汎用超並列オペレーティングシステムカーネル SSS-CORE の基本構想

3T－6

古荘 進一　松本 尚　平木 敬

東京大学 大学院理学系研究科 情報科学専攻

1 はじめに

近年多くの超並列計算機が研究開発され実用化に向かっている。コストパフォーマンスにすぐれた超並列計算機システムは次世代のメインフレームとして有望であり、用途を限定せずに使用されるべきものである。しかしながら、現在のところ科学技術計算の分野に用途が限られているのが実情である。これは OS が超並列計算機に並列計算の高速性を活かしつつ汎用性を持たせることに失敗しているからである。

共有メモリ計算機において、従来の UMA (Uniform Memory Access) 型ではシステムの規模を大きくとることが出来ないので、NUMA (Non-Uniform Memory Access) 型であることが必然である。以前より我々は UMA 型共有メモリ計算機上で高性能と汎用性の両立を目指した OS 核 SS-CORE[1] の研究を行ってきた。本稿では、SS-CORE を NUMA 型共有メモリ計算機に対応して拡張した SSS-CORE (Scalable SS-CORE) の実現において考慮すべき問題点について述べる。

2 SSS-CORE

想定する計算機のモデルは図 1 のように、いくつかの PE (Processor Element) と CM (Cluster Memory) からなるクラスタが相互結合網により接続された NUMA 型共有メモリ計算機である。

本稿の SSS-CORE は NUMA 型共有メモリ超並列計算機の計算機資源を管理する OS 核である。SSS-CORE はマルチユーザ・マルチジョブ環境をシステムの性能を維持しつつ実現するものである。

SSS-CORE の基本は SS-CORE[1] と同じく、(1) ユーザレベルに危険でない限り資源管理について最大限の自由度を与えること、(2) 計算機資源についての情報を、ユーザが実行時最適化が出来るように適切なインタフェースにより公開すること、(3) 個々のユーザが最適化を行いつつ、システム全体としても最適化された資源管理を行うことである。

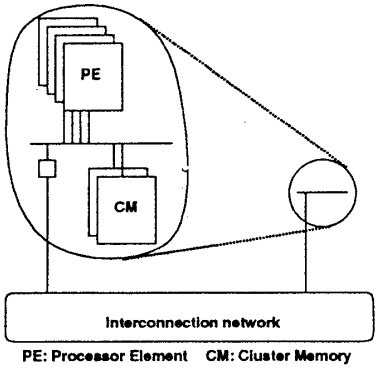

図 1: NUMA 型共有メモリ型計算機

SS-CORE では、効率良くプロセッサ資源のスケジューリングを行うために、スケジューリングはカーネルレベル及びユーザレベルのスケジューラにより二階層で行われる。カーネルレベルスケジューラはジョブの要求に応じて、要求通りの台数またはユーザが許すならそれ以下の台数の PE の集合を与える。ユーザレベルスケジューラは与えられた PE を自由に用いて、実プロセッサの割当状況を考慮しつつ、ユーザレベルにおいてジョブ内のスケジューリングを行う。

上に示すように SS-CORE においてプロセッサ資源のみに焦点を当ててスケジューリングを行っているのは UMA 型共有メモリ計算機においてメモリは全てのプロセッサに対し均一かつ等距離にあるからである。超並列計算機の必然である NUMA 型共有メモリ計算機では、プロセッサとメモリとの距離が一定ではないために、さらにメモリの割当状況を考慮してプロセッサおよびメモリのスケジューリングを行う必要があり、SS-CORE の資源管理方針はそのままでは NUMA 型共有メモリ計算機には適用できない。

本稿で述べる SSS-CORE の満たすべき基本的要請は以下の通りである。

1. 動的マルチユーザ・マルチジョブを効率良く実現すること。
2. ユーザレベルのプロセッサ及びメモリスケジューリングを提供すること。
3. 二次記憶階層を含んだ効率の良いメモリスケジューリングを提供すること。

SSS-CORE: Operating System Kernel for General Purpose Massively Parallel Machine

Shinichi FURUSO, Takashi MATSUMOTO, Kei HIRAKI

Department of Information Science, Faculty of Science, the University of Tokyo

{furuso,tm,hiraki}@is.s.u-tokyo.ac.jp

4. ユーザによるI/Oの最適化および高速化。

2.1 SSS-COREにおけるスケジューリングについて

SS-COREにおいては、SS-Wait[2]を用いて通信・同期を行う際に、プロセッサの割当状況を安価に知ることが出来るユーザレベルスケジューラが、同期がすみやかに成立するようにスケジューリングを行えた。対象がNUMA型共有メモリ計算機の場合も同様の方法が適用可能である。通信・同期の相手が同じクラスタにある場合には、SS-COREと同様の方法を用いることにより、スケジューリングを行える。通信・同期の相手が異なるクラスタにある場合、各クラスタのスケジューリング情報を共有メモリを介して安価に獲得できれば、ユーザスケジューラレベルで通信同期を最適化するようにスレッド管理が行える。このように複数のクラスタに跨っている場合もSS-COREと同様に飢餓状態のないプロセッサスケジューリングをSS-Waitで行うことが出来る。

さらに、NUMA型共有メモリ計算機では、他のクラスタメモリや二次記憶上にあるデータをアクセスすることが大きなコストとなる。通常これらのデータはクラスタ内にキャッシュされていたり、主記憶に割り当てられている。しかし、使用時にキャッシュミスが起こったり、二次記憶へのSwap Outが起こっていたりすると大きな遅延を生じプログラムの実行が滞ってしまう。データがキャッシュや主記憶にエントリやページを割り当てられていることを調べる手段があれば、SS-Waitを用いて遅延を避けることができる。

2.2 メモリ管理 — 特にジョブのパーティション割り付けについて —

SS-COREはUMA型共有メモリ計算機を対象としていたので、パーティション分割があまり大きな意味を持たなかった。しかしながら、共有メモリ超並列計算機の必然であるNUMA型共有メモリ計算機では、パーティション分割は重要な役割を占めている。

ここでジョブを、ジョブが要求するパーティションの形状に関して、次の通り分類する。

1. パーティションの形状を指定したもの。
2. パーティションの形状に何らかの制限を指定したもの。(例: クラスタ間の最大距離の指定)
3. パーティションの形状に制限を設けないもの。

それぞれの分類は(1)クラスタ間の通信が多いジョブのうち、数値計算のジョブのように必要とする通信のパターンが静的に読み切れるもの、(2)クラスタ間の通信が多いジョブのうち、通信のパターンが読み切れないもの、(3)クラスタ間の通信が少ないものに対応している。

ジョブをシステムにロードする時は、パーティションの形状についてなんら条件を持たないジョブについては空いたメモリにロードする。パーティションの形状に制限を設けたものは、条件を満たすパーティションにロードすることを原則とする。

ロードする際にシステムに空のクラスタが存在する場合には問題はないが、すでにジョブが存在するクラスタにロードする場合、元のジョブをクラスタのメモリから追い出す必要がある。この時、メモリから追い出されるジョブのパーティションの形状の制限などを考え、他のクラスタに移動するか、二次記憶に書き出すかを決定する。

特に大きなジョブの場合に問題となるのが、ジョブのロードにかかる時間である。このロードにかかる時間を隠蔽するために実行とロードとを同時に行う必要がある。効率良くロードする方法として考えられるのが、ジョブがプリエンプトされる時に、クラスタごとのWorking Setに関する情報をOSに与える。OSはジョブが必要とするメモリについては実メモリから追い出されにくくすることが出来る。ジョブを再開する時は、プリエンプトされた時にOSに与えた情報を元に、ジョブが必要とするメモリを優先して実メモリにロードすることが出来る。

ジョブは大きさやパーティションの形状に要求する条件により、システム全体の公平性のために優先度などを考える必要がある。例えば、パーティションに対して特に条件を設けないようなジョブに対しては、クラスタ間の移動が多くなる代わりにメモリから二次記憶に追い出されにくくすることが可能である。また、パーティションの形状に制限を設けたジョブをロードすることが出来なかった場合は、当該ジョブの優先度を高くするなどにより、ジョブがロードされやすくするようにできる。

3 おわりに

本稿ではNUMA型共有メモリ超並列計算機上に、性能を低下させずにマルチユーザ・マルチジョブのOS核を実現する時に考慮すべき問題点を指摘し、スケジューリング、パーティション分割について考察した。保護機構については今後の検討課題である。

参考文献

[1] 松本、根岸、渦原、森山:「粒度に基づいた並列計算の分類法とマルチプロセッサの資源管理法について」、日本ソフトウェア科学会第7回大会論文集(October 1990)。

[2] 松本 尚:「マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察」、情報処理学会 計算機アーキテクチャ研究会報告No. 79-1、pp. 1-8、(November 1989)。