計算機アーキテクチャ 89−24
（1991. 7. 19）

分散共有メモリ型並列計算機における『過去共有／未来多重』型データとキャッシング機構

鈴木敏浩　冨澤眞樹　五十嵐智　阿刀田央一
東京農工大学工学部数理情報工学科

現在の状態から到達可能な複数の状態を並列に生成し、その間で選択を行う形式で並列化できる問題がある。このとき未来の状態間は独立なので、現在の状態を表すデータは、それぞれの状態を生成するプロセスにコピーした後変更されなければならない。しかしコピーの際の、複数のプロセスからの同一データへのアクセスバーストは当然ボトルネックとなる。これを回避する戦略は、本当に必要なものだけをコピーすることと、コピーの時刻を分散させることである。本稿はこのためのハードウェア機構を、アドレシングモードと考えて、並列計算機のプロセッサエレメントに組み込む方法について論ずる。

THE PAST SHARING ／MULTIPLE FUTURE DATA AND CACHING MECHANISM ON A DISTRIBUTED SHARED MEMORY MULTIPROCESSOR

Toshihiro SUZUKI　Masaki TOMISAWA　Satoshi IGARASHI　Oichi ATODA

Faculty of Technology, Tokyo University of Agriculture and Technology
2-24-16 Naka-machi, Koganei-shi, Tokyo 184, Japan

In a parallel program in which multiple processes competes respective future states reachable from the present state, change of state parameters are made on respective copies of present parameters since all possible futures are independent. When many processor elements try to acquire copies of common data at a burst, performance degradation or even hot-spot may be caused. We introduce a cache scheme with a special addressing mode into respective processor elements to solve this difficulty by dispersing the occurence of common data access in time domain and by eliminating unused data accesses.

1．はじめに

共有メモリ型並列計算機において、深さ優先探索を幅優先探索にして並列化を行うとき、親プロセスから複数の子プロセスへ環境を継承するにはバスに重い負荷がかかる。子プロセス側では、他のプロセスからの非干渉性を得るために環境のデータは自分の私的領域にそれぞれ確保しなければならない。このとき必然的に、バス競合を招き性能低下の一因となりうる。

この問題を緩和する一つの方式として、子プロセス側で必要となったデータだけを、必要になった時点で親プロセスからコピーし、コピー時期を時間的に分散させることにより、バス負荷の軽減を図ることが考えられる。本稿ではそのようなデータ継承技法を実現するためのデータ構造とハードウェア機構について報告する。

2．過去共有／未来多重データ

図1のように親プロセスを過去、そこから派生する子プロセスを現在と考え、複数の子プロセスにコピーされていくデータ、およびその領域を、「過去共有データ」、「過去共有領域」と呼ぶことにする。親プロセスから派生した複数の子プロセスは、互いに異なる現在データを持っているが、その初期状態では過去のデータを共有している。また、親プロセスのデータを現在とみれば、子プロセスのもつデータは未来のデータであり、一つの現在から多重な未来が発散しているようにも考えられる。

そのような状況のデータは、探索問題などで頻繁に

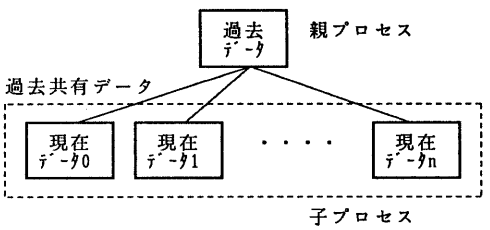

図1　過去から派生した現在

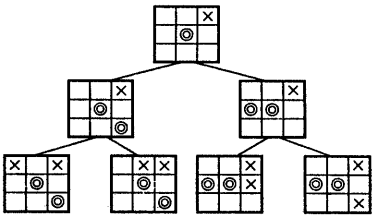

図2　盤面の並列探索

現れる。例えば、盤を使うゲーム等において次の手を、子プロセスを使って並列に検索する場合を考える。初期の盤面はすべてのプロセスで同一のデータをもつ。しかし各プロセスで独立に変更が加えられたとき、プロセスの持っているデータ同士は他に干渉してはならない（図2）。したがってこのようなデータは、親プロセスからそれぞれの子プロセスの私的領域にコピーされなければならない。

Prolog等の論理型言語においても、深さ優先探索を幅優先にしてOR並列処理を行う場合に同じ状況が発生する。深さ優先探索においては、処理に現れる変数に一意な環境を与えればよい。しかし幅優先探索で並列処理をするときは、多重な環境で束縛される変数が現れる。そのため環境は実行できるプロセスが生成されるたびに確保されなければならない。

このような過去共有データには、バス競合という大きな問題が含まれる。これはデータのすべてを親プロセスから、一度に複数の子プロセスへコピーするために起きる現象である。また、コピーされたデータがすべて参照されるとは限らないとき、すべてのデータをコピーするのは効率的なことではない。データのコピーは、その参照が行われたときに初めて行えば、参照されないデータがコピーされない分、プロセス当りのバスの占有率を抑えることができる。従って、コピー要求があったときに要求されたデータだけをコピーする方法によりバス競合を抑制することができる。

3．データ継承技法の概要

大量なデータの値渡しは、バスに大きな負荷を与える。しかし、プロセスごとに独立した環境をもつには

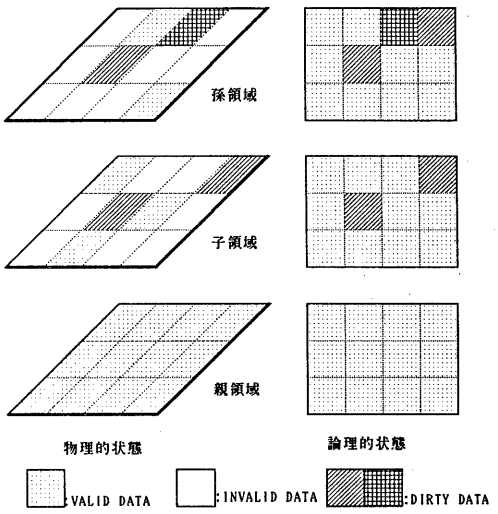

図3　データ継承状態

データは値渡しされなければならない。そのデータの値渡しを効率的に行うのが、本稿で述べる過去共有データの継承技法である。

親のプロセスが、データを共有する子プロセスをいくつか生成したとする。子プロセスは、親からデータを継承しなければならないが、データの継承は子プロセス側でデータに対するアクセスが発生しない限り行ない。初期状態において子プロセスには、対応するデータの領域だけを確保する。子プロセスがデータをアクセスして初めて、対象のデータを親プロセスの領域から子プロセスの領域にコピーする。このときにデータは物理的に確定する。

子プロセスが、データを親プロセスからすべて継承する前に、子プロセス（親プロセスからみれば孫プロセスに相当する）を生成する場合も有り得る。その場合には、孫プロセスが継承するデータが子プロセスに存在しない場合がある。該当のデータが存在しない場合には、親のプロセスに遡り、多段に継承を行わなければならない。

領域だけが確保されていて、実体の存在しないこの過去共有のデータ継承の様子は透明な板の上に色付きのタイルをはめ込んで行くことに例えられる。色付きのタイルは実体のあるデータ、透明な板は実体がなく領域だけを持っているデータである（図3）。

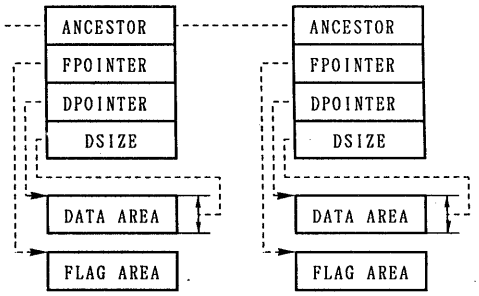

図4　四つ組の対応関係

さらに、呼び出されるプロセスの関係を木構造に例えると、最初のプロセスはルートプロセス、そこから生成されたプロセスはノードプロセスとみなすことができる。このときルートプロセスのデータはすべて確定している。つまりすべて色付きのタイルで埋められている。ノードプロセスは生成されたときすべての領域が透明な板で埋められているので、ルートプロセスの持っているタイルを自分の領域から透かしてみることができる。このため、物理的に実体が存在していなくても、論理的にはデータが確定しているように見える。

4．継承技法のデータ構造

あるデータ領域を過去共有領域として使用するために、次の四つのデータ（以下四つ組という）を管理構造としてもつ。それは、ANCESTOR, FPOINTER, DPOINTER、および DSIZEである。この四つ組データは過去共有を行うプロセスがそれぞれもつ。

ANCESTORは、親プロセスのもつ四つ組データへのポインタである。このポインタによって、過去共有領域を多段に遡ることができる。FPOINTERは、データが実際に継承されているか否かを示すフラグ領域へのポインタである。DPOINTERは、過去共有領域へのポインタである。DSIZE は過去共有領域の大きさを示す。これら四つ組の対応関係を図4に示す。

領域内のデータが実体であるか否かを示すためにフラグが必要になる。データ継承の処理はワード単位で行うため、1ワードごとに2ビットのフラグを割り付ける。次頁図5にフラグの状態遷移図を示す。

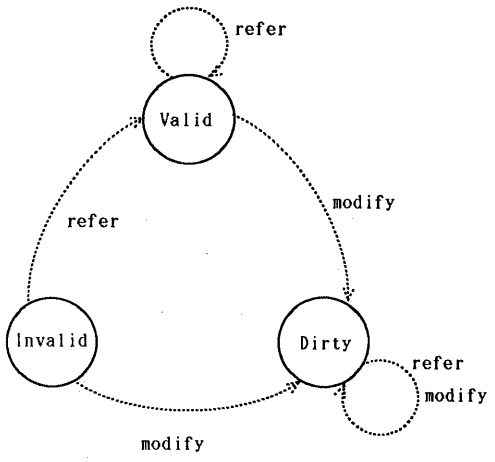

図5　フラグの状態遷移

VALID は自分の領域に物理的なデータが存在している状態、INVALIDはデータがない状態である。DIRTYは書込みが行われ、親のデータ、あるいは親が見ることのできるデータと異なる状態である。

プロセスの生成と共に、フラグ領域とデータ領域が取得される。領域が取得されるとき、フラグ領域はINVALID に初期化されるので、データはまだ親から継承されていない状態を示す。

5．参照手続き

孫プロセスからのデータ参照を四つの場合に分類し、図6を用いて説明する。

（1）　親プロセスにだけデータが存在する場合

①のデータを参照する。①が実体であるか確認するために、フラグ1を見る。フラグが INVALIDなので、子プロセスのフラグ1を見る。ここでも INVALIDなので親のフラグ1を見る。親のデータは VALIDなので、そのデータを参照し、孫の①に書き込む。孫のフラグ1をVALIDに変更する。

データの継承は、データ参照を行うプロセスが参照したデータについてだけ行うので、参照するために通過した子プロセス等、先祖のプロセスにデータを継承することはしない。

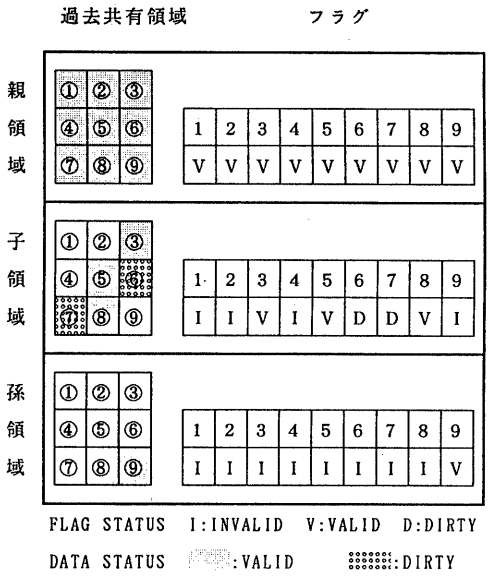

図6　データ継承手続き

（2）　子プロセスにデータが存在する場合

③のデータを参照する。まず、孫のフラグ3を見ると INVALIDである。そこで、子のフラグ3を見る。これは VALIDなので、ここからデータを参照し、孫の③に書き込んだ後、孫のフラグをVALIDに変更する。

（3）　子プロセスにデータの変更があった場合

これは（2）のケースと等しい。⑥のデータを参照する。孫のフラグ6は INVALIDなので、子のフラグを見る。ここはDIRTYである。DIRTYフラグは親データとは内容が異なるが、データは存在している状態なので、このデータを参照する。孫の⑥にデータを書き込んだ後、孫のフラグ6を VALIDに変更する。DIRTY ではなくVALID にするのは、自分の親と同じデータを持っているからである。

（4）　孫プロセスにデータが存在する場合

⑨のデータを参照する。孫フラグ9は、VALID を示している。それ故、⑨はデータが存在していることになり、このデータを参照する

```
struct  tetra
{
    struct tetra *ancestor;
    int          *dpointer;
    char         *fpointer;
    int           dsize;
};

int    offset ;          /*  given from cpu */
struct tetra *current;   /*  given from cpu */
int    fetchdata;        /*  fetch data     */
int    mask;
struct tetra *adrlatch;      /*  work    */

    adrlatch = current;

    mask = フラグビットの抽出;

while( !(*(adrlatch->fpointer
          + (offset >> 3)) & mask ))
       adrlatch = adrlatch->ancestor;

fetchdata =  *(adrlatch->dpointer + offset);

        /* set flag & store data */

*(current->fpointer + (offset >> 3)) |= mask;

*(current->dpointer + offset) = fetchdata;
```

図７　機能の記述

以上の参照手順を言語Ｃで記述したものを図７に示す。

過去共有の四つ組を構造体で表し、while 文でフラグを見ながら先祖をたどっていく動作を記述している。構造体 currentは自分、つまり現在データの四つ組を格納し、構造体 adrlatch は先祖をたどるために使用する。while 文の中では、参照するデータに対応するフラグを検出しながら有効データを検索している。

６．継承機構の実現

データ継承の処理は、ソフトウェアでもシミュレート可能である。しかし、過去共有は、本来データ参照の一方式であるため、一種のアドレッシングモードとして考えられる。そのためライブラリ等、ソフトウェアの形で存在するのではなく、CPU 内部に取り込まれるべきである。本研究では、このアーキテクチャを確認するために CPUの外側にハードウェアによる試作を行った。

データ継承機構は、MC68000をCPUにもつ各プロセッサ要素に直接接続される（図８）。MC68000 ではメモリアクセスをハンドシェイクで行うため、メモリから

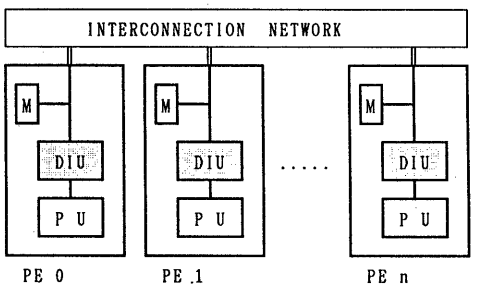

図８　データ継承機構の物理的配置

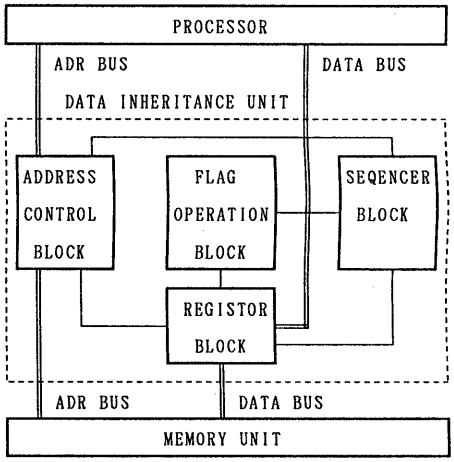

図９　継承機構ブロック図

のアクノリッジ信号が返るまでメモリサイクルは終了しない。データ継承処理は、バスに過去共有領域へのアクセスが現れたとき、CPU をウェイト状態にしたまま継承機構が行う。そのため CPUは、継承の処理を意識せずにデータを使用することができる。アドレッシングモードの一つとして振舞うこの継承機構は、本来プロセッサ内部に取り込まれるべきである。

図９に継承機構のブロック図を示し、各部の動作について述べる。回路は大きく分けて４つのブロックからなる。すなわちレジスタ部、フラグ操作部、アドレス制御部、シーケンサ部である。

アドレス制御部には、過去共有領域へのアクセスを

検出するための機能があり、ここで検出された信号によって継承動作が起動する。また継承動作中、プロセッサは対象アドレスからのデータを待っている状態なので、その間、先祖の四つ組の読み出し等継承動作に必要なアドレスはここで生成される。

レジスタ部は、過去共有に必要な四つ組データを格納しておくレジスタ群からなる。この四つ組のレジスタは、現在データの四つ組を格納するためのカレントレジスタと、先祖を遡るために先祖データの四つ組を格納するワーク用のアドレスラッチレジスタの二つに分類される。カレントレジスタはI/O レジスタとしてプロセッサから見ることができ、メモリ上に連続して割り付けられる。

フラグ操作部は、参照するデータに対応するフラグの位置の計算や、Dirty、Validなどのフラグを立てる作業をする。該当するフラグは、DPOINTERと参照しているデータのアドレスの差、つまりオフセットの値から抽出される。

図１０に各ブロックの内部構造を示す。各ブロックはラッチ、バッファ等から構成されており、これらはすべてシーケンサによって制御されている。シーケンサはマイクロプログラミング方式を採用しており、プログラムはアクセスタイム45nsのSRAMに格納されている。

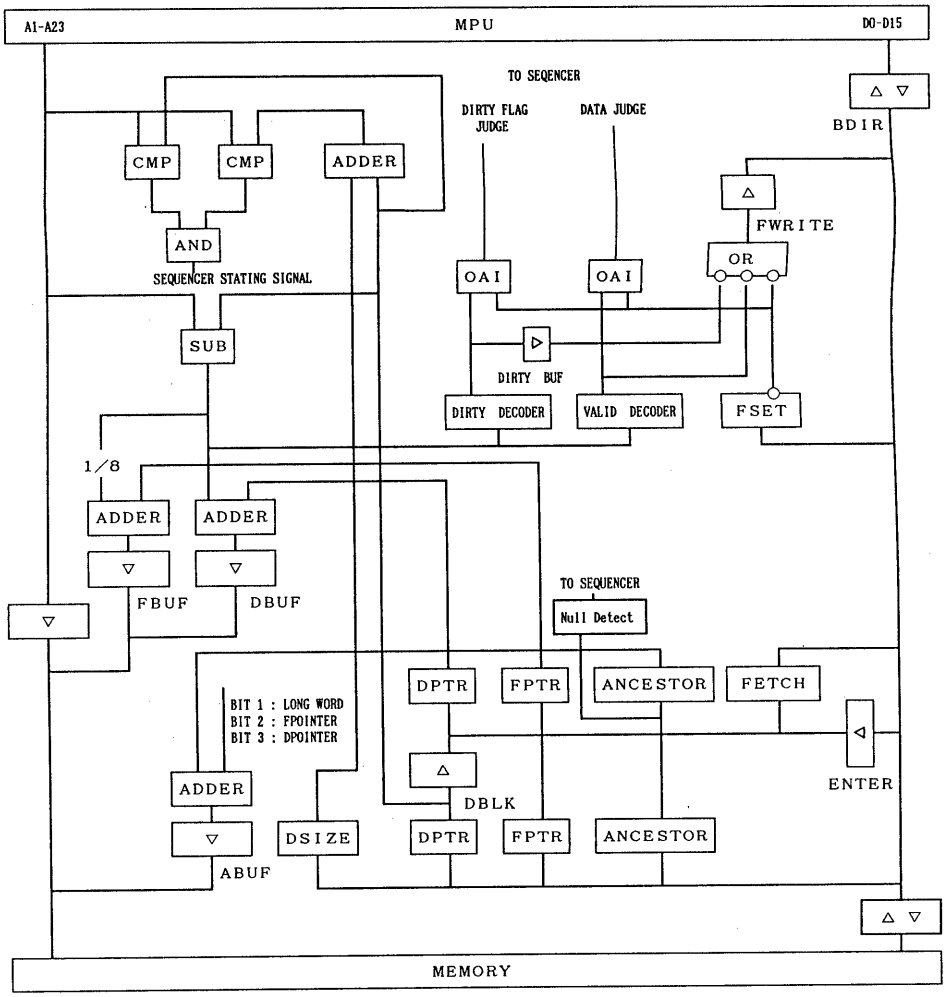

図１０　各ブロックの内部構造

7．評価

この継承技法は、参照しないデータを継承せずプロセス間のデータ継承を時間的に分散させるため、バス競合の抑制に有効である。この継承機構の効率は次のような要因によって影響される。

1） 先祖のプロセスの数

2） 兄弟のプロセスの数

3） 継承したデータの使用頻度

4） 過去共有領域内で継承されるデータの割合

兄弟のプロセスとは、同じプロセスから生成された子プロセス同士のことである。上記の要因は、実行するアプリケーションによって変化する。もっとも効率的な使用法は、過去共有データはすべて継承する可能性があるが、どのデータをいつ継承するかを確定できず、さらに一度継承したデータは何度も使用する、という場合である。

データ継承に必要となるクロック数は継承するデータが、どこに存在しているかということで、変化する。表1に継承に必要なマイクロプログラムの動的ステップ数を示す。

表1 継承の動的ステップ

動作	ﾌﾗｸﾞ状態	データの所在	動的ｽﾃｯﾌﾟ数	メモリアクセス数(ﾘｰﾄﾞ/ﾗｲﾄ)	所要ｸﾛｯｸ数
読み出し	I	一親等	34	11(9/2)	100(77)
	I	二親等	53	16(14/2)	149(112)
	V	自分	8	2(2/0)	20(14)
	D	自分	8	2(2/0)	20(14)
書き込み	I	一親等	33	10(7/3)	93(70)
	I	二親等	53	15(12/3)	143(105)
	V	自分	11	3(1/2)	29(21)
	D	自分	11	3(1/2)	29(21)

マイクロプログラムの動的ステップは、1ステップが1クロックに対応している。メモリアクセスは1回で7クロック必要である。以上からデータ継承に必要なクロック数は次のように求められる。

CLOCK = (MICRO STEP - MEMORY ACCESS)
+ MEMORY ACCESS * 7

この技法をハードウェア化する前にソフトウェアによるシミュレートを行った結果、実行クロック数は、ハードウェア化によっておよそ10分の1程度に削減される。ハードウェアはTTL 約150個で構成されており、将来的には十分CPU 内部に取り込むことが可能な規模である。

並列の場合ではなく、プロセッサが一台で干渉の発生しない状態において継承機構を使用した場合と、使用せずに領域内のデータをすべてコピーした場合の継承時間を考える。このときMC68000 のメモリ間転送のクロック数は1ワードあたり20クロックとして扱った。継承機構を使用しない状態では、プロセスが生成されるごとにすべてのデータをコピーしていくため、コピーに必要な時間は単純に増加していく。子プロセスでデータを使用しない場合は、孫プロセスにすべてコピーするので無駄なコピーが行われる。継承機構を使わずに、すべてのデータを子プロセスにコピーする時間は、継承機構を使用して約20％のデータを継承する時間と等しい。孫プロセスへのコピーの場合は、継承機構を使ってすべて二親等の親プロセスから継承した場合、約27％のデータを継承した時間が、継承機構を使わずにすべてコピーした時間に等しい。

しかし、この比較はあくまでプロセス間の干渉を考慮しない場合なので、実際には継承時期の分散がある分、継承機構を使用した方が効率的である。

メモリへの競合は違うアドレスへのアクセスであっても、同じメモリモジュールに対するアクセスであれば発生する。継承機構を用いずに、領域内のブロック全体をコピーした場合、メモリモジュールは特定の時間占有されることになる。継承機構を用いた場合には、継承するデータの単位は1ワードごとになるため、メモリモジュールを占有している時間はブロックコピーの場合より短い。メモリアクセスが行われている時間が連続ではなく、細かく分散されることで競合の発生

する可能性はかなり低くなるものと考えられる。

継承処理において、先祖をたどる際のフラグ参照はボトルネックとなる可能性がある。しかしこれは、プロセッサ内部にこの技法を取り込み、キャッシュと併用することで緩和することができる。すなわち、フラグの参照は一度行われるとキャッシングされるので、先祖のフラグを参照するためのメモリへの直接のアクセスは減少していく。現在の試作段階においても、全体の５分の１のデータだけを使用する場合には、継承機構を利用した方が有利な結果が得られることはすでに述べた。さらに、これをアドレッシング機構としてプロセッサ内部に取り込むことで性能の向上が見込まれる。このとき、キャッシュコントローラには変更が必要になる。つまり、アクセス領域が過去共有領域に行われる場合には、継承動作を行うためにフラグの状態により動作が変化しなければならないのである。

８．まとめ

本稿では、多重環境が必要とされる処理において、複数のプロセスに値渡しされる共有データを過去共有データと呼び、コピー時におけるバス競合を緩和する方式を提案し、ハードウェアによる支援機構について報告した。

Prolog等の論理型言語においてOR並列処理を行うには、環境を多重化する必要がある。環境生成のために多重な環境で束縛される変数をコピーする際のバス競合は不可避である。本稿で述べた方式により、データコピーの時間が分散されると共に、必要のないデータのコピーは行われないので、その効果は大きいものと考えられる。

試作したハードウェアによる過去共有データの継承機構は、規模的に十分プロセッサ内部に取り込むことが可能であり、プロセッサのアドレッシングモードの一部としてサポートされることが望ましいと考える。

今後、この継承機構を並列計算機上に実装し、データ継承技法のバス競合抑制効果を評価して行きたい。

謝辞

本研究を進めるにあたり、有益なご助言をいただいた本学斎藤延男教授に深く感謝する。また、方式の設計と試作に協力をいただいた本学修士課程の星野浩志氏、田村仁氏、および北野博氏（現ニコン㈱）に深謝する。

参考文献

［１］中村克彦：“Prolog処理系”，情報処理，Vol.25，No.12，pp.1329-1335(1884).

［２］田中英彦：“非ノイマン型コンピュータ”，電子情報通信学会，1990

［３］星野 他：“密結合並列計算機における「過去共有データ空間」とそのアドレッシングモードによる実現”，情報処理学会　第４２回全国大会(1991)

［４］冨澤 他：“手続きフロー型並列計算機の制御機構と並列記述言語”，情報処理学会計算機アーキテクチャ研究会資料，79-15(1989).

［５］冨澤 他：“手続きフロー型並列計算機における分散型組み込み制御機構”，信学論(D)，J71-D，10，pp.1921-1930(1988).