ICOT Technical Memorandum: TM-0887

TM-0887

メタインタプリタを用いた パフォーマンスモニタ

田口 毅, 本城 哲, 中島俊介(沖)

May, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

メタインタプリタを用いたパフォーマンスモニタ

Performance Monitor using Metainterpreter

田口 毅† 本城 哲‡ 中島俊介‡
Takeshi TAGUCHI Akira HONJO Shunsuke NAKAJIMA

†沖電気工業(株) ‡沖通信システム(株)
Oki Electric Industry Co.,Ltd. Oki Telecommunication Systems Co.,Ltd.

概要

KL1プログラムの設計支援システムにおけるプラグマ設定支援ツールとして,メタインタプリタを用いたパフォーマンスモニタについて報告する.本稿では,メタインタプリタを用いて,プラグマ設定に必要な負荷情報の収集を行い,収集された負荷情報を三次元グラフとして表示するまでの概要を示す.ここでメタインタプリタは,ゴールキューに存在するゴール数を,負荷量として扱った.さらに今回は,現在作成中である並列メタインタプリタの検討及び,パフォーマンスモニタにおける拡張イメージを示す.

1 はじめに

我々は,KL1プログラムの設計支援システムを開発中である.本システムは,プログラムの検証,デバッグ,プラグマ設定等の支援を目的としている.今回は,プラグマ設定を支援するツールとしてパフォーマンスモニタを紹介する.本ツールは,KL1プログラムを実行した時の,各プロセッサ負荷の推移を測定するものであり,負荷に関する情報を収集する手段として,メタインタプリタを利用している.

本稿では,GHCメタインタプリタに単純な拡張を行うことにより,マルチプロセッサエレメント環境が非常に簡易にエミュレート可能であるということを示した後,これを用いた動作情報収集について述べる.また,メタインタプリタを用いて収集した情報をパフォーマンスモニタを用いて表示し,負荷分散状況を確認する.

2 メタインタプリタによる動作情報収集

2.1 GHC メタインタプリタ

以下にGHCメタインタプリタを示す.このメタインタプリタは,ゴールキューとメタレベルへのトレース情報出力用ストリームを持ち,ゴールのリダクションを試みるたびに実行のトレース情報を出力する.

```
mi(Clauses,Goals,Reports) :- true |
    freeze_term(Clauses,0,VarID1),
```

```
        freeze_term(Goals,VarID1,VarID2),
        put_goals(Goals,[],GoalQueue),
        pes(Clauses,Goals,GoalQueue,VarID2,[],Reports).

    pes(_,Goals,[],_,BindTBL,Reports) :- true |
        eval_term(Goals,EGoals,BindTBL),
        Reports=[exit(EGoals)].
    pes(Clauses,Goals,GoalQueue,VarID,BindTBL,Reports) :- true |
        get_goal(GoalQueue,GoalQueue1,BindTBL,Goal),
        exec_body(Goal,RGoals,VarID,VarID1,BindTBL,BindTBL1,Clauses,Result),
        eval_term(GoalQueue,EGoalQueue,BindTBL),
        eval_term(Goal,EGoal,BindTBL),
        eval_term(RGoals,ERGoals,BindTBL1),
        Reports = [goal_queue(EGoalQueue),{Result,EGoal,ERGoals} | Reports1],
        put_goals(RGoals,GoalQueue1,GoalQueue2),
        pes1(Result,Clauses,Goals,GoalQueue2,VarID1,BindTBL1,Reports1).

    pes1(fail,_,_,_,_,_,Reports) :- true |
        Reports = [].
    otherwise.
    pes1(_,Clauses,Goals,GoalQueue,VarID,BindTBL,Reports) :- true |
        pes(Clauses,Goals,GoalQueue,VarID,BindTBL,Reports).
```

述語 mi(Clauses,Goals,Reports) は, メタインタプリタのトップレベルのゴールであり, Clauses にプログラム, Goals に起動ゴールを与えて呼び出すと, 報告ストリーム Reports にトレース情報を返す. このメタインタプリタでは全ての変数は, 述語 freeze_term(X, VarID, NewVarID) によりフリーズされた形で扱われる. X はフリーズする入力項であり, freeze_term/3 は X に含まれる未定義変数全てにベクタ $var(ID) をユニファイする. また, フリーズ処理が終了すると, NewVarID に ID を更新した値をユニファイする. ID は変数識別番号 VarID から始まる整数であり, 変数毎にユニークな変数識別子である. 述語 mi/3 では, プログラム Clauses, 及び起動ゴール Goals を freeze_term/3 によりフリーズしている. また述語 put_goals(Goal,GoalQueue, NewGoalQueue) は, ゴールキュー GoalQueue にゴール Goal を格納し, 格納済みのゴールキューを NewGoalQueue にユニファイする. 述語 mi/3 内では, キューの中に起動ゴール Goals を格納している.

述語 pes(Clauses,Goals,GoalQueue,VarID,BindTBL,Reports) は, 実際にゴールの評価を行なう述語である. Clauses,Goals は freeze_term/3 によってフリーズされたプログラム, 起動ゴールであり, GoalQueue はゴールキュー, VarID は変数識別番号, BindTBL は変数の束縛表, Reports は報告ストリームである.

pes/6 は GoalQueue が空リストになるまでゴールの評価を繰り返す. 先ず, get_goal(GoalQueue, GoalQueue1,BindTBL,Goal) により評価すべきゴールをゴールキューより 1 つ取り出し, 変数 Goal にユニファイする. GoalQueue1 には Goal が取り除かれたゴールキューがユニファイされる. 述語 exec_body(Goal,RGoals,VarID,VarID1,BindTBL,BindTBL1, Clauses,Result) は, ゴール Goal のリダクションを GHC の実行規則に従って試みる. すなわち, ゴール Goal とパッシブユニフィケーション可能な節がプログラム Clauses 内にあるならば, これを試み, 成功する節があるならばその節にコミットし, そのボディ部をリダクションの結果生成された子ゴールとして変数 RGoals に, コミット成功を示すアトム succ を変数 Result にユニファイする. この変数 RGoals

内のゴールは, 述語 put_goal/4 でゴールキューに格納される. 一方, 全ての可能節に対するパッシブユニフィケーションがサスペンドするならば, 変数 RGoals に [Goal] を, 変数 Result にリダクションがサスペンドしたことを示すアトム susp をユニファイする. また, パッシブユニフィケーションが可能な節のない場合, 及び, 全ての候補節のパッシュブユニフィケーションに失敗した場合は変数 RGoals に空リスト [] を, 変数 Result にアトム fail をユニファイする.

なお, リダクションの結果 Result は述語 pes1/8 で調べられる. pes/8 は Result が fail ならば, 直ちに報告ストリーム Reports を閉じ, メタインタプリタの実行を中断する.

このリダクションを試みる度に述語 pes/6 では, ゴールキュー GoalQueue, ゴールキューから取り出されたゴール Goal, 及び, そのリダクションの結果 RGoal の代入例を述語 eval_term/3 で EGoalQueue,EGoal,ERGoal として求め, リダクションの結果 Result と共にトレース情報として報告ストリーム Reports に流す. GoalQueue からはメタインタプリタで評価を待つゴールを, Goal からはリダクションしようとするゴールを, Result からは, リダクションの成功／失敗／サスペンド, RGoals からは, リダクションの結果生成された子ゴールが何であるかを知ることが出来る.

ゴールキュー GoalQueue が空になった時, 述語 pes/6 は全てのゴールの評価が終了したと見做し, プログラムの実行を終了する. この時 pes/6 は起動ゴール Goals の代入例を eval_term/3 によって求め, 報告ストリーム Reports に返す.

2.2 マルチプロセッサのエミュレート

本システムのように情報を時系列に求めようとする場合, 情報収集のサイクルである“時間”の定義は困難である. もし Multi-PSI 実機のように, 各々のプロセッサエレメントでのリダクションが完全に非同期で行なわれるならば, その出力情報は本システムではまったく意味をなさない. この為, 本システムでは, 各々のプロセッサエレメントでのリダクションは同期して行なわれるものとする. すなわち, 各々のプロセッサエレメントでのリダクションは, 全てのプロセッサエレメントでのメタレベルへの情報出力がされる以前に次のリダクションを開始することはないものとする.

以下にこの様な考えに基づいたマルチプロセッサエミュレート版の述語 pes/8 を示す.

```
pes(NumOfProc,NumOfProc,_,Goals,[],VarID,BindTBL,Reports) :- true |
    eval_term(Goals,EGoals,BindTBL),
    Reports=[exit(EGoals)].
pes(NumOfProc,NumOfProc,Clauses,Goals,GoalQueue,VarID,BindTBL,Reports) :-
    list(GoalQueue) |
    eval_term(GoalQueue,EGoalQueue,BindTBL),
    Reports=[goal_queue(EGoalQueue)|Reports1],
    pes(0,NumOfProc,Clauses,Goals,GoalQueue,
        VarID,BindTBL,Reports1).
pes(ProcID,NumOfProc,Clauses,Goals,GoalQueue,VarID,BindTBL,Reports) :-
    ProcID < N |
    get_goal(ProcID,GoalQueue,GoalQueue1,BindTBL,Goal),
    exec_body(Goal,RGoals,VarID,VarID1,BindTBL,BindTBL1,Clauses,Result),
    put_goals(ProcID,RGoals,GoalQueue1,GoalQueue2),
    eval_term(Goal,EGoal,BindTBL),
    eval_term(RGoals,ERGoals,BindTBL1),
    Reports = [{Result,EGoal,ERGoals}@ProcID | Reports1],
    pes1(R,ProcID,NumOfProc,Clauses,Goals,GoalQueue2,
```

```
            VarID1,BindTBL1,Reports1).
pes1(fail,_,_,_,_,_,_,_,Reports) :- true |
    Reports = [].
otherwise.
pes1(_,ProcID,NumOfProc,Clauses,Goals,GoalQueue,
    VarID,BindTBL,Reports) :- true |
    pes(~(ProcID + 1),NumOfProc,Clauses,Goals,GoalQueue,
        VarID,BindTBL,Reports).
```

述語 pes/8 は, 前述の述語 mi から pes(NumOfProc,NumOfProc,Clauses,Goals, GoalQueue, VarID,BindTBL,Reports) という形で呼ばれる. ここで NumOfProc は全プロセッサ数である. 呼ばれた直後は GoalQueue は空ではないので 2 番目の節がコミットされ, ゴールキューの中身, すなわち起動ゴールの代入例が eval_term/3 で求められ, 報告ストリーム Reports に出力される. この出力の後, プロセッサ識別子 0 番から NumOfProc-1 番までが順に実行される. すなわち, あるプロセッサ識別子 ProcID を持つプロセッサのリダクションを試みたならば, ProcID を 1 インクリメントし, その ProcID を持つプロセッサの行なうリダクションとして実行を試みるのである. ProcID がインクリメントされてゆき, 全プロセッサ数と等しくなったならば, ゴールキューの代入例を報告ストリームに流し, 再びプロセッサ識別子 0 番よりリダクションを試みる.

前述のメタインタプリタと大きく異なるのは, ゴールキューである. このゴールキューではキュー内のゴールに, タグとしてプロセッサ識別子が取り付けられている. 述語 get_goal(ProcID,GoalQueue, GoalQueue1,BindTBL,G) は get_goal/4 に対して, 変数 ProcID で示されたプロセッサ識別子と同じ識別子のタグの付いたゴールのみを返すよう変更されている. またトレース情報の出力においてもどのプロセッサでのリダクションであるかを示すため, 出力情報にタグとして ProcID を付加している.

3 パフォーマンスモニタによる動作情報表示

パフォーマンスモニタにおける動作情報表示部は, Multi-PSI FEP 上で ESP 言語を用いて開発したものである. この表示部は, メタインタプリタから出力される動作情報を一定の形式に変換し, 対象プログラム実行時における, プロセッサ負荷状況の時間的推移を視覚的に表現することを目的としている. ここでメタインタプリタが負荷として扱っているのは, ゴールキュー中に存在するゴール数である.

動作情報の表示は, 全プロセッサの負荷状況を同時に表示するために, 時間 - 負荷 - プロセッサの三次元グラフで示した. また, 目的に合わせた表示を得るために, 三次元コンピュータ・グラフィックス [Sato 84] を用いて表示した. 動作情報を任意の視点で表示し, 変更することを可能とした. さらに, 任意の時間あるいはプロセッサに注目した表示を行うために, 時間 - 負荷及び, プロセッサ - 負荷の各二次元グラフ表示を可能とした.

3.1 三次元表示機能

三次元表示では, 座標原点側をプロセッサ 0 としており, 負荷の最も大きい点を 100% に合わせてスケールを決定している. また, グラフ上の時間は, プロセッサが一つでも立ち上がった時点を 0 とし, 全てのプロセッサに負荷がなくなるまでを表している.

メタインタプリタから収集した負荷情報を三次元表示した例を図 1. に示す.

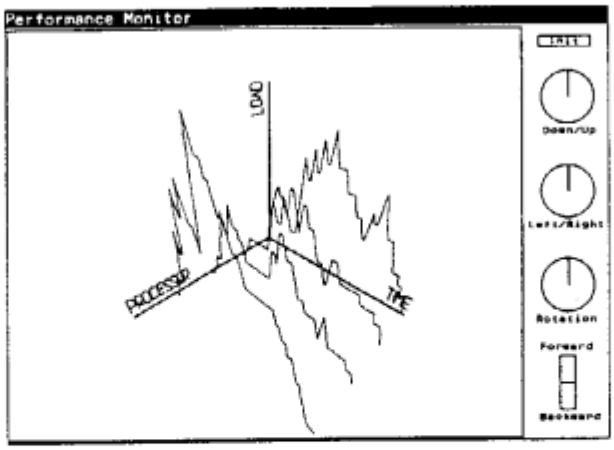

図 1. パフォーマンスモニタの三次元表示

パフォーマンスモニタでは, メニュー操作を行うことにより, 視点または視線を変更して表示することが可能である. 本システムでは, これらの操作をより簡単に行うために, 以下のような自動変更機能も作成した.

Auto Zoom 機能： 対象グラフをウィンドウ内に最大に表示するような視野角を自動選択する.

Auto View 機能： 対象グラフをウィンドウ内に最大に表示するような倍率を自動選択する.

Auto Target 機能： 対象グラフをウィンドウの中心に表示するような視線を自動選択する.

3.2 二次元表示機能

図 1. で示した三次元表示から, 任意のプロセッサまたは, 任意の時間を選択して表示した時間－負荷グラフ及び, プロセッサ－負荷グラフの各二次元表示例を図 2., 図 3. に示す.

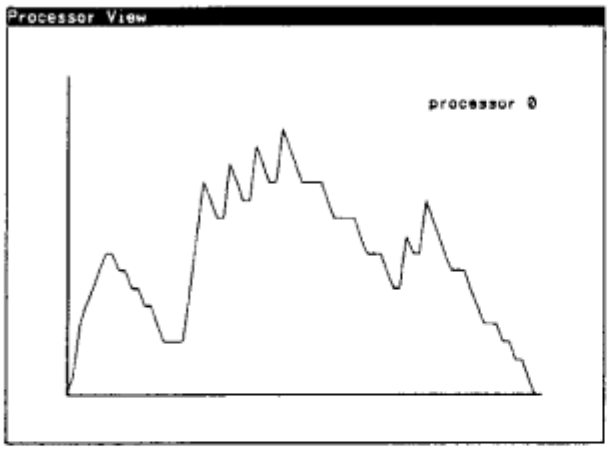

図 2. 時間－負荷グラフ

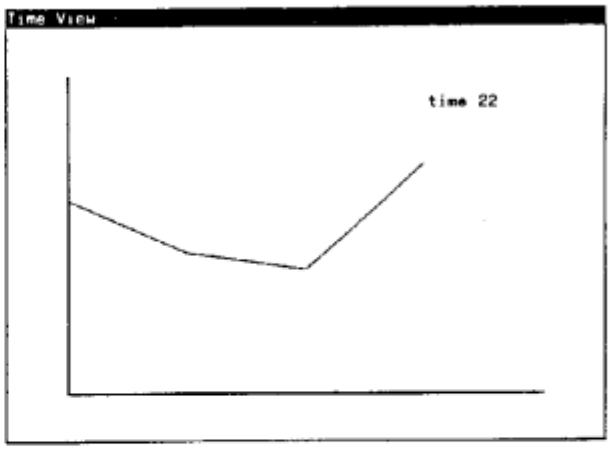

図 3. プロセッサ－負荷グラフ

4 今後の拡張

4.1 並列メタインタプリタの検討

今回作成したメタインタプリタは, あまり並列性を考慮せずに情報収集を行った. 並列性の少ない今回の方式における最大の欠点は, やはり, その実行速度である. しかし, メタインタプリタ

を並列実行させたからといって飛躍的な台数効果が得られるとは限らない. それは, 並列メタインタプリタでは, 実行終了条件 (総てのゴールキューが空である状態) やデッドロック条件 (総てのゴールがサスペンドしている状態) を 1 リダクション毎に同期をとって判定する必要があるからである. 図 4. にその処理イメージを示す.

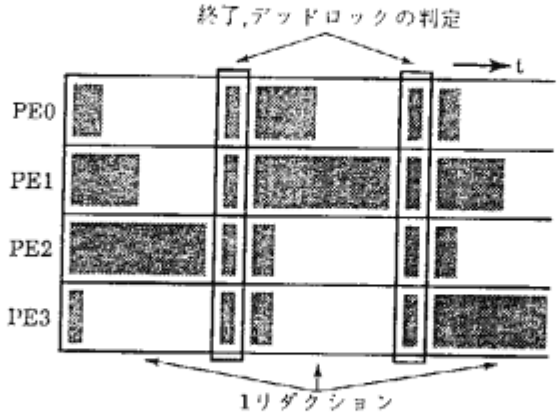

図 4. 情報収集のための同期処理

4.2 パフォーマンスモニタの拡張

パフォーマンスモニタによる負荷状況の推移を用いて, 詳細なプラグマ設定支援を行うためには, トレース情報を参照できる機能が必要である.

この, トレース情報の参照機能は, パフォーマンスモニタで選択したある時点におけるゴールキューの内容, あるいは, その時点の負荷を発生させるきっかけとなったリダクションを示すことを目的とする. そこで, 負荷情報とメタインタプリタから出力させたトレース情報を対応させることにより, プラグマ設定支援を行えるような機能の拡張を考えている.

5 まとめ

KL1 プログラムの設計支援システムにおいて, プラグマ設定支援を目的として開発した, メタインタプリタを利用したパフォーマンスモニタを紹介した. 現時点におけるメタインタプリタの実行速度は, 実用的とは言い難いものであり, この, 実行速度の向上を目指した並列メタインタプリタを現在開発中である. 今後は, さらにトレース情報と負荷情報のリンクおよび, プロセッサ間の通信量を示す機能等の開発を目指す.

参考文献

[Tanaka 88] 田中, 的場 : 変数管理をする *GHC* の自己記述, *ICOT Technical Report:TR-374, 1988*

[Tanaka 87] 田中, 太田, 的場, 神田 : *GHC* による仮想ハードウェアの構築とリフレクト機能について, 日本ソフトウェア科学会第 *4* 回大会論文集, *B-5-3, 1987*

[Sato 84] 佐藤義雄 : 実習グラフィックス, アスキー出版局, *1984*