Vol.2017-ARC-225 No.18
Vol.2017-SLDM-179 No.18
Vol.2017-EMB-44 No.18
2017/3/9

協調設計環境SWORDSフレームワークのツールチェーン実装および設計空間探索への適用

谷　祐輔[1,a)]　高瀬　英希[1,b)]　高木　一義[1]　高木　直史[1]

概要：我々は，プログラマブル SoC のためのソフトウェア志向の協調設計環境である SWORDS フレームワークの研究に取り組んでいる．本稿では，まず，Xilinx 社 Zynq を対象とした SWORDS フレームワークのツールチェーン実装を示す．設計者は，実装されたツールチェーンを活用することにより，JSON 形式による構成情報記述の変更のみで，プログラマブル SoC 上に合成される SW/HW 協調システムの様々な構成を実現できるようになる．さらに本稿では，プログラマブル SoC のための設計空間探索手法を提案する．提案手法は，設計記述から対応する探索空間を指定することで，システム構成の候補に対応する構成情報記述の生成，および，設計記述への高位合成指示子の追加を行う．そして，SWORDS フレームワークによる合成結果に対して機械学習アルゴリズムを適用し，性能および FPGA の各種リソース量の予測モデルを生成する．これらの予測モデルを探索アルゴリズムの評価関数として活用することで，設計空間探索を実現する．行列積演算を対象とした適用事例によって，実装成果および提案手法の有用性を示す．

1. はじめに

近年の組込みシステムでは，プロセッサおよび FPGA が専用の通信バスで密結合された SoC であるプログラマブル SoC の活用が注目されている．本 SoC では，柔軟性の必要な逐次処理はプロセッサ上のソフトウェア（SW）で，負荷の大きい並列処理は FPGA 上のハードウェア（HW）で実行できる．これにより，双方の機能と利点を活かした SW/HW 協調システムを実現できるようになる．

プログラマブル SoC を用いて高性能な組込みシステムを設計するためには，SW と HW の双方に関する深い開発知識が必要となる．また，プロセッサと FPGA を接続する通信バスに関する知識も必要となる．このため，プログラマブル SoC 上で高性能な組込みシステムを設計することの難易度が高くなっている．さらに，設計対象の大規模化および複雑化に伴い，選択可能なシステム構成の候補が増大する．このため，要求を満たす高性能なシステム構成の候補を発見することが困難な課題となる．ここで，高性能なシステム構成とは，システム全体の実行時間が短く FPGA の各種リソースの使用量が少ないものであることを指す．

我々は，プログラマブル SoC のための協調設計環境である SWORDS（SoftWare ORiented Design and Synthesis）フレームワークを提案している [1][2]．本フレームワークでは，SW 志向のシステム設計記述および構成情報記述から，SW および HW の実行可能モジュールと通信インタフェースを自動合成することを目指している．貢献として，HW および SW/HW 間のバス通信に関する知識を必要とせずに，プログラマブル SoC 上の協調システムを設計できるようになることが挙げられる．文献 [3] では，SWORDS フレームワークによる設計生産性および自動合成が期待される通信インタフェースの性能を，他の設計環境と比較しながら評価している．

本稿では，まず，Xilinx 社 Zynq-7000 All Programmable SoC（Zynq）[4] を対象として，SWORDS フレームワークを実用的なツールチェーンとして実装する．実装成果を用いることで，設計者は C 言語による設計記述および JSON 形式による構成情報記述で SW/HW 協調システムを設計可能となる．さらに，構成情報記述の軽微な変更によって，様々なシステム構成を検討できるようになる．

さらに本稿では，SWORDS フレームワークを活用した設計空間探索手法を提案する．提案手法によって，高性能なシステム構成を低コストで探索できるようにする．設計者が指定した設計記述に対応する探索空間から，システム構成の候補に対応する構成情報記述の生成，および，設計記述への高位合成指示子の追加を行う．SWORDS フレームワークによるシステム合成結果に対して機械学習アルゴリズムを適用し，性能および FPGA の各種リソース量の

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
[a)] tani@lab3.kuis.kyoto-u.ac.jp
[b)] takase@i.kyoto-u.ac.jp

Vol.2017-ARC-225 No.18
Vol.2017-SLDM-179 No.18
Vol.2017-EMB-44 No.18
2017/3/9

予測モデルを生成する．これらの予測モデルを探索アルゴリズムの評価関数として活用することで，設計空間探索を実現する．

2. 準備

2.1 Zynq-7000 All Programmable SoC

Zynq はデュアルコア Cortex-A9 を中心としたプロセッシングシステム（PS）および FPGA を中心としたプログラマブルロジック（PL）によって構成されている．PS-PL 間の通信はマスタ・スレーブ方式であり，表 1 に示す 3 種類の通信ポートが備わっている．PS-PL 間における通信では，一方がマスタとなりもう一方がスレーブになることが要求される．GP-AXI（GP）は汎用ポートであり，PS と PL がそれぞれマスタとなるものが 2 個ずつ用意されている．AXI-HP（HP）は，オンチップメモリまたは DDR メモリへのアクセスが可能である．AXI-ACP（ACP）は，L2 キャッシュを介したアクセスが可能である．

Zynq が採用する AMBA AXI4 インタフェースにおけるプロトコルの実装を表 2 に示す．メモリマップ方式で最大 256 ワードバースト転送可能な AXI4，回路規模が小さい AXI4-Lite，および，アドレスフェーズを持たずストリーミングに特化した AXI4-Stream がある．AXI4-Lite は，バースト転送は不可能だが，一部の信号を省いているため，AXI4 と比べて実装が簡単化されている．

2.2 SWORDS フレームワーク

我々がこれまでに提案してきた SWORDS フレームワーク [1][2] のワークフローはタスク切り分け・生成，タスク・インタフェース合成および実行可能モジュール合成の 3 フェーズからなる．SW 向けの高級言語で記述された設計記述と構成情報記述を入力とし，SW 実行ファイルと HW モジュールおよびそれらの間の通信インタフェースを生成する．構成情報記述には，SW で実行する関数や HW で実行する関数，およびそれらの通信用のインタフェースなどの情報を記述する．

SWORDS フレームワークの入力である設計記述は C 言語で記述されるものとし，その処理粒度は関数単位とする．ここで，処理粒度はタスクと呼ぶこととし，プロセッサで処理されるタスクを SW タスク，FPGA 上で処理されるタスクを HW タスクと定義する．HW タスクの生成には高位合成を用いる．

SWORDS フレームワークにより，FPGA 上への HW 設計および SW/HW 間通信に関する深い開発知識を有していないシステム設計者でも，プログラマブル SoC 上のシステムの設計が可能となる．

表 1: Zynq の通信ポート

	ビット幅	ポート数	マスタ
GP-AXI Master	32b	2	PS
GP-AXI Slave	32b	2	PL
AXI-HP	32b/64b	4	PL
AXI-ACP	64b	1	PL

表 2: 3 種類の AXI インタフェースプロトコル

	チャネル	バースト転送可能回数
AXI4-Lite	アドレス/データ	不可
AXI4	アドレス/データ	256
AXI4-Stream	データのみ	無制限

2.3 関連研究

機械学習を用いた高位合成向け設計探索手法を紹介する．文献 [5] では，ランダムフォレスト法が高位合成ツール CyberWorkbench における使用リソースおよび実行時間両方の指標において予測モデル生成に適していることを示している．また，学習のためのサンプルを効果的に探索するため，トランスダクティブ実験デザイン（TED）を学習モデルのトレーニングセット抽出のために導入している．また，文献 [6] では，予測モデルとして，機械学習ツール Weka の M5P が適していることを示している．また，探索手法として遺伝的アルゴリズムを使用し，既存手法である焼きなまし法との比較を行っている．ただし，これらの手法では使用リソースの情報として ASIC として実装した時の面積を使用しており，FPGA の使用リソースの予測モデルに対する有効性は示されていない．

3. SWORDS フレームワークのツールチェーン実装

3.1 要件定義

本研究の実装では，入力である設計記述および構成情報記述から Zynq のための FPGA ビットストリームおよび SW 実行ファイルを合成することを目指す．

入力として与える設計記述は，高位合成に適用できない記法を除いた C 言語を扱うこととする．構成情報記述の形式は，文献 [7] で提案した JSON 形式による記法を用いる．これによって，記述の可読性および保守性が確保され，JSON Schema による型のチェックが可能となる．構成情報記述の例を，図 1 に示す．

3.2 ワークフローの実装

SWORDS フレームワークのツールチェーンのフローを図 2 に示す．ツールチェーン全体は，Windows のバッチファイルで swords.bat として実装している．swords.bat は，C ソースファイルおよび構成情報ファイルを入力とし，論理合成およびコンパイルまでツールチェーンの全てを実行する．swords.bat 内では，3 つのフェーズ内の処理を実行するためのそれぞれのコマンドを順に呼び出している．

SWORDS フレームワークの実装に用いるツールとそ

Vol.2017-ARC-225 No.18
Vol.2017-SLDM-179 No.18
Vol.2017-EMB-44 No.18
2017/3/9

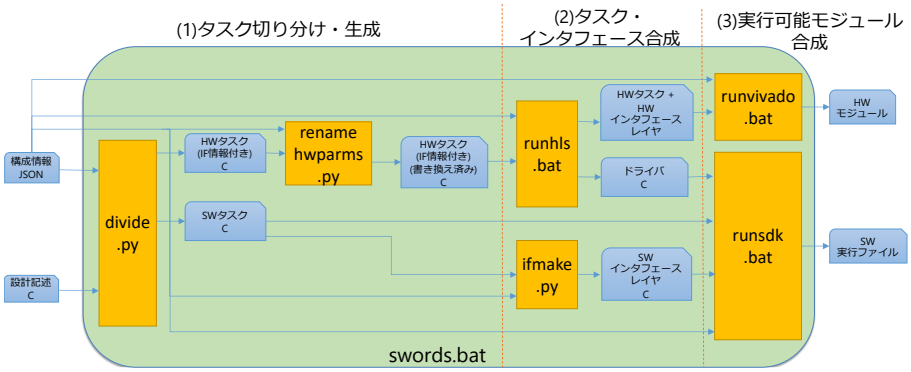


図 2: SWORDS フレームワークのツールチェーン

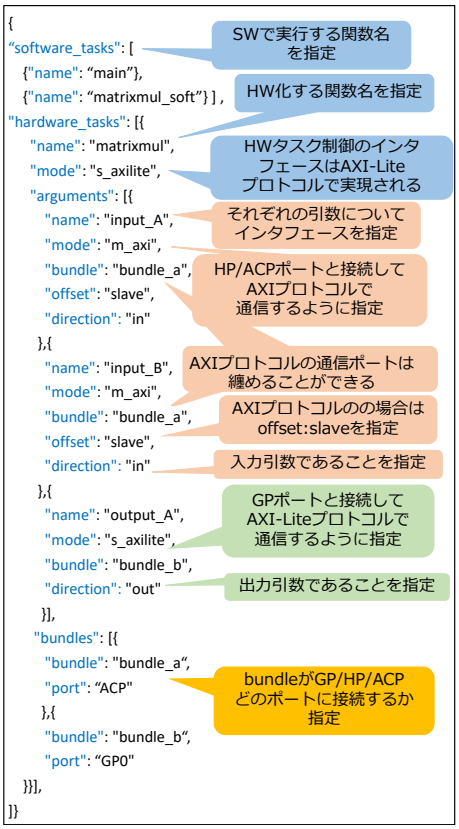


図 1: JSON 形式の構成情報記述の例

のバージョンを表 3 に示す．C 言語の設計記述の解析は python を用いて実装する．C 言語の設計記述の構文解析を行うため，Clang[8] および python で Clang を呼び出すためのパッケージである python binding を用いる．構成情報の型チェックを行うため，JSON Schema を用いる．実装した python プログラムを実行するため，Bash のシェルスクリプトを用いる．Bash シェルは，Bash on Ubuntu on Windows または Cygwin を用いる．HW タスクを IP コアに合成するため，Xilinx 社の高位合成ツール Vivado HLS を用いる．ビットストリームを合成するため，Xilinx 社の論理合成ツール Vivado を用いる．SW 実行ファイルを生成するため，Xilinx 社のソフトウェア開発キット Xilinx SDK を用いる．Xilinx 社のツールを実行するための tcl ファイルの生成は python を用いて実装する．Xilinx 社のツールを実行するスクリプトはコマンドプロンプトのバッチファイルとして実装する．

表 3: ワークフロー実装に用いるツールとそのバージョン

ツール	バージョン
python	2.7.1
Clang	3.4 以降
Bash	4.3.11(1)-release
Vivado HLS/Vivado/Xilinx SDK	2015.4

3 つのフェーズの実装について詳述する．C ソースファイルおよび構成情報記述を入力とするタスク切り分け・生成フェーズは，divide.py および renamehardwareparms.py として実装している．divide.py により，SW/HW タスクの切り分けおよび HW インタフェースレイヤを生成する．また，renamehardwareparms.py により，より高速な通信インタフェースを生成できるように HW タスク記述を修正する．生成した SW/HW タスクの情報を用いて，タスク・インタフェース合成フェーズを実行する．本フェーズは runhls.bat および ifmake.py として実装している．runhls.bat では，Vivado HLS を用いて高位合成を実行し HW タスクの HDL 記述を合成すると同時に，ドライバファイルが生成される．ifmake.py では，ドライバを用いて HW タスクを制御する SW インタフェースレイヤファイルを生成する．実行可能モジュール合成フェーズは runvivado.bat および runsdk.bat として実装している．runvivado.bat では，Vivado を用いて論理合成を行いビットストリームファイルを合成する．runsdk.bat により，Xilinx SDK を用いたプロジェクトのビルドによって SW 実行可能モジュールを合成する．

コマンドプロンプトにおける実行方法を以下に示す．

```
$ swords.bat <source_file>.c <config_file>.json \\
    <project_name> [<llvm_path>]
```

ツールチェーンの入力として，ソースファイル，構成情報ファイル，プロジェクト名およびオプションで Clang の

Vol.2017-ARC-225 No.18
Vol.2017-SLDM-179 No.18
Vol.2017-EMB-44 No.18
2017/3/9

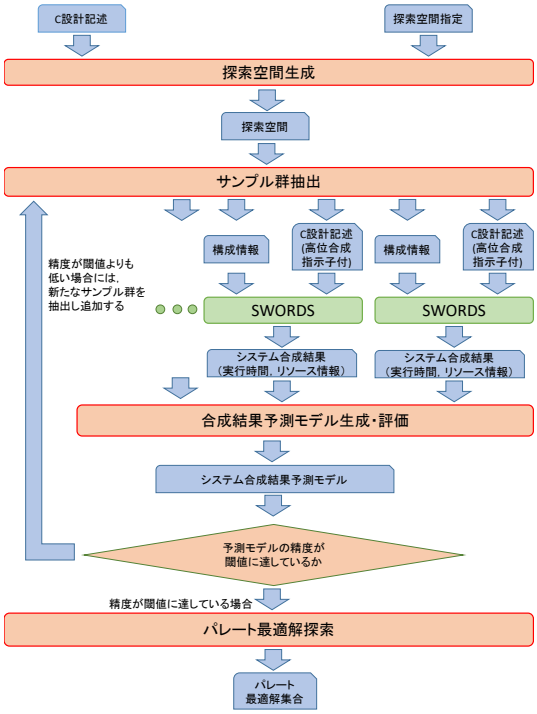


図 3: 提案手法のワークフロー

llvm ライブラリのパスを実行引数とする．指定しない場合，<llvm_path> は swords.bat 内に定義されているものが使用される．

4. 設計空間探索手法

提案する設計空間探索手法は，C ソースファイルおよび探索空間指定を入力とし，パレート最適な解の集合を出力する．探索対象は，合成結果の FPGA のシステムの実行時間および各リソースの使用量である．FPGA のリソースは，LUT，FF，BRAM および DSP である．ここで，パレート最適な解とは，あるシステム構成が，構成情報記述または高位合成の最適化記述を変更して実行時間または使用リソースのどちらかを改善するには，他の一方が悪化してしまう変更になることを指す．

本手法の特徴として，まず，設計空間に HW タスクの通信インタフェースの選択が含まれることが挙げられる．また，FPGA 上に実装したときの HW タスクの各リソース使用量に着目していることが挙げられる．プログラマブル SoC 上のシステム構成の探索時には，FPGA に搭載されている各リソースを評価関数として考慮する必要がある．

図 3 に，提案手法のワークフローを示す．ワークフローは設計空間生成，サンプル群抽出，合成結果予測モデル生成・評価およびパレート最適解探索の 4 つのフェーズから構成されている．以降，各フェーズについて詳述する．

4.1 設計空間生成フェーズ

本フェーズでは，設計記述と与えられた設計空間指定から，説明変数を要素とする探索対象の設計空間を生成する．説明変数とは，SW/HW 化の選択，HW タスクの引数の通信インタフェースの選択，HW タスクの高位合成指示子の記述といった，選択されたシステム構成における合成結果に影響を与える変数である．説明変数におけるそれぞれの選択は，0 または自然数にマッピングされる．これにより，設計空間を説明変数の数の次元を持つ離散的なベクトル空間として表すことができる．

提案手法で扱う説明変数の例として，まず，HW タスクにおいて，引数の通信方式の選択肢がある．HW タスク内の最適化については，さらに多くの選択がある．高位合成では，HW タスク内のループ，関数呼び出しおよび配列が最適化対象として扱える．Vivado HLS において適用可能な高位合成指示子の例として，ループまたは関数内での演算を同時処理できるようにする PIPELINE 指示子や，大きな配列を小さな配列に分割してアクセス競合のない BRAM に配置する ARRAY_PARTITION 指示子が挙げられる．

設計者は高位合成指示子の適用箇所や通信インタフェースの選択肢を説明変数として定義し，探索空間を指定する．ただし，両立できない説明変数の組み合わせがある．例えば，ネストされたループにおける PIPELINE 指示子は，内側または外側のループどちらかにのみ適用できる．この場合，両立できない組み合わせは設計空間から除外される．

4.2 サンプル群抽出フェーズ

1 つのシステム構成は，1 点のサンプルとして表される．サンプル群は，それぞれ対応する複数のシステム構成によって抽出される．抽出されたサンプル群は，合成結果予測モデル生成およびモデル精度評価に使用される．

本フェーズへの初回の移行時には，N 個のサンプルから構成される，予測モデルの生成および精度評価のためのサンプル群を双方とも抽出する．合成結果予測モデル生成・評価フェーズからサンプル群抽出フェーズに移行した場合，合成結果予測モデル生成のためのサンプル群にモデル精度評価のためのサンプル群を追加し，新たな合成結果予測モデル生成のためのサンプル群を生成する．また，N 個のサンプルから構成される新たなモデル精度評価のためのサンプル群の生成も行う．

サンプルとして選ばれたシステムを SWORDS フレームワークを用いて合成し，そのシステムの合成結果情報を得る．合成結果を取得したのち，合成結果予測モデル生成・評価フェーズに移行する．

4.3 合成結果予測モデル生成・評価フェーズ

本フェーズでは，サンプル群抽出フェーズで選択したサンプルの説明変数の組み合わせと合成結果から，合成結果予測モデルを生成する．予測モデルは，実行時間および各リソース情報それぞれについて生成する．本手法では，Python の機械学習ライブラリである Scikit-learn[9] に用意

されているランダムフォレスト法を採用する．これによってモデル生成のためのそれぞれのサンプルの合成結果から合成結果予測モデルを出力する．

生成した合成結果予測モデルがパレート最適解探索において十分な精度を持っているか評価するため，前フェーズで抽出した合成結果予測モデル生成のためのサンプル群を用いる．実行時間および各リソースの予測モデルの結果が，合成結果と比較して誤差率の二乗平均が閾値以下である場合，予測モデルの精度が十分であると判定して次フェーズに移行する．ただし，サンプル群抽出フェーズと合成結果予測モデル生成・評価フェーズのループにおいて予測モデルの精度が 2 回連続で十分であると判定されたときのみ，パレート最適解探索フェーズに移行するものとする．これは，サンプル選択の偏りにより，偶発的に予測モデルの精度が十分であると判定される可能性を避けるためである．予測モデルの精度が十分でない場合，サンプル群抽出フェーズに戻り，新たな予測モデルを生成するための準備を行う．

4.4 パレート最適解探索フェーズ

本フェーズでは，探索アルゴリズムの評価関数として生成した合成結果予測モデルを用いて，パレート最適な解集合を出力する．

提案手法では，全数探索または遺伝的アルゴリズム（GA）を用いる．設計空間が比較的小さい場合には，全数探索の結果からパレート最適解を探索する手法が適用できる．設計空間が大きく全数探索が実質的に不可能な場合は，GA などの多目的最適化が可能な手法を適用する必要がある．

全数探索を行う場合，全ての説明変数の組合せに対する実行時間およびリソース量の予測値を比較することで，パレート最適な解集合を得る．GA を用いる場合は，初期集団としてサンプル群選択で選ばれた説明変数の組合せの中でパレート最適な解候補の集合を用いる．子の生成および突然変異においては両立しない説明変数の組み合わせを持つ解候補は生成されないようにする必要がある．最終的な母集団がパレート最適解探索における解集合となる．

5. 適用事例

5.1 適用方法

実装成果物および提案手法の適用事例によって，研究成果の有用性を評価する．適用事例には，C 言語によって記述した行列積演算関数を用いる．

行列積演算の関数の引数には，入力となる二次元配列が 2 つ，出力となる二次元配列が 1 つある．それぞれの引数は，AXI4-Lite プロトコルまたは AXI4 プロトコルどちらかを用いた通信を選択する．AXI4 プロトコルを用いる場合，ACP ポートに接続するものとする．ここで，AXI4 プロトコルを用いる場合，HW タスク内部にバッファのための配列が用意されるため，内部バッファ配列は ARRAY_PARTITION 指示子を適用することが可能である．つまり，1 つの引数につき存在する選択肢は 3 種類となる．簡単のため，各引数の選択はそれぞれ 1 つの説明変数として抽出する．次に，for ループのパイプライン化の選択肢については，行列積演算を構成する三重ループのうち，内側または中間のループへの PIPELINE 指示子の適用を選択する．ここで，パイプラインの開始間隔についても選択肢を設け，PIPELINE 指示子の適用について計 21 個を選択する．以上により，説明変数は 5 個となり，探索設計空間は 567 通りの説明変数の組合せとして生成される．サンプル群はランダムに選択し，1 回のフェーズ移行で 20 個のサンプルを抽出することとする．実行時間の情報は評価ボード ZedBoard[10] 上での実行を 10 回行ったときの平均を用いることとする．誤差率の閾値は 10%とする．

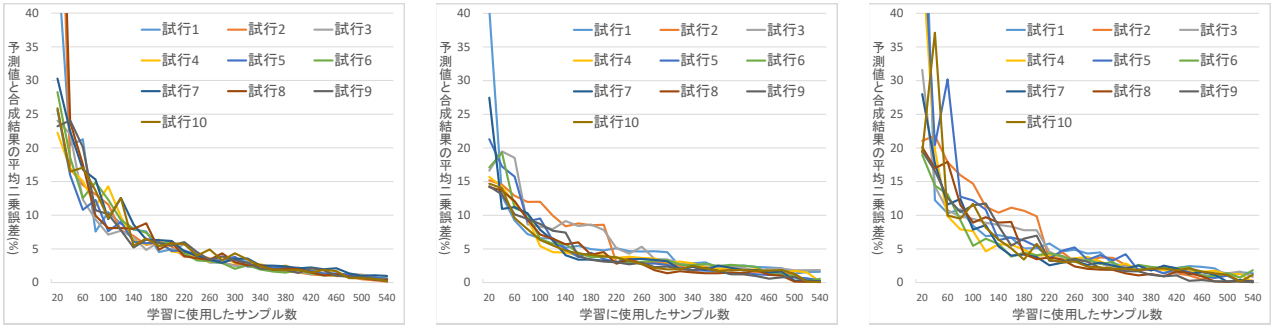
パレート最適解探索は，全数探索を行うもの，および，GA をそれぞれ適用する．GA を用いる場合，遺伝子の交配は一様交叉法を用いる．突然変異確率は 20%とし，5 つの説明変数のうちいずれかが変化するものとする．新しい解候補は，母集団の全ての解候補と比較され，どの解候補にもすべての評価値が劣っていない場合，母集団に追加される．新しい解候補にすべて評価値が劣る解候補が存在した場合，その解候補は母集団から除外される．新しい解候補の生成を 100 回繰り返して行うものとする．

5.2 結果および考察

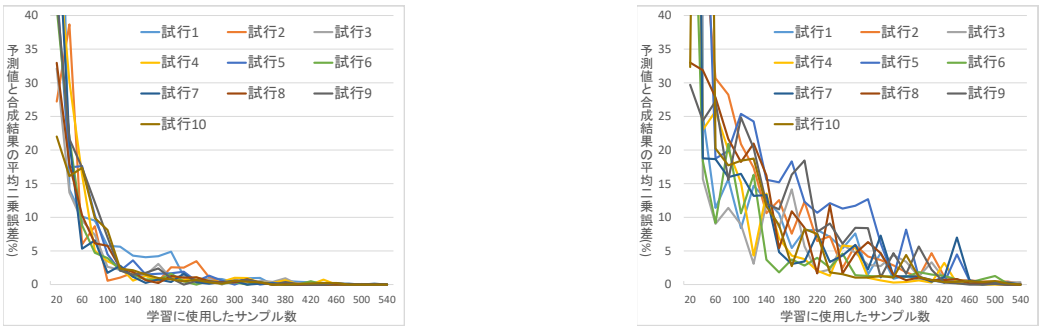
図 4 に，ランダムなサンプル選択における実行時間およびリソース量予測モデルの学習サンプル数による精度の変化を示す．図中の横軸はサンプル群のサンプル数を示し，縦軸はそのサンプル群を用いた時に生成された予測モデルの精度を表す．モデルの精度は，567 個の全ての合成結果と予測結果の平均二乗誤差割合によって評価した．本評価の結果，サンプル数の増大に応じてモデルの精度が高くなっていることがわかる．DSP 以外の評価関数では，平均的にはおおよそ 100 個のサンプルの合成結果を得ることにより，予測精度が 10%以内のモデルを生成できた．しかし，図 4(e) に示す DSP の予測モデルは他の予測モデルと比較し，精度が低いものとなっている．また，サンプル数を増加させた場合においても精度が不安定となる場合がある．

次に，全数探索および GA を用いたパレート最適解探索の結果について考察する．ここで，全ての選択肢を合成した結果から得られるパレート最適解集合を，真のパレート最適解集合とする．真のパレート最適解集合は 11 個の解から成っている．パレート最適解探索において，真のパレート最適解集合に含まれる解をどれだけ発見できたかを評価する．パレート最適解探索において，1000 回実行したときに，発見できた真のパレート最適解集合に含まれる解の個数の統計を，表 4 に示す．これより，全数探索を行った場合，平均で 6.6 個の真のパレート最適な解を発見でき

Vol.2017-ARC-225 No.18
Vol.2017-SLDM-179 No.18
Vol.2017-EMB-44 No.18
2017/3/9



(a) 実行時間　(b) LUT　(c) FF



(d) BRAM　(e) DSP

図 4: サンプル数による予測モデルの学習精度

たことがわかる．ただし，発見できる真のパレート最適な解の個数についてのばらつきが多かった．これは，設計空間が小さく，ランダムなサンプル選択を行うときに発生する偏りの影響が大きいことが理由として挙げられる．GA を用いた場合，平均で 5.1 個の真のパレート最適な解しか発見することができなかった．これは，設計空間が小さい場合は，GA は探索アルゴリズムとして適さないことを示している．全数探索が不可能な程度の大きな設計空間上では，GA の有用性があると考えられる．

6. おわりに

本研究では，まず，我々がこれまでに提案した SWORDS フレームワークを実用的なツールチェーンとして実装した．さらに，これを用いたプログラマブル SoC のための設計空間探索手法を提案した．適用事例によって，提案手法の有用性を示した．今後の方針としては，設計空間探索手法の各フェーズに用いる適切なアルゴリズムの検討や，他の設計による評価が挙げられる．

表 4: パレート最適解探索フェーズにおいて発見できた真のパレート最適解の数

個数	0	1	2	3	4	5
全数探索	0	0	4	35	64	145
GA	2	6	57	108	188	227
個数	6	7	8	9	10	11
全数探索	213	270	161	77	30	1
GA	200	128	54	26	4	0

謝辞

本研究はローム株式会社との共同研究による成果である．

参考文献

[1] Hideki Takase, et al.: A Study of a Software-Centric System Design Environment for Programmable SoCs, *In Proc. of ITC-CSCC 2014*, pp. 737–740 (2014).
[2] 谷祐輔，他：プログラマブル SoC のためのシステム設計環境における SW/HW インタフェース生成手法，電子情報通信学会技術研究報告，Vol. 115, No. 109, pp. 73–78 (2015).
[3] 谷祐輔，他：プログラマブル SoC のためのシステム設計環境における SW/HW 間通信方式の比較評価，電子情報通信学会技術研究報告，Vol. 175, No. 24, pp. 1–6 (2016).
[4] Xilinx 社：Zynq-7000 All Programmable SoC, `https://japan.xilinx.com/products/silicon-devices/soc/zynq-7000.html`.
[5] Liu, H.-Y. and Carloni, L. P.: On learning-based methods for design-space exploration with high-level synthesis, *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, IEEE, pp. 1–7 (2013).
[6] Schafer, B. C. and Wakabayashi, K.: Machine learning predictive modelling high-level synthesis design space exploration, *IET computers & digital techniques*, Vol. 6, No. 3, pp. 153–159 (2012).
[7] 谷祐輔，他：SWORDS フレームワークにおける Stream 通信への対応および構成情報記述の JSON 化，研究報告組込みシステム (EMB)， Vol. 2016, No. 3, pp. 1–2 (2016).
[8] a C language family frontend for LLVM: `http://clang.llvm.org/`.
[9] Pedregosa, Fabian, et.al.: Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830 (2011).
[10] Avnet 社：ZedBoard, `http://www.zedboard.org/`.