汎用計算機上のＧＨＣ処理系

森田正雄*，吉光宏*，太細孝**，上田和紀***

（株）三菱総合研究所　　** 三菱電機（株）　　*** ＩＣＯＴ

1．はじめに

言語の普及には処理系の普及が必須である。Guarded Horn Clauses（以下ＧＨＣ［Ueda 86］）は、並列プロセスが記述できる論理型言語である。我々の目的はＧＨＣによる並列プログラミング環境を広く提供することにあり、実用システムを目指している。

実用化に当って、処理性能は極めて重要である。従って、我々は汎用計算機上での高いレベルの最適化技法を模索し、効率のよい逐次処理系の開発を目指している。

以下、現在汎用計算機 VAX-11/780 上に実現しているＧＨＣ処理系の処理方式、実現方法及びその評価について述べる。

なおこのＧＨＣ処理系は、ガード部に制限された述語しか記述できない Flat ＧＨＣをベースとしている。

2．処理方式

ＧＨＣ処理系の機能は、論理型言語的側面の統合化機能と並列言語的側面のプロセス管理機能とに大別することができる。

統合化機能は、Warren の抽象マシンの命令セット［Warren 83］に近い中間コードにより実現している。本ＧＨＣ処理系では、ガード部での一方向の統合化のため中間コードを拡張している。

プロセス管理機能

本ＧＨＣ処理系は、ガード部を逐次に実行し、ボディ・ゴール群は疑似並列に実行する。その際のボディ・ゴールの動作をプロセスという。このプロセスを管理する機能には、次のような処理がある。

① プロセスの生成・実行及び終了（成功又は失敗）処理

② 同期処理（中断及び再開始処理）

これらのプロセス管理の処理はアプリケーション側から見た場合、処理系側のオーバ・ヘッドであり、当然ながらこのオーバ・ヘッドは少ない程よい。そこで、このオーバ・ヘッドを小さくするためにはどうすればよいか？　それはまずプロセス管理の処理自体を最適化すること、そしてプロセス管理機能の実行回数を減らすことである。

①の処理の実行回数を減らす技法は、端末再帰呼び出しの最適化［Warren 83］があり、本処理系もこの技法により、最適化を行なっている。

以下、②の処理の実行回数を減らす方法を吟味する。

（I）　ＧＨＣプログラム実行上の問題点

処理効率を上げるためには、ガード部（候補節群から１つの節を選択する役割を行なう部分）をうまくコンパイルして、最適な決定木に展開する必要がある。しかし、中断する場合（呼び出し側の引数が具体化されるのを待たなければならない場合）は他の手続き型言語のようにうまく最適化することができない。例えば、図１の mergeの手続きを実行する際、第一引数が具体化されていなくて中断した場合、同期のための情報（呼び出し側の引数が具体化された場合に mergeの手続きを駆動させるためのきっかけを与える情報）を動的に生成した上で、第一引数が具体化されていなくとも実行できる第２節及び第４節の処理に移らなければならない。（このような外部から手続きを駆動するきっかけを２つ以上持つしくみを多重 wait と呼ぶ。）

このような処理を行なうため、１つの手続きの処理の流れは１本の線ではなく、また１つの手続きに対して、複数の中断情報を持たなければならなく、プロセス管理のオーバ・ヘッドを大きくする。

```
merge( [], Y, Z):-true
merge( X, [], Z):-true
merge([A|X],Y,Z):-true
merge(X,[A|Y],Z):-true
```

図1 merge のｶﾞｰﾄﾞ 部

```
split([], X, L, R):-true
split([H|T],X,L,R):-X>H |
split([H|T],X,L,R):-X=<H|
```

図2 split のｶﾞｰﾄﾞ 部

しかし、図２の splitの手続きでは、全ての節のガード部の実行を行う際に第一引数が具体化されている必要があり、また第一引数がリストの場合第２節と第３節が候補となり、それらの節は第一引数のリストのＣＡＲと第２引数が具体化されて後、節を決定する。従って、このような手続きはガード部を最適化することにより、１つの変数で中断が起こった場合に、他の節の処理に移らなくともよい手続きであり、多重wait を必要としない手続きである。

ここで重要なことは、「実際のＧＨＣで書かれた多くのアプリケーション・プログラムが多重 wait を必要としない手続きであり、そのような手続きはガード部を決定木に展開し、決定的に処理することができる」ということである。

（II）　ガード部の実行方式

現状の汎用計算機ではプログラムを決定木に展開し、決定的に処理することがもっともシンプルであり、かつ効率的である。また、コンパイラに対して最適化の可能性を残すことは重要であり、隠れた所（上位から見て最適化不可能部分）で大きなオーバ・ヘッドを持つようなアーキテクチャは避けたい。このような観点から、ベースのアーキテクチャは１つの手続きを決定的に処理するようにした。すなわち、１つの手続き中で多重waitしない。具体的には節のガード部を実行する際、失敗した場合のみ他の節の実行に移り、中断した場合は手続きごと待つようにした。また、中断から

GHC system on a General-Purpose Computer.

M. MORITA *, H. YOSHIMITSU *, T. DASAI **, K. UEDA***

*MRI, **MELCO, ***ICOT

再実行される場合は、中断した所から再開するようにした。

また、多重 wait を必要とする手続きは、他のＧＨＣ処理系と同様な処理となる。但し、多重waitを必要とする代表的な問題（複数ストリームの併合）を効率のよい組込み述語で提供するようにした。

(III)　多重 wait checking

上述のアーキテクチャはコンパイル時の静的な解析による（手続きごとの）多重 wait checkingを前提としている。そのアルゴリズムを以　に示す。

まず前処理として、ガード部に中断する可能性を持つゴールがある場合、そのゴールを節の頭部に反映させる（たとえば統合化であれば、中断の原因となる変数を、統合化する項に置き換え、また算術比較の引数として現れる変数は、適当な定数で置き換える）。こうして得られた節頭部の集合が下記①または②を満たすことが、多重 wait を要しない十分条件である。

① 変数が全く現れないか、または全ての節頭部において変数の出現位置（occurrences)が同一である。

② 下記のように、節頭部の分類を行なったとき、どのクラス（節頭部の集合）も多重 wait を要しない。

(イ)　次のような条件を満たす出現位置を求める。
- 各節頭部の、その出現位置の項は全て非変数。
- それらの項の上位の各レベルの項の主関数記号は、全ての節頭部で同一。

(ロ)　(イ)で求めた出現位置の項の主関数記号によって、節頭部の集合を分類する。

このような技法は、不必要な実行時の同期を回避するためのコンパイル技法といえるが、同時にガード部の検査を決定木に展開する方法の基本的枠組も与えている。

3.　実現方法

ＧＨＣ処理系は VAX-11/780 上にインプリメントした。

ユーザのＧＨＣプログラムは、コンパイル時に中間コード列に変換され、中間コード列は更に、効率上重要な部分はアセンブル・コードにインライン展開され、それ以外はアセンブラで書かれたラン・タイム・ライブラリを呼び出すアセンブル・コードに変換される。プロセス管理及びＩ／Ｏ以外の組込み述語はアセンブラで記述されており、Ｉ／Ｏ関係の組込み述語及び初期化モジュールはＣ言語及び yacc 、lex により記述されている。またコンパイラはＧＨＣで記述されている。

なお図３にプロセス中心に見たプロセス管理の事象処理を図示した。

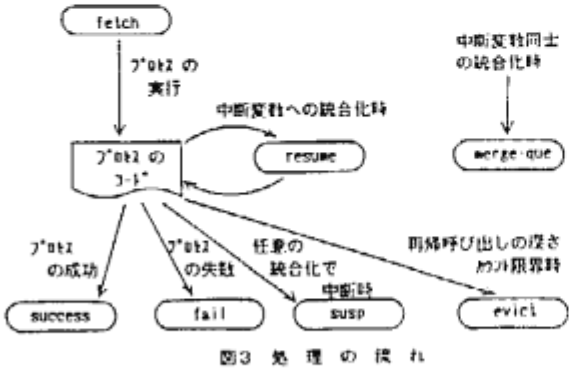

図3　処　理　の　流　れ

4.　評価

いくつかの手続きを、多重 wait を必要としない手続きとしてコンパイルし、性能測定した所、表１のような結果となった。

表１　処理性能

プログラム	append	quick sort	prime
性能（rps*）	27.6 K	15.1 K	11.2 K

*) number of Reductions Per Second

プロセス管理の同期処理（プログラム実行中に中断が起こる際の中断処理＋再開始処理）自体の最適化に努めた所、その処理にかかるＣＰＵ時間は約４０μsec であった。これは　append　の１回のreduction 時間に相当する時間であった。

しかし、プログラムの大きさとして実践的に近いと考えられるＧＨＣコンパイラ（1400行程度）を実行する際、同期処理にかかるオーバ・ヘッドを計測した所、少なくとも先の計測値の３倍以上のオーバ・ヘッドがあった。仮想記憶方式、キャッシュ方式を採用している計算機全てに言えることだが、同期処理によるプロセス切換えの多用が、処理のランダムなアクセスに繋がり、それがむしろ単に同期処理実行のみのオーバ・ヘッドをはるかに超えるオーバ・ヘッドであることがわかった。

このような現状の汎用計算機の動特性が、（多重 wait checking　により）不必要な同期処理を回避する技法の有用性をさらに重要なものとした。

5.　おわりに

ＧＨＣの本来のプログラミング・スタイルは好んで同期処理を利用する。同期処理の最適化は重要である。

ここに提示した多重 wait checkingは、論理型言語において clause indexingが重要な技法であるのと同様、並列論理型言語において重要な技法の１つであると考えている。また、この技法は、マルチ・プロセッサ環境でも有用な技法である。

今後、本ＧＨＣ処理系にＧＣ及びデバッガ機能の拡張を行う予定である。またもし可能ならば他計算機への移植も行ないたいと考えている。

なお本研究は第５世代計算機プロジェクトの一環としてＩＣＯＴの委託で行なったものである。

＜参考文献＞

[Ueda 86]　Kazunori, Ueda:
" Guarded Horn Clauses "
LNCS 221, Springer

[Warren 83] Warren, D.H.D.:
" AN ABSTRUCT PRULOG INSTRUCTION SET "
SRI International, Technical Note 309