計算機アーキテクチャ　144－14
（２００１．７．２６）

自動並列化コンパイラPROMIS–NWUの概要

山口智美　　石内寿子　　岩坂麻実　　羽田昌代　　庄野逸　　城和貴

j-ken@alice.ias.nara-wu.ac.jp

* 奈良女子大学大学院人間文化研究科　† 奈良女子大学理学部情報科学科

概要

コンパイラは高コスト・ソフトウェア開発の代表のように扱われているが，近年様々な技術革新により，その開発コストの削減が図られている．例えばオブジェクト指向技術の導入により，コストは数分の一に減少する．このようなソフトウェア工学的なアプローチの他に，コンパイラの構成要素を再利用する形での開発の低コスト化が期待されている．本研究グループでは，イリノイ大学で開発された自動並列化コンパイラ PROMIS の中間表現を入れ替えることで，異なるコンセプトに基づくコンパイラを低コストで開発している．本稿では開発中の自動並列化コンパイラ PROMIS–NWU の概要について報告する．

The Overview of the Parallelizing Compiler PROMIS–NWU

Tomomi Yamaguchi　Hisako Ishiuchi　Asami Iwasaka
Masayo Haneda　Hayaru Shouno　Kazuki Joe

* Graduate School of Human Culture, Nara Women's University
† Department of Information & Computer Sciences, Nara Women's University

Abstract

Compilers have been regarded as a cost expensive software. However, the implementation cost tends to be reduced recently by various new software development technologies. For example, software developments with object oriented technologies require extremely inexpensive costs compared with conventional ones. Besides such software engineering approach, low–cost developments of compilers by the re–use of compiler components are expected. We are developing such a low–cost compiler by replacing the intermediate representation of PROMIS developed at UIUC with our new intermediate representation, aiming at other concepts for the compiler. In this report, we introduce the concept of the parallelizing compiler PROMIS–NWU which is under development at Nara Women's University.

1　はじめに

これまでに開発されている自動並列化コンパイラは，基本的に共有メモリシステムを対象とし，分散メモリシステム（DSM）を対象とした自動並列化コンパイラは開発途上にあり，多くの議論が行われている研究テーマである．

コンパイラによる DSM を対象としたプログラムの最適化と再構築はプログラムの並列化とデータの配置（データの分散配置，再割り当て）という 2 点についてそれぞれ行うことが必要である．これらは共に NP 完全な問題として知られており，一方の解がもう一方の解に干渉する可能性がある．従って，それぞれに対して（近似的な）最適解を得るためには，二つの最適化を同時に行うべきであるというのが我々の着眼点である．

我々は，このためにタスクと変数の情報を同一のフレームワークで表現する自動並列化コンパイラの中間表現 (IR)Data Partitioning Graph (DPG) [5] を既に提案している．DPG は，Hierarchical Task

Graph (HTG)[7] を拡張して変数の情報を付加した IR であり，HTG とはループと関数に基づいてプログラムから階層構造を抽出したタスクグラフである．DPG は，HTG の各階層において変数参照を含む完全なプログラム情報を表現するように定義されている．我々は過去の研究 [4] において HTG の各層におけるプログラム分割のアルゴリズムが扱うことが可能であるタスクグラフのサイズに限界があることを示している．この傾向は我々が既に提案しているプログラムとデータの同時分割アルゴリズム [6]（DPG 上での適用を前提）でも同様であると推測される．

そこで我々は当該アルゴリズムが対象とする情報を絞り込むことで，この問題の解決を図ろうとしている．この制限はタスクではなく変数の情報に対して行うものであり，DPG のデータ表現を改良することで表現される．改良された新しい IR は Task and Variable Representation Graph(TVRG) と呼ばれ，文献 [1] にて発表済みである．

TVRG の有効性を検証するためには，実際に自動並列化コンパイラの中間表現として実装すべきであるが，コンパイラの開発はコストが高く容易ではない．一方，コンパイラの構成要素を再利用する研究が多方面で行われているが，TVRG の母体となる HTG も，そのようなコンパイラ構成要素である．HTG はイリノイ大学で開発が進められている自動並列化コンパイラ PROMIS[9] の UIR (Universal IR) であり，本研究グループは PROMIS の UIR を HTG から TVRG に置きかえることで，異なるコンセプトの自動並列化コンパイラ PROMIS–NWU の開発に着手している．

PROMIS–NWU は WS クラスタや PC クラスタ等の分散メモリ環境をターゲットとした自動並列化コンパイラであり，入力された逐次プログラムを MPI を組み込んだ並列プログラムに変換する．本稿では PROMIS–NWU の設計方針ならびに実装方針について報告する．

以下，本稿の構成は次の通りである．第 2 章では PROMIS の概要を，第 3 章では PROMIS–NWU の設計方針を，第 4 章では TVRG の概説を，第 5 章では PROMIS–NWU の実装方針について述べる．また，第 6 章では関連研究について説明する．

2 PROMIS の概要

PROMIS は C, C++, JAVA, Fortran で書かれた逐次プログラムを，マルチスレッド，スーパースカラ，VLIW 等のプラットフォーム上で高速に実行させるための最適化を行うコンパイラである．PROMIS の特徴は，フロントエンド用の中間表現（IR）とバックエンド用の IR を同じ枠組みの中で定義した UIR(Universal Intermediate Representation) を採用していることである．この UIR は HTG (Hierarchical Task Graph) をベースに，フロントエンド用の HUIR (Highlevel UIR) とバックエンド用の LUIR (Lowlevel UIR) とが実装されており，両 IR では情報の共有が可能となっている．

フロントエンドでの並列性とバックエンドでの並列性の両方を抽出する試みは文献 [9] にある．ここではフロントエンドのコンパイラとして UIUC の Parafrase-2 が，バックエンドのコンパイラとして UCI の EVE が採用され，高レベルと低レベルの並列性の協調抽出が可能であることが示された．PROMIS はその成果を元に，1997 年より UIUC と UCI でスクラッチから開発が進められてきた自動並列化コンパイラである．

現在のところ，PROMIS の高レベルでの最適化には Loop distribution, fusion, unrolling, peeling, interchange 等のループ変換，Scalar privatization, 並列 reduction, サブルーチン・インライン化等の諸変換が実装されている．その他，定数伝播，定数畳込み，コピー伝播，デッドコード削除等の古典的な最適化手法も実装されている．並列実行可能として検出されたループは，IML マルチスレッド・ランタイムライブラリ [8] を呼び出すように変換される．バックエンド部分は，VLIW とスーパースカラの非最適化アセンブリ・コードの出力を行う．ただし，IA32 に対しては最適化コード生成の実装が完了している．また，PROMIS は入力されたプログラムを C 言語に変換するツールとしても利用可能である．

3 PROMIS–NWU の設計方針

PROMIS–NWU 開発の目的は，1）共有メモリ，分散共有メモリ，ソフトウェア分散メモリ，分散メモリという並列システムのメモリ・アーキテクチャ

に対してシームレスな自動並列化コンパイラを実現すること，2）コンパイラを再利用可能な構成要素に分割し，標準的なコンパイラのインフラストラクチャを整備すること，の二点である．

1）には PROMIS の開発目的に直交する課題としての位置付けがある．すなわち，PROMIS はソース・コード・レベルでの並列性抽出から，命令レベルでのコード最適化までを，統一的な枠組みの中で行うものであるのに対し，PROMIS–NWU は，出力を高レベルのものに限定する代わりに，PROMIS の主なプラットフォームである共有メモリ環境から，近年急速に市場を拡大している WS クラスタや PC クラスタに代表される分散メモリ環境までを，統一的な枠組みの中で対応する．そのため，プログラムの自動並列化に加えてデータの自動分散化も必須の要件となり，第 4 章で説明する TVRG を UIR として採用する．さらにコード生成は，MPI ライブラリを使用した並列プログラムを出力するものとする．

2）は既存のコンパイラの構成要素を再利用して新しいコンパイラを低コストで開発する手法の確立を目指すものである．第 6 章で述べるように，そのような試みは各方面で精力的に推進されているが，本研究グループでは UIR インタフェイスの標準化 [3] を目指している．PROMIS–NWU でこの目的を達成するためには，PROMIS の UIR である HTG を TVRG に完全に入れ替えるべきであるが，そのための UIR インタフェイスの整備は現状では困難であるため，今回の実装では HTG を拡張する形で TVRG を実装するものとする．これに伴い，UIR インタフェイスの整備をある程度整えた後，UIR の完全な入れ替えを図る．

TVRG が実装され，UIR インタフェイスがある程度整備された時点で，PROMIS には未実装の最適化手法を実装することができる．特にプログラムとデータの同時分割は現在アルゴリズムの検討を行っているところである．

4 UIR としての TVRG

本章では PROMIS–NWU の UIR である TVRG について概説する．TVRG の詳細な定義ならびに各ノードのラベル付けアルゴリズムに関しては，文献 [1] を参照されたい．

TVRG の母体となる HTG を構成するノードの集合を V_H，エッジの集合を E_H とすると，$HTG = (V_H, E_H)$ と定義できる．V_H の要素であるノードをタスクノード，E_H の要素であるエッジをフローエッジと呼ぶ．ノード $N \in V_H$ がコンパウンドノード[1]である時，N が包含するグラフ全体と N 自身は，1 つの HTG を構成する．その HTG を HTG(N) と表記し，この部分グラフを構成するノードとエッジをそれぞれ $V_H(N)$，$E_H(N)$ とすると，$HTG(N) = (V_H(N), E_H(N))$ と表せる．

[7] で定義される HTG において，条件分岐の then 節，else 節を明示するノードは存在しない．TVRG では，これらを明確に表現することが必要となるために，以下に示す新しいノードを V_H に追加する．また，これらのノードの追加によって生じるエッジは E_H に追加する．

IF ノード	分岐の開始
ELSE ノード	分岐の else 節の開始
THEN ノード	分岐の then 節の開始
ELSE-IF ノード	else 節内の分岐の開始
THEN-IF ノード	then 節内の分岐の開始
ENDIF ノード	分岐の終点
ENDIF-IF ノード	分岐終点に引続く分岐の開始

TVRG はタスクノードの他に，変数を表す変数ノードを持つが，この変数ノードがスコープを明確に表現する．タスクを表現する HTG は，関数ごとに生成され，ループの構造に基づいてタスクを階層化している．これにより，関数又はループ単位での並列化と同時に，変数のスコープがループ又は関数によって定義されることで，同期の最適化が可能になることが [7] において言及されている．[7] が，共有メモリシステムにおける最適化のみを目的としていたのに対して，我々は分散メモリシステムを対象としているために，変数ノードとそのスコープ情報を用いて，分散メモリに配置されるタスクと変数の組み合わせを表現することで最適化を行おうと考えている．

変数ノードの集合を V_V，タスクノードとの間を連結する辺の集合を E_V で表すと，TVRG は次のように表現される．

$$TVRG = (V_H, V_V, E_H, E_V)$$

各変数 (配列を含む) は基本的に 1 つの変数ノードで表現する．表現する情報は，スコープと配列である場合の添え字の値域である．

[1] サブタスクを含むタスクノード

変数のスコープは，変数が定義されるループ又は関数を示すことで表現することができる．定義された変数は，そのループ，関数が含まれるタスクすなわちコンパウンドノードにおいてのみ有効である．変数の定義が含まれる集合に変数ノードを埋め込み，その集合に含まれる全てのタスクから参照可能とすることで，スコープを表現できる．この集合を示すコンパウンドノードをタグノードと呼ぶ．

スコープはプログラムから得られる定義位置によって決定される．十分な解析を行うことなくタグノードを決定すると，変数の定義がある階層に集中する場合，その階層のノード数のみが増大してしまうため，最適化手法適用のコストが高くなる．多くのプログラマは変数の使用の有無にかかわらず，1 箇所での変数定義を行う傾向にあるが，実際に使用される領域とは等しくない場合が多い．TVRG では変数ノードに，そのような領域を示すタグノードを割り当てることで，ノード数が局所的に増大することを防ぐ．

変数ノード s に対して参照を持つタスクノードが s のタグノードより下位の階層に位置する場合があるため，両者の関係を明示的に表現するためには全てのノードは HTG の階層構造における絶対的な位置の情報を持つことが必要となる．TVRG では全てのタスクノードに一意なラベルを付けることで，その位置を表現する．ラベル付けアルゴリズムやタグノードの決定方法に関しては文献 [1] を参照されたい．

TVRG においてプログラムは階層構造で表現され，プログラムの分割は各階層を対象に行われる．この時，分割の対象となる階層を選択することで，分割の粒度を調節することが可能になる．

タスクのみを分割する際には，分割の対象となるタスクの実体が最下層にあるために，どの階層を選択しても情報を欠くことなく分割することが可能であるが，変数ノードはスコープを示す階層（タグノードで代表される階層）に配置されているため，対象とする階層より上位の階層に配置された変数ノードは，分割の対象とすることができない．

TVRG でデータ分割を表現するために，スコープを示す変数ノードとは区別される表現が必要となる．この表現は変数フィールドと呼ばれる．変数フィールドにおいて表現される変数は，より上位の階層において定義され，分割の対象となる階層以外においても使用される．同じ変数を使用する他の分割に対して，変数の参照の結果を反映する必要がない場合は，他のスコープを表現する変数ノードと同様に扱うことが可能であるため，反映する必要がある変数と区別して表現するために，変数フィールドは参照パターンによって『参照による内容の更新が行われる変数ノードの集合』,『内容が更新されない変数ノードの集合』の 2 つの集合に分類される．前者をスコープ・フィールド，後者をアクセス・フィールドと呼ぶ．

5 実装方針

第 3 章で述べたように，今回の実装では PROMIS の UIR を拡張する形で TVRG を実装し，PROMIS–NWU の中核を構築する．この拡張は PROMIS の Extensible Compiler Interface (ECI) [10] を利用して行う．

ECI は PROMIS の機能拡張のために用意されているインタフェイスである．PROMIS の UIR は，正確には CORE IR とユーザ定義部分とに分類される．CORE IR は変数，型，式，ステートメント・ノード，制御ならびにデータ・フロー辺で構成され，その他の中核データ構造は ECI を通してユーザが追加定義できるような形になっている．ユーザ定義は External Data Structure (EDS) を使って行われる．EDS を用いた PROMIS の拡張機能としては，Privatization 解析，Reduction 解析，制御依存解析，データ・フロー情報，イタレーション変数解析，添え字解析などが既に実装されている．このような形で追加機能を独立して実装できる形態の欠点として，追加された新機能がいかにしてコンシステンシを保ったまま CORE IR を管理できるか，という点が考えられる．これに対して ECI では CallBack と呼ばれるメカニズムを導入している．CallBack は簡単に言えば CORE IR の再構築である．ECI を使って新機能を追加する際には，CallBack を CORE IR の更新時に挿入することで，UIR の一貫性を保つことができる．

このように ECI を使って TVRG を PROMIS の UIR に追加して PROMIS–NWU の中核とするわけだが，この時追加されるデータ構造を実装形態で分類すると，1）データ構造自体を新たに定義するもの，2）既存のデータ構造の拡張を行うもの，3）既存のデータに対する属性情報の付加を行うもの，の三種類が考えられる．

変数ノードや参照を表すエッジの実装は１）に該当する．TVRG の様々な IF ノードの実装は２）に該当する．また，ラベルや変数フィールドなどの情報は３）に該当する．以下に，各実装について説明する．

プログラム中で使用される変数を TVRG の変数ノードとして実装する場合，変数ノードの保持すべき属性情報としては,『タグノード』,『配列であるときの添え字の値域』がある．実装には EDS を用いる．

タスクノードと変数ノードとの間のエッジは，プログラム中の変数へのアクセスを表すので，参照エッジに必要な属性情報は,『参照が情報の更新を伴う参照であるかどうか』と，エッジの参照先の変数が配列だった場合,『参照領域の開始位置，参照領域の終点，参照間隔の情報』である．実装には EDS を用いる．

TVRG の種々の IF ノードとは，4 章で述べられている IF，ELSE，THEN，ELSE-IF，THEN-IF，ENDIF，ENDIF-IF の各ノードのことである．これらは CORE IR のデータ構造を拡張して実装するべきであるが，既存の最適化手法等の整合性を保つために，各種 IF ノードの実態は EDS で定義し，IF ノードのアクセスには ECI の機能の一つである intrinsic 命令を使って，CORE IR の IF ノードではなく，EDS で定義した IF ノードをアクセスするようにする．

ラベルは TVRG において絶対位置を決定するものであり，HTG の属性情報に TVRG 用の ID として加えることも可能であるが，前述の IF ノードと密接に関係するため，intrinsic 命令を使って属性情報の追加を行う．

変数フィールドは HTG の属性情報として，自身がタグノードとなる変数のリストを持つことで実現できる．

6　関連研究

IR をベースにコンパイラの拡張を行う研究に，スタンフォード大学の SUIF[11] が知られている．SUIF は AST (Abstract Syntax Tree) を元にした標準的なフォーマットを持ち，ファイルとして出力も行える IR である．しかしながら，並列性や制御情報，依存情報といった文法以外の情報は保持しておらず，全てのパスで共有できる情報は SUIF というフォーマットそのもののみである．そのような付加情報を持たせたい場合，SUIF では注釈 (notation) の形で情報を他のパスに伝えることができる．この注釈を利用した SUIF の拡張として，Machine SUIF ライブラリ [12] や OPI (Optimization Programming Interface)[13] が知られている．これらは低レベルでの最適化をターゲットに依存しない形でライブラリの形で実現したものであり，主にバックエンドで用いられる．これらのライブラリを使えば，他のプラットフォームに対するバックエンド出力は，Machine Description Files を変更するだけで可能となる．ただし，OPI は高レベルの最適化には何も寄与することはできない．

コンパイラを個々の構成要素から構築するフレームワークを与えるものとして，Zephyr[14] が知られている．SUIF 同様，Zephyr の IR も AST をベースにしており，異なる IR の間で情報を渡したり更新したりするための Zephyr IR ツールが用意されている．特に，Zephyr は SUIF のフロントエンドを持つ．

PROMIS で採用されている UIR の概念は，SUIF/Zephyr の提供する IR ツールを利用して得られる IR 間の情報共有とは，目的が異なる．SUIF/Zephyr の IR ツールは，各パスにおいて使われる独自の IR を扱うのに用意されているものであり，一つの IR が全ての情報を持つ UIR とは対峙する考えである．しかしながら，PROMIS において，新しい IR に対処するときの実装のしかた [10] は，SUIF/Zephyr の手法にむしろ近いことは興味深い．

本研究グループでは，異なる UIR 間での情報共有や変換 [2] を目指しており，そのために UIR インタフェイスの定義と設計 [3] に着目している．

7　結論

本稿では分散メモリ環境を対象として，MPI による並列プログラムを自動生成するコンパイラ PROMIS–NWU の設計方針と実装方針について報告した．今回の実装の主目的は UIR の入れ替えであり，既存の UIR を拡張する方針とは言え，新しい UIR を実装して別のコンセプトのコンパイラを安価な実装コストで開発できることを示すことの意義は大きい．

現在，PROMIS の ECI を利用して TVRG の実装を行っているが，この実装が終われば MPI 出力を行うコード生成や，データとプログラムの同時分割ア

ルゴリズムの実装等を順次行っていく予定である．

謝辞

PROMIS–NWU の設計方針に関し議論していただいたイリノイ大学 CSRD の Steve Carroll 氏ならびに Constantine Polychronopoulos 教授に感謝致します．本研究の一部は文部科学省科研費基盤研究 (B)13480085 による．

参考文献

[1] Haneda, M., Shouno, H. and Joe, K.: *Task and Variable Representation Graph: An Intermediate Representation of Parallelizing Compilers for Distributed Shared Memory Systems*, Int'l workshop on Advanced Compiler Technology for High Performance and Embded Systems, accepted (2001).

[2] Soyama, N., Nakanishi, T., Joe, K., Kunieda, Y. and Kako,F.: *Converting Different Intermediate Representations of Parallelizing Compilers: A Case Study*, Int'l Conf. on Parallel and Distributed Processing Techniques and Applications 99, Vol.IV, pp.1874–1880 (1999)

[3] Yamaguchi, T., Shouno, H. and Joe, K.: *The Design and Implementation of a UIR Interface for the MIRAI Parallelizing Compiler*, Int'l Conf. on Parallel and Distributed Processing Techniques and Applications 2001, accepted (2001).

[4] Takata, M., Kunieda, Y. and Joe, K.: *A Heuristic Approach to Improve a Branch and Bound based Program Partitioning Algorithm*, Innovative Architecture for Future Generation High–Performance Processors and Systems, IEEE CS Press, pp.105–114 (2000).

[5] Nakanishi, T., Joe, K., Polychronopoulos, C., Fukuda, A. and Araki, K.: *The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning*, 7th Int'l workshop on Languages and Compilers for Parallel Computing, pp.170–185 (1994).

[6] Nakanishi, T., Joe, K., Saito, H., Fukuda, A. and Araki, K.: *CDP^2 Algorithm: Combined Data and Program Partitioning*, Int'l Conf. on Parallel Processing, vol.II, pp.177–181 (1995).

[7] Girkar, M. and Polychronopoulos, C.: The Hierarchical Task Graph as a Universal Intermediate Representation, Int'l J. of Parallel Programming, Kluwer Academic, Vol.22, No.5, pp.519–551 (1994).

[8] http://www.csrd.uiuc.edu/IML

[9] Saito, H., Stavrakos, N., Polychronopoulos, C. and Nicolau, A.: *The Design of the PROMIS Compiler*, Int'l J. of Parallel Programming, Kluwer Academic, Vol.28, No.2, pp.195–212 (2000).

[10] Carroll, S., Ko, W., Yankelevsky, M. and Polychronopoulos, C.: *Optimizing Compiler Design for Modularity and Extensibility*, 14th Int'l workshop on Languages and Compilers for Parallel Computing, accepted, (2001).

[11] Wilson, R., French, R., Wilson, C., Amarasinghe, S., Anderson, J., Tjiang, S., Liao, S., Tseng, C., Hall, M., Lam, M. and Hennessy, J.: *The SUIF compiler system: a parallelizing and optimizing research compiler*, CSL-TR-94-620, Stanford U., (1994).

[12] Smith, M.: *Extending SUIF for Machine-dependent Optimizations*, 1st SUIF Compiler workshop, (1996).

[13] Holloway, G. and Smith, M.: *An Extender's Guide to the Optimization Programming Interface and Target Descriptions*, Harvard U, (2000).

[14] Appel, A., Davidson, J. and Ramsey, N.: *The Zephyr compiler infrastructure*, IEEE Supercomputing98, research exibition (1998).