LB-002

メタプロセスモデル

―明示的並列処理記述のための分散共有メモリプログラミングモデル―

Meta Process Model

- A Distributed Shared Memory Programming Model for Explicit Parallelism Description -

緑川 博子　片野真吾　飯塚 肇

Hiroko Midorikawa　Shingo Katano　Hajime Iizuka

１．はじめに

並列プログラムのための代表的なプログラミングモデルには，MPI，PVM に代表されるメッセージパッシングモデル（MP モデル）と OpenMP，Pthread などの共有メモリモデル（SM モデル）がある．

MP モデルは，MPI が計算機の物理的メモリ構成に拘わらず標準 API として多くのシステムで使用できるようになり，並列プログラムの一般化と普及に大きく貢献した．

一方，SM モデルは，従来の逐次プログラムからの継続性がよく，MP モデルに比べ，煩雑なメッセージ送受信の記述がない点で記述性・可読性が良い．そこで SM モデルの標準化をめざして OpenMP が提案され，幾つかの専用並列計算機で実装されてきている．また最近，安価な並列システムとして広く用いられるようになった計算機クラスタなどにおいても，仮想共有メモリシステムを構築し，その上で OpenMP を利用する研究が行われている．

しかし，計算機クラスタでは，ノードが CPU 速度に比べ遅いネットワークでつながれており，ノード内外のメモリアクセス速度差が大きいため，SM モデル用に提案された API や言語を適応しようとすると，分散メモリを意識したデータの分散割付 I/F がなかったり，メモリ一貫性のための無駄な同期が増加し，速度性能が落ちてしまう．

このような性能低下を防ぐために，OpenMP などに，本来 SM モデルには不要な付加的情報を付け加えて API を拡張し，分散共有メモリにおいても性能を向上させようとする試みがある．しかし，これにより SM モデル API の整合性や統一性が損なわれ，わかり易いはずの API が，ユーザの分散メモリを意識した付加された明示的な指示と，分散メモリを共有メモリであるかのように見せるためにシステムが行う暗黙の処理などがからみ合い，ユーザにとって動きのつかみにくい複雑な API になってしまうことがある．

どのようなシステムでも，同一 API による並列プログラムが効率よく実行できることが理想であるが，現時点では計算機クラスタ上で性能を出すには，データがノード内にあるのか否かをユーザがある程度意識せざる得ない．これを同一に扱う SM モデルを用いて高性能を達成するには自ずと限界があり，分散メモリのために多くの拡張を施した SM モデル API はすでに，厳密には SM モデル API とは言えないのではないだろうか．

そこで本報告では，SM モデルとは異なる新しい***分散共有メモリプログラミングモデル***を提案し，設計した API について述べる．

Seikei University

成蹊大学

２． 明示的な並列処理の記述

SM モデルとして提案されている API や言語は，大きく２つに分けられる．一つは従来の逐次プログラムとの継続性を重んじるもので，他方は共有メモリ上での並列アルゴリズム記述を重視するものである．

例えば，OpenMP や HPF は，従来の逐次プログラムを拡張して，容易に処理の並列化ができるという点が大きな言語上の特徴である．従来の逐次コードに pragma 文という並列化プリミティブを付加することによって，ユーザに最小限のコストで，逐次コードのままで可能な範囲の並列処理を行い，性能をあげることを特徴とする．pragma 文を除けば，そのまま逐次コードになるようなプログラムを対象としている．すなわち，もともと独自の並列アルゴリズムを実装するための言語というより，逐次コードの簡易並列化言語とも言うべきもので，ループ文やベクトル処理の並列化など，規則的で定形な処理の並列化が主目的である．並列アルゴリズムや並列プログラムに不慣れなユーザであっても，暗黙にシステムが並列実行を行うため，並列実行の詳細を知らずにある程度の並列処理を行うことが可能である．しかし，SPMD でない，非定形，非対称な並列アルゴリズムを記述するのに適しているとはいえない．

一方，pthread プログラミングなどは，もともとプログラマが並列処理を意識してプログラムを書かなければならず，たまたま実行時にスレッドが１つで実際には逐次実行になることはあっても，あくまで並列プログラムを書くための API で，並列プログラム作成者向きといえる．並列アルゴリズムを実装しようとする並列プログラマにとって，SM モデルの書きやすさ，読みやすさは重要であるが，並列処理を明示的に自由に記述したい時，逐次コードとの関わりは不要であるばかりか，上記の OpenMP や HPF のような実装システムによる暗黙の並列処理は，むしろ明示的並列記述の妨げになり可読性が損なわれる．

並列アルゴリズムの記述には，従来の逐次コードにはない並列処理のためのプリミティブが必要ではあるが，実装に依存した特殊な記述は避けるべきである．ここで提案するモデルでは，メモリ一貫性同期，排他制御として，バリア，ロック，条件変数など，pthread や代表的なソフトウエア DSM でもよく知られている API を用いて，並列処理の記述を明示的にプログラマが行うためのモデルである．

また，このモデルでは，従来の SM モデルにはない（あるいは本来あるべきはでない），メモリの配置，共有か否かを明示的に記述するための API が提供されている．

３．分散共有メモリモデル：メタプロセスモデル

図１のように，クラスタ上での並列処理は各計算機ノードにあるプロセスが複数で共同して，一つの処理を行う．ここで提案する実行モデルでは，ある処理を共同で行う複数のプロセス群全体を，***メタプロセス***と呼ぶ．メタプロセス中のプロセスは，通常のプロセスと同じで，ファイルやメモリなどの資源に関連づけられた OS 上の一つの実行単位である．一方，メタプロセスはここで導入した新しい概念で，通常 OS では認識されないが，分散共有メモリ機能をサポートするなんらかのシステムソフトウエアなどによって認識されるユーザにとっての実行単位で，一つの応用処理に対応する．

3.1 プロセス独立実行による非定形並列処理

メタプロセスとしては，各プロセスが同じ処理を行う SPMD 型でもよいし，各プロセスが異なる処理を行う MPMD 型も可能で，柔軟性がある．それぞれのプロセスは独立に実行を行い，プロセス間の通信・干渉は，ロック，バリア，共有データなどのプログラマの明示的な指示により行われる．したがって，従来の MP モデルによる記述のように，自由な並列処理記述が可能である．

メタプロセスを構成するそれぞれのプロセスでどのような処理をするのかは，***メタプロセス記述ファイル***で指定する．このファイルは，メタプロセス（応用）を構成する各プロセスで実行するプログラムファイル名や引数，SPMD 型の場合には並列度などを指定する．ここで指定するプロセスとは仮想的な論理プロセスで，実際の計算機ノードやホスト名に依存する記述ではない．ここでは，並列アルゴリズムとは無関係な，プロセスと物理的なプロセッサとの関連付けは行わない．

3.2 メタプロセス内プロセスの共有アドレス空間

このモデルでは，プロセスとメタプロセスの２階層の実行単位を導入し，それぞれのレベルのスコープを持つ２種のデータを利用できる．一つは従来のプロセス内で扱うデータで，もう一つはメタプロセス中の複数プロセスで共有する大域（共有）データである．新しく導入した共有データに関しては，すべてのプロセスにおいて同一アドレスが一つの共有データを指しており，従来の SM モデルと同様に，共有データのアクセスに煩雑なメッセージ送受信記述が不要である．

図２に実行時におけるメモリのスコープを示す．１プロセス内の変数スコープは，ソースプログラムの記述により制御され，global，local 変数などの種類によりスコープの有効範囲と寿命が定められている．これらの変数を直接他のプロセスからアクセスすることはできない．このモデルではすべてのメタプロセス内プロセスから参照可能な ***shared*** 変数を新しく定義する．

3.3 共有データの緩和型メモリ一貫性

共有データのメモリ一貫性は，従来のソフトウエア DSM などで用いられる，緩和型メモリコンシステンシモデルを前提にする．バリア，ロックなどを使い，ユーザが明示的にプログラムに記述する．

3.4 共有データのプロセス分散割付 API

共有データを各プロセスに関連づけるための API を提供する．各プロセスでの処理で最もアクセスの多い，関連が深い共有データを指定することにより，性能の向上が期待できる．そのような関連性が特になかったり不明な場合には，割付指定を省略でき，システムが自動的に１プロセスに集中しないように毎回異なるプロセスに割り付ける．

スカラーデータの指定以外の配列データに関しては，一定サイズの領域に分割して，サイクリックに指定したプロセス群に割り付けることができる．例えば応用に応じて，ライン型，バンド型，タイル型などと共有データ宣言時に割り付け指定することもできる．また実行時に malloc 形式での割付指定も可能とする．

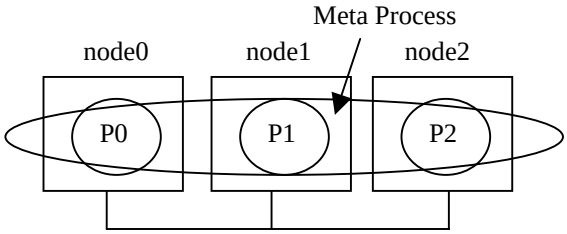

図１　メタプロセスとプロセスの関係

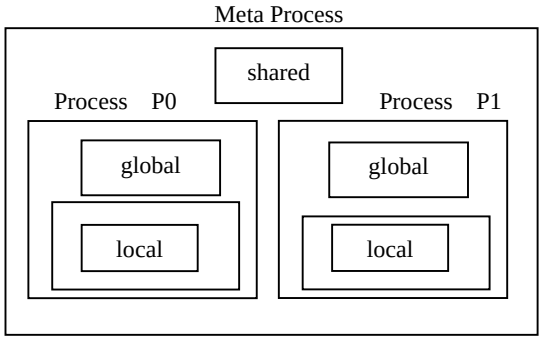

図２　shared 変数のスコープ

４．メタプロセスモデルを実現する MpC

メタプロセスモデルを実現するためのユーザ API として，MpC（メタプロセス C）言語を設計し，メタプロセス実行のために，ソフトウエア分散共有メモリ SMS[1]上での実行方式を構築した．

4.1 shared 共有変数

MpC は，shared というデータ修飾子や，共有データのプロセス関連づけのためのデータ分散割付指定子を新たに定義した拡張 C 言語である．

予約語 shared ではじまる共有変数は，各プロセスの実行プログラムモジュールを構成するソースプログラムファイルのうち少なくとも一つに，関数外部に宣言されることを前提とする．これにより，同一メタプロセス中のプロセス間で同じ共有データをアクセスできる．

shared 変数のスコープは，メタプロセス全体で，メタプロセス内の全プロセスからの参照が可能である．同一モジュール内の別ファイルからの参照には，従来の extern のよ

うに（extern）shared 宣言が必要になる．スコープレベルは外側から，shared，global，local の順の入れ子になっており，同一変数名の場合には内側が有効となる．

shared 変数は，今回の実装では動的に生成されるが，寿命としては，メタプロセス実行開始時に作成され終了時に消滅するので，従来の static 変数と同様に使用できる．

4.2 共有変数のプロセス割付 API

共有変数とプロセスとを関連づける割付 API は図３のように宣言形式で書くことができる．従来の C の変数宣言の先頭に ***shared*** を付加し，後尾に :: から始まる分散割付指定子を付加する．ただし，分散割付指定子全体を省略することも可能であるし，割付開始プロセス番号と割付プロセス数のみを省略したり，割付プロセス数のみを省略することも可能である．図４に宣言例，図５に割付例を示す．

4.3 メタプロセスの記述

メタプロセスを構成するプロセスで用いるプログラムファイルを MpC コンパイラでコンパイルする際に，実行モジュールとは別に，共有変数に関する情報がファイル出力される．この***共有変数情報ファイル***とメタプロセス記述ファイルをもとに，shared 変数の整合性がチェックされ，***メタプロセス実行ファイル***が作成される．ただし，SPMD 型の単純なメタプロセスである場合には，メタプロセス起動時に並列度とプロセス実行ファイルを指定するだけで，メタプロセス記述ファイルを必ずしも使用しなくてもよい．

実行時には，この***メタプロセス実行ファイル***と，実際に使用する物理プロセッサ名などを記述した***プロセッサファイル***を使って，プロセッサに論理プロセスの割付がされて実行される．

５．分散共有メモリシステムへの実装

筆者らはソフトウエア分散共有メモリシステム SMS において，複数プロセスで共有されるデータを，従来の malloc 形式の他に，shared という予約語を用いた変数宣言形式でも使用できる API[2]を既に開発している．これにメタプロセスのスコープ概念を取り入れ，メタプロセス実行ファイルを生成するための MpC コンパイラ mpcc を作成した．さらに，TreadMarks や JIAJIA での実行も可能にする mpcc における実装系指定機能も構築中である．

６．おわりに

分散共有メモリモデルを用いることで，プロセス間で共有するデータとその他のデータを明確に区別でき，従来の拡張型共有メモリモデルにおける，動作やメモリ一貫性の曖昧さと複雑さが排除され，並列プログラマにとってむしろわかりやすい．またユーザに隠された暗黙のメモリ一貫性同期処理などの無駄がないため高速実行可能なプログラムを書くことが可能である．メッセージパッシングにおけるデータの送受信などの面倒な記述も不必要で，プログラムの可読性，記述性がよい．

メタプロセスモデル API は，pthread におけるスレッドとプロセスの関係を，プロセスとメタプロセスに置き換えたモデルに似ており，従来の共有メモリプログラマにとっても違和感が少ない．しかし pthread は大域データだけでなくファイルなどの資源も共有しているなどの点で，通常 OS のクラスタ上にそのまま pthread モデルを実装するには無理があり，プロセスとして扱うほうが自然である．

我々と同様に分散共有メモリをモデルとした言語に UPC[3][4]があるが，これは OpenMP などと同様に SPMD モデルを対象にしており，forall 構文を組み込んでいる，strict/relaxed の２種のメモリ一貫性がある，ポインタ変数に対する考え方などが我々のモデルとは異なっている．

参考文献

[1]緑川，飯塚，"ユーザレベル・ソフトウエア分散共有メモリ SMS の設計と実装"，情報処理学会論文誌ハイパフォーマンスコンピューティングシステム,Vol.42, No.SIG9(HPS 3), pp.170-190 (2001,August)

[2] 緑川，片野，飯塚，"分散共有メモリ SMS における共有変数の分散割り付け API", FIT02 (1)B-42,pp.171-172（2002,9）

[3] Tarek El-Ghazawi, Francois Cantonnet, ”UPC Performance and Potential: A NPB Experimental Study,” Supercomputing2002, IEEE,(2002)

[4] W. Carlson, J. Draper, D. Culler, K. Yelick, E. Brooks, and K. Warren. “Introduction to UPC and Language Specification,” CCS-TR-99-157, IDA Center for Computing Sciences, 1999.

```
shared 型名　変数名[sm]..[si]..[s0]::[dm]..[di]..[d0](st,n)
    変数名[sm]..[si]..[s0]:  変数　各次元サイズ
    [dm]..[di]..[d0] :　各次元の分割数（省略時は分割数１）
    st :　割付開始プロセス番号（省略時は任意プロセス）
    n :　割付プロセス数　（省略時は全使用プロセス数）
```

図３　共有変数の宣言形式

```
shared int x::(st);  指定プロセス st に割付
shared double y[M];  任意プロセスに一括割付
shared float z[M]::[n](st,n);  n等分　st～st+n-1
shared int q[M][L][N]::[n][ ][ ]; n等分　0～n-1
```

図４　共有変数の宣言例

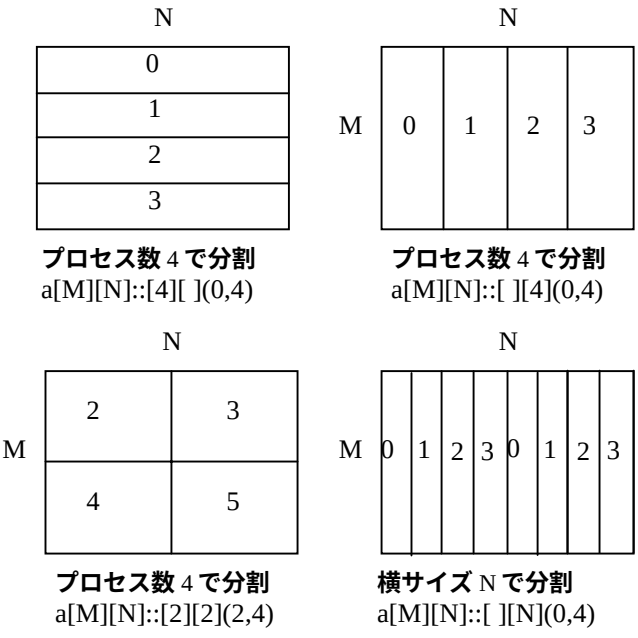

図５　共有変数のプロセス割付例