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

2007－HPC－112 (3)
2007／9／9

メモリ効率を考慮した組み込み向け高信頼ソフトウェア分散共有メモリの検討

李 珍泌† 木村英明† 佐藤三久†

アプライアンス機器などこれからの組み込みシステムでの高性能・高信頼の要求に答えるための並列実行プラットフォームとしてソフトウェア分散共有メモリシステムを用いることを提案する．物理的に独立したメモリを持ち，ネットワークによって並列に接続された並列システムを対象にする．各ノードに冗長なデータを持たせることで耐故障性を保証し，高信頼化を図る．また，2 次記憶を持たないシステムのためにソフトウェア分散共有メモリシステムでのメモリの効率的な利用について検討した．SuperH プロセッサで予備評価を行い，問題点について明らかにした．

Design of Reliable Software Distributed Shared Memory System with efficient memory usage for Embedded System

JINPIL LEE,† HIDEAKI KIMURA† and MITSUHISA SATO †

We propose a software distributed shared memory (DSM) system as an execution platform for high performance and high reliable parallel system for embedded systems, which has distributed memory and are connected by network. To realize fault tolerance, each node has redundant data for recovery from node failure on our system. We also consider efficient memory usage in the implementation of software DSM for diskless systems. We found some serious problems in the implementation of software DSM on embedded parallel system of SuperH processors.

1. はじめに

今後，ますます発展するユビキタス情報社会においては，センサー，アクチュエータなどの機器，人間とのインタフェースをつかさどるユビキタス端末での高機能化，高信頼化のみならず，これらの機器からの情報を統合して活用することが必要となると考えられる．このための，いわゆるアプライアンス機器などの組み込みシステムにおいては，高性能と高信頼性が求められている．

一方，ハイエンドのプロセッサのみならず組み込みプロセッサにおいても，マルチコアなどの並列化が進みつつある．また，テクノロジの進歩による高機能化により，組み込み用途のプロセッサも MMU（Memory Managing Unit）などの一般プラットフォーム向けのプロセッサと同様の機能を備えているものが多くなった．これにより，Linux などの OS が組み込みプロセッサに対応するようになり，多くの既存のソフトウェアが使えるようになっている．

本研究ではアプライアンス機器などの組み込みシステムを対象に，高性能・高信頼の要求に答えるための並列実行プラットフォームとしてソフトウェア分散共有メモリシステム（Software Distributed Shared Memory，Software DSM）を用いることを提案する．物理的に独立したメモリを持ち，ネットワークによって並列に接続された組み込みシステムを前提とし，各ノードに冗長なデータを持たせることで高信頼なシステムを構築する．また，2 次記憶を持たないシステムのためにソフトウェア分散共有メモリシステムでのメモリの効率的な利用について検討する．

まず，そのための予備評価として，代表的な組み込みプロセッサであるルネサス テクノロジ社の SuperH[1] プロセッサを用いた並列システムを構築し，ソフトウエア分散共有メモリシステムを試作して問題点について検討した．

次章では，提案する組み込みシステム向け高信頼ソフトウェア分散共有メモリシステムの背景について述べる．3 章でベースとした分散共有メモリシステムである SCASH について述べ，4 章において高信頼化を行うための方式について述べる．5 章では限られたメモリを効率的に使えるように検討を行い，6 章では SuperH を用いた予備評価を行う．最後に研究の内容をまとめ，今後の課題について述べる．

† 筑波大学 大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

2. 組み込みシステム向け高信頼ソフトウェア分散共有メモリシステムの利点

組み込みシステムは日常生活に密接な関係を持つものが多く，高い信頼性が求められる場合も多い．システムの一部が故障しても全体の動作への影響を最小化し，動作し続けることが望ましい．信頼性を高めるための基本的な方策の一つは冗長性である．すなわち，プロセッサに関して言えば，複数のプロセッサを用意しておき，お互いに補完することである．

また，ルネサス テクノロジ社の SH2A-DUAL や ARM 社の MPCore などの登場から分かるように，組み込みシステムではマルチコア化による高性能化が進んでいる．マルチコアは基本的に共有メモリシステムであるが，さらに高性能化・高機能化が必要となる場合には，プロセッサをネットワークで結合したマルチプロセッサシステムになっていくと予想される．共有メモリでは OpenMP などの簡便なプログラミングモデルが使えるが，このような形態のシステムではメモリシステムを意識した複雑なプログラミングをしなくてはならない．

提案するソフトウエア分散共有メモリシステムは，以上の要請に対し，以下の利点がある．

- ソフトウエア分散共有メモリにより，マルチコアプロセッサと連続した共有メモリモデルでプログラミングが可能になる．例えば，OpenMP のような簡便なプログラミング環境が利用できる可能性がある．
- アプリケーション毎に耐故障性を持たせることは非生産的であり，実行プラットフォームとしてソフトウェア分散共有メモリを用いることで並列アプリケーションの下で耐故障性を保証することで，システムの信頼性を上げると同時にプログラマの負担を軽減し，生産性を上げることができる．

これまでのいくつか高信頼性を持たせたソフトウエア分散共有メモリシステムが提案されている．従来の汎用サーバや高性能計算などの場合とは異なり，組み込みではディスクがない場合もあり，メモリの効率を効率を考慮するなどのコンパクトな実装を検討する必要がある．

3. SCASH の概要

我々は，提案するシステムについて SCASH[2] をベースに検討を行う．SCASH は新情報処理開発機構で開発されたクラスタシステムソフトウェア SCore[3] 上に提供されているソフトウェア DSM である．ユーザレベルのライブラリとして提供されており，API 関数の使用によって仮想的な共有メモリを分散メモリ上で実現する．ユーザが明示的に API 関数を挿入することで共有メモリの割り当て，メモリバリアによる同期などの一貫性制御を行う．

ソフトウェア DSM のメモリ一貫性維持のために様々なモデルが提案されている[4]．SCASH では Home-based Lazy Release Consistency（以下 HLRC）が用いられており，メモリバリアで同期を行う際に変更されたページのホームノードがそれを共有する他のノードのデータを更新，または無効化することでメモリの一貫性を保証する．ここでは無駄な通信を避けるため無効化プロトコル（invalid protocol）を用いる．

図 1 に SCASH の仮想共有メモリを示す．DSM start と DSM end の間の領域が SCASH によって使われる．page info にはページを管理するために必要な情報が収納される．dsm area は仮想共有メモリである．システム全体の物理的に独立されたメモリを一つのメモリのようにマッピングしている．home i となってる部分はそのノードによって管理されるページ（ホームページ）の領域である．remote a，b，c は他のノード a，b，c によって管理されるページ（リモートページ）が置かれる領域である．各ページは INVALID,READ-ONLY,READ/WRITE の三つの内，一つの状態を持ち，アクセスにより状態が転移する．

SCASH は通信機構として Myrinet 及び Ethernet 上に低レイテンシかつ高バンド幅通信を提供するユーザレベルの高速通信ライブラリ PM を用いている．また，小島らによって MPI を用いて通信を行う SCASH-MPI[8] が提案されている．

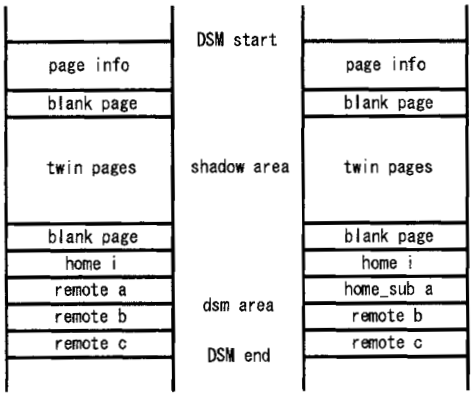

図 1 仮想共有メモリ　　図 2 冗長性を持つ仮想共有メモリ

4. 耐故障性を考慮したソフトウェア DSM

本章では耐故障性を実現するためのソフトウェア DSM の仮想共有メモリ構造や一貫性モデルについて考察する．並列に実行されるノードの中でその一部が故障したとしてもシステム全体は停止せず，動作し続けることが望ましい．多くの研究ではあるタイミングでノードのスナップショットを作成し，故障が発生した場合，そのスナップショットを用いて耐故障性を実

現している．本研究ではソフトウェア DSM の仮想共有メモリの各ページに対して冗長なホームノードを設定することで耐故障性を実現する．

4.1 冗長性を持ったホームノードの設定

既存の HLRC モデルでは各ページが持つホームノードは一つである．リモートページにアクセスがあった場合，ホームノードにそのページのデータを要求する．しかし，このようなシステムではホームノードが故障した場合，そのノードのページにアクセスする全てのノードが動作不能となり，システム全体が停止する．このような事態を避けるために仮想共有メモリの各ページにもう一つの冗長なホームノードを持たせることが考えられる．

本研究で提案する仮想共有メモリの構造を図 2 に示す．図 2 はノード i の仮想共有メモリを表す．その構造は従来のものとほとんど変わらない．但し，ノード a のリモート領域の変わりに home_sub a という従来と異なる領域が存在する．これはノード i がノード a の冗長なホームノード（以下 a のサブホーム）であることを表す．サブホームのデータはホームノードのものと同一であり，メモリバリアによって一貫性を保証するようにする．従って，ノード i でノード a をホームに持つページをアクセスする場合，ノード間通信は発生せず，home_sub a のデータを利用することができる．

メモリバリアでは以下のようにしてサブホームの更新を行い，ホームノードとの一貫性を保証する．その説明に図 2 を用いる．

- ノード a がホームページを更新した場合，メモリバリア同期の際にサブホームであるノード i の home_sub a を正しい値に更新する．従来はリモート領域であるため無効化される．
- ノード i が home_sub a を更新した場合，元のデータとの差分を送り，メモリバリア同期で home i のデータを更新する．ここまでは従来と同じだが，invalidate protocol と異なり，ノード i のサブホーム領域を無効化しない．
- 上記の二つの場合においてホームノードでもサブホームでもない第三のノードの remote a は従来と同じく無効化される．
- 第三のノード b が remote a を更新した場合，メモリバリアでノード a の home a とノード i の home_sub a を更新する．ノード b の remote a は無効化される．

以後，a のサブホーム i で home_sub a にアクセスした場合，メモリバリア同期によってその値が信頼できるものであるため，ノード間通信を持ちいらずに直接利用する．他のリモート領域の場合は，従来通りホームノードとの通信によりページデータを獲得する．

このようにして一貫性維持のプロトコルを変更したソフトウェア DSM 上で故障が発生した場合，チェックポインティングを用いて耐故障性を実現する．

4.2 チェックポインティングを用いた耐故障性の実現

チェックポインティングは耐故障性を実現する代表的な手法である．定期的にアプリケーションの実行状況を記録し，故障が起きた時に記録した情報を元に実行を再開する．チェックポインティングを行うためのライブラリとして，BLCR[5] などが開発されている．ソフトウェア DSM の耐故障性を実現するため，チェックポインティングを用いる．

まず，以下のような動作をメモリバリア同期に追加する．

- 各ノードはメモリバリア毎に現在のプロセスの実行情報とローカルな（他のノードと共有されていないノード固有のデータ）データを記録する．
- 記録した情報をサブホームに転送し，保持する．

この追加によって，故障が起きた時の再開ポイントとしてメモリバリアを用いることができる．故障が発生した時，システムの全てのノード（故障したノードを除く）が現在の仮想共有メモリに対する変更を全て廃棄し，過去の最新のメモリバリア同期から実行を再開する．前節で行った変更によって故障したノードのホームページとサブホームの home_sub 領域の一貫性が保証されるため，故障したノードへのアクセスをサブホームによって行うことができる．サブノードは自分自身のホームノードとしての役割はもちろん，故障したノードの変わりとしての役割をも果たすことになる．

ソフトウェア DSM の動作を以下のように変更する．対応する動作を図 3 に示す．

- ホームノードの故障でページアクセスできない場合，以後そのページのアクセスにサブホームを用いる（1，2）．
- サブホームは故障の間ホームノードのように振る舞い，ホームページのデータ更新やリモートページの無効化を行う（3）．

故障によって停止したノードが正常な状態に戻った場合，サブホームから自分のホームページを獲得し，故障前にサブホームを勤めていたページを持つホームノードからもデータをコピーする（サブホームとしての役割にも復帰する）．サブホームから最新のメモリバリアで記録した実効状況やローカルデータを持ってきて回復を行う．故障の時と同様，システムの全てのノードが過去の最新のメモリバリアから実行を再開する．

サブホームの割り当てを，自分ではないもので 1 対 1 に対応するようにして，単独故障に対する耐故障性を持たせることができる．但し，故障が起きている時はそのサブホームに多くのアクセスが発生し，メモリバリア同期のコストが大きくなるため，システム全体の性能が低下する．

今後はノードの故障に加え，ネットワークなど他の要素の故障に対する耐故障性についても検討を行う必要がある．

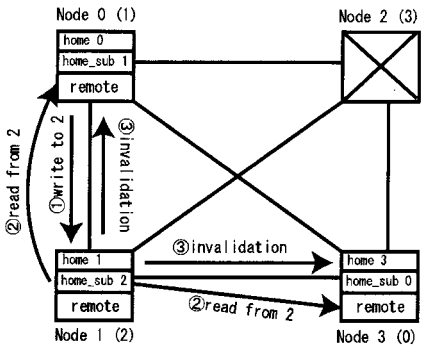

図 3　ノードの故障に対する耐故障性

5. メモリ効率を考慮したソフトウェア DSM

本章ではソフトウェア DSM の効率的なメモリ利用に関する検討を行う．組み込みシステムでは 2 次記憶を持たない場合が珍しくなく，メモリ上ですべての作業を行わなければならない．そのようなシステムで並列計算を行う場合，限られたメモリを効率的に利用することが重要である．メモリに入り切らないデータはリモートページングにより他のノードにスワップアウトする必要がある．現在ネットワークの性能は凄まじい進化を遂げており，リモートページングの性能が 2 次記憶を用いたページングに匹敵する (もしくは改善する) という研究結果も出ている[6)9)]．今井ら[6)] はネットワークブロックデバイスドライバによる分散ページングを用いて 2 次記憶に頼らない大規模仮想メモリ空間を実現している．また，GVVM[7)] では使用頻度の低いページを他のノードのメモリにスワップアウトさせることでメモリ負荷分散を実現している．本研究ではそれらのようなレイヤを使う前の段階として，DSM の共有メモリ空間の効率的な利用について考える．リモートページングを用いる前に要らない，またはその後利用される確立が低いページを解放することで限られたメモリを効率的に使うようにする．

5.1 ページ状態による解放の優先順位

各ノードの計算は主にホームページを用いて行われ，そのデータ局所性が性能の肝となる．アプリケーションのデータをシステムの各ノードにバランスよく分散させることは DSM の中で重要な作業の一つである．従って，ホームページを解放の対象とすることはあまり良いアイデアではない．解放したページがまた必要になる可能性が高く，ノード間通信が頻繁に発生し，システムの性能を低下させる．また，耐故障性を実現するためにもメモリバリアと次のメモリバリアの間にホームページを常時持っていることが望ましい．サブホームページに対しても同様のことが言える．したがって，今回の研究ではシステムの各ノードがホームページとサブホームページを収容できる位の物理メモリを持つと仮定する．そして解放する対象をリモートページに限定する．ページの状態により以下のようなことが言える．

- INVALID ページ：この状態のページは無効な値を持つため，実メモリを無駄に消費する．よって，高い優先順位で解放することができる．
- READ-ONLY ページ：READ-ONLY の場合，リモートノードによるページデータの変更はない．またホームノードに正しいデータが存在するため，ページアウトは実メモリの開放だけで良い．しかし，ページをスワップアウトした後にアクセスが行われるとホームノードとの通信が必要となる．従って開放するページを慎重に選択しなければならない．
- READ/WRITE ページ：READ/WRITE の場合，書き込んだデータをホームノードに反映させる必要があるため，ページアウト・ページインの両方においてノード間通信が発生する．従って READ-ONLY ページより解放の優先順位を低くすべきと考えられる．

5.2 解放するページの選択

リモートページの状態により解放するページを優先順位を付けることができる．

まずページが INVALID 状態である，もしくは書き込みによるホームノードからの無効化メッセージで INVALID 状態になった場合は積極的に実メモリから開放するようにする．

INVALID な状態のページが存在しない場合，次に READ-ONLY の状態のページの中で解放するページを探す．データ参照の局所性を前提としてアクセスが古いページを用済みと判断し，積極的に解放する．そのために INVALID 状態から読み込みによるページフォルトが起きた場合，そのタイミングを記録する（最初のアクセスタイミング）．そのタイミングを比較し，一番前にページフォルトが起きたものをページアウトの対象とする（First-In Fist-Out の手法である）．

READ/WRITE 状態のページは READ-ONLY のページよりも更に低い優先順位を持つ．INVALID なページに書き込みが行われページフォルトが起きた時，その書き込みのタイミングを記録する（最初の書き込み）．解放するページを選択する時に初期書き込みのタイミングを比較してもっとも古いページを解放する．READ-ONLY に対する書き込みでページフォルトが行われた場合も書き込みのタイミングを記録し，優先順位を READ/WRITE ページと同じようにする．

FIFO は実装が簡単で動作が軽いが，アプリケーションによってはより適した置き換えアルゴリズムが存在

する可能性がある．今後，様々な種類のアプリケーションに対して実験を行い，最適なアルゴリズムを探すことが課題となる．

6. SuperH プロセッサによる予備評価

6.1 評価環境

我々は実装設計の前段階として，組み込みシステムに SCASH ベースのソフトウェア DSM を試作し，予備評価を行った．今回使用したプラットフォームはルネサス テクノロジ社の SuperH プロセッサ（以後，SH）のクラスタである．図 4 に SH 評価ボードを示す．このボードがクラスタの 1 ノードとなっている．表 1 に実験環境を示す．評価には最大 4 ノードを使用した．

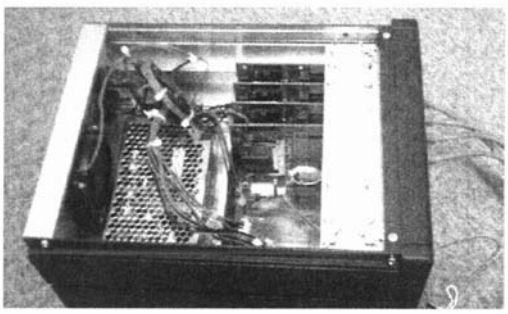

図 4 SuperH クラスタ

表 1 実験環境

項目	名称
CPU	HD6417750R(SH4) (core clock 240MHz)
Memory	128MB (256Mb × 4)
Network	100Base-Tx ethernet
OS	Linux kernel 2.6.20.1

6.2 TCP/IP を用いた通信レイヤの実装

通信のレイヤとして，TCP/IP のソケットライブラリを用いて通信ライブラリを作成した．通信の初期化と終了，同期・非同期通信を行う関数を実装した．初期化によって各ノードはシステムの自分以外の全てのノードに対し，送信用と受信用のコネクションを持つ．通信関数はこのコネクションを用いて他のノードとの通信を行う．他に，プログラムをシステムの各ノードにコピーし，実行するプログラムを作成した．ノードリストを読み込み，SSH によるリモートアクセスを行うことで実装を行った．これで MPI のような SPMD (Single Program Multiple Data) 実行モデルを実現している．

試作したソフトウェア DSM では計算とは別のスレッドを作り通信を行う．メッセージキューを設け，計算スレッドで通信が必要になった時，通信メッセージをキューに入れる．通信スレッドはポーリングしながらキューを監視し，通信要求があった場合，相手ノードにメッセージを送る．また，他のノードからのメッセージの受信やリモートメモリアクセスによるメモリの読み込み・書き込みも通信スレッドで行う．

6.3 SH-Linux での実装

Linux のカーネルが SH プロセッサに対応しているため，SCASH の実装は簡単である．但し，2.6.20 より古いバージョンのカーネルはシグナルハンドラの引数にフォルトアドレスを書き込まないため，SCASH が正しく動作しない．また，最新のカーネルでもフォルトを起こしたアクセスが read/write のどちらなのかを判断するコンテキスト情報をユーザが得ることができない（カーネル情報なのでユーザからアクセスできない）．このような問題を回避するために，SCASH のページフォルト処理を以下のように変更した．

フォルトしたページが，

- INVALID:read fault → 状態を READ-ONLY に
- READ:write fault → 状態を READ/WRITE に

この変更は INVALID ページに write fault が起きた時，2 回のページフォルトが起きるので性能面ではよくない実装である．

6.4 評価結果

評価のため，1024 × 1024 の 2 次元ラプラス方程式を SCASH 上で計算した（ガウス・ザイデル法の 50 回反復，以後このアプリケーションを laplace と呼ぶ）．このアプリケーションでは他のノードのホームページに書き込みを行わないため，上記の変更は性能に影響しない．計算の結果を表 2 に示す．

表 2 laplace on SuperH

node	time(sec)
serial	31
2node	5979
4node	3042

表 3 laplace on Xeon

node	time(sec)
serial	1.011
2node	1.535
4node	0.873

表 2 の結果により，SCASH を用いて並列化を行った場合の性能が著しく低下することが分かった．その原因としてページフォルト処理にかかる時間が非常に長いことが考えられる．

laplace のデータ初期化を利用して，ページフォルト処理のコストを計った．laplace はデータ初期化関数で自分のホームページに書き込みを行う．SCASH のホームページの初期状態は READ-ONLY なのでページフォルトが起き，READ/WRITE 状態になる．従って，laplace のデータ初期化にかかる時間はページフォルトの処理にかかる時間と考えることができる（初期化にかかる時間は実行時間に含まれない）．その値をシグナルハンドラの内部で消費されるユーザ時間（初期化の間の累積時間）と比較することにする．

測定の結果を表 4 に示す．1024x1024 のサイズを計算する laplace の初期化時間を計ったものである．シ

グナルハンドラで使われるユーザ時間と比べ，初期化にかかる時間が非常に長いことが分かる．初期化関数のユーザ時間が無視できるほど小さいため，OS が長い時間をシグナル処理のために使っていることが推測される．

表 4 SH のページフォルト処理 （単位：秒）

ノード数	初期化の全体時間	シグナルハンドラの累積時間
2node	241 sec	0.125 sec (0.052%)
4node	123 sec	0.066 sec (0.054%)

次は，この結果と性能の低下の関係について考える．現在の SCASH の実装では書き込みを検知するため，各メモリバリア毎に全ての READ/WRITE 状態のホームページを READ-ONLY にする．このような実装では laplace の反復毎に全てのホームページに対してページフォルトが起きる．SH-Linux のページフォルト処理にかかる時間が非常に長いため，laplace の性能が低下したものと考えられる．図 2 で 2 ノードと 4 ノードの間で時間が短縮したことも各ノードでフォルトが発生するページ数が半減したからと（計算する要素数が半減するため）解釈することができる．

比較のため，2.4GHz の Intel Xeon プロセッサによる同様な実験の結果を表 5 に示す．表 3 には laplace を Xeon プロセッサのクラスタで動かした結果を示す．SH と同様の TCP/IP を用いた通信ライブラリを使っている．

Intel Xeon の場合，逐次実行時間と比べてページフォルトの処理にかかる時間が短い．従って，現在の実装でも 2 ノードと 4 ノードの間で性能の改善が見られる．逐次と比べ 2 ノードの性能が悪いのは通信のコストが計算のコストに勝っているためである．

表 5 Intel Xeon のページフォルト処理 （単位：マイクロ秒）

ノード数	初期化の全体時間	シグナルハンドラの累積時間
2node	131791 usec	13251 usec (10.05%)
4node	66765 usec	6533 usec (9.78%)

SH での予備評価はページフォルトの処理時間が性能向上を阻む大きな原因となっていることを示す．今後，OS の動作を追跡し，正確な原因とその解決法を調べることが課題となる．また，SH 以外の組み込みプロセッサに対しても検証を行い，今回の結果が組み込みプロセッサにおいてどれだけの一般性をもつものか明確にする必要がある．

7. まとめと今後の課題

本研究では組み込み向けの実行プラットフォームとしてソフトウェア DSM を使用するために，冗長なホームノードによる耐故障性とページ解放によるメモリの効率的な利用を提案した．SuperH プロセッサによる予備実験の結果，ページフォルト処理の実行時間が性能低下の大きな原因となることが分かった．今後はその明確な原因を探すと共に，本稿で設計したモデルを実装していく．また，SH 以外の組み込みプロセッサに対しても検証を行う．現在，PCI-EXPRESS をベースとしたインターコネクションを開発中であり，それによって通信の性能向上によるソフトウェア DSM の高速化が期待される．

謝辞 本研究の一部は科学技術振興事業団・戦略的創造研究推進事業 (CREST) の研究プロジェクト「省電力でディペンダブルな組込み並列システム向け計算プラットフォーム」による．

参 考 文 献

1) ルネサス テクノロジ SuperH RISC engine ファミリ，japan.renesas.com/sh
2) SCASH, http://www.pccluster.org/score/dist/score/html/en/reference/scash/index.html
3) SCore, http://pccluster.org
4) K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems", ACM Trans. on Computer Systems, Vol. 7, No. 4, pp. 321-359, 1989
5) Duell, J., Hargrove, P., and Roman., E. "The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart". Berkeley Lab Technical Report
6) 今井, 松葉, 石川, "分散ページングによる大規模仮想メモリ空間", 情報処理学会 研究報告 2007-HPC-109, HOKKE2007, pp-85-90, 2007
7) 平野, 田沼, 一杉, 須崎, 塚本, "大域的仮想仮想記憶（GVVM）の RWC-1 上での性能予測", 情報処理学会 OS 研究会 SWoPP'94, 1994
8) 小島, 佐藤, 朴, 高橋, "MPI を通信レイヤに用いるソフトウェア分散共有メモリシステム", 情報処理学会論文誌コンピューティングシステム, Vol. 49, No. SIG7（ACS 10）, 2005
9) S. Dwarkadas, N. Hardavellas, L. Kontothanassis, R. Nikhil and R. Stets, "Cashmere-VLM:Remote memory paging for sofrware distributed shared memory", 13th International and 10th Symposium on Parallel and Distributed Processing, pp. 153-159, 1999
10) 大澤, 弓場, "ワークステーションクラスタに用いる回復可能共有記憶システム", 情報処理学会 研究報告 96-OS-73, pp. 13-18, 1998
11) Zoraja, A. Bode and V. Sunderam, "A Framework for Process Migrationin Software DSM Environments", Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing, pp 158-165, 2000