UXREROLLING
(brand-new rerolling algorithm)

今田俊寛

目的

- 最適な ループリローリングアルゴリズム を検討する。
  - 期待する効果
    - 命令キャッシュの有効利用
    - ベクトル化の促進
    - レガシーなコードの性能向上

課題

- 一般に知られている“ループリローリング”アルゴリズムでは、比較的単純なループしか、リローリング(巻き戻し)出来ない。

- 何故か？
  - 既存の技術では、複雑なループに対して適応するのが、困難であり、かつ複雑なロジックが必要。

解決法1 (前提)

- ループアンロールされたコードは・・・
  - 手動・自動問わず、
    - 誘導変数の増分値
    - 配列の位置を示す添字
    - 配列の型
  - に対し、最大公約数アルゴリズム (GCD) を適用することで、アンロールされた回数を算出することが可能。

解決法2 (前準備)

- アンロールされたコードから、演算の連鎖を表す演算木を生成する。
- 演算木は変形を施さない、固定のアルゴリズムを用いる。
  - これにより、元の構造を保った変形が可能である。
- 演算木を深さ優先探索(後順走査)し、演算の“文字列”とする
  - 左の部分木と、右の部分木の内どちらを先に辿るかは固定すればどちらでも構わない (一意性は保たれる)。
  - ロードの特性を含めることで一意性を保証可能である。
- 以降、一例として“3アンロールされたコードイメージ”を対象とし、説明を行う。

解決法3 (“リローリング可能性” その1)

- 出来た“文字列”は
  - LCS(Longest Common Subsequence)
  - 編集距離
  - 文字列長
- の3つのパラメータから類似度が判別可能である。

- 例.
  - 演算の“文字列”として “DEBFGCA”・“BFGCA”・“BCA”の3つがあったとする。
  - LCS長を最短の“文字列”を起点に算出した場合、この3つのLCS長は一致する。かつ、編集距離は異なるが同じ“文字列”への編集は挿入だけで可能である。

解決法4 (“リローリング可能性” その2)

- **定義**
  - LCS長が共通
  - 挿入だけで同じ“文字列”へと“編集”することが可能

  - このグループは“リローリング可能”であると定義する。

- **他にも同様に3つの“文字列”のグループがいくつかあった場合・・・**
  - 全て“リローリング可能”であり、かつ、誘導変数の計算を別途考慮に入れることにより、“リローリング可能性”があるかどうか判断する。

解決法5 (“ループリローリング可能”)

- 定義
  - “リローリング可能性”があった時
    - ロードとストアの内どちらか (殆どのケースでストア) が操作する配列の、位置を表す添字により、全ての“リローリング可能”なペアが同じ増分値比率 (変数の型により異なる) で配列にアクセスしていることを確認する。
  - これが出来た時、“ループリローリング可能”であると定義する。

解決法6 (ループリローリング処理 その1)

- “文字列”を 編集距離算出 のアルゴリズムの内、“挿入操作のみ”で冗長化 (共通化) する。
  - 共通式を除去された演算木を対象とする場合において、元の演算を再現する為。

- 挿入操作をする時に、一意性の保証の為、演算木の形状を保つ必要がある。
  - そこで、木の形状情報を付加して、演算木の“文字列”化を行う。

解決法7 (ループリローリング処理 その2)

- “ループリローリング可能”な時は・・・
  - ループアンローリングを施す前の演算木を、形状情報を元に再現することが可能。
  - 形状が再現出来れば、後は“ループリローリング可能”であることから、実際にループ巻き戻しが可能となる。
- ベースとなるアルゴリズムはここまで。

追加機能

- 最大の演算木以外の木“のみ”に演算が含まれていた場合。
- 巻き戻す時に 3項演算子 でマスクを作り、演算を行うか行わないかのフラグとして用いる。

- 更にこの際、静的解析が可能・かつSIMDの幅に依っては予めマスクを 不変式 としてループ外に出してやり、predicate化することも考えられる。

留意すべき点

- デメリットとして
  - ループ長
  - SIMD幅
- によって、共通式を取った場合の効果の方が大きくなる可能性が考えられる。

- それを防ぐ為に、事前見積もりをして良い方を採用する必要がある。

結論

- 従来知られている方法では、複雑なケースのループリローリングは、かなり困難。
- 本手法により・・・
  - 演算の類似性を複数グループに渡って解析する事により、“容易に”ループリローリングを適用可能である。
- と同時に、当初の以下の目的が達成される。
  - 命令キャッシュの有効利用
  - ベクトル化の促進
  - レガシーなコードの性能向上

終わり