5K-10

Valgrindを利用した自動並列処理システムにおけるスレッドの実行制御

星 孝幸 † 大津 金光 † 大川 猛 † 横田 隆史 † 馬場 敬信 †
† 宇都宮大学大学院工学研究科情報システム科学専攻

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スレッドレベルの並列処理が必須である．しかし，手動で並列処理のためのコードを作成することは困難である．また，ソースコードを用いる並列化手法ではソースコードを参照できない場合に並列化を行うことができない．さらに，並列化においてはデータの依存関係の把握が必要になり，それにはプログラムの実行時の情報が重要となる．そこで，我々は動的なバイナリレベル並列処理を実現するシステムを開発している．本稿では，開発中の自動並列処理システムに必要な機能のうちスレッド実行制御の実現方法について述べる．

2 バイナリレベル並列処理システム

本研究で開発するシステムの要件は次に示すものである．まず，本システムはマルチコアプロセッサ上でプログラムを並列実行することで，プロセッサの性能を効果的に活用することを目的としているので実機上で実行することが前提である．

次に，実行時の情報を扱いバイナリレベルで並列化を行うために動的バイナリ変換機能が必要である．実機上で実行することでプロセッサによって対応する命令セットが固定されるので，任意のマルチコアプロセッサ上で実行するために複数の命令セットに対応したバイナリ変換を行えることが必要である．

そして，並列化する対象はループである．一般的に，プログラム中のループがプログラムの実行時間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．

以上を満たすシステム上でシングルスレッドプログラムを実行すれば，そのプログラムを並列実行した結果を得ることができる．

3 動的計測ツール Valgrind とシステムの概要

本システムを実現するためには，複数の命令セットに対応した動的バイナリ変換を実機上で行う必要がある．そこで，動的バイナリ変換によって対象のプログラムの命令の操作を行うフレームワークである Valgrind[1] を利用した．

Valgrind は図 1 のように構成され，Guest Application，User Plug-in Tool，Valgrind core の 3 つのプログラムが同じプロセス上で実行される．Guest Application は Valgrind core に与えるプログラムである．Valgrind core 内ではバイナリ変換を行い，得られた変換コードを Code Cache に格納する．Dispatcher は Valgrind core と Code Cache 内のコード間の制御の受け渡しを行う．User Plug-in Tool は中間表現コードに対して操作を行うものであり，ユーザが自由に作成し処理を追加することができる．

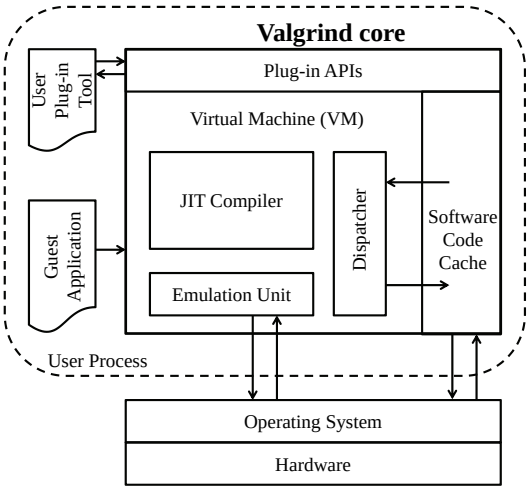

図 1: Valgrind システムの全体構成

バイナリ変換の手順は，まず，元のプログラムのマシンコードをプロセッサ独立の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命令の操作を行う．最後に，操作が加えられた中間表現コードをマシンコードに変換する．これらの処理を基本ブロック単位で行い，変換コードを実行することを繰り返す．また，変換済みの基本ブロックを再び実行するときは Code Cache 内の変換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．

本研究の自動並列処理システムでは，並列化対象であるループに到達したらその部分の基本ブロックを並列化して並列実行する．このとき，並列化コードの実行は新たに生成したスレッドのみで行う．並列化対象以外についてはシングルスレッド実行する．この処理をプログラム終了まで繰り返す．ここで，システムを初めから実行しているスレッドを親スレッド，新たに生成したスレッドを子スレッドとする．これを実現するためには，スレッド生成，スレッド実行制御，並列化コード生成が必要である．現在は Linux OS 上でシステムを開発しているが，将来的には任意のプラットフォームで動作することを目指している．

Thread Control in Automated Parallel Processing System using Valgrind
†Takayuki Hoshi, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota and Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

4　スレッド実行制御

スレッド生成には Linux の clone システムコールを用いているが，システムコールはコストが大きい．並列化コードの実行は子スレッドのみで行うので，並列化対象に到達する度にスレッド生成をしていてはオーバーヘッドが大きくなる．そこで，スレッド生成コストを削減するために一度生成したスレッドを再利用するスレッドプールを導入した．

子スレッドは並列化対象に到達したときに並列化コードを実行するだけなので，それ以外のときは待機する必要がある．子スレッドを制御するのは親スレッドになるので，全てのスレッドからアクセスできる共有変数を子スレッドごとに用意して子スレッドを制御する．

スレッド実行制御では次のことを実現する必要がある．一つ目に，スレッドプールにある子スレッドを取り出すことである．二つ目に，その子スレッドで並列化した変換コードを実行させることである．三つ目に，並列実行は子スレッドだけで行うので，その間親スレッドの処理を待機させることである．四つ目に，子スレッドの計算結果を親スレッドに書き戻すことである．

スレッドプールでは共有変数の値で子スレッドを制御している．そのため，親スレッドが子スレッドを取り出すためには共有変数の値を変更すればよい．また，値の変更は並列化した変換コードが存在することを確認してから行う．

子スレッドで並列化した変換コードを実行するためには，その変換コードのアドレスが必要になる．そこで，Core と変換コード間の制御の受け渡しを行う Dispatcher に着目した．Dispatcher は現在対象にしている基本ブロックの変換コードが Code Cache に存在するかどうかを確認し，存在する場合はその変換コードのアドレスにジャンプする．そのため，子スレッドが Dispatcher を実行すれば並列化した変換コードを実行することができる．

親スレッドを待機状態にするためには，複数存在する子スレッドの実行が全て終了するまで親スレッドの処理を進めないようにすればよい．そこで，子スレッドを制御するために用意した共有変数の値を利用する．親スレッドは共有変数の値を監視し，その値が子スレッドの実行終了を表すまでは何もせずに待機する．全ての子スレッドについて共有変数の値が実行終了を表していたら親スレッドの処理を再開する．

並列化した場合，子スレッドはループの一部分だけを実行することになるのでループ全体の結果を得るためには子スレッド全ての結果を親スレッドに集める必要がある．このとき，複数の子スレッドが親スレッドに書き込みを行うので排他制御をする．

以上のことから，スレッド実行制御を Dispatcher に実装する．Dispatcher は親スレッドと子スレッドの両方が実行することになるが，親スレッドと子スレッドで処理が異なるので，Dispatcher の中で親スレッドと子スレッドを区別する．

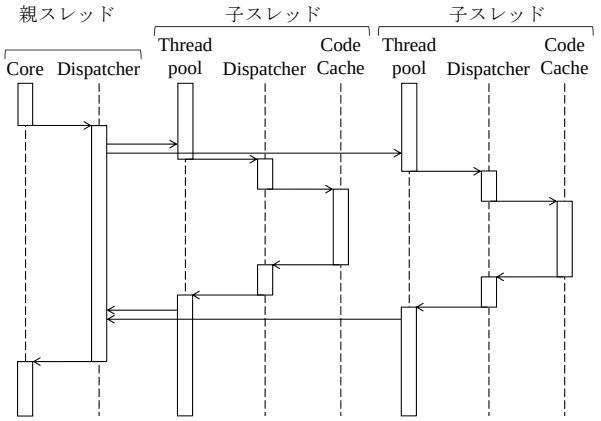

図 2: スレッド実行制御の流れ

スレッド実行制御の流れは図 2 のようになる．この図では，子スレッドを 2 つ生成して並列実行した場合を表している．また，並列化コードは Code Cache 上に存在しているとする．スレッドプールを利用しているので子スレッドでは Thread pool が既にアクティブの状態になっている．まず，親スレッドの制御が Core から Dispatcher に移る．ここで，並列化対象の基本ブロックの変換コードが存在するか確認する．並列化した変換コードが存在することを確認したら，子スレッドを制御する共有変数の値を変更してスレッドプールの子スレッドを取り出す．すると，子スレッドの制御はスレッドプールから Dispatcher に移り，Code Cahce にある並列化された変換コードを実行する．この間，子スレッドを制御している共有変数の値は実行終了を表していないので，親スレッドは待機状態になる．子スレッドの実行が終了すると制御はスレッドプールに戻り，子スレッドを制御している共有変数の値を実行終了にする．親スレッドは全ての子スレッドの共有変数の値が実行終了を表していることを確認したら次の処理に進む．

5　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必要な機能のうちスレッドの実行制御について述べた．

今後の課題として，並列化コードの自動生成処理を開発することが挙げられる．

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)24500055，同 (C)24500054) 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.