2007－OS－105 (3)
2007／4／5

スレッドの実動作情報に基づく細粒度マルチスレッド制御法

乃村能成† 小川泰彦† 日下部茂††
谷口秀夫† 雨宮真人††

細粒度マルチスレッドアーキテクチャのプロセッサにおけるOSでは，OSが全てのスレッドの実行を管理することは，非効率的である．そこで，OSのスケジューリングオーバヘッドを低減するために，スレッド群をOSが管理すべき物ととそうではない物にクラス分けすることを考える．本稿では，スレッドのクラス分け手法として，スレッド動作中の情報に基づいた動的な制御法について述べる．

A Process Scheduling Mechanism for Fine Grain Multithread Architecture Based on Actual Thread Behavior

YOSHINARI NOMURA,† YASUHIKO OGAWA,† SHIGERU KUSAKABE,††
HIDEO TANIGUCHI† and MAKOTO AMAMIYA††

Microprocessors based on the fine grain multithread architecture are characterized by its thread level parallelism. It essentially generates a large number of threads in a second. Therefore, it is very difficult to schedule all the threads effectively by a conventional software process scheduler. In this paper, we propose a cost-effective process scheduling method suitable for the fine grain multithread architecture based on actual thread behavior.

1. はじめに

近年，SMT（Simultaneous Multi-Threading）やCMP（Chip Multi-Processor）といったプロセッサが提案[1]され，実用化[2]されている．また，命令レベルでの並列性の抽出には限界があるとし，データフローモデルを基盤にしてスレッドの並列実行を追及する手法も提案[3]されている．Fuce[4]は，この考えに基づき開発されている，スレッド実行ユニットを複数搭載したCMPの一種である．

Fuceは，複数スレッドを並列に処理できるが，通常のSMTプロセッサとは，以下の点で異なる．

(1) 継続と呼ばれるデータフローモデルを基盤とした同期手法によって，スレッド間でデータを受渡し，処理を実現する．
(2) 小さいスレッドが大量にユーザレベルで生成される．
(3) スレッドの実行管理はハードウェアで行われる．
(4) スレッドは，短時間で走り切るが，終了まで横取りされない．

これらFuceのハードウェアに管理されるスレッドを以降，マイクロスレッドと呼ぶ．

Fuce上でのOSを考えた場合，OSは，マイクロスレッドの集合体であるプロセスを管理し，スケジューリングすることが要求される．しかしながら，OSが大量のマイクロスレッドに対して個々の走行状態を管理することは，オーバヘッドの観点から現実的ではない．しかも，マイクロスレッドの実行管理はハードウェアで行われるため，OSが実行途中を横取りできない．

そこで，スレッド間の同期を制御する同期カウンタ値をある確率に基づいてOSが増減することで，スレッド間の同期を意図的に遅延させ，マイクロスレッドの動作を調整する基本手法が提案[5]されている．また，基本手法の具体例を提案し，それらの実現性についての基本評価も示されている[6]．

しかしながら，既存の提案手法は，制御を低オーバヘッドで実現できるものの，動的に生成されたマイクロスレッドの制御に対応できないなど，いくつかの問題点がある．

本稿では，まず基本手法の問題点を明らかにする．

† 岡山大学大学院自然科学研究科
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

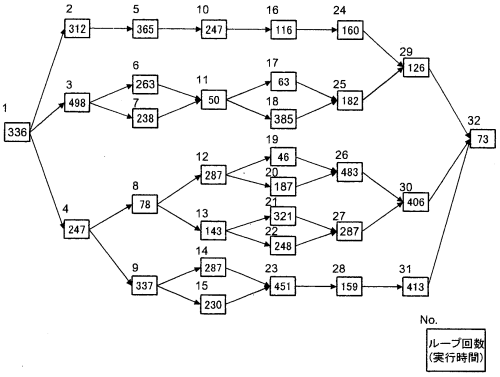


図 1 マイクロスレッドの依存関係例

次に，それらに対処するための手法を提案する．具体的には，個々のマイクロスレッドをある確率で本来の継続先に継続させずに，OS 本体に継続をさせ，OS が継続を中継する．これによって本来の継続に遅延を発生させる．さらに，提案手法に対して，これまでの手法と同様の評価を行い，本手法の特徴について議論する．

2. 既存手法とその問題点

本章では，Fuce 上においてソフトウェアによってマイクロスレッドを制御する既存手法である静的制御方式[5] のエリア分割法[6] について簡単に説明し，制御法の基本機構とその課題について述べる．

2.1 基本機構

図 1 にマイクロスレッドの依存関係の例を示し，例に基づいてこれらのスレッドの同時実行数を制御する基本機構について説明する．図中，マイクロスレッドを表す枠内の数字は，マイクロスレッドの実行時間を表す．具体的には，アセンブラで記述された命令の消費するクロック数を表している．

図 1 の 32 個からなるマイクロスレッドは，一旦走行をはじめると，個々の矢印で示された継続関係に従ってハードウェアにより動作開始され，横取りされることはない．したがって，マイクロスレッドの動作を OS が制御するためには，マイクロスレッド間の継続関係を操作することによって継続を意図的に遅延させるしかない．そこで，基本手法では，**図 2** で示すように，OS からの継続をコンパイラによってあらかじめ挿入させておく．それによって個々のマイクロスレッドが OS からの継続を得なければ，実行を開始できないようにする．

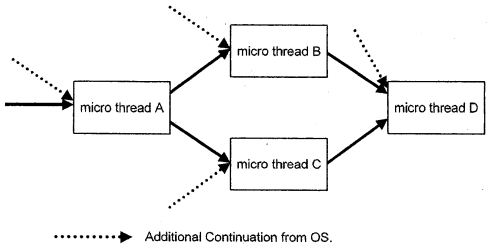


図 2 継続の追加による制御

本機構の制御の善し悪しは，OS がマイクロスレッドに継続を発行する順序と周期によって決まる．例えば図 1 では，1 から 32 の順番で継続を発行した場合は，マイクロスレッドの実行数を効率よく制御できるが，逆順で発行した場合は，制御の意味がない．したがって，実行プログラム中におけるマイクロスレッド間の継続関係をあらかじめ知ることが望ましいが，これらを正確に知ることは難しい．

2.2 エリア分割法

エリア分割法[6] は，プログラム中のマイクロスレッド宣言順序と実行時の継続発生順序には強い相関があると仮定し，それに基づいて OS の継続発行順序を決定する手法である．具体的には，プログラム中の宣言順序として，ACM と呼ばれるスレッド管理表への登録順序を採用する．ACM 表をエリアに分割して，先頭から OS が継続発行先スレッドを決定する．以下にエリア分割法による制御の概要を示す．

(1) 制御開始前に全マイクロスレッドに OS からの継続を挿入しておく．
(2) 継続を制御する ACM 表を N 個毎にエリアに分割し，ACM の先頭に位置するものから順に選択する．
(3) 選択したエリア内の全マイクロスレッドに継続を与える．
(4) 選択エリアを ACM の並び順で次のエリアにする．
(5) 上記を全てのエリアについて実行する．

この手法によりマイクロスレッドを制御することができる[6]．この手法は制御が簡単でオーバヘッドも少ないといった利点がある一方で，以下の課題がある．

(課題 1) 制御の確実性の確保

エリア分割法ではマイクロスレッドは ACM エントリの先頭の方から実行順序に近い順番で登録されているという前提に基づいているが，そうではない場合，有効な制御ができない．

(課題 2) マイクロスレッドの動的生成への対応
制御対象のマイクロスレッドは，制御開始時に全てACMに登録されている必要がある．そのため，制御の途中でマイクロスレッドを動的に生成することはできない．

(課題 3) スレッドループへの対応
マイクロスレッドは，ハードウェアによって実行時間が短かく制限されているため，ループ処理は，自分自身に継続を発行することによって実現される．このため，このループするマイクロスレッドの同期を解決させるためには，ループの度にOSからの継続を必要とするが，エリア分割法では，1つのスレッドには1度しかOSからの継続を発行しないため，対処が必要となる．

3. スレッド実動作情報に基づく制御法

エリア分割法の課題を解決するために，マイクロスレッドの実行時に動的に継続先を変更する手法を提案する．つまり，本手法は，個々のマイクロスレッドをある確率で本来の継続先に継続させずに，OS本体に継続をさせ，OSが継続を中継する．これによって継続間に遅延を発生させ，全体の実行を制御する．

本手法では，ハードウェアが管理するACM表とは別に，OSが管理するデータ構造として，マイクロスレッドIDプールを導入する．制御に必要なマイクロスレッドの情報（マイクロスレッドのID）は，全てこのマイクロスレッドIDプールに登録する．

マイクロスレッドIDプールを用いた制御の処理流れを**図3**に示し，以下に述べる．

(処理 1) 継続元マイクロスレッド（図中src）は，自身が制御対象であることを示す制御フラグを確認する．制御フラグのon/offはOSによってある確率に基づいて行われるが，その手法はここでは検討しない．

(処理 2) マイクロスレッドsrcは，制御フラグに従って継続先マイクロスレッドを変える．制御フラグが無効の場合，マイクロスレッドは，本来の継続先（図中dst）に継続する．制御フラグが有効な場合，srcは，本来の継続先に継続せず，OS内の中継用マイクロスレッド（図中register）に継続する．registerへの継続の際は，本来の継続先マイクロスレッドID（図3中0x000000001）を引数とする．

(処理 3) OS内のregisterは，受け取った継続先マイクロスレッドIDをマイクロスレッドIDプールに登録する．登録エントリの位置は，図中register_addrで示している．registerは，複数のマイクロスレッドから継続され得るため，排他制御される．

(処理 4) contスレッドは，timerスレッドにより一定周期ごとに継続される．contスレッドは，マイクロスレッドIDプールから継続すべきマイクロスレッドのIDを取得する．取得するIDのエントリの位置は，cont_addrで示す．

(処理 5) contスレッドは，マイクロスレッドIDプールから取得したIDのマイクロスレッドに継続する．これにより，継続先のマイクロスレッドは同期が解決する．

本手法は，以下の利点を持つ．

(利点 1) マイクロスレッドIDプールには，マイクロスレッドの継続の発行順序，つまり実行順序に近い順序でマイクロスレッドのIDが登録されていく．このため，マイクロスレッドIDプールに登録された順にそのマイクロスレッドに継続を与えることで，ほぼマイクロスレッドの実行順序に従いマイクロスレッドを走行させることができる．

(利点 2) マイクロスレッド側に継続すべきマイクロスレッドのIDを通知させるため，OS側でハードウェアの機能であるACM表を意識する必要がない．このため，マイクロスレッドが複数のACMブロックに分散して登録されたり，動的に生成されたりしても制御できる．

(利点 3) ループを行うマイクロスレッドを特別扱いする必要がない．ループする度にマイクロスレッドIDプールにIDを登録させることで，ループなど，繰り返し継続を必要とするマイクロスレッドも他のマイクロスレッドと同様に扱うことができる．

欠点としては，オーバヘッドが増加する，各スレッドによる制御フラグの取り扱いが必要となることが挙げられる．

以降，本手法をIDプール法と呼び，各部の詳細について述べる．

継続部 ：
アプリケーションを構成するマイクロスレッド(図3中のsrcやdst)の動作を従来のものから変更する必要がある．具体的には，継続のためのcont命令を拡張する．拡張前と拡張後のアセンブリコードを**図4**に示す．
拡張後のマイクロスレッドは，命令列の先頭部分で制御フラグとregisterスレッドのIDを読み込む．さらに，継続を行う度に制御フラグをチェッ

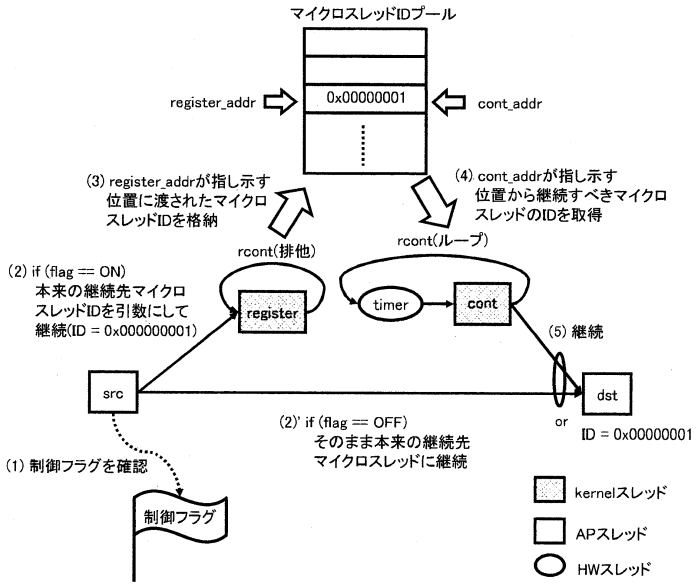


図 3 制御法の構成と処理流れ

クし，制御フラグの状態によって継続先を変える．register スレッドに継続する場合は，まず register スレッドを tsacm 命令によってロックする．これは，register スレッドを排他的に実行するためである．

マイクロスレッド ID プール ：
マイクロスレッド ID プールは，マイクロスレッド ID を保存する領域であり，OS が確保する．ID 登録エントリの位置は register_addr で示し，ID 取得エントリの位置は cont_addr で示す．

register スレッド ：
register スレッドは，まずマイクロスレッド ID プールに空きがあるか調べる．空きがない場合は，register スレッドは，マイクロスレッド ID プールへの ID 登録を断念し，渡された ID(本来の継続先) のマイクロスレッドに継続する（図 3 の例では，dst に継続する）．ID プールに空きがある場合，register スレッドは，マイクロスレッド ID プールに渡された ID を登録する（図 3 では，dst の ID をプールに登録する）．

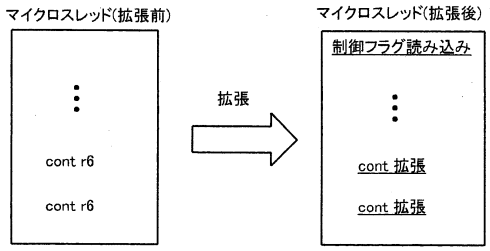
cont スレッド ：
cont スレッドは，スケジューリングパラメータ M（継続を与えるマイクロスレッドの個数）で指定されたマイクロスレッドの数だけ，マイクロスレッド ID プールからマイクロスレッド ID を取り出し，そのマイクロスレッドに継続を与える．cont スレッドが 1 度の走行で継続を与えるマイクロスレッドは 1 つである．したがって，$M = 3$ ならば，cont スレッドはスレッドレベルで 3 回ループする．M 回ループすると，cont スレッドはタイマスレッドに継続する．ただし，マイクロスレッド ID プールが空の場合，cont スレッドは無条件にタイマスレッドに継続する．

timer スレッド ：
timer スレッドは，指定された時間後に，指定されたマイクロスレッドへ継続を行う．

このように ID プール法ではマイクロスレッドの実動作情報として個々のマイクロスレッドの継続先マイクロスレッド ID を取得する．このマイクロスレッド ID は，マイクロスレッドが継続を発行する度に動的に収集される．OS は，マイクロスレッド ID プールに溜まった情報に基づき次に継続を与えるマイクロスレッドを決定し，走行させることで制御を実現している．

4. 評　　価

本章では，細粒度マルチスレッド制御法を評価するための評価尺度を示し，その評価尺度を用いて制御法を評価した結果と考察を述べる．



制御フラグ読み込み

```
lw      r29, 0(r0)    # r29 = 制御フラグ
lw      r30, 8(r0)    # r30 = register スレッドのID
```

cont 拡張

```
beq     r29, r0, L    # 制御フラグをチェック
tsacm   r28, r30      # thRegisterを排他
bne     r28, r0, L    # 排他できなかった場合は登録を
                      # 断念(本来の継続先へ継続)
sw      r6, 4(r0)     # 引数(登録スレッドID)
cont    r30           # thRegisterに継続
j       L
L:
cont    r6            # thDstに継続
L:
```

図 4　マイクロスレッドの拡張

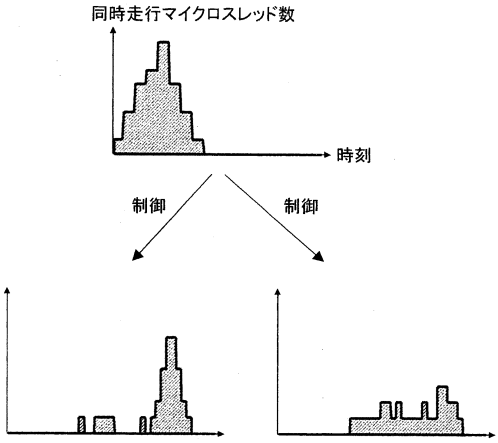


図 5　同時走行マイクロスレッド動作の変化

4.1 評価尺度

細粒度マルチスレッド制御法では，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，マイクロスレッド間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，制御法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．評価尺度の詳細は，文献 6) を参照のこと．

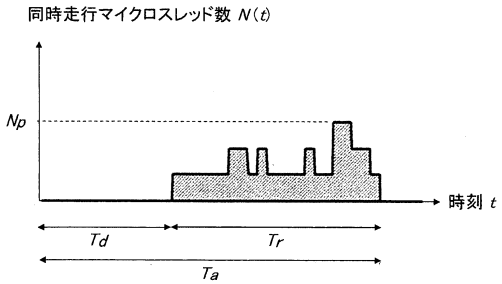


図 6　評価尺度

$N(t)$　：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．

N_p：　$N(t)$ の最大値．つまり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．

N_a　：$N(t)$ の平均．

N_v　：$N(t)$ の分散．

T_d　：制御開始後にプロセスが走行を始めるまでの遅延時間．

T_r　：プロセスの走行時間．

T_a　：制御開始からプロセスが終了までの時間（$T_d + T_r$）．

overhead　：制御のオーバヘッド．つまり制御のために費やしたクロック数が全体のクロック数に占める割合．

4.2 評価条件

評価は，図 1 に示す依存関係のマイクロスレッドに対して行う．制御のパラメータ，およびスレッド実行ユニットの数は以下のように設定した．

(1)　$M = 1, 2, 4$

意味：一度に継続を与えるマイクロスレッド数

(2)　$T_\omega = 125, 250, 375, 500$

意味：継続を与える周期

単位は，マイクロスレッドの実行時間と同じクロック時間である．

(3) スレッド実行ユニット数 $= 8$

ここで，評価対象のマイクロスレッド群は，優先度の低いプロセスに属すると仮定し，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走行させることを目的とする．具体的には，N_a および N_p の値を抑えることを主な目的とする．

評価環境は，ハードウェア記述言語 VHDL を用いて作成された Fuce プロセッサシミュレータを利用した．これらの条件に基づき提案手法を評価した．

4.3 評価結果

評価結果を**表 1** にまとめる．

また，各パラメータ M, T_ω における時刻 t と時刻 t でのマイクロスレッド走行数 $N(t)$ との関係を**図 9** から**図 11** に抜粋して示す．

また，$N(t)$ は，アプリケーションを構成するマイクロスレッド以外にも register スレッド，cont スレッドおよび timer スレッドを走行数に含む．ただし，制御なしの場合（図 9）と制御フラグ OFF の場合は，register スレッド，cont スレッドおよび timer スレッドは走行しないため，純粋に AP を構成するマイクロスレッドの走行数が $N(t)$ となる．

4.4 考　　察

4.4.1 スレッド実行ユニット占有度

スレッド実行ユニット占有度の調整については，多くの場合，N_p および N_a の値が低下し，マイクロスレッド群の制御ができている．ただし，$M = 4, T_\omega = 125$ の場合などは，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．これは，OS から AP のマイクロスレッドに継続を与える周期が短か過ぎるためであると考えられる．そこで，マイクロスレッド群を制御するための適切な M および T_ω の値について考察する．

(1) T_ω の設定

M の値が同一の場合，T_ω の値が大きいほど N_p および N_a の値が小さくなっている．また，T_ω の値が同一の場合，M の値が小さいほど N_p および N_a の値が小さくなっている．つまり，T_ω の値が大きいほど，また，M の値が小さいほどスレッド実行ユニットの占有度を抑えることができるといえる．

ただし，T_ω の値はマイクロスレッド制御の周期であるため，T_ω の値を大きくし過ぎると制御の応答性に問題が生じると考えられる．また，M と T_ω という 2 つの可変なパラメータを用意し，これら 2 つのパラメータがそれぞれ制御に与える影響を考慮して，個々のパラメータを設定することは煩雑である．

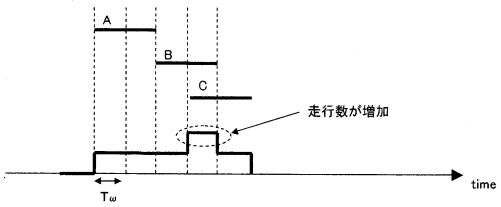


図 7 マイクロスレッド同時走行数増加

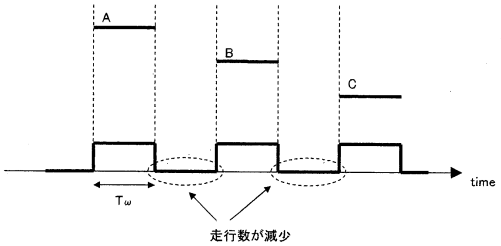


図 8 マイクロスレッド同時走行数減少

そこで，T_ω の値は固定にする．具体的には，制御対象マイクロスレッドの走行時間の平均値とする．これは，以下の理由による．

T_ω の値を制御対象マイクロスレッドの走行時間の平均値以下にした場合，マイクロスレッドの同時走行数が増加する傾向にある．これは，先に走行を開始したマイクロスレッドが走行を終了する前に，OS から継続を与えられた次のマイクロスレッドが走行を開始するためである．この様子を**図 7** に示す．逆に，T_ω の値を制御対象マイクロスレッドの走行時間の平均値以上にした場合，マイクロスレッドの同時走行数が減少する傾向にある．これは，先に走行を開始したマイクロスレッドが走行を終了してから，OS から継続を与えられた次のマイクロスレッドが走行を開始するまで，マイクロスレッドが走行しない時間ができるためである．この様子を**図 8** に示す．このため，最も同時走行マイクロスレッド数の増加と減少のバランスがとれた値は，制御対象マイクロスレッドの走行時間の平均値であると考えられる．

(2) M の設定

制御パラメータ M と T_ω のうち，T_ω は制御対象マイクロスレッドの走行時間の平均値で固定される．したがって，プロセスのスレッド実行ユニッ

表 1　評価結果

M	T_ω	N_a	N_v	N_p	T_d	T_r	T_a	overhead
制御なし		3.45	4.53	8	0	28,740	28,740	0.0%
制御フラグ OFF		3.45	4.53	8	0	28,822	28,822	0.22%
1	125	2.41	0.85	5	23	70,486	70,509	2.74%
1	250	1.77	0.42	5	23	128,873	128,896	2.66%
1	375	1.51	0.29	4	23	191,880	191,903	2.68%
1	500	1.39	0.24	4	23	254,880	254,903	2.68%
2	125	3.14	1.76	8	23	46,405	46,428	2.84%
2	250	2.33	0.90	6	23	74,715	74,738	2.72%
2	375	1.93	0.57	5	23	106,194	106,217	2.74%
2	500	1.74	0.51	4	23	134,640	134,663	2.70%
4	125	3.58	4.18	8	23	38,203	38,226	2.82%
4	250	3.08	2.37	8	23	47,663	47,686	2.80%
4	375	2.51	1.76	6	23	65,596	65,619	2.73%
4	500	2.33	1.70	7	23	74,474	74,497	2.62%

ト占有度は，M の値を変化させて調整する．

ここで，M の値としてプロセスの優先度に応じた値を用いる方法が考えられる．評価結果から，M の値が小さいほどスレッド実行ユニット占有度が抑えられる．このため，優先度の低いプロセスは，スレッド実行ユニット占有度が抑えられることになる．

図 1 のマイクロスレッド群の走行時間の平均値は，251.59 である．そこで，制御パラメータ $T_\omega = 250$ で制御した場合に注目する．$T_\omega = 250, M = 1$ の場合，$N_a = 1.77, N_p = 5$ である．制御なしの場合の $N_a = 3.45, N_p = 8$ と比較して，値が低く抑えられている．これは，スレッド実行ユニット占有度を低く抑えられていることを意味する．$N_v = 0.42$ と N_v の値が小さいことから，制御の安定性もよいといえる．

また，$M = 2, M = 4$ の場合にも着目すると，M の値を大きくするほど，N_a, N_p の値が大きくなっている．これは，M（プロセスの優先度）が大きいほど，スレッド実行ユニット占有度を大きくすることができるといえる．

したがって，T_ω をマイクロスレッド走行時間の平均値とし，M をプロセスの優先度とすることでマイクロスレッド群の制御ができると考えられる．

4.4.2 制御のオーバヘッド

制御のオーバヘッドについては，制御フラグが OFF の場合，オーバヘッドの値は 0.22% と非常に小さい．また，他の評価尺度の値は，制御なしの場合とほぼ同じ値となる．したがって，制御フラグが OFF の場合，マイクロスレッドは制御の影響をほぼ受けることなくフルスピードに近い状態で走行することができる．

制御を行った場合でも，オーバヘッドは 3% 以内に収まっているため，十分実用的に制御できると考えられる．

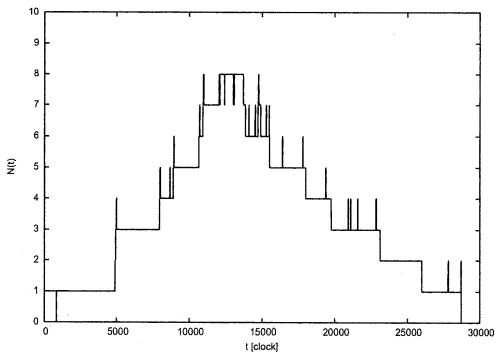


図 9　同時走行マイクロスレッド数（制御なし）

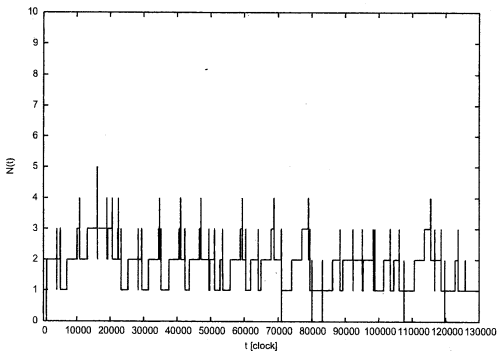


図 10　同時走行マイクロスレッド数（M = 1, T_ω = 250）

5. おわりに

本稿では，データフローモデルに基づきスレッドの

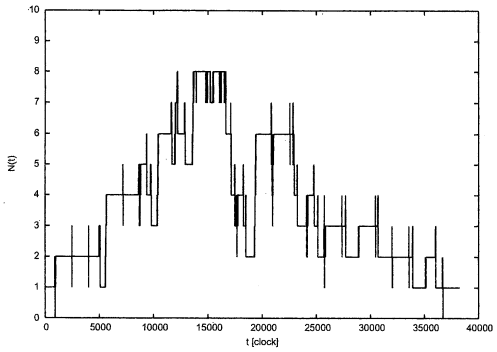


図 11 同時走行マイクロスレッド数（M = 4, T_ω = 125）

並列実行を追求した Fuce アーキテクチャ上で，OS がプロセススケジューリングをするために必要となる細粒度マルチスレッド制御法を述べた．まず，Fuce におけるスレッド並列実行モデルと Fuce アーキテクチャについて述べた．次に，細粒度マルチスレッド制御法の基本機構を述べた．そして，この基本機構に基づく，より具体的な制御法の一例としてエリア分割法を示し，エリア分割法の持つ問題に対処した ID プール法を提案した．

ID プール法では，マイクロスレッドの実動作情報として個々のマイクロスレッドの継続先マイクロスレッド ID をマイクロスレッド ID プールに登録する．マイクロスレッド ID は，マイクロスレッドが継続を発行する度に動的に収集される．OS は，マイクロスレッド ID プールに収集された情報に基づき次に継続を与えるマイクロスレッドを決定する．

また，提案した制御法の評価を行った．評価結果から，制御パラメータ T_ω（マイクロスレッドに継続を与える周期）および M（一度に継続を与えるマイクロスレッドの個数）の適切な値について考察した．提案した制御法は，T_ω を制御対象マイクロスレッド群の走行時間の平均値とし，M をプロセスの優先度に応じた値とすることにより，プロセスの優先度に基づくマイクロスレッドの制御が可能であることを示した．

残された課題として，複数のアプリケーションを同時走行させた環境での評価がある．

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) 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, "情報処理学会研究報告, Vol.2006, No.15, pp.47-53, 2006.