2H-02

ＰＣクラスタにおける共有メモリの実装実験(2)

--ロックの実装--

成蹊大学大学院工学研究科情報処理専攻

伊藤 慎、緑川 博子、飯塚 肇

【背景と目的】

従来、メッセージ通信によってプログラムされていたクラスタ型並列コンピュータシステムにおいて、共有メモリプログラミング環境を実現する研究が各所で行われている。本研究室においても、同期関数としてバリアを持つシステム(sms.0.1)を構築し、その概要については既に報告した。[1]

今回さらにロック／アンロック関数を加え、新しい緩和型メモリコンシステンシを導入して、より汎用性を高めた実装(sms.0.2.1)を試みたので報告する。

【sms.0.2.1の概要】

sms.0.2.1は以下のような特徴を持っている。

- ユーザレベルソフトウェアによる実装
- 仮想共有メモリに対するページベースでの管理
- 同期関数としてバリア、ロック関数の実装
- 仮想共有メモリの一貫性を保つための更新データとしてページのコピーとの差分を採用

【sms0.2.1に於るメモリコンシステンシモデル】

できるかぎりネットワーク通信量を減らして性能を向上させるため、アンロック時に全プロセッサのメモリ内容を一致させる従来のリリースコンシステンシモデルに代えて、レージーリリースコンシステンシモデル[2]を用いた。すなわち更新されたデータ(正確にはそのデータを含むページ)を次にロックを使用してアクセスしたプロセッサのみに受け渡す。このモデルはTreadMarks[2]でも用いられているが、invalid方式を用いておりinvalid情報と更新データ情報は2段階で他のプロセッサへ送られる。ｓｍｓでは、ロック時に更新データ情報を一度に送付するupdate方式を採用している。

更に細かく述べると図1のように、次に同じIDのロックを獲得したプロセッサ(P1)のみに、前回の同じIDのロックを獲得したプロセッサ(P0)から更新データが渡される。したがって、(3)、(4)のデータアクセスでは(2)、(3)の更新情報がそれぞれ反映されているが、ロックを使用しなかった(5),(7)のアクセスではバリア(1)時のデータ内容のままである。またアクセス(6)でも(4)の更新情報は反映されない。

なおロックにはIDがあるので、それぞれのデータオブジェクトに関連づけることができる。また、バリアでは全システムのメモリのコンシステンシがとられるが、ロックとアンロックの間でバリアを呼ぶことや、ロックの入れ子は許されていない。

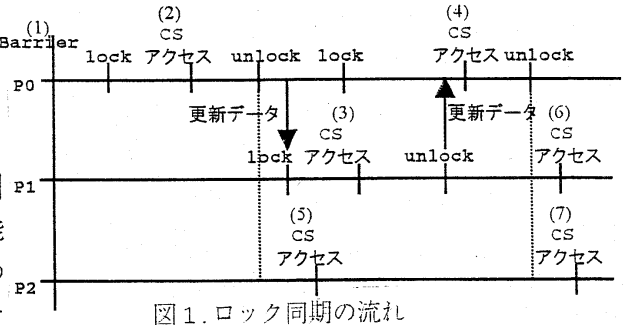

図1.ロック同期の流れ

【ユーザインターフェース】

本システムでは表1に示すsmsライブラリを提供している。これを用いて書いたサンプルプログラムを図2に示す。このプログラムは配列の中から最大値を求めるもので、共有にある最大値の変数にアクセスするためにロック同期を使用している。

【システムの内部構造】

今回の実装ではバリアマネジャ、ページマネジャに加えロックマネジャを新設し、ロック/アンロック間の更新差分diffLをバリア間の更新差分diffBとは別に設けた。diffLはタイムスタンプが付加されており、バリア間で複数のロック同期を行った際に、前回ロック同期時のタイムスタンプ以降の

Implementation of Shared Memory Model on PC Cluster2

Shin Ito, Hiroko Midorikawa, Hajime Iizuka,

Department of Information Sciences, Seikei University

```
sms_nproc    ：並列動いているプロセス数
sms_proc_id  ：プロセス ID(0,1,2,...)
void sms_startup(int argc,char *argv)
  システムの初期化を行う関数
void sms_shutdown()
  システムの終了を行う関数
void *sms_alloc(int size,int PageM_id)
  共有メモリを確保する関数
void sms_barrier(int BarrierM_id)
  バリア同期関数
void sms_lock(int LockM_id)
  ロック獲得関数
void sms_release(int LockM_id)
  ロック解放関数
```

表1　定数及び関数

```
#include<Sms.h>
#define NUM 1200
void main(int argc, char **argv)
{
  int *data,*max,n=NUM/sms_nproc,i;
  sms_startup(argc, argv);
  data =(int *)sms_alloc((sizeof(int))*NUM,0);
  max  =(int *)sms_alloc(sizeof(int),1);
       for(i=sms_proc_id*n;i<(sms_proc_id+1)*n;i++)
              spage[i]=rand();
       if(sms_proc_id==0)
              *max=0;
       sms_barrier(0);
       for(i=sms_proc_id;i<NUM;i+=sms_nproc){
              sms_lock(0);
              if(*max < spage[i])
                     *max = spage[i];
              sms_release(0);
       }
       sms_barrier(0);
       printf("¥nmax value = %d¥n¥n",*max);
       sms_shutdown();
}
```

図2　プログラム例

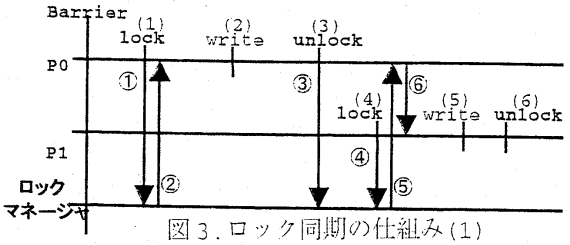

図3．ロック同期の仕組み(1)

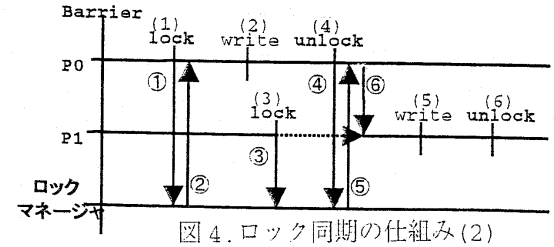

図4．ロック同期の仕組み(2)

diffLを適用する。

【ロック同期の仕組】

基本動作を図3示す。次に各部の動きを説明する。

(1)ロック関数がコールされるとロックマネージャ(LM)にロック要求メッセージ①を送る。LMはバリア後の初めてのロックであることを判断し、diffLが無いことを知らせるメッセージ②を送る。ロックを要求したプロセッサ(P0)はロックを獲得したことを知る。

(2)ロック同期中の初めての書き込みアクセス時にdiffL作成のためにページのコピー(twinL)を作成する。

(3)ロック解放時にLMに解放メッセージ③を送りdiffLを作成する。

(4)次にロックを獲得しようとするプロセス(P1)は、LMにロック獲得要求メッセージ④を送る。LMはP1がロックを獲得することが出来ると判断し、P0に対して、更新データ(diffLのリスト)をP1に送るように依頼するメッセージ⑤を送る。P0はこのメッセージを受け取るとP1に対してdiffLリストのメッセージ⑥を送る。

また図4の様に、ロック同期中に別のプロセスのロック要求③が行われた場合には、ロック要求はロック獲得待ちのキューへ入れられ、コールしたプロセスはブロックされる。P0のロックがリリースされると、キューの先頭にあるプロセスがロックを獲得する。以下は図3と同様である。

【おわりに】

以上で本システムに於けるロックの実装を述べた。現在、システムの動作の安定化、及び性能評価を行っている。

【参考文献】

[1]斎木、緑川、飯塚:"PCクラスタにおける分散メモリ環境の構築",情処学会58回大会3H-02

[2]P.Keleher,et.al:"TreadMarks:Shared memory on Networks of Workstations" IEEE computer, Vol.29(2),PP18-28,1996-02