高速通信機構を用いたソフトウェアDSMのパフォーマンス解析

バルリ ニコ デムス　　渡辺 正泰　　坂井 修一　　田中 英彦
東京大学 工学系研究科

概要

近年、高速ネットワークの研究が進んでおり、低オーバヘッド高バンド幅のネットワークが開発されている。このようなネットワークを用いると、ノード間の通信がボトルネックとなるソフトウェアDSMの性能を大きく向上できる可能性がある。

本稿では低オーバヘッド通信機構とメモリマップ通信機構に着目して、このような通信機構をページベースソフトウェアDSMに用いるときにどれぐらい性能向上が得られるかを定量的に解析した。解析した結果、100 Mbps イーサネットに比べると共有メモリオーバヘッドが 10 % 〜 49 % 削減されたが FFT や Ocean のような、メモリアクセスパターンによってページフォルトが頻繁に起こるようなアプリケーションでは充分な高速化は得られなかった。

Performance Analysis on Software DSM System Connected with High Speed Network

Niko Demus Barli　Masahiro Watanabe　Shuichi Sakai　Hidehiko Tanaka
Graduate School of Engineering, University of Tokyo

Abstract

Recent researches on High Speed Network have resulted in significant improvement on the performance of network. There are now many low overhead, high bandwith networks available. By using these high speed networks, there is possibility that we can greatly improve performance of software DSM system, whose bottleneck is in communication overhead between nodes.

In this paper, we put our attention on low overhead communication mechanism and memory mapped communication mechanism, and quantitatively analyzed how much performance improvement we can get from using these mechanisms in page-based software DSM system. We found that compared to 100 MBps Ethernet, the overhead of software DSM system is reduced by 10 % 〜49 %. But for applications whose memory access pattern causes frequent pagefaults, like FFT and Ocean, we still cannot get enough speed up.

1　はじめに

ソフトウエアDSMは非常に小さいコストで共有メモリを実現した。しかし、このようなソフトウェアベースシステムは専用ハードウェアを用いた分散共有メモリシステムに比べれば処理性能が低い。その原因はメモリコンシステンシを保持するためにノード間の通信が頻繁に起こり、ソフトウェア処理時間と通信時間が増大してしまうからである。

この問題を解決するために様々な研究が行なわれてきた。その成果の一つは共有メモリオーバヘッドを軽減するRelease Consistencyプロトコルがソフトウェア仮想分散共有メモリシステムのプロトコルとして確立してきた。

このような研究が進んでいる中で、近年、Myrinet[2], DEC Memory Channel[3], Gigabit Ethernet, ATM Network, といった高速ネットワークが商用化されてきた。ソフトウェアDSMはこのようなネットワークを用いることによって性能が大きく向上できることが期待されている。

本研究は高速ネットワークの技術の中の、(1) 低オーバヘッド通信機構及び (2) メモリマップ通信機構に着目する。まずページベースソフトウェアDSMを実装し、このシステムを用いて、通信コストの解析を行ない、上述の通信機構を用いたときのソフトウェアDSMのパフォーマンスを定量的に解析する。解析結果から、どれぐらいの速度向上が期待できるか、またソフトウェアDSMのボトルネックが解消されるかどうかを明確になる。

本稿では、2 章で低オーバヘッド通信機構及びメモリマップ通信機構について述べ、ソフトウェアDSMでどのように使われるかを述べる。3 章でシステムの実装とそのシステムを用いた解析手法を述べる。4 章で解析結果をまとめ議論を行なう。最後に 5 章でまとめをする。

2　高速通信機構

2.1　低オーバヘッド通信機構

低オーバヘッド通信機構はネットワークインタフェースに専用プロセッサやメモリを搭載し、従来ホストプロセッサが処理するプロトコル処理の一部をネットワークインタフェースに任せることで通信のソフトウェアオーバヘッドを軽減する。またユーザメモリに直接読み書き

できるような 0-copy 通信 や、 DMA を活用することでさらにオーバヘッドの少ない通信を可能にする。

低オーバヘッド通信機構を代表する Myrinet のネットワークインタフェースの概略図は図 1に示す。このような通信機構を用いると高速な通信が可能になり、例えば、従来 100 MBps のイーサネットで小パケットを送受信するのに 100[μs] 以上かかったものは 10[μs] 程度にすることができた。

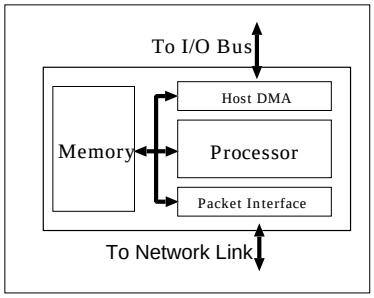

図 1: Myrinet NIC の構成

2.2 メモリマップ通信機構

メモリマップ通信機構は各ノードでグローバルメモリ空間を送信バッファ(outgoing buffer) あるいは受信バッファ(incoming buffer) としてマップし、送信バッファに書き込まれたデータはリモートメモリライト機構を用いて自動的に受信バッファにコピーされる (図 2)。

メモリマップ通信機構は次のような特徴をもつ。

1. 受信バッファはローカルメモリ領域として存在するが送信バッファはネットワーク I/O アドレスにマップされる。
2. 書き込み・読みだしのオーバヘッドは通常のローカルメモリアクセスと同じである。また、受信バッファへの書き込みは DMA を用いるのでホストプロセッサを割り込むことなく書き込むことができる。
3. あるグローバルメモリ空間を送受信バッファにマップしたいときは図 2のノード 1 のように送信バッファと受信バッファ、2つのアドレス空間にマップしなければならない。

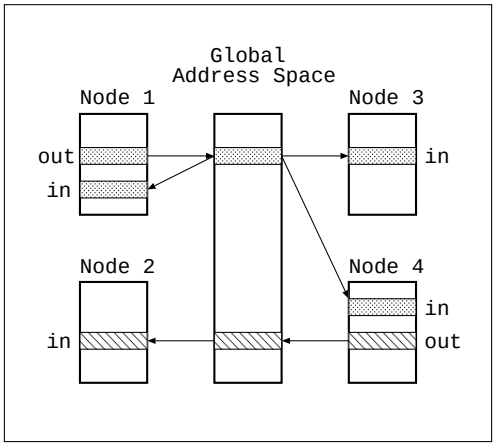

図 2: メモリマップ通信機構

2.3 高速通信機構とメモリコンシステンシプロトコル

低オーバヘッド通信機構とメモリマップ通信機構は抽象レベルの異なる通信機構と考えることができる。図 3 に示すようにメモリマップ通信機構は通信レイヤにおいてより上位にあることがわかる。

ソフトウェア DSM のコンシステンシプロトコルにはメッセージパッシング通信機構で実装される Sequential Consistency (SC)、Lazy Release Consistency (LRC)[6], Home-based Lazy Release Consistency (HLRC)[9] プロトコルがある。またメモリマップ通信機構を生かしコンシステンシを保持する Automatic Update Release Consistency (AURC)[4] プロトコルがある。

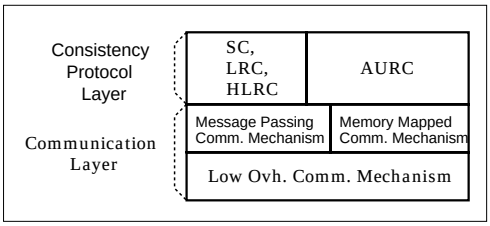

図 3: 高速通信機構とコンシステンシプロトコル

これらのコンシステンシプロトコルの性質を簡単にまとめると次のようになる。

- **SC**、 共有メモリへの更新は更新した直後に他のノードに反映される。
- **LRC**、共有メモリへの更新は Acquire 同期の時に反映される。
- **HLRC**、LRC と同様に共有メモリへの更新は Acquire 同期の時に反映される。但し、各ページにホームノードを割り当て、共有メモリへの更新は Release 同期の時にホームノードに送られる。ホームノードにあるページのコピーは常に有効なコピーである。
- **AURC**、LRC と同様に共有メモリへの更新は Acquire 同期の時に反映される。但し、各ページにホームノードを割り当て、共有メモリへの更新はメモリマップ通信機構を用いて更新する都度にホームノードに送られる。ホームノードにあるページのコピーは常に有効なコピーである。

3 パフォーマンス解析とその手法

3.1 解析概要

解析を行なうにはまずページベースソフトウェア DSM システムを実装する。このシステムに SPLASH-2 のベンチマーク郡からいくつかのプログラムを移植し実験を行ない、実行時間、共有メモリオーバヘッド、通信トレースなどのデータを収集する。通信トレースから実験に用いた 100 Base-TX ネットワークにおける通信コストを解析する。さらに Myrinet のような低オーバヘッドネットワークを想定し、その通信コストを推定する。最後に、低オーバヘッド通信機構の効果及びメモリマップ通信機構効果を求めるために

- 低オーバヘッドネットワークを用いたとき (Myrinet) と用いないとき (100Base-TX)
- メモリマップ通信機構を用いたとき (AURC) と用いないとき (SC, LRC, HLRC)

の性能を比較する。

3.2 システムの実装

システムはライブラリとして実装した。メモリコンシステンシ管理は 8[kB] のページ単位で行ない、ノード間通信には UDP/IP プロトコルを用いている。このシステムは SC、LRC、及び HLRC プロトコルをサポートする。実行形態は 1 ノード当たり 1 スレッドのみである。このシステムの API は表 1にまとめる。

AURC の解析に関しては HLRC を用いて AURC の動作をシミュレーションする。AURC のページの更新はコストが生じないと仮定し HLRC の実行時間から twin の作成、diff の作成、diff の送信・適用コストを削除し求める。

表 1: システムのインタフェース

tsm_startup()	システムの初期化
tsm_alloc()	共有領域の確保
tsm_create_procs()	リモートプロセスの起動
tsm_finish()	システムの終了
tsm_barrier()	barrier 同期
tsm_lock()	lock 同期
tsm_unlock()	unlock 同期
TSM_PID	プロセスの識別子
TSM_NUMNODES	システムのノード数

3.3 通信コストの解析

通信コストの解析はプログラム実行時間からどれぐらい通信コストが占めているかを求めるためである。まず、イーサネット上の UDP/IP の通信コストを抽出し、その他の計算時間・共有メモリオーバヘッド時間から分離する。次に低オーバヘッド通信機構を想定した場合の通信コストを推定し、それをもともとの通信コストのところに入れ換えることで低オーバヘッド通信機構のを用いたときの実行時間・共有メモリオーバヘッドを求めることができる。

通信コストは通信レイテンシと通信オーバヘッドと 2 つの場合に分けて解析する。通信コストが通信レイテンシであるのは「要求を出してその応答を待つ」の場合である。このとき要求を出す側からみると通信コストは要求メッセージの通信レイテンシと応答メッセージの通信レイテンシの和である。一方、通信コストが通信オーバヘッドであるのは「要求をもらってそれに応答する」という場合である。つまり応答する側からみると通信コストは要求メッセージを受信するときの受信オーバヘッドと応答メッセージを送信するときの送信オーバヘッドの和である。

3.3.1 イーサネット上の UDP/IP の通信コストの解析

イーサネット上の UDP/IP 通信過程は図 4に示している。図 4(a) は 1 つのイーサネットパケットに収まるような、小さいパケットを転送する場合を示す。また、図 4(b) はパケットがが 2 つのイーサネットパケットに分割された場合を示す。

図 4の sendmsg overhead、observed network latency、recvmsg overhead は通信レイテンシの測定可能な量である。図 5は Sun SparcStation 20 (SuperSPARC-II 75 MHz Processor、100Base-TX NIC) 上で測定した通信レイテンシを表している。通信コストが通信レイテンシである場合、single trip latency の測定結果を用いて通信コストを直接計算することができる。一方、sendmsg overhead と recvmsg overhead は図 4からわかるようにそれぞれ真の送信オーバヘッドと受信オーバヘッドの一部にしかすぎないので、通信コストが通信オーバヘッドである場合これらの値を直接用いることができない。

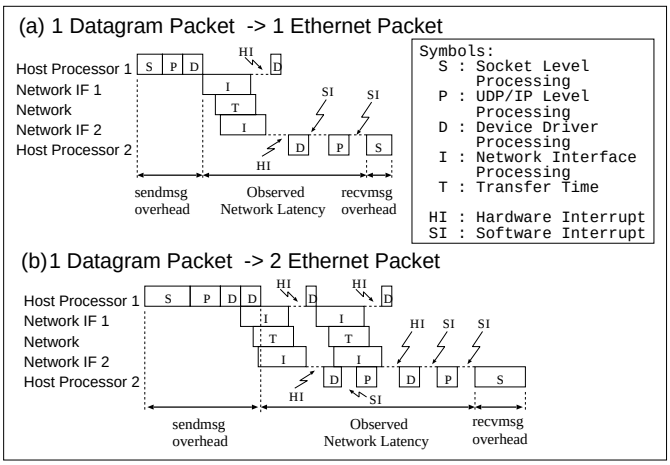

図 4: イーサネット上の UDP/IP の通信過程

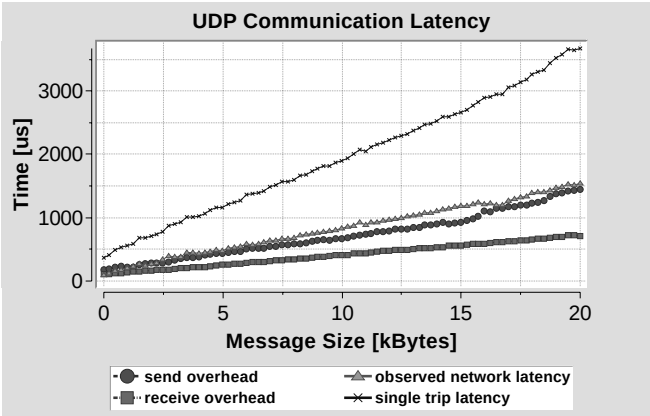

図 5: UDP 通信レイテンシ、測定結果

真のオーバヘッドは一般に ソケットレベルの処理 (S)、UDP/IP レベルの処理 (P)、デバイスドライバの処理 (D)、と割り込みコスト (HI/SI) からなる。通信オーバヘッドの解析は次のように行なう。ソケットレベルの処理 (S) は recvmsg overhead で近似する。また、UDP/IP レベルの処理 (P)、デバイスドライバの処理 (D) は 1 イーサネットパケット当たりに固定であると仮定する。これを測定結果に適用し近似を求めた結果、 イーサネットパケット当たりの P + D は 4000 cycle、また割り込みコストは 1 回当たり 3000 cycle と求まった。通信オーバヘッドは

これらの値を用いて計算する。

3.3.2 低オーバヘッド通信機構の通信コストの推定

低オーバヘッド通信機構は Myrinet のようなネットワークを想定する。データを送受信するときの様子は図6に示している。ユーザプロセスに呼び出された通信ライブラリはデータの送信を準備し、デバイスドライバを呼び出す。ドライバはデータのアドレスをネットワークインタフェースに通知し、DMA 転送を開始させる。データはネットワークインタフェースのメモリにコピーされ、専用プロセッサがそれを処理してパケットインタフェースを通して相手に送信する。このとき DMA コントローラ、専用プロセッサ、パケットインタフェースはオーバーラップして処理を行なう。DMA 転送が終ったらネットワークインタフェースは割り込みを起こしドライバに通知する。受信側では届いたパケットが専用プロセッサに渡され、パケットから取り出したデータを DMA 転送でユーザプロセスの受信バッファにコピーする。

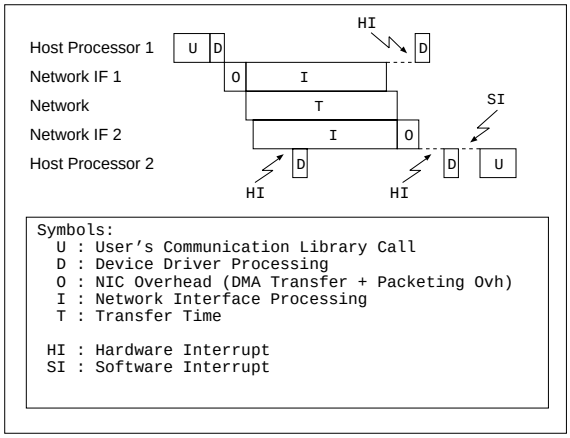

図 6: 低オーバヘッド通信機構の通信過程

通信コストを求めるには通信レイテンシ・送信オーバヘッド・受信オーバヘッドを推定しなければならない。これらの値を推定するには各通信パラメタを仮定し図 6の通信過程に適用する。用いられた通信パラメタは表 2にまとめた。

表 2: 低オーバヘッド通信機構のパラメータ

Software Overhead	400 [cycle]
I/O Bus Bandwith	132 [MBps]
Network Bandwith	100 [Mbps] 1 [Gbps]
Interrupt Cost	3000 [cycle]

Software Overhead は データを準備する時間であり、データのサイズによらず 400 cycle 固定と仮定する。I/O Bus Bandwith はネットワークインタフェースとメモリとの間のデータ転送のバンド幅を表し、32 bit - 33 MHz PCI バスの理想的な場合を仮定する。Network Bandwith はネットワークバンド幅を表し、100 Mbps と 1 Gbps 、２つの場合に分けて解析する。これは同じバンド幅で 100 Mbps イーサネットに比べて低オーバヘッド化の効果とさらにバンド幅をあげたときの効果をみるためである。最後に Interrupt Cost は UDP/IP の場合と同様に 3000 cycle とする。

4 実験とパフォーマンス解析の結果

4.1 実験環境及びベンチマークパラメタ

実験に用いられるワークステーションクラスタは 100 Base-TX で接続されている 4 つのノードから構成されている (表3)。また、測定対象となるプログラムは SPLASH-2[8] ベンチマーク群から FFT, LU, Ocean 及び Water-Nsquared を使用した。プログラムのパラメタ及び必要な共有メモリ領域は表 4に示している。

測定に用いた４つのプログラムはレギュラーなプログラムである。つまり、各ノードの計算量が動的に変化することはない。これはクラスタが同性能のノードから構成されていないことによる負荷アンバランスの影響を小さくするためである。以下は測定結果・解析結果を説明するが、全ての測定結果は性能の一番低いノード (1) における測定結果である。

表 3: クラスタの構成

Node	Machine	Processor	Mem	OS
1	SparcStation 20	SuperSparc-II 75 MHz	128 [MB]	Solaris 2.5
2	Ultra 1	UltraSPARC 167 MHz	96 [MB]	Solaris 2.5
3	Ultra Enterprise 3000	UltraSPARC-II 248 MHz	504 [MB]	Solaris 2.5.1
4	Ultra 10	UltraSPARC-IIi 300 MHz	128 [MB]	Solaris 2.6

表 4: ベンチマークのパラメタ

Benchmark	Problem Size	Memory
FFT	262144 points	12 MB
LU (Contig.)	1024 × 1024 matrix	8 MB
Ocean (Contig.)	258 × 258 ocean	11 MB
Water-Nsquared	1331 molecules	1 MB

4.2 並列実行による高速化

各ベンチマークの実行時間は図 7-図 10に表す。実行時間は上からの順番で次のように分割して表示する。

- **barrier**, barrier 同期に消費された時間。
- **readsegv**, リードページフォルトの処理時間。
- **writesegv**, ライトページフォルトの処理時間。
- **sigio**, アプリケーションコードを実行する最中に起こった SIGIO のハンドラの処理時間。
- **(un)lock/sigalrm**, lock 同期の時間, unlock 同期の時間, 及び アプリケーションコードを実行する最中に起こった SIGALRM のハンドラ処理時間を合わせたもの。
- **computing**, アプリケーションコードを実行する時間。

また、各ベンチマークの高速化は表 5に示している。

この結果から、LU と Water-Nsquared は複数ノードで並列実行されることによって高速化が見られるが、FFT と Ocean では高速化が得られず、逆に遅くなったことがわかる。FFT と Ocean では共有メモリコンシステンシ

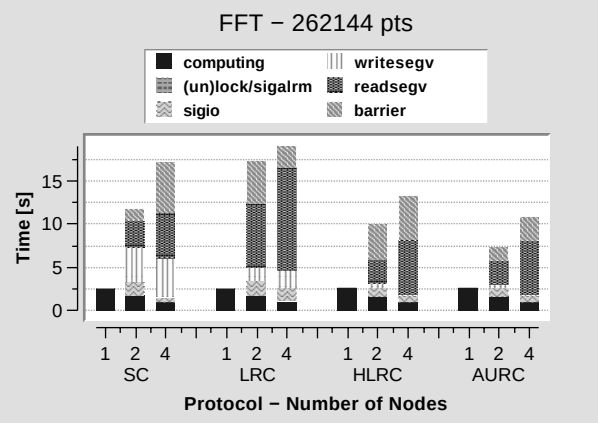

図 7: FFT 262144 points - 実行時間分割

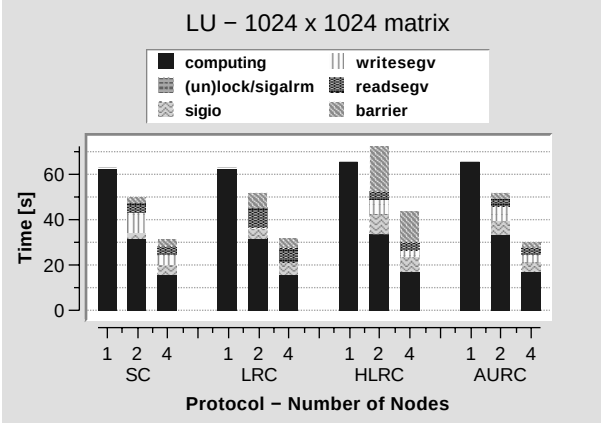

図 8: LU 1024 × 1024 matrix - 実行時間分割

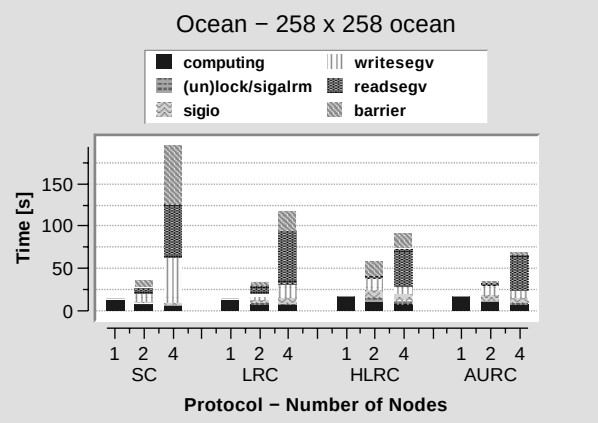

図 9: Ocean 258 × 258 ocean - 実行時間分割

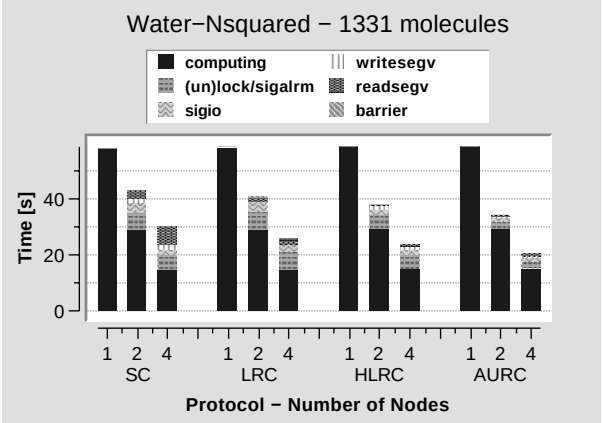

図 10: Water 1331 molecules - 実行時間分割

管理の時間及び同期時間が大きくなり並列実行によるメリットを打ち消してしまうからである。

このようにソフトウェア DSM の性能はアプリケーションの特徴によって大きく左右される。一般に高速化を得るためには高い computation to communication ratio が必要になる。

FFT の場合を考えてみると、 FFT は N 個のデータ点を$\sqrt{N} \times \sqrt{N}$ の 行列として表現する。各ノードに$\frac{\sqrt{N}}{P}$ 行 (P はノードの数) をブロックとして割り当てる。FFT の計算は radix-$\sqrt{N}$ six-step FFT アルゴリズムを用いるが、このアルゴリズムの 6 つのステップの中の 3 ステップは行列の転置 (transpose) である。図 11は 262144 points FFT の転置を示している。各ノードは自分のブロックに書き込むための必要な要素を要求し転置行列を作る。このとき他のノードのブロックは読み込み不可になっているためリードページフォルトが起こる。一回の転置において 2 ノード構成では 128 回、4 ノード構成では 192 回のリードページフォルトが起こる。リードページフォルトが起こるとリモートノードから有効なページあるいは diff を持って来なければならない。一回の転送で数 ms から数十 ms がかかり、全体的には一回の転置では数秒の時間がかかってしまう。一方、行列転置の計算内容は主にメモリコピーだけであり、メモリコピーにかかる時間は数十 [ms] である。このように FFT の転置は非常に小さい computation to communication ratio をもち並列実行による高速化を得るのは困難である。

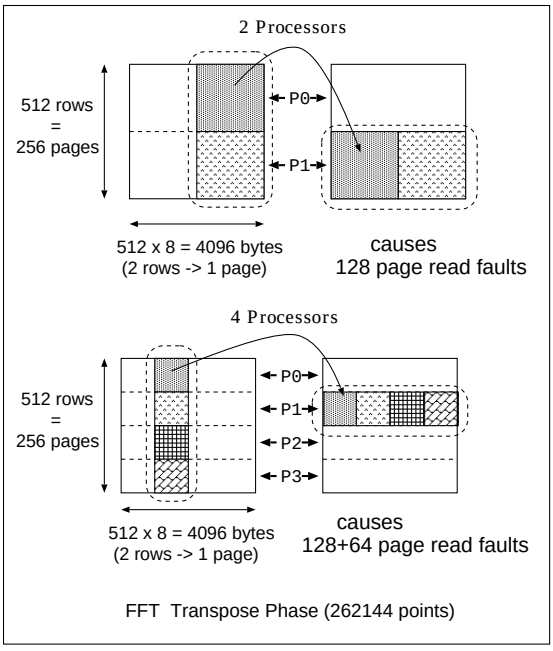

図 11: FFT のデータ行列の転置

Ocean についても同じように解析することができ、隣合うノードのブロックをアクセスすいるとき、FFT と同じようにページフォルトが頻繁に起こり遅くなってしまうことがわかる。

表 5: スピードアップ (100Base-TX)

Benchmark	2 Nodes				4 Nodes			
	SC	LRC	HLRC	AURC	SC	LRC	HLRC	AURC
FFT	0.21	0.14	0.25	0.35	0.14	0.13	0.19	0.21
LU	1.3	1.2	0.86	1.2	2.0	2.0	1.5	2.2
Ocean	0.39	0.40	0.23	0.40	0.07	0.11	0.15	0.20
Water-Nsquared	1.4	1.4	1.5	1.7	1.9	2.3	2.5	3.1

表 6: スピードアップ (Low Overhead - High Bandwith (1 GBps) Network)

Benchmark	2 Nodes				4 Nodes			
	SC	LRC	HLRC	AURC	SC	LRC	HLRC	AURC
FFT	0.29	0.19	0.38	0.47	0.18	0.19	0.30	0.32
LU	1.4	1.3	1.1	1.4	2.2	2.2	2.1	2.3
Ocean	0.49	0.49	0.34	0.49	0.09	0.15	0.22	0.28
Water-Nsquared	1.5	1.5	1.6	1.8	2.1	2.4	2.8	3.2

表 7: 低オーバヘッド通信機構による共有メモリオーバヘッドの削減率

Benchmark	2 Nodes				4 Nodes			
	SC	LRC	HLRC	AURC	SC	LRC	HLRC	AURC
FFT	33 %	26 %	42 %	36 %	23 %	32 %	40 %	32 %
LU	27 %	24 %	45 %	30 %	22 %	27 %	49 %	23 %
Ocean	22 %	19 %	39 %	30 %	19 %	25 %	35 %	34 %
Water-Nsquared	20 %	18 %	32 %	37 %	17 %	17 %	32 %	35 %

表 8: メモリマップ通信機構による共有メモリオーバヘッドの削減率

Benchmark	to HLRC		to Best Protocol	
	2 Nodes	4 Nodes	2 Nodes	4 Nodes
FFT	26 %	11 %	(HLRC) 26 %	(HLRC) 11 %
LU	42 %	25 %	(SC) 10 %	(LRC) 11 %
Ocean	41 %	23 %	(LRC) 19 %	(HLRC) 23 %
Water-Nsquared	47 %	43 %	(HLRC) 47 %	(HLRC) 43 %

4.3 高速ネットワークを用いたときのパフォーマンス

4.3.1 低オーバヘッド化・バンド幅の拡大による通信コストの削減

図 12-図 15は通信コストを解析した結果を表している。図の「Eth」「LOH」「LOH-HB」の記号は想定したネットワークを表し、それぞれ次のようである。

- Eth : Ethernet 100Base-TX、UDP/IP プロトコル、バンド幅 100 [Mbps]
- LOH : 低オーバヘッド通信機構、0-copy 通信プロトコル、バンド幅 100 [Mbps]
- LOH-HB : 低オーバヘッド通信機構、0-copy 通信プロトコル、バンド幅 1 [Gbps]

図 12-図 15の「Eth」と「LOH」を比べると「Eth」と「LOH」が同じバンドであっても「LOH」の通信コスト (割り込みコストも含む) は 「Eth」より 40 % - 63 % も少ない。また、バンド幅を 100 [Mbps] から 1 [Gbps] に拡大するとさらに通信コストが減少する。「LOH」と「LOH-HB」の場合を比べると 5 % - 38 % 通信コストが削減されることがわかる。このように低オーバヘッド化の効果とバンド幅の拡大の効果を合わせるとイーサネット 100Base-TX の場合に比べると通信コストが 43 % - 72 % 削減されることがわかる。

4.3.2 低オーバヘッド通信機構の効果・メモリマップ通信機構の効果

低オーバヘッド通信機構による、共有メモリオーバヘッド (メモリコンシステンシ管理のオーバヘッドと同期のオーバヘッド) の、削減率を表 7に示す。全体的には 17 % - 49 % の削減率が見られるが、特に HLRC プロトコルにおける削減率が 32 % - 49 % 一番大きいことがわかる。その次は AURC で 20 % - 37 % の削減率がみられる。HLRC と AURC は、通信量が多く、またソフトウェア処理が軽いため、通信コストの割合が大きい。従って低オーバヘッドネットワークの効果が一番大きくみられる。

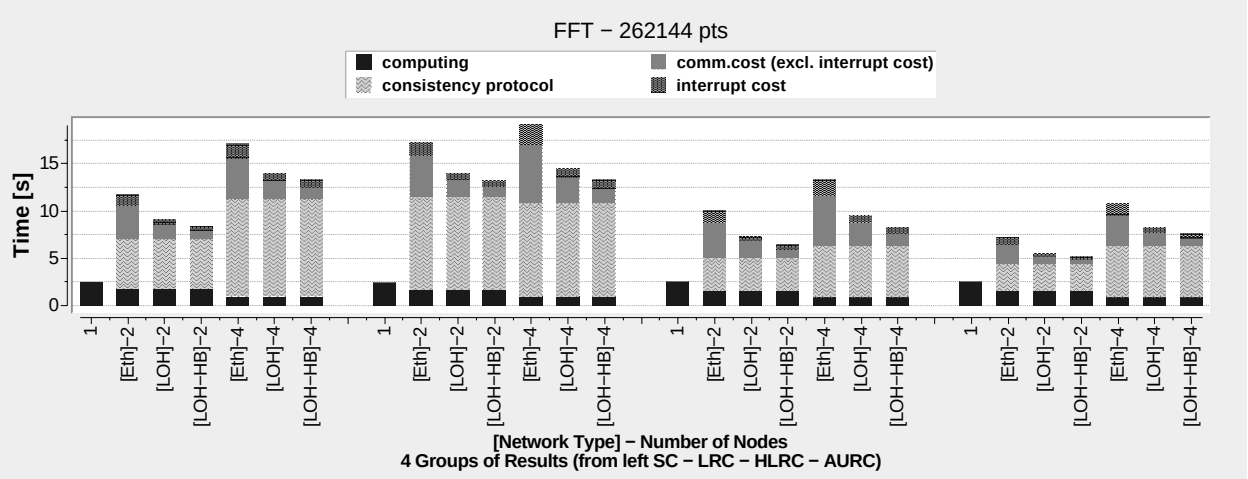

図 12: FFT 262144 points - 実行時間と通信コスト

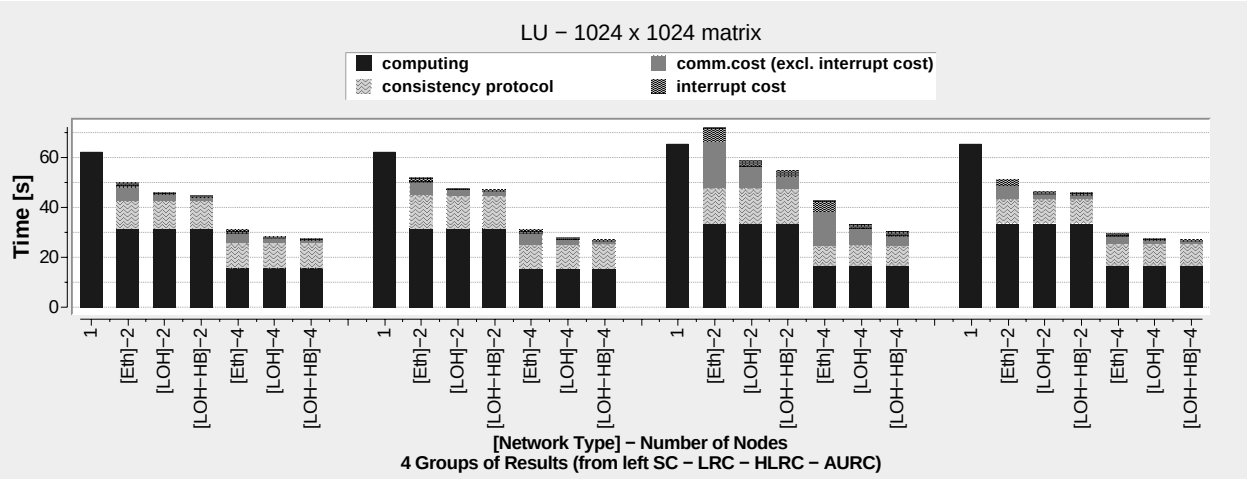

図 13: LU 1024 × 1024 matrix - 実行時間と通信コスト

表 8はメモリマップ通信機構によってどれぐらい共有メモリオーバヘッドが削減されるかを表している。この表に示した値は AURC の共有メモリオーバヘッドを、HLRC の共有メモリオーバヘッド及び (SC・LRC・HLRC) の中で一番性能のよいプロトコルの共有メモリオーバヘッドと比較した場合を表している。この表からメモリマップ通信機構を利用することによって他のプロトコルに比べて 10 % - 47 % の共有メモリオーバヘッドの削減が得られることがわかる。

バンド幅 1 [Gbps] の低オーバヘッドネットワークを用いたときのスピードアップは表 6に示す。アプリケーション別にみてみると FFT 及び Ocean は通信コストが大きく削減されたものの並列実行による高速化がえられなかった。これは図 12及び図 14をみればわかるように FFT と Ocean の場合、通信コスト以外の、主にページフォルトによるコンシステンシ管理のオーバヘッドが大きいからである。通信コストが大きく削減されてもコンシステンシ管理のソフトウェア処理コストが解消されないためボトルネックになってしまう。

このように低オーバヘッド通信機構及びメモリマップ通信機構によって共有メモリオーバヘッドが大きく短縮されたが、既存のコンシステンシプロトコルではシステムのボトルネックが解消されず充分な高速化が得られない場合もある。

5 まとめ

本研究は低オーバヘッド通信機構及びメモリマップ通信機構に着目して、このような通信機構を用いたページベースソフトウェア DSM の性能はどれぐらい向上できるかを定量的に解析した。

ページベースソフトウェア DSM は共有メモリ領域をページ単位で管理するが、細粒度のシステムに比べると性能がアプリケーションの特徴によってより大きく左右される。並列実行による高速化を得るには充分高い computation to communication ratio が必要になる。測定を行なった結果、LU 及び Water-Nsquared では並列実行による高速化が得られるが FFT 及び Ocean では得られなかった。

解析結果から低オーバヘッド通信機構及びメモリマップ通信機構によって共有メモリオーバヘッドが大きく削減されたことがわかる。しかし、FFT や Ocean のような、メモリアクセスパターンによってページフォルトが頻繁に起こるような場合は性能が向上されたものの、システムのボトルネックが解消されず充分な高速化が得られなかった。この場合は通信コストが小さくなるがメモリコンシステンシ管理のソフトウェア処理コストがボトルネックになってしまう。

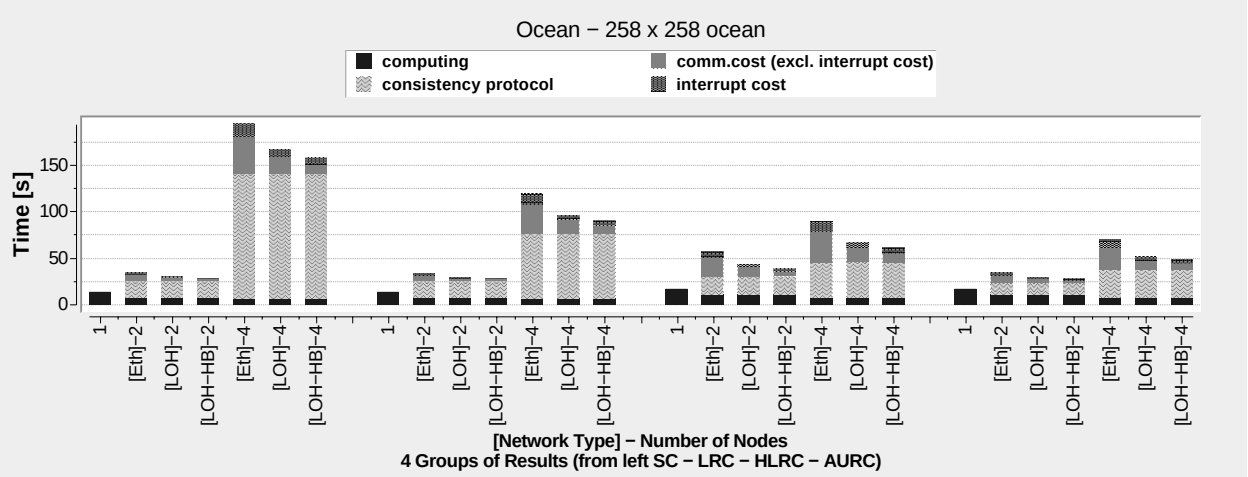

図 14: Ocean 258 × 258 ocean - 実行時間と通信コスト

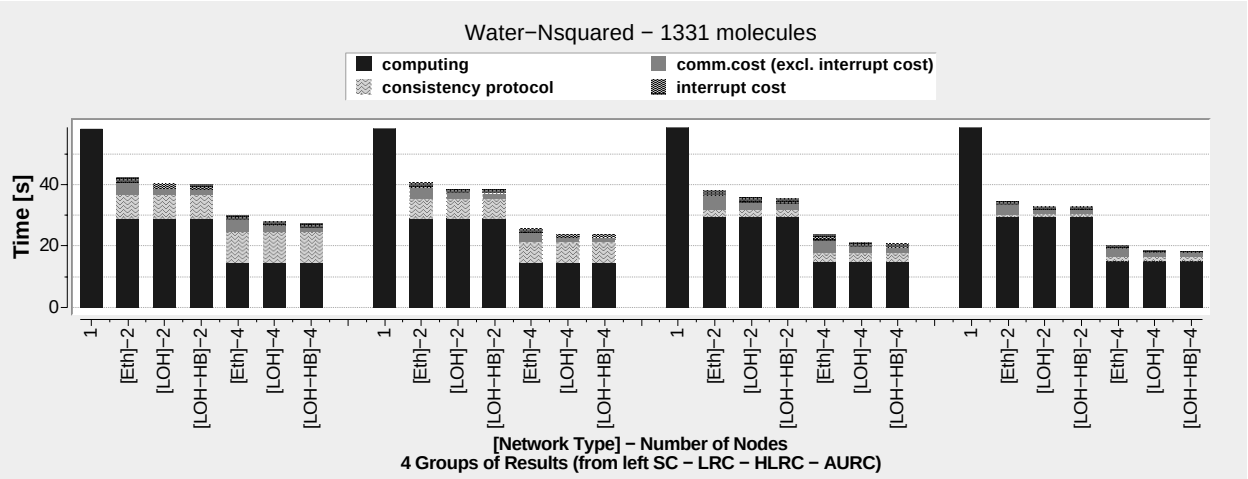

図 15: Water-Nsquared 1331 molecules - 実行時間通信コスト

今後の課題

ページベースソフトウェア DSM の性能を向上させるには通信ネットワークの部分の高速化だけでなく、メモリコンシステンシ管理のソフトウェア処理の部分も充分小さく抑える必要がある。このため今後の課題として以下のような点があげられる。

- アーキテクチャ的なサポートを利用し、その特徴を最大に利用できるコンシステンシプロトコルを開発する。
- アプリケーションの特徴を理解し、ページベースのソフトウェア DSM に適用すると 充分高い、 computation to communication ratio が得られるかどうかを調べる。得られない場合は アルゴリズムを考え直すか、computation to communication ratio が低いところだけを逐次実行させる、などの対策が考えられる。

参考文献

[1] Angelos Bilas and Jaswinder Pal Singh. 「**The Effects of Communication Parameters on End Performance of Shared Virtual Memory Clusters**」. *Proceedings of Supercomputing 97, San Jose, CA*, November 1997.

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L.Seitz, Jakov N. Seizovic, Wen King Su. 「**Myrinet – A Gigabit-per-Second Local-Area Network**」. *IEEE Micro, 15(1):29-36*, February 1995

[3] Marco Fillo and Richard B. Gillett. 「**Architecture and Implementation of MEMORY CHANNEL 2**」. *Digital Technical Journal, Volume 9, Number 1*, 1997.

[4] Liviu Iftode, Cezary Dubnicki, Edward W. Felten and Kai Li. 「**Improving Release-Consistent Shared Virtual Memory using Automatic Update**」. *2nd IEEE Symposium on High-Performance Computer Architecture*, February 1996

[5] Liviu Iftode, Jaswinder Pal Singh, Kai Li. 「**Understanding Application Performance on Shared Virtual Memory**」. *Proceedings of 23rd Annual Symposium on Computer Architecture*, May 1996

[6] Pete Keleher. 「**Distributed Shared Memory Using Lazy Release Consistency**」. *PhD Thesis, Rice University*, December 1994.

[7] Leonidas Kontothanassis, Galen Hunt, Robert Stets, Nikolaos Hardavellas, Michal Cierniak, Srinivasan Parthasarathy, Wagner Meira, Sandhya Dwarkadas, and Michael Scott. 「**VM-Based Shared Memory on Low-Latency, Remote-Memory-Access Networks**」. *Proceedings of the Twenty-Fourth International Symposium on Computer Architecture, pages 157-169, Denver, CO*, June 1997.

[8] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrue, Jaswinder Pal Singh, and Anoop Gupta. 「**The SPLASH-2 Programs: Characterization and Methodological Considerations**」. *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995

[9] Yuanyuan Zhou, Liviu Iftode and Kai Li. 「**Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems**」. *Proceedings of the Operating Systems Design and Implementation Sysmposium*, October 1996