マルチプロセッサにおけるプロセスの「からだ」と「たましい」の管理

東京大学工学部　永松　礼夫
Leo Nagamatsu

1．はじめに

ＶＬＳＩ技術の進歩により、安価で高性能のプロセッサが入手できるようになり、これらを結合した高性能並列処理計算機を構築することが可能となってきた。このようなシステムでは数多くの処理装置を効率よく接続するために種々の方式が検討されている。

その方式は、大別すれば疎結合と密結合の二方式であるが、疎結合では接続媒体を選べば台数をかなり増やすことができる利点があるものの、通信遅延と通信プロトコルの起動の手間とによって性能が制限される点が問題になる。一方、密結合では共有変数方式のおかげで従来と同様の手法で問題の記述ができるのだが、マルチプロセッサ特有のＰＥ間の距離についての意識がない。キャッシュなどを工夫して効率を保ったまま透明性向上をはかるアプローチでは、複雑なハードウェアを投入していかに従来の思想を守るかに努力がされている。

本稿では、空間的に近いところ(ひとつの箱の中など)に分散したプロセッサ群のなかで計算オブジェクトの移動の手間を考慮した問題の分割について論じるとともに、それをインプリメントする際の支援ハードウェア機構についても述べる。

2．からだとたましい

形のある構造(からだ)とない物(たましい)に分けて考察することで、より鮮明にシステムの問題点を捉えることを試みた。まず言葉を定義する。はじめは単純に計算の前線の数が変化しない状況である。

からだ：プロセス・コントロール・ブロックやスタック上のコンテクストのような記憶領域を確保しプロセスを表現する構造(図１)

たましい：その場所(プログラムカウンタの指す、あるいはアクセスの起きる点)で処理の進行するところ

とする。からだとたましいは対応するデータ構造が存在するか否かで区別されており、プロセッサが複数かにはよらない。マルチプロセッサで問題が顕著になるのはプロセッサ・エレメント(ＰＥ)の間の壁が厚いので、たましいは容易に移動できるが、からだは困難という実質的な差が生じるからである。

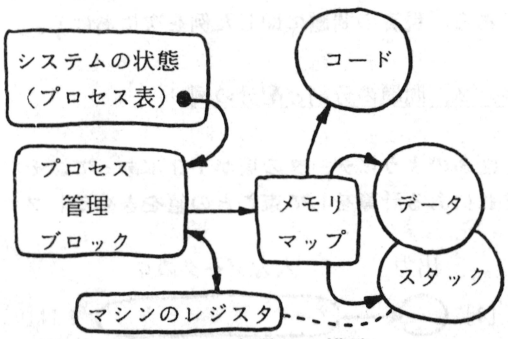

図1　プロセスを表現する構造

3．具体的な問題

3．1．リモート・プロシージャ・コール

ＲＰＣでは、図２のように制御の移動が進行する。これを、能動的なプロセスと受動的な手続きとのモデルでみれば、あるものがプロセッサＡからＢへゆき、またＡに戻ってきたと解釈できる。つまりからだを置いてたましいが飛回っていると解釈できる。

また、Ｂで準備をして待っていた能動的実体に

Aからメッセージが送られ、その返事がくるまでAは自主的に待っていたともみれる。

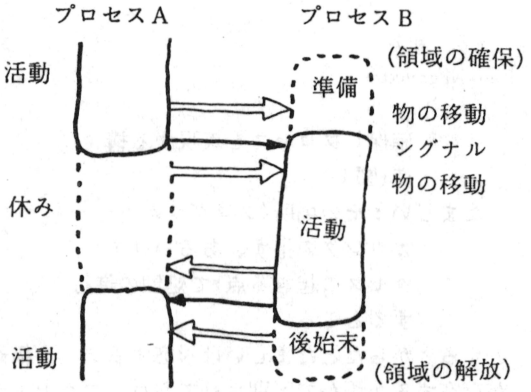

図2　Remote Procedure Call

いずれにせよ、実際の処理が起動されるまでにそのアクティビティの使うからだがそのノードにできていなければならない。

さらに、図2でシグナルと示された因果の流れのほかに、パラメータなどのデータの転送が必要である。現実の問題に即した例を次にあげる。

3．2．問題の分割と配分の例

図3のようにデータの束が10本あって、それぞれにある計算をして束ごとの値をきめる。プロ

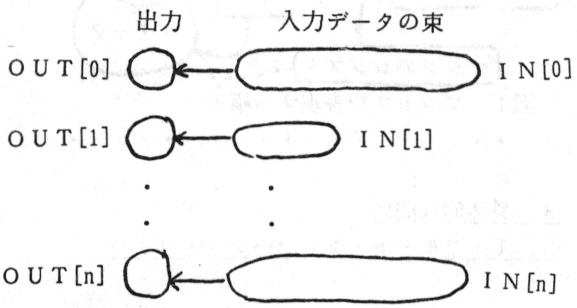

図3　分割して問題をとく

セッサあるいはプロセスは4個しか用意せず、ひとつの束について計算が終ったならば、次の束をとりにゆく、なお計算の時間は一定ではない。プログラムは、

```
while(未処理の束がまだある){
    束をとってくる
    値を求める
}
```

と書けるであろう。

このような問題を解くときに、計算が終って暇になったプロセスは、仕事をとりにゆくことができる。そして、とってくる行為が仕事を引受けたことを意味する。

3．3．データ送受の対称性

Sendに主体性があるのなら、ふたつのプロセッサのうち、データをおくりたい側、自分にはデータが余っていて正の圧力がある側は、メッセージ通信として他へ送ることができる。しかし、自分にデータが欠乏していて、他からデータが欲しい、負の圧力があるときはReceiveだけでなく対応するSendの助けが必要となる。この点で、システムは対称でない。

もし「暇な」プロセッサが、データさえ受取れば計算活動を起こせる状況ならば、忙しいプロセッサの邪魔をする、「仕事を切り分けて送ってもらう仕事」をさらに頼むのではなく、こちらから仕事の一部を取ってきて負荷を分担するほうがよい。

どちらが忙しいか実行まで判らないならば、パイプラインのようなプロセッサAの結果をBが利用するようなシンプルな場合でも、必要なデータがそろったとき計算を開始するという因果関係と、二者のうち暇なほうがデータ実体の移動作業を分担することとは、分離できる。

上の因果関係については、プロセスの対について条件が成立したかの管理をすればよく、この情報は固定長であるので、支援回路としてハードウェア化するのも容易である。データの転送に関しては転送量などに応じて最適な媒体が異なることもある。例えば、二つの方式が使用可能で――起動に要する手間はかかるが、転送レートの高いパケット通信と一様な遅延時間のかかるリモート・メモリ・アクセス方式のどちらでも、他のプロセッサのメモリの内容を盗み読み／書きできる――

場合などである。

3．4．他の同期法

ランデブーでは待合わせをするプロセスの双方が準備完了した場合にさきに進むことができる。成立に際してやりとりされるデータは送り側では送り放しでよい。因果では、ランデブー終了はどちらのプロセスの意志の表明よりも後になる。

さらに、先読みにも応用できる。よいＯＳとは利用者の挙動を予知して、先回りして準備しておくものである[3]との説もある。控え目にいっても、確率的に儲かるような保険をかけてデータの先読みなどをしておくのはよい。

また、自転車操業のように明日つかうリソースを今日届けてもらう方式も考えられる。演算以外の処理では実際かなりの命令が移動である。

4．同期決定ユニット

4．1．シンクロナイザ

このように、因果(たましい)とデータ転送(からだ)が分離できるならば、それ専用の支援ハードウェアを設けることが考えられる。図4のようにシステムの中心に位置し、イベントの順序と待合わせを管理するハードウェアを考え、シンクロナイザと呼ぶことにする[4]。ひとつのイベントは要求とそれを発したプロセスの識別子よりなり、不可分の一サイクルで記録される。満たされた要求は割込みで通知され、そうでないならば時期が来るまで内部のキューで待たされる。

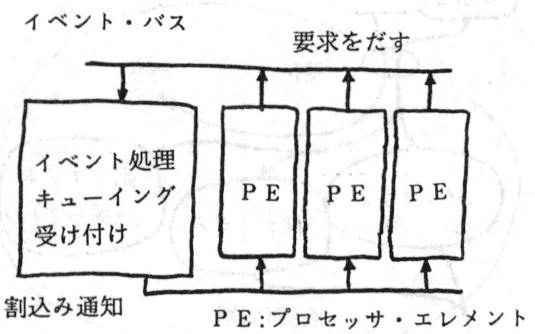

図4　同期決定ユニット
(シンクロナイザ)

もちろん、データフロー計算機のように粒度の小さい場合には計算の一命令ごとに待合わせがおこるのでその頻度は膨大になり、集中して管理すればホットスポットになってしまい実用的ではない。ここでは、プロセス間の同期など粒度のあらい共有資源に関わる現象の管理に用いることにする。

類似のもので、BalanceのＳＬＩＣ[5]ではシステムのなかでハードウェアの支援により短時間でイベントの順序付けを行なっている。データフロー計算機のために、専用に２つの演算オペランドを待合わせるタグつきのメモリを設けた例もある。

4．2．割込みの分配

同じ問題を解決できるサーバが複数システム中にある際、図5のように要求をどこでもよいから受付けてもらえるよう決定するメカニズムがほしい。ＳＬＩＣでは、外部の割込みを共有チャンネルに流し、パケットが流れ終るまでの時間内にハードウェアが衝突検出し「入札して」決めている。

要求の順序を記憶するキューをもつことは、公平性に寄与するが、ある管理方式を固定してしまうことにもなる。状況に応じて開くゲートのようなサービスを実現するには、各ノードが時々刻々報告をしないといけない。

また、ひとつのプロセッサに複数のプロセスを置く場合には、スイッチの回数はなるべく少なくしたい。状況の変化にあわせ、割込みによって現

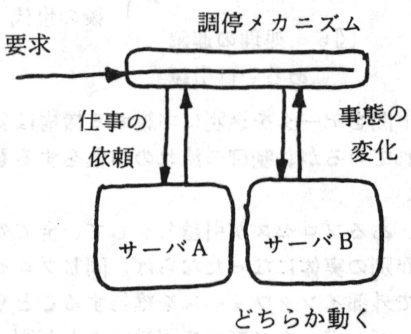

図5　競争入札の方式

在のコンテクストから割出す機能を持つことが重要となる。

4．3．引越し

マイグレーション(引越し)では、ＰＥ(プロセッサ・エレメント)の間でたましいとからだの両方が移動する。実際には、プロセスを停止させて、たましいを抜き、からだを運んで、たましいを戻すという一連の操作となる。からだを運ぶためには通常、からだのコピーをつくること――プログラムとデータを移すこと――が必要である。

同じプログラムが既にある場合は、データのみの移動でよい。いずれにせよ、引越しをしている最中は活動をしない。忙しい会社の引越しのように責任の所在を明確にし、新旧どちらの社屋でも業務を受付けるケースでは、移動中に活動ができる。この場合は処理の前線は複数あり、内部は多重アクティビティである。プロセスが繋がっている媒体を押えられれば、引越しの最中に送られたメッセージをシステムが待たせ、活動再開のあとで正しく送るようにすれば良い。

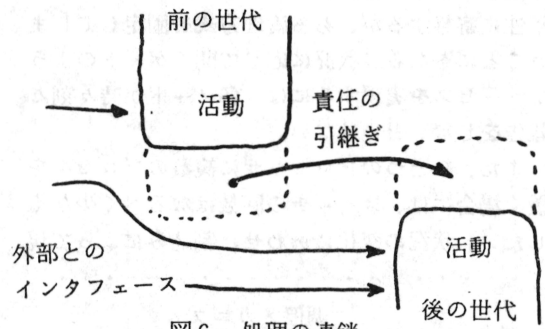

図6　処理の連鎖
あるいは引越し

ノード間をデータを透明に中継する機構は多く実装されているが、制御の流れの中継をするものは少ない。

さて、あるプロセスが引越しをして、全てのレコードが別の実体になったならば、同じプロセス識別子で外部インタフェースを維持することや、メッセージ・ポートの責任を引継ぐことが同一性の基準となる。

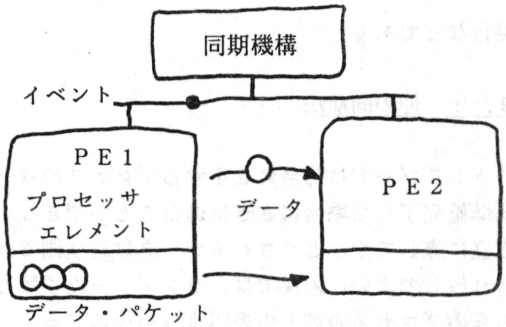

図7　転送媒体

限定された引越しの例として処理の連鎖がある。それは、図6のように委譲にちかくなる。

5．関連する問題
5．1．転送媒体

プロセッサ間の媒体を、データが着いたときにＣＰＵへ通知があるか、完了を必ず待つのか、起動の手間がかかるか、１ワードをいくらの時間で送るか、などを基準に使いわける(図７)。

試作では、他のＰＥの内部メモリを別のＰＥから読み書きできないハードウェアの制限[1]があったが、メッセージ転送ハードウェアの有効な利用ができた。

5．2．多重アクティビティ

たましいをある目的のための行為のグループと

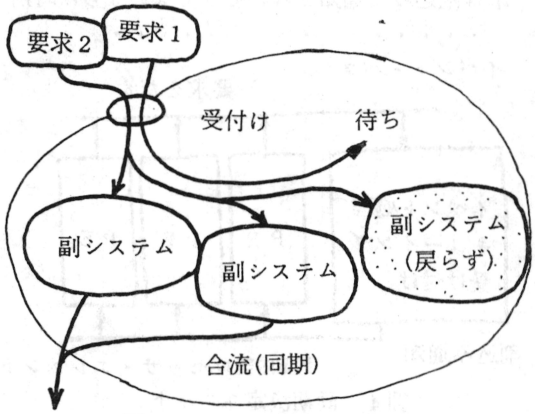

図8　フローの分割・追越し

すれば、サブゴールの問題である。コストが全くなければ各段階のたましいに別の識別子を付ければよいが、仕事が終るまで不滅の同一者が行なうのではなくなる。処理が分裂収束する場合の、フローの枝分れや追越し(図8)を扱うには、プロセス識別子のリストを管理すれば可能である。しかし、膨大な情報となる恐れもある。マルチスレッドも同様である。

5．3．処理前線の分散

現在の処理の進行によって変化の起きつつあるところに たましい があるとする。ポインタ(あるいはアクセスのための手掛かり)を受取って、それを用いてアクセスを行なうと、ノード間のメッセージを経由して相手ノードへのアクセスがおこる例を考える。アクセスは図9のように両ノードにまたがり、両ノードで たましい が活躍していることを示す。

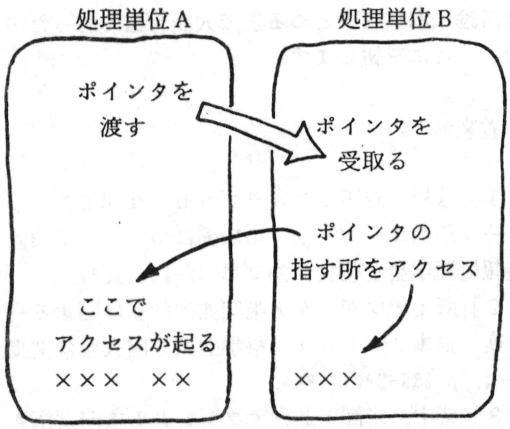

図9　分散した計算前線

「よいシステム」の基準として、たましいの働きがひとつのノードにどれだけ固まっているかが利用できる。

5．4．能動と受動

メモリ・アクセス・サーバの場合にメッセージによりメモリの書きかえを　起動する例を考える。要求はリクエスタからサーバへゆく。サーバに主体性がないのなら、サーバのなかでの「動きは」リクエスタの起こしたアクションの波及であり、たましい をリクエスタに貸したことになる。能動性をその行為に責任をもつ度合いと解釈すれば、たましい を貸したほうにも責任がある。フローの木で部分だけ持主がちがうとき、根までの所有権を問うことと同じである。

5．5．集中の問題

シンクロナイザを集中した機能として説明したが、信頼性からは特別なキーコンポネントが存在するのはよくない。いっぽう、分散して実装する際に、システム全体の中で同時性の厳密な管理をめざすと大規模システムでは不利になる。

6．オブジェクト・システムへの適用

6．1．背景

オブジェクト指向の枠のなかで並行処理を考える際に2つの立場がある。ひとつの立場は受動オブジェクトとプロセスを用いて表現するもので、ひとつの受動オブジェクトのなかで複数のプロセスが並行に実行できる。オブジェクトのなかに多重のアクティビティを許す方式であり、共有メモリ多重プロセッサシステムの出現と対応している。別の立場では、データとプロシジャを持つ受動オブジェクトとさらに処理装置をもつ能動オブジェクトを用いる[2]。後者では全てをオブジェクトで表現できるものの、能動／能動間と能動／受動間で2種類の交信手段が必要となる。

6．2．図書館の例

図10のような図書館の例で、受付は順序化して、つまり一度には一人ずつおこなう。ユーザからの要求は本の借り出しと返却である。既に本を借りている人には二冊目は貸さない。内部には本の原簿とユーザの原簿があり独立に管理されている。要求は

＜"借出"　利用者名　本の名＞
＜"返却"　利用者名　本の名＞
の形でくる。本を借りたいとする要求がきて、在庫があり、そのユーザにとって二冊目ではなくすぐに借りられた場合、制御の流れは図11のようになる。

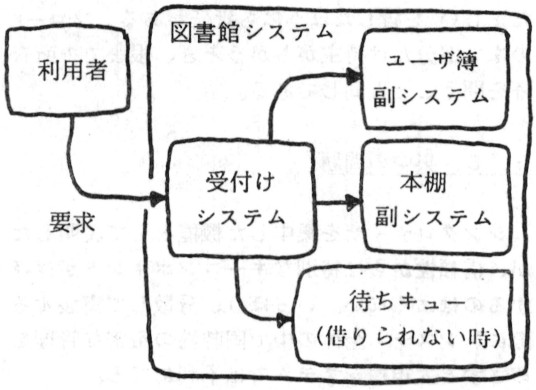

図１０　図書館の例(構造)

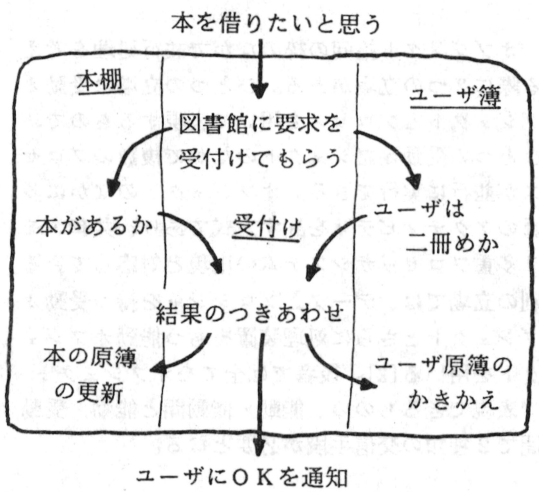

図１１　図書館の処理の流れ

６．３．検討

問題となるのは、使用者からは要求をすれば応答が返ってくる普通の手続きに見えるが、内部はマルチアクティビティであり、原簿の更新などは応答を返した後に実行していることである。また、遅延評価のためのオブジェクトをつくって処理を先おくりするfutureオブジェクトも同じく、その活動はどちらの行為か、 たましい の所有権の問題として統一できる。ただ、オブジェクトの生成コストを考慮しないといけない。

７．まとめ

マルチプロセッサ環境という距離の概念を無視できない状況下で、計算実体の表現を形あるものとないものに分離することで、インプリメントに際しての見通しの良い展望をえることができた。

また、計算前線がより頻繁に分離・収束するオブジェクトを基礎とする処理系への応用についても同様な議論の適用を試みた。オブジェクト・システムの実現方式との対応づけは今後の課題である。

謝辞

東京大学工学部計数工学科森下研究室の方々との討論は本稿をまとめる上で大変有益であったので、ここに感謝します。

参考文献

［１］永松、森下：マルチプロセッサ間でのメッセージ通信オーバヘッドの短縮について，第33回情報処理学会全国大会論文集, 4C-10(1986).

［２］所：オブジェクト指向並列計算に関する一考察，日本ソフトウェア科学会第５回大会論文集, C6-4, pp. 289-292(1988).

［３］柴山，米澤：並列オブジェクト指向言語ＡＢＣＬ／１(2)、ｂｉｔ，vol.20 no.8，pp. 940-954(1988).

［４］永松，神徳，ワタリ，森下：マルチプロセッサ上のイベントとプロセスの管理を行う機構、第35回情報処理学会全国大会論文集，4C-4、pp. 153-154(1987).

［５］B. Beck, B. Kasten, S. Thakker, "VLSI assist for a Multiprocessor," ACM/IEEE Proc. ASPLOS 2nd Conf., pp. 10-20(Oct. 1987).