修士論文

題目

ヘテロジニアスマルチコア対応のキャッシュシステム自動生成ツールの研究

指導教員

近藤　利夫　教授

2015年

三重大学 大学院 工学研究科
博士前期課程 情報工学専攻
計算機アーキテクチャ研究室

岡本 昂樹 (413M505)

内容梗概

近年，プロセッサアーキテクチャの分野において，シングルパイプラインコアの並列度をより高めたスーパースカラコアを，1 つのチップに複数搭載したヘテロジニアスマルチコアの研究が注目されている．ヘテロジニアスマルチコアシステムは，アプリケーションプログラムの特徴によって最適なスーパースカラコアを使用する事で高性能と省電力を両立している．しかし，ヘテロジニアスマルチコアプロセッサの設計・検証を行う過程で，最適な構成のスーパースカラコア，それに対応したキャッシュシステムや共有バスシステムの組み合わせを手動で設計するには，膨大な時間が必要となり困難である．そこで，当研究室では，様々な構成のスーパースカラコアを自動生成するツール FabScalar，キャッシュシステムを自動生成するツール FabCache，また共有バスシステムを自動生成する FabBus を用いてヘテロジニアスマルチコアシステムを自動設計する FabHetero というプロジェクトを研究している．本研究では，キャッシュシステム自動生成ツール FabCache について記述する．

特徴の異なるスーパースカラコアに対応したキャッシュシステムを手動で設計するには，キャッシュ容量，命令フェッチ幅，ラインサイズ，連想度，アクセスレイテンシ，キャッシュ階層間でのライン転送幅など，考慮すべきパラメータが多数ある事により困難である．そこで，FabCache は任意のパラメータを与えるだけで対応するキャッシュシステムを自動生成する事ができる．

本論文では FabCache の詳細な内部設計及び，手動で最適化されたキャッシュシステムと FabCache で自動生成されたキャッシュシステムの面積，遅延，電力の評価を行う．評価結果によると，自動生成によるオーバーヘッドを含む，FabCache によって生成されたキャッシュシステムの面積は約 3.5 %，遅延は 0.1ns，電力は 0.1 %以下程度の増加に抑えられた事により，手動で設計されたキャッシュシステムと遜色のない回路が生成されている事が確認できた．

Abstract

Single-ISA heterogeneous multi-core architecture which consists of diverse superscalar cores is increasing importance in the processor architecture. Using a proper superscalar core for characteristic in a program contributes to reduce energy consumption and improve performance. However, designing a heterogeneous multi-core processor requires a large design and verification effort. Therefore, FabHetero has been proposed which generates diverse heterogeneous multi-core processors automatically using FabScalar, FabCache, and FabBus which generate various designs of superscalar core, cache system, and flexible shared bus system, respectively. This paper presents the detail of FabCache and shows that the caches generated by FabCache with arbitrary parameter values such as cache capacity, line size, associativity, access latency, and line transmission width between cache hierarchies work correctly. This paper also focuses on performance estimation and the physical design of the caches. According to the estimation results, FabCache generates cache systems which have almost the same area and power consumption as hand-tuned cache because the ratio of L1 instruction and data cache controller including extra circuits is only 3.5% and the increased power consumption by comparing with hand-tuned cache is less than 0.1% although having the overhead of automatic generation.

目 次

図 目 次

表 目 次

1　はじめに

近年，特徴の異なるプログラムやプログラム中のフェーズを効率的に実行するために，構成の異なるプロセッサコアを複数個用いるヘテロジニアスマルチコアプロセッサが注目を集めている [1, 2, 3, 4]．構成の異なるプロセッサコアをプログラムの特徴に合わせて使い分ける事は，計算性能の向上や消費電力の低減に大きく貢献する．しかしながら，設計・検証にかかる時間がヘテロジニアスマルチコアプロセッサを研究・開発する上で大きな障害となっている．この問題を解決するために，様々な構成のスーパースカラコアの RTL(Register Transfer Level) コードを自動生成するツールセットとして FabScalar [5, 6, 7, 8, 9, 10, 11] が提案されている．FabScalar は，任意のパラメーターを与えるだけでフェッチ幅やイシュー幅，パイプライン段数等の構成が異なるスーパースカラコアを自動生成するツールであり，ヘテロジニアスマルチコアプロセッサの設計・検証にかかる時間を大幅に短縮できる．しかし，FabScalar が自動生成するのはプロセッサコア部分のみであり，それに付随する最適な構成のキャッシュシステムや共有バスシステムを自動生成する仕組が実装されていない．特にキャッシュシステムに関して，キャッシュ容量，ラインサイズ，連想度，階層やコヒーレンシプロトコルをはじめ，命令フェッチ幅

やメインメモリ間のデータ転送幅等考慮すべき要素が多数あり，これらの組み合わせから最適な構成を手動で設計するには膨大な時間がかかってしまう。この問題を解決するため，著者らの研究グループはヘテロジニアスマルチコアプロセッサを自動生成する FabHetero [12] を提案している。FabHetero は 3 つのツールから構成されており，スーパースカラコア生成に FabScalar，キャッシュシステム生成に FabCache [13]，共有バスシステム生成に FabBus [14, 15] を用いてヘテロジニアスマルチコアプロセッサ全体を自動で設計することができる。

以降，本論文は次のように構成する．まず，次章でヘテロジニアスマルチコア・キャッシュシステムについて，第 3 章では著者ら研究グループが開発を行う FabHetero プロジェクトについて説明する．第 4 章でキャッシュシステム自動生成ツールに関する先行研究について議論する．第 5 章で提案手法である FabCache について説明し，第 6 章でその実装方法の詳細を示す．最後に，第 7 章で今回提案・実装した FabCache について性能・面積・消費電力について評価する．

2　背景

2.1　ヘテロジニアスマルチコアプロセッサ

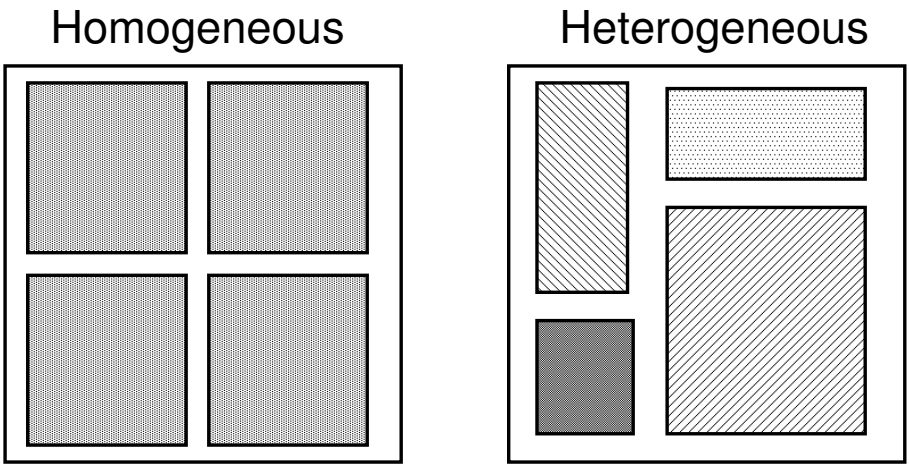

図 2.1: Homogeneous and Heterogeneous multi-core

現在，同じアーキテクチャの CPU コアを 1 チップに複数塔載するホモジニアスマルチコア (図 2.1．左) が広く使われている．ホモジニアスマルチコアでは，特性の違う様々なアプリケーションに対しアーキテクチャが同じコアで処理するためハードウェアリソースの過不足が生じてしまい，性能と電力効率を低下させる一因となっている．そこで性質の異なる複数のコアを組合せ，アプリケーション毎に適切なコアを割当てる事で高性能と省電力の両立を目指すヘテロジニアスマルチコア (図 2.1．右) の研究が注目されている．

ヘテロジニアスマルチコアは構成の異なる，複数のコアを組合せる事により高性能と省電力を両立している．しかし，各コアのフェッチ幅やパイプライン段数等の構成や，それに付随する最適な容量，ラインサイズやインターフェース，さらにはキャッシュコヒーレンンシのプロトコル，そしてキャッシュとコア，メインメモリを接続する共有バスシステムなど，考慮すべき組合せが膨大となってしまい，設計・検証に要する時間がヘテロジニアスマルチコアプロセッサを研究する上で大きな障害となっている．

2.2 キャッシュシステム

キャッシュシステムとは，プロセッサに併設されるメモリユニットのことであり，今日におけるプロセッサの約50%程度の面積と消費エネルギーを占めている．そのため，高性能かつ省電力アーキテクチャプロセッサの分野において，多数の研究者が注目している．

図 2.2 はL1，L2 キャッシュを持つキャッシュシステムの例を示している．キャッシュメモリはプロセッサとアクセス速度の遅いメインメモリとのデータ送受信を中継する小容量高速メモリである．現在，広く普及している市販の高性能プロセッサキャッシュメモリは2，3階層に分かれて

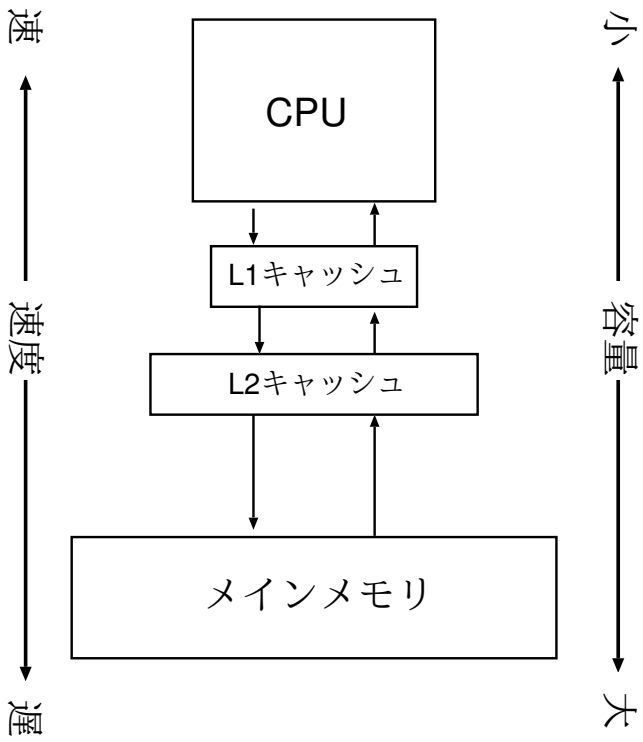

図 2.2: Example of Cache System.

おり，上階層のキャッシュメモリ程アクセス速度が高速かつ小容量となっており，大容量メモリへのアクセスレイテンシを隠蔽できる構成となっている．一方，シングルパイプラインプロセッサ等，組み込みシステムの分野においてよく用いられるプロセッサについては，キャッシュメモリを必要としない．

近年，第 2.1 節にて述べたように，高性能かつ省電力プロセッサ実現のため，このような種類の異なるプロセッサを 1 つのチップに混在させた

プロセッサが研究されている．しかし，前述したように，異種の各プロセッサが必要とする最適なキャッシュメモリが異なるため，組み合わせの観点から手動設計では困難である．そこで，本論文では，ヘテロジニアスマルチコア環境対応のキャッシュシステム自動生成ツールを提案・実装する．

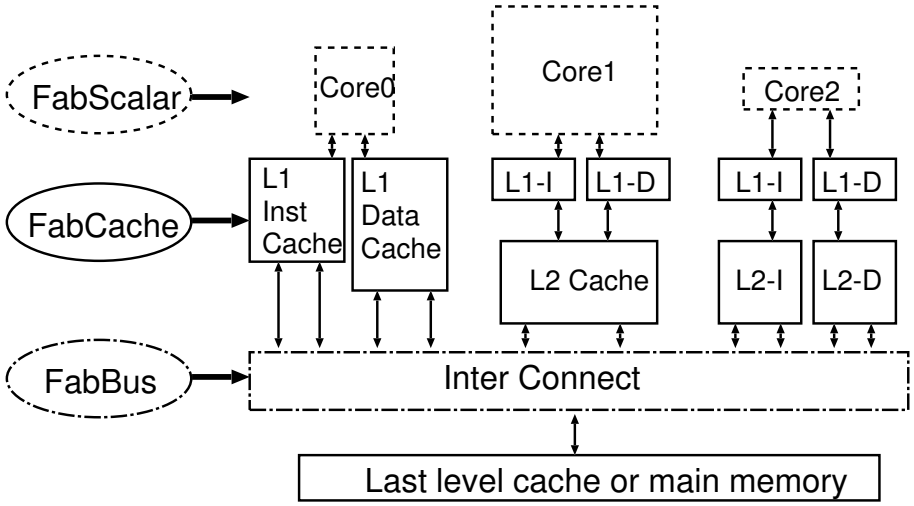

図 3.3: FabHetero

3 FabHeteroの概要

提案手法 FabCache の説明に入る前に，ヘテロジニアスマルチコアプロセッサ全体を自動生成する FabHetero プロジェクトについて説明する．最適な構成のヘテロジニアスマルチコアプロセッサを設計・開発するためには，プロセッサコアやキャッシュシステム，それらを結合する共有バスシステムに膨大な組み合わせが存在することから，非常に時間が掛かるという問題点がある．そこで著者ら研究グループは，様々な構成のプロセッサコアや，そのコアに最適なキャッシュシステム及び共有バスシステムを自動生成する FabHetero プロジェクトを研究している．FabHetero は，ノースカロライナ州立大学と共同で研究しており，様々な構成のスー

パースカラコアを自動生成する FabScalar，そのコアに最適なキャッシュシステムを自動生成する FabCache，また，それらを結合する共有バスシステムを自動生成する FabBus の 3 つのツールで構成され，ヘテロジニアスマルチコアを自動生成することができる．図 3.3 は，FabHetero によって生成されたヘテロジニアスマルチコアプロセッサの例である．3 つのコア (Core 0, Core 1, Core 3) はそれぞれ異なる構成のスーパースカラコアで生成されており，また異なるキャッスシステムを有している．共有バスは様々な構成のキャッシュシステムとラストレベルキャッシュ，もしくはメインメモリとを結合している．Core 0 は L1 命令キャッシュと L1 データキャッシュ，Core 1 は L1 キャッシュに加えて共有の L2 キャッシュ，さらに Core 2 は L1 キャッシュに加え，分散 L2 キャッシュで構成されており，また各キャッシュの容量やラインサイズ，命令フェッチ幅や連想度も異なる．L1 キャッシュと L2 キャッシュ階層は，コアがキャッシュシステムを必要としない組み込みプロセッサの設計を理想とする場合，生成させないことも可能である．このように，様々な構成のプロセッサコア，キャッシュシステム，共有バスシステムを自動生成するために，FabHetero では FabScalar, FabCache, FabBus をそれぞれ用いている．まず始めに，FabScalar, FabBus について説明し，著者の提案手法であるキャッシュシ

ステム自動生成ツール，FabCache の詳細について説明していく．

3.1 スーパースカラコアの自動生成

FabScalar は，N.K.Choudhary らによって提案されている，様々な構成を持つスーパースカラコアの論理合成可能な RTL デザインを自動生成するツールである [16]．FabScalar は，フェッチ・イシュー幅，パイプライン段数や ILP，ファンクションユニット等，様々な構成のスーパースカラコアを任意のパラメータを与えるだけで生成可能である．さらに，近年における高性能スーパスカラプロセッサの要求を満たすため，1〜8 命令フェッチ幅に対応しており，整列化制約を無視した任意のアドレスから連続した命令をフェッチすることが出来る．

Load store unit (LSU) では，アウトオブオーダを効率良く実行するため load store queue (LSQ) のサイズまで投機ロードを発行することが出来る．このような FabScalar の仕様により，対応するキャッシュシステムはノンブロッキング手法をはじめとする様々な構成が必要と考えられる．

3.2 バスの自動生成

FabBus はヘテロジニアスマルチコアプロセッサを想定とした，柔軟な共有バスシステムを自動生成するツールである．ヘテロジニアスマルチ

コアプロセッサにおいて，キャッシュとコア間での共有バスシステムは，各コアが有するキャッシュの階層が異なるため，複雑さを増している．この問題を解決するため，ヘテロジニアスマルチコアプロセッサ全体を設計するために必要な共有バスシステムを自動生成する FabBus が提案されている．FabBus は現在組み込みプロセッサで広く使用されている ARM 社製の AMBA プロトコルをベースとしている．

3.3 キャッシュの自動生成

構成の異なる複数のスーパースカラコアを持つヘテロジニアスマルチコアプロセッサが，プロセッサアーキテクチャの分野において注目されている．実行するアプリケーションプログラムに対し，最適な構成のスーパースカラコアを割り当てることで高性能と省電力を両立することができる．今日，多数の研究者がこのヘテロジニアスマルチコアプロセッサに注目し，高性能かつ省電力モバイルプロセッサの実現を目指している [1, 2, 3, 5, 17]．これらの研究から，キャッシュシステムが重要な要素であることが考えられる．そこで，B. de Abreu Silva らが異種混在型キャッシュシステムに焦点を当てている [17]．ヘテロジニアスマルチコア環境において，各コアに対して容量の異なるキャッシュシステムを割り当てることで平均的

なキャッシュミス率の低減を目指し，高性能かつ省電力プロセッサを実現している．しかし，実際にはキャッシュ容量だけでなく，ラインサイズ，キャッシュ階層，連想度，階層の異なるキャッシュ間のインターフェース等，高性能と省電力を実現する為に考慮すべきパラメータは多数存在する．さらに，ヘテロジニアスマルチコアシステムを対象とした研究を行うためには，様々な構成のプロセッサを想定しなければならないため，命令フェッチ幅やプロセッサとキャッシュ間のインターフェースも汎用的に対応できるように実装する必要がある．

そこで，ヘテロジニアスマルチコアシステムを設計するために必要な可変フェッチ幅，連想度，キャッシュ階層等のパラメータを設定することができるだけでなく，ノンブロッキング手法といった今日における高性能プロセッサの要求を満たすキャッシュシステムを自動生成するツール FabCache を提案・実装する．

4　先行研究

キャッシュシステムは，プロセッサの設計や仕様に依存することが多く，汎用性のあるキャッシュの自動生成に関する研究はあまり行われていない．キャッシュの自動生成に関する研究としては，[18, 19, 20, 21] があるが，対象とするプロセッサが固定であったり，ソースコードが生成スクリプト方式を採用しているため，可変性に乏しく，ヘテロジニアスマルチコア環境において最適なキャッシュシステムを構成するのは困難である．

本章では，その中でも代表的ないくつかを紹介する．

4.1　FPGA のキャッシュ自動生成ツール

提案手法の詳細に入る前に既存のキャッシュシステム自動生成ツールについて言及する．FPGA プロセッサを対象としたキャッシュシステム自動生成ツールが P. Yiannacouras らによって提案されている [18]．このツールを用いることで，様々なデータ格納構造，連想度，レイテンシ，キャッシュ階層を持ったキャッシュシステムを自動生成することにより，ターゲットとなるシステムに最適なキャッシュシステムを設計することができる．しかし，このツールが生成できる連想度はダイレクトマッピング，2 ウェイセットアソシアティブ，フルアソシアティブの 3 タイプなう

え，キャッシュ階層はパラメータ化されておらず，フェッチ幅もパラメータ化されていないことから異なる構成のスーパースカラコアに対応できないため，ヘテロジニアスマルチコアシステムを設計する際に使用することは困難である．

4.2 LEON のキャッシュ自動生成ツール

一方，Leon4 [19] はダイレクトマッピングからフルアソシアティブを含めた 2^n の連想度を設定でき，データ格納構造，キャッシュ階層，レイテンシもパラメータ化されている柔軟なキャッシュシステム自動生成ツールとなっている．しかし，ターゲットとなるシステムがスカラーパイプラインプロセッサであるため，異なる種類のフェッチ幅に対応することが不可能であり，異なる構成のスーパースカラコアに対応することができず，ヘテロジニアスマルチコアシステムの設計には用いることができない．その他にも多数のキャッシュシステム自動生成ツールが提案されているが，対象とするプロセッサが固定であったり，ソースコードが生成スクリプト方式を採用しているため，可変性に乏しく，ヘテロジニアスマルチコア環境において最適なキャッシュシステムを構成するのは困難である [20, 21].

5　キャッシュ自動生成ツールの提案

5.1　FabCache の概要

図 3.3 で示すように，ヘテロジニアスマルチコアでは，システムを構成する個々のプロセッサコアの特徴によって最適なキャッシュ構成が異なる．例えば，図 3.3 左の様に独立した L1 キャッシュのみを持つ構成や，図 3.3 中央の独立した L1 キャッシュに対して共有の L2 キャッシュを持つ構成，図 3.3 右の L1L2 それぞれ独立したキャッシュを持つ構成のコアなどに加え，命令キャシュのフェッチ幅を変更したり，L2 キャッシュの一貫性を保つアルゴリズムを変更したり，L2 キャッシュへのアクセスレイテンシを変更したりなど，最適なキャッシュ構成を手動で設計・評価するのは組合せの膨大さから非常に困難である．この問題を解決する為に，ヘテロジニアスマルチコアプロセッサ用のキャッシュシステムを自動生成する FabCache を提案する．

現在，広く普及しているホモジニアスマルチコアの場合では，プロセッサコアに対するキャッシュシステムを 1 つ設計するだけであったが，任意のパラメータのスーパースカラコアを生成する FabScalar を用いてヘテロジニアスマルチコアを設計する為，それぞれのコアに対する最適なキャッシュシステムを自動で生成しなければならない．異なる構成のキャッ

シュシステムを複数生成するという点において，FabCache は従来にはないキャッシュジェネレータとなっている．また，FabCache によって生成されたキャッシュシステムは，シングル又はデュアルポートメモリによって論理合成可能であるため，様々な構成のキャッシュシステムシミュレーションに使用でき，多ポートメモリで構成されていないため，スタンダードセルベース ASIC デザインに適していると考える。

5.2 生成可能なキャッシュシステム一覧

表 5.1 は現在の FabCache で設定可能なパラメータ一覧を示している．1 行目はキャッシュ階層を，2 行目はラインサイズ，セットサイズ，連想度の設定可能な範囲をそれぞれ示している．セットサイズは全ての階層で可変となっており，ラインサイズでは高階層キャッシュから 1 回のキャッシュアクセスで実行完了できるように多少の制約はあるが，可変となっている．また，全てのキャッシュ階層において，ダイレクトマッピングからフルアソシアティブを含む 2^n ウェイセットアソシアティブの連想度が設定可能である．3 行目については，各階層の特殊な内部アーキテクチャを示している．

表 5.1: Available designs in FabCache

Memory hierarchy	Dimensions(L = line size, S = set size, W = associativity)	Specific microarchitectures
L1 instruction cache	L = (fetch width to 2^n)×4(byte) S = 1 to 2^n W = 1, 2^n-way, full	**two banks interleaved** vs. non-interleaved **1 to 8 fetch width** **Interface with L2 cache** line size transmission vs. burst transmission **enable** vs. **disable**
L1 data cache	L = (1 to 2^n)×4(byte) S = 1 to 2^n W = 1, 2^n-way, full MSHR = 1 to 8 entry	**Miss handling** blocking vs. non-blocking **Writing approach** write-through vs. write-back Interface with L2 cache line size transmission vs. burst transmission **enable** vs. **disable**
L2 cache	L = wider than higher hierarchy S = 1 to 2^n W = 1, 2^n-way, full	**dedicated instruction and data** vs. **unified** **Cache coherency** **MOESI** vs. MOSI vs. MEI vs. **dedicated for each processor core.** interface with shared memory **processor num to/from one** vs. processor num to/from multi-ported memory cache replacement policy **LRU** vs. Pseudo-LRU **enable** vs. **disable**

5.3 インターフェースデザインの仕様

高階層レベルキャッシュ，またはメインメモリ間インターフェースの転送幅は，1 ワードから最大でパラメータファイルで指定したラインサイズ長まで指定できる．ラインサイズ長の転送は，一度にラインサイズ幅のデータを転送することができ，十分なバンド幅を持ったオンチップ通信

に用いることができる．また，バースト転送では，必要なラインのデータが全て揃うまで，ユーザーが定義したバス幅分のデータが毎サイクル転送される．ラインサイズ長の転送には，多数のI/Oピンが必要だが，I/Oピンの数を十分用意する事は難しい．このため，FabCacheでは，この転送幅を可変化し，パラメータとして指定する事で製作するチップに最適な転送方法・転送幅を選択できるようになっている．セットアソシアティブキャッシュのリプレース方法として，LRUを採用している．各階層のキャッシュを有効・無効にすることで，キャッシュ階層を変えることができる．

キャッシュを無効にした場合，キャッシュは100 %ヒットする理想的なメインメモリに直接アクセスすることで，パーフェクトヒットキャッシュとして振る舞い，シミュレーションに用いることができる．また，このキャッシュ無効モードは，各階層キャッシュのベストパフォーマンスを見積もる際にも使用可能である．

6　実装

本章では，提案手法における実装方法について述べる．詳細設計の説明に入る前に，実装戦略として採用したスーパーセット戦略において記述する．その後，L1 命令，データキャッシュ及び L2 キャッシュの概要について述べる．次に，今日における高性能プロセッサの仕様としてキャッシュシステムに要求される機能に向けての改良法を記述する．最後に，提案手法の移植性に関して述べる．

6.1　スーパーセット戦略

FabCache はハードウェア記述言語である SystemVerilog で記述されており，後述する特殊な記述法スーパーセット戦略を用いて様々な構成のキャッシュシステムを自動生成している．スーパーセット戦略とは，パラメータ化された全ての内部アーキテクチャが 1 つの RTL コードのソースファイルを共有する記述法である．対照的に，P. Yiannacouras [18] らが生成スクリプトを用いて RTL コードを作成する手法を提案している．この手法は，生成スクリプトがパラメータを解析し，そのパラメータを元にターゲットとなる RTL コードを生成する．スーパーセット手法と比べた生成スクリプトのメリットとしては，各パラメータ毎に最適化されたコー

ドが作成されることである．しかし，パラメータ毎に最適化したコードをあらかじめ記述しておくため，新しいアーキテクチャを実装する際，全ての生成スクリプトを記述し直さなければいけないという致命的な問題点がある．ヘテロジニアスマルチコアプロセッサ対応の新しい内部アーキテクチャを開発するという本質的な目標がFabCacheにはあるため，スーパーセット手法を採用した．生成スクリプトは一度アーキテクチャを実装した後，新しい機能を追加する際，バックアノテーションが必要な反面，スーパーセット戦略では直接RTLコードに実装可能なため，新機能追加が容易である．しかし，全ての内部アーキテクチャが1つのソースファイルを共有しているため，パラメータ数が極端に増加してしまうとコード可読性が低下してしまう，意図しないハードウェアが生成されてしまうという2つの問題点がある．コード可読性については，ユーザーにとって本質的な問題ではないが，特殊な実装方法を用いることにより解決している．また，意図しないハードウェアについては，第6.2節にて説明する特殊な実装方法により対処し，また，手設計による最適化されたキャッシュとFabCacheによって生成されたキャッシュの面積・遅延・電力消費を比較することにより，スーパセット戦略の妥当性を確認した．

6.2 L1 命令キャッシュの概要

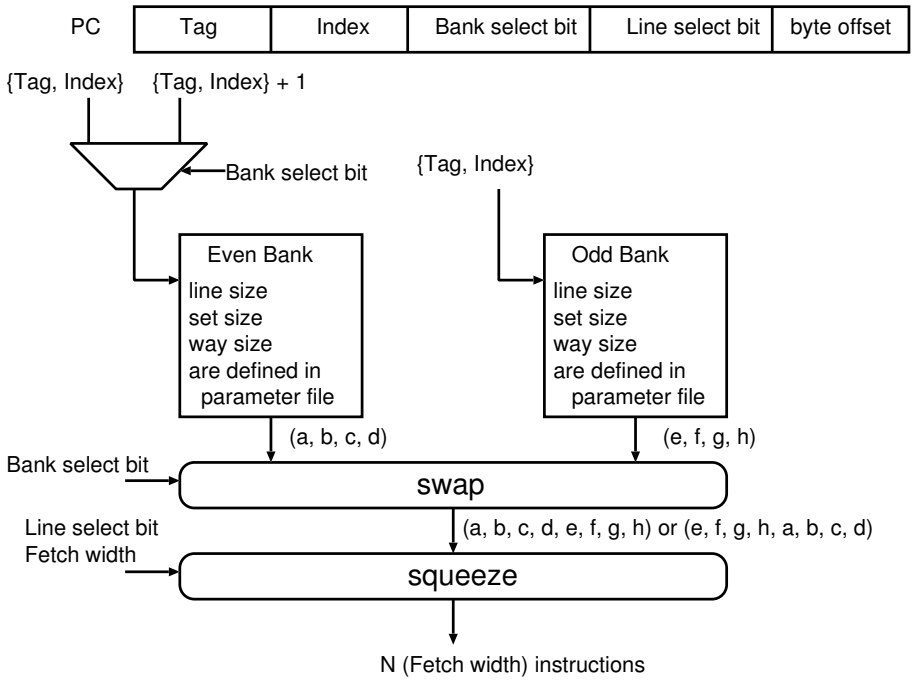

図 6.4: Implementation of interleaved L1 instruction cache

図 6.4 はインターリーブド L1 命令キャッシュの詳細を示している．L1 命令キャッシュは，整列化制約を無視したメモリ番地からの命令フェッチに対応するため，奇数と偶数の 2 バンクを持つバンクドメモリで構成されており，swap ユニットと squeeze ユニットを持っている．

各バンクはスーパーセット戦略を用いて実装され，ラインサイズ，セットサイズ，連想度が設定可能である．整列化制約を無視した命令フェッチに対し，正しい順番でキャッシュに格納するため，各バンクからの出力を

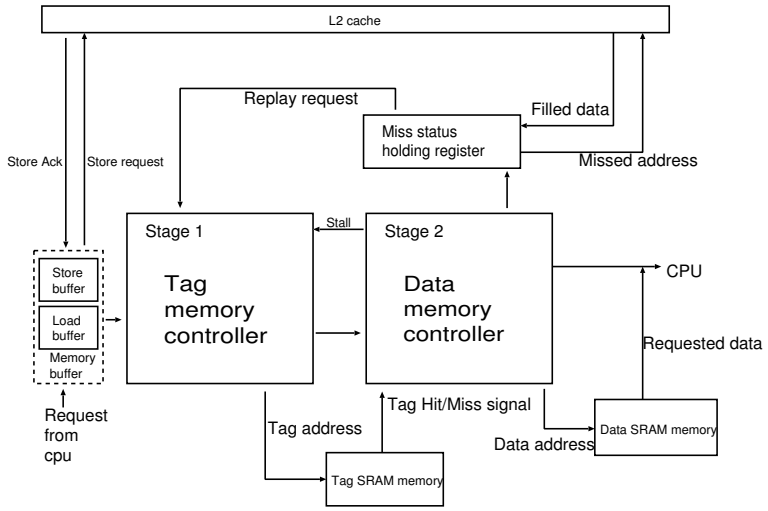

図 6.5: L1 Data Cache

bank select bit を用いてスワップしている Line select bit はフェッチする命令の先頭を決定し，その後パラメータファイルで指定したフェッチ幅まで命令を絞る．この実装を実現するため，最小ラインサイズの上限は最も近い 2 のべき乗に丸めている．(例えば，3 命令フェッチの場合でも，1 ラインに 4 命令存在する)．そのため，この squeeze 機構によるオーバーヘッドはない．

6.3 L1 データキャッシュの概要

図 6.5 は L1 データキャッシュ全体のブロック図を示している．L1 データキャッシュは 2 ステージのパイプラインで構成されており，毎サイクル miss status holding register (MSHR) に空きがあれば新規アクセスを受け

付ける．キャッシュストールにより，プロセッサが即ストールしないよう，ロード用，ストア用 2 種類のメモリバッファを持っている．メモリバッファの容量はパラメータファイルにより設定可能である．ステージ 1 では，先に処理されキャッシュミスを起こし，再実行される必要があるリクエストがなければ，次に実行するリクエストを処理し，タグメモリに対してリードリクエストを発行する．

LRU の更新もステージ 1 で実行される．Holding register はステージ 1 でストールが起こった際，リクエストを保持し続ける．もしキャッシュミスを起こしたリクエストがなければ，メモリバッファから新規リクエストがステージ 1 へ送信される．ステージ 2 ではタグを参照し，ヒットかミスを決定する．もしヒットであればリード又はライト信号がデータメモリへ送信される．ミスを起こした場合では，MSHR へミス情報を送信する．キャッシュコヒーレンシを解消するため，もし無効化されるべきエントリーがあれば，MSHR はステージ 1 へ無効化信号を送信する．それ以外では，MSHR は L2 キャッシュへミスリクエストを送信し，必要となるデータを受け取る．

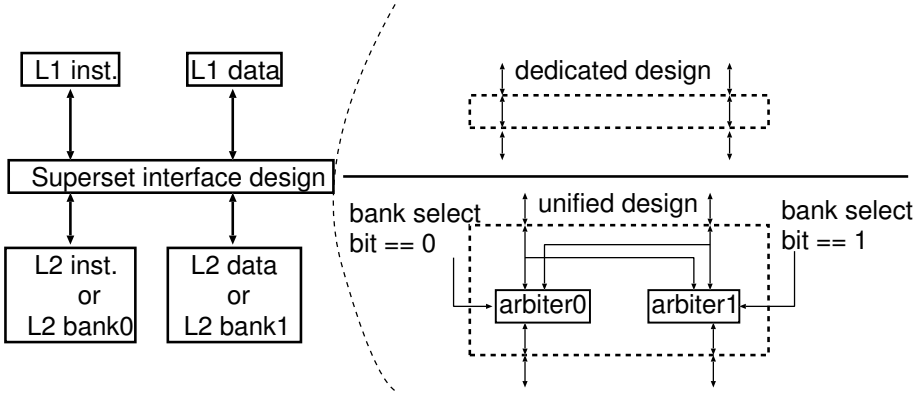

図 6.6: L2 cache design

6.4 L2 キャッシュの概要

L2 キャッシュは 2 バンク以上のバンクドメモリで構成されており，分散・共有 2 種類のタイプを生成することができる．図 6.6 はそれぞれ分散，共有 L2 キャッシュの例である．説明を簡略化するため，図の L2 キャッシュは 2 バンクで構成されているとする．分散 L2 キャッシュとして使用する場合，1 つのバンクを L2 命令キャッシュとして，他方を L2 データキャッシュとして使用する．逆に，共有 L2 キャッシュとして使用する場合，2 バンクのインターリーブド L2 キャッシュとして動作し，アドレス競合が起きないとき，L1 命令・データキャッシュ両方からの同時アクセスに対応する．説明を簡略化するため 2 バンクとしたが，アドレス競合を減らすため，実際には 3 バンク以上も設定可能である．この実装は，異なる設計

の L2 キャッシュを L1-L2 キャッシュ間のスーパーセットインターフェース構造を変化させるだけで生成を可能としている．分散 L2 キャッシュを使用するときは，L1 命令・データキャッシュは直接それぞれの L2 キャッシュへ接続する．一方，共有 L2 キャッシュとして使用する場合，アービタを 2 つ (2 バンクドメモリの場合)，インターフェースの中に追加する．このアービタにより，対応するバンクを決定し，リクエストが送信される．この構造はマクロによって定義されているため，意図しないハードウェアはインターフェースの中に残らない．さらに，インターフェースのみがスーパーセットコードで記述されているため，可読性も高い．しかし，バンクドメモリによって構成されているため，命令キャッシュとして使用される部分についても使用されないストア命令を実行するハードウェアが残ってしまうが，その回路は SRAM メモリを含むキャッシュメモリ全体の面積と比べ微小なことから無視できると考える．

この手法により，分散・共有 L2 キャッシュが僅かなオーバーヘッドで実装することができ，コード可読性も保たれる．

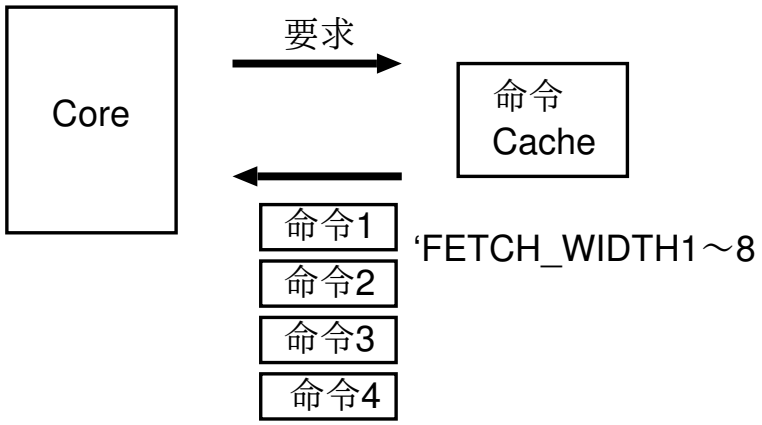

図 6.7: Fetch image of superscalar

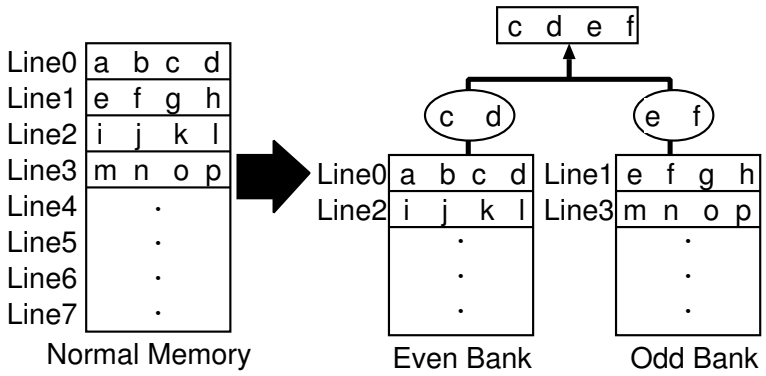

図 6.8: Interleaved memory

6.5 高性能プロセッサ向けの改良

6.5.1 インターリーブドキャッシュの詳細設計

FabScalar では性能向上の為に，任意の場所から連続した命令を 1 サイクルでフェッチする事を想定している．ここで，スーパースカラの命令フェッチの概念図を図 6.7 に示す．スーパースカラは並列に複数の命令を

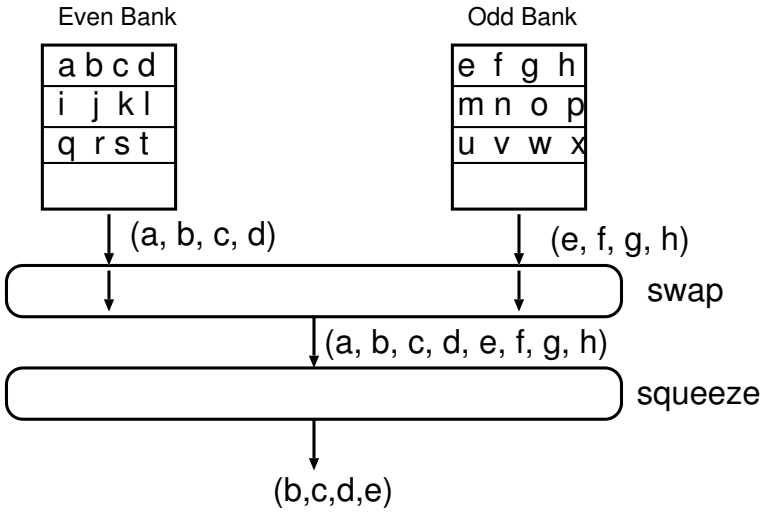

図 6.9: Interleaved memory

同時に実行する為，一度の命令キャッシュへのアクセスで複数の命令をフェッチしなければならない．しかしながら，通常のキャッシュを用いてこの機能を実装すると，ライン境界を跨ぐアクセスが発生した時に 1 サイクルで完了させる事ができない．このことを図 6.8 を用いて説明する．図 6.8 では，FabScalar は 4 命令フェッチのプロセッサとして構成されており，a から p はそれぞれ 1 つの命令を意味している．図 6.8 左の通常キャッシュでは 1 ラインにつき，4 つの命令が格納されており，ライン境界を跨がない場合 (例えば，連続した a，b，c，d の命令をフェッチする場合) には 1 サイクルで必要なデータを全て揃える事が可能である．しかしながら，通常のキャッシュでは 1 サイクルに 1 ラインのアクセスしかできない

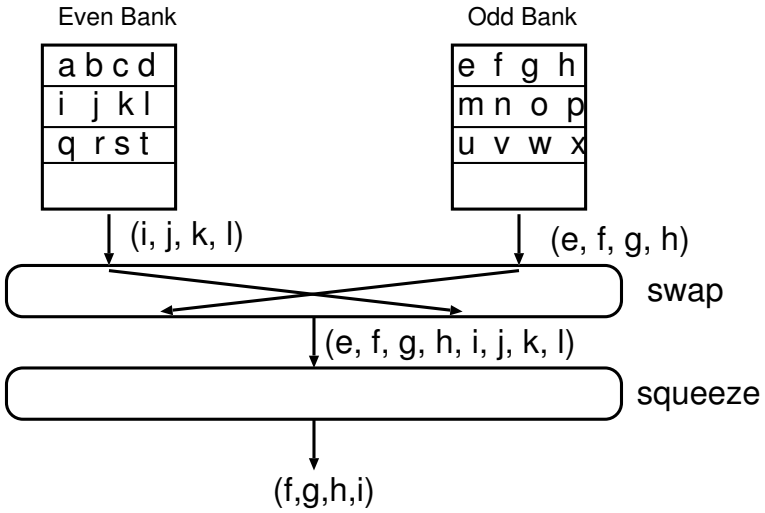

図 6.10: Interleaved memory

為，c，d，e，fのように，2つのラインを跨いで必要な命令が格納されている場合，データを揃える為に2サイクルを必要とする．そこで本研究では，キャッシュをインターリーブドメモリとして構成する事を提案する．このことで，任意の連続した命令を1サイクルでフェッチする事を可能としている．

インターリーブドメモリとは，メモリを複数のバンクに分割し，それぞれのバンクに対して同時にデータをアクセスする事でメモリアクセスを高速化する技術である．図6.9は，提案手法におけるL1命令キャッシュブロック図の一部を用いた，2バンクのインターリーブドメモリの例を示している．偶数バンクには偶数番地のラインが，奇数バンクには奇数番

地のラインがそれぞれ格納される．このようにデータを格納する事で c，d，e，f のような 4 命令フェッチアクセスが発生した場合に，偶数バンクから c, d が存在するラインを，奇数バンクから e，f が存在するラインを並列に読み出し，squeeze ユニットによって必要な部分を絞ることで任意の連続した命令を 1 サイクルで揃える事が可能となる．swap ユニットを用いる例として，必要な命令が f, g, h, i のように格納されている場合，同様に i が存在するラインと f, g, h が存在するラインを各バンクから並列に読み出し，swap ユニットによって正しい命令順に並び替えることで実現している．また，バンクドメモリを使用する事により，ポート数を増やす事なく並列にメモリアクセスを可能にする事でハードウェア規模の低減も実現している．

6.5.2　ノンブロッキングキャッシュ実装方法

多くの高性能プロセッサがロード・ストア命令を含むアウトオブオーダ実行をサポートしている [22]．ロード・ストア命令を含むアウトオブオーダ実行を処理するためには、スプリットバストランザクションとノンブロッキング手法がキャッシュシステムに求められる．FabCache では，CPU と L1 キャッシュ間のバスプロトコルとして AMBA4 を採用し，高性能プロセッサに対応するため，最大で 16 エントリの MSHR を持つノン

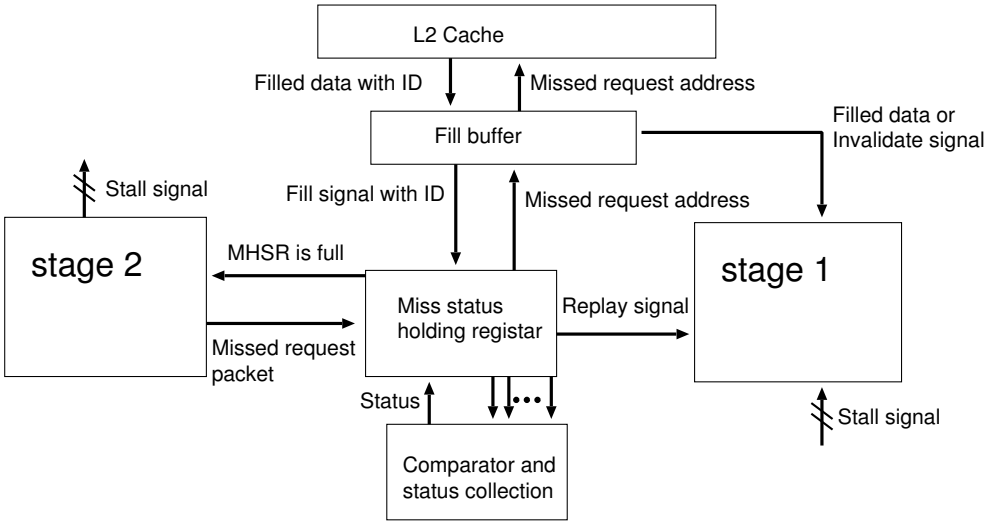

図 6.11: Miss status holding register

ブロッキングキャッシュを生成することができる．AMBA4 は 4 ビットのトランザクション ID を持っており，一度に 16 トランザクションまで扱えるため，MSHR のエントリを 16 に制限している．

しかし，ノンブロッキングキャッシュのコントローラーは，エントリ数に比例して面積と消費電力が増加してしまう．特に，コントローラーは高速実行が要求されるため，低リークトランジスタを使うことができない．つまり，余計に動的・静的電力を消費してしまう．一方，インオーダ実行スーパースカラやシングルパイプラインプロセッサのような組み込みシステムで使用される省電力プロセッサの場合，ノンブロッキングキャッシュは必要でない．つまり，このような組み込みシステムにノンブ

ロッキングキャッシュを実装するには，面積・電力の増加を招いてしまう．逆に，ブロッキングキャッシュを高性能プロセッサに実装してしまうと，アウトオブオーダ実行に対応できなくなり，高速実行が困難になってしまう．つまり，ヘテロジニアスマルチコアにおいて，省電力プロセッサと高性能プロセッサが混在する場合，最適なキャッシュシステムが異なってしまい，ブロッキング・ノンブロッキングキャッシュ両方を実装することは難しい．この問題を解決するために，スーパーセットを用いて可変MSHRエントリを持ったノンブロッキングキャッシュを実装した．図 6.11はMSHRの詳細を示している．第 6.3節にて述べたように，ステージ 2はミスリクエストを MSHRへ送信する．もし，パラメータファイルによって指定した MSHRエントリが一杯の場合，ステージ 2はストール信号をステージ 1へ送信する．その後，MSHRはミスリクエストに IDを付け，fill bufferへ送信する．Fill bufferは対応するラインのデータを受け取るため，ミスリクエストアドレスを L2キャッシュ，もしくはメインメモリへ送信する．もし，そのラインが無効化されるべきであれば，fill bufferは無効化信号をステージ 1へ送信する．それ以外では，fill bufferは対応するラインデータを受け取った後，MSHRへ ID付で送信し，同時に fill信号も送信する．fill信号を受け取った後，MSHRは IDを比較し，replay

信号と共に対応するミスリクエストを再度ステージ 1 へと送信する．

6.6 FabCache の移植性

FabCache は FabHetero プロジェクトの一部として実装しているが，バスプロトコルとして AMBA を採用しているため，通常のキャッシュシステムの研究にも使用することが出来る．AMBA プロトコルは，現在広く普及している System on Chip (SOC) における機能ブロックの接続と管理のための，オープンスタンダードなオンチップインターコネクト仕様である．FabCache によって生成されるキャッシュ間だけでなく，キャッシュープロセッサ間，メインメモリーキャッシュ間についても AMBA バスプロトコルを採用しているため，同様に AMBA プロトコルを採用している別のプロセッサに対して容易に接続することがきでる．また，FabCache は投機ロードや整列化制約を無視した命令フェッチ，ノンブロッキング機構など，今日における高性能スーパースカラプロセッサにおける要求を満たしているため，幅広いプロセッサを対象とすることができる．特に，柔軟なキャッシュシステム構成に加え，1 命令フェッチにも対応しているため，組み込みプロセッサ用キャッシュシステムも生成することができるため，高い移植性を持っていると考える．

表 7.2: EDA environment.

Phase	EDA tool
functional verification	Cadence NC-Verilog 09.20-S038
synthesis	Synopsys Design Compiler 2013.03-SP2
place & route	Synopsys IC Compiler G-2012.06
power estimation	Synopsys XA G-2012.06-SP2

7 評価

本章では提案手法によって自動生成されたキャッシュシステムが正しく動作し，また，手動設計により最適化されたキャッシュシステムと比べ遜色のない性能であることを示す．第 6.1 節にて述べたように，提案手法では RTL コード可読性を確保するためにスーパーセット戦略を用いており，意図しないハードウェアが生成され，結果として面積や消費電力の増加を招く可能性がある．そこで，手動設計により最適化したキャッシュシステムと，提案手法により生成したキャッシュシステムを比較し，オーバーヘッドを見積もった．評価環境として，使用するベンチマークは SPEC2000INT，EDA ツールは表 7.2 に示す．

7.1 性能評価

FabCache によって生成されたキャッシュが正しく動作することを確認するため，SPEC2000INT ベンチマークより 1 億命令実行した．図 7.12

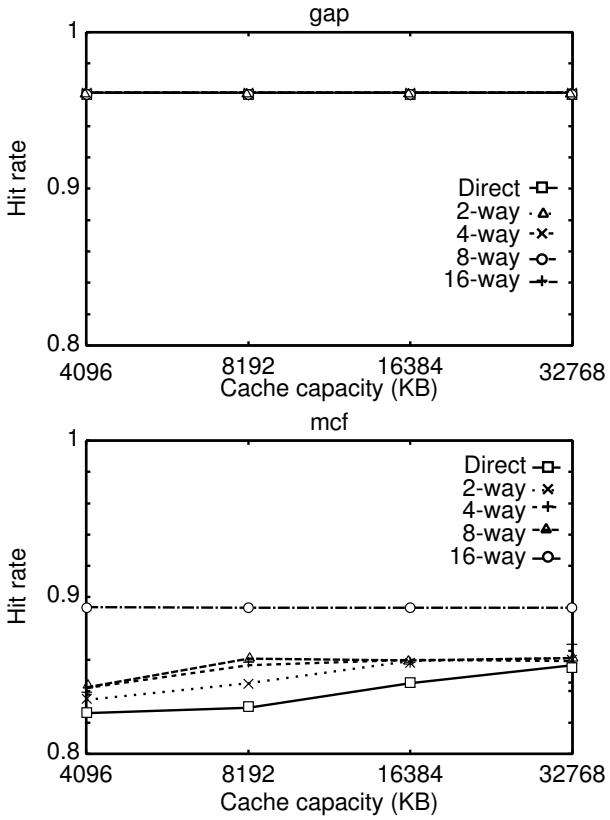

図 7.12: Cache hit rate

は，連想度をダイレクトマッピングから 16 ウェイセットアソシアティブまでの実行結果を示している．キャッシュ容量が増加するにつれ，ヒット率が上昇していることが確認出来たことから，FabCache によって自動生成されたキャッシュシステムが正しく動作していることが考えられる．

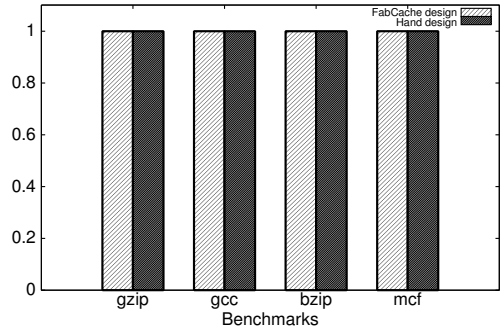

図 7.13: L1Icache Power Consumption.

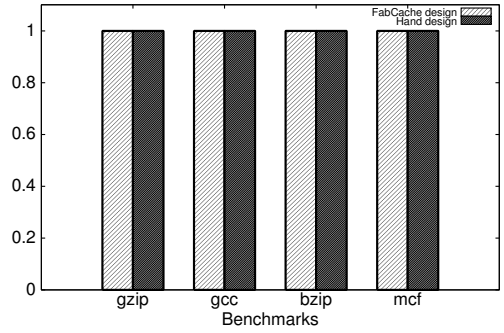

図 7.14: L1Dcache Power Consumption.

表 7.3: Delay.

Design	L1 instruction cache	L1 data cache
FabCache	2.39ns	2.45ns
Hand-tuned	2.27ns	2.32ns

7.2 電力評価

L1 命令キャッシュ及び L1 データキャッシュは RTL コード可読性のため，スーパーセットコードで実装されている．これにより，L1 命令キャッシュではダイレクトマッピングのとき，LRU とコントロール回路が，L1

データキャッシュではブロッキングキャッシュのとき MSHR が存在してしまう．これらの回路が電力消費を向上させる可能性があるため，それぞれ手動設計により最適化されたキャッシュシステムと消費電力を比較した．評価方法として，SPEC2000 INT ベンチマークより 5000 万命令実行し，EDA ツール Synopsys XA G-2012.06-SP2 を用いて電力を計測した．図 7.13, 7.14 はそれぞれ L1 命令・データキャッシュ電力消費を示しており，値は FabCache design によって正規化されている．図 7.13, 7.14 中の FabCache design はそれぞれ，FabCache によって自動生成されたスーパーセットコードによるオーバーヘッドを含むキャッシュシステムを示し，他方はオーバーヘッドを一切含まない，手動により最適化されたキャッシュシステムを示している．また，表 7.3 は FabCache によって生成されたキャッシュと手動設計によるキャッシュの遅延時間を比較したものである．評価結果によると，増加した電力は 0.1%以下，遅延時間の差は 0.1ns であるため，RTL コード可読性を保つためのスーパーセット戦略は妥当であると考える．

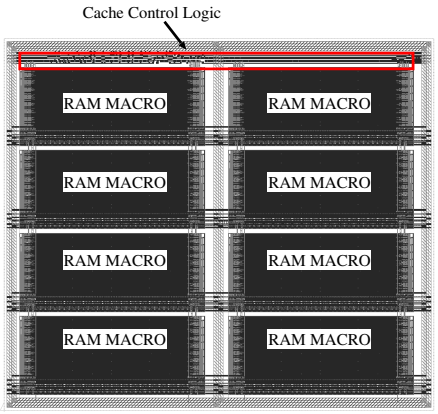

図 7.15: Chip image of L1 instruction cache.

7.3 面積評価

自動生成によるオーバーヘッドを見積もるため，物理チップレイアウトを作成し，面積評価を行った．図 7.15,7.16 は L1 命令・データキャッシュの物理チップレイアウトを示している．L1 命令キャッシュのパラメータは，容量 8KB，ラインサイズ 4，1 ウェイセットアソシアティブで構成されており，ダイレクトマッピイングにも関わらず LRU メモリ，及びコントローラが含まれている．L1 データキャッシュのパラメータは，容量・ラインサイズ・連想度は同様で，ブロッキングキャッシュではあるが 1 エントリの MSHR を含んでいる．FabCache によって生成されたキャッシュシステムは，Rohm 180nm，京都大学スタンダードセルライブラリ [23] を用いて論理合成を行った．図 7.15,7.16 中の Cache control logic はキャッシュ

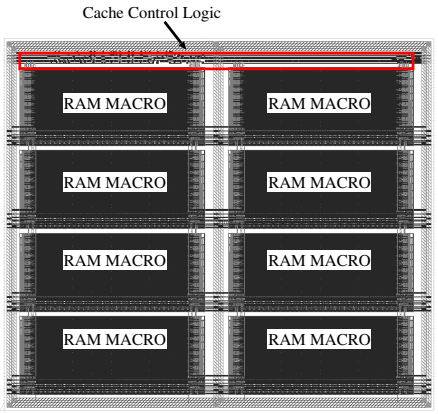

図 7.16: Chip image of L1 data cache.

制御部を示しており，RAM MACRO は SRAM メモリを示している．物理レイアウトの評価結果によると，キャッシュ制御部の面積は命令・データキャッシュそれぞれ 58,496.25μm^2・60,232.16μm^2，SRAM メモリを含むキャッシュシステム全体の面積は 1,668,016.628μm^2・1,669,752.538μm^2 となり，全体の面積に対する，自動生成によるオーバーヘッドを含むキャッシュ制御部の割合は 3.5%，3.6%となった．つまり，キャッシュ制御部の割合が非常に小さいことから，RTL コード可読性を確保するためのスーパセット戦略による意図しないハードウェアは無視することができるため妥当といえる．

8　結論

本論分では，ヘテロジニアスマルチコア対応のキャッシュシステム自動生成ツール，FabCache の詳細と評価について述べた．FabCache の詳細な設計より，組み込む向けプロセッサから高性能向けプロセッサの要求を満たす様々な高性能キャッシュシステムを自動生成できることが確認できた．さらに，手設計により最適化された L1 キャッシュと，FabCache によって生成された，自動生成によるオーバーヘッドを含む L1 キャッシュを比較したところ，面積では約 3.5%，遅延では 0.1ns，電力では 1%以下の増加に抑えられたことから，スーパーセット戦略により手設計と遜色ない品質のキャッシュシステムを少ないオーバーヘッドで実現できることが確認できた．今後の展望として，他の研究者や開発者を対象として FabCache を公開し，ヘテロジニアスマルチコアプロセッサとキャッシュシステム自体の研究を促進させたいと考える．

謝辞

本研究を行うにあたり，多数の助言を頂きました近藤利夫教授，深澤さん，並びにご指導を頂きました佐々木敬泰助教に深く感謝いたします．また，計算機アーキテクチャ研究室院生・学生のメンバーには常に刺激的な議論を頂き，精神的にも支えられました．また，本研究は日本学術振興会の科学研究費補助金，Synopsys 社 CAD ツールによる東京大学 VDEC，Rohm 社 VDEC，凸版印刷社の支援により実施されたことを並びに感謝します．

参考文献

[1] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *31st International Symposium on Computer Architecture (ISCA31)*, pp. 64-75, June 2004.

[2] H. H. Najaf-abadi, E. Rotenberg. Configurational Workload Characterization. *International Symposium on Performance Analysis of Systems and Software 2008 (ISPASS-2008)*, pp. 147-156, April 2008.

[3] P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. ARM WHITE PAPER: http://www.arm.com/ja/files/downloads/big.LITTLE Final.pdf.

[4] P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. ARM WHITE PAPER: http://www.arm.com/ja/files/downloads/ big.LITTLE_Final.pdf.

[5] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi and E. Roten-

berg. FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a Canonical Superscalar Template. *38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38)*, pp. 11-22, June 2011.

Rationale for a 3D Heterogeneous Multi-core Processor. *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD-31)*, pp. 154-168, Oct. 2013.

[6] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. FabScalar: Automating Superscalar Core Design. *Micro, IEEE (Volume:32 , Issue: 3)*, pp. 48-59, June 2012.

[7] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan and D. M. Tullsen. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. Int'l Symposium on Microarchitecture, Dec. 2003.

[8] H. H. Najaf-abadi, N. K. Choudhary and E. Rotenberg. Core-Selectability in Chip Multiprocessors. 18th Int'l Conference on Parallel Architectures and Compilation Techniques, Sep. 2009.

[9] 中林智之, 佐々木敬泰, Eric Rotenberg, 大野和彦, 近藤利夫, FabScalar の Alpha 21264 命令セット対応とマルチプロセッサ環境フレームワークの構築, SACSIS2012.

[10] E. Rotenberg, B. H. Dwiel, E. Forbes, Z. Zhang, R. Widialaksono, R. Basu Roy Chowdhury, N. Tshibangu, S. Lipa, W. R. Davis, and P. D. Franzon.

[11] N. K. Choudhary, B. H. Dwiel, E. Rotenberg. A physical design study of fabscalar-generated superscalar cores. *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on* , pp. 165-170, Oct. 2012.

[12] T. Nakabayashi, T. Sasaki, E. Rotenberg, K. Ohno and T. Kondo. Research for Transporting Alpha ISA and Adopting Multi-processor to FabScalar. *Symposium on Advanced Computing Systems and Infrastructures 2012 (SACSIS2012)*, pp. 374-381, May 2012. (in Japanese)

[13] T. Okamoto, T. Nakabayashi, T. Sasaki, T. Kondo. FabCache: Cache Design Automation for Heterogeneous Multi-core Processors.

Proceedins of the 1st International Symposium on Computing and Networking, pp.602-606, Dec. 2013.

[14] 瀬戸　勇介，佐々木　敬泰，大野　和彦，近藤　利夫, ヘテロジニアスマルチプロセッサ環境を対象としたAMBA バスフレームワークの設計と評価,SWOPP2012.

[15] Y. Seto, T. Nakabayashi, T. Sasaki, and T. Kondo. FabBus: A Bus Framework for Heterogeneous Multi-core processor. *28th International Technical Conferench on Circuits/Systems, Computers and Communications (ITC-CSCC2013)*, pp. 254-257, July 2013.

[16] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi and E. Rotenberg. FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a Canonical Superscalar Template. Proceeding of the 38th IEEE/ACM Int'l Symposium on Computer Architecture (ISCA-38), pp. 11-22, June 2011.

[17] B. de Abreu Silva, L.A. Cuminato and V. Bonato. Reducing the overall cache miss rate using different cache sizes for Heterogeneous

Multi-core Processors. *Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1-6, Dec. 2012.

[18] P. Yiannacouras and J. Rose. A Parameterized Automatic Cache Generator for FPGAs *Field-Programmable Technology (FPT),* pp. 324-327, Dec. 2003.

[19] Leon 4 and GRLIB. http://www.gaisler.com.

[20] Thomas D. Tessier, Designing, Verifying and Building an Advanced L2 Cache Sub-System using SystemC. *ISCUG*, April 2012.

[21] Akgul, B.E.S., Mooney, V.J,PARLAK: Parametrized Lock Cache Generator *Design, Automation and Test in Europe Conference and Exhibition*, pp.1138–1139, April 2003.

[22] D. Kroft., Lockup-free instruction fetch/prefetch cache organization. *International Symposium on Computer Architecture Proceedings of the 8th annual symposium on Computer Architecture*, pp. 81–87, May 1981.

[23] H. Onodera, A. Hirata, A. Kitamura, K. Kobayashi, and K. Tamaru, P2Lib:Process Portable Library and Its Generation System, *Journal*

of Information Processing, vol.40, no. 4, pp. 1660–1669, April, 1999, (In Japanese).

A　プログラムリスト

B　評価用データ