WARASA：軽量プロセス上での並列オブジェクト指向プログラミング言語

江 允† 牧之内 顕文†

並列オブジェクト指向プログラミング言語は，並列性と同期機能がオブジェクトに埋め込まれているため，理解しやすい並列処理記述法といわれている．われわれはマルチプロセッサ計算機のOSが提供する並列プリミティブとオブジェクト指向パラダイムとを結合するような並列オブジェクト指向プログラミング言語 WARASA を提案している．WARASA は C++ に並列機能を追加した拡張言語として設計された言語である．この言語は，従来の並列オブジェクト指向プログラミング言語と違って，並列機能の実現にはカーネル機能を利用し，プログラムの並列制御をユーザが書かなくて済むように設計されている．WARASA で書いた並列プログラムは効率的かつ実用的である．本論文は，以上の特徴を持った WARASA の設計思想，実現法および予備的な評価について述べる．

WARASA: A Language for Parallel Object-Oriented Programming on Lightweight Processes

YUN JIANG† and AKIFUMI MAKINOUCHI†

Since parallelism and synchronization mechanisms can be covered by an object, parallel object-oriented programming languages are considered as a useful descriptive method that makes parallel processing easily understood. We combine the parallel primitives provided by a multiprocessor operating system with the object-oriented paradigm and propose a new parallel object-oriented programming language called WARASA. WARASA is an extension of C++ and allows users to develop parallel object-oriented programs. The features of WARASA are that its parallel functions are realized by using the parallel primitives of a multiprocessor operating system, and that it hides implementation details which users would have to know in order to write parallel programs. These features enable users to implement parallel programming easily and efficiently. In this paper, we report concepts of the WARASA design, implementation methods, and its preliminary estimation.

1. はじめに

近年，VLSI 技術の発展に伴い，高性能なマルチプロセッサシステムが安価に実現できるようになってきた．それに応じて，高速処理を求める応用分野では，応用プログラムの性能をマルチプロセッサ計算機の並列機能を用いて向上させようという要求がますます高まってきている．しかし，並列処理による高速化はそれほど容易ではない．その理由の一つとして，実際的な並列プログラミング言語がないことがあげられる[12]．現実には，並列計算機の利用者は応用問題自身の並列性を考えるよりも，解くべき問題に関係のない並列プログラミング技術そのものに対する工夫に時間をとられている[13]．このような状況を考えると，マルチプロセッサ計算機に有効なソフトウェア開発環境を提供する研究が重要な課題である[14]．並列マシンを活用するため，対象となる問題のもつ並列性を十分に抽出し記述できる言語が必要となる．

一方，並列オブジェクト指向プログラミング言語は並列性と同期機能がオブジェクトに埋め込まれているため理解しやすいといわれてきた[21]．また，オブジェクトはデータと制御を同時に司るため，並列・分散オブジェクトのメッセージ通信による動作記述は並列問題を記述する自然な表現方法と考えられている[23]．実際，並列オブジェクト指向プログラミング言語の研究がなされ，ABCL/1[22]，Concurrent Smalltalk[20]，Presto[2] などが今までに提案された．

しかし，並列オブジェクト指向プログラミング言語の有効性と使いやすさを追求するためには，その並列

† 九州大学工学部情報工学科
Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University

機能，同期機能及び通信方式とをオブジェクト指向パラダイムで完全に統合するためのさらなる工夫が必要であることが指摘されている[19]．例えば，多くの並列オブジェクト指向言語では，オブジェクトはメッセージを受け取ることによって喚起される．そのために，プログラマは並列オブジェクトでそれ自身に関する操作記述を行う必要があると同時に，同期および共有資源を管理する必要もある．そのために，利用者，特に初心者がそれらを利用するのは，容易ではない[19]．

そこで，われわれはC++を拡張した，使いやすい並列オブジェクト指向プログラミング言語WARASAを提案している[8]．WARASAはC++に並列機能を追加した並列オブジェクト指向プログラミング言語である．WARASAの設計方針は，マルチプロセッサ計算機OSのカーネル（以下カーネルといえばOSのカーネルを指す）が提供する並列プリミティブとオブジェクト指向パラダイムとを結合することである．すなわち，カーネルの並列プリミティブを有効に利用することよりWARASAによる並列プログラムの性能を保障するとともに，それをオブジェクトにカプセル化する記述により，抽象度の高いプログラミングが可能な言語をユーザに提供する．

WARASA並列プログラムの実行効率をよくするために，並列実行の単位はスレッド（thread）とする．スレッドは“軽量プロセス”（lightweight processes）とも呼ばれる．従来のプロセスと比較すると，プロセスの生成・消滅，プロセス間の同期・通信，および，コンテキスト切り替えのオーバヘッドが小さいので，従来のプロセスを用いる方法より効率よく実現することが可能である[10),15)]．

本論文では，WARASAの設計思想を述べ，WARASAの並列オブジェクトをマルチプロセッサ計算機における軽量プロセス上で実現する方法を論じ，例題の実験結果を示す．また，他言語との比較を行い，WARASA並列オブジェクト指向プログラミングの記述能力を評価する．

2. WARASAの設計方針

WARASAを，マルチプロセッサ計算機の並列機能を有効に利用し，ユーザに使いやすい高抽象度の並列機能を有する言語とするため，

（1） 並列機能のC++への追加
（2） 並列機能記述の隠蔽
（3） カーネルの並列プリミティブの利用

を3つの設計方針とした．本章では，それらについて述べる．

2.1 C++への並列機能の追加

WARASAは基本的にはC++言語である．C++言語はCの派生言語であり，データ抽象化と継承機能など，典型的なオブジェクト指向の特性を持つ．さらに，低い実行時コストでこれらの特性を利用できることがC++の長所として評価されている[3)]．これがC++をWARASAのベース言語として選択する理由である．

C++に並列機能を導入するため，新しいオブジェクトの型を提案している．提案するオブジェクトは3種類ある．それらは自律オブジェクト（Autonomous Object），排除オブジェクト（Exclusive Object）および条件同期オブジェクト（Synchronous Object）である．3種類の新たなオブジェクトを用いることにより，ユーザは並列プログラムを書くことができる．新しい種類のオブジェクトについては4章で詳しく述べる．

また，WARASAでは，C++言語本来のオブジェクト型を利用することができる．C++本来のオブジェクトは，WARASAでは通常のオブジェクト（Ordinary Object）と呼ばれ，並列機能を持たず，C++のオブジェクトと全く同じように振舞う[16)]．

2.2 並列機能記述の隠蔽

並列プログラムをどういうふうに書くか，並列プログラムは，どのように作れば効率がよいかについてはさまざまなアプローチがあり得る．例えば，結果並列（Result Parallelism），手順書並列（Agenda Parallelism）と専門家並列（Specialist Parallelism）などの3つの並列プログラミング方式が考えられる[4)]．しかし，並列プログラミング方式が異なっても，並列プログラムにおけるプロセスの配置とデータの分配や，並列実行プログラム間の同期制御および共有データアクセスのための排他制御などの機能は並列プログラミングの中で最も基本的で不可欠な機能である．これらの機能は通常の逐次処理プログラミングではない機能であるため，普通のプログラマにとっては必ずしもなじみやすいものではない．したがって，これらの機能をどのようにうまく利用して並列プログラミングするかがプログラマの悩みになる．

WARASAの並列機能の設計の基本は，Concurrent Pascal[6)]にC++流のオブジェクト指向パラダイムを付加し，基本的な並列機能を分担するオブジェクト

を提供することである．特に，これらの基本機能をオブジェクトの宣言の中に隠すことにしたので，プログラマはこれらのオブジェクトを用いて並列プログラムを書けば，例えば，fork/join 操作や，共有データアクセスに必要な lock/unlock 操作といった排他制御操作を書かなくても済む．

2.3 カーネルの並列プリミティブの利用

多くの並列オブジェクト指向言語では，並列オブジェクトを実行するための制御機能を，オブジェクトで記述しなければならなかった．例えば，Presto は排他制御のために，ロック・オブジェクトを使うことにより排他制御を行う[2]．そこで，WARASA はマルチプロセッサ計算機のカーネルが提供するこれらのプリミティブを効果的に利用する方針を立てた．しかも，それら機能をプログラマが陽に記述することはわずらわしいので，それらを WARASA のオブジェクト型の宣言と結合し，プログラムからは見えなくしている．例えば，WARASAの条件同期オブジェクトでは Mach カーネルの同期プリミティブをカプセル化しており，プログラマはカーネルの同期機能について知らなくてもよい．

3. 軽量プロセスの並列環境

WARASA はマルチプロセッサ計算機 OS のカーネル並列プリミティブ，同期プリミティブをオブジェクト指向パラダイムに統合する並列オブジェクト指向プログラミング言語を提供する方針のもと，軽量プロセスをサポートする Mach OS を対象として設計されている．

Mach はカーネギーメロン大学 (CMU) で開発されたマルチプロセッサ用分散 OS である[18]．Mach では，UNIX 4.3 BSD との互換性を保ちながら，スレッドやメッセージなどの Mach 独自の機能も同時に使用することができる．さらに，タスク，スレッド，ポート，メッセージ，メモリ・オブジェクトなどのカーネル内の基本機能をユーザに開放しているので，OS の低レベル機能をアプリケーションプログラムで利用できることが大きな特徴である．

以下，Cスレッド・パッケージが提供する相互排除及び同期プリミティブについて，WARASA の理解に必要な諸事項のみを簡単に述べる．

● Cスレッド・パッケージ

Mach では，タスクレベルとそれをより細分化したスレッドレベルでの並列実行をサポートしている．タスクは実行環境であり，リソース割り当ての基本単位である．スレッドはいわゆる軽量プロセスと呼ばれる並列実行単位であり，1つのタスクのなかで複数のスレッドを同時に走らせることができる．Mach OS はこのスレッドの生成・消滅，相互排除や，スレッド間の同期などのプリミティブをユーザに公開するため，Cスレッドユーティリティ・ライブラリを提供している[5]．

● 相互排除プリミティブ

Cスレッド・パッケージは共有データアクセスの排他制御のために，相互排除プリミティブを提供している．このプリミティブを利用するときに，mutex 変数の割り当てとロック／アンロック操作を行う必要がある．この相互排除プリミティブでは，複数のスレッドが mutex_lock 操作を利用し，同じ mutex を同時にロックしようとした場合，1つのスレッドだけがロックできることを保証する．残りのスレッドは mutex のロックが解除されるまでブロックされている．また，mutex_unlock 操作は mutex のロックを解除して，他のスレッドがロックできるようにする．

● 同期プリミティブ

並列処理の場合，あるスレッドが別のスレッドの処理終了を待つときなどに，Cスレッド・パッケージの同期プリミティブが使われる．WARASA に利用される主な同期プリミティブは condition_t, 条件変数に関する condition_wait, condition_signal および condition_broadcast 操作である．condition_wait 関数はスレッドが期待する条件が成立するのを待つときに使う．condition_signal 関数は指定した条件変数が表す条件が真になったときに，それを待っているスレッドの1つを喚起する．また，1つだけ起こすのではなく，すべてのスレッドを喚起する condition_broadcast 関数も利用できる．

しかしながら，Cスレッド・パッケージを使った複数スレッドでの並列実行プログラムは書くのに手間がかかる上に容易ではない[11]．

4. WARASA の並列機能

基本的な並列機能を分担するオブジェクトを提供して，C++ に並列機能を導入するため，われわれは**表1**に示すように並列プログラミングに最も基本的であると思われる機能をサポートする新たなオブジェクトを提案した[9]．これらは，並列実行の主体としての自律オブジェクト，データを共有するための排除オブジェ

クトおよび自律オブジェクト間での同期をとるための条件同期オブジェクトである．自律オブジェクトはアクティブ（active）オブジェクトで，排除オブジェクトと条件同期オブジェクトはパッシブ（passive）オブジェクトである．以下，これらオブジェクトについて機能を説明する．

4.1 並列実行

自律オブジェクトは WARASA における並列実行単位である．

●自律オブジェクトの機能

（1） 自律オブジェクトは軽量プロセス環境での並列実行単位であり，複数の自律オブジェクトは**図1**に示すようにマルチスレッド上で並列的に実行される．

（2） 自律オブジェクトの生存期間はそのために生成されたスレッドの生存期間と同じである．

（3） 自律オブジェクトには2つの状態（実行状態と待ち状態）があり，その状態は条件同期オブジェクトの操作によって変更される．複数の自律オブジェクトにより，条件同期オブジェクトを通じて，同期をとりながら相互に協力して実行できる．

（4） 自律オブジェクトは幾つかの通常のC++のオブジェクトから構成されうる．

●自律クラスの宣言

[例1]

```
autonomous class producer{
  private:
    int number;
    int data_w;
  public:
    producer (struct_t* struct_pointer);
    ~producer();
}
```

自律クラスは，キーワード“autonomous”をクラスの先頭に付ける．自律オブジェクト自身に使われるデータはそのクラスの私的（private）部で宣言される．自律クラスの公開（public）部は構築子（constructor）メソッドと消滅子（destructor）メソッドからなる．

自律クラスの構築子と通常のクラス構築子の定義には違いがある．通常のクラスはオブジェクトの初期設定を構築子で宣言する．自律オブジェクトはメッセージを受けず，生成時点から自律的に実行するため，オブジェクトの初期設定を行う操作と他の操作を併せて1つの構築子として定義する．

表1 並列言語の基本機能と WARASA の3種類オブジェクト
Table 1 Parallel functions of CPL and three kinds of object in WARASA.

*Primitives of Usual CPL**	*Parallel Primitives of Mach OS*	*WARASA*
create and delete processes/threads	cthread_fork( ) cthread_join( ) cthread_detach( )	autonomous objects
exclusive control to shared variables	mutex_lock( ) mutex_unlock( )	exclusive objects
synchronization between processes/threads	condition_wait( ) condition_signal( ) condition_broadcast( )	synchronous objects

* CPL: Concurrent Programming Language.

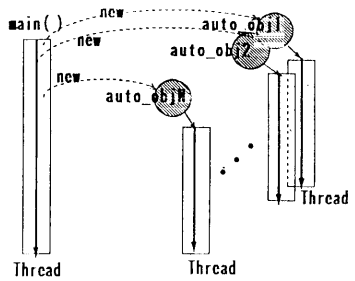

図1 自律オブジェクトの生成
Fig. 1 Creating autonomous objects.

●自律オブジェクトの生成と消去

自律オブジェクトは通常のオブジェクトと同様に new 演算子によって生成され，delete 演算子によって消去される．

4.2 排他制御

複数の自律オブジェクトが他のオブジェクトを共有することがある．そのために，WARASA は排除オブジェクトと呼ぶ排他的に動作するオブジェクトを提供している．

●排除オブジェクトの機能

排除オブジェクトは，複数の自律オブジェクトから複数のメッセージが同時に到着した場合，一時にメッセージを1つしか受け取らない．すなわち，受理しているメッセージの操作を終わらないと，他の自律オブジェクトのメッセージを受けとらない．

●排除クラス宣言

例2に示すように“excl”キーワードを付けて宣言されるクラスは排除クラスである．

[例2]

```
excl class 〈class_name〉{
  private:
    〈データの宣言〉
  public:
    〈メソッドの定義〉
};
```

排除オブジェクトのすべてのメソッドはカーネルの排除プリミティブを暗黙的に利用している．

4.3 同期機構

同期機構は自律的に動作しているオブジェクト群が相互に協力して並列実行するために欠かせない機構である．例えば，一方のオブジェクトの仕事が終了したことを他のオブジェクトに知らせて，そのオブジェクトの実行状態を制御することや，複数のオブジェクトを同じ時点から並列的に実行再開させることなどが同期機能を使って可能となる．このような機能を実現するために導入したのが条件同期オブジェクトである．

●条件同期オブジェクトの機能

条件同期オブジェクトの同期機能によって，自律オブジェクトの同期状態を変えることができる．

例えば，**図2**に示すように3つの自律オブジェクトが，ある条件同期オブジェクトを用いて同期をとる必要があるとき，自律オブジェクト1と自律オブジェクト2はそれぞれ条件同期オブジェクトにメッセージを出す．同期条件が満足されないとき，メッセージを発信した自律オブジェクト1と自律オブジェクト2は一時待たされる．その後，自律オブジェクト3が条件同期オブジェクトにメッセージを出す．これが自律オブジェクト1と自律オブジェクト2が待っている同期条件を満足させると，条件同期オブジェクトは自律オブジェクト1と自律オブジェクト2の待ち状態を解除して，3つの自律オブジェクトが共に実行状態になる．

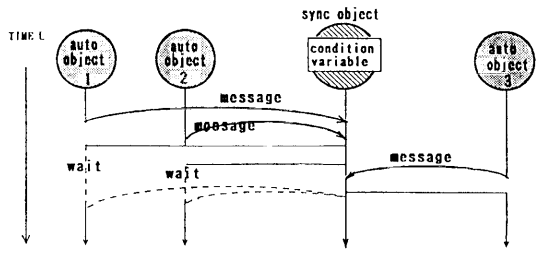

図2 条件同期オブジェクトの機能
Fig. 2 Functions of a synchronous object.

●条件同期クラス

[例3]

```
sync class buff_full_sync(){
  private:
    condition_t buff_full;
  public:
    buffer_full_sync();
    ~buffer_full_sync();
    void wait_buff_available();
    void make_buff_available();
};
```

条件同期クラスの宣言は例3のようになされる．注意すべきは以下のことからである．

（1） 条件同期クラスは“sync”キーワードをclassの前に付ける．

（2） 条件同期クラスの私的部の中に condition_t 型条件変数を含める．例3では，buff_full_sync という条件同期クラスは buff_full という条件変数を持っている．この条件変数は自律オブジェクト同士が同期をとる条件を表す変数という意味である．

（3） 条件同期クラスの公開部でのメソッドの主な役割はこの条件変数に関する同期操作である．ここで，3章で述べた Mach OS の同期化プリミティブを利用する．

5. WARASA プログラムの実現例

以上のように設計された WARASA の並列機能の動作を確認するため，われわれは生産者と消費者問題を WARASA で記述し，並列計算機 LUNA 88K 上で実行した．

5.1 WARASA での生産者・消費者問題の記述

あるプロセスが，他のプロセスの入力となる出力系列を生産するとき，生産者／消費者関係（producer/consumer relationship）があるという[7]．生産者から消費者にデータを送るため，生産者と消費者の2つのプロセスは，共有バッファを介して通信している．共有バッファはデータを蓄える待ち行列として使われる．この待ち行列は，FIFO で管理されるので，データは常に送られてきた順序で受け取られる．

そこでは，バッファが生産者と消費者に共有されるので，バッファの排他制御が必要であるし，データがなくなるときおよびデータがそれ以上バッファに入らないときの生産者と消費者間での同期機能も不可欠である．

WARASA による生産者・消費者問題の記述では，**表2**に示すように5つのクラスを設定した．それらは，生産者自律クラス，消費者自律クラス，バッファ排除クラス，バッファ・フル条件同期クラスおよびバッファ・エンプティ条件同期クラスである．そして，これらのクラスのインスタンスオブジェクトが new 演算子によって生成され，delete 演算子によって消去される．

（1） 生産者と消費者自律オブジェクト

生産者自律クラスの宣言を**図3**に示す．生産者オブジェクトの操作はその構築子の中に定義されている．生産者自律オブジェクトは生成されてから「実行停止」条件（CONTINUOUS=0）が成立するまで，ずっと生産活動を行っている．オブジェクトは「実行停止」条件が成立したら，後仕末をして，return によって終了する．

生産者オブジェクトは，まずデータを生成する．そして，生産されたデータをメッセージ通信によってバッファオブジェクトに送ると同時に，バッファの状態情報を得る．バッファがいっぱいになる（B_full==1）と，生産者オブジェクトはバッファ・フル条件同期オブジェクトにメッセージ '(obj_pointer->full_sync)->wait()' を出す．そして，バッファ・フル条件同期オブジェクトの操作によって生産者オブジェクトは一時待ち状態にされる．ある消費者オブジェクトがデータをバッファから取った後，バッファ・フル条件同期オブジェクトを通じてバッファが利用可能になるのを待っている生産者の待ち状態を解除する．待ち状態から解除されるべき生産者オブジェクトは明示的にメッセージを受けなくても，自動的に実行状態になる．待ち状態から解除された生産者オブジェクトはデータをバッファオブジェクトに再送する．データがバッファに入れられるとき，待ち状態の消費者オブジェクトがあればその待ち状態は解除される．

```
autonomous class producer{
    private:
        int       B_full;
    public:
        producer(obj_t* obj_pointer)
       ~producer()
};
producer(obj_t* obj_pointer){
  while<CONTINUOUS>{
    for(int i=0; i<=4095; i++){
       obj_pointer->send_data[i]=<PRODUCE DATA>;
       B_full=(obj_pointer->buff)->
              deposit(obj_pointer->send_data[i]);
       while(B_full==1){
          (obj_pointer->full_syn)->wait();
          B_full=(obj_pointer->buff)->
                 deposit(obj_pointer->send_data[i]);
       }
       (obj_pointer->empty_syn) -> signal();
    }
  }
  return;
}
```

図3 生産者自律オブジェクト
Fig. 3 A producer autonomous object.

表2 生産者と消費者問題プログラムの WARASA クラス設定
Table 2 WARASA classes defined in producer-consumer problem algorithm.

Autonomous Class	*Exclusive Class*	*Synchronous Class*
producer class consumer class	buffer class	buffer_full_sync class buffer_empty_sync class

消費者オブジェクトはバッファオブジェクトからデータを取って消費する．バッファが空の時には，待ち状態に入り，バッファ・エンプティ条件同期オブジェクトを通じて待ち状態から実行状態にされるのを待つ．もし，消費者オブジェクトがバッファオブジェクトからデータを取ってくるに成功したら，待ち状態の生産者オブジェクト（もしあれば）は解除される．

（2） バッファ排除オブジェクト

バッファはデータの倉庫であり，生産者は生産したデータをそこに格納し，消費者はそこからデータを取り出す．したがって，バッファは両者に共用されるので，排除型オブジェクトで実現される．

バッファ排除オブジェクトには，**図4**に示すよう

```
excl class buffer{
 private:
    int     *p;
    int     *head;
    int     *tail;
    int     size;
    int     buff_full;
    ss_t    s_c;
 public:
    buffer(int buff_size);
   ~buffer();
    int  deposit(int c);
    ss_t remove();
};
int deposit(int c){
    if <BUFFER IS NOT FULL>{
          *tail=c;
          *buff_full=0;
    }
    else{
          *buff_full=1;
    }
    return buff_full;
 }
```

図4 バッファ排除オブジェクト
Fig. 4 A buffer exclusive object.

に，バッファに関する操作が2つある．データを置く(deposit)メソッドとデータを取る(remove)メソッドである．

(3) バッファ・フルとバッファ・エンプティ条件同期オブジェクト

生産されたデータがバッファに入れられないときには，生産者はバッファにデータを置く空間ができるまでデータの生産を待つ必要がある．あるいは逆に，バッファが空ならば，消費者は生産者がデータをバッファに入れるまで消費を待たなければならない．この同期のため，**図5**に示すようにバッファ・フル条件同期クラスとバッファ・エンプティ条件同期クラスを設定する．

5.2 実験環境

われわれは，OMRON 社製の LUNA 88 K 並列計算機を用いて，WARASA で書いた生産者・消費者問題の並列オブジェクトプログラムを実行した．OS は Mach Version 2.5，メモリサイズは 32 M バイトであり，250 M バイトのディスクと最大4つの MC-88100 CPU (25 MHz) を有する．

5.3 実験結果と考察

この実験は，カーネルの並列プリミティブをカプセル化したオブジェクトの並列機能動作を確認すると共に，複数の自律オブジェクトが実際のマルチ CPU 環境下で同期を取りながら並列実行する効果を観察するために行った．

(1) カーネルの並列プリミティブのカプセル化

WARASA の自律オブジェクト，排除オブジェクトおよび条件同期オブジェクトはそれぞれ Mach カーネルのマルチスレッド，排他制御および同期プリミティブをカプセル化して得られるオブジェクトである．この実験で，3種類のオブジェクトが正しく動くことを確認した．これにより，オブジェクト指向パラダイムとカーネルの並列機能とを結合するような WARASA 並列オブジェクトの提案が実現できることを確かめた．

また，この簡単な実験から，ユーザは WARASA を使えば，並列機能を持つオブジェクトを利用するだけで，具体的な並列制御を書かなくても容易に並列プログラミングできることを確認した．この点に関して言えば，WARASA は初心者ユーザに役に立つと言える．しかも，従来の並列オブジェクト指向プログラミング言語と違って，これらのオブジェクトの並列制御にカーネルの並列プリミティブを利用するので，それらを使って独自の並列処理機構を作る場合よりプログラムの実行効率も悪くないと思われる．

(2) 自律オブジェクトの並列実行

この実験では，WARASA の生産者・消費者問題プログラムを用いて，生産者と消費者自律オブジェクトが異なる CPU 数を持つ並列環境で一定量のデータ(4098 個)を生成・消費する過程の処理時間を測定した(**図6**)．

図6では，生産・消費するための時間が異なる並列プログラムの実行結果を示している．横軸は並列環境の CPU 数である．縦軸は全体の実行時間である．生

```
sync class buffer_empty_syn{
   private:
      mutex_t      sync;
      condition_t  sig1;
   public:
       buffer_empty_syn(){
           sync= new mutex();
           sig1= new condition();
       }
      ~buffer_empty_syn(){
           delete  sync;
           delete  sig1;
       }
       void wait(){
           condition_wait(sig1, sync);
       }
       void signal(){
           condition_signal( sig1 );
       }
};
```

図5 バッファエンプティ条件同期オブジェクト
Fig. 5 A buffer-empty synchronous object.

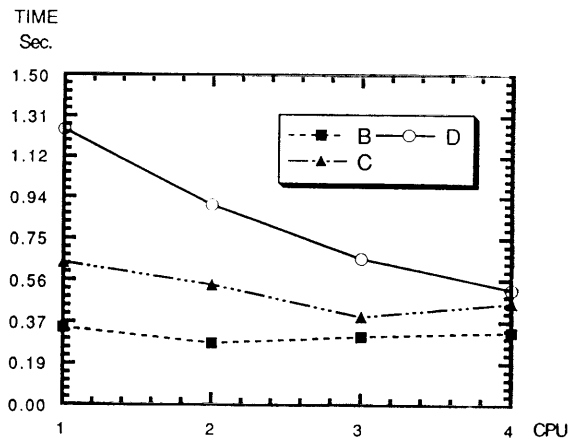

図6 マルチプロセッサ環境での WARASA 生産者と消費者並列プログラムの実行測定
Fig. 6 Producer-consumer programs measured on multiprocessor environments.

産者・消費者自律オブジェクトの生産・消費時間の関係はBを1とすると，Cは4，Dは6である．すなわち，C，Dのデータ1件の生産および消費にかかる時間はそれぞれBのそれの4倍，6倍である．

生産者・消費者オブジェクトの内部処理時間（データを生産・消費する時間）が長ければ，各々が独立に動く割合が大きく，したがって並列効果が大きい．この結果はその意味では予想通りである．

6. 他の並列オブジェクト指向言語との比較

並列オブジェクト指向プログラミング言語は高度の並列性をもつシステムのモデル化とその記述，分析および設計に有効な枠組を提供できるので，現在までさまざまな並列オブジェクト指向プログラミング言語が提案されている．実際，文献19)では14個の代表的な並列オブジェクト指向プログラミング言語があげられている．その中で，ABCL/1や，Concurrent Smalltalkおよび C++ に並列性を加味した拡張言語とした並列オブジェクト指向プログラミング言語としてはPresto, Parmace[3] などがあげられる．WARASAは C++ の拡張言語として提案された言語であるので，ここでは，Prestoなどの言語と記述能力や，実現法に関しての比較を行う．

（1） プロセス／スレッドの生成

並列に動作するオブジェクトはプロセスあるいはスレッドに対応する．プロセスあるいはスレッドを生成する方式は明示的な方式と暗黙的な方式の2種類である[19]．

Prestoの場合，オブジェクトを並列的に動かせるために，プログラマは，まず，スレッドオブジェクト(thread object)を生成しなければならない．その後，オブジェクトのメソッドと生成されたスレッドとを対応させる文を明示的に用いて並列実行を実現する．

WARASAではPrestoと違って，暗黙的なプロセス／スレッド管理方式を用いる．例えば，プログラマは従来のC++オブジェクトの生成法と全く同じnew演算子によって自律オブジェクトを生成する．しかも，自律オブジェクトは生成時点から自動的に生成元スレッドとは別の新たなスレッド上で実行される．したがって，WARASAはPrestoより並列オブジェクトを生成するプログラムをより自然に書ける．

（2） オブジェクトの喚起

多くの並列オブジェクト指向プログラミング言語，例えば，Actors[1] や Concurrent Smalltalkでは，オブジェクトはメッセージを受けることによって活動状態（active）になるといわれている[19]．Concurrent Smalltalkでは，眠っているオブジェクトにrunというメッセージを送らないと，オブジェクトがリスタートできない．そのために，オブジェクトを書くときに，それに対応する記述を書かなければならない．

WARASAでは，自律オブジェクトの宣言時に，応用に必要なメソッドだけ記述すれば良い．眠っている自律オブジェクトを活動させるための特別のメッセージは不必要である．したがって，プログラマは待ち状態の自律オブジェクトを活動させることを特別意識する必要はない．

（3） 排他制御の方法

① クリティカル・セクション（critical section）

自律オブジェクト間の共有データは排除オブジェクトとして宣言される．この排除オブジェクトはABCL/1のオブジェクトと同様に，並列処理環境で複数メッセージが同時に到着する場合，そのオブジェクトのローカルメモリ（インスタンス変数）の一貫性を保つためにメッセージを一時に1つしか受け付けないことを保証する．

Prestoでは1つのオブジェクトが同時に複数のメッセージを受け取ることを許すため，オブジェクトの私的なデータが排他制御されねばならない．これはPrestoのクリティカル・セクションがWARASAより小さいことを意味する．

② 排他制御

Prestoは，排他制御のために，ロック・オブジェクトを明示的に使って共有変数を保護する．Dallyの Concurrent Smalltalkでも，ユーザのプログラムで明示的にロック・アンロック変数を使って排他制御を行う[17]．

WARASAでは，排除オブジェクトが排他制御機能を暗黙的に提供している．ユーザは共有データを使う場合に，プログラムでロック・アンロック操作を書く必要がない．排除オブジェクトを宣言すれば，普通のオブジェクトと全く同じような使い方をしても共有データの一貫性を保持できる．

（4） 粒度の設定

WARASAの並列操作は自律オブジェクトのみに関係する．すなわち，複数の自律オブジェクトは異なるスレッドによって並列的に動作できる．しかし，1つの自律オブジェクトが実行している間に，通常のC++オブジェクトを複数個起動することがある．こ

の意味で，WARASAの並列単位である自律オブジェクトは任意の通常オブジェクトから構成できるので，Concurrent Smalltalk[20]，ABCL/1[22]言語より粒度が大きいといえる．

7. おわりに

本論文では並列オブジェクト指向プログラミング言語 WARASA の設計思想，特に，オブジェクトの並列機能の設計法およびその軽量プロセス環境上での実現について述べた．提案された3種類のオブジェクトを使った例題を実際に書き，それをハンドコンパイルして，マルチプロセッサ計算機上で実行し，設計の実現可能性を検証した．

マルチプロセッサ計算機を有効かつ容易に利用するために，マルチプロセッサ計算機 OS のカーネルが提供する並列プリミティブとオブジェクト指向とを結合した抽象化レベルの高い並列オブジェクト指向プログラム言語をユーザに提供するのが WARASA の設計目標であるので，今後より高い抽象化レベルについてさらに検討する必要があると思われる．

謝辞 本研究を行うにあたり，適切な指導と助言をいただいた九州大学工学部吉田助教授，大型計算機センタ天野助教授に感謝致します．なお，査読者からは懇切で丁寧なご助言をいただいたことを心から感謝致します．

参考文献

1) Agha, G. and Hewitt, C.: Actors: A Conceptual Foundation for Concurrent Object-Oriented Programming, in Shriver, B.D. and Wegner, P. (eds.), *Research Directions in Object-Oriented Programming*, pp. 199-220, MIT Press, Cambridge, Mass. (1987).
2) Bershad, B.N., Lazawska, E.D. and Levy, H.M.: Presto: A System for Object-Oriented Parallel Programming, *Software-Practice and Experience*, Vol. 18, No. 8, pp. 713-732 (Aug. 1988).
3) Beck, B.: Shared-Memory Parallel Programming in C++, *IEEE Software*, Vol. 7, pp. 38-48 (July 1990).
4) Carriero, N. and Gelernter, D.: How to Write Parallel Programs: A Guide to the Perplexed, *ACM Computer Surveys*, Vol. 21, No. 3, pp. 323-357 (1989).
5) Cooper, E.C. and Draves, R.P.: C Threads, Technical Report CMU-CS-88-154, Dept. of CS, CMU, Feb. (1988).
6) Hansen, P.B.: The Programming Language Concurrent Pascal, *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, pp. 199-207 (June 1975).
7) Holt, R.C.: *Concurrent EUCLID, The UNIX System, and TUNIS*, Addison-Wesley Publishing Company, Tokyo (1983).
8) Jiang, Y. and Makinouchi, A.: WARASA: An Enhanced C++ for Concurrent Programming on Shared Memory Multiprocessor Computers, *Proc. 16th IEEE Conf. on Computer Software and Applications Conference*, pp. 257-262 (1992).
9) 江，牧之内：並列オブジェクト指向永続プログラミング言語 WARASA について，第43回情報処理学会全国大会論文集，4M-11 (Oct. 1991).
10) Kepecs, J.: Lightweight Processes for UNIX Implementation and Applications, USENIX (1985).
11) Okazaki, K.: 並列ハッシュジョイン方式の共有メモリマルチ CPU 計算機上への適用と評価，Technical Report CSCE-91-C 02 (Feb. 1991).
12) Pancake, C.M.: Where Are We Headed, *Communication of the ACM*, Vol. 34, No. 11, pp. 53-64 (Nov. 1991).
13) Pancake, C.M. and Bergmark, D.: Do Parallel Languages Respond to the Needs of Scientific Programmers?, *Computer*, pp. 13-23 (Dec. 1990).
14) Quinn, M.J. and Hatcher, P.J.: Data Parallel Programming on Multicomputers, *IEEE Software*, Vol. 7, pp. 69-76 (Sep. 1990).
15) 新城，清木：並列プログラムを対象とした軽量プロセスの実現方式，情報処理学会論文誌，Vol. 33, No. 1, pp. 64-73 (1992).
16) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley, Reading, Mass. (1986).
17) Tomlinson, C. and Scheevel, M.: Concurrent Object-Oriented Programming Language, in Kim, W. and Lochovaky, F.H. (eds.), *Object-Oriented Concepts, Database, and Application*, pp. 79-124, ACM Press, New York (1989).
18) Walmer, L.R. and Thompson, M.R.: *Guide to the Mach System Calls*, Dep. of Computer Science, Carnegie Mellon University, Version of Feb. (1988).
19) Wyay, B.B., Kavi, K. and Hufnagel, S.: Parallelism in Object-Oriented Language: A Survey, *IEEE Software*, Vol. 9, pp. 56-65 (Nov. 1992).
20) Yokote, Y. and Tokoro, M.: Concurrent Programming in Concurrent Smalltalk, *Object-Oriented Concurrent Programming*, pp. 129-158, The MIT Press, London (1987).
21) Yokote, Y. and Tokoro, M.: Object-Oriented

Concurrent Programming: An Introduction, *Object Oriented Concurrent Programming*, pp. 1-7, The MIT Press, London (1987).
22) Yonezawa, A., Briot, J. P. and Shibayama, E.: Object-Oriented Concurrent Programming in ABCL/1, *Proc. ACM Conf. on OOPSLA '86*, pp. 258-268 (1986).
23) Yoshida, M. and Tanaka, H.: Intra-Object Parallelism on Parallel Object Oriented Languages, *JSPP '91*, pp. 245-252 (1991).

(平成5年4月28日受付)
(平成6年3月17日採録)

江　　允（正会員）

昭和58年中国北京計算機学院計算機技術学科卒業．同年より北京計算機学院計算機技術学科助手．平成2年九州大学工学研究科情報工学専攻修士課程入学．平成4年九州大学工学研究科情報工学専攻修士課程修了．現在，九州大学工学研究科情報工学専攻博士課程在学中．並列オブジェクト指向プログラミング言語の研究に従事．分散データベース，並列・分散処理に興味がある．IEEE学生会員．

牧之内顕文（正会員）

昭和42年京都大学工学部電子工学科卒業．昭和45年グルノーブル大学理学部応用数学科 Docteur-Ingènieur 取得．同年富士通(株)入社．以後，コンパイラ-コンパイラ，データベース，知識ベース，自然言語インタフェースの研究開発に従事．京都大学工学博士．(株)富士通研究所を経て現在九州大学工学部教授（情報工学科）．ACM, IEEE Computer Society, 人工知能学会各会員．