Cenjuにおける並列プリミティブの実現

5L-5

松下 智, 中田 登志之, ‡浅野 由裕, 小池 誠彦

日本電気(株) C&Cシステム研究所, ‡日本電気技術情報システム開発(株)

1 はじめに

我々は並列回路シミュレーション向けの64台構成のマルチプロセッサシステムCenjuを開発した[1]. Cenjuの通信網はバスと多段ネットワークからなる階層構成であり, 疑似分散共有メモリのユーザビューを提供している. Cenjuではクラスタ間アクセスのうちwriteのみハードウェア化している. ユーザはソフトウェア的に提供されるクラスタ間read, および排他制御/非同期事象管理のための遠隔手続き呼びだし(RPC: Remote Procedure Call)を用いてアプリケーションを記述することができる[2].

さらに我々は, C言語, Fortran77からライブラリルーチンとして使用可能な並列プリミティブを整備した.

本稿ではCenjuの並列プリミティブについて, 設計指針, 内部インプリメント, 記述性, 評価結果について報告する.

2 並列プリミティブの設計指針

Cenju並列プリミティブの利点

- 高水準化
  - 記述の容易化.
  - バグ混入の防止. クリティカルセクションの見落としを減らす, 予期せぬ非決定性によるバグの混入を防止する.
  - アーキテクチャへの依存性を減らし, 後継機に対するソースの互換性を上げる.
- 高速化
  - 並列プリミティブはハードウェア構造を意識し, 最適なインプリメントを行なう. これにより, ユーザは容易にCenjuの性能を充分引きだしうる.

並列プリミティブの選択に当たっては, 既存のプリミティブ[3]を参考に, (1)記述性が良いものであること, さらに, (2)Cenjuでハードウェア化されているクラスタ間writeのみを用いて記述できるものを選択した. このため, ユーザはハードウェア構成を熟知しなくても最大限の性能を引きだすことが可能になる.

3 並列プリミティブの種類

Cenjuの並列プリミティブのうち, 代表的なものを以下に示す.

CJForki 並列実行の開始, Mach[4], Dynix[3]のforkと同様に, コンテクストをCenju内のプロセッサにブロードキャストし並列実行を開始する. ただし, データ, コードは共有できない.
CJSysBarrier 全てのプロセッサ間で同期をとる.
CJBarrier 設定したプロセッサグループで同期をとる.
CJRpc 他PEの手続きを呼びだす, 呼びだされたルーチンの終了まで呼びだし側はブロックする.
CJNbRpc 他PEの手続きをポインタで指定し呼び出す. 呼びだし側はブロックしない.
CJSend, CJRecv 領域(配列)の転送. 転送が終了するまで双方はブロックする(同期型転送).
CJASend, CJARecv ブロックしない(非同期型)転送.

4 並列プリミティブの内部インプリメント

第2章で述べた性能向上の理由から, 本並列プリミティブはインプリメントにおいて, PE間のデータ転送をできる限りwriteで記述している. 以下, barrierを例にとり概要を示す.

barrier barrierを行なうPEは, 同期テーブルをもつマスタPEとその他のスレーブPEに分類される. barrierを行なう前に, 並列プリミティブbarron(master)によって, マスタPEを指定する. マスタPEごとに同期グループを分割することが可能である.

図1に示すように, マスタPEにはmaskテーブルとsyncテーブルが作成される. マスタプロセッサが同期の成立をスレーブに伝達するため, スレーブにはackフラグが置かれる. maskテーブルは, barronディレクティブによりスレーブからセットされ同期の対象になる. 全てのスレーブがセットを終了したことを確認するために, 動作中の全てのプロセッサを同期対象にしたsys-barrierを用意しこれを解決する. 以下, Cライクにアルゴリズムを記述する. 図中, SET, RESETは定数, 記号<-は, クラスタ間writeアクセスを示す.

マスタ側

```
foreach(pe)
      if ((mask[pe] == SET) and (sync[pe] != SET))
            continue;
foreach (pe)
      ack <- SET;
```

スレーブ側

```
ack = RESET
sync[pe] <- SET
while (ack == RESET)
        ;
```

Implementation of Parallel Primitives on Cenju

Satoshi MATSUSHITA, Toshiyuki NAKATA, ‡ Yoshihiro ASANO and Nobuhiko KOIKE

NEC Corporation and ‡NEC Scientific Information System Development Ltd.

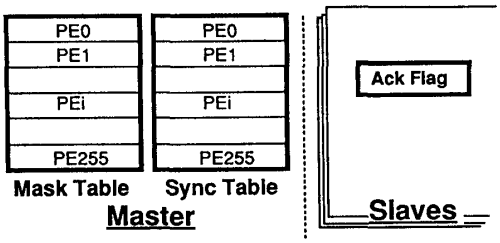

図 1: Barrier のデータ構造

5 記述力

本プリミティブを用いた場合, (1)barrier と非ブロック型 send, receive を用いた記述例を下に示す. このほか, (2)barrier と RPC による動的データの扱い, (3)send, receive によるパイプライン処理, (4) RPC による分散型のスケジューラ (5) 非ブロック型 RPC による集中型のスケジューラなど, さまざまなスタイルが容易に記述可能である.

並列プリミティブの記述例 (静的なデータ交換をするもの)

```
Forki(); /* 並列実行の開始 */
if (groupA) {
        CJBarron(peA);  /* PE A をマスタにした,
                           同期グループ A */
} else {
        CJBarron(peB);  /* 同期グループ B */
}

CJSysBarrier(); /* 同期グループの設定終了待ち */
/* 配列 vtx の計算 */
      :
/* vtx を近隣のプロセッサに配布 */
/* CJASendDbl(送り先, 配列の先頭, 要素の個数,
                                  要素間の飛び幅) */
CJASendDbl(ilpid, &vtx[0][nrl], ld+1, nd+1);
CJASendDbl(iupid, &vtx[0][nru], ld+1, nd+1);
/* 同期 */
CJBarrier(groupA ? peA: peB);
/* CJARecvDbl(送り元, 受け配列の先頭, 要素の個数,
                                  要素間の飛び幅)*/
CJARecvDbl(ilpid, &vtx[0][nrl-1], ld+1, nd+1);
CJARecvDbl(iupid, &vtx[0][nru+1], ld+1, nd+1);
```

6 性能評価

メモリアクセスおよび RPC の性能を 表 1 に示す. クラスタ間 read は, ソフトウェア的に記述しているためクラスタ間 write に比較して遅い.

第4章で示した, barrier, send の性能を 表 2 に示す. 64 台構成時に, 最後の barrier 発行後, もっとも早い PE が barrier から抜けるまでの時間を First-done, 最後の PE が抜けるまでの時間を Last-done で示す. First-done の値がテーブルサーチのオーバヘッドを First-done と Last-done の差がマスタからスレーブへの ack 配布のオーバヘッドにあたる. これまで, Cenju に実装されてきたアプリケーションは粗粒度のものが多かったため, 同期のオーバヘッドは問題にならなかったが, 今後, データ構造の工夫により高速化しより細粒度の処理にも対応していく予定である.

表 1: 通信性能

Access Operation		Access Time
Local Access		250 nsec
Intra-Cluster Global Access		750 nsec
Inter-Cluster Global Write Access	(Access)	5.2 μsec
	(Throughput)	800 nsec
	(... + Pipe Clog)	2.4 μsec
Inter-Cluster Read Access		240 μsec
Inter-Cluster RPC	(Latency)	640 μsec to 1.2msec
	(Throughput)	200μsec

表 2: barrier, send-receive の性能

Access Operation		Waitng Time
CJSysBarrier	(First-done)	770 μ sec
	(Last-done)	1500 μ sec
CJSend, CJRecv	(per Long Word)	13μsec

send については, ブロック型の send, receive を用いて先に receive が発行された条件での転送時間をバスサイクル 1 回当たりに換算したものを示す. 表 1 と比較すれば, write アクセスに近い性能が引きだせていることがわかる.

7 終わりに

Cenju は write アクセスのみを強化した, 回路シミュレータに特化したアーキテクチャを採用しているが, 今回, 汎用的な並列プリミティブを効率良くこの上にマップできることを示した.

さらに, 我々は現在, Cenju 上で回路シミュレータ [5], 故障シミュレータ, LSI ルータ [6] など CAD 応用, モンテカルロシミュレータ, 偏微分方程式など科学技術計算, ニューラルネットワークシミュレータなどの並列化を通して本プリミティブの評価整備を行なっている.

今後はアプリケーションの評価をもとに並列プリミティブの整備を進め, 記述性の良い並列言語 / 並列 OS に発展させていくとともに, Cenju のアーキテクチャをより汎用な並列処理に適したものに発展させていきたいと考えている.

参考文献

[1] 松下, 中田, 梶原, 浅野, 小池 : 並列シミュレーションマシン, 情処 37 回全国大会予稿集, (1988)

[2] 松下, 中田, 梶原, 浅野, 小池 : 並列 *CAD マシン Cenju のソフトウェア体系*, 情処研報, Vol.89, No.99, pp. 93-100 (Nov. 1989)

[3] Robert G.Babb II : *Programming Parallel Processors*, Addison-Weisley, (1987)

[4] Michael B.Jones, Richard F.Rashid: *Mach and Machmaker;Object-Oriented Distributed Systems*, CMU Techinical Report, CMU-CS-87-150, (1987)

[5] Nakata,Tanabe,Onozuka,Kurobe,Koike: *A Multiprocessor System for Modular Circuit Simulation*, Proc. ICCAD '87, pp.364-367, (1987)

[6] 山内, 中田, 石塚, 西口, 小池: 並列シミュレーションマシン *Cenju* 上の *LSI ルータ*の評価, 情処 40 回全国大会予稿集, (1989)