4X－9

分散・並列化コンパイラMCの開発基盤 —中間表現—*

首藤一幸　村岡洋一
{shudoh,muraoka}@muraoka.info.waseda.ac.jp
早稲田大学 理工学部

1　はじめに

プログラムのコンパイラ内部表現である中間表現は並列化コンパイラ研究の基盤であり、続く研究に大きな影響を与える。

本稿では、我々が研究[2]の基盤として設計、実装したMC中間表現の設計を説明する。

2　MC中間表現の特徴

- 基本構造はAbstract Syntax Tree(AST)。制御フローに着目すると階層フローグラフ、依存関係エッジに着目すると階層タスクグラフとなる。
- 高レベルな表現のみを持つ。
- Javaのパッケージとして実現。
- 中間表現に対応する可読表現を敢えて排除。
- 中間表現をソフトウェア部品の蓄積フォーマットとしている。

3　設計

3.1　基本構造

ASTを基本としている。いわゆる高級言語の文がノードに相当し、ノード間の関係をエッジで表す。エッジには制御フローエッジ、依存関係エッジ、通信エッジなどがある。中間表現で表されたプログラムの例を図1に示す。

基本構造を決定するにあたっては次の選択肢がある。

高レベル 伝統的なASTやHierarchical Task Graph(HTG)など。単位はいわゆる高級言語の文程度。

低レベル 四つ組や、SUIFの一部であるlow-SUIFで採用されているRISC assembly codeなど。DOループなどを表現できる比較的高レベルな四つ組も利用されている。

*Intermediate Representation
for the MC Distributing and Parallelizing Compiler
Kazuyuki SHUDOH, Yoichi MURAOKA
School of Science and Engineering, Waseda Univ.

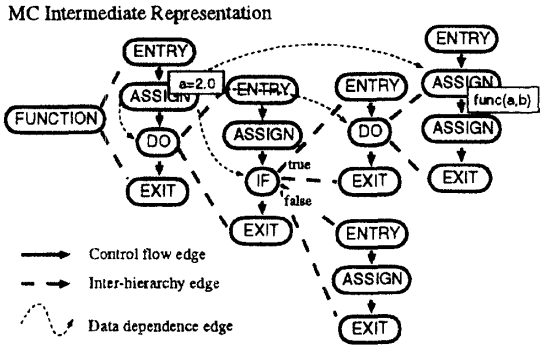

図1: 中間表現

bi-level 高、低レベル表現のhybrid。SUIF、SCOREで採用され、最近PROMISプロジェクトでも採用された。

MCで高レベル表現であるASTを採用した理由は次の通りである。

- source-to-source translationを想定している
- 機械語レベルの最適化は既存コンパイラに任せる。

3.2　実装言語

中間表現の実装言語はコンパイラの実装言語となる。中間表現、コンパイラに求められる性質を考慮して決定する必要がある。

以前はCを選択するプロジェクトが多かったが、最近ではC++が選ばれることが多い[1]。

MCではJava言語環境を選択した。選択理由は次の通りである。

- Garbage Collectorがあり、ユーザによるメモリ管理が不要であること。
  中間表現をデーモンプロセスが扱うのでメモリリークが許されない。
- marshaling(Sunの言葉でserialization)を利用することで、作成、維持の人的コストが非常に高い中間言語を排除できること。

Javaを採用することの唯一の問題は、プログラムがソフトウェアのVirtural Machine上で実行されることからくる処理の遅さである。これについてはJITコンパイラやJava to Cトランスレータで解決できると考えている。

	C	C++	Java
実行速度	○	○⁻	×*1
メモリ管理	×*2	△*2	○
保存	×	△	○
開発効率	△	△	○

*1 Java to C translator, JIT compiler で改善可能

*2 C, C++用 conservative garbage collector を用いることで改善可能

表 1: 中間表現の実装言語の比較

3.3 可読表現の排除

本稿では、抽象的な構造または計算機内部表現を中間表現、可読なテキスト表現を中間言語と呼ぶ。

MC では敢えて可読な中間言語を排した。中間表現と可読表現の相互変換を行うためには中間言語のジェネレータおよびパーザが必要となる。その作成は非常に手間がかかる上、中間表現、言語の仕様変更、拡張に応じてメンテナンスしていく必要がある。この、作成、維持コストにかかる人的コストが非常に高い。

可読表現を用意するのは次の理由からである。

- 保存、ネットワーク経由の受渡し、周辺ツールからの利用が可能になる
- 中間表現を可視化フォーマットとして利用
- 手で操作しやすい。テキストの編集によって中間表現を操作できる。

この内、中間表現の保存、受渡しは、オブジェクトの marshaling で対応した。オブジェクト指向言語で実装された中間表現ライブラリでは内部表現はオブジェクトのグラフである。Java では、オブジェクトを整列 (marshal) して、ファイルに保存したりストリームに流し込むことが可能であるが、C++でこれを実現するのは難しい。

可視化、中間表現の操作は

- MC プリプロセッサが疑似コードを生成
- 中間表現エディタ (図 2) を開発

することで対処している。

3.4 ソフトウェア部品としての中間表現

保存した中間表現を後でリンクして再利用することが可能である。この仕組みでライブラリを構成できる。実際に、組み込み関数、手続きは、中間表現の形で保存、ライブラリ化してある。

関数、手続きが一旦機械語になってしまうとアーキテクチャ中立ではなくなるし、インライン展開も並列化もできない。再利用の方法が非常に限られてしまう。MC では、関数、手続きは中間表現

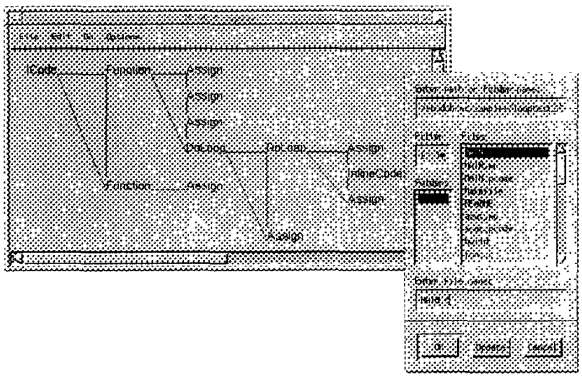

図 2: 中間表現ビューア

の形式で蓄積されるので、過去に記述、保存された組み込み関数をインライン展開したり並列化することも可能である。

レポジトリ　中間表現レポジトリの開発を計画している。現在は、MC プリプロセッサで中間表現のリンクを行う際に、中間表現の格納場所を URL で指定できる。リンク作業の例を示す。

```
% mcc -o aout.mc main.mc \\
        http://foo.ac.jp/irlib/dgemm.mc
```

4 まとめ

並列化コンパイラ研究の基盤として設計、実装した MC 中間表現について次を述べた。

- 基本構造、実装言語の選択方針
- 可読表現の排除と、問題への対策
- ソフトウェア部品の蓄積形式としての可能性

今後は、MC の目標実現の前段階として構成した並列プログラミング環境を基に、並列化コンパイラを構成する。

参考文献

[1] Kathryn S. McKinley, J. Eliot B. Moss, Sharad K. Singhai, Glen E. Weaver, and Charles C. Weems. Compiling for heterogeneous systems: A survey and an approach. Technical Report CMPSCI Technical Report 95-82, University of Massachusetts, November 1995.

[2] 首藤一幸, 菅原健一, 浜中征志郎, 村岡洋一. 分散環境を対象とした並列プログラミング環境 MC. 情報処理学会 SWoPP'97(HPC)(to appear), August 1997.