# メモ化の適用を効率化するソフトウェアアーキテクチャの提案

2018SE065 岡田寿希也
指導教員：野呂昌満

## 1　はじめに

コード最適化を行うにあたり冗長計算除去に着目したとき，有効な手法にメモ化がある．コード最適化とはプログラムの実行効率を向上させるために行われる，コードの記述内容の調整である．最適化手法は多岐に渡るが，その中でも冗長計算除去はコード最適化において効果的な手法であり，以前から多くの研究がなされてきた．[1][2]

メモ化は同じパラメータによる計算が繰り返されるとき，事前に計算した値を用いることで高速化を行う．メモ化を適用する際に問題となるのが，変換対象を決定することである．理由は以下の通りである．

1. パラメータが変わったら結果が再利用できない．
2. 参照透過性のある関数でないと適用できない．
3. 同一性を検査するためのオーバーヘッドが発生する．
4. 結果やパラメータを保存するごとにメモリ消費量が増加する．

これらが原因でメモ化を適用する際に，変換対象を決定し，最適化を施すことは難しい．したがって，メモ化の適用には人間の判断が必要になる．本研究ではその労力を軽減する為に，メモ化の適用を効率化するソフトウェアアーキテクチャを提案する．

## 2　メモ化による高速化の例

メモ化による高速化を行う例として，フィボナッチ数列の項を求める場合について考える．再帰関数を用いたJava プログラムに，配列 memo を用意し，メモ化を適用したものが下記のソースコード 1 である．ソースコード 1 では項が計算される度に配列 memo に格納されていくので，計算時間が削減される．

ソースコード 1　メモ化

```
1  ...
2      public static long calc(int n) {
3          if (memo[n] != 0) return memo[n];
4          if (n == 0) return 0;
5          if (n == 1) return 1;
6          long ret = 0;
7          ret += calc(n-1) + calc(n-2);
8          memo[n] = ret;
9          return ret;
10     }
```

## 3　関連研究

神尾ら [3] はメソッドごと参照・時刻同値という手法を用いた同値性プロファイラを使うことでメモ化適用を効率化する方法を提案した．Java のようなオブジェクト指向言語では，オブジェクトの状態が代入によって変化する場合がある．したがって，メモ化を適用する場合，メソッド呼び出しのパラメータが過去に実行されたものと同じであるかどうかを判断する必要がある．

提案されたメソッドごと参照・時刻同値を用いる手法では，各オブジェクトに更新時刻ベクトルを持たせることで，同一性の検証にかかる時間と正確性を向上させた．実行時間が長く，同一引数による呼び出し回数の多いメソッドを抽出することでメモ化にかかる労力を削減することができる．

同値性プロファイラは各メソッドの参照フィールド集合とメソッド呼び出しの引数を記録する．全てのメソッド呼び出し時に実行されるアドバイスを定義し，引数の値の記録を行う．AspectJ の型間定義を用いることで全てのオブジェクトに更新時刻の配列を持たせ，更新時刻の記録も行う．

## 4　ソフトウェアアーキテクチャの提案

関連研究にて述べた同値性プロファイラからの情報とデータフロー解析により取得する情報を用いて，メモ化の適用を効率化するソフトウェアアーキテクチャを提案する．

### 4.1　構成

コンパイラはコード切り替えポリシーに従い，対象となるメソッド内の共通部分式にメモ化を適用する．

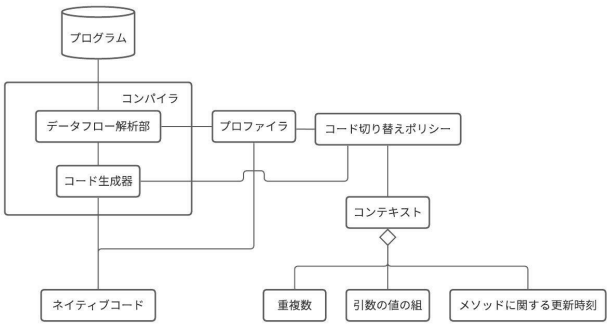


図 1　静的構造

図 1 は提案するソフトウェアアーキテクチャの静的な構造を表している．プログラムをコンパイルする際に，データフロー解析部により解析した情報をプロファイラに送る．プログラムの実行時には実行情報をプロファイラに送る．それによりプロファイラはプログラムの静的解析情報と動的解析情報を得る．重複数，引数の値の組，メソッド

に関する更新時刻をコンテキストとして実行コードの切り替えを行う．具体的にはコンテキストの更新情報をコード切り替えポリシーにより判定し，切り替えを行う場合，該当する部分のコードにメモ化を適用し再コンパイルを行う．

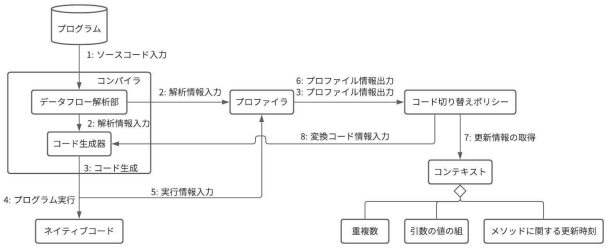


図 2 動的構造

図 2 は提案するソフトウェアアーキテクチャの動的な構造を表している．コンテキストの更新情報をコード切り替えポリシーが横取りすると，ポリシーに従い，コンパイラにメモ化を適用し再コンパイルを行うためのメッセージを送信する．

メモ化を適用する場合，プロファイラより取得した共通部分式の計算結果と共通部分式に関するアドレスを用いる．メモ化を適用可能と判断した共通部分式に対して，事前に実行されたときの計算結果を利用して，再コンパイルを行う．コードの切り替えの判定と再コンパイルは動的に行うので，JIT コンパイラを用いる必要がある．コード切り替えポリシーにより，コードの切り替えが必要と判定された場合，JIT コンパイラを起動してネイティブコードを生成する．具体的には，データフロー解析部により得たパラメータの情報とプロファイラから得たアドレスの情報を用いて，パラメータが変化していないベーシックブロックの始まりと終わりを判定する．判定した箇所に対して，事前に実行されたときの計算結果で置き換え，JIT コンパイラにより，動的に再コンパイルを行うことでメモ化を適用する．これにより，再度実行されるときに，計算が省略されるので高速化を図ることができる．

### 4.2 入出力表とコードの変換

同値性プロファイラとデータフロー解析を用いて入出力表にメモ化適用に利用する情報を記憶させる．具体的には以下の情報となる．

- 重複数
- 引数の値の組
- 参照フィールド集合
- メソッドに関する更新時刻
- 共通部分式に関するアドレスと計算結果

これらの情報を用いてコード切り替えポリシーに従い，メモ化が可能と判断されたメソッド内の共通部分式にメモ化を適用する．

## 5 考察

本研究で提案したソフトウェアアーキテクチャを用いることで，メモ化適用の判断と手動で行うコード書き換えの必要がなくなる．メソッドごとにメモ化を適用する場合，メソッドが外部との入出力を行うとき，メモ化を適用することができない．本研究では，メソッド内の共通部分式に着目して，データフロー解析を行っているので，その問題が解決できる．

提案したソフトウェアアーキテクチャでは，同じメソッドが繰り返し呼び出される場合，メソッド内の共通部分式に対してメモ化を適用する．同一のメソッドが呼び出されるたびに，コンテキストである重複数が変化する．重複数が 2 以上に更新されたとき，更新情報をコード切り替えポリシーが横取りし，引数の値の組とメソッドに関する更新時刻を確認する．引数の値の組とメソッドに関する更新時刻が同一であった場合，共通部分式のパラメータが変化していないので，メソッド内の共通部分式に対してメモ化を適用する．引数の値の組又はメソッドに関する更新時刻が変化している場合，共通部分式のパラメータが変化している可能性があるので，メソッド内の再利用可能な共通部分式にだけメモ化を適用する．

以上のことから，提案したソフトウェアアーキテクチャを用いることでメソッド内の共通部分式に対してメモ化を適用することができると考えられる．

## 6 おわりに

本研究ではメモ化の適用を効率化するソフトウェアアーキテクチャを提案した．今後の課題は，提案したソフトウェアアーキテクチャを実装することである．より多くのプログラムに的確にメモ化を適用することが可能なソフトウェアアーキテクチャを設計するには，メモ化適用時に発生するオーバーヘッドを考慮する必要がある．その為には，実装したソフトウェアアーキテクチャを用いて，メモ化適用の処理時間を計測する必要がある．

従来，メモ化を適用するには人間の判断が必要である．本研究を進めることで，コード最適化の為にメモ化を適用する場合にかかる労力を削減したい．

## 参考文献

[1] 滝本宗宏・原田賢一：「拡張値グラフに基づく効果的な部分冗長除去法」．情報処理学会論文誌，第 38 巻 (1997)， pp. 2237–2250.

[2] A. V. エイホ，R. セシィ，J. D. ウルマン（原田賢一 訳）：『コンパイラ II —原理・技法・ツール—』．サイエンス社，東京，1990.

[3] 神尾貴博・増原英彦：「オブジェクト指向プログラムの高速化を支援するプロファイラ」．情報処理学会論文誌，第 46 巻 (2005)， pp. 1–9.