計算機アーキテクチャ　89－18
（１９９１．　７．　１９）

発注／受領型並列計算機の例外処理とデバッグ支援機構

田村　仁　　富澤眞樹

東京農工大学　工学部

我々は並列手続き呼出しを計算モデルとした並列計算機の分散制御機構を提案している。これをハードウェアに組み込んだＰＥの通常状態では、ユーザの手続きが論理的に多重環境下で並行動作する。しかし例外処理は物理的に１つの環境で実行されなければならない。このために別の動作状態を設定する。この動作状態モデルと整合する例外処理機構を構築した。並列計算機の例外を例外発生ＰＥだけで処理でき、他ＰＥに影響しない『解析例外』と、全ＰＥを凍結し並列計算機全体で処理する『凍結例外』に分ける。そしてこれらを任意に起こして並列計算機用ブレークポイントとして利用したり、イベントヒストリーを取得するデバッグ支援機構を構築した。

Exceptions Handling and Debugging Support Mechanism for the Demand/Accept Multiprocessor

Hitoshi TAMURA　Masaki TOMISAWA

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

We have proposed a distributed control mechanism based parallel procedure call for a multiprocessor. This mechanism generates multiple-environments for procedures in a PE, and executes cocurrently. But exceptions must be handled on a static environment. So, we make a "singular" mode of a PE. We design a exceptions handring mechanism matched this mode. We divide exceptions into "analisys" and "freeze". "Analisys" exception does'nt affect other PEs, and a PE caused the exception handles it. "Freeze" exceptions suspends all PEs, and a kernel PE handles it. We design a debugging mechanism used these exceptions as proves and breakpoints for the multiprocessor.

1．はじめに

我々は、分散共有メモリ型並列計算機の各プロセッサエレメント（以下、ＰＥ）に組み込むための制御機構[1]及び言語Ｃを拡張した並列記述言語Ｃ//[2]を提案している。その制御機構は、並列手続き呼出しの命令とインスタンスの実行管理をハードウェアとして提供する。この制御機構を持つ並列計算機を発注／受領型並列計算機と呼んでいる。本稿では、発注／受領型並列計算機の例外機構と、Ｃ//で記述された並列プログラムを発注／受領型並列計算機でのデバッグするための支援機構について報告する。

2．発注／受領型並列計算機の概要

Ｃ//では関数呼出しを拡張し、図１のように並列呼出しを記述する。まず演算子『//=』を用いて関数を並列に呼び出し（発注）、『//』によって返り値を受け取る（受領）。この発注と受領の間を『Transfer変数』（図１の t1,t2,t）によって対応付ける。

Ｃ//の制御フローは図２に示すように、並列呼出しによって関数（以後テンプレート）のインスタンスを動的に生成し、返り値を受け取る時に同期が取られる。インスタンスは実行終了と同時に消滅する。１つの関数に対してインスタンスは複数生成されてよく、テンプレートを共有するインスタンスは並行実行される。

実際に発注／受領型並列計算機では図３に示すように、各ＰＥは分散された共有メモリに割り当てられたテンプレートだけを実行する。すなわち、ＰＥ＃１は、

```
void main()
{ int i,j, //t1,//t2, funca(int);
        ...
  t1 //= funca(512);
  t2 //= funca(1024);
        ...
  i = //t1;
  j = //t2;
        ...
}

int funca(int p)
{ int i;
  int //t, funcb(int);
        ...
  t //= funcb(p+i);
        ...
   return i + //t;
}

int funcb(int p)
{ int ret;
        ...
  return ret;
}
```

図１　Ｃ//プログラム例

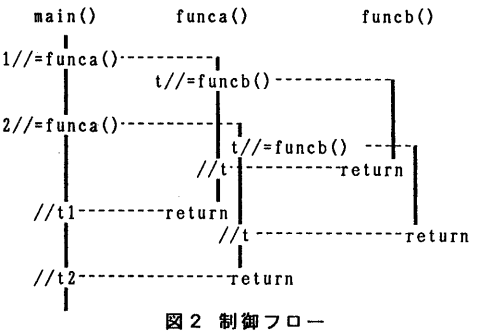

図２　制御フロー

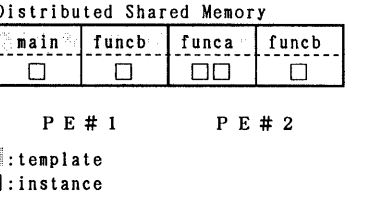

図３　テンプレートとインスタンスの空間配置

main と funcb だけのインスタンスを生成し並行実行する。main のインスタンスが、ＰＥ＃２の funca を呼び出すことによって、ＰＥ＃１とＰＥ＃２とで図１のプログラムが実際に並列実行される。一つのＰＥ内でのインスタンスの実行管理と並列呼出しは、３．で述べるようにハードウェアとして提供されている。

各ＰＥの通常の実行状態は、インスタンスの実行管理はハードウェア化されているので、複数のインスタンスを並行実行している状態である。この状態は、プロセッサを多重化している状態と言える。一方、例外処理など物理的なハードウェアに依存する処理は、プロセッサを多重化しない状態で逐次的に実行しなければならない。このようにＰＥは、プロセッサを多重化する状態としない状態を持つ。２つの実行状態を基本とした保護機構については４．で述べる。

Ｃ//のプログラムは、main のインスタンスから並列関数呼出しによって拡散的に各ＰＥにインスタンスが生成されゆく。このような実行環境下では、ＰＥで発生する例外やインスタンスのデバッグも並列計算機全体で処理する必要がある。すなわち、例外処理やデバッグも各ＰＥで協調して動作する必要があり、そのための例外機構については５．で述べる。

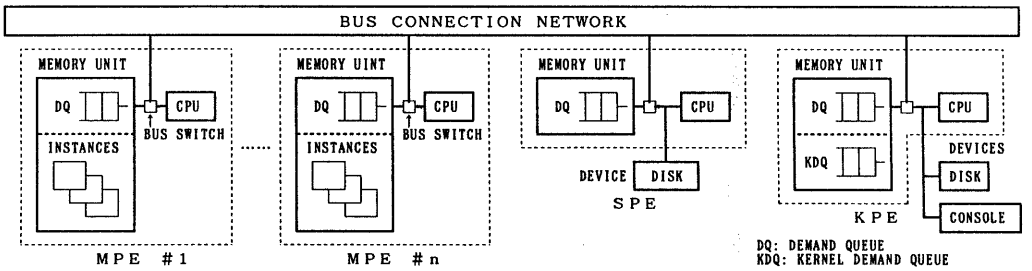

図４　並列計算機の構成

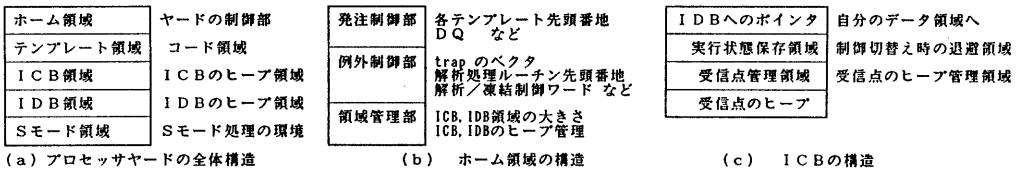

図５　プロセッサヤードの構造

３．発注／受領型並列計算機の並列リンク機構

図４に示すように発注／受領型並列計算機は分散共有メモリを持つ。この各ＰＥに分散された共有メモリ上のそれぞれに、プロセッサヤードと呼ぶＰＥ固有の制御構造をおく（図５）。この構造はホーム領域、テンプレート領域、ＩＣＢ（Instance Control Block）領域、ＩＤＢ(Instance Data Block) 領域およびＳモード領域から構成される。

ホーム領域はＰＥの制御部であり、例外処理、発注制御、および領域管理用の領域である（図５ｂ）。テンプレート領域は各ＰＥのコード領域である。ＩＣＢ領域とＩＤＢ領域はそれぞれインスタンスの制御部とデータ部のためのヒープ領域である。ＩＣＢとＩＤＢは、インスタンスの生成時に環境として動的に割り当てられる。そしてインスタンスとは別に『Ｓモード』の処理（４．）のための静的な環境を用意しておく。

ホーム領域中の発注制御部にＤＱ(Demand Queue)を設定する。ＤＱは他ＰＥからの発注に使われ、ＤＱには受信点(Recipient) のアドレスが発注として積まれる。受信点は発注するテンプレートの識別子とそれへの引き数を格納した領域のアドレス、およびそのインスタンスの状態フラグからなる。この受信点を介して発注／受領の同期を行う。受信点はＩＣＢ内にあるヒープ領域から動的に確保される（図５ｃ）。

この各ＰＥで構築された構造を前提に、２．で述べた制御モデルを基本的な３つの並列リンク命令 DMND、ACPT および DLVR によって実現する。

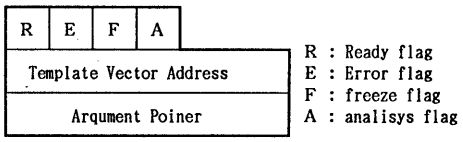

図６　受信点(Recipient)

表１　並列リンク命令

命令	ニモニック	機能
Demand	DMND	発注
Accept	ACPT	発注結果の受領
Deliver	DLVR	結果納入とインスタンス消滅
Select	SLCT	複数の発注の中から受領するものを非決定的に選択
Test & Set with Switch	TASS	失敗の場合は制御を切り替える TAS 命令
Retract	RTCT	発注取り消し

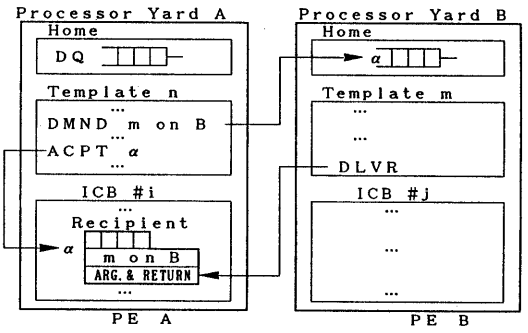

図７　並列リンク

まず DMND 命令で発注する。この命令では受信点を確保して値を初期化し、発注先のＤＱに受信点のアドレスを積む。各ＰＥはＤＱに受信点アドレスが積まれると、ＩＣＢとＩＤＢ両領域から領域を取得して環境を設定し、受信点に示されたテンプレートの実行を開始する。インスタンスは DLVR 命令によって実行終了する。この命令で引き数領域に結果を返し、ＤＱから自分に対応する受信点アドレスを取り除き、環境を解放して消滅する。そして発注側で ACPT 命令により受領する。 DMND 命令で生成された受信点のフラグを参照し、インスタンスが終了していれば引き数領域から実行結果を読み出し、受信点を解放する。 TASS 命令は、単純なテスト＆セット命令にインスタンスの制御を切り替える機能を付加したものである。その他の命令は本稿では触れない。

発注／受領型並列計算機では ACPT 命令で発注先のインスタンスがまだ実行中であった時や、 TASS 命令でテストに失敗した時など同期待ちになると、インスタンスの制御を自動的に切り替える。ＤＱから新たに実行可能な発注のインスタンスを生成したり、サスペンドしていたインスタンスに制御を戻す。

以上のように発注／受領型並列計算機では、各ＰＥに分散して組み込まれたた制御機構によって、手続きのインスタンスを動的に生成する並列呼出し型の制御フローを実現している。

4．発注／受領型並列計算機のモード

4．1　ＰＥの動作モード

発注／受領型計算機ではユーザの手続きに対する制御は多重化される。3．で述べたようにＰＥは通常多重環境を持ち、手続きに対する複数のインスタンスを並行実行している。言い替えればインスタンスに対して個々に論理的なＰＥが与えられ、実行が行われる。

しかしエラー処理のようにただ一つの物理的なＰＥに依存したり、または入出力処理など物理的なデバイスに依存しなければならない処理がある。この処理では制御が途中で勝手に切り替えられ、他の関係する制御に追い抜かれては困る。

この論理的に多重なＰＥの状態と、物理的なイメージに依存したＰＥの状態の境界は明確に区分すべきである。そこで前者の動作状態をM(Multiplex) モード、後者をS(Singular)モードと呼ぶことにする。

さらにこの他に並列計算機全体でただ1箇所に存在し、系全体を監視するモードが必要になる。これをK(Kernel)モードとする。

発注／受領型並列計算機の通常のＰＥは、メンバＰＥ（ＭＰＥ）と呼ばれ、ユーザのインスタンスをＭモードで処理するが、例外の発生などの要因でＭモードからＳモードに遷移する。

Ｋモードで動作できるＰＥはハード的に特別な機能（凍結用割込み元回路等）を持つカーネルＰＥ（ＫＰＥ）1つだけである。このＫＰＥには計算機の立ち上げ用に入出力装置が複数接続される。この管理のためＫＰＥにはＳモードも必要となる。ＫＰＥは通常はＳモードでユーザからの入出力要求を処理し、系の中で例外が発生した場合にＫモードに遷移する。

また入出力専用ＰＥや特殊なハードウェアを持つＰＥが含まれる場合、それらを専用処理ＰＥ（ＳＰＥ）と呼び、Ｓモード専用とする。

4．2　特権レベルの設定

4．1で述べたモードの性格から発注／受領型並列計算機の特権レベルは次のように設定できる。並列計算機の系全体を監視するＫモードに対して最上位の特権レベルを与え、入出力などデバイスを扱うＳモードはＰＥ内での特権を与える。そしてユーザの手続きを実行するＭモードの順に特権レベルを階層化する。

さらに1つのＰＥに複数のデバイスが接続され、それを扱う複数のＳモード環境が存在する場合、それらを管理するために、より上位のＳモードが必要になるかもしれない。一般的にはさらに特権レベルの細分化が考えられるが、必要最小限としてはＫ、Ｓ、Ｍモードの境界上に設定すればよい。

単純なメモリ保護の例として図8のように設定できる。ＳモードまたはＫモードからアクセスできる空間にホーム領域など制御構造おくことで、通常Ｍモードで動作する他ＰＥから制御構造を保護する。

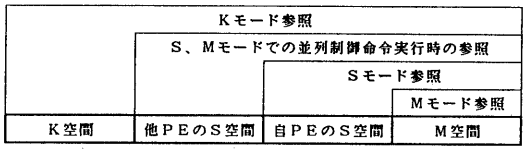

図８　並列計算機のメモリ保護

4．3　モード間の交信・遷移

ＭＰＥではＳモード管理下で複数のＭモードインスタンスが多重に並行動作している。そして他ＰＥ上のＳモード、ＭモードインスタンスおよびＫモードで１つの並列プログラムの処理を実行する。これら１つのプログラムを処理するために構成されたＭモードインスタンス間では空間は共有され、ＳモードとＫモードの処理からもアクセス可能でなければならない。

ＰＥがＭモードの場合、発注によって動的に環境が割り当てられ、インスタンスが生成される。ＰＥがＳモードの場合は、インスタンスを生成せずに自分の環境を用いて処理を行う。Ｓモードの環境は静的に存在し、 DLVR 命令を実行しても消滅せず、ＤＱに積まれた次の処理に制御が移る。５．で述べる例外処理の場合もＳモードの環境下で処理され、RTE 命令で例外処理から復帰するときも環境は消滅しない。

Ｓモードでは同期待ちが生じても制御を自動的に切り替えない。このことによりＳモードのＰＥに対して発注された処理は、ＤＱに積まれた順に処理が進むことを保証する。見かけ上は、Ｓモード上の処理は物理的なＰＥやデバイスに張り付く静的なプロセス（ただし空間は共有する）のように振る舞う。

図９にモード間の交信、遷移図を示す。Ｍモードインスタンスに対しては呼出しによる動的な生成消滅だけを用意し、Ｍモードインスタンス間の交信は制御機構としてサポートしない。Ｓモード、Ｋモードの処理に対しては、 DMND 命令による発注動作が疑似的に通信に対応する。ＭモードからＳＰＥに対しては特定形式の DMND を行うことができる（②）。Ｓモードでは同期待ちが起こっても制御が切り替わらず、飢餓状態に陥る危険性をもっている。これを避けるため、自分より上位の特権レベルを持つＳモードへしか発注できないように DMND 命令の方向性を限定する。

モード間遷移は例外によって行われる。並列計算機における例外には当該ＰＥだけで処理される『解析例外』と全ＰＥを凍結する『凍結例外』の２種類がある。これらは５．で説明する。

ユーザ手続きのインスタンスをＭモードで動作させていたＭＰＥは、各種のエラーによる例外でＳモードに遷移する（④）。解析例外ではＳモードに入り、ホーム領域のベクタで示されたエラー処理手続きを行う。一方凍結例外では、ＳモードになったＰＥが直ちにＫＰＥに対して暗黙的な DMND を行う（③）。ＫＰＥはこの時ＳモードからＫモードに遷移する（⑤）。

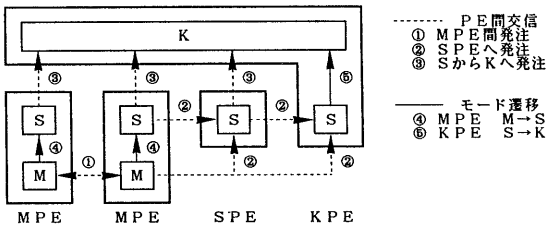

図９　モード間遷移・交信

5．例外処理機構

5．1　例外処理機構の概要

並列計算機の例外は、バスエラーやアドレスエラーのような並列計算機全体の例外として処理すべきものと、 TRAP 命令などのＰＥ単体の例外として処理できるものの２つに分けて考える。前者を凍結例外と呼び、Ｋモードで動作するＫＰＥを除く全ＰＥを凍結して処理する。後者を解析例外と呼ぶ。これら２つとリセットに相当する立ち上げ処理を含めて、３つの例外処理がある。

解析例外に関しては、各ＰＥが従来と同様の例外処理機構を持てばよいが、凍結例外と立ち上げ処理にはＰＥ間を結ぶ割込み機構が必要になる。

凍結例外が発生した場合、デバッグに用いるためにも、全ＰＥに速やかに例外の発生を伝達する必要がある。このためＭＰＥ、ＳＰＥおよびＫＰＥの間には、メモリバスとは別の凍結用割込みバスが必要である。ＭＰＥまたはＳＰＥのどれか１つ以上が凍結例外を起こすと、オープンコレクタバスを通してＫＰＥに伝達され、凍結フリップフロップ群がすべてセットされる。１つ１つの凍結フリップフロップはそれぞれのＭＰＥまたはＳＰＥに専用線を通して割込みを発生し、ＫＰＥを除く全ＰＥを凍結状態にする。一方凍結の解除は、ＫＰＥがＫモードのプログラムで、個々の凍結フリップフロップを個別に解除することにより行われる。

立ち上げ処理にも一斉に割込みがかかるようにしておく。次節で説明するＫＰＥによる初期化処理の終了時にＫＰＥが一斉にＭＰＥ、ＳＰＥに立ち上げ割込みをかける。

５．２　例外処理手続き

①凍結例外処理　致命的な例外がＰＥのどれかで生じると、凍結割込みを生じ、全ＭＰＥを凍結する。

例外が発生したＭＰＥは、Ｓモードに遷移し、すぐに凍結割込みを発生する。その後ＫＰＥからも参照できるようにＳモードの環境内のスタックに実行情報を吐出し、ＫＰＥへ暗黙的に DMNDを行う。

凍結割込みをうけＫＰＥはＫモードに遷移し、例外発生ＰＥからの暗黙的な発注を受ける。ＭＰＥには致命的な例外用のベクタは存在せず、代わりにＫＰＥが受け取った発注から、例外の発生したＰＥと例外の要因を知り、それに対応する処理を走らせる。

その他のＰＥは凍結割込みによってＳモードに遷移し、例外発生ＰＥと同様に実行状態を吐き出して割込み解除待ちにはいる。

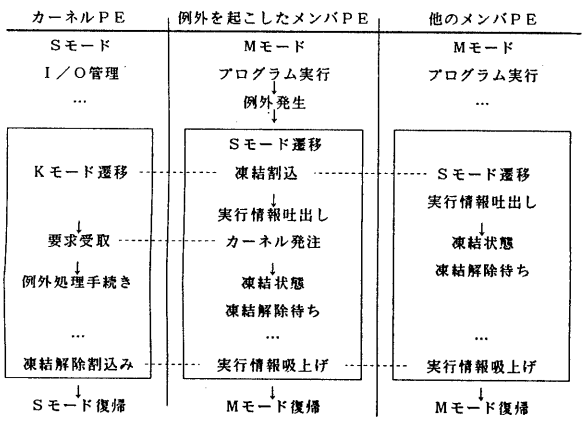

図１０　凍結割込み時の状態遷移

②解析例外処理　AOF 命令、TRAP 命令によって例外が生じると、該当ＭＰＥがＳモードに遷移し、ホーム領域の例外ベクタで指定された処理手続きを実行する。凍結例外とは異なり他ＰＥには影響しない。この解析例外は逐次計算機の割込みとほとんど同様の処理を行えばよい。ＰＥの資源を利用したり、ＰＥに実行状態のデータを取得できる。

③立ち上げ処理　ＫＰＥは電源投入後、従来の逐次計算機と同様にブートし、Ｋモードで立ち上がる。この状態で複数のＭＰＥ、ＳＰＥの立ち上げを管理する。その他のＰＥは電源投入後、そのままＫＰＥからの立ち上げ割込み待ち状態に入る。

ＫＰＥは各ＰＥのホーム領域中の領域のアドレスや大きさなどの各定数の初期化や、プログラムのロードを行う。そして全ＭＰＥ、ＳＰＥに立ち上げ割込みをかけ、各ＰＥの立ち上げ手続きを実行させる。

各ＰＥの立ち上げ手続きはＫＰＥに設定されたホーム領域の定数に従い、領域の取得や初期化、ＤＱの初期化など、自分のプロセッサヤードを構築する。ヤードの構築を終えるとＤＱの監視に制御を移し、他ＰＥからの発注待ちに入る。

ＫＰＥも自領域のヤードを構築し、立ち上げの最後に、任意のＭＰＥへプログラム最初の発注を行う。ユーザプログラム実行時にはＫＰＥはＳモードに遷移し、ＳＰＥと同様に入出力要求待ちに入る。

６．デバッグ支援機構

デバッグ支援機構として、ソフトウェア的に例外を発生させる命令とそれを支援するハードウェア機構を用意した。各ホーム領域中の例外制御部に解析制御ワード、凍結制御ワードを持たせている。この両制御ワードの状態によって並列リンク命令実行時に凍結／解析例外を引き起こす。またデバッグでの利用を目的とした AOF 命令も両制御ワードの状態によって、解析、凍結例外の両方を引き起こす。

表２　デバッグ用命令

命令	ニモニック	機能
Analisys Or Freeze	AOF	並列ブレークポイント
Freeze	FRZE	凍結
trap	TRAP	ソフトウェア例外
Demand cause Analisys	ADMND	発注先で解析例外
Demand cause Freeze	FDMND	発注先で凍結例外

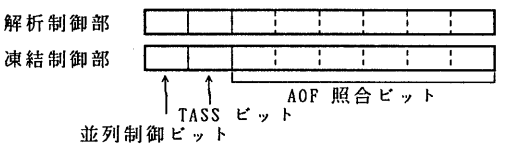

図１１　解析／凍結制御ワード

解析制御ワードは図に示したとおり並列制御ビット、TASS ビット、AOF 照合ビットから構成される。並列制御ビットにビットが立っていると並列リンク命令ごとに解析例外を引き起こし、TASS ビットが立っている場合は TASS 命令毎に解析例外を引き起こす。これ

らは従来のトレース例外に対応する。

TASS 命令を特別にした理由は、共有変数へのアクセス動作だけをブレークするためである。この機能を用意することで、プログラマが任意の共有変数へのアクセスだけをトレースできる。

また、AOF 照合ビットはブレークポイント命令である AOF 命令実行時に参照される。AOF 命令の即値オペランドと、AOF 照合ビットの論理和がとられ、もし結果が0でなければそれぞれ解析／凍結例外を引き起こす。これから、AOF 照合ビットのそれぞれに対応して、系列の違う並列ブレークポイントをビット数個分用意できる。

なおこの機構によって解析と凍結が両方発生し得る場合は、凍結例外が優先される。凍結例外が生じた場合、ＫＰＥへの暗黙デマンドの引き数として、例外の要因すなわち凍結制御ワード中のどのビットで凍結例外が生じたかを示すコードなどを与える。

この解析／凍結制御ワードは従来の計算機の状態語の拡張であり、並列リンク命令の度に参照される。実行中の参照は、ＰＥの立ち上げ時にＫＰＥから与えられた値で初期化される特定レジスタから行うか、ホーム領域においたまま行うかは選択の問題になる。レジスタにのせると実行中らＫＰＥから制御できず、メモリにおくとアクセスの負荷が大きい。ホーム領域をキャッシュできれば負荷は減少するが、キャッシュの一貫性管理の負荷ができる。またレジスタにのせたままでも割込みを使用してＫＰＥから制御できるが、ＰＥ別の割込みが必要で、ＰＥ数が多いと実現が面倒になる。

解析制御ワードの状態に関わらず解析状態に移行する命令として TRAP 命令を用意する。これは従来のソフトウェア例外に対応し、独自のベクタを参照して解析状態に移行する。これをＰＥ資源の要求に用いることができる。

また凍結制御ワードの状態に関わらず凍結例外を引き起こす命令として FRZE 命令を用意する。この命令の即値オペランドはＫＰＥへの DMND の引き数となる。この命令は並列プログラムの一番最後で、プログラムの終了命令として使用される。すなわちこの命令のオペランドがプログラムの終了コードになる。

このほかに発注先で例外を引き起こす命令として、ADMND 、FDMND 命令を用意する。これは発注先がインスタンス生成直後にそれぞれ凍結、解析例外を引き起こす。これはデバッグ用に用意された命令だが、この命令を特権命令とするかユーザに解放するのかは検討の余地がある。ユーザが使えれば、インスタンス内のユーザ定義の環境初期化に用いることができよう。

解析、凍結例外に移行するタイミングは次表に示すとおりである。

表３　例外状態への移行時期

命令	移行時期
DMND	受信点確保後、発注をＤＱに積む前
TASS	命令実行後（ TASS 成功後）
AOF, FRZE, TRAP	命令実行後
FDMND, ADMND	インスタンス生成直後、発注先で例外
その他	フェッチ直後、実行前

DMND が受信点確保後に例外を起こすのは勝手にユーザが受信点を用意できないためで、 TASS の場合は履歴をとる都合を考えたためである。

7: デバッグ支援機構を用いたデバッギング

並列プログラムのデバッグでは並列プログラムにデバッグ用の処理（プローブ）を付け加えることで、プログラムの振舞いが変化してしまうプローブ効果が問題となる。無論、プログラムの逐次部分のデバッグには従来の手法も有効である。問題は並列特有の共有資源に対しての競合にかかわるバグで、ブレークポイントを用いた従来の反復手法ではデバッグできない。並列計算機のデバッガではこれに対処する機構を提供しなければならない。

プローブ効果を避ける１つの明快な方法は、ブレークポイントで同時に全ＭＰＥを凍結し、その間にＫＰＥがデバッグ処理を行うことである。発注／受領型並列計算機は５．で述べたように凍結例外が生じた瞬間に、厳密には数命令のずれの範囲で全ＭＰＥを停止できる。そのずれに影響されるバグも考えられるが、発注／受領型並列計算機が制御する中粒度のインスタンスに対しては十分小さいと考えることができる。

デバッグ支援用の機構として用意した解析／凍結制御ワード、AOF 命令などを利用して、特定の並列制御命令ごとやプログラム中任意の場所で凍結および解析例外を引き起すことができる。これから凍結例外を発注／受領型並列計算機用のブレークポイントとして使

用でき、従来の繰返し式のデバッグ手法も有効に利用できる。また DMND 命令など並列リンク命令で発生させる解析例外を利用し、スタブモジュールで他ＰＥの動作をシミュレートすれば、ＰＥ単体での逐次部分のデバッグも容易になる。

しかし従来型のデバッグ手法が使えるだけでは不十分である。並列プログラムのデバッグが困難なもう一つの理由が、一般的に逐次プログラム以上に並列プログラムが複雑であることが挙げられる。

これを解決する方法には、実行時にイベントヒストリーを取得して後で解析したり、またはプログラムとは別にその振舞いについて記述し、実行時のイベント発生列とチェックする方法などが挙げられる。

イベントヒストリーを取得する単純な方法は、プログラマ自身が必要なイベントを取得するプログラムを書くことである。そのためのライブラリを提供したり、また処理系が自動的にコード中に埋め込んでしまう方法も考えられる。発注／受領型並列計算機では解析例外を利用してソフトウェアプローブを構築し、イベントヒストリーの取得に用いることができる。

ソフトウェアプローブの問題点はやはりプローブ効果を引き起こすことにある。プローブの実行負荷が小さければ、常にプログラムに挿入したままにしておけるが、理想的には実行への影響が小さいハードウェアプローブを用意すべきであろう。発注／受領型計算機ではまだ検討中であるが、最低限の機能として共有メモリ中の特定番地へのアクセスが凍結例外を引き起こす機構を考えている。

さらにヒストリーを取得した場合、そのヒストリーをどう処理するかが問題である。このヒストリーは非常に膨大になる。このためデバッガ側は、最低限として膨大なヒストリーを解析し、その中から重要な部分を拾い出す機能は持たなければならない。

発注／受領型並列計算機のデバッガはＫＰＥのＫモード上におかれる。そして各ＰＥのＳモードにデバッグ用のプローブやさらに高機能なリモートデバッガをおく構成が考えられる。このデバッガがこれらの機構を用い、並列計算機用のブレークポイントや並列制御構造（３．）の状態表示、イベントヒストリーの取得とそれを解析する機能を構築することができる。

８．おわりに

本稿では発注／受領型並列計算機では例外処理機構を構築するために、３つの動作モードが必要であることを述べた。特権レベルはこのモード境界上に設定することで整合する。

そしてこのモデルの上で例外処理機構を構築した。並列計算機の例外としては、他ＰＥに影響せず例外発生ＰＥだけで処理できる解析例外と、計算機全体で処理する必要があり、全ＰＥを凍結する凍結例外に分けて考え、それぞれの処理手続きを述べた。また、ソフトウェア的にこれらの例外を発生させるデバッグ支援機構を構築し、どのように並列計算機のデバッグに応用するかを論じた。

これらの機構は、現在は MC68010上にソフトウェアでエミュレートしている。またこの機構上でのデバッガの実装が今後の課題となる。本稿では触れなかったが、実現に際してはデバッガの表示機構も問題であり、複雑な並列プログラムの状態を理解しやすいように表示するための方法も検討されなければならない。それにからみ、従来のソースレベルよりもっとアブストラクトレベルのデバッガが必要になる。

謝辞

本研究について日頃ご指導いただいている本学斎藤延男教授、阿刀田央一教授、五十嵐智助手に感謝する。また発注／受領型並列計算機の実行系の検討を行った星野浩志氏に感謝する。

参考文献

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

[2] 冨澤ほか：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).

[3] E.F.Gehringer, D.P.Siewiorek, Z.Segall:Parallel Processing, Digital Press, pp.80-84(1987)

[4] C.E.Mcdowell, D.P.Helmbold:Debugging Concurrent Programs, ACM Computing Surveys, Vol.21, No.4, pp.593-622(1989).