計算機アーキテクチャ　150－13
（２００２．　１１．　２８）

オブジェクトキャッシュを用いたJVM命令互換プロセッサの設計

近　千秋 †　　清水尚彦 ††
東海大学 大学院 工学研究科 †　東海大学電子情報学部コミュニケーション工学科 ††
259－1292 神奈川県平塚市北金目 1117
E-mail:{1aepm024, nshimizu}@keyaki.cc.u-tokai.ac.jp

概要　Javaプロセッサを使用してJava実行環境を実現する際，オブジェクト参照の解決を伴う命令の実装方法が問題となってくる．我々は，これらの命令の実行を最適化する手法としてオブジェクトキャッシュを提案している．今回，オブジェクトキャッシュを実装したJava仮想マシンで各種ベンチマークプログラムを実行し性能評価を行った．この結果より，オブジェクトキャッシュを実装したJava仮想マシンの特性を得る事が出来た．オブジェクトキャッシュを実装することで，クラスの再利用性の高いプログラム，小型なプログラムでは性能向上が見込めることが分かった．性能評価の結果とオブジェクトキャッシュによる命令処理動作について説明する．

キーワード: Java, Java仮想マシン，オブジェクト指向，オブジェクト参照

The Design of a Java Processor with the Object Cache

Chiaki Kon †　Naohiko Shimizu ††
Graduated School of Engineering, Univ. TOKAI †
School of Information Technology and Electronics, Univ. TOKAI ††
E-mail:{1aepm024, nshimizu}@keyaki.cc.u-tokai.ac.jp

Abstract　We have continued development and research of our original Java processor, named TRAJA, since 1995. We implement the "Object Cache" system to the Kaffe virtual machine and measured the performance of the virtual machine. In this paper, we described the design of the Java processor with the "Object Cache".

Keyword: Java, Java Virtual Machine, Object Oriented, Object Reference

1　はじめに

組み込み機器分野においてJava実行環境実現のために，Javaプロセッサを使用する方法が注目されている．快適なJavaの実行環境を実現するためには，CPUの能力，メモリ容量が必要になってくるが，組み込み機器分野においては，様々な制約上，十分なハードウェア能力を得ることが難しい．そこでJava仮想マシン命令の実行エンジンとしてJavaプロセッサを使用することで，Java実行環境を著しく改善できる可能性がある．1995年以来Javaプロセッサ TRAJAの開発，研究を続けてきた[9]．TRAJAは，Java仮想マシン命令をハードウェアにより直接実行するJavaプロセッサである．我々は，Javaプロセッサのオブジェクト操作命令の実行を補助するオブジェクトキャッシュ[11][12]と，配列アクセス命令の実行を補助するアレイキャッシュ[10]を提案している．これらの機構の実装により，オブジェクト操作命令の実行処理の高速化，配列データアクセスの高速化，インスタンスオブジェクトの小型化，バイトコードの静的データ化等が可能になると考えている．我々は今回，オブジェクトキャッシュをオープンソースのJava仮想マシンkaffe[8]のインタプリエンジンに実装し評価を行った．本稿では，オブジェクト操作命令と配列命令について説明し，オブジェクトキャッシュによる実行処理に関して説明する．また，ベンチマークプログラムの実行による性能評価に関して報告する．

2　オブジェクト操作命令

Java仮想マシン命令中には，Fig.1に示されるような，実行時にオブジェクト操作を伴う命令が存在する．これらの命令で実行される処理は，メソッドのINVOKE，フィールドへのGET，PUT操作，オブジェクトの検査などである．オブジェクト操作命令は，一般的に長く複雑な実行処理を必要とする．これらの命令について説明する．

2.1　命令動作

Fig.2より，オブジェクト操作命令の命令動作を説明する．ここでは，例として`getfield`命令の命令動

```
anewarray
checkcast
getfield
getstatic
instanceof
invokeinterface
invokespecial
invokestatic
invokevirtial
ldc
ldc_w
ldc2_w
multianewarray
new
putfield
putstatic
```

Fig. 1: オブジェクト操作命令

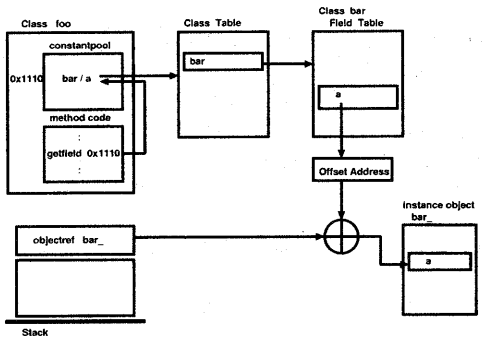

Fig. 2: getfield 命令の実行処理

作について説明する．命令列から getfield がフェッチされると，Java 仮想マシンは直渡しオペランドとして2バイトのコンスタントプールインデックスを得る．getfield 命令の場合，コンスタンとプール中の CONSTANT_Fieldref エントリーを指し，更に CONSTANT_Fieldref が CONSTANT_Class エントリーと CONSTANT_NameAndType エントリーを指す．CONSTANT_Class と CONSTANT_NameAndType から目的のフィールドを指す Utf8 文字列を得，この文字列を使用してまずクラステーブルよりクラスの検索を行い，該当するクラスのフィールドテーブルより目的とするフィールドの情報を得る．Fig.2 の場合，フィールド情報よりフィールドエントリーのオブジェクト先頭からのオフセットアドレスを取得し，スタック上のオブジェクト参照に加算を行い，目的とするフィールドのアドレスを得ている．ここで問題となるのが，フィールドのアドレスを得るまでの処

```
iaload
laload
faload
daload
aaload
baload
caload
saload
iastore
lastore
fastore
dastore
aastore
bastore
castore
sastore
arraylength
```

Fig. 3: 配列アクセス命令

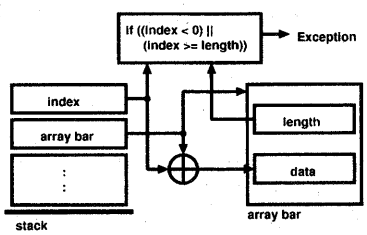

Fig. 4: 配列ロード命令の実行処理

理の過程である．前述したようにクラス検索，及びフィールド検索で使用される検索キーは Utf8 文字列であり，各々のテーブルの中から文字列比較を行いながら目的とするフィールドを検索しなくてはならない．この処理は，他のオブジェクト操作命令に於いても実行しなければならず，これら命令の実行処理が複雑化する主な原因となっている．

2.2 解決方法

このように，コンスタンとプール中の Utf8 文字列より実際のオブジェクトのアドレスを得る事を，参照の解決と呼ぶ．ここで，参照の解決ルーチンをオブジェクト操作命令が実行される毎に行うとすれば，Java プログラム実行上の大きなオーバーヘッドとなってくる．そこで，オブジェクト参照の解決によるオーバーヘッドを避けるため，Java 仮想マシンには一度解決されたオブジェクト参照を記憶する機能が必要となる．Sun Microsystems[1] の Java 仮想マシンではオブジェクト参照が解決したオブジェクト操作命令

を，*_quick*修飾のついた別の命令に書き換えることでオブジェクトアクセスの最適化を行い，kaffe[8]ではアクセスフラグに解決ビットを定義することで参照の解決の有無を判別する．オブジェクトキャッシュも同様に，オブジェクト参照の解決の有無を記憶する．また，命令動作に必要なデータも保持することで，以降の命令動作を高速に行うことが可能である．オブジェクトキャッシュの動作については後述する．

3 配列アクセス命令

Java仮想マシンの配列アクセス命令は，データ保護の為のチェックを行なう必要があり，違反があった場合には例外を発生させる．配列アクセス命令はFig.3に示される．配列アクセス命令について説明する．

3.1 命令動作

Fig.4に配列ロード命令の動作を示す．Java仮想マシンは，配列データへのアクセスを行う際データ保護の為に，スタック上の`index`が配列内に収まっているかをチェックする必要があり，配列外へのアクセスであれば，`ArrayIndexOutOfBounds`の例外を発生させる必要がある．配列長は配列オブジェクトが保持しているため，数回のメモリアクセスが必要となる．また，`aastore`命令では，加えて配列の型のチェックも行なう必要がある．

3.2 アレイキャッシュによる高速化

配列アクセス命令においては，例外のチェックが命令実行のオーバーヘッドとなり得る．そこで，例外チェックに必要なデータを保持するアレイキャッシュを用意することで，配列アクセスの高速化を計る．アレイキャッシュは配列長とデータ型を保持し，スタック上の`arrayref`によって探索される．アレイキャッシュを使用することで，メモリアクセス回数を低減することが可能であり，命令動作の高速化が期待できる．

4 オブジェクトキャッシュ

オブジェクト操作命令の実行時におけるオブジェクト参照解決ルーチンのオーバーヘッドと，それに伴う解決済みの参照を記憶する事の必要性，また配列アクセスに伴う例外チェックの為のオーバーヘッドについて述べてきた．我々は，これらの問題を解決するためにオブジェクトキャッシュを使用する事を提案する．オブジェクトキャッシュは，Javaプロセッサのオブジェクト操作命令の実行をサポートする．また，アレイキャッシュもこのオブジェクトキャッシュの枠組みに取り入れることで，ハードウェア量の削減を計る．オブジェクトキャシュの構造と，オブジェクトキャッシュ使用時の命令動作について説明する．

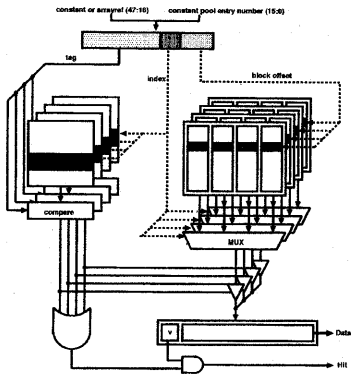

Fig. 5: オブジェクトキャッシュの構成

4.1 構造

オブジェクトキャッシュの構成をFig5に示す．オブジェクトキャッシュは通常のキャッシュの様に，メモリ値のコピーは格納する記憶域ではなく，オブジェクトへのポインタと，情報を格納する記憶域である．しかしながら，オブジェクトキャッシュは，通常用いられるキャッシュと同等の構造で実現する事が可能である．オブジェクトキャッシュ管理も通常のキャッシュ管理の延長で可能である．検索キーは，コンスタントプールエントリーに固有となるように決定する必要があるので，クラスのIDとコンスタントプールエントリー等を用いる．参照の解決がソフトウェアエミュレーションで実行された場合，そのルーチンの中でオブジェクトキャッシュに情報を格納する．また，Java仮想マシン中でガーベッジコレクタが起動され実行されると，実際のオブジェクトの位置と，オブジェクトキャッシュが保有するオブジェクトの位置の間で一貫性が保てなくなる可能性がある．そこで，ガーベッジコレクションが実行された場合，オブジェクトキャッシュを無効化するため，キャッシュのフラッシュが必要となる．

4.2 命令動作

Fig.1とFig.3に示される命令の実行処理は，オブジェクトキャッシュを使用する場合，次に示すように動作する．参照するオブジェクトの種別ごとに説明をする．

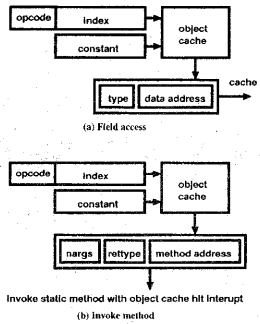

Fig. 6: 静的オブジェクトの参照をする場合

4.2.1 静的参照

命令が静的オブジェクトの参照をする場合，Fig.6のように動作する．フィールド参照の場合は，フィールドのアドレスとデータ型がオブジェクトキャッシュからメモリアクセス実行ユニットに渡される．メソッド呼び出しの場合は，メソッド構造体のアドレス，引数の数等をスタックに積み，ソフトウェアエミュレーションを起動する．

4.2.2 動的参照

命令が動的オブジェクトの参照をする場合，Fig.7のように動作する．フィールド参照の場合は，オブジェクトの先頭からフィールド先頭へのバイトオフセット値とデータ型がオブジェクトキャッシュから渡され，スタック上のオブジェクト参照とオフセット値を用いてアドレス計算した後，メモリアクセス実行ユニットに渡される．メソッド呼び出しの場合は，引数の数とスタックポインタよりスタック上のオブジェクト参照を取得し，情報をスタックに積みソフトウェアエミュレーションを起動する．

4.2.3 定数参照

命令が定数の参照をする場合，Fig.8のように動作する．定数が1ワード長のデータであれば，定数がオブジェクトキャッシュより渡され，2ワード長であれば定数アドレスが渡される．

4.2.4 クラス参照

命令がクラスの参照をする場合，Fig.9のように動作する．オブジェクトキャッシュからは参照すべきデータが複数の場合はクラス構造体のアドレスと，参照するデータへのオフセット値が，単一の場合はそのアドレスが渡され，スタックに積みソフトウェアエミュレーションを起動する．

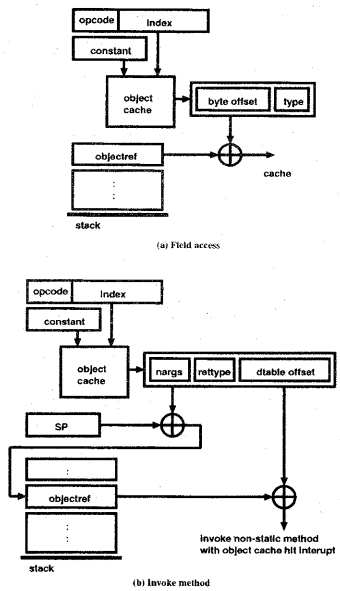

Fig. 7: 動的オブジェクトの参照をする場合

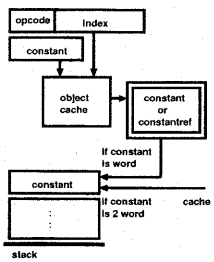

Fig. 8: 定数の参照をする場合

4.2.5 配列アクセス

命令が配列アクセスを行う場合，Fig.10のように動作する．オブジェクトキャッシュには配列長とデータ型が保持されている．配列要素へのアクセスを行う場合，例外のチェックとアドレス計算を行い，データのロード，或いはストアを行なう．`arraylength`の場合，配列長をスタックに積む．

5 kaffeへの実装による性能評価

オブジェクトキャッシュを実装したJava仮想マシンの評価のため，ソフトウェア仮想マシンであるkaffe Virtual Machine [8]にオブジェクトキャッシュを実装し，ベンチマークプログラムによって評価を行った．命令実行エンジンはインタプリタ型を使用した．イン

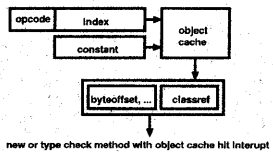

Fig. 9: クラスの参照をする場合

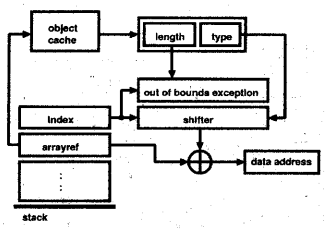

Fig. 10: 配列アクセスを行う場合

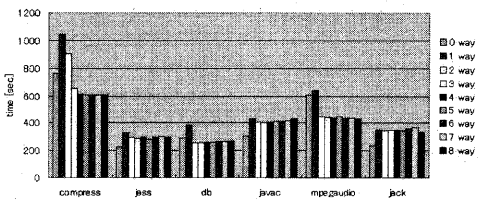

Fig. 11: SpecJVM98 の結果

デックス数は 4 でブロック容量 ×4× 連想度がキャッシュの総容量になる．キャッシュの入れ替えはラウンドロビンで行なっている．実行環境は，Fig.11 のみ Pentium 4 1.7 GHz，RAM 容量 768M，Redhat 7.1 OS のマシンで，以降は Pentium 4 1.6 GHz，RAM 容量 880M，Vine 2.5 OS のマシンである．実行したベンチマークプログラムは，Spec JVM 98 [5]，Caffeine Mark 2.5 [6]，Embedded Caffeine Mark [7] である．なお，このシミュレーションではアレイキャッシュは実装していない．結果について述べる．

5.1 連想度による変化

キャッシュの連想度を 0 から 8 に変化させて測定を行なった．ただし $n = 0$ の場合，オリジナルの kaffe をインタプリタエンジンで実行している．キャッシュブロックの容量は 1k バイトである．Fig.11 から Fig.13 に，ベンチマークプログラムの結果を，Fig.14 に各プログラムにおけるオブジェクトキャッシュのヒット率を示す．なお，Caffeine Mark 2.5 ではキャッシュヒット率が算出不可能であり，Embedded Caffeine Mark では全体に対してのみ算出可能であった．ベンチマーク結果より，Spec JVM 98 の compress，db，mpegaudio 等のクラスの再利用性が高いと考えられるプログラム，Embedded Caffeine Mark の様な小さなプログラムでは性能向上が認められるものの，jess，javac，jack 等のクラスの再利用性が低いと考えられるようなプログラムでは，ヒット率が低くなり，キャッシュミスペナルティーが性能上の大きなオーバーヘッドになっている．キャッシュ構成による性能の変化に注目すると，ダイレクトマップキャッシュにした場合は低下がみられ，以降は連想度が 2 から 4 でほぼ最大となっていることがわかる．

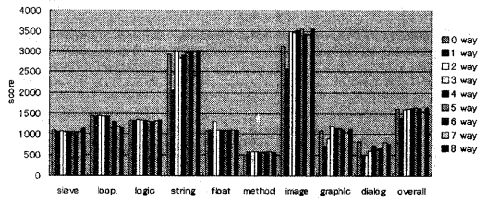

Fig. 12: Caffine Mark 2.5 の結果

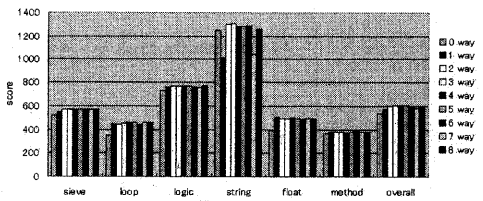

Fig. 13: Embedded Caffeine Mark の結果

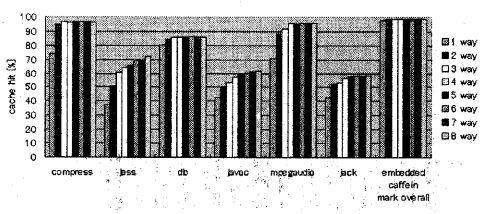

Fig. 14: オブジェクトキャッシュヒット率

5.2 ブロック容量による変化

次に，連想度 2 と 4 の場合においてブロック容量を 128 バイトから 1k バイトまで変化させて測定を行なった．Fig.15 から Fig.17 に，ベンチマークプログラムの結果を，Fig.18 に各プログラムにおけるオブジェクトキャッシュのヒット率を示す．結果として，キャッシュが同容量，或いは低容量の場合でも，連想度が 4 の場合の方が全体的に性能が上がっているこ

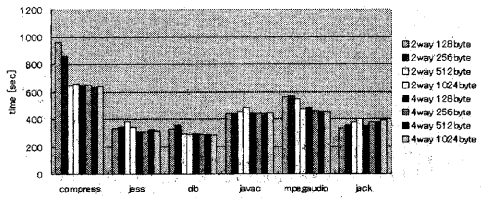

Fig. 15: SpecJVM98 の結果

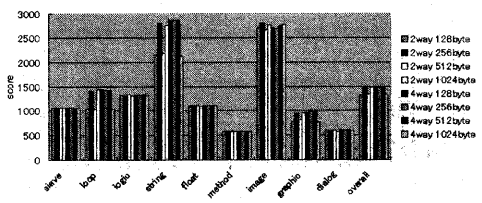

Fig. 16: Caffine Mark 2.5 の結果

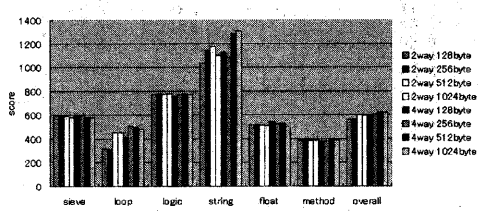

Fig. 17: Embedded Caffeine Mark の結果

とがわかる.

6 結論

オブジェクト操作命令のハードウェア的な解決方法の1つとしてのオブジェクトキャッシュと，配列アクセス高速化のためのアレイキャッシュについて述べてきた．今回，性能評価を行なうことで，設計に関して有用なデータを得る事が出来た．Fig.19 に設計中の Java プロセッサのブロック Fig. を示す．オブジェクトキャッシュの実装に関しては，ハッシングアルゴリズム，置き換えアルゴリズムに関しては，まだ検討の余地があると考えている．これらは，Java 仮想マシンのガーベッジコレクションアルゴリズムと密接に関わっている．Java 実行の大きなオーバーヘッドはガーベッジコレクションにあり，これらを複合的に考えることで性能を伸ばせる可能性がある．今後は，ソフトウェア部を含めた実行環境について検討を進めていく．

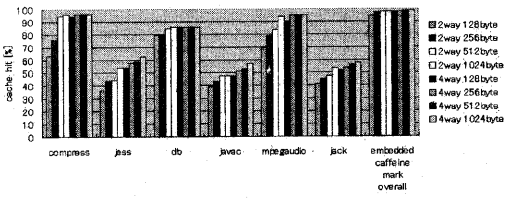

Fig. 18: オブジェクトキャッシュヒット率

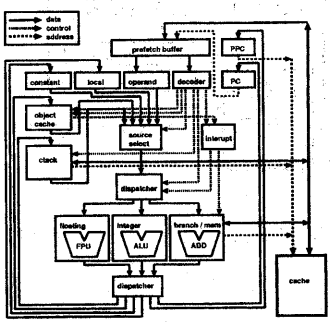

Fig. 19: Java プロセッサブロック Fig.

参考文献

[1] Sun Microsystems
http://java.sun.com

[2] Lindholm, T. and Yellin, F. 著, 野崎裕子 訳: Java 仮想マシン仕様, アジソン・ウェスレイ・パブリッシャーズ・ジャパン (1997)

[3] Lindholm, T. and Yellin, F. 著, 村上雅章 訳: Java 仮想マシン仕様第 2 版, ピアソン・エデュケーション (2001)

[4] Meyer, J. and Downing T. 著, 鷲見 豊 訳:"Java バーチャルマシン" オライリージャパン

[5] Spec
http://www.spec.org/

[6] Caffeine Mark 2.5
http://www.webfayre.com/pendragon/cm2/

[7] Embedded Caffeine Mark
http://www.webfayre.com/pendragon/cm3/

[8] Kaffe Virtual Machine
http://www.kaffe.org

[9] 研究室 HP
http://shimizu-lab.et.u-tokai.ac.jp/

[10] 内藤 亮, 清水尚彦: パルテノンによるパイプライン JAVA チップの設計, 第 12 回パルテノン研究会予稿集

[11] 近 千秋, 清水尚彦: オブジェクトキャッシュを用いた JVM 命令互換プロセッサの設計, 情処学会研報 ARC-145-7 2001

[12] 近 千秋, 清水尚彦: オブジェクトキャッシュを用いた JVM 命令互換プロセッサの設計, 信学技報 CPSY2001-108