汎用超並列 OS *SSS–CORE* における資源管理機構
— スケジューリング方式とメモリ置換方式の性能評価 —

信国　陽二郎[†]　松本　尚[†]　平木　敬[†]

NUMA 環境で並列プロセスの効率的な実行を妨げることなくマルチユーザ / マルチジョブ環境を構築するには、メモリページなどの実資源の使用状況を考慮したスケジューリングを行ない、システム全体の性能をあげることが有効である。また複数のプロセスが並行に動作する汎用的環境では、実メモリが溢れる場合を想定したシステム構築が求められる。NUMA 環境上で分散共有メモリを構築する場合、メモリページを参照頻度および再アクセス時の参照コストにより区別すれば、効率的な置換方式が可能となる。本稿ではメモリアクセスベースの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り、並列プロセス毎に所有する実ページ情報を利用したスケジューリング方式、およびページの分類に基づいたメモリ置換方式の評価を行う。

Resource Management Mechanisms for General Purpose Massively Parallel OS *SSS–CORE*
— Performance of Scheduling and Memory Replacement —

Yojiro Nobukuni,[†] Takashi Matsumoto[†] and Kei Hiraki[†]

Preventing parallel processes from unexpected inefficiencies is a major concern for constructing multiple user/multiple job environment in NUMA systems. Systems can achieve higher performance by using scheduling policies which reflects resource consumption states. For a general environment, which must support concurrent execution of multiple processes, a way is needed to keep systems' effectiveness when physical memories are full. In NUMA systems, memory pages can be classified by access frequencies and required costs for accesses when target pages are not found locally. Selecting victim pages according to the classification enhances system performance. We built a probabilistic model with a concrete memory management scheme and differentiated memory access costs, and simulated processes sets with given access frequencies. The paper describes an evaluation of scheduling policies using resource informations for each process and of page replacement policies based on page coloring under the model.

1. はじめに

NUMA 型システムは、安価な構成要素を多数結び付けて大規模なシステムを構築し、高い処理能力を得ることができる特徴を持つ。近年のネットワーク周辺の性能向上を背景に、並列アプリケーションのための実行支援機構 [13]、並列環境下での各種最適化技法の研究 / 開発 [7] [6] により NUMA 型並列計算機上での高効率で汎用性を持った実行形態が可能となってきた。しかしながら、汎用なマルチジョブ / マルチプロセス使用形態において高効率な実行を実現するためには、個々のプロセスにおける効率化 / 最適化技法だけでなく、オペレーティングシステムレベルによるジョブ / プロセスを跨った高効率化が不可欠である。

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 [11] 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、OS レベルによるコストの高いアクセスを減少させる高効率化が可能である。

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 [14] があるが、容量の小さな

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, University of Tokyo

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 [4] と Intel Paragon [2] 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。

さらなる OS による高効率化として、参照頻度および再アクセスに要するコストを考慮した置換を行なうことにより、性能向上を図ることが可能である。分散共有メモリでは、二次記憶よりコスト安な遠隔アクセスが可能な、共有コピーページを優先的に置換する戦略をとることが考えられる。

本稿では汎用超並列オペレーティング・システム *SSS–CORE* [5] における資源管理機構について述べ、プロセスの実ページ情報を利用したスケジューリング方式ならびに、メモリページの分類に則ったメモリ置換方式のシミュレーションによる評価を示す。まず 2 節で *SSS–CORE* について簡単に触れ、3 節でその資源管理機構を解説する。シミュレーション・モデルならびに実験結果についてはそれぞれ 4、5 節で述べ、6 節でまとめる。

2. SSS–CORE

SSS–CORE [5] 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、並列アプリケーションの効率良い実行の実現を目的とする。

NUMA 型の並列計算機システムでは、メモリ参照コストおよび通信コストがその距離に影響され、またそれらの同期への効果が大きく実行効率に影響する。そこで、並列アプリケーションの実行効率を向上するには、各スレッドを同時にスケジューリングし、割り当てられた要素プロセッサ間の距離を最小化し、実行時のメモリページの移動を最小限に抑えることが必要である。またこのような性質の並列プロセスを、複数同時に動かす汎用な環境を構築するためには、オペレーティングシステムレベルによるジョブ / プロセスを跨った高効率化が欠かせない。それを支援するため *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ている。*SSS–CORE* ではこの機構を利用し、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、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。次節においてスケジューリングおよびメモリ置換の具体的な方法について説明する。

3. *SSS–CORE* の資源管理機構

3.1 *SSS–CORE* のスケジューリング方式

3.1.1 資源管理木

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 (3.1.2節) に従って最も適当な資源を、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。2 レベルスケジューリングであるため、各プロセスは割り当てられた資源を自由に再スケジュールすることで、最適化を図ることが可能である。

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。資源管理木上ではシステム全体の資源情報および、プロセス毎の資源の使用状況等を管理する。大規模な NUMA 型システムにおいては、管理用資源量の制約や管理上のボトルネックを考慮すると、階層的な資源管理は必須である。資源管理木は自身の構造を実システムの構成に適応させることで、様々なシステムをサポートすることが可能である。その結果資源管理木は、例えば階層的に接続された WS-Cluster においては要素処理装置間の距離を自然に表現することができ、メッシュ結合の超並列計算機においては要素処理装置間の距離的な近傍を表現することができる。すなわち、割り当てを行なう要素処理装置間の通信コストを把握したうえでのスケジューリングをも可能としているといえる。

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。図 1に 4 プロセッサ構成システムの例を示す。

3.1.2 スケジューリング制約

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。また *SSS–CORE* では、動的な資源割り当ての変化にプロセスが対応できるように、資源割り当て状況等をカーネルからユーザに低コストで開示する。

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ことができる。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

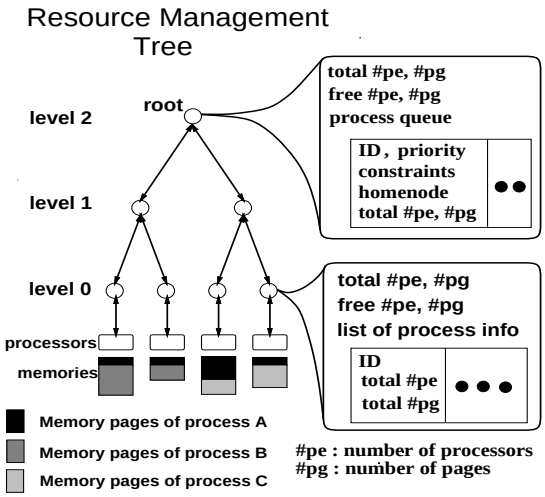

図1　資源管理木

ことが可能となる。例えば十分な並列性があるために可変なプロセッサ台数に対応してプログラムされたプロセスに対しては、この制約が有効に働く。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。

3.1.3　優先度の計算法

各プロセスの欲求に沿って資源割当を行うため、プロセス間の公平性を保つ機構が必要である。$SSS\text{–}CORE$ では占有資源量とスケジューリング制約の強度を基にスケジューリング優先度を計算する。従ってこの二観点について公平なスケジューリングが達成される。

$SSS\text{–}CORE$ では優先度の計算 [8] [5] にあたり、次の値を計算する。1) 使用した資源量: U_r、2) 制約条件の厳しさ: R_c、3) 制約条件の満足度: $S_c = 0\ or\ 1$、4) 無駄にした資源量: W_r、5) 待ちプロセスの有無: $f_w = 0\ or\ 1$、6) 若返り系数: C_r。優先度は値が小さいほど高い。プロセス毎の優先度のエイジング値は、次の式により求められる。

$$aging = (U_r R_c S_c + W_r) f_w * t - C_r(1 - t)$$

また優先度の値の発散を防ぐため、直前までスケジュールされていたプロセスのエイジングの総和を、待ち状態だったプロセスに若返り分として均等に分配する。

3.2　ページ置換方式

複数のプロセスが並行に動作する汎用な環境では、実メモリが溢れる場合を想定したシステム構築が求められる。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。

分散共有メモリでは、参照頻度については共有領域であるか否か、走行中のプロセスの空間であるか否かで分類できる。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自然に次のように与えることができる。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。

置換方式としては、他のプロセスのページを置換対象とする場合 (上記 1,3,4) に、選択するプロセスの順序により、複数の戦略が可能である。$SSS\text{–}CORE$ ではプロセスの優先度順を管理しているため、優先度の低いプロセスのページから置換対象とすることが可能である。

4. シミュレーション・モデル

4.1　モデルの概要

並列アーキテクチャにおいては、OS 機能の評価のための命令レベル等の非常に精密なモデルによるシミュレーション [12] は、非評価プログラムの任意性や一般性から事実上不可能であり、メモリアクセスモデルに基づく方法が最も適用可能な方法である [10]。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 [9] 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。

4.2 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優先度計算 [8] 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 2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。 5 つの方式のうち algo2, 3, 4 が資源

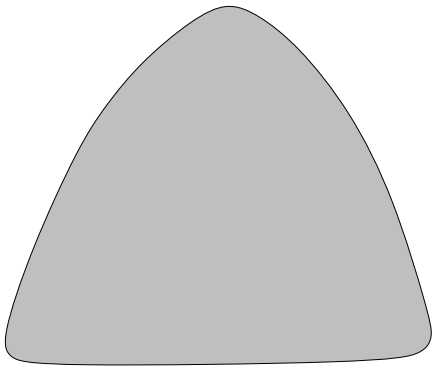

図2 資源管理木とスケジューリング対象領域

管理木構造を利用する方式である。この 3 方式の違いはプロセスへのプロセッサの割り当て領域が、使用中の実メモリページの位置に固執する度合にある。図 2の例では領域 2, 3, 4 がそれぞれ algo2, 3, 4 が割当を試みる可能性のある領域に対応している。ここで領域 4 が前回の割り当て位置で、プロセス A が実ページを所有している領域である。また領域 3 が領域 4 に対応するホームノードによって代表される領域である。 algo4 の場合には基本的に、プロセス毎の実ページの存在位置に対応して毎回同じ領域が割り当てられる[☆]。 algo3 の場合には、領域 3 にあたるホームノード以下の領域において割当を試みる。前回のスケジュール時の割り当て領域が、他のプロセスに既に割り当て済みな場合には、必要な台数が確保できれば実ページを持たない領域であっても、領域 3 の中であれば割り当てる。 algo2 の場合には、algo3 の要領で領域 3 で必要台数を確保できない場合に、システム全体に相当する領域 2 内の残りの領域で割当を試みる。したがって上記 3 方式の間では algo4, 3, 2 の順に実メモリのある領域を割り当てる傾向にあり、その逆順でプロセッサの利用台数が多くなる可能性が高い。この三方式では、対象領域に必要台数を確保できないプロセスについては、その時点ではスケジュールされない。 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
- 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

[☆] プロセスの実ページが再スケジュールまでに一部クラスタにおいて完全にクラスタメモリから追い出された場合には、ホームノード以下の領域 (領域 3) で足りない分のプロセッサを確保する。

表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

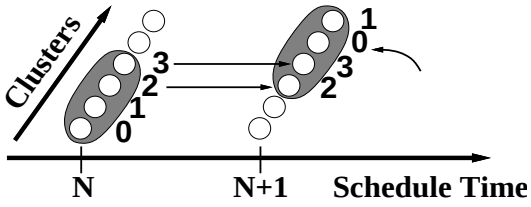

図3　ユーザレベル・スケジューリング例

図 3にスレッド・スケジューリングの例を示す。図 3の例では、割り当て領域が N+1 回目と N 回目で重なっている領域のスレッド 2, 3 はそのままスケジュールされ、割り当て領域が変化したスレッド 0, 1 についてはスケジュールされるクラスタが移動している。

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。

なお、実行効率の向上を目的として時分割の間隔を長め (数百ミリ秒 ~ 数秒程度) に設定すれば、スケジューリングに要する計算時間は相対的に十分小さい☆。ここでは十分に長い時分割間隔を仮定し、シミュレーションではスケジューリングの計算時間を無視する。

4.3　ページ置換方式

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。

方式 **0**　ページの種類 / プロセスの区別なしに単純な LRU 順

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

方式 1 は走行中以外のプロセスについて、優先度を利用した順序づけを行う。優先度の低いプロセスのページほど参照されにくい性質を利用している。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。

5.　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

☆ *SSS–CORE* では時分割単位を大きめにとる

表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	10000 clk
ページ移動基本コスト	50 clk
通信基本コスト	5 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	100000 clk

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。ここでの実験の目的は、プロセスの生成・消滅によるシステムの負荷不均衡を考慮した長期間におけるスケジューリング方式の性能ではなく、 OS にとって短期間での一定負荷状態における性能評価をすることである。そこで実験は、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。またこの規模の並列システムにおける OS レベルのシミュレーションを、実時間サイズで行うのは不可能である。そこで実験は時間方向に圧縮して行い、各種コスト差をオーダーの仮定に沿った値に設定した。

実験が終了した時点での評価結果を図 4, 5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。各グラフにおいて 6 本ずつの各グループが、それぞれスケジューリング方式 0 から 4 を採用した場合の結果である。各グループにおいて、左 3 本が置換方式 0 の場合、右 3 本が置換方式 1 の場合に対応する。これら 3 本のグラフは、左が “正味の有効実行率”、右がそれにアイドル時間を加算した “最大有効実行率”、中央がアイドル時間を正味の有効実行率分だけ加算した “補正有効実行率” を表す。

5.1　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。どのプロセス組においても、正味の有効実行率による比較で algo4 が最良の結果を出している。 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、この活用されていない分を含めた補正有効実行率や最大有効実行率で比較すると、 algo4 と他の方式の性能差はさらに広がる。実際のシステムではアイドル時

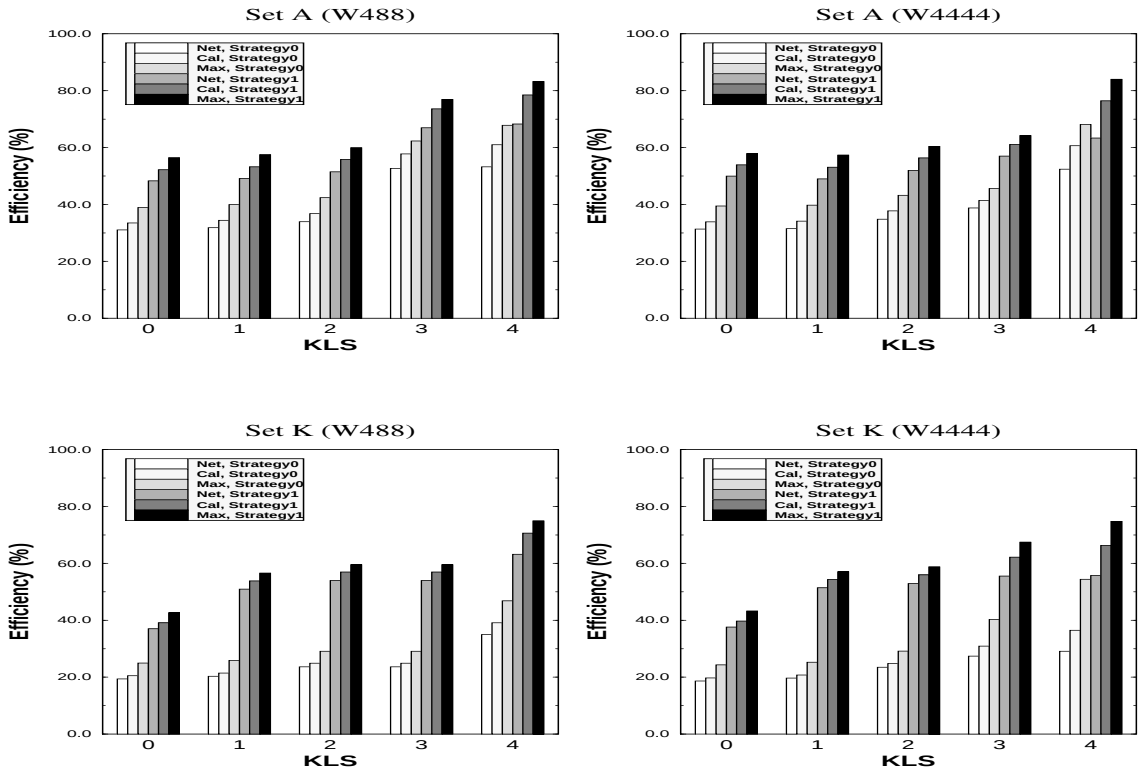

図4　セット A, K の結果 (左列:w488, 右列:w4444)

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 ($SSS\text{–}CORE$ 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。

1 では補正有効実行率程度の性能を見込め、2 では最良のケースで最大有効実行率までの性能を見込むことができる。本稿に挙げたごく一部の特定のプロセス組についてだけでなく、より多様なケースを考慮した評価を行うことは、汎用 OS としての性能ならびに $SSS\text{–}CORE$ 本来の性能を論じる上で必要不可欠である。この考え方から補正有効実行率や最大有効実行率での比較が有効であり、 algo4 が常に最良の結果を出しているといえる。

4.2節で説明したように、 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った。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、表 3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。さらにプロセスの割り当て領域を頻繁に変更すると、各クラスタに各プロセスのメモリが集中してしまい、互いに実ページの奪い合いを起こす量が増加するため、 algo4 に比べて algo3、 algo2 が効率が悪くなっている。また、メモリ溢れがほとんど起きない☆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 (表 3)。さ

表3　実行時間内訳 (%)(w488, 置換方式 3)

Set	Type	algo0	algo1	algo2	algo3	algo4
A	最大有効	57.89	57.36	60.41	64.22	84.02
	同期	19.98	19.59	20.84	20.03	12.45
	通信	22.10	23.01	18.72	15.70	3.49
K	最大有効	42.65	56.57	59.53	59.53	74.91
	同期	25.87	22.08	23.24	23.24	17.82
	通信	31.46	21.32	17.20	17.20	7.22
L	最大有効	38.81	48.93	54.98	54.98	65.85
	同期	28.52	24.07	23.58	23.58	22.23
	通信	32.64	26.96	21.39	21.39	11.87
M	最大有効	24.79	35.73	47.42	61.47	64.37
	同期	22.83	19.03	18.68	16.57	16.06
	通信	52.36	45.21	33.86	21.90	19.51

ただしディスクアクセス時間を除く

☆ 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。

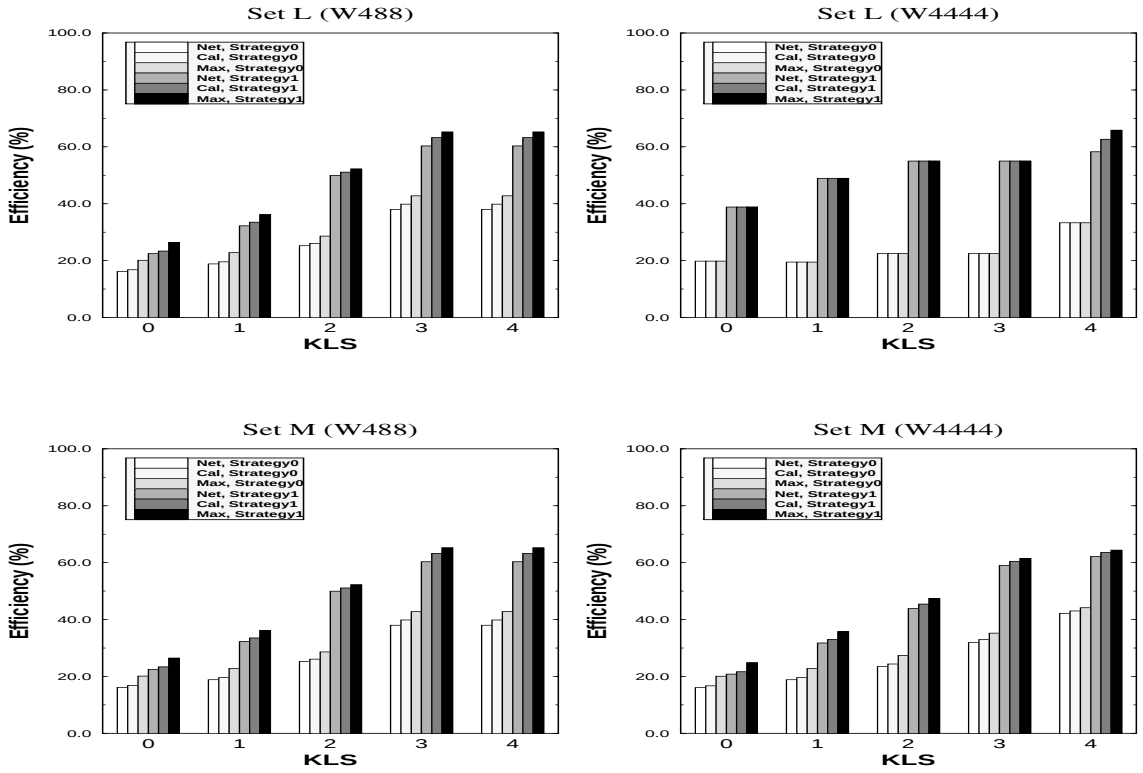

図5　セット L, M の結果 (左列:w488, 右列:w4444)

て、置換方式 1 で正味の有効実行率の結果 (グラフ中各グラフ内右から 3 つ目) を見ると、 algo2=algo3 となっているケース (K, W488; L, w44444) が 2 つ、 algo3=algo4 となっているケース (M, L; w4444) が 2 つある。 algo2, 3, 4 では、システムの階層性を反映させた資源管理構造を利用してスケジューリングを行うため、プロセスへの割り当て領域を管理木の部分木の範囲で限定することができる。そこでスケジューリング方式が異なっても、プロセスの組み合わせによっては等しいスケジューリングを行っている。 algo4 ではこの領域を限定する特質を、メモリの利用状況の情報と併せることであらゆるプロセスに対して利用することが可能となっており、資源管理構造を導入した利点が発揮されている。

5.2　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。性能の良い algo4 の場合において、 w488 の実験での各プロセス組での置換の様子を表 4に示す。

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。

5.3　汎用並列環境の可能性について

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。しかし今回の実験では、プロセスの共有領域へのアクセス率を小さく設定してあり (1–3%)、多様なアプリケーションを考えた場合には、スケジューリング方式間の性能差の現れ方の傾向に変化はないと推定する。さらに、緩和されたメモリモデルの上で共有領域へのアクセス率が高いアプリ

表4　ページ種類毎の置換回数 (algo4, w488)

置換方式	プロセス組	ページ種類 other's copy	own copy	other's last	other's local	own last	own local
方式1	A	10944	4770	723	4763	303	1130
	K	33105	10487	830	22598	65	3803
	L	92253	18063	4911	63885	136	9978
	M	46394	23581	6873	11725	4054	6577
方式3	A	38976	0	0	0	0	0
	K	77665	0	0	0	0	0
	L	414287	0	0	0	0	0
	M	245705	11578	0	0	0	0

ケーションについては、各種コンパイル技術と階層化マルチキャストやAckのコンバイニングといった通信技術を組合わせて導入することにより、実行効率の向上をはかることが可能である。

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。

6. 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。

本来は相容れない並列アプリケーションの効率的実行と効果的な汎用性の実現を考えたとき、実資源が分散配置される大規模NUMA型システムにおいては、アプリケーションによる安価な資源利用コストの保証と、資源管理におけるボトルネックの排除が重要となる。さらに、ある程度の高負荷状態におけるシステム性能の保証も必須である。本稿で述べた *SSS–CORE* で採用するスケジューリング方式ならびにメモリ置換方式は、上記の目的を実現する資源管理機構の重要な要素となるものである。本研究における評価を通して、NUMA型システム上に高効率な汎用環境を提供する実際的なオペレーティングシステムの資源管理機構の可能性を示すことができた。

謝辞

本研究は情報処理振興事業会（ＩＰＡ）が実施している独創的情報技術育成事業の一環として行なった．

参考文献

1) R. Chandra, et al. Scheduling and Page Migration for Multiprocessor Compute Servers. In *ACM, ASPLOS VI*, pp. 12–24, October 1994.
2) Intel SupercomputerSystems Division. *Paragon User's Guide*, order number 312489-003 edition, June 1994.
3) D. G. Feitelson. Packing schemes for gang scheduling. In *Proc. IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 54–66, April 1996.
4) D.G. Feitelson and L. Rudolph. Distributed Hierarchical Control for Parallel Processing. *IEEE Computer*, Vol. 23, No. 5, pp. 65–77, May 1990.
5) 松本尚ほか. 汎用超並列オペレーティングシステム SSS–CORE. 日本ソフトウェア科学会第11回大会論文集, pp. 13–16, October 1994.
6) 松本尚. マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察. 計算機アーキテクチャ研究会報告 No.79-1, pp. 1–8, November 1989.
7) 松本尚. 細粒度並列実行支援マルチプロセッサの検討. 情報処理学会論文誌, Vol. 31, No. 12, pp. 37–42, August 1989.
8) 信国陽二郎ほか. 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.
9) 信国陽二郎ほか. 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.
10) 信国陽二郎ほか. 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.
11) J. K. Ousterhout. Scheduling techniques for concurrent systems. *3rd Intl. Conf. Distributed Computer Systems*, pp. 22–30, October 1982.
12) M.Rosenblum, etal. Complete Computer Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology*, 1995.
13) T.E.Anderson, et al. Scheduler activations: Effective kernel support for the user-level management of parallelism. *Proc. of the 13th ACM Sympo. on Operating Systems Principles*, Vol. 25, No. 5, pp. 95–109, October 1991.
14) R. Vaswani and J. Zahorjan. The implication of cache affinity on processor scheduling. In *Proc. of 13th ACM Symp. on Operating Systems Principles*, pp. 26–40, October 1991.