5E−8

RISCプロセッサ向け最適化コンパイラの構成要件

堀田 耕一郎, 林 正和, 渡辺 照洋, 滝内 政昭
富士通 株式会社

1. はじめに

昨今のダウンサイジングの流れの中で，ワークステーションの CPU性能は上昇の一途を辿り，従来の汎用大型機の性能に匹敵するレベルに到達している．

この流れの原動力としてマイクロプロセッサの進歩が挙げられる．特にRISCアーキテクチャの採用に始まる CPUの進歩が著しい．しかし，RISCアーキテクチャはコンパイラによる最適化を前提としているという特徴があり，ワークステーションの性能を向上させるためには，コンパイラの最適化能力を高めることが，極めて重要である．

本稿では，RISCアーキテクチャの性能を十分に引き出す最適化機能を持ったコンパイラの構成要件を考察し，その評価結果を報告する．

2. RISCプロセッサの特徴

RISCという名の通り命令の種類が少ないため，CISCと比べて命令数が増える傾向にある．基本的に命令は1クロックサイクルで実行を完了するが，メモリアクセスを行うロード/ストア命令は，複数クロックサイクルを要することがあり，この傾向はプロセッサの進化に伴って増大することが予想できる．メモリアクセスを行う命令が，ロード/ストア命令に限定されるので，ロード/ストア命令に対する最適化を確実に行ない，レジスタを効率的に使用してメモリアクセスの頻度を下げることが重要である[1]．

また，スーパスカラ，VLIWといった複数の命令を並列に動作させることにより高速化を図る手法が採用されており，これを活かすために命令スケジューリングを行い，並列度を高めることが必要である[2]．

3. 最適化の要件と問題点

RISCアーキテクチャのハードウェア資源を有効に使うことのできるオブジェクトコードを生成するためには，ハードウェア命令と 1 対 1 になる中間コードを最適化対象にすることが望ましい．特に，命令スケジューリングを行って個々の命令を並列実行させるには，中間コードはハードウェア命令と 1 対 1 であることが必須と言える．

しかし，中間コードをハードウェア命令と 1 対 1 にすると，中間コードの量が増えて最適化のオーバヘッドが増えたり，プログラムの意味情報が欠落して最適化ができなくなったりして，他の最適化に悪影響を与えることが考えられる．特に，図1.に示したような構造体データのアクセスにおいては，二つのデータ 'x.a' 及び 'x.b' の重なり関係をソースプログラムから判定することは容易であるが，アドレス情報だけから判定するには，変数 'i' の値の変動範囲を認識する必要があり，最適化が不可能になる場合が多い．

```
struct {
  char  a[16] ;
  char  b ;
} x ;

x.a[i]=x.b*2 ;
```

```
ld   [x+16],r0
add  r0,r0,r1
st   r1,[x+i]
```

(1)ソースコード　　(2)命令レベルの中間コード

図1. 構造体データのアクセスの表現

また，RISCプロセッサの進歩には著しいものがあるので，ソース解析部で生成する中間コードをハードウェアに合わせて作ると，プロセッサの世代交代にコンパイラが追随しにくくなるなど，プログラム開発効率上の弱点にもなる．

4. コンパイラ構成の決定

上記問題点を考慮し，RISCプロセッサ向けのコンパイラ構成を以下のように決定した．(図2.)

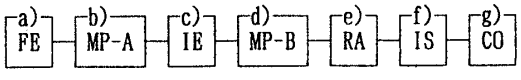

図2. RISCプロセッサ向けコンパイラ構成

a) FE : Front End

ソース解析部が生成する中間コードとハードウェ

Optimizing Compiler Structure for RISC processor.
Kohichiro Hotta, Masakazu Hayashi, Teruhiro Watanabe, Masaaki Takiuchi
FUJITSU LIMITED

ア命令との対応関係は 1 対 N であり，ハードウェア依存性が低い．

b) MP-A : Middle Pass-A

(a)の中間コードに対してプログラム構造を変更する大規模な最適化及び基本的な最適化を行い，基本性能を向上させる．

c) IE : Instruction Expansion

最適化の後に中間コードを変換し，ハードウェア命令との対応関係を 1 対 1 にする．

d) MP-B : Middle Pass-B

ハードウェア命令と 1 対 1 になった中間コードに対して再度最適化(b) で行った基本的な最適化を施す．

e) RA : Register Assignment

f) IS : Instruction Scheduling

g) CO : Code Out

このコンパイラにおいて，(c)のIEフェーズとその前後に置かれたMiddle Pass(最適化)フェーズの存在が，性能向上のために本質的である．IEフェーズ前の中間コードでは，ソースプログラムに近いレベルの情報を持っているため，マクロレベルの最適化を行ない易いが，命令レベルの最適化は行ない難い．そこで，IEフェーズ後のハードウェア依存性の高い中間コードに対して，きめ細かく最適化し直すことで，オブジェクトコードの質を高めている．

IEフェーズが対象とする中間コードは，ハードウェア依存性の高いものであるが，その表現形式をIEフェーズ前における中間コードと同等のものにしたため，最適化の手続き自体はMP-A，MP-Bともにハードウェア無依存に開発することができた．

さらに，MP-Bでの最適化の過程においてレジスタの存在を意識することによって，レジスタ割り付けの負荷を軽くすることも可能になった．

また，命令スケジューリングを行うためには，ハードウェア命令と 1 対 1 に対応した中間コードが必須であるが，複数のメモリアクセスの依存関係を解析するには，ハードウェア命令レベルでのアドレス計算を元にするのでは困難であり，ソースプログラムレベルの情報を使うことが望ましい．そのために，IEフェーズの前に依存関係の解析を済ませ，その情報をISフェーズに引き継いでいる．

5. 性能評価結果

図3.は，当コンパイラを用いた性能評価結果である．大域データフロー情報を使用した共通式の除去，不変式の移動などの基本的な最適化を高レベルの中間言語(MP-A)，及び低レベルの中間言語(MP-B)に対してそれぞれ施し，その性能を評価した．インライン展開のようなプログラム構造を変更するような大規模な最適化やPeephole最適化は，全ての場合において動作させている．

コンパイラ内で異なるレベルの中間コードに対して2回の最適化を行うことにより，大きな性能向上が実現できていることが判る．基本的にはソースプログラム情報が多く残っているMP-Aの段階での最適化の効果が高いが，Dhrystone2.1では手続き呼出し等で ABIを意識した最適化が効果的なため命令レベルの中間コードに対する最適化(MP-B)の効果が高いことがわかる．

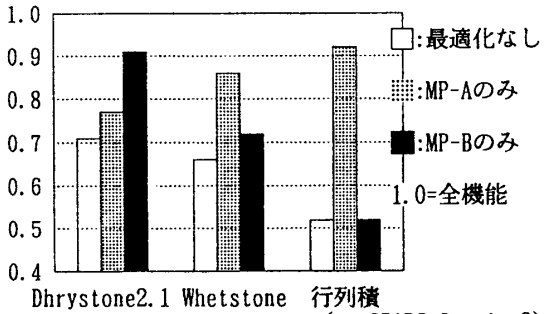

図3. 最適化タイミングと性能

6. まとめ

今回，RISCプロセッサの特性を意識した最適化コンパイラの基本構成について考察し，それをもとにRISCプロセッサの性能を引き出すことのできるコンパイラを開発した．その結果，2段階の最適化が効果的であることを確認することができた．また，中間コードの表現形式をハードウェア依存性の低いものにしたため，ハードウェア依存性の強い手続きを局所化することができた．この結果，コンパイラの多ターゲット展開を容易にすることが可能になった．

今後，RISCプロセッサを使用したワークステーションの性能をさらに発揮させるには，このような構成のコンパイラに，グローバル命令スケジューリングのようなきめ細かい最適化技術やキャッシュメモリの有効利用を図るブロッキング技術のようなプログラム構造の変更を伴う大規模な最適化技術の適用が必要である．従来，スーパコンピュータの分野で適用されていた技術も応用して，このような新しい最適化技術を加え，各種RISCプロセッサ向けの高性能なコンパイラを開発していく所存である．

〔参考文献〕

(1)Hennessy, J., Patterson, D. :"*Computer Architecture : A Quantitative Approach*", Morgan Kaufmann Publishers, Inc., 1990

(2)林，他 「RISCプロセッサ向け命令スケジューリング」情報処理学会第46回全国大会予稿集，5E-7，1993