B-032

ユビキタスプロセッサ用並列化コンパイラの開発

Development of a Ubiquitous Processor compiler

村上 亮輔† 横山 温子† 深瀬 政秋† 佐藤 友暁‡

Ryosuke Murakami, Atsuko Yokoyama, Masa-aki Fukase, Tomoaki Sato

1. はじめに

著者らは組込みプロセッサの負荷軽減と高機能化の両立を目指し，PC プロセッサ以上のレベルで主流となったマルチコアとマルチパイプの並列化技術及び省電力化を狙ったウェーブ化パイプライン技術を応用したユビキタスプロセッサ HCgorilla の開発を進めてきた．

HCgorilla はハードウェア暗号組み込み型でしかも Java 対応型であるため、開発には H/S 協調が不可欠である。本研究は HCgorilla の H/S 協調設計の一環として，小規模なハードウェア資源に対して十分な並列度を抽出するためのコンパイラを開発する．

2. HCgorilla における H/S 協調開発

2.1 HCgorilla とソフトウェア支援システム

HCgorilla の H/S 協調開発では，ユビキタス環境への対応に必要な機能と性能の抜本的な向上のため，Java クラスファイルに関わる重い処理は，大規模サーバで集中的に行う．そのための Java インターフェース とコンパイラを大規模サーバに配置する．ハードウェア側では，サーバから受け取ったバイトコードを Java CPU で直接実行する．図 2 は，ソフトウェア支援システム全体と HCgorilla の関係を表す．

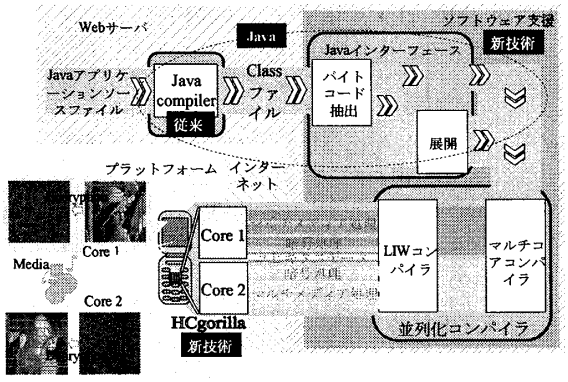

図 1. HCgorilla とソフトウェア支援システム

2.2 Java CPU と Java インターフェース

HCgorilla は，JVM を使わず，インタープリタ型 Java CPU による Java byte code の直接実行を行う．これによりメモリの節約，ハードウェア資源の有効活用が可能になる．HCgorilla 上で java byte code を直接実行するためには，HCgorilla において未実装の JVM 命令セットの命令を既存の命令の組み合わせにより，同様の動作をさせる展開の必要がある．

具体的には Java クラスファイルから java byte code を抽出し，java byte code が格納されている場所までバイナリデータの判別を行いながら，インストラクションコードのバイト長(4byte)まで解析を行う．得られたバイナリデータの値と同じ数の java byte code 列を書き込み用ファイルへ書き込む．この操作を Method の個数分繰り返し，未実装命令の展開を行う．

2.3 マルチコアと Muliticore コンパイラ

マルチコアは ILP より粒度の粗い並列性に対処可能で，プロセッサ全体として性能が向上するため，近年ではサーバ，デスクトップシステムのみに留まらず，組み込み系のプロセッサでもマルチコア化は進んでいる．

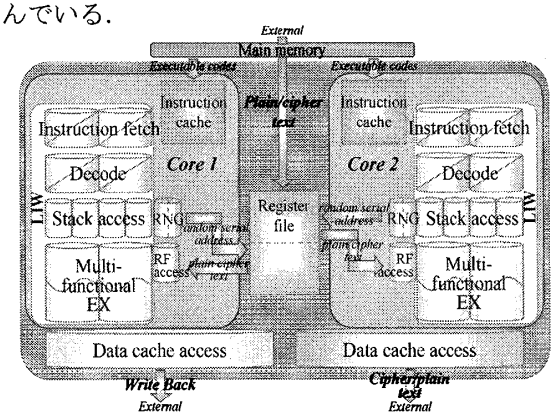

図 2. HCgorilla の構成

図 1 のように HCgorilla は，二つのコアを持つデュアルコアプロセッサである．マルチコア化により TLP(Thread Level Parallerism)を活用した性能向上が期待できる．Multicore コンパイラは，HCgorilla で実行できる java byte code のマルチスレッド化を行い，性能向上を目指す．

2.4 LIW(Long Instruction Word)

HCgorilla は，命令レベル並列化 (Instruction-Level Parallelism : ILP)を備えている．ILP は VLIW アーキテクチャと同じ方式である．しかし， VLIW のように大規模ではないので， LIW (long instruction- word)と称することにした．

† 弘前大学大学院理工学研究科電子情報システム工学専攻
Graduate School of Sciense and Technology ,
Hirosaki University

‡ 弘前大学総合情報処理センター,
Information Processing Center, Hirosaki University

LIW スキームは，１つの命令語中に依存関係の無い２つの命令(ロード，ストア，演算など）を格納しておき，それらを全て同時に実行するものである．

3. 並列化コンパイラ

並列化コンパイラは Multicore コンパイラと LIW コンパイラから構成され，Java インターフェースの出力を入力とする．

3.1 Multicore コンパイラ

Multicore コンパイラは java byte code を二つのコアに振り分けるため二つのスレッドに分割する． 並列化は ILP より粗い粒度のタスクを対象としており，java プログラムのメソッド，ライブラリ関数，ループ関数等を抽出し，スレッド単位でそれぞれのコアに割り当てる．この際アドレス競合が起こらないようにメモリアドレスを変換している．図 3 に Muliticore コンパイラのアルゴリズムを示す．

3.2 LIW コンパイラ

LIW コンパイラでは，先に分けられたスレッドでそれぞれ，命令の実行順序などを変更・編集する．並列性の抽出は，store 命令などの一つの処理の終わりとなる部分を一区切りとし， 一つの処理の終了の直後からその次の処理の終了までを一区切りと見て行う．
前後の二つの処理で store， load 先のアドレスを見て並列化判定を行い，前の処理の対象アドレスと後の処理の対象アドレスが同じでなく，かつ jump 命令が存在しない場合において並列化可能と判定し，LIW 化を行う．図 4 に LIW コンパイラのアルゴリズムを示す．

3.3 IPC

並列化コンパイラを使用した前後での IPC(Instruction per clock cycle)の値の比較を行った．IPC とは 1clock 当たりの実行命令数を示した値で、この値が大きいほど実行効率が良いことを示す．加算/Combination の計算を行うテストプログラムにおいて，並列化前後で IPC の値が向上していることが確認できた．図 5 に IPC 比較のグラフを示す．

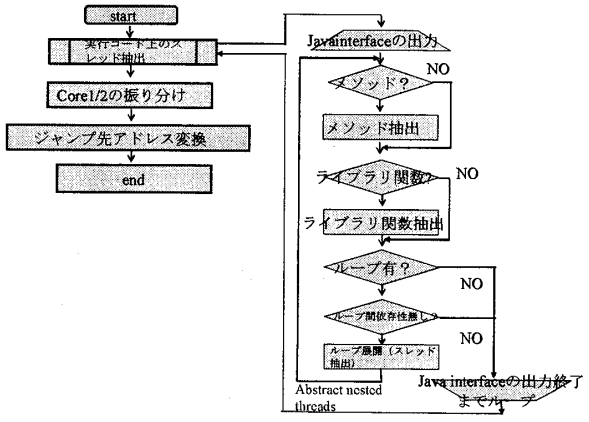

図 3．Multicore コンパイラのアルゴリズム

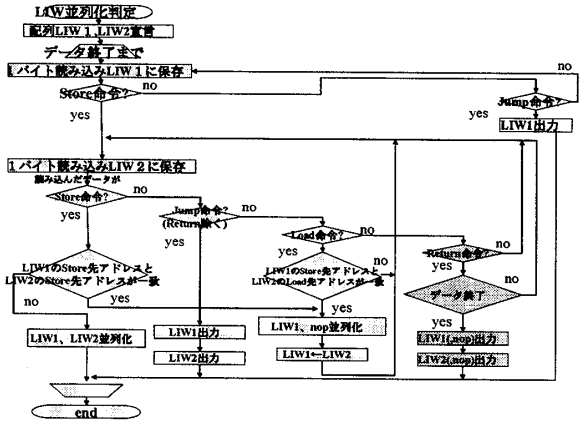

図 4．LIW コンパイラのアルゴリズム

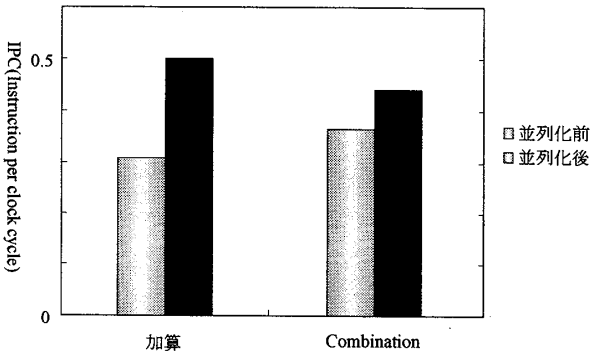

図 5．並列化前後の IPC 比較

4. まとめ

本研究では，TLP 及び ILP を利用した，ユビキタスプロセッサ HCgorilla 用並列化コンパイラの開発を行った．並列化コンパイラは JAVA プログラムのメソッドを二つのスレッドに分け，さらにそれぞれのスレッドについて命令の依存性をしらべ並列化を行い HCgorilla の実行コードを生成する．開発言語は JAVA 言語を用い，スレッド分割を行う multicore コンパイラ，ILP に基づく java byte code の LIW 化を行う LIW コンパイラを作成した．

今後の課題は以下の通りである．

a. multi-core コンパイラのスレッド振り分け手法の改良
b. LIW コンパイラの並列度の向上．
c. 並列性を考慮したテストアプリケーションの試作

参考文献

[1] 村上亮輔，横山温子，深瀬政秋，佐藤友暁，“負荷分散型次世代ユビキタスシステム用並列化コンパイラ”，平成 20 年度 電子関係学会東北支部連合大会，pp. 224 (2008)
[2] 村上亮輔，深瀬政秋，佐藤友暁，“負荷分散型次世代ユビキタスシステム用並列化コンパイラの評価”，情報処理学会東北支部 第二回研究会 (2008).