スタック変数の導入による並列論理型言語KL1の高速実行

4E－3

杉山　奈美代†　　大野　和彦‡　　中島　浩†
†豊橋技術科学大学 情報工学系　　‡京都大学 工学研究科

1. はじめに

並列論理型言語KL1の高速実行を妨げる要因として、実行時の並行性制御によるオーバーヘッドが挙げられる。そこで伊川らは、静的に実行順序を決定されたゴール系列をスレッドとし、このスレッドを並行実行単位とすることでオーバーヘッドの軽減を試み、無駄な実行中断／再開の削減による性能向上を達成した[1]。しかし一つのスレッドの実行性能に関しては、ゴールの実行管理にスタックを用いて高速化を図ったものの、その効果は小さかった。

我々はこの理由が変数の割付方法にあると考え、論理変数をスタックに割り当てる方法(Goal Stacking、Environment Stacking)を採用することによって、KL1の高速実行を図る。

2. スタック変数

KL1の標準的な処理系であるKLIC[2]では、複数のボディ・ゴールを有するクローズの実行時に、2番目以降のゴールに対してゴールレコードと呼ぶ構造をヒープに生成する。ゴールレコードには、対応する述語が何かを示す情報と、ゴール引数が格納される。例えば、リスト1に示すようなプログラムでは、図1のように二つのゴールレコードが生成される。また代表的なProlog処理系であるWAM[3]における局所変数に相当する変数、すなわち上記の例のL1とL2は、ゴールレコード中に割付けられる。

一方、伊川らの方法[1]では、ゴールレコードに相当する構造をスタックに生成するが、局所変数はヒープに割付けられ、スタック上のゴールレコードには変数へのポインタが格納される。その結果、たとえば上記のL2の値を第2ゴールが参照する際に、1段の余分なデレファレンスが必要となる。

そこで本研究では、余分なデレファレンスが生じないように、局所変数をスタックに割付けることとした。また、このスタック変数の導入によって、

- ヒープ消費量が減少し、GC回数が少なくなる。
- 参照局所性が向上する。

という利点もある。

リスト1　quick sortの一部

```
qsort([X|L],R,R0) :- true |
        partition(L,X,L1,L2),
        qsort(L2,R1,R0),
        qsort(L1,R,[X|R1]).
```

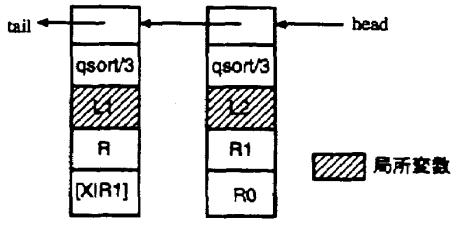

図1　標準KLIC

スタック変数は、それを参照する最後のゴールが呼び出される時点でスタックから除去されるので、以下の条件を満たすようにしなければならない。

(1) スタック中に存在するスタック変数へのポインタは、「祖先側」を指す。
問題となるのは未定義スタック変数間の単一化であり、WAMでは両者のアドレス比較によりこの条件を満たすようにしている。しかし我々の処理系ではスタック領域の動的拡張を行なうため、変数の世代とアドレスの間に一定の大小関係が成立しない。そこで、未定義スタック変数間の単一化では、新しい未定義ヒープ変数を生成し、両スタック変数がそのヒープ変数を指すようにする「大域化」を行なう。

(2) ヒープにはスタック変数へのポインタは存在しない。
未定義変数間の単一化では、アドレス比較によりヒープ／スタック変数の区別が可能であるので、条件を満たすようにすることができる。また生成される構造体に含まれる変数については、ヘッド引数の場合は動的大域化を行ない、それ以外の場合は静的に大域化するか、あるいは静的解析により未定義スタック変数でないことを保証する。

(3) スレッドの実行環境には、他のスレッドのスタック変数へのポインタは存在しない。
スレッド間共有変数を静的大域化することで解決できる。

(4) スタック変数を除去して引数レジスタに移すとき、変数は未定義ではない。
静的解析により、未定義スタック変数でないことが保証される。

3. Goal Stacking

Goal Stacking (GS) は伊川らの方法とほぼ同様であり、ゴールレコードに相当する構造をスタックに積む(図2)。ただし、局所変数はスタック変数としてスタックに割付けられる。また標準KLICのようなゴールレコードをつなぐリンクはなく、その操作も不必要である。

4. Environment Stacking

Environment Stacking (ES) では、WAMと同様に第2

An Efficient Implementation of a Concurrent Logic Language KL1 by Introducing Stack Variables
Namiyo Sugiyama†, Kazuhiko Ohno‡, Hiroshi Nakashima†
† Toyohashi University of Technology
Tempaku-cho, Toyohashi, Aichi 441-8580, Japan
‡ Kyoto University
Yoshida Honmachi, Kyoto 606-8501, Japan

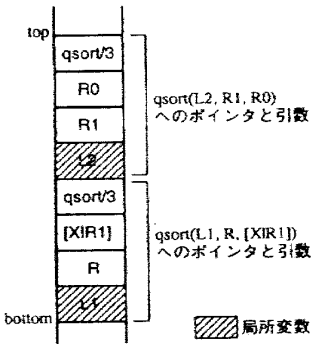

図2　Goal Stacking

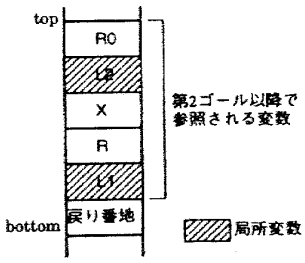

図3　Environment Stacking

ゴール以降で参照される変数と戻り番地からなる環境をスタックに積む。すなわち、n 個のゴールを持つクローズ Cでは、$i(<n)$ 番目のゴール g_iが完了すると Cに制御が戻り、$i+1$ 番目のゴール g_{i+1}の引数を環境を参照しながら引数レジスタに設定して、g_{i+1}を呼び出す。

たとえばリスト1の例では、第1ゴールを呼び出す前にスタックに図3のような環境が生成される。この環境中のR0とL2は第3ゴールでは参照されないので、enviroment trimingによって第2ゴールの呼び出しの際に除去される。KL1にはdeep backtrackがないため、WAMとは異なりこの除去操作を決定的に実行でき、また第2ゴールで初めて出現するR1を除去操作の後に割付けることができる。したがってWAMよりもスタックの伸長を抑制することができる。

5. 性能評価

GSとESの処理系を試験的に実装し、KLIC標準配布パッケージに含まれる9つのプログラムを用いて、標準KLIC (org)、GS、ESのそれぞれの性能を測定した。またGSについては全変数をヒープに割付けるもの (GSH) と、スタック変数を用いるもの (GSS) の両者の性能を測定した。この測定結果に基づき、9つのプログラムを表1のように分類し、それぞれの代表例の結果を表2に示す。

表1　プログラムの分類

分類	基準	プログラム
type1	org/GSS ≈ 1.0	nrev, deriv, primes
type2	org/GSS < 1.5	quick sort, kkqueen, pascal
type3	org/GSS < 2.0	factorial, hanoi

表2　計測結果

分類	例		org	GSH	GSS	ES
type1	nrev	実行時間 [ms]	199	207	206	197
		速度比	1.00	0.96	0.97	1.01
		GC[回数]	44	39	37	37
type2	kkqueen	実行時間 [ms]	560	452	430	405
		速度比	1.00	1.24	1.30	1.38
		GC[回数]	78	29	21	21
type3	factorial	実行時間 [ms]	214	231	120	114
		速度比	1.00	0.93	1.78	1.88
		GC[回数]	22	4	0	0

- nrev：リストの反転。30要素、1000回繰り返し
- kkqueen：N-Queens。queenの数=10
- factorial：11の階乗。10000回繰り返し

表2より、GC回数の減少と実行時間が短縮に強い相関があることがわかる。このGC回数の減少要因を調べるために、ヒープ消費量を測定したところ、標準KLICの消費量を1とした場合の各方法での消費量は図4のようになった。すなわちゴールレコードによるヒープ消費の比率が大きいものほど、スタックの利用やスタック変数導入の効果が高いことが明らかになった。

一方GSとESを比較すると、いずれのプログラムにおいてもESの方が高速である。Prologではバックトラックが生じる可能性があるため一般にGSよりもESが有利であるが、KL1ではバックトラックはなく、またヒープ消費量も同一である。したがって、ESの優位性は以下のような要因によるものと考えられる。

(1)　スタック変数が n 回出現するとき
　　GS：読出 $2n-1$ 回、書込2回
　　ES：読出 n 回、書込2回

(2)　構造体生成タイミング
　　GS：第1ゴール呼び出し直前
　　ES：構造体参照ゴール呼び出し直前

すなわちESの方が時間的参照局所性がよい。

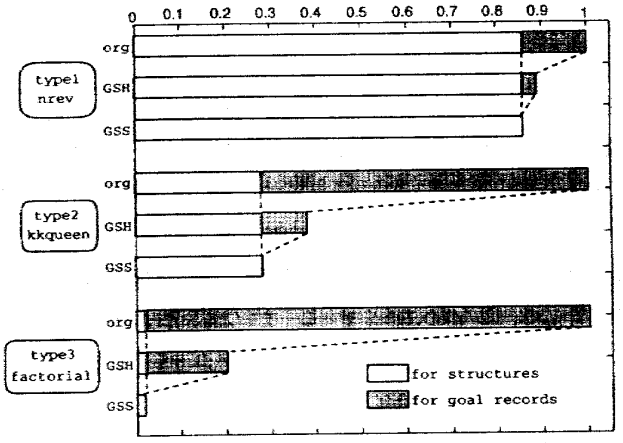

図4　ヒープ消費量

6. まとめ

スレッド化KL1へのスタック変数の導入を提案し、試験的な実装と評価を行なった。その結果、GS/ES共に標準KLICや従来のスレッド化実装に比べて、最大で約2倍の性能向上が得られることが明らかになった。この理由は、ヒープ消費量の削減とそれに伴うGC回数減少のためである。また、変数の参照回数や構造体の参照局所性により、ESがGSよりも優れていることが明らかになった。

参考文献

1) 伊川 雅彦 他, KLICにおけるゴール・スケジューリング最適化, 情処研報, PRO-10-8 (1996).
2) T.Chikayama, et al., A Portable and Reasonably Efficient Implementation of KL1, *Proc. PLILP'94* (1994).
3) D. Warren, An Abstarct Prolog Instruction Set, TR-309, SRI International (1983).