疎行列反復解法ライブラリにおける自動チューニング機能の開発

Development of Auto-tuning Facility for Sparse Matrix Iterative Libraries

東京大学情報基盤センター

片桐　孝洋

RIMS研究集会

科学技術計算アルゴリズムの数理的基盤と展開

日時： 2010年10月18日(月)～21日(木)

場所： 京都大学数理解析研究所 4階 420号室

10月19日(火)　若手セッション　10:40-11:25（招待講演）

謝辞

- Xabclibプロジェクト
  - 東京大学情報基盤センター
    - 大島聡史助教、伊藤祥司特任准教授、黒田久泰准教授（兼任、本務愛媛大学）、中島研吾教授
  - 日立製作所 中央研究所
    - 櫻井隆雄氏、猪貝光祥氏、直野健博士

  に感謝いたします。
- d-splineによる標本点抽出法の資料について、日立製作所 田中輝雄 博士 にご提供いただきました。感謝いたします。

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- 第二部：疎行列反復解法ライブラリへの適用事例
  - 疎行列反復解法適用への問題点
  - 開発事例：OpenATLib とXabclib
  - 性能評価
- まとめ

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- 第二部：疎行列反復解法ライブラリへの適用事例
  - 疎行列反復解法適用への問題点
  - 開発事例：OpenATLib とXabclib
  - 性能評価
- まとめ

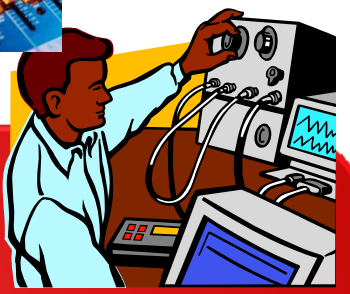
自動チューニング機構とは

•計算機アーキテクチャ
•計算機システム

•プログラム
•アルゴリズム

性能調整つまみ
（性能パラメタ）

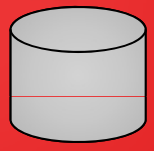
調整機構
•最適化
•パラメタ探索
•学習／自己適応



性能モニタ機構

•プログラム
•アルゴリズム

つまみ自動生成機構



性能蓄積機構
性能データベース

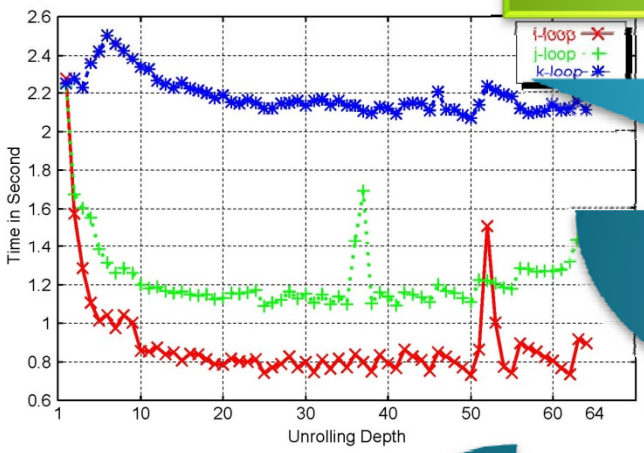
自動チューニング機構

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

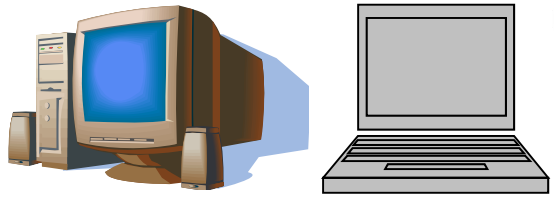
```
do i=1, n, 2
  do j=1, n
    Ctmp1 = C( i,   j )
    Ctmp2 = C( i+1, j )
    do k=1, n, 2
      Btmp1 = B( k,   j )
      Btmp2 = B( k+1, j )
      Ctmp1 = Ctmp1 + A( i,   k   ) * Btmp1
                    + A( i,  k+1) * Btmp2
        Ctmp2 = Ctmp2 + A( i+1, k   ) * Btmp1
                    + A( i+1, k+1) * Btmp2
    enddo
    C( i,   j )=Ctmp1
    C( i+1, j )=Ctmp2
  enddo
enddo
```

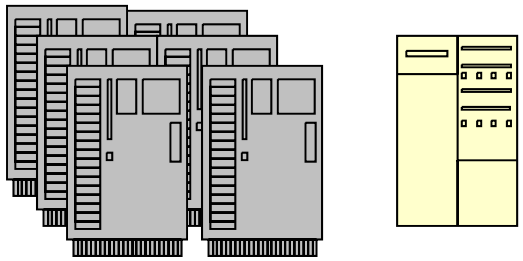
コンパイルと実行



3. 最適化フェーズ

実行結果の解析



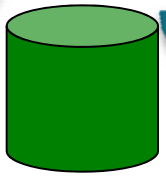


4. データベース化とチューニング知識探索フェーズ

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
```

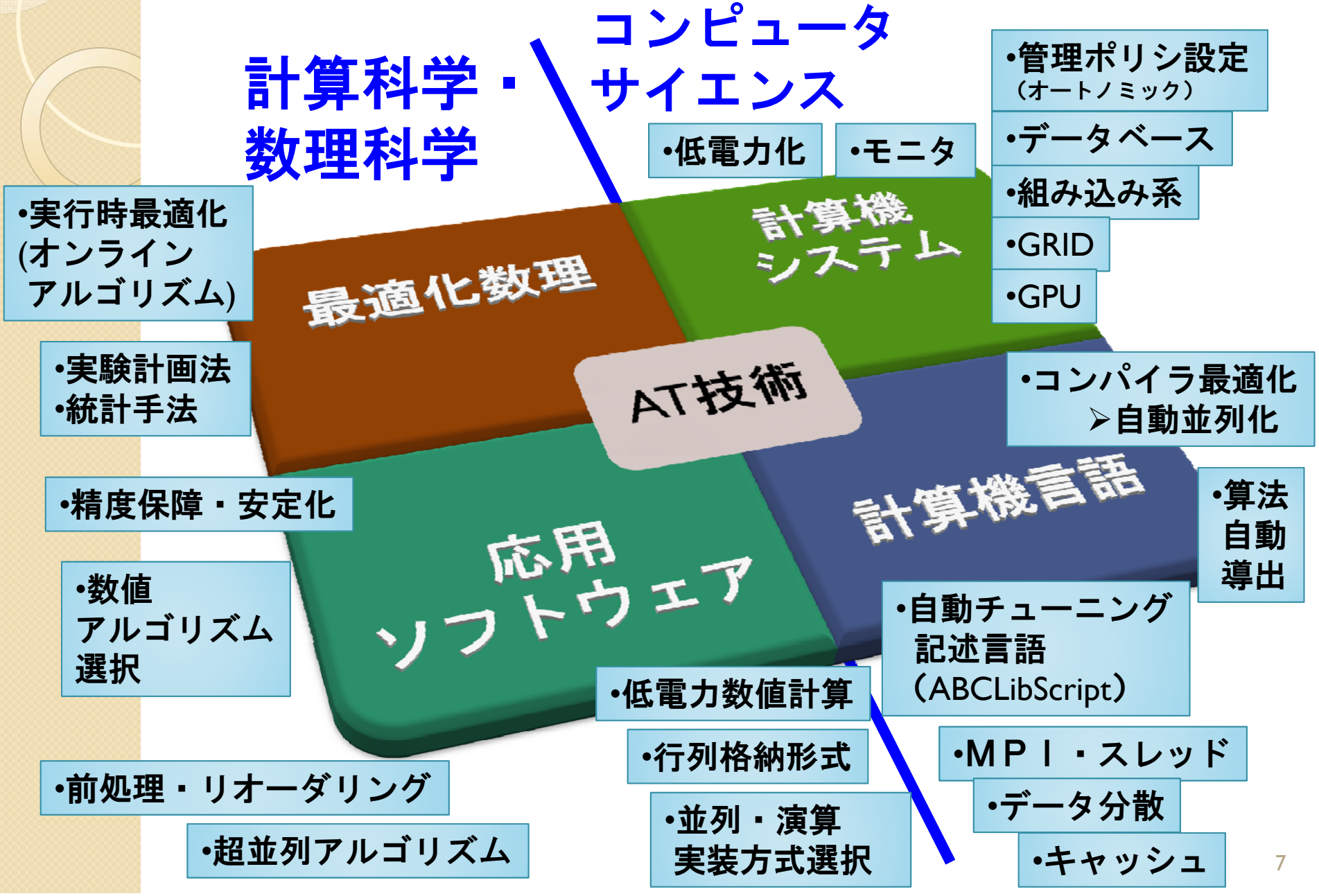
対象計算機



チューニング知識データベース

1. 仕様策定フェーズ

自動チューニング技術の鳥瞰図



自動チューニング（AT）とは

- i. プログラム上の対象個所を決定する。
- ii. 対象個所中において、性能に影響を及ぼす
  パラメタ集合X を抽出する。
- iii. チューニングする評価基準となる
  関数F をパラメタ集合X で定義する。
- iv. 関数F ： X →Y を最小とするパラメタ集合X の
  値を、プログラムを実行しながら見つける。
  （これを、解パラメタ集合X^* とする）
- v. 解パラメタ集合X^* でプログラムを実行する。

i.~iv.が全自動になっていることが望ましいが、
一部のみ自動になっていること（半自動）も、
自動チューニングと呼ぶ。

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- 第二部：疎行列反復解法ライブラリへの適用事例
  - 疎行列反復解法適用への問題点
  - 開発事例：OpenATLib とXabclib
  - 性能評価
- まとめ

性能パラメタ変数と取りうる範囲

- プログラム上の箇所iの性能にかかわる変数（**性能パラメタ変数**）を x_i とする。
- x_i の取りうる範囲は

$$x_i \in [IS_{x_i}, IE_{x_i}]$$

である。$[IS_{x_i}, IE_{x_i}]$ を**定義域**と呼ぶ。

- プログラム上のすべての性能パラメタ変数について、**何らかの操作**により値を固定し、その値を元$\overline{x_i}$として構成する集合Xを

$$X = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_m}\}$$

を**性能パラメタ集合**とよぶ。

コスト定義関数

- 最適化する対象の挙動を定義する関数

$$F : X \rightarrow Z$$

  - 出力集合Zについて、
    - 多くは、
      対象プログラムの実行時間
    - メモリ量、演算精度、電力量なども考えられる

パラメタ変数の種別

- **基本パラメタ** *(BP)*
  - 数値計算ライブラリが提供する手続き上に現れる引数（たとえば、問題サイズ）や、計算機システムで必要となる資源量（たとえば、実行するCPU数）などの性能パラメタ変数
  - 対象が実行される前に固定される変数
- **性能パラメタ** *(PP)*
  - BPを固定した時、全体の性能に影響する性能パラメタ変数
  - ユーザによるPPの設定を必要としないが、数値計算ライブラリ上の性能を決定する変数

チューニングの定義

- **条件付き最適化問題**([直野ら,1999])

$$\min \; F(PP),$$

$$s.t.\, BP = \{\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_n\}$$

ここで

$$X = PP \cup BP,$$

$$PP \cap BP = \phi$$

FIBERによるATのタイミング

- FIBER[Katagiriら, 2003]による
  自動チューニングのタイミング

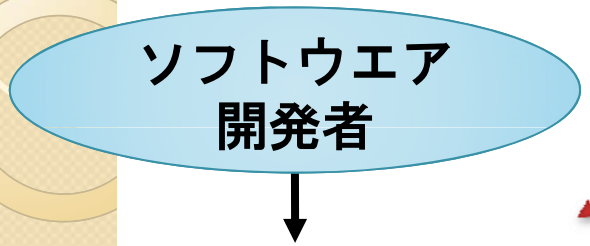
$$X = IP \cup BEP \cup RP$$

IP	BEP	RP
インストール時 自動チューニング	実行起動前 自動チューニング	実行時 自動チューニング

- 自動チューニングの適用順序
  1. インストール時
  2. 実行起動前時
  3. 実行時

自動チューニング専用言語（任意の箇所の性能パラメタ変数の自動抽出）

ＡＴ専用言語による自動チューニング ソフトウェア開発手順



●ソフトウエア開発者が記述
●計算機環境に依存しない

ABCLibScriptによる
自動チューニング記述

```
!ABCLib$ install unroll region start
!ABCLib$ varied (i,j,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 region end
```

専用言語処理系
（プリプロセッサ）の起動

自動チューニング機能が
付加されたプログラム

■自動付加される機構
●対象蓄積
●コスト定義関数
●モニタ
●学習
●パラメタ推定

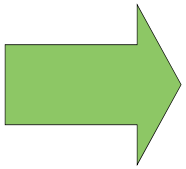
コンパイラ
ではできない
自動最適化
手法

数値計算ライブラリ
（ソースコード）

プリプロセッサの処理

ディレクティブで記述

```
!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**

■最適化
■実測とモデル化

**AT領域選択
コンポーネント**

■実行時最適化
■最適化済みパラメタ設定

**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

インストール時自動チューニング指定：行列積コードのループアンローリング

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

インストール時自動チューニング
アンロール処理の指定

- 性能パラメタ変数の指定（ループ変数i）
- 最大アンローリング段数の指定（定義域[1,8]）

対象領域（ＡＴ領域）

$i \in [1,8]$

$PP = IP = \{i\}$

$BP = \{N\}$

F ：AT 領域の実測時間

行列積コードのループアンローリング（続き）

- プリプロセッサ実行後、最外側の *i* ループがアンロールされ、AT領域ライブラリコンポーネントに自動登録

```
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
```

アンローリング段数が自動的にパラメタ化（PP）される

インストール時自動チューニング指定：ブロック幅調整

```
!ABCLib$ install variable (MB) region start
!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
（性能パラメタMB）

ブロック幅調整指定
（MBの定義域[1,64]）

対象領域
（ＡＴ領域）

$MB \in [1,64]$

$PP = IP = \{MB\}$

$BP = \{n\}$

F：AT 領域の実測時間

実行起動前自動チューニング指定：アルゴリズム選択

実行起動前自動チューニング指定、
アルゴリズム選択処理の指定

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

性能パラメタ変数の定義

コスト定義関数

対象領域1、2
(AT領域1、2)

対象1 と 2 の選択情報がパラメタ化される

実行起動前自動チューニング指定：アルゴリズム選択

- 定式化

$$PP = BEP = \{CacheS, NB, \mathrm{NPrc}\}$$

$$BP = \{\}$$

$$\min\{F_1(PP), F_2(PP)\}$$

$$F_1(PP) = (2CacheS \times NB)/(3\mathrm{NPrc})$$

$$F_2(PP) = (4CacheS \times NB)/(2\mathrm{NPrc})$$

定義域を縮小させる条件

- **定義域の縮小条件*P***

$$L = \{ l_i \in [IS_{x_i}, IE_{x_i}] \mid P \}$$

$$(i = 1,..., m),$$

$$L \subset PP$$

●ATソフトウェア構築の鍵

●多くは、処理の特性を経験的に抽出し
条件Pを決定（例：BLASのDGEMMカーネル）

●特性を抽出できない場合、全探索

$$X \times X$$

の範囲をすべて調べる（調べられる範囲の定義域を経験的に定める）

関連研究

- 条件*P*の定義に関しての研究
  - ベイズ統計により試行回数を削減する方式 [須田ら，2009]
  - 数値ライブラリの階層性を利用し、下位階層の演算部分を小規模行列の実測によりモデル化し、大規模行列の実行時間を動的計画法で推定する方式 [深谷ら，2009]
  - 関連研究１：サンプリング的の実行時間の多項式近似法FIBER[片桐ら，2003]
  - 関連研究２：サンプリング点のd-splineによる動的追加法[田中ら，2007]

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- 第二部：疎行列反復解法ライブラリへの適用事例
  - 疎行列反復解法適用への問題点
  - 開発事例：OpenATLib とXabclib
  - 性能評価
- まとめ

関連研究１：
サンプリング点導入による定義域の縮小
（ＦＩＢＥＲ方式）

Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba, Springer LNCS 2858, pp.146--159, The Fifth International Symposium on High Performance Computing (ISHPC-V), October 20-22, 2003: "FIBER: A General Framework for Auto-Tuning Software"

適用例：並列密対称固有値ソルバのインタフェース

call CalEigen(A, x, lambda, n, ←基本パラメタＢＰ

Coefficient Matrix, Eigenvectors, Eigenvalues, Matrix Size

nprocs, myid, IDIST, ←基本パラメタＢＰ

Number of Processors, Processor ID, Data Distribution Information

imv, iud, ihit, icomm, kbi, kopt, ←性能パラメタＰＰ

Matrix-Vector Product, Matrix Updating, Householder Inverse, Communication Method, Bisection Method Implementation, Re-Orthogonalization Method

isepternum, imaxiter, deps) ←性能パラメタＰＰ

Iteration Number for Separated Eigenvalue Max Iteration Number for Eigenvectors, Machine Epsilon

ＡＴの適用

call CalEigen (A, x, lambda, n)

固有値ソルバの性能パラメタ変数

- **性能パラメタ集合PPとBP**
  - *PP* = { *imv*, *iud*, *icomm*, *ihit*, *kbi*, *kort* }
  - *BP* = { *n*, *nprocs* }
- **各処理の対応**
  - Householder**三重対角化**:
    PP = { *imv*, *iud*, *icomm* }
  - **二分法**: *PP* = { *kbi* }
  - **逆反復** *PP* = { *kort* }
  - Householder**逆変換**: *PP* = { *ihit* }

サンプリング点（標本点）とは

- 性能パラメタ*PP*において、実測回数を減らす目的で、定義域の中から選んだ観測点の集合
- 基本パラメタ*BP*において、固定回数を減らす目的で、定義域の中から選んだ観測点の集合
- コスト定義関数の形状に依存し、うまく選ぶ必要がある
  - 解の品質に影響する
  - AT実行時間　と　解の品質　のトレードオフ
  - 計算機ハードウエア依存となる

パラメタ変数の定義域と対象

- 性能パラメタ（ＰＰ）の定義域
  - *imv* ∈ {1,2,…,16} :
    Householder三重対角化の行列ーベクトル積(BLAS2)のアンローリング段数
  - *iud* ∈ {1,2,…,16} :
    Householder三重対角化の行列更新部分(BLAS2)のアンローリング段数
  - *kbi* ∈ {vec, non-vec} :
    二分法における、ベクトル計算機向き実装かスカラ計算機向き実装選択
  - *kort* ∈ {MG-S, CG-S, IRCG-S, NoOrt} :
    逆反復法における直交化アルゴリズム選択
  - *ihit* ∈ {1,2,…,16} : Householder 逆変換処理(BLAS1)のアンローリング段数

実験でのAT詳細とサンプリング点

- AT詳細
  - インストール時ＡＴ
  - 性能パラメタ *iud* :
    Householder三重対角化における
    行列更新のアンローリング段数
- 基本パラメタ（*BP* ）
  - 利用ＣＰＵ数：*nprocs* = 8
  - 行列サイズ
    $n \in \{200, 400, 800, 2000, 4000, 8000\}$
- 性能パラメタ(*PP*)
  - $iud \in \{1, 2, 3, 4, 8, 16\}$

固有値ソルバにおける性能パラメタ抽出例：行列更新カーネル(Householder三重対角化)

```
!ABCLib$  install  unroll ( j ) region start
!ABCLib$  varied ( j ) from 1 to 16
!ABCLib$  fitting polynomial 5 sampled (1-4,8,16)
   do j=0, local_length_y-1
     tmpu1 = u_x(j)
     tmpr1 = mu * tmpu1 - y_k(j)
     do i=0, local_length_x-1
          A(i_x+i, i_y+j) = A(i_x+i, i_y+j)
               + u_y(i)*tmpr1 - x_k(i)*tmpu1
     enddo  enddo  enddo
!ABCLIB$  install  unroll ( j )  region  end
```

コスト定義関数と実行時間推定法

1. コスト定期関数 F を実行時間とする。
   BPであるnを固定し、変数iudの関数 F_n を
   k次多項式で近似する。

$$f_n(iud) = a_1 \cdot iud^{k} + a_2 \cdot iud^{k-1} + a_3 \cdot iud^{k-2} + \cdots + a_k \cdot iud + a_{k+1}$$

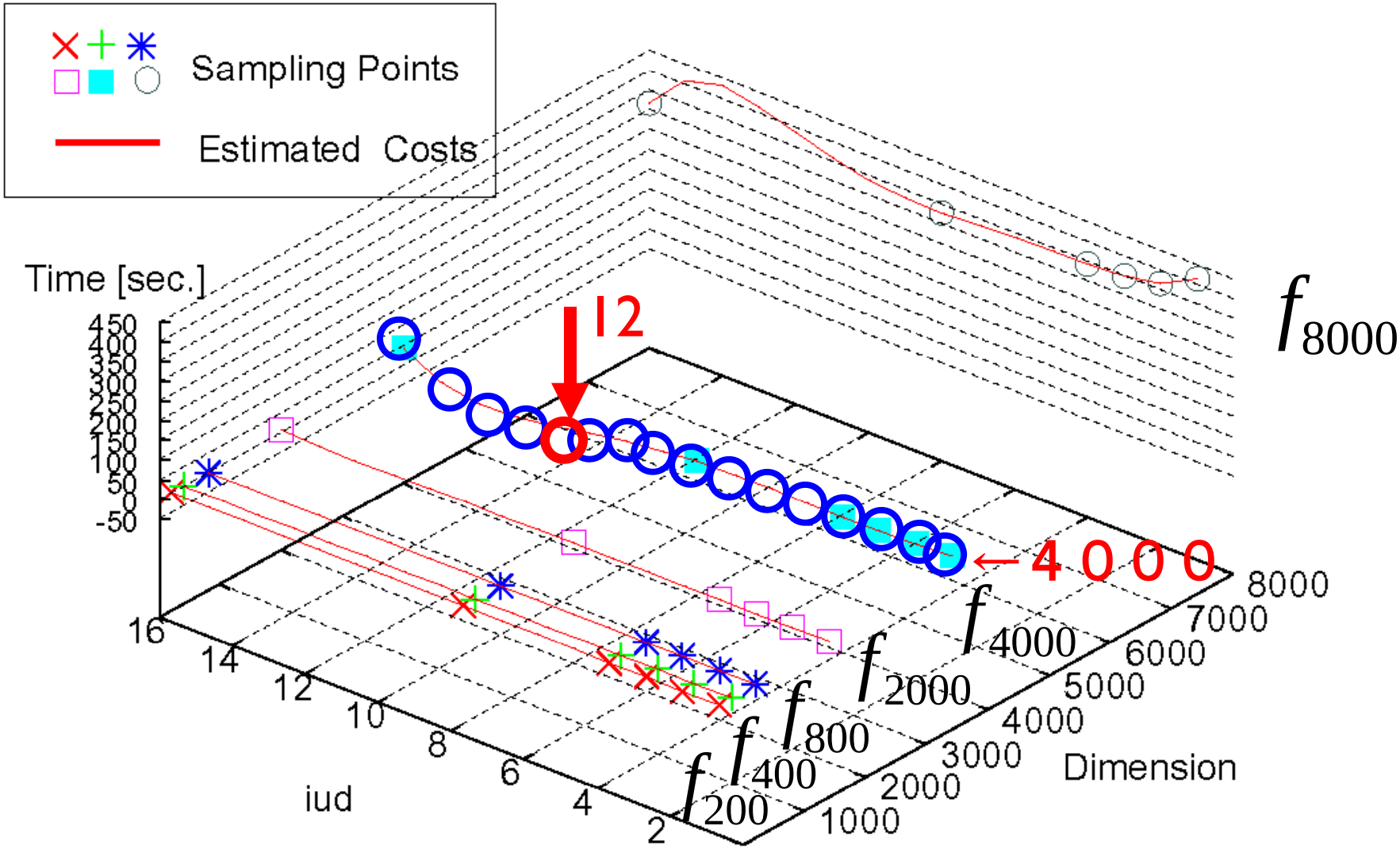
2. 係数a_1 からa_k をiudのサンプリング点の
   測定値をもとに、最小二乗法で推定する。

3. 係数の決定後，最適となる変数iudの値を
   関数Fの値（予測実行速度）をもとに、
   実行時に決定する。

推定された係数の値（5次近似の例）
(計算機：HITACHI SR8000/MPP)

f_n / a_i	a_1	a_2	a_3	a_4	a_5	a_6
f_{200}	-2.2e-6	6.7e-5	-6.5e-4	2.4e-3	-3.8e-3	6.4e-2
f_{400}	-1.3e-6	4.2e-5	-4.6e-4	2.3e-3	-7.8e-3	1.8e-1
f_{800}	9.6e-6	-2.3e-4	7.9e-4	1.1e-2	-8.4e-2	8.1e-1
f_{2000}	2.4e-4	-6.3e-3	3.5e-2	1.1e-1	-1.3	8.6
f_{4000}	3.0e-3	-8.3e-2	6.1e-1	-4.9e-1	-7.7	6.1e+1
f_{8000}	-1.3e-2	5.0e-1	-7.0	4.5e+1	-1.4e+2	5.1e+2

コスト定義関数の形状：iudの定義域からのサンプリング点　(HITACHI SR8000/MPP)



FIBER推定法

問題

- パラメタ変数 *iud* の値しか見積もれない
- エンドユーザが、事前にATシステムが与えたサンプリング点 {200, 400, 800, 2000, 4000, 8000} を行列サイズ*n*に与えた時しか使えない。

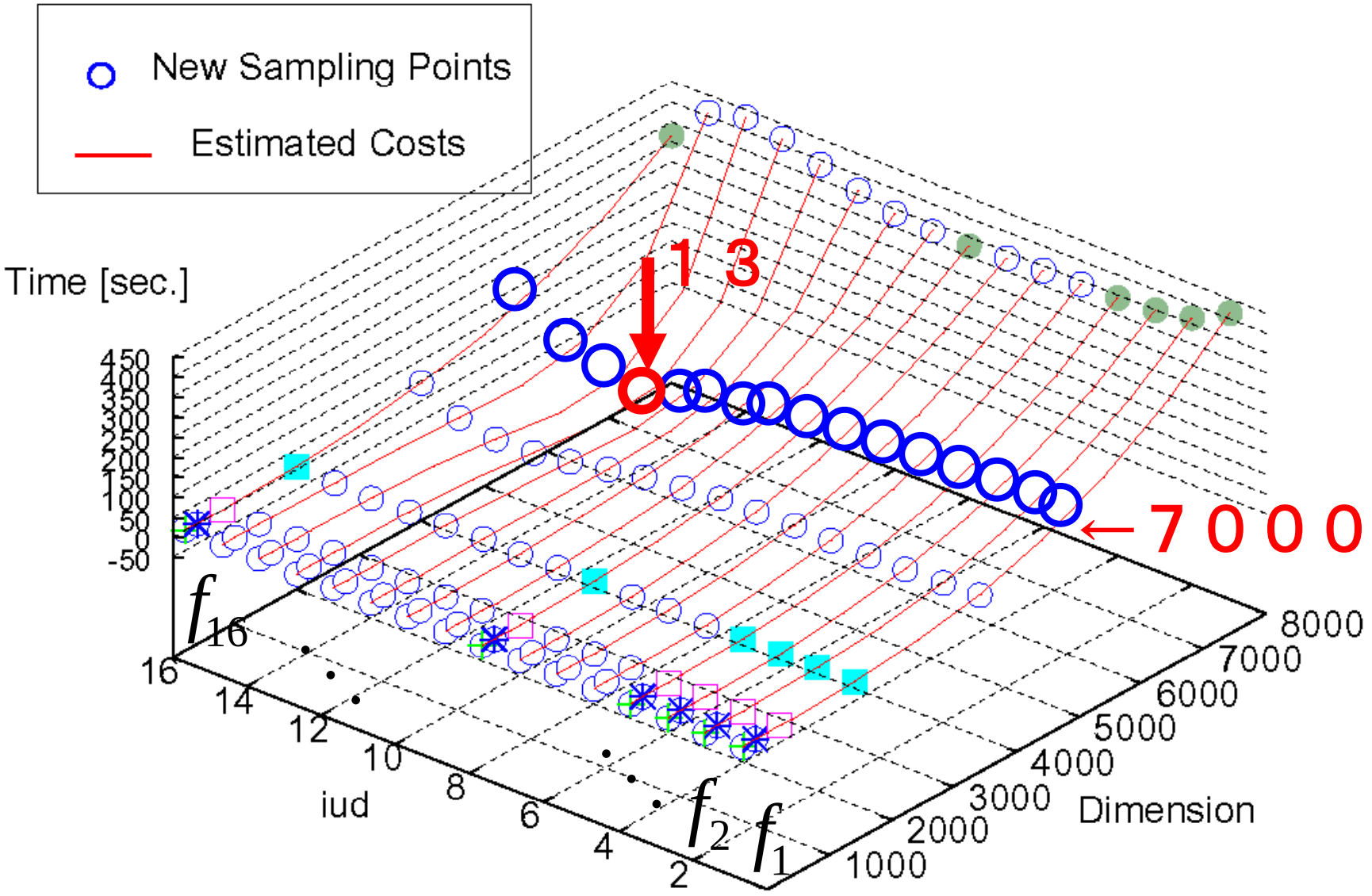
●FIBER推定法

4. *iud*を固定し、*n* に関するコスト定義関数 F_{iud} もk次の多項式で近似する。以下5次多項式の例

$$f_{iud}(n) = \overline{a}_1 n^5 + \overline{a}_2 n^4 + \overline{a}_3 n^3 + \overline{a}_4 n^2 + \overline{a}_5 n + \overline{a}_6$$

5. *BP*のサンプリング点、ここでは、*n* = {200, 400,800, 2000, 4000, 8000} についてiudの**実測値と推定値を新サンプリング点として**採用し、最小二乗法で係数 $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_6$ を決定

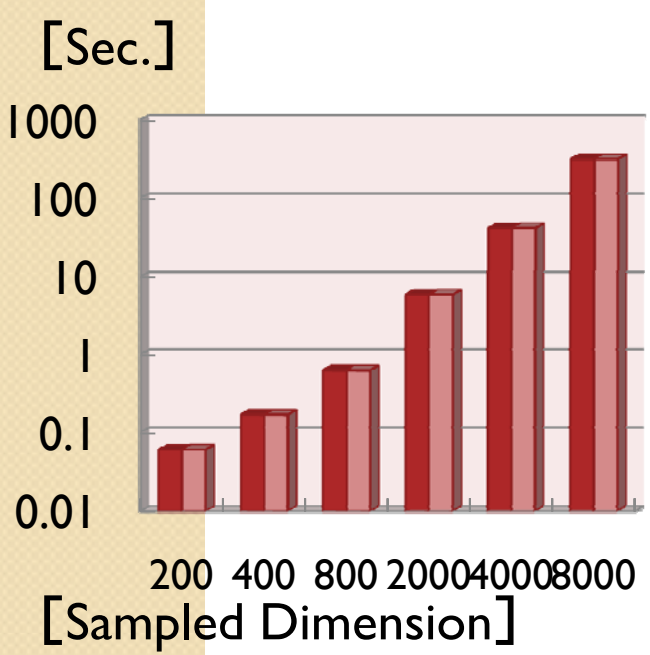
行列次元自由なコスト定義関数の形状
(The HITACHI SR8000/MPP)



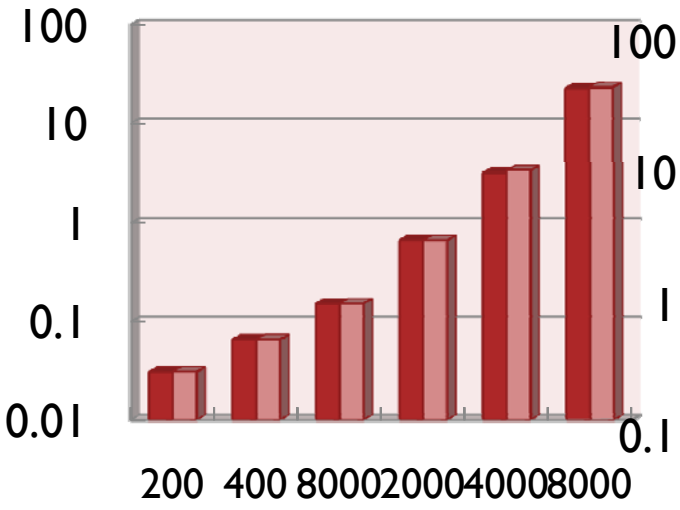
FIBER推定法による誤差（１）

■性能パラメタ*iud*の定義域内における推定誤差（推定パラメタによる実行と最適パラメタによる実行時間）

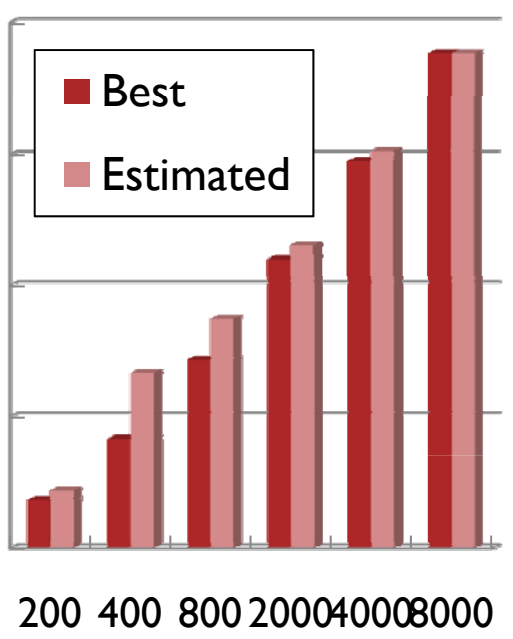
HITACHI SR8000/MPP
(1Node, 8PEs)



Fujitsu VPP800/63
(8PEs)



A PC Cluster (4PEs)



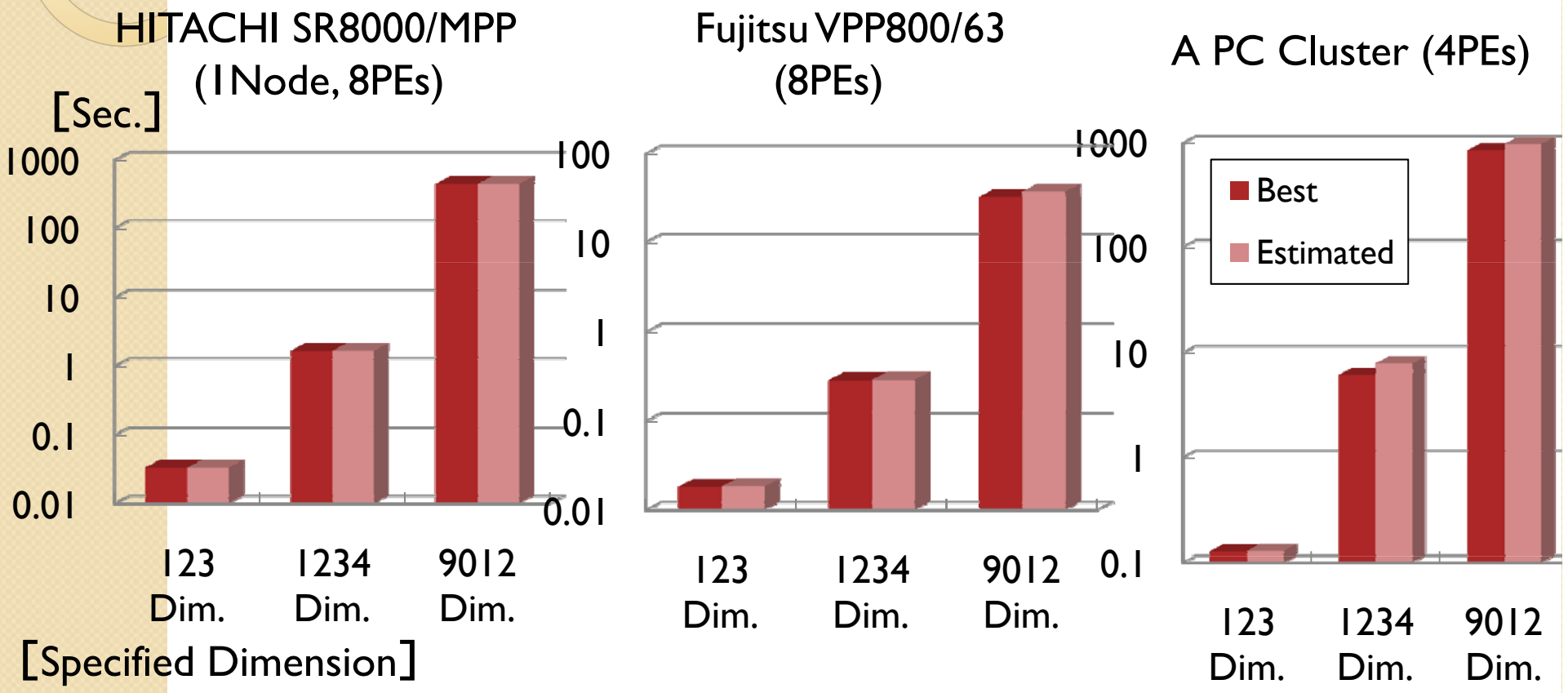
最大性能劣化0.6%

最大性能劣化6.7%

最大性能劣化319%

FIBER推定法による誤差（２）

■ 性能パラメタ *iud* について、*n*の
サンプリング点以外を指定した場合



最大性能劣化 0.3%

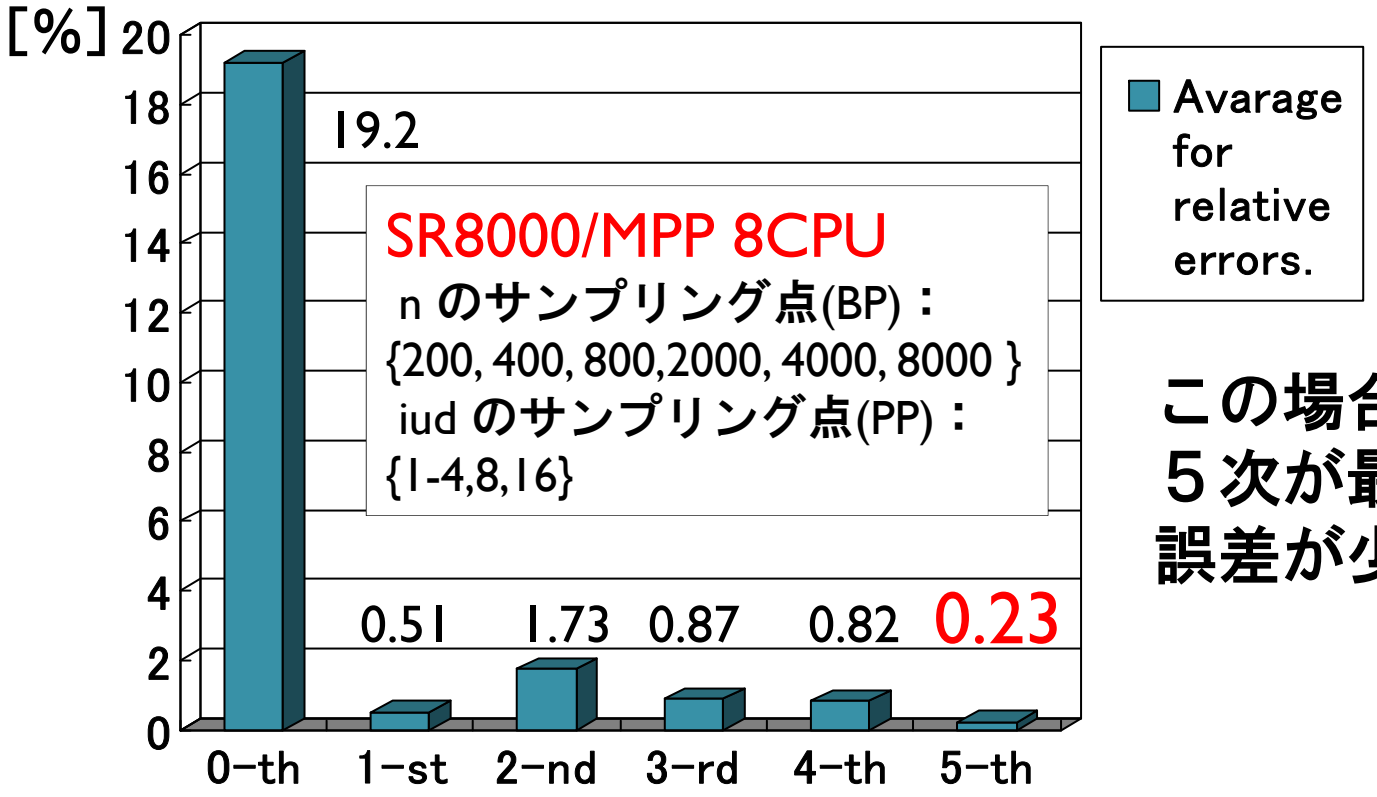
最大性能劣化 14.4%

最大性能劣化 28.7%

コスト定義関数の次数

iud に関する予測誤差の評価

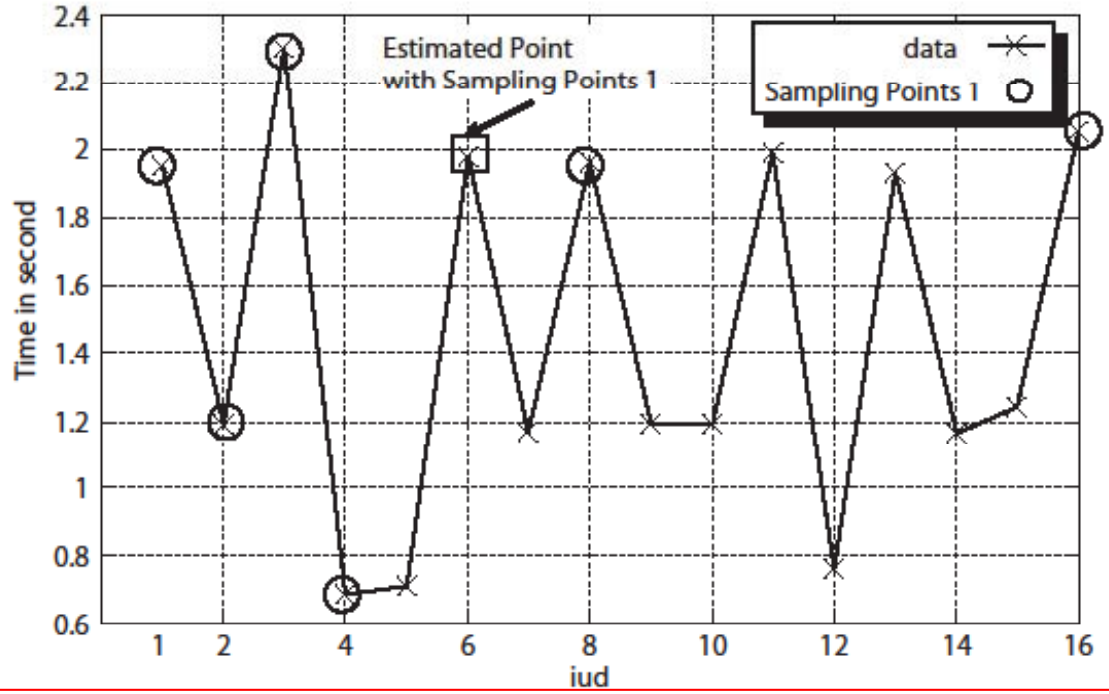
- 推定パラメタによる実行時間と、実測による最速時間の相対誤差
- 0次近似とは，つねにパラメタ１を指定するもの



この場合は、
５次が最も
誤差が少ない

多項式近似の問題点

- PCで行列サイズが小さい時
  - Householder三重対角化
  - 行列更新処理のアンローリング段数(iud)
  - N=400
  - Intel Pentium4 (2.0 GHz) , 1 GB (Direct RDRAM/ECC 256 MB*4) ,ASUSTek P4T-E+A (Socket 478).



挙動が不安定なものは、固定サンプリング点による多項式近似では推定できない

関連研究２：
動的サンプリング追加による定義域の縮小
（D-SPLINE法）

田中輝雄、片桐孝洋、弓場敏嗣、電子情報通信学会論文誌 A、
Vol.J90-A, No.4, pp .281--291 (2007)
「ソフトウェア自動チューニングにおける標本点追加型性能パラメタ推定法」

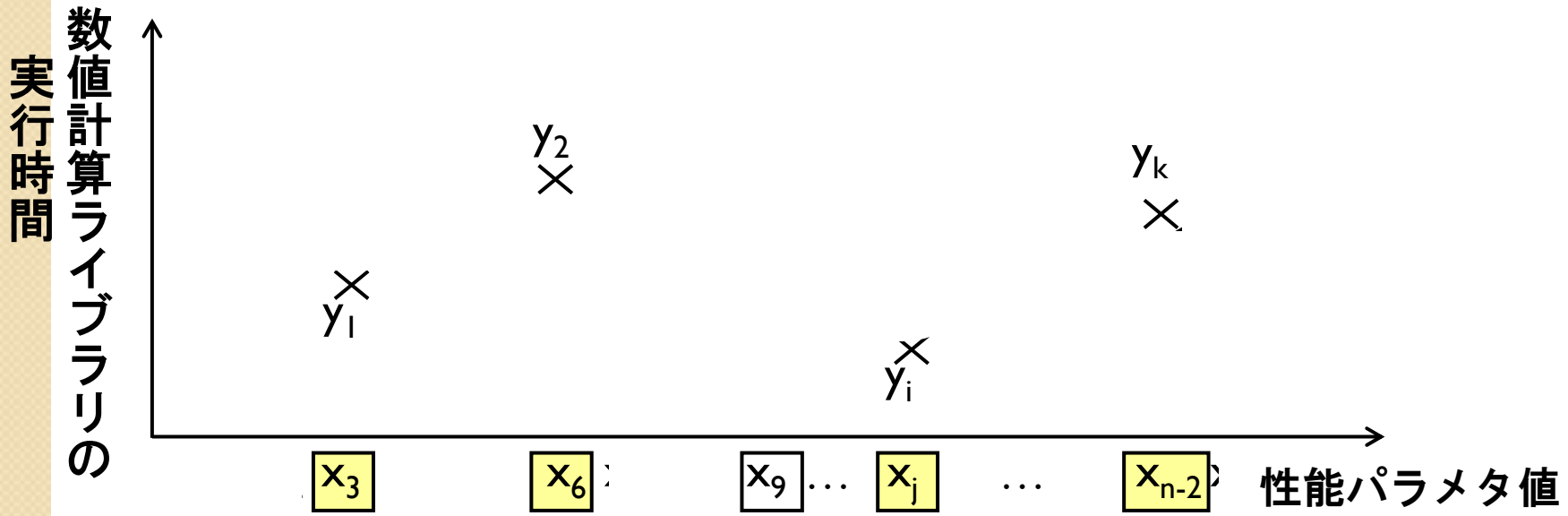
標本点逐次追加型性能パラメタ推定法の設計

- コスト定義関数の選択
  - (1) 少ない標本点から推定できること
  - (2) 再計算のための計算コストが小さいこと

- 標本点追加時の基準
  - (1) 追加するときの選択基準
  - (2) 追加するか否かの終了判定基準

コスト定義関数 *d-Spline*

コスト定義関数 $f(x)$ を $f=(f_1,f_2,f_3,\ldots,f_j,\ldots,f_n)^t$ で表現

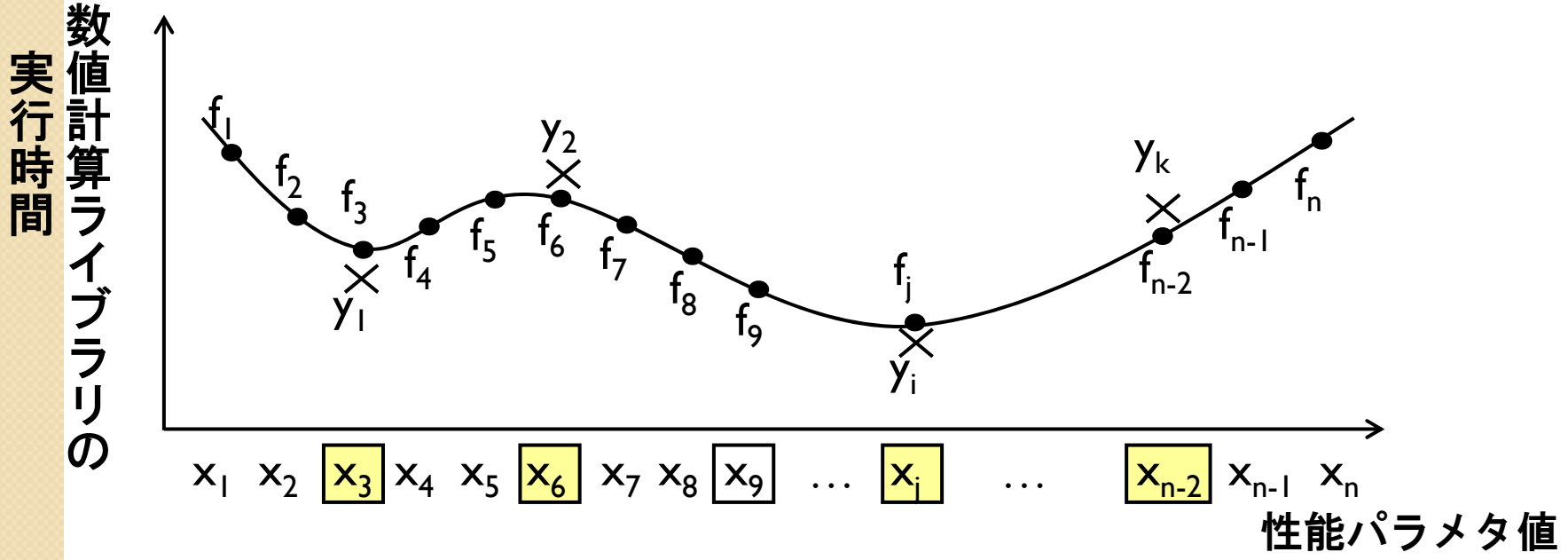


□：性能パラメタの取り得る値（内，■：標本点）

y_i：標本点に対する数値計算ライブラリの実測値（$1 \leqq i \leqq k$, $k<n$）

コスト定義関数 *d-Spline*

コスト定義関数 f(x) を $f=(f_1,f_2,f_3,\ldots,f_j,\ldots,f_n)^t$ で表現



□：性能パラメタの取り得る値（内，■：標本点）

y_i：標本点に対する数値計算ライブラリの実測値（$1 \leqq i \leqq k,\ k < n$）

コスト定義関数 *d-Spline*

- コスト定義関数 f を確定するために，f の滑らかさを

  $|f_{j-1}-2f_j+f_{j+1}|,\ 2 \leqq j \leqq k-1$ で表現

  ２階差分→標本点は等間隔

  αは関数当てはめのときはABICで決定可能だが、関数形状を求めないので十分小さな値を設定

- コスト定義関数 f を次の評価関数の最小化により選択

  （式1）　$\min_{f}\left(\|y-Ef\|^2+\alpha^2\|Df\|^2\right)$

$$E=\underbrace{\begin{pmatrix} 1 & & & & 0 \\ & 1 & & & \\ & & 0 & 1 & \\ & & & \cdots & \\ 0 & & & & 1 \end{pmatrix}}_{n\text{列}}\ \Big\}\ k\text{行} \qquad D=\underbrace{\begin{pmatrix} 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \cdots & & \\ & & 1 & -2 & 1 \end{pmatrix}}_{n\text{列}}\ \Big\}\ n-2\text{行}$$

実測値y（k個）と関数値f（n個）の対応行列

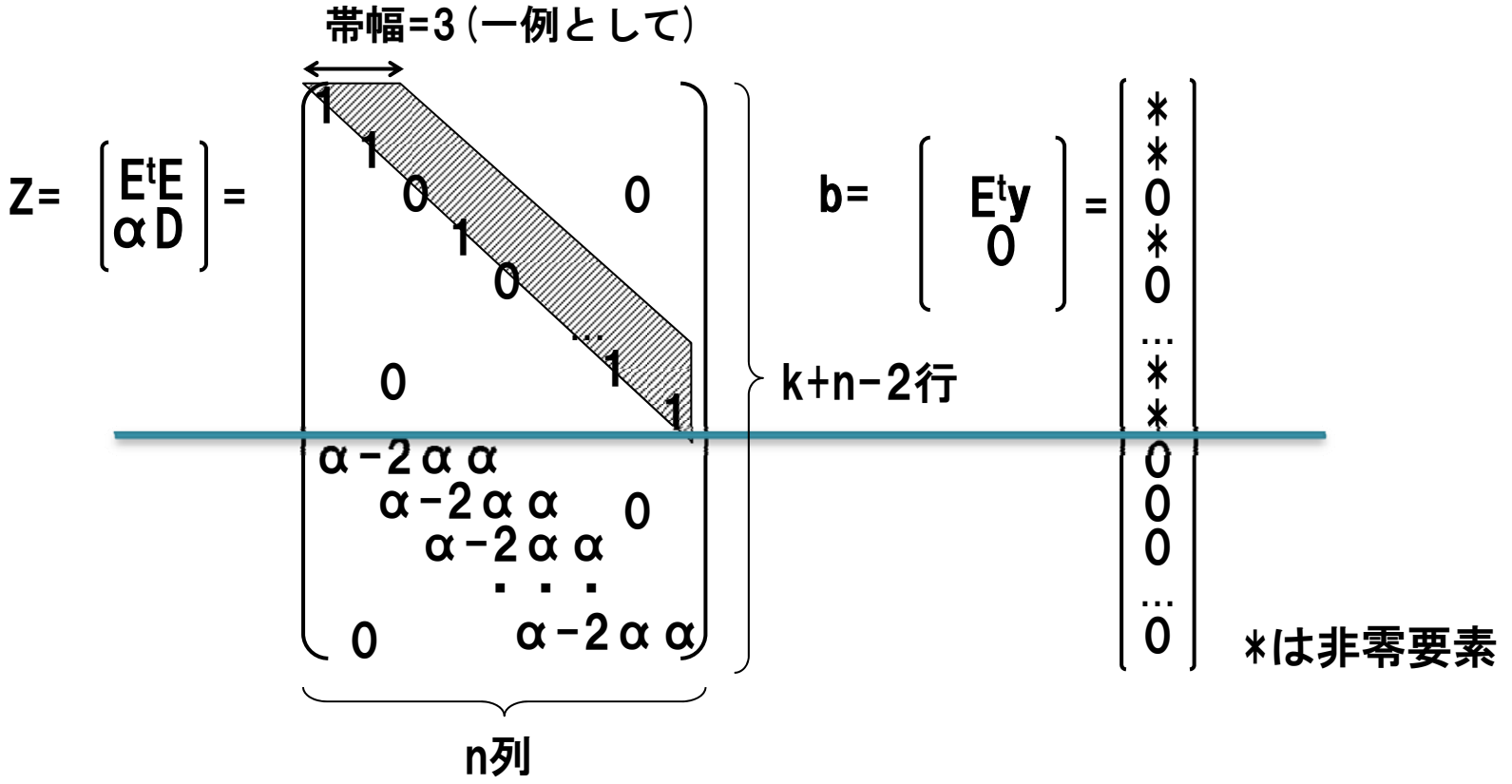
滑らかさ計算行列

コスト定義関数 f を *d-Spline* と呼ぶ

コスト定義関数 *d-Spline*

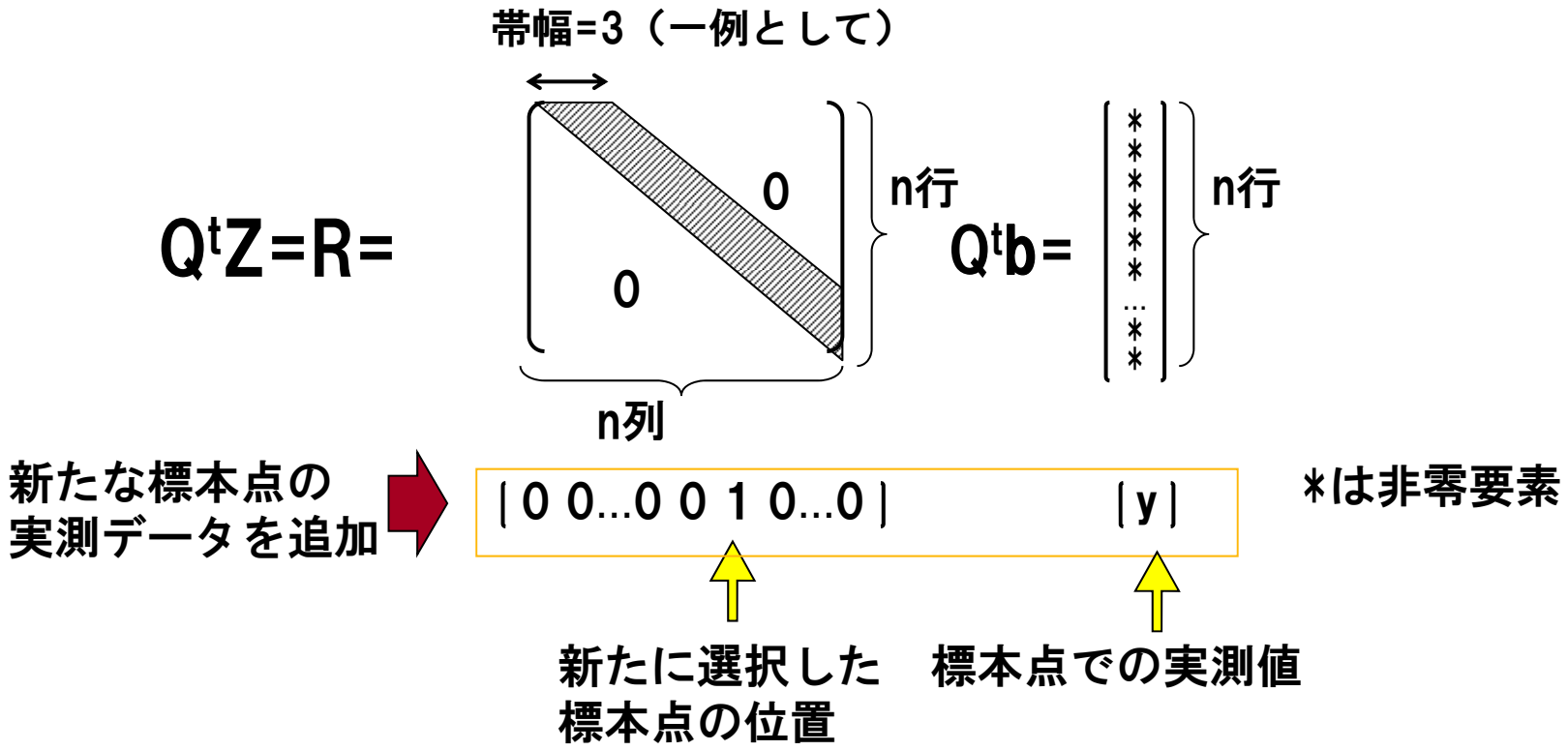
- （式１）は最小二乗問題 $\min_{f} \| b - Zf \|^2$ を解くことと同値

Givens変換によるQR分解法で効率良く解くことが可能



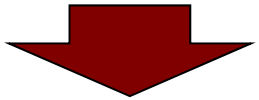
コスト定義関数 *d-Spline* -標本点追加時-

- 一度Rを作成すると，新たに選択した標本点の実測データを追加するだけで，Givens法を用いてRを更新可能



コスト定義関数 *d-Spline*

- *d-Spline*の特徴
  - ✓ 計算量は$O(n)$，nは f の要素数（数十～百）
  - ✓ Rの繰り返し計算と高い親和性
    （繰り返し計算量$O(1)$）
- 一方，数値計算ライブラリの計算量は
  $O(N^2) \sim O(N^3)$，Nは行列サイズ（数百～数千以上）
- したがって計算量は，
  d-Spline ≪ 数値計算ライブラリ



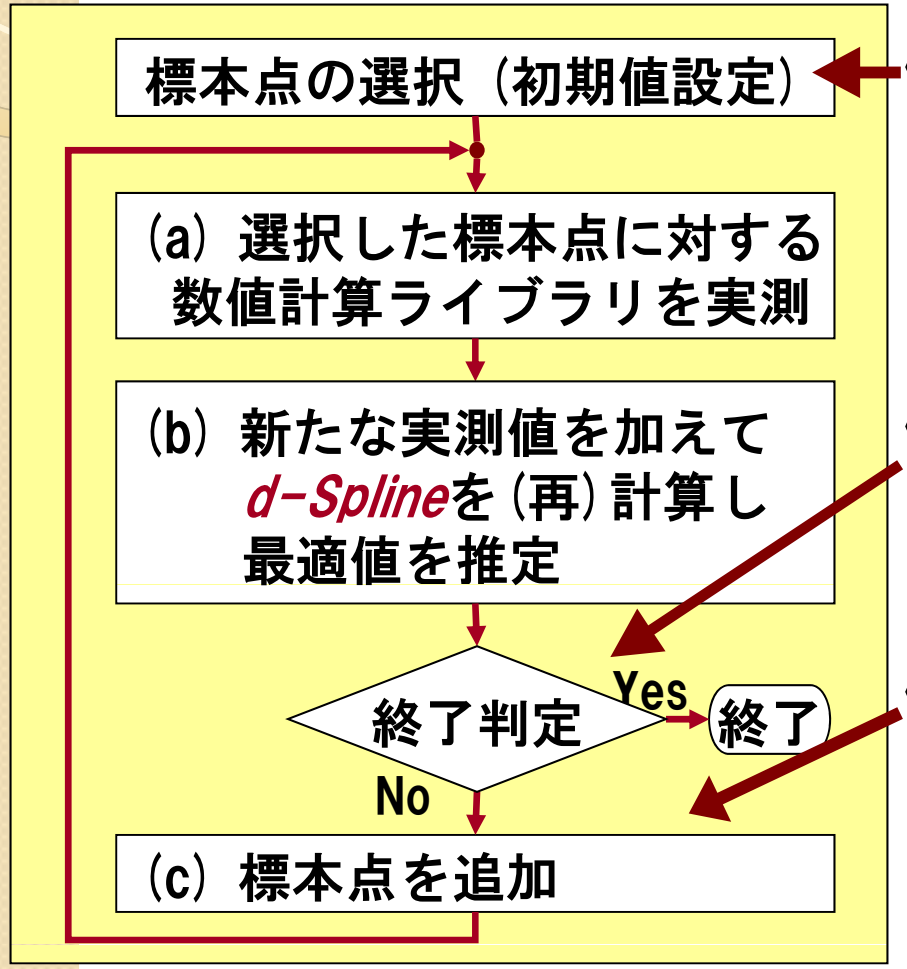
***d-Spline*の計算量は無視可能**

標本点逐次追加型性能パラメタ推定法の設計

- コスト定義関数の選択
  - (1) 少ない標本点から推定できること
  - (2) 再計算のための計算コストが小さいこと

- **標本点追加時の基準**
  - (1) 追加するときの選択基準
  - (2) 追加するか否かの終了判定基準

標本点逐次追加型パラメタ推定手順

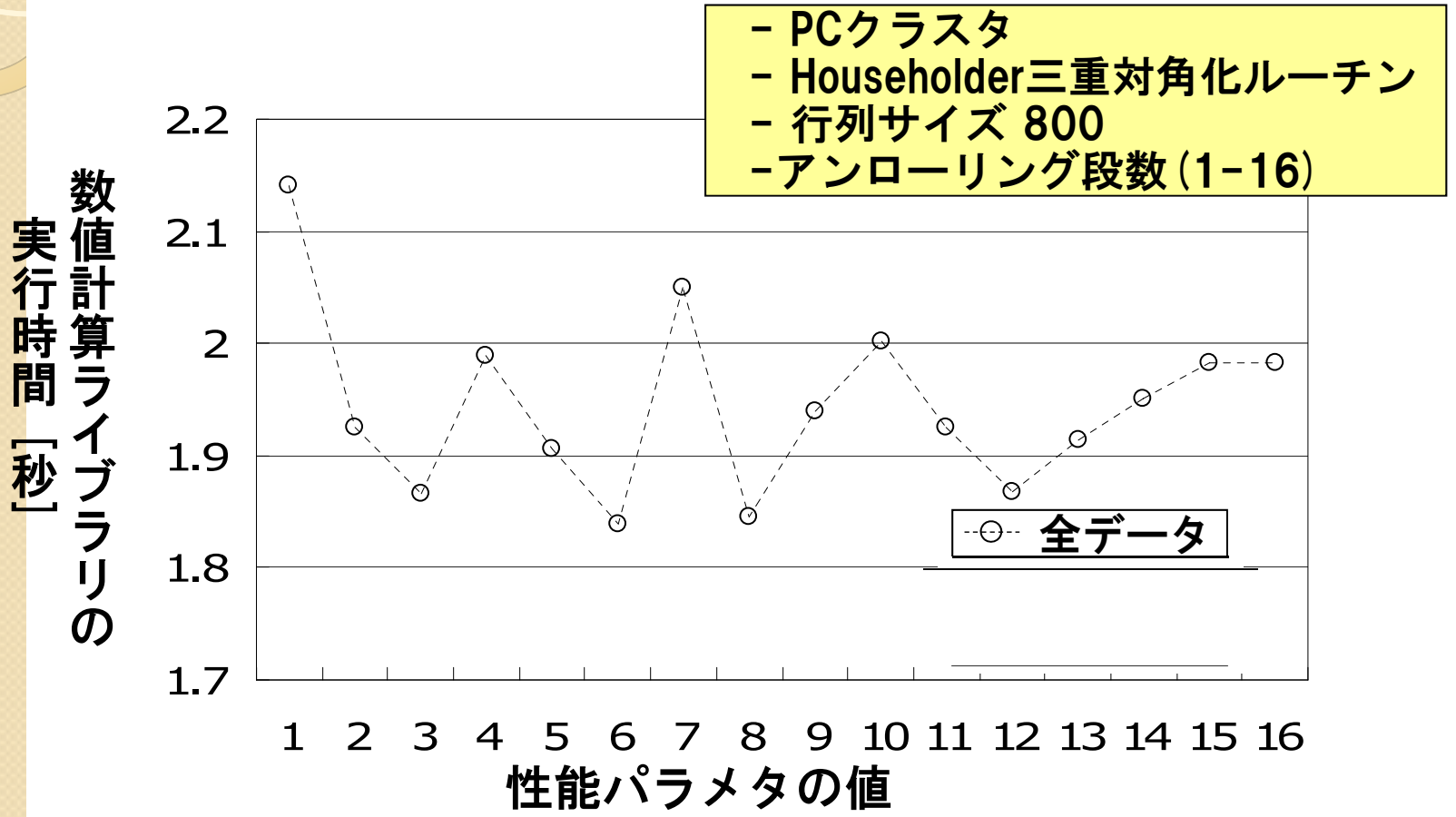


《選択基準:初期値》
① 等間隔の４点（両端を含む）

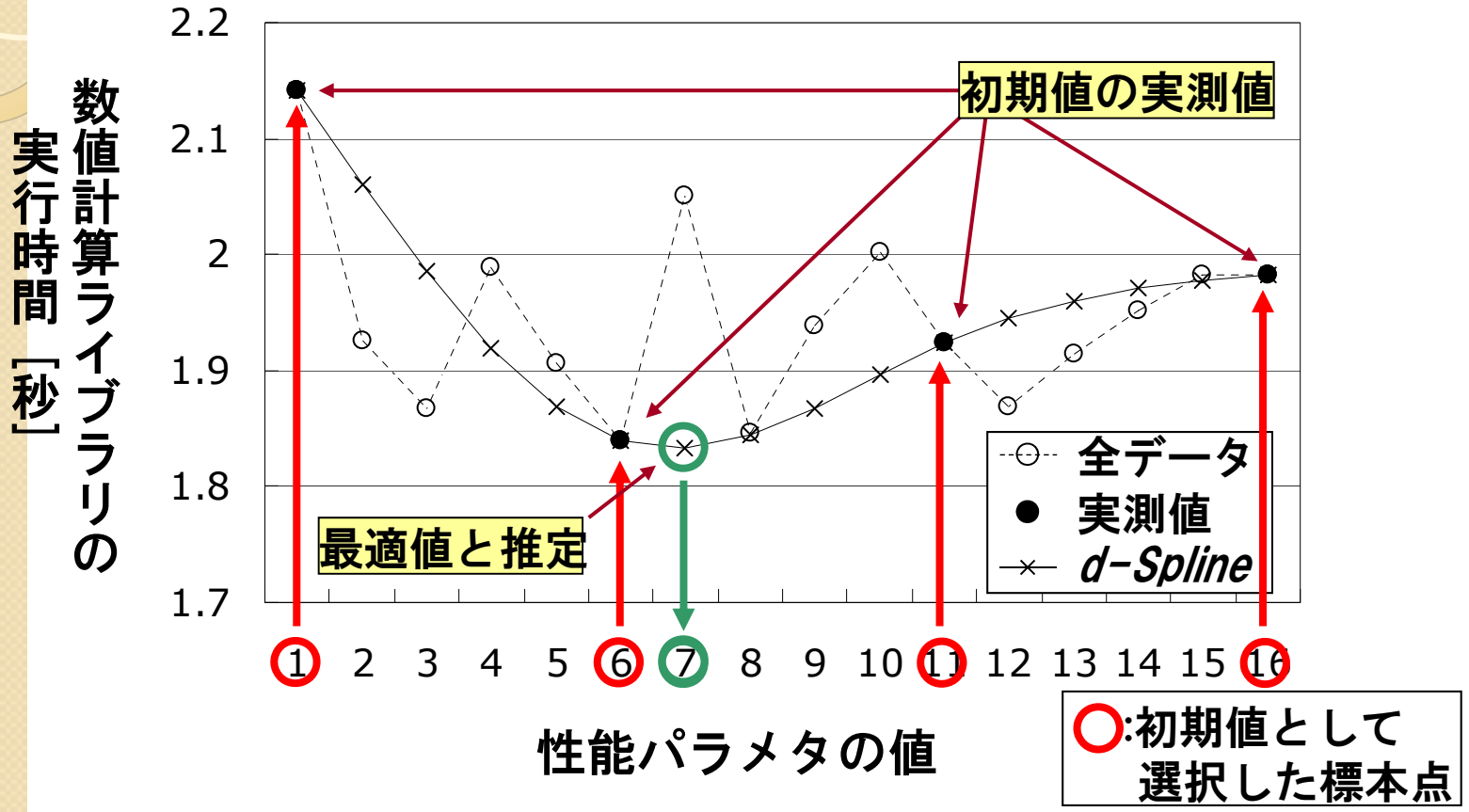
《終了判定基準》
① 推定した最適値がｎ回連続で
同一（ｎ=2,3,4,5…）

《選択基準:追加》
① (b)で推定した最適値
② $\max_j |f_{j-1}-2f_j+f_{j+1}|$となる$x_j$

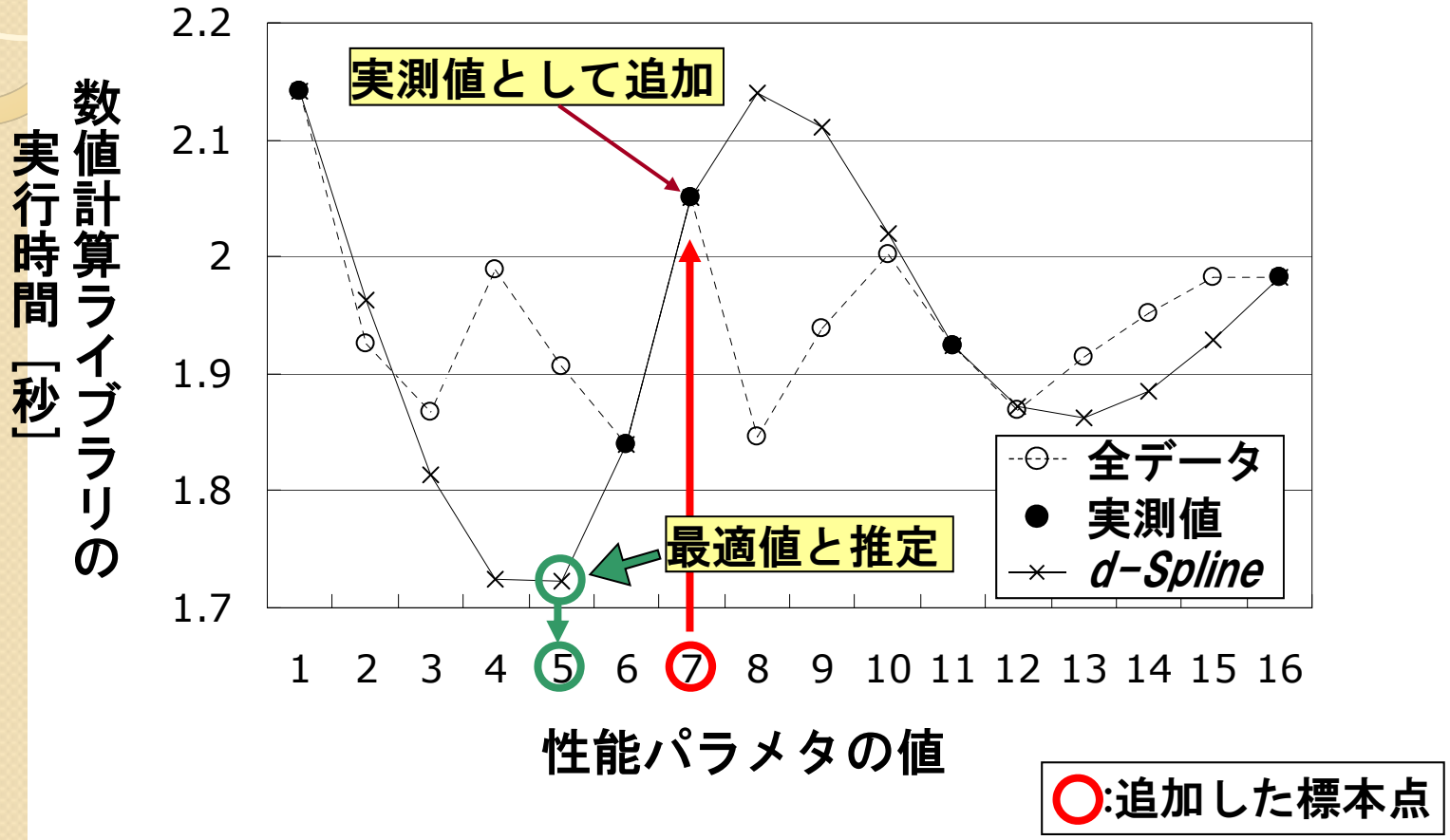
事例１


- PCクラスタ
- Householder三重対角化ルーチン
- 行列サイズ 800
-アンローリング段数(1-16)
数値計算ライブラリの実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
全データ

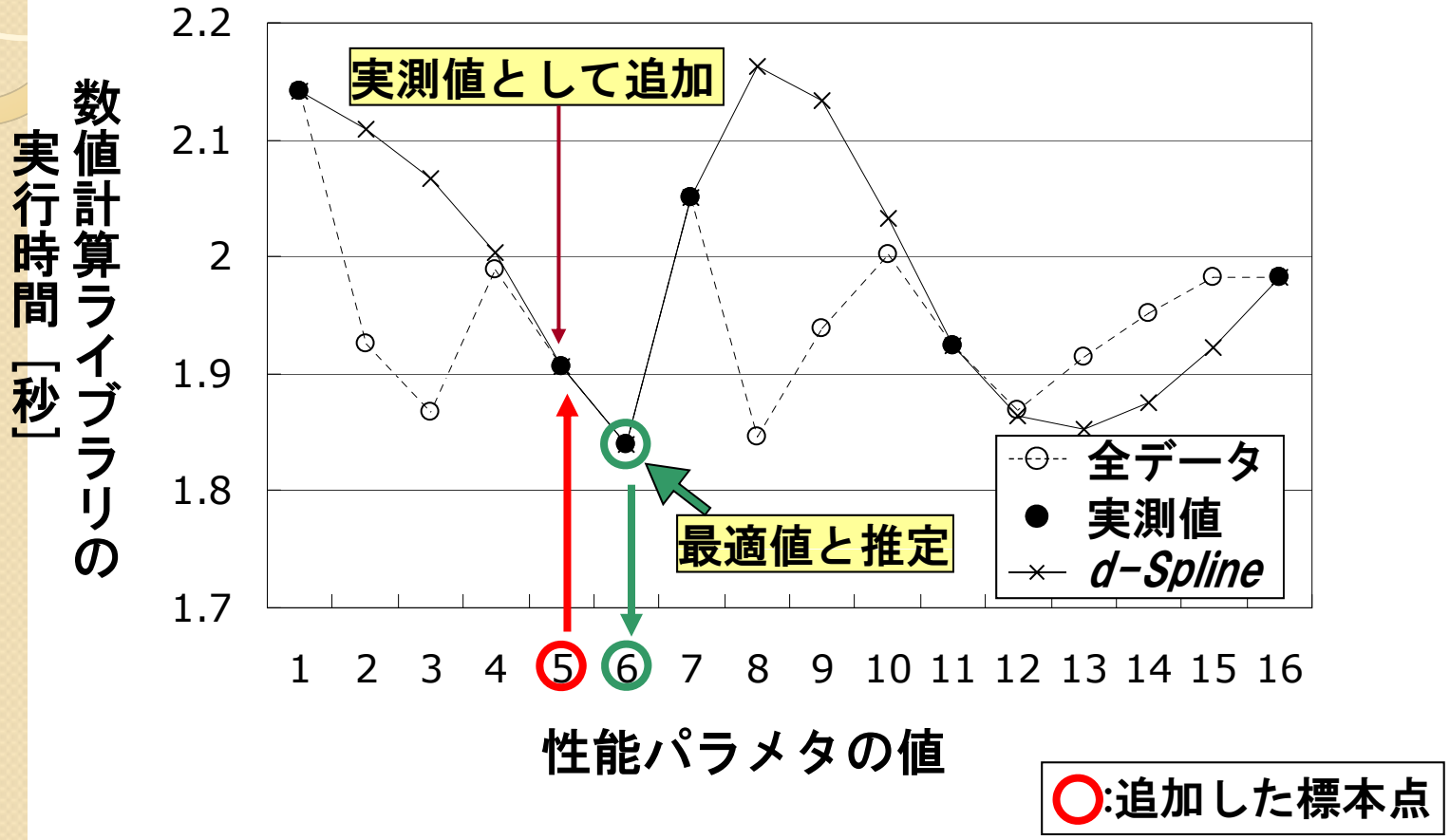
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
初期値の実測値
最適値と推定
全データ
実測値
d-Spline
性能パラメタの値
○:初期値として
選択した標本点

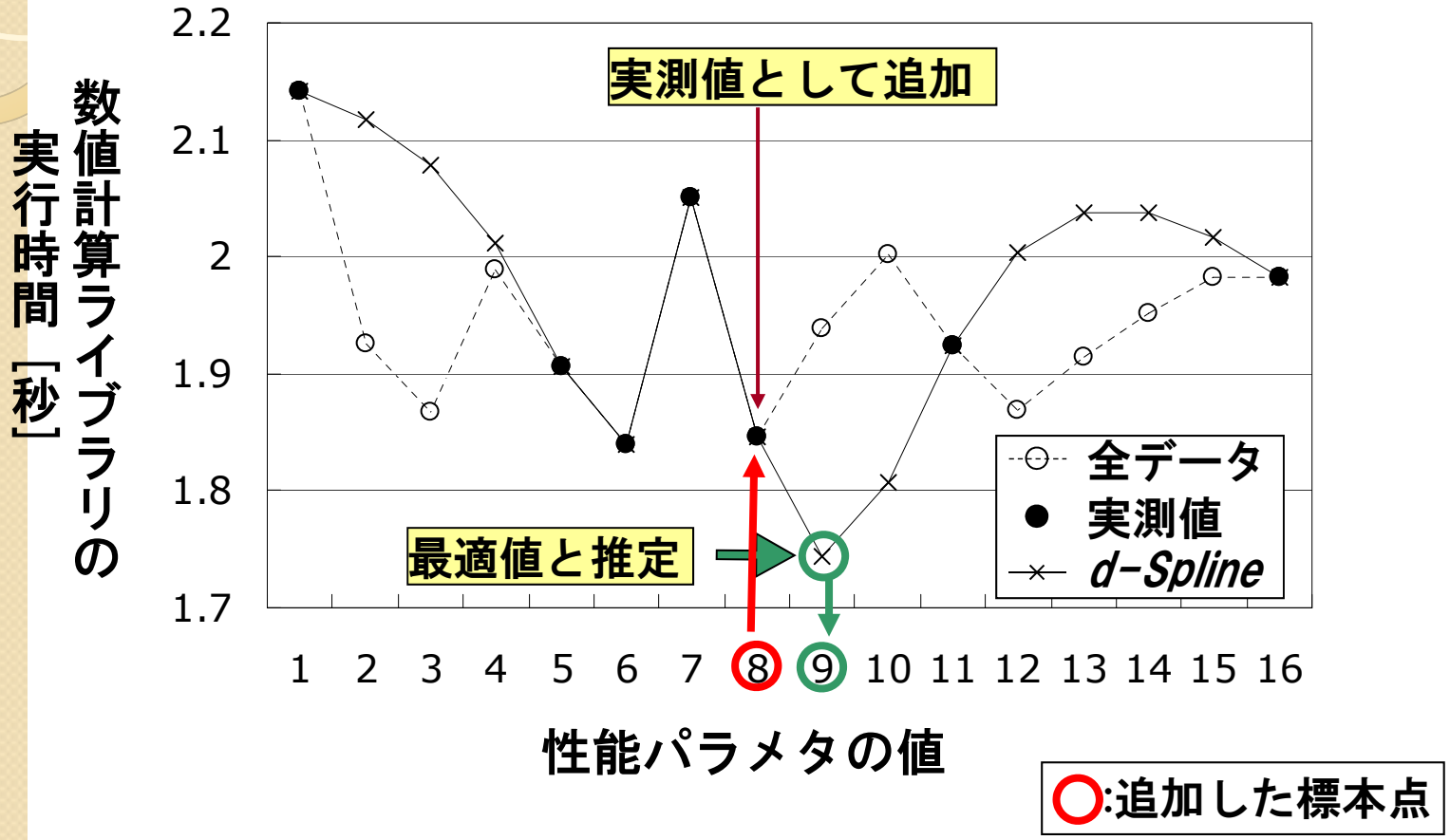
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

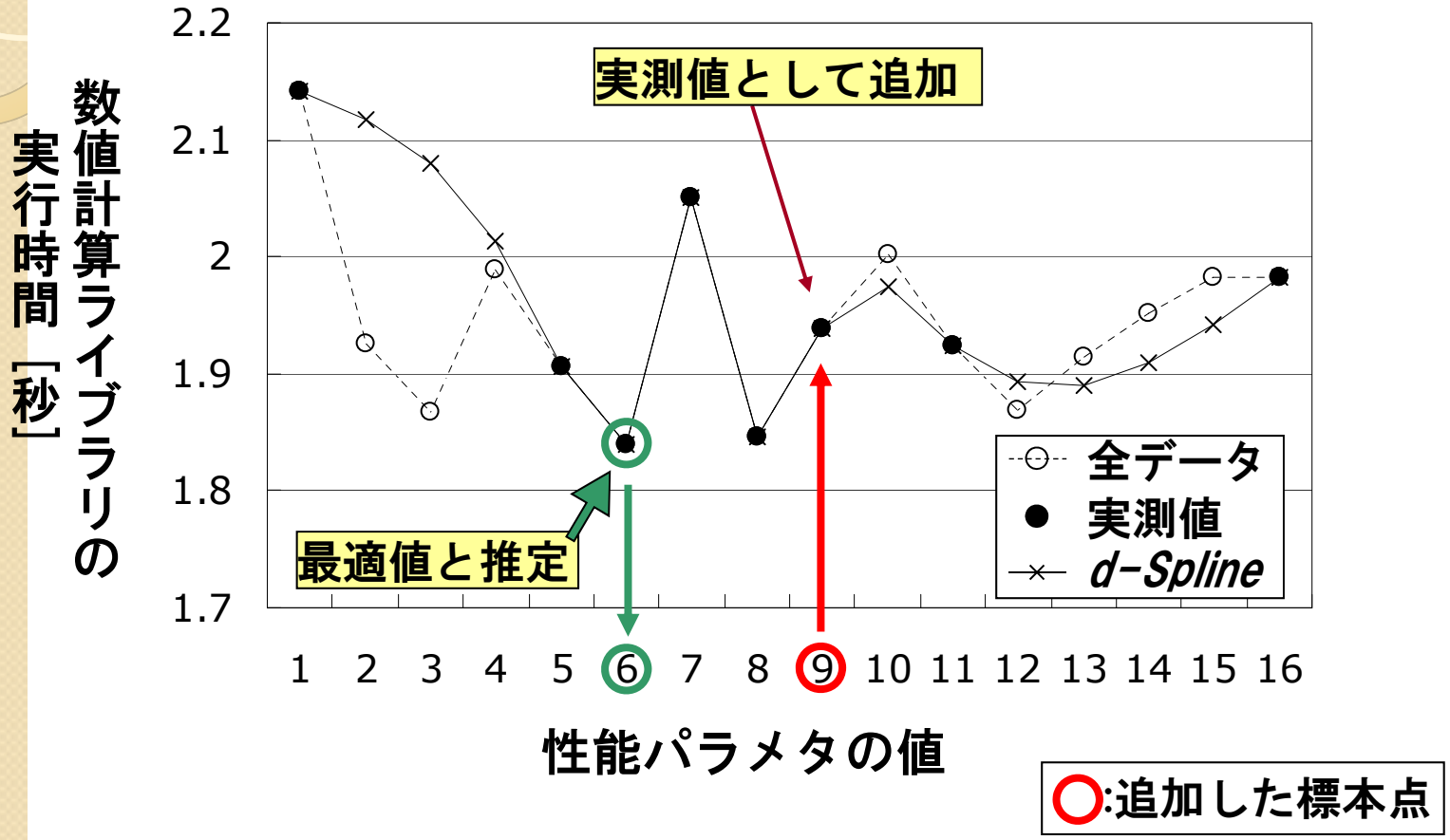
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
:追加した標本点

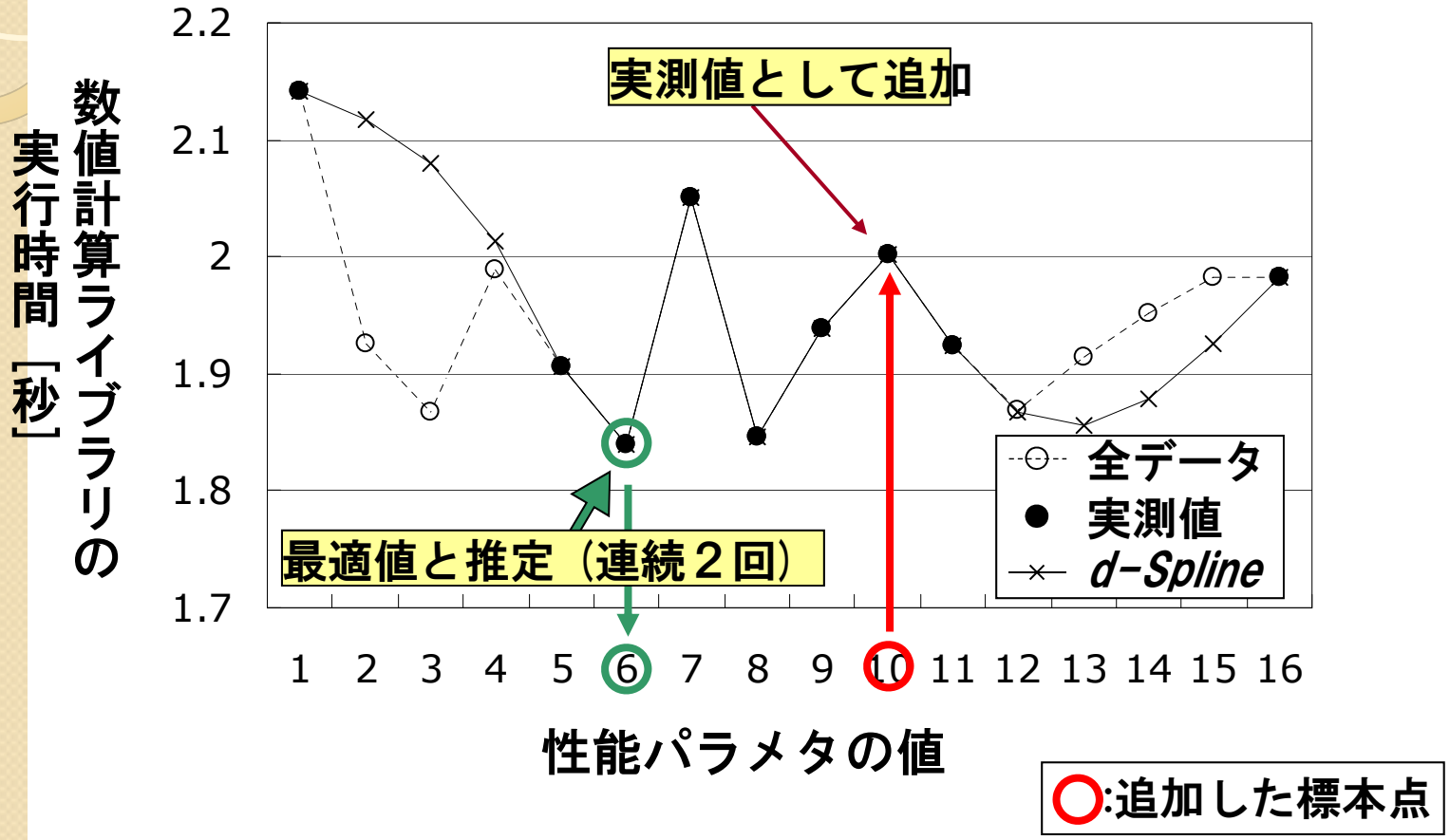
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

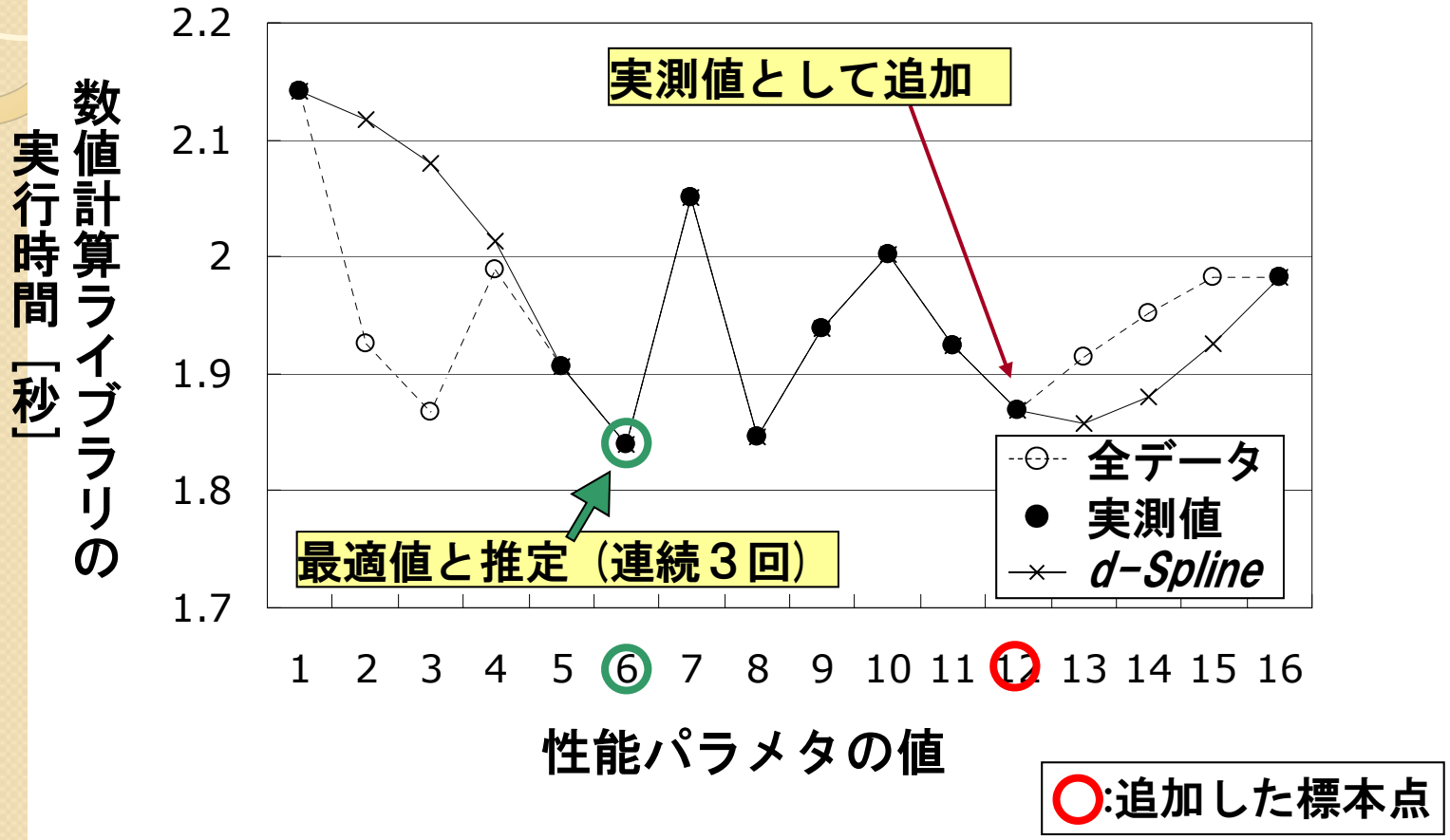
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

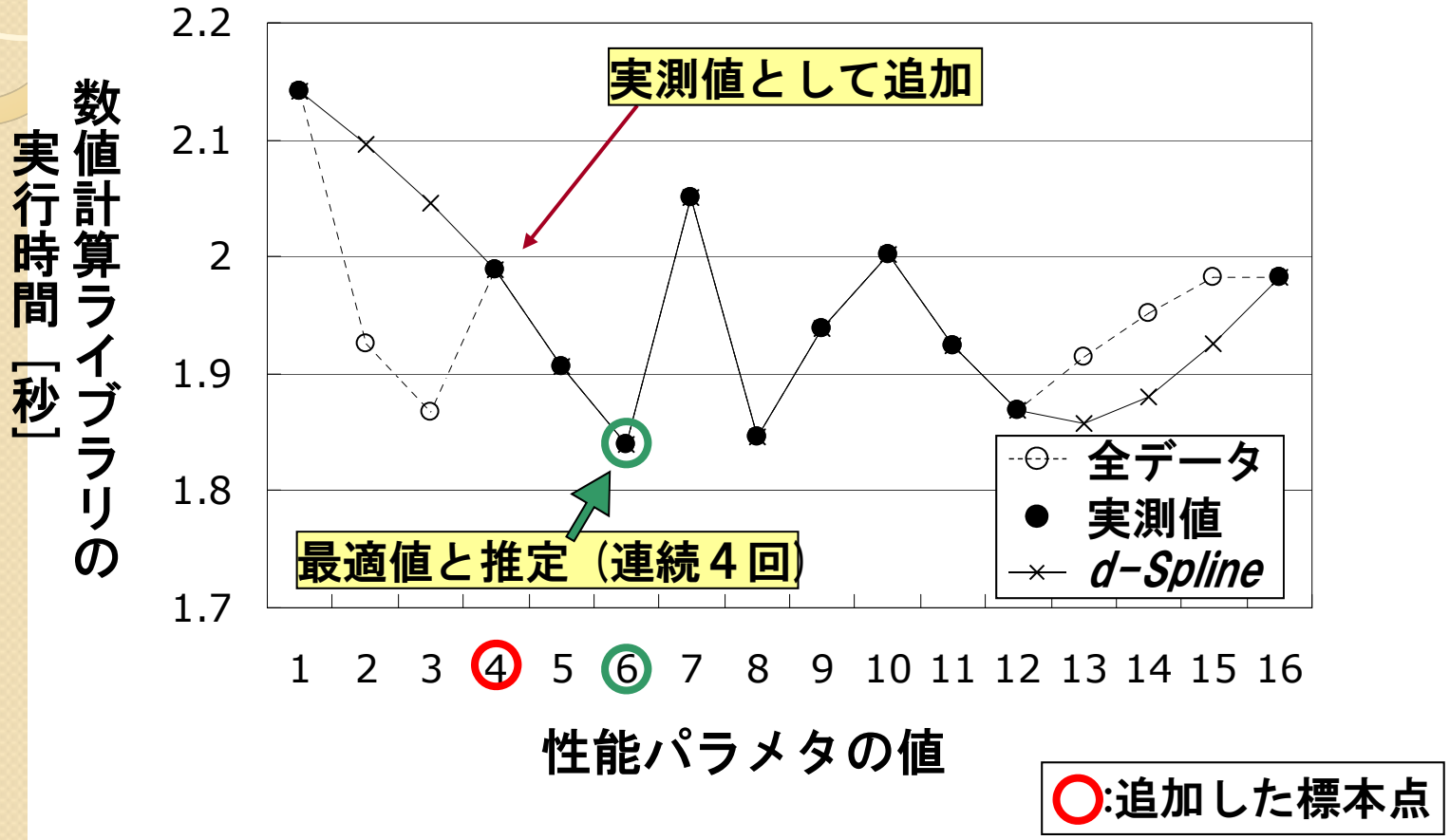
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
実測値として追加
最適値と推定（連続２回）
全データ
実測値
d-Spline
性能パラメタの値
:追加した標本点

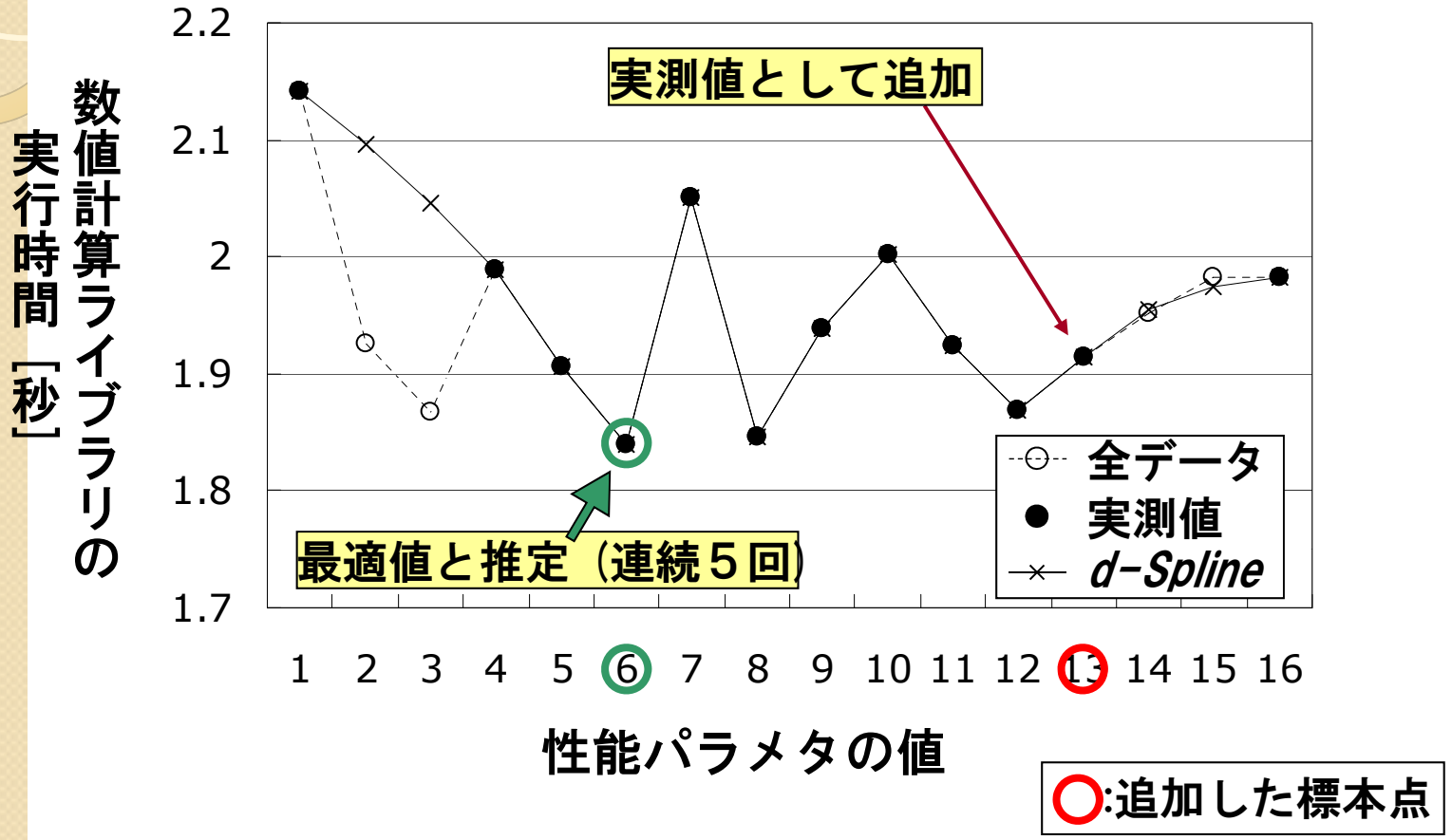
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定（連続３回）
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

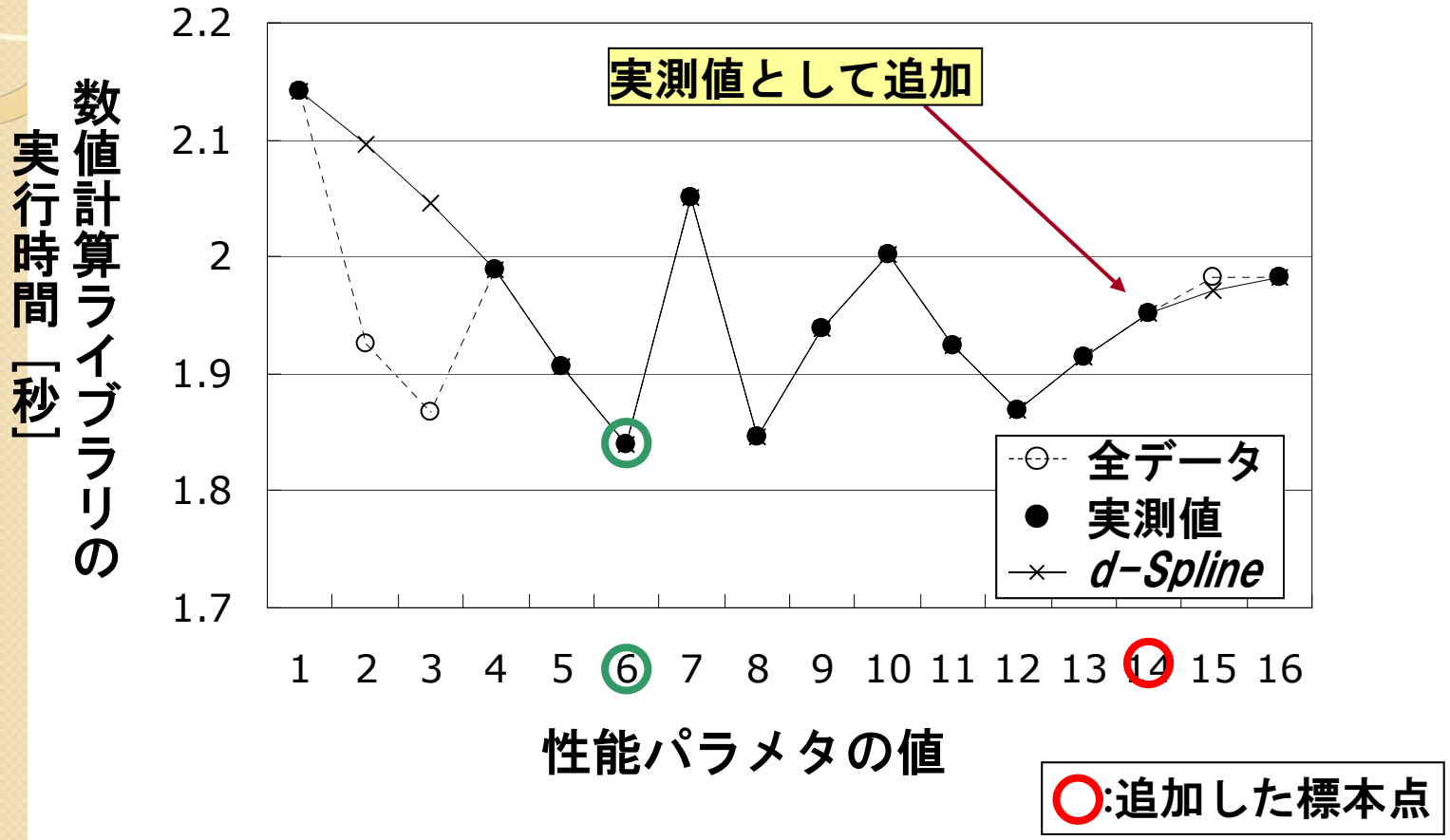
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定（連続４回）
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

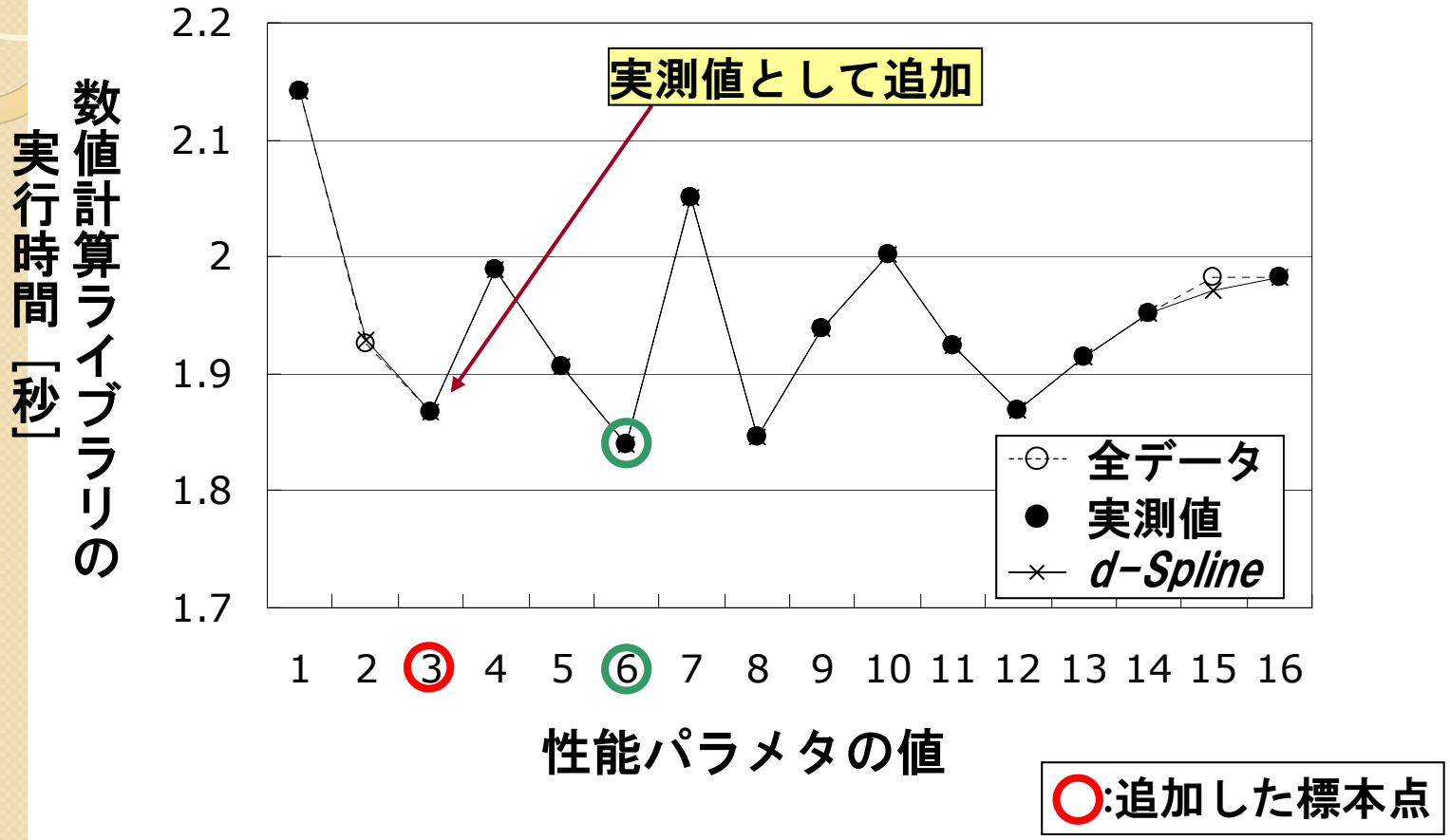
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
最適値と推定（連続５回）
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

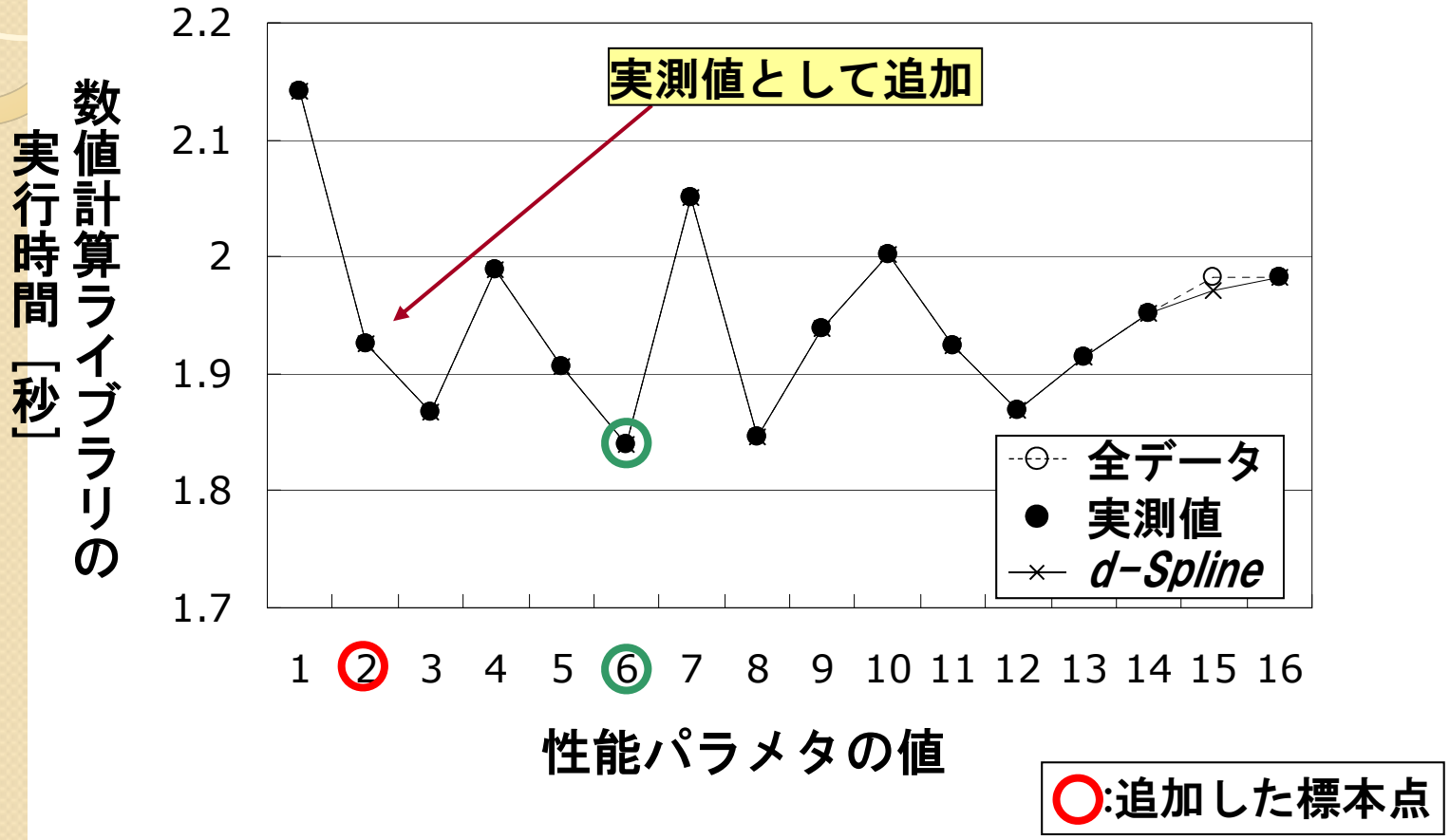
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

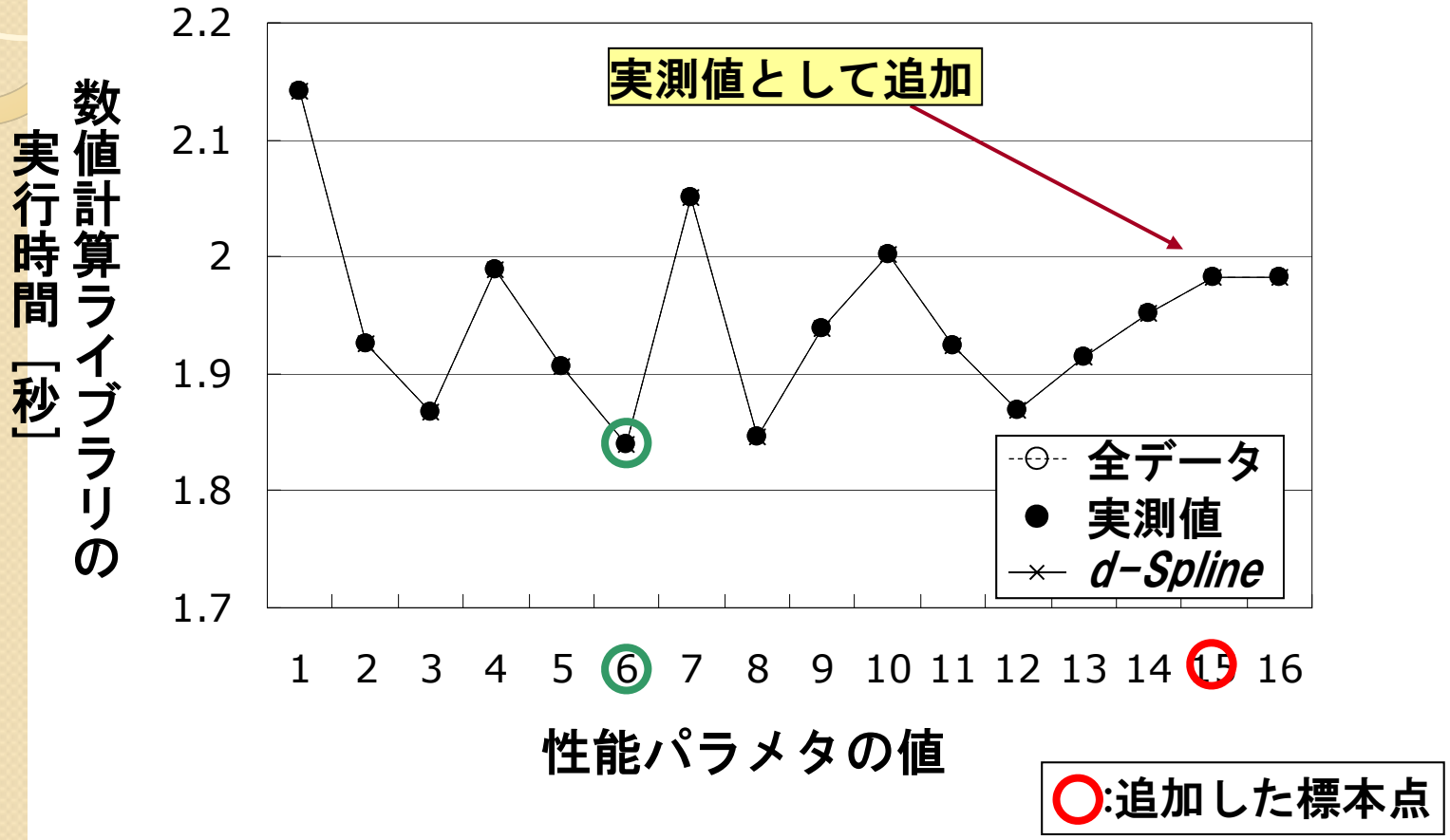
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
:追加した標本点

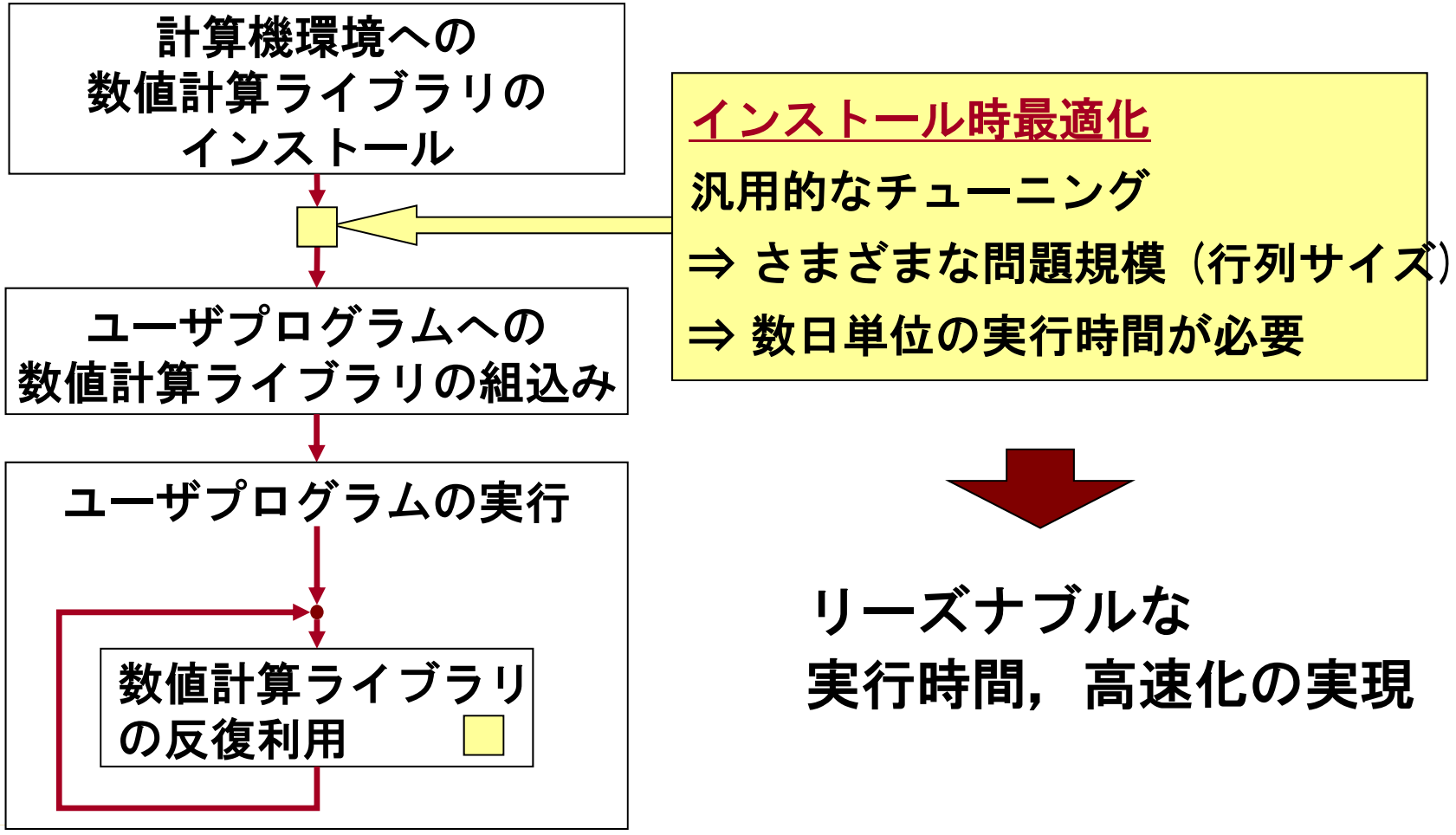
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
:追加した標本点

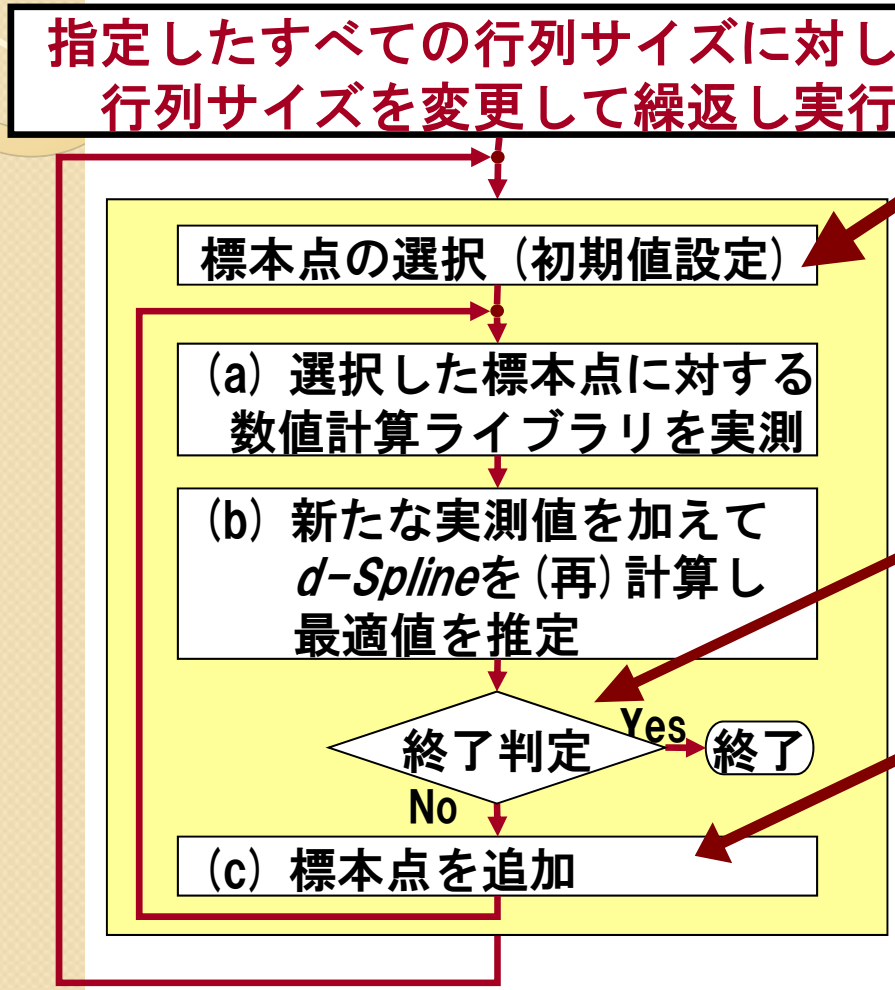
事例１


数値計算ライブラリの
実行時間［秒］
2.2
2.1
2
1.9
1.8
1.7
実測値として追加
全データ
実測値
d-Spline
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
性能パラメタの値
○:追加した標本点

ソフトウエア自動チューニング-最適化フェーズ-



インストール時標本点逐次追加型パラメタ推定手順



《選択基準:初期値》
① 等間隔の４点（両端を含む）
② 直前の行列サイズで選択した最適値（５番目の初期値）

《終了判定基準》
① 推定した最適値がｎ回連続で同一（ｎ=2,3,4,5…）

《選択基準:追加》
① (b)で推定した最適値
② $\max_j |f_{j-1}-2f_j+f_{j+1}|$となる$x_j$

評価実験：計算環境– 数値計算ライブラリ –

１．数値計算ライブラリ

１.１ Householder三重対角化ルーチン

(a) 行列･ベクトル積のアンローリング段数（1～16） ⇒ 【TRD1】

(b) 行列更新処理のアンローリング段数(1～16) ⇒ 【TRD2】

$$PP = IP = \{TRD1, TRD2\}$$
$$TRD1 \in [1,16],\ TRD2 \in [1,16]$$

１.２ Householder逆変換ルーチン

(a) 最外側ループ処理のアンローリング段数(1～16) ⇒ 【HIT】

$$PP = IP = \{HIT\}$$
$$HIT \in [1,16]$$

１.３ Gram-Schmidt法を用いたQR分解ルーチン

(a) ブロックアルゴリズムのブロック幅(1～16) ⇒ 【MGS】

$$PP = IP = \{MGS\}$$
$$MGS \in [1,16]$$

評価実験：計算環境 –計算機とその構成 –

２．計算機

(a) スーパコンピュータSR8000

- コンパイラ：Hitachi Optimized Fortran 90 V02-04 -opt4
- ノード間通信：日立最適化MPI

(a1) １ノード，８プロセッサ ⇒ [SRn1p8]

(a2) ２ノード，16プロセッサ ⇒ [SRn2p16]

(b) PCクラスタ（４ノード，IA32） ⇒ [PCn4]

- Intel Pentium4 (2.0GHz)
- OS：Linux 2.4.9
- コンパイラ：PGI Fortran90 4.0-2 -Fast
- ノード間通信：MPICH1.2.1， 10/100Base-TXスイッチングハブ接続

$$BP = \{node, procs\}$$
$$SRn1p8 : node = 1, procs = 8$$
$$SRn2p16 : node = 2, procs = 16$$
$$PCn4 : node = 4, procs = \{\}$$

評価実験：計算環境 –計算機とその構成 –

３．評価対象の問題規模（行列サイズ）

(a) 100～1000まで100毎および2000, 3000, 4000の13パタン
（ただし，[SRn1p8]は4000を除く12パタン）

(b) PCn4にて，さらに16～256まで16毎の16パタン

⇒[PCn4small]

$$BP = \{N\}$$

$$N \in \{100,200,...,1000,2000,3000,4000\}$$

$$if\ SRn1p8\ N \in \{100,200,...,1000,2000,3000\}$$

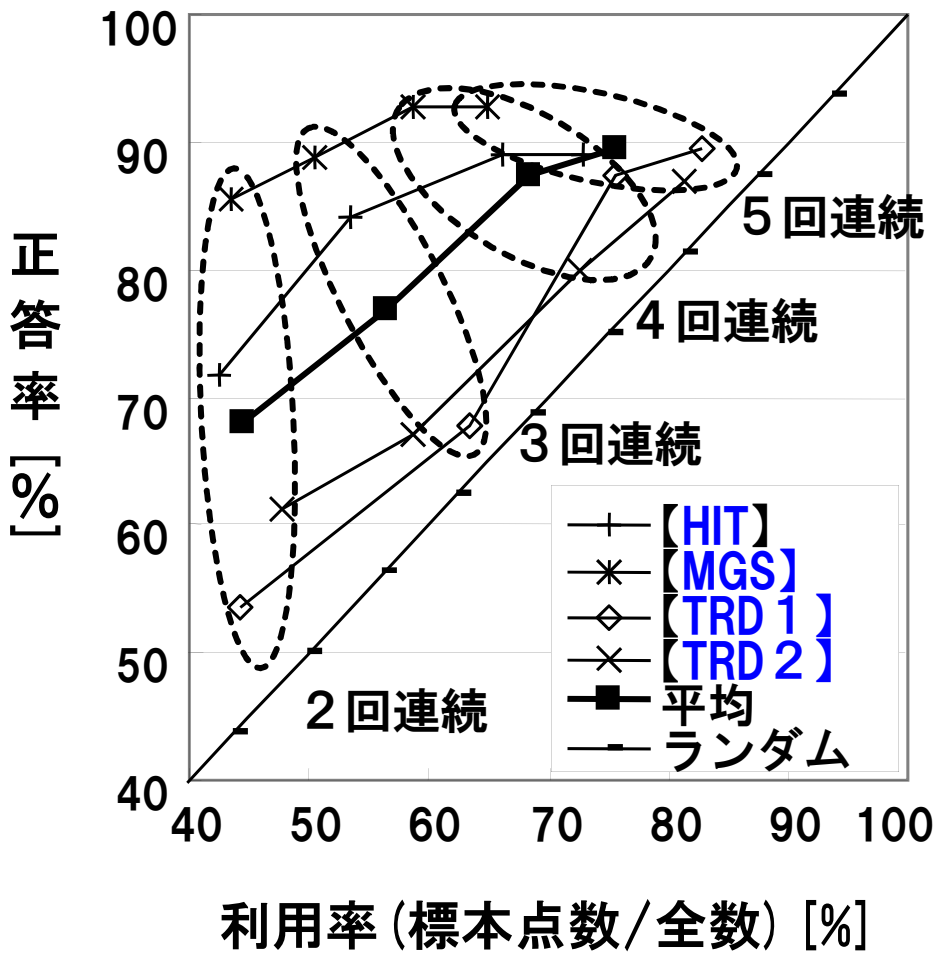
$$if\ PCn4$$

$$N \in \{16,32,64,...,256,100,200,...,1000,2000,3000,4000\}$$

$$F : d - spline$$

$$\min F(PP)\ s.t.\ BP = \{n, node, procs\}$$

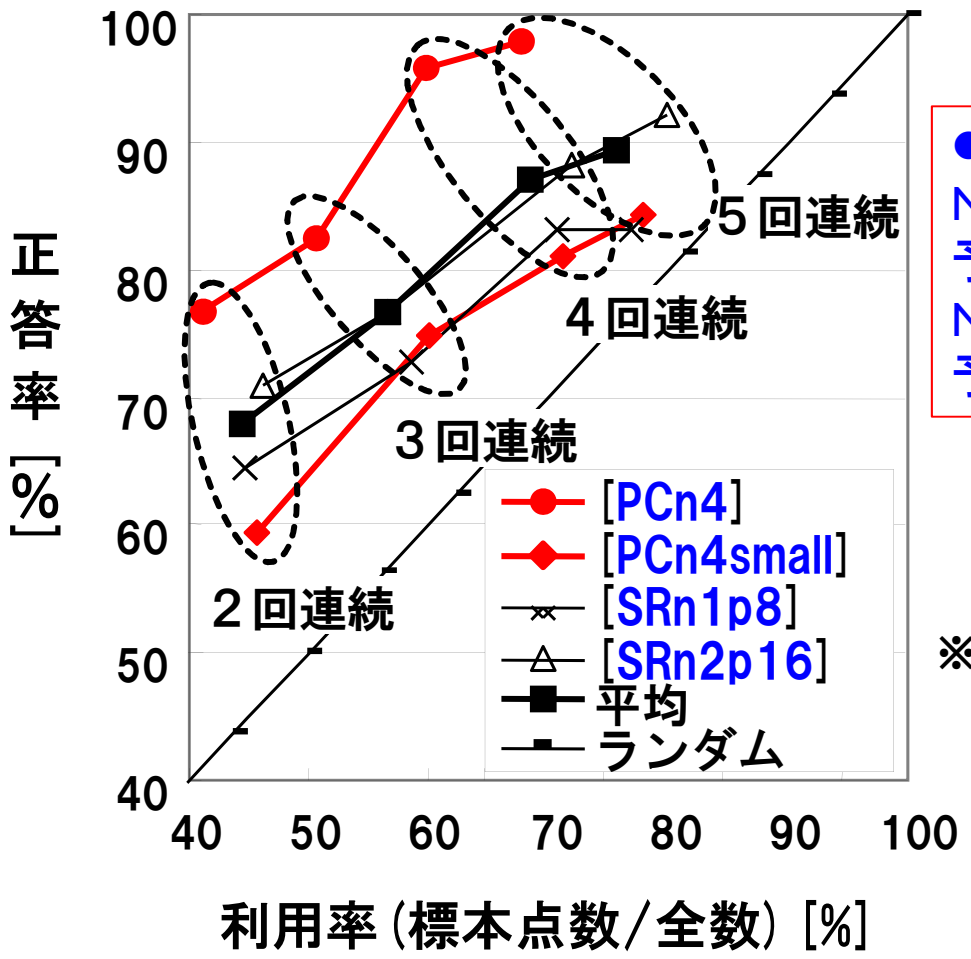
評価結果 － 計算対象の違いによる比較 －



●80%程度の
標本点の利用で
正答率は90%程度
●最悪50%程度の
正答率を許容するなら
50%の標本点数まで
削減可能

※各々の値は，
すべての計算機の
実測結果の平均値

評価結果 － 計算機の違いによる比較 －

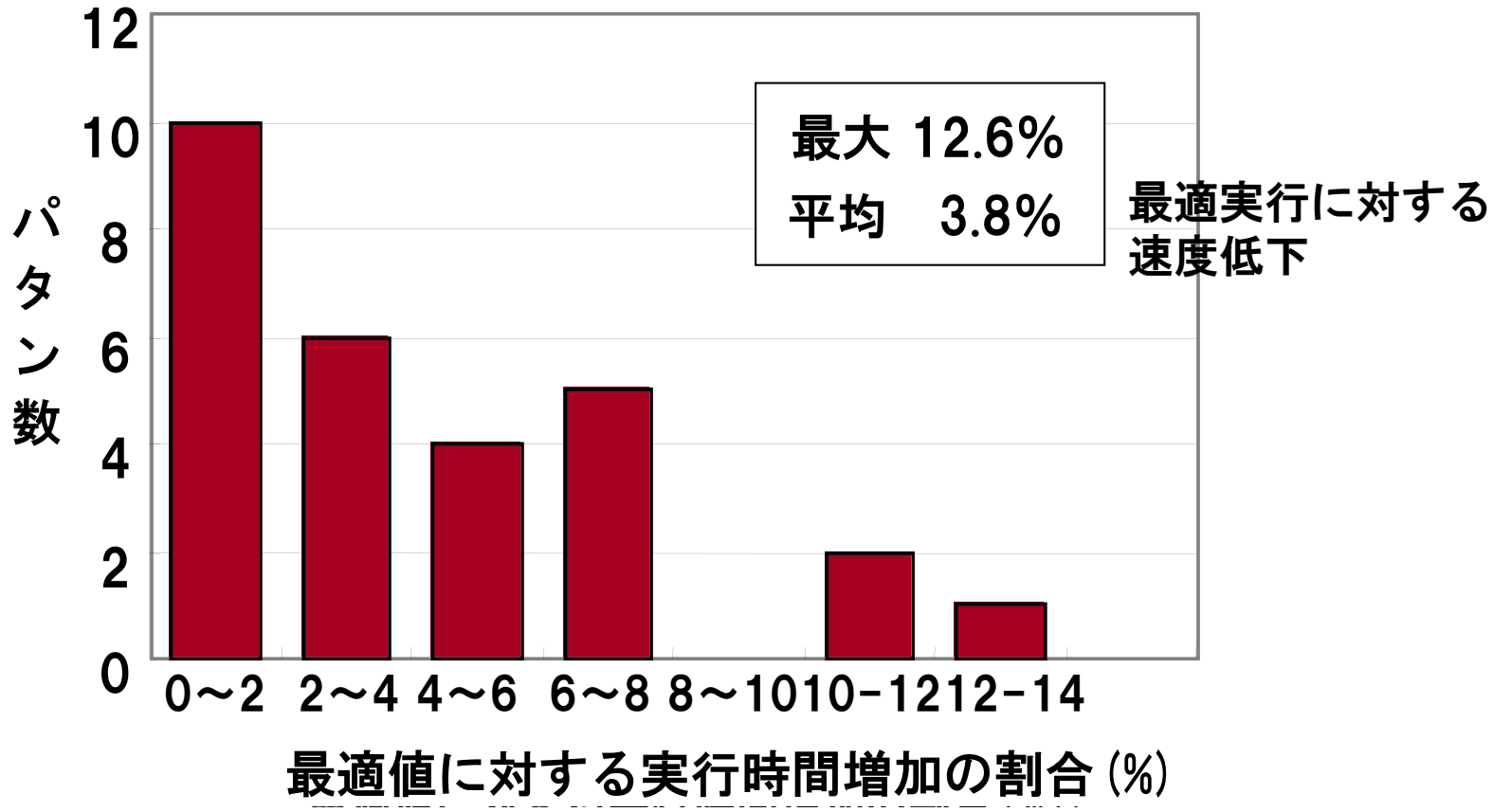


●PCクラスタは
Nが大きくなると
予測精度が最も良いが、
Nが小さいと
予測程度が最も悪い

※各々の値は，すべての
数値計算ライブラリの
実測結果の平均値

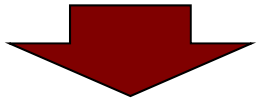
評価結果 － 選択を誤ったときのペナルティ －

－ 終了判定４回連続の場合，全216パタン中28パタンが誤選択－



評価結果 － 他コスト定義関数との比較 －

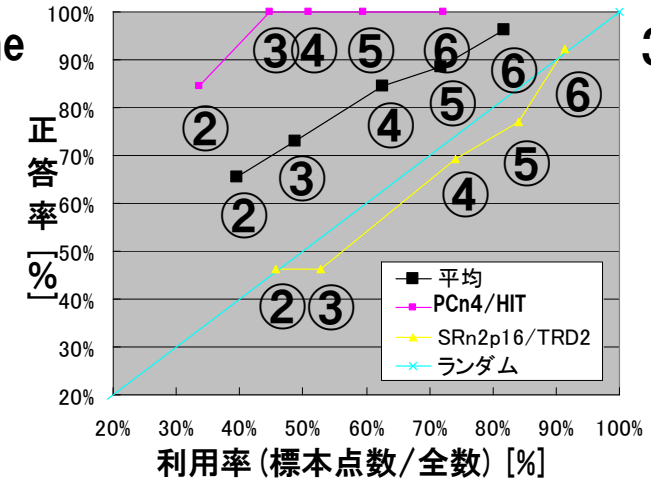
- コスト定義関数の選択
  （1）少ない標本点から推定できること
  （2）再計算のための計算コストが小さいこと



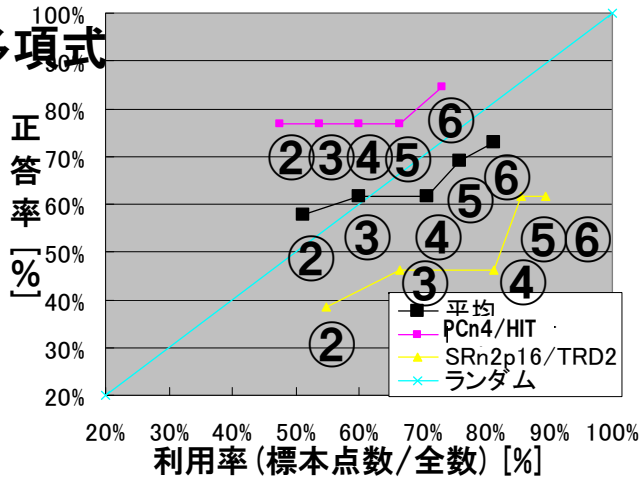
（1）３次多項式近似
（2）５次多項式近似
（3）３次スプライン補間【補間】
（4）ランダム

評価結果 － 他コスト定義関数との比較 －

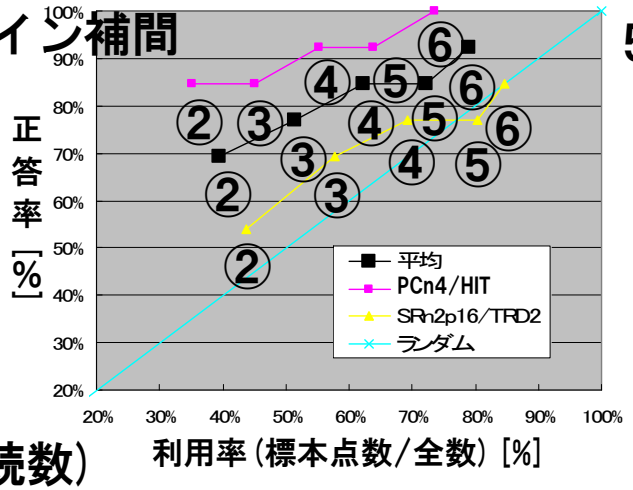
d-Spline



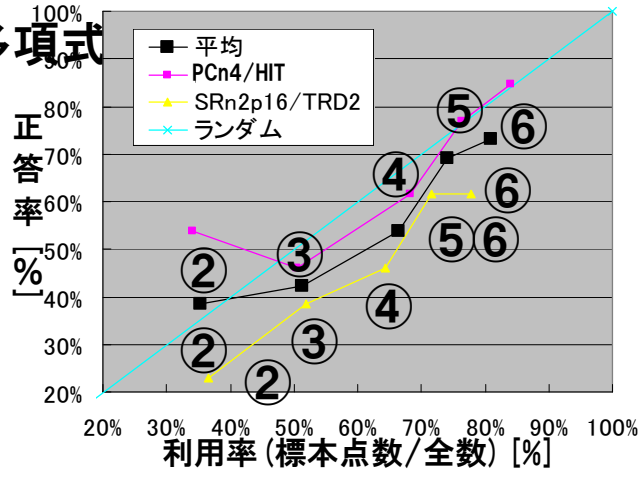
３次多項式



３次スプライン補間



５次多項式

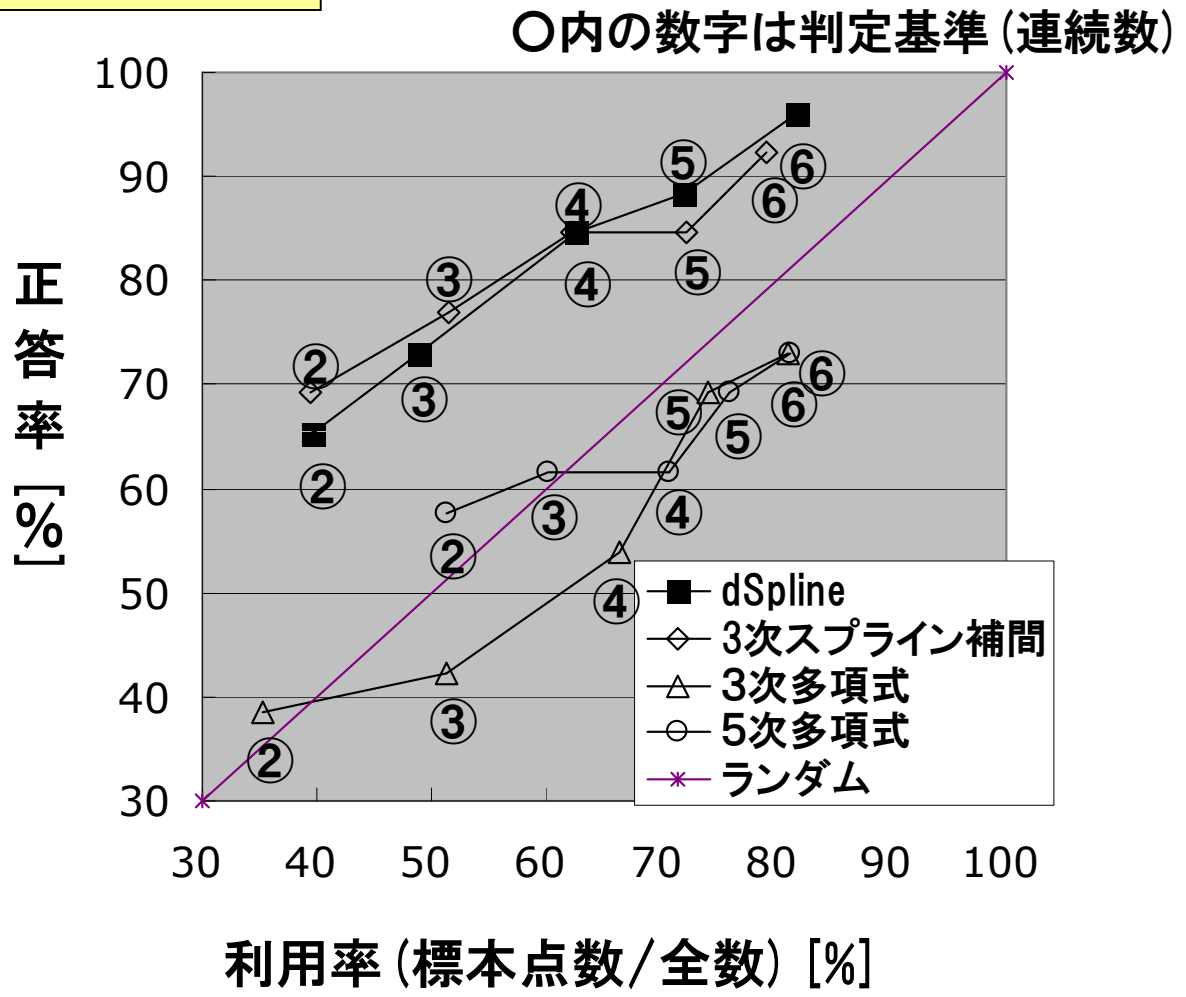


2パタン
PCn4/HITと
SPn2p16/TRD2
の平均

○内の数字は
判定基準(連続数)

評価結果 － 他コスト定義関数との比較 －

2パタンPCn4/HITと
SPn2p16/TRD2の平均



評価結果　３次スプライン補間との比較

－ 選択を誤ったときのペナルティ －

最適値に対する実行時間増加の割合

平均値	連続3回	連続4回	連続5回
d-Spline	3.7%	3.8%	4.0%
3次スプライン補間	5.1%	5.0%	5.1%
最大値	**連続3回**	**連続4回**	**連続5回**
d-Spline	22.0%	12.6%	12.6%
3次スプライン補間	27.3%	27.3%	22.2%

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- **第二部：疎行列反復解法ライブラリへの適用事例**
  - **疎行列反復解法適用への問題点**
  - **開発事例：**OpenATLib とXabclib
  - **性能評価**
- **まとめ**

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- **第二部：疎行列反復解法ライブラリへの適用事例**
  - **疎行列反復解法適用への問題点**
  - **開発事例：**OpenATLib と Xabclib
  - **性能評価**
- **まとめ**

疎行列反復解法へのAT適用

- 数値計算ライブラリレベルでは、
  ユーザが入力する疎行列の
  - 非零要素の分布
  - 数値特性

  が、事前に不明

  →実行時ATが必要
- 実行時に性能パラメタ推定をするため、
  計算量の少ない手法が必要

疎行列ソルバの階層性と機能

- 必要なAT機能は，数値計算ライブラリ中の階層で異なる
  - **メタ・ソルバ階層機能：**
    アプリケーションプログラムに最も近い機能群
  - **外部ソルバ階層機能：**
    メタ・ソルバ階層中で使われている機能群
  - **内部ソルバ階層機能：**
    外部ソルバ階層で使われている機能群

メタ・ソルバ階層

1. エンドユーザによる最適化方針の指定
   - (i)実行時間
   - (ii)メモリ量
   - (iii)利用するCPU数などの計算機資源
   - (iv)演算精度

   のうちどれを選ぶか，もしくは複数の方針を設定した場合はその比重の指定.
2. ATを行うタイミングおよび頻度の指定
3. エンドユーザによるヒント
   - 疎行列形状情報の指定.
4. 反復解法の選択
   - GMRES(m)法やIDR(s)法などの数値解法の指定.

外部ソルバ階層

1. 高性能な疎行列BLASの実装選択
   - 疎行列 - ベクトル積（SpMxV）の実装法.
   - アンローリング段数、ブロック幅.
   - 7点差分など，特定の疎行列形状に特化した実装用データ構造への変換.
2. 前処理方式の選択
   - ILU や SPAIの選択．前処理に付属するパラメタ.
     - フィルインの深さ，零とみなす許容値.
3. 数値解法上の基本演算の選択
   - Gram-Schmidt直交化の実装方式.

内部ソルバ階層

1. 解法に特有のパラメタの設定
   - リスタート周期の値.
2. 疎行列データ構造に依存する実装方式
   - CRS形式用かJDS形式用か.

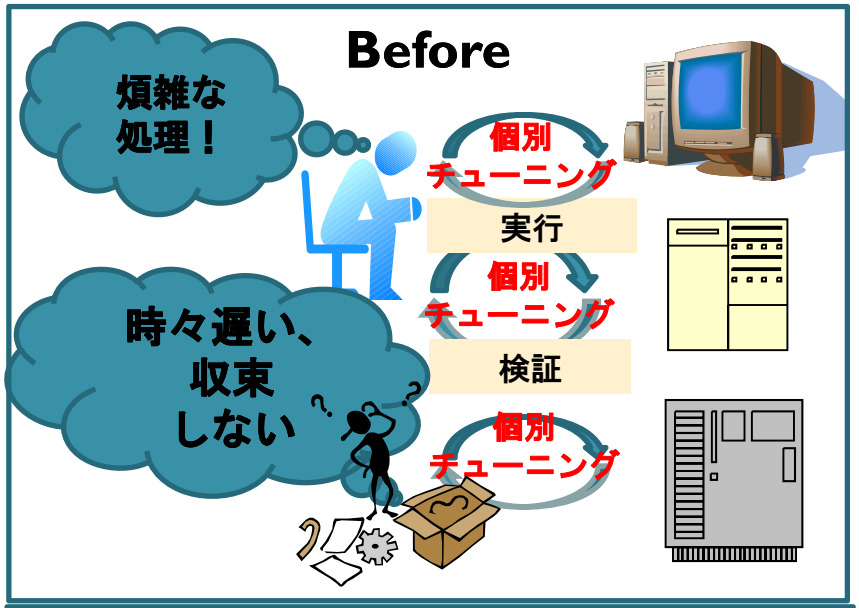
発表の流れ

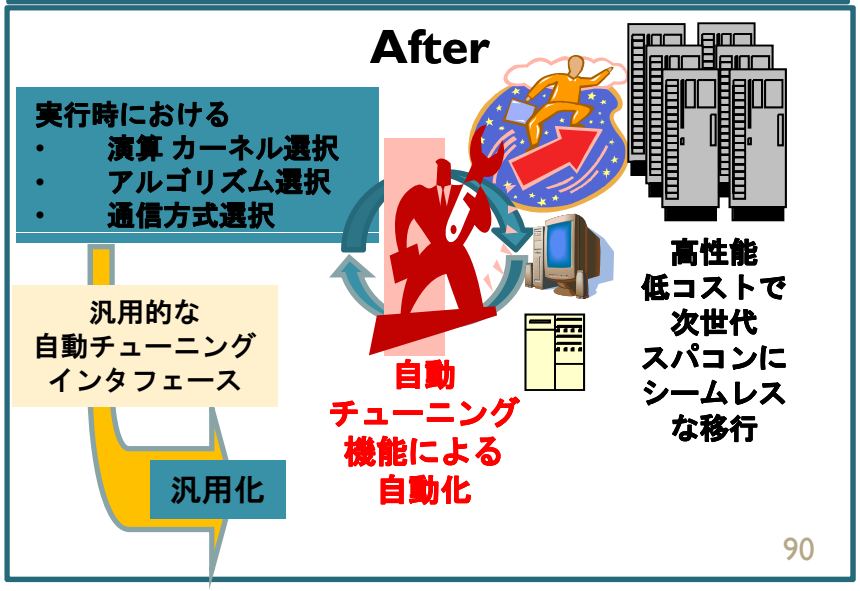
- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- **第二部：疎行列反復解法ライブラリへの適用事例**
  - 疎行列反復解法適用への問題点
  - **開発事例：OpenATLib とXabclib**
  - **性能評価**
- **まとめ**

疎行列反復解法ライブラリXABCLIB(OPENMP並列化版)

e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発
高生産・高性能計算機環境実現のためのシステムソフトウェアの研究開発
「シームレス高生産・高性能プログラミング環境」高性能高可搬性ライブラリ
（平成20年度～平成23年度）：代表　東京大学　石川裕　教授

- 問題
  - 最適化が職人芸に依存 、生産性がない、可搬性がない、 煩雑な処理、コストがかかる
  - 理論的に範囲外のパラメタ指定による収束失敗
- 目標
  - 高生産性／性能移植性をもつ数値計算ライブラリの提供
- 疎行列の非ゼロ成分の分布を考慮し
  実行時における以下の自動チューニング機能を提供
  1. ***演算カーネル選択***
  2. ***数値アルゴリズム選択***
  3. ***並列実装方式選択***
  4. ***汎用的なAPI （OpenATLib）***





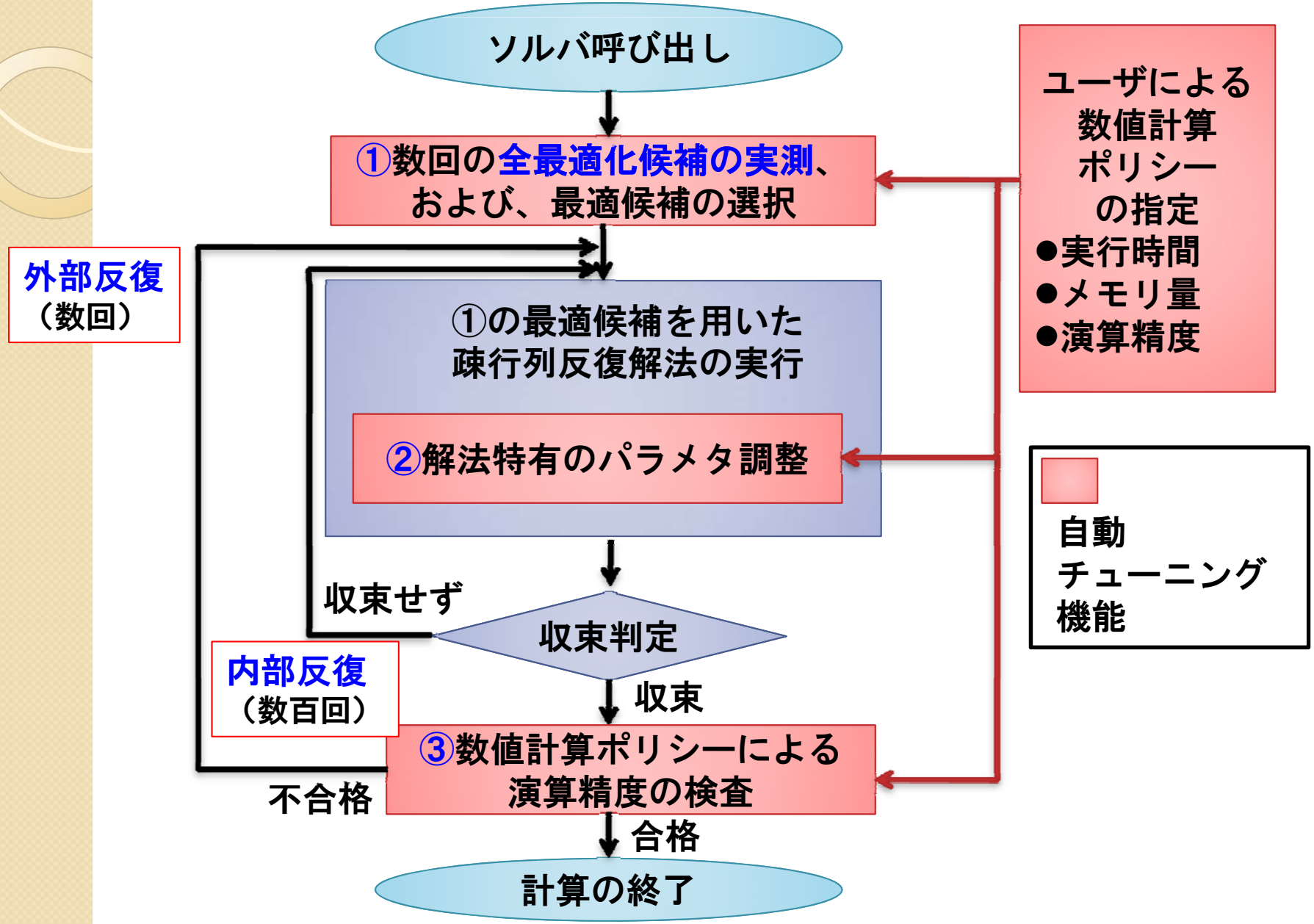
XabclibとOpenATLib

- **Xabclib**
  - 自動チューニング機能付き数値計算ライブラリ
  - 数値シミュレーションユーザ向け
  - 対称疎行列用　固有値計算
  - 非対称疎行列用　連立一次方程式の求解
  - OpenATLibを用いて構築されている
  - OpenATLib上ではメタ・インタフェースとして位置づけ（後述）
- **OpenATLib**
  - 自動チューニング機能のインターフェース集
  - 数値解析研究者、数値計算ライブラリ開発者向け
  - 例：疎行列-ベクトル積（SpMxV）ルーチン
- 双方とも、OpenMP実装のみ提供（β版）

OpenATLibの設計方針

- 実行時に行う自動チューニング
  インターフェースを（疎行列を扱う）
  ATライブラリ開発者やユーザに提供
- 以下の性能パラメタ最適化機能を提供
  1. アルゴリズム選択処理
     - Krylov部分空間のリスタート周期選択
       （内部ソルバ階層機能）
     - 直交化方式選択
       （外部ソルバ階層機能）
  2. 実装方式選択
     - 疎行列 - ベクトル積 (SpMxV)（外部ソルバ階層機能）
       - 負荷バランス改良、ベクトル計算機向け、並列化向き
  3. 計算機資源選択
     - 実行時のメモリ量・コア数などを考慮したSpMxVと
       リスタート周期（メタ・ソルバ階層機能）
- ユーザポリシ設定機能（メタ・ソルバ階層機能）
  - 実行速度、メモリ量、演算精度

OpenATLibの自動チューニング概要（実行モデル）



OpenATLib (β版)の提供関数

関数名	機能の説明
OpenATI_DAFRT	Krylov部分空間のリスタート周期を増加するか判定
OpenATI_DSRMV	CRS形式において、倍精度演算　対称行列用の 疎行列ーベクトル積で、最もよい実装方式の選択
OpenATI_DURMV	CRS形式において、倍精度演算　非対称行列用の 疎行列ーベクトル積で、最もよい実装方式の選択
OpenATI_DSRMV_Setup	OpenATI_DSRMVのための初期データ設定処理
OpenATI_DURMV_Setup	OpenATI_DURMVのための初期データ設定処理
OpenATI_DAFGS	Gram-Schmidt直交化処理において、 ４種類の実装から最もよい実装方式の選択
OpenATI_BLDATA	デフォルトデータ設定処理 (Fortran言語のBlock data format)
OpenATI_LINEARSOLVE	数値計算ポリシを適用する連立一次方程式ソルバ用 メタ・インタフェース
OpenATI_EIGENSOLVE	数値計算ポリシを適用する固有値ソルバ用 メタ・インタフェース

※Xabclib（β版）で初めて提供した関数

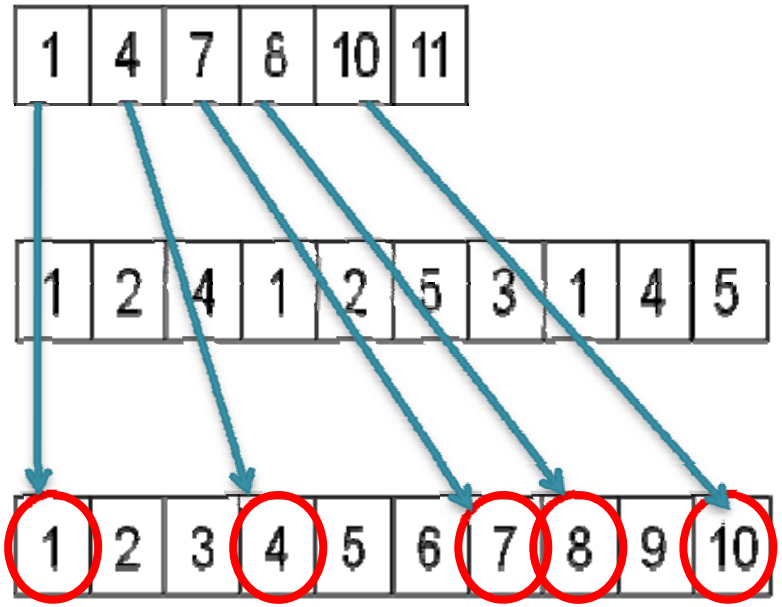
Xabclibの疎行列データ構造：行圧縮形式（CRS Format）

$$\begin{pmatrix} 1 & 2 & 0 & 3 & 0 \\ 4 & 5 & 0 & 0 & 6 \\ 0 & 0 & 7 & 0 & 0 \\ 8 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{pmatrix}$$

Pointers to first row elements.

Row indexes for non-zero elements.

Values for non-zero elements.



○ 各行で最初の非零要素

OpenATlib:リスタート周期調整関数

- **OpenATI_DAFRT(NSAMP, SAMP, IRT,INFO)**
  - NSAMP:サンプリング点の数
  - SAMP: サンプリングデータ (double)
  - IRT: 判断フラグ
    - 0 : 増加させる必要なし
    - 1 : 増加させる必要あり
  - **判断関数:　MM比** [T.Sakurai,2008]

$$R\_i\ (s, t) = max\_z\ \{\ r\_i\ (z);\ z=s-t+1,\ldots,s\}$$
$$/\ min\_z\ \ \{r\_i\ (z);\ z=s-t+1,\ldots,s\}$$

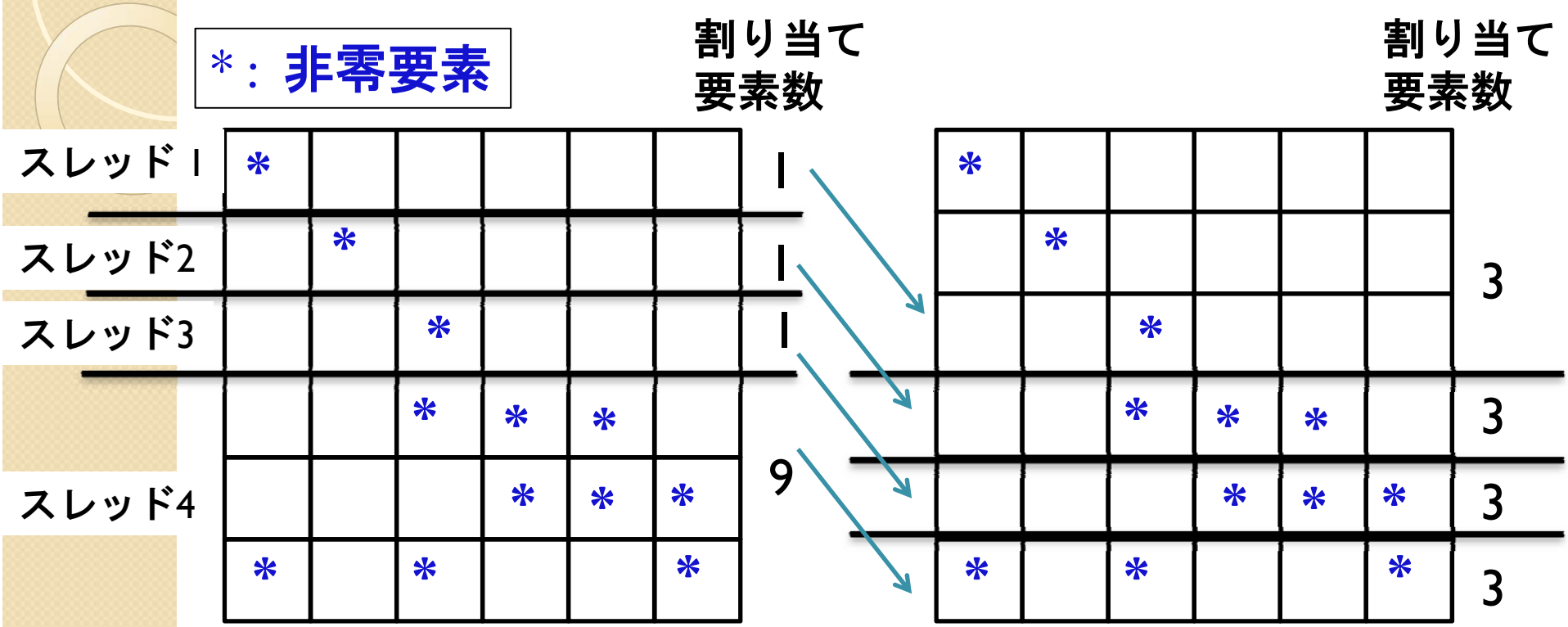
  - 直近のs個のサンプリング点r_i （通常は残差のノルム）について、残差の最大と最小の比（MM比）を検出する。
    →最新の S個の残差を実行時にモニタする。
  - MM比が小さい→残差が停滞→リスタート周期を増加する

疎行列-ベクトル積の実装方式

- OpenATI_DSRMV (対称疎行列用 API)
  - S1) 行分割方式 (RDM)
  - S2) 非零要素均等化方式 (NNZ)(負荷バランス化RDM)
  - S3) 並列リダクション時に零演算を除去したRDM
- OpenATI_DURMV (非対称疎行列用 API)
  - U1)行分割方式 (RDM)
  - U2)非零要素均等化方式 (NNZ)(負荷バランス化RDM)
  - U3) Branchless Segmented Scan(BSS)法 [1]
    (スカラマシン用)
  - U4) オリジナルSegmented Scan法 [2]
    (ベクトルマシン用)

[1] Takao Sakurai, et.al. SIAM PP2010, MS6 (2010)
[2] Guy E. Blelloch, Michael A. Heroux, and Marco Zagha: Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors, Technical Report of CMU-CS-93-173 (August 1993)

行分割方式(RDM)と非零要素均等化方式(NNZ) (対称と非対称行列双方)



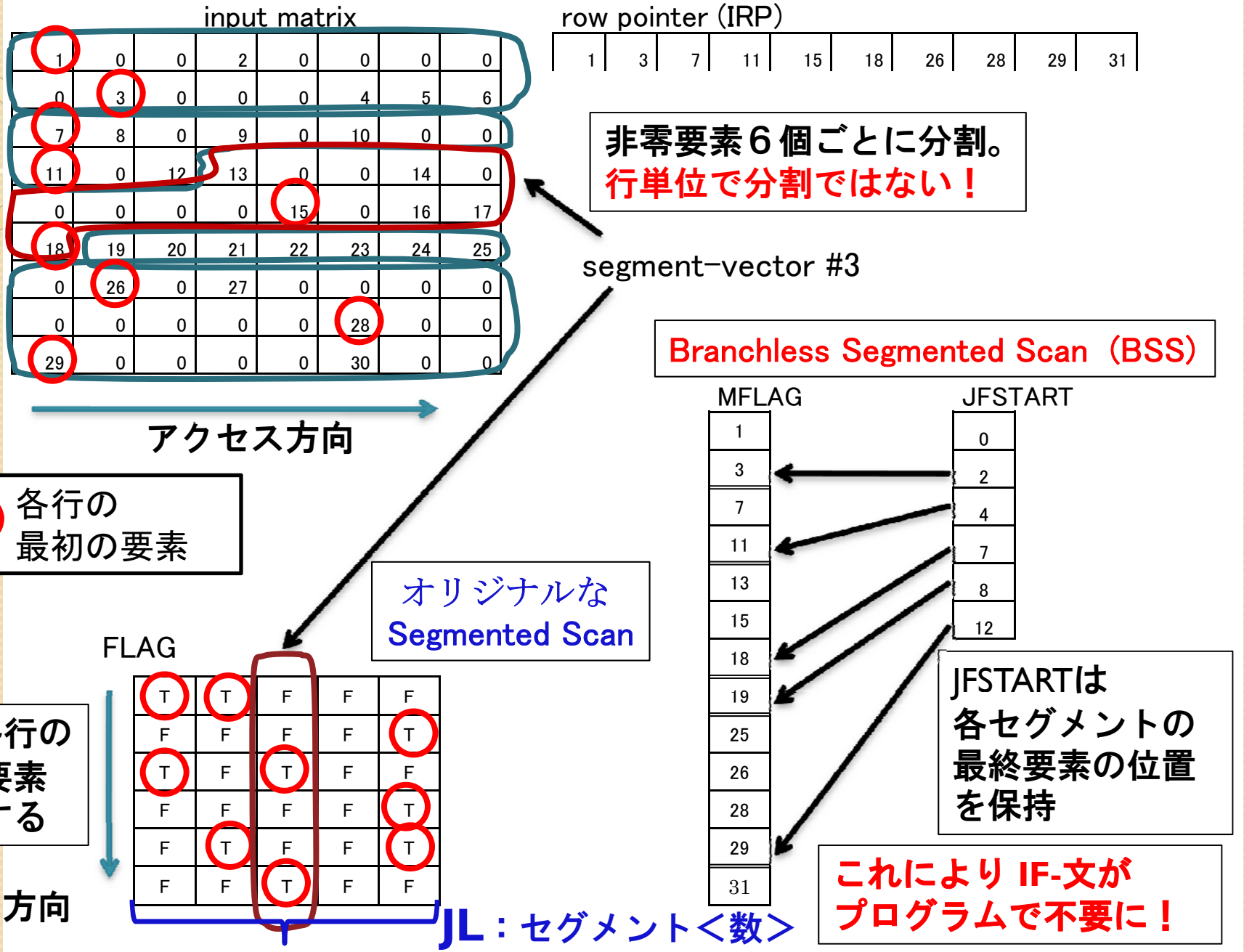
行分割方式（RDM）
(SW: S1, U1)

それぞれのスレッドでの割り当て
行数を均等化するように分割。

非零要素均等化方式（NNZ）
(SW: S2, S3, U2)

それぞれのスレッドで割当てられる
非零要素数を均等化するように
割り当て行数を決定。

Segmented Scan法とBSS法 (非対称行列用)

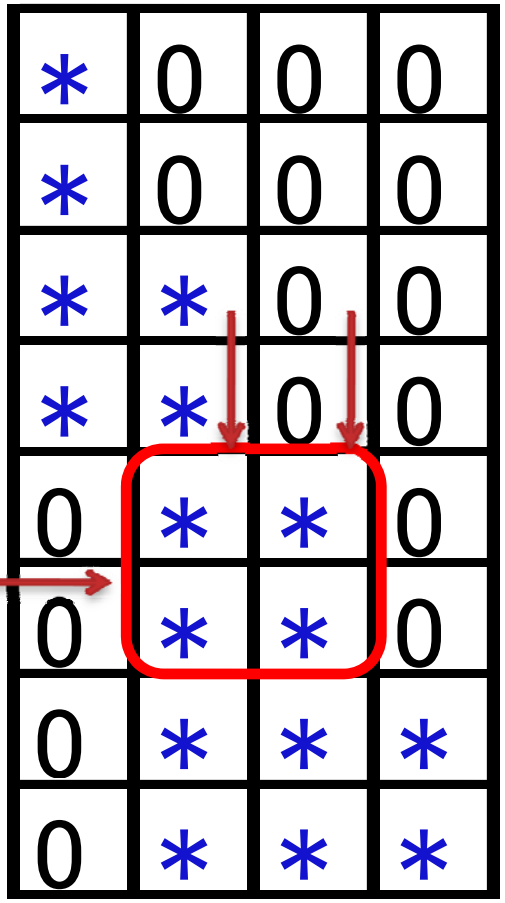


SpMxV の無駄な計算の削減方式

(Sw: S3, 対称行列のみ)

- 疎行列形状によっては、リダクション演算がほとんど零要素の加算になる
- 無駄な零要素の加算を避けるため、非零要素の値の開始列と終了列の情報を所有。

WK
(並列化のためのワークスペース)



- ここの部分のリダクションは第２行と第３行のみ

反復解法中の再直交化方式の自動チューニング（数値ポリシ依存）

1. 古典 Gram-Schmidt法 (CGS)
   - 直交精度は低い
   - 並列性は高い
2. DGKS法
   - CGSを残差をみたうえ、 2回実行する
3. 修正 Gram-Schmidt法 (MGS)
   :デフォルト
   - 直交精度が、多くの場合高い
   - 並列性は低い
4. ブロック化 Gram-Schmidt法 (BCGS)
   - 内部ブロックの再直交化にはＣＧＳを使い、外部ブロックの再直交化にはＭＧＳを使う。
   - ブロック幅は４（本実装の制約）。

OpenATLibにおけるAT機能

●性能パラメタと定義域

$$PP = RP = \{DSRMV, DURMV, DAFGS, DAFRT\}$$

$$BP = \{N\}$$

$$DSRMV \in \{S1, S2, S3\}$$

$$DURMV \in \{U1, U2, U3, U4\}$$

$$DAFGS \in \{CGS, DGKS, MGS, BCGS\},$$

$$DAFRT \in [1, MAXM]$$

ユーザポリシ記述（１）

- 全体形式

<keyword> = <value>

<keyword> : = **POLICY / CPU / RESIDUAL / MAXMEMORY / MAXTIME / PRECONDITIONER**

- 最適化ポリシ指定機能

POLICY = <value>

<value> := TIME / ACCURACY / MEMORY

“TIME”がデフォルト

- **POLICY = TIME**
  - メタ・インターフェース性能を、実行時間の観点で最適化する。最も高速なアルゴリズムが選択される。
- **POLICY = ACCURACY**
  - メタ・インターフェース性能を、解の精度の観点から最適化する。偽収束が生じる場合、要求精度を満たすまで、内部反復で再実行する。
- **POLICY = MEMORY**
  - メタ・インターフェースの性能を、メモリ使用量の観点から最適化する。

ユーザポリシ記述（２）

- CPU数指定機能

CPU = <value>
<value> ：= 実行時の [スレッド数] を記述
OMP_GET_NUM_THREADS　をデフォルトで設定。
注意） 1 ≦ <value> ≦ OMP_GET_MAX_THREADS()

- 残差精度指定機能

RESIDUAL = <value>
<value>：= 倍精度で [要求精度] を記述
デフォルトは 1.0D-8。
“POLICY = ACCURACY” の場合で、偽収束が生じると、
ここで与えられた要求精度を満たすまで内部反復する。

- 最大メモリ量指定機能

MAXMEMORY = <value>
<value>：= 許容されるメモリ容量を[Gbyte]で記述
デフォルトは /proc/meminfo (Linux)の値。
もし /proc/meminfo　の値の取得ができない場合は、実行時配列確保を行った結果から、数値設定する。
注意) MAXMEMORY の上限は 16Gbyte.（本実装の制約）

ユーザポリシ記述（３）

- 最大実行時間指定機能

MAXTIME = <value>
<value>：= 最大許容実行時間[秒]を設定
デフォルトは、無制限。
指定時間を反復中に過ぎた場合、計算を停止する。

- 前処理方式指定機能

PRECONDITIONER = <value>
<value>：= NO / JACOBI / SSOR / ILU0
ILU(0) がデフォルト。
本機能は、OpenATI_LINEARSOLVEのみで利用。

●PRECONDITIONER = NO
：前処理なし
●PRECONDITIONER = JACOBI
：JACOBI前処理
●PRECONDITIONER = SSOR
：SSOR前処理
●PRECONDITIONER = ILU0
：ILU(0)前処理

OpenATLibにおけるコスト定義関数

●コスト定義関数

$$F_T(PP) = \text{実測時間}$$

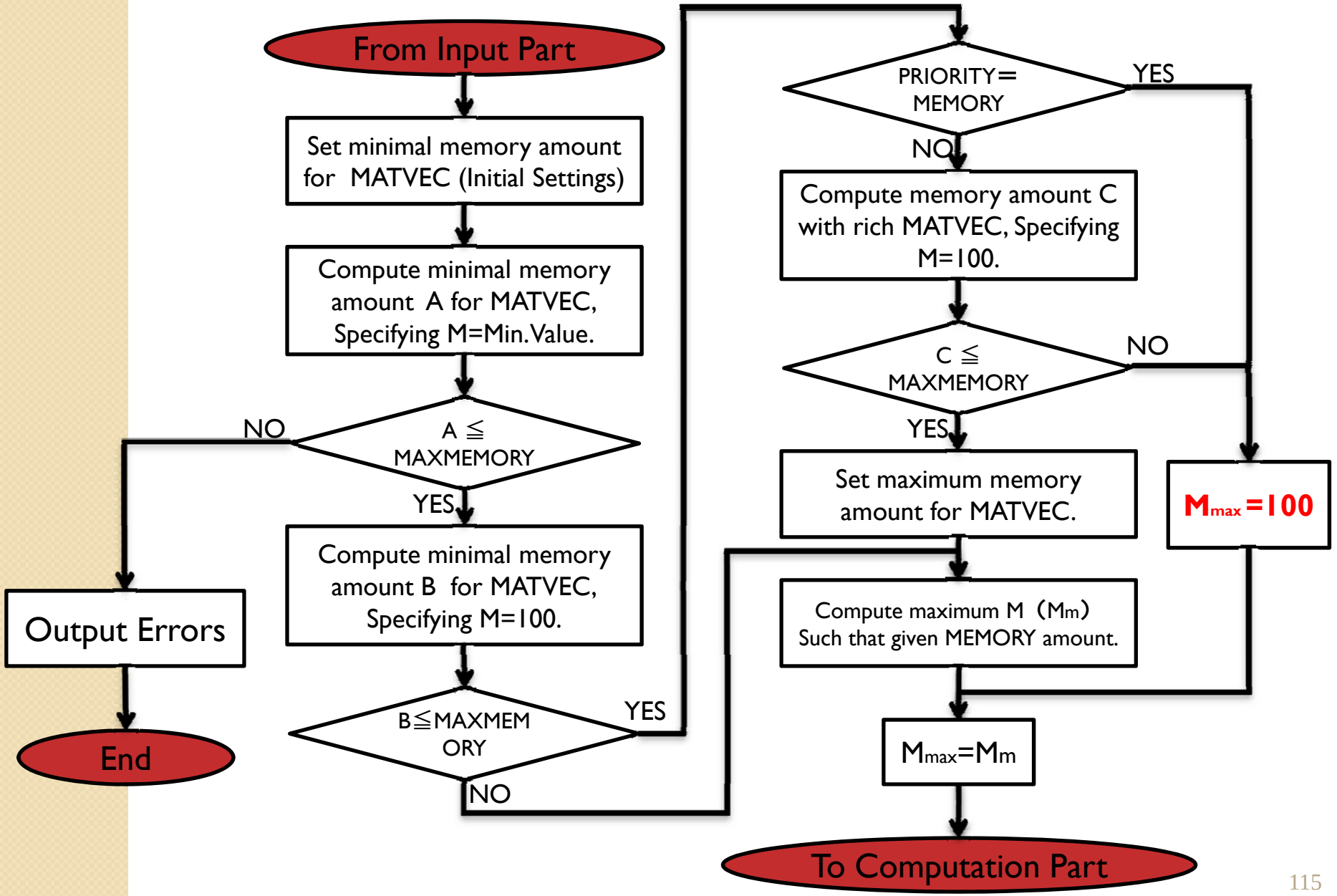
$$F_M(PP) = \text{利用メモリ量}$$

$$F_A(PP) = \{\text{実測時間} \mid \text{精度} < \text{要求精度}\}$$

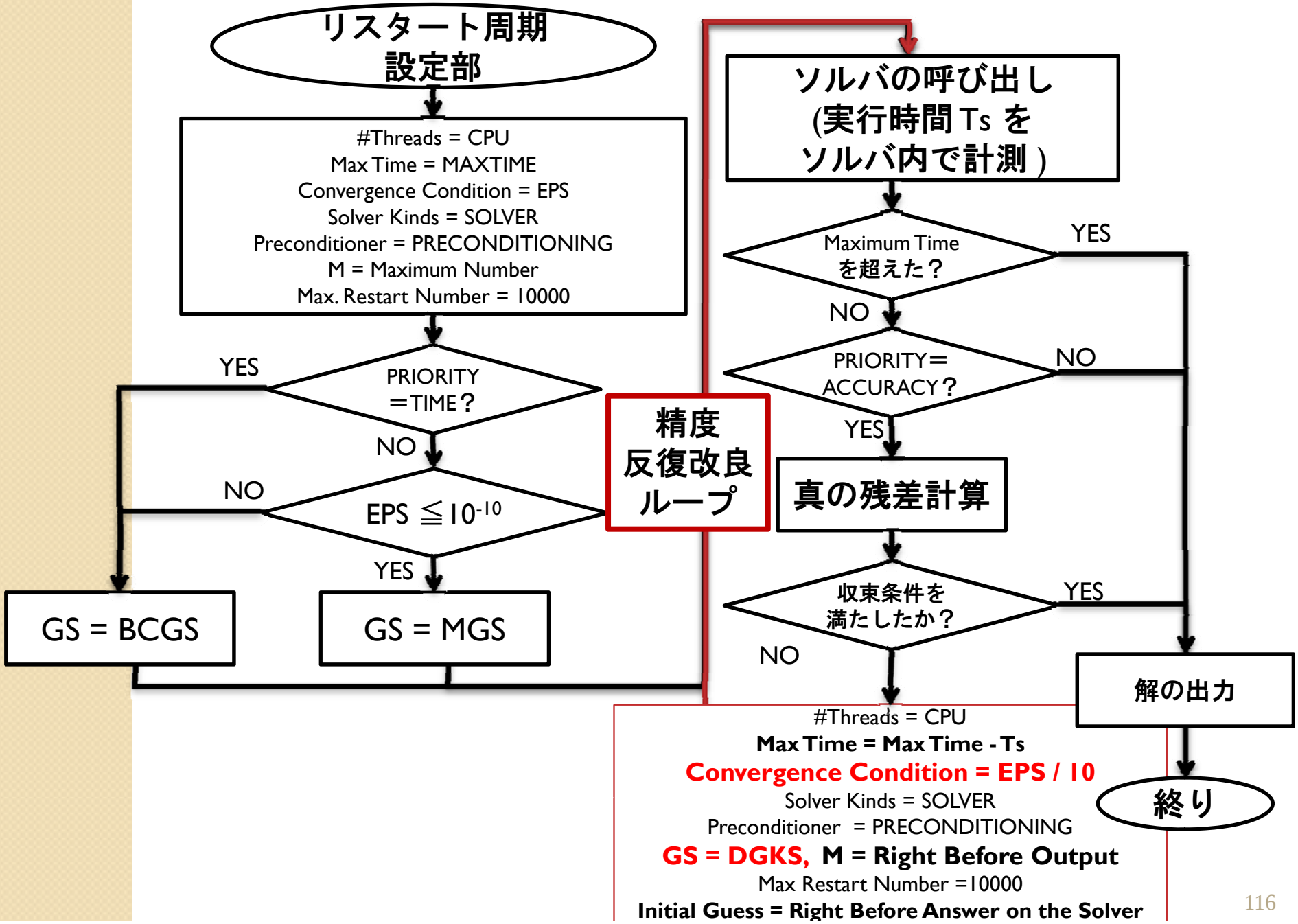
$$F(PP) = \{F_T, F_M, F_A \mid \text{ユーザの要求}\}$$

$$\min F(PP)\ s.t.\ BP = \{n\}$$

POLICY=MEMORY Strategy



演算精度ポリシーのパラメタ設定戦略



数値計算ポリシ実行　報告機能

- 右のような
  「設定ファイル」を用意

```
POLICY  =  ACCURACY
RESIDUAL  =  1.0D-10
CPU  =  16
PRECONDITIONER =  ILU0
MAXMEMORY = 1.0
MAXTIME  =  500.0
```

- 実行例（報告ファイルの中身）
  - OPENATI_POLICY_REPORT.txt

```
**************************************************
 *****  OpenATI  LINEAR SOLVER POLICY REPORT    *****
 *****                2010.0114   11:30                *****      ←report date / time
 **************************************************
 [Environment variables]                              ↓ input parameters
   OPENATI_DEBUG  =
   OPENATI_POLICY = ./input_policy.dat

 [Policy Definitions]
   POLICY       =ACCURACY
   SMPs          =          16
   SOLVER        = XABCLIB_GMRES
   PRECONDITIONER = ILU0
   REQUIREMENT WORKING MEMORY =     16.0000000000000
     <<< Upper Bound 16GBYTE >>>
   REQUIREMENT RESIDUAL       =   1.000000000000000E-008
   REQUIREMENT MAX.TIME       =    500.000000000000
   MAX. SUBSPACE SIZE   =       14214
   RUNTIME MEMORY USE   =      3.24 [GBYTE]
   KRYLOV SUBSPACE EXPAND AT =  1   ,MATVEC AT =  1
   Initial Gram-Schmidt Strategy = BCGS

 ====== OPENATI_LINEARSOLVE SUCCESSFULY ENDED ======        ↓ successfully exit
 [OPENATI_LINEARSOLVE RESULT]                               ↓ result report
   MATRIX DATA : N=        14214  NNZ=       259688
   FASTEST MATVEC NO. =          11                  ←fastest OpenATI_DURMV case
   FINAL KRYLOV SUBSPACE SIZE =         42           ←Msize for convergence
   FINAL Gram-Schmidt Strategy = DGKS
   2-Norm of RHS =   25.2388589282479                ←initial norm of RHS
   NUMBER OF RETRYED GMRES =         6               ←retried iterations
  TOTAL RESTARTS of GMRES =         197
   RESIDUAL NORM          =  3.005885687924543E-010
   SET-UP TIME          =  1.1267900466918 95E-002  [SEC]
   SOLVER TIME           =   1.32032704353333       [SEC]
   TOTAL  TIME           =   1.33159494400024       [SEC]
```

発表の流れ

- 第一部：自動チューニングとは
  - 自動チューニングとは
  - 定式化
  - 関連研究
    - 関連研究１：実行時間の多項式近似法FIBER
    - 関連研究２：d-splineによる動的標本点の追加法
- **第二部：疎行列反復解法ライブラリへの適用事例**
  - 疎行列反復解法適用への問題点
  - 開発事例：OpenATLib とXabclib
  - **性能評価**
- **まとめ**

性能評価

計算機環境：
T2Kオープンスパコン (東大版)

CPU	Quad-Core AMD Opteron(TM) Processor 8356 2.3GHz ,16 core/node
L1 Cache Size	Data: 64 Kbyte/core, Instruction: 64 Kbyte/core
L2 Cache Size	512 Kbyte/core
L3 Cache Size	2 MByte/4core
主記憶	32GByte（8GBytes/Socket）
OS	Red Hat Enterprise Linux 5
コンパイラ	Intel Fortran Compiler Professional Version 11.0
コンパイラ オプション	–O3　–m64　–openmp　–mcmodel=medium

数値実験の条件

●OpenATI_EIGENSOLVE（固有値ソルバ）

ユーザからの要求精度	1.0E-08
計算する固有値数	10
最大実行時間	1000 [秒]

●OpenATI_LINEARSOLVE（連立一次方程式ソルバ）

ユーザからの要求精度	1.0E-08
右辺ベクトルの値	**全ての要素が 1**
初期ベクトルの値	**全ての要素が 0**
前処理方式	ILU(0)
最大実行時間	1000 [秒]

●**従来法のデフォルトのリスタート周期**

30 ：PETScのデフォルト値と同じ

OpenATI_EIGENSOLVEのテスト行列

- University Florida Sparse Matrix Collection (21種、対称行列)

Matrix	N	NNZ	Field
vibrobox	12328	177578	Acoustics
Lin	256000	1011200	Chemistry
cfd1	70656	949510	Fluid Dynamics
cfd2	123440	1605669	
gyro	17361	519260	Model Reduction
t3dl	20360	265113	
c-71	76638	468096	Optimization
c-73	169442	724348	
Si5H12	19896	379247	Structural
SiO	33401	675528	
dawson5	51537	531157	

Matrix	N	NNZ	Field
H2O	67024	1141880	Structural
F2	71505	2682895	
oilpan	73752	1835470	
shipsec1	140874	3977139	
bmw7st_1	141347	3740507	
SiO2	155331	5719417	
shipsec5	179860	5146478	
Si41Ge41H72	185639	7598452	
bmw3_2	227362	5757996	
Ga41As41H72	268096	9378286	

OpenATI_LINEARSOLVEのテスト行列

- University Florida Sparse Matrix Collection (22種、非対称行列)

Matrix	N	NNZ	Field
chipcool0	20082	281150	2D/3D
chem_master1	40401	201201	
torso1	116158	8516500	
torso2	115067	1033473	
torso3	259156	4429042	
memplus	17758	126150	Electric circuit
ex19	12005	259879	Fluid dynamics
poisson3Da	13514	352762	
poisson3Db	85623	2374949	
airfoil_2d	14214	259688	
viscoplastic2	32769	381326	Materials

Matrix	N	NNZ	Field
xenon1	48600	1181120	Materials
xenon2	157464	3866688	
wang3	26064	177168	Semiconductor device
wang4	26068	177196	
ecl32	51993	380415	
sme3Da	12504	874887	Structural
sme3Db	29067	2081063	
sme3Dc	42930	3148656	
epb1	14734	95053	Thermal
epb2	25228	175027	
epb3	84617	463625	

Xabclib_LANCZOSにおけるリスタート周期AT(Ver.Alpha)

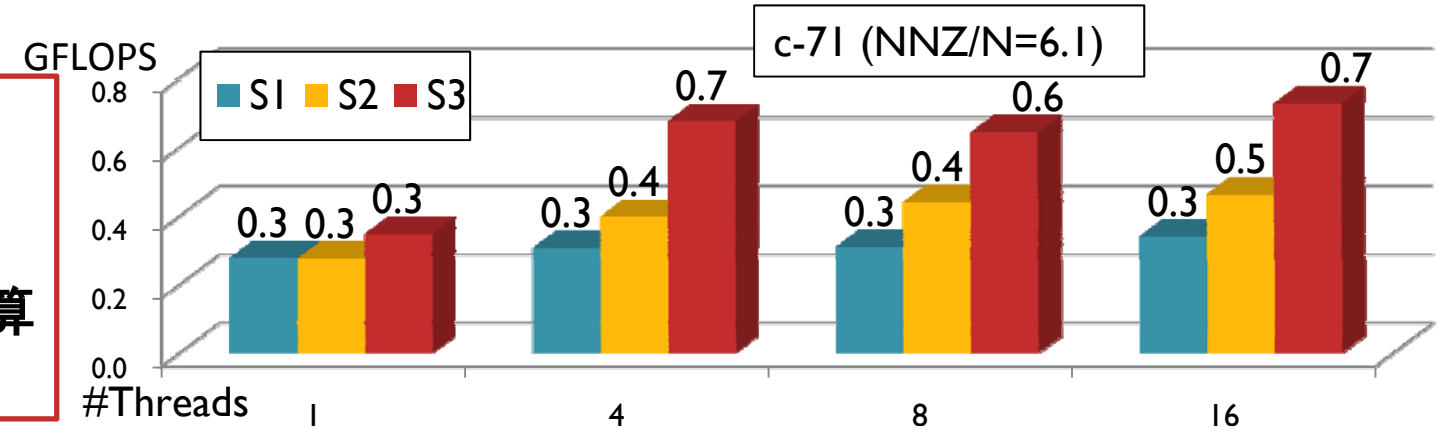
Matrices	Fixed Restart Frequency			Auto-tuning			Speedup with AT
	M	#Restart	time(sec)	Final M	#Restart	time(sec)	
vibrobox	30	24	1.13	35	20	1.04	1.09
Lin	30	1047	601.65	150	54	214.49	2.81
cfd1	30	55	12.42	80	24	11.16	1.11
cfd2	30	45	18.69	70	28	21.31	0.88
gyro	30	10	1.13	35	16	1.43	0.79
t3dl	30	1878	125.62	90	33	6.69	18.79
c-71	30	4	0.70	25	17	1.38	0.51
Si5H12	30	192	17.99	115	34	9.34	1.93
SiO	30	161	24.32	100	37	13.72	1.77
dawson5	30	1052	135.75	105	34	14.79	9.18
H2O	30	623	168.46	130	40	42.01	4.01
F2	30	27	15.04	50	21	15.42	0.97
oilpan	30	28	10.63	45	24	11.97	0.89
shipsec1	30	26	20.89	50	30	30.58	0.68
bmw7st_1	30	1	1.06	15	5	1.27	0.83
SiO2	30	699	805.68	145	45	207.74	3.88
shipsec5	30	53	59.31	75	27	60.41	0.98
Si41Ge41H72	30	411	679.91	150	40	274.68	2.48
bmw3_2	30	3	4.20	25	19	12.93	0.33
Ga41As41H72	30	10000	**NOT CONV.**	150	44	204.05	**Converged**

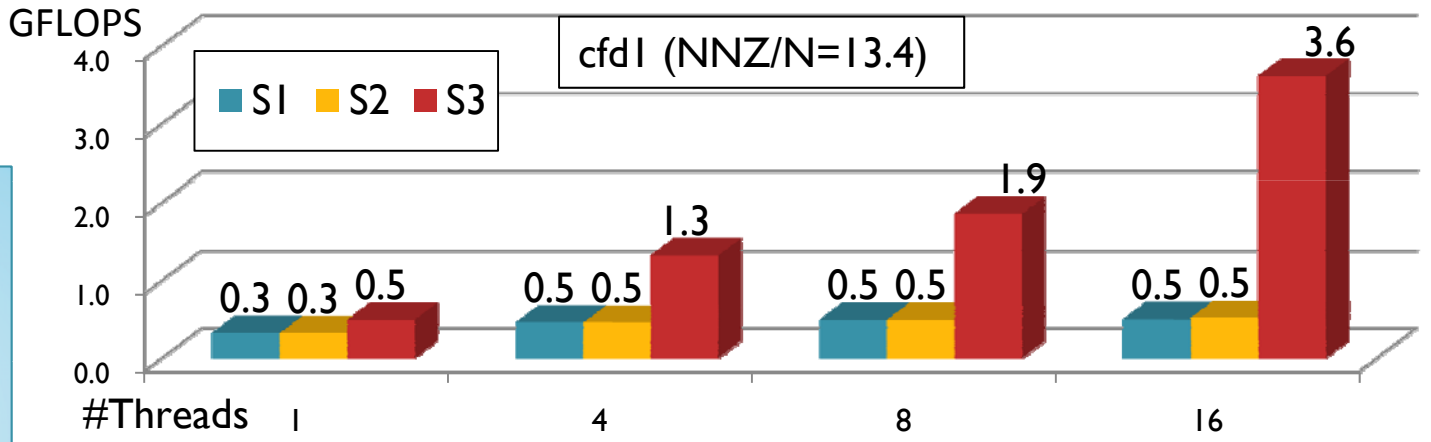
Xabclib_GMRESにおけるリスタート周期AT (Ver. Alpha)

Matrices	Fixed Restart Frequency			Auto-tuning			Speedup with AT
	M	#Restart	time(sec)	Final M	#Restart	time(sec)	
chem_master1	30	22	2.55	52	14	2.25	1.13
torso2	30	1	0.68	7	2	0.31	2.19
torso1	30	1	2.54	2	1	0.72	3.53
torso3	30	12	33.57	32	14	34.11	0.98
memplus	30	5	0.25	22	10	0.20	1.25
ex19	30	1000	**NOT CONV.**	100	60	26.23	**Converged**
poisson3Da	30	3	0.48	17	7	0.54	0.89
airfoil_2d	30	7	0.73	22	14	0.83	0.88
poisson3Db	30	7	10.38	17	14	11.03	0.94
viscoplastic2	30	19	2.93	37	15	1.70	1.72
xenon1	30	30	20.16	62	19	16.18	1.25
xenon2	30	40	92.96	72	20	64.29	1.45
wang4	30	5	0.37	17	9	0.29	1.28
ecl32	30	1000	**NOT CONV.**	92	22	11.61	**Converged**
sme3Da	30	670	215.49	100	90	101.33	2.13
sme3Db	30	1000	**NOT CONV.**	100	120	377.29	**Converged**
sme3Dc	30	1000	**NOT CONV.**	100	122	575.63	**Converged**
epb1	30	11	0.38	32	14	0.35	1.09
epb2	30	3	0.22	12	9	0.21	1.05
epb3	30	11	3.22	42	14	3.02	1.07

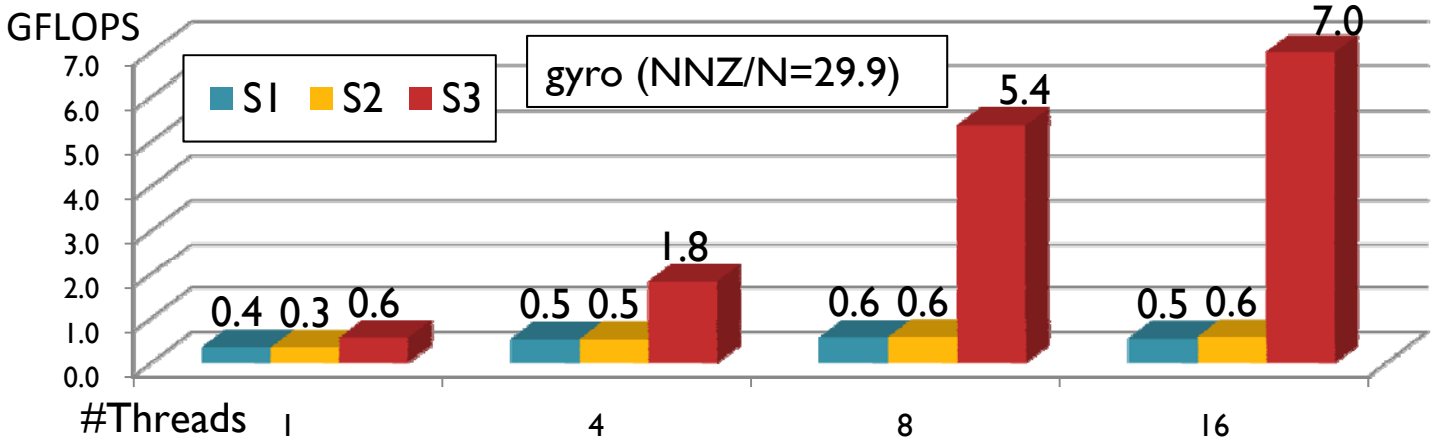
典型的な対称行列に対する疎行列-ベクトル積性能

S1: RDM
S2: NNZ
S3 : S2+
零要素計算
フリー





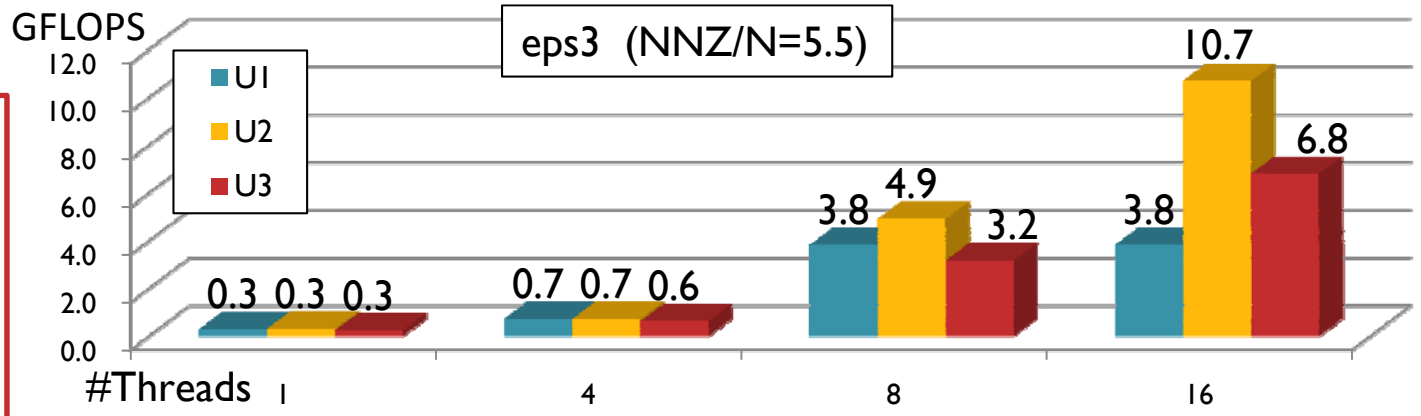
S3：
零要素
計算
フリーな
アルゴリズ
ムのみが、
台数効果
を得る。

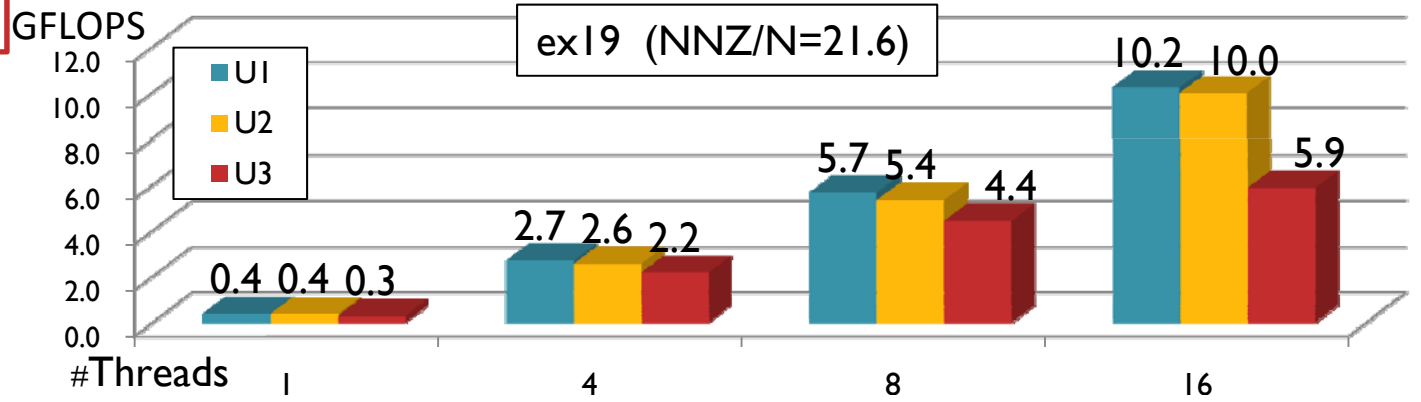


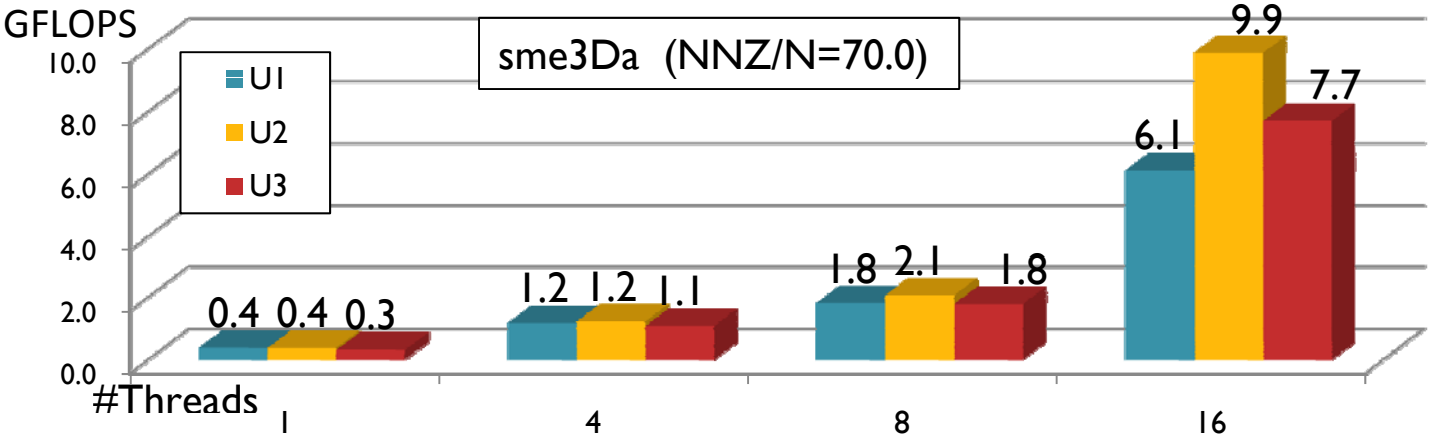
典型的な非対称行列の疎行列-ベクトル積性能

U1: 行分割
U2: 非零要素均等化
U3 : BSS

U2: **非零要素均等化方式**が多くの場合、よい性能を達成。







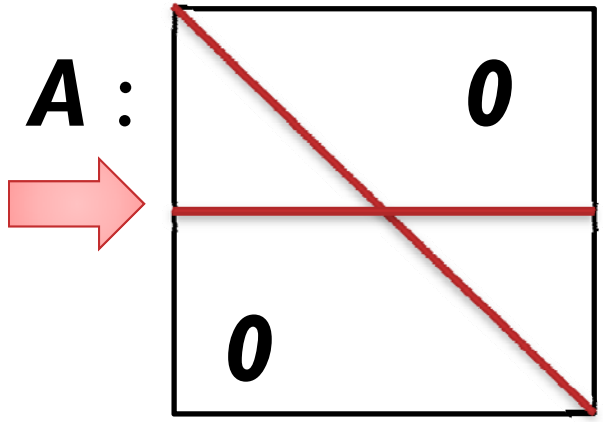
行分割方式で並列化できない例

- 以下の行列

$$A = (a_{i,j}), (i,\ j = 1,2,\ldots,\ N)$$

$$\begin{cases} if\ \ ((i = j).and.(i.ne.N/2))\ then\ \ \#non-zero\ = 1. \\ if\ (i = N/2)\ then\ \ \#non-zero\ = N. \\ The\ elements\ \ are\ generated\ \ by\ rand\ (). \end{cases}$$

第N/2行のみが密な行



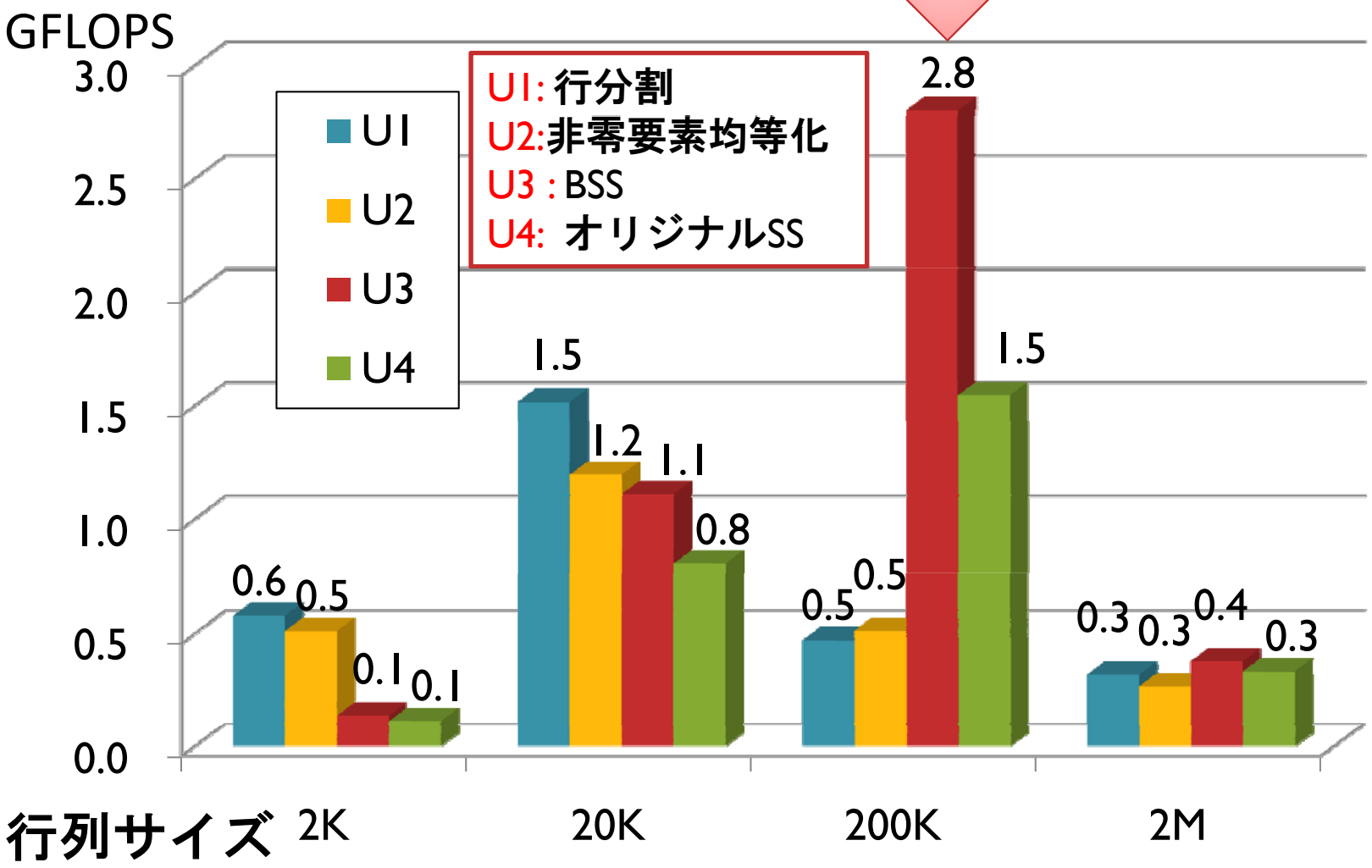
その他の行の非零要素数は１

●従来の行分割による計算方式では、
第Ｎ／２行の計算について並列性が抽出できない

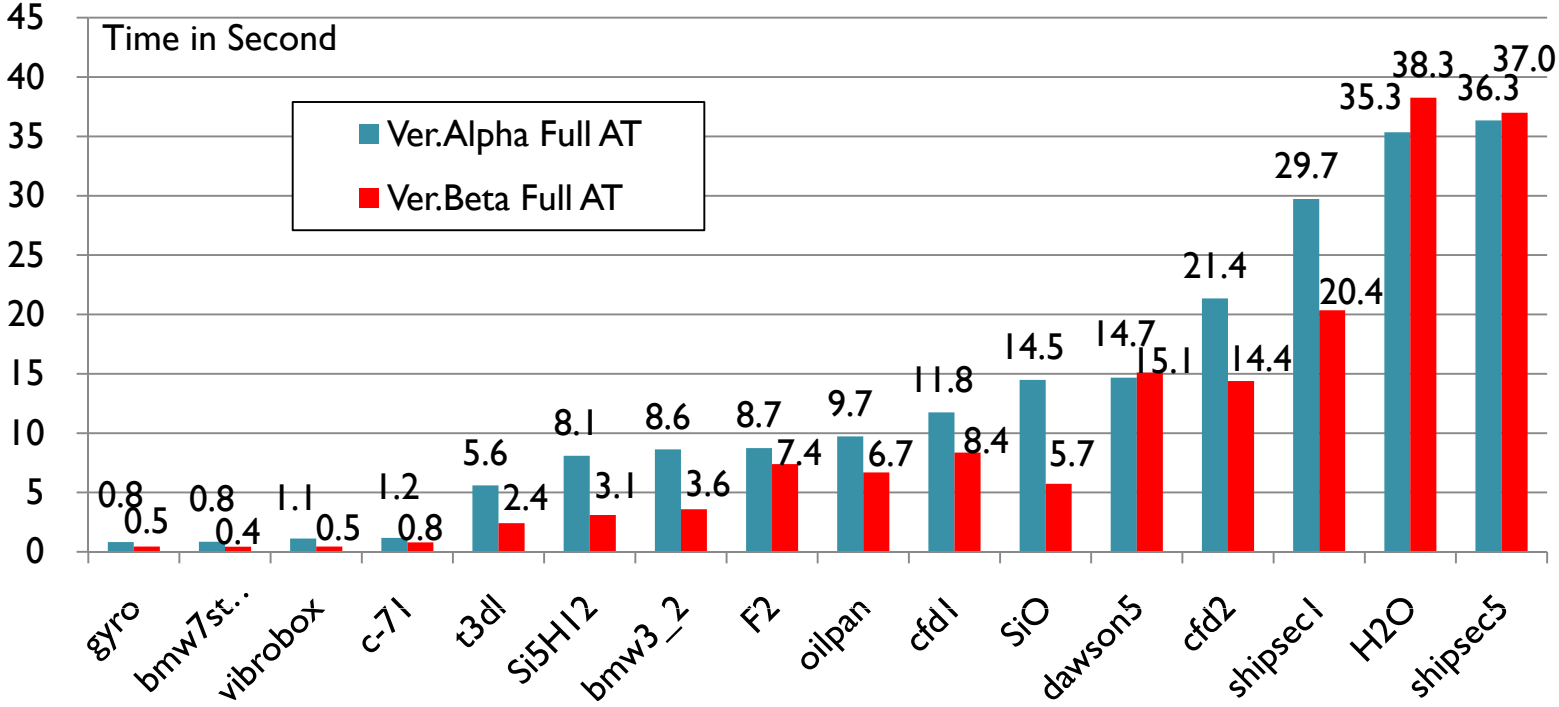
OpenATI_DURMV(β版)の性能

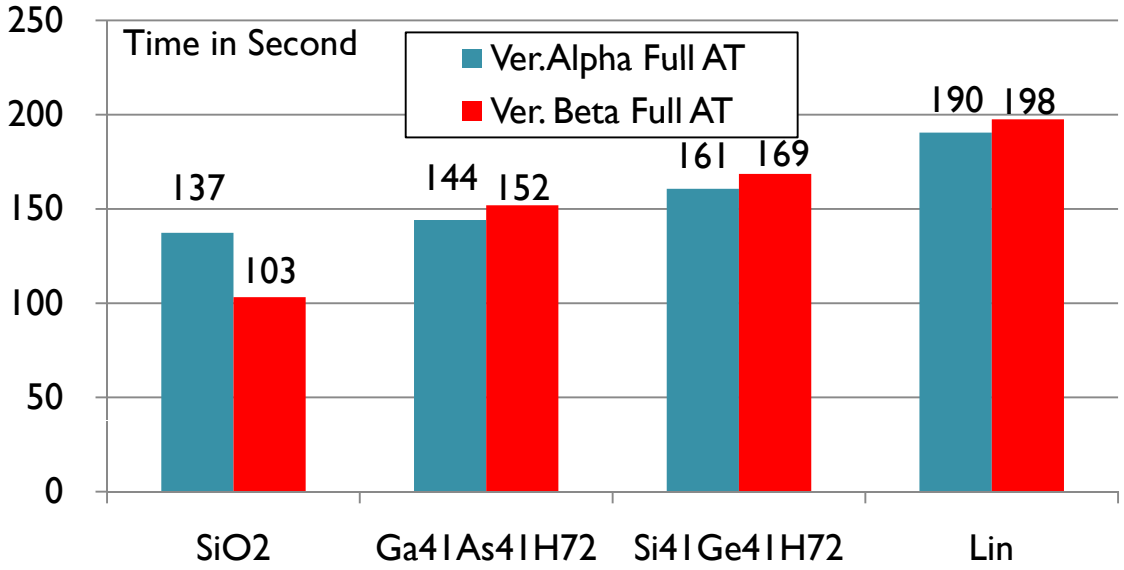
- #Cores = 16
- JL = 128 (Constant)

BSSで約5.6倍の速度向上



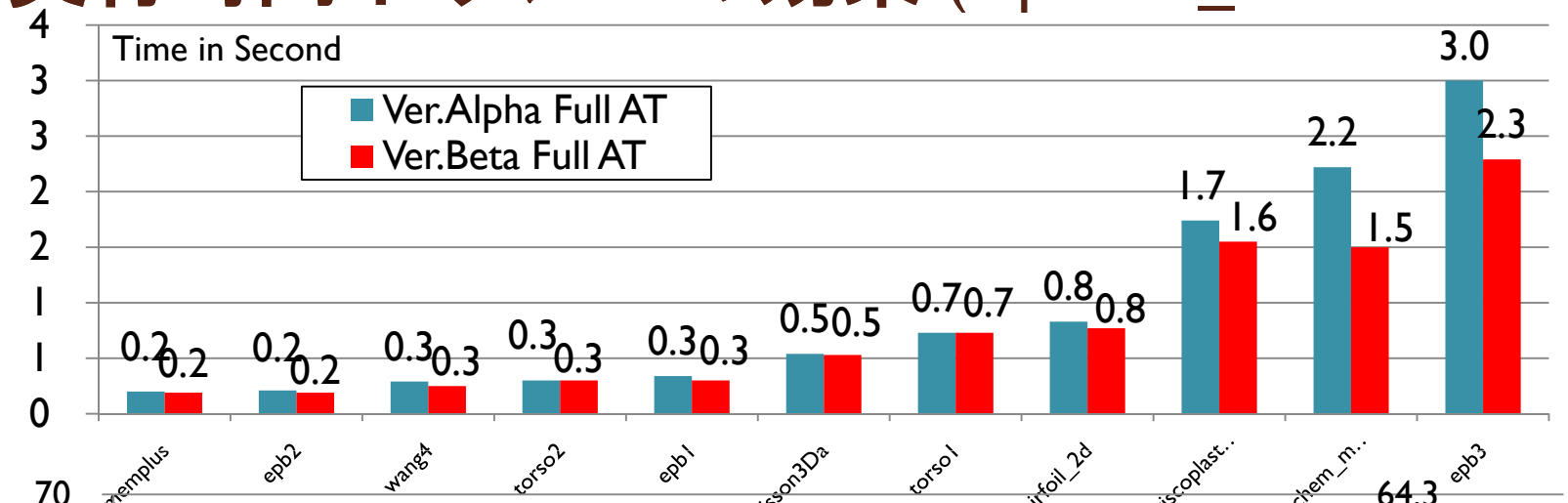
実行時間ポリシーの効果(OpenATI_EIGENSOLVE)

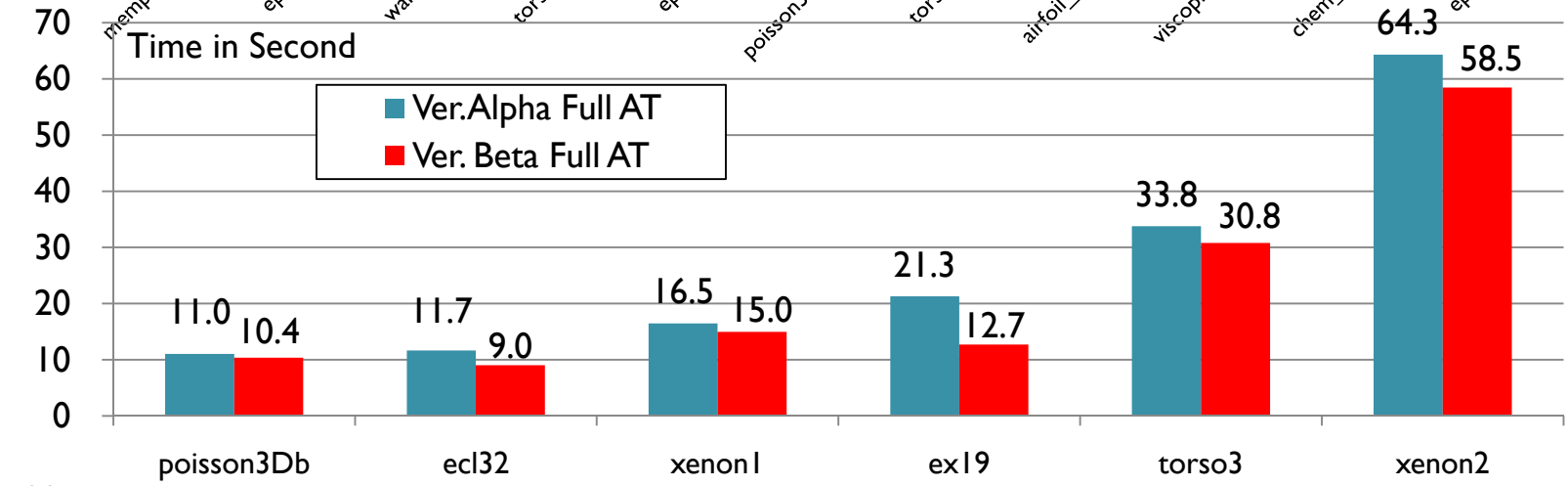


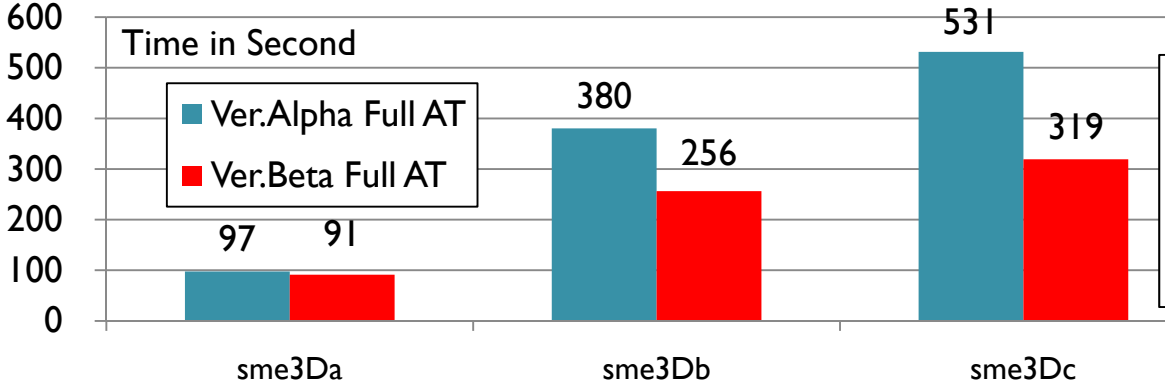


平均速度向上:
1.59x
最大速度向上:
2.61x (Si5H12)

実行時間ポリシーの効果 (OpenATI_LINEARSOLVE)





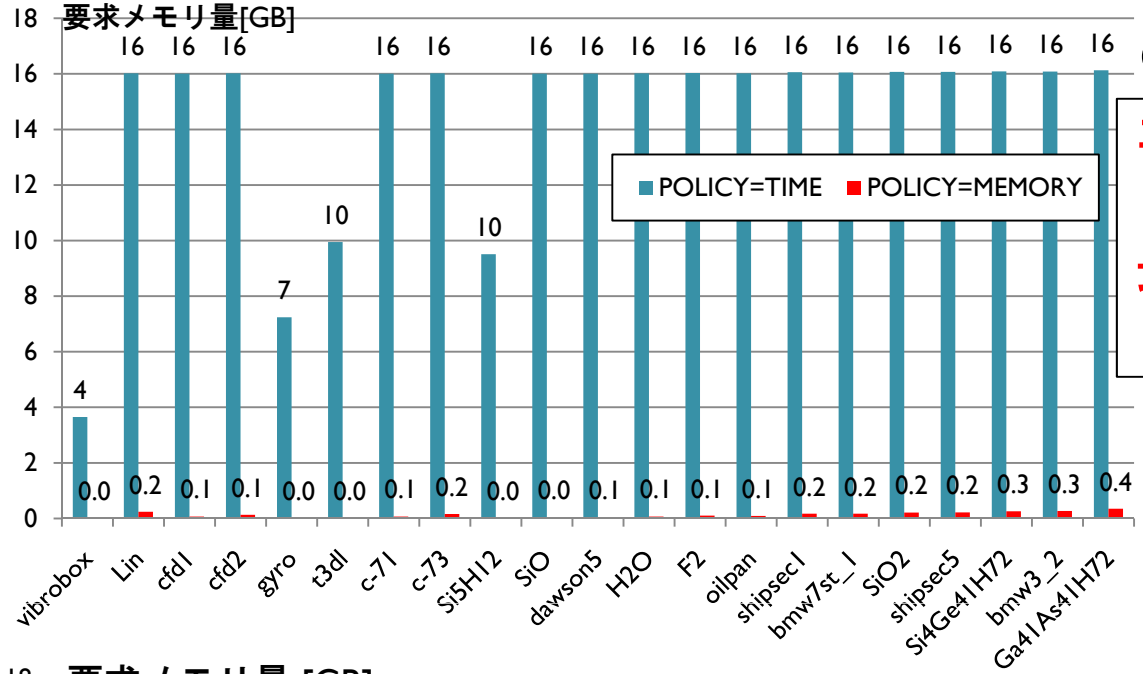


平均速度向上:
1.20x
最大速度向上：
1.66x (sme3Dc)

メモリポリシー最適化条件

- 最大リスタート周期
  - POLICY=MEMORY : 100 (固定値)
  - POLICY=TIME のとき:
    最大リスタート周期を確保可能な最大メモリ量を調査して設定.
- 直交化方式をBCGSに固定
- SpMxVの選択
  - 対称行列用
    - 全てのSpMxV実装を検査
  - 非対称行列用
    - BSS抜きの２種のSpMxVを検査

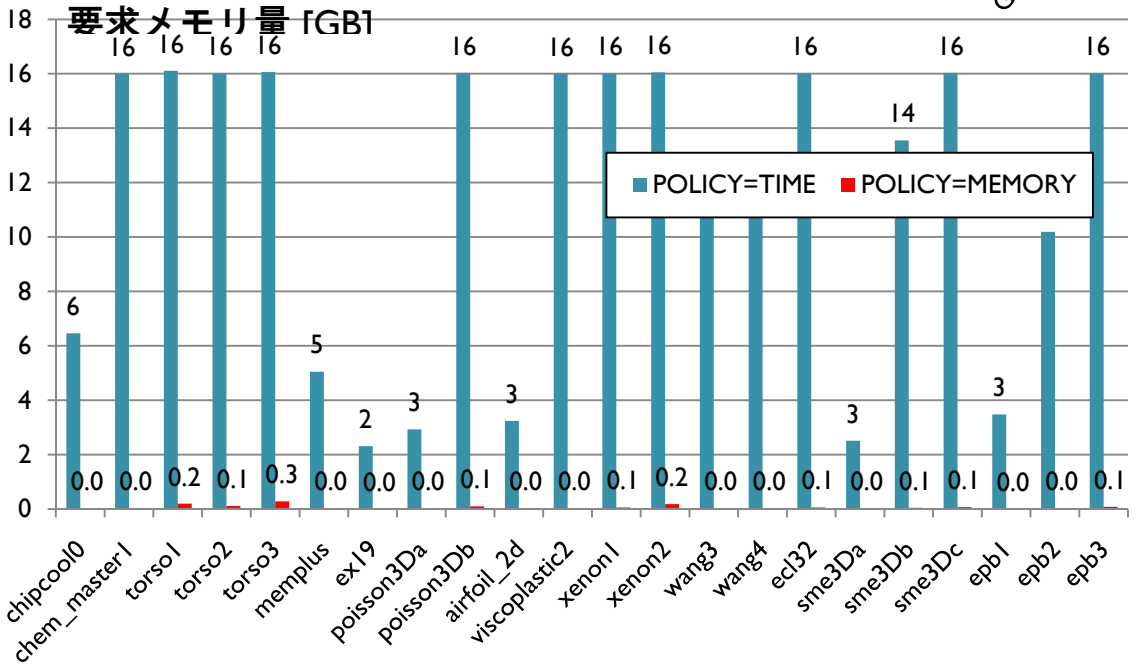
メモリポリシーの効果



OpenATI_EIGENSOLVE

平均速度低下:
1.24x
最悪速度低下:
6.12x (Ga41As41H72)

最大で200倍の削減効果

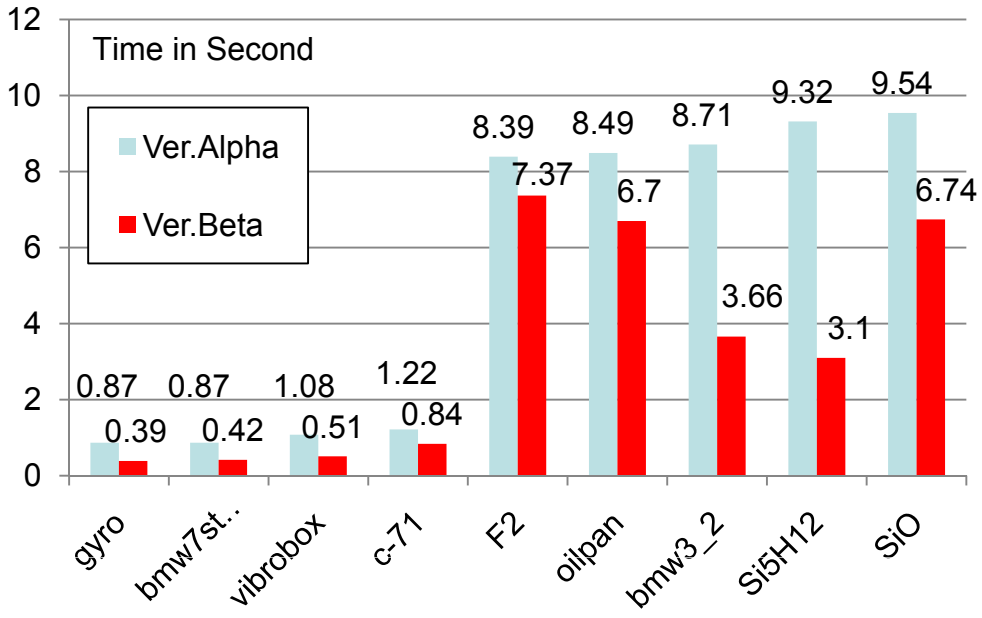


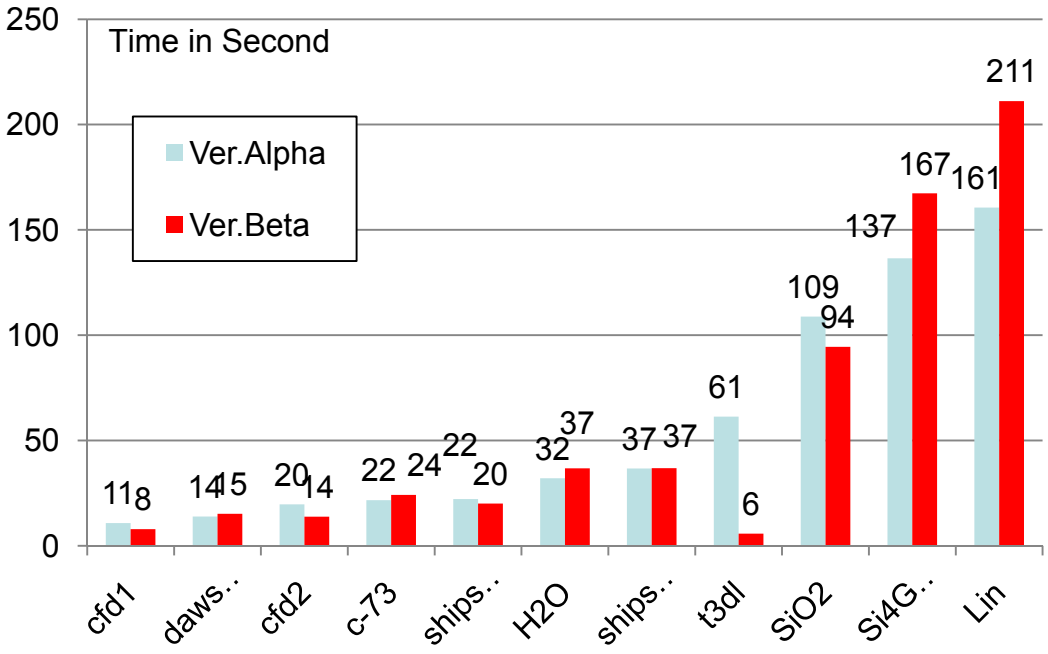
OpenATI_LINEARSOLVE

平均速度低下:
1.09x
最悪速度低下:
2.04x (ex19)

精度ポリシの効果(OpenATI_EIGENSOLVE)

ユーザからの要求精度：
1.0e-8

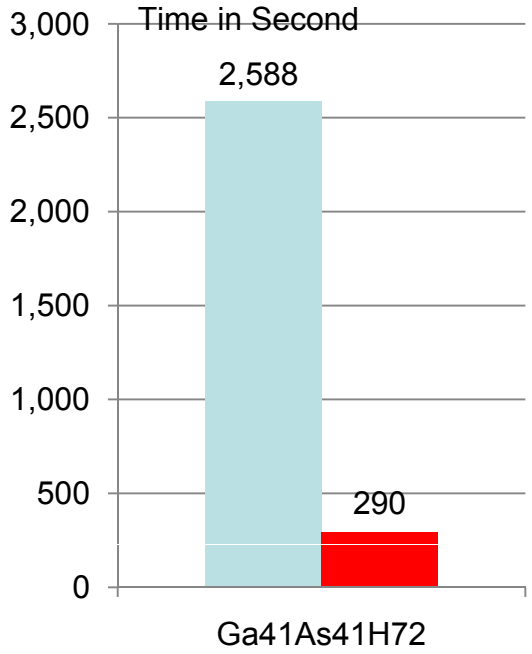




α版において、以下の行列は要求精度を満たさず

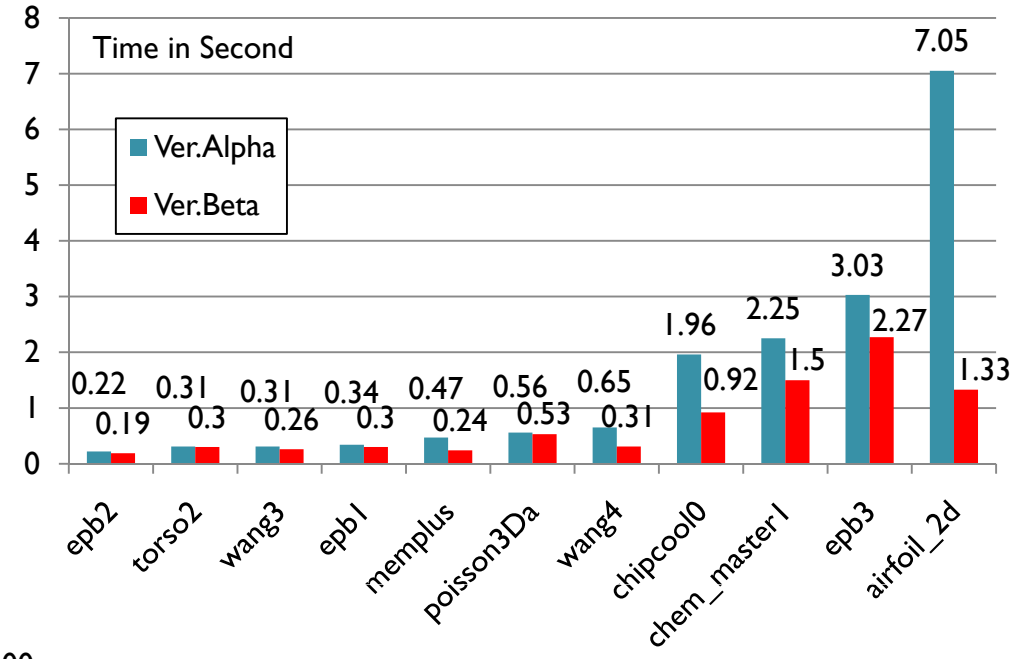
- T3dl :
  - 相対残差：2.24E-01
  - 直交性: 1.27E-02
- Ga41As41H72
  - 相対残差: 2.52E-08
  - 直交性: 4.81E-09

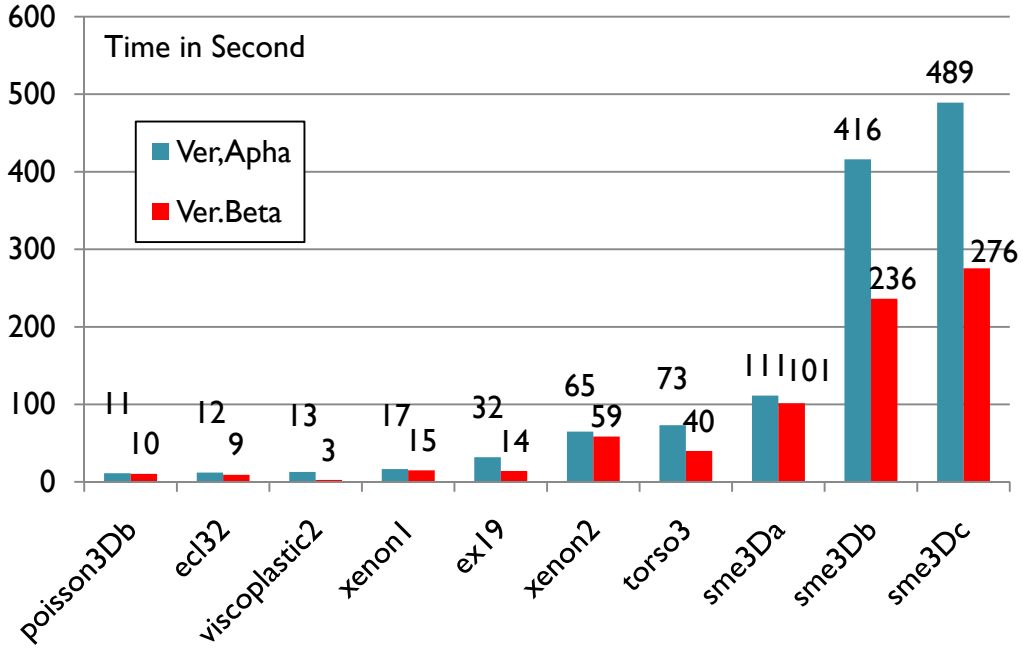
平均速度向上: 2.24倍
最大速度向上: 8.92倍
(Ga41As41H72)



精度ポリシの効果(OpenATI_LINEARSOLVE)

ユーザからの要求精度：1.0e-8





torso1 は双方で収束せず。
α版は以下の行列で相対残差の要求精度を満たせず
- torso3： 2.28E-08
- memplus： 1.92E-08
- airfoil_2d： 5.17E-05
- viscoplastic2： 3.67E-05
- wang4： 4.35E-08

平均速度向上
1.69倍
最大速度向上:
5.29倍 (airfoil_2d)

速度向上は、
β版で提供される高精度再直交化ルーチン（DGKS 型G-S）による収束性の向上が理由。

おわりに

- 自動チューニング研究
  - 自動チューニング時間の削減が問題
  - 多くの場合「経験則（職人芸）」を利用
    - 多くは、「全探索」で実現
  - 試行回数や標本点の削減がカギ
  - このあたりは、「数理」研究者の参入を期待
- OpenATLibとXabclib
  - 実行時に全探索して最適な候補を決める
    - （主に）反復開始前に一度行う
  - 自動チューニングポリシ設定機能
    - メモリ量最適化：平均1.2倍の速度低下で平均200倍のメモリ量削減が可能
    - 演算精度最適化：旧バージョン（α版）に対し，
      - 平均1.6倍～2.4倍の速度向上
      - 実誤差を満たさない場合（偽収束）を改善
      - 要求精度改善工程の自動化