2006－OS－101（7）
2006／2／16

細粒度マルチスレッド環境での スケジューリングオーバヘッド低減機構の評価

小川泰彦[†]　乃村能成[†]　日下部茂[††]
谷口秀夫[†]　雨宮真人[††]

細粒度マルチスレッド実行環境では大量のスレッドが実行されるため，OS は効率のよいスケジューリング方式が要求される．これにはスレッド間の同期を管理する同期表を用いて，同期表の中の同期値のみ操作することでスケジューリングのオーバヘッドを低減するという機構が考えられる．本論文では，同期値を操作する際の具体的な方針を検討し，評価を行う．そして，評価結果から機構がスレッドを制御可能であることを示す．

Evaluation of a Process Scheduling Mechanism for Fine Grain Multithread Architecture

Yasuhiko Ogawa,[†] Yoshinari Nomura,[†] Shigeru Kusakabe,[††]
Hideo Taniguchi [†] and Makoto Amamiya[††]

In a fine grain multi-thread environment, operating systems are required to control innumerable threads efficiently. As a solution for the thread scheduling overhead problem, it is a good idea to control threads by injecting extra delays to their inter-thread synchronization. In the manner of this idea, we have to consider the optimal frequency and pattern of the delay injection. In this paper, we consider and evaluate some patterns of the delay injection.

1. はじめに

近年，プロセッサの性能向上のため，SMT（Simultaneous Multi-Threading）や CMP （Chip Multi-Processor）といった 1 つのチップ上で複数の命令流を実行するプロセッサが提案[1]され，実用化[2]されている．我々は，並列処理と親和性の高いデータフローモデルを基盤として，スレッドの並列実行を追及した Fuce プロセッサ[3)4)]を開発している．

Fuce は，複数のスレッド実行ユニットを搭載し，複数のスレッドを並列に処理することができる．Fuce におけるスレッド（以降，マイクロスレッド）は，継続概念によるイベント駆動によってのみ制御される．また，マイクロスレッドは，排他的に実行され，走行終了まで中断されることはない．マイクロスレッドはハードウェア（以降，H/W）で管理されるため，マイクロスレッド間の継続による同期処理は，H/W によって高速に処理される．また，同期が解決したマイクロスレッドは，H/W のキューに投入され，高速に処理される．従って，Fuce では，大量のマイクロスレッドを高速に実行可能である．しかし，H/W でマイクロスレッドの管理と同期処理を行うことで，マイクロスレッドはオペレーティングシステム（以降，OS）の関知しないところで次々と実行される．このままでは，OS は多数のマイクロスレッドの集合体として構成されるプロセスのスケジューリングを行うことができない．

そこで，このような大量のスレッドが並列に走行する環境で OS のスケジューリングオーバヘッドを低減しつつ，OS がマイクロスレッドの実行を制御する方法として，マイクロスレッドの継続を制御する同期カウンタ値を増減することで，マイクロスレッド間の同期を意図的に遅延させ，プロセス全体の動作速度を調整するという機構[5]が考えられる．

本稿では，上記の機構の評価を行い，機構が多数のマイクロスレッドスレッドの集合体として構成される

† 岡山大学大学院自然科学研究科
The Graduate School of Natural Science and Technology, Okayama University
†† 九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

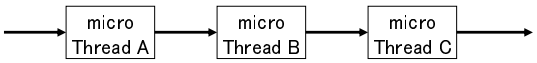

図 1 継続概念

プロセスの動作を制御可能であるか考察する．以降，2 章で Fuce アーキテクチャについて述べ，3 章で評価を行うための尺度について述べる．さらに，4 章で評価を行うためのシミュレータについて述べ，5 章で機構の評価を行う．最後に，6 章でまとめと今後の課題を述べる．

2. Fuce アーキテクチャ

Fuce（Fusion of communication and execution）は，通信処理も通常処理と同等に扱い，すべてをマイクロスレッドとして統一的に処理するという思想で設計されている．本章では，Fuce アーキテクチャと，Fuce 上でマイクロスレッドを制御する機構について述べる．

2.1 Fuce のスレッド並列実行モデル

Fuce のスレッド並列実行モデルは，データフロー計算モデルに基づいた継続という概念を核にして成立している．**図 1** に継続の概念を示す．図 1 は，3 つのマイクロスレッド A，B，C の依存関係を示している．B は A の計算結果を必要とし，C は B の計算結果を必要としている．これら 3 つのマイクロスレッドを実行するためには，A は計算結果とともに B に対して通知を送り，B は計算結果とともに C に対して通知を送らなければならない．Fuce では，この結果の通知を継続と呼ぶ．また，A は B に継続し，B は C に継続するという．

任意のマイクロスレッドに対して継続される元のマイクロスレッドの数を fan-in 値，継続する先のマイクロスレッドの数を fan-out 値と呼ぶ．Fuce のスレッド実行制御は，すべて継続によって決まる．マイクロスレッドは，継続されるたびに自 fan-in 値をデクリメントし，その値が 0 になったときに実行される．そして，そのマイクロスレッドは，処理を終えるまでいかなる干渉も受けず，中断されることなく走り切る．

2.2 スケジューリングオーバヘッド低減機構

Fuce では，マイクロスレッドの情報を Activation Contorl Memory（ACM）に登録し，管理する．ACM の構造を**図 2** に示す．ACM は，マイクロスレッドの現在の fan-in 値，fan-in 値の初期値である fan-init 値といったマイクロスレッドの同期情報を含んでいる．スケジューリングオーバヘッド低減機構は，これら ACM に登録されている同期情報を操作することでマイクロスレッドを制御する．

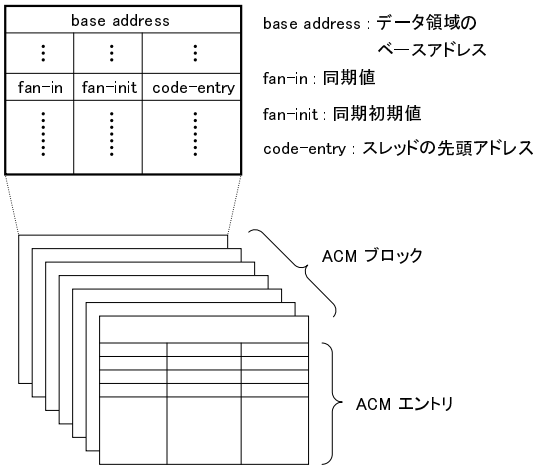

図 2 ACM

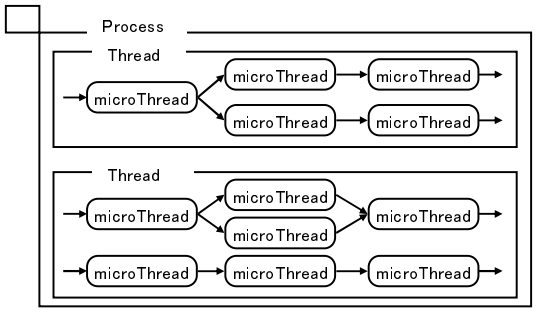

図 3 プロセス，スレッド，マイクロスレッドの構成

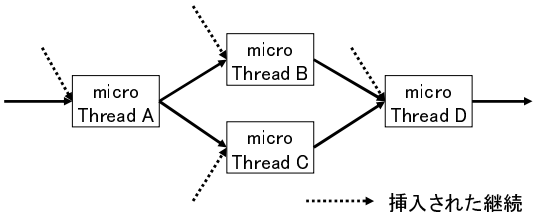

図 4 継続挿入

Fuce アーキテクチャ上で OS を構成し，プロセスの実行を制御する場合を考える．Fuce アーキテクチャ上では，プロセスは複数のスレッドから構成され，スレッドは複数のマイクロスレッドから構成される．この様子を**図 3** に示す．プロセスの走行を停止させるためには，プロセスを構成するマイクロスレッドに継続を挿入する．具体的には，同期カウンタ値である fan-in 値を 1 増やす．この様子を**図 4** に示す．継続を挿入されたマイクロスレッドは，マイクロスレッド間の継続による同期が解決しなくなるため，走行を開始できなくなる．プロセスの走行を再開するには，マ

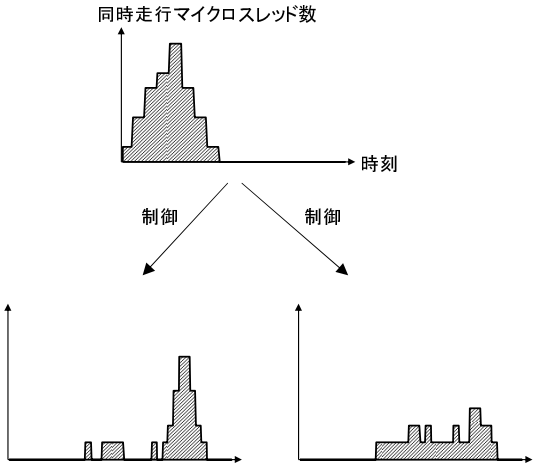

図 5 同時走行マイクロスレッド動作の変化

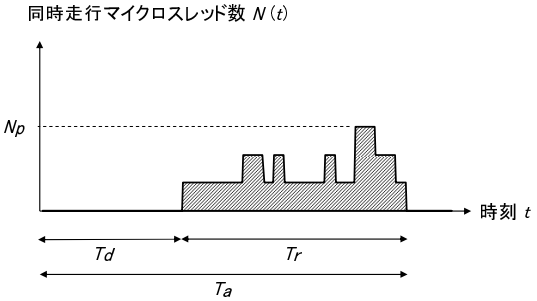

図 6 評価尺度

イクロスレッドから挿入した継続を外す．つまり，同期カウンタ値である fan-in 値を 1 減らす．そうすることで，マイクロスレッド間の同期状態が元の状態に戻り，プロセスは，走行を再開できるようになる．

このように，スケジューリングオーバヘッド低減機構では，マイクロスレッドの走行状態そのものを細かく制御するのではなく，マイクロスレッドの同期関係に手を加え，マイクロスレッドの実行開始を制御することで，プロセスおよびスレッドの走行と停止を制御する．また，プロセスを構成するマイクロスレッドの中からある確率でいくつかを選択し制御することで，並列に走行するマイクロスレッドの数を調整できる．そうすることで，プロセスの動作速度の調整，およびスレッド実行ユニットの使用状況（プロセッサ占有率）の調整ができると見込める．

3. 評価尺度

スケジューリングオーバヘッド低減機構では，個々のマイクロスレッドの走行状態を厳密に制御するのではなく，ACM エントリの fan-in 値を増減させ，マイクロスレッドの同期関係に手を加えることで制御を行う．

プロセスを構成するマイクロスレッドを制御することにより，プロセスはマイクロスレッドを制御しない場合とは異なる動作をする．この様子を**図 5** に示す．図 5 は制御開始時からの同時に走行したマイクロスレッド数をグラフにしたものである．このグラフの違いは，機構がマイクロスレッドの同期関係にどう手を加えるかという方針によって生じる．従って，多数存在する方針間の優劣を定量的に評価するために，以下の尺度を設定する．**図 6** にその尺度に用いる値の関係を示し，以下に説明する．

(1) $N(t)$：時刻 t において同時に走行しているマイクロスレッドの数
この値は，時刻 t においてプロセスがいくつのスレッド実行ユニットを占有しているかを示す．

(2) N_p：$N(t)$ の最大値
この値は，プロセスが実行中に最大いくつのスレッド実行ユニットを占有したかを示す．あるプロセスが使用できるスレッド実行ユニットの数を調整してプロセス間の CPU 資源利用を調整したい場合，この値を一定値以下にすることが要求される．

(3) N_v：$N(t)$ の分散
この値は，同時走行マイクロスレッド数の安定性を示す．$N_v = 0$ ならば，マイクロスレッドの同時走行数は常に一定である．逆に，N_v の値が大きい場合，マイクロスレッドの同時走行数は一定にならず，激しく変化する．この同時走行数を一定に保つことができれば，$N(t) \fallingdotseq N_p$ と仮定できるようになるため，OS が各プロセスに CPU 資源を割り当てる処理が簡便になる．

(4) T_d：制御を開始してからプロセスが走行を始めるまでの遅延時間
この値は，機構がマイクロスレッドを制御することにより発生するプロセス実行開始までの遅延時間を示す．この遅延時間が大きいとプロセスの応答性に問題が発生すると考えられるため，この値は小さいほど良い．

(5) T_r：プロセスの走行時間
この値は，機構がマイクロスレッドを制御することにより生じたプロセスの処理時間の変化を比較するために用いる．

(6) T_a：制御を開始してからプロセスが走行を終えるまでの時間 $(T_d + T_r)$
リアルタイム処理を行うプロセスの場合，この値を一定値以下にすることが要求される．

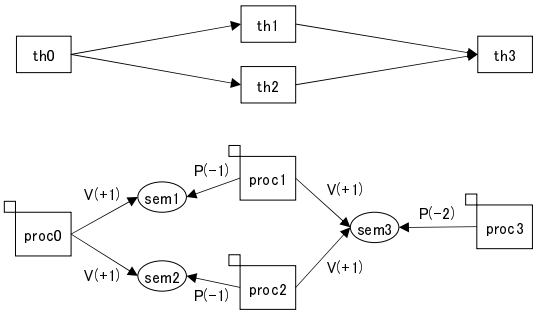

図 7 マイクロスレッドと評価用シミュレータの対応関係

4. 評価用シミュレータ

スケジューリングオーバヘッド低減機構の効果を評価するため，制御によりマイクロスレッドの動作がどのように変化したかを観測できる必要がある．本章では，マイクロスレッドの動作を観測するための評価用シミュレータについて述べる．

4.1 評価用シミュレータ概要

スケジューリングオーバヘッド低減機構を評価するにあたって，以下の課題がある．

(1) Fuce プロセッサは現在開発中であるため，Fuce プロセッサを搭載したマシンでの評価はできない．

(2) Fuce プロセッサの仕様として，マイクロスレッドは H/W 側で管理されるため，ソフトウェアからマイクロスレッドの動作を観測することができない．

従って，評価にはマイクロスレッドの動作をシミュレートするシミュレータを用いる．この評価用シミュレータでは，マイクロスレッドの動作を UNIX 環境上のプロセスでシミュレートする．具体的には,「マイクロスレッド」を「UNIX のプロセス」に対応させ,「マイクロスレッド間の継続による同期制御」を「プロセス間のセマフォによる同期制御」によって実現する．このようにして，Fuce 上のマイクロスレッドを UNIX 環境上のプロセスとして動作させる．**図 7** にマイクロスレッドと評価用シミュレータの対応関係を示す.

4.2 マイクロスレッド生成

Fuce では，多数のマイクロスレッドが頻繁に生成消滅する．評価用シミュレータでは，マイクロスレッドの生成は，マイクロスレッドの動作をシミュレートするプロセスの生成に対応付けられる．この処理の流れを**図 8** に示す.

評価用シミュレータでは，プロセスを生成するとき

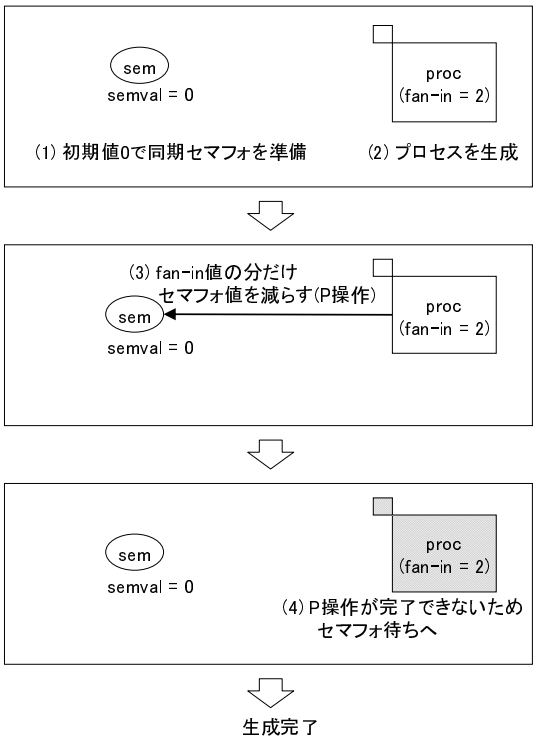

図 8 マイクロスレッド生成の流れ

に，同期を制御するためのセマフォも同時に生成する．そして，マイクロスレッドの動作をシミュレートするプロセスは，このセマフォの待ち状態にする．これにより，マイクロスレッドの動作をシミュレートするプロセスは，継続待ちと同様の状態になる．

4.3 継　　続

評価用シミュレータにおいて，マイクロスレッドの動作をシミュレートするプロセス間の継続による同期は係数セマフォを用いて実現する．4.2 節で述べたように，マイクロスレッドの動作をシミュレートするプロセスは，生成時にこのセマフォの待ち状態になっている．マイクロスレッドの動作をシミュレートするプロセスへの継続は，同期制御セマフォに対する V 操作で行う．**図 9** に継続による同期の流れを示す.

評価用シミュレータのセマフォ値は，Fuce における ACM の fan-in 値に対応付けられる．このことから，ACM の特定エントリに登録されているマイクロスレッドへの継続は，このエントリに対応付けた同期制御セマフォへの V 操作として実現する.

また，スケジューリングオーバヘッド低減機構の fan-in 値を増減させるという方法は，同期制御セマフォのセマフォ値を増減させることで対応する.

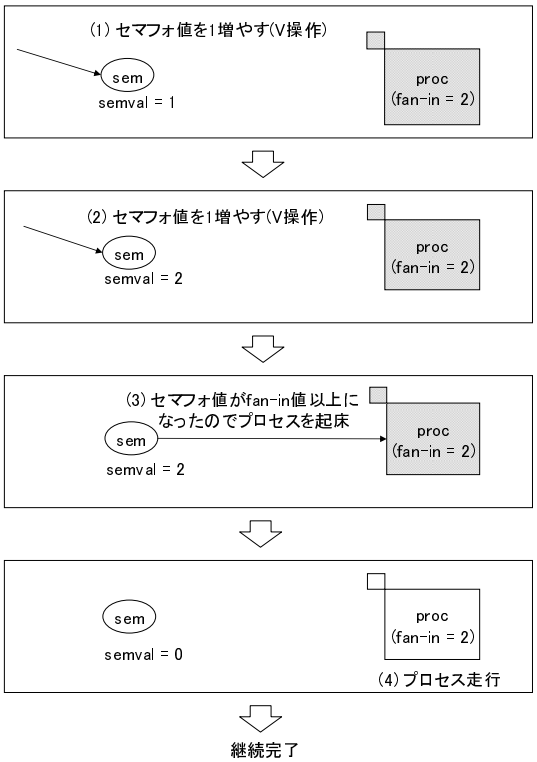

図 9 継続による同期の流れ

4.4 動作観測

UNIX 環境でマイクロスレッドの動作をプロセスにシミュレートさせ，プロセスを複数生成しても，シングルプロセッサでは同時に 1 つのプロセスしか走行しない．シングルプロセッサ機を用いて複数個の実行ユニットを持つ Fuce をシミュレートし，走行状態を観測するためには，何らかの方法で本来同時に実行されるべきプロセスの数を測定しなければならない．そこで本評価用シミュレータでは，RUN 状態のプロセスに加え READY 状態のプロセスも同時走行されているとして，RUN 状態と READY 状態のプロセス数の合計をマイクロスレッドの同時走行数とした．従って，評価用シミュレータでは一定周期で UNIX OS のスケジューラの READY キューの長さを観測する．

これにより，3 章で述べた $N(t)$ の値を取得することができる．そして，この $N(t)$ の値から他の評価尺度の値を計算によって求める．この評価用シミュレータを Linux（Kernel 2.4.7）上に実装した．

5. 評　　価

本章では，3 章で述べた評価尺度と 4 章で述べた評価用シミュレータを用いて，スケジューリングオーバヘッド低減機構の評価を行う．ここで機構がマイクロスレッドの制御を行う際に，各 ACM エントリの fan-in 値をどのような方針に基づき操作するかが制御結果に大きく影響すると考えられる．このため，fan-in 値を操作する際の方針とその方針に従った制御法をいくつか検討し，評価を行う．

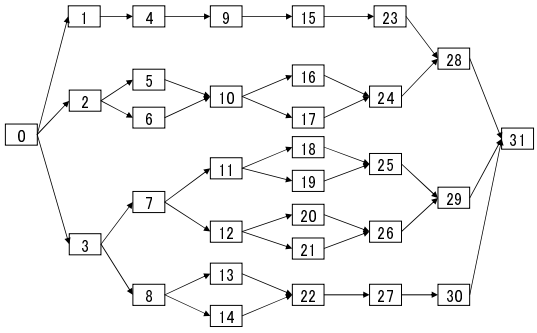

図 10 評価対象のマイクロスレッドの依存関係

5.1 評価条件

ここでは，基本的な評価条件を述べる．評価は，**図 10** に示す依存関係にあるマイクロスレッドに対して行う．各マイクロスレッドの走行時間は，実時間で 1 秒間とする．また，図 10 の各マイクロスレッドは，0 から 31 までこの順序通りに ACM のエントリに登録する．つまり，マイクロスレッド 0 は，ACM の 0 番目のエントリに登録され，マイクロスレッド 1 は，ACM の 1 番目のエントリに登録される．

ここで，図 10 に示すマイクロスレッド群から構成されるプロセスは，優先度の低いものと仮定し，本来の動作よりマイクロスレッドの同時走行数を抑えて動作させる制御方法を検討し，評価する．具体的には，N_p の値を抑えることを主な目的とする．

また，制御を行わない場合，図 10 のマイクロスレッド群の動作における評価尺度の値は，以下のようになる．

(1)　$N_p = 8$
(2)　$N_v = 5.25$
(3)　$T_a = 8.0$
(4)　$T_d = 0.0$
(5)　$T_r = 8.0$

5.2 ランダム法

スケジューリングオーバヘッド低減機構は，マイクロスレッドの走行状態を厳密に管理するのではなく，マイクロスレッド間の同期を制御する同期値だけに手を加える．このとき，どのマイクロスレッドの同期値を操作するかという選択は，何らかの確率に従って行う．そこで，同期値を操作するマイクロスレッドの選択は，完全にランダムによる方法が考えられる．**図 11**

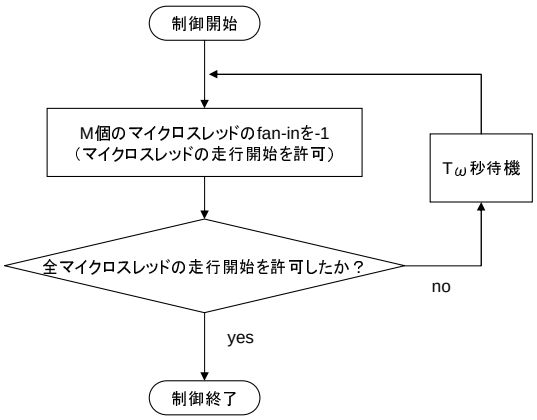

図 11 ランダム法の処理の流れ

に同期値を操作するマイクロスレッドをランダムに選択することにより，本来の動作よりマイクロスレッドの同時走行数を抑える制御法を示す．この制御法は，動作を止めるために継続が挿入されているマイクロスレッドの中の M 個に T_ω 秒ごとに継続を与える（同期値 -1）ことで，マイクロスレッドを少しずつ動作させようとするものである．このとき，どのマイクロスレッドに継続を与えるかをランダムに選択する．ただし，既に継続を与えたマイクロスレッドは，選択しない．また，制御開始時には，全マイクロスレッドに既に継続が挿入（同期値 +1）されている．この制御法をランダム法と呼ぶことにする．

表 1 に $T_\omega = 1$，$M = 1$ として評価した結果を示す．また，評価のための試行回数は，1000 回である．ランダム法は，N_p の値は平均で 5.15 であり，制御しない場合の N_p の値である 8 と比較して，マイクロスレッドの同時走行数を抑えることができている．しかし，N_p の最大値が 8 であることから，マイクロスレッドの同時走行数を抑えることができていない場合もある．また，ランダム法では，T_d の値に大きな影響を与える．これは，プロセスの実行開始までの時間が大きく変化することを意味する．つまり，ランダム法では，プロセスの実行開始を大きく遅延させるものの，肝心のマイクロスレッドの同時走行数を抑える効果は低い．従って，有効な制御ができているとは言えない．

表 1 ランダム法による評価結果 ($(T_\omega, \mathrm{M})=(1.0, 1)$)

	最大値	最小値	平均	分散
N_p	8.00	2.00	5.15	2.64
N_v	5.80	0.30	2.20	2.30
T_a	38.80	32.70	35.85	1.60
T_d	31.00	0.00	15.11	84.51
T_r	37.70	7.70	20.73	80.86

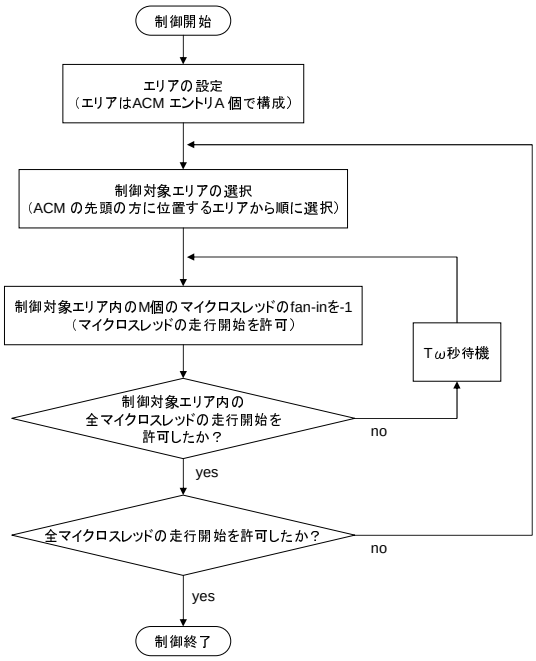

図 12 エリア分割法の処理の流れ

5.3 エリア分割法

次に，同期値を操作する ACM エントリを完全にランダムに選択するのではなく，プログラムの構造を考慮して選択する制御法を考える．具体的には，マイクロスレッドの実行順序に着目する．Fuce では，マイクロスレッドが生成されると ACM に登録され，管理される．一般的な手続き型プログラミング言語によるプログラム記述を仮定した場合，マイクロスレッドは実行される順序に近い順序で ACM エントリの先頭から登録されるのが自然である．従って，最初は先頭の方に位置する ACM エントリの同期値を操作し，時間の経過とともに後方に位置する ACM エントリの同期値を操作する方法が考えられる．**図 12** にこの制御法の具体的な処理の流れを示す．

この制御法は，同期値を操作する ACM エントリを特定のエントリに限定するため，まず A 個の連続した ACM エントリから構成されるエリアを設定する．この様子を**図 13** に示す．図 10 のマイクロスレッドでは，マイクロスレッドは 32 個の ACM エントリに登録されている．図 13 では，これを $A = 4$ として 8 個のエリアに分割している．このとき，互いのエリアは，重ならないようにする．

同期値を制御する ACM エントリは，制御対象エリア内の ACM エントリから選択する．この制御対象エリアは，ACM の先頭に位置するものから順に選択する．エリア内での同期値を制御する ACM エントリの

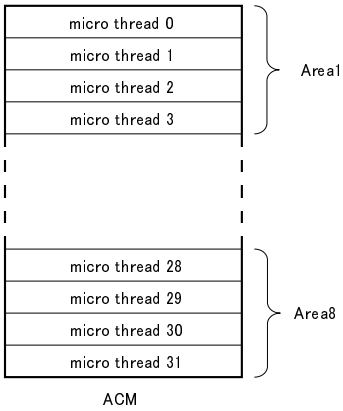

図 13 ACM のエリア分割 (A=4)

選択方法は，ランダム法に従う．そして，制御対象エリア内の全マイクロスレッドに継続を与えた場合に，制御対象エリアを次のエリアにする．また，制御開始時には，全マイクロスレッドに既に継続が挿入（同期値 +1）されている．この制御法をエリア分割法と呼ぶことにする．

表 2 に $T_\omega = 1$，$M = 1$，$A = 4$ として評価した結果を示す．また，評価のための試行回数は，1000 回である．制御しない場合，N_p の値は 8 であるが，エリア分割法により制御を行った場合，N_p の値は最大でも 3 である．また，N_v の値が小さいことから，制御を行っている間，マイクロスレッドの同時走行数をほぼ一定に保つことができている．ランダム法と比較して T_d の値が小さいことから，制御を開始してからプロセスが実際に実行されるまでの遅延時間が短い．従って，このエリア分割法では，マイクロスレッドの同時走行数を抑えつつ，プロセスを走行させることができている．

6. おわりに

本稿では，全てのマイクロスレッドの動作を逐一把握して制御するのではなく，一定の割合で同期表の中の同期値のみを操作し，マイクロスレッドの継続動作を抑制することで全体の実行速度を制御するスケジューリングオーバヘッド低減機構の評価を行った．

評価により，機構はマイクロスレッドの制御が可能であることがわかった．ただし，有効な制御を行うためには，機構はマイクロスレッドの実行順序といったプログラムの構造を考慮する必要がある．

また，本稿で行った評価条件では，ループや条件分岐の動作をするマイクロスレッドが含まれていない．また，実行中に新しいマイクロスレッドが生成されることもなく，評価は非常に基本的なものである．今後は，より実際のプログラムに則した検討を行う．

表 2 エリア分割法による評価結果 ((T_ω, M, A)=(1.0, 1, 4))

	最大値	最小値	平均	分散
N_p	3.00	1.00	2.21	0.69
N_v	0.21	0.00	0.10	0.0048
T_a	32.80	31.80	32.55	0.18
T_d	3.10	0.00	1.43	1.23
T_r	32.80	28.80	31.11	1.41

謝辞 本研究は，文部科学省科学研究費補助金・基盤研究（A）(2)「細粒度マルチスレッド処理原理による並列分散処理カーネルウェアの研究」(課題番号:15200002）による．

参 考 文 献

1) Lo, J. L., Eggers, S. J., Emer, J. S., Levy, H.M., Stamm, R. L. and Tullsen, D.M.,“ Converting Thread-Level Parallelism via Simultaneous Multithreading, ”ACM Transaction on Computer System, Vol.15, No.3, pp.322-354, 1997.
2) Marr, D.T., Binns, F., Hill, D.L., Hinton, G., Koufaty, D.A., Miller, J.A. and Upton, M. “ Hyper-threading technology architecture and microarchitecture: a hyperhtext history, ”Intel Technology J. 6,1 2002(online journal).
3) Amamiya, M., Taniguchi, H. and Matsuzaki, T.,“ An Architecture of Fusing Communication and Execution for Global Distributed Processing, ”Parallell Processing Letters, Vol.11, No.1, pp7-24, 2001.
4) 雨宮聡史, 松崎隆哲, 雨宮真人, “ 排他実行マルチスレッド実行モデルに基づくオンチップ・マルチプロセッサの設計, ”情報処理学会研究報告, Vol.2003, No.119, pp.51-56, 2003.
5) 乃村能成, 雨宮聡史, 日下部茂, 谷口秀夫, 雨宮真人, “ 細粒度マルチスレッド環境でのスケジューリングオーバヘッド低減機構, ”情報処理学会研究報告, Vol.2004, No.63, pp.129-134, 2004.
6) 日下部茂, 富安洋史, 村上和彰, 谷口秀夫, 雨宮真人, “ 並列分散オペレーティングシステム CEFOS(Communication-Execution Fusion OS), ”信学技報, Vol.99, No.251, pp.25-32 1999.
7) 福冨和弘, 乃村能成, 日下部茂, 谷口秀夫, 雨宮真人,“ マルチスレッド実行機構を考慮したプログラム実行制御法, ”情報処理学会研究報告, Vol.2004, No.63, pp.135-140, 2004.