Vol.2019-HPC-168 No.16
2019/3/6

凸多面体モデルを利用したループ分割機能の実装

津金 佳祐[1,a)]　一場 利幸[1,b)]　新井 正樹[1,c)]　田原 司睦[1,d)]

概要：計算機環境が多様化する中でアプリケーションの開発者は，環境に合わせたプログラムの最適化を求められる．最適化には様々な手法があるが，その多くはプログラムの変更を伴い，計算機環境の特性を理解した上での実装が求められるため，プログラムの変更コストは高いと言える．そこで，凸多面体モデルによるプログラムの自動最適化を実行可能な Polly が注目されている．Polly はプログラム中のループを自動検出し最適化するが，計算機環境の特性に合わせた自動最適化には不十分な点が多い．我々は HPC166 にて，Polly の最適化機能の中でループ分割に着目し，任意のループ分割粒度を指定可能なコンパイルオプションの提案とその実装に関する報告を行った．本稿では，分割アルゴリズムの改善と計算機環境の情報を用いた自動的なループ分割粒度を設定する実装とその評価を行う．計算機環境の情報として，ハードウェアプリフェッチを実行するために用いられるメモリフェッチストリームの数を考慮する．実装した Polly を PolyBench と NAS Parallel Benchmarks（NPB）に適用して，ARM プロセッサ上で性能評価を行う．併せて，既存の GCC や Clang/LLVM と性能比較を行うことで，メモリフェッチストリーム数を考慮した自動的なループ分割粒度の設定による性能向上を調査した．既存コンパイラと比較して提案実装により，最大で 30%の性能向上を確認した．

1. 序論

Intel，AMD や富士通などが High Performance Computing（HPC）向けのプロセッサを提供しており，各社独自の特性を持つプロセッサを用いた様々なスーパーコンピュータの開発が進められている．そのため，アプリケーションの開発者には多様な環境に合わせたプログラムの最適化が求められている．最適化には様々な手法があるが，性能ボトルネックとなりやすいループに着目すると，ループ分割，融合，交換，タイリング，アンローリングやベクトル化などが挙げられる．これらのループ最適化の適用にはプログラムの変更を伴う場合が多く，キャッシュサイズ，メモリフェッチストリーム数，レジスタ数やベクトル長など計算機環境の特性を理解した上での実装が求められるため，最適化のプログラミングコストは高い．

そのような背景から，凸多面体モデル [1, 2] によるプログラムの自動最適化に注目が集まっている．代表的な実装として Polly [3, 4]，PLUTO [5] や Graphite [6] などがあるが，本稿では Polly を対象とする．Polly は，C/C++や Fortran を対象とした LLVM [7] ベースのオープンソースコンパイラ Clang [8] や Flang [9] に付随する外部モジュールである．コンパイル時にプログラム中のループを自動検出し，上記に示したループ最適化を自動的に適用する．コンパイルオプションにより適用する最適化の選択やパラメータの変更も可能であり，アプリケーション開発者はプログラム変更することなく様々な最適化を試すことができる．

しかし，計算機環境において考慮すべきパラメータは多く，全通りの組み合わせを試すことは困難である．そのため，コンパイラによる自動的な最適化手法の選択やパラメータ設定が求められるが，現状の Polly に十分な機能は実装されていない．ループ分割機能に着目すると，2019 年 1 月時点でのメジャーリリースである LLVM 7.0.0 の Polly では，計算機環境に合わせた自動的な分割粒度の設定は実装されていない．また，コンパイルオプションによる指定は最大限ループ分割/融合を実行する 2 種類のみであり，ユーザによって分割粒度を指定することもできない．

我々は HPC166 [10] にて，Polly における自動的なループ分割粒度を設定する実装の前段階として，ユーザが任意にループ分割粒度を指定可能なコンパイルオプションの提案とその実装に関する報告を行った．結果として，任意のループ分割粒度を指定可能となったが，分割粒度を指定することで既存の最大限ループ分割/融合した以上の性能が得られたベンチマークプログラムは少なかった．そこで本稿では，以下の 3 点を行うことを目的とする．

[1] 株式会社富士通研究所
FUJITSU LABORATORIES LTD.
[a)] tsugane.keisuke@fujitsu.com
[b)] t.ichiba@fujitsu.com
[c)] arai.masaki@fujitsu.com
[d)] tabaru@fujitsu.com

- ループ分割アルゴリズムの改良による性能向上.
- 計算機環境の情報を用いた自動的なループ分割粒度を設定する実装.
- ベンチマークプログラムを用いた性能評価.

自動的な分割粒度の設定には，ハードウェアプリフェッチを実行するために用いられるメモリフェッチストリームの数を考慮する．実装した Polly を PolyBench [11] と NAS Parallel Benchmarks（NPB）[12] に適用して，ARM プロセッサ上で性能評価を行う．併せて，既存コンパイラとの性能比較を行うことで，メモリフェッチストリーム数を考慮した自動的なループ分割粒度の設定による性能向上を調査する．

本稿の構成は次の通りである．第 2 章で Clang/LLVM や Polly を紹介する．第 3 章でメモリフェッチストリームについて述べる．第 4 章ではループ分割アルゴリズムの改良及びメモリフェッチストリーム数を考慮した自動的なループ分割粒度を設定する実装の詳細を示す．第 5 章で提案実装を PolyBench と NPB に適用して性能評価を行い，第 6 章にて結論と今後の課題を述べる．

2. Polly

Clang/LLVM の説明に加え，凸多面体モデルによるプログラムのループ最適化を実行可能な Polly の概要とその実装の詳細を示す．

2.1 Clang/LLVM

LLVM [7] はイリノイ大学により開発が始められたあらゆる段階（コンパイル，リンクや実行）でプログラムを最適化するよう設計されたオープンソースのコンパイラ基盤である．基本的な構造は，ユーザ記述のプログラムを LLVM の中間表現である LLVM Intermediate Representation（IR）へと変換するフロントエンド，LLVM IR に対して解析・最適化を行うパスと LLVM IR を実行オブジェクトへと変換するバックエンドの 3 種類により構成される．代表的なフロントエンドの実装としては C/C++対応の Clang [8] がある．Clang 7.0.0 では x86 や ARM の命令セットに対応し，言語仕様 C++11/14 や OpenMP 3.1 への完全対応がされている．

2.2 Polly の概要

図 1 に Polly の実行フローを示す．Polly は Clang/LLVM の中で，LLVM IR に対して最適化し LLVM IR を出力する外部モジュールとして存在する．Polly では凸多面体モデルにより，線形代数学的にプログラム（特にループ）を解析，モデル化し，データの依存関係や境界条件を計算することで，並列性の抽出や最適化手法の適用を可能とする．適用可能な最適化手法としては，ループ分割，融合，交換，タイリング，アンローリングやベクトル化に加えて

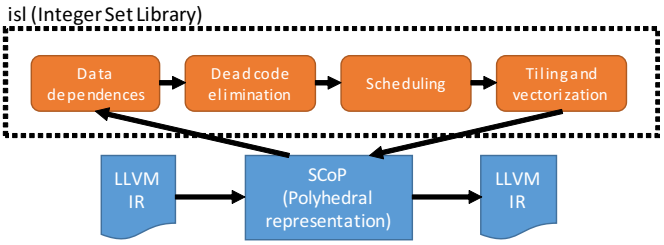

図 1 Polly の実行フロー

OpenMP 指示文の自動生成によるループの並列化などがあり，ユーザにより最適化手法を選択することもできる．

Polly はプログラムを解析し Static Control Part（SCoP）[2, 13] 単位でループを検出する．SCoP として検出できるループは，配列インデックス，ループ条件文が全てアフィン式（ループ変数の線形結合と定数項の加算である式）の for ループである．近年ではさらに，while ループやポインタ演算も対象となっている．SCoP 以外では Polly による最適化は実行されない．検出された SCoP は Polly の中間表現である Polyhedral Representation へと変換され，保持される．Polyhedral Representation では，LLVM の BasicBlock か BasicBlock の 1 文を 1 Statement（Stmt）とし，Stmt の実行される index の範囲を Domain，Stmt の実行順序を Schedule，アクセスするメモリ領域の依存関係を Access と表現する．

Polly による最適化は Integer Set Library（isl）[14] と呼ばれるライブラリを用いて実装されている．図 1 の橙色のブロック群が最適化フェーズを表す．4 種類の処理で構成され，依存解析，不要コードの削除，スケジューリング，タイリングやベクトル化の順に SCoP 単位で実行される．以下に各処理の説明を示す．

(1) 依存解析：SCoP が持つ Domain，Schedule と Access の情報を用いて，各 Stmt がアクセスするメモリ領域に対する処理を Read や Write などアクセスパターンに合わせて分類する．また，Stmt 間のデータ依存（フロー，出力と逆依存）の解析も行う．
(2) 不要コードの削除：依存解析結果を用いて定数畳み込みや伝搬などの最適化を行い，不要変数やループの削除を行う．
(3) スケジューリング：依存解析より得られた Stmt 間のデータ依存をエッジとし，Stmt をノードとするスケジュールグラフを生成する．このグラフは依存関係の方向をエッジの向きとした有向グラフとなる．生成されたスケジュールグラフより SCoP 内の Stmt のスケジューリングを行い，グラフ中の Stmt の位置や依存関係を基にループ分割，融合や交換を実行する．
(4) タイリングやベクトル化：最後にループ長や Stmt のメモリアクセスを基にループのタイリングやベクトル化を行う．Polly ではパターンマッチングによる最適

```
1 int A[N][N][5], B[5];
2 for (int i = 0; i < N; i++)
3   for (int j = 0; j < N; j++)
4     for (int k = 0; k < 5; k++)
5       A[i][j][k] += B[k];
```

図 2 有向グラフのエッジが生成されない例

化も実装されており，特定の演算に特化したループ変形を行う．Polly 7.0.0 では行列積のみ対応している．上記の処理を全ての SCoP に対して実行した後，LLVM IR へと変換を行い，Polly による処理は終了する．

2.3 スケジューリング

本稿では Polly のループ分割機能の拡張を行うため，ループ分割や融合を実行するスケジューリングのみ詳細を示す．スケジューリングでは，生成されたスケジュールグラフに対して強連結成分分解 [15] を行う．強連結成分分解は有向グラフに対する分割アルゴリズムである．グラフ中の循環部分を強連結成分と呼び，強連結成分を一つのノードとして扱う．Polly の場合はノードが Stmt，エッジが依存関係とその向きを表す．依存関係の向きは，その順序にノードを実行しなければならないことを示す．つまり，実行順序さえ守ればノードが持つ Stmt を異なるループへと分割可能であることを表す．一方で，強連結成分は循環する依存関係を持つため，異なるループへと分割不可能な Stmt 群となる．ループ分割は各ノードをそれぞれ単一のループとし，エッジの向きに実行するように並び変えることで実行される．

Polly におけるループ分割や融合のためのコンパイルオプションには`-polly-opt-fusion` がある．`min/max` により，最大限ループ分割/融合の実行を指定できる．`min` が指定された場合は，上記に示したスケジュールグラフが生成される．分割不可能な強連結成分内の Stmt 以外は，エッジの向きに実行するように並び替え，一つのノードが一つのループとなるように分割される．`max` が指定された場合は，`min` が指定された場合とスケジュールグラフの生成方法が異なる．ノード間のエッジが双方向のエッジとして生成される．生成されたグラフは常に強連結成分となるため，本来ループ分割可能であっても異なるループへと分割されない．つまり，最大限のループ融合となる．

2.4 ループ分割アルゴリズムの問題点

Polly のループ分割や融合はデータ依存に基づくスケジュールグラフを用いて実行されるため，データ依存が無ければ最適化の対象外となる．例として，**図 2** のような場合が挙げられる．最内ループが定数のため Clang の最適化レベルによっては，ループアンローリングが適用される．この時生成された LLVM IR に Polly を適用すると，5 つの Stmt からなる SCoP が検出される．各 Stmt は配列 *A[i][j][0]* から *A[i][j][4]* に対するロード，ストアと配列 *B[0]* から *B[4]* に対するロードを実行する．Stmt 間で演算範囲が重なっておらず，データ依存が無いためエッジは生成されない．従って，`-polly-opt-fusion` で `min/max` のどちらを指定しても，5 つの Stmt は最大限ループ分割が実行された状態となる．これにより，元の実装のループアンローリングの効果が得られずに，性能が低下する可能性がある．

Polly では，エッジがあればループ長やループのネスト数が異なるノード間でもループ融合が可能である．しかし，ループ長が異なるノードを融合した場合は，ループ内で条件分岐により一つのループとして表現されるため，融合前と比較して実行される条件分岐数が増加する．また，Polly の実装として，タイリングが適用されるループは，少なくとも 2 次元のループが完全ネストしている場合である．従って，不用意にループ融合を実行することで他の最適化を妨げ，性能が低下する可能性がある．

3. メモリフェッチストリーム

本稿ではループ分割粒度を自動的に設定するためにメモリフェッチストリームのハードウェア情報を利用する．CPU には低速なメインメモリから高速なキャッシュメモリへ，予めデータを転送しておくことで，メモリアクセスレイテンシの隠蔽を行うプリフェッチ機能が搭載されている．プリフェッチはソフトウェアとハードウェアによる 2 種類の実装がある．ソフトウェアプリフェッチはコンパイラによる解析や，ユーザが手動で記述することで実行される．一方で，ハードウェアプリフェッチは，プログラムの実行中にメモリアドレスやメモリアクセスの規則性などを基に自動的に実行される．このメモリアドレスやメモリアクセスの規則性などを保持しておくバッファをメモリフェッチストリームと呼ぶ．メモリフェッチストリームは有限であるため，その数を超える異なるメモリアドレスよりロードが発生した場合には，メモリフェッチストリームのエントリの追い出しが発生する．ループ内で追い出しが発生した場合，ハードウェアプリフェッチが効率よく実行されず，性能が低下する可能性がある．従って，ループ内で使用されるメモリフェッチストリーム数を各計算機環境が持つ値以下にしておくことで性能向上が見込める．

4. 実装

メモリフェッチストリーム数を用いた自動的なループ分割粒度を設定する実装の詳細を示す．基本的な方針として，一度最大限にループ分割を実行してから，条件に合うループのみを融合する．条件については後述する．本実装は HPC166 [10] で報告した粒度を指定可能なループ分割実

装を基にした．実行の流れは以下の通りである．

(1) プログラムの依存解析を実行
(2) 得られた Stmt 間のデータ依存を基に，スケジュールグラフを生成．
(3) スケジュールグラフに対して強連結成分分解を実行し，最大限ループ分割を実行．
(4) 条件に合うエッジを選択し，両端ノードを融合．
(5) (4) を繰り返し，条件に合うエッジが無い場合に処理を終了．

提案実装では (1) のプログラムの依存解析と (4) で用いる条件の拡張を行う．

4.1 仮エッジの追加

2.4 節にて，データ依存が無い場合にスケジュールグラフのエッジが生成されず，Polly の最適化の対象外となる問題を示した．そこで，同じ配列に対するアクセスがある場合に仮エッジを追加する手法を提案する．この手法で得られたエッジは，スケジュールグラフで用いられるエッジと同等に扱われるものとする．

Polly の依存解析では，各 Stmt がアクセスする配列情報を取得することができる．その情報より (1) にて，同じ配列をアクセスする Stmt 間に依存関係があるとし，正規プロセスで得られたデータ依存に加える．その後は，(2) の処理により仮エッジを含むスケジュールグラフが生成される．余分な依存関係を追加することで生成される仮エッジを用いるため，(3) のスケジュールグラフ内に本来できないはずの強連結成分ができ，ループ分割ができなくなる場合も想定される．そこで，自 Stmt に最も実行順序が近い Stmt から依存解析をし，Stmt の実行順序を遡った仮エッジは生成しないこととした．各 Stmt から生成される仮エッジを 1 以下に制限することで，仮エッジの生成を最小限におさえ，できる限り強連結成分とならない実装とした．この手法により，Polly の最適化の適用範囲を増やすことができる．

4.2 エッジ選択条件の追加

任意数の分割粒度を指定する実装では，エッジの両端ノードが持つループのネスト数が最大である事を (4) の条件としていた．これはネストが深いループは演算量が多く，最適化効果が大きい場合が多いためである．提案実装では，この条件に加えて，任意数のメモリフェッチストリームを超えない場合に両端ノードを融合するという条件を追加する．一般的に異なる配列に対するロードが実行される場合に，異なるメモリフェッチストリームが用いられる．そのため，エッジの両端ノードが持つ Stmt がアクセスする配列情報を取得し，融合した場合の異なる配列へのアクセス数をカウントする．この値が任意数を超えない場合に (4) のノードの融合を実行する．

表 1 評価環境

CPU	Cavium ThunderX2(R) CN9975 v2.1 ×2 2.2GHz
Memory	DDR4 2666MHz 128GB (16GB ×8)
Software	GCC version 8.2.0 Clang/LLVM 7.0.0 Polly 7.0.0 + impl.
OS	Ubuntu 18.04 LTS

```
1 /* メモリフェッチストリーム数: 2 */
2 for (int i = 0; i < N; i++) {
3   A1[i] = A1[i] + A2[i];
4 }
5 /* メモリフェッチストリーム数: 3 */
6 for (int i = 0; i < N; i++) {
7   A1[i] = A1[i] + A2[i] + A3[i];
8 }
9 /* メモリフェッチストリーム数: … */
```

図 3 メモリフェッチストリーム数調査用ベンチマークプログラム

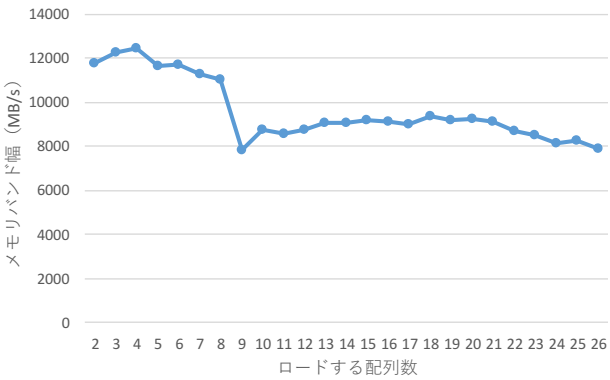

図 4 メモリフェッチストリーム数の調査結果

2.4 節で示した通り，Polly ではエッジがあればループ長やループのネスト数が異なるノード間の融合も実行されるため，条件分岐の増加や他の最適化を妨げるという問題がある．そのため，ループ長とループのネスト数が同じノードのみを融合するという条件も追加した．

5. 評価

本章では，自動的にループ分割粒度を設定する実装の性能評価を行う．実行環境として**表 1** に示す ARM プロセッサを用いる．Cavium ThunderX2 は，メモリフェッチストリーム数が公開されていないため，まずその値の調査を行う．その後，得られたメモリフェッチストリーム数を入力として提案実装をベンチマークプログラムに適用し，性能評価を行う．

5.1 メモリフェッチストリーム数の調査

メモリフェッチストリームは，一般的に異なる配列のロードが行われる場合に，異なるメモリフェッチストリー

ムが用いられる．そこで，Stream ベンチマーク [16] を元に，**図 3** のようなベンチマークプログラムを作成した．メモリフェッチストリーム数を 2 使う場合，配列 *A1*, *A2* の 2 種類を配列 *A1* に加算する．同様にメモリフェッチストリーム数を 3 使う場合は，配列 *A1*, *A2*，*A3* の 3 種類を配列 *A1* に加算する．このように，異なる配列を増やすことで，使用されるメモリフェッチストリーム数を増加させる．メモリフェッチストリーム数を増加させ，性能が低下した時の値がハードウェアが持つメモリフェッチストリームの値と考える．一つの配列は double 型のサイズ 10000000（Stream ベンチマークの default 値）で評価する．

図 4 にメモリフェッチストリーム数の調査結果を示す．結果として，ロードする配列数が 8 を超えたところで約 3000MB/s の性能低下を確認した．この時にメモリフェッチストリーム数を超えるロードが実行されたと考えられる．よって以降の評価では，メモリフェッチストリーム数として 8 を採用する．

5.2 環境設定

対象のベンチマークプログラムとして，PolyBench/C benchmark suite 4.2.1-beta [11] と NPB [12] を用いた．PolyBench は linear algebra，stencils，datamining と medley の 4 種類で構成され，linear algebra 内で blas，kernels と solvers に分類される，全 30 本のベンチマークプログラム集である．行列積やステンシル演算など HPC 分野で広く用いられる演算も含まれる．NPB は，NASA Advanced Supercomputing Division によって開発された，並列コンピューティングのためのベンチマークプログラム集である．カーネルコードと Computational Fluid Dynamics（CFD）アプリケーションで構成されており，一部を除き Fortran で実装されている．本稿では，CFD アプリケーションの中の Lower-Upper Gauss-Seidel solve（LU）を対象とした．また，LU は Fortran で実装されているため，University of Versailles Saint Quentin en Yvelines で開発された C 言語版の NPB [17] の LU を用いた．

Polly は並列化可能ループに対して OpenMP によるループ並列も適用できるが，ループ分割による性能の差異を評価するため，本稿では 1 スレッドのみを用いた性能評価を行う．比較対象は，GCC，Polly を用いない Clang，Polly と`-polly-opt-fusion=max` を用いて最大限のループ融合を実行する Clang，Polly と`-polly-opt-fusion=min` を用いて最大限のループ分割を実行する Clang の 4 種類でコンパイルしたバイナリとする．コンパイルオプションは GCC，Clang ともに`-O3`，`-march=armv8.1-a`, `-mcpu=thunderx2t99`, `-ffp-contract=fast` とする．問題サイズは，PolyBench は *LARGE* と *EXTRALARGE*，NPB LU は *CLASS W* と *A* を用い，各パターンを 10 回実行した内の最良値を用いる．

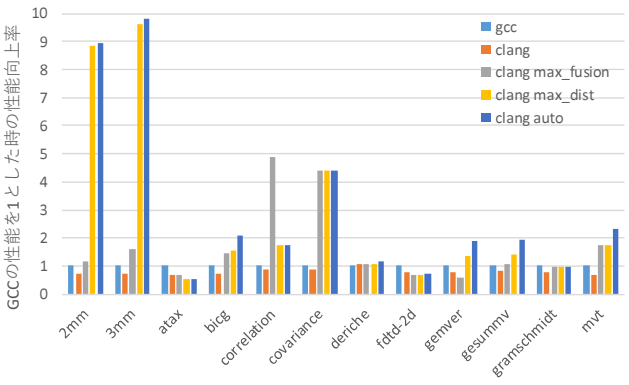

図 5 PolyBench: LARGE の性能評価（GCC の性能を 1 とした時の性能向上率）

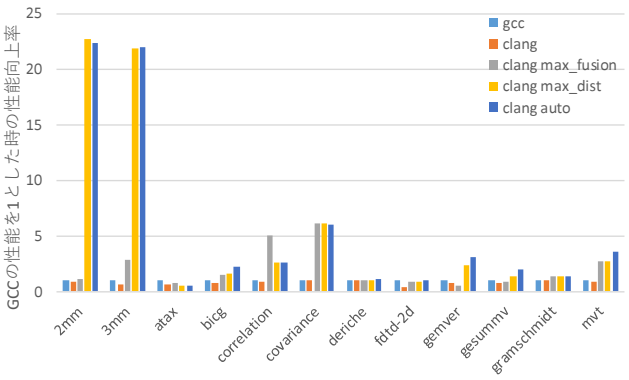

図 6 PolyBench: EXTRALARGE の性能評価（GCC の性能を 1 とした時の性能向上率）

表 2 NPB LU の性能評価（Mop/s）

CLASS	W	A
gcc	1916.96	1661.81
clang	1931.98	1760.45
clang max_fusion	1917.93	1734.46
clang max_dist	1917.59	1735.98
clang auto	1950.08	1767.97

PolyBench は小規模なベンチマークプログラム集のため，既存実装の最大限ループ分割/融合した場合と提案実装の分割粒度の設定が同値となる場合がある．そこで，提案実装が`-polly-opt-fusion` の `min/max` 以外の分割粒度となるかを調査する．確認は，LLVM IR とコンパイルオプション`-debug-only=polly-ast` を指定し，Polly による最適化を適用した後の SCoP の出力を得て行った．結果として，提案実装により`-polly-opt-fusion` の `min/max` 以外の分割粒度となったベンチマークプログラムは 12 種類（2mm, 3mm, atax, bicg, correlation, covariance, deriche, fdtd-2d, gemver, gesummv, gramschmidt, mvt）とわかった．よって，この 12 種類のみの性能評価を行う．

5.3 性能評価

図 5 に問題サイズ *LARGE*，**図 6** に *EXTRALARGE* の

PolyBench の性能評価を示す．縦軸を GCC の性能を 1 とした場合の性能向上率とし，横軸にベンチマークプログラムをとった．凡例の clang max_fusion は最大限のループ融合，clang max_dist は最大限のループ分割，clang auto が提案実装を表す．結果として，5 種類（bicg，deriche，gemver，gesummv，mvt）のベンチマークプログラムで提案実装により，GCC，Clang，Polly の最大限ループ分割/融合した場合の最良値と比較して，最大 30%性能が向上した．また，5 種類（2mm，3mm，covariance，fdtd-2d，gramschmidt）のベンチマークプログラムでは，ほぼ同等の性能が得られた．その一方で，2 種類（atax，correlation）のベンチマークプログラムでは，性能が低下，もしくは GCC 以下の性能となった．

表 2 に NPB LU の性能評価を示す．単位は一秒あたりに何百万回の演算を行ったかを表す Mega Operation Per Second（Mop/s）である．結果として，提案実装により数 Mop/s ではあるが性能が向上した．LU では，Polly を適用しなかった場合と比較して，最大限ループ分割/融合した場合に性能が低下している．LU で用いられる演算の多くは，最内ループが定数であり，2.4 節で示した問題が頻発し性能が低下したと考えられる．

6. 結論

本稿では，Polly のループ分割機能を拡張し，メモリフェッチストリーム数を考慮して自動的にループ分割粒度を設定する機能を実装した．また，Polly のループ分割や融合で用いられるスケジュールグラフに対して，仮エッジを追加することで Polly の最適化の適用範囲を広げた．さらに，エッジ選択時にループ長とループのネスト数を考慮する条件の追加も行った．提案実装を PolyBench と NPB に適用し ARM プロセッサ上で評価した結果，GCC，Clang，Polly の最大限ループ分割/融合した場合の最良値と比較して，最大で 30%の性能向上を確認した．

今後の課題として，提案実装により性能が低下したベンチマークプログラムもあったため，今後はそれらの性能解析を行い，特定の演算パターンの場合は提案実装を適用しない実装を行う予定である．また，他のベンチマークプログラムや実アプリケーションへの適用を通して，更なる実装の最適化を行う予定である．

参考文献

[1] Paul Feautrier, Some efficient solutions to the affine scheduling problem: I. One-dimensional time, International Journal of Parallel Programming, Volume 21, Issue 5, pp.313–347, October 1992.
[2] Sylvain Girbal, Nicolas Vasilache, Cédric Bastoul, Albert Cohen, David Parello, Marc Sigler, Olivier Temam, Semi-Automatic Composition of Loop Transformations for Deep Parallelism and Memory Hierarchies, International Journal of Parallel Programming, Volume 34, Issue 3, pp.261–317, June 2006.
[3] Polly LLVM Framework for High-Level Loop and Data-Locality Optimizations, https://polly.llvm.org
[4] Tobias Grosser, Hongbin Zheng, Ragesh Aloor, Andreas Simbürger, Armin Größlinger, Louis-Noël Pouchet, Polly - Polyhedral Optimization in LLVM, First International Workshop on Polyhedral Compilation Techniques (IMPACT 2011), Chamonix, France, April 2011.
[5] Uday Kumar Reddy Bondhugula, Effective automatic parallelization and locality optimization using the polyhedral model, Doctoral Dissertation, Ohio State University Columbus, 2008.
[6] Sebastian Pop, Albert Cohen, Cédric Bastoul, Sylvain Girbal, Georges-andré Silber, Nicolas Vasilache, GRAPHITE: Polyhedral Analyses and Optimizations for GCC, In Proceedings of the 2006 GCC Developers Summit, 2016
[7] Chris Lattner, Vikram Adve, LLVM: a compilation framework for lifelong program analysis & transformation, International Symposium on Code Generation and Optimization (CGO 2004), pp.75–86, San Jose, CA, USA, 2004.
[8] Clang: a C language family frontend for LLVM, http://clang.llvm.org
[9] Flang, https://github.com/flang-compiler/flang
[10] 津金佳祐, 一場利幸, 中村祐次郎, 新井正樹, 田原司睦, Polly のループ分割機能の拡張, 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-166(13), pp.1–6, 北海道, 2018.
[11] PolyBench/C: the Polyhedral Benchmark suite, http://web.cse.ohio-state.edu/∼pouchet.2/software/polybench
[12] NASA Advanced Supercomputing Division, NAS Parallel Benchmarks, https://www.nas.nasa.gov/publications/npb.html
[13] Paul Feautrier, Dataflow analysis of array and scalar references, International Journal of Parallel Programming, Volume 20, Issue 1, pp.23–53, February 1991.
[14] Sven Verdoolaege, isl: An Integer Set Library for the Polyhedral Model, Third International Congress on Mathematical Software (ICMS 2010), pp.299–302, Kobe, Japan, September 2010.
[15] Robert Tarjan, ‘Depth-first search and linear graph algorithms, 12th Annual Symposium on Switching and Automata Theory (swat 1971), pp.114–121, East Lansing, MI, USA, 1971.
[16] John D. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers, https://www.cs.virginia.edu/stream
[17] University of Versailles Saint Quentin en Yvlines, NAS Parallel Benchmarks 3.0 Unofficial OpenMP C Version, https://github.com/benchmark-subsetting/NPB3.0-omp-C