Vol.2018-HPC-166 No.13
2018/9/28

Pollyのループ分割機能の拡張

津金 佳祐[1,a)]　一場 利幸[1,b)]　中村 祐次郎[2,c)]　新井 正樹[1,d)]　田原 司睦[1,e)]

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，多くはプログラムの変更を伴い，キャッシュサイズ，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必要なため，プログラムの開発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．そこで，本研究ではループ分割機能に着目し，Polly が自動的にループ分割粒度を決定する処理の前に，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を目的とする．実装した Polly を PolyBench と呼ばれるベンチマークプログラムに対して適用することで，様々な種類の演算に対して本実装による分割粒度を指定可能であることを示す．併せて Intel と ARM プロセッサ上で既存コンパイラとの性能比較を行い，分割粒度の指定による性能向上を調査した．結果として，本実装が PolyBench の全てのベンチマークプログラムに対して適用可能であることを示し，一部ループ分割粒度の指定による性能向上も確認した．

1. 序論

Intel，AMD や富士通などが HPC（High Performance Computing）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，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，最適化のため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，本研究では 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の 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の自動最適化を適用する．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を行うことなく最適化を試せる．

しかし，計算機環境に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パラメータチューニングが求められるが，現状の Polly にそのような機能が十分に実装されているとは言えない．ループ分割機能に着目すると，2018 年 8 月時点でのメジャーリリースである LLVM 6.0.0 に含まれる 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としてもループ分割を最大/最小限行う 2 種類の指定のみであり，ユーザによる細かな粒度での分割指定もできない．ユーザが任意に粒度を指定可能な機能も必要であるが，計算機環境に合わせてコンパイラが自動的にループ分割の粒度を設定することが理想である．そこで，本研究では Polly における自動的なループ分割粒度の設定の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を目的とする．PolyBench [10] と呼ばれる行列積やステンシル演算など HPC 分野における典型的な演算パターンを持つベンチ

[1] 株式会社富士通研究所
FUJITSU LABORATORIES LTD.
[2] 株式会社メトロ
Metro Inc.
[a)] tsugane.keisuke@jp.fujitsu.com
[b)] t.ichiba@jp.fujitsu.com
[c)] toyo@metro.co.jp
[d)] arai.masaki@jp.fujitsu.com
[e)] tabaru@jp.fujitsu.com

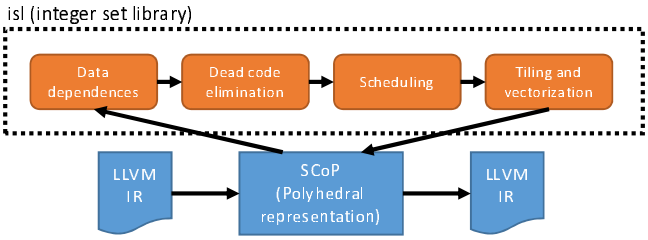

図 1　Polly の実行フロー

マークプログラム集に対して本実装を適用し動作検証をすることで，様々な種類の演算に対して本実装により分割粒度を指定可能であることを示す．また，Intel と ARM プロセッサ上で既存コンパイラとの性能比較を行い，分割粒度の指定により性能向上が見られるかを調査する．

本稿の構成は次の通りである．第 2 章にて Clang/LLVM や Polly の説明を述べる．第 3 章にて本研究で提案するループ分割粒度を指定可能なコンパイルオプションの記述方法と実装を示す．第 4 章で実装したコンパイルオプションを PolyBench へ適用して動作検証を行うとともに性能評価を行い，第 5 章において結論と今後の課題を述べる．

2. Polly

本章では Clang/LLVM の説明と，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 IR（Intermediate Representation）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 Pass 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] があり，Clang 6.0 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 OpenMP 指示文の自動生成によるループの並列化があり，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 SCoP（Static Control Part）[2, 11] 単位でループを検出する．SCoP として検出できるループは，ループ変数の線形結合と定数項の加算であるアフィン式を持つ for ループが主であるが，アフィン式以外にも 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 を一つの Statement（Stmt）とし，Stmt の実行される index の範囲を Domain，ループ中での Stmt の実行タイミングを 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 isl（integer set library）[12] 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．
(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：依存解析結果より Stmt 間のフロー，出力と逆依存をエッジとし，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により 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 6.0.0 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

本研究では 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 [13] 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方向を表す．エッジで示される依存関係は，エッジの向きにノードが持つ Stmt を実行する場合に，異なるループへと分割できることを示す．従って，強連結成分は循環する依存関係を持つため，分割不可能な Stmt 群である．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，最小/最大限ループ融合をする指定ができる．`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ることで一つの Stmt が一つのループとなるように分割される．`min` は最小限のループ融合を行うという指定ではあるが，オリジナルのプログラムがループ分割可能であれば，ループは分割される．つまり，最大限にループ分割すると言える．`max` が指定された場合はスケジュールグラフの生成方法が異なる．Stmt 間に依存関係があるが，実行順序によって分割可能であった場合に，その Stmt を持つノード間に双方向のエッジが生成される．分割可能な Stmt 群を強連結成分とすることで，異なるループへと分割されない．その後，強連結成分内でループ融合が実行される．強連結成分内の Stmt を対象として，依存関係やメモリアクセスを係数とした線形計画問題を作成する．依存関係を満たしつつもメモリアクセス距離が最小となる解を求めることで，キャッシュを考慮したループ融合を行う．`min` と同様に `max` の場合もループ分割に置き換えると，最小限にループ分割すると言える．今後の説明では用語の統一のため，`-polly-opt-fusion` をループ分割を実行するコンパイルオプションとして示す．

3. 実装

本章ではループ分割粒度を指定可能なコンパイルオプションの記述方法の提案とその実装を示す．2.3 節で示した通り，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は`-polly-opt-fusion` のみであり，最大限ループ分割をするか，しないかの 2 種類の選択しかできない．そこで本研究ではループ分割粒度を指定可能なコンパイルオプションとして以下を提案する．

`-polly-opt-distribution-level=`*#_OF_LOOPS*

このコンパイルオプションが指定された場合は，SCoP 内の複数 Stmt を指定されたループ数へと分割する．この時のループ数は SCoP 直下のループブロック数を表す．以下に挙動と制約を示す．

- `1` が指定された場合は，`-polly-opt-fusion=max` と同様に最小限のループ分割となる．
- Stmt 数と同値が指定された場合は，`-polly-opt-fusion=min` と同様に最大限のループ分割となる．
- Stmt 間に依存関係があり，指定された値のループ数を満たせない場合は，指定された値に可能な限り近くなるようにループ分割が行われる．
- Stmt が全て同一ループ内に存在し，分割不可能なループを最外ループとして持つ場合，最外ループ内でループ分割が実行される．
- `-polly-opt-distribution-level` が指定された場合は，`-polly-opt-fusion` は無視される．

本実装は`-polly-opt-fusion=min` が指定された場合の挙動を基とする．強連結成分は相互に依存関係を持つ分割不可能な Stmt 群であるため，そのノードを単一のループとする．その他のノードは一つの Stmt を一つのループとなるように分割し，エッジの向きに実行するように並び替えることで，最大限のループ分割を実行する．提案コンパイルオプションに Stmt 数と同値，もしくは大きい値が指定された場合は，この時点でループ分割を終了する．

Stmt 数より少ない値が指定された場合は，最大限にループ分割が実行された状態から指定されたループ数となるようにループ融合を行う．オリジナルの Polly の実装では，`-polly-opt-fusion=max` が指定された場合に全ての Stmt に関する線形計画問題を作成，解を得ることで融合可能なループを同時に融合する手法を用いている．本実装においては融合可能なノードを一つずつ選択しループ融合を実行する．融合するノードはノード間のエッジに対して重み付けを行うことで選択する．重みはエッジの両端ノードが持つ Stmt のメモリ領域へのアクセスを基に決定する．依存関係は同一メモリ領域に対してアクセスがあることを意味するため，プログラム中のノード間の距離を重みとすることで，同一メモリ領域へアクセスする Stmt を同一ループとする，キャッシュを考慮したループ融合となる．以上の動作を繰り返し，ノード数が指定された値と同値となった場合に処理を終了する．

4. 評価

本章では，実装したループ分割粒度を指定可能なコンパイルオプションの動作検証を行う．ベンチマークプログラムを対象に適用し，分割後のループの形状を示す．また，Intel と ARM プロセッサ上で性能評価を行い，ループ分割粒度の変更による性能向上が見られるかも併せて調査する．

4.1 動作検証

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 [10] を用いた．PolyBench は linear argebra，stencils，datamining と medley の 4 種類で

```
#define _PB_NI 799
#define _PB_NJ 899
#define _PB_NK 1099
#define _PB_NL 1199
/* D := alpha*A*B*C + beta*D */
for (i = 0; i < _PB_NI; i++)
  for (j = 0; j < _PB_NJ; j++) {
    /* Stmt1 */
    tmp[i][j] = SCALAR_VAL(0.0);
    for (k = 0; k < _PB_NK; ++k) {
      /* Stmt2 */
      tmp[i][j] += alpha * A[i][k] * B[k][j];
    }
  }
for (i = 0; i < _PB_NI; i++)
  for (j = 0; j < _PB_NL; j++) {
    /* Stmt3 */
    D[i][j] *= beta;
    for (k = 0; k < _PB_NJ; ++k) {
      /* Stmt4 */
      D[i][j] += tmp[i][k] * C[k][j];
    }
  }
```

図 2 2mm ベンチマークプログラム

```
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 899; c1 += 1) {
    Stmt1(c0, c1);
    for (int c2 = 0; c2 <= 1099; c2 += 1)
      Stmt2(c0, c1, c2);
    for (int c2 = 0; c2 <= 1199; c2 += 1) {
      if (c1 == 0)
        Stmt3(c0, c2);
      Stmt4(c0, c2, c1);
    }
  }
```

図 3 `-polly-opt-distribution-level=1` を指定した場合の 2mm の SCoP（`-polly-opt-fusion=max` と同一）.

構成され，linear argebra 内では 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．この全 30 本のベンチマークプログラムに対して提案コンパイルオプションを適用し，指定した値へのループ分割の実現を確認し，`-polly-opt-fusion` の `min` と `max` 以外の分割が可能かを調査する．確認は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．提案コンパイルオプションは Stmt 数分の全ての分割パターンを適用した．

結果として，提案コンパイルオプションにより `-polly-opt-fusion` の `min` と `max` 以外の分割粒度を指定可能なベンチマークプログラムは 12 種類（2mm, 3mm, adi, atax，correlation，covariance，deriche，doitgen，gemver，gesummv，gramschmidt，trmm）とわかった．残りの 18 種類のベンチマークプログラムは，提案コンパイルオプションによって得られた SCoP と，`min` と `max` の SCoP が同一であった．その内 14 種類（cholesky，durbin，fdtd-2d，floyd-warshall，head-3d，jacobi-1d，jacobi-2d，ludcmp，lu，mvd，nussinov，seidel-2d，symm，trisolv）は，`min` と `max` の SCoP 自体も同一であった．これは，Polly によって検出された SCoP が持つ Stmt 数が 1，もしくは依存関係によりループ分割が不可能と言える．残りの 4 種類（bicg，gemm，syr2k，syrk）は `min` と `max` の SCoP は異なるが Stmt 数が 2 であった．これは，Stmt 数が少なく `min` と `max` 以外の分割パターンが無いと言える．従って，PolyBench の全てのベンチマークプログラムに対して提案コンパイルオプ

```
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 899; c1 += 1) {
    Stmt1(c0, c1);
    for (int c2 = 0; c2 <= 1099; c2 += 1)
      Stmt2(c0, c1, c2);
  }
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 1199; c1 += 1)
    Stmt3(c0, c1);
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 1199; c1 += 1)
    for (int c2 = 0; c2 <= 899; c2 += 1)
      Stmt4(c0, c1, c2);
```

図 4 `-polly-opt-distribution-level=3` を指定した場合の 2mm の SCoP.

```
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 899; c1 += 1)
    Stmt1(c0, c1);
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 899; c1 += 1)
    for (int c2 = 0; c2 <= 1099; c2 += 1)
      Stmt2(c0, c1, c2);
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 1199; c1 += 1)
    Stmt3(c0, c1);
for (int c0 = 0; c0 <= 799; c0 += 1)
  for (int c1 = 0; c1 <= 1199; c1 += 1)
    for (int c2 = 0; c2 <= 899; c2 += 1)
      Stmt4(c0, c1, c2);
```

図 5 `-polly-opt-distribution-level=4` を指定した場合の 2mm の SCoP（`-polly-opt-fusion=min` と同一）.

表 1　評価環境（Intel プロセッサ）

CPU	Intel(R) Xeon(R) Gold 6148 ×2 2.4GHz
Memory	DDR4 2666MHz 192GB (16GB ×12)
Compiler	GCC version 8.1.0 Clang/LLVM 6.0.0
OS	CentOS Linux release 7.5.1804

表 2　評価環境（ARM プロセッサ）

CPU	Cavium ThunderX2(R) CN9975 v2.1 ×2 2.2GHz
Memory	DDR4 2666MHz 128GB (16GB ×8)
Compiler	GCC version 8.1.0 Clang/LLVM 6.0.0
OS	Ubuntu 18.04 LTS

ションによるループ分割粒度の指定が正常に動作したと言える．

実際にループ分割粒度を指定した場合の SCoP の例を示す．図 2 に示す 2 回の行列積を実行する 2mm を対象とする．for ループ内の式全てが Polly によって Stmt として判定されるため，分割可能なループ数は最大 4 となる．タイリングやパターンマッチングによる最適化も適用可能なベンチマークプログラムではあるが，ループ分割適用によるループ形状を確認したいため，本節で示す SCoP では適用しない場合のループ分割例を示す．

図 3，4，5 に`-polly-opt-distribution-level` にループ数 1，3，4 をそれぞれ指定した場合のループ分割後の形状を示す．Stmt 数が 4 であるため，ループ数 1 と 4 を指定した場合は，`-polly-opt-fusion` に `max` と `min` を指定した場合と同一である．結果として，ループ数が 1 と 4 へと分割されていることがわかる．ループ数 3 を指定した場合が本実装において実現可能とした分割例である．ループ融合が可能である Stmt1 と Stmt2 が単一のループとなり，合計で 3 ループとなるように分割されている．また，ループ数 2 を指定した場合は，図 2 に示すオリジナルの実装とループ数が同一となるため省略した．

4.2　性能評価

提案コンパイルオプションによりループ分割粒度を指定した場合の性能評価を行う．評価に用いるプロセッサは表 1，2 に示す Intel，ARM プロセッサである．評価対象は提案コンパイルオプションが適用可能である Poly-Bench の 12 種類とするが，3mm は 3 回の行列積を実行するベンチマークプログラムであり，2mm とほぼ同等の性能傾向であったため省略する．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可能であるが，ループ分割による性能の差異を評価するため，本研究においては 1 コアのみを用いて性能評価を行う．比較対象は，GCC，Polly を用いない Clang，Polly と`-polly-opt-fusion=max` を用いて最小限のループ分割

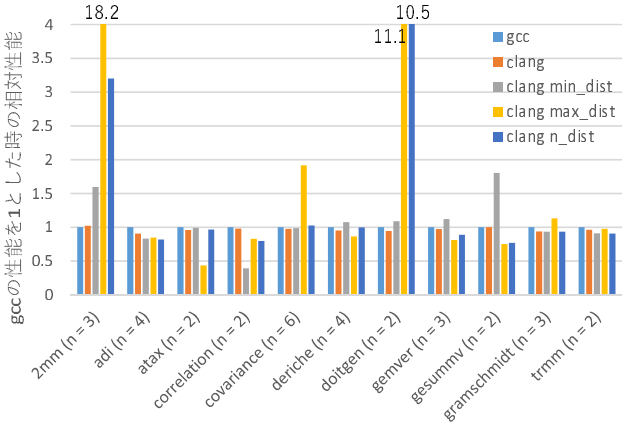

図 6　PolyBench の性能評価（Intel プロセッサ）

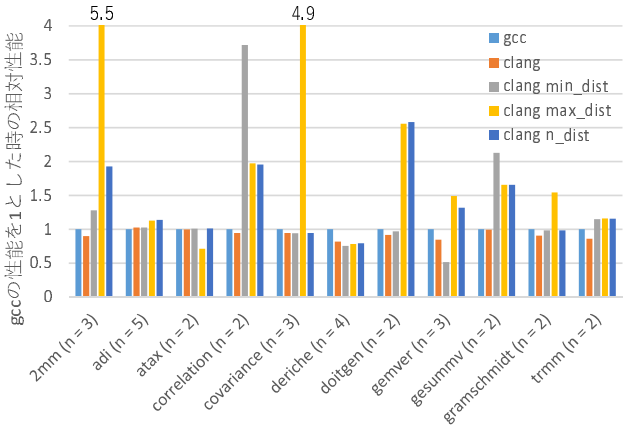

図 7　PolyBench の性能評価（ARM プロセッサ）

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．最大と最小を除いたループ分割数が複数選択可能な場合は，全てのパターンを試し最良値のみを示す．問題サイズは *LARGE_DATASET* 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 6，7 に Intel，ARM プロセッサ上で実行した Poly-Bench の性能評価を示す．縦軸を GCC の性能を 1 とした場合の相対性能とし，横軸にベンチマークプログラムをとった．凡例の min_dist は最小限のループ分割，max_dist は最大限のループ分割，n_dist が提案実装である任意数でのループ分割の結果を示す．ベンチマークプログラム名の括弧内に記述された n の値は，最良値を与えたループ分割数を表す．これらの図から，Intel，ARM プロセッサ上の両実行共に 2mm，covariance，doitgen，gramschmidt の最大限にループ分割した場合に比較的良い性能が出ていることがわかる．現在の Polly の実装では，最大限にループ分割を実行する`-polly-opt-fusion=min` がデフォルトとなっている．最大限にループを分割し，単純なループとした後にタイリングやパターンマッチングなどによる最適化を適用する．ループ融合によりループが複雑化し，Polly

による最適化が妨げられることのないように，配慮したと思われる．実際に最小限のループ分割をした場合の SCoP を確認したところ，2mm はパターンマッチングによる行列積の最適化，covariance，doitgen，gramschmidt はタイリングが適用できておらず，性能が低下したと考えられる．

今後の課題として，スケジューリングにより最大限にループ分割を行い，タイリングやベクトル化などの最適化を適用後に指定されたループ数へとループの変形を行うことで，Polly による最適化を取り入れたループ分割を行うことが挙げられる．また，ARM プロセッサ上においては，adi，atax，doitgen が提案コンパイルオプションを用いてループ分割粒度を指定することで性能向上が見られたが，理由は現在調査中である．

5. 結論

本研究では Polly におけるループ分割機能を拡張し，ユーザが任意の粒度を指定可能なコンパイルオプションの提案を行った．Polly に対してその機能の実装を行い PolyBench へ適用した結果，全 30 本全てのベンチマークプログラムに対して，提案コンパイルオプションを用いてループ分割が可能であることを示した．また，Intel，ARM プロセッサ上で性能評価を行った結果，ループ分割粒度の指定により一部性能向上が見られた．しかし，ループ分割によって Polly による最適化を妨げる場合があり，ループ分割，融合を適用するフェーズに問題があることがわかった．

今後の課題として，詳細な性能解析を行うことが挙げられる．また，スケジューリングにより最大限にループ分割を行い，タイリングやベクトル化などの最適化を適用後に指定されたループ数へと分割することで，Polly による最適化を取り入れたループ分割を行うことが挙げられる．

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] PolyBench/C: the Polyhedral Benchmark suite, http://web.cse.ohio-state.edu/ pouchet.2/software/polybench/
[11] Paul Feautrier, Dataflow analysis of array and scalar references, International Journal of Parallel Programming, Volume 20, Issue 1, pp.23–53, February 1991.
[12] 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.
[13] 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.