システムソフトウェアと
オペレーティング・システム 75－17
（1997．6．5）

WS クラスタにおけるスケーラブルネットワーク機構

中條 拓伯[1]　中野 智行[2]　金田 悠紀夫[1]

[1] 神戸大学 工学部 情報知能工学科
[2] 日立ソフトウェアエンジニアリング

概要

本論文では，ネットワーク資源を入出力デバイスとして見なすのではなく，メモリ管理用のハードウェアとオペレーティングシステムの管理及び保護されたメモリ資源として見なす通信モデルである LMMC (Link Memory Management based Communication) モデルを提案する．通信をメモリへのアクセスとして行うことにより，従来のネットワーク機構で見られるようなオーバヘッドを回避し，低オーバヘッドでマルチプログラム，マルチユーザ環境におけるプロセス間通信を実現することができる．

Scalable Network of Workstation Clusters

Hironori NAKAJO[1], Tomoyuki NAKANO[2] and Yukio KANEDA[1]
email: nakajo@jedi.seg.kobe-u.ac.jp

[1] Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University
[2] HITACHI Software Engineering

Abstract

In this paper, we propose a new communication model called LMMC (Link Memory Management based Communication) for workstation clusters. In LMMC model, network resources are considered as memory resources which are managed and protected by memory management hardware and operating systems. LMMC enables low overhead interprocess communication in multiprogram, multiuser environment. We consider the hardware and operating system structure to support LMMC model interprocess communication, and build LMMC model experiment system using serial links called STAFF-Link.

1 はじめに

近年のワークステーション (WS) の性能向上とネットワーク技術の進歩により，ネットワークで接続されたWS群を用いた並列処理システムは WS クラスタ，あるいは NOW(Network of Workstations) と呼ばれ，各方面で盛んに研究が行われている [1].

しかしながら，現状のネットワークにおいてはバンド幅が狭く，また多くのオペレーティングシステムでは，トランスポート層以下の処理はカーネル内部で処理が行われ，システムコールのオーバヘッドやユーザ空間からカーネル空間へのデータの移動に伴うオーバヘッドのため，通信遅延が大きくなる．さらに，TCP などの通信プロトコルの処理もオーバヘッドの原因となる．このような問題から，WS クラスタの応用範囲は計算処理に比べ通信の割合が少ない問題に限定されたものとなっていた．

したがって，WS クラスタ環境を実用的なものにするためには，並列処理を目的とした以下のような，新たなネットワーク機構 (ネットワークデバイス，オペレーティングシステム，プロトコル等) が必要と考えられる．

- システム全体の通信バンド幅が十分であり，ノード数が増加してもそれに見合うバンド幅を確保.
- 分散環境とは異なり通信距離を数十～数百 m に限定することで，高速かつ信頼性のある通信路を提供.
- 様々な結合トポロジを柔軟に構成.

このようなネットワーク形態は，超並列計算機の相互結合網と LAN の中間のカテゴリに位置するものと考えられ，System Area Network (SAN) と呼ばれている [2].

本研究室において，これまでに STAFF-Link (Serial Transparent Asynchronous First-in First-out Link) というネットワークデバイスと，STAFF-Link を利用して多数の WS を結合した超並列計算機のディスク入出力システムの構築を行ってきた [3]. STAFF-Link は，1 台の WS に多数のシリアルポートを持たせることができるネットワークデバイスであり，WS 間の相互結合網として利用することで，以下のような特徴を持つ WS クラスタを構成でき，SAN の形態として有効であると考えられる．

- 多数のポートによる高次元の相互結合網が構成可能.
- 同時に複数のリンクで通信が行えるため，システム全体で高いバンド幅を維持.
- シリアルリンクの取り扱いの簡単さから，結合トポロジを柔軟に変更.

本研究では，WS クラスタ上での効率の良い並列処理のための通信機構として **LMMC(Link Memory Management based Communication)** という通信モデルを提案する．LMMC モデルでは，ネットワーク資源をメモリ資源としてみなし，通信処理をシステムによるメモリ管理とアプリケーションによるメモリアクセスで行う．ネットワーク資源をメモリ資源とみなすことによって，従来の入出力処理に起因した種々のオーバヘッドを回避でき，また，通常のメモリ資源と同様，MMU などのハードウェアとオペレーティングシステムの保護により低オーバヘッドでマルチプログラム，マルチユーザ環境におけるプロセス間通信を実現できることになる．

ここでは．まず現状の研究背景と LMMC モデルの提案を行う．次に，LMMC モデルでの通信を効率よく実現するためのハードウェア構成としての LMMC アーキテクチャについて述べる．続いて，LMMC アーキテクチャにおいて，オペレーティングシステムに要求される機能について述べ，STAFF-Link を用いた LMMC モデルの実験システムの実装について説明する．

2 LMMC アーキテクチャ

2.1 技術的背景

WS (あるいはハイエンドのパーソナルコンピュータ) に搭載されるプロセッサの処理能力はここ数年で急速に向上している．それに伴い，ワークステーションのシステムアーキテクチャ(プロセッサ-メモリ間のデータパス，プロセッサ- I/O 装置間のデータパスの形態など) に改善が求められている．特に，プロセッサ速度とメモリアクセス速度の格差により生じる問題はメモリウォール問題として注目され，主記憶混載型プロセッサ等による解決策が提案されている [4].

松本らはメモリウォール問題の解決策として，プロセッサとメモリ間の接続に高速シリアルリンクを用いる Memory String Architecture (MSA) を提案している [5]. 高速シリアル通信は，近年特に注目を集めている技術である．半導体技術の進歩により，シリアルインタフェースでも十分な転送帯域を確保できるようになり (Dally らはツイステッドペアケーブルあるいは基板上の銅配線による 4.0Gbps の高速通信の可能性を示している [6])，パラレルインタフェースに比べ，ノイズ，コスト，接続の柔軟性などの面で優れているため，これまでシステムの構成要素としてパラレルインタフェースを用いられていた部分をシリアルインタフェースに置き換えようとする動きが近年盛んである．例えば，USB (Universal Serial Bus), IEEE1394 (P1394: Fire Wire), SSA (Serial Storage Architecture), Fibre Channel などが挙げられる．

メモリ資源は通常，メモリ管理用のハードウェアとページング機構により保護，管理されている．メモリ資源へのアクセスは (ページングなどの処理を除くと) オペレーティングシステムが介在することなくユーザプロセスから直接アクセスすることができる．このため，入出力デバイスにみられるようなオーバヘッドは存在しない．つまり，LMMC モデルでの通信を実現できれば低オーバヘッドでマルチプログラム，マルチユーザ環境におけるプロセス間通信を実現できることになる．

2.2 LMMC モデル

LMMC アーキテクチャでは，WS 間の接続に用いられるポートは LMMC ネットワークインタフェースに接続される (図 1).

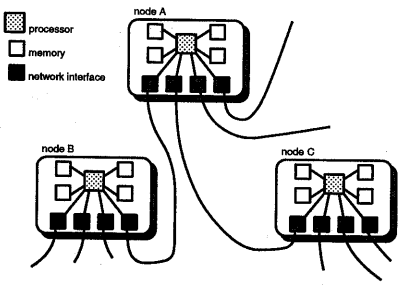

図 1: LMMC アーキテクチャ

このとき，LMMC ネットワークインタフェース はプロセッサに対して MSA[5] と同様のインタフェースを提供する．すなわち，プロセッサは LMMC ネットワークインタフェースに付随する通信用のバッファを自ノード内にあるメモリ素子と同様に扱う．これにより，ノード間の通信は (リモート) メモリ操作に置き換えられ，ノード間の通信路を入出力デバイスとしてではなく，MMU とオペレーティングシステムの管理により保護されたメモリ資源として扱うことができる．したがって，ノード間通信に関して，ユーザプロセスが入出力デバイスを扱う際のシステムコールや割り込み処理等に起因する種々のオーバヘッドを回避することが可能となる．

通信バッファを実メモリ空間にマッピングする方法として，LMMC アーキテクチャでは，自ノード内の受信バッファと接続先の受信バッファを実メモリ空間へ配置する方法を採った．受信バッファへマッピングされた領域への書き込み操作は接続先の受信バッファへの書き込み操作になる．つまり，データの送信は (非同期の) リモートメモリへのアクセスで，データの受信はローカルメモリへのアクセスで行われる (図 2)．ノード間で自由に読み書きできるようにバッファを共有することは物理的に可能であるが，ノード間でのバッファの管理が繁雑になるため，効率的であるとはいえない．また，自ノードの送信バッファのマッピングでは，他ノードからのデータの到着を自ノードで判断できないため，同期受信を効率よく実現できないという問題がある

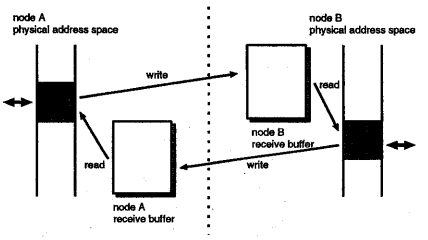

図 2: 通信バッファのマッピング

これまで述べてきた LMMC アーキテクチャでは，ネットワークインタフェースはプロセッサ-メモリ間のインタフェース上に存在する．しかしながら，プロセッサがメ

モリマップド I/O をサポートしていれば，既存のワークステーションの入出力バスを用いてネットワークインタフェースを構築することも可能である．

2.3 MMU による通信バッファの管理

前節でも述べたように，通信バッファ用のメモリは通常のメモリと同等に扱われる．すなわち，自ノードの受信バッファと接続先の受信バッファ(自ノードの送信バッファと見なせる) はページフレームに分割され，MMU とオペレーティングシステムによりページ単位で管理される．物理メモリに対する仮想記憶と同様に，ユーザプロセスが利用できる通信用のメモリ領域は，実際の通信バッファの容量による制限を受けない．

通信バッファを管理する上で通常のメモリと大きく異なるところは，通信バッファに対するアクセス時にページフォルトが起こった際のページング動作にある．あるプロセスがデータを送信する際にページフォルトが起こったとすると，ページアウトすべきページは自ノードには存在せず，接続先の受信バッファに存在するため，接続先のプロセッサにトラップをかけてページアウト処理を依頼しなければならない (図 3)．本稿ではこのようなページング動作をリモートページング **(remote paging)** と呼び，通常のページングと区別する．

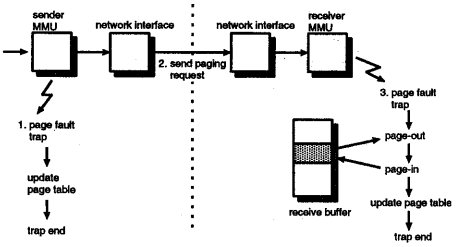

図 3: リモートページング

2.4 LMMC ネットワークインタフェース間のメッセージ

LMMC ネットワークインタフェース間で送受信されるメッセージは以下の 3 種類である．

- **通信路の確立のメッセージ:** 通信路を確立するため，接続先のノードにトラップをかける．パラメータとその処理の内容はオペレーティングシステムに依存する．
- **リモートライトメッセージ:** 接続先の通信バッファに対するデータの書き込み要求である．パラメータはアドレス，データサイズ，データである．このメッセージを受け取ったノードは実際に要求された書き込みを行う．
- **リモートページングメッセージ:** 接続先のノードに対するページングの要求である．このメッセージを受けとったノードは，トラップをかけてオペレーティングシステムにページング処理を依頼する．パラメータとその処理内容はオペレーティングシステム依存である．

2.5 LMMC ネットワークインタフェースの構成

LMMC ネットワークインタフェースはプロセッサとのインタフェース用のポート，LMMC ネットワークインタフェース間の通信用のポート，受信バッファ(デュアルポート RAM)，及びそれらを制御するコントローラから構成される (図 4)．

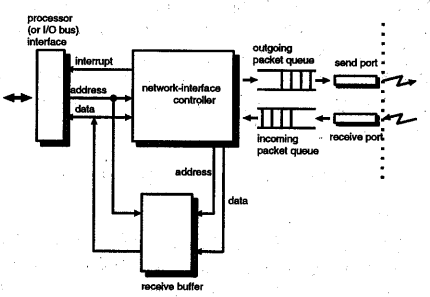

図 4: LMMC ネットワークインタフェース

コントローラは LMMC ネットワークインタフェース間のリンクのフロー制御，パケットの送受信，受信バッファへのデータの書き込み等を行う．リモートページングの要求を受信したときは，プロセッサへページフォルトを通知し，該当する処理が終了するまでパケットの受信操作を停止する．リンク間のフロー制御により，パケットの到着順序は保証されるため，リモートページングの要求を送信した直後から次のパケットを送信することができる (接続先でのページアウト処理の終了を待つ必要がない)．

3 LMMC ソフトウェアインタフェース

3.1 基本概念

LMMC アーキテクチャ上で LMMC モデルのプロセス間通信を行うためのオペレーティングシステムモジュールを **LMMC ソフトウェアインタフェース**と呼ぶ．ユーザプロセスはこのインタフェースを直接利用したり，このインタフェース上に構築されたメッセージパッシングシステムや DSM (Distributed Shared Memory) システムを用いることでプロセス間通信を行う．

ユーザプロセスに対して LMMC ソフトウェアインタフェースが提供するサービスは，プロセス間での仮想ページの対応づけと，対応づけられた仮想ページの属性の設定のみである．ユーザプロセスは対応づけられた仮想ページに対するメモリアクセスを行うことにより他のプロセスと通信を行うことができる．通信に関するその他の処理は，通常ユーザプロセスから意識されることはない．

LMMC ソフトウェアインタフェースでは，シリアルリンクで接続されたノードの受信バッファを送信側のノードで管理する点に特徴がある．そのため，ここでの動作の記述は送信側を主体に述べられている点に注意されたい．

3.2 ポート

LMMC ソフトウェアインタフェースでは複数のプロセスでページを共有することにより，多対多の通信を行うことができる．ページの共有は各プロセスのページテーブルエントリの項目を一致させることで容易に実現可能である．LMMC ソフトウェアインタフェースでは，このような共有可能な単方向の通信路を**ポート (port)** と呼ぶ．リモートページングは物理的な通信バッファにポートを対応させる動作とみなすことができる．

各ポートは**ポート識別子**と呼ばれる正の整数によって管理される．ポート識別子はネットワークインタフェース間で一意に定まる必要があり，ネットワークインタフェースで接続された両ノードにおいて，同じポートに対する識別子は必ず一致している必要がある．つまり，WS クラスタ上のすべてのポートは，

$$(sender\ node,\ receiver\ node,\ portID)$$

で表現できる．

物理アドレス空間に配置された通信バッファをリンクメモリ **(link memory)**，リンクメモリをページサイズで分割した単位をリンクページフレーム **(link page frame)**，仮想アドレス空間上の通信用のページを**仮想リンクページ (virtual link page)** とそれぞれ呼び，通常のメモリ，ページフレーム，仮想ページと区別する．

3.3 ポート管理のためのデータ構造

3.3.1 ページテーブル

ページテーブルはユーザプロセス毎に用意され，主に仮想ページと物理ページフレームとの対応表として用いられ，仮想ページの種々の状態を管理する (図 5)．

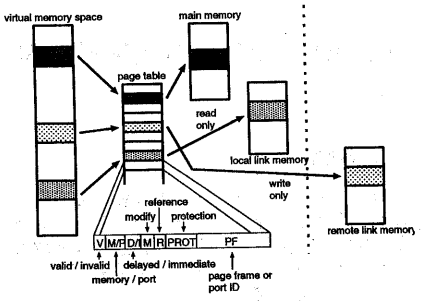

図 5: ページテーブル

LMMC ソフトウェアインタフェースでは，ページテーブルで仮想リンクページとリンクページフレームを対応させることによりリンクメモリへのポートの割り当て状態を管理する．仮想リンクページと通常の仮想ページとの相違点はユーザプロセスからは意識されないが，LMMC ソフトウェアインタフェース内では両者の取り扱いが異なるため，両者を区別するための手段が必要である．つまり，各仮想ページについて，その使用用途 (主記憶/通信) に関する情報が必要となる．また，仮想リンクページについては，それが送信用ページなのか受信用ページなのか，さらに送信用ページであればその更新方式についての情報も必要になる．これらの情報は，ページテーブルと独立して管理するより，ページテーブルエントリに項目を追加する形で管理するほうが効率的であると考えられる

また，ページテーブルエントリの保護情報に関して，データ送信用にマップされた仮想リンクページのページテーブルエントリは書き込み専用，データ受信用にマップされた仮想リンクページのページテーブルエントリは読み込み専用に設定しなければならない．また，仮想リンクページのキャッシングはすべて無効化しなければならない．

3.3.2 リンクページフレームテーブル

リンクページフレームテーブルは現在のリンクページフレームとポート **識別子**の対応を保持するテーブルであり，1 つのネットワークインタフェースに対して送信用と受信用のテーブルを用意する (図 6)．主に，ネットワークページングの際にどのリンクページフレームを操作の対象にするかを決定するのに用いられる．

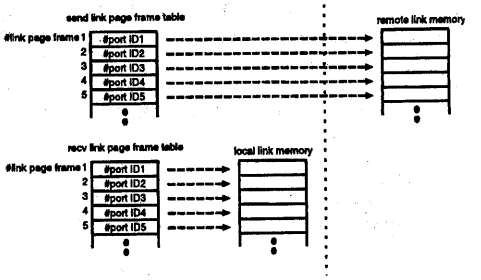

図 6: リンクページフレームテーブル

3.3.3 ポートテーブル

ポートテーブルはポート識別子とポートに対応づけられたプロセスの識別子と仮想ページとの対応を保持するテーブルである (図 7)．

ポートは複数のプロセスで共有されるので，テーブルのエントリは各要素の線形リストへのポインタとなる．リンクページフレームと同様に，1 つのネットワークインタフェースに対して送信用と受信用の 2 つのテーブルを用意する．それらに加え，送信用のポートテーブルのエントリにはそのポートがすぐに更新 (即時更新) するか，更新情報をまとめて更新 (遅延更新) するかを示すフラグの領域，受信用のポートテーブルにはページアウト時にページテーブルを退避する領域をそれぞれ用意する．

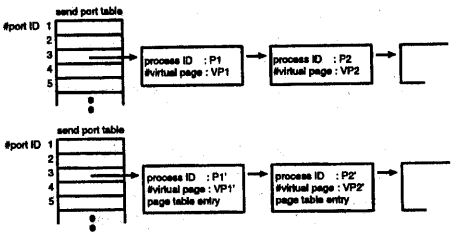

図 7: ポートテーブル

3.4 リモートページング

リモートページングは送信用のポートに対して書き込み操作を行った時，ポートがリンクメモリに存在せず，ページフォルトが発生した際に行われる動作である．その動作は 図3 に示した通りであるが，ここでは上述したデータ構造がどのように関係するかを中心に述べる．ページフォルトが発生したノードでは次の動作が行われる．

1. ページフォルトにより，トラップが発生する．
2. ページテーブルを用いて，ページフォルトを起こした仮想リンクページがどのポートに対応しているのかを調べる (有効/無効ビットがクリアされている仮想リンクページのページテーブルエントリにはリンクページフレームの代わりにポート識別子が格納されている)．
3. リンクページフレームテーブルを調べ，どのポートをページアウトさせるかを決定する．
4. ネットワークインタフェースに対し，ページインするポート識別子とページングの対象となるリンクページフレーム番号をパラメータとして，リモートページアウトの要求を発行する．
5. ポートテーブルを調べ，ページアウトされたポートを共有している各プロセスに対して，対応するページテーブルエントリを変更する．このとき，2. で述べたように有効/無効ビットをクリアするだけでなく，対応するポート識別子をページフレーム番号の項目に格納する．
6. ポートテーブルを調べ，ページインするポートを共有している各プロセスに対して，対応するページテーブルエントリを変更する．このとき，有効/無効ビットをセットし，ページフレーム番号の項目には対応するリンクページフレーム番号を格納する．
7. トラップから復帰する．

一方，リモートページアウトを受けとったノードでは次の処理が行われる．

1. ネットワークインタフェースがプロセッサにトラップをかける．
2. ネットワークインタフェースからパラメータを受けとり，ページアウトすべきポートを調べる．
3. ページアウトの処理を行う．
4. ページアウトされた仮想ページフレームに対するページテーブルエントリを更新する．
5. 指定されたポートをページインする．
6. ページインされた仮想ページフレームに対するページテーブルエントリを更新する．
7. トラップから復帰する．

ここで，ページアウトに指定されたリンクページフレームのデータをどこに退避するのかが問題となる．通常のページングでは，ディスク上の特定の領域に記憶されるが，リンクページに対してこの方法をそのまま用いると効率が悪い．例えばデータの受信を待ち合わせるために特定のアドレスをポーリングしているプロセスは送信側のプロセスがリモートページングを行うまで毎回ディスクをアクセスすることになる．そのため，LMMC ソフトウェアインタフェースでは受信用の仮想リンクページに対して，ページアウトの際のデータの退避用の仮想メモリを用意し，ポートがページインされていない時は通常の仮想メモリへアクセスされるようにページテーブルを操作する．そのため，リモートページングでページアウトされる際に，対応する仮想リンクページに関するページテーブルエントリをポートテーブルに退避しておく．

3.5 アプリケーションプログラミングインタフェース (API)

ここでは，ユーザプロセスが LMMC ネットワークインタフェースを利用するにあたって必要となるシステムコールについて述べる．これらは最も低レベルな機能しか提供していないため，実用レベルで用いるためには上位層のサポートが必要である．

ポートの生成
通信のためのポートを生成するためのシステムコールである．パラメータはネットワークインタフェース番号と送信用か受信用かを示すフラグである．オペレーティングシステムは利用可能なポート識別子を調べ，その値を返す．

port_id port_create(#network_interface, R/W)

ポートのマッピング
ポート識別子をパラメータとし，ポートとプロセスの仮想ページをマッピングする．オペレーティングシステムはポートテーブルを設定し，マップされたページの仮想アドレスを返す．指定されたポートが受信用であれば，ページアウト時の退避用の仮想メモリも同時に用意する．

virtual_address port_map(port_id)

ポート属性の設定
送信用のポートに対する即時/遅延送信を設定する．

set_write_policy(port_id, immediate/late)

ポートの消去
ポート識別子で指定されたポートを消去する．

delete_port(port_id)

3.6 プロセススケジューリングと同期機構

多くの並列マシンでは，協調的スケジューリング (co-scheduling or gang scheduling) が用いられているが [7]，WS クラスタ環境で実現するには困難である．また，各ノードでのローカルスケジューリングでビジーウエイトが用いられると，プロセッサ資源を無駄に消費する恐れがある．

WS クラスタ環境でプロセス間通信を考慮したスケジューリング方式に adaptive gang scheduling [8] がある．この方式では，メッセージを受信したノードが，そのメッセージの宛先のプロセスを優先的にスケジュールすることで協調的なスケジューリングを行う．LMMC ソフトウェアインタフェースにおいても，リモートページインされたポートにアクセスするプロセスを優先的にスケジューリングすることにより同様の協調的スケジューリングが実現できる．

また，同期メッセージの待ち合わせについて，一定時間ビジーウエイトで待ち，その後ブロックウエイトに切替える two-phase blocking が提案され，有効性が示されている [9]．LMMC インタフェースにおいては，ある仮想アドレスをポーリングすることで同期メッセージの待ち合わせが行われる．このとき，ポーリングされている仮想アドレスに対応する仮想リンクページがページインされていないときには，ポーリングしているプロセスをスリープさせる (つまりブロックウエイトさせる) ことにより，two-pahse blocking と同様の同期機構が実現できる．

4 関連研究との比較

Active Messages [10] はメッセージが到着した時に起動されるユーザレベルハンドラへのアドレスをメッセージに含ませることにより，通信のオーバヘッドを低減している．しかしながら，メッセージの到着毎にトラップが発生し，ハンドラの起動によるコストも大きい．LMMC モデルでは普段の通信ではトラップは発生せず，リモートページアウトが発生した時のみトラップが生じる．

通信路をメモリ空間へマッピングするアプローチは SHRIMP のネットワークインタフェース [11] や Hamlyn [12] などで用いられている．SHRIMP ネットワークインタフェースでは，送信プロセスと受信プロセスの仮想メモリ空間のマッピングにより通信を行うという点で LMCC モデルと同様のインタフェースを提供する．しかしながら，SHRIMP ネットワークインタフェースでは通信用のメモリ管理が通常のメモリと統合されておらず，LMMC モデルにおけるリモートページングに相当するものがない．また，Hamlyn では，通信用にマッピングされたメモリ領域が物理メモリ空間に固定されなければならず，LMMC に比べて柔軟性に欠ける．

5 おわりに

ここでは，WS クラスタ上で効率の良い並列処理環境を構築するためのネットワーク機構として LMMC モデルを提案した．LMMC モデルではネットワーク資源をメモリ管理用ハードウェアとオペレーティングシステムにより管理と保護がなされたメモリ資源として扱うことができ，従来のネットワーク機構で見られる種々のオーバヘッドを排除できることを示した．また，LMMC モデルを実現するためのアーキテクチャとオペレーティングシステムについて考察した．

参考文献

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson, "A Case for NOW(Networks of Workstations)," *IEEE Micro*, Vol. 15, No. 1, pp. 54–64, (1995).

[2] John L. Hennessy and David A. Patterson, "Computer Architecture: A Quantitative Approach (2nd Edition)," Morgan Kaufmann, (1995) pp.563–629.

[3] 中條拓伯, 中野智行, 松本尚, 小畑正貴, 松田秀雄, 平木敬, 金田悠紀夫, "分散共有メモリ型超並列計算機 JUMP-1 におけるスケーラブル I/O サブシステムの構成," 情報処理学会論文誌, Vol. 37 No. 7, pp. 1429–1439, (1996).

[4] A. Saulsbury, F. Pong, and A. Nowatzyk, "Missing the Memory Wall: The Case for Processor/Memory Integration," *Proceedings of 23rd International Symposium on Computer Architecture*, pp. 90–101, (1996).

[5] 松本尚, 平木敬, "Memory String Architecture – メモリウォールを越えて –," 情報処理学会研究会報告 *96-ARC-120-1*, pp. 1–6, (1996).

[6] William J. Dally, and John Poulton, "Transmitter Equalization for 4Gb/s Signaling," *Proceedings of HOT Interconnects IV Symposium*, pp. 29–39, (1996).

[7] Dror G. Feitelson, "A Survey of Scheduling in Multiprogrammed Parallel Systems," *IBM Research Division Research Report RC 19790*, (1994).

[8] Patrich. G. Sobalvarro, and Willaim. E. Weihl, "Demand-based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors," *Proceedings of the IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 63–75, (1995).

[9] Andrea C. Dusseau, Remzi H. Arpaci, and David E. Culler, "Effective Distributed Scheduling of Parallel Workload," *Proceedings of the ACM SIGMETRICS conference*, (1996).

[10] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," *Proceedings of the 19th International Symposium on Computer Architecture*, pp. 256–267, (1992)

[11] Matthias A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, and Edward W. Felton, "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pp. 142–153, (1994).

[12] Greg Buzzard, David Jacobison, Scott Marovich and John Wilkes, "Hamlyn: a high-performance network interface with sender-based memory management," *Proceedings of HOT Interconnects III Symposium*, (1995).