4K−1

適応型ロック機構を用いた並行プロセスのコスケジューリングについて

三栄 武　　高橋 直久
NTT ソフトウェア研究所

1 はじめに

メモリ共有型マルチプロセッサ環境では、高並列処理のため小粒度のプロセスを多数生成して実行させる。本稿では、並行プロセスを効率良く実行させるために、各プロセスの状態や競合関係に従って実行時にプロセスをグループ化し、資源の割り当て制御を行う動的コスケジューリング(co-scheduling)を提案し、その実現法について述べる。

2 背景

2.1 排他制御

資源へのアクセスを正しく制御するために排他制御機構が使用される。排他制御機構は、ロックに失敗したプロセス(待機プロセスと呼ぶ)がCPU使用権を放棄してブロック状態で待機するブロック型と、待機プロセスがCPU使用権を確保したまま繰り返しロックを試みるビジーウエイト型に分類される[1]。ブロック型ではロックに失敗すると必ずプロセス切り換えが発生するため、小粒度プロセス等で同期頻度が高い場合には排他制御のオーバーヘッドが大きな問題となる。ビジーウエイト型では、ロックに成功したプロセス(ロックプロセスと呼ぶ)がCPUのプリエンプションや事象待ち等で実行を停止した場合、待機プロセスは割当られた資源(CPUやメモリ等)を浪費し続けてしまう問題を生じる。

2.2 コスケジューリング

マルチプロセッサ環境でのスケジューリング法として、複数プロセスをグループ化してCPUやメモリの割当/解放単位とするコスケジューリングが提案されている。その実現法として、1つのアプリケーションを構成するプロセス群をグループ化するtask force[2]や、ユーザーが指定したプロセス群をグループ化するgangスケジュールが提案されている。またMach[3]では、タスクを単位にアドレス空間を設定するので、同一タスクに属するスレッドにまとめてメモリを割り当て/解放することになる。

これらの実現法では、プロセスが実行前にグループ化されている(ここでは静的コスケジューリングと呼ぶ)ので、密結合型マルチプロセッサでの、CPU割当等のきめ細かい資源割り当て制御が要求される場合には適さない。このためスワップアウトの制御や分散メモリ型マルチプロセッサでのプロセッサ割り付け等のように比較的割当/解放の間隔が長い場合に用いられる。

Co-scheduling Concurrent Processes with an Adaptive Lock.
Takeshi MIEI (take@lucifer.ntt.jp),
Naohisa TAKAHASHI (naohisa@lucifer.ntt.jp),
NTT Software Laboratories.

3 動的コスケジューリング

頻繁に同期・通信を行うプロセスの実行では、資源の競合関係にあるプロセス群は短時間で変化する。本稿ではこれらのプロセス群を実行時に検出しグループ化することによりコスケジューリングを行う(動的コスケジューリングと呼ぶ)手法を提案する。ここでは、プロセス間の競合関係を実行時に調べ、ロックプロセスの状態に応じて待機プロセスの状態を変化させる新たなロック機構(適応型ロック機構と呼ぶ)を用いて動的コスケジューリングを実現する。

3.1 適応型ロック機構の概要

従来のロック機構は資源の割り当て状態に応じて待機プロセスの動作を変化させている。すなわち、資源が空いているならばプロセスは実行を継続でき、塞がっているならば、空くまでプロセスは待機する。ユーザーは資源の使い方を考慮して、ビジーウエイト型とブロック型のロック機構を使い分けるが、密結合型マルチプロセッサで高効率性、高速性を実現するようにロック機構を使い分けるのは難しい場合がある。このため、使い分けを自動化することが望まれる。我々はこれまでに、密結合型マルチプロセッサにおいて、キャッシュにあるロック変数を参照するループを利用してロックプロセスの状態監視を行うwatch-lockと呼ぶ相互排除機構[4]を提案した。本稿ではこの機構をもとに、実行時にロックプロセスの状態に応じて待機方法を変化させる適応型ロック機構に発展させる。

適応型ロック機構ではロックプロセスの状態の変化から、ロックの解除に多くの時間を要すると判断できる場合に、待機プロセスの状態をロックプロセスと同様に変化させて、資源割り当て優先順位を下げて待機させる。具体的には次の制御を行う。ロックプロセスが実行中あるいは実行可能な状態ならば待機プロセスはビジーウエイト型で待機する。ビジーウエイト中にロックプロセスがプリエンプションされたならば、待機プロセスもプリエンプションする。またロックプロセスがスワップアウト状態やブロック状態ならば、ブロックして待機する。この結果、待機プロセスはロックプロセスと同じ資源割り当て状態となり、一種の動的コスケジューリングが実現されている。

3.2 適応型ロック機構の動作

適応型ロックでは、プロセスの状態を制御するために次の制御データを用いる。

- ロック変数lock: 現在ロックされているならばlock=1、空いているならばlock=0。

- ロック型変数 l_class: 現在のロック機構がビジーウエイト型ならば l_class= busy、ブロック型ならば l_class= block。
- ロックプロセス識別子 l_pid: ロックプロセスを表す識別子
- ロックプロセス状態 l_status: ロックプロセスの状態を表し、run(実行中)ready(実行可)block(事象待ち) swapout(スワップアウト中) のいずれかの値をとる。

以下にロックの入口および出口操作手順を述べる。

(1) ロック入口操作

a. ロックが空いている (lock = 0) 場合: ロックを行い (lock = 1)、ロックの型をビジーウエイト型とし (l_class = busy)、l_pid に自プロセスの識別子を記録し、ロック入口動作を終了する。

b. ロックが塞がっている (lock = 1) 場合: ロックプロセス識別子 (l_pid) を読み出す。ロックの型がブロック型 (l_class = block) の場合ブロックする。ロックの型がビジーウエイト型 (l_class = busy) の場合、ロックプロセスの状態 (l_status) を読み出し、その値に従い、次のいずれかの動作を行う。

- ロックプロセスが実行中 (l_status = run) ならば再びロック入口動作を最初から行う (ビジーウエイト)。
- ロックプロセスがプリエンプションされている (l_status = ready) ならば自ら CPU を解放する。再び CPU を割り当てられた時にロック入口動作を最初からやり直す。
- ロックプロセスが停止している (l_status = swapout or block) ならばロック型をブロック型に変え (l_class = block) ブロックする。

(2) ロック出口操作

ロック型を調べ、ビジーウエイト型 (l_class = busy) ならばロック解除のみを行う (lock=0)。ブロック型 (l_class = block) ならば待機プロセスをブロック状態から回復させ、ロックを解除する (lock = 0)。

3.3 実現上の課題

適応型ロック機構は次の機能で実現される。

(1) ロックプロセス識別子の記録および読みだし
(2) ロックプロセスの状態の記録および読みだし
(3) 割り当てられた CPU 資源の解放

(1)(2) は従来のビジーウエイト型ロック機構と比べ処理時間を増加させる要因となる。従ってこれらの機能に要する時間を短くすることが重要である。また並列性を損なわずに実現する必要がある。(3) の CPU の割当解放機能は通常カーネルスケジューラー内部で実現されている。このため排他制御を行うユーザープロセスとカーネルとの間に効率的なインターフェースを提供する必要がある。

4 Mach 上での実現

動的コスケジューリングの動作確認と有効性評価のため、メモリ共有型並列計算機 Sequent Symmetry の Mach[3] 上に適応型ロック機構を実現した (図 1)。Mach ではスレッド単位で CPU の割当スケジュールを行い、またタスク単位でメモリ等 CPU 以外の資源割当スケジュールを行っている。Mach は各スレッドの状態を表すスレッドテーブルをカーネル内部データとして持っているので、適応型ロック機構ではこれを用いる。スレッドは生成時に自分のスレッドテーブルを指すポインタを計算し、その値をスレッド識別子として使用する。待機スレッドはロックに失敗した時点でロックスレッドのテーブルを自分のアドレス空間内にマッピングする。以降は自アドレス空間内のスレッドテーブルを参照すれば良いので、高速にスレッドの状態監視ができる。待機スレッドは Mach のプリミティブである cthread_yield()[5] を用いて自分の CPU 使用権を解放する。

5 おわりに

動的に競合関係が変化する小粒度プロセスに適したコスケジューリングを提案し、その実現法について述べた。今後は、Mach 上で実現したコスケジューリング機構について定量的評価を行うとともに、例外処理への適用法について検討する予定である。

謝辞

本研究の機会を与えて下さった磯田部長に感謝します。日頃、御指導・御討論いただく尾内リーダーをはじめ分散型ソフトウェア研究グループの皆様に深謝します。

参考文献

[1] K.Hwang,et al.,"Computer Architecture and Parallel Processing",McGraw-Hill,1984.

[2] J.K.Ousterhout,"Scheduling techniques for concurrent sysmtems",3rd ICDCS,pp.22-30,1982.

[3] A.Tevanian,et al.,"Mach Threads and the Unix Kernel :The Battle for Control",Carnegie-Mellon Report,CMU-CS-87-149, 1987.

[4] 高橋直久," メモリ共有型マルチプロセッサにおける小粒度プロセスの相互排除機構 ", 第 39 回情報処理全国大会,1989.

[5] E.C.Cooper,R.P.Draves,"C Threads",Carnegie-Mellon Report CMU-CS-88-154,1988.

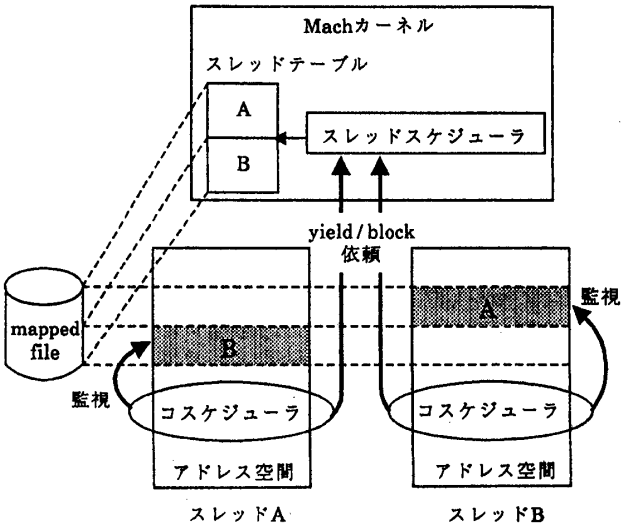

図1. Mach上での適応型ロック機構の構成