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

信国　陽二郎[†]　松本　尚[†]　平木　敬[†]

本稿では，分散並列環境における OS レベルの資源管理方式を実験可能にするシミュレーションモデルについて議論する．OS レベルの資源管理方式をシミュレーションにより実験可能とするには，資源の消費状況を把握できること及び様々な資源操作機能が組み込まれていることが重要である．そのためのモデルは特に，並列アプリケーションにとって重要な共有データのアクセスが組み込まれていることと，OS レベルの資源管理方式の性能差を導くために計算機に内在する種々の物理資源制約ボトルネックを反映する仕組みを持つことが重要である．

Simulation Models for Performance Estimation of Parallel OS

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

In the paper, we discuss on simulation model for the performance estimation of resource management mechanisms of parallel OS. Enabling the performance estimation requires grasping conditions of resource consumption and various resource handling function to be integrated. For the purpose, simulation models must support shared data access mechanisms, which are important for parallel applications, and be designed to reflect various hardware resource limitatons for importing the performance differences of OS level resource management mechanisms.

1. はじめに

新しく設計した OS 機能のシミュレーションは，その OS を実装する前にその機能の性能評価及び実行の振舞いが解析できるという点で，非常に有効である．また，ある OS 機能の様々なバリエーションを実装することがしばしば困難であるのに対して，それらをシミュレーションに組み込むことは比較的簡単に行なえる可能性がある．さらに，パラメータの設定により様々なハードウェア構成の計算機をシミュレーションすることが可能である．特に経済的な理由や手持ちのハードウェア資源の制限から，実際にはその OS を実装することが不可能なシステム構成について実験及び解析することが可能になる．

シミュレーションが真に有効であるためには，そのシミュレーションの正当性が保証されていること，評価したい機能の性能評価及び解析が行なえること，そしてシミュレーション自体の実行が現実的に可能であることが重要である．並列 OS のシミュレーションの場合，プロセッサやメモリ，ネットワークの詳細なモデル化は，シミュレータの製作に多大な時間と労力を要求するばかりか，シミュレーションの実行自体が現実的に不可能である．そもそもそれほどの高機能なシミュレータを作るぐらいなら，本物の OS を実装してしまえば良い．そこでモデル化することにより対象を単純化し，簡略化したシミュレーションを可能とする必要がある．その場合，どの程度詳細な評価が必要なのかを検討し，それにあったシミュレーションを行なうには，計算機の振舞いの各部分をどのようにモデル化するべきかを考えなければならない．

分散並列環境における OS レベルの資源管理方式をシミュレーションにより実験可能とするには，資源の消費状況を把握できること及び様々な資源操作機能が組み込まれていることが重要である．そのためのモデルは特に，並列アプリケーションにとって重要な共有データへのアクセス方式や同期方式が反映されていることと，OS レベルの資源管理方式の性能差を導くために計算機に内在する種々の実資源制約を反映する仕組みを持つことが重要である．

本稿では，分散並列環境における OS レベルの資源管理方式を実験可能にし，性能評価を行なうことのできるシミュレーションモデルについて議論する．モデル化をした結果問題になる現象については，適時 SSS–CORE[1] のスケジューリング法のシミュレーション[2] ☆の結果を例示する（サンプル 1–4）．

SSS–CORE1) は，現在我々が開発中の NUMA 型

† 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

☆ 幾つかの部分はこれより後に改善されている．

並列計算機をターゲットとした汎用並列 OS である．これは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．そのためには，並列プロセスの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．そこで 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．さらにカーネルによって管理される資源情報は，ユーザレベルからの実行時方針のヒントとして利用が可能であり，アプリケーションは割り当てられた実資源を自由に再スケジューリングすることができる 3)4)．したがって SSS–CORE のシミュレーションでは，アプリケーション毎のメモリ使用量を管理する必要がある．

なお付録で，サンプルシミュレーションに利用したプロセスの組（**表5**–**表7**）及び比較したスケジューリング法について説明する．

2. モデル化の方針

簡略化したシミュレーションを可能とするには，モデル化することにより構成要素を単純化する必要がある．その場合，どの程度詳細な評価を行なう必要があるのかを検討し，計算機の振舞いの各部分をどのようにモデル化するべきかを考えなければならない．

シミュレーションにより並列 OS の性能評価を行なうことの目的は，様々なアプリケーション・プログラムを実行したとき，ハードウェア / ソフトウェア環境の違いによりシステム全体の性能がどのように変化するかを論じることにある．そこでシミュレーション・モデルについて考える上で特に次の二点が重要になる．

第一に，実行されるアプリケーションの組の振舞いである．主要なものとしては，個々のアプリケーションの行動を決定するメモリの参照パターンと，各アプリケーションの並列度 / 同期の変化などの時間軸に沿った実行パターンがある．これは，シミュレーション時のシステム全体での負荷に関係する．とりわけ，共有データへのアクセス方式は並列アプリケーションの性能に大きく影響する．そのような方式には例えばハードウェア実装された分散共有メモリ，ソフトウェアでサポートされた分散共有メモリ，またはメッセージパッシング方式がある．これら方式の違いとは即ち共有データの正しい値の管理 / 獲得方法の違いであり，論理的には参照 / 操作するメモリ位置とその操作を行なうタイミングの違いと捉えられる．従って，命令レベルの解釈実行を行なわない場合，対象とする並列計算機における共有データへのアクセス方式は，アプリケーションの共有領域へのメモリ参照パターン及び，通信オーヴァヘッドのコスト設定で抽象化できる．メモリ参照列はまた，どのような意味を持ったアプリケーションをモデル化しているのかとも関係がある．従って，アプリケーションにどのようなメモリ参照パターンを発生させ，それにはどのような機構とパラメータが必要であるかが重要となる．

第二に，並列計算機上でアプリケーションの実行の性能差を生み出す原因は何であるかである． OS の資源管理方式の性能差は，シミュレーション実行時に発生する計算機に内在する種々のボトルネックが原因である．ボトルネックは与えられた物理資源量に制限があるから生じる．特に影響が大きいのは，プロセッサ数及び実メモリ量の制限とネットワークのバンド幅である．

並列計算機のプロセッサ数は有限である．この制約は，アプリケーションが一度に利用できるプロセッサ数や， OS の資源管理の処理能力に影響する．

また実メモリは容量が限られているために，メモリ上でリプレースを起こす原因となる．余計なリプレースは実行効率を大幅に低下させる原因になる．

ネットワークの場合にはバンド幅が限られていることでボトルネックとなり得る．アプリケーションは同期やリモートメモリアクセス，通信によりネットワーク・トランザクションを発生する．例えばネットワークが飽和している状況での同期は，同期待ちプロセッサを余計に遊ばせることになる．一つの同期での余計な待ち時間は後続の同期にまで伝搬するため，実行効率への影響は大きい．また OS の資源管理方式が異なればネットワーク・トランザクションの発生パターンも異なる．このように，ネットワーク・トランザクションの量が性能差に現れると考えられる．

一般にメモリの使用量が少なく済み，ネットワーク・トランザクション数を少なく抑えられるような資源管理方式が性能が良いと言える．従って以上のような物理資源量の制約を如何に表現するかが，モデル化の上で重要になる．

注意すべきことは，メモリ参照列と資源量の制約の密接な関係である．多くのメモリ断片を参照すればそれだけ通信量やメモリリプレース量を増加させる．メモリリプレースの増加は通信量を増加させる．そうした相乗性が資源量の制約を受けてシステム全体の効率に大きく影響する．

2.1 性能の評価法

並列 OS の性能は，システム全体での実行効率で評価できる．実行効率とは，各プロセッサが有益な仕事をした割合と考えられる．プロセッサの “有益” な仕事とは，メモリアクセスしない一般の命令実行や，プロセッサ・キャッシュでのヒットを含めたローカルメモリ・アクセスなどが考えられる．それ以外の “無益” な仕事としては，ページフォールトが原因のページ待ちや同期待ちが挙げられる．前者を有効実行，後者を非有効実行と呼ぶことにすると，実行効率とは全時間に占める有効実行時間で定義できる．当然，有効実行率が高いほど性能

が高いといえる．クロックベースにシミュレーションを行なう場合の有効実行率は，シミュレーションした全実行クロック数に対する，総有効実行クロック数となる．ただし，対象とするOSのスケジューラの方針でスレッドを割り当てられずにアイドルとなるプロセッサの分は，有効実行に含めるべきである．

モデルを詳細にすればそれだけ現実に近い結果が得られる．反対にモデルを簡略化すれば，シミュレーションの実行時間及び開発時間を短縮する可能性が高い．次節以降では，どのようなモデル化でどの程度詳細なシミュレーションが可能になるか議論する．ただし本稿では，プロセッサ毎の実行単位をスレッドと呼び，相互に関連のあるスレッドの集合を（並列）プロセスと呼ぶことにする．

3. 何をモデルに取り込むか

3.1 メモリ参照列モデル

実際のアプリケーションのメモリ参照を模倣するには，トレースを利用する方法がある．トレースの一番の利点は，ある条件下では実際の実行に近いことである．また，プロセスの並列度の変化や同期の情報をトレースの中に埋め込めば，それらを別にモデル化する必要がなくなる．しかしトレースは，そのトレースが採取されたときの環境や条件に依存した動的な要因を含んだ情報であり，同期などはその環境や条件に固有なものといえる．つまり，対象とする条件下でのトレースの採取が困難だからといって，異なった条件下でのトレースを流用しようというのは虫がよすぎるのである．そもそも対象とする環境向けのトレースは，それと同じ環境上で採取されるべきであり，言わば“ニワトリと卵”問題なのである．

トレースを利用しない場合，何らかの方法を用いて，メモリへの参照列を計算しなければならない．ポピュラーな方法に参照単位毎のアクセスを確率分布に従って計算する方法がある．メモリへのアクセス単位で，与えられた分布関数にしたがって確率的に参照位置を決定するのである．この方法の利点として，パラメータの数が少なく容易に実験ができることが挙げられる．しかしそれ以上に多くの欠点を指摘することができる．

一つ目は，この方法では参照の順序，言い替えればプロセスにとって意味のあるアクセス単位の間の関連性や，参照フェーズの推移を表現できない．そこでせめて参照の空間局所性を考慮できるように，参照確率の高い領域を持つような分布関数を利用することが考えられる．しかし，少々のパラメータで自由な分布曲線を作ることは困難であり，また複雑な曲線はイメージを思い描きにくいので好ましくない．当然，参照確率の高い領域と低い領域の調節は難しい．確率が低過ぎる領域は全く参照されないし，それを解消するために微小確率の一様分布を重ね合わせても，シミュレーションする時間スケールとプロセスの空間サイズの兼ね合い次第では，短時間に全空間がアクセスされてしまう（サンプル1）．まして，一様分布などもってのほかである5)．

表1 サンプル結果1 (有効実行率)
Table 1 Sample results 1

	V		
U	0.01	0.1	0.5
0.0	96.74	93.20	91.82
0.01	84.06	81.24	81.01
0.1	79.16	80.98	82.36

サンプル1 確率分布では失敗する例．**表1**は，平均をずらした三つの正規分布を重ね，一様分布で底上げした分布でアクセスパターンを計算しシミュレーションした結果である．Uが一様アクセスの確率，Vが正規分布の分散である．4段2進木構成のシステムで，アルゴリズム3を使用した．U=0.0ではV=0.001,0.1の極端に狭いピークのみにアクセスが集中したため，メモリのリプレースさえ起きなかった．そこでU=0.01,0.1で底上げしてみると，全空間をアクセスしてしまいVの違いは性能差に現れない．

さらに，分布をプロセス毎に指定する方法をとると，そのプロセスの全スレッドで分布のピークの位置に当たる領域を共有することになる．頻繁にアクセスする領域を全スレッドで共有するわけだから，あまりにも現実離れしてしまう．スレッド毎に分布関数をずらして対処する方法もあるが，関数同志の重なり具合の調節はどのみち容易ではない．いわんや意味づけについてをや．

これらの制約をもう少し緩和するために，分布関数の代りに参照頻度表を使って離散的に表現する方法がある．参照単位毎に表のエントリに参照頻度を書いておく．そして参照位置は，全エントリでの総頻度に対する各エントリの頻度から確率的に選択される．この手法であれば参照確率の高低も含めて，分布関数よりは自由に参照頻度を指定できる．またスレッド毎に頻度表を持たせれば，メモリの共有の具合（共有するスレッドの組や共有する量）を調節することも可能である．さらに頻度表では，表のエントリ毎に対応する領域の属性を，必要に応じて指定することができる．属性とは例えば，リード/ライト比，一貫性制御プロトコルである．しかしその他の点では，確率分布と同じ制約を抱えている．

3.2 物理資源制約モデル

3.2.1 プロセッサのモデル化

プロセッサの場合に考えられるパラメータには，総数及びクラスタ当たりのプロセッサ数がある．ネットワークの構造が決定されれば直接指定される必要はない．

モデル化された他の部分の仕組みに合わせて，要求される動作を組み込まなければならない．

3.2.2 メモリのモデル化

並列計算機を汎用に利用するには，メモリの物理量を越える仮想メモリシステムに対応しなければならない．実際にはメモリ量は有限なため，メモリのリプレースが起こる．複数のプロセスに資源を割り当てながら如何に

リプレース量を抑えるかは，OS にとって大きな問題である．従って OS の資源管理方式のシミュレーションを行なう場合，メモリの容量に制限を設けることが重要である．モデルとしては何らかのメモリリプレース方式が必要になる．

性能への影響が大きい共有データへのアクセス方式のモデル化も大切である．何らかの共有データへのアクセス方式を想定し，方式のモデル化及びその方式に見合ったパラメータ設定が要求される．

またプロセス空間に共有 / 非共有の概念が存在する場合には，メモリモデルも対応して共有 / 非共有な領域に分離しておく方が望ましい．この領域分離がないと，両者の領域におけるアクセスパターン次第で，もう一方の領域がリプレースされる度合が変化する．さらに詳細なシミュレーションのためには，プロセスの共有 / 非共有領域がリプレースに際して区別されるべきである．加えて非共有なメモリ領域についてはスレッド毎に所有されるため，どこのメモリ上に存在するかが問題になり，スレッドにとってローカルなメモリの管理が必要になる．

3.2.3 ネットワークのモデル化

ネットワークのもっとも簡単なモデル化に，メッセージコストを規定し結合網上の競合を無視する方法がある．メッセージコストとは簡単に，“初期化コスト＋遅延” で表せられる．メッセージコストを現実的にするには，遅延はメッセージのサイズと通信距離の関数にする．しかし競合を無視した結果，ネットワークがボトルネックとはなり得ないために，OS 機能の方式の違いが性能差となって結果に現れないことがある（サンプル 2）．

ネットワーク上に資源制約を導入するには，まず結合網のバンド幅をパラメータ化し導入する必要がある．これで単位時間当たりに一定量以上のメッセージが結合網上を流れなくなる．さらに，競合が起きた場合に対応するパスの使用権の取り合いを制御すれば，現実に近い制約を再現することになる．さらに同時に接続可能なパスの組を制御することで，自然なネットワークをモデル化できる．例えば，バス結合なら同時に一方向にしか流れないが，ネットワークにスイッチを仮定する場合には，同時に接続可能な方向の組が複数存在する．このように結合網を実装すると，ネットワーク上のコストパラメータは不要になる．ただし送受信の処理コストは状況に応じて設定しても良い．

又これとは別に，ネットワーク上のトランザクション量を監視して，それを通信コストパラメータに反映させる方法も考えられる．しかしこの方法の場合には，接続パスの制御は行なえない．

3.3 プロセスモデル

3.3.1 プロセスの実行モデル

並列プロセスの一生は生成時刻及び，終了条件，並列度の変化パターン，同期のパターンでほぼ表現することができる．生成時刻を指定できれば，システムの負荷を変化させて柔軟なシミュレーションを行なうことが可能になる．トレースを利用している場合，終了条件はトレースの終了点となるが，メモリ参照列の生成にそれ以外の方法を用いている場合には，プロセスが終了する条件を与えなければならない．例えば有効実行時間でこの条件は指定することができる．並列プロセスは同期を繰り返すうちに，動的な要因の影響により同期待ちのペナルティーが少しずつ加算されていく．この手法ならこのような，スレッドが体験する動的要因による同期の遅れを反映することができる．実行効率に大きく影響する同期のパターンは，是非とも考慮されるべきものである．また，プロセスの並列度の変化は資源の獲得 / 解放を司る OS の資源管理方式と深く関わる．並列度の変化を導入することで，モデルがまた一歩詳細になる．これには，並列度の変化するタイミングと変化後の並列度を指定しなければならない．

表2 サンプル結果 2 (上段: 有効実行率, 下段: 終了時間)
Table 2 Sample results 2

algorithm	set B	set C
0	230 57.36	257 59.11
1	224 58.89	253 60.04
2	220 59.95	246 61.74
3	221 59.68	250 60.76

サンプル 2 ネットワーク上の競合を無視して失敗する例．
表2は，2 段 4 進木で競合を無視しリモートアクセスのコストを距離に比例させたネットワーク・モデルで，セット B,C をシミュレーションした結果である．競合がないためページの移動コストが一定で，プロセスのページの位置を考慮しないスケジューリング法でも，コピーページをシステム中で広域に持つことに恩恵を受けるため，性能差が出ない．セット C は並列度の大きなプロセスのメモリ使用量を多くしたためリプレースは起きやすい．しかし結果は，セット B と同じようにスケジューリング方式でほとんど性能が変わらない．ネットワークが 2 段でリモートアクセスのコスト差が小さく，競合がないためリプレースの影響が反映されなかったと考えられる．リードとライトの区別がないモデルのため，コピーページを多く持つことによる効率化が有利に働いた．

3.3.2 プロセスの組合せ

プロセスの組合せは，システム全体の負荷と関係する．OS の資源管理方式はシステムの負荷によって効果が異なる可能性がある．またより柔軟な実験を行なうには一回のシミュレーションの中で，システムの負荷を変化できることが大切である．

ここで負荷として考えられるものには主に，全プロセスでの総並列度やメモリの総使用量がある．前者は個々のプロセスがスケジューリングされる間隔に，ひいては後者と共にメモリリプレースの起こる度合に影響する．

またプロセスの並列度の組合せは，スケジューリングのされ方においてネットワークの形状との相性がある

表3 サンプル結果 3 (上段: 有効実行率, 下段: 終了時間)
Table 3 Sample results 3

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

サンプル 3 プロセスの組合せとネットワーク形状．**表3**は，4 段 2 進木のネットワークでのシミュレーションである．セット A のシミュレーションのアルゴリズム 1 と 2 だけほとんど性能差がないことが観察できる．セット A は並列度が 2，4，8，のプロセスのみで構成されており，スケジューリングされると常にちょうど全クラスタを埋めるようになる．これがアルゴリズム 1，2 の違いを吸収し，スケジューリングの状況は結局ほとんど同じであった．これが性能に差がでなかった原因だと思われる．

（サンプル 3）．

以上のようなプロセスの組合せの問題を考慮して，個々のプロセスの一生は設計されなければならない．

4. 方式モデルの具体化

さて，先に述べたような方法でプロセスのメモリ参照列を計算した場合，それが実際どのような振舞いを起こすかは，OS の資源管理方式と，シミュレーション環境の様々な部分のモデル化の具体的な方式に依存する．それが最終的にネットワークやメモリなどの実資源制約の影響を受けて動的な振舞いを決定し，その違いがシミュレーションでの性能差に現れるのである．ここでは幾つかの具体的な方式モデル化の例について述べる．

4.1 メモリリプレース

理論上もっとも効率の良いリプレース方式は LRU 方式である．実際の計算機では，LRU 方式に準じたアルゴリズムが実装してあることが多い．一般に，LRU 方式とこれらの方式との性能差はそれほどないと考えられている．また分散メモリ型並列計算機の場合には，リプレース方式の性能差が全体性能へ及ぼす影響度は小さいと思われる．従って，メモリのリプレースが対象とする OS の資源管理方式から独立したものである場合には，メモリのリプレース方式として理想的な LRU 方式を採用することは妥当である．

4.2 共有メモリ方式

並列プロセスにとって，スレッド間で共有しているデータの管理方式は性能に大きく影響する．分散共有メモリを仮定する場合，特に共有領域の一貫性制御方式は，ネットワーク・トランザクションを必要とするためシミュレーションの性能評価には欠かせない要素となる（サンプル 4）．従ってまずリードとライトの区別が存在することが必要である．共有された領域にライトが起こることにより一貫性の処理が必要になり，同一領域へのアクセスがシリアライズされる．

表4 サンプル結果 4 (有効実行率)
Table 4 Sample Results 4

アクセス比	2 段 4 進木	1 段 16 進木
99-1	86.96	95.24
95-5	74.07	90.91

サンプル 4 共有領域へのライトの影響．**表4**は，簡単な共有メモリ方式を実装し，並列度 16 のプロセスを一つだけ走らせシミュレーションした結果である．一貫性制御には update protocol を用いた．プロセスは 3:1:15 の比でリード，ライト，その他の実行をし，非共有領域 300 ページに対して共有領域を 100 ページとした．平坦な頻度分布なので共有するスレッド数は小さい．共有の度合が増加するに従い，効率が劣化していることが分かる．ネットワークの競合を考慮すればさらに効率は低い結果になると思われる．

最も一般的な共有メモリ方式はページ単位に一貫性を保つ方式である．この場合ページテーブルを管理し，このテーブル上でホームやコピーセットの管理，アクセスの同期制御を行なう．特定の一貫性管理方式の実装法が，シミュレーションの結果に影響するのを避けるために，ページテーブルはグローバルに管理し，ホームやコピーセットの管理分のコストはなしとしても良い．また，各種 ack や invalidate，update のメッセージなどの制御用のメッセージは，ネットワークモデルにあわせて適当にサイズを指定したり，コストパラメータを調節することで，影響する度合を考慮することも可能である．つまり，コスト 0 にして無視することも可能であるし，共有メモリ方式のハード実装かソフトウェアレベルでの実装かを表現することも可能である．

4.3 その他の方式モデルの具体例

モデル化するには，様々な部分について特定の方式を仮定しなければならない．逆にそれらを適切な方式で実装すれば，一歩詳細なモデルを作ることができる．

例えば，リプレースされたメモリ断片の退避先の問題がある．空きがあればリモートなメモリに退避する方法も考えられるが，簡単にディスクに退避する方法で十分であろう．

ネットワーク上でのパス選択の方法は，バス結合を仮定するならある点に接続されている線をラウンドロビンに探索し，メッセージを見つけたところでその線を選択する方法が考えられる．

プログラムのロードは通常のメモリアクセスと同じ扱いにすると簡単であるが，初期アクセスのみコストを軽くするアイディアもある．しかしロード機構を実装するにはプロセスのプログラム部の大きさが分かっていなければならない．

目的のシミュレーションにとって取り立てて重要でない部分を実装する場合には，その部分での現象が複雑過ぎたり，実装した方式に固有の性能評価への影響が強く

なり過ぎないように注意しなければならない.

5. 時間と空間

モデルの詳細度を上げていくと，現実の並列実行と同じ量のメモリなどの資源を使用するプロセスを，実際と同じ時間分だけ走らせるようなシミュレーションを行なうには多大な時間を要するようになる．そこで，プロセスが使用する資源量とシミュレーションする時間を適当にスケールダウンしてシミュレーションを行なう必要がある．このような場合，空間と時間軸方向のスケールダウンの関係に注意しなければならない.

例えば，スケールダウンしたプロセス空間に対してページサイズの設定が相対的に多き過ぎれば，ネットワーク上のページの移動コストを大きく見積もり過ぎたことになる．また，スケールダウンされた空間でのプロセスの行動は時間的にはどれくらいのことに相当するのかも分かりにくくなる．そこでタイムシェアリング・システムの一時分割の時間設定や，プロセッサの動作時間への欲求度の計算法などを調節する必要がある． OS 機能の処理コストについても，見積もる必要がある場合には適当にスケールダウンしなければならない.

またスケールダウンした結果，シミュレーションでのプロセッサの単位動作が実際の一処理には対応しなくなる．従ってシミュレーションがクロックベースに実行されるなら，シミュレーションの結果得られた性能の差は，精度が低くいと考えられる.

6. ま と め

分散並列環境における OS レベルの資源管理方式を実験可能にするシミュレーションモデルについて述べた．以上の議論に基づいてシミュレータを実装している．現在我々のシミュレータに実装してある主な機構は以下の通りである.

- 有効実行率の検出
- 木構造のネットワーク生成
- 共有 / 非共有分離したページ単位の参照頻度表 (属性としてライト / リード / 演算比，一貫性制御プロトコル) を利用した参照列生成
- 分散共有メモリシステム
- LRU 方式のメモリリプレース
- ネットワークのバンド幅をパラメータ化
- ランダムな組合せでスレッドの同期 (有効実行時間で間隔指定)
- プロセスを自由に組合せ可能

今後はこのシミュレータを用いて SSS–CORE のスケジューリング法の評価及び他のスケジューリング法との比較を行なう予定である．また本稿で述べたように厳密に現実をシミュレーションするには限界があるため，並行して SSS–CORE の実装を行なう.

付　　録

表5　プロセス構成 – セット A
Table 5　Process set A

要求プロセッサ数	プロセス数	空間サイズ [pages]∗ プロセス数
4	4	$50 * 2, 300 * 2$
8	4	$50 * 2, 300 * 2$
16	3	$50 * 2, 100 * 1$

表6　プロセス構成 – セット B
Table 6　Process set B

要求プロセッサ数	プロセス数	空間サイズ [pages]∗ プロセス数
3	5	$50 * 2, 100 * 2, 150 * 1$
6	6	$200 * 6$
13	3	$50 * 3$
16	1	$50 * 1$

表7　プロセス構成 – セット C
Table 7　Process set C

要求プロセッサ数	プロセス数	空間サイズ [pages]∗ プロセス数
3	5	$50 * 2, 100 * 2, 150 * 1$
6	6	$200 * 6$
13	3	$75 * 3$
16	2	$50 * 2$

アルゴリズム 0　ランダムなスケジューリング.
アルゴリズム 1　一定の順序でプロセッサを割り当て.
アルゴリズム 2　プロセスのページを持つクラスタから．足りなければフリーなところへ.
アルゴリズム 3　プロセスのページを持つクラスタのみを割り当て.

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) 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 (1994).
2) 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, 信学技法, Vol. 95, No. 210, pp. 111–118 (1995).
3) 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).
4) 松本尚ほか: 粒度に基づいた並列計算の分類法とマルチプロセッサの資源管理法について, 日本ソフトウェア科学会第 7 回大会論文集, pp. 133–136 (1990).
5) Squillante, M. S. and Lazowska, E. D.: Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 131–143 (1993).