A-024

依存性情報を持つ中間コードを使ったロード時最適化手法
A Method of Load-time Optimization Using Intermediate Code with Dependency Information

秋山 健† 森本 晃弘‡
Takeru Akiyama Akihiro Morimoto

1. はじめに

本論文では閉路のない DFD のような構造を持ったプログラムのロード時の実行速度最適化を短い処理時間で実現する手法を提案する．閉路のない DFD のような構造は計測装置・監視装置・制御装置などに搭載されるソフトウェアでよく利用される．このような装置には計測対象を切り替えたり，計測方式を切り替えたりできるものがある．切り替え後に実行速度最適化が行われるのが望ましいが，全ての最適化をロード時に行うのは時間がかかり許容できない．そこで，最適化の一部をコンパイル時に行い，ロードにかかる時間を削減する方法を開発した．

2. 対象とするプログラム

ここで対象とするプログラムについて詳細を述べる．

プログラムは複数のモジュールからできており，モジュールは複数のブロックからできている．ブロックは入力と出力と計算を持ち，入力と計算が同じであれば，出力は同じである．また，ブロックへの入力は他のブロックの出力やモジュール外からの入力に繋がれる．

図 1 はモジュールとブロックの関係の例で，X,Y 二種類のモジュールを表している．外側の四角形がモジュールを表し，内部の四角形がブロックを表している．モジュール X にはブロック A,B,C が含まれ，モジュール Y にはブロック A,B,D が含まれている．ブロックは仮入力や別のブロックと矢印で繋がれる．矢印はデータの流れを表しており，矢印の始点から終点へ向かってデータが流れる．仮入力はモジュール外からのデータを受け取る場所で，ロード時に実入力が割り当てられる．実入力の例としてはセンサの値を格納した変数などがある．

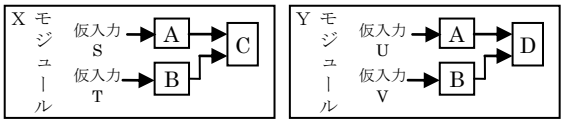

図 1 モジュール

ロードされたモジュールを実行すると，モジュール内のすべてのブロックが実行される．このときブロックは入力側から順に実行され，複数の入力を持つブロックは入力が全てそろった後に実行される．

3. 課題

実入力の割り当て方によって重複した構造が現れるため，この重複した構造をロード時に時間をかけずに除去したい．

例としてモジュール X をふたつロードする状況を挙げる(図 2)．この図では X1 のブロック A と X2 のブロック A は異なる実入力からデータを得ているが，X1 のブロック B と X2 のブロック B は同じ実入力からデータを得ている．

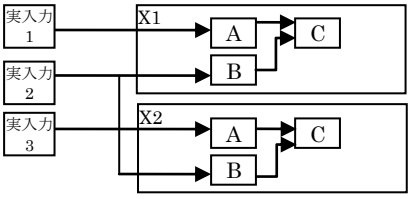

図 2 ロードされたモジュール

この場合，ふたつのブロック B は同じ処理を行う．これは冗長である．これを回避するため，図 3 のように，ブロック B の出力をふたつのモジュールで共有したい．

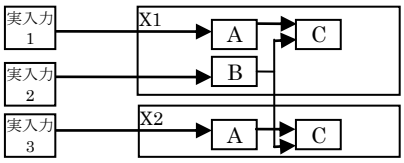

図 3 最適化されたモジュール

しかしどの仮入力にどの実入力が割り当てられるかはロード時に決まる．そのため，重複した構造の除去はロード時に行う必要がある．

4. 従来の手法

今回対象としているプログラムから重複した構造を除去するには，コンパイラ技術の一つである共通部分式除去が利用できる[1]．この手法では構文木の節を作成する際，生成しようとする節が既存の節と同じであれば，新しい節を生成せず既存の節を使うというものである．しかしこの方法は高速ではないため，ロードの利用に適していない．

5. 提案する手法

5.1 概要

今回提案する手法では次の二つのステップで最適化処理を行う．

1. 中間コード作成
2. 最終コード作成

中間コード作成はコンパイル時に行う．中間コードは全ての仮入力が等しいと仮定して最適化を施したコードであり，ブロックが実行すべき順序を満たすよう一列に並べられている．また，中間コードはモジュール毎に作成される．

最終コード作成はロード時に行う．最終コードは実入力が特定された後に最適化されたコードで，ブロックが実行すべき順序を満たすよう並べられている．また，最終コードは全てのロードされたモジュールに対してひとつだけ作成される．

5.2 中間コード作成手順

ここでは図 1 のモジュール X,Y を例に中間コードの作成手順を説明する．

† 株式会社 東芝 Toshiba Corporation
‡ 東芝システムテクノロジー株式会社 Toshiba System Technology Corporation

まず全モジュールをひとつのモジュールに統合する．これにはまず Root ブロックを作成し，出力が使用されていないブロックの出力を仮の Root ブロックに接続する．そして仮入力は全て同じものとする． 図 4 は図 1 のモジュールを統合したものである．

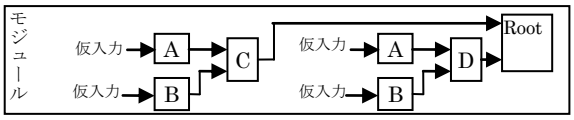

図 4 統合されたモジュール 1

次にこのモジュール内から重複した構造を取り除き，帰りがけ順の深さ優先探索で連番を振る．図 5 はこれを行った後の状態を表している．ブロック内の数字は付加された連番である．

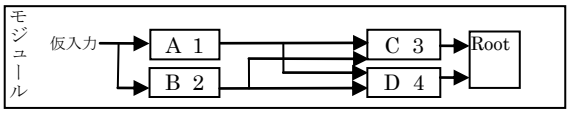

図 5 統合されたモジュール 2

次にモジュール内の各ブロックに全体の中間コードの連番を書き戻す．この書き戻す連番はブロック名の末尾に付加する(図 6).

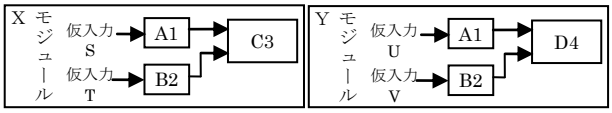

図 6 連番を付加したモジュール

次にモジュール内の各ブロックに依存する仮入力を付け足す．ブロックに依存する仮入力は，そのブロックから入力側に帰りがけ順の深さ優先探索で辿ったときに現れる全ての仮入力を現れた順に並べたものである．依存する入力はブロックの連番の後に付加する(図 7).

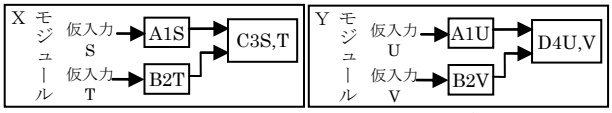

図 7 依存する入力を付加したモジュール

次にこれを連番順に整列させる．このとき各ブロックの入力 は，その接続されている先のブロックの名前に置き換える(図 8)．これが中間コードとなる．

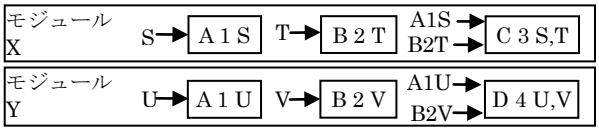

図 8 中間コード

5.3 最終コード作成手順

ここでは例として「実入力 I を仮入力 S に結び付け，実入力 J を仮入力 T に結び付けたモジュール X」と「実入力 I を仮入力 U に結び付け，実入力 K を仮入力 V に結び付けたモジュール Y」の最終コードを作成する手順を説明する．

まずロードするモジュールの仮入力を実入力で置き換える．図 9 はモジュール X,Y の仮入力を実入力で置き換えた図である．

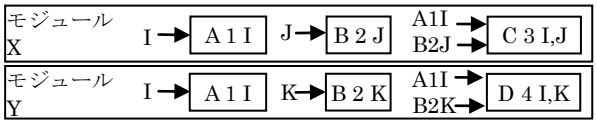

図 9 仮入力を実入力で置き換えたモジュール

次に，空の最終コードを作成し，ロードされるモジュール内のブロックを前から順に追加する．その際，同じ名前のブロックを最終コードに追加しないようにする．

まず X のブロックをリストに追加する．X 内には重複したブロックは存在しないので，全てのブロックが順に追加される(図 10).

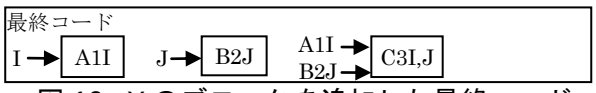

図 10 X のブロックを追加した最終コード

次に Y のブロックをリストに追加する．A1I は既にリストに含まれているので，追加しない． 残りのブロック B2K と D4I,K はリスト内に存在しないので，順に追加される(図 11).

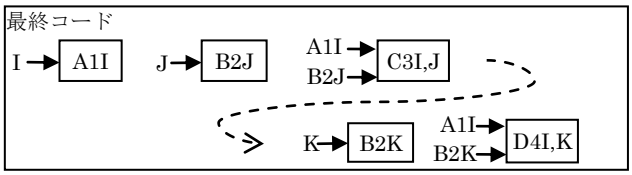

図 11 Y のブロックを追加した最終コード

以上の手順で作成されたリスト内のブロックは，重複した構造が排除され，実行すべき順序を満たした状態で整列されている．

6. 性能評価

本提案の利点は，従来の手法と同じ最適化を行いながらロード時間を削減できる点である．そこで性能比較のため，従来の方法と今回の手法とでモジュールのロードにかかる時間とを比較した．

結果，従来の方法では 1 モジュールあたり約 350μs であり、今回の手法では 1 モジュールあたり約 4μs であった．これにより、ロード時間削減の効果を確認できた．なお，計測環境は 2.7GHz の Core i7 を搭載した PC で，計測対象は 15 個のブロックと 8 個の仮入力を持つモジュールである．また，計測時間の中にはディスクアクセス時間が含まれないようにしている．

7. 最後に

今回，ロード時の最適化にかかる時間を削減するため，依存性情報を含む中間コードを考案し，コンパイル時に最適化の一部を行う手法を提案した．また，性能評価により効果を確認した．

今後の課題として，ロード時間を増やさずにより多くの最適化を行うことがある．たとえばブロック実行順序をより細かく制御することでキャッシュヒット率を高めたり，ブロック間のデータ受渡し用一時記憶を減らしたりすることが挙げられる．

参考文献

[1] A.V.エイホ, M.S.ラム, R.セシィ, J.D.ウルマン, 原田賢一訳, “コンパイラ[第 2 版]”, (2009).