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 —

信国　陽二郎[†]　松本　尚[†]　平木　敬[†]

分散メモリ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。分散環境では、参照頻度及び再アクセスのコストにより実メモリページを区別すれば、効率的な置換方式が可能である。本稿ではメモリアクセスベースの確率モデル上で、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、及びメモリ置換方式の評価を行う。

Scheduling in General Purpose OS *SSS–CORE*
— An Evaluation by Detailed Probabilistic Simulation —

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

Preventing parallel processes from unexpected ineficiencies is a major concern for constructing multiple user/multiple job environment in distributed memory systems. Systems can achieve high performance by using shcheduling policies which reflects resource comsumption states. For a general environment, which must support concurrent execution of multiple processes, a way is needed to keep systems' effectiveness when phisical memories are full. In distributed systems, memory pages can be classified by access frequencies and required costs for accesses after target pages has been replaced. Selecting victim pages according to the classification may enhance system performance. We built a probabilistic model with a concrete memory management scheme and differntiated memory access costs, and incorporated memory reference frequencies to it. The paper describes an evaluation of scheduling policies using resource informations for each process and of page replacement policies 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 [1]、並列環境下での各種最適化技法の研究 / 開発 [4] [3] は、 NUMA 型並列計算機上での高効率な汎用性を持った使用形態を可能としつつある。我々は、NUMA 型並列計算機上に汎用な環境を提供する能力を備えたオペレーティング・システム *SSS–CORE* を開発中である。

分散メモリ環境では、メモリへのアクセスコストが距離によって異る。複数のプロセスが並行に動作する汎用的環境では、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、コストの高いアクセスを減らすことでシステム全体の性能を上げることが可能である。また参照頻度及び再アクセスに要するコストが小さなメモリページから置換を行なうことにより、全体の性能向上を計ることができる。

我々はこれまでの研究で利用したモデル [5] を拡張し、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メモリアクセスベースの確率モデルを構築した。本稿ではこのモデル上でのシミュレーションによる、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法ならびに、メモリページの分類に則ったメモリ置換方式の評価について述べる。

2. SSS–CORE

SSS–CORE [2] は NUMA 型並列分散計算機を対象とした汎用並列 OS である。時分割 とパーティショニ

† 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

ングを併用してマルチユーザ / 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を実現することが目的である。

NUMA 型の並列計算機システムでは、メモリアクセスコスト及び通信コストが距離に影響される。並列アプリケーションを効率良く実行するには、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、及び実行時のメモリページの移動を最小限に抑えることが重要である。

SSS–CORE では実メモリの使用状況のスケジューリングへの反映、ユーザレベル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 3 つのアプローチをとる。

SSS–CORE では、計算機の相互結合網の階層構造と一致した構造の資源管理構造を用意し、この構造上でシステム全体の資源情報及び、プロセス毎の資源の使用状況等を管理する。 2 レベルスケジューリングを採用し、カーネルはこの資源情報と各プロセスから提示されるスケジューリング制約にしたがって、プロセス単位に資源の割当を行なう。資源情報を利用することで、タイムスライス毎の資源の割り当て状況をスケジューリングに反映することが可能。更に、提示されたスケジューリング制約に従ってスケジューリングすることで、プロセス毎に異なる資源への欲求を満たすことができる。資源管理構造は並列計算機の構造とを反映しているため、割り当てを行なうプロセッサ間の通信コストを把握したうえでのスケジューリングが可能である。 2 レベルスケジューリングであるため、プロセス内での資源割り当てはユーザに一任される。

スケジューリングされたプロセスは、使用した資源量とスケジューリング制約に従い優先度がエイジングされ、タイムスライス毎に優先度順にスケジューリングされる。これにより公平なマルチユーザ / マルチタスク環境を実現する。

3. *SSS–CORE* のスケジューリング方式

3.1 資源管理木

SSS–CORE では、システムの階層構造と同一の階層性を持った資源管理用の木構造を仮想的に持つ。本稿ではこの構造のこと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 及び使用しているプロセッサと実ページの総数を記憶する。またルートノードにはこの他に、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。資源管理木を利用することにより、資源の消費状況を反映したスケジューリングが可能となる。

3.2 スケジューリング制約

並列プロセスへの資源割当がプロセスの要求と異なるために、プロセス実行の静的な予測に基づいた最適化が機能せず、効率的な実行ができなくなるケースが起こる。カーネルによるスケジューリングが、プロセス毎に異なる資源への要求を満たすために、*SSS–CORE* ではユーザからカーネルに対してスケジューリング制約を提示できるようにする。

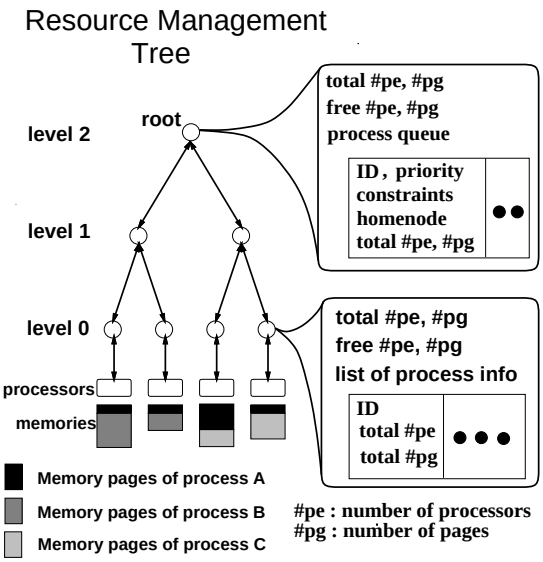

図1　資源管理木

例えば台数固定制約では、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。また可変台数制約があれば、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。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扱う。

3.3 優先度の計算法

優先度は値が小さいほど高い。*SSS-CORE* では優先度の計算 [2] [5] にあたり、まず以下の値を計算する。

- 使用した資源量： U_r
- 制約条件の厳しさ： R_c
- 制約条件の満足度： $S_c = 0\ or\ 1$
- 無駄にした資源量： W_r
- 待ちプロセスの有無： $f_w = 0\ or\ 1$
- 若返り系数： C_r

これらの値を (一部はプロセス毎に) 計算すると、次の式によりプロセス毎のエイジングの値が求められる。

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

また優先度の値の発散を防ぐために、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公平に均等に分

配する。

4. シミュレーション・モデル

4.1 主な特徴

これまでの研究で利用したモデル [5] を拡張し、OS レベルのシミュレーションのため簡略化されたモデル [6] 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である。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。

各プロセスには共有空間と非共有空間を、それぞれ別々のページ単位のメモリ参照頻度表により与える。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。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び、各スレッドは異なった空間を与えられる。各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。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。ページがリモートにも存在しない場合には、ディスクアクセスが生じる。共有空間に対しては分散共有メモリシステムを構築し、プロセスの各スレッドは参照頻度表を共有する。コヒーレンシ管理は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。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。

プロセスの実行はクロックベースの確率モデルで、各クロック毎にリード又はライトのメモリ参照を行なう。ただし、対象とするページが自クラスタのメモリに存在しない場合は、ページ待ち状態となる。またプロセスのスレッドは、パラメータで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。ここで有効実行 (時間 / クロック) とは、ページ待ち / 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。ページ待ちまたは同期待ち状態の場合には、スレッドは何も実行できない。シミュレーションでは、システム全体において有効実行の占める割合 (有効時効率) を計測する。

4.2 スケジューリング方式

シミュレータ上に 5 つのスケジューリング法を実装した。各方式共に、タイムスライス毎に資源の占有 / 消費状況に応じた優先度計算 [5] 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でページのある領域から割り当てる方式

algo4 *SSS–CORE* で使用する、ページのあるクラスタを割り当てる方式

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、最下位レベルに当た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、プロセッサを 3 台要求するプロセス A に対しては、3 台以上のプロセッサ (を含むクラスタ) を包含する部分木のルートに当たるノードがホームノードである。

5 つの方式のうち algo2, algo3, algo4 が資源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 3では、領域 2, 3, 4 がそれぞれ algo2, algo3, algo4 が割当を試みる可能性のある領域に対応している。この場合領域 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

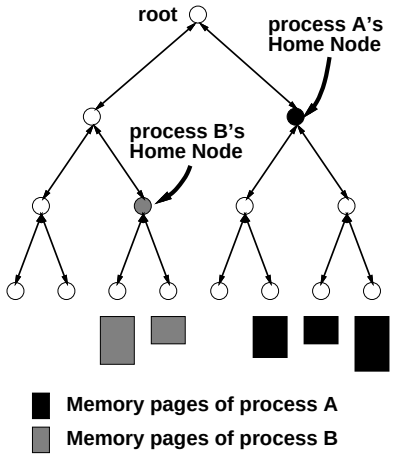

図2 資源管理木におけるホームノードの例

☆ 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。

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も割り当てる。 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といえる。

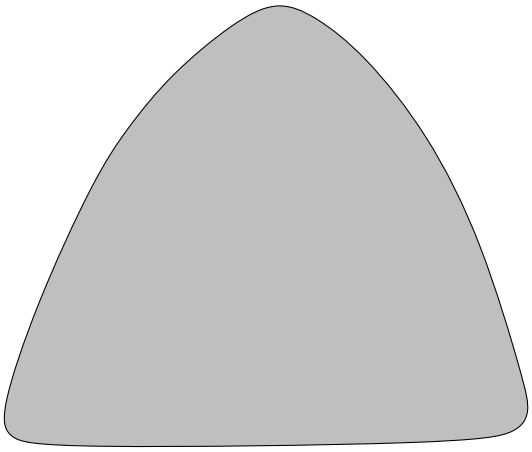

図3　資源管理木と各スケジューリング方式の割り当て対象領域

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。

図 4に、スレッドス・スケジューリングの例を示す。図 4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部分のスレッド 3、 4、 5 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、 2 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、同一のスレッドが同一のクラスタで実行されれば、ローカル空間は再利用される。しかし、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。

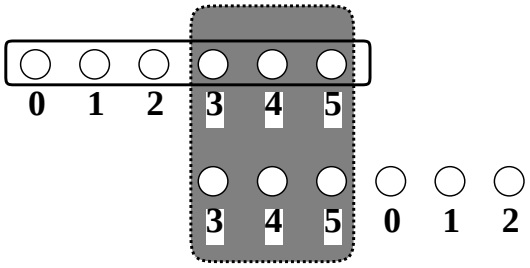

図4　ユーザレベル・スケジューリング例

なお、 $SSS\text{–}CORE$ では時分割の間隔を長め (数百ミリ秒 ~ 数秒程度) に設定するため、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。従って、シミュレーションではスケジューリングの計算時間は無視する。

4.3　ページ置換方式

分散メモリ環境上では分散共有メモリを仮定すると、参照頻度及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方式が可能となる。

例えば、明らかに実行中のプロセスが利用するページよりは、それ以外のプロセスが所有するページの方がアクセスされにくい。また一つのプロセスのページの中でも、クラスタにローカルなページよりは共有ページへのアクセスの方が一般的に少ない。また共有ページはそれが複数存在するコピーのうちの一つであれば、再アクセス時には遠隔地からネットワークを経由して、低コストでアクセスすることが可能である。それに対して、再アクセス時にディスクアクセスを生じるようなページ置換は極力抑えた方が得である。そこでページの種類により、次のように置換対象とする順序を与えることができる。以下ではコピーを持たない共有ページを、便宜的にラストワン・ページと呼ぶ。

(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

置換方式としては、置換対象とするプロセス順をどのように選択するかにより、複数の戦略が可能である。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。

表1　プロセス・セット (セット J2)

#DP	#PS	非共有空間サイズ * プロセス数
4	4	$30pages * 2, 80pages * 2$
8	4	$30pages * 4, 80pages * 1$
16	2	$30pages * 2$
総並列度 80		総非共有空間 3200pages

#DP: 要求プロセッサ数, #PS: プロセス数

表2　プロセス・セット (セット K2)

#DP	#PS	非共有空間サイズ * プロセス数
3	5	$50pages * 5, 80pages * 2$
6	5	$50pages * 5, 80pages * 1$
13	1	$30pages * 1$
16	1	$30pages * 1$
総並列度 74		総非共有空間 3120pages

#DP: 要求プロセッサ数, #PS: プロセス数

方式 **1**　ページの種類 / プロセスの区別なしに、単純な LRU 順

方式 **3**　スケジューリング優先度の低いプロセスから、上記の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 LRU 順を管理する

プロセスのスケジューリングは優先度の高い順に行なわれるため、優先度の低いプロセスのページほど参照されにくい。方式 3 はこの性質を利用する。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。

表3　各種コスト及びパラメータ

項目	値
ディスクアクセス	10000 clk
リモートアクセス	50 clk
通信	10 clk
プロセッサ数	16 台
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	25.6 Mbyte
1 quantum	100000 clk

5. 結果及び考察

二組のプロセス組に対してシミュレーションを行なった。表 1、 2が各組の内訳である。両セットの各プロセスはどれも 80 ページの共有空間を持ち、コピーページが生成されることによりメモリ溢れが起きる。また各プロセスは、 1000 有効実行クロック毎にランダムな組合せで同期する。各種パラメータは表 3のように設定した。ページの移動に対しては、ネットワークの各エッジ上でリモートアクセスパラメータ分のコストを付加する。

図 5から図 8で各スケジューリング方式 (横軸) に対して、システム全体の有効実行率 (縦軸) を示した。なおデータの取得はシミュレーション開始後、一つ目のプロセスが終了 (20 タイムスライス分の有効実行) した直後の時分割の時点で行なった。各メモリ置換方式を採用した場合の結果を示してある。 16 台構成の小規模システムでの固定台数制約を利用した状況では、プロセッサの利用効率が大きく評価に影響する。この環境で実際のシステムにおいて十分なプロセッサの利用効率がある場合の評価を行なうため、割当から洩れたプロセッサの実行は有効実行にカウントした。

図 5はセット J2 の 2 進 4 段の 16 台システム構成上での結果である。スケジューリング方式 algo4 が最良の結果を示している。 algo4 と algo3 が同じ性能なのは、 2 の巾の並列度のプロセスの集合であるセット J2 とネットワークの形状の相性のため、同じスケジューリングを行なったからである。図 7で同じくセット J2 の 4 進 2 段システムでの結果を見ると、 algo4 と algo3 にも明らかな差が見られる。同様にセット K2 に対して図 6に 2 進 4 段での結果、図 8に 4 進 2 段での結果を示す。やはり algo4 が最良の結果を示している。

従ってどの結果でも、スケジューリング方式 algo4 が最良の結果を示している。また algo4、 3、 2 を比較すると、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、実行効率が高いことがわかる。セット J2 の 4 進 2 段の場合での実行時間の内訳 (表 4) を例に見ると、通信 (主にページ要求やページ移動) や同期に要した時間に差があることが分かる。

これは、参照頻度の高いローカル空間の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。

表4　同期 / 通信時間☆(セット J2, 4 進 2 段, 置換方式 3)

type	algo0	algo1	algo2	algo3	algo4
有効実行	76.79	80.92	85.27	89.66	92.45
同期	9.70	7.56	6.71	5.42	4.17
通信	13.47	11.48	7.96	4.87	3.32
ディスク	0.04	0.05	0.06	0.05	0.06

またメモリの置換方式についてみると、どの場合にも置換方式 3 が方式 1 に勝っている。最も効率の良かったセット J2 の 4 進 2 段の場合で見ると、置換の様子は表 5のようになっている。方式 3 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 1 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る。したがって、方式 1 はまず置換の回数が方式 3 より多く、場合によっては再アクセス時にディスクアクセスを発生している。

表5　ページ種類毎の置換回数 (セット J2, 4 進 2 段)

ページ種類	0	1	2	3	4	5
方式 1	2815	661	60	1707	46	281
方式 3	7820	0	0	0	0	0

ページ種類は 4.3 節での分類上の番号

☆ 情処提出バージョンを訂正した。

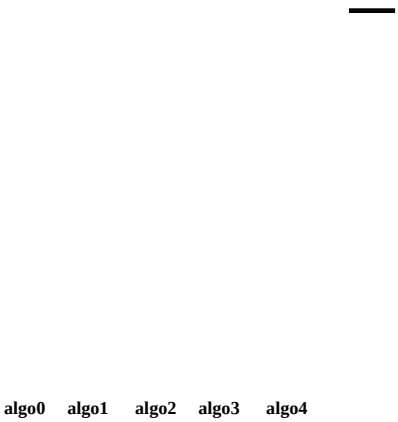

図5　シミュレーション結果 (セット J2、 2 進 4 段)

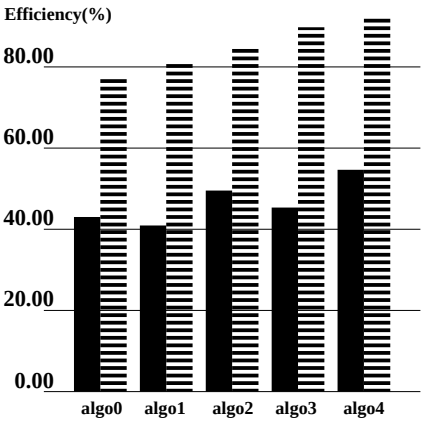

図7　シミュレーション結果 (セット J2、 4 進 2 段)

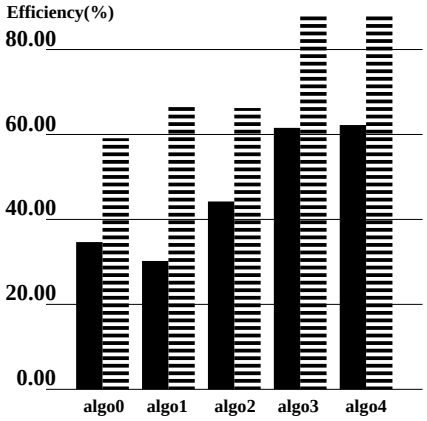

図6　シミュレーション結果 (セット K2、 2 進 4 段)

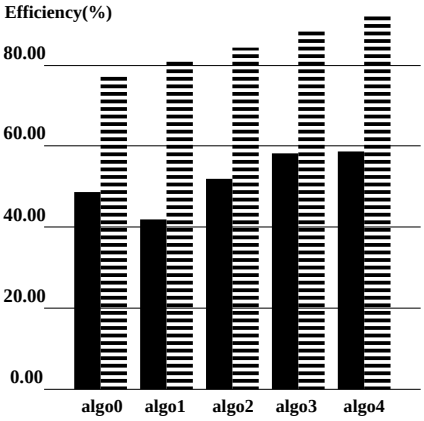

図8　シミュレーション結果 (セット K2、 4 進 2 段)

6. おわりに

汎用並列 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について、シミュレーションにより評価を行なった。その結果、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た。その原因は、ページの移動や置換による悪影響を抑える効果によるためであると推定される。またページの置換方式については、参照頻度及び再アクセスのコストが小さなメモリページから置換を行なう方式が良い結果を示し、コピーページによるメモリ溢れが起きる程度の負荷では、この方式を採用すれば複数プロセスの実行に十分な効率を得られることが期待できる。

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) Thomas E. Anderson, Brian N. Bershad, Edward D.Lazowska, , and Henry M. Levy. Scheduler activations: Effective kernel support for the user-level management of parallelism. *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, Vol.25, No.5, pp. 95–109, October 1991.

2) 松本尚, 古荘進一, 平木敬. 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.

3) 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.

4) 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.

5) 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.

6) 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.