社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－167 (31)
2006－HPC－105 (31)
2006／3／1

ソフトウェアDSM開発支援ツールを利用したアプリケーションの高速化

鈴木 祥[†] 坂口朋也[†]
吉瀬謙二[†] 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，効率のよいS-DSMシステムMochaの開発，およびS-DSM上での高速なアプリケーションの構築を支援するツール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．そのため，新しいS-DSMの開発に際しても，性能の隘路の発見に利用できる．本稿では，Mochaの上でいくつかのアプリケーションを，S-CATを利用して高速化した結果について報告する．行列積の計算において，ノードごとに行列計算の担当範囲を変更することにより8.3%の速度向上を得ることができた．

Speedup of Applications using Software-DSM Development Supporting Tool

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

Software distributed shared memory (S-DSM) systems achieve a virtual shared memory without special hardware features. By using S-DSM, a parallel programming based on the shared memory model can be possible in distributed memory parallel computer environment. We are developing an efficient S-DSM, Mocha, and a tool, S-CAT, which supports development of new S-DSMs and applications on S-DSMs as well. S-CAT visualizes the execution history information of an application on an S-DSM. In this paper, we show that some benchmarks such as Matrix Multiply were speeded up by utilizing the S-CAT function, and the performance of Matrix Multiply was improved 8.3% by reallocation of the calculation range for each node.

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)3)]．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[4)5)]．

S-DSMでは仮想共有メモリを構築する際，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[6)]．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を用いて得られた実行履歴の情報を利用した性能チューニングを行うことで，アプリケーションの速度向上を試みた．

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章で今回の高速化の対象とするアプリケーションである行列

† 電気通信大学 大学院情報システム学研究科，東京都調布市
Graduate School of Information Systems,
The University of Electro-Communications,
Chofu, Tokyo

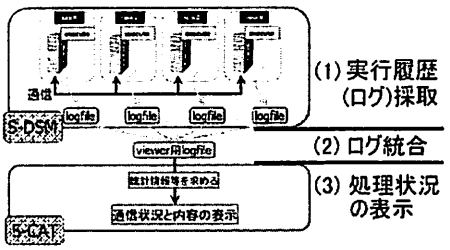

図 1 S-CAT の利用手順

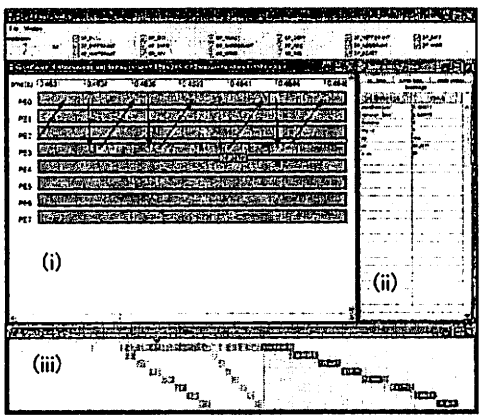

図 2 S-DSM 開発支援ツール S-CAT のスクリーンショット

積 (MM) の実行の様子をまとめ，4 章では S-CAT を使用したアプリケーションの高速化を試みる．5 章では MM 以外の様々なアプリケーションに関して議論する．6 章で本稿をまとめる．

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[6]．

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．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．

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 メモリ一貫性管理の中でユーザから見えない通信処理に焦点をあて，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 1 つの通信に含まれるメッセージ内容を表示する．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．

(iii) 通信密度の表示 通信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個々の通信バイト数等の統計情報を表示する機能，使用している S-DSM の通信メッセージを定義することができる機能等が実装されている[6]．

3. 実行環境とアプリケーション

本章では，今回の高速化の対象とするアプリケーションである行列積 MM の実行の様子をまとめる．

評価に用いた PC クラスタの環境を表 1 に示す．使用する S-DSM は Mocha Ver 0.2.1 である．Ver 0.2[7] からの変更点として，S-CAT を利用するためのログを実行時に採取する関数が追加されている．Mocha の起動時にオプション指定を行うことで，Mocha の初期化処理から終了処理までのすべてのログを採取することができる．また，ログ採取開始および終了の API が実装されている．それぞれをアプリケーション中の任意の箇所に挿入することで，必要とする実行部分のログをファイルに出力することができる．これを用いて，S-DSM の初期化および終了処理，また行列の初期化処理などの部分のログは省略し，行列積の計算を行っている部分のログのみを採取する．

Mocha では，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

表 1 評価環境

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

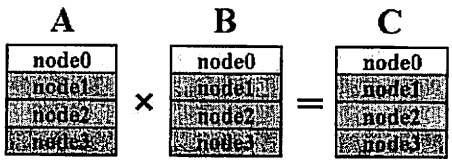

図 3 MM のデータ分割のモデル図

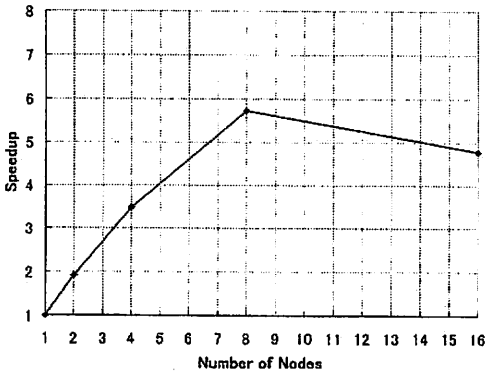

図 4 MM の速度向上率

現している．Mocha ではメモリ管理をページ単位で行っている．データの送受信は常に 1 ページ (Mocha デフォルトの設定で 8KBytes) ずつ行われる．

対象とするアプリケーションである行列積 MM(Matrix Multiply) は JIAJIA Version 2.2[2] 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．S-DSM を用いて PC クラスタ上で MM を動作させる場合，行列データを各ノードに分割して計算を担当させることが一般的である．例えば 4 ノードで動作させる場合は，図 3 のように行方向にブロック分割して実行する．JIAJIA 添付の MM も，これと同様の分割手法を用いてホームノードの設定を行っている．

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．

4. 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の表示機能である．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 5 に示す．

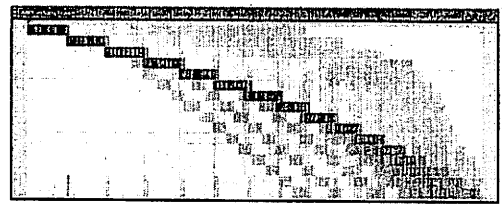

図 5 MM の大域表示フレーム，16 ノードで実行

表 2 16 ノードで実行した MM の各ノードの計算終了時刻

ノード ID	計算担当範囲 (行)	計算時間 (sec)
00	128	4.99
01	128	5.65
02	128	6.34
03	128	6.91
04	128	7.11
05	128	7.26
06	128	7.74
07	128	7.51
08	128	7.59
09	128	7.65
10	128	7.70
11	128	7.81
12	128	8.04
13	128	8.13
14	128	8.21
15	128	8.27

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 0 から 15 まで表示されている．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．

4.1 アイドル状態となっているノードの存在

今回用いたアプリケーション MM では，大域表示フレーム (図 5) 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ノード (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．

MM は，実行の最後にバリア同期をとる．計算の終了したノードは，バリアのサーバとなる 1 つのノード

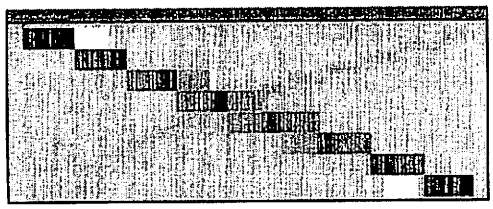

図 6 MM の大域表示フレーム，8 ノードで実行

へバリアの通信を行い，サーバノードは他のすべてのノードからの受信が完了すると，その旨を各ノードに返信し，バリア同期が完了する．各ノードのバリアの送信時刻が，そのノードでの計算終了の時刻を示しているとみてよい．また，ログの採取を開始する時刻 0 からすべてのノードは同時に計算を開始するため，各ノードの計算終了時刻はそのまま各ノードの計算時間と見てよい．

S-CAT の通信視覚化機能から，各ノードが送信しているバリアの通信の時刻を求めることでそれぞれの計算終了の時刻を調査した．

結果を表 2 にまとめる．表中において，ノード 15 に関しては，バリアのサーバとなるノードであるためにバリアの送信を行わない．そのためノード 15 に関しては，バリアの返信の通信時刻を計算終了の時刻，すなわち計算時間としている．

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顕著に見られる．8 ノードまでの場合にはこの傾向は見られない．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 6 に示す．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．

4.2 データの再分割による高速化

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．表 2 には各ノードの計算担当範囲 (行列の行数) も記載してある．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．16 ノードでの実行であれば，1 つのノードの計算担当範囲は 2048/16 = 128 行である．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．

表 2 で得られた各ノードの計算時間から，全 16 ノードの計算時間の総和は 116.91 秒となる．1 つのノードの平均計算時間は 116.91/16 = 7.31 秒と求まる．各ノードの計算時間と平均計算時間との比をとり，元の

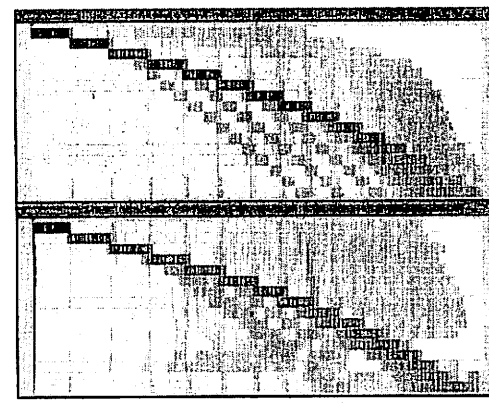

図 7 上：通常の MM の大域表示フレーム，16 ノードで実行 (図 5 と同一)
下：計算担当範囲を変更した MM の大域表示フレーム，16 ノードで実行

表 3 計算担当範囲を変更した MM の各ノードの計算終了時刻

ノード ID	計算担当範囲 (行)	計算時間 (sec)
00	188	5.22
01	164	5.30
02	148	4.99
03	136	5.76
04	128	6.09
05	128	6.20
06	124	6.61
07	124	6.96
08	124	7.18
09	124	7.21
10	112	7.34
11	112	7.22
12	112	7.23
13	108	7.43
14	108	7.55
15	108	7.58

計算担当範囲である 128 行から新しい計算担当範囲を導き出す．

ただし，ページ単位でデータの送受信を行う Mocha では，ページの端数の発生は望ましくない．そのため，上述の手法で導き出された新しい計算担当範囲の行数には微調整を施した．

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．上段に通常の MM の大域表示フレーム (図 5 と同一)，下段に計算担当範囲を変更した MM の大域表示フレームを示す．通常の 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，新しい計算担当範囲の行数と，その行数での

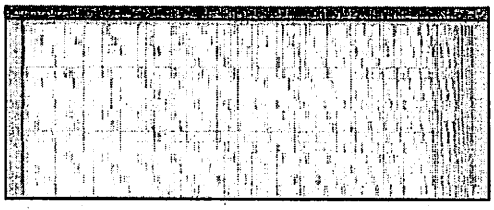

図 8 LU の大域表示フレーム，16 ノードで実行

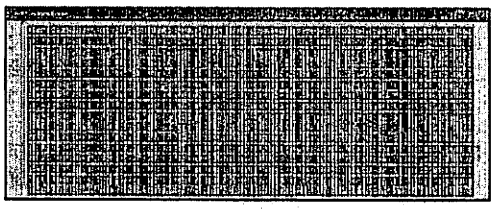

図 9 SOR の大域表示フレーム，16 ノードで実行

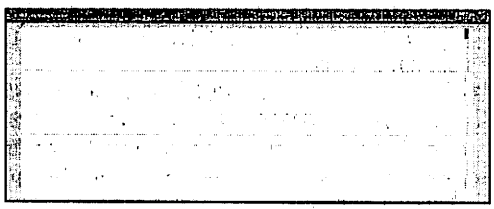

図 10 EP の大域表示フレーム，16 ノードで実行

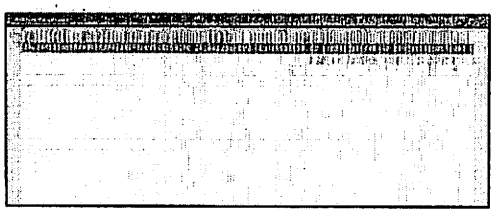

図 11 N-queens の大域表示フレーム，16 ノードで実行

分割を適用し実行した MM の各ノードの計算時間を表 3 にまとめる．図 7 にみられた傾向の通り，各ノードの計算終了時刻の格差は減少していることが確認できた．また，同時にノード 15 の計算時間も短縮されていることがわかる．アプリケーション MM 全体の実行時間は，通常の MM と比較しておよそ 8.3%の速度向上が得られた．

5. その他のアプリケーション

JIAJIA Version2.2 には MM の他にも，LU(parallel dense blocked LU factorization, no pivoting)，SOR(Red-Black Successive Over-Relaxation)，EP (The Embarrassingly Parallel) 等のアプリケーションが添付されている．これら 3 つのアプリケーションと，N-queens の世界記録を樹立したベンチマーク[8]

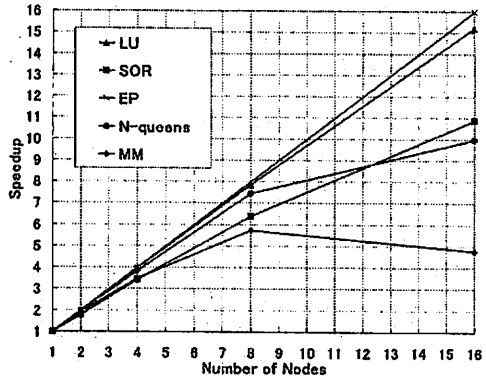

図 12 MM, N-queens, LU, SOR, EP の速度向上率 (MM は図 4 と同一)

を Mocha のために書き直したものの 4 つについてそれぞれ 16 ノードで実行し，処理状況をログファイルとして採取した．MM のような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．4 つのアプリケーションの処理状況を，S-CAT を利用して視覚化した大域表示フレームを図 8 から図 11 に示す．また，MM を含めた 5 つのアプリケーションの動作速度を測定した結果のグラフを図 12 に示す．

図 8 の LU の結果を議論する．LU は計算量と比較して通信量が少ない．そのため LU は実行ノード数に対して理想的な速度向上率が得られている．通信の密度も小さく，大域表示フレームは断続的に薄い色の表示がされている．LU において，各ノードの計算時間は一定であり，格差はみられなかった．

図 9 の SOR の結果を議論する．SOR の計算は，1 つの計算式を for ループで指定回数繰り返すものであるが，1 回の計算ごとにバリアをとってから次の計算に移る．大域表示フレームには，他のアプリケーションと比較して非常に濃い色が表示されている．SOR の実行で発生する通信の大部分はバリアに伴う通信であり，またバリアは非常に頻繁に発生するため，このような図が得られた．SOR も LU と同様，頻繁にバリアがとられるために実行ノード数の増加が実行時間の短縮に大きく影響し，1 ノードでの実行時間を 1 として 16 ノードではおよそ 12.2 倍の速度向上率が得られている．各ノードにおける計算時間に関しても，頻繁なバリアの発生により常に実行の進行が統一されており，ノード間の格差はみられなかった．

図 10 の EP の結果を議論する．EP はもともとほぼ完全な並列アプリケーションで，ノードごとに独立して並列に実行することができる．実行の最後で結果を統合するときにのみ，通信が発生している．図 10 では，実行の最後にわずかに通信の発生を示す色がついているのみである．そのため EP の実行ノード数に

対する速度向上率は，今回調査したアプリケーション中で最も高く，線形な結果が得られている．各ノードの計算時間はほぼ均一で，ノード間の格差はみられなかった．

図11のN-queensの結果を議論する．N-queensはタスク分割のアルゴリズムを用いており，マスターとワーカのノードが存在する．割り当てたタスクを終了したワーカノードに対してマスターノードが次々と新しいタスクを与えていく．そのため，各ノードの計算時間はほぼ均一化されておりノード間の格差はみられなかった．ただしN-queensに関しては，マスターとなるノードに通信の集中が発生していることが確認された．マスターノードには個々のワーカにタスクを分配していくための通信が集中することとなり，図中で色が濃く表示されている．マスターノードの通信集中を回避することで，N-queensは性能の向上が得られる可能性がある．

以上4つのアプリケーションに関して，S-CATを用いた処理状況の視覚化を行った．LU，SOR，EPに関しては，ノード間にMMのような特異な通信状況は確認できなかった．もともと16ノードでの実行においても8ノードと比較して速度向上が得られていたこともあり，大域表示フレームから速度向上の糸口を見つけるには至らなかった．N-queensに関しては，8ノードを超えるところでの速度向上があまり得られていない原因の1つがマスターノードへの通信の集中にある可能性が示唆された．

6. おわりに

本稿では，S-DSM上でアプリケーションMMを実行したときに，8ノードを超えるところでの性能向上が得られない原因の解明および性能の向上を目的としてS-DSM開発支援ツールS-CATを用いた．

S-CATによって視覚化された処理状況から，性能低下の1つの要因を特定した．得られた結果に即したチューニングを行うことで，アプリケーションの高速化を試み，8.3%の性能向上を達成することができた．

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) 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).

3) 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).

4) 松葉浩也，石川裕: 動的アクセスパターン解析によるソフトウェア分散共有メモリ，情報処理学会論文誌，Vol. 45, No. SIG11(ACS 7), pp. 1–13 (2004).

5) 坂口朋也: 通信粒度予測機構をもつソフトウェア分散共有メモリ，卒業論文，電気通信大学 電気通信学部情報工学科 (2005).

6) 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).

7) 吉瀬謙二，田邊弘志，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).

8) 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).