Vol.2018-ARC-231 No.1
2018/6/14

# 動的バイナリ変換を用いた透過的コード最適化における タイルサイズ選択の評価

佐藤 幸紀[1,a)]

**概要**：コンパイル済みのコードを実行時に最適化されたコードに変換する動的バイナリ最適化機構においてループタイリングに代表されるループ変換を適応することは、階層化されたメモリサブシステムに向けてのデータ局所性に可制御性を与えるため、高いコード実行性能を目指す上で非常に重要である。本報告では、オープンソースのコンパイラ基盤である LLVM の上で Polyhedral モデルに基づく高位最適化を実行時に透過的かつ自動で行う機構を用いて、タイルサイズをターゲットの環境に合わせて調整することで得られる性能利得の調査を行う。

## 1. はじめに

半導体微細化技術、単一 CPU 向けの高度なアーキテクチャ技術、そして、マルチコア CPU による並列処理技術の向上によって、スーパーコンピュータからモバイル機器・組み込み機器に至るまであらゆる計算機システムは指数関数的に性能を進化させてきた。しかしながら、微細加工技術の物理的限界により、ムーアの法則により駆動されてきた半導体集積度の向上が近い未来に終焉を迎えるであろうとの認識は広く共有されつつある。このような背景の下、半導体微細化技術の進化に由来する性能向上を補うためのアーキテクチャ技術や並列処理技術の開発に益々の期待が高まっている。

自動運転や知的ロボットなどの新しい応用の現場からも、高性能で高効率な計算機システムを実現する技術は、それらの応用を展開する上で鍵（Enabler）となる技術として高い期待を集めている。近年の人工知能技術の中核をなす深層学習処理においても現状の CPU/GPU 技術だけでは性能がまだまだ足りないといわれており、今後期待される人工知能社会を実現する上では計算機システムの性能不足問題とその生産的な解決法の確立は深刻な課題となっている。

計算機システムの設計においては、性能と電力は相反するトレードオフの関係があり、単に処理速度にて計測される性能を向上させるということに限定するのではなく、性能電力比で考えた実行効率の改善を目指す必要がある。近年の計算機システムにおいては、処理を消費電力および速度の面で効率化するために、汎用 CPU に加えて GPU に見られるような SIMT 処理機構、512 ビット幅を超えるベクトル演算器といった計算コア、ドメイン特化型のアクセラレータなどを混載することによる処理要素コアのヘテロ化、更に 3 次元積層技術を用いた多様なメモリデバイスの利用するというアプローチが急速に普及している。このようなシステム側の進化に合わせてメモリ階層は複雑化の一途をたどり続けている。複雑に階層化されたメモリサブシステムを持つアーキテクチャにおいて高い性能効率を得るためには、既存のキャッシュメモリや一般的なコンパイラでは考慮されないメモリ階層間の特性の違いを意識したメモリ参照局所性を活用する必要がある。すなわち、来るべき時代の計算機システムにおいて性能や消費電力の面での高い処理効率を達成するためには、アプリケーションに内在する多様なパターンのメモリアクセスを複雑なメモリ階層向けにチューニングしていくことが鍵となると予測されている。

一方で、アプリケーションプログラムの構造は高機能化や高精度化に駆動される形で複雑化の一途をたどり、そのコードの量も年々増加の傾向にある。性能効率への要求が厳しい HPC 分野においては、コードチューニングは熟練の HPC 技術者がプログラムのソースコードにアクセスし、手動でプログラムの書き換えをすることにより行われてきた。しかしながら、年々多様化が進み増加の一途をたどるアプリケーションプログラムに対して、その都度ソースコードレベルでのチューニングを行うことは非常に労力がかかる作業であり、自動化や効率化が求められている。

このような背景において、性能不足の問題を解決する糸口はソフトウェアのメモリ参照局所性を的確にハードウェ

[1] 豊橋技術科学大学 大学院工学研究科 情報・知能工学系
[a)] yukinori@cs.tut.ac.jp

© 2018 Information Processing Society of Japan

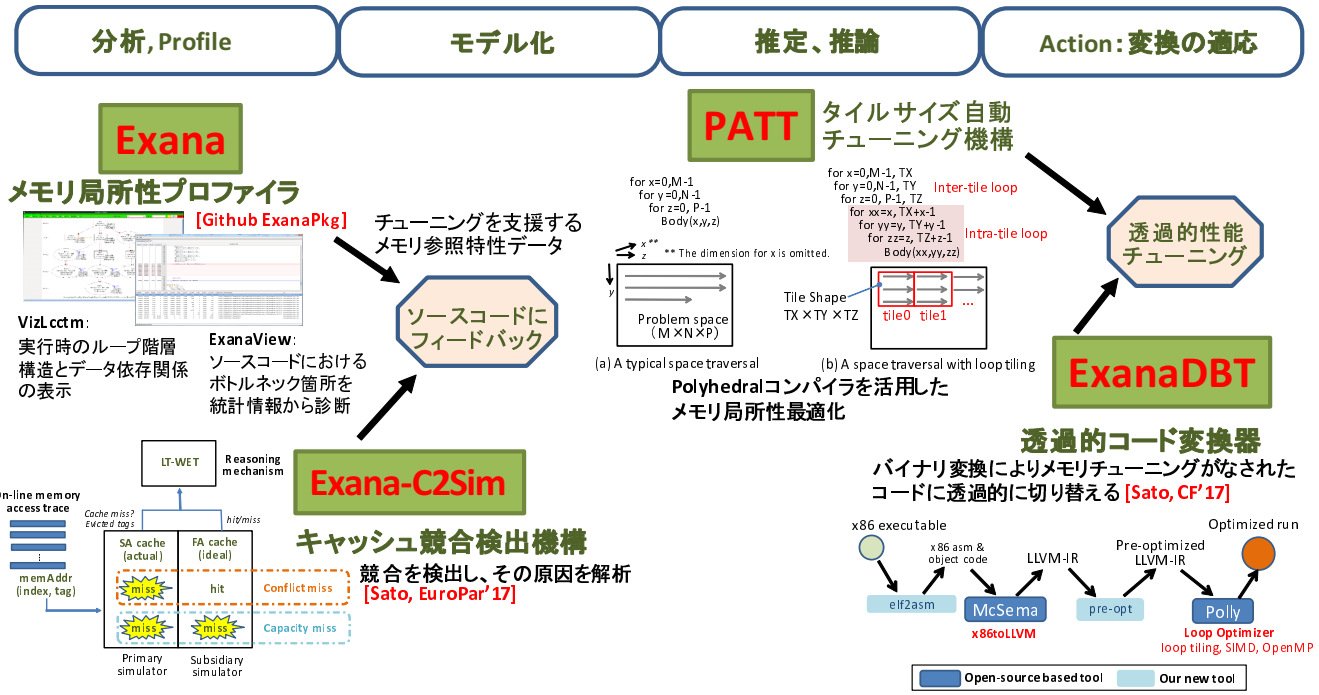


図 1: 透過的メモリ局所性チューニングのためのツールチェイン

アの特性にカスタマイズすることとの着想の下、我々のグループではメモリシステムを中心としたカスタム化やコデザインを行うためのコード変換技術や最適化技術に関する研究を進めてきた。

HPC 分野のアプリケーションコードに高度なチューニングを施すことを支援しチューニングの生産性向上を劇的に向上させることを目的として、メモリ階層性能プロファイラーを核とする Exana ツールセット [1], [2], [3] や、メモリ局所性チューニングをコード実行時に動的に行う ExanaDBT[4] を開発してきた。これらは、動的バイナリ変換（Dynamic Binary Translation; DBT）技術を利用してコンパイルおよびリンク済みの実行バイナリコードを入力としたメモリ参照局所性のプロファイリングやコード変換を行う*1。

本報告では、コンパイル済みのコードを実行時に最適化されたコードに変換する動的バイナリ最適化機構である ExanaDBT において、ループタイリングによりメモリ階層や異種メモリのパラメータの相違をアプリケーションのデータ参照局所性に適応的にマッピングすることを透過的に行えることを示すと同時に、タイルサイズをターゲット CPU の特性に合わせて調整することで得られる性能利得の調査を行う。ループタイリングを実際に動作させる実行コードに適応することは、階層化されたメモリサブシステムに向けてのデータ局所性に対して可制御性を与えるため、高いコード実行性能を目指す上で非常に重要である。そこで、オープンソースのコンパイラ基盤である LLVM の上で Polyhedral モデルに基づくループ部分のタイリングを行いながらその最良なタイルサイズを反復的に探索する PATT というタイルサイズ自動チューニング機構 [5], [6] を用いて事前に最良となるであろうタイルサイズを探索し、そのパラメータを ExanaDBT に与えることにより、実行時にタイルサイズをターゲットとなるプログラムや CPU の環境に合わせて調整することで得られる性能利得を調べる。

## 2. 透過的メモリ局所性チューニングのためのツール群

図 1 に本報告の評価の基盤となる透過的メモリ局所性チューニングのためのツールチェインを示す。チューニングの流れとしては、性能の分析・プロファイリング、モデル化、性能の推定・推論、変換の適応という流れに従って進められる。

性能の分析、プロファイリングをおこなうことをターゲットとする Exana は、多様なチューニングを支援する統計データを生成する一方で、最終的にその結果はプログラマが手作業でソースコードにフィードバックすることが求められている。一方で、ExanaDBT はプログラマから透過的に自動でコード最適化を行うフレームワークであるが、現状においては最適化のシナリオは必ずしもプロファイリングや実環境の性能推定に基づいて適応的に選択されるという実装に至っていない。PATT におけるタイルサイズ自動チューニング機構は、実環境の性能推定を行うため、性能の推定・推論とコード変換を強く結びつける上で有用となるツールとなることが期待される。そこで、これらのツール群について、本節で説明した後、それらを融合する

*1 Exana に関しては Github を通してそのソースコードが公開されている。https://github.com/YukinoriSato/ExanaPkg

© 2018 Information Processing Society of Japan

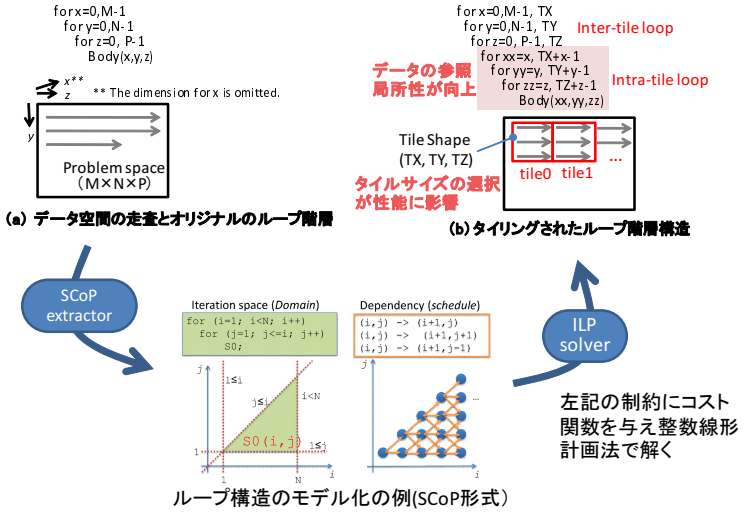


図 2: ループタイリングの概要と Polyhedral モデル

プロトタイプ環境を構築し、評価を行う。

### 2.1 Exana ツールによるメモリ階層チューニング

Exana ツールは、コード実行時に透過的に解析コードを挿入することによりキャッシュメモリに代表されるような多階層構造メモリへのアクセス局所性傾向やその性能特性を抽出する機能を提供する。すなわち、得られた性能特性に関する統計情報を熟練した HPC プログラマが深化するメモリ階層向けのメモリチューニングに活用することに主眼を置き開発を進めてきた。例えば、Exana-C2Sim[2] においては、オンラインで取得したメモリアクセストレースをオンラインでキャッシュシミュレータにと投入し、詳細なキャッシュメモリの挙動の模倣した環境においてキャッシュライン競合の発生をモデル化し、特にキャッシュライン競合に着目したプロファイラとしてツールのパッケージ化を行った。結果として、ユーザーがプロファイリング結果を踏まえて経験的に判断している最適化シナリオの作成を支援、あるいは半自動することはできるが、見込まれる性能利得の推定や具体的な修正のアクションが必要なコードの変換については最終的にはプログラマの手作業が介在することが必須となっている状況であった。すなわち、このような性能利得の推定やコードの変換に関しては経験に基づく職人芸的な技術を要することが多かったため、チューニング技術をコモディティ化するという観点からも完全な自動化を行う技術開発が求められている。

### 2.2 メモリチューニングと Polyhedral コンパイレーション技術

メモリ参照における空間的局所性や時間的局所性を活用するための最も一般的な手法は、ハードウェアで実装されるキャッシュメモリを用いることである。しかしながら、キャッシュメモリによる局所性管理は万能ではなく、アプリケーションに備わる参照局所性が十分に効果的に活用しきれていない場合があることが知られている。このキャッシュの弱点を補うために、プログラム側のループ部分をキャッシュにヒットしやすいように変換するループタイリングを中心としたループ変換が HPC 分野ではメモリ局所性チューニングの手法として広く用いられてきた。

**図 2** にループタイリングの概要と Polyhedral モデルによる最適化における流れとの関係を示す。ループタイリングはターゲットとするマシンのメモリ階層やアプリケーションのデータ再利用性に合わせるためにアクセスをタイル単位で進め、アプリケーションのデータのワーキングセットの時間変化量を調整するための変換である。図 2(a) において 3 次元のデータを操作する 3 重ループのオリジナルコードと、単純化のために x 軸の走査を除いた 2 次元の解析空間の図を示す。2 次元の空間を走査する場合は、プログラムにおいては 2 重ループとして記述され、z 軸方向への連続するアクセスがメモリ上の連続アクセスとなると想定すると、z 軸方向の空間的な参照局所性は十分活用される。一方で、各次元の近傍点へのアクセスを考えた場合、y 軸方向の参照局所性はキャッシュ容量の制約のため活用できないことが多い。そこで、実際のメモリ階層構造に合わせて再利用を最大化するようにループネスト構造の再編成とタイルサイズと形状を選択することによりターゲットとなるコードの局所性を最適化する。

左記の例の 2 次元の空間走査にタイリングを適応すると、2 重ループによる空間走査から、タイルの選択も含めた 4 重ループにより空間にアクセスすることとなる。すなわち、オリジナルの 3 重ループにおいては、図 2(b) のように 6 重ループに変換される。タイル内のアクセスにおいては各方向の空間的および時間的な参照局所性が活用でき、参照局所性が高まる。

近年、伝統的にプログラマが手作業で行っていたループ

© 2018 Information Processing Society of Japan

タイリングに代表されるループ変換を自動化するアプローチとして、Polyhedral モデルに基づく最適化を行うフレームワークが注目を集めている [7], [8]。Polyhedral モデルに基づく最適化器が実装されたコンパイラにおいては、入力となるソースコードのループ構造をモデル化し、ループ反復レベルで計算順序のアルゴリズム的な変更を行い、メモリアクセス順序を再編することによりメモリ参照局所性を改善させる。ループ誘導変数にて制御されるループ反復における相互のデータ依存関係を満たすために、ループ内部でステートメント単位で配列を演算していく順序を満たすスケジューリングを探索し、コストモデルに基づき条件に合う最も良い解を導出する。

図 2 の例においては、オリジナルのソースコードにおけるループ階層は SCoP 表現と呼ばれる形式に変換され、反復空間やステートメントごとのスケジューリング制約と共に定式的にモデル化される。この定式化された制約にコスト関数を与え整数計画法のソルバを用いて、最も適するループ変換を導出し、その変換を適応したコードを生成する。従来、ループにおいて依存関係を確認する作業はプログラマがソースコードを丹念に読みプログラムの意味を理解する必要があり生産性の面で問題となっていたが、Polyhedral モデルに基づく自動化により、これまで一部の HPC 分野のコードに限られていたタイリングによる最適化が幅広い分野のアプリケーションに展開可能となることが期待されている。

本報告では、Polly を用いてオープンソースのコンパイラインフラストラクチャである LLVM のレイヤで Polyhedral モデルに基づき最適化を行う Polly というツール [9] を用いて主にキャッシュメモリをターゲットとしたタイリングについて評価を行う環境を構築した。本報告では取り扱わないが、ループタイリングは、ストレージ、DRAM メモリ、キャッシュ、レジスタなど様々な異なる階層について複数階層をターゲットに適応可能 [10] であり、今後登場が期待される様々な NVRAM によりますます深化・複雑化するメモリサブシステムの局所性を制御する有効な手段と考えられる。

### 2.3 ExanaDBT

ExanaDBT は、コンパイル済みの実行形式のバイナリを入力として実行時に Polyhedral モデルに基づきループ変換や並列化をプログラマから透過的に行うダイナミックコンパイレーションシステムである [4]。ExanaDBT は実行のホットスポット検出から始まり、最適化の利得を動的に見積もり、最適化の対象とする領域に対して Polyhedral モデルに基づくループ最適化を施した後、オリジナルのコードの実行を最適化されたものに透過的に切り替える。トレースや命令を超えるレベルでの高度なループレベルの最適化を実現するために、ExanaDBT は深いメモリ階層を持つシステムの上で持続的にスケーラブルに性能向上を達成できるように Polyhedral モデルに基づきループ変換を実施しコードを最適化する。

**図 3** に ExanaDBT における実行バイナリ変換に基づくコード変換機構の構成を示す。実行形式のバイナリを入力として実行する過程において、P-E-T-S から構成される 4 つのステージからなる変換機構から構成される。P（Profile）ステージは実行時プロファイリングを行うステージである。P ステージにおいては、Exana ツールとして実装してきた各種プロファイラを活用して性能チューニングに利用する統計情報を実行時に抽出することが可能である。E（Estimate）ステージは P ステージにて得られたプロファイリング情報を入力としてメモリ階層性能モデルを用いて性能利得のある最適化の対象領域や最適化の戦略を出力する。今回の実装においては、P ステージにて HotSpot 検出を行い、E ステージにおいては HotSpot として検出されたところを常に最適化するという戦略とした。

実際にバイナリコードを最適化する T（Translate）ステージについては、E ステージにて得られたチューニングによる性能利得があると判定されたカーネル部分をバイナリコード中から抜き出し、逆アセンブルした後、LLVM の IR に変換し、IR レベルで Polyhedral model に基づくループ変換を行い、最適化されたバイナリを生成するという機構の実装を行った。本実装においては既存のコンパイラインフラストラクチャの LLVM、Polly や McSema という x86 のバイナリコードを LLVM-IR に変換するツール [11] といったオープンソースのツール群を活用し開発を進めた。本フローにおいて目的とする最適化を成功させるためにはループを最適化が可能な品質の Polyhedral モデルで記述する必要があったが、単純に McSema を用いてオリジナルのバイナリコードを LLVM IR に復元するだけでは不十分であったので、Polyhedral 最適化が可能な IR に記述形式を引き上げる変換器 (pre-opt) が独自のツールとして実装されている。

本機構の最適化を担う Polly は LLVM-IR のレベルで Polyhedral モデルに基づく最適化によりループタイリング、ベクトル化、並列化を行うことができるので、ExanaDBT により最適化を行っていない逐次の実行コードが、ループタイリング、ベクトル化、スレッド並列化が実施されたコードに透過的に変換され、これらの効果を得ることが可能となる。なお、オリジナルの Polly でのコード変換の過程においては、タイルサイズが全ての次元で 32 に固定されたループタイリングが実行されるため、ExanaDBT においても 32x32x32 以外のタイルサイズを環境に合わせて適応することは評価されてこなかった。

最適化されたコードが T ステージにて生成された後、S（Switch）ステージにおいては、コード実行をオリジナルのものから T ステージで変換したものに切り替えることを行

© 2018 Information Processing Society of Japan

う。本機構もバイナリ変換技術を利用して実装を行った。

### 2.4 PATT

PATT (Polyhedral compilation based AuTo Tile size optimizer) は、性能チューニングにおいてループタイリングが局所性を適応する上で重要な役割を果たしていることに着目し、その特性を決定づけるタイルサイズの最良なパラメータを見つけ出すことを行う [6], [12]。最適化を施していないソースコードを入力として、Polyhedral コンパイラを用いてコンパイル時に自動的にタイリングを実施し、チューニングを行ったコードを生成する。膨大なタイリングのパラメータ空間から最良なタイルサイズを探索するために、PATT においては Hill-climbing 法をベースとするタイルサイズ選択に特化した最適化アルゴリズムが実装されている。現状の Polly の実装においては、変数としてタイリングパラメータを指定することができないので、コンパイル時にタイルサイズのパラメータをその都度指定し、対応する実行コードを生成したのち、実環境における実行時間の計測を行い、探索を進める。

## 3. ExanaDBT におけるタイルサイズ選択

アプリケーションをソフトウェアの面からチューニングしていくことは非常に重要な反面、アプリケーション分野の経験的な知識とシステム構成に固有のチューニングに関するノウハウや、メモリ階層の特性やヘテロな処理要素の構成などシステム構成に関する深い知識が必要であり、それらに対して適応的にカスタマイズしていくことを考えると、コード開発とチューニングにおける NRE コスト（Non-Recurring Engineering cost）の増大や、増え続けるソフトウェア量やチューニング対象に対する持続性が課題となっている。また、書き換えられたプログラムはシステム構成に依存することが一般的で、その移植性にも問題がある。

ExanaDBT は、動的バイナリ変換 (DBT) に駆動される実行時の変換を主体とする柔軟かつ持続的な性能チューニングシステムを実現可能である一方で、先行論文 [4] においては、タイルサイズを固定して与えているなど、この柔軟性が十分に活用されていなかった。

そこで、本報告では ExanaDBT において、メモリ階層や異種メモリのパラメータの相違を反映したループタイリングを実施するプロトタイプを実装し、アプリケーションのデータ参照局所性に適応的にマッピングすることを透過的に行うことを試みる。図 3 に ExanaDBT において最良となるであろうタイルサイズを入力する手法を示す。特に、今回は評価の初期段階として、タイルサイズを調整する利得を調査することを主な目的として、事前に対象環境において PATT を用いてタイルサイズ自動チューニングを行うものとした。ExanaDBT の設計思想を考えると、P ステージ及び E ステージにてタイルサイズ選択を行うことが理想であるが、オンラインでの現実的な探索にかけられる時間は限られているため、探索法を含めて課題が多い状況にある。そこで、事前に探索時間の面の制約をなくしたパラメータを用意した理想的な状況において、得られる利得の期待値を調べる。

## 4. 評価

### 4.1 実験環境

本評価においては 2 基の Intel Xeon E5-2650v2 CPU と 64GB の DDR3 1866MHz DRAM からなる主記憶メモリを備え、CentOS 7.2 が OS としてインストールされている 64bit の x86 のマシンを用いた。本マシンにおいては、16 の物理コアが利用可能であるため、並列実行の評価においては 16 スレッドを'taskset -c 0-15' によりコアにバインドし計測を行った。

ベンチマークプログラムとして Polybench4.2 を用いた。特に、先行研究 [4] において ExanaDBT で動的チューニングに効果が見られた 5 つのプログラム（2mm, 3mm, gemm, syr2k, syrk）を用いた。データセットとしては LARGE を用いて、配列のデータ型として double を使用した。

Exana-DBT を実装するための DBT システムのインフラとしては Pin tool set[13] を用いた。ExanaDBT の入力として、clang -O0 にてコンパイルして生成した並列化および最適化が施されていない実行バイナリを用いた。P ステージにおける HotSpot プロファイリングのパラメータとしては 50M 命令毎にトレースのサンプリングを行い、トレースの出現が 30 を超えたものをホットスポットとして検出するとした。ホットスポットとしてカーネルが検出された後に改めてカーネル部分の関数が呼び出された際に最適化を行う実装であるため、最適化のためにはカーネル部分が複数回実行される必要がある。今回はカーネル部分を 100 回反復するようにソースコードを拡張している。

PATT におけるタイル化されたコードの評価においては、計測に用いる実行コードは ExanaDBT が動的に生成するものではなく、ソースコードからコンパイルするものを利用した。その他の詳細に関しては、先行研究 [6], [12] で用いたものと同様である。

### 4.2 実験結果

**表 1** に PATT によるタイルサイズ探索を行った際に得られた速度向上比と、選択されたタイルサイズを示す。最も利得が大きかった syrk において、デフォルトのタイルサイズ（32x32x32）の場合と比べて 1.41 倍の速度向上が確認された。なお、本評価においては 16 コアのマシンを使ったため、先行研究 [6] における 64 コアのマシンを対象とした PATT の利得よりも速度向上が小さい。

**表 2** に、ExanaDBT への入力とした逐次コード (clang

© 2018 Information Processing Society of Japan

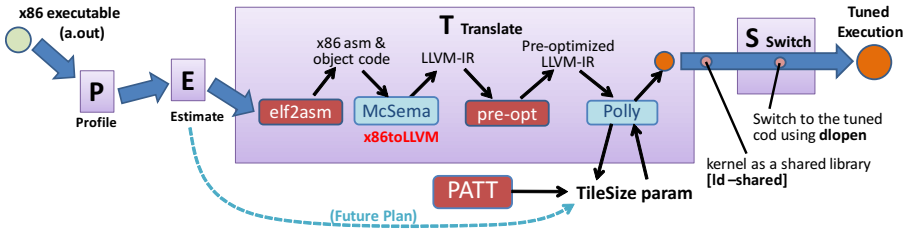


図 3: ExanaDBT への最良のタイルサイズの入力

表 1: PATT によるタイルサイズ選択の効果

| | 2mm | 3mm | gemm | syr2k | syrk |
|---|---|---|---|---|---|
| PATT speedup | 1.18 | 1.11 | 1.04 | 1.37 | 1.41 |
| Tile size (i_patt ) | 28,289,8 | 20,326,8 | 16,97,25 | 4,9,1018 | 4, 853, 1027 |

-O0) の実行時間をベースとした ExanaDBT の速度向上比を示す。結果より、ExanaDBT を用いることで、入力とした最適化を施していない逐次コードがプログラマから透過的にタイル化、ベクトル化、並列化され、最大 17.8 倍の処理速度向上が達成できることが分かる。また、先行研究における実装の ExanaDBT.16 と最良なタイルサイズを適応した場合の ExanaDBT(PattTile).16 を比較することにより、タイルサイズを対象のアプリや CPU の特性に合わせて調整することで得られる性能利得を推測することができる。5 つのプログラムを通して、最良なタイルサイズによりタイリングしたほうが性能が向上していることが見受けられるが、両者を比較すると syrk において 7%向上したのが最大であり、PATT においてソースコードから実行コードを生成した場合よりも利得が減少している。

この利得の減少の原因としては、静的にソースコードから生成するコードと動的生成のコードの品質の違いに由来してキャッシュの挙動が異なっているためと考えられる。動的生成のコードはその生成過程において多数の変換が行われてきたこともあり、静的生成版には見られない冗長な命令実行も数多くみられる。このため、メモリレイアウト、メモリアロケーションの挙動が変化しキャッシュのヒット率の低下等が起こっていると考えらえる。また、動的バイナリ変換のためのコードとのキャッシュリソースの競合も性能低下の要因として考えられる。

今後、より精度の高い利得の推定を行うために、動的生成のコードに対して、PATT によるタイルサイズ自動チューニングを適応するインターフェースを確立し、評価を行っていく計画である。また、動的生成されるコード実行において観測された実行時間には、若干のばらつきもあり、動的生成されるコードの品質の改善も含めて、性能ばらつきに対する解析を今後詳細に行う計画である。

### 4.3 関連研究

DBT を用いてタイリングを含むループ変換を動的に行うというアプローチに米国ミシガン大学の研究チームの提案する ShapeShifter が挙げられる [14]。ShapeShifter は、Protean code という DBT システムと LLVM Polly を組み合わせ、ランタイムにタイリングパラメータを最適化したコードを生成するシステムであり、クラウド環境のようなリソースを高度に共有するダイナミックなマルチコア・マルチプロセッサの環境を対象に、同時に走るプロセスやバックグラウンドジョブの影響を踏まえキャッシュタイリングによりメモリフットプリントを実環境に合わせて適応していく。本研究で対象としている OpenMP でスレッド化されているコードにおける高速化を主な目的としたタイリングと、彼らの想定しているマルチプログラミング環境のような複数の独立したプログラム（タスク）を同時に走らせた際のメモリリソースの最適な配分のためのタイリングではその目的が大きく異なる。また、ShapeShifter はソースコードを起点に LLVM の中間言語である LLVM IR を生成することを前提としたアプローチであり、我々が目指している任意の実行バイナリを起点としてループ最適化を行っていくアプローチとは異なる。

Damschen らは DBT を用いてバイナリコード中のホットスポットをメニーコアプロセッサにオフロードする機構を提案している [15]。彼らの手法においても LLVM-Polly を用いて最適化が行われているが、想定しているバイナリコードが LLVM IR の.bc であり、x86 のコードを起点に最適化を行っていく我々のアプローチとは異なる。

## 5. まとめ

本報告では、Polyhedral モデルに基づく高位最適化を透過的かつ自動で行う ExanaDBT においてタイルサイズをターゲットの環境に合わせて調整することで得られる性能利得の調査を行った。ソースコードを反復的コンパイレーションを行いタイルサイズを自動チューニングする PATT により事前に対象の環境で最良となるタイルサイズを探索し、ExanaDBT の動的コード生成時のパラメータとしてタ

© 2018 Information Processing Society of Japan

表 2: 入力した逐次コード (clang -O0) と比較した ExanaDBT の速度向上比とタイルサイズ入力の効果

| | 2mm | 3mm | gemm | syr2k | syrk |
|---|---|---|---|---|---|
| ExanaDBT.16 | 15.0 | 15.9 | 11.6 | 17.5 | 12.4 |
| ExanaDBT(PattTile).16 | 15.1 | 16.1 | 11.8 | 17.8 | 13.2 |

イルサイズの指定を行った結果、処理系のデフォルトのタイルサイズを適応する場合と比べて若干ではあるが性能向上があることを確認した。

今後の課題としては、動的生成のコードに対して PATT によるタイルサイズ自動チューニングを適応するインターフェースを確立し評価を行っていくことや、動的生成されるコード実行における性能ばらつきに対する解析を行っていく計画である。

## 謝辞

本研究は JSPS 科研費 JP17K12658 の助成を受けたものです。

### 参考文献

[1] Sato, Y., Sato, S. and Endo, T.: Exana: An Execution-driven Application Analysis Tool for Assisting Productive Performance Tuning, *Proceedings of the 2nd International Workshop on Software Engineering for Parallel Systems*, SEPS 2015, pp. 1–10 (2015).

[2] Sato, Y. and Endo, T.: An Accurate Simulator of Cache-line Conflicts to Exploit the Underlying Cache Performance, *In Proceedings of 23rd International European Conference on Parallel and Distributed Computing (Euro-par 2017)*, pp. 119–133 (2017).

[3] Exana tool kit: http://www.perf.cs.tut.ac.jp/ yukinori/Exana.html (2018).

[4] Sato, Y., Yuki, T. and Endo, T.: ExanaDBT: A Dynamic Compilation System for Transparent Polyhedral Optimizations at Runtime, *ACM International Conference on Computing Frontiers 2017 (CF'17)*, p. 10 pages (2017).

[5] Yuki, T., Sato, Y. and Endo, T.: Evaluating Autotuning Heuristics for Loop Tiling, *International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018)*, p. 2 pages (2018).

[6] 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，並列/分散/協調処理に関するサマーワークショップ (SWoPP2017), 研究報告ハイパフォーマンスコンピューティング（HPC）, 2017-HPC-160，pp. pp.1–8 (2017).

[7] Benabderrahmane, M.-W., Pouchet, L.-N., Cohen, A. and Bastoul, C.: The Polyhedral Model is More Widely Applicable Than You Think, *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, CC'10/ETAPS'10, pp. 283–303 (2010).

[8] Pouchet, L.-N., Bondhugula, U., Bastoul, C., Cohen, A., Ramanujam, J., Sadayappan, P. and Vasilache, N.: Loop Transformations: Convexity, Pruning and Optimization, *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pp. 549–562 (2011).

[9] 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).

[10] Kim, D., Renganarayanan, L., Rostron, D., Rajopadhye, S. and Strout, M. M.: Multi-level Tiling: M for the Price of One, *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, pp. 51:1–51:12 (2007).

[11] McSema: https://github.com/trailofbits/mcsema (2017).

[12] 幸　朋矢，佐藤幸紀，遠藤敏夫：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，研究報告ハイパフォーマンスコンピューティング（HPC）, 2018-HPC-163，pp. pp.1–7 (2018).

[13] Luk, C.-K. et al.: Pin: building customized program analysis tools with dynamic instrumentation, *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pp. 190–200 (2005).

[14] Jain, A., Laurenzano, M. A., Tang, L. and Mars, J.: Continuous Shape Shifting: Enabling Loop Co-optimization via Near-free Dynamic Code Rewriting, *International Symposium on Microarchitecture* (2016).

[15] Damschen, M., Riebler, H., Vaz, G. and Plessl, C.: Transparent Offloading of Computational Hotspots from Binary Code to Xeon Phi, *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pp. 1078–1083 (2015).

© 2018 Information Processing Society of Japan