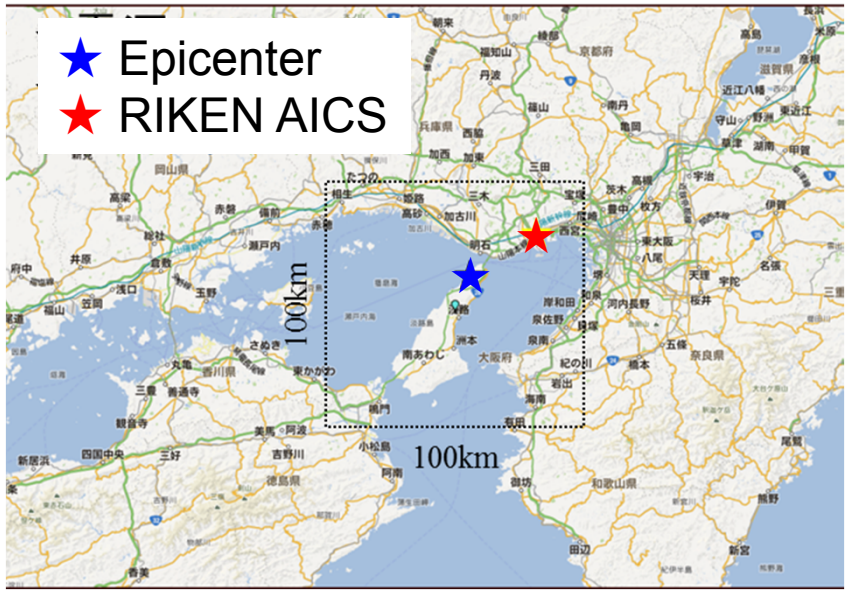
left : COCO (Ocean: Structured), right : NICAM (Atmospheric: Semi-Unst.)

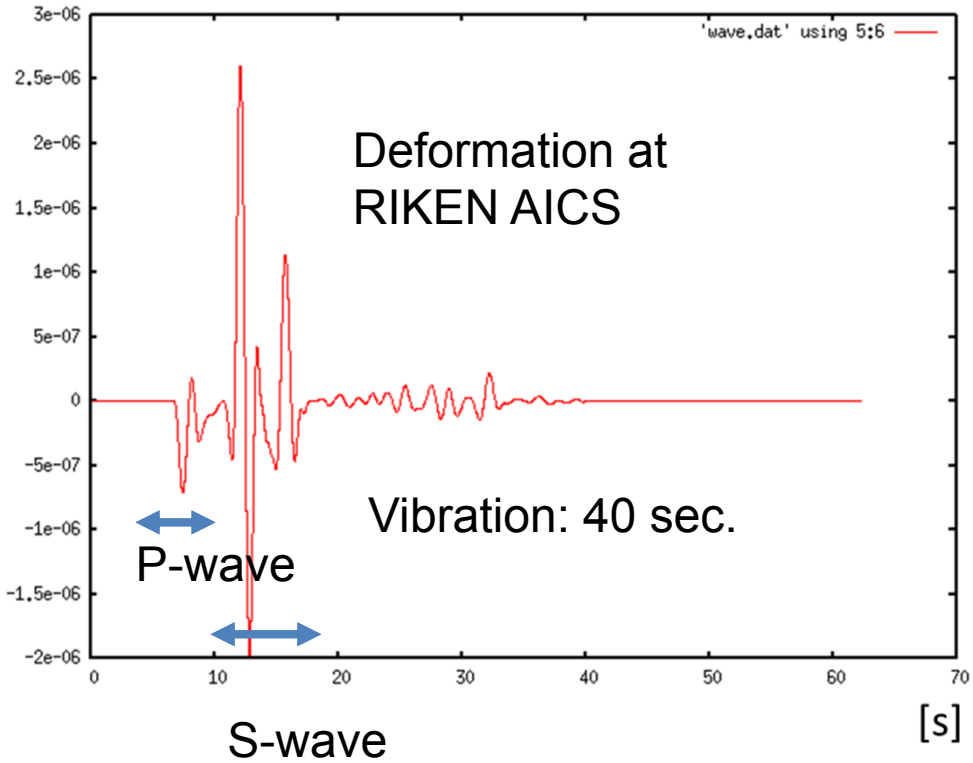
Weak-Coupled Simulation by the ppOpen-HPC Libraries

Two kinds of applications (Seism3D+ based on FDM, and FrontISTR++ based on FEM) are connected by the ppOpen-MATH/MP coupler.



1995 Kobe Earthquake M.7.3

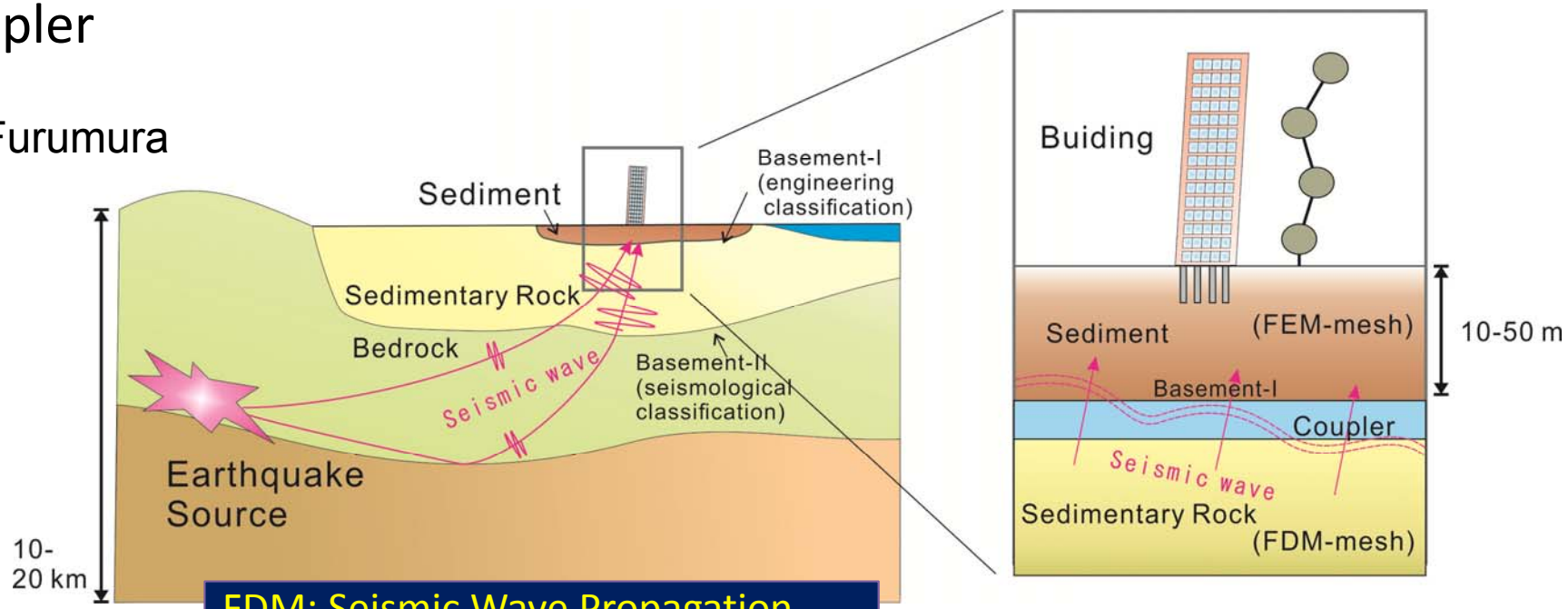


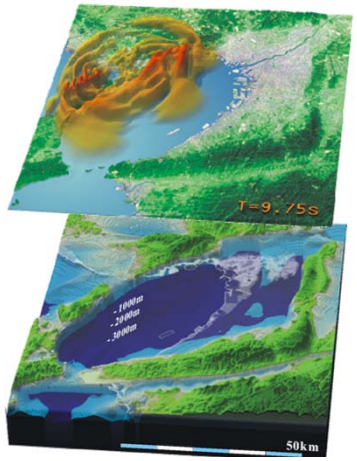


Large-Scale Coupled Simulations in FY.2013-2014

A test of a coupling simulation of FDM (regular grid) and FEM (unconstructed grid) using newly developed ppOpen-MATH/MP Coupler

c/o T.Furumura





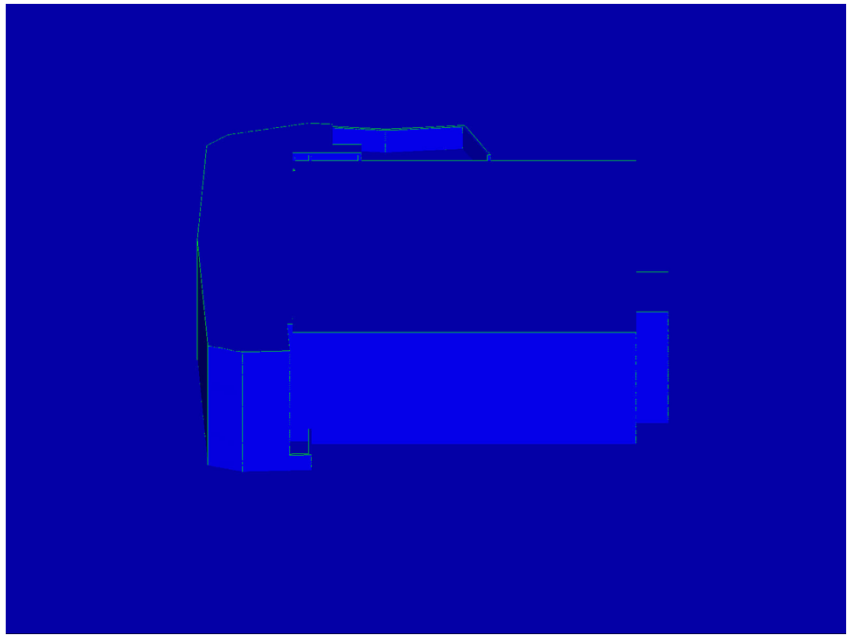
FDM: Seismic Wave Propagation
- Model size: 60x60x32 km
- Time : 90 s
- Resolution (space) : 40x40x20 m
- Resolution (time) : 1.00 ms

FEM: Building Response
- Model size : 400x400x200 m
- Time : 90 s
- Resolution (space) : 1 m, 6M nodes
- Resolution (time) : 0.2 ms

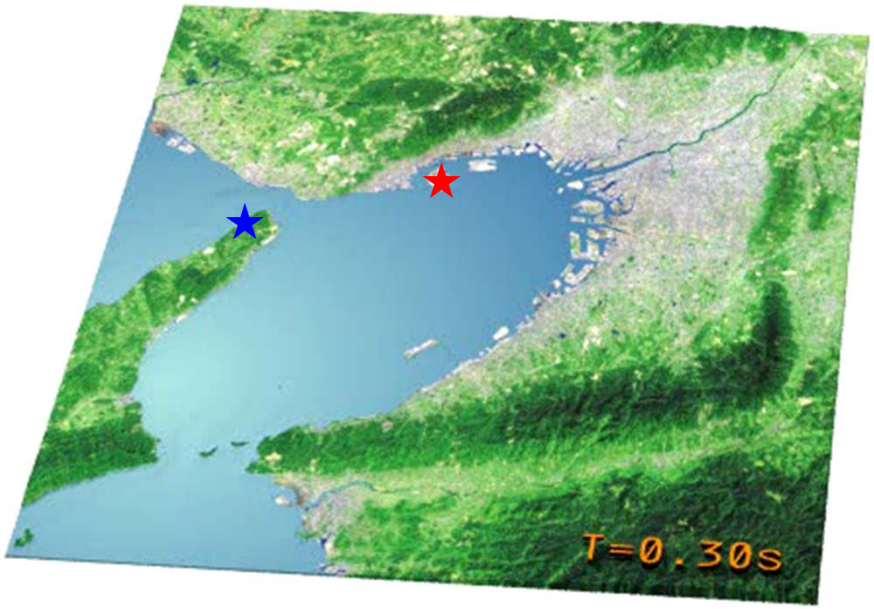
ppOpen-MATH/MP: Space-temporal interpolation, Mapping between FDM and FEM mesh, etc.

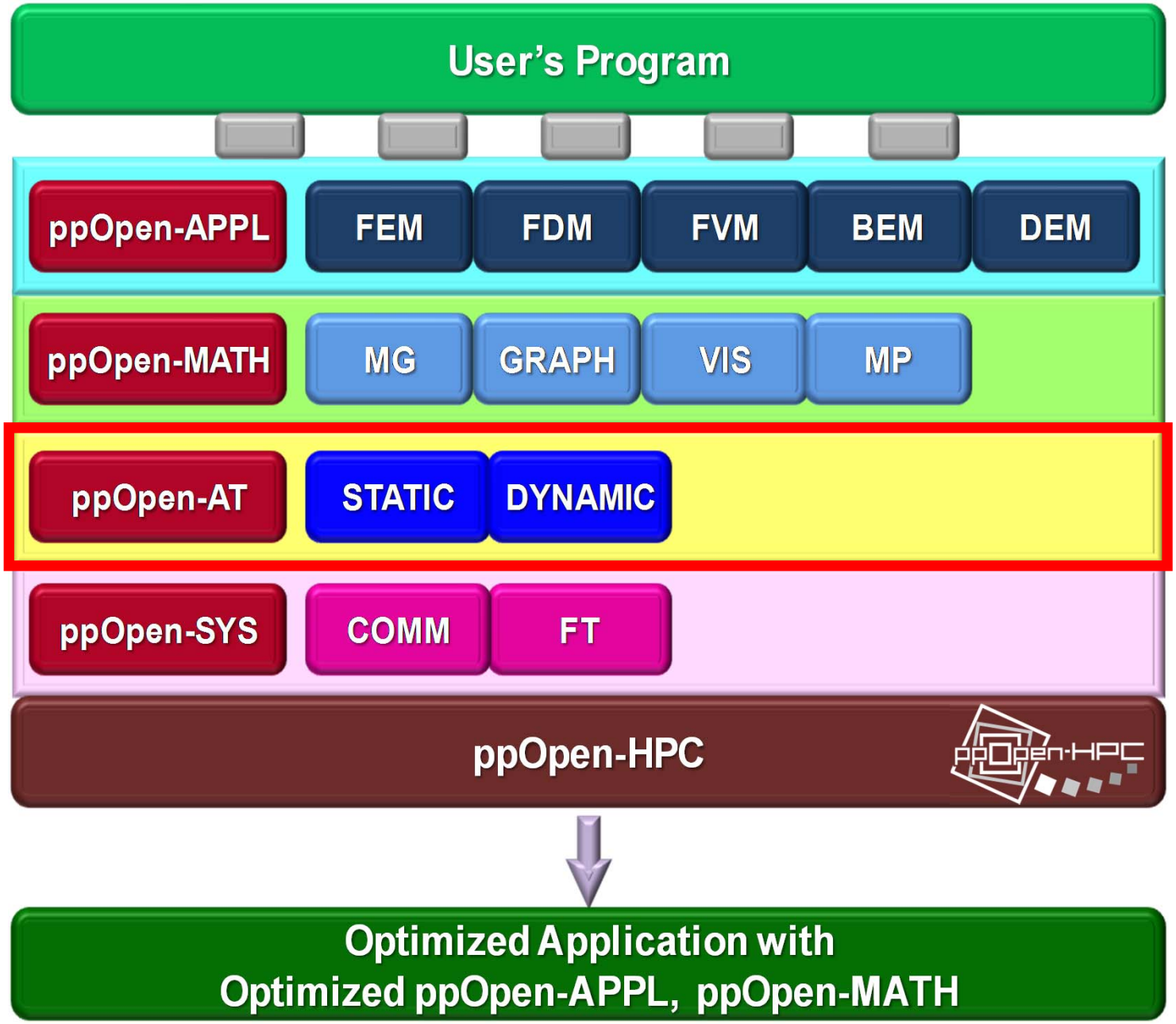
2,560 nodes for FDM, 2,000 nodes for FEM = 4,560 nodes of FX10

RIKEN AICS Building



★ Epicenter
★ RIKEN AICS




User's Program
ppOpen-APPL
FEM
FDM
FVM
BEM
DEM
ppOpen-MATH
MG
GRAPH
VIS
MP
ppOpen-AT
STATIC
DYNAMIC
ppOpen-SYS
COMM
FT
ppOpen-HPC
ppOpen-HPC
Optimized Application with
Optimized ppOpen-APPL, ppOpen-MATH

ppOpen-AT

- Automatic Tuning (AT) enables development of optimized codes and libraries on emerging architectures
- ppOpen-AT automatically and adaptively generates optimum implementation for efficient memory accesses in procedures of scientific computing in each component of ppOpen-APPL
- A directive-based special AT language is developed
  - Well-known loop transformation techniques are utilized
- ppOpen-APPL/FDM, ppOpen-APPL/BEM
- AT applied to 3D FDM code for seismic simulations developed on ppOpen-APPL/FDM for Intel Xeon/Phi

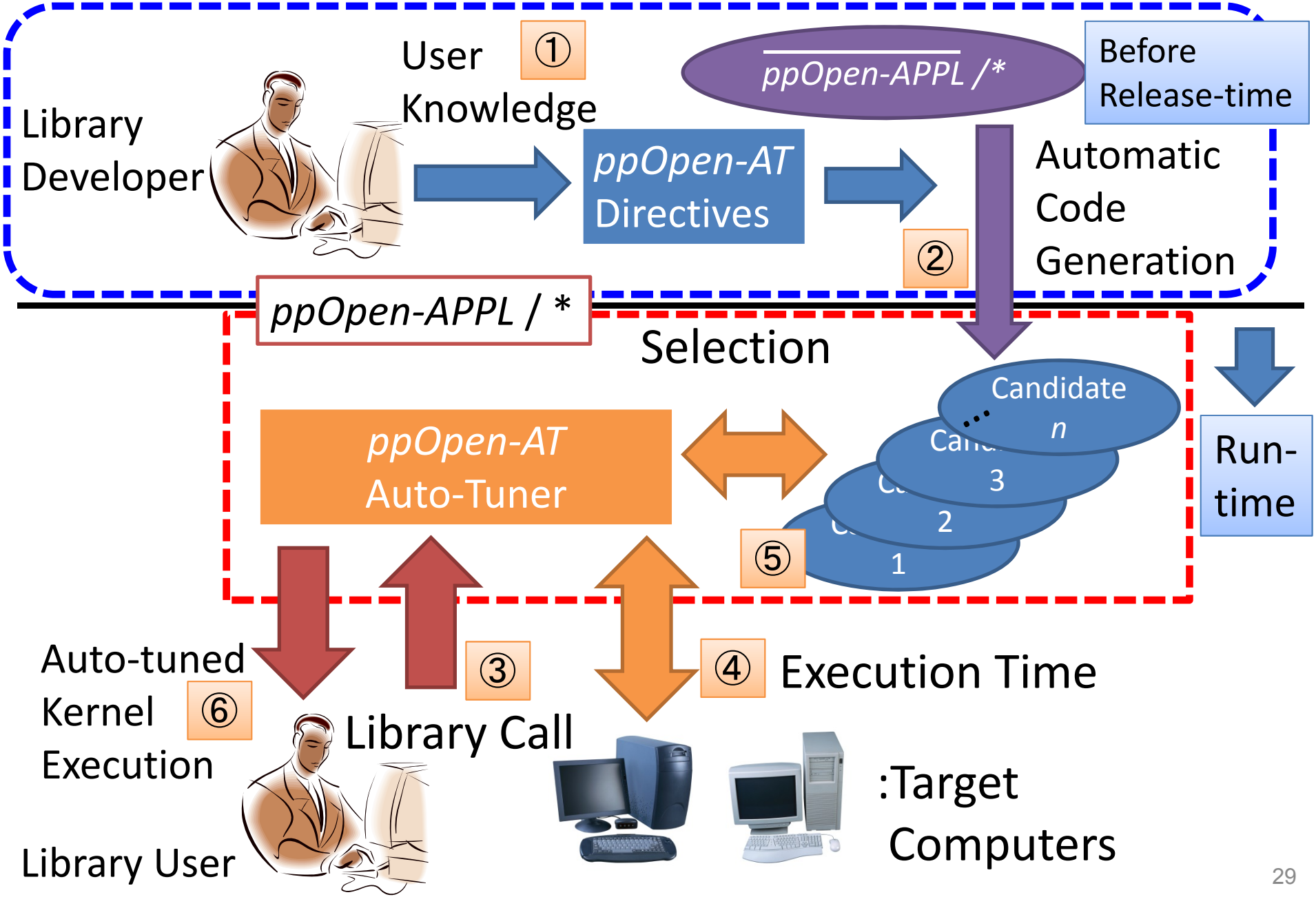
Originality (AT Languages)

AT Language / Items	# 1	# 2	# 3	# 4	# 5	# 6	# 7
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

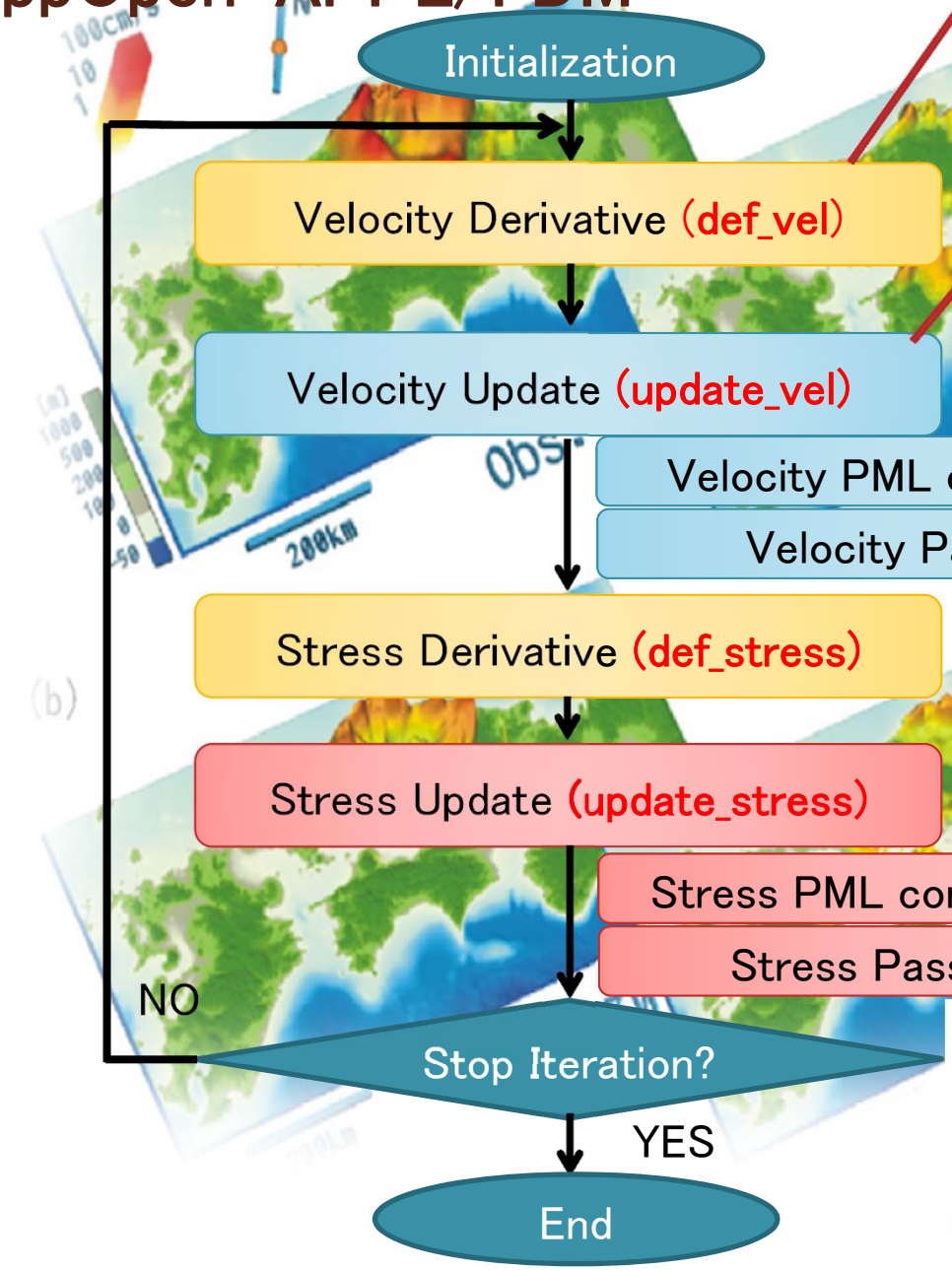
#1: Method for supporting multi-computer environments. #2: Obtaining loop length in run-time.
#3: Loop split with increase of computations, and loop fusion to the split loop.
#4: Re-ordering of inner-loop sentences. #5: Algorithm selection.
#6: Code generation with execution feedback. **#7: Software requirement.**

c/o T.Katagiri

ppOpen-AT System



Seism3D on ppOpen-APPL/FDM



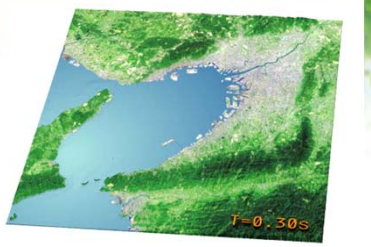
- Space difference by FDM.

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

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

- Explicit time expansion by central difference.

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



Seism3D: Code for Seismic Wave Sim. Triple-nested loops, Most Expensive

```
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
     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
  END DO
END DO
```

c/o T.Katagiri

Loop 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
```

```
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;
```

c/o T.Katagiri

Loop Fusion: Double-nested

Longer loops

Inner loop: nice for prefetching

```
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
```

c/o T.Katagiri

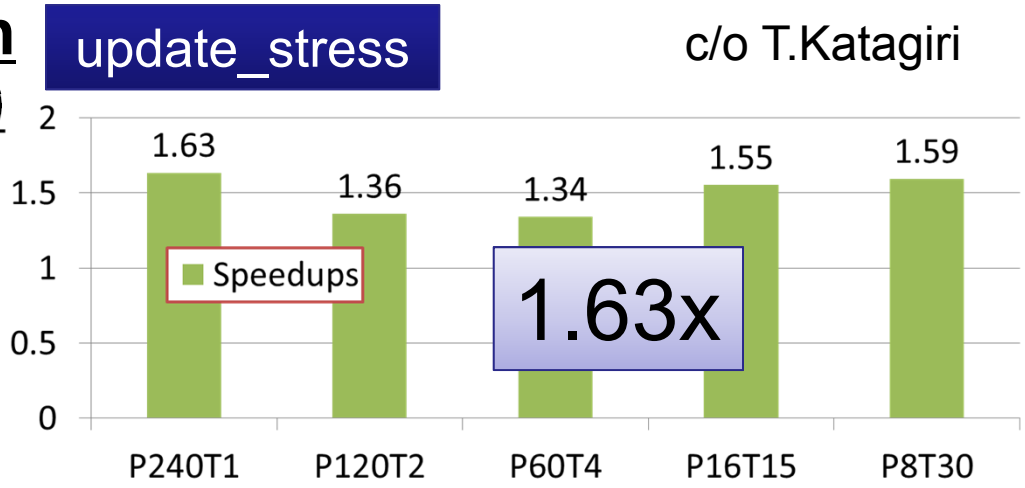
Example of Directives of ppOpen-AT

```
!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
```

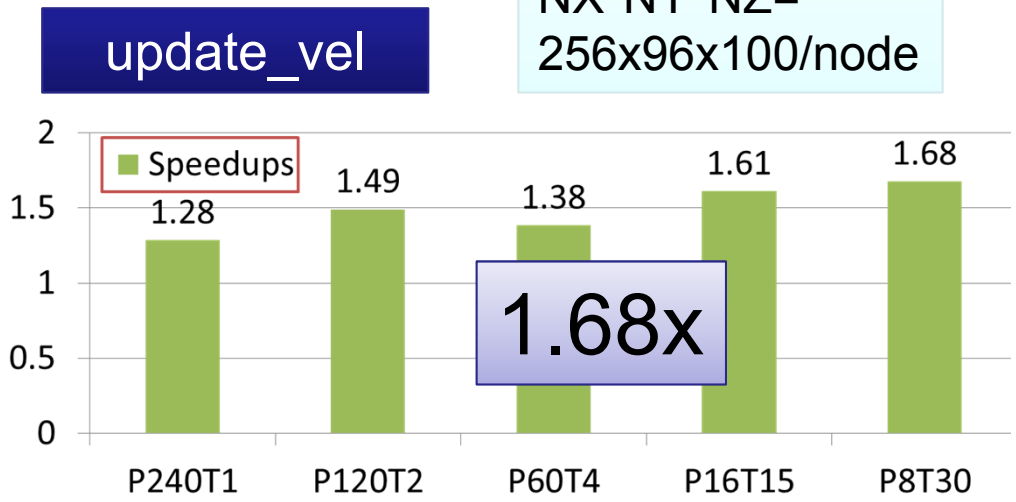
Optimization of ppOpen-APPL/FDM (Seism3D) by ppOpen-AT (FY.2013)

- **A single node of Intel Xeon Phi (60 cores, 240 threads)**
  - P240T1: Flat MPI (240 process with 1 thread)
  - P60T4: 60 proc's with 4 threads
  - Speed-up's based on execution without auto-tuning
- **`update_stress`**
  - 3-nested FDM loops, with a lot of operations
  - "Loop Splitting" is effective
- **`update_vel`**
  - 3-nested FDM loops, medium amount of operations
  - "Loop Fusion" is effective (i-j-k -> i*j-k)

c/o T.Katagiri



Problem Size:
NX*NY*NZ=
256x96x100/node



Automatic Generated Codes for the kernel 1

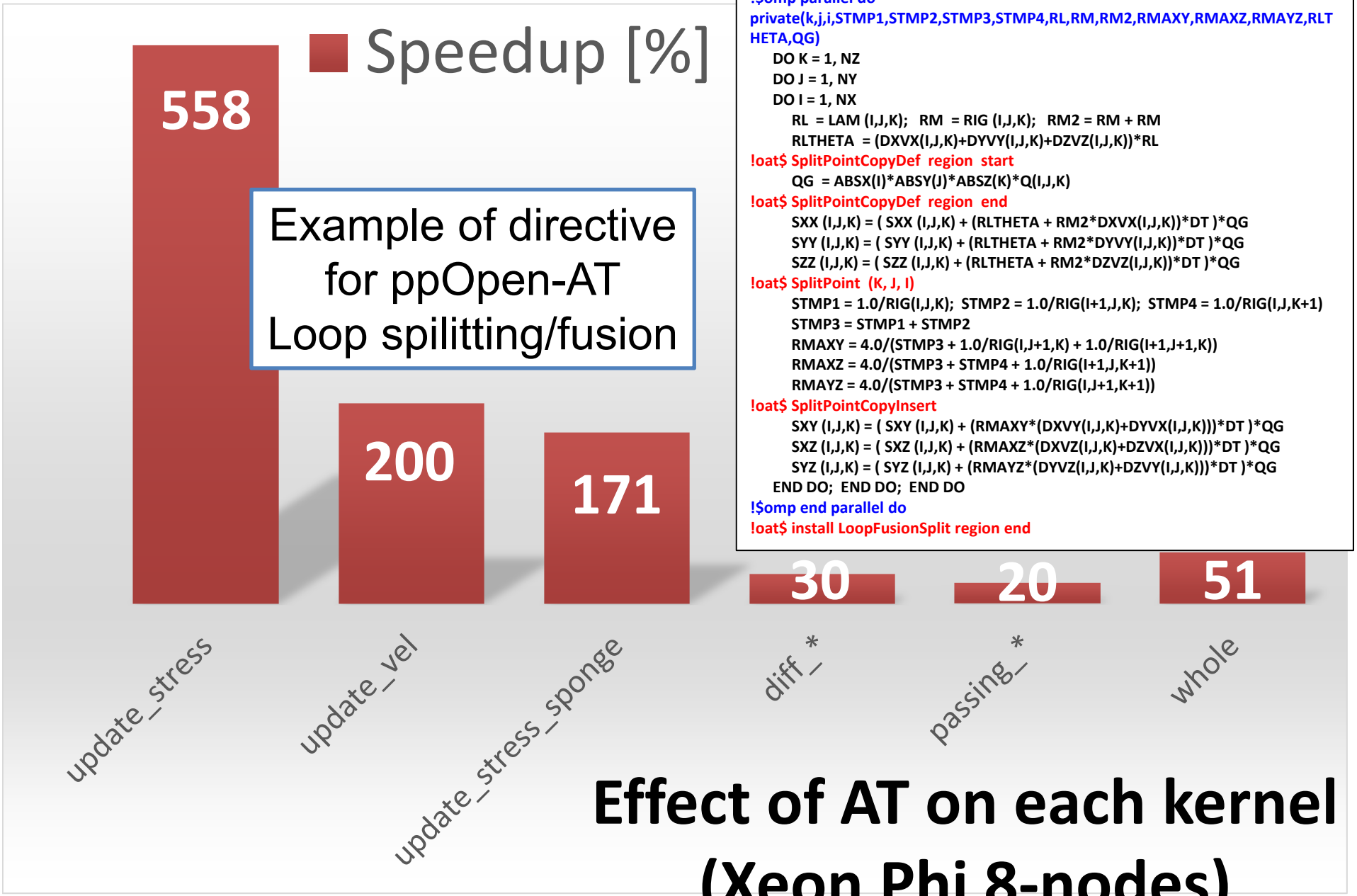
ppohFDM_update_stress

- #1 [Baseline]: Original 3-nested Loop
- #2 [Split]: Loop Splitting with K-loop
  (Separated, two 3-nested loops)
- #3 [Split]: Loop Splitting with J-loop
- #4 [Split]: Loop Splitting with I-loop
- #5 [Split&Fusion]: Loop Fusion to #1 for K and J-loops
  (2-nested loop)
- #6 [Split&Fusion]: Loop Fusion to #2 for K and J-Loops
  (2-nested loop)
- #7 [Fusion]: Loop Fusion to #1
  (loop collapse)
- #8 [Split&Fusion]: Loop Fusion to #2
  (loop collapse, two one-nest loop)

Automatic Generated Codes for the kernel 2

ppohFDM_update_vel

- #1 [Baseline]: Original 3-nested Loop.
- #2 [Fusion]: Loop Fusion for K and J-Loops. (2-nested loop)
- #3 [Fusion]: Loop Split for K, J, and I-Loops. (Loop Collapse)
- #4 [Fusion&Re-order]: Re-ordering of sentences to #1.
- #5 [Fusion&Re-order]: Re-ordering of sentences to #2.
- #6 [Fusion&Re-order]: Re-ordering of sentences to #3.

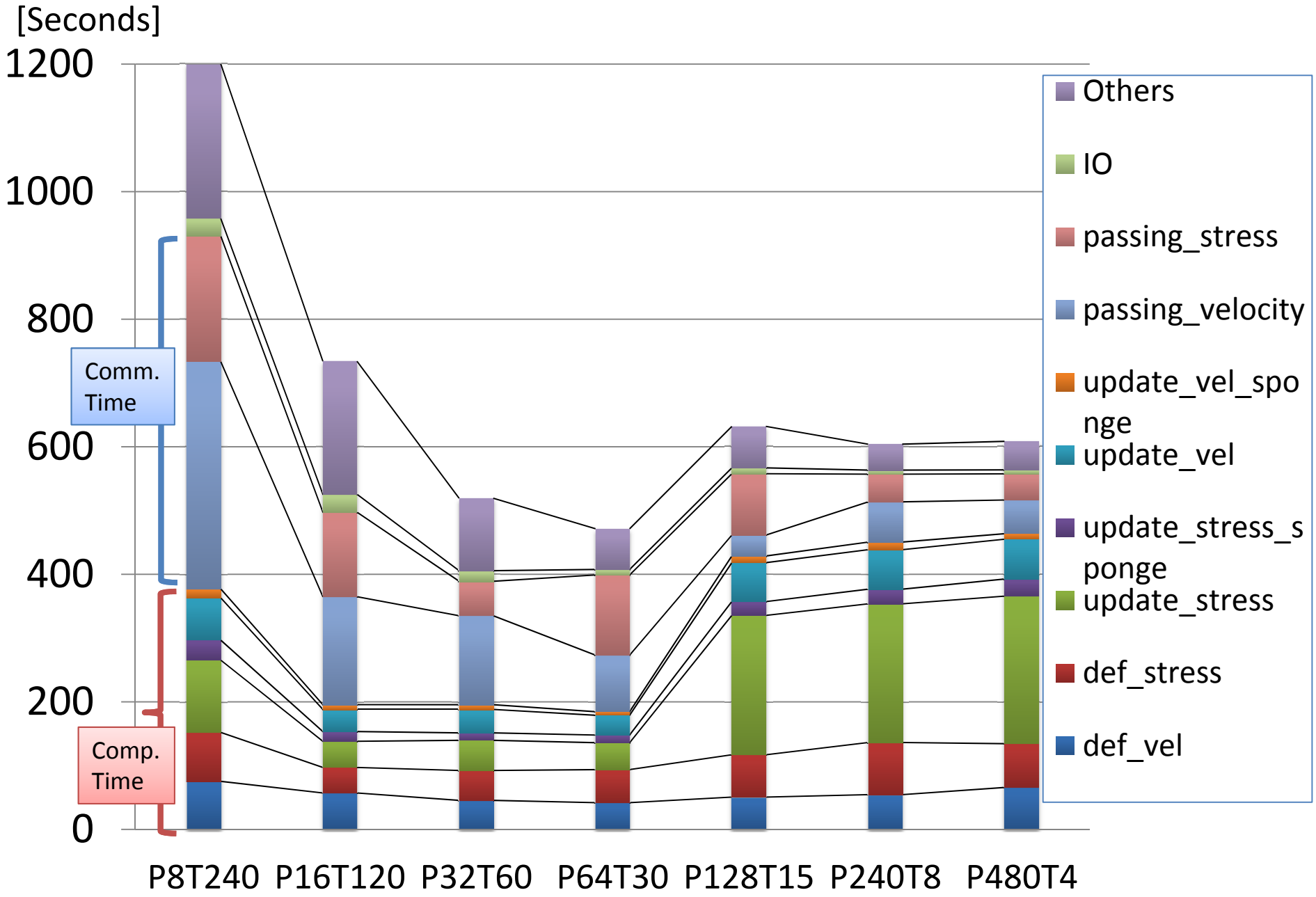


Example of directive for ppOpen-AT Loop spilitting/fusion

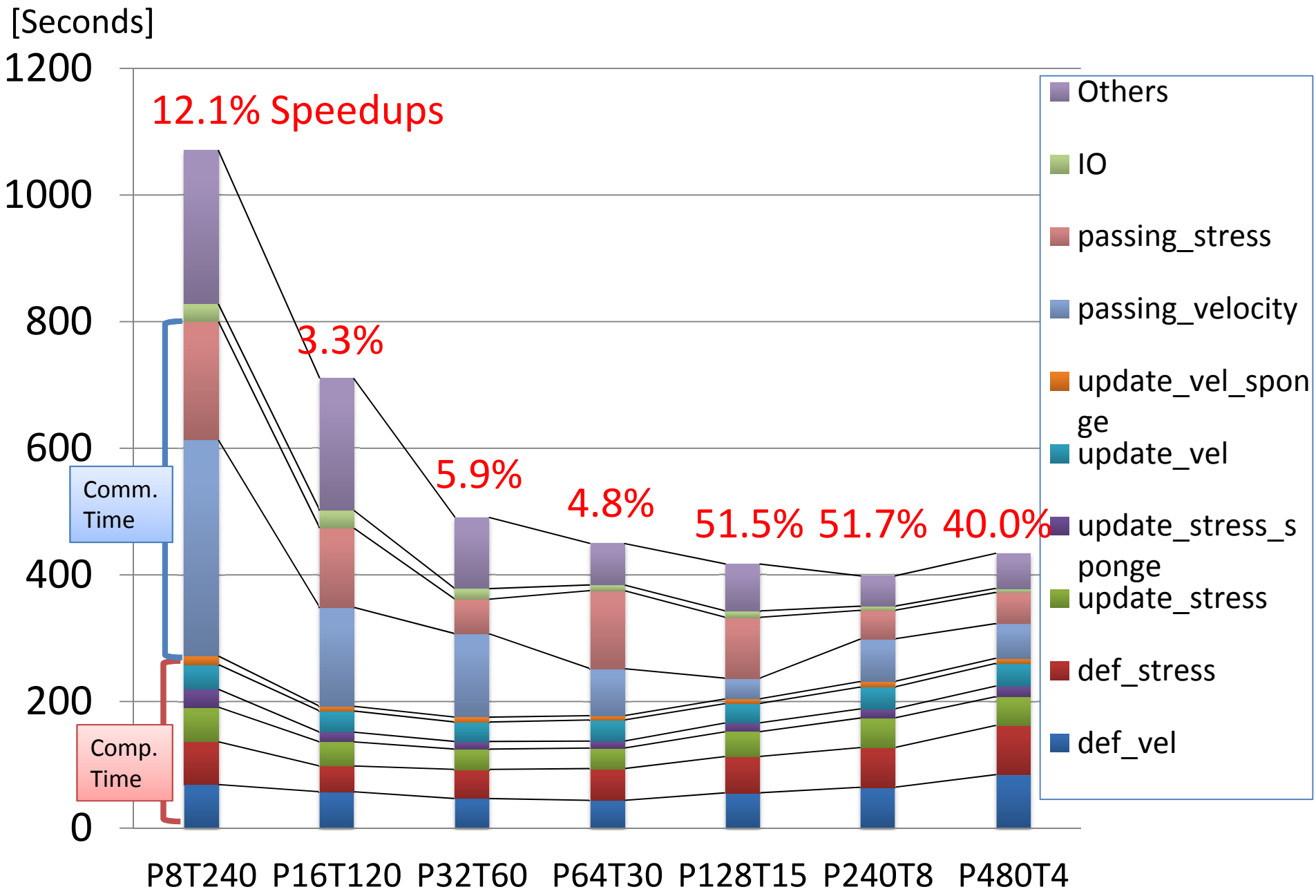
```
!oat$ install LoopFusionSplit region start
!$omp parallel do
private(k,j,i,STMP1,STMP2,STMP3,STMP4,RL,RM,RM2,RMAXY,RMAXZ,RMAYZ,RLT
HETA,QG)
    DO K = 1, NZ
    DO J = 1, NY
    DO I = 1, NX
        RL  = LAM (I,J,K);   RM  = RIG (I,J,K);   RM2 = RM + RM
        RLTHETA  = (DXVX(I,J,K)+DYVY(I,J,K)+DZVZ(I,J,K))*RL
!oat$ 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
```

Effect of AT on each kernel (Xeon Phi 8-nodes)

Whole Time（ppOpen-AT/Static, without AT）



Whole Time（ppOpen-AT/Static, with AT）



略　称	FX10	MIC	IvyB
名　称	Fujitsu SPARC64 IX fx	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon E5-2680 v2 (Ivy-Bridge-EP)
動作周波数（GHz）	1.848	1.053	2.80
コア数（有効スレッド数）	16 (16)	60 (240)	10 (20)
メモリ種別	DDR3	GDDR5	DDR3
理論演算性能（GFLOPS）	236.5	1,010.9	224.0
主記憶容量（GB）	32	8	64
理論メモリ性能（GB/sec.）	85.1	320	59.7
キャッシュ構成	L1:32KB/core L2:12MB/socket	L1:32KB/core L2:512KB/core	L1:32KB/core L2:256KB/core L3:25MB/socket

The Fastest Code (update_stress)

- Xeon Phi

#5 [Split&Fusion]: Loop Fusion to #1 for K and J-loops (2-nested loop)

```
!$omp parallel do private
(k,j,i,RL1,RM1,RM2,RLRM2,DXVX1,DYVY1,DZVZ1,D3V
3,DXVYDYVX1,DXVZDZVX1,DYVZDZV1)
 DO k_j = 1 , (NZ01-NZ00+1)*(NY01-NY00+1)
   k = (k_j-1)/(NY01-NY00+1) + NZ00;
   j = mod((k_j-1),(NY01-NY00+1)) + NY00;
   DO i = NX00, NX01
     RL1 = LAM (I,J,K);  RM1 = RIG (I,J,K);
      RM2 = RM1 + RM1; RLRM2 = RL1+RM2;
     DXVX1 = DXVX(I,J,K); DYVY1 = DYVY(I,J,K);
     DZVZ1 = DZVZ(I,J,K);
     D3V3  = DXVX1 + DYVY1 + DZVZ1;
     SXX (I,J,K) = SXX (I,J,K)
       + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
     SYY (I,J,K) = SYY (I,J,K)
       + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
     SZZ (I,J,K) = SZZ (I,J,K)
       + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1) ) * DT
     DXVYDYVX1 = DXVY(I,J,K)+DYVX(I,J,K);
     DXVZDZVX1 = DXVZ(I,J,K)+DZVX(I,J,K);
     DYVZDZVY1 = DYVZ(I,J,K)+DZVY(I,J,K)
     SXY (I,J,K) = SXY (I,J,K) + RM1 * DXVYDYVX1 * DT
     SXZ (I,J,K) = SXZ (I,J,K) + RM1 * DXVZDZVX1 * DT
     SYZ (I,J,K) = SYZ (I,J,K) + RM1 * DYVZDZVY1 * DT
   END DO
 END DO
!$omp end parallel do
```

- Ivy Bridge

#4 [Split]: Loop Splitting with I-loop

```
!$omp parallel do private
(k,j,i,RL1,RM1,RM2,RLRM2,DXVX1,DYVY1,DZVZ1,D3V
3,DXVYDYVX1,DXVZDZVX1,DYVZDZV1)
   do k = NZ00, NZ01
    do j = NY00, NY01
     do i = NX00, NX01
       RL1   = LAM (I,J,K);  RM1   = RIG (I,J,K);
       RM2   = RM1 + RM1; RLRM2 = RL1+RM2
       DXVX1 = DXVX(I,J,K);  DYVY1 = DYVY(I,J,K)
       DZVZ1 = DZVZ(I,J,K)
       D3V3  = DXVX1 + DYVY1 + DZVZ1
       SXX (I,J,K) = SXX (I,J,K)
      + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
       SYY (I,J,K) = SYY (I,J,K)
      + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
       SZZ (I,J,K) = SZZ (I,J,K)
      + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1) ) * DT
     end do
     do i = NX00, NX01
       RM1   = RIG (I,J,K)
       DXVYDYVX1 = DXVY(I,J,K)+DYVX(I,J,K)
       DXVZDZVX1 = DXVZ(I,J,K)+DZVX(I,J,K)
       DYVZDZVY1 = DYVZ(I,J,K)+DZVY(I,J,K)
      SXY (I,J,K) = SXY (I,J,K) + RM1 * DXVYDYVX1 * DT
      SXZ (I,J,K) = SXZ (I,J,K) + RM1 * DXVZDZVX1 * DT
      SYZ (I,J,K) = SYZ (I,J,K) + RM1 * DYVZDZVY1 * DT
     end do
    end do
   end do
!$omp end parallel do
```

- FX10

#1 [Baseline]: Original Loop

```
!$omp parallel do private
(k,j,i,RL1,RM1,RM2,RLRM2,DXVX1,DYVY1,DZVZ1,D3V
3,DXVYDYVX1,DXVZDZVX1,DYVZDZV1)
  do k = NZ00, NZ01
    do j = NY00, NY01
      do i = NX00, NX01
        RL1   = LAM (I,J,K);  RL1   = LAM (I,J,K)
        RM1   = RIG (I,J,K);  RM2   = RM1 + RM1
        RLRM2 = RL1+RM2;
        DXVX1 = DXVX(I,J,K); DYVY1 = DYVY(I,J,K)
        DZVZ1 = DZVZ(I,J,K)
        D3V3  = DXVX1 + DYVY1 + DZVZ1
        SXX (I,J,K) = SXX (I,J,K)
    + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1) ) * DT
        SYY (I,J,K) = SYY (I,J,K)
    + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1) ) * DT
        SZZ (I,J,K) = SZZ (I,J,K)
    + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1) ) * DT
        DXVYDYVX1 = DXVY(I,J,K)+DYVX(I,J,K)
        DXVZDZVX1 = DXVZ(I,J,K)+DZVX(I,J,K)
        DYVZDZVY1 = DYVZ(I,J,K)+DZVY(I,J,K)
     SXY (I,J,K) = SXY (I,J,K) + RM1 * DXVYDYVX1 * DT
     SXZ (I,J,K) = SXZ (I,J,K) + RM1 * DXVZDZVX1 * DT
      SYZ (I,J,K) = SYZ (I,J,K) + RM1 * DYVZDZVY1 * DT
      end do
    end do
  end do
!$omp end parallel do
```

The Fastest Code (update_vel)

- Xeon Phi

#5 [Fusion&Re-order]:
Re-ordering of sentences to #2.

- Ivy Bridge

#5 [Fusion&Re-order]:
Re-ordering of sentences to #2.

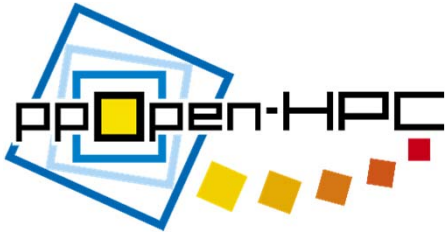
- FX10

#1 [Baseline]: Original Loop

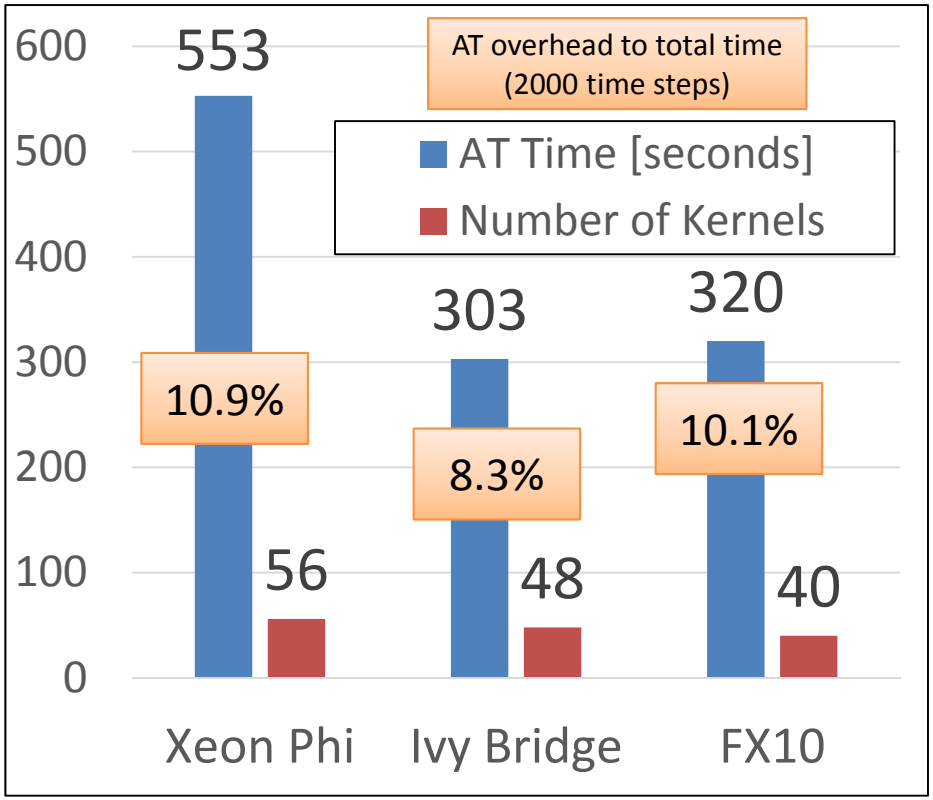
```
!$omp parallel do private (i,j,k,ROX,ROY,ROZ)
DO k_j = 1 , (NZ01-NZ00+1)*(NY01-NY00+1)
   k = (k_j-1)/(NY01-NY00+1) + NZ00
   j = mod((k_j-1),(NY01-NY00+1)) + NY00
   do i = NX00, NX01
      ROX = 2.0_PN/( DEN(I,J,K) + DEN(I+1,J,K) )
      VX(I,J,K) = VX(I,J,K) &
         + ( DXSXX(I,J,K)+DYSXY(I,J,K)+DZSXZ(I,J,K) )*ROX*DT
      ROY = 2.0_PN/( DEN(I,J,K) + DEN(I,J+1,K) )
      VY(I,J,K) = VY(I,J,K) &
         + ( DXSXY(I,J,K)+DYSYY(I,J,K)+DZSYZ(I,J,K) )*ROY*DT
      ROZ = 2.0_PN/( DEN(I,J,K) + DEN(I,J,K+1) )
      VZ(I,J,K) = VZ(I,J,K) &
         + ( DXSXZ(I,J,K)+DYSYZ(I,J,K)+DZSZZ(I,J,K) )*ROZ*DT
   end do
end do
!$omp end parallel do
```

```
!$omp parallel do private (i,j,k,ROX,ROY,ROZ)
 do k = NZ00, NZ01
    do j = NY00, NY01
       do i = NX00, NX01
          ROX = 2.0_PN/( DEN(I,J,K) + DEN(I+1,J,K) )
          ROY = 2.0_PN/( DEN(I,J,K) + DEN(I,J+1,K) )
          ROZ = 2.0_PN/( DEN(I,J,K) + DEN(I,J,K+1) )
          VX(I,J,K) = VX(I,J,K)  +
( DXSXX(I,J,K)+DYSXY(I,J,K)+DZSXZ(I,J,K) )*ROX*DT
          VY(I,J,K) = VY(I,J,K) +
( DXSXY(I,J,K)+DYSYY(I,J,K)+DZSYZ(I,J,K) )*ROY*DT
          VZ(I,J,K) = VZ(I,J,K) +
( DXSXZ(I,J,K)+DYSYZ(I,J,K)+DZSZZ(I,J,K) )*ROZ*DT
       end do
    end do
  end do
!$omp end parallel do
```

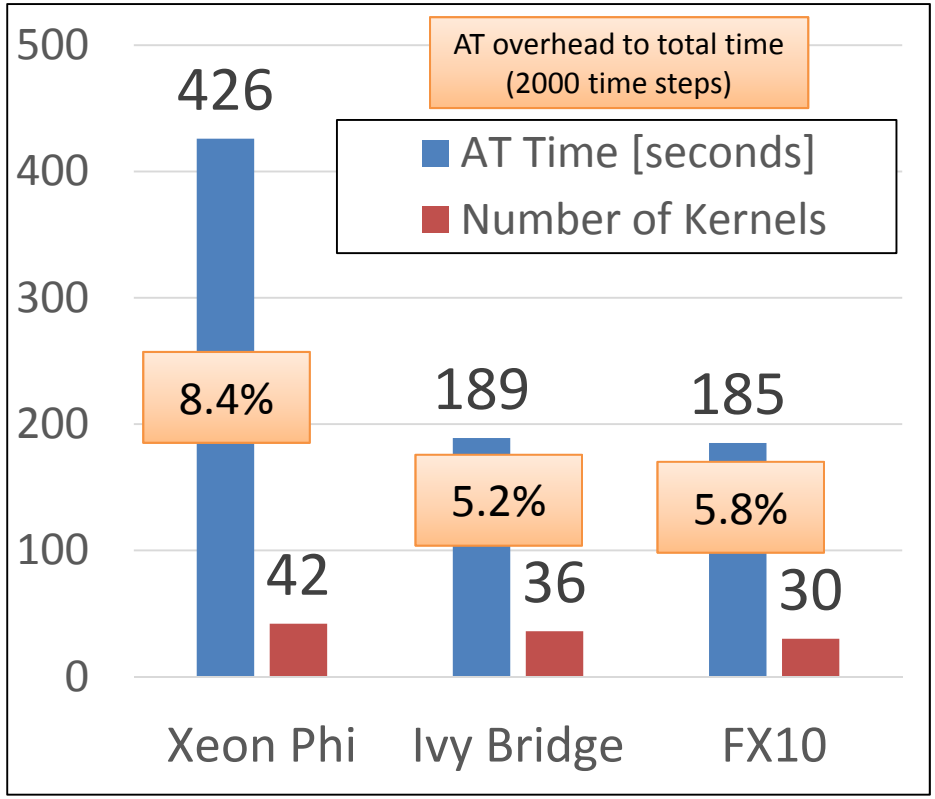
Auto-tuning Time



- update_stress



- update_vel



Speed-up by AT

略　称	FX10	MIC	IvyB
名　称	Fujitsu SPARC64 IX fx	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon E5-2680 v2 (Ivy-Bridge-EP)
動作周波数（GHz）	1.848	1.053	2.80
コア数（有効スレッド数）	16 (16)	60 (240)	10 (20)
メモリ種別	DDR3	GDDR5	DDR3
理論演算性能（GFLOPS）	236.5	1,010.9	224.0
without AT (Flat MPI)（GFLOPS/socket）	**20.0**	**19.4**	**20.4**
with AT（GFLOPS/socket）	**20.4 P240T8**	**30.0 P128T1**	**23.4 P80T1**

Features & Future Works in ppOpen-AT

- Ref.: Talk by Prof. Katagiri in this morning
- Strongly depends on intelligence/experiences of the users: manual operations ... but it is ok (education)
  - Configurations of scenarios, Data dependency
  - Locations of directives
  - Effects of problem size, hardware parameters etc.
  - Parallel prog. model (#MPI processes x #OpenMP threads)
- Analyses of assembly codes (for research paper)
- Operations for sparse matrices (not limited to SpMV)
  - Blocking + X-**ELL**-Y-Z
  - Different from FDM kernels
- DSL's for stencil computing: Physis, ExaStencil etc.

Schedule of Public Release

(with English Documents, MIT License)

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

- Released at SC-XY (or can be downloaded)
- Multicore/manycore cluster version (Flat MPI, OpenMP/MPI Hybrid) with documents in English
- We are now focusing on MIC/Xeon Phi
- Collaborations with scientists are welcome

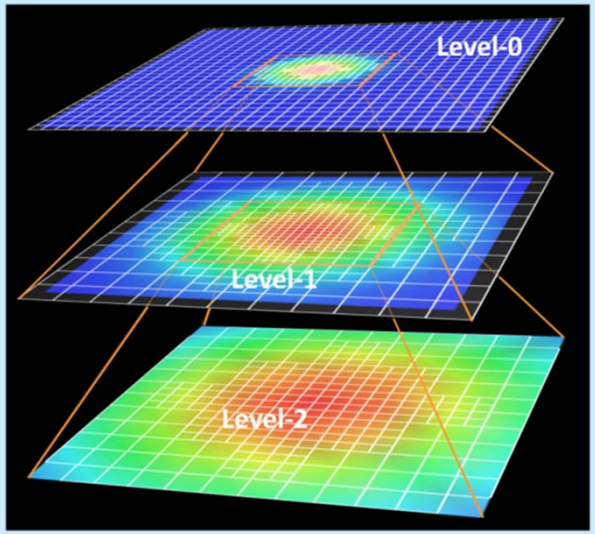
History

- SC12, Nov 2012 (Ver.0.1.0)
- SC13, Nov 2013 (Ver.0.2.0)
- SC14, Nov 2014 (Ver.0.3.0)

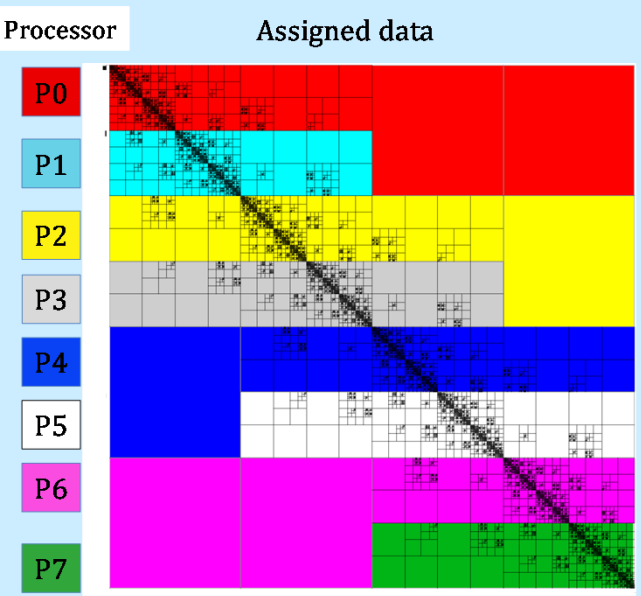
New Features in Ver.0.3.0

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

- ppOpen-APPL/AMR-FDM: AMR framework with a dynamic load-balancing method for various FDM applications



- HACApK library for H-matrix comp. in ppOpen-APPL/BEM
  - Akihiro Ida (Kyoto U.)



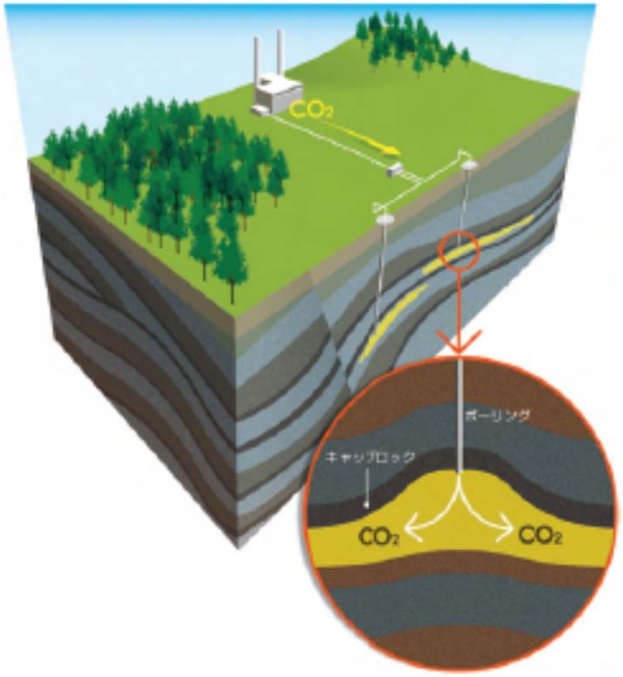
Assign a^c to P_k if $S(a^c_{ij})$ in $R(P_k)$

A : Whole matrix
A_{IJ} : The entry in the I-th row and the J-th column of A
a^c: A small submatrix of A
$a^c{}_{ij}$: The entry in the i-th row and the j-th column of a^c
N : Number of rows of A
n : Number of processors
P_k : The k-th processor
l_k : $1 = l_0 < \cdots < l_k < \cdots < l_{n+1} = N + 1$
R : $R(P_k) = \{i \mid l_k \le i < l_{k+1}\}$
S : $S(a^c{}_{ij}) = I$ when $a^c{}_{ij} = A_{IJ}$

- Utilities for pre-processing in ppOpen-APPL/DEM

Collaborations, Outreaching

- Collaborations
  - International Collaborations
    - Lawrence Berkeley National Lab.
    - National Taiwan University
    - IPCC（Intel Parallel Computing Center）
- Outreaching, Applications
  - Large-Scale Simulations
    - Geologic CO_2 Storage
    - Astrophysics
    - Earthquake Simulations etc.
    - ppOpen-AT, ppOpen-MATH/VIS, ppOpen-MATH/MP, Linear Solvers
  - Intl. Workshops (2012, 2013)
  - Tutorials, Classes



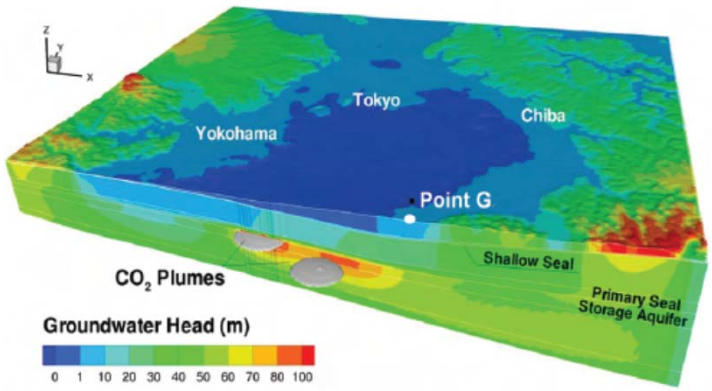


図-4 CO_2圧入後の地下水圧（全水頭換算）の分布（100年後）

from Post-Peta to Exascale

- Currently, we are focusing on Post-T2K system by manycore architectures (Intel Xeon/Phi)
- Outline of the Next Generation Systems is much clearer than which were in 2011 (when this project started).
  - Frameworks like ppOpen-HPC are really needed
    - More complex, and huge system
    - More difficult to extract performance of applications
  - Smooth transition from post-peta to exa will be possible through continuous development and improvement of ppOpen-HPC