プログラミング 2－1
（1995. 8. 24）

プロセス移送を指定可能とする並列プログラムの実行系

岡本秀輔　曽和将容

電気通信大学　大学院　情報システム学研究科

メッセージパッシングを行う並列言語を用いて、問題を機能別に分割した並列プログラムを作成した場合、そのプロセスが行う通信の頻度や量は様々なものとなる。したがって、その実行に際してはプロセスのプロセッサへの割り当てが重要となってくる。プロセッサへの割り当てには、静的な方法と実行中に割り当てを変更する動的な方法とがあるが、ユーザ指定による動的なプロセッサ割り当てを考え、プロセス移送をプログラム中に指定することにより、プロセッサの動的な割り当てを可能とする言語実行系について検討する。

A Run Time System for Parallel Programs Specified The Process Migration

Shusuke OKAMOTO and Masahiro SOWA

Graduate School of Information Systems, The University of Electro-Communications

When we write a parallel program using a language with message passing facility and using the functional division scheme, processes of its program communicate the various amount of data with the various frequency. So, the run time allocation of the process to processor is important for this kind of programs. In this paper, we examine the run time system for parallel program in which the process migration is specified as well as its dynamic processor allocation scheme.

1 はじめに

本稿では、メッセージパッシングを用いた並列プログラムの効率的な実行を行うにあたり、ユーザ指定によるプロセス移送の導入を検討し、それを実現する言語実行系について考察する[1]。

メッセージパッシングを用いる並列プログラムを、その設計方法から分類すると、データ領域を分割して並列性を抽出する方法と、処理すべき機能を分割して並列性を抽出する方法とに分かれる。前者では、それぞれのプロセスはほぼ同一の処理を行うために、プロセス間の通信もほぼ同一であり、ある特定のプロセスのみが大量の通信を行うということはない。一方後者では、プロセスの処理が機能別となっていることから、処理内容は様々であり、通信頻度や通信量を比較してもプロセス毎に様々なものとなり得る。したがって、このような処理内容が均一でない並列プログラムの効率的な実行を考える場合には、プロセスのプロセッサへの割り当てが重要となる。

一般にプロセスのプロセッサへの割り当て方法としては、大きく分けて、静的な割り当てと動的な割り当ての2つがある。

静的な割り当てとは、プログラムの作成時やロード時といったプログラムの実行前にプロセッサへの割り当てを決め、その後の実行中にその割り当てを変更しない方法である。この方法は現在いくつかのシステムで利用されている。例えば、商用の並列マシン Transputer における並列処理言語 Occam[2]では、プロセッサ割り当てが言語仕様の一部とされている。同じく商用並列マシン nCUBE2 における C 言語実行環境[3]では、ローダに対して指定する方法と、プロセッサ番号を実行中に取得し、その番号から処理内容決定する方法とがある。後者は、実行中に割り当てを決めているが、割り当ての変更を行うものではない。また、LAN 等で使用可能な PVM[4]プログラム実行環境では、新たにプロセスを生成する際にプロセッサの種類やプロセッサ名を用いて指定する方法を用いることができる。この方法も実行時に割り当てを決めるものであるが、割り当てを変更しないので静的な方法といえる。

これら静的な手法は、プログラム中の最も重要な計算および通信を分析し、得られた特徴を直接的にプロセッサ割り当てに反映する方法である。しかし、実行中に割り当てを変更しないために、実行に際して、想定していたプロセッサ数が得られない場合や、プロセス間通信の負荷が実行状況で変化する場合などに柔軟に対応することができない。

一方、後者の動的なプロセッサ割り当てとは、実行中のプロセスを状況に応じて他のプロセッサへ移動する方法である。この方法は、*プロセス移送*と呼ばれ、Mach や Amoeba といった分散オペレーティングシステム（以下分散 OS)でサポートされている[5]。これら分散 OS で行われる動的なプロセッサ割り当ては、大域的なプロセス管理としてプロセッサ間の負荷分散をはかることがその主たる目的となる。

一般に分散 OS では位置透過性を提供していることが多く、動的なプロセッサ割り当てにおいてプロセス移送が生じる場合にも、それをプログラムで関知する必要がない。したがって、プログラムの作成が複雑にならないという利点につながる。しかし、プログラムが移送と無関係ということは、プログラマが知り得る情報を記述する場所がないと見ることもできる。事実、分散 OS ではこの情報不足のためにプロセス移送の機構が複雑になっている。例えば、プロセス全体を移動する必要のない場合でもすべてを転送しなければならず、それらの通信量を減らすために、遅延転送といった様々な対策が施される。これらは

まさに移送のための情報が少ないための苦労といえる。

本研究では、並列プロセスの移送をプログラム中に記述し、これを実行の効率化に利用する方法で、動的なプロセッサ割り当てを行う。特に、プロセス移送の処理自信もプロセッサ間の通信であるという事実から、プロセス移送によるプロセス間通信のオーバヘッドの削減が可能な実行環境を考える。

プログラム内でプロセス移送の情報を指定することにより、プロセス全体ではなく一部の一時的な移動を行う。これにより、プロセス移送を行うためのプロセッサ間の通信コストを減らすことができる。また、並列プログラムのような、並列プロセス間の互いの処理が深く関係している状況では、プロセス移送のための通信の効率化のほかに、並列プロセス間の通信の効率化や大域的なプロセススケジューリングの質の向上にもつながる。

2 並列プロセスの移送

本研究の対象が通常のプロセス移送ではなく、並列プロセスの移送であるということを明確にするために、ここでは、ユーザ指定による並列プロセス移送の基本的な考え方および前提条件について述べる。

並列プロセス

本研究ではメッセージパッシングにより通信を行う並列プログラム扱い、プロセス移送の対象は、その並列プログラムを構成する並列プロセスとする。

各々の並列プロセスは、逐次的な実行を行い、実行すべき命令列と大域変数及び局所変数といった変数、そして実行の状態や履歴の情報を持つものとする。ただし、すべてのプロセスの命令列は、実行を行うすべてのプロセッサ上で利用可能であるとする。すなわち、ここで考えるプロセス移送とは、変数とコンテクストの移動である。

分散OSにおけるプロセス移送では、プロセスを構成する変数及びコンテクストの他に、システムコールの状態といったプロセスを構成するすべてを他のプロセッサへ転送する[6]。そのために、プロセッサ間の通信量がかなり大きなものとなる。一方、本研究が扱う並列プロセスの移送では、限定されたある期間のみプロセスを移動することで、プロセス移送のための通信オーバヘッドの軽減をはかる。つまり、プロセス全体ではなく、期間中に必要な最小限のデータのみを移動する。そして、移動する時期および内容についても、そのすべてをユーザによりプログラム中に指定するものとする。

実行環境

実行環境としては、複雑なプロセススケジューリングを単純化して考えるために、プロセッサ数はプロセス数よりも多いが、プロセッサ間の通信速度がすべて均一ではないものを仮定する。

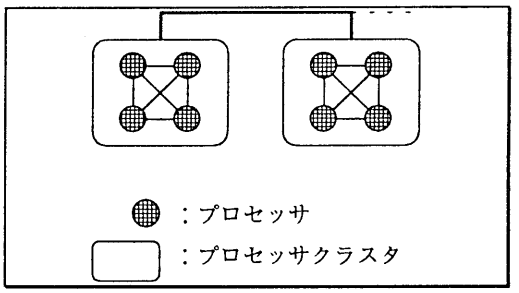

図 1：実行環境の例

例えば、図 1のような実行環境である。この図は複数のプロセッサがプロセッサクラスタを構成している例で、クラスタ内の通信はクラスタ間に比べ高速である。実行される並列プログラムのプロセス数は、クラスタの中にはすべて収まらないが、全プロセッサ数よりは少ないような状況である。

2階層スケジューリング

逐次プログラムのみが実行される状況下でのプロセススケジューリングは、実行に優先順位があるとしても、原則的には公平性を保つことがその目的となる。しかし、並列プログラムが複数実行される状況においては、他のプログラムの実行イメージであるプロセスのほかに、同じ1つのプログラムを構成するプロセスが存在する。したがって、プロセススケジューリングは、単に公平性を保つことばかりではなく、同じプログラムを構成するプロセスに対する処理内容に応じたスケジューリングも必要であると考えられる。

そこで、本研究での考え方として、独立したプログラムを構成するプロセス間のスケジューリングは、それぞれの処理内容とは無関係に行われ、1つの並列プログラムを構成する並列プロセス間のスケジューリングは、プログラム内で指定された情報を基に行われるという、2階層のスケジューリングを想定する。具体的には、図 2のように、並列プログラムを構成するプロセスのまとまりは OS によってスケジューリングが行われ、並列プログラム内の各々プロセスのスケジューリングはユーザの指定により実行系が行う。

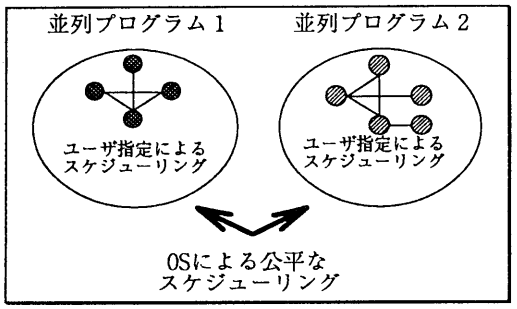

図 2：2階層スケジューリング

このような2階層のスケジューリングを想定することにより、他のプログラム間の関係をまずは排除して考えていく。2階層のそれぞれの対応関係は今後の検討事項となる。

3 実行系の設計

ここでは、実行系を設計するにあたって考えられる様々な選択を、プログラム中での指定とともに考察していく。

移送の指定方法

プロセス移送をプログラム内で指定する場合に、移送先の指定には2つの方法が考えられる。1つはプロセッサをプログラム内で指定する方法であり、他方は通信相手といった他のプロセスを指定し、そのプロセスが実行しているプロセッサへ移動する方法である。

前者は、プロセッサ割り当てに関しては直接的ではあり、プロセスの移動に関して処理が容易になる。しかし、プログラム内で他のプロセスが実行している位置をプロセッサとして意識しなければならず、結果として、プロセス移送のためにプログラムの構造自体が影響を受ける可能性がある。

後者では、プロセス移送の指定が他プロセスを指定しているために間接的となり、プロセス名からプロセッサ名への変換機構を必要とする。しかしその代わりに、プログラムはある並列マシンやその結合関係といった特定の実行環境に依存しないので、並列プログラムの移植性を向上させるという利点につながる。

本研究では、プログラマに負担をかけることで、移送の情報を得ることを考えているが、その情報の指定のためにプログラムの構造が大きく変わるのは好ましくない。また、将来的には前述の2階層スケジューリングに対応させることも考慮にいれて、ここでは後者の他のプロセス指定によるプロセス移送を行う方法を選択する。この選択により、並列プログラムの作成方法として、まずプロセス間の通信関係をプロセッサの結合や配置とは無関

係に記述し、最適化の作業として後からプロセス移送を指定することが可能となる。

もう1つの選択は、プロセス移送により並列プロセスは全体を移動する方法と、一部のみ移動して実行した後に、プロセスを生成した元のプロセッサに戻る方法である。前者では、プロセス移送の時期のみをプログラムで指定することになり、プログラムは煩雑にならないが、分散OSが行うのと同様に情報の少ない所で、プロセス移送の最適化を考えねばならない。後者では、プログラム上での指定が煩雑になるが、プロセス移送の機構そのものは単純かつ高速なものとすることができる。したがって、この選択はプロセス移送した後、やがて元のプロセッサに戻る後者の方法とする。この元のプロセッサへ戻ることを、後の議論では“戻り移送”と呼ぶことにする。

メッセージ転送

プロセス移送が行われる状況でのメッセージ配送に関しては、以下の方針をとることとする。

a)移送中のメッセージ配送

移送中のプロセスへのメッセージは、“移送中のプロセス宛”と明示してあるもののみ移送先へ配送する。その他のメッセージは、プロセスが戻り移送を考えて、指定プロセスの生成元へ配送する。ただし、上記の指定をしたメッセージでも、実際に宛て先のプロセスが移送中でなければ、プロセスの生成元へ送る。

b)移送先の未読のメッセージ

移送先へ配送されたが、実際には読まれなかったメッセージは、戻り移送とともにそれらのメッセージも転送する。

これらの設計方針は、プロセス移送をプログラム中で指定するので、それに関連する通信もプログラム中で指定するべきであるという思想に基づいている。

プロセス移送と関数呼び出し

プロセス移送に伴う通信量を削減するために、実行を再開した後に読み出す変数を指定する。この指定方法を関数呼び出しとの関係から分析する。まず、1度だけ移送する場合を考察し、その応用として元のプロセッサへ戻る場合を考察する。

a)関数呼び出しと無関係

関数呼び出しとプロセス移送が無関係の場合には、プログラム中では、移送後に使用する大域変数及び局所変数の全てを指定することができる。例えば次のコード片では、p1 への移送後に大域変数 a と b および局所変数 j を読み出し参照するために、移送の対象として指定している。

```
int a,b,c;
main()
{
 int i,j,k;
 ...
 migrate to p1 with a,b,j;
 ...
 ... = a + b + j;
}
```

b)移送先で関数が return する

移送先で関数が return する場合は、大域変数には問題はないが、変数のスコープの関係上、局所変数を直接指定できない場合が生じる。例えば次のコードにおいて、関数 func の中でのプロセス移送を指定しようとした時に、main 関数の局所変数を指定できない。

```
main()
{
   int i,j,k;
   ...
   func();
   ...
}
func()
{
   ...
   migrate to p1 with ???
   ...
}
```

図 3はこの状況をあらわしている。この図

は点線を境にプロセス移送が生じている。Call/Return はそれぞれ関数 func への呼び出しと func からの戻りを意味する。○と曲線はプロセス移送に伴って、変数の対応が必要であることを意味している。関数 func が常に main から呼ばれることを仮定しない限り、func 上で main の変数を指定することはできない。また、移送先での関数の戻りが、main であるという実行履歴も必要である。したがって、このような状況では、変数のみではなく、実行履歴を含むスタックを指定し、それを移動する手段が必要となる。

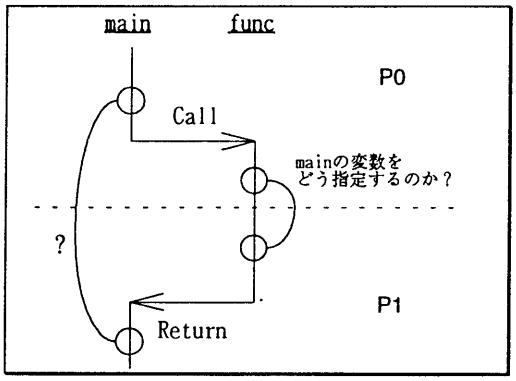

図 3：局所変数を指定できない例

c)移送と関数呼び出しが一致する

b)の問題が対処されれば、基本的には 2 回以上のプロセス移送は、a),b)の組み合わせで考えることができる。しかし、特殊な場合として、関数の Call/Return が、移送してまた元に戻るという動作と一致する場合も考えられる。つまり、遠隔手続き呼び出しと類似の状況である。この場合、移動すべきものの指定として、関数へのパラメタおよび戻り値のみとして、移動のための通信を更に削減可能である。

スタックの移動指定

並列プロセスが持つスタックは、関数へのパラメタ、局所変数、関数の戻り先命令アドレス、フレームポインタから成ると考えられる。この他に関数の戻り値を一時的にスタックに置く場合もある。いずれにせよ、スタックは関数呼び出しを 1 つの単位として、その内容を分割することができる。

スタックをプロセス移送に伴って移動させる場合には、次の 3 つが考えられる。

1.現在実行している関数部分
2.スタック全体
3.指定数の関数部分の集まり

1.はプロセス移送の後に、関数が return することなく戻り移送を行う場合に相当し、2.は状況に関係なくすべて移動することを意味する。3.は指定した回数の関数 return が起こった時に、戻り移送が行われるような状況である。しかし、この指定は非常に困難である。

結果として、スタックの移動に関して実行系は、1.および 2.の手段を備えるものとする。

移送操作のまとめ

これまでの考察から、並列プロセスの移送の指定及び実行系が持つべき機能をまとめる。

a)移送指定と実行系の処理

プロセス移送の指定は、プロセス名を移送先の代わりに使用する。このため実行系はプロセス名から適当かつなプロセッサを選ぶ機構が必要となる。

また、ある移送の次には戻り移送という元のプロセッサへ戻る移送を指定することとし、この制限に反する指定には実行系は対応しない。

上記の条件において、関数呼び出しに関連して 3 種類指定可能とする。すなわち、

1. 移送後に関数 return をすることなく同一関数内から戻る指定。
2. 移送後に return を行った後に戻る指定。
3. 移送とその戻りが関数呼び出しと一致する指定。

これら 3 つに応じて実行系では、指定された変数のみを移動する機能およびスタックを

移動する機能を必要とする。

b)メッセージ配送

移送中のプロセスとの通信は、プログラム中で明示的に指定するものとし、実行系はこの指定がされたときのみ対応する。また、移送先に配送されたメッセージの中で、未読のままプロセスが戻り移送をした場合には、実行系がこれを戻り先のプロセッサへ再配送する。

おわりに

本報告では、メッセージパッシング型の並列言語において、プログラム内でプロセス移送の指定を行う方法について検討してきた。現在、この考察に基づいて実行系の実現を行っている。

参考文献

[1] 岡本秀輔, 曽和将容, "メッセージ通信型並列処理言語の実行系におけるプロセス移送の機能", 情処50全大 pp.5-33～5-34(1995).

[2] INMOS Ltd., "occam2 Reference Manual".

[3] "nCUBE2 Programmer's Guide, r.2.0", nCUBE corp.(1990).

[4] Al Geist, et al. "PVM - Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing", MIT Press(1994).

[5] S. Mullender, "Distributed System Second Edition", Addison-Wesley(1993).

[6] 前川守, 所真理雄, 清水謙太郎 編, "分散オペレーティングシステム －UNIXの次にくるもの", 共立出版(1991).