並列ジョブ効率的実行のための協調スケジューリング

2H－7

相場 雄一　青木 久幸
日本電気(株) C&C研究所

1　はじめに

並列計算機の普及に伴い、マルチスレッド機構を採用する並列オペレーティングシステムが発展してきた。このような環境で利用者が並列プログラムを動作させることが一般的となりつつあり、今後さらに様々な条件の下で並列プログラムが動作する状況が考えられる。そのような状況で並列処理の様々な利用形態を考えた場合、利用者の指示する並列処理が想定した通りに動作しない場合がある。これは、利用者の指定した並列プログラム内のスケジューリングとカーネルのスレッドスケジューリングに協調性がないために起こる。

本論文では、並列処理での問題点を解決するため、カーネル支援により実現される協調スケジューリングの方式について報告する。

2　並列処理スケジューリング方式の問題

利用者が並列プログラミングを行なう場合、並列性を抽出してタスクと呼ばれる一連の処理単位に分割する。一方、マルチスレッド機構を採用する並列OSでは、仕事をプロセッサへ割り当てる単位としてスレッドと呼ばれるカーネルの管理単位が設定される。

並列処理を行なう形態としていくつかの方式がある。最も簡単な方式はタスクをスレッドに固定的に割り付け、スケジューリングをカーネルの管理に任せる方式である。しかし、これではスレッド切替が重く、タスクの分割数にも限度がある。そこで、ライブラリ層のスケジューリングによりスレッド上でタスクを切替える方式が考えられた。この方式ならば、タスクを切替えるコストが軽く、スレッド数に関係なくタスク分割できる。

ところが、ライブラリ層でのタスクスケジューリングとカーネル内のスレッド管理という2階層のスケジューリング間に協調性がないために、ある並列度を期待して精巧に書かれた並列プログラムでも実際には利用者の意図した実行並列度が得られない場面があり得る。

図1の例は、1つのプログラムで4つのタスクを使って並列処理している様子を示す。この例では並列度2を想定し、2つのスレッドを生成している(1)。4つのタスクはライブラリ層の機能により2つのスレッド上でスケジューリングされる(2)。ここでタスクの1つがIOを要求すると、そのタスクのスレッドがカーネル内で

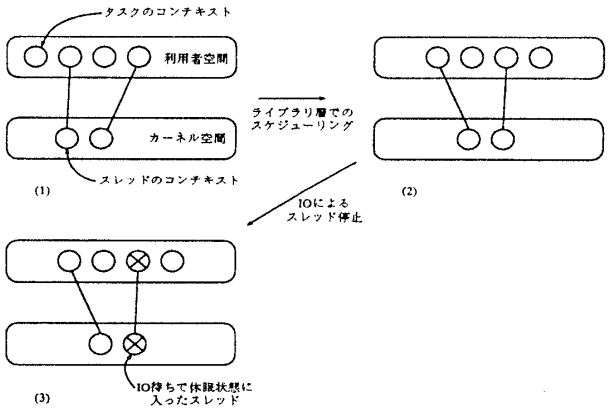

図1: 従来のスケジューリング方式

IO待ちを行ない、休眠状態となって停止する(3)。この状態では実行可能なスレッドの数が減少してしまい、他に実行させたいタスクが存在していても、実行させることができない。実質的に実行並列度が1に下がる。

3　協調スケジューリング方式

協調スケジューリング方式の目的は、並列プログラム実行時の並列度変動を抑え、期待した並列度を保持することであり、そのためにカーネルとライブラリ層で必要な情報を交換するための支援機構を導入し、協調性のあるスケジューリングを行なう。協調スケジューリング方式の基本的な考え方を以下に示す。

- 使用するタスク数に応じ、期待する並列度よりも多くのスレッドを生成し、余分なスレッドは動作しない状態(待機状態)で確保する。
- 実行並列度が減少しようとする際に待機状態のスレッドを動作可能にする。
- ライブラリ層でのタスクスイッチの時にその時の実行並列度を確認し、大き過ぎる場合スレッドを動作しない状態に戻す。

大まかな流れを図2に示す。図1の例と同様、並列度2を期待して書かれた並列プログラムが4つのタスクを使って処理されている様子を示す。この例では、スレッドを3つ生成するが、利用者の期待する並列度より多いので、2つを実行可能状態、1つを待機状態として確保する(1)。4つのタスクのスケジューリングは実行可能な2つのスレッドに対して利用者層で行なわれる(2)ので、この時の実行並列度は2となる。ここで、タスクの1つがIOを発行すると、そのタスクのスレッドがカーネル内でIO待ちを行ない、休眠状態に入り停止

Cooperative scheduling for effective processing of parallel job
Yuichi AIBA, Hisayuki AOKI
NEC Corporation

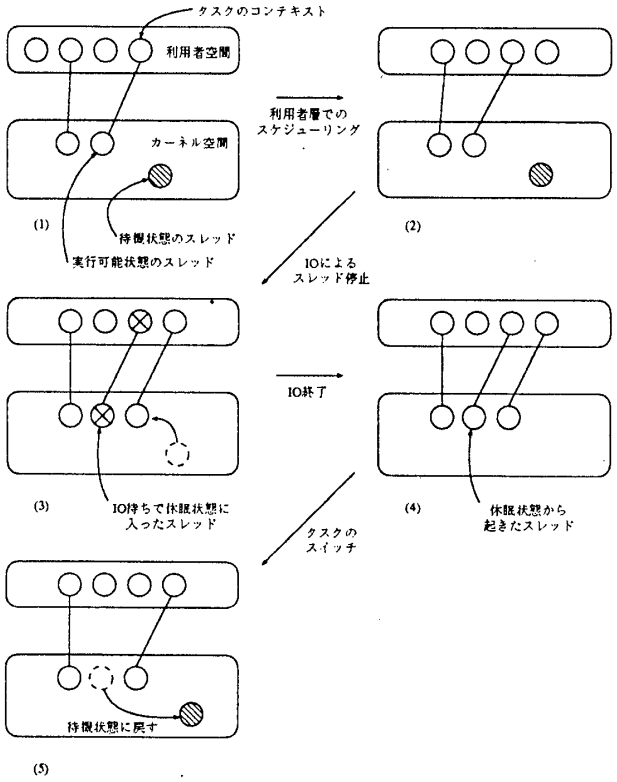

図 2: 協調スケジューリング方式

する。そのままでは実行並列度が減少するので、待機状態のスレッドを実行可能な状態に移し、他のタスクの実行のために代用する (3)。その後、2 つのスレッドに対して利用者層でスケジューリングが行なわれる。IO が終了し休眠状態のスレッドが起きると、一時的に実行可能なスレッド数が増え実行並列度が 3 に上がる (4)。しばらくはそのまま処理が進むが、利用者空間中でタスクのスイッチが行なわれる際、実行可能なスレッドの数が多いのを見て、そのスレッドを待機させることにより実行並列度が 2 に戻る (5)。以上により、期待した並列度を保った動作が可能となる。

4 実現方式

協調スケジューリング方式では、カーネルとライブラリ層で情報を共有することが必要となり、そのための支援機構を用意する。共有する情報を以下に示す。

1. 期待される並列度　　2. ある時点の実行並列度
3. 待機状態スレッドの有無

1. の情報は利用者が指定するので、従来はカーネルは関知しなかったが、図 2(3) のような場面で必要となる。カーネル内部でスレッドが休眠状態に移行する前に、期待される並列度と実行並列度を比較し、待機状態から実行可能にするか判断する。

図 2(1) に至るスレッド準備の段階では、1. ～ 3. 全ての情報が必要となる。プログラムのタスク分割時にカーネルにスレッドを要求するが、これらの値に応じてスレッドの準備方法は変化する。我々の実装では、ライブラリ層でスレッドを準備する方法を判別しているので、生成したスレッド数の情報も共有する必要がある。

また、図 2の (4) から (5) に移行する場面では、ライブラリ層で 1., 2. の情報が必要となる。利用者空間中の動作でタスクがスイッチする時、プログラムで期待される並列度とその時点の実行並列度を比較してスレッドを待機状態に戻すかどうか判断する。スレッドを待機状態に戻す時はシステムコールを発行することになる。

図 3に情報交換の様子をまとめる。

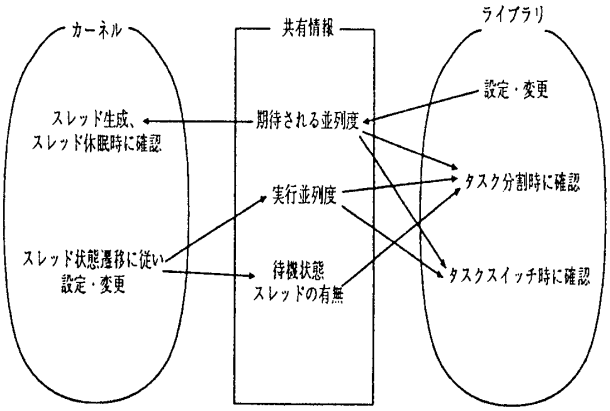

図 3: 情報の共有

いずれの情報も値の格納場所が問題となる。カーネル空間中とした場合、値を確認する都度システムコールを呼んでいては性能上望ましくない。そこで、多くのシステムでは利用者空間中に格納することになるが、プログラム実行の安全性を保証できない。我々の実装したシステムでは、利用者空間から見えるカーネル空間中の領域があるので、そこを利用し格納した。

5 まとめ

本稿では、並列ジョブのスケジューリング方式の問題点を取り上げ、この問題を解決する方式として協調スケジューリング方式を提示した。本協調スケジューリング方式は、NEC 製スーパーコンピュータ SX-3 上の OS である SUPER-UX に実装を行なった。今後評価を行なっていく。

参考文献

[1] Thomas E. Anderson 他, Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, *ACM Transactions on Computer Systems*, Feb. 1992.

[2] 「SUPER-UX FORTRAN77/SX 並列処理機能利用の手引」, NEC, SX システムソフトウェア GUF 25-1, 1992.