レイヤ単位の並列処理によるプロトコル実装方式の性能評価

4N－5

加藤 聰彦† 佐藤 友実†† 鈴木 健二†
†国際電信電話(株)研究所 †† 電気通信大学

1. はじめに

伝送路の高速化に伴い、通信プロトコルの高速化が重要な課題となっている。これに対し筆者らは、近年広く普及している共有メモリ型マルチプロセッサ構成のワークステーション上において、並列処理方式によりプロトコル処理を高速化する手法について検討している[1, 2, 3]。筆者らの手法は、1つのレイヤのプロトコル処理を1つのCPUに割り当て、複数レイヤの処理をパイプライン的に並列処理することを特徴としている。このため、レイヤ間のインタフェースのためのキューや、複数のレイヤにまたがったグローバルバッファの管理に対して同期を必要としない方式を考案している。これまでに、HIPPI (High Performance Parallel Interface)を介した通信実験などにより本方式の評価を行い、その有効性を確認しているが[3]、本稿では、さらに並列度を高めた場合の本方式のオーバヘッドを評価した結果について報告する。

2. 並列処理方式の概要

筆者らが提案しているプロトコル並列処理方式においては、図1に示すように、各レイヤのプロトコル処理が、カーネルのサポートするスレッドで実現され、それぞれのスレッドが別個のCPUに割り当てられる[1]。また、PDU (Protocol Data Unit)の格納などに使用されるバッファは、すべてのスレッドからアクセス可能なバッファ領域に作成される(グローバルバッファ)。さらに、プロトコルスレッド間の情報のやり取りは、バッファ上に作成されたデータへのポインタをレイヤ間キューを介して受け渡すことにより行われる。

並列処理の効果を高めるためには、各プロトコルを処理するスレッドが、他のスレッドと同期なしに独立に動作できることが重要である。そこで、本方式では以下のような手法を用いて、スレッド間の同期制御の必要性を減らしている。

(1) レイヤ間キューは、読み手と書き手を1つのスレッドに制限すること、固定長の配列により実現し、キューの要素の動的な確保・解放の制御を行わないことなどにより、危険領域のロックなどを行うことなしに、読み手と書き手がキューにアクセスすることを可能としている。

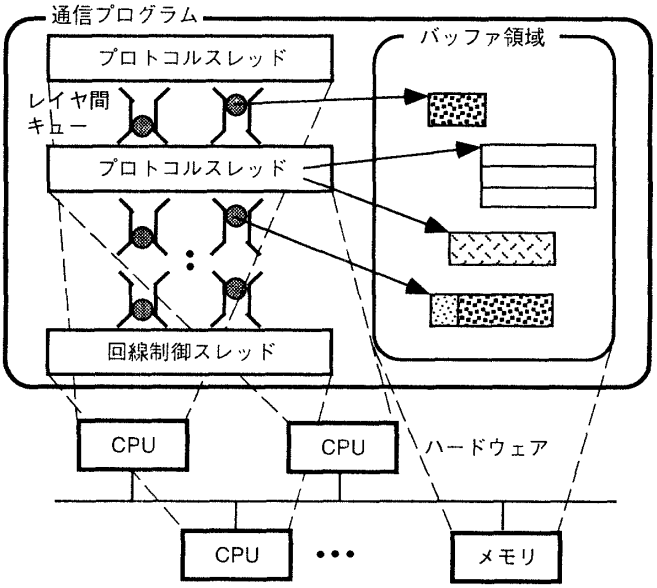

図 1: 並列処理プログラムの全体構成

(2) 1つのレイヤにおけるPDUの処理が終了し隣接レイヤに処理を依頼した後は、元のレイヤのスレッドではそのPDUの操作を行わないという、プロトコル処理の性質を用いて、PDU用のグローバルバッファに格納された情報に対する同期制御の必要性を少なくしている。

(3) グローバルバッファのためのバッファ領域については、複数のレイヤがバッファの確保／解放を行うため、基本的には同期制御が必要となる。ただし、バッファ領域において連続的な空き領域からバッファを獲得する処理については、危険領域のロックを行うことなく処理可能であるようなバッファ管理方式を導入している。

3. 性能評価実験

上記のようなプロトコルの並列処理方式をサポートするライブラリを、SiliconGraphics社のONYXワークステーション(OS: IRIX R5.3 System V, CPU: MIPS4400 250MHz × 12)を用いて実装した[2]。さらに、本ライブラリを用いて、次のような形で評価実験を行った。

(1) データの受信確認、フロー制御、誤り再送手順などの機能を含むコネクション型プロトコルを、複数レイヤ積み重ねてデータ転送の性能を評価する。このプロトコルの手順は、OSIのトランスポートプロトコルクラス4[4]の手順に準拠した。

"Performance Evaluation of Parallel Processing Implementation of Communication Protocols"
Toshihiko KATO†, Tomomi SATO†† and Kenji SUZUKI†
†KDD R & D Laboratories
††The University of Electro-Communications

(2) 800Mbps の伝送速度を有する HIPPI を用いた折り返し通信を用いて評価する。このため、最下位には、HIPPI への送信と HIPPI からの受信の処理を担当する回線制御スレッドを用意する。

(3) 12 個の CPU を用いて、プロトコルレイヤ数を 1 から 11 まで変更して実験を行う。測定においては、メモリ間のデータ転送を行い、転送するユーザデータの大きさを変更して評価する。

(4) 評価においては、500 個のユーザデータの転送して、その転送時間から性能 (スループット) を測定する。

この結果、表 1から表 3に示すような結果を得た。これらの表においては、それぞれのユーザデータサイズに対して、レイヤ数を変化させた場合の、以下の数値を示している。

- 測定において得られた性能 (Mbps)。
- プロトコルの処理能力の相対値、すなわち、性能 × レイヤ数を、レイヤ数 1 の場合の性能で割った値。
- 並列化の効率、すなわち、相対処理能力をレイヤ数で割った値。

表 1: 性能評価結果 (ユーザデータサイズ:1024 バイトの場合)

レイヤ数	1	3	5	7	9	11
性能 (Mbps)	4.6	4.3	3.9	3.6	3.2	2.9
相対処理能力	1	2.80	4.24	5.48	6.26	6.93
効率	1	0.93	0.85	0.78	0.70	0.63

表 2: 性能評価結果 (ユーザデータサイズ:8192 バイトの場合)

レイヤ数	1	3	5	7	9	11
性能 (Mbps)	33.2	31.1	28.8	26.8	24.2	22.5
相対処理能力	1	2.81	4.34	5.65	6.56	7.45
効率	1	0.94	0.87	0.81	0.73	0.68

表 3: 性能評価結果 (ユーザデータサイズ:32768 バイトの場合)

レイヤ数	1	3	5	7	9	11
性能 (Mbps)	95.1	91.2	87.9	84.2	75.4	70.1
相対処理能力	1	2.88	4.62	6.20	7.14	8.11
効率	1	0.96	0.92	0.88	0.79	0.74

また、HIPPI 回線を用いた折り返し通信の性能を測定した結果、データサイズが 1024 バイト、8192 バイト、32768 バイトで、それぞれ、4.7Mbps、35.8Mbps、98.2Mbps であった。

4. 考察

上記の結果から以下の考察を得た。

(1) 本実験における性能は、HIPPI 回線の通信性能に制限されている。従って、本方式により、プロトコル処理がボトルネックとならない高性能なプロトコル実装が可能であると考えられる。

(2) 評価の結果、プロトコルレイヤ数が 5 の場合で、並列度 (使用された CPU 数) の 90%程度、レイヤ数が 9 から 11 の場合で、70%程度の効率を保っている。この結果から、本方式は並列度が高くなった場合でも、スレッド間での同期などの並列処理のオーバヘッドを低く押さえることができたと考えられる。

(3) 従来は、プロトコル並列処理の手法として、1 つのメッセージに対する複数レイヤの処理を、1 つの CPU に割り当てる方式が広く検討されている。しかし、この方式では、コネクション管理テーブルなどの情報を共有する必要があり、並列度が増加するにつれて、その情報にアクセスするための同期制御のオーバヘッドが無視できなくなるという結果が報告されている[5]。これに対し筆者らの手法では、並列処理のオーバヘッドが無視できないという理由で、従来採用されていなかったレイヤ単位の並列処理を用いて、オーバヘッドの低い並列処理方式を実現した。

(4) 本方式では、スレッド間で同期をとる必要がある処理は、以下の通りである。

- PDU 用のグローバルバッファのリファレンスカウントへアクセスする場合に、それをロックする。
- グローバルバッファを開放する場合と、グローバルバッファの獲得時において現在使用している連続空き領域からバッファが確保できない場合に、グローバルバッファの管理情報全体にロックをかける。

実験結果において、レイヤ数が増加した場合に、並列化の効率が低下しているのは、これらの処理のオーバヘッドに起因すると考えられる。

5. おわりに

本稿では、筆者らが提案するレイヤ単位のプロトコル並列処理方式に対して、12 個の CPU を搭載するワークステーションを用いて大きな並列度におけるオーバヘッドを評価した結果について示した。これにより、並列度が 10 程度となった場合も約 70%の効率を保持し、筆者らの並列処理方式が有効であることが明らかとなった。

参考文献

[1] 加藤, 鈴木, "共有メモリ型マルチプロセッサを用いた通信プロトコルの並列処理方式," 情処マルチメディア通信と分散処理研究会, 67-17, March 1995.

[2] 佐藤, 加藤, 鈴木, "レイヤ単位の並列処理を用いた通信プロトコルプログラムの実装と評価," 情処マルチメディア通信と分散処理研究会, 73-19, Dec. 1995.

[3] 佐藤, 加藤, 鈴木, "レイヤ単位の並列処理方式を用いた通信プロトコルプログラムの HIPPI による通信実験," 情処第 52 回全大, 1Bb-4, March 1996.

[4] CCITT X.224, "Transport Protocol Specification for Open Systems Interconnection for CCITTApplications," 1988.

[5] M.Bjorkman and P. Gunningberg, "Locking Effects in Mutiprocessor Implementations of Protocols," Proc. SIGCOMM '93, Sept. 1993.