2007－DSM－44 (7)
2007／3／9

通信量の密度に着目したS-DSM開発支援ツールS-CATの機能拡張

鈴木 祥[†] 坂口朋也[†] 吉瀬謙二[††]
片桐孝洋[†] 弓場敏嗣[†]

ソフトウェア分散共有メモリ (S-DSM) システムは，分散メモリ環境においてソフトウェアで仮想的な共有メモリを実現する．アプリケーションを記述するユーザに対して，共有メモリモデルの並列プログラミングインタフェースを与えるという利点を備えている．我々は，S-DSM システムのための開発支援ツール S-CAT の開発を行っている．S-CAT は，通常隠蔽されている S-DSM の実行履歴情報を視覚化する機能をもっている．本稿では，S-DSM 上で S-CAT を利用したアプリケーションの高速化を行い，現状のツールの改善点の検討を行った．得られた改善点から，通信量の密度を大域的に視覚化する機能の実装を行い，使用事例からその有用性を確認した．

Extension of S-DSM Development Assistance Tool S-CAT Focusing on Density of Communication Volume

Sho Suzuki,[†] Tomoya Sakaguchi,[†] Kenji Kise,[††]
Takahiro Katagiri[†] and Toshitsugu Yuba[†]

Software distributed shared memory (S-DSM) system achieves a virtual shared memory with software in the decentralized memory environment. It provides the user who describes the application with the advantage of giving the parallel programming interface of the shared memory model. We developed development assistance tool S-CAT for S-DSM system. S-CAT visualizes execution history information on S-DSM, which is usually concealed. In this paper, the application got speed up on S-DSM using S-CAT, and the improvement of a current tool was examined. The function to visualize the density of communication volue globally was implemented from the obtained improvement, and its effect was confirmed by use cases.

1. はじめに

近年，高性能計算のための環境として汎用の PC を用いたクラスタシステム (PC クラスタ) が普及してきている．クラスタシステム上でアプリケーションを実行する際，効率よく利用できる並列プログラミング環境が重要となる．

ソフトウェアで仮想的な共有メモリを実現するソフトウェア分散共有メモリ (Software-Distributed Shared Memory, S-DSM) は，分散メモリ環境において共有メモリモデルの並列プログラミングが可能であるという利点を備えており，いくつかのシステムが提案されている[1)2)]．また，より高速な S-DSM システムの実現を目指した研究も行われている[3)]．

S-DSM では，分散したメモリ間の一貫性をとるために通信を行う．通信は非同期に行われ，実行状況の把握は困難である．S-DSM 開発やアプリケーション評価を円滑に行うには，通信などの実行履歴の情報を表示する支援ツールが求められる．

我々は S-DSM 開発支援ツール S-CAT の開発を行っている[4)]．S-CAT は実行状況の詳細な情報を取得し，わかりやすい形で提示する開発支援ツールである．

本稿では，S-CAT を利用したアプリケーションの性能チューニングの事例からツールの改善点を検討した．また，得られた改善点からツールの機能拡張を行い，使用事例からその有用性を確認した．

本稿の構成を示す．2 章では，我々が開発している S-DSM 開発支援ツール S-CAT の説明を行う．3 章では，S-CAT を利用したアプリケーションの性能チューニングを行い，ツールの改善点の検討を行う．4 章では，アプリケーションの性能チューニング機能の拡張を行う．5 章では，実装した新機能を利用したアプリケーションの性能チューニングを行い，有用性の確認を行う．6 章で本稿をまとめる．

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems,
The University of Electro-Communications
†† 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

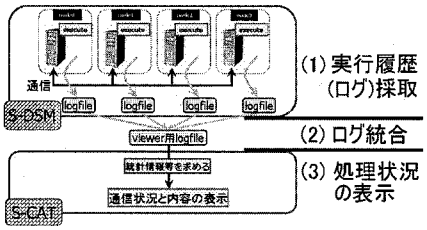

図 1 S-CAT の利用手順

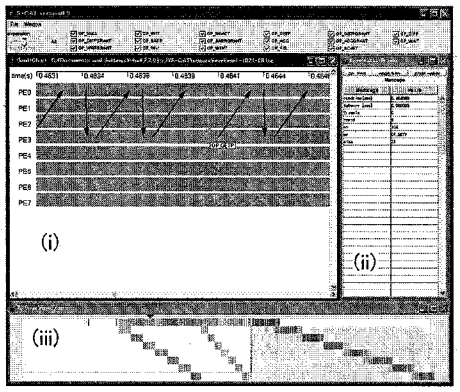

図 2 S-DSM 開発支援ツール S-CAT の画面写真

2. S-DSM 開発支援ツール S-CAT

本章では，我々が開発を行っている S-DSM 開発支援ツール S-CAT の利用手順と，S-CAT の機能について述べる．

2.1 S-CAT の利用手順

アプリケーション実行時における S-DSM に与える影響を抑えるため，ツールのために必要な処理は最小限に止めるべきである．従って，S-CAT のために実行時に行う処理は「情報の保存」のみとしている．実行後に，採取した実行履歴を保存したログファイルから S-CAT による処理状況の表示を行う．ログの保存に要する時間はアプリケーション実行時間の 0.1%程度であり，ツールの利用に伴う S-DSM への影響は無視できるものである．S-CAT の利用手順を**図 1** に示す．

(1) ログ採取 S-DSM 上でログを採取しながらアプリケーションを実行する．ログファイルはすべてのノードのローカルディスク領域に格納される．

(2) ログ統合 それぞれのログファイルには送信時刻と通信時間，およびメッセージの内容が記されている．それらのファイルを集め，1 つのファイルに統合する．

(3) 処理状況の表示 生成されたログファイルをもとに，オフラインで S-CAT による視覚化を行う．

表 1 実験環境

ノード数	16
CPU	Intel Pentium4 Xeon 2.8GHz × 2
OS	RedHat Linux 7.3
メモリ	1GB
ネットワーク	ギガビットイーサネット
コンパイラ	gcc version2.96
コンパイラオプション	-O2

2.2 S-CAT の機能

S-CAT の画面を**図 2** に示す．S-CAT は，主に以下の機能をもつ．

(i) 通信視覚化 (GanttChartFrame)
ユーザから見えない通信処理に焦点をあて，1 つのノード間通信を 1 つの矢印として表示する．これを時系列的に表示することにより処理の流れを視覚的に把握することができる．

(ii) 通信メッセージの内容表示 (InformationFrame)
送信時刻，通信時間，送信元ノード ID，送信先ノード ID，送信バイト数など，1 つのノード間通信に含まれるメッセージ内容を表示する．

(iii) 通信密度の表示 (AllOverViewFrame)
通信回数の密度を視覚的に表示することで，アプリケーション実行全体の大域的な通信状況を提示する．本稿では，このウィンドウを大域表示フレームと呼ぶこととする．

上述の機能以外に，ノードごとの通信回数や実行全体の通信量などの統計情報を表示する機能，使用している S-DSM の通信メッセージの定義ファイルを反映させる機能等が実装されている．

3. S-CAT を利用したアプリケーションの性能チューニング

3.1 実験環境とアプリケーション

本章では，S-CAT を利用して S-DSM 上のアプリケーションの高速化を行う．

実験に用いた PC クラスタの環境を**表 1** に示す．S-DSM システムは Mocha[5] を利用する．変更点として，S-CAT を利用するためのログを実行時に採取する関数を追加した．Mocha では，仮想共有メモリ空間上で最新のデータを保持するノードをホームノードと呼ぶ．ホームノードがもっているデータを他のノードが使用するときは，ホームノードからそのノードへデータを受け渡すことで仮想共有メモリを実現している．

対象とするアプリケーションは行列積 MM(Matrix Multiply) である．MM は JIAJIA Version 2.2[6] に添付されているものと同一のものである．行列サイズ 2048 x 2048 の double 型正方行列の行列積計算を行う．MM は，行列データを各ノードに行方向に分割して計算を担当させる．例えば 4 ノードで動作させる場

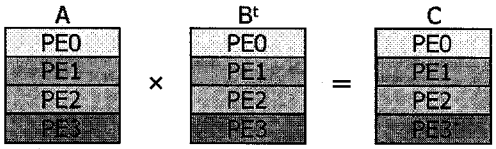

図 3 MM のデータ分割のモデル図

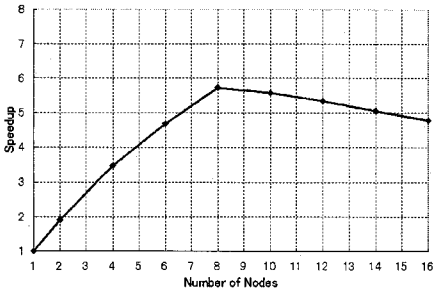

図 4 MM の速度向上率

合は，**図 3** のようになる．

上述の環境を用いて MM の動作速度を測定した結果のグラフを**図 4** に示す．横軸はノード数，縦軸は速度向上率で，1 ノードでの性能を 1 として正規化されている．図 4 に示すように，ノード数が 8 を超えるところで性能向上が得られておらず，性能の飽和がみられる．S-CAT を利用することで，その原因の解明と性能の向上を目指す．

3.2 S-CAT を利用した MM の高速化

今回注目した S-CAT の機能は，ウィンドウの 1 つである大域表示フレームに表示される通信密度である．MM を 8 ノードで実行し採取したログファイルを S-CAT で視覚化した際の大域表示フレームを**図 5** に示す．

図の縦軸はノード番号を表しており，上からノード番号の小さい順に表示されている．図の横軸は時刻に対応しており，左端と右端がそれぞれログファイルの採取開始から終了までに対応している．図中の色の濃い部分は通信が密な箇所，薄い部分は通信が疎な箇所を表している．

MM を 16 ノードで実行し採取したログファイルを S-CAT で視覚化した際の大域表示フレームを**図 6** に示す．16 ノードでの実行では，大域表示フレームに特徴的な形がみられた．

最も色の濃い部分が，図の左上部から右下部に向けて階段状に発生している．これは他のノードからのページ要求を受信し，ページ転送が行われている部分である．ページ要求を受けるホームノードが，時間の経過とともに移り変わっていく様子がみられる．

一方，図の右上部には全く色がついていない．これは通信の発生がないことを示している．つまり，ここ

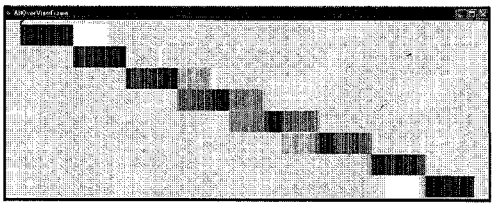

図 5 MM の大域表示フレーム．8 ノードで実行．

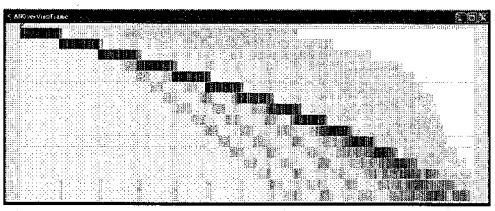

図 6 MM の大域表示フレーム．16 ノードで実行．

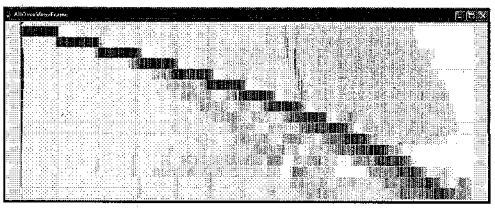

図 7 計算担当範囲を変更した MM の大域表示フレーム．16 ノードで実行．

ではノードは自身の実行すべき計算担当範囲を終了しているということになる．ノード番号の小さいノード，つまり図中で上に表示されているノードであるほど，計算の終了を速く終える傾向にある．ノード 0 では実行時間の 1/3 以上がアイドルの状態となっている．

ノード番号の小さいノードであるほど計算の終了を速く終えてアイドル状態にあるという傾向は，実行ノード数が 8 ノードを超えるところで見られる．図 5 に示すように，8 ノードまでの実行の場合，各ノードの計算終了時間はほぼ一定に揃っており，アイドル状態のノードの発生はみられない．

アプリケーション MM 全体の実行時間を短縮するためには，ノードがアイドル状態となっていることを極小化する必要がある．MM は，2048 行の行列計算をノード数で均等に分割して計算を行っている．自身が担当している計算範囲の終了が速いノードと遅いノードが存在することから，計算終了の速いノードに対する計算担当範囲を多く，遅いノードに対する計算担当範囲を少なく設定することで各ノードの計算時間の格差の軽減を図る．

新しい計算担当範囲を適用した MM を 16 ノードで実行し，採取したログファイルを S-CAT で視覚化した．その際の大域表示フレームを**図 7** に示す．通常の

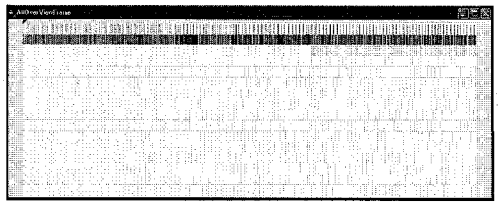

図 8　N-queens の大域表示フレーム．16 ノードで実行．

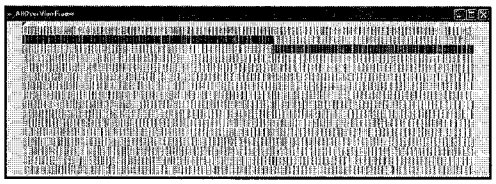

図 9　アルゴリズム変更後の N-queens の大域表示フレーム

MM の各ノード間の計算時間の格差は 3.28 秒であったが，計算担当範囲を変更した MM の各ノード間の計算時間の格差は 2.36 秒と，格段に軽減された．また，アプリケーション MM 全体の実行時間も通常の MM と比較して 8.3%の速度向上が得られた．

3.3　その他のアプリケーション

MM の他にもいくつかのアプリケーションについて同様の視覚化を行い，ノード間での実行時間の格差の発生等，大域表示フレームに特徴的な形の確認ができるか調査した．N-queens の世界記録を樹立したベンチマーク[7]において，特徴的な形がみられた．N-queens の処理状況を S-CAT を利用して視覚化した大域表示フレームを**図 8** に示す．N-queens はタスク分割型のアルゴリズムで，ロックを用いた排他制御を行っており，ロックを発行するノードへの通信集中がみられている．

そこで，ロックを発行するノードを変更し，通信の集中を回避するようアルゴリズムを変更した．アルゴリズム変更後の N-queens の大域表示フレームを**図 9** に示す．ロックに伴う通信集中が改善され，6.6%の性能向上が得られた．しかし，なおも通信集中が発生していることがわかる．これはロックに伴う通信集中の他にもう 1 つ，共有データ送受信による通信集中が発生していたためである．最も回数の密度の高かったロックの集中を改善したことで浮かび上がってきたのである．

4.　S-CAT の機能拡張

本章では，3 章で行った S-CAT を利用したアプリケーションの性能チューニングの事例から，S-CAT の改善点を検討し開発支援ツールとしての新しい機能の提案と実装を行う．

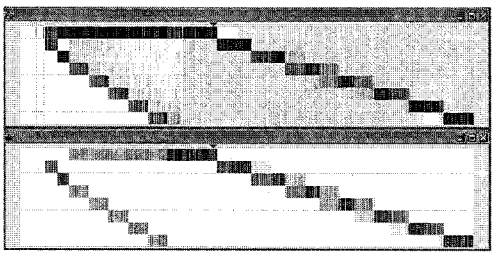

図 10　通信回数および通信量の密度の表示 (上段：通信回数の密度表示，下段：通信量の密度表示)

4.1　S-CAT の改善点

3 章で行った S-CAT を利用したアプリケーションの性能チューニングの事例から，現状のツールの問題点を検討し以下にまとめる．

・通信回数の密度の情報だけでは不十分

N-queens のように異なる 2 種類の通信集中の要因が存在している場合，ユーザがその 2 つを区別できるような視覚化を行って情報の提供をすることが望ましい．

・InformationFrame の有効活用法

3 章の事例では InformationFrame における統計情報の表示機能はあまり利用しなかった．アプリケーション実行全体での通信情報の統計表示機能の有効活用法の提案を考えたい．

4.2　通信量の密度の表示機能

従来の S-CAT は，通信回数の密度の表示機能をもっていた．しかし，通信回数と同様に通信量もアプリケーションの重要な実行情報である．通信量の観点から，アプリケーションの実行全体における密度を大域的に表示する機能は有用である．

そこで，通信回数の密度を表示する大域表示フレームに加えて，通信量の密度を表示するもう 1 つの大域表示フレームを実装した．**図 10** に 2 つの大域表示フレームを示す．上段は従来の S-CAT ももつ通信回数の密度表示機能，下段が今回実装した通信量の密度表示機能である．通信回数の密度表示機能と同様に，縦軸にノード番号，横軸に時間軸をとり，通信量の値に基づいて色の濃淡によって密度の視覚化を行う．ユーザは通信回数と通信量の 2 つの観点からアプリケーションの大域的な通信状況を把握することで性能チューニングの指針を得ることができる．

この他にも，アプリケーションのソースプログラムとの対応表示機能を実装した．アプリケーションのソースプログラム中に API を挿入することで，その時刻情報をログに記録し S-CAT のフレーム上に表示する．API を計算ループの前後などに挿入することで，任意の箇所を特定することができる．

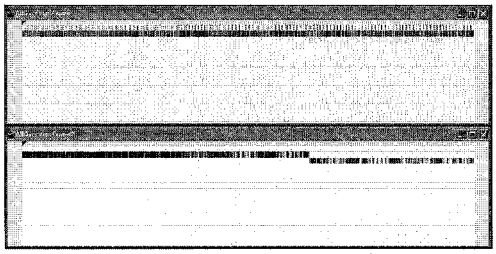

図 11 N-queens の新大域表示フレーム

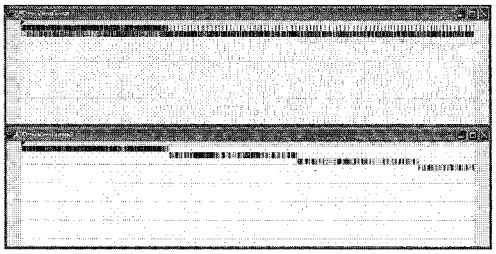

図 12 ページサイズ変更後の N-queens の新大域表示フレーム

5. 新機能の使用事例

本章では，新しく実装した機能を使用した事例を示しその有用性を検証する．従来の S-CAT における機能では視覚化されていなかった部分の情報を提供することで，より効率的な性能チューニング支援を実現した事例を示す．

5.1 実験環境とアプリケーション

実験には表 1 と同一の PC クラスタを使用した．対象とするアプリケーションは，3.3 節で使用した N-queens とする．3.3 節では，大域表示フレームに示される通信回数の密度に着目し，通信回数が最も多いロックの集中を回避することで性能チューニングを行った．これにより通信集中は改善され，ある程度の性能向上を得ることができた．

しかし，チューニング後の大域表示フレームをみると，もう 1 つの通信集中が発見された．つまり N-queens では 2 種類の通信集中が発生しており，それぞれが性能低下の要因であると考えられた．通信量の密度表示機能を用いることで，この 2 種類の通信集中を個別に表示させることができる．

5.2 N-queens の高速化

3.3 節において，ロックの集中を改善した後に発見されたもう 1 つの通信集中は，個々のタスクの実行済みフラグを格納している配列へのアクセスであった．通信回数の観点から言えば，ロックの通信回数のほうが多い．そのため通信回数の密度表示では，配列アクセスに伴う通信は比較的薄く表示されてしまっていた．しかし通信量の観点から言えば，配列アクセスの通信量はロックのそれよりもはるかに大きい．

以上の事柄は，従来の S-CAT では一度ロックの集中を改善しなければみえてこなかったことである．しかし通信量の密度表示機能を実装したことで，2 種類の通信集中が発生していることを最初から把握することができる．**図 11** に，N-queens の実行を視覚化した新大域表示フレームを示す．

通信量の集中が発生している原因は，Mocha の共有データ転送サイズに原因があった．Mocha ではデータの転送をページ単位で行っており，そのページサイズは 8kBytes 固定である．共有データをホームとして保有しているノードは，他のノードからデータの要求を受けるとそのデータを含む 1 ページを転送して返す．したがって，例えばある配列の 1 つの要素だけが必要な場合であっても，ページ単位，つまり 8kBytes でのデータ通信が発生することとなる．

MM のように，要求されているデータと一緒に送信された 8kBytes のデータが無駄なく使用されるアプリケーションの場合では，この手法は有効である．しかし N-queens でこのときに要求されているデータは，配列中のたった 1 つの要素である．しかもそのデータは読み取り専用ではなく順次上書きされていくため，一緒に送信されたおよそ 8kBytes のデータは破棄されることとなってしまう．

上述の理由から，N-queens では送受信されるデータに無駄が多く通信量の増加，集中につながり，性能低下の要因の 1 つとなっていると考えられる．そこで，1 回のデータ転送の単位の変更，つまりページサイズの変更を行うことで無駄なデータ送受信の削減を行う．

S-DSM Mocha の仕様ではページサイズは 8kBytes となっている．Mocha のページサイズを静的に 4kBytes〜64kBytes まで変更できる機能を実装し，その実行情報のログを採取できるよう改良を行った．

ページサイズを 4kBytes に変更してノード数 16 で実行した N-queens の実行状況を視覚化した S-CAT2 の画面写真を図 12 に示す．図 11 と比較して，下段，通信量の密度表示では変化が見られた．図 11 では 2 つのノードに色の濃い部分が見られていたが，図 12 では 4 つのノードに渡って色の濃い部分が見られている．N-queens において，配列データはページサイズ単位でノード ID 順にサイクリックにホームノードとして割り当てられている．ページサイズが 8kBytes の場合ではフラグ用の配列は 2 つのノードに配置されていた．ページサイズを 4kBytes にすることで，各ノードへの割り当てが半分の 4kBytes ずつとなる．そのため 4 つのノードに配置されることとなり，図 12 に示される通りのパターンとなる．このため通信回数の集中も分散されている．

GanttChartFrame にてデータ転送の矢印の 1 つを選択すると，InformationFrame にて表示されるデータ

転送のサイズは確かに 8kBytes から 4kBytes に変化していることが確認できた．また，実行全体での総送受信量もほぼ半減しており，無駄なデータの送受信が削減されたことが確認できた．これにより，性能チューニングの効果として N-queens の実行時間が 6.14[sec.](ページサイズ変更前，8kBytes) から 5.46[sec.](ページサイズ変更後，4kBytes) に減少し，11.1%の速度向上が得られた．

5.3 MM への適用

今回 N-queens で行ったページサイズの変更という手法は，行列積 MM についても有効であると考えられる．N-queens では大きなページサイズでデータ通信を行うと無駄が大きくなり，性能低下が発生していた．しかし MM のアルゴリズムでは，各ノードがホームとして保持している行列 B のデータをいずれはどのノードでも必要とするため，アプリケーション実行中には行列サイズに依存して一定量のデータ送受信が発生する．また，行列 B のデータは読み取り専用であり上書きされることがないほか，計算実行中にバリア同期が入ることもなく同時に送信されたデータを破棄しなければなくなるといったことも起こらない．以上の理由から，MM は大きなページサイズでのデータ通信を行っても無駄にならない特徴をもっている．

したがって，MM においては N-queens とは逆に，ページサイズを大きくすることで 1 回のデータ通信の粒度を大きくし，アプリケーション実行における全体の通信回数を削減することが可能であると考えられる．3 章で使用した行列積アプリケーション MM を，Mocha のページサイズを通常の 8kBytes から 64kBytes とし，ノード数 16 で実行した．

アプリケーション実行全体の統計情報を表示する InformationFrame で確認すると，1 回のデータ送信サイズは 64kBytes となっていることが確認できた．また，ページサイズを 8 倍としたことで，実行全体でのデータの送受信回数が 1/8 となっていることが確認できた．しかし，データの総送受信量はページサイズにかかわらず一定であった．以上のことは MM においてページサイズを大きくし 1 回のデータ送受信の粒度を大きくしても，データが無駄にならず有効に利用されていることの証拠でもある．

これにより，アプリケーションの実行も高速化された．ページサイズをデフォルトの 8kBytes としてノード数 16 で実行した場合では，ノード数 1 で実行した場合と比較して 4.99 倍の速度向上であった．それに対し，ページサイズを 64kBytes としてノード数 16 で実行した場合では，ノード数 1 で実行した場合と比較して 7.11 倍の速度向上が得られている．また，ページサイズ 8kBytes での実行と比較して，ページサイズ 64kBytes での実行では 29.7%の速度向上が得られた．

6. おわりに

本稿では，ソフトウェア DSM 開発支援ツール S-CAT の機能拡張および新機能を利用したアプリケーションの性能チューニングを行った．従来の S-CAT とは異なる性能チューニングのアプローチを示唆し，その有用性を確認できた．また，通信量の密度表示機能は，統計情報の表示機能と密接に結びついており，連携した活用法の提案もすることができたと考える．

今後の課題として，ノード毎の状態表示やページ毎の状態表示など，機能の充実が考えられる．アプリケーション実行中にリアルタイムでの視覚化という方針も考えられるが，実現には S-DSM の実行を任意の箇所で中断させる機構が必要となる．

謝辞 本研究の一部は，文部科学省科学研究費補助金 (課題番号 16300004 「スーパクラスタを指向した性能拡張性を持つソフトウェア分散共有記憶方式の研究」) の援助による．

参考文献

1) P.Keleher, S.Dwarkadas, A.L.Cox and W.Zwaenepoel: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. of the Winter 1994 USENIX Conference*, pp. 115–131 (1994).
2) 緑川博子, 飯塚肇: ユーザレベル・ソフトウエア分散共有メモリ SMS の設計と実装, 情報処理学会論文誌, Vol. 42, No. SIG9(HPS 3), pp. 170–190 (2001).
3) 坂口朋也, 鈴木祥, 今村昌之, 大島聡史, 片桐孝洋, 吉瀬謙二, 弓場敏嗣: 相乗り通信を利用したソフトウェア DSM の通信回数削減手法, 情報処理学会研究報告 2006-ARC-169, pp. 151–156 (2006).
4) 多忠行, 吉瀬謙二, 片桐孝洋, 弓場敏嗣: 複数の S-DSM を対象とする開発支援ツール S-CAT の設計と実装, 情報処理学会研究報告 2005-ARC-161, pp. 39–44 (2005).
5) 吉瀬謙二, 田邊浩志, 多忠行, 片桐孝洋, 本多弘樹, 弓場敏嗣: S-DSM システムにおけるページ要求時の受信通知を削減する方式, 先進的計算基盤システムシンポジウム SACSIS2005 論文集, pp. 349–358 (2005).
6) M. Rasit Eskicioglu, T. Anthony Marsland, Weiwu Hu and Weisong Shi: Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures, *the Hawaii International Conference on System Sciences(HICSS)* (1999).
7) 吉瀬謙二, 片桐孝洋, 本多弘樹, 弓場敏嗣: PC クラスタを用いた N-queens 問題の求解, 電子情報通信学会論文誌レター, Vol. J87-D-I, No. 12, pp. 1145–1148 (2004).