GHC のメッセージ指向の並列処理 — 初期評価 —

5M−10

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

我々は，図1のような，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を効率よく処理するメッセージ指向の並列処理方式を検討している[1]．本稿では，共有メモリの並列計算機上で，メッセージ指向の処理方式の初期評価を行なう．

2. メッセージ指向のメッセージ送信

メッセージ指向処理では，送ったメッセージを受信側が直ちに処理できるようにゴールをスケジューリングする．メッセージは通信レジスタと呼ばれるハードウェアレジスタに乗せて受け渡す．ストリームはリストではなく，制御の移動に必要な情報を記録した通信セルで表現する．そこで，通常は図2のように，メッセージ送信と同時に受信プロセスに制御を移すことができ，メッセージをバッファリングする必要がない．しかし，プロセス間通信の通信形態などにより，受信プロセスがすぐにメッセージを受信できない場合もある．そのときは通信セルが図3のように変化して，バッファリングによる処理に切り替わる．

3. メッセージ指向の並列処理方式

我々の当面の目標は，共有メモリ型汎用並列計算機上のメッセージ指向の並列処理方式を確立，評価することであり，[1]で述べたゴール共有方式を詳細化している段階である．しかし，メッセージ指向方式の並列処理の研究は端緒についたばかりであり，今回は第一歩として，単純なスケジューリングでどの程度の効率が得られるのかに着目し，評価することとした．

処理系の開発は，Sequent 社の Symmetry (80386 プロセッサを用いた共有メモリ型並列計算機) 上にまず逐次処理系を作成し，その処理系を並列処理できるように改良するという手順で行なった．記述言語は，処理効率上重要な実行時サブルーチンや中間コードなどはアセンブラを，その他の部分は C 言語を用いている．

現在の処理系の並列実行方式は，図4のような単純な構成となっている．各プロセサは競争的に1本のシステム・スタックから仕事を獲得し，それを処理する．プロセサが複数個動作する場合には，1台の親プロセサと残りの子プロセサに分かれる．システム・スタックにプロセサ数に見合う仕事がある間は，親・子の区別なく動作

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

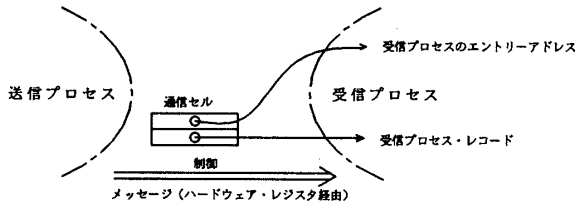

図2　メッセージ送信（ノーマル）

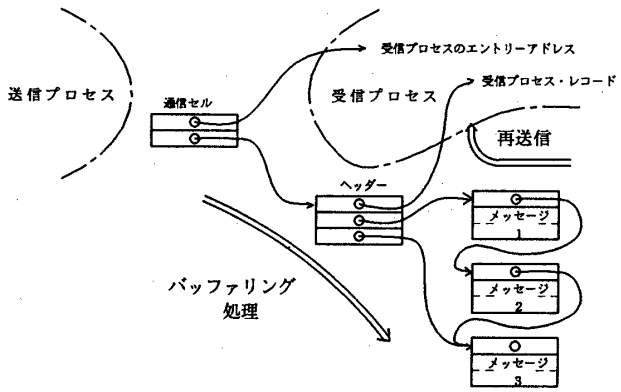

図3　メッセージ送信（バッファリング）

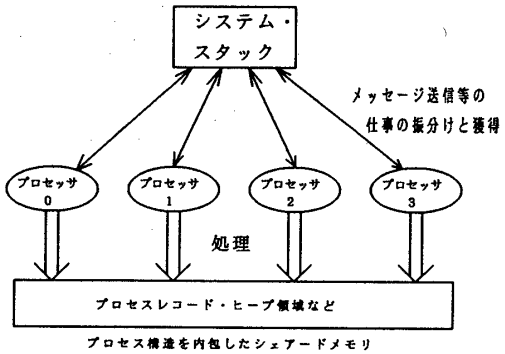

図4　並列処理

Messege-Oriented Parallel Implementation of GHC Programs: Preliminary Evaluation

Masao Morita[1], Kazunori Ueda[2]
[1]Mitsubishi Research Institute
[2]Institute for New Generation Computer Technology

するが，一度親プロセサがシステム・スタックの仕事を獲得できなくなると，親プロセサは監視プロセサに変化して，終了判定や異常事態の検出などを行なう．

ゴール共有型のメッセージ指向並列処理の施錠には，プロセス指向処理で行なう変数アクセスやシステム・スタックの施錠のほか，プロセスに対する施錠及びバッファリング・データに対する施錠がある．これらの施錠は，空間的にも時間的にも局所化するように努めている．なお，解錠の動的待合せには，Symmetry の xchg 命令を用い，プロセサ間の排他制御を実現している．

4. 評価

我々は図1のプログラムをハンドコンパイルし，3000個の search コマンドの実行に要する時間を Symmetry 上で測定した．計測時のプログラムの状態図を図5に示す．測定は，(1) 逐次処理系，(2) プロセサ台数をパラメタ化した並列処理系，および (3) 2進木データベースをレコードとポインタで実現したCプログラム (逐次)，の三つについて行なった．その結果を図6に示す．

並列処理系のプロセサ1台の場合の性能が逐次処理系と比較して落ちているのは，並列処理のための施錠，解錠などのオーバーヘッドが加わっているためである．しかし，そのオーバーヘッドはそれほど大きくはない．

グラフから，プロセサ6台程度までは着実に性能向上が見られることがわかる．だが，プロセサを6台以上に増やしても，ほとんど時間短縮が見られない．これは，図5の2進木データベースにメッセージを流すドライバが逐次処理になっており，この部分がボトルネックになっていることが一因であると考えられる．また，プロセサ台数を増加させると，ゴールへのアクセス競合によるオーバーヘッドも増加する．なお，台数効果のグラフは直線的に見えるが，上のような理由で，台数に比例する効率向上にはなっていないことに注意してほしい．

この評価では，プロセサ5台で，最適化したCプログラムの時間効率を上回ることができた．しかし，空間効率は，Cプログラムと比べて数倍劣っている．

最後に，1000要素のリストの反転処理 (naïve reverse) の測定結果を図7に示す．このプログラムも，2進木データベースとほぼ同様の特性となっている．

5. まとめ

メッセージ指向の処理方式は，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逐次処理系上で効率よく処理するが，この技法を拡張して並列処理をすればさらに十分な効率向上ができるという見通しが得られた．今後も，方式の一般化，詳細化を図ってゆきたいと考えている．

参考文献

[1] 上田，森田：GHC のメッセージ指向の並列処理 — 概要 —．本予稿集，1991.

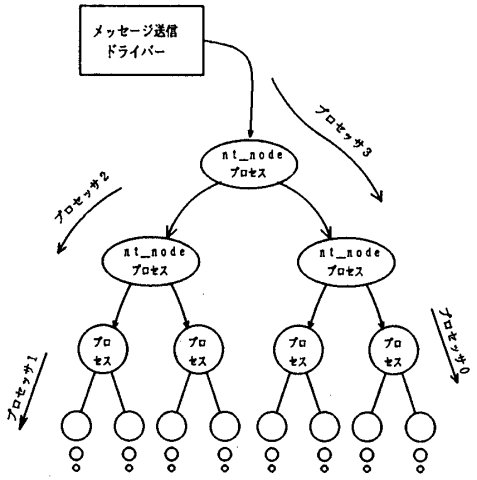

図5　2進木データベース・プロセス状態図

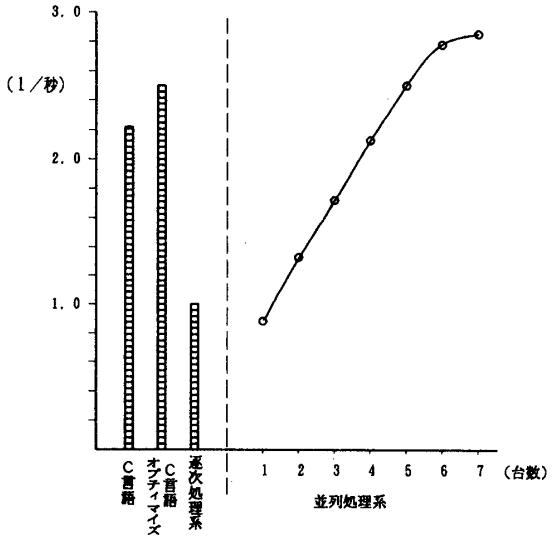

図6　2進木データベース・測定結果

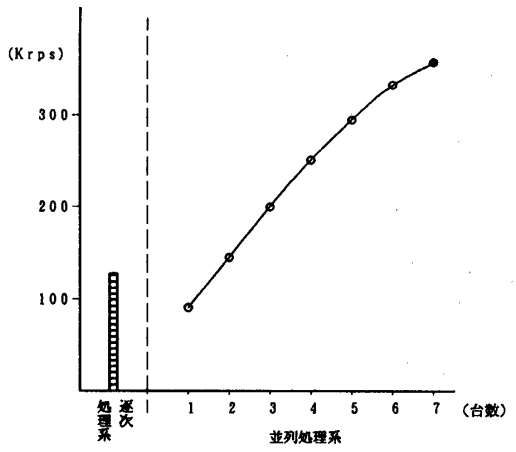

図7　n-reverse（１０００要素）・測定結果