勉強会資料
「並列オペレーティングシステム」

青木 秀貴

平成7年5月12日

1 はじめに

分散OSに関する本は比較的多く出回っているのだが、並列OSはまだそれ程研究の進んでいない分野なのか、並列OSの概要を述べたような手頃な資料はまだないようである。

そこで今回は、並列OSに関するいくつかの論文から、具体的なシステムや、並列OS研究における問題点等を見てみる。不十分な点も多いと思われるが、御容赦願いたい。

2 COS

COSは超並列コンピュータ用のOSとして研究開発されており、JUMP-1を最初のターゲットとして設計されている[1, 2]。

2.1 従来のOSの超並列マシンへの適用の問題点

初期の並列マシンでは、事実上一人のユーザの単一のアプリケーションがもっぱらマシンを占有し、メッセージ送受信などのハードウェア資源を直接操作していた。これは確かにユーザにマシンの速度の限界を追及する可能性を与えはするものの、ソフトウェアシステムやソフトウェア工学的な観点からは極めて原始的な状態であり、マルチタスク/マルチユーザ化など、大規模化や本格的な商用化には必然となる構成には、ハードウェア的にもソフトウェア的にも対応できていなかった。

近年の商用MPPでは、少数人数による大規模数値計算主体の利用法から、より多くのユーザを主体とした汎用計算機や汎用計算・資源サーバの側面を主体とした利用法に重点が移りつつある。このためには、従来のOSの資源管理や資源保護を、MPP全体に拡張した現代的なマルチタスク/マルチユーザOSが必須となる。この実現のためによく取られる手段は、MPPの各要素プロセッサ、あるいは要素プロセッサを制御するコントロールプロセッサに、Machなどのマルチプロセッサをサポートするマイクロカーネル技術を土台とする高機能なSymmetrical OS(特にMPP用に拡張したUNIX(OSF-1など))を実装する手法である。MPPへ対応するための拡張としては、低位の拡張としては、異なるノードにおけるタスクやスレッド生成があり、また、高レベルの機能としては、パーティショニングなどのノード分割によるユーザの保護ドメインの設定や、ギャングスケジューリングのサポートなどが挙げられる。

しかし、従来のMachのようなネットワーク結合されたUMAアーキテクチャを前提としたマルチプロセッサ用のSymmetrical OS拡張では、次のような本質的な問題がある。

1. マルチユーザ/マルチタスクの実現の為のハードウェア資源の抽象化のレベルが高すぎて、性能が低下すること。つまり「保護と性能の両立」が難しいこと。

2. 複数の異質 (heterogeneous) なユーザアプリケーションが実行されるときのアプリケーションとシステムの協調した資源管理のパラダイムがなく、結果として複数のアプリケーション間での効果的な資源の割り当てなどが難しいこと。
3. さらに、システムの構成や実行状況をアプリケーションに伝える統一的/標準的な手法が確立されていないので、MPP からネットワーククラスタまで様々なプラットフォームに自動的に適用 (adapt) する統一的な枠組みがなく、結果として性能まで含んだ可搬性 (portability) を得るのが必ずしも上手く行かないこと。
4. 逐次アルゴリズム/プログラムからのインクリメンタルな並列化のサポートが弱いこと。

2.2 COS の設計方針

OS の役割は、大型コンピュータから個人用コンピュータに至るまで基本的には同じであるが、超並列コンピュータという特殊性は設計時には当然考慮すべき主要な要因となっている。

汎用並列計算機における OS では、並列システムの存在意義である高速なプロセス実行を損なわないようにしなければならない。並列計算機上でシングルタスク環境と同等の性能を保ちつつマルチタスク環境を構築するためには、通常の意味での保護である「他のタスクからの機能上の干渉の排除 (機能の保護)」に加えて、次の 2 つが必要である。

- 性能の保護
  他のタスクからの性能上の干渉の排除。プロセッサ、メモリ、ネットワークなどのハードウェア資源を他のタスクと同時に使用した場合でも、他のタスクが発生するハードウェア資源への負荷によって、自分のタスクの性能が影響をできるだけ受けないようにすることを指す。
- ハードウェア資源の開放
  高速計算をおこなうためには、ハードウェア資源を開放する必要がある。ハードウェア資源の開放とは、多くの並列計算機にみられる通信や同期のためのハードウェアサポートを直接ユーザタスクに提供することを指す。

実際にはこれらは本質的に相反する要求であり、これまで十分な解決が得られていなかった。例えば現在の並列マシン用 OS は、シングルタスク環境のみを提供するライブラリ群に近いものか、従来の分散 OS の延長としての並列 OS が一般的である。シングルタスク環境のみの OS は、ハードウェア資源の開放のためにマルチタスクを犠牲にしているといえる。一方分散 OS 用のマイクロカーネルを各要素プロセッサ (PE) 上で実行するソフトウェアアーキテクチャの決定的な問題点は、PE 同士の通信や同期がカーネルによる実行時保護を毎回通過するために、非常に大きなオーバヘッドが生ずることである。

COS の設計方針は、以下の通りである。

1. エンドユーザの要求への柔軟な対応
   エンドユーザがその応用ソフトウェアを効率良く実行できるように自前の資源管理のポリシーなどをカスタマイズできるようにする。OS は、ユーザの要求に応じて、程良いサービスの提供をすることを目指す。これによって、「ユーザの処理のポリシーのシステムへの反映・組み込み」や「システムの提供するアブストラクションレベルに対して異なる要求を持つユーザの共存」が可能となる。
2. アーキテクチャ独立の設計
   並列コンピュータシステムのアーキテクチャに依存しないような機能・構成を採る。COS は、あくまでも汎用の超並列用 OS を目指す。

3. 分散共有メモリの支援
JUMP-1で提唱されている効率のよい細粒度並列処理のための分散共有メモリ管理機構を最大限に生かしたOSにする。これは、前項に矛盾するように見えるが、分散共有メモリは今後のコンピュータシステム全般にとって最重要な課題であり、JUMP-1だけに依存したものではない。これにより、強力で効率のよい保護機構の実現、一様なプログラミングモデルの採用による応用プログラムの書き易さの向上など、さまざまな効用をもたらす。

4. 徹底したモジュール構成による、柔軟な実現
COSの構成が自由に変更でき実証実験を繰り返し出来るようにするための構成方法を採る。そのためには、マイクロカーネルを基本として使うなど、モジュール化・ビルディングブロック化を徹底的にはかる。

5. マルチユーザの実行環境を提供
より具体的には、「性能と保護が両立されたマルチユーザ・マルチタスク処理の提供」といえる。

2.3 COSソフトウェアアーキテクチャ

超並列における各タスクの異なる要求を聞き入れ、かつ適当な資源配分や制御をおこなうことにより、超並列処理においても従来のワークステーションのような使い心地を提供することが、COSの役割である。

COSは、ユーザの専用資源空間として利用出来るパーティションを管理するPOS(Partition OS)と、さまざなまサービスを提供するSOS(Service OS)とから構成される。

POSでは、与えられた超並列システムの資源を効率良く複数のユーザに提供する機能を実現する。ここでは、COSにおけるハードウェア資源の管理を直接実施し、ハードウェア資源の集合の基本単位であるパーティションの管理をおこなう。エンドユーザは、強力に保護されたパーティションを確保し、その応用プログラムを実行していく。POSは以下の構成要素からなる。

- MMK(Micro Meta Kernel)
超並列システムの各プロセッサに常駐し、ハードウェアの基本機能の制御をおこなう。

- CLM(Cluster Manager)
ハードウェアの実装単位であるクラスタに一つ存在し、クラスタに特有のハードウェアの制御管理をおこなう。具体的には、クラスタ内のアドレス管理、クラスタ内のPEの管理、クラスタ間通信、スケジューラの管理、入出力制御などの機能を持つ。

- PTM(Partition Manager)
一つのパーティションに一つ存在し、パーティション全体に関する管理、他のパーティションとの通信管理、パーティションの動的拡大縮小の管理などをおこなう。具体的には、パーティション内の資源の管理、他パーティションとの通信、パーティションの生成/消滅、システムで一意なネットアドレスとクラスタアドレスの対応の管理などの機能を持つ。

一方、SOSは応用プログラムの起動などのユーザ操作のインタフェース、あるいは応用プログラムを記述するために必要なインタフェース(例えばPVM, MPIのようなAPIなど)を提供する。

エンドユーザは、まず標準的なインタフェースを使って応用プログラムを開発していく。その中で、問題に応じて自前でカスタマイズすべき箇所を見い出し、そのようなプログラムを開発するとともに、カスタマイズが可能となるような資源管理に関する方式を自前のポリシーモジュールの開発とその組み込みによって実現していく。この応用ソフトウェア開発/実行プロセスが、他のユーザとの相互干渉なく実現できれば、ユーザは充分超並列コンピュータを効率よく使いこなすことが出来る。

2.4 プログラミングモデル

COS は、JUMP-1 が分散共有メモリマシンであることを意識して、分散共有メモリを中心としたプログラミングモデルを提供する。タスクは 1 つの仮想アドレス空間 (または仮想空間) と複数のスレッドを持つ。タスクは JUMP-1 のクラスタ構成の制約は受けない。例えば、1 つのタスクが複数のクラスタにまたがって存在することができ、また複数のタスクが 1 つのクラスタ内で同時実行されうる。タスクの仮想空間は、他のタスクと共有可能な複数のメモリオブジェクトからなっている。メモリオブジェクトは、タスクへの貼り付け操作 (マップ) によって、内容の参照および変更が可能となる。COS のメモリオブジェクトは、単にメモリ共有をおこなうのみならず、通信・同期機構を提供する点で、従来 OS のものとは大きく異なっている。

2.5 キーワードなど

2.5.1 単一ネットワークアドレス

COS マイクロカーネル (COS-MK) では、JUMP-1 において分散共有メモリを実現するためにハードウェアサポートされたシステムワイドなアドレス変換機構を利用して、メモリ、通信、同期機構の統一的かつ高速な保護を実現する。COS-MK では、マシン全体で統一された 64 ビットの単一ネットワークアドレスにより、全てのリソース管理とケーパビリティを主体とした保護機構が統括される。これは COS-MK が提供する最も基本的な保護機構であり、全てのマシンのリソース (メモリ空間のみならず、スレッド、タスク、パーティション、デバイス等) がこの名前空間にマッピングされる。ネットアドレスのケーパビリティはユーザは直接扱えないので、暗号化などの手段を施さなくても、PE と MBP のアドレス変換機能によりケーパビリティチェックがおこなえ、かつ、絶対に詐称できないものとなっている。

2.5.2 ネットワークコンテキスト

パーティション内のスケジューリングについては、PTM 自身のポリシーによって変わるが、従来の概念であるプロセッサコンテキストの他にネットワークコンテキストの概念を導入する必要がある。タスクスイッチ時のネットワーク上に浮遊するパケットの取り扱い方であるが、近くのノードに退避させる方法と全てのパケットが到達するまで待つ方法が考えられるが、JUMP-1 では後者の方式を用いても性能が落ちないと考えられるため後者の方式を採用している。

2.6 今後の研究課題

上位レベルでのアプリケーションとの協調プロトコルや、システム全体の大域的な資源の管理ポリシーはまだ全ては定まっておらず、今後の研究課題となっている。現在の課題は、次の通りである。

- メモリオブジェクトの管理ポリシー
  COS では様々な高機能メモリが提供され、かつ、それらが異なるアプリケーションで共有される可能性があるため、あるタスクの仮想空間をどのように各ノード上に配置するか、そこでの異なるタスク間でどのように共有するか、共有メモリのホームページをどこに設定するか、などのポリシーが定められる必要がある。

- ページング時などの仮想記憶の管理ポリシー/実記憶の管理ポリシー
  COS では、通常の OS と比較して、主に二つの新たな仮想記憶に対する要求/状況を考慮する必要がある。

1. COS におけるページングは、クラスタ間の物理ページ使用量およびネットワーク使用量を平均化するためのネットワークページングと、システム全体のページ使用量を制御するためのディスクページング (従来の OS におけるページング) の両方がサポートされる。ページングをおこなう際に、ネットワークの高速性を生かし、直接二次記憶にページアウトせず、一旦他のノードの空きメモリ領域にスワップするネットワークページングをおこなう予定であるが、その管理ポリシーはまだ未確認である。
2. スケジューリングとの連携を考慮しなければならない。並列アプリケーションとその複数の組み合わせによって、スケジューリングのポリシーはかなり異なる。スケジューリングと記憶管理は密接な関係にあり、その効果的な連携のポリシーは研究課題である。

- スケジューリング
  アプリケーションの性質により、様々のスケジューリングの形態が考えられる。ロングターム、ミディアムターム、そしてショートタームのスケジューングは、自由度の高いポリシーが提供される必要がある。さらに、上記のメモリ管理との連携とあいまって、アプリケーションの実行状況を適切に反映するメカニズムが必要である。

- 自動粒度調整
  COS における実行環境は、JUMP-1 のような密結合の高性能 MPP から、ワークステーションクラスタまで様々であり得る。アプリケーションの適切な粒度は、以下の要因の現状により様々に変化する。

  1. アーキテクチャの特質: 例えばノード数, メモリの量, ネットワークのバンド幅/レーテンシなど
  2. アプリケーションの性質: SPMD/MIMD, 通信量, 通信の頻度, 同期通信/非同期通信主体など
  3. アプリケーション間の関係: アプリケーション間の連携の度合い, それぞれのアプリケーションの優先度, スケジューリングの形態, それぞれの資源の確保/競合など

  これらの要因に対して、COS はある程度の粒度調整の支援をアプリケーションに対して提供するのが望ましいが、その効果的なインタフェースは研究課題である。

- メモリマップ I/O
  COS は全ての資源を単一アドレス空間で管理するため、I/O の基本はメモリマップドなものになる。例えば JUMP-1 では、ディスクやフレームバッファは完全にユーザ空間にメモリマップされ、MBP を経由して異なるノードに接続されている I/O 資源に対して直接のユーザレベルアクセスが可能である。しかし、これらはシステムのサポートが必要であり、その管理のポリシーを研究する必要がある。

3 超流動 OS

超流動 OS は、大規模な汎用超並列システムに向けて開発中の OS である [3, 4]。超並列システムではネットワークトポロジへの依存性が高く規則的な動作をするデータパラレルのプログラムや、規則性が低いコントロールパラレルのプログラムといったさまざまなパラダイムのプログラムが混在して実行される。超流動 OS の目標はこの超並列システム環境下で、非常に多数の PE やプロセスを効率良く管理することである。

3.1 大域的仮想仮想記憶 (GVVM)

3.1.1 背景

仮想記憶 (Virtual Memory) の目的は、プログラムに実記憶よりも大きな仮想記憶空間を提供し、実記憶の大きさを越える問題を解くことである。たとえ大量のメモリを備え広大な実記憶が使用可能な超並列システムが登場しても、より大きな問題を解きたいという利用者の願いは変わらないであろう。超並列システムにおいては、多数のPEに対して見合うだけのバンド幅を備える二次記憶を用意することは困難である。1PEあたりの二次記憶バンド幅は逐次マシンの場合よりも狭くなり、仮想記憶による性能低下は逐次マシンの場合よりも著しくなる。また、プログラムの有するメモリ使用量の遍在性が原因で、少数のPEが多量のメモリを消費し、システム全体のメモリには余裕があるにもかかわらず、それらのPEでページングが頻発することもある。同期をとりながら動作する多くの並列プログラムでは、ページングが起きているPEを他のPEが待たなければならないため、実行性能が著しく低下する。同時にPEの使用率も低くなる。これらの問題のため、現在の仮想記憶機構とは異なる超並列システムのための記憶管理機構が必要である。

そこで、超流動OSの仮想記憶管理方式として、大域的仮想仮想記憶 (Global Virtual Virtual Memory, GVVM) が提案されている。GVVMは、多数のプログラムが混在して動作する分散メモリアーキテクチャ、および分散共有メモリアーキテクチャの超並列システム上での実行時間の短縮を目的として、PE空間全体でのページ使用頻度に基づくデマンドページング、および、他PE上のメモリ負荷の低いメモリページをスワップ領域として用いることにより、メモリの自動的な負荷分散をおこなう。

3.1.2 GVVMの基本概念

複数のプログラムが混在して動作するOSのもとでは、PE内でのメモリアクセスの局所性と伴に、複数のプロセスのメモリ使用量の差により生ずるPE空間内でのメモリ使用量の偏りも期待できる。そこで、GVVMは、

- 二次記憶装置へのページアウトの対象となるページを、システム全体のPEのメモリを対象にして決定する大域的仮想記憶 (Global Virtual Memory)
- 他PE上に未使用のページがある場合、そのページをスワップ領域として用い仮想的にページアウトをおこなう仮想仮想記憶 (Virtual Virtual Memory)

を融合し、ネットワークの高速性を生かした高速な仮想記憶機構とメモリの自動負荷分散を実現する。

GVVMでは、仮想仮想記憶において他PE上のスワップ領域として使用可能なページを選択する際、大域的仮想記憶の概念を用いる。図1は、GVVMの概念を図示したものである。四角形の箱はページを表し、色の濃さはそのページの使用頻度を表している。PE[i] においてページフォルトが発生したとする。その際、PE[i] 上で未使用の実記憶がなかった場合、最も使用頻度の低いページをページアウトし、ページフォルトが発生したページを読み込むための領域を確保しなければならない。そこで、PE[i] 上で最も使用頻度の低いページ (図中では左下のページ) をPE[i] から排除するのであるが、実際に二次記憶装置へページアウトするのはシステム全体で最も使用頻度の低いページとする。例えば、図ではPE[d] の左上のページが該当する。このページを二次記憶装置へページアウトし、空いた領域をスワップ領域としてPE[i] のページを仮想的にページアウトする。スワップ領域を提供するPE[d] をスワップ先PE、PE[i] をPE[d] へページアウトすることを仮想ページアウトと呼ぶ。PE[i] でページフォルトが発生した場合、PE[d] から仮想ページインをおこなう。

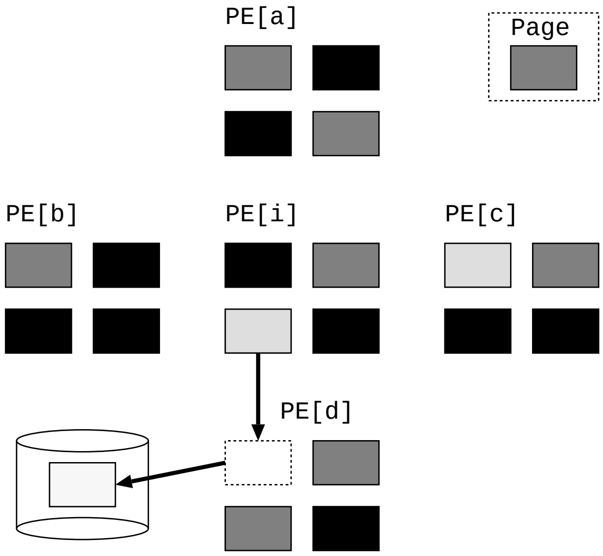

図 1: 大域的仮想仮想記憶の概念

3.1.3 GVVM によって得られるもの

アプリケーションプログラマのプログラマの視点から見ると、メモリはGVVMによって仮想化され、実記憶以上の大きさの空間が使用可能になると共に、プログラム毎に異なった分布を有するメモリの使用量という負荷が、OSによって自動的に分散されることになる。一方、ネットワークには計算以外のトラフィックが生ずるため、プログラマはネットワークを完全に管理下におくことはできない。このため、RWC-1のようにI/O専用ネットワークを備えることが望ましい。

仮想記憶は便利ではあるが、プログラムの書き方いかんではプログラムの実行を大幅に遅らせる可能性がある。従って、超並列マシンで仮想記憶が実用になるかという疑問がある。他PEへの仮想的なページアウトをできる限りおこなって、時間のかかる二次記憶へのページアウトを減らすことにより、プログラムの性能低下を抑えることが重要である。同様なPE間のページの移動は分散共有メモリでもおこなっている。分散共有メモリはプログラムが必要としているページを移動するのに対し、GVVMはプログラムが不要なページを移動する。

3.2 管理情報共有機構 (MetaShare)

超並列システムを管理する上で、以下の点が問題となる。

1. 計算機資源やアクティビティを管理するには、システムの状態を適切に把握することが必要である。超並列システムでは膨大な数のPEがネットワークにより接続された構成をとる。このためシステムの状況、例えば個々のPEのCPUの負荷、メモリの使用状況、ネットワークの負荷などを瞬時に正確に把握することは非常に困難な問題である。

2. OSは通常、実行制御、記憶管理などのサブシステムはそれぞれ目的に応じてシステムの状況の把握をおこなうが、そのベースとなる情報は、例えばCPU負荷やメモリ負荷、ネットワーク負荷といった共通するものが多い。このような状況は各々のサブシステムの機能の重複であると共に、個別に同一情報を収集することにより例えばネットワークトラフィックなどの増大を招く。

これらの問題を解決するために、管理情報共有機構 (MetaShare) を提案している。MetaShare は OS のサブシステムで共通して使用されるメモリ負荷、CPU 負荷、ネットワーク負荷といったシステムの管理情報を収集し、統合的に管理し、必要とする各サブシステムに配布するシステムである。

GVVM においてページアウト要求が生じた時、スワップアウト先となる PE を捜す必要がある。このスワップアウト先 PE の探索に MetaShare が使用される。

4 スケジューリングについて

4.1 時分割方式と空間分割方式

マルチプログラミング環境下のマルチプロセッサのスケジューリング方式は、時分割方式 (time-multiplexing) と空間分割方式 (space-multiplexing) の 2 つに大別される。

時分割方式は、同一プロセス内の全スレッド (並列実行可能な単位) をプロセッサ群に同時に割り当てて、実行し、タイムスライスを使いきると、次のプロセスを実行する方式である。代表的なものとして、コスケジューリングが挙げられる。時分割方式は、いくつかの利点がある反面、スケジューラボトルネックやキャッシュ活用上の問題点があり、大規模マルチプロセッサには適さないと考えられる。

一方、空間分割方式は、プロセッサをグループ化し、グループ単位にプロセスを割り当ててスケジューリングする方式であり、大規模マルチプロセッサのスケジューリング方式として有望な方式である。

4.2 パーティショニング

空間分割方式においてプロセスに割り付けられた PE 領域を、パーティションと呼ぶ。「閉パーティション」とは、パーティション外部を通らずにメッセージをルーティングできるパーティションである。

パーティショニングアルゴリズムは、投入されるタスクが要求するプロセッサ数や形状に合わせて、FCFS(First Come First Served) で物理プロセッサに効率良く割り当てるアルゴリズムである。これによって複数のタスクを同時に実行できるようになり、また、全体のプロセッサのうち稼働しているプロセッサの比率 (プロセッサ利用率) を高めることができる。

しかし、パーティショニングアルゴリズムで処理してもプロセッサ空間の大部分を占有するタスクが到着すると、その前後のタスクを組み合わせてプロセッサ空間に割り当てることができない閉塞状態になる。このような状況ではプロセッサの利用率が低下し、効率的利用ができない。さらに、個々のタスクの応答性 (レスポンス) も閉塞状態のために悪化する。

メッシュ結合並列計算機を対象とするパーティショニングアルゴリズムには、Frame Slide, First Fit, Best Fit, Adaptive Scan, 2D Buddy などがある。

4.2.1 2D Buddy

2D Buddy(2DB) では、メッシュ結合しているプロセッサが 2 のべき乗の正方形であることを仮定している。2DB ではサブメッシュの形状を並列計算機より小さい 2 のべき乗の正方形とする。この正方形のサブメッシュは、メッシュ結合しているプロセッサを 4 分割 (縦方向 2 分割, 横方向 2 分割) を繰り返して得られる正方形の位置に割り当てられる。

上記の 2DB は Quad 2D Buddy とも呼ばれ、その他に次のようなバリエーションもある。

- Binary 2D Buddy without Process Rotation
  2 次元平面を等面積の矩形に 2 分割していく。

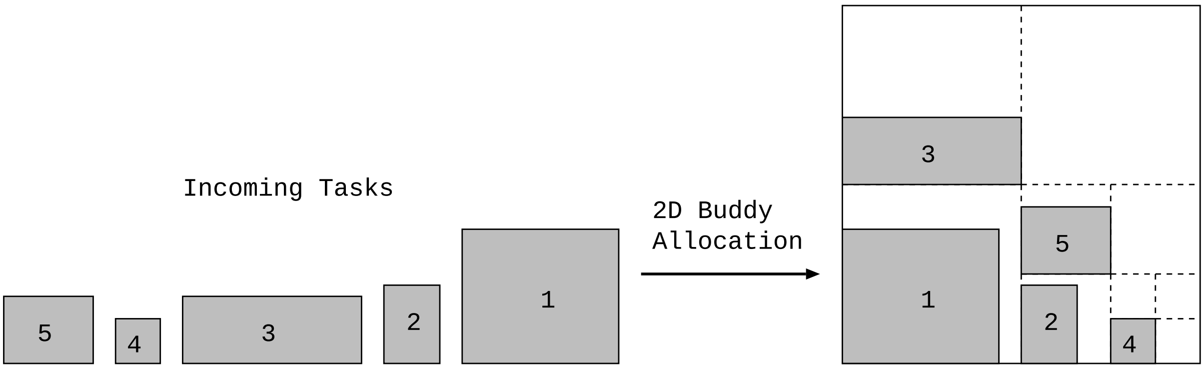

図 2: 2D Buddy によるサブメッシュの割り当て

- Binary 2D Buddy with Process Rotation
  2 次元平面を等面積の矩形に 2 分割していく。ただし、タスクの縦横を効果してもよいものとする。

2D Buddy によるサブメッシュの割り当ての例を、図 2 に示す。タスクの大きさは必ずしも 2 のべき乗の正方形ではないため、割り当てられたパーティション中に、タスクによって使用されないプロセッサが存在する。この領域を内部フラグメンテーションと呼ぶ。また、割り当てがおこなわれていない領域を外部フラグメンテーションと呼ぶ。

4.2.2 その他のアルゴリズム

2 のべき乗にとらわれずに 2 次元空間をパーティショニングするアルゴリズムには、First Fit, Adaptive Scan, Best Fit などがある。これらのアルゴリズムでは、一般的に閉パーティションとはならないが、内部フラグメンテーションは発生しない。

- First Fit
  最初に発見した投入可能な空間にタスクを投入する。

- Adaptive Scan
  最初に発見した投入可能な空間にタスクを投入する。ただし、タスクの投入ができなかった場合は、タスクの縦横を交換して再び試みる。Adaptive Scan によるサブメッシュの割り当ての例を、図 3 に示す。

- Best Fit
  フラグメンテーションが最小となる空間にタスクを投入する。

4.3 スケジューリングに関する最近の研究

4.3.1 2 レベルスケジューリング

2 レベルスケジューリングは空間分割方式の 1 つであり、NUMA(Non-Uniform Memory Access) マルチプロセッサを対象としたメモリ管理との協調動作などとあわせて研究が進められている [5]。
図 4 に示すように、システムは 2 レベルのスケジューラとメモリマネジャから構成される。

- スケジューラ
  スケジューラは、上位スケジューラ (グローバルスケジューラ) と下位スケジューラ (プライベートス

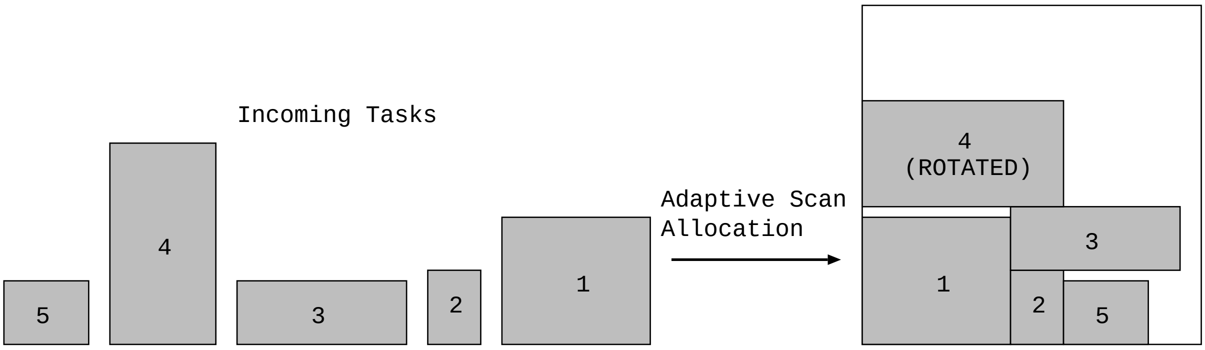

図 3: Adaptive Scan によるサブメッシュの割り当て

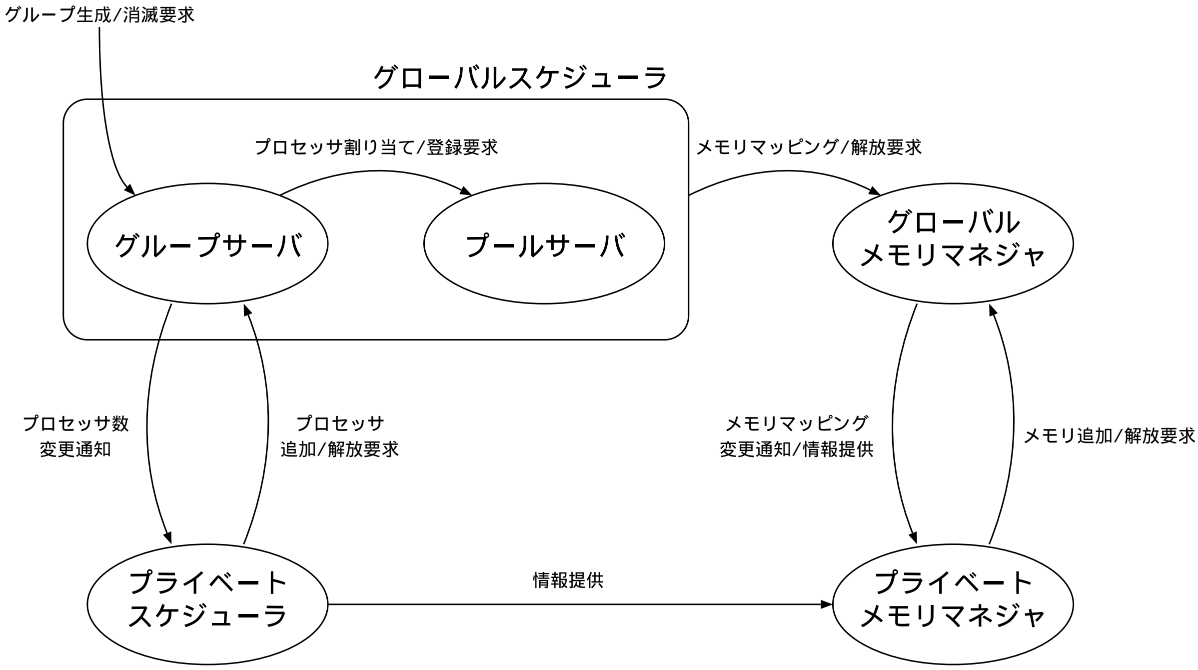

図 4: スケジューラとメモリマネジャ

ケジューラ) から構成される。グローバルスケジューラはさらに、プロセッサの動的グルーピングをおこなうグループサーバと、フリープロセッサ (どのグループにも属さないプロセッサ) を置くプロセッサプールの管理をおこなうプールサーバから構成される。各プロセスはグループサーバによって 1 つのプロセッサグループに割り当てられる。グローバルスケジューラはシステム内に一つ存在し、カーネル空間で動作する。プライベートスケジューラは、グループ内プロセッサを用いて対応するプロセス内のスレッドをスケジューリングし、対応するプロセス空間に存在する。

- メモリマネジャ
  スケジューラと同様に、グローバルメモリマネジャとプライベートメモリマネジャから構成される。グローバルメモリマネジャは、複数のプロセス空間の物理メモリへの割り当て/管理をおこなう。プライベートメモリマネジャは、プロセス対応に存在し、対応するプロセス空間の物理メモリへの割り当て/管理をおこなう。

このシステムでは、グループに対する動的なプロセッサ追加/解放が可能である。

- プロセッサ追加
  プライベートスケジューラからのプロセッサ追加要求により、プールサーバから割り当てられたプロセッサを該当グループに追加する。
- プロセッサ解放
  プライベートスケジューラからのプロセッサ解放要求により、該当するグループからプロセッサを解放する。

4.3.2 分散樹状待ち行列 (DQT)

並列マシン上での時分割スケジューリングの基本的アイデアは、Ousterhout により提案されている。しかしながらこの提案ではプロセス走行待ち行列を集中管理しているため、大規模システムではボトルネックを生じる可能性があると同時に、ネットワークトポロジーからくるパーティショニングの制約を反映していない。

時分割スケジューリングと動的パーティショニングを組み合わせた新しいスケジューリング方式として、分散樹状待ち行列 (Distributed Queue Tree, DQT) が提案されている [6]。DQT の特徴を以下に示す。

- DQT は並列マシンにおける横取り可能なスケジューリング方式である。
- プロセス走行待ち行列を分散させているため、ボトルネックを生じ難い。
- バッチスケジューリング方式に比べ、より高いプロセッサ利用率を得ることができる。
- 高負荷時においても DQT は安定かつ実用上十分な性能を発揮できる。

図 5 に DQT の例を、表 1 に図 5 の DQT における時分割スケジューリングの例を示す。表中 $Q_i(j)$ とあるのは、i 番目の DQT のノードの待ち行列中の j 番目のプロセスが走行していることを示す。

DQT は動的パーティションの入れ子構造を反映した木構造を成す。DQT ノードはパーティションに 1 対 1 で対応している。DQT ノードにはそのパーティションに割り当てられたプロセスの走行待ち行列を持つ (図 5 で、DQT ノードを示す丸の右側にあるのが待ち行列を表す)。負荷を分散させる意味から、DQT ノードは対応するパーティション内の適当なプロセッサに分散配置される。

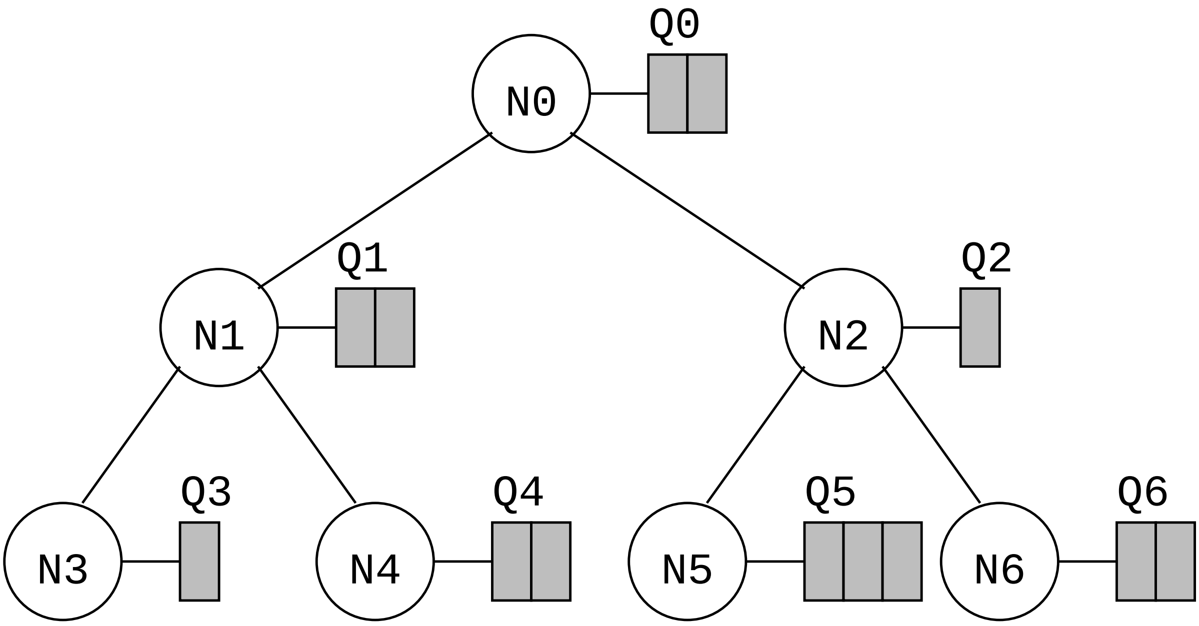

図 5: DQT の例

表 1: DQT スケジューリングの例

Time Slot	PE0	PE1	PE2	PE3
0	$Q_0(0)$			
1	$Q_0(1)$			
2	$Q_1(0)$		$Q_2(0)$	
3	$Q_1(1)$		$Q_5(0)$	$Q_6(0)$
4	$Q_3(0)$	$Q_4(0)$	$Q_5(1)$	$Q_6(1)$
5	$Q_3(0)$	$Q_4(1)$	$Q_5(2)$	$Q_6(0)$
6	$Q_0(0)$			
7	$Q_0(1)$			
8	$Q_1(0)$		$Q_2(0)$	
9	$Q_1(1)$		$Q_5(0)$	$Q_6(1)$
10	$Q_3(0)$	$Q_4(0)$	$Q_5(1)$	$Q_6(0)$
11	$Q_3(0)$	$Q_4(1)$	$Q_5(2)$	$Q_6(1)$
12	$Q_0(0)$			
⋮	⋮			

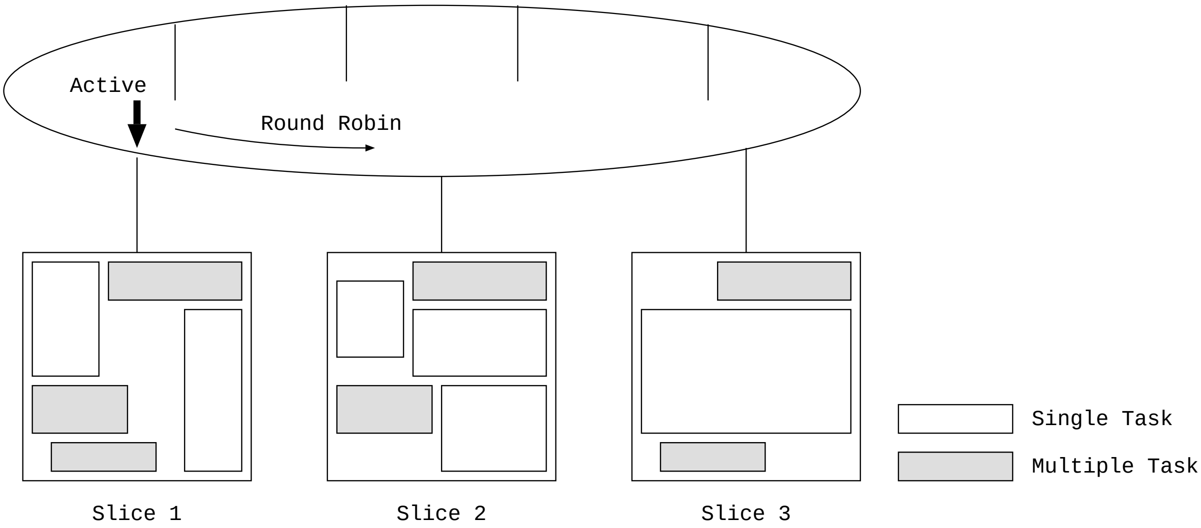

図 6: スライスに基づくタスクの割り当て

4.3.3 2D Buddy と Adaptive Scan の時分割処理への拡張

メッシュ結合並列計算機用のパーティショニングアルゴリズムとして提案されている 2D Buddy と Adaptive Scan を時分割処理 (TSS) に拡張する研究もなされている [7]。この TSS では仮想並列計算機を用意し、これらのパーティショニングアルゴリズムを仮想並列計算機上でのサブメッシュの割り当てに用いる。ある仮想並列計算機でタスクが占有している領域を別の仮想並列計算機上で未使用の場合、そのタスクは複数の仮想並列計算機に存在することができる。これにより、応答性とプロセッサの利用効率を上げることができる。

逐次計算機ではタスクに一定時間ごと計算機資源を割り当てていたが、本 TSS では実並列計算機と同一の構成をする仮想並列計算機を用意し、この仮想並列計算機を一定時間ごとに実並列計算機に割り当てる。この仮想並列計算機をスライスと呼ぶ。タスクはスライスにパーティショニングアルゴリズムに従って割り当てられ、スライス数は投入されるタスクの数によって変化する。スライスへの時間割り当てはラウンドロビンでおこなわれる。この状況を図 6 に示す。

並列計算機の TSS では、逐次計算機では問題にならなかったプロセッサ利用率の問題がある。一つのスライスにおいてはパーティショニングアルゴリズムによって利用率を上げることができる。本 TSS では更に、複数のスライスの同一領域にサブメッシュを割り当てることのできるタスクはプロセッサ利用率を上げるため、複数のスライスに存在する。一つのスライスにのみ存在するタスクを single、他のスライスに存在することができるタスクを multiple と呼ぶ。

参考文献

[1] 斎藤信男 他. “科研費重点領域 超並列 C 班—超並列システム用オペレーティングシステム COS の研究”. 文部省重点領域研究「超並列原理に基づく情報処理基本体系」第 6 回シンポジウム予稿集, 1995.

[2] 猪原茂和, 松岡 聡, 松本 尚. “分散共有記憶型超並列オペレーティングシステム COS マイクロカーネルの保護機構”, 並列処理シンポジウム JSPP’94, 1994.

[3] 平野 聡, 田沼 均, 須崎有康, 一杉裕志, 塚本享治. “大域的仮想仮想記憶 (GVVM) のマルチプロセス環

境での評価”, 並列処理シンポジウム JSPP’94, 1994.

[4] 平野 聡, 田沼 均, 一杉裕志, 須崎有康, 塚本享治. “大域的仮想仮想記憶 (GVVM) の RWC-1 上での性能予測”, 情報処理学会研究報告 94-OS-65-16, 1994.

[5] 甲斐久淳, 藤木亮介, 藤田 晃. “メモリ管理と協調動作する 2 レベルスケジューリング”, 並列処理シンポジウム JSPP’94, 1994.

[6] 堀 敦史, 石川 裕, 小中裕喜, 前田宗則, 友清孝志. “超並列マシンにおける時分割スケジューリング”, 情報処理学会研究報告 94-OS-65-5, 1994.

[7] 須崎有康, 田沼 均, 平野 聡, 一杉裕志, 塚本享治. “メッシュ結合並列計算機のパーティショニングアルゴリズム 2D Buddy または Adaptive Scan を使った時分割処理”, 情報処理学会研究報告 94-OS-65-18, 1994.