B-001

補助スレッドによるソフトウェアの信頼性向上
Enhancing Software Reliability with Helper Thread

本山　英典 † 古関　聰 ‡ 小松　秀昭 ‡ 深澤　良彰 †
Hidenori Motoyama Akira Koseki Hideaki Komatsu Yoshiaki Fukazawa

1. 始めに

SMT 技術 [1] やマルチコア化などにより、CPU リソースは今後余ると考えられる。本稿ではこのリソースをソフトウェアの信頼性にあてる手法を提案する。信頼性の向上にあたり、システムの実行速度の遅延を招かないように、システムを実行するメインスレッド（MT）とエラー／バグを発見する補助スレッド (HT) によって構成する。HT は Assert 文に対して依存関係が張られているコードを MT から複製・切り取ることで生成するが、これによりスレッド間のコードには依存関係（真依存・出力依存・制御依存）が発生し、これを解消するためにデータ通信・同期が必要となる。本手法では、PDG (Program Dependency Graph) でこれらのデータの共通 Predecessor を見つけ、スレッド間の通信量を削減し、通信にバッファを用いることで出力依存を緩和させ、同期命令を削減させる。これにより MT のプログラム実行コストを最小限にする。

2. 本手法の特徴

複数のプロセスを用いたソフトウェア信頼性の手法として、Shadow Processing[2] がある。Shadow Processing はメインプロセス (MP) で ”プログラマが任意に記述した Assert 文などの脆弱性のチェック ”とそれに対して依存関係のあるコードを MP からコピーし、それをもとに Shadow Process(SP) を生成しエラー／バグをモニタリングする。2 つのプロセスはマルチプロセッサー上で実行させ、MP では本来のシステムのみを実行できるため、エラー／バグのモニタリングを隠蔽することができる。この手法では SP がすべてのコントロールフローを自ら計算するため、エラー／バグのチェックには本来不要であるコードまで実行することが必要となる。そのため、MP の速度は遅くならないものの、SP はオリジナルのコードにモニタリングコードを挿入して実行したものと同等の速さになってしまうことがある。

本手法での HT は、MT のコードを複製・変形し Assert 文の実行に必要な箇所を切り取り生成する。コードの複製・変形は、Assert 文に対して依存関係が張られているコードを PDG により見つけ、HT のために切り取るコードと本来のシステムに必要なコードとの真依存関係が最小になるように複製・変形をおこなう。変形・生成された MT・HT には依存関係が残っており、これを解消するためにデータのやり取りが必要となり、このデータのやり取りにはスレッド間のコードに対して発生する出力依存関係を考慮しなければならない。そこで、このデータに対して生存グラフを作成し、スレッド間で同期をとる方法と MT がバッファにデータをストアして HT にデータを通信する方法を考え、MT のコストが最小になる方法を選択する。この様に HT を生成し同期を取ることにより、MT のコストは最小限にとどめ、HT の実行にかかるコストも減らすことができる。

†早稲田大学大学院理工学研究科
‡日本 IBM（株）東京基礎研究所

3. 本手法の概要

始めに本手法の概要を提示し、それから HT 生成のステップに関して説明する。

本手法では、MT と HT のコード間に発生する依存関係（真依存・出力依存・制御依存）を緩和する方法として以下を適用する。

1. 真依存関係の緩和による通信量の削減 (3.1)
2. 制御依存関係の緩和による通信料の削減（3.2 節）
3. 出力依存関係の緩和 (3.3)
4. バッファによるデータの転送 (3.4 節)

HT の実行する Assert 文が EXP という値を判定する場合、EXP が $a_1 + a_2 <= a_3$ だとすると a_1, a_2, a_3 という EXP を構成する要素の値が必要となる。つまり、$EXP(a_1, a_2, ..., a_n)$ であれば $a_1, a_2, ..., a_n$ の値が必要となる。各要素の PDG を作成し共通の Predecessor が見つかれば、その値をスレッド間で通信するように MT を複製・変形して HT を生成し、HT はその値を基に自ら必要な値を導き出す。また、Assert 文に対して制御依存関係を持つ分岐の判定要素についても PDG を作成し、Predecessor を通信するか、分岐要素の真偽を通信するかを、MT のオーバヘッドが最小となるように決定する。そして、通信するデータは出力依存関係を生存グラフを生成し、バッファや同期によって MT のオーバヘッドが最小になる方法でデータを通信する。これにより、MT・HT のコード間に張られている依存関係を緩和し、MT のオーバヘッドを最小限に抑え、効率的にエラー／バグをチェックすることができる。

3.1 真依存関係の緩和

Assert(EXP) の EXP を構成する要素について PDG を作成することにより真依存関係のあるステートメントを求め、各要素の Predecessor の中で共通なものを探し、その数が最小になる値を見つけだす。

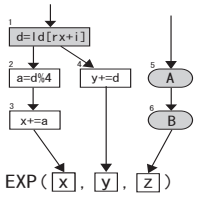

図 1: PDG による共通 Predecessor の検索

図 1 の例では Assert 文の実行に必要となる要素は 3 つあるが、x,y 共通の Predecessor であるステートメント 1 の変数 d を通信するように HT を生成する。この過程で、HT が Predecessor から要素を導き出す過程でグローバル・ヒープ変数に対する読み書きが存在する場合、

HT を生成する際に各変数への読み書きに対し私有化を適用し、要素の正確性やプログラムの正確性の破壊を防ぐ必要がある。MT はこの Predecessor の値を通信し、HT はこれをもとに自ら計算をして要素を導き出す。これを適用することにより、スレッド間の真依存関係を減少させ、通信量を削減することが出来る。

3.2 制御依存関係の緩和

分岐が Assert 文に対して依存関係が存在するとき、Assert(EXP) の EXP と同様に、分岐に実行に必要な要素に対して PDG を生成し、Assert 文の EXP と共通の Predecessor を見つける。見つけられた場合は新たに MT にオーバーヘッドはかからないが、見つけられなかった場合については分岐の要素か真偽のどちらかコストの低い方を通信する。

3.3 出力依存関係の緩和

HT が MT のデータを用いるには、その値の正確性を保つためにスレッド間のコードに張られている出力依存関係を考慮する必要がある。そこで、そのデータの生存グラフを生成して、出力依存がないものとあるものに分類する。出力依存がないデータについては、1 つのデータ生成時からそれに対して真依存関係のある Assert 文の実行までの間にあるステートメントを調べ、すべてのデータ生成の情報を MT から HT へまとめて通知する。そして、そのデータのメモリが開放される前、つまり関数終了時にスレッド間の同期を取る。出力依存があるデータについては、MT がそのデータをバッファ(3.4 参照) にストアすることにより出力依存関係を緩和させる。

3.4 バッファリング

本手法では ”通常のバッファ ”とリングバッファの 2 種類を用いる。MT が HT に受け渡すデータ数が判定可能でメモリに十分入りきる大きさの場合は ”通常のバッファ ”を用いる。また、データ数が判定不可能か容量が多すぎる場合にはリングバッファを用いる。リングバッファは数個のバッファを用い、1 つのバッファが埋まったら次のバッファへ移り、最後のバッファが埋まったら最初のバッファに戻るようにしたものである。HT は MT がどこまでデータを書き込んだかを、MT は HT がどこまでデータを読み込んだかを確認する必要がある。そこで、各バッファの最後の 1bit はフラグ専用にし、MT はすべてを書き込んだ場合にフラグを立て、HT はすべてを読み込んだ後に下ろす。

3.5 適用プロセス

以下に、HT 生成ステップを示す。以下の手順をすべての Assert 文に対して適用する。

(1) Assert(EXP) の EXP の要素の PDG を作成し、各要素の Predecessor の中で共通な値を見つける。
(2) Assert 文に制御依存のある分岐の判定要素の PDG を作成し、各要素と (1) の Predecessor の中で共通な値を見つける。
(3) (1)(2) で見つけた共通 Predecessor 数が最小となるように MT を複製・変形し、HT を生成する。
(4) スレッド間のコードで依存が残った値についての生存グラフを作成し、通信・同期方法を決定する。
(5) MT で Assert 文を実行する場合と HT に Assert 文を割り当てることによって発生したオーバヘッドを比較し、割り当てを決定する。

4. 本手法の適用例

本手法の適用例を図 2 に示す。図 2(a) の配列演算を含むプログラムに対して手法 [2] を適用した例が (b)、(a) を本手法を適用した例が (c)(d) である。手法 [2] では、

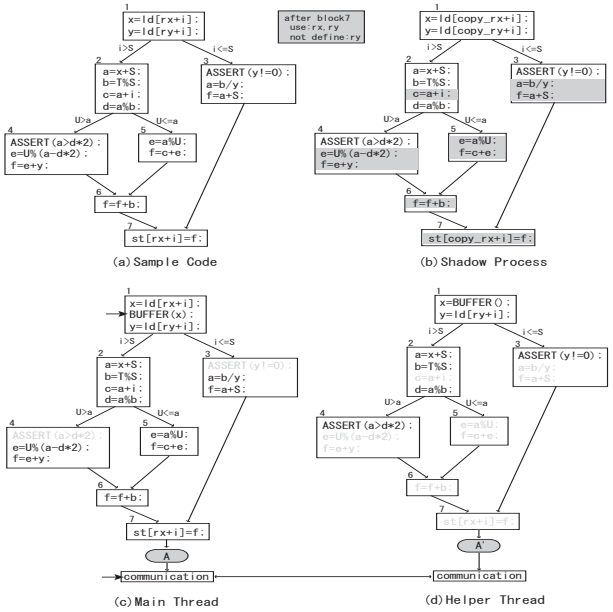

図 2: 本手法適用例

block7 の copy __ rx+i の値が次に実行される Assert 文に必要であれば、斜線部分すべての実行が必要となる。そのため、MT にはオーバーヘッドがかからないものの、HT のプログラム実行コストが大きくなり、チェック終了時に同期をとると MT が HT を待たなければならないことがある。本手法では、(c) の矢印部分のオーバーヘッドはあるが、HT のプログラム実行コストは少なく、エラー／バグチェック終了後に同期をとる場合には、HT は MT より先に実行を終えることができ、MT の遅延を引き起こさずにすむ。

5. 終わりに

本論文では HT の生成時に、MT・HT 間のコードに張られる真依存関係を最小にすることで、MT のデータ通信にかかるオーバーヘッドを削減した。そして、そのデータ通信をバッファや同期を使うことにより行い、出力依存関係を緩和し、MT の同期に必要な遅延を最小限にすることができる。

参考文献

[1] Tullsen,D.M. et al:”Simultaneous Multithreading:Maximizing On-Chip Parallelism”In 22nd International Symposium on Computer Architecture, June 1995.

[2] H. Patil and C. N. Fischer:”Efficient run-time monitoring using shadow processing”, Proc. of 2nd Inter. Workshop on AADEBUG ’95.