



ACRiルーム・ハンズオン 2024/6/27 14:00-17:00

HLS入門

ACRiルーム副室長 / ザイリンクス株式会社

安藤 潤

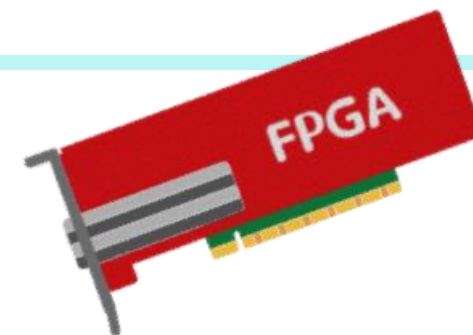
本セミナーはACRi参加団体の協賛金で運営されています

大学 4	 東京工業大学 東京工業大学	 筑波大学 筑波大学	 愛知工業大学 愛知工業大学	 信州大学 信州大学
プラチナ 2	 サイリンクス株式会社	 株式会社ジーデップ・アドバンス		
ゴールド 1	 アヴネット株式会社			
シルバー 13	 アイベックステクノロジー株式会社	 株式会社インテリジェント ウェイブ	 株式会社インターネットイニシアティブ	インテル株式会社
グリーン 7	 株式会社ゴフェルテック	 株式会社サイバーエージェント	 株式会社セック	 株式会社デンソーウェーブ
	 株式会社フィックスターズ	 GIGA-BYTE TECHNOLOGY CO., LTD	Intellectual Highway合同会社	 株式会社SUSUBOX
	TD SYNnex株式会社	株式会社イーツリーズ・ジャパン	株式会社インサイト	株式会社エッチ・ディー・ラボ
	Chiptip Technology株式会社	株式会社 TRIPLE-1	合同会社リトルウイング	株式会社ネフロック

ACRi 参加団体 (2024年6月)

自己紹介

- 安藤 潤（あんど う じゅん）
 - ザイリンクス株式会社 アプリケーションエンジニア
 - ACRIルーム副室長
- 主な担当製品
 - Alveo（PCIe FPGAカード）
 - Vitis（設計ツール）、HLS（高位合成）
 - Vivado



いらすとや風FPGAカード



V70 ビデオ解析AIデモ

お願い

- お手元の「情報工学系計算機室利用の注意事項」を良くお読みください。
 - 喫煙と飲食は厳禁。
 - 室内は土足厳禁、雨傘の持ち込みも禁止。
 - 作法を守ってコンピュータを正しく使う。
 - 整理・整頓を心がける。
 - このイベントは「わいわいしても大丈夫」です。
- 配布した「情報工学系計算機室 利用誓約書」にご記入ください。
- 受付で指定された座席にご着席ください。
- マスクの着用は個人の判断でお願いします。

本日の流れ

- 14:00 - 14:30
 - HLS概要
 - HLSチャレンジ概要、利用の流れ
- 14:30 - 15:00
 - ACRIルーム / HLSチャレンジ体験（ハンズオン）
- 15:00 - 15:30
 - 高速化テクニック紹介
- 15:30 - 17:00
 - テキストに沿って課題に挑戦（ハンズオン）
 - 質問や雑談など

ハンズオンで使用するアカウントについて

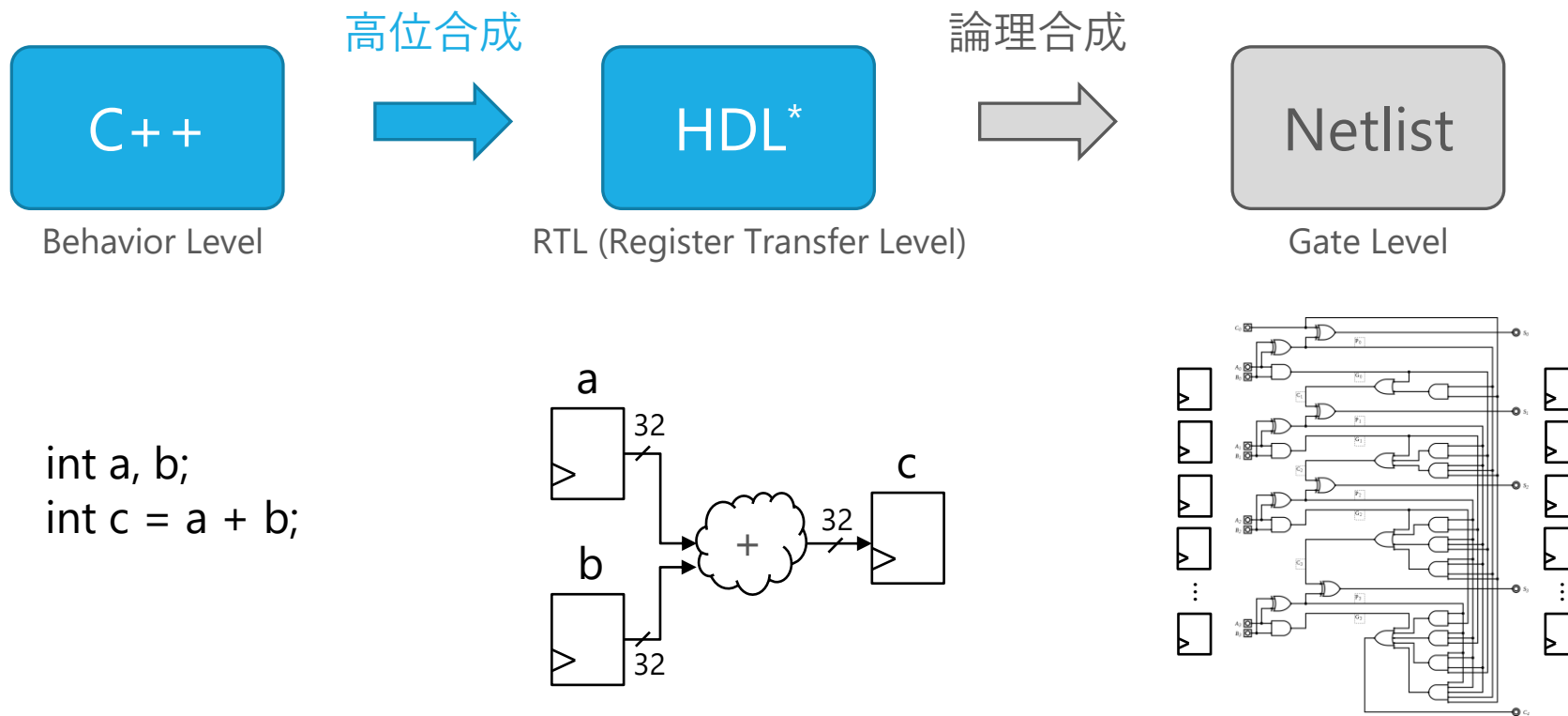
- 必要なアカウント
 - ACRIルーム
 - HLSチャレンジ
- お知らせください
 - アカウントがない
 - ユーザー名、パスワードを忘れてしまった
 - ログインできない
 - その他お困りのこと



HLS概要

HLSとは

- High Level Synthesis（高位合成）
 - プログラミング言語でハードウェアを設計する技術



HLSによる設計生産性の追求

「コーディング」生産性の追求

1. C/C++ (HLS-C) 記述

```
void sum_hls(float a, float b, float c, float d, float &dout)
{
    #pragma HLS INTERFACE ap_vld port=dout
    #pragma HLS INTERFACE ap_vld port=d
    #pragma HLS INTERFACE ap_vld port=c
    #pragma HLS INTERFACE ap_vld port=b
    #pragma HLS INTERFACE ap_vld port=a

    #pragma HLS PIPELINE

    dout = a + b + c + d;
}
```

HLS



2. 高位合成後のRTL

```
1 `timescale 1 ns / 1 ps
2
3 (* CORE_GENERATION_INFO="sum_hls,hls
4
5 module sum_hls (
6     ap_clk,
7     ap_rst,
8     ap_start,
9     ap_done,
10    ap_idle,
11    ap_ready,
12    d_ap_vld,
13    c_ap_vld,
14    b_ap_vld,
15    a_ap_vld,
16    a,
17    b,
18    c,
19    d,
20    dout,
21    dout_ap_vld
22 );
23
24 parameter    ap_ST_fsm_pp0_stage0 = 1'd1;
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52 assign ap_enable_reg_pp0_iter0 = ap_start;
53
54 assign dout = grp_fu_75_p2;
55
56 endmodule //sum_hls
```

Vivado



3. 論理合成後のネットリスト

```

:
:
6453 FDRE \din1_buf1_reg[8]
6454     (.C(ap_clk),
6455     .CE(ap_block_pp0_stage0_11001),
6456     .D(\din1_buf1_reg[8]_0 ),
6457     .Q(din1_buf1[8]),
6458     .R(1'b0));
6459 FDRE \din1_buf1_reg[9]
6460     (.C(ap_clk),
6461     .CE(ap_block_pp0_stage0_11001),
6462     .D(\din1_buf1_reg[9]_0 ),
6463     .Q(din1_buf1[9]),
6464     .R(1'b0));
6465 (* SOFT_HLUTNM = "soft_lutpair88" *)
6466 LUT3 #(
6467     .INIT(8'hAC))
6468     \dout[0]_INST_0
6469     (.I0(r_tdata[0]),
6470     .I1(\dout_r_reg_n_0_[0] ),
6471     .I2(ce_r),
6472     .O(dout[0]));
6473 (* SOFT_HLUTNM = "soft_lutpair93" *)
6474 LUT3 #(
6475     .INIT(8'hAC))
6476     \dout[10]_INST_0
6477     (.I0(r_tdata[10]),
6478     .I1(\dout_r_reg_n_0_[10] ),
6479     .I2(ce_r),
6480     .O(dout[10]));
6481 (* SOFT_HLUTNM = "soft_lutpair93" *)
:
:
```

← : 回路表記モデルの発展

→ : 実装に向けて自動生成

ACRi ブログ : [高位合成で加速するアクセラレータ開発 \(2\) ~ 高位合成と C ベース設計 1章](#) より

HLSの利点

- ツールがハードウェアを生成してくれる
 - マイクロアーキテクチャ（パイプライン、ステートマシン）
 - インターフェース（設定、データ）
 - ターゲット周波数、デバイスに合わせて
 - コード行数短縮、単純なコードで複雑なハードウェアが実現できる
 - アルゴリズムの実装に集中できる
- プログラミング言語で設計できる
 - コーディングに関する情報が豊富
 - メタプログラミングができる
- 設計期間短縮
 - 人手によるコーディングミスがない
 - ソフトウェアレベルで高速シミュレーション、容易にデバッグ

HLSの弱点

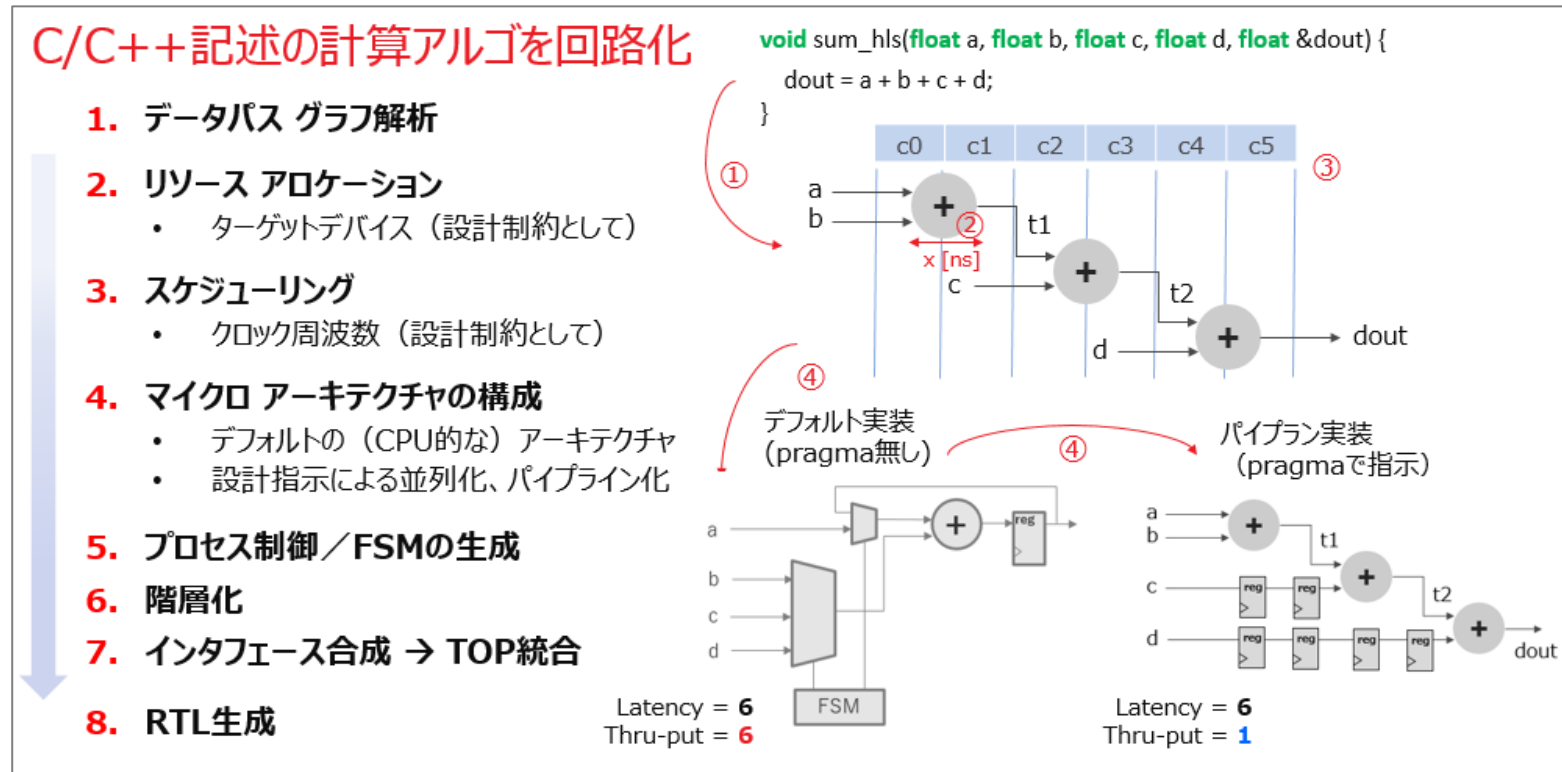
- 詳細なハードウェア設計はできない
- ハードウェア設計の知識はある程度必要
- ツールをうまく使いこなすために経験が必要

HLS設計の流れ

- アルゴリズム、仕様を決定
 - 計算内容
 - インターフェース
 - 要求性能、回路規模
- リファレンスモデルを作成
 - 好きな言語で期待値を出力するソフトウェアを作成する
- HLSコードを作成
 - 実現したい回路を思い浮かべながらコーディングする
- HLSコードを検証
 - リファレンスモデルと同じ出力をするか
 - 性能や回路規模が要求を満たすか

HLSプラグマ

- どのような回路にしたいかツールに伝える手段
 - 例：#pragma HLS PIPELINE



ACRi ブログ： [高位合成で加速するアクセラレータ開発 \(2\) ～ 高位合成と C ベース設計 1章](#) より

パイプライン

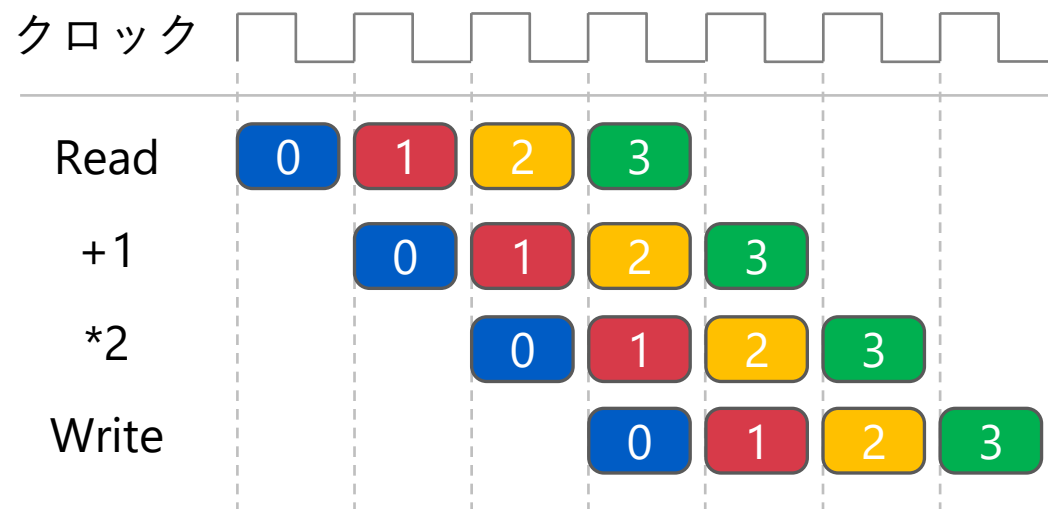
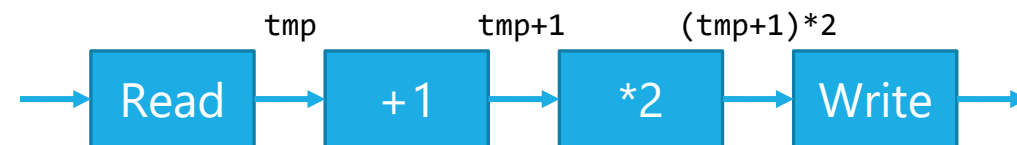
- ソフトウェア

- 上から順番に実行するイメージ

```
void func(const float in[256], float out[256]) {  
    for (int i = 0; i < 256; i++) {  
        float tmp = in[i];  
        tmp = tmp + 1;  
        tmp = tmp * 2;  
        out[i] = tmp;  
    }  
}
```

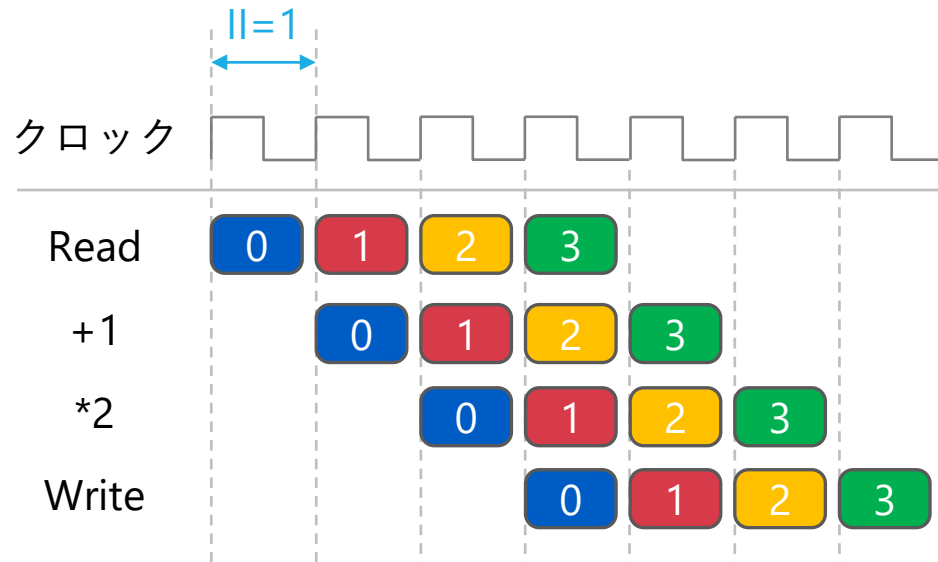
- パイプライン化されたハードウェア

- すべての回路が並列（同時）に動作

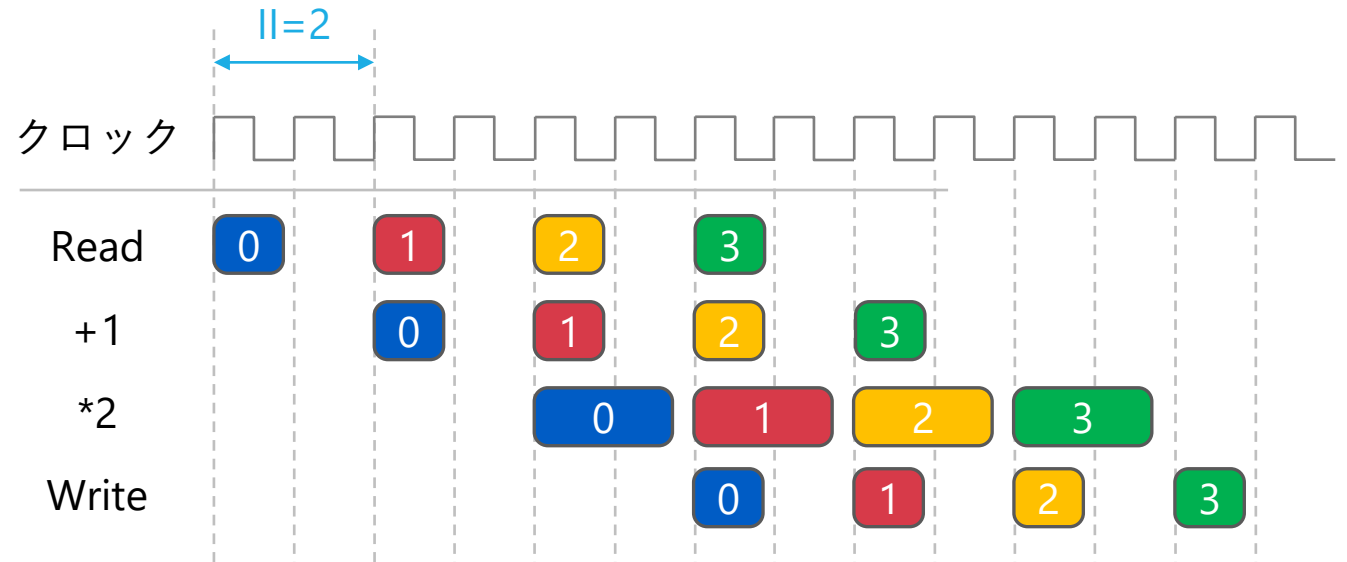


パイプラインの重要な概念

- II (Initiation Interval、開始間隔)
 - 何サイクルおきにループ内の処理を開始できるか
 - 理想は II=1



II=1の理想的なパイプライン



ボトルネックがあり II=2 になってしまったパイプライン、性能は半分



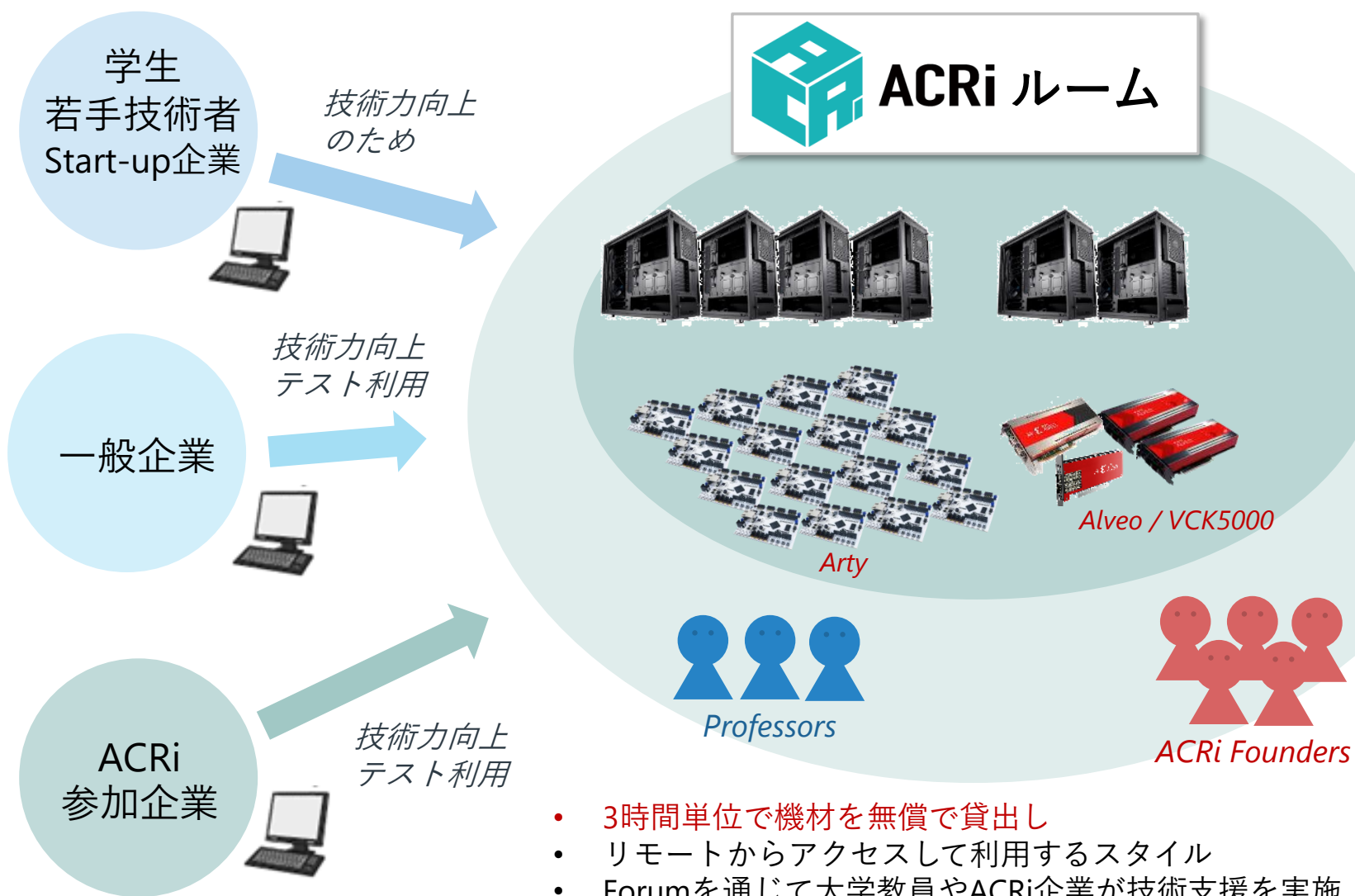
HLSチャレンジ概要

ACRi HLS チャレンジとは？

- お題に沿った回路を HLS（高位合成）で設計し、性能を競います
- HLS の活用促進
 - ザイリンクス FPGA 向けの無償 HLS ツールを使用
 - C++だけでFPGAアクセラレータの開発ができる
 - HLS を学んでみたい方の練習の場
 - 高速な回路の設計技術を共有する場
- ACRi ルームを活用



ACRi ルーム



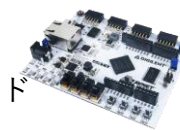
FPGA Server

- CPU: Core i9 (8 core /16 thread)
- メモリ: DDR4 128GB (32GB x 4)
- ストレージ: SSD M.2 1TB x2
- 開発環境インストール済み



Arty A7-35T

- Digilent社 Arty A7-35T カード
- 1サーバにArtyを15枚接続
- ユーザ毎にVMを割り当て



Alveo

- Xilinxアクセラレータカード
- Alveo U50 / U200 / U250 / U280 / VCK5000



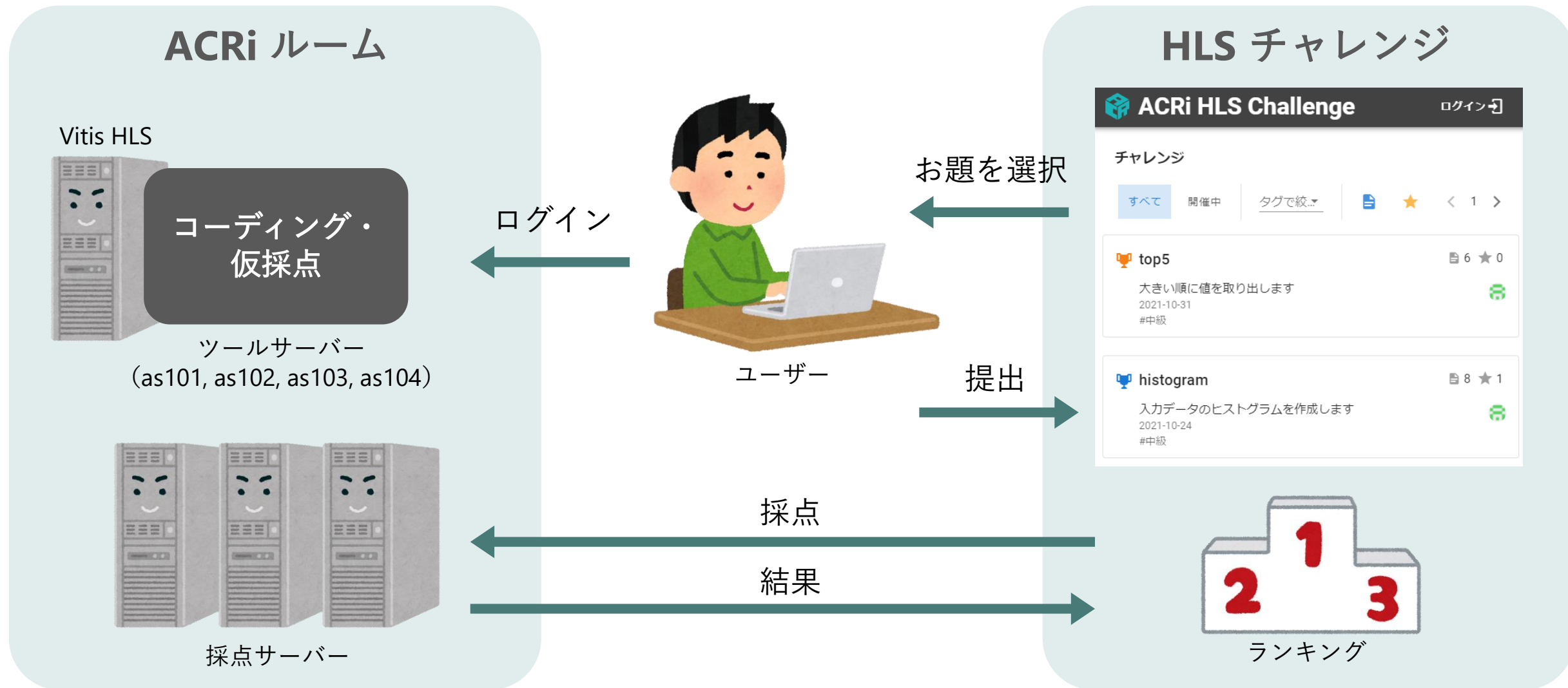
Instinct

- AMD Instinct GPU
- MI210 64GB



New!

HLS チャレンジの利用方法



人気チャレンジ bai-gaeshi

- 投稿数：141件
(2021年11月時点)
- 配列の値を倍にして返すシンプルなお題

bai-gaeshi 数を倍にせよ

初級

浮動小数点演算

チャレンジ

提出一覧

ランキング

問題

入力されるfloat配列の要素の値をすべて倍にして出力しなさい。

ただし入力される値の範囲は $[-1000, 1000)$ とします。



カーネルヘッダー

```
#pragma once

extern "C" {
    void kernel(const float in[1024], float out[1024], int size);
}
```

bai-gaeshi の解答例

- 素直なソフトウェア実装
- 採点結果
 - 動作周波数：599.9 MHz
 - サイクル数：1219 cycles
 - タイム：2032.073 ns

 [bai-gaeshi](#) への提出コード  0
提出時刻：2021/8/31 22:10:17

✓ bai-gaeshi 2021/8/31 22:10:17

結果	パス		
FF	2615	周期	1.667
LUT	2319	周波数	599.9
DSP	3	サイクル	1219
BRAM	2	タイム	2032.073
URAM	0		

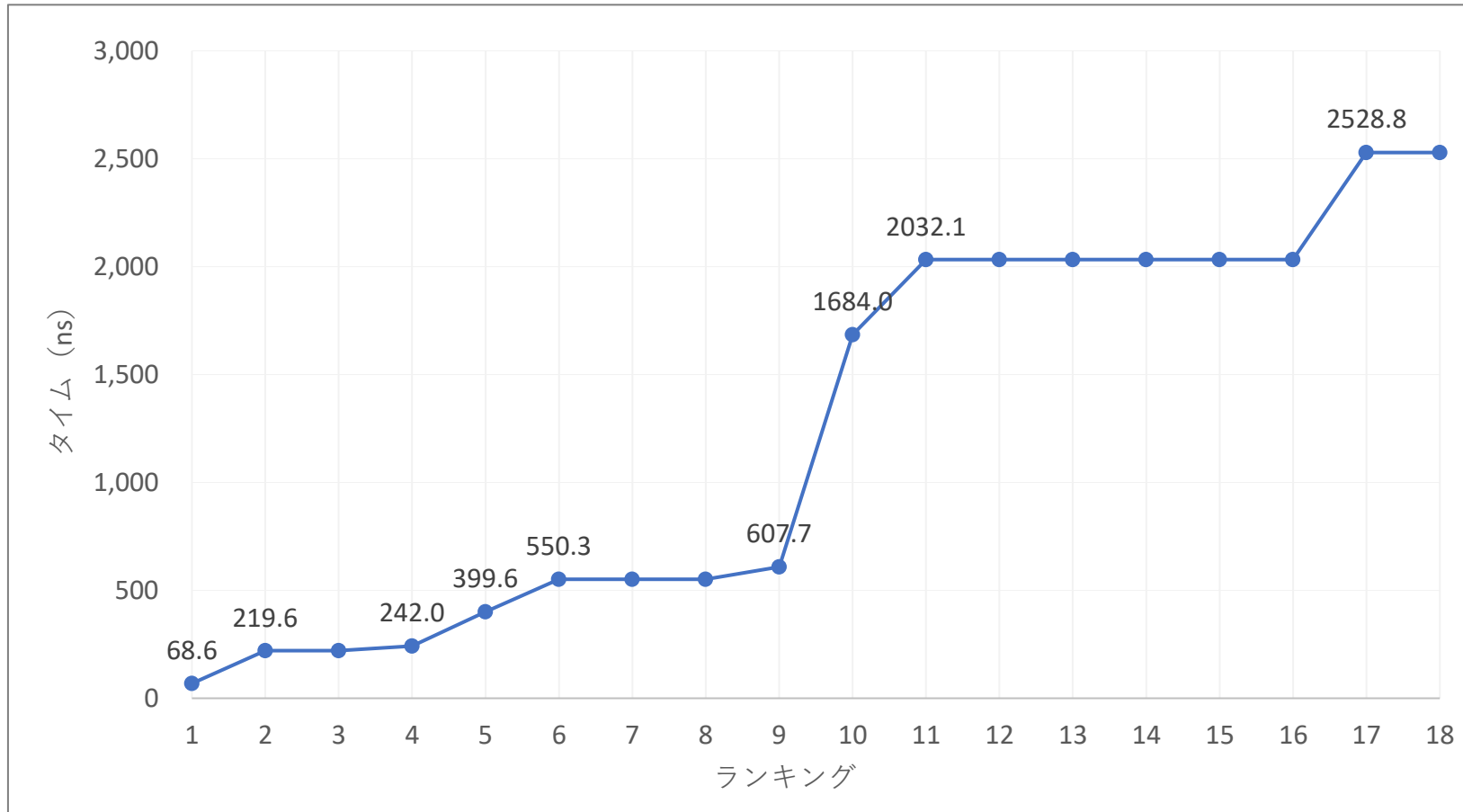
 anjn  0

コード CSIMログ HLSログ COSIMログ

```
#include "kernel.hpp"

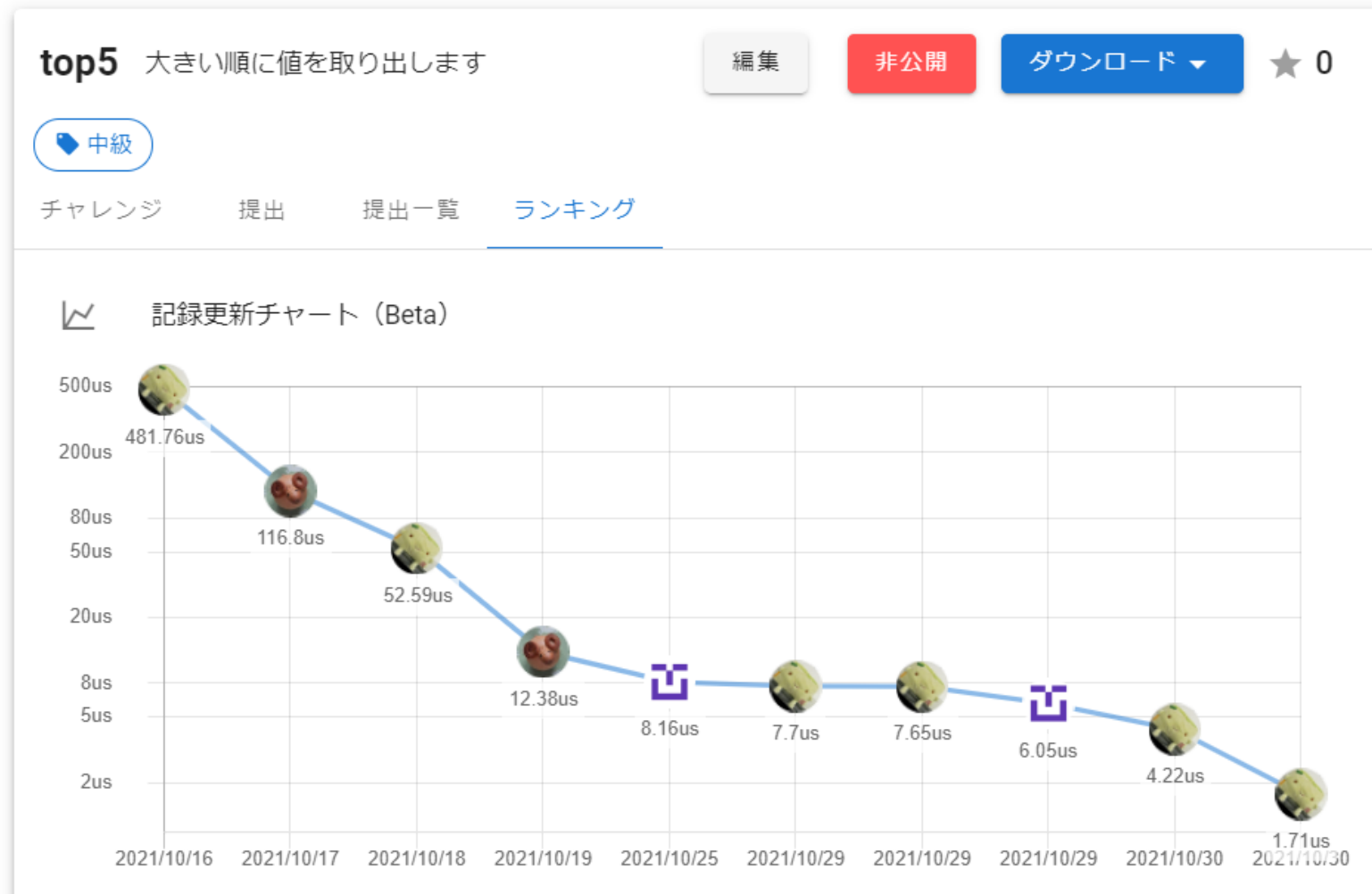
void kernel(const float in[1024], float out[1024], int size) {
  for (int i=0; i<size; i++) out[i] = in[i] * 2;
}
```

bai-gaeshi のランキング



コードの記述方法によって36倍の差！

チャレンジ記録更新の様子



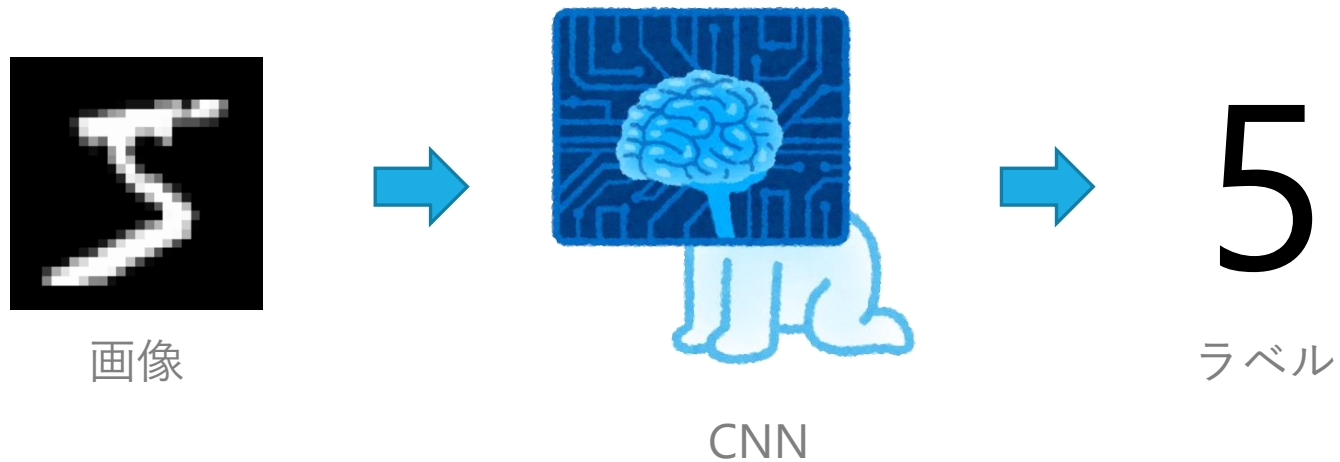
抜きつ抜かれつの熱い闘い！



AIチャレンジ紹介

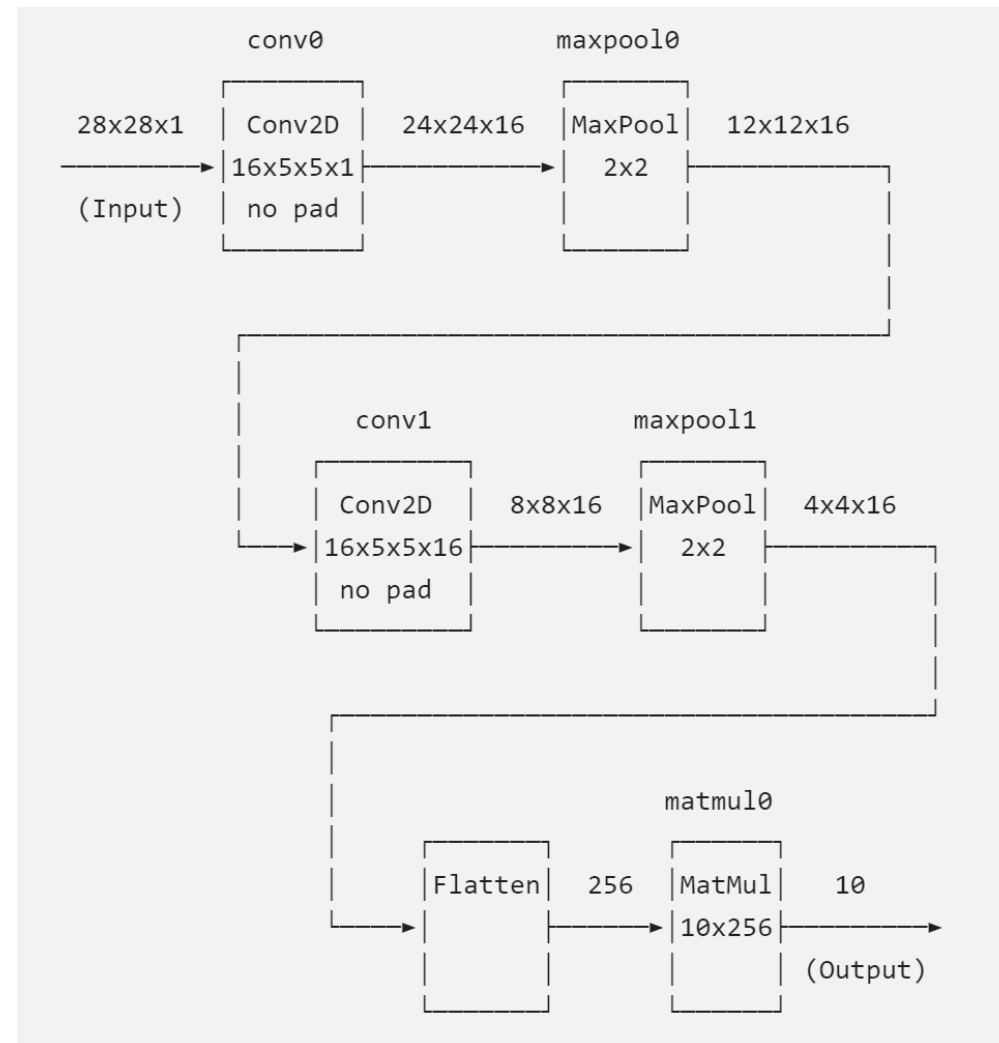
AIチャレンジ

- MNIST AI
 - 画像から数字を認識する回路を実装します



AIチャレンジ

- 実装する回路
 - 畳み込みニューラルネットワーク (CNN)
 - 畳み込み層 (Conv2D)
 - Max Pooling層
 - 全結合層 (MatMul)
 - 学習済みの重みを使って計算
 - 重み：符号付き2bit (3値：-1, 0, 1)
 - アクティベーション：符号なし2bit (0~3)
 - 簡単なレイヤーから順に段階を踏んで実装



AIチャレンジ

- 6つの難易度別チャレンジ

- 初級

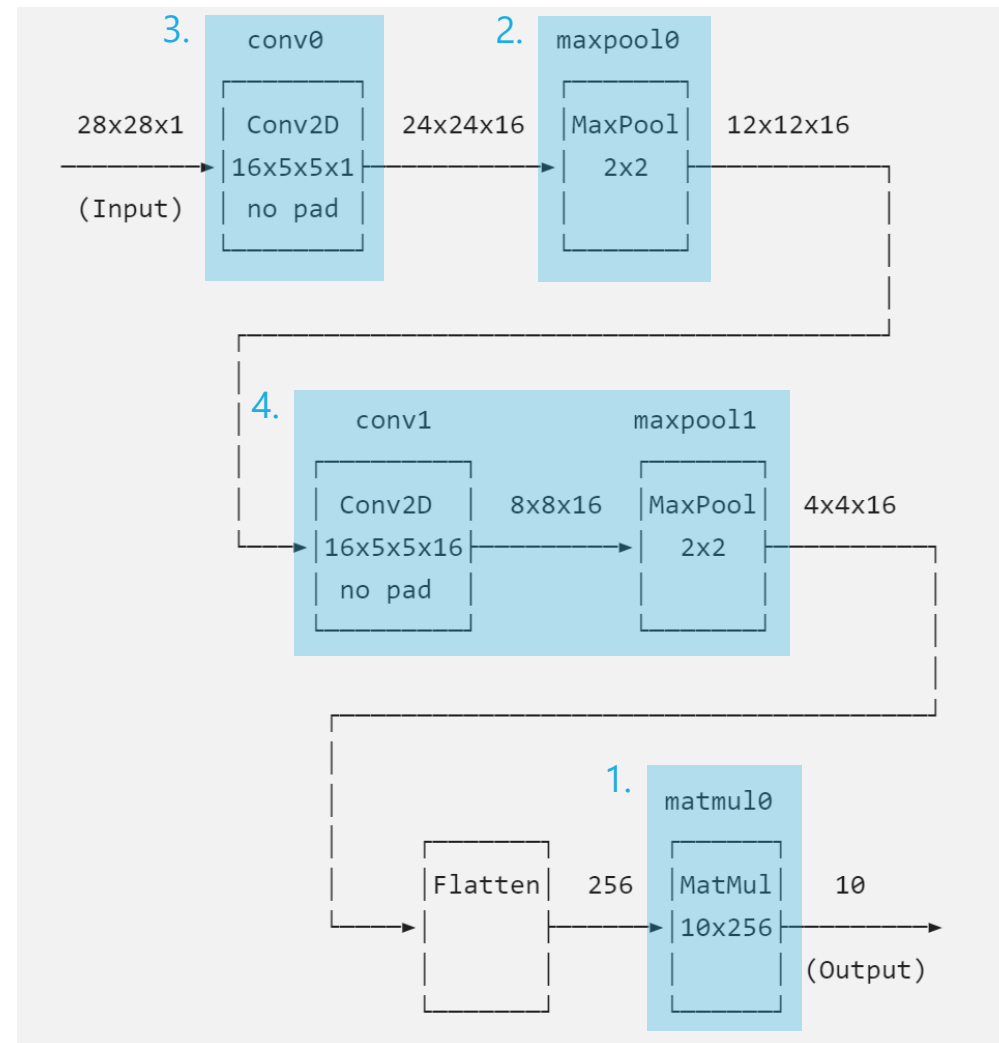
1. mnist1-matmul0
2. mnist2-maxpool0

- 中級

3. mnist3-conv0
4. mnist4-conv1-mp1

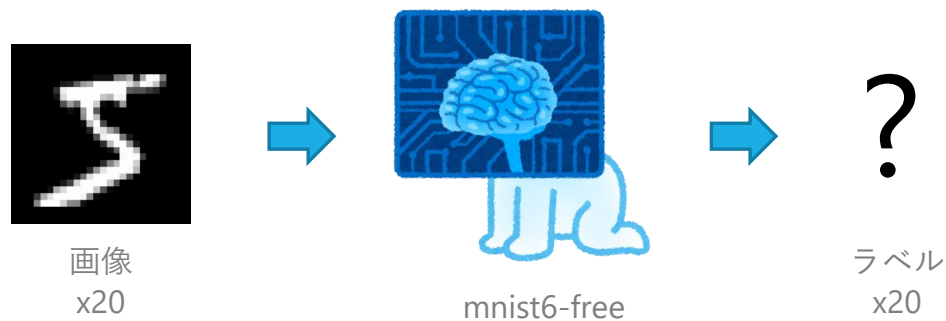
- 上級

5. mnist5-all (すべてのレイヤー)
6. mnist6-free (実装内容自由)



AIチャレンジ

- 変わり種チャレンジ : mnist6-free
 - 新しい実験的な試み
 - 回路の実装内容を問わずラベルの正答率のみでパス判定
 - アルゴリズムから作成してもらう
 - mnist5-all をベースにしても OK
 - 画像20枚のテストで90%以上正答することがパスの条件



チャレンジの状況

- mnist6-free
 - 1.37us まで高速化
 - ベースライン実装から45,900倍
 - 1入力あたり47サイクル！
 - オーバーヘッド込みのサイクル数
 - 1秒あたり1063万枚以上
 - FPGAに特化したNNを独自フレームワークで学習されたそう
 - ACRIウェビナーで講演していただきました！
 - <https://www.youtube.com/watch?v=5Q-2yTU3rbs&t=572s>
 - <https://speakerdeck.com/ryuz88/orizinarunoshen-ceng-xue-xi-de-hls-challenge-nitiyarenzi>





HLSチャレンジ利用の流れ

利用の流れ

1. HLSチャレンジにログイン
 2. チャレンジ（お題）を選択
 3. チャレンジをダウンロード
 4. コードを作成
 5. コードをチェック
 6. HLSチャレンジに投稿
 7. ランキング上位を狙ってコードをブラッシュアップ！
- } ACRIルームを活用

1. ログイン

- 右上の「ログイン」から[GitHubアカウント](#)または[メールアドレス](#)でログインします



※ ACRiルームのアカウントとは連携していません

2. チャレンジを選択

- 取り組んでみたい
チャレンジを選択
- チャレンジの例
 - ベクトル演算の基本
 - ストリーム処理の基本
 - ソート
 - SHA256ハッシュ
 - マンデルブロ集合

チャレンジ

すべて 開催中 タグで絞り込む

bai-gaeshi sha256

初級 第0回チャレンジ 浮動小数点演算 中級 第0回チャレンジ

期限 : 2021-07-17 期限 : 2021-07-17

数を倍にせよ ハッシュを実装します

anjn 0 ★ 2 anjn 0 ★ 0

mandelbrot-set stream-accum

中級 第0回チャレンジ ストリーム 初級 第0回チャレンジ

期限 : 2021-07-17 期限 : 2021-07-17

マンデルブロ集合で遊ぼう 流れてくるデータを処理してみよう

anjn 1 ★ 0 anjn 1 ★ 0

チャレンジの例

- ベクトルの要素を倍にして返す問題
- <https://acri-vhls-challenge.web.app/challenge/bai-gaeshi>

bai-gaeshi 数を倍にせよ ダウンロード ★ 2

初級 第0回チャレンジ 浮動小数点演算

チャレンジ 提出一覧 ランキング

問題

入力されるfloat配列の要素の値をすべて倍にして出力しなさい。

カーネルヘッダー

```
#pragma once

extern "C" {
void kernel(const float in[1024], float out[1024], int size);
}
```

問題文

この関数宣言に対応する
HLSコードを自由に記述します

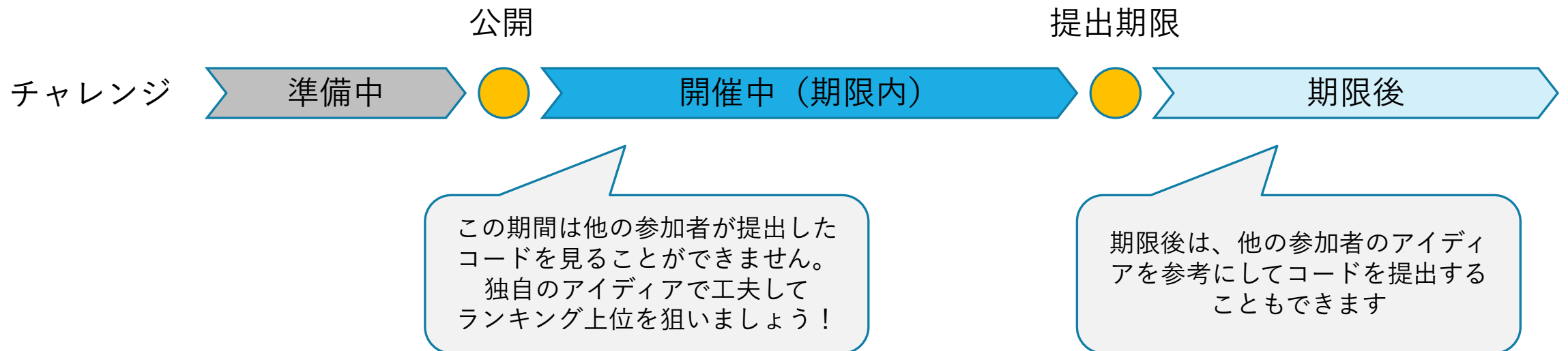
チャレンジの例

- チャレンジには提出期限や、コードが満たすべき条件が設定されています
 - 最初は気にする必要はありません
 - ランキング上位を目指す場合には気をつけてください

提出期限	2021-07-17	実行時間制限	
合成制約		CSIM	1 分
クロック周期	2 ns	HLS	2 分
リソース制限		CoSIM	5 分
FF	無制限	論理合成	30 分
LUT	50,000	ビルド条件	
DSP	無制限	コンパイルフラグ	
BRAM	無制限	リンクフラグ	
URAM	無制限	ツール	2021.1

チャレンジの提出期限について

- すべてのチャレンジには提出期限が設定されています
- 提出期限を過ぎるとすべてのコードが公開されます
- 期限内に上位にランクした記録は期限後も残ります



3. チャレンジをダウンロード

- 「ターミナルにコピー」を使ってACRiルームのサーバーにチャレンジを転送します
- TeraTermまたはPower Shellを使ってACRiルームにログインし、ターミナルに貼り付けて実行します（リモートデスクトップではうまくいきません）
- ファイルに保存してscpで転送しても構いません

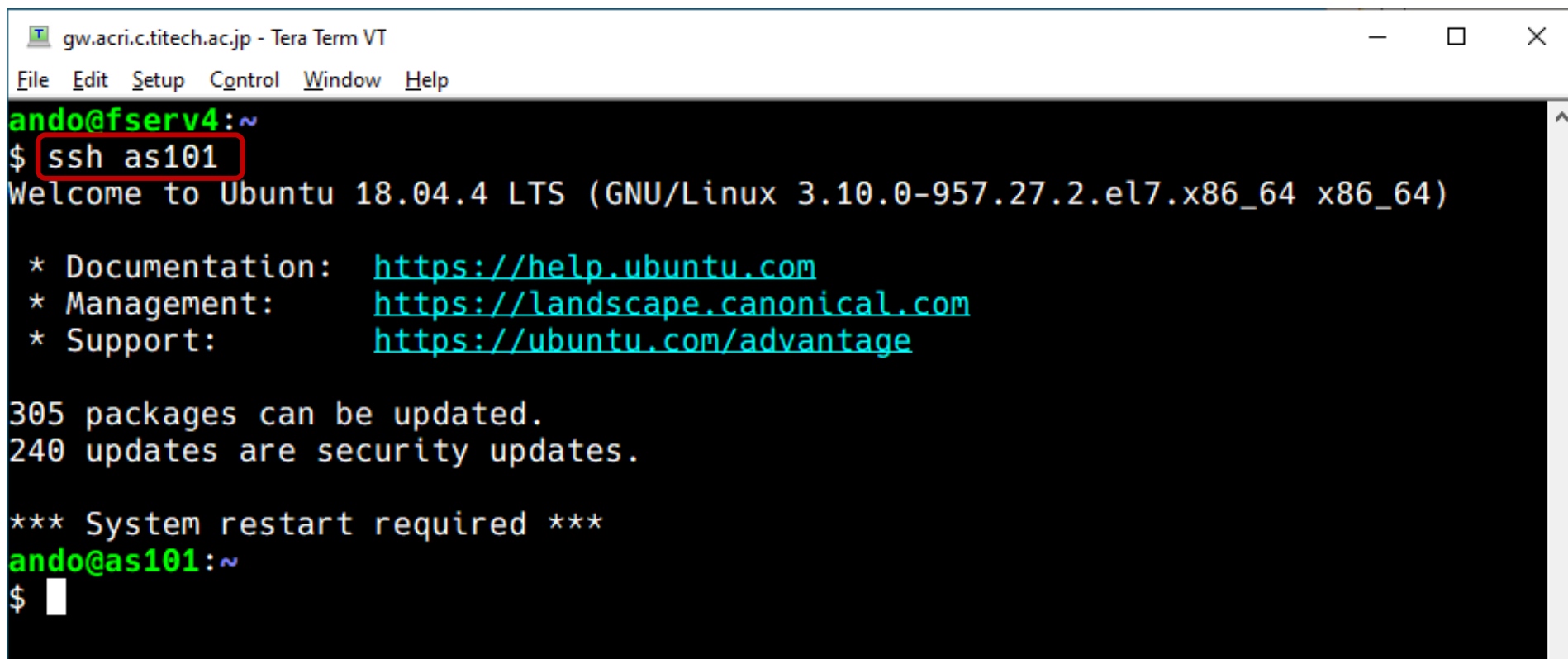


ACRiルーム

- ACRiルームでコードを作成・仮採点してから、HLSチャレンジへ提出します
 - ACRiルームの登録はこちら
 - <https://gw.acri.c.titech.ac.jp/wp/manual/apply-for-account>
 - サーバーの利用方法はこちら
 - <https://gw.acri.c.titech.ac.jp/wp/manual/how-to-reserve>
- いつでも使用できるサーバーを用意しています
 - 予約不要、時間制限なし！
 - ホスト名：as101、as102、as103、as104、as105
 - ご一読ください：<https://gw.acri.c.titech.ac.jp/wp/manual/alveo-server#toc2>

ACRiルームにログイン

- ACRiルームのツール専用サーバーへ接続
 - 例：ssh as101



```
gw.acri.c.titech.ac.jp - Tera Term VT
File Edit Setup Control Window Help
ando@fserv4:~
$ ssh as101
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 3.10.0-957.27.2.el7.x86_64 x86_64)

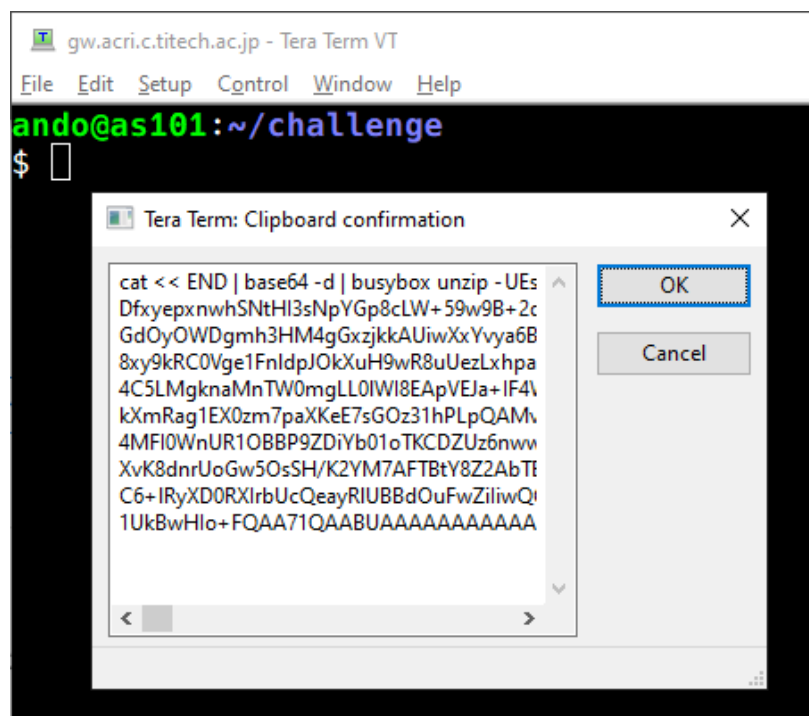
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

305 packages can be updated.
240 updates are security updates.

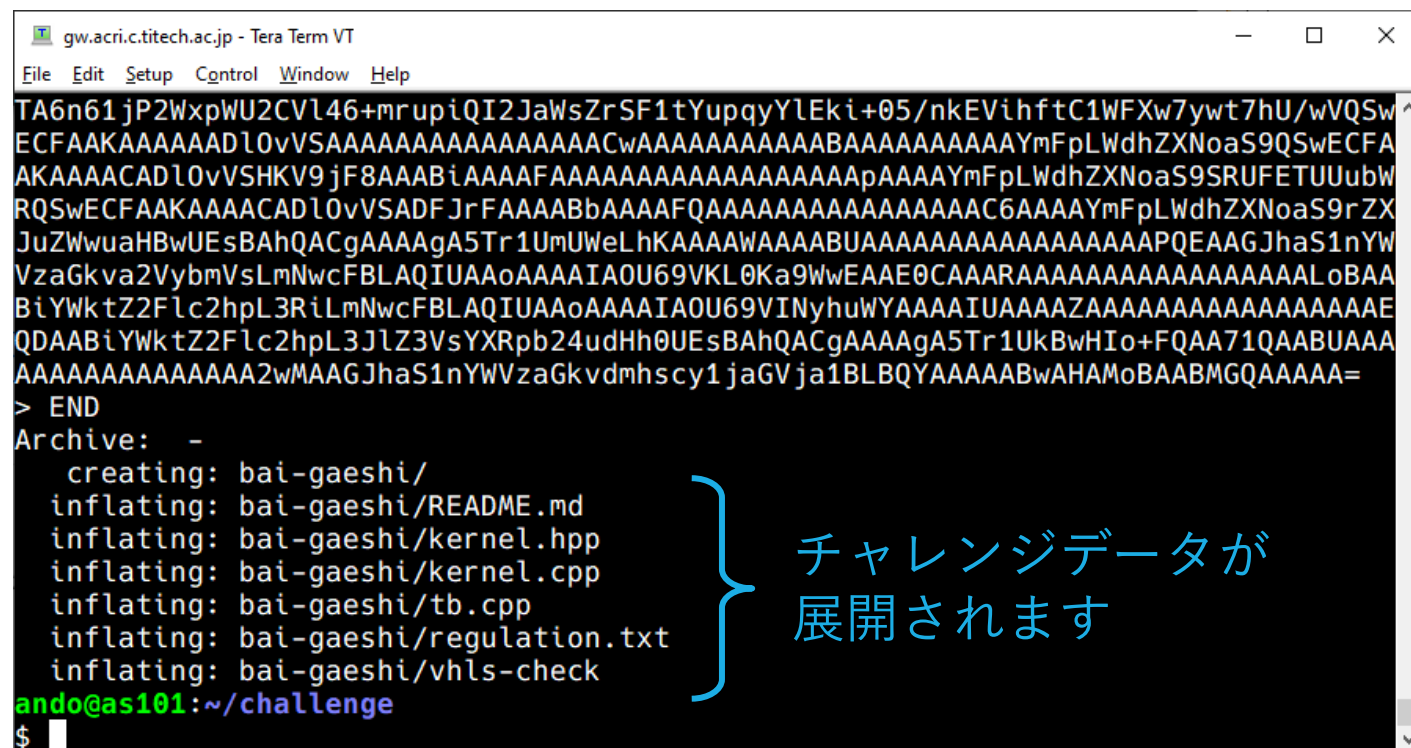
*** System restart required ***
ando@as101:~
$
```

チャレンジを転送

- ターミナルに貼り付けて、エンター



The screenshot shows a terminal window titled "gw.acri.c.titech.ac.jp - Tera Term VT". The prompt is "ando@as101:~/challenge" and the input is "\$". A "Tera Term: Clipboard confirmation" dialog box is open, displaying a long base64-encoded string. The dialog has "OK" and "Cancel" buttons.



The screenshot shows the same terminal window after the challenge data has been pasted. The data is a long base64 string followed by a list of files to be inflated. A blue bracket on the right side of the terminal points to the list of files, with the text "チャレンジデータが展開されます" (Challenge data will be expanded) written next to it.

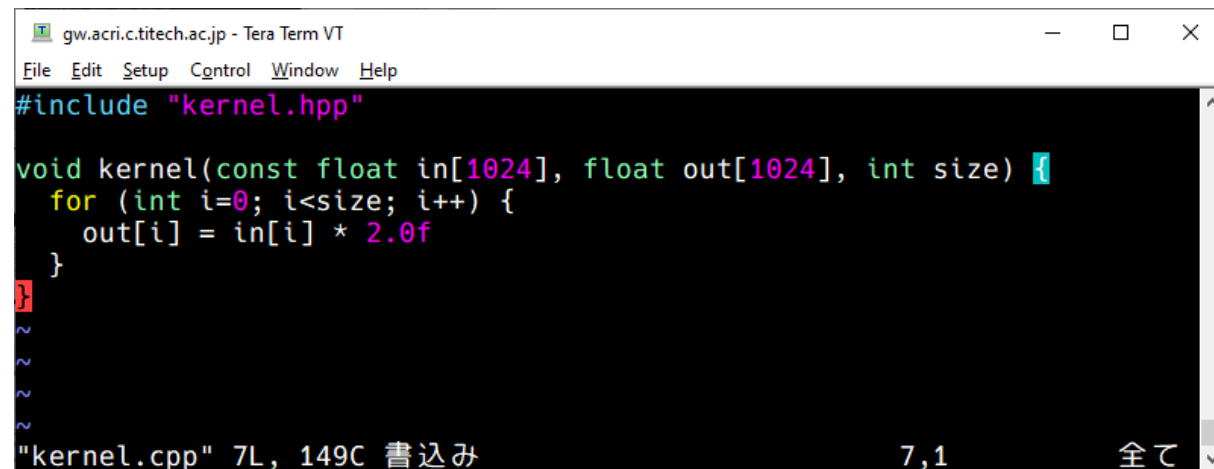
```
TA6n61jP2WxpWU2CVl46+mrupiQI2JaWsZrSF1tYupqyYlEki+05/nkEVihftC1WFXw7ywt7hU/wVQSw^
ECFAAKAAAAAADl0vVSAAAAAAAAAAAAAAAAACwAAAAAAAAABAAAAAAAAAYmFpLWdhZXNoaS9QSwECFA
AKAAAAACADl0vVSHKV9jF8AAABiAAAAFAAAAAAAAAAAAAAAAAAApAAAAAYmFpLWdhZXNoaS9SRUFETUUbW
RQSwECFAAKAAAAACADl0vVSADFJrFAAAABbAAAAAFQAAAAAAAAAAAAAAAAAAC6AAAAAYmFpLWdhZXNoaS9rZX
JuZWwuaHBwUESBAHQACgAAAAGA5Tr1UmUWeLhKAAAAWAAAAABUAAAAAAAAAAAAAAAAAAAPQEAGJhaS1nYW
VzaGkva2VybmVsLmNwcFBLAQIUAAoAAAAIAOU69VKL0Ka9WwEAAE0CAAAARAAAAAAAAAAAAAAAAAAALoBAA
BiYWktZ2Fjc2hpL3RiLmNwcFBLAQIUAAoAAAAIAOU69VINyhuWYAAAAIUAAAAZAAAAAAAAAAAAAAAAAAAE
QDAABiYWktZ2Fjc2hpL3JlZ3VsYXRpb24udHh0UESBAHQACgAAAAGA5Tr1UkBWHiO+FQAA71QAABUAAA
AAAAAAAAAAAAAAAAAA2wMAAGJhaS1nYWVzaGkvdmhscy1jaGVja1BLBQYAAAAABwAHAMoBAABMGQAAAAA=
> END
Archive: -
  creating: bai-gaeshi/
  inflating: bai-gaeshi/README.md
  inflating: bai-gaeshi/kernel.hpp
  inflating: bai-gaeshi/kernel.cpp
  inflating: bai-gaeshi/tb.cpp
  inflating: bai-gaeshi/regulation.txt
  inflating: bai-gaeshi/vhls-check
ando@as101:~/challenge
$
```


4. コードを作成

- チャレンジデータ

- README.md ... 問題文
- kernel.hpp ... カーネルヘッダー
- **kernel.cpp** ... **カーネルソース（雛形）**
- tb.cpp ... テストベンチ
- regulation.txt ... 採点条件
- vhls-check ... 採点スクリプト

カーネルを実装します



```
gw.acri.c.titech.ac.jp - Tera Term VT
File Edit Setup Control Window Help
#include "kernel.hpp"

void kernel(const float in[1024], float out[1024], int size) {
    for (int i=0; i<size; i++) {
        out[i] = in[i] * 2.0f
    }
}

"kernel.cpp" 7L, 149C 書込み
```

5. コードをチェック

- 採点スクリプトを実行します

```
gw.acri.c.titech.ac.jp - Tera Term VT
File Edit Setup Control Window Help
ando@as101:~/challenge/bai-gaeshi
$ chmod a+x ./vhls-check
ando@as101:~/challenge/bai-gaeshi
$ ./vhls-check -v -f
```

セミコロンを忘れていた
のでエラーに...

```
gw.acri.c.titech.ac.jp - Tera Term VT
File Edit Setup Control Window Help
csim.mk:81: recipe for target 'obj/kernel.o' failed
../../../../../../../../kernel.cpp: In function 'void kernel(const float*, float*, int)
':
../../../../../../../../kernel.cpp:6:3: error: expected ';' before '}' token
    }
    ^
make: *** [obj/kernel.o] Error 1
ERROR: [SIM 211-100] 'csim_design' failed: compilation error(s).
INFO: [SIM 211-3] ***** CSIM finish *****
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 0.59 seconds. CP
U system time: 0.3 seconds. Elapsed time: 1.96 seconds; current allocated memory
: 251.079 MB.
4
    while executing
"source csim.tcl"
  ("uplevel" body line 1)
   invoked from within
"uplevel \#0 [list source $arg] "
INFO: [Common 17-206] Exiting vitis_hls at Wed Jul 21 16:43:44 2021...
Command exited with non-zero status 1
3.42user 0.86system 0:05.99elapsed 71%CPU (0avgtext+0avgdata 1215604maxresident)
k
88inputs+872outputs (0major+396285minor)pagefaults 0swaps
CSim: Compile error, see log file: /home/ando/challenge/bai-gaeshi/work/csim.log
ando@as101:~/challenge/bai-gaeshi
$
```

採点の流れ



- CSIM
 - コードをソフトウェアとして実行し、正しく動作することを確認します
- HLS
 - 高位合成を実行し、問題なくハードウェア（RTL）に変換されることを確認します
- CoSIM
 - RTLシミュレーションを実行し、正しく動作すること、実行サイクル数を確認します
- 論理合成
 - RTLを論理合成し、回路のリソース使用量を見積もります
 - 時間がかかるため仮採点ではスキップします
 - 実行するにはvhls-checkに-sオプションを渡します

採点がパスした例

- 採点結果、リソース使用量の見積もりが出ます
- 動作周波数とシミュレーションサイクル数から実行時間が見積もられます

```
gw.acri.c.titech.ac.jp - Tera Term VT
File Edit Setup Control Window Help
ando@as101:~/challenge/bai-gaeshi
$ ./vhls-check -f
Bytes of kernel code: 241
CSim: Pass
HLS: Pass
CoSim: Pass
Resource usage
  FF   : 1427
  LUT  : 2261
  DSP  : 3
  BRAM : 2
  URAM : 0
Clock period (ns): 1.460
Clock frequency (MHz): 684
Simulation cycle: 1199
Simulation time (ns): 1750.540
```

作成したコードを提出

- 作成したコードをHLSチャレンジに提出します
- 自動的に採点が行われます
 - 採点が完了するまで数十分かかります
- 採点がパスすれば、実行時間の短い順でランキングされます

bai-gaeshi 数を倍にせよ 編集 非公開 ダウンロード 2

開催中 初級 第0回チャレンジ 浮動小数点演算

チャレンジ 提出 提出一覧 ランキング

カーネルソース (kernel.cpp)

```
#include "kernel.hpp"

void kernel(const float in[1024], float out[1024], int size) {
    for (int i=0; i<size; i++) {
        out[i] = in[i] * 2.0f;
    }
}
```

150 / 10000

提出



ACRiルーム / HLSチャレンジ体験 (ハンズオン)

サーバーに SSH 接続

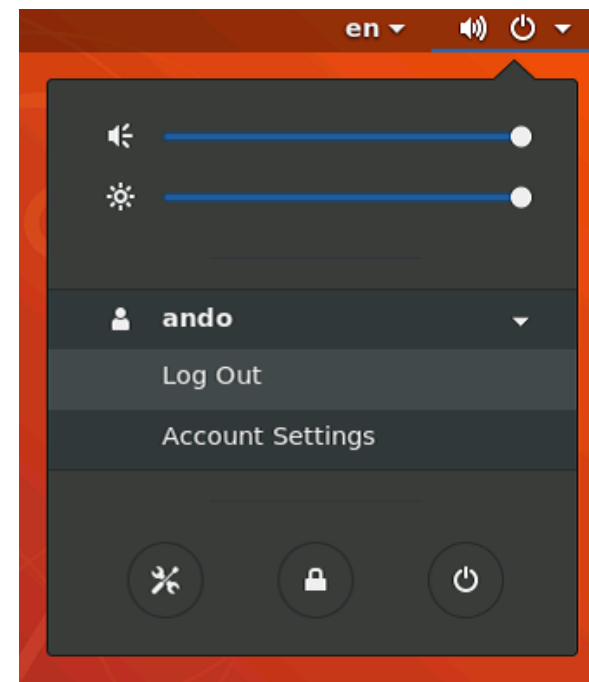
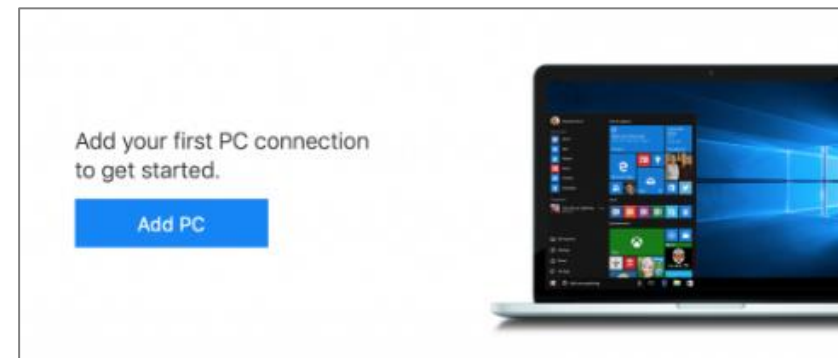
- ターミナルを開き以下のコマンドを入力

```
~$ ssh -L 13389:<サーバー名>:3389 <アカウント名>@gw.acri.c.titech.ac.jp
```

- <サーバー名>は as101 から as105 をランダムに選択
 - <アカウント名>は ACRI のアカウント名 (u_*)
 - The authenticity of host ... と表示されたら、yes と入力
 - password: と表示されたら、サーバー用のパスワードを入力してログイン
- ターミナルは開いたままにしておく

リモートデスクトップ接続

- Microsoft Remote Desktop を起動
 - Add PC ボタン
 - PC name に `localhost:13389` と入力して Add ボタンを押す
 - 追加された PC をダブルクリック
 - The identity of the remote PC ... と表示されたら、Connect をクリック
 - Enter Your User Account と表示されたら、ユーザー名とサーバー用のパスワードを入力してログイン
 - 使用後は右上のメニューからログアウトしてください



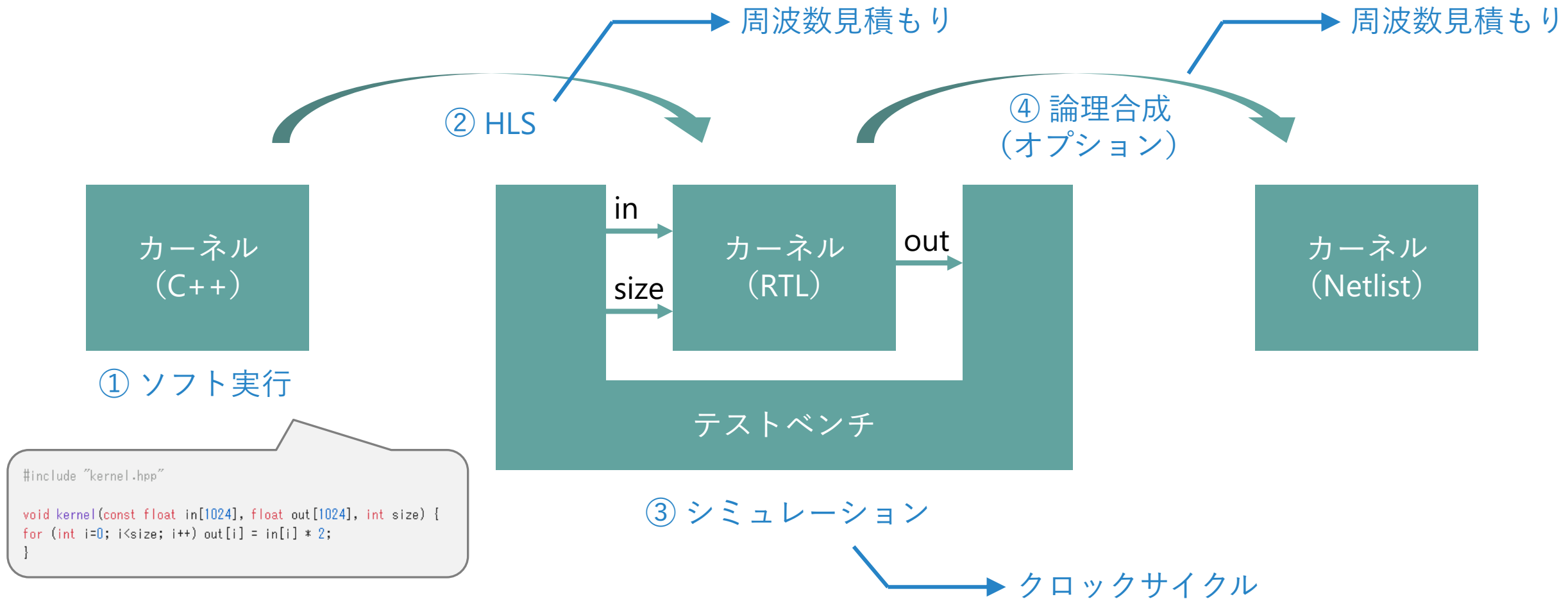
ハンズオン：ACRiルーム / HLSチャレンジ体験

- ブラウザで HLS チャレンジを開いてログインしてください
<https://acri-vhls-challenge.web.app>
- または「HLSチャレンジ」で検索
- bai-gaeshi3 を開いてコードを投稿してみましょう
- このスライドはこちらから参照できます
<https://github.com/acri-room/hls-challenge-labs/blob/master/slide.pdf>
- 演習を進めたい方はこちら
<https://github.com/acri-room/hls-challenge-labs>

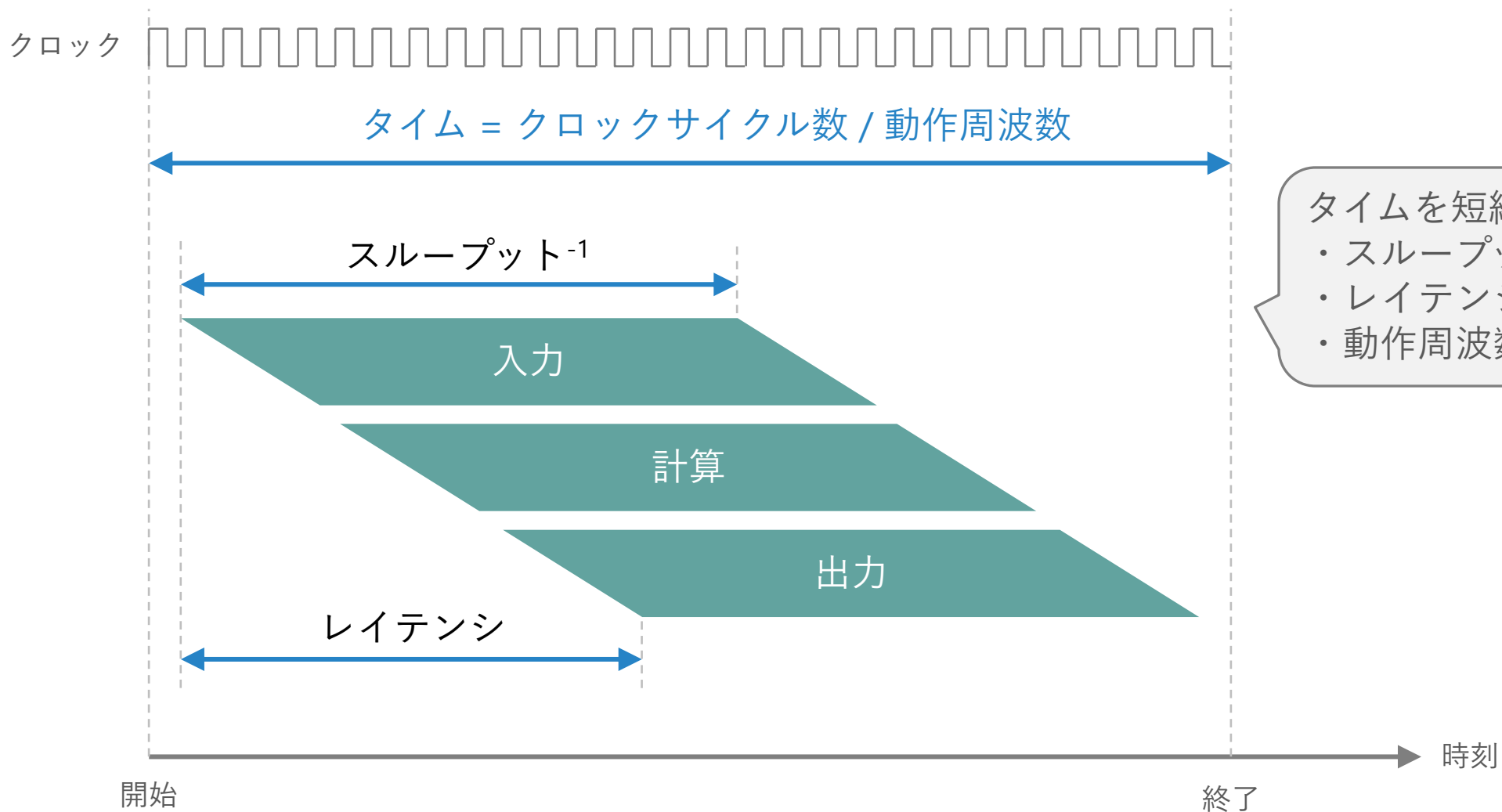


高速化テクニック紹介

採点の仕組み



タイム算出の仕組み

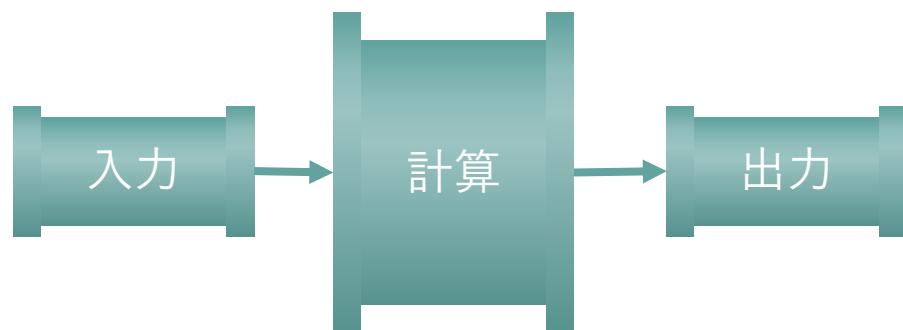


タイムを短縮するには

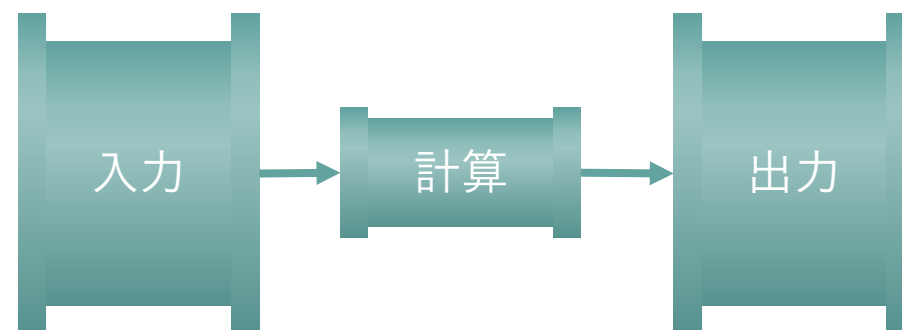
- ・スループット向上
- ・レイテンシ削減
- ・動作周波数向上

スループットを向上するには

- 入出力と計算のバランスが取れていること
 - 計算だけではなくインターフェースも重要
 - インターフェースの帯域にマッチする演算器を作る



入出力がボトルネックに



計算がボトルネックに

パイプライン化 (PIPELINE)

- ツールが自動で行ってくれる
 - ループ内のタスクが並列に実行される (タスク並列)

```
void kernel(const float in[1024], float out[1024]) {  
    for (int i = 0; i < 1024; i++) {  
        out[i] = in[i] * 2;  
    }  
}
```

パイプラインになる

- パイプラインにしたくない場合や、
パイプラインの詳細を制御したいときにプラグマを使う

```
void kernel(const float in[1024], float out[1024]) {  
    for (int i = 0; i < 1024; i++) {  
#pragma HLS PIPELINE off  
        out[i] = in[i] * 2;  
    }  
}
```

パイプラインにならない

データ並列化 (UNROLL)

- 演算器を増やして処理時間を短縮
 - 複数のデータを同時に処理する (データ並列)

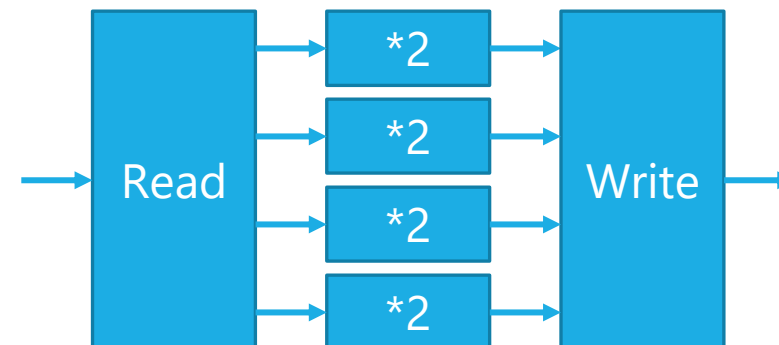
```
void kernel(const float in[1024], float out[1024]) {  
    for (int i = 0; i < 1024; i++) {  
        #pragma HLS UNROLL factor=4  
        out[i] = in[i] * 2;  
    }  
}
```

演算器が4つ生成され、ループは256回になる

UNROLLなしのとき



UNROLL factor=4のとき



データ並列化 (UNROLL)

- ループ回数が未知のとき
 - 余分なアクセスが発生してしまわないように32bit単位のインターフェースになる
 - インターフェースがボトルネックになり速くならない

```
void kernel(const float in[1024], float out[1024], int size) {  
    for (int i = 0; i < size; i++) {  
        #pragma HLS UNROLL factor=4  
        out[i] = in[i] * 2;  
    }  
}
```


データ並列化 (UNROLL)

- ループ回数が未知のとき
 - sizeがfactorの倍数であることが分かっている
または
 - 余分に読み書きしても問題ない
とき
 - `skip_exit_check`オプションにより終了条件チェックを削除
 - インターフェースが並列化される

```
void kernel(const float in[1024], float out[1024], int size) {  
    for (int i = 0; i < size; i++) {  
#pragma HLS UNROLL factor=4 skip_exit_check  
        out[i] = in[i] * 2;  
    }  
}
```

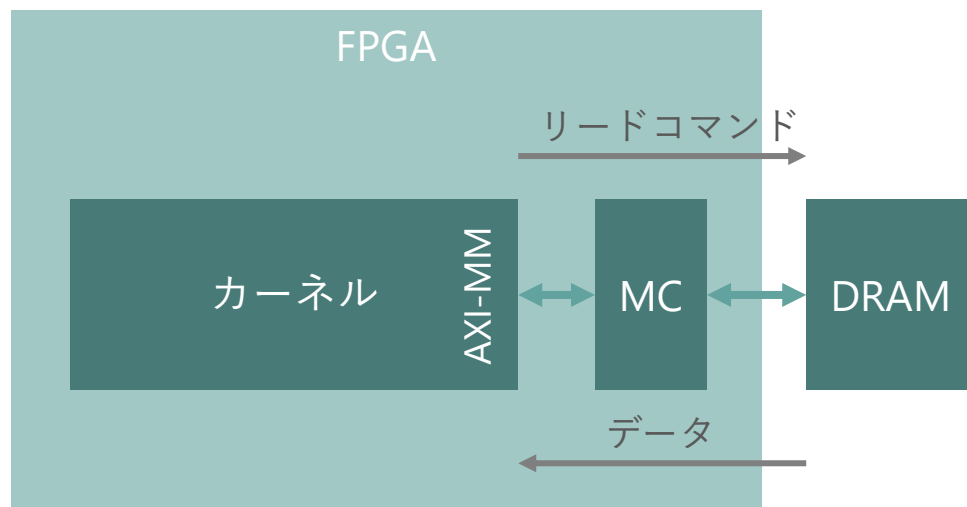
インターフェースのデータ幅

- ツールが自動でデータ幅を増やす場合もある
 - これまでの例ではfloat (32bit) を4つ同時、128bit単位で読み書きするインターフェースが自動で生成される
- データ幅を明示的に指定するにはカーネル引数の型を変更する必要がある
 - hls::vector型を使う
 - HLSチャレンジではカーネル引数の型は変更できない

```
void kernel(const hls::vector<float, 4> in[256], hls::vector<float, 4> out[256], int size) {  
    for (int i = 0; i < size; i++) {  
#pragma HLS UNROLL factor=4 skip_exit_check  
        out[i] = in[i/4][i%4] * 2;  
    }  
}
```

インターフェースの軽量化

- 配列引数へのアクセスはデフォルトでAXI-MMインターフェースになる*
 - MM : Memory Mapped (アドレスとデータのインターフェース)
 - 外部DRAMへのアクセスに相当するレイテンシが考慮される



* Vitis kernel flowのとき

インターフェースの軽量化

- インターフェースをAXI-Sに変更する
 - INTERFACEプラグマでインターフェースの種類を指定
 - ARRAY_PARTITIONプラグマでAXI-Sを並列化

```
void kernel(const float in[1024], float out[1024], int size) {  
    // インターフェースを指定するプラグマ  
    #pragma HLS INTERFACE mode=axis port=in  
    #pragma HLS INTERFACE mode=axis port=out  
    // 配列アクセスポートを並列化するプラグマ  
    #pragma HLS ARRAY_PARTITION variable=in type=cyclic factor=4 dim=1  
    #pragma HLS ARRAY_PARTITION variable=out type=cyclic factor=4 dim=1  
  
    for (int i = 0; i < size; i++) {  
        #pragma HLS UNROLL factor=4 skip_exit_check  
        out[i] = in[i] * 2;  
    }  
}
```



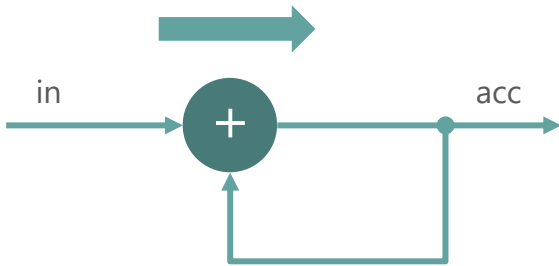
通常インターフェースは仕様で決まる。
HLSチャレンジでのみ有効なテクニック。

パイプラインのボトルネックを取り除く

- 浮動小数点数の累積演算がボトルネックに
 - パイプラインのII (Initiation Interval) を 1 にできない状況

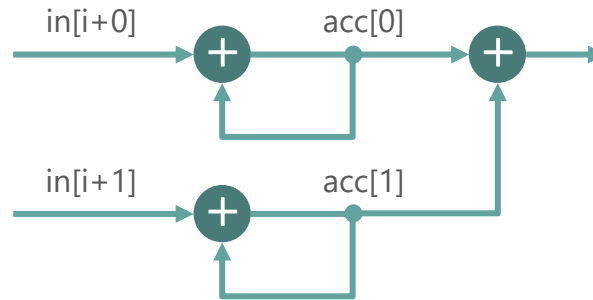
```
float acc = 0;
for (int i = 0; i < 1024; i++) {
    acc += in[i];
}
*out = acc;
```

2サイクル (II=2)



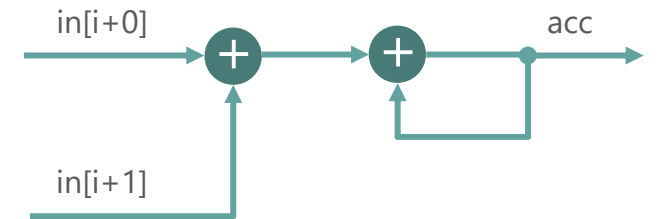
累積演算を並列化

```
float acc[2] = {};
for (int i = 0; i < 1024; i++) {
    #pragma HLS UNROLL factor=2
    acc[i % 2] += in[i];
}
*out = acc[0] + acc[1];
```



足し合わせてから累積

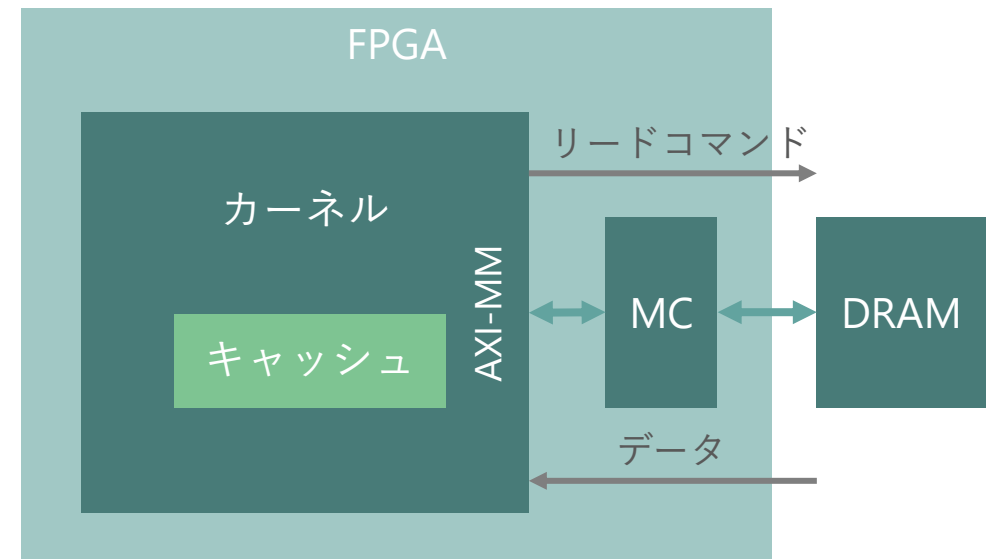
```
float acc = 0;
for (int i = 0; i < 1024; i += 2) {
    acc += in[i] + in[i + 1];
}
*out = acc;
```



外部へのアクセスを減らす

- 配列引数に繰り返しアクセスすると時間がかかる
 - カーネル内に一時配列を作る（CPUのキャッシュに相当する役割）

```
void kernel(const int in[1024], int out[1024]) {  
    // 一時配列  
    int tmp[1024];  
  
    // 入力  
    for (int i = 0; i < 1024; i++) { tmp[i] = in[i]; }  
  
    // バブルソート  
    for (int i = 0; i < 1024-1; i++) {  
        for (int j = 1024-1; j > i; j--) {  
            if (tmp[j] < tmp[j-1]) {  
                int t = tmp[j]; tmp[j] = tmp[j-1]; tmp[j-1] = t;  
            }  
        }  
    }  
  
    // 出力  
    for (int i = 0; i < 1024; i++) { out[i] = tmp[i]; }  
}
```



適したアルゴリズムを選択する

- simple-sort（データの並び替え）
 - バブルソート : 1.05M
 - Radixソート : 67.98k
 - バイトニックソート : 11.55k
- vector-mean-var（データの平均と分散）
 - ① 平均を計算してから分散を計算する
 - ② 「二乗の平均」と「平均の二乗」の差から計算する

$$\textcircled{1} \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

$$\textcircled{2} \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - \left(\frac{1}{n} \sum_{i=1}^n x_i \right)^2$$

高校数学の美しい物語「分散の意味と2通りの求め方・計算例」より

<https://manabitimes.jp/math/1081>

高速化テクニックまとめ

- パイプライン化 (PIPELINE)
- データ並列化 (UNROLL)
- インターフェースのデータ幅
- インターフェースの軽量化
- 外部へのアクセスを減らす
- パイプラインのボトルネックを取り除く
- 適したアルゴリズムを選択する



テキストに沿って課題に挑戦 (ハンズオン)

演習テキスト

- ブラウザでこちらのリンクを開いてください
- <https://github.com/acri-room/hls-challenge-labs>

