並列オブジェクト指向言語による並列計算機上での効率的並列計算

児玉靖司 †
Yasushi KODAMA

† 南山大学数理情報学部
yas@nanzan-u.ac.jp

並列計算機上で並列計算をする場合，全体をあたかも一つの計算機とするための抽象化が必要である．我々は，抽象化レベルを記述言語にまで引き上げ，並列計算の単位を並列オブジェクトとし，記述言語として，並列オブジェクト指向言語 SPL を設計，開発した．本論文は，並列計算機として，SMP と同様の構造をもつ PC クラスタ上での PVM を利用した実現に絞り，並列オブジェクトを，同一計算機上では，軽量プロセス (スレッド) に割当て，異なる計算機上では，新たに重量プロセスを生成した後，内部に軽量プロセスとして生成する方法を提案する．結果として，ユーザーが記述言語レベルで，並列計算機の静的要因を考慮し，プログラムとして指定することによって静的負荷分散を実現することができる．

1 はじめに

並列計算機として，複数の計算機をあたかも一つの計算機として計算を行う PC クラスタ上や，マルチプロセッサシステムを用いた SMP などの形態を考えることができる．さらに，並列計算機上で並列計算を行うためには，システムを抽象化し，負荷分散や，位置透過を実現することが必要である．抽象化により，1. ユーザが指定した並列計算の単位を，システムが負荷の軽い実行システム (負荷の軽いプロセッサ，重量プロセスの代わりの軽量プロセスなど) に割当てたり (負荷分散)，2. プログラミングの際に，並列計算をどのプロセッサで実行するかを意識することなく，並列計算を効率よく行うことができる (位置透過)，を実現することができる．我々は，抽象化の度合いを記述言語にまで引き上げる．よって，ユーザが並列計算の単位を並列オブジェクトとして明確に指定することによって，言語処理系が，並列オブジェクトをもとに，負荷分散，位置透過を実現することができる．

記述言語として，並列オブジェクトモデル ABCM/1[6] に基づいた並列オブジェクト言語 SPL(a Simple Parallel Language) を設計開発した．さらに，さまざまな並列計算機上での設計開発を進めている．

(1) PC クラスタ上での PVM を用いた実現
(2) pthread ライブラリを用いた実現
(3) やゑ [5] のスケジューラ上での実現

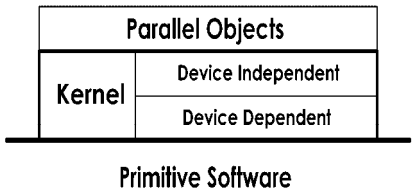

図 1: SPL の構成

(1) は，本論文で中心となる実現であるが，非共有メモリ並列計算機上での実現である．(2) は，共有メモリ並列計算機上での実現である．(3) は，オペレーティングシステムのスケジューラ上での実現で，SPL 自身がシステム記述言語として効率よいシステムを記述することができるかを実験するためのものである．我々の研究は，ユーザがどのプロセッサで実行するかを意識することなくプログラミングしたコードをさまざまなプロセッサで実行できることを，位置透過ととらえている．よって，さまざまな並列計算機上での実現も研究目的の一つであり，実現の際には，PVM，pthread ライブラリなど，既存の効率よい処理系をできるだけ利用する．SPL の構成を図 1 に示す．Primitive Software 上に Kernel として，デバイス依存の部分と，デバイス非依存の部分がある．SPL によるプログラムは最も上位レベルの Parallel Objects(並列オブジェクト群) である．本論文では，並列計算機として PC クラスタ上での PVM を用いた実現に絞り，議論を行う．

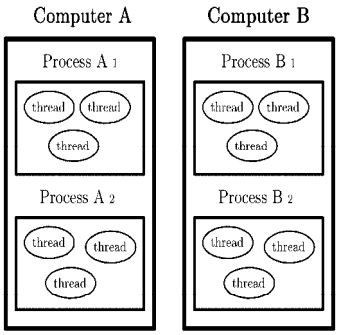

図 2: 提案するシステム構成

以下，2 節では問題提起を述べ，3 節で言語仕様の変更と設計を述べ，4 節でサンプルプログラムを用いた評価をする．5 節で議論と今後の課題を述べる．

2 問題提起

並列計算機の一つの形態である PC クラスタ上で，効率よく並列計算をするための処理系として，PVM や MPI がある．これらのシステムは，UNIX 系オペレーティングシステム上で重量プロセスを複数の計算機に分散させ負荷分散を実現する．しかし，同一計算機内の処理でも重量プロセスとして分割するため，軽量プロセス (スレッド) として分割した場合より効率が悪い．よって，効率的並列計算をするためには以下の見通しを立てることができる．

(1) 異なる計算機には，重量プロセスを単位として並列計算を分割する．
(2) 同一計算機には，軽量プロセスを単位として並列計算を分割する．

PVM のみを用いた並列計算では，(1) のみが達成されるが，(2) は達成されない．この見通しを達成するためには，機能を抽象化し，図 2 の構成とする必要がある．並列計算の単位を軽量プロセス (thread) とし割当て，重量プロセス ($ProcessA_1$，$ProcessA_2$，$ProcessB_1$，$ProcessB_2$) の内部に複数生成する．重量プロセスは，同一計算機内に複数生成することができるが，できるだけ複数の計算機に分散させる．同様の研究として，SMP クラスタ向け並列実行環境として，並列プログラムのオブジェクト互換を保ちつつ負荷分散を実現するライブラリを提供するもの [1] があるが，既存のソフトウエア (PVM など) を利用せず，一からシステムを設計し，開発した例であり，さまざまな並列計算システムに適用するすることは難しい．一方，我々は既に，並列オブジェクト言語 SPL をさまざまな並列計算機上で設計開発しており，既存のソフトウエア (PVM) を利用し，その機能をさらに抽象化する方向で，上の見通しを実現することは容易である．

PC クラスタの PC の台数や，各 PC の性能差などの静的要因を自動的 (動的) に調べる動的負荷分散も考えることができるが，処理が複雑となり容易ではない．我々は，ユーザが，記述言語 SPL によるプログラミングの際に，上の静的要因を指定し静的負荷分散を実現する方式を提案する．

3 言語仕様の変更と設計

並列言語 SPL は，ABCL/c+[4] の後継に位置する言語であり，並列オブジェクト指向モデル ABCM/1 に基づいた言語である．並列計算の単位は，並列オブジェクトである．以下のように並列オブジェクトを定義する．

```
start {
    var 変数名 = 初期値; ...
    var 変数名 =
      [object オブジェクト名
        delegate-to 委譲先
        state var 識別子 = 初期値; ...
        script
          (=> メッセージパターン {
            var 識別子 = 初期値; ...
            文並び; })
            . . .
          ];
    文並び;
}
```

start { から } までの部分にプログラムを記述する．この部分は，最初に起動される main 並列オブジェクトの一つのメソッドを示す．メソッドは，var から始まる宣言部と，文並びからなる．宣言部は var 変数名 = 初期値 と記述し,「変数名」を名前とする変数を宣言する．この宣言は，参照される前であればどこに宣言してもよい．各変数の型は，初期値を用いて型推論される．初期値には，定数，オブジェクト式など式を記述する．新たに，並列オブジェクトを一つ生成する場合は，宣言部で行う．メソッド本体である文並びは，メッセージ受渡し文，select 文を除き，C 言語と同様の記述をすることができる．変数への代入は，演算子 := を用いる．

オブジェクト定義は，3 つの部分 delegate-to 部，state 部，script 部からなり，各々委譲先オブジェクト指定，状態定義，メソッド定義をする．状態定

義，メソッド定義中での変数宣言は，初期値から型を推論する[1]．

メッセージ受渡しは，ABCM/1 で提案された，過去型，現在型，未来型のメッセージ受渡し型を定義することができる．

3.1 並列オブジェクト生成と複製

並列オブジェクト生成は，並列オブジェクト定義で述べた [object Name ...] により，生成，起動される．Name は，オブジェクト名を示す．

並列オブジェクト複製には，演算子 clone を用いる．以下の 2 通りの方法がある．

(1) 上記の並列オブジェクト生成で起動されたオブジェクトを複製し，新たに起動する．

clone ObjectID

(2) プログラムを文字列として受取り，新たに生成，起動する．

clone (ObjectName, Program)

ObjectID は，oid 型の変数，ObjectName, Program は，各々オブジェクト名，プログラムソースであり，文字列として受渡す．(1) は，[object Name ...] により生成されたオブジェクトを複製し，起動する．(2) は，文字列としてのオブジェクトの状態，メソッド定義を含んだ並列オブジェクトを動的生成 (コンパイル) し起動する．動的コンパイルにより，動的にさまざまなオブジェクトを生成，起動することができ，柔軟性の高いプログラミングが可能となる．

本論文では，並列計算機の一つである PC クラスタ上での実現に絞り，実行効率を高めるのが目的であるため，演算子 clone の仕様 ((1) の方法) に変更を加えた．

clone1(flag, ObjectID)

flag には，整数型の変数または値を指定し，並列オブジェクトの複製を切替える．

負の数の場合 同一重量プロセス内の軽量プロセスとして複製し，起動する．

0 以上の数の場合 重量プロセスを新たに生成し，その内部に軽量プロセスとして複製し，起動する．

[1]メッセージパターン部では，型を宣言しなければならない．

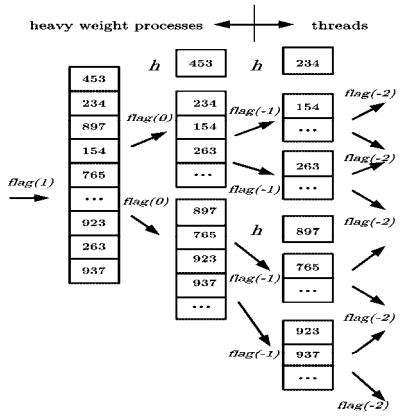

図 3: 整数配列のソート

動的コンパイルを用いた並列オブジェクトの複製 ((2) の方法) は，別稿 [2] により，実行時間はコンパイル時間のみ異なり，起動後の実行にはほとんど影響しないことを確認したので，本論文では，演算子 clone の (1) の方法のみ変更し，評価を行う．

3.2 その他

その他の言語仕様は，一切変更しない．よって，これまで作成したプログラムの少し変更で，提案する方式を実行させることができる．しかし，図 1 の Kernel 部分の処理系は，以下の 2 点の変更が必要である．

(1) メッセージ受渡しの際に，同一重量プロセス内への通信か否かをチェックし，同一重量プロセス内への通信の場合は，共有メモリモデルを前提とした通信を行う．
(2) 演算子 clone1 の実現として，flag の値にしたがって，上記複製の切替えを行う．

以上の言語仕様の変更，システム設計に基づいて Kernel の部分の修正を行った．

4 評価

100 個の整数値のソートを，並列再帰アルゴリズムの手法を用いて，SPL で記述し，8 台の PC からなる PC クラスタ上での実行時間を計測した[2]．各 PC は，CPU が Pentium III 750MHz，OS は Linux を用いた．計算の様子を図 3 に示す．

[2]プログラムは，付録に示した．なお，詳しい言語仕様については，紙面の都合上省略する．

main 並列オブジェクトに，N 個の要素を持つ配列を用意し，以下の処理をする並列オブジェクト (Unit 並列オブジェクトという) を 1 つ生成する．結果として，main 並列オブジェクトには，最初に生成された Unit 並列オブジェクトから，ソートした要素を持つ配列がメッセージとして受渡される．

Unit 並列オブジェクトでは，最初に M 個の要素を持つ配列と，整数値を保持する flag をメッセージとして受渡される．最初の要素 h を取り出し，h 未満の値を集めた配列，h 以上の要素を集めた配列に分割する．次に，2 つの配列を別々に並列にソートするため，Unit 並列オブジェクトの 2 つの複製を clone1 演算子により複製し，上記 2 つの配列を別々にメッセージとして受渡す．その際に，ディクリメントした flag の値も受渡す．結果として，動的に複製された Unit 並列オブジェクトが，複製元 Unit 並列オブジェクトに，ソートした要素を持つ配列をメッセージとして返答する．この処理を繰り返すことにより，最初に生成された Unit 並列オブジェクトが，main 並列オブジェクトに，ソートした要素を持つ配列を返答する．flag の値は，各 Unit 並列オブジェクトの複製の際にディクリメントするので，負の数になった時点で，並列オブジェクトに，同一重量プロセス内部の軽量プロセスに割当てる複製に切替える．よって，flag の初期値によって，並列オブジェクトに対して，図 3 のどのレベルまで，新たな重量プロセスを生成し割当てるか，を調節することができる．

1 台の PC では，重量プロセスの生成は数百個程度が限界であり，このアルゴリズムでは，最も細かく分割される場合，整数値 1 個につき，1 つの重量プロセスが割当てられることが考えられるため，N の値を 100 とし，flag の初期値を変えソートが完了するまでの実行時間を計測した．flag の初期値 2 の場合，重量プロセスが最大 7 生成され，明確に，台数効果を表すことができる．flag の初期値 0 の場合は，2 つめの Unit 並列オブジェクトの生成から[3]，同一重量プロセス内部の軽量プロセスに割当てる．最も台数効果が表されない場合である．よって，flag の初期値 2 の場合 (上段)，flag の初期値 0 の場合 (下段) の実行結果を図 4 に示す．横軸に PC の台数，縦軸に秒単位の実行時間とした．

flag の初期値 2 の場合，1 〜 8 台にかけて台数効果が表されている．flag の初期値 0 の場合，最大 2 つの重量プロセスが生成されるため，1 〜 2 台にかけては，台数効果が表されるが，以降台数を増やしても並列オブジェクトが異なる計算機に分散されないので，実行時間はほぼ変わらない．以上より，演算子 clone1 で，並列オブジェクト複製の際に指定する flag の値を調節することにより，PC の台数に応じた効率的並列計算ができることが示された．

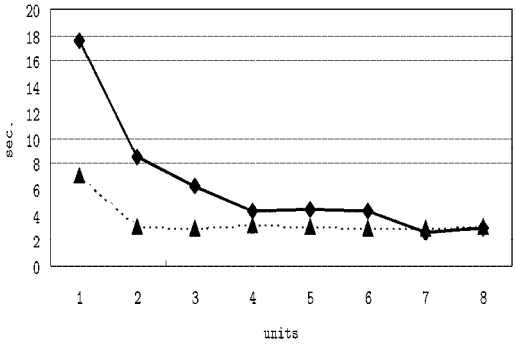

図 4: PC クラスタ上での並列計算の台数効果

5 議論と今後の課題

並列計算機の一つである PC クラスタ上で効率的並列計算をするために，

(1) 異なる計算機には，重量プロセスを単位として並列計算を分割する．
(2) 同一計算機には，軽量プロセスを単位として並列計算を分割する．

方式を提案する．我々は，既にさまざまな並列計算機上で並列言語 SPL の実行環境を設計，開発しており，並列オブジェクト複製を示す演算子 clone の仕様を変更することで，上記方式の実現は容易である．変更した演算子 clone1 により，並列オブジェクトを，新たに生成する重量プロセス内部に軽量プロセスとして割当てるか，同一重量プロセス内部の軽量プロセスとして割当てるかを切替えることができる．PC クラスタ上の重量プロセスは，PVM により効率よく各 PC に分散される．この手法により，並列単位である並列オブジェクトの起動方法を制御することができ，SPL の汎用性により，他の並列計算機にも容易に応用することができる．

[3]現在の SPL の仕様では，並列オブジェクト生成 (演算子 clone でない場合) は，必ず重量プロセスを新たに生成する．

PC クラスタを構成する PC の台数，各 PC の性能差など，自動的 (動的) に調べる動的負荷分散は，処理が複雑になり容易ではない．本論文では，ユーザが演算子 clone1 により指定する flag の値により，上記処理を調節する静的負荷分散を採用した．4 評価より，flag の初期値を調節することにより，PC の台数に応じた効率的並列計算ができることを示した．

我々は SPL をさまざまな並列計算システム上で設計開発している．ユーザがどの計算機で実行されるかを意識することなく，プログラミングされたコードをさまざまな計算機で実行できるため，位置透過は実現されている．本論文では，位置透過に加え，(静的) 負荷分散に対する考察を行った．

今後の課題として，SPL 処理系 (図 1 の Kernel) の効率化があげられる．PVM のみを用いて同様のプログラムを実行したところ，PC 1 台で 1 秒程度，以下台数に応じて台数効果を確認した．PVM の標準の通信ライブラリを用いた場合，図 4 では，実行時間が，その数倍であったが，コネクションベースソケットを別途用いることにより，1 秒以下となり，グラフの形状は，同様となった．しかし，さらなる改良が必要である．さらに，並列計算システムを記述言語にまで抽象化したことにより，複雑な処理を処理系に含めることができると考えている．動的負荷分散に関する考察は，今後の課題とする．

謝辞

本研究は，2002 年度南山大学パッヘ研究奨励金 I-A-2(特定研究助成) より研究助成を受けて行っています．

参考文献

[1] 石畑宏明，小林健一，石井光雄: SMP クラスタ向け並列処理実行環境の構築，信学論 (D-I), Vol.J84-D-I, No.6, 2001, pp. 568-575.

[2] 児玉靖司: 動的コンパイル可能な並列言語の設計と実現, プログラミング言語研究会発表資料, 情報処理学会, 2001.

[3] 芝公仁，大久保英嗣: 分散オペレーティングシステム SolelC の設計と実装, 信学論 (D-I)，Vol.J84-D-I, No.6, 2001, pp. 617-626.

[4] Doi, N.,Kodama, Y. and Hirose, K.: An Implementation of an Operating System Kernel using Concurrent Object-Oriented Language ABCL/c+, ECOOP'88, ACM, 1988, pp.250-266.

[5] Kodama, Y. and Asumi, T.: Yawe:Yet Another Window Environment, コンピュータシステムシンポジウム, 情報処理学会, 1998, pp. 1-8.

[6] Yonezawa, A., Shibayama, E., Takada, T. and Honda, Y.: Modeling and Programming in an Object-Oriented Concurrent Language ABCL/1, Object-Oriented Concurrent Programming (ed. Yonezawa, A. and Tokoro, M.), The MIT Press, 1987.

付録

```
start {
 var Buffer=int[1000]; var Result=int[1000];
 var count=100;
 var ut =
  [object Unit
   state var Buffer=int[1000] var Lower=int[1000]
    var cnt=0 var Upper=int[1000] var Result=int[1000]
   script
    (=> :ARRAY int c int B[1000] {
     var i=0; cnt:=c;
     for(i:=0;i<cnt;i++) Buffer[i]:=B[i];
     })
    (=> :CAL int f int flag @ S {
     switch(cnt) {
      case 1: {
       [S <= :ARRAY f 1 Buffer[1]]; break; }
      case 2: {
       var t=0;
       if(Buffer[0]>Buffer[1]) {
        t:=Buffer[0];Buffer[0]:=Buffer[1];Buffer[1]:=t;
       }
       [S <= :ARRAY f 2 Buffer[2]]; break; }
      case 3: {
       var t=0;
       if(Buffer[0]>Buffer[1]) {
        t:=Buffer[0];Buffer[0]:=Buffer[1];Buffer[1]:=t;
       }
       if(Buffer[1]>Buffer[2]) {
        t:=Buffer[1];Buffer[1]:=Buffer[2];Buffer[2]:=t;
       }
       if(Buffer[2]>Buffer[0]) {
        t:=Buffer[2];Buffer[2]:=Buffer[0];Buffer[0]:=t;
       }
       [S <= :ARRAY f 3 Buffer[3]]; break; }
      default: {
       var h=Buffer[0]; var j=0; var k=0; var i=0;
       for(i:=1;i<cnt;i++) {
        if(h>=Buffer[i]) Lower[j++]:=Buffer[i];
        else Upper[k++]:=Buffer[i];
       }
       flag--;
       if(j>0) {
        var l=clone1(flag,Me);
        [l <= :ARRAY j Lower[j]];[l <= :CAL 0 flag @ Me];
       }
       if(k>0) {
        var u=clone1(flag,Me);
        [u <= :ARRAY k Upper[k]];[u <= :CAL 1 flag @ Me];
       }
       Result[j] := h; var scnt=1;
       [select-loop
        (=> :ARRAY int f int c int R[1000] {
         scnt+=c; var k=0;
         if(f==0) {
          for(k:=0;k<c;k++) Result[k]:=R[k];
         } else {
          for(k:=0;k<c;k++) Result[j+1+k]:=R[k];
         }
         [SenderOfSelect <= :Quit];
         if(scnt==cnt) exit;
         })];
      [S <= :ARRAY f cnt Result[cnt]]; }
     })];
 Primitive::srand(2234782); var i=0;
 for(i:=0;i<count;i++) Buffer[i]:=Primitive::rand;
 for(i:=0;i<count;i++)
  Primitive::printf("[%d]",Buffer[i]);
 Primitive::printf("\n");
 [ut <= :ARRAY count Buffer[count]];
 [ut <= :CAL 0 2 @ Me];
 [select
  (=> :ARRAY int f int c int R[1000] {
    for(i:=0;i<c;i++) Result[i]:=R[i];
    [SenderOfSelect <= :Quit];
   })];
 Primitive::printf("* count[%d]\n",count);
 for(i:=0;i<count;i++)
  Primitive::printf("%4d:%d\n",i,Result[i]);
}
```