マルチコアプロセッサの自動最適化環境の構築

宮内 哲夫[1]　田中 清史[1]

概要：本稿では，FPGA 内にソフトプロセッサコアを生成する場合に，プロセッサ上で実行するアプリケーションプログラムに対応して最適な回路構成となるようなプロセッサの回路を自動的に生成する環境について提案する．FPGA 回路の集積度の向上により，FPGA 内にソフトプロセッサコアを構成することが可能となってきているが，利用できる資源には限りがあるため，アプリケーションプログラムで使用しない機能は削除してできるだけ小さい回路とすることが望ましい．マルチコアプロセッサのソフトプロセッサ回路を構築する際に，アプリケーションプログラムで使用する命令を解析し，実際に使用される命令から最適な回路を自動生成する方法を提案し，提案環境を使用した場合の回路の規模および速度を評価する．

キーワード：FPGA, マルチプロセッサ, ソフトプロセッサコア, 最適化

Building Automatic Optimizing Environment for Multicore processors

Miyauchi Tetsuo[1]　Kiyofumi Tanaka[1]

Abstract:
This paper introduces an automatic optimizing environment for multicore processors when soft-processor cores, which are optimized for an application program, are built in an FPGA. Due to progress of FPGA circuit integration, it becomes possible to build a soft-processor circuit in an FPGA. As the resources in an FPGA are limited, it is desirable that the size of a processor in an FPGA is as small as possible. We propose a technology for building application specific multicore processors with analyzing instructions which the application program actually uses and evaluate the size and speed of the circuit which are built in this environment.

Keywords: FPGA, multi-processor, soft-processor, optimization

1. はじめに

FPGA(Field Programmable Gate Array) の集積度向上に伴い, プロセッサを FPGA 内に書き換え可能なソフトプロセッサとして搭載することが可能になってきたが，資源の有効利用の観点から搭載するソフトプロセッサが利用する資源はできるだけ小さいことが望ましい．また，ソフトプロセッサが利用する資源を最適化することで複数のプロセッサコアをひとつの FPGA 内に搭載することが可能な場合がある．

一般に，アプリケーションプログラムではすべてのプロセッサ資源が使用される訳ではない．アプリケーションプログラムで利用される命令の種類を調べて，実際に利用される命令で使用する資源のみを構成要素とするプロセッサを構築できればアプリケーションにより最適なプロセッサを構成できることになる．また，性能向上のためにはマルチコアプロセッサ化できることが望ましい．

著者らは過去に，FPGA を対象としてアプリケーションプログラムに特化したマルチコアプロセッサを開発した [1]．4 つの対象プログラムが使用する命令とハードウェア資源に限定して実装を行うことにより，比較的小規模の FPGA（Xilinx 社 Spartan-6 LX45 [2]）上で 8 コアのマルチコアプロセッサが実現可能となった．

本研究では効率的に FPGA 資源を有効活用した開発を進めるため，上記のようなアプリケーションプログラムに特化したプロセッサ回路の最適化を自動的に行うことを目的とする．今回このようなマルチコアプロセッサの FPGA 回路を自動生成する環境の構築を行い，生成された回路の性能や FPGA の資源数の評価を行ったためその内容について説明する．

本論文は次のように構成される．2 節で今回作成したプロセッサの構成について述べる．3 節では本自動最適化環

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

境の構築の最初のステップであるアプリケーションプログラムの解析について述べ，それによる最適化されたプロセッサ回路の構成について述べる．さらに指定されたコア数のマルチプロセッサの回路を生成する手法について述べる．4 節では自動最適化環境により構築されたプロセッサについての評価を行う．最後に 5 節でまとめについて述べる．

2. プロセッサの構成

2.1 アーキテクチャ

本研究では Verilog HDL によりプロセッサの実装を行った．プロセッサのアーキテクチャは MIPS アーキテクチャ [3] を採用した．MIPS アーキテクチャのプロセッサは RISC 型のプロセッサであり，組込み用途に広く使われている．プロセッサの基本的な構成は文献 [4] に述べられている構成に準拠している．

本プロセッサの実装における特徴としては，以下の点が挙げられる．

- パイプライン：5 段のパイプラインとし，IF ステージ (命令フェッチ), ID ステージ (命令デコード), EX ステージ (命令実行), MEM ステージ (メモリアクセス), WB ステージ (レジスタへの書き戻し) からなる．
- 分岐条件の判断：分岐命令の分岐条件の判断と分岐時のプログラムカウンタの設定は ID ステージで行う．
- ディレイスロット：各分岐命令はディレイスロットを一つ持つ．すなわち，分岐の有無にかかわらず分岐命令の後の命令が一つ実行される．
- フォワーディングユニット：フォワーディングユニットにより直前の命令の実行結果を次の命令が利用する際に，パイプラインをストールさせることなく直前の命令の結果を直後の命令に引き渡すフォワーディングができるかの検出を行う．
- パイプラインストールの検出：直前の命令の実行結果を次の命令が利用する際に, フォワーディングができずパイプラインをストールさせて結果が利用できるまで待つ必要があるケースがある．例えば，ロード命令の結果を次の命令が使う場合ロード命令によりデータが利用できるのは MEM ステージの完了以降であるため，次の命令が EX ステージで利用できるようになるにはパイプラインをストールさせて MEM ステージの結果を待つ必要がある．このように，パイプラインをストールさせる必要性を検出する．
- コア ID の取得：マルチコアプロセッサ環境で，自コア ID を取得するため，MIPS 命令のコプロセッサからの読み出し命令 `mfc0` 命令を利用する．

今回，文献 [4] の命令リファレンスにある命令および，乗算命令，除算命令，レジスタ転送命令 (`mfhi`, `mflo`), および，コプロセッサからの読み出し命令 (`mfc0`) を実装した．

2.2 マルチコア化

本プロセッサの実装では上記のアーキテクチャのコアを 1 個から 8 個まで生成可能とした．各コアは，命令を格納する命令メモリ，プログラムで使用するデータを格納するデータメモリ領域をコア毎に持つ構成とした．生成されるコアの数は，後述するコンフィグレータにより選択され, 選択された数のコアの回路の Verilog によるソースコードがコンフィグレータにより生成される．

3. 自動最適化環境の構築

本研究においてマルチコアの回路を，アプリケーションプログラムに応じて自動生成するためのコンフィグレータを作成した．本コンフィグレータは文献 [5] で提案したコンフィグレータをマルチコアプロセッサ用に拡張したものである．

本コンフィグレータによるプロセッサ回路生成の処理の流れは図 1 のようになる．

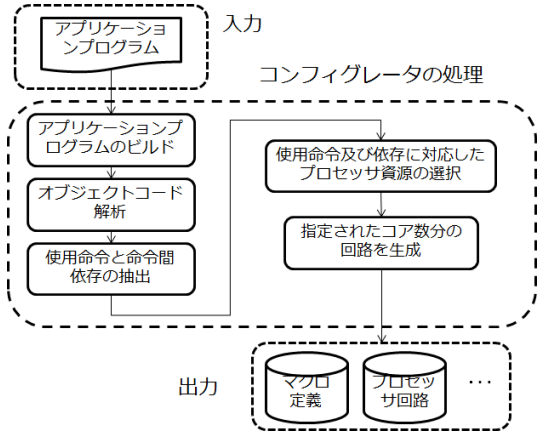

図 1 処理フロー

以下にコンフィグレータが行う処理の詳細を述べる．

3.1 コンパイラ起動処理

C 言語または MIPS のアセンブリ言語で記述されたアプリケーションのソースプログラムを入力として，MIPS 用の GCC コンパイラを起動して実行形式のプログラムを生成する．

3.2 逆アセンブル処理

生成された実行形式のプログラムを GCC のユーティリティであるダンプコマンド (`objdump`) でダンプして，アセンブリ言語のコードを生成する．

3.3 使用命令抽出処理

逆アセンブル結果から，アプリケーションプログラムで使用されている命令を抽出し，あらかじめ作成しておいた命令と使用する資源の対応に関する一覧表から，アプリ

ケーションプログラムで実際に使用されるプロセッサの資源を抽出する．

3.4 フォワーディング有無の検出

本プロセッサでは，次のような場合にフォワーディングが発生する．

- ALU 計算結果のフォワーディング (算術論理演算命令の結果を直後の算術論理演算命令が使用)
- ALU 計算結果のフォワーディング (2 命令後) (算術論理演算命令の結果を 2 命令後の算術論理演算命令が使用)
- 分岐命令へのフォワーディング (算術論理演算命令の結果を 2 命令後の分岐命令が使用)
- ストア命令へのフォワーディング (算術論理演算命令の結果を直後のストア命令が使用)
- jal,jalr からのフォワーディング (ジャンプアンドリンク命令で格納された戻りアドレスの格納レジスタ ($31) をジャンプ直後のジャンプ命令で使用)

3.5 ストール有無の検出

命令間の依存関係により，前の命令の結果を利用できるようになるまでパイプラインをストールさせる必要がある場合がある．本プロセッサでは，下記のような場合にパイプラインをストールさせる必要がある．

- ロード命令の結果を直後の算術論理演算命令が使う場合
- 算術論理演算命令，ロード命令の結果を直後の分岐命令が使用する場合
- ロード命令の結果を 2 命令後の分岐命令が使用する場合
- 算術論理演算命令の結果を直後の jr, jalr 命令が使用する場合
- ロード命令の結果を 2 命令後の jr, jalr 命令が使用する場合

3.6 最適化された回路を出力する処理

3.6.1 プロセッサ資源の分類

アプリケーションプログラムで利用される命令に従って最適なプロセッサ回路を生成するために，プロセッサの命令毎に使用されるプロセッサの資源の分類を行う．本プロセッサの実装におけるプロセッサの資源には，表 1 のようなものがある．

アプリケーションプログラムをコンパイルし，生成されたオブジェクトコードを逆アセンブルすることで，使用される命令の種類，フォワーディング，ストールの可能性がわかる．(コンパイラが出力するアセンブリコードを使用して使用命令の解析を行うことも可能であるが，本コンフィグレータはシミュレーションのためのプログラムコードを生成する機能も含んでいるため，リロケーションによってアドレスが解決されたコードを入手するために逆アセンブルを行っている．)

表 1　プロセッサの資源

資源	内容
マルチプレクサ	各ステージに複数のマルチプレクサが存在する．各マルチプレクサにおいて，そのマルチプレクサを利用するデータパス，制御パスがあるかは，命令により決まる．また，使用される命令群により，マルチプレクサでどちらのパスが選択されるかが固定的に決まる場合があり，そのような場合には常に選択される方向のみ接続しておけばよい．
PC の加算器	PC+4 と即値の加算により分岐先アドレスを算出する．
条件分岐のための比較器 (=)	`beq`, `bne` 命令のための同値比較．
条件分岐のための正判定	`bgez`, `bltz` 命令のための正判定．
条件分岐のための負判定	`blez`, `bgtz` 命令のための負判定．
乗算器	乗算命令で使用．
除算器	除算命令で使用．
フォワーディング検出ユニット	前の命令の実行結果を後続の命令が利用する場合，パイプラインをストールさせずに実行するために，実行結果のレジスタへのライトバックを待たずに次の命令にフォワーディングを行う場合がある．
ストール検出ユニット	前の命令の実行結果を後続の命令が利用する場合，命令の種類によっては，パイプラインをストールさせる必要がある場合がある．

3.6.2 マクロファイル出力処理

表 1 の各資源に対して，命令毎に使用されるデータパス，制御パスを整理して，使用する資源の一覧表をあらかじめ作成し，アプリケーションプログラムの逆アセンブル結果から，使用する命令を抽出し，実際に使用されている命令に対応して資源を選択する．

フォワーディング検出ユニットやストール検出ユニットは，アプリケーションプログラムの命令の並びによって実際に使用されるかどうかが決まるため，命令の並びを調べてそれぞれの検出ユニットが使われる可能性があるかを確認する．

以下に例を用いて資源の選択について説明する．図 2 の M_MUX2 は MEM ステージにおいて，ロードバイト命令が使用された場合にワード内のバイト位置を選択するマルチプレクサである．

このマルチプレクサはロードバイト命令が使用されない場合には使用されない．さらに，マルチプレクサ M_MUX4 の選択においても M_MUX2 からの入力線が必要なくなることになる．すなわち，ロードバイト命令を使用しないアプリケーションでは，M_MUX2 および M_MUX4 を削除

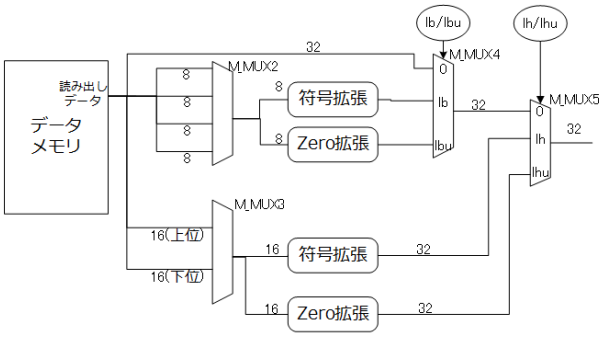

図 2 マルチプレクサの選択例 (一部データパス，制御パスは省略)

することが可能となる．

プロセッサの Verilog HDL で記述されたソースファイル中に，あらかじめ資源選択ができるようにディレクティブを記述しておく．

ディレクティブはソースファイル内に次のよう記述されている．

マルチプレクサを選択するためのマクロ例

```
/* MUX2 MEM */
‘ifdef memmux_2_nouse
;
‘elsif memmux_2_mux
MUX8_4to1 CPU_MUX8(.A(DATA_IN[31:24]),
.B(DATA_IN[23:16]), .C(DATA_IN[15:8]),
.D(DATA_IN[7:0]),
.SEL(DATA_ADDR[1:0]), .Z(DATA_IN2_8));
‘else
/* ERROR */
Never Reached;
assign DATA_IN2_8 = 32’hf0200bad;
‘endif
```

マクロ名は，

‘define ステージ_資源番号_選択

の形式で記述されており，ステージはそれぞれ，

ID ステージ	idmux
EX ステージ	exmux
MEM ステージ	memmux
WB ステージ	wbmux

で表されている．資源番号は通し番号で記述されている．

選択については，マルチプレクサが選択されるとき mux または mux3 (3 入力のとき), 資源が使用されないときは nouse, マルチプレクサの 0 側のみが選択されるときは sel0, 1 側のみが選択されるときは sel1 など選択の組み合わせが示される．マルチプレクサの一方の側しか選択されない場合には，一方の入力が出力に直接配線されればよくマルチプレクサを実装する必要がない．

アプリケーションプログラムの命令の解析により，使用される資源が確定した後次のようなマクロ定義ファイルがコンフィグレータによって出力される．

マクロ定義ファイル出力例

```
‘define idmux_1_mux3
‘define idmux_2_mux
‘define idmux_3_mux
‘define idmux_4_mux3
‘define idmux_5_mux3
...
(以下略)
```

上記のようにアプリケーションで使用されている命令に応じてマクロ定義ファイルを出力することで，適切な資源を選択する．

3.7 マルチコアプロセッサの生成

最適化された構成のプロセッサを組み合わせてマルチコアプロセッサの回路を生成する．コア数はコンフィグレータの入力として指定する．

マルチコアプロセッサ化にあたっては，コンフィグレータは Verilog HDL のプロセッサ回路のコードに対して次の処理を行う．

プロセッサコア回路の生成

プロセッサコア回路のコードに対してコア毎に変更が必要な配線名の記述などを変更してコア数分のプロセッサコア回路を生成する．

結合モジュールの生成

コンフィグレータは，プロセッサコア，命令メモリ，データメモリを結合するモジュールのコードを指定されたコア数に応じて生成する．

これらの処理によりマルチコアに対応した Verilog HDL のコードが生成される．

3.8 GUI

上記の一連の操作をコンフィグレータの GUI 画面で実行する．図 3 に GUI 画面を示す．

GUI 画面では次の操作を行うことができる．

- コア数の指定
- アプリケーションプログラムのソースコードのフォルダの指定
- アプリケーションプログラムのソースコードのファイル名の指定
- 関連するライブラリファイルの格納フォルダの指定
- 関連するライブラリファイルのファイル名の指定
- プロセッサ回路の生成 (コンフィグレーション)
- 生成されたファイルの削除

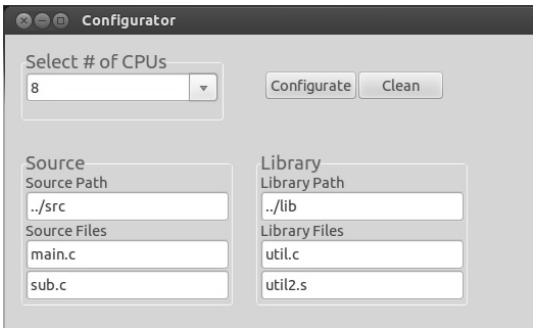

図 3 GUI

4. 評価

4.1 評価環境

3 節で述べた一連の動作を行うコンフィグレータは，Linux(Ubuntu 12.04.4_64) の上で実行される．生成された Verilog HDL による回路に対して FPGA に合成，インプリメントを行う環境としては，Xilinx 社製の ISE Design Suite 14.7 [6] を使用し, FPGA として Sprtan-6 (xc6slx75csg484-3) を対象として合成およびインプリメントを行った．

実装されたプロセッサ回路の正当性は，ISE Design Suite に付属の Behavioral Simulation tool 上でアプリケーションプログラムを動作させ，実行時の波形，データメモリに格納された実行結果を確認することで行った．

評価対象のプログラムに対応したプロセッサ回路をコンフィグレータを用いて生成し，生成された回路の規模，実行可能速度を ISE で出力されたレポートにより確認した．

4.2 アプリケーションプログラム

シングルコアプロセッサの場合に，行列積，クイックソート，SHA1 の C 言語で書かれたプログラムを用いて，コンフィグレータで各プログラムに最適化して生成された回路の評価を行った．また，マルチコアプロセッサの場合には行列積のプログラムを並列化し，そのプログラムに対して最適化して生成された回路の評価を行った．以下に各アプリケーションプログラムの詳細について述べる．

4.2.1 行列積

16 行 x 16 列の行列積を行う．行列のサイズは FPGA 内の BlockRAM 容量の範囲でさらに拡大することができるが，行列のサイズにより使用される命令の種類は変わらないため出力されるプロセッサの回路は変わらない．

マルチコアプロセッサで並列に実行するために，次のように計算をプロセッサコア毎に割り当てるようにプログラム作成を行った．行列 A と行列 B の積 $A*B$ を求める場合，行列 B を列方向にコア数と同数に分割し，それぞれのコアの演算に割り当てることで並列に演算を行う．

行列積 (マルチコアプロセッサ)

```
columns = N/N_CORE;
for (i = 0; i < n; i++) {
  for (j = (columns * my_rank);
    j < (columns * (my_rank + 1)); j++) {
    for (k = 0; k < n; k++) {
      matrixC[i][j] +=
      matrixA[i][k]*matrixB[k][j];
    }
  }
}
```

N: 行列のサイズ
N_CORE: コア数
my_rank : 自コアのコア ID

4.2.2 クイックソート

クイックソートアルゴリズムにより整数の整列を行う．クイックソートのプログラムは [7] を参考に作成した．あらかじめ 100 個のランダムなデータを作成しておき，それらが期待通りにソートされたことを確認した．

4.2.3 SHA1

SHA1 は一般に広く使われているハッシュアルゴリズムである．プログラムのソースは [8] を参考にし，入力データに対して期待どおりの SHA1 の結果が出力されていることを確認した．

4.3 生成回路の評価

FPGA (xc6slx75csg484-3) を対象としてインプリメントを行った．インプリメントされた結果のレポートは，表 2, 表 3, 表 4 のようになる．比較のため，各アプリケーションプログラムに対して自動最適化を行った場合と行わなかった場合の (全ての命令実行が可能な) フル構成のプロセッサ回路のサイズと動作性能についての評価を行った．

4.3.1 シングルコアプロセッサ

インプリメント後のレポートによるそれぞれのアプリケーションに対する回路の規模，性能は表 2 のようになる．

表 2 シングルコアプロセッサインプリメント結果

	行列積	Qsort	SHA1	フル
Register	662	600	603	710
LUT	1248	1431	1459	9405
Slice	419	471	515	3164
RAMB16BWER	2	2	2	2
RAMB8BWER	4	4	4	4
DSP48A1	8	-	-	8
Min period(ns)	11.797	11.829	11.81	13.684
Max freq.(MHz)	84.767	84.538	84.674	73.078

最適化された回路はフル構成に対して，レジスタ要素,

LUT がそれぞれ最大 15%, 86%削減され, 最大動作周波数が最大 15%向上した.

4.3.2 マルチコアプロセッサ

1 コアから 8 コアの場合に，本環境で自動的に最適化した結果と，フル実装のプロセッサでのインプリメント結果を表 3, 表 4 に示す.

表 3 マルチコアプロセッサインプリメント結果 (行列積)

コア数	1†		2	
	最適化	フル	最適化	フル
Register	662	710	1316	1575
LUT	1248	9405	2461	18856
Slice	419	3164	782	6182
RAMB16BWER	2	2	4	4
RAMB8BWER	4	4	8	8
DSP48A1	8	8	16	16
Min period (ns)	11.797	13.684	12.866	15.809
Max freq. (MHz)	84.767	73.078	77.724	63.255

† シングルコアプロセッサの結果の表 2 から比較のため再掲.

表 4 マルチコアプロセッサインプリメント結果 (行列積)

コア数	4		8		利用
	最適化	フル	最適化	フル	可能数
Register	2595	2572	5227	5643	93296
LUT	5089	34077	10285	85653†	46648
Slice	1750	10388	3409	-	11662
RAMB16BWER	8	8	16	-	172
RAMB8BWER	16	16	32	-	344
DSP48A1	32	32	64	-	132
Min period (ns)	15.854	19.301	14.784	N/A	-
Max freq. (MHz)	63.076	51.811	67.641	N/A	-

† 利用可能数を超えたためインプリメントできない.

結果から，コア数に比例して Register 数，LUT 数が増加している．Slice 数についてもコア数が増えるにつれて増加している．また，フル構成のプロセッサ回路については，マルチプレクサ，乗算器，除算器がすべて構成要素に入るため，最適化されたプロセッサに対して，LUT 数, Slice 数が数倍になっている．特に，8 コアの場合，フル実装のプロセッサでは，今回対象とした FPGA (xc6slx75csg484-3) の場合には，ひとつの FPGA 内にインプリメントすることができないが，最適化することでインプリメント可能となることがわかる.

アプリケーションプログラムで使用される命令を解析して，その命令に対応した最適化を手作業で行うことには大きな工数を必要とするが，本環境では，アプリケーションプログラムと，コア数の指定により，自動的に最適な回路を生成することができ，手作業で行う場合より効率的に行うことができる.

5. まとめと今後の課題

本稿ではプロセッサ回路の構成について述べ，続いて自動最適化環境の構築について説明した．FPGA 上にソフトプロセッサを構成する際にアプリケーションプログラムを解析して，そのアプリケーションプログラムを実行するために必要とされる資源のみを搭載するプロセッサを構成することで，利用される FPGA の資源を最適に利用できる．特に，マルチコアプロセッサの構成では最適化により単一コアのサイズが大幅 (数分の 1) に削減されるため実装可能なコア数が増加する．このような構成をアプリケーションプログラムを解析して手作業で行うのは効率的でないが，提案したコンフィグレーション環境を用いることでアプリケーションに応じて，自動的に最適なプロセッサコア回路が構成できる.

今後の課題として,

(1) キャッシュメモリを導入し，アプリケーションに対応してキャッシュメモリの構成を最適化すること
(2) 非対象マルチコアの生成を対象とし，コア毎に資源を最適化すること

が挙げられる.

謝辞

本研究の一部は JSPS 科研費 15K00073 の助成を受けて行われた.

参考文献

[1] Fengxiang Xie, 田中清史: JAIST23-Pro : FPGA 用マルチコアプロセッサの設計，情報処理学会研究報告，ARC, Vol.2014-ARC-208, No.7, 情報処理学会電子図書館, 2014.
[2] http://japan.xilinx.com/products/silicon-devices/fpga/spartan-6/lx.html (2015 年 9 月 13 日閲覧)
[3] MIPS®Architecture For Programmers Volume II-A: The MIPS32®Instruction Set
[4] パターソン&ヘネシー：コンピュータの構成と設計 第 4 版 日経 BP 社 2013.
[5] 宮内哲夫, 田中清史 : FPGA 用ソフトプロセッサ向け自動最適化コンフィグレータ, 情報処理学会第 77 回全国大会講演論文集 (1), pp.23-24, 2015.
[6] Plan Ahead: http://japan.xilinx.com/tools/planahead.htm (2015 年 9 月 13 日閲覧)
[7] 第 2 回 ARC/CPSY/RECONF 高性能コンピュータシステム設計コンテスト　アプリケーションプログラム: http://aquila.is.utsunomiya-u.ac.jp/contest/ (2015 年 9 月 13 日閲覧)
[8] SHA1: http://tools.ietf.org/html/rfc3174 (2015 年 9 月 13 日閲覧)