Vol.2019-HPC-169 No.9
2019/5/10

性能パラメータ推定における評価対象プログラムの実行時間の揺らぎに対応した自動チューニング手法の提案

関直人[†1]　范谷瑛[†1]　多部田敏樹[†1]　藤井昭宏[†1]　田中輝雄[†1]

概要：自動チューニングは，ユーザプログラム中に存在する実行時間を司る性能パラメータの推定を行う技術である．チューニングの処理はユーザプログラムの実行には無関係であるため，その計算量や使用メモリ量が軽微なことが要求される．自動チューニングの主目的は，ユーザプログラムの総実行コストを削減することである．我々は効率的な性能パラメータ探索手法として，少数の性能パラメータの組み合わせから推定を始めて，推定値が安定するまで性能パラメータの組み合わせを選択し追加する探索方法を提案している．探索は，選択した性能パラメータでユーザプログラムを実行した際の実行時間を利用して行う．自動チューニングにおいて，実行時間の揺らぎに頑健な探索方法を提案することが研究の方針である．実行時間の揺らぎとは，同じ性能パラメータにおける毎回の実行時間の変化を表し，探索を困難にする要因のひとつである．本論文では，実行時間の揺らぎを考慮する機能を追加することで，自動チューニングにおける総実行コストを削減することを目的とする．そのために，同じ性能パラメータにおいて複数回の実行を行い，実行時間の揺らぎの計測を行う．その際，実行時間が長い性能パラメータで複数回の計測を行うことを避けるため，探索がある程度行われ推定が安定してから揺らぎの計測を開始するようにした．提案手法による評価の結果，これまでの揺らぎを組み込んでいない方式よりも総実行コストを削減することを可能にした．

キーワード：ソフトウェア自動チューニング，実行時自動チューニング，実行時間揺らぎ

1. はじめに

ユーザが自動的にソフトウェアの性能チューニングを行う手法として，ソフトウェア自動チューニングがある[1][2]．ソフトウェア自動チューニングでは，プログラム中の実行時間，消費電力等に影響を与える処理をパラメータ化し，そのパラメータの最適化を行なう．このパラメータを性能パラメータと呼ぶ．性能パラメータとしては，たとえば，アンローリング段数，キャッシュブロッキングサイズ等がある．ソフトウェア自動チューニングのひとつの形態として，実行時自動チューニングがある[3]．これは，実行時に疎行列の非零パターンが決定されるような問題において行われる自動チューニングである．実行時自動チューニングでは毎ステップ実行する性能パラメータを選択し，探索を行うことで，効率的に性能パラメータを推定する．近似関数を含むチューニング処理は，ユーザのターゲットプログラムの実行に影響を及ぼす．よって，推定に用いる近似関数は計算コストが小さく，実測データに柔軟に追従することが要求されており，それらの条件を満たす近似関数として，d-Spline を提案している[4].

我々は，ソフトウェア自動チューニングの研究として，一次元探索を反復して行うことにより複数性能パラメータを探索する反復一次元探索[5]，実時間のブレを考慮した探索[6]について取り組んできた．

反復一次元探索は，少ない探索コストで優れた性能パラメータを探索することができる．一次元の d-Spline を利用し，IPPE(Incremental Performance Parameter Estimation) により一次元探索を行っている[7]．IPPE は，まず初めに少数の標本点を実測し，得られた情報を利用して近似関数の生成を行い，その後，推定値が安定するまで次に追加する標本点を選択し近似関数を逐次的に更新していく探索方法である．

一方，実時間のブレを考慮した探索については，文献[6]では，OS 割り込み等の突発的に発生する異常値による近似関数への影響を抑える手法について論じている．さらに，実環境においては，突発的な異常値に加え，同じ性能パラメータにおける毎回のプログラム実行時にも実行時間の揺らぎが見られ，これは適切な推定を行うことを困難にしている要因のひとつである．本研究では，計算機で数値計算プログラムを実行する際に毎回発生する実行時間の揺らぎに頑健な手法を検討し，自動チューニングにおける総実行コストを削減することを目的とする．総実行コストを削減するため，ある程度推定が進んだことを判定してから揺らぎの計測を開始するようにした．また，推定後も常に最適な性能パラメータを求め，実行する機能も追加した．

2. 実環境における実行時間の揺らぎ

2.1 実行時間の揺らぎの調査

ここでは，推定を困難にしている要因のひとつである実行時間の揺らぎについて，実環境でどれくらいの大きさとなるのか調査を行う．本論文では，最先端共同 HPC 基盤施設（JCAHPC）により運用されている Oakforest-PACS を用いて評価を行う[8]. 実測環境として, 表 1 に Oakforest-PACS のノード構成を，表 2 に対象問題とコンパイラ，並列化方法を示す．実測結果として，図 1 と図 2 に 500 回連続して

†1 工学院大学
Kogakuin University

表 1　Oakforest-PACS のノード構成

プロセッサ名	Intel Xeon Phi 7250 (codename:KnightsLanding)
CPU 数	1
コア数	68
メモリ	96 GB(DDR4) and 16 GB(MCDRAM)
ベース動作周波数	1.40 GHz
ターボ・ブースト周波数	1.60 GHz

表 2　対象問題とコンパイラ，並列化方法

対象問題	行列行列積(DGEMM)
行列サイズ	3000×3000
コンパイラ	mpiicc
コンパイルオプション	-axMIC-AVX512 –qopenmp –O3
並列化	MPI,OpenMP
プロセス数および スレッド数	各ノード 1 プロセス 256 スレッド

DGEMM を計算させた際の実行時間の推移を示す

図 1 から，1 ノードにおいては毎回の実行時間に一定の変動が見られることが分かる．図 2 から，8 ノードにおいては時折インパルスのような遅延が発生している．1 ノードと 8 ノードにおけるこのような状況を本論文ではそれぞれホワイトノイズ的な揺らぎとインパルスノイズ的な揺らぎと呼称する．

変動係数は，標準偏差を平均値で割った値である．変動係数の値を見ると，8 ノードの方が高くなっているが，これはインパルスノイズによって変動係数が大きな値を示していると考えられる．

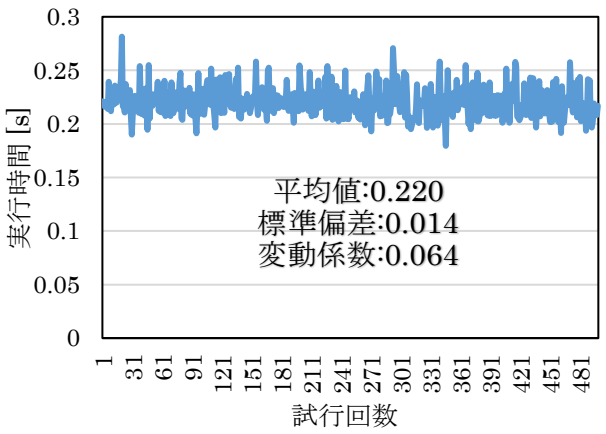


図 1　1 ノードにおける実行時間の揺らぎ

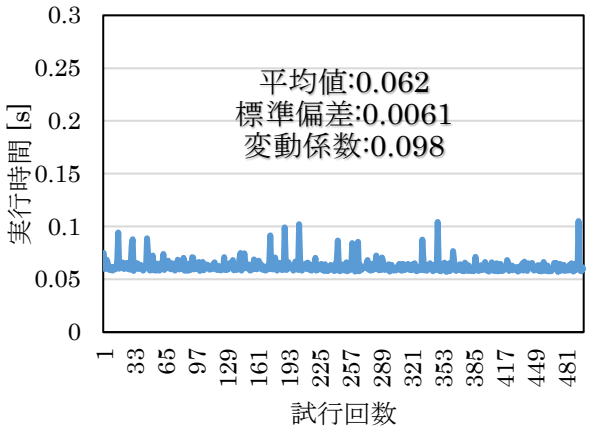


図 2　8 ノードにおける実行時間の揺らぎ

2.2 実行時間揺らぎの原因考察

次に，この実行時間の揺らぎの発生要因について考察する．Oakforest-PACS のキャッシュ構成は，L1 データキャッシュが 1 コア当たり 32[KB]，L2 キャッシュが 2 コア当たり 1[MB]である．1 ノードの合計を計算すると約 36[MB]の L1, L2 キャッシュが存在することになる．今回の MPI による計算方法は，C = A × B という行列行列積を行う場合，C と A を行分割，B をブロードキャストして行う．よって，8 ノードにおける 1 ノード当たりのメモリ使用量は 90[MB]となる．この値を利用して，1 ノードあたりどれほど L1, L2 キャッシュからデータが溢れているかを計算した結果を次の表 3 に示す．

表 3　ノード当たりのキャッシュから溢れるデータ量

ノード数	1 ノード当たりの メモリ使用量[MB]	L1,L2 キャッシュから 溢れるデータ量[MB]
1	216	180
8	90	54

表 3 より，1 ノード当たりの L1, L2 キャッシュから溢れるデータ量が増え，キャッシュミスの回数が増加すると，大きなホワイトノイズが出現すると思われる．また，8 ノードにおけるインパルスのような遅延が見られるのは，OS の割り込み（OS ジッター[9]）が考えられるが，断言できるほどではないため，具体的な要因分析は今後の課題のひとつである．この突発的な遅延の対策については，実時間のブレを考慮した探索[6]において検討されており，今回の評価対象とはしない．

3. d-spline と反復一次元探索

3.1 一次元 d-Spline

推定に用いる近似関数は実測データに柔軟に追随し，少

ない標本点でも解が得られ，計算量が少ないことが要求される．そこで，微分連続性を用いない離散型の近似関数（d-Spline と呼ぶ）が用いられている．この d-Spline による推定も標本点推定法と同様，少数の標本点を用いて探索するため，自動チューニングの時間は全探索より抑えることができる．また，最適な性能パラメータを推定する確率も高い．この近似関数 d-Spline について説明する．

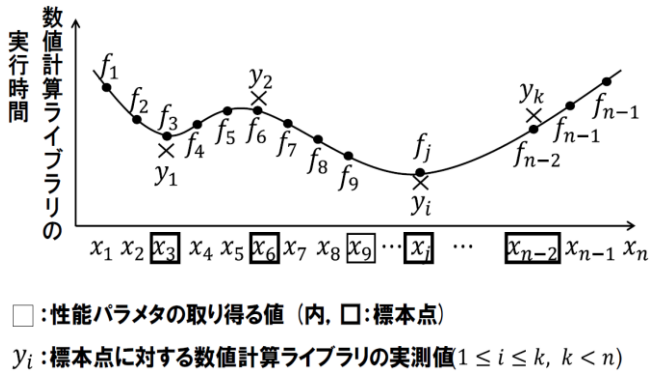
まず，d-Spline を近似関数 $f(x)$とし，n個の離散点x_j上の値$f_j = f(x_j), 1 \le j \le n$で表現する．

$$\boldsymbol{f} = (f_1, f_2, f_3, \cdots, f_j, \cdots, f_n)^t$$

ここで，tは転置を表す．滑らかさを出すために，nは性能パラメータの取り得る値の個数Nより十分大きく取る$(N < n)$．次に，性能パラメータの取り得るN個の値からk個の実測データを得る．それを$y_i(1 \le i \le k)$とする．

$$\boldsymbol{y} = (y_1, y_2, y_3, \cdots, y_i, \cdots, y_k)^t$$

近似関数$\boldsymbol{f}$と実測データ$\boldsymbol{y}$, および性能パラメータの取り得る値との関係を図 3 に示す．図 3 において，横軸は，近似関数 $\boldsymbol{f}$ の定義域である．定義域のうち四角で囲んだ値が性能パラメータの取り得る値となるが，すべてが標本点となるわけではない．縦軸は，$\boldsymbol{f}$の形状と，標本点に対する実測データy_iであり，単位は実行時間となる．



□: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$1 \le i \le k,\ k < n$)

図 3　近似関数$\boldsymbol{f}$と実測データ$\boldsymbol{y}$の関係

$\boldsymbol{y}$のみでは，個数はy_iよりf_jが多いため，$\boldsymbol{f}$を規定するために$\boldsymbol{f}$の滑らかさを 2 階差分で表す．

$$|f_{j-1} - 2f_j + f_{j+1}|, 2 \le j \le n-1$$

この近似関数$\boldsymbol{f}$を次の評価式で選ぶ．

$$min(\|y - Ef\|^2 + \alpha^2 \|Df\|^2)$$

ここで，EとDは図 4 に示すような行列であり，サイズはそれぞれk×n，$(n-2)$×nである．Eは実測データy_iと近似関数f_jの距離を，Dは近似関数の滑らかさを表す．αは実測データへの追随の度合いを表し，小さく設定するほど実測値に追従し，大きくするほど直線になる．

評価関数$min(\|\boldsymbol{y} - E\boldsymbol{f}\|^2 + \alpha^2 \|D\boldsymbol{f}\|^2)$は，次の最小二乗問題を$\boldsymbol{f}$について解けばよい．

$$min(\|\boldsymbol{b} - Z\boldsymbol{f}\|^2)$$

Zと$\boldsymbol{b}$は図 5 のように表すことができる．fill-in（零要素が非零要素になること）を抑えるために，E^tをEと$\boldsymbol{y}$の左側から掛けるように変形を行なっている．

この最小二乗問題を解く手法として，Zを直行行列Qと上三角行列Rの積に分解する QR 分解を行なう．QR 分解には Givens 変換法を用いる．$\|Q\| = \|Q^t\| = 1$に注意すると，$\|\boldsymbol{b} - Z\boldsymbol{f}\|^2$は$\|Q^t\boldsymbol{b} - R\boldsymbol{f}\|^2$と変換することができ，後退代入により$\boldsymbol{f}$が求まる．このとき，標本点を追加するたびに$Z$に戻って Givens 変換を施す必要はない．$k$−1 個目の標本点を追加する場合は，$k$個の標本点から得られた行列$R$に 1 行追加して，この 1 行にのみ Givens 変換を施せばよい．このように，近似関数 d-Spline は一次元の場合，行列Rを再利用することで$\boldsymbol{O}(n)$の計算量で求めることができる．

$$E = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & 0 & \\ & & 0 & 1 & & & \\ & & & & 0 & 0 & 1 \\ 0 & & & & & \ddots & \\ & & & & & 0 & 1 \end{pmatrix},\ D = \begin{pmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & 0 & \\ & & & & \ddots & & \\ 0 & & & & & 1 & -2\ 1 \end{pmatrix}$$

図 4　E, D の構造

$$Z = \begin{pmatrix} E^tE \\ \alpha D \end{pmatrix},\ b = \begin{pmatrix} E^t\mathrm{y} \\ 0 \end{pmatrix}$$

図 5　Z, $\boldsymbol{b}$ の構造

3.2 標本点逐次追加型性能パラメータ推定法(IPPE)

標本点逐次追加型性能パラメータ推定法は，d-Spline による最適値推定の手順である．まず初めに，初期の d-Spline を計算するための標本点の選択を行う．反復一次元探索で用いている一次元 d-Spline では，一次元の直線上を三等分する 4 点を計測し，4 点の計測結果から初期の d-Spline を計算する．d-Spline の計算が終わると，次に追加する標本点を選択する．選択基準は次の通りである．

1. d-Spline における最適値となる標本点
2. 未計測の標本点のうち，d-Spline の 2 階差分の値が最大となる標本点

1.の基準が優先され，もし d-Spline の最適値となる標本点が計測済みだった場合，2.の標本点が選択される．

その後最適値が 3 回連続して同じ点となるまで逐次的に標本点を選択して追加，d-Spline の計算を繰り返す．以上が一次元探索の仕組みである．

3.3 複数性能パラメータ推定法-反復一次元探索

我々は，複数の性能パラメータを，多次元の d-Spline を求めて同時に推定する手法の研究[]を行なってきた．その手法では，一度にすべての性能パラメータ空間の近似を行なうことができる．しかし，d-Spline を二次元，三次元で求めようとすると，d-Spline を求めるために必要となるメモリ量と計算量が指数的に増加してしまう．一次元から三次元までの d-Spline のメモリ量と計算量を表 4 に示す．表中

表 4　d-Spline のメモリ量と計算量

次元数	メモリ量	計算量
一次元	$O(n_1)$	$O(n_1)$
二次元	$O(n_2^2 \times n_1)$	$O(n_2^3 \times n_1)$
三次元	$O(n_2^2 \times n_3^2 \times n_1)$	$O(n_2^3 \times n_3^3 \times n_1)$

のn_1, n_2, n_3は性能パラメータの取り得る値の数を表している．

表 4 より，複数性能パラメータの同時推定が一次元の d-Spline により可能であれば，メモリ量と計算量を大きく削減できる．反復一次元探索は，一次元の d-Spline を用いて効率的に複数性能パラメータを同時に推定することができる．これから図 6 を用いて反復一次元探索の説明を行う．

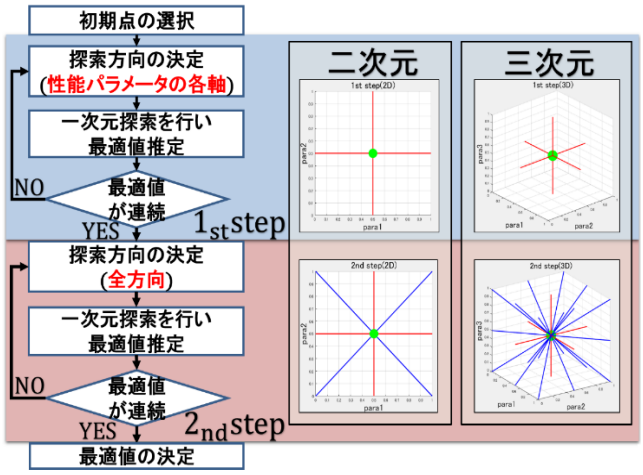


図 6　反復一次元探索の全体図

最初に，探索を開始する初期点を選択する．その次の青（1^{st}step）と赤（2^{nd}step）で分かれている処理は，赤字の部分以外は同じである．1^{st}step と 2^{nd}step における共通の処理は，探索方向の決定と一次元探索による最適値推定を最適値が連続するまで反復するという流れである．「最適値が連続」とは，2 回連続して同じ点を一次元探索の最適値と推定することである．

次に，なぜ1^{st}step と 2^{nd}step に分かれているのかを説明する．右側の 4 つのプロットは左の 2 つが二次元性能パラメータ空間，右の 2 つが三次元性能パラメータ空間における探索方向を示しており，それぞれの軸（para1,para2,para3）は性能パラメータである．1^{st}step のときの空間の探索方向は，緑の点を基準として赤の直線で示した性能パラメータの各軸方向（二次元 2 方向，三次元 3 方向）となる．次に，2^{nd}step のときは青の直線で示した方向を加えて全方向（二次元 4 方向，三次元 13 方向）で探索を行なう．各方向について基準点の両隣の 2 点を計測するため，二次元では 1^{st}step は最大 4 点，2^{nd}step は最大 8 点の周囲の点の計測が必要である．三次元では 1^{st}step は最大 6 点，2^{nd}step は最大 26 点の周囲の点の計測が必要である．もし基準点が性能パラメータ空間の境界上の点の場合，境界を超える点については計測を行なわない．また，すでに探索済みの方向に関しても計測を行なわないため，常にすべての周囲の点を計測しているわけではない．

仮に最初から全方向を探索しようとすると，この青の直線の方向に関しても計測を行なう必要がある．三次元の場合を見ればわかるように，全方向を調べるため 26 点の計測が必要となり，1^{st}step の性能パラメータの各軸方向を調べるときの 6 点と比べると 20 点も増えてしまう．1^{st}step で示した赤の直線の方向のみで性能パラメータ空間をある程度探索を行い，その後2^{nd}step で細かく探索を行なうことで，推定の精度はほぼ変わらずに計測を行なう回数を減らすことができる．この効果は次元数が増えるほど大きくなる．

4. 提案手法

実行環境における揺らぎを測るため，同じ性能パラメータを複数回計測し，その実行時間を利用して分散を求める．しかし，実行時間が長いパラメータを何度も実行してしまうと，総実行コストが増加するという問題が発生する．これを解決するため，一次元探索終了時の推定値を調べ，直前の一次元探索終了時の推定値からの減少率が 5%未満となった場合，揺らぎの計測を開始するようにしている．揺らぎの計測回数は，初回は 18 回とし，以後は揺らぎの大きさに比例して多くなるようにし（揺らぎが平均値に占める割合が 1%なら 3 回，5%なら 15 回），上限を 18 回とする．

初回と上限の回数を 18 回としている理由は次の通りである．揺らぎを求めるために同一性能パラメータで複数回の計測を行うが，自動チューニングの一般的な課題としてできるだけ少ない実行回数で推定することが挙げられる．そのため，同一性能パラメータでの実行も極力減らす必要がある．図 7 は，片側確率が 95%となる標準正規分布の値（1.645）を利用し，それが t 分布表で自由度を変化させた際の片側確率の推移を表す．0.95 の 99%である 0.9405 以上であればその性能パラメータにおける実行時間を求めるには十分の信頼度とし，0.9405 を超える一番小さな自由度は 17 となるため，サンプル数上限は自由度+1 の 18 回とした．

複数回計測した結果を利用し，実行環境における揺らぎを測定するとともに，平均値から本当にその性能パラメータが直前の一次元探索での推定値よりも優れているのか比較を行い，実行時間が短い方の一次元探索の推定点から探索を続行する．提案手法におけるこの流れの説明を図 8 に示す．

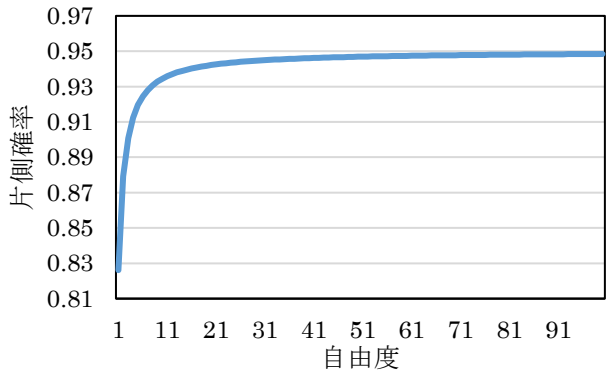


図 7　X=1.645 での t 分布表の自由度と片側確率の推移

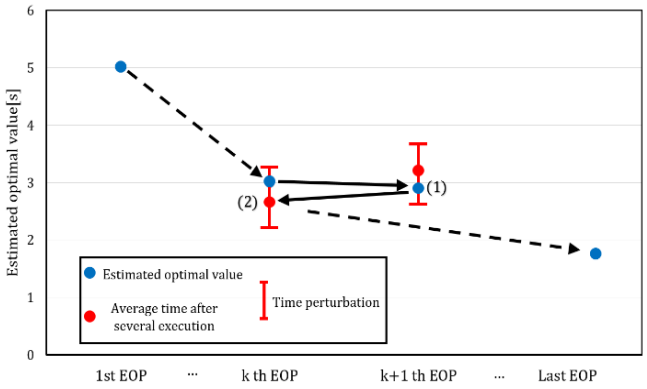


図 8　提案手法における揺らぎの計測

図 8 において，EOP(estimated optimal point)は一次元探索で推定されたパラメータで，左から順番に 1 番目，k 番目，k+1 番目となり，Last EOP が最終的な揺らぎ計測機能を付加した提案手法の推定点である．k 番目までの一次元探索では，直前の一次元探索からの推定値減少率が 5%以上であり，既存の揺らぎを考慮していない反復一次元探索（以下従来手法とする．）と同じ挙動を示す．次に，k+1 番目の一次元探索が終了した際(1)，推定値減少率が 5%未満となり，最適値は現在の推定値に近い値だと判断する．そして，このパラメータであれば連続して実行しても実行時自動チューニングにおける総実行コストへの大きなオーバーヘッドにならないとみなし，揺らぎを計測するために残りステップ数の 10%の回数だけそのパラメータを実行する．計測が終わったら，k+1 番目と k 番目のパラメータのうち，平均実行時間が短いパラメータの方から探索を続行する(2)．

従来手法では，各性能パラメータは 1 回ずつ計測し，探索を行っていた．本論文では，推定された性能パラメータは偶然推定に利用した 1 回の実行時間は速かったが，その後何度も実行するとその性能パラメータは揺らぎが大きく不安定である可能性を考慮する機能も追加する．そのために，従来の反復一次元探索が終了した後も，最適な性能パラメータは別である可能性を考慮し，ステップ毎に実行済みの各性能パラメータの平均実行時間を計算し，それが最短となる性能パラメータを実行するようにした．これにより，反復一次元探索終了後に実行する性能パラメータは適切なものへと切り替わっていく．この様子を図 9 に示す．図 9 において，横軸は反復一次元探索終了後何ステップ目かを表しており，0 は反復一次元探索が終了した際の最適値を表している．反復一次元探索が終了した直後に実行される性能パラメータは，推定されたパラメータ a であるが，実測された時間は 0.7 秒に近い値であり，最適値である可能性が下がるため，より平均実行時間が短いパラメータ b を実行する．パラメータ b の実行をしたところ，再び平均実行時間が悪化し，より平均実行時間が短いパラメータ c が選ばれる．このように，貪欲法的に毎ステップ最短の平均実行時間となるパラメータを実行することで，最終的にはパラメータ e で落ち着く．これにより，平均の実行時間が遅い性能パラメータが偶然推定されてしまったとしても，実行すべき性能パラメータを毎ステップ求めるため，総実行コストを削減できる．

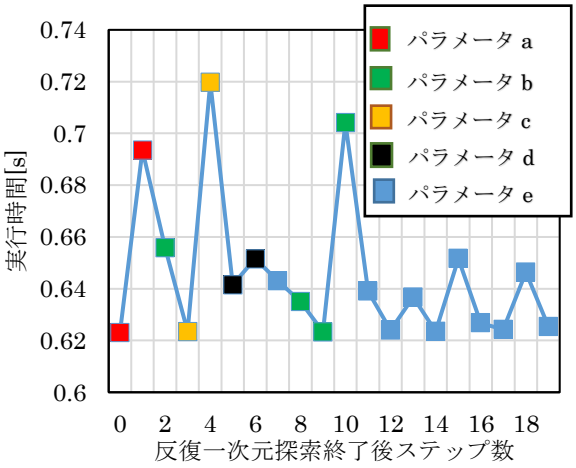


図 9　反復一次元探索終了後の挙動

5. 予備実験

Virtual library of simulation experiments[10]から power sum function と styblinski-tang function を使用し，提案手法と従来手法の評価を行う．それぞれの式は次の通りである．

Power sum function

$$f(x,y) = [(x+y)-8]^2 + [(x^2+y^2)-18]^2$$

Styblinski − Tang function

$$f(x,y) = \frac{1}{2}[(x^4 - 16x^2 + 5x) + (y^4 - 16y^2 + 5y)]$$

それぞれの関数の概形を図 10 と図 11 に示す．

これらの関数は連続関数であるが，50×50 点となるよう離散化を行い，これを性能パラメータとして扱う．また，関数値を実行時間として評価するため，関数値を 0 以上の値に変換する．関数のすべての点（2500 点）を初期点として，500 ステップの探索を行う．そして，疑似的な揺らぎ

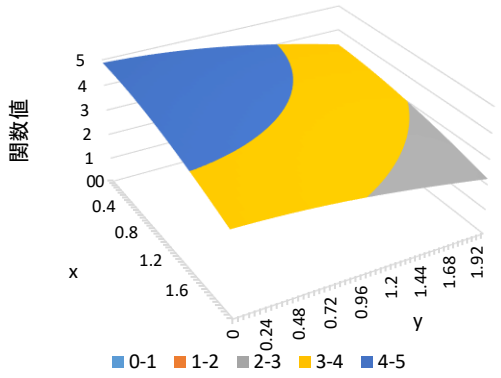


図 10　Power sum function

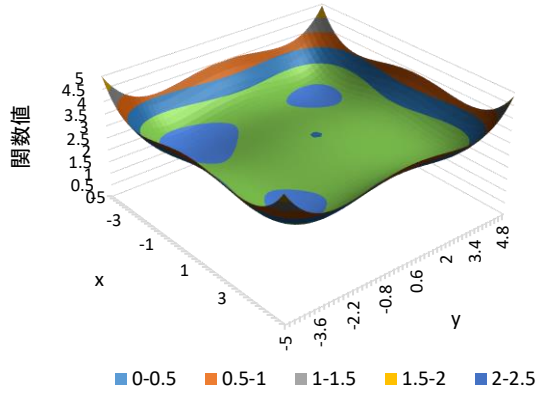


図 11　Styblinski-tang function

として，$N(0,\sigma^2)$の正規分布に従う乱数を加えた．σはテスト関数の真値の 1%,2%,…,10%の大きさで取っている．

ここでは，相対リグレットと相対ロスという二つの指標を利用して評価を行う[11]．相対リグレットと相対ロスの計算方法は次式で表される．

$$\text{相対リグレット}(rr) = \frac{T_m - T_{opt}}{T_{opt}}$$

$$\text{相対ロス}(rl) = \frac{t_m - t_{opt}}{t_{opt}}$$

上式において，T_mは提案手法における総実行コスト，T_{opt}は最適な性能パラメータにおける実行時間に全ステップ数を乗算した値である．t_mは提案手法における推定された性能パラメータにおける実行時間，t_{opt}は最適な性能パラメータにおける実行時間である．

上記 2 つのテスト関数を使用し，すべての点から 500 ステップの探索を行った際の相対リグレットと相対ロスの結果を図 12 と図 13 に示す．図 12，図 13 において，青色が提案手法，オレンジ色が従来手法を表している．相対リグレット，相対ロスはブレを大きくしていくとともに増加しているのがわかる．これは，ブレが大きくなると反復一次元探索の方向決定の際に探索すべき方向とは別の方向に探索してしまう，d-Spline が適切な当てはめになっていない，といった事象が発生し，正しく推定することが困難になっているということを表している．次に，提案手法と従来手法の差についてみると，相対リグレットも相対ロスもブレを大きくすると差が大きくなっていることがわかる．相対リグレットの結果から，提案手法はブレが大きくなっても従来手法に比べ総実行コストの増加が抑えられていると言える．相対ロスの結果からは，推定値も従来手法ほど悪化していないと言える．以上の考察から，提案手法は総実行コストの削減と推定値の改善の両方が同時に達成できていることがわかる．提案手法は従来手法に比べて最大で 46%の相対リグレット削減と 90%の相対ロス削減を達成した．

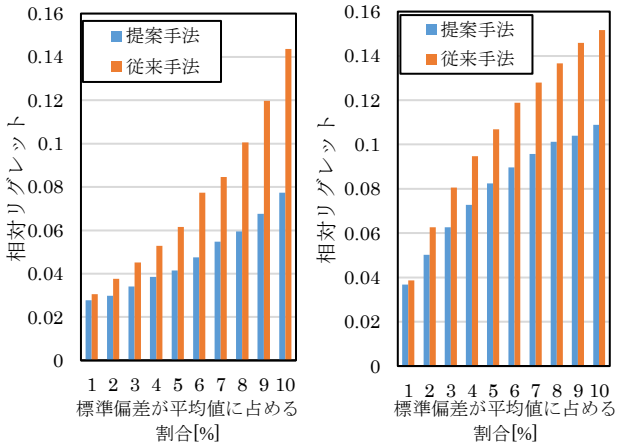


図 12　Power sum function（左）と Styblinski-Tang function（右）の相対リグレット

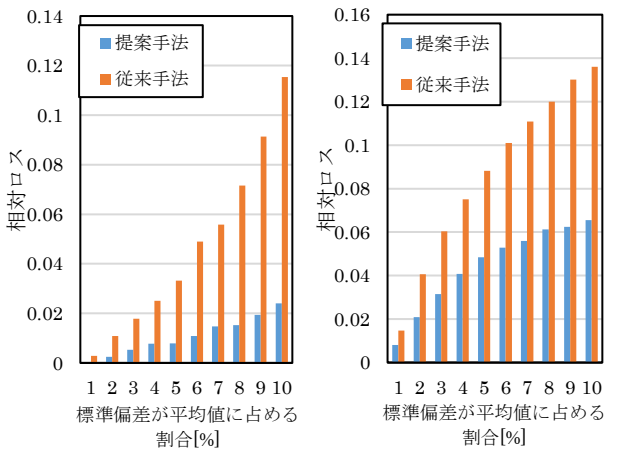


図 13　Power sum function（左）と Styblinski-Tang function（右）の相対ロス

6. 実環境での評価

2 章と同様に行列行列積問題を対象とし，実行は 1 ノードで行う．行列サイズは 2000×2000 で評価する．行列行列積ルーチンの ijk ループのうち j と k のループに対してキャッシュブロッキングを行い，そのブロッキングサイズを性能パラメータとする．今回は j,k ともにブロッキングサイズは 16,32,…,512(32 通り)とする．各性能パラメータを 1

回ずつ実行した際の実行時間の等高線図を図 14 に示す．

本評価においては，性能パラメータ空間の中心の点（j ブロックサイズ 272,k ブロックサイズ 256）を初期点として探索を開始する．

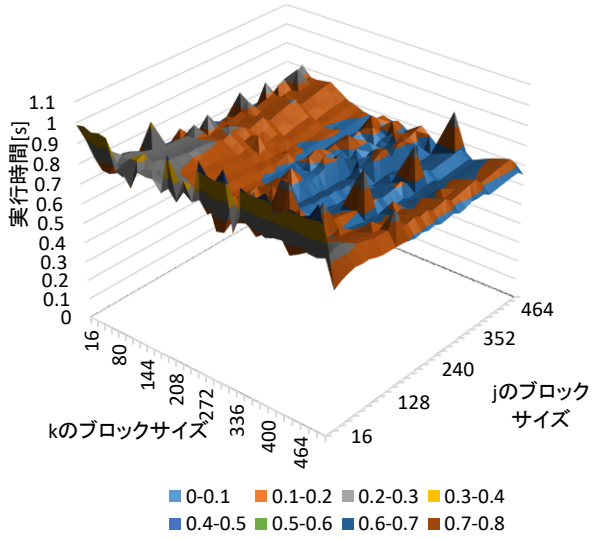


図 14　1 ノードにおけるキャッシュブロッキングの等高線図

予備評価と同様に，提案手法，従来手法のそれぞれで 500 ステップの実行を 10 回ずつ行い，500 ステップの合計時間の平均値と推定値の平均値を用いて評価を行う．結果を表 5 と表 6 に示す．

表 5　実環境での評価結果

使用手法	500 反復合計時間	500 反復合計時間相対リグレット	推定値	推定値相対ロス
提案手法 1	328.5	0.0725	0.648	0.0571
提案手法 2	328.8	0.0735	0.642	0.0473
従来手法	330.1	0.0777	0.654	0.0669
乱数探索	340.2	0.1107	0.662	0.0799
最適値	306.3	0	0.613	0

表 5 中の従来手法以外の手法の意味は次の通りである．提案手法 1：4 章の提案手法．提案手法 2：一回目の一次元探索終了時から揺らぎ計測を開始する．乱数探索：提案手法 1 の推定が終了するまでの計測回数の平均試行回数だけ乱数で性能パラメータをランダムに探索し，その後計測した内の最適な性能パラメータで実行する．

表 6　従来手法に対する相対リグレット，相対ロス比率

使用手法	従来手法を基準とした 500 反復合計時間相対リグレット比率	従来手法を基準とした推定値相対ロス比率
提案手法 1	0.933	0.854
提案手法 2	0.946	0.707
乱数探索	1.425	1.194

表 6 を見ると，提案手法 1 は従来手法より 7%の相対リグレット削減，15%の相対ロス削減できていることが分かる．最初から複数回計測を行う提案手法 2 においても従来手法よりも両方削減できているが，提案手法 1 よりも相対リグレットが大きく，相対ロスは小さい結果となった．初めから複数回計測を行うことで，性能の良くないパラメータを複数回実行してしまい，その結果総実行コストが増えたが，初めに実行時間揺らぎを測れるのでその後の推定は途中から揺らぎを計測する提案手法 1 よりも安定していたと考えられる．乱数探索は，今までの探索の結果を一切活用していないため非効率的となり，従来手法に比べて相対リグレットと相対ロスの両方が悪化したと考えられる．

従来手法と提案手法の実行例を図 15 と図 16 に示す．

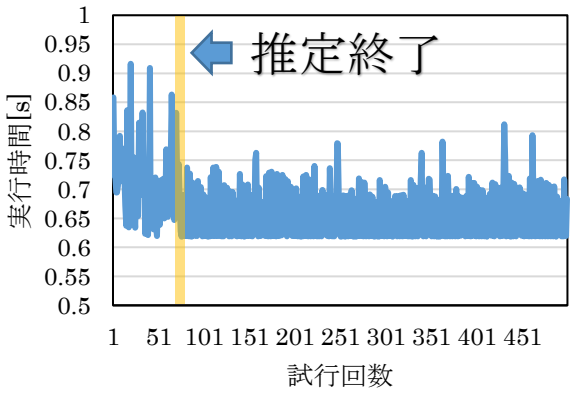


図 15　従来手法の実行例

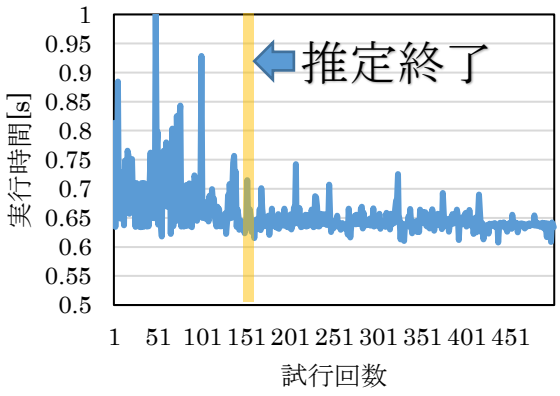


図 16　提案手法 1 の実行例

従来手法の特徴は，少ない試行で推定を終えられること（平均試行回数 52.3 回）であるが，推定後は性能パラメータを変更しないため，図 15 のように不安定な性能パラメータが推定されてしまうことがある. 図 16 が提案手法 1 の実行例だが，従来手法よりも推定には多くの試行回数がかかる（平均試行回数 88.8 回）．しかし，推定中における性能の悪いパラメータの実行は多くなく，推定後も安定した性能パラメータを実行できるため，500 反復の合計時間も短縮できたと考えられる．

提案手法と従来手法の結果を利用して片側 t 検定を行った．その結果を表 7 に示す．値が小さいほど提案手法は有意な差があることを表す.

表 7　提案手法 1，2 の 500 反復合計時間，推定値を従来手法を基準として片側 t 検定を行った結果

提案手法 1 における 500 反復合計時間の t 検定結果	0.280
提案手法 1 における推定値の t 検定結果	0.185
提案手法 2 における 500 反復合計時間の t 検定結果	0.313
提案手法 2 における推定値の t 検定結果	0.0273

提案手法 2 における推定値に関しては 0.0273 となり有意水準 0.05 で検定を行っても有意差があると言える．しかし，他の値に関しては有意水準 0.05 で検定すると有意差があるとは言えない結果となった．各手法の実行回数が 10 回ずつであったため，その回数を増やして標準誤差の値を小さくすることで，有意水準 0.05 で検定を行っても有意な差が見られるようになると考えられる．

7. まとめ

本論文では，推定を困難にする実行時間の揺らぎを考慮する機能を追加することで，自動チューニングにおける総実行コストを削減することを目的とした．まず初めに実際の環境における実行時間の揺らぎを Oakforest-PACS を用いて調べた．分析を行い，キャッシュから溢れるデータ量が増えると毎回の実行時間揺らぎが大きくなると考察した．

提案手法として，複数の性能パラメータを探索する反復一次元探索において，次の 2 点の拡張を行った．

1. 一次元探索の推定点を調べ，探索が安定したことを判定してから，推定点において複数回の計測を行い, 揺らぎを計測し, 探索の基準となる推定点を決定する拡張を行った．
2. 探索終了後も常に実行時間が最短となるパラメータを求め，実行する拡張を行った．

テスト関数を用いた評価で，提案手法は従来手法に比べて最大で 46%の相対リグレット削減と 90%の相対ロス削減を達成しており，1~10%までの全ての揺らぎの大きさに対しても提案手法の方が従来手法よりも相対リグレットと相対ロスが小さい結果となり，総実行コストと推定値の両方を同時に改善できていたことを示した．実環境における評価では，提案手法は従来手法に対し最大で 7%の相対リグレット削減と 30%の相対ロス削減を達成した．

今後の課題としては，本手法の多次元化や異なる探索手法を用いた更なる評価．現在のチューニング度合いを測定し，柔軟に探索方法を変化させる方法．最適と推定された性能パラメータにおける分散の推定などが挙げられる．

謝辞

本研究の一部は JSPS 科研費 JP17K00164, JP16H02823, JP18K19782, JP18K11340 の助成を受けて行われた．

参考文献

[1] J. Bilmes, K. Asanovic, C.-W. Chin and J. Demmel, Optimizing Matrix Multiply using PHiPAC : a Portable, High-Performance, ANSI C Coding Methodology, in: Proceedings of the 11th international conference of Supercomputing, Vol. 97, pp. 340-347 (1997).

[2] R. Clint Whaley, A. Petitet and J.J. Dongarra, Automated Empirical Optimization of Software and ATLAS project, Parallel Computing, Vol. 27, pp. 3-35 (2001).

[3] 片桐孝洋，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，(2004).

[4] T. Tanaka, T. Katagiri and T. Yuda, d-Spline Based Incremental Parameter Estimation in Automatic Performance Tuning, in: Proceedings of the 8th International Conference on Applied Parallel Computing: State of the Art in Scientific Computing, LNCS, Vol. 4699, Springer, pp. 986-995 (2007).

[5] M. Mochizuki, A. Fujii and T. Tanaka, Fast Multidimensional Performance Parameter Estimation with Multiple One-dimensional d-Spline Parameter Search, in the Twelfth International Workshop on Automatic Performance Tuning (iWAPT2017), pp. 1426-1433 (2017).

[6] G. Fan, M. Mochizuki, A. Fujii, T. Tanaka and T. Katagiri, D-Spline Performance Tuning Method Flexibly Responsive to Execution Time Perturbation, 12th International Conference on Parallel Processing and Applied Mathematics (PPAM2017), pp. 381-391, (2017).

[7] T. Tanaka, R. Otsuka, A. Fujii, T. Katagiri and T. Imamura, Implementation of d-Spline-based Incremental Performance Parameter Estimation Method with ppOpen-AT, Scientific Programming 22, pp. 299-307 (2014).

[8] Oakforest-PACS スーパーコンピュータシステム, https://www.cc.u-tokyo.ac.jp/supercomputer/ofp/system.php

[9] F. Petrini, D.J. Kerbyson and S. Pakin, The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, pp. 55-, (2003).

[10] Virtual Library of Simulation Experiments: Test Functions and Datasets, https://www.sfu.ca/~ssurjano/index.html

[11] Suda Reiji, A Bayesian Method for Online Code Selection: Toward Efficient and Robust Methods of Automatic Tuning, in the Second International Workshop on Automatic Performance Tuning (iWAPT2007), pp. 23-31, (2007).