Committed-Choice 型言語 Fleng の ワークステーションクラスタ上処理系の定量的評価

大内 敦夫、荒木 拓也、田中 英彦
東京大学工学部

1 はじめに

筆者らは、Committed-Choice 型言語の一つである Fleng をワークステーションクラスタ上へ実装した [1]。本稿では、実装した処理系の定量的な評価を行ない、今後改善すべき点を指摘する。

2 Committed-Choice 型言語 Fleng

Committed-Choice 型言語の一種である Fleng のプログラムは、次のような形をしたホーン節から構成されている。

$$H :- B_1, B_2, \cdots, B_n. \ (n \geq 0)$$

Fleng プログラムの実行は次のようにして行なわれる。

定義されたホーン節の集合に対して初期ゴールが与えられる。それに対応する定義節を一つだけ選び出し、そのボディ部の各述語 $B_1, B_2, \cdots, B_n$ を新たなゴールとする。この操作をリダクションと呼ぶ。そして、新たに生成されたゴールに対してまたコミットする定義節を選ぶ。これを、すべてのゴールが処理されるまで繰り返す。

Fleng では、単一代入変数や、ゴール中のバインドされていない変数を待つサスペンド・アクティベートの機構等により、プロセス間の排他制御や同期をあまり意識することなくプログラムを作成することができる。

3 実装に使用した技法

ワークステーションクラスタ等の分散メモリ型並列計算機の弱点として、通信のレイテンシが大きいことが挙げられる。このため、Committed-Choice 型言語を並列計算機に実装する際に、通信を効率よく行うための技法が考案されている。本節では実装に用いた技法を概説する。

3.1 Eager transfer

リストのような構造データを読み出して他のプロセッサに渡す時には、その時点で具体化しているデータをすべて渡す、eager transfer と呼ばれる戦略によってデータを転送する。この戦略は、構造データのトップレベルだけを転送する lazy transfer と呼ばれる戦略に比べて全体として速度が速いことが、Committed-Choice 型言語の一つである KL1 の処理系 KLIC の分散メモリ環境への実装において示されている。

Quantitative Evaluation of an Implementation of Committed-Choice Language Fleng on Workstation Cluster
Atsuo OUCHI, Takuya ARAKI, Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

3.2 外部メモリ参照と分散ガーベジコレクション

本実装では、各プロセッサの持つ空きメモリが不足した時に、ローカルにガーベジコレクションが行われる。この時、他のプロセッサにあるゴールから参照されているようなメモリをガーベジとして回収しないよう、輸入表 / 輸出表及び WEC という技法を用いている。

3.3 外部参照変数とのユニフィケーション

バインドされていない外部参照変数がユニフィケーションの対象となる場合、その変数の輸出元に対してバインドするよう通知し、また外部参照のループができないようにしなければならない。現在の実装では、ユニフィケーションの際に一旦外部参照を完全に解決し、プロセッサに付けた通し番号によって外部参照の方向を一意に定めるようにしているが、この点に関してはまだ改良の余地があると考えられる。

4 処理系の評価

4.1 通信時間

本処理系では、1 台のホストと複数台のプロセッサを用い、ホストは入出力や実行管理を行ない、各プロセッサが実際の計算を行なう。各マシン間は図 1のように接続されている。

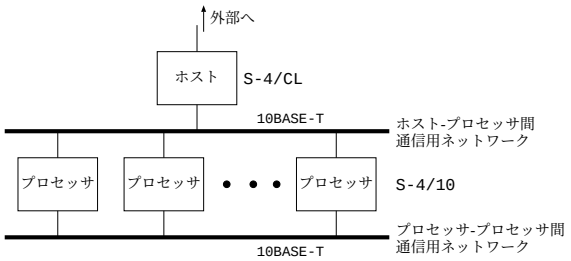


図 1: マシン間接続図

ホスト - プロセッサ間 [A]、及びプロセッサ - プロセッサ間 [B] の通信時間を測定した。測定方法は、一方からデータを送信し、他方がそれを送り返して、ラウンドトリップ時間を測定し、その 1/2 を 1 回の通信時間とした。結果は表 1の通りである。

基本的に、データサイズが大きくなるほど通信時間も増加しているが、データサイズが大きくなるほど、データサイズと通信時間の比は小さくなっている。つまり、通信を行なう際には 1 回のデータサイズが大きいほど有利ということが示された。

サイズ (byte)	[A](msec)	[B](msec)
64	12.9	12.1
256	22.4	19.8
1024	22.0	23.1
4096	55.8	26.4
16384	188.5	44.4

表 1: 通信に要する時間

また、殆どの場合通信には 20ms 以上掛かっているが、データのサイズが特に小さい場合 (256 バイト以下) には、時折 5ms 以下で通信できることがあることが観察された。これは、通信に有利な何らかの条件が揃ったタイミングで小さいサイズのデータが送られた時に、通信が高速に行なわれることがある、ということであると考えられる。

4.2 計算時間

プロセッサを 1 台だけ使用し、負荷分散ルーチンを使用しないクイックソート (データ数 500) のプログラムを実行して実行履歴を取り、1 回のリダクションにどの程度の時間を要するか測定した。結果は表 2 の通りである[1]。

実行時間 (μsec)			ゴール数
0	〜	5	0
5	〜	10	0
10	〜	15	4844
15	〜	20	5338
20	〜	25	6696
25	〜	30	2989
30	〜	35	85
35	〜	40	38
40	〜	45	20
45	〜	50	1
50	〜		170
総計			20181

平均 22.5μsec

表 2: リダクションに要する時間

この表から、ほとんどのゴールは 30μsec 以内にリダクションが終了することが分かる。これを前述の通信時間と比較すると、通信時間とリダクションに要する時間との比はほぼ 1000 倍のオーダーとなっている。

4.3 通信回数とデータサイズ

4 台のプロセッサを用いてクイックソート (データ数 1000) のプログラムを実行し、通信の行なわれた回数と通信したデータのサイズを測定した。また同時に比較のためにリダクション回数を測定した。結果は表 3 の通りである。

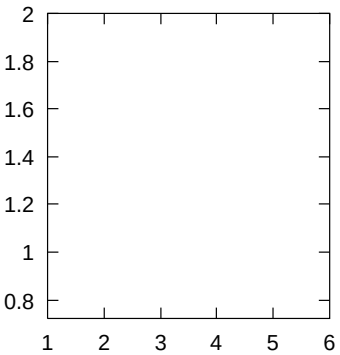
大きいデータサイズの通信は、ゴールの移送や外部参照のユニフィケーションの際の eager transfer 戦略によるものであり、64byte 以下のサイズの通信は応答やプログラムの終了通知のためのものである。Eager transfer 戦略によって、小さいサイズの通信が多数発生するのを抑え、通信効率を改善していることが分かる。

		データサイズ (byte)	通信回数 (回) プロセッサ番号 P1	P2	P3	P4
	〜	63	3	3	3	3
64	〜	255	0	0	0	0
256	〜	1023	1	0	1	1
1024	〜	4095	0	0	0	0
4096	〜		2	1	0	0
総計			6	4	4	4
リダクション回数 (回)			26371	17577	2027	564
実行時間 (msec)			701.5	415.6	65.4	13.8

表 3: 通信回数及びデータサイズ

しかし一方で、リダクション回数及び実行時間がプロセッサによって大きく異なっていることが分かる。理想的に負荷分散が行なわれていれば、リダクション回数は全プロセッサでほぼ等しくなる筈である。つまり、負荷分散が効率的に機能していないことが分かる。

負荷分散機構の影響を示すデータとして、同じくクイックソート (データ数 1000) でプロセッサ数を変えた時の実行速度比を図 2 に示す。この図では実行速度がプロセッサ数に比例していないが、その原因の一つとして負荷分散機構が不十分であることが挙げられると考えられる。



[1] 一度サスペンドして後でアクティベートされたゴールは 2 回カウントされている。以下同様。