ハイパフォーマンス
コンピューティング 73−13
（1998. 10. 9）

Myrinetを用いた分散共有メモリシステムの評価

原田 浩[†] 手塚宏史[†] 堀 敦史[†]
住元真司[†] 高橋俊行[†] 石川 裕[†]

既存のUnixオペレーティングシステムと低通信遅延かつ高通信バンド幅を有するネットワークシステム上にSCASHと呼ぶソフトウェア分散共有メモリシステムを開発している。ソフトウェア分散共有メモリシステムの性能評価はこれまでにも多数行われてきたが、そのほとんどがノード数が16台以下のクラスタ上での性能評価であった。64ノードまでの台数規模で、SPLASH2のLU、FFT、Barnes-Hutプログラムを実行し、SCASHの評価を行った。その結果、ノード数に比例して実行時間に占めるSCASHの実行時間の割合も増加するため、充分な台数効果を獲得できないことが判った。

Evaluation of Software Distributed Shared Memory System on Myrinet

HIROSHI HARADA ,[†] HIROSHI TEZUKA ,[†] ATSUSHI HORI ,[†]
SHINJI SUMIMOTO ,[†] TOSHIYUKI TAKAHASHI [†]
and YUTAKA ISHIKAWA[†]

We have been developping a software DSM system called SCASH using low latency and high bandwidth network on standard OS. Although many evaluations on software DSM systems have been reported, most of DSM systems are evaluated on 16 nodes cluster. We evaluated the SCASH system using LU, FFT, and Barnes-Hut from SPLASH2 suites on upto 64 nodes in the cluster. We could not get scalable speedup because of the overhead of SCASH processing increases along with the increases number of nodes.

1. はじめに

我々はギガビットネットワークの一つであるMyrinet[1]上に低通信遅延かつ高通信帯域幅を提供する高速通信ライブラリPM[2),3)]を用いて、オペレーティングシステムのメモリ管理機能を利用したSCASHと呼ぶ分散共有メモリを実現している。使用しているプラットフォームは128台のIntel PentiumProプロセッサ(200MHz)とMyricom Myrinetネットワークから構成されるPCクラスタで、各ノードプロセッサのカーネルとしてLinux[4)]を使用している。SCASHは、一貫性モデルとしてERC(Eager Release Consistency)[8),9)]を採用し、その実装にはマルチプルライタプロトコル[10)]を用いた。さらに、ページ単位の一貫性維持プロトコルとして、ページの無効化を通知する無効化プロトコルと、ページデータを送信し、ページの更新を通知する更新プロトコルの双方を実装し、実行時に選択できるようにした。近年、NOW[13)]、Beowulf[14)]プロジェクト等に代表されるように多数のPC、ワークステーションを高速ネットワークで結合することにより、コストパフォーマンスに優れた、大規模かつ高性能なクラスタシステムを構築する研究が盛んに行われている。しかしノード数が16台以上のクラスタシステム上でのソフトウェア分散共有メモリシステムの評価はこれまで殆んど行われていない。本研究の目的は、ノード数が16台以上の大規模クラスタシステム上で、実用的アプリケーションを用いてソフトウェア分散共有メモリシステムを評価し、ソフトウェア分散共有メモリシステムの問題点を明らかにすることである。本論文の構成は以下の通りである。第2章では、SCASHの設計方針とAPIに関して述べる。第3章ではSCASH上で、SPLASH2[11)]のLU、FFT、及びBarnes-Hutを動作させ、実行時間とソフトウエアオーバヘッドを詳細に計測する。第4章では計測結果を基に、実装方式ならびにSCASHの問題点について検討する。第5章で関連研究を紹介し、第6章でまとめと今後の課題について述べる。

2. SCASHの概要

2.1 設 計

本研究の目的は、弱い一貫性モデルを実現したソフトウエアによる分散共有メモリ（以下DSMと呼ぶ）

† 新情報処理開発機構
Real World Computing Partnership

システム上で、実用的アプリケーションを動作させた場合の性能を評価し、DSM システムのオーバヘッドがどの程度顕在化するかを検証することにある。そこで SCASH は、次のような DSM システムとすることにした。

- ユーザレベルでの実現。
  SCASH は移植性を考慮し、オペレーティングシステムカーネルの提供するメモリ管理機構を用いて、ユーザレベルによって DSM を実現している。ユーザレベルではページ単位でプロテクションモードを変更し、例外処理ルーチンで一貫性処理を実現する。例外処理ルーチンはランタイムライブラリとして提供されユーザプログラムとともにリンクされる。
- 一貫性モデルとして ERC の実装。
  代表的な制限の弱い一貫性モデルとして、LRC、ERC 等があげられる。SCASH は PM のゼロコピー通信機能[12]を用いることにより、転送先ノードの計算を中断することなく更新データを低コストで、各ノードへコピーできる。PM のゼロコピー通信機能を用いることにより、ERC モデルを単純なプロトコルで効率的に実装することが可能であるため、SCASH は ERC モデルを採用することにした。
- マルチプルライタプロトコルの実装。
  シングルライタプロトコルでは、ページのフォールスシェアリングによる性能劣化が予想されるため、SCASH はマルチプルライタプロトコルを採用することにした。
- メッセージ通信のカーネルオーバーヘッドの削減。
  ページ転送要求、無効化要求等の制御メッセージはカーネルの提供する非同期通信機構を使わず、アプリケーションからメッセージの到着をポーリングすることにした。メッセージをポーリング処理することによって、非同期通信のためのカーネルオーバヘッドを削減した。

2.2 API

以上を踏まえ、SCASH は、以下の API を提供している。

- scash_init
  SCASH の初期化を行う。確保可能な共有メモリの大きさ、共有メモリが確保されるメモリ番地等を指定できる。
- scash_malloc、pdsm_free
  共有メモリの確保、開放を行う。
- scash_barrier
  全ノード間でバリア同期を取る。共有メモリデータへの全ての更新が、全ノードプロセッサ上に反映される。章 2.3で後述する通り SCASH は、無効化プロトコルと更新プロトコルの 2 つのページ更新プロトコルを選択することができる。
- scash_lock、scash_unlock
  ロックの確保と開放を行う。各ノード上に分散されたロックキューにより実現している。
- scash_update
  指定した共有メモリ領域中のデータ更新を全てのノード上に反映させる。但し、データの反映は、次の scash_barrier 又は scash_unlock まで遅延される。
- scash_polling
  ページ転送要求、無効化要求等の制御メッセージをポーリングし、必要に応じてメッセージを処理する。

2.3 ページ更新プロトコル

第 2.2章で述べたように、SCASH のバリア同期は、以下の 2 種類のページ更新プロトコルを選択することができる。

- 無効化プロトコル
  無効化プロトコルでは、更新が行われたページを共有しているノードに対して、ページ無効化メッセージを送信する。ページ無効化メッセージを受信したノードは、該当ページのプロテクションをを read/write 禁止に変更する。該当ページに対してアクセスを行うと、ページ例外が起き、ページ例外処理ルーチンが起動される。ページ例外処理ルーチンは、該当ページをコピーする。
- 更新プロトコル
  更新プロトコルでは、更新が行われたページを共有しているノードに対して、ページの最新データをゼロコピーし、ページ更新メッセージを送信する。ページ更新メッセージを受信したノードは、該当ページのプロテクションをを read 可に変更する。該当ページに対して write アクセスを行うと、ページ例外が起き、ページ例外ハンドラへ処理が移行する。ページ例外ハンドラは、該当ページの複製を作成し、プロテクションをを read/write 可に変更する。

3. 評　　価

本節では、SPLASH2 の LU(CONT)、FFT、BARNES-HUT(以下 BH と記す) を用いて SCASH の性能とオーバーヘッドの詳細を測定する。測定に用いた問題サイズは表 1の通りである。3 つのプログラム共、更新プロトコルと無効化プロトコルの 2 種類のプロトコルを用いた場合の実行時間とオーバーヘッドを測定した。

表 1　問題サイズ

LU	1024x1024 Matrix, 64x64 Block
FFT	65536 Complex Doubles, 16 Byte line size
BH	16384 nbody , dtime 0.02500, tstop 0.075

LU、FFTはノード数を1台から64台、BHについては1台から32台で動作させた場合の実行時間とオーバーヘッドを測定した。ノードが1台の実行時間は、SPLASH2のnullmacroを用いた実行時間である。プログラムの測定は全て10回行い、平均値を採用した。測定に用いたプログラムは全て、ページのホームノードを移動している。SCASHのオーバーヘッドは、バリア、ロック、アンロック等アプリケーションから呼び出されるランタイムライブラリのオーバーヘッドと、ページ例外処理ルーチンのオーバーヘッドであり、以下の通りにまとめる事ができる。

- バリア
  バリア同期のためのオーバーヘッド。scash_barrierの実行時間である。本章の図ではBarrierで示す。
- ページ例外処理
  ページ例外処理ルーチンのためのオーバーヘッド。今回の測定では、ページ例外処理ルーチンが呼び出されてから、終了するまでの時間を測定した。今回の測定では、ページ例外が検出されてから、ページ例外処理ルーチンが呼び出されるまでのオーバーヘッドは測定していない。本章の図ではExceptionで示す。
- ロックの獲得、解放
  scash_lock、scash_unlockに要する時間。scash_unlockは、scash_updateによる、データ更新の完了を待つ時間を含んでいる。本章の図ではLock、Unlockで示す。
- データの更新
  scash_updateに要する時間。scash_overheadによるデータ更新は、次のscash_unlock又は、scash_barrierまで遅延される場合があるので、データを実際に更新するオーバーヘッドは含まれない。本章の図ではUpdateで示す。
- メッセージのポーリング処理
  scash_pollingに要する時間。本章の図ではPollingで示す。

本論文では以上の全てのオーバーヘッドについて実行回数と実行時間を測定した。オーバーヘッドの測定には、低コストで正確な値を得るために、PentiumProプロセッサのタイムスタンプレジスタを用いた。測定は全て我々が開発したPCクラスタ上で行った。PCクラスタの主な仕様を表2に示す。Myrinet用通信ライブラリPMの最小遅延時間と最大帯域幅はそれぞれ、7.2μsec、117.6 MBytes/secである。

3.1 LU

LUの実行時間とSCASHのオーバーヘッドを図1及び図2に示す。LUはバリア同期のみで記述されているので、バリアとページ例外処理以外のオーバーヘッドは存在しない。

3.2 FFT

FFTの実行時間とSCASHのオーバーヘッドを図3及び図4に示す。FFTは、バリア同期で記述されているので、バリアとページ例外処理以外のオーバーヘッドは存在しない。FFTは行列の転置を行っているが、行列の転置に要した時間を表3及び表4に示す。

表2 PCクラスタの主な仕様

# of Node	128
CPU	Intel PentiumPro 200Mhz
Cache	512KBytes
Chipset	440FX
Memory	EDO 256MBytes/Node
Network	Myrinet 1.28GBits/sec

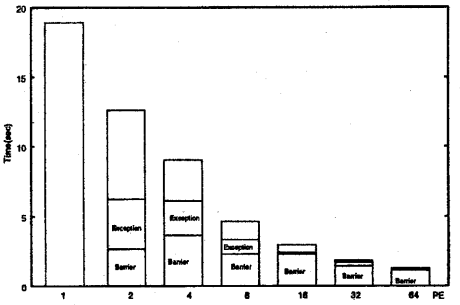

図1 LU(無効化プロトコル)の実行時間とオーバヘッド

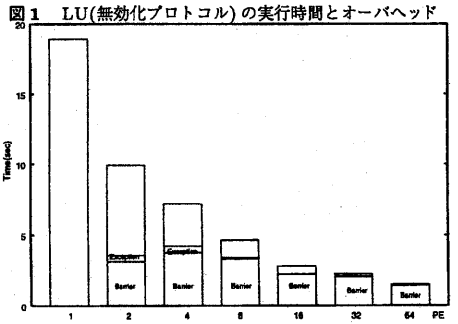

図2 LU(更新プロトコル)の実行時間とオーバヘッド

表3 FFT(無効化プロトコル)の転置時間

Node	Transpose(sec)
1	0.0809
2	0.159
4	0.152
8	0.171
16	0.206
32	0.315
64	0.317

3.3 BARNES-HUT

BHは粒子間の相互作用を計算するN体問題を解くプログラムである。BHはトータル時間の他に、粒子の木構造を構成する時間(Tree Build)と、粒子間の相互作用を求める時間(Force Calc)も測定した。以上の2つの実行時間を表5及び表6に示す。BHの実行時間とSCASHのオーバーヘッドを図5及び図6に示す。

表4　FFT(更新プロトコル) の転置時間

Node	Transpose(sec)
1	0.0809
2	0.0860
4	0.0671
8	0.0617
16	0.0613
32	0.0999
64	0.102

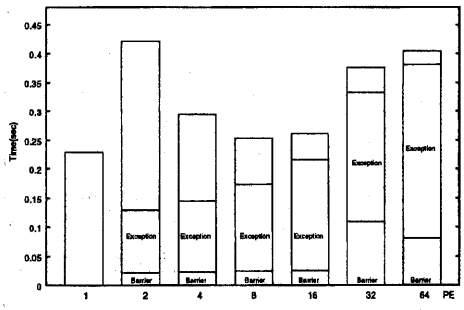

図3　FFT(無効化プロトコル) の実行時間とオーバヘッド

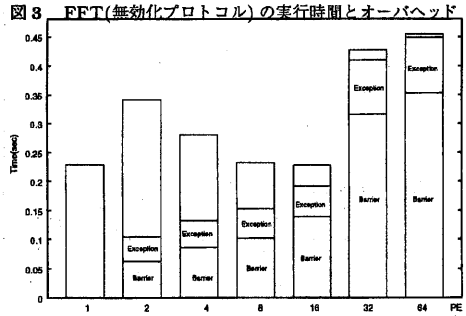

図4　FFT(更新プロトコル) の実行時間とオーバヘッド

BH は、バリア同期だけではなく、ロック、アンロック、ロック中のデータ更新を行っている。そのため、オーバーヘッドは、バリア同期、ページ例外処理の他に、ロック、アンロック、データの更新、メッセージのポーリング処理の全てのオーバーヘッドが存在する。図中ではこれらのオーバーヘッドの実行時間はそれぞれ下から順に、Barrier、Exception、Lock、Unlock、Update、Polling という名前の箱で示される。ロック／アンロック、データの更新、メッセージのポーリング処理の3つのオーバーヘッドの実行回数を表7及び表8に示す。

4. 検　　討

測定した実行時間から求めた、各プログラムの台数効果を図7に示す。LU、BH は台数効果が得られているが、FFT は、無効化プロトコル、更新プロトコルいずれのページ一貫性維持プロトコルを用いた場合も、台数効果を得られないことが判った。

表5　BH(無効化プロトコル) の実行時間

Node	Tree Build (sec)	Force Calc (sec)
1	0.665	30.7
2	4.93	14.2
4	4.92	7.38
8	4.77	3.84
16	4.56	2.09
32	4.47	1.22

表6　BH(更新プロトコル) の実行時間

Node	Tree Build (sec)	Force Calc (sec)
1	0.665	30.7
2	5.19	14.2
4	5.82	7.29
8	6.20	3.65
16	6.62	1.92
32	7.05	1.05

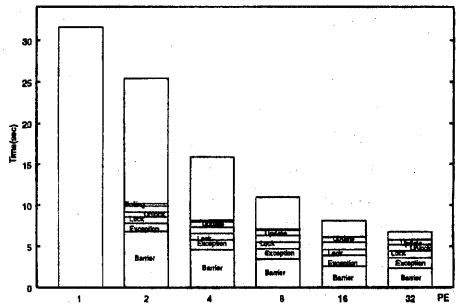

図5　BH(無効化プロトコル) の実行時間とオーバヘッド

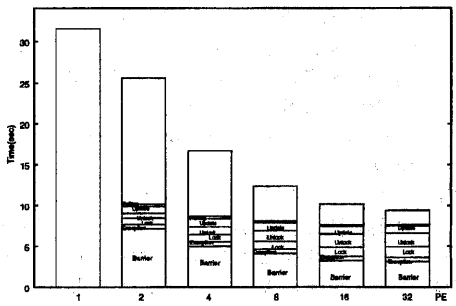

図6　BH(更新プロトコル) の実行時間とオーバヘッド

図2及び図1から LU は更新プロトコル、無効化プロトコルいずれのプロトコルにおいても、ノード数の増加に従い、バリアオーバーヘッドの割合が増加していく傾向にあることが判る。64 ノードで実行した場合、バリアのオーバーヘッドは、更新プロトコル、無効化プロトコルとも実行時間の 80 %以上を占めており、台数効果を妨げる大きな原因になっている。LU の性能を改善するには、バリア同期のコストを削減する必要がある。無効化プロトコルと更新プロトコルを

表7　BH(無効化プロトコル)のオーバーヘッド実行回数

Node	Lock/Unlock	Update	Polling
2	34479	142220	441254
4	16967	69694	265815
8	8951	37162	185107
16	4829	19506	145922
32	2411	10047	138043

表8　BH(更新プロトコル)のオーバーヘッド実行回数

Node	Lock/Unlock	Update	Polling
2	34471	142045	623957
4	16968	69869	541594
8	8952	37187	555631
16	4834	19764	614568
32	2399	9707	720332

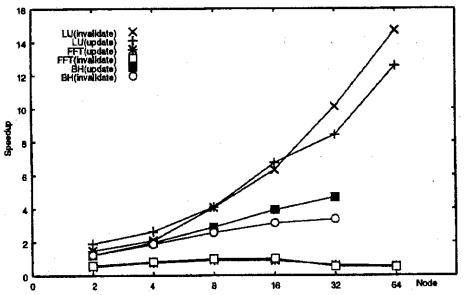

図7　台数効果

比較すると、ノード数の増加に従い、無効化プロトコルの方が、大きな台数効果を獲得している一方、更新プロトコルは、実行時間に占めるバリア同期の割合が増加していることが判る。

図3及び図4から、FFTは更新プロトコルでは、バリア同期のオーバーヘッドが大きく、無効化プロトコルでは、ページ例外処理のオーバーヘッドの占める割合が大きいことが判る。更新プロトコルは、バリア実行時にページの内容を更新するため、バリア同期の実行時間は増えるが、読み込みアクセスによるページ例外が発生しないため、ページ例外処理の実行時間は小さくなる傾向にある。無効化プロトコルは、バリア実行時に、ページを更新せず、無効化を通知するだけなので、バリアの実行時間は小さくなるが、読み込みアクセスによるページ例外が起きるため、ページ例外処理の実行時間が増える傾向がある。特にFFTでは行列の転置を行っているため、上記の特徴が顕著に表れたと考えられる。

BHは、粒子間相互作用の計算時間に関しては、ほぼリニアな台数効果が得られているにも拘らず、粒子の木構造の作成に要する実行時間が、ノード数2以上の時に増大している。そのためBHは32ノードで4.67倍の台数効果を得るに留まっている。メッセージのポーリング処理に要するオーバーヘッドは、ノード数、ページ更新プロトコルに関係なく1.7％以下でほぼ一定であり、台数効果を得る妨げにはなっていないことが判る。バリアのオーバーヘッドが実行時間に占める割合は、ノード数の増加に比例して、微増する傾向にあるが、ノード数に拘りなく実行時間の約30％を占めており、最も大きなオーバーヘッドとなっている。ロック、アンロック、データ更新の実行回数は、ノード数に反比例して減少しているにも拘らず、ロック、アンロック、データ更新、ページ例外処理に要するオーバーヘッドは、ノード数の増加と共に、微増する傾向にあり、台数効果を得る上での大きな障害となっていることが判る。

5. 関連研究

ソフトウェアによるDSMシステムの性能評価はこれまでにも多数行われている。しかし、ノード数が16台を超えるような、大規模なクラスタシステム上での性能評価は皆無であると言ってよい。Myrinetを用いたDSMシステムの性能評価としては、Zhou[15]がある。上記論文では、複数のメモリ一貫性モデルと、複数のページサイズの粒度についてSPLASH2を用いた性能評価を行っているが、16ノードまでの性能評価しか行われていない。また、ページへのアクセスチェックには専用のハードウェア[16]を用いている。Keleher[17]は、シングルライタとSCによるDSMの実装とシングルライタとLRCによる性能の比較評価を行っている。Keleher[18]は、TreadMarkの性能評価を行っているが、8ノードまでの性能評価しか行われていない。

6. まとめと課題

本論文は、最新の高速ネットワーク技術を用いて、制限の弱い一貫性モデルを実現したソフトウエア分散共有メモリシステム上で、SPLASH2のLU、FFT、BHの3つのアプリケーションを用いてソフトウエア分散共有メモリシステムの性能を評価した。SPLASH2の3つのアプリケーションの内、LU、BHについては、台数効果を得られる事が判った。しかし、ノード数が64台又は32台で、ソフトウェア分散共有メモリシステムのためのオーバーヘッドが実行時間の大部分を占めてしまうため、台数効果が頭打ちになってしまうことが判った。FFTに関しては、無効化プロトコル、更新プロトコルどちらのページ一貫性維持プロトコルを用いても、メモリの一貫性維持のためのオーバーヘッドが大きく全く台数効果を得ることが出来ないことが判った。以上から、SCASHはノード数が16台を超えるようなクラスタシステムの性能を活用できないことが判った。

大規模クラスタシステムを高効率で活用するためには、多数のノードプロセッサを用いた場合でも、大きな台数効果を獲得できる分散共有メモリシステムを開発することが重要であり、今後の大きな課題となる。

その為には、

- バリア、ロック／アンロック等同期プリミティブのハードウェア化
- マルチスレッディングによる実行の効率化
- コンパイラによる共有メモリ操作の最適化
- ホームノードの最適化

などの手法を用いる事が考えられる。

謝　辞

本研究における実装、評価に関して貴重な御意見を頂いた新情報処理開発機構工藤 知宏氏、山本 淳二氏に感謝いたします。

参 考 文 献

1) http://www.myri.com.

2) Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking '97*, 1997.

3) 手塚, 堀, O'Carroll, 原田, 石川. ピンダウンキャッシュを用いたユーザレベルゼロコピー通信. 情報処理学会研究報告. 情報処理学会, August 1997.

4) http://www.linux.org.

5) 原田 浩, 手塚 宏史, 堀 敦史, 石川 裕. Myrinetにおける分散共有メモリシステムの実装と評価. 情報処理学会コンピュータシステムシンポジウム, pp. 109–116. 情報処理学会, November 1997.

6) P. Keleher S. Dwarkadas A.L. Cox and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proceedings of the Winter 94 Usenix Conference*, pp. 115–131, January 1994.

7) A.L. Keleher P. Cox and Zwaenepoel W. Lazy release consistency for software distributed shared memory. In *In Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA '92)*, pp. 13–21, May 1992.

8) 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 Symposium on Computer Archtecture*, pp. 15–26, May 1990.

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

10) Amza C. Cox A. L. Dwarkadas, S. and Zwaenepoel W. Software DSM Protocols that Adapt between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pp. 261–271, February 1997.

11) Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *In Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36. ACM, June 1995.

12) Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *IPPS'98*, April 1998.

13) http://now.cs.berkeley.edu/.

14) T. Sterling, D. J. Becker, D. Savarese, M. R. Berry, C Reschke. "Achieving a Balanced Low-Cost Architecture for Mass Strage Management through Multiple Fast Ehternet Channels on the Beowulf Parallel Workstation". In *Proceedings of the 10th International Parallel Processing Symposium*, April 1996.

15) Y. Zhou, L. Iftode, J.P. Singh and K. Li, B.R. Toonen, I. Schoinas, M.D. Hill and D.A. Wood. Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation. In *Proceedings of the 6th ACM Symposium on Principles and Practice of Parallel Programming*, June 1997.

16) Robert W. Pfile. Typhoon-Zero Implementation: The Vortex Module Technical Report. Technical report, Wisconsin University, CS department, 1995.

17) P.J. Keleher. The relative importance of concurrent writers and weak concsistency models. In *Proceedings of the IEEE COMPCON'96 Conference*, February 1996.

18) Keleher, P., Cox, A. L., Dwarkadas, S., and Zwaenepoel, W. "An Evaluation of Software-Based Release Consistent Protocols". *Journal of Parallel and Distributed Computing*, Vol. 29, No. 2, September 1995.