4J-9

Valgrindベース自動並列処理系におけるスレッド実行制御の最適化

星 孝幸 † 小渕 裕之 †† 大津 金光 † 大川 猛 † 横田 隆史 †
† 宇都宮大学大学院工学研究科情報システム科学専攻 †† 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有効に活用するためには，スレッドレベルの並列処理が必要である．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は並列化することができない．また，並列化ではデータの依存関係の把握が重要で，それにはプログラムの実行時の情報が必要である．そこで，いつでも利用可能であるバイナリコードに着目し，動的バイナリ変換によって自動並列処理を実現するシステムを開発している．

動的バイナリ変換には実行時のオーバヘッドが発生する．このオーバヘッドはプログラムの性能向上を妨げる原因となっている．開発しているシステム内の冗長に行われる検索処理を削減する方法を開発したが，性能向上には不十分であった [1]．そこで，さらに実行時間を短縮するためにスレッド実行制御のオーバヘッドを削減する方法を検討する．

2 Valgrindベース自動並列処理系

本研究で開発するシステムはバイナリコードレベルでプログラム並列化するためにバイナリ変換技術が必要である．バイナリコードはプラットフォームごとに異なるので，様々なプラットフォームで開発されたプログラムを並列化するためにはプラットフォームごとに並列化システムを開発しなければならない．そこで，動的バイナリ変換フレームワークの Valgrind[2] に着目した．Valgrind は複数のプラットフォームでの動的バイナリ変換を行えるので，Valgrind に並列化の機能を追加すれば複数のプラットフォームに対応する自動並列処理システムを実現することができる．

Valgrind は図1のような構成をしている．Guest Application, User Plug-in Tool, Valgrind core が同じプロセス上で実行される．Guest Application は Valgrind に対して入力として与えられ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 Software Code Cache に格納する．Dispatcher は Valgrind core と Software 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い，ユーザが自由に作成し処理を追加することができる．

システムの基盤として利用した Valgrind はマルチスレッド実行に対応していないので，新たにスレッドを生成し制御する機能と，バイナリコードレベルで並列化する機能が必要である．スレッド実行制御では，スレッドの生成・終了，処理の割り当てを行う．

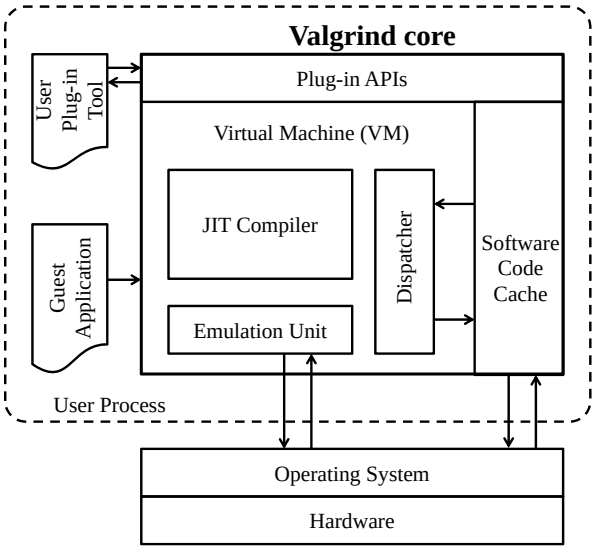

図 1: Valgrind の全体構成

3 従来のスレッド実行制御の問題点

スレッド生成にシステムコールを用いるが，システムコールはコストが大きいので並列化対象に到達するたびにスレッド生成していてはオーバ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ている．スレッド実行制御では，各スレッドに共有変数を用意しその値を変更することでスレッドプールの複数のスレッドを制御する．

並列化対象のループは新たに生成した子スレッドだけで行う．まず，システムの開始後にあらかじめ設定した数の子スレッドを生成する．子スレッドは並列化対象に到達するまで親スレッドからの指示をスレッドプール内で待ち続ける．親スレッドは並列化対象に到達するまで通常の実行をする．並列化対象コードに到達したら，親スレッドはその基本ブロックを並列化し，子スレッドに対して並列実行の開始を指示する．また，親スレッドは子スレッドの並列実行が終了するまで待機する．子スレッドは並列実行終了時に並列実行が終了したことを表す値を自身の共有変数にセットする．親スレッドは全ての子スレッドが終了したことを確認して，次の基本ブロックの処理に移る．

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

共有変数の確認にはスピンウェイトを用いており，短時間の待機であれば無駄が少ないが，長い時間の待機では無駄が多くなり他のスレッドの処理を妨げる．親スレッドが子スレッドの実行終了を待つのは長時間にはならないが，子スレッドが親スレッドの並列実行開始の指示を待つのは長時間になる可能性がある．そのため，子スレッドをスピンウェイトで待機させるのは問題がある．

また，並列実行に必要なゲストプログラムのレジスタ情報を親スレッドから子スレッドにコピーする処理を並列実行開始の指示の後に行っている．そのため，レジスタ情報のコピーがオーバヘッドになっている．そこで，スレッドプール内の処理を効率的に行う方法を検討した．

4　スレッド実行制御の最適化

スピンウェイトで長時間待機することを避けるために，長時間の待機では sleep を使用する．しかし，sleep を使用すると同期変数が頻繁にチェックされなくなる．そこで，長時間の待機には sleep を使用し，短時間の待機には sleep を使用しないように切り替える方法を検討する．

まず，子スレッドは sleep を使用して待機する．親スレッドは並列化対象の基本ブロックに到達したら，並列実行開始を指示する前にスレッドプール内の子スレッドを sleep を使用しない待機方式に切り替える．また，子スレッドでの並列実行が終了したら，sleep を使用して待機する方式に切り替え，次の並列化対象の基本ブロックに到達するまで待機する．これは，待機方式を指定する共有変数を用意し，それを子スレッドが確認することで実現できる．

並列実行開始の指示の後にゲストプログラムのレジスタ情報をコピーする処理を，ゲストプログラムのレジスタ情報が確定した段階であらかじめ子スレッドにコピーしておけば，並列実行開始の指示の後にレジスタ情報のコピーをする必要はない．そこで，現在対象にしている基本ブロックについてのレジスタ情報が確定した段階で，親スレッドが子スレッドに対してレジスタ情報のコピーを指示する．こうすることで，親スレッドの処理の最中に子スレッドでレジスタ情報のコピーをすることになるので，レジスタ情報のコピーのオーバヘッドを隠蔽することができる．

レジスタ情報のコピーをあらかじめ行うことを実現するためには，スピンウェイト中に並列実行開始の指示だけでなく，レジスタ情報のコピーの指示も確認する．子スレッドがレジスタ情報のコピーの指示を確認したら，親スレッドが保持しているゲストプログラムのレジスタ情報を子スレッドごとに用意したレジスタ情報を格納する変数にコピーする．そして，並列実行開始の指示を待つ．

図 2(a) のように，親スレッドが並列実行の開始を指示した後にレジスタ情報のコピーを行うと，子スレッドの並列実行時の処理における並列化コードの実行の割合が相対的に小さくなり，効率的ではない．図 2(b) のように，並列実行開始を指示する前にレジスタ情報のコピーを行うことで，子スレッドを効率的に利用することができる．

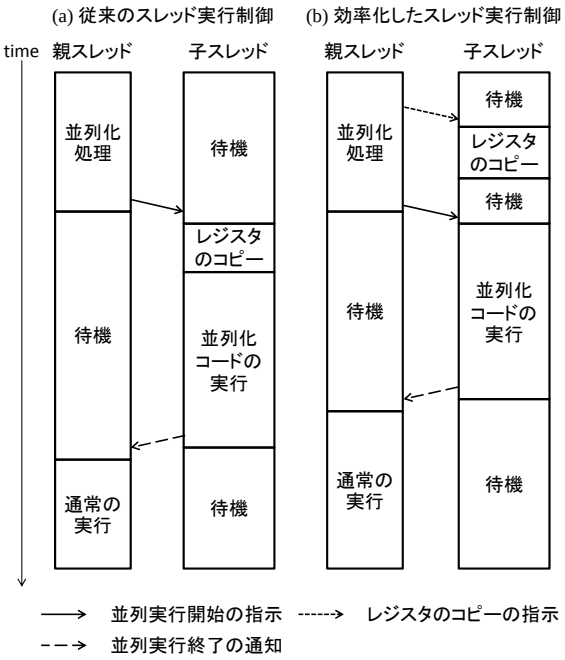

図 2: スレッド実行制御の効率化

5　おわりに

本稿では，開発しているシステムのオーバヘッドを削減し，最大限の性能向上を実現するためにスレッド実行制御におけるオーバヘッド削減方法について述べた．

今後の課題として，親スレッドが現在変換している基本ブロックの分岐先の基本ブロックの変換を子スレッドで行うことで，子スレッドの待機している時間を減らし，より効率的に処理を行う方法が挙げられる．

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，同 (C)25330055，若手研究 (B)25730026) の援助による．

参考文献

[1] Takayuki Hoshi, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota, “Runtime Overhead Reduction in Automated Parallel Processing System using Valgrind”, Proc. 1st International Symposium on Computing and Networking ― Across Practical Development and Theoretical Research ― (CANDAR’13), pp.572-576, 2013.

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