汎用超並列オペレーティングシステムカーネル SSS–CORE — 資源管理およびメモリベース通信の実現と評価 —

Implementation and Evaluation of resource-management and memory-based communication facility of the general-purpose massively-parallel operating system: SSS–CORE

松本 尚 [*1] tm@is.s.u-tokyo.ac.jp
信国 陽二郎 [*2] yo@artstech.co.jp
渦原 茂 [*3] uzu@axe-inc.co.jp
竹岡 尚三 [*3] take@axe-inc.co.jp
平木 敬 [*1] hiraki@is.s.u-tokyo.ac.jp

[*1] 東京大学 大学院理学系研究科 情報科学専攻
[*2,] 株式会社 アーツテック [*3] 株式会社 アックス

SSS–CORE はワークステーションクラスタおよび分散メモリ型並列計算機を対象とした汎用超並列オペレーティングシステムである。本オペレーティングシステムはマルチユーザ / マルチジョブという汎用計算機に不可欠な機能を実現しつつ、この汎用性と従来相容れなかった並列計算の効率を落さないための各種機構機能を実現する。

本稿では SSS–CORE の特徴である、資源状況情報を利用したスケジューリング方式、詳細確率モデル上で、具体的なメモリ管理方式 / アクセス頻度 / アクセスコストを付加したシミュレーションによる評価、汎用ネットワークを用い超高速通信を実現したメモリベース通信ファシリティの概要と評価を述べ、ワークステーションクラスタ版 SSS–CORE の実装方式ならびに開発状況について述べる。

1 はじめに

SSS–CORE [11]は NUMA 型並列分散計算機を対象とした汎用超並列 OS である。時分割 とパーティショニングを併用したマルチユーザ / マルチジョブ環境を提供し、メモリ間通信を基本とする低レイテンシ高スループットを実現するメモリベース通信ファシリティを使用することによる並列アプリケーションの効率良い実行の実現を目的としている。

汎用 NUMA 型並列分散計算機上でのマルチユーザ / マルチジョブ環境における並列アプリケーションの実行効率を向上は、単一ユーザ / 単一ジョブによる使用を前提とした従来からのＭＰＰ実行に比べ：

1. マルチユーザ / マルチジョブ環境を実現するパーティショニングとスケジューリングが、並列プログラムを作成、コンパイル時に想定したシステムの動作状況を満たされないことによる同期時におけるアイドル時間増加
2. 各スレッドにスケジューリングによる割り当てられた要素プロセッサ間の距離増加
3. 実行時のプロセス / メモリ移動による非効率
4. 仮想化され、保護されたプロセス間通信実現のオーバーヘッド

を代表とする困難点が発生する。

上記問題点中 (1),(2) を支援するため *SSS–CORE* では、ユーザ / カーネル間での情報の受渡しのための機構を提供し、この機構を利用することにより実メモリの使用状況のスケジューリングへの反映、ならびにユーザからのスケジューリング制約の利用、そして優先度を利用した資源の公平な割り当てという方策による実行効率の向上を実現している。またさらに、後述するページの分類に則ったメモリ置換方式も、同様の機構を利用することで可能となる。

また、問題点 (3),(4) は、通信をメモリベースで実現し、メモリに関する仮想化と保護を直接的に利用することにより支援することが可能である。

文献 [9]では、 *SSS–CORE* の概要および実現方式を述べた。本稿では、 *SSS–CORE* の特徴である、資源管理方式と密接に関連付けられたスケジューリング方式と、メモリベース通信ファシリティの実現方式の詳細と評価を報告する。

2 *SSS–CORE* の資源管理機構

2.1 *SSS–CORE* のスケジューリング方式

SSS–CORE は、ユーザ / カーネル分離の 2 レベルスケジューリングを採用する。カーネルは各プロセスが提示するスケジューリング制約に従って最も適当な資源

† 本研究は情報処理振興事業協会「独創的情報技術育成事業」の一環として行われたものである。

を、階層化管理された資源情報をもとにプロセス単位に割当てる。階層資源管理を実現するため、*SSS–CORE* では仮想的な資源管理木構造を用意する。本稿においてはこれを資源管理木と呼ぶ。資源管理木では論理的に各ノード毎に、ノードをルートとする部分木の領域に含まれるプロセッサと実ページの総数および、フリーなプロセッサと実ページの総数、ならびにプロセス毎に ID および使用しているプロセッサと実ページの総数を記録する。またルートノードはこの他に、プロセス毎にスケジューリング制約、優先度、ホームノード (後述) を保持する。

現在の並列化コンパイラは、資源が単一プロセスに占有されていると仮定している。我々の目標はそのようなコンパイラが吐くオブジェクトコードを汎用環境上で実行することにある。マルチユーザ / マルチジョブ汎用環境では、各プロセスの資源への様々な欲求を満たすことで効率化が可能となる。スケジューリング制約は、プロセスがカーネルに資源への欲求を伝えるために利用される。スケジューリング制約の例として台数固定制約では、プロセスはカーネルに対して自分が必要とする要素プロセッサ数を一定数指定する。また可変台数制約は少しずつでもプロセスにプロセッサを割り当てることが可能となる。今回のシミュレーションではプロセスの並列度は変化しないので、スケジューリング制約としては台数固定制約を用いる。

3 シミュレーション・モデル

本稿では評価の精度向上を目的としてこれまでの研究で利用したモデル [12] [14]を拡張し、 OS レベルのシミュレーションのための簡略化された詳細確率モデルを構築した。

対象とするのは分散メモリ環境である。メモリとプロセッサの対をクラスタとし、相互結合網は木構造を仮定する。ネットワークは二重化されたスイッチングネットワークで、スイッチングノードにおいてメッセージをバッファリングするワンホップ通信を行なう。ノードは各出力に対して、ノードへの入力数分のエントリを持つバッファを持つ。ページ移動およびその他の通信の二通りについて、ノード間の 1 ホップを通過するための基本コストをパラメータとして与える。ページの移動およびその他の通信に対してはそれぞれ、ページ移動基本コストおよび通信基本コストと各枝の帯域倍率とから算出したコストを付加する。進行方向のバッファおよび枝が空いていない場合は待たされる。

プロセスは要求プロセッサ台数と同数のスレッドにより構成される。並列度は一定で、生成時から終了時まで変化しない。一クラスタにおける並列プロセスの実行コンテキストをスレッドと呼ぶ。各スレッドはプロセス内に共有空間と固有の非共有空間を持つ。各プロセスの両空間には、それぞれ別々のページ単位のメモリ参照頻度表を与える。各空間における参照列は、参照頻度表における総頻度に対する各ページの参照数を比として確率的に生成する。

共有空間に対しては分散共有メモリシステムを構築する。一貫性管理は sequential consistency メモリモデルに則ったアップデート方式で、従ってページは置換されない限りメモリに残る。共有ページへのアクセスはアクセス時にキューに繋がれ、ライトの場合にはオーナーからのアップデートの送信とそれに対する Ack の返信とが行われる。またメモリ参照コストは、クラスタ内 $\ll$ クラスタ間 $\ll$ 2 次記憶という仮定を反映するように設定する。スケジューリングによりスレッドの割り当て位置が変化した場合には、ローカルページはネットワークを介して on-demand に移動する。プロセスの割り当て領域が変化した場合には、現在の割り当て領域外に存在する共有ページはコスト 0 で消去する。

プロセスの実行はクロックベースの確率モデルで、同期待ち又はページ待ち状態でなければ各クロック毎にメモリ参照を行なう。またプロセスのスレッドは、与えられた有効実行クロック間隔でランダムな組合せのスレッド間で同期を起こす。同期は、同期点へ到達したスレッドからランダムに選択されたバリアのルート・スレッドへの送信と、それに対する返信とで実現される。ここで有効実行 (時間 / クロック) とは、ページ / 同期待ち以外の動作をした (時間 / クロック数) のことである。

3.1 スケジューリング方式

前記シミュレータ上に 5 つのスケジューリング方式を実装した。各方式共に、タイムスライス毎に資源使用のエージングを考慮した優先度計算 [13]を行ない、優先度の高いプロセスから順に、それぞれ以下に示す方法でスケジューリングを行なう。

algo0 ランダムに必要な数だけクラスタを選択し割り当てる方式

algo1 システム中のクラスタを端から順に必要なだけ割り当てる方式

algo2 初めにホームノード以下の領域に割当を試み、失敗した場合にはルートノード以下の領域で、自分のページが存在する領域から割り当てる方式

algo3 ホームノード以下の領域のみでページのある領域から割り当てる方式

algo4 ホームノード以下の領域で自分のページが存在する領域のみを割り当てる方式

ただし上記でホームノードとは資源管理木上のノードで、プロセスの要求台数以上のプロセッサを含む部分木に対応するもので、最下位レベルに位置するもののことである。すなわちホームノードは、一つ前のスケジューリング時の割当領域を代表している。図 1にホームノードの例を示す。図中でプロセス A のホームノードは、自分のページを保持する領域 4 を包含する部分木のルートに当たるノードである。

台数固定スケジューリング制約を用いているため、algo2, 3, 4 では割り当て対象領域に必要台数を確保できないプロセスのスケジュールは諦める。初めに必要台数の確保を試みる領域と、それに失敗した場合の対処法が異なっている。　またプロセスに割り当てられたクラスタへのスレッドのスケジューリングは、以下の通りである。

- 前回の割当と重なるクラスタには前回と同じスレッドをスケジュールする

Table 1　プロセス・セットの概要

プロセスセット	プロセス数	並列度 (複数の場合の個数)	総並列度	総非共有空間サイズ [pages]	総共有空間サイズ [pages]	メモリの最大要求度 [総実メモリ比]	同期間隔 [clocks]
A	12	16,36,48,50(2),64,70 96,100,128,192,200	1050	35500	10120	1.00	1000–2000
K	12	48,96,208,256	1184	49920	6360	1.30	1000
L	11	64(5),128(2),192(2),256(2)	1472	68480	7280	1.68	1000
M	23	16(16),50(6),256(1)	812	35480	30320	1.70	1000

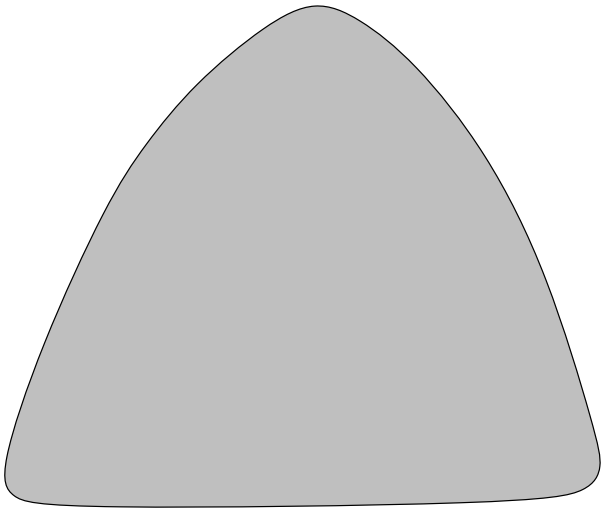

Figure 1　資源管理木とスケジューリング対象領域

- それ以外のクラスタには、残りのスレッドを順にスケジュールする

前節で述べたように、同一スレッドが同一クラスタで実行されればローカル空間は再利用される。スレッドのスケジュール位置が移動した場合には、ローカル空間のページはクラスタを越えてネットワーク上を移動しなければならない。従って割り当て領域がなるべく重なるようにした方が、ローカルページの移動が少ない。

3.2　ページ置換方式

分散メモリ環境システム上で分散共有メモリを想定すると、参照頻度および再アクセス時のアクセスコストの違いによりページを分類することが可能である。ページの分類ができれば、参照頻度が少なく再アクセス時のコストが小さいページから置換対象とする効率的なページ置換方式が可能となる。

並列プロセスの場合一般には共有領域の参照率は非共有領域に比べて小さい。再アクセス時のコストについては、ネットワーク内へのアクセスか、二次記憶へのアクセスであるかによって分類が可能である。さらにそのコスト差はオーダーとして違うと仮定できる。そこで、走行する可能性の低いプロセスの共有コピーページから置換対象とすることで、性能向上をはかることが可能である。以上を踏まえてページの分類ならびに、分類間の置換対象とする順序を次のように与える。

1) 他のプロセスの共有コピーページ、2) 実行中プロセスの共有コピーページ、3) 他のプロセスのラストワン・ページ、4) 他のプロセスのローカルページ、5) 実行中プロセスのラストワン・ページ、6) 実行中プロセスのローカルページ、ただしこれらの厳密な順序関係は、共有・非共有各領域についての参照頻度や容量、プロセスの組合せ、ネットワーク及び二次記憶の性能に依存する。

今回のシミュレーションでは、次の二つの置換方式を実装し比較を行なった。

置換方式 **0**　ページの種類 / プロセスの区別なしに単純な LRU 順

置換方式 **1**　スケジューリング優先度の低いプロセスから 3.2節におけるページの種類順に選択。各プロセスの共有 / 非共有空間毎にページの LRU 順を管理

ただしどちらの方式も、一貫性処理中の共有ページの場合は置換対象としない。

Table 2　各種コストおよびパラメータ

項目	値
プロセッサ数	256 台
ディスクアクセス	100000 clk
ページ移動基本コスト	500 clk
通信基本コスト	50 clk
1 メモリのページ数	400 pages
1 ページのサイズ	4096 Byte
総メモリ量	409.6 Mbyte
1 quantum	1000000 clk

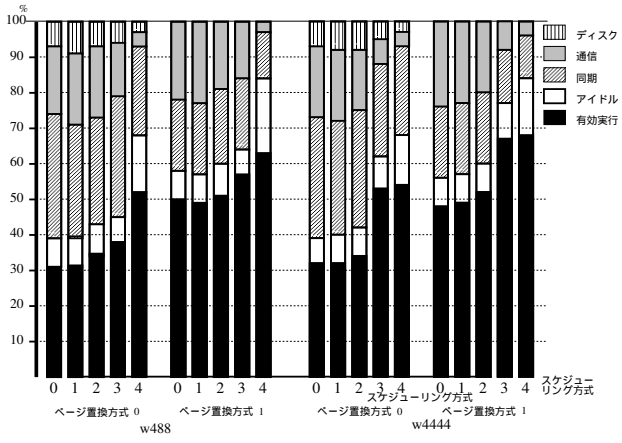

Figure 2　プロセスセット A のシミュレーション結果

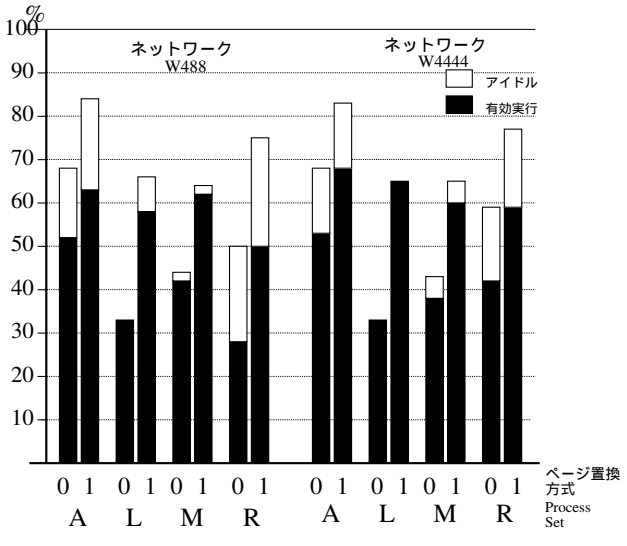

Figure 3　ページ置換方式の比較

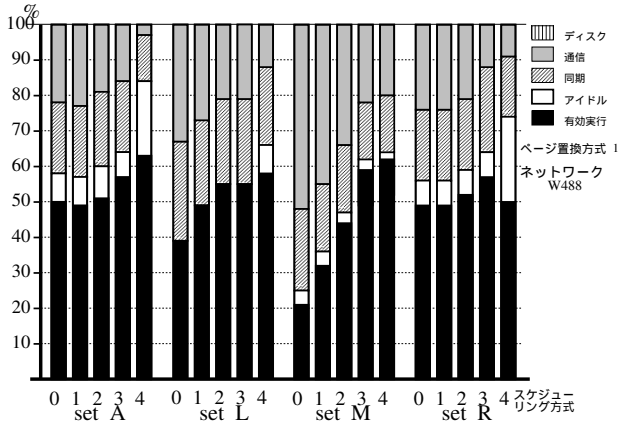

Figure 4　プロセスセットA、L、M、Rのシミュレーション結果

4　結果および考察

実験としては表 2のようなパラメータを採用し、 256 台プロセッサの 2 つのシステム (ルート階層から順に 8-8-4 分岐する 3 段ネットワーク [w488 と表記]、および同 4-4-4-4 分岐する 4 段ネットワーク [w4444]) において、表 1の各プロセス組を実行した。実験は、プロセスが一つ以上終了する (20 タイムスライス分の有効実行後) か、 100 タイムスライスの実行を終えるまで継続した。

プロセスセットＡに対する実験が終了した時点での評価結果を図 2のグラフに示す。左列のグラフが w488、右列が w4444 の場合の結果である。また、ページ置換方式の比較結果を図 3に示す。図 3ではスケジューリング方式４、 w488 のネットワーク構成を用いている。プロセスセットＡおよび図示していないがプロセスセットＬ、Ｍ、Ｒについて常に安定してスケジューリング方式４が最も優れた性能を与えることに基づいている。次に、ページ置換方式の比較結果を図 4に示す。ここでも、スケジューリング方式４を用いている。

4.1　スケジューリング方式について

A 以外のプロセス組では、共有ページのコピーが生成されることでメモリ置換が発生している。汎用な分散並列環境ではこのようなシステムの利用状況が自然であろう。 algo4 のようにプロセスの割り当て領域を固定した場合、プロセス組の組合せの性質から十分にプロセッサを活用ないため、実際のシステムではアイドル時間が減る可能性として、以下のことが挙げられる。

1. 台数可変制約 (*SSS–CORE* には実装予定) の採用により、プロセッサ台数について自由度の高いスケジューリングを行うことができる。
2. 特定のプロセス組にとってスケジューリングの “穴” となる領域を、他のプロセスが有効利用することができる。

前述したように、 algo2, 3, 4 の違いはメモリページの位置に固執する度合である。実験結果は、プロセスを実ページを所有する領域に割り当てる傾向が高い方式ほど良いことを物語っている。 algo2, 3, 4 の順にページの移動量が多いわけだが、図 2を見ると、その順で通信 (ページ処理) や同期が占める時間が大きくなっていることが分かる。これは、参照頻度の高い非共有空間ページの移動が少なくなるようにスケジューリングする効果が影響したと推定される。また、メモリ溢れがほとんど起きない[1]プロセス組 A の場合についても、どの指標を用いても algo4 が最良となっている。それ以外のスケジューリング方式は、結局ページの移動コストが高くついて性能が出ていない。

4.2　メモリ置換方式について

メモリ置換方式については方式 1 が方式 0 に常に勝っている。方式 1 ではプロセスの共有ページが置換されるのに対して、方式 0 ではプロセスのローカル空間のページやラストワン共有ページが置換されている。メモリへの要求度が軽いセット A, K, L では、他人のコピーページを置換することしかない。もう少しメモリへの要求度が重いセット M では自分のコピーページを置換することもあるが、より参照され易い非共有ページを追い出さないことで効率の劣化を防いでいる。ここにあげたケースでは、共有コピーページを優先的に置換する効果が極めて大きくなっているといえる。以上のことから NUMA 環境においては、ページをアクセス頻度および置換後の再アクセスコストを考慮した分類に基づいたメモリ置換を行うことで、高効率な汎用環境を提供することが可能であることが示されている。

今回の実験でもっとも性能の良い、スケジューリング方式 alog4 で置換方式 1 の場合の結果で、最大有効実行率が 65% から 85% 程度である。共有ページの一貫性管理が Sequential consistency に従うため、ページのアップデート処理自体のコストより、共有領域へのアクセス処理における先行アクセス処理の終了待ち時間が非常に大きい。

これは緩和されたメモリモデルの採用により小さくすることが可能である。ただ緩和されたメモリモデルを使

1　プロセスの割り当て領域の重なり具合で、一部領域においてメモリ置換が発生している。

えば、ページを移動した時のコストのプロセスに対する悪影響の効果は小さくなる。

また本稿におけるシミュレーション・モデルでは、メモリアクセスとスレッド間の同期とが独立した事象としてモデル化されているため、ページ処理に費やされる時間の減少は、そのまま同期時間の短縮につながる。

以上のことから、本稿における実験での結果の値の低さが、並列環境におけるシステムの汎用な利用形態を否定することはない。

5 資源情報を活用したスケジューリングの関連研究

時分割を利用した並列環境向けのスケジューリング方式としては、 gang scheduling の優位性が報告されている [1]。並列プロセス単体の実行性能を考慮した場合にも、同期待ちを極力減らすことが可能な gang scheduling は有効である。資源情報を利用しない matrix algorithms [5] は、プログラム実行単位のプロセッサへのマッピングにおけるヒューリスティックスに頼る。そのため資源の占有状況を無視したマイグレイションや、processor fragmentation が問題となる。メモリへのアクセスコストが距離によって異る分散環境では、スケジューリングに実資源の使用状況を反映させることによりコストの高いアクセスを減少させる高効率化が必須である。

資源情報を利用する方式に、集中記憶型並列計算機向けの cache affinity 方式 [6]があるが、容量の小さなキャッシュでは効果が出ない。大規模分散環境上で協調した資源管理を行うには、拡張性や管理効率の点から階層化された枠組が必要である。階層的なスケジューリングを行うものに DHC [2]と Intel Paragon [4]がある。DHC は Buddy システムによる階層化管理を行い、ラウンドロビンを基本としており、プロセス数に基づいた負荷情報 [3]のみを利用する。 Paragon では柔軟なプロセッサ分割が可能であるが、資源情報は利用しない。また各プロセッサ上でのスケジューリングもカーネルが実行する。

6 メモリベース通信ファシリティの概要

分散メモリ型の並列分散計算環境を大型 / 超大型計算機システムとして利用するためには汎用性 (マルチユーザ・マルチジョブ) の導入が不可欠である。しかしながら、汎用性導入による保護機能の実現や実資源の仮想化は、並列分散処理の高効率化と相容れない要素があり、並列分散処理の大幅な性能低下の原因となる。

並列処理および分散協調処理の最大の特徴はプロセッサおよびノード間の通信と同期であり、高速なユーザレベル通信同期なしには効率の良い並列実行環境はあり得ない。そして、このユーザレベル通信同期は保護と仮想化の要件十分に満足しつつ高速に実装される必要がある。我々はユーザレベル通信専用ハードウェアを用いることなしに、これらの条件を満たすソフトウェアメモリベース通信 (**MBCF**: Memory-Based Communication Facilities) [7] [8] [15]を考案した。

7 MBCF の特徴

集中共有メモリを持つ並列計算機では、プロセッサは共有メモリ領域への通常のメモリアクセスで通信同期を行う。ユーザプログラムはマッピングされたページにしかアクセスできないため、ページ管理機構によってジョブ間の不当干渉を排除することが可能である。つまり、ユーザレベルの通信 (同期) を通常のメモリの load/store で実現しており、保護に関してはプロセッサのメモリ保護機構の方式がそのまま流用可能である。

しかし、集中共有メモリ型並列計算機は集中共有メモリへのアクセスがボトルネックとなり、プロセッサ台数の大規模なものを製造することが困難である。そこで、松本は従来のページ管理機構を遠隔メモリアクセスに拡張した Memory-Based Processor (**MBP**) [10]を考案した。 MBP を持つ分散メモリ実装の並列計算機やワークステーションクラスタ (**NOW**: Network of Workstations) では、集中共有メモリ型計算機と同様に通常のメモリアクセスとして高速かつ保護され仮想化されたユーザレベル通信同期が実現できる。

しかるに、 MBP タイプのハードウェア付加機構は現時点において一般的ではなく、ソフトウェアの助力なしに主記憶を大容量キャッシュとして流用するためには主記憶に付加的なタグ情報を持たせる必要がある[2]。また、MBP は主要素プロセッサのメモリアクセス動作と密に協調して働くため、 MBP の実装はプロセッサのメモリ周りの実装に依存してしまう可能性がある。

これらの理由から、汎用並列分散オペレーティングシステムの開発に当たって、我々は集中共有メモリや MBP と同様な通信同期ハードウェアを仮定しない分散メモリ実装の並列計算機環境 (NOW を含む) において実現可能な、高速かつ保護され仮想化されたユーザ通信 / ユーザ同期を考案する必要にせまられた。これに対して我々が出した回答が MBCF である。

MBCF は以下のような方式により、基本的に MBP の動作と機能を高速ソフトウェアエミュレーションで実現している。

- 高性能プロセッサのローカル (キャッシュヒット) 処理の高速性
- 余分な処理をしない軽い送信専用システムコール
- 余分な処理をしない受信割り込みルーチン
- コンテクスト識別子を含む複数コンテクストの混在できる TLB
- 軽いアドレス空間切替えハードウェア
- ページエイリアス機能
- 物理アドレスタグを持つプロセッサキャッシュ
- 論理アドレスによる通信相手空間の直接操作

MBCF の定性的な議論は文献 [8]を、 100baseTX 版および 10baseT 版の MBCF の実装技術に関する詳細は文献 [15]を参照されたい。なお、専用ハードウェアの不要な高性能分散共有メモリシステム「非対称分散共有メ

2 false sharing の問題を緩和するために必要である。

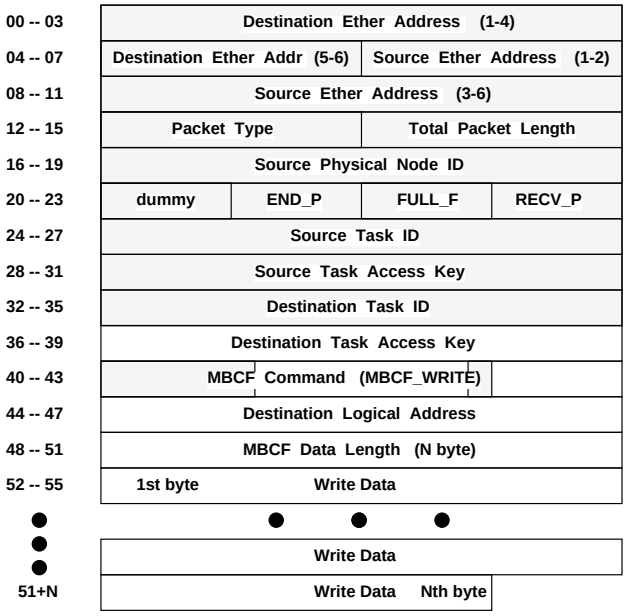

Figure 5　単一 MBCF_WRITE のパケット

モリ（**ADSM**）[8]」にも MBCF が使用され、MBCF と同じ方針が採用されている。

8　MBCF のパケットフォーマット

文献 [15]で公開したパケットフォーマットに関して、ユーザレベルのパケットコンバイニング（パケットマージ）の性能を改善するために、若干の変更が施されフォーマットは現在以下のようになっている。

図 5に MBCF_WRITE を行うイーサパケットを示す。便宜上、パケットの先頭を 0 番地として byte 単位のアドレスによってパケット内の位置を示すことにする。0 番地から 19 番地までは Ether 上の IP パケットと同形式をしている。ただし、Packet Type として IP 等の他のプロトコルと衝突しないものを使用し、Source Physical Node ID は IP アドレスではなく MBCF 用の独自のノード ID を採用している。20 番地は空いている。21 番地から 23 番地までの 3 個の 1byte 長の制御データは通信到着保証と FIFO 性保証と Acknowledge 制御のために使用されている。24 番地、28 番地にはそれぞれパケット送信元のタスク ID とアクセスキーが格納され、32 番地にはパケットの宛先のタスク ID が格納される。これら、0 番地から 35 番地までのパケットの内容は必ずカーネルレベルのルーチンによってセットアップされる。宛先のタスク ID と宛先ノード ID およびその ether address はユーザが指定した宛先の論理タスク番号からオペレーティングシステム内において表引きで求められる。36 番地から 51 番地までのデータはユーザによる直接パケット生成時にはユーザレベルでセットアップされる。ただし、40 番地の MBCF Command の上位ビットはパケット再送制御等に使用されるため、カーネルルーチンで正しく再設定される。図中のハッチ部分はカーネルが値を設定する領域であり、ハッチが施されていない部分はユーザが値を自由に設定できる。36 番地

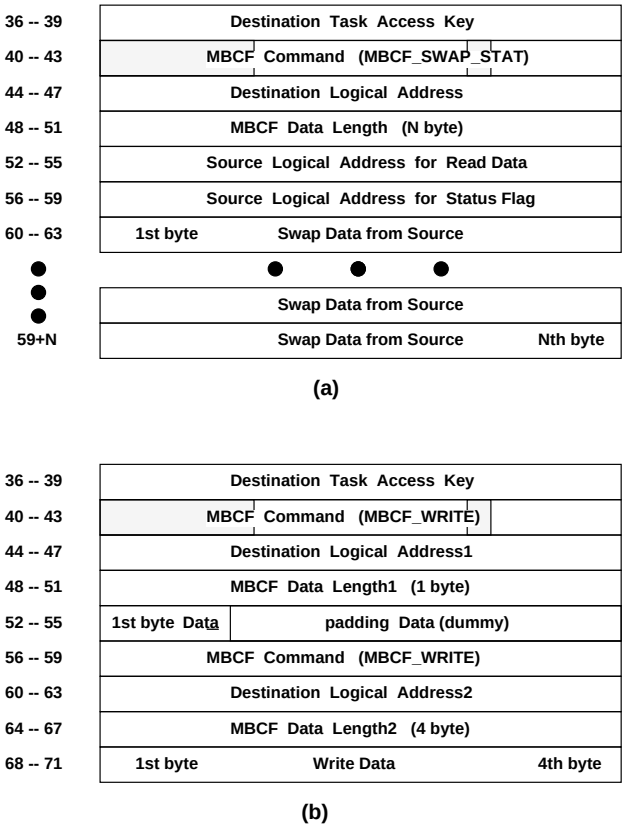

Figure 6　MBCF パケットのバリエーション

には宛先タスクのアドレス空間へのアクセスキーが格納され、40 番地には MBCF の機能を示すコマンドが書かれる[3]。44 番地に宛先タスク内の書き込みを開始する論理アドレスが格納され、48 番地に書き込みデータ長が格納される。52 番地からは実際の書き込みデータ続く。保護のため、アクセスキーが間違っている場合や宛先論理アドレスが宛先タスクにおいてアクセス可能になっていない場合は単に通信（この場合は書き込み）が失敗するだけではなく、セキュリティ保持のためのペナルティが発信元等に課される場合もある。

図 6に MBCF パケットの他の実例を示す。0 番地から 35 番地までは図 5の MBCF_WRITE パケットと同一であるため省略してある。図 6(a) には遠隔ノード上の複数データを不可分に SWAP するコマンド「MBCF_SWAP」のバリエーションの一つである「MBCF_SWAP_STAT」を示す。MBCF_SWAP_STAT はこの処理が終了したことを発信元タスクのフラグ変数（56 番地にポインタ）に 1 を書き込むことで通知する。また、SWAP で遠隔操作対象のタスク内のデータを返送するためのポインタ（送信元タスク内の論理アドレス）が 52 番地に含まれている。コマンドバリエーションを示すことは本稿の主旨ではないので、全コマンドの詳細な説明は他稿に譲る。図 6(b) には同一タスク宛の複数の MBCF コマンドが 1 パケットのまとめられた形式のパケット

3　MBCF コマンドと宛先タスク ID および宛先アクセスキーのパケット内位置が以前の発表から変更されている。

を示す。図では同一タスクへの 1byte と 4byte の二つの MBCF_WRITE を 1 パケットにコンバイニングして（まとめて）いる。複数の MBCF Command が含まれていても再送制御やフロー制御等は 40 番地の最初の MBCF Command の上位ビットのみで行うため、任意の MBCF コマンドが一つのパケットにまとめられるわけではない。複数の MBCF コマンドがまとめられる（コンバイニング可能）かどうかは、返送パケットの有無や返送パケットのサイズおよびマルチキャスト動作の経路等で決まる。

MBCF の発信用の軽い専用システムコールには二種類ある。両者とも、送信先は論理タスク番号で渡され、35 番地以前のパケット領域はカーネルによって設定される。一つのシステムコールは宛先タスクの操作対象論理アドレス等のパラメータと パケット内データ（MBCF_WRITE の書き込みデータ等）へのポインタが構造体で渡されるものである。そして、もう一つはユーザがパケットのユーザ設定領域（図 6に相当する 36 番地以降）を完全に組み立ててから組み立てたパケットへのポインタをシステムコールに渡す。前者はユーザがすでに使っているメモリ領域をデータとして転送する場合に、ポインタ渡しできるのでユーザモード内におけるコピーが省略できる。ただし、複数の MBCF コマンドをコンバイニングする場合はシステムコール内のパケット組み立てのための処理量が増える。後者は、パケット組み立てが完全にユーザ（コンパイラ、ユーザモードのランタイムを含む）に任されているので、MBCF コマンドのコンバイニング等が自由に行える。

現在の 10baseT または 100baseTX による MBCF の実装では再送のためパケットのコピーをノード内に残す必要があるため、二種類のシステムコール共にデータのカーネル内におけるパケットデータのコピーが避けられない[4]（ユーザに再送バッファの管理まで任せればコピー不要となるが、これではユーザ側の処理が非常に繁雑）。ハードウェア的にデータ転送が保証されていれば、後者のユーザによるパケット組み立てのシステムコールでは、カーネル内におけるデータコピーが完全に省略できる。

9 性能評価

本稿で述べる MBCF の評価は以下の環境で行った。使用した NOW 環境は Axil 320 model8.1.1 (Sun SS20 互換機, 85MHz SuperSPARC CPU × 1) を 8 台、10baseT のハブで接続している。この 8 台のうちの 5 台は Sun Microsystems 社製の Fast Ethernet SBus Adapter 2.0 を追加して、100baseTX のハブで Fast Ethernet 接続されている。オペレーティングシステムは MBCF および ADSM のテストベッドとして開発された汎用超並列超分散オペレーティングシステム SSS–CORE [11] [7] Ver.1.0 を使用した。SSS–CORE Ver.1.0 にはこれまでに公表した MBCF の機能（保護やセキュリティ面を含む）がフルスペック（Memory-Based FIFO, Memory-Based Signal 等を含む）で実装されている。

9.1 100baseTX 版 MBCF の基本性能

MBCF の基本性能に関して 2 ノード間の通信性能測定プログラムを使用して評価を行った。

時間は 0.5μsec 単位の時計で性能測定プログラム内でソフトウェア的に計測した。ただし、この時計の読み出し 1 回に約 1.2μsec のオーバヘッドがハードウェア構成上かかるため、計測した値は開始と終了時の二回の時計読み出しにより、約 1.2μsec だけ大きな値になっている。なお、いくつかのケースについてはロジックアナライザを使用した厳密な波形測定に基づく時間測定を後に示す。

基本性能は Round-trip タイム、送信システムコールのオーバヘッド、Peak Bandwidth で示す。Round-trip タイムの表においては三種類の MBCF コマンドの種類別に性能を示す。これらの表における各通信コマンドの機能と測定条件は以下の通りである。

- **MBCF_WRITE**
  通信要求時に data を運び、対象メモリに書き込み後、書き込み完了を要求元のタスクに通知する。時間の測定は通信要求のシステムコールの直前から書き込み完了のフラグをスピンウェイトで検知するまでである。
- **MBCF_READ**
  通信要求時にアドレス・コマンド情報のみを運び、対象メモリを読み出し後、データをパケットで転送し指定されたバッファに格納の後に読み出し完了のステータスを指定アドレスに書き込む。時間の測定は通信要求のシステムコールの直前から読み出し完了のフラグをスピンウェイトで検知するまでである。
- **MBCF_SWAP**
  通信要求時に data を運び、対象メモリから古い data を読み出し後、運んで来た data を書き込み、読み出した data をパケットで転送し指定されたバッファに格納の後に SWAP 完了のステータスを指定アドレスに書き込む。時間の測定は通信要求のシステムコールの直前から SWAP 完了のフラグをスピンウェイトで検知するまでである。

表 3に 100baseTX による MBCF の Round-trip タイムを示す。

Table 3 100baseTX による MBCF の Round-trip タイム

data size (byte) コマンド種別	4	16	64	256	1024
MBCF_WRITE (μs)	51	54	60.5	88	200
MBCF_READ (μs)	51	54.5	61	88	200.5
MBCF_SWAP (μs)	51.5	58	71.5	125.5	351

4 MBCF 現実装ではこのコピーを最小限の 1 回に抑えている。

参考までに表 4に 10baseT によるメモリベース通信の Round-trip タイムを示す。

Table 4　10baseT による MBCF の Round-trip タイム

data size (byte) コマンド種別	4	1024
MBCF_WRITE (μs)	213	1080
MBCF_READ (μs)	228	1090
MBCF_SWAP (μs)	229	1930

次に、ソフトウェアで測定した送信時オーバヘッドの表 5を示す。測定は遠隔書き込み時のシステムコール呼び出し直前から呼び出しから戻るまでを前出の 0.5μsec の時計で計測した。なお、 10baseT の場合も、送信ルーチンが 100baseTX と通信ハードウェアレジスタの操作部分以外同じであるため、本オーバヘッドはほとんど同じである。

Table 5　100baseTX による MBCF の送信オーバヘッド

data size (byte)	4	16	64	256	1024
送信コスト (μs)	5	5.5	6	8.5	20

表 6に 100baseTX と 10baseT のメモリベース通信のピーク転送性能を示す。測定は MBCF_WRITE をデータサイズを変えながら実測した。表中の値は Ethernet のヘッダーや MBCF のプロトコルデータを省いた遠隔書き込みの転送データの正味サイズだけから計算した値である。 Round-trip タイムの測定ではないので、通信ごとの操作完了フラグの返送は行っていない[5]。転送は同一アドレスに対してバースト状に行い、ネットワークが過負荷状態にならないように 16 転送毎に完了フラグを返送させてチェックするアクノリッジによって流量を調節した。ユーザレベルの保護され仮想化された通信方式としては良好な値であり、ほとんど生のハードウェア性能 (100baseTX: 12.5Mbyte/s, 10baseT: 1.25Mbyte/s) を使い切ることに成功している。

9.2　ロジアナによる基本性能の厳密測定

100baseTX による MBCF は非常に高速であるため、アクセスに 1.2μsec を要する 0.5μsec 刻みのタイマでは正確な測定は難しい。そこでワークステーションと Fast Ethernet board にロジックアナライザを接続して波形の計測を行い、正確な時間を求めた。キャッシュのヒット状況によってコストが変動する。以下の測定値はキャッシュエントリのスラッシングが発生しない状況での値である。システムコールならびに受信ルーチンのオーバヘッドの測定はプロセッサチップ (SuperSPARC-II) にある supervise アクセスピンを測定することで行った。

図 7にメモリベース通信の成否を返答するオプション付きの 4byte MBCF_WRITE 時のロジック信号波形を示す。この例では成否返答オプション付きの遠隔書き込

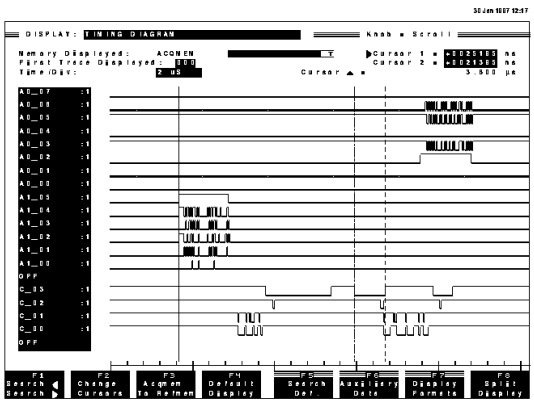

Figure 7　4byte 遠隔書き込み時の送信側信号波形

みが終了する度に、同じアドレスへの同じメモリベース通信を繰り返している。この信号波形は遠隔書き込み要求側の波形であり、信号 A0_03 から A0_06 は送信データを A1_01 から A1_04 は送信側の受信データ (ステータスの返答パケット到着) を示している。 C_03 は low で supervisor アクセスモードに居ることを示す。図 7の C_03 の最初の low 期間はステータスの返送による受信割り込み処理区間 (8.0μsec) を示す。 2 回目の low 期間が遠隔書き込み送信のためのシステムコール期間を示す。この図では **3.80μsec** である、他の通信箇所では **3.40μsec** も観測された。ソフトウェアによるタイマ測定では、 5.0μsec を下回ることがないのは計時カウンタの読み出しオーバヘッドによるものである。なお、 3 回目の C_03 の low 区間は Fast Ethernet Adapter の送信終了割り込みによる処理期間である。

同一条件において、送受信つまり書き込み要求から成否返答パケットの処理終了までの時間を測定した。測定値は **50.0μsec** 近辺に集中しており、最良値は **48.40μsec** であった。この値は 100baseTX の MBCF の Round-trip タイムに他ならない。

同じく 4byte MBCF_WRITE を成否返答オプションなしの条件を用いて、受信側のオーバヘッドを測定した。キャッシュミスの発生する初回のアクセスを除き、**6.40μsec** 程度に観測値が集中する。この値が受信割り込みルーチンのオーバヘッド値である。

9.3　MBCF と MPP の通信性能比較

以下に、高並列計算機 (MPP) のソフトウェアを含んだ通信性能を参考に表 7に掲げる。なお、これらは保護および仮想化の度合が MBCF と比べて低く[6]、機能的にも大幅に劣るので、本来は定性的側面から比較対象から除外されるべきものである。

表中の SSAM と MBCF 以外は専用通信ハードウェアを持つ並列計算機システムであり、通信ハードウェア

5　完了フラグの返送は MBCF コマンドのオプションの一つである。

6　ノードを跨るギャングスケジューリングが強制されたり、通信ネットワークが一つのアプリケーションに占有されたりする。

Table 6　100baseTX/10baseT のメモリベース通信の Peak bandwidth

data size (byte)	4	16	64	256	1024	1408
100baseTX (Mbyte/s)	0.29	1.06	4.03	8.28	10.86	11.24
10baseT (Mbyte/s)	0.04	0.17	0.48	0.89	1.13	1.17

自体の性能は今回の MBCF 実装が使用した 100baseTX よりも大幅に高い。SSAM [17]は MBCF と同様にワークステーションクラスタベース (ただしネットワークは 156Mbps ATM) のソフトウェアによる通信機構である。ただし、SSAM はパケットの到着保証と順序保証のプロトコルを省略しているため、実用化時にはこの値よりも悪くなることが予想される。

表中の SP–2 の二つのエントリは MPL/udp が UDP/IP 上に作られたユーザ通信インタフェースを使用した場合の性能、MPL/p が一つのアプリケーションが高速通信ネットワークを占有する通信インタフェースを使用した場合の性能である。保護や汎用性の側面を考慮すると MBCF と比較すべき数値は MPL/udp の方である。

表 7中の MBCF の Round-trip タイムはロジックアナライザによる厳密測定の値を採用している。

SP–2 の性能値は文献 [16]から引用し、表中の MBCF および SP–2 以外の性能値は文献 [17]から引用した。

Table 7　通信性能比較 (MBCF vs MPPs)

Machine	Peak band (Mbytes/s)	Round-trip latency(μs)
SP–1 + MPL/p	8.3	56
Paragon + NX	7.3	44
CM–5 + Active Message	10.0	12
SP–2 + MPL/udp	10.8	554.0
SP–2 + MPL/p	35.5	78.0
SS20 cluster + SSAM	7.5	52
SS20 cluster + MBCF	**11.2**	**49**

我々の MBCF は CM–5 上の Active Message (AM) に Round-trip タイムで劣っているが、MBCF が一切特殊ハードウェアを必要としないこと、AM/CM–5 が仮想化や保護に十分に対応していないことを考慮すれば、我々の MBCF の方式および 100baseTX 上の MBCF の実装が非常に優れていることが判る。現在、MBCF の通信能力 (Peak bandwidth) は 100baseTX の性能によって上限が規定されているため、通信ハードウェアとして他の MPP と同程度に高速なものを使用すれば、さらに大幅な性能アップが期待できる。

10　おわりに

汎用超並列 OS *SSS–CORE* における資源情報を利用したスケジューリング方式、ならびに分散共有メモリを想定したメモリページの分類に従ったメモリページ置換方式について述べた。さらにこれら資源管理機構の性能を、詳細確率モデル上でのシミュレーションにより評価した。その結果スケジューリング方式については、資源管理構造を利用した、プロセスが所有する実ページが存在する領域を割り当てる方式の優位性が認められ、メモリ置換方式については、ページの分類に従ったメモリ置換方式が良い性能を示した。この両方式を組み合わせた場合の実験結果では、負荷状況が割合高いケースでも最大有効実行率による評価が 65% 程度以上で、この値は緩和されたメモリモデルの採用等によりさらに向上することが可能であると考えられる。

保護と仮想化の徹底した高速ユーザレベル通信であるソフトウェアメモリベース通信 (**MBCF**) を 10baseT ならびに 100baseTX を使って、そのサポートオペレーティングシステム SSS–CORE と共に実装した。100baseTX 版 MBCF の基本性能を測定したところ、Peak bandwidth が **11.2**Mbyte/s、Round-trip latency が 49μs であった。これらの値は、MBCF の現実装より大幅に転送能力の高い通信ハードウェアを持ち仮想化や保護のレベルが低い並列計算機のユーザレベル通信能力と比べて、優とも劣らないものである。

さらに、MBCF を既存オペレーティングシステム上の UDP/IP 上に実現して並列アプリケーションによる性能比較を行ったところ、粒度の小さな通信では既存オペレーティングシステムの通信オーバヘッドで SSS–CORE 上の MBCF に大きく及ばないことが明らかになった。MBCF のようなユーザレベル通信を高速に実現するためには、ハードウェアレベルのドライバからユーザインタフェースまで一貫したポリシで開発することが重要である。

今回、我々が使用した SS20 タイプのワークステーションはすでに一世代古いタイプのマシンとなっている。そこで、現在最新鋭のワークステーションへの MBCF ならびに SSS–CORE の移植作業を進めている。これと並行して 100baseTX より高速なファイバチャネルインタフェースや Gigabit Ethernet を使った MBCF の実装も進める予定である。これらの研究開発作業が進めば、レイテンシは現在の 3 分の 1 から 5 分の 1、ピーク転送能力は 5 倍から 10 倍に性能アップが可能となる予定である。

11　NOW 版 SSS–CORE の開発現状

現在、NOW 版 SSS–CORE Ver.1.0 は Sun Microsystems 社の SPARCstation 20 またはこの互換機を EtherNet で接続した環境で動作している (図 11)。SSS–CORE Ver.1.0 の WS 単体担当部分 (SSS-MC: Micro Core Ver.3.0 図 10参照) では、メモリ保護、タイムシェアリング、プロセス管理、UDP 通信、TCP/IP 通信、キー入力、画面出力、外部 UNIX ワークステーションからのファイルの読み書き / プログラムロード / 実行等が実装されている。メモリベース通信機能は遠隔メモリ書き込み、フラグ付きメモリ書き込み、遠隔メモ

Table 8　10baseT MBCF: 転送サイズと台数による並列レイトレ計算時間

サイズ (byte)	64	32	16	8	4	2	1
1 台時間 (sec)	16.90	16.97	17.12	17.39	17.96	19.26	24.63
2 台時間 (sec)	8.67	8.75	8.83	8.94	9.26	12.96	*23.43
3 台時間 (sec)	6.41	6.07	6.07	6.10	7.85	*12.93	*23.16
4 台時間 (sec)	4.55	4.60	4.65	4.83	*7.48	*12.95	*23.54
5 台時間 (sec)	3.73	3.74	3.77	*4.28	*7.34	*12.86	*23.46
6 台時間 (sec)	3.43	3.44	3.32	*4.26	*7.27	*13.04	*23.59
7 台時間 (sec)	2.79	2.80	2.88	*4.16	*7.44	*13.06	*23.65

Table 9　100baseTX MBCF: 転送サイズと台数による並列レイトレ計算時間

サイズ (byte)	64	32	16	8	4	2	1
1 台時間 (sec)	16.87	16.93	17.03	17.23	17.64	18.42	20.02
2 台時間 (sec)	8.65	8.73	8.78	8.82	9.00	9.37	10.24
3 台時間 (sec)	6.40	6.05	6.03	6.02	6.15	6.39	6.88
4 台時間 (sec)	4.55	4.59	4.62	4.67	4.72	4.91	5.29

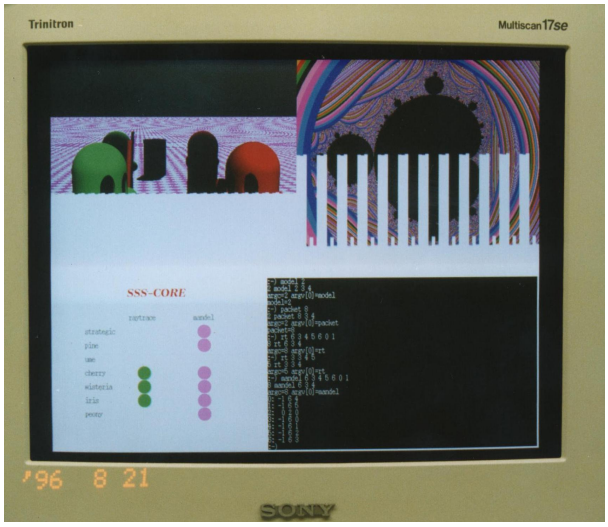

Figure 9　NOW 版 OS 上でのアプリケーション実行 2

リ読み出し、遠隔不可分操作、 Memory-Based FIFO、 Memory-Based Signal、 Elastic Memory Barrier 機能、フロー制御機能、レポート返送機能がすでに実装されており、現在他の階層マルチキャスト機能を実装中である。メモリベース通信機能を利用した並列レイトレーシング、並列マンデルブロー集合表示プログラム、メモリベース通信性能測定ツール等、動画像再生プログラム、が現時点の SSS–CORE Ver.1.0 上で並列実行可能である。図 8および図 9に NOW 版 SSS–CORE 上で、レイトテーシングならびにマンデルブロー集合表示プログラムを実行中の出力画面を掲載する。図 8は、 6 台で動画画像をキューブに張り付けて表示する画面であり、図 9はレイトレーシングを 3 台、マンデルブロープログラムを 6 台で実行中 (ただし、 3 台は二つのアプリケーションを時分割で実行) の画面である。

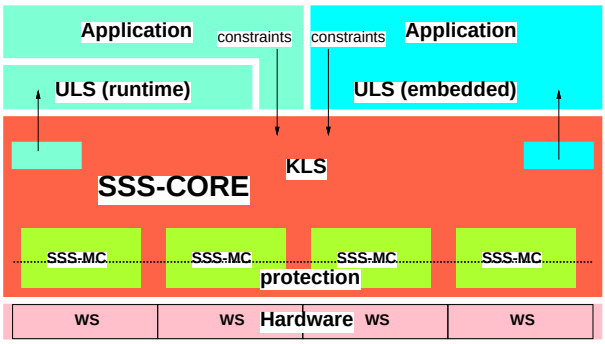

Figure 10　NOW 版 SSS–CORE の構成

References

[1] R. Chandra, et al. "Scheduling and Page Migration for Multiprocessor Compute Servers", Proc. ACM, ASPLOS VI, pp. 12–24, October 1994.

[2] D.G. Feitelson and L. Rudolph, "Distributed Hierarchical Control for Parallel Processing", IEEE Computer, Vol. 23, No. 5, pp. 65–77, May 1990.

[3] D. G. Feitelson, "Packing schemes for gang scheduling", Proc. IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 54–66, April 1996.

[4] Intel Supercomputer Systems Division, "Paragon User's Guide", order number 312489-003 edition, June 1994.

[5] J. K. Ousterhout, "Scheduling techniques for concurrent systems", Proc. 3rd Intl. Conf. Distributed Computer Systems, pp. 22–30, October 1982.

[6] R. Vaswani and J. Zahorjan, "The implication of cache affinity on processor scheduling", Proc. of 13th ACM Symp. on Operating Systems Principles, pp. 26–40, October 1991.

[7] 松本 尚, 平木 敬: 汎用超並列オペレーティングシステム: SSS–CORE — ワークステーションクラスタにおける実現 —. 情報処理学会研究報告 96–OS–73, 情報処理学会, Vol.96, No.79,

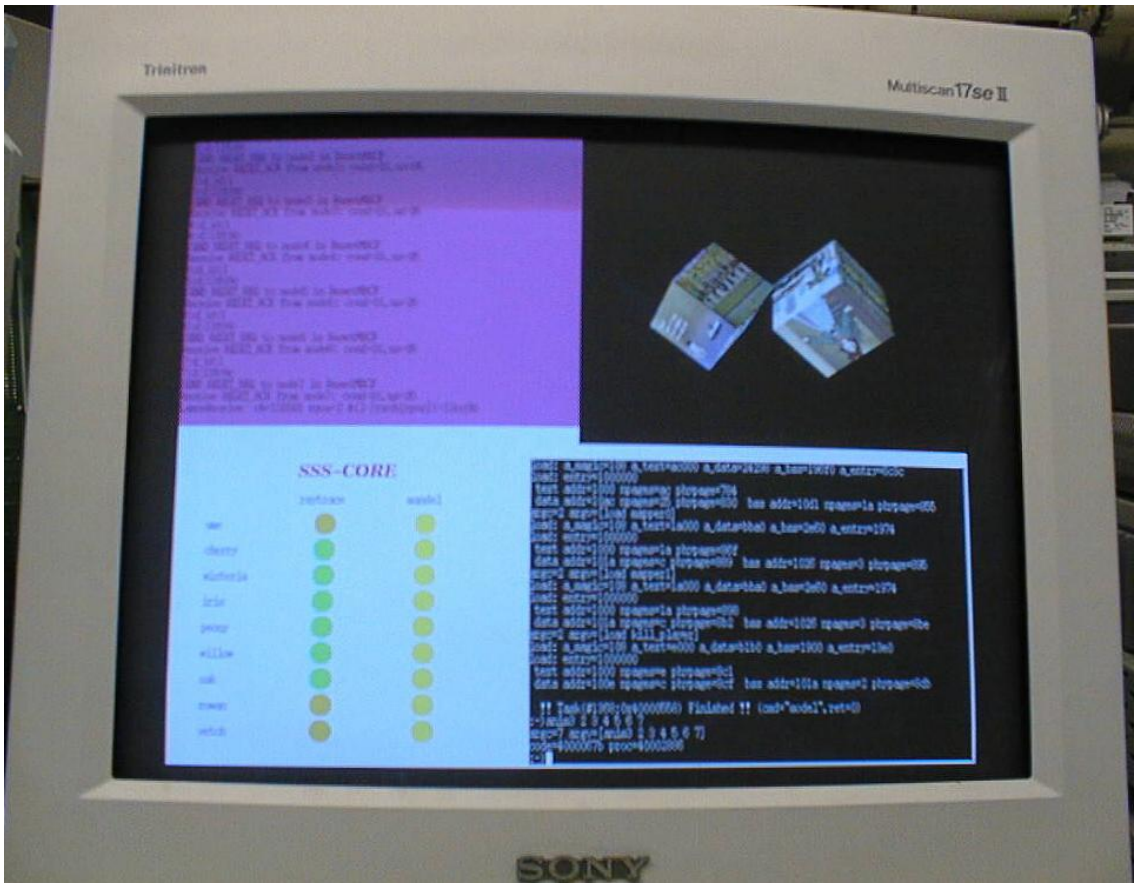

Figure 8 NOW 版 OS 上でのアプリケーション実行 1

pp.115–120 (August 1996).

[8] 松本, 駒嵐, 渦原, 平木: メモリベース通信による非対称分散共有メモリ. コンピュータシステムシンポジウム論文集, 情報処理学会 pp.37–44 (November 1996).

[9] 松本、信国、駒嵐、竹岡、平木: 汎用超並列オペレーティングシステムカーネル SSS–CORE — 基本方式および NOW 版の実装について —. 情報処理振興事業協会第 15 回技術発表会論文集, pp.87–99 (October 1996).

[10] 松本 尚, 平木 敬: Memory-Based Processor による分散共有メモリ. 並列処理シンポジウム JSPP '93 論文集, pp.245–252 (May 1993).

[11] 松本 尚, 平木 敬: 汎用並列オペレーティングシステム SSS–CORE の資源管理方式. 日本ソフトウェア科学会第 11 回大会論文集, pp.13–16 (October 1994).

[12] 信国陽二郎, 松本尚, 平木敬: 並列ＯＳの性能予測を可能にするシミュレーションモデル. 情処研報 96-ARC-117, Vol. 96, No. 23, pp. 19–24, March 1996.

[13] 信国陽二郎, 松本尚, 平木敬: 汎用並列ＯＳのための資源情報を利用したスケジューリング方式の検討, 信学技報, Vol. 95, No. 210, pp. 111–118, August 1995.

[14] 信国陽二郎, 松本尚, 平木敬: 汎用並列 os *sss–core* におけるカーネルスケジューリング方式 — 詳細確率モデルによる性能評価 —, 情処研報 96-OS-73, Vol. 96, No. 79, August 1996.

[15] 松本 尚, 平木 敬: 汎用並列オペレーティングシステムにおける資源保護と仮想化. 情報処理学会研究報告 97–OS–75, 情報処理学会, Vol.97, No.56, pp.37–42 (June 1997).

[16] M. Snir et al.: The Communication software and parallel environment of IBM SP2 *IBM Systems Journal*, Vol. 34, No. 2, (1995).

[17] T. von Eicken, A. Basu, and V. Buch: Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, pp.46–53 (February 1995).

Figure 11　NOW 版 SSS–CORE の動作環境