1J-05

ソフトウェア自動チューニングツール DSICE の開発環境の統合化

森戸建太郎 矢島雄河 藤家空太郎 楊暄 藤井昭宏 田中輝雄

工学院大学

1. はじめに

ソフトウェア自動チューニング[1]は，実行性能に影響する複数の性能パラメタの最適な組み合わせを推定し，性能向上を図る技術である．我々は手法のひとつとして，反復一次元 d-Spline 探索を提案[2]し，d-Spline の二次元化への拡張の検討や推定の効率化を行ってきた．

反復一次元 d-Spline 探索を適用するチューニングプログラムである d-Spline Iterative Collinear Exploration(DSICE)は，本来 C を対象としていた．機械学習のハイパーパラメタに対し自動チューニングを適用するため，Python 版 DSICE を作成した[3]．しかし，新たに作成した Python 版 DSICE を今後も運用することは，開発環境が C/Python の 2 言語となるため，二重管理が発生する．

本研究では，C と Python で作成された DSICE を 1 本化し，2 言語で実行可能なソフトウェア自動チューニングプログラム DSICE の開発環境の統合化を実現する．

2. ラッピング処理を用いた拡張

2.1 Python からの C の呼び出し

DSICE の統合化を図る上で，今後の開発環境をいままで開発していた C とした．よって，Python から C の関数呼び出しをできるようにする必要がある．

ラッピング処理とは，他言語の関数に対し，呼び出し元の変数型を変換することで，関数を使用可能にする手法である．このラッピング処理が記述された API の役割を果たすプログラムをラッパー関数と呼ぶ．また，ラッパー関数と DSICE を共有ライブラリ化する．Python 側で呼び出すため，開発環境の統合化を実現する．共有ライブラリへのアクセスと変数型の変換を図 1 に示す．

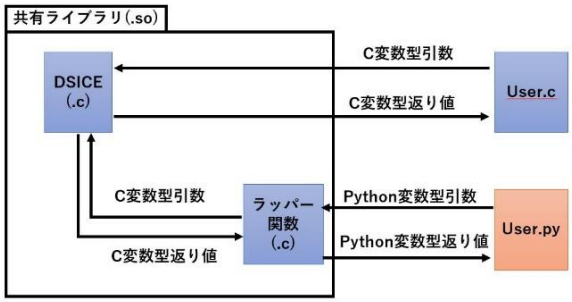

図 1 共有ライブラリへのアクセスと変数型の変換

2.2 Python/C API

Python/C API は Python インタプリタに対しアクセス手段を C に提供する．主に Python インタプリタを拡張するモジュールの記述に使用される．本研究では，ラッパー関数の作成にこの Python/C API を利用した．理由としてラッパー関数内部に事前に作成した関数を利用することが容易となるためである．

int x，int y を引数に返り値を int 型で持つ関数 func(x,y) に対応するラッパー関数の作成例を図 2 に示す．ラッピング対象の関数に対し C の変数型に変換した後，関数を実行，返り値を Python で受け取り可能な型へ変換する．

```
1  extern int func(x,y);
2  PyObject *wrap_func(PyObject *self,PyObject *args){
3      int x,y,result;
4      if (!PyArg_ParseTuple(args, "ii", &x, &y)){
5          return NULL;}
6      result = func(x,y);
7      return Py_BuildValue("i", result);
8  }
```

・PyArgParseTuple(): Cの変数型に変換
・PyBuildvalue(): Python で受け取り可能な型へ変換

図 2 ラッピング処理の例

3. DSICE にチューニング構造

DSICE はチューニング対象関数の前後に以下の関数群を組み込み，それらを反復させることで自動チューニングを行う．また，利用時の事前準備として，(1)チューニング対象の関数化，(2)性能パラメタの設定，(3)指標データの設定を行う必要がある．チューニング構造の全体像と関数群の組み込み箇所について図 3 に示す．

・DSICE_INIT(): チューニングに必要な基本情報の登録
・DSICE_BEGIN(): 性能パラメタの組み合わせ変更
・DSICE_SET_DOUBLE(): 性能指標データの取得
・DSICE_END(): チューニング領域の終了地点を指定

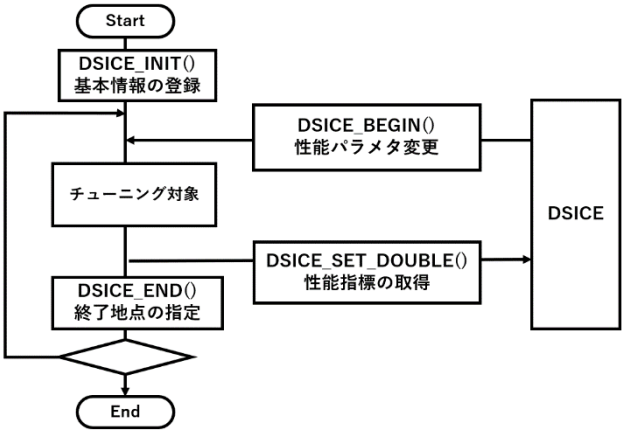

図 3 チューニングの位置づけと関数対応

Integrated development environment for automatic software tuning tool DSICE
Kentaro Morito, Yuga Yajima, Sorataro Fujika, Xuan Yang, Akihiro Fujii, and Teruo Tanaka
Kogakuin University

```c
void sampl_use_DSICE_C(){
    int pp_dimsize = 2 , pp_Nums = {11, 11};
    int *trial_idx;
    double metric;
    int pp1[] = {-15, -14, ..., -6, -5};
    int pp2[] = {-3, -2.2, ..., 2.6, 3};
    int count = 0;
    DSICE_INIT(pp_Nums, pp_dimsize);
    while (count < 50){
        trial_idx = DSICE_BEGIN();
        metric = Bukin_func_N6(
            pp1[trial_idx[0]],
            pp2[trial_idx[1]]);
        DSICE_SET_DOUBLE(metric);
        DSICE_END();
        count += 1;}
}
```

図 4 C での記述例

```python
import DSICE , ctypes # "DSICE" のインポート
def sampl_use_DSICE_py():
  pp_dimsize = 2
  pp_pat = ctypes.c_int * 2 # 変更点 ctypes.int
  pp_Nums = pp_pat(11, 11)
  pp1 = (-15, -14, ..., -6, -5)
  pp2 = (-3, -2.2, ..., 2.6, 3)
  count = 0
  DSICE.DSICE_INIT(pp_Nums, pp_dimsize) # 基本情報登録
  while count < 50:
    trial_idx = DSICE.DSICE_BEGIN() # 性能パラメタの変更
    metric = Bukin_func_N6( # 関数化したチューニング対象
      pp1[trial_idx[0]],
      pp2[trial_idx[1]])
    DSICE.DSICE_SET_DOUBLE(metric) # 性能指標の取得
    DSICE.DSICE_END() # チューニング領域の終了地点の指定
    count += 1
```

図 5 Python での記述例

4. 実験

4.1 DSICE の組み込みの記述

図 4 と図 5 に DSICE を組み込んだ C/Python のプログラムを示す．チューニング対象関数の前後に，前述の関数を記述し，そのうち 3 つを反復させている．

プログラムの差違を Python の記述例である図 5 を用いて説明する．Python での利用では，DSICE と ctypes の 2 つのモジュールをインポートする[1 行目]必要がある．

DSICE は，自動チューニングを行うための関数群の呼び出しに使用する．このとき，ラッパー関数にアクセスし変数型の変換を行い関数を実行する．

ctypes は，C と互換性のあるデータ型を Python に提供する外部関数ライブラリである．これは，性能パラメタの設定の一部[4 行目]を ctypes.c_int 型にするときに利用する．変更する理由としてラッパー関数内部に ctypes 型の変数を利用したため，ユーザ側で変更が必要となるためである．

4.2 検証実験

Bukin N.6 関数[4]をモデル関数とし，Python で DSICE を適用する．図 6 に関数形状，最適解，性能パラメタの取りうる値を示す．最適解は(-10,1)である．

反復一次元 d-Spline 探索の経路について図 7 に示す．探索は，x 軸，y 軸，45°，135°の 4 軸となる．横軸に性能パラメタ pp1 の要素番号を，縦軸に pp2 の要素番号を取り探索経路が数字で確認できる散布図を示す．開始地点(5,5)の周辺を探索をした後，方向を決定し一次元探索を行った．このとき，最適解である(5,7)を探索したため以降この点を中心に探索する．続けて開始地点を(5,7)とし周辺を探索をした．その後，方向を x 軸方向，45°，135°の方向に決定し一次元探索をし，最適解で収束した．探索経路から，拡張したライブラリの探索機構が C/Python で想定の動作している事が確認できる．

5. おわりに

本研究では，反復一次元 d-Spline 探索によるチューニングプログラム DSICE に対しラッピング処理を施すことで，2 言語(C/Python)で実行可能なライブラリの開発環境を C で統合化した．

今後の課題として，新しいアプリケーションへの言語対応が挙げられる．本研究では Python の対応を行った．今後，適用先のアプリケーションが広がるに伴い，対応すべき言語も増えると考える．

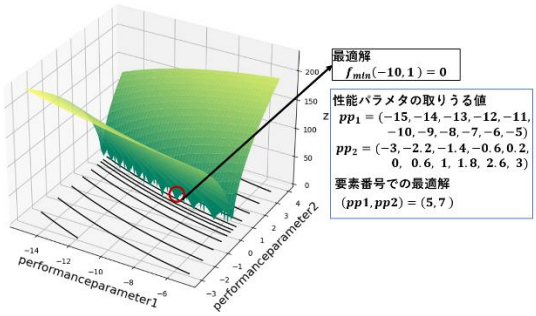

図 6 Bukin N.6 関数形状，最適解，性能パラメタ

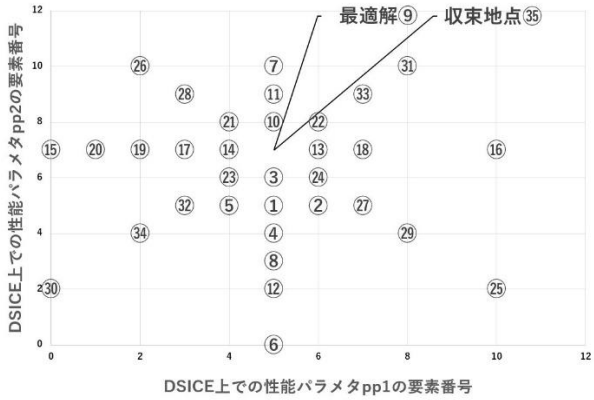

図 7 収束するまでの探索経路

謝辞

本研究の一部は JHPCN(jh210019-NAH)及び JSPS 科研費 JP18K11340 の助成により実施した．

参考文献

[1] 今村，荻田，尾崎，片桐，須田，高橋，滝沢，中島：ソフトウェア自動チューニング:科学技術計算のためのコード最適化技術，森北出版(2021).
[2] 望月，藤井，田中：ソフトウェア自動チューニングにおける複数同時性能パラメタ探索手法の提案と評価，情報処理学会論文誌 ACS，Vol.11，No.2，pp.1-16(2018).
[3] 藤家，多部田，藤井，田中，加藤，大島，片桐：GPU クラスタを用いて並列化した自動チューニングの機械学習プログラムへの適用と安定性の検証，情報処理学会 HPC 研究会，2019-HPC-178，No.16，pp.1-8(2021).
[4] Virtual Library of Simulation Experiments:Test Functions and Datasets ,available from 〈https://www.sfu.ca/~ssurjano/bukin6.html〉(参照 2021-12-22).