並列PROLOGシステム"Paralog"の性能測定

5D-5

相田 仁，田中英彦，元岡 達
（東京大学工学部）

1．はじめに

我々は、論理プログラミング言語を並列処理・実行するシステム"Paralog"（Parallel logic programming system）の研究を行なってきたが、この度，その第一版が完成したので、その性能を測定し、システムを拡張した際の性能向上の可能性について検討を行なった。ここではその結果を報告する。

2．Paralog 第一版の概要

論理プログラミング言語の並列処理における一般的問題点，およびインプリメントの基本的方針については、前回の大会で報告した[1]。今回インプリメントした第一版の特徴は以下のようである。

i) 言語仕様はcut symbol 等を含まない pure prolog（=horn clause）であり、データ型としてはsymbol,functorの他.small integer がサポートされている。
ii) or-並列（altenative の並列実行）のみを行なう。
iii) alternativeの実行順序は、原則として breadth-first に従う。
iv) 入力ストリームをfileに切換えることが可能である。

このParalog第一版は、TOPSTAR-II上に、当面、CM（制御モジュール）1台，PM（処理モジュール）5台に限定して実装を行なった。システムは、割込処理を含め、C言語で記述され、大きさは、CM側約18 kB，CM側約10 kBである。また、性能評価の参考とするため、同一構造のPROLOG処理系を、depth-firstの実行順序に従うものとbreadth-firstのものの2通り単一プロセッサ上に作成した。

3．応用プログラム例による性能測定

図1に数式簡単化のプログラム例を示す。プログラムでは、効率の点から、数式をより簡単にする述語"simpl"と、そのままでもよい"equiv"を使いわけている。このプログラムの一実行例において実行時間を測定し、その逆数（すなわち実行速度）をPM台数に対しプロットしたものを図2に示す。なお、この実行例で引数として与えたのは、別のプログラムで$x^2 \cdot \exp(x^2)$を数式微分した結果である。図1のプログラムの場合、depth-first探索を行なえば、ほぼ最簡形が最初の答として得られるのに対し、Paralog第一版はbreadth-firstである

```
+equiv(*x,*y)-simpl(*x,*y).

+simpl(exp(0),1).

+simpl(power(*x,0),1).
+simpl(power(*x,1),*y)-equiv(*x,*y).

+simpl(times(0,*x),0).
+simpl(times(1,*x),*y)-equiv(*x,*y).
+simpl(times(*x,0),0).
+simpl(times(*x,1),*y)-equiv(*x,*y).

+simpl(plus(0,*x),*y)-equiv(*x,*y).
+simpl(plus(*x,0),*y)-equiv(*x,*y).

+simpl(times(*a,*b),*c)-ARITH(*a,*b,0,*c).
+simpl(plus(*a,*b),*c)-ARITH(1,*a,*b,*c).

+simpl(times(power(*x,*a),power(*x,*b)),*y)
  -equiv(power(*x,plus(*a,*b)),*y).

+simpl(plus(times(*a,*x),times(*b,*x)),*y)
  -equiv(times(plus(*a,*b),*x),*y).

+simpl(plus(times(*x,*a),times(*x,*b)),*y)
  -equiv(times(*x,plus(*a,*b)),*y).

+equiv(times(*a,times(*b,*c)),*x)
  -equiv(times(times(*a,*b),*c),*x).
+equiv(plus(*a,plus(*b,*c)),*x)
  -equiv(plus(plus(*a,*b),*c),*x).

+simpl(exp(*x),*z)
  -simpl(*x,*y)-equiv(exp(*y),*z).
+simpl(power(*x,*y),*z)
  -simpl(*x,*u)-equiv(power(*u,*y),*z).
+simpl(power(*x,*y),*z)
  -simpl(*y,*v)-equiv(power(*x,*v),*z).
+simpl(times(*x,*y),*z)
  -simpl(*x,*u)-equiv(times(*u,*y),*z).
+simpl(times(*x,*y),*z)
  -simpl(*y,*v)-equiv(times(*x,*v),*z).
+simpl(plus(*x,*y),*z)
  -simpl(*x,*u)-equiv(plus(*u,*y),*z).
+simpl(plus(*x,*y),*z)
  -simpl(*y,*v)-equiv(plus(*x,*v),*z).

+equiv(*x,*x).
```

図1　数式簡単化プログラム例

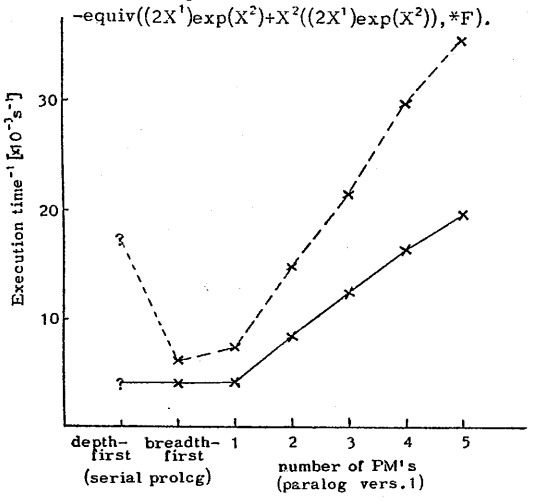

図2　実行速度

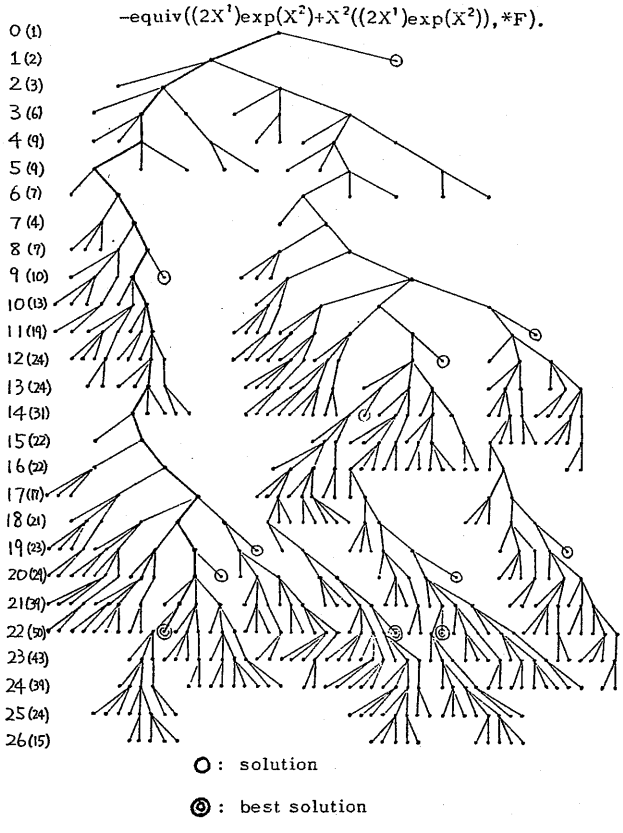

図3　探索実行の木

最簡解を得るまで			全実行		
最短	depth-first	breadth-first	レベル数	node数	最大幅
(a)	(b)	(c)	(d)	(e)	
23	127	353	27	513	50
	(b/a)	(c/a)		(e/a)	
	5.5	15.3		19.0	

表1　たどるべきnode数

ため、全実行が終了しないとどれが最簡な解であるかわからない。そこで図2では、全実行が終了するまでの時間を実線で、最簡解が得られるまでの時間を点線で示してある。図2から、実行速度はPM台数に対しほぼlinearに向上し、単一プロセッサdepth-firstで最簡解を得るよりも早く全実行を終了できることがわかる。また、PM台数を増すことにより処理の追い越しがおこり、良い解が早く得られる傾向があることなどもわかった。

4．システム拡張時の性能向上の可能性

今回実装したPM5台の範囲より今後さらにシステムを拡張した際、どの程度まで性能向上が期待できるかを検討するため、図2の例題について実際の探索実行の様子を示す木を求めた。これを図3に示す。図3で○印のnodeは解の得られたnode，特に◎印は最簡解が得られたnodeを表している。この木において実行の際にたどるべきnodeの数を求めると表1のようになる。表から、PM台数を増やすことにより（この例の場合50台程度迄）まだ一層の性能向上が期待される。しかし、breadth-first探索は木の最大幅にみあうだけのメモリ等の資源を必要とするため、さらに複雑な本格的応用に用いるためには、優先度評価などによるスケジューリングが必要になると考えられる。

5．おわりに

Paralogオー版は応用プログラムの持つ並列性をとり出し、PM台数5台までの範囲でほぼlinearに性能向上することが確認された。また、実行木の解析から、システムを拡張することにより一層の速度向上が期待できることを示した。

今後はand-並列の導入，スケジューリングの検討などを進めてゆきたい。

参考文献

[1]相田，田中，元岡：データフローマシン"TOPSTAR-II"を用いたPROLOG並列処理試作システムの検討，第23回情報処理全国大会 4E-3，1981