

発展的課題： 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

2015年12月22日(火) 10:25－12:10

スパコンプログラミング(1)、(I)



東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

講義日程（工学部共通科目）

レポートおよびコンテスト課題
(締切:
2016年2月11日(木)24時 厳守

~~1. 10月6日: ガイダンス~~

~~2. 10月13日~~

- ~~● 並列数値処理の基本演算(座学)~~

~~3. 10月20日: スパコン利用開始~~

- ~~● ログイン作業、テストプログラム実行~~

~~4. 10月27日~~

- ~~● 高性能演算技法1
(ループアンローリング)~~

~~5. 11月10日~~

- ~~● 高性能演算技法2
(キャッシュブロック化)~~

~~6. 11月24日~~

- ~~● 行列ベクトル積の並列化~~

~~7. 12月1日(8:30-10:15)~~

- ~~● ベキ乗法の並列化~~

~~8. 12月1日(10:25-12:10)~~

- ~~● 行列・行列積の並列化(1)~~

~~9. 12月8日~~

- ~~● 行列・行列積の並列化(2)~~

~~10. 12月15日~~

- ~~● LU分解法(1)~~
- ~~● コンテスト課題発表~~

9. 12月22日

- 計算機保守のため座学
- ソフトウェア自動チューニング
- 非同期通信

10. 2016年1月5日

- LU分解法(2)

12. 1月12日

- LU分解法(3)

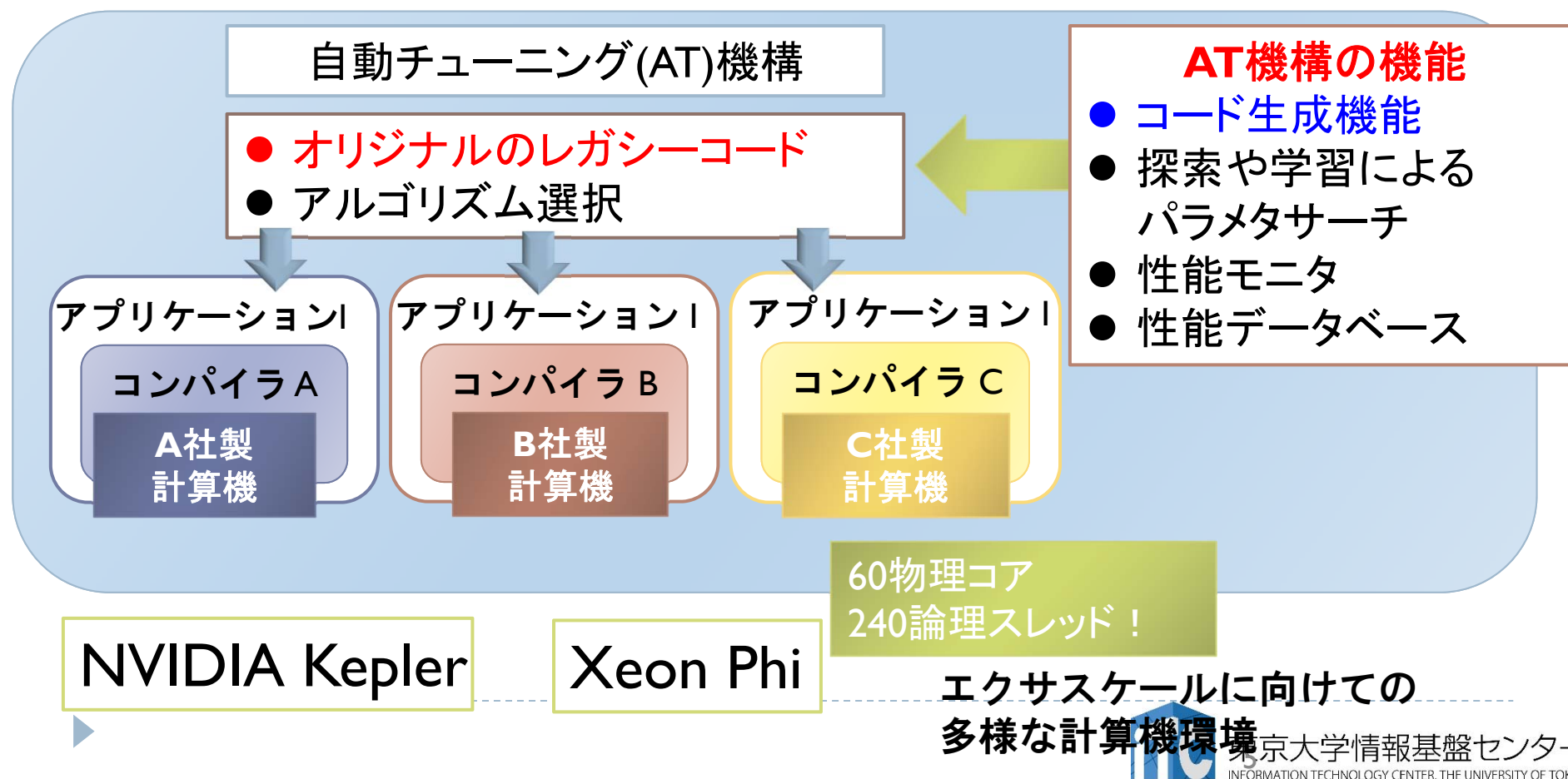
講義の流れ

1. 背景
2. ソフトウェ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ABC LibScript
5. ppOpen-HPCプロジェクトとppOpen-AT
6. レポート課題

背景

「性能可搬性」の実現

- 複数計算機で有効な最適化が提供できるようにする
最適化に関するパラダイム（HPCI 技術ロードマップ白書、
数値計算ライブラリのための自動チューニング、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コスト高の問題

● なぜ、コストが高いか

1. 探索空間爆発の問題
 - ソフトウェ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....

自動チューニング技術要求

- ▶ 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
 - ... など多数の要求
- ▶ 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

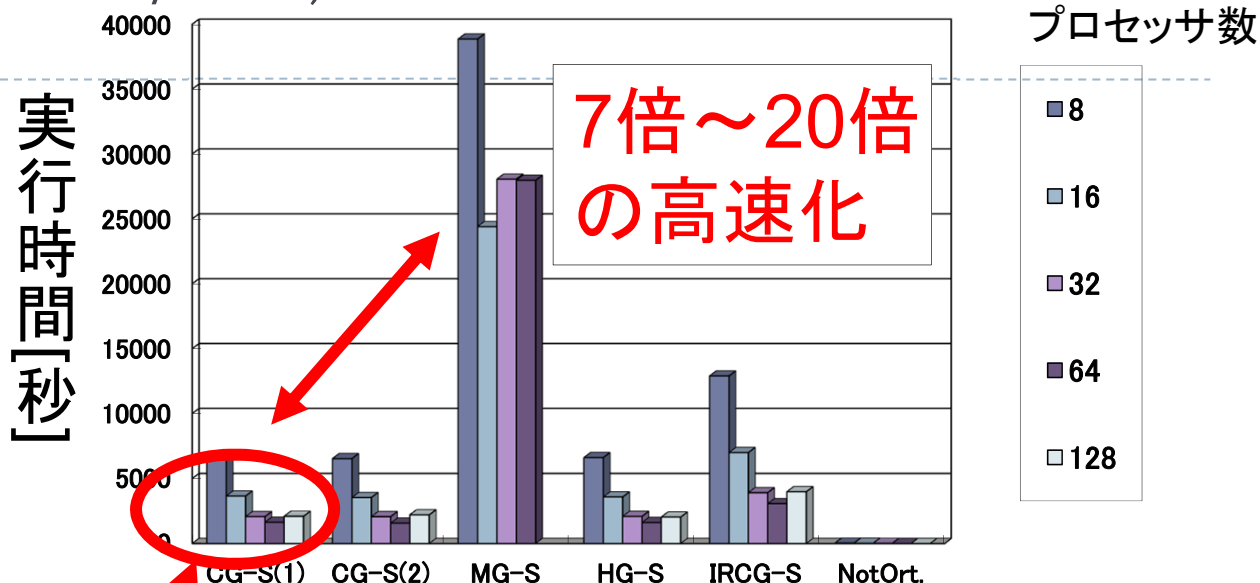


実例：実行時アルゴリズム選択

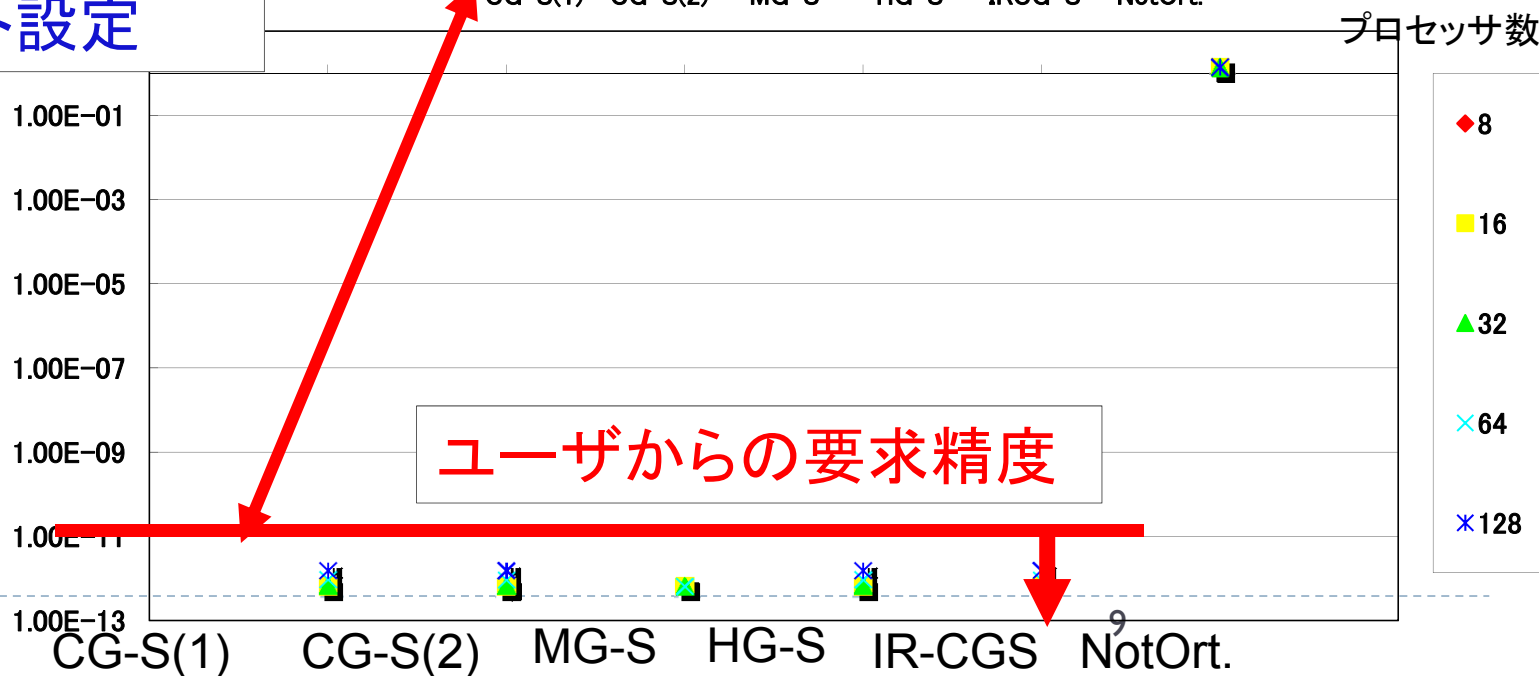
(HITACHI SR8000/MPP)

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

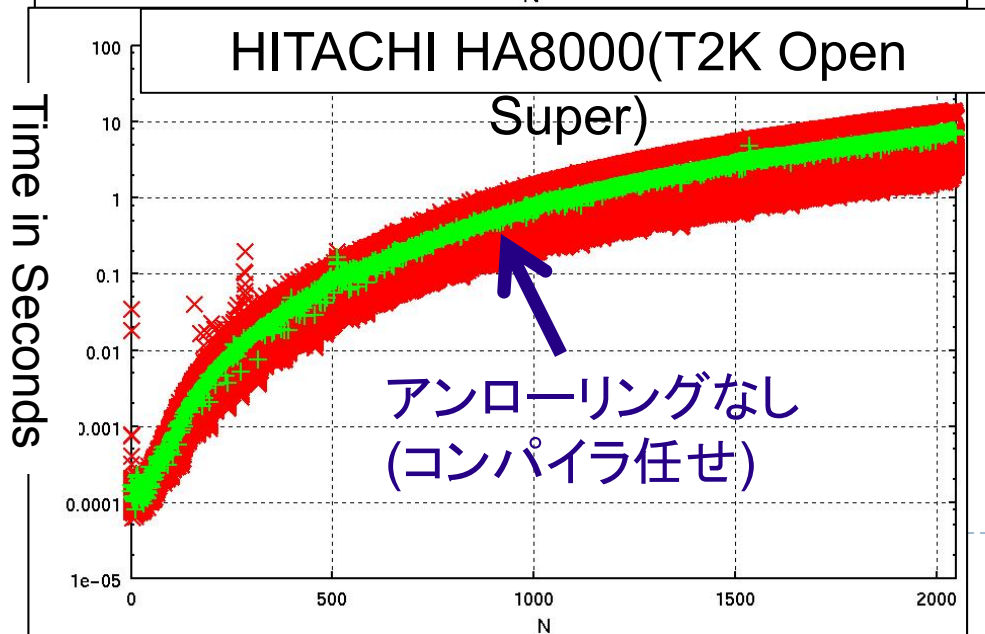
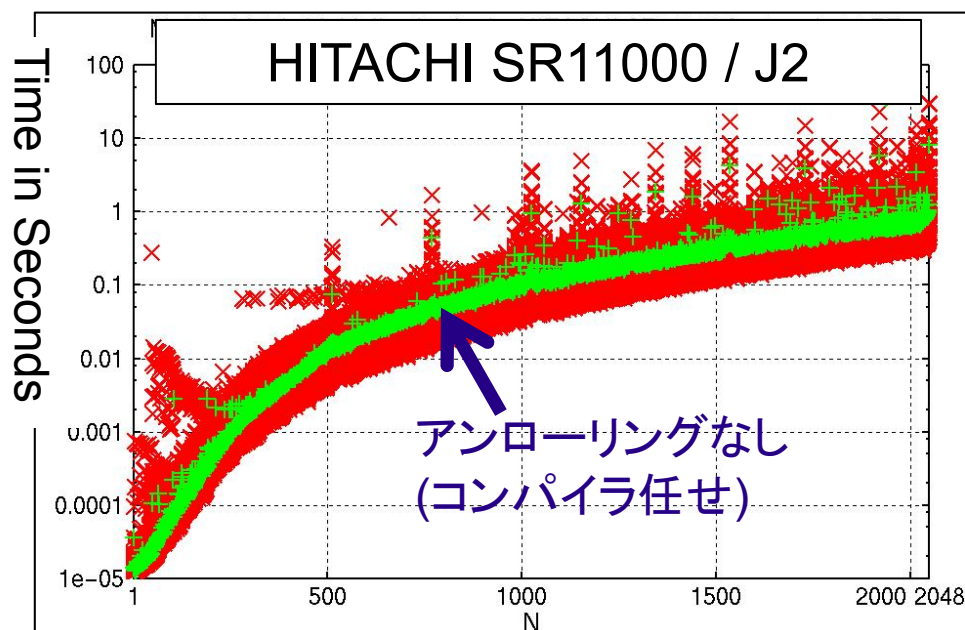
MG-S(修正G-S)：
多くの場合の
デフォルト設定



精度[Frobenius]



実例: 先進アーキテクチャによる不安定性



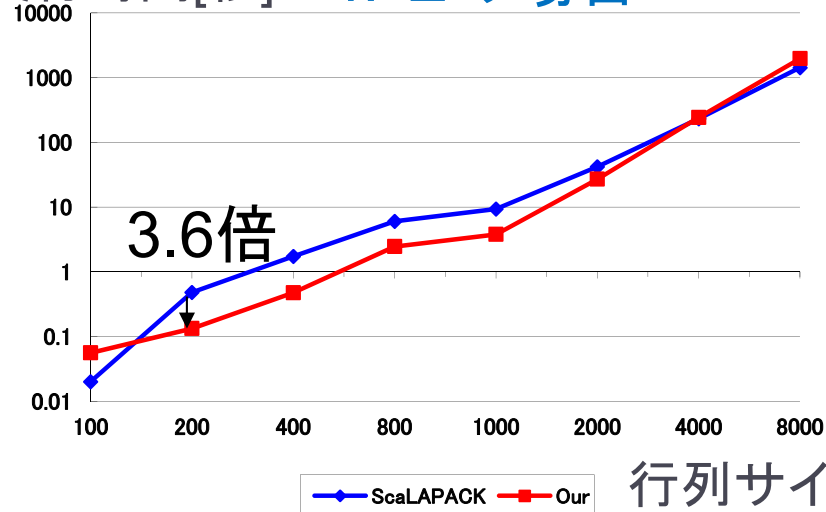
- 密行列の行列-行列積
BLASを用いていない単純コード
- 3重ループ (i,j,k), アンローリング1段~4段
- 次元Nに関し $4*4*4=64$ 種類 の実装
- 1 から2048次元まで1刻みのデータ.
- コンパイラ HITACHI Optimized Fortran90.
オプション: -Oss
- 自動並列化(ノード内)
- 計算機構成:
 - HITACHI SR11000/J2
 - HA8000 (T2K Open Supercomputer (Todai Combined Cluster))
 - Installed in Information Technology 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!

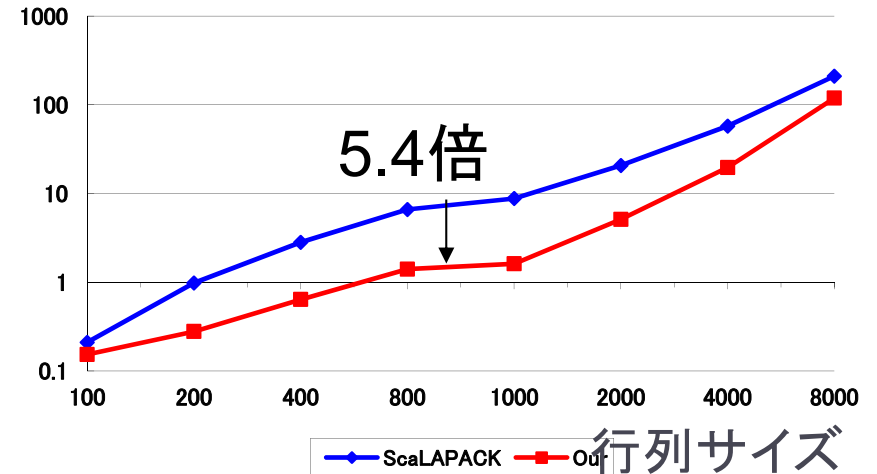
実例：超並列アルゴリズムの構築と適応的選択

(日立SR2201、Householder三重対角化)

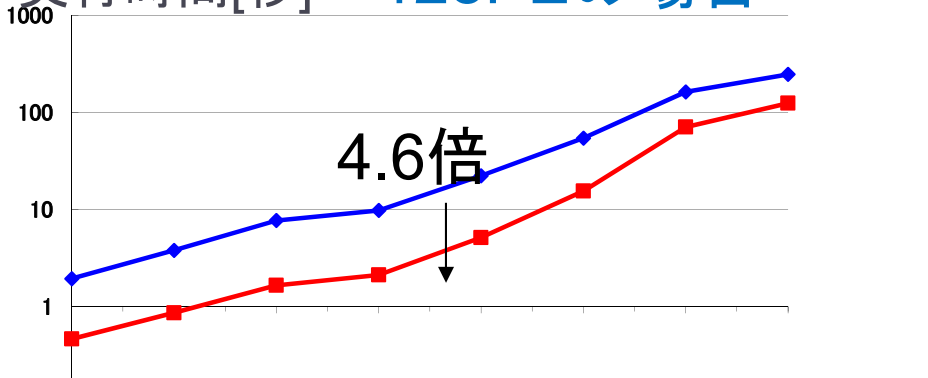
実行時間[秒] 4PEの場合



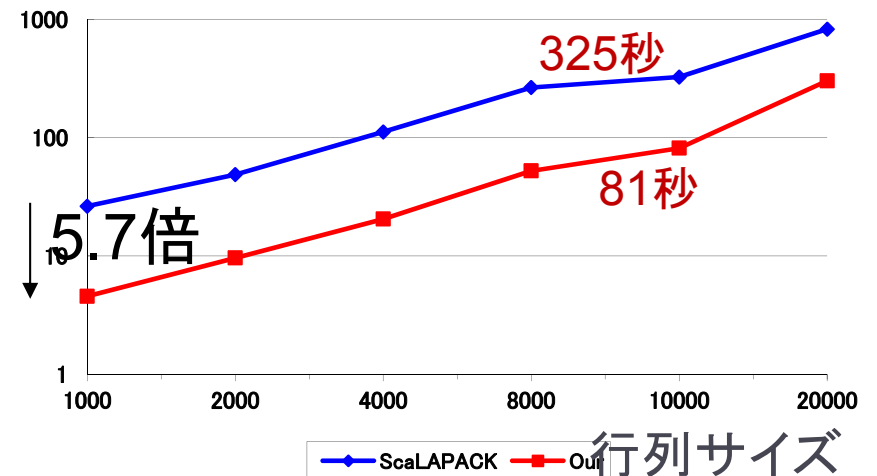
実行時間[秒] 64PEの場合



実行時間[秒] 128PEの場合



実行時間[秒] 512PEの場合



従来アルゴリズム(ライブラリ)は
超並列環境(10万並列)を指向した
アルゴリズムになっていない

ソフトウェア 自動チューニングとは

自動チューニング機構とは

- 計算機アーキテクチャ
- 計算機システム



性能調整つまみ
(性能パラメタ)

- プログラム
- アルゴリズム



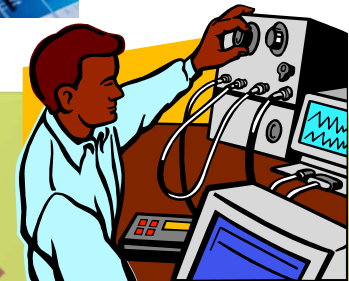
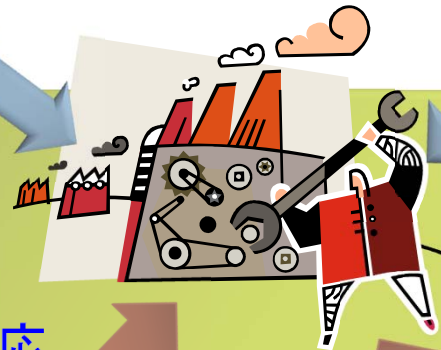
調整機構

- 最適化
- パラメタ探索
- 学習／自己適応

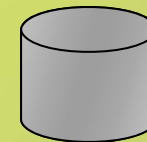
- プログラム
- アルゴリズム



つまみ
自動生成
機構



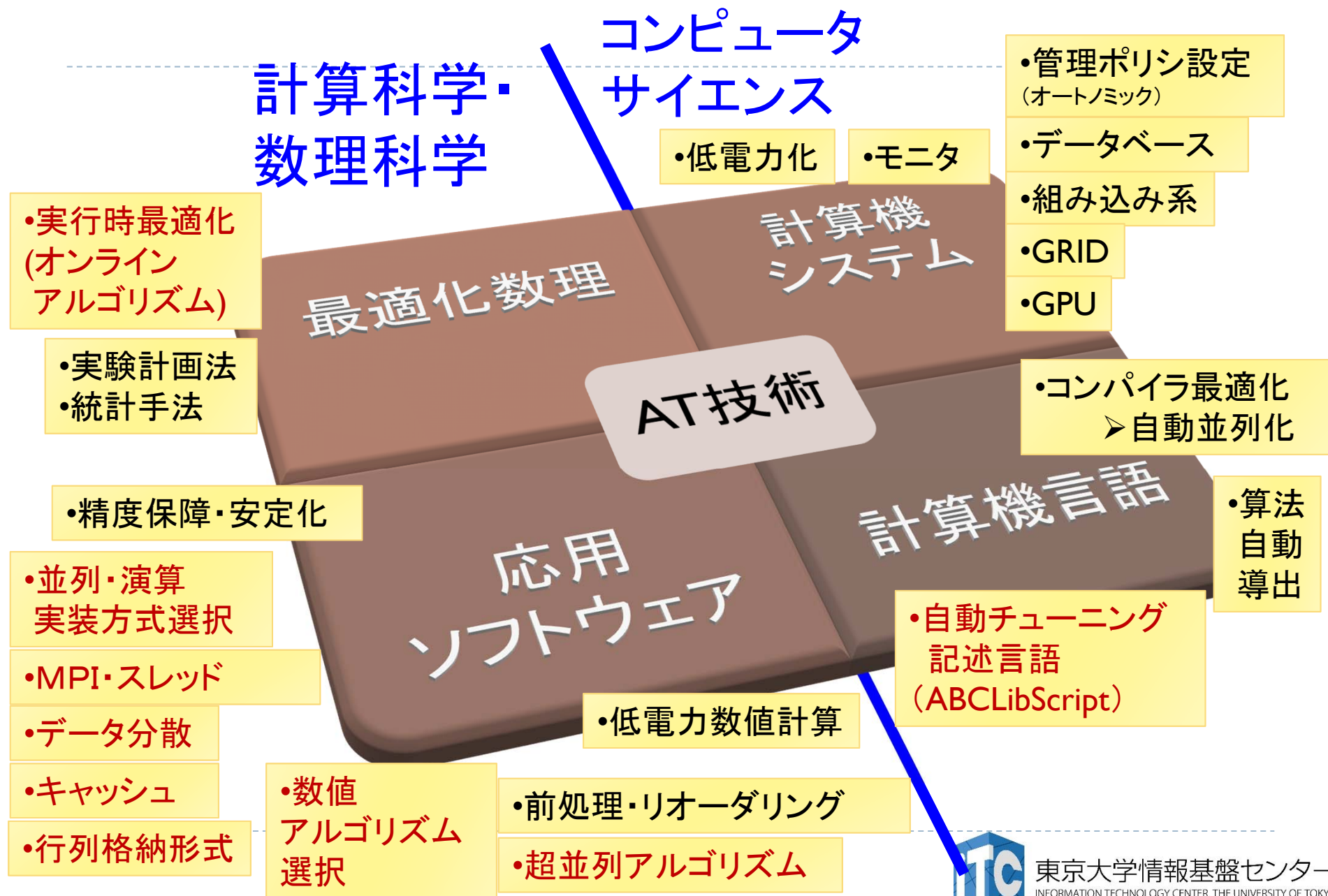
性能モニタ
機構



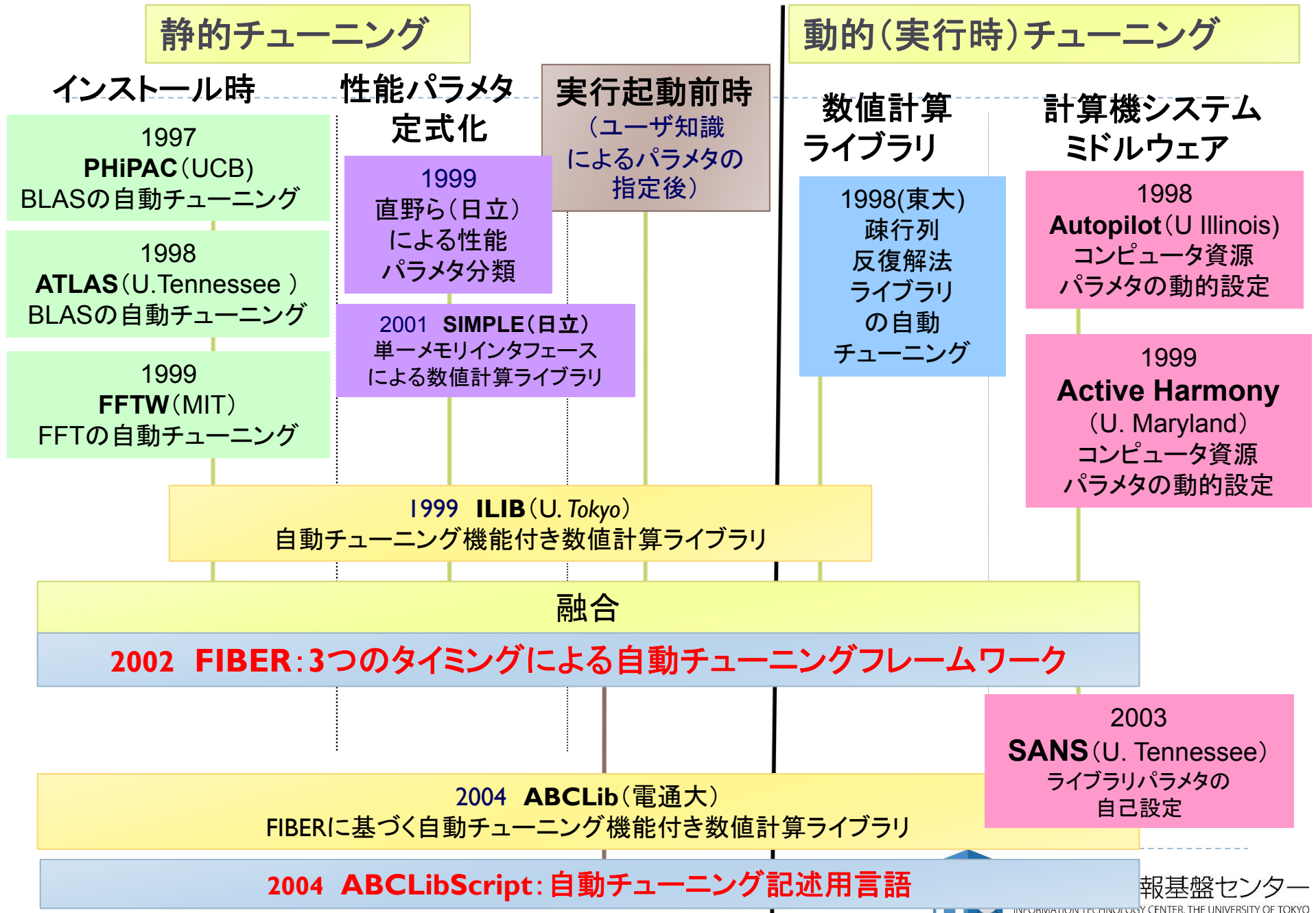
性能データベース

自動チューニング機構

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ェア工学 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
 - ▶ ソフトウェア単位、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. システムソフトウェア性能の検討
 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. Iなどに戻る

自動チューニングソフトウェア工学 達成のために（2 / 2）

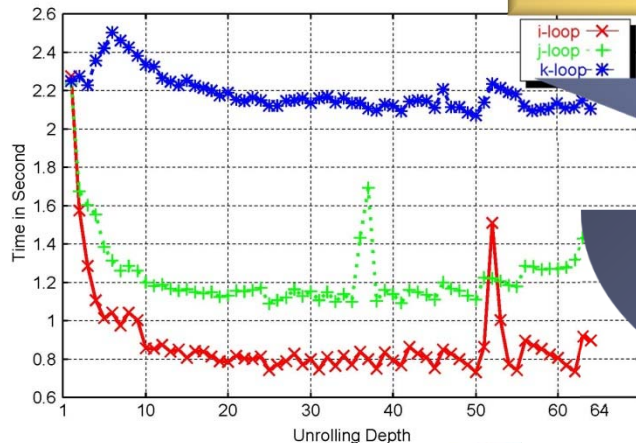
1. ソフトウェア・アーキテクチャの確立
 - ▶ 自動チューニング機構が考慮されていること
2. チューニングのための計算機言語と言語処理系があること
3. 「実行→測定→解析→ソース変更→実行」というチューニング作業のサイクルが、できるだけ自動化されていること
 - ▶ 自動チューニングツールの提供
 - ▶ 自動解析手法（知識探索手法）の確立
 - ▶ データベース化のための方式の確立



ソフトウェア自動チューニング のソフトウェア工学的観点

コード生成

コンパイルと実行



3. 最適化フェーズ

実行結果の解析

2. プログラミング
フェーズ

```
!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
```

対象計算機

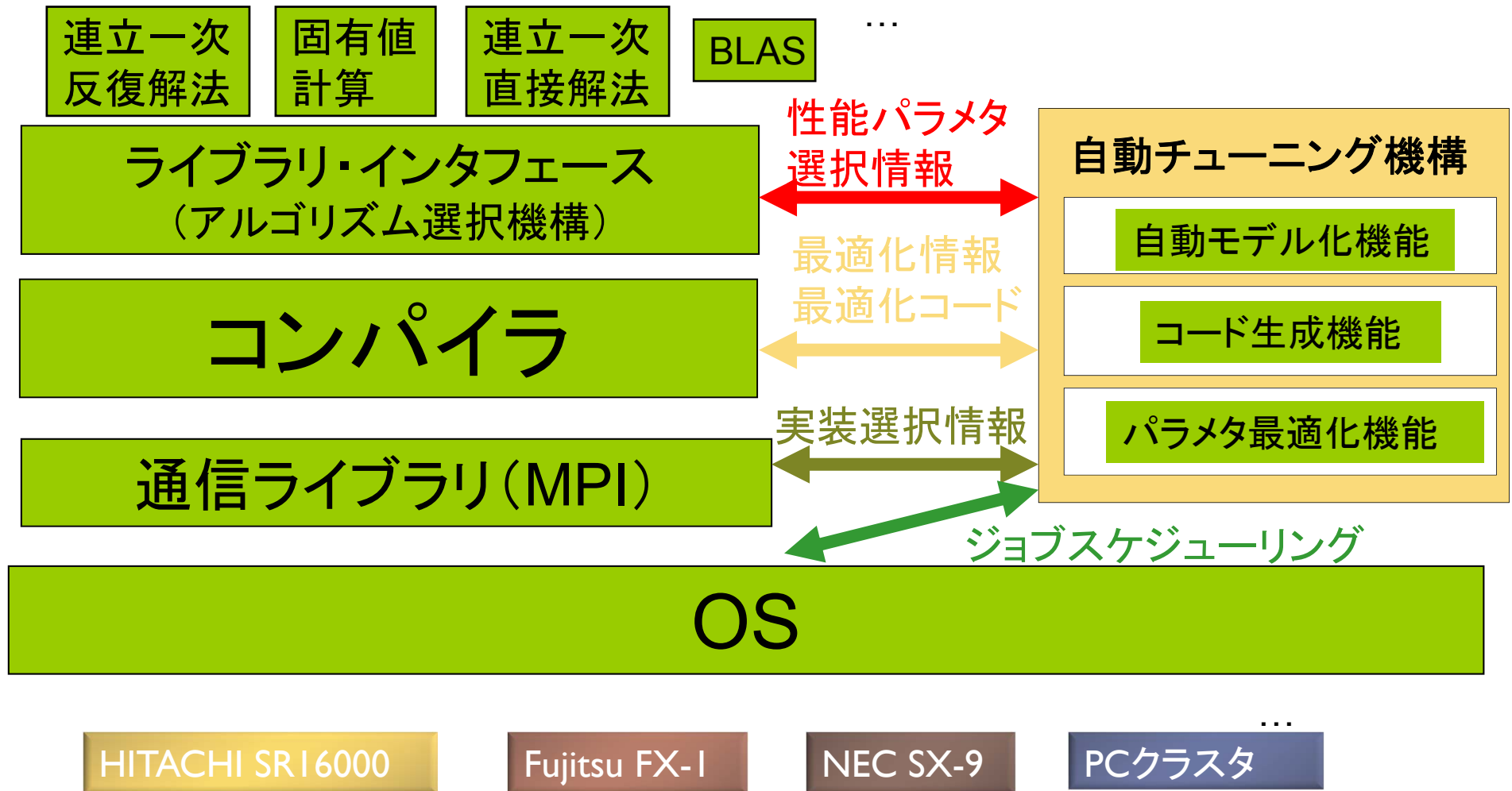
1. 仕様策定フェーズ

4. データベース化
とチューニング
知識探索
フェーズ

チューニング知識
データベース

構想

ミドルウェアとしての自動チューニング機構の確立



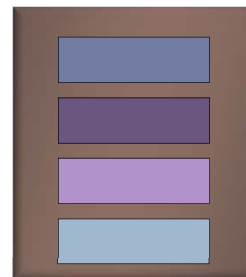
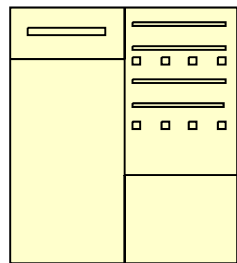
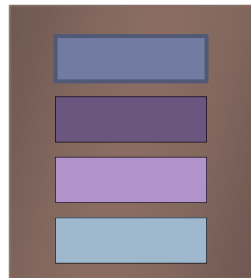
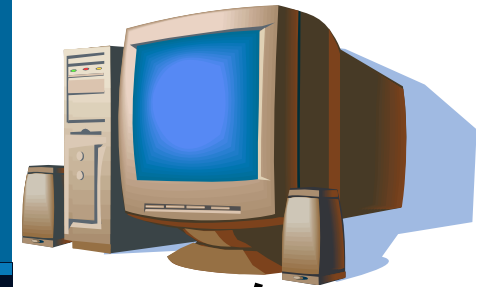
FIBERフレームワークの概要

- 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

ソフトウェア
開発者

計算機

自動チューニング
機能付きソフトウェア



公開サーバ

ダウンロード

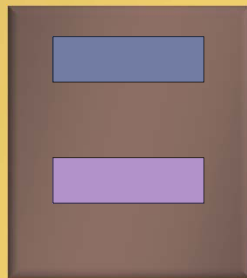
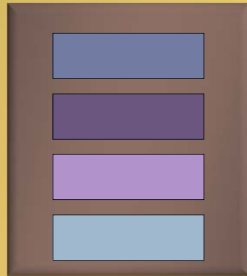
エンドユーザ



東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

FIBERフレームワークにおける自動チューニング

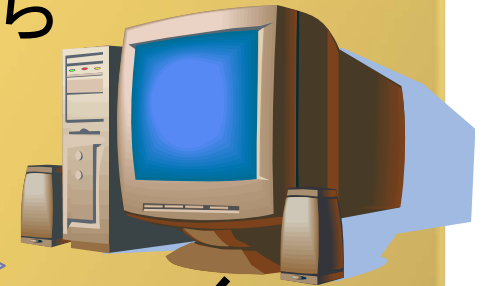
自動チューニング
機能付きソフトウェア



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

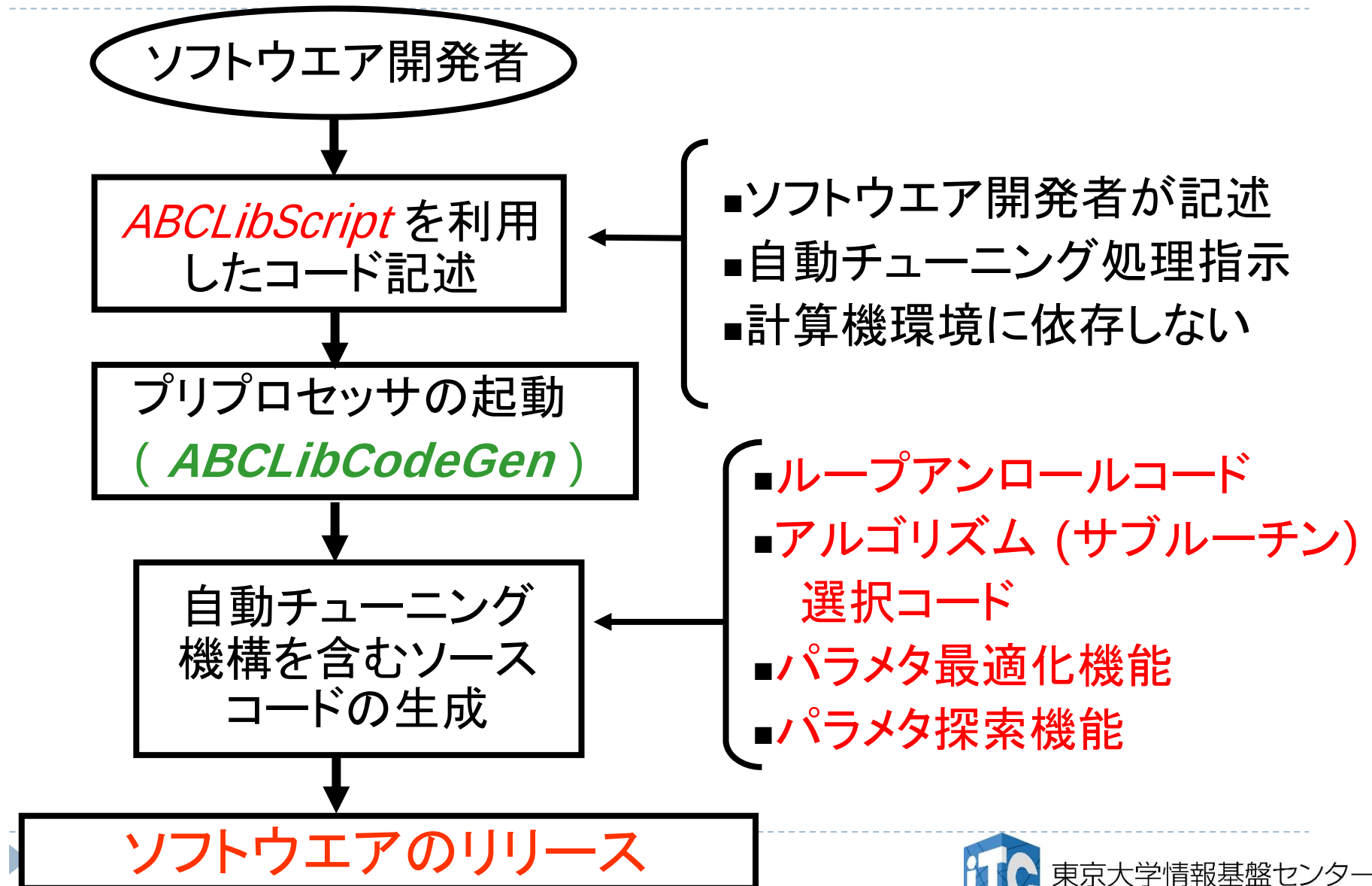
1. **インストール時**
2. **実行起動前時**
3. **実行時**



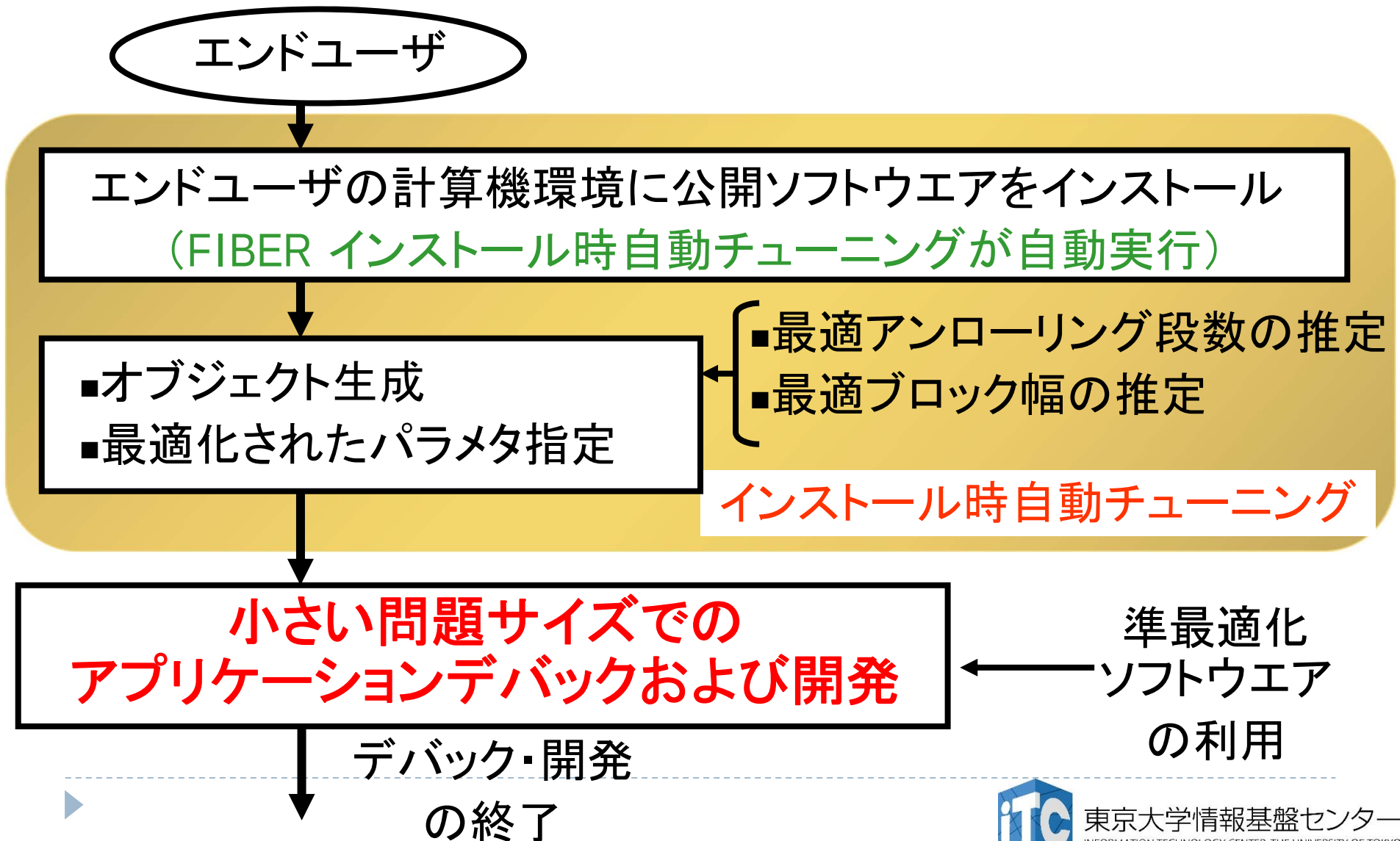
エンドユーザ

自動チューニング記述言語 ABCLibscript

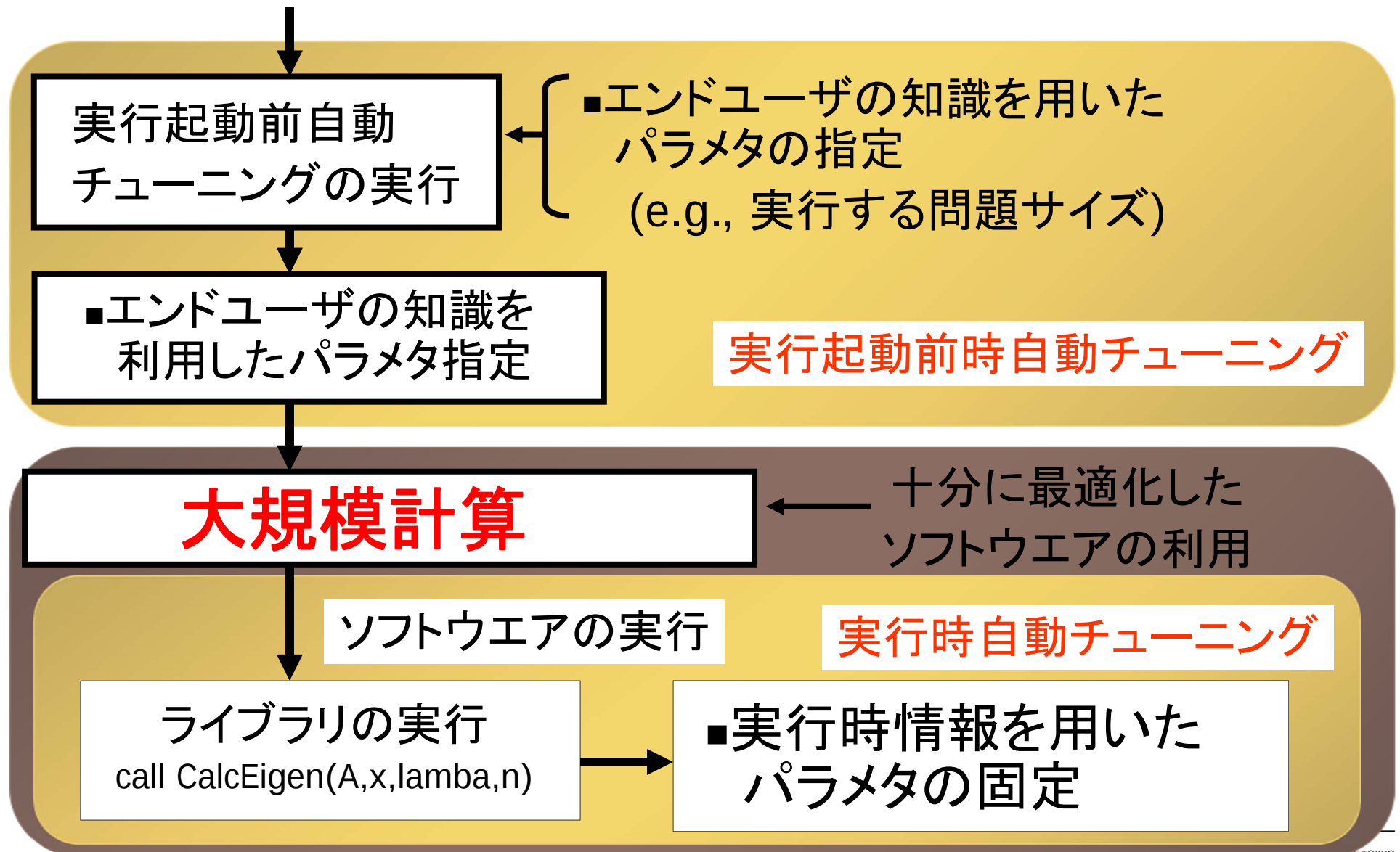
ソフトウェア開発者における F I B E R方式利用のシナリオ



エンドユーザにおける F I B E R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)

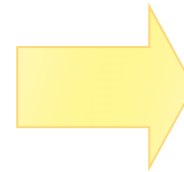


ABCLibScript設計方針

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-I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

ソフトウェ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)

ソフトウェ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

ソフトウェ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

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

```
!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*NPr
```

対象1(アルゴリズム1)

対象領域1
(チューニング領域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*NPr
```

対象2(アルゴリズム2)

対象領域2
(チューニング領域2)

```
!ABC-LIB$ select sub region end
!ABC-LIB$ static select region end
```

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

ソフトウェア開発者用ディレクティブ： ブロック幅調整

```
!ABCLib$ install variable (MB) region  
!ABCLib$ name BlkMatMal  
!ABCLib$ varied (MB) from 1 to 64  
do i=1, n, MB  
  call MyBlkMatVec(A,B,C,n,i)  
enddo  
!ABCLib$ install variable (MB) region end
```

変動パラメタの
自動チューニング指定

ブロック幅調整指定
(1から64まで)

対象領域
(AT領域)
(ソフトウェア開発者
が知っている)

ソフトウェ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

コスト定義関数で使用する
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)

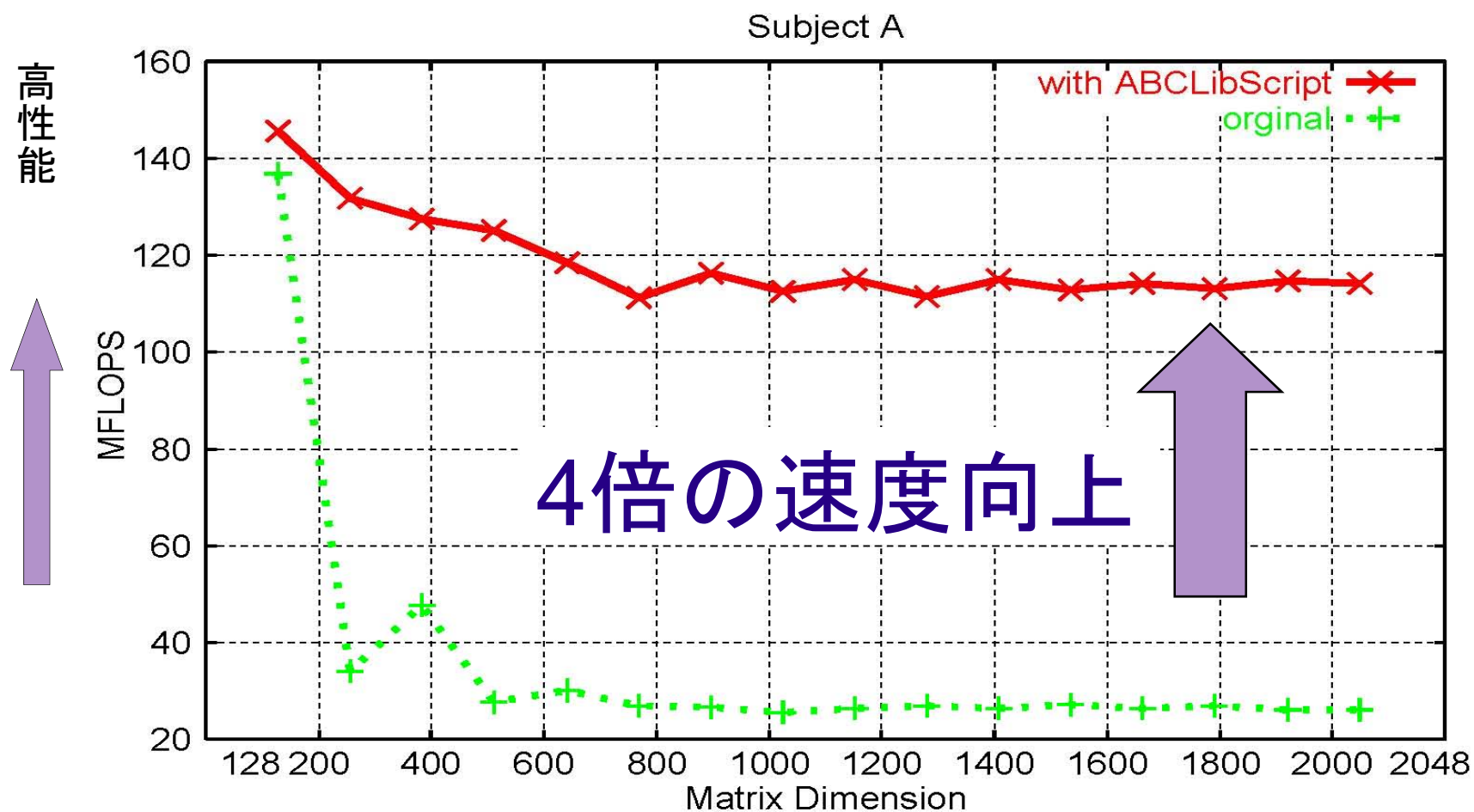
対象領域1、2
(チューニング領域)



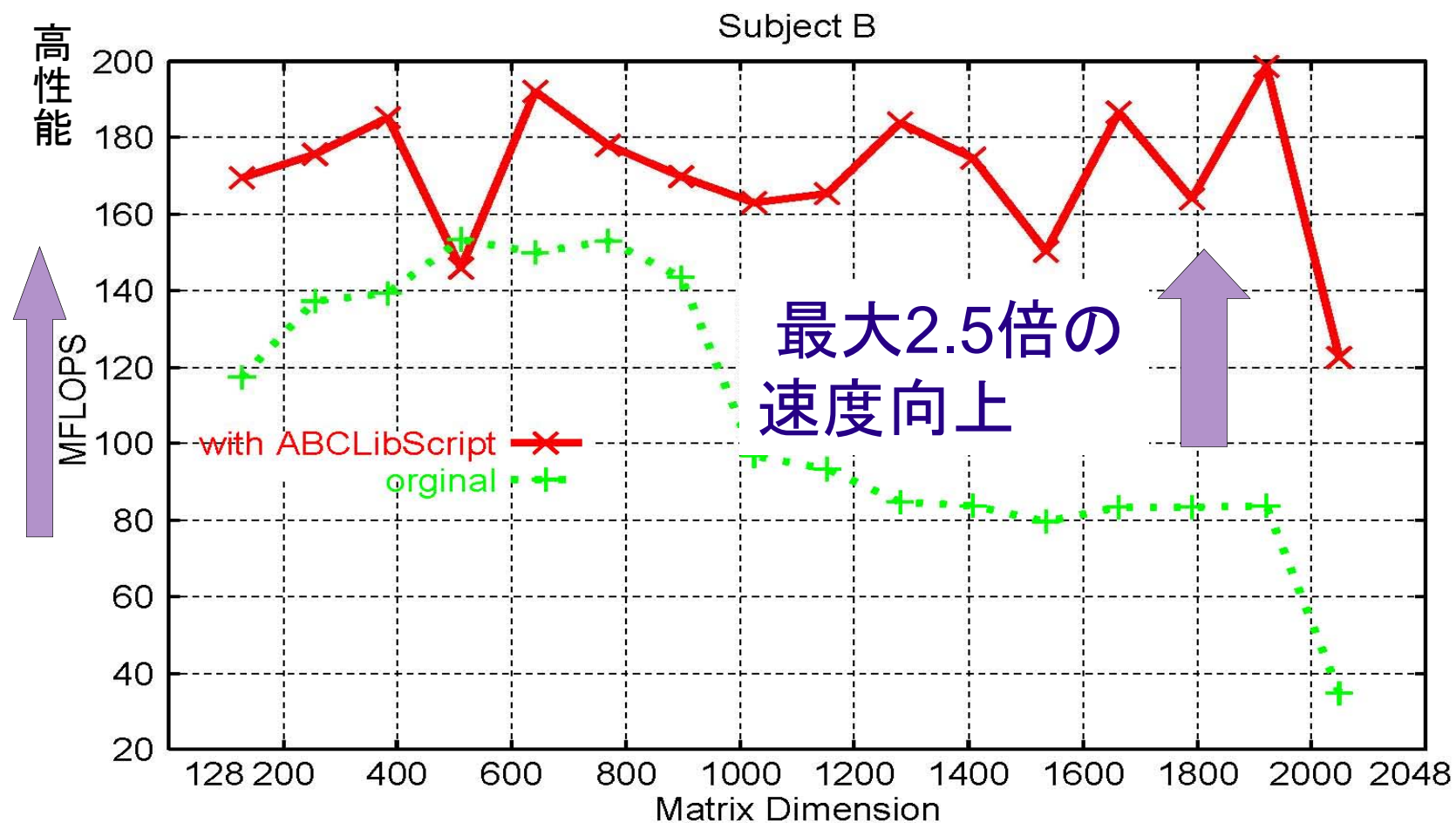
ABCLibScriptの効果検証

- ▶ 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)



実験結果 (2/2)



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

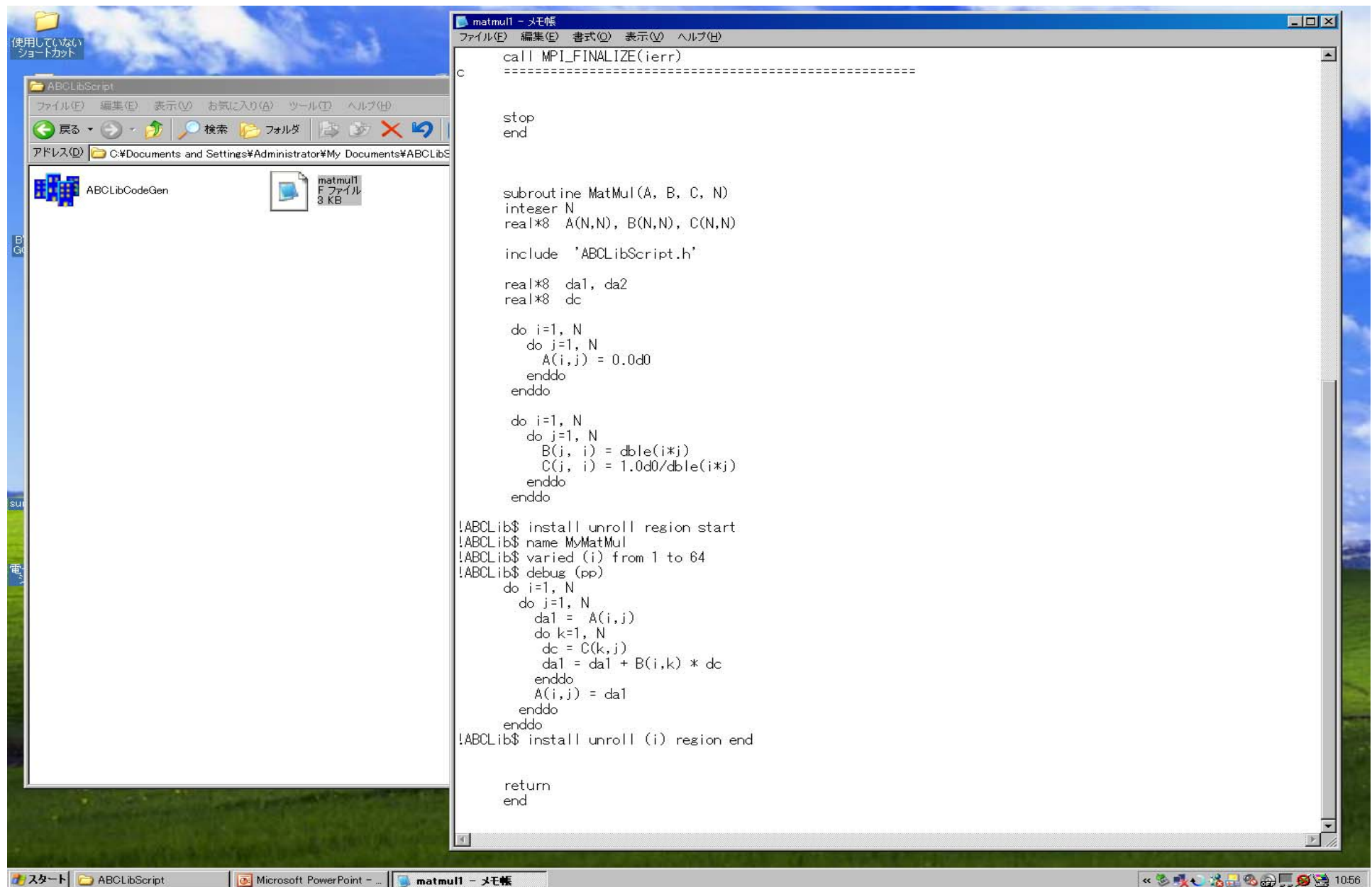
開発ソフトウェアデモ

1. ABCLibScriptのプリプロセッサ ABCLibCodeGen

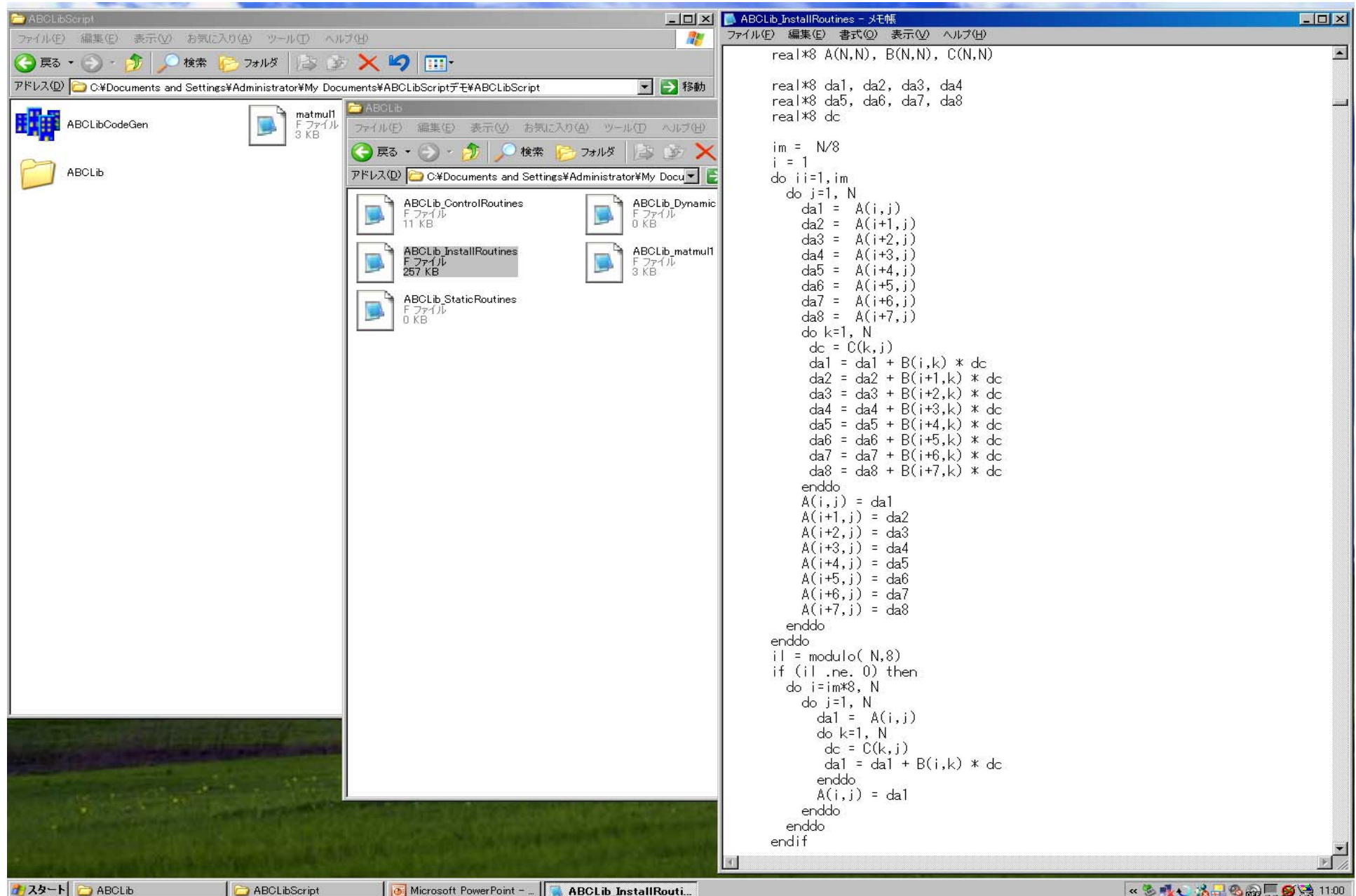
- ▶ 対象処理: 行列積コードに対する i, j, k -ループアンローリング
- ▶ 元のプログラム (ABCLibScript + Fortran90 + MPI)
- ▶ 自動チューニング機能が付加された、自動生成プログラム (Fortran90 + MPI)

2. VizABCLibによる、上記自動生成プログラムの実行ログに対するビジュアル表示

ABCLibScript 画面



ソースコード自動生成画面



VizABCLib 起動画面

The screenshot shows a Windows XP desktop environment. On the left, a file explorer window displays the contents of the 'ABCLib' folder. The files listed are:

- bar0-
- ABCLib_DynamicRoutines (F ファイル, 0 KB)
- ABCLib_matmal_fit (F ファイル, 4 KB)
- ABCLibATLog (HTML Document, 11 KB)
- MyMM_ILSM (DAT ファイル, 1 KB)
- etable-
- ABCLib_In (DAT ファイル, 1 KB)
- ABCLib_S (F ファイル, 0 KB)
- ABCLibAT (DAT ファイル, 1 KB)
- vlapplet (Executable, 164 KB)

The main window, titled 'ABCLibScript - Microsoft Internet Explorer', displays the source code for the program. The code is as follows:

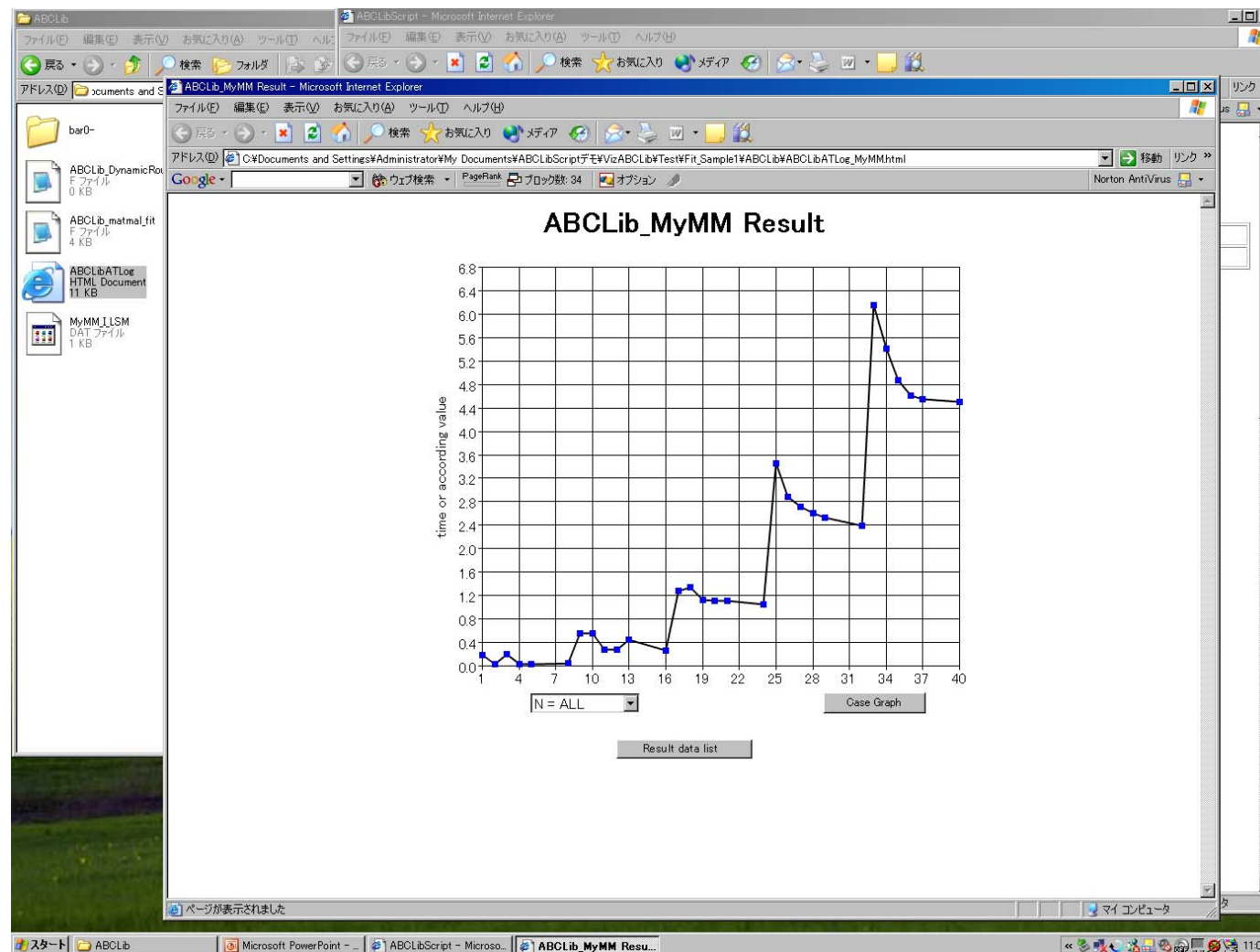
```
SrcFile = C:\MPI\2004_12_14\Test\Fit_Sample1\Source\matmal_fit.f

Auto tuning region name      Execute status      Link to result page
MyMM                        100%                        Result

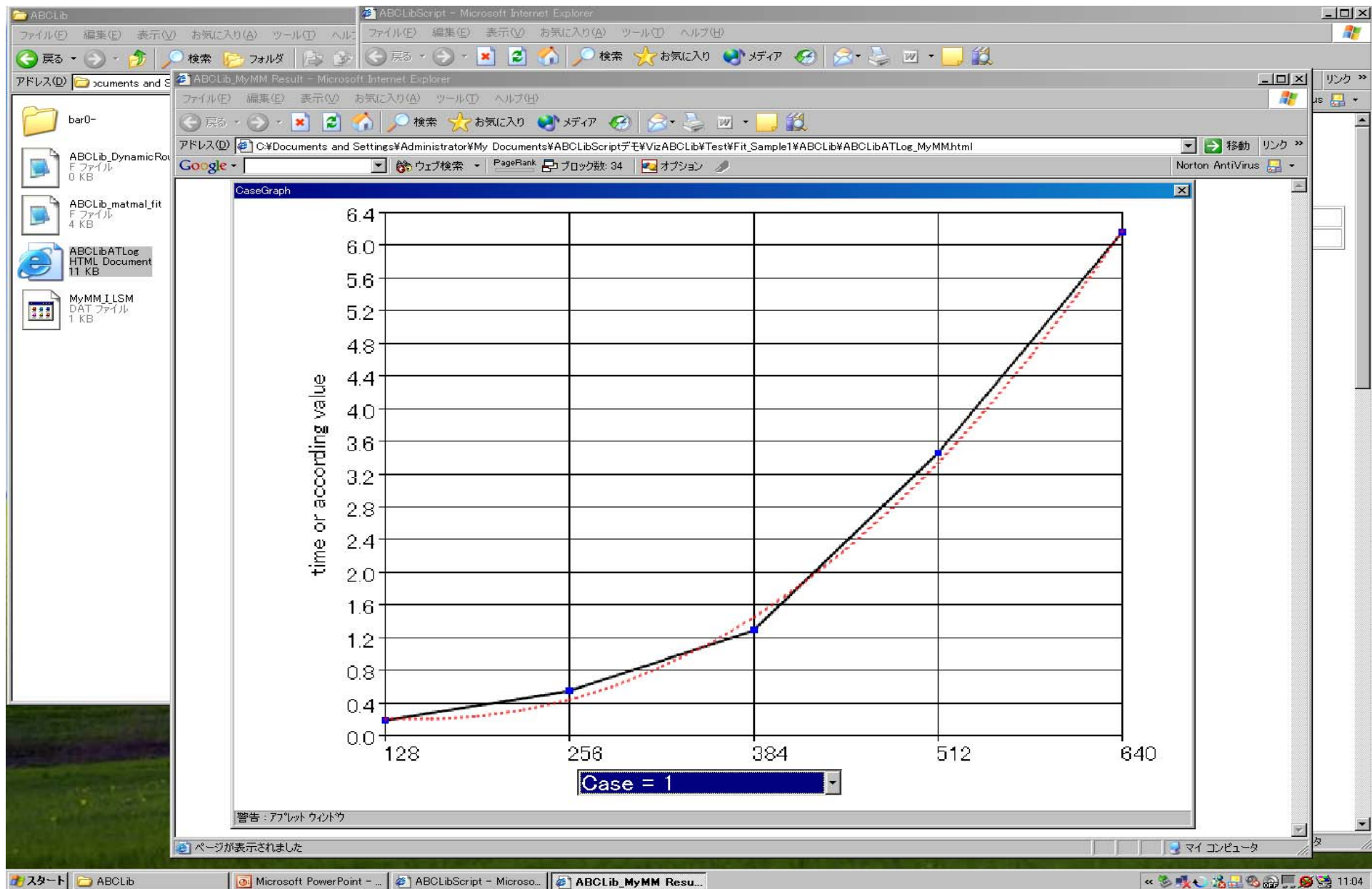
program main
include 'ABCLibScript.h'
integer iauto
integer N, NN
parameter (NN=3000)
real*8  A(NN,NN), B(NN,NN), C(NN,NN)
double precision t1, t2, t_all, bt
real*8  dtemp
c      === MPI Init.
c      =====
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, nprocs, ierr )
c      =====
call ABCLib_ATset(ABCLib_ALL, ABCLib_AllRoutines)
call ABCLib_ATset(ABCLib_INSTALL, ABCLib_InstallRoutines)
call ABCLib_ATset(ABCLib_STATIC, ABCLib_StaticRoutines)
call ABCLib_ATset(ABCLib_DYNAMIC, ABCLib_DynamicRoutines)
c      === for chacking debug mode
iauto = 1
N=128
if (iauto .eq. 1) then
  ABCLib_NUMPROCS = 4
  ABCLib_STARTTUNESIZE = 128
  ABCLib_ENDTUNESIZE = 640
  ABCLib_SAMPDIST = 128
  call ABCLib_ATexec(ABCLib_INSTALL, ABCLib_InstallRoutines)
  stop
else
  ABCLib_DEBUG = 1
endif
do N=100, 2000, 100
call Init(A,B,C,N)
call MPI_BARRIER(MPI_COMM_WORLD, ierr)
t1 = MPI_WTIMEO
c      === Please write your code in this resion.
c      =====
```

The taskbar at the bottom shows the Start button, the 'ABCLib' folder, and the 'Microsoft PowerPoint' application. The system clock in the bottom right corner indicates the time is 11:02.

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
「ソフトウェア自動チューニング
—数値計算ソフトウェアへの適用と
その可能性—」

慧文社

ISBN4-905849-18-7



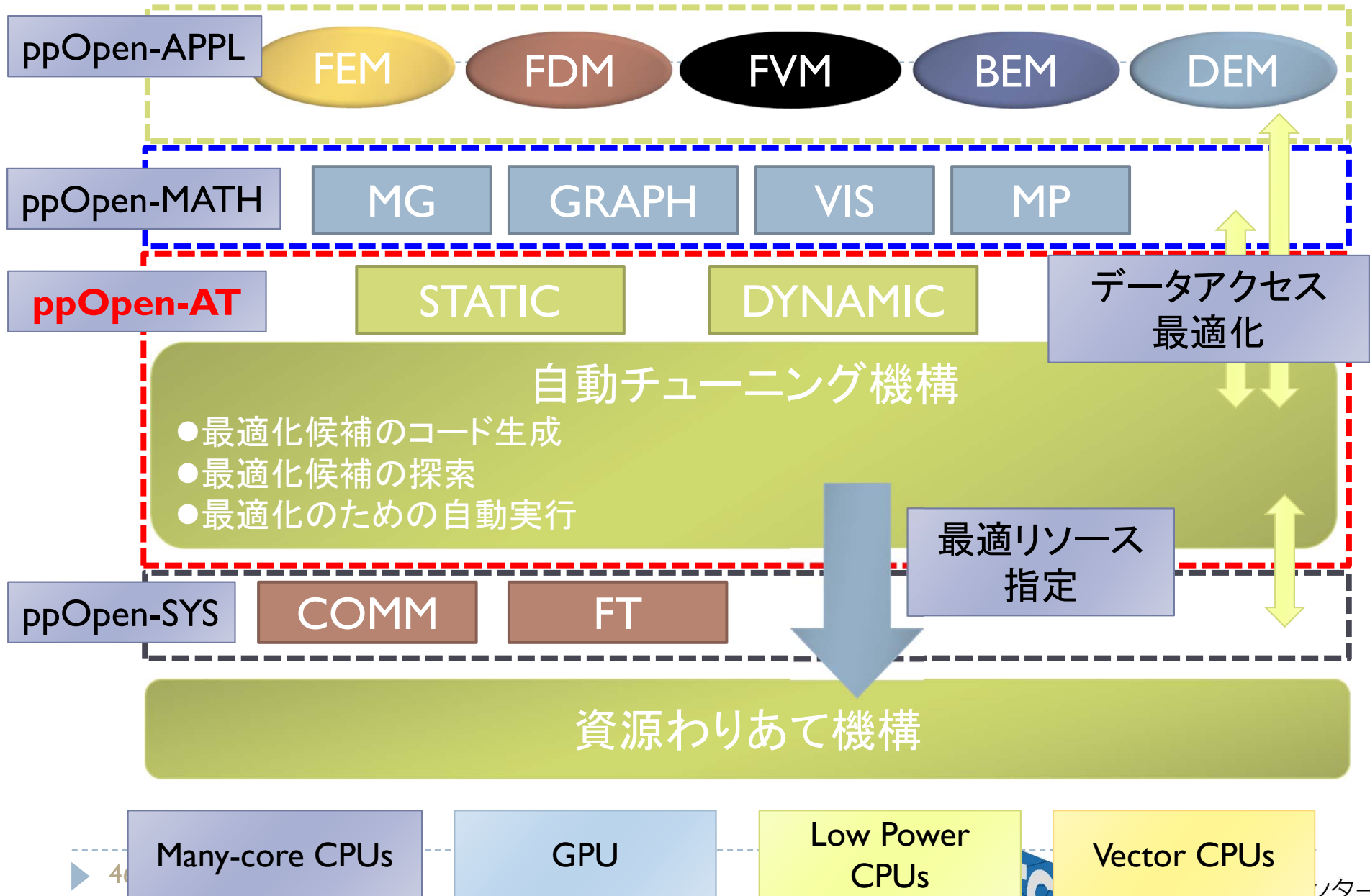
Ppopen-hpc プロジェクトと ppopen-AT

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

ユーザプログラム



科学技術計算のためのAT技術に求められること

1. 計算科学分野のアプリで「実用」となること

- 計算科学分野で使われているアプリケーションにおいて、ATの効果を出す(コ・デザインの推進)
- ベンチマークではなく、アプリの実コードを用いて性能評価する

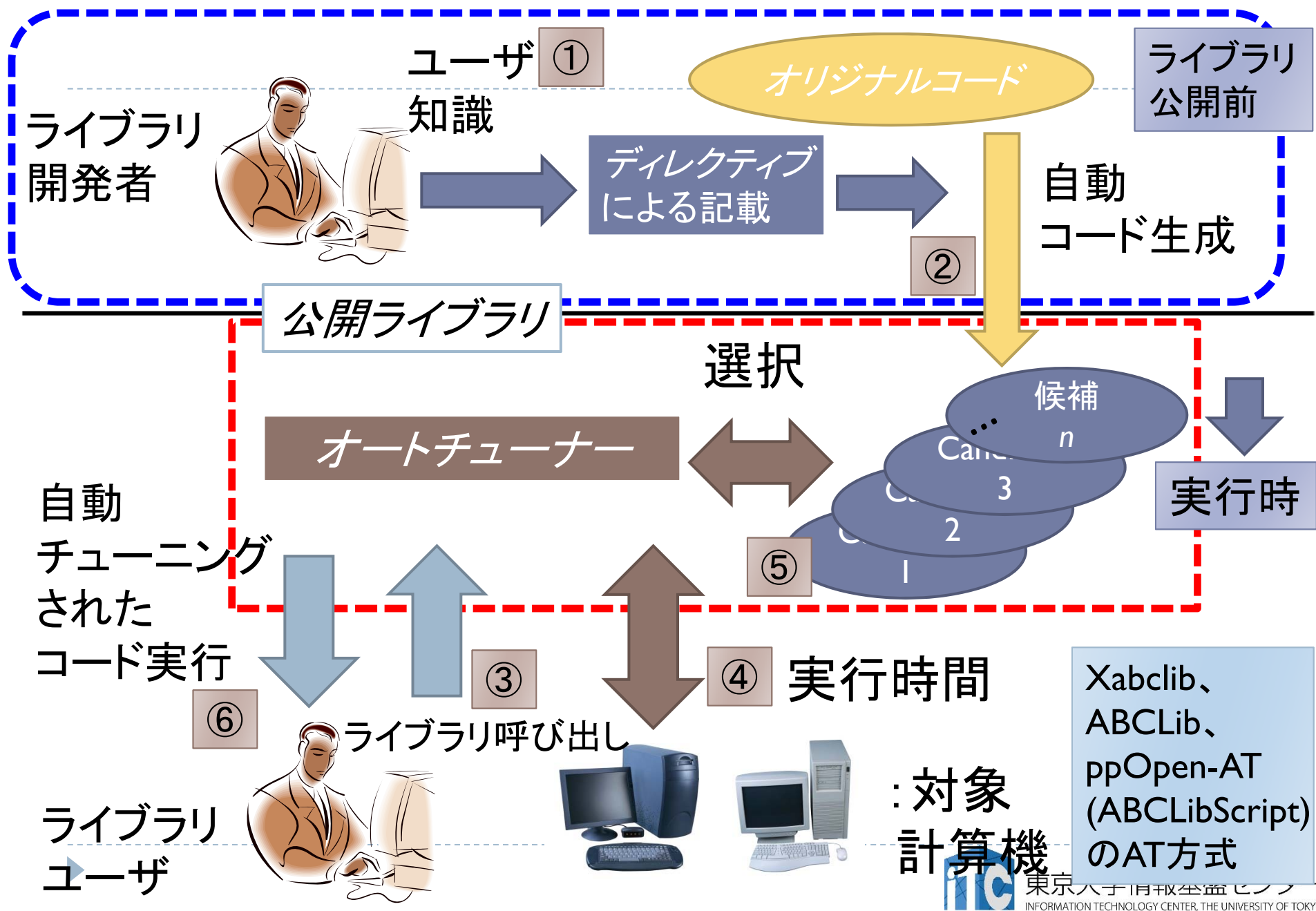
2. 運用中のスパコンでも適用できること

- スパコン運用環境でAT機能付きソフトウェアが動作
- 多数ユーザを有するスパコンセンターでも使える
 - ✓ ATのための計算機資源利用が極力少ない(低オーバーヘッド)
 - ✓ 他ユーザの利用を阻害しない

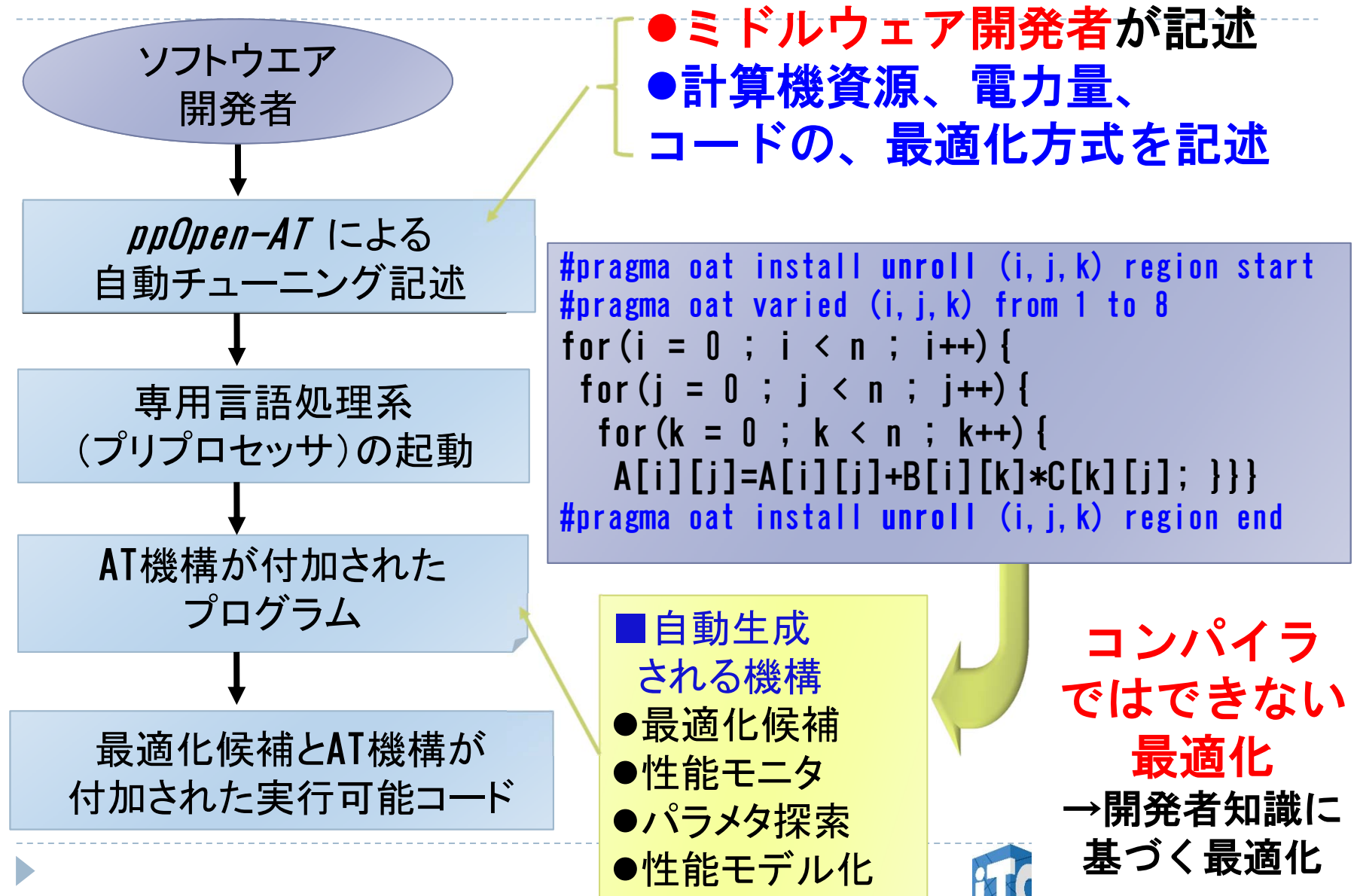
3. ソフトウェア・スタックの要求が少ないこと

- 特に、動的な「コードジェネレータ」において
 - ✓ デーモンが不要
 - ✓ OSカーネル修正不可(ベンダ保守契約の問題)
 - ✓ スクリプト言語が不要
 - ✓ バッチジョブシステムに依存しない、...など実際は問題山積

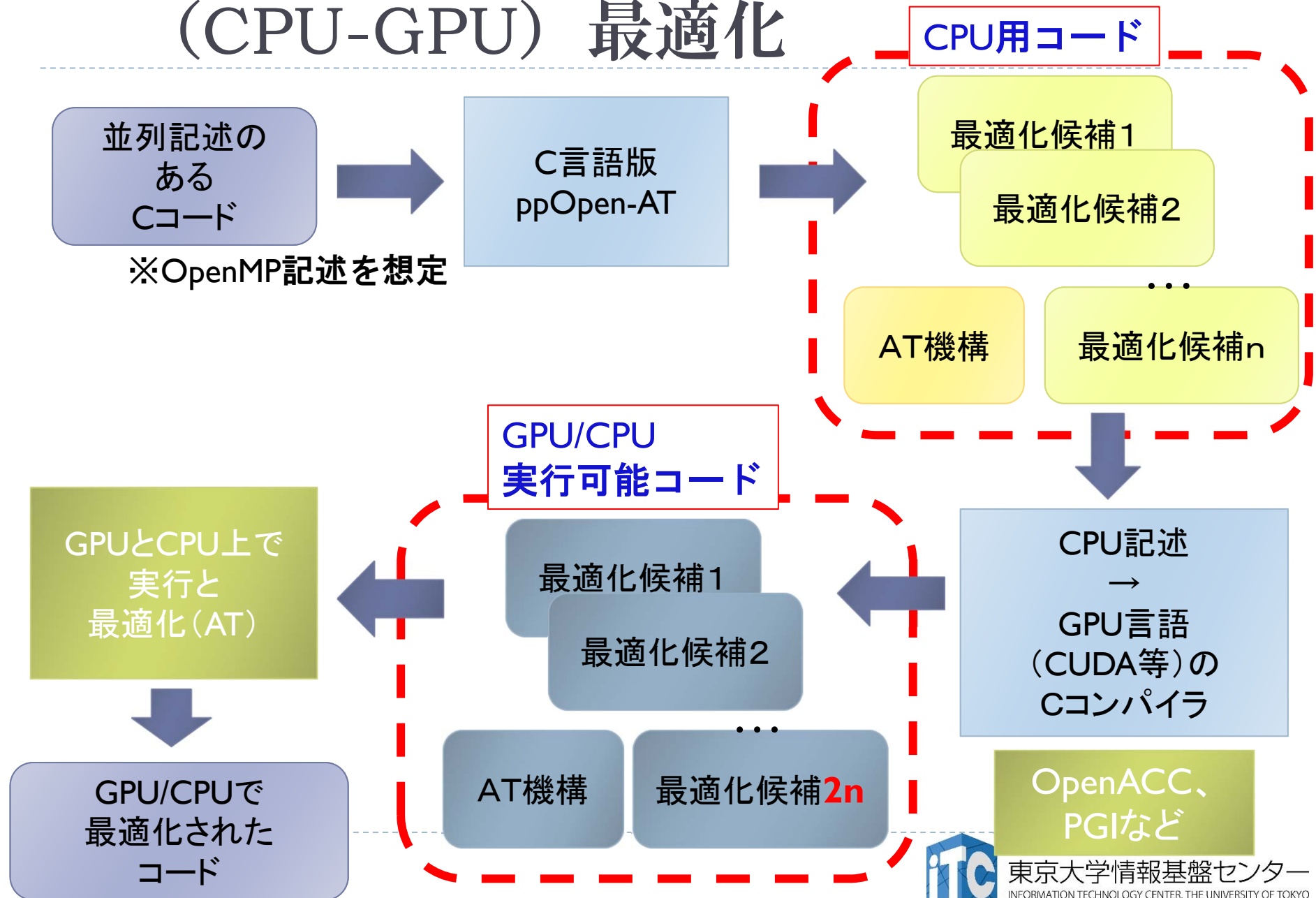
2003]



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

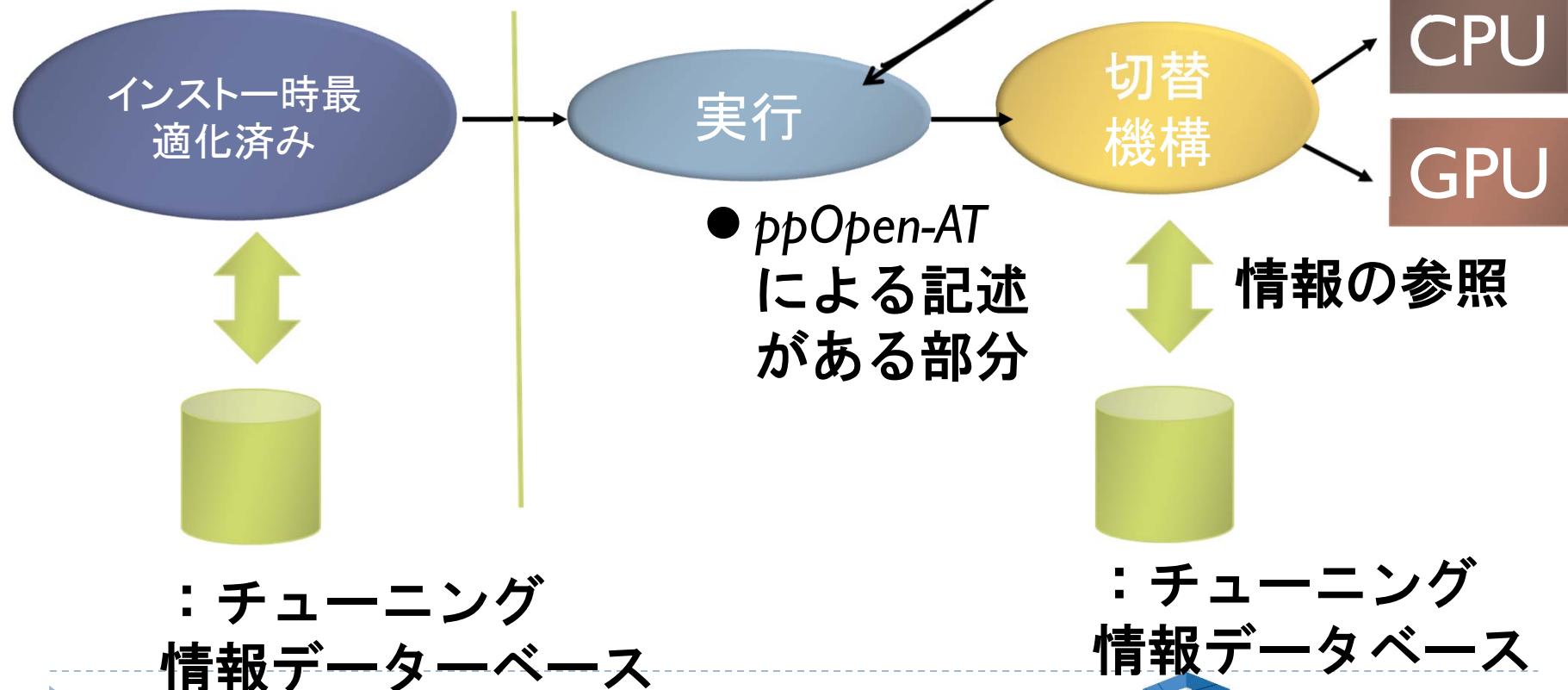


ATに関する実行時の挙動

インストール時最適化
の場合

ppOpen-ATによる
記述がある

対象関数の
ユーザによる呼び出し



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

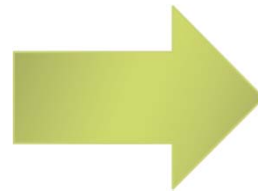
```
#pragma oat install unroll (i,j,k) region start
#pragma oat name MyMatMul
#pragma oat varied (i,j,k) from 1 to 2
#pragma oat allocate (auto)
    for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
            for(k = 0 ; k < n ; k++){
                A[i][j] = A[i][j] + B[i][k] * C[k][j];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

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

AT候補

実装切り替え部

候補 #1

候補 #2

...

候補 #n

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)



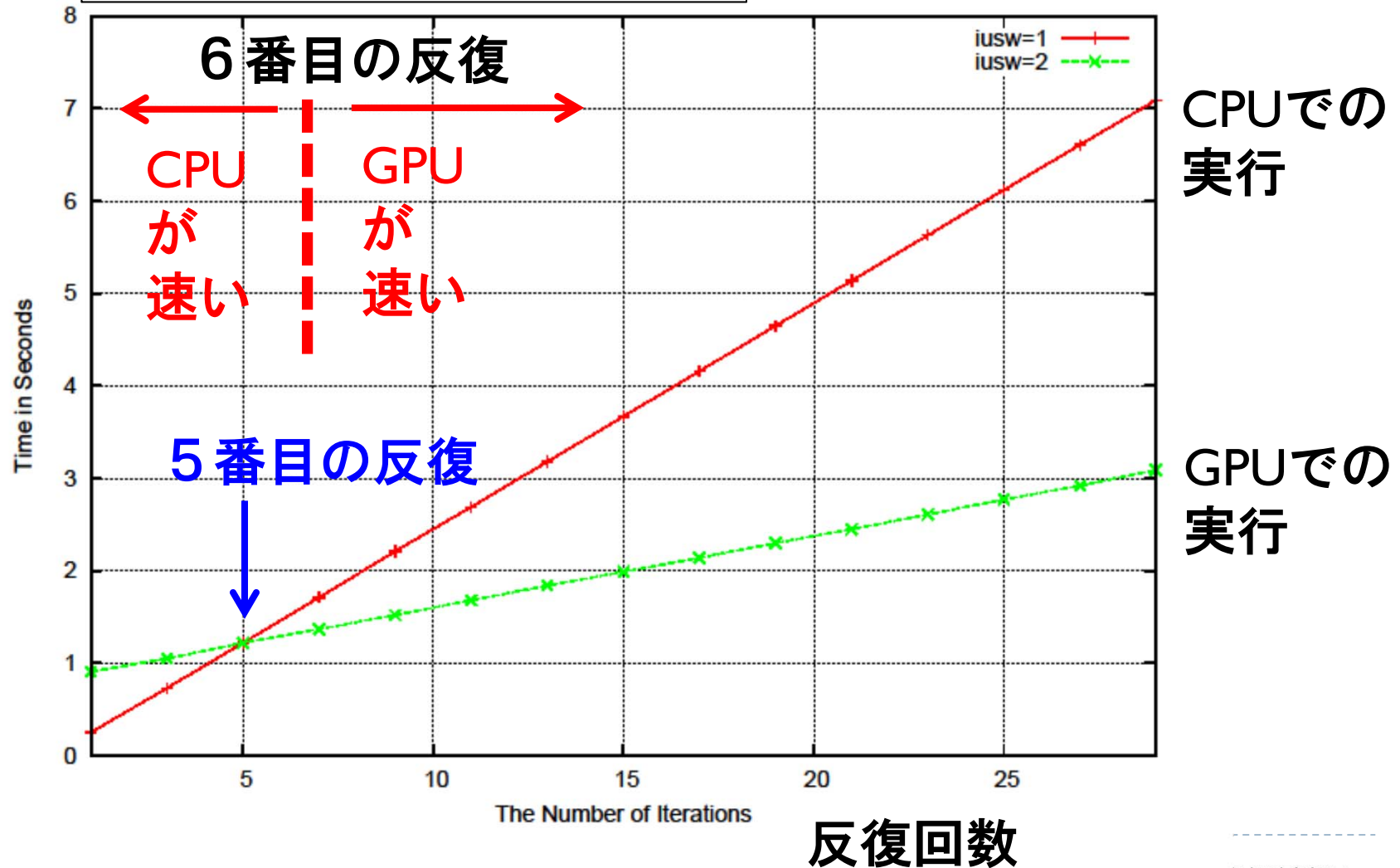
例) 姫野ベンチマークのCPUとGPU切り替え

```
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

問題サイズ : LARGE



レポート課題

- I. [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. [L40] 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. [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。

スパコンプログラミング(1)(I) おわり