プログラム実行監視システム用コンパイラとシミュレータについて*

1B－9　合原 正男　内田 美穂子　藤原 譲　宇都宮 公訓†

筑波大学‡

1　はじめに

シンボリックデバッガなどでのプログラム実行監視の速度を向上させるため、著者らは多重プロセッサの“多重”の利点を生かしたプログラム実行監視システムを開発中である。このシステムでは、メモリを共有する２台のプロセッサ系を想定している。一方のプロセッサはE-Processorといい、通常の意味でのプログラムの実行を行なう。他方のプロセッサはM-Processorといい、E-Processorの実行を監視する。M-Processor上で実行されるコードはM-codeといわれ、E-Processor上で実行されるコードはE-codeといわれる。

この監視システムは

1. ステートメント実行前中断
2. 変数参照中断
3. 変数更新中断
4. 値中断

等の中断条件の成立を監視し、

5. トータルステートメント実行回数
6. ステートメント実行回数

等を測定する。

M-code,E-codeは専用のコンパイラを用いて生成する。プログラミング言語はPascalとし、コンパイラはPysterのLL(1)コンパイラ[1]を改造して作った。このコンパイラと、性能評価のために開発したシミュレータについて報告する。

2　パーサとアクションルーチン

生成規則の一部を次に示す。選択集合は省略している。

```
EXEC_STMT  → m_code('c','increment','@stc')
             insert('c','m',gen('stmt_ctr',x),counter,mutable,0,yes,1)
             m_code('c','increment',x) ASSIGN_STMT
           → m_code('c','increment','@stc')
             insert('c','m',gen('stmt_ctr',x),counter,mutable,0,yes,1)
             m_code('c','increment',x) IF_STMT
           → WHILE_STMT
           → REPEAT_STMT
           → NULL_STMT
           → BEGIN_END_STMT

IF_STMT    → 'if' EXPRESS 'then' pop_e_operand(x)
             e_code('u','signal(control)',x) e_code('c','wait(start)')
             m_code('u','wait(control)') m_code('c','check(cnd_break)')
             m_code('c','check(break)') m_code('c','signal(start)')
             e_code('u','then',x) m_code('u','then',x)
             EXEC_STMT ELSE_PART
ELSE_PART  → 'else' e_code('u','else',pop_e_operand)
             m_code('u','else',pop_m_operand) EXEC_STMT
             e_code('u','post_if',pop_e_operand)
             m_code('u','post_if',pop_m_operand)
           → λ pop_books e_code('u','post_if',pop_e_operand)
             m_code('u','post_if',pop_m_operand)

WHILE_STMT → e_code('u','while') m_code('u','while',)
             m_code('c','increment','@stc')
             insert('c','m',gen('stmt_ctr',x),
                                  counter,mutable,0,yes,1)
             m_code('c','increment',x) EXPRESS 'do' pop_e_operand(x)
             e_code('u','signal(control)',x) e_code('c','wait(start)')
             m_code('u','wait(control)') m_code('c','check(cnd_break)')
             m_code('c','check(break)') m_code('c','signal(start)')
             e_code('u','do',x) m_code('u','do') EXEC_STMT
             e_code('u','post_while',pop_e_operand,pop_e_operand)
             m_code('u','post_while',pop_m_operand,pop_m_operand)
```

m_code()はM-codeを、e_code()はE-codeを生成する手続きである。これらの手続きの第１引数(a1)は‘u’か‘c’のいずれかであり、‘u’のときは必ずコードを生成するが、‘c’のときは、監視指定がある場合だけコードを生成する。手続きinsert()の第１引数(a1)も‘u’か‘c’のいずれかである、‘u’のときは必ず記号表に記入するが、‘c’のときは、監視指定がある場合だけ記号表に記入する。

e_code(a1,‘wait(start)’) wait(start)命令を生成する

m_code(a1,‘signal(start)’) signal(start)命令を生成する

e_code(a1,‘signal(control)’,x) x中の真偽の値をM-codeに渡すsignal(control)命令を生成する

m_code(‘wait(control)’) signal(control)命令で渡された値を受け取るwait(control)命令を生成する

m_code(‘check(break)’) 成立した中断が中断条件成立キュー中にあるかどうかチェックし、あれば中断を発生させる

m_code(‘check(cnd_brk)’) cnd_brkが指定されていれば、その成立を監視し、成立していれば、中断条件成立キューに記入するM-codeを生成する

insert(a1,‘e’,a3,…) a3をE-code用の記号表に登録する

gen(‘stmt_ctr’,x) ステートメント実行回数カウンタ変数を１個生成し、それを関数値として返すとともに、xにも入れる

m_code(a1,‘increment’,x) xの値を１増やすコードを生成する

push_e_operand(x) xをe_operand_stack(E-code用のオペランドスタック)に積む

pop_e_operand(x) e_operand_stackから要素を１つ降ろしてこの関数の値とする

e_code(a1,‘then’,x) レジスタ管理表のコピーを作り、books_stackに積む。新しいラベルを１つ生成し、それをe_operand_stackに積むとともに、xの値が偽であれば、そのラベルをもつ命令にジャンプする命令を生成する

---

*On a compiler and simulator for a program monitoring system

†Masao AIHARA, Mihoko UCHIDA, Yuzuru FUJIWARA, Kiminori UTSUNOMIYA

‡University of Tsukuba

m_code(a1,'then') 新しいラベルを1つ生成し、それをm_operand_stackに積むとともに、wait(control)で受け取った値が偽であれば、そのラベルをもつ命令にジャンプする命令を生成する

e_code(a1,'else',a3) ifステートメントの終りを示す新しいラベルを1つ生成し、そのラベルに無条件にジャンプする命令を生成する。さらに、そのラベルをe_operand_stackに積む。a3をラベルとするNOP命令を生成し、books_stackからレジスタ管理表を降ろす

m_code(a1,'else',a3) ifステートメントの終りを示す新しいラベルを1つ生成し、そのラベルに無条件にジャンプする命令を生成する。さらに、そのラベルをm_operand_stackに積み、a3をラベルとするNOP命令を出力する

e_code(a1,'post_if',a3) a3をラベルとしてifステートメントの終りに当たるNOP命令を生成し、レジスタ管理表をクリアする。

e_code(a1,'while') ラベルを1つ生成し、e_operand_stackに積むとともに、それをラベルとするNOP命令を出力する。レジスタ管理表をクリアする

m_code(a1,'while') ラベルを1つ生成し、m_operand_stackに積むとともに、それをラベルとするNOP命令を出力する。

e_code(a1,'do',x) ラベルを1つ生成し、e_operand_stackに積むとともに、xの値が偽であればそのラベルにジャンプする命令を生成する

m_code(a1,'do') ラベルを1つ生成し、m_operand_stackに積むとともに、wait(control)で受け取った値が偽であればそのラベルにジャンプする命令を生成する

e_code(a1,'post_while',a3,a4) ラベルa4に無条件にジャンプする命令、a3をラベルとするNOP命令をこの順に生成する。m_codeも同様である

本システム用にPysterのコンパイラを改造することは容易であった。その理由としては次のようなことが挙げられる。

- 生成規則については、LL(1)文法としての部分は変わらず、アクション記号に関する部分だけの変更で済む
- E-codeのアクション記号の位置から判断して、M-codeのアクション記号を挿入すべき位置が容易に推察できる
- M-codeを生成する手続きとE-codeを生成する手続きでデータの共有がない
- M-codeとE-codeの同期のとり方が単純で、相互干渉がほとんどない
- M-codeは通常の意味でのプログラムの実行は一切しない

3 シミュレータ

2章で述べたコンパイラが生成したコードを動かす実機がないので、シミュレータを開発し、性能を評価する。このシミュレータはオブジェクト指向開発パラダイムで設計し、C++でコーディングした。そのオブジェクトモデル図[2]を図1に示す。

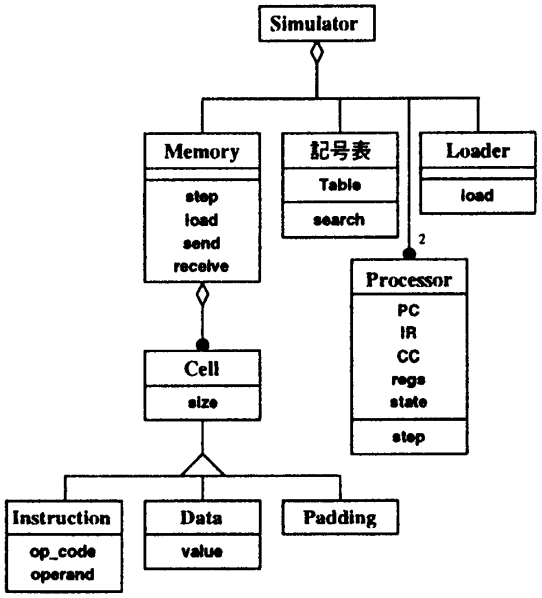

図 1: シミュレータのオブジェクトモデル

4 おわりに

監視プロセスと実行プロセスの間でかなり頻繁に同期をとる必要があるが、それでも本システムによってプログラム監視の性能が向上することが確認された。並行プログラムであれば、その中の1プロセスとして監視プロセスを組み込むことができ、監視プロセスによるオーバヘッドは逐次プログラムの場合よりずっと改善される。

今後、同じアプローチで高速アルゴリズムアニメーションシステムを開発することを計画している。

参考文献

[1] Pyster,A.B.: Compiler Design and Construction,Van Nostrand Rein hold(1980), (邦訳) 松尾正信: コンパイラの設計と構築, 近代科学社 (1983)

[2] James Rumbaugh: Object-Oriented Modeling and Design,Prentice Hall(1991), (邦訳) オブジェクト指向方法論 OMT, トッパン (1992)