オペレーティング・システム 51－1
（１９９１．７．１８）

マルチコンピュータ上で並行オブジェクト指向計算を支援するための通信サービスの設計

国生 泰広　　土居 範久
慶應義塾大学 大学院 理工学研究科 計算機科学専攻

並行オブジェクト指向言語の一つであるABCL/c+のプログラムを直接結合型マルチプロセッサ機械上で実行するための基盤の一つとして、オブジェクト間通信機能を提供するABCL/c+オブジェクト・スケジューラを設計した。ABCL/c+のselect文のセマンティクスをほぼ完全にサポートしたメッセージ受信機能と非同期送信機能を持ち、割込みの発生に対する高速な応答とオブジェクトの効率的な状態管理、スケジューリングに留意した設計がなされている。

A Design of a Communication Facility to Support Object-Oriented Concurrent Computation on Multi-Computers

Yasuhiro Kokusho, Norihisa Doi
Department of Computer Science, Graduate School of Science and Technology,
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223 JAPAN

As an attempt to construct an infrastructure for executing the programs of ABCL/c+ (an example of Object-Oriented Concurrent Languages) on the multi-computers, we designed the "ABCL/c+ Object-Scheduler" that facilitates inter-object communications. It supports the message receiving facility that almost completely implements the semantics of the select primitive of ABCL/c+ as well as the asynchronous message sending facility. Also, it is designed to take into account rapid response of the interrupts and the efficient management of the object's state information and scheduling.

1 まえがき

現在、並行オブジェクト指向計算モデルに基づいたプログラミング環境の実装例の多くのものは逐次型の処理系の上に実現されており、分散環境や並列機械上での本格的な実装が今後の課題である。

本研究では、並行オブジェクト指向言語の代表的な実装例の一つである ABCL/c+[土居 87][Doi 88] の分散並列化へ向けての一つの試みとして、並行オブジェクト指向言語をマルチプロセッサ機械上に実装する際に必要となるプログラム実行環境である "ABCL/c+ オブジェクト・スケジューラ" を設計した。このシステムは以下の二つの機能を合わせ持っている。

1. ABCL/c+ 言語のセマンティクスを反映した、言語実装のための基盤となるオブジェクト間通信サービス。これには、異なるプロセッサ要素上のオブジェクト同士の通信に必要となるプロセッサ間通信のためのプロトコル・ハンドラの機能を含む。
2. 通信サービスと連動してオブジェクトの実行をスケジュールするスケジューラ。

2 基盤となるハードウェアとシステムの設計指針

基盤となるハードウェアの特徴を述べ、それに応じてシステム設計の際に採るべき方針を明確にする。

2.1 ハードウェアの概要

基盤となるハードウェアは HyperCube 機械 [Fox 88] [DeCegama 89] 等に代表されるプロセッサ直接結合型のマルチプロセッサ計算機である。個々のプロセッサ要素はそれぞれのローカル・メモリを持ち、通常の分散環境よりも十分に高速で信頼性のあるプロセッサ間結合で Point-to-Point で結ばれており、全二重通信が可能である。以後、このプロセッサ間結合を Direct Link と呼ぶことにする。

このようなハードウェアの特徴として、以下の事柄に注目した。

1. 回線へのアクセス権を巡るコンテンションとパケットの紛失および再送の発生が無く、通信速度が分散環境よりも十数倍以上速い (UNIX ネットワークの ICMP のラウンド・トリップ時間との対比)。さらに Direct Link のトポロジによっては、一つのプロセッサ要素に多数の Direct Link が接続されている場合がある。したがって、Direct Link からのデータ受信割込みの発生頻度が極めて高い。
2. 各プロセッサ要素は一般にコンソールやユーザ端末を持たないので、プログラム実行のために最低限必要とする以上のコンテキスト・スイッチは全て排除できる。これは CPU のスループットを向上させるために重要である。

本研究で実装の対象とする機械は intel 社の iPSC2 を考えている。iPSC2 はローカル・メモリと Direct Link のインタフェース回路、i80386CPU、浮動小数点演算プロセッサ等から成るノードから成り、各ノードは HyperCube 結合の Direct Link で結ばれている。

2.2 システム設計における要点

前節で述べた事柄から、システム設計において考慮した点は以下の通りである。

1. Direct Link からのデータ受信によって割込み駆動で実行される処理は効率良く実行され、またカーネル・コード内で割込みがロックアウトされている時間はできる限り短くしなければならない。
2. コンテキスト・スイッチの発生頻度はできる限り少なくする。
3. カーネル・コードのサイズをできる限り小さくし、プロトコル・ハンドラとスケジューラが素早く連動できるようにするために、両者は最初から一つのプログラム・モジュールとして設計し、オブジェクト間の同期とスケジューリングのために必要な状態情報はできる限り共有させる。
4. データリンク・レベルのフロー制御を行なう場合、ウィンドウ制御方式 [Comer 88] ではなく、最大転送長の長さを持つ固定長フレームを各々送信先のプロセッサとのハンドシェイクによってバックグラウンドで行なうとともに、ハンドシェイクによって送信先のプロセッサ要素の計算負荷や記憶容量を情報として得る。この情報に基づいてフレームを送信する時間間隔を制御する。

3 並行オブジェクト指向言語 ABCL/c+ におけるオブジェクトのモデル

ABCL/c+ はアクタ・モデル [Agha 86] に類似したセマンティクスを持ち、並行して実行されるオブジェクトによって計算が行なわれる。オブジェクトは他のオブジェクトにメッセージを送信することができ、到着メッセージと受信側のメッセージ・パターン式とのパターン・マッチングにより同期をとる。各オブジェクトは、休眠、活性、待機の三つの状態の内の一つをとり、それは、図 1 の状態遷移図に従って変化する。

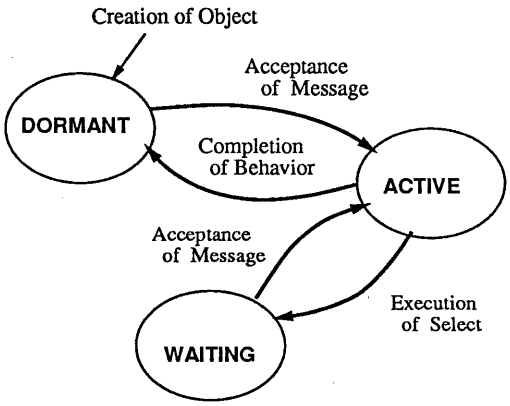

図 1: オブジェクトの状態遷移

オブジェクトは生成直後は休眠状態にあり、受信可能なパターンを持つメッセージが到着すると活性状態に移行する。このとき、それ以前に到着した受け入れ不能のメッセージは捨てられる。活性状態では、到着したメッセージはオブジェクトのキューに繋がれ、select を実行すると、キュー内のメッセージの中から受け入れ可能なメッセージを先頭から探して行く。受け入れ可能なメッセージがキュー内に見つからない時には、オブジェクトは待機状態になる。メッセージ送信は基本的にはアクタ系の言語と同様に継続点の指定可能な非同期式送信であり、ABCL/c+ ではこれを過去型のメッセージ送信 (図2 a) と呼んでいる。アクタ言語と同様に、幾つかのメッセージ・パターン式の中のどれか一つとパターン・マッチングの成功したものを受け入れる選択的な受信機能を持つ。ABCL/c+ ではこの選択的受信機能は select 文として定義されている。ABCL／c+ にはこの他に、図2b, 図2c に示した現在型および未来型の通信方式がある。さらに、ABCL/c+ のオブジェクトにおいて割込み処理を実現するための機能として、オブジェクトの状態が活性状態の時でもパターン・マッチングが成功すれば常にメッセージを受け入れ、それに対応した挙動をオブジェクトが割込み駆動で実行できる通信方式である速達モード通信が可能となっている。

図3 に ABCL/c+ のプログラム例を示す。上記の例では、Spooler という名のオブジェクトを定義しており、state 文の内部において受信した任意のメッセージを画面に出力するオブジェクトを定義し、Printer という変数に代入している。state 文内部で宣言された変数はそのオブジェクトの内部状態を表す静的な変数であり、状態変数と呼ばれている。また、select 文の

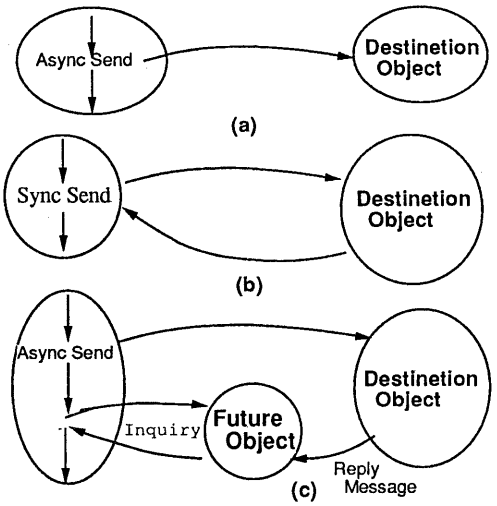

図 2: ABCL/c+ のメッセージ送信方式

```
[object Spooler
   state {
       oid Printer := [object
                              script {
                                   (=> M
                                     char *M;
                                     {
                                      printf("%s\n",M);
                                     })
                              }];
   }
script {
   (=> :StartPrinting
     {
      oid SenderObject := Sender;
       [select-loop
         (=> :END where(SenderObject==SelectSender)
                  {
                     Exit();
                  };
         (=> Message where(SenderObject ==
                                  SelectSender)
                char *Message;
                {
                    [Printer <= M];
                 }})];
         (=> :GetStatus
                {
                   ......
                })
                  .......

}]
```

図 3: ABCL/c+ のプログラム例

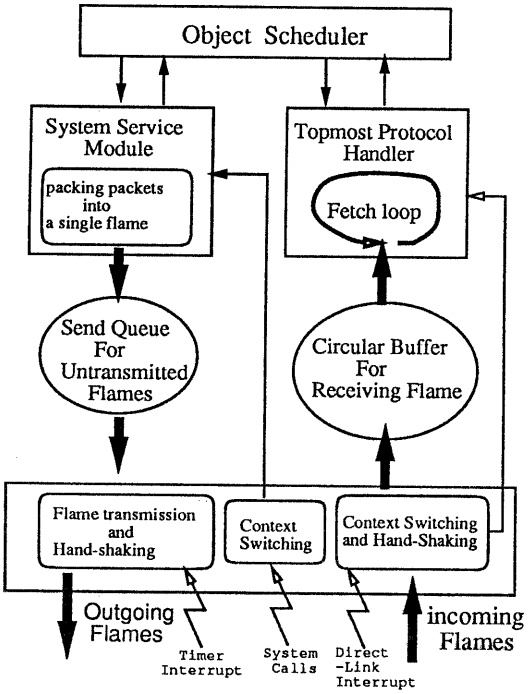

図 4: ABCL/c+ オブジェクト・スケジューラの構造

中の [Printer <= M] は変数 Printer で示されるオブジェクトへメッセージ M を過去型で送信することを意味している。メッセージ・パターンに続いてキーワード where で始まる式はメッセージ受け入れのための条件式であり、パターン・マッチングに成功し、かつ、この式の値が真になることが必要である。

4 ABCL/c+ オブジェクト・スケジューラのソフトウェア構造

2. で述べたような設計方針に従って設計した ABCL/c+ オブジェクト・スケジューラの概略を述べる。

4.1 ソフトウェア構造の概観

ABCL/c+ オブジェクト・スケジューラのカーネル・コードは二層構造となっており、下位層はデータリンク・フレームを Direct Link を通じて送受信しフロー制御と上位層の起動を行なう部分である。Direct Link を通じて他のノードからフレームを受信すると CPU に割込みを掛け、その時にユーザ・レベル・コードが実行中であったならば、上位層の起動の前にコンテキスト・スイッチを行なう。フレームの送信手続きはタイマ割込みによって起動され、送信キューに繋がれている幾つかの送信待ちパケットを取り出して、送信フレーム内にパックし送信する。効率上の問題から受信フレーム用のバッファ領域は予め静的に割り当てられた固定長の領域を 1024Byte 単位で区切り、巡回バッファとして使用する。上位層実行中にフレーム受信割込みが生じた場合は上位層を再帰的に呼び出さずに、受信フレームのバッファリングのみを行なう。上位層は、受信フレーム領域にバッファリングされている全てのフレームを取り出して解釈し終ってから、処理を終える。(図 4)

上位層は、受信フレームを幾つかのパケットに分解し、個々のパケットのヘッダを取り出して最上位プロトコル・ハンドラ (TPH) を呼び出す。TPH は後述する同期エントリと呼ぶデータ構造を用いてオブジェクトの同期状態の更新や到着したメッセージのパターン・マッチング、メッセージのユーザ領域へのコピーなどの処理を行なう。

ABCL/c+ オブジェクト・スケジューラはこの他に、ユーザ・プログラムからのシステム・コールによって起動されるシステム・サービス・モジュール (SSM) を含む。

TPH や SSM が呼び出された際には、それにひき続いて、オブジェクトの実行のスケジューリングを行なうスケジューラが呼び出される。

4.2 オブジェクト間通信のための *select* プリミティブとメッセージの非同期送信機能の実現

ABCL/c+ オブジェクト・スケジューラの TPH と SSM は、ABCL/c+ プログラムがオブジェクト間通信を行なうのに必要な、メッセージの select 受信プリミティブと非同期送信プリミティブを実現している。

特に、ABCL/c+ の select は非常に複雑なセマンティクスを持つものであるが、このセマンティクスの殆ど全てをシステム・コールとしてカーネル・コードで実現すれば、ユーザ・プログラムの大きさを小さくでき、実行速度の最適化を図るのにも都合がよい。

ここで、実際の OS における入出力の select の例を見てみる。メッセージ受信における select 機能として 4.3 BSD UNIX[Samuel 89] 等では、socket[Sun 88a] としてオープンした幾つかのファイル・ディスクリプタに対して選択的入出力を行なうための select システム・コールが提供されている。

BSD UNIX の select はビットマップ形式で指定された幾つかのファイル・ディスクリプタの中で読み書きが可能なものが一つ以上ある場合、そのファイル・ディスクリプタに該当するビットマップのビットをセ

ットする。これを実現するために、個々のファイル・ディスクリプタに対して図5のデータ構造が与えられている。図5のFile Descripter Tableの各エントリはファイル・ディスクリプタに対して実行可能な手続きを指すポインタを幾つか含んでおり、さらにそのファイルがsocketとしてオープンされていた場合、socket構造体というsocket入出力制御用のデータ構造へのポインタを持たされている。このsocket構造体へのアクセスはFile Descripter Tableエントリ内に含まれる手続きを通してのみ可能である。

あるプロセスがUNIXのselectシステム・コールを実行した場合、図5に示すPolling手続きが起動され、各File Descripter Tableエントリ内の*fo_select*()プロシージャを順番に呼び出し、そのファイル・ディスクリプタが入出力可能であるかを調べ、ビットマップを更新する。selectしている全てのディスクリプタが入出力不可ならば、プロセスはsleepし、socket構造体内にsleepしたプロセスのスケジューリング・ブロックへのポインタが記録される。その後、デバイス割込みの発生により入出力の状態が変化すると、その都度Polling手続きが起動されると共に、割込みの対象となったsocketのsocket構造体に記録されていたプロセス・スケジューリング・ブロックへのポインタを用いて、sleepしていたプロセスを起こす。

ここで述べた、UNIXのselectは非常にオーバーヘッドの大きな処理である。その主な原因は、File Descripter Tableエントリによって抽象化された形でしかsocket構造体にアクセスできないことや、selectしている全てのFile Descripter Tableエントリについて、個別に*fo_select*()を呼び出す必要があるためである。

そこでABCL/c+オブジェクト・スケジューラでは、図6に示すようなオブジェクト・スケジューリング・ブロックおよびselecting listというデータ構造を定義する。オブジェクト・スケジューリング・ブロックはスケジューラがオブジェクトを優先順序つきのラウンドロビン・スケジューリングによってスケジュールするためのデータ構造であり、個々のオブジェクトが記憶管理のためにローカルに使用するセグメント表を所有している。selecting listは同期エントリというデータ構造を要素とするリストであり、リストの要素がさらにselecting listを指している場合もある。図6は3.で示したSpoolerオブジェクトのプログラム例に対応するselecting listである。全てのselecting listはいずれかのオブジェクトに所有されており、各同期エントリはそのオブジェクトがメッセージのselect受信を行なう際に宣言するメッセージ・パターン式と一対一に対応している。各同期エントリはメッセージ・パターンを表すバイト列、パターン・マッチングの際に評価されるwhere式の手続きへのポインタ、パターン・マッチングが成功した時にCPUにセットされるコンテキスト、受信メッセージバッファへのポインタなどを含む。

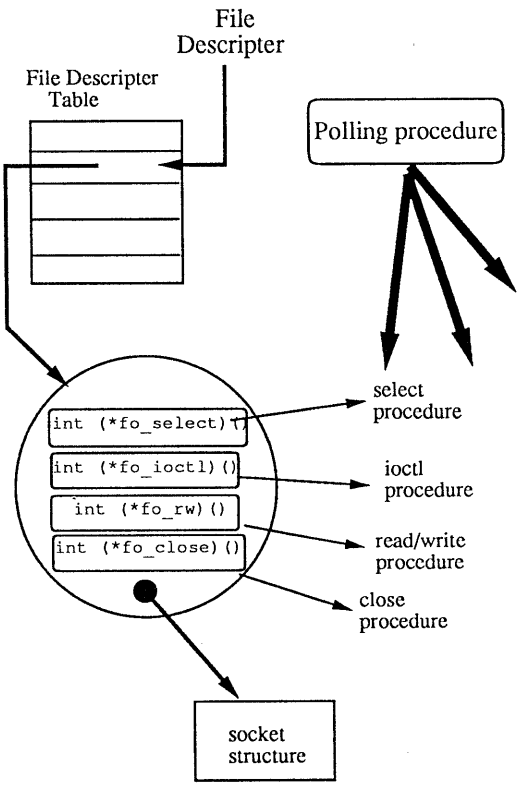

図5: UNIX socketのselect機構

同期エントリには、受信用同期エントリと送信用同期エントリがある。selecting listを構成する受信同期エントリの中で、同時にselectされるメッセージ・パターンに対応するものは全てselecting listの同じ階層内に属する。selecting listを所有しているオブジェクトのオブジェクト・スケジューリング・ブロックは、そのselecting listの中で現在有効となっている部分の先頭へのポインタを持っている。Spoolerオブジェクトの例でいうと、オブジェクトの初期の状態では、ポインタは図6の*hdr_1*で表されるフィールドを指し、:StartPrintingメッセージを受け取った後に実行されるselect-loop文を実行した時には、ポインタは*hdr_2*を指す。オブジェクトが受け入れ可能なメッセージの到着を待ってブロックしている時には、このポインタによって指される部分に属する受信同期エントリはSuspendingEntryキューというキュー内に記録されており、受信同期エントリはさらにオブジェクト・スケジューリング・ブロックのアドレスを保持している。

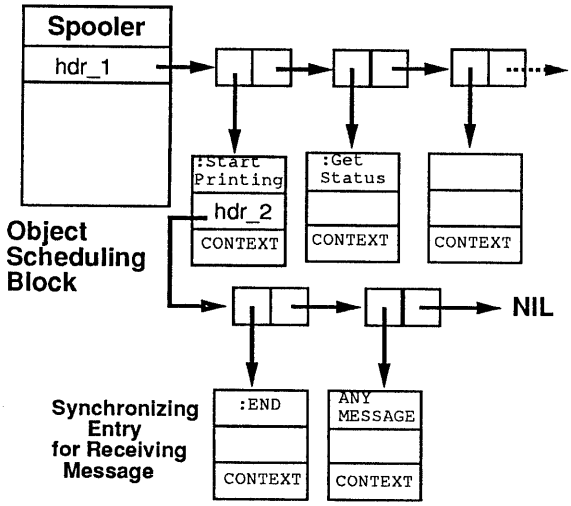

図 6: オブジェクト・スケジューリング・ブロックと Selecting List

一方、オブジェクトが非同期メッセージ送信を行なおうとする時には、送信メッセージと同じメッセージ・パターンを表すバイト列データを含む送信用同期エントリを selecting list 内に生成する。この他に、各オブジェクトはセッション・レコードというデータ構造を所有しており、これは遠隔オブジェクトとのメッセージ通信を行なう際に TPH によって通信制御のための状態情報を保持するために使われる。セッション・レコードと対応する同期エントリはポインタで互いに参照しあっている。このように、オブジェクト・スケジューリング・ブロックを中心に、そのオブジェクトの状態を表す全てのデータ構造が互いに直接参照可能になっている。

オブジェクト同士がメッセージ通信とパターン・マッチングによって同期をとる動作は、対応する送信者オブジェクトの送信同期エントリと受信者オブジェクトの selecting list 内で有効となっている受信同期エントリとの間でパターン情報同士のマッチングが成功することによって実現される。同じノード上のオブジェクト同士のメッセージ通信は以下のように行なわれる。

1. 送信同期エントリは SuspendingEntry キュー内からパターン情報のマッチングが成功するものを検索し、該当するものがあればステップ 3 に移る。そうでなければ、ステップ 2 へ移る。

2. 送信同期エントリを、受信者オブジェクトのオブジェクト・スケジューリング・ブロックの持つメッセージ・キューに繋ぎ、ステップ 4 に移る。

3. 検索した受信同期エントリ内に記録されているコンテキストとメッセージ受理後に実行されるべき挙動ルーチンのエントリーポイントを受信者オブジェクトのローカル・スタックに積む。

4. スケジューラを呼び出し、スタック・ポインタを、新たにスケジューリングされたオブジェクトのローカル・スタックにセットし、割込みリターン命令を実行することにより、そのオブジェクトの実行を再開する。

ここで、ステップ 3 の挙動ルーチンの起動方式は、UNIX に於ける signal ハンドラの起動方式と類似していると共に、受理可能なメッセージの到着イベントを契機とする一種のイベント駆動方式でもある。したがって、速達モードでのメッセージ受信によるユーザ・プログラムへの割込みを実現する場合も、特別なことをする必要はない。通常のメッセージ受信による挙動部の呼出しと速達モードによる場合の違いは単にメッセージ受け入れイベントを直列化するかしないかの違いだけである。

4.3 遠隔オブジェクト間メッセージ通信を実現するためのプロセッサ間通信プロトコル

遠隔オブジェクト間通信は以下の手順で実行される。ここで、送信オブジェクト a がノード A に、受信オブジェクト b がノード B 上にあるとする。

1. a がカーネルに対して遠隔オブジェクトへのメッセージ送信を要求すると、TPH は新しいセッション・レコードを生成し、PendingSession テーブルという配列に登録すると共に、b の識別子と送信メッセージのパターンを表すバイト列情報、セッション ID 番号を含む BIND パケットを B へ送信する。

2. ノード B の TPH が BIND パケットを受信すると、1. で生成されたセッション・レコードと対になるセッション・レコードが生成され、BindedSession テーブルという配列に、登録され、それに対応する送信用同期エントリの B 側コピーを生成し、セッション・レコードと互いに参照を可能にする双方向のポインタを張る。

3. ノード B 上において 4.2 で述べた手順に従い、送信用同期エントリのコピーを用いてパターン・マッチングを行ない、それが成功したらセッション ID

番号および、BindedSession テーブル・インデックスと共に、ACK パケットを A へ送信する。送信同期エントリのコピーがオブジェクト・スケジューリング・ブロックの到着メッセージ・キューに入れられた場合には、RETAINED パケットを送信し、キューへの登録を拒否された場合には REJECTED パケットを送信する。キューに入れられた送信同期エントリが後に受理されたら、その時点で ACK を送る。

4. ノード A は、ACK パケットが返ってきた時点でパケット内のセッション ID から PendingSession テーブルのインデックスを計算し、セッション・レコードを得る。タイムアウトや REJECT の受信があると、送信同期エントリに登録されている例外ハンドラを起動して、通信の失敗を報告する。

5. ステップ 4. で得られたセッション・レコードの情報に従って、メッセージ・データ本体を必要に応じて幾つかのパケットに分解して送信する。

6. ノード B から受信済みのデータ・パケットの集合を表すビットマップが送られてくるので、セットされていないビットに対応するデータ・パケットを再送する。

7. A で全てのデータ・パケットの到達が確認されたらセッション・レコードを削除し、処理を終る。

上記のプロトコル手順で、PendingSession テーブルや BindedSession テーブルの検索は、Sun RPC [Sun 88] や Cedar RPC [Birrell 83] プロトコルに見られる、Binding エージェントへのポート・アドレス問い合わせの手続きと同様のものであり、できるだけ高速に実行される必要がある。

ここで、TPH はスケジューラと完全に連動しているという点が効率上の利点となっており、オブジェクト・スケジューリング・ブロックを通して、通信制御に必要な全ての状態情報をスケジューラとプロトコル・ハンドラの間で共有できるようになっている。UNIX では socket 層やプロトコル・モジュールがスケジューラとは別のモジュールとして設計されているので、通信制御情報とスケジューラ内のプロセス・スケジューリング情報を対応させるには、面倒な検索処理を介さねばならない。

5 検討および現状

5.1 検討

オブジェクトが同期をとる際の受信同期エントリの検索と送信同期エントリとの間のパターン・マッチングは非常に時間の掛かる処理であり、重点的に効率の最適化が図られるべきである。実際、受信同期エントリをキューの先頭から単純に検索する方法をとると、TPH や SSM の実行時間の 70 パーセント近くをこの処理に費やすことになり、効率上問題がある。解決策の一つとして、メッセージ・パターン情報から二つのハッシュ値を計算し、各々の値から対応する受信同期エントリのアドレスのページ・フレーム番号とオフセットを得る方法などが考えられる。

5.2 現状

現在、ABCL/c+ の実装作業は AppleTalk ネットワークで結ばれたマッキントッシュ上で動作する HyperCube 機械のシミュレータ上で行なっていたものが完成した状態であり、iPSC2 上での実装のために、実行効率向上のためのアルゴリズムの最適化と性能測定を行なっている。TPH がフレーム受信割込み発生から処理を終えるまでの所要時間は、受信同期エントリをハッシュを用いて検索した場合、およそ 3 から 6 ms であり、ハッシュを全く用いない場合、8 から 20 ms 前後も掛かってしまう。また、バッファの割り当てルーチンは内部でマッキントッシュ OS のメモリ・マネージャを呼び出しているので、iPSC2 上の実装ではこの点で改良の余地があると思われる。

また、マッキントッシュ上での TPH の平均実行時間とネットワークの通信速度の比率についてみてみると、AppleTalk 上のデータリンク・レベルの通信のラウンド・トリップ時間が約 20 から 25ms であり、最大で 1 秒間に 40 から 50 回の割込みが発生する。TPH の 1 回の実行時間を 10ms 以下程度に抑えることは必要不可欠であることが分かる。iPSC2 では、Direct Link の通信速度は 2.8Mbyte/sec 程度であり、i80386CPU の速度が 4MIPS 程度であるから、フロー制御や TPH の実行効率は今後非常に大きな問題となると思われる。

6 まとめ

直接結合型マルチプロセッサ機械上で ABCL/c+ のプログラム実行の際に必要となるオブジェクト間通信機構を実現するための ABCL/c+ オブジェクト・スケジューラを設計した。同期、通信制御、記憶管理、およびスケジューリングに関するオブジェクト固有の状態情報をオブジェクト・スケジューリング・ブロックを中心としてひとまとまりの自己完結なデータ構造とした。そのため、オブジェクトの位置独立性が高まり、同期と通信に必要な全ての情報をスケジューラとプロトコル・ハンドラの間で共有でき、Direct Link からの割込みに対して、両者が素早く連動できる。Direct

Link は十分に信頼性の高い通信路であり、遠隔オブジェクト間通信のためのプロトコルは、メッセージの到達保証に関して最低限の機能のみ実現するように設計した。

参考文献

[土居 87] 土居範久, 児玉靖司. 並行オブジェクト指向言語 *ABCL/c+* 仕様書, *1987.*

[Doi 88] Doi, N., Kodama, Y., Hirose, K. *"An Implementation of an Operating System Kernel Using Concurrent Object-Oriented Language ABCL/c+", ECOOP,1988,pp.250-266.*

[DeCegama89] Angel L.DeCegama. *"The Technology of Parallel Processing", Vol.1, Prentice-Hall, 1989.*

[Fox 88] Fox Johnson,Lyzenga Otto, Salmon Walker. *"Solving Problems on ConcurrentProcessors ", Vol.1, Prentice-Hall, 1988.*

[Samuel 89] Samuel J.Leffler,Marshall K.McKusick. *"4.3BSD UNIX Operating System", Addison-Wesley,Raeding,Mass., 1989.*

[Comer 88] Douglas Comer. *"Internetworking with TCP/IP:Principles,Protocols,and Architecture", Pretice-Hall, 1988.*

[Sun 88] *"Remote Procedure Calls:Protocol Specification",Sun Microsystems 800-1779-10, May(1988).*

[Sun 88a] *"Socket Based IPC Implementation Notes", Sun Microsystems 800-1779-10, May(1988).*

[Birrell 83] Birrell,Nelson. *"Implementing Remote Procedure Calls", Xerox CSL-83-7, October(1983).*