システムソフトウェアと
オペレーティング・システム 65−21
（１９９４．７．２２）

# 分散共有メモリ上の分散スレッド実行環境の構成と評価

数藤 義明, 鈴木 茂夫, 吉本 雅彦, 柴山 茂樹

キヤノン株式会社
情報メディア研究所

概要

本稿では，高速なネットワークで相互に接続された計算機資源をワークステーションクラスタとして効率よく利用するために，分散共有メモリの問題点について主にページスラッシングの点から議論し，分散共有メモリ上にタスク/スレッドモデルに基づくアプリケーションプログラムの実行環境を提供する分散スレッド実行環境 (**PARSEC**: PARallel Software Environment for workstation Cluster) の構成ついて述べる．また，**PARSEC** に用いられているページスラッシング低減と負荷分散を独立に扱うための２レベル・スケジューリング方式について述べる．

# The Design and Evaluation of a Distributed Threads Execution Environment on Distributed Shared Memory

Yoshiaki Sudo, Shigeo Suzuki, Masahiko Yoshimoto, Shigeki Shibayama

Media Technology Laboratory
Canon Inc.

Abstract

In this paper, we first discuss page-thrashing in distributed shared memory, then present the design and evaluation of **PARSEC** (PARallel Software Environment for workstation Cluster) which provides application software with a distributed threads execution environment on workstation clusters. We also propose a two-level scheduling method which is used in **PARSEC** for both load sharing and page-thrashing prevention.

## 1 はじめに

高速なネットワークによって相互に接続された計算機資源を効率良く利用するために，数多くの分散オペレーティングシステム (Operating System:OS) の研究がなされている．分散OSでは，各計算機上に存在する分散したプロセス間の通信方式について多くの研究がされてきた．近年，高速なネットワークを効率よく利用し，アプリケーションプログラマに共有メモリという形でプロセス間の通信手段を提供する方法として，分散共有メモリ (DSM:Distributed Shared Memory)[1,2] が注目されている．

また，密結合型並列計算機において，複数のプロセッサ資源を効率良く利用するためにタスク/スレッドモデルが提案されている．これは，プロセッサ資源の割り当ての単位をスレッドとし，それ以外の資源割り当ての単位をタスクとして，タスク内に複数のスレッドを実行可能とした物である．高速なネットワークによってクラスタ結合された計算機上に構築した分散共有メモリ上で，タスク/スレッドモデルに基づくアプリケーションプログラムを実行する環境が考案されている．

本論文では，高速なネットワークで相互に接続された計算機資源をワークステーションクラスタとして効率よく利用するために，分散共有メモリの問題点について主にページスラッシングの点から議論し，分散共有メモリ上にタスク/スレッドモデルに基づくアプリケーションプログラムの実行環境を提供する分散スレッド実行環境 (**PARSEC**: PARallel Software Environment for workstation Cluster) の構成と評価について述べる．

## 2 分散共有メモリにおける問題点

分散共有メモリ [1,2] はネットワークで接続された計算機 (ノード) 上で仮想的に提供される共有メモリである．ソフトウェアによる分散共有メモリの実現は，分散システムの各ノード上の主記憶に対してマルチプロセッサシステムのキャッシュメモリと同様にコンシステンシ保持動作を行なうことで，仮想的に共有メモリを実現する．このコンシステンシ保持動作は，プログラムが起こす仮想記憶空間に対するページフォールトに従ってページ単位で行なわれ，プロトコルとしては多くの物には Write-Invalidate 方式が用いられている．

分散共有メモリ上の仮想記憶空間で複数のスレッドを動作させるプログラムの実行効率の向上を妨げるものとして最も重大なものは，ページスラッシング (Page-Thrashing, Ping-Pong effect) と呼ばれるノード間でページデータの移動が頻繁に発生する現象である．また，分散共有メモリへのアクセス遅延を充分に小さくし，分散したスレッドの並列性を保持することも性能上重要な鍵となる．そこで以下では，そのページスラッシングについて分析し，ページスラッシングの低減方法についてアクセス遅延による並列性の低下と共に考察を行なった．

### 2.1 ページスラッシング

分散共有メモリのコンシステンシ保持動作として Write-Invalidate 方式を用いる場合には，この方式が分散共有メモリ空間のページについては基本的に1つのノードに対してのみしか書込みを許可しない Single-Writer プロトコルであるために，各ノード上のスレッドが同じデータに対して書込みを行った場合には，頻繁にそのページデータがノード間を転送されたり，他のノードへの無効化要求のメッセージ転送が行なわれたりする．このようなページスラッシングは，プログラムの実行効率を著しく低下させる．ここでページスラッシングを，次に述べる性質の異なる2つに分類する．

**Thrashing 1.** あるノード上のスレッドがデータ構造に書込み中に，他のノードで実行されているスレッドが同じデータ構造内にある同期変数に対してアクセスする場合や，そのデータ構造を読み込む場合．

**Thrashing 2.** 同じページ内に配置された異なるデータ構造を，複数のノードで実行されているスレッドがアクセスする場合[1]．

### 2.2 ページスラッシングの低減方法

以上に述べたようなページスラッシングを低減するための方法として，大きく分類して3つの方法が考えられている．

#### 2.2.1 データの配置，割り当てを変更する方法

データ構造の配置を変更しデータ構造毎に1ページの領域を割り当てたり，分散したスレッドにデー

[1] この場合全てのノード上のスレッドが読み込み動作しかしない場合は，各ノード上にデータが書込み不可状態でキャッシングされるのでスラッシングは発生しない．

タ構造をあらかじめ割り当てて計算させる場合，そのデータ構造の割り当て方法を変更して分散したスレッド毎に異なるページ上のデータを割り当てる方法である．これは，分散共有メモリへのアクセス遅延を抑え，各ノード上のスレッドの並列性を保持した上で **Thrashing 2.** を根本的に解決する手段である．ただし，異なるノード上のスレッドでアクセスされる可能性のあるデータ構造を完全にページ単位に配置することは，一般的なアプリケーションプログラム上では非現実であるが，コンパイラによる大きなデータ構造の配置の最適化の研究も行なわれている．また，ライブラリ関数等ではデータ構造の配置を最適化したものを用意することは重要であろう．ただしこれだけでは **Thrashing 1.** を低減することは不可能である．

#### 2.2.2 メモリコンシステンシ保持を緩くする方法

また，分散共有メモリのメモリコンシステンシ保持を緩くする方法によってページスラッシングの低減を行なうことが可能である．

**緩いメモリコンシステンシモデル** Release consistency[4,5,6,7] に代表される緩いメモリコンシステンシモデルを，分散共有メモリ管理サーバで使用する．

→ **Thrashing 1. 2.** 共に低減可能である．分散共有メモリへのアクセス遅延も低減可能であり，並列性も保持できる．

**Time Window Lock 方式 [8]** あるページが書込み可能になった場合，そのページはある時間内 (Time Window) その状態を保持させ，他のノードからの要求を遅延させる．

→**Thrashing 1. 2.** 共に低減可能である．しかし，アクセス遅延が増加し，スレッドのストールが大きくなって並列性が低下する．最適な Time Window 幅を決定するのも困難である．

**Multiple-Writer プロトコル [5,7]** あるページを複数のノードで書込み可能にし，サーバ側で書き換えられた部分をチェックしコンシステンシを保持する．

→ **Thrashing 2.** を低減可能である．アクセス遅延も低減可能であり，並列性を保持できる．しかし，分散共有メモリ管理サーバの負荷が増す．

#### 2.2.3 スレッドマイグレーションによる方法

ページスラッシングを低減する方法として，スレッドマイグレーションを行なうことによって，同じページにアクセスするスレッドを一つのノード上で動作させる方法が提案されている [9]．これは，各スレッドがページフォールトしたメモリアドレスについての情報から，スレッドが現在主にアクセスしているデータが存在するノードにスレッドを移送して，ページスラッシングを低減する方法である．このように，ページスラッシングを低減する目的でスレッドマイグレーションを行なうことは，分散システム全体の負荷分散をも考慮すると，負荷均等化の妨げになるし，スレッドマイグレーションが大きな負荷になり容易ではない．

## 3 PARSEC の構成

**PARSEC** は，前章で述べたページスラッシングを低減し，分散システム内の各ノードの負荷分散を効率的に行なうことを目的として開発した，分散共有メモリ上の分散スレッド実行環境である．**PARSEC** は，図 1 に示されるように，分散したノード上で動作するスレッドと，そのスレッドが動作する枠組みである分散共有メモリ上の分散タスクを提供する．

分散共有メモリを管理し，分散タスク内の各ノード上のメモリのコンシステンシ保持動作を行なう分散共有メモリ管理サーバ (DSMmanager) と，分散タスクの生成・消滅やカーネルスレッドの割り当てなどを管理する分散タスク管理サーバ (Dtaskmanager) と，分散タスク内でスレッドの制御やユーザレベルスレッド (アクティビティ) のスケジューリングを行なう分散スレッドライブラリ (Dthread ライブラリ) によって構成されている．**PARSEC** では，スケジューラを 2 レベルに分割したことを特徴としている．分散システム全体の負荷分散を目的とするグローバルスケジューラと，ページスラッシングの低減を目的とするローカルスケジューラとで，効率の高い分散環境を提供することを狙う．

### 3.1 分散共有メモリ

**PARSEC** では，分散共有メモリ上にマルチスレッドの分散実行環境を構築するために，分散共有メモリの性能がプログラムの実行効率に大きな影響を与える．そのため，前章で述べたようなページスラッシングの低減方法の多くを使用する．

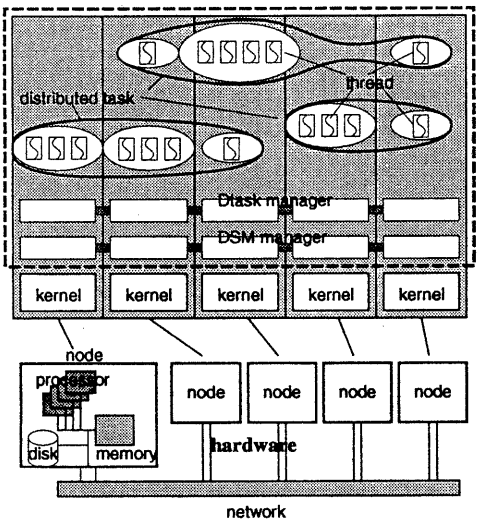


図 1: **PARSEC** の基本構成

具体的には，ライブラリ関数で用いるデータ構造の配置をページ単位を意識して再配置し，分散スレッドに対して最適化したものを用意する．また，アプリケーションプログラムに対して緩いメモリコンシステンシモデルをサポートする分散共有メモリを提供する[2]．さらに，提案する 2 レベル・スケジューリング方式で，スレッドマイグレーションを行なってページスラッシングを低減する．

### 3.2 分散タスク/スレッド

タスク/スレッドモデルを分散システム内の分散共有メモリ上に実現した物を分散タスク/スレッドと呼ぶ．分散タスクは単一の仮想記憶空間を保持し，分散スレッドは分散システムの各ノード上からその仮想記憶空間全体にアクセス可能である．また，**PARSEC** では図 2 に示されるように分散タスク/スレッド上にアクティビティと呼ぶユーザレベルスレッドを動作させる．分散タスク内の仮想記憶空間が各ノード上で共有されているために，アクティビティのノード間移動が容易となる．そのためにカーネルスレッドとは異なり，スレッドマイグレーションがアクティビティ移動によって実現されコストが低い．

[2]分散共有メモリの使用者が特に指定しない限りは，キャッシュのコンシステンシプロトコルとして Write-Invalidate 方式を，メモリコンシステンシモデルとして Sequential Consistency[3] を用いる．

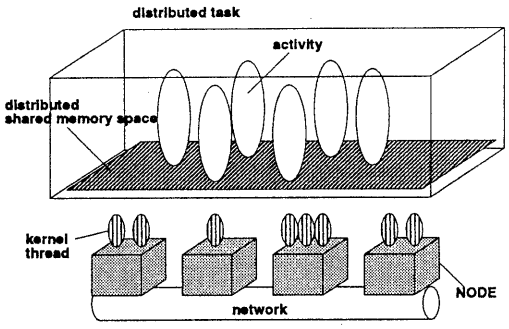


図 2: **PARSEC** の分散タスクとアクティビティ

また，遠隔ノードでの操作 (ノードに固定されたタスクとの通信を行なう場合や，各ノード上の入出力装置を扱う場合等) がアクティビティのノード間移動によって簡単に実現できる．

### 3.3 分散タスク管理サーバ

分散タスク管理サーバは，主に分散システム全体の負荷分散を目的として，分散タスクの生成や消滅を行なう．主な機能としては，分散タスクが他のノードへ拡張を要求した場合に負荷の小さなノードを選択して，そのノード上に分散タスクを拡張するためのタスクを生成し分散タスクに結合する．また，以下で述べるスケジューリング方式における動的なタスクスケジューリングも行なう．

### 3.4 2 レベル・スケジューリング方式

**PARSEC** におけるスケジューラは，分散システム内の各ノードの負荷分散を目的としたグローバルスケジューラと，分散タスク内のページスラッシングによるノード間の通信量 (通信回数) の低減と分散スレッドの並列性の保持を目的としたローカルスケジューラとに分割されている．これらは，従来のカーネルスレッドだけからなるシステムでは分割が困難であったものであるが，本システムのアクティビティを用いた方式では，それぞれが完全に独立にスケジューリングを行なうことが可能となる．このスケジューリング方式を，2 レベル・スケジューリング方式と呼ぶ．

#### 3.4.1 グローバルスケジューラ

グローバルスケジューラは分散タスク管理サーバ内で，主として各ノード間の負荷分散を行なう．分

散タスク管理サーバは，分散タスクから新しいノードへの拡張を要求された場合には，分散システム内で負荷の小さなノードを選択して分散タスクの拡張を行なう．また，各ノードの負荷を均等化するために，一度ある分散タスクが拡張したノードであってもその分散タスクを圧縮してそのノード上での動作を停止したり，分散タスクのノード内のカーネルスレッドを減少させたりすることも行なう．このような分散システム内の負荷均衡は，従来の分散OSではプロセス(タスク)マイグレーションによって行なわれていたものであるが，**PARSEC**では複数の分散タスクを分散システム内の各ノードに流動的に拡張したり圧縮したりすることによって，プロセス(タスク)という大きな単位でなくスレッドという単位で柔軟に行なうことが可能となる．

#### 3.4.2 ローカルスケジューラ

ローカルスケジューラはDthreadライブラリ中で，分散タスク内のスレッドが効率良く動作出来るように，さらにページスラッシングを低減するために，分散タスク内のアクティビティのスケジューリングを行なう．基本的なローカルスケジューラのスケジューリング方式は，分散共有メモリ管理サーバから提供される情報を用いて行なう．分散共有メモリ管理サーバは，分散タスク内のスレッドのページフォールト回数を計測し，あるアクティビティのアクセス情報を調査する．また，Dthreadライブラリはスレッド毎の排他制御や同期に関する情報を保持している．この2つの情報を用いてスケジューリングを行なう方法として，以下の2方式を考案した．

**スレッド主導方式** スレッド主導方式は，各アクティビティ間の関連を基にスケジューリングを行なうスケジューリング方式である．各アクティビティのページフォールト回数を計測し，それによってスレッドに設定された変数を変化させてその変数を評価してスケジューリングを行なう．**PARSEC**では，計測したアクセス情報によってアクティビティ同士の関連度を評価する．つまり，あるスレッドがページフォールトを起こしたページに対して，他のスレッドがページフォールトを起こした場合に，それらのスレッドの関連度を示す変数が増加させる．このような評価に従って，あるノード内で実行されている，または直前まで実行されていたアクティビティと最も関連度の評価の高いアクティビティを実行対象とする．こうすることによって間接的にページスラッシングを低減することが可能となる．

**データ主導方式** データ主導方式は，アクティビティがアクセスしようとするデータ構造に関する情報を利用するスケジューリング方式である．これは，アクティビティの再スケジューリング時が条件同期やサスペンドロックの解除の場合であることを利用して，サスペンドロックの解除時にはそのサスペンドロックの排他制御対象であるデータ構造のページが存在するノードで実行されるようにスケジューリングを行なう．

以上の2方式を比較すると，ページスラッシングを低減するという点では，データ主導方式のほうが頻繁にアクセスされるデータをノードに固定出来るので有利であるが，分散スレッドの並列性の保持という点では，データ主導方式はそのような頻繁にアクセスされるデータが存在するノードにスレッドが集中しやすいため，スレッド主導方式に劣る．

ただし，この2方式は排他的なものではなく双方が補間する形で実装することも可能である．例えば，通常はスレッド主導方式でスケジューリングを行ない，非常にアクセスの激しいデータ構造だけをあるノードに固定し，そのデータ構造にアクセスする時だけデータ主導方式のスケジューリングを行なう方法などが考えられる．

## 4 予備評価

Motorola MC68040を4個搭載し，Mach3.0マイクロカーネルが動作するマルチプロセッサワークステーションを用いて予備実験を行なった．このマシンのページサイズは8Kバイトである．各プロセッサを別々のプロセッサセットに割り当て，生成した分散タスク内の各タスクをそのプロセッサセット上で動作させた(図3)．分散タスク管理サーバと分散共有メモリ管理サーバは，集中式のサーバとして実装しており一つのノード上で動作する．分散共有メモリのメモリコンシステンシモデルとしては，Sequential Consistencyモデルを用いた．また，今回の予備評価では，上で述べた2レベル・スケジューリング方式は使用していない．

評価用のマルチスレッドプログラムとして，画素数1024×1024のフラクタル図形(マンデルブロ集合)

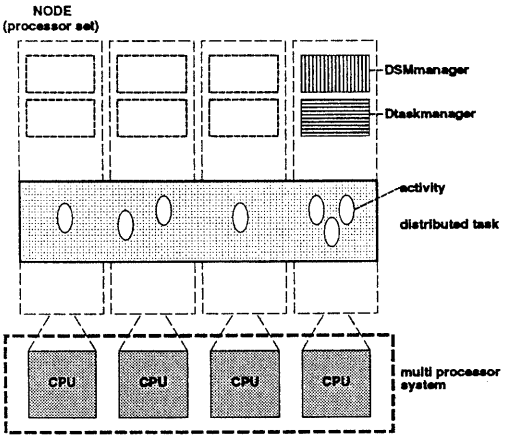


図 3: マルチプロセッサマシンを用いた予備実験システム

を各ラインを一つの処理単位としてスレッドに割り当てて計算するプログラムを次の 3 種類を作成した.

1. cthread ライブラリを用いて作成したマルチスレッドプログラム
2. 上のプログラムを dthread ライブラリに置き換えた分散スレッドプログラム
3. 分散共有メモリのページ単位を意識したデータ割り当てを行なった分散スレッド最適化プログラム

上記のプログラムの実行時間の測定結果を図 4 に示す. (a) はプログラム 1 をマルチプロセッサ環境で実行した時間, (b) はプログラム 2 を物理共有メモリ上の分散タスク/スレッドとして実行した時間, (c) はプログラム 2 を分散共有メモリ上の分散タスク/スレッドとして実行した時間, (d) は最適化したプログラム 3 を (c) と同じく分散共有メモリ上の分散タスク/スレッドとして実行した時間である. これから, (b), (d) ともに (a) の実行時間からのオーバヘッドは 5% 以内に収まり, 充分な処理能力を持つといえる. しかし, 分散共有メモリに対する最適化を行なわないプログラム 2 の分散スレッド環境下での実行時間は, 1 ノード上でプログラムを実行した場合の数倍である. これは, 分散共有メモリの同一ページに対するアクセスが発生して, ページスラッシングが起るためである.

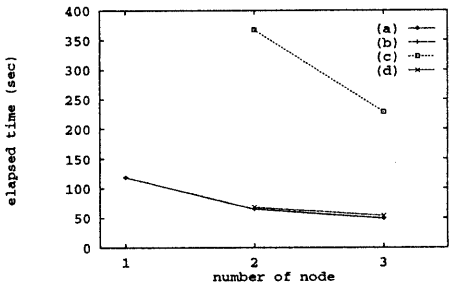


図 4: フラクタル図形描画の実行時間 (a):プログラム 1(マルチプロセッサ環境) (b):プログラム 2(物理共有メモリ) (c):プログラム 2(分散共有メモリ) (d):プログラム 3(分散共有メモリ)

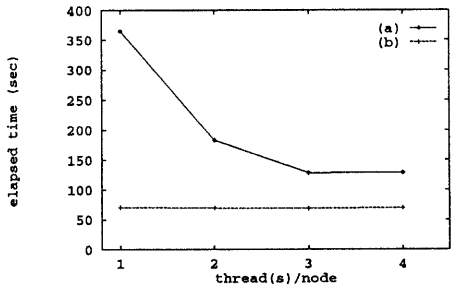


図 5: 各ノード上のスレッド数に対するフラクタル図形描画の実行時間 (2 ノード) (a):プログラム 2 (b):プログラム 3

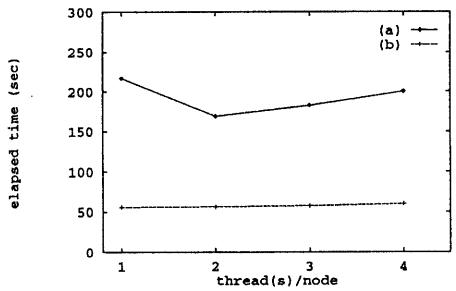


図 6: 各ノード上のスレッド数に対するフラクタル図形描画の実行時間 (3 ノード) (a):プログラム 2 (b):プログラム 3

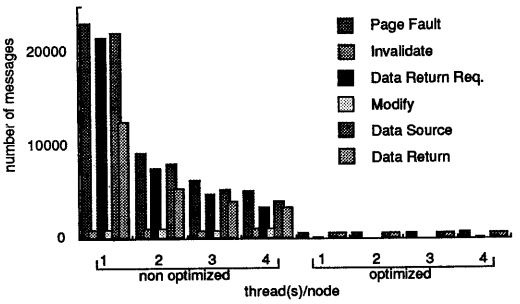


図 7: 各ノード上のスレッド数に対する種類別のメッセージ数 (2 ノード)

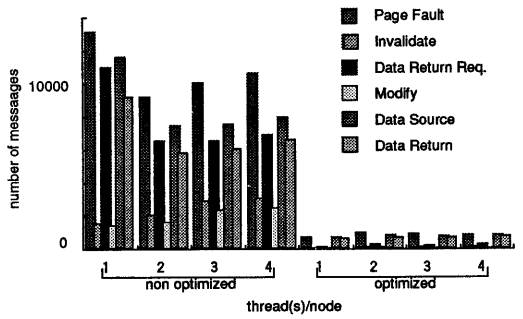


図 8: 各ノード上のスレッド数に対する種類別のメッセージ数 (3 ノード)

また，各ノード上で分散タスクに割り当てるカーネルスレッド数を変化した場合の計測結果を図 5 と図 6 に示す．また，これらの場合の種類別のメッセージ数を図 7 と図 8 に示す．2 ノードの場合，各ノード上で動作させるスレッド数が増えるに従い実行時間が減少した．評価に用いたプログラムは，主に画像を格納する配列への書込みにおいて各スレッドが競合する．そのため，各スレッドに対して 1 ライン (32bit 整数で 1024 画素) の処理を割り当てている場合，同一ページへのアクセスで競合するのは前後のラインを処理しているスレッドである．各スレッドに対して順番にラインを割り当てているために，各ノードで動作させるスレッドを増やすことでノード間のアクセス競合が減ることになる．

このことはメッセージ数からも読みとれ，図 7 において各ノードに 1 スレッドのみ動作させる場合には，膨大なページデータ供給とページデータ返却要求メッセージが出されている．これは，各ノード上


図 9: 各ノード上のスレッド数に対する実行時間の内訳 (2 ノード)

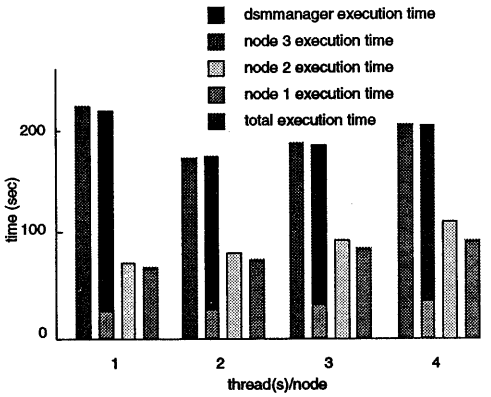


図 10: 各ノード上のスレッド数に対する実行時間の内訳 (3 ノード)

で動作するスレッドが同じページに対しての書込みをおこない，ページデータが行ったり来たりする典型的なページスラッシングが発生していることの裏付である．各ノードで動作させるスレッド数が増加するにつれて，このページデータ供給とページデータ返却要求のメッセージは減少していき，ページスラッシングの発生が押さえられている．ただし，最適化プログラムの場合と比較してメッセージ総数は非常に多く，ページスラッシングの発生は低下しているものの，充分に少ないというわけではない．3 ノードの場合も同様にページスラッシングが発生し多くのメッセージの転送が行なわれている．この場合，スレッドが増加するにつれて，スレッドライブラリ内等の同期変数，ロック変数などへのアクセスが増え，無効化要求のメッセージが増加してスラッ

表 1: 0.1 秒間のメッセージの個数とページ転送回数 (2 ノード)

| 各ノード上のスレッド数 | 1 | 4 |
|---|---|---|
| メッセージ個数 | 16.5 | 8.3 |
| ページ転送回数 | 9.8 | 4.7 |

シングが発生している.

分散共有メモリ上でプログラム 2 の実行時間の内訳を図 9 と図 10 に示す. どちらの場合も, 分散共有メモリ管理サーバが動作している時間が非常に長く, それが全体の実行時間を大きくしている原因である. ノード 2 やノード 3 での評価プログラムの実行時間を見ても, 分散共有メモリ管理サーバの処理がネックになり, スレッド数を増やすことでストールしている時間を見かけ上少なくするという効果はあまり見られない.

また, この評価プログラムが動作中の 3 秒間のメッセージ数の統計を 0.1 秒間隔で収集した. その結果を表 1 に示す. これはメッセージ数と, ページ移送を伴うメッセージ数の平均値を示している. これによると, ページスラッシングが多量に発生する場合, 0.1 秒間で約 10 ページの転送と約 7 回のメッセージの転送がネットワークの負荷として与えられることになる. これは, ページ転送だけで 6.1Mbps の帯域を要求する.

## 5 まとめ

分散共有メモリ上に分散したスレッドを実行する場合の課題を主としてページスラッシングの点から議論し, 分散共有メモリ上で分散タスク/スレッドのプログラム実行環境を提供する **PARSEC** の構成について述べた. また, **PARSEC** のスケジューリング方式として, 分散システム全体の負荷分散と, ページスラッシングの低減およびスレッドの並列性の保持を目的とする 2 レベル・スケジューリング方式を提案した. さらに, マルチプロセッサマシンによる分散スレッドの処理能力に対する予備評価を行なった. その結果, ネットワーク遅延を無視した評価にもかかわらずページスラッシングの影響が非常に大きいことや, 上手く最適化されたプログラムでは分散共有メモリのオーバヘッドは無視出来るレベルであることが分かった. 実際のアプリケーションプログラムは容易には最適化が出来ないと思われるので, 今後はページスラッシングを低減するローカルスケジューリングについて詳細設計を進め, 実用的なアプリケーションを用いてネットワーク遅延を考慮した評価を行なう. さらに, 最終的には **PARSEC** と共に動作するファイルシステムサーバや, I/O サーバの実装を行ない, それらと複数の分散タスクが動作した分散システム上で負荷分散を行なうグローバルスケジューラの評価と総合的な **PARSEC** の評価を行なう予定である.

## 参考文献


[1] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer Systems*, Vol.7, No.4, November 1989, pp.321–359.

[2] B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms," *Computer*, Vol.24, No.8, August 1991, pp.52–60.

[3] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocessor Programs," *IEEE transactions on Computers*, Vol.c-28, No.9, September 1979, pp.690–691.

[4] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, May 1990, pp.15–26.

[5] J. B. Carter, J. K. Bennette, and W. Zwaenepoel, "Implementation and Performance of Munin," In *Proceedings of the 13th ACM Symposium on Operating System Principles*, May 1990, pp.152–164.

[6] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon, "The Midway Distributed Shared Memory System, " Technical Report CMU-CS 93–119, Department of Computer Science, Carnegie-Mellon University.

[7] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory," In *Proceedings of 19th Annual International Symposium on Computer Architecture*, May 1992, pp.13–21.

[8] B. Fleisch and G. Popek, "Mirage: A Coherent Distributed Shared Memory Design," In *Proceedings of the 12th ACM Symposium on Operating System Principles*, December 1989, pp.211–223.

[9] 篠原 拓嗣, 藤川 賢治, 大久保 英嗣, 津田 孝夫, 「分散共有記憶に基づくオペレーティングシステム DM-1 の構成」, 情報処理学会研究会報告, 93-OS-61, pp.49–56.