自動チューニング処理記述用ディレクティブ ABCLibScript

片桐孝洋[†,††] 吉瀬謙二[†,††]
本多弘樹[†] 弓場敏嗣[†]

この論文[a]では，FIBER フレームワークによる自動チューニング処理記述用ディレクティブである ABCLibScript の設計方針と実装について述べる．ABCLibScript では処理対象を数値計算処理に限定し，ブロック化アルゴリズムのブロック幅調整，アンローリング段数調整，およびアルゴリズム選択，といった自動チューニング機能のディレクティブを提供する．このことで自動チューニング処理の付加を実際に行うライブラリ開発者において，自動チューニングライブラリ作成の際の労力の削減とその開発支援をねらう．

[a] この論文は，2004 年 SACSIS (Symposium on Advanced Computing Systems and Infrastructures) シンポジウムに 2004 年 1 月 21 日に投稿した論文「自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装」をもとに，加筆・修正を行ったものである．また，電気通信大学大学院情報システム学研究科の IS テクニカルレポート UEC-IS-2004-1 として，2004 年 1 月 27 日に登録した論文である．

ABCLibScript: A Directive to Support Specification of Auto-tuning

Takahiro KATAGIRI,[†,††] Kenji KISE,[†,††] Hiroki HONDA[†] and Toshitsugu YUBA [†]

In this paper, we describe a design and an implementation of ABCLibScript, which is a directive to support the addition of auto-tuning processes based on the framework of FIBER. ABCLibScript limits the function of auto-tuning to numerical computations. For example, the functions of the block-length adjustment for blocked algorithms, loop unrolling depth adjustment, and algorithm selection, are crucial functions in ABCLibScript. We focus on the reduction and support of development for auto-tuning library to innovate the limited functions of ABCLibScript.

1. はじめに

近年，PHiPAC[1)]，ATLAS[2)]，FFTW[3)]，および ILIB[4)] に代表される，いわゆる「自動チューニング機能付き」の数値計算ソフトウエア（以降，自動チューニング機能付きソフトウエア，Software with Auto-Tuning Facilities (SATF) とよぶ）が多数開発されるようになってきた．現在 SATF の性能評価が多くなされており，自動チューニング機能の有効性が周知のものとなりつつある[4)]．

ところが性能の観点で SATF は有効であるのにもかかわらず，ライブラリ開発者が SATF を自分の開発ライブラリに付加する場合，現状では個別に SATF の機能を開発して付加するしかない．例えば数値計算ライブラリの場合，一般的に行列の直接解法ルーチン，反復解法ルーチン，密行列用解法ルーチン，および疎行列用解法ルーチンが含まれているが，それらのルーチン全てに容易に SATF を付加できる基盤ソフトウエアが存在しない．

そこで本論文では機能を数値計算処理に限定し，自動チューニング機構の付加が容易なディレクティブである ABCLibScript[5),6)] を設計，実装する．

本論文の構成は以下の通りである．2 章で，ABCLibScript で前提とする自動チューニング方式の FIBER について説明する．次に 3 章で，ABCLibScript の設計方針について述べる．4 章は，API (Application Programing Interface) を中心とする ABCLibScript の実装について説明する．5 章では，ABCLibScript を用いたプログラム例を示す．また 6 章では，行列積コードに ABCLibScript を用いて自動チューニング機構を付加したコードの動作検証，およびそのコードに

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 科学技術振機構 戦略的創造研究推進事業 さきがけプログラム「情報基盤と利用環境」領域
“Information Infrastructure and Applications”, PRESTO, Japan Science and Technology Agency (JST)

おける自動チューニング機構の性能評価を行う．7 章では関連研究を述べ，さいごにこの論文での知見をまとめる．

2. FIBER による自動チューニングの概要

FIBERは，SATF に幅広く適用できるソフトウエア枠組の確立を目指し提案された自動チューニングの枠組である[7)~9)]．具体的には SATF の構成方式を，(1) ライブラリインストール時に行う**インストール時最適化階層** (Install-time Optimization Layer, IOL)，ユーザが指定した特定パラメタ (たとえば問題サイズなど) が固定した地点で行う**実行起動前最適化階層** (Before Execute-time Optimization Layer, BEOL)，そして実際にライブラリが実行された地点で行う**実行時最適化階層** (Run-time Optimization Layer, ROL)，の 3 種の最適化階層に分けて構成する枠組である．この SATF のためのソフトウエア構成を，FIBER (Framework of Install-time, Before Execute-time, and Run-time optimization layers，ファイバー) と呼ぶ．

2.1 2 種のユーザと FIBER のソフトウエア構成

FIBER では，数値計算ライブラリにおける自動チューニング処理を主なターゲットとしている．そのため FIBER におけるユーザは，以下の 2 種が想定されている．

- ライブラリ開発者
- ライブラリ利用者 (エンドユーザ)

ライブラリ開発者は，FIBERが提供する自動チューニング指定子である ABCLibScript を利用して，自分が作成したライブラリに自動チューニング機能を付加するユーザである．一方で，ライブラリ利用者 (エンドユーザ) はライブラリ開発者が開発した FIBER による自動チューニング機構を付加したライブラリを利用するユーザである．

図 1 に，上記 2 種のユーザの観点での FIBER における処理手順をのせる．

図 1 では，2 種のユーザごとに処理手順が異なる．まずライブラリ開発者は，FIBER が提供するディレクティブ (ABCLibScript) を用いて自動チューニングを施したい箇所 (チューニング対象領域とよぶ) の指定と，どのような自動チューニング機能を適用するかを指定する．その後，FIBER が提供するプリプロセッサを起動する．プリプロセッサ起動後，以下の 3 種の構成要素がもとのプログラムに自動付加される．

- **パラメタ最適化コンポーネント：**ライブラリ開発者が指定したパラメタを最適化する部分． 最適化の方式は多種考えられるが，現在の実装では全探索 (brute-force search)，ユーザが指定する関数 (コスト定義関数) による探索空間の絞り込み，および最適パラメタ推定法指定 (最小二乗法による) の機能を有する．

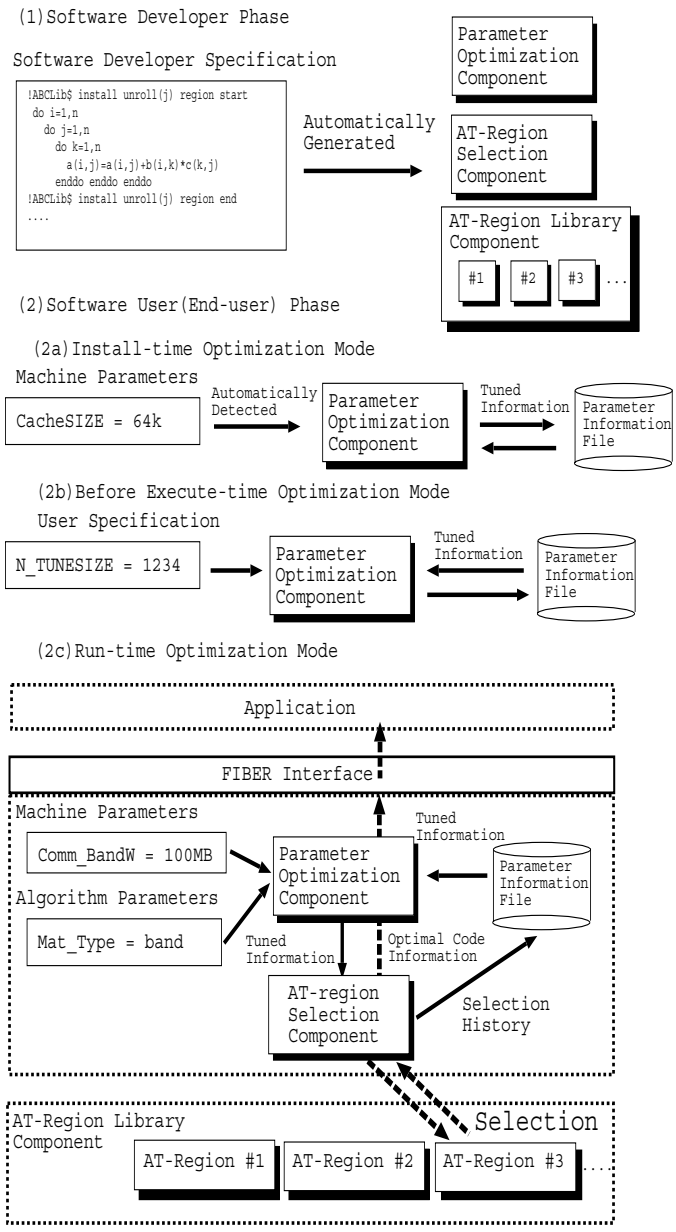

図 1 FIBER での処理手順

- AT **領域選択コンポーネント：**現在の最適化の前に行われた最適化済データを蓄積しているファイル (パラメタ情報ファイル) 上の情報と実行時情報から，AT 領域ライブラリコンポーネント内にある最適な自動チューニング対象ルーチンを選択する部分．
- AT **領域ライブラリコンポーネント：**ライブラリ開発者が指定したチューニング対象領域を，FIBER 内で識別可能なサブルーチンとして蓄積する部分．

3. ABCLibScript の設計

3.1 設計目標

ABCLibScript は，以下の 4 点に考慮して設計がなされている．

1. 数値計算処理に特化した自動チューニング機能を用意することによるチューニング指定の容易さ
2. ライブラリ開発者が注釈の形式でプログラム中に指示を与えることで，もとのプログラムの実行を阻害しないチューニング処理指定方式
3. ライブラリ開発者の注釈を解釈し可読性の高いプログラムを自動生成するプリプロセッサの提供
4. 性能に影響する 2 種のパラメタである性能パラメタ（Performance Parameter，PP）と基本パラメタ（Basic Parameter，BP）の概念の導入によるチューニング (パラメタ最適化) 処理の単純化

ABCLibScript は，数値計算処理を指向した以下の機能を有する．

- ブロック化アルゴリズムに対応するブロック幅調整機能
- ループアンローリングに対応するループアンローリング段数調整機能
- ライブラリ利用者が与える知識により，最適なアルゴリズムを切替えるアルゴリズム選択機能

ライブラリ開発者による注釈を解釈し，ライブラリ開発者が解釈できるプログラムを自動生成するプリプロセッサを提供する．このことで，自動チューニング処理の付加後のプログラムさえもライブラリ開発者自身がとり扱える．またソースコードが生成され，かつライブラリ開発者がそれを所有することで，自動付加された自動チューニング処理もライブラリ開発者に透過性がある (処理のホワイト・ボックス化)．

ABCLibScript では，FIBER の特徴である性能に影響する 2 種のパラメタ (PP と BP) の概念を導入することにより自動チューニングの定義をしている．すなわち自動チューニングとは，チューニングの対象領域におけるコストの挙動を定義した関数（コスト定義関数 F）について，パラメタ BP を固定した上で，PP を 3 種のタイミング（インストール時，実行起動前，および実行時）においてコスト定義関数 F を最小化する処理と定義する．(文献 7)〜9) を参照．) 本論文で提案する ABCLibScript は，ライブラリ開発者に PP と BP の定義をさせた上で，最適化処理である自動チューニング処理を記述できるディレクティブといえる．

3.2 指定形式

3.2.1 ライブラリ開発者による性能パラメタ指定

FIBER では概述のとおり，ライブラリ開発者がソースプログラム中に性能パラメタ PP，基本パラメタ BP☆，およびチューニングの範囲 (チューニング対象領域) の指定をする．

ソースプログラムにおいて，`!ABCLib$`で始まる行は ABCLibScript による自動チューニングのための指示行とみなす．この表記法の概略を**図 2** に示す．

```
!ABCLib$ ⟨ 自動チューニング種類 ⟩ ⟨ 機能名 ⟩
        [(対象変数)]  region start
[!ABCLib$ ⟨ 機能の詳細 ⟩ [sub region start]]
    チューニング対象領域
[!ABCLib$ ⟨ 機能の詳細 ⟩ [sub region end]]
!ABCLib$ ⟨ 自動チューニング種類 ⟩ ⟨ 機能名 ⟩
        [(対象変数)]  region end
```

図 2 ABCLibScript における自動チューニング指示形式

図 2 における ⟨ 自動チューニングの種類 ⟩ や ⟨ 機能名 ⟩ のことを指定子とよぶ．また ⟨ 機能の詳細 ⟩ のことを補助指定子とよぶ．

3.2.2 指定子および補助指定子の説明

2004 年 1 月における ABCLibScript の仕様である指定子一覧を**図 3** に，および補助指定子一覧を**図 4** にのせる．

- ⟨ 自動チューニング種類 ⟩ :: =
  （`install` | `static` | `dynamic` | ⟨ 式 ⟩）
  `install`：インストール時チューニングの指定
  `static`：実行起動前チューニングの指定
  `dynamic`：実行時チューニングの指定
  ⟨ 式 ⟩ :: =処理対象の計算機言語の文法に従う式
- ⟨ 機能名 ⟩ :: =
  （`define` | `variable` | `select` | `unroll`）
  `define`：パラメタ設定指定
  `variable`：変動するパラメタ指定
  `select`：複数の手続きからの選択指定
  `unroll`：ループアンローリング指定

図 3 ABCLibScript における指定子

3.3 処理対象の計算機言語

ABCLibScript の仕様の本質は，ライブラリ開発者が所有するライブラリが記述されている計算機言語に依存しない．しかし ABCLibScript を処理するプリプロセッサの実装を考慮すると，対象となる言語を特定する必要がある．

ここでは並列数値計算ライブラリを指向し，ライブラリ記述言語は Fortran90，および通信ライブラリとして MPI(Massage Passing Interface)-1 が利用されていることを前提とする．この ABCLibScript の実装を考慮した仕様を，$ABCLibScript(Fortran90, MPI-1)$ と表記する．

☆ 基本パラメタ BP については，行列サイズ n がデフォルトのパラメタとなっている．ただし ABCLibScript の仕様として，ライブラリ開発者により基本パラメタを追加する API の `ABCLib_BPset`(後述) を用意している．

- `name` ⟨ 文字列 ⟩ : チューニング領域の名称を記述.
  （全ての機能で利用可能）
- `parameter` （ ⟨ 属性指定 ⟩ ⟨ 変数名 ⟩, [⟨ 属性指定 ⟩ ⟨ 変数名 ⟩, …]）
  : パラメタ情報ファイルに出力か，パラメタ情報ファイルから入力するパラメタを指定する.
  もしくは，基本パラメタであることを宣言する.
  ⟨ 属性指定 ⟩ :: = [`in` | `out` | `bp`]
  `in` ： 入力される（外部で定義され参照される）パラメタ,
  `out` ：出力する（このチューニング領域で定義される）パラメタ,
  `bp` : 基本パラメタであることの宣言.
  (全ての機能で利用可能)
- `select sub region` (`start` | `end`) : 選択する手続きであることを指定.
  (機能名 select 指定時)
- `according` （ ⟨ 条件式 ⟩ | `estimated` ） : 以降に指定する基準で，手続きを選択することを指定.
  ⟨ 条件式 ⟩ :: =[（`min` ⟨ 変数名 ⟩ | `condition` ⟨ 条件 ⟩）⟨ 接続演算子 ⟩]
  ⟨ 接続演算子 ⟩ :: = [`.and.` | `.or.`] ⟨ 条件式 ⟩
  `estimated` ⟨ 数式 ⟩: ユーザが定義した選択に関するコスト（数式）を基に，最適な処理を選択する処理であることを指定.
  (機能名 select 指定時)
- `varied from X to Y` : 指定するパラメタの変動範囲（X から Y まで）の指定.
  （機能名 variable, unroll 指定時）
- `fitting` ⟨ 方式 ⟩ `sampled` ⟨ 範囲 ⟩ ：パラメタの推定に利用する方式を指定.
  ⟨ 方式 ⟩ :: =[`least-square` ⟨ 次数 ⟩ | `user-defined` ⟨ 数式 ⟩ | `auto`]
  `least-square` ：線形多項式による最小二乗法でパラメタ推定することを指定.
  ⟨ 次数 ⟩ で，線形多項式の次数を設定.
  `user-defined` ：ユーザ指定による数式を用いて最小二乗法で推定.
  `auto` : システム側にパラメタの推定を任せる.
  ⟨ 範囲 ⟩ :: = [⟨ 数値 ⟩ | `auto`] : パラメタ推定に必要な範囲を指定.
  なお ⟨ 方式 ⟩ =`auto` 指定時は省略可能.
  ⟨ 数値 ⟩ : 具体的なパラメタの数値を指定.
  `auto` ： パラメタのサンプリング間隔を自動決定．fitting 補助指定子省略時，varied 補助指定子で指定した範囲全てを測定（＝全探索）して最適パラメタを決める.
  （機能名 variable, unroll 指定時）
- `pripro sub region` (`start` | `end`) : チューニング領域を呼び出す前に適用する処理を指定.
  (全ての機能で利用可能)
- `postpro sub region` (`start` | `end`) : チューニング領域を呼び出した後に適用する処理を指定.
  (全ての機能で利用可能)
- `debug` (⟨ 変数 ⟩, [⟨ 変数 ⟩, …]) : チューニング領域が実行されるときに，デバック表示する変数を指定.
  （全ての機能で利用可能）
  ⟨ 変数 ⟩ :: = [`bp` | `pp` | **任意変数**]
  `bp` ： 基本パラメタ情報を表示する,
  `pp` ： 性能パラメタ情報を表示する,
  任意変数 ： 記述された変数情報を表示する.

図 4 ABCLibScript における補助指定子

3.4 プリプロセッサ利用方法

FIBER フレームワークでは，ABCLibScript で記述した注釈を解釈し，自動チューニング機能を付加したコードを自動生成するプリプロセッサを提供する．このプリプロセッサ名は ABCLibCodeGen である．

ABCLibScript(*Fortran*90, *MPI* − 1) の仕様に基づき，並列プログラム (test.f) に対し，実際の自動チューニング機能付き並列数値計算プログラムのコード（Fortran90 + MPI-1）を生成するには，以下のコマンドを実行する．

```
>ABCLibCodeGen test.f
```

またプリプロセッサABCLibCodeGen は，以下の実行時オプションを指定できる．

【実行時オプション】-debug
OFF：デバック用コードを生成しない（デフォルト値）
ON：ABCLib_DEBUG 変数で指定されるデバックレベル x のデバック用コードを生成する

【実行時オプション】-visualization
OFF：自動チューニング軌跡ファイルを出力しない（デフォルト値）
ON：自動チューニング軌跡ファイルを出力し，そのファイルを利用してビジュアライゼーションする

【使用例 1】

```
> ABCLibCodeGen -debug ON
  -visualization ON test.f
```

ABCLibScript による指示子が記入されているプログラム test.f に対して，デバック用コードを生成し，かつ自動チューニングモードでの軌跡を自動チューニング軌跡ファイルに残すことを指定する．この自動チューニング軌跡ファイルを利用して，開発予定のビジュアライザを用いてチューニング経過を閲覧できる．

4. ABCLibScript の実装

4.1 ABCLibScript の API

ABCLibScript では，自動チューニングを実行するためのユーザインタフェース (API) を用意しており，それを利用することでユーザ (ライブラリ開発者および利用者) が自動チューニングの詳細な処理を記述する．具体的には，自動チューニングをするために ABCLibScript が提供する，以下の API を利用する．

- ABCLib_ATexec
  (ABCLib_ATkinds, ABCLib_ATroutines)

ABCLib_ATexec 手続きは，ABCLib_ATkind で指定した自動チューニングを，ABCLib_ATroutines で指定したチューニング領域について実行する．引数ABCLib_ATkinds は，自動チューニングの種類を指定するための引数である．ヘッダファイル ABCLibScript.h で定義された以下の 3 種の定数が指定できる．ABCLib_INSTALL：インストール時自動チューニング；ABCLib_STATIC：実行起動前自動チューニング；ABCLib_DYNAMIC：実行時自動チューニング；ABCLib_ALL：すべての自動チューニング；

引数ABCLib_ATroutines は，どのチューニング領域の処理についてか指定する引数である．この引数はヘッダファイルABCLibScript.h で定義されている型ABCLib_ATname を用いてユーザが宣言するか，ABCLibScript.h でcommon 定義されている大域変数，ABCLib_AllRoutines：すべてのルーチン用；ABCLib_InstllRoutines：インストール時自動チューニングルーチン用;ABCLib_StaticRoutines:実行起動前自動チューニングルーチン用;ABCLib_DynamicRoutines:実行時自動チューニングルーチン用；を利用して指定する．

【使用例 2】

```
!ABCLib$ call ABCLib_ATexec
   (ABCLib_INSTALL, ABCLib_InstallRoutines)
```

：インストール時最適化について，インストール時最適化を指定した全てのチューニング領域について行う．

- ABCLib_ATset
  (ABCLib_ATkinds,ABCLib_ATroutines)

ABCLib_ATset 手続きは，ABCLib_ATroutines で指定したチューニング領域名を，ABCLib_ATkind で指定する自動チューニングの種類として設定する．

- ABCLib_ATdel
  (ABCLib_ATroutines,DelName)

ABCLib_ATdel 手続きは，ABCLib_ATroutines で指定したチューニング領域名情報が入っている変数から，DelName で指定するチューニング領域名を削除する．引数DelName に削除したいチューニング領域名を記述する．なおチューニング領域名は，注釈の形式で各チューニング領域に記述しておく．

【使用例 3】

```
!ABCLib$ call ABCLib_ATdel
  (ABCLib_InstallRoutines,"MyMatMul")
```

：インストール時最適化を行う候補から，”MyMatMul” と名付けたチューニング領域を除外する．

- ABCLib_ATInstallInit
  (ABCLib_InstallRoutines)

ABCLib_ATInstallInit 手続きは，ABCLib_InstallRoutines で指定されるインストール時自動チューニング領域名のチューニングを未実行とする．

- ABCLib_BPset (BPvalName)

ABCLib_BPset 手続きは，引数BPvalName に指定したパラメタ名を，新たな基本パラメタ BP とする．

【使用例 4】

```
!ABCLib$ call ABCLib_BPset( "nprocs" )
```

：変数nprocs を基本パラメタ（BP）とする．

- ABCLib_BPsetName(Kind,BPvalName,Name)

ABCLib_BPsetName 手続きは，引数Kind で指定される自動チューニングを行う際の基本パラメタ BP の

固定に関する変数について，引数BPvalName に指定した基本パラメタ名に関する名称を引数Name とする．ここで引数Kind は，以下のとおりである．Kind ::= [STARTTUNESIZE | ENDTUNESIZE | SAMPDIST]. また，これら変数の意味は以下の通りである．STATTTUNESIZE: 基本パラメタBPvalName に関するサンプリング開始点情報；ENDTUNESIZE ： 基本パラメタBPvalName に関するサンプリング終了点情報；SAMPDIST：基本パラメタBPvalName に関するサンプリング間隔情報．

【使用例 5】

```
!ABCLib$ call ABCLib_BPsetName("STARTTUNESIZE",
   "nprocs","ABCLib_NprocsStartSize")
```

：基本パラメタnprocs における自動チューニング開始点を指定する変数をABCLib_NprocsStartSize とする．

- ABCLib_BPsetCDF(BPvalName,CDFKind)

ABCLib_BPsetCDF 手続きは，引数BPvalName に指定した基本パラメタに関するサンプリング点以外のパラメタ推定方式を，引数CDFKind で指定される種類のコスト定義関数とする．ここで，引数CDFKind は，以下のとおりである．CDFKind ::= [least-square ⟨ 次数 ⟩ | user-defined ⟨ 式 ⟩ | auto]. また，これら変数の意味は以下の通りである：least-square ⟨ 次数 ⟩: 線形多項式による最小二乗法でパラメタ推定することを指定する．⟨ 次数 ⟩ で線形多項式の次数を指定する；user-defined ⟨ 式 ⟩: ユーザ定義による式を用いて最小二乗法でパラメタ推定する；auto：システム側にパラメタ推定の方式決定を任せる；

なおデフォルトでは，チューニング領域で定義されたコスト定義関数と同一の方式を用いて推定する．さらにユーザがチューニング領域でのコスト定義関数を省略した場合は，3 次の線形多項式を用いた最小二乗法による推定方式が選択される．

【使用例 6】

```
!ABCLib$ call ABCLib_BPsetCFD("nprocs",
   "least-square 5")
```

：基本パラメタnprocs に関するパラメタ推定方式を，5 次の線形多項式を用いた最小二乗法で行うことを指定する．

4.2 API 記述例

例えば，ライブラリ開発者が所有するサブルーチン EigenSolver の利用に関して，ライブラリ開発者が記述する自動チューニング処理手順を記述したサブルーチンfoo は，ABCLibScript の API を用いて以下のように記述する．

```
subroutine foo(...)
include (ABCLibScript.h)
  …
C ===チューニング領域の登録など初期化
!ABCLib$ call ABCLib_ATset
!ABCLib$&  (ABCLib_ALL,ABCLib_AllRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_INSTALL,
!ABCLib$&  ABCLib_InstallRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_STATIC,
!ABCLib$&  ABCLib_StaticRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_DYNAMIC,
!ABCLib$&  ABCLib_DynamicRoutines)
C ===インストール時自動チューニングの実行
C  (以下はライブラリ開発者が記述)
C  !全体を通して 1 回のみしか実行できない
!ABCLib$ call ABCLib_ATexec(ABCLib_INSTALL,
!ABCLib$&  ABCLib_InstallRoutines)
C        ===インストール時最適化のみ実行済
!ABCLib$ call EigenSolver(…)
  …
C ===実行時自動チューニングの実行許可
C  (以下はライブラリ開発者が記述)
C  !この地点では実行されない．
C   対象の以下の EigenSolver コール後の
C     指定箇所実行時に行われる
!ABCLib$ call ABCLib_ATexec (ABCLib_DYNAMIC,
!ABCLib$&  ABCLib_DynamicRutines)
C        ===呼び出し後，対象箇所でチューニング
C  （インストール時，実行起動前 (利用者が行う)，
C     および実行時最適化が実行済）
!ABCLib$ call EigenSolver(…)
…
return
end
```

ライブラリ開発者が提供したライブラリの機能である EigenSolver の利用に関して，ライブラリ利用者が記述する実行起動前最適化処理を実行するサブルーチンpooh は，ABCLibScript の API を用いて以下のように記述する．

```
subroutine pooh(...)
include (ABCLibScript.h)
  …
C ===実行起動前自動チューニングの実行
C  (以下はライブラリ利用者が記述)
C  !この地点で実行される
C       ===ライブラリ開発者定義の BP の固定
        N_TUNESIZE_START=1234
        N_TUNESIZE_END=1234
        call ABCLib_ATexec(ABCLib_STATIC,
  &            ABCLib_StaticRoutines)
C    ===インストール時，実行起動前最適化実行済
        call EigenSolver(…)
  …
return
end
```

このように実行起動前自動チューニングの記述は，ライブラリ利用者にとって負担が大きい．この負担を

削減するため，専用 GUI の開発が必要である．

5. ABCLibScript によるプログラム例

5.1 インストール時最適化

以下のプログラム例 1 は，インストール時最適化において行列積のアンローリング段数調整処理を記述した例である．

【プログラム例 1】行列積

```
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 16
!ABCLib$ fitting least-square 5
!ABCLib$&  sampled (1-5, 8, 16)
!ABCLib$ 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
```

プログラム例 1 では，デフォルトである基本パラメタ BP を問題サイズに関する変数 n とする．また性能パラメタ PP は，対象となる行列積の最外側ループ変数 i に関するアンローリング段数である．

アンローリングは 1 段から 16 段まで行うことを定義している．最適化の対象となるコスト定義関数は，5 次の線形多項式と定義しており，BP を固定した場合での PP のサンプリング点（＝アンローリング段数）は，1 段から 5 段，8 段，および 16 段である．また PP を推定するための方法は最小二乗法である．

5.2 実行起動前最適化

以下のプログラム例 2 は，アルゴリズム選択処理におけるプログラミング例である．アルゴリズム選択基準として，ライブラリ開発者が定義するコスト定義関数を利用する．

【プログラム例 2】ユーザ定義によるコスト定義関数によるアルゴリズム選択

```
!ABCLib$ static select region start
!ABCLib$ name TestSelect
!ABCLib$ parameter (in CacheS,in NB,in NPrc)
!ABCLib$   select sub region start
!ABCLib$   according estimated
!ABCLib$&  (2.0d0*CacheS*NB)/(3.0d0*NPrc)
           チューニング対象領域 1
!ABCLib$   select sub region end
!ABCLib$   select sub region start
!ABCLib$   according estimated
!ABCLib$&  (4.0d0*CacheS*dlog(NB))
!ABCLib$&   /(2.0d0*NPrc)
           チューニング対象領域 2
!ABCLib$   select sub region end
!ABCLib$ static select region end
```

プログラム例 2 では実行起動前最適化において，アルゴリズムの選択処理をすることを指定している．性能パラメタ PP として，チューニング対象領域 1 もしくは 2 の実行指定がパラメタ化される．

ユーザが定義するコスト定義関数において，この最適化が行われる前に定義されている浮動小数点変数（CacheS，NB，NPrc）が利用される．チューニング対象領域 1 のコストは $(2.0d0 * CacheS * NB)/(3.0d0 * NPrc)$ で見積もられ，チューニング対象領域 2 のコストは $(4.0d0 * CacheS * dlog(NB))/(2.0d0 * NPrc)$ で見積もられる．これらコスト評価の結果，コストが小さい処理に実行指定がなされる．

5.3 実行時最適化

以下のプログラム例 3 では，ライブラリ開発者が定義しアルゴリズム中で定義参照される変数（eps，$iter$）を用いて，最適なアルゴリズムを選択する処理を指定した例である．具体的には eps が最小となるような処理を，条件 $iter < 5$ 以内で決定する．このプログラム例 3 は，反復解法における前処理方式の自動選択を指向している．

【プログラム例 3】反復解法における前処理方式の実行時選択

```
!ABCLib$ dynamic select (eps,iter)
!ABCLib$&  region start
!ABCLib$ name PricondSelect
!ABCLib$ parameter (in eps, in iter)
!ABCLib$ according min (eps) .and.
!ABCLib$&   condition (iter<5)
!ABCLib$   select sub region start
       チューニング対象領域 1（前処理 1）
             ...
             eps = ...
!ABCLib$   select sub region end
!ABCLib$   select sub region start
       チューニング対象領域 2（前処理 2）
             ...
             eps = ...
!ABCLib$   select sub region end
!ABCLib$ dynamic select (eps,iter)
!ABCLib$&  region end
```

プログラム例 3 では実行時において，浮動小数点変数 eps と $iter$ に基づきアルゴリズムの選択処理をすることを指定している．性能パラメタ PP として，チューニング対象領域 1 もしくは 2 の実行指定がパラ

メタ化される．

ライブラリ開発者が定義するコスト定義関数は，この最適化が呼ばれる前に定義されている変数 *eps* と *iter* で定義される．このコスト定義関数とは，*iter* が 5 より小さい場合について各チューニング対象領域で *eps* の値を保存しておき，*iter* の値が 5 以上になったとき，*eps* の値が最も小さいチューニング領域の処理を検索して選択する関数である．

6. 動作検証と性能評価

本章では，行列積のコードに ABCLibScript を用いて FIBER による自動チューニング機構を自動付加したコードの動作検証と，そのコードにおける FIBER 自動チューニング機構の性能評価を行う．

6.1 テスト環境

本検証評価における，計算機環境は以下の通りである．

- CPU: Pentium4 2.4GHz
- Memory: 256MByte
- OS: Linux RedHat 8
- コンパイラ：PGI 社製コンパイラ 4.0-2
- コンパイラオプション：-O0
- MPI: MPICH 1.2.1

また，検証評価対象は以下の通りである．

- 自動チューニング対象：インストール時
- ABCLibScript の unroll 指定子によるアンローリング段数指定：1–64 段
- 対象：3 重ループの行列積コードにおける，最外側ループ (*i*-ループ)，第 2 ループ (*j*-ループ)，および最内側ループ (*k*-ループ)
- 行列の次元： 500

6.2 テストコード

以下に，この検証評価で用いたテストコードを示す．

【テストコード 1】行列積コード

```
      program  main
      include  'ABCLibScript.h'
      integer  iauto
      integer  N
      parameter (N=500)
      real*8   A(N,N), B(N,N), C(N,N)
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     ================================
!ABCLib$ call ABCLib_ATset(ABCLib_ALL,
!ABCLib$&   ABCLib_AllRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_INSTALL,
!ABCLib$&   ABCLib_InstallRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_STATIC,
!ABCLib$&   ABCLib_StaticRoutines)
!ABCLib$ call ABCLib_ATset(ABCLib_DYNAMIC,
!ABCLib$&   ABCLib_DynamicRoutines)
      iauto = 1
      in = 350
      if (iauto .eq. 1) then
!ABCLib$ ABCLib_NUMPROCS = 4
!ABCLib$ ABCLib_STARTTUNESIZE = 100
!ABCLib$ ABCLib_ENDTUNESIZE = 500
!ABCLib$ ABCLib_SAMPDIST = 100
!ABCLib$ call ABCLib_ATexec(ABCLib_INSTALL,
!ABCLib$&   ABCLib_InstallRoutines)
      else
!ABCLib$ ABCLib_DEBUG = 1
        call MatMul(A, B, C, in)
      endif
c     ===== MPI finazize
c     ================================
      call MPI_FINALIZE(ierr)
c     ================================
      stop
      end

      subroutine MatMul(A, B, C, N)
      integer N
      real*8  A(N,N), B(N,N), C(N,N)
      include  'ABCLibScript.h'
      real*8  da1, da2
      real*8  dc
      do i=1, N
        do j=1, N
          A(i,j) = 0.0d0
        enddo
      enddo
      do i=1, N
        do j=1, N
          B(j,i) = dble(i*j)
          C(j,i) = 1.0d0/dble(i*j)
        enddo
      enddo
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 64
!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
    return
    end
```

6.3 結果と考察

プリプロセッサABCLibCodeGen により，テストコード 1 のプログラムが処理され，FIBER による自動チューニング機構が付加されたコード (Fortran90+MPI) が自動生成された．

図 5 に，この自動生成されたコードにおいて，FIBER のインストール時自動チューニングを指定して実行した場合の自動チューニング結果 (自動チューニング軌跡) をのせる． 図 5 のインストール時自動

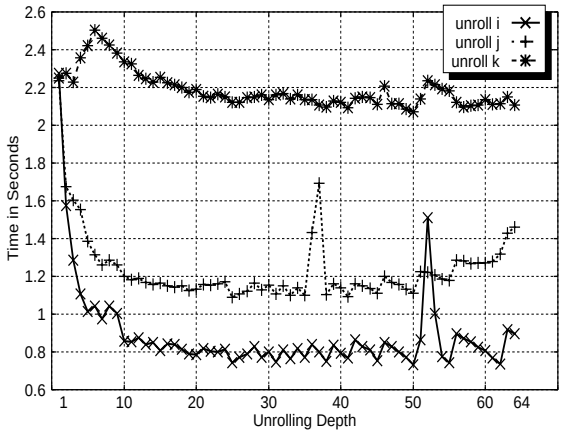

図 5 テストコード 1(行列積) における自動チューニング結果

チューニング結果から，以下の情報を得た．

- i-ループ：最適段数 50 段，速度向上比 3.1 倍
- j-ループ：最適段数 25 段，速度向上比 2.1 倍
- k-ループ：最適段数 50 段，速度向上比 1.1 倍

なお速度向上比は，アンローリング 1 段の実行時間を最適段数での実行時間で除算した値である．

このようにディレクティブ ABCLibScript により，FIBER の自動チューニング機構が付加されたプログラムが自動生成され，かつそれが正しく動作することを確認した．またテストコード 1 に対して，FIBER のインストール時自動チューニング機能により最大で 3.1 倍のチューニング効果が得られることを確認した．

なおテストコード 1 における i, j, k-ループのうち，どのループに対してアンローリングを行うと最も高速となるかを自動選択する処理を実装する場合，select 指定子が必要となる．このselect 指定子に関する実装，機能検証，および性能評価は今後の課題である．

7. 関連研究

パラメタ自動チューニングに関する枠組やソフトウエアの研究は，以下の 2 種に分類できる．

まず計算機システムにおける実行時最適化の枠組をもつソフトウエアがある．たとえば，パラメタ (I/O バッファサイズなど) の決定を実行時に行うソフトウエアとして Active Harmony[10]，および Autopilot[11] がある．

次に数値計算ライブラリにおけるインストール時最適化の枠組をもつソフトウエアがある．例えば，PHiPAC[1]，ATLAS[2] および経験的手法を用いた枠組の AEOS[12], および FFTW[3] が知られている．

自動チューニングの定式化では，SIMPL[13] においてインストール時最適化に関する定式化を行った．この SIMPL において，性能パラメタ (PP) と基本パラメタ (BP) の概念がはじめて提案された．一方，今村と直野[14] は，対象となるコードのレジスタ数などを算出し，固有値ソルバ中の特定処理に関するアンローリング処理の最適段数を決定する式を導出した．その結果パラメタ空間の探索に関し，大幅なコストの削減に成功した．今村と直野により導出された式は，固有値計算におけるアンローリング処理の振舞いを実行時間において定義したコスト定義関数と解釈でき，ABCLibScript での記述や ABCLibScript で用いる新しい最適化方式として実装が期待できる．

また Brewer[15] はソースコードの構成情報や実行時間計測から，対象となるプログラムの実行時間に関する挙動を自動的にモデル化する方式を提案し，これをライブラリのアルゴリズム選択に利用した．Brewer の自動モデル化方式は，ABCLibScript でのユーザ定義によるコスト定義関数指定の自動化を行う方式であると解釈できる．

しかしながらこれらの研究のいずれも，本論文で示した ABCLibScript のように，自動チューニング処理自体の容易な記述を目指した言語処理系開発に関連するものではない．

8. おわりに

この論文では，FIBER による自動チューニング処理記述用ディレクティブである ABCLibScript の設計方針と実装について述べた．ABCLibScript は，処理対象を数値計算に限定した自動チューニング機能を提供することで，自動チューニングの付加を行うライブラリ開発者において，自動チューニングライブラリ作成の際の労力の削減とその開発支援をねらう．

現在，FIBER フレームワークに基づく自動チューニング機能付き並列数値計算ライブラリ *ABCLib* を公開している．この *ABCLib* は，科学技術振興機構さきがけプログラム「情報基盤と利用環境」領域の研

究成果の一部として開発された．また ABCLibScript の機能の一部 (アンローリング処理等) を実装した試用版を開発済である．

ABCLib のソースコード，および ABCLibScirpt 試用版は，科学技術振興機構の助成による WWW サーバ (`http://www.abc-lib.org/`) 上でマニュアル等を含め公開される予定である．

謝辞 本研究は科学技術振興機構さきがけプログラムの助成による．

参 考 文 献

1) Bilmes, J., Asanović, K., Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, *Proceedings of International Conference on Supercomputing 97*, pp. 340–347 (1997).
2) ATLAS project; http://www.netlib.org/atlas/index.html.
3) Frigo, M.: A Fast Fourier Transform Compiler, *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, Georgia, pp. 169–180 (1999).
4) 片桐孝洋, 黒田久泰, 大澤清, 工藤誠, 金田康正: 自動チューニング機構が並列数値計算ライブラリに及ぼす効果, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol.42, No.SIG 12 (HPS 4), pp. 60–76 (2001).
5) 片桐孝洋: 計算装置、計算方法、プログラムおよび記録媒体, 日本国特許出願, Vol. 特願２００３－０９２５９２ (平成１５年3月２８日).
6) 片桐孝洋: 計算装置、計算方法、プログラムおよび記録媒体, 日本国特許出願, Vol. 特願２００３－１４９７０１ (平成１５年5月２７日). 特願２００３－９２５９２の国内優先権出願.
7) Takahiro, K., Kise, K., Honda, H. and Yuba, T.: FIBER: A Framework of Installation, Before Execution-invocation, and Run-time Optimization Layers for Auto-tuning Software, *IS Technical Report, Graduate School of Information Systems, The University of Electro-Communications*, Vol. UEC-IS-2003-3 (May 2003).
8) Takahiro, K., Kise, K., Honda, H. and Yuba, T.: FIBER: A General Framework for Auto-Tuning Software, *Proceedings of The Fifth International Symposium on High Performance Computing*, Vol. Springer Lecture Notes in Computer Science, No. 2858, pp. 146–159 (2003).
9) 片桐孝洋, 吉瀬謙二, 本多弘樹, 弓場敏嗣: FIBER: 汎用的な自動チューニング機能の付加を支援するソフトウエア構成方式, 情報処理学会研究報告, Vol. 2003-HPC-94, pp. 1–6 (2003).
10) Tapus, C., Chung, I.-H. and Hollingsworth, J. K.: Active Harmony : Towards Automated Performance Tuning, *Proceedings of High Performance Networking and Computing (SC2002)*, Baltimore, USA (2003).
11) Ribler, R. L., Simitci, H. and Reed, D. A.: The Autopilot Performance-Directed Adaptive Control System, *Future Generation Computer Systems, special issue (Performance Data Mining)*, Vol. 18, No. 1, pp. 175–187 (2001).
12) Whaley, R., Petitet, A. and Dongarra, J. J.: Automated Empirical Optimizations of Software and the ATLAS project, *Parallel Computing*, Vol. 27, pp. 3–35 (2001).
13) 直野健, 山本有作: 単一メモリ型インターフェイスを有する自動チューニング並列ライブラリの構成方法, 情報処理学会研究報告, No. 2001-HPC-87, pp. 25–30 (2001).
14) 今村俊幸, 直野健: 性能安定化を目指した自動チューニング型固有値ソルバーについて, SACSIS2003 論文集, pp. 145–152 (2003).
15) Brewer, E. A.: Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization, Technical report, Ph.D Thesis, Massachusetts Institute of Technology (1994).