修 士 論 文

キャッシュブロックの配置法の実用性に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

広山　貴之

2013 年 3 月

修 士 論 文

キャッシュブロックの配置法の実用性に関する研究

指導教官　田中清史 准教授

審査委員主査　田中清史 准教授
審査委員　井口寧 教授
審査委員　金子峰雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1010056 広山　貴之

提出年月: 2013 年 2 月

概 要

階層型キャッシュの配置方式には、Inclusion Property 方式と Exclusion Property 方式の２種類が存在する。両者の方式を用いた場合、Inclusion Property 方式では、キャッシュ容量を圧迫し、Exclusion Property 方式では、コヒーレンス維持のオーバヘッドが大きくなる。この問題点に着目した先行研究として、”階層型キャッシュシステムにおける高効率なブロック配置法 ”がある。この研究では、キャッシュ内に配置されるブロックの局所性に着目したカテゴリ分けを提案した。本研究では、先行研究で提案したカテゴリ分けによる潜在性能を図る事を目的とする。

目 次

第1章　はじめに

1.1　研究背景

従来の階層型キャッシュの配置方法には、Inclusion Property 方式と Exclusion Property 方式の二種類の配置方法が存在する。Inclusion Property 方式は、下位階層に上位階層のコピーを保持する。メニーコア環境において、コア毎に L1 キャッシュを持ち、共有 L2 キャッシュを持つキャッシュ構成では、常に L2 キャッシュに各コアの L1 キャッシュのコピーを持たなければならないため、L2 キャッシュの容量を圧迫する。その一方で、キャッシュのコヒーレンス維持のためのキャッシュ参照は、L2 キャッシュに限定されるため、参照オーバヘッドが小さい。Exclusion Property 方式は、上位階層と下位階層で異なるブロックを保持する。このため、先述の例の場合、L2 キャッシュ容量のオーバヘッドが小さい。その一方で、コヒーレンス維持のための参照の範囲が各コアの L1 キャッシュに及ぶため、参照オーバヘッドが大きくなる。このトレードオフ関係に着目した先行研究として、階層型キャッシュシステムにおける高効率なブロック配置法がある。この研究では、キャッシュ内に配置するブロックのデータ参照の局所性に着目したカテゴリ分けを提案している。しかしながら、提案手法によるカテゴリ分けの潜在性能を把握することができていない。

1.2　研究目的

先行研究では、キャッシュ内に配置するブロックのデータ参照の局所性、共有・非共有に着目した配置ブロックのカテゴリ分けを行った。このカテゴリ分けを行う際、ブロックの長期的・短期的な参照、高い・低い参照局所性、プロセッサ間の共有ブロックに着目した。この着目点から、各ブロックをどの階層に格納するかを分類する。これにより、キャッシュの有効利用を図った。しかしながら、先行研究では、このカテゴリ分けによる潜在性能を把握できていない。そこで、本研究では、先行研究の原点に立ち返る。キャッシュ内に格納されるブロック間の競合関係を調べ、適切なキャッシュ階層に格納する方法を模索することにより、この手法による潜在性能を調査する。

1.3　論文の構成

本論文の構成を以下に示す。

第 2 章 既存手法の問題点に対する先行研究について述べる。

第 3 章 既存手法のキャッシュブロック配置の問題点と提案手法について述べる。

第 4 章 提案手法の性能評価について述べる。

第 5 章 まとめと今後の課題について述べる。

第2章　先行研究

2.1　先行手法

ブロックのデータの局所性に基づき、データの参照パターンの分析を行い、適切なキャッシュ階層への格納を行う。このデータの参照パターンの分析を行う際には、キャッシュブロックのアクセス頻度とアクセス間隔を基準として行う。1次キャッシュへの格納ブロックは、アクセス頻度が高くかつアクセス間隔が短い傾向を持つブロックとする。2次キャッシュへの格納ブロックは、長期間アクセスされる傾向を持つブロック。各キャッシュ階層内に不適合なブロックを格納しないことで、キャッシュメモリ容量の浪費を防ぐ。また、プロセッサ間で共有されるデータを必ず最下位キャッシュに格納することで、ブロックの一貫性維持を行う為の探索範囲を最下位キャッシュに限定させる。その結果、一貫性維持のためのオーバヘッドの増加を防ぐ。

2.1.1　カテゴリ分け

先行研究では、コヒーレンス維持のオーバヘッド時間の削減とキャッシュ資源の高効率利用を同時に実現するために、キャッシュへ配置するブロックを5つのカテゴリに分ける提案をした。キャッシュへの配置ブロックのデータの局所性に基づき以下のカテゴリに分類する。

1. 長期的に使用され、高い局所性を持つブロック
   L1 と L2 に存在すべきブロック

2. 長期的使用されるが、低い局所性を持つブロック
   L2 のみに存在すべきブロック

3. 短期的に高い局所性を持つブロック
   L1 のみに存在すべきブロック

4. 時間的局所性の無いブロック
   小容量バッファに存在すべきブロック

5. プロセッサ間共有ブロック
   L2 に必ず配置すべきブロック

2.1.2 小容量バッファ

小容量バッファは、１次キャッシュと同階層に位置する。空間的局所性のみを持つ参照列に対応する機構である。従って、バッファのサイズは、数ブロックほどの小さなもので十分である。バッファは１次キャッシュと並列参照し、１次キャッシュと同等かそれ以上早く参照が完了するサイズを想定する。

2.1.3 ロード／ストア命令

先行研究で提案するキャッシュシステムが扱うロード／ストア命令を以下に示す。

- ld／st with L1・L2
  ブロックを１次キャッシュと２次キャッシュ両方に配置する命令。
- ld／st with L2・B
  ブロックを２次キャッシュとバッファに配置する命令。
- ld／st with L1
  ブロックを１次キャッシュにのみ配置する命令。
- ld／st with B
  ブロックをバッファにのみ配置する命令。

2.1.4 ブロック配置の動作

１次キャッシュとバッファは第一階層のキャッシュとして扱われ、同時に参照される。第一階層キャッシュがミスした場合に２次キャッシュが参照される。各階層のヒット／ミスにおける各ロード／ストア命令の動作を以下に示す。

- 第一階層キャッシュヒット
  １次キャッシュ、または、バッファからロード／ストアを行う。
- 第一階層キャッシュミス・２次キャッシュヒット
  ２次キャッシュから各ロード／ストア命令が指定する第一階層キャッシュへブロックが渡される。データは各ロード／ストア命令が指定する第一階層キャッシュからブロックから参照される。
- 第一階層キャッシュミス・２次キャッシュミス
  主記憶から各ロード／ストア命令が指定するキャッシュ階層へブロックが渡される。データは、各ロード／ストア命令が指定する第一階層キャッシュから参照される。

２次キャッシュ内のブロックがリプレースされる時は第一階層キャッシュ内に存在するブロックも一緒に追い出す。これは第一階層キャッシュ内に共有データが残ることにより、一貫性維持のための探索範囲が第一階層キャッシュまで拡大される事を防ぐためである。

2.1.5 配置情報の取得

事前実行により取得するメモリアクセストレースを用いて、メモリアクセスの履歴の解析を行う。解析には、ブロックのリプレース時間とアクセス間隔を用いて行う。ブロックのリプレース時間は、ブロックがキャッシュに格納されてから、他のブロックとの競合により、リプレースされるまでの時間である。事前実行を行う際、シミュレータはブロックのリプレース時間を知るためにブロックの格納時刻とリプレース時刻をトレースし記録する。このトレースを用いて解析を行うことにより各ブロックのリプレース時間を算出する。解析により取得した全てのブロックのリプレース時間を平均した平均リプレース時間を判断基準に用いる。基本的には、平均リプレース時間より長いアクセス間隔を持つブロックを、ミスを起こしやすいブロックであると判断する。従って、平均リプレース時間より長いアクセス間隔を持つブロックは、キャッシュ内に格納しない方針を取る。

１次キャッシュへの格納可否の判断には、上記のアクセス間隔・平均リプレース時間情報とと共に、後述する連続アクセス情報を用いる。格納可否の判断は、アクセス間隔と連続アクセス回数に基づき、アクセス間隔と平均リプレース時間との比較から、点数を出し、予め決めた基準点数と比較を行う。点数の算出方法は、アクセス間隔が平均リプレース時間より長い場合、マイナス点数を加算する。アクセス間隔が平均リプレース時間より短い場合、連続アクセス回数が多い場合、高い点数を、連続アクセス回数が少ない場合、低い点数を加算する。これらの合計値が基準点数を満たさない場合、該当ブロックを１次キャッシュ内に格納しない。これにより、平均リプレース時間を越えるアクセス間隔を持たないブロックの中で、アクセス回数の少ないブロックをキャッシュ内に格納させない。その結果、既にキャッシュ内に存在するより有用なブロックをリプレースすることを防ぐ事ができる。アクセス回数の少ないブロックは、バッファで対応できる。この為、１次キャッシュに格納されないブロックは、バッファに格納する。

1次キャッシュへの格納可否の情報を以下に示す。

- L1 Short term access(st)
  平均リプレース時間を超えないアクセス間隔を示す。1回のL1 Short term accessが発生するとL1 Short term accessを表す数字0を記録する。1回のL1 Short term accessを表す0は点数を持たない。

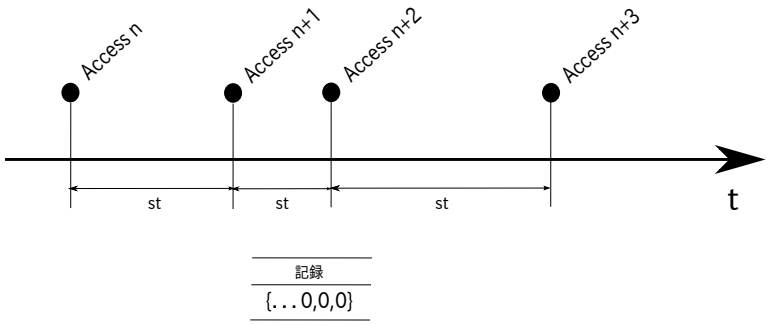

図 2.1: L1 Short term accessの記録の例

- L1 Long term access(lt)
  平均リプレース時間を超えるアクセス間隔を示す。1回のL1 Long term accessが発生するとL1 Long term accessを表すアルファベットHを記録する。1回の平均リプレース時間を超えたアクセスを表すHは、マイナス点数を持つ。

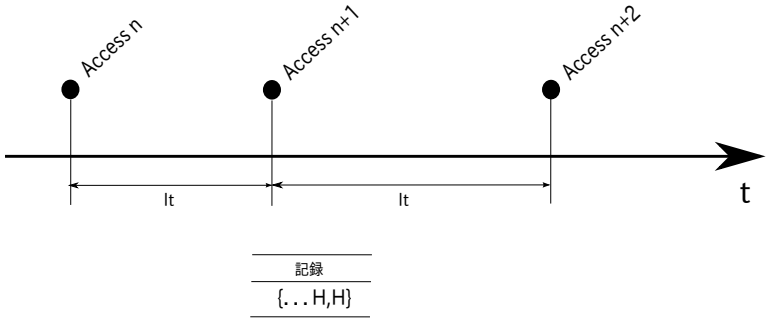

図 2.2: L1 Long term accessの記録の例

- 連続アクセス
  l1 short term access がｎ回（ｎは設定値）以上連続して発生し、連続発生回数により９個のグループに分類する。連続アクセスの各グループを以下の表に記述する。

連続アクセス回数	n	2n	3n	4n	5n	6n	7n	8n	9n 以上
分類グループ	1	2	3	4	5	6	7	8	9

表 2.1: 連続アクセスグループ

このグループにおいて、１が最も少ない連続アクセス回数、９が最も多い連続アクセス回数を表す。これにより、アクセス点数を付ける。１が最も低いアクセス点数、９が最も高いアクセス点数を持つ。

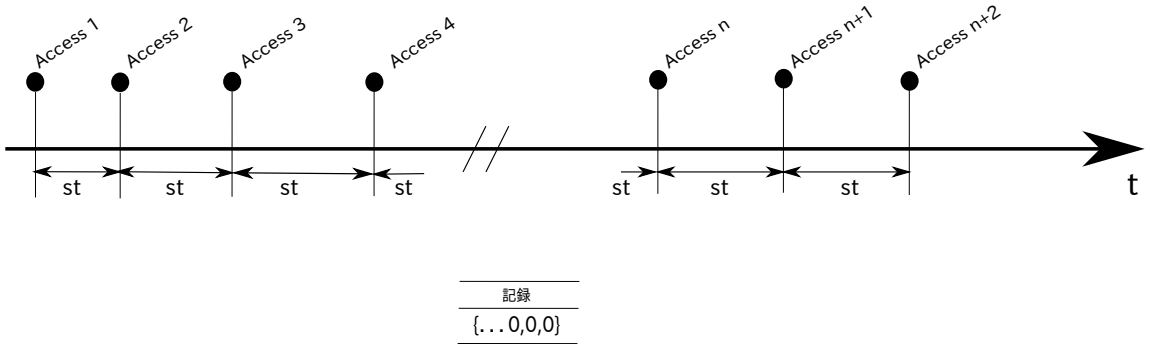

図 2.3: 連続アクセスの例

２次キャッシュへの格納可否には、アクセス間隔・平均リプレース時間を用いる。そして、格納可否の判断は、該当ブロックで発生したアクセス回数と平均リプレース時間を越えるアクセス間隔の回数の百分率で行う。百分率が高いブロックを、ミスを起こしやすいブロックであると判断する。これにより、２次キャッシュに格納しない方針を採る。

２次キャッシュへの格納可否の情報を以下に示す。

- L2 Short term access(st)
  平均リプレース時間を超えないアクセス間隔を示す。１回の L2 Short term access が発生すると L2 Short term access を表す数字０を記録する。

- L2 Long term access(st)
  平均リプレース時間を越えるアクセス間隔を示す。1回のL2 Long term accessが発生するとL2 Long term accessを表すアルファベットHを記録する。

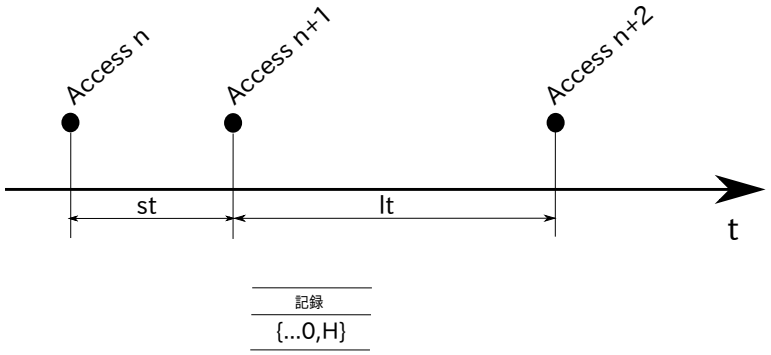

図 2.4: L1 Long term accessの記録の例

第3章　キャッシュブロック配置

3.1　既存手法の問題点

既存の配置手法には、Inclusion Property 方式と Exclusion Property 方式の 2 種類の配置方法が存在する。それらの配置方法には、次に示す問題点が存在する。

3.1.1　Inclusion Property 方式の問題点

Inclusion Property 方式は、下位階層に上位階層のブロックのコピーを保持する。各コア毎に独立した上位階層を持つキャッシュは、シングルコアの場合、コピーを保持することは大きな問題にはならない。しかし、コア数の増加に伴い、上位階層の総容量が下位階層に対して、無視できない大きさとなる。この為、下位階層に存在する上位階層のブロックのコピーがキャッシュメモリ資源を浪費することになる。その結果、下位階層における有効なキャッシュメモリ容量が減ることになる。

3.1.2　Exclusion Property 方式の問題点

Exclusion Property 方式は、上位階層と下位階層で異なるブロックを保持する。これにより、マルチコア・マルチプロセッサ環境において、コヒーレンス維持のための参照範囲が、下位階層だけではなく、上位階層に及ぶことになる。その為、コヒーレンス維持の為の外部参照によるオーバヘッドが大きくなる。

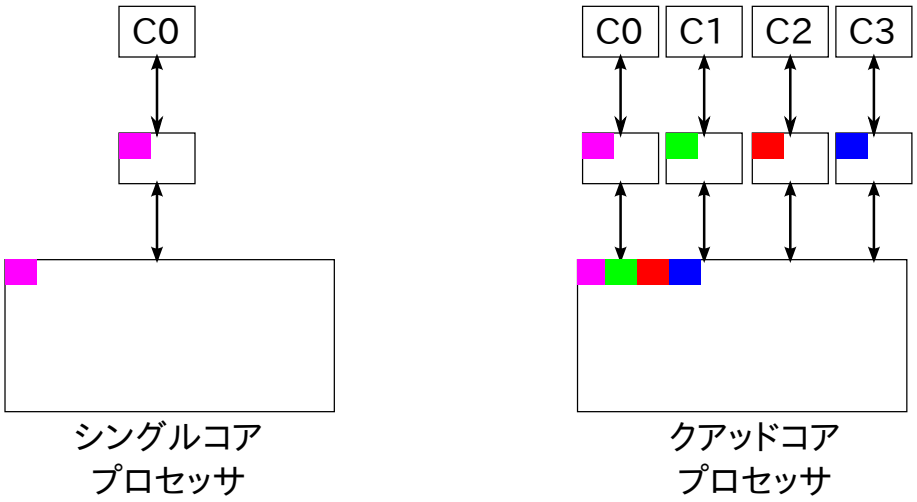

図 3.1: Inclusion Property 方式

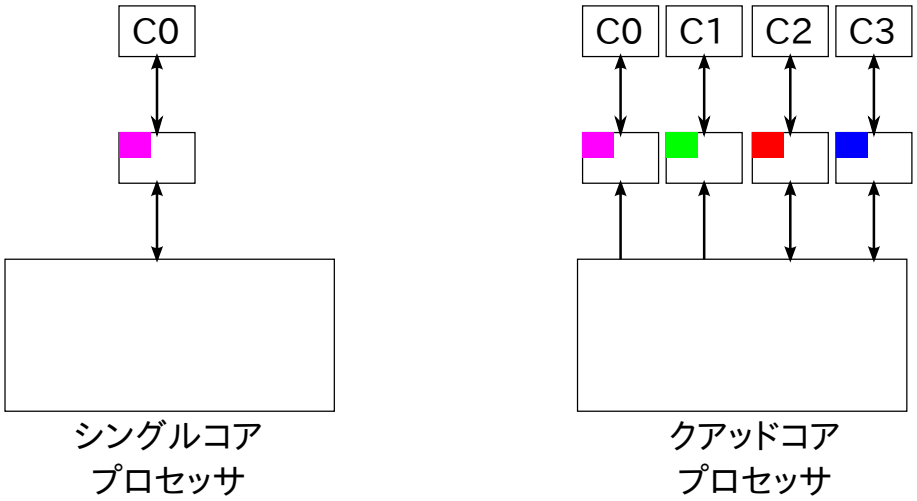

図 3.2: Exclusion Property 方式

3.2 先行研究の問題点

１次キャッシュへの格納可否を行う際、全ブロックのリプレース時間を平均化した平均リプレースと各ブロックのアクセス間隔の比較を行う。２次キャッシュへの格納を行う際には、平均リプレース時間を超えるアクセス回数と該当ブロックで発生したアクセス回数との百分率の比較を行う事で実現している。これらの格納可否判断に共通することは、メモリアクセスの間隔に着目していること。この方法では、キャッシュへ格納される各ブロックのヒット／ミスといったキャッシュの挙動について、全く考慮していない格納方法である。その結果、キャッシュ内に格納するブロックの格納精度が低下することになり、キャッシュ性能を低下することに繋がる。

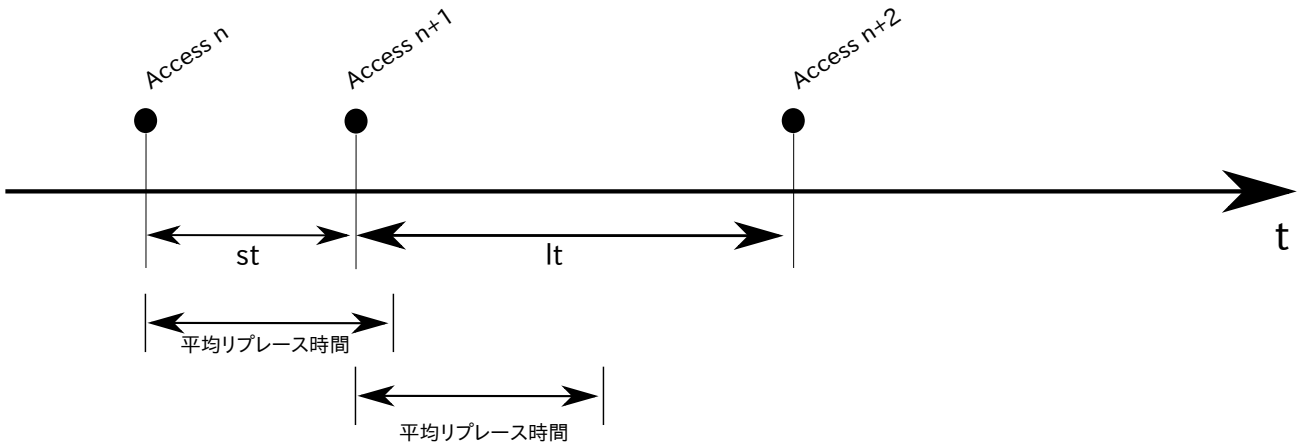

図 3.3: メモリアクセス間隔との比較による問題

3.3 解析手法

本研究では、共有・非共有性を持つキャッシュブロックとキャッシュへの格納後の再利用性に着目した格納方式を提案することで、格納精度を向上を図る。本解析手法では、事前実行によって取得したメモリアクセストレースを解析する。これにより、各ブロックを静的にカテゴリ分けを行う。

L1 キャッシュへの格納可否判断には、ブロックをキャッシュへ格納後、ブロックがリプレースされるまでにおいて、再利用性があるかを調べる。再利用性のあるブロックのみをキャッシュに格納することで、性能向上を図る。

L2 キャッシュへの格納可否判断には、L1 キャッシュへの格納可否判断と同様の方式を採用し、L2 キャッシュに格納されるブロックの再利用性を調べ、再利用性のあるブロックのみを格納することで、性能向上を図る。

プロセッサ間で共有されるブロックは、先行研究同様に、最下位キャッシュに格納することによって、ブロックの一貫性維持による参照オーバヘッドを減少させる。

3.3.1 世代と再利用された世代

本研究では、キャッシュに格納されたブロックの再利用性を調べる為、次の 2 つの用語を定義する。

- 世代
  キャッシュミスが発生し、参照ブロックがキャッシュ内に格納されてから、リプレースされるまでの区間。

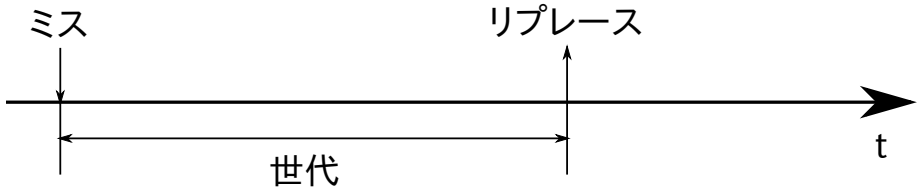

図 3.4: 世代

- 再利用された世代
  世代の区間内で、1 回以上、キャッシュヒットが発生する世代。

キャッシュへの格納可否の判断材料として、これらを用いる。その結果、各ブロックの再利用性を把握し、適切なキャッシュ階層へ格納する。

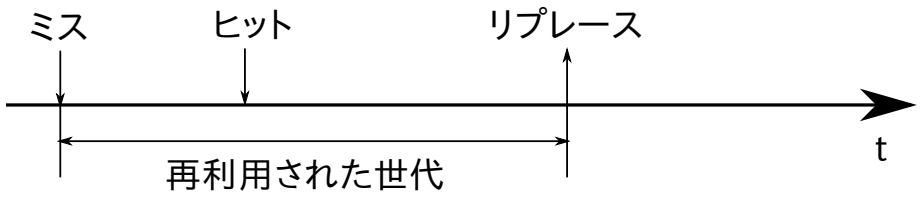

図 3.5: 再利用された世代

3.3.2 カテゴリ分け手法の調査

参照される各ブロックのカテゴリ分けを行う際、次に述べる 3 ステップの順に調査を行う。これにより、各ステップにおける格納されるブロックの再利用性を調査する。その調査結果から、各ブロックを配置する最適なキャッシュ階層を決定する。

- ステップ１
  キャッシュへの格納可否判断には、全世代の中で、一つ以上、再利用された世代を持つブロックかを調べることで行う。これにより、キャッシュへの格納ブロックがキャッシュを有効利用したかが把握できる。この有効利用されたブロックのみをキャッシュへ格納することにより、性能向上が図れるかを調査する。

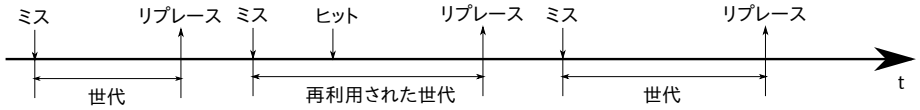

図 3.6: ステップ１のブロック

- ステップ２
  ステップ１における再利用された世代への判断基準に、閾値を加えた調査を行う。この調査は、各世代の生成時刻より、閾値時刻以内のアクセスのみを持つブロックを再利用された世代と判断しない方式を採る。これにより、世代が生成されてから、閾値以内のみのアクセスをキャッシュに格納しないことで、キャッシュへより有用性の高いブロックのみを格納することができ、ミス数を軽減できる。

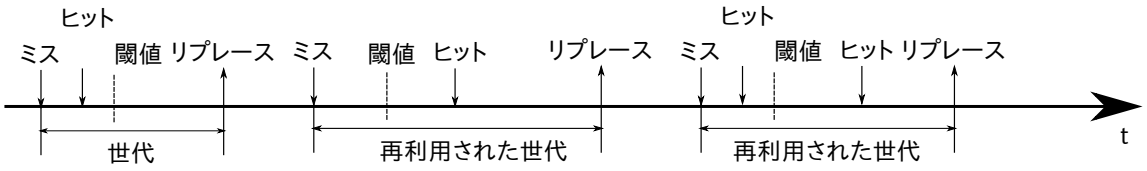

図 3.7: ステップ２のブロック

- ステップ３
  ステップ２で取得した各ブロックの世代数と再利用された世代数との割合を調べる。これと新たに設ける閾値との比較を行う。この手法により、生成される全世代中、再利用される世代の割合が少ないブロックをキャッシュに格納させない。この結果から、より有用性の高いブロックのみをキャッシュに格納することができ、ミス数を削減できる。

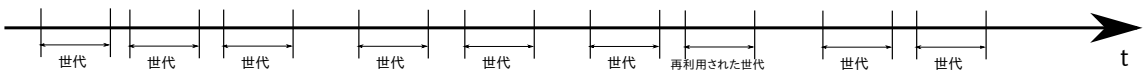

図 3.8: ステップ３のブロック

上記の図のように、再利用された世代数が少ないブロックをキャッシュ内に格納しないことで、性能向上を図る。

３ステップを各キャッシュ階層に格納される全ブロックに対して適用する。適用した結果、各ブロック毎に各キャッシュ階層に格納するかを決定する。

3.3.3　ブロックの配置動作

１次キャッシュとバッファは、第一階層のキャッシュと考え、同時に参照されるものと想定する。第一階層キャッシュがミスし、２次キャッシュがヒットした場合、カテゴリに沿って、第一階層に該当ブロックを格納する。２次キャッシュがミスした場合、カテゴリに沿って、各階層に該当ブロックを格納する。

第4章　評価

4.1　評価環境

4.1.1　PIN

本研究では、提案手法を検証する為に、ベンチマークプログラムを実行させ、メモリアドレストレースを取得する。このメモリアドレストレースを取得するため、PIN を用いたプログラムを作成する。PIN とは、intel 社が提供する API 集である。

4.1.2　キャッシュシミュレータ

作成したキャッシュシミュレータは、２コア・２プロセッサの２階層の Inclusion Property 方式と Exclusion Property 方式と提案シミュレータです。各コアには、独立した１次キャッシュを保持する。２次キャッシュには、各 L1 キャッシュで、共有するキャッシュとなる。一貫性を維持する為に、スヌープ方式を採用する。

各階層のブロックがリプレイスされる際、リプレイス対象の選択方法として、LRU 方式を採用する。

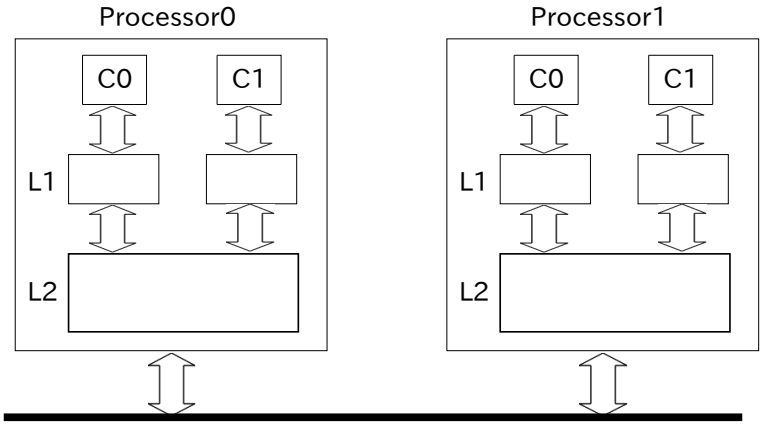

図 4.1: ２コア・２プロセッサ構成

4.1.3 ツール

提案手法を実現・評価するためには、ベンチマークマークプログラムのメモリアクセストレースを取得するプログラムと各カテゴリ分けを行うプログラムが必要となる。これらのツールは、以下の通りとなる。

- SASCache
  ベンチマークプログラムを入力ファイルとした命令実行シミュレーションプログラム。出力ファイルとして、メモリアクセストレースを生成する。

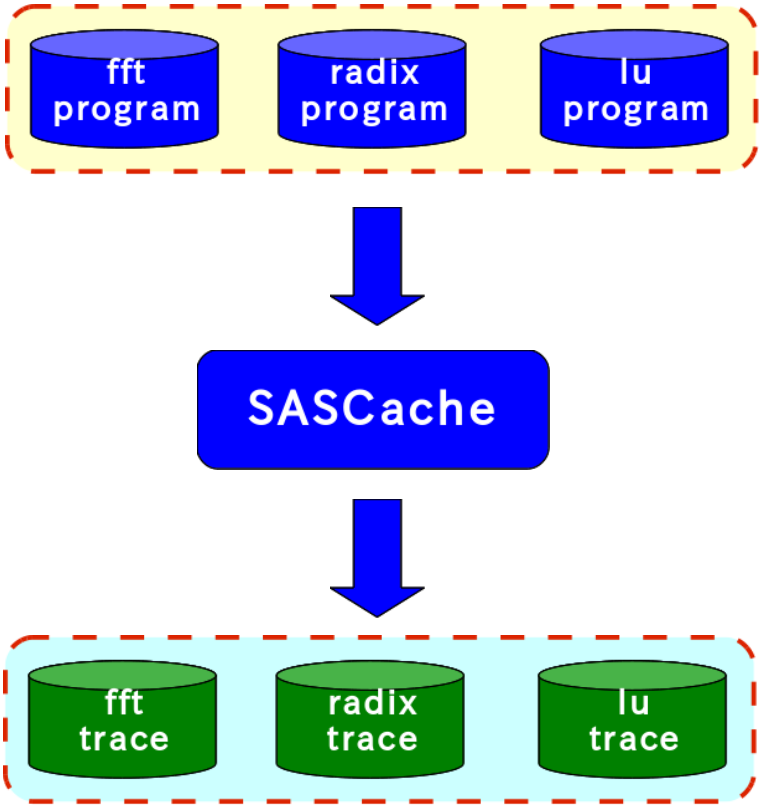

図 4.2: メモリアクセストレース取得

取得するメモリアクセストレースの仕様は、以下の通りとなる。

時刻	スレッド	ロード／ストア	アドレス
10000	0	load	0xFF24
10001	1	store	0xED34
10002	3	store	0x67AB
10003	2	load	0x78CA
10004	2	load	0x4367

図 4.3: メモリアクセストレース

- analy_cat1_to_4
  メモリアクセストレースを入力ファイルとして、キャッシュに格納される各ブロックをカテゴリ分けされたファイルを生成する。生成するために、内部にはキャッシュシミュレータが実装されている。このキャッシュシミュレータから、ミス・ヒット・リプレースの履歴を取得し、この履歴を解析する。解析結果として、カテゴリ情報ファイルを出力する。

アドレス	カテゴリ	プロセッサ番号
0x12AB	1	0
0x23CD	3	1
0xAC42	2	1
0x23DE	4	1
0x3412	3	0

図 4.4: カテゴリ情報ファイル

- analy_cat5
  メモリアクセストレースを入力ファイルとし、プロセッサ間の共有ブロックを抽出し、カテゴリ情報を作成するプログラム。

- integrate_cat
  カテゴリ情報の取得プログラム：analy_cat1_to_4,analy_cat5 より取得したカテゴリ

情報を統合させるプログラム。このプログラムより、プロセッサ間で共有されるブロックと判断された場合、そのブロックを最下位キャッシュ階層に格納させるカテゴリに変更する。

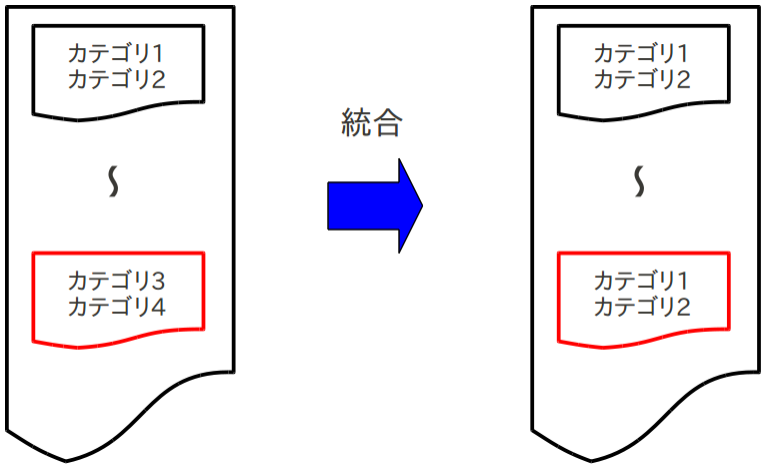

図 4.5: カテゴリ情報の統合

上記の図では、赤枠のカテゴリ３，４がプロセッサ間で共有されるブロックであると判断さたので、最下位キャッシュに格納されるカテゴリに変更した。

カテゴリ情報作成プログラム：analy_cat1_to_4,analy_cat5,integrate_cat を用いて、最終的に必要となるカテゴリ情報を作成する流れを次に示す。

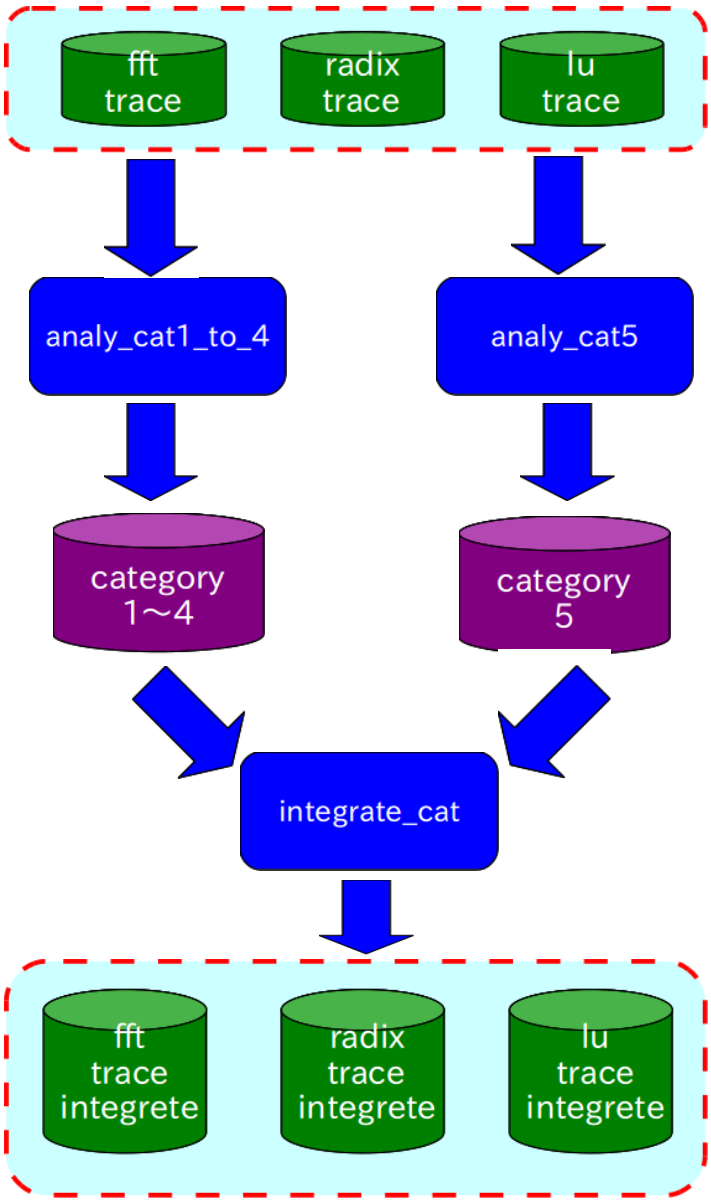

図 4.6: カテゴリファイル出力

4.1.4 パラメータ

各カテゴリに分けるツール・キャッシュシミュレータに入力するパラメータを以下に記述する。

パラメータ	L1 ウェイ数	4
	L2 ウェイ数	4
	L1/L2 ブロックサイズ	32 Byte
	プロセッサ数	2
	コア数	2
	バッファサイズ	128 Byte
	L1 キャッシュサイズ	8 KByte
	L2 キャッシュサイズ	128 KByte

表 4.1: パラメータ

4.1.5 ベンチマークプログラム

キャッシュシミュレータに入力させるメモリアクセストレースを取得するためのベンチマークプログラムとして、radix,fft,lu を使用した。以下に、パラメータ値を記述する。

ベンチマークプログラム	パラメータ	パラメータ値
Radix	number of keys to sort	1048576
	number of processors/cores	4
fft	even integer 2**M total complex data points transformed	18
	Log base 2 of cache line length in bytes	5
	number of processors/cores	4
LU contiguous block	Decompose N*N matrix	224
	number of processors/cores	4
LU non contiguous block	Decompose N*N matrix	384
	number of processors/cores	4

表 4.2: ベンチマークプログラムパラメータ

4.2 評価方法

4.2.1 評価対象

評価対象として用いる配置方法は、Inclusion Property 方式と Exclusion Property 方式と先行研究方式とする。

4.2.2 評価仕様

Inclusion Property 方式・Exclusion Property 方式と提案キャッシュのシミュレータにベンチマークの各メモリアドレストレースを入力させ、実行させる。入力メモリアドレストレースは、マルチスレッド仕様となる。

4.2.3 各階層への格納可否

カテゴリ分け手法の３ステップを用いて、各階層への格納可否を判断する。

- L1 キャッシュへの格納可否

1. ステップ１
   L1 キャッシュへ格納されるブロックの世代が生成され、各世代において、ヒットが発生するかをチェックする。一回以上のヒットがあるブロックを L1 キャッシュに格納する。今回、本ステップを採用した結果、各ベンチマークにおいて、Inclusion Property 方式よりも性能向上が確認されたので、本ステップを有効な方式と判断する。
2. ステップ２
   再利用された世代と判断する為に用いられる閾値を変更しながら、ステップ１よりも性能向上が確認される間、調査を続ける。その結果、各ベンチマークプログラムにおいて、性能向上が確認されたので、本ステップを有効な方式と判断する。
3. ステップ３
   再利用された世代と世代数との割合を算出する。この算出値と閾値と比較し、ステップ２よりも性能向上が確認できるまで、比較を続ける。今回、比較を継続した結果、ステップ２と同等の結果が得られた為、効果が無いと判断する。

- L2 キャッシュへの格納可否

1. ステップ１
   L1 ミスが発生し、L2 アクセスが生じた際、L2 における世代を生成する。生成される世代の中で、一回以上のヒットがあるブロックを L2 キャッシュに格納する。今回、本ステップを採用した結果、各ベンチマークプログラムにおいて、Inclusion Property 方式ようりも性能向上が確認できたので、本ステップを有効な方式と判断する。
2. ステップ２
   L1 キャッシュにおけるステップ２では、同階層にバッファが存在した為、本ステップを採用したのだが、L2 キャッシュの階層では、バッファに相当するものが無いので、評価は省略した。
3. ステップ３
   再利用された世代と世代数との割合を算出する。この算出値と閾値と比較し、ステップ１よりも性能向上が確認できるまで、比較を続ける。L2 キャッシュ階

層では、ステップ１よりも性能向上が確認できたので、本ステップを有効な方式と判断する。

ステップ２においては、閾値を設け、それ以降のアクセスがなければ、再利用性のない世代と判断する。この閾値を求める際には、任意に決めた数値を入力し、ミス数の移り変わりを検証していく。その結果、ミス数の削減率が最も高い箇所をピークとし、それを実現する閾値を評価に採用することとした。

ステップ３においても、ステップ２と同様に、任意に決めた閾値を設定し、ミス数の移り変わりを検証していく。その結果、ミス数の削減率が最も高い箇所をピークとし、それを実現する閾値を評価に採用することとした。

4.2.4 性能評価

提案キャッシュシミュレータに、ベンチマークプログラムより取得したメモリアドレストレースと統合カテゴリファイルを入力し、IP/EP キャッシュシミュレータには、メモリアドレストレースを入力し、実行する。実行結果より、評価を行う。

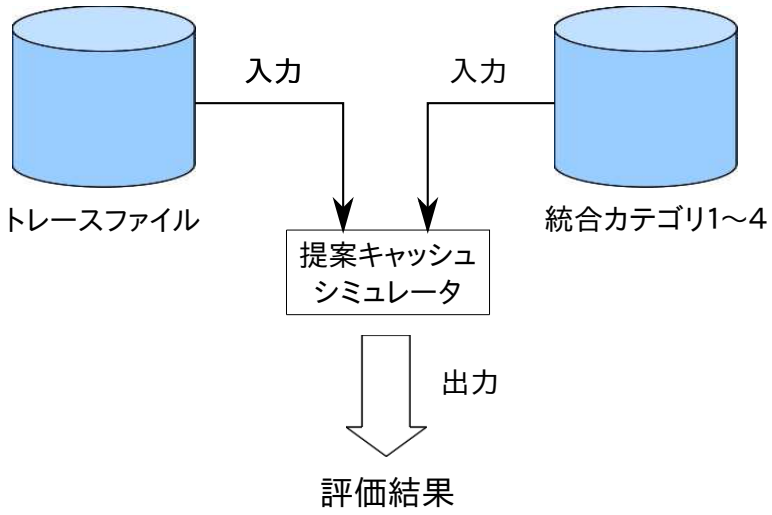

図 4.7: 提案キャッシュ性能評価

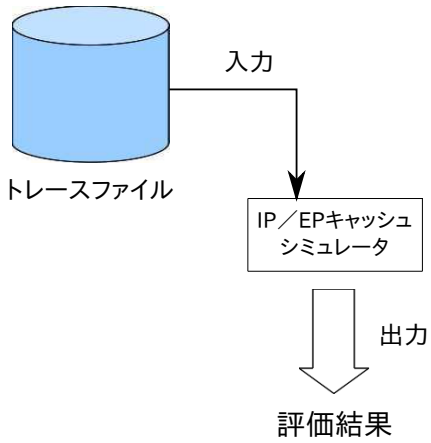

図 4.8: IP ／ EP 性能評価

シミュレーション結果より取得した値を評価する。評価対象は、各階層で、最もミス数が大きい箇所を焦点に当て、どれだけミス数が削減できたかを検証する。ステップ３の評価で用いた閾値は、２～２０%の範囲で評価を行い、ミス数を最も低下させた閾値を採用する。

- Radix
  第一階層において、全コアの中で、プロセッサ１のコア１が最もミス数が高い。このコアのＬ１キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、5. 02 倍となる。Inclusion Property 方式と L1 におけるステップ１を適用した結果と比較すると、0.967 倍となる。Inclusion Property 方式と L1 におけるステップ２までを適用した結果を比較すると、0.961 倍となる。この時評価に用いたステップ２の閾値は、16384 となる。Inclusion Property 方式と L1 におけるステップ３までを適用した結果を比較すると、0.961 倍となる。この時評価に用いたステップ３の閾値は、２～２０%であるが、どの閾値でも変化が見られなかった。

Inclusion Property 方式	先行研究方式	L1-ステップ１	L1-ステップ１，２	L1-ステップ 1,2,3
633331(0.0339)	3183248(0.1705)	612818(0.0328)	608974(0.0326)	608974(0.0326)

表 4.3: 性能評価：radix(L1)

第二階層において、各プロセッサ間で、プロセッサ０の L2 キャッシュのミス数が最も高い。この L2 キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、8.69 倍となる。Inclusion Property 方式と L2 におけるステップ１を適用した結果と比較すると、0.973 倍となる。Inclusion Property 方式とステップ２までの評価は省略した。Inclusion Property 方式と L2 におけるステップ３までを適用した結果と比較すると、0.967 倍となる。この時評価に用いたステップ３の閾値は、１５%となる。

Inclusion Property 方式	先行研究方式	L2-ステップ１	L2-ステップ１，２	L2-ステップ 1,3
587700(0.4768)	5110499(0.8133)	571898(0.479)	-	568838(0.477)

表 4.4: 性能評価：radix(L2)

- fft
  第一階層において、全コアの中で、プロセッサ０のコア０が最もミス数が高い。このコアのＬ１キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、12.99 倍となる。Inclusion Property 方式と L1 におけるステップ１を適用した結果と比較すると、0.999 倍となる。Inclusion Property 方式と L1 におけるステップ２までを適用した結果を比較すると、0.988 倍となる。この時評価に用いたステップ２の閾値は、1024 となる。Inclusion Property 方式と L1 におけるステップ３までを適用した結果を比較すると、0.988 倍となる。この時評価に用いたステップ３の閾値は、２～２０%であるが、どの閾値でも変化が見られなかった。

Inclusion Property 方式	先行研究方式	L1-ステップ１	L1-ステップ１，２	L1-ステップ 1,2,3
975844(0.0244)	12679115(0.377)	975771(0.0244)	965004(0.0241)	965004(0.0241)

表 4.5: 性能評価：fft(L1)

第二階層において、各プロセッサ間で、プロセッサ０の L2 キャッシュのミス数が最も高い。この L2 キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、11.33 倍となる。Inclusion Property 方式と L2 におけるステップ１を適用した結果と比較すると、0.990 倍となる。Inclusion Property 方式とステップ２までの評価は省略した。Inclusion Property 方式と L2 におけるステップ３までを適用した結果と比較すると、0.989 倍となる。この時評価に用いたステップ３の閾値は、６%となる。

Inclusion Property 方式	先行研究方式	L2-ステップ１	L2-ステップ１，２	L2-ステップ 1,3
1086852(0.659)	12324359(0.875)	10761841(0.658)	-	1075862(0.658)

表 4.6: 性能評価：fft(L2)

- contiguous_block
  第一階層において、全コアの中で、プロセッサ０のコア０が最もミス数が高い。このコアのＬ１キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、1 5.44 倍となる。Inclusion Property 方式と L1 におけるステップ１を適用した結果と比較すると、0.997 倍となる。Inclusion Property 方式と L1 におけるステップ２までを適用した結果を比較すると、0.995 倍となる。この時評価に用いたステップ２の閾値は、32 となる。Inclusion Property 方式と L1 におけるステップ３までを適用した結果を比較すると、0.995 倍となる。この時用いたステップ３の閾値は、２～２０%であるが、どの閾値でも変化が見られなかった。

Inclusion Property 方式	先行研究方式	L1-ステップ１	L1-ステップ１，２	L1-ステップ 1,2,3
78084(0.003524)	1206243(0.0544)	77994(0.00352)	77762(0.0035)	77762(0.0035)

表 4.7: 性能評価：contiguous_block(L1)

第二階層において、各プロセッサ間で、プロセッサ０の L2 キャッシュのミス数が最も高い。この L2 キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、14.58 倍となる。Inclusion Property 方式と L2 におけるステップ１を適用した結果と比較すると、0.994 倍となる。Inclusion Property 方式とステップ２までの評価は省略した。Inclusion Property 方式と L2 におけるステップ３までを適用した結果と比較すると、0.992 倍となる。この時用いたステップ３の閾値は、１９%となる。

Inclusion Property 方式	先行研究方式	L2-ステップ１	L2-ステップ１，２	L2-ステップ 1,3
66742(0.582)	973296(0.457)	66408(0.5820)	-	66228(0.5809)

表 4.8: 性能評価：contiguous_block(L2)

- non_contiguous_block
  第一階層において、全コアの中で、プロセッサ０のコア０が最もミス数が高い。このコアのＬ１キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、3 .36 倍となる。Inclusion Property 方式と L1 におけるステップ１を適用した結果と比較すると、0.999 倍となる。Inclusion Property 方式と L1 におけるステップ２までを適用した結果を比較すると、0.999 倍となる。この時評価に用いたステップ２の閾値は、64 となる。Inclusion Property 方式と L1 におけるステップ３までを適用した結果を比較すると、0.999 倍となる。この時評価に用いたステップ３の閾値は、２～２０%であるが、どの閾値でも変化が見られなかった。

Inclusion Property 方式	先行研究方式	L1-ステップ１	L1-ステップ１，２	L1-ステップ 1,2,3
2063333(0.0927)	6933530(0.3118)	2063259(0.0927)	2063019(0.0927)	2063019(0.0927)

表 4.9: 性能評価：non_contiguous_block(L1)

  第二階層において、各プロセッサ間で、プロセッサ０の L2 キャッシュのミス数が最も高い。この L2 キャッシュに焦点を当てる。Inclusion Property 方式と先行研究提案方式と比較すると、5.30 倍となる。Inclusion Property 方式と L2 におけるステップ１を適用した結果と比較すると、0.998 倍となる。Inclusion Property 方式とステップ２までの評価は省略した。Inclusion Property 方式と L2 におけるステップ３までを適用した結果と比較すると、0.998 倍となる。この時評価に用いたステップ３の閾値は、１５%となる。

Inclusion Property 方式	先行研究方式	L2-ステップ１	L2-ステップ１，２	L2-ステップ 1,3
782204(0.2051)	4149231(0.3723)	781024(0.2409)	-	780664(0.2048)

表 4.10: 性能評価：non_contiguous_block(L2)

第5章　おわりに

5.1　まとめ

既存のキャッシュ配置法には、Inclusion Property 方式と Exclusion Property 方式がある。Inclusion Property 方式は、キャッシュ容量を圧迫し、一方として、Exclusion Property 方式は、コヒーレンス維持のオーバヘッドが大きくなる。これらに着目した先行研究として、階層型キャッシュシステムにおける高効率なブロック配置法がある。本研究では、この先行研究の提案手法の潜在性能を図ることを目的とした。キャッシュ内の再利用性のあるブロックをキャッシュに格納することで、潜在性能を図った。

参考文献

[1] HUH Younsuk, 階層型キャッシュシステムにおける高効率なブロック配置法, 修士論文, 2011.

[2] S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta, SPLASH-2 Programs: Characterization and Methodological Considerations Proc. of ISCA pp.24–36, 1995.

[3] Intel, PIN, http://www.pintool.org