2008－OS－109 (15)
2008／8／7

継続概念によるH/Wスケジューラにおける S/Wスケジュール法

森山英明† 谷口秀夫† 乃村能成†

細粒度マルチスレッド実行環境における継続概念のH/Wスケジューラでは，スレッド間の依存関係である継続によって実行制御する．このような実行環境において，目的に応じたサービスの実行を実現するには，S/Wにより複数のスレッド群を実行制御する必要がある．また，多数の細粒度スレッドが頻繁に生成消滅を繰り返すため，スレッド実行制御のオーバヘッドを抑制することは重要である．

本稿では，スレッド間の依存関係である継続に着目し，継続流れを効率的にスケジュールするS/W制御法を提案する．提案する制御法により，オーバヘッドの低減が期待できる．

S/W Scheduling for H/W Scheduler Based on Continuation Model

Hideaki Moriyama ,† Hideo Taniguchi †
and Yoshinari Nomura†

In a fine grain multi-thread environment based on continuation model, H/W scheduler takes dependency among threads as continuation order. In such environment, execution of service requires scheduling of many thread-groups by S/W. In addition, reduction of scheduling overhead is important so that a large number of fine-grain threads repeat generation and extinction regularly.

In this paper, we focus on continuation that is dependency among threads, and show effective S/W scheduling method for reduction of scheduling overhead based on continuation flow.

1. はじめに

近年，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．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開発[3]されている．Fuce(FUsion of Communication and Execution) プロセッサは，この考えに基づき，スレッドの並列実行を追求したCMPの一種として開発[4]されている．

Fuceプロセッサは，以下の特徴を持つ．

(1) スレッド実行ユニット (TEU : Thread Execution Unit) を複数有する．

(2) 継続と呼ぶ同期手法により，スレッド間でデータを受け渡し，処理を実行する．

(3) ハードウェア (H/W) により，スレッドの実行スケジュール管理を行う．

(4) スレッドは，TEUを横取りされることなく処理終了まで実行される．

したがって，サービス実現を多数のスレッドで構成することにより，処理の並列化を促進でき，効率的なサービス提供が期待できる．一方，多数の細粒度スレッドが頻繁に生成消滅を繰り返すことになり，スレッド実行制御のオーバヘッドを抑制することは重要である．

一方，プロセッサ性能の向上に伴い，一つのプロセッサ上に複数のサービスを同時並行して実行することが求められる．この場合，各サービスの実行においては，システムの目的に応じた実行制御が必要である．例えば，「特定のサービスを優先して実行したい」，あるいは「各サービスを均等に実行したい」である．スレッドを実行制御の基本単位とするプロセッサでは，サービスは多数のスレッド (スレッド群) から構成される．このため，目的に応じたサービスの実行制御のためには，複数のスレッド群の実行制御が必要である．

Fuceプロセッサのようにスレッドの実行制御がH/W

† 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

により行われている場合，目的に応じたサービスの実行制御を行うには，スレッドの実行制御に対し，オーバヘッドが少ない方法でのS/Wの関与が重要である．つまり，オペレーティングシステム（OS）のプロセス（あるいはスレッド）の実行制御のように，プロセスひとつひとつの属性（優先度やプロセッサ使用量など）を管理し，その属性に基づいて実行を制御する方法では，オーバヘッドが大きく好ましくない．そこで，継続による同期でスレッド間の実行順序が関連づけられていることに着目し，この継続をS/Wで制御することにより，目的に応じたサービスの実行制御を行うことが考えられる．具体的には，継続する数（同期カウンタ値）を制御する方法[5)6)]，あるいは継続する先をS/Wスケジューラとする方法[7)]がある．

ここでは，S/Wスケジュール法として，スレッド間の実行順序を関連づけている継続をS/Wで制御する方法を述べる．具体的には，複数のスレッド群の実行制御における目的を示し，評価尺度を明らかにする．次に，S/Wスケジュール法として，同期カウンタ値を制御する方法，および継続する先をS/Wスケジューラとする方法について，その特徴と問題点を明らかにする．最後に，オーバヘッドが少ないS/Wスケジューラ法を提案する．

2. 制御の目的

2.1 Fuceプロセッサ

Fuceプロセッサは，TEUを複数搭載したCMPの一種である．Fuceプロセッサでは，命令レベル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としたスレッドの並列実行を追及している．

Fuceプロセッサの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，スレッドは複数の細粒度スレッド(マイクロスレッド)から構成される．この様子を**図1**に示す．以降，細粒度スレッドをスレッドと呼ぶ．

図2に継続の概念を示す．図2は，3つのスレッド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プロセッサでは，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．

任意のスレッドに対して継続される元のスレッドの数をfan-in値，継続する先のスレッドの数をfan-out値と呼ぶ．Fuceプロセッサのスレッド実行制御は，すべて継続によって決まる．Fuceプロセッサのスレッドは，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きる．

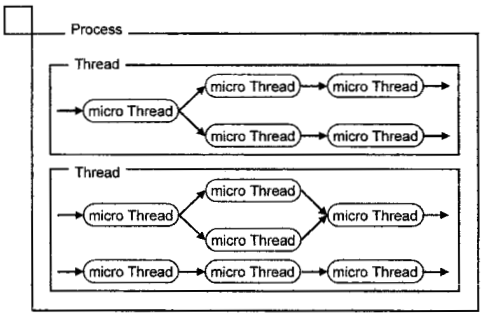

図1 プロセス，スレッド，細粒度スレッドの構成

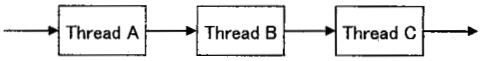

図2 継続概念

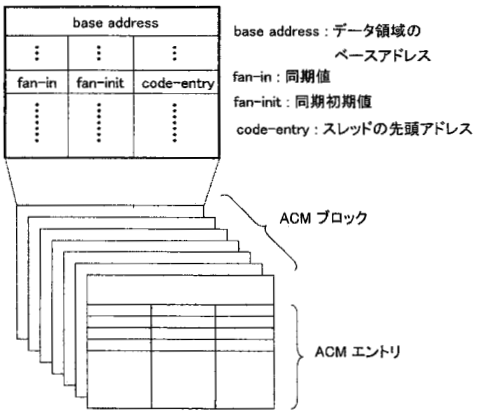

図3 ACM

スレッドの情報は，ACM（Activation Control Memory）に登録され，管理される．ACMの構造を**図3**に示す．ACMは，スレッドの現在のfan-in値，fan-in値の初期値であるfan-init値といったスレッドの同期情報を含んでいる．

Fuceプロセッサは，以下の特徴を持つ．

(1) TEUを複数有する．

(2) 継続と呼ぶ同期手法により，スレッド間でデータを受け渡し，処理を実行する．

(3) H/Wにより，スレッドの実行スケジュール管

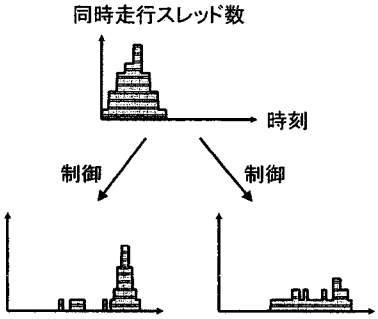

図 4　同時走行スレッド動作の変化

理を行う．

(4)　スレッドは，TEU を横取りされることなく処理終了まで実行される．

したがって，サービス実現を多数のスレッドで構成することにより，処理の並列化を促進でき，効率的なサービス提供が期待できる．一方，多数のスレッドが頻繁に生成消滅を繰り返すことになり，スレッド実行制御のオーバヘッドを抑制することは重要である．

2.2 目　　的

スレッド実行制御の目的は，サービスを構成するスレッド群について，使用 TEU 数を制御し，システムの目的に応じた実行制御を行うことである．制御を行った場合の様子を**図 4** に示す．具体的には，特定のスレッド群の使用 TEU 数を制限し，他スレッド群への影響を抑制する．これにより，次のことが可能になる．

(1)　スレッド群の使用 TEU 数が同様になるように制御すれば，各スレッド群の均等な実行が可能である．つまり，サービスの均等な実行が可能である．

(2)　特定のスレッド群に多くの TEU 使用を認めれば，このスレッド群を優先的に実行できる．つまり，サービスの優先実行が可能である．

さらに，使用 TEU 数の制限をスレッド群の実行中に制御できれば，次のことが可能になる．

(3)　サービス処理の要求終了時刻が近づいた場合は，対応するスレッド群に多くの TEU 使用を認めることで，要求終了時刻を満足できる．

2.3 評 価 尺 度

スレッドの実行制御の目的を果たしているか否かを評価する尺度として，スレッド群が使用している TEU 数と処理時間がある．各評価尺度の関係を**図 5** に示し，以下に説明する．

- $N(t)$：時刻 t における使用 TEU 数
- N_{max}：最大 TEU 数
- N_v：使用 TEU 数の分散

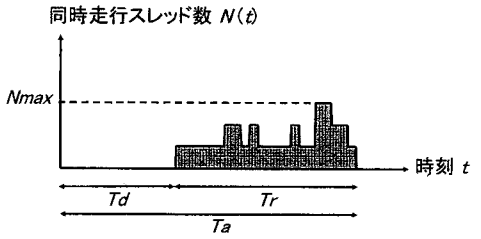

図 5　評価尺度

- T_d：実行開始時間
- T_r：実行時間
- T_a：処理時間

スレッドの実行制御を行う各スケジュール法において，スケジュール法が有する制御パラメータを操作することにより，最大 TEU 数 N_{max} を意図どおりに制御できれば，良いスケジュール法といえる．このとき，分散 N_v が小さければ，そのスレッド群は単位時間当たりの実行処理量の変動が小さいことを意味し，複数のスレッド群が同時走行した際の実行制御が行いやすいといえる．

スレッドを制御することにより，実行開始時間 T_d は増大する．T_d の値が大きいと，プロセスの応答性に問題が発生すると考えられるため，T_d の値は小さい程良い．実行時間 T_r は，スレッドを制御することにより生じる実行時間の変化を比較するために用いる．処理時間 T_a は，T_d と T_r の合計であり，T_a をサービス処理の要求処理時間内に収めることで，要求終了時刻を満足できるといえる．

3. 継続に着目したスケジュール

3.1 制御継続の挿入によるスケジュール

3.1.1 考　え　方

スレッドの実行スケジュール管理は，H/W によって行われる．また，スレッドは TEU を横取りされることなく処理終了まで実行される．

そこで，制御継続の挿入によるスケジュールでは，スレッドの走行状態を厳密に管理するのではなく，スレッド間の継続関係を制御することで，同時走行 TEU 数を制御する．制御の様子を**図 6** に示す．制御では，各スレッドに対して本来の継続とは別に継続を挿入する．以降，この継続を制御継続と呼ぶ．制御継続を挿入されたスレッドは，制御継続を発行されるまで走行を開始することができない．制御継続の発行をスケジュールすることにより，同時走行 TEU 数を制御する．

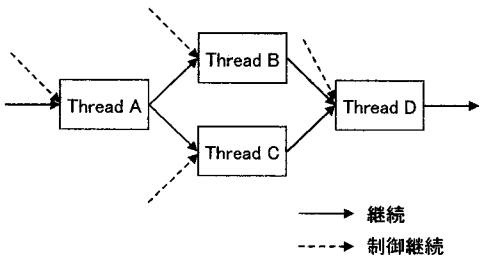

図 6　継続挿入

3.1.2 制 御 法

制御継続を挿入する手段としては，コンパイラによりコンパイル時に挿入する方法やコーディング時に挿入する方法が考えられる．また，制御継続の発行契機は，スレッド終了個数を考慮した方法や一定周期で発行する方法が考えられる．

ここでは，制御継続をコンパイラによってコンパイル時に挿入し，一定周期で制御継続を発行する制御法について説明する．具体的には，周期 T_ω でスレッド M 個に制御継続を発行する．T_ω と M を調整することで，同時走行 TEU 数を制御する．

ここで，制御継続を発行する際，対象スレッドの決定規則として，以下の案がある．

(1) **ランダム法**

ランダム法[6] は，対象スレッドをランダムに選択する．

(2) **エリア分割法**

エリア分割法[6] は，H/W へのスレッド登録順序と実行時の継続発行の順序には相関があると仮定し，スレッド登録順序に基づいて，対象スレッドを決定する．具体的には，スレッドは ACM と呼ばれるスレッド管理表へスレッドエントリを追加することで登録を管理されているため，スレッド登録順序は，ACM に登録されているエントリの順番とする．この際，ACM 表を A 個のエントリを持つエリアに分割して，先頭に位置するエリアのエントリに登録されているスレッドから順次選択する．

(3) **リスト解析法**

リスト解析法[8] は，ソースプログラムに基づいてスレッドの継続関係を静的に解析し，その情報を基に対象スレッドを決定する．具体的には，スレッド間の継続関係を図 6 に示すような有向グラフで表し，継続関係から，各スレッドに優先度を設定し，優先度の高いスレッドから順次選択する．優先度の設定は，次のようにして行う．まず，スレッドを表す有向グラフ上の点 v に対し，開始スレッドからスレッド v までの長さを $l(v)$，点 v の出次数 (継続命令発行数) を $o(v)$ とする．このとき，優先度 $p(v) = a \cdot l(v) + o(v)$ とする．

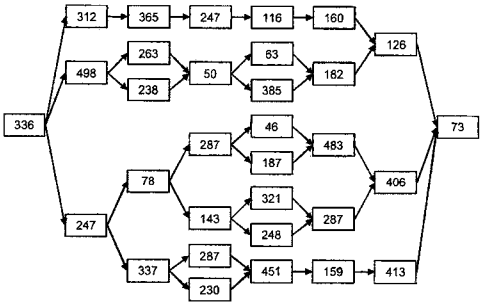

図 7　評価プログラムの依存関係

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

	最大値	最小値	平均	分散
N_{max}	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

3.1.3 効果と問題点

ソフトウェア ModelSim 上で，VHDL で記述した Fuce プロセッサをシミュレートする環境を用い，評価プログラムを動作させて制御法を評価した．

評価プログラムのスレッドの継続関係を**図 7** に示す．図 7 のスレッド中の数字はスレッドの処理時間を示している．

表 1 に，ランダム法で，$T_\omega = 1$，$M = 1$ 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_{max} の値は平均で 5.15 であり，制御しない場合の N_{max} の値である 8 と比較して，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_{max} の最大値が 8 であることから，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，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 2 に，エリア分割法で，$T_\omega = 1, M = 1, A = 4$ 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_{max} 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_{max} 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，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

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

	最大値	最小値	平均	分散
N_{max}	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

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，エリア分割法では，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．しかし，エリア分割法には，制御の確実性が確保できないという問題がある．具体的には，エリア分割法では，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．

リスト解析法では，評価により，次のことが明らかになった．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る．また，エリア分割法ではスレッドの登録順序と実行順序がほぼ等しいという前提で制御を行う必要があったが，リスト解析法では，スレッドの継続関係を解析することで実行順序を予測し制御を行うことができる．

以上に述べたように，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．しかし，以下の課題がある．

(課題 1) スレッドの動的生成への対応
制御対象のスレッドは，制御開始時に全てACMに登録されている必要がある．このため，スレッドを動的に生成するプログラムについて動作を制御できない．

(課題 2) スレッドループへの対応
スレッドはハードウェアによって実行時間が短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，このループするスレッドの同期を解決させるためには，ループの度にOSによる制御継続の発行が必要となる．この手法は1つのスレッドに1度しか制御継続を発行しないため，スレッドループを制御できない．

3.2 継続流れの制御によるスケジュール

3.2.1 考 え 方

制御継続の挿入によるスケジュール制御方式の課題を解決する手法として，スレッドの実行時に動的に継続先を変更し，本来の継続先スレッドへの継続を制御することで同時走行TEU数を制御する手法がある[7]．具体的には，登録スレッドと継続スレッドを用意し，次の処理を行う．各スレッドには，本来の継続先への継続を発行せずに，登録スレッドへ継続を発行させる．登録スレッドは，本来の継続先のスレッドIDをメモリへ保存する．継続スレッドは，保存されているスレッドIDを基に，本来の継続先へ継続を発行する．この発行を制御することでスレッドの実行を制御する．

3.2.2 制 御 法

制御法として，IDプールを用いた制御の処理を**図8**に示す．ここでIDプールとは，登録スレッド(register)により，本来の継続先のスレッドIDを登録された領域である．継続スレッド(cont)は，IDプールに登録された順に一定周期でスレッドへ継続を発行する．以降，本手法をIDプール法と呼ぶ．IDプール法では，H/Wが管理するACM表とは別にS/Wが管理するデータ構造として，IDプールがある．図8に基づき処理の流れを説明する．

(処理 1) srcスレッドは，本来の継続先に継続せず，registerスレッドに継続する．registerスレッドへの継続の際は，本来の継続先スレッドID（図中0x00000001）を引数とする．

(処理 2) registerスレッドは，受け取った継続先スレッドIDをIDプールに登録する．登録エントリの位置は，register_addrで示している．registerスレッドは，複数のスレッドから継続され得るため，排他制御される．

(処理 3) contスレッドは，timerスレッドにより一定周期T_ωごとに継続される．contスレッドは，IDプールから継続すべきスレッドIDを取得する．取得するスレッドIDのエントリの位置は，cont_addrで示す．

(処理 4) contスレッドは，IDプールからスレッドIDをM個取得し，各スレッドに継続する．これにより，継続先の各スレッドは同期が解決する．

3.2.3 利 点

IDプール法は図8で示した処理流れであるため，以下の利点を持つ．

(利点 1) 実行順序に基づいた走行が可能
スレッドIDプールには，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スレッドのIDが登録されていく．このため，スレッドIDプールに登録された順にそのスレッドに継続を与えることで，ほぼスレッドの実行順序に従い，スレッドを走行させることができる．

(利点 2) スレッドの動的生成への対応
スレッドが継続を発行する度に，制御側に本来の継続先スレッドIDを通知するため，ACM表を意識した制御をする必要がない．このため，スレッドが複数の

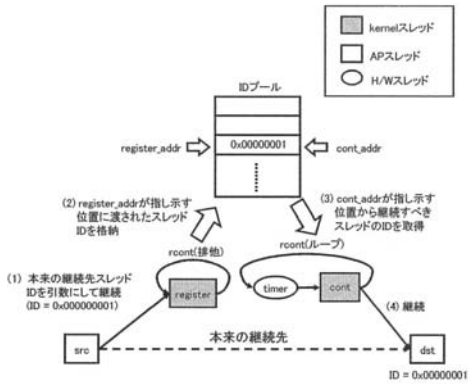

図 8 ID プール法の構成と処理流れ

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) スレッドループへの対応

ループを行うスレッドを特別扱いする必要がない．ループする度にスレッド 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スレッドも他のスレッドと同様に扱う事ができる．

3.2.4 効果と問題点

文献[7] において図 7 の評価プログラムを用い ID プール法を評価した結果を，**表 3** に示す．

表 3 について，$N(t)$ は，アプリケーションを構成するスレッド以外にも 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 を構成するスレッドの走行数が $N(t)$ となる．

表 3 より，M の値が同一の場合，T_ω の値が大きいほど，N_{max} の値が小さくなっている．また，M の値が小さいほど，N_{max} の値が小さくなっている．このことから，T_ω の値が大きいほど，また，M の値が小さいほど，TEU 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．TEU 数と応答性の最もバランスがとれた T_ω 値は，制御対象とするスレッドの走行時間の平均値であることがわかっている[7]．

図 7 におけるスレッド群の走行時間の平均値は，251.59 である．そこで，$T_\omega = 250$，$M = 1$ の場合に着目すると，$N_a = 1.77$，$N_{max} = 5$ である．制御なしの場合の $N_a = 3.45$，$N_{max} = 8$ と比較して，値が低く抑えられており，TEU 占有度を低く抑えられていることを意味する．また，$N_v = 0.42$ であり，N_v の値が小さいことから，制御の安定性も良いといえる．

表 3 において，制御を行った場合について，オーバヘッドは 3% 以内に収まっており，十分実用的に制御できると考えられる．

ここで，図 7 のスレッド依存関係において，スレッドの処理時間を変化させた場合について評価し考察する．

図 7 における各スレッドの処理時間を 10 分の 1 にし，表 3 と同様の評価尺度で ID プール法を評価した結果を**表 4** に示す．

図 7 において，各スレッドの処理時間を 10 分の 1 にすると，スレッド群の走行時間の平均値は，25.159 になる．

ここで，表 4 における $T_\omega = 25$，$M = 1$ の場合から分かるとおり，スレッド処理時間を 10 分の 1 にしても十分安定した制御が可能であるといえる．

一方，オーバヘッドの値に着目すると，表 4 より，オーバヘッドは 23% となり，実用的に制御できているとはいえない．また，スレッドの処理時間が短くなるほど制御のオーバヘッドは大きくなることがわかる．したがって，課題として，

(課題 3) 制御オーバヘッドの抑制

が残る．

4. 継続流れを効率的に制御するスケジュール

ID プール法による制御で，TEU 占有度を低く抑えることができる．また，制御継続の挿入によるスケジュールで問題となった，スレッドの動的生成への対応と，スレッドループへの対応が出来ている．しかし，ID プール法では，すべてのスレッドの継続に対して ID プールへ登録し制御を行うので，制御のオーバヘッドが高くなる．特に，スレッドの処理時間が短くなるにつれオーバヘッドは増加する傾向にある．本章では，ID プール法で問題となるオーバヘッドを抑制するために，継続流れを効率的にスケジュールする制御法について考察する．

継続流れを効率的に制御するスケジュールでは，ID プールを用いた制御を利用する．さらに，ID プールを用いた制御で課題であったオーバヘッドの抑制を達成するために，制御の対象とするスレッド数を制限する．具体的には，ソースコードからスレッドの依存関係を静的に解析し，解析情報を基に制御の対象スレッドを選択する．また，各スレッドへの継続について制御フラグを設定する．制御フラグは，制御の対象スレッドへの継続の 1 つを選択し ON に設定する．制御フラ

表 3　図 7 を ID プール法で制御した評価結果

M	T_ω	N_a	N_v	N_{max}	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%

表 4　処理時間 10 分の 1 で ID プール法で制御した評価結果

M	T_ω	N_a	N_v	N_{max}	T_d	T_r	T_a	overhead
制御なし		3.17	5.28	8	0	3,885	3,885	0.0%
1	25	1.83	0.49	6	23	15,469	15,492	23.08%
1	50	1.46	0.32	4	23	28,084	28,107	23.32%

グを ON に設定された継続は ID プールによる制御を行い，OFF に設定された継続は本来の継続先へ継続を行う．

継続流れを効率的に制御するスケジュール法の，制御組み込み処理の流れは以下のようになる．

(処理 1)　ソースコードの静的解析

(処理 2)　制御対象スレッド数の決定

(処理 3)　継続へ制御フラグの設定

ソースコードの静的解析では，制御の対象とするスレッドの数を決定するためにスレッドの情報を取得する．ここでは，3.1.2 のリスト解析法で示したスレッドの情報に加え，新たにステージ s を設定する．ステージ s は同一の $l(v)$ を持つスレッドの集合であり，s の要素数を N_s で表す．

各 s について，全スレッド数 N_s の内，S 個のスレッドは制御を行わない．これによって，使用 TEU 数を S に近い値で制御することが期待できる．制御対象となる $N_s - S$ 個のスレッドをどれにするかの選択は，スレッド v における fan-in 値 $i(v)$ を基に設定する．ここで，fan-in 値が大きいスレッドは，すべての継続の発行を受け取るまで走行を開始することが出来ないため，$i(v)$ が小さいスレッドに比べ走行の開始が遅くなる可能性が高い．つまり，$i(v)$ が大きいスレッドを制御の対象スレッドとして選択すると，走行が遅れ，かつ $N(t)$ の値がばらつくと考えられる．よって，$i(v)$ の小さいスレッドを優先して制御する．

各継続への制御フラグの設定は，制御対象スレッドへの継続のうち 1 個を ON とし，その他を OFF に設定する．

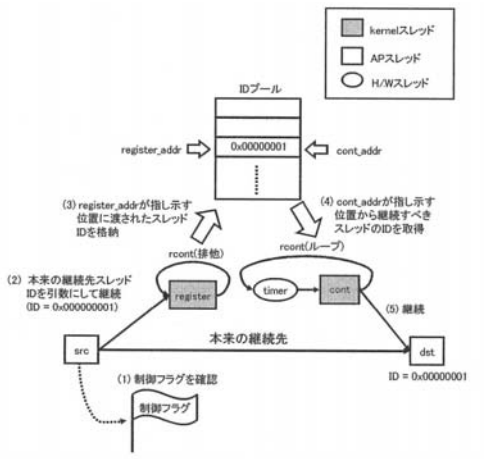

図 9　継続流れを効率的に制御するスケジュールの処理流れ

図 9 に，継続流れを効率的に制御するスケジュールの処理を示す．ID プール法と異なるのは，各スレッドの継続にフラグ情報を持たせている点である．図 9 中 src スレッドについて，フラグが ON の場合は図中 reg スレッドへ継続が行われ，元の継続先スレッド ID をプールに登録し，制御を行う．フラグが OFF の場合は，図中 src スレッドから dst スレッドへ継続が行われ，制御を行わない．

継続流れを効率的に制御するスケジュール法では，

以下の利点がある．

(利点 1) 使用 TEU 数の制限

(利点 2) スレッドの動的生成への対応

(利点 3) スレッドループへの対応

(利点 4) 制御オーバヘッドの抑制

利点 1 から利点 3 に関しては，継続流れの制御によるスケジュールの利点と同じ理由で効果が期待できる．利点 4 については，制御の対象とするスレッド数を減らすことで，制御オーバヘッドの抑制が期待できる．

5. おわりに

本稿では，継続概念による H/W スケジューラにおいて，S/W スケジュール法により，スレッドの使用 TEU 数を制御する方法について述べた．同期カウンタ値を制御するスケジュール法では，実行前に静的に継続制御を挿入する必要があり，このためスレッドの動的生成やスレッドループに対応できない．この問題の解決として，継続流れに着目したスケジュール法があるが，すべてのスレッドに対して制御を行うため，スレッド処理時間が短くなるにつれ，オーバヘッドが大きくなるという問題がある．

本稿では，継続流れに着目したスケジュール法を基に，オーバヘッドを低減する制御法を提案した．提案する制御法では，継続流れに着目したスケジュール法を基に制御する．制御の対象となるスレッドは，ソースプログラムからスレッドの継続関係を静的に解析して決定する．この制御法により，スレッドの動的生成やスレッドループに対応でき，さらにオーバヘッドを抑制した制御が期待できる．

今後は，提案する制御法の評価を行う．

謝辞 九州大学大学院 システム情報科学府 泉雅昭氏に Fuce プロセッサについて多くの助言を頂きました．また，岡山大学大学院自然科学研究科 (現 株式会社中国銀行) 小川泰彦氏に制御法について多くの助言を頂きました．深く感謝いたします．

参考文献

1) J.L., L., S.J., E., J.S., E., H.M., L., R.L., S. and D.M., T.: Converting Thread-Level Parallelism via Simultaneous Multithreading, *ACM Transaction on Computer System*, Vol. 15, No.3, pp.322–354 (1997).

2) D.T., M., F., B., D.L., H., G., H., D.A., K., J.A., M. and M., U.: Hyper-threading technology architecture and microarchitecture: a hyperhtext history, *Intel Technology J*, Vol.6, No.1 (2002).

3) M, A., H, T. and T, M.: An Architecture of Fusing Communication and Execution for Global Distributed Processing, *Parallel Processing Letters*, Vol.11, No.1, pp.7–24 (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).

7) 乃村能成，小川泰彦，日下部茂，谷口秀夫，雨宮真人：スレッド実動作情報に基づく細粒度マルチスレッド制御法，報処理学会研究報告，Vol.2007, No.36, pp.15–22 (2007).

8) 森山英明，乃村能成，谷口秀夫：ソースコード解析情報に基づく細粒度マルチスレッド制御法の検討，情報処理学会第 69 回全国大会講演論文集第 1 分冊，Vol.2007, No.1, pp.33–34 (2007).