コードクローンに基づく括出し法

那須 孝志[1]　滝本 宗宏[1]

概要：プログラム中で，制御分岐にしたがって排他的に実行される同じ計算を，一つの計算として，分岐の外に出すプログラム変形を括出しと呼ぶ．括出しは，プログラムのコードサイズを小さくするコード圧縮や，GPU のウォープ中の SIMD 実行が，分岐発散によって生じる効率の低下を改善する最適化として有効である．しかしながら，まとまったサイズの計算の括出しはコストが高く，効率的なアルゴリズムが知られていないという問題があった．本研究では，前処理としてプログラム中の類似部分列であるコードクローンを検出し，そのコードクローンに基づいて，制御分岐の True 側と False 側にある同じ計算を括り出す手法を提案する．現在，本手法を用いて，分岐発散を除去する GPU 向け最適化を実装中であるが，これまでに得られた知見と中間結果を合わせて報告する．

キーワード：分岐発散，コードクローン，括出し

1. はじめに

GPU(Graphics Processing Unit) は，描画処理に特化した専用プロセッサであり，最近では，本来の利用にとどまらず，GPU が持つ高い演算能力を他の一般的な用途に利用しようという試みが，広く行われている [1].

GPU は，この演算性能を，SIMD[*1] 実行による高い並列処理によって実現する．一方で，GPU の SIMD 実行は，プログラム内に分岐が存在すると，ウォープ [*2] 中で分岐発散と呼ばれる効率の低下を起こすことが知られている．これは，分岐判定が各ウォープ単位で行われるために，ウォープ内のスレッドで分岐の方向が異なる場合，GPU が両方の分岐先の命令を順に実行しなければならないからである．

本稿では，GPU の分岐発散によって生じる効率の低下を改善する最適化として，コードクローンに基づく括出し法を提案する．本手法は，まず，括り出せる可能性のあるコードを，コードクローンとして検出する．その後，コードクローン中で依存グラフを生成し，合同な部分グラフを検出する．最終的に，検出した合同なグラフ情報を基に，各条件分岐の True 部と False 部から括出しを行う．

本論文の以降の構成は次の通りである．第 2 章で，括出しの詳細を例を用いて説明する．第 3 章で，本手法で用いるコードクローン検出手法について説明し，第 4 章で，依存グラフの定義および，部分依存グラフのグループ化について説明する．第 5 章で，本手法によるプログラム変形法を説明し，第 6 章で，現段階での評価を述べる．最後に，第 7 章で，結果と考察を述べる．

[1] 東京理科大学

[*1] Single Instruction Multiple Data：ひとつの命令を複数のデータに対して行う並列処理方式.

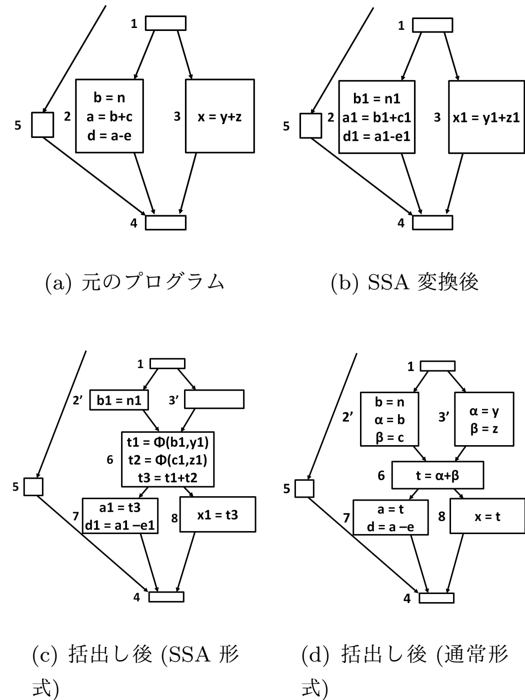
[*2] GPU 内で SIMD 実行が行われるスレッド・グループの単位.

2. 括出し

本研究の目的は，両方の分岐先で共通している計算をなるべく多く分岐内から括り出すことである．分岐外へ計算を括り出すことによって，分岐内で実行される計算量を削減し，分岐発散によって生じる効率の低下を改善する．

最適化は静的単一代入 (Static Single Assignment，以降 SSA と呼ぶ．) 形式の 3 アドレスコード上で行う．SSA 形式とは，プログラム表現形式の一つで，そのプログラム上の全ての変数に対して，その使用に対応する定義は一カ所しかないように表現したものである [3]．SSA 形式への変換は，基本的に，代入される変数に対して新しい名前を付けることによって行う．分岐の合流点に，複数の変数が到達し，その後一つの変数として扱う必要があるときは，Φ 関数と呼ばれる仮想関数を導入し，明示的に一つの変数に代入するようにする．Φ 関数は，制御が n 番目のブロックからきたときに，n 番目の引数の値を返す関数である．

実際に括出しを行う過程を，図 1(a) のプログラムを用いて示す．図 1(b) は，このプログラムの SSA 形式を表している．括出しは，一時変数を導入し，ブロック 1 と同じ分岐条件の新しい分岐を，合流ブロック (図 1(a)，(b) ブロック 4) の直前に挿入した後，新しい分岐に，共通の計算を移動する．括出し先としては，既存の合流ブロックを用いる方が望ましいが，図 1(b) ブロック 2 の $a1 = b1 + c1$ のように，括出し対象の計算が，ブロック内に $d1 = a1 - e1$ のような代入変数の使用をもつ場合，依存関係を壊さずに括出すのは簡単ではない．また，ブロック 5，4 を通る実行経路のように，分岐を通らない経路が存在する場合，存在しなかった計算 (括り出された計算) を挿入してしまう可能性があるので，括り出すことはできない．図 1(c) は，$a1 = b1 + c1$ と $x1 = y1 + z1$ の計算を新しい分岐のブロック (ブロック 6) に括り出す様子を示している．まず，ブロック 2 と 3 を，それぞれの計算の直後で，ブロック 2' と 7，ブロック 3' と 8 にそれぞれ分割する．その後，新しい分岐ブロック 6 を，ブロック 2'，3' とブロック 7，8 の間に挿入する．一旦，新しい分岐ブロック 6 を挿入すると，依存の問題や，新たな計算の挿入の問題を考慮することなく，$a1 = b1 + c1$ と $x1 = y1 + z1$ をブロック 6 に括り出すことができる．図 1(d) は，括出し後，SSA 形式から通常形式へ逆変換した結果を示している．



(a) 元のプログラム (b) SSA 変換後

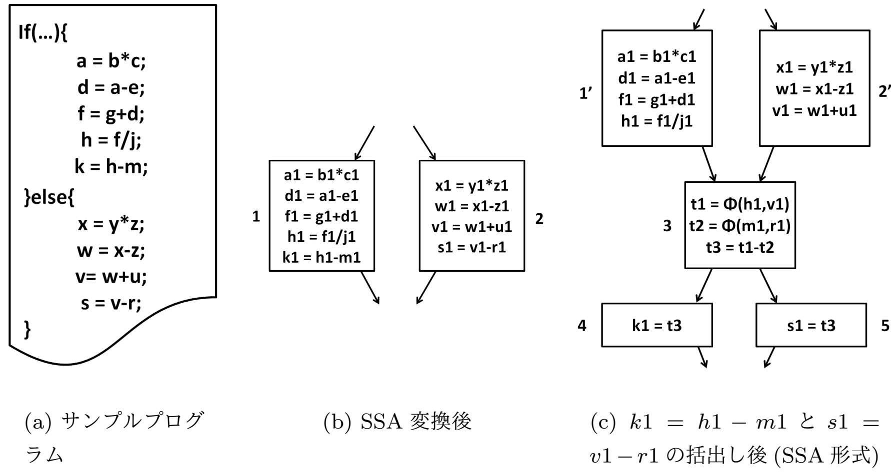
(c) 括出し後 (SSA 形式) (d) 括出し後 (通常形式)

図 1 括出しの例 1.

このように，新しい分岐の挿入によって，複数の計算を分岐外へ括り出すことができるようになるが，単純に演算子が一致する計算を括り出すと，本来括り出せる他の計算が括り出せなくなる可能性がある．この問題を図 2(a) のサンプルプログラムを用いて示す．

括出しを行う際は，制御フローグラフの各節点をトポロジカルソート順序 *3 で訪問し，各節点から分岐する 2 つの後続節について，命令を後ろ向きに辿りながら括り出していく．複数の計算を括

*3 どのノードもその出力辺のノードよりも前にくるように並べた順序．



(a) サンプルプログラム　(b) SSA 変換後　(c) $k1 = h1 - m1$ と $s1 = v1 - r1$ の括出し後 (SSA 形式)

図 **2**　括出しの例 2.

り出す場合も同様に，最初の括出しで挿入したブロックへ，依存関係を考慮しながら括り出す．

図 2(b) はサンプルプログラムの SSA 形式を示しており，図 2(c) は SSA 形式上で $k1 = h1 - m1$ と $s1 = v1 - r1$ の計算を括り出した結果を示している．これらの計算を括り出した後も，他の括出し可能な計算を見つけるために後ろ向きに辿っていく．例では，$f1 = g1 + d1$ と $v1 = w1 + u1$ が次の括出しの候補になる．しかしながら，$f1 = g1 + d1$ は，後続に $f1$ の使用 $h1 = f1/j1$ があるために，ブロック 3 に括り出すことができない．ここで，$f1 = g1 + d1$ の括出しを妨げている $h1 = f1/j1$ は，ブロック 2' に同じ演算子をもつ計算が存在しないことからブロック 3 に括り出すことができず，ブロック 3 の Φ 関数との依存関係からブロック 4 に移動することもできないことに注意されたい．したがって，この括出しを実現するためには，さらに新しい分岐の挿入が必要になる．一般に，複数分岐の挿入は，実行コストを増大させる可能性があるので，新しい分岐の挿入を 1 回に制限したとすると，この例では，1 つの計算しか括り出せないことになる．一方，最初に，$f1 = g1 + d1$ と $v1 = w1 + u1$ を括り出したとすると，次に，$d1 = a1 - e1$ と $w1 = x1 - z1$ の計算も新たな分岐の挿入なしに，括り出すことができる．さらに，$a1 = b1 * c1$ と $x1 = y1 * z1$ の計算も括り出せるので，合計で 3 つの計算を括り出せることになる．これは，これらの括り出す計算が依存関係にあり，さらに，それらの依存した計算の間に括出しの対象にならない式を含まないからである．本手法では，これらの依存関係を満たす計算の集合を検出するために，コードクローンを基に依存グラフを変形し，依存グラフ上で合同な部分グラフを検出することによって，対応する計算をグループ化する．そのうえで，括出しの対象を決定する．最終的に，このグループ化情報を利用して，多くの計算の括出しを実現する．

3. コードクローン

コードクローンは，ソースコード中の同一あるいは類似するコード片のことであり，数多くのコードクローン検出手法が提案されている [4]．本研究では，コードクローンの検出に Smith-Waterman アルゴリズムを拡張した手法 [4] を用いる．

3.1 Smith-Waterman アルゴリズム

Smith-Waterman アルゴリズム [5] とは，2 つの配列中から類似する部分配列を検出するアルゴリズムである．配列中にギャップが含まれていても検出できるという特徴をもつ．2 つの配列からなる表に

ついて, 各セルのスコアを計算した後, 最大スコアを基に各セルのスコアが計算されたセルへと後戻りしながら類似列を抽出するトレースバックを行う．図3に”ABCDACDBA”と”DBCADBCBD”という2つの配列に対してSmith-Watermanアルゴリズムを適用した例を示す．

		A	B	C	D	A	C	D	B	A
	0	0	0	0	0	0	0	0	0	0
D	0									
B	0									
C	0									
A	0									
D	0									
B	0									
C	0									
B	0									
D	0									

(a) 表の作成・初期化

		A	B	C	D	A	C	D	B	A
	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	1	0	0	1	0	0
B	0	0	1	0	0	0	0	0	2	1
C	0	0	0	2	1	0	1	0	1	1
A	0	1	0	1	1	2	1	0	0	2
D	0	0	0	0	2	1	1	2	1	1
B	0	0	1	0	1	1	0	1	3	2
C	0	0	0	2	1	0	2	1	2	2
B	0	0	1	1	1	0	1	1	2	1
D	0	0	0	0	2	1	0	2	1	1

(b) スコアの計算

		A	B	C	D	A	C	D	B	A
	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	1	0	0	1	0	0
B	0	0	1	0	0	0	0	0	2	1
C	0	0	0	2	1	0	1	0	1	1
A	0	1	0	1	1	2	1	0	0	2
D	0	0	0	0	2	1	1	2	1	1
B	0	0	1	0	1	1	0	1	3	2
C	0	0	0	2	1	0	2	1	2	2
B	0	0	1	1	1	0	1	1	2	1
D	0	0	0	0	2	1	0	2	1	1

(c) トレースバック

配列1: A B C D A C D B A
配列2: D B C A D B C B D
類似部分列

(d) 類似部分列

図 3 Smith-Waterman アルゴリズム適用例.

表の一番上の行と一番左の列は入力配列，2番目の行と列は0で初期化する．トレースバックは，各セルのスコアを計算するのに使用したセルからのポインタを順に遡る．セルのスコアは以下の数式に従って計算する．

$$v_{i,j}(2 \leq i, 2 \leq i) = max \begin{cases} v_{i-1,j-1} + s(a_i, b_j), \\ v_{i-1,j} + gap, \\ v_{i,j-1} + gap, \\ 0. \end{cases}$$

$$s(a_i, b_j) = \begin{cases} match & (a_i = b_j), \\ mismatch & (a_i \neq b_j). \end{cases}$$

$v_{i,j}$ は表の i 行 j 列のセルのスコア，a_i は一方の入力配列の i 番目の要素，b_j はもう一方の入力配列の j 番目の要素，$s(a_i, b_j)$ は a_i と b_j の類似度を表す．また，$match$，$mismatch$，gap はスコアパラメータを表している．例ではスコアパラメータを，$(match, mismatch, gap) = (1, -1, -1)$ に設定している．

3.2 Smith-Waterman アルゴリズムを用いたギャップを含むコードクローン検出手法

Smith-Waterman アルゴリズムを用いたコードクローン検出手法 [4] は，次の手順によってプログラム中のコードクローンを検出する．

ステップ1： 字句を検出し，一部を省略して正規化する．

ステップ2： 各文に対してハッシュ値を計算する．

ステップ3： 類似するハッシュ値列を検出する．

ステップ4： ギャップに対応する字句を検出する．

ステップ5： 出力を整形する．

入力は，ソースファイル，最小クローン長（字句数)，最大ギャップ率（検出された字句数に対するギャップ数の割合)，スコアパラメータ ($match$，$mismatch$，gap) であり，出力はコードクローンのペアのリストである．

4. 合同検出

本研究では，依存グラフとしてSSAグラフを用いる．SSAグラフは，演算子をラベルとする節点と，使用から定義への有向辺からなる [8]．各節点の横には，その値を保持する変数名を示してある．

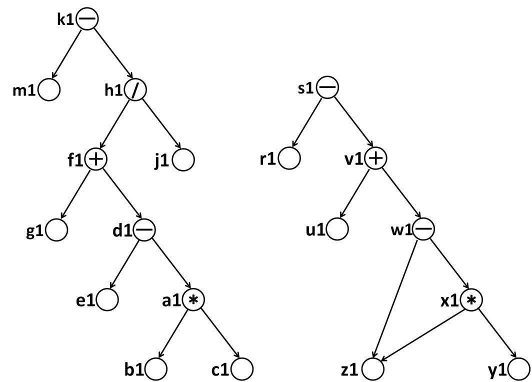


図 4 サンプルプログラムの依存グラフ.

図 2 のサンプルプログラムを依存グラフに変形したものを図 4 に示す．ラベルが空白の節点は，サンプルプログラムに定義が記述されていないものを示す．

4.1 部分依存グラフのグループ化

依存グラフ上で節点の合同を検出する，Alpern，Wegman，Zadeck の手法 (以降，AWZ 法と呼ぶ．)[6] について説明する．ここで，依存グラフ上の 2 つの節点が合同とは，次の 2 つの条件を満たす節点のことである．

(1) 同じ演算子ラベルを持つ．
(2) 辺の先が合同である．

AWZ 法は値の一致性を検出するために，Hopcroft のパーティショニングアルゴリズム [7] を用いる．本研究では，コードクローンを基に依存グラフに変形を加えた上で AWZ 法を適用する．AWZ 法は，合同な節点のグループを，パーティションと呼び，条件を満たす最小のパーティションに分割することをパーティショニングと呼ぶ [6]．パーティショニングは，次の 2 ステップで行う．

ステップ 1： 同じラベルをもつ節点を全て同じパーティションに入れる．

ステップ i+1： i 回目のパーティショニングで同じパーティションに属し，かつ，それらの辺の先が異なるパーティションに属する節点を i+1 回目のパーティショニングで分割する．

最終的に，同じパーティションに含まれる節点は，その節点を根とする部分依存グラフが合同であることを示す．

5. 提案手法

提案手法は，次の 4 つのステップからなる．

ステップ 1： コードクローンを検出する．

ステップ 2： 依存グラフを作成し，コードクローンに基づいて変形する．

ステップ 3： 部分依存グラフをグループ化する．

ステップ 4： 制御フローグラフの各 CFG 節点を訪問し，同じグループに属する計算を括出す．

以降で，各ステップについて詳細に説明する．

ステップ 1：コードクローンの検出

Smith-Waterman アルゴリズムを用いたコードクローンの検出手法 [4] では，プログラム全体からコードクローンを検出するので，ソースコードを入力としていたが，本手法では，分岐の True 部と，False 部の計算列の中から，同じ演算子をもつ計算を検出すれば十分なので，入力を各分岐先の演算子列としている．しかし，プログラム全体からコードクローンを検出することによって，括出し後に更なる括出しの対象を検出することも期待できる．この方法については，現在検討中である．スコアパラメータは第 6 章にて記述する実験で求めた値を使用する．

本手法では，Smith-Waterman アルゴリズムを用いた検出手法 [4] 同様，検出した部分列に対して，LCS アルゴリズム [*4] を適用し，ギャップ部を取り除いた後，一致演算子に対する依存グラフにステップ 2 の変形を適用する．図 5 は，サンプルプログラムから検出したコードクローンを表している．

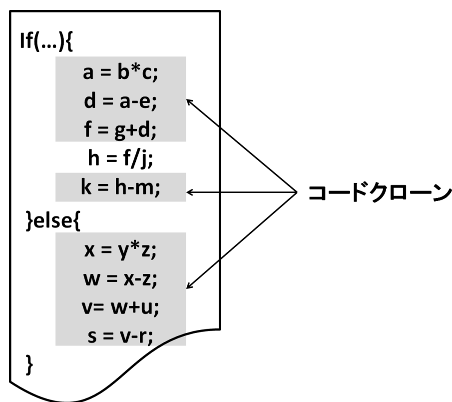


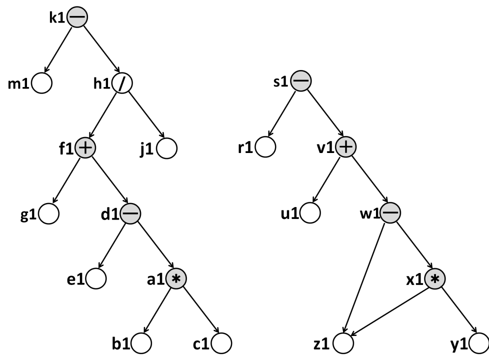
図 5 サンプルプログラムから検出したコードクローン．

ステップ 2：依存グラフの作成・変形

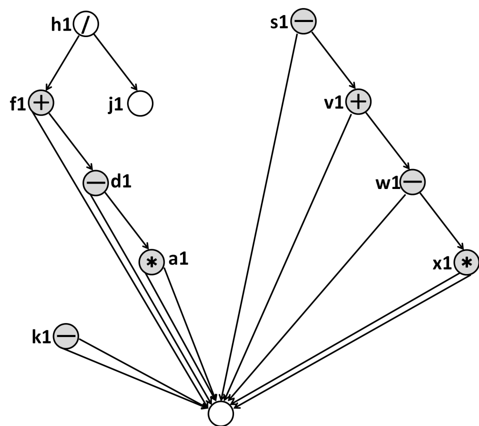
ステップ 3 の部分依存グラフのグループ化を行うために，ステップ 1 で抽出したコードクローン

[*4] Longest Common Subsequence:2 つの列の中から共通の部分列を検出するアルゴリズム．

に基づき，依存グラフを変形する．これは，変形を行わずにグループ化を行うと，たとえ演算子が等しかったとしても，オペランドが異なると同一グループに分類できないからである．変形では，ステップ 1 で検出したコードクローンに含まれる依存グラフの節点から，コードクローンに含まれない節点への依存先を 1 つの仮節点に付け替える．図 6(a) は，サンプルプログラムの依存グラフを示し，図 6(b) は，ステップ 1 で得られたコードクローンを基に (a) を変形したものを表している．ここで，網掛けの節点は，コードクローンとして検出された節点を示している．



(a) 依存グラフ (変形前)



(b) 依存グラフ (変形後)

図 6 依存グラフの変形．

ステップ 3：部分依存グラフのグループ化

変形後のグラフに対し，AWZ 法をそのまま適用する．演算子が一致している類似部分列は，それらが依存関係になくてもコードクローンとして検出されるが，依存グラフ上でグループ化を行うことによって，さらに，同一の依存関係にある計算を抽出する．以下が，ステップ 2 で変形したサンプルプログラムの依存グラフに，AWZ 法を適用した結果である．$k1$，$s1$ は依存している節点が別のグループなので，これら 2 つの変数は同じグループに分類されない．

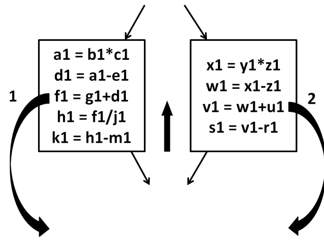
- $(a1, x1)$
- $(d1, w1)$
- $(f1, v1)$
- $(k1)$
- $(s1)$

ステップ 4：制御フローグラフ節点の訪問と，括出し

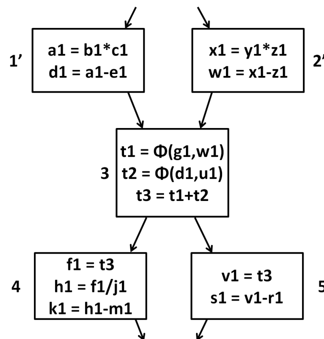
制御フローグラフ上の各節点をトポロジカルソート順序で訪問し，各節点から分岐する 2 つの後続節について，各分岐先の計算を後ろ向きに辿り，ステップ 3 で得られた同一のグループに属する計算を見付ける．括出しの対象が見付かったら，その直後で，各節点を分割し，新しい分岐を挿入した後，一時変数を導入して括り出す．図 7(a) は，サンプルプログラムの各分岐先の命令を後ろ向きに辿る操作を表している．サンプルプログラムでは，初めに同じグループに属する $f1 = g1 + d1$ と $v1 = w1 + u1$ が見付かる．これらを括り出す際，$f1$，$v1$ はそれらの下にある計算 ($h1$ と $s1$ の計算) に依存しているので，対象計算よりもブロック内で下にある計算は新しい分岐先に移動させる．$f1$ と $v1$ の計算を括り出した後の図が図 7(b) である．

分岐先の命令を後ろ向きに辿る操作は，ブロックの入り口に達するまで行う．つまり，括出し可能な計算が複数あれば，さらなる括出しが可能である．複数回の括出しを行う場合も，最初の括出しで挿入した分岐ブロックに括り出す．サンプルプログラムでは，$f1 = g1 + d1$ と $v1 = w1 + u1$ を

括り出した後，同じグループに属する $d1 = a1 - e1$ と $w1 = x1 - z1$ が次に見付かる．これらは，最初の括出しで作成したブロック (図 7(b) ブロック 3) に括り出す．$a1 = b1 * c1$ と $x1 = y1 * z1$ も同様に括り出す．このように提案手法を用いることで，演算子の一致だけで括出しを行った場合よりも多くの計算を括り出すことができる．



(a) 分岐先のブロックを後ろ向きに辿り，同じグループ属する計算 ($f1 = g1 + d1$ と $v1 = w1 + u1$) を発見する．



(b) $f1 = g1 + d1$ と $v1 = w1 + u1$ の括出し結果.

図 7 サンプルプログラムに対する制御フローグラフ節点の訪問と，括出し．

6. 実験と評価

本手法の振舞いを確認するために，プロトタイプを実現し，簡易的なプログラムに対して，2 つの調査項目の評価を行った．

6.1 調査項目

以下の 2 つに関して調査を行う．

調査項目 1： Smith-Waterman アルゴリズムで用いる 3 つのパラメータ $(match, mismatch, gap)$ の適した組み合わせを調べる．

調査項目 2： AWZ 法を用いた類似構造の検出結果を調査する．

6.2 実験対象

評価には，以下の特徴を持った C 言語のプログラムを用いた．

(1) 最大ギャップ率 0.5 の，20 個から 50 個の演算からなる分岐を含む．

(2) ギャップ部以外の計算が依存関係をもつ．

(1) は，調査項目 1 を評価するための条件であり，そこで求まったコードクローンに基づき依存グラフを変形し，依存関係に従って正しくグループ化を行えているかを (2) の条件から評価した．

6.3 評価項目 1

本実験では，以下の 27(= 3 × 3 × 3) 通りのパラメータにいて調査を行った．なお，$\mathbb{Z}$ は整数の集合を表す．

$$match = \{x \in \mathbb{Z} | 1 \leq x \leq 3\}. \tag{1}$$

$$mismatch = \{y \in \mathbb{Z} | -2 \leq y \leq 0\}. \tag{2}$$

$$gap = \{z \in \mathbb{Z} | -2 \leq z \leq 0\}. \tag{3}$$

また，評価は F 値に基づいて行う．F 値は，$Recall$ を再現率, $Precision$ を適合率, $F\!-\!measure$ を F 値とすると，次の式で表される．なお，検出されたコードクローンの集合を S_1，正解のコードクローンを S_2，それらに共通に含まれるコードクローンの集合を S で表している．

$$Recall = \frac{S}{S_2} \tag{4}$$

$$Precision = \frac{S}{S_1} \tag{5}$$

$$F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{6}$$

評価の結果，パラメータが以下の場合，F 値が平均を下回り，それ以外の値では，全て最大値が得られることがわかった．

$$(1,-2,-2)\ ,(1,-1,-2),(1,0,-2),$$
$$(1,-2,-1)\ ,(1,-1,-1),(1,0,-1),$$
$$(2,-2,-2)\ ,(1,-1,-2),(1,0,-2).$$

6.4 評価項目 2

両方の分岐先で一致した演算を持ち，かつ各分岐先でそれら全てが依存関係にある計算を類似構造と呼ぶことにする．また，類似構造に含まれる演算の数を類似構造の大きさと表現すると，評価用プログラムの各分岐先で最大となる類似構造の大きさは 25 である．正しくグループ化が行えているかの判断は，これら 25 個のグループの中に，ギャップ部の計算が含まれていないかを目視確認で行った．ギャップ部の計算は互いに依存しているが，最大の類似構造とは，別の構造をもった計算とした．演算子の一致だけで判断した場合，これらの計算はギャップ部も含めて同じグループになるが，評価の結果，これらの計算は別のグループに属することが確認できた．

7. 結果と考察

本稿では，各分岐先から，ギャップを含む大まかな一致箇所をコードクローンとして検出し，その中から一致した依存構造をもつ類似構造を発見する手法を提案した．簡易的な評価の結果，本手法は，類似構造を正しくグループ化することが確認できた．グループ化の後，同一グループに属する計算が括出しの対象となるが，括出しの対象が複数見つかった場合，それらに優先順位をつける必要がある．現在は，その中で最も大きな構造をもつ計算を括出しの対象とすることを計画している．括出しの対象を決定する方法としては，依存グラフ上に番号付けを行う方法を検討している．また，実際の括出しの方法に関しても，Φ 関数を用いて依存グラフ上で行う方法 [9] などを検討している．

参考文献

[1] 青木尊之，額田彰：はじめての CUDA プログラミング
[2] COINS(A,Compiler,Infra,Stracture):http://coins-compiler.sourceforge.jp/
[3] 中田育男：コンパイラの構成と最適化
[4] 村上寛明，堀田圭佑，肥後芳樹，井垣宏，楠本真二：Smith-Waterman アルゴリズムを利用したギャップを含むコードクローン検出
[5] Smith,T.F.and Waterman,M.S：Identification of common molecular subsequences
[6] Bowen,Alpern,Mark,N.Wegman,F.Kenneth,Zadeck：Detecting Equality of Variables in Programs
[7] J.Hopcroft：An n log n algorithm for minimizing the states of finite automaton
[8] 滝本宗宏，佐々政孝：静的単一形式を用いた最適化 (発展編)
[9] Oliver,Ruthing,Jens,Knoop,Bernhard,Steffen：Detecting Equalities of Variables：Combining Efficiency with Precision
[10] Andrew,W.Appel/著, 神林靖，滝本宗宏/訳：最新コンパイラ構成技法
[11] Saumya,K.Debray,William,Evans,Rovert,Muth,Bjorn,De,Sutter:：Compiler Techniques for Code Compaction
[12] IBM,DeveloperWorks:http://www.ibm.com/developerworks/jp/java/library/j-seqalign/