超並列プログラムでのデバギングの考察

4H－7

関田 大吾　市吉 伸行　西岡 利博　吉光 宏
技術研究組合 新情報処理開発機構 超並列 MRI 研究室

1　はじめに

本研究室では一般的に困難であるといわれている並列プログラムのデバギングの環境について本年度より研究/開発を開始した．研究をはじめるに当たってその方向を定めるため，並列プログラムのデバギングを困難にしているものは何か，どうすればこの困難は克服できるのかという点について考察を行なった．

2　並列プログラムをデバグする上での問題点

一般に逐次プログラムとは1つしかスレッドを持たないプログラムのことである．したがって全てのプログラム箇所の前後関係は明らかに定まる．しかしながらスレッドを複数本もつ並列プログラムでは，スレッドを跨った実行の順序は一般的には定まらない．このため「同期」が必要となるが，その同期記述の不足/誤りのため，スレッド間で必要な順序の保証を満たせなくなる，という並列特有のバグが発生する．つまり，「非再現性」「非決定性」と呼ばれる並列特有の現象が起こることになる．

この「同期バグ」はとることも厄介なものである．何故なら，同期バグの入ったプログラムには必要なプログラム実行順の保証がないため，プログラム実行順は実行毎に異なる可能性があり，その結果問題にしている状況が特殊なケースでしか再現しないことになり易い．

また，逐次プログラムでは，プログラムの実行順は，ユーザが記述した通りの，いうなればプログラム記述者が期待する順で実行が行なわれ，その実行を単純な一次元的「実行ログ」として提示するのみでも簡単に理解することができた．

しかしながら，並列プログラムではこれも成り立たない．スレッドが多数存在する並列プログラムにおいて単純に実行順に並べられた実行ログから期待された動作が現実に行なわれているかどうかを確かめることは容易ではない．さらに，複数スレッドを別々の並びとして提示することも可能であるが，スレッドの本数が多くなった時には全てを見渡すことは容易ではない．

A Study of Debugging of Massively Parallel Programs
Daigo SEKITA, Nobuyuki ICHIYOSHI, Toshihiro NISHIOKA, and Hiroshi YOSHIMITSU
Massively Parallel MRI Lab., Real World Computing Partnership 2-3-6 Otemachi, Chiyoda-ku, Tokyo 100, JAPAN

3　並列向きのデバギングツール

一般的にデバギング手法は「実行前」「実行時」「実行後」の3種類に大別される．

- 実行前に，プログラムを直接静的に解析する(以後これを「実行前デバギング」と呼ぶ)．
- 実行時に，プログラムを一時的に止め，プログラム実行の状況を観察する(以後これを「実行時デバギング」と呼ぶ)．
- 実行時にプログラムに関するデータを出力しておき，プログラム実行後にそれを観察する(以後これを「実行後デバギング」(＝ ポストモーテム型デバギング)と呼ぶ．

3.1　実行前デバギング

実行前デバギングは，そのプログラム記述対象言語，さらにはプログラミングモデルに大きく依存する．先に挙げたような「同期」がその言語/モデル意味論より明確にされている，もしくは，その言語/モデル機能により同期が問題とならぬようになされているのであれば，実行前のデバギングでこの問題はかなり対処可能あろう．[1]つまり，「実行前」デバギングを有効に活用するためには，十分明確に同期機構を取り込んだ言語/モデルを用いることが必要である．仮に言語自身が十分な同期機構をサポートしておらずとも，明解な並列プリミティブのみを利用可能としているようなプログラミング環境では同様に実行前デバギングは有効であると考える．

しかしながら，十分な同期機構が取り込まれていない場合や，また，十分な同期機構が取り込まれている場合であっても，その機能の誤用などにより，全ての場合についてこの同期の問題が実行前に対処できるとは限らない．よって，実行時/実行後デバグは依然として必要であり，(その利用頻度は別としても)これらで同期の問題に対して対処を行なう必要がある．

3.2　実行時デバギング

実行時デバギングはプログラム(つまりスレッド)を停止させるが，その停止によって着目している部分の実行状況が非デバギング時と違ってしまったり，観察中に

[1]例えば，並列オブジェクト指向言語，関数型言語，論理型言語といった枠組を利用した暗黙の同期を導入している言語はこの範疇にはいる．

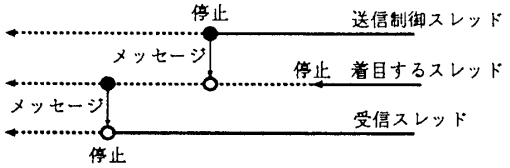

図 1: 観察の時間差の例

変更されてしまったりしては正しい観察はできない．そのためには少なくとも，着目している部分に影響を与えるようなスレッドを停止させる必要がある．さらに，同期のバグが存在する時には，その結果，プログラムの挙動が全く異なったものとなることも考えられる．この場合は，非決定性を起こす根本部分を実行時に記録しておき，その記録を元に，同じ実行を再現するような再現実行と呼ばれる仕組みが有効である．

しかし，実行時デバギングに求める機能によっては，単に同じ結果を保証するのみでは不十分であることも考えられる．

例えば，メッセージ通信によってのみスレッド間のやりとりを行ない，受信と送信は必ず同期をとって行なわれる(つまり，通信相手の準備ができていないような場合には送受信操作はブロックされる)ようなプログラミングモデルを仮定する．この場合，あるスレッドを停止させた場合，そのスレッドとメッセージの送受信をする全てのスレッドは送受信の時点で停止する．その結果このようなプログラミングモデルの上ではあるメッセージ受信点での通信相手スレッドをあらかじめ決定しておき，他のスレッドからのメッセージは受けとらないようにする「再現実行機構」を用意すれば実行結果は一定となる．

しかしながら，この場合でもなお実行時デバギングをサポートするためにはまだ問題がある可能性がある．このモデルでは停止されたスレッドとメッセージの送受信をする部分以外では全く停止は起こらない．したがって，もし，停止されたスレッド以外の状態を調査も行なうのであれば，その調査タイミングによりその調査結果が異なる．また，着目されるスレッドと直接/間接にメッセージのやりとりを行なわないスレッドは停止しない(図1参照)．つまり，「あるスレッドを停止させる」ということは他のスレッドに対してどのような影響を与えるべきか，という意味を，単に結果を保証するという以外にも，デバギングのために別途設定する必要があるかもしれない．

3.3 実行後デバギング

実行後デバギングでは，制御スレッドを止めることは通常は必要がない．したがって，実行順の乱れにより着目点の実行状況に与える可能性ははるかに少ないため，並列システムでは有利である．しかしながら，実行時にデバギング用データを記録するためのオーバヘッドは零ではなく，そのオーバヘッドにより着目点の実行状況に影響が出る可能性は存在する．その可能性をできる限り小さくするためにはそのオーバーヘッドをできる限り小さくするより方法はない．

我々は実行時にできるだけ低いオーバヘッドで実行時に必要な情報をとり，その情報を実行後に解析することによりバグの所在を突き止める方法を検討中である[2].

4 情報の提示

以上に挙げたような問題以外にも，先に挙げた様に，ユーザに対してどのように情報の提示を行なうことがデバギング時にユーザのプログラムの挙動を理解させるのに役立つか，という問題が存在する．

先に挙げたように実行ログの順について考えると，実行ログをユーザが観察する時にユーザは自分のもつ「実行イメージ」に合わせてそのログの順を変更しながら確認を行なうこととなり，この労力は決して小さいものではない．このユーザの実行イメージに基づいた情報の提示を行なうことが可能であれば，デバグの労力は大きく削減できると考える．

例えば，「データ並列」プログラミングモデルにおいては，個々のデータ要素に対しては同質のスレッドが作用する，という性質を利用することにより「配列」というデータ群をひと塊に表示することにより，容易に理解できる形式でデバギング用の情報を提示することが可能と考える．同様の事柄を他の様々なプログラミングモデルについても可能とすることを検討している．

この実行イメージは対象プログラムが基づいているプログラミングモデル，さらには対象プログラムの問題領域の情報まで取り込むことも考えられる．つまり，一般的には情報提示の方法を何らかの方法でプログラム可能としておくべきであろう．

5 まとめ

並列システムでのデバギングについての問題点を洗いだし，その対処方法について考察した．ここでの考察に従い，並列プログラミングのデバギングの問題を克服したプログラミング環境を検討する．

参考文献

[1] 市吉伸行, 他. 超並列マシン RWC-1 における再現実行方式の検討. 第48回情処全大 4H-8, 1994.

[2] 西岡利博, 他. 並列プログラムのポストモーテム型デバッグ環境に関する考察. 第48回情処全大 4H-6, 1994.