PCクラスタにおける共有メモリの実装実験

3F－2

斉木 雅弘、緑川 博子、飯塚 肇
成蹊大学大学院工学研究科情報処理専攻

【はじめに】

近年，低コスト・高性能ＰＣと高速ＬＡＮの普及により、複数のPCをネットワーク接続した計算機クラスタが並列処理に広く用いられるようになってきた。従来、このような分散メモリ型並列マシンでは、メッセージパッシングによるプログラミングが一般的であったが、最近では、高速ネットワークや低遅延通信プロトコルの出現により、共有メモリプログラミング環境を実現する研究も盛んに行われている。[1]

筆者らは、PC クラスタ上にユーザレベルソフトウエアによる分散共有メモリシステムの実装を試みたので、その設計、実装結果について報告する。

【システムの特徴】

本研究で、設計した分散共有メモリシステムは以下のような特徴をもつ。

・ユーザレベルソフトウエアによる実装

通信ハードウエアに依存しないソケット通信を用い、OS に手を加える必要のないユーザレベルソフトウエアによる実装とした。これにより、多くのUNIX システムに容易に移植が可能となる。

・緩和型メモリコンシステンシーの採用

各PCに分散したメモリの一貫性をとる方式は、ネットワーク通信遅延による性能低下を防ぐために、Week メモリコンシステンシモデルを用いた。

・ページベース管理

一貫性維持の粒度はページ単位で、応用プログラムによる共有メモリデータのアクセス検知には、OS で提供されるページ保護機構を利用する。

・差分(diff)データ転送

共有メモリデータの一貫性を保つための更新データは、前回のバリア同期時のページコピーとの差分(diff)を取り、これを該当ページコピーを所有するプロセッサにのみ更新データとして送り、通信量の削減を図った。

【システムの内部構造と動き】

・ページマネジャ

共有データ（ページ）を管理するプロセッサ（プロセス）をユーザがデータアロケート時に指定できる。各プロセスが最初にそのページにアクセスにすると、そのページの管理マネジャからページコピーとそのページコピーをもつプロセッサマスクが送られる。

・バリアマネジャ

ページマネジャと別にバリア同期処理を行うプロセッサ（プロセス）をバリアマネジャとしてユーザが指定でき、ページ管理と同期管理の負荷を分散できる。

・sigsegv ハンドラ

ページ保護機構により共有メモリアクセス時に呼び出され、要求されたページのコピーを持っていなければページマネージャにページ要求し、バリア同期後の最初の書き込みならdiff生成のためにそのページのコピー(Twin)を作る。

・メッセージハンドラ

ページ要求、diff 転送など、プロセッサ間通信時に呼び出される通信ハンドラである。

・共有メモリライブラリ

図１に示すＣの関数が用意されており、応用プログラムから使用できる。sms_startup 関数はリモートプロセスの起動、通信路の確保、共有メモリページの状態初期化、シグナルハンドラの登録など各

Implementation of Shared Memory Model on PC cluster
Masahiro SAIKI, Hiroko MIDORIKAWA, Hajime IIZUKA,
Department of Information Sciences, Seikei University

種初期化を行う。ページはアップデートプロトコルで更新される。sms_alloc 関数はその内部に同期機構を持ち、指定されたページマネージャが実際の割付を行い、これを他のプロセスに知らせる。sms_barrier 関数は diff の生成と送信、他のプロセスから受信した diff の適用を行い、最後にページ属性などを初期化した後、全てのプロセスがこの関数に到着するまで待つ。 図２にプログラム例を示す。

定数
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,...)
関数
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 id)
　共有メモリを確保する関数
void sms_barrier(int id)　バリア同期関数

図１　共有メモリ関数

```
#include <stdio.h>
#include "Sms_usr.h"   /* システム用ヘッダ */
#define MAX 500
#define START  sms_proc_id*MAX/sms_nproc
#define END  (sms_proc_id+1)*MAX/sms_nproc

void main(int argc,char *argv[])
{
  int i,*array;
  sms_startup(argc,argv);   /*システム開始*/
  /* 共有メモリ確保 */
  array=(int *)sms_alloc(sizeof(int)*MAX,0);
  for( I=START; i<END; i++)   /* ジョブ分割 */
    array[ i ]= i;   /* 共有メモリへの書き込み */
  sms_barrier(0);   /* バリア同期 */
  for(i=0; i<MAX && sms_proc_id==0; i++)
    printf("%d ",array[ i ]);   /* 結果出力 */
  sms_shutdown();      /* システム終了 */
}
```

図２　プログラム例

【評価】

本実験は図３の環境で評価を行った。図４に基本部分のオーバヘッド時間を示す。アロケートは一定で約１ms、diff 転送のない場合のバリアも変動はほとんどなく数 ms である。この例では起動後の初回のページ配布に比べ、遠隔プロセス起動やソケット確立を行う startup に時間がかかる。

共有２次元データに対する近傍繰り返し処理プログラムを、計算負荷を変えて行った場合の PC８台の性能向上比を図５に示す。この例では計算負荷パラメタｋが１０以下になると極端に性能が悪くなる。

ＰＣ	CPU:MMXPentium166MHz メモリ:64MB OS:FreeBSD2.2.2R
ネットワーク	100Mbps イーサネット

図３　システム環境

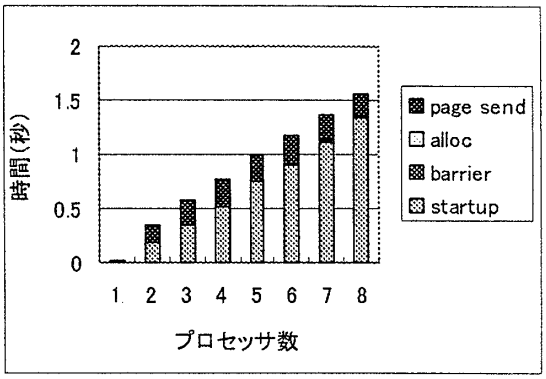

図４　システム開始時のオーバーヘッド

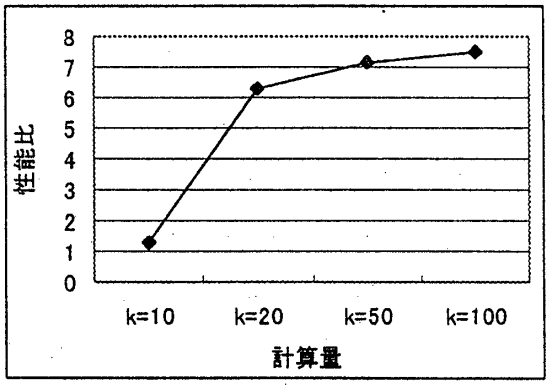

図５　サンプルプログラムの性能向上比

【おわりに】

以上で本システムの設計と簡単な性能評価を述べた。現在、ミリネット上に実装した場合の評価も行っている。

【参考文献】

[1] Kai Hwang, Zhiwie Xu:"Scalable Parallel Computing",WCB McGraw-Hill,Chapter 5,10,