時間的集約を行うOSスケジューラによる関数型言語Erlangのスループット向上の試み

山田 賢[†] 日下部 茂[††]

1. はじめに

関数型言語は暗黙の並列性を持っており，様々なレベルの並列性が抽出しやすい．例えばスレッドレベルの並列性を抽出すれば，SMT や CMP などのプロセッサ上で並列実行によりスループットの向上が期待できる．しかし実際のアプリケーションの実行には OS のサポートが必要であり，上記のような実行においても，OS はプロセッサ上でメモリ参照やコンテキストスイッチといったオーバーヘッドを減らすようなタスクスケジューリングを行う必要がある．

我々は OS のスレッドのスケジューリング方式を改良し，同じメモリアドレス空間を共有するスレッド同士を集約するスケジューリングを適用することを提案開発している．実際に SMP 上での実行をサポートしている関数型言語 Erlang を用いて性能を評価したところ，最大で約 11%のスループット向上を確認した．

2. スレッド集約スケジューラ

Linux ではバージョン 2.5 から $O(1)$ スケジューラ[1] が導入された．$O(1)$ スケジューラは多数のスレッドの中から定数オーダーで実行するスレッドを選択でき，スケジューリングのオーバーヘッドを削減できる．しかし，$O(1)$ スケジューラではスレッド間のアドレス空間の共有を考慮したスケジューリングを行っておらず，充分にメモリ階層を活用した実行が難しい．そこで我々は同一アドレス空間を考慮したスケジューリング手法[2] を適用する．この手法では，同一アドレス空間を共有するスレッドを集約して実行することで，参照の局所性を活用するとともに，アドレス空間の切り替えに伴うオーバーヘッドを削減する．

† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†† 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

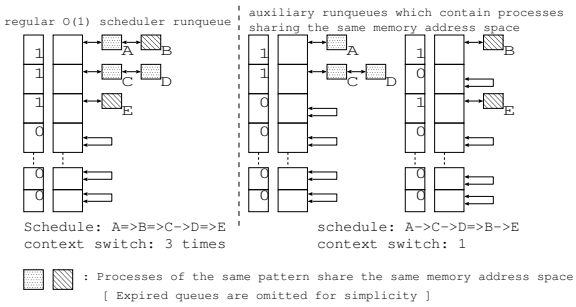

図 1 アドレス空間を共有するスレッドの集約

図 1 を用いて具体例を示す．図 1 は通常の $O(1)$ スケジューラのランキュー (図中左) と我々の提案するスケジューラのランキュー (図中右) を示している．この図でスレッド A，C，D が同じアドレス空間を共有し，スレッド B，E は別のアドレス空間を共有するスレッドを示している．通常の $O(1)$ スケジューラでは優先度の高い順にスレッドが選択されて実行される．図 1 の例では A，B，C，D，E の順で実行され，アドレス空間が異なるコンテキストスイッチは 3 回実行される．一方，我々の提案するスケジューラではアドレス空間ごとにランキューを用意する．そしてアドレス空間を共有するスレッドに対しては優先度を 1 上げるようなスケジュールを行う．図 1 の例では A，C，D，B，E の順で実行され，アドレス空間が異なるコンテキストスイッチは 1 回に削減される．

このように，アドレス空間を共有するスレッドを集約するようなスケジューリングを行うことで，コンテキストスイッチのオーバーヘッド削減や，参照の局所性向上といった効果が得られると考える．本論文では次節に示すように関数型言語で書かれたアプリケーションをマルチスレッド実行する際のスループット向上に着目して議論する．

3. Erlang での評価

我々は Erlang の *big*[3] ベンチマークを用いて我々のスケジューラの評価を行った．関数型言語 Erlang

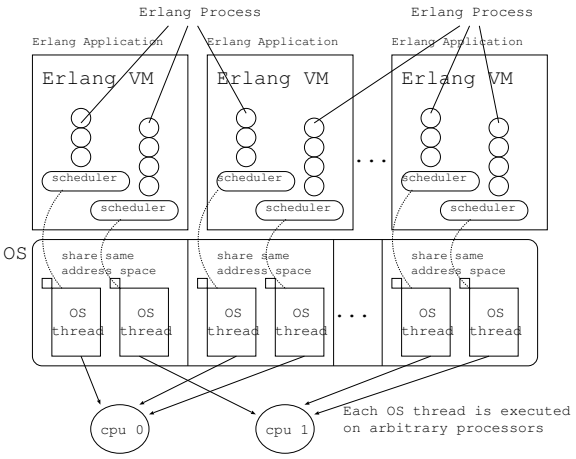

図 2 Erlang を実行する際の VM，スケジューラ，OS スレッドとの関係

は Erlang 用の VM 上で実行される．現在 Erlang の VM は SMP 上での実行をサポートしており，複数のスケジューラを用いることで複数の CPU を活用した実行が可能である．この Erlang VM のスケジューラは OS スレッドと一対一に対応しており，複数のスケジューラを用いることで OS レベルのマルチスレッド実行を行うことが出来る[4]．図 2 はデュアルコアプロセッサ上における以上の関係を図示したものである．我々はこのマルチスレッド実行にアドレス空間を考慮したスケジューリングを適用する．

同一の Erlang の VM から生成される Erlang スケジューラ (OS スレッド) は，すべてアドレス空間を共有する複数の Erlang プロセスをマルチタスク実行している．今回我々は複数の VM を用意し，各 VM 上でそれぞれ 2 つの Erlang スケジューラを生成した環境で評価を行った．例えば VM 数が 4 の状況を想定した場合，OS スレッド (Erlang スケジューラ) 数が 8 のマルチスレッド実行が行われる．評価アプリケーションには *big* ベンチマークを用いた．*big* ベンチマークでは Erlang のプロセスを複数生成し，それぞれのプロセスがメッセージを送受信を行う際に要する時間を計測する．今回我々は 1000 の Erlang プロセスを生成するよう指定した．VM の数は 1 から始めて今回は 7 まで計測を行った．測定環境は表 1 のようになっている．

ベンチマークの結果を表 2 に示す．結果はすべての VM での経過時間の平均である．表 2 より VM 数が 4 から 7 にかけて我々のスケジューラを用いた際のスループットの向上が確認できた．特に VM 数が 7 の時に約 11%の向上率を確認でき，全体的な傾向としてシステム内の多重度が大きいほど有効であったと言える．VM の数が 2，3 のときも若干向上したが，誤差の範囲をでないため記していない．我々はこの結果をアドレス空間の異なるコンテキストスイッチ数の削減や局所性の向上によるものと考える．

表 1 測定環境

Processor	Intel Core Duo 1.66GHz
OS	Fedora Core 5
L1 data cache size	32KB
L2 cache size	512MB
Memory Size	1GB

表 2 Erlang での評価結果 (単位:秒)

VM の数	*O(1)*	集約スケジューラ	向上率 (%)
4	94	87.3	7.62
5	114.3	109.0	4.88
6	140.2	128.4	9.24
7	164.4	147.9	11.1

4. ま と め

本論文ではアドレス空間を共有した OS スレッドの集約が，関数型言語で書かれたアプリケーションの実行する際にも有効であることを示した．Erlang の *big* ベンチマークを用いて測定したところ，最大で約 11%のスループット向上を確認した．今後の課題としては更なるアプリケーションで，より一般的な状況での有効性の確認がある．また他のプロセッサ上での有効性の確認などがある．さらに今回は全体的なスループット時間について述べたが，ターンアラウンド時間との関係についても考察していく．

参 考 文 献

1) Ingo Molnar; *Ultra-scalable O(1) SMP and UP Scheduler*, http://www.uwsg.iu.edu/hypermail/linux/kernel/0201.0/0810.html, (2002)
2) Takuya Kondoh and Shigeru Kusakabe: *Enhancing Throughput of Threaded Servers on Off-the-shelf Linux Platforms*, The 3rd IASTED International Conference on Communications, Internet, and Information Technology (CIIT), (2004)
3) Rickard Green; *Message passing benchmark on smp emulartor*, http://www.erlang.org/ml-archive/erlang-questions/200603/msg00111.html, (2005)
4) *Erlang goes multi-core*, http://www.ericsson.com/technology/opensource/erlang/news/archive/erlang_goes_multi_core.shtml, (2006)