ハッシュ関数 Luffa の性能測定と最適化

指導教員 : 市川 周一　　学籍番号 : 083738 浜田勝光

1 はじめに

ハッシュ関数には多くの応用分野があり，その 1 つが情報セキュリティである．セキュリティ応用で用いるハッシュ関数は，特に暗号学的ハッシュ関数と呼ばれている．暗号学的ハッシュ関数の 1 つに SHA（Secure Hash Algorithm）[4] がある．SHA は一群の関連したハッシュ関数であり，米国商務省国立標準技術研究所（NIST）によってアメリカ政府標準のハッシュ関数として採用されている．NIST は，2007 年に新しいハッシュ関数アルゴリズムを選定する SHA-3 コンペティション [2] を開始し，公募を行った．本研究では，コンペティションの第一次選考を通過した 14 方式の中から，Luffa [1] というハッシュ関数について性能測定と最適化を行った．

2 測定方法

測定は，CPU が AthlonXP 2600+（2.09GHz），メモリ 1GB のマシンで，OS に knoppix 6.0.1 を使用し，コンパイルには gcc ver.4.3.3 を用いた．

測定には，公募サイトにあるテストプログラム [3] の中の Monte Carlo Test を使用した．このテストでは，1024 ビットの種メッセージにハッシュ関数を適用して，新たなメッセージを生成するという作業を 10 万回行う．このとき 1000 回ごとにハッシュ値が出力され，記録される．本研究では出力 512 ビットのコードを使用し，測定の有効桁数を確保するため最内周ループの処理回数を 1000 回から 10000 回に変更して用いた．

3 性能測定と最適化

Luffa にはリファレンスコードと最適化コードがあるため，2 つの性能比較を行った．最適化コードは 32 ビット最適化版（luffa_for_32.c）を使用した．その結果，リファレンスコードが 133.1 秒，最適化コードが 53.4 秒であったため，以下の測定には最適化コードを用いた．最適化コードの実行時間プロファイルをとったところ，rnd512 という関数が処理時間の約 95%を占めていため，rnd512 の最適化を試みた．性能と同様にコードサイズも重要であるため，rnd512 のコードサイズも測定した．

表 1 は，コンパイラオプションによる性能の違いをまとめたものである．表 1 のように，-O3 が一番速くなった．-O3 にアンローリングオプションを適用し計測を行ったが，実行時間，サイズともに悪化した．以降の計測では-O3 を使用した．

表 1: 最適化オプションを使用した性能とコードサイズ

compiler option	time [sec]	size [byte]
none	53.4	8501
-O	22.9	4048
-O2	24.0	2776
-O3	20.5	4130
-O3 -funroll-loops	21.7	6407
-O3 -funroll-all-loops	21.7	6407

コンパイラオプションによるアンローリングには効果が見られないので，プログラムの書き換えによりループの展開を行った．最内周ループの展開を行い計測（innermost），ループを全て展開し計測（all_level），の 2 種類の測定を行った結果を表 2 に示す．

表 2 から明らかな通り，all_level よりも innermost の方が性能も高くコードも小さいことがわかった．

表 2: ループ展開による比較

improvement	time [sec]	size [byte]
original	20.5	4130
innermost	8.7	18745
all_level	8.9	18555

配列変数はメモリに格納され，スカラ変数はレジスタに格納される．そこで，配列変数をスカラ変数に置き換えて性能を測定した．（例：int a[2];　→　int a0 , a1;）rnd512 で書き換える対象となる変数は，uint32 t[40] と uint32 chainv[8] である．

計測は，t のみをスカラ化（t），chainv のみをスカラ化（chainv），t と chainv をスカラ化（t + chainv）の 3 種類について行った．配列 chainv[8] をスカラ変数化するには，ループ展開が必要である．この部分についてスカラ変数化を行わず，ループ展開のみを行った場合の測定を行った（unroll_only）．

表 3: スカラ変数化結果

improvement	time [sec]	size [byte]
original	20.5	4130
t	20.0	5468
chainv	8.4	4108
t + chainv	8.5	5428
unroll_only	9.8	4382

表 3 に示した通り，chainv[8] のみをスカラ変数化した方法が速度もサイズもよい結果が出ることとなった．

4 おわりに

本研究では，ハッシュ関数 Luffa の性能測定と最適化を行った．最適化コード（luffa_for_32.c）の chainv[8] のみをスカラ変数化することにより，コンパイラオプション（-O3）のみの場合と比べて，ほぼ同じコードサイズのまま実行時間を約 41%に縮めることができた．このことより今回の実験環境では，Luffa のソースは配列変数のスカラ変数化を行うことにより，最も効率的に高速化することができた．しかし，他の実験環境において計測を行ったところ，結果が変わっていたが，紙面の都合上省略することとする．さらに詳細な計測を行う必要があると考える．

参考文献

[1] Hitachi. The Hash Function Family Luffa. http://www.sdl.hitachi.co.jp/crypto/luffa/.

[2] NIST. Cryptographic Hash Algorithm Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[3] NIST. SHA-3 Reference and Optimized Implementations. http://csrc.nist.gov/groups/ST/hash/sha-3/Submission_Reqs/test_vectors.html.

[4] NIST. Secure Hash Standard (SHS). FIPS PUB 180-3, October 2008.