修 士 論 文

リアルタイム処理に適した
キャッシュ制御法の研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

室岡 健太郎

2007年3月

修 士 論 文

リアルタイム処理に適した キャッシュ制御法の研究

指導教官　日比野 靖教授

審査委員主査　日比野 靖 教授
審査委員　田中 清史 助教授
審査委員　井口寧 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410120 室岡 健太郎

提出年月: 2007 年 2 月

概 要

リアルタイムシステムではタスクのコンテキスト切替えが頻繁に起こる。これはキャッシュが定常状態になる前、あるいは、なり始めてすぐに再ロード過渡状態にへ遷移する現象が頻繁に起きていること示している。このことが原因でリアルタイムシステムではキャッシュがほとんど効かない。

本稿では、コンテキスト切替えが頻繁に発生するリアルタイムシステムに対して有効に機能するキャッシュシステムを提案する。論文はキャッシュの物理構成に関するものとキャッシュブロックの置換に関するもので構成される。物理構成に関しては、キャッシュの再ロード過渡状態でのミス回数に注目し、ミス回数が最も低く押さえられるキャッシュ容量、連想度などを検討する。また、キャッシュブロックの置換に関してはタスク ID をタグに付与し、アクセス頻度の高いタスク、あるいはシステム設計者が残しておきたいタスクを置換対象とはしないようにする制御法を検討する。これによって使用頻度の高いタスク、あるいは残したいタスクは常時オンキャッシュにすることが可能になる。

以上の手法を実装したキャッシュシステムをシミュレータにより評価することで、提案したキャッシュ構成と制御法がコンテキスト切替えに対し有効に機能することを示す。

目 次

第 1 章　はじめに **1**

第 2 章　研究の概要 **2**

第 3 章　キャッシュの再ロード過渡状態とフットプリント **4**
3.1　タスクの切替えと再ロード過渡状態 4
3.2　フットプリントの概念 4
3.3　再ロード過渡状態の連想度とミス回数の関係 5
3.4　複数タスクの再ロード過渡状態 7
3.5　修正フットプリントと修正フットプリントサイズ 9

第 4 章　対象とするリアルタイムシステム **10**
4.1　アドレス系列のプロファイル項目 10
4.2　タスクスケジューリング方針 12

第 5 章　コンテクストスイッチに有効なキャッシュシステム **13**
5.1　キャッシュ容量の検討 13
5.2　キャッシュリソースの静的優先度 14
5.3　タスク ID を用いた置換制御法 15
5.3.1　静的優先度に基づく置換制御 16
5.3.2　タスクの走行頻度に基づく置換制御 16
5.3.3　キャッシュリソースの静的優先度とタスクの走行頻度の双方に基づく置換制御法 16

第 6 章　シミュレーション実験と評価 (1) **19**
6.1　実験の概要 19
6.2　タスクのプロファイルとタスクの主メモリ領域の配置 20
6.2.1　タスクの生成とそのプロファイル 20
6.2.2　タスクの主メモリ領域の配置 26
6.3　キャッシュ容量 27
6.3.1　キャッシュ容量の見積り 27
6.4　タスクの優先度割り当て 27

6.5 実験結果 31
6.5.1 通常の LRU で置換制御を行った場合 31
6.5.2 キャッシュのリソースに対する静的優先度に基づいた置換制御を行った場合 33
6.5.3 タスクの走行回数に対する動的優先度に基づいた置換制御を行った場合 35
6.5.4 静的優先度と動的優先度の組み合せに基づいた置換制御を行った場合 37

第 7 章 考察 **39**
7.1 通常の LRU で置換制御を行った場合 39
7.2 キャッシュのリソースに対する静的優先度に基づいた置換制御を行った場合 39
7.3 タスクの走行頻度に対する動的優先度に基づいた置換制御を行った場合 . . 39
7.4 静的優先度と動的優先度の組み合せに基づいた置換制御を行った場合 . . . 40

第 8 章 シミュレーション実験と評価 (2) **41**
8.1 実験概要 41
8.2 実験結果 43
8.2.1 通常の LRU で置換制御を行った場合 43
8.2.2 静的優先度に基づいた置換制御を行った場合 43
8.2.3 動的優先度の組み合わせに基づいた置換制御を行った場合 43
8.2.4 静的優先度と動的優先度の組み合わせに基づいた置換制御を行った場合 43

第 9 章 考察 **48**

第 10 章 結論 **49**

第1章　はじめに

現在のように高度に発達した情報化社会では、家電製品や情報端末、産業器機などのほとんどの機器にはプロセッサが組み込まれている。

機械や機器に組み込まれて制御を行うコンピュータシステムを、組み込みシステムと呼ぶ。組み込みシステムでは特に信頼性とリアルタイム性が求められる。

機械や機器を制御するための組み込みシステムは高い信頼性が求められる。自動車のエンジン制御やプラント制御など、誤作動が人命に関わるため、極めて高い信頼が求められる。

リアルタイム性というのは、システムが定められた時間要件を満たして動作する性質のことをいう。特に緊急性を要する処理の場合、いかなる状態においてもその処理を期限内(デットライン)に終了することを求められる。

本論文では処理の中核となるプロセッサのリアルタイム処理能力の向上を目標とする。特にプロセッサのキャッシュに焦点を当てる。

リアルタイムシステムではタスクのコンテクスト切替えが頻繁に起こってる。切替え時に発生する再ロード時でのミスがプロセッサのリアルタイム処理能力に大きな影響を与える。この再ロード時でのキャッシュミスに注目し、このミスを低減することで頻繁に発生するコンテクスト切替えに十分効くキャッシュシステムを提案することで、組み込み用プロセッサのリアルタイム処理能力の向上を目指す。

本論文は次の各章で構成される。第2章では研究概要を述べる。第3章ではフットプリントの概念を説明し、再ロード過渡状態におけるミスの発生現象を論じる。第4章では、本論文で用いるアドレストレースの生成法とリアルタイム環境をシミュレートするタスクスケジューリングについて論じる。第5章では、コンテクストスイッチに対して有効なキャッシュ制御法を提案する。提案する制御法はキャッシュライン毎の優先度を知る方法を与えることで実現される。具体的には、キャッシュタグにタスクIDを付与し、タスクIDからタスク毎のキャッシュ優先度を参照する方法である。第6章では、シミュレーション実験による本論文で提案するキャッシュ制御法の評価を行う。第7章で結論をまとめる。

第2章　研究の概要

研究内容は大きく 3 つの内容から構成されている。第 1 は対象とするリアルタイムシステムに関する内容、第 2 はキャッシュパラメータに関する内容、第 3 は置換アルゴリズムに関する内容である。研究の概要を図 2.1 に示す。

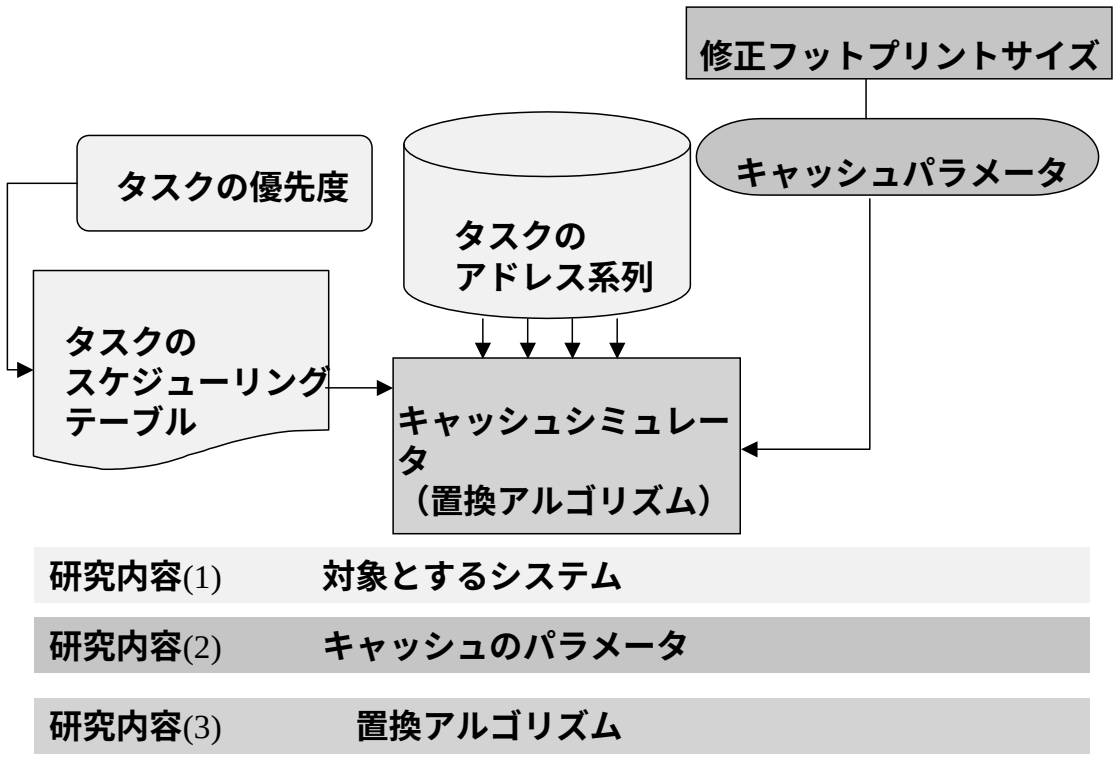

図 2.1: 研究概要

第 1 の、対象とするリアルタイムシステムは、複数のタスクがコンテクスト切替えを頻繁に行いながら走行するシステムを想定する。タスクは割り込みを受けると、制御を他のタスクに移す。割り込みを受けたタスクは再び制御を移されると、割り込れた次の時点から実行を継続する。詳細は第 4 章で論ずる。

第 2 の、キャッシュパラメータに関する内容であるが、複数のタスクを収容し、タスクスイッチに対してキャッシュが有効となるキャッシュサイズ、キャッシュの連想度 (ウェイ

数) について検討する。キャッシュ容量には、フットプリントとフットプリントサイズが重要である。また、コンテクストスイッチが頻繁に生ずる環境では、キャッシュの連想度が重要なパラメータとなる。詳細は第 3 章で論ずる。

第 3 の置換アルゴリズムについては、通常の LRU の問題点を克服するため、キャッシュラインの割り当ての優先度を導入する方法を提案する。

キャッシュラインの優先度は、キャッシュタグにタスク ID を付与し、タスク ID からタスク毎のキャッシュ優先度を知る方法を与える。評価は第 5 章で論ずる。

第3章　キャッシュの再ロード過渡状態とフットプリント

本論文ではフットプリントという概念を用いてキャッシュシステムを解析していく。そこでリアルタイムシステムで頻繁に起こるキャッシュラインの再ロード過渡状態におけるミスモデルに注目し、最適なキャッシュパラメータを求める。

3.1　タスクの切替えと再ロード過渡状態

リアルタイムシステムではタスクのコンテクスト切替えが頻繁に起こる。各タスクは制御を得る度に、しばらくは参照ラインをキャッシュに再ロードすることに時間を費やす。参照ラインをキャッシュに再ロードする過渡的な期間におけるミスを、再ロード過渡状態でのミスと言い、タスクのコンテクスト切替えが頻繁に起こるリアルタイムシステムではこのミスが重大な影響を及ぼす。

本研究ではこの**再ロード過渡状態でのミス**を低減することで、タスクのコンテクスト切り替えに十分効くキャッシュシステムを提案する。つまり過渡状態でのミス回数の大きさをいかに少なくするかということが研究の焦点となる。そこでこの過渡状態を測定するために**フットプリント**と**フットプリントサイズ**という概念を用いて、キャッシュ内の挙動を解析していく。

3.2　フットプリントの概念

フットプリントというは連想度無限大のキャッシュに、任意のタスクを動作させた時に触れるラインの集合である。図 3.1 はプロセス A を単独でセット数 4、連想度無限大のキャッシュに動作させた時に触れるラインの集合を示したものである。図 3.1 の例ではこのラインの集合がプロセス A のフットプリントで、その集合内のライン数がフットプリントサイズとなる。この例では、プロセス A のフットプリントサイズは 7 である。

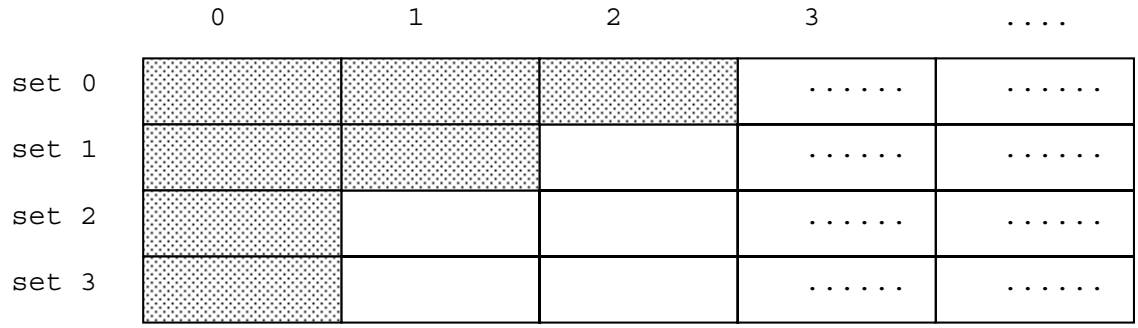

図 3.1: セット数 4、無限連想度キャッシュ中のプロセス A のフットプリント

3.3 再ロード過渡状態の連想度とミス回数の関係

図 3.2 はキャッシュを求めて競合する 2 つのプロセスの片方に注目し、そのプロセスの再ロード過渡状態におけるキャッシュサイズと連想度の関係を示したミスモデルである。

図 3.2 ではプロセス A とプロセス B のフットプリントはそれぞれ 1900、7900 でプロセス A の再ロード過渡状態でのキャッシュミスの様子を示している [1]。

このミスモデルからキャッシュサイズはタスクのフットプリントサイズの合計よりも大きくすることが必要であり、なおかつ連想度が十分にないといくらキャッシュサイズを大きくしてもキャッシュミスがなかなか減少しないということがわかる。

この例では、キャッシュサイズは、プロセス A のフットプリント 1900 とプロセス B のフットプリント 7900 の合計値 9800 よりも少し大きめのサイズを用意し、なおかつ連想度を 32 ウェイまで上げればプロセス A の再ロード過渡状態でのミスはほとんど無くなることがわかる。従ってこの結果を利用すれば、適切なキャッシュサイズや連想度などのキャッシュパラメータを設定することができ、再ロード過渡状態でのミス回数を低く押さえることが可能である。

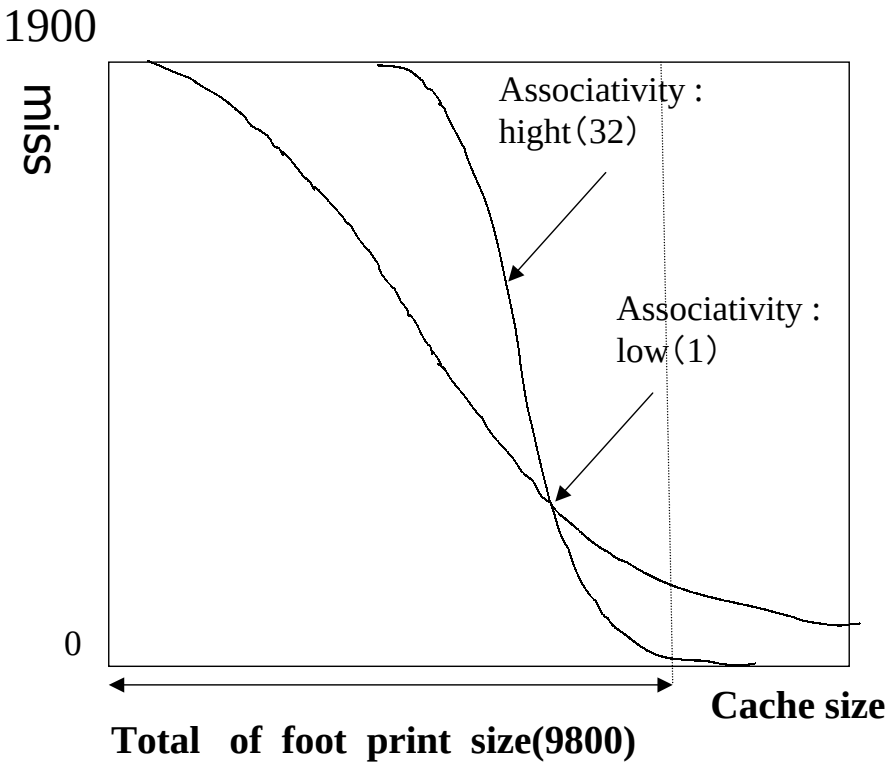

図 3.2: キャッシュの再ロード過渡状態

表 3.1: フットプリントサイズ

プロセス	フットプリントサイズ
A	1900
B	7900
合計	9800

3.4 複数タスクの再ロード過渡状態

図 3.2 は 2 タスクしか想定しておらず、このモデルの特性が多数のタスク間でも成り立つかどうか検証されていなかった。

そこで複数のタスク間でこのミスモデルの特性が実際成り立つかどうかシミュレーションを行った。シミュレーションの場合、用意したタスクは 8 個でその中のタスク 1 が再度参照された場合を想定する。シミュレーションによる結果を図 3.3 に示す。

図 3.3 から、連想度 1(ダイレクトマップ) の場合、フットプリントがすべて収容可能なキャッシュサイズ (1637) であってもミス回数は 110 回を越えており、フットプリントの半分以上がタスクスイッチ後、追い出されていることがわかる。

一方、連想度を 16(一点鎖点) とすれば、キャッシュサイズをすべてのフットプリントを収容可能な 1637 以上のフットプリントサイズをとれば、ミス回数をゼロにすることが可能であることがわかる。すなわち、コンテクストスイッチ後、タスク 1 のフットプリントは追出されずにすべて残っている。

以上の結果から複数のタスクの場合も 2 タスクの場合と同様の性質が導かれる。

すなわち、キャッシュサイズとしてタスク集合の合計フットプリントサイズをとり、連想度を十分大きく取れば、再ロード過渡状態のミス回数をゼロにすることが可能である。

しかし、この結果をそのままリアルタイムシステムへ適用することはできない。なぜならば多数のタスクのフットプリントサイズの合計値を上回るキャッシュを用意することは、すべてのタスクがオンキャッシュすることになってしまい現実的ではないからである。

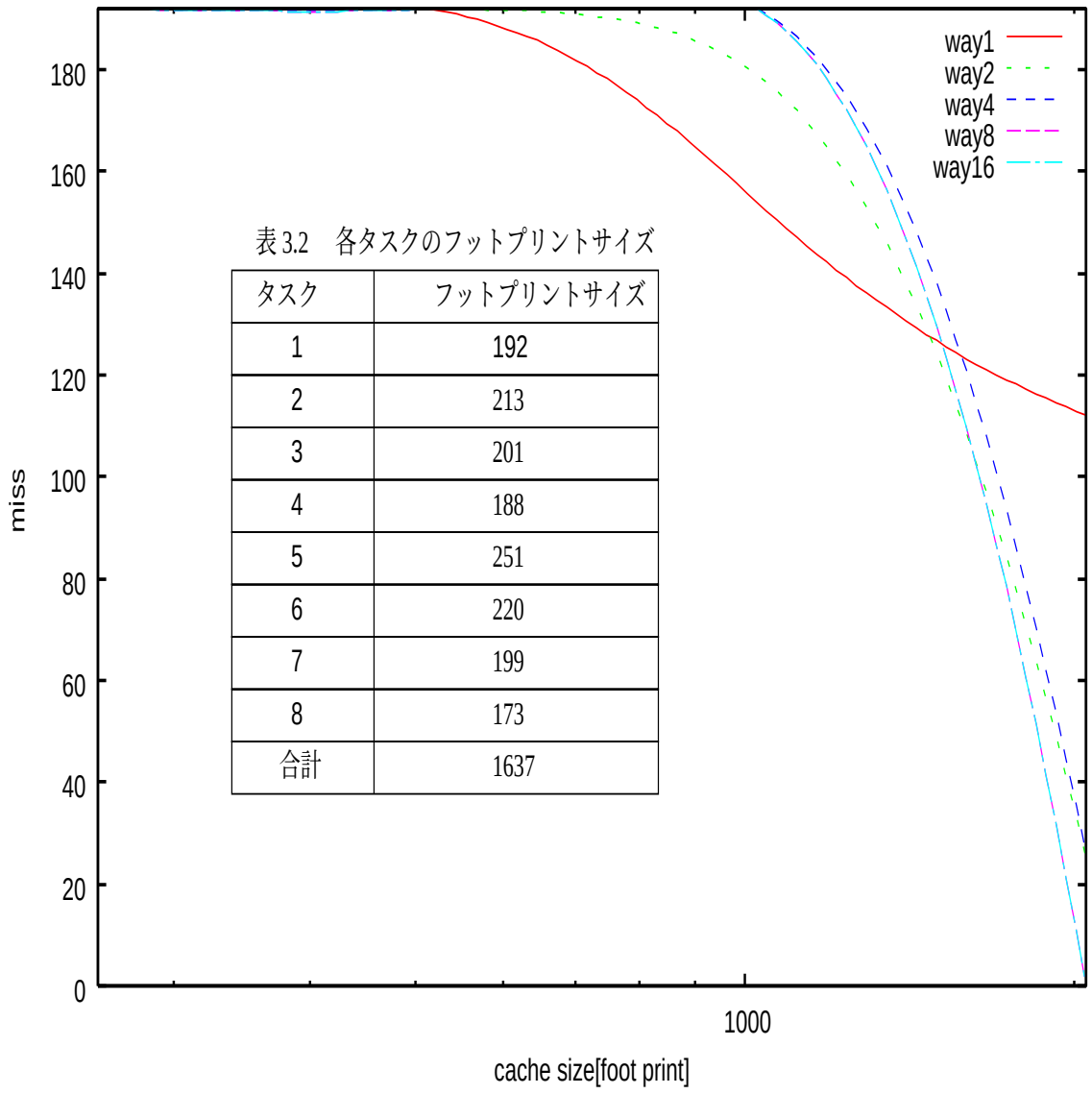

表 3.2　各タスクのフットプリントサイズ

タスク	フットプリントサイズ
1	192
2	213
3	201
4	188
5	251
6	220
7	199
8	173
合計	1637

図 3.3: シミュレーションによるキャッシュの再ロード過渡状態

3.5 修正フットプリントと修正フットプリントサイズ

本論文では、タスク毎のアクセス回数が高いフットプリントに注目し、そのフットプリントサイズを用いてキャッシュサイズを見積る。アクセス回数の高いラインを基準にキャッシュリソースを割り当て、本論文で提案するタスク ID を用いた置換制御を行うことでアクセス回数の高いラインをオンキャッシュする。それにより小さいキャッシュサイズで高いヒット率を実現することが可能となる。

本論文では、特にタスク毎のフットプリントに対する累積アクセス回数で必要なキャッシュリソースを削減することにする。具体的には、タスク毎の累積アクセス回数が 90%のフットプリントを**修正フットプリント**、そのサイズを**修正フットプリントサイズ**としこれらを用いて必要なキャッシュサイズを検討していく。

表 3.2: 修正フットプリントサイズ

タスク	フットプリントサイズ	修正フットプリントサイズ
1	192	84
2	213	90
3	201	85
4	188	71
5	251	93
6	220	99
7	199	82
8	173	78
合計	1637	682

第4章　対象とするリアルタイムシステム

リアルタイムシステムのアドレストレースは入手困難であるため、本論文ではアドレスジェネレータによって生成したタスクセットのアドレストレースを、スケジューラでスケジューリングすることで、リアルタイムシステムの実働作を疑似したアドレストレースを得る。各々のタスクのアドレストレースは、タスクの特徴を決めるプロファイルをパラメータとして、アドレスを生成する。

4.1　アドレス系列のプロファイル項目

タスクのアドレストレースを生成するために表4.1のタスクプロファイルのパラメータをアドレスジェネレータに渡し、各々のタスクのアドレストレースを生成する。各々のパラメータを説明する。

1. 連続アクセスの平均長
   プログラムは、数ステップから数十ステップの間、分岐命令が出現するまでプログラムカウンタを1づつ増加させて進行する。分岐命令が出現するまでの平均長である。

2. データのアクセス頻度
   命令オペランドがレジスタまたは分岐アドレスの場合は、オペランドメモリアクセスは生じない。
   オペランドがメモリアクセスである場合の頻度を指定する。ロードとストアは区別しないことにする。

3. 上方向分岐の確率
   数命令から数十命令毎に出現する分岐命令の分岐方向を指定する。分岐方向が下方の場合、プログラムの進行方向への分岐である。分岐方向が上方向である場合、プログラムは既に通ったステップを繰り返すことになる。すなわちループを形成する。上方向分岐の確率を与えることにより、ループの出現をシミュレートし、間接的にループ関数を制御する。
   また、下方分岐の確率 (1 − 上方向分岐確率) はプログラムの下方への成長の速さを制御することになる。

4. データアクセスの範囲
   メモリアクセスのアドレス範囲である。本論文では、アドレス範囲内で一様なメモリアクセスを発生させるのでなく、正規分布に近いメモリアクセスを発生させる。
5. 総ステップ数
   タスクが起動して終了するまでの間に、タスクが進行方向に進んだステップ数である。

表 4.1: タスクプロファイル

項目	パラメータ
1	乱数の種
2	連続アクセスの平均長
3	分岐距離
4	上方向分岐確率
5	データアクセスの頻度
6	データアドレスのベースアドレス
7	データアドレスの範囲
8	総ステップ数

4.2 タスクスケジューリング方針

スケジューリング方針として、ランダムスケジューリングを適用する。ランダムスケジューリングを用いることによりタスクの切替えが頻繁に起こるリアルタイムシステムの実動作を疑似することができる。

本論文で行うキャッシュシステムに対する評価は、このランダムスケジューリングを採用しているので、実際のリアルタイムシステムで使用されいるスケジューリングよりもずっと厳しいはずである。というは、実際のリアルタイムシステムで使用されているスケジューリングでは、ある特定のタスクが頻繁に起動するなど、タスク間での走行回数にかなりの偏りが見られる。このため走行頻度の高いタスクは、自然にオンキャッシュすることになり、タスクのコンテキスト切替えによって発生する再ロード過渡状態でのキャッシュミスは、ランダムスケジューリングに比べて相対的に低くなる。これとは対照的にランダムスケジューリングは、タスクの走行回数がほぼ一様になるため、特定のタスクが自然にオンキャッシュすることはない。従って、タスクのコンテクスト切替えによって発生する再ロード過渡状態でのキャッシュミスが実際使用されているスケジューリングよりも相対的に高くなる。

このためランダムスケジューリングによる評価が十分なものであるなら、実際にリアルタイムで使用されているスケジューリングでの評価もそれ同等か、それ以上の評価が期待される。　また、実際にアプリケーション毎に走行回数の優劣がある場合を想定し、予めタスク毎に走行頻度を付与した。これによってリアルタイムシステムの実動作の近似モデルが得られる。

第5章　コンテクストスイッチに有効なキャッシュシステム

5.1　キャッシュ容量の検討

タスクセット内の各タスクのプロファイル、タスク毎に割り与えられた優先度、そして図 3.2 のミスモデルからキャッシュ容量を決定する。

ここで fsN という記号を導入する。これはあるタスクの総アクセス回数の N%が集中するタスクのフットプリントサイズである。すなわち fs90 は総アクセス回数の 90%が集中するタスクのフットプリントサイズであり、fs100 は総アクセスのフットプリントサイズである。これはタスクのフットプリントサイズそのものである。

キャッシュサイズの見積りは、タスク毎のプロファイルと図 3.2 のミスモデルを参照し決定する。特にタスク毎の重要なプロファイルは fs100 と fs90 である。fs90 の値が小さければ小さいほど、タスクの局所性が高くなる。そのようなタスク群からなるシステムだと小さいキャッシュサイズでも、シングルタスクシステムなら十分キャッシュが効くはずである。

本論文では、リアルタイムシステムを想定しているが、たとえアルタイムシステムであろうと、プログラムの局所性が高ければキャッシュのヒット率は高くなる。また図 3.2 より適切なキャッシュサイズと連想度を取れば、あるタスクの再ロード過渡状態時でのミス回数はほとんどなくなる。つまりそのタスクがオンキャッシュすることになる。

以上のことから、常時オンキャッシュになっていてほしいタスクには、fs100 のキャッシュリソースを与え、残りのタスクに対しては、fs90 より少ないキャッシュリソースを与えるようにする。fs90 より少ないキャッシュリソースを割り与えられたタスク群に関しては、適切な置換制御を施すことで再ロード過渡状態でのミス回数を極力少なくする。

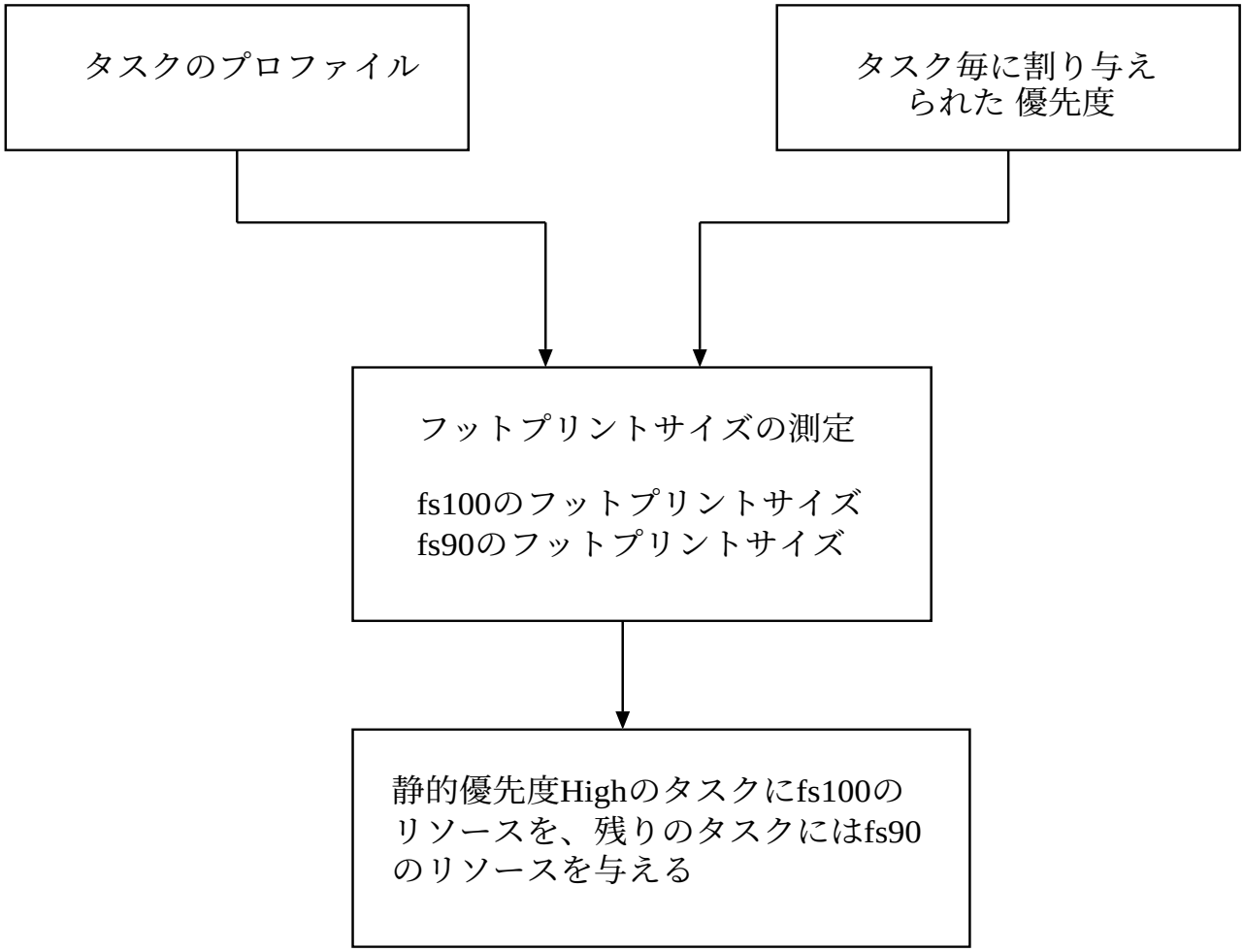

図 5.1: キャッシュ容量の見積り

5.2 キャッシュリソースの静的優先度

システム設計者は、厳しい時間制約を求められるタスクに対して、優先的にキャッシュリソースの割り当てを行うことができるようにする。優先的にリソースを割り与えられたタスクは、常にオンキャッシュが可能となり、タスク切り替えに対する厳しい時間制約を十分に満たすことができる。

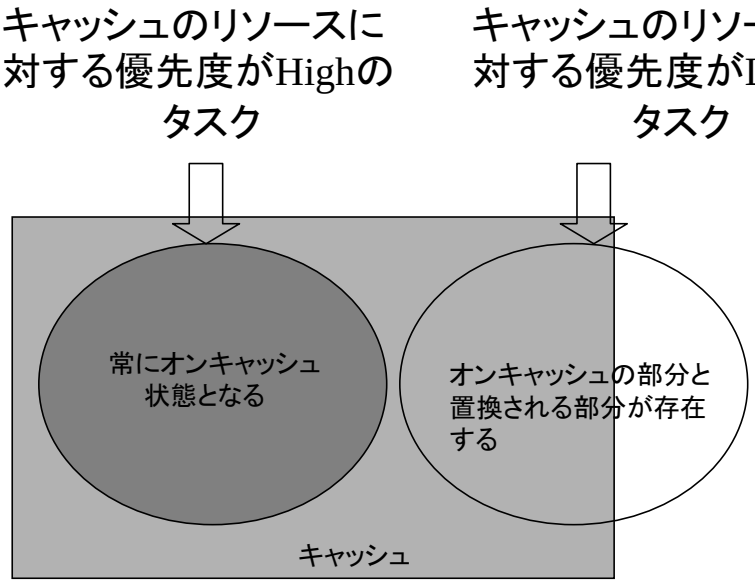

図 5.2: 高い優先度を割りあてられたタスクのキャッシュ内での様子

5.3 タスク ID を用いた置換制御法

提案する置換制御法は、キャッシュのタグに付与したタスク ID を使用してキャッシュ内のラインを置換する。この置換制御ではタスクに付随する 2 つの情報を基に置換を行う。一つは、タスクに割り当て得られたキャッシュリソースに対する静的な優先度であり、もう一つはタスク自身の走行頻度である。本研究では 2 つの情報をそれぞれ単体のみで使用した場合の置換制御と、双方を使用した場合での置換制御を行い、それぞれを評価する。以下に置換アルゴリズムの原則を示す。また、図 5.3 にそれら置換制御のアルゴリズムを示す。

1. まず、走行中のタスクに属するラインは極力追い出さないという方針をとる。以下の方法で追い出し候補が見つからなかった場合のみ走行中のタスクのラインを LRU で追い出す。

2. 走行中のタスクより低いキャッシュ優先度のラインの中で、最も低い優先度を選び、優先度が等しい場合は LRU で選択する。この原則により走行中のタスク及び、走行中のタスクより高い優先度のタスクに属するラインはオンキャッシュとなる。

3. 走行中のタスクと等しいか高いキャッシュ優先度をもつラインの中で, 最も優先度の低いラインを置換する。優先度が等しい場合は LRU で置換する。この原則により、

走行中のタスクより高い優先度のラインで、セット中のラインがすべて占められてしまうことを防止する。

5.3.1 静的優先度に基づく置換制御

システム内のタスクに予め割り当てた優先度に基づいて置換制御を行う。この場合優先度のレベルは High、Low の 2 種類とした。

提案した優先度に基づくアルゴリズムによれば、優先度が High、Low の 2 レベルの場合、走行中のタスクのキャッシュ優先度は High または Low のいずれかである。

走行中のタスクのキャッシュ優先度が High の場合は、走行中タスクのラインを除き、優先度が Low の中から LRU で追出される。

走行中のタスクのキャッシュ優先度が Low の場合は、走行中タスクのラインを除き、優先度が Low の中から LRU で追出される。

この場合、結局、優先度 Low のラインが追出され、優先度 High のラインが残ることになる。

5.3.2 タスクの走行頻度に基づく置換制御

システムが稼動してから動的に変化するタスクの走行頻度に基づいて置換制御を行う。実際は動的に変化するタスクの走行回数を任意のタイミングで、ある閾値に基づいて優先度の優劣を判定する。この場合も優先度のレベルは High と Low の 2 種類とする。

この場合は、優先度を動的に決定すること以外は静的優先度を与えた場合と同様である。

優先度 High と Low に対するキャッシュ置換の挙動は、静的優先度による置換制御の場合と同様である。

すなわち、優先度 Low のラインが追出され、優先度 High のラインが残る。

5.3.3 キャッシュリソースの静的優先度とタスクの走行頻度の双方に基づく置換制御法

この置換制御法は上記の 2 つの制御法を組み合わせたものである。この場合、優先度のレベルはキャッシュのリソースに対する優先度と、タスクの走行回数に基づく、動的優先度を掛け合わせた数に分けることができる。各々の優先度の組み合わせに対して任意にレベルを与えて評価を行う。

本論文では、キャッシュのリソースに対する優先度と、タスクの走行頻度に対する優先度のレベルは High と Low の 2 種類のみとした。ゆえに、この制御法における優先度のレベルは 4 つに分けることができる。

優先度 High と High、Low と Low に対するレベルの割り当ては、それぞれ最大の 4 と最低の 1 であることは自明である。そこで本論文では、優先度 High、Low と優先度 Low、High の組み合わせに対して、それぞれレベル 3、2 を割り与えることにする。本論文で指定した優先度の割り当てを表 5.1 に示す。

キャッシュ優先度が 4 レベルの場合の挙動について考える。極端な例としてセット数が 1 でフルアソシアティブキャッシュについて考えてみる。

キャッシュ優先度レベルが最低の 1 のタスクを除いたタスクセットのフットプリントサイズの合計が、キャッシュサイズを越える場合、もし高優先度のラインを必ず残すとすると、すべてのラインが優先度レベル 4、3、2 のラインで占められてしまい、最低優先度 1 のラインはロードされてアクセスが終了するとすぐに追出されてしまう。

従って、走行中のタスクのキャッシュ優先度より高い優先度を与えられたラインを追出す必要がある。走行中のタスクのキャッシュ優先度が最低の 1 であっても、他に優先度 1 のラインがない時は、優先度の低い順に LRU で追出しラインを置換する。

表 5.1: 優先度のレベル

キャッシュリソースの静的優先度	走行頻度の動的優先度	優先度レベル
High	High	4
High	Low	3
Low	High	2
Low	Low	1

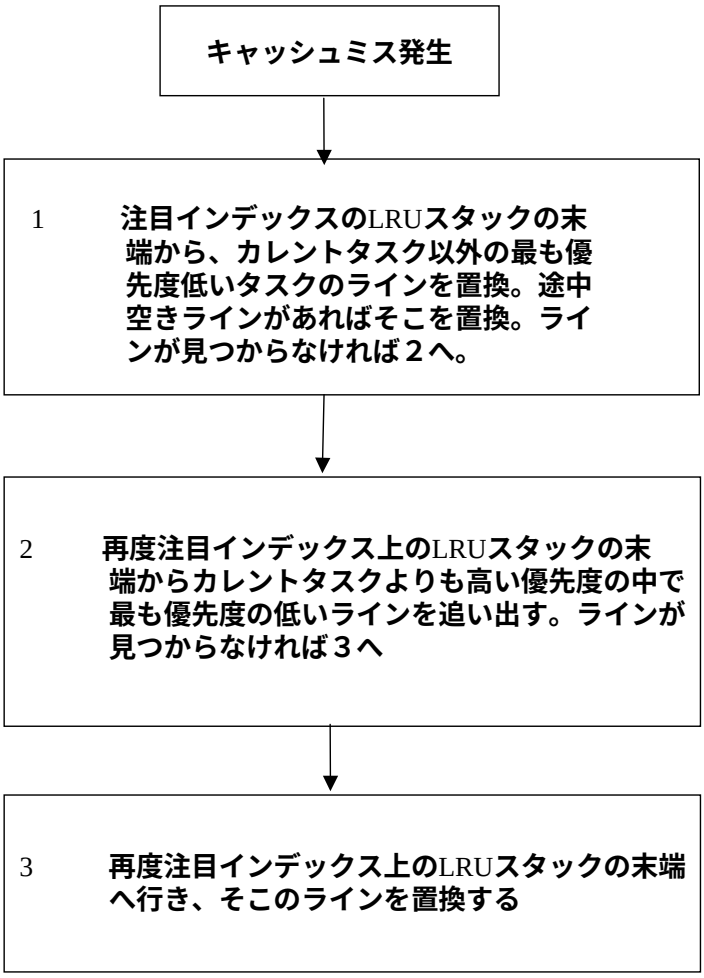

図 5.3: 置換アルゴリズム

第6章　シミュレーション実験と評価(1)

6.1　実験の概要

実験の全体構成を図6.1に示す。実験内容は大きく3つに分けることができ、その各々は、(1)対象となるリアルタイムシステムに関する内容と、(2)キャッシュパラメータに関する内容と、(3)置換アルゴリズムに関する内容である。

実験の流れとしては、まず対象となるリアルタイムシステムを構成し、そのシステム内の各々のタスクプロファイルを基にキャッシュ容量を見積る。そして提案した置換制御法を実装したキャッシュシミュレータを作成し、置換アルゴリズムを変化させて評価を行う。

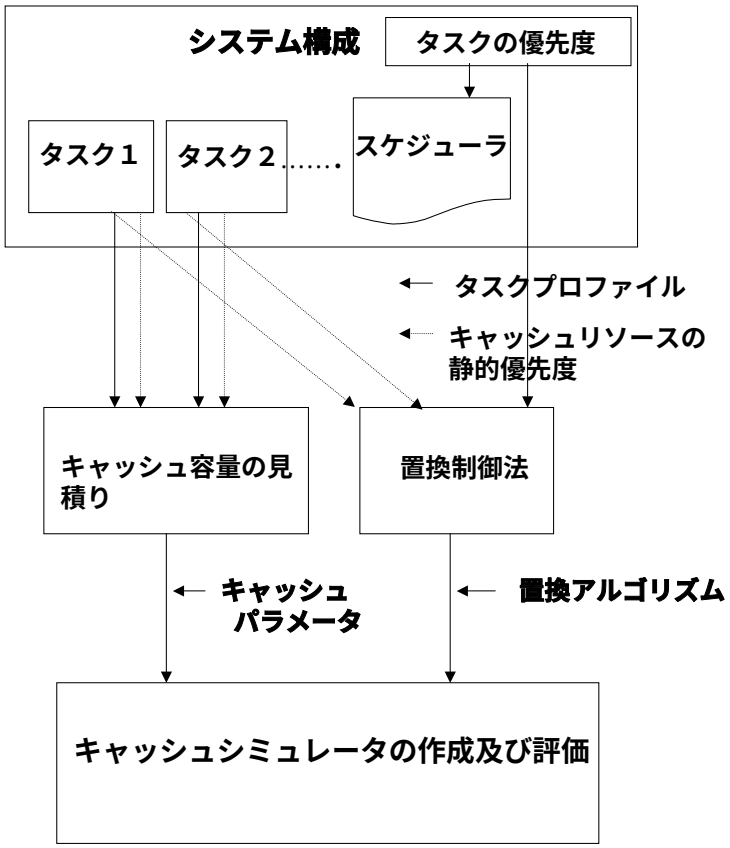

図 6.1: 実験の全体図

6.2 タスクのプロファイルとタスクの主メモリ領域の配置

システム内の各タスクのプロファイルとして、表 4.1 のパラメータを与える。これらのプロファイルとパラメータにより生成したアドレストレースにより、各々のタスクのフットプリントサイズ、累積アクセス回数が 90%に達する修正フットプリントサイズ、及びアドレス分布範囲といったものが得られる。また各タスクの主メモリ上での配置は、テキスト領域の後にデータ領域を配置するようにしており、各タスクには領域が重ならないようにプログラムサイズより大きなオフセットを与えた。

6.2.1 タスクの生成とそのプロファイル

各タスクのアドレストレースは表 6.1 で示したパラメータを引数として与え、アドレスジェネレータによって生成される。

生成したアドレストレースから得られたタスク毎のフットプリントサイズを表 6.2 に示す。

fsN はタスクのアクセス回数全体の N%が収まるフットプリントサイズを表す。従って fs100 はアクセス回数全体の 100%が収まるフットプリントサイズを表す。すなわちプログラムのフットプリントサイズそのものである。fs90 はアクセス回数全体の 90%が収まる修正フットプリントサイズであり、キャッシュサイズを決定する上での重要なパラメータとなる。

表 6.1: タスクプロファイルパラメータ

	パラメータ	値
1	連続アクセスの平均長	10(ステップ)
2	分岐距離	40(ステップ)
3	データアクセスの頻度	5(ステップ)
4	上方向分岐の確率	90(%)
5	データアクセスの範囲	1024(バイト)
6	総ステップ数	100000(ステップ)

アドレストレースの重要なプロファイルとしてブロック (4 ワード/ブロック) に対するアクセス回数の累積分布を図 6.2 から図 6.9 に示す。ブロックサイズは一つのフットプリントと同じサイズである。

表 6.2: タスクのフットプリントサイズ

タスク	fs100	fs90	アドレス範囲		オフセット	
	fs*	fs*	バイト	fs*	バイト	fs*
1	192	84	4008	250	0	0
2	213	90	4008	250	6000	375
3	201	85	4005	250	12000	750
4	188	71	3999	249	18000	1125
5	251	93	4001	250	24000	1500
6	220	99	4001	250	30000	1875
7	199	82	3998	249	36000	2250
8	173	78	4001	250	42000	2625
平均	204.6	85.3	4002.6	249.8	—	—
合計	1637	682	32021	1998	168000	10500

fs*はフットプリントサイズを表す。本実験ではフットプリントサイズ 1 は 16 バイトとする。

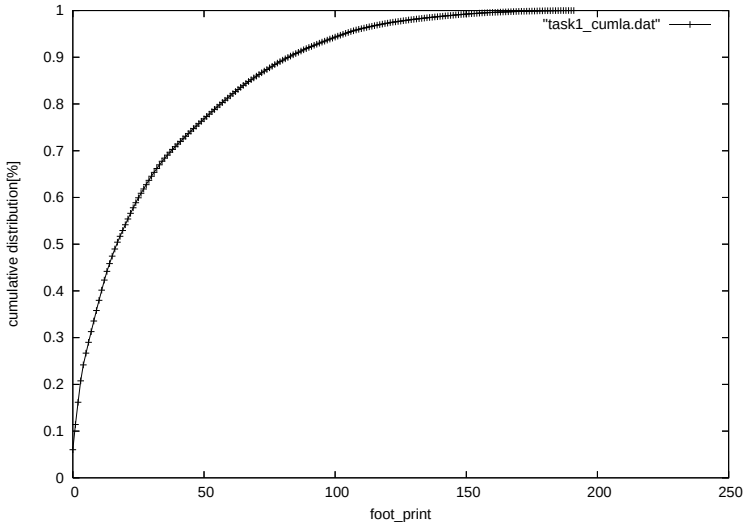

図 6.2: タスク 1 のアクセス回数の累積分布

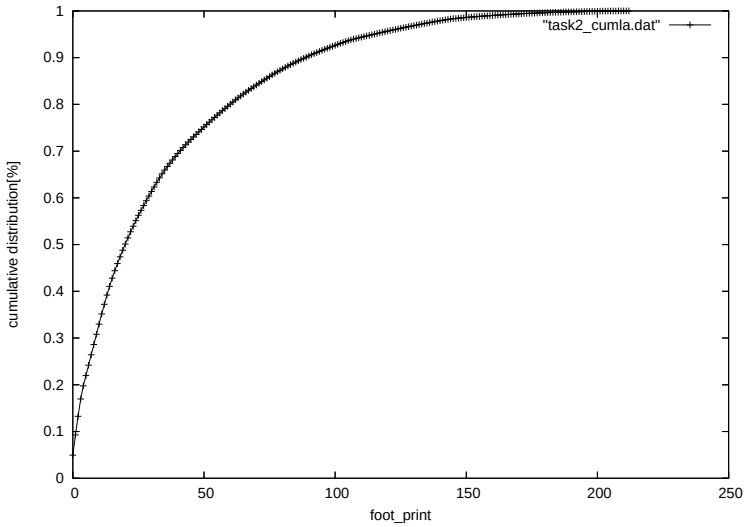

図 6.3: タスク 2 のアクセス回数の累積分布

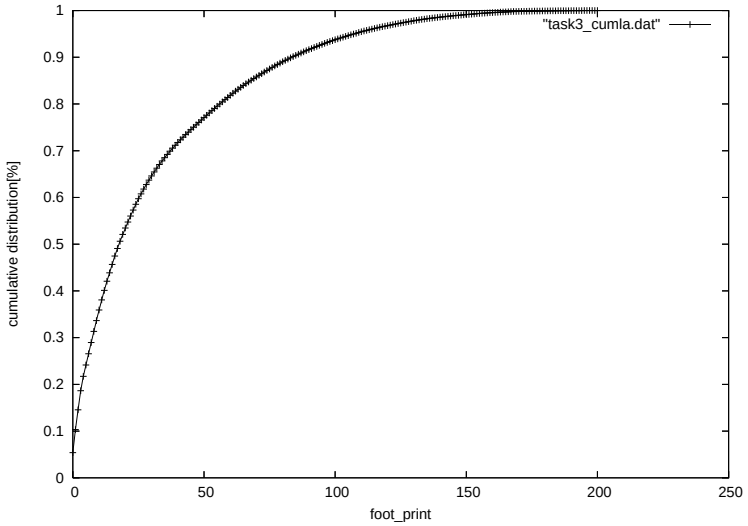

図 6.4: タスク 3 のアクセス回数の累積分布

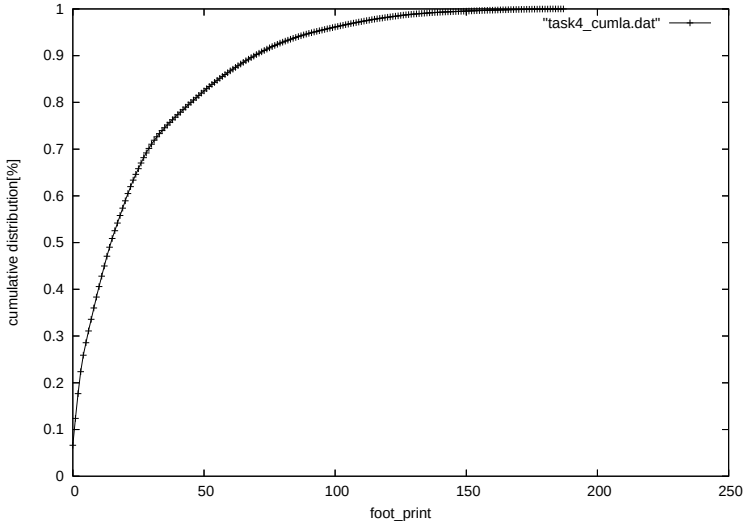

図 6.5: タスク 4 のアクセス回数の累積分布

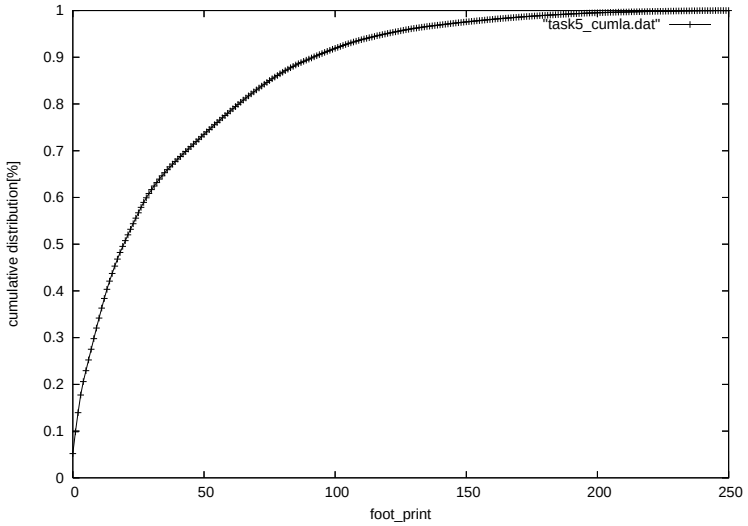

図 6.6: タスク 5 のアクセス回数の累積分布

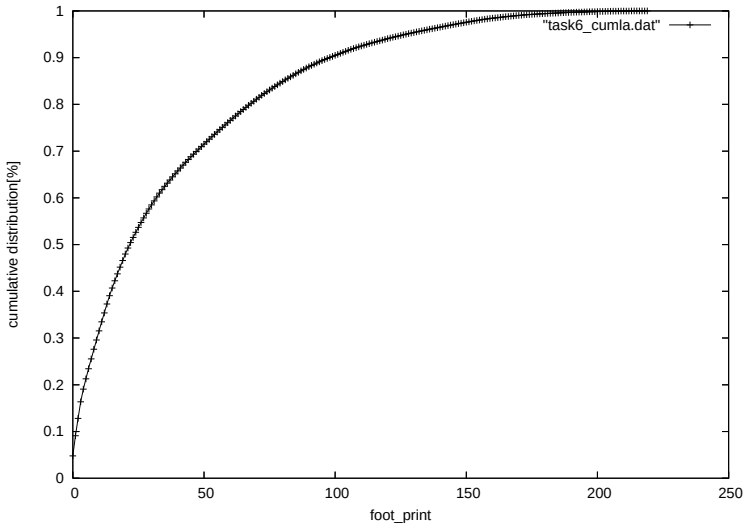

図 6.7: タスク 6 のアクセス回数の累積分布

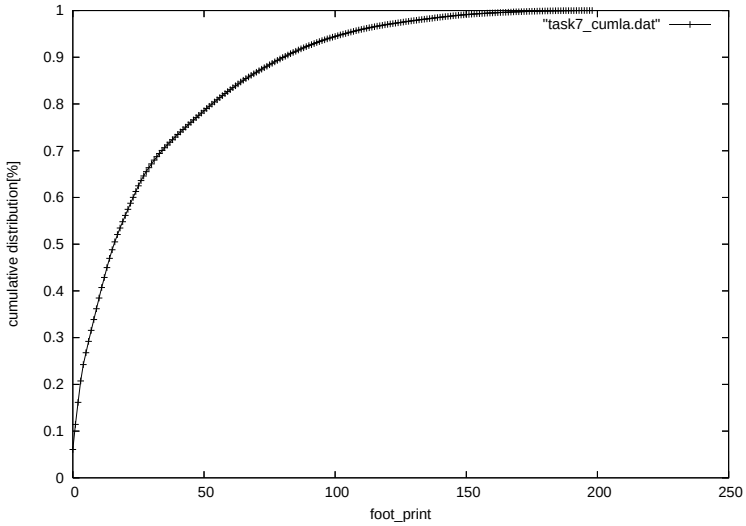

図 6.8: タスク 7 のアクセス回数の累積分布

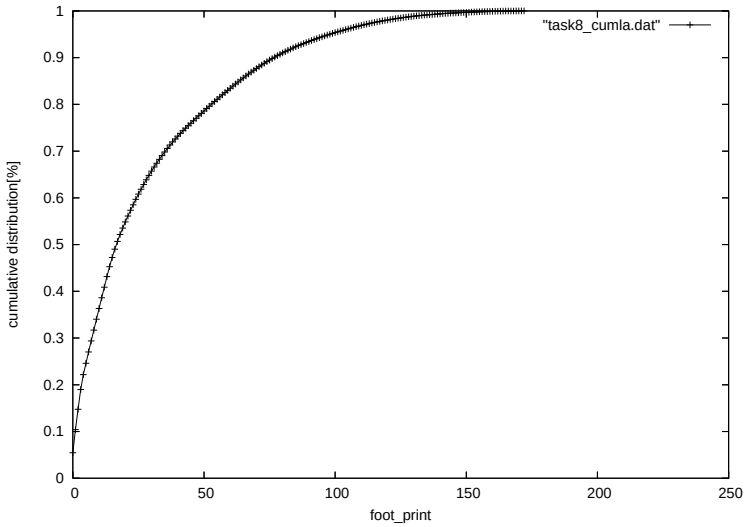

図 6.9: タスク 8 のアクセス回数の累積分布

6.2.2 タスクの主メモリ領域の配置

図 6.10 で示すように各タスクのアドレストレースにオフセット (プログラムサイズ以上) を加え、主メモリ上にそれぞれ異なる領域に配置させる。また、各々のタスクの領域内で命令とデータはそれぞれ異なる領域に保存される。

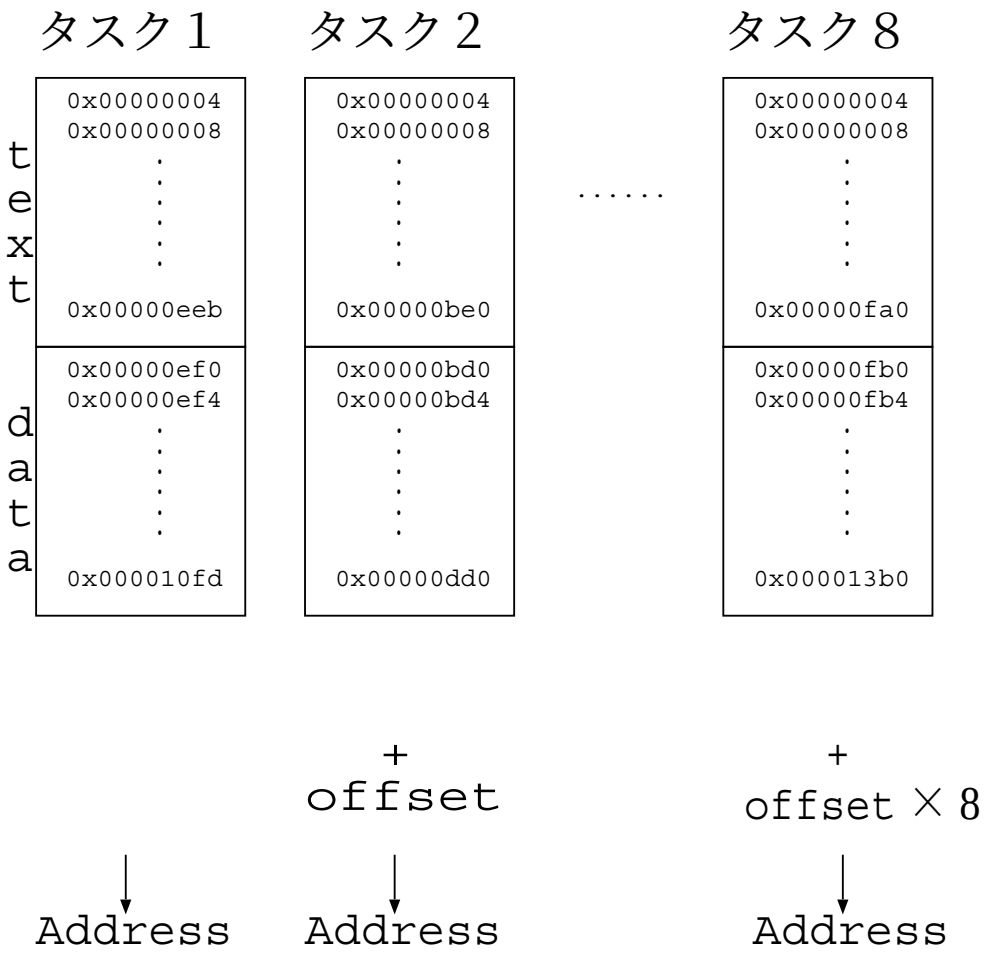

図 6.10: メモリ内の各タスク配置

6.3　キャッシュ容量

図 3.2 で示したミスモデルや表 6.1 で示したタスクのプロファイルを基に用意されたタスクセットに対して最適なキャッシュ容量を見積る。

6.3.1　キャッシュ容量の見積り

キャッシュ容量は、キャッシュリソースに対する静的な優先度を High に割り付けたタスクの fs100 と、Low に割り付けたタスクの fs90 の合計値よりも若干小さめのサイズを用意することにする。キャッシュリソースに対する静的優先度を High に割り付けたタスクの fs100 の合計値、Low に割り付けたタスクの fs90 の合計値、及びタスク全体のフットプリントサイズを表 6.3 に示す

High の fs100 と Low の fs90 の合計は 1138 となる。従って今回の実験ではキャッシュ容量は 16384 バイト、すなわちフットプリントサイズで 1024 のキャッシュ容量を用意することにする。

表 6.3: フットプリントサイズ

	静的	fs100	fs90	
1	Low	–	84	
2	High	213	–	
3	Low	–	85	
4	High	188	–	
5	Low	–	93	
6	High	220	–	
7	Low	–	82	
8	High	173	–	
合計		794	344	1138

6.4　タスクの優先度割り当て

本研究ではアドレスジェネレータで生成されたタスクに対して、任意に優先度の割り当てを行う。表 6.3 で示すように、キャッシュのリソースに対する優先度の割り当ては、簡単のため、偶数番号のタスクに対して High を、奇数番号には Low を割り付けた。また走行頻度に基づく動的優先度に関しても、ある閾値よりも高いなら High、低いのなら Low に割り付けた。表 6.4 に結果を示す。

キャッシュのリソースに対する静的優先度と走行頻度に対する動的優先度を組み合わせた場合における優先度は表 6.5、各タスクの優先度は表 6.6 に示す。

表 6.4: キャッシュのリソースに対する静的な優先度の一覧

タスク番号	静的優先度	走行頻度	動的優先度
1	Low	High	(High)
2	High	High	(High)
3	Low	High	(High)
4	High	Low	(Low)
5	Low	Low	(Low)
6	High	Low	(Low)
7	Low	Low	(Low)
8	High	Low	(Low)

表 6.5: 優先度レベル

キャッシュリソースの静的優先度	走行頻度の動的優先度	優先度レベル
High	High	4
High	Low	3
Low	High	2
Low	Low	1

表 6.6: 各タスクの優先度レベル

タスク番号	静的	動的	優先度レベル
1	Low	High	2
2	High	High	4
3	Low	High	2
4	High	Low	3
5	Low	Low	1
6	High	Low	3
7	Low	Low	1
8	High	Low	3

本実験では表 6.1 をパラメータとして、アドレスジェネレータにより 8 つのタスクのアドレストレースを生成した。乱数の範囲とタスクスケジューリングに関する情報を表 6.7 に示す。

スケジューリングは乱数を用いて、ランダムスケジューリングを行うことで疑似リアルタイムシステムをシミュレートし、そのアドレストレースで提案したキャッシュシステムを評価した。タスク 1、2、3 の走行頻度を高めるため、スケジューリングのための乱数の範囲は 0 10 とし、modulo8 をとり、0、1、2 の発生頻度を 2 倍にしている。各々のタスクの重要なプロファイルは図 6.2 から図 6.9 に示す。

キャッシュサイズは表 6.4 で見積った 16384 バイト、フットプリントサイズで 1024 である。またキャッシュのリソースに対する静的な優先度の割り当ては表 6.4 に示した通りである。

表 6.7: タスクスケジューリング

項目		ランダムの範囲
タスク切替え回数	1000	—
タスク数	8	—
スケジューリング方針	ランダム	0—10 (mod8)
割り込み発生頻度	5000(ステップ)	0—9999

6.5 実験結果

本論文では、タスク 1、タスク 2、タスク 3 は他のタスクに比べて 2 倍多くスケジューリングされるように設定している。

6.5.1 通常の LRU で置換制御を行った場合

通常の LRU で置換制御を行った場合のミス回数と way 数 (連想度) の関係の実験結果を図 6.11 に、ミス率と way 数 (連想度) の関係の実験結果を図 6.12 に示す。また、タスク毎のミス回数と way 数 (連想度) の関係の実験結果を表 6.8 に示す。

タスク 1、タスク 2、タスク 3 は他のタスクに比べて 2 倍多くスケジューリングされるため、タスク 1、タスク 2、タスク 3 のミス率が他のタスクのミス率に比べて低くなっている。

表 6.8: タスクのミス回数

way (連想度)	タスク番号							
	1	2	3	4	5	6	7	8
1	6318	6590	3862	924	1538	4427	5988	5458
2	3410	3971	2721	1831	2293	3891	3639	2977
4	2584	2256	2620	2713	2748	3207	2304	2665
8	2410	2589	2455	2743	2862	3136	2445	2530
16	2255	2316	2244	2800	2915	3185	2610	2788
32	2262	2380	2280	2856	2955	3223	2570	2871
64	2240	2416	2306	2889	3015	3279	2545	2906
128	2237	2442	2288	2884	3029	3301	2516	2919
512	2221	2463	2291	2894	3047	3319	2532	2911
1024	2227	2472	2282	2897	3057	3315	2514	2933

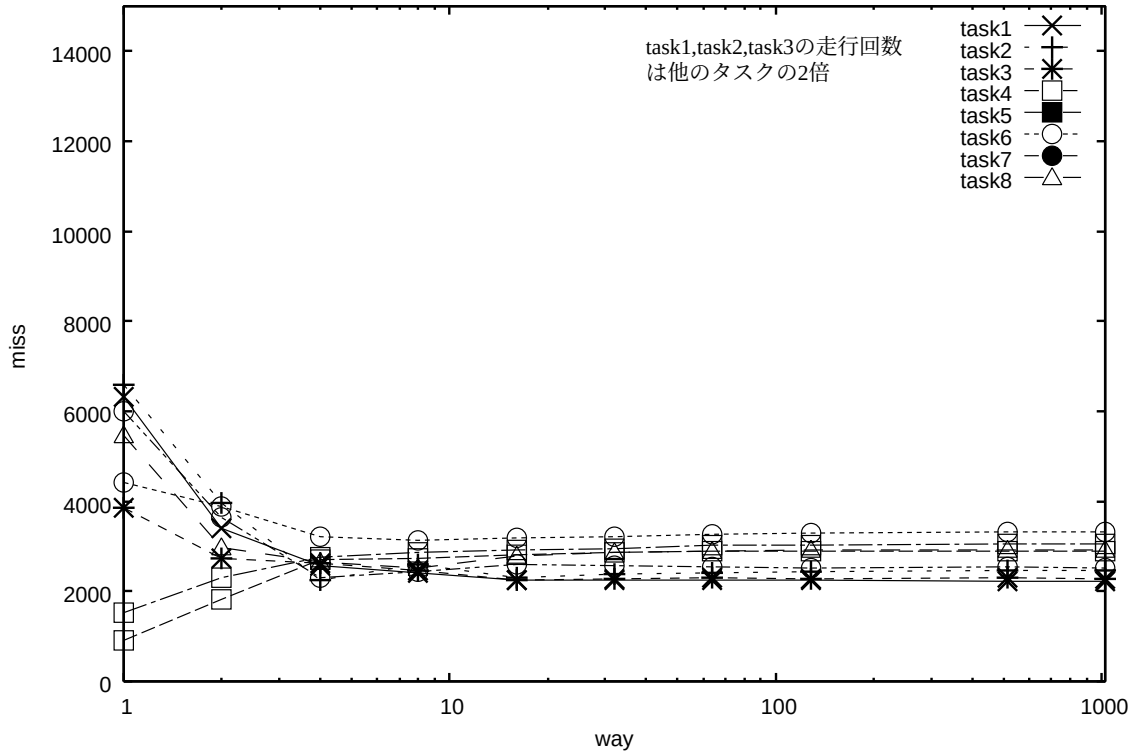

図 6.11: 通常の LRU で置換制御を行った場合のミス回数

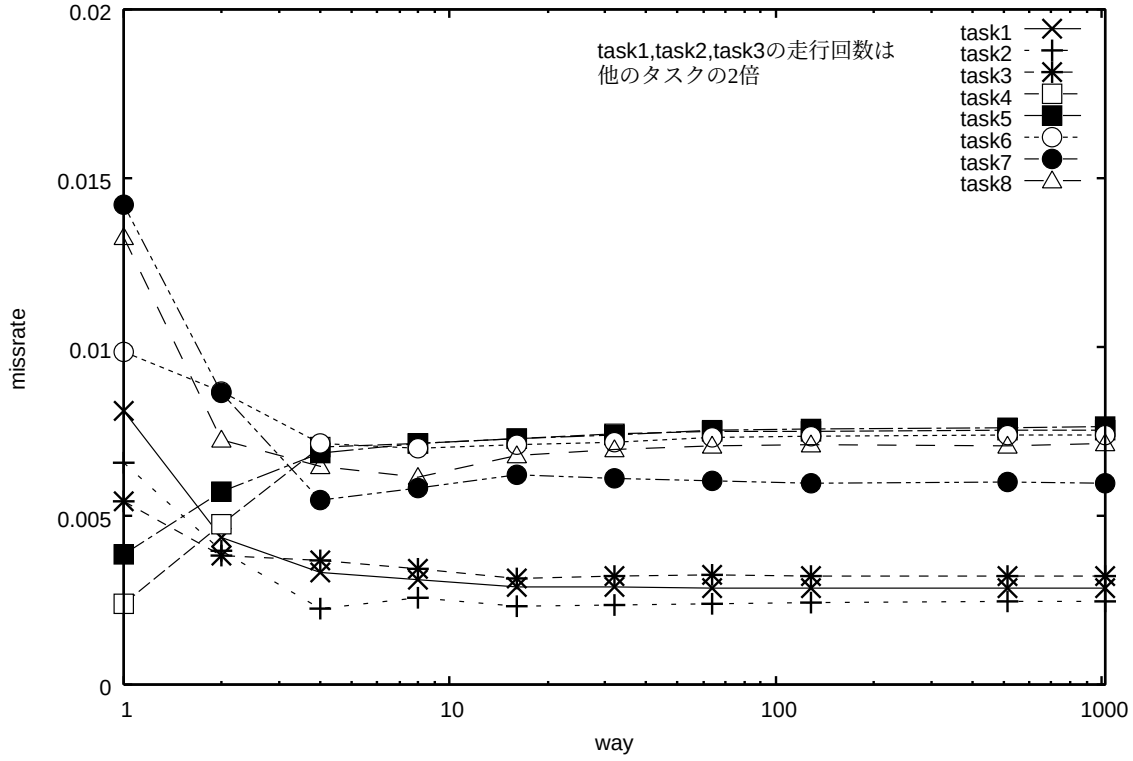

図 6.12: 通常の LRU で置換制御を行った場合のミス率

6.5.2 キャッシュのリソースに対する静的優先度に基づいた置換制御を行った場合

キャッシュのリソースに対する静的な優先度のレベルは High、Low の 2 種類とした。まず各タスクのキャッシュに対する静的な優先度のレベルを格納する配列を用意し、各タスクのキャッシュのリソースに対する優先度のレベルを予め格納しておく。優先度が High に割り与えられたタスクのラインは追い出さないように置換制御を行う。ミス回数と way 数 (連想度) の関係の実験結果を図 6.13、ミス率と way 数 (連想度) の関係の実験結果を図 6.14 に示す。また、タスク毎のミス回数と way 数 (連想度) の関係の実験結果を表 6.9 に示す。

タスク 1、タスク 2、タスク 3 は他のタスクに比べて多くスケジューリングされるため、静的優先度 Low のタスク 1、タスク 3 のミス率が他の静的優先度 Low のタスクのミス率に比べて低くなっている。

表 6.9: タスクのミス回数

way (連想度)	タスク番号							
	1	2	3	4	5	6	7	8
1	6318	6590	3862	924	1538	4427	5988	5458
2	6744	2536	4572	1260	4573	2659	5337	2118
4	8006	470	8250	997	6998	808	5943	877
8	8054	229	7659	402	7077	408	5862	268
16	9017	238	8360	202	7735	234	6364	192
32	8727	213	8274	194	7921	240	6275	188
64	8747	213	8404	189	7956	223	6316	188
128	8715	213	8402	189	7959	220	6309	190
512	8709	213	8394	188	7983	220	6302	191
1024	8708	213	8392	188	7985	220	6298	191

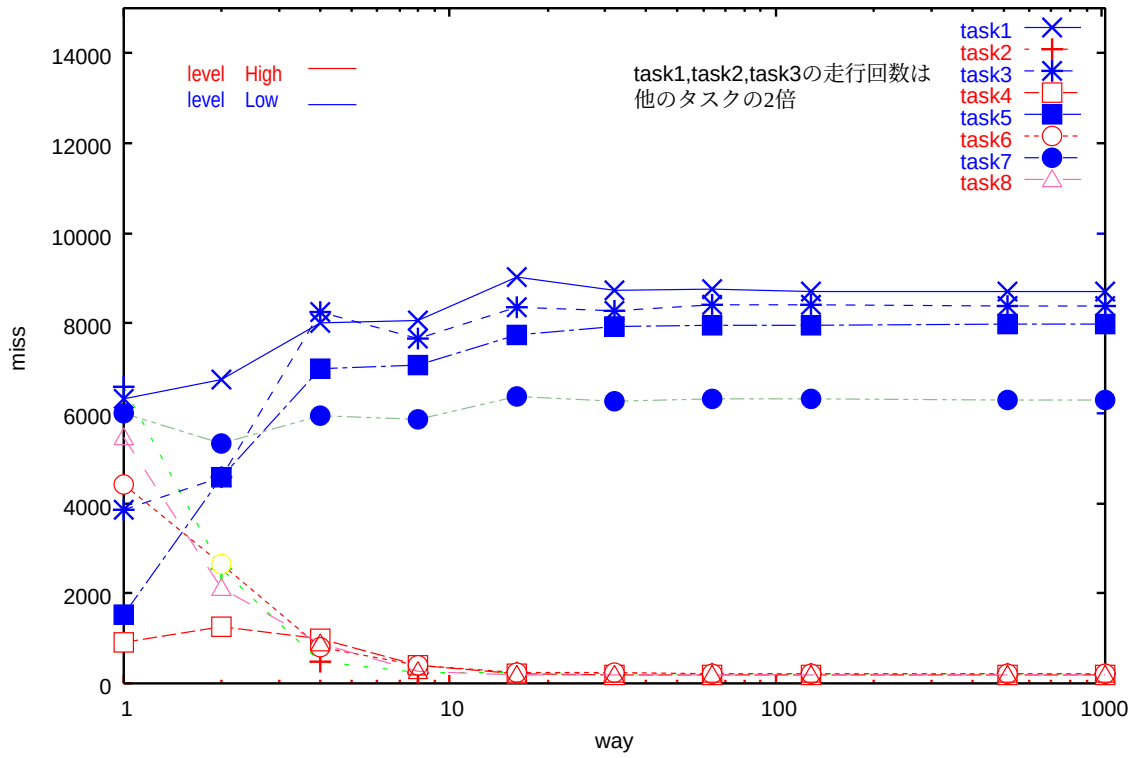

図 6.13: 静的な優先度に基づいた置換制御を行った場合のミス回数

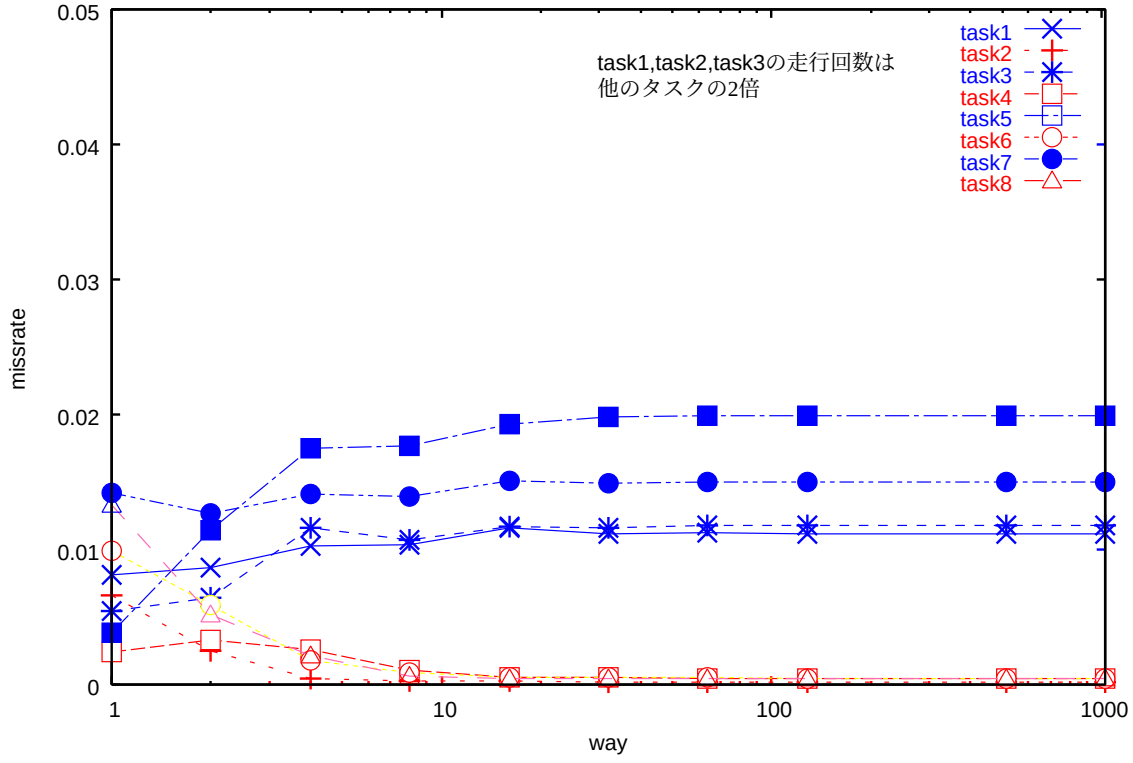

図 6.14: 静的な優先度に基づいた置換制御を行った場合のミス率

6.5.3 タスクの走行回数に対する動的優先度に基づいた置換制御を行った場合

走行頻度に対する動的な優先度のレベルは High、Low の 2 種類までとした。まず各タスクの走行回数を格納する配列と、走行回数に対する優先度のレベルを格納する配列を用意する。タスクが切り替わるタイミングで、切り替わったタスクの走行回数をインクリメントしていく。本実験ではタスクスイッチが 200 回起る度に配列を 0 に更新する。更新の度にまず配列内の各タスクの走行回数の平均値を求め、それを閾値とする。次にこの閾値と各タスクの走行回数を比較し閾値以上であるならばそのタスクの走行頻度を High、低ければ Low と、動的な優先度を格納する配列に設定する。ミス回数と way 数 (連想度) の関係の実験結果を図 6.15、ミス率と way 数 (連想度) の関係の実験結果を 6.16 に示す。また、タスク毎のミス回数と way 数 (連想度) の関係の実験結果を表 6.10 に示す。
タスク 1、タスク 2、タスク 3 は他のタスクに比べて 2 倍多くスケジューリングされるため、優先度に基づく置換制御によりタスク 1、タスク 2、タスク 3 のミス回数、及びミス率が低く抑えられている。

表 6.10: タスクのミス回数

way (連想度)	タスク番号							
	1	2	3	4	5	6	7	8
1	6318	6590	3862	924	1538	4427	5988	5458
2	2314	2587	1597	2774	2769	4557	3746	4243
4	519	694	721	4341	4262	5279	2752	4823
8	489	527	685	4138	4689	5021	3140	4351
16	398	442	643	3965	4452	4937	3210	4255
32	393	422	678	3888	4352	4909	3235	4306
64	383	412	688	3883	4308	4943	3282	4300
128	379	409	689	3883	4303	4944	3310	4302
512	377	408	689	3885	4295	4943	3322	4304
1024	379	407	689	3884	4290	4943	3325	4303

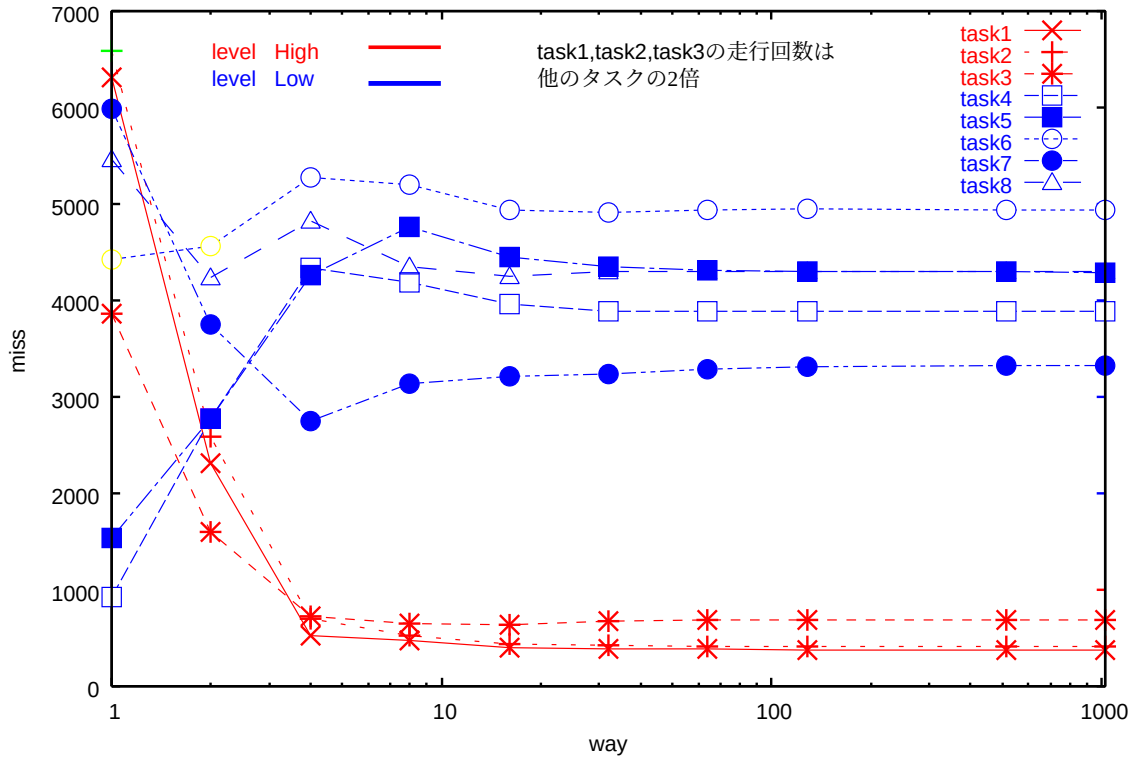

図 6.15: 動的な優先度に基づいた置換制御を行った場合のミス回数

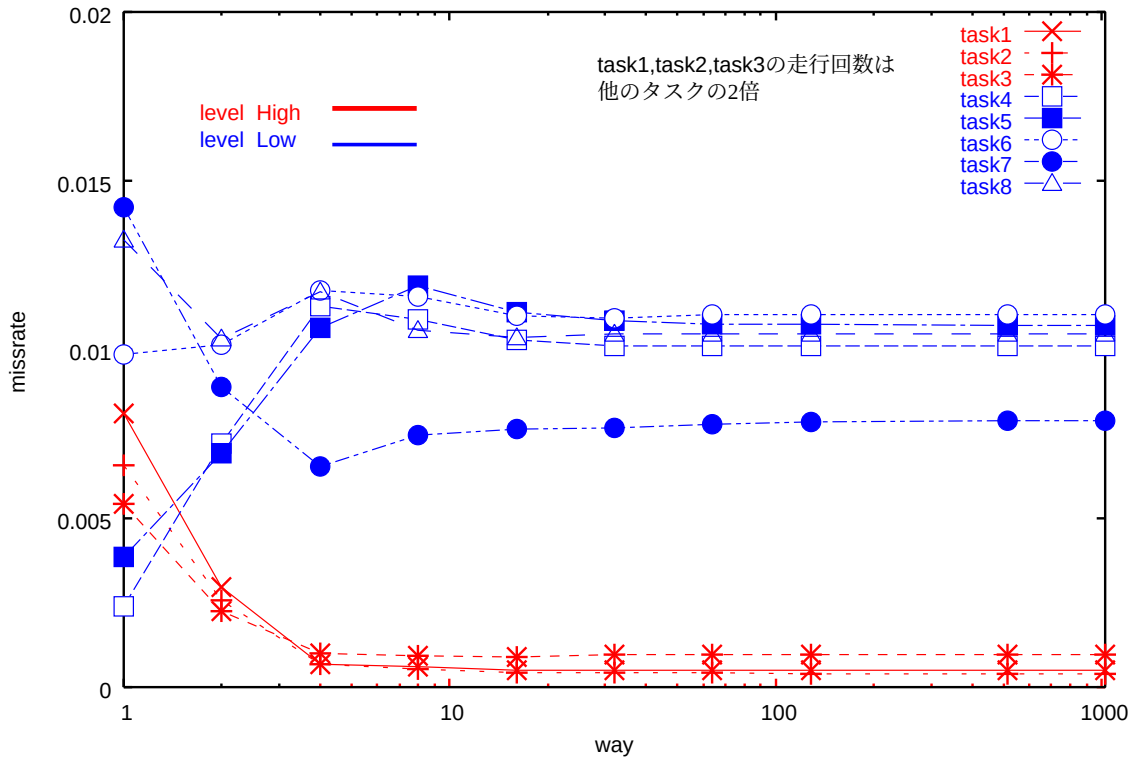

図 6.16: 動的な優先度に基づいた置換制御を行った場合のミス率

6.5.4 静的優先度と動的優先度の組み合せに基づいた置換制御を行った場合

表 5.1 によって走行頻度に対する動的な優先度のレベルと、キャッシュのリソースに対する静的な優先度のレベルの組み合わせとから、タスクの優先度のレベルを決定する。置換アルゴリズムに従い、ラインを追い出すように置換制御を行う。ミス回数と way 数 (連想度) の関係の実験結果を図 6.17、ミス率と way 数 (連想度) の関係の実験結果を図 6.18 に示す。また、タスク毎のミス回数と way 数 (連想度) の関係の実験結果を 6.11 に示す。

タスク 1、タスク 2、タスク 3 は他のタスクに比べて 2 倍多くスケジューリングされるため、タスク 1、タスク 3 のミス率がタスク 5、タスク 7 のミス率よりも低くなる。

表 6.11: タスクのミス回数

way (連想度)	タスク番号							
	1	2	3	4	5	6	7	8
1	6318	6590	3862	924	1538	4427	5988	5458
2	6669	541	4295	1372	4509	3394	5172	2421
4	7450	224	7839	1060	7463	949	6039	964
8	7268	213	6839	402	7793	416	6242	271
16	8314	214	7405	222	8301	233	6620	192
32	7941	213	7206	194	8488	240	6541	188
64	7940	213	7282	189	8540	223	6594	188
128	7909	213	7262	189	8557	220	6594	190
512	7907	213	7259	188	8562	220	6593	191
1024	7906	213	7263	188	8562	220	6596	191

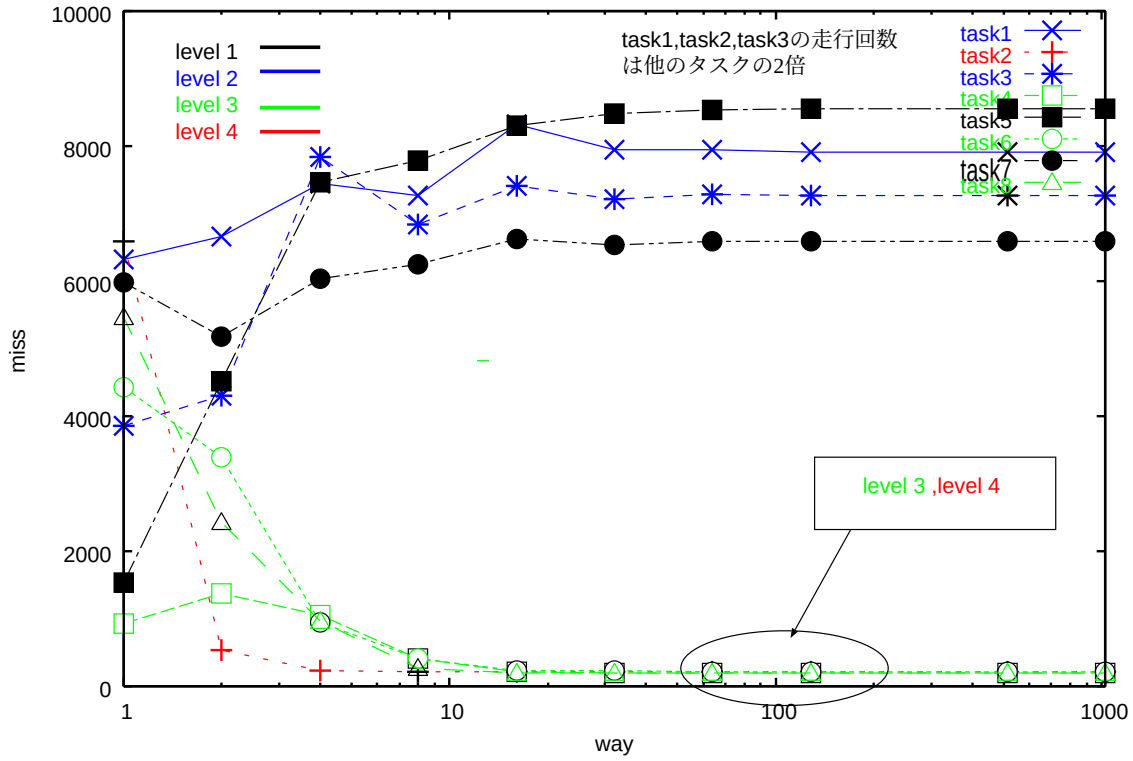

図 6.17: 動的な優先度と静的な優先度の組み合せに基づいた置換制御を行った場合のミス回数

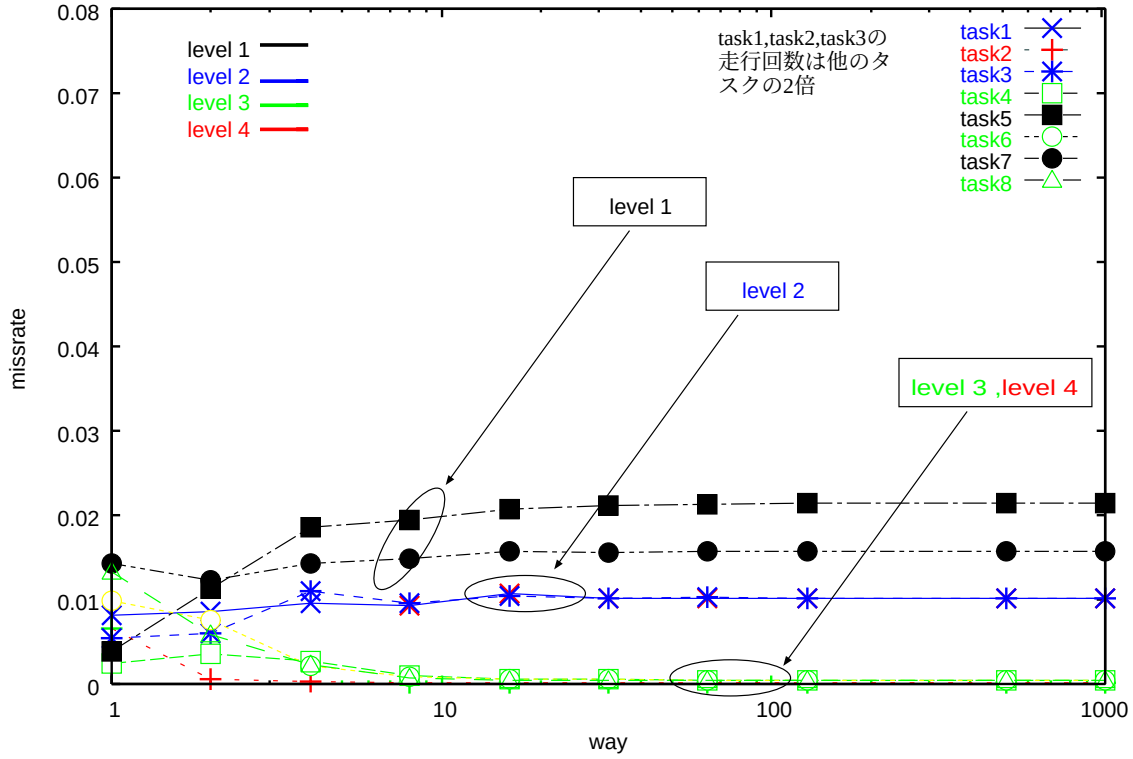

図 6.18: 動的な優先度と静的な優先度の組み合せに基づいた置換制御を行った場合のミス率

第7章　考察

7.1　通常のLRUで置換制御を行った場合

通常のLRUで置換制御を行った場合、連想度8以上でミス回数及びミス率が低く安定する。理由として、タスクセットのfs90の合計フットプリントサイズが682なので、これは本実験で用意したキャッシュ(フットプリント換算で1024)に収まる。従って、連想度を8以上に設定するとタスクセットのfs90の合計フットプリントがオンキャッシュし、高い安定したヒット率が得られる。

7.2　キャッシュのリソースに対する静的優先度に基づいた置換制御を行った場合

表6.4で示されるように、キャッシュのリソースに対する優先度のレベルを、各タスクに静的に割り付けて、その優先度のレベルを基に置換制御を行う。この場合タスク2、タスク4、タスク6、タスク8に対してキャッシュのリソースに対する優先度をHighに設定する。

これらのタスクの合計フットプリントは794で用意したキャッシュサイズよりも小さい。すなわちこれらのタスクはこの置換制御法で常にオンキャッシュできる。実際、図6.13から明らかなように連想度を徐々に上げていくと、優先度Highに割り与えられたタスクはほぼ初期ミスに等しくなる。この場合、優先度Highのfs100によって794のラインが占められ、優先度Lowのタスクには実質230のキャッシュリソースしか割り与えられず、この少ないリソースをめぐって優先度Lowのレベルのタスクが競合する。このためLowに設定されたタスクのミス回数が通常のLRUの場合よりも増えしまう。

7.3　タスクの走行頻度に対する動的優先度に基づいた置換制御を行った場合

図6.15からタスク1、2、3のミス回数, 及びミス率を低く抑えられている。この理由として本実験では、タスク1、タスク2、タスク3には多くスケジューリングされるように

予め設定し、走行回数の高いタスクには、高い優先度を与えてキャッシュ内にレジデントするように置換制御を行ったからである。実際走行回数が高いタスクは、実際のシステムにおいて優先度が高いタスクと考えられる。ここでは走行回数の高いタスクの優先度をHigh に、そうでなければ Low に設定した。そして High に設定されたタスクを残すように置換制御を行ったため、タスク 1、2、3 のミス回数及びミス率が低く抑えられたということである。

図 6.15 から残したい優先度のタスクを、自動的にほとんどオンキャッシュできることを示している

この結果から実際のリアルタイムシステム上で動作する優先度の高いアプリケーションのラインを、自動的に残すことができるということが可能であるということがシミュレーションによって示された。

7.4 静的優先度と動的優先度の組み合せに基づいた置換制御を行った場合

この場合、優先度のレベルは 4 つに分類され、ミス率の結果では優先度のレベル順毎に低いミス率となった。また高優先度 (レベル 3、レベル 4) のタスクに関しては、ミス回数がほぼ初期ミスに等しくなった。特に最高優先度 (レベル 4) のタスクと、レベル 3 のタスク群の 2 タスクは完全にそのタスクのフットプリントサイズと一致した。つまり完全にオンキャッシュできたということになる。

第8章　シミュレーション実験と評価(2)

8.1　実験概要

実験概要は図6.1と同様の構成であるが、タスク数を倍の16個に増加してある。また、増加分のタスクは表8.1と同様のパラメータを用いて生成した。タスク毎のフットプリントサイズの一覧を表8.1に示す。また、キャッシュリソースに対する静的優先度と動的優先度の組み合わせた場合の優先度を表8.2、各タスクの優先度は表8.3に示す。

表 8.1: タスクのフットプリントサイズ

タスク	fs100	fs90	アドレス範囲		オフセット	
	fs*	fs*	バイト	fs*	バイト	fs*
1	192	84	4008	250	0	0
2	213	90	4008	250	6000	375
3	201	85	4005	250	12000	750
4	188	71	3999	249	18000	1125
5	251	93	4001	250	24000	1500
6	220	99	4001	250	30000	1875
7	199	82	3998	249	36000	2250
8	173	78	4001	250	42000	2625
9	205	92	3994	249	48000	3000
10	219	109	3996	249	54000	3375
11	185	85	3997	249	60000	3750
12	179	86	4005	250	68000	4250
13	170	78	4005	250	74000	4625
14	224	95	3996	249	80000	5000
15	181	78	3999	249	86000	5375
16	229	98	3999	249	92000	5750
平均	201.8	87.7	3750.8	349.5	6000	375
合計	3229	1403	60013	3992	96000	6000

表 8.2: 優先度レベル

キャッシュリソースの静的優先度	走行頻度の動的優先度	優先度レベル
High	High	6
High	Low	5
Mid	High	4
Mid	Low	3
Low	High	2
Low	Low	1

表 8.3: 各タスクの優先度レベル

タスク番号	静的	動的	優先度レベル
1	Low	High	2
2	High	High	6
3	Low	High	2
4	High	Low	5
5	Low	Low	1
6	High	Low	5
7	Low	Low	1
8	High	Low	5
9	Low	Low	1
10	Mid	Low	3
11	Low	Low	1
12	Mid	Low	3
13	Low	Low	1
14	Mid	Low	3
15	Low	Low	1
16	Mid	Low	3

8.2 実験結果

この場合もタスク 1，タスク 2、タスク 3 は他のタスクに比べて 2 倍多くスケジューリングされるように設定してある。

8.2.1 通常の LRU で置換制御を行った場合

タスク１，タスク２、タスク３は通常のタスクよりも２倍多くスケジュールされているため、図 8.2 のミス率で見るとタスク１，タスク２のミス率が他のタスクに比べて低くなっている。タスク３だけ少し離れてしまったのは、実験 (1) と同じスケジューリング回数に対してタスク数が増えるので、乱数の発生に変化が生じたためと考えられる。

8.2.2 静的優先度に基づいた置換制御を行った場合

キャッシュリソースに対する静的優先度を High、Mid、Low の 3 種類用意した。静的優先度が High のタスクに関しては完全にほぼミスに収まった。また、図 8.4 のミス率の結果では、静的優先度 Mid、Low のミス率は分離せずに同じグループに収まった。

8.2.3 動的優先度の組み合わせに基づいた置換制御を行った場合

この場合もタスク１，タスク２、タスク３は通常のタスクよりも２倍多くスケジューリングされているため、図 8.6 のミス率で見ると他のタスクよりも低く押さえられている。若干タスク３が高いのは、実験 (1) と同じスケジューリング回数に対してタスク数が増えるので、乱数の発生に変化が生じたためと考えられる。

8.2.4 静的優先度と動的優先度の組み合わせに基づいた置換制御を行った場合

図 8.8 のミス率で見ると、静的優先度 High と静的優先度 Mid、Low の２種類に分離し、静的優先度 Mid、Low は同じグループに収まった。

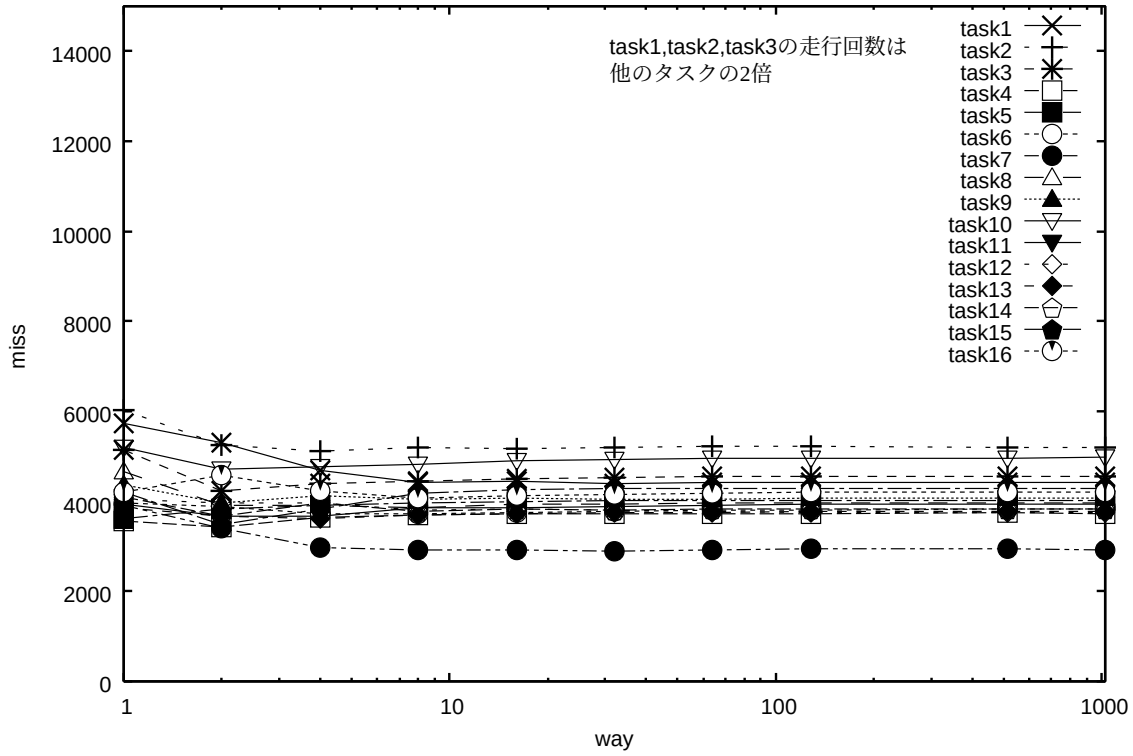

図 8.1: 通常の LRU で置換制御を行った場合のミス回数

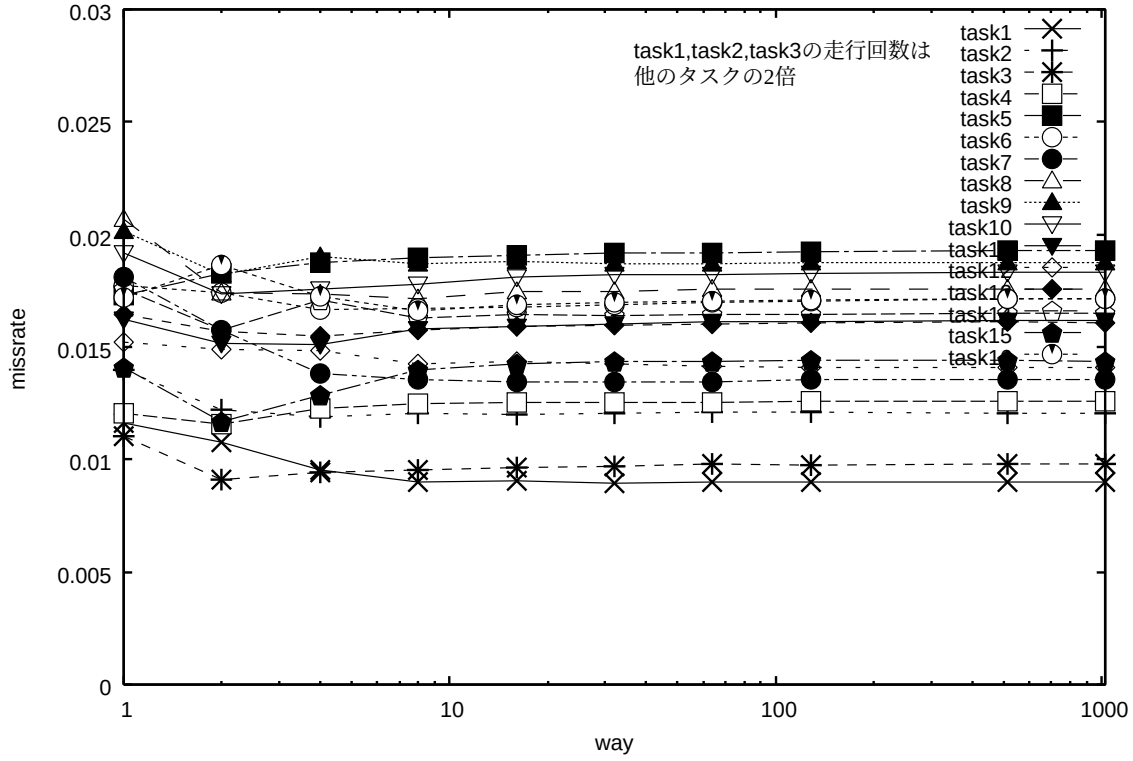

図 8.2: 通常の LRU で置換制御を行った場合のミス率

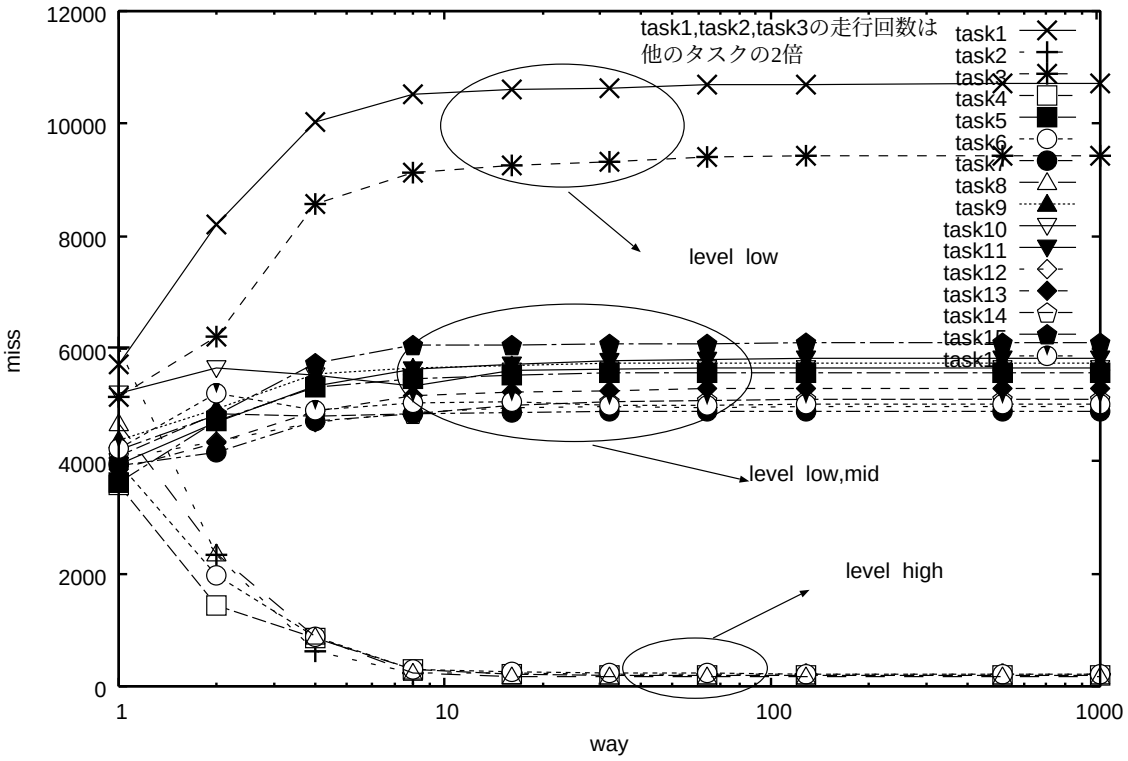

図 8.3: 静的な優先度に基づいた置換制御を行った場合のミス回数

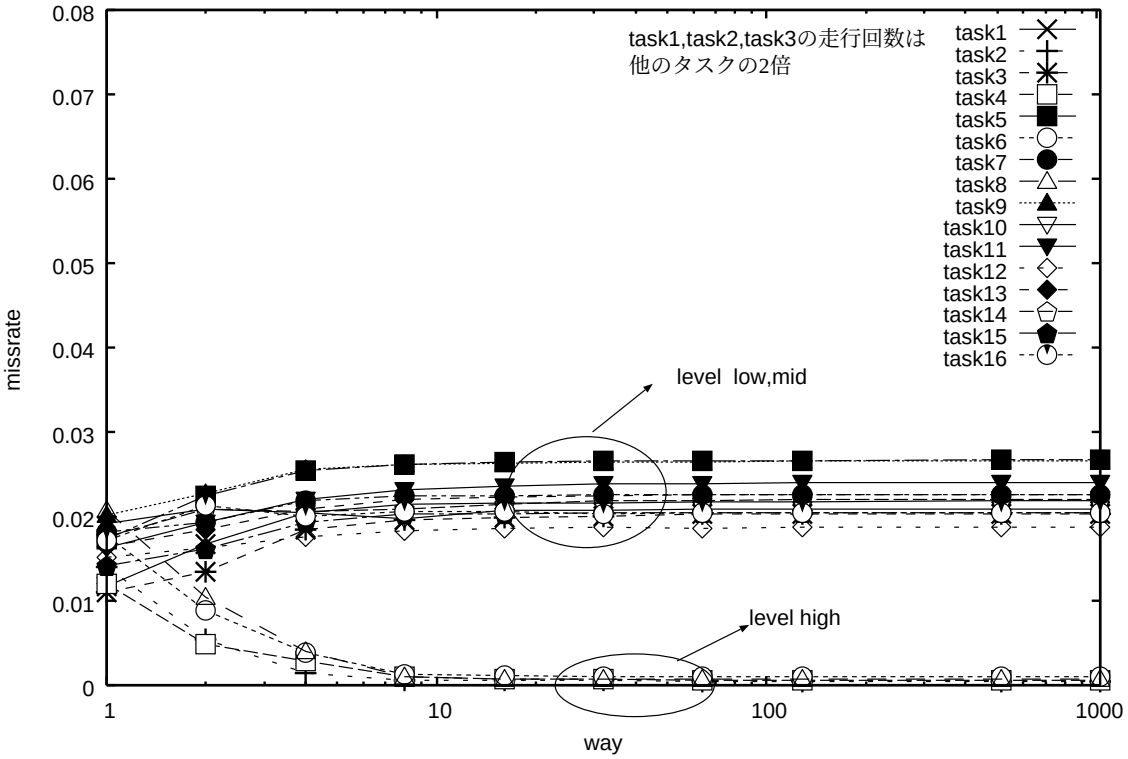

図 8.4: 静的な優先度に基づいた置換制御を行った場合のミス率

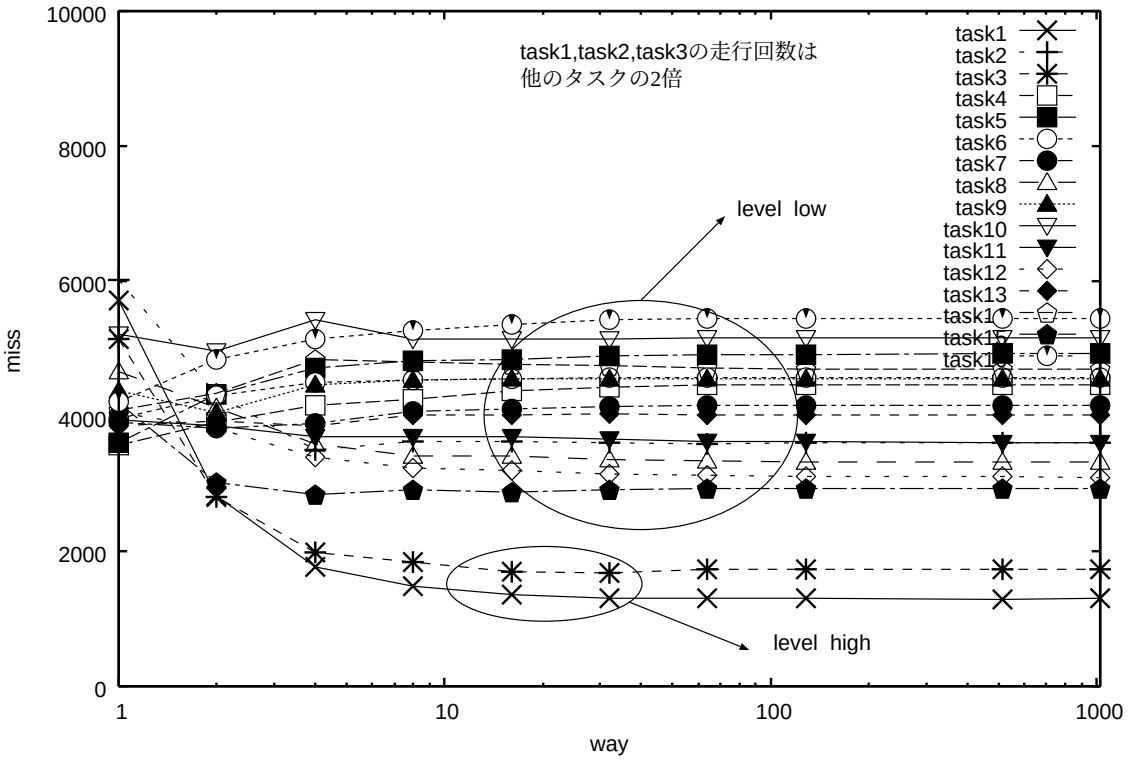

図 8.5: 動的優先度に基づいた置換制御を行った場合のミス回数

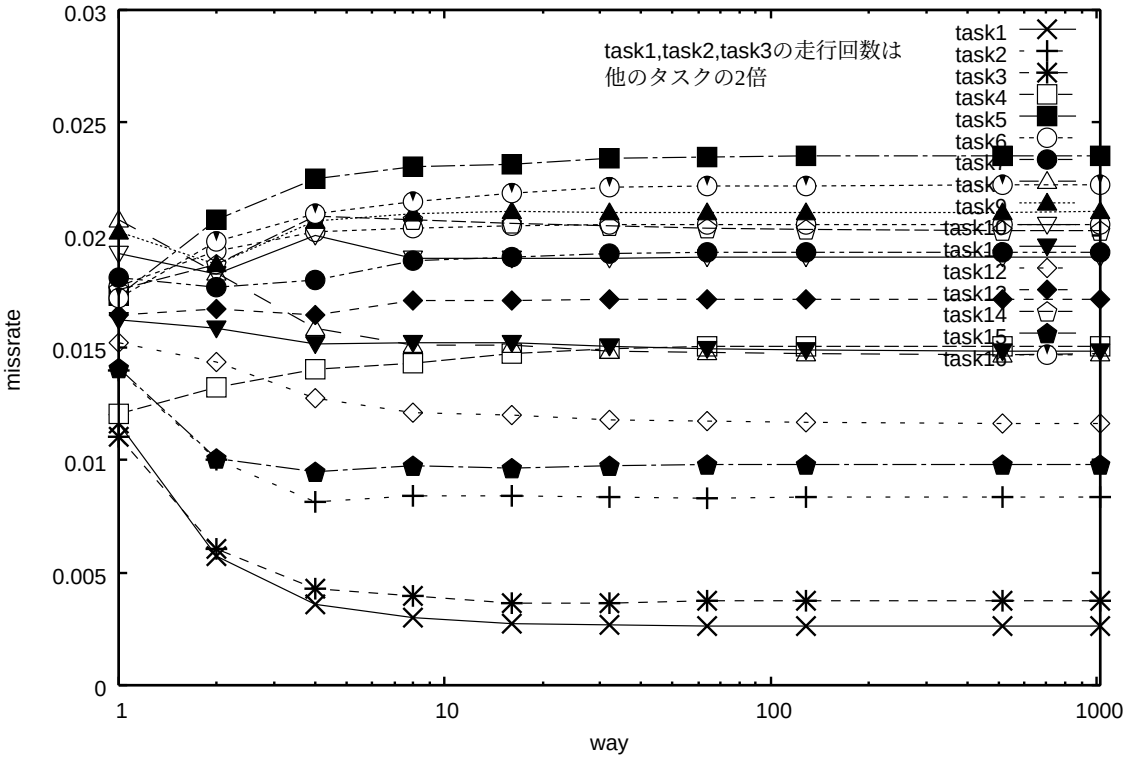

図 8.6: 動的優先度に基づいた置換制御を行った場合のミス率

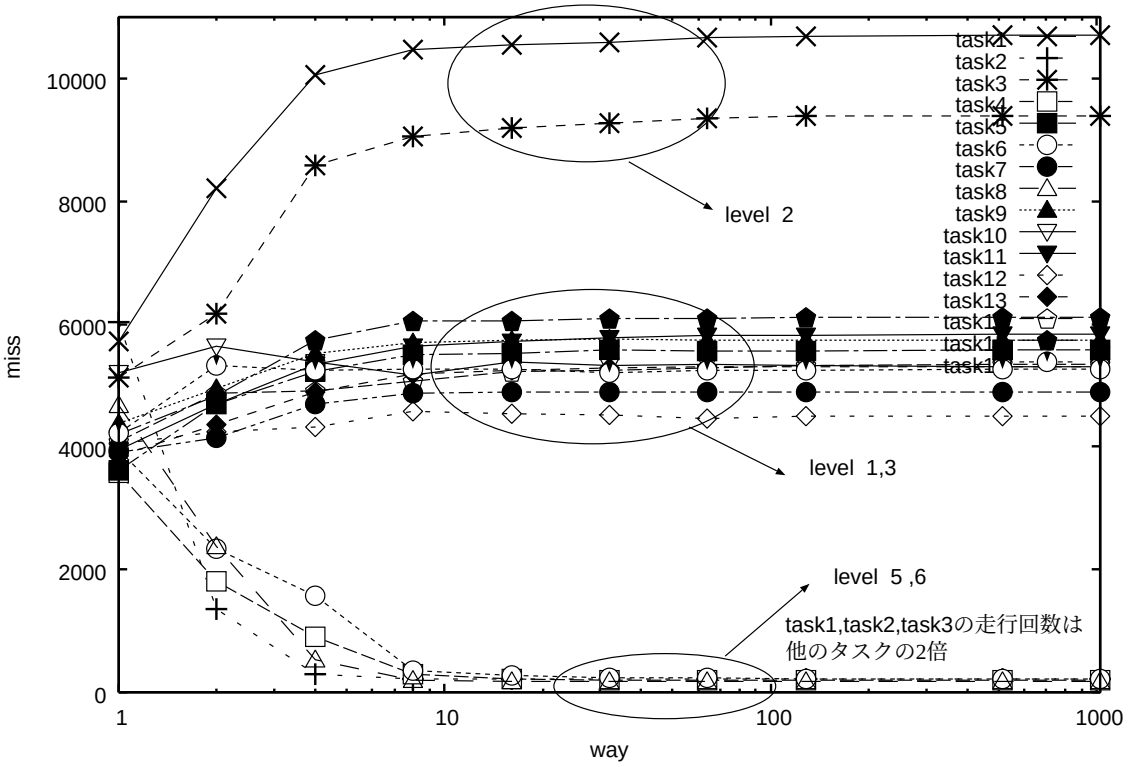

図 8.7: 動的な優先度と静的な優先度の組み合わせに基づいた置換制御を行った場合のミス回数

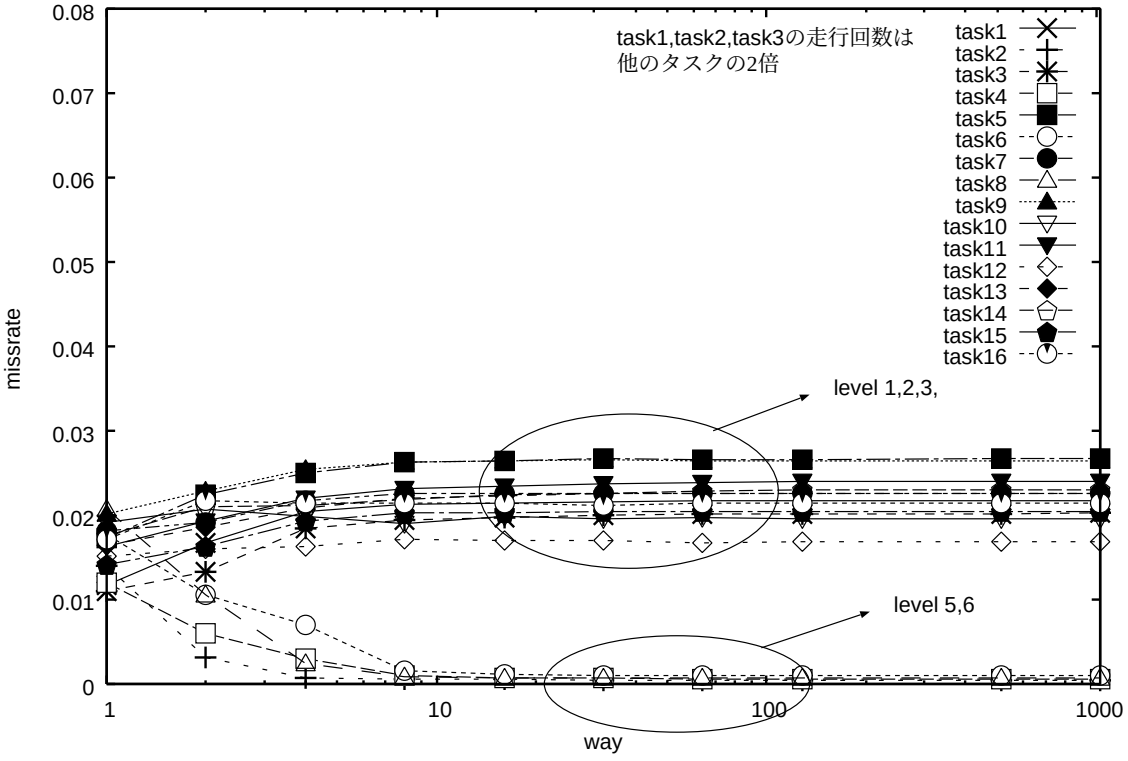

図 8.8: 動的な優先度と静的な優先度の組み合わせに基づいた置換制御を行った場合のミス率

第9章　考察

この実験はキャッシュリソースに対する優先度 High のタスクの fs100 を差し引いた残り少ないキャッシュリソースを、多くのタスクが競合することを想定したシミュレーションである。本研究で提案したキャッシュシステムにより、シミュレーションの想定が可能となった。

この実験の目的は、リソースに対する優先度のレベルを３種類に増やし、リソースに対する優先度 Mid、Low のタスクのミス率が優先度順で低くなるかを検証することである。実験結果では、いずれもリソースに対する優先度 High のタスクのミス回数が、ほぼ初期ミスに一致した。そしてそれ以外のタスクに関しては、ほぼ横這い状態となり分離する事はなかった。

リアルタイムシステムでは、タスクの正確な実行時間を見積ることが重要である。本論文で提案するキャッシュシステムにより、リソースに対する優先度 High のタスクの正確な実行時間を見積ることが可能になった。しかし、優先度 High 以外のタスクに関してはミス回数が一気に増加するため、正確な実行時間の見積りが不可能である。さらにこの実験から、リソースに対する優先度レベルを増やしても、High 以外のタスク間で優先度レベル順毎に低いミス率とはならず、ほぼ同じグループとしてまとるということがわかった。

以上のことから、リソースに対する優先度レベルは High、Low の高々2つで十分であるということが言える。

第10章　結論

コンテクスト切替えが行われても十分に効くキャッシュ機構には、キャッシュパラメータの適切な見積りが一番重要だったといえる。つまりコンテクスト切り替えに十分効くキャッシュ構成は一体どういうものであるのかということである。

それは図3.2のミスモデルによるとタスクセット全体を許容するキャッシュサイズを用意し、連想度を十分に大きくすればすべてのタスクがオンキャッシュし、コンテクスト切り替えに十分効くキャッシュシステムが構成できるということである。

また図3.2のミスモデルはキャッシュサイズを大きくしたところで、連想度が小さければ一向に初期ミスが無くならないという非常に重要な事実も隠されている。

しかし、すべてのタスクセットが入りきるキャッシュサイズを用意することは現実的ではない。そこで本論文では、タスクIDを用いて置換するタスクのラインを制御できるようにした。これによって、オンキャッシュしたいタスクを指定できるようにし、そのタスクのfs100が入るキャッシュサイズをまず用意した。通常のLRUでは、過去にアクセスされないラインは自動的にLRUスタックの後ろに移動されるため、必要なラインがもう無くなっているということが起こる。これをタスクIDを用いた優先度のレベルによる置換制御を行うことで、必要なラインが必要な時に存在するということが可能になった。

また実験(２)から少ないキャッシュリソースを巡って、リソースに対する優先度が低い多くのタスクが競合した場合、リソースに対して複数の優先度レベルを割与えることにあまり意味がないということがわかる。実験結果の図8.3から図8.4で示されているように、リソースに対する優先度Mid、Lowに割与えられているタスク群はお互いに分離せず、ほぼ同じグループに収まってしまうからである。　このことから、リソースに対する優先度は高々２種類で十分であり、重要な事実はリソースに対する優先度Highのタスクのミス回数がほぼ初期ミスに収まるため、正確な実行時間が見積もれるということである。このおかげで、リソースに対する優先度Highのタスクの正確な実行時間が見積もれるからである。

このようなキャッシュシステムは、コンテクスト切り替えが頻繁に起こるリアルタイムシステムにおいて非常に有効である。

通常のLRUでは、コンテキスト切り替えが起こる度に、新たなラインをロードしなければならないが、この提案したキャッシュシステムでは、優先度の高いタスクのラインを優先的に残すため、必要なタスクのラインが残っている確率は高くなる。そうなればコンテクスト切り替えによる再ロード時のミスを大幅に低減することができ、プロセッサのリアルタイム処理能力を一段と向上することが可能となる。

謝辞

本研究を進めるにあたり、終始熱心かつ寛容なご指導をいただきました、日比野 靖教授に心から感謝いたします。

また、適切な御助言をしていただきました田中 清文 助教授、菅原 英子 助手、井口 寧 助教授に深く感謝致します。

さらに、貴重なご意見、御討論をいただきました同じ計算機アーキテクチャ講座内の皆様に厚く御礼申し上げます。

参考文献

[1] Harold S.Stone 著
斎藤 忠男 , 発田 弘, 監訳 丸善株式会社 (1989)
高性能コンピュータアーキテクチャ (p63-p71).

[2] 高田広章 監修. 著
岸田 昌己/宿口 雅弘/南角茂樹 著 CQ 出版社 (2003) , リアルタイム OS と組み込み技術の基礎

[3] パターソン & ヘネシー 著 日経 BP 社 (1996)
成田 光彰 訳
コンピュータの構成と設計、ハードウェアとソフトウェアのインターフェイス (下)

[4] 1997 年 2 月 14 日、相原 孝一 、日比野靖、”マルチスレッド型プロセッサ向きのキャッシュメモリの構成と評価”
北陸先端科学技術大学院大学修士論文、1997 (p31-p35)