2N-1

軽量プロセス・ライブラリ Lesser Bear: ロックを必要としない並列スケジューリング機構の提案

小熊 寿　中山 泰一
電気通信大学 情報工学科

1 はじめに

並列計算機は近年、より身近なものとなってきた。筆者らは現在、SMP 型計算機を対象とし、並列性と移植性に優れたスレッド・ライブラリ Lesser Bear の設計・実現を行っている。

Lesser Bear は、仮想プロセッサとして生成した複数の UNIX プロセスと巨大な共有メモリ空間によって、スレッドの並列動作を実現している。また、先述の共有メモリ空間を各仮想プロセッサごとに分割し、スケジューリングの並列動作を行っている [3]。

すべての仮想プロセッサが任意のスレッドを扱うためには、仮想プロセッサ間のロックが必要となる。しかしロック操作は、仮想プロセッサ間の逐次化を起こし、SMP 型計算機の特徴である並列動作を妨げてしまう [1]。

本稿では Lesser Bear 内部で、一切のロックを利用せずにスケジューリングを実現する機構を提案する。提案する機構では、分割した共有メモリ空間とスレッドを保存するキューを利用して、循環状のトポロジを形成する。

提案したスケジューリング機構を SMP 型計算機上で性能評価実験を行った結果、ロック操作削除の効果が確認された。

2 Lesser Bear の概要

Lesser Bear はスレッドを並列に実行するために、複数の UNIX プロセスを仮想プロセッサとして生成する。スレッドの並列実行が可能なライブラリである LinuxThreads [2] や PPL [4] でも同様に、複数の仮想プロセッサを生成している。スレッド・コンテクストをすべての仮想プロセッサで共有するために、Lesser Bear は初期化時に巨大な共有メモリ空間を生成し、すべてのコンテクストを保存する。

また Lesser Bear はスケジューリングを並列に行うために、先述の共有メモリ空間を仮想的に分割し、各仮想プロセッサがそれぞれの空間を管理する。各空間には Protect Queue と Waiver Queue が存在し、仮想プロセッサはこの 2 本のキューを利用してスレッドを管理する。 Protect Queue は所有者の仮想プロセッサのみがアクセスするため、仮想プロセッサ間のロックを必要としない。 Waiver Queue は所有者からの挿入のみ受け付ける。実行可能なスレッドを持たない仮想プロセッサは、他の仮想プロセッサが所有する Waiver Queue からスレッドを獲得する。そのため Waiver Queue からのスレッド削除には、ロックを利用する。

Lesser Bear は 1 つのキューを 2 つの異なる仮想プロセッサで共有し、一方が挿入、もう一方が削除のみを行うときにロックを必要としないアルゴリズムを利用している [5]。これにより従来の Lesser Bear は、キューへの挿入の時に一切のロックを必要としない。

しかしキュー操作では、挿入の回数だけ削除は起こる。その結果、削除時のロックにより、仮想プロセッサの逐次化が起こり、 SMP 型計算機の特徴である並列動作を妨げる可能性がある。

3 循環型スケジューリング

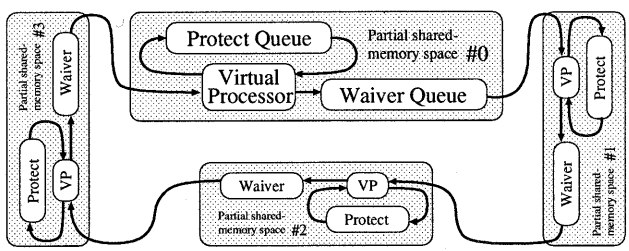

図 1: 循環型スケジューリング機構のモデル

本稿では、従来の Lesser Bear がもつ Protect Queue と Waiver Queue の特徴を利用した循環型スケジューリング機構を提案する。図 1 は、仮想プロセッサ数を 4 とした場合のモデルである。

Lesser Bear: a Lightweight Process Library for SMP Computers – Scheduling Mechanism without a Lock Operation

Hisashi OGUMA and Yasuichi NAKAYAMA
Department of Computer Science, The University of Electro-Communications
E-mail: oguma-h@igo.cs.uec.ac.jp

まず従来の Lesser Bear と同様に、巨大な共有メモリ空間を仮想プロセッサごとの空間に分割し、空間ごとに Protect Queue と Waiver Queue を用意する。これら分割した空間から新たに、Waiver Queue を介して循環状のトポロジを形成する。仮想プロセッサ i はキューに対し、以下のアルゴリズムに従ってスレッドの挿入・削除を行う ($0 \leq i \leq n-1$ | n は仮想プロセッサ数)。

- 仮想プロセッサ i は基本的に、処理を中断したスレッドを Protect Queue に挿入し、次に処理すべきスレッドを Protect Queue から取り出す。
- Protect Queue に登録しきれないスレッドは、Waiver Queue に挿入される。
- コンテクストの切替えごとに仮想プロセッサ i は、仮想プロセッサ $\{(i+n-1) \bmod n\}$ の Waiver Queue に存在するすべてのスレッドを取り出し、 Protect Queue に挿入する。この時、挿入しきれないスレッドは、自身の Waiver Queue に挿入する。

提案する機構は 1 つの Waiver Queue を 2 つの異なる仮想プロセッサで共有する。しかし Waiver Queue に対する操作は、先述のアルゴリズムを利用するためロックを必要としない。そのため、キューへの挿入・削除に限らず、仮想プロセッサ間をスレッドが移動する場合であってもロックをしない。スケジューリングは、これら一連の操作を利用して行う。

スケジューラは一般に、アプリケーションが生成するスレッドを、各仮想プロセッサに分配する必要がある。これは、ある仮想プロセッサがスレッドを独占した場合、残りの仮想プロセッサはアイドルとなり、計算機の CPU 利用率を下げてしまうためである。従来の Lesser Bear は仮想プロセッサ間のスレッド移動にロックを伴うため、移動は仮想プロセッサがアイドルになった場合にのみと制限していた。しかし、本稿で提案するスケジューリング機構はロックを必要としないため、スレッド分配のための移動を頻繁に行ってもアプリケーションに与える影響は小さい。 これにより Lesser Bear は、仮想プロセッサごとのスレッド数を均一にし、仮想プロセッサがアイドルになりにくくなる結果、 CPU 利用率の向上が期待できる。

4 評価

提案した循環型スケジューリング機構を Lesser Bear 上で実現し、従来の Lesser Bear と比較を行った。実験では、スレッドの分配作業に要する時間を測定した。計算機には、8 つの CPU (40MHz) とメモリを 640M バイトもつ SPARC Server 1000、OS に SMP に対応した SunOS 5.5.1 を利用した。

表 1: 均一に割り当てられるまでの時間 (ミリ秒)

	従来の設計	提案した設計
時間	131.6	30.8

表 1 は、 一度に 256 個のスレッドを生成し、均等に各仮想プロセッサへ分配されるまでの時間を示している。従来の機構は、スレッドが仮想プロセッサ間を移動するときにロックが必要であった。表 1 は、ロック削除による効果を示している。

5 おわりに

本稿では、一切のロックを利用せずにスケジューリングを実現する機構の設計を行った。

この機構を Lesser Bear 上で実現し性能評価実験を行った結果、各仮想プロセッサへのスレッド分配をより小さなコストでできることを確認した。

今後はより詳細な実験と、実際のアプリケーションによる評価を行う予定である。

参考文献

[1] A. Kaieda *et al.*: Analysis and Measurement of the Effect of Kernel Locks in SMP Systems, *Concurrency: Practice and Experience*, Vol. 13, No. 1, pp. 1–12 (2001).

[2] X. Leroy: Linuxthreads – POSIX 1003.1c kernel threads for Linux, `http://pauillac.inria.fr/~xleroy/linuxthreads`.

[3] H. Oguma and Y. Nakayama: Lesser Bear – a Light-weight Process Library for SMP computers –, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 2451–2457 (2000).

[4] 坂本ほか: 並列性と移植性をもつユーザレベルスレッドライブラリー PPL の設計および実装, 電子情報通信学会論文誌, Vol. J80–D–I, No. 1, pp. 42–49 (1997).

[5] M. Suzuki and T. Watanabe: A Lightweight Solution for the Producer-consumer Problem, *Report CS 00-05, Department of Computer Science, University of Electro-Communications* (2000).