分散メモリ型並列計算機上での仮想共有メモリの実装について*

3P－1

成瀬 彰† 渡邉 豊英†
名古屋大学大学院工学研究科情報工学専攻§

1 はじめに

大規模な並列アプリケーション開発において、メッセージ送受信の効率的な同期処理を実現することは一般に困難である。また、メッセージの送受信のタイミングが想定できない場合、例えば機能並列化されたタスク間でデータを共有させる場合に、メッセージ・パッシングは能率的に動作しない。

このような問題に対し、分散メモリ型並列計算機で仮想的に共有メモリを実現する手法、すなわち仮想共有メモリ (Pseudo Shared Memory) を提案する。本PSMではユーザの代わりに通信の同期を取るので、ユーザは通信のタイミングを意識することなく、プログラミング可能となる。また、機能の異なるプロセス(以下タスクという)間通信で発生する同期のためのアイドル時間を回避できる。以下、PSMの構成・機能・評価について述べる。

2 仮想共有メモリ

メッセージ・パッシング以外に、データを共有する手段をもたない並列計算機に、ソフトウェアで共有メモリを実現する。そのためのアプローチとして、各プロセッサ(以下セルという)の局所メモリを共有メモリの一部として割り当て、これらへのアクセスを排他的に制御する方法を用いてPSM機能を実現する。

PSMはPSMデーモンとPSMライブラリから構成され、PSMデーモンは各セルの局所メモリ内に確保されたPSMデータ領域へのデータ・アクセスを管理し、PSMライブラリはPSMデータ領域に対するアクセス関数をユーザに提供する。ユーザはPSMライブラリ・アクセス関数を使用することで、データ・ロケーションを考えないで、容易にプログラムを作成することができる。

2.1 構成

PSMの構成は図1の通りである。図1が示すように、ユーザ・タスク間の通信媒介として常に受信可能状態にあるPSMデーモンを配置することで、同期予測困難な通信をサポートする。

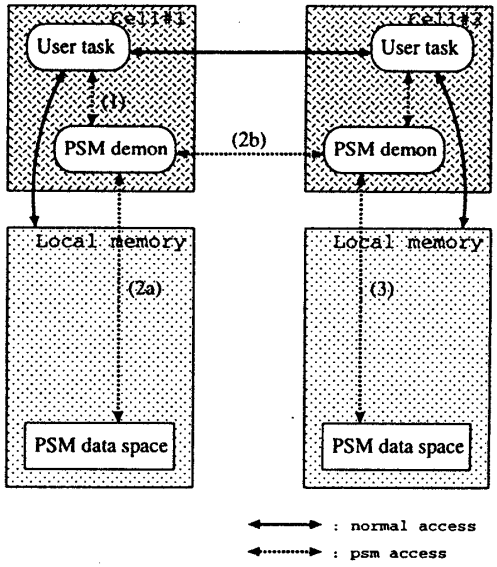

図 1: PSMの構成

2.2 仮想共有領域の管理

PSMデータ領域へのアクセスはPSMデーモンが一括して管理する。PSMデータ領域を確保するアクセス関数が呼び出されると、その時点でPSMデータ領域を静的に各セルに割り当てる。以後、データの配置はPSMデーモンが管理するので、ユーザはPSMデーモンを経由することで実際にデータが保持されている場所に関係なく、PSMデータ領域のデータにアクセス可能である。

基本的にPSMデータ領域へのアクセスは以下の手順で実行される。

(1) ユーザ・タスクで呼び出されるPSMのアクセス関数はセル内のPSMデーモンに要求を送る

(2a) アクセス・データが自己の局所メモリ内にあればPSMデーモンはそれにアクセスする

(2b) それ以外の場合、アクセス・データを保持する局所メモリを管理しているセルのPSMデーモンに要求を送る

(3) その要求を受け取ったPSMデーモンは局所メモリにアクセスする

*An Implementation of Pseudo Shared Memory on Distributed Memory Parallel Computer

†Akira NARUSE and Toyohide WATANABE
§Department of Information Engineering, Graduate School of Engineering, Nagoya University

2.3 アクセス関数

PSM ライブラリが提供する主要なアクセス関数を以下に示す。

- ◇ **psm_malloc** PSM データ領域を局所メモリに確保する
- ◇ **psm_free** PSM データ領域を解放する
- ◇ **psm_read** PSM データ領域のデータを読み出す
- ◇ **psm_write** PSM データ領域にデータを書き込む
- ◇ **psm_lock** 他セルからのアクセスをロックする
- ◇ **psm_unlock** ロックを解除する

ユーザは最初に psm_malloc で PSM データ領域を確保する。図 1の PSM データ領域が psm_malloc で確保される領域である。以下のプログラム例では、識別名 "a" の変数が全体で 32*32(1024) 個確保される。各局所メモリで確保されるサイズはセル数に依存し、セルが 4 台なら 256 個、16 台なら 64 個である。

```
(ex)
  cell_main()
  {
        ......
    psm_malloc("a", 32, 32, 1);
        ......
    val = psm_read("a", ai, aj);
        ......
    psm_write("a", ai, aj, val);
        ......
    psm_free("a");
        ......
  }
```

排他制御は psm_lock, psm_unlock を明示的に呼び出すことで実現される。排他制御の際に発生するデッドロックは現在特に考慮していない。

3 実装

実装は分散メモリ型並列計算機である富士通社製 AP1000 上に行なった。AP1000 はプログラムを実行するときのみ、セル OS を動作させる方式を採用しているので、PSM デーモンをユーザ・タスクと同等の一つのタスクとして実装した。

4 評価

行列のかけ算 $(A = B*C)$ を PSM を使用した場合と使用しなかった場合とで、同様のアルゴリズムを用いて、それぞれプログラムを作成し、比較した。行列の要素は全て PSM データとし、PSM データへのアクセスと PSM を使用しない場合の対応するデータ・アクセスとをアクセスの種類に応じて 4 項目に分類、測定した。その結果を表 1に示す。表内の数字は単位データあたりのアクセス時間 $(\mu sec/data)$ である。

表 1: PSM と non-PSM の比較結果

	($\mu sec/data$)	PSM 使用	使用せず
read	local	52.2	8.0
	remote	328.8	290.9
write	local	52.6	8.1
	remote	92.4	66.1

PSM は速度的には PSM を使用しない場合の 0.15 ～ 0.88 倍であり、全ての項目で劣っている。特に自己の局所メモリに対するアクセスのディレイが顕著である。これは、アクセス毎に PSM デーモンがアクセス・データの保持されている局所メモリを計算しているためである。

5 おわりに

分散メモリ型並列計算機にソフトウェアで仮想的に共有メモリを実現する方法について言及し、実際に AP1000 で PSM を実装した。PSM は処理速度の側面から考えると、それほどメリットはないが、セルへのデータ分散を意識する必要がない、通信相手を特定する必要がない、コーディングが容易になるなど、速度を補うだけのユーザビィリティがある。今回は、PSM デーモンをユーザ・タスクとして実装したため、OS が定期的に PSM デーモンに CPU を割り当ててしまい、結果として CPU のアイドル時間が長くなってしまった。処理速度を考慮した場合、OS に取り込む形態で実装する必要がある。

参考文献

[1] J. Nieplocha, R. J. Harrison and R. J. Littlefield: "*Global Arrays: A Portable "Shared-Memory" Programming Model for Distributed Memory Computers*", Proc. of Supercomputing '94, pp. 340–349, 1994.

[2] 石畑宏明, 稲野聡, 堀江健志, 清水俊幸, 池坂守夫: "高並列計算機 AP1000 のアーチテクチャ", 電子情報通信学会論文誌 D-I, Vol. J 75-D-I, No. 8, pp. 637–645, 1992.