数理モデル化と問題解決 33－10
（2001．3．16）

自動並列化コンパイラの統一的中間表現に対する基本APIの実装

山口 智美　　城 和貴
奈良女子大学 理学部 情報科学科
tomomi@ics.nara-wu.ac.jp

概要

自動並列化コンパイラの開発に際し，様々な問題を考慮しそれらの問題を克服し得るコンパイラを実装することは非常に難しく，開発期間も長くなることが一般的である．この問題に対して，複数の中間表現を統合した統一的中間表現 (UIR) を定義し，最適化等の変換手法をそのインタフェイスを通じて行うことが提唱されている．本稿ではUIRインタフェイス・ライブラリの整備とそれを使ったループ変換手法の実装を通じて，UIRインタフェイスを定義することの意義について議論を行う．

An Implementation of a Primitive API of a UIR Interface for a Parallelizing Compiler

Tomomi Yamaguchi　　Kazuki Joe
Faculty of Science, Nara Women's University

Abstract

This paper presents an implementation of a primitive API that constructs a Universal Intermediate Representation(UIR) interface library for a parallelizing compiler. We investigate how to design the UIR interface library through the implementation of two well-known loop transformations; loop distribution and loop permutation.

1　はじめに

自動並列化コンパイラは，原始逐次プログラムを入力として受け取り，それに対して様々な解析や変換を行った後，自動的に並列プログラムに変換する．自動並列化コンパイラの実装に関しては，単に逐次原始プログラムから並列性を検出し，プログラム変換を施して並列可能な部分を作り出すのみならず，分散メモリシステムでは，スケジューリング，コード及びデータの配置，リモートメモリアクセス，プロセッサ間通信，同期などのオーバーヘッドの軽減など，多種多様の問題を考慮しなければならない．これらの問題を克服し得るコンパイラの実装は非常に難しく，開発期間も長くなることが一般的である．

さらに，開発済みのコンパイラでも，内部構造の改良に伴うプログラムの変更，新たな変換手法の作成による更新など，メンテナンスも難しい．

こういった問題を解決するために，変換手法がよく利用する中間表現の統合化を行い，一貫性を持たせた統一的中間表現 (UIR:Universal Intermediate Representation)[1] とし，これに対して様々な変換手法を任意順序で施すようなコンパイラが開発されている．自動並列化コンパイラであるSUIF[2]，PROMIS[3]などがこのUIRを実装しており，今日では中間表現の実装形態の主流になっていると言える．

また，UIRには変換に必要な全ての情報が格納されている為，そのデータ構造は一般に膨大かつ複雑なものとなるが，これに対するあらゆる変換は，整合性を保持しつつ行われなければならない．そこで，UIRに対しインタフェイスを定義することが提案されている [4]．インタフェイスを通じてのみUIRにアクセスできることとし，様々な変換手法は全てこのUIRインタフェイスを用いて行う．UIRインタフェイスの定義に沿って設計された最適化手法は，異なるUIRに対しても適用可能となる．

UIRインタフェイスは具体的には，UIRインタフェイス・ライブラリで実装される．我々は，このインタフェイス・ライブラリを基本API(プリミティブ・ライブラリ) と応用API(アプリケーション・ライブラリ) に分類することを提案する．基本APIとは，UIRの基本的な操作を行うものであり，応用APIは，最適化手法に典型的に使われる一連の基本API群をまとめたものである．このように分類しておくと，既存のコンパイラに新たなUIRを組込む際には，基本APIを新たなUIRの定義に沿った形で書き換えさえすれば，応用APIは変更しなくてよいので，最適化手法もそのまま使える．よって，コンパイラ開発の労力の大幅な軽減が期待できる．

本稿では，MIRAIコンパイラ [5] に対して基本APIの整備を行い，それを用いたループ変換を実装する

ことで，このようなUIRインタフェイスを定義することの意義を論じる．

以下2章では，UIRとそのインタフェイスについて詳しく述べ，3章では基本APIとその実装について，4章では基本APIを用いたループ変換の実装について，例をあげて説明する．5章では実装した結果を検討し，まとめとこれからの課題について述べる．

2 UIRとそのインタフェイス

2.1 UIR

従来の自動並列化コンパイラは複数の中間表現を保持しており，最適化などの変換はそれぞれが必要とする中間表現に対してのみ操作が行われるため，(1) 変換手法を任意の順序で適用できない，(2) 同じ変換手法なのに中間表現の粒度やレベルによって個別に実装しなければならない，(3) 中間表現の変換時に情報損失の可能性が高い，といった問題があった．

こういった問題を解決するために，変換手法がよく利用する中間表現の統合化を行い，統一的中間表現(UIR)として，これに対して変換手法を施すようなコンパイラが開発されてきた．

今回実装の対象となるMIRAIコンパイラ[1]は，分散メモリ型並列計算機を対象とする自動並列化コンパイラである．このMIRAIコンパイラでもUIRの概念が採用されており，UIRとしてHTG(Hierarchical Task Graph)[1]が用いられている．

2.2 UIRインタフェイス

UIRに対し，その標準インタフェイスを定義することで，(1)UIRの更新の際に整合性が保たれる，(2)UIRの仕様が変更されてもインタフェイス部の修正だけで済む為，UIRの保守，拡張が容易になる．

ところが，UIRは実際には様々な形があるので，それに対する標準の操作というものは非常に定義しにくい．理想的には，UIRの定義された理論からそのインタフェイスも定義されることが望ましいが，UIRの定義が様々である以上それは困難である．よって，実際にあるUIRにインタフェイスを実装し，他のUIRインタフェイスと比較検討しつつ，標準化していくことが必要であると思われる．今回はその試みの前段階として，MIRAIコンパイラのUIRインタフェイスの定義について考察する．

2.3 UIRインタフェイス・ライブラリ

UIRのデータ構造は，そのUIRをアクセスするのに使われるメソッドと共にクラスとして一体化して実装される．一般に，UIRを操作するためにはその複雑なクラス構造を理解しなければならない．

MIRAIコンパイラでは，HTGについて細部のクラス構造を知らない人が最適化手法を実装しようとすると，HTGのクラス構造を理解しなければならないので，HTGに対する基本的なクラス・メソッドを使う場合でさえ，ソースもしくはHTGの仕様書を読まなければならない．これに対して，UIRインタフェイスの基本APIがしっかり定義されていれば，そのコンパイラのUIRの種類にかかわらず，つまり，HTGのクラス構造は全然知らなくとも，UIRインタフェイスの基本APIの仕様書を読むだけで最適化手法の実装はできるし，同じUIRインタフェイスを持つ他のコンパイラの最適化手法も全く同じように実装できる．

3 基本API

3.1 インタフェイス・ライブラリの設計

先に述べたように，我々は，UIRインタフェイスライブラリを基本API（プリミティブ・ライブラリ）と応用API（アプリケーション・ライブラリ）とに分類することを提案している．基本APIは，UIRの基本的な操作を行う．応用APIは，基本APIを組合わせて，最適化手法に典型的に使われる一連の基本API群をまとめたものである．例えば，最適化手法AとBを実装するのに，基本APIのa1,a2,a3,...,が共通に使われていたとすると，この部分を応用APIとしてまとめておく．以後，最適化手法AやBに類似した最適化手法Cを実装する場合，その応用APIを流用できる可能性が非常に高い．よって実装が劇的に簡単になる．さらに，応用APIが行う操作は複雑であるが，その本質は基本APIの組み合わせであり，基本API単位でバグのテストを行っているので，その集合体である応用APIもバグが発生しにくくなる．

UIRインタフェイスがしっかり定義されていれば，新たなUIRを既存の自動並列化コンパイラに組み込むことも極めて容易になる．この場合，新たなUIRのクラス構造の定義を行った後，基本APIを新しいUIRのクラス構造に沿った形で書き換えてやればいいだけである．

3.2 基本APIの例

ここでは，実際に実装した基本APIの例をあげて説明する．実装する際に，操作する対象に応じて基本APIをいくつかのグループに分けた．それぞれのグループ毎に，UIRインタフェイスを構成する1つのクラスとする．HTGのノードはCFGを継承しているので，CFGのメンバに対する操作はHTGノードにも適用できる．よって，CFG,HTGに共通のものとして，CFG/HTGと記述した．CFG/HTGを

[1] 日本学術振興会・未来開拓学術研究推進事業・知能情報高度情報処理『分散・並列スーパーコンピューティングのソフトウェアの研究』(代表：京都大学・島崎教授)

操作するグループ，CFG/HTG ノードが持つ式ノードを操作するグループ，HTG を操作するグループ，各ノードに固有の情報を操作するグループは，オプションクラスとした．尚，オプションクラスは UIR インタフェイスの定義に少しそぐわないが，今回は拡張性を残しておくために実装した．実際の最適化手法では，これを使用しないことが多い．

CFG/HTG 関連 ：

CH-1：ノードの複製
CFG/HTG ノードを複製する．そのノードに保持される情報も全て複製する．

CH-2：枝の更新
CFG/HTG ノードからの枝を，他のノードに繋ぎ直す．

CH-3：ノードの削除
CFG/HTG ノードを削除する．枝の更新も同時に行う．

CH-4：ノードの交換
CFG/HTG ノードを交換する．枝の更新の関数を利用する．

CH-5：ノードの枝の情報を得る
CFG/HTG の枝を得る．ノードの操作の前処理として必要．

CH-6：そのノードは CFG ノードか HTG ノードか判定する
CH-7：そのノードはループの先頭か判定する
CH-8：そのノードの到達可能ノードのリストを得る
CH-9：ノードの CDG 枝を更新する
CH-10：ノードが条件式を持つ場合に式を得る
CH-11：ノード間に依存があるかを判定する
CH-12：ノードに探索済みのフラグを立てる
CH-13：ノードが探索済みかどうか判定する
CH-14：ノードの探索済みフラグをクリアする
CH-15：ノードの親を得る

式関連 ：

EX-1：親 CFG ノードの情報の更新
EX-2：親式ノードの情報の更新
EX-3：式ノードをつなぐ枝の情報を得る
EX-4：式ノードをつなぐ枝の更新
EX-5：式ノードで演算子の種類を得る
EX-6：各式ノードの値を操作する

HTG 関連 ：

HT-1：HTG ノードの子・親の情報の更新
HTG ノードに含まれる CFG を辿ることにより，子親の関係の情報を更新する．ノードの操作の後処理として必要．

HT-2：HTG の子に CFG ノードを加える
HT-3：HTG の子ノードのリストから指定した CFG ノードを削除する
HT-4：HTG の後続ノードのリストに CFG ノードを加える
HT-5：HTG の後続ノードのリストから指定した CFG ノードを削除する
HT-6：その HTG の階層の深さを返す
HT-7：HTG の子のリストを得る
HT-8：その階層でのループのインデックス変数を得る
HT-9：その階層でのループのインデックス変数を設定する
HT-10：その階層でのループの初期値式を得る
HT-11：その階層でのループの初期値式を設定する
HT-12：その階層でのループの終値式を得る
HT-13：その階層でのループの終値式を設定する
HT-14：その階層でのループのループ幅を得る
HT-15：その階層でのループのループ幅を設定する
HT-16：HTG の子のグラフで始点から終点までのサイクルを含まない道を得る (深さ優先探索)

オプション ：

OP-1：その他各 CFG ノードに固有の情報の操作を行うもの

4 基本 API を用いたループ変換手法の実装

ループ変換は，代表的な最適化手法の一つである．並列化の対象となる科学技術計算を行うプログラムでは，ループを多用しており，ループに対する最適化を施すことで，高速化の大きな効果が期待できる．

ここで，基本 API を用いたループ変換の実装の例として，ループ分配とループ置換の 2 つをあげる．

4.1 ループ分配 (loop distribution)

(ステップ 1) 分割するループ内のデータ依存グラフ D を作成
(ステップ 2) D の強連結部分を検出し，その非サイクル凝縮グラフを作成する．
(ステップ 3) 強連結部分を位相的に並べ替える
(ステップ 4) 中間表現の操作

以下のサンプルプログラムの例で，実際に基本 API を使ったループ分配を説明する．

```
DO I = 1, N
  DO J = 1, N
    S1: A(I, J) = ...
    S2: A(I, J+1) = A(I, J) - 1
  END DO J
  S3: B(I) = 2 * D(I) + 3
  S4: C(I) = B(I) - 4
END DO I
```

(ステップ 1)
・CH-11
ノード間に依存があるかどうか判定し方向ベクトルを得，データ依存グラフ D を作成
(ステップ 2)

D で強連結部分 (サイクル) を探す
・CH-8
到達可能ノードを順に調べて，サイクルを探す．
・CH-13
・CH-12
強連結部分 (サイクル) があれば，それをリストに入れ，まとめる．
・CH-14
(ステップ 3)
非サイクル凝縮グラフに対し，有向グラフにおけるトポロジカルソートを適用し，位相的に並べ替える．
・HT-16
・CH-12
・CH-13
・CH-14
(ステップ 4)
3 で得たグラフの順に，強連結部分はまとめたまま，ループ分割を行う．
・CH-1
・CH-2
・HT-1
・CH-9
結果

```
DO I = 1, N
  DO J = 1, N
   S1: A(I, J) = ...
   S2: A(I, J+1) = A(I, J) - 1
  END DO J
END DO I

DO I = 1, N
  S3: B(I) = 2 * D(I) + 3
END DO I

DO I = 1, N
  S4: C(I) = B(I) - 4
END DO I
```

4.2 ループ置換 (loop interchange)

(ステップ 1)
最外側ループから完全ネストループを検出する
・HT-6
(ステップ 2)
1 で検出した完全ネストループのループ本体の全ての方向ベクトルを得る．
・CH-11
方向ベクトルからキーとする 2 つの配列を得る．
(ステップ 3)
2 つのキーの配列から，新たなループのネスト状態を得る．
(ステップ 4)
3 で得た順に，中間表現を操作する．
・CH-4
・CH-2
・HT-1
・CH-9

5 まとめ

本稿では，自動並列化コンパイラの統一的中間表現インタフェイスである基本 API を整備し，それを用いてコード最適化手法の一つであるループ変換のうち，ループ分配とループ置換を実装した．

その結果，前章の例のように，基本 API を用いることによってループ変換が簡潔なコードで実装された．またさらに，この 2 つのループ変換手法は UIR インタフェイスである基本 API を用いて実装されているので，UIR が変更されてもそのまま適用できるし，同じインタフェイスを持つ別の UIR に対しても適用可能となる．

今回は，グラフに対する様々な操作を考え，列挙することで基本 API を実装したが，実際に最適化手法を実装して，よく使う操作とあまり使わない操作があり，削除可能なものもあると思われる．さらに複数の最適化手法を実装し，必要なものを拡張・選択していきたい．どのような操作を基本 API として定義するかは，今後も考察が必要と思われる．

また，例えば，4.1 のループ分配の例で，(ステップ 3) においてトポロジカルソートを行っているが，このような操作は応用 API として実装されることが望ましい．今後は，このような応用 API の整備も行っていきたい．

参考文献

[1] M.Girkar and C.D.Polychronopoulos. *The Hierarchical Task Graph as a Universal Intermediate Representation,* International Journal of Parallel Programming, 22(5):519-551, October 1994.

[2] Robert Wilson et al. *An infrastructure for reserch on parallelizing and optimizing compilers,* Technical Report, Computer System Laboratory, Stanford University.

[3] H.Saito, N.Stavrakos, S.Caroll, C.Polychronopoulos and A.Nicolau. *The Design of the PROMIS Compiler,* Technical Report, UIUCCSRD 1539(revised), March 1999.

[4] K.Kambe, T.Nakanishi, K.Joe, Y.Kunieda, F.Kako. *An Implementation of Loop Transformations with a Universal Intermediate Representation Interface Library,* Proceedings of the International Conference on Parallel and Distributed Proceeding Techniques and Applications, Vol. IV, pp.1905-1911, June 1999.

[5] Y.Kunieda, K.Joe, A.Fukuda, T.Uehara, S.Saito, T.Saito, M.Sasakura, T.Nakanishi. *Design and Implementation of an Automatic Parallelizing and Distributing Compiler with Visualization Tools and the Runtime Environment,* Proceedings of the International Conference on Parallel and Distributed Proceeding Techniques and Applications, Vol. II, pp.713-719, June 2000.