コンピュータコロニーにおける高速移送可能な並列アクティビティの実現

山添 博史[†] 鶴崎 剛大[††] 増田 峰義[†]
伊達 新哉[†] Damien Le Moal[†] 五島 正裕[†]
森 眞一郎[†] 富田 眞治[†]

分散共有メモリをベースとしたシステム、コンピュータ・コロニーにおいて、ミッションとユニットからなる並列アクティビティを採用する。ミッションはタスク-スレッドモデルにおけるタスク、ユニットはスレッドに相当するが、ユニットはそれぞれ固有の領域を持ち、不必要な共有を行わない。ユニットを移送する際には、同じミッションが存在するノードに移送し、少量の管理構造だけを高速に移送する。そして、移送先でユニットを動作させながら、固有のメモリを転送するので、移送によるユニットの動作の中断は小さくなる。このようにして、ユニットの移送コストを小さくすることで、システム全体の運用効率を高くできる。

Design of Quickly Migratable Parallel Activity on Computer Colony

Hiroshi Yamazoe,[†] Takehiro Tsurusaki,[††] Mineyoshi Masuda,[†]
Shin-ya Date,[†] Damien Le Moal,[†] Masahiro Goshima,[†]
Shin-ichiro Mori[†] and Shinji Tomita[†]

On computer colony based on distributed shared memory, we design the parallel activity model which includes unit and mission. Mission resembles task in task-thread model, unit resembles thread, but unit has its own memory area, cuts the cost for unnecessary memory sharing. On migration, a unit should be migrated to the node which has the same mission with the unit, then the structure to be migrated is small and migrated quickly. And the unit's own memory are migrated after the unit started execution on the destination node, so the unit stops execution during little time. In this way, the whole system is efficiently utilized by little cost for unit migration.

1. はじめに

本研究では、独立した計算機の集合から、1 つの高性能計算機としての性能を実現するコンピュータ・コロニーを開発している。ユーザはコンピュータ・コロニーが複数の計算機からできていることを意識せず、1 つの計算機としてログインし、どの端末からでも同じインターフェースで、同じ資源を利用できる。このような計算機の実現のため、SSI クラスタ[1)]のように、ライブラリや DSM などのさまざまな階層でのアプローチがなされている。コンピュータ・コロニーは、現在だけでなく将来の計算機環境も考慮に入れ、ハードウェアおよび OS から設計を行っている。

コンピュータ・コロニーでは、プロセッサとメモリおよび、複数の計算機を用いるときに必須であるネットワークインターフェースを基本要素とし、それらとその上で動作する OS である Colonia の集合によって、システムを構成する。この上で、複数のユーザそれぞれが並列処理やリアルタイムジョブなど、最大の処理性能を利用することができ、しかも特定のユーザの独占利用による応答性能の低下などを阻止しなければならない。

このため、システムはユーザの優先順位やジョブの性質などを考慮して自動的に負荷の配置を行うが、その際動的に負荷の移送を行わなくてはならない。移送のコストがじゅうぶん小さくなければ、動的負荷分散によるシステムの利用効率の最大化は困難である。本稿では、コンピュータ・コロニーの分散共有メモリの環境に適し、移送のコストを小さくできる並列アクティ

[†] 京都大学大学院情報学研究科通信情報システム専攻
Department of Communications and Computer Engineering, Graduate School of Infomatics, Kyoto University
[††] 京都大学工学部情報工学科
Department of Information Science, Faculty of Engineering, Kyoto University

ビティを導入し、その移送の際の実際の動作を示す。

以下、2章で我々の開発するコンピュータ·コロニーの概要、3章でその並列アクティビティであるミッション-ユニットモデルの概要、4章で移送を中心とした実装を説明し、5章で結論に導く。

2. コンピュータ·コロニー

この章では、コンピュータ·コロニーの目標および構成について述べ、それに対応するのに必要な並列アクティビティについて論じる。

2.1 コンピュータ·コロニーの目標

本研究では、並列処理による高速計算を実現し、かつ複数のユーザが気軽に自分のマシンのように自由に使える、価格性能比の高い計算機システムとして、コンピュータ·コロニーを提案する。colony は、複数の個体が集合して 1 個体のように生活する生物の状態すなわち群体を意味し、コンピュータ·コロニーは同様に、複数のマシンが集合して 1 つの大きな高性能マシンとなることを目標とする。コンピュータ·コロニーは分散共有メモリを支援するハードウェアと、分散 OS である Colonia から成り、次のような目標を持つ。

低価格性

コンピュータ·コロニーのハードウェアは、メモリ 1 つあたりに少数のプロセッサと、共有メモリ管理機能を持つネットワークインターフェースを基本単位として構成する。現在、マルチプロセッサとメモリを搭載したチップが実現されつつあり、近い将来、これにネットワーク機能を加えたモジュールカードをつくることが可能になることが考えられる。このモジュールカードが、従来の LAN における 1 つの要素マシンの中核であり、コンピュータ·コロニーのノードとなる。

家庭用あるいは職場での端末マシンは、筐体にこのモジュールカードを装着し、画面とキーボード、マウス、その他の周辺機器で構成する (図 1)。さらに処理性能が必要になった際には、古いマシンを捨てて新しいマシンを購入する必要はなく、単に新しく速いモジュールカードのみ購入して装着すれば、既存のモジュールカードも利用しつつ、倍以上の性能向上を見込める。この筐体の中は分散共有メモリ環境であり、基本的にモジュールカードの数を増やしても、集中共有メモリのようなボトルネックは生じない。

このような端末マシンを複数用意すれば、マルチユーザで利用できる巨大な並列計算機環境が構成できる。また、大きな筐体に多くのモジュールカードを装着することにより、同数プロセッサを持つ大型並列計算機よりはるかに低価格で購入が可能で、つまり同じ価格でより多くのプロセッサを持つ並列計算機が構成できる。

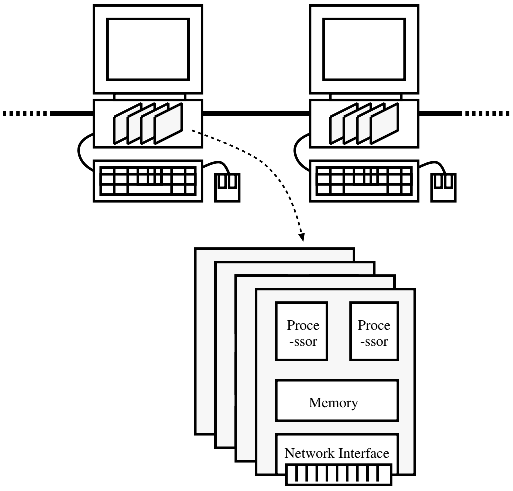

図 1 コンピュータ·コロニーとモジュールカード

本研究の目的は、このようなモジュールカードを構成するための、標準規格を確定することである。モジュールカードでの、分散共有メモリ管理を実現できるネットワークプロトコル、プロセッサアーキテクチャなどを標準化すれば、さまざまなメーカーが参入してモジュールカードの性能、価格競争に努め、大量生産による低価格化が期待できる。このようにすれば、現在パソコンにメモリを追加するような感覚で、気軽にプロセッサ数を倍増させることができるようになる。さらに、LAN 全体を計算サーバとして利用することができ、大型並列計算機も、現在よりはるかに安価に導入が可能になる。

処理性能の最大利用

ユーザがシステムコールなしに通信が行えるような、ハードウェア支援の共有メモリやメッセージ通信の機構を提供する。そのうえで、アクティビティをシステム全体に平均的に配置したり、動作中のアクティビティを移送することで、全体の処理性能を最大限に利用する。このため、分散共有メモリの環境に適応し、移送のコストが小さい並列アクティビティが必要である。

フェアネス、リアルタイム

マルチユーザ間の資源配分の調整を合理的に行う。各ユーザはシステムから “チケット” を与えられ、システムはそれに比例した資源配分を行う。つまり、プロセッサを占有する性質があるプログラムを多数投入することによる、特定のユーザによる独占は、チケットの配分によって抑制される。

また、応答性能を重視する会話型アプリケーションについては、プロセッサ利用時間が少ないかわりに、入力に対する応答の優先順位を上げる必要がある。さらに、個人的端末としての利用では、動画 1 コマあたりの計算量と時間制限など、リアルタイム性を重視するアプリケーションもある。コンピュータ·コロニーでは、これらにも対応するスケジューリングポリシーを併用する。

2.2　コンピュータ ·コロニーの構成

コンピュータ ·コロニーは、モジュールカードと分散 OS である Colonia から成る。ジョブを持ってきて利用するユーザに対して、ユーザに資源を提供する側を全体としてシステムと呼ぶ。1 つのメモリ、1 つのネットワークインターフェースと複数プロセッサからなるノードは、それぞれハードウェア的に独立性が高く、それに応じてソフトウェア的にも分散した構成をとる。ノードごとの独立性を高めることで、一部のノードが故障あるいは断線した場合での耐故障性やスケーラビリティを高める。以降、コンピュータ ·コロニーのシステムの内部構成について述べる。

分散共有メモリ

コンピュータ ·コロニーにおいて、ノードは独立した計算機として、それぞれローカルメモリを持つ。これらを Colonia およびハードウェアであるネットワークインターフェースによって、仮想アドレスでの分散共有メモリという形でユーザに提供する。各ノードごとにローカルメモリでの位置を指定する物理アドレスが定義されており、アクティビティそれぞれが持つ仮想アドレスとの対応を定義するローカルページテーブルがある。ローカルページテーブルが示すのはローカルメモリでの物理ページ番号であり、ノード間で共有できるアドレスではないので、ノード間で共有することはできない。

また、ノード間でページを共有する場合、そのページの管理責任を負うホームページがあり、それがないノードは、ローカルメモリ上にその内容をキャッシュする。これをコピーページという。Colonia はローカルメモリ上の物理ページの確保と、仮想ページごとのホームの位置を管理しており、この対応を定義するものを、グローバルページテーブルと呼ぶ。ネットワークインターフェースはグローバルページテーブルを参照して、ホームページのコピー内容をキャッシュライン単位で一貫性制御する。

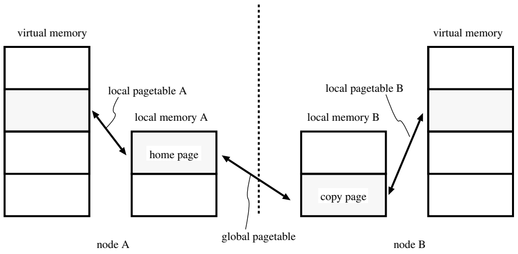

図 2　ローカルメモリと共有

ローカルページテーブルのほか、あるノードにしか存在しないアクティビティの構造体などの管理構造は、ノードごとに分散している。分散していても、互いに関連性があり一貫性保持の必要なものについては、明示的な操作が必要であり、システムのオーバヘッドの原因となる。これに対し、グローバルページテーブルなど、ノード間で完全に共有できる管理構造は、ノードごとのシステム領域の間の、低レベルな共有メモリ上で維持される。

システムスレッド

システムユニットの行う処理のうち、一定時間以内に終了することが保証されず、長く継続するようなものは、ユーザのアクティビティと同様に、プロセッサコンテキストを持つシステムスレッドが担当する。システムスレッドには、常にバックグラウンドで動作してシステムの管理、保守に当たっているものや、通常待機していて必要なときに特定の処理を行うものなどがある。

スケジューリング

スケジューリングは、ローカルスケジューラとグローバルスケジューラに分けて行う。ローカルスケジューラは、プロセッサごとにアクティビティ待ちキューを持つシステム内ルーチンで、自らはスケジューリングの対象にならない。アクティビティは、何もなければ常に同じプロセッサ上にスケジュールされる。グローバルスケジューラは、アクティビティがどのノードのどのプロセッサで動作するか、静的な配置および動的な移送を決定する。これはシステムスレッドであり、バックグラウンドで負荷情報などを他のノードと交換して、アクティビティの移送を指示したりする。

2.3　並列アクティビティモデル

コンピュータ ·コロニーでは、分散共有メモリという状況下で、並列アクティビティを複数のノードに分散させて動作させ、さらに動的に移送を行うことになる。これらのコストを、以下で検討していく。

従来共有メモリを用いた並列アクティビティモデルとして、タスク-スレッドモデルがある。これは、タスクごとに仮想メモリを持ち、その上で複数のスレッドが並列に動作するというものであるが、スレッドが利用するメモリ領域には、本質的に各スレッドごとに固有な領域と、全スレッドで共有する領域とがある。これらの特徴については、次のとおりである。

固有領域　スレッドごとに必要なスタックやヒープなどを置く。共有する全空間の中で、異なるアドレスとなるように選んで使用する。同期を必要としない。空間的に小さい領域に、高い頻度でアクセスする。

共有領域　コードや共有データなどを置く。同期が必要である。アクセス頻度は固有領域より小さい可能性があるが、空間的な広がりが大きい。

これらの領域に関して、OS が管理するべき方針は異なるはずである。共有領域については共有のための操作は不可欠だが、固有領域については不必要なためオーバヘッドとなる。また、共有領域は資源および管理構造を減量できる可能性があり、これは並列アクティビティの移送の際に大きく関わる。

この節では、コンピュータ・コロニーの環境において、タスク-スレッドモデルおよびUNIXのプロセスモデルを並列アクティビティとして採用した場合に、システムが費やすコストを以上のような観点で検討し、必要とされる並列アクティビティを導く。

タスク-スレッドモデル

タスクが固有の仮想メモリを持ち、その上で複数のスレッドが並列に動作するモデルである。タスクの持つ空間すべてを、全スレッドで共有している。メモリおよび管理構造が集中している環境では、共有の維持やスレッド移送に関するシステムの負荷はほとんどない。

しかし、コンピュータ・コロニーの環境では、分散した管理構造の一貫性保持が必要である。例えば、ローカルページテーブルはノードごとに分散しているが、あるノードでページを確保した場合、他のノードに対して、そのページを確保したことを何らかの形で通知しなければならない。このような処理は、共有領域については不可避であるが、固有領域については不要であるはずなので、このモデルでのオーバヘッドとなる。

また、スレッド生成の際には、固有領域の位置をスレッドごとに異なるところに決める必要があり、これにも分散した管理構造間の通信が必要である。

スレッドを移送するには、メモリ全体とその管理情報、スレッドの管理情報の移送が必要である。その際、スレッドの移送先に、同じタスクが動作しているノードを選べば、メモリとその管理構造の大部分を共有できるため、実際に移送するメモリは、ほとんど固有領域のみとなる。

プロセスモデル

独立した仮想メモリを持つプロセス間のメッセージ通信によって、並列処理を行うモデル。共有に関わる管理構造の一貫性保持は必要ない代わりに、管理構造の共有利用は一切できない。移送を行うときは、全メモリを移送する必要がある。

このように、タスク-スレッドモデルでは、固有領域をも共有領域として扱うことによって、分散共有メモリ環境でのオーバヘッドが大きくなる。これに対し、プロセスモデルでは、共有メモリプログラミングは基本的に行えないし、管理構造や資源の共有を行えないため、移送するべきデータ量が大きくなるが、分散されている管理構造のための通信は必要ない。

コンピュータ・コロニーの分散共有メモリ環境では、これらの特徴を考慮し、共有領域と固有領域に分けて管理する並列アクティビティが必要である。

3. ミッション-ユニットモデル

コンピュータ・コロニーでは、分散共有メモリ環境において維持および移送などのコストを比較し、並列アクティビティとしてタスク-スレッドモデルではなく、ミッションとユニットからなるミッション-ユニットモデルを導入する。ミッションはタスク-スレッドモデルにおけるタスク、ユニットはスレッドに近い存在である。

図3に、ミッション-ユニットモデルおよびユーザとシステムの概念図を示す。ユーザの代理としてプログラムを実行する実体はミッションである。1つのミッションには複数のユニットがあり、これが並列アクティビティとして複数のプロセッサに分散して動作する。ユニットは、プロセッサを割り当てる対象となり、それぞれプロセッサコンテキストを保持する。

また、ユニットは、それぞれ1つの仮想メモリを持つ。仮想メモリは、仮想ページの集合であり、このためユニットは管理構造としてローカルページテーブルなどを持つことになる。ユニットの仮想メモリ中には、ミッション内で共有するコードやデータを含むユニット共有領域と、ユニットが動作するのに必要なスタックやヒープを含むユニット固有領域がある。

ユーザと同様に、システムにもミッションとユニットがある。単にユニット、ミッションと表記した場合は、いずれもユーザのものを表すものとする。システムユニットはノードごとに1つずつ存在し、システムミッションはこれらすべての集合体である。システムユニットの持つ領域には、固有領域(ノードごとのローカルな管理構造を主に扱う領域)と共有領域(グローバルな管理構造を置く領域)がある。システムユニット内には、複数のシステムスレッドが動作している。

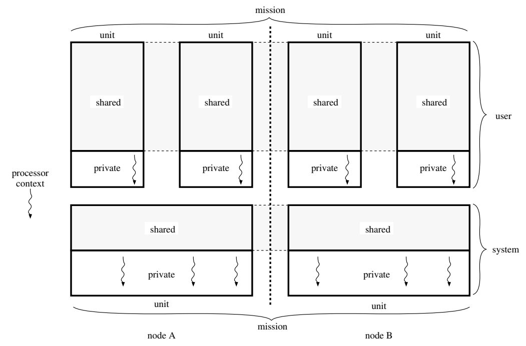

図3 ミッション-ユニットモデル

以下、ミッション-ユニットモデルの扱いに関する概要を説明する。

3.1 スケジューリング

ユニットもシステムスレッドも、それぞれプロセッサコンテキストを持つので、ローカルスケジューラは同じキューでこれらを扱う。ただし、ユーザにはチケットが与えられているので、ユーザのユニットについては、チケットに比例したプロセッサ時間配分を行う$^{2)}$。また、プライオリティや、時間的制約の強い(リアルタイム性が高い)ものについては、別のスケジューリ

ングポリシーを併用する。

グローバルスケジューラは、システムスレッドとして動作し、ユニットが動作するノードとプロセッサを静的および動的に決定する。例えば、図 4の (a) のように、チケット 300 を持つユーザ A が、12 個のユニットからなるミッションで並列処理を行っているとする。4 つのプロセッサはそれぞれ 3 つのユニットを担当することになる。

この状況で、チケット 100 を持つユーザ B が、1 個のユニットのみのミッションを起動したとすると、グローバルスケジューラは、システム全体がユーザ A とユーザ B に 3:1 のプロセッサ時間を与えられるように、図 4の (b) のようにユーザ A のユニットを移送する必要がある。このとき、移送されるユニットのうち 1 つはノード内での移送であり、ローカルスケジューラの待ちキューどうしのつけかえのみでよく、ほとんどコストはかからない。問題となるのは、残り 2 つのユニットの、ノード間の移送である。ノード間での共有していないデータはすべて移送しなければならないが、この例であれば、それはほとんど固有領域のみであり、移送のコストを小さくできる。このように、ユーザは、ノードあたりに多数のユニットを生成するようにプログラムを作成すれば、移送によってユニットの動作が長く停止したり、ユニットどうしの速さに偏りが生じたりしなくなる。

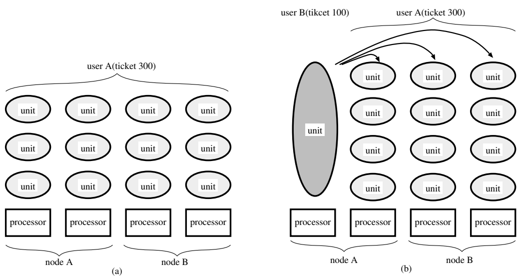

図 4　グローバルスケジューリングと移送

3.2　共有メモリを用いた並列処理

コンピュータ ·コロニーにおける共有メモリでは、ページごとにホームとなるノードが必要である。このホームは所有責任を持ち、一貫性制御に関するノード間の通信のすべてに関わる。このため、他ノードよりホームノードでの書き込み頻度が高いほうが、一貫性制御のオーバヘッドを削減することができる。

よって、並列プログラムを作成する場合には、共有ページのホームノードにあるユニットが専ら書き込みを行うように努める必要がある。このため、共有ページのホームの位置は、ノードではなくユニットに依存し、ホームであるユニットが移送されたときには、ホームページもそこに移送するようにする。

ユニットは通し番号による識別子を持ち、またそれにもとづいて自分がホームとなる領域を持ち、認識する。このようなユニットごとの分担は、ミッションの生成の際に、実行ファイルの初期化ヘッダ部分を読んで、最初から複数のユニットを生成して決定するか、あるいはユニットが 1 つだけ生成してから、プログラム中のシステムコールによって、複数のユニットと領域を指定して決定すればよい。

3.3　ミッション、ユニットが持つ管理構造

ミッション、ユニットは、それぞれ固定長の構造体を持つ。ミッションは複数のノードにまたがっているため、システムユニットの共有領域に構造体を持ち、ミッションの持つ共有領域に対して定義されるグローバルページテーブルも、同じくシステムユニットの共有領域で維持、管理される。ミッション構造体は、属するユーザの識別子、コード領域やデータ領域の範囲などを持つ。

ユニットはローカルにしか存在しないので、システムユニットの固有領域に構造体を持つ。ユニット構造体は、プロセッサコンテキスト、属するミッションの識別子、ページテーブルの識別子、スタックやヒープの範囲、プライオリティなどのスケジューリングパラメータ、実行可能や停止などの状態、プロセッサ使用時間などを持つ。また、ユニットはそれぞれローカルページテーブルを持つが、これはノード内で同じミッションに属するものどうしで、共有できる部分が多い。

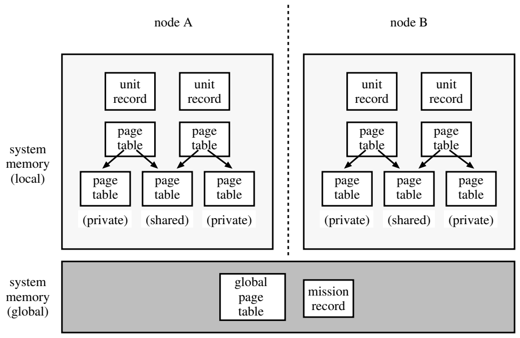

図 5　管理構造

3.4　ページ管理

共有ページと固有ページでは、管理方法が異なる。固有ページであれば、確保および解放は 1 つのユニットだけに関わり、他ノードに連絡する必要はないが、共有ページの場合、いずれかのノードで確保する場合、他のノードにその確保を通知しなければならない。これは、複数のノードにおいて新しいページの確保および初期化を同時に行うのを防ぐためであり、ホームノードにおいて集中管理される。

図 6は、いずれもコード部分である共有ページ (a)、(b) のノード A～C のページテーブルおよびローカルメモリについて、アクセス時とページ管理の結果をそ

れぞれ上下に示している。ページ (a) はノード A がホームで、ここにページ内容がある。ノード B、C では、ページテーブル上でネットワークスワップアウト、つまり他ノードのローカルメモリ上に内容がある状態であることを記録しておく。ここでノード B にアクセスがあれば、ノード B は物理ページを確保してコピーページとする。つまり、ネットワークインターフェースに指示し、ホームであるノード A に連絡してネットワークインターフェースが常にページ内容の一貫性を保つように設定する。

ページ (b) はまだどのノードにも内容がなく、無効となっている。このページにアクセスしたときには、物理ページを確保してディスク上のファイルからコードを読みこむとともに、他ノードに連絡して、ページテーブル上でネットワークスワップアウトと記録する。

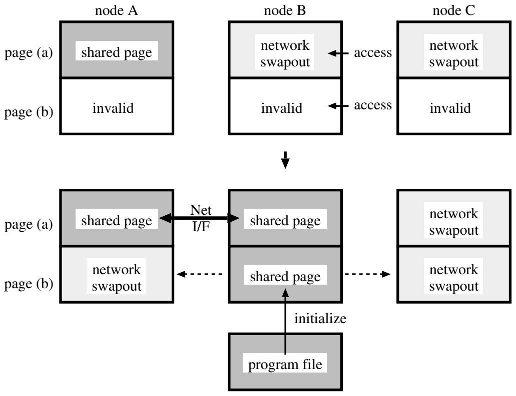

図 6　共有ページ管理

4. ユニットの移送

この章では、ユニットの移送を中心とした、実装方法について説明する。ユニットを移送するには、移送先で次のものを用意し、移送元で消去することが必要である。

ユニット構造体　プロセッサの上で動作するのに最低限必要。ユニットに固有。固定長で、1 回のメッセージで送信可能。

ローカルページテーブル　仮想ページを用いて動作するのに最低限必要。全体は大きいが、ミッションごとに共有できる部分が多い。

ページ　すべてが最低限の動作に必要ではない。量が大きく、1 回のメッセージで数ページ送信するのが限度である。

また、ユニットの移送に伴うミッションの操作に関して、次のような段階が考えられる。

(1) 移送先ノードにミッションが存在しなければ、そのノードにミッションで共有できる構造を用意する。これをミッションの拡大という。

(2) ユニットの移送を行う。ユニット構造体とローカルページテーブルを移送先で用意した段階で動作を再開し、ページはあとから移送する。これをページの遅延移送という。

(3) 移送元ノードにミッションが存在しなくなれば、ミッションが使用していた資源を解放する。これをミッションの縮小という。

ノードに対して十分な数のユニットが生成される並列プログラムであれば、つねにユニットの移送が行われる範囲にミッションが拡大しており、ユニットの移送そのもののみを行えばよい。もしミッションの拡大や縮小が必要でも、それはユニットの移送の前後にバックグラウンドで行われ、実際にユニットが動作を中断しなければならないのは、ユニットの移送そのもののときだけである。またミッションの存在により、ユニットの移送そのものにかかるコストは小さくなっており、さらに、ページの遅延移送によって、移送するユニットの動作の中断を最小限にすることができる。

以下、メモリの管理単位であるページについての扱いについて触れたあと、これらの段階の順に、実際の動作を説明する。

4.1　ユニット移送先におけるページの扱い

移送されたユニットは、ローカルページテーブルと、移送前からこのノードにある共有ページを持つ。共有領域については、すでに動作しているユニットと同様であり、ページフォルトが起これば、ネットワークインターフェースに指示して、ホームと連絡して共有の状態にする。以降のメモリ転送は、ネットワークインターフェースがキャッシュライン単位で行う。固有ページは、移送元にしかなく、また移送元に残す必要がないため、ページフォルトが起これば移送元に連絡して、ページ全体を転送させる。移送元では、そのページを解放することができる。

4.2　ミッションの拡大

ローカルページテーブルの共有領域部分は、すでに他ノードに実体があるページにはネットワークスワップアウトと記載されている。ミッションの拡大は、ローカルページテーブルの共有領域部分を、このようにネットワークスワップアウトの状態にする。拡大が終了した後には必ずユニットが生成あるいは移送されるので、図 7のようにミッション拡大の段階で、そのユニット (図ではユニット 0) のための完全なローカルページテーブル (テンプレート) として用意する。固有領域は無効にしておく。

4.3　ページの遅延移送

移送元では、ローカルスケジューラのキューからユニットをはずし、ユニット構造体を移送先に送信する。移送先ではユニット構造体を受信し、移送元のノードを記録しておく。

ローカルページテーブルは、あらかじめミッション

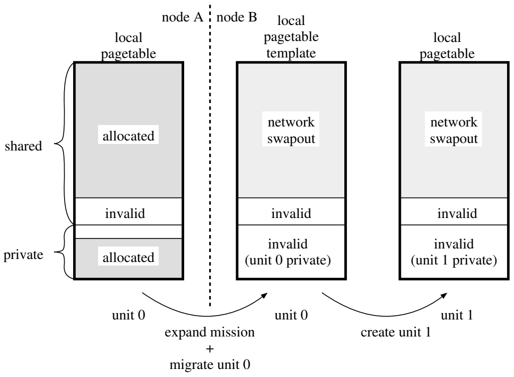

図 7　ミッションの拡張とテンプレート生成

が拡大しておくことでほとんど用意されている。そのノードで最初のユニットであれば、ミッションの拡大のときに用意されているテンプレートを、そのままユニットのローカルページテーブルとして利用する。すでにユニットが存在すれば、そのローカルページテーブルの共有部分にポインタを張ったローカルページテーブルを用意すればよい。固有領域については、最初はすべて無効でよい。

これらを用意した段階で、ユニット構造体をローカルスケジューラのキューに入れれば、ユニットは動作を再開できる。以降、ページの遅延移送について述べる。

4.3.1　移送先でのページフォルトによるもの

図 8に、ノード A からノード B へユニット 0 を移送した直後のようすを上に、ページ移送の後のようすを下に示す。共有ページ (a) にアクセスしてページフォルトが起こり、そこがネットワークスワップアウトであれば、ホームであるノード C に連絡して、共有の状態にする。以降の内容の一貫性はネットワークインターフェースがキャッシュライン単位で制御する。固有ページ (b) であれば、移送元に要求してページを転送させる。固有ページは、移送元では不要となるので、物理ページを解放できる。

4.3.2　移送元でのシステムスレッドによるもの

移送先のオンデマンドによるページ取得に伴い、移送元でも、システムスレッドが能動的に移送先へページを転送する。移送元に置く必要のない固有ページを解放する必要がある。そのうち、移送直前に使用していたスタックは、ユニットの動作再開後すぐに利用することが予想できるので、移送先から要求される前に、ユニット構造体の移送後直ちに移送元から転送する。そのあと、ローカルページテーブルなどを参照し、残っている固有ページを順に移送先に転送してゆく (図 8 の (c))。すべての固有ページを転送し終われば、移送元のローカルページテーブルその他のユニット管理構造を消去する。

このあと、ミッション縮小の必要があれば、このシステムスレッドが引き続いてその操作を行う。なければ、ホームが移送したユニットに属するページを移送し、移送元ノードにはコピーページを残しておく。

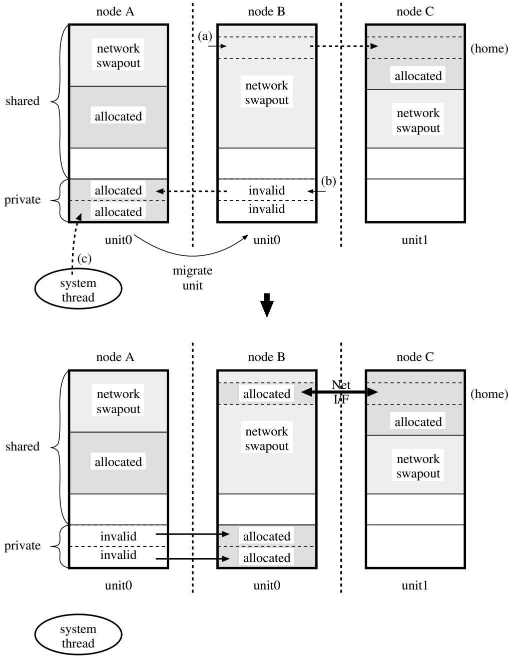

図 8　ページの移送

4.4　ミッションの縮小

ミッションの縮小では、ミッションの存在によって保持していたローカルページテーブルの共有領域および共有ページを解放する。まず、移送されたユニットがホームであるページから転送し、移送元からは消去する。残った共有ページは、ネットワークインターフェースに指示して最新の内容をホームに戻し、無効化して解放する。すべてのページが解放されれば、ローカルページテーブルを消去し、システムスレッドは停止する。

5. まとめ

本稿では、分散共有メモリをベースとしたシステム、コンピュータ・コロニーにおいて、高速移送可能な並列アクティビティモデルとしてミッション-ユニットモデルを説明した。移送のコストが低ければ、移送によってシステムの負荷を大きく増やすことなく、公平で効率のよい負荷の再配置を行うことができ、並列処理の効果も大きくなる。

コンピュータ・コロニーの分散共有メモリ環境では、ノードごとにローカルメモリに対する物理アドレス空間があり、ローカルなページテーブルが必要である。タスク-スレッドモデルにおいては、スレッドがそれぞれ固有に利用する領域についても、分散したローカル

ページテーブルなど管理構造の一貫性を保たなければならないため、オーバヘッドが大きくなる。ミッション-ユニットモデルでは、固有領域に対して、固有していることにもとづいた操作を行い、このようなオーバヘッドを取り除く。

ノード内では、ローカルメモリ上のページやローカルページテーブルをミッション内で共有することが可能なため、ミッションが存在するノードへの移送には、あまり大きな転送量を必要としない。また、ミッションの拡大が必要でも、その操作はユニット移送前にバックグラウンドで行うことができ、移送するべきページの多くもユニット移送後にバックグラウンドで遅延移送するため、ユニットの動作が中断される時間が短い。

このように、ミッション-ユニットモデルでは分散ノード間の移送コストが低く、またノード内での生成コストも、ローカルページテーブルの共有などを利用して、極力小さくすることは可能である。よって集中共有メモリ環境においても、タスク-スレッドモデルに比べて大きく性能が落ちるわけではなく、コンピュータ・コロニーにおける並列アクティビティモデルとして、ミッション-ユニットモデルが最もふさわしいと考えられる。現在は、実験環境の上でミッション-ユニットモデルの実装をすすめ、今後グローバルスケジューリングなどと組み合わせた評価を行っていく予定である。

謝辞

メンター・グラフィックス・ジャパン（株）には、Higher Education Program の一環として製品とサービスをご提供いただきました。ここに感謝いたします。

なお本研究の一部は、文部省科学研究費補助金（基盤研究 (B)(2) 課題番号 10558045 ならびに (C) 課題番号 09680334）による。

参考文献

1) Kai Hwang, Hai Jin, Edward Chow, Cho-Li Wang, Zhiwei Xu : Designing SSI Clusters with Hieratchical Checkpointing and Single I/O Space, IEEE Concurrency,pp.60-69,Jan-Mar,1999

2) Damien Le Moal, 他 分散システムにおける Fair Share プライオリティ・スケジューラ, 情処研報 (SWoPP 予稿集),99-OS-80,1999

3) 山添博史, 他 :並列アプリケーションを指向した分散システムコンピュータ・コロニーの構想, 情処研報,97-OS-76,pp55-60,1997

4) 青木秀貴, 他 :共有メモリベースのシームレスな並列計算機環境を実現するオペレーティングシステムの構想, 情処研報,97-OS-34,1997

5) SPARC International, INC. (相越克久, 田中長光 訳 / 多田好克監訳): SPARC アーキテクチャ・マニュアル バージョン 8, トッパン (1992)