汎用超並列 OS *SSS–CORE* の資源スケジューリング方式の評価

信国 陽二郎[†] 松本 尚[††] 平木 敬[††]
Yojiro Nobukuni Takashi Matsumoto Kei Hiraki

†(株)アーツテック Arts Teck Inc.
††東京大学理学系研究科 University of Tokyo

概 要

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。本稿では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する。

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 [6]、並列環境下での各種最適化技法の研究 / 開発 [9] [8]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 [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 [7]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 [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。

本稿では汎用超並列オペレーティング・システム *SSS–CORE* [10]における資源管理機構の概略述べ、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。

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 [13] [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 [11]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

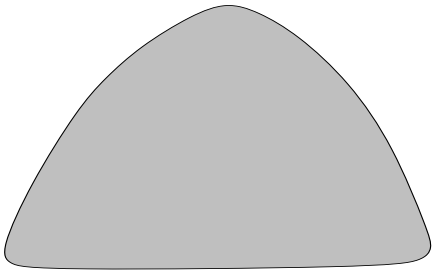

図 1 資源管理木とスケジューリング対象領域

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

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

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

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。

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	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

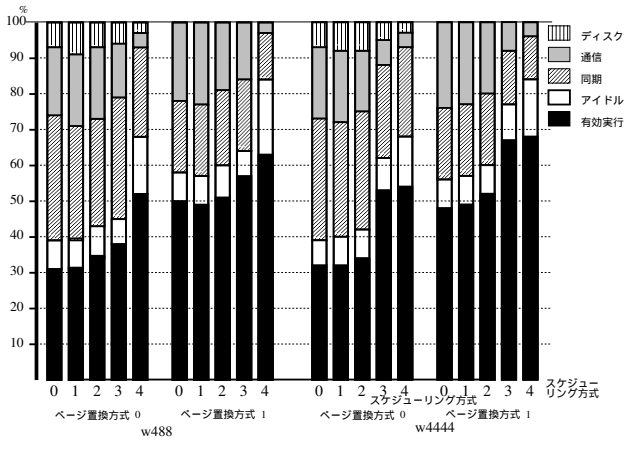

図 2　プロセスセット A のシミュレーション結果

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。

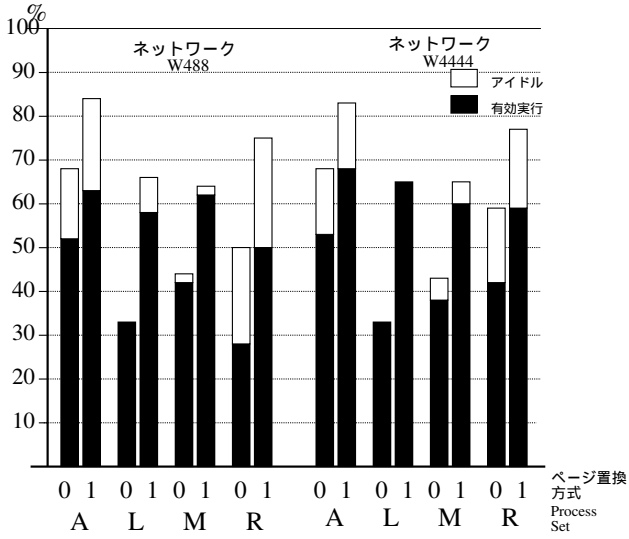

図 3　ページ置換方式の比較

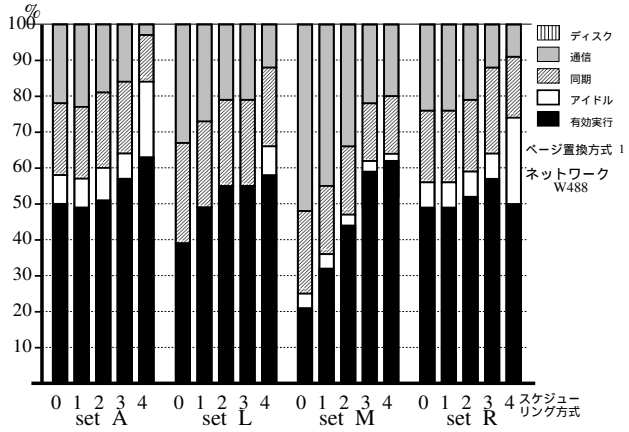

図 4　プロセスセット A、L、M、R のシミュレーション結果

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 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。

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*, pages 12–24, October 1994.

[2] Intel Supercomputer Systems 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*, pages 54–66, April 1996.

[4] D.G. Feitelson and L. Rudolph. Distributed Hierarchical Control for Parallel Processing. *IEEE Computer*, 23(5):65–77, May 1990.

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

[6] 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*, 25(5):95–109, October 1991.

[7] R. Vaswani and J. Zahorjan. The implication of cache affinity on processor scheduling. In *Proc. of 13th ACM Symp. on Operating Systems Principles*, pages 26–40, October 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*, pages 1–8, November 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, 31(12):37–42, August 1989.

[10] 松本尚 et al. 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, pages 13–16, October 1994.

[11] 信国陽二郎 et al. 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, 95(210):111–118, August 1995.

[12] 信国陽二郎 et al. 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*, 96(79), August 1996.

[13] 信国陽二郎 et al. 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*, 96(23):19–24, March 1996.

†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。