

タイルサイズ自動調整ツールにおける ヒューリスティックスの実装と比較

幸 朋矢^{1,a)} 佐藤 幸紀¹ 遠藤 敏夫¹

概要：ループ最適化手法の一つであるループタイリングにおいて、タイルサイズの選定は重要なテーマとなっている。我々は、タイルサイズ自動調整ツールに焼き鈍し法やネルダーミード法などのヒューリスティックスを実装すると同時に、開発しているタイルサイズ調整専用の探索アルゴリズムの改良を行った。これら各ヒューリスティックスの比較実験を行ったところ、収束スピードに関して我々の手法が他を上回ることを確認した。最終的に得られるタイルサイズの性能としては、他のメタヒューリスティックスが理論上の最速に対し約 90%という性能に留まるなか、我々の手法が約 99%の性能を達成することを確認した。

1. はじめに

近年の計算機システムでは、計算高速化のためにアクセラレータを混在させた環境が主流になっており、これらのアクセラレータはそれぞれ内部に特殊なメモリ構造を持っているものが多い。Xeon Phi を例にとると、メニーコアという設計から、L2 キャッシュや MCDRAM より選択的に構成される L3 キャッシュの構造は、従来の CPU のそれと大きく異なっている。この特殊なキャッシュ構造について考慮したチューニングを施すことがこれらのアクセラレータのポテンシャルを最大限引き出す鍵となるが、様々な対象及び特殊な構造に手作業で最適化を施すことはコストが高く、自動化が望まれている。

キャッシュ構造に代表される多重メモリ階層においては、データの局所性が性能に対して重要となる。ここで、ループタイリングというループ最適化手法を紹介する。これは、多重ループを細かく分割しそのスケジュールを変化させることで、データの参照局所性を最大限に高めるように調整し、その結果として性能向上を狙うというものである。このループタイリングは以前は人の手でその変換作業を行う必要があったが、近年開発が活発な Polyhedral コンパイラというものを使うことで、完全に自動的に変換することが可能となる。

ここで重要となるのがそのループの分割の大きさ、つまりタイルサイズである。このサイズを適切に設定することで最大のポテンシャルを引き出すこともできれば、サイズ次第では逆に本来のコードより性能が下がることもある。

歴史的に、従来の CPU のキャッシュ構造を想定したタイルサイズ調整の研究は多く行われてきているが、近年の新しいデバイス、アクセラレータ等は特殊な構造を持つため、従来のテクニックを単純に適用することはできないケースがある。

そこで、我々はこういった最新の状況にも対応するため、メニーコア環境を想定したタイルサイズ自動調整ツール PATT (Polyhedral compiler based Auto Tile size optimizer) を開発している [1]。本ツールは、ループタイリングの自動化として Polyhedral コンパイラを内部で使用し、各種タイルサイズをひとつひとつ試しながらの速度計測を可能とする。さらに、タイルサイズ調整に特化したオリジナルの探索アルゴリズムも実装し、その有効性も確認されている。しかしながら Polyhedral コンパイラのデフォルトタイルサイズに対しての有効性は確認できているものの、他のメタヒューリスティックスとの比較は未だ行われていなかった。

本報告では、PATT の探索アルゴリズムをより広い状況に対応できるタイルサイズ調整専用アルゴリズムに改良する。また、他のメタヒューリスティックスとして焼き鈍し法とネルダーミード法を他の自動チューニングツールから我々のツール内に移植・再調整し、我々の手法との比較実験を行う。

2. タイリングとタイルサイズ自動調整ツール

2.1 タイリング

タイリングとは、データ局所性を高め性能向上を狙う、ループ最適化手法のひとつである。具体的には、ある多重ループがあった時そのスケジュールをより細かい領域に分

¹ 東京工業大学 学術国際情報センター

^{a)} yuki.t.ab@m.titech.ac.jp

割するようなループ変換を施すことで、計算に用いる配列の参照局所性を高め、キャッシュヒット率を上げ、その結果性能を向上させるという流れとなる。この時の分割サイズであるタイルサイズだが、大きすぎても意味がなく、小さすぎるとキャッシュ性能のポテンシャルを十分に引き出すことができない。実行する環境に合った、適切なタイルサイズを設定することが望まれている。このテーマでの研究は昔から行われてきたが、今日のメニーコア型 CPU はその内部に特殊なキャッシュ構造を持っているため、これらを対象として適切なタイルサイズを見つけるとなった場合、問題はより複雑となる。

ここで、タイルサイズ調整という問題が重要かつ複雑であることを示すために、各タイルサイズとその性能を調査しそれらの関係をヒートマップとして可視化したものを図 1 にまとめた。これは各タイルサイズにてバイナリを作成、実行、時間計測を行ったものである。ドット 1 点が 1 つのタイルサイズの組み合わせに相当し、1 回の時間測定の結果となる。実測時間を色としているため、青が速く、赤が遅いということになっている。この図から分かることは、カーネルごとにヒートマップの特徴が違うということである。一概にタイリングといってもどのようなループ構造・演算処理かによってその適切なタイルサイズは変わってくる。また、タイルサイズの変化というだけでこれだけ性能に差が出るということもこの図から見て取れる。このヒートマップは問題サイズ全体でなく、最速点が存在する付近の範囲を切り出したものだが、それでも最速と最遅では 2 倍以上の差があることが分かる。この結果からも分かる通り、タイルサイズ調整は性能を追求する場合の重要な課題となっている。

次に、具体的なタイリング変換処理の実現方法について確認する。人力でコード変換を行うと大きな労力が必要だが、近年、Polyhedral コンパイラというもの盛んに研究されており、これらのツールを使うと完全に自動でのループ変換が可能となる。タイリングだけでなく、ベクトル化やパラレル化まで自動的に行ってくれるものもある。その中でも現在、Polly[2] と Pluto[3] という 2 つのツールが特に有名だが、そのうち LLVM-IR[4] のレベルでループ変換処理を行う Polly を今回ピックアップする。Polly はタイルサイズを IR 変換時にユーザーが指定することができ、図 1 もこの機能を利用して取得した。この時、タイルサイズを指定しない場合、デフォルトのタイルサイズである 32 というパラメータが各次元に適用される。このタイルサイズは広い状況で比較的良好に動作するとされており、実際に相応の性能が出るケースが多い。しかしながらさらなる最適化の余地が存在することが多く、さらにメニーコア環境においては特に遅くなってしまうケースが存在することを確認している。

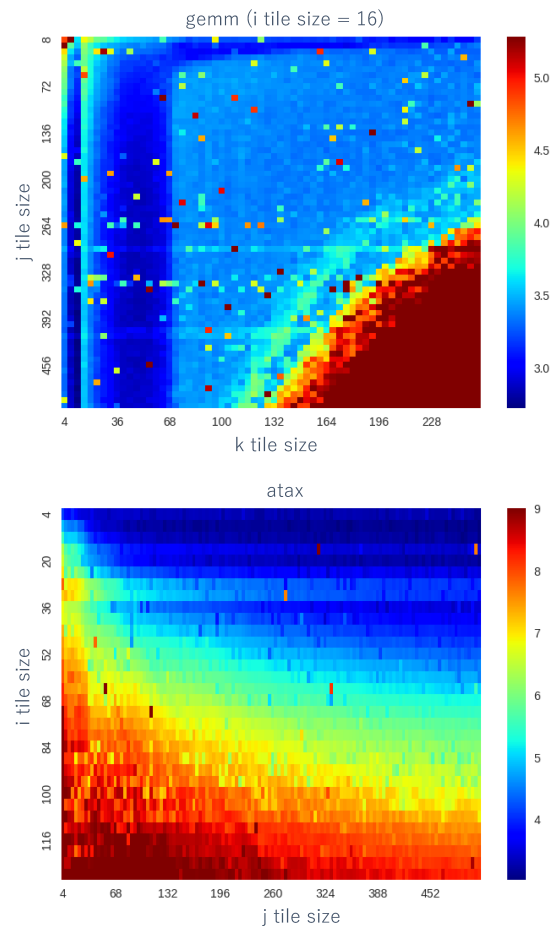


図 1: タイルサイズとその性能を表すヒートマップ

2.2 タイルサイズ自動調整ツール PATT

このようなメニーコア環境におけるタイルサイズの調整という問題に対処するため、我々はタイリングパラメータ自動調整ツール PATT を開発している。PATT には複数の動作モードがあるが、基本はタイルサイズの自動チューニングを行うものとなる。図 2 に自動チューニングモードの場合の PATT の概要を示す。インターフェイスとしては、ソースコードを代表とするコンパイル情報と PATT 設定ファイルを入力とし、PATT 内部での自動チューニングにより最終的に割り出した最速タイルサイズの情報とそのタイルサイズでビルドされたバイナリを出力とする。Polly を使用し特定のタイルサイズでバイナリをビルドした後、そのバイナリを実行し、その計測データを見て再度バイナリをビルド、といった流れを繰り返す。この時の探索アルゴリズムとして、後述する複数のヒューリスティクスを動作モードとして分けて用意しており、切り替えて実験することが可能となっている。

PATT 設定ファイルでは、PATT の動作モード、カーネルのループサイズやバイナリ実行時スレッド数、Polly による最適化の対象としたいカーネルファイルの指定などを行う。また、PATT の各ヒューリスティクスに使用する定数もここで指定する。

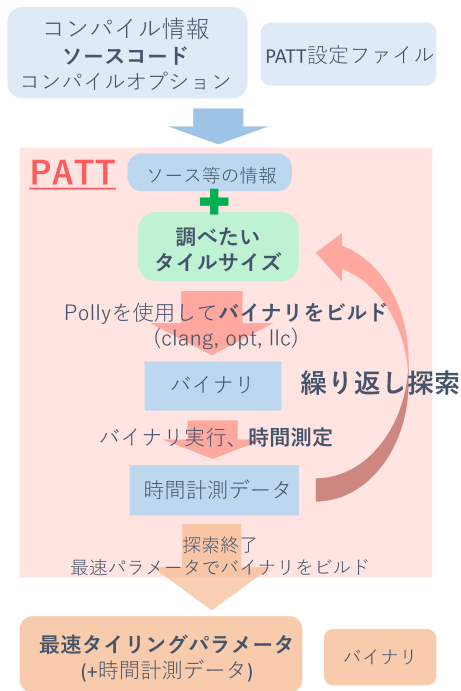


図 2: PATT 概要

3. タイルサイズ調整特化ヒューリスティック ス S-PATT

本節では我々が開発しているタイルサイズ調整に特化したヒューリスティックスである S-PATT の説明を行う。S-PATT はこれまで評価してきた PATT の実装 [1] の面で見直しを行ったアルゴリズムとなる。

3.1 ロードバランス

メニーコア環境において、小さい問題サイズに対して比較的大きいタイルサイズはロードインバランスを引き起こすということが分かっている。特に現在の Polly は最外ループのみを OpenMP でパラレル化するため、最外ループのタイルサイズがロードバランスの問題に直結することになる。図 3 は、Polly で最適化した行列積カーネルを Xeon Phi (Knight Landing) 64 スレッドで動作させた際のグラフとなる。この時の問題サイズとしては、最外ループサイズが 2000 となっている。2000/64 = 31.25 から、32 というタイルサイズの時、丁度 1 スレッドで 1 回分のループが回る (1 イテレーションと表記する) という計算となる。ここからタイルサイズが増えていくと徐々にロードインバランスが進むため、速度としても段々と遅くなっていく。また、31 というタイルサイズの場合、ほとんどのスレッドでは 1 スレッド 1 イテレーションだが、少量だけ 1 スレッドに 2 イテレーション入ることにより、極端なロードインバランスを引き起こす。その結果、31 や 30 といったタイルサイズでは速度が急に遅くなるといったことが起こる。ロードバランスが良い点は他にもあり、1 スレッド 2 イテ

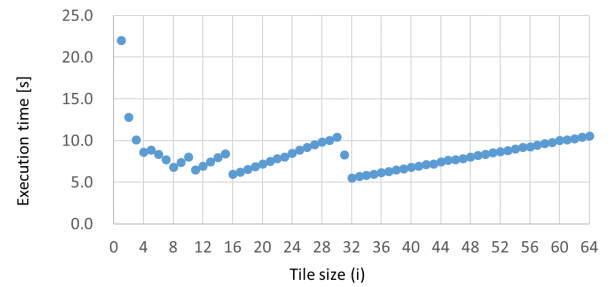


図 3: 最外ループタイルサイズのみ変化させた時の性能変化 (gemm XL サイズ)

レーションが丁度収まるのがタイルサイズ 16 となる。実際に、16 というタイルサイズはその周辺のタイルサイズの中で局地的に最速となっている。

ここから分かる通り、メニーコア環境においてロードインバランスを回避するためには、全てのスレッドに丁度良く収まるイテレーション数になるようなタイルサイズが必須であることが分かる。我々はここに着目し、最外ループのタイルサイズをロードバランスが最大限良くなるような点を候補として最初にチェックするという処理を考えた。イテレーション数としては念のため 1,2,4,8 の 4 点を採用する。導出としては以下の式を使用する。

$$\begin{aligned} \text{tile_size_candidate} \\ = \text{ceil}(\text{loop_size} / \text{num_of_threads} / \text{iteration}) \end{aligned} \quad (1)$$

さらにカーネルの特性によってこのピンポイントの候補点から多少ズレるようなケースを考慮し、この式で導き出した 4 点のタイルサイズに対し、それぞれ前後 1 ずつずらしたタイルサイズも候補点に加えるという安全策を取る。よって最外ループタイルサイズの候補は合計で 12 点となる。

S-PATT のアルゴリズムとしては、3 重ループ以上のカーネルの場合、最初にこの最外ループタイルサイズの決定を独立に行う。この時、内側ループのタイルサイズは全てデフォルトタイルサイズである 32 を指定する。計算によって導出された 12 点全ての候補点の測定を行い、その中の最速のものを最外ループタイルサイズとして採用する。採用されたサイズに固定したまま、次に内側ループタイルサイズの探索へと処理を進める。

3.2 探索アルゴリズム

最外ループタイルサイズが決定した後は、内側ループタイルサイズの探索となる。3 重ループの場合の最外ループについては前節で議論したので、ここでは残り 2 重ループが対象となる。2 重ループカーネルの場合は最初からこの探索アルゴリズムを使うことになる。

図 4 に探索イメージを示す。最初に問題サイズ、つまりループサイズの最大の空間を均等に N-1 分割し、両端も

含め全部で N 個の測定点を列挙する。この図では例として $N = 5$ としてある。2つのループを同時に対象とするため、これを2つの次元方向に行う。この時細かい工夫として極端に小さいタイルサイズを省き、さらにタイルサイズを4の倍数に揃えるという操作をする。前者は極端に小さいタイルサイズの場合性能が極端に遅くなるケースがありそれを除外するため、後者はベクトル化等の影響で4の倍数からズレると性能が下がるケースがありそれを避けるためである。列挙された測定点对し、タイルサイズの小さい方から、より内側のループの方から、順番に探索していく。この時、山登り法 [5] により速度向上が見られなくなった時点で次点の探索を打ち切る。今まで実験してきたカーネルのヒートマップでは、全体的に広い勾配を持ち極値をあまり持たないという特性を確認しているため、この山登り法による探索打ち切りを効果的に活用することが可能となっている。尚、最内ループタイルサイズの打ち切り判定は素直に前回計測点と今回計測点の比較をするが、ひとつ外側のループのタイルサイズの探索においては、各最内ループタイルサイズの探索結果を使用し、その次元での最速点同士を比較するというを行う。こうして一旦この平面においての最速点が発見できたら、問題サイズを最速点の近傍に設定し直し、再度測定点の列挙から同様の操作を行う。これを繰り返し、十分に収束したら終了する。

このような最初は広く粗く、徐々に狭く細かく探索をしていくといったアルゴリズムを採用した理由として、以下の2つが挙げられる。ひとつは単純に探索回数の削減のためである。問題空間の端や中心をスタートとして等差や等比で探索していった場合、最速点の位置によっては探索に非常に時間がかかることになる。スタートを広く粗い探索としておけば、どのようなケースであっても一定のコストで最速点付近を最初に見つけることが可能となる。もうひとつは、性能に変化のない領域を広いステップで超えるためである。ヒートマップで可視化すると、様々なカーネルにおいて、本当に知りたい最速点のある付近の領域は狭く、それ以外の性能に大きな変化がない領域が大半となる。この性能に大きな変化がない領域を平地と呼ぶことにすると、平地を探索することはなるべく避けたい。そのため、広いステップでこの平地を超え、探索の有効性のある領域を早々に見つけることが重要となる。

4. 実装したメタヒューリスティックス

今回我々は、S-PATT との比較のために焼き鈍し法とネルダーミード法という2つのメタヒューリスティックスを PATT 内部に実装した。これら2つのメタヒューリスティックスは Orio[6] という自動チューニングシステムツールのソースコードを移植し、調整している。

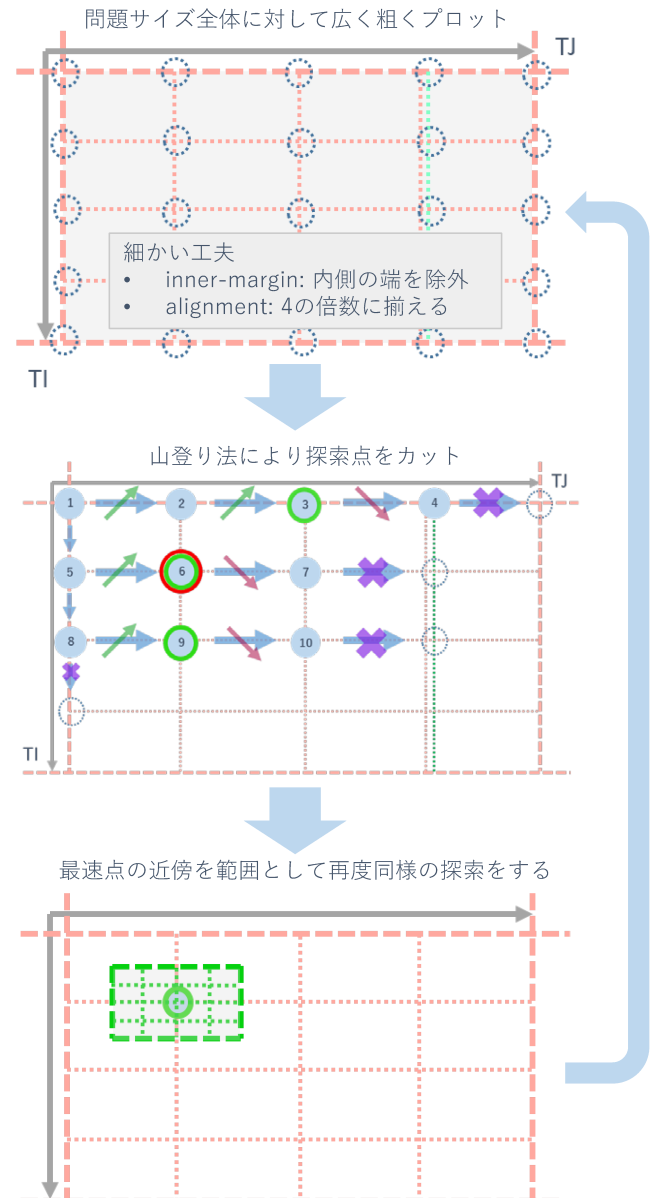


図 4: S-PATT 探索イメージ

4.1 焼き鈍し法

焼き鈍し法は、ランダム性を取り入れたメタヒューリスティックスのひとつである。Simulated Annealing と呼ばれるため、以後 SA と表記する。図 5 に SA の探索イメージを示す。初期点として1点を取り、次にその近傍の1点を選択する。今回、タイルサイズ探索ということで整数問題かつ多次元問題のため、近傍は各次元の距離1の範囲とし、選択はそこからランダムに行うとしている。選択した近傍の1点を測定し、元の点の測定結果と比較する。速くなった場合はその新しい点に移動する。遅くなった場合でも、“温度”と呼ばれる値を元に遷移確率を計算し、確率的に移動する。この温度は、最初高く、徐々に下がっていく。高ければ高い程遅い点への移動が発生しやすくなる。この点の選択と移動を繰り返し、温度が十分に低くなった

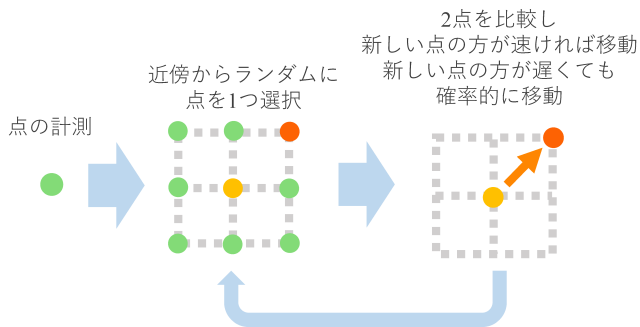


図 5: SA 探索イメージ

ところで探索を終了する。この時の最終地点が最速点となる。

SA のメリットとしては、ローカルな極値を超えられるということが挙げられる。極値が複数あるような探索空間であっても、特に序盤は性能の低い点へも確率的に遷移し広く様々な領域を移動することができる。温度が下がるためその遷移も徐々に安定していき、全体の最適値へ収束しやすいという特徴がある。デメリットとしては、近傍を一步步ずつ探索していくため、その移動幅によっては移動が遅くなるということが挙げられる。

4.2 ネルダーミード法

ネルダーミード法 (NelderMead method) は、シンプレックスというコンセプトを用いるメタヒューリスティックスである。以後 NM と表記する。図 6 に NM の探索イメージを示す。最初に、探索空間の次元+1 個の点の群を用意する。これをシンプレックスと呼ぶ。このシンプレックスの形状を変化・移動させながら、探索していくというのが NM の基本となる。シンプレックスが用意できたら、そのシンプレックスを構成する点を全て計測する。その各点の速度関係に応じて Reflection, Expansion, Contraction, Shrinkage という 4 つの操作の中からひとつを選び、点を移動する。その結果新しいシンプレックスが出来上がるため、この新しい点を測定する。これらの操作を十分な回数繰り返し、最小値へと近づけていく。

NM はシンプレックスの形状がダイナミックに変化するため、勾配が急な部分では一気に大きく移動し、勾配が緩やかな領域では小さいシンプレックスで少しずつ移動することになる。そのため収束スピードが速いということが長所となる。短所としては、4 種類の操作に使用する係数によって挙動が変わり、値によってはローカルな極値に陥りやすくなることが挙げられる。

5. 各ヒューリスティックスの比較実験

5.1 実験条件

タイリングの効果を確認するために対象とするメニーコア型 CPU として、Intel Xeon Phi (Knight Landing) Pro-

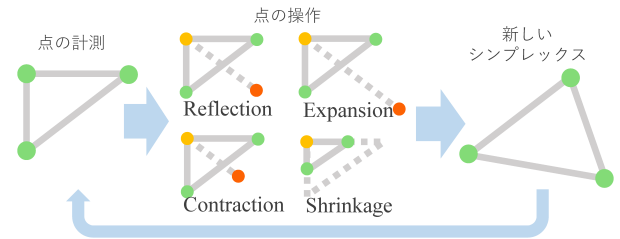


図 6: NM 探索イメージ

cessor 7210 を使用した。スレッド数は 64 で実行した。評価用ベンチマークとしては、Polybench[7] を採用した。これは Polyhedral コンパイラと相性の良いベンチマークスイートとして知られている。今回はその中から 3 重ループの代表として gemm カーネルを、2 重ループの代表として atax カーネルをそれぞれ採用した。問題サイズは Polybench の LARGE サイズを使用した。Polly の最適化処理としては、自動タイリング、ベクトル化、パラレル化の全てを有効化している。

バイナリのビルドを Xeon Phi 上でそのまま実行すると 1 コアあたりの周波数が低いため時間がかかり、全体の実験時間が長くなることが分かっている。そのため、バイナリのビルドのみ高速な Xeon マシン上で行い、バイナリの実行はリモートの Xeon Phi 上で行うとした。

S-PATT の設定として、1 次元辺りの領域分割数を 8 とした。SA の設定として、近傍を各次元距離 1 と定め、その中から一様ランダムに点を選択するとした。さらにスタート地点として、Polly のデフォルトタイルサイズである 32 を採用した。NM の設定として、Reflection の係数を 1.0、Expansion の係数を 2.0、Contraction の係数を 0.5、Shrinkage の係数を 0.5 とした。これは NM においてあらゆるケースで良好に動作すると広く知られている一般的なパラメータである。また、初期シンプレックスとして各次元 32 という点を中心として、各次元が直交するように距離 4 の点を設定した。これは予備実験において、この設定が他に比べて比較的良好な結果を示したためである。

5.2 結果考察

図 7 と図 8 に各ヒューリスティックスの収束スピードの結果を示す。SA に関しては探索手法にランダム要素を含み実行ごとに結果が変化するため、そのうち最終的に到達した性能が最も良かったものを SA Best、最も悪かったものを SA Worst とした。

2 つのカーネルとも、S-PATT の収束スピードが他と比べて速いことが見て取れる。これは S-PATT が最初の段階でロードバランスを調整することと、広く粗い探索において最速付近の領域を素早く検出しているためである。しかしながら gemm の場合は最外ループタイルサイズの決定に時間がかかり、NM のダイナミックな動きによる収束ス

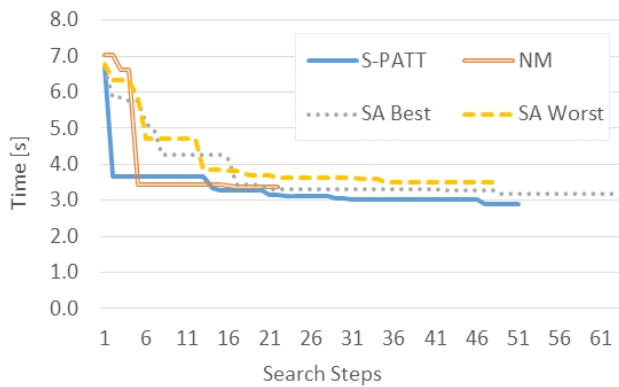


図 7: 各ヒューリスティックスの収束スピード (gemm)

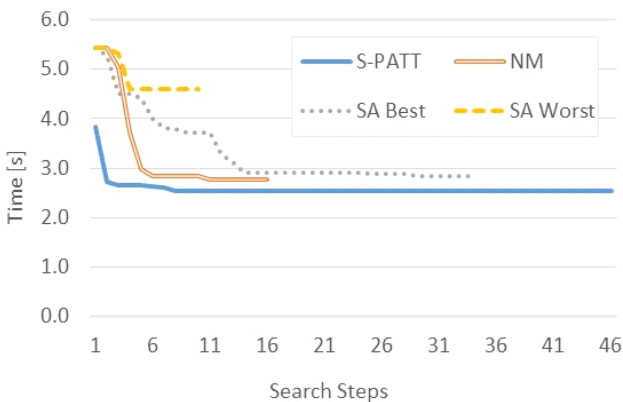


図 8: 各ヒューリスティックスの収束スピード (atax)

表 1: 各ヒューリスティックスの最終結果

	Search Method	Tile Sizes	Time	%
gemm	Brute-Force	16, 1124, 12	2.89	100.0
	S-PATT	16, 1116, 12	2.91	99.3
	NM	16, 36, 48	3.36	86.0
	SA (worst)	16, 28, 44	3.50	82.6
	SA (best)	16, 24, 80	3.19	90.6
atax	Brute-Force	8, 1940	2.50	100.0
	S-PATT	8, 2096	2.53	98.8
	NM	8, 80	2.78	89.9
	SA (worst)	36, 36	4.60	54.3
	SA (best)	8, 64	2.84	88.0

ピードに少し遅れを取っている部分があることが分かる。両方に共通して、NM は S-PATT に次いで比較的良好に動作し、SA に関してはランダムの結果次第で収束スピードに差が出ている。SA は最速点が初期点から離れている場合、その距離を進むのに時間がかかってしまう。そのため 2 つのカーネルとも SA は収束スピードとしては遅くなっている。

表 1 に各ヒューリスティックスが最終的に出力したタイルサイズとその速度をまとめた。Brute-Force は実質的に総当たりで全てのタイルサイズの組み合わせを実験した中

での最速という意味である。2 つのカーネルとも、S-PATT は Brute-Force に対して約 99% の性能のタイルサイズを見付け出すことに成功している。実際、出力されたタイルサイズも Brute-Force のそれと近く、距離的に近くに位置していることが分かる。これに対し、SA、NM は最大でも約 90% 程度に留まるという結果になった。この原因としては、まず各次元の重要度をフラットに扱ってしまっていることが挙げられる。S-PATT ではロードバランスに注目するため最外ループタイルサイズを特別視したが、SA や NM はこのような扱いをしていない。つまり、性能に繊細な最外ループタイルサイズがある程度ダイナミックに動いてしまう。また、内側ループタイルサイズの探索としては平地が広く超えられず、端まで移動することができなかったということも理由として挙げられる。gemm も atax も 2 番目のループのタイルサイズは大きい値、つまり問題サイズの端近くに最速があるということが Brute-Force の結果から分かる。それに対し SA も NM も小さいタイルサイズの部分で止まってしまっている。これは、SA や NM の移動幅では広い平地を超えられず、その結果速度向上が見込めないとしてそこで探索を終了してしまっているということである。このことから、S-PATT の広く粗い探索の重要性が見て取れる。

6. 関連研究

Mehta らは、キャッシュサイズなどのハードウェア情報及び対象カーネルの情報を利用して静的なタイルサイズ導出を目指す TurboTiling[8] という手法を研究している。これはメニーコア環境を想定していないため、活用するためにはメニーコア環境という前提での手法の見直しが必要となる。彼らの研究と比較すると、我々のアプローチは自動チューニングをベースとしているため、様々な環境で幅広い活用が可能となっている。

白子らは、タイルサイズの探索空間を解析的モデルによって制限するという手法を提案している [9]。キャッシュヒットの上限と下限をモデリングした DL/ML モデルを利用して、探索空間の絞り込みを行うというものである。彼等の手法はシングルスレッド環境をベースとしており、メニーコア環境で問題となるロードバランスに関しては言及していない。

Hartono らは、Orio というアノテーションベースの自動チューニングシステムツールを開発している [10]。調整したいパラメータをアノテーションとしてユーザーが設定することで、タイルサイズだけでなく、ループアンローリングパラメータやパラレル化やベクトル化の有無なども自動チューニングすることができる。各設定値で実際にバイナリを作成して実行し、それをユーザーが指定した探索方法を使って繰り返し、最終的に性能の高かった設定を出力する。この時の探索方法として、SA や NM が実装されてい

る。しかしながら現状では、アノテーションで実現する自動タイリングの際に正しいタイリングコードが生成されないという問題点がある。

7. まとめと今後の課題

本報告では、タイリングとそのタイルサイズの調整の重要性について述べた後、我々が開発しているタイルサイズ自動調整ツール PATT の説明を行った。メニーコア環境におけるロードバランスの問題やタイルサイズ調整時に発生する問題とその対処法について言及し、タイルサイズの探索手法として、その探索に特化したヒューリスティックスである S-PATT を開発した。また、メタヒューリスティックスとして SA, NM の 2 つを実装し、これらヒューリスティックスの比較実験を行った。その結果、我々の S-PATT が探索の収束スピードと最終出力の性能の両面において、他の手法を上回ることを確認した。

今後の S-PATT の改善としては、SA のような確率的遷移のメカニズムを導入し極値があるようなカーネルにも広く対応できるようにすることや、2 重ループカーネルの場合にもロードバランスの影響を考慮しその後の探索に生かす、などの方向性が挙げられる。しかしながら既に現状の S-PATT を使うことで、時間のかかる Brute-Force の実験を行わずとも実際に最速に近いタイルサイズが短時間で取得可能であるという状況がある。これを生かし、幅広いカーネルや様々な環境においての最速タイルサイズの知見を深めていき、理論的アプローチによるタイルサイズ静的導出を最終的に狙うことができれば、活用の幅もより一層広くなると考えている。そのためにはメニーコア環境に向けた TurboTiling の見直しや、様々な環境でのタイルサイズの実験が必要だと思われる。

謝辞

本研究は、JST、CREST の支援を受けたものである。

参考文献

- [1] 幸 朋矢, 佐藤幸紀, 遠藤敏夫: Polyhedral コンパイラを用いたタイリングパラメータ自動調整ツールのメニーコア環境での評価, 研究報告ハイパフォーマンスコンピューティング (HPC), Vol. 2017-HPC-160, No. 34, pp. 1–8 (2017).
- [2] Grosser, T., Groesslinger, A. and Lengauer, C.: Polly - Performing polyhedral optimizations on a low-level intermediate representation, *Parallel Processing Letters*, Vol. 22, No. 04, pp. 1–28 (2012).
- [3] Bondhugula, U., Hartono, A., Ramanujam, J. and Sadayappan, P.: A Practical Automatic Polyhedral Parallelizer and Locality Optimizer, *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pp. 101–113 (2008).
- [4] Lattner, C. and Adve, V.: LLVM: A Compilation Frame-

- work for Lifelong Program Analysis & Transformation, *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pp. 75–88 (2004).
- [5] Russell, S. J. and Norvig, P.: *Artificial Intelligence: A Modern Approach*, Pearson Education, 3 edition (2009).
 - [6] Norris, B.: Orio, <http://brnorrison03.github.io/Orio/> (2008).
 - [7] Yuki, T. and Pouchet, L.-N.: PolyBench, <https://sourceforge.net/projects/polybench> (2016).
 - [8] Mehta, S., Garg, R., Trivedi, N. and Yew, P.-C.: TurboTiling: Leveraging Prefetching to Boost Performance of Tiled Codes, *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pp. 38:1–38:12 (2016).
 - [9] Shirako, J., Sharma, K., Fauzia, N., Pouchet, L.-N., Ramanujam, J., Sadayappan, P. and Sarkar, V.: Analytical Bounds for Optimal Tile Size Selection, *Proceedings of the 21st International Conference on Compiler Construction*, CC'12, pp. 101–121 (2012).
 - [10] Hartono, A., Norris, B. and Sadayappan, P.: Annotation-based empirical performance tuning using Orio, *2009 IEEE International Symposium on Parallel Distributed Processing*, pp. 1–11 (2009).