GHCのメッセージ指向の処理方式

3G－2

森田正雄
(株)三菱総合研究所

上田和紀
(財)新世代コンピュータ技術開発機構

1．はじめに

並行プロセスの機能には、計算と記憶の両面がある。並列言語の処理系を考える場合、従来はプロセスの計算の側面に照準を当てて最適化が行なわれてきた。我々も、汎用計算機上にGHC[Ueda 86]の処理系(GHC／V処理系)[森田 87]を開発し、いくつかの最適化技法を提案してきたが、そのような最適化では、ほとんどのプロセスが中断(suspension)状態であるようなプログラム(例えば図1のプログラム)を効率よく処理できない。本稿では、プロセスの記憶の側面に着目したGHCプログラムの最適化技法を提案する。

```
nt_node([],              _,  _,L,R) :- true |
    L=[], R=[].
nt_node([search(K,V)|Cs],K, V1,L,R) :- true |
    V=V1, nt_node(Cs,K,V1,L,R).
nt_node([search(K,V)|Cs],K1,V1,L,R) :- K<K1 |
    L=[search(K,V)|L1], nt_node(Cs,K,V1,L1,R).
nt_node([search(K,V)|Cs],K1,V1,L,R) :- K>K1 |
    R=[search(K,V)|R1], nt_node(Cs,K,V1,L,R1).
nt_node([update(K,V)|Cs],K, V1,L,R) :- true |
    nt_node(Cs,K,V,L,R).
nt_node([update(K,V)|Cs],K1,V1,L,R) :- K<K1 |
    L=[update(K,V)|L1], nt_node(Cs,K,V1,L1,R).
nt_node([update(K,V)|Cs],K1,V1,L,R) :- K>K1 |
    R=[update(K,V)|R1], nt_node(Cs,K,V1,L,R1).

t_node([]             ) :- true | true.
t_node([search(_,V)|Cs]) :- true |
   V=undefined, t_node(Cs).
t_node([update(K,V)|Cs]) :- true |
   nt_node(Cs,K,V,L,R), t_node(L), t_node(R).
```

図1　2進木データベース

2．　プロセス指向とメッセージ指向スケジューリング

1台のプロセッサが複数のプロセスを実行する擬似並列処理では、一つのプロセスの処理が始まったら、できるだけの多くの仕事をしてから別のプロセスの処理に移る方法と、複数の仕事ができる状態でも一つだけ仕事をやり、すぐにその結果により動作できるようになったプロセスの処理に移る方法とがある。前者は、GHC／Vをはじめ従来の処理系がとっている方式で、プロセス指向スケジューリングと呼ぶ。一方、後者は新しい処理方式であり、これをメッセージ指向スケジューリングと呼ぶ。

プロセス指向スケジューリングの目的は、できるだけプロセス切替えの回数を減らすことである。これは、プロセス間をつなぐストリームが処理結果をバッファリングすることによって実現される。プロセス指向スケジューリングはスループット指向のスケジューリングとも言える。

一方メッセージ指向スケジューリングでは、プロセスが他のプロセスへメッセージを送信するたびに送信先に制御を渡し、直ちにそのメッセージの処理に移る。(ここでは図1のプログラムのような1対1の通信を想定する。)これを可能にするために、モード解析情報を利用して、メッセージの受信プロセスの実行が送信プロセスの実行に先行するようにする。メッセージ指向スケジューリングはレスポンス指向のスケジューリングとも言える。具体的に、入力ストリームのメッセージを単に出力ストリームにコピーするプロセスで考えてみよう。

```
p([A|X1],Y) :- true | Y=[A|Y1], p(X1,Y1).
```

2つのボディゴールのうち、プロセス指向スケジューリングでは、Y=[A|Y1]を先に実行することにより、Aをストリーム Y の先頭にバッファリングする。そして p(X1,Y1) を終端再帰呼出しの最適化により効率的に処理する。それに対してメッセージ指向のスケジューリングでは、先に p(X1,Y1)を実行することにより入力メッセージの待合せ状態を回復したあと、Y=[A|Y1]をメッセージ送信の形で実行する。

メッセージ指向スケジューリングで、プログラムを効率よく処理するためには、プロセス・キューやデータ・バッファを用いずに制御の切り替えやメッセージの授受を効率よく処理することが重要である。そこで、ストリームをリストではなく、通信セルという特別なセルにより実現した。このセルは、受信プロセスの①環境(引数値等を記録したプロセス・レコード)へのポインタ及び②処理再開始点(機械コードのアドレス)の情報を持つ。またメッセージはメモリでなく、ハードウェア・レジスタ(通信レジスタ)により授受する。

3．　メッセージ指向スケジューリングの一般化

一般のプログラムにメッセージ指向方式を適用するためには、次のような問題がある。

① ストリーム通信用の変数と通常の変数の判別をどうするのか?

② 1対1の通信でない場合、どのように処理するのか?

①については、モード解析[Ueda 89]と同様の技法を用いた型推論を行えばよい。次に②は、(a)受信プロセスがないか、または複数の受信プロセスがある場合と、(b)一つのプロセスが複数のストリームを受信している場合とに分けて考える。

(a)のケースでは、1対1の通信に変換すればよい。たとえば、メッセージを受け取る受信プロセスがなくなるのであれば、残りのメッセージを最後まで読み捨てるダミープロセスを生成する。また、複数の受信プロセスがある場合には、メッセージを分配するプロセスを生成する。

Message Oriented Implementation of GHC

1. Masao MORITA　　2. Kazunori UEDA
1. Mitsubishi Research Institute 2. ICOT Research Center

(b)のケースはさらに二つのケースに分かれる。非選択的受信と選択的受信である。

非選択的受信とは、ストリームの非決定的併合器のようなプロセスで行われるメッセージ受信であるが、これは通常の１対１の通信と同様に処理できる。つまり、各入力ストリームに対応する通信セルに、到着メッセージを処理する異なったコードへのポインタを持たせ、一方の入力ストリームからのメッセージを他方のストリームとは独立に処理する。

選択的受信とは、たとえば次のようなストリームの順序付き併合器で行われる受信のことである。

```
merge([A|X1],[B|Y1],Z) :- A< B |
    Z=[A|Z1], merge(X1,[B|Y1],Z1).
merge([A|X1],[B|Y1],Z) :- A>=B |
    Z=[B|Z1], merge([A|X1],Y1,Z1).
```

このプロセスがリダクションするためには双方の入力ストリームからの２つの数値が必要である。一方の入力ストリームから数値が到着した場合、他方の入力ストリームの数値の到着を待たなければならない。しかし、最初に数値が到着したストリームから次の数値が到着することもありうる。このような場合、プロセスは後の処理のために二つめの数値をバッファリングしなければならない。選択的受信のもう一つの例として append プログラムがある。

```
append([],    Y,Z) :- true | Z=Y.
append([A|X1],Y,Z) :- true |
   Z=[A|Z1], append(X1,Y,Z1).
```

このプロセスは、第一引数のストリームが閉じるまで、第二引数のストリームから受信したメッセージをバッファリングしなければならない。いずれのケースも、プロセスは受信メッセージを直ちに処理できるとは限らず、その場合バッファリングをしなければならない。

一般に、バッファリングが必要となるのは、受信メッセージが直ちに処理できる状態にないストリームに、メッセージが送信される可能性のある場合である。その一例が、上に示した選択的通信の場合である。もう一つの例は環状プロセス構造のある場合である。次のようなプログラムで考えてみよう。

```
p([a|X1],Y,Z) :- true |
   Y=[b|Y1], Z=[c|Z1], p(X1,Y1,Z1).
```

このプログラムは第一引数から、ａというメッセージをもらったら、第二引数にｂ、第三引数にｃというメッセージを送る。このとき第三引数へのメッセージ送信の前に、第二引数への送信の結果として第一引数にメッセージが到着することがありうる。このような場合、到着したメッセージをバッファリングしなければならない。

プロセスがメッセージを１個受信する度に高々１つのメッセージだけを送信するのであれば、このようなバッファリングは必要としない。一般にはプロセスがメッセージを１個受信した結果、ｎ個のメッセージを送信する場合、最初のｎ－１個のメッセージ送信により、新たなメッセージが自分自身に対して送られてこなければ、最後のメッセージ送信は、バッファリングの用意をせずに効率よく処理できる。これを最終送信の最適化と呼ぶ。

4，　評価

我々は、メッセージ指向処理系の中間コードを設計[森田89]し、ハンドコンパイルにより、ＧＨＣ／Ｖ処理系との性能比較を行った。対象のプログラムは図１で、まずＧＨＣ／Ｖ処理系で８００個の search メッセージを一挙に与え、処理時間等を測定した。次に search メッセージを１件流し、応答が返ってきたら、次のメッセージを流すと言うように間欠的にメッセージを与え、それに要した時間等を計測した。次にメッセージ指向処理系により、同様に処理時間等を測定した（この場合、メッセージを連続的に与えても、１件毎に与えても、同じ結果となる）。

表１　２進木データベースの測定結果（VAX11/780上）

測定対象（検索データ８００個）	処理時間（秒）	中断回数
ＧＨＣ／Ｖ　ＤＢの連続検索	１．０４	０
ＤＢの間欠的検索	２．０９	９８１７
メッセージ指向　ＤＢの検索	０．７５	０

時間効率については、表１からわかるように、メッセージ指向処理系の効率は、search メッセージの与え方とは無関係に、ＧＨＣ／Ｖ処理系上での連続検索の効率を上回った。また空間効率については、メッセージ指向の処理系は、メッセージの送信時にストリーム構造を作らないので、検索時にメモリが消費されないという利点がある。さらにメッセージ指向の処理方式の通信形態は１対１を基本としており、通信の終了時に通信のための領域（通信セル及びバッファ）を解放することができ、即時回収が不可能なゴミは一切発生しない。

5，　まとめ

ここに示した技法の意義は、ＧＨＣ／Ｖをはじめとする従来の処理方式が不得意としていた散発的なメッセージ通信を効率よく処理する枠組みを示したことにある。この処理方式がうまく実用化できれば、ＧＨＣによるオブジェクト指向計算、データベース、要求駆動計算などの記述の実用性が高まると期待できる。また、並列処理系でもこの技法は適用できると考えるが、多くの問題が残されている。

《参考文献》

[Ueda 86]　Ueda,K.:　“ Guarded Horn Clauses ”, in Logic Programming '85, LNCS 221,Springer-Verlag, 1986, pp.168-179.

[森田 87]　森田、吉光、太細、上田：　汎用計算機上のＧＨＣコンパイラ,第３５回情処全国大会 5Q-4, 1987.

[森田 89]　森田、上田：　ＧＨＣプログラムの最適化，ロジックプログラミング・コンファレンス’８９，ＩＣＯＴ，pp.203-214

[Ueda 89]　Ueda,K. and Morita,M.:　“ Optimization of GHC Programs ”，日本ソフトウェア科学会第６回大会論文集，1989, pp.41-44