Fortranベース高位合成ツールFortRockの開発

山下 貴大[1]　五十嵐 雄太[1]　中條 拓伯[1]

概要：近年，集積回路の集積度の向上に伴い，FPGA の性能が飛躍的に向上してきた． それに伴い，FPGA を用いたハードウェア・アクセラレーションが注目されてきた．また，科学技術計算では Fortran が広く利用されてきたが，Fortran で記述されたライブラリ， パッケージは今もなお改良が行われている． この莫大な Fortran 資産をハードウェアを用いたアクセラレーションによって再利用したいというニーズが存在するが，FPGA に実装する場合には，ハードウェアの知識が必要となる． これらを解決するものとして高位合成ツールが存在するが，Fortran から Verilog HDL に 自動的に変換する手法が現在では存在していない．そこで本研究では，Fortran ベースの高位合成ツール FortRock の開発を行うとともに， その性能を評価する．また，DFG(Data Flow Graph) を用いたスケジューリングなどによる最適化を行い，その性能評価を行うことにより，高性能計算のソフトウェア資産をハードウェア化するための可能性を示す．

キーワード：高位合成，FPGA，Fortran，アクセラレーション，高性能計算

Development of Fortran base high level synthesis tool FortRock

Takahiro Yamashita[1]　Yuta Ikarashi[1]　Hironori Nakajo[1]

Abstract: In recent years, along wint improvement of VLSI, performance of VLSI has been improved drastically. Also, hardware acceralation using FPGA has attracted hardware designers. In addition, though Fortran has been used, in High Performance Computing(HPC) as for a library, or a package which is described in Fortran, improvement has been still needed. Reusage of such vast Fortran assets by acceleration using hardware has been still expected, however knowledge of the hardware is required for FPGA in implementation. Though high level synthesis tools which translate exist to solve such problems, but technique to convert into Verilog HDL from Fortran does not exist now. Therefore, in this study, we have developed a Fortran-based high level synthesis tool called FortRock. In addition, We describe possibility to build software assets of the HPC into hardware with optimization by scheduling using DFG(Data Flow Graph).

Keywords: High Level Synthesis, FPGA, Fortran, Acceleration, High Performance Computing

1. はじめに

近年の半導体の集積度の向上に伴い，FPGA(Field Programmable GateArray) の性能が飛躍的に向上し，搭載できるゲート量の増加とともに，消費電力においても大幅な改良が施されるようになった．

高性能計算 (HPC: High Peforance Computing) のための大規模計算機発展においては，複数のコアを持つ多数のプロセッサノードを並列に接続した構成に加え，GPU を汎用計算の高速化に利用したハードウェア・アクセラレーションが主流となり，TOP500 の上位を占めるようになっている [1].

高性能計算の性能指標として，処理能力の他に低消費電力性が挙げられ，いかに少ない電力で高い計算性能を発揮できるかを競い合うといった方向もあり [2] ，電力対性能比の優れた FPGA が，アクセラレータとして注目されるようになってきた．

その流れを受け，Microsoft や Google，Baidu などのいくつかの企業がデータセンタに設置する大規模計算機のノード内に FPGA を採用する [3][5][4] など，様々な分野で FPGA の利用が注目され始めている．また，Intel が FPGA ベンダ大手のアルテラを買収し [7]，Xeon に FPGA を組み込んだ新たな SoC (System on Chip) を発表 [6] するなど，今後のさらなる発展が期待されている．

FPGA は，単に電力対性能比だけでなく，その容量が許す限り，自由に設計することが可能であるため，汎用プロ

[1] 東京農工大学
Tokyo University of Agriculture and Technology

セッサやメニーコアプロセッサをはるかに超えるような大規模並列性を活かした処理が可能となる．これに伴い，この大規模並列性を活かした HPC における FPGA を用いたハードウェア・アクセラレーションにおいて，その開発環境の拡充が注目されるようになった．

これまで HPC では，歴史的に科学技術計算に適した Fortran 言語を中心とし，高級言語により様々なライブラリ，パッケージが開発されてきた．これに伴い，Fortran や C 言語をソースコードとする膨大なプログラム資産が蓄積されており，今もなお改良が加えられ，利用され続けている．

FPGA 上における回路を設計する場合には，ハードウェア記述言語 (HDL) を用いた設計がこれまで一般的であった．しかしながら，HDL による設計・実装の後，十分な回路性能を得るためには，ディジタル設計を中心にハードウェアに関する知識が必要となり，このことがソフトウェア開発者にとって敷居が高くなっている要因となっている．

HPC においても，FPGA を用いたハードウェア・アクセラレーションが行われてきたが [8][9]，その回路の開発は HDL 熟練者の手によるものであり，HPC のアルゴリズム開発者が直接記述できれば，さらに開発効率は高まるものと考えられる．

HDL による RTL 設計とともに，近年抽象度の高い高級言語により記述したアルゴリズムをハードウェア化する高位合成技術が注目されている．これにより，C 言語や Java 言語などによる FPGA の開発が可能になった．

C 言語ベースの高位合成ツールでは，SystemC，ImpulseC，CyberWorkBench や LegUp などが挙げられる．Java 言語ベースの高位合成ツールでは，JHDL，Lime，JavaRock，JavaRock-Thrash ，Synthesijer などがある．

これら高位合成ツールにより，ソフトウェア技術者に対して FPGA 設計に対する学習コストを下げるとともに，ハードウェア・ソフトウェア協調設計の可能性が広がることとなった．これにより，ソフトウェア技術者はハードウェアに関する知識が乏しくても，アルゴリズムのハードウェア化を支援することができ，ハードウェア技術者に頼らずとも，独自にハードウェア・アクセラレーションシステムを設計することが可能となる．

そこで我々は，膨大な HPC 資産のある Fortran に着目し，先行研究において，Java ベースの高位合成ツール JavaRock-Thrash[10][11] のソースとなるように，Fortran のプログラムを Java に変換するシステム F2JRT(Fortran to JavaRock-Thrash) を開発し，評価を行った [12]．しかしながら，F2JRT には出力される回路の性能において，種々の問題点が生じた．

そこで，本研究ではそれらの問題点を克服し，Fortran から他の高級言語を経由せず, 直接 HDL に変換する FortRock を開発した．

本論文では，FortRock の実装時の工夫や回路出力の特徴などを説明するとともに，出力された回路の性能評価を行い，FortRock の現状と今後について報告する．2 章では関連研究である LLVM や数値流体力学 (CFD: Computational Fluid Dynamics) などについて説明する．3 章では FortRock の設計方針やスケジューリング手法などの内部の処理を説明するとともに，実際に Fortran プログラムを入力し，その回路の性能を比較する．

2. 関連研究

2.1 Fortran を取り巻く高位合成

FortRock では，LLVM IR(LLVM Intermediate Representation) を中間言語表現に用いることで，LLVM IR レベルでの最適化を可能にしている．LLVM とは，イリノイ大学で 2000 年に開発が開始されたコンパイラ基盤である．LLVM の成果を FortRock の開発で利用することにより，中間言語表現レベルでの最適化が行えるなど，実装面で様々な利益を得ることができる．

高位合成技術に LLVM を利用しているものとして LegUp が挙げられるが，これは C 言語ベースの高位合成技術であるため，Fortran ベースの高位合成技術としては FortRock のみが利用している．

Fortran ベースの高位合成の手法として，F2c を用いたものが想定される．F2c は，Fortran で記述されたプログラムを C 言語のプログラムに変換することを可能にするソフトウェアである．これを用いて，Fortran のプログラムを一旦 C 言語に変換し，CyberWorkBench 等の C 言語ベースの高位合成技術を利用して Fortran ベースの高位合成が行えると考えられるが，F2c は FORTRAN 77 までしか対応しておらず，かつ開発が終了しているため，近年の科学技術計算における高位合成の手法としては不適であると考えられる．したがって，FortRock による Fortran ベースの高位合成技術に有用性があると考えられる．

2.2 High Performance Computing における FPGA の利用

近年，High Performance Computing で FPGA のもつ大規模並列性を活かしたハードウェア・アクセラレーションが注目されている．[13] これは，CFD(Computer Fluid Dynamics) プログラムを FPGA を用いてハードウェア化したもので，独立行政法人宇宙航空研究開発機構 (JAXA) が開発した流体解析ソフトウェア UPACS[14] を対象に検証を行って得られたものである．プログラムを処理ごとに分割し，8 つの処理に対してハードウェア化を行った．その結果以下の 8 項目の課題が得られた．

(1) 大量のデータ入力
(2) 回路規模と実装
(3) メモリスケジューラの実装

(4) ホスト計算機とのインタフェース
(5) 浮動小数点演算器のチューニング
(6) CFD 研究者による Fortran プログラムの FPGA 回路化
(7) 実行計算機を意識したソースプログラムのチューニング
(8) IP コア化されていない演算処理の実装

(1)-(5) については，ハードウェア開発に関する一般的な問題点である．(6)-(8) については，プログラムで記述されたアルゴリズムのハードウェア化にあたり，回路設計の知識が必要とされるため，ソフトウェア技術者にとっては敷居が高くなっているという問題点である．CFD におけるハードウェア・アクセラレーションに関する研究として，JAXA が開発した流体解析ソフトウェア FaSTAR[15] のハードウェア・アクセラレーションの研究 [9] がある．この研究は前述した UPACS と同様に，FPGA を用いてハードウェア・アクセラレーションを行ったものである．前述の課題のうち「メモリスケジューラの実装」を行っている．FaSTAR のプログラムを解析したところ，メモリの同一アドレスに対して連続して複数回アクセスするため，ストールが多発することが分かった．これを解決するために，Out-Of-Order 実行をする機構を作成することで，ストール回数を減らすことに成功した．最終的に CPU での実行速度と比較して FPGA の実行速度が 2.35 倍速いという結果が得られた．

これらはいずれも，Fortran 資産を FPGA でハードウェア・アクセラレーションすることによって再利用可能であるということを表している．しかし，そのためには回路設計に関する知識が必要という課題が存在する．

2.3 性能向上への方向性

FortRock では先行研究として，F2JRT(Fortranto JavaRock-Thrash) を開発した．しかし，F2JRT では最終的に JavaRock-Thrash で出力される回路の性能を出すという点においていくつかの問題点が生じた．まず，goto 文などの Java 言語がサポートしていない文法の変換が困難なことである．JavaRock-Thrash は Java 言語に多少の制限を加えるだけで使用できるが，Java 言語がサポートしていない文法は使用することができない．

また，JavaRock-Thrash で回路の性能を出すためには，単純に Java 言語でプログラムを記述するのではなく，JavaRock-Thrash で回路の性能を出すための記述が必要となる．したがって，F2JRT が回路の性能を出すためには，Fortran のプログラムを単純に一対一で変換するのではなく，入力されたプログラムを解析した上で，JavaRock-Thrash に適した Java プログラムを出力する必要がある．

しかしこれらの問題は，Java 言語を経由しないで Fortran から直接 Verilog HDL に変換することで解決されると考えられる．そこで本研究では Fortran から LLVM IR を経由して最終的に Verilog HDL を出力する FortRock の開発を行った．

3. 高位合成ツール FortRock

本章では，Fortran から Verilog HDL を出力する高位合成ツール，FortRock について説明する．FortRock は，Fortran を入力言語とし，Verilog HDL を出力言語とする高位合成ツールである．Fortran から Verilog HDL に変換し，Fortran プログラムを FPGA に実装可能とすることで，Fortran 資産を有効活用することが可能である．FortRock では，Fortran 資産の再利用にとどまらず，Fortran を用いた新規のプログラム開発でもハードウェア・アクセラレーションをすることを想定して設計・開発を行う．

3.1 FortRock の設計方針

FortRock は，HPC などの計算速度が求められる計算や，低消費電力が求められる計算を行うことを想定している．FPGA ではゲート数が回路の製造コストに比例するため，ゲート数を最小とすることを目標とし実装を行う．

3.2 FortRock のコンパイルフロー

FortRock では次のようなコンパイルフローで，Fortran で記述されたソースコードを Verilog HDL で記述された回路に出力する．

(1) xml 形式の設定ファイルを読み込む
(2) Fortran プログラムを gfortran でパース
(3) パースした情報を DragonEgg を用いて LLVM IR として出力
(4) LLVM IR を FortRock Core に入力
(5) FortRock Core が LLVM IR の各命令を DFG に変換
(6) 変換した DFG のデータ構造に変換
(7) DFG を用いてスケジューリング
(8) スケジューリング後の DFG を Verilog HDL で出力

FortRock ではまず，xml 形式で記述された設定ファイルを読み込む．設定ファイルには生成される回路で使用される IP コアの情報などに関する情報が含まれる．IP コアの入出力の数や入出力信号名，レイテンシ，モジュール名などが情報として与えられる．次に，GCC(GNU Compiler Collection) の Fortran コンパイラである gfortran を用いて，入力された Fortran プログラムをパースする．LLVM プロジェクトの DragonEgg を用いることで，gfortran のパース結果を LLVM IR に出力する．次に，FortRock Core に先ほどの結果である LLVM IR を入力として与える．FortRock Core は LLVM IR の各命令を DFG に変換する．その後，その DFG を用いて命令の並列実行などのスケジューリングを行う．最後に，スケジューリング後の DFG のを Verilog HDL で出力することで Fortran プログラムか

ら Verilog HDL への変換を完了する．

3.3 スケジューリング

FortRock ではまず，LLVM IR レベルでの最適化を行う．その後，先述したとおりに，LLVM IR の命令列から DFG(Data Flow Graph) のデータ構造にパースする．その後，DFG レベルでの命令の並列化などのスケジューリングを行うことで，最終的な回路のステートマシンを得る．

FortRock では，ステートとステップ呼ばれる 2 つの状態によってステートマシンを構成する．ステートとは，入力プログラムを LLVM IR に変換した段階での CFG(Control Flow Graph) に対応する．但し，ステート 0 はリセット状態として予約済みである．ステップとは，1 クロックで実行される処理の状態である．演算器のレジスタへ値を代入や，ワイヤ接続，比較命令や function 文の呼び出しなどを行う．

3.4 出力される回路

FortRock では，入力として与えられた Fortran プログラムを最終的に Verilog HDL の回路として出力する．出力される回路では，演算で使用されるレジスタやワイヤ，IP コアなどが自動的に宣言される．

FortRock には次のような信号が用意されている．

- clk
- res
- req
- ce
- fin
- Input/Output

clk 信号は，親モジュールから提供されるクロックで，回路全体で共有されると想定している．このクロック信号に基づいてモジュール内のステートマシンが処理を実行する．

res 信号はモジュールに対して正論理でリセットを行う信号である．この信号の入力が High になると，ステートマシンの状態にかかわらずステート及びステップがリクエスト待機状態となり，ステートマシンに関わるレジスタが常にリセットされ続ける．

req 信号は，モジュールに対して親モジュールから処理の実行を開始を要求するための信号である．この信号の入力が High になった場合，クロックの立ち上がりに同期して，ステートマシンの状態がステートマシンの最初の状態に遷移する．以降はステートマシンの処理にしたがって，ステートマシンが終了状態になるまで実行される．なお，ステートマシンが終了状態になった場合は，fin 信号を High にして直ちにステートマシンのリクエスト待機状態に遷移する．したがって，親モジュールは fin 信号が High になる 1 クロックの間に出力を受け取らなければ，その値が有効であることが保証されない．但し，新たにリクエストがない場合は，ステートマシンに関するレジスタのみがリセットされ続けるため，次のリクエストを発行するまでは，前回の出力が保持される．

ce 信号は Clock Enable 信号であり，この信号が High でない場合はモジュール内のステートマシンは動作しない．この場合は親モジュールからのリセット信号によるリセットのみ動作する．

fin 信号は前述したとおり，ステートマシンが終了状態になった場合に 1 クロックだけ High になる信号である．親モジュールでは，req 信号によって処理のリクエストを行ったあと，この fin 信号を監視し，High になったクロックに同期してモジュールから出力される値を読み取る必要がある．

最後に入出力信号が与えられる．FortRock で出力されたモジュールを使用する場合には，これに対して入力を与えリクエストを行う．

これらの信号線は FortRock で自動的にモジュール内に宣言される．しかし，名前の衝突を避けるために，それぞれの信号線名には prefix を与える．これによって，入出力に fin などの変数名を与えた Fortran プログラムがあった場合にも，前述の fin 信号と衝突することはない．モジュール内部で使用される，テンプレートレジスタについても同様である．

以上の特徴から，FortRock を用いた場合，以下の 3 つの設計手法を取ることが出きる．

- 回路全体を Fortran で記述
- 回路の一部の処理を Fortran で記述
- 回路や IP コアを呼び出すトップモジュールを Fortran で記述

回路全体を Fortran で記述する場合は，トップモジュールから子モジュールまでを Fortran で記述し，それぞれの Fortran プログラムを FortRock の入力とする．これらによって生成される回路は，FortRock によって自動的に接続される．

回路の一部の処理を Fortran で記述することも可能である．この場合トップモジュールなどの階層が上の回路のみを Verilog HDL で記述し，回路の一部の処理を SUBROUTINE として記述し，それを FortRock によって回路化する．Verilog HDL で記述された回路で，この FortRock で作成された回路に入出力を与え req 信号を発行することにより，回路を利用することができる．

FortRock 入出力にはいくつかの制限が存在する．まず，入力プログラムの返り値が一つであることを想定しているので，モジュールの出力は 1 つに限定する．出力は Fortran の SUBROUTINE で最後の引数として与えられたものを自動的にモジュールの出力とする．入力の数に制限はない．

```
SUBROUTINE LCM(I, J, ret_lcm)
INTEGER I, J, IR1, IR2, IR, ret_lcm

IF(I < J) THEN
    IR1 = J
    IR2 = I
ELSE
    IR1 = I
    IR2 = J
ENDIF

IR = IR1 - (IR1/IR2) * IR2

DO WHILE(IR>0)
    IR1 = IR2
    IR2 = IR
    IR = IR1 - (IR1/IR2) * IR2
ENDDO

ret_lcm = I*J/IR2

RETURN
END
```

図 1 最小公倍数例 (Fortran ソースコード)
Fig. 1 LCM program example (Fortran source code)

表 1 評価結果
Table 1 Results of the study

評価項目	FortRock (old)	FortRock
動作周波数	15.204MHz	392.758MHz
Register	167 (0%)	6916 (5%)
LUTs	3626 (5%)	3901 (6%)

3.5 性能評価

FortRock の性能を評価するために，以前の FortRock[12] の性能評価を行った条件と同じ条件で回路を出力した．入力プログラムには Fortran で記述された最小公倍数を求めるプログラムを使用した．使用したプログラムを図 1 に示した．

このプログラムでは，入力 I, J を受け取り，その 2 つの入力の最小公倍数を ret_lcm に格納することで返り値とする．このプログラムを現状の FortRock に入力したところ図 2 のような Verilog HDL の回路を得られた．(但し，プログラムの一部のみを抜粋している．)

図 2 は FortRock が出力した回路の一部である．図 2 から，FortRock の出力した回路がステートマシンを構成していることが確認できる．

この現在の FortRock によって得られた回路と，以前の FortRock で得られた回路の性能比較を行った．その結果，表 1 のような結果が得られた．

表 1 から，現在の FortRock は以前に比べて，回路の動作周波数が飛躍的に向上していることが確認できる．これは，以前の FortRock と異なる以下の点によって起こっていると考えられる．

```
always @(posedge i_clk)
  begin
    if (i_res_p ==  p_TRUE)
      begin
        o_fin_p <= p_FALSE;
        r_sys_current_state <= p_ZERO;
        r_sys_step_name <= p_ZERO;
      end
    else if (i_ce_p == p_TRUE)
      begin
        case (r_sys_current_state)
          3'h0:
            begin
              o_fin_p <= p_FALSE;
              r_sys_step_name <= p_ZERO;
              if (i_req_p)
                begin
                  r_sys_current_state <= 3'h1;
                end
            end
          3'h1:
            begin
              r_sys_step_name <= r_sys_step_name + 1'h1;
              case (r_sys_step_name)
                3'h0:
                  begin
                    r_tmp <= i_i;
                  end
                3'h1:
                  begin
                    r_tmp1 <= i_j;
                  end
```

図 2 FortRock の出力 ステートマシン
Fig. 2 Converted example (Verilog HDL code)

- IP コアへの対応
- 1 クロックで実行する処理の単純化

現状の FortRock は，以前のものと比較して IP コアに対応しているという点で，出力される回路の性能が向上していると考えられる．

以前の FortRock では，CFG 内における全ての計算を逐次実行していた．これにより，回路のクリティカル・パスの長さが長くなり，結果的に回路全体の動作が遅くなっていると考えられる．これに対して現在の FortRock では，ステートマシンで実行するのは IP コアやその他モジュールの入出力信号に対する値の代入や，単純な条件演算や function 文の呼び出しなどを同一クロックで並列実行する．以上の違いから，現在の FortRock の出力する回路の性能が以前のものよりも高性能になったと考えられる．

前回の評価条件と合わせるために，浮動小数点演算などは行わなかったが，現状の FortRock では浮動小数点演算も可能となっている．このことから，単純に出力される回路の性能だけでなく，出力することが可能な回路の機能も増えている．

3.6 FortRock の現状と今後

現状の FortRock では，様々な制限の上で動作することが保証されている．現状の制限は次のとおりである．

- レジスタが再利用されることはない
- モジュールや IP コアは 2 入力 1 出力である

レジスタが再利用されることがないという制限は，DragonEgg の出力が SSA(Static Single Assignement) であることに起因する．LLVM では，LLVM IR レベルでの最適化を容易にするために，LLVM IR レベルでは SSA で出力している．これにより，各レジスタへの命令はプログラム中

で 1 つの命令に限定される．しかし，実際のプログラムでは変数の使い回しが行われているので，スケジューリングに影響の出ない範囲ではレジスタの共有が可能であると考えられる．レジスタの共有によって回路で使用するレジスタの総数が削減されれば，回路規模の縮小につながるため，出力される回路の性能を向上させることができると考えられる．しかし，Verilog HDL レベルでのレジスタの共有が必ずしも回路の性能を向上させるとは限らないことが分かっている [16]．したがって，どの条件でレジスタを共有すれば回路の性能を向上させるかを明らかにすると共に，その条件でスケジューリングに影響を与えない範囲でレジスタを共有する機能を実装する必要がある．

現状の FortRock では，モジュールや回路で使用する IP コアの入出力が 2 入力 1 出力の形式に制限されている．これは，IP コアの入出力が基本的に 2 入力 1 出力であるため，それ以外に対応する必要性があまり生じなかったことに起因する．しかし，今後 HPC で利用することを前提により実践的なプログラムを用いた性能評価を行うことを想定して，2 入力 1 出力にとらわれない入出力の与え方を可能とする機能の実装が必要である．例えば，グローバル変数による値の参照や，独自関数 (SUBROUTINE) の 2 入力 1 出力以外の入出力の与え方などに対応する必要がある．

4. まとめ

本論文では，Fortran ベースの高位合成ツール FortRock の現状と今後について説明した．FortRock は LLVM IR を中間表現で用い，中間表現レベルや DFGd レベルで様々な最適化を行う．FortRock はフロントエンドに gfortran 及び DragonEgg を用いて，中間表現から Verilog HDL のコードを出力する部分は LLVM のバックエンドとして実装した．FortRock の現状は，様々な制限がある状態で動作することが確認できた．今後の課題としては HPC で利用することを前提として，より良い計算性能が得られるようにスケジューリングにさらなる最適化を施すなどの必要がある．

謝辞 本研究の一部は，文部科学省特別経費「持続可能社会にむけた知的情報空間技 術の創出」及び JSPS 科研費基盤研究 (C) 25330067 による支援を得た．ここに記して感謝する．

参考文献

[1] TOP500: http://www.top500.org
[2] THE GREEN500: http://www.green500.org/
[3] Catapult: http://research.microsoft.com/en-us/projects/catapult/
[4] Altera and Baidu Collaborate on FPGA-based Acceleration for Cloud Data Centers: http://newsroom.altera.com/press-releases/altera-baidu-fpga-cloud-data-centers.htm
[5] Dennis Abts and John Kim: High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities (2011)
[6] Diane Bryant: Disrupting the Data Center to Create the Digital Services Economy, https://communities.intel.com/community/itpeernetwork/datastack/blog/2014/06/18/disrupting-the-data-center-to-create-the-digital-services-economy (2014.6).
[7] Intel to Acquire Altera: http://www.intc.com/releasedetail.cfm?ReleaseID=915707&Releases
[8] 飯塚尊則, 佐野健太郎, 山本悟：FPGA による数値流体力学専用計算機, 電子情報通信学会技術研究報告. RECONF, リコンフィギャラブルシステム, Vol.106, No.50, pp.13-18 (2006.5).
[9] 赤嶺公之, 田舎片健太, 長名保範, 藤田直行, 天野英晴：FaSTAR における流束の面積分計算高速化のための Out-Of-Order 機構 (科学技術計算)，電子情報通信学会技術研究報告. RECONF, リコンフィギャラブルシステム, Vol.111, No.31, pp.73-78 (2011.5).
[10] 小池恵介，三好健文，船田悟史，中條拓伯: JavaRock-Thrash の実装，組込みシステムシンポジウム (ESS2013) 論文集, pp.41-48, (2013.10).
[11] 小池 恵介，三好 健文，五十嵐 雄太，船田 悟史，中條 拓伯：“Java 言語ベース高位合成ツールによるアクセラレータ開発環境”，電子情報通信学会論文誌 D，Vol.J98-D，No.3，pp.373-383 (2015.03).
[12] 山下貴大，五十嵐雄太，中條拓伯: Fortran による高性能計算のハードウェア化と高位合成ツール FortRock 組み込みシンポジウム (ESS2014) 論文集，pp.90-95，(2014.10).
[13] 藤田直行: 実用 CFD コードの FPGA 回路化における技術課題の実験的検討, 第 23 回 数値流体力学シンポジウム講演論文集, E2-3 (2009).
[14] 山本一臣：並列計算 CFD プラットフォーム UPACS について，航空宇宙数値シミュレーション技術シンポジウム’99 講演集 (1999).
[15] 橋本敦, 村上桂一, 青山剛史, 菱田学, 大野真司, 坂下雅秀, ラフールパウルス，佐藤幸男：高速流体ソルバ FaSTAR の開発, 第 42 回流体力学講演会/航空宇宙数値シミュレーション技術シンポジウム 2010 (2010).
[16] Matsuba, T. Grad. Sch. of Inf. Sci., Nagoya Univ., Nagoya, Japan Hara, Y. ; Tomiyama, H. ; Honda, S. ; Takada, H. “Aggressive Register Unsharing Based on SSA Transformation for Clock Enhancement in High-Level Synthesis” Internal Symposium on Electronics Design, Test & Application (DELTA’10), pp.87-92, Hochi Minh City, Vietnam, Jan 2010.