解　説

仮想計算機システムの高性能化方式†

梅野英典†† 久保隆重†† 田中俊治††
井上太郎†† 安久津修†††

1. はじめに

仮想計算機（VM）とは，実計算機と類似のアーキテクチャを有する論理的な計算機で，しかも，その大部分の命令が実計算機によって直接的に実行される[1]．

これにより，仮想計算機のある程度の処理性能が保証される．この仮想計算機を実計算機の上で複数台定義し，それらの同時走行を可能とするシステムを仮想計算機システム（VMS）と呼ぶ．その制御プログラムをVMモニタ（VMM）と呼ぶ．VMSは，1台の中央処理装置（CPU）下で運用されるとは限らず，主記憶装置を共有する複数台（N台とする）のCPU下で運用されてもよい．このときは，N-wayのマルチプロセッサシステム下でM台の仮想計算機が定義され，それらが並列に運用されることとなる．仮想計算機システムは，最初1台の実計算機システム（マルチプロセッサシステムでも良い）下で，効率よく複数のオペレーティングシステム（OS）のテストやデバッグを行いたいというニーズから考え出された[1]．その後の性能や機能の改善により，さらに，OSの移行，システムの移行，複数OSの運用，複数システムの運用，サブシステム複合システムの運用，新旧アーキテクチャの同時運用，及び性能評価ツールとして利用されてきている．VMSは上記のような多くの用途を有するが，その実用性は，各VMの性能に大きく依存している．ここでの性能とは主にCPUの性能のことである．たとえばVMのCPUオーバヘッドが実CPUに対し50%存在するとVMのCPU性能が実CPUの1/1.5=2/3になることを意味する．そこで，以下に，VMSの実用性を高めるために，現在までに各社で検討され，採用されてきた，VMを高速化するための方式と評価について解説する．

† Performance Improvement Methods for Virtual Machine System by Hidenori UMENO, Takashige KUBO, Shunji TANAKA, Taro INOUE (Systems Development Laboratory, Hitachi Ltd.), Osamu AKUTSU (Software Works, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所
††† (株)日立製作所ソフトウェア工場

2. 仮想計算機システムのタイプ

以下の用語で実計算機のことを特にVMと対比させるときにベアマシン（bare machine）ともいう．またOSは種々の機能をもつマクロ命令をユーザに提供するが，このユーザからみたマクロ命令のアーキテクチャを拡張マシンと呼ぶ．仮想計算機システムは，VMのアーキテクチャがベアマシンのアーキテクチャであるか，それとも，マクロ命令を主体とする拡張マシンのアーキテクチャを有するかという観点と，VMMがベアマシンアーキテクチャ上で動作するか，または拡張マシン上で動作するかにより**表-1**のように分類される．これを分かりやすく**図-1**に示す．表-1において，ホストアーキテクチャとはVMMからみたアーキテクチャのことである．それぞれのタイプの仮想計算機システムについて，VMの性能，機能について比較し，**表-2**にまとめて示す．本解説は性能及び機能面において最も優位性の高い可能性を有するタイプ1仮想計算機システムについて，その高性能化方式について解説する．しかしその多くはタイプ2仮想計算機システムに対しても適用可能である．以下，VMといえばタイプ1または，タイプ2の仮想計算機システムにおけるVMを示すこととする．

表-1 仮想計算機システムのタイプ

VMのアーキテクチャ ＼ ホストアーキテクチャ	ベアマシン	拡張マシン
ベアマシン	1	2
拡張マシン	3	4

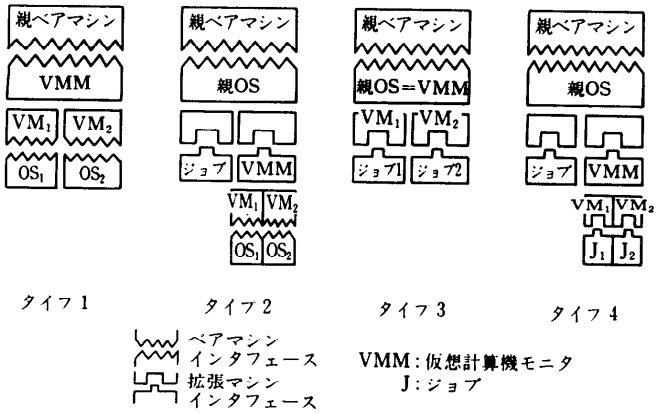


図-1　仮想計算機システムのタイプ

3. 仮想計算機システムの実現方式

1.で述べたように，VM は，ホスト実計算機と類似のアーキテクチャを有する．そこで，本章では，前提となるホスト実計算機のアーキテクチャと仮想計算機システムの実現方式について説明する．このアーキテクチャは，IBM 社の System/370 のアーキテクチャ[2]を基本とするが，それに限定するわけではない．**表-3** にホスト実計算機のアーキテクチャの概要を示す．今後の用語については表-3 または文献[2]を参照されたい．

表-2　仮想計算機システムのタイプと特徴

#	項目＼タイプ	1	2	3	4
1	VM の性能	●VMM のオーバヘッドを受ける	●親 OS と VMM の両方のオーバヘッドを受ける	●親 OS のオーバヘッドを受ける	●親 OS と VMM の両方のオーバヘッドを受ける
2	VMM の機能	●複数 OS の走行 ●プロセッサ論理分割	●複数 OS の走行 ●親 OS の機能使用可能	●ひとつの OS が走行するのみ ●通常のベアマシン OS	●ひとつの OS が走行するのみ ●親 OS の機能を使用可能
3	VM の機能	●VM 上の OS による	●VM 上の OS による ●親 OS の機能使用可能	●VM 上のジョブによる	●VM 上のジョブによる親 OS と別系統のマクロ命令使用可能

表-3　ホスト実計算機アーキテクチャの概要

#	項　　目	概　　要
1	特権状態と非特権状態	CPU の二つの状態，特権状態でしか実行することができない命令を特権命令という．この中には，I/O 命令や制御命令がある．
2	Program Status Word (PSW)	CPU の各種状態，特権状態と非特権状態，割込み可能と不可能状態，プログラムの実行アドレス（論理アドレス）などを含む状態レジスタ
3	ア　ド　レ　ス	プログラムで使用するアドレスを論理アドレスという．各アドレスは 8 ビットからなる 1 バイトの領域を指し示し，4 バイトを 1 語と呼ぶ．
4	仮　想　空　間	セグメント 0 | セグメント 1 | セグメント 2 | … | セグメント n ページ 0 | ページ 1 | … | ページ m
5	アドレス変換	仮想アドレスは図-2 に示すアドレス変換装置により実アドレスに変換される。
6	プレフィクス変換	プレフィクスアドレスを α，実アドレスを y とすると，絶対アドレス z は，$y=(0,\alpha,y_0)$ のとき $z=(\alpha,0,y_0)$ ただし $y_0 \neq 0$，α
7	Translation Look-aside Buffer (TLB)	論理アドレスと対応する絶対アドレスの変換対．図-2 に示すアドレス変換後ここに登録される．TLB に登録されているときは，TLB により高速に変換される．
8	制御レジスタ	PSW に含まれない多くのシステム制御情報を含むレジスタ群．各種制御テーブルのアドレス，各種割込み用のサブマスクなどを含む．
9	I/O プロセサ (IOP，チャネルシステム)	CPU の主記憶装置とディスクなどの補助記憶装置との間のデータ転送を制御する．データ転送の具体的内容はチャネルコマンド語 (CCW) に記述される．

仮想計算機のアーキテクチャは，ホスト実計算機アーキテクチャと類似のアーキテクチャを有する．すなわち，VM の特権状態と非特権状態，実計算機と同様の命令セット，Program Status Word (PSW)，及び，主記憶，仮想記憶，チャネルシステムなどを有する．これは，VM 上の OS からみた場合，実計算機と同様のアーキテクチャを有することを意味するものである．この VM は，ソフトウェア的には以下の三つの方法により実現される[3]．

（1） プロセッサシミュレーション：VM のほとんどの非特権命令は，直接実プロセッサにより実行される．VM の特権命令は，その実行がハードウェアにより中断され，VMM によりシミュレーションされる．プロセッサのもつ命令を機能的にみたとき，一般命令，制御命令，I/O 命令などに分かれる．このうち後者の二つは特権命令であることが多い．制御命令は CPU 全体の制御にかかわる命令である．

（2） メモリシミュレーション：VMM は主に，以下の二つの方法で VM の主記憶を実現する．（a）ホスト主記憶装置のあるアドレス α から β ($\alpha<\beta$) までの領域をある VM に専有させることにより，VM の主記憶装置を実現する．（b）VMM の構成した仮想空間のあるアドレス α から β までを VM の主記憶として専有させる．

（a）による VM を常駐 VM という．常駐 VM で，特にアドレス変位 $\alpha=0$ のものすなわち，VM の絶対アドレス＝ホスト絶対アドレスという属性を有するものを V＝R VM と呼ぶ．この VM は仮想計算機システムのなかで唯一しか存在しない．また，常駐 VM で，特にアドレス変位 $\alpha\neq0$ のものを V＝Resi VM と呼ぶ．この VM は，アドレス変位 α をかえることにより，ホスト主記憶の容量の許す範囲内で複数台用意することができる．（b）による VM を V＝V VM と呼ぶ．VM 上の OS が仮想空間を構成するときは，以下の三つのレベルのメモリアドレス空間が構成される．

ホスト計算機の主記憶（以下単にホスト主記憶ということがある）
VM の主記憶
VM の仮想記憶

このとき，2 段階のアドレスマッピングが以下のように行われる（**図-3** 及び **図-4** 参照）．この図の中で主記憶アドレスとは絶対アドレスのことである．

第 1 段階：VM の仮想記憶アドレスは，VM 上の OS の管理するアドレス変換テーブルにより，VM の実アドレス（VM 上の OS からみたときの実アドレスという意味，以下同じ）に変換される．VM の実アドレスは，VM のプレフィクスレジスタによってプレフィクス変換され，VM の絶対アドレスに変換される．

第 2 段階：VM の絶対アドレスはホスト主記憶の絶対アドレス（単にホスト絶対アドレスともいう）に変換される．その変換方法は，VM の主記憶の種類により異なる．

常駐 VM のとき：図-3 に示すように，常駐 VM 対応に与えられたアドレス変位 α を用いてホスト絶対アドレス＝VM の絶対アドレス＋α により変換される．

V＝V VM のとき：図-4 に示すように，VM の主記憶はホスト仮想記憶アドレスの α から β までを専有するので，

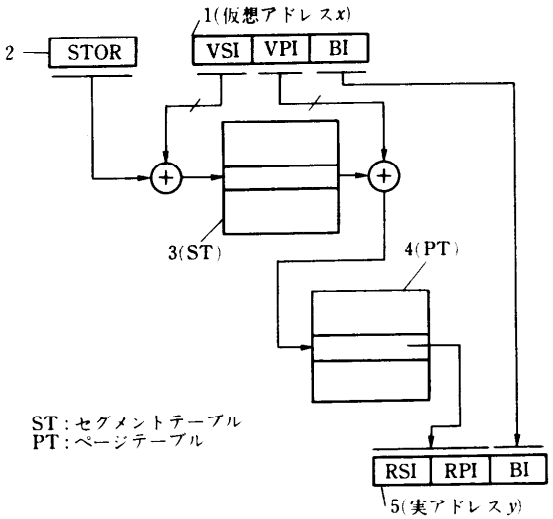


図-2 ペアマシンでの動的アドレス変換

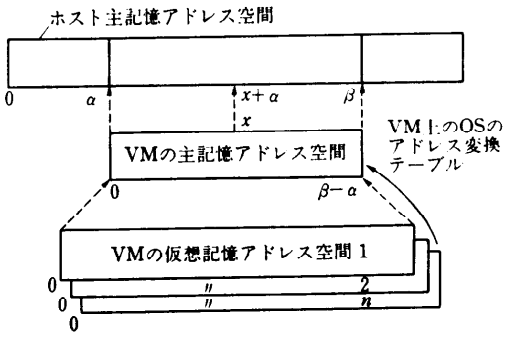


図-3 常駐 VM のアドレス空間の階層

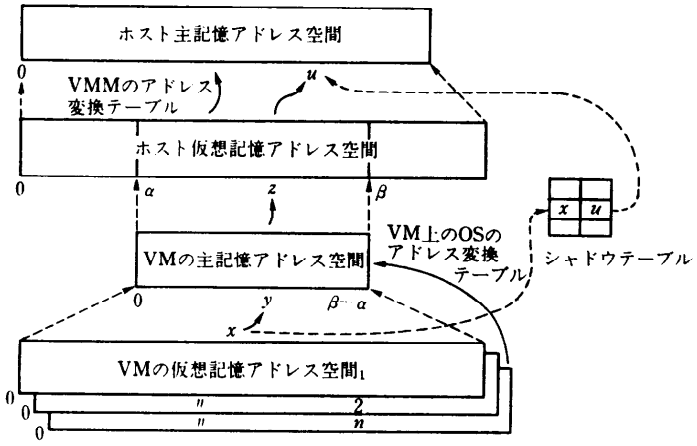


図-4 V=V VM のアドレス空間の階層

VM の絶対アドレス+α → ホスト仮想記憶アドレス

このホスト仮想記憶アドレスは，VMM の管理するアドレス変換テーブルによりホスト主記憶の実アドレス（単にホスト実アドレスともいう）に変換され，これはまた，ホストプレフィクスレジスタによりプレフィクス変換されて，ホスト絶対アドレスに変換される．

常駐 VM のとき，VM の主記憶アドレス空間をホスト仮想アドレス空間上に置き，VMM のアドレス変換テーブルにより，図-3 に示す+α を構成して，V=V VM と同じアドレス空間階層とすることも可能である．

図-3 及び図-4 に示すメモリアドレス空間の階層は論理的概念である．ホスト実計算機がベアマシンとしてのアドレス変換機構（図-2）しかもたないとき，このようなアドレス空間階層をハードウェアとして直接サポートすることはできない．このため VMM は図-4 に示すように VM 上の OS のアドレス変換テーブルと VMM のアドレス変換テーブルを合成したシャドウテーブルを構成する．これは VM の仮想アドレスからホスト主記憶アドレスへのアドレス変換を与えるテーブルである．常駐 VM の場合も図-4 と同様にしてシャドウテーブルを構成しなければならない．このシャドウテーブルであれば図-2 に示すベアマシンの動的アドレス変換機構が動作可能である．

（3） I/O シミュレーション：表-3 の #9 に I/O プロセッサ (IOP) 及び Channel Command Word (CCW) について簡単に説明した．さて，VM 上の OS の発行した I/O 命令は，中断され，VMM に制御が渡る．VM 上の OS は絶対アドレスで CCW を構成する．よって，このとき，この OS を VM 上で動作させたとき，その CCW は一般的には VM の絶対アドレス空間上にあり，そのデータアドレスもまた VM の絶対アドレスとなる．VM の絶対アドレスは，一般に，ホスト絶対アドレスとは異なる．このため IOP は，この VM の CCW をそのままの形で実行することができない．このため VMM はこの VM の CCW と等価な機能を有し，ホスト主記憶上に存在し，そのデータアドレスがホスト絶対アドレスで記述された CCW に作りなおす．これを CCW 変換と呼ぶ．この後，VMM がこの CCW を用いて VM 上の OS に代わって，I/O 命令を発行する．I/O 割込みはいったん VMM にはいり，これが VMM の管理する I/O 制御ブロックに保留された後，もし該当 VM が割込み可能であれば，該当 VM のプレフィクスに割込みが報告される．もし割込み不可能ならば，VMM が保留したままである．

4. 仮想計算機システムのオーバヘッド要因

仮想計算機システムの実用性，適用性は，その性能により大きく左右される．仮想計算機システムは，ホスト実計算機の CPU，主記憶装置，I/O 系（チャネルシステム，I/O 制御装置，I/O 装置）を多重化することにより実現される．したがって仮想計算機のオーバヘッドはこの多重化により発生する．この多重化ということは，単に 1 台しかないハードウェアリソースを N 台に見せかけるというだけを意味するのではない．さらに，それは，ベアマシンとして単一システムイメージをもつ N 台のハードウェアリソースを論理的に独立した M 台のそれに分割する，あるいは見せかけるということをも意味している．ここで，一般には $M \geqq N$ である．この多重化による主要なオーバヘッド要因は以下のとおりとなる．

（1） 制御命令シミュレーション
（2） 割込みシミュレーション
（3） アドレス変換
（4） I/O シミュレーション
（5） 実リソース管理

これらの中で，（1），（2）は CPU シミュレーショ

ンオーバヘッドであり，(3)はメモリシミュレーションオーバヘッドである．(4)は，I/O 命令，I/O 割込み及び I/O 系に関するシミュレーションオーバヘッドである．(5)は，VMM 固有の実リソース (CPU，メモリ，I/O 系) 管理オーバヘッドであるが，(1)～(4)までのシミュレーションを実現するための実リソース管理であり，したがって，前者4つのオーバヘッドの中に含めて表現される．表-4 に，これらオーバヘッドの概要について載せる．

5. シミュレーションの高速化方式

本章では，ホスト実計算機は特別にハードウェアとして VM 支援用機構をもたない場合を考える．このようなとき，仮想計算機システムを高性能化するための第一の手段は VM を実現するための VMM やマイクロプログラムによるシミュレーションオーバヘッドを削減することである．このために，VM の各オーバヘッド要因に対して表-5 に示すシミュレーションの高速化方式がとられている．表-5 の中で VMA (Virtual Machine Assist)，VMA の高速化及び，

表-4 仮想計算機システムのオーバヘッド要因

#	項目	概要
1	制御命令シミュレーション	制御命令のアクセスするリソース (PSW，制御レジスタ，TLB など) の多重化 (VMM，各 VM 用)
2	割込みシミュレーション	VM の割込み可能性判断，保留，VM のプレフィクスへの報告
3	アドレス変換オーバヘッド	ホスト実 CPU がベアマシンのアドレス変換機構しかもたないとき (1) シャドウテーブルの作成…VM の仮想アドレス→ホスト実アドレス (2) VMM のアドレス変換テーブル…VM の実アドレス→ホスト実アドレス
4	I/O シミュレーション	デバイスアドレス変換，CCW 変換，I/O 制御ブロックの更新，実 I/O 装置スケジューリング，割込みの解析，割込みの保留，割込み優先順位判断，割込み可能性判断，割込み報告，VM のスケジュールとディスパッチ
5	実リソース管理	VM 制御ブロック，I/O 制御ブロック，実メモリ管理，実 CPU スケジュール

表-5 シミュレーションの高速化方式の例

#	高速化方式	概要
1	VMA (Virtual Machine Assist)	IBM 社の VM/370 の VMA[7] は，使用頻度の高い，11 個の特権命令と SVC をシミュレーションするマイクロコード．さらに，シャドウテーブルの保守も行う．VM の CPU オーバヘッドを 70～90% 削減．ソフトウェアの解析処理とディスパッチ処理を不要化．
2	CPA (Control Prog. Assist)	VMM の処理をマイクロコード化．中小型機構に対して有効．
3	VMA 自体を高速化	常駐 VM のメモリ属性を利用して制御命令である LRA，RRB，ISK，SSK 命令の VM 上での実行速度を 1.3～2.3 倍高速化．
4	シャドウテーブルバイパス	V=R VM 上の OS のアドレス変換テーブルをシャドウテーブルとして使用．多重仮想空間を有する OS に対して効果大．あるベンチマーク測定では，VM の全 CPU オーバヘッドを 70% 程度削減[4]．
5	マルチシャドウテーブル	VM 上の OS の仮想空間対応にシャドウテーブルを用意する方式．IPTE 命令，Purge TLB 命令に対するシミュレーションオーバヘッド増を招くので，#6 と組み合わせる必要あり．
6	シャドウテーブル選択的無効化	ベアマシン仕様の IPTE 命令は，全シャドウテーブル無効化を招く．VM 仕様 IPTE 命令とすることにより，各シャドウテーブルの該当エントリのみを無効化とする．VM の全 CPU オーバヘッドを 40～60% 削減[3]．
7	高速 I/O シミュレーション	VM の I/O シミュレーションの正常ケースを高速化．
8	高速 CCW 変換	常駐 VM のメモリ属性を利用して CCW 変換を高速化．#7 と合わせて，I/O シミュレーションオーバヘッドを 1/2～1/3 に削減[3]．
9	リソース専有化	実計算機のリソースを分割して専有化させる．二重管理の防止．
10	ハンドシェーキング	VM 上の OS が，VM 上で走行していることを認識し，VMM と提携して性能向上を図ること．

表-6 シミュレーションの高速化方式と有効性

#	高速化方式 / オーバヘッド要因	1 VMA	2 CPA	3 VMA自体を高速化	4 シャドウテーブルバイパス	5 マルチシャドウテーブル	6 選択的シャドウテーブル無効化	7 高速I/Oシミュレーション	8 高速CCW変換	9 リソース専有化	10 ハンドシェーキング
1	制御命令シミュレーション	○	○	○	○	○	○				○
2	割込みシミュレーション	○	○	○							○
3	アドレス変換	○		○	○	○	○				○
4	I/O シミュレーション		○					○	○		○
5	実リソース管理									○	○

注) ○: 有効

CPA (Control Program Assist) はマイクロプログラムによる高速化であり，その他はソフトウェアによる方式である．さらに，**表-6** に各高速化方式とその有効性について概要を載せる．

これらのシミュレーションの高速化による全体的な性能向上効果は，**6.3** に直接実行方式と一緒にまとめて示す．

6. 直接実行方式

前章で述べたシミュレーションの高速化方式では，実計算機のアーキテクチャはそのままにしておいて，ソフトウェアまたはマイクロプログラムによる高速化方式であるのでその効果にはおのずと限界がある．これらの方式では，どうしても VMM 及び VMA のオーバヘッドが残り，VM の性能は，実計算機の性能に近づくことはできない．このためホスト実計算機が，VM の各種制御命令，I/O 命令，各種割込みを直接実行する方式が採用されている[7),8)]．本章は，この直接実行方式を述べる．

6.1 V=R 方式

これは，ホスト実計算機と同一のアーキテクチャを有する V=R VM だけを直接実行する方式である[9)]．これは，V=R VM のメモリアドレッシングは，プレフィクシングを利用して絶対 0 ページを VM が使用すれば，そこを含めて

VM の絶対アドレス＝ホスト絶対アドレス

となる．よって，V=R 領域の上限値さえハードウェア的にチェックすれば，V=R VM のメモリアドレッシングは，他にハードウェアを付加することなくベアマシンのそれと同一となる．しかも，シャドウテーブルも不要である．そこで，V=R VM 走行中は，VMM は停止しているから，アーキテクチャ上のリソース (PSW，制御レジスタ，タイマなど) をそのまま，該 VM に与えてしまう方式が考えられる．このようにすれば，ほとんど全ての制御命令や割込みは直接実行可能となる．さらに，チャネル以下の制御装置及び装置を全て V=R VM に専有化させれば，VM 上の OS の I/O 命令，I/O 割込みも VMM の介入なしに，すなわち，VM 上の OS とハードウェアだけで直接実行することができる．

V=R 方式は，走行中の VM の割込み要因だけしかハードウェアとしては判断できない．しかも，走行中は，全て V=R VM 上の OS に制御されるから VMM の割込み制御手段も存在しない．この V=R 方式では，ひとつしかない実リソースを V=R VM 上の OS が直接アクセスするから，該 VM 走行中は，実計算機全体が V=R VM 上の OS に制御されることになる．たとえば割込み可能性の制御がそうである．しかし，この方式では，V=R VM 上の OS がシステム全体の中で第 1 の優先順位をもつ OS である場合が多く，該 OS が走行中，システム全体を制御しても運用上差し支えないと思われる．

6.2 仮想アーキテクチャサポート方式

これは，ホスト実計算機がまともに，VM のアーキテクチャをハードウェアとしてサポートする方式である．

6.2.1 VM のメモリアドレッシング直接サポート方式

V=R VM 及び V=Resi VM の主記憶は図-3 に示すようにホスト主記憶を分割して専有している．V=V VM の主記憶は，ホスト仮想アドレス空間に用意される．これらのメモリアドレス空間の階層は図-3，図-4 に示すとおりである．これらの VM の制御命令，I/O 命令を直接実行するためには，ハードウェア (VM 支援機構と呼ぶ) が以下の機能を装備することが必要である．

(1) VM がアドレス変換モードのとき，VM の論理アドレスを，該 VM 上の OS の作成したアドレス変換テーブルにより VM の実アドレスに変換すること．

(2) 該 VM のプレフィクス値により，プレフィクス変換を行うこと．

(3) VM の絶対アドレスを対応するホスト絶対アドレスに変換すること．

V=R VM 及び V=Resi VM に対して，これを実現するアドレス変換機構として V=Resi VM 用アドレス変換機構 (V=Resi 用 DAT ということもある) が提案されている[10]．**図-5** に V=Resi VM 用アドレス変換機構を示す．この V=Resi 用 DAT はベアマシンの DAT (native DAT) とは以下の点で異なる．以下図-3 に示す V=Resi VM の主記憶領域を単に領域ということもある．

(1) V=Resi 用 DAT の使用するアドレス変換テーブルは領域内実 (または絶対) アドレスで構成されている．しかし native DAT の使用するアドレス変換テーブルはシステムの実 (または絶対) アドレスで構成されている．

(2) アドレス変換テーブルを構成するセグメントテーブルの先頭アドレス，そのエントリアドレス，ページテーブルのエントリアドレスは全て領域内の実 (または絶対) アドレスである．したがって V=Resi 用 DAT は，上記の各アドレスに，領域の開始アドレス α を加算してシステム絶対アドレスに変換して，VM のアドレス変換テーブルを検索する機構を有さなくてはならない．これに対して native DAT は，セグメントテーブル先頭アドレスからはじめてアドレス変換テーブルの各エントリアドレスをそのままシステム絶対アドレスとして検索すればよい．

(3) 論理アドレスは複数の領域において同時に発生するため，これを領域間で区別する必要性がある．このため，V=Resi 用 DAT は，Translation Look-a-side Buffer (TLB：表-3 の #7 参照) に領域を区別するための情報として領域識別子を登録する．

(4) V=Resi 用 DAT は，VM がアドレス非変換モードのときは，論理アドレスをそのまま領域内実アドレスとする．

このように V=Resi 用 DAT により実 CPU は，V=Resi VM の主記憶領域内の論理アドレス，仮想アドレス，実アドレス，絶対アドレスを認識することができる．これにより，V=R 及び V=Resi VM についてシャドウテーブルは不要となる．さらに，VM の主記憶からホスト主記憶へのアドレス変換テーブル (すなわち VMM のアドレス変換テーブル) も不要となる．なぜなら V=Resi 用 DAT をもつ実 CPU は，VM の主記憶を直接アクセスすることができるからである．

V=V VM に対しては，図-4 に示すメモリアドレス空間の階層を直接サポートする方式が実現されている[8]．すなわち，VM 上の OS の作成したアドレス変換テーブル (VM の仮想アドレス→VM の実アドレス) と VMM の作成したアドレス変換テーブル (ホスト仮想アドレス→ホスト実アドレス) の両方をハードウェア/マイクロプログラムにより検索する方式である．この場合，VM 上の OS のアドレス変換テーブルは，VM の実 (または絶対) アドレスで構成されているから，それを検索するにも，VMM のアドレス変換テーブルを検索しホスト実アドレスに変換して検索しなければならない．これを V=V 用 DAT と呼ぶことがある．これにより，V=V VM に対してもシャドウテーブルは不要となる．

常駐 VM に対しても VMM が VM の実アドレスをホスト仮想アドレスとして，そこからホスト実アドレスへのアドレス変換テーブルを構成すれば，この V=V 用 DAT を

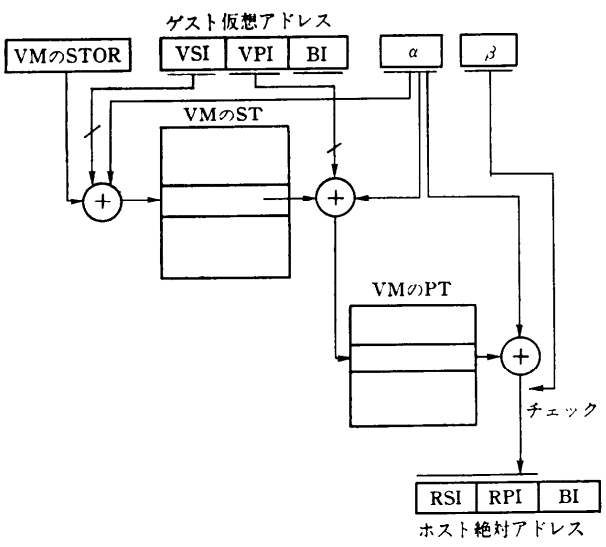


図-5 V=Resi 用動的アドレス変換機構

適用することができる[12].

V=Resi 用 DAT は native DAT とほとんど同一性能であるが，V=V 用 DAT は，3～4倍の時間がかかる．この性能差はアドレス変換対が TLB に登録されているときは影響ないが，Not in TLB ratio が1～2% になると平均命令実行時間に 10～20% のオーダの悪影響を及ぼす．よって，常駐 VM に対して実計算機と同等の性能を要求するときは，V=Resi 用 DAT を使用するほうが望ましい．

以上の VM 用のアドレッシング機構により，実 CPU は，シャドウテーブルなしに VM の論理アドレス，VM の仮想アドレス，VM の実アドレス，VM の絶対アドレスをハードウェアとして直接アクセスすることができる．

6.2.2 制御命令の直接実行方式

仮想計算機システムのオーバヘッドの中で，制御命令シミュレーションが主要な位置を占めることは，4. で説明したとおりである．VMA は VMM の介入なしに VM の制御命令を実行することができるが，全て VMM が主記憶上に構成した VM の制御ブロックをアクセスしてシミュレーションしており，VMM の一部分を代行しているだけである．このため，VMA の適用範囲は，狭く，しかも比較的遅い．これに対して VM の制御命令をハードウェア/マイクロプログラムが直接実行するということは，VM のアーキテクチャをハードウェアがサポートするということである．すなわち，VMM 用のハードウェアリソースとは独立に，VM 上の OS からみえるハードウェアリソースである PSW，制御レジスタ，汎用レジスタなどをハードウェアとして用意することである．VMM からみて，ハードウェアにより，VM のアーキテクチャがサポートされていればよい．したがって，VM のリソースを文字どおりハードウェアレジスタとして用意してもよく，あるいは，ハードウェア内部のマイクロプログラム操作により，1個しかない実リソースを退避回復して実現しても良い．VM のリソースとしてハードウェアレジスタが使用されれば性能が良いが，ハードウェア内部のローカルストレジであっても主記憶よりは2～3倍は速い．

VMM は，VM をスタートさせるとき，

（1） その VM のリソースの値を VM のアーキテクチャに対して設定する．

（2） 実 CPU を VM モードとして VM をスタートさせる．

通常は，VM アーキテクチャをサポートする実 CPU は，この(1)，(2)を実行するための命令を用意しており，VMM は，その命令を使用することにより，VM をスタートさせることができる[8]．このほか，VM 走行中に，この VM 支援機構が，処理しきれない事象が発生することがある．このときは，VMM にコントロールが渡され，VMM により VM のスケジュールや割込み保留処理などが行われる．

6.2.3 割込みの直接実行方式

仮想アーキテクチャサポート方式では，VM 用の PSW や制御レジスタがハードウェアとしてサポートされる．したがって，VM の I/O 割込みや，外部割込みなどの制御（すなわち，割込み保留，割込み可能性の判断，割込みの報告など）は，ハードウェアにより実現される．その実現方式にはいくつか考えられる．第1は，走行中の VM の割込みについてだけハードウェアが制御する方式である．この場合，ハードウェアとしては，通常の VMM 用の割込み制御手段に加えて，走行中の VM 用の割込みマスクレジスタや割込み可能性判断回路などをもつ．非走行の VM の割込み要因については，該 VM 用の制御手段をもたず，ハードウェアで保留したり，VMM の割込み要因にしたり，または，割込みインタセプションにより VMM にコントロールを渡したりする．

第2は，非走行 VM についてもハードウェア側で割込み制御論理をもつ方式である．これは，ホストマルチプロセッサになると割込み先プロセッサの選択手段や，現在非走行 VM が次の瞬間に他のプロセッサにより走行状態となる可能性があるため各プロセッサから共通の位置に各 VM の割込み制御手段をもたなくてはならなくなるであろう．さらに，ゲストマルチプロセッサになると，その多重度に比例した割込みマスクレジスタや割込み可能性判断手段をもたなくてはならなくなる．このため，この第2の方式は特許としては，出願されているが，製品としてリリースされるには至っていないようである．

表-7 に以上の制御命令及び割込みの直接実行方式についてまとめる．

6.2.4 I/O 命令直接実行方式

VM の制御命令や，タイマなどの外部割込みを直接実行しても，VM の I/O 命令や I/O の割込みをシミュレーションしていては VM の性能は上がらない．高 I/O 負荷の OS（及びその下の応用プログラム）を VM 上で動作させると，I/O シミュレーションのオー

表-7 制御命令及び割込み直接実行方式

#	方式		概要	例	性能	VM台数	VM用ハード
1	V=R 方式		V=R VM 走行時実リソースをそのまま VM に与える.	• IBM VM/SP HPO • アムダール VM/PE • 日立 HPA	○	1	少
2	仮想アーキテクチャサポート方式	(1)	走行中の VM のアーキテクチャをハードウェアとしてサポートする.	• IBM PR/SM™ • 日立 VM/EX • 富士通 AVM/EF • 日電 VMX	△〜○	N	中
		(2)	非走行 VM についてもハードウェアでサポート.	特許レベル	○	N	大

バヘッドが目立ってくる. たとえば, Disk to MT などの負荷では, 制御命令を全て直接実行したとしても, VM の CPU オーバヘッドは, ベアマシンと比較して 100% 程度にも達する. このため, VM の I/O シミュレーションを低減させなければ, 実計算機と同等の性能は望めない.

このために, VM の I/O 命令, I/O 割込みを VMM の介入なしに VM 上の OS とハードウェア/マイクロプログラムだけで実行しようとするのが, ここでいう I/O 直接実行方式である. I/O 命令や I/O 割込みは, CPU のみならず, チャネルや制御装置側の論理が関係してくるため, 制御命令の直接実行ほど簡単ではない. 仮想アーキテクチャサポート方式では, チャネルや制御装置の共有を前提とし, しかも, 複数の VM に対して, I/O 直接実行をサポートするのが目的である.

VM の I/O 命令を直接実行するには, まず, VMM が行っている装置スケジューリングを不要にする必要がある. このため, VM の専有装置に対してだけ, I/O 命令の直接実行をサポートするのは妥当である. ただし, システム間の装置共有の場合は, OS 間で各チャネルから該装置に対するリザーブ/リリースを実行することにより, I/O 直接実行が可能である.

VM の I/O 命令が, チャネルコマンド語 (CCW) の実行系列すなわちチャネルプログラムを指定するときは, これをハードウェアで直接実行できなければならない. これについて以下に述べる.

話を簡単にするために, I/O 起動命令のハードウェア仕様として, その指定する CCW のアドレス及びその中のデータアドレスは共に絶対アドレスでなければならないものとする. VM 上の OS の用意した CCW を直接実行するためには, その CCW や, データの存在するページがホスト主記憶上に常駐すなわち固定されていなければならない. V=V VM の場合は, その主記憶は, 必ずしもホスト主記憶上に常駐ではない (図-4 に示すメモリアドレス空間参照). したがって, V=V VM の場合このホスト主記憶へのページ固定ということは, VMM が介入して, 該 CCW をチェックしないかぎり困難である. よって, V=V VM の CCW の直接実行は困難な上に効果も少ないから採用しないほうがよい. ところが, V=R VM や V=Resi VM の場合は, 初めから, その主記憶はホスト主記憶に常駐である (図-3 に示すアドレス空間参照). 今, I/O 起動命令の仕様により, VM の CCW アドレスやそのデータアドレスは VM の絶対アドレスで構成されているから, その CCW やデータは, VM の主記憶上に常駐している. したがって, 常駐 VM の場合は, 初めからホスト主記憶上にも常駐していることになる. よって, ホスト主記憶へのページ固定の必要性はない. さらに, VM の絶対アドレスからホスト絶対アドレスへの変換もアドレス変位 α の加算だけでよい. よって, この場合は, チャネルシステム側でこの定数加算による CCW のアドレス変換を行えば, チャネルシステムが, この VM の CCW を直接実行することができる. この方式は, I/O 命令直接実行において広く使用されている[7),10),12),13)].

6.2.5 I/O 割込みの直接実行方式

ベアマシンの I/O アーキテクチャとして, I/O 割込み要因がチャネルごとにキューイングされるアーキテクチャと, チャネルとは独立で, 論理的な割込み優先度を表すサブクラス (0〜N) にキューイングされるアーキテクチャがある. 前者の例には, IBM 社の 370 アーキテクチャのチャネルシステムがあり, 後者の例には, 同社の 370-XA : 拡張アーキテクチャのチャネルシステムがある. ホスト実計算機が前者の場合は, V=R 方式で I/O 直接実行をサポートする. ホスト実

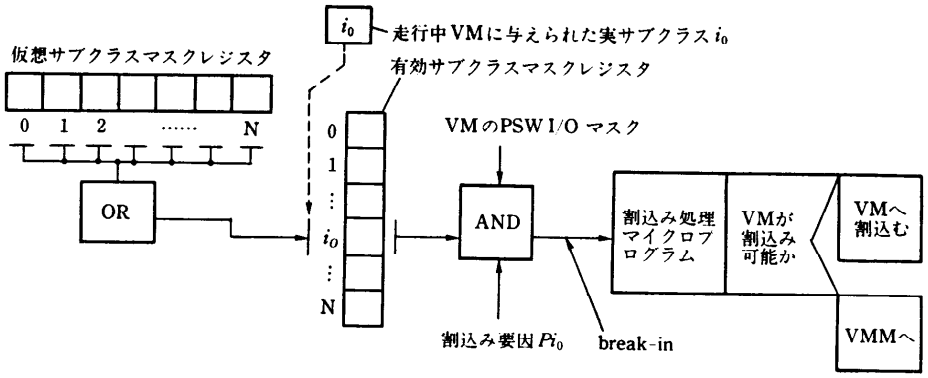


図-6　単一サブクラス方式

計算機が，後者の場合は，チャネルスケジューリングは，チャネルシステム自体が行うので，必ずしもチャネルを専有する必要はない．この場合，I/O 割込みの直接実行に関していくつかの方式が考えられている．これらについて以下に述べる．

（1）　単一割込みサブクラス方式

これは，実割込みサブクラスとしては (0～N) の1組であり，I/O 直接実行を行う VM に対してただ一つだけ実サブクラスを排他的に与える方式である．IBM 社の PR/SM™ (Processor Resource/Systems Manager) は，この方式と思われる[12]．このとき，VM 上の OS が使用するサブクラスすなわち仮想サブクラスは 0～N の N+1 個存在してもよい．このときのハードウェアの割込み制御手段は，以下のようになる（**図-6** 参照）．すなわち割り当てられた実サブクラス i (0～N のいずれか) の VM 側のサブクラスマスク値としては，仮想サブクラス 0～N のマスク値の OR を設定する．この結果と，VM の PSW の I/O マスクとによる割込み可能性を判断する．これで，割込み可能と判断されたときは，I/O 割込み処理用マイクロプログラムへブレークイン (break-in) する．このマイクロプログラムで，該当 VM の本当の割込み可能性をその VM のサブクラスマスクレジスタすなわち該当の仮想サブクラスマスクと VM の PSW の I/O マスクにより判断し直す．この結果，割込み可能であれば，マイクロプログラムにより直接 VM に割込みを報告する．割込み不可能ならば，VMM へコントロールを渡し，VMM が割込みを保留する方式である．

この方式は，仮想サブクラスマスクの全ての OR をとることからハードウェアとしては比較的簡単となるが，マイクロプログラムによる再判断のためやや I/O 割込み直接実行の性能が低下することとなるであろう．しかし OS の使用する全ての割込みサブクラスの割込み要因が直接実行の対象となるという良さもある．

（2）　複数割込みサブクラス方式

これは，I/O 直接実行を行う VM に，1個以上の実サブクラスを専有的に与え（これを専有サブクラスという），さらに，仮想サブクラスとは1対1の対応をつける方式である（**図-7** 参照）．この場合は，VM の専有サブクラスについてのマスクレジスタ（単に専

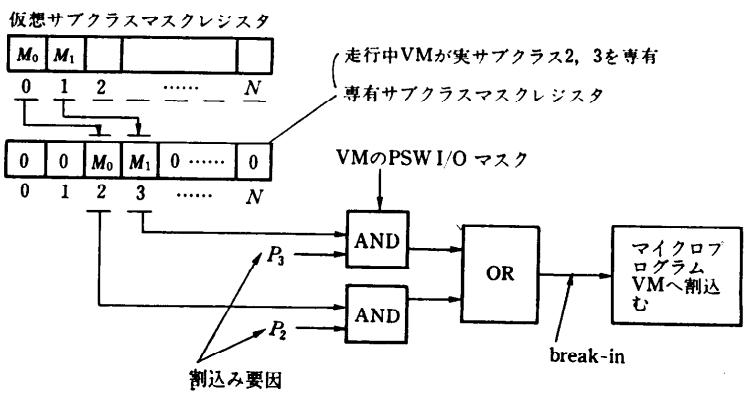


図-7　複数サブクラス方式

有サブクラスマスクレジスタと呼ぶ）を用意し，そこに，対応する仮想サブクラスマスクの値を設定する．これと，VM の PSW の I/O マスクの論理積をとることにより，専有サブクラスに関しては，図-7 に示すハードウェア論理により，該 VM の割込み可能性を正確に判断することができる．したがって，このとき，I/O 割込み処理用のマイクロプログラムへブレークインしたときは，そのマイクロプログラムは直接 VM に割り込めばよい．このマイクロプログラムの処理は，ベアマシンの場合と，それほど差はない．日立の高速 VM 機構 VM/EX は，この方式を採用している[14]．この方式は，ハードウェア論理だけで正確に VM の割込み可能性を判断することができるので，方式(1)に比べマイクロプログラムのオーバヘッドが少ない．しかし，仮想サブクラスと専有サブクラスとを1対1に対応づけることから OS の全てのサブクラス 0～N の割込み要因を直接実行することはできない．それは，ハードウェアの実サブクラスは，0～N の1組しかないから，これを一つの OS だけで全て専有するわけにはいかないからである．もっとも，既存の OS で実際に使用するサブクラスは1～3個であるので，I/O 直接実行を行う VM の台数を数台に制限すれば，その上の OS のサブクラスに排他的に実サブクラスを割り当てることはできる．

(3) VM 台数比例方式

これは，各 VM ごとにハードウェアとして，サブクラス 0～N を用意し，VM ごとに割込み要求キューと，サブクラスマスクレジスタ及び PSW の I/O マスクを用意する．さらに，VM ごとにハードウェアとして割込み可能性判断論理が存在する．この場合，I/O 割込み要因を発生している装置を専有している VM が走行中であれば，該 I/O 割込みを直接実行し，非走行中のときは，ハードウェア的に保留するか，または，VMM にコントロールを渡す．この方式は，各 VM の割込み可能性を正しく判断でき，VMM の介入を低減できる良さがある．しかし，サブクラスマスクレジスタや，割込み可能性判断論理が，VM 台数と VM がマルチプロセッサ構成のときの多重度の積に比例して増加することから，実現性は低いであろう．この方式は特許では出願されているが[26]，現在のところ製品としてはリリースされていないようである．

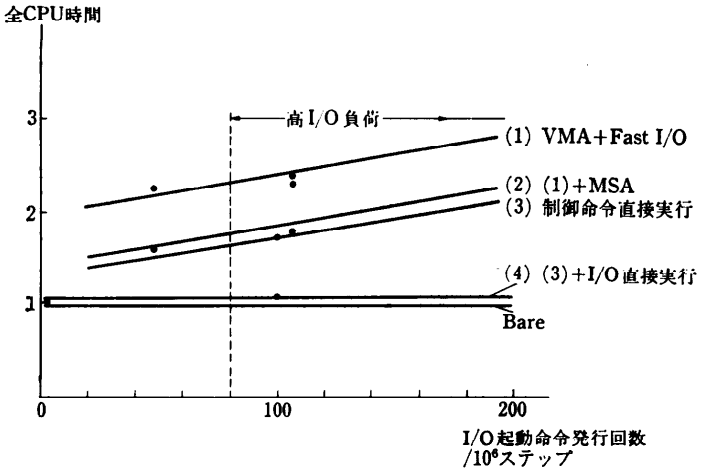
以上の I/O 直接実行方式を表-8 にまとめて示す．

6.3 性能評価

ここでは，5. のシミュレーション高速化から直接方式までの全体的な性能効果を述べる．VM の CPU 性能は，測定の経験からみて，おおよそ負荷の I/O 起動命令の発行頻度に反比例している．この様子を，図-8 に示す．OS を含む全体の CPU 時間で考えて，ベアマシンのときの CPU 時間を1とする．VMA がないときはベアマシンの 6～10 倍の CPU 時間がかかりこの図には示されない．I/O 起動命令発行頻度がベアマシン上での OS を含む全 CPU 走行ステップの 100 万ステップ当たり 100 回の負荷を考える．この I/O 負荷は相当に高く，通常のほとんどのバッチジョブは，これより I/O 発行頻度は低い．図-8 に示すとおりシミュレーションの高速化で 1.8～1.9 までにはなる（同図(1)，(2)参照）．これはマルチシャドウでさらにその選択的無効化処理を採用した場合である．一方 I/O 以外の制御命令の直接実行により 1.7 ぐらいまでに下げることができる（同図(3)参照）．後者（同図(3)のケース）の場合は 6.2 で述べたようにハードウェアアーキテクチャが図-3，図-4 に示す VM のメモリアドレス階層を直接サポートするので，VMM はシャドウテーブルを作成する必要がない．このためシャドウテーブルにともなうオーバヘッドがなく，したがって安定した性能が得られる．前者（同図(1)，(2)のケース）の場合は，シャドウテーブルの最適処理を行った場合であるので，VM 上の OS の

表-8 I/O 直接実行方式の比較

#	方式		概要	例	性能	VM台数	VM用ハード
1	V=R 方式		チャネル，制御装置，装置を V=R VM にだけ専有化させる．	• IBM 社 VM/SP HPO PMA • 日立 HPA	○	1	小
2	仮想アーキテクチャサポート方式	単一サブクラス方式	実割込みサブクラスをひとつだけ専有化	• IBM 社 PR/SM™	△～○	N	中
		複数サブクラス方式	複数の実割込みサブクラスを専有化	• 日立 VM/EX	○	N	中
		VM 台数比例方式	VM ごとにサブクラス，割込み要求キューを用意	• 特許レベル	○	N	大



注) 1　●は測定点
2　Fast: I/O: 高速 I/O シミュレーション+高速 CCW 変換
3　MSA: シャドウテーブル選択的無効化方式

図-8　VM の CPU 性能

仮想空間数が多くなってくると次第にオーバヘッドが1を越える傾向にある．さらに I/O 直接実行で 1.05〜1.07 にまで削減される（同図(4)参照）．なお，直接実行方式においては，V=R VM と V=Resi VM の性能は同じと考えてよい．I/O 直接実行まで含めることにより，I/O 負荷にかかわらず，複数の VM に対して Near-Native Performance が達成されるといえる．

7. マルチプロセッサシステムにおける VM 制御方式

ホスト実計算機がマルチプロセッサシステムである場合（これをホストマルチプロセッサシステムということがある）と，VM 上で動作する OS がマルチプロセッサモードで動作する場合（これをゲストマルチプロセッサシステムということがある）とがある．さらに，両者が同時に運用される場合もある．以下これらについて説明する．

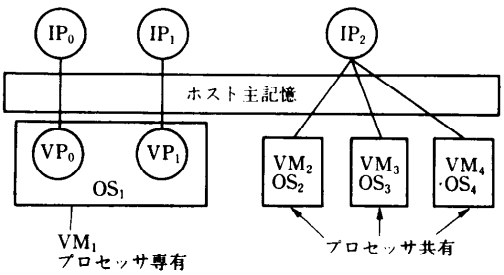
7.1 プロセッサ割当て方式

ホストマルチプロセッサのとき，VM がプロセッサを専有する方式と，各 VM 間でプロセッサを共有する方式とがある（**図-9** 参照）．

複数プロセッサを専有化させるのは，VM 上の OS のマルチプロセッサ性能を引き出すためである．このとき，他のプロセッサで別の OS を動かしテストなどに使用したいため VM 運用とするのである．このようなソフトウェアによるプロセッサの論理的なグループ分けは，プロセッサ，主メモリ，及びチャネルの割当を自由に変えられることから，物理的な分割方式よりも自由度が高い．

7.2 マルチプロセッサ命令制御方式

マルチプロセッサのサポート方式では，マルチプロセッサを制御する命令をどう処理するかが主要な課題である．プロセッサ間通信命令やプレフィクスレジスタの設定，参照の命令は，VMM のシミュレーションまたは，直接実行により処理することができる．VM 上の OS の発行する TLB 無効化命令については各プロセッサの TLB が VM 用エントリを含んでいるので，それらの適正な無効化方式をサポートする必要がある．たとえば，Purge TLB 命令を例にとると，これはこの命令発行元の CPU の TLB の全エントリをパージする命令であるこれを VM 上の OS が，発行したとき，TLB 内の全エントリをパージすると他の VM や VMM の性能に悪い影響を与える．したがって，TLB 内の該当 VM の該当仮想プロセッサ（すなわち，Purge TLB 命令を発行した仮想プロセッサ：VM がユニプロセッサモードなら仮想プロセッサは1台だけ）のエントリをパージするようにしなければならない．ベアマシンの場合は，仕様により，PTLB 命令を発行した CPU の TLB だけをパージすればよい．今，VM を構成する仮想プロセッサで PTLB 命令を発行した仮想プロセッサを VP_0 とする．この VP_0 用の TLB エントリが各 CPU に



IP: 命令プロセッサ
VP_0, VP_1……VM_1 のプロセッサ（ゲストマルチプロセッサ）

図-9　プロセッサ専有と共有

存在する可能性があるためそれらをパージしなければならない．このため，VMM が VP_0 をサービスする CPU を切り換えて，新 CPU で VP_0 を走行させるとき，その CPU の VP_0 用 TLB エントリをパージする方式がとられている[8]．

このほか，たとえば，IPTE (Invalidate Page Table Entry) 命令は，OS のページングに使用される．この命令は，OS のページテーブルの該当エントリを無効化するとともに，マルチプロセッサを構成する全CPU の TLB の該当エントリをパージする命令である．VM 上の OS がこの命令を発行したときは，VM を構成する各仮想プロセッサに，通信して，その全仮想プロセッサ用の TLB エントリをパージしなければならない．ホストマルチプロセッサのときは，実際に該当 VM の全仮想プロセッサの識別子をパラメータとして，各 CPU にパージ信号をおくることとなる[8]．各 CPU では，それに基づいて，自分の TLB の上記全仮想プロセッサについて該当エントリをパージすることとなる．

7.3 スピン制御方式

マルチプロセッサシステムの OS は，ロックを確保するために割込み不可能状態でスピンしたり，他のプロセッサでの処理完了を待つためにスピンしたりすることがある．この OS をゲストマルチプロセッサシステム上で動作させたとき，上記のスピンが本来の目的に沿ったスピンとなるかは，プロセッサの運用形態によることとなる．たとえば，図-9 に示すように，VM を構成する各仮想プロッセサが実プロセッサを専有しているときは，相手の仮想プロセッサが動作していることは保証されているから，上記のスピンは本来の意味をもつことができる．ところが，各 VM がプロセッサを共有する場合は，相手の仮想プロセッサが動作しているとは限らず，したがって，上記スピンが無駄な CPU 消費となることがある．特に，相手の仮想プロセッサの処理完了を待つような場合は，自分の仮想プロセッサがいくらスピンしても相手の処理完了を検知できず，したがって，OS が異常終了することがあり得る．これを防止するためには，OS とのハンドシェーキングが必要である．たとえば，OS がスピンにはいるときには，OS から VMM をコールしてもらい，VMM が相手の仮想プロセッサを動作させるようにしなければならない (**図-10** 参照)．

7.4 性能評価

ホストマルチプロセッサをひとつの OS で動作させたときと VM に分けて複数の OS を使用して運用したときではどちらが性能(スループット)を向上させやすいであろうか．2-way ホストマルチプロセッサのときの運用形態として，VM の運用形態とベアマシン運用形態を考える．VM 運用の場合は，VM ごとにシステムプログラムが存在し，リソースを VM 間で分離することが容易である．その分 VM ごとに I/O 系のリソースが必要となりシステムプログラムは二重にもつことになるが VM 間の I/O 系，及びメモリ競合を比較的少なくすることができる．したがって，二つの VM のジョブ間でもリソース競合を少なくすることができる．これに比べ，ベアマシン運用では，OS はひとつだから，単にリソースを増やすだけでは多重ジョブ間におけるリソース競合を減らすことはできず，ライブラリやデータなどの分割やシステムプログラムの常駐化やジョブ制御文の変更などの多くのチューニングが必要である．このシステムチューニングによりスループットを向上させることはできるが，プロセッサに比例した性能を実現することは非常に困難であろう．

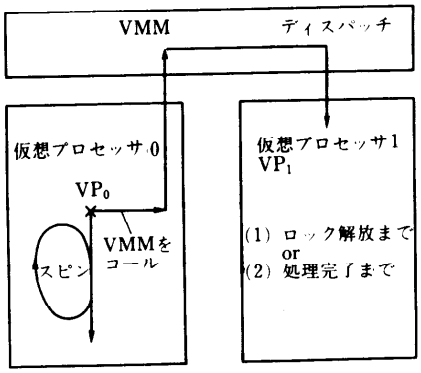
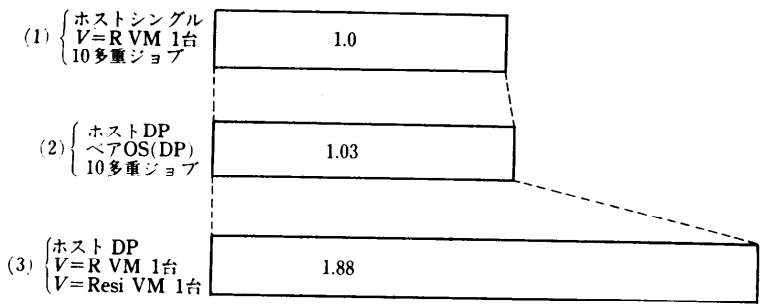


図-10 スピンに関するハンドシェーキング

図-11 にベンチマークによるスループット測定例を示す．同図に示すように，ベアマシンのときは，プロセッサ数を2台にし，主記憶容量を倍に増やしてもスループットはわずかしか増えない．ジョブの多重度を上げてもイニシエータ間の競合によりかえって性能が低下する傾向にある．これは OS を含むシステム常駐ボリューム (SYSRES) の中に測定用のジョブ，ライブラリ，ユーザデータが登録されているから，そこで装置競合が発生しているからである．したがって，OS 側のチューニングをしなければ，たとえベアマシン運用側に2倍の I/O 系リソースを与えたとしてもスループットの向上は困難である．一方，VM 運用ではスループットは1.8倍となっている．これは，VM の場合は I/O 系のリソースをベアマシンよりも2倍使



(1) ゲストシングル V=R 32 MB
(2) マルチプロセッサモード OS, 64 MB, 13多重だと 0.96
(3) 各ゲストシングルかつ各10多重ジョブ, 各32 MB
使用リソース, (3)の I/O 系リソース (チャネルパス, DKC, Disk) は, (2)の2倍である.
測定用ジョブ, (3)は2系統の SYSRES にそれぞれ格納
(2)は唯一の SYSRES に格納
DP: Dyadic Processor
CPU 利用率1台当り 20~30%

図-11 二つの VM による I/O 系統分割の効果 (スループット) 測定

用しており, これらの測定用リソースが二つの I/O 系の二つの SYSRES に分割されているため, 装置競合はベアマシンのときと比べて半分以下となるためである. 以上のことにより, I/O 系リソースを VM ごとに用意できる場合は VM 運用は, 高多重プロセッサの運用にとって性能を引き出しやすい形態であるといえる.

8. 各社の仮想計算機技術

表-9 に各社の仮想計算機技術にについて示す.

9. おわりに

おわりに今後の仮想計算機技術の発展方向について述べる. そのひとつは多重ホスト多重 OS 運用である. 主記憶装置を共有するマルチプロセッサの多重度が上がるにつれ, それ全体をひとつの OS で制御するのは性能的にもまた保守の上からも得策ではない. したがって, 一般の仮想計算機システムやプロセッサ論理分割方式で, マルチプロセッサシステムのプロセッサをグループ分けしたり, 主記載, 拡張記憶を分割したり, I/O 系リソースをグループ分けし, いくつかの OS で運用したほうが, 全体的な性能, 拡張性, 及び柔軟性に優れているといえる. アムダール社の MDF や IBM 社の Logical Partition は, プロセッサ論理分割方式の例である.

その二つはサブシステム複合システムの運用である. これは, VM ごとにサブシステムを構成しサブシステム相互間の連携により全体的にひとつのシステムを構成する方式である. このようなサブシステム複合システムの利点は以下のように考えられる. (1)性能: 各サブシステム間の独立性と依存性が明らかとなり, プロセッサ割当スケジューリングが容易となる. その分ロックオーバヘッドを低減することができるであろう. (2)保守性, 拡張性, 柔軟性: 各サブシステム単位に独立に保守ができる. さらに, いくらでも新しいサブシステムを追加していくことができる. 全体として増殖可能 (incremental) なシステムとなる. このようなシステムの例として, IBM 社の VM/XA SP の Group Control System (GCS) をあげることができる. この GCS は, ネットワークサブシステムを標準的にサポートしており, GCS 配下には, VTAM, RSCS, Net View などのサブシステムが VM として動作する[16].

その三つは大規模テストシステムの運用である. 仮想計算機システムのひとつの大きな用途は, OS の開発用としての VM である. このため, OS のテストシステムとしての用途は, これからも大きく発展していくであろう. OS のテストシステムの形態は以下のように考えられる. (1)VM オープン使用型: 各 VM 上で, それに割り当てられたリソースを使って, それぞれの OS の開発者が立ち会ってテストする形態である. 最も初歩的な VM を利用した OS のテスト形態といえる. (2)ドライバ VM 型: これは, 各被テスト VM のほかに, 全体のテストシステムを制御する VM (ドライバ VM) を設定する方式である. このドライバ VM 上の OS のサポートする会話型端末から各被テスト VM 上の OS を制御する方式である. この例としては, 日立製作所の OSTD (Operating System Test Driver)[27] がある. (3)分散システムテスト型: これは, ひとつは, LCMP (Loosely Coupled Multi-Processor) システムを, 複数 VM と仮想 CTCA (Channel To Channel Adaptor) または仮想 CNU (Connection Unit) を用いて, VM 間で仮想的

表-9 各社の仮想計算機技術

#	各社	主要仮想計算機技術
1	IBM 社	(1) 1973 (S48) : VMA 付きの VM/370 リリース[3] (2) 1981 (S56) 10月 : VM/SP HPO PMA アナウンス，翌年半リリース[15] 優先 VM (V=R VM のこと) の性能を実計算機と同等にする. (3) 1987 (S62) 6月 : VM/XA SP アナウンス，翌年 3Qにリリース[16] ここではじめて V=F VM 導入. 4台の優先 VM (V=R と V=F) に対して I/O 直接実行 (Sie アシスト) └→7台まで拡張 (4) 1988 (S63) 2月 : Logical Partition 方式アナウンス，同年 3Q リリース[17] このとき PR/SM™ (Processor Resource/Systems Manager) 発表 これは，特許[12]によれば，単一割込みサブクラス方式. アムダールの MDF 対抗
2	アムダール社	(1) 1984 (S59) 12月 : Multiple Domain Feature (MDF) 発表 翌年 2Q リリース[18]
3	富士通	(1) 1982 (S57) 6月 : AVM/EF アナウンス，同年12月リリース[7],[19] V=D VM を導入 制御命令直接実行， VM 上の OS の CCW を直接実行 (2) 1989 (H1) 6月 : AVM/EX 及び EVM をアナウンス[20] H2，7月リリースの予定 オーバヘッドを平均 1/2 (最大 1/3)
4	日電	(1) 1983 (S58) 2月 : VM/4 on ACOS-4 試作[21] (2) 1986 (S61) : VMX on ACOS-2000[22] タイプ2の仮想計算機システム
5	日立	(1) 1975 (S50) : VMA 付きの仮想計算機システム VM/M 試作 (2) 1976 (S51) : シャドウテーブルバイパス方式試作[4] (3) 1977 (S52) 5月と10月 : 常駐 VM 基本特許出願[10],[11] (4) 1978 (S53) : Creep-in/Creep-out VM 試作[23] (5) 1979 (S54) 9月 : 仮想計算機システム VMS リリース[24] (6) 1983 (S58) : シャドウテーブル選択的無効化方式リリース[5] (7) 1984 (S59) 1月 : I/O 直接実行方式特許出願[25] (8) 1988 (S63) 6月 : I/O 直接実行方式特許出願 (ゲスト／ホスト有)[14] (9) 1988 (S63) 3月 : VMS/ES リリース……制御命令直接実行 (10) 1988 (S63) 12月 : VMS/ES リリース……I/O 直接実行[13] (複数サブクラス方式)

に構成し，LCMP 環境でテストしたいというニーズに応える．もうひとつは，複数台の VM と，仮想 CCP (Communication Control Processor) と仮想回線により仮想的なネットワークを構成するテスト形態である．同様に，ドライバ VM 下の1台の会話端末を用いて，これら複数の被テスト VM を制御する．(2)，(3)のテスト形態で，ドライバ VM でのテストプロシージャやテストデータの蓄積ができ，その選択が容易になれば，テスト工数削減に大きく寄与するであろう．

その4つは多重アーキテクチャシステムの運用である．大型及び超大型汎用計算機アーキテクチャは，現在，IBM 社の 370-XA 及び ESA (Enterprise Systems Architecture)/370 が事実上の世界標準である．しかし，同じ IBM 社の AS/400 の one level storage アーキテクチャなどの出現もあり，今後ますますアーキテクチャが多様化され拡大されていくと思われる．このような中で，ソフトウェアの互換性を維持していくことは大切なことである．このためアーキテクチャ自体にもいろいろと工夫がなされていくであろう．たとえば，370-XA の論理アドレス 24 ビットと 31 ビットのバイモーダル CPU がその例である．この場合ユーザプログラムの互換性はこれにより保たれているが OS は新たに作り直すところが多い．アーキテクチャが飛躍的に拡大または変更されるときは，新 OS は新アーキテクチャだけを高性能にサポートするようにし，仮想計算機システムを用いて新旧アーキテクチャ，新旧 OS を同時運用し，ユーザプログラムの互換性を保持したほうが OS 開発費や全体のシステム性能という点においてよいと思われる．

参 考 文 献

1) Goldberg, R. P.: Architectural Principles for Virtual Computer System, Ph. D. dissertation Div. Eng. Appl. Phys., Harvard Univ., Cambridge, MA (1972).
2) IBM System/370-XA Principles of Operation SA 22-7085.
3) IBM, Virtual Machine Facility/370, IBM SYST J., Vol. 18, No. 1 (1979).
4) Taguchi, T.: Design and Experiments of a Virtual Machine System, J. Inform. Processing, Vol. 2, No. 3, pp. 149 (1979).
5) Umeno, H. et al.: Reduction of 2-0-Translation Table Maintenance Overhead in a Virtual Machine System, J. Inform. Processing, Vol. 8, pp. 28-39 (Mar. 1985).
6) Umeno, H. et al.: Development of a High Performance Virtual Machine System and Performance Measurements for It, J. Inform. Processing, Vol. 4, pp. 68-78 (July 1981).
7) 小野：仮想計算機システム高性能化の一手法，富士通，情報処理学会計算機システムの制御と評価研究会資料 18 (1983年2月4日).
8) IBM: IBM System/370-XA Start Interpretive Execution, SA 22-7095.
9) Bean, G. H.: Method and Means Switching System Control of CPUs, Patent No. 4, 494, 189, Filed Apr. 26, 1982, IBM.
10) Umeno, H. et al.: Virtual Machine System, Japanese laid-open Patent Applications 54-52929, 1979. 特願昭 52. 10.
11) 広沢他：記憶制御装置，特開昭 53-142137，特願昭 52. 5. 18.
12) ジョージ・ヘンリー・ビーン, et al.: データ処理システムの制御方式，優先権 1987. 7. 29，特開昭 64-37636，IBM, PR/SM™ の特許.
13) 梅野他：仮想計算機における I/O 直接実行方式の提案，第 40 回情報処理学会全国大会論文集 (平成元年3月).
14) 田中他：仮想計算機の入出力実行方式，特願昭 63. 6. 30，日立.
15) IBM: VM/SP HPO PMA announcement (Oct. 1981).
16) Virtual Machine/Extended Architecture System Product (VM/XA SP) Release 2, Programming Announcement (June 11, 1987).
17) IBM: IBM 3090 Processor Resource/Systems Manager (PR/SM™) Feature, IBM Product Announcement (Feb. 17, 1988).
18) Amdahl Announces Dual Operating System Option, Computer World (Dec. 3, 1984).
19) 富士通：FACOM ジャーナル, Vol. 8, No. 9 (1982).
20) 日経 BP 社：日経コンピュータ，1989年7月31日号，pp. 147.
21) 難波他：VM/4 (ACOS-4 仮想計算機) のアーキテクチャ，情報処理学会計算機システムの制御と評価研究会資料 18 (1983年2月4日).
22) 海老野他：ACOS システム 2000 シリーズの拡張仮想計算機システム，NEC 技報，Vol. 40, No. 11 (1987).
23) 田口他：実計算機モードと仮想計算機モード間の動的切り換え制御方式について，情報処理学会論文誌，Vol. 22, No. 3 (昭和56年5月).
24) 仮想計算機システムの開発，日立製作所HITAC製品ニュース (昭和54年7月).
25) 梅野他：仮想計算機システムの入出力実行方式，特願昭 59. 1. 18，日立.
26) 金田他：入出力割込み処理方式，特願昭 60-194070，富士通.
27) Yoshizawa, Y. et al.: Test and Debugging Environment for Large Scale Operating Systems: OSTD, COMPSAC, Tokyo, Japan (Oct. 7-9, 1987).

(平成2年3月20日受付)