並列通信ライブラリ 参考資料

黒田 圭

平成８年４月２４日

Part I
Parallel Virtual Machine

1 PVM の概要

PVM (Parallel Virtual Machine) は、さまざまなネットワーク上のシリアル、パラレル、ベクトル計算機などの異機種の計算機の集合を一つの大きな分散メモリ型の計算機 (Virtual Machine) として使用することを可能にしたソフトウエアシステムである。PVM は、バーチャルマシンにおいて自動的にタスクを起動する機能を備えており、タスク間の通信および同期を実現する。タスクは PVM における計算の単位であり、UNIX におけるプロセスに類似したものである。

2 PVM の機能・特徴

ユーザは C または Fortran を用いてアプリケーションを記述する。その際、共通なメッセージパッシングライブラリを利用して並列化を行なう。アプリケーションを構成する複数のタスクは協調して動作し並列に計算を行なう。

また、PVM のもとでは、アプリケーションを構成するタスクは、問題にもっとも適したアーキテクチャを利用することができる。PVM は、異なるコンピュータ間の整数あるいは浮動小数点の表現の違いを吸収するための、データ変換を行なうことができる。

比較的コンパクトなシステムであることもその特徴である。

3 PVM の構成

PVM は大きく分けて pvmd と libpvm の二つの部分から構成されている。pvmd とは、PVM のデーモンプロセスであり、Virtual Machine を構成

するすべての計算機に常駐する。

- デーモンの例 :sendmail ... sandmail は、UNIX システムに常駐し、メイルの送信および受信を扱う。

PVM アプリケーションを実行する場合、まず最初にユーザはどれか一つのコンピュータで pvmd を起動する。次にこの pvmd は、ユーザが定義したバーチャルマシンを構成するコンピュータそれぞれにおいて順次 pvmd を起動する。最後に、どれか一つのコンピュータに表示された UNIX プロンプトに対してコマンドを入力することにより、PVM アプリケーションを実行する。複数のユーザが互いのコンピュータをオーバーラップさせてバーチャルマシンを構成してもよく、もた、各ユーザは一人で複数の PVM アプリケーションを同時に実行することも可能である。

また、libpvm は、PVM のインタフェースルーチンのライブラリであり、メッセージの送受信, プロセスの実行、などに関するルーチン群を含んでいる。ユーザが PVM のアプリケーションを実行するためには、ユーザが定義した Virtual Machine を構成する各計算機上で pvmd を動作させ、さらに libpvm とリンクされなければならない。

4 主な関数の例

- pvm_mytid : そのプログラムが pvm を使用することの宣言およびその識別子の取得

  使用例 :mytid = pvm_mytid()

- pvm_exit() : pvm の終了

  使用例 :info = pvm_exit()

- pvm_send() : メッセージ送信実行

  使用例 :info = pvm_send()

- pvm_recv() : メッセージ受信

  使用例 : bufid = pvm_recv()

5 PVM における問題点

PVM はさまざまなネットワーク上の異機種の計算機を複数台用いることにより、並列計算機に匹敵するほどの処理を行なうことができる環境を提供している。しかし、PVM はその構成によっては、使用している計算機の台数に比べて処理時間があまり短縮されない場合がある。その原因としては、

- 計算機の処理能力の違い
- プロセス間の通信時間

が挙げられる。

Part II
Message Passing Interface

6 メッセージ・パシング方式とは

メッセージ・パシング方式は、メモリを分散して配置する並列処理マシンで広く使われてきた。

分散メモリ型の並列マシンでは、各プロセッサ上で並列に動作するプロセスが、それぞれ独立したアドレス空間を持つ。各アドレス空間は、分散して配置したメモリ上に展開する。メッセージ・パシングは、こうした状況で行われるプロセス間通信の一形態である。メッセージは、データと制御情報からなる。送信側のプロセスは送ろうとするデータを含むメッセージを作って、受信側のプロセスに渡す。受信側のプロセスは受け取ったメッセージからデータを取り出してメモリに書き込む。つまり、メッセージを送受信するときに、送信側または受信側のメモリを直接アクセスしない。

7 MPI とは

MPI とは、Message Passing Interface の略語である。並列処理を行うアプリケーション用のメッセージ通信インタフェースの標準として考案された。この MPI は初めから標準インタフェースとなることを目的に開発されたので、既存のメッセージ通信ソフトウェアの特徴をほとんど取り入れている。プログラムを最適化するためのインタフェースを豊富に用意し、ハードウェアに合わせた実装も可能としている。

8 MPI 登場の背景

既存のメッセージ通信ソフトウェアは、並列処理マシンのメーカーが自社のハードウェア専用に作ったものが多かった。それぞれのハードウェアの特徴を引き出すように作られていたが、他のメーカーのマシンとの互換性は後回しになる傾向があった。

一方、並列処理マシンとは異なる進展を見せたのが、ワークステーション・クラスタである。つまり、LAN でつながった複数のワークステーションを、１台の並列処理マシンと見なして利用する方式である。このワークステーション・クラスタに対応したメッセージ通信ソフトウェアが大学や国立研究機関で開発されて普及し、PVM、Express、P4 などといったいくつかは多くのユーザを獲得した。これらのソフトウェアは、並列処理マシンに移植され、異機種間で共通な通信インタフェースになった。特に PVM は有名になり、事実上の標準の地位を獲得した。

しかし、並列処理マシンのための標準としてみると、PVM には幾つか問題があった。一つは、ワークステーション用のソフトウェアだったため、並列処理マシンへの実装に難しい部分があったこと。もう一つは、PVM は少人数の開発者が作成したので、仕様が開発者によって恣意的に変更される可能性があることである。

そのような背景から、最初から標準となることを目的として MPI は開発された。

9 MPI の特徴

大きく分けて、特徴は二つある。

- 一つは、アプリケーション開発を容易にするために、ライブラリ化を支援する機能を備えること。この機能を使えば、複数のプログラマが開発したライブラリをそのまま組み合わせてアプリケーションを作成しても、正常に動作するようにできる。従来は、ライブラリで使うメッセージ通信と、ライブラリを組み込むプログラム本体で実行しているメッセージ通信とを区別する方法がないので、容易にライブラリ化を行えない、という問題があった。ライブラリ側で送信したメッセージがプログラム本体で、プログラム本体で送信したメッセージがライブラリ側で受信されてしまう可能性がある。そのため、正しい結果を保証できなかった。

- もう一つは、並列処理マシンの特徴を生かしたチューニングをアプリケーションに施せるようにしたことである。このために多くの通信関数が用意されている。既存のメッセージ通信ソフトウェアの多くは、少ないメッセージ通信用関数で、多くのプラットホームに対応していた。このため、ハードウェアに依存した最適化を施すのが難しかった。MPI には、どのような条件下でも通信が成功する汎用性の高い関数や、限定された条件でしか使えない関数がある。利用条件を制限している関数は、条件判定を省けるのでその分処理を高速化できる。場合によって

は、通信ハードウェアの特徴を生かせる場合がある。プログラマは、汎用性のある関数だけを使えば確実に動作するプログラムを記述できる。その上で利用条件を吟味して複数の通信関数を使い分ければ、より高速なプログラムを書くことが出来る。

以上の二つが大まかな特徴である。以下、細かい点についてのべていく。

9.1 MPI のメッセージ通信方式

MPI のメッセージ通信方式は、他のメッセージ通信ソフトウェアと基本的に変わらない。既存のメッセージ通信ソフトウェアでつかわれている優れたインタフェースを多数採用していることに特徴がある。メッセージ通信のときに、送信側プロセスは送信関数を、受信側プロセスは受信関数を呼び出す。ユーザは通信関数の引数として、「通信相手」と「通信内容」を与える。

通信相手としては、プロセス名やメッセージ・タグなどを指定する。メッセージ通信は、同じタグを指定した送信関数と受信関数のあいだで起きる。

通信内容としては、ユーザ・バッファの先頭アドレス、送信するデータの個数とデータ型を指定する。データ型としては、新たなデータ型を定義してそれを渡すことも出来る。(9．6 参照)

9.2 2 種類の通信先指定方法 - グループ、ランク、プロセス・トポロジ

MPI には、通信先のプロセスを指定する方法を 2 種類定めている。

- 一つは、グループと番号で指定する方法である。グループとは、複数のプロセスの集まりである。n 個のプロセスからなるグループでは、0 から n-1 までの番号をプロセスに割り当てる。(これをランクという。) グループとランクを指定すると、プロセスを一意に指定できる。MPI を使って通信するプロセスは、必ず一つ以上のグループに所属することになる。(全てのプロセスが所属するグループ名は、あらかじめ決められている。) グループによる指定を利用すると、複数のプロセスと同時に通信を行うときに便利である。

- もう一つのプロセスの指定方法は、プロセス・トポロジである。グループとランクだけでは、プログラマに理解しやすい形でプロセスを指定できない場合がある。プロセス・トポロジは、プログラマがプロセスの指定方法を自分にとって分かりやすい形で定義するための機能である。例えば、3 次元格子状に空間を分割して計算するプログラムでは、各空間に関係するデータの処理を各プロセスに割り当てることが多い。この場合は、各プロセスをグループ名とランクで指定するよりも、座標値

として指定できたほうがプログラムを作り易い。MPI ではグループごとにプロセス・トポロジを登録することが出来る。特に n 次元格子状のトポロジについては処理が容易になっている。

9.3　多くの集団通信機能

MPI は、複数のプロセッサ間でメッセージを通信する方法を多数用意している。集団通信は、グループ単位で実行される。ある集団通信に参加するプロセスは、全て同じグループに属していなければならないし、また、集団通信を行うグループに所属するプロセスは全て、その集団通信に参加しなければならない。もしも、あるグループ内の一部だけで集団通信をしたかったら、その一部を新しいグループとしてまとめる。また、例えば、ブロードキャストでは、送信側プロセスと受信側プロセスの両方が MPI_BCAST という関数を呼び出すことになる。ブロードキャストされたデータを、MPI_RECV などの 1 対 1 通信用の関数で受信する、というようなことは出来ない。集団通信ではそれに参加する全てのプロセスが同じ関数を呼ぶことになっているからである。

9.4　タグの使用とその問題

MPI では、メッセージにタグをつけて送受信する。同じ値のタグを持つメッセージは、受信側のプロセスは届いた順序で受けとる。タグの値をコントロールすることでメッセージを受ける順番を制御したり、特定のタグの値を持つメッセージだけを取り込むようにできる。(注:タグは整数値で、プログラマが自由に決める。)

しかし、ここで注意すべきことは、複数のプログラマが共同でプログラムを行なっている時に、たまたま同じタグの値をつけてしまう可能性がある、ということである。このようなことを防ぐには、同じプログラムに関わるプログラマ全員の間で、値の指定方法をあらかじめ決めておかねばならない。

9.5　ライブラリ化支援のためのタグの強化 - コミュニケータ

MPI は、タグの他に、メッセージを区別するための、「コミュニケータ」と呼ぶ値を用いる機能をもっている。MPI では、このコミュニケータを通常のタグに加えてメッセージのやりとりを行なう。その際、同じコミュニケータを用いる受信側プロセスだけが受信を行なえる、というようになっている。このようにして、メッセージの送受信関係を限定するのである。ここで当然、そのような識別能力を持つコミュニケータをどのようにして生成するのか、という疑問が生じるわけだが、MPI にはコミュニケータの生成を行なう関数が用意されている。そうした関数によって生成されたコミュニケータは、MPI

を使って通信するすべてのプロセスが共有している。ゆえに、生成したコミュニケータは必ず、既存のコミュニケータと異なるのである。この結果、一つのプロセスは同じコミュニケータを二つ以上持つことができず、混乱は避けられる。プログラマは、メッセージ通信を行なう各ライブラリごとに、異なるコミュニケータを指定すればタグの衝突を避けることができる。

9.6　不連続データの転送

MPI では、送受信するデータを特定のフォーマットに変換するなどの処理を行なう必要はない。アプリケーションによっては、不連続に配置されたデータなどを頻繁にメッセージ通信しなくてはならないことがある。こうした場合に対応して、複雑な配置のデータを新しいデータ型として登録し、メッセージ通信のときに一度に転送する機能を、MPI は備えている。プログラムは、送ろうとするデータの配置を新しいデータ型として定義し、そうした新しいデータ型をメッセージ通信関数の引数として使える。

9.7　通信処理と他の処理の並列化

MPI のメッセージ通信関数は, 関数を実行中のプロセスをブロックするか、しないかによって 2 種類にわかれる。

- ブロッキング通信では, ユーザ・バッファ対する読み書きが完了するまで、次の処理に移れない。その代わり、プロセスがユーザ・バッファを書き換えても、送信用関数を使って送り出されるメッセージに影響はない。受信関数の場合は、ユーザ・バッファにはメッセージによって届いたデータが入っていることが保証される。
- 非ブロッキング通信だと、ユーザ・バッファに対して読み出しや書き込みを終えていなくても、処理は終了する。このため、ユーザ・バッファに対して読み出しや書き込みを行なうと、誤った結果を招く可能性がある。すなわち、受信データで置き換えられる前のデータを読み出したり、送信されるべきデータを書き換えるおそれがある。しかし、データ領域への読み書きをプロセッサに代わって実行する通信ハードウエアがあれば、通信処理と他の処理を並列に実行して性能をあげられる。非ブロッキング通信での読み書きが終ったかどうかをしるために、MPI_TEST や MPI_WAIT といった関数が用意されている。

10　参考文献

- MPI ＞日経エレクトロニクス　１９９４ー４ー１１,PVM ＞ WWW