システムソフトウェアと
オペレーティング・システム 61−15
（１９９３．８．１８）

ＮＵＭＡマルチプロセッサにおける２レベルスケジューリング

藤木 亮介† 甲斐 久淳† 福田 晃†‡

†: 九州大学工学部
‡: 奈良先端科学技術大学院大学

NUMA(Non-Uniform Memory Access) マルチプロセッサは，大規模システムの性能向上に有望とされ，種々の並列プロセスを実行するために用いられている．しかし，スケジューリングアルゴリズムがその性能に大きく影響を与えるにもかかわらず，これまであまり研究が行なわれていない．そこで，大規模マルチプロセッサに対し有効であると思われる空間分割スケジューリングの一種である２レベルスケジューリングについての研究を行なう．

本稿では，２レベルスケジューリングの構造，およびインタフェースを述べるとともに，スケジューリングアルゴリズムにおける選択肢として，フリープロセッサの処理に関しては，１）実行プロセス優先方式，と２）待ちプロセス優先方式，アイドルプロセッサの処理に関して，１）アイドルプロセッサ保持方式，と２）アイドルプロセッサ解放方式を挙げる．これらの選択肢は，シミュレーションによって評価される．

Two-level Processor Scheduling for NUMA Multiprocessors

Ryousuke Fujiki† Hisa-aki Kai† Akira Fukuda†‡

†: Faculty of Engineering, Kyushu University
‡: Graduate School of Information Science
Advanced Institute of Science and Technology, Nara

fujiki@csce.kyushu-u.ac.jp

Non-Uniform Memory Access multiprocessors have potential gains for achieving high performance. These machines are frequently used as computation servers with multiple parallel processes (we use this word instead of jobs) executing at the same time. Although, in such environments, the effciency of a parallel process can be significantly affected by the processor scheduling strategy, there are few studies on processor scheduling to date. Two-level scheduling strategy, which is a kind of space-multiplexing scheduling strategy, is a promising one for large-scale multiprocessors.

This paper describes the structure and interface of the two-level scheduling strategy. We also propose basic alternatives in this strategy: as alternatives of handling free processors, 1)Running-Process-First scheme and 2)Waiting-Process-First scheme; as alternatives of handling idle processors, 1)Idle-Processor-Hold scheme and 2)Idle-Processor-Release scheme. These alternatives are examined through simulation experiments.

1 はじめに

クラスタ型NUMA(Non-Uniform Memory Access)マルチプロセッサにおけるスケジューリングアルゴリズムの提案，及びシミュレーションによる評価を行なう．

共有メモリ型マルチプロセッサにおけるスケジューリングでは多くの研究が行われているが[1, 2, 3, 4]，その多くはUMA(Uniform Memory Access)マルチプロセッサを対象としたものであり，NUMAマルチプロセッサを対象とした研究はほとんど行われていない[5, 6]．本稿では，NUMAマルチプロセッサを対象とし，大規模マルチプロセッサにおけるスケジューリング方式として有望な空間分割スケジューリングの一種である2レベルスケジューリングにおけるいくつかのアルゴリズムをシミュレーションにより評価する．本稿では，プログラムまたはジョブをプロセス，プロセス内のアクティビティをスレッドと呼ぶことにする．

本稿の構成は以下のとおりである．2章で，2レベルスケジューリングの構成，およびインタフェースについて述べる．3章で，NUMAマルチプロセッサを対象としたスケジューリングアルゴリズムについて述べ，4章で，それらのアルゴリズムについてシミュレーションにより評価する．5章で結果と考察について述べ，6章でまとめを行なう．

2 2レベルスケジューリング

2.1 スケジューラの構造

2レベルスケジューラは，図1に示すようにグローバルスケジューラとプライベートスケジューラによって構成される．

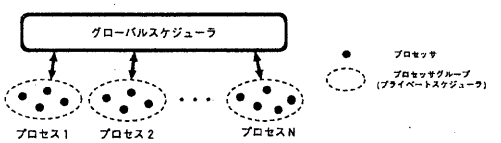

図 1: 2レベルスケジューリング

2.1.1 グローバルスケジューラ

グローバルスケジューラは，プロセッサグループとプロセッサプールの管理を行なう．1つのプロセスに対し，1つのプロセッサグループを与える．どのプロセッサグループにも属していないプロセッサ（フリープロセッサ）はプロセッサプールに置かれ，プロセスからプロセッサの要求があった時に，グループに割り当てられる．

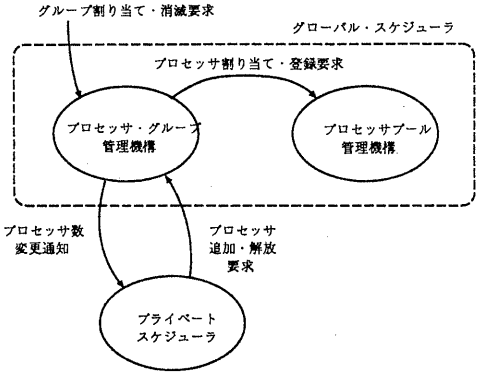

図 2: インタフェース

2.1.2 プライベートスケジューラ

プライベートスケジューラは，プロセッサグループ内でのスレッドに対するプロセッサ割り当てを行なう．プライベートスケジューラは，各プロセッサグループ内に存在し，各々がレディスレッドキューを持っている．

2.2 インタフェース

各管理機構と提供するインタフェースを図2に示す．

2.2.1 プロセッサグループ管理機構

プロセッサグループ管理機構は，主に以下のインタフェースを提供する．

- グループ割り当て要求

  プロセスの生成の際，新たにプロセッサグループを生成するためのインタフェース．要求プロセッサ数，プロセスIDなどを引数とし，グループID，割り当てプロセッサリスト等を返す．

- グループ消滅要求

  プロセスの終了により，プロセッサグループを消滅させるためのインタフェース．グループIDを引数とし，成功，または失敗を返す．

- プロセッサ追加要求

  プロセッサ追加を要求するためのインタフェース．グループID，追加要求プロセッサ数を引数とし，追加プロセッサリストを返す．

- プロセッサ解放要求

  プロセッサグループからプロセッサを削除するためのインタフェース．解放要求プロセッサリストなどを引数とし，解放プロセッサリスト等を返す．プライベートスケジューラが，グループ内にプロセッサが余ったと判断した時に使用される．

2.2.2 プロセッサプール管理機構

プロセッサプール管理機構は，フリープロセッサの処理に関して，以下のインタフェースを外部に提供する．

- プロセッサ割り当て要求

  プロセッサプールに属するフリープロセッサを割り当ててもらうためのインタフェース．割り当て要求プロセッサ数等を引数とし，割り当てプロセッサリスト等を返す．

- プロセッサ登録要求

  フリープロセッサができた時に，プロセッサプールに登録するためのインタフェース．登録プロセッサリスト，グループID等を引数とし，成功，または失敗を返す．

2.2.3 プライベートスケジューラ

プライベートスケジューラは，主に以下のインタフェースを提供する．

- プロセッサ数変更通知

  グループ内プロセッサ数が変更されたことを通知してもらうためのインタフェース．変更プロセッサリストとを引数とし，成功，または失敗を返す．プロセッサグループ管理機構が強制的にグループ内プロセッサ数を変更したい場合などに使用される．

3 スケジューリングアルゴリズム

本稿で対象とする2段階クラスタ型NUMAマルチプロセッサ（図3参照）では，プロセッサがメモリにアクセスする時のコストが，キャッシュ，ローカルメモリ，リモートメモリの3段階になる．NUMAマルチプロセッサ上でのスケジューリングでは，リモートメモリアクセスコストが非常に高いため，それを減らすことが重要である．ここではプロセッサをグループ化（プロセッサグループ）してプロセッサグループ単位にプロセスを割り当てる「2レベルスケジューリング」

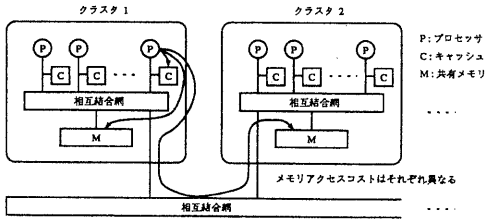

図 3: 2段階クラスタ型NUMAマルチプロセッサ

（図1参照）を基本として，大きく分けて2つのアルゴリズムを提案する．1つは，プロセスのプロセッサ要求を満たすようできるだけ多くのプロセッサを与える「プロセッサ数優先アルゴリズム」で，もう1つは，リモートメモリアクセス回数を削減するため，プロセッサを要求したプロセスのデータ等がロードされているクラスタの中のプロセッサのみを，要求プロセスに対して与える「クラスタ優先アルゴリズム」である．

3.1 プロセッサ数優先アルゴリズム

プロセスの生成やプロセス内のスレッドの生成などによってプロセッサが要求された場合，フリープロセッサがあれば要求数を満たすように割り当てるアルゴリズムである．フリープロセッサが複数あった場合，どのフリープロセッサを割り当てるかが問題となる．キャッシュ，ローカルメモリ，リモートメモリのアクセス距離を考慮して割り当てる．

具体的なアクションを以下に述べる．

(1) プロセス生成時

プロセスはまず，プロセッサグループ管理機構にグループ生成要求を出すと同時に，プロセッサを1つ要求する．プロセッサグループ管理機構はプロセッサプール管理機構に対しプロセッサの割り当てを要求するが，この時プロセッサグループ管理機構は以下の順序でフリープロセッサの割り当てを試みる．

1. 要求プロセスのデータ等がロードされているクラスタに属するフリープロセッサ
2. プロセスの終了によりフリーとなったプロセッサ，すなわちプロセッサグループの消滅によりフリーとなったプロセッサ
3. その他のフリープロセッサ

プロセッサが割り当てられない場合，プロセスはグループ生成待ちとなる．

(2) プロセス終了時

プロセスの終了により解放されたプロセッサは，プロセッサを要求中のプロセスに割り当てられるが，その方式として以下の２つの選択肢を挙げる．

(a) 実行プロセス優先方式

システム内のプロセス数を増やさず，実行中のプロセスに，より多くのプロセッサを与えることによって，スレッド間同期にかかる時間を短縮し，キャッシュヒット率を高くする．

割り当てる順番は不足数の多いプロセスからとし，解放プロセッサ数が不足数よりも少ない時は，無条件に全ての解放プロセッサをそのプロセスに割り当てるが，解放プロセッサ数が不足数よりも多い時は，そのプロセスが属するクラスタ内のプロセッサを優先的に割り当てる．

実行中のプロセスの全要求を満たした後でもプロセッサが余っていれば，グループ生成を待つ待ちプロセスに割り当てる．

(b) 待ちプロセス優先方式

グループ生成待ちのプロセスに，優先的にプロセッサを与える．実行プロセス優先方式に比べ，公平なプロセッサ割り当てができ，負荷の軽いプロセスに対し有効である．割り当てる順番は，FIFO，すなわち到着順とするが，解放プロセッサ数が要求数よりも多い時は，実行プロセス優先方式と同様，そのプロセスの属するクラスタ内のプロセッサを優先的に割り当てる．また，待ちプロセスの全要求を満たした後でもプロセッサが余った場合，余ったプロセッサを実行中のプロセスに割り当てる．この時も，不足数の多いプロセスから割り当てる．

(3) プロセス実行時

グループ内の状態によりプロセッサの追加・解放要求を行なう動的プロセスに対し，スケジューラは以下のアクションをとる．

プロセッサ不足によるプロセッサ要求の場合，プロセッサプール管理機構は，以下の順序で割当を試みる．

1. フリーとなる前に要求プロセッサグループ（要求プロセス）に属していたフリープロセッサ（キャッシュの利用）
2. 要求プロセスのデータ等がロードされているクラスタに属するフリープロセッサ
3. プロセスの終了によりフリーとなったプロセッサ，すなわちプロセッサグループの消滅によりフリーとなったプロセッサ
4. その他のフリープロセッサ

また，グループ内にアイドルプロセッサができた場合は，以下のどちらかの選択肢をとる．

(a) アイドルプロセッサ解放方式

プロセッサ利用率を上げるため，グループから解放し，他の要求プロセスへと割り当てる．ただし，キャッシュのヒット率は悪くなる．

プロセッサを要求しているプロセスが無い場合は，アイドルプロセッサの解放を行なわない．

(b) アイドルプロセッサ保持方式

アイドルになっても，また同じグループに再度割り当てられる可能性もあるため，プロセスの終了までそのまま保持する．キャッシュヒット率は高くなるが，使用されないままグループ内に置かれることもあり，プロセッサ使用率は低くなる．

3.2 クラスタ優先アルゴリズム

クラスタ優先アルゴリズムでは，プロセスが１つのクラスタのメモリ内におかれることを前提として，そのクラスタ内のプロセッサのみを割り当てるものである．ただし，局所性を重んじるため，システム全体としてはプロセッサが余っているのに割り当てられないこともあり，負荷の偏りがおきやすい．このアルゴリズムは，リモートメモリアクセスコストが非常に高い時に有効である．

(1) プロセス生成時

要求プロセスの属するクラスタの中で，まず，キャッシュの情報が使われる可能性のないフリープロセッサを割り当てる．次に，その他のフリープロセッサを与える．たとえ他のクラスタ内にフリープロセッサがあっても，自分のクラスタ内にフリープロセッサがなければ，グループ生成を待たされる．

(2) プロセス終了時

解放されたプロセッサは，プロセッサ優先アルゴリズムと同様，以下の選択肢のどちらかをとる．

(a) 実行プロセス優先方式

(b) 待ちプロセス優先方式

この時も，同じクラスタ内のプロセスが対象となる．

(3) プロセス実行時

プロセッサ割り当て要求の場合，キャッシュヒットを最優先とした後，生成時と同様の順序で割当を試みる．この時も，別のクラスタのプロセッサは割り当てない．

また，グループ内にアイドルプロセッサができた場合は，プロセッサ数優先アルゴリズムと同様に，以下のどちらかの選択肢をとる．

(a) アイドルプロセッサ解放方式

(b) アイドルプロセッサ保持方式

4 シミュレーションモデル

4.1 アーキテクチャモデル

クラスタ内のプロセッサ数（クラスタサイズ）が8で，8クラスタ，計64台のNUMAマルチプロセッサとする．各プロセッサにキャッシュ，クラスタに1つのローカルメモリがある．

また，1つのプロセスのプログラムやデータ等は1つのクラスタ内の共有メモリに入るものとし，プロセスの到着順に，各クラスタに順番におかれる．

4.2 プロセスモデル

1つのプロセスはfork-joinタイプのスレッド生成／消滅を繰り返す（図4）．fork-joinは2回とし，サスペンドジョインとする．すなわち，ジョイン時点でまだ他のスレッドがジョイン時点に達していない時は，当該スレッドはサスペンドされる．最後のスレッドがジョイン時点に達したら，サスペンドされているスレッドを起こす．

各フレーズの実行時間は，$t_b = t_m = 13$，$t_e = 29$ を固定とし，forkで生成されるスレッド数，および実行時間等は以下のように変化させる．

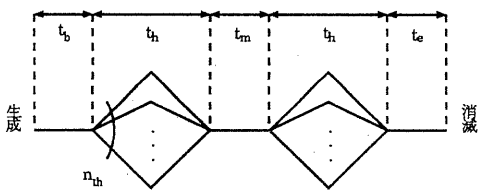

図 4: プロセスモデル

○タイプA

- t_hは(33,99)の一様分布とする．
- n_{th}は6,12,24の3通りを評価するが，$n_{th} = 6$は(2,10)，$n_{th} = 12$は(8,16)，$n_{th} = 24$は(20,28)の一様分布をとる．

○タイプB

- $n_{th} \times t_h$を一定とする．つまり，仕事量を一定とし，粒度（Fork数）を変化させる．ここでは，$(n_{th}, t_h) = (2,198),(6,66),(12,33),(24,16.5)$とする．

4.3 オーバヘッド

キャッシュ，ローカルメモリ，リモートメモリアクセスのオーバヘッドを考慮している．メモリアクセスの際にかかるコストの比率は，キャッシュヒットを1とした場合，ローカルメモリへのアクセスをO_{local}倍，リモートメモリへのアクセスは，どのクラスタであってもO_{remote}倍として，実行時間を伸長させるものとする（以前実行したスレッドが属するグループIDと，割り当てられたプロセスのグループIDが同じ時，キャッシュヒットとする）．O_{local}，O_{remote}はそれぞれ$(O_{local}, O_{remote}) = (1.5,7.5),(1.2,2.4)$とする．

また，各管理機構は排他制御され，グループ変更によりグローバルスケジューラが動かされる時，管理機構のアクション時間としてオーバーヘッド2が加えられる．

4.4 アルゴリズムにおける選択肢

本稿では，2章で述べた2つのアルゴリズムを比較するが，その前に選択肢について整理する．

(1) フリープロセッサ処理方式

プロセッサの割り当てを要求しているプロセスのタイプとしては，プロセスを生成したが1台もプロセッサが割り当てられず（すなわち，プロセッサグループを割り当てられていない），その割り当てを待っているプロセス（待ちプロセス）と，プロセッサが割り当てられて実行しているときにさらにプロセッサを要求したプロセス（実行プロセス），の2つがある．2つのどちらを優先するかによって以下の選択肢がある．

(a) 待ちプロセス優先方式：待ちプロセスに優先して割り当てる．

(b) 実行プロセス優先方式：実行プロセスに優先して割り当てる．

(2) アイドルプロセッサ処理方式

以下の2つの選択肢を挙げる．

(a) 解放方式：アイドルプロセッサを当該プロセッサグループから解放する．そのプロセッサはフリープロセッサとなる．

(b) 保持方式：当該プロセッサグループに割り当てられたままである．

5 シミュレーション結果

プロセッサ数優先アルゴリズムとクラスタ優先アルゴリズムにおいてスレッドの数 n_{th} が，$n_{th} = 6, 24$ の場合について，4.4節で述べた選択肢のシミュレーション結果を示す．下記に示す4通りの組み合わせがある．

方式A：(待ちプロセス優先，解放方式)

方式B：(待ちプロセス優先，保持方式)

方式C：(実行プロセス優先，解放方式)

方式D：(実行プロセス優先，保持方式)

図5〜8に平均応答時間を示す．図5,7がプロセッサ数優先アルゴリズム，図6,8がクラスタ優先アルゴリズムのものである．横軸はプロセスの生成間隔，縦軸は応答時間である．パラメータは，fork-joinは2回，スケジューリングオーバーヘッド2，メモリアクセスオーバーヘッド $O_{local} = 1.5$，$O_{remote} = 7.5$ である．このパラメータでは，クラスタ優先アルゴリズムの方が優れている．方式を比較した場合，アイドルプロセッサの処理に関しては，保持方式が優れているといえる．これは，リモートメモリアクセスコストが非常に高いため，プロセッサの切替回数を減らし，キャッシュミスヒットを減らした方が，プロセッサの利用率を上げるよりも効果的であるからである．

次にメモリアクセスオーバーヘッド比を，$O_{local} = 1.2$，$O_{remote} = 2.4$ とした場合の応答時間を，図9（プロセッサ数優先アルゴリズム），および図10（クラスタ優先アルゴリズム）に示す．メモリアクセスコスト比が1:1.5:7.5のときに比べ，負荷の軽い所で，プロセッサ数優先アルゴリズムが優れていることがわかる．つまり，必要処理時間に比べ，メモリアクセスのオーバーヘッドを含めた全実行時間が比較的短いプロセスに対しては，プロセッサ数優先アルゴリズムが有効であるといえる．しかし，負荷が重くなるにつれ，1グループあたりのプロセッサ数は減少し，プロセッサ数優先アルゴリズムの利点が生かされなくなり，応答時間が悪くなる．

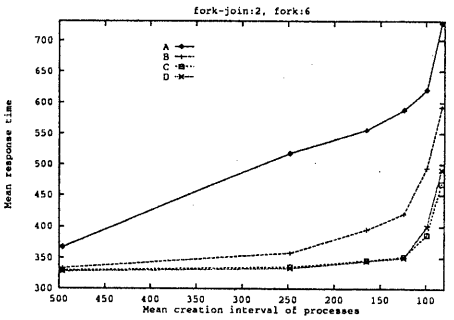

図5: 平均応答時間（タイプA：$n_{th} = 6$：プロセッサ数優先）

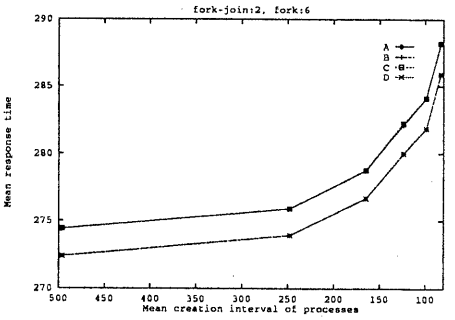

図6: 平均応答時間（タイプA：$n_{th} = 6$：クラスタ優先）

図11，12にプロセスタイプBについての結果を示す．スケジューリングオーバーヘッド2，メモリアクセスオーバーヘッド $O_{local} = 1.5$，$O_{remote} = 7.5$ である．横軸はfork-join時におけるスレッド数，縦軸に応答時間である．図中のaはプロセスの生成間隔を示す．a = 124は中負荷，a = 496は軽負荷となる場合である．図では，fork-join回数を2とし，fork-join時におけるスレッド数とスレッドの実行時間の積が一定となるようにしている．つまり，仕事量が一定で，forkされるスレッド数（図中の横軸）が多くなるほど，forkされたスレッドの実行時間が短くなる．クラスタ優先アルゴリズム（図12参照）の場合やプロセッサ数優先アルゴリズム（図11参照）の軽負荷の場合は方式に差がないが，プロセッサ数優先アルゴリズムの中負荷の場合，fork数24のときは，図7の結果とは逆にアイドルプロセッサ解放方式の方が優れている．これは，リモートメモリアクセス率（図13参照）は増えるが，1本のスレッドの必要処理時間が短いため，

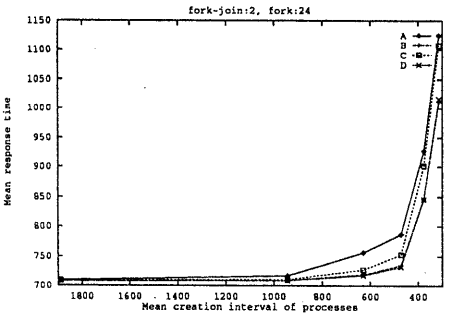

図 7: 平均応答時間（タイプA：$n_{th} = 24$：プロセッサ数優先）

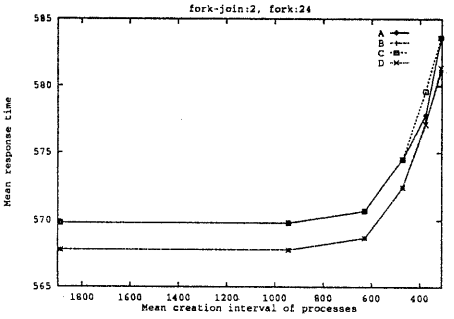

図 8: 平均応答時間（タイプA：$n_{th} = 24$：クラスタ優先）

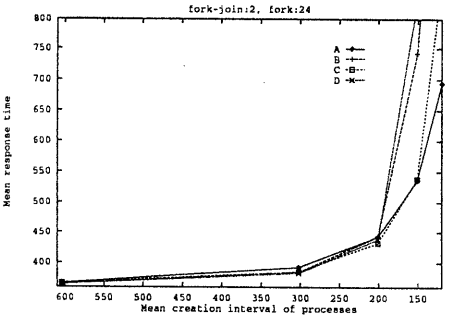

図 9: 平均応答時間（タイプA：$n_{th} = 24$：プロセッサ数優先）

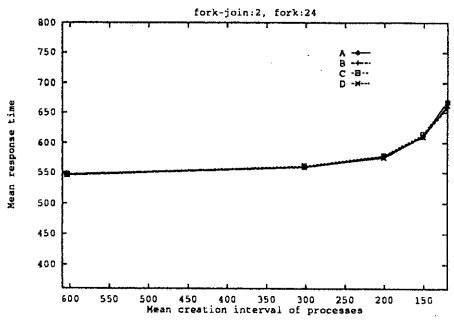

図 10: 平均応答時間（タイプA：$n_{th} = 24$：クラスタ優先）

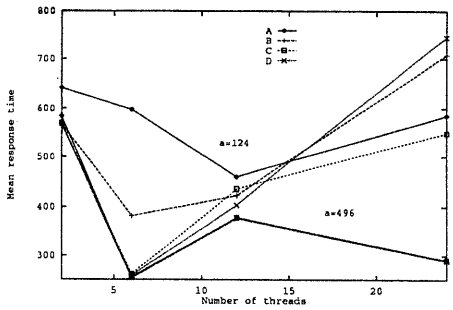

図 11: 平均応答時間（タイプB：プロセッサ数優先：プロセス生成間隔 a=496,124）

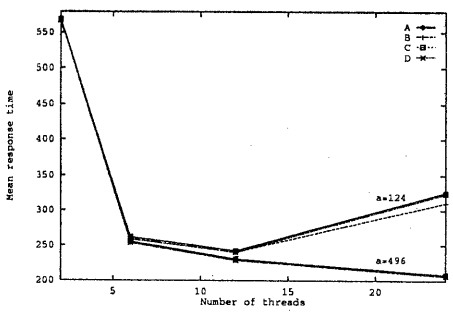

図 12: 平均応答時間（タイプB：クラスタ優先：プロセス生成間隔 a=496,124）

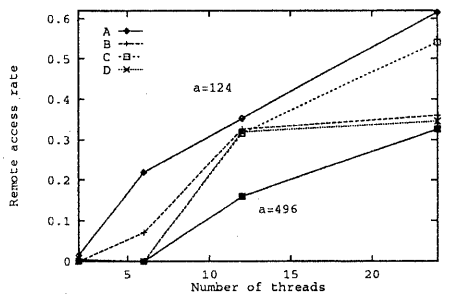

図 13: リモートメモリアクセス率（図 11）

プロセッサを解放して利用率を上げてやると，同期にかかる時間が短くなり，結果として，1つのプロセスが，複数のプロセッサを占有している時間も短くなるからである．

プロセッサ数優先アルゴリズムの中負荷の場合について簡単に考察すると，スレッド数がクラスタサイズ（＝8）程度になるように粒度を設定した方がよいことが図 11より分かる．また，スレッド数がクラスタサイズ以下については以下のことが言える．

- アイドルプロセッサ処理方式の違いよりも，フリープロセッサ処理方式の違いの方がシステムの性能に大きな影響を与える．
- ほとんどの場合，実行プロセス優先方式の方が待ちプロセス優先方式よりも優れている．

6 おわりに

クラスタ型 NUMA マルチプロセッサを対象としたスケジューリングアルゴリズムの提案，および評価を行なった．規模拡大への対応と不均一メモリアクセスを考慮して，4つのアルゴリズムの比較・検討を行なった．今後は，様々なプロセスパターンについてのアルゴリズムの比較や，さらにプロセッサ台数を増やした場合に，クラスタサイズがシステムに与える影響についても評価を行なう予定である．

参考文献

[1] Tucker, A. and Gupta, A.: "Process Control and Scheduling Issues for Multiprogrammed Shared Memory Multiprocessors," Proc. the 12th ACM Symp. Operating Systems Principles, pp.159-166(1989).

[2] Zahorjan, J. and McCann, C.: "Processor Scheduling in Shared Memory Multiprocessors," Proc. the 1990 ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems, pp.214-225(1990).

[3] Gupta, A., Tucker, A., and Urushibara, S.: "The Impact of Operating System Scheduling and Synchronization Methods on the Performance of Parallel Applications," Proc. the 1991 ACM SIGMETRICS Conf. on Measurement and Modelling of Computer Systems, pp.120-132(1991).

[4] Kai, H., Fujiki, R., and Fukuda, A.: "A General Interface for Two-level Processor Scheduler on Multiprogrammed Multiprocessors and Its Performance;" Joint Conference on Software Engineering '93（発表予定）

[5] Zhou, S. and Brecht, T.: "Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors," Proc. the 1991 ACM SIGMETRICS Conf. Measurement and Modelling of Computer Systems, pp.133-142(1991).

[6] Fukuda, A., Fujiki, R., and Kai, H.: "Two-level Processor Scheduling for Multiprogrammed NUMA Multiprocessors," COMPSAC'93（発表予定）