

マルチマイクロプロセッサシステム向けの I/Oスケジュール方式の提案

6 G - 5

岡崎 耕太郎 木村 誠 秋葉 治

富士通 (株)

1. はじめに

マイクロプロセッサが高性能になっても、単一のマイクロプロセッサでは、汎用の中、大型機の性能には及ばない。そのため、複数のマイクロプロセッサを多数結合したマルチマイクロプロセッサとして、性能の良いシステムを、従来のコンピュータよりも安価に実現する試みがソフトウェア、ハードウェアとも進められている。^{1), 2)} 反面、飛躍的に向上するMIPSに比べ、相対的に研究が遅れているI/Oアーキテクチャの問題により、MIPSの活用はI/O負荷の低い、エンジニアリング系のアプリケーションが中心である。

即ち、マルチマイクロプロセッサシステムで解決しなければならない大きな問題の一つは、複数のマイクロプロセッサで共用する入出力装置をいかにコスト・パフォーマンスよくスケジュールするかということである。そこで、本稿は、マルチマイクロプロセッサシステムにおけるOS、特にデバイスドライバのI/Oスケジュールのアーキテクチャとして、DACC方式を提案する。

2. 従来のI/Oアーキテクチャ

従来のマルチマイクロプロセッサシステムにおけるI/Oアーキテクチャは、各プロセッサに、特定の入出力装置を接続したものであり、プロセッサが、直接接続されていない入出力装置に対してアクセスを行う場合、プロセッサ間通信を使用するリモートファイルアクセスが用いられている。^{3), 4)} (図1)。

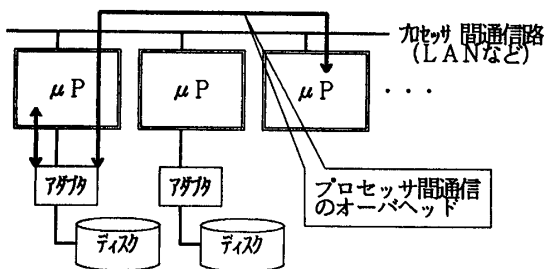


図1 従来のI/Oアーキテクチャ

しかし、これは、入出力アクセスにおけるプロセッサ間通信のオーバーヘッドが問題となり、プロセッサからの直接アクセスの場合と、プロセッサ間にまたがるアクセスでは、その性能に

ばらつきがある。したがって、プロセッサ台数が増加するにつれて、ディスクに格納するファイル、データベースの配置と、アプリケーションプログラムを起動するプロセッサのチューニングが必要となる。

3. 目標とするI/Oアーキテクチャ

上記の問題点を解決するためには、任意のプロセッサから、任意の入出力装置に直接アクセスできることが必要である。そこで、検討したI/Oアーキテクチャを図2に示す。

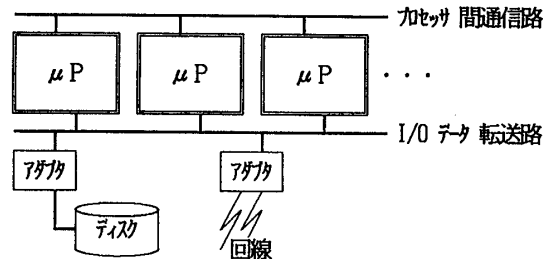


図2 目標とするI/Oアーキテクチャ

— I/Oデータ転送用バス

I/Oデータ転送用バスはプロセッサ、アダプタ間の通信と、I/Oデータの転送のための専用のバスである。

4. マルチμPシステム向けI/Oスケジュール方式

上記3.のI/Oアーキテクチャのもとで、マルチマイクロプロセッサシステム向けI/Oスケジュール方式として問題となるものに、どのプロセッサからアクセス要求を実行するかというアクセス方式と、入出力装置の構成変更を行うための装置管理や、ハードウェア障害発生時に対処を行う障害管理などの管理方式の2つがある。

4.1 分散型アクセス方式

アクセス方式としては、複数プロセッサの中の特定のプロセッサが集中してアクセスする方式と複数のプロセッサに分散してアクセスする方式の2つの方式が考えられる。

前者は、2.で述べた問題点に加え、すべてのプロセッサから入出力装置が共用できるアーキテクチャを活かせないことから、後者の分散型アクセス方式(DA方式、Distributed Access)

についてのソフトウェアの実現方式を検討した(図3)。

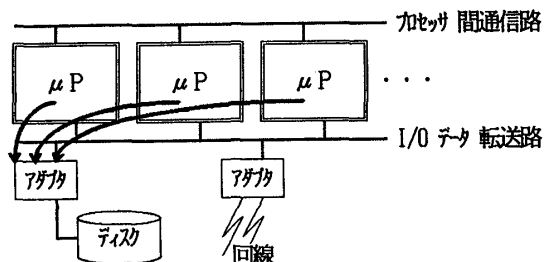


図3 DA方式

一般的に入出力装置は、一連のシーケンスに従って動作するものが多いため、アクセスを逐次化する必要がある。分散している複数のプロセッサが、非同期に同じ入出力装置にアクセスを行った場合、アクセスは競合し矛盾が生じる。これを、ソフトウェアのみでプロセッサ間通信を使用して逐次化すると、通信オーバーヘッドは特定のプロセッサに集中してアクセスすると同様に増大する。そこで、アダプタのファームウェアにより、入出力装置に対する入出力要求をプロセッサ単位で逐次化することとした(図4)。

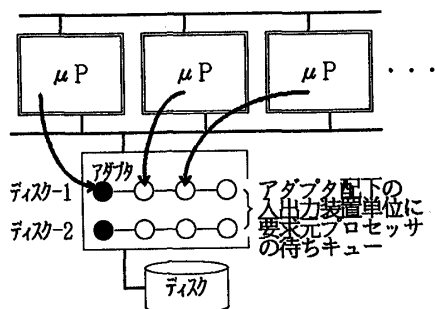


図4 アダプタによるキューイング

その結果、物理的には一つしか存在しないアダプタが、ソフトウェアからは仮想的にプロセッサの数だけ存在しているように見え、かつ、仮想的にアダプタを専有して使用することが可能となる。これにより、ソフトウェアはプロセッサ間での逐次化を行う必要がなくなり、通信オーバーヘッドをなくすることができる。以上のことから、DA方式の特徴は、

- I/Oスケジュールを行うプログラムをプロセッサに閉じて動作させることが可能となるため、入出力装置へのアクセスは、直接プロセッサ内のOSを呼び出すことで可能となり、プロセッサ間通信のオーバーヘッドが排除できる。

さらに、以下に示すような副次的な効果も期待できる。

- システム全体のプロセッサ間通信の負荷が低減する。
- 特定のプロセッサの停止による影響範囲が、停止したプロセッサで動作するアプリケーションのみとなる。そのため、障害の影響範囲は部分的であり、システム全体としての耐障害性が向上する。

4.2 集中型管理方式

入出力装置の構成変更を行う装置管理や、ハードウェア障害発生時に対処を行う障害管理などは、各プロセッサで分散して処理を行うのではなく、分散しているプロセッサを代表し、特定のプロセッサ上で動作するプログラムで集中的に管理し、入出力装置の構成やハードウェア障害時の対処は、このプログラムから各プロセッサに同期をとりながら指示を行う集中型管理方式(CC方式、Centralized Control)とした。

CC方式(図5)の特徴は、

- 管理情報の一貫性保証のための相互監視のオーバーヘッドが不要となる。
- 入出力要求の起動やデータ転送などのアクセスと比較すると、動作する頻度が低く、プロセッサ間通信のオーバーヘッドはあまり問題にならない。
- 入出力障害の解析では、他プロセッサからの再試行などシステム全体に関連する処理が容易となる。

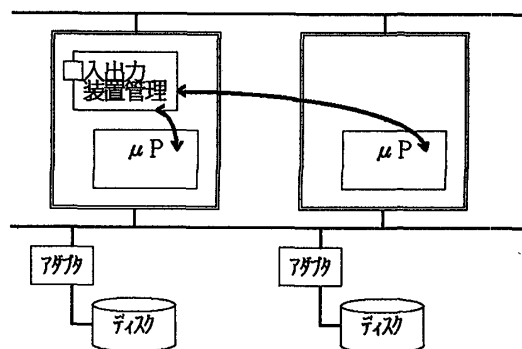


図5 CC方式

5. まとめ

マルチマイクロプロセッサシステムにおけるI/Oスケジューリング方式として、従来のI/Oアーキテクチャとは異なる任意のプロセッサから任意の入出力装置に直接アクセスできるI/Oアーキテクチャのもと、デバイスドライバ・レベルでコスト・パフォーマンスに優れたDACC方式を提案した。

今後さらに、ここに示したアーキテクチャのもとでのI/Oスケジューリング方式についての検討を充実させていく予定である。

参考文献

- 1) Smith III, T.B. et al. : The Fault-tolerant Multi-processor Computer, Noyes Publications (1986).
- 2) Heath, M.T. : Hypercube Multiprocessors 1987, Society for Industrial and Applied Mathematics (1987).
- 3) Sandberg, R. et al. : Design and Implementation of the Sun Network Filesystem, USENIX Conference Proceedings, Portland, Summer (1985).
- 4) Rifkin, A.P. et al. : RFS Architectural Overview, USENIX Conference Proceedings, Atlanta, Summer (1986).