4G-9 並行プログラミング言語に対する高水準デバッグシステム

渡辺慎哉　　宮本衛市
(北海道大学　工学部)

1. はじめに

最近のマルチプロセッサ技術の発展や分散システムの流行を反映して、occam のような並行プログラミング言語や、Ada、Modula-2 など構文中に並行記述を取り入れたプログラミング言語の開発が活発になっている。これに伴い、並行プログラムのためのデバッグシステムが幾つか提案されている[1),2)]。これらのデバッガは、並行プロセス群にまたがるイベントの動的な振舞いに着目し、異常な振舞いを発見する手段の開発に主眼を置いている。今回開発中のデバッグシステムは、異常の発見に加えて、その原因を探索するための情報をプログラマに提供することができるシステムを目指している。

2. 並行プログラムのデバッグにおける問題点

並行プログラムをデバッグする際に問題となる事柄を列挙すると、以下のようになる。

a) プロセスの内部状態が、同時に実行されている他のプロセス群により影響を受ける。この結果、1つのプロセス内の調査だけでは、原因の究明は難しい。

b) プログラムの実行に再現性がない。これはプロセス間通信等の実行順序に非決定性が存在するためであり、デバッグのための再実行を困難なものとしている。

c) デバッガの介入により、命令の実行順序が影響を受ける。

並行プログラムのデバッグにおいては、ここに挙げた問題点を何らかの方法で解決する必要がある。

3. デバッグシステムの概要

本システムの構成をfig.1 に示す。デバッグはソースプログラムのレベルで行ない、対象とする言語は、そのモデル的性質・簡潔性から HoareのＣＳＰを選択した。また、前節で述べた問題点の１つである、デバッガの介入による特定プロセスの遅延を除去するために、単一ＣＰＵ上での論理時間に基づいたシミュレーションによりプログラムの実行・デバッグを行なう。これにより、遅延時間からくる問題点を解消することができる。また、実行の履歴を保存することにより、実行の再現・逆トレース等の機能も実現している。

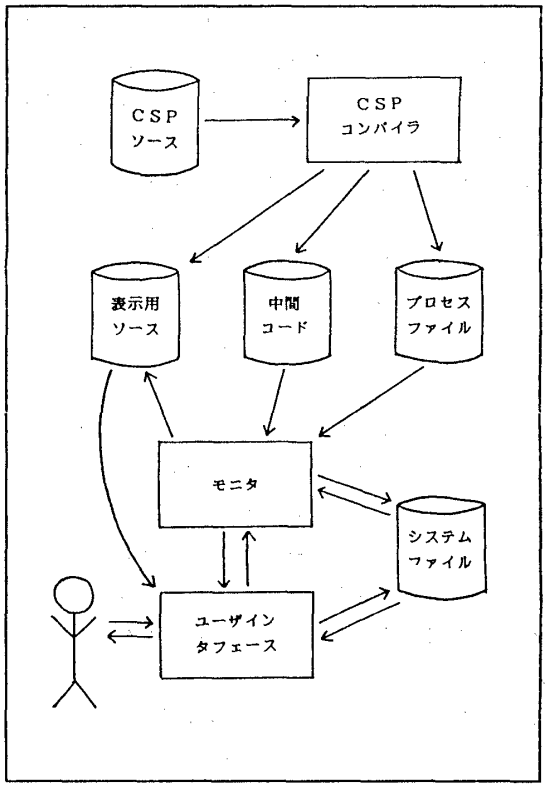

fig.1　システムの全体図

4. 異常発見の手段

並行プログラムにおいては、複数のプロセスが複雑にかかわり合いながら処理が進行していく。これに対応するため、デバッグシステムは、複数のプロ

High-Level Debugging System for Concurrent Programming Language
Shinya WATANABE, Eiichi MIYAMOTO
HOKKAIDO, Univ.

```
[ P:: *[... → produce data; B!data]
‖B:: buf(0..9),in,out integer;
    *[ in<out+10; P?buf(in mod 10)
                → in:=in+1
     ‖out<in; C?more()
                → C!buf(out mod 10);
                  out:=out+1
     ]
‖C:: *[... → B!more(); B?data;
                consume data]
]
```

fig.2　buffer プロセス

セスにまたがるデバッグ手段を用意することが重要である。本システムでは、プログラマが思惑として持っている、プロセス間通信の動的実行順序を順路式(Path-Expression) の形でデバッガに与え、その情報を基に異常を発見する手段を用意している。fig.2 はbufferプロセスのプログラム例であるが、この例に対する順路式は、次のようになる。

path　*(P→B,(C→B ; B→C))　end

ここで、$P_i \rightarrow P_j$は、プロセスP_iからプロセスP_jにメッセージが渡されたことを意味する。 ユーザインタフェースは、この式を有向グラフに変換してシステムファイルに登録する。モニタは、この登録されたグラフと実際の通信パスとの比較を行ない、通信の異常を発見する。この機能に加えて、細部のデバッグのためにアサーションやトレース・逆トレース等の機能も本システムでは用意している。

5. 原因探索のための手段

4.節で述べた機能を用いて異常を発見した場合、その原因となっている部分は、それ以前に存在している。通常、プログラマは、異常が発生した状態を参考にして、その異常が発生した原因を探索する。しかし、並行プログラムにおいては、通信経路を通じて他のプロセスから影響を受けるため、プログラマは、原因の探索を他プロセスにまで拡げなければならない。そこで、本システムでは、この探索を効率化するためにデバッガが保存されている履歴を用いて変数の依存関係を調べ、それをプログラマに提供する。例えば、fig.3aに示した実行の履歴から変数Xに対する依存関係を図示すると、fig.3bのようになる。プログラマは、この情報を基に変数Xの異常の原因を複数のプロセスに渡って探索することができる。

6. おわりに

並行プログラムにおける、デバッグ上の問題点を示し、それを解決するためのデバッグシステムの構築について述べた。このシステムでは、異常を発見するだけではなく、その原因探索に助言を与える機能を導入する試みも行ない、デバッグの効率化を目指したものとなっている。このシステムにおける問題点は、履歴のための記憶容量とループへの対応や、原因探索時のオーバーヘッドなどが考えられ、省スペース化・高速化に対する対応が必要である。

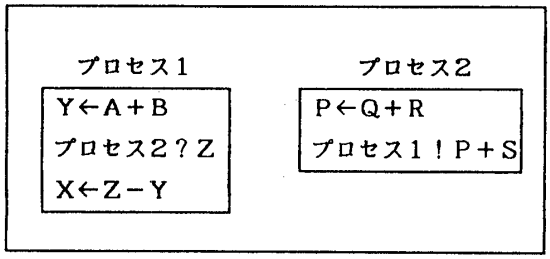

fig.3a　実行の履歴

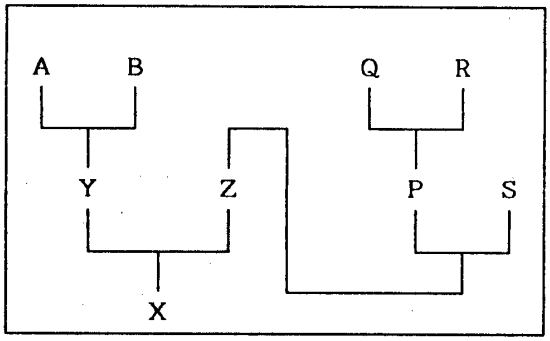

fig.3b　変数の依存関係

〈参考文献〉

1) P.C.Bates and J.C.Wileden, "EDL:A Basis For Distributed System Debugging Tools," In 15th Hawaii International Comference on System Sciences, pages 86-93. (1982)

2) F.Baiardi,N.D.Francesco and G.Vaglini, "Development of a Debugger for a Concurrent Languag e," IEEE Trans. on SE, Vol.12, No.4(Apr.1986), 547-553.