6J-3

動的計測ツールのバイナリ変換機能を利用した自動並列処理システムの開発

星 孝幸 † 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

1 はじめに

近年普及しているマルチコアプロセッサの性能を効果的に活用するためには並列処理が重要である．しかし，並列処理のためのコードを手動で作成するのは困難である．そこで，プログラムを自動並列処理するシステムの開発が必要となる．

本研究室では，ソースコードが参照できない場合でも並列化が行えるよう，バイナリ変換によってシングルスレッドコードからマルチスレッドコードを生成するシステムを開発している．現在は，主に静的なバイナリ変換で並列化を行っている．しかし，バイナリコードには実行時にしか分からない情報が存在することがあるので，静的に全てを解析しバイナリ変換することは困難であるという問題がある．

本稿では，バイナリコードを動的に解析することで，静的に解析が困難なバイナリコードを動的バイナリ変換によって並列化を行い，プログラムを並列実行する手法を述べる．

2 動的計測ツール Valgrind

本研究では動的計測ツール自体に変更を加えて，プログラムの並列化機能を実現するのでオープンソースであることが望ましい．また，様々なアプリケーションプログラムの並列化を行えるよう，複数の命令セットに対応している必要がある．そこで，これらを満たす動的計測ツールの Valgrind[1] を用いる．

Valgrind は，図 1 のように Valgrind core と Plug-in Tool で構成されている．Valgrind core はマシンコードから中間表現コードへ，およびその逆のバイナリ変換を行う．Plug-in Tool は中間表現コードに対して計測用コードの挿入などの様々な処理を行う．この Plug-in Tool はユーザが自由に作成し追加することができる．本研究では，Valgrind の Plug-in Tool に自動並列化機能を作成し，アプリケーションプログラムの並列実行機能を実現する．ここで，中間表現コードはプロセッサ非依存で，Plug-in Tool は中間表現コードに対して操作を行うのでプロセッサに依存しない．つまり，任意のプロセッサ上で並列化コードを生成することができる．現在，Valgrind は x86, AMD64, PPC32, PPC64, S390X の命令セット，Linux, Darwin (Mac OS X) の OS に対応している．

Valgrind は JIT コンパイラと同様の動作を行い，アプリケーションプログラムのバイナリ変換を行うことで計測用コードを動的に挿入することを実現している．バイナリ変換の流れは，まずアプリケーションプログラムのマシンコードを中間表現に変換する．次に，その中間表現コードに対して計測用コードを挿入する．最後に，計測用コードが追加された中間表現コードをマシンコードに変換する．Valgrind ではこの処理をプログラム実行時に基本ブロック単位で計測用コードの挿入を行いながら，実行することを繰り返す．また，一度変換された基本ブロックのマシンコードは変換コードキャッシュに格納され，それを再び実行するときは変換コードキャッシュ内のマシンコードを直接実行することで動的変換によるオーバーヘッドを抑えている．

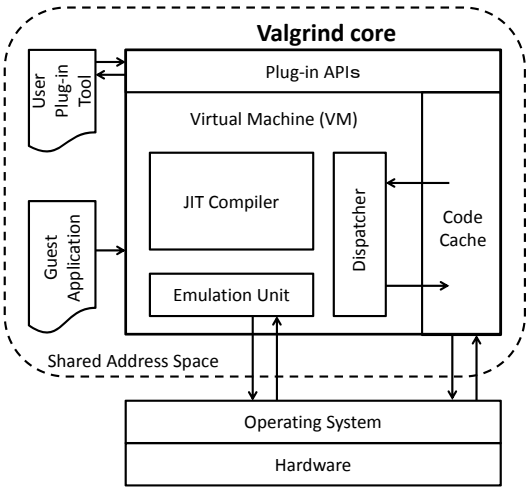

図 1: Valgrind を使ったシステムの構成

3 並列化コード生成処理

一般的に，プログラム中のループがプログラムの実行割合の多くを占めるのでループの高速化がプログラム全体の高速化につながる．そのため，本システムではループを並列化の対象としている．しかし，プログラムは単純なループ構造だけで構成されているわけではないので，並列化の割合を増やすためには複雑なループ構造も並列化できるように検討が必要である．また，並列化対象のループを動的解析によるプログラムの実行割合の情報から自動で見つける機能が必要である．

アプリケーションプログラムを並列実行するためには，Valgrind で最終的に実行される変換後のマシンコードを並列実行できればよいので，本来は中間表現コードに対して計測用コードの挿入を行っている部分

Development of Automated Parallel Processing System Based on Binary-Level Instrumentation
†Takayuki Hoshi, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

を，並列化コードを生成するように変更する．

並列実行で問題となるのがデータの競合である．それを防ぐためには，各スレッドが変更した値を書き込む順序を制御し，全てのスレッドの処理が終わるまで次の基本ブロックの処理に進まないよう同期を行うことを考慮したコードを生成する必要がある．

並列化コード生成処理では，後述するスレッドプールで待機しているスレッドで並列実行するので並列実行時にスレッド実行を開始する処理を追加する必要がある．また，そのスレッドはシステムの実行開始時に生成されたものなので並列実行対象に関する情報を持っていない．そのため，並列実行対象の情報を渡す必要があるが，本システムで最終的に実行されるのは変換後のマシンコードであるので，並列化対象を並列実行するためには変換後のマシンコードが格納されているアドレスを渡せばよい．

図 2 に本システム内での処理の遷移を示す．バイナリ変換の流れは，まずアプリケーションプログラムのマシンコードを中間表現コードに変換する．次に，中間表現コードに対して並列化コード生成処理内で操作を行う．最後に，その中間表現コードをマシンコードに変換する．本システムでは並列化対象以外に対しては，中間表現コードに変更を加える必要はないので，受け取った中間表現コードを変更せずにそのままバイナリ変換処理に渡す．並列化対象の基本ブロックであった場合，並列化コード生成処理内では受け取った中間表現コードを並列実行可能なように変更する．具体的には，ループを並列化対象にしているのでループのイテレーションを各スレッドで分割して実行できるようにする．そして，並列化された中間表現コードをマシンコードに変換する．最後に，並列化されたマシンコードをあらかじめ生成しておいたスレッドで並列実行する．以上の処理を実行される全ての基本ブロックに対して行う．

4 スレッドプールによるスレッド生成コストの削減

本システムはシングルスレッドプログラムであるアプリケーションプログラムを，実行時にバイナリレベルで並列化を行いマルチスレッド実行するので複数のスレッドを生成する必要がある．

本システムは現在 Linux OS 上で開発を行っているが，Linux OS ではスレッドを生成する方法として clone システムコールがある．clone システムコールはメモリ空間を共有したスレッドを生成する．しかし，システムコールはカーネルを呼び出して処理を行うのでコストが大きい．スレッド生成をマルチスレッド実行毎に行っていては，スレッド生成にかかる時間がオーバーヘッドになる．

そこで，システムコールの呼び出しを最小限にするために，一度生成したスレッドを何度も使いまわすスレッドプールを用いる．これにより，実際に clone システムコールを実行してスレッドを生成するのはシステムの実行開始時のみで済む．

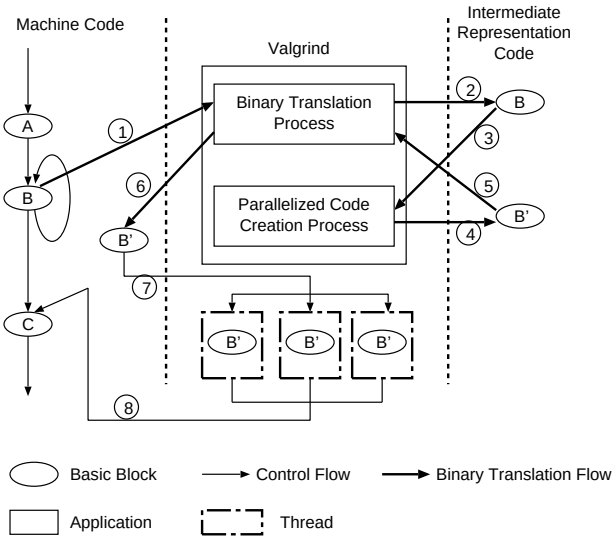

図 2: 本システム内での処理の遷移

5 開発の現状

現在，clone システムコールを用いて Valgrind 内でスレッドを生成し，ソースコードレベルで待機させたスレッドを任意のタイミングで実行を開始することを実現した．

本システムは，ユーザが手動で並列化する基本ブロックのアドレスを Valgrind のソースコードに記述することで並列化コード生成処理に情報を与えている．それにより，ユーザが指定した基本ブロックに到達したときだけ並列化コード生成処理内で中間表現コードを操作できることを確認している．

Valgrind は基本ブロックを単位としてプラグインに処理を移すため，そのままでは複数の基本ブロックをまとめて処理することができない．現在，この問題に対処するための技術を開発中である．

6 おわりに

本稿では，動的なバイナリレベル並列処理を実現するために，バイナリ変換から実行までを実機上で行うことができる動的計測ツールを利用した自動並列処理手法について述べた．

今後，Valgrind に実装中の自動並列処理機能を完成させ性能評価を行う予定である．

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)21500050，同 (C)21500049) の援助による．

参考文献

[1] Nicholas Nethercote and Julian Seward: “Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation”, Proc. of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), pp.89-100, 2007.