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

4H－6

西岡 利博　市吉 伸行　関田 大吾　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

並列プログラムのデバッグは、逐次プログラムのそれに比べて難しいと言われる。その最大の理由は、実行スレッドが複数存在することにより、逐次プログラムに比べて非決定的な要因が多くなり、プログラムの実行の再現性に乏しい点である。すなわち、実行ごとに異なる挙動を示し、毎回違う場所でエラーが生じるなどの状況に陥ってしまう。これに対し、デバッグを実行終了後に行なう、いわゆるポストモーテム型のデバッグは、この問題の対策として有望である。超並列 MRI 研究室では、RWC プロジェクトにおける超並列ソフトウェア分野の研究の一環として超並列プログラミング環境の研究に着手しているが[3]、その中でこの問題に対する考察を行なったので、本稿にて報告する。

2　ポストモーテムデバッグ

逐次プログラムのデバッグには、インクリメンタルデバッグと呼ばれる手法が多用される。すなわち、(例えば dbx の様な) スレッド追跡ツールを用い、ブレークポイントをかける場所を変えながら同じプログラムを何度も実行し、次第にバグの識別を行なう手法である。しかし、これは逐次プログラムの実行に再現性があることに依存したデバッグ手法であって、一般に再現性に乏しい MIMD 型の並列プログラムのデバッグに適用するには困難が伴う。

ポストモーテムデバッグとは、実行時に何らかの方法で記録された実行に関する情報を、実行終了後に解析し、バグの識別を行なうデバッグ手法である。ポストモーテムという概念でまとめることができるデバッグ手法には以下のようなものがある。

A Study of a Post-Mortem Debugging Environment for Parallel Programs
Toshihiro Nishioka, Daigo Sekita, Nobuyuki Ichiyoshi and Hiroshi Yoshimitsu
Massively Parallel MRI Lab., Real World Computing Partnership
3-6, 3-Chome, Ohtemachi, Chiyoda-ku, Tokyo 100, Japan

イベントトレース

イベントトレースは代表的なポストモーテムデバッグ手法である。

実行スレッドが予め設定したトレースポイントに達した時点でログを出力し、実行後にログを観察/解析することで障害を識別する。十分なログを出力させれば、プログラムの再現性によらない、デバッグのための情報が入手できる。

バグを局所化するためにどの情報を得ればよいか (例えば、主な手続きの入口でのパラメータの値など) が予めわかっている場合には、自動的な手段でバグを迅速に局所化することができる。

性能プロファイリング

性能プロファイラも、実行時に蓄積した情報を実行後に解析するという点で、性能上のバグをデバッグするためのポストモーテムデバッガと見ることができる。

例えば、 Pablo[2] は、イベントトレースに基づく性能解析環境である。通常のイベントトレースツールと同様、どこで性能情報をトレースするかを指定できたり、ユーザ定義のプローブを作成して挿入したりすることができる。

再現実行

再現実行とは、プログラムの実行を、後で再現して実行するデバッグ機能であり、一般に実行の再現性が問題となる MIMD 型並列プログラムのデバッグには大きな福音となり得る。本研究室でも、プログラムの再現実行に関する研究を進めている[4]。

再現実行のためには、最初の実行時に、十分な情報を記録する必要がある。この実行記録も実行時の情報の一種だと考えれば、再現実行をスレッド追跡するようなデバッグも、一種のポストモーテムデバッグとみなせる。

3　再現実行に基づくポストモーテム型統合デバッグ環境

ところで、ポストモーテム型のデバッグツールにも、以下のような問題がある。

イベントトレース 多くのプロセッシングエレメントが動作する並列プログラムでは特に、詳細なログを取るとトレースファイルが大きくなり過ぎてしまったり、実行に対する擾乱が大きくなり過ぎて、本来の実行の姿がとらえられないことがある。これに対する対策の一つとして、最初おおまかにトレースを行ない、障害が局所化されるにつれ徐々にトレースも局所化/詳細化してゆく、インクリメンタルトレースと呼ばれる手法がある。しかしインクリメンタルトレースは、実行に再現性がなければ効果が期待できない。

性能プロファイル 多くのプログラムでは、何度か実行しても性能上の特徴にそれほどの差はない。しかし、特に細粒度並列プログラムの場合、アーキテクチャやアルゴリズムの問題から、実行ごとに性能上の特徴が変化するような性能バグも存在する。このようなケースでボトルネックを特定するのは難しい。

ここでの問題は、並列プログラムの再現性である。よって、前節で述べた再現実行機能を基礎とし、各種ポストモーテム型デバッグツールを統一されたインタフェースのもとに有機的に結合した統合デバッグ環境が有望である。イベントトレースや性能プロファイラが、再現性のある実行のもとに動作することで、さらに効果的なデバッグが可能となるだろう。しかも、統合されていることによる、それ以上の効果も期待できる。それは単にインタフェースの統一に伴う操作性の向上だけには留まらず、例えば、以下のようなことが可能となるだろう。

インクリメンタルトレースで局所化したバグを、スレッド追跡ツールで識別するときには、トレース中のあるイベントが発生したその時点で実行がブレークされるようにブレークポイントを設定したいことがあるだろう。統合デバッグ環境であれば、イベントトレース中に、再現実行用に記憶されている情報を一緒に書き込んでおくことで、その機能を実現できる。

また、時系列でグラフ表示している性能プロファイル中に異常を発見した場合、イベントトレース情報をもとに、そこで何が起きたかを突き止めたいことがあるだろう。統合デバッグ環境であれば、再現実行用に記憶されている情報を頼りに、性能プロファイル情報とイベントトレースの同期をとれるだろう。

再現実行機能の上に、いわゆるシンボリックデバッガと性能プロファイラを統合した統合デバッグツールとして、ParaRex[1] がある。本研究では、ParaRex が達成した環境に加え、さらに、イベントトレースの自動解析などを駆使できる多角的なデバッグ環境を目指す。また、RWC プロジェクトで開発中の RWC-1 のような、細粒度超並列アーキテクチャにもスケールさせることを目指す。すなわち、超並列アーキテクチャ上でもスケーラブルに動作し、なおかつ、プロセッサ間通信が比較的多い、細粒度並列プログラムのデバッグでも、性能が著しく低下することなく動作させることを目標とする。

4 まとめ

ポストモーテム型デバッグ環境について考察した結果、再現実行機能を基礎とする統合デバッグ環境が有望であることが分かった。今後、この方針をもとに、実現に向けて検討を始める予定である。

参考文献

[1] E. Leu and A. Schiper. "ParaRex: a Programming Environment Integrating Execution Replay and Visualization". In *Environments and Tools for Parallel Scientific Computing*, 1993.

[2] D.A. Read. "An overview of the Pablo Performance Analysis Environment". The University of Illinois Board of Trustees (Available by FTP), 1992.

[3] 市吉伸行, 他. "超並列プログラミング環境の検討". 情報処理学会第48回全国大会 4H-5, 1994.

[4] 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". 情報処理学会第48回全国大会 4H-8, 1994.