Sustainable HPC software: From Auto-tuning Technology for Performance Portability

Takahiro Katagiri

Information technology Center, Nagoya University, Japan

eScience2024, Senri Life Science Center (Osaka Senri)
WS: Sustainable Project Pathways for HPC Software and Applications, 14:00 – 17:00,
Room: MEGARON B, 17th September, 2024

名古屋大学
NAGOYA UNIVERSITY

Outline

1. Background: Performance Portability and Auto-tuning
2. Case1: CMOS Annealing Machine
3. Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK
4. Conclusion

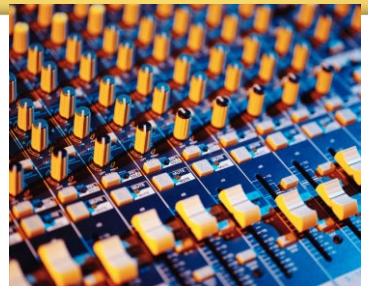
Outline

1. Background: Performance Portability and Auto-tuning
2. Case1: CMOS Annealing Machine
3. Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK
4. Conclusion

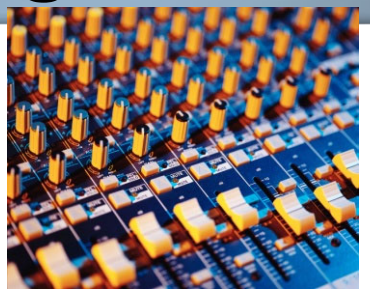
What is Auto-tuning (AT)?

- Computer Architectures
- Computer Systems

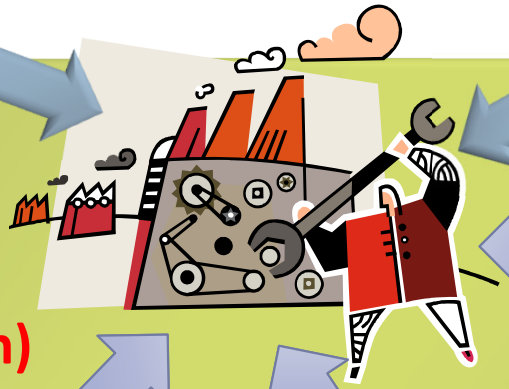
- Programs
- Algorithms

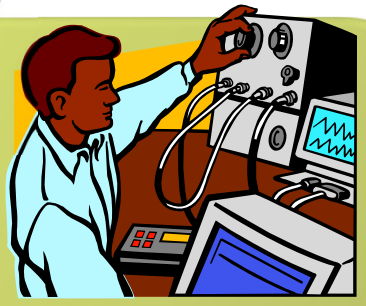


Performance Tunable Knobs
(Performance Parameters)



Adjustment Facility
- Optimization
- Parameter Search
- **AI (Learning / Self-Adaption)**

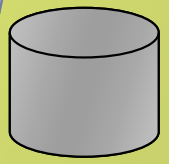




Performance Monitoring Function

- Programs
- Algorithms

Knob Auto-Generation Function

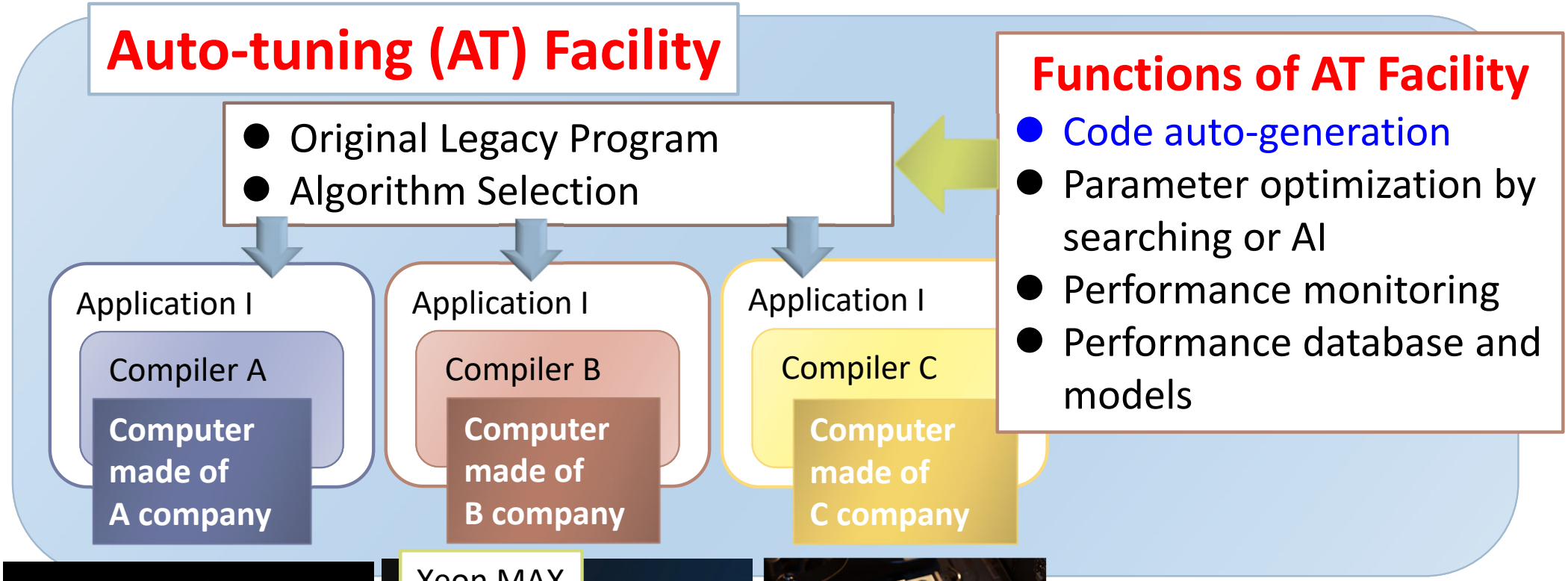


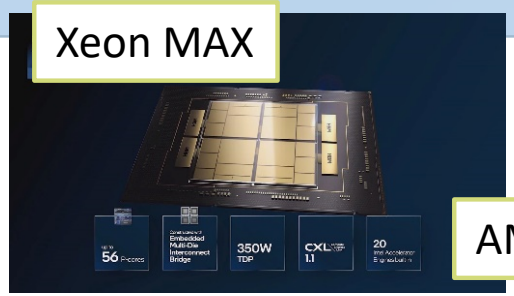
Performance Database

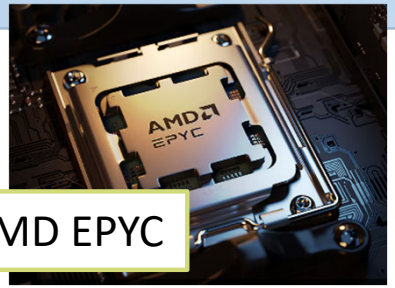
Auto-tuning (AT) Facility

Performance Portability (PP)

- **A paradigm for optimizing multiple computer environments.**
  - Ensuring high performance with legacy programs across diverse systems. -> "Performance Portability (PP)"







Several novel computer environments are emerging as we move toward the post-exascale era.

Metric of PP

1. Set a test suite *T** (input data and problem size, etc.)
2. Determine standard environment (System *S**)
3. Measure **performance *S**** on System *S** using Test Suite *T**.
4. Determine target environment (System S^1)
5. Measure **performance S^1** on System S^1 using Test Suite *T**.

Performance Portability *PP(System, Test Suite)* is defined by:

$$PP(System\ S^1, Test\ Suite\ T^*) = \frac{Performance\ S^1}{Performance\ S^*}$$

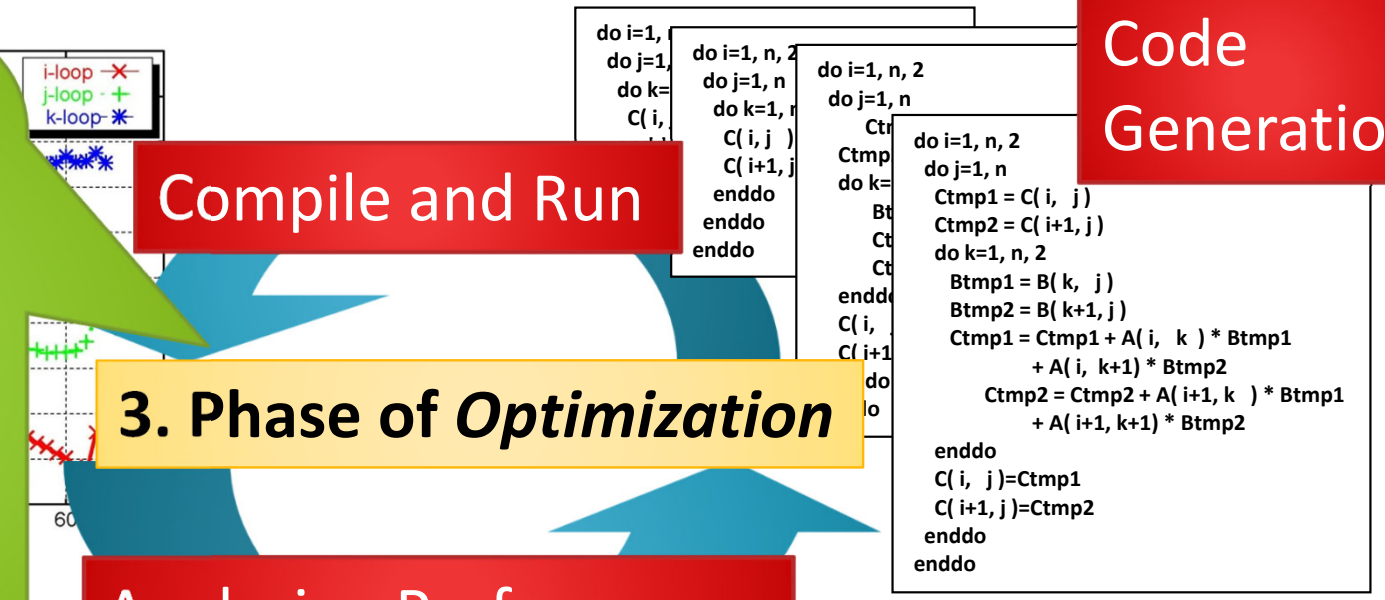
$$\Sigma_{i=1}^{N} PP(System\ S^i, Test\ Suite\ T^*)/N$$

(Average of each *PP()* on system S^i, i = 1, …, N) is a candidate of metrics of PP.

Development Flow of HPC Software

Programming Models, Code Optimizations, Resource Allocations, etc.

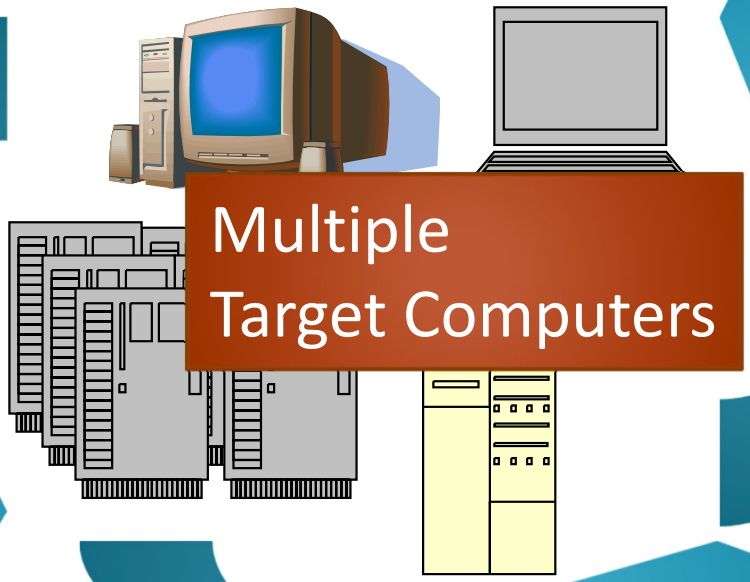
Code Generation



Compile and Run

3. Phase of *Optimization*

Analyzing Performance



Multiple Target Computers

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
```

4. Phase of *Knowledge of Discovery for Tuning*

Database and Performance Model for Tuning Knowledge

1. Phase of *Specification*

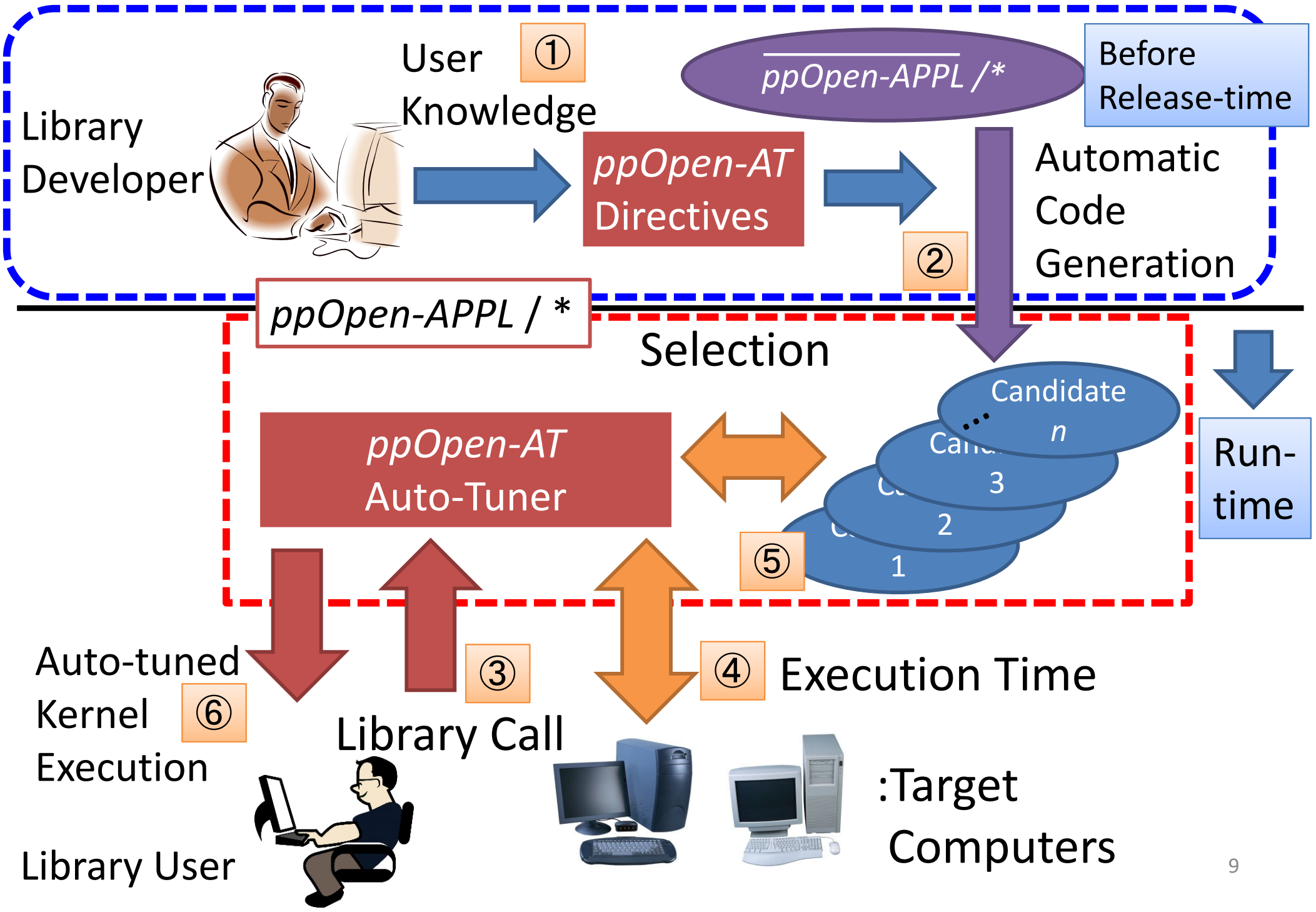
Cost of HPC Software Development

1. Set a test suite (input data and problem size, etc.)
2. Set a target performance (with respect to Performance Portability)
3. Make specification
4. Programming
5. Performance tuning (Performance profiling and analysing, Modification of codes, and Tuning performance parameters, etc.)
6. Summarize tuning experience and make a performance model
7. Check the target performance
8. If current performance is not established for the target performance, back to 3.

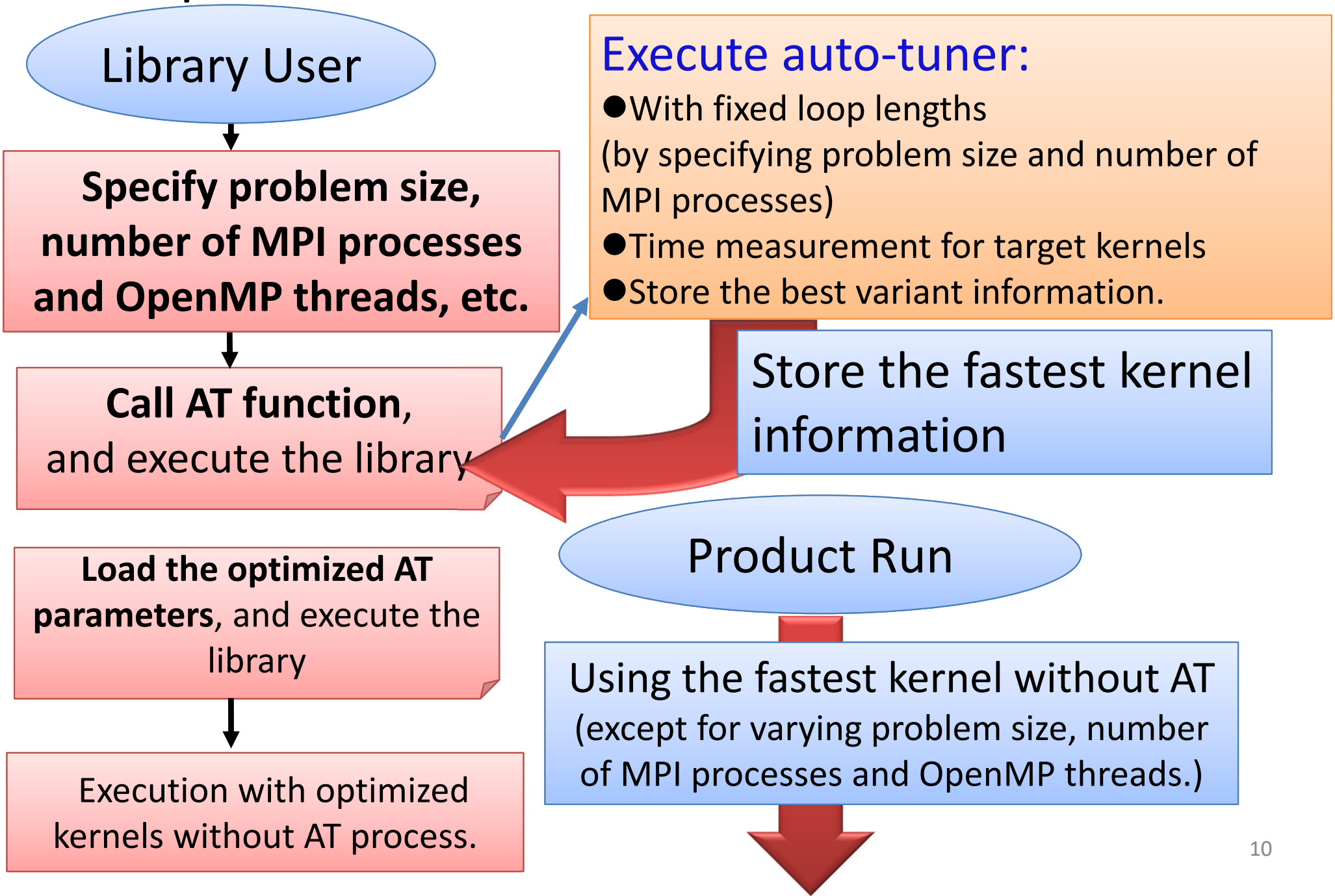
$$Cost = \Sigma_{i=3}^{7} Time(i)$$

$\min(Cost)$ is the aim by adapting AT technology, but current AT focuses on process 5 as usual.

ppOpen-AT System (Based on FIBER)



A Scenario of AT by ppOpen-AT for code optimization of a kernel of a simulation



An example: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

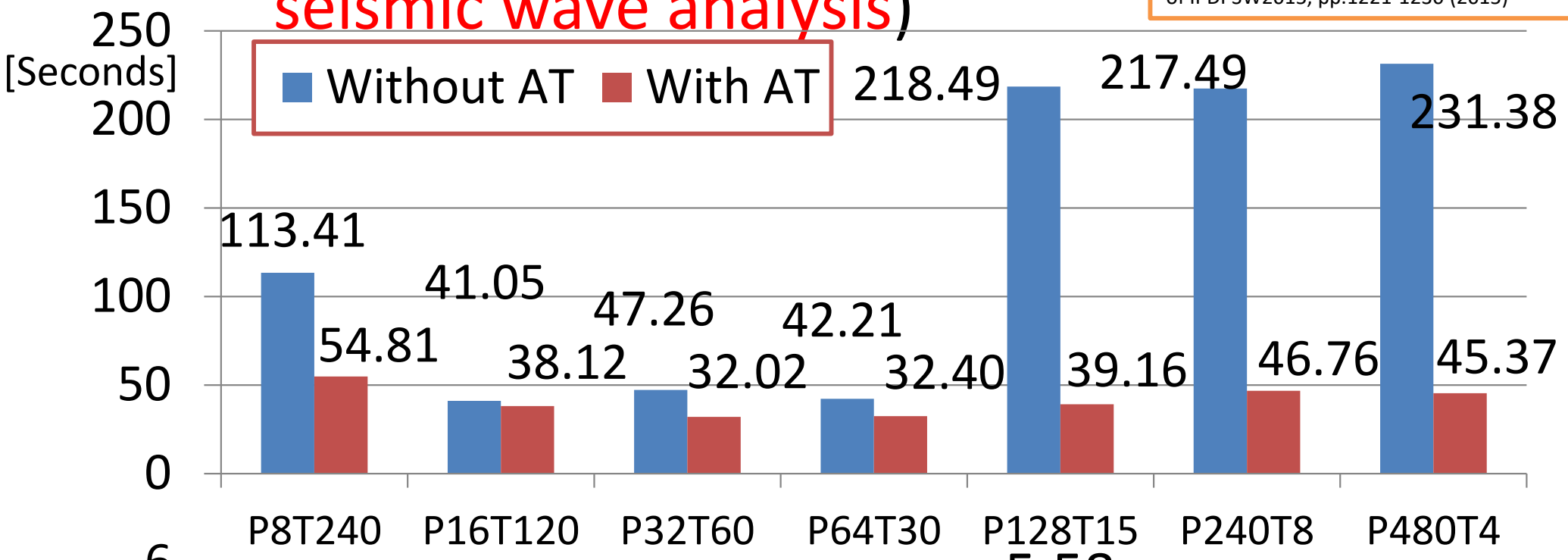
Re-calculation is defined.

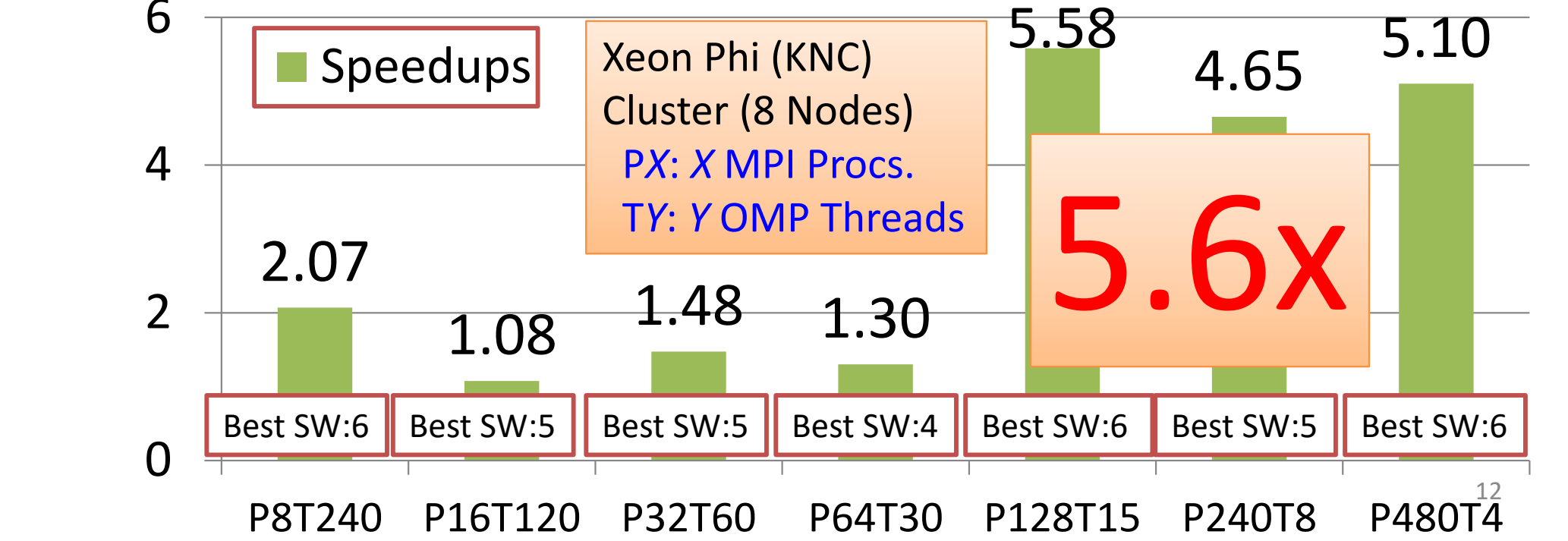
Loop Split Point

Using the re-calculation is defined.

AT Effect (A Kernel for simulator of seismic wave analysis)

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)





Outline

1. Background: Performance Portability and Auto-tuning
2. Case1: CMOS Annealing Machine
3. Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK
4. Conclusion

Case1:
CMOS Annealing Machine

Collaborator: **Mr. Makoto Morishita**
(D3, Nagoya University)

名古屋大学
NAGOYA UNIVERSITY

Background

- Experiments are being conducted to confirm **quantum supremacy** by Google and the University of Science and Technology of China, and quantum computers are attracting attention[1].
- Quantum-inspired computers are being developed in Japan to follow quantum computers.
  - Example of quantum-inspired computers:
    1. **CMOS Annealing Machine（Hitachi）[2]**
    2. Digital Annealer（Fujitsu）
    3. Simulated Bifurcation Machine（Toshiba）
- Development of quantum circuit simulators utilizing GPUs is also gaining momentum.
  - Cirq, Qiskit, cuQuantum, *etc*.

[1] ”Hello quantum world! Google publishes landmark quantum supremacy claim”, Nature, 23 October 2019
[2] Masanao Yamaoka, et al. "20k-spin Ising chip for combinational optimization problem with CMOS annealing", ISSCC, 2015

CMOS Annealing Machine (1/2)

"A business-card size"
CMOS annular (ASIC, 4 bits)

- Positioning of Quantum-inspired machines

Quantum computers

Gate-base
- IBM Quantum
- System One

Annealing-base
- D-Wave 2000Q
- D-Wave Advantage

Cooling system is required, low number of qubits

Quantum-inspired machine

Annealing-base
- **CMOS Annealing Machine**
- Digital Annealer
- Simulated Bifurcation Machine

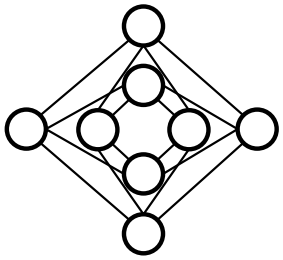
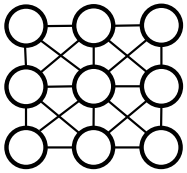
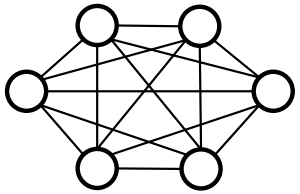
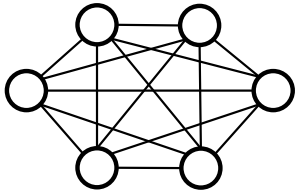
Operates at room temperature, relatively large number of qubits

- What is CMOS Annealing Machine?
  - Developed by Hitachi, Ltd. in 2015
  - A specialized computer that performs ground state search for the **Ising model**
  - Providing cloud services **Annealing Cloud Web**[3] (GPU version, 32bits, Float) to expand the number of users

[3] https://annealing-cloud.com/ja/index.html

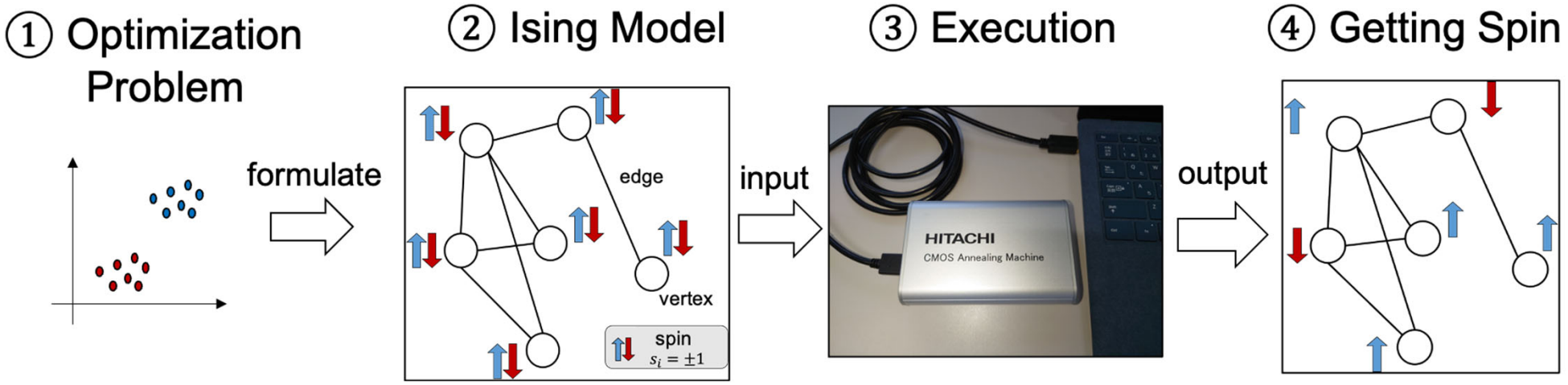
CMOS Annealing Machine (2/3)

- Specification of current annealing machines

	D-Wave 2000Q	CMOS Annealing Machine		Digital Annealer	Simulated Bifurcation Machine
		ASIC version	GPU version		
Hardware	QPU	Digital circuit	GPU	Digital circuit	GPU
Qubits	2,048	61,952	262,144	8,192	10,000
Graph	Chimera graph 	King's graph (Only ASIC)  *GPU version is fully connected.		Complete graph 	Complete graph 

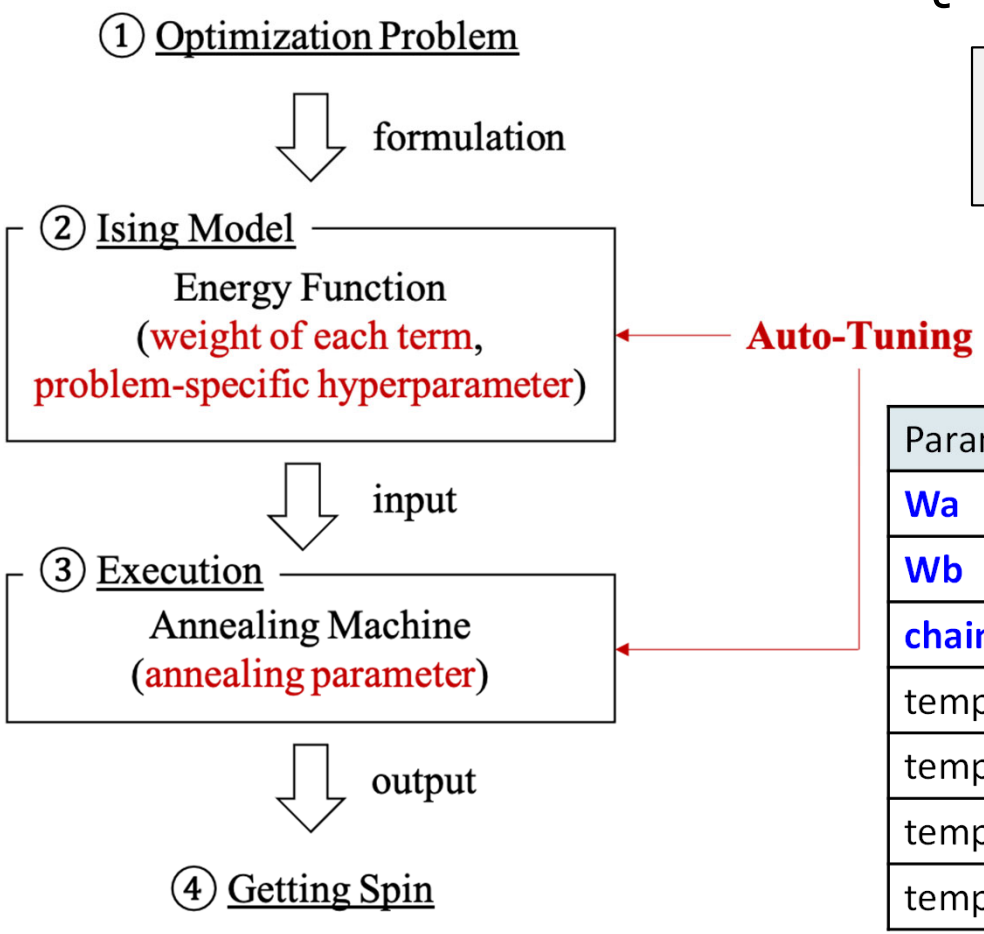
CMOS Annealing Machine（2/2）

- Solving procedure using CMOS annealing machine



Tunable Parameters on Quantum-Inspired Annealers

Auto-tuning Framework for Hyper-parameters (Annealing-base)



QUBO formula

$$H = w_a \sum_{(u,v)\in E} (1 - x_u)(1 - x_v) + w_b \sum_{v \in V'} x_v$$

constraint

cost

Parameters to be tuned when solving problems with CMOS annealing machines

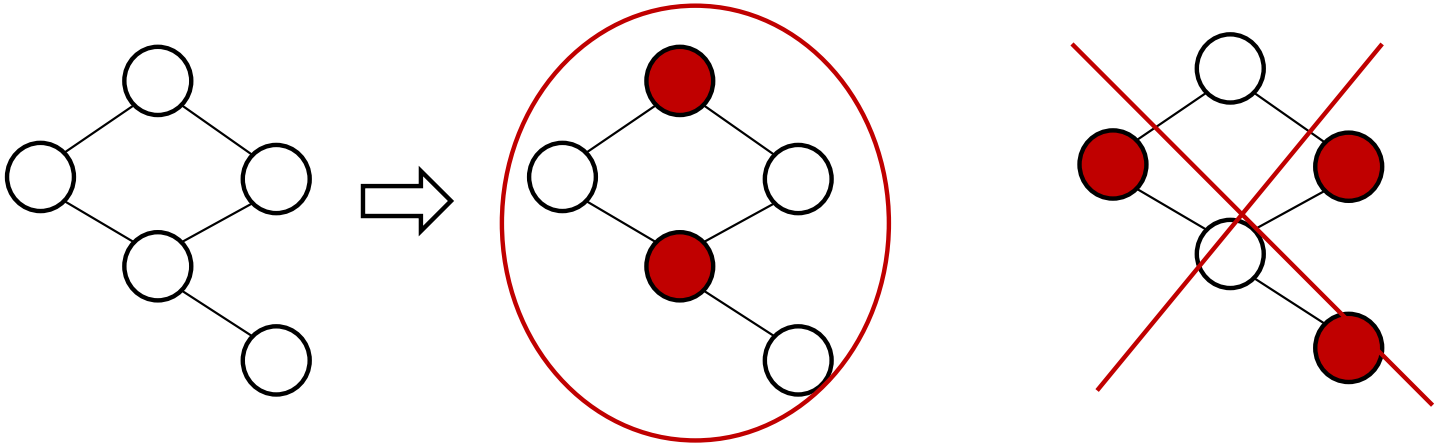
Parameters	Overview
Wa	**Coefficient of constraint term**
Wb	**Coefficient of cost term**
chain_strength	**Strength of chain**
temperature_num_steps	Number of steps in annealing
temperature_step_length	Length of steps in annealing
temperature_initial	Initial temperature in annealing
temperature_target	Final temperature in annealing

名古屋大学 NAGOYA UNIVERSITY

Experiment Settings（1/3）

- Minimum Vertex Cover
  - Find V′, which is the vertex covering set (where $|V'|$ is the minimum, Graph $G = (V, E)$, $V' \subseteq V$)

※Vertex set $V' \subseteq V$is the vertex covering of $G = (V, E)$:
"For all edge $e \in E$, at least one of the endpoints is included in V' "



Example : $|V| = 5$ $\quad$ **$|V'| = 2$ (Minimum)** $\quad$ $|V'| = 3$ (Not minimum)

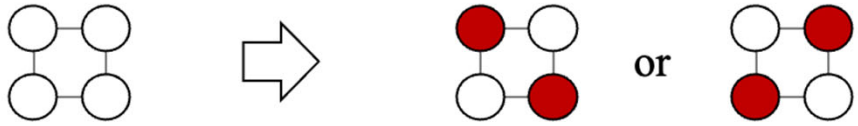
名古屋大学 NAGOYA UNIVERSITY

Experiment Settings（2/3）

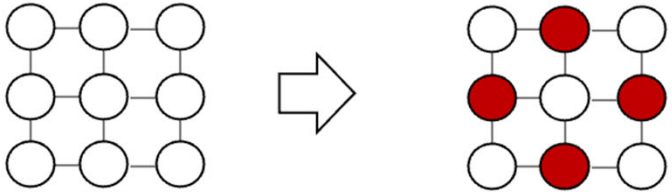
- **Artificial Problem:**
  Minimum Vertex Cover in Square Lattice Graphs

Length of one side $N = 2$

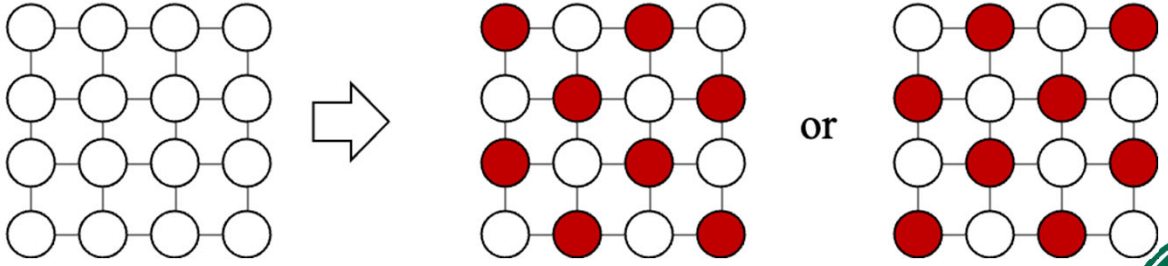
Theoretical Optimum Answer



Length of one side $N = 3$



Length of one side $N = 4$



名古屋大学 NAGOYA UNIVERSITY

Experiment Settings（3/3）

- Minimum Vertex Cover (QUBO formula)
  - Energy function (QUBO variable $x \in \{0,1\}$)[5]

Amplify
- Developed by Fixstars Corp.
- The Python library for Ising machine

$$H = w_a \sum_{(u,v)\in E} (1 - x_u)(1 - x_v) + w_b \sum_{v \in V'} x_v$$

(constraint: $\sum_{(u,v)\in E} (1 - x_u)(1 - x_v)$; cost: $\sum_{v \in V'} x_v$)

- Implementation with **Amplify**

```
15    # コスト関数（w_b で調整）
16    cost_function = sum([sum_poly(q[i]) for i in range(N)])
17
18    # 制約（w_a で調整）
19    constraint_x = sum([greater_equal(q[i][j] + q[i + 1][j], 1) for i in range(N - 1) for j in range(N)])
20    constraint_y = sum([greater_equal(q[i][j] + q[i][j + 1], 1) for i in range(N) for j in range(N - 1)])
21    constraint = constraint_x + constraint_y
```

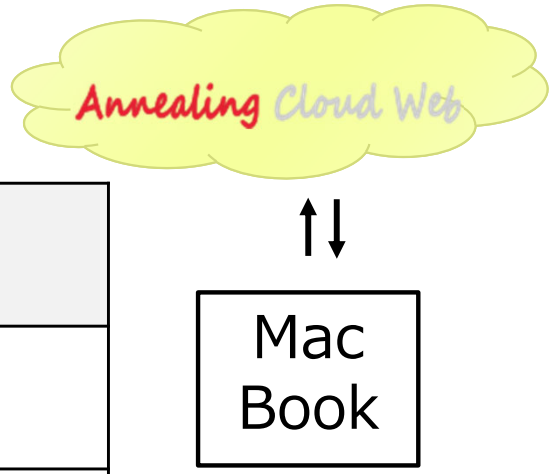
```
52        # 最終的なエネルギー関数
53        energy_function = w_a*constraint + w_b*cost_function
```

[5] https://amplify.fixstars.com/ja/techresources/research/ising-model-formulation/vertex-covering/

Hardware Environment

Table 4. Experiment environment (annealing-base)

Hardware / Software	Details
CMOS Annealing Machine	• **Annealing Cloud Web API v2: GPU version 32bit (float)**
MacBookAir (macOS Big Sur)	• Machine for executing Python (Version 3.8.2) • 1.6GHz Dual Core Intel Core i5 • Memory 8GB
Amplify	• Library for using CMOS annealing machine via Web API • Version 0.5.13



名古屋大学 NAGOYA UNIVERSITY

Result: Annealing-base

$$H = w_a \sum_{(u,v)\in E} (1 - x_u)(1 - x_v) + w_b \sum_{v \in V'} x_v$$

constraint cost

Optimal Answer Rate [%]

w_a : Constraint Weight

w_b : Cost Weight

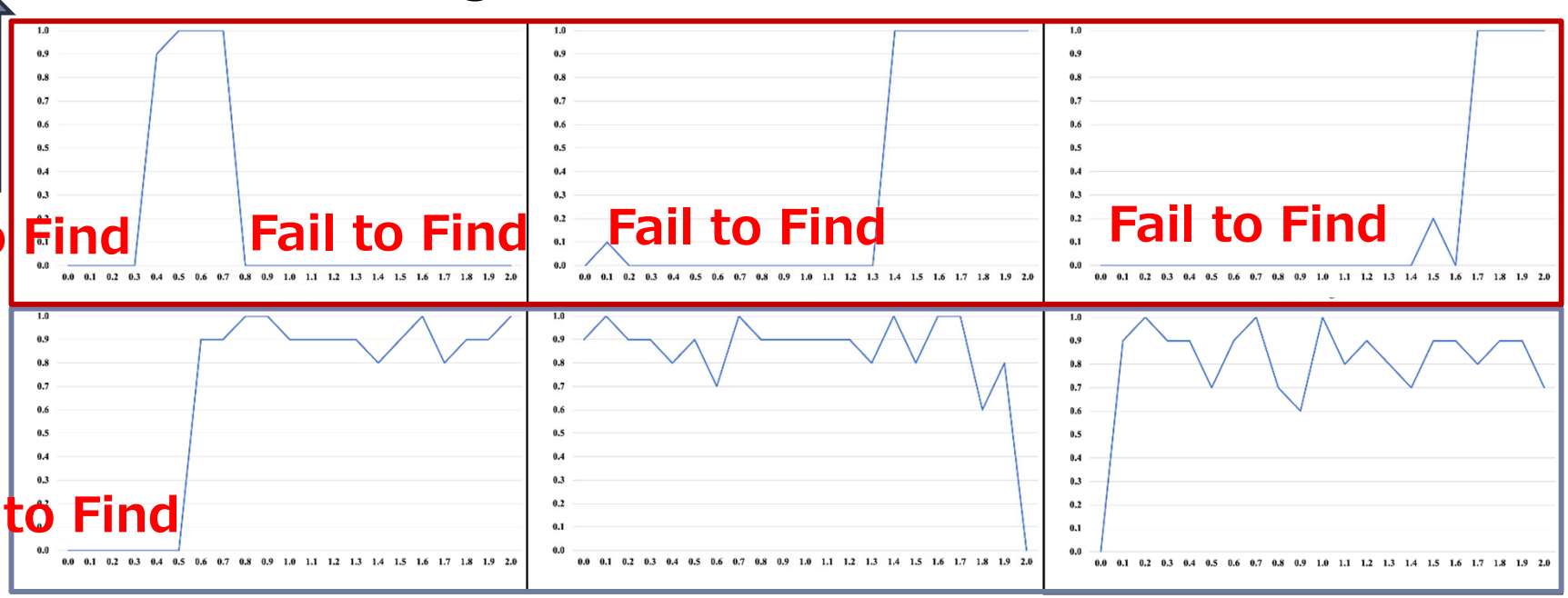
chain_strength

$N = 7$

Fail to Find Fail to Find Fail to Find Fail to Find

$N = 8$

Fail to Find



Weight [0:2] Weight [0:2] Weight [0:2]

Comparison of problem sizes N=7 and N=8

名古屋大学 NAGOYA UNIVERSITY

Auto-tuning Effect by Optuna[3]

[3] Preferred Networks, Inc. : An open source hyperparameter optimization framework to automate hyperparameter search. https://optuna.org/

Hundreds of trials are automated!

The optimal solution ratio [%] with default values

	w_a	w_b	$chain\_strength$	Optimal solution ratio
$N = 3$	1.00	1.00	1.00	**5%**
$N = 5$	1.00	1.00	1.00	**13%**

Default Settings

Table 8. The optimal solution ratio [%] with auto-tuned settings

	w_a	w_b	$chain\_strength$	Optimal solution ratio
$N = 3$	9.76	0.01	6.42	**86%**
$N = 5$	6.26	0.007	1.27	**18%**

Auto-tuned Settings

Outline

1. Background: Performance Portability and Auto-tuning
2. Case1: CMOS Annealing Machine
3. Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK
4. Conclusion

Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK

Collaborators: **Prof. Shuji Morisaki** (Nagoya University)

Mr. Hiroto Kashimura (M1, Nagoya University)

名古屋大学
NAGOYA UNIVERSITY

Conventional Method for Priority Problem of Software Testing

- Many existing methods have assumptions about testing software and systems.
  1. Targets of the Test
     - It is assumed that the test is to verify a specific part of the source code or a specific function.
     - Prioritize by selecting tests that have the greatest coverage of your code.
  2. Timing of the Test
     - The test is set to be shorten the test execution time. In addition, the test assumes that the test will be executed every time the code is changed.

We have almost no experience of numerical library for the software testing.

STCollection [4]

- **STCollection**: A test program of LAPACK routines for symmetric tridiagonal eigensolvers.
- Verifying calculated results with analytical answers for eigenvalue problems.
- Setting multiple tests with respect to difficulties of eigenvalue problem.
  - Different distributions for theoretical eigenvalues are utilized to check the “computational” bug.

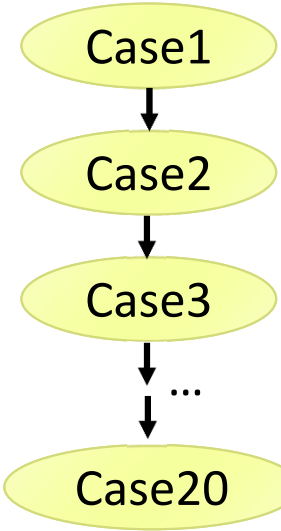
[4] Osni Marques : GitHub - oamarques/STCollection: Collection of symmetric tridiagonal and bidiagonal matrices, GitHub, 〈https://github.com/oamarques/STCollection〉

Test Set: *test_easy* in STCollection

- The default test order is as follows:

- Case1-4:Eigenvalue distribution is followed by Type 3 and 4.
- Case5-10: (1, 2, 1)-matrix.
- Case11-16: Wilkinson matrix
- Case17: Glued matrix
  - T_1 : Eigenvalue distribution is followed by Type 1.
  - T_2 : (1, 2, 1)-matrix
  - T_3 : Eigenvalue distribution is followed by Type 3.
  - Glue coefficients are: $\gamma_1 = 0.001, \gamma_2 = 0.002$
- Case18: Same as Case 17, but glue coefficients are $\gamma_1 = -0.001, \gamma_2 = -0.002$
- Case19-20: Read a user specified file

Type	Test Matrices
1	$\lambda_1 = 1, \lambda_1 = \frac{1}{k}, i = 2, 3, \dots n$
2	$\lambda_i = 1\ , i = 1, 2, \dots n-1, \lambda_n = \frac{1}{k}$
3	$\lambda_i = k^{-(\frac{i-1}{n-1})}\ , i = 1, 2, \dots n$
4	$\lambda_i = 1 - \left(\frac{i-1}{n-1}\right)\left(1 - \frac{1}{k}\right), i = 1, 2, \dots n$
5	n randomized eigenvalues within $\left(\frac{1}{k}, 1\right)$.The distribution is uniform.
6	N randomized eigenvalues by user specified range.
7	$\lambda_i = ulp \times i, i = 1, 2, \dots n-1, \lambda_n = 1$
8	$\lambda_1 = ulp, \lambda_i = 1 + \sqrt{ulp} \times i,$ $i = 2, 3, \dots n-1, \lambda_n = 2$
9	$\lambda_1 = 1, \lambda_i = \lambda_{i-1} + 100 \times ulp, i = 2, 3, \dots n$



Test Order

Experimental Environment

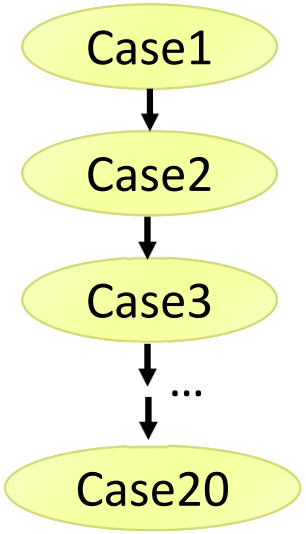
Supercomputer “Flow” Cloud Sub-System at Information Technology Center, Nagoya University

Machine Type		HPE ProLiant DL 560
Node	CPU	Intel Xeon Gold 6230 × 4
	Main Memory	DDR4 384 GiB (Memory bandwidth: 563.136 GB/s)
Maximum number of Nodes		100
Total Theoretical FLOPS		537.6 TFLOPS
Total Memory Amount		37.5 TiB

Source: Supercomputer “Flow”, https://icts.nagoya-u.ac.jp/ja/sc/overview.html

Experimental Setting

- We implemented intentionally injected "Bug" in BLAS routine (*dgemm*).
  → A Frequently called routine in LAPACK.
- A scalar value of "alpha" in the *dgemm* routine is forced to change from 1.0 to 0.01.
  → Artificial Bug situation in the *dgemm* routine.



Default Test Order

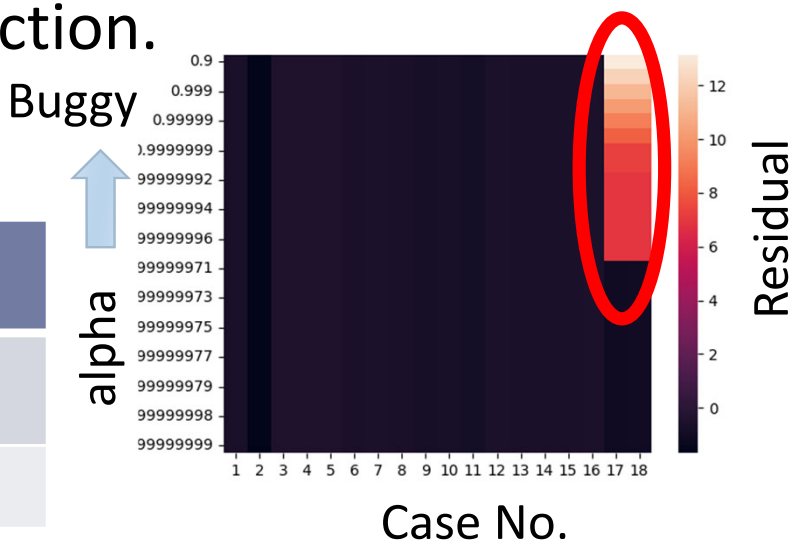
名古屋大学
NAGOYA UNIVERSITY

Preliminary Result

- The bug is only found in the Case 17 in view point of residual and orthogonality in the test routine of STCollection.

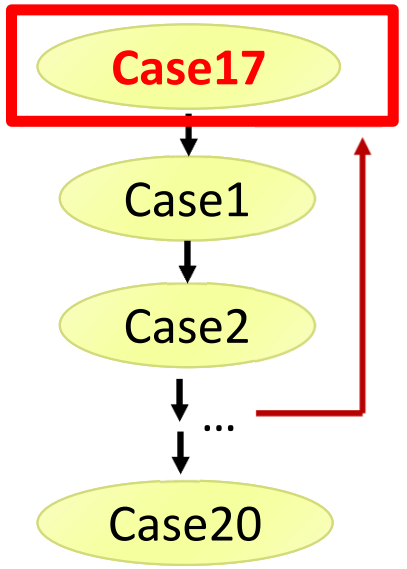
Test Results in Case 17

alpha	Execution Time [Sec.]	Residual	Orthogonality
1.0 (normal)	9.12×10^{1}	3.34×10^{-3}	1.80×10^{-2}
0.01	1.27	2.42×10^{11}	4.84×10^{11}



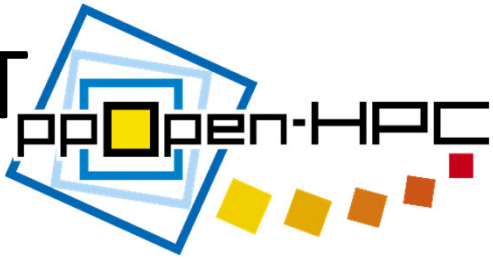
Test Time Speed-up: 9.17x

Test Re-Order



It has a potential that execution time of the testing can be reduced if the order of Case 17 is set to the first test sequence.

Test order optimization by ppOpen-AT



Install-Time

Specify target environment
(CPU, Compiler, number of Threads and MPI Processes, etc.)

Set target routines and problem size by end-user

Set test configuration and execute the test

Re-use the order in next test

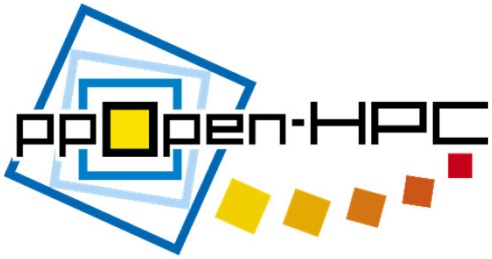
Execute auto-tuner:

- Obtain error-threshold for all routines
- Time measurement for each routine
- Store the best order information

Store the test order information

Using the best test order without AT

Extension of ppOpen-AT Directives : Order optimization for select directive



In code of the test routine:

```
!oat$ install select region start
!oat$ according min ( $time ) condition ( err > th )
!oat$ according opt (order)

!oat$ select sub region  start
        call TestCase1()
!oat$ select sub region end
!oat$ select sub region start
        call TestCase2()
!oat$ select sub region end
              …
!oat$ select sub region start
        call TestCaseN()
!oat$ select sub region end

!oat$ install select region end
```

Conventional directive:
Select the best region for the following sub regions.

Conventional directive:
Specify optimization condition.

- Minimalize execution time.
- Over a threshold value to mesured error.

Extended directive:
Order optimization for the following sub regions.

Outline

1. Background: Performance Portability and Auto-tuning
2. Case1: CMOS Annealing Machine
3. Case2: Adaptation of Software Engineering: Optimization of Test Sequences for LAPACK
4. Conclusion

Conclusion

- Auto-tuning (AT) technology has a potential to increase HPC software productivity.
- Sustainability
  - Using directives for AT to conventional programing languages, Fortran, C, etc., keeps sustainability in performance.
  - Testing (Tuning) is automated by the AT framework.
  - Tuning costs can be reduced, such as computational efficiency, energy, etc.
- But still remain as traditional problems…
  - Maintainability, Reusability, Portability, Reproducibility, etc…