解　説

IPS

並列処理のためのシステムソフトウェア

2. 並列オペレーティング・システム†

福　田　晃††

1. はじめに

情報化社会を反映して，計算機の世界に下（シーズ）と上（ニーズ）から，多様化の波が押し寄せている．シーズからみると，ハードウェア技術の進歩により，多様なハードウェア・アーキテクチャをもつ中規模並列マシンが商用化され，さらに大規模／超並列マシンへの模索が行われている[1]．一方，ニーズからみると，ユーザの限りない計算能力への要求，および新しい応用分野への適用の期待がある．また，多様なニーズに対応してプログラミング・モデルの多様化がある．

このような中で，並列オペレーティング・システム（並列 OS；マルチプロセッサを対象としたOS）に求められている機能は何であろうか？　またどのような機構を提供したらよいのか？　静的さらには動的スケジューリング，データ配置のサポートなど，高度化する並列化言語処理系技術の中で，並列 OS の役割が問われている．さらに言えば，並列処理を一つの契機として，OS そのものが問われている．

本稿では，これらについて，最新の動向に触れながら解説する．まず，**2.** では，多くのレベルにおける計算機環境の多様化を概説し，並列 OS の必要性・役割について並列化言語処理系との関係に触れながら述べる．**3.** では，並列 OS の現状を概説する．さらに，**4.** では，多様化に対処すべく，最近注目されているソフトウェア技術として，並列化言語処理系と密接な関係がある並列処理実行環境モデルとページ配置問題について述べる．**5.** ではまとめで今後の展望を簡単に述べる．

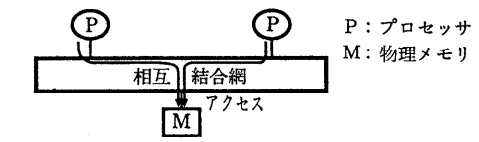
2. 計算機環境の多様化と並列 OS の必要性・役割

2.1 計算機環境の多様化

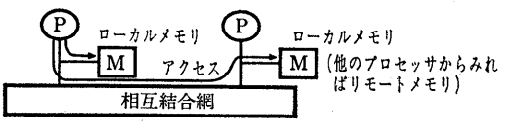
（1） ハードウェア・アーキテクチャ

現存するマルチプロセッサは，メモリ・アーキテクチャの観点から大きく次の三つに分けられる（**図-1** 参照）．

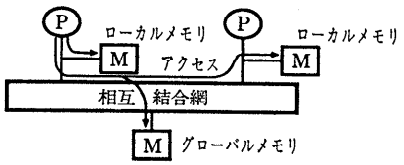
1) UMA (Uniform Memory Access) 型（**図-1**(a)）：物理メモリは共有メモリでかつ，どのプロセッサからも任意のメモリ・アドレスに大体同



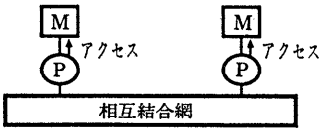
（a） UMA 型



（b） NUMA 型
（ローカル／リモート・アーキテクチャ）



（c） NUMA 型
（ローカル／グローバル／リモート・アーキテクチャ）



（d） NORMA 型

図-1 メモリアーキテクチャからみたマルチプロセッサの分類

† Parallel Operating Systems by Akira FUKUDA (Department of Computer Science and Communication Engineering, Kyusku University, Nara Institute of Science and Technology).

†† 九州大学工学部情報工学科（奈良先端科学技術大学院大学）

じ時間でアクセスできるアーキテクチャである．バス結合のマルチプロセッサがその典型である．商用機としては，Sequent 社の Symmetry, S シリーズ，Silicon Graphics 社の Challenge および Power Challenge などがある．

2) NUMA (Non-Uniform Memory Access) 型(図-1(b), (c))：物理メモリは共有メモリでかつ，プロセッサからのメモリ・アクセスが均一でないアーキテクチャである．プロセッサ間の相互結合網として，スイッチング機能をもつエレメントから構成されている多段結合網をとる．NUMA 型マルチプロセッサは，さらに各プロセッサから等距離でアクセスできる共有メモリ（グローバル・メモリ）があるかどうかにより，ローカル／リモート・メモリ・アーキテクチャ(図-1(b))，ローカル／グローバル／リモート・メモリ・アーキテクチャ（図-1(c)）の二つに分かれる．商用機としては，BBN 社の Butterfly シリーズ，最近では Kendall Square Research 社の KSR 1[2] などがある．

3) NORMA (No Remote Memory Access) 型(図-1(d))：共有メモリをもたず，プロセッサからのアクセスはプロセッサ対応のローカル・メモリに対して行われるアーキテクチャである．nCUBE 社の nCUBE, Intel 社の iPSC などがある．

上記は大きな分類であり，実際は，クラスタ構成，共有メモリと私的メモリ (private memory) の混在，ハードウェア・キャッシュの装備，また，ディスク構成などに関して，多くのハードウェア構成が存在する．

（2） プログラミング・モデル

プログラミング・パラダイムと呼ぶべきかもしれないが，従来の手続き型に加えて，関数型，論理型，オブジェクト指向型，さらには制約型など多様化の傾向にある．その他にも種々のプログラミング・モデルが出現しつつある[3]．

（3） 応用分野拡大への期待

従来の数値計算に加えて，大規模シミュレーション（ゲノム解析，物理／社会現象，VLSI 設計などの CAD)，人工知能など，応用分野拡大への模索[1]，さらには「柔らかな情報処理」を目指した通産省のプロジェクト[4]が始まっている．

2.2 並列 OS の必要性と役割

（1） 必 要 性

以上のように，多くのレベルの多様化が始まろうとしている今，OS の存在価値が問われている[5]．

1) OS は有益な仕事をしないオーバヘッドのかたまりである，2)並列化言語処理系研究者またはマニアックなユーザからみれば，OS には何もして欲しくない（OS が勝手なことはしないで欲しい)，など，OS の必要性に関し否定的な意見がある．これは一部真実である．では，OS は必要ないのか？ 答はノーである．多様なプログラミング・モデル，多様なハードウェア環境，および今後の変化に適応し，ユーザに並列処理を浸透させるための OS が必要なのであると考える．ただし，従来の単一プロセッサ用およびシーズ・ニーズの多様性があまりないときに構築され，限られたニーズに対する要求を単層で処理していた OS とは異なり，昨今の多様性に対処するための基盤・枠組みを与える OS が求められている．

（2） 並列化言語処理系との役割分担

プログラミング言語自身にシステム資源割当要求の命令を埋め込むかどうかに関する研究は文献6)に譲るとして，並列化処理系レベルでユーザ・プログラム内のスケジューリング（静的／動的を含む)，およびデータ配置を行う研究が最近盛んである[6]．これは，言語処理系がそのプログラムの実行に関する多くの情報をもっているので，プログラムの高速実行のための自然な研究の流れである．並列 OS が介在すると，プログラムの実行に関する情報が不足し，数々のオーバヘッドが生じるためである．プログラムごとに適したスケジューリング方策，データ配置は異なり（**4.2** で述べる)，プログラム内の実行は並列化言語処理系が責任をもち，並列 OS は，マルチプログラミング環境において，並列化言語処理系が要求するシステム資源の管理・調整の役割を基本とする．このとき，並列化言語処理系からは並列 OS が提供する仮想マシンしかみえないが，効率的な実行を行うためには，プログラムがどのように実際にハードウェア資源（物理プロセッサ，物理メモリなど）にマッピングされているかを並列化言語処理系が知る必要がある．並列 OS は，並列化言語処理系に必要なハードウェア資源割当情報を提供

しなければいけない．これに関しては，**4.1** で述べる．また，並列 OS の機能として，プログラムおよびユーザ間の保護機能が特に重要となる．

3. 並列 OS の現状

分散 OS (Distributed OS) と呼ばれているものでもマルチプロセッサをも対象とした OS を並列 OS に含めると，実際に稼働している並列 OS としては，海外では，Mach[7)~9)], Chorus[10)], Topaz[11)], Psyche[12)~15)], 日本では TOP-1 OS[16)], SKY-1[17)], Omicron V 3[18)], MUSTARD[19)] などがある．さらに研究段階の並列 OS がいくつかある[20),21)]．Mach, Chorus が有名であるが，これらは種々の文献で紹介されているので，ここでは Psyche を紹介する．

Psyche は，Rochester 大学で開発されている NUMA 型マルチプロセッサを対象とした並列 OS で，現在 BBN Butterfly Plus マルチプロセッサ上で稼働している．マルチモデルの並列プログラミング環境の提供を主な目的とし，メモリ管理に特徴がある．メモリ管理はスケーラビリティ (scalability)，移植性 (portability)，汎用性 (generality) を目的としている．Psyche 上のプログラミング・モデルはデータ抽象化された realm と呼ぶデータ抽象に基づいている．単一空間をユーザ空間に提供し，すべての realm はこの単一空間にマッピングされることになる．**図-2** に示す 4 階層のメモリ空間構造を有している[13)]．NUMA (non-uniform memory) 空間は，ローカルな物理メモリを管理し，ハードウェアのページテーブルをもっている．UMA (uniform memory) 空間は均一アクセス・メモリ空間を見せるためのものであり，そのために UMA ページの移動 (migration) や複製 (replica) などの手法が採られている．一般的なページの移動，複製に関する議論は，**4.2** で述べる．VUMA (virtual memory) 空間は，仮想メモリをサポートしている．PUMA (Psyche memory) 空間はカーネル・インタフェースを与えている．また，マルチモデルの並列プログラミングをサポートする枠組みを Psyche 上で開発している[22)]．メモリ管理が階層化されているので異なるハードウェアへの移植性に優れている，ページの移動，複製方策が変更しやすいなどの特徴がある．

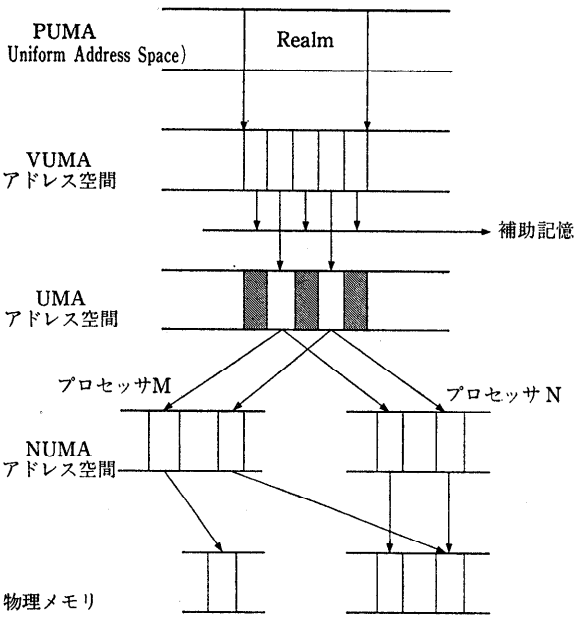


図-2 Psyche メモリ管理階層

Mach の外部ページャ (用語解説参照)，Psyche のマルチモデルの並列プログラミングのサポート，後述する新たな並列実行環境モデルなど，並列 OS は多様化への対処に向けて動き出していると言える．

4. ソフトウェア技術

4.1 並列処理実行環境モデル

ユーザ・プログラムは，ユーザまたは並列化言語処理系によって複数のアクティビティに変化し，それらがカーネルの提供する仮想プロセッサ (virtual processor) にマッピングされ，さらに仮想プロセッサはカーネルのスケジューラによって物理プロセッサ (physical processor) にマッピングされて並列実行されることになる (**図-3** 参照)．

ここで UNIX カーネルを考える．仮想プロセッサは UNIX プロセスとなる．一つの UNIX プロセスは一つの仮想アドレス空間と一つのコンテクスト (用語解説参照) しか提供していないので，複数のアクティビティを並列実行させようとすると，複数の UNIX プロセスを生成させて，それらにアクティビティを割り当てて実行することになる．UNIX プロセスの生成には，仮想アドレス空間の生成およびファイル資源などの管理をとも

なうので，比較的粒度の小さなアクティビティを並列実行させる場合，UNIX プロセス操作のオーバヘッドが無視できなくなり，効率的な実行ができない．そこで，UNIX プロセスを用いた実行環境よりも軽い並列実行環境の提供が必要となる．

軽い実行環境モデルとして，スレッド・モデルが注目されている．スレッドとは，コンテクストのことである．UNIX プロセスでは一体化されていた，仮想アドレス空間（およびファイル資源）とコンテクストを分離したものである．このスレッドをどのレベルで管理するかに関して，まず次の二つが考えられた．

（1） カーネルレベル・スレッド・モデル（図-4(a)参照）

カーネルが仮想プロセッサとしてスレッドを提供し（カーネル・スレッド），ユーザ・アクティビティはカーネル・スレッドに1対1にマッピングされて実行されるモデルである．カーネルがスレッドを提供している OS に Mach, Topaz などがある．本モデルは，UNIX プロセスの自然な改良と言える．本モデルは，ユーザ・アクティビティがカーネル・スレッドとして実行されるので，後述するユーザレベル・スレッド・モデルに比べて，ユーザ・アクティビティのカーネル内での動きがユーザ（空間）に見えやすいという利点がある反面，以下の欠点が分かってきた．

1) 高い性能を引き出せないこと．

ユーザ・アクティビティの生成，消滅などの操作はシステム・コールとなる．システム・コールの発行は，空間をユーザ空間からカーネル空間へ切り替える，ユーザ空間のスタックに詰まれているシステム・コールの引き数をカーネル空間にコピーする，などの操作をともなう．さらに引き数のチェックなども必要となる．これらのオーバヘッドによって，高い性能が引き出せない．

2) 柔軟性に欠けている．

スレッド・モデルはカーネルが規定することになるので，ユーザはカーネルが規定したスレッド・モデルに合わせてプログラミングしなければいけなくなる．また，スレッド・スケジューリングをカーネルが行っているため，異なるユーザ・プログラムに対しても同一のスケジューリング方策を用いることになり，おのおののユーザ・プログラムに適したスケジューリング方策を実現しにくい．

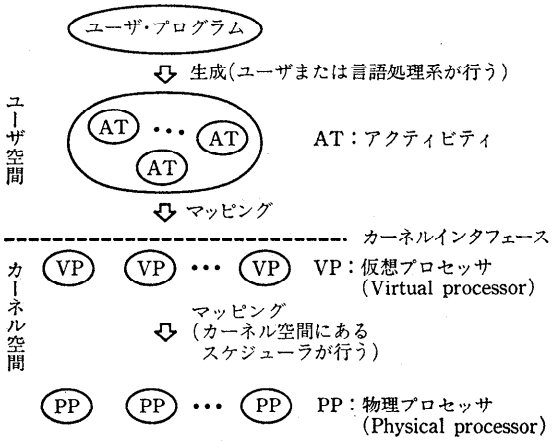
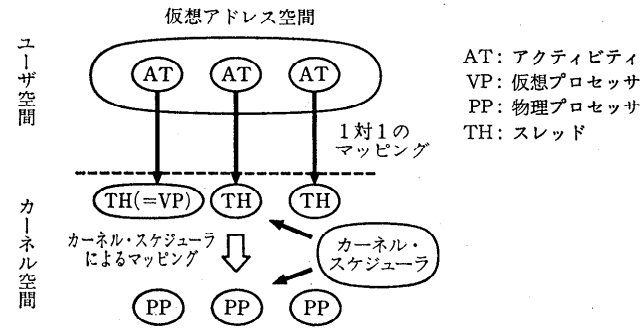
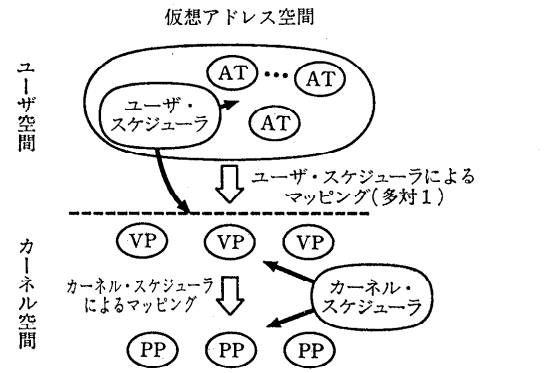


図-3 並列実行環境モデル



（a） カーネルレベル・スレッド・モデル



（b） ユーザレベル・スレッド・モデル

図-4 カーネルレベル・スレッドとユーザレベル・スレッド

3) オーバスペックになりやすい.

カーネル・スレッドで，より多くのスレッド・モデルを包含しようと思えば，カーネルが種々の機能を提供しておく必要がある．すなわち，あるプログラミング・モデルでは必要がない機能でも，他のプログラミング・モデルにとって必要な機能は提供しておかなくてはならない.

(2) ユーザレベル・スレッド・モデル (**図-4** (b)参照)

ユーザレベル・スレッド・モデルは，ユーザ空間でスレッドの生成，消滅およびコンテクスト・スイッチなどのスレッド管理を行うものである[23)~26)]．ユーザ空間内の実行時ルーチンでユーザ・スレッドをスケジューリングできる．すなわち，ユーザ・スレッドのスケジューリング方策をそのプログラムに適した方策にユーザが定義できるという観点からは，効率的な並列実行が行える可能性がある（しかし，後述する問題が発生し，必ずしも効率的な並列実行は望めない)．コンテクスト・スイッチをユーザ空間のみで行えるかどうかは，プロセッサ・アーキテクチャに依存するが，ほとんどのマイクロプロセッサではユーザ空間のみで可能である．ただし，SPARC プロセッサはレジスタ・ウインドの操作が特権命令（カーネル空間でしか実行できない命令）になっているので，ユーザ空間のみでは実現できない．本モデルは，従来言語レベルで提供されていたものである[27)]．実行形態としては，ユーザ・スレッドをカーネルが提供する仮想プロセッサにユーザ空間レベルで割り当てて実行することになる．ユーザレベル・スレッド・モデルは，カーネルレベル・スレッド・モデルのような空間切替えを必要とせず，関数呼出程度の軽さでスレッド操作が実現できることが最大の長所である．最大の欠点は，ユーザ空間から仮想プロセッサの物理プロセッサへのマッピング状況を知ることができない点である．特に，仮想プロセッサがブロックされたことがユーザ空間から見えないことが問題となる．仮想プロセッサがブロックされる契機として以下がある.

●カーネル空間実行中のブロック：入出力関連のシステム・コールの実行にみられるように，システム・コール実行中に仮想プロセッサがブロックされる.

●ユーザ空間実行中のブロック：ページ不在によるページフォールト（用語解説参照）がその典型である.

また，仮想プロセッサからの物理プロセッサの横取りは，タイマ割込みにより生じる.

仮想プロセッサのブロックは，以下の弊害を生じる.

●ユーザまたは並列化言語処理系が物理プロセッサ数以上のユーザ・スレッドを生成し，それらを物理プロセッサ数分の仮想プロセッサに割り当てて実行している場合を考える．このとき，ある仮想プロセッサがブロックされると，実行できるユーザ・スレッドおよびアイドルな物理プロセッサがある場合でも，仮想プロセッサが足りないために実行可能なユーザ・スレッドを実行できなくなる (**図-5**(a)参照).

●ユーザ・スレッドが物理プロセッサに割り当てられることを想定して，スレッド同期方策をユーザ空間が実現しても，実際に割り当てられているかどうか分からないため，効率的な実行ができない.

(3) 協調型ユーザレベル・スレッド・モデル (**図-5**(b)参照)

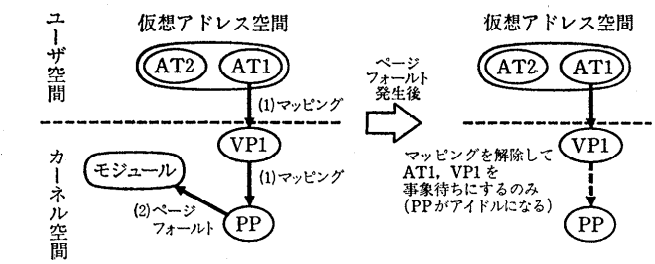
上記の問題は，仮想プロセッサの物理プロセッサへの割当てがユーザ空間から見えないことに本質がある．すなわち，システム・コールという形で新たな情報の流れがユーザ空間からカーネル空間への一方向しかない点である．そこでカーネル空間からユーザ空間への情報伝達をも行う研究が最近注目され始めている[22),28)]．これは，カーネルレベル・スレッド・モデルがもつ機能性 (functionality) とユーザレベル・スレッド・モデルがもつ高性能と柔軟性 (flexibility) の両方の長所の提供を意図したものである.

Washington 大学では，スケジューラ・アクティベーション (scheduler activation) と呼ばれる機構を提案している[28)]．また，本機構を提供する仮想プロセッサをもスケジューラ・アクティベーションと呼んでいる.

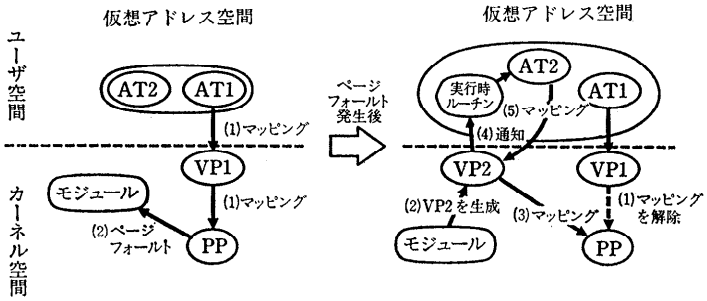
従来のユーザレベル・スレッド・モデルとスケジューラ・アクティベーションとの主な違いは，仮想プロセッサがブロックされたときの対処方法である (図-5 参照)．ユーザレベル・スレッド・モデルでは，仮想プロセッサがブロックされるとその上で走っているユーザ・スレッドも自動的に

ブロックされ，仮想プロセッサがブロックされたことがユーザ空間に通知されない．一方，スケジューラ・アクティベーションでは，スケジューラ・アクティベーションがブロックされると，カーネルは新たなスケジューラ・アクティベーションを生成し，この生成されたスケジューラ・アクティベーションがユーザ・スレッドがブロックされたことをユーザ空間に通知し，処理をユーザ空間に委ねている．**表-1**(a),(b)にカーネルとユーザ空間とのインタフェースを示す．

スケジューラ・アクティベーションと似たスレッド・モデルが Rochester 大学で提案されている[22]．



(a)　ユーザレベル・スレッド・モデル



(b)　協調型ユーザレベル・スレッド・モデル

図-5　二つのユーザレベル・スレッド・モデル
(ページフォールト時の処理)

上記は，スケジューリングに関する情報をユーザ空間に通知し，ユーザレベルのスケジューラを起動させるものであるが，この概念は，一般的にユーザ空間内のサーバを起動する機構へと拡張できると考えられ，今後の研究が待たれる．

4.2　仮想ページ配置問題

NUMA 型マルチプロセッサでは，プロセッサからのメモリアクセス時間が物理アドレスによって異なるので，仮想ページ（用語解説ページフォールトの項参照）をどの物理ページ（用語解説ページフォールトの項参照）にマッピングするかという仮想ページ配置方法がプログラムの実行時間に大きな影響を与える．ここでは，ページングシステムの NUMA 型マルチプロセッサにおける仮想ページ配置方法について述べる．

深刻な問題は，複数のプロセス（またはスレッド）で共有された仮想ページの配置である．仮想ページ配置方法は以下の二つに分類できる．

1)　静的方法：仮想ページは一つの物理ページにマッピングされた後は，物理メモリが不足して2次記憶に追い出される（ページアウト）場合を除いて，そこに固定される．

2)　動的方法：仮想ページの物理ページへのマッピングをプログラム実行中に動的に変更する．

表-1　スケジューラアクティベーションにおけるインタフェース

(a)　スケジューラアクティベーションからユーザ空間(アップコール)
({ }内はユーザレベル・スレッドシステムがとるアクション)

Add this processor (プロセッサ番号): {実行可能なユーザレベルスレッドを実行する.}
Processor has been preempted (横取りされたアクティベーション番号，マシン状態): {横取りされたアクティベーション上で実行されていたユーザレベル・スレッドをレディーキューにつなぐ.}
Scheduler activation has blocked (ブロックされたアクティベーション番号):
Scheduler activation has unblocked (ブロック解除されたアクティベーション番号，マシン状態): {ブロックされていたアクティベーション上で実行されていたユーザレベル・スレッドをレディーキューにつなぐ.}

(b)　ユーザ空間からカーネル(システムコール)
({ }内はカーネルがとるアクション)

Add more processors (必要なプロセッサ数): {プロセッサを当該ユーザ空間に割り当てて，割り当てられたプロセッサでアクティベーションを実行する.}
This processor is idle(): {他のユーザ空間がプロセッサを必要としていたら，当該プロセッサを横取りしてそのユーザ空間に割り当てる.}

静的方法は実現方法が単純であり，管理のオーバヘッドが2）に比べて少ないが，物理メモリアクセスの不均一性を十分に活かし切れていないという問題がある．動的方法は，物理メモリアクセ

スの不均一性を考慮して，極力ローカル・メモリにマッピングしようとする方法であり，管理のオーバヘッドが生じるという欠点があるが，不均一性を活用できる可能性があり，最近研究されている[29),30)]．以下，動的方法について述べる．

一般に，共有する仮想ページの動的管理として，以下の二つがある．

1) ページを複製する方法：仮想ページをアクセスするすべてのプロセッサのローカル・メモリに仮想ページをコピーして，アクセス時間を短縮しようとする方法である．

2) ページを移動する方法：仮想ページをアクセスするプロセッサの中でいずれか一つのプロセッサのローカル・メモリに仮想ページをマッピングする方法である．他プロセッサからはリモート・アクセスとなるので，アクセス時間がかかる．

1)，2)のいずれを用いるかは，仮想ページへのアクセス属性とアクセス頻度などに依存する．読出ししか行わない仮想ページには方法 1) が適している．ただし，アクセス頻度が低い場合は，ページを複製せず，リモート・アクセスが有利となる．一方，読み書きする仮想ページに方法 1) を用いると，ハードウェア・キャッシュ・コヒーレンス問題と同じ問題が生じることになる．すなわち，ページの複製間のデータの一貫性を保つ制御（コヒーレンス制御）が必要となり，このオーバヘッドが生じる．読み書きする仮想ページに対しては，1)，2)両方の研究が行われている．

動的方法は，ローカル・メモリをキャッシュとして利用する考え方であるから，従来から研究されている共有メモリ型マルチプロセッサのハードウェア・キャッシュの管理方法を流用する考え方が出てくる．ここで問題は，以下に示すハードウェア・キャッシュの管理と動的仮想ページ配置との違いである．

1) ハードウェア・キャッシュではコヒーレンス制御をハードウェアで行っていることが多いのに対し（最近では，ソフトウェア制御の研究もある），動的仮想ページ配置では，通常ハードウェアがサポートしておらず，ソフトウェアで制御，実現しなければならないこと．

2) 複製または移動などの制御を行う単位の大きさが，ハードウェア・キャッシュではキャッシュ・ブロック（大きさは十数バイトから百バイト程度）と比較的小さいのに対し，動的仮想ページ配置では，ページ単位にしか制御できず，ページの大きさが通常数Kバイトと大きいこと．

1)で述べたソフトウェアによる実現は，オーバヘッドになる．動的ページ配置では，2)の問題が大きな影響を与える．まず，ページ単位の複製，移動は大きなコストがかかること．さらに，制御単位がページ単位と粗いのでプログラムの仮想アドレス空間へのマッピング方法が性能に大きな影響を与える．すなわち，データの共有／非共有，さらには共有する場合，読出し専用データか読み書きデータ（さらには，それらの頻度なども必要になり得る）などのデータアクセス属性によってマッピングを決める必要がある．並列化言語処理系は，データアクセス属性単位で仮想ページへマッピングしないと性能が劣化する．たとえば，一つの仮想アドレス空間内のスレッドa，bと，三つのデータ・オブジェクトA，B，Cを考える．データ・オブジェクトA，Bにはそれぞれaのみ，bのみがアクセスし，a，bで共有しないとする（**図-6** 参照）．データ・オブジェクトCはa，bで共有し，読み書きするとする．また，A，B，Cの大きさの合計は仮想ページの大きさ以下とする．このとき，処理系がA，B，Cを一つの仮想ページにマッピングすると，本来共有されないA，Bまでもが共有のための制御の対象となって，ローカル・メモリ間を移動する（または複製される）ことになり，効率が悪くなる（**図-6**(a) 参照）．このとき，処理系は，少なくともA，BとCは異なる仮想ページにマッピングする必要がある（**図-6**(b) 参照）．コンパイル時に静的に分かるデータアクセス属性はマッピングに少なくとも反映させる必要がある．

この問題は，共有メモリ型マルチプロセッサのハードウェア・キャッシュ内のキャッシュ・ブロック内でも生じる問題であるが，NUMA 型マルチプロセッサではページ単位でしか制御できないので，問題がさらに深刻となるものである．

動的ページ配置方法として，ここではローカル／グローバル・メモリ・アーキテクチャ上で提案されている簡単な方法を紹介する[29)]．ここで，ローカル・メモリは，対応するプロセッサだけがアクセスできる私的メモリ (private memory) と

して用いるとする．概略は，ハードウェア・キャッシュ・コヒーレンス問題において，キャッシュ—メモリ間コヒーレンス制御としてライトバック (write-back) 方式（データへの書込みアクセスは，キャッシュにのみ行い，必要となったときにメモリに書戻しにいく方式），キャッシュ—キャッシュ間コヒーレンス制御として書込み時無効化 (write-invalidate) 方式（キャッシュ内データへの書込みアクセスが生じると，当該データが他キャッシュにあれば，それを無効にする方式）を採用した方法である．このとき，ハードウェア・キャッシュ・コヒーレンス問題におけるハードウェア・キャッシュ，メモリを，本方式では，それぞれ，ローカル・メモリ，グローバル・メモリに対応させる．具体的には，以下である．

まず，仮想ページの状態として，以下の三つを定義する．

1) 読出し専用(Read-Only, RO と表記)：複数のローカル・メモリに複製が存在する可能性があり，かつデータの内容はローカル・メモリおよびグローバル・メモリ内にあるものがすべて一致している．

2) 局所書込み可能 (Local-Writable, LW と表記)：一つのローカル・メモリ内にしか存在せず，かつグローバル・メモリのそれと一致していない．

3) 大域書込み可能 (Global-Writable, GW と表記)：ローカル・メモリにはなく，グローバル・メモリにしか存在していない．

また，仮想ページをローカル・メモリにもってくるか (LOCAL)，グローバル・メモリにだけ置く (GLOBAL) かの方策を決定するモジュールがあり，ある方策に従って，LOCAL, GLOBAL いずれかを決定する．方策決定が与えられた場合，ページの処理を**表-2**(a)（読出しアクセスの場合），**表-2**(b)（書込みアクセスの場合）に示す．ここで，表項目の中で，最上段は，ローカル・メモリに現在ある仮想ページへの対処方法を示す．

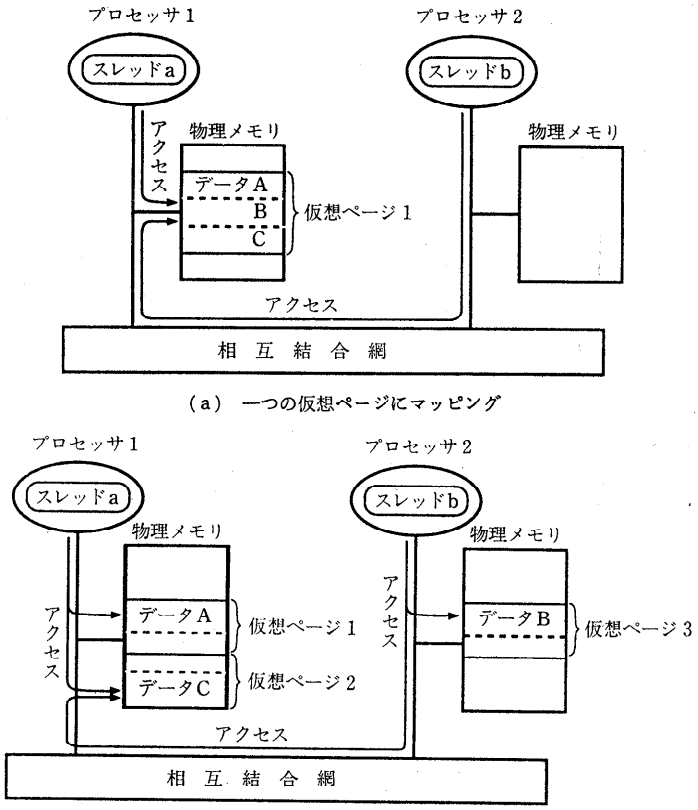


図-6 共有／非共有データの仮想ページへのマッピング

中段は，アクセスしたプロセッサのローカル・メモリにもってくるかどうかを示し，最下段は，遷移する仮想ページの状態を示す．また，sync は，ローカル・メモリ内の当該ページをグローバル・メモリへ書込むことを意味し，unmap は，仮想ページのローカル・メモリ内の物理ページへのマッピングを無効にすること，flush は unmap に加えて，当該物理ページを解放することを意味する．たとえば，LOCAL と決定したページ状態 LW をもつページへ読出しアクセスすると，当該ページが他のローカル・メモリに存在する場合，以下の処理となる．

- ローカル・メモリからグローバル・メモリへのページの書戻し (sync，ライトバック方式に対応) とマッピングの無効化とローカル・メモリ内の物理ページの解放 (flush) を行う．
- 自分のローカル・メモリへコピーする．
- 仮想ページ状態を読出し専用 (RO) 状態にする．

LOCAL, GLOBAL かの方策決定方法として，

表-2 処 理 方 法

(a) 読出しアクセス時

方策決定	仮想ページの状態			
	RO	GW	LW	
			自ノードにある場合	他ノードにある場合
LOCAL	ローカルへコピー	全てを unmap ローカルへコピー RO	何もせず	他のものを sync&flush ローカルへコピー RO
GLOBAL	全てを flush GW	何もせず	自分のものを sync&flush GW	他のものを sync&flush GW

(b) 書込みアクセス時

方策決定	仮想ページの状態			
	RO	GW	LW	
			自ノードにある場合	他ノードにある場合
LOCAL	他のものを flush ローカルへコピー LW	全てを unmap ローカルへコピー LW	何もせず	他のものを sync&flush ローカルへコピー LW
GLOBAL	全てを flush GW	何もせず	自分のものを sync&flush GW	他のものを sync&flush GW

文献 29) では，ローカル・メモリ間での仮想ページの移動回数を用い，ある値（4を用いている）以下であれば LOCAL，それを越えると GLOBAL の決定を下している．ただし，これでは，最終的にはグローバル・メモリ内に落ちつく（凍結）ことになり，凍結を解除させる方策が必要となる．

5. おわりに

従来の硬い OS では，ニーズとシーズからのますます多様化する並列処理環境を吸収できない．並列 OS は，これらをユーザ空間で吸収するための枠組みだけを提供する方向へと向かいつつあると考える．物理プロセッサだけでなく，物理メモリおよびその他のシステム資源に関する情報をユーザ空間に提供し，ユーザ空間と密接に連携することが，今後さらに必要になると考える．

紙面の都合上，並列 OS の実装技術，仮想プロセッサのスケジューリング方策，メモリ管理，OS レベルのコヒーレンス制御，同期機構，マイクロカーネル・アーキテクチャなどは割愛した．本稿が，並列処理に興味のある方々の一助となれば幸いである．

参 考 文 献

1) 情報処理学会：(特集) 超並列マシンとその応用，情報処理学会論文誌，Vol. 32, No. 4 (1991).
2) Kendall Square Research : Technical Summary, Kendall Square Reserach (1992).
3) 井田哲雄編：新しいプログラミング・パラダイム，共立 (1989)，および，井田哲雄，田中二郎編：続 新しいプログラミング・パラダイム，共立 (1990).
4) 通産省機械情報産業局編：リアルワールドコンピューティング・パラダイム，産調出版 (1992).
5) Karshmer, A. and Nehmer, J. (Eds.) : Operating Systems of the 90 s and Beyond, Lecture Notes in Computer Science 563, Springer-Verlag (1991).
6) 本多弘樹：自動並列化コンパイラ，情報処理，Vol. 34, No. 9, pp. 1150-1157 (Sep. 1993).
7) Proc. the USENIX Mach Symposium, USENIX Association (1991).
8) Black, D. L., Golub, D. B., Julin, D. P., Rashid, R. F., Draves, R. P., Dean, R. W., Forin, A., Barrera, J., Tokuda, H., Malan, G. and Bohman, D. : Microkernel Operating System Architecture and Mach, Proc. the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, pp. 11-30 (1992).
9) 乾 和志，菅原圭資：分散 OS Mach がわかる本，日刊工業新聞社 (1992).
10) Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., Herrman, F., Kaiser, C., Langlois, S., Leonard, P. and Neuhauser, W. : Overview of the Chorus Distributed Operating System, Proc. the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, pp. 39-69 (1992).
11) Thacker, C., Stewart, L. and Satterthwaite, E. : Firefly : A Multiprocessor Workstation, IEEE Trans. on Computer, Vol. 37, No. 8, pp. 909-920

(1988).
12) Scott, M. L., LeBlanc, T. J. and Marsh, B. D.: Design Rationale for Psyche, a General-Purpose Multiprocessor Operating System, Proc. the 1988 Int'l Conf. Parallel Processing, pp. 255-262 (1988).
13) LeBlanc, T. J., Marsh, B. D. and Scott, M. L.: Memory Management for Large-Scale NUMA Multiprocessors, Technical Report 311, Computer Science Dept., Univ. of Rochester (1989).
14) Scott, M. L., LeBlanc, T. J. and Marsh, B. D.: Evolution of an Operating System for Large-Scale Shared-Memory Multiprocessors, Technical Report 309, Computer Science Dept. Univ. of Rochester (1989).
15) Scott, M. L., LeBlanc, T. J., Marsh, D. D., Becker, T. G., Dubnicki, C., Markatos, E. P. and Smithline, N. G.: Implementation Issues for the Psyche Multiprocessor Operating System, Comput. Syst., Vol. 3, No. 1, pp. 101-138 (1990).
16) 鈴木, 清水, 山内: 共有記憶型並列システムの実際, 並列処理シリーズ 16, コロナ社 (1993).
17) 斉藤, 上脇, 山口: マルチスレッド実行環境に適した並列処理システムのメモリ管理方式, 情報処理学会論文誌, Vol. 32, No. 4, pp. 481-489 (1991).
18) 岡野, 横関, 並木, 高橋: 並列処理用 OS カーネル "Omicron V3" の開発とハイパ OS による共有メモリ型マルチプロセッサへの実装, 情報処理学会論文誌, Vol. 32, No. 5, pp. 673-683 (1991).
19) Hiroya, S., Momoi, T. and Nihei, K.: MUSTARD: A Multiprocessor UNIX for Embedded Real-Time Systems, Proc. of the Int'l Symp. on Shared Memory Multiprocessing, pp. 131-137 (1991).
20) 今村, 桑山, 宮崎, 林, 福田, 富田: 並列オペレーティング・システム K1 の設計と実現, 並列処理シンポジウム JSPP '92, pp. 305-312 (1992).
21) 平野, 田沼, 須崎, 浜崎, 塚本: 超並列システム用オペレーティングシステム「超流動 OS」の構想, 情報処理学会オペレーティング・システム研究報告, 93-OS-58-3 (1993).
22) Marsh, B. D., Scott, M. L., LeBlanc, T. J. and Markatos, E. P.: First-Class User-Level Threads, Proc. 13th ACM Symp. on Operating System Principles, pp. 110-121 (1991).
23) Bershad, B. N., Lazowska, E. D., Levy, H. M. and Wagner, D. B.: An Open Environment for Building Parallel Programming, Proc. 1st ACM Conf. Parallel Programming: Experience with Applications, Languages and Systems, pp. 1-9 (1988).
24) Doeppner, T. W.: Threads: A System for the Support of Concurrent Programming, Technical Report CS-87-11, Department of Computer Science, Brown Univ. (1987).
25) Sun Microsystems, Inc.: Lightweight Processes, SunOS Programming Utilities and Libraries, 800-3847-10 (1990).
26) Weiser, M., Demers, A. and Hauser, C.: The Portable Common Runtime Approach to Interoperability, Proc. 12th ACM Symp. Operating Systems Principles, pp. 114-122 (1989).
27) 白川洋充: ユーザレベルのプロセス管理はオペレーティングシステムを越えられるか?, 情報処理学会計算機アーキテクチャ研究会, 99-4 (1993). (豊富な参考文献も参照されたい).
28) Anderson, T. E., Bershad, B. N., Lazowska, E. D. and Levy, H. M.: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, Proc. 13th ACM Symp. on Operating System Principles, pp. 95-109 (1991).
29) Bolosky, W. J., Fitzgerald, R. P. and Scott, M. L.: Simple But Effective Techniques for NUMA Memory Management, Proc. 12th ACM Symp. on Operating Systems Principles, pp. 19-31 (1989).
30) LaRowe, R. P., Ellis, C. S. and Kaplan, L. S.: The Robustness of NUMA Memory Management, Proc. 13th ACM Symp. on Operating Systems Principles, pp. 137-151 (1991). (巻末の参考文献も参照されたい).

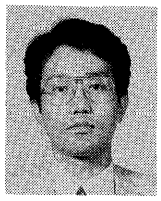
(平成5年7月28日受付)

用 語 解 説

外部ページャ: ページ不在によるページフォールトが生じると, 当該仮想ページが実在する場所 (通常は2次記憶でディスクが用いられる) から当該仮想ページを物理メモリにもってこなければならない. この処理を行うプログラムをページャと呼ぶ. 通常ページャはカーネル空間内で実現され, 仮想ページが実在する場所がカーネルによって決められており, ユーザが定義できない. 外部ページャとは, ユーザ空間で実現するページャのことであり, ユーザが仮想ページが実在する場所を定義でき, 柔軟性がある.

コンテクスト: 広義の意味では, プロセッサで命令を実行しているときの状態を完全に定義するに十分な情報を指す. この情報には, 少なくともプログラム・カウンタ, レジスタの内容, 仮想アドレス空間, ファイル資源の割当情報が必要である. 本論文では, 狭義の意味で用いており, プログラム・カウンタとレジスタの内容を指す.

ページフォールト: 仮想メモリの管理方法の一つであるページングシステムでは, 仮想アドレス空間を構成する固定長の仮想ページを物理メモリ内の固定長の物理ページへマッピングするマッピングテーブル (ページテーブル) が必要となる. 仮想アドレスから物理アドレスへのアドレス変換はハードウェアが行う. このアドレス変換する際, ハードウェアが認めていないアクセスによる割込みのことをページフォールトとよぶ. これには, 仮想ページが物理ページにマッピングされていない (ページ不在), 読出ししか許されていない仮想ページに書込みを行おうとした (アクセス保護例外), などがある.



福田　晃（正会員）

1954年生．1977年九州大学工学部情報工学科卒業．1979年同大学院修士課程修了．同年NTT研究所入所．1983年九州大学大学院総合理工学研究科情報システム学専攻助手．1989年同大学助教授．現在，奈良先端科学技術大学院大学と併任．工学博士．オペレーティング・システム，並列処理，計算機システムの性能評価などに興味をもつ．情報処理学会平成2年度研究賞受賞．訳書「オペレーティングシステムの概念」(共訳，培風館)．ACM, IEEE Computer Society, 電子情報通信学会，日本 OR 学会各会員．