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

信国陽二郎　　松本尚　　平木敬

東京大学 大学院理学系研究科 情報科学専攻
〒 113 東京都文京区本郷 7–3–1
Email: {nobukuni, tm, hiraki}@is.s.u-tokyo.ac.jp

あらまし　エレクトロニクス技術の向上によるネットワーク周辺の性能向上や並列環境下での各種最適化技法の開発を受け、分散メモリ環境での並列処理が現実味を帯びてきた。本稿では、ＮＵＭＡ型並列計算機やワークステーションクラスタ上に汎用並列 OS を構築するためのカーネルレベル・スケジューリングについて検討する。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必要である。並列プロセス毎に所有する実ページ情報を管理し、その情報にしたがってスケジューリングを行なった。シミュレーションにより、簡略化されたモデル上でのスケジューリング法の比較を行なった。

キーワード　NUMA, 汎用並列 OS, カーネルレベル・スケジューリング

Evaluation Of A Scheduling Method Using Resource Informations For General Purpose Parallel OS

Yojiro NOBUKUNI　　Takashi MATSUMOTO　　Kei HIRAKI

Department of Information Science, Faculty of Science, University of Tokyo
3-7-1 Hongo Bunkyo-ku Tokyo 113, Japan
Email: {nobukuni, tm, hiraki}@is.s.u-tokyo.ac.jp

Abstract　Parallel processing on parallel/distributed systems is showing greater availability as network-surrounding micro-electronics evolves and many optimization mechanisms are realized. This paper describes a kernel-level scheduling method to build a general purpose parallel OS on NUMA-type parallel machines. In a distributed memory environment, constructing a multi-user/multi-job world without decreasing high efficiency of parallel applicatoins can be achieved by managing resource informations and scheduling according to them. We simulated on a simplified model and evaluated four scheduling methods.

key words　NUMA, parallel OS, kernel level scheduling

1 はじめに

NUMA 型システムは、より単純でより安価な構成要素をより多数結び付けて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。そこで、 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・システムの開発が求められる。

我々が開発中の SSS–CORE[2] は、 NUMA 型並列計算機上に時分割 とパーティショニ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を目標とする。 NUMA 型並列計算機ではメモリアクセスのコスト及び要素プロセッサ間の通信コストが一様ではない。従って、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になる。 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。

本稿では、クラスタ構成で n 段 k 完全木の形をした対称かつ均質なシステムを簡略化してモデル化し、台数固定制約のみを用いたスケジューリング法を実装し、シミュレーションを行なった。更に、実メモリの使用状況をスケジューリングへ反映することの有効性の検証を行なった。

2 SSS–CORE

現在開発中の SSS–CORE は、 NUMA 型並列計算機を対象とした汎用並列ＯＳである。対象とする NUMA 型並列計算機のインターコネクションネットワークの形状は問わない。時分割 とパーティショニ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を目標とする。

NUMA 型の並列計算機システムでは、メモリアクセスコスト及び通信コストが、距離に影響される。これらコストが小さく抑えられるようにスケジューリングすると、個々の並列アプリケーションは効率良く実行される。ただしこれらコストが与える影響はプロセスに固有のものであり、従って受ける影響もさまざまである。

SSS–CORE では、計算機のインターコネクションネットワークの階層構造と一致した構造の資源管理構造を用意して資源情報の管理をする。また、各プロセスはスケジューリングに対して制約を提示できる。カーネルレベル・スケジューラは資源情報と、プロセスから提示された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となる。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同じであるために、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。ただし、プロセス内での資源割り当てはユーザレベル・スケジューラに一任される。

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達成する。

SSS–CORE ではこの他にも、システムの性能向上のための機構 [2] を取り込む。対象並列計算機では二次記憶へのアクセスはプロセッサパワーを消費しないで行なえる想定になっている。そのため、時分割の計算中に次の時分割でスケジュールされるプロセスのための二次記憶のプリフェッチを行なう。これを支援するため、ユーザレベルからカーネルレベルへ必要なデータ領域のヒントを示す。カーネルレベル・スケジューリングを 1 タイムスライス分先行しておこない、以上の機構を利用する。

3 スケジューリング法

SSS–CORE ではタイムスライスの間隔を 100msec から数秒程度の比較的大きな単位を仮定しているので、スケジューリングに必要とされる処理コストは無視できるものと仮定する。従ってシミュレーションではスケジューリングは実行時間を 0 として実行され、シミュレーションの評価には含まれていない。またこの仮定のため、資源管理木のデータ構造の実メモリ上へのマッピングやスケジューリングの計算方法、さらにカーネルへのユーザレベルからのスケジューリング制約の開示法 (インタフェース、実装法) は本稿では理想的なものを仮定する。

3.1 資源管理木

仮想的な資源管理構造を用意し、これにシステムの階層構造と同一の木構造を持たせる。本稿ではこの構造のことを資源管理木と呼ぶ。資源管理木の各ノードでは、そのノード以下に含まれるプロセッサと実ページの総数及び、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 及び使用され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) を持つ。図 1に 4 プロセッサ構成システムの場合の例を示す。

3.2 スケジューリング制約

今回のシミュレータに実装したカーネルレベル・スケジューリング法では、台数固定制約 のみを扱う。台数固定

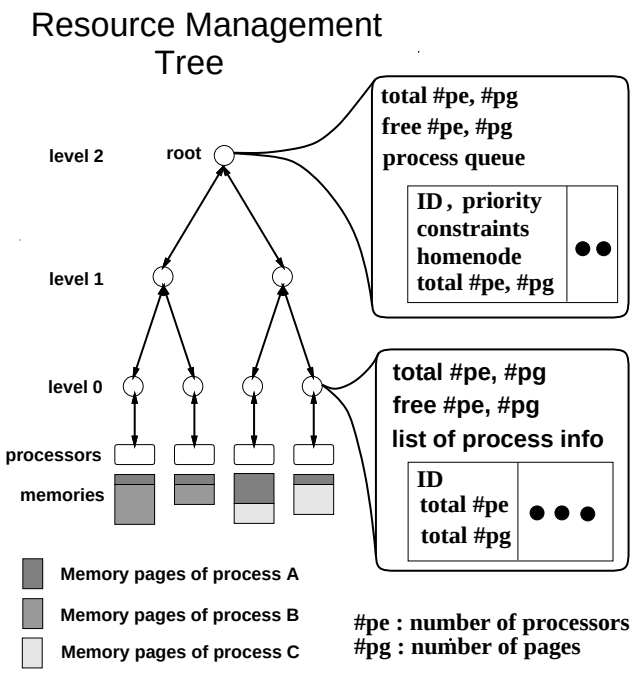

図 1: 資源管理木

制約では、各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の一定の数を指定することができる。このシミュレーションでは、プロセスにスケジューリング制約を変更すること (台数固定制約では希望する一定台数を変更すること) を認めていないので、プロセスは生成されてから消滅するまで同じ並列度を持つ。

3.3 優先度の計算法

優先度は値が小さいほど高い。優先度の計算法を以下に示す。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
- 制約条件の厳しさ：
  $R_c =$ 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
- 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 をペナルティーとしている。これらの値を (一部はプロセス毎に) 計算すると、次の式によりそのプロセス毎のエイジングの値が求められる。

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

シミュレーション中のプロセスは、タイムスライス時にスケジュールされるか、されないかのため、 $t = 0\ or\ 1$ である。

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均等に分配する。

各係数の値としてシミュレーションでは、表 1にあるように簡単な値を利用した。 C_m の値は、 1 メモリ当たりのページ数を 400 としたシミュレーションでの値である。 1 メモリの半分の 200 ページを占有することと要素プロセッサ 1 台を使用することとが 1:1 になるように調節されている。

表 1: 優先度計算のための係数の値

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

3.4 スケジューリング・アルゴリズム

今回のシミュレータに実装したカーネルレベル・スケジューリング法の計算法を説明する。まず、言葉を定義する。

ホームノード プロセスが前回のスケジューリングで割り当てられていた全てのプロセッサを含むようなノードのこと (図 2)。各プロセスは資源管理木のノードのうち一つを自分のホームノードとする。

タイムスライスが来る度に 3.3節に示した方法で、各プロセスの優先度を計算し直す。新しい優先度が高いプロセスから順に、以下に示すアルゴリズムに従ってスケジューリングを行なう。

1. 新しいプロセス又は実ページを 1 ページも持たないプロセスの場合は、まずホームノードを決定し、そのノードに移動する。それには資源管理木のルートノードから始めて、要求台数以上のプロセッサを持ちかつフリーなページを一番多くもつ子ノードを降りていく。疑似コードにすると図 3のようになる。
   それ以外のプロセスの場合には、資源管理木のルートノードを見て自分のホームノードに移動し、次のステップに進む。
2. もし現在のノード以下に十分なフリープロセッサがなければ、割り当ては失敗。 0 を返して終了。そうでなければ次のステップへ。
3. 現在のノードの内、どの子供にそれぞれ何台ずつ割当を依頼するか決定する。その際、そのプロセスがページを占有するノードを優先して依頼する。

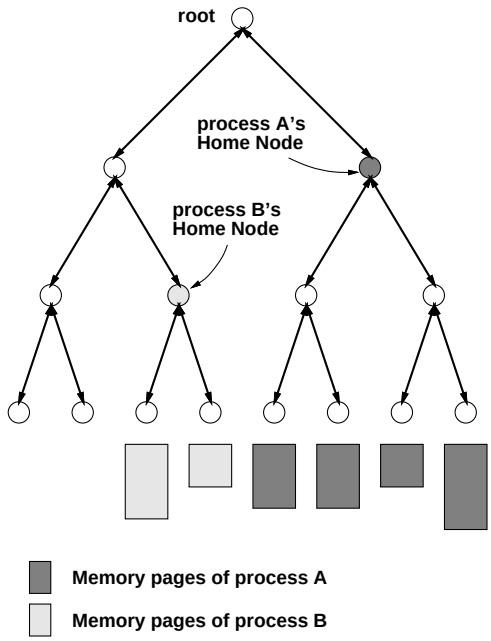

図 2: 管理木におけるホームノードの例

4. 各子供に対して、それぞれ決められた台数で割当を試みる。即ち各子供に対して、第 2 ステップ以降を実行する。

5. 試みた全ての子供で割当が成功したなら、割り当てに成功。このノード以下に割り当てた台数を返して終了。もし、 1 つでも失敗したら割り当て失敗なので、 0 を返して終了する。

以上を疑似コードで表すと図 4のようになる。

4 シミュレーション・モデル

シミュレーション自体の実行時間への制約から簡略化されたモデルを構築した。スケジューリングの性能評価の一指標として、ページフォールトが原因のページ待ちと同期待ち以外に利用されたクロックの割合を調査できるようにした。本稿ではページ待ち / 同期待ち以外の動作を有効実行と呼び、そのような動作を起こしたクロック (時間) のことを、有効実行クロック (時間) と呼ぶ。

4.1 ネットワークモデル

階層性のある木構造をしたインターコネクションネットワークを持ったシステムをモデル化する。イーサネットや ATM スウィッチでつながれた階層構造を持ったワークステーションクラスタや、複数のクラスタボードを層状につなぎ合わせた並列計算機がこれに当たる (図 5)。シミュレータはネットワークの木構造の階層の深さと、階層内のクラスタ数をパラメータとする。つまり、 n 段 k 完全木を扱うことができる。したがってネットワークの形状は対称性を持ち、簡単にするため要素プロセッサは全て同一のも

```
NODE
FindHomeNode(
   int reqPEs,
   NODE this_node)
{
   NODE cand; /* CAndidate NoDe */

   if (this_node is a leaf node)
      return this_node;

   if (reqPEs > total PEs of a child)
      return this_node;

   cand = child[0];
   for each child[i]
      if (child[i]'s free PEs > reqPEs &&
         child[i]'s free pages > cand's free pages)
         cand = child[i]
   return FindHomeNode(reqPEs, cand);
}
```

図 3: ホームノード探索アルゴリズム

のと仮定する。今回の実験では 4 段構成の 2 分木 (すなわち、各階層で 2 つのクラスタを持つ) で、 1 6 台の要素プロセッサを持ったシステムをシミュレーションした。

実際のシステムでは、 1 つの結合線上を同時に複数のパケットが流れることはできない。従ってネットワークを厳密にシミュレートするには全ての結合線をプログラムする必要がある。これには多大の計算時間を要求し、分散 / 並列環境でのＯＳレベルのシミュレーションを実メモリの振舞いに注目して行なう場合、それ以外の計算量が大きいためにふさわしくない。そこで今回は、ネットワーク上での競合は全く無視した。したがってこのモデルでは、ページの転送、ページの処理用のメッセージ、さらにプロセッサ間通信用のメッセージが、同一線上を同時に複数ながれている。そのため、 1 つの要素プロセッサにおいて送信 / 受信されるメッセージは、必要があれば同時に複数処理される。

コストには、ディスクへのアクセスコスト、ページの転送コストとそれ以外の要求 / 通信メッセージの 3 種類がある。しかし、 SSS–CORE では二次記憶へのアクセス、リモートなメモリへのアクセス、そしてクラスタ内メモリへのアクセスはこの順にコストがかかると仮定している。特にリモートなメモリアクセスの距離の違いを反映するために、通信やメモリアクセスのコストは転送する間に通過するネットワークの段数に比例するものとする。

```
int
ALLOCATE(
  int reqPEs,
  NODE this_node
  PROCESS P)
{
  int # prep=0; /* total # PEs can be prepared.*/
  int # alloc=0; /* total # PEs allocated .*/
  int req[i]; /* # PEs to ask to child[i]. */

  if (this_node's free PEs < reqPEs) return 0;

  if (this_node is a leaf node)
    allocate this_node's PE to P;

  /* among children, in a fixed order, decicde
    how many PEs to ask to each child. */
  for each child[i] below which P occupy ANY page
    req[i] = MIN(reqPEs−# prep, his free PEs);
    # prep += req[i];
  for each child[i] below which P occupy NO pages
    req[i] = MIN(reqPEs−# prep, his free PEs);
    # prep += req[i];

  if (# prep < reqPEs) return 0;
  else
    for each child[i]
      alloc += ALLOCATE(req[i], child[i], P);

  return alloc;
}
```

図 4: アロケーション・アルゴリズム

4.2 メモリモデル

システム全体に分散共有メモリが用意されているものとする。こう仮定することで、スケジューリングで利用するために管理されるページの情報がシステム全体で一貫して管理されることになる。その他の特徴は以下のようになる。

- あるページが初めてディスクから読まれるとホームページとなる。アクセスすべきページがローカルに存在しない時は、そのページのホームページを持つクラスタにページ要求メッセージを送信し、そのページのコピーを受けとる。ページ毎にコピーページが作られた順を記録し、ホームページがメモリからリプレースされる際には、FIFO 順にコピーページが新しいホームページとなる。

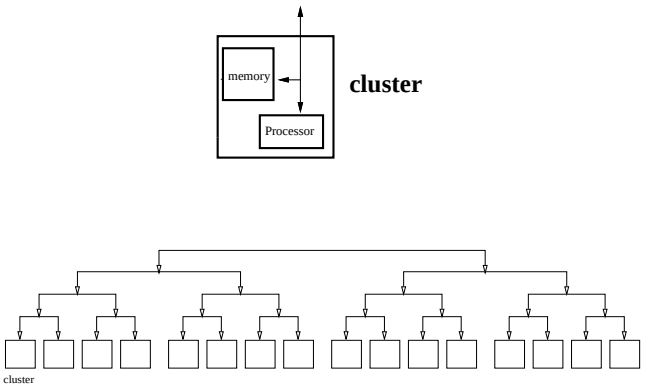

図 5: 対象計算機システム

- 各メモリ内ではページのリプレースは LRU で行なわれる。ただしリプレースはコピーページから行なわれ、ホームページはコピーページに優先してメモリに残される。

- ページのコヒーレンシ管理は無視し、リモートアクセスの度にコピーページが生成されローカルメモリに転送される。つまり一度生成されたページはメモリーからリプレースされるまで存在する。

- 二次記憶へのアクセス、リモートなメモリへのアクセス、そしてクラスタ内メモリへのアクセスはこの順にコストがかかると仮定している。式で表すと

  $$\text{クラスタ内} \ll \text{クラスタ間} \ll \text{二次記憶}$$

  従って各種コストパラメータは、これの仮定に違反しない値が与えられる。

4.3 プロセスの実行モデル

シミュレーションの開始時に全てのプロセスが生成される。また、一度生成されたプロセスは、スケジューリング制約を変更することはない。すなわち、プロセスは固定された並列度を持つ。ただし与えられた条件を満たしたとき、プロセスは終了する。

また、シミュレーションはクロックベースに行なわる。プロセスに用意された振舞いは以下のものであり、クロック毎に可能な一つが選択される。

演算 メモリへの参照を必要としない実行で、一般のレジスタ演算、メモリへの影響のないキャッシュへのアクセスが含まれる。シミュレータ上では何も動作を起こさない。

メモリ参照 リードとライトの区別はない。参照するアドレスは、二つのパラメータ U、V により計算される。確率 U でプロセスごとに与えられた空間内から一様に一つのアドレスが選ばれる。残りの $1-U$ の確率でプロセスの空間の四分点を平均とする、分散

が V の 3 つの正規分布のうちの一つに従がってアドレスを計算する。正規分布の選択は 3 つの内の一つが一様に選ばれる。

同期 プロセス毎に同期を起こす間隔を有効実行クロック数で与える。同期が起こるたびに、次の同期への参加プロセッサをその時点でそのプロセスがスケジュールされている要素プロセッサの中から乱数で選んでおく。要素プロセッサはこの値だけ有効実行を経験する度に同期待ちになる。同期に参加する要素プロセッサ全てが同期待ちになったところで、これらの要素プロセッサは各々他の同期待ちプロセッサに対して同期を表すメッセージを送信する。同期に参加する各要素プロセッサは、全ての同期参加プロセッサからこのメッセージを受信したところで、通常の実行に戻ることができる。こうすることにより、同期と同期の間にページフォールトが起これば、ページ待ちをした分だけ同期のタイミングが遅れることになる。図 6にシミュレーションでの同期の例を示した。

各要素プロセッサは、ページ待ちもしくは同期待ちでなければ以下のようにして動作を決定する。

1. 同期を起こすタイミングか調べる。そうであれば同期の処理をし、それ以外の時には次のステップに進む。
2. パラメータとしてプロセスに共通に与えられた、演算 / メモリ参照 の起こる頻度に従って、確率的にこれらの内一つを選択する。

5 シミュレーション

5.1 実験したスケジューリング法

4 つのスケジューリング法を実装し、それぞれシミュレータで実行し比較・評価を行なった。どのスケジューリングの場合も、 3.3節で説明した優先度の計算法を用いてプロセスの優先度の計算をし、各タイムスライス毎に優先度の高いプロセスからそれぞれの方法にしたがってスケジューリングを試みる。

algo0 ランダムなスケジューリング法。各プロセスに対して制約で示された必要台数分の要素プロセッサをランダムに選び割り当てる。物理ページの使用状況とは無関係に、未割当な要素プロセッサがある間はスケジュールを試みる。

algo1 端から順にプロセッサを割り当てるスケジューリング法。各プロセスに対して制約で指定された必要台数分の要素プロセッサを、システムの端から常に同じ順序で割り当てる。

algo2 algo3 と基本的には同じ方法。ただし、プロセスがより多くページを持つ範囲のプロセッサがすでに優先度のより高いプロセスに割り当てられていた場合には、資源管理木のルートノードからたどり、なるべく多く自分のページが存在する領域にスケジューリングする。

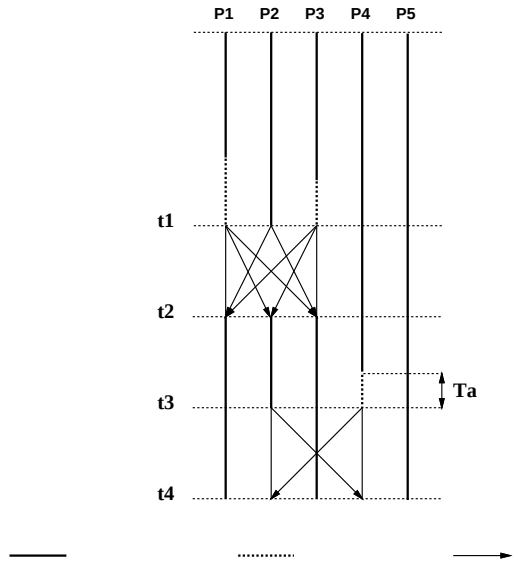

図 6: シミュレータ上での同期の実現法

algo3 3.4節で説明した本稿で提案するスケジューリング法。

5.2 シミュレーション・パラメータ

使用したパラメータは表 2、表 3の通り。表 3の U、 V はそれぞれ、プロセスが参照するアドレスを生成する際の一様アクセスの確率、それ以外の場合に利用される正規分布の分散を表す。また、同期頻度は全てのプロセスで同じ値にした。比較的大きな時分割をシミュレーションできるように、ネットワーク上のコストパラメータの値を小さめにとり、シミュレーションでの 1 クロックは実際の 10 から 20 クロック程度であると仮定する。そのため、各プロセスの起こす振舞いはメモリの参照のみとしてシミュレーションを行なった。

5.3 シミュレーション結果及び評価

シミュレーションに使用されるプロセスはセット A（表 5）とセット B（表 6）の 2 組である。両表中で、空間サイズ欄は各プロセスに与えられたアドレス空間のページ数を表す。空間サイズの総数欄にはプロセッサ毎の要求プロセッサ数と空間サイズの積の総和を表示した。シミュレーションの開始時に全てのプロセスが生成され、スケジュー

表 2: 各種コスト

項目	値
ディスクアクセス	10000 clk
リモートアクセス	100 clk
通信	10 clk

表 3: その他のパラメータ

項目	値
プロセッサ数	16 台
システムの形状	2 進 4 段
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
U	0.1
V	0.01
同期頻度	1000 clk

リングの対象となる。各プロセスは 20 タイムスライス分の有効実行を過ごすと消滅される。各スケジューリング法は、全てのプロセスが終了するのに要したタイムスライス数で評価比較される。

性能について

図 4がセット A、セット B のシミュレーションの結果である。スケジュール法ごとに、終了までに要したタイムスライス数と有効実行時間 (プロセスがスケジュールされなかったプロセッサの分を含む) の占める割合を示した。

どちらのシミュレーションでも、 Algo3 の場合が最も早く実行が終了し、 Algo0 の場合が最も遅く実行が終了している。同様に有効実行率も Algo3 が一番良く、 Algo0 が最も性能が悪くなっている。

ここで、 4 つのスケジューリング法がどのように違うか考えてみる。 Algo0 はプロセスに割り当てる要素プロセッサをランダムに選択するために、プロセスのページの所在は全く考慮されない。また選ばれる要素プロセッサ同士の間には何も関係がないために、お互いの距離が非常に遠くなる傾向になる。これに対して Algo1 はプロセスのページの所在を考慮しない点では Algo0 と同じであるが、一続きのプロセッサたちを一つのプロセスに割り当てるために、 Algo0 よりは要素プロセッサ同士の距離が近いくなる。さらに、以前にプロセスがスケジュールされたのと同じ範囲のプロセッサ群が割り当てられる可能性もあり、そのような時にはプロセスはある程度自分のページをローカルに持っていることになる。実際セット A の Algo2 と Algo1 の差がないのは、プロセスの数が少なくなった時に同じような状況に対してほとんど同じスケジューリングを行なっていたためである。 Algo2 と Algo3 はスケジューリングの際に資源管理木の情報を利用して、なるべくページを多く持つ要素プロセッサを割り当てようとする。さらに資源管理木はシステムの相互結合網と同じ構造をしているので、トポロジー上の距離的に近いプロセッサたちを選んでプロセスに割り当てることが可能である。ただし Algo2 の場合にはそのような場所にスケジュールできなかったときにはプロセスのページの有無に関わらず必要な台数を割り当てようとするため、プロセスが自分のページを持つ可能性が低くなり、またプロセッサ同士の距離も Algo3 に比べて遠くなる。

このように Algo3、 Algo2、 Algo1、 Algo0 の順に、より多く必要なページを持つようにかつプロセッサ同士がなるべく近くなるようにスケジュールする可能性が高い。反対にこの逆順に、より多くのプロセッサを割り当てて遊ばせないスケジューリングをすることができる。

シミュレーションの結果を見ると、 Algo3、 Algo2、 Algo1、 Algo0 の順に高い性能を出している。つまり、ページの使用状況及びプロセス毎の占有実ページ数を管理し、スケジューリング時にプロセスには占有ページの多いプロセッサを割り当て、ネットワークのトポロジーを反映した資源管理木を利用してより距離的に近いプロセッサを選択するスケジューリング法方が好結果を出しているといえる。

表 4: シミュレーション結果

algorithm	set A	set B
0	427 28.82	401 50.81
1	214 57.55	363 54.46
2	213 59.00	303 61.01
3	183 71.30	270 70.66

1 段目：終了時間 (タイムスライス数)
2 段目：有効実行率

6 まとめ

ＮＵＭＡ型並列計算機やワークステーションクラスタ上に汎用並列 OS を構築するためのカーネルレベル・スケジューリングについて検討した。複数の並列アプリケーションの効果的な実行を妨げずにマルチユーザ / マルチジョブ環境を提供するために、システムのネットワークと同じ構造をした資源管理木を導入し、資源の利用状況を考慮したスケジューリング法を提案した。さらにシミュレーションにより、ページの使用状況及びプロセス毎の占有実ページ数を管理し、スケジューリング時にプロセスには占有ページの多いプロセッサを割り当て、ネットワークのトポロジーを反映した資源管理木を利用してより距離的に近

表 5: プロセス構成 – セット A

プロセス	要求プロセッサ数	空間サイズ [pages]
0	4	300
1	4	50
2	4	300
3	4	50
4	8	300
5	8	300
6	8	50
7	8	50
8	16	100
9	16	50
10	16	50
総数	96	11600

表 6: プロセス構成 – セット B

プロセス	要求プロセッサ数	空間サイズ [pages]
0	3	150
1	3	150
2	3	100
3	3	100
4	3	50
5	6	200
6	6	200
7	6	200
8	6	200
9	6	200
10	6	200
11	13	50
12	13	50
13	13	50
14	16	50
総数	106	11600

いプロセッサを選択するスケジューリング方法が好結果を出すことを示した。

今後は、プロセスのマイグレイションや、より多くのスケジューリング制約を扱うスケジュール法の研究 / 開発及び、実機への実装が課題となる。

7 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*, 25(5):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, pages 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*, pages 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. ＣＰＳＹ *89-37*, pages 37–42, August 1989.