Dalvik アクセラレータのハードウェア実装

吉實大輔[†] 太田淳[†]
三輪忍[††] 中條拓伯[†]

これまで，Android アプリケーションの高速化手法として，Dalvik アクセラレータを提案してきた．Dalvik アクセラレータはプロセッサ内部に搭載することで Dalvik バイトコードを直接実行する方式である．ここでは，Verilog-HDL により，Dalvik アクセラレータの実装を行い，論理合成後にそのハードウェア量を評価した．これにより，搭載しない場合と比べて約 17%のハードウェア量増加となることがわかった．

Hardware Implementation of a Dalvik Accelerator

Daisuke Yoshizane,[†] Atsushi Ohta,[†] Shinobu Miwa[††]
and Hironori Nakajo [†]

In this paper, we describe a hardware implementation of Dalvik Accelerator to accelerate Android applications. A Dalvik Accelerator is coupled into a processor in order to execute Dalvik Bytecodes directly. We have implemented a Dalvik Accelerator with Verilog-HDL and have estimated the amount of additional hardware to the base processor. From the estimation, the amount of hardware increases in 17% than without the accelerator.

1. はじめに

Google 社の携帯情報端末向けプラットフォーム，Android で用いられるアプリケーションでは，Dalvik バイトコードという独自の中間言語の形で実行される．その Dalvik バイトコードは，VM により逐次翻訳されるため，実行速度が低速であるという問題がある．

これまでこの問題に対し，Dalvik アクセラレータ[1)]による高速化を提案してきた．Dalvik アクセラレータをプロセッサに組み込むことで，Dalvik バイトコードをプロセッサで直接実行できるようになる．

Dalvik アクセラレータを携帯情報端末に搭載するにあたり，ハードウェア量を抑える必要がある．そこで，Verilog-HDL により Dalvik アクセラレータの実装を行い，ハードウェア量を評価した．

2. Dalvik アクセラレータ

Dalvik アクセラレータは，ARM 社の Jazelle 方式[2)]に倣った方式であり，既存のプロセッサに組み込むことで Dalvik バイトコードを直接実行することができる．既存のプロセッサのフェッチステージおよびデコードステージと並列に接続され，以降のステージは既存のプロセッサのものを用いる (図 1).

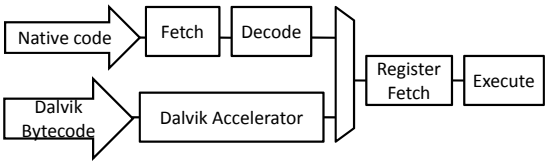

図 1 Dalvik アクセラレータを搭載したプロセッサ
Fig. 1 Processor with Dalvik Accelerator

Dalvik アクセラレータは Dalvik バイトコードをフェッチし，機械語群に変換して，以降のステージに送る．よって以降のステージでは機械語として処理されるため，既存のステージが利用可能である．

3. Dalvik アクセラレータの実装

Dalvik アクセラレータは以下のモジュールからなる (図 2).

3.1 フェッチバッファ

フェッチバッファは 96bit 幅のリングバッファである．Dalvik バイトコードは可変長であり，一度バッファでバイトコードの形を作る必要がある．メモリ

† 東京農工大学
Tokyo University of Agriculture and Technology
†† 東京大学
Tokyo University

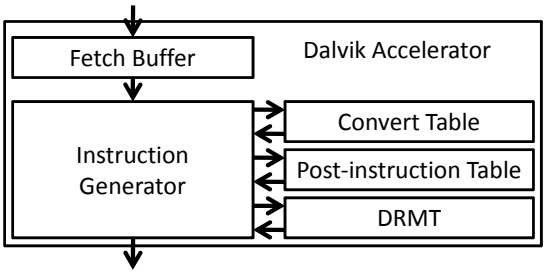

図 2 Dalvik アクセラレータのブロック図
Fig. 2 Block Diagram of Dalvik Accelerator

から Dalvik バイトコード群を 32bit ずつフェッチし，バッファへ格納する．その後，オペコードからコード長を判別して，格納済みのビット数がそのコード長を超えていたら命令ジェネレータへ送る．

3.2 命令ジェネレータ

命令ジェネレータは Dalvik バイトコードを機械語群に変換する機構である．機械語の発行は 3.3〜3.5 節に挙げる 3 つのテーブルを用いて行う．

命令ジェネレータはフェッチャからバイトコードを受け取るとデコードを行う．バイトコードが可変長であるため，デコードの仕方もオペコードごとに異なる．その後，オペコードやオペランドをもとに各テーブルを参照し，機械語を生成する．

3.3 変換テーブル

変換テーブルは，Dalvik バイトコードから最初に出力される機械語を格納している．各エントリへの参照は，Dalvik バイトコードのオペコードをインデックスとして用い行われる．

機械語に組み込まれる値については，即値，Dalvik バイトコードのオペランド，Dalvik レジスタの 3 種類を定義している．即値は，格納されている値を直接機械語に組み込む．Dalvik バイトコードのオペランドは，Dalvik バイトコードから取得した値を組み込む．Dalvik レジスタは後に述べる DRMT を参照して，対応する物理レジスタ番号を組み込む．

3.4 後続命令テーブル

後続命令テーブルは，変換テーブルで出力された機械語に後続して出力すべき機械語を格納している．各エントリは変換テーブルと同様の形式で格納され，次に出力すべき機械語のエントリへのポインタを持つ．

3.5 DRMT(Dalvik Register Mapping Table)

DRMT は，物理レジスタと Dalvik レジスタのマッピングを記録するテーブルである．Dalvik レジスタはメモリ上の仮想的なレジスタであり，物理レジスタに割り当てることで利用される．

3.6 実装および動作確認

以上のモジュール群を MIPS アーキテクチャ向けに実装し，動作を検証した．テストベンチを作成し，Dalvik バイトコード群を順に読み込ませたところ，事前の手計算で得られたものと同一の MIPS 命令群が順に生成された．

4. ハードウェア量の評価

実装した Dalvik アクセラレータのハードウェア量を評価した．比較には，MIPS プロセッサのシミュレータである MipsCore[3] を用い，これに搭載した場合を想定してハードウェアの増加量を評価した．

それぞれを Xillinx ISE Design Suite 13.2 で論理合成したところ，スライス数が MipsCore は 1530，Dalvik アクセラレータは 267 となった．また，4 入力 LUT の数は MipsCore は 4345，Dalvik アクセラレータは 942 となった．よって MipsCore に Dalvik アクセラレータを搭載する場合，ハードウェア量は約 17%の増加となることがわかった．

5. まとめと今後の課題

本稿では，Dalvik バイトコードをプロセッサ上で実行するための機構，Dalvik アクセラレータの実装を紹介した．実装した Dalvik アクセラレータのハードウェア量を評価したところ，既存プロセッサに対し約 17%の増加となるとわかった．

今後の課題として，機械語の発行方法の見直しが挙げられる．現在の設計では，1 つの機械語の発行に 6〜8 クロックかかる．アクセラレータとして活用するには，このクロック数を抑える必要がある．

謝辞 本研究の一部は，独立行政法人日本学術振興会とフィンランドアカデミーとの二国間交流事業（共同研究）による支援を得た．

参 考 文 献

1) 太田淳, 三輪忍, 中條拓伯: Android 端末におけるハードウェアによる Java の高速化手法の提案, 情報処理学会論文誌コンピューティングシステム（ACS）, Vol. 4, No. 3, pp. 115–132 (2011).
2) Capewell, P. and Watson, I.: A RISC Hardware Platform for Low Power Java, *Proceedings of the 18th International Conference on VLSI Design*, pp. 138–143 (2005).
3) 渡邉伸平, 藤枝直輝, 若杉祐太, 高前田伸也, 森洋介, 吉瀬謙二: MIPS システムシミュレータ SimMips を活用した組込みシステム開発の検討, 情報処理学会研究報告. EMB, 組込みシステム, Vol. 2008, No. 116, pp. 23–28 (2008-11-20).