汎用超並列OS *SSS-CORE* におけるスケジューリング方式

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，有効に使用するスケジューリングを行い，システム全体の性能をあげることが必須である．本論文では，並列プロセスごと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ページ置換方式を提案する．さらに，提案した方式を詳細確率モデル上で，具体的なメモリ管理方式，アクセス頻度およびアクセスコストを付加したシミュレーションにより様々な使用状況の下で評価する．評価の結果，実メモリの局所性を利用したスケジューリング方式と，置換するメモリを分散共有メモリモデル上のページ属性を直接利用して求める方式が優れた性能を与えることが示された．

Scheduling Mechanisms for Scalable Parallel OS *SSS-CORE*

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 reflect 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 information 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合することにより，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[6]や並列環境下での各種最適化技法の研究開発[8),9)]によりNUMA型並列計算機上での高効率な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・マルチプロセスという使用形態において複数の並列アプリケーションの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や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およびプロセス☆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)]．プロセス単体の実行性能を考慮した場合，同期待ちを極力減らすことが可能なgang schedulingは有効である．しかしながら，gang schedulingは時間的スケジューリングを解決するだけであり，プロセッサ割付けの問題は解決しない．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が性能低下の原因となる．

メモリへのアクセスコストが距離によって異なるNUMA型並列計算機では，スケジューリングに実資

† アーツテック株式会社
Arts Tech Inc.
†† 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

☆ 以下，特に述べない限り，並列プロセスを表す．

源の使用状況や通信コスト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[7]があるが，容量の小さなキャッシュでは効果が出ない．一方，大規模NUMA型並列計算機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みを付加することが必要である．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するが，プロセス数に基づいた負荷情報[3]のみの利用にとどまっている．Paragonでは柔軟なプロセッサ分割が可能であるが，資源情報は利用せず，また各プロセッサ上でのスケジューリングはカーネルだけが実行するため柔軟性を持たない．

本論文では，階層化された大規模NUMA環境における汎用で高効率なスケジューリングおよびページ置換方法を求め，評価する．まず，2章で汎用超並列オペレーティング・システム *SSS-CORE*[10] におけるスケジューリング方式の概略，すなわち，階層的に資源情報を管理する資源管理木，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ページ置換方式を示し，3章ではスケジューリングのシミュレーションモデルを示す．4章においてそれらのシミュレーションによる評価を示す．最後に5章でまとめる．

2. *SSS-CORE* におけるスケジューリング方式

SSS-CORE[10] 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・マルチジョブ環境と超高速通信環境を提供し，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，メモリアクセスコストおよび通信コストがその距離に影響され，またメモリアクセスと通信の同期への効果が大きく実行効率に影響する．したがって，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求められる．

それを支援するため *SSS-CORE* では，ユーザとカーネル間で大域的資源情報を階層化して管理し，資源情報の受渡しのための機構として資源管理木を用いる．*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，ユーザからのスケジューリング制約の利用を下記に示す方法で実現している．

2.1 *SSS-CORE* の資源管理機構

大規模なNUMA型システムにおいては，資源間の距離の自然な表現，資源管理に用いる資源量の制約および資源情報を管理するためのオーバヘッドを考慮すると，階層的な資源管理が必須である．階層化された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いる．本論文で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，プロセスごとの資源の使用状況等を管理する．資源管理木の構造を実システムの構成に適応させることで，様々な接続構造を持つシステムをサポートすることが可能である．たとえば，階層的に接続されたワークステーションクラスタにおいては，資源管理木は要素処理装置間の距離を自然に表現する．また，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を近似的に表現する．これらの例では，仮想的な資源管理木におけるノード間距離が，割当てを行う要素処理装置間の通信コストを近似することにより，通信コストを考慮したスケジューリングを可能としている．

資源管理木では各ノードごとに当該ノードをルートとする部分木の領域に含まれる要素プロセッサと実ページの総数，フリーな要素プロセッサと実ページの総数，ならびにプロセスごとのIDと使用している要素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，プロセスごと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プロセッサ構成における資源管理木を示す．

2.1.1 スケジューリング制約

現在の並列化コンパイラは，単一のプロセスが資源を占有して実行することを仮定しているため，多くのプロセスが資源を共有するマルチユーザ・マルチジョブ汎用環境ではコンパイル時の仮定から逸脱することによる実行時効率の低下が現れる．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要求を伝える．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．また台数可変制約では少しずつプロセスに要素プロセッサを割り当てることが可能となる．たとえば，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

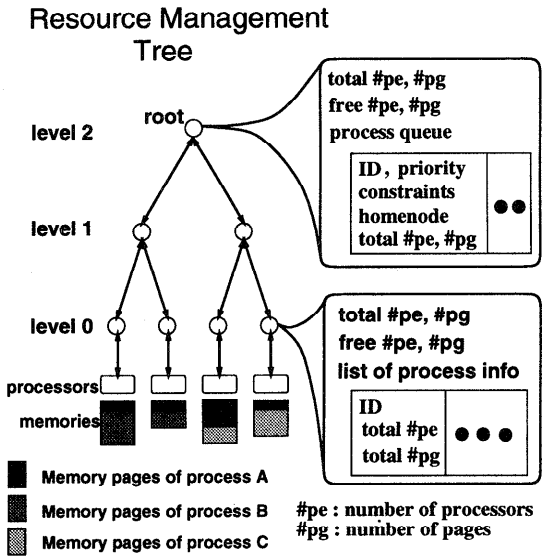

図1 資源管理木
Fig. 1 Resource management tree.

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．

2.1.2 スケジューリング優先度

マルチジョブ・マルチプロセスを実現する汎用オペレーティングシステムでは，各々のジョブ・プロセスが，重み付けされた資源利用量などを基準にして同じ優先度となるよう，公平性を持ったスケジューリングを行うことが求められる．並列実行の効率化を目的としたスケジューリング，スケジューリング制約に基づくスケジューリングは，各プロセスごとの効率に沿って資源割当てを行うため，プロセス間の公平性を保たない．*SSS-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し，優先度に対してエージングを施すことにより，スケジューリングにおけるプロセスの公平性を保つ．

優先度は値が小さいほど高い．まず以下の値を計算する．

- 使用した資源量：
  $U_r = C_p *$ 使用プロセッサ総数+
  $C_m *$ 使用ページ総数
- 制約条件の強度：
  $R_c =$ 要求プロセッサ台数/総プロセッサ数
- 制約条件の満足度：
  $S_c = 0\ or\ 1$
- 無駄にした資源量：
  $W_r = C_{wp} * nwp *$
  要求プロセッサ台数/総プロセッサ数

表1 優先度計算のための係数の値
Table 1 Coefficients for priority calculation.

優先度計算用係数	値
C_p	1.0
C_m	0.005
C_{wp}	1.0
C_r	1.0

- 待ちプロセスの有無：$f_w = 0\ or\ 1$
- 若返り系数：C_r

ここで，nwp は直前の時分割で使用されなかった要素プロセッサの数で，直前までスケジュールされていたプロセスの間で加重配分した値 W_r をペナルティとする．これらの値を（一部はプロセスごとに）計算すると，次の式によりそのプロセスごとの優先度のエージング値が求められる．

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

プロセス実行は，タイムスライス時にスケジュールされるか，されないかいずれかのため，$t = 0\ or\ 1$ である．

また，優先度の値の発散を防ぎ公平性を実現するため，直前までスケジュールされていたプロセスの優先度の総和を，待ち状態だったプロセスにエージング量として均等に分配する．

性能評価時のシミュレーションでは各係数の値として，**表1** にある値を利用した．表1では，要素プロセッサ1台とメモリ200ページを使用することが等価であり，C_{wp}，C_r は本論文におけるシミュレーションにおいて大きな性能差を生まないため各々 C_p と同じ値とした．

2.2 *SSS-CORE* のスケジューリング戦略

SSS-CORE は，ユーザとカーネル分離の2レベルスケジューリングを採用する．カーネルスケジューラは各プロセスが提示するスケジューリング制約（2.1.1項）と優先度に従って資源を，階層化管理された資源情報をもとにプロセス単位に割り当てる．各プロセスはカーネルスケジューラにより割り当てられた資源をユーザスレッド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．

SSS-CORE におけるカーネルスケジューラにおけるスケジューリング方式として下記の5つのスケジューリング方式を用いる．各方式ともに，タイムスライスごとに前記の資源使用のエージングを考慮した優先度計算を行い，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ルする．なお，要素プロセッサとメモリの対をクラスタと呼ぶ．

algo0 ランダムに必要な数だけクラスタを選択し割

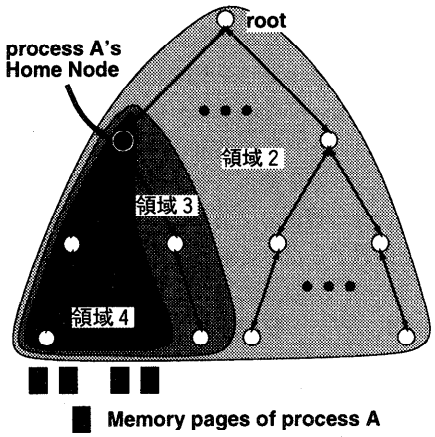

図2 資源管理木とスケジューリング対象領域
Fig. 2 Target areas of the scheduling in the resource management tree.

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てを試み，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 初めにホームノード以下の領域でページのある領域を割り当て，失敗するとホームノード以下の領域でページのない領域から割り当てる方式

algo4 ホームノード以下の領域で厳密にページが存在する領域のみ割当てを試み，失敗すると割当てを諦める方式

ただし，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クラスタ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，1つ前のスケジューリング時の割当て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あたるノードである．

台数固定スケジューリング制約を用いている場合，algo2, 3, 4では割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リングは諦め，スケジュール可能になるまで待つ．

5つの方式のうちalgo2, 3, 4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クラスタの割当て領域が，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が割当て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

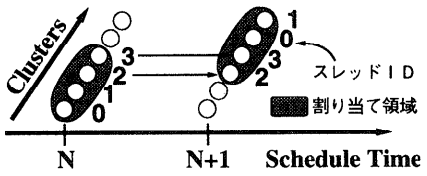

図3 ユーザレベル・スケジューリング例
Fig. 3 An example of user-level scheduling.

割当て位置で，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，プロセスごと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てを試みる．前回のスケジュール時の割当て領域が，他のプロセスにすでに割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てを試みる．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，その逆順でクラスタの利用数が多くなる可能性が高い．

また次節以降でのシミュレーションで用いた，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，以下のとおりである．

- 前回の割当て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

図3 にスレッド・スケジューリングの例を示す．ここでは，割当て領域が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．

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．

☆ 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の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）で足りない分のクラスタを確保する．

2.3 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，個々のクラスタ上の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．この場合の資源管理では，実行に必要な新たなページを割り当てるために置換するページの選択方式が実行効率に大きく影響する．

メモリシステムが分散共有メモリである場合には，ページの共有状態からアクセス頻度および再アクセス時のアクセスコストの違いを分類することが可能である．ページの分類ができれば，アクセス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ことが実行効率向上のために求められる．

一般的に，並列プロセスの場合共有領域のアクセス率は非共有領域に比べて小さいと仮定できる．また，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が違うと仮定できる．そこで，ただちに実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，性能向上を図ることが可能である．以上をふ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アクセス頻度や容量，プロセスの組合せ，ネットワークおよび二次記憶の性能に依存する．

今回の評価では，次の2つの置換方式を実装し比較を行った．

置換方式0 ページの種類および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上記ページの種類順に選択．各プロセスの共有または非共有空間ごとにページの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．

3. シミュレーションモデル

本論文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[11),12)]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共有メモリ環境である．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2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．

プロセスは要求クラスタ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．1クラスタにおける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アクセス頻度表を与える．各空間におけるメモリアクセス列は，メモリアクセス頻度表における総頻度に対する各ページのメモリアクセス数を比として確率的に生成する．

共有空間は分散共有メモリシステムで実現する．一貫性管理はsequential consistency modelに則ったアップデート方式で，したがって，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．メモリアクセスコストは，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．プロセスの割当て領域が変化した場合には，現在の割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，同期待ちまたはページ待ち状態でなければ各クロックごとにメモリアクセスを行う．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（すなわちクロック数）とは，ページ待ちおよび同期待ち以外の動作をした時間のことである．

4. 実験および考察

実験としては**表2**のようなパラメータを採用し，プロセッサ256台の2種類のシステム（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3**の各プロセスセットを実行した．実験は，プロセスが1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4**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，グラフの縦軸は，プロセッサ時間の内訳を相対比率で表したものである．次に，ページ置換方式の比較結果を**図5**に示す．図5ではスケジューリング方式algo4，w488のネットワーク構成を用いている．これは図4においてプロセスセットA，L，M，Rについてalgo4がつねに安定して最も優れた性能を示すことに基づいている．また，プロセスセットA，L，M，Rを用いたスケジューリング方式の比較結果を**図6**に示す．ここでは，ページ置換方式1を用いている．

4.1 スケジューリング方式について

A以外のプロセスセットでは，共有ページのコピーが生成されることでページ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当て領域を固定した場合，プロセスセットの組合せの性質から十分にクラスタを活用しないため，実際のシステムではアイドル時間が減る可能性として，以下のことがあげられる．

(1) 台数可変制約の採用する場合には，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．

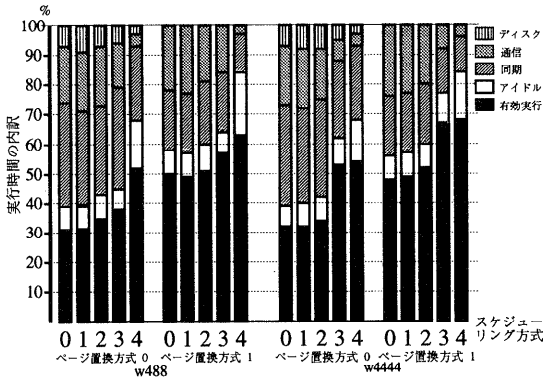

図4 プロセスセットAのシミュレーション結果
Fig. 4 Simulation result of the process set A.

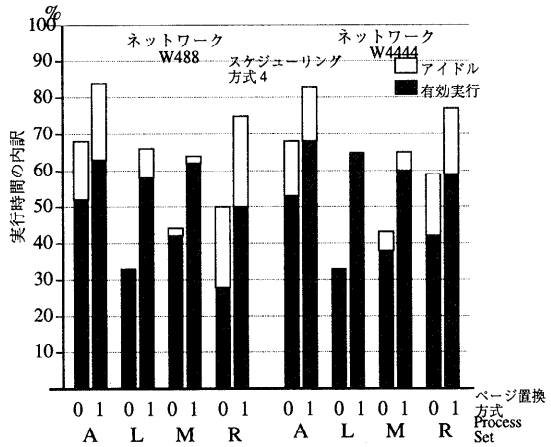

図5 ページ置換方式の比較（スケジューリング方式4）
Fig. 5 Comparison of the page replacement policy (scheduling policy 4).

表2 各種コストおよびパラメータ
Table 2 Costs and parameters.

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

表3 プロセスセットの概要
Table 3 Description of process sets.

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
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
R	14	16,36,48,50 (2),64,70,96	1514	65320	14980	1.67	1000–3000

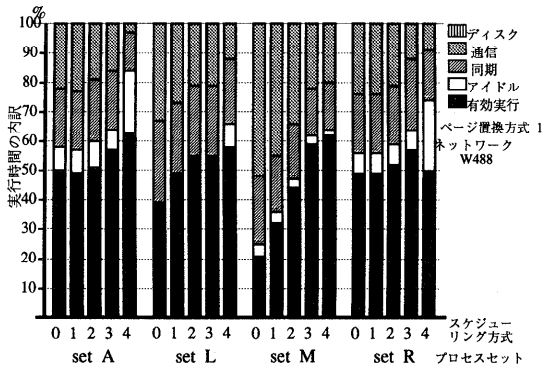

図6 プロセスセットA, L, M, Rのシミュレーション結果
Fig. 6 Simulation results of process sets A, L, M, R.

(2) 特定のプロセスセット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，プロセスが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が，図4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，メモリアクセス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☆プロセスセット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 ページ置換方式について

ページ置換方式については方式1が方式0につねに勝っている．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，よりアクセスされやす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きわ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ページ置換を行うことで，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．

今回の実験で最も性能の良い，スケジューリング方式algo4で置換方式1の場合の結果で，有効実行率が50%から68%程度である．共有ページの一貫性管理がsequential consistency model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☆☆の採用によりさらに向上することが可能である．また，緩和されたメモリモデルを使えば，ページを移動したとき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．

また本論文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，本論文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. 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，資源管理木を利用した，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，ページ置換方式については，ページの分類に従ったページ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，負荷状況が高いケースでも有効実行率による評価が50%程度以上である．

SSS-CORE では，台数可変制約を用いるが，今回評価した有効実行率に加え各クラスタがアイドルする時間も有効な実行時間に組み入れることが可能となる．したがって，図6より65%から85%の実行時間が有効に計算に用いられる．これらのことから本論文で提案した方式により実用的に十分なレベルのスケジュー

☆ プロセスの割当て領域の重なり具合で，一部領域においてページ置換が発生している．

☆☆ release consistency model や weak consistency model 等同期と逐次化を sequential consistency model と比較して減少させたメモリコンシステンシモデル

リング性能が得られていると考えられる.

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った.

参考文献

1) Chandra, R., et al.: Scheduling and Page Migration for Multiprocessor Compute Servers, *ACM, ASPLOS VI*, pp.12–24 (Oct. 1994).
2) Intel Supercomputer Systems Division: Paragon User's Guide, order number 312489-003 edition (June 1994).
3) Feitelson, D.G.: Packing schemes for gang scheduling, *Proc. IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, pp.54–66 (Apr. 1996).
4) Feitelson, D.G. and Rudolph, L.: Distributed Hierarchical Control for Parallel Processing. *IEEE Computer*, Vol.23, No.5, pp.65–77 (1990).
5) Ousterhout, J.K.: Scheduling techniques for concurrent systems, *3rd Intl. Conf. Distributed Computer Systems*, pp.22–30 (Oct. 1982).
6) Anderson, T.E., et al.: Scheduler activations: Effective kernel support for the user-level management of parallelism, *Proc.13th ACM Sympo. on Operating Systems Principles*, Vol.25, No.5, pp.95–109 (1991).
7) Vaswani, R. and Zahorjan, J.: The implication of cache affinity on processor scheduling, *Proc. 13th ACM Symp. on Operating Systems Principles*, pp.26–40 (Oct. 1991).
8) 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 (1989).
9) 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 (1989).
10) 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 (Oct. 1994).
11) 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，情報処理学会研究報告，96-OS-73, 96 (79) (1996).
12) 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，情報処理学会研究報告，96-ARC-117, 96 (23): 19–24 (1996).

(平成 9 年 10 月 31 日受付)
(平成 10 年 4 月 3 日採録)

信国陽二郎

1971 年生．1997 年東京大学大学院理学系研究科情報科学専攻修士課程修了．理学修士．同年 4 月から（株）アーツテックに勤務．並列分散オペレーティングシステムに興味を持つ.

松本 尚（正会員）

1962 年生．1985 年東京大学工学部計数工学科卒業．1987 年大阪市立大学大学院理学研究科物理学専攻修士課程修了．日本アイ・ビー・エム（株）東京基礎研究所研究員を経て，1991 年 11 月より東京大学大学院理学系研究科情報科学専攻助手．並列計算機アーキテクチャ，並列分散オペレーティングシステム，最適化コンパイラに関する研究に従事．他に数値計算による制約解消系，グラフィックス，ニューラルネットワーク等に興味を持つ．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員.

平木 敬（正会員）

1976 年東京大学理学部物理学科卒業．1982 年同大学大学院理学系研究科物理学専攻博士課程修了．理学博士．1982 年通商産業省工業技術院電子技術総合研究所入所．1988 年より 2 年間 IBM 社 T.J. Watson 研究センタ客員研究員．1990 年より東京大学理学部情報科学科（現在大学院理学系研究科情報科学専攻）に勤務．現在，超並列アーキテクチャ，超並列超分散計算，並列オペレーティングシステム，ネットワークアーキテクチャ等の高速計算システムの研究に従事．日本ソフトウェア科学会会員.