卒業論文

題目

ヘテロジニアスマルチコアプロセッサ環境を対象としたキャッシュシステム自動生成ツールの開発

指導教員

佐々木　敬泰　助教

2013年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

岡本 昂樹 (409812)

内容梗概

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，構成の異なるスーパースカラコアを複数個用いた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．複数のスーパースカラコアをプログラムの特徴に合わせて使い分けることは，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，各コアの構成やコア数，キャッシュコヒーレンシのプロトコル，バスシステムなどの組合せの膨大さや，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．そこで当研究室では，この問題を解決する為に FabHetero を提案している．FabHetero は，任意のヘテロジニアスマルチコアプロセッサを自動生成する事ができる．本研究では，FabHetero のフレームワーク上の様々な構成のキャッシュシステムを自動生成するツールである FabCache の提案と実装を行う．実装した FabCache を用いて生成されたキャッシュが正しく動作している事を示す為，SPEC2000INT から 6 つのベンチマークをそれぞれ 1000 万命令ずつ実行したところ，キャッシュの容量を増加させるにつれ IPC も上昇している事から，正しく実装できている事が確認できた．また，FabCache によって生成された回路が手設計した回路と遜色がない事を示す為，それぞれの面積を比較したところ，ハードウェア規模の増加は 0.076%程度に抑えられている事から実装の妥当性が確認できた．

Abstract

Single-ISA heterogeneous multi-core processors are increasing importance in the processor architecture. However, designing a single-ISA heterogeneous multi-core requires design and verification effort which is multipliedby the number of different cores. processor. Therefore, FabHetero is proposed to solve this problem. FabHetero generates diverse heterogeneous multi-core processors automatically using FabScalar, FabCache, and FabBus. This paper proposes and implements FabCache which can automatically generate diverse cache systems of FabHetero. To indicate that the cache generated by FabCache works correctly, we execute 10 million instructions on SPEC2000INT benchmarks.

we compared the area of auto-generated cache by FabCache with the area of hand designed cache. According to the evaluation results, caches generated by FabCache work correctly and occupy almost the same area compared with hand-designed cache.

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．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[1] が提案されている．FabScalar は，構成の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．この問題を解決するために当研究室は、様々な構成のヘテロジニアスマルチコアを自動生成するツールセットとして FabHetero[2] を提案している．FabHetero は，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，現状

では高速なメモリアクセスを実現する為のキャッシュシステムを自動生成する部分が未実装である．そこで本研究では，FabHetero におけるキャッシュシステム自動生成ツールである FabCache を提案・実装する．以降，本論文は次のように構成する．まず，次章でスーパースカラについて，3 章でヘテロジニアスマルチコアについて説明する．4 章で関連研究である FabScalar について，5 章で当研究室で提案している FabHetero について説明する．6 章で今回提案・実装した FabCache について，7 章で実装と評価について説明する．

2　ヘテロジニアスマルチコア

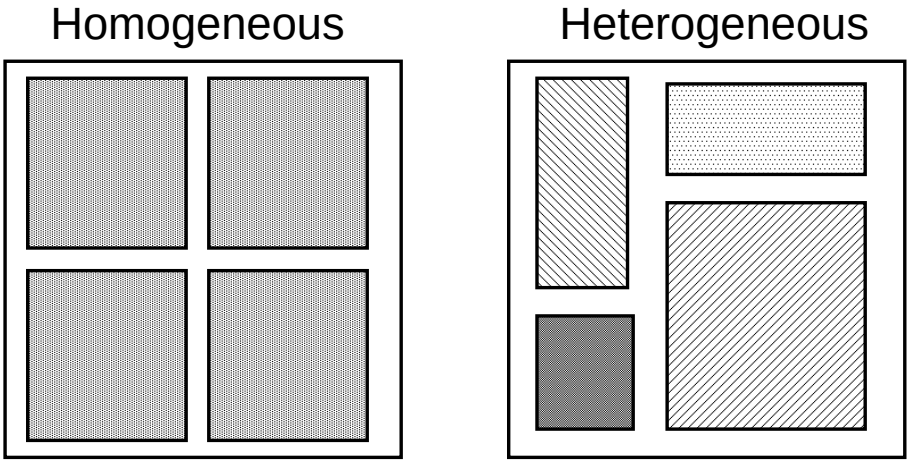

図 2.1: ホモジニアスマルチコアとヘテロジニアスマルチコア

現在，同じアーキテクチャの CPU コアを 1 チップに複数搭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，電力効率の低下の一因となっている．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，各コアの構成やコ

ア数，キャッシュコヒーレンンシのプロトコル，バスシステムなどの組合せの膨大さや設計・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．

3　関連研究

3.1　FabScalar

組合せの膨大さから，ヘテロジニアスマルチコアを設計・検証する時間が膨大になるという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を自動生成するツールセットとして FabScalar[2] が提案されている．FabScalar はフェッチ幅，パイプライン段数，各ユニット数等のパラメータを与える事で様々な構成のスーパースカラプロセッサを自動生成するツールである．FabScalar を使用する事により，設計者は構成の異なるスーパースカラコアを短時間で容易に設計する事が可能となり，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．そこで当研究室では，この問題を解決する為に FabHetero を提案している．

3.1.1 スーパースカラ

パイプラインとは，命令の実行やメモリアクセスなどオーバーラップ可能な部分を並列に処理し性能を向上させる為の手法である．スーパースカラとは，パイプラインを並列に複数並べ，複数の命令を並列に実行する機構である．スーパースカラとスーパースカラでないプロセッサの概念を図3.2に示す．図3.2のように，複数存在する命令実行パイプラインに同時に送り込む事で並列実行を実現している．図3.2中のF，D，E，M，Wはそれぞれ命令フェッチ，命令デコード，実行，メモリアクセス，ライトバックを表し，横軸の数字はクロックを表している．スーパースカラでは，並列に実行でいる命令数をウェイと呼ぶ．ここで図3.2はシングルパイプラインと2ウェイスーパースカラとの，同じクロック数で実行できる命令数を表している．シングルパイプラインの場合，7サイクルで命令1から3の，3つの命令が実行完了しているのに比べ，スーパースカラの場合，同じ7サイクルでシングルパイプラインの倍である命令1から6の，6つの命令が実行完了している．しかし，図3.2中で命令2の実行結果を用いて命令3を実行するなど，命令間に依存がある場合が多々あるので単純にウェイ数を増加させる事で性能が向上するとは限らない．

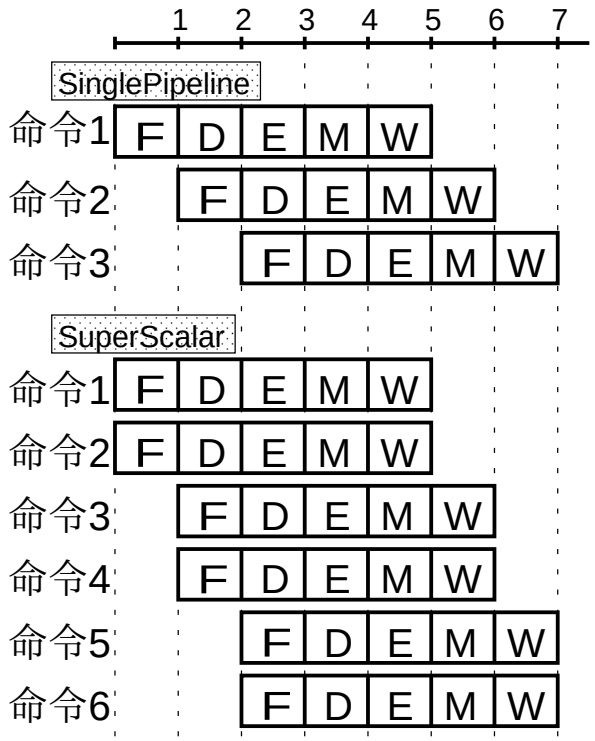

図 3.2: シングルパイプラインと 2 ウェイスーパースカラ

4 FabHetero

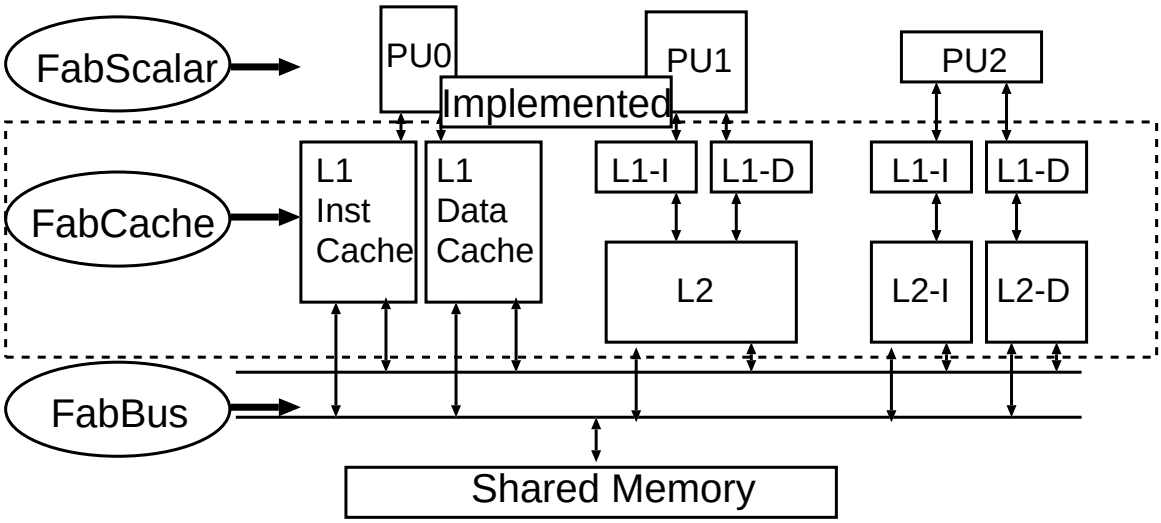

図 4.3: FabHetero

FabHetero によって生成されるヘテロジニアスマルチコアプロセッサの例を図 4.3 に示す．FabHetero は様々な構成のスーパースカラコアを部分を生成する FabScalar，それぞれのコアに付随するキャッシュシステムを自動生成する FabCache，それらのモジュールを接続する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プロセッサを自動生成するフレームワークである．設計者は FabScalar，FabCache，FabBus のパラメータファイルに対し，任意のパラメータを入力する事で目的とするヘテロジニアスマルチコアを生成する事ができる．本研究で

は，FabHetero のフレームワーク上の様々な構成のキャッシュシステムを自動生成するツールである FabCache の提案と実装を行う．

4.1 FabBus

FabBus はヘテロジニアスマルチコアプロセッサ環境を対象とした柔軟なシステムバスフレームワークを自動生成するツールである．組み込み分野ではコストやハードウェア資源等の制約によりキャッシュシステムの実装方法やコヒーレンシアルゴリズムを決定することが難しい．また，それに加えて本研究で対象とするヘテロジニアスマルチコアにおいては，搭載するコアの種類により性能が大きく変化するために性能の予測が行い難く，一方で，すべての組み合わせを手で設計し評価することも困難である．そこで，基本的なシステムバスフレームワークの設計に加えて，バスやユニットの追加によるキャッシュコヒーレンシ制御機構の実装も行っている．

5 FabCache

図 4.3 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 4.3 左の様に独立した L1 キャッシュのみを持つ構成や，図 4.3 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 4.3 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．

本研究では主に FabCache の L1 命令キャッシュの実装を行う．本研究で実装する L1 命令キャッシュの仕様を表 5.1 に示す．Fetch Width は 1 サイクルで何命令同時にフェッチするかを示し，Cache Size はセット数，Way Size は連想度，Line Size は 1 ラインのワード数，Word Size between L1 and L2 は，L1 キャッシュと L2 キャッシュ間のデータ受渡しサイズを示し，SRAM Access Latency は SRAM にアクセスするのに何サイクルかかる

かを示すパラメータである．それぞれのパラメータの指定できる範囲は表 5.1 の通りである．また，従来の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．

表 5.1: L1 キャッシュのパラメータ仕様

PARAMETER	RANGE
Fetch Width	1〜8
Cache Size	2^n エントリ
Way Size	1〜32
Line Size	1〜8
Word Size between L1 and L2	1〜LineSize
SRAM Access Latency	depend on SRAM module

5.1 インターリーブドキャッシュ

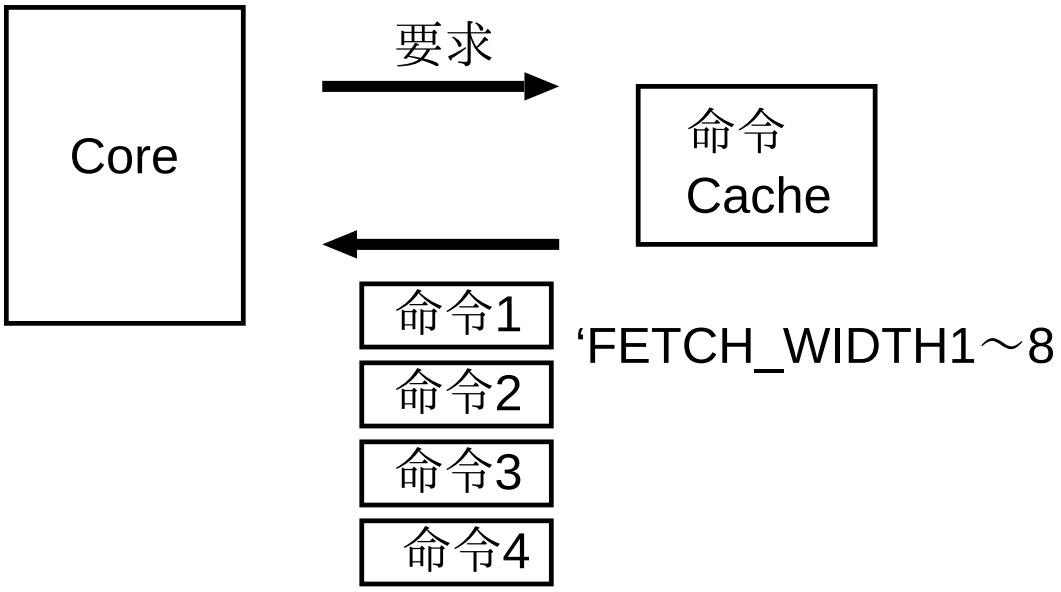

図 5.4: スーパースカラ命令フェッチ概念図

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 5.4 に示す．スーパースカラは並列に複数の命令を同時に実行する為，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てこなければならない．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 5.5 を用いて説明する．図 5.5 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 5.5 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 5.5 は，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 のようなアクセスが発生した場合に，偶数バンクから c，d を，奇数バンクから e，f を読み出す事で，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．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．

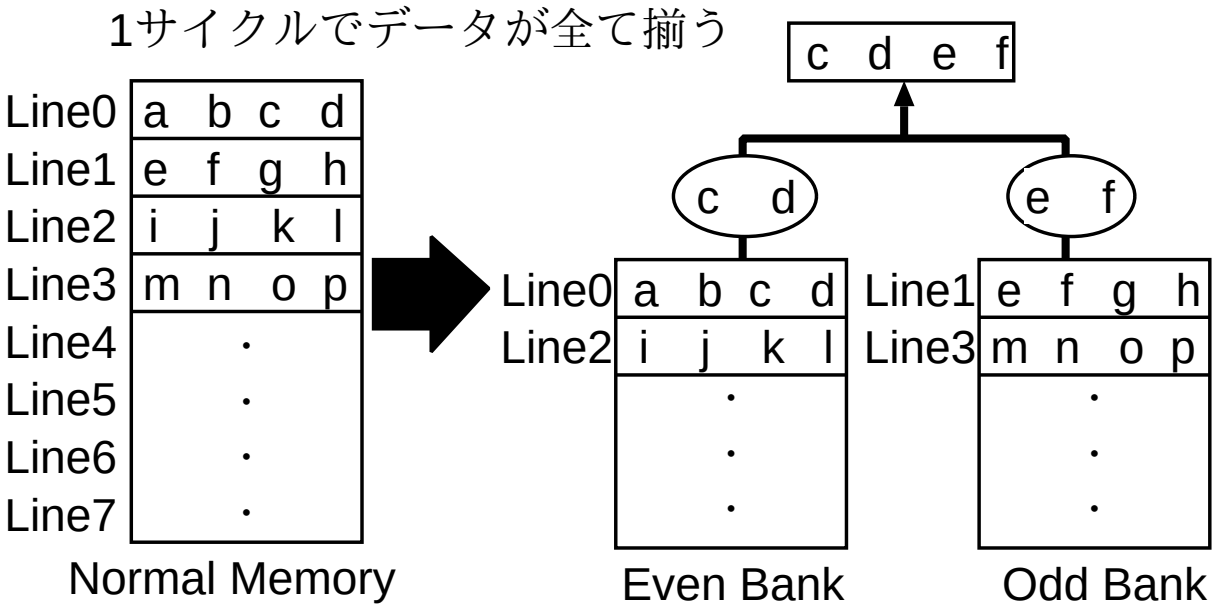

図 5.5: インターリーブドメモリ

5.2 実装

本研究では，L1 命令キャッシュの容量を任意のサイズで生成出来るように System Verilog で実装した．その他のパラメータについては現在固定値となっており，変更ができない．また，2 のべき乗でない命令フェッチ幅にも対応できるように，例えば命令フェッチ幅 3 のときは 4，命令フェッチ幅 5〜7 のときは 8 で処理している．このことを図 6.6 を用いて説明する．図 6.6 におけるプロセッサの構成等は，命令フェッチ幅を除き，図 5.5 と同じとする．命令フェッチ幅が 3 のとき d，e，f のようなアクセスが発

生した場合，命令フェッチ幅 4 のときと同様に偶数バンクから c，d を，奇数バンクから e，f を読み出す．ここで，必要とするデータ d，e，f のみを取り出す事で 2 のべき乗でない命令フェッチを実現している．

6　評価

FabCache が生成したキャッシュが正しく動作している事を検証する為に，フェッチ幅，キャッシュサイズを変更して SPEC2000INT から bzip, gzip，gap，mcf，parser，voretx の 6 つのベンチマークを実行した．命令フェッチ幅を 1，2，4，8，と変更したときの，1 サイクルあたりの命令コミット数（IPC) をそれぞれ図 6.7〜図 6.10 に示す．図 6.7〜図 6.10 により，キャッシュサイズを増加させるにつれ，IPC が増加している事がわかる．また，キャッシュサイズが 128KB の時の IPC は 100%ヒットする理想的なキャッシュの IPC を用いており，FabCache が生成したキャッシュの IPC が飽和した時の性能に遜色がないことから，正しく動作するキャッシュが実装できている事が示された．また，FabCache によって生成された回路が手動設計した回路と遜色がない事を示す為，FabCache で生成したキャッシュと，手動で設計したキャッシュの面積を比較する．キャッシュ面積を比較した評価結果を表 6.2 に示す．この結果から，ハードウェア規模の増加が 0．076%程度に抑えられている事から実装の妥当性が確認できた．また，遅延は FabCache によって生成された回路の方が小くなっているが，その差は 1 ロジック程度のものなので論理合成のツールによる誤差と考えられる．

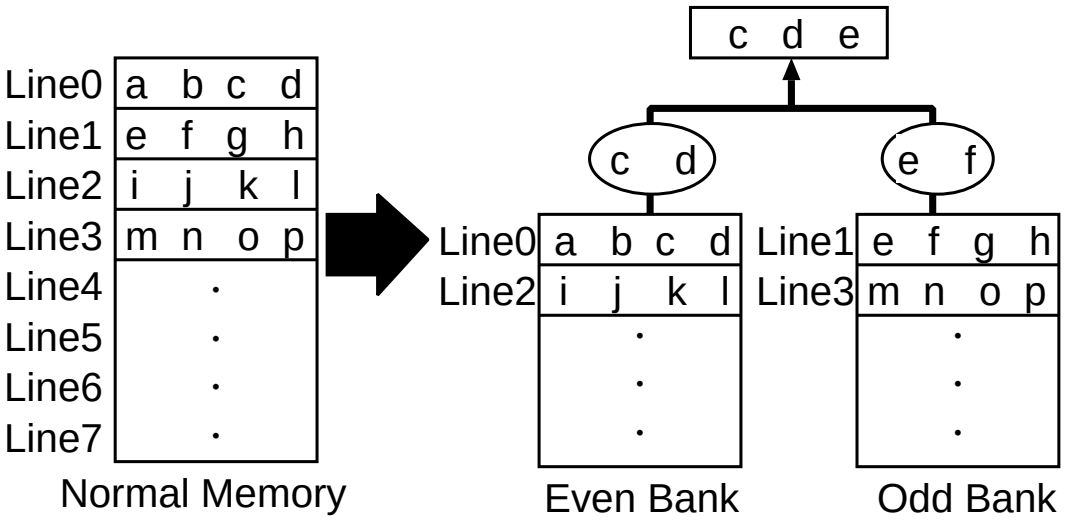

図 6.6: 2 のべき乗でない命令フェッチ図

表 6.2: ハードウェア規模・遅延の比較

生成方法	面積	遅延
FabCache	819209 μ m^2	2.27ns
手設計	818590 μ m^2	2.43ns

7 おわりに

本研究では，ヘテロジニアスマルチコアプロセッサ環境を対象とした柔軟なキャッシュシステム自動生成ツール FabCache の実装を行った．また，ベンチマークの実行結果から，生成された回路は正しく動作し，論理合成による回路規模の比較より，手設計した回路と比較してハードウェア規模の増加が 0．076%程度に抑えられている事から妥当な規模の回路が生成されている事が確認できた．今後の研究として，表 5.1 の未実装の

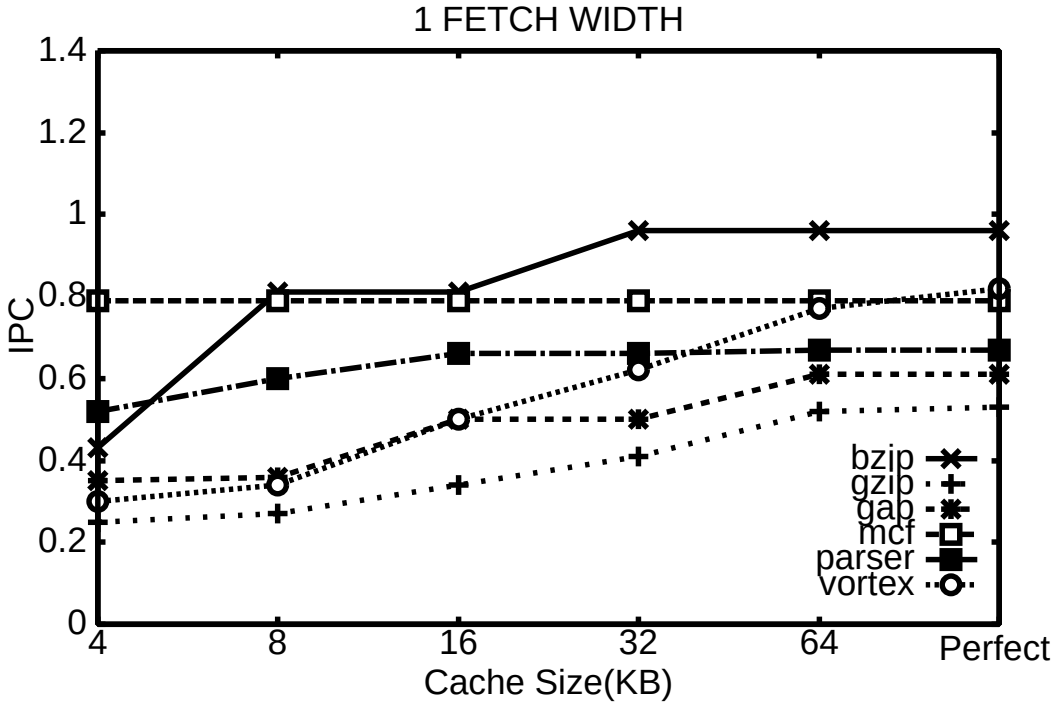

図 6.7: Instruction Per Cycle1

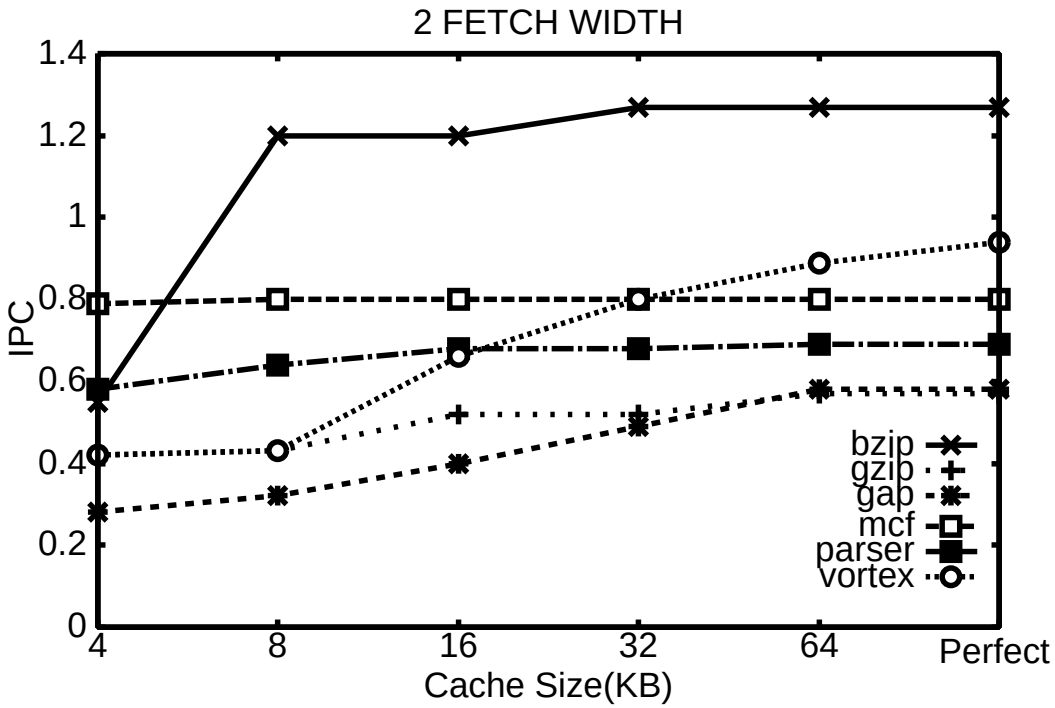

図 6.8: Instruction Per Cycle2

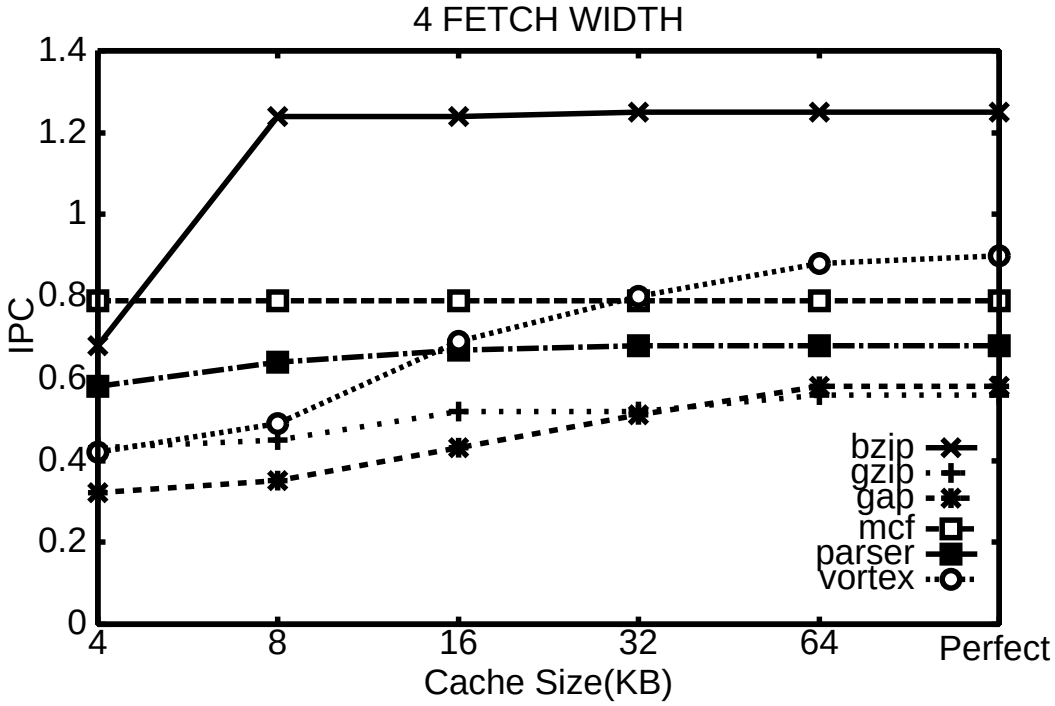

図 6.9: Instruction Per Cycle4

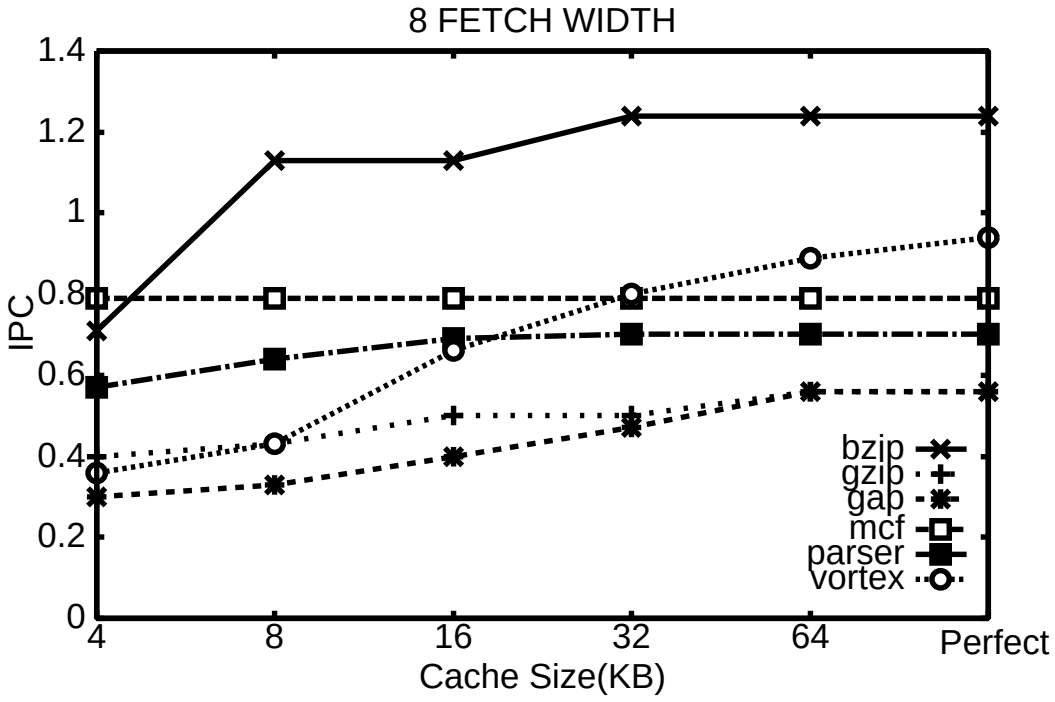

図 6.10: Instruction Per Cycle8

機能を実装および，L1 データキャッシュ，L2 キャッシュの実装を行い，より柔軟なキャッシュを自動生成できるシステムを構築していく．

謝辞

本研究を進めるにあたり，ご指導を頂いた指導教員の佐々木敬泰助教，並びに近藤利夫教授，大野和彦講師に感謝致します．また、日常の議論を通じて多くの知識や示唆を頂いた計算機アーキテクチャ研究室の皆様に感謝します．最後に，事務等を担当して頂いた田中さんに感謝します．

参考文献

[1] 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.

[2] 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.

[3] 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.

A　プログラムリスト

B　評価用データ

	PARAMETER	RANGE
FabCache	SIZE_ICACHE	1024〜32768
	SIZE_ICACHE_WAY	1
	SIZE_DCACHE	8192
	SIZE_DCACHE_WAY	1
	L2LATENCY	1
FabScalar	FETCH_WIDTH	1〜8

表 2.3: 評価時のパラメータ