(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5145936号
(P5145936)
(45)発行日　平成25年2月20日(2013.2.20)　(24)登録日　平成24年12月7日(2012.12.7)

(51)Int.Cl.　F I
G06F 9/50 (2006.01)　G06F 9/46 462A

請求項の数 14 (全 25 頁)

(21)出願番号	特願2007-519071 (P2007-519071)	(73)特許権者	899000079
(86)(22)出願日	平成18年6月1日 (2006.6.1)		学校法人慶應義塾
(86)国際出願番号	PCT/JP2006/311022		東京都港区三田２丁目１５番４５号
(87)国際公開番号	WO2006/129767	(74)代理人	110000279
(87)国際公開日	平成18年12月7日 (2006.12.7)		特許業務法人ウィルフォート国際特許事務所
審査請求日	平成21年5月28日 (2009.5.28)	(72)発明者	山崎　信行
(31)優先権主張番号	特願2005-162549 (P2005-162549)		神奈川県横浜市港北区日吉３丁目１４番１号　慶應義塾大学理工学部内
(32)優先日	平成17年6月2日 (2005.6.2)		
(33)優先権主張国	日本国(JP)	審査官	鈴木　修治
(31)優先権主張番号	特願2005-167427 (P2005-167427)		
(32)優先日	平成17年6月7日 (2005.6.7)		
(33)優先権主張国	日本国(JP)		

最終頁に続く

(54)【発明の名称】マルチスレッド中央演算装置および同時マルチスレッディング制御方法

(57)【特許請求の範囲】
【請求項１】
複数のスレッドを並列的に処理するマルチスレッド中央演算装置において、
各スレッドの優先度を記憶する優先度記憶手段と、
前記各スレッドの優先度を用いて、前記複数のスレッドの命令の処理の順序または頻度を制御する第１の制御手段と、
各スレッドの一定の繰り返し周期あたりの実行命令数の目標値を記憶する目標記憶手段と、
前記各スレッドの命令実行状態を監視し、前記繰り返し周期あたりの実際の実行命令数と前記目標値を用いたフィードバック制御動作により、前記第１の制御手段により制御される前記複数のスレッドの命令の処理の順序または頻度に対して調整を加える第２の制御手段と
を備え、
前記第１の制御手段が、前記装置内の所定の資源の前記複数のスレッドへの割当量を監視し、前記複数のスレッドへの割当量が前記複数のスレッドの優先度に従うように、前記複数のスレッドの命令のフェッチ、発行または実行のそれぞれの順序または頻度を制御するマルチスレッド中央演算装置。
【請求項２】
請求項１記載の装置において、
前記第１の制御手段が、第１の優先度をもつ１以上の第１のスレッドへの前記資源の割

当量が、前記第１の優先度より下位の第２の優先度をもつ１以上の第２のスレッドへのそれ以下であるとき、前記第１のスレッド中の少なくとも一つのスレッドの命令のフェッチ、発行または実行を促進し、または、前記第２のスレッドの命令のフェッチ、発行または実行を抑制するマルチスレッド中央演算装置。

【請求項３】

請求項１記載の装置において、

前記第１の制御手段が、各スレッドへの前記資源の割当量が少なくて所定の下限条件を満たさないとき、前記各スレッドの命令のフェッチ、発行または実行を促進し、または、前記各スレッドの命令のフェッチ、発行または実行の抑制を解除するマルチスレッド中央演算装置。

【請求項４】

請求項２又は３のいずれか一項記載の装置において、

前記第１のスレッド中の前記少なくとも一つのスレッドには、前記資源の占有量が前記第１のスレッド中で最小であるスレッドが含まれるマルチスレッド中央演算装置。

【請求項５】

請求項１～４のいずれか一項記載の装置において、

前記所定の資源には、フェッチされた前記複数のスレッドの命令をそれが発行されるまで保持する命令バッファが含まれる、マルチスレッド中央演算装置。

【請求項６】

請求項５記載の装置において、

前記所定の資源には、前記命令バッファから発行された前記複数のスレッドの命令をそれが実行されるまで保持する１以上のリザベーションステーションが含まれる、マルチスレッド中央演算装置。

【請求項７】

請求項１～６のいずれか一項記載の装置において、

前記第２の制御手段が、前記各スレッドの前記実際の実行命令数が前記各スレッドの前記目標値に近づくように、或るスレッドの前記実際の実行命令数が前記目標値を満たさない場合、前記或るスレッドの命令のフェッチまたは発行を促進し、または、前記或るスレッドより下位の優先度をもつ他のスレッドの命令のフェッチまたは発行を抑制するマルチスレッド中央演算装置。

【請求項８】

請求項１～７のいずれか一項記載の装置において、

前記第２の制御手段が、前記各スレッドの前記実際の実行命令数が前記各スレッドの前記目標値に近づくようにするために、或るスレッドの前記実際の実行命令数が前記目標値を満たさない場合、前記或るスレッドの命令の処理の頻度を増加させるための調整を、前記第１の制御手段の制御に加え、そして、前記調整を加えても前記或るスレッドの前記実際の実行命令数に所定条件を満たす改善が現れない場合、前記加えられた調整を解除するマルチスレッド中央演算装置。

【請求項９】

請求項１～８のいずれか一項記載の装置において、

前記第２の制御手段が、

監視周期毎に、スレッド毎に実行された命令数をカウントする監視部と、

スレッド毎に、現在の監視周期における前記実際の実行命令数が、予め設定されている目標値を満たすかチェックする第１の比較を行い、且つ、スレッド毎に、前回の監視周期における実行命令数の不足分と前記目標値との加算である繰越残値を求め、前記現在の監視周期における前記実際の実行命令数が前記繰越残値を満たすかチェックする第２の比較を行う比較部と、

スレッド毎に、前記第１と第２の比較の結果に基づいて命令フェッチ又は命令発行の頻度を可変する制御部と

を有するマルチスレッド中央演算装置。

【請求項１０】
請求項９記載の装置において、
前記制御部が、
スレッド毎に、前記命令実行状態が、前記第１と第２の比較結果の双方が肯定的である第１状態、前記第１の比較結果は肯定的であるが前記第２の比較結果が否定的である第２状態、および、前記第１と第２の比較結果の双方が否定的である第３状態のいずれに該当するかを判断する手段と、
スレッド毎に、所定の第１時間長にわたり前記第１状態が続く場合、前記命令フェッチ又は前記命令発行の頻度を減少させる手段と、
スレッド毎に、所定の第２時間長にわたり前記第２状態が続く場合、前記命令フェッチ又は前記命令発行の頻度を増加させる手段と、
スレッド毎に、所定の第３時間長にわたり前記第３状態が続く場合、前記命令フェッチ又は前記命令発行の頻度を増加させる手段と、
スレッド毎に、前記命令フェッチ又は前記命令発行の頻度を増加させた場合、前記実行命令数に所定程度以上の増加が生じたかチェックするリソースアップチェックを行う手段と、
スレッド毎に、前記リソースアップチェックの結果が否定的である場合、前記命令フェッチ又は前記命令発行の頻度を減少させる手段と
を有するマルチスレッド中央演算装置。
【請求項１１】
請求項１記載の装置において、
前記第１の制御手段が、通常の状況下では上位優先度のスレッドの命令が下位優先度のスレッドの命令より先に処理され、前記上位優先度のスレッドが前記装置内の所定の資源を無駄にするか非効率的に使用するような所定状況下では、前記下位優先度のスレッドの命令が前記上位優先度のスレッドの命令より先に処理されるように制御を行うマルチスレッド中央演算装置。
【請求項１２】
請求項１記載の装置において、
前記第２の制御手段が、前記各スレッドの前記実際の実行命令数が前記各スレッドの前記目標値に近づくように、前記複数のスレッドの命令のフェッチまたは発行の順序または頻度を制御するマルチスレッド中央演算装置。
【請求項１３】
請求項１記載の装置において、
前記第２の制御手段が、前記各スレッドの前記実際の実行命令数が前記各スレッドの前記目標値に近づくようにするための調整を、前記第1の制御手段の制御に加えるマルチスレッド中央演算装置。
【請求項１４】
複数のスレッドを並列的に処理するマルチスレッド中央演算装置における同時マルチスレッディング制御方法であって、
優先度記憶手段が各スレッドの優先度を記憶するステップと、
第１の制御手段が、前記各スレッドの優先度を用いて、前記複数のスレッドの命令の処理の順序または頻度を制御するステップと、
目標記憶手段が、各スレッドの一定の繰り返し周期あたりの実行命令数の目標値を記憶するステップと、
第２の制御手段が、前記各スレッドの命令実行状態を監視し、前記繰り返し周期あたりの実際の実行命令数と前記目標値を用いたフィードバック制御動作により、前記第１の制御手段により制御される前記複数のスレッドの命令の処理の順序または頻度に対して調整を加えるステップと有し、
前記第１の制御手段が、前記装置内の所定の資源の前記複数のスレッドへの割当量を監視し、前記複数のスレッドへの割当量が前記複数のスレッドの優先度に従うように、前記

複数のスレッドの命令のフェッチ、発行または実行のそれぞれの順序または頻度を制御する、同時マルチスレッディング制御方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、一般的には中央演算装置（CPUまたはMPU）（以下、単に「プロセッサ」と呼ぶ）に関わり、特に、プロセッサ内で複数のスレッドを並列処理できるマルチスレッドプロセッサおよび同時マルチスレッディングを制御するための方法に関する。

【０００２】

本発明は、特に、各種ロボット、自動車、プラント、ホームオートメーション等の実時間処理を行うアプリケーションにおいて、各スレッドに課された時間制約を守りつつプロセッサのスループットを向上するために好適なマルチスレッドプロセッサに関する。

【背景技術】

【０００３】

実時間処理を行うリアルタイムシステムでは、各スレッドの時間制約を守るために、オペレーティングシステム（OS）のスケジューラは各スレッドの実行周期や時間制約により、各スレッドに優先度を付与する。そして、付与された優先度を基に、優先度の高いスレッドから順番に演算資源を割り当てて実行を行う。ここで、「スレッド」とは、プログラムの実行単位であり、通常、一つのアプリケーションプログラムは多数のスレッドから構成される。

【０００４】

図１に、一時に１スレッドのみを実行する従来のシングルスレッドプロセッサを用いて実時間処理を行った場合の例を示す。この例では、スレッド#0の優先度が最も高く、スレッド#7の優先度が最も低いとする。また、リリース時刻はそのスレッドが実行可能になる時刻、期限はそのスレッドの時間制約（その実行が完了されるべき最終期限）を示す。

【０００５】

図１に示された例では、最初にスレッド#1、スレッド#2およびスレッド#3が実行可能になる。これら実行可能なスレッド#1、#2および#3の中で最も優先度の高いスレッド#1にプロセッサ資源が与えられ、スレッド#1が実行される。スレッド#1の実行が完了すると、次に優先度の高いスレッド#2の実行を行うために、プロセッサ外のメモリとプロセッサとの間でコンテキストスイッチが起こる。コンテキストスイッチでは、現在実行しているスレッド#1のコンテキスト（すなわち、そのスレッドを実行するために必要なプロセッサ内のプログラムカウンタ、レジスタファイル、ステータスレジスタ等の各種資源）をプロセッサ外のメモリに退避させ、次に実行するスレッド#2のコンテキストをメモリからプロセッサ内に復帰させる。その後、プロセッサはスレッド#2の実行を開始する。スレッド#2の実行が完了すると、コンテキストスイッチが起こり、スレッド#2のコンテキストがメモリに退避し、スレッド#3のコンテキストがメモリからプロセッサ内に復帰した後、スレッド#3の実行が開始される。スレッド#3の実行中に、より優先度の高いスレッド#0が実行可能になると、スレッド#3の実行が中断され、コンテキストスイッチが起こり、スレッド#3のコンテキストがメモリに退避し、スレッド#0のコンテキストがプロセッサ内に復帰して、スレッド#0の実行が先に行われる。スレッド#0の実行が完了すると、コンテキストスイッチにより、スレッド#0のコンテキストがメモリに退避し、スレッド#3のコンテキストがプロセッサ内に復帰することにより、中断していたスレッド#3の実行が再開する。

【０００６】

このように従来のシングルスレッドプロセッサを用いた処理では、優先度の高いスレッドの実行が完了した場合や、現在実行しているスレッドよりもより優先度の高いスレッドが実行可能になった場合は、コンテキストスイッチが発生する。OSは、現在実行しているスレッドのコンテキストをメモリに退避し、次に実行するスレッドのコンテキストをプロセッサ内に復帰させなければならない。リアルタイムシステムでは、このコンテキストスイッチが大きなオーバヘッドとなる。一方、コンテキストスイッチのオーバヘッドを削減

する技術として、プロセッサ内で複数のスレッドを並列処理するマルチスレッドプロセッサがある。マルチスレッドプロセッサは、プロセッサ内に複数スレッド分のコンテキストを保持し、これらをプロセッサ内のハードウェアで切り替えながらそれぞれのスレッドを実行するため、コンテキストスイッチを行わずに複数のスレッドを実行することが可能である（例えば、特許文献１参照）。

【０００７】

また、実時間処理に適したマルチスレッドプロセッサのスループット向上のための別の制御技術として、各スレッドに割り当てられた優先度に応じて、複数のスレッドの命令の実行順序をリザベーションステーションにて入れ替える技術が知られている（例えば、特許文献２参照）。

【０００８】

【特許文献１】特開２００４－２２００７０号公報

【特許文献２】特開２００４－２９５１９５号公報

【発明の開示】

【発明が解決しようとする課題】

【０００９】

マルチスレッドプロセッサにおいて、複数のスレッドを並列に実行するとき、キャッシュアクセスや演算器等の演算資源においてスレッド間の競合が起こり得る。演算資源の競合が起こると、1スレッド単体の実行時間が増加する。そして、プロセッサ内で並列的に実行される複数のスレッドの組み合わせが異なれば、演算資源の競合の仕方や頻度が異なってくるため、各スレッドの実行時間が異なってくる。その結果、各スレッドの実行時間の予測精度が低下する。特にリアルタイムシステムでは、各スレッドに時間制約があるため、各スレッドの実行をその時間制約内に完了させ得るよう、スケジューラによりスレッド実行のスケジューリングがなされる。しかし、マルチスレッドプロセッサを用いたシステムでは、上記理由から時間予測精度が低いために、スレッドの時間制約を守れるようなスケジューリングが難しい。

【００１０】

従って、本発明の目的は、マルチスレッドプロセッサにおいて、時間予測精度の低下の原因であるスレッド実行時間の変動を抑制することにある。

【００１１】

本発明の別の目的は、マルチスレッドプロセッサのスループットを向上させることにある。

【課題を解決するための手段】

【００１２】

本発明に従えば、複数のスレッドを並列的に処理するマルチスレッド中央演算装置は、各スレッドの優先度を記憶し、各スレッドの優先度を用いて、複数のスレッドの命令の処理の順序または頻度を制御する。加えて、このマルチスレッド中央演算装置は、各スレッドの命令実行目標を記憶し、各スレッドの時間当たりの実行命令数を監視し、そして、実行命令数と目標値を用いたフィードバック制御動作により、複数のスレッドの命令の処理の順序または頻度を制御する。

【００１３】

このマルチスレッド中央演算装置によれば、各スレッドに付与された優先度を用いた命令処理順序／頻度の制御により、命令処理に必要な各種資源のスレッド間での競合が、各スレッドの優先度に応じて調停される。それに加えて、各スレッドの命令実行状態(例えば所定周期当りの実行命令数、IPCあるいはCPIなど）の監視に基づくフィードバック制御により、同時に処理されている他のスレッドからの影響が減り、各スレッドの実行時間の変動が抑制される。引いては、各スレッドの実行時間の予測精度の向上につながり、実時間処理の時間制約を守りつつ、プロセッサ全体のスループットを向上することが可能となる。

【００１４】

好適な実施形態では、このマルチスレッド中央演算処理装置内の所定の資源（例えば、命令バッファ、リザベーションステーションまたはリオーダバッファ等）の複数スレッドへの割当量が監視され、監視されたそれらスレッドへの資源割当量がそれらスレッドの優先度に従うように、それらスレッドの命令フェッチ、発行または実行の順序または頻度が制御される。例えば、より下位の優先度をもつ１以上のスレッドの資源割当量が、より上位の優先度をもつ1以上のスレッドの資源割当量を超えた場合、その上位優先度をもつスレッド中の少なくとも一つのスレッドの命令フェッチ、発行または実行が促進され、または、その下位優先度をもつスレッドの命令フェッチ、発行または実行が抑制または禁止される。

【００１５】

より具体的には、各スレッドの命令バッファの占有量を監視して、より上位の優先度をもつ１以上のスレッドによる命令バッファの占有量が、下位の優先度をもつ１以上のスレッドによるそれ以下であるとき、その上位優先度をもつスレッド中の少なくとも一つのスレッドの命令のフェッチを促進し、または、その下位優先度をもつスレッドの命令のフェッチを抑制するように制御を行なうことができる。また、いずれかのスレッドによる命令バッファの占有量が少なくて所定の下限条件を満たさないときには、そのスレッドの命令のフェッチを促進し、または、そのスレッドの命令のフェッチの抑制を解除するように制御してもよい。更に、各スレッドのリザベーションステーションの占有量を監視して、より上位の優先度をもつ１以上のスレッドによるリザベーションステーションの占有量が、下位の優先度をもつ１以上のスレッドによるそれ以下であるとき、その上位優先度をもつスレッド中の少なくとも一つのスレッドの命令の発行を促進し、または、その下位優先度をもつスレッドの命令のフェッチを抑制するように制御を行なうこともできる。また、同じ優先度をもつ複数のスレッド間では、命令バッファの占有量がより小さいスレッドに優先的に、命令フェッチ権を与え、または、リザベーションステーションの占有量がより小さいスレッドに優先的に、命令発行権を与えるように制御を行なうことができる。

【００１６】

このようにして、好適な実施形態では、上位優先度のスレッドの命令が下位優先度のスレッドの命令より先に処理されるように制御が行われる。しかし、上位優先度のスレッドが処理資源を無駄にするか、または非効率的に使用するような所定の状況下（例えば、キャッシュミスの発生、分岐予測ミスの発生または高い発生可能性、またはリザベーションステーションの占領など）では、下位優先度のスレッドの命令が上位優先度のスレッドの命令より先に処理されるように制御してもよい。

【００１７】

また、好適な実施形態では、各スレッドの命令実行状態（例えばIPC）をスレッド毎に設定された命令実行目標（例えばIPCの目標値）に近づけるように、特に命令フェッチまたは発行のステージにおいて、上述した優先度による制御動作の制御条件に調整が加えられる。例えば、或るスレッドの命令実行状況が上記目標を満たさない場合、そのスレッドの命令のフェッチまたは発行の頻度が増加させられ、または、そのスレッドより下位の優先度をもつ他のスレッドの命令のフェッチまたは発行の頻度が抑制される。優先度による制御動作を調整させるために、優先度による制御の制御条件に組み込まれた「インフレーション値」と呼ばれるスレッド毎の操作値（好適な実施形態では、「インフレーション値」と呼ばれる）が増減され、それにより、各スレッドの命令のフェッチまたは発行が促進されたり、抑制されたりする。しかし、その操作値を変化させても、その変化に応じた命令実行状態の改善が現れない場合には、操作値が元に戻される。

【発明の効果】

【００１８】

本発明によれば、マルチスレッドプロセッサにおいて、各スレッドの実行時間またはIPCの変動を小さくすることが可能となる。これにより、実時間処理のためのより良いスケジューリングが容易となり、ひいては、プロセッサ全体のスループットの向上に貢献できる。

【図面の簡単な説明】
【００１９】
【図１】従来のシングルスレッドプロセッサによる実時間処理の例を示すタイムチャート図。
【図２】本発明に従うマルチスレッドプロセッサの一実施形態の主要部の構成を示すブロック線図。
【図３】スレッドのIPCを制御するための構成を示すブロック線図。
【図４】スレッドのIPCを制御するための制御関数の状態遷移とインフレーション値増減の動作を示すブロック線図。
【図５】優先度による調停を行なうマルチスレッドプロセッサを用いて実時間処理を行った場合の例を示すタイムチャート。
【図６】優先度による調停と共にIPC制御を行なうマルチスレッドプロセッサによる実時間処理の例を示すタイムチャート。
【図７】４つのパイプラインをもち、優先度による調停と共にIPC制御を行なうSMTプロセッサによる実時間処理の例を示すタイムチャート。
【符号の説明】
【００２０】
１０　マルチスレッドプロセッサ
１２　発行ユニット
１４　キャッシュユニット
１６　実行ユニット
２０　スレッドコントロールユニット
２４　フェッチスレッドセレクタ
３２　命令発行セレクタ
３６　レジスタファイル
３８　コンテキストキャッシュ
４０　リオーダバッファ
４４　命令キャッシュ
４８　命令ウェイトバッファ
５２　データリード／ライトバッファ
５８　データウェイトバッファ
６０　リザベーションステーション
１００　パイプライン処理機構
１０２　実行命令数監視部
１０４　比較部
１０６　制御関数部
【発明を実施するための最良の形態】
【００２１】
以下、図２から図６を参照して、本発明の一実施形態について説明する。
【００２２】
図２は、本発明の一実施形態にかかるマルチスレッドプロセッサの主要部のブロック構成を示す。
【００２３】
図２に示されるマルチスレッドプロセッサ(以下、単に「プロセッサ」という）１０は、SMT（Simultaneous Multi Threading：同時マルチスレッディング）アーキテクチャを採用し、複数（例えば最大８つ）のスレッドを同時に処理することができるとともに、より多く（例えば最大４０）のスレッドのコンテキストを内部に保持して、外部のメモリとの間でのコンテキストスイッチなしに多数のスレッドを処理することができる。図示のように、このプロセッサ１０は発行ユニット１２、キャッシュユニット１４および実行ユニット１６を備える。これらのユニット１２，１４および１６の基本的な機能は次のとおり

である。キャッシュユニット１４は、外部のメモリ（図示省略）にアクセスし、その外部メモリから命令をフェッチしてキャッシュし、外部メモリからデータ（オペランド）をロードしてキャッシュし、また、ストアされるべきデータをキャッシュして外部メモリへストアする。実行ユニット１６は、メモリアクセスユニットや各種の演算器等の演算資源を有し、発行ユニット１２から発行された命令をアウトオブオーダで実行し、命令の実行結果を発行ユニット１２へライトバックする。発行ユニット１２は、複数のスレッドを並列的に処理できるよう、複数のスレッドのコンテキストを内部で保持し、それら複数のスレッドの命令をキャッシュユニット１４からフェッチし、その命令をデコードし、その命令（その命令のデコード結果の動作指示）を実行ユニット１６へ発行し、実行ユニット１６からライトバックされた命令実行結果を受け、そして、その命令をコミットする。

【００２４】

上述した基本的な機能の上に、このプロセッサ１０は、本発明の原理に従う次のような特筆すべき２つの機能を有する。一つは、優先度による調停機能である。すなわち、それは、複数のスレッドに予め割り当てられた優先度に基づいて複数のスレッド間での資源の競合を調停する機能である。別の一つは、各スレッドのIPC（Instructions per Clock Cycle: クロックサイクル当り命令実行数）の変動を抑えできるだけ所望値で一定に保つためのIPC制御機能である。すなわち、それは、スレッド毎の一定時間当たりの実行命令数を監視し、各スレッドの実行命令数が各スレッドに予め割り当てられた実行命令数の目標値にできるだけ一致するように、複数のスレッドの命令の実行頻度を調整する機能である。この２つの機能は相互に関係付けられ同時に働く。ここで、各スレッドの優先度や命令実行数目標値は、例えば、OSのスケジューラやプログラマからプロセッサ１０に対して設定される。

【００２５】

優先度による調停機能を実現するために、発行ユニット１２は、各スレッドに予め割り当てられた優先度を予め記憶する。その優先度はキャッシュユニット１４および実行ユニット１６にも通知される。発行ユニット１２、キャッシュユニット１４および実行ユニット１６は、個別に、複数のスレッドの優先度に基づいて、複数のスレッドのそれぞれに提供される命令フェッチ、発行、実行およびコミットのための資源の量を制御する。なお、この目的のために、上位優先度のスレッドの実行が下位優先度のスレッドにより阻害されないようにするだけでなく、上位優先度のスレッドの性能を低下させずに、下位優先度のスレッドも実行されるような制御方法が実装される。

【００２６】

さらに、IPC制御機能を実現するために、発行ユニット１２は、各スレッドに予め割り当てられた実行命令数の目標値を予め記憶する。発行ユニット１２は、一定の繰返し周期でスレッド毎の実際の実行命令数（コミットされた命令の数）をカウントし、それを予め記憶している目標値と比較する。或るスレッドの実際の実行命令数が目標値に達してなければ、そのスレッドの命令のフェッチまたは発行の頻度が増えるように、上述した優先度による資源競合調停の制御をフィードバック制御により修正する。例えば、或るスレッドの命令のフェッチまたは発行の頻度を増加させるために、そのスレッドより下位優先度のスレッドのフェッチまたは発行が抑制される。逆に、例えば、或るスレッドの命令のフェッチまたは発行の頻度を低下させるために、そのスレッドより下位優先度のスレッドのフェッチまたは発行が促進される。

【００２７】

以下、図２を参照しつつ、プロセッサ１０のより具体的な構成と機能、とりわけ、優先度による調停機能とIPC制御機能に関わる部分に重点をおいて説明する。

【００２８】

発行ユニット１２は、スレッドコントロールユニット２０、同時処理可能なスレッドの最大数分（例えば８つ）のPC（プログラムカウンタ）コントロールユニット２２、フェッチスレッドセレクタ２４、命令デコーダ２６、命令アナライザ２８、命令バッファ３０、命令発行セレクタ３２、リネームレジスタ３４、同時処理可能なスレッドの最大数分（例

えば８つ）のレジスタファイル３６、およびリオーダバッファ４０を有する。キャッシュユニット１４は、命令MMU(メモリマネジメントユニット）４２、命令キャッシュ４４、命令ヴィクティムキャッシュ４６、命令ウェイトバッファ４８、データMMU５０、データリード／ライトバッファ５２、データキャッシュ５４、データヴィクティムバッファ５６およびデータウェイトバッファ５８を有する。実行ユニット１６は、所定の複数個（例えば５つ）のリザベーションステーション６０と各種の演算資源６２－８０を有する。演算資源６２－８０には、メモリアクセスユニット６２、複数のブランチユニット６４、整数除算器６６、複数の整数演算ユニット６８、FP（浮動小数点）除算器７０、複数のFP演算ユニット７２、６４ビット整数演算ユニット７４、ベクタ整数演算ユニット７８およびベクタFP演算ユニット８０などがある。同時処理されるスレッドが同じ演算資源を同時に使おうとする可能性は低いので、リザベーションステーションの個数や各演算資源の個数は、同時処理可能なスレッドの最大数（例えば８つ）よりは少なく、資源の無駄な冗長を避けている。

【００２９】

以下では、命令が処理される流れにほぼ沿って、各部の機能と動作を説明する。

【００３０】

スレッドコントロールユニット２２は、複数スレッドの並列的な処理を制御するためのコアとして働くユニットであり、その内部に複数のスレッドの優先度を記憶し保持し、そして、その優先度をプロセッサ１０全体に配布する。ここで、スレッドコントロールユニット２２に記憶され得る優先度のスレッド数は、好ましくは、このプロセッサ１０が内部に保持し得るコンテキストのスレッドの最大数（例えば４０）以上である。複数のPCコントロールユニット２２は、プロセッサ１０内で同時処理され得る複数のスレッドにそれぞれ割り当てられ、それぞれのスレッドからのプログラムカウンタの値をフェッチスレッドセレクタ２４に与える。複数スレッド間の競合は、まず命令キャッシュアクセスにおいて生じる。そこで、フェッチスレッドセレクタ２４は、命令キャッシュアクセスを要求しているスレッドの中から、各スレッドの優先度を用いて、一つのスレッドを選択し、選択されたスレッドのプログラムカウンタを命令MMU(メモリマネジメントユニット）４２に送る。ここで、フェッチスレッドセレクタ２４は、原則として命令キャッシュアクセスを要求しているスレッドの中で最上位優先度をもつスレッドを選択する。しかし、常に最上位優先度のスレッドが選択されるわけではなく、最上位優先度のスレッドの実行が圧迫されない所定の状況下では、最上位優先度のスレッドに代えて下位優先度のスレッドが選択される場合もある。

【００３１】

命令MMU４２は、命令キャッシュ４４をアクセスして、上記選択されたスレッドの命令を命令キャッシュ４４から命令デコーダ２６に渡させる。ここで命令キャッシュミスが発生した場合、外部のメモリ（図示省略）をアクセスしキャッシュラインを読み込む必要がある。メモリアクセスはキャッシュアクセスに比べてレイテンシが大きいため、キャッシュミスが続くと、命令ウェイトバッファ４８にて複数のメモリアクセス要求が待たされる。この場合、命令ウェイトバッファ４８は、各スレッドの優先度に基づいて、原則として最上位優先度のスレッドのメモリアクセス要求から先に処理する。それにより、より上位の優先度をもつスレッドの実行が優先される。しかし、常に優先度の高いスレッドのメモリアクセス要求が優先されるわけではなく、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドよりも下位優先度のスレッドのメモリアクセス要求が先に処理される場合もある。

【００３２】

命令キャッシュ４４から命令デコーダ２６へ、複数（キャッシュの１ライン分である例えば８つ）の命令が同時に送られる。命令デコーダ２６は、命令キャッシュ４４から同時にフェッチした複数命令を同時にデコードし、それら複数命令のデコード結果（これも、以下の説明では便宜上「命令」という）は同時に命令バッファ４０に入れられる。その際、命令アナライザ２８が、同時デコードされた複数命令の命令タイプ（どのスレッドの命

令であるか、どの演算資源を使用するか、および命令間の依存関係はどうかなどの判断を行うために必要な、スレッド、オペコードおよびオペランドなどに関する情報）を把握し、その命令タイプを命令発行セレクタ３２に与える。命令発行セレクタ３２は、リオーダバッファ４０、リネームレジスタ３４およびリザベーションステーション６０の空きや、命令バッファ３０内の命令間の依存関係を調べ、その結果に基づいて、命令バッファ３０の中から、リザベーションステーション６０に発行可能である命令を選択する。所定数（同時処理可能なスレッド最大数より少ない例えば４つ）の命令が同時に命令バッファ３０から発行され得る。その際、命令発行セレクタ３２は、命令バッファ３０内に発行可能な命令が複数（とりわけ、上記所定数（４つ）より多く）ある場合、各スレッドの優先度に応じて、原則としてより上位の優先度をもつスレッドの命令から先に発行する。しかし、後述するように、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドよりも下位の優先度のスレッドの命令が先に発行される場合もある。

【００３３】

各リザベーションステーション６０に接続された整数演算ユニット６８やFP演算ユニット７２等の各種の演算ユニットについても、スレッド間で競合が発生する。そこで、各リザベーションステーション６０でも、各スレッドの優先度を用いた競合の調停が行なわれる。各リザベーションステーション６０は、そこで待機している命令を、演算に必要なオペランドがそろった命令から順に、アウトオブオーダで演算ユニットに送る。その際、各リザベーションステーション６０は、複数の命令が同時に演算可能になった場合、各スレッドの優先度に基づいて、原則として優先度の高いスレッドの命令から先に演算ユニットに送ることにより、優先度の高いスレッドの実行を優先する。しかし、常に優先度の高いスレッドの命令が優先されるわけではなく、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドに代えて下位の優先度のスレッドの命令が先に演算ユニットに送られる場合もある。

【００３４】

データキャッシュアクセスにおいても競合が発生する。そこで、データリード／ライトバッファ５２は、各スレッドの優先度に基づいて、原則として上位優先度のスレッドのデータキャッシュアクセスから先に実行することにより、上位優先度のスレッドの実行を優先する。しかし、常に優先度の高いキャッシュアクセスが優先されるわけではなく、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドに代えて下位の優先度のスレッドのキャッシュアクセスが先に処理される場合もある。また、データキャッシュミスが起こった場合、外部のメモリ（図示省略）をアクセスしキャッシュラインを読み込む必要があるが、命令キャッシュの場合と同様に、データウェイトバッファ５８が、各スレッドの優先度に基づいて、原則として上位優先度のスレッドのメモリアクセス要求から先に処理することにより、優先度の高いスレッドの実行を優先する。しかし、ここでも、常に優先度の高いスレッドのメモリアクセスが優先されるわけではなく、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドに代えて下位の優先度のスレッドのメモリアクセスが先に処理される場合もある。

【００３５】

リオーダバッファ４０は、アウトオブオーダで実行された命令の実行結果を一時的に保持し、インオーダで命令をコミットする。所定数（同時処理可能なスレッド最大数より少ない例えば４つ）の命令が同時にコミットされ得る。その際、リオーダバッファ４０は、複数（とりわけ、上記所定数（４つ）より多く）の命令がコミット可能であれば、原則として上位優先度のスレッドの命令から先にコミットする。しかし、ここでも、常に優先度の高いスレッドが優先されるわけではなく、上位優先度のスレッドの実行が圧迫されない所定の状況下では、上位優先度のスレッドに代えて下位の優先度のスレッドが先にコミットされる場合もある。

【００３６】

レジスタファイル３６は、同時処理され得るスレッド最大数（例えば８つ）だけ存在し、それぞれ、同時処理される複数のスレッドに割り当てられる。各レジスタファイル３６

には、各スレッドのコンテキストが格納される。コンテキストキャッシュ３８は、より多くのスレッド数（例えば３２スレッド）分のコンテキストが格納できる。コンテキストキャッシュ３８内の任意のスレッドのコンテキストと任意のレジスタファイル３６内のコンテキストとを高速に交換することができる。従って、このプロセッサ１０では、レジスタファイル３６の数（例えば８つ）のスレッドがコンテキストスイッチを行うことなしに処理でき、コンテキストキャッシュ３８がサポートするスレッド数（例えば３２個）のスレッドが、高速にコンテキストスイッチを行うことができる。

【００３７】

上述したように、プロセッサ１０内における命令のフェッチ、発行、実行およびコミットのステージにそれぞれ関わる諸ユニットにおいて、複数のスレッドの優先度を用いて、スレッド間での資源競合を調停するための命令の処理順序または処理頻度の制御が行われる。これにより、上位優先度のスレッドが優先的に実行され、かつ、上位優先度のスレッドの性能を低下させないようにして下位優先度のスレッドも実行されることになる。

【００３８】

さて、上記制御に加えて、各スレッドの実行時間の予測を容易にするために、各スレッドのIPCの変動を抑えてそれをできるだけ所望値の近傍に維持するためのIPC制御が、プロセッサ１０で行われる。上述したスレッドの優先度を用いた命令の処理順序または処理頻度の制御が、このIPC制御により調整または修正される。この実施形態では、スレッドコントロールユニット２０、フェッチスレッドセレクタ２４および命令発行セレクタ３２が、このIPC制御に直接的に関与する。スレッドコントロールユニット２０は、複数のスレッドの各々に予め割り当てられた実行命令数の目標値を記憶し保持する。ここで、スレッドコントロールユニット２２に記憶され得る目標値のスレッド数は、好ましくは、このプロセッサ１０が内部に保持し得るコンテキストのスレッドの最大数（例えば４０）以上である。スレッドコントロールユニット２０は、一定の監視周期毎（例えば、数百クロックサイクル毎）にスレッド毎の実際の実行命令数（IPC ｘ １監視周期中のクロックサイクル数）をカウントし、カウントされた実行命令数と記憶されている目標値とを比較し、そして、その比較の結果に基づくフィードバック制御の方法で、フェッチスレッドセレクタ２４および命令発行セレクタ３２による各スレッドの命令フェッチと命令発行の頻度を制御する。

【００３９】

図３は、このIPC制御を行なう機構の構成を示す。

【００４０】

図３において、ブロック１００は、プロセッサ１０内における上述された複数スレッドの命令のパイプライン処理を行う機構を示し、そこでは、上述されたように、各スレッドの優先度を用いて、スレッド間の資源競合の調停制御が行われる。IPC制御機構は、実行命令数監視部１０２、比較部１０４および制御関数部１０６を有し、図２に示されたスレッドコントロールユニット２０に組み込まれる。

【００４１】

実行命令数監視部１０２は、一定の監視周期毎に、並列処理されている複数スレッドの各々の実行命令数をカウントする。比較部１０４は、スレッド毎に、カウントされた実際の命令実行数と、予め記憶されている目標値とを比較する。後述するように、この実施形態では、後述するような状態遷移に従う制御を行う関係から、比較部１０４は、実際の命令実行数と目標値との比較だけでなく、次のような比較も行なう。すなわち、スレッド毎に、前回の監視周期において実際の実効命令数が実行されるべきであった命令数に足りなかった命令数（繰越命令数）が計算され、その繰越命令数と目標値とが加算されて繰越残値（つまり、現在の監視周期で実行されるべきである命令数）が計算され、その繰越残値と今回の実際の命令実行数とが比較される。

【００４２】

制御関数部１０６は、比較部１０４による比較の結果に基づいて、所定のフィードバック制御動作を行って、各スレッドの命令フェッチと命令発行の頻度を調節するための操作

値を決定し、その操作値をパイプライン処理機構１００に適用する。ここで、フィードバック制御動作には、PD制御やPID制御などを用いることも可能であるが、この実施形態では、後に図４を参照して説明されるような、状態遷移に従って操作値を制御する制御動作が採用される。操作値としては、後に図４を参照して説明されるような、スレッド毎に用意されたインフレーション値というパラメータが採用される。

【００４３】

制御関数部１０６から出力されるスレッド毎のインフレーション値は、パイプライン処理機構１００中の命令フェッチと命令発行の頻度を制御するユニット、すなわち、この実施形態では図２に示されたフェッチスレッドセレクタ２４および命令発行セレクタ３２に与えられる。各スレッドのインフレーション値は、各スレッドの実行命令数と目標値および繰越残値との比較結果に応じて、制御関数部１０６により所定の可変レンジ内で漸次的に増減される。インフレーション値が増加すると命令フェッチや命令発行の頻度が増し、インフレーション値が低いと命令フェッチや命令発行の頻度が減少するように、フェッチスレッドセレクタ２４および命令発行セレクタ３２が、インフレーション値に応じて、優先度を用いた制御動作を調整する。例えば、或るスレッドのインフレーション値が増加すると、フェッチスレッドセレクタ２４および命令発行セレクタ３２は、それぞれ、そのスレッドより下位の優先度をもつスレッドの命令フェッチおよび命令発行の頻度を抑制し、その結果、そのスレッドの命令フェッチおよび命令発行の頻度が増える。逆に、例えば、或るスレッドのインフレーション値が減少すると、フェッチスレッドセレクタ２４および命令発行セレクタ３２は、それぞれ、そのスレッドより下位の優先度をもつスレッドの命令フェッチおよび命令発行の頻度を促進させ、その結果、そのスレッドの命令フェッチおよび命令発行の頻度が低下する。

【００４４】

上述した構成をもつIPC制御機構は、それぞれのスレッドのIPCをある一定の監視周期PERIODで監視する。プログラマは、各スレッドのIPC目標値をipcとして、「PERIOD× ipc」の値を、各スレッドの実行命令数の設定値として各スレッドのIPC設定レジスタに書き込む。IPC制御機構は、いずれかのスレッドについて、設定値「PERIOD× ipc」が満たされていない状態が続いた場合、そのスレッドのインフレーション値を増加させる。

【００４５】

IPC制御機構は、スレッド毎に、以下の値を保持し、インフレーション値を制御する。

【００４６】

ipc_target: 実行命令数の設定値、つまりPERIOD×ipc;

com_cnt: 今期実行命令数、すなわち現在の監視周期PERIODで実際に実行された命令数;

prev_com_cnt: 前期実行命令数、すなわち一つ前の監視周期PERIODで実際に実行された命令数;

carry_over: 繰越命令数、すなわち現在の監視周期PERIODで実行されるべき命令数であり、この値が現在の監視周期PERIODの終了時にゼロでなければ、この値は次の監視周期PERIODへと繰り越される;

stat_cnt: 現在の状態（後述する図４に示される各状態）に留まっている監視周期PERIODの回数。

【００４７】

ここで、繰越命令数carry_overの値は、監視周期PERIOD毎に更新され、

carry_over = carry_over + ipc_target-com_cnt

というように決められる。すなわち、或る監視周期PERIODにおいて繰越命令数carry_overで指定された数より少ない数の命令しか実行できなかった場合、不足命令数が繰越命令数carry_overとして次の監視周期PERIOD繰り越され、それにより、次の監視周期PERIODにおける繰越総命令数carry_overが増加する。逆に、或る監視周期PERIODにおいて繰越命令数carry_overで指定された数より多くの命令が実行された場合、次の監視周期PERIODでは、繰越命令数carry_overが減少する。なお、或る監視周期PERIODで今期実行命令数com_cnt

が繰越命令数carry_overを上回った場合、その監視周期PERIOD内では該当するスレッドの命令フェッチは行われなくなる。

【００４８】

IPC制御機構の制御関数部１０６は、上述した値を用いて、図４に示される状態遷移を監視周期PERIOD毎に行い、インフレーション値を制御する。その詳細は、以下のとおりである.

【００４９】

図４は、制御関数部１０６の状態遷移とインフレーション値増減の動作を示す。

【００５０】

制御関数部１０６は、複数のスレッドのそれぞれについて並列的に、図４に示された動作を行う。

【００５１】

図４において、「ノーマル」１１０は、今期実行命令数com_cntが設定値ipc_targetを満たしており、繰越命令数carry_overも満たしている状態であり、換言すれば、そのスレッドの性能が保証されている状態である。今まで状態が「ノーマル」１１０であったところ、現在の監視周期PERIODで実行した命令数が設定値ipc_targetの両方を満たせなかった場合、状態は「IPCフェイリング」１１２に遷移し、他方、繰越命令数carry_overのみを満たせなかった場合には、状態は「リクエストフェイリング」１１４に遷移する。また、ある回数以上の監視周期にわたり「ノーマル」１１０が続いた場合には、インフレーション値が減少され、その結果、そのスレッドの命令フェッチおよび命令発行の頻度が抑制される。

【００５２】

「リクエストフェイリング」１１４は、今期実行命令数com_cntが設定値ipc_targetを満たしているが、繰越命令数carry_overを満たしていない状態である。つまり、短期的にはIPC目標値が満たされているが、それ以前の監視周期PERIODでIPC目標値が満たされなかったために、不足命令を余分に実行しなくてはならない状態である。この状態がある回数REQ_FAIL_THRESH以上の監視周期にわたり続く場合、インフレーション値が増加され、そして、状態は「リソースアップチェック」１１６に遷移する。他方、「リクエストフェイリング」１１４において、現在の実行命令数が繰越残値を満たした場合は、状態は「ノーマル」１１０へ遷移する。

【００５３】

「IPCフェイリング」１１２は、今期実行命令数com_cntが繰越命令数carry_overと設定値ipc_targetを共に満たしていない状態である。つまり、短期的にも長期的にもIPC目標値が満たされていない状態である。この状態がある回数IPC_FAIL_THRESH以上の監視周期にわたり続く場合、インフレーション値が増加させられ、状態は「リソースアップチェック」１１６に遷移する。他方、「IPCフェイリング」１１２において、繰越命令数carry_overと設定値ipc_targetが共に満たされた場合は、状態は「ノーマル」１１０に遷移する。

【００５４】

「リソースアップチェック」１１６は、インフレーション値を増加させた後、インフレーション値の増加に見合うIPCの向上があったかどうかをチェックするための状態である。この状態において、前期実行命令数prev_com_cntと比較した今期実行命令数com_cntの増分が、予め設定された閾値EFFICIENT_THRESH以上であるが、まだ繰越命令数carry_overが満たされていない場合は、インフレーション値がさらに増加される。他方、繰越命令数carry_overが満たされた場合は、状態は「ノーマル」１１０に遷移する。また、「リソースアップチェック」１１６において、上記増分が上記閾値EFFICIENT_THRESH未満である場合は、インフレーション値は減少させられ、状態は「リソースダウン」１１８に遷移する。

【００５５】

「リソースダウン」１１８は、インフレーション値の増加に見合ったIPCの向上が認め

られなかった場合に相当する状態である。この状態に入ると、インフレーション値が減少させられ、状態は「リソースダウンチェック」１２０に遷移する。

【００５６】

「リソースダウンチェック」１２０は、インフレーション値を減少させた後、IPCが大幅に低下していないかどうかをチェックするための状態である。この状態において、前期実行命令数prev_com_cntと比較して、今期実行命令数com_cntの予め設定された閾値以上に低下した場合、インフレーション値は一周期前の値に戻される。そうでない場合は、インフレーション値は現在の値に維持される。状態は「ノーマル」１１０、「IPCフェイリング」１１２または「リクエストフェイリング」１１４に遷移する。状態が「ノーマル」１１０と「IPCフェイリング」１１２と「リクエストフェイリング」１１４のいずれに遷移するかは、今期実行命令数com_cntと設定値ipc_targetおよび繰越命令数carry_overとの比較結果により決まる。

【００５７】

以上のように、各スレッドについて、IPCが所望値を満たせていない状態が続くと、インフレーション値が増加させられる。ただし、インフレーション値を増加した場合であっても、その増加に見合ったIPCの向上が得られない場合は、インフレーション値を元に戻す。その理由は、プログラムの最大性能が時間と共に変化することに対応するためである。すなわち、プログラムのIPCは時間と共に変化するため，ある監視周期PERIODではIPC目標値を満たせない場合がある。そのような場合にインフレーション値を増加させても無意味であり、他スレッドの実行が阻害されるだけである。そこで、インフレーション値を増加させてもその増加に見合ったIPCの向上が生じない場合には、将来の監視周期PERIODで取り返しがつくことが期待できるので、インフレーション値の増加を取り消す。不足命令数は繰越命令数carry_overに加算される。将来プログラムのIPCが向上しやすくなった時点でインフレーション値を増加させることで、その不足命令数が解消されることになる。

【００５８】

上記のIPC制御により、性能の変動が激しいプログラムにおいても、他スレッドの実行を著しく阻害することなく、特定のスレッドの性能を制御することができる。各スレッドのIPCが所望値近くに制御されることになるので、各スレッドの実行時間の予測の精度が向上する。その結果、実時間処理において、複数スレッドを並列に実行していても時間制約を守ることができるようなスケジューリングが容易になる。

【００５９】

図５は、上述した優先度による調停を行なうマルチスレッドプロセッサを用いて実時間処理を行った場合の例を示す。なお、図５の例では、説明の都合上、上述したプロセッサ１０のような複数スレッドを同時処理できるSMTプロセッサではなく、同時処理できるスレッドは１つのみであるシングルパイプラインのマルチスレッドプロセッサを用いた場合であって、理想的な実行(キャッシュミスや分岐予測ミスが発生しない)の場合を想定する。

【００６０】

図５に示される例では、スレッド#0の優先度が最も高く、スレッド#7の優先度が最も低いとする。この例では、最初にスレッド#1、スレッド#2およびスレッド#3が実行可能になる。ここで、優先度による資源の調停を行なわない従来のマルチスレッドプロセッサでは、スレッド#1、スレッド#2およびスレッド#3が並列的に実行されることになる。これに対し、優先度による資源の調停を行なうマルチスレッドプロセッサでは、優先度による演算資源の調停により、原則として、最も優先度の高いスレッド#1の実行が優先される。そのためスレッド#1が先に実行される。スレッド#1の実行が完了すると、優先度による調停により、次に優先度の高いスレッド#2が優先的に実行される。このとき、スレッド#2のコンテキストがプロセッサ内に保持されているため、スレッド#2はコンテキストスイッチを行うことなく直ちに実行される。スレッド#4の実行中に、より優先度の高いスレッド#0が実行可能になった場合においても、優先度による演算資源の調停によりスレッド#0の実行が優先される。このときも、スレッド#0はコンテキストスイッチを行うことなく直ちに実行

される。スレッド#0の実行が完了すると、次にスレッド#4に演算資源が割り当てられて、スレッド#4が直ちに実行を再開する。

【００６１】

マルチスレッドプロセッサは、一般に1つのスレッドがキャッシュミスなどでストールしている場合でも、別のスレッドを実行することにより、プロセッサ全体のスループットを高く維持できる。この側面に関して、本発明に従うマルチスレッドプロセッサでは、優先度の高いスレッドがストールした場合、次に優先度の高いスレッドを実行するように、優先度による調停を行うことができ、それにより、スループットを向上することが可能である。

【００６２】

図６は、優先度による調停と共にIPC制御を行なうマルチスレッドプロセッサより複数スレッドの同時実行を行った場合の例を示す。図６の例は、同時処理できるスレッドが1つのみのシングルパイプラインのマルチスレッドプロセッサの実際の実行(キャッシュミスや分岐予測ミスが発生する)場合を示している。

【００６３】

図６に示した例でも、図５に示した例と同様に、スレッド#0の優先度が最も高く、スレッド#7の優先度が最も低く、最初にスレッド#1、スレッド#2およびスレッド#3が実行可能になる。優先度による演算資源の調停により、実行可能なスレッド#1、スレッド#2およびスレッド#3中で最も優先度の高いスレッド#1の実行が優先される。上述の図５に例示された理想的な実行の場合、キャッシュミスや分岐予測ミスが発生しないため、優先度に従って順番にスレッドが実行される。この理想的な場合では、スレッド間の干渉がないため、IPCの変化はない。これに対し、図６に例示される実際の実行の場合、キャッシュミスや分岐予測ミスなどが発生することがある。例えばスレッド#1がこの種のミスでストールすると、次に優先度の高いスレッド#2が実行される。スレッド#2のコンテキストはプロセッサ内に保持されているため、コンテキストスイッチを行うことなく直ちにスレッド#2を実行することができる。このため、見かけ上「スレッド#1とスレッド#2が（或いは、より多くのスレッドが）同時並列的に実行される」ことになる。スレッド#2は演算資源を使用するため、優先度のより高いスレッド#1の実行時間に影響を与える。そのため、スレッド間の干渉が発生し、IPCの変化が発生する。IPC制御を採用しない場合には、スレッド#1の時間予測精度が低下する。これに対して、IPC制御を採用することにより、同時に実行されている各スレッド#1、#2の実行命令数（IPC）をそれぞれの目標値に近づけるように、各スレッド#1、#2の実行の頻度が制御される。よって、複数スレッドを同時実行した場合でも、時間予測精度の低下は小さい。そのため、時間制約を守りつつ、プロセッサ全体のスループットを向上することが可能となる。

【００６４】

図５と図６は、理解を容易にするするために、シングルパイプラインのプロセッサの場合を例示した。しかし、同様の説明は、図２に示したような複数のスレッドを同時処理できるSMTプロセッサにも適用される。図７は、図２に示した４パイプラインをもち優先度による調停と共にIPC制御を行なうSMTプロセッサより複数スレッドの同時実行（実際の実行）を行った場合の例を示す。

【００６５】

図７に示すように、マルチパイプラインのSMTプロセッサでは、より多くの数のスレッドが、見かけ上、同時並列的に実行される。同時並列的に実行されるスレッドの数が多いほど、スレッド間の資源の競合が発生する頻度が多くなり、各スレッドについての時間予測精度がより低下する。故に、IPC制御の採用による時間予測精度の低下を抑制できる利点は大きい。

【００６６】

さて、以下では、図２に示したSMTアーキテクチャを採用したプロセッサ１０における、優先度による資源調停とIPC制御のより具体的で詳細な制御方法を説明する。

【００６７】

まず、優先度による資源調停の制御方法を説明する。

【００６８】

この制御は、複数のスレッドを同時に実行する場合における、命令フェッチスロット、命令発行スロット、演算ユニットなどの各演算資源におけるスレッド間の競合を調停し解決するものである。そこでは、スケジューラが付与した優先度を、スレッド間の競合解決のために使用する。プロセッサ１０内で生じるスレッド間の競合解決に優先度を用いることで、優先度の低いスレッドが優先度の高いスレッドの実行を阻害することを防ぎ、優先度の高いスレッドの実行を保証する。ただし、優先度を単純に全ての競合処理に導入すると、システム全体の性能が低下してしまい、マルチスレッドによるレイテンシの隠蔽の効果を得ることができない。そこで、優先度の高いスレッドの性能を低下させずに、優先度の低いスレッドを実行する機構が採用される。SMTアーキテクチャは、スーパースカラアーキテクチャを採用するので、単一スレッドの性能が通常のシングルパイプラインと比較して高い。ただし、単一スレッドの性能をできるだけ高くするには、実行資源をスレッド間で共有化し、全ての実行資源を一つのスレッドが利用できるようにする必要がある。実行資源を共有化すると、単一のスレッドの性能の向上が期待できるが、スレッド間で性能に影響を与えやすくなるので、優先度の低いスレッドが優先度の高いスレッドの性能に悪影響を与えてしまう。その対策として、フェッチスロットや発行スロットといったスロットの優先度制御に加え、共有資源の占有率の制御も実装することができる。また、フェッチスレッド選択でのボトルネックをできるだけ回避し、命令供給機構全体を制御するような制御方法が採用される。その際、ソフトリアルタイムタスクとハードリアルタイムタスクの性質の違いに着目し、単一スレッドの性能に大きく比重を置く制御方法と、単一スレッドの性能の低下を抑制しつつ全体の性能に比重を置く制御方法とを実装することができる。

【００６９】

フェッチスレッド選択では、最上位優先度スレッドの性能低下をできるだけ抑制しつつ、他のスレッドのフェッチを行うことができる制御方法を採用することができる。最上位優先度スレッドの性能低下をできるだけ抑えることに主眼を置く場合、所定の状況下で最上位優先度スレッドにより無駄にされる（使用されない）スロットを、他の下位優先度スレッドが使用することで、ある程度のシステム全体の性能向上が期待できる。そのような制御方法として次のものを挙げることができる。

【００７０】

(1)　命令フェッチでのキャッシュミス

キャッシュミスした後に同じスレッドが命令フェッチを行うことはハードウェアを不必要に複雑化するのみである。キャッシュから命令が返ってくるまでは他のスレッドがフェッチを行う。

【００７１】

(2)　分岐予測ミスからの回復

分岐予測ミスによるパイプライン中の命令の破棄はその分岐命令のコミット時に行われる。分岐命令がライトバックされたときにはその予測の成否がわかるので、もし予測ミスしていた場合にはそれ以後のフェッチは無駄になる。そこでライトバックした分岐命令が予測ミスしていた場合には、その命令がコミットするまで他のスレッドからフェッチを行う。

【００７２】

(3)　命令バッファ中の命令数

最上位優先度スレッドの性能を落さないためには、実行機構への発行を可能な限り絶やさない必要がある。そのためには、命令バッファ中に常に命令が格納されている必要がある。図２に示したプロセッサ１０の場合、命令発行まで５ステージあるので、スレッド選択の際に最上位優先度スレッドの命令が命令バッファ３０内に６クロック分あれば、そのクロックに他のスレッドからフェッチを行ったとしても命令バッファ中の命令が不足することはない。４命令の同時発行のため、２４命令が命令バッファ３０内に存在する場合に

、優先度の低いスレッドからフェッチを行う。ただし次に最上位優先度スレッドがフェッチした命令中にいくつの有効な命令があるかはこの時点では不明であるのに加え、次の命令フェッチがキャッシュミスを起こす可能性があるので、それを考慮にいれると、さらに命令数を増やした方が性能の低下は防げる。

【００７３】

以上の３つの制御方法は、最上位優先度スレッドが無駄にするスロットを使用して下位優先度スレッドの命令を格納するという方法である。それに対し、優先度の高いスレッドが実行資源を非効率的に使うと予測して、優先度の低いスレッドのフェッチを行うという制御方法も採用し得る。その例を以下に挙げる。

【００７４】

(1) パイプライン中の条件分岐命令数

条件分岐命令を多く実行するほど分岐予測ミスの可能性が高くなり、実行資源を無駄にする可能性が高くなる。そのため、パイプライン中の条件分岐命令の数が閾値を越えた場合に優先度の低いスレッドからフェッチを行う。

【００７５】

(2) パイプライン中の命令数

パイプライン中に命令数が多くなると、それが依存するデータを待つ命令が多くなり、実行スロットを埋めることが難しくなる。特に単一のスレッドの命令数が多くなったときに、この現象は顕著になる。そのため、パイプライン中の命令数が閾値を越えた場合に優先度の低いスレッドからフェッチを行う。

【００７６】

(3) 命令バッファ中の命令数

前述した制御方法では、常に最大限命令が発行されると仮定したが、リザベーションステーション６０などの問題で最大限命令を発行できないことがある。そこで、閾値を上述した２４命令よりも低く設定する。閾値が低い程、優先度の高いスレッドの性能が落ちると考えられる。

【００７７】

(4) フェッチ制御ユニット内の占めているステージ数

デコード時に用いられる分岐予測器と比較して、BTB（Branch Target Buffer: 分岐ターゲットバッファ、図示省略）は分岐命令があるかどうかという情報がない分、予測精度が大きく劣る。そのため、BTBによる投機フェッチ数を制限する。フェッチ制御パイプライン中のステージ数によって制御することで、連続するフェッチ回数を抑える。

【００７８】

(5) リザベーションステーション内の待ち命令数

使われる実行ユニットに偏りがある場合や、命令間に強い依存関係がある場合、パイプライン中の命令数による制御ではリザベーションステーション６０を一杯にしてしまう可能性がある。そこで、リザベーションステーション６０内の命令数を数え、閾値を越えた場合に優先度の低いスレッドからフェッチを行う。

【００７９】

以上の５つの制御方法は、システム全体の性能を向上させることを重視している。

【００８０】

さて、発行命令選択では、命令実行ユニット１６に直接命令を供給するために、命令フェッチ機構から十分な命令が来ている場合には、スレッドの性能に大きな影響を及ぼすと考えられる。そこで、優先度の高いスレッドの性能の低下を抑制するために、命令発行機構では発行できる限り優先度の高いスレッドの命令を発行する。優先度による発行命令選択ではフェッチスレッド選択と同様に、次のような方法で優先度の低いスレッドの命令を発行する。

【００８１】

(1) 対象のリザベーションステーションが一杯の場合

優先度の低いスレッドの命令に他のリザベーションステーションを利用する命令が含ま

れている場合はその命令を発行する。

【００８２】

(2) リオーダバッファやリネームレジスタが一杯の場合

これらの実行資源をスレッド毎に所持している場合は他のスレッドの命令を発行する。

【００８３】

(3) 投機実行が禁止されている命令の場合

コントロールレジスタへの書き込み命令のように、投機実行が禁止されている命令の場合はその命令より前の命令が全てコミットされない限り命令発行を止める。

【００８４】

(4) 分岐予測ミスからの回復の場合

フェッチスレッド選択と同様に、ライトバックされた分岐命令が予測ミスをしていた場合にその命令がコミットするまで命令発行を止める。

【００８５】

対象のリザベーションステーション６０が一杯の場合は、優先度の高いスレッドと低いスレッドの利用するユニットがそれぞれ偏っていてなおかつ異なる場合でない限り、優先度の低いスレッドから多くの命令を発行できるわけではない。また、リオーダバッファ４０やリネームレジスタ３４をスレッド間で共有する構成にした場合には、優先度の低いスレッドからの命令発行は行うことが出来なく、投機実行が禁止されている命令は通常のプログラムにおいては頻度が低い。つまり、キャッシュミスが起こり得るフェッチスレッド選択と比較して、優先度の低いスレッドの命令が発行される可能性が低い。複数スレッドの並列実行によるレイテンシの隠蔽には、更なる制御方法が採用される。さらに、システム全体の性能に重きを置き、次のパラメータ、

・パイプライン中の条件分岐命令数
・パイプライン中の命令数
・リザベーションステーション内の待ち命令数

を参照して優先度の高いスレッドの命令発行を止める。

【００８６】

以上、命令フェッチ、発行といったパイプライン中のスロットに関する競合を解決する制御方法について述べた。それに加え、SMTアーキテクチャでは複数スレッドで実行資源が共有されるため、その競合制御についても考慮にいれる必要がある。ここでいう実行資源とは、典型的には、

・命令バッファ３０
・リネームレジスタ３４
・リオーダバッファ４０

などを指す。これらの実行資源は、スレッドごとに用意されるか、または共有化されるかの２通りの実装方法がある。

【００８７】

実行資源がスレッドごとに用意される場合、スレッド間の影響を低く抑えることが可能である。或るスレッドがストールしても、他のスレッドは容易に実行できる。しかし、一つのスレッドが使える量が小さく抑えられるので、単一のスレッドの性能が低くなる。一方、実行資源がスレッド間で共有された場合、単一のスレッドを優先して実行した場合に、実行資源を十分に利用できるので性能が高くなる。しかしスレッド間の影響が大きくなるため、低い優先度のスレッドが高い優先度のスレッドを阻害してしまう。またスレッドがストールしたときに他のスレッドが実行できないことがある。

【００８８】

このようにスレッド間の影響を考えると、スレッド毎に実行資源を用意する方法の方が好ましいが、チップサイズの制限があるのでスレッドあたりの資源の量は小さく抑えられてしまう。そのため、優先度の高いスレッドの性能が低くなるので、ソフトリアルタイム処理に耐え得る高い性能という面で問題がある。そこで、図２に示されたプロセッサ１０では、実行資源を共有化することで、単一のスレッドの性能向上を目指す。

【００８９】
各バッファをパーティション化(幾つかの単位に分割)することにより、ハードウェアの複雑化を防ぐことができる。一つのパーティションを単一のスレッドのみが使用するように設計することで、スレッドごとにパーティションの使用順番とパーティション内の現在位置だけ記憶しておけば良いので、ハードウェアが単純になる。
【００９０】
各バッファを共有することで、優先度の低いスレッドがバッファを占有することにより優先度の高いスレッドの実行を阻害してしまう。そこで、共有バッファにおけるスレッド間の性能に対する影響を抑えるために、各スレッドが使用できるバッファのエントリ数をソフトウェアによりパーティション単位で設定する。これに関し、次の２通りの制御方法が採用できる。
【００９１】
(1) 資源予約
制御レジスタを用いてスレッド毎にパーティション使用権を設定する。同じパーティションを複数のスレッドが利用することができる。
【００９２】
(2) 最大数設定
制御レジスタを用いてスレッド毎に使用できる最大のパーティション数を設定する。
【００９３】
資源予約方式では、スレッド毎に利用できる資源を個別に指定できるので、安定した性能が期待できるが、周期タスクが実行を終え、次の周期を待っている場合など、スレッドが実行できない場合に対応がしづらく、実行資源を有効に使うことができない。最大数設定方式では、他のスロットの制御を優先度制御で行っているために、優先度の高いスレッドから指定された最大限の量まで実行資源を利用することができ、状況に応じて最大限に実行資源を利用しやすい。ただし、各スレッドの最大数の合計がパーティション総数を上回っている場合には、優先度の低いスレッドが優先度の高いスレッドの実行を阻害してしまう。資源予約方式は静的に性能の予測ができ、最大数設定方式は、動的に実行資源の利用の効率化を計ることができる。しかし、これらの制御方法のみでは最上位優先度スレッドの性能を維持することは難しい。例えば優先度の高いスレッドがスケジューリング可能な状態になり、最上位優先度スレッドが切り替わる場合には、それまで実行されていた優先度の低いスレッドが実行資源を占有しており、優先度の高いスレッドの実行が阻害されてしまう。そこで、優先度の高いスレッドの命令が格納されているパーティションの数が閾値以下で、空きパーティションがない場合に、現在その資源を利用しているスレッドの中から最も優先度の低いスレッドの命令を破棄する。命令バッファ３０の場合ではそのスレッドの先頭の命令からフェッチし直し、リオーダバッファ４０では分岐予測ミスの場合と同様の機構を用いてパイプライン中の命令を破棄する。
【００９４】
SMTアーキテクチャでは、優先度による制御を行っていても、同時に実行されるスレッドによっては最上位優先度の実行に影響を与える場合がある。そこで、プロセッサの性能の予測性、及び全体スループットの改善のために、各スレッドがその優先度に従った量のプロセッサ資源を割り当てるように制御することができる。例えば、命令フェッチに関しては、命令バッファの各スレッドへの割当量を、各スレッドの優先度に従うように制御することができる。これにより下位優先度スレッドがフェッチ権を獲得しやすくなるため、最上位優先度スレッドの性能が多少低下する可能性があるが、性能の変動は小さくなり、予測性が高まるものと考えられる。また、命令バッファの利用効率の改善により全体性能の向上が期待できる。
【００９５】
命令フェッチに関するこの制御について、以下に具体的に述べる。この制御は、図２に示されたプロセッサ１０内の主としてフェッチスレッドセレクタ２４によって実行される。この制御は、各スレッドの命令バッファの占有量を監視し、より下位の優先度のスレッ

ドの占有量が、より上位の優先度のスレッドの占有量を超えると、その下位優先度のスレッドからの命令フェッチを抑制または禁止するものである。以後の説明には以下の記号を用いる。

【００９６】

Ti: スレッド;

Gj: 同じ優先度を与えられたスレッドのグループ。グループ間の優先度はGj > Gj-1 であるとする;

ThOfMinIQ(Gj): グループGj中で命令バッファ占有数が最小のスレッド;

THNUM(Gj): グループGjに属するスレッドTiの数;

IQSUM(Gj): グループGjが占有する命令バッファ数（グループGjに属するスレッドの命令バッファ占有数の合計）;

MINIQ(Gj): グループGjが占有する命令バッファ数の最小値;

MIN_IQ_THRESH: １スレッド当りの命令バッファ占有数に対する所定の下限値;

FREQj: グループGj のフェッチ要求;

FETCHj: フェッチ要求FREQj が認められた場合にフェッチを行なうスレッド（フェッチスレッド）。

【００９７】

フェッチ要求FREQjとフェッチスレッドFETCHj は、以下のような制御条件(1)および(2)に従って決められる。

FREQj =

MINIQ(Gj)≦IQSUM(Gj-1)

or

IQSUM(Gj) ≦MIN_IQ_THRESH × THNUM(Gj) … 制御条件(1)

FETCHj = ThOfMinIQ (Gj) … 制御条件(2)

ここで、値MIN_IQ_THRESH は、各スレッドの命令バッファが空になる頻度を低くすることを目的とした下限値である。各スレッドの命令バッファ占有数がこの下限値MIN_IQ_THRESHを下回ると、各スレッドは無条件にフェッチ要求を出すことができる。この値MIN_IQ_THRESHは以下の条件値、

・1回のフェッチでフェッチされる命令数

・フェッチ開始から命令発行までに要する最短サイクル数

・1サイクルの最大命令発行数

によって決めることができる。例えば図２に示されたプロセッサ１０では、1サイクルで８つの命令がフェッチされ、フェッチ開始から命令発行まで6サイクルを要し、1サイクルで４命令が発行される。そのため、命令バッファの命令数が２４になった時点でフェッチを開始することで、命令供給が途絶える頻度を減らすことができると考えられる。そこで、値MIN_IQ_THRESHとして「２４」を採用することができる。

【００９８】

制御条件(1)によれば、或るグループGjの命令バッファ占有数の最小値MINIQ(Gj)が、それより１ランク優先度の低いグループGj-1の命令バッファ占有数IQSUM(Gj-1)と同等かより少ないときには、または、或るグループGjの命令バッファ占有数IQSUM(Gj)が、そのグループGj により占有されるべき命令バッファ数の下限値MIN_IQ_THRESH × THNUM(Gj) と同等かより少ないときには、そのグループGjのフェッチ要求FREQjは認められるが、より優先度の低いグループGj-1のフェッチ要求FREQ j-1は認められない。そして、制御条件(2)によれば、そのグループGj のフェッチ要求FREQjが認められる場合、そのグループGj 中で命令バッファ占有数が最小であるスレッドThOfMinIQ (Gj)に、フェッチ権が与えられる。

【００９９】

以上により、FETCHj 、FREQj が決定されるが、プロセッサ１０では、1サイクルで１つのスレッドのみがフェッチを行うため、複数のフェッチ要求から1つのスレッドを選択しなければならない。そこで、最上位優先度スレッドの実行を優先するために、最上位優先

度のグループのフェッチ要求を選択し、フェッチスレッドを決定するような制御方法が採用できる。また、フェッチスレッドとして選択されたスレッドがキャッシュミス等を起こしフェッチを行えない場合であっても、下位優先度のスレッドにフェッチ権を譲ることはないようにしてもよい。これは、資源占有量の逆転を防ぎ下位優先度スレッドが上位優先度スレッドの性能を圧迫しないようにするためである。

【０１００】

この制御方法では、一方において、下位優先度スレッドがフェッチ権を得やすくなっており、プロセッサの全体性能が向上すると考えられる。そして、他方において、プログラムの特性によらず高優先度スレッドほど多くの命令バッファを獲得できるように制御されており、低優先度スレッドに多くの命令バッファが割かれる問題や、長期間命令バッファが占有される問題は生じにくいと考えられる。この利点は、後述の命令発行の制御方法との組み合わせにより、より強化される。また、この制御方法では、同じ優先度のスレッドが複数あった場合には、それらの中で最も命令バッファ占有数が少ないスレッドからフェッチを行う。

【０１０１】

次に命令発行の制御方法について述べる。

【０１０２】

命令フェッチの制御と同様に、資源の占有量が優先度に従うように、それぞれのスレッドの命令発行が制御される。この命令発行の制御は、図２に示されたプロセッサ１０内の例えば命令発行セレクタ３２によって実行される。この命令発行の制御では、監視対象の資源として、リザベーションステーションまたはリオーダバッファを採用することができる。以下に説明する制御では、リザベーションステーションのスレッド毎の占有量が監視され、より下位の優先度のスレッドの占有量が、より上位の優先度のスレッドの占有量を超えると、その下位優先度のスレッドからの命令発行が抑制または禁止される。以降の説明では以下の記号を用いる。

【０１０３】

Tj,k: グループGj に属するスレッド;

RSNUM(Tj,k): スレッドTj,k のリザベーションステーション占有数;

RSSUM(Gj.): グループGjのリザベーションステーション占有数（グループGjに属するスレッドのリザベーションステーション占有数の合計）;

IREQj,k: スレッドTj,k の命令発行要求。

【０１０４】

命令発行要求IREQj,k は以下のような制御条件(3)に従って決められる。

$$IREQj,k = RSNUM(Tj,k) \leqq RSSUM(Gj-1) \quad \cdots \text{制御条件(3)}$$

【０１０５】

制御条件(3)によれば、或るスレッドTj,k のリザベーションステーション占有数RSNUM(Tj,k)が、それより１ランク優先度の低いグループGj-1のリザベーションステーション占有数と同等かそれより少ないときには、そのスレッドTj,kの命令発行要求IREQj,kは認められるが、その優先度の低いグループの命令発行要求IREQj-1,kは認められない。ここで、図２に示されたプロセッサ１０では、1サイクルで最大で４つのスレッドが命令を発行し得る。命令フェッチと異なり、同じグループの複数のスレッドから命令発行要求が来る場合がある。そこで、命令発行の制御では、命令発行要求を出しているスレッドを以下のように順位付けし、その順位の上位４つのスレッドを発行スレッドとする。

【０１０６】

・スレッドに与えられた優先度の高い順

・同じ優先度（グループ）のスレッド間では、リザベーションステーション６０の占有数が少ない順

【０１０７】

命令発行の制御方法は、前述の命令フェッチのそれとよく似たものである。相違点は、命令バッファ３０の占有数の代わりにリザベーションステーション６０の占有数を用いる

点と、占有数の下限値MIN_IQ_THRESHに対応する条件が無い点である。発行スレッド選択の基準にリザベーションステーション６０を用いる理由は次の通りである。すなわち、発行スレッドとして選択されたスレッドが実際には発行できない場合が存在する。これは主に次のような場合である。

【０１０８】

・リオーダバッファ４０に空きがない
・リネームバッファ３４に空きがない
・発行先の実行ユニットのリザベーションステーション６０に空きがない

プロセッサ１０では、リオーダバッファ４０とリネームバッファ３４は連動しており、実際に制約となるのはリオーダバッファ４０かリザベーションステーション６０である。このうち、実際上は、リザベーションステーション６０に空きがなく命令発行が行えない場合が多い。そのため、上記の制御条件(3)では、より貴重な資源を高優先度スレッドに割り当てるためスレッド選択の基準にリザベーションステーション６０の占有数が用いられる。実際、発明者が実施した評価試験によっても、リオーダバッファ４０よりリザベーションステーション６０の方が、より影響的であることがわかった。

【０１０９】

次に、IPC制御の具体的方法を説明する。

【０１１０】

リアルタイム処理においてはタスクの実行時間の予測性が重要である。そこで、前述の命令フェッチ、命令発行機構にIPCを制御する機構を加えられる。IPC 制御では、一定の間隔で各スレッドの実行命令数を監視し、それが指定された目標値に満たない場合には命令フェッチ、発行が行われる頻度を高くする。また、プログラムの性能は時間とともに大きく変化することがあり、以下に説明するIPC制御ではその点についても考慮される。

【０１１１】

このIPC制御では、グループGjの現在のIPCに応じて変化するインフレーション値INFLjが導入され、前述の命令フェッチ、発行の制御条件(1)と(3)が以下の制御条件(4)と(5)のように変更される。

FREQj =

MINIQ(Gj) ≦ IQSUM(Gj-1) + INFLj × THNUM(Gj)

or

IQSUM(Gj) ≦ MIN_IQ_THRESH × THNUM(Gj)　　… 制御条件(4)

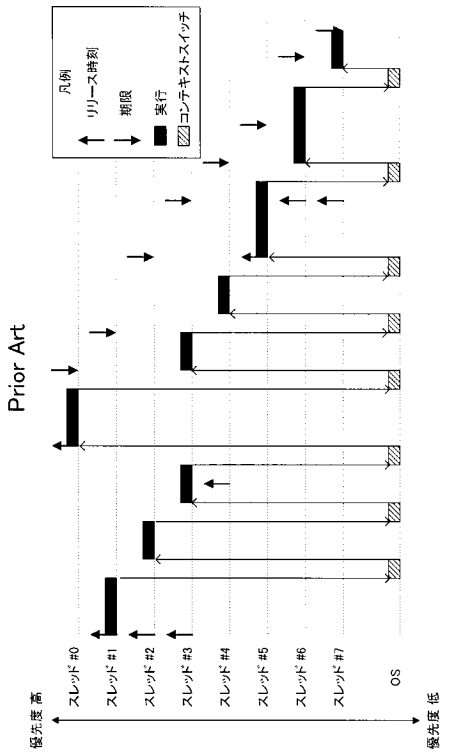
IREQj,k = RSNUM(Tj,k) ≦ RSSUM(Gj-1) +INFLj　　… 制御条件(5)

ここで、既に図４を参照して説明したように、インフレーション値INFLjは、所定の値範囲、例えば0 ≦ INFLj ≦32の範囲で、グループGj の現在の性能によって増減される。すなわち、グループGj が指定されたIPC を得られていない場合には、インフレーション値INFLjが増加し、それにより、グループGjより優先度が下位のスレッドのフェッチと発行が抑制され、グループGjのスレッドの性能を上昇させる。インフレーション値INFLjの可変範囲が０から３２であるのは、図２に示されたプロセッサ１０では命令バッファ３０およびリザベーションステーション６０が３２エントリをもつことに基づいている。

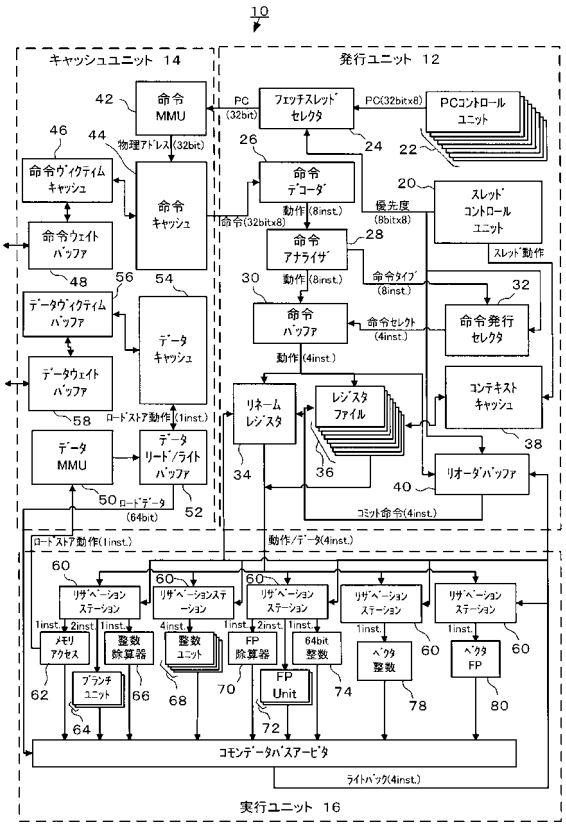
【０１１２】

以上、本発明の実施形態を説明したが、この実施形態は本発明の説明のための例示にすぎず、本発明の範囲をこの実施形態にのみ限定する趣旨ではない。本発明は、その要旨を逸脱することなく、その他の様々な態様でも実施することができる。

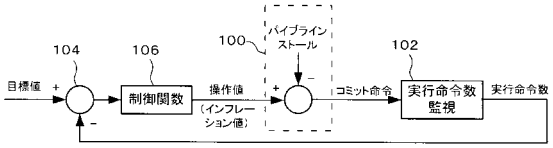
【図１】


凡例
リリース時刻
期限
実行
コンテキストスイッチ
Prior Art
スレッド #0
スレッド #1
スレッド #2
スレッド #3
スレッド #4
スレッド #5
スレッド #6
スレッド #7
OS
優先度 高
優先度 低

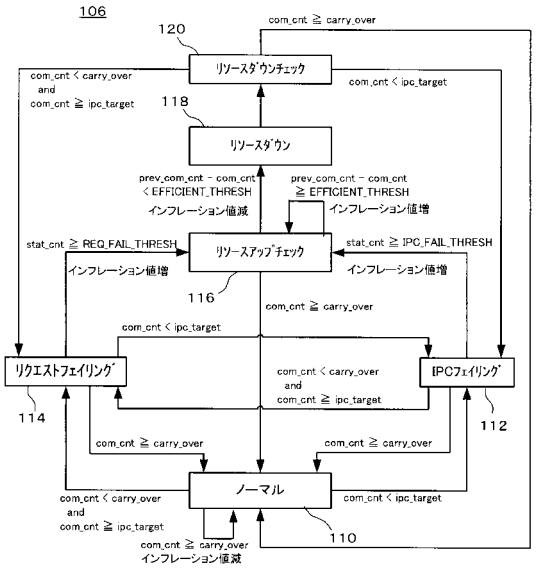
【図２】


10
キャッシュユニット 14
発行ユニット 12
命令 MMU
フェッチスレッド セレクタ
PCコントロール ユニット
物理アドレス(32bit)
命令ヴィクティム キャッシュ
命令 キャッシュ
命令 デコーダ
スレッド コントロール ユニット
命令ウェイト バッファ
命令 アナライザ
スレッド動作
データヴィクティム バッファ
データ キャッシュ
命令 バッファ
命令発行 セレクタ
データウェイト バッファ
コンテキスト キャッシュ
リネーム レジスタ
レジスタ ファイル
データ MMU
データ リード/ライト バッファ
リオーダバッファ
ロードデータ (64bit)
コミット命令(4inst.)
ロードストア動作(1inst.)
動作/データ(4inst.)
リザベーション ステーション
メモリ アクセス
整数 除算器
整数 ユニット
FP 除算器
64bit 整数
ベクタ 整数
ベクタ FP
ブランチ ユニット
FP Unit
コモンデータバスアービタ
ライトバック(4inst.)
実行ユニット 16

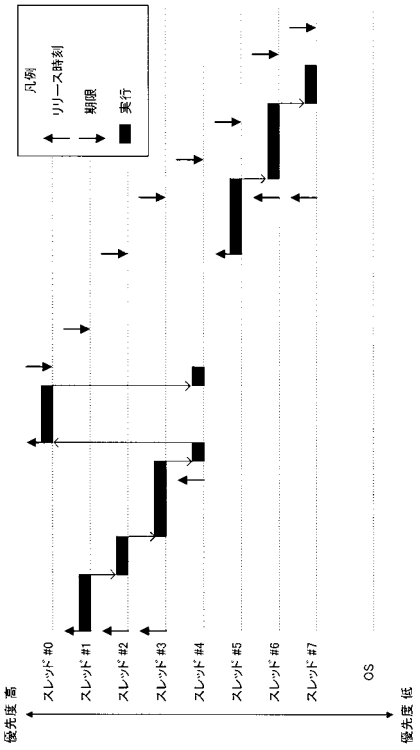
【図３】


目標値
制御関数
操作値
(インフレーション値)
パイプライン ストール
コミット命令
実行命令数 監視
実行命令数

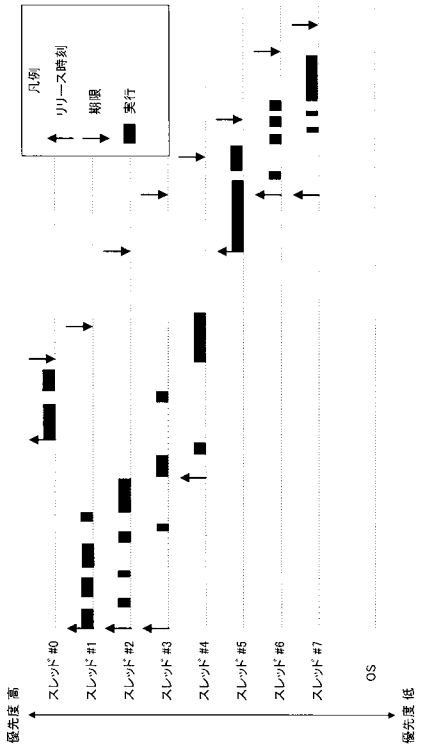
【図４】


106
リソースダウンチェック
リソースダウン
リソースアップチェック
リクエストフェイリング
IPCフェイリング
ノーマル
com_cnt ≧ carry_over
com_cnt < carry_over and com_cnt ≧ ipc_target
com_cnt < ipc_target
prev_com_cnt − com_cnt < EFFICIENT_THRESH
インフレーション値減
prev_com_cnt − com_cnt ≧ EFFICIENT_THRESH
インフレーション値増
stat_cnt ≧ REQ_FAIL_THRESH
stat_cnt ≧ IPC_FAIL_THRESH

【図５】


凡例
リリース時刻
期限
実行
スレッド #0
スレッド #1
スレッド #2
スレッド #3
スレッド #4
スレッド #5
スレッド #6
スレッド #7
OS
優先度 高
優先度 低

【図６】


凡例
リリース時刻
期限
実行
スレッド #0
スレッド #1
スレッド #2
スレッド #3
スレッド #4
スレッド #5
スレッド #6
スレッド #7
OS
優先度 高
優先度 低

【図７】


凡例
リリース時刻
期限
実行
スレッド #0
スレッド #1
スレッド #2
スレッド #3
スレッド #4
スレッド #5
スレッド #6
スレッド #7
OS
優先度 高
優先度 低

フロントページの続き

(56)参考文献 特表２００４－５３２４４４（ＪＰ，Ａ）
特許第２８８２４７５（ＪＰ，Ｂ２）
特開２００４－２８７８８３（ＪＰ，Ａ）
特開平１０－１２４３１６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/46-9/54
G06F 9/38