C++ 言語への並行性の導入

3G－8

崔梗淙　村山尚　原田賢一
慶應義塾大学理工学部

1　はじめに

近年，マルチプロセッサシステムが普及している．また，分散並行型の計算も一般化しつつあり，そのための有効な利用技術が必要となっている．オブジェクト指向プログラミングにおいてはオブジェクト間でメッセージを送受することで問題を解くように記述するが，そのための言語は，オブジェクトを並行に動作するように記述できることが望ましい．本研究では，オブジェクト指向言語 C++ ほとんど変更することなく並行性を導入する方法を提案する．

2　目標

本研究では，既存の言語の枠組を出来るだけ変更すること無く，並行性を導入するために，以下のような目標を設定した．

- 基底となる言語として，広く使われている C++ を使うこと
- C++ に並行実行の機能を付け加えること
- 言語には必要最小限の変更のみを加えること
- 小さなオーバヘッドで実用的なプログラムに使えること

3　並行性の導入

並行実行の単位としては，オブジェクト，プロセスなどが考えられるが，ここではオブジェクトを並行実行の単位として用いる．本研究では，C++ にもとから備わっているクラスに加えて，そのインスタンスとして内部に単一のスレッドを持つクラスを導入する．これを**並行クラス**，そのインスタンスを**並行オブジェクト**と呼ぶ．これにより複数の並行オブジェクトが同時に動作することができるようになる．また，従来のクラスとオブジェクトをそれぞれ**逐次クラス**，**逐次オブジェクト**と呼ぶ．

3.1　並行オブジェクト

並行クラスは，そのことを示すために class の代わりに cclass というキーワードを用いて定義する．インスタンスの生成などは逐次クラスと同様である．

```
cclass Conc_Class{
    private:    // private member
        .....
    public:     // public member
        .....
};

Conc_Class obj;
```

並行オブジェクトは，他の並行オブジェクトと並行に動作可能なオブジェクトである．実際には，他からのメッセージの受信によって，そのメッセージに対応したメソッドが起動される受動的オブジェクトである．また，一つのオブジェクトには一つのスレッドだけが与えられるために，あるメッセージに対する動作を行っている間に到着したメッセージは到着順にキューに蓄えられる．ただし，並行オブジェクトが自分自身に送るメッセージは，デッドロックを防止するために，通常の C++ のセマンティックで行われる．

並行オブジェクト間のメッセージ通信には，同期通信と非同期通信がある．同期通信では，メッセージを送ったオブジェクトは，その結果を得るまでブロックされる．これに対して，非同期通信では，メッセージを送った後も実行を続けることができる．結果を必要とする場合は，結果を得るために C++ のテンプレートを用いて定義されるオブジェクトを用いて，次の例のように記述できる．結果が用いられようとした時点で，返信が無い場合は，結果が送られて来るのを待ち，そうでない場合は，ただちに結果を取り出すことができる．

```
Conc_Class_A *a;
Conc_Class_B *b;
future<rettype> ret;

    x = b->func();      // 同期通信
    (void)a->func1();   // 非同期送信
    ret = a->func1();   // 後で結果を
                        // 使う非同期通信の例
    ....
    x = ret.value() + ...;
```

3.2　逐次オブジェクト

逐次オブジェクトは，通常の C++ でのオブジェクトを指す．このオブジェクトを残しておくことですべてのオブジェクトを並行オブジェクトにするよりもオーバヘッドを軽減することができる．ただし，並行オブ

Introducing Concurrency to C++ Language
Kyong-Rok Choi, Hisashi Murayama and Ken'ichi Harada
Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama 223, JAPAN

ジェクトから同時に一つの逐次オブジェクトの内部状態を変更するようなことが無いように, プログラマは注意しなければならない.

3.3 継承

並行オブジェクトも, 逐次オブジェクトと同様に継承を行うことができる. 並行クラスから継承されるメソッドは, 並行クラスで定義されたメソッドと同じように, 他の並行オブジェクトのメソッドと並行に実行される. ただし, 逐次クラスから継承されるメソッドの呼出しは, 逐次的に行われる.

4 実装

並行オブジェクトは, Sun Workstation 上で SunOS が提供する軽量プロセス (Light Weight Process) ライブラリを用いて, 並行オブジェクトに対して一つのスレッドを作ることによって, 実現する. 軽量プロセスは, 通常のプロセスと同様に OS のカーネルがスケジューリングを行うが, アドレス空間などの各種のプロセスの資源のほとんどを共有するために, オーバヘッドが小さい.

4.1 軽量プロセス

SunOS の軽量プロセスライブラリは, スレッドの生成, 廃棄, スレッド間のメッセージ通信, 条件変数やモニタの操作, 例外処理, 非同期のイベントの例外へのマッピング及びスレッドのスケジューリングなどのための機能を提供する.

4.2 前処理系

本研究で作成した前処理系は, 並行オブジェクトを含むプログラムを軽量プロセスライブラリを用いた C++ のコードに変換する.

`cclass` を用いて記述されたクラスは並行支援ライブラリに含まれるクラスのサブクラスに変換される. 変換後の並行クラスには, その構築子でスレッドを生成するようなコードが追加され, 並行オブジェクトとのメッセージ通信は, 軽量プロセスライブラリの提供するスレッド間のメッセージ交換のプリミティブを使ったものに置き換えられる. これにより, プログラマは, 軽量プロセスを扱うための原始的なコードを意識せずに, プログラムを記述することができる. 逐次オブジェクトへのメッセージの送信は変換されない.

実際には直接, 軽量プロセスライブラリを用いるわけではなく, 以下に述べる並行性支援ライブラリを利用している. これにより, 軽量プロセスが支援されない OS でもこのライブラリを作成, あるいは他のスレッドライブラリを用いるように変更することで移植が可能である.

4.3 並行性支援ライブラリ

並行クラスの親クラスとなるクラスや, スレッドのスケジューリングや並行オブジェクトとのメッセージの通信, 逐次オブジェクトの局所記憶をより安全にアクセスするためのルーチンなどは並行性支援ライブラリとして用意しておく. 前処理系で出力された C++ のコードは, このライブラリに含まれるクラス, サブルーチンを利用する.

5 まとめ

互換性を保ったまま C++ に並行性を導入する方法として, OS の軽量プロセスを用いる方法を提案した. 並行処理の単位として並行オブジェクトを導入し, 同期・非同期の通信を可能にしている. また, すべてのオブジェクトを並行オブジェクトとするのではなく, 逐次オブジェクトと共存することで, 効率の低下を抑えることができる. これらは前処理系と並行性を支援するライブラリによって比較的簡易に実現できた.

最近では単なる並行処理に加えて, 分散処理が計算機利用で一般的になりつつあり, さらに分散処理を自然な形プログラミングできるように導入することが今後の課題である.

参考文献

[1] H. Assenmacher, T. Breitbach, P. Buhler, V. Hübsch and R. Schwarz, "PANDA — Supporting Distributed Programming in C++," *ECOOP '93 Object — Oriented Programming*, Lecture Note in Computer Science 707, 1993.

[2] R. Trehan, N. Sawashima, A. Morishita, I. Tomoda, T. Imai and K. Maeda, "Concurrent Object Orient "C"," *ACM SIGPLAN Notices*, February 1993.

[3] Sun Microsystems Inc.,"SunOS 4.1 Manual Programing Utilities and Librares," 1990.

[4] M. A. ELLIS and B. Stroustrup, "THE ANNOTATED C++ REFERENCE MANUAL," *Addison Wesley Publishing Company*, 1990.

[5] D. Keppel, "Toos and Techniques for Building Fast Portable Threads Packages," *Techinical Report UWCSE 93-05-06*, University of Washington, 1993.