修士論文

可変長セグメントを用いたフェーズ検出手法

A phase detection technique with variable-length segments

2013年2月6日提出

指導教員　坂井修一 教授

東京大学大学院 情報理工学系研究科

電子情報学専攻

48-116403 阿部 高大

概要

プロセッサの開発にはシミュレーションによる評価が必須である．しかしシミュレーションにかかる時間は膨大であるため，高速化が必要とされる．高速化手法の 1 つであるフェーズ検出を用いることで，プログラムの一部をシミュレーションするだけで全体の結果を推定できる．

既存のフェーズ検出手法である SimPoint では命令列を固定長インターバルに分割し，クラスタリングすることでシミュレーションポイントを推定する．インターバルは固定長で長いため，複数のフェーズが含まれる．よって一般的なクラスタリング手法を用いてまとめる必要があり，クラスタ数が制限されるなどの問題が存在する．

それに対し，我々の研究室では可変長セグメントを導入し，フェーズ検出を行う手法を提案した．可変長セグメントはフェーズの変化に合わせて生成された命令列であるためまとめるのは容易である．そのため SimPoint で発生していた問題は解決できる．しかし今までに提案されていた固定長ユニットを用いて生成された可変長セグメントでは，ユニット長整数倍程度ののセグメントしか生成できなかった．本稿では基本ブロック列に対し一次元のメディアンフィルタを通し，その出力を基に生成された可変長セグメントを用いてフェーズ検出を行う手法について提案する．この手法により最小単位での可変長セグメント生成が可能になり, より高精度のフェーズ検出を行うことができる．

第1章　はじめに

プロセッサの研究・開発には，**シミュレーション**によるプロセッサの性能測定が必要不可欠である．シミュレーションとは，プロセッサの 1 サイクルごとの振る舞いをソフトウェア上で模擬することである．シミュレーションにより，実際にハードウェアを組むことなくプロセッサの性能を測定できる．

しかし，シミュレーションは非常に長い時間がかかるという問題がある．実機に対する実行時間の割合を **SD** (Speed-Down) と呼ぶ．SD は，cycle-accurate なシミュレータでは 1000 以上にもなり，実機で 10 分かかる処理がシミュレーションでは 10,000 分 ≃7 日以上かかることになる．そのため，シミュレーションの高速化に対するニーズは大きい．

シミュレーションの高速化の方法としては，シミュレータ自体の高速化の他に，シミュレーション対象のプログラムの**実行命令数の削減**が考えられる．プログラムには，その**繰り返し構造**に起因して，そこだけを実行すれば全体の振る舞いが推定できるような部分が存在する．そのような部分を**シミュレーション・ポイント**と呼ぶ．シミュレーション・ポイント選択には，いわゆる**フェーズ検出**手法を用いることができる．

SimPoint　シミュレーション・ポイントを選択する代表的な手法として，**SimPoint**[1, 2] が挙げられる．SimPoint は，以下のようにしてフェーズ検出を行う :

1. **インターバルへの分割**　まず，実行された PC の列を固定長の 1M〜100M 命令程度の固定長の**インターバル**に区切る．

2. **基本ブロック・ベクトル生成**　次に，各インターバルに対して，**基本ブロック・ベクトル**を生成する．基本ブロック・ベクトルは各基本ブロック（途中に分岐や合流を含まない命令列, Basic Block, BB）が何回実行されたかを示すベクトルである．基本ブロック・ベクトルのインデックスはプログラム中に存在する基本ブロックの ID に対応する．基本ブロッ

ク・ベクトルは，次元数が数万〜数十万以上の，多次元のスパースなベクトルになる．

3. **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合に**クラスタリング**を施す．SimPoint は，多次元ベクトルのクラスタリング手法として代表的な **k-means 法**を用いている．同一のクラスタに分類されたインターバルがフェーズとみなされる．シミュレーション・ポイントとしては，各クラスタの代表的なインターバルを選択すればよい．

SimPoint は，1M〜100M 命令程度という長い固定長のインターバルを用いているため，その精度に関して，以下のような 2 つの問題がある：

1. インターバルより十分に長いフェーズしか検出できない．
2. 以下で述べるように，内分点の基本ブロック・ベクトルが存在するため，正しいクラスタリングが難しい．

内分点の基本ブロック・ベクトル　図 1.1 上に，固定長インターバルによるフェーズ検出の様子を示す．横軸は命令数で数えた時間で，縦軸は基本ブロックの ID である．同図は，2 種類のループ L_A，L_B が順に実行されている様子を表しており，それぞれがフェーズとして検出されることが期待される．しかし，インターバルは 1M〜100M 命令程度と非常に長いため，インターバル I_2 と I_4 には，L_A と L_B の両方が含まれている．I_2，I_4 の基本ブロック・ベクトルは，L_A，L_B が含まれる割合に応じて，I_1 の基本ブロック・ベクトルと I_3 の基本ブロック・ベクトルの内分点になる．

L_A と L_B が切り替わる度に，このような内分点の基本ブロック・ベクトルが現れる．その結果，図 4.1 左に示すように，基本ブロック・ベクトルは多次元空間に散在することになる．図 1.1 の例では，インターバル I_2 と I_4 は，含まれる L_A と L_B の割合に応じてクラスタに分類されることになる．どちらかに偏っていれば，I_1 や I_3 と同じクラスタに分類されるだろう．偏りが少なければ，I_2 と I_4 からなるクラスタが生成されるかもしれないし，I_2 のみ，I_4 のみからなるクラスタが生成されるかもしれない．このような状態では，クラスタリングが難しいというだけでなく，正解を定義することすら難しい．

可変長セグメントを用いたフェーズ検出手法　固定長インターバルを用いると上記のような問題が起こる．フェーズの切れ目に近い所で基本ブロック列を分割すれば，SimPoint の問題は解決される．そのようにフェーズの切れ目に近い所で分割された基本ブロック列を**セグメント**と呼ぶ．

セグメントを用いたフェーズ検出は以下のように行う：

1. **セグメントへの分割**　フェーズの切れ目を見つけて, セグメントに分割する．
2. **基本ブロック・ベクトル生成**　次に，セグメントの基本ブロック・ベクトルを生成する．基本ブロック・ベクトルは，セグメントに含まれる基本ブロック数で正規化しておく．
3. **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合に対してクラスタリングを施す．

この手法では，前述した固定長インターバルを用いる手法の問題は，以下のように解決される：

1. 　セグメントはフェーズの切れ目に合わせて基本ブロック列を分割するので検出可能なフェーズの大きさに制限がない．
2. セグメントの基本ブロック・ベクトルには内分点が存在しないので．クラスタリングは容易である．

セグメントを生成するために固定長ユニットを用いた手法がある．[3] この手法ではまず基本ブロック列全体を短い固定長のユニットに分割する．次に連続する 2 つのユニットの基本ブロック・ベクトルのマンハッタン距離を計算し，閾値以上の値であれば分割点とすることでフェーズの切れ目を検出していた．しかし，図 1.2 上に示されているように，ユニットの中にフェーズの切れ目があった場合に適切に基本ブロック列を分割できない可能性があった．

そこで本稿では，基本ブロック列をメディアンフィルタに通し，セグメントを生成する．この方法により，図 1.2 下のようにフェーズの切れ目に合わせたセグメントを生成が可能になるため，ユニットで発生する問題を解決できる．

本稿は，以下のように構成されている．続く 2 章でフェーズ検出について述べ，次に 3 章では既存手法である SimPoint についてまとめる．4 章でユニットを用いた手法について説明する．5 章で提案手法に詳しく説明し，5.3 章で評価結果を示す．

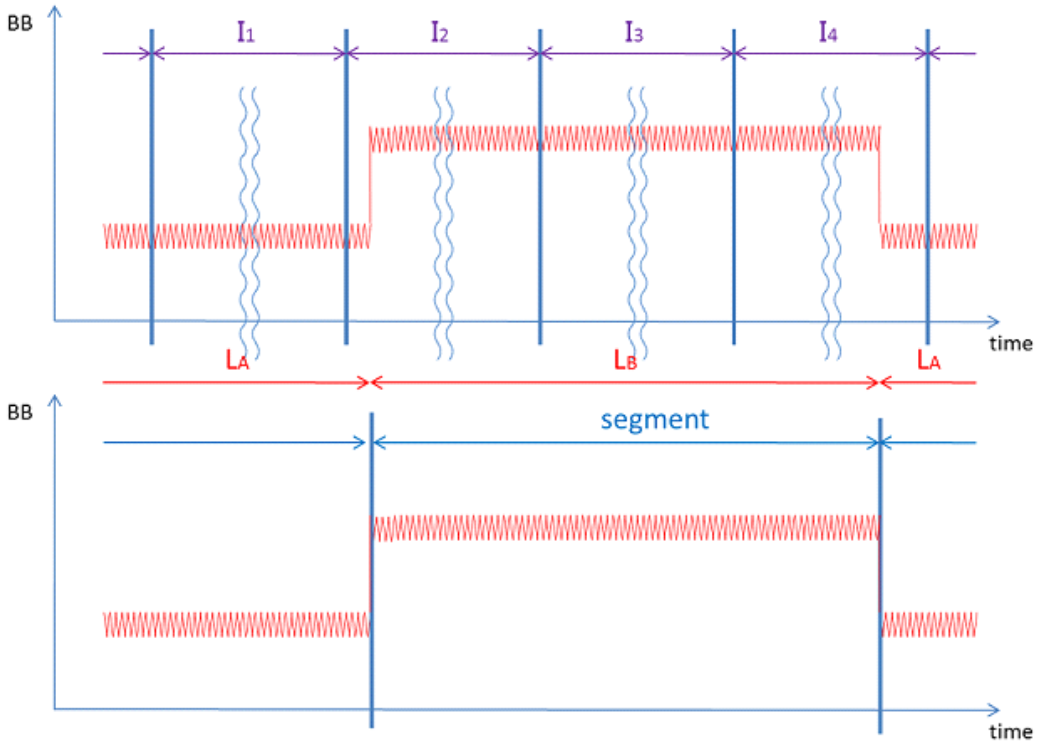

図 1.1: SimPoint（上）と可変長セグメントを用いた（下）のフェーズ検出

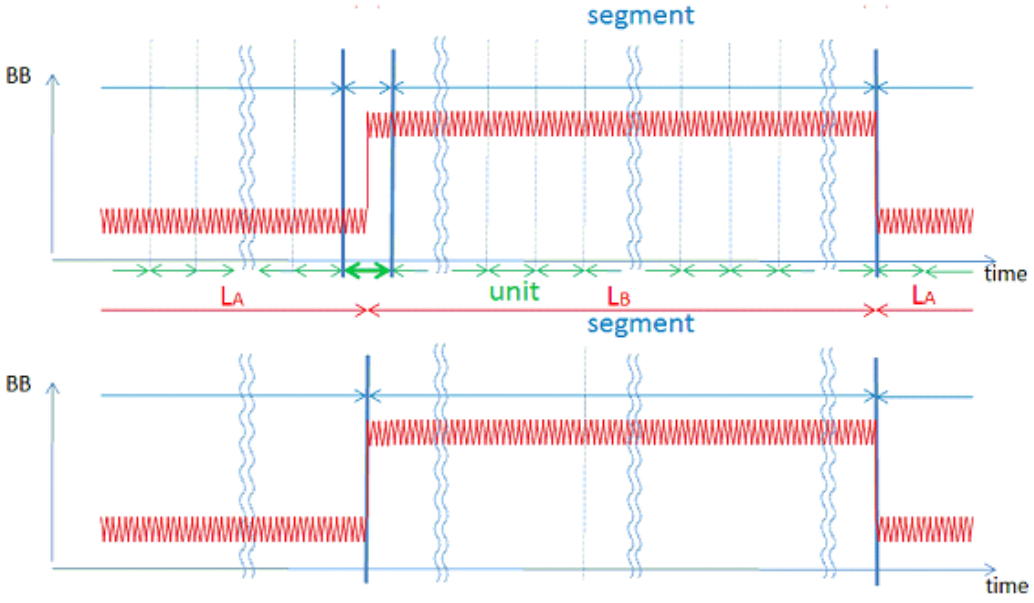

図 1.2: 固定長ユニットを用いた手法（上）と提案手法（下）のフェーズ検出

第2章　フェーズ検出

前述したように，シミュレーション・ポイントを適切に選択することができれば，ターゲット・プログラムのシミュレーションすべき命令の数を大幅に削減することができる．シミュレーション・ポイントの選択には，フェーズ検出手法が用いられる．本章では，一般的なフェーズ検出の手法とその評価方法について述べる．

2.1　シミュレーション・ポイントとフェーズ

シミュレーション・ポイントとは，プログラム実行の一部で，そこだけシミュレーションすれば全体のプロセッサの振る舞いが推定できる部分のことである．シミュレーション・ポイントが存在するのは，プログラムには繰り返し構造が存在するからである．

このプログラムの繰り返し構造により，プログラムの実行は，そこを実行するプロセッサの振る舞いが似ている部分に分割できる．本稿では，曖昧さを避けるため，この各部分のことを**フェーズ**，プロセッサの振る舞いが互いに似ているフェーズの集合を**クラスタ**と定義する[1]．この定義によれば，フェーズ検出とは，プログラムをフェーズに分け，フェーズをクラスタに分類することとなる．

2.2　シミュレーション･ポイントのためのフェーズ検出

フェーズ検出はPCに着目して行うことが一般的である．その場合，PCの列を**基本ブロック** (basic block) の列に変換することで扱うデータの量を大幅に削減することができる．基本ブロック1つあたりの平均命令数は10程度で

[1]既存の論文では，クラスタもフェーズと呼ばれている [4, 5].

あるので，PC 列を基本ブロック列に変換することで情報量を減らさずに扱う列のデータサイズを 1/10 程度に減らすことができる．

2.3 フェーズ検出手法の評価方法

フェーズ検出手法の評価の手順は以下の通りである：

1. あらかじめプログラム全体を実行し，プロセッサの評価値（CPI など）を得ておく．
2. **エミュレーション**（後述）により基本ブロック列を得る．
3. 基本ブロック列に対してフェーズ検出を行い，シミュレーション・ポイントを選択する．
4. シミュレーション・ポイントのシミュレーションを行い，その区間の評価値を得る．
5. 重みづけ平均により，プログラム全体の評価値を取得し，あらかじめ取得しておいた評価値との誤差を測る．

なおエミュレーションとは，プロセッサの内部状態までは再現せずに，プログラム実行の出力が実機の出力と同じになるようにソフトウェア上で模擬することである．

最終的に取得した誤差の値，およびシミュレーション・ポイントがプログラム全体に対し占める割合が小さいほど，良いフェーズ検出手法だと言える．

2.4 クラスタリング手法

フェーズ分類の際にはデータの量が膨大となるため，計算量をできるだけ少なくする必要がある．

クラスタリングの手法には，階層的な手法と非階層的な手法がある．階層的手法とは，似ているデータを階層的にまとめていきクラスタを作る手法である．非階層的手法とは，データをランダムにクラスタに割り振り結果的に似たものが同じグループに入るようにする方法である．計算量は，階層的手法の場合 $O(N^3)$，非階層的手法の場合は $O(N)$ であることが多い．

非階層的手法の代表的なものとして，SimPoint で採用されている k-means 法 [6] がある．k-means 法については，3.2 節で詳しく述べる．

第3章　SimPoint

シミュレーション・ポイント選択のためのフェーズ検出手法としては，SimPoint[7, 8] が代表的である．

本章では SimPoint について述べる．

3.1　SimPoint の概要

1 章で述べたように，SimPoint は，プログラムの実行を 1M〜100M 命令の固定長のインターバルに分割し，それらの基本ブロック・ベクトルをクラスタリングすることによりフェーズを検出している．

SimPoint は，以下のようにしてフェーズ検出を行う：

1. **インターバルへの分割**　まず，PC の列を 1M〜100M 命令程度の固定長のインターバルに区切る．インターバルの長さは，計算量と精度のトレードオフによって決める．

2. **基本ブロック・ベクトル生成**　次に，各インターバルに対して，基本ブロック・ベクトルを生成する．

3. **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合にクラスタリングを施す．同一のクラスタに分類されたインターバルがフェーズとみなされる．

3.2　K-means 法

基本ブロック・ベクトルの各次元はプログラム中に存在する基本ブロックに対応し，各次元の値はそのインターバル内でその基本ブロックが実行された回数を表す．したがって基本ブロック・ベクトルは，次元数が数万〜数十万以上の，多次元のスパースなベクトルになる．

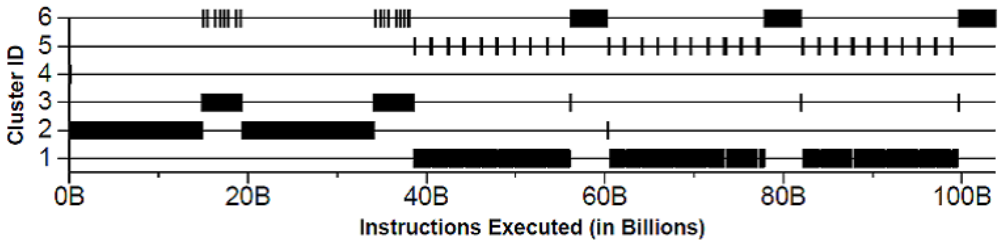

図 3.1: SimPoint による gzip のフェーズ分類

SimPoint は基本ブロック・ベクトルのクラスタリングのために，多次元ベクトルのクラスタリング手法として代表的な K-means 法を用いている．

K-means 法のアルゴリズムは，以下のとおりである：

1. 各データをランダムに k 個のクラスタに分類する．
2. 各クラスタの平均値を計算する．
3. 各データがどのクラスタの平均値に近いか計算し，最も近いクラスタに分類し直す．
4. (2) と (3) を繰り返す．データの移動がなくなった時点で終了する．

K-means 法では，k の値を予め決める必要があり，K-means 法自体によっては最適な k の値は分からない．したがって，さまざまな k を用いて K-means 法を実行し，最適な k を選択するという方法を採らざるを得ない．

図 3.1 に，SPEC2000 の gzip に対してクラスタリングを施した結果を示す [1]．同図からは，gzip は大きく 6 つにクラスタリングされることが分かる．

SimPoint は，シミュレーション・ポイントとして，各クラスタの平均値に最も近いインターバルの基本ブロック・ベクトルを選ぶ．

3.3 SimPoint の問題点

1 章で述べたように，SimPoint の問題点は，命令列を固定長インターバルで分割していることに起因する．SimPoint は，1M〜100M 命令程度という長い固定長のインターバルを用いているため，その精度に関して，以下のような 2 つの問題がある：

1. インターバルより十分に長いフェーズしか検出できない.

2. 内分点の基本ブロック・ベクトルが存在するため，正しいクラスタリングが難しい.

K-means 法のような一般的なクラスタリング手法を用いるのは，内分点の基本ブロック・ベクトルが多数存在するためであると言える.

第4章　固定長インターバルを用いないフェーズ検出手法

SimPoint の問題点は固定長の基本ブロック列に分割していることであった．その問題を解決するために基本ブロック 100 個程度を最小単位とする可変長のセグメントを用いる手法が提案された．

本章では，固定長ユニットを用いて可変長セグメントを生成しフェーズ検出を行う手法について述べる．以下，4.1 節でユニットについて述べた後，4.2 節でユニットを用いたセグメントへの生成について，4.3 節でクラスタリングについて，それぞれ説明する．そして 4.4 節で SimPoint との比較とこの手法の問題点について述べる．

4.1　ユニット

ユニットは固定長の基本ブロック列である．インターバルと比べ十分に小さく，基本ブロック 100 個程度とされる．次節で述べるセグメントへの分割を行うために導入されたものである．

4.2　ユニットを用いたセグメントの生成

まず基本ブロック列全体を固定長の基本ブロック列であるユニットに分割する．次に，連続する 2 つのユニットの基本ブロック・ベクトルのマンハッタン距離を計算し，閾値以上の値であれば分割点とする．マンハッタン距離は 2 つのベクトルの各座標の差の絶対値の総和と定義される．分割点毎に区切った基本ブロック列をセグメントとする．

フェーズの切れ目ではその前後のユニットの基本ブロック・ベクトルの距離が大きな値となるので，分割点はフェーズの切れ目であるための十分条件と言える．ただしフェーズの切れ目でなくても基本ブロック・ベクトルの距離が大

きくなることはあり得るため，分割点毎に区切った基本ブロック列をセグメントとし，フェーズと区別している．

以上のようにしてこの手法では基本ブロック列を可変長であるセグメントに分割をしている．

4.3 クラスタリング

次に分割したセグメントに対しクラスタリングを行う．セグメントの長さはセグメント毎に異なるので，クラスタリングするためにはセグメントの基本ブロック・ベクトルを正規化する．その結果，例えば同じ命令が 100 回繰り返されるループと 500 回繰り返されるループは同じクラスに分類される．

また可変長のセグメントに分類したことはクラスタリング手法にも影響する．インターバルの基本ブロック・ベクトルはどのフェーズにも属さない基本ブロック・ベクトルが存在するため，基本ブロック・ベクトルの分散が大きくなる．一方セグメントの基本ブロック・ベクトルはセグメント毎に類似度の高いものとなり，分散が小さいことが期待できる．(図 4.1)

よって可変長セグメントを用いる手法では以下のような単純なアルゴリズムを用いてクラスタリングを行う．

1. 各セグメントの基本ブロック・ベクトルと全てのクラスタの中心との距離を比較
2. 距離が閾値以内のクラスタがあれば，最も距離の短いクラスタに分類し，中心を再計算
3. 閾値以内のクラスタがなければ新しいクラスタを作成
4. 次のセグメントに対して，(1) から (3) を繰り返す

このクラスタリング方法では，閾値の大小によってクラスタ数が決まる．したがって，k-means 法のように何回も実行して最適なクラスタ数を求める必要はない．

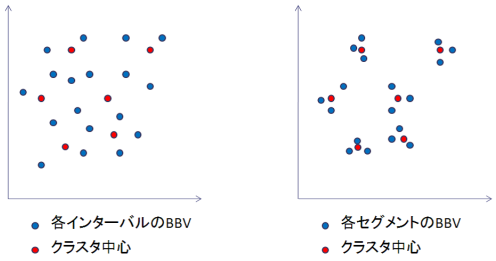

図 4.1: 基本ブロック・ベクトルの分散

4.4 SimPoint との比較と問題点

この手法では，前述した固定長インターバルを用いる手法より以下の点で優れている：

1. ユニットは基本ブロック 100 個程度と小さく，その程度のフェーズを検出することができる．1M〜100M 命令程度のインターバルに比べて，基本ブロック 100 個程度のユニットの大きさは 1/1, 000〜1/100, 000 に過ぎない．

2. ユニット自体は固定長であるので，内分点的なユニットが存在することは避けられない．しかし，基本ブロック・ベクトルは，ユニットごとにではなくセグメントごとに計算されるので，セグメントに含まれるユニットの数が十分多ければ，その影響は無視できる．

しかし，SimPoint とこの手法を比較すると優れていることがわかるが，問題点もある．上にも書いたように, ユニットは固定長であるため，内分点的なユニットが存在することは避けられない．そのため，セグメントに含まれるユニットの数が少ないと内分点の影響が大きくなる．

第5章　提案手法:可変長セグメントを用いたフェーズ検出

4 章では固定長ユニットから生成した可変長セグメントを用いる手法についてとその問題点を説明した．その問題を解決するために，提案手法ではメディアンフィルタを用いてセグメントを生成し，フェーズを検出する．

本章では，5.1 節で提案手法におけるセグメント生成について，5.2 節でクラスタリングについて述べる．最後に 5.3 節でユニットを用いた手法との比較を行う．

5.1　提案手法における可変長セグメントの生成

提案手法におけるセグメント生成にはメディアンフィルタを用いる．メディアンフィルタは、n × n の局所領域における値を並び替え、 その中央の値を領域の中心に対し出力する．

本手法では，一次元のメディアンフィルタを用いる．基本ブロック ID を入力して，出力を元の基本ブロック列の中心と関連付けをする．以下メディアンフィルタを用いたセグメントの生成方法の流れを述べる：

1. 基本ブロック列全体に対しフィルタを 1 基本ブロックずつスライドさせ，基本ブロック ID を入力する．

2. フィルタから出力された基本ブロック ID の値を元の基本ブロック列の中心と関連付ける．

3. 全ての関連付けが終了後，隣り合う出力値の差の絶対値が指定された閾値以上なら基本ブロック列を分割する

4. 分割された基本ブロック列をセグメントとして扱う

メディアンフィルタを用いたセグメント生成についての具体例を図 5.1 を用いて説明する．図ではそれぞれ縦軸に基本ブロック ID を取り，横軸は各基本ブロックが先頭から何番目のものであるかを表す．上の図は元の基本ブロック列に対しフィルタがスライドしている様子を表しており，下の図はフィルタからの出力された値の関連付けが終了した後の様子を表している．今回の例ではフィルタのサイズを 5 とし，閾値を 50 とする．

上の図で，黄緑色のフィルタへの入力は 5〜9 の位置にある基本ブロック列の ID(100,.101,102,101,100) である．また 1 基本ブロック分スライドした緑色のフィルタへの入力は 6〜10 の位置にある基本ブロック列の ID(101,102,101,100,1) である．黄緑色のフィルタからの出力は 101 であり，7 の位置にある基本ブロックとの関連付けを行う．また緑フィルタからの出力も 101 であり 8 の位置にある基本ブロックとの関連付けを行う．下の図で 9 の位置にある基本ブロックへ関連付けされた値である 100 と，10 の位置にある基本ブロックへ関連付けされた値である 3 の差の絶対値は 97 であり，閾値を超えるので基本ブロック列の分割を行う．同様に 18 から 19 の位置でも基本ブロック列を分割し，10〜18 までの基本ブロック列を一つのセグメントとして扱う．

以上のような方法を用いると 1 基本ブロックごとにフェーズの切れ目があるかどうかを確認できるため，生成されるセグメントには基本ブロック・ベクトルの内分点が存在しない．また，図の 14 の位置にあるような小さな基本ブロック ID の変化を受けることなくフェーズの切れ目を検出することができる．

5.2 クラスタリング

提案手法におけるクラスタリングは 4.3 節で説明したものと同様の方法を用いている．前節に述べたようにセグメントの基本ブロック・ベクトル内分点が存在しないのでより類似度の高いセグメントが同じクラスタにまとまりやすくなる．

5.3 固定長ユニットを用いた手法と提案手法におけるセグメントの比較

4.4 節で述べたように，ユニットを用いた手法では生成されたセグメントの基本ブロック・ベクトルに内分点が存在することは避けられないという問題

があった．提案手法ではメディアンフィルタを基本ブロック列に対し 1 基本ブロックずつスライドさせ，その出力値を用いてセグメントを生成する．基本ブロック列を固定長に区切ることをしないため，結果図 1.2 下のようなセグメントが生成される．このようにして生成されたセグメントの基本ブロック・ベクトルには内分点が存在しない．よってユニットを用いた場合の問題を解決することができた．

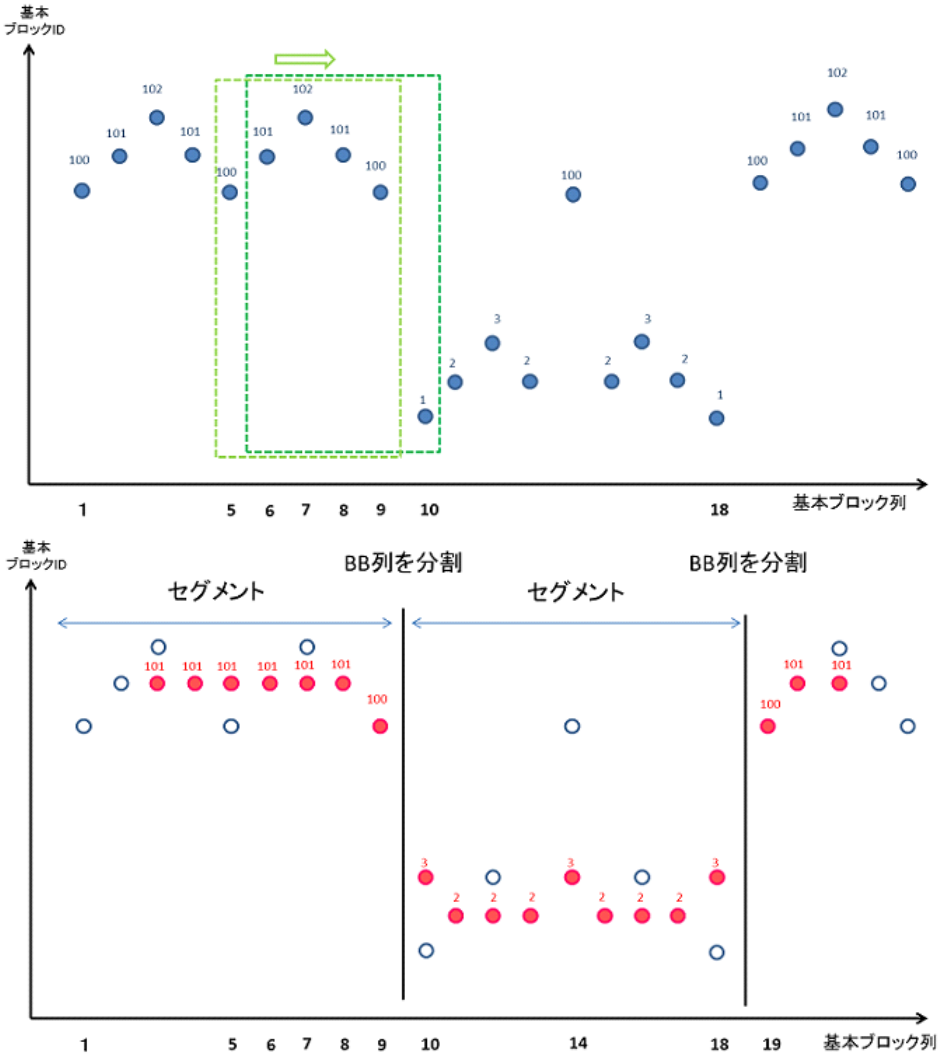

図 5.1: メディアンフィルタを用いたセグメント生成の具体例

第6章　評価

6.1　クラスタリング結果

SPECCPU2006 の soplex の基本ブロック列とそれぞれの基本ブロック列を実行している時の CPI を図 6.1 に示す．同図は横軸に基本ブロック数，縦軸に基本ブロックの番号を表すグラフである．横軸，縦軸とも基本ブロック列をそのままプロットすると膨大な量になるため，1000*1200 のブロックに圧縮している．そして，圧縮されたブロックに含まれる基本ブロックの数を色で表している．また，CPI は黄色の点でプロットされている．同様の方法で libquantum，perlbench の BB 列を図 6.6 と図 6.9 にそれぞれ示す．

soplex に対してユニットを用いた手法と提案手法のクラスタリング結果を図 6.2 と図 6.3 に示す．表示するための方法と内容は図 6.1 と同様であるが，表示しているクラスタリング結果のカラーマップはクラスタ毎に所属している基本ブロック数が多い順に，基本ブロック列を並び替えて表示されている．また，所属しているクラスタが変わるたびに黄色の線が表示されている．よって，黄色の線で挟まれた間のカラーマップがより似通った形状，色を示していればクラスタごとに類似度の高い基本ブロック列をクラスタリングできていると言える．さらに，黄色い線で囲まれた間にプロットされた CPI が一定の値を示していれば適切なフェーズ検出ができていると言える．

両手法とも，soplex と libquantum についてはカラーマップで確認できる範囲においてはクラスタ内でより類似度の高い基本ブロック列が集まっていることが確認できる．perlbench については両手法とも各クラスタごとの基本ブロック列が非常に少ない状態になっている．perlbench の test 入力は 17M 命令ほどの非常に短いプログラムであり，そもそも実行全体において存在するフェーズが少ないということが理由として考えられる．

そして，soplex，libquntum について CPI のプロットを確認すると各手法ともプロットされた CPI が一定の値を示しているクラスタと，一定の値を示しておらずクラスタ内で CPI がばらついているものが混在していることが確認

できる．より詳しい情報を得るために，soplex をクラスタリングした際のあるクラスタについてのカラーマップと CPI の表示を行った．それぞれの手法について所属している基本ブロック数が多いかつ，CPI がばらついているクラスタとして図 6.2 と図 6.3 から一番左にあるクラスタ，左から二番めにあるクラスタのカラーマップをそれぞれ図 6.4，図 6.5 に示す．同図における黄色の線はセグメントが変化するたびに引かれている．

ともに全体を表示した図では CPI が分散しているように見えたが，提案手法の図においてはセグメントごとにプロットされている CPI の変化が同じ形をしていることが確認できた．

両者は同じフェーズについて表示しているわけではなく，また全てのクラスタについての CPI のプロットを示したわけではない．しかし，それぞれの手法において所属している基本ブロック数が多いと判断されたクラスタであるため，手法としてフェーズの切れ目に合わせてセグメントを生成できているのかの判断材料になるのではないかと考えられる．

6.2 CPI 推定

6.2.1 評価環境と評価した手法のパラメータ

次節で CPI 推定の結果を示すために，この節では評価を行った環境と評価した手法のパラメータについて説明する．

プロセッサ・シミュレータ「鬼斬弐」を用いてシミュレーション・ポイントを実行し，CPI 推定を行った．評価したプロセッサの基本的なパラメータは表 6.1 の通りである．

評価は，SimPoint とユニットを用いた手法，提案手法についてそれぞれ行った．

評価対象として SPECCPU2006 の 400.perlbench，450.soplex，462.libquantum の 3 本のベンチマーク・プログラムを用いた．入力データ・セットには *test* を用いた．SimPoint ではのインターバルの長さを 1M 命令とした．

ユニットを用いた手法に関しては，ユニット長を 100BB とし，それぞれのベンチマークに対するセグメントの判定閾値を表 6.2 に示す．またクラスタの閾値を 1，0.8，0.6，0.4 と変化させた場合のデータを計測した．

提案手法におけるパラメータを表 6.3 に示す．また，クラスタリングの閾値を 1，0.8，0.6，0.4，と変化させた場合のデータを計測した．

表 6.1: Configuration of Processor

ISA	Alpha21164A
pipeline stages	Fetch:3,Rename:2,Dispatch:2,Issue:4
fetch width	4 inst.
issue width	Int:2，FP:2，Mem:2
instruction window	Int:32,FP:16, Mem:16
branch predictor	8KB g-share
BTB	2K entries,4way
RAS	8 entries
L1C	32KB,4way,3cycles,64B/line
L2C	4MB,8way,10cycles,64B/line
main memory	200cycles

6.2.2 評価結果

評価結果を図 6.12 に示す．グラフの横軸にシミュレーション全命令に対してシミュレーションポイントの占める割合を，縦軸は推定された CPI とベンチマーク全体をシミュレーションした時の CPI の相対誤差を示している．色でベンチマークの種類を，プロットの形で手法の種類を分類している．また，参考データとして先頭から一定区間シミュレーションし，それぞれの割合まで実行した時の CPI と全体をシミュレーションした時の CPI の相対誤差をプロットしている．

一般に，シミュレーションの実行割合と，CPI 誤差はトレードオフであり，どちらとも数値が小さくなるようなプロットが出来ればよい手法だと言える．図 6.12 から解るように，soplex と perlbench に関しては提案手法を用いると他のどの手法よりも原点に近い点が存在する．これは提案手法を用いると CPI 誤差が小さいかつ, シミュレーションの実行割合が少なくなるシミュレーションポイントを選択できていることを表している．参考データにおいて，libquantum については数%の実行で相対誤差が 0 になる点が得られた．しかし，その他のベンチマークについては提案手法において選択されたシミューレーションポイントの方がよりよい選択であることが確認できる．これより，先頭から一定区間を適当にシミュレーションするより，フェーズ検出手法を用いてシミュレーションポイントを選択した方がより精度のよいシミューレーションが行えると

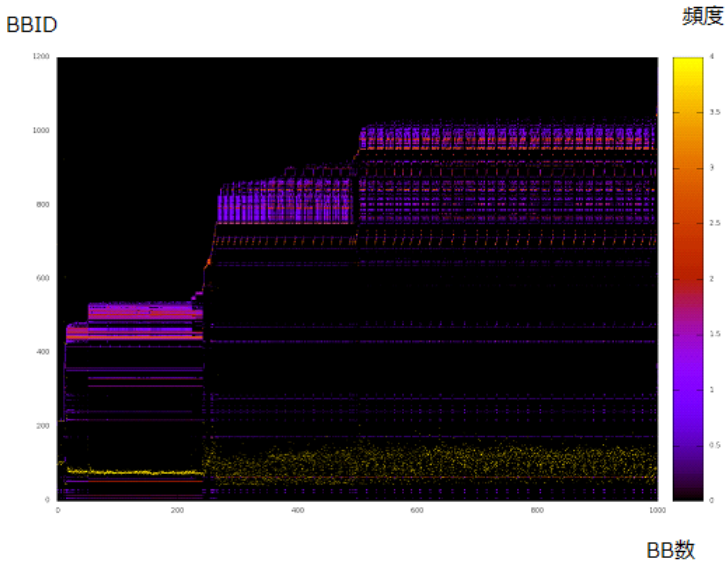

図 6.1: soplex の基本ブロック列と CPI

表 6.2: ユニットを用いた手法に関するパラメータ

benchmark	セグメント判定閾値
libquantum	100
perlbench	60
sopelx	40

言える.

また，libqunatum ではユニットを用いた手法と提案手法はほぼ同程度の結果が得られた.

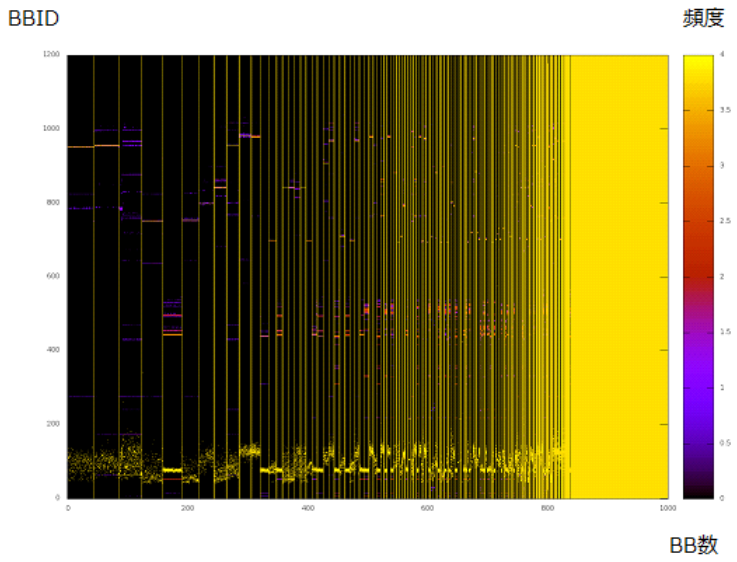

図 6.2: soplex におけるユニットを用いた手法のクラスタリング結果と CPI

表 6.3:　提案手法に関するパラメータ

benchmark	フィルタサイズ	セグメント判定閾値
libquantum	256	16
perlbench	256	16
sopelx	64	128

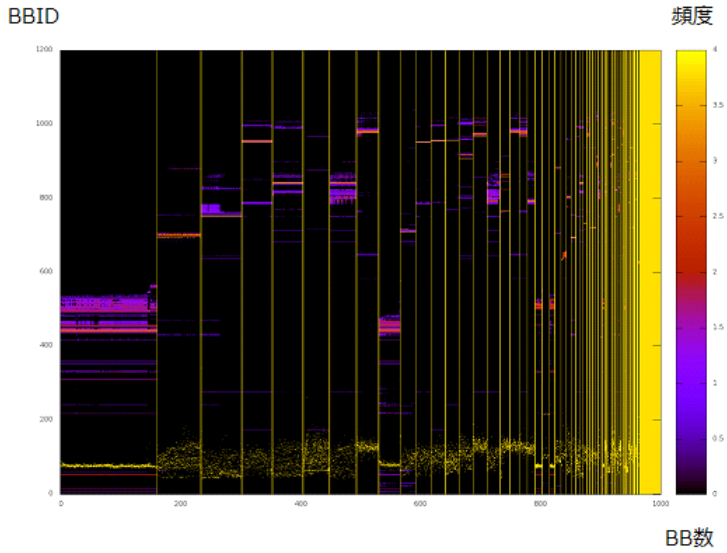

図 6.3: soplex における提案手法のクラスタリング結果と CPI

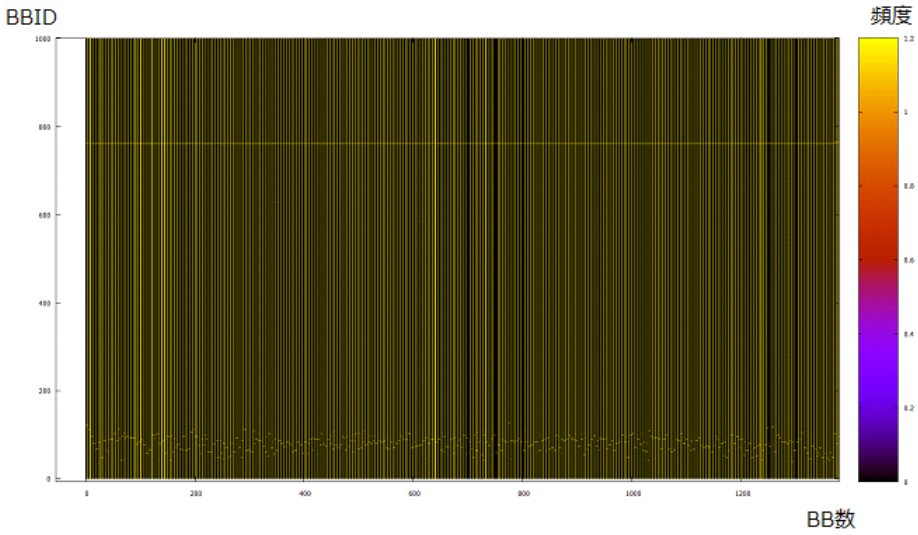

図 6.4: ユニットを使った手法を用いて soplex のクラスタリングした際のクラスタ 1 の基本ブロック列と CPI

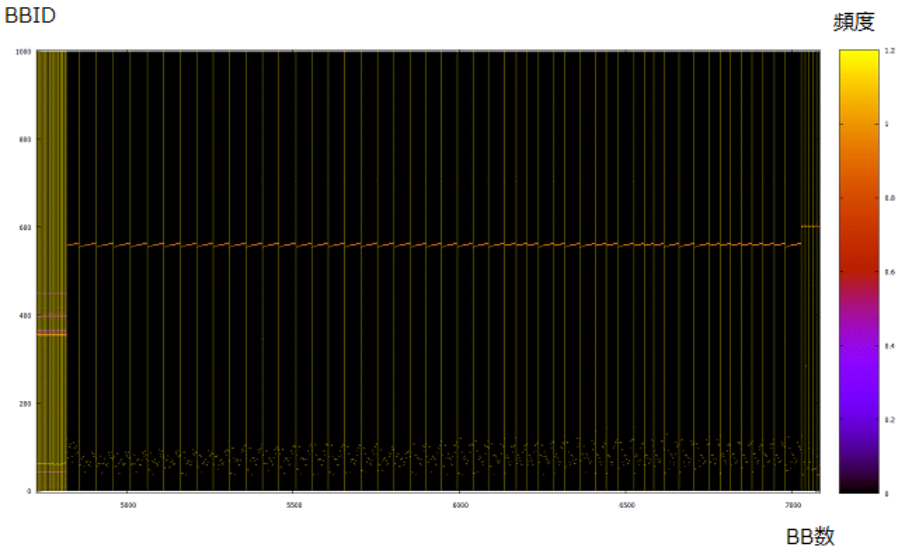

図 6.5: 提案手法を用いて soplex のクラスタリングした際のクラスタ 2 の基本ブロック列と CPI

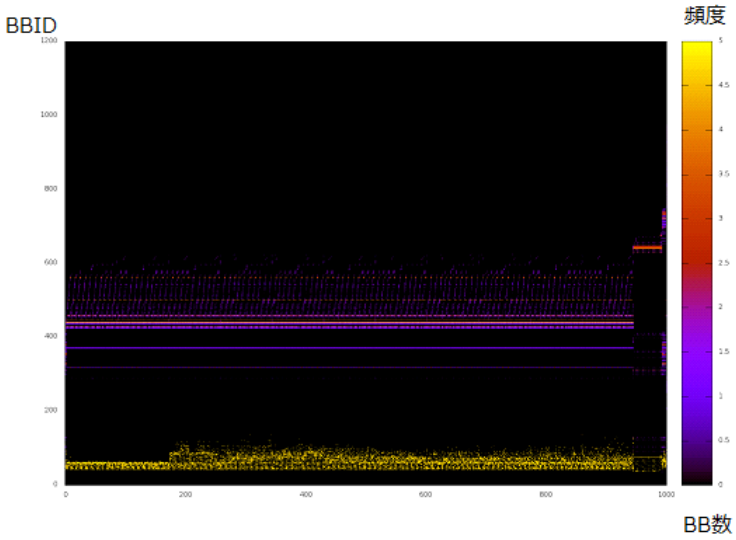

図 6.6: libquantum の基本ブロック列と CPI

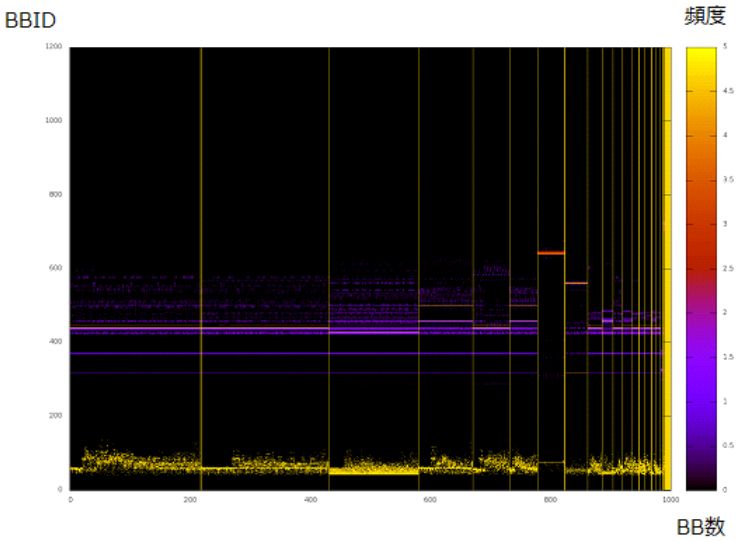

図 6.7: libquantum におけるユニットを用いた手法のクラスタリング結果と CPI

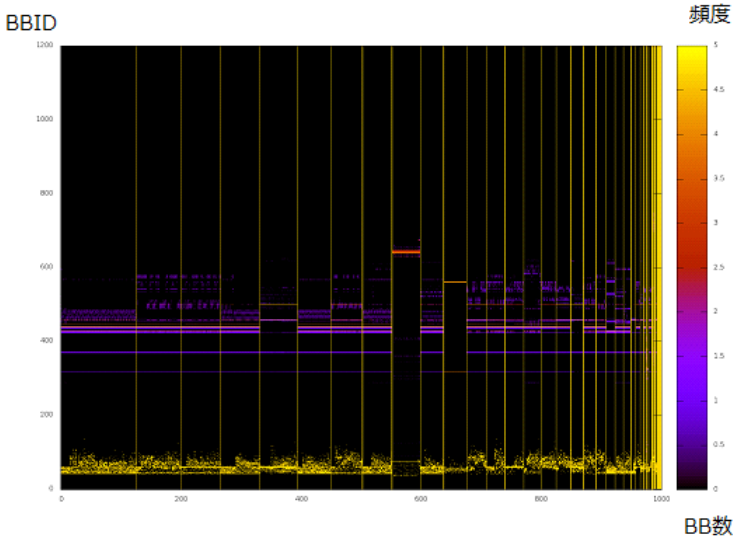

図 6.8: libquantum における提案手法のクラスタリング結果と CPI

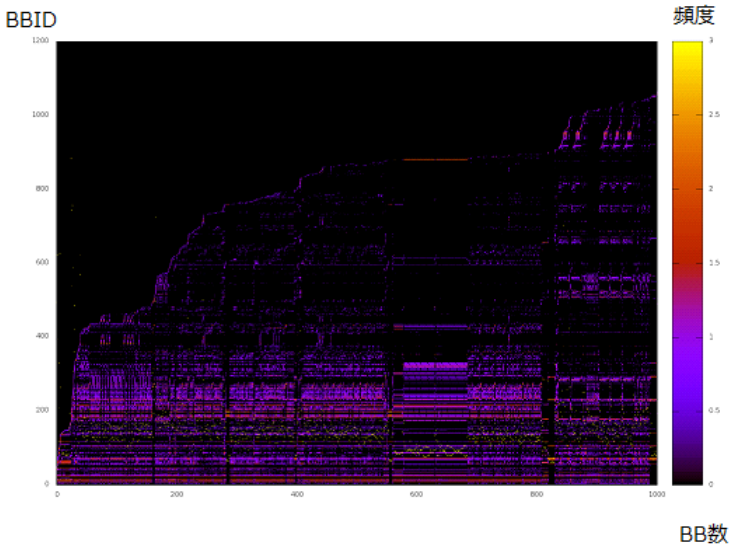

図 6.9: perlbench の基本ブロック列と CPI

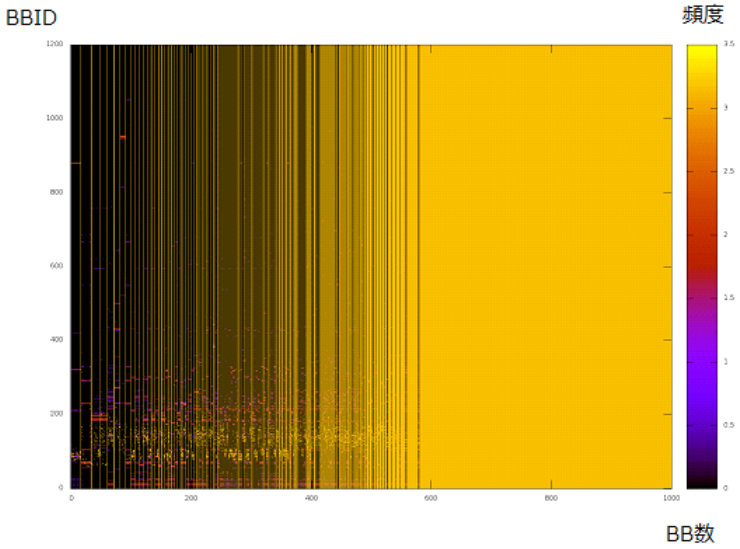

図 6.10: perlbench におけるユニットを用いた手法のクラスタリング結果と CPI

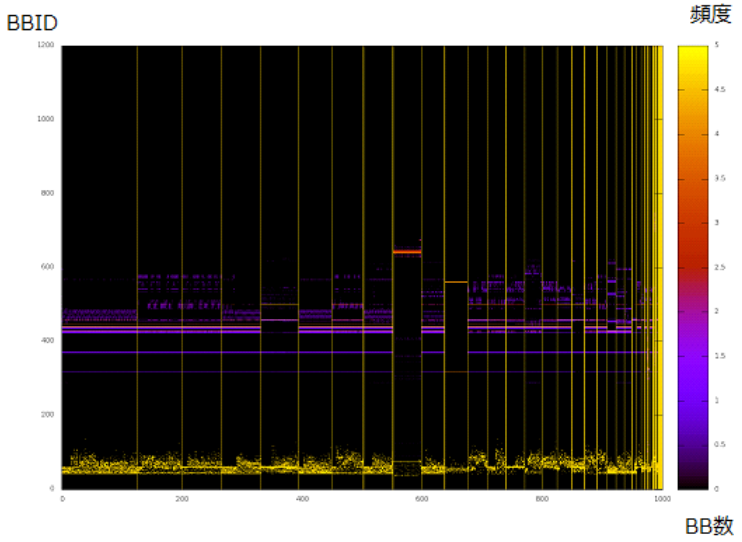

図 6.11: perlbench における提案手法のクラスタリング結果と CPI

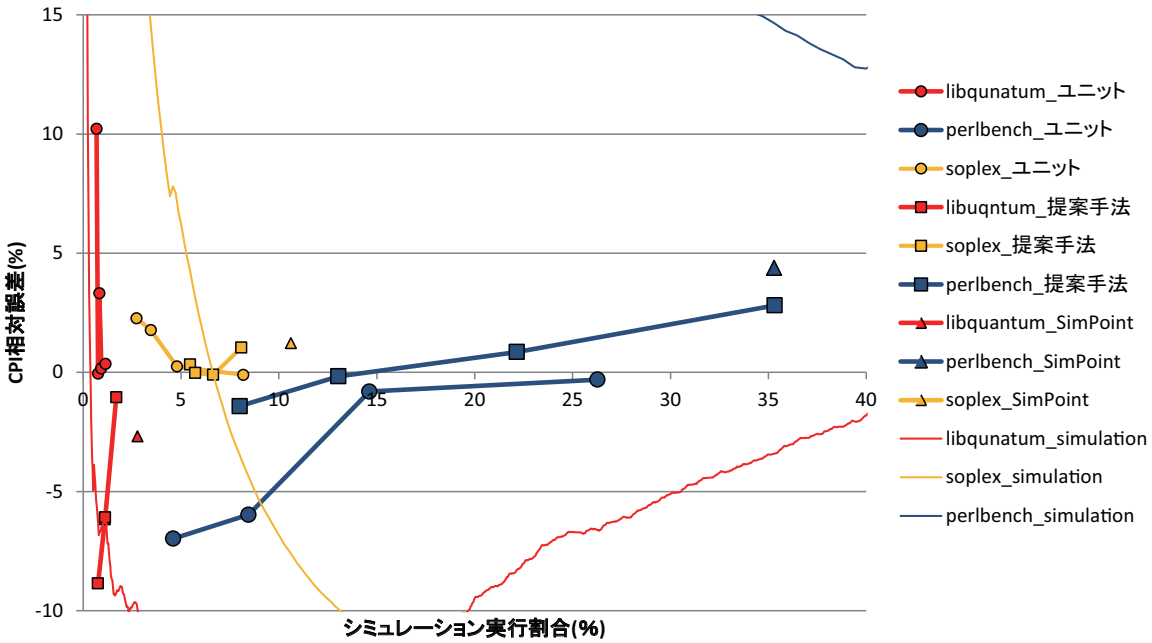

図 6.12: SPECCPU2006 における CPI 推定とシミュレーション実行割合

第7章　終わりに

本稿では一次元のメディアンフィルタを基本ブロック列に対しスライドさせ，その出力結果を用いることで生成された可変長セグメントを用いるフェーズ検出手法を提案した．提案手法を用いて，SPECCPU2006 ベンチマークから libquantum,perlbench,soplex の test 入力のシミューレーションポイントを選択し CPI 誤差と実行割合を評価した．その結果，それぞれのベンチマークについて CPI の相対誤差が 1% 以下の精度を持つシミュレーションポイントを選択できることを確認した．今回は 3 つのベンチマークの test 入力のみで評価を行ったが，今後は他のベンチマークや，実際のシミュレーションで使われる ref 入力などのより長いプログラムを用いた場合の性能評価を行う必要がある．

現在のクラスタリングの方法ではシミュレーションポイントの選択の際にセグメントの長さを考慮していない．この事により，可変長であるセグメントを用いるとシミュレーションポイントとして短かすぎるまたは長すぎる命令列を選択する可能性がある．シミュレーションポイントが短すぎると実行された命令列内でキャッシュミスなどが発生した場合に CPI 推定に大きな影響を与える可能性がある．クラスタリングの閾値を低くした時に誤差が大きくなってしまったのはより多くのクラスタを作ってしまい，短いセグメントをシミュレーションポイントとして選択する割合が高まり誤差が生じたからだと考えられる．長いシミュレーションポイントを選択すると実行命令列が削減できない可能性がある．よって今後はシミュレーションポイントとして選択するセグメントの長さも考慮に入れる必要があると考えられる．

さらに，現手法ではセグメントを生成する際，適切なシミューレーションポイントを選択するためにメディアンフィルタのサイズやセグメント判定距離をベンチマークごとにチューニングしている．今後は多くの数のプログラムで適切なシミュレーションポイントを検出するための共通のパラメータを発見することや，またはフェーズに合わせて自動でパラメータを設定するような方法を考える必要があると考えられる．

参考文献

[1] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. Discovering and exploiting program phases. In *ISCA*, 2002.

[2] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Int'l Conf. on Parallel Architectures and Compilation Techniques*, 2001.

[3] 赤松雄一, 五島正裕, 坂井修一. 固定長インターバルを用いないフェーズ検出手法. 先進的計算基盤システムシンポジウム SACSIS2011, pp. 271–278, 2011.

[4] T. Sherwood and B. Calder. Time varying behavior of programs. Technical Report UCSD-CS99-630, UC San Diego, 1999.

[5] M. Hind, V. Rajan, and P. F. Sweeney. Phase detection: A problem classification. Technical Report 22887, IBM Research, 2003.

[6] G. Hamerly and C. Elkan. Learning the k in k-means. Technical Report CS2002-0716, University of California, 2002.

[7] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. Using simpoint for accurate and efficient simulation. In *SIGMETRICS*, 2003.

[8] G. Hamerly, E. Perelman, and B. Calder. How to use simpoint to pick simulation points. *ACM SIGMETRICS Performance Evaluation Review*, 2004.

発表文献

主著論文

1. 過去の競合命令にチェックポイントを設定するトランザクショナル・メモリ
   阿部 高大，倉田 成己，塩谷 亮太，五島 正裕，坂井 修一
   情報処理学会研究報告 2012-ARC-01(2012).

2. 可変長セグメントを用いたフェーズ検出手法
   阿部高大，早川薫，倉田成己，五島正裕，坂井修一
   先進的計算基盤システムシンポジウム SACSIS 2013 （投稿中)

謝辞

本研究を進めるにあたり，本当にたくさんの方々にお世話になりました．

指導教員である坂井先生には相談会などにおいて多くのご指導をいただきました．また五島先生には研究の進め方だけには限らず，あらゆる面においてご助言を頂きました．本当にありがとうございました．

倉田成己氏には相談会を始めとし研究に関する様々な点でご指導やご助言を頂きお世話になりました．

八木原晴水さん，長谷部環さんには，研究室における設備の導入や各種事務手続きなど，研究室で過ごすための様々なご支援を頂きました．

また，同期の方々には精神面で大変お世話になりました．皆のお陰で研究室生活が楽しいものになりました．その他多くの研究室メンバーにも研究活動において様々な面でご協力いただきました．本当に，本当に皆様ありがとうございました．