動的最適化のためのハードウェアホットパス検出機構

中島 伸吾 † 横田 隆史 † 大津 金光 † 馬場 敬信 †
† 宇都宮大学大学院工学研究科情報工学専攻

1 はじめに

プログラム実行の高速化ために，プログラム実行中にプログラムの解析を行い，その解析結果から実行挙動に依存した情報を用いて最適化を行う動的最適化という手法が注目されている．プログラム最適化において，全てのコードに最適化を施すのにはコストがかかりすぎる．一般に“プログラム実行時間の 90%はコードの 10%が占めている” といわれており，コード中の実行頻度の高い 10%の部分を解析し，その部分に特化した最適化を行うことでプログラム実行の高速化を実現できる．

我々は，最適化対象をホットスポットになる可能性が高いループ部分に限定し，実行頻度の高いループ (ホットループ) とループ内の実行頻度の高い実行経路 (ホットパス) を特定するために，“ループパス” を提案し，シミュレータ上でホットパス検出機構の研究・開発を行っている [1].

しかしこの機構はハードウェア実装のことは考慮されておらず，信号の入出力のタイミングやオーバーフローの対応など様々な問題点がある．本研究ではこれらの問題を解決し，ホットパス検出機構がハードウェアで実現できることを実証し，動的最適化システム構築のためにホットパス検出機構のハードウェア実装を行う．

2 ループパス

ループパスとは，パスの末尾の後方分岐先のアドレスがパスの先頭アドレスと一致するパスのことである．またループパスは，“先頭が一致するループパスの集合は，同じループ内のパスの集合である” という性質をもっている．ループパスは以下の情報を用いて表現する．

- 開始アドレス (bt_start_addr)
  Bit Tracing の開始アドレス，つまりループの先頭のアドレスを格納する．
- 分岐履歴 (bt_history)
  分岐結果である Taken/Not-Taken をそれぞれ“1” と“0”で記録する．
- 継続アドレス (bt_next_addr)
  通常は後方分岐先のアドレス (NPC) を格納するが，分岐履歴オーバーフローや間接分岐との遭遇でパスが終端した際には，次に続くパスの開始アドレスを格納する．
- パス情報 (bt_info)
  最適化を行う際に参考となる情報及び Bit Tracing を行う際に必要な情報を格納する．
  - O:分岐履歴オーバーフローを示す．
  - C:継続パスであることを示す．
  - R:サブルーチンコール発生を示す．
  - I:間接分岐が発生したことを示す．
  - S:システムコール発生を示す．
- パスに含まれる命令数 (bt_weight)
  ループパスに含まれている命令数を管理する．

A Hardware Hot Path Detector for Dynamic Optimization
†Shingo Nakajima,Takashi Yokota, Kanemitsu Ootsu and Takanobu Baba
Department of Information Science, Graduate School of Engineering, Utsunomiya University (†)

3 ホットパス検出機構

ホットループパス検出機構は，Bit Tracing[2] によってループパスの検出を行う Bit Tracing Stack (BTS), BTS が検出したループパスの情報を基に実行頻度の高いループ (ホットループ) を検出する Hot Loop Detector (HLD), BTS で検出したループパスを管理する Hot Path Accumulator (HPA) で構成される．

図 1 にその構成図を示す．

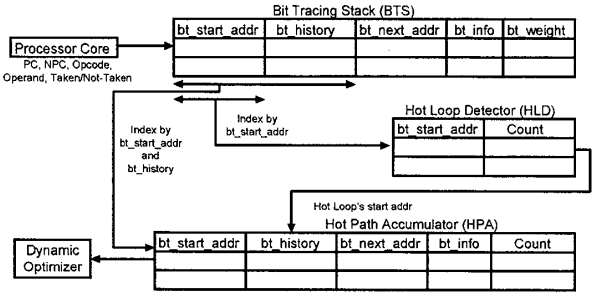

図 1: ホットパス検出機構

3.1 BTS におけるループパス検出

BTS はプロセッサコアから，命令コード (Opcode)・オペランド・現在の命令のプログラムカウンタ (PC)・次に実行する命令プログラムカウンタ (NPC)・分岐の成立/不成立 (Taken/Not-Taken) を受け取り，Bit Tracing[2] によってループパスの検出を行う．

Bit Tracing 中であることを示すために，bt_start フラグを用意し，初期値を False に設定しておく．後方分岐が発生した際にその分岐先 (NPC) を bt_start_addr に格納し，bt_start を True にして Bit Tracing を開始する．次の後方分岐が発生するまで，前方分岐の分岐結果 (Taken/Not-Taken) を bt_history に記録していき，次の後方分岐が発生したら分岐先 (NPC) を bt_next_addr に格納する．そして格納された bt_start_addr と bt_next_addr を比較し，一致したらループパスと判断し HLD, HPA へそのループパスの情報を出力し，不一致なら内容を破棄する．

Bit Tracing 中に分岐履歴オーバーフローが発生，またはレジスタ間接分岐に遭遇した場合は，bt_next_addr に次の命令のアドレス (NPC) を格納し，bt_info に ‘O’ または ‘I’ を記録し，パスを分割する．そして，新たなエントリの bt_start_addr に NPC を格納し，bt_info に継続パスであることを示すフラグ ‘C’ を格納し，Bit Tracing を継続する．

3.2 HLD におけるホットループの検出

HLD テーブルの各エントリは，BTS が検出するループパスの情報のうち bt_start_addr を格納するフィールドとそのループパスに含まれる命令数をカウントする count フィールドで構成されている．テーブル内の探索は bt_start_addr をインデックスとしてハッシングを行い，同じ bt_start_addr のループパスがすでにテーブル内に格納されている場合は，そのエントリの count に検出されたループパスに含まれる命令数である bt_weight を加算する．存在しない場合は，新たなエントリを追加する．このとき，LFU (Least Frequently Used) 置換方式に従い，置換を行っている．

3.3 HPA におけるホットループパスの検出

HPA テーブルの各エントリは，BTS が検出する 5 つのループパスの情報の中から bt_weight を除いた 4 つの情報を格納するフィールドとループパス内で実行された命令数をカウントする count フィールドで構成されている．テーブル内の探索は，bt_start_addr と bt_history をインデックスとしてハッシングを行い，同じループパスが HPA テーブル内に存在している場合は，そのエントリの count に bt_weight を加算する．存在しない場合は，新たなエントリを追加する．このとき，LRU (Least Recently Used) 置換方式に従い，置換を行っている．

また HPA は，HLD がホットループを検出した際に，ホットループの bt_start_addr に該当するループパスをホットループパスとして出力する．これにより，動的最適化機構にホットループ，ホットループパスの情報を提供する．

4 ハードウェアホットパス検出機構

ホットパス検出機構は，ハードウェア実装のことは考慮されておらず，様々な問題が残されている．以下に問題とその検討結果について述べる．また図 2 にハードウェアホットパス検出機構の概略図を示す．

- 入力データのタイミング調整
  対象プロセッサは，6 段パイプライン構成になっており，Bit Tracing に必要な情報はそれぞれ異なるステージから出力される．そのため，対象プロセッサと同様のパイプラインレジスタを挿入することで，入力データのタイミングを調整する．
- FIFO の挿入 (BTS)
  BTS が動作中に新たな分岐命令を受け取ってしまう可能性があるため，プロセッサ，BTS 間に FIFO を挿入する (以降，BTS-FIFO と称す)．受け取った命令が分岐命令だった場合，BTS-FIFO にデータを書き込む．書き込むデータは，現在の命令のプログラムカウンタ (PC)，次の命令のプログラムカウンタ (NPC)，分岐命令 (Branch)，分岐の成立/不成立 (Taken/Not-Taken)，分岐命令間で実行された命令数 (weight) である．そして，BTS が待機状態のときに BTS-FIFO の内容を読み出して Bit Tracing を行う．
- 置換の方法 (HLD, HPA)
  シミュレータ版ホットパス検出機構の HLD，HPA で行っている LFU や LRU などの置換方式をハードウェア上で実現することは難しい．そのため，テーブルを 2 つに分割し，1 つの入力アドレスに対し 2 つのデータを保持出来るようにする．そして，Reference bit を設け，どちらのテーブルが最近更新されたかを記録しておく．置換が必要になった際は，Reference bit を参照し古いほうのテーブルを選択し，上書きする．
- ホットループパスの出力 (HPA)
  HPA は bt_start_addr と bt_history をインデックスとしてハッシングをしている．そのため，HLD からホットループの先頭アドレスを受け取ったときに，対応するループパスが格納されているエントリを探すために HPA 内を全走査しなくてはならない．このとき，HPA に格納されている継続パスの bt_start_addr はループの先頭アドレスではないため，継続パスの出力ができない．そこで，HPA に loop_start_addr というフィールドを設ける．loop_start_addr には，ループの先頭アドレスを格納する．HLD からホットループの先頭アドレスを受け取ったときは，この loop_start_addr と比較を行い，一致するパスをホットループパスとして出力する．
- BTS，HPA 間のタイミング (HPA)
  上記のように HPA におけるホットループパスの出力には，HPA 内を全走査するため時間がかかってしまう．この間に BTS が新たなループパスを検出してしまう可能性がある．そのため，BTS，HPA 間に FIFO を挿入する (以降，HPA-FIFO と称す)．BTS が検出したループパスの情報を HPA-FIFO に格納し，HPA が待機状態のときに HPA-FIFO の内容を読み出す．
- オーバーフローの対応
  オーバーフローを起こす可能性があるのは，BTS，BTS-FIFO，HPA-FIFO である．BTS がオーバーフローした場合は，その時点で BTS に格納されているデータを全て破棄し，Bit Tracing を継続する．BTS-FIFO がオーバーフローした場合は，プロセッサを停止させる．HPA-FIFO がオーバーフローした場合は，BTS を停止させる．

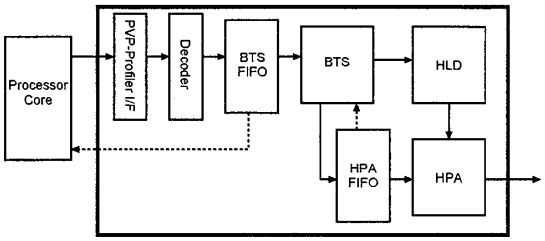

図 2: ハードウェアホットパス検出機構の概略図

4.1 シミュレーションによる動作検証

設計したハードウェアホットパス検出機構をアサーション検証により正常に動作していることを確認した．

4.2 論理合成結果

設計したハードウェアホットパス検出機構の論理合成結果を表 1 に示す．() 内は対象ボードである DN3000K10 の 1 つの FPGA に書き込み可能な総資源量に対する使用率である．

表 1: ホットパス検出機構の論理合成結果

	ホットパス検出機構	総資源量
F/F	840 (5%)	67,584
LUT	1,123 (2%)	67,584
スライス	741 (2%)	33,792
ブロック RAM	5 (3%)	144

5 おわりに

本稿では，ホットパス検出機構のハードウェア実装を行い，シミュレーションにより動作を確認した．またこのときの論理合成結果から，ハードウェアホットパス検出機構が低コストで実現可能なことがわかった．

謝辞

本研究は，一部日本学術振興会科学研究費補助金（基盤研究 (B)18300014，同 (C)19500037，同 (C)20500047）および宇都宮大学重点推進研究プロジェクトの援助による．

参考文献

[1] 矢野目 秀人，増保 智久，大津 金光，横田 隆史，馬場 敬信，“ループパスに基づいたプログラムの挙動解析”，電子情報通信学会コンピュータシステム研究会 (CPSY)，信学技報，Vol.108, No.303 (CPSY2008-38), 2008 年 11 月．

[2] Vasanth Bala, “Low overhead path profiling”, HP Laboratories Technical Report HPL-96-87, 1996.