システムソフトウェアと
オペレーティング・システム 68－1
（１９９５．３．２４）

モジュール構成のマルチプロセッサ・スケジューリング・シミュレータ

藤崎 直哉* 福田 晃
{naoya-f,fukuda}@is.aist-nara.ac.jp

奈良先端科学技術大学院大学 情報科学研究科

*富士通（株）

本稿では、マルチプロセッサシステムの空間分割スケジューリング方式の１つである２レベルスケジューリングを対象として、スケジューリング方式およびアーキテクチャやアプリケーションなどの多角的、総合的な性能評価を行うためのシミュレータである、モジュール構成の２レベルスケジューリング・シミュレータについて述べる。本シミュレータは、主にスケジューラモジュール、アーキテクチャ依存モジュール、プロセス依存モジュール、およびスレッド実行モジュール等で構成されている。本モジュール化により、スケジューリング方式、ターゲットアーキテクチャおよびアプリケーションの実行パターンについて、各種のシミュレーションが容易になった。

Multiprocessor Scheduling Simulator with Modularity

Naoya Fujisaki*, Akira Fukuda
{naoya-f,fukuda}@is.aist-nara.ac.jp

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma 630-01, JAPAN

*Fujitsu Ltd.
1-17-25 Shin-Kamata, Ota-ku, Tokyo 144, JAPAN

We study two-level scheduling policy which is a kind of space-multiplexing scheduling one for multiprocessors. This paper describes a two-level scheduling simulator with modularity to evaluate performance of strategies in the two-level scheduling for a wide variety of target hardware architectures and applications. The simulator consists of modules of scheduling strategies, target architectures, process patterns, thread execution and so on. By the modularization of the simulator, we can evaluate the performance of the scheduling for a variety of environments where scheduling strategies, target architectures, and execution patterns of applications are easily changed.

1 はじめに

マルチプログラミング環境下におけるマルチプロセッサシステムのスケジューリングには，時分割 (time-multiplexing) 方式と空間分割 (space-multiplexing) 方式に大別される．我々の研究室では，空間分割方式の１つである２レベルスケジューリングに関する研究をシミュレータを用いて行っている [1, 2, 3]．現在のシミュレータでは，構成の見通しが悪いため，スケジューリン グ方策，ターゲットハードウェアアーキテクチャ（プロセッサ台数，メモリアーキテクチャ），アプリケーションプロセスの実行パターンの変更に多大な労力を必要としている．そこで，これらについて容易に変更できるシミュレータを作成する必要が生じた．本稿では，シミュレータのネットワークモデルの工夫とシミュレータプログラムのモジュール化によって，上記の変更に容易に対処できる２レベルスケジューリング・シミュレータについて，設計と開発について述べる．本稿では，実行するプログラムをプロセス，プロセス内の並列実行可能なアクティビティをスレッドと呼ぶことにする．

2 ２レベルスケジューリング

２レベルスケジューリングは，システム全体の物理プロセッサ群を静的／動的にグループ（プロセッサグループ）に分割し，そのプロセッサグループをプロセス対応に割り当てて，プロセス内のスレッドのスケジューリングは，プロセッサグループ対応のスケジューラによって行うものである．

２レベルスケジューラの構造を図1に示す．２レベルスケジューラは，グローバルスケジューラと呼ばれる上位スケジューラとプライベートスケジューラと呼ばれる下位スケジューラから構成される [2]．

(1) グローバルスケジューラ

物理プロセッサの動的なグルーピングを行なうグループサーバと，グループに属さないフリーな物理プロセッサを管理するプールサーバから構成される．システム内の任意のプロセッサは，グループサーバによって管理されているプロセッサグループ，もしくはプールサーバによって管理されているプロセッサプールのいずれかに属する．プロセッサグループはプロセスに対応に存在する．グローバルスケジューラはシステム内に唯一存在し，カーネル空間で動作する．

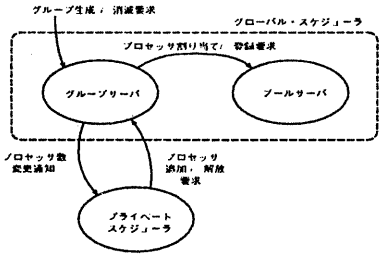

図 1: ２レベルスケジューラの構造

(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対応に存在する．各プライベートスケジューラは，各々レディキューを有し，プロセス内のスレッドのスケジューリングを行う．プライベートスケジューラは，所属するプロセスと同じユーザ空間で動作する．

3 現有のシミュレータ

本節では，我々が使用しているシミュレータ実行環境と現有のシミュレータの問題点を述べる．

3.1 シミュレータ実行環境

シミュレーション用言語として SLAM[4, 5] を用いている．SLAM では，システム内を動き回る” もの” を要素と定義している．ここでは，SLAM でいう要素のことをトランザクションと呼んでおく．そのトランザクションの動く規則を

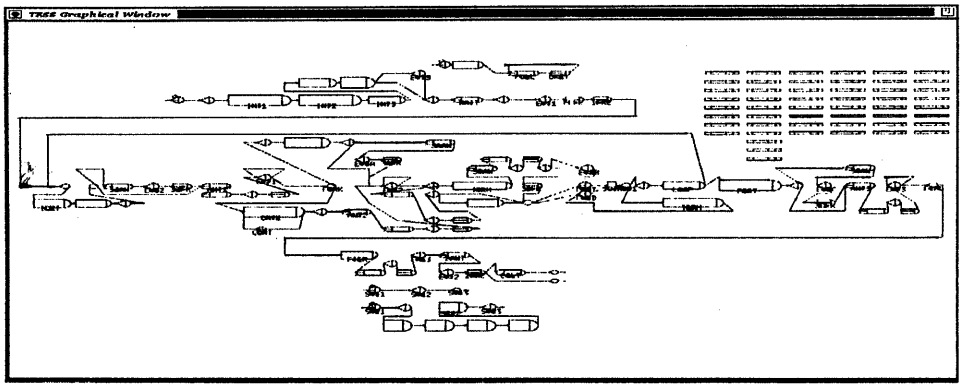

図 2: 現有シミュレータの SLAM ネットワークモデルの例

SLAM シミュレーション言語で記述したものが SLAM ネットワークモデルである．SLAM ネットワークモデルは，ノードとアークから構成される．つまり，シミュレートしたいシステムは，ノードとアークを用いて表現されたネットワークの中をトランザクションが移動するということでモデル化されている．また，ノードやアークには，SLAM に標準装備されている機能があるが，これらの機能では表現できなかったり，表現しにくい場合に，カスタムメイドのシミュレーションモデルが構築できるようにするために，FORTRAN インタフェースを提供している．この場合，あるノードから自分が作成した FORTRAN プログラムにサブールチンコールするように記述すると，トランザクションが当該ノードに入ったときに，当該サブルーチンが実行されることになる．

3.2 現有のシミュレータの問題点

現有シミュレータは，シミュレータのネットワークモデルのトポロジが，対象とするハードウェアアーキテクチャ（主にメモリアーキテクチャ），2 レベルスケジューリング方式，プロセスの実行パターンに強く依存した構造になっている．さらに，ユーザ記述の FORTRAN プログラムもネットワークモデルのトポロジに依存している．図 2 に，あるハードウェアアーキテクチャ，スケジューリング方式，プロセス実行パターンをシミュレーションする場合のネットワークモデルを示す．

従って，例えば，プロセッサ台数の変更を行なう場合，ネットワークモデルにおいて，ハードウェアアーキテクチャを反映している部分のトポロジを変更する必要がある．さらにユーザ記述 FORTRAN プログラムも書き換える必要があるが，変更部分を特定するのに労力を要している．このように，現有のシミュレータは，構成の見通しが悪く，シミュレーション対象を変更しようとすると，多大な労力を必要としている．そこで，これらについて容易に変更できるシミュレータを作成する必要が生じた．

4 モジュール構成の 2 レベルスケジューリングシミュレータ

本節では，上記の変更，すなわちターゲットハードウェアアーキテクチャ（主に，プロセッサ台数とメモリアーキテクチャ），2 レベルスケジューリングポリシ，プロセス実行パターンの変更に，容易に対処できる 2 レベルスケジューリング・シ

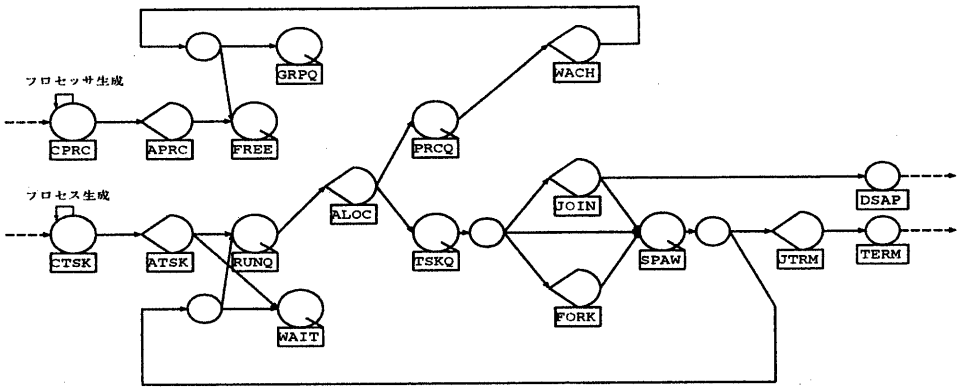

図 3: モジュール構成シミュレータのSLAM ネットワークモデル

ミュレータについて、設計と開発について述べる.

4.1 設計方針

設計方針は，以下の通りである.

- シミュレータのSLAM ネットワークトポロジを，ターゲットとするシミュレーションモデルから独立させる.

  ターゲットとするハードウェアアーキテクチャ，スケジューリング方式，プロセス実行パターンに依存しないように，シミュレータのネットワークトポロジを決定する. これにより，上記変更の際，SLAM ネットワークのトポロジを変更する必要がなくなる.

- 機能モジュール化を採用する.

  上記変更を行う場合，ユーザ記述のFORTRAN プログラムを書き換えることになるが，書き換えを容易にするため，ターゲットとするシミュレーションモデルを機能的にモジュール化を行なう.

4.2 設計

上記設計方針に従い，設計したシミュレータ（SLAM ネットワークモデルと FORTRAN プログラムから構成される）について述べる.

4.2.1 SLAM ネットワークモデル

まず，ネットワークモデルの構築に際し，以下の2つを検討する必要がある.

1) ターゲットとするシミュレーションモデルの要素（プロセッサ，プロセス，スレッドなど）をSLAM が提供する要素（ノード，トランザクション）にどのように対応づけるか？

2) 任意のプロセス実行パターンをシミュレートするには，どのような構造をSLAM ネットワークモデルに反映させるか？

上記の問題に対し，新規のシミュレータでは以下のように対処する。

1) プロセッサ，プロセス，スレッドをSLAM においては全てトランザクションに対応させる.

   現有シミュレータでは，プロセス，スレッドをトランザクションに対応させていた. この対応自身は，問題が生じなかった. 問題は，プロセッサをSLAM においてはノードの1つであるサーバに対応させていたことである. この対応は，待ち行列モデルなどのシ

ミュレータでは，多くのシミュレータ作成者が行う，自然な対応である．しかし，この対応によって，プロセッサ台数を変更する場合，各種のSLAMリソースの再定義および定義の追加を余儀なくされ，多大な労力を要することになった．そこで，プロセッサをトランザクションに対応させるようにした．この対応により，プロセッサ台数の変更は，対応するトランザクションの発生個数を単に変更するだけで済む．

2) SLAMネットワークに汎用性のあるFORKノードとJOINノードを設ける．

現有シミュレータでは，プロセス実行パターンとして，1つのスレッドから複数のスレッドが同時生成され，それらがまた1つのスレッドに集束する平行処理モデルであるparbegin/parendを複数回繰り返すパターンのみを対象とし，本パターンに強く依存したSLAMネットワークモデルになっていた．従って，他タイプの平行処理モデルをシミュレートしようとすると，ネットワークトポロジ自身を変更する必要があった．一般に，プロセスは，スレッドが生成，消滅しながら実行されるので，この機能のみをSLAMネットワークを持たせることにした．スレッドの生成形態，消滅形態は，プロセス内の平行処理モデルに依存するので，これらの形態は，FORTRANプログラムの方で，吸収させることにした．

上記に従い，さらに若干の検討を加えて設計したSLAMネットワークを，図3に示す．

4.2.2 モジュール化

FORTRANプログラムのモジュール化にあたっては，まずFORTRANプログラムが担当する機能を分類する必要がある．これらは，大きく以下から構成される．

1) ターゲットハードウェアアーキテクチャに関する機能

2) プロセス実行パターンに関する機能

3) 2レベルスケジューラに関する機能

4) シミュレータ自身に関する機能

上記の項目に対して，シミュレータが必要とする機能について検討する．

1) ターゲットハードウェアアーキテクチャに関する機能

ハードウェアアーキテクチャには，プロセッサ／ネットワーク／メモリ・アーキテクチャがあるが，プロセッサに関しては，それをSLAMではトランザクションに対応させているので，残りのネットワーク／メモリアーキテクチャに関するシミュレートを，SLAMでは1つのモジュール（アーキテクチャ依存モジュール）で行うことにした．

2) プロセス実行に関する機能

これに関しては，a) プロセス内で複数のスレッド同士がどのような形態で平行実行するかというプロセス実行パターンと，b) 各スレッドの実行時間の算出の2つがある．a) に関しては，1つのモジュール（プロセス依存モジュール）を設け，各スレッドが実行を完了したとき，当該スレッドが次に起こすアクション（新たなスレッドを生成させるフォークなのか，他スレッドと同期をとるジョインなのか，の2つのアクション）を決定する．b) に関しても，1つのモジュール（スレッド実行モジュール）を設ける．スレッド実行時間は，データ配置およびメモリ／ネットワークのハードウェアアーキテクチャに依存するので，スレッド実行モジュールは，アーキテクチャ依存モジュールを呼び出して，実行時間を算出することになる．

3) 2レベルスケジューラに関する機能

2節で述べたように，2レベルスケジューラは，グループサーバ（プロセッサグループ

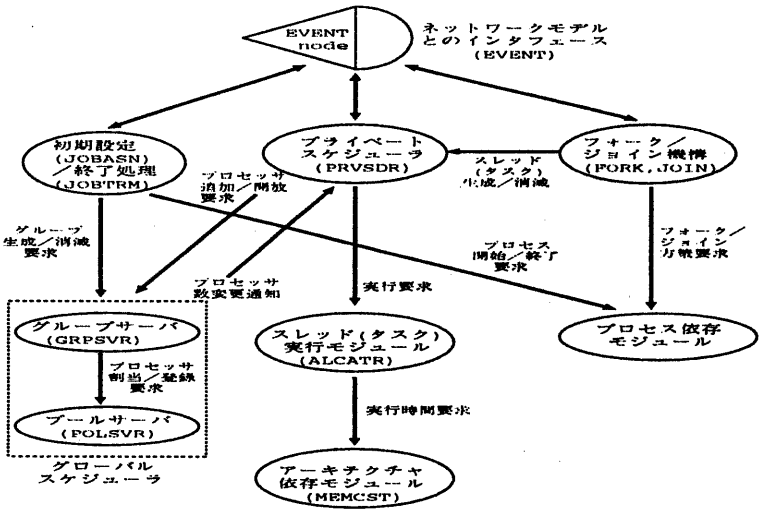

図 4: FORTRAN プログラムのモジュール間インタフェース

の割り当て管理を行う)，プールサーバ (フリープロセッサを管理する)，プロセッサグループ (またはプロセス) 対応のプライベートスケジューラから構成されるので，これら 3 つのモジュールを設ける．

4) シミュレータ自身に関する機能

シミュレータ自身に関する機能として，SLAM ネットワークから FORTRAN サブルーチンへのエントリであるモジュール (インタフェースモジュール)，スレッドの実行実行完了の際の他モジュールを呼び出すための中間モジュール (フォーク／ジョイン機構モジュール)，およびトランザクションの初期化／終了処理を行うモジュール (初期設定／終了処理モジュール)，が必要となる．

上記に従って，設計したモジュールおよび各モジュール間インタフェースを図 4に示す．

4.3 シミュレータの動作概要

図3,4を参照して，シミュレータの動作概要を述べる．

(1) プロセッサの初期化

以下の処理を行う．

1) CPRC ノードから対象とするプロセッサ台数分のトランザクション (プロセッサ) を発生させる．

2) 生成されたプロセッサは，APRC ノードに入り，ここで FORTAN プログラムにとび，初期設定モジュールで初期化情報を付加され，SLAM ネットワークに戻る．

3) APRC ノードからさらに FREE ノードに進み，フリープロセッサとして格納される．

(2) プロセス (またはスレッド) とプロセッサの動作

1) CTSK ノードからある決められた分布に従って，プロセス (トランザクション) が発生される．

2) ATSK ノードで，FORTRAN プログラムにとび，初期設定モジュールおよびプロセス依存モジュールで，初期化情報と次にとるべきアクション (フォークするか否か) を付加されるとともに，グループサーバをよぶ．

3) グループサーバは，グループ割り当てポリシに従って，プロセッサグループを割り当てる．割り当てられるプロセッサがフリープロセッサである場合は，FREE ノードに格納されている当該フリープロセッサを GRPQ ノードに格納する．ここで，GRPQ ノードは，プロセッサグループに属しているプロセッサを管理するノードであり，本来ならば，グループ対応にノードを設けて管理する必要があるが，ここでは，簡単のため1つのノードで管理している．この後，SLAM ネットワークに戻る．

4) グループを割り当てられた場合は，当該プロセスは RUNQ ノードに入り，割り当てられなかった場合は，WAIT ノードに入り，グループ割り当てを待つ．

5) RUNQ ノードに入ったプロセスは，ALOC ノードに進む．ここで，プロセス実行パターンとしては，始めから複数スレッドとなる平行処理パターンではなく，まず1つのスレッドから始まるパターンを仮定する．従って，プロセスの実行は，1つのスレッド実行の開始となる．この仮定は，当該スレッドの実行時間を非常に小さく設定することによって，はじめから複数スレッドの平行処理パターンを近似できるので，さほど厳しい仮定ではないと考えている．ALOC ノードから，FORTRAN プログラムにとび，排他制御によりプライベートスケジューラをよぶ．プライベートスケジューラは，プロセス内のスケジューリングポリシに従って，対応グループ内のプロセッサを当該スレッドに割り当て，スケジューラとスレッドの実行時間分，当該プロセッサをハントする．具体的には，GRPQ ノードに格納されている対応グループに属するプロセッサの中から，スケジューリングポリシに従って，プロセッサを選択し，そのプロセッサを PRCQ ノードに移動させる．さらに，スレッド実行モジュールをよび，プロセスの実行時間を算出する．実行時間は，アーキテクチャに依存するので，アーキテクチャ依存モジュールを呼んで，実行時間を算出することになる．この処理後，SLAM ネットワークに戻る．

6) PRCQ ノードに進んだプロセッサは，算出された実行時間分，当該ノードに滞在した後，WACH ノードに進む．WACH ノードでは，FORTRAN プログラムにとんで，プライベートスケジューラを呼んで，スケジューリングポリシに従って，当該プロセッサの次にとるアクションを決定し，WACH ノードに戻る．決定されたアクションがグループに属したままと決定された場合は，GRPQ ノードに進む．アクションがグループからの解放である場合は，FREE ノードへ進む．

7) ALOC ノードに戻ったスレッドは，算出された実行時間分，ALOC ノードに滞在した後，TSKQ ノードに入り，次にとるべきアクション（フォークするか，他スレッドと同期してジョインするか，というアクション）に従って，JOIN ノードまたは FORK ノードへ進む (フォークと決定されていれば FORK ノードへ，ジョインと決定されていれば JOIN ノードへ進む)．

8) FORK ノードへ進んだスレッドは，スレッドを新たに生成させるため，FORTRAN プログラムにとんで，当該スレッドと新たに生成するスレッドのとるべき次のアクション (さらにフォークするのか，もしくはジョインするのかというアクション) を決定する．さらに，プラーベートスケジューラを呼んで，スレッド生成のための実行時間を算出して，SLAM ネットワークに戻り，算出された実行

時間分FORKノードに滞在した後（このとき，プロセッサもハントしておく），SPAWノードへ進み，さらに，新たなスレッドであるトランザクションを実際に生成する．これらのスレッドは，RUNQノードへ進む．

9) 一方，JOINノードへ進んだスレッドは，FORTRANプログラムにとんで，プロセス依存モジュールを呼んで，さらに，当該スレッドのとるべき次のアクションを決定する．この決定には，当該スレッドの実行完了の後，a) 他スレッドとジョイン後も，さらに当該スレッドを実行する，b) 他スレッドとジョイン後，当該スレッドは消滅する，c) 他スレッドとジョインして，さらに，当該プロセスの実行が終了する，d) 単に当該プロセスの実行が終了する，の4つの場合がある．

10) 上記9) で，a) と決定された場合は，SPAWノードへ進んだ後，RUNQノードへ進む．b) と決定された場合は，DSAPノードへ進み，スレッドであるトランザクションを消滅させる．c) の場合には，1つのスレッドを残して，残りのスレッドはDSAPノードへ進み，消滅する．残った1つのスレッドは，プロセス終了の処理を行うため，SPAWノードへ進む．さらに，JTRMノードへ進んで，FORTRANプログラムへとび，グローバルスケジューラおよび終了処理モジュールを呼んで，プロセッサグループの消滅処理を行い，SLAMネットワークへ戻り，さらにTERMノードへ進んで当該トランザクションを消滅させる．d) の場合は，c) で述べた残りの1つのスレッドと同様の処理をする．

11) RUNQノードへ進んだスレッドは，スケジューリングポリシに従って，プロセッサの割り当てを待つ．割り当てられると，上記の処理を繰り返す．

5 まとめ

本稿では，マルチプロセッサにおける2レベルスケジューリング方式を対象として，ターゲットハードウェアアーキテクチャ，スケジューリングポリシ，プロセス実行パターンについて，多角的な評価を行うためのシミュレータについて述べた．設計・開発したシミュレータは，シミュレータ自身のネットワーク（SLAMネットワーク）トポロジを変更する必要はなく，かつFORTRANプログラム内のモジュールの変更のみで，上記評価を行うことができる．

I/Oのモデル，本シミュレータを用いた多角的な評価等は今後の研究課題である．

参考文献

[1] 藤木亮介, 甲斐久淳, 福田晃：, “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アルゴリズムの評価”, 情報処理学会コンピュータシステム・シンポジウム, pp.67-74(1993).

[2] 甲斐久淳, 藤木亮介, 福田晃：, “マルチプログラミング環境のマルチプロセッサにおける2レベル・スケジューリング ースケジューリング構造と性能評価ー”, 情報処理学会論文誌, Vol.35, No. 10, pp.2115-2128(1994).

[3] 甲斐久淳, 藤木亮介, 福田晃：, “メモリ管理と協調動作する2レベルスケジューリング”, 情報処理学会並列処理シンポジウム JSPP'94, pp.319-326(1994).

[4] A. Alan B. Pritsker and Claude Dennis Pegden, “Introduction to Simulation and SLAM II”, *Systems Publishing Corporation*(1986).

[5] 森口晋, 相沢りえ子, “SALM IIによるシステム・シミュレーション入門” 構造計画研究所(1986).