数理解析研究所講究録 1456 巻 2005 年 100-107

コンパイラと数式処理
コンパイラ・インフラストラクチャCOINSの活用
- 構想 -

藤瀬 哲朗
(株) 三菱総合研究所*
TETSURO FUJISE
MITSUBISHI RESEARCH INSTITUTE, INC.

西岡 利博
(株) 三菱総合研究所†
TOSHIHIRO NISHIOKA
MITSUBISHI RESEARCH INSTITUTE, INC.

渡邊 坦
電気通信大学‡
TAN WATANABE
THE UNIVERSITY OF ELECTRO-COMMNUNICATIONS

1 動機

数式処理系は数値計算の前処理として使われることがある。数値計算自身は、簡単な計算の場合、記号処理系で計算を実施するケースもあるが、それなりの計算量がある場合は、プログラム生成を行い、実際に計算を実施するマシン向けのコンパイラを活用することとなる。多くの場合、中間生成物のプログラムを得ることは副次的なことであり、数式からコード生成ができれば、人間が記述するために用意されているプログラミング言語処理系の制約から解放される。

我々はコンパイラの研究開発を支援するために、コンパイラを構成する部品群を Java 言語で開発している。この部品群のことを COINS (COmpiler INfraStructure) と呼んでいる。コンパイラの研究開発は非常に時間がかかるため、この研究インフラを活用するすることで目的とする研究の実施が著しく容易となる。

コンパイラを構成するための様々な機能を実現してみると、数式処理 (数式記号処理の方が的を射ているかもしれない) と共通する部分、特に数式の標準化と最適化などの部分で、数式処理に期待したい部分があることに気付かされる。数式処理系を部品として併せることができれば、さらなる研究インフラとして充実することができ、より高級なコンパイラが容易に開発できる可能性がある。特に Symbolic Analysis 系の研究結果をコード生成まで行って実証することができれば、既存の研究のうち実際的で有効な研究成果を見出すことができ、最適化技術もさらにすすむ可能性がある。

一方かつて汎用機が中心であった時代はある意味でアーキテクチャがユーザから遠かった時代とも言え、多様なコード生成ができる必要もなかった。現在は PC、WS、並列計算機までマイクロプロセッサで構成される時代となり、アーキテクチャが非常に身近になってきている。特に組込みシステム向けプロセッサは

*fujise@mri.co.jp
†nishioka@mri.co.jp
‡tan@cs.uec.ac.jp

プログラマからの距離が非常に近く、コンパイラの最適化によりプログラマが意図しないコードが生成されることにも対応する必要があるため、コード生成の方式について精通しているプログラマを必要としている。

研究者、技術者がコード生成に関わることが可能な時代なはずだが、実は可能であるだけで、実際にそのようなことは少ない。アーキテクチャが身近であっても、コンパイラ、特にコード生成部が身近ではないことが原因ではないだろうか。コード生成についてはよい教科書がないこともその一因であると思う。COINS はコード生成部が非常にわかり易いコンパイラ・インフラストラクチャである。筆者らは COINS を使えば、数式処理系とコード生成が直結することが可能になり、両者の研究を融合することができると考えており、実証することを目指している段階である。本稿では、COINS を数式処理分野での利用が可能であることを説明するために、まず COINS について説明し、数式処理とコンパイラを融合できる例を挙げ、最後に補足する。

2 コンパイラ・インフラストラクチャCOINSの概要

COINS の目的は次のとおり。

- 新しいコンパイラ方式を容易に実験、評価できるような、コンパイラの共通インフラストラクチャを開発する。
- コンパイラに関する研究用基盤、教育用基盤、ならびに企業でも利用可能な基盤を目指す。

図 1 に COINS システム全体を示す。

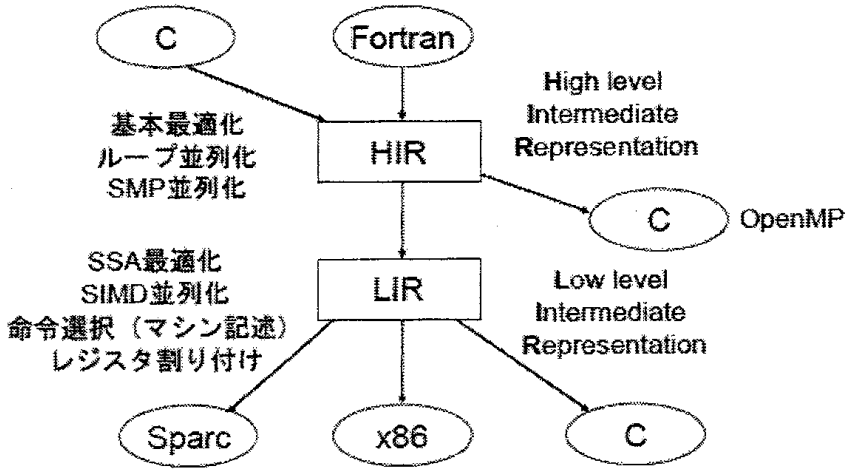

図 1: COINS システム

この COINS システムの特徴は次のとおりである。

- 2 レベルの中間表現
  ソース言語レベル（ HIR ）と機械語レベル（ LIR ）の 2 階層からなる。

- リターゲッタブルなコード生成系
  Sparc, Intel x86 および SIMD 対応の x86 のコード生成系は用意されている。ARM、PowerPC、SH4、MIPS のマシン記述の試験実装もある。

- 複数ソース言語対応
  C と Fortran 77 を用意している。Java を開発中である。

- SSA 最適化に関するモジュール完備
  部分冗長性除去の新方式を開発し実装した。

- SIMD 並列化の方式を開発

- 基本最適化、ループ並列化、粗粒度タスク並列化の基本機能
  これらの並列化機能は OpenMP 拡張 C プログラムに変換して生成することで並列化を実現している。

- コンパイラ・ドライバによる種々のコンパイラへの適応

COINS システムの複数言語および複数機種の対応について、具体的に説明すると以下のとおりである。

- 言語・機種非依存な中間表現の設定

  HIR: ほとんど言語・機種非依存
  依存情報は SourceLanguage, MachineParam クラスに封じ込めている。

  LIR: HIR-to-LIR 変換直後はほとんど言語・機種非依存
  命令選択、レジスタ割付後は機種依存だが文法は非依存で処理は共通化可能である。

- マシン仕様記述に基づくリターゲット型コード生成

- 構成要素を組み合わせ可能にするコンパイラドライバ

後者 2 つについては後述する。これらの特徴を活かした新しいコンパイラの開発方法は次のとおりである。新しいマシンのコンパイラを作るには、マシン記述ファイルだけ作ればよい。その上で最適化を実現し、ドライバのオプションで指定すればよい。新しい言語についてはソース言語から HIR への変換プログラムだけ開発すればよい。また新しい最適化機能は標準ドライバを変更して組み込み、またコンパイラのデバッグ、各種のトレース情報を規定とおり加えた上で、実行時にはドライバのオプションで指定すればよい。

コンパイラ・インフラストラクチャとしてすべてを完備しているわけではない。現状用意している機能は次のとおりである。なお HIR および LIR について C 言語変換や Visualizer が用意されている。

- HIR レベルの機能
  - 基本最適化
    制御フローグラフの作成、データフローの解析、別名解析、定数の畳み込み・伝播、共通部分式削除、無用命令削除
  - 並列化
    ループ解析・並列化、粗粒度並列化
  - C 出力
    OpenMP 出力

- LIR レベルの機能

- SSA 最適化
  LIR → SSA 形式 (3 種類の変換法)、各種の SSA 最適化、SSA 形式→ LIR (2 種類の変換法)
- SIMD 並列化
  マルチメディア用特殊命令を利用した目的コード生成
- リターゲット型コード生成
  マシン記述による命令選択、干渉グラフによるレジスタ割付

ループ並列化機能はループ解析機能と並列化機能に分けている。HIR の解析結果に基づいて並列化指示による並列化プログラムを生成する。並列化プログラムとして OpenMP 拡張 C 言語プログラムを生成する。粗粒度並列化も HIR を対象として、同様に OpenMP 拡張 C 言語プログラムを生成する。一方 SIMD 並列化は命令レベルで並列化を行うため、LIR を対象とする。SIMD 並列化の例として、2 つの short 配列の平均の計算に対応するコードを図 2 に示す。

```
#define AVE(x,y)
(((x)>>1)+((y)>>1)+(((x)|(y))&1))

int func(short *a, short *b, short *c)
{int i;
 for (...;i+=8) {
  a[0] = AVE(b[0],c[0]); a[1] = AVE(b[1],c[1]);
  ......
  a[6] = AVE(b[6],c[6]); a[7] = AVE(b[7],c[7]);
  a += 8; b += 8; c += 8;
 }}
```

for IA-32/MMX

```
movq (%edi),%mm1 # %edi = *b   psraw $1,%mm1
movq (%esi),%mm2 # %esi = *c   psraw $1,%mm2
movq %mm2,%mm3                 paddw %mm1,%mm3
por  %mm1,%mm3                 paddw %mm2,%mm3
pand %mm0,%mm3                 movq %mm3,(%eax) # %eax = *a
```

図 2: SIMD 並列化の例

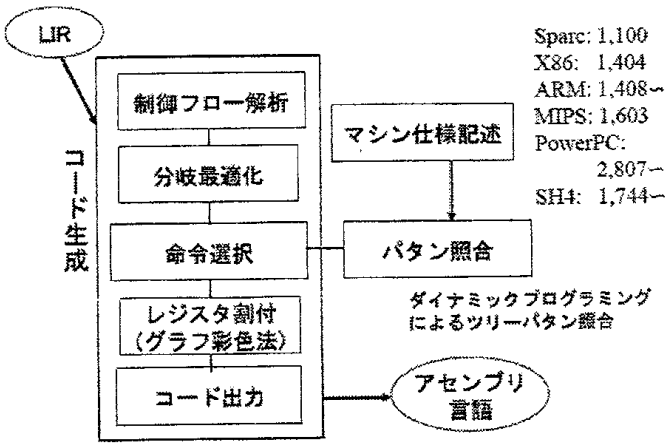

図 3: リターゲット型コード生成

本例は x86 の MMX を活用したコードを生成することできるため、高い性能が得られる。
またリターゲット型コード生成の構成を図 3 に示す。
これらのコードは、マシン仕様記述に従って生成される。マシン仕様記述の例を図 4 にそれぞれ示す。

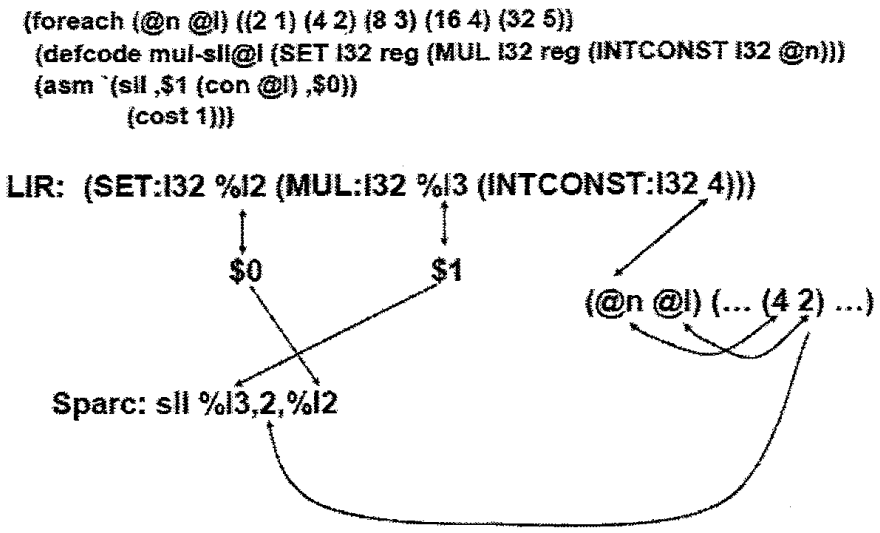

図 4: マシン記述の例

3 COINS を使ったコンパイラの作り方

COINS のインフラは豊富なモジュールを持っている。Visualizer などもあるが、全体を Java で開発しているため、Java の開発環境を活用することができる。そのなかでコンパイラを実際に開発した者にしか実感できないものであるが、コンパイラ・ドライバは全体を統合化・制御するものとして開発環境において重要な機能である。現在インフラに用意している C コンパイラは、豊富なオプション指定 (対象機種とその環境も) をもつが、そのサブクラスとして一部のモジュールをオーバライドして新コンパイラを開発することができる。Fortran ドライバは 80 行 (フロントエンド呼出しをオーバライド) で作成することができた。本コンパイラ・ドライバの特徴は次のとおりである。

- ドライバ実装とドライバ API の分離
- 標準コマンドラインドライバ実装
- gcc との互換性、容易にカスタマイズ可能
- 継承による多言語拡張を意識した実装
- コンパイル過程に 5 箇所のプローブタイミングを用意
- ドライバ API の提供機能
  前処理、コンパイラ、アセンブラ、リンカ呼出し制御、サフィックス解釈規則 (カスタマイズ可能)、オプション API (カスタマイズ可能)、トレース API、警告 API (警告メッセージ出力の制御)、サイト (およびユーザ) コンフィグレーション読込み

コンパイラの基本フローは図5のとおりであり、このプロセスをコンパイラ・ドライバを中心とした機能が支える。

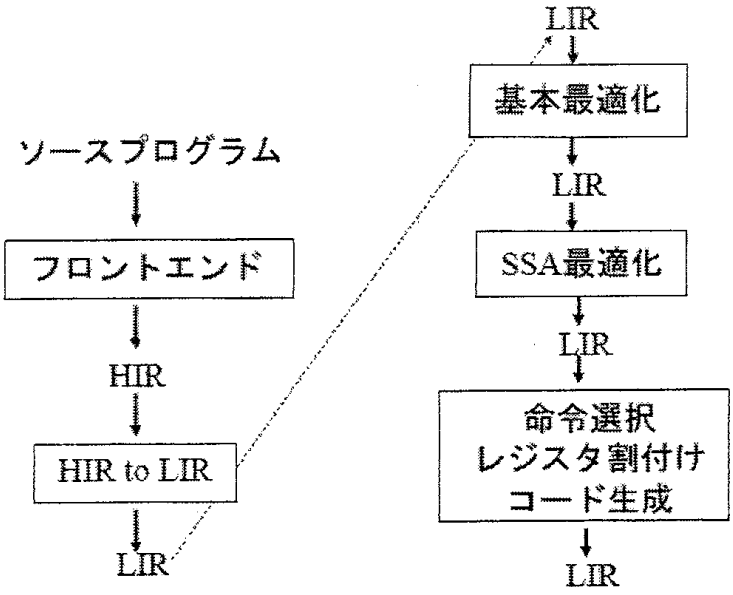

図 5: 基本フロー

コンパイル過程を可視化する Visualizer は、フロー・グラフ、支配木、HIR/LIR との対応表示マクロフロー・グラフの表示と HIR との対応が可能である。一種のコード生成過程の可視化が可能となっている。また Eclipse と連携するツール開発も考えられる。

4 数式処理からコンパイラ部品の利用例

数式処理からコンパイラ部品を活用する例を示す。

(1) 数式のコード化例

数理モデルを形成してそのままベクトル計算機で計算するシステムとして、偏微分方程式向き数値シミュレーション言語 DEQSOL ([1]) が挙げられる。DEQSOL は定式化して得た数理モデルから問題定義記述と解法記述を行い、その記述からベクトル計算機向けプログラムおよびコード生成を行うことができるものである。ベクトル化はベクトル計算機メーカーならではの特別なノウハウを活用できるため、より効率的なコードを得られる。もちろん専用パッケージのデメリットも存在するが、発想は非常に面白いものである。

DEQSOL 風の機能を COINS を活用して実現すると次のようになる。

- 行列計算などへの定式化後、コード生成を行う。
- コード生成に自分なりの最適化を加える。
- 並列化解析による並列化、 SIMD 並列化による SIMD 命令化などによる対応を行う。

(2) 高速計算法での活用
高速計算法による数式変換後、実際のコード生成を行い、実証評価する。例えば、キャッシュサイズを意識した高速計算法による数式変換後、データ配置までコミットしたコード生成を行い、実証評価する。

5 コンパイラから数式処理の利用例

逆に数式処理からコンパイラ部品を活用する例を示す。

(1) コンパイラの最適化・並列化研究での活用
ループ内の配列要素の依存解析・領域解析では、Diophantine 方程式、区間解析、除算の乗算化などを必要とする。コンパイラを開発するために実装するのではなく、最新の成果を盛り込んだ数式処理系を活用する。

(2) canonical form の活用 (パターンマッチ等)
数式の正規化、ループの標準化は最適化、並列化で必要とされる。特にループの標準化の研究は Symbolic Analysis 分野ではいくつか研究されているようである。数式処理側では [3] などで長年研究されている理論の応用もあり、活用が考えられる。

(3) 数式処理を使った省電力組込みソフトウェアの最適化
[2] では省電力化を目的に、三角関数などの基本関数の級数展開をコンパイル時に行い、予想される必要精度に合わせた級数展開系の乗算などから数式の最小化を試み、生成されたコードを基にシミュレータにかけてメモリアクセスを中心とした消費電力を求めて省電力化ソフトウェアの最適化を行っている。プログラム変換もしくは部分計算のひとつとして数式処理を活用した典型例である。つまり、

$$y = cosx + cos2x + cos3x$$

は、

$$y = 1 - 1/2x^2 + 1/24x^4 + 1 - 1/222x^2 + 1/2424x^4 + 1 - 1/232x^2 + 1/2434x^4$$

となり、

$$y = 3 + (-7 + 49/12x^2)x^2$$

と数式処理による最適化を行って、この数式を命令に変換し、電力消費量を計算して最適化する。

(4) ループのポインタ配列アクセス変換による DSP 向け並列化
ポインタアクセスのあるループの並列化は困難であるといわれている。そこで Symbolic Analysis の手法を使ってポインタアクセスとループを標準形式化に変換した上で、配列表現に逆変換、一般のループ最適化・並列化へと変換することができる ([3])。

また特殊な区間演算を活用することでループ内の if 文を除去して、ループ最適化・並列化することも可能である。

6 まとめと補足

数式記号処理の成果を容易にコード生成まで適用することで研究分野が広がる。コンパイラ側から見ると数式処理系が次の機能を持っているとインフラ側がさらに軽いものになる。

- メモリ効率の良い集合と集合演算処理
- グラフ・ネットワークと可視化
- 整数計画法などの最適化

さらに数式処理系はロジックをサポートできると Symbolic Analysis や型理論を使ったコード生成の研究など用途が広がると思われる。このあたりの研究が進むことも期待したい。

参 考 文 献

[1] 梅谷征雄, 辻みちる, 岩澤京子: 数値シミュレーション言語 DEQSOL, 情報処理学会論文誌 26 巻 1 号, pp.168-180 (1985).

[2] Peymandoust, A., Simunic, T. and De Micheli, G.: Low Power Embedded Software Optimization using Symbolic Algebra, Proc. of 2002 Design, Automation and Test in Europe Conference and Exhibition, pp.1052-1059 (2002).

[3] van Engelen, R.A. and Gallivan, K. A.: An Efficient Algorithm for Pointer-to-Array Access Conversion for Compiling and Optimizing DSP Applications, Proc. of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp.80-89, 2001.