

修士論文

題目

セル・アロケーションキャッシュの
領域割り当てとその評価手法に
関する研究

指導教員

大野 和彦 講師

2019 年

三重大学大学院 工学研究科 情報工学専攻
コンピュータ・ソフトウェア研究室

鬼頭 優人 (417M507)

内容梗概

近年、スマートフォンやパソコンなどのプロセッサでは、高性能化と低電力化の両立が求められており、様々な高速化・低消費電力化手法が提案されている。そのうち、高性能化の手法の一つとして複数のコアを用意し、並列に処理を行うマルチコア化が挙げられる。しかし、マルチコア環境では複数のメモリアクセスが同時に発生することで、シングルコア環境よりデータの競合が発生しやすくなり、結果としてキャッシュミス率が高くなるという問題がある。メモリアクセスは性能のボトルネックの一つであるため、高性能化を達成するためには主記憶へのメモリアクセスの低減、すなわちキャッシュミス率の低減が重要である。そこで、この問題を解決する手法としてキャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュが提案されているが、トレースドリブン・シミュレータでしか評価していなかった。一般に並列処理ではほかのスレッドやプロセスとの依存関係によりプログラムの振る舞いや性能が変わるため、より現実的なキャッシュの評価を行うためにはアーキテクチャレベルのシミュレーションを行うことが望ましい。そこで、本研究では、まずセル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 上に実装する。トレースドリブン・シミュレータでは効果が得られなかったマルチスレッドベンチマーク単体を用いて再評価した結果、通常キャッシュや従来の固定的なキャッシュ・パーティショニングと比較して、ヒット率がそれぞれ最大 44.3 %, 5.2 % 向上することが確認できた。また、より現実的に近い評価を行うために、負荷の違うベンチマークを複数組み合わせた複合ベンチマークを作成し、それによってロードやメモリアクセスのようなプログラムの振る舞いが頻繁に変化する場合を評価した。その結果、ヒット率がそれぞれ最大 11.8 %, 11.7 % 向上した。また、セル・アロケーションキャッシュの領域を分割する単位のセルには各コアのみが主記憶からキャッシュへ書き込める固有セルと、通常キャッシュと同様にアクセスできる共有セルの 2 種類が存在する。この 2 種類のセルの切り替えにより各コアに最適なキャッシュ領域を与え、効率的にデータを記憶する。セルの切り替えには閾値を使用し、データの共有率が閾値を超えれば固有セルから共有セルに切り替わる。評価結果から、その閾値によってセル・アロケーションキャッシュのヒット率が大きく変わることが新たに分かった。そこで、

本研究では Phase Detection を用いてプログラムの領域を分割し，その領域がどのようにプログラムの振る舞いとして動いているのかを把握した上で，従来，固定的に手動で定めていた閾値を各領域に応じて自動で設定し，新たな領域割り当てを行うことを提案した．これによりさらなるセル・アロケーションキャッシュの性能向上を目指した．

Abstract

Multi-core processors are widely used to improve performance of computer systems. To achieve both high performance and low power consumption, Cell-allocation cache is proposed. It allocates cache spaces called ‘ Cell ’ which is smaller than a way and dynamically assigns it to a core. However, it is only evaluated with trace-driven simulation, so evaluations of execution speed and energy consumption in realistic environments are not presented. To solve the problems, in this paper, at first, we implement Cell-allocation cache on the cycle accurate simulator Gem5 and evaluates more detailed performance. the hit rates improved by max 44.3% and 5.2%, respectively at the Himeno benchmark. We also propose evaluation methodology by mixing plural benchmark programs to evaluate performance under unbalanced load works. At the result the hit rates improved by max 11.8% and 11.7%, respectively. From the result, we found Hit rates change depending on thresholds to decide Private cell or Shared cell. So we propose deciding thresholds automatically. We aim to improve the performance of Cell-allocation cache.

目次

1	はじめに	1
2	キャッシュシステム	5
2.1	ダイレクトマッピングキャッシュ	5
2.2	セットアソシアティブキャッシュ	7
3	先行研究	9
3.1	キャッシュ・パーティショニング	9
3.2	セル・アロケーションキャッシュ	10
3.2.1	セル・アロケーションキャッシュの概要	10
3.2.2	セル・アロケーションキャッシュの動作	12
4	従来研究の問題点	14
4.1	トレースドリブンシミュレーションによる評価	14
5	提案手法	16
5.1	提案評価手法	16
5.2	複合ベンチマークを用いた評価の提案	16
6	性能評価	19
6.1	評価方法	19
6.2	評価結果	20
6.3	考察	23
7	適応的な閾値の自動決定手法の提案	24
8	おわりに	28
	謝辞	30
	参考文献	30

図 目 次

2.1	ダイレクトマッピングキャッシュの概念図	5
2.2	セットアソシアティブキャッシュの概念図	7
3.3	セル・アロケーションキャッシュの概念図	11
3.4	セルの詳細	12
4.5	ロックプログラムの例	15
5.6	画一化されたメモリアクセス	17
5.7	画一化されていないメモリアクセス	18
6.8	キャッシュ・ヒット率 (姫野ベンチマーク)	21
6.9	キャッシュ・ヒット率 (複合ベンチマーク)	22
7.10	閾値決定の概要	24
7.11	Phase Detection を用いた閾値自動決定手法の概要	25
7.12	提案手法のハードウェアブロック図	26

表 目 次

6.1 評価環境のパラメータ一覧	19
----------------------------	----

1 はじめに

近年，スマートフォンやパソコンなどのプロセッサでは，高性能化と低電力化の両立が求められており，様々な高速化・低消費電力化手法が提案されている．そのうち，高性能化の手法の一つとして複数のコアを用意し，並列に処理を行うマルチコア化が挙げられる．しかし，マルチコア環境では複数のメモリアクセスが同時に発生することで，シングルコア環境よりデータの競合が発生しやすくなり，結果としてキャッシュミス率が高くなるという問題がある．メモリアクセスは性能のボトルネックの一つであるため，高性能化を達成するためには主記憶へのメモリアクセスの低減，すなわちキャッシュミス率の低減が重要である．

この問題を解決する手法としてキャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュが提案されている [1]．セル・アロケーションキャッシュは，キャッシュ・パーティショニング [2] を応用した手法で，メモリを「セル」と呼ばれる細かい単位に分割・管理することで高性能と低消費電力の両立を目指した手法である．しかしながら，これまでは事前にメモリアクセスの全履歴を保存し，その履歴を再現することでシミュレーションを行うトレースドリブン・シミュレータでしか評価されていなかった．一般に並列処理

では他のスレッドやプロセスとの依存関係によりプログラムの振る舞いや性能が変わり，その結果メモリアクセスパターンも変動するため，より現実的なキャッシュの評価を行うためにはアーキテクチャレベルのシミュレーションを行うことが望ましい．

そこで，本研究では，まずセル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 上に実装する．トレースドリブン・シミュレータでは効果が得られなかったマルチスレッドベンチマーク単体を用いて再評価した結果，通常キャッシュや従来の固定的なキャッシュ・パーティショニングと比較して，ヒット率がそれぞれ最大 44.3 %，5.2 % 向上した．また，より現実的な評価を行うために，負荷の違うベンチマークを複数組み合わせた複合ベンチマークを作成し，それによってロードやメモリアクセスのようなプログラムの振る舞いが頻繁に変化する場合を評価した．その結果，ヒット率がそれぞれ最大 11.8 %，11.7 % 向上した．

さらに，本論文ではセル・アロケーションキャッシュのパラメータ自動最適化手法も提案する．セル・アロケーションキャッシュには，割り当てられたコアのみが主記憶からキャッシュへ書き込める固有セルと，通常キャッシュと同様にどのコアからでもアクセスできる共有セルの 2 種類が存在する．この 2 種類のセルの切り替えにより各コアに最適なキャッシュ

領域を与え、効率的にデータを記憶する。セルの切り替えには事前に与えた閾値を使用し、データの共有率がその閾値を超えれば固有セルから共有セルに切り替わる。本論文で行った評価結果から、閾値によってセル・アロケーションキャッシュのヒット率が大きく変わることが分かった。そのため、セル・アロケーションキャッシュで高性能を出すためには、最適な閾値の算出が重要であるが、閾値は実行する計算機環境やプログラム、データにより変動するため、最適値を事前に求めるのは困難である。そこで、本研究では Phase Detection を用いてプログラムの領域を分割し、その領域がどのようにプログラムの振る舞いとして動いているのかを把握した上で、従来、固定的に手動で定めていた閾値を各領域に応じて自動で設定し、新たな領域割り当てを行うことを提案した。これによりさらなるセル・アロケーションキャッシュの性能向上を目指した。

本論文は、以降次のように構成される。まず、第2章では、キャッシュシステムについて簡単に説明する。次に、第3章では従来のキャッシュパーティショニング、及びセル・アロケーションキャッシュについて概括し、第4章でその問題点を指摘する。第5章では提案手法について説明し、第6章で評価を行う。第7章では、評価結果から新たに適応的な閾値の自動決定手法の提案をする。最後に、第8章で結論と今後の展望について述

べる.

2 キャッシュシステム

キャッシュはコアと主記憶の間に設置される，主記憶より小容量で高速なメモリである．使用頻度が高いデータをキャッシュに格納し，キャッシュから読み出すことにより，主記憶へのアクセスを減らし，実行時間を抑えられる．最も単純なキャッシュは，アドレスから格納先が一意に決まるダイレクトマッピングキャッシュである．

2.1 ダイレクトマッピングキャッシュ

ダイレクトマッピングキャッシュの概念図を図 2.1 に示す．

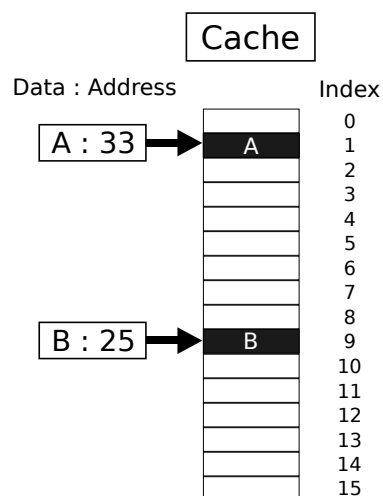


図 2.1: ダイレクトマッピングキャッシュの概念図

一般にダイレクトマッピングキャッシュはアドレスから剰余を用いて，インデックスと呼ばれる番地を計算する．具体的にはメモリアドレスを

address, キャッシュの管理単位であるライン長を line, キャッシュに格納できるデータ数を表す総エン트리数を entry とすると, インデックス index は下記の式 (1) で求まる (mod は剰余を表す).

$$\text{index} = \frac{\text{address}}{\text{line}} \bmod \frac{\text{entry}}{\text{line}} \quad (1)$$

そして, キャッシュの中のインデックスが一致するエントリに格納する. 例えば図 2.1 のように, ライン長が 1, エン트리数が 16 のキャッシュの場合を考える. データ A のアドレスが 33 の場合, $33 \bmod 16 = 1$ となり, インデックスが 1 のエントリに格納する. 次にデータ B を格納する. データ B のアドレスが 25 の場合, $25 \bmod 16 = 9$ となり, インデックスが 9 のエントリに格納する. このようにインデックスにばらつきを持たせて, なるべく異なる場所にデータを格納していく. しかし, 同じインデックスのデータを格納する場合, 現在格納しているデータを追い出してしまう. もし使用頻度の高いデータが追い出されると, キャッシュミスが増加し, 主記憶へのアクセスが増えて実効性能が悪化する. そこで, 多くのプロセッサではエントリを水平方向に増やした, セットアソシアティブキャッシュが用いられている.

2.2 セットアソシアティブキャッシュ

セットアソシアティブキャッシュの概念図を図 2.2 に示す。

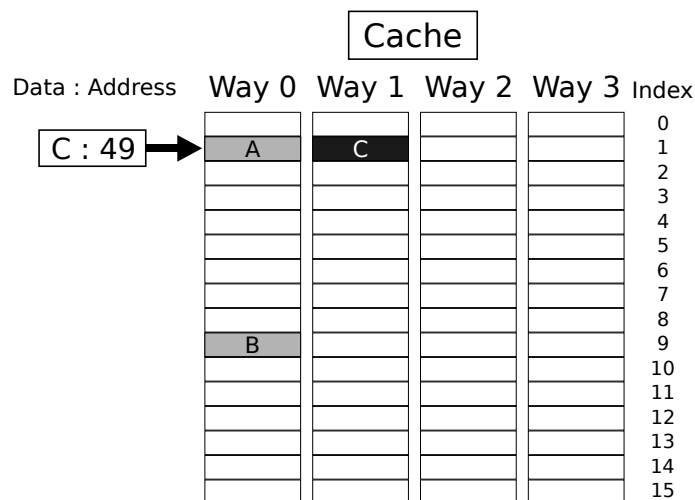


図 2.2: セットアソシアティブキャッシュの概念図

セットアソシアティブキャッシュは、1つのエントリに複数のエントリを配置、すなわちウェイ数を増加し連想度を上げる。これにより、同じインデックスのデータを一度に複数保持できる。なお、同一インデックスの複数のエントリをまとめてセットと呼ぶ。

例えば図 2.2 のように、ウェイ 0 にデータ A とデータ B が格納されており、次にデータ C を格納する場合を考える。データ C のアドレスが 49 の場合、 $49 \bmod 16 = 1$ となり、インデックスが 1 のエントリに格納しようとする。ここでダイレクトマッピングキャッシュでは、データ A とインデックスが同じであるため、データ A を追い出す。しかしセットアソ

シアティブキャッシュでは、図 2.1 のようにセット 1 のウェイ 1 が空いているため、ウェイ 1 に格納できる。次にデータ A を読み出す場合、ダイレクトマッピングキャッシュではキャッシュミスするが、セットアソシアティブキャッシュでは、キャッシュヒットとなり、ミスを低減できる。

このようにセットアソシアティブキャッシュでは、同じインデックスのデータを複数格納できるため、データの追い出しが減少し、主記憶へのアクセスが減り実効性能の低下を防げる。しかし、マルチコア環境ではコア同士で特定のインデックスを奪い合うと、データの追い出しが増加する恐れがある。

3 先行研究

3.1 キャッシュ・パーティショニング

従来より，シングルコア用キャッシュシステムにおいても，データの競合を避けるためセット・アソシアティブキャッシュが広く用いられている．セット・アソシアティブキャッシュは，ウェイ数を増やすことで，同じインデックスのデータを複数格納できるためデータの競合が減少し，メモリアクセスの削減に繋がる．しかし，マルチコア環境においては各コアが異なるアドレス空間にアクセスするため，メモリ競合が発生しやすい．この問題を解決する手法の一つとして，コアにウェイを動的に割当て，ミス減少数でウェイ数を調節し，各コアに最適な領域を確保することで競合を低減するキャッシュ・パーティショニング [2] がある．プロセス同士が独立である場合，データを共有しないため，メモリ競合を引き起こしやすい．そこでキャッシュ・パーティショニングは，各コアの負荷に応じてウェイを割り当てることで，メモリ競合を防ぐ．キャッシュ・パーティショニングでは，各コアの負荷を 1 ウェイあたりのミス減少数で判断する．各コアのミス減少数が同程度の場合，ウェイはほぼ均等に割り当てられる．ミス減少数の低いコアが，よりメモリを必要とする場合は，ミス減少量の高いコアの保持するキャッシュを渡す．割り当てられたウェイ

は、そのコアからしかアクセスできない。ウェイにアクセスできるコアを制限すると、各コアが使うデータのみが割り当てられたウェイに格納される。そのため、別のコアのデータによる追い出しが減少する。このように動的にウェイをコアに割り当てることで、複数のコアによるメモリ競合を減らし、キャッシュを効率良く使うことができる。しかし、未使用のウェイでは無駄に電力を消費する。また、実際のプログラムで使われる、共有データについては考慮外である。

3.2 セル・アロケーションキャッシュ

3.2.1 セル・アロケーションキャッシュの概要

セル・アロケーションキャッシュはキャッシュ・パーティショニングを応用した手法で、より細かい粒度でキャッシュメモリを管理できる。一般にマルチコアプロセッサでは各コアが共有キャッシュメモリを奪い合って性能が落ちることがある。そこで、キャッシュ・パーティショニングではコアにウェイを割当て、コア間での競合を解消している。しかしながら、プログラムには時間的局所性、空間的局所性があり、アクセスするメモリ領域は偏っている。そのためコアにウェイを丸ごと割り付けるキャッシュ・パーティショニングでは、使用頻度の低い領域ができてしまい、キャッシュを有効活用できない危険性がある。そこで、セル・アロケーションキャッ

シユでは1個のウェイを複数の細分化した領域である「セル」を単位としてコアに割当てる．セル・アロケーションキャッシュの概念図を図 3.3 に示す．

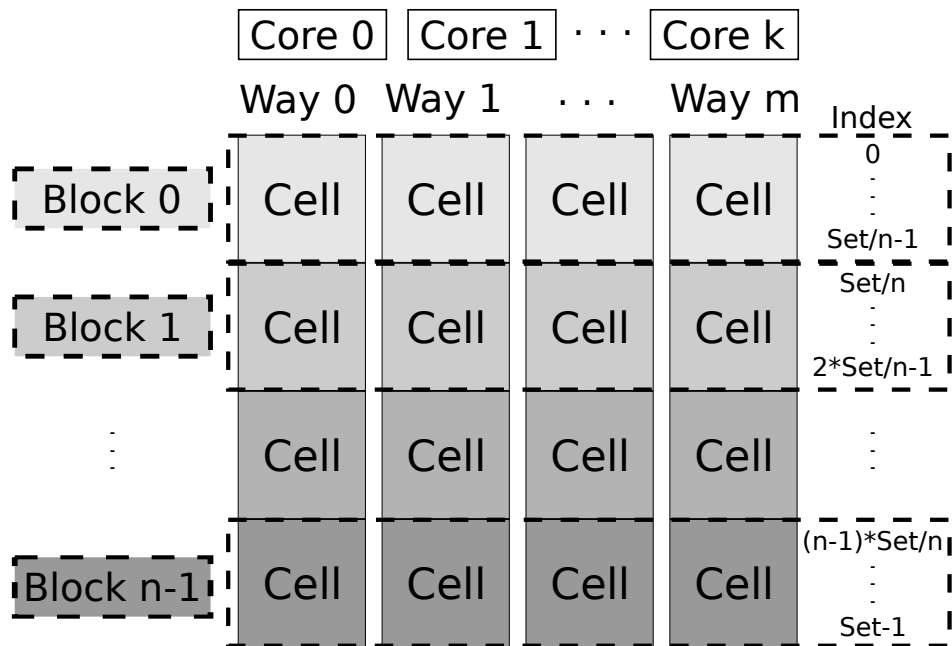


図 3.3: セル・アロケーションキャッシュの概念図

セルは複数のエントリをまとめたグループである．また，図 3.3 の破線のように，ウェイ方向に並んだセルのグループをブロックとする．図 3.4 にセルの詳細を示す．セルはエントリごとにブロック番号を格納するビット列を持つ．また，データ共有を考慮し，各セルをデータ共有率とミス率に応じて，共有セルと固有セルに分ける．共有セルは通常キャッシュと同様にアクセスできるセルである．一方，固有セルは，キャッシュミス時の

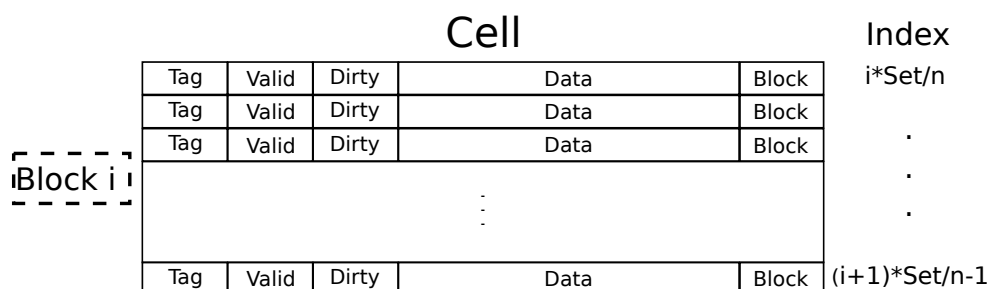


図 3.4: セルの詳細

リプレース対象先を、割当てられたコアに限定するセルである。この2種類のセルを使い分けることで、データを共有しつつ、キャッシュメモリの競合を減らし、キャッシュの利用効率を高める。

3.2.2 セル・アロケーションキャッシュの動作

セル・アロケーションキャッシュと通常キャッシュとの違いは、キャッシュミス時のデータの格納先を決める場合に、同じコアに属する領域へ優先的に書き込む点である。共有セルと固有セルのキャッシュミス時の動作を説明する。あるコアでキャッシュミスが発生した場合、メモリからデータを取得する。その際の書き込み先は、共有セルかそのコアの固有セルが優先される。共有セルとそのコアの固有セルの両方が存在する場合、その中からLRU法でどのセルに書き込むか決定する。共有セルとそのコアの固有セルに書き込めない場合にのみ、他のコアの固有セルを共有セルにし

て書き込む。共有セルが無く、そのコアの固有セルが存在する場合、固有セルの中から LRU 法でどの固有セルに書き込むか決定する。そのコアの固有セルに書き込めない場合にのみ、他のコアの固有セルを共有セルにして書き込む。

4 従来研究の問題点

4.1 トレースドリブンシミュレーションによる評価

文献[1]では、トレースドリブンベースのシミュレーションにより性能評価を行っている。トレースドリブンシミュレーションは事前にベンチマークプログラムを実行してメモリアクセスを記録し、それを用いてオフラインでシミュレーションを行うものである。しかし、一般に並列プログラムでは、他のスレッドやプロセスの動きにより、プログラムの振る舞いや性能が大幅に変わる場合がある。例えば、マスタ・ワーカー・モデルのプログラムの場合、スレッド間の僅かなタイミング差で処理の割り当てや振る舞いが大幅に変わる可能性がある。また、トレースドリブンシミュレーションでは、命令間の依存関係は表現できないため、ロード命令の遅延が他の後続命令に与える影響を正確にシミュレーションできない。図4.5にロックプログラムにおけるトレースドリブンシミュレーションの問題点を示す。

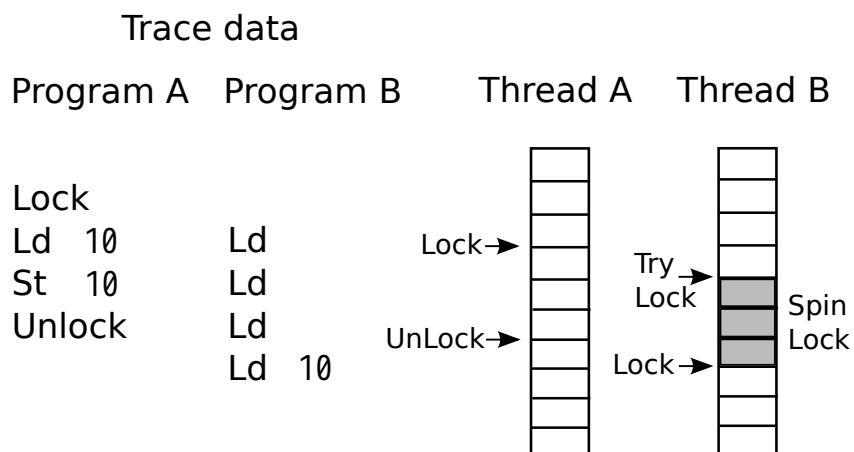


図 4.5: ロックプログラムの例

図 4.5 のように, 2 つのプログラムを 2 スレッドのマルチスレッドで並列に動かしている時に, A のスレッドで先にロックのプログラムが発生した場合, B のスレッドでも後にロックが発生すると, B がロックを獲得するまでスピンロックになって待つ時間ができてしまう. しかし, 実際の環境ではこのスピンロックが互いのスレッドの挙動によって, 発生しない可能性がある. トレースドリブンシミュレーションではこれを模擬することができない.

5 提案手法

5.1 提案評価手法

文献[1]では,トレースドリブンベースのシミュレーションにより性能評価を行っている.具体的には,事前にベンチマークプログラムを実行してメモリアクセスを記録し,オフラインのトレースドリブン・シミュレータを用いて評価を行っていた.しかし,一般に並列プログラムでは,他のスレッドやプロセスの動きにより,プログラムの振る舞いや性能が大幅に変わる場合がある.トレースドリブンシミュレーションでは,命令間の依存関係は表現できないため,ロード命令の遅延が他の後続命令に与える影響を正確にシミュレーションできない.その問題を解決するために,本研究ではより正確な評価,及び動作解析を目指して,アーキテクチャシミュレータである Gem5 にセル・アロケーションキャッシュを組み込むことを提案する.

5.2 複合ベンチマークを用いた評価の提案

姫野ベンチマークのみを使用したシミュレーションでは,セル・アロケーションキャッシュとキャッシュパーティショニングを比較した際,ヒット率の向上の差が小さかった.姫野ベンチマークのみを使用してシミュ

レーションした場合を図 5.6 に示す.

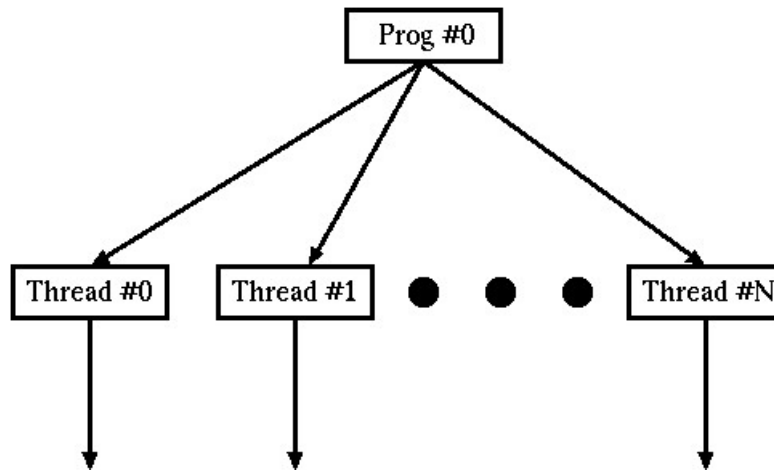


図 5.6: 画一化されたメモリアクセス

この場合, それぞれのコアやスレッドにかかる負荷はほぼ同じになるため, そのような結果になったと考えた. より現実的な環境下でセル・アロケーションキャッシュを評価するにはロードやメモリアクセスのようなプログラムの振る舞いが頻繁に変化するため, 図 5.7 に示されるような複合ベンチマークを使ったシミュレーションをする必要がある.

しかし, Gem5 はシングルスレッドのマルチプログラムまたは 1 つのマルチスレッドプログラムのどちらか 1 つしかサポートしていない. そして, ほとんどのマルチスレッドのベンチマークプログラムでは効果的にプロセッサコアを使用するため, すべてのスレッドの振る舞いやメモリ

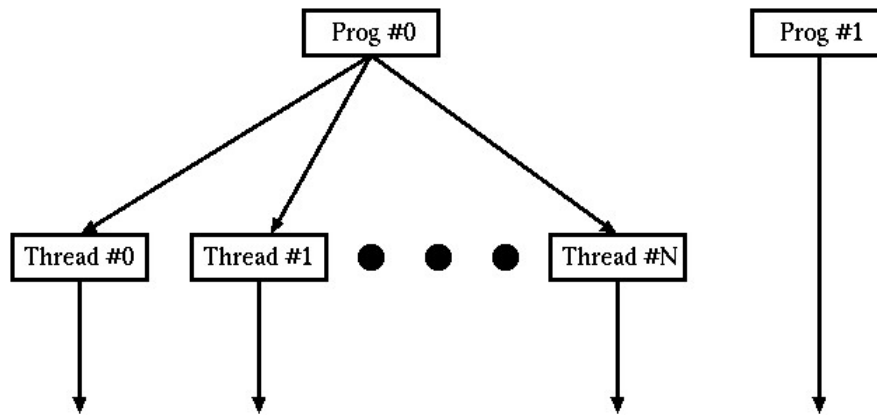


図 5.7: 画一化されていないメモリアクセス

アクセスのパターンがほぼ同じになる．そのため，本研究では Gem5 における図 5.7 に示されるような複数のマルチスレッドプログラムを用いたシミュレーション手法を提案し，セル・アロケーションキャッシュによる効果を検証する．具体的には，MiBench[4] や SPLASH2, SPEC ベンチマークのようなベンチマークのメインルーチンをスレッド生成やスレッド待ちの間に姫野ベンチマークのメインファンクションに挿入する．

6 性能評価

6.1 評価方法

セル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 の L2 共有キャッシュに実装し、評価した。比較対象は通常キャッシュと 1 コアあたり 2 ウェイを割り当てた場合の固定的なキャッシュ・パーティショニングとする。評価項目は高性能を目標とするためキャッシュヒット率とする。評価環境は表 6.1 のとおりで、L2 共有キャッシュの全容量は 2MB とする。

表 6.1: 評価環境のパラメータ一覧

スレッド数	4 threads
コア数	4 cores
ウェイ数	8 ways
セル数	16 cells

またベンチマークとして姫野ベンチマーク [3] と複合ベンチマークを使用し、評価した。複合ベンチマークは MiBench[4] と姫野ベンチマークを組み合わせたもので、姫野ベンチマーク単体と比較して、それぞれのコアやスレッドにかかる負荷を変えている。今回 4 コア中、1 コアを MiBench, 3 コアを姫野ベンチマークとする。モードを切り替えるインターバルはロードストア命令数が 5, 10, 20, 25, 30[$\times 10^4$] の倍数の時とする。また、

共有データかどうか判定する閾値は 0.00 から 0.50 までを 0.05 刻み, 0.50 から 1.00 までを 0.1 刻みで変化させ, 評価した. なお, 本研究では 2 つ以上のスレッドから一度でもアクセスのあったものを共有変数と考え, セル内のデータのうち閾値を超える割合の共有変数があれば共有セルに遷移させる. 共有セルはどのコアでも読み書き, リプレースできる.

6.2 評価結果

図 6.8, 図 6.9 に改造した Gem5 と各ベンチマークを用いて評価したセル・アロケーションキャッシュのヒット率を示す.

閾値が 0.00 の時は通常キャッシュと, 1.00 の時は 1 コア当たり 2 ウェイを割り当てた理想的なキャッシュ・パーティショニングと同等となっている. 姫野ベンチマークのみを使用した結果を図 6.8 似に示す. 通常キャッシュ, 従来のキャッシュ・パーティショニングと比較して, ヒット率はそれぞれ最大 44.3 %, 5.2 % 向上した. また, 閾値 0.30, インターバル $20[\times 10^4]$ の時, 最もヒット率が高く 38.03 % になった. 閾値 0.3 の時が平均して最もヒット率が高く, それ以降はヒット率が伸びない傾向になり, インターバルが違っていてもヒット率はほぼ同じくらいになった. 最終的に閾値 1.0 の時には全てヒット率が同じになった. また, 閾値が 0.00 から 0.25 ま

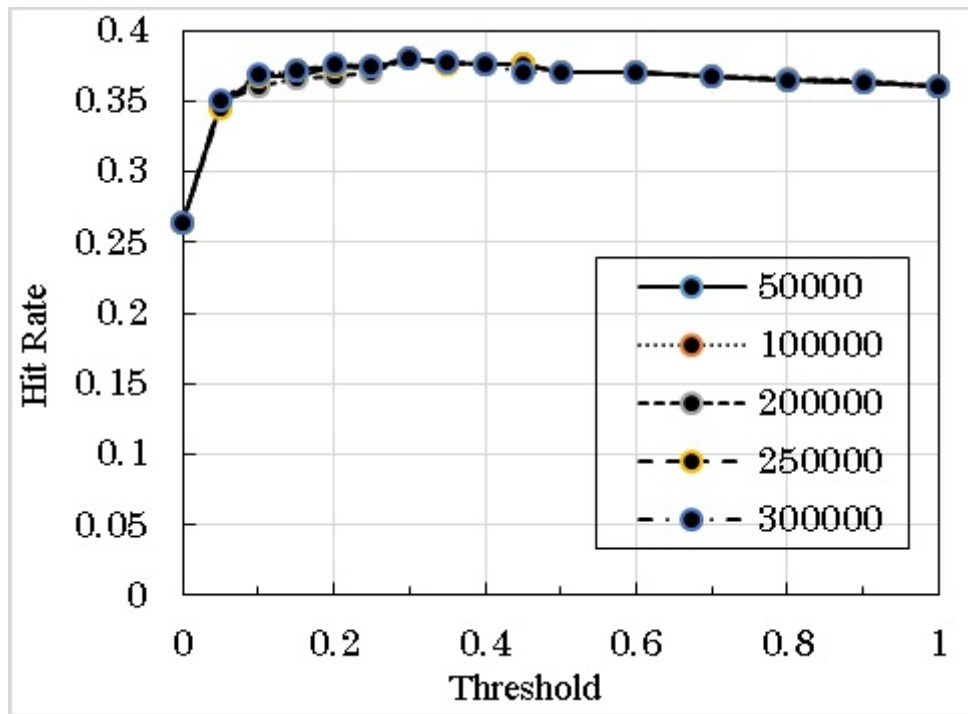


図 6.8: キャッシュ・ヒット率 (姫野ベンチマーク)

ではインターバル 30×10^4 でヒット率が最大になったが、0.30 から 0.90 までは 20×10^4 で最大になった。キャッシュ・パーティショニングと比較して、閾値によって全体的にヒット率が向上したため、セル・アロケーションキャッシュによる効果があることを確認できた。閾値は 0.30 の時、共有セルと固有セルの使い分けが効率良く行われたため、ヒット率が一番良くなったと考えられる。それ以降の閾値では、共有率が超えられなくなったため、固有セルが多い状態となりキャッシュ・パーティショニングに近い状態になったため、ヒット率は最大時に比べて低下したと考え

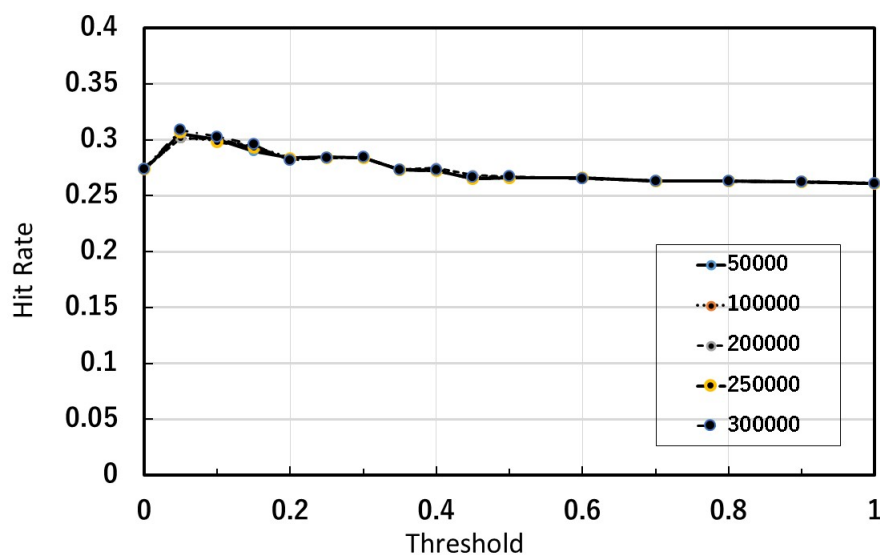


図 6.9: キャッシュ・ヒット率 (複合ベンチマーク)

られる。また、複合ベンチマークを使用した結果を図 6.9 に示す。こちらは、通常キャッシュ、従来のキャッシュ・パーティショニングと比較して、ヒット率はそれぞれ最大 11.8 %, 11.7 % 向上し、どちらよりもヒット率が向上した。また、各インターバルにおいて閾値 0.05 の時、最もヒット率が向上した。しかし、複合ベンチマークでは閾値 0.35 以上の時は、通常キャッシュよりも、セル・アロケーションキャッシュの方がヒット率が低くなってしまう傾向になった。また、通常キャッシュと比較してキャッシュパーティショニングによる効果が得られなかった。

6.3 考察

トレースドリブンシミュレーション [1] では共有セルと固有セルの導入による性能向上は見られなかった。しかし、今回、アーキテクチャレベルのシミュレーションをすることで通常キャッシュ、キャッシュ・パーティショニングと比較して、ヒット率が上がり性能向上を確認できた。また姫野ベンチマークのみを使用した時、トレースドリブンシミュレーションでは通常キャッシュに比べて性能が落ちていたが、今回、大幅なヒット率向上が見られた。これらのことからセル・アロケーションキャッシュの効果が検証できたと考えられる。またそれぞれのベンチマークにおいて、ヒット率はインターバルごとにはあまり差が見られなかったが、閾値ごとには大きく差が出た。これにより、適切な閾値設定によって大幅な性能向上が可能であると考えられる。

7 適応的な閾値の自動決定手法の提案

Gem5 による評価結果より固有セルから共有セルに切り替えるために使用する閾値によって，ヒット率が大きく変わることが新たに分かった．従来，閾値は固定的に定めておりプログラムの始めから終わりまで同じ値であった．そこで，本研究では適応的な閾値の自動決定手法の提案をする．閾値決定の概要を図 7.10 に示す．

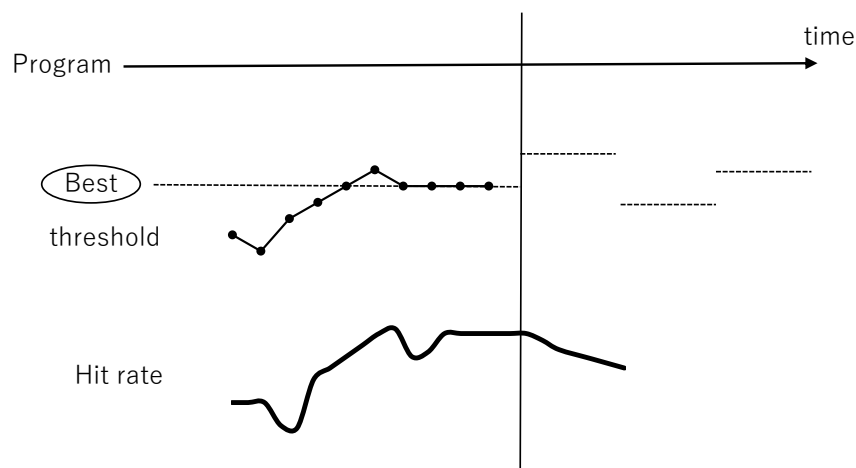


図 7.10: 閾値決定の概要

プログラムの時間的局所性，空間的局所性からデータをアクセスする場所は偏ってくる．その時，一定のインターバルごとのヒット率に応じて，前のインターバルのヒット率が上昇していれば次のインターバルで

の閾値を増やしていき、その後、低下した時は閾値を下げる．これを収束するまで繰り返すことで、そのときの最適な閾値を決定することが可能となる．しかしこの手法では局所性のある部分のみしか対応できない．そこで Phase Detection[6] を用いた閾値自動決定手法を提案する．Phase Detection はプログラムを領域ごとに分割し、その領域がどのようにプログラムの振る舞いとして動いているかを把握するものである．Phase Detection を用いた閾値自動決定手法の概要を図 7.11 に示す．

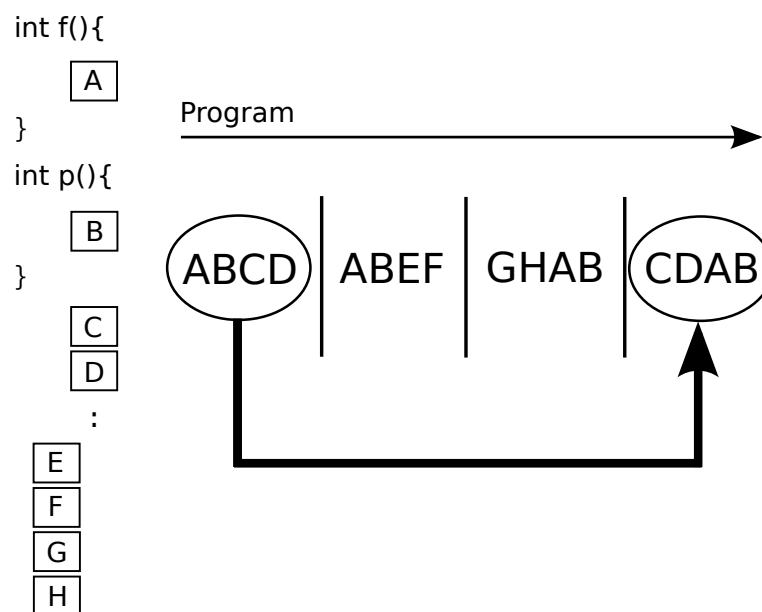


図 7.11: Phase Detection を用いた閾値自動決定手法の概要

これによって前のフェーズと似たフェーズがあればそのときに定めた最適な閾値をすぐに反映させることが可能となる．この提案手法のハー

ドウェアブロック図を図 7.12 に示す。

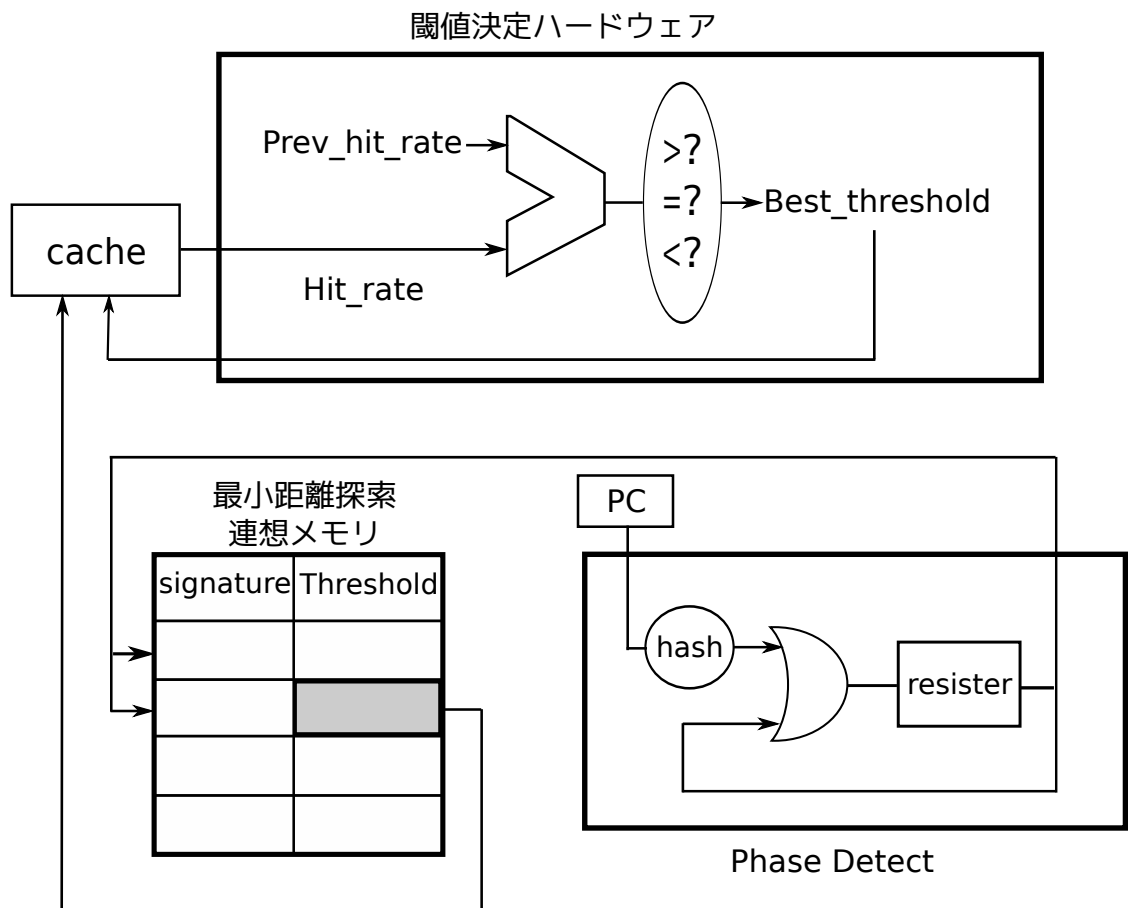


図 7.12: 提案手法のハードウェアブロック図

具体的には、まず前のフェーズのヒット率と現在のフェーズのヒット率を比較し、ヒット率が上がったか下がったかによって最適な閾値を求めていく。それに加え Phase Detection を用いる部分では、まずプログラムカウンタのハッシュ関数を取り、それを 2 ビットシフトすることでどこか

のビットに1が立つ．これを一定時間ごとに繰り返す．そうすることで，各領域が実行された時に1が立つ位置がわかり，それをシグネチャとして取る．そしてそのシグネチャ同士のハミング距離（排他的論理和をとったときの1の数）をとる．そうすることで同じような領域のパターンをプログラムが実行しているところはハミング距離が小さくなり，逆に違う領域を実行しているところは大きくなる．ハミング距離が大きくなった場合，最小距離探索連想メモリ [7] によって以前実行したフェーズの中から一番類似しているフェーズの時の最適な閾値と同じものを利用して閾値を新たに定める．メリットとしてブロック図の通り Phase Detection を使用する機能は小さなハードウェア規模で実現が可能である．これによりさらなるセル・アロケーションキャッシュのヒット率向上が可能であると考えられる．

8 おわりに

近年，スマートフォンやパソコンなどのプロセッサでは，高性能化と低電力化の両立が求められており，様々な高速化・低消費電力化手法が提案されている．そのうち，高性能化の手法の一つとして複数のコアを用意し，並列に処理を行うマルチコア化があるが，マルチコア環境では複数のメモリアクセスが同時に発生することで，シングルコア環境よりデータの競合が発生しやすくなり，結果としてキャッシュミス率が高くなるという問題がある．そこで，この問題を解決する手法としてキャッシュをウェイより細かい単位である「セル」に分割して動的にコアに割り当てるセル・アロケーションキャッシュが提案されているが，トレースドリブン・シミュレータでしか評価していなかった．そこで，まず本研究では，セル・アロケーションキャッシュをアーキテクチャシミュレータ Gem5 上に実装する．より現実的な環境下での評価を行うために，トレースドリブン・シミュレータでは効果が得られなかったマルチスレッドベンチマーク単体を用いて評価した結果，通常キャッシュや従来の固定的なキャッシュ・パーティショニングと比較して，ヒット率がそれぞれ最大 44.3 %，5.2 %向上した．また，より現実的に近い評価を行うために，負荷の違うベンチマークを複数組み合わせた複合ベンチマークを作成して評価することを提案

し、それによってロードやメモリアクセスのようなプログラムの振る舞いが頻繁に変化する場合を評価した。その結果、ヒット率がそれぞれ最大 11.8 %,11.7 % 向上した。さらに、評価結果からセル切り替えのための閾値によって、セル・アロケーションキャッシュのヒット率が大きく変わることが新たに分かった。そこで、Phase Detection を用いてプログラムの領域を分割し、その領域がどのようにプログラムの振る舞いとして動いているのかを把握した上で、従来、固定的に手動で定めていた閾値を各領域に応じて自動で設定し、新たな領域割り当てを行うことを提案した。これによりさらなるセル・アロケーションキャッシュの性能向上を目指した。

今後の展望としては、セル・アロケーションキャッシュでは動的な固有セルの割り当て数の変更や疑似的なウェイ拡張等も行っているが、今回は反映していないのでその実装すること、共有セルに一度なったセルは固有セルに戻らないので戻るようにすること、セル割り当てアルゴリズムの改良が挙げられる。また、提案した適応的閾値自動決定手法を詳細設計し、性能評価や消費電力評価を行うことが挙げられる。

謝辞

本研究の作成にあたり，ご指導を頂いた大野和彦講師，佐々木泰敬准教授，山田俊行講師にお礼申し上げます．

参考文献

- [1] 刀根舞歌，佐々木敬泰，深澤祐樹，近藤利夫，“キャッシュの分割領域の動的管理による高速化”，電子情報通信学会技術研究報告，Vol.116，No.177，pp.119-124，August 2016.
- [2] G. E. Sue, L. Rudolph, and S. Devadas, “Dynamic Partitioning of Shared Cache Memory,” Journal of Supercomputing, vol.28, no.1, pp.7-26, January 2004.
- [3] R.Himeno, “Himeno Benchmark,” <http://accc.riken.jp/supercom/himenobmt/>, (accessed July 2016 in Japanese).
- [4] M. R. Guthaus, et al., “MiBench: A free, commercially representative embedded benchmark suite,” Proc. of the Fourth Annual IEEE International Workshop on Workload Characterization, pp.3-14.

- [5] N. Rafique, et al., “Architectual support for operating system-driven CMP cache management , ” in 15th PACT, pp. 2-12, 2006.
- [6] Shaymaa M. Seraga, et al., “ A Run-Time Program Phase Detection Technique for Optimizing Per-Phase L2 Cache Demand ” EIJEST vol. 20, pp. 1-9, July 2016.
- [7] T. Koide, et al., “ A nearest-Hamming-distance search memory with fully Parallel mixed digital-analog match circurity ” , Proc. of Asia and South Pacific Design Automation Conf ., pp.591-592, Jan 2002.