「情報システム学特別講義３」
ソフトウェア自動チューニング

東京大学情報基盤センター准教授　片桐孝洋

2014年7月22日（火）14:40－16:10

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

講義日程
（情報システム学特別講義３）

レポートおよびコンテスト課題
（締切：
2014年8月11日（月）24時　厳守

1. ~~4月8日：ガイダンス~~
2. ~~4月15日~~
   - ~~プログラム高速化の基礎（その1）~~
3. ~~4月22日~~
   - ~~プログラム高速化の基礎（その2）~~
4. ~~5月13日~~
   - ~~MPIの基礎~~
5. ~~5月20日~~
   - ~~OpenMPの基礎~~
6. ~~5月27日~~
   - ~~Hybrid並列化技法（MPIとOpenMPの応用編）~~
7. ~~6月3日~~
   - ~~プログラム高速化の応用~~
8. ~~6月10日~~
   - ~~行列ーベクトル積の並列化~~
9. ~~6月17日~~
   - ~~べき乗法の並列化~~
10. ~~6月24日~~
    - ~~行列ー行列積の並列化~~
11. ~~7月8日~~
    - ~~LU分解の並列化~~
12. ~~7月15日~~
    - ~~非同期通信~~
    - ~~疎行列反復解法の並列化~~
13. 7月22日
    - ソフトウェア自動チューニング
14. 8月5日（補講日）
    - エクサフロップスコンピューティングに向けて

講義の流れ

1. 背景
2. ソフトウエア自動チューニングとは
3. FIBER方式
4. 自動チューニング記述言語*ABCLibScript*
5. *ppOpen-HPC* プロジェクトと *ppOpen-AT*
6. レポート課題

背景

HPCソフトウェアの生産性の問題

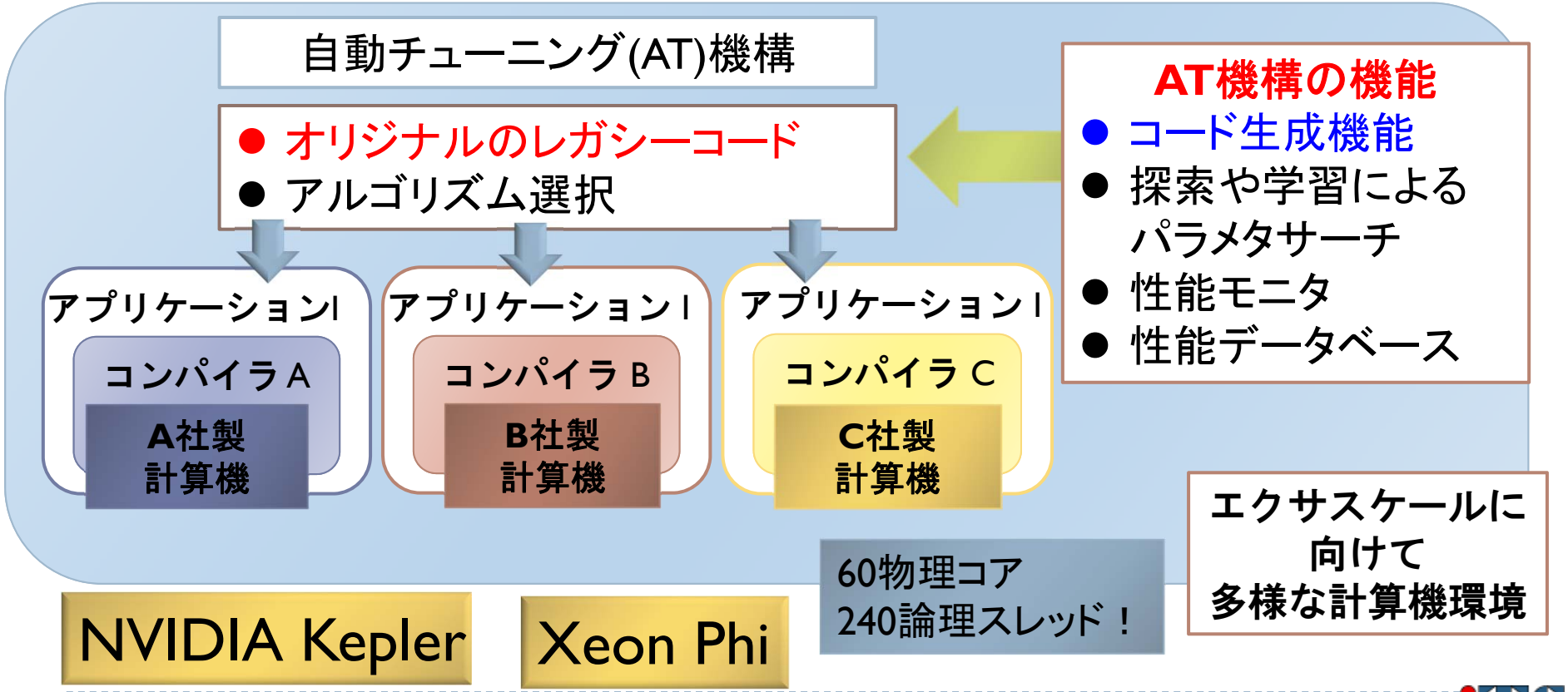
- 高性能なソフトウェアを安価に開発するには？
  - 従来、HPCなソフトウェア開発は研究色が強く、コストを度外視して開発されることが多い
    - 高性能重視、かつ、開発コストは考慮しない
  - 大量生産（開発）が出来ないソフトウェアと思われている
- 近年、企業利用など、多岐にわたる分野で、HPCソフトウェアが必要となっている。開発コストを度外視できない。
  - 生産効率を上げる工夫が必要

HPCソフトウェアの生産性の問題

- 従来の「ソフトウェア工学」とは異なり、HPC分野は、高い実行性能の要求がきわめて強い
  - というか、高い実行性能は必須となる「要求仕様」
  - 単に、「バグなく開発できる」ソフトウェアでは使い物にならない
    ⇒新しい「ソフトウェア工学体系」が必要

「性能可搬性」の実現

- **複数計算機で有効となる最適化が提供できるパラダイム**
  （HPCI 技術ロードマップ白書、2012 年3 月）
  - **同一プログラムで計算機が変わっても高性能を維持**



計算機環境の変化

- マルチコア・アーキテクチャの浸透
  - 非均質メモリアクセス（ccNUMA）
  - 多階層化されたキャッシュ構造
  - チップ内のコア数の増大
- 並列実行モデルの変化
  - ピュアMPI　VS.　ハイブリッドMPI
- コンパイラ最適化では手に負えない
  - 手動チューニングはコスト高
  - コスト削減のため、実用的観点から、性能自動チューニング技術へ期待

数値アプリケーション開発における生産性の問題

- なぜ、コストが高いか
  1. 探索空間爆発の問題
     - ソフトウエア開発コストが爆発的増加
  2. チューニングは科学でなく、職人芸
     - 職人の賃金は高い＆引き継ぎが難しい

1. 探索空間爆発の問題
   - 多数のアルゴリズムパラメタ
     - 前処理方式、やり直し間隔、ブロック幅、…
   - 複雑化された計算機アーキテクチャ
     - マルチコア、非対称メモリアクセス、…
2. 賃金コスト増加の問題
   - 複雑な高性能を達成するための実装
     - 職人のみができる
   - コンパイラがうまく働けば良いが、複雑化した計算機アーキテクチャ上ではより困難に….

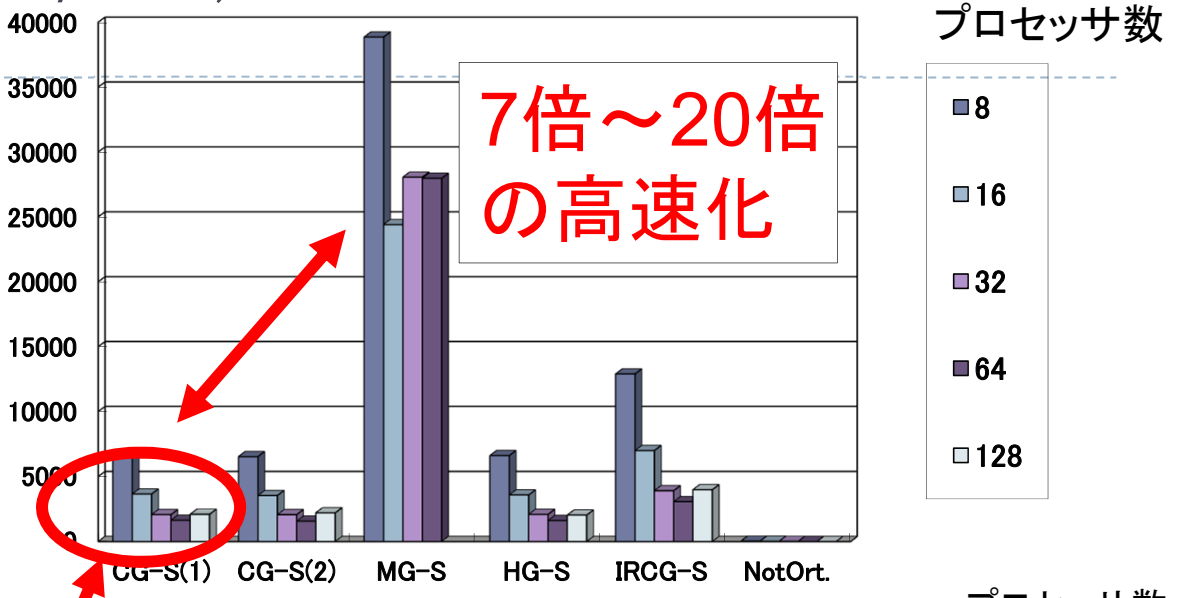
自動チューニングにおける要求技術

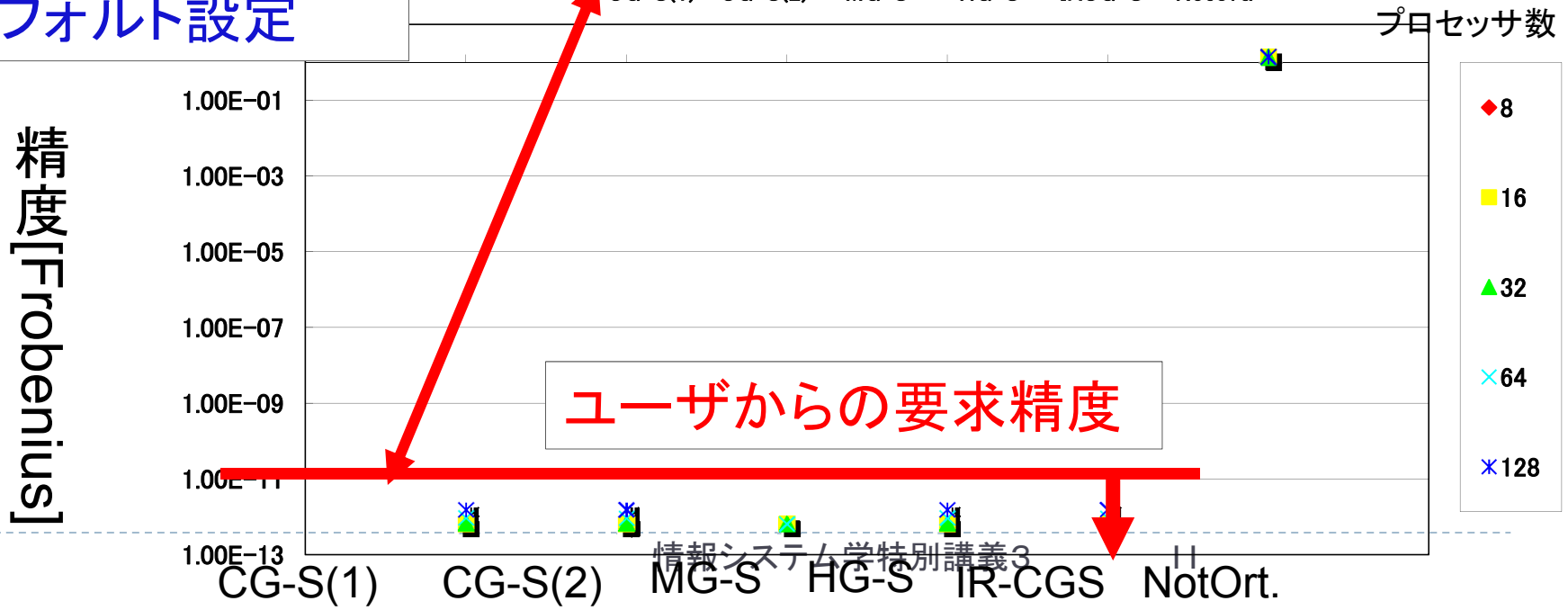
- どのようなAT機能が必要か？
  - コンパイラでできそうなこと
    - アンローリング
    - キャッシュサイズ調整
  - コンパイラでできないこと
    - 数値計算アルゴリズム選択
      - 数値計算精度を保証した上でのアルゴリズム選択
    - 実行時ユーザ知識情報を活用した最適化
  - ミドルウェアレベルの最適化
    - MPI実装方式選択 、など
- 自動チューニングのタイミング
  - インストール時から＜実行時＞へ
    - ユーザプログラムにおいて、実行時の知識抽出と利用

実例：実行時アルゴリズム選択
(HITACHI SR8000/MPP)

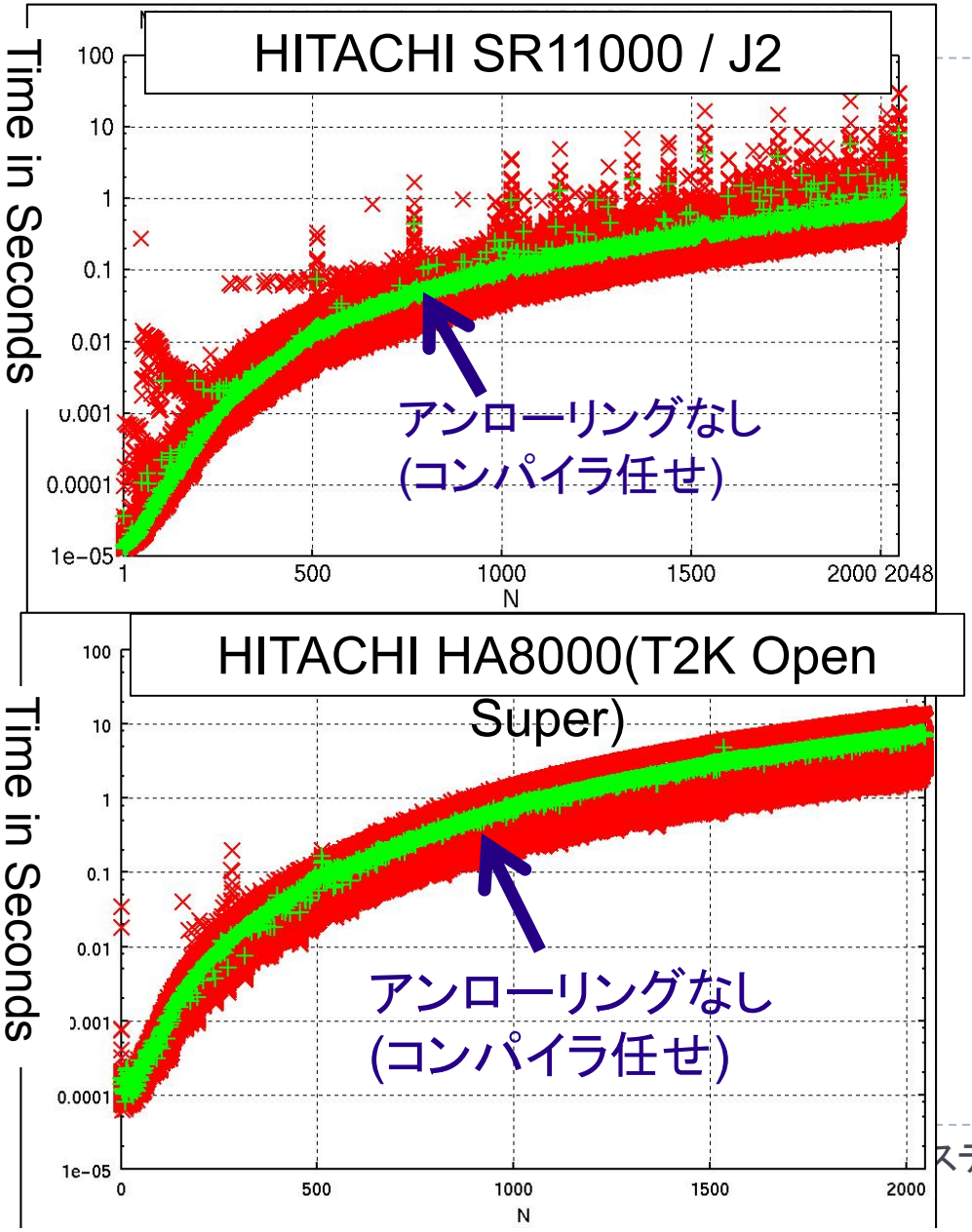
フランク行列から三重対角化した行列：
逆反復法中の再直交化時間

MG-S(修正G-S)：
多くの場合のデフォルト設定





実例: 先進アーキテクチャによる不安定性



- **密行列の行列ー行列積**

BLASを用いていない単純コード

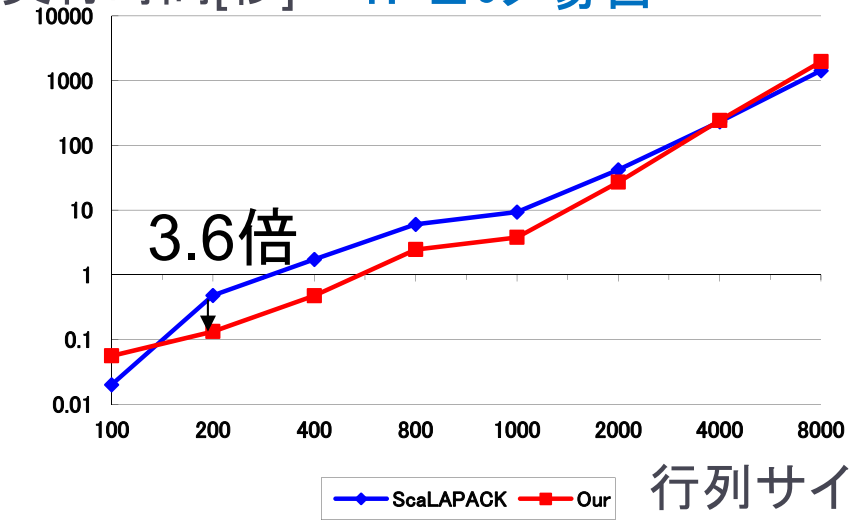
- 3重ループ (i,j,k)，アンローリング1段～4段
- **次元Nに関し 4*4*4=64種類 の実装**
- 1 から2048次元まで1刻みのデータ.
- コンパイラ HITACHI Optimized Fortran90. オプション: -Oss
- **自動並列化（ノード内）**
- 計算機構成:
  - HITACHI SR11000/J2
  - HA8000 (T2K Open Supercomputer (Todai Combined Cluster))
  - Installed in Information Technology Center, The University of Tokyo.
  - 16コア／ノード.

- ●コンパイラ任せは遅い
- ●常時＜特定の実装＞が速くない
- ●10倍ぐらい実行時間がぶれる（実行時間の＜不安定性＞）
- ●場合により100倍も遅い！

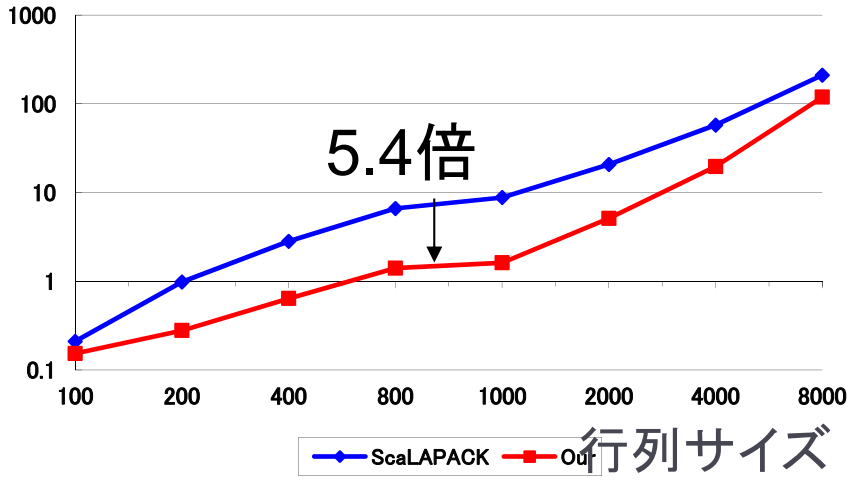
実例：超並列アルゴリズムの構築と適応的選択

（日立SR2201、Householder三重対角化）

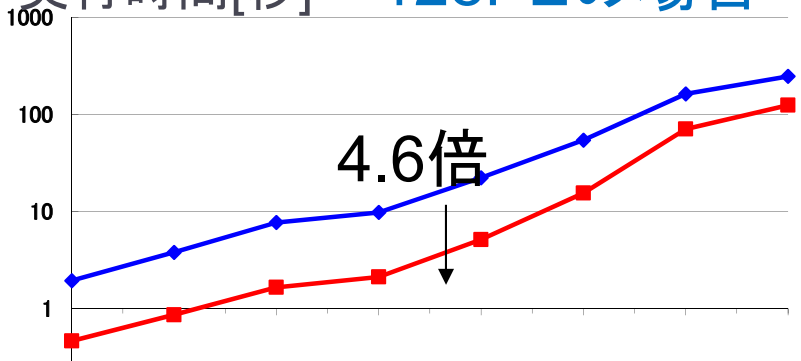
実行時間[秒]　4PEの場合



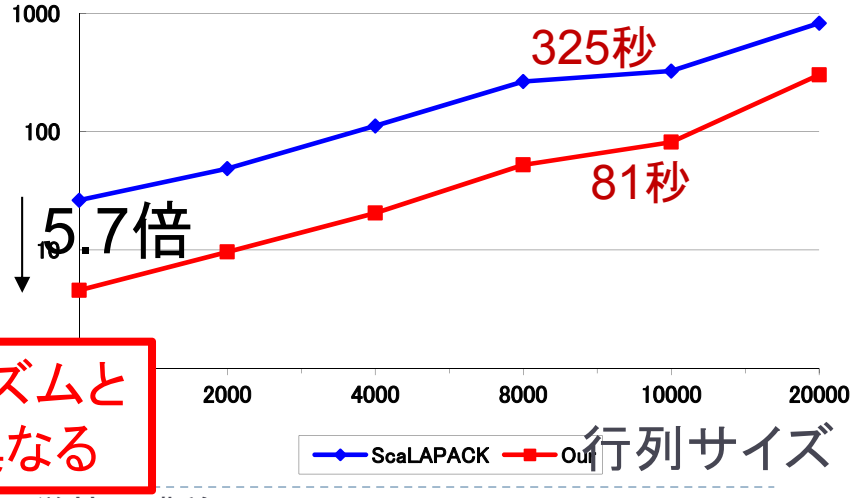
実行時間[秒]　64PEの場合



実行時間[秒]　128PEの場合



実行時間[秒]　512PEの場合



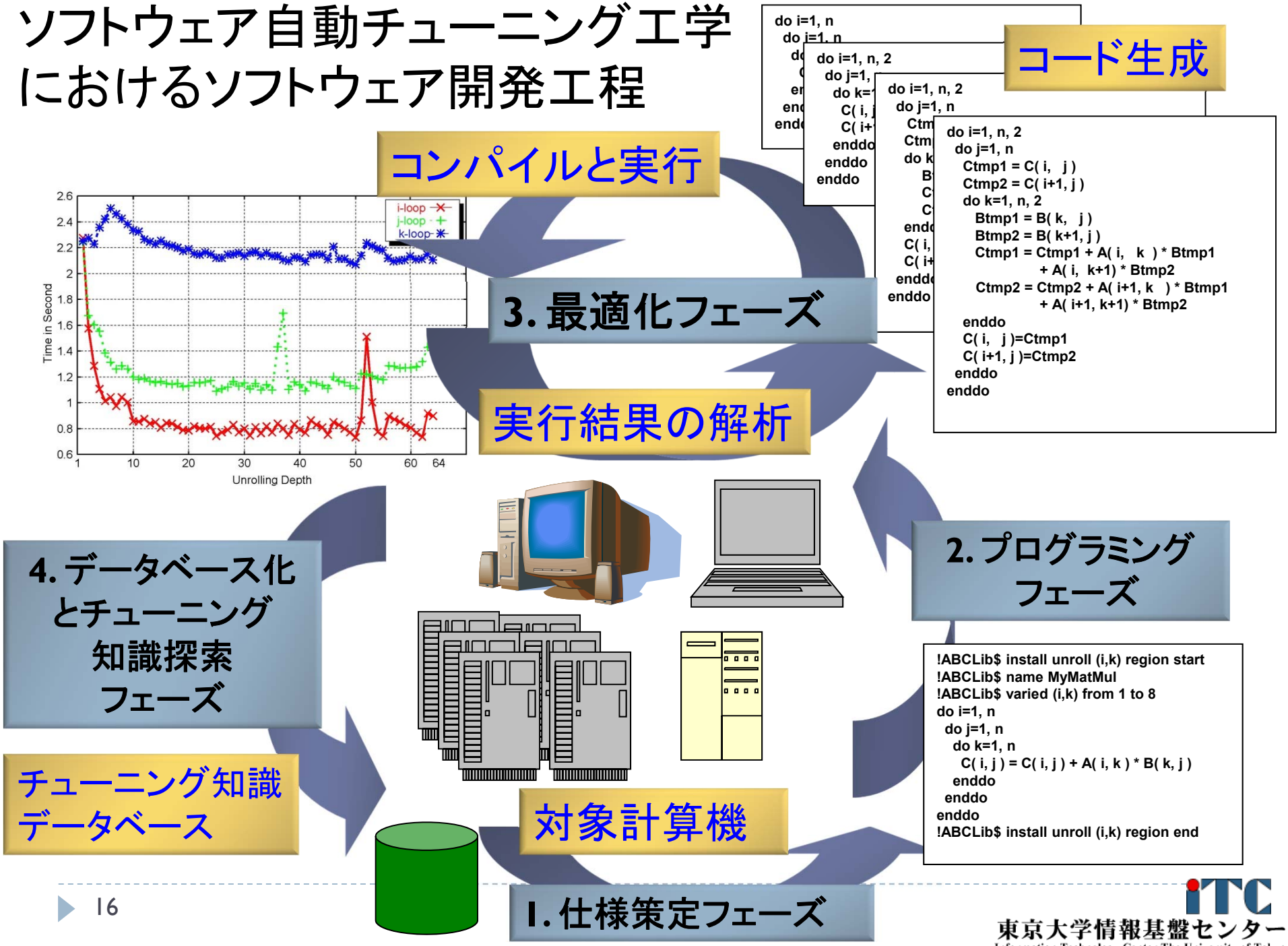
低並列・大規模問題に向く（従来の）アルゴリズムと
超並列・小規模問題に向くアルゴリズムは異なる

ソフトウェア自動チューニングとは

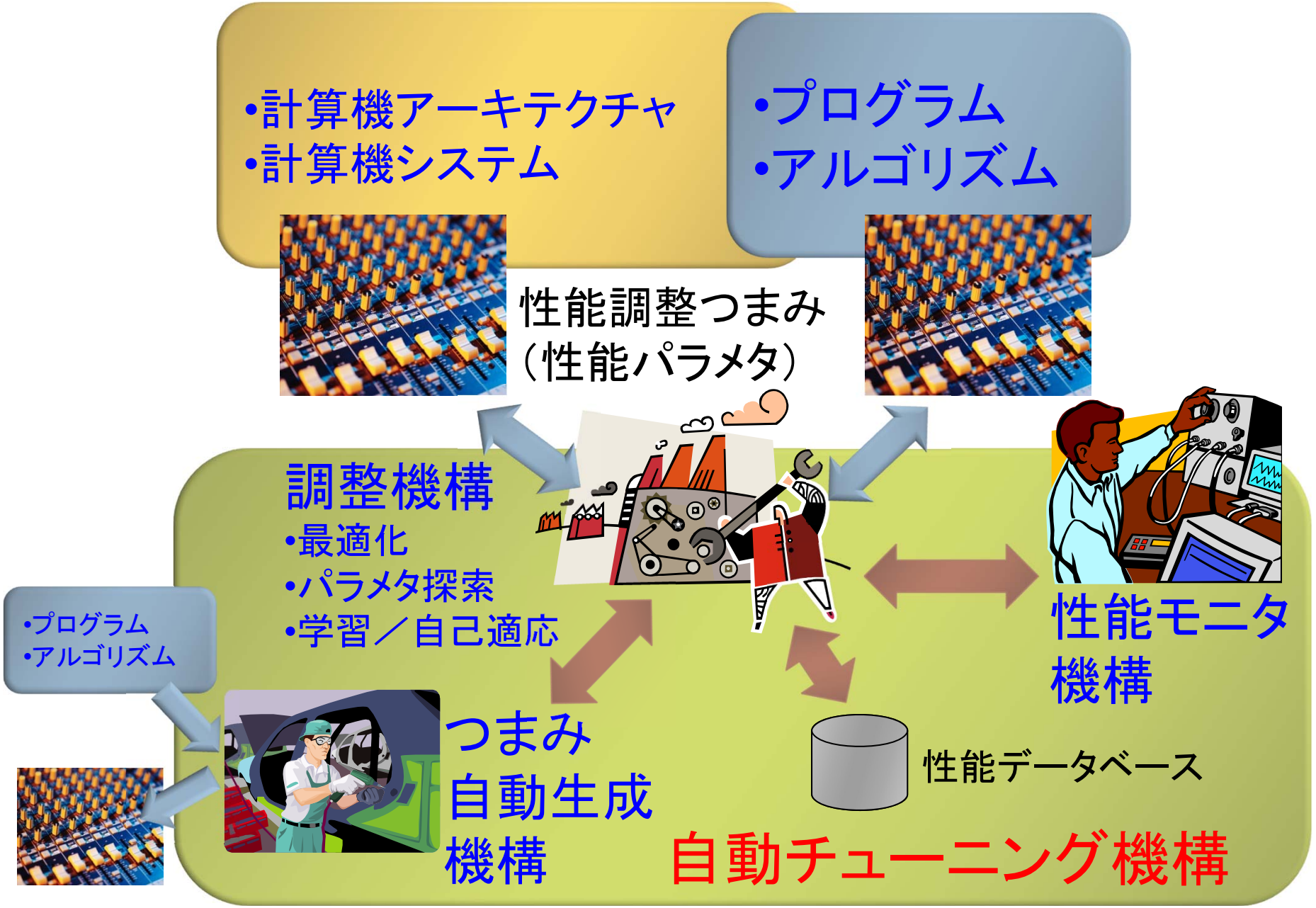
ソフトウェア自動チューニングとは

- 自動チューニングソフトウェア工学
  - 「性能可搬性」の実現を主な問題設定とし、安価で高機能なHPCソフトウェアを開発するための、工学体系を研究する分野
  - ソフトウェア開発の方法論
    (Software Development "Methodology")
  - 以下の全てにおいて、「実行性能」を考慮した上で、ソフトウェア開発のコストを削減することを研究する分野
    - 設計(要求仕様策定)、プログラミング、チューニング、テスト(検証)、配布、保守・管理
  - 自動チューニングソフトウェア工学分野での研究開発活動をソフトウェア自動チューニングという
- 「実行性能」は、多様な尺度があり、開発ソフトウェアごとに定まる
  - 実行時間、メモリ量、演算精度、電力量、その他

ソフトウェア自動チューニング工学におけるソフトウェア開発工程

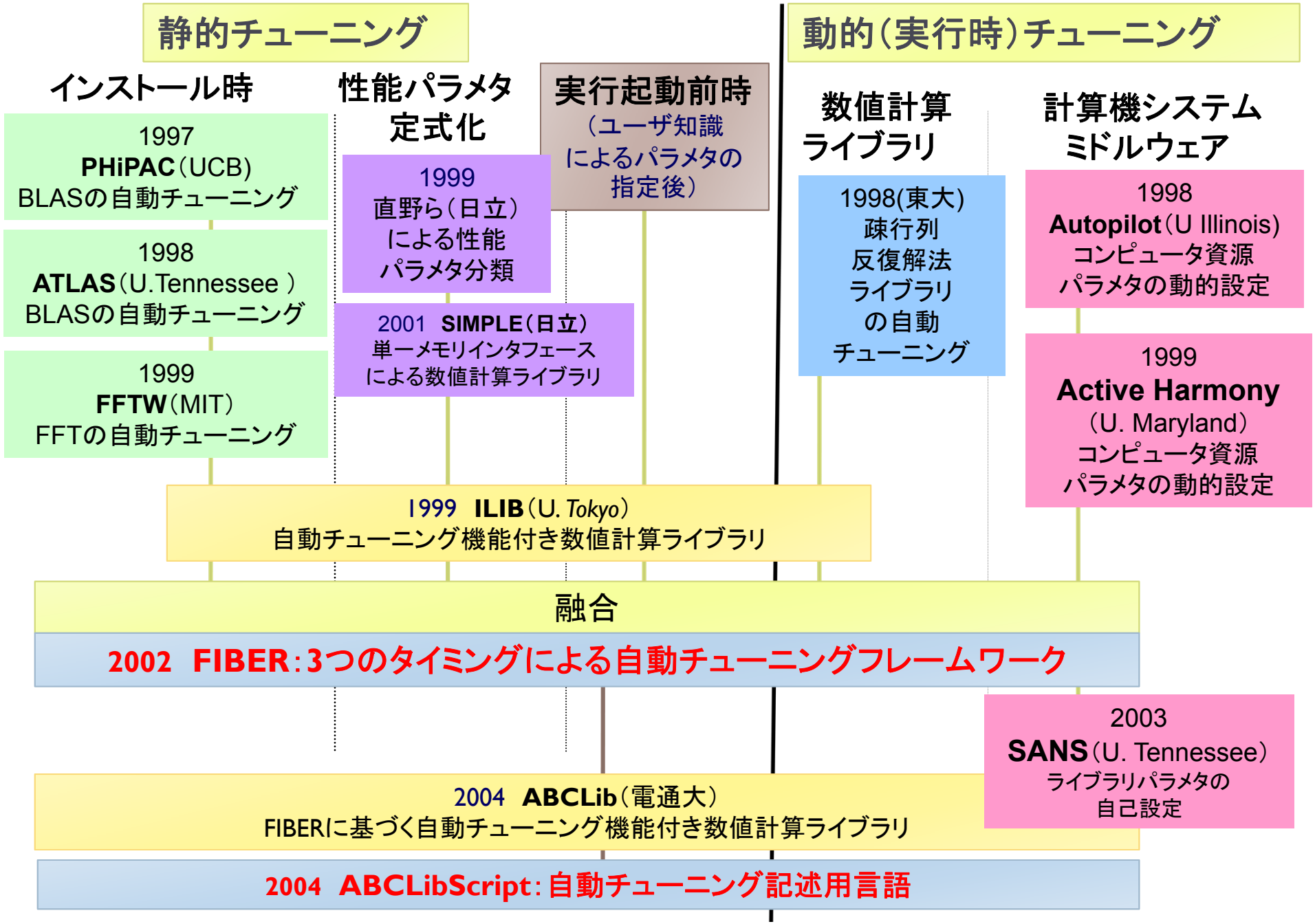


自動チューニング機構とは



自動チューニング技術の鳥瞰図

自動チューニング研究の分類（~2004）



自動チューニングソフトウエア工学達成のために（１／２）

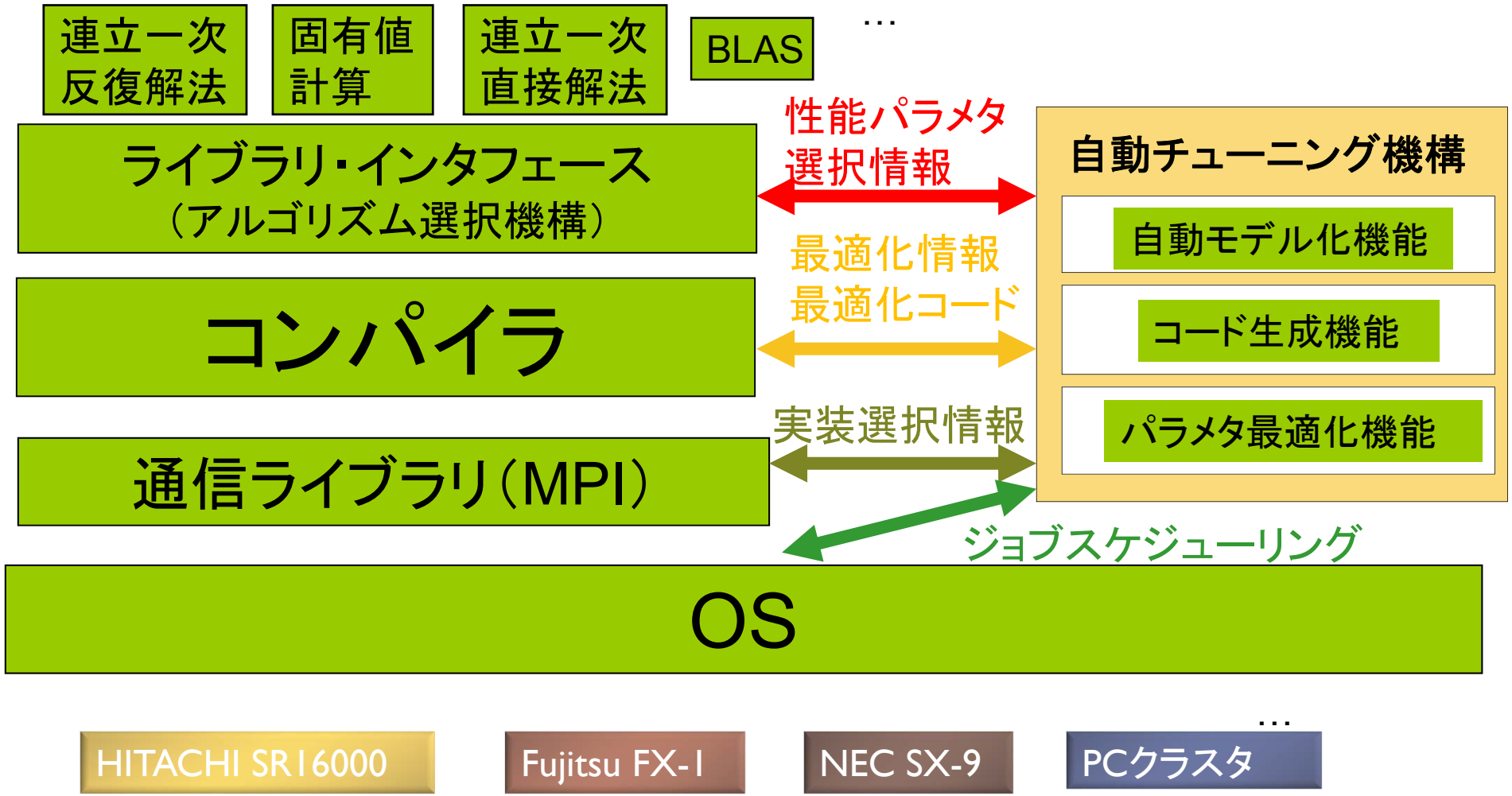
- チューニング方法論の確立
  - チューニングする対象
    - ソフトウエア単位、関数単位、ループ単位、中間言語・機械語単位
  - チューニングするタイミング
    - インストール時、実行時、その他
  - チューニング作業の工程定義
    1. 計算機アーキテクチャ性能の検討
    2. システムソフトウエア性能の検討
    3. コーディング手法の検討
    4. コーディング作業
    5. 実測
    6. 実測性能の解析・検証
    7. 1などに戻る

自動チューニングソフトウエア工学達成のために（２／２）

1. ソフトウエア・アーキテクチャの確立
   - 自動チューニング機構が考慮されていること
2. チューニングのための計算機言語と言語処理系があること
3. 「実行→測定→解析→ソース変更→実行」というチューニング作業のサイクルが、できるだけ自動化されていること
   - 自動チューニングツールの提供
   - 自動解析手法（知識探索手法）の確立
   - データベース化のための方式の確立

自動チューニングソフトウェア工学に基づくソフトウェア開発事例

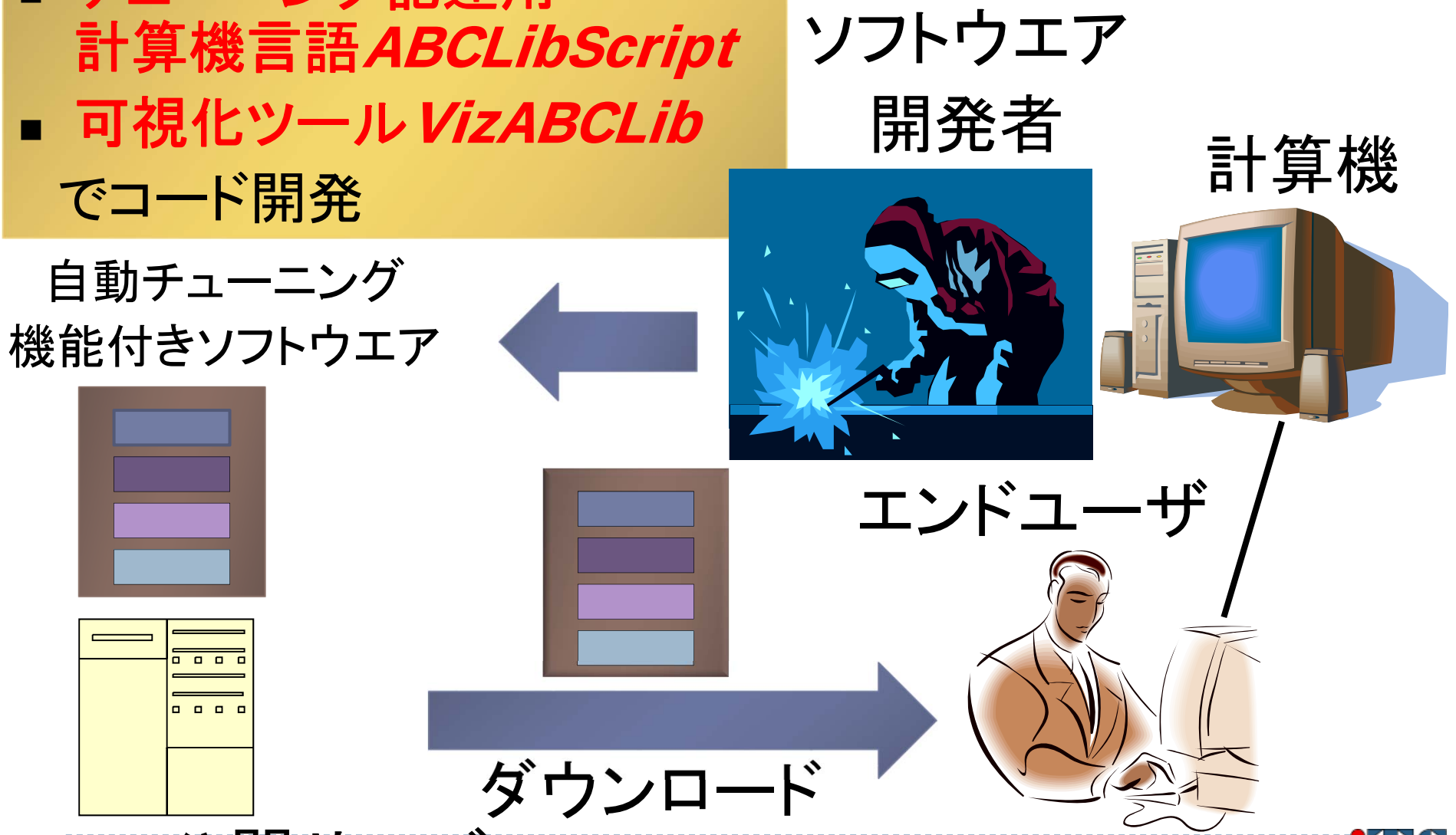
構想
ミドルウエアとしての自動チューニング機構の確立



FIBERフレームワークの概要

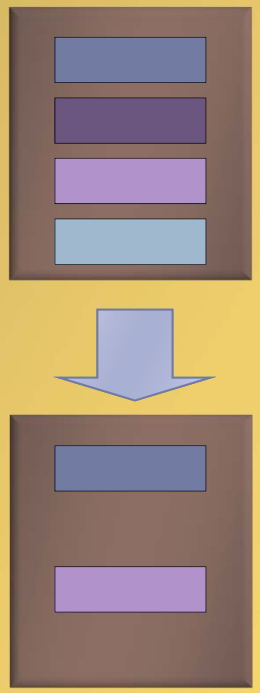
- チューニング記述用計算機言語*ABCLibScript*
- 可視化ツール*VizABCLib*

でコード開発



FIBERフレームワークにおける自動チューニング

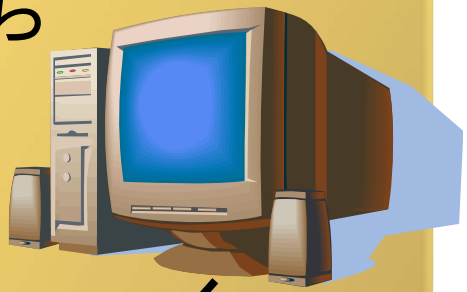
自動チューニング
機能付きソフトウエア



1. **計算機システム性能**
2. **ユーザが知る情報**
3. **実行時情報**

を取得し、実行しながら
自動チューニング

計算機



自動チューニングのタイミング

1. **インストール時**
2. **実行起動前時**
3. **実行時**

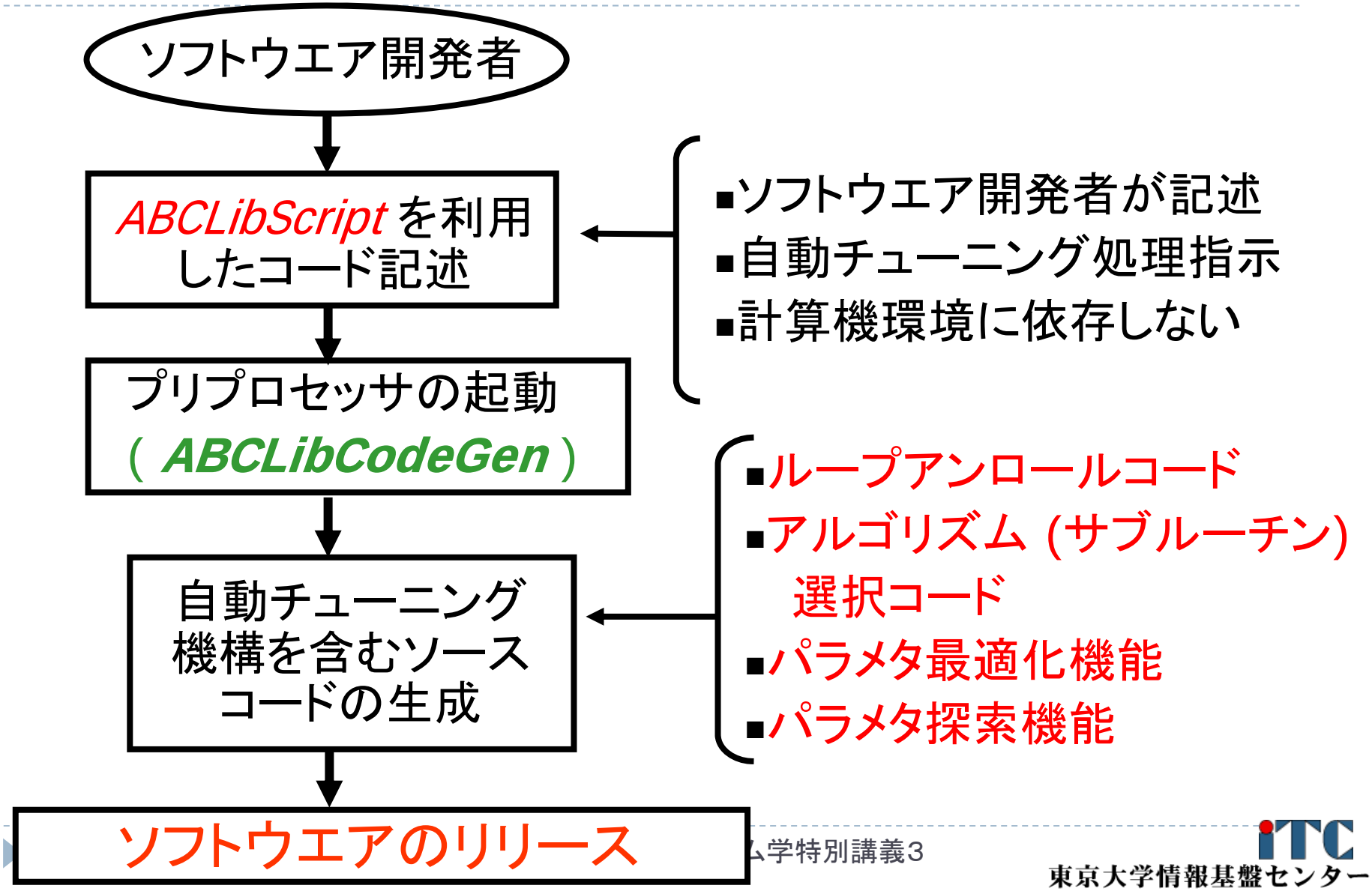
エンドユーザ

自動チューニング記述言語 ABClibscript

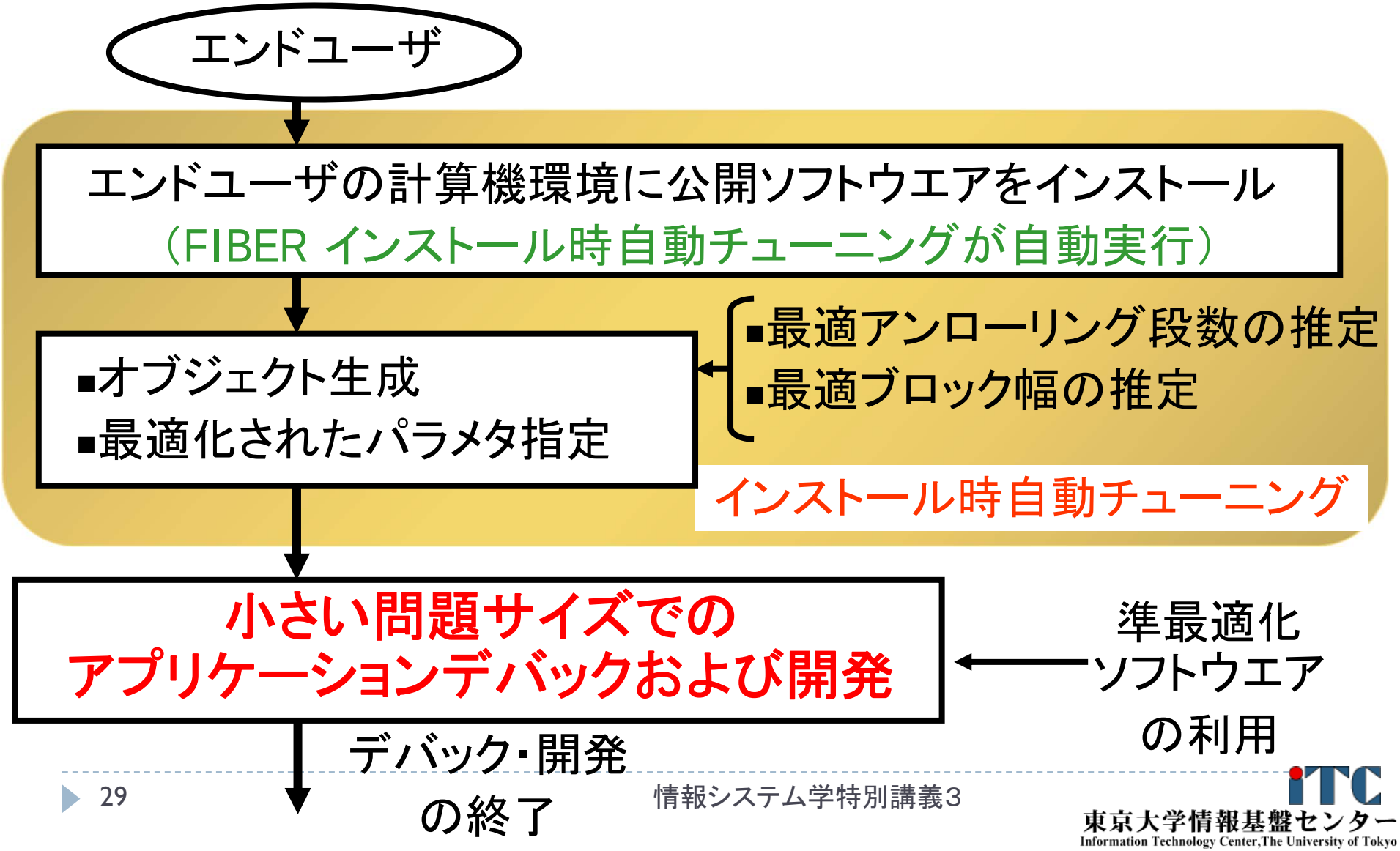
ABCLibScript

- ソフトウェア自動チューニング工学におけるソフトウェア開発工程のうち、チューニング工程における工数削減（開発のための最小作業単位、コストに直結する一般尺度）を目指し開発されている計算機言語
- 自動チューニングのタイミングをFIBERフレームワークにより規定
  - インストール時、実行起動前時、実行時
  - 実行起動前時は特許申請、承認済み
- 工数削減に寄与する事項
  1. チューニングのための実行時間測定
  2. チューニングのための結果の検証
  3. チューニングのためのプログラミング
  4. 職人の技能の違いの吸収（従来手法をディテクティブで書くだけで実現）
  5. 技術の引継ぎ（ディレクティブをみると有効なチューニング手法がわかる）

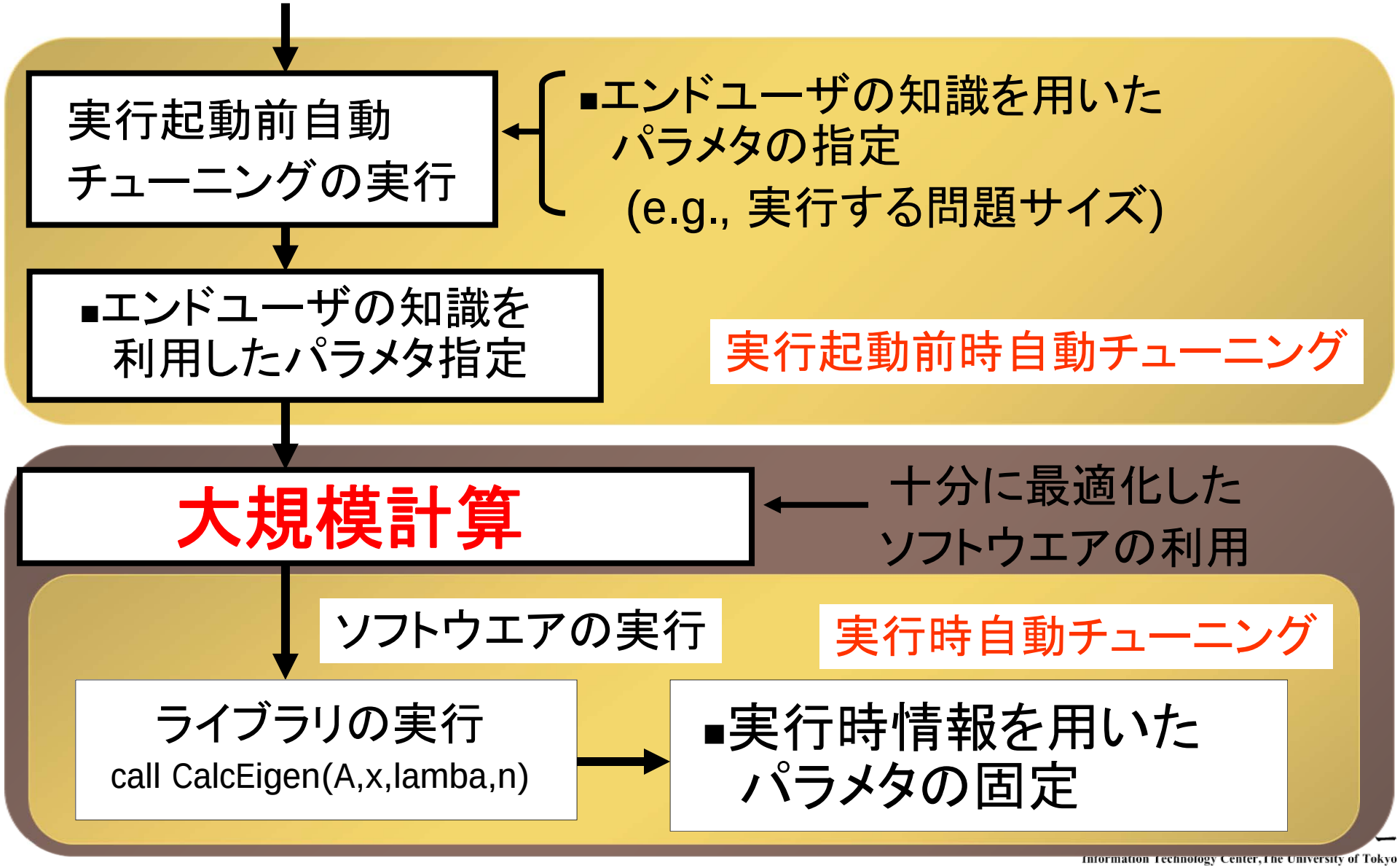
ソフトウエア開発者における ＦＩＢＥＲ方式利用のシナリオ



エンドユーザにおける ＦＩＢＥＲ方式利用のシナリオ (Part 1)



エンドユーザによる ＦＩＢＥＲ方式利用のシナリオ (Part 2)



ABCLibScript設計方針

1. 容易に自動チューニング指定できる
   - 数値計算処理に機能限定:
     - アンローリング指定子(*unroll*)
     - 変動パラメタ指定子(*variable*)
     - アルゴリズム選択指定子(*select*)
2. もとのプログラムの実行を阻害しない
   - ディレクティブ形式でプログラム中に記述
3. 高い可読性コードを自動生成
   - Fortran90 + MPI-1 のコードを自動生成
4. 自動チューニングの見通しがよい
   - 2種の内部パラメタ(*BP*, PP)概念の導入

プリプロセッサの処理

```
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 8
do i=1, N
   do j=1, N
      da1 =  A(i, j)
      do k=1, N
         dc = C(k, j)
         da1 = da1 + B(i, k) * dc
      enddo
      A(i, j) = da1
enddo  enddo
!ABCLib$ install unroll (i) region end
```

自動生成

パラメタ最適化コンポーネント

■パラメタの最適化
■実測とそれに基づくモデル化

AT領域選択コンポーネント

■ランタイム
■最適化済みパラメタ指定

AT領域ライブラリコンポーネント

■チューニング対象領域のサブルーチン・ライブラリ化

ソフトウエア開発者用ディレクティブ：ループアンローリング指定子

- アンローリング段数：ディレクティブを用いた指定
  - 例： 行列-行列積コード

```
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 8
!ABCLib$ debug (pp)
      do i=1, N
        do j=1, N
          da1 =  A(i, j)
          do k=1, N
            dc = C(k, j)
            da1 = da1 + B(i, k) * dc
          enddo
          A(i, j) = da1
        enddo
      enddo
!ABCLib$ install unroll (i) region end
```

インストール時指定;
アンローリング指定;

アンローリング段数

対象領域
(チューニング
領域)

ソフトウエア開発者用ディレクティブ：ループアンローリング指定子（つづき）

- プリプロセッサ起動後、以下のコードが自動生成

```
if (i_unroll .eq. 1)  then
      Original Code
  endif
if (i_unroll .eq. 2) then      /* i is dividable by 2 */
  im = N/2
     i = 1
     do ii=1, im
       do j=1, N
         da1 =  A(i,  j);  da2 =  A(i+1,j)
         do k=1, N
           dc = C(k, j)
           da1 = da1 + B(i,   k) * dc;  da2 = da2 + B(i+1, k) * dc; enddo
         A(i,  j) = da1; A(i+1,j) = da2
       enddo
       i = i + 2;
     enddo
  endif
  ...
```

コード生成終了後 、アンローリング段数が自動的に性能パラメタとしてシステムに登録される

ソフトウエア開発者用ディレクティブ：アルゴリズム選択指定子

- アルゴリズム選択処理

```
!ABCLib$  static  select region start
!ABCLib$  parameter (in CacheS, in NB, in NPr
!ABCLib$      select sub region start
!ABCLib$      according estimated
!ABCLib$         (2.0d0*CacheS*NB)/(3.0d0*NPrc)
                 対象1（アルゴリズム1）
!ABC-LIB$     select sub region end
!ABC-Lib$     select sub region start
!ABC-Lib$     according estimated
!ABC-Lib$         (4.0d0*ChcheS*dlog(NB))/(2.0d0*NPrc)
                 対象2（アルゴリズム2）
!ABC-LIB$     select sub region end
!ABC-LIB$ static select  region  end
```

実行起動前時;
選択指定子;

コスト定義関数内で使われる変数

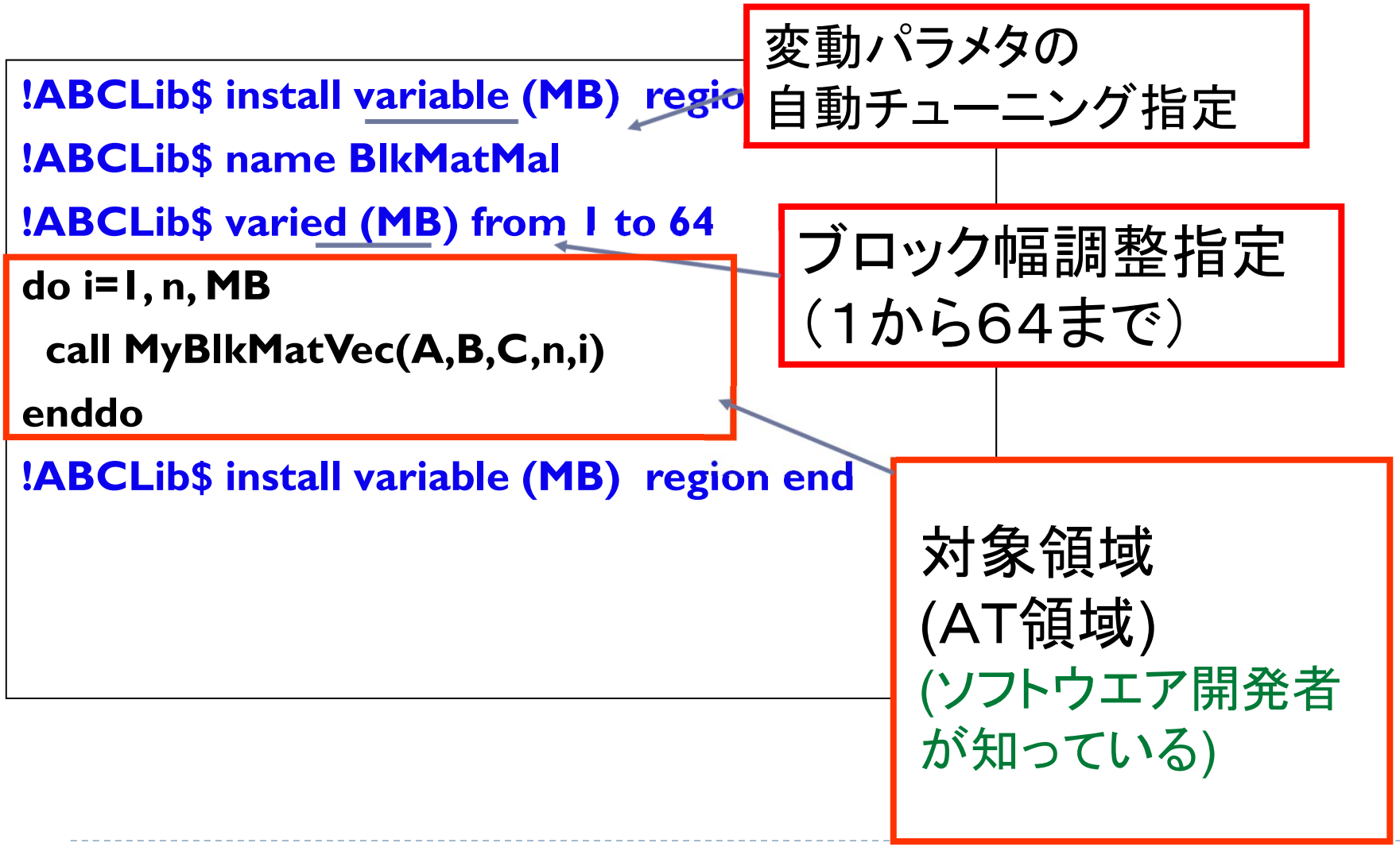
コスト定義関数：
この値をもとに選択がなされる

対象領域1
（チューニング領域1）

対象領域2
（チューニング領域2）

領域1と2の選択が、性能パラメタとしてシステムに自動登録される

ソフトウエア開発者用ディレクティブ：ブロック幅調整



ソフトウエア開発者用ディレクティブ：実行時の反復解法の前処理方式選択

```
!ABCLib$ dynamic select (eps,iter)
!ABCLib$ &  region start
!ABCLib$ name PreCondSelect
!ABCLib$ parameter (in eps, in iter)
!ABCLib$ according min (eps) .and.
!ABCLib$ &  condition (iter<5)
!ABCLib$   select sub region start
        AT領域1（前処理1）
        ...
        eps = ...
!ABCLib$   select sub region end
!ABCLib$   select sub region start
        AT領域2（前処理2）
        ...
        eps = ...
!ABCLib$   select sub region end
!ABCLib$ dynamic select (eps,iter)
!ABCLib$ & region end
```

実行時自動チューニング、
アルゴリズム選択処理の指定

コスト定義関数で利用する
入力変数の指定

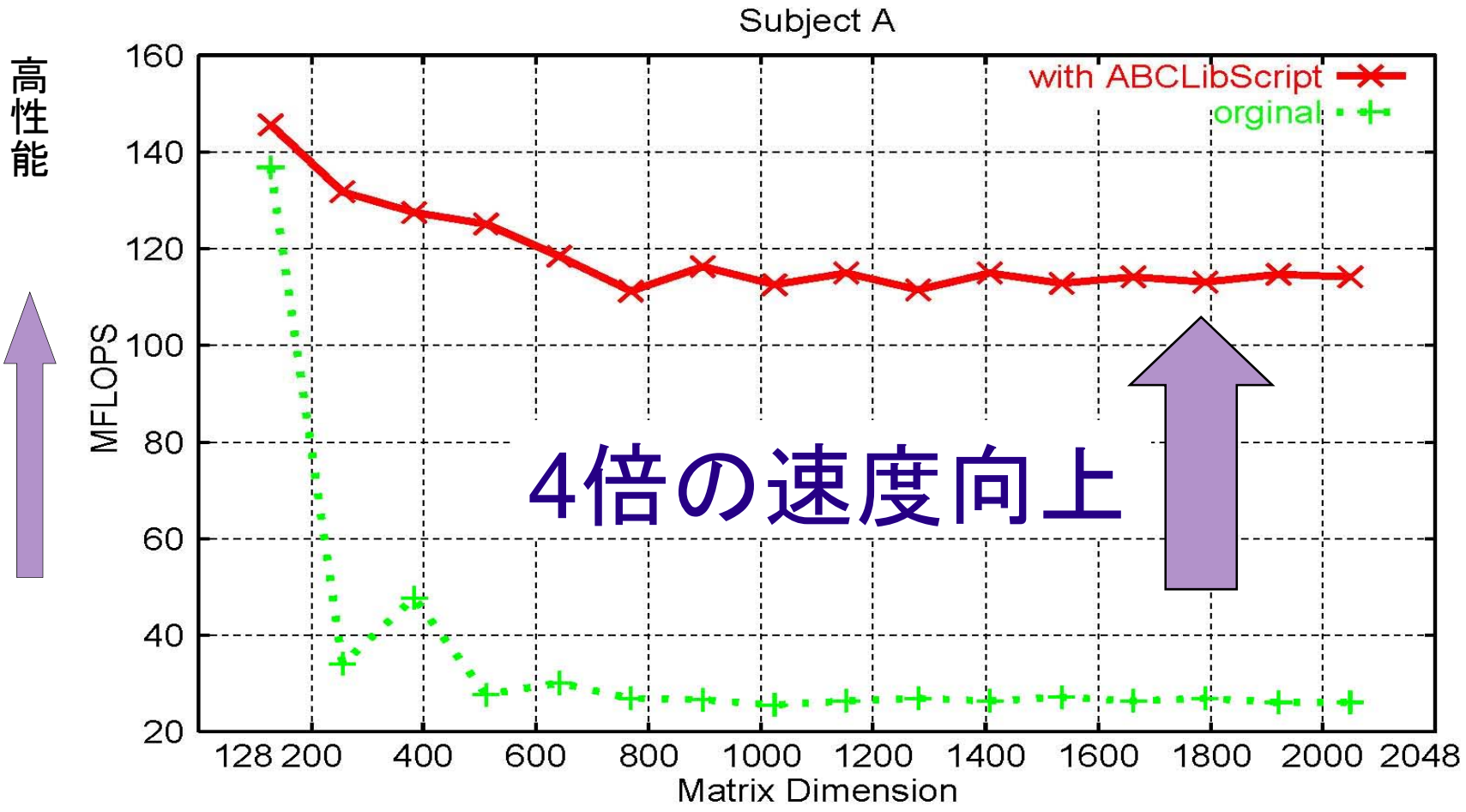
コスト定義関数
（epsが最小となるAT領域を
iter<5以下の条件で決定）

対象領域1、2
（チューニング領域）

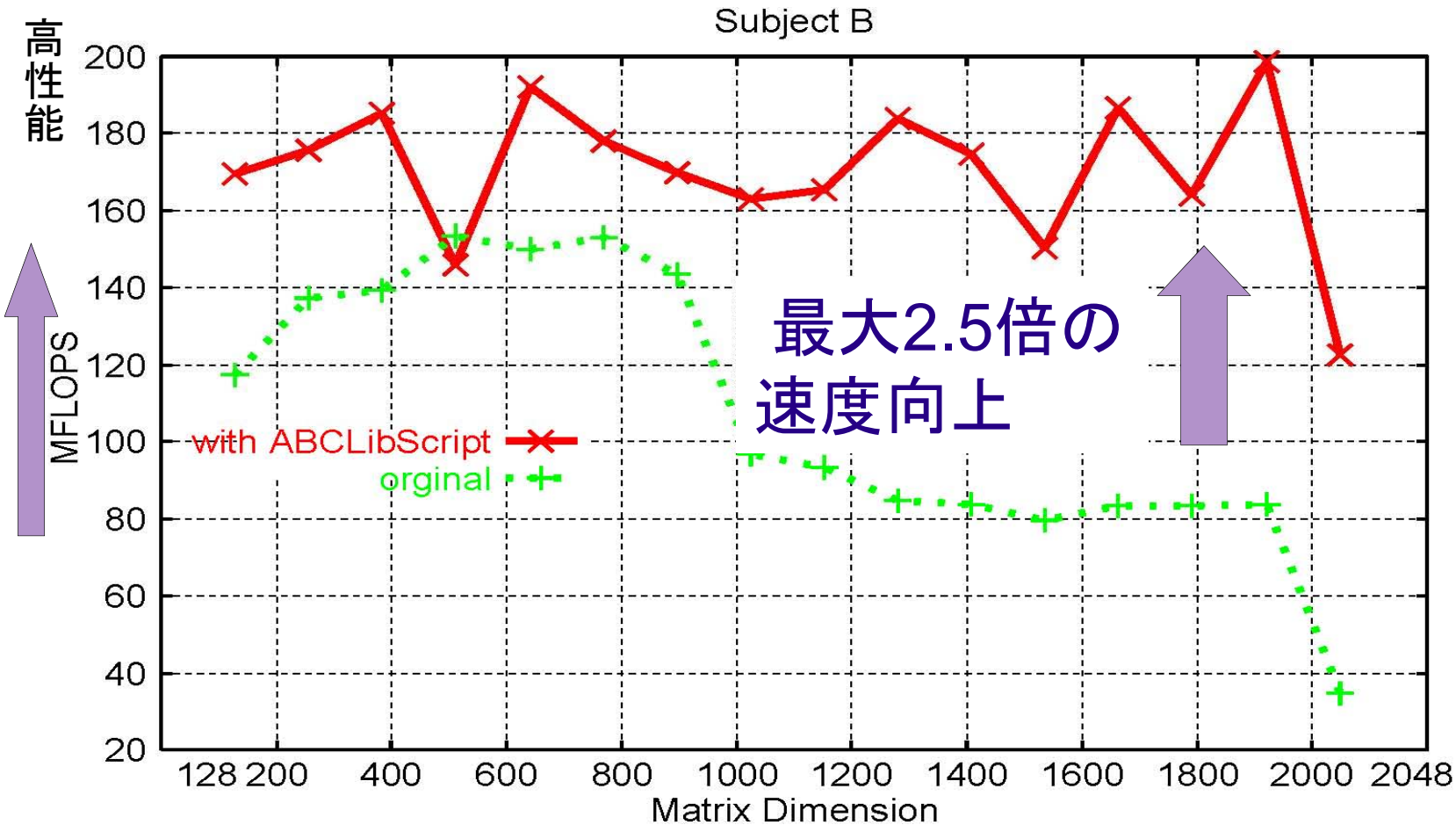
ABCLibScriptの効果検証

- 対象アプリケーション
  - 行列-行列積
- ABCLibScript ディレクティブ
  - アンローリング指定子
- 計算機環境
  - Intel Pentium4 (2.0GHz), PGI compiler
- 被験者
  - 被験者 A : ノン・エキスパート
  - 被験者 B : セミ・エキスパート
    (ブロック化アルゴリズムを知っている)
- 実験期間
  - 手によるチューニング : 2週間
  - ABCLibScript プログラミング : 2時間

実験結果 (1/2)


Subject A
高性能
MFLOPS
160
140
120
100
80
60
40
20
128 200 400 600 800 1000 1200 1400 1600 1800 2000 2048
Matrix Dimension
with ABCLibScript
orginal
4倍の速度向上

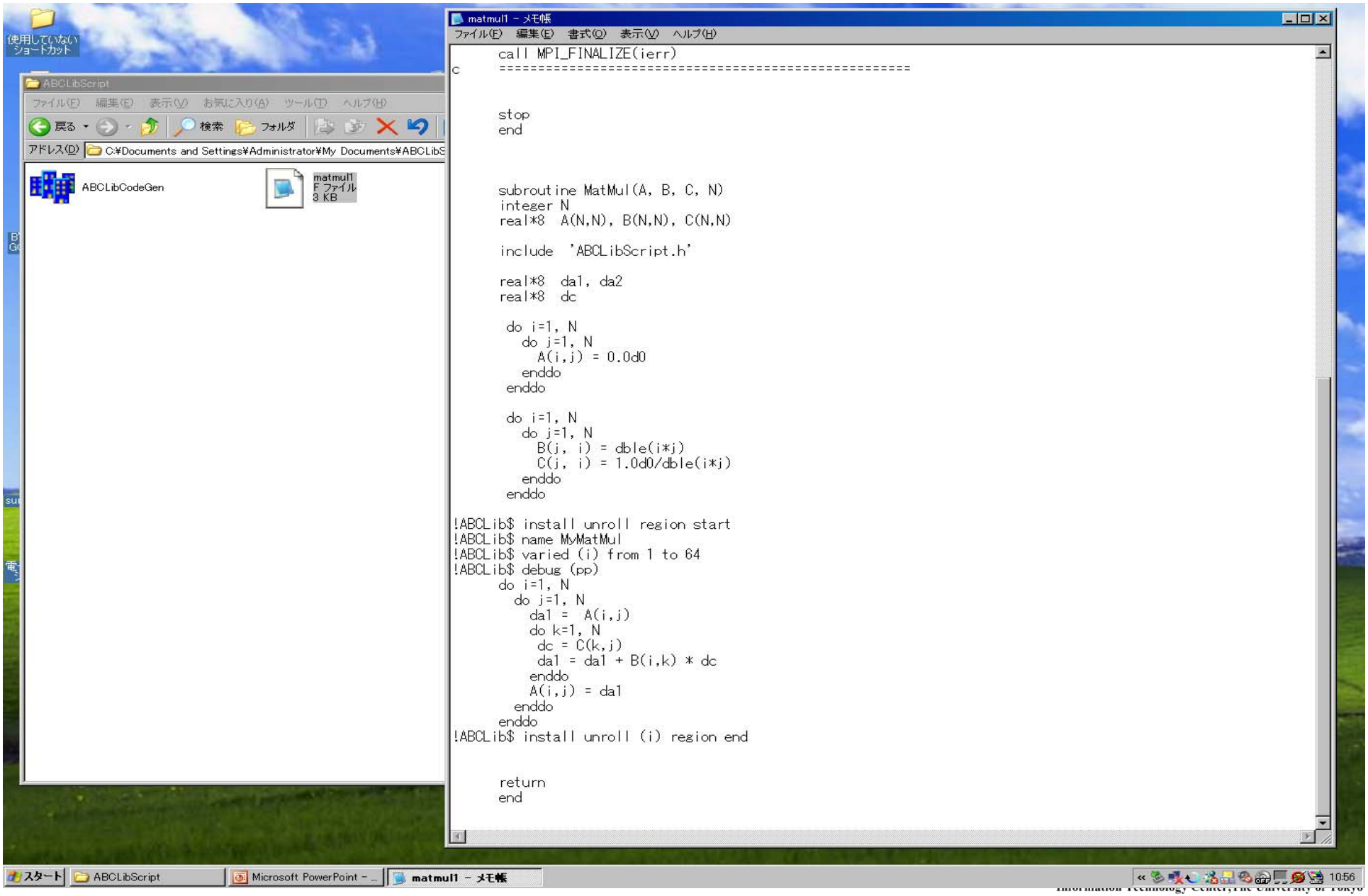
実験結果 (2/2)


Subject B
高性能
MFLOPS
200
180
160
140
120
100
80
60
40
20
128 200 400 600 800 1000 1200 1400 1600 1800 2000 2048
Matrix Dimension
with ABCLibScript
orginal
最大2.5倍の
速度向上

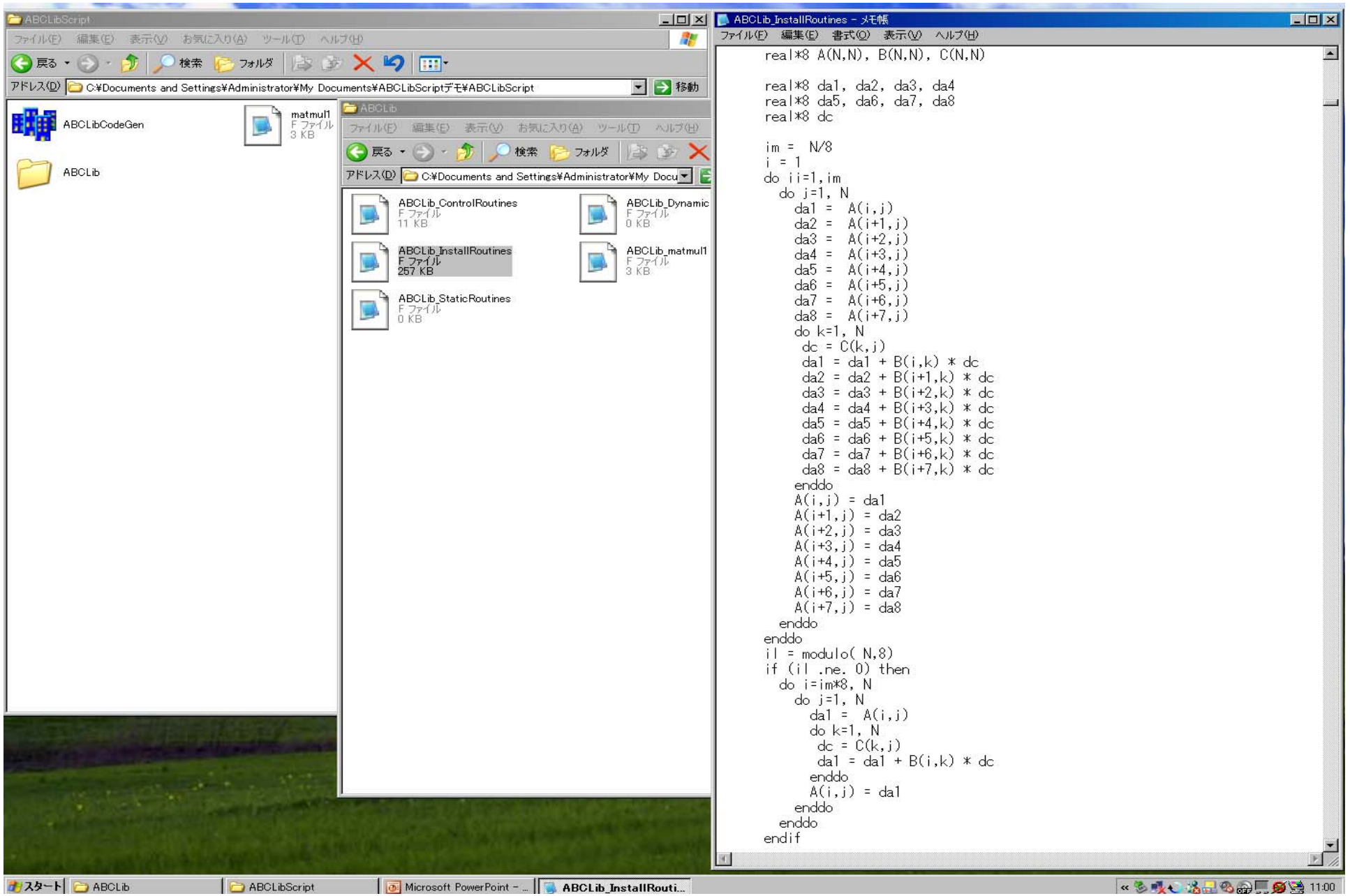
ABCLibScriptの効果（まとめ）

- ノン・エキスパート、セミ・エキスパート共に、手によるチューニングよりも高速なコードが自動作成できた
- 開発期間を **２週間から２時間** 程度に、より良い性能を保ったまま、短縮できる可能性がある

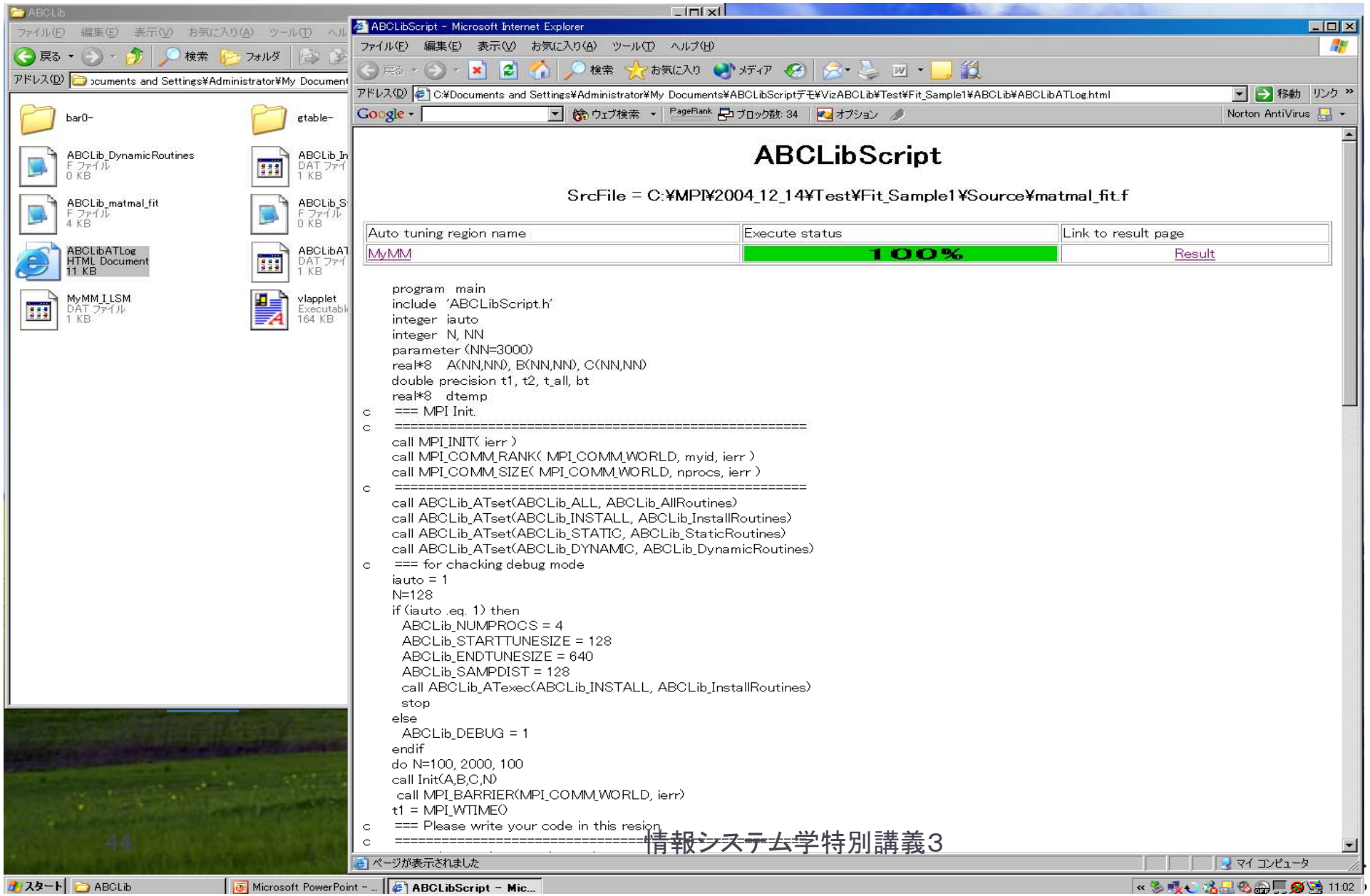
ABCLibScript 画面



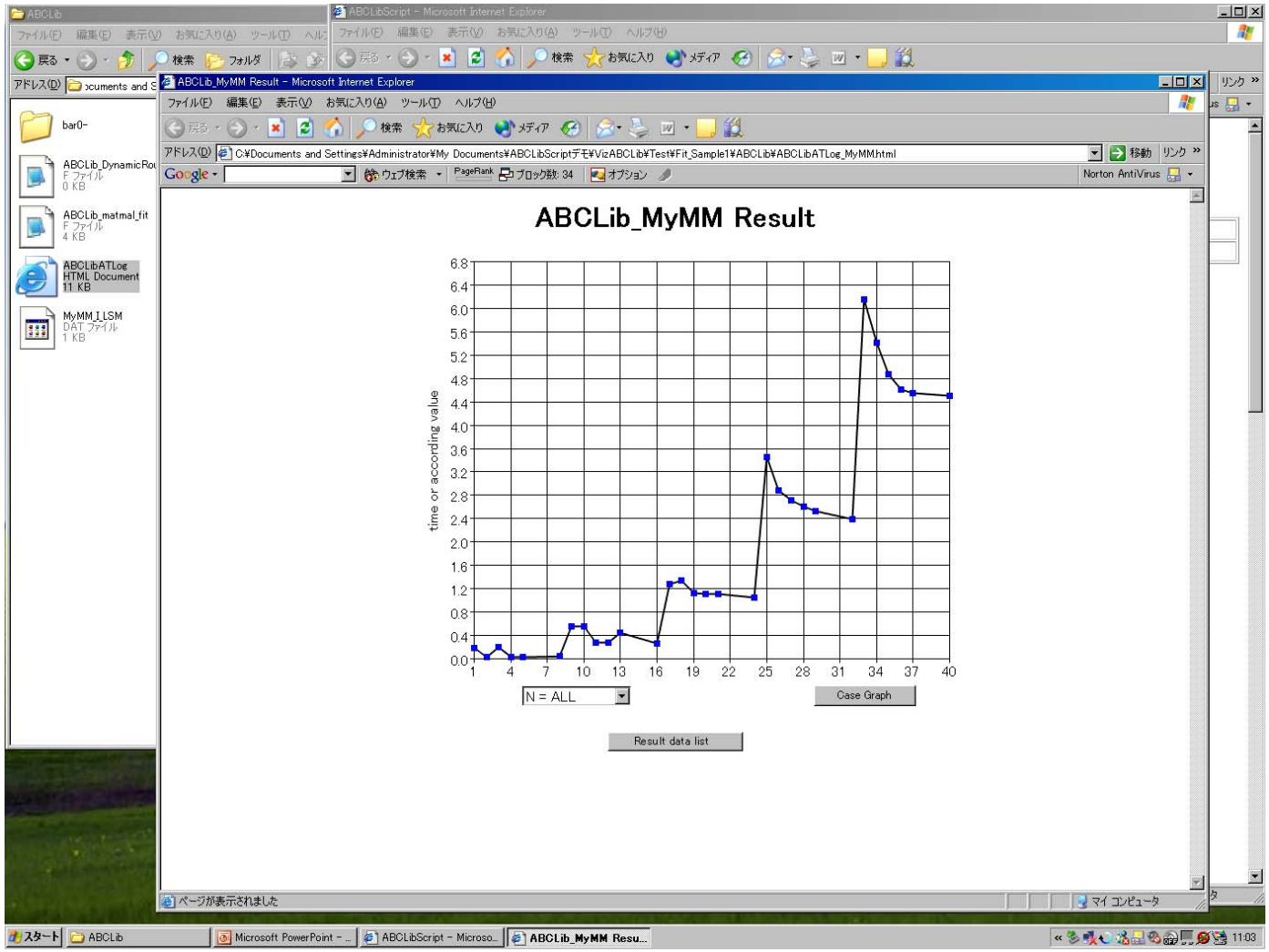
ソースコード自動生成画面



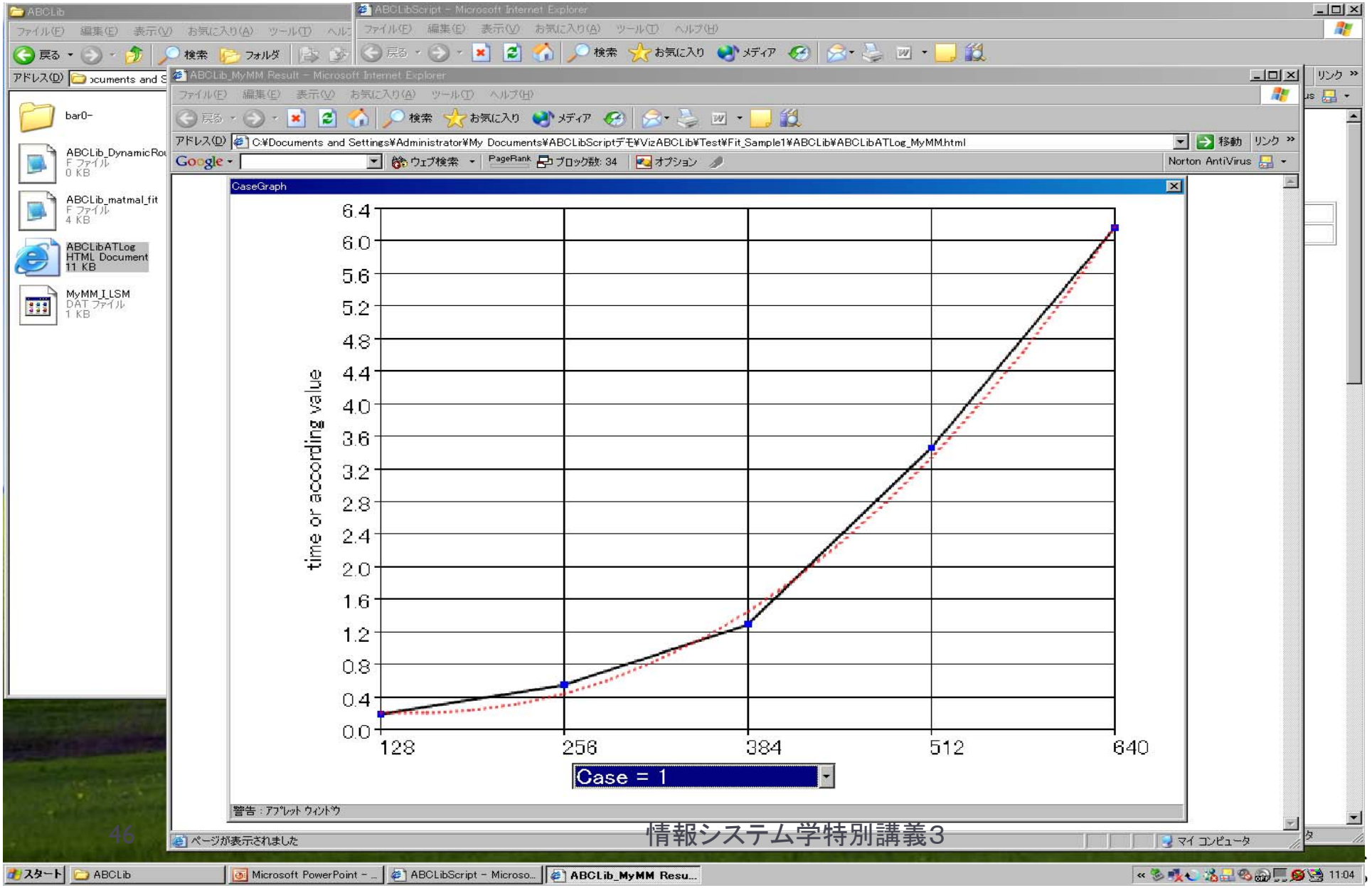
VizABCLib 起動画面



VizABCLib ログ表示画面



VizABCLib ログ詳細表示画面



詳細な情報を得るには

- ソースコード等の公開場所

  www.abc-lib.org

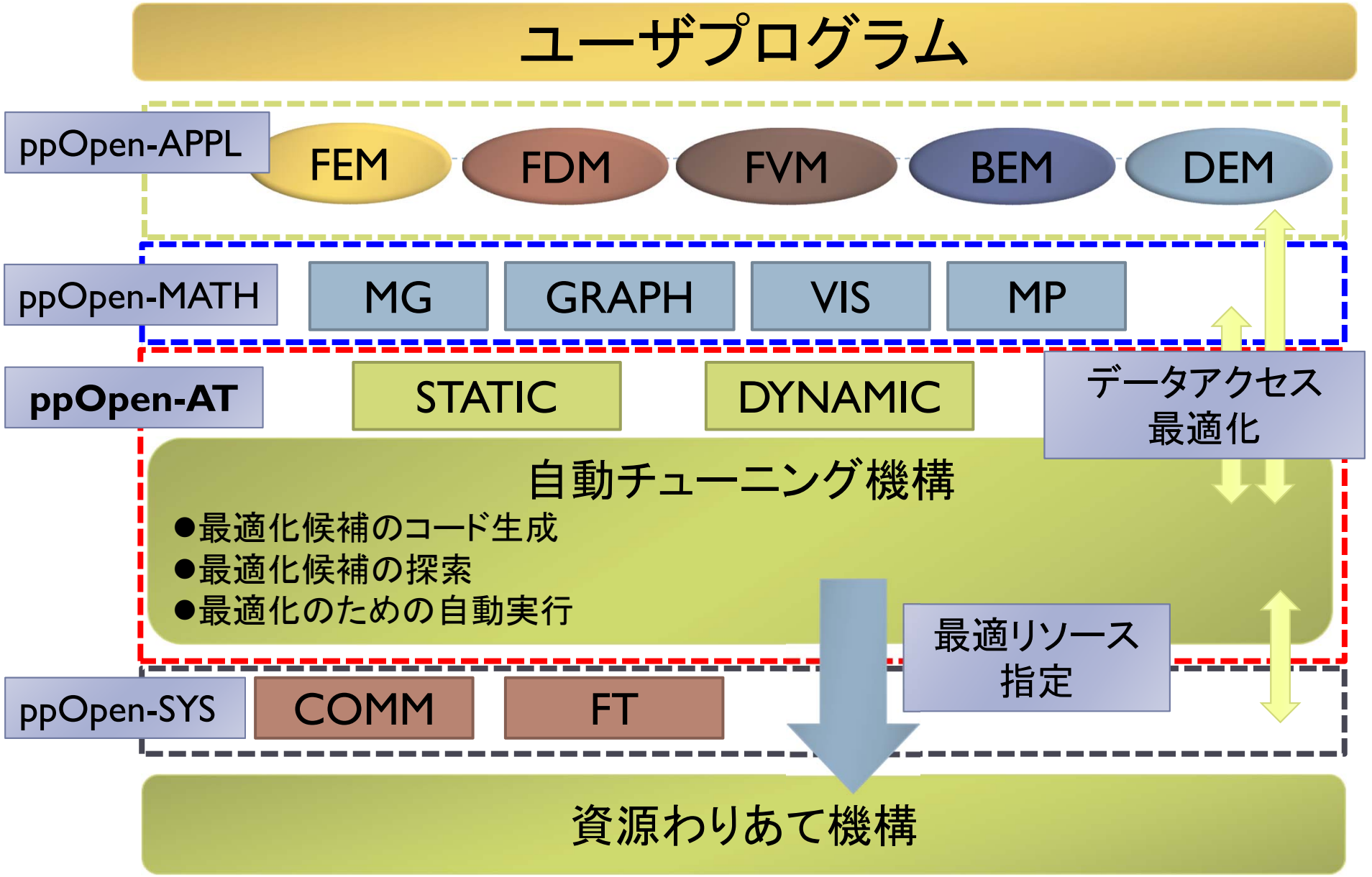
- 解説書
  「ソフトウエア自動チューニング
  ―数値計算ソフトウエアへの適用と
  　その可能性」
  慧文社

  ISBN4-905849-18-7

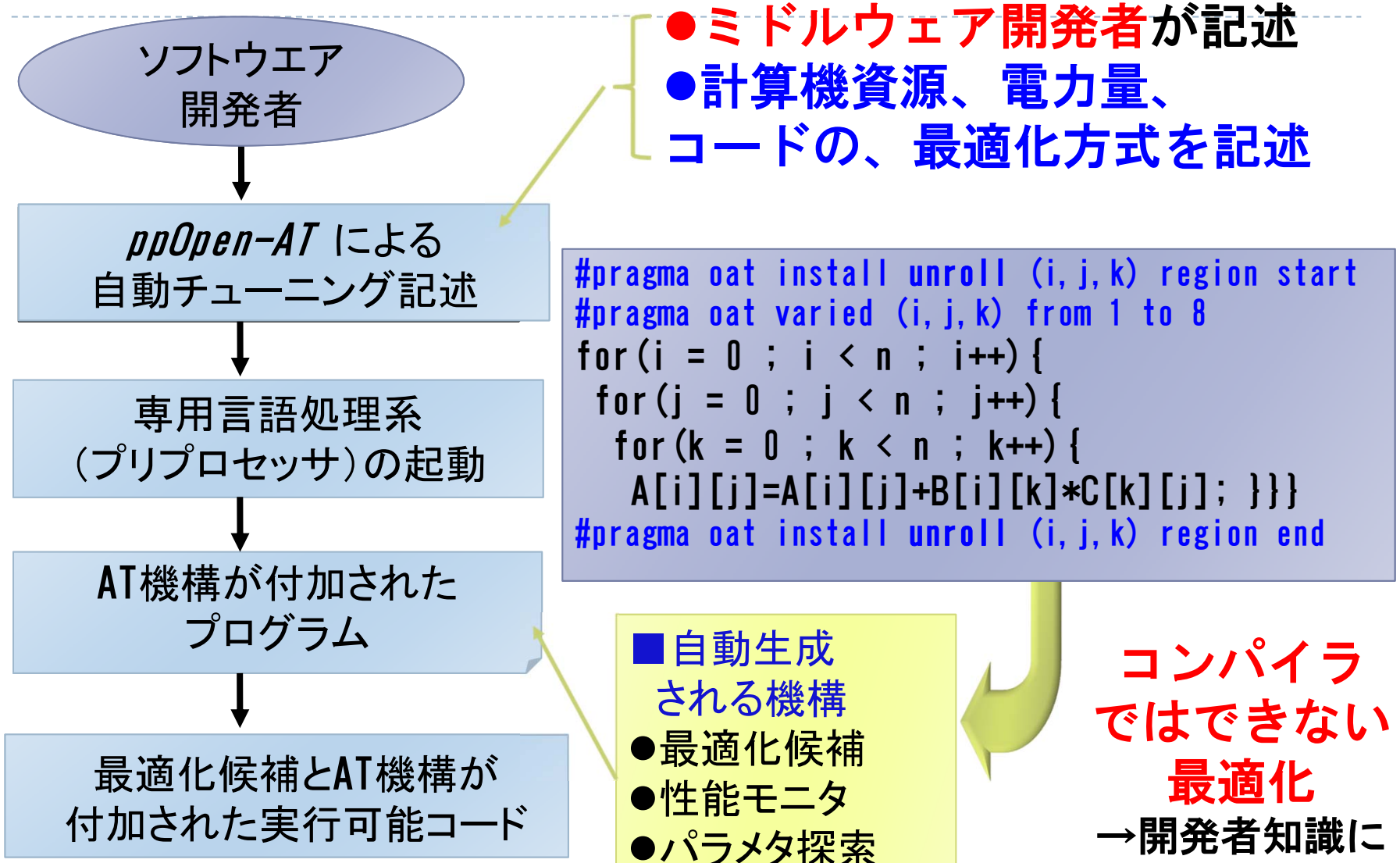
PpOpen-HPCプロジェクトと *ppOpen-AT*

*ppOpen-HPC*プロジェクト

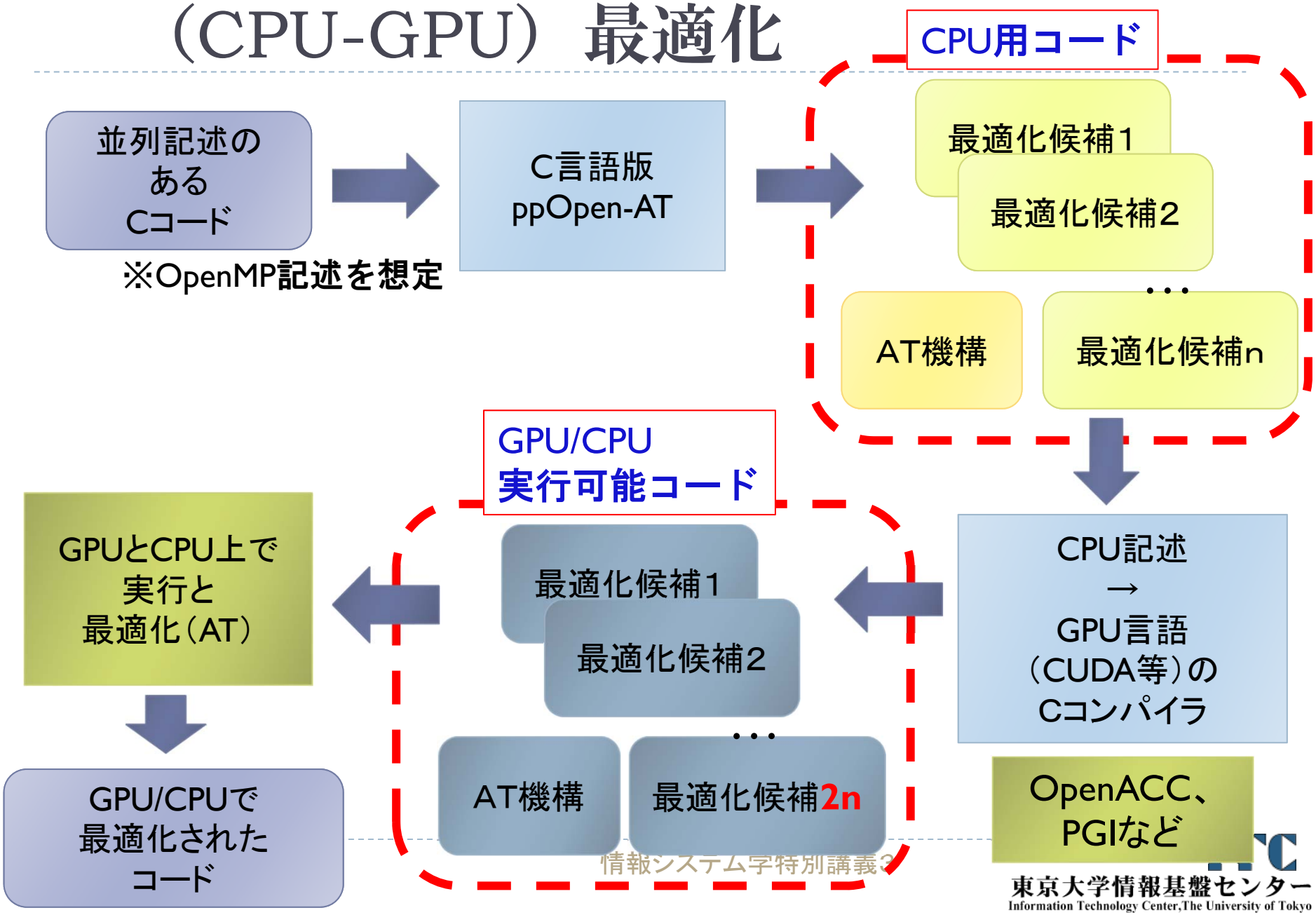
- <HPC用ミドルウェア>と<自動チューニング(AT)>
  - 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」（統括：米澤明憲教授）
  - H23年度JST CREST採択課題（2011～2015）
    「自動チューニング機構を有するアプリケーション開発・実行環境」
    - 代表：中島研吾教授（東京大学）
  - *ppOpen-HPC*：ポストペタ（post-peta, pp）スケールの並列計算機上でのシミュレーションコードについて、開発と最適化実行を支援するオープンソース基盤（ミドルウェア）
  - *ppOpen-HPC* は、科学技術計算に使われる多種の数値解法のライブラリから構成
- ***ppOpen-AT*** : *ppOpen-HPC* のプログラムのための自動チューニング専用言語
  - 先行研究*ABCLibScript* の成果を用いて開発


ユーザプログラム
ppOpen-APPL
FEM
FDM
FVM
BEM
DEM
ppOpen-MATH
MG
GRAPH
VIS
MP
ppOpen-AT
STATIC
DYNAMIC
データアクセス
最適化
自動チューニング機構
●最適化候補のコード生成
●最適化候補の探索
●最適化のための自動実行
最適リソース
指定
ppOpen-SYS
COMM
FT
資源わりあて機構
Many-core CPUs
GPU
Low Power
CPUs
Vector CPUs

AT専用言語*ppOpen-AT*によるソフトウェア開発手順

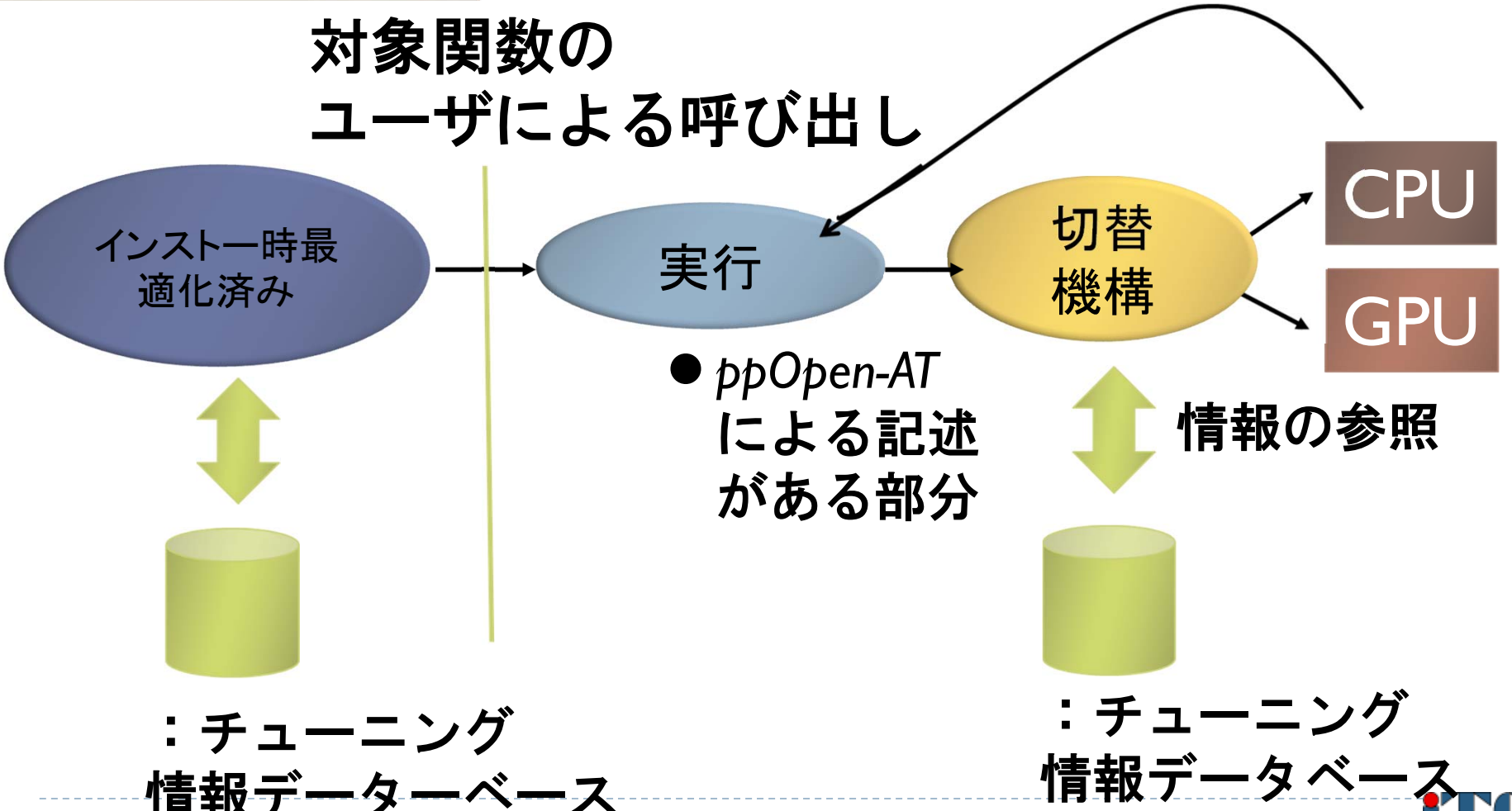


ppOpen-AT による非均質計算機環境（CPU-GPU）最適化



ATに関する実行時の挙動

インストール時最適化の場合



CPUとGPUの切り替えのための指示文

- ***#pragma oat allocate (<Object>)***
  - <Object> := { CPU | GPU | auto }
    - CPU : CPUでの実行
    - GPU : GPUでの実行
    - auto:
      CPU と GPU間でのコードを測定した後に最適なリソースを選択

例: 行列-行列積での CPU と GPU の切り替え例

```
#pragma oat install unroll (i,j,k) region start
#pragma oat name MyMatMul
#pragma oat varied (i,j,k) from 1 to 2
#pragma oat allocate (auto)
    for(i = 0 ; i < n ; i++){
      for(j = 0 ; j < n ; j++){
        for(k = 0 ; k < n ; k++){
              A[i][j] = A[i][j] + B[i][k] * C[k][j];
        } } }
#pragma oat install unroll (i,j,k) region end
```

CPU と GPU で最適な実行を選択

ppOpen-AT による自動生成コードの構成（CPUコード）

```
…
#pragma oat install unroll (i,j,k) region start
#pragma oat name MyMatMul
#pragma oat varied (i,j,k) from 1 to 4
        for(i = 0 ; i < n ; i++){
          for(j = 0 ; j < n ; j++){
            for(k = 0 ; k < n ; k++){
            A[i][j] = A[i][j] + B[i][k] * C[k][j];
            } } }
#pragma oat install unroll (i,j,k) region end
…
```

プログラム中の *ppOpen-AT* による記述

ppOpen-AT のプリプロセッサ

コードの修正：
OAT_<User File Name>.c

AT制御コード
OAT_ControlRoutines.c

AT候補

- 実装切り替え部
- 候補 #1
- 候補 #2
- …
- 候補 #n

インストール時ATの候補
OAT_InstallRoutines.c

実行起動前時 ATの候補
OAT_StaticRoutines.c

実行時 ATの候補
OAT_DynamicRoutines.c

“#pragma oat allocate(auto)”からの自動生成コード

```
int OAT_InstallMyMatMul(int n, Int iusw1) {
switch(iusw1){
    case 1:   OAT_InstallMyMatMul_1(n); break;
    case 2:   OAT_InstallMyMatMul_2(n); break;
              ….
    case 8:   OAT_InstallMyMatMul_8(n); break;
    case 9:   OAT_InstallMyMatMul_9(n); break;
              …
    case 16:  OAT_InstallMyMatMul_16(n); break;
```

CPUコード

GPUコード

```
int OAT_InstallMyMatMul_9 (int n) {
int i, j, k;
#pragma acc region
{ // #pragma allocate region start.
      for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
          for(k = 0 ; k < n ; k++){
            A[i][j] = A[i][j] + B[i][k] * C[k][j];

          } } }    } // #pragma allocate region end.  }
```

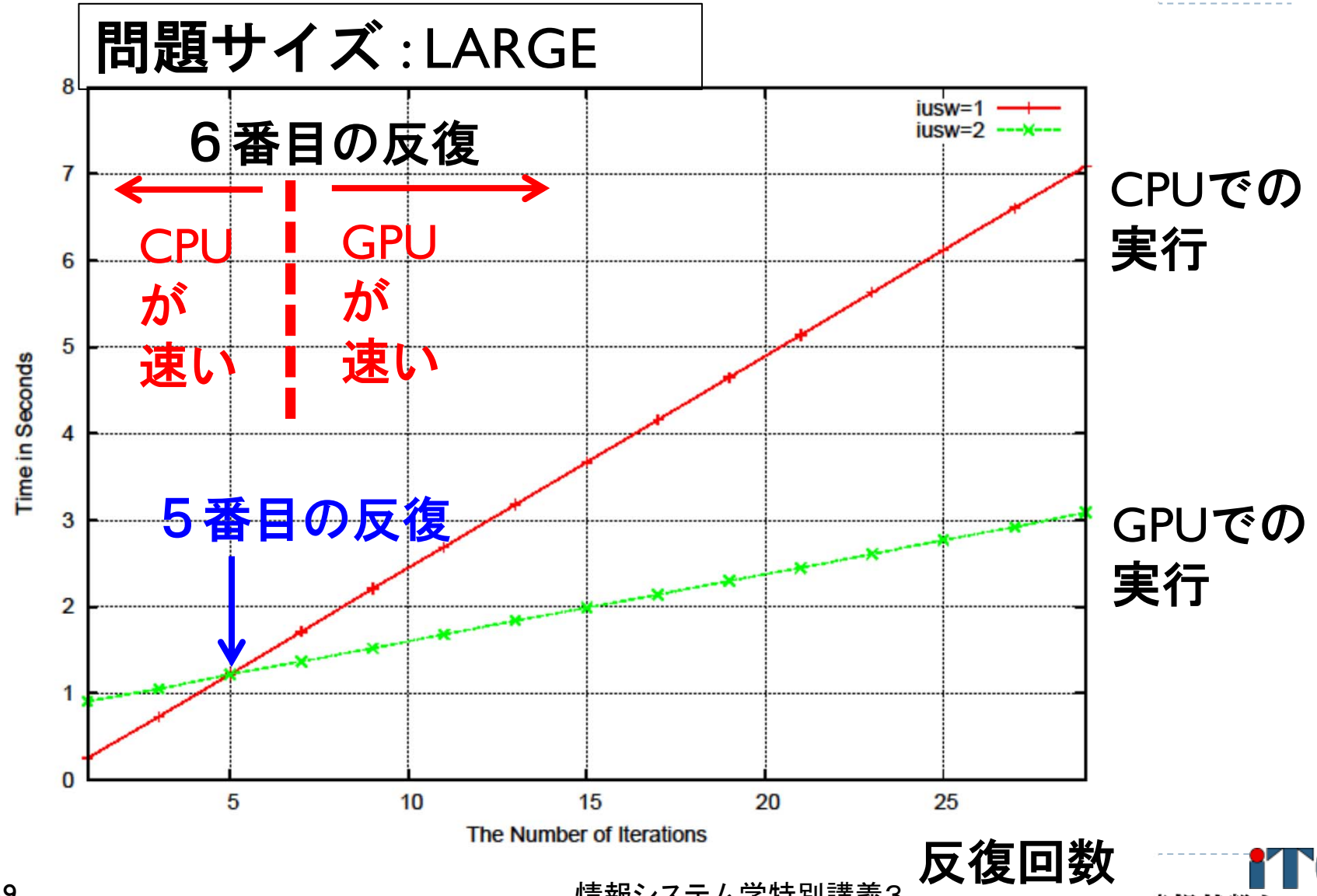
PGI Accelerator からの変換例（OpenACC 記述からも同様に可能）

例）
姫野ベンチマークのCPUとGPU切り替え

```
float jacobi(int nn) {
  int i,j,k,n;
  float gosa, s0, ss;
#pragma OAT static select region start
#pragma OAT name SelectHimeno
#pragma OAT allocate (auto)
#pragma OAT select sub region start
  for(n=0 ; n<nn ; ++n){
    gosa = 0.0;
    for(i=1 ; i<imax-1 ; i++)
      for(j=1 ; j<jmax-1 ; j++)
        for(k=1 ; k<kmax-1 ; k++){
          s0 = a[0][i][j][k]*p[i+1][j][k]+a[1][i][j][k]*p[i][j+1][k] + a[2][i][j][k]*p[i][j][k+1]+b[0][i][j][k]*(p[i+1][j+1][k]-p[i+1][j-1][k]-
p[i-1][j+1][k]+p[i-1][j-1][k]) +b[1][i][j][k]*(p[i][j+1][k+1]-p[i][j-1][k+1]-p[i][j+1][k-1]+p[i][j-1][k-1]) + b[2][i][j][k]*(p[i+1][j][k+1]-
p[i-1][j][k+1]-p[i+1][j][k-1]+p[i-1][j][k-1]) +c[0][i][j][k]*p[i-1][j][k]+c[1][i][j][k]*p[i][j-1][k]+c[2][i][j][k]*p[i][j][k-1] + wrk1[i][j][k];
          ss = ( s0 * a[3][i][j][k] - p[i][j][k] ) * bnd[i][j][k];
          gosa+= ss*ss;   /* gosa= (gosa > ss*ss) ? a : b; */
          wrk2[i][j][k] = p[i][j][k] + omega * ss;
        }
    for(i=1 ; i<imax-1 ; ++i)
      for(j=1 ; j<jmax-1 ; ++j)
        for(k=1 ; k<kmax-1 ; ++k)
          p[i][j][k] = wrk2[i][j][k];
  }   /* end n loop */
#pragma OAT select sub region end
#pragma OAT static select region end
  return(gosa);
}
```

← CPUとGPUの切り替え指定

姫野ベンチマークのCPUとGPUの切り替え効果



レポート課題

1. [L30] *ABCLibScript* の後続プロジェクトで開発されている*ppOpen-AT* をダウンロードせよ。
   サンプルプログラムのコードを処理し、自動チューニングコードを自動生成せよ。
   その後、自動チューニングコードをFX10で起動し、自動チューニングの効果を調べよ。

- *ppOpen-AT* ダウンロードページ

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

レポート課題

2. [L4O] 自分の研究で取り扱っているコードや、興味のあるアプリケーションコードに*ppOpen-AT*を適用し、自動チューニングコードを自動生成せよ。

   また、その自動チューニングコードをFX10で動かし、自動チューニングの効果を検証せよ。

- 注意：

  現在公開している*ppOpen-AT* はプロトタイプなため、コードが処理できないかもしれません。

  その場合は、動く範囲を特定したうえで性能を調べてください。この場合、レポート中でバグ報告を行ってください。

  また、*ppOpen-AT* の効果、欠点、改良点などを、自分のコードにおいて考察してください。

レポート課題

3. [L10] 国内、国外で行われている、自動チューニングソフトウェア工学に関連する研究を調べ、どの技術項目か関連するかを自分なりの解釈で説明した上で、その研究の概要を説明せよ。
4. [L5～] テキストブックの間違いをみつけて、レポートで報告せよ。
5. [L？] 各自が持つプログラムをFX10に移植し、性能を評価せよ。MPIやOpenMPで並列化することが好ましいが、そうでないものも認める。