6L-5

# 長時間シミュレーションにおける効率的デバッグ手法の提案

辻祐喜 † 十鳥弘泰 †† 大津金光 †† 大川猛 †† 横田隆史 †† 馬場敬信 ††
† 宇都宮大学工学部情報工学科 †† 宇都宮大学大学院工学研究科

## 1 はじめに

新規アーキテクチャの開発では，ソフトウェア・シミュレータによるアーキテクチャのシミュレーション評価が必要であり，評価対象のプログラムによってはシミュレーションが長時間に及ぶことがある．シミュレータのバグとして特に問題になるものとして，長時間実行した後に初めて検出されるバグが挙げられる．バグの原因箇所を特定する方法の１つに，シミュレーションを最初から実行し直し，いくつかのポイントで変数等の値を調べることで，バグの原因となった場所を絞り込んでいく方法がある．これを何度も繰り返すことで，バグの原因を発見することが可能であるが，実行が長時間に及ぶプログラムに対し，何度も最初から実行し直す必要があるこの方法を用いてデバッグを行うことは非効率である．

本研究では，シミュレーション中にチェックポインティングを行うことによりプロセスを保存し，保存した状態から実行再開することで，バグの原因箇所を見つけ出すまでに掛かる時間を短縮する方法を提案する．

## 2 長時間シミュレーションにおけるデバッグ

一般的にプログラム中のバグの原因を見つける作業には時間を要する．なぜなら，バグが検出される直前の場所に原因があるとは限らず，バグの原因箇所を特定するために，何度も最初からプログラムを実行し直し，原因を調査する必要があるためである．

図１に，一般的によく用いられるデバッグ方法である，二分探索法を用いたバグの原因箇所特定の様子を示す．二分探索法とはプログラムの実行中，いくつかのポイントで変数やスタック等の値を調べることで，バグの原因箇所を特定していく方法である．何度もプログラムを実行し直し，繰り返し値を検査することで，バグの原因を見つけ出すことが可能である．

新規アーキテクチャを開発する際に行うシミュレーションでは，プログラムの実行が非常に長時間になることが多い．ここで，長時間経過した後にバグが検出された場合，プログラムを最初から何度も実行し直す方法では，デバッグに膨大な時間が掛かってしまう．

この問題に対する解決案として，任意の場所で実行中のプロセスを保存し，保存した状態から実行再開することで，バグの原因箇所特定に要する時間を短縮する方法を提案する．図１の点線矢印で示した範囲は，変数等の調査により，明らかにバグの原因がないと考えられる実行部分である．この実行を省略することで，バグの原因箇所に到達するまでの時間を大幅に短縮することができる．

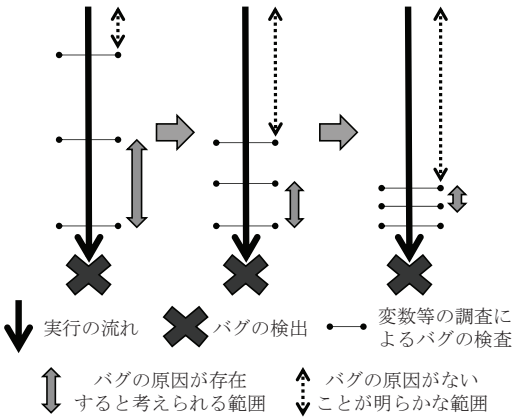


図 1: 二分探索法による原因箇所特定

## 3 チェックポインティング手法

プロセスを保存し，実行再開を実現する手段として，チェックポインティング手法を用いる．チェックポインティング手法とは，実行中のプロセスをチェックポイントファイルに保存し，保存状態からの実行再開を可能とする技術である．チェックポイントファイルから実行を再開したプロセスに対して，GDB デバッガを用いてデバッグを行うことが可能である．具体的には GDB の起動オプションとして，プログラムファイル名とプロセス ID をコマンドライン引数として渡すことで，実行中のプロセスを GDB の制御・監視の対象にすることができる．こうすることにより，対象プログラムの実行を停止させたり，変数やスタック等の値を調べることが可能となる．

現在，いくつかのチェックポインティング・ソフトウェアが開発されているが，本研究では，シミュレータおよび評価環境に対する変更は，最小限にすることを前提とする．そのため，使用するチェックポインティング・ソフトウェアは評価環境への変更が少ないものがよい．また，デバッグ所要時間短縮を目的とした場合，チェックポイントファイルの取得は，ユーザーが任意のタイミングで行える必要がある．以上を踏まえ，チェックポインティング・ソフトウェアに求められる要件は以下となる．

- 評価環境への影響が少ないこと

---

A Proposal of Efficient Debugging Method for Long-Term Simulation

†Yuki Tsuji, ††Hiroyoshi Jutori, ††Kanemitsu Ootsu, ††Takeshi Ohkawa, ††Takashi Yokota and ††Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)
Graduate School of Engineering, Utsunomiya University (††)

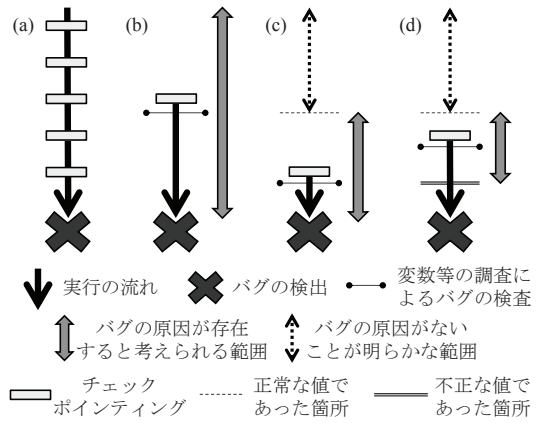


図 2: 提案手法を用いたバグの原因箇所調査の流れ

- 任意のタイミングでチェックポインティングが行えること

これらを満たすチェックポインティング･ソフトウェアとして，DMTCP (Distributed MultiThreaded CheckPointing) [1] を用いる．DMTCP は，OS への変更が不要なユーザーレベルのチェックポインティングシステムであり，評価環境への影響が少ない．また，チェックポイントファイルの取得方法として，ユーザーのコマンド入力による，手動でのチェックポインティングのほかに，指定した間隔で，自動的にチェックポイントファイルを取得する方法がある．また，ライブラリとしてチェックポインティング機能を他のプログラムから利用することができ，デバッグ対象のシミュレータから DMTCP の機能を呼び出すことで，シミュレータ中の任意の場所でチェックポインティングを行うことも可能である．

## 4 提案手法によるデバッグ

本稿では，現在我々が開発している投機的マルチコアプロセッサシミュレータ [2] に，提案手法を適用する．本シミュレータは，プログラムの実行バイナリを入力データとして与え，そのプログラムの実行サイクルや，投機実行の成功率などの統計情報を出力する．

提案手法を実現するために，具体的なチェックポイントファイルの取得位置を検討する．チェックポイントデータを取得する間隔が短い程，バグの原因箇所へ早く到達することができる．しかし，プロセッサ 4 台で並列実行する場合の本シミュレータのチェックポイントファイルのサイズは，一回につき約 200Mbyte となるため，チェックポインティングを行う回数を考慮しつつ，適切なタイミングでチェックポインティングを行う必要がある．本稿では，チェックポイントファイルの取得方法として，一定間隔でチェックポインティングを行う方法を用いる．記憶装置の容量を圧迫しない程度に一定間隔でチェックポインティングを行い，効率良くシミュレータのデバッグを行えるようにする．

図 2 は，実際にチェックポインティングを用いてデバッグを行う場合の，バグの原因箇所調査の流れを示している．初めに，一定間隔でチェックポインティングを行いながらシミュレータを実行する (a)．バグが検出された場合，本来であればバグの原因箇所を調べるために，プログラムを最初から実行し直す必要がある．ここで，およそ中間位置で取得したチェックポイントファイルから実行再開し，即座にレジスタやスタック等の値の検査を行う (b)．検査した値が正常なものであった場合，バグの原因箇所は検査位置以降にあると考えられる．次に，最も新しいチェックポイントファイルから実行再開し，再び値の検査を行う (c)．ここで検査した値が不正なものであった場合，バグの原因箇所は値の検査を行ったこの 2 点間にあると考えられる．更にこの 2 点間で取得したチェックポイントファイルから実行再開し (d)，同様にして値の検査を行う．

## 5 提案手法の有効性

提案手法の有効性を示すため，チェックポインティングを使用した場合と使用しない場合のそれぞれで，シミュレータ中の，同じバグの原因を見つけ出すまでに掛かる時間の比較を行う．

仮に，最初にバグが検出されるまでに掛かかる時間を $T_{detect}$，実行開始からバグの原因箇所へ到達するまでに掛かる時間を $T_{cause}$ とし，シミュレータを実行し直す回数を $N$ とする．チェックポインティングを利用しない場合，バグの原因を見つけ出すまでに，少なくとも $T_{detect} + T_{cause} \times N$ の時間が掛かる．一方，チェックポインティングを利用した場合，チェックポインティングを行った間隔を $T_{check}$ とすれば，実行を再開してからバグの原因箇所へ辿り着くまでに掛かる時間は最大で $T_{check}$ となり，少なくとも $T_{detect} + T_{check} \times N$ の時間でバグの原因を見つけ出すことができる．ここで，$T_{check}$ は $T_{cause}$ よりも小さい値をとるため，シミュレータを実行し直す回数 $N$ が増えるほど，相対的にデバッグに要する時間が削減できる．

## 6 おわりに

本稿では，長時間シミュレーションにおけるデバッグ時間削減の為に，任意の場所から実行再開を行う方法を提案し，チェックポインティングを利用した具体的な実現方法について述べた．

**謝辞**

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)24500055，同 (C)24500054) の援助による．

**参考文献**

[1] Jason Ansel, Kapil Arya and Gene Cooperman: “DMTCP:Transparent Checkpointing for cluster computations and the desktop,” IEEE International Parallell and Distributed Processing Symposium, IPDPS 2009, pp.1-12, 2009.

[2] 十鳥弘泰ほか: “2 パス限定投機方式を実現するマルチコアプロセッサ PALS の提案,” 信学技報, Vol.109, No.319(CPSY2009-46), pp.19-24, 2009.