Vol. 27 No. 9 情報処理学会論文誌 Sep. 1986

ショートノート

クロススタックキャッシュを用いたブロック構造言語のためのアドレシング機構†

板野肯三†† 佐藤 豊††† 中村敦司††††

ブロック構造を持つ言語の動的変数に対する高速なアドレシング機構として不可欠なディスプレイレジスタを，クロススタックキャッシュ (CSC) と呼ぶ機構によって高速に維持管理する方式について述べる．CSC 方式では，再帰的な呼び出しや非局所的 GOTO，手続きパラメータなどを含む，すべてのブロックの入口・出口でのディスプレイの更新が，単純なハードウェアで高速に実現される．

1. はじめに

ブロック構造を持つ言語の実行時の記憶モデルを実現するためには，動的に生成される構造的記憶とそれに対するアドレシング機構が要求される．このため，各ブロックの動的変数を励起レコードとしてまとめ，これを単一のスタック上に配置して管理するが，この時入れ子に構成されたブロック中で，下位の（内側の）ブロックから上位の（外側の非局所的な）ブロックの動的変数を参照するために励起レコードのベースアドレスを求める方法が問題となる．

これに対処する方法は基本的には参照時に静的リンクをたどる方法[2),4),5)]と入れ子の各レベルで現在活性化されている励起レコードのベースアドレスの組であるディスプレイにより直接アクセスする方法[1),3),4),6)]の二つがある．静的リンクをたどる方法では非局所的な動的変数への参照の比率が大きい時には参照のコストが高くなり，ディスプレイを用いる方法より不利となる．しかし，手続きの呼び出しが頻繁に起こるような場合はディスプレイの管理コストは無視できなくなるので，この場合はディスプレイを用いる方法が不利になる．

このような欠点を補う方法として，ディスプレイを静的リンクを用いないで管理する方法[8)]がある．この方法では，各ブロックの入口で更新すべきディスプレイのスロットを保存し，各ブロックの出口ではその入口で保存したスロットを回復する．これはディスプレイの管理に関して無駄がなく，アルゴリズムも単純である．しかし，ブロック構造を持つ言語が標準的に備えている非局所的な GOTO や PASCAL の手続きパラメータ，ALGOL の名前呼びの取り扱いが非常に低速であるという欠点は取り除けない．

本論文では，このようなブロック構造を持つ言語のアドレシング機構をハードウェアで高速に実現するのに適した方式として，クロススタックキャッシュ (CSC) と呼ぶ方式を提案する[7),9)]．CSC 方式のディスプレイ管理アルゴリズムは基本的には前述の文献 10) と共通した概念をもとに，非局所的 GOTO や手続きパラメータなども効率的に取り扱えるとともに，単純なハードウェアで高速に実現することができる．以下に，CSC のハードウェアの構成と動作アルゴリズムを詳細に説明する．

2. クロススタックキャッシュ

ディスプレイは手続きの呼び出しと戻り時に伸縮するスタックとしての性質を持っているが，再帰的な呼出しと戻り時にはこのスタックとしての性質がそこなわれ，ディスプレイが破壊されるために，それを保持，回復することが必要となる．CSC はこれを実現するためにディスプレイをキャッシュとし，それを保持するためのスタックをバッファリング機構で結合する．

2.1 ハードウェアの構成

CSC は図1のように，静的なブロックの入れ子の

† A Dynamic Data Addressing Mechanism for Block Structured Languages Using Cross Stack Cache by KOZO ITANO (Institute of Information Sciences and Electronics, University of Tsukuba), YUTAKA SATO (Doctoral Program in Engineering, University of Tsukuba) and ATSUSHI NAKAMURA (College of Information Sciences, University of Tsukuba).

本研究は一部を文部省科学研究費補助金・試験研究(1) 58850063 および 61850061 によって補助された．

†† 筑波大学電子情報工学系
††† 筑波大学工学研究科
†††† 筑波大学情報学類

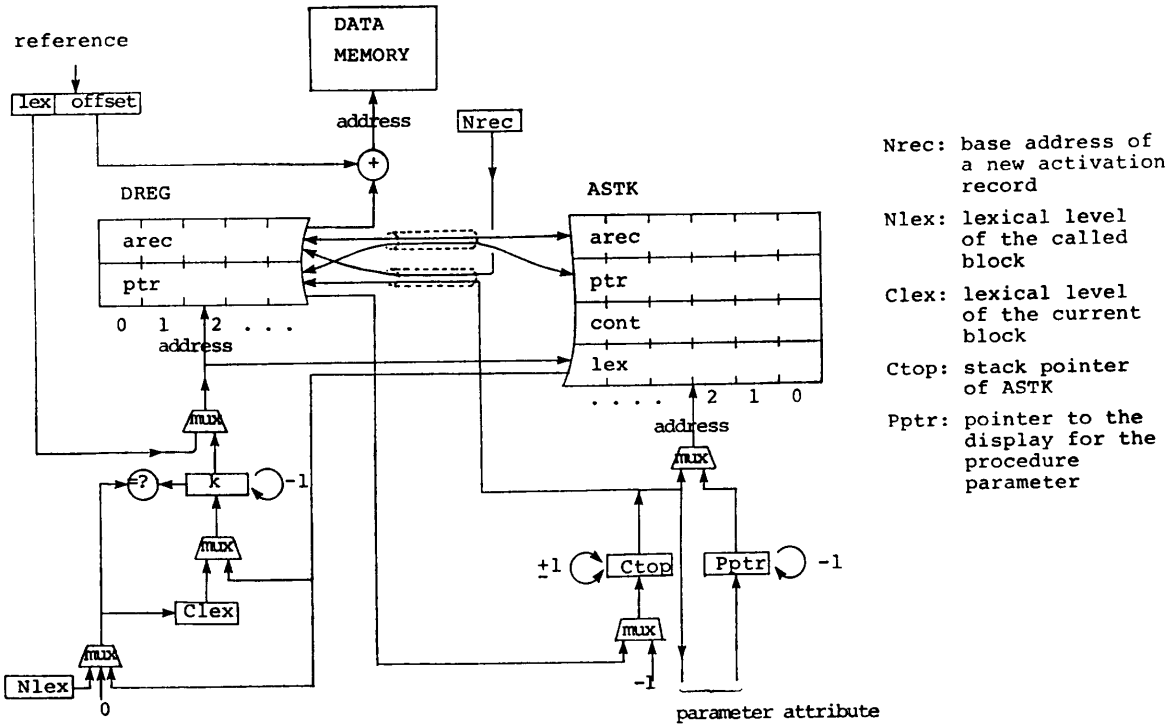

図 1 クロススタックキャッシュのハードウェア構成
Fig. 1 Hardware organization of cross stack cache (CSC).

最大レベル分のスロットを持つディスプレイレジスタ (DREG), 励起履歴保存用スタック (ASTK) と数個のレジスタで構成される.

DREG の i 番目のスロット (DREG [i]) は, 静的な入れ子レベル i のブロックの最新の励起レコードのベースアドレスを保持する arec フィールドと, このスロットの一つ以前の値が保存されている ASTK 中のエントリのアドレスを保持する ptr フィールドを含む. 一方, ASTK に保存される各要素は, DREG 中でのスロットを示す番号を保持する lex フィールドと, そのときの DREG の内容を保持する arec フィールドと ptr フィールド, および復帰時に一つ上のスロットとともに回復されるスロットであることを示すフラグを保持する cont フィールドを組として保持している.

また, 図中に示されるレジスタのうち, Ctop は, ASTK スタックのトップを指すポインタ, Pptr は手続きパラメータの環境を設定するためのポインタ, Clex は現在のブロックの入れ子レベル, Nlex は呼ばれた, または飛び先の手続きの入れ子レベル, Nrec は呼ばれた手続きに与えられる励起レコードのベースアドレスを保存する. このうち, Nlex, Nrec, Pptr には実行時に必要な値が外部から供給されるが, 残りのレジスタは内部でのみ使用される.

2.2 動作原理

CSC は, 手続きの呼び出し・復帰, 非局所的 GOTO の実行, 手続き／関数パラメータの受け渡しおよび呼び出し時に動作する. 以下の説明では, 動的変数のアドレシングに関する最小限の情報の保存と回復に関してのみ述べる.

(1) 手続きの呼び出しと戻り

手続きの呼び出し時に, 現在の入れ子レベルと同じかあるいはさらに深いレベルにある手続きを呼び出す場合と, 上位の親のレベルにある手続きを呼び出す場合では, CSC の動作がやや異なる. まず, 前者の場合の動作をアルゴリズムAに示す. ここでは, 呼ばれた手続きのレベルに該当するディスプレイのスロットが ASTK スタックに退避され, ディスプレイ上に新たな値が設定される.

アルゴリズムA. {同レベル以下の手続き呼び出し (自分自身への再帰も含む)}

A1. {Save the current slot in display register}
Increase Ctop by 1, and

set k←Nlex ,
ASTK[Ctop]. ptr←DREG[k]. ptr,
ASTK[Ctop]. arec←DREG[k]. arec,
ASTK[Ctop]. cont←'OFF',
ASTK[Ctop]. lex←k .

A2. {Set the current slot in display register}
Set Clex←k,
DREG[k]. ptr←Ctop,
DREG[k]. arec←Nrec .

上位の親のレベルへの手続きが呼び出される時は，非局所的 GOTO の処理に対処するために，レベル差の数だけのディスプレイのスロットを ASTK スタック上に退避したのち，ディスプレイ上に新たな値を設定する．この時，あとで一括して復元できるように，ASTK の cont フィールドを 'ON' にする．この動作をアルゴリズムBに示す．

アルゴリズム B. {上位レベルの手続き呼び出し}

B1. {Save the slot in display register corresponding the distance of lexical level}
Increase Ctop by 1, and
set ASTK[Ctop]. ptr←DREG[k]. ptr,
ASTK[Ctop]. arec←DREG[k]. arec,
ASTK[Ctop]. cont←'ON',
ASTK[Ctop]. lex←k ,
for Nlex<k≦Clex .

B2. {Procedure call}
アルゴリズムAを実行する．

手続きからの戻り時には，ASTK 上に退避されているもとの値をディスプレイ上に復元する．この動作をアルゴリズムCに示す．この時は ASTK スタック上の cont フィールドを検査し，まとめて復元すべきものはすべて復元する．

アルゴリズムC. {手続きからの戻り}

C1. {Recover display register}
Set k←ASTK[Ctop]. lex
DREG[k]. ptr←ASTK[Ctop]. ptr,
DREG[k]. arec←ASTK[Ctop]. arec,
decrease Ctop by 1, and
if ASTK[Ctop]. cont='ON', repeat this step.

C2. {Recover lexical level}
Set Clex←ASTK[Ctop]. lex .

(2) 非局所的 GOTO

現在のレベルからより上位の親のレベルに非局所的に GOTO が実行される場合は，そのレベルのディスプレイの ptr フィールドに保存されている ASTK のエントリより先を切り捨てるだけでよい．これをアルゴリズムDに示す．実際のディスプレイの復元は手続きからの戻りの時点で行われる．

アルゴリズムD. {非局所的 GOTO}

D1. {Recover pointer}
Set Clex←Nlex,
k←Clex,
Ctop←DREG[k]. ptr .

(3) 手続きパラメータ

名前呼びでパラメータとして渡された手続きなどはそれを渡す時点での環境で実行される．このため手続きをパラメータとして渡した環境（ディスプレイの内容）を，その手続きの実行時に回復するためにディスプレイを退避することが必要である．そこで，現在のレベルまでのディスプレイのスロットを ASTK スタックに一括して退避し，実際にこの手続きが呼び出される時点でこれをディスプレイに復元して使用するという方式をとる．ディスプレイの退避はアルゴリズムEに，この手続きの呼び出し時のディスプレイの復元はアルゴリズムFに与える．手続きパラメータが複数ある時も生成する手続きパラメータの数によらずディスプレイを1回だけ保存すればよい．また，アルゴリズムFを実行する前にアルゴリズムEにおいて得られた ASTK へのポインタを Pptr に設定するものとする．

アルゴリズムE. {手続きパラメータの生成}

E1. {Save the current slot in display register}
Increase Ctop by 1, and
set ASTK[Ctop]. ptr←DREG[k]. ptr,
ASTK[Ctop]. arec←DREG[k]. arec,
ASTK[Ctop]. cont←'ON',
ASTK[Ctop]. lex←k,
for k=Clex, Clex−1, ⋯, 0 .

E2. {Pass ASTK top address}
Ctop を手続きのアドレスと一緒にパラメータとして渡す．

アルゴリズムF. {手続きパラメータの実行}

F1. {While Pptr has old environment do}
If ASTK[Pptr]. cont='OFF', go to step F5.

F2. {Save the current slot in display

register}
Increase Ctop by 1, and
set k←ASTK[Pptr]. lex,
ASTK[Ctop]. ptr←DREG[k]. ptr,
ASTK[Ctop]. arec←DREG[k]. arec,
ASTK[Ctop]. cont←'OFF',
ASTK[Ctop]. lex←k.

F3. {Recover old display}
Set DREG[k]. ptr←ASTK[Pptr]. ptr,
DREG[k]. arec←ASTK[Pptr]. arec, and
decrease Pptr by 1.

F4. {Repeat step F1～F3}
Go back to step F1.

F5. {Procedure call}
実行する手続きと復活した環境の字句レベルによりアルゴリズムAまたはアルゴリズムBを実行する.

3. 考　　察

以上で説明したアルゴリズムは，非局所的 GOTO や手続きパラメータに対処するために，上位レベルの手続きの呼び出し時に一般に1スロット以上のディスプレイの保存動作が必要である．もし，非局所的 GOTO が存在しない場合は，アルゴリズムBはアルゴリズムAで代用でき，DREG 中の ptr フィールドは不要となる．さらに，非局所的 GOTO と手続きパラメータの両者が存在しない場合は，ASTK 中の cont フラグも不要となる．これらの情報はプログラムのコンパイルによって容易に得られる．

CSC では，すべてのデータ操作は ASTK と DREG のみに集中しており，この部分を高速なハードウェアで置き換え，手続き呼び出しや戻りの時に実行すべき実引数のプッシュ，戻り番地の保存，新しい励起レコードの生成などの他の処理に重ね合わせれば，ディスプレイの更新が全体の実行のオーバヘッドになることは，ほとんどないと考えられる．

動的な手続きの呼び出しの深さは ASTK の容量によって束縛されているが，ASTK スタックを仮想化して主記憶に拡張するか，ASTK スタックを主記憶に割り当ててキャッシュ化するなどの方法が考えられるので，ASTK を専用のハードウェアとすること自体には実用上の問題はないと考えられる．

4. おわりに

本論文で示した方式は，基本的にはディスプレイレジスタを一種のバッファメモリとしてとらえ，そのLIFO 型の振る舞いを利用した入れ替えアルゴリズムを適用するものである．このアルゴリズムは，異なる入れ子のレベルを制御が推移するときに，内部的には1レベルずつ移動させるように設計されており，入れ子の深い部分に移動する時と浅い部分に移動する時に必要な操作が対称になっており，この点からハードウェアの制御自体は単純に実現できる点に特徴がある．また，このような方式は，高級言語を指向したアーキテクチャの中でアドレシング機構の一部として実現することで従来のアーキテクチャとも整合するので，十分実用性のある方式であると考える．

参考文献

1) Wirth, N.: The Design of a Pascal Compiler, *Softw. Pract. Exper.*, Vol. 1, No. 4, pp. 309-333 (1971).
2) Wirth, N.: *Algorithms+Data Structures=Programs*, Prentice Hall, Englewood Cliffs (1976).
3) Gries, D.: *Compiler Construction for Digital Computers*, John Wiley & Sons, New York (1971).
4) Barron, D. W. (ed.): *Pascal—The Language and Its Implementation*, John Wiley & Sons, New York (1981).
5) Penberton, S. and Daniels, M. C.: *Pascal Implementation, P4 Compiler*, Ellis Horwood, West Sussex (1982).
6) Pyster, S.: *Compiler Design and Construction*, Van Nostrand Reinhold, New York (1980).
7) 山形朝義，杉原敏昭，沢辺正彦，板野肯三：UDEC (4)連想テーブルのハードウェア構成，情報処理学会第31回全国大会講演論文集，pp. 203-204 (1985).
8) 金田 泰：Pコードを中間コードとする Pascal 処理系の implementation，情報処理学会プログラミングシンポジウム報告集，pp. 143-152 (1980).
9) 中村敦司：Cross Stack Cache の実行過程のシミュレーション，テクニカルノート HLLA-141，筑波大学電子情報工学系 (1986).

(昭和60年12月12日受付)
(昭和61年6月18日採録)

板野　肯三（正会員）

昭和23年生．昭和46年東京大学理学部物理学科卒業．昭和48年同大学大学院修士課程修了．昭和51年同博士課程単位取得後退学．理学博士．筑波大学計算機センタ技官，同大学電子情報工学系助手を経て，現在，同講師．コンピュータアーキテクチャ，オペレーティングシステム，プログラミング言語に興味を持つ．ソフトウェア科学会，IEEE，ACM 各会員．

佐藤　豊（正会員）

昭和35年生．昭和57年筑波大学第三学群情報学類卒業．昭和59年同大学院修士課程理工学研究科修了．現在同大学院博士課程工学研究科に在学中．プログラミング・システムのユーザ・インタフェースおよび構成法の研究に従事．ソフトウェア科学会会員．

中村　敦司（学生会員）

昭和40年生．筑波大学第三学群情報学類在学中．現在，高級言語計算機の研究に従事．