ハイパフォーマンス
コンピューティング 77−24
（1999. 8. 3）

マルチスレッドを利用した分散共有メモリシステムにおけるスケジューリング属性の影響

南里 豪志† 佐藤 周行‡ 島崎 眞昭*

† 九州大学大型計算機センター, ‡ 東京大学大型計算機センター, * 京都大学大学院工学研究科

概要

ソフトウェア DSM システムの核となるリクエスト処理システムの実装において, マルチスレッドは最も有効な手法の一つである. このようなマルチスレッドを用いたシステムの性能は, スレッドのスケジューリング方法に大きく依存する. そこで本稿では, POSIX スレッドインタフェースを用いたソフトウェア DSM システムを実装し, スケジューリング方法が性能に与える影響の評価を行った. この DSM システムでは, メインスレッドが計算を行う傍ら, 非同期に到着するリクエストをハンドラスレッドが処理する. 実験の結果, スケジューリング方法がシステムの性能に与える影響はリモートリクエストやバリア同期を行う間に実行される計算の量に依存することが分かった.

Effects of Thread Scheduling on the Performance of Multithread-Based Software DSM System

Takeshi Nanri† Hiroyuki Sato‡ Masaaki Shimasaki*

† Computer Center, Kyushu University, ‡ Computer Centre, University of Tokyo,
*Faculty of Engineering, Kyoto University

Abstract

Using a multithread model is one of the most efficient ways to implement a request handling mechanism, a core part of a software DSM system. The performance of multithread-based systems depends heavily on the scheduling of threads. Therefore, in this paper, a software DSM system is introduced to evaluate the effects of thread scheduling with POSIX thread interface. On each PE of this DSM system, the handler thread handles asynchronous remote requests, while the main thread computes. Results from experiments show that the effect of this scheduling strategy on the performance of the system depends on the amount of computation between remote requests or barrier synchronizations.

1 Introduction

Ethernet 等のネットワークで接続された複数の PC や WS で構成されたクラスタ環境で, 特別なハードウェアを用いずに仮想的な共有メモリを提供するソフトウェア DSM システムは, 容易に構築し, 利用できる共有メモリ型並列環境として, 広く研究が行なわれている [1, 2, 3, 4, 5, 6, 7]. このようなシステムを構築する場合に核となるのは, 非同期に到着するリクエストを処理するリクエスト処理機構である. この機構の, 最も簡潔で効率の良い実装手段の一つがマルチスレッドを用いる方法である. マルチスレッドを用いたシステムでは, スレッドのスケジューリングがシステム全体の性能に大きく影響する. そこで本稿では, マルチスレッドを用いたソフトウェア DSM システムの性能に対するスレッドスケジューリングの影響を計測し, 評価する.

本稿では, 実験環境として, 簡単なソフトウェア DSM システムを構築し, 利用する. この DSM システムでは, 各 PE 上でメインスレッドとハンドラスレッドの二つのスレッドが動作する. メイン

スレッドが計算を行なう傍ら，ハンドラスレッドは非同期に到着するリクエストの処理を行なう．この DSM システムは，POSIX スレッドインタフェースを提供するスレッドライブラリ Linux Thread を用いて構築される．POSIX スレッドインタフェースでは，スケジューリングポリシーおよび優先度で構成されるスケジューリング属性によって，各スレッドのスケジューリング方法を決定する．そこで，4 つの並列プログラムについて，この DSM システムのメインスレッド，ハンドラスレッドのスケジューリング属性をそれぞれ変化させて，処理時間を計測し，評価を行なう．

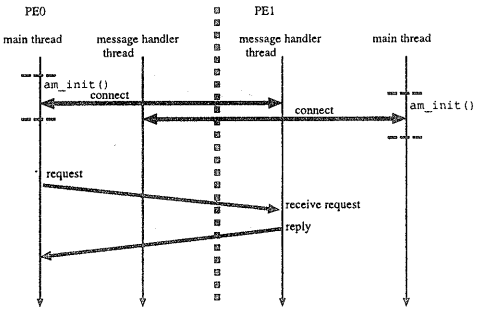

図 1: マルチスレッドを用いた DSM システムのリクエスト処理機構

2 マルチスレッドを用いたソフトウェア DSM システム

本稿の DSM システムでは，各 PE でメインスレッドとハンドラスレッドの二つのスレッドを動作させることにより，到着するリクエストの処理を行なう．ほとんどのプログラムで，リクエストは受信側と同期せずに送信されるため，polling のように受信側が一つのコンテキストで計算とリクエスト処理を行なう方法では，効率良く並列処理を行なうことは困難である．マルチスレッドを用いた DSM システムの実装では，計算を行なうコンテキストからリクエスト処理を行なうコンテキストへのスイッチが自動的に，しかも高速に行なわれるため，簡潔に，性能の高いシステムを構築できる．

本稿の DSM システムのリクエスト処理機構を，図 1 に示す．

まず `am_init` では，各 PE 上にハンドラスレッドを生成し，メインスレッドから他の各 PE のハンドラスレッドにそれぞれ TCP ソケットを接続する．以降の計算で行なわれる PE 間のすべての通信は，これらのソケットを用いて行なわれる．

ここで，例えば PE0 が PE1 の所有するデータを読み込む場合，以下の手順で通信が行なわれる．

1. PE0 のメインスレッドが PE1 のハンドラスレッドに対してリクエストを送信し，返信メッセージが到着するまでブロックする．
2. PE1 のハンドラスレッドにコンテキストがスイッチし，リクエストが処理される．
3. PE1 のハンドラスレッドが PE0 のメインスレッドに返信メッセージを送信し，PE1 のメインスレッドにコンテキストがスイッチする．
4. PE0 のメインスレッドが返信メッセージを受信し，処理を再開する．

3 POSIX スレッドのスケジューリング属性

本稿では，CPU を一つだけ持つシングルプロセッサ計算機で構成されるクラスタ上に構築された DSM システムを対象とする．シングルプロセッサ計算機上で複数のスレッドを動作させる場合，システムの性能はスレッドのスケジューリング方法に大きく依存する．

本稿の DSM システムは，POSIX スレッドのインタフェースを提供する Linux Thread ライブラリを用いて実装する．POSIX スレッドのインタフェースでは，スレッドのスケジューリング方法を各スレッドのスケジューリング属性によって指定する．このスケジューリング属性はスケジューリングポリシーと優先度で構成される．スケジューリングポリシーは，同じ優先度を持つ複数のスレッドに対するスケジューリング方法

を指定するもので, POSIX 規格で定められているのは以下の二つである.

FIFO : 現在実行中のスレッドは, I/O 要求によってブロックされるか, もしくはより高い優先度を持つスレッドに資源を横取りされるまで実行を続ける.

RR(Round Robin) : 実行中のスレッドは, 定められた時間しか実行を続けられない.

これらの他に, 実装依存のスケジューリングポリシーとしてOTHER が用意されているが, 本稿では扱わない.

4 実験

4.1 実験の概要

本稿の実験は, 100Mbps Ethernet のスイッチングハブで接続された PC (Pentium 166MHz, 128MB) 8 台を用いて行なう. OS は Linux 2.0.35, スレッドライブラリは Linux Thread 0.7 を用いる. この環境において, プログラムの処理時間を計測し, スレッドスケジューリングが DSM システムに与える影響を議論する.
まず, システムの基本性能を計測するため, 他のプロセッサが所有するデータに対する read リクエストの応答時間を計測する. 次に, 3 つの並列プログラム (LU 分解, 行列積, Bitonic Sort) を用いて実験を行う. これらの 3 つのプログラムについては, 全体の処理時間及び通信を全く行わない場合の計算時間を計測する. この計算時間は, それぞれのプログラムからメモリアクセス部分と同期部分を削除して実行させた場合の経過時間である. すなわち, 処理時間と計算時間の差が, 通信, 同期及びメッセージ処理に要した時間の合計である.
これらのプログラムについて, FIFO, RR それぞれのスケジューリングポリシーについて, 優先度を以下のように設定し, 計測を行なった.

even 各 PE の両スレッドが同じ優先度を持つ.

high 各 PE のメインスレッドがハンドラスレッドよりも高い優先度を持つ.

表 1: スケジューリング属性の応答時間及びリクエスト処理コストへの影響

pattern	ave	max	min	calc
FIFO even	0.7777	77.579	0.0017	77.546
FIFO high	0.7777	77.586	0.0016	77.547
FIFO low	0.0023	0.0399	0.0017	86.474
RR even	0.2090	0.2400	0.1297	78.306
RR high	0.7771	77.522	0.0017	77.484
RR low	0.0022	0.0363	0.0017	86.386

(sec)

low 各 PE のメインスレッドがハンドラスレッドよりも低い優先度を持つ.

4.2 Basic Performance

スケジューリング属性がシステムの基本性能に与える影響を計測した. 実験に用いたプログラムは, PE1 が 400×400 の行列積を計算中に PE0 が PE1 から 10KByte のデータを 100 回読み出すものである. このプログラムについて, PE0 におけるリクエストの応答時間の平均値, 最大値及び最小値を, さらに PE1 における全体の処理時間を計測した.
表 1 に実験結果を示す. これにより, スケジューリング属性がリクエストの応答時間, 及びリクエストの処理コストに与える影響は次の 3 つに分類できる.

1. メインスレッドの優先度が低い場合:
   スケジューリングポリシーに係わらず, メインスレッドの優先度が低い場合, リクエストの応答時間が最短になる反面, PE1 の計算時間が最長になる. これは, PE0 からのリクエストが到着する度に PE1 の計算が割り込まれ, ハンドラスレッドに処理が移るためである.

2. メインスレッドの優先度が高い場合, もしくは FIFO ポリシーで両スレッドの優先度が同じ場合:
   この場合, リクエストの応答時間が最長になる. 特に, 応答時間の最大値は PE1 の計算

時間とほぼ同じ値になる．これは，PE1 のメインスレッドが計算を終了し，最後にバリア同期を実行するまでハンドラスレッドに処理が移らないためである．そのため，PE1 の計算時間は最短となる．

3. RR ポリシーで両スレッドの優先度が同じ場合:
この場合，PE0 の応答時間が 0.12 ～ 0.24 秒となる．これは，各スレッドが連続して実行できる時間に制限があるためである．Linux Thread の場合，RR でスケジュールされたスレッドが連続して実行できる時間は 0.2 秒である．

4.3 LU 分解

次に，並列 LU 分解プログラムの処理時間に対するスケジューリング属性の影響を計測した．実験に用いたプログラムでは，行列は各行毎にサイクリックに分割され，各 PE に割り当てられる．また，計算は owner-computes rule によって割り当てられる．そのため，3 重ループ中の外側ループの i 番目のイタレーションの最初に，各 PE が i 番目の行を，その行を所有する PE のメモリから読み出す．

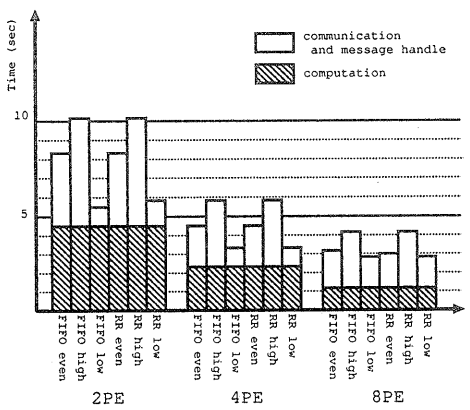

図 2: スケジューリング属性の影響 (LU 分解)

図 2 に，大きさが 400 × 400 の行列の LU 分解の処理時間に対するスケジューリング属性の影響を示す．これにより，4.2 節で示したスケジューリング属性の影響の分類 1，すなわちメインスレッドの優先度が低い場合，最も良い性能が得られることがわかる．これは，リクエストに対する応答時間を短縮することにより，システムの性能が向上されることを示している．
しかし，スケジューリング属性の性能への影響は，使用するプロセッサ台数に依存する．プロセッサ台数が 4PE の場合，スケジューリング属性の相違による処理時間の差の最大値は 4.8 秒であったのに対し，プロセッサ台数が 8PE の場合，この最大値は 1.3 秒であった．すなわち，プロセッサ台数が 2 倍になると処理時間の差が約 1/4 になった．

4.4 行列積

行列積については，各行列を行方向のブロック単位に各 PE に分割し，owner-computes rule で計算を分割することによって並列化したプログラムを用いた．そのため各 PE は，他の PE に割り当てられた部分行列を一括して読み出し，そのデータを利用して計算を行なう，というフェーズをプロセッサ台数と同じ回数繰り返す．また，各 PE は割り当てられたすべての計算を終えた後にバリア同期を実行する．

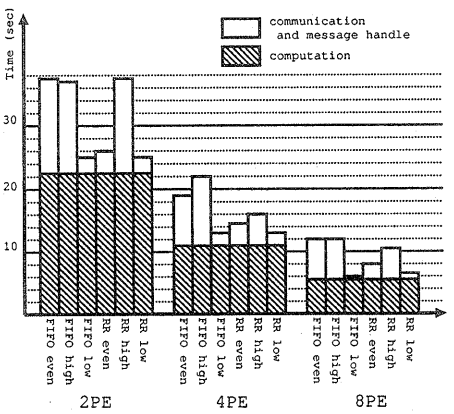

図 3: スケジューリング属性の影響 (行列積)

行列の大きさがそれぞれ 400 × 400 の行列積の

処理時間に対するスケジューリング属性の影響を図 3 に示す．これにより，LU 分解の場合と同様，メインスレッドの優先度が低い場合に最も良い性能が得られることがわかる．また，スケジューリング属性の相違による処理時間の差の最大値は，プロセッサ台数が 4PE の場合に 12.3 秒であったのに対し，プロセッサ台数が 8PE の場合は 6.0 秒であった．すなわち，プロセッサ台数が 2 倍になると，処理時間の差が約 1/2 になった．

4.5 Bitonic Sort

最後に，Bitonic Sort を用いた実験の結果を示す．Bitonic Sort のアルゴリズムは，まず各 PE に割り当てられた配列をローカルに整列した後，バタフライ型で通信を行ないながら他の PE の配列と自分の配列を融合するものである．実験で用いたプログラムでは，データの転送終了と配列の融合の終了を確認するため，他の PE から配列を読み出す直前と直後にバリア同期を実行する．そのため，このプログラムでは，ある PE の計算中に他の PE からのリクエストが到着することがない．

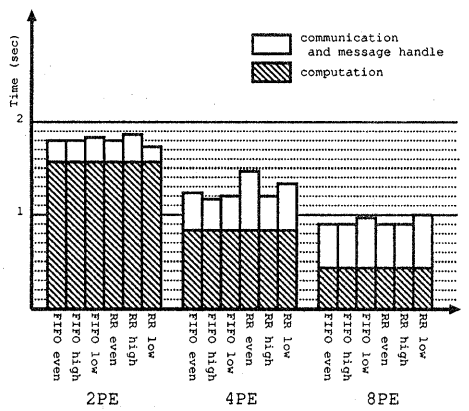

図 4: スケジューリング属性の影響 (Bitonic Sort)

図 4 に実験結果を示す．Bitonic Sort の場合は，スケジューリング属性の相違による処理時間の有意な差は見られなかった．

5 考察

実験結果より，LU 分解や行列積では，スケジューリング属性が性能に大きく影響することがわかった．また，メインスレッドの優先度を低くし，リクエストの応答時間を短くすることによって良い性能が得られることもわかった．これに対して Bitonic Sort では，スケジューリング属性の性能への影響は見られなかった．これは，Bitonic Sort では各フェーズで他の PE から配列を読み出す前と後にバリア同期を実行するためである．バリア同期や他の PE へのメモリアクセスを行なうと，メインスレッドはスケジューリング属性に関わらずブロックする．すなわち，送信リクエストの前後にバリア同期を行う Bitonic Sort では，スケジューリング属性に関わらず，リクエストの応答時間が短くなるため，性能に有意な差が見られなかった．

また，LU 分解や行列積において，スケジューリング属性の性能への影響は使用するプロセッサ台数に依存する．これは，バリア同期や他の PE へのメモリアクセスの間に連続して行なわれる計算量が使用プロセッサ数に依存するためである．N を行列の大きさ，P を使用プロセッサ台数，$comp$ を最内ループの 1 イタレーションを実行する時間として，LU 分解，行列積の処理性能へのスケジューリング属性の影響を定式化する．

LU 分解では，最外ループのイタレーション i で 1 回の共有メモリへのアクセスと $\frac{N(N-i)}{P} \times comp$ の計算を行なう．そのため，メインスレッドがハンドラスレッドより高い優先度を持つ場合，すなわち，メインスレッドがブロックするまでコンテキストスイッチが起こらない場合，メインスレッドがハンドラスレッドより低い優先度を持つ場合に比べ，共有メモリへのアクセスの応答時間が最大で $\frac{N(N-i)}{P} \times comp$ だけ遅れる．この差がスケジューリング属性の影響である．プログラム全体では，この差は $\frac{N^2(N-1)}{2P} \times comp$ となる．すなわち，スケジューリング属性の影響はプロセッサ数に反比例する．

これに対して行列積では，各 PE が，プログラム全体で $P-1$ 回の共有メモリへのアクセスと，$\frac{N^3}{P} \times comp$ の計算を行なう．そのため，

$\frac{N^3}{P^2} \times comp$ 毎に共有メモリへのアクセスを実行する．行列積では，メインスレッドが計算中にブロックするのは共有メモリへのアクセスを行なう場合のみなので，$N^3(\frac{1}{P} - \frac{1}{P^2}) \times comp$ が，スケジューリング属性の性能への影響となる．この式より，行列積でも，プロセッサ台数が増加するとスケジューリング属性の影響が減少するが，減少の度合は LU 分解の場合より小さいことがわかる．

6 まとめ

本稿では，マルチスレッドを利用したソフトウェア DSM システムにおける，スケジューリング属性の性能への影響を計測し，評価を行なった．計測の結果，メインスレッドの優先度をハンドラスレッドより低くすることにより，リクエストの応答時間が短くなり，システム全体でも最高の性能が得られることがわかった．また，スケジューリング属性の影響が，実行するプログラムや使用するプロセッサ台数に依存することもわかった．
今後は，キャッシュメモリや非同期通信等を提供する DSM システム，さらに，SMP クラスタ上に構築された DSM システムのように，より複雑なシステムについて解析を行なう予定である．

参考文献

[1] Dubnicki, C., Bilas, A., Li, K. and Philbin, J.F., "Design and Implementation of Virtual Memory-Mapped Communication on Myrinet," In Proceedings of 11th International Parallel Processing Symposium, 1997.

[2] Amza, C., Cox, A.L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaenepoel, W., "TreadMarks: Shared Memory Computing on Networks of Workstations," IEEE Computer, 29(2):18–28, 1996.

[3] Liang, W.Y., King, C.T. and Lai. F., "Adsmith: An Object-based Distributed Shared Memory for Network of Workstations," IEICE Transaction on Information and Systems, E80-D(9):899–908, 1997.

[4] Nanri, T., Sato, H. and Shimasaki, M., Implementing a Portable SPMD Shared-Memory Model Parallel Language in a Distributed Computing Environment. In *Proceedings of International Symposium on Parallel and Distributed Supercomputing*, pages 243–252, 1995.

[5] Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M., "PM: An Operating System Coordinated High Performance Communication Library," In High-Performance Computing and Networking, volume 1225 of Lecture Notes in Computer Science, pages 708-717. Springer-Verlag, April 1997.

[6] Carter, J.B., Bennett, J.K. and Zwaenepoel, W., "Implementation and Performance of Munin," In Proceedings of the Thirteenth Symposium on Operating Systems Principles, pp.152–164, October 1991.

[7] Chien, A., Pakin, S., Lauria, M., Bunchanan, M., Hane, K., Giannini, L. and Prusakova, J. "High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance," Eighth SIAM Conference on Parallel Processing for Scientific Computing (PP97); Mar. 1997.

[8] D.E. Culler, A. Dusseau, S.C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick, "Parallel programming in Split-C," Supercomputing '93, pp. 262–273, 1993.