

プロファイルを利用したコード最適化システム

4D-5

船間政昭

磯崎博子

日本電気(株)

1 はじめに

プログラムコード実行性能向上のため、プログラムの実行状況(プロファイル情報)を利用したプログラムコード最適化技術が重要視されている。本論文では組み込みシステム向けに実行性能評価、プロファイル情報取得、プログラムコード最適化などの最適化作業を効率良く行う、プロファイルを利用したプログラムコード最適化システム及びその評価に関して述べる。

2 開発の背景と目的

これまで組み込みシステムにおけるコード生成に関しては、プログラムの実行性能よりもサイズを重要視する傾向があり、実行性能の向上はパイプライン、キャッシュメモリなどのハードウェアアーキテクチャに頼る所が大きかった。しかし現在ではハードウェアアーキテクチャの特徴を最大限に利用したプログラムコード生成による実行性能向上も同時に求められている。

これを達成するためには、そのハードウェアアーキテクチャについての深い知識が必要である事と実行性能評価、コード最適化などの作業を繰り返し行う処理プロセスの複雑さがユーザにとって大きな負担となっていた。また、ハードウェアアーキテクチャの利用効率を向上させるためのコード最適化手法の開発は研究レベルでは進んでいるがこれを簡単に利用できる環境が存在しなかった。この状況を改善するため、コード最適化作業の効率を向上し、ハードウェアアーキテクチャの利用効率を向上させる最適化を利用可能とする開発環境の提供が本システムの開発の目的である。

3 最適化システム

プロファイル情報を利用した最適化としては命令キャッシュ、データキャッシュ、パイプラインなどを考慮したコード最適化が考えられるが、本論文では現在実装済である命令キャッシュ利用効率の向上を目的とした最適化を中心に述べる。

3.1 システム構成

本最適化システムの構成を図1に示す。本最適化システムはプログラムの実行性能を評価する「性能評価部」、プログラムを実行し得られるプロファイル情報を利用してコード最適化を行う「プロファイル-キャッシュ最適化部」、従来のリンカ機能に加え、GUIによるメモリおよびキャッシュ上でのプログラムのマッピング情報表示やドラッグ&ドロップによる関数再配置を可能とした「ビジュアルリンカ」、以上3つの機能のフロントエンドとしてのGUI機能を提供し、ソースファイル管理などを行う「ドライバ」の4つで構成される。

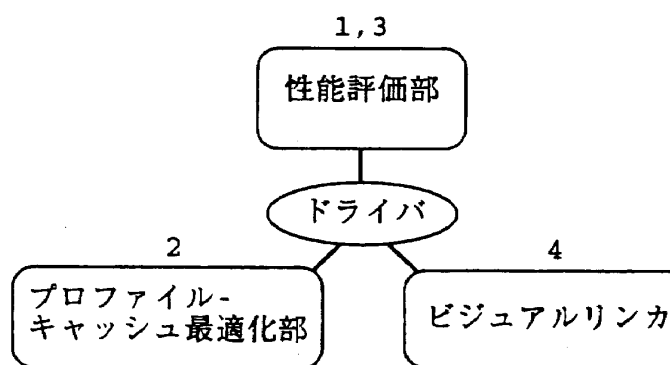


図1: システム構成

ユーザはGUIベースのドライバのみを介して次の4つの行程を繰り返すことで効率的に最適化作業を行うことができる。それぞれの処理は図1の対応する番号の部分で行われる。

1. プログラムの現在の実行性能評価
2. プロファイリング及びキャッシュ最適化
3. プログラムの最適化後の実行性能評価
4. ビジュアルリンカを用いた手動による関数のメモリ配置の微調整

ドライバのウィンドウ画面を図2に示す。

3.2 性能評価とプロファイリング

性能評価部ではシミュレータを用いて対象とするプログラムのキャッシュミス率、キャッシュ動作情報、実行クロック数などの性能評価を行い、結果を表示する。

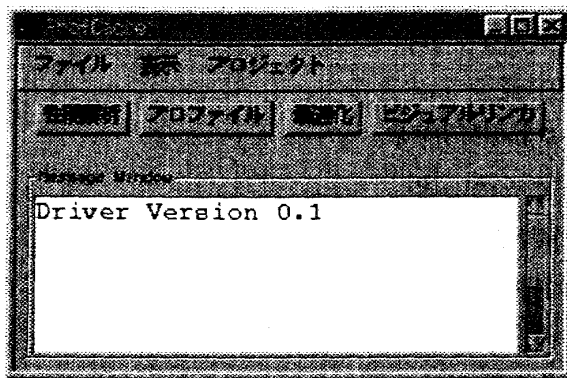


図 2: ドライバウィンドウ

「プロファイル-キャッシュ最適化部」はプログラムをシミュレータ上で実行し最適化に必要な関数コールペア（関数の呼出し元と呼出し先関数のペア）とその呼出し回数などのプロファイル情報を得る。

3.3 コード最適化

「プロファイル-キャッシュ最適化部」は命令キャッシュの衝突の発生を抑えるために前述のプロファイル情報を用いて、次の2つの最適化処理を行う。1つはライブラリ関数を含んだ関数単位でのコード再配置(図3(a))、もう1つは関数の複製(図3(b))である。

前者は Cache Line Coloring[1] をベースとしたアルゴリズムを用いて頻繁に呼び出される関数と呼び出し元関数とを、命令キャッシュ上において重ならないようにそれぞれの関数を配置しようとする。

前者の関数単位のコード再配置は呼び出される関数について複数の呼び出し元関数が存在する場合、すべての呼び出し元関数との衝突を回避しようとするが、これは困難な場合が多い。このため優先度の高いコールペアについて呼び出される関数を複製しこれと呼び出し元関数が呼び出すように修正した後、関数単位のコード再配置を行うように改善を加えたのが後者である。

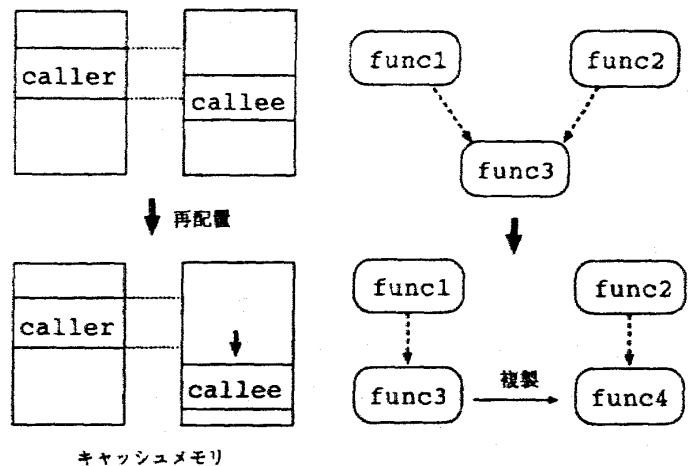
3.4 ビジュアルリンカ

ビジュアルリンカは従来のリンカ機能に、GUI を付加しメモリおよびキャッシュ上でのプログラムのマッピング情報表示やマウスによるドラッグ&ドロップによる関数再配置を行う機能を付加したリンカである。

本システムではコード最適化部により関数の再配置を行うことで実行性能を向上させるが、手動で配置を微調整する事でさらに実行性能を向上できる場合がある。このような場合に、ビジュアルリンカを用いれば容易に関数配置を変更し性能を改善することが可能となる。

4 評価結果

本システムを dhrystone, mpeg_play, rayshade(レイトレーシング処理) の3つのアプリケーションに対して



(a) 関数単位での再配置

(b) 関数の複製

図 3: 最適化

適用した結果を表1に示す。ターゲットCPUとして4KBの命令キャッシュを持つNEC製32bit RISC CPU V830を対象として評価を行った。

dhrystoneでは大きくミス率が減少しているがこれは関数再配置により命令キャッシュ内に、実行に必要なすべての関数が配置された事によるものである。他の2つのアプリケーションは命令キャッシュに入り切らないものであるが関数の再配置により命令キャッシュミス率を平均37%削減でき、これにより実行性能は9%～20%向上している。

表 1: 評価結果

評価対象 プログラム	命令キャッシュミス率		速度向上
	最適化前	最適化後	
dhrystone	0.7%	0.1%	+10.8%
mpeg_play	1.3%	0.7%	+9.0%
rayshade	8.2%	6.9%	+20.1%

5 まとめと今後の課題

本論文では、プロファイルを利用したコード最適化システムとその評価について述べ、本コード最適化システムがプログラムのコード実行性能向上のために有効である事を示した。

今後は最適化のためのガイド機能の追加などのユーザインタフェースの改良と関数単位よりもさらに細かい基本ブロック単位でのプログラムコード再配置、データキャッシュに関する最適化、パイプラインを考慮した最適化などの最適化手法の実装を予定している。

参考文献

- [1] Amir H.Hashemi, David R.Kaeli, Brad Calder, "Efficient Procedure Mapping Using Cache Line Coloring", WRL Research Report 96/3