2020年度未踏ターゲット事業

# アプリケーション特化型イジングマシンの設計とFPGAへの実装

## ー　カスタムイジングコンピューティングの足掛かり　ー

川村一志（東工大）

**従来のイジングマシン**

幅広い組合せ最適化問題に対応できるよう設計されたイジングマシン

**本プロジェクトで開発したイジングマシン**

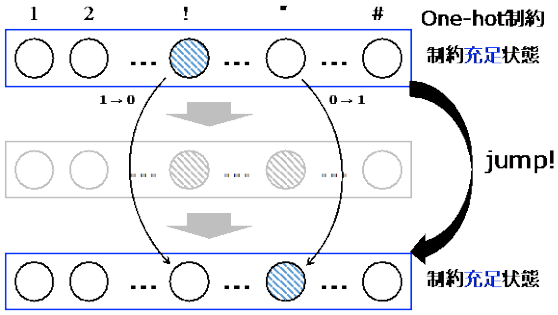
特定の組合せ最適化問題を効率的に解けるようにカスタマイズされたイジングマシン

＝＝

One-hot制約を含む問題 ｛ グラフ頂点彩色問題
ジョブショップスケジューリング問題
…

### 開発フロー

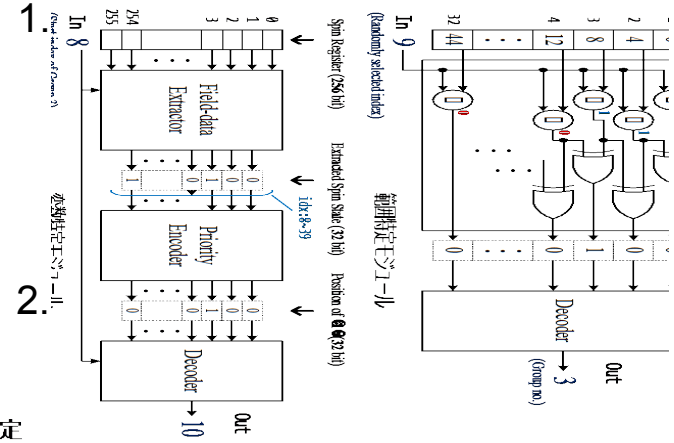
One-hot制約に対応した探索アルゴリズムの構築 → アルゴリズムを効率的に実現する並列処理アーキテクチャの設計 → FPGAへの実装

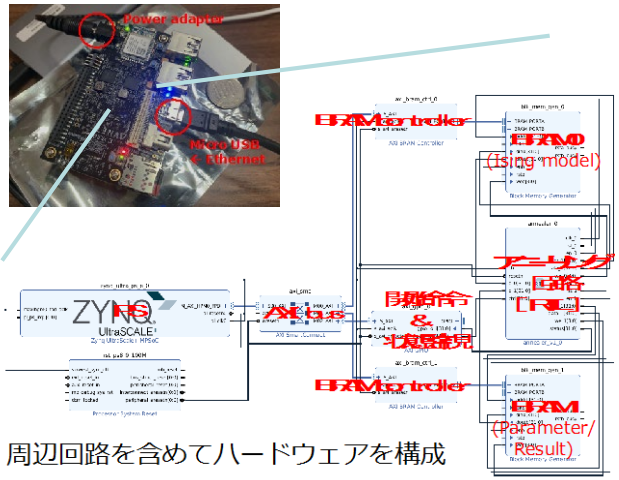



提案アルゴリズムにおける変数の更新過程

2変数同時更新のためには、以下の追加動作が必要

1. ランダムに選択された変数を含むOne-hot制約範囲の特定
2. 1で特定した範囲から現状態が1の変数を特定




周辺回路を含めてハードウェアを構成

## 実行環境

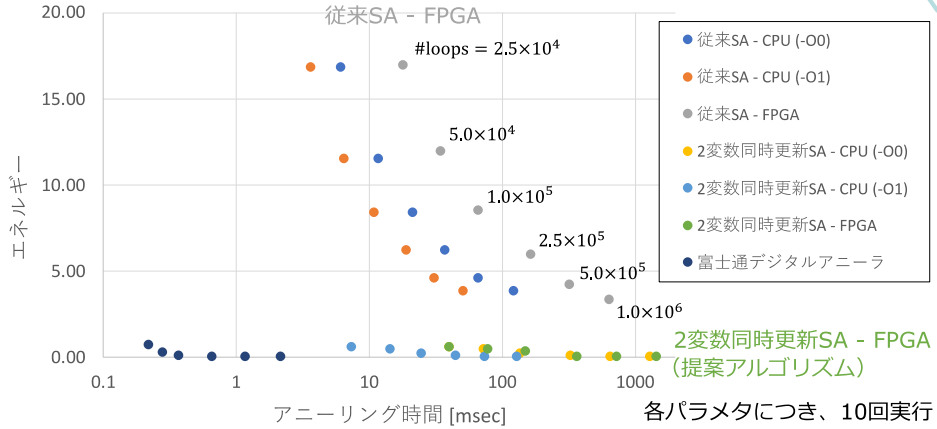
ssh

ユーザPC

Ultra96-V2（FPGA + CPU）

- CPU：ハードウェアの設定、データ送受信、実行開始命令
  +
- FPGA：アニーリング処理の実行

## 性能評価@93頂点5色のグラフ頂点彩色問題




各パラメタにつき、10回実行

- CPU：Intel Xeon W-2125 @4.00GHzでソフトウェアプログラムを実行
  注）実装はC++、疎結合モデル用に最適化、コンパイル時オプション：-O0 または -O1
- FPGA：Ultra96-V2に実装されたイジングマシン（16bit、512変数、全結合）
- 富士通デジタルアニーラ（第二世代）

## Jupyter notebook上での実行の様子

① ハードウェアの設定

```
In [21]: from pynq import Overlay
         from pynq import MMIO

         base = Overlay("/home/xilinx/pynq/overlays/201210_QUBO_Solver/design_1.bit")

         bram0_addr = base.ip_dict['axi_bram_ctrl_0']['phys_addr']
         bram1_addr = base.ip_dict['axi_bram_ctrl_1']['phys_addr']
         print(hex(bram0_addr))
         print(hex(bram1_addr))
         bram0 = MMIO(base_addr = bram0_addr, length = 256 * 1024)
         bram1 = MMIO(base_addr = bram1_addr, length =  16 * 1024)

         ctrl_addr = base.ip_dict['axi_gpio_0']['phys_addr']
         print(hex(ctrl_addr))
         ctrl = MMIO(base_addr = ctrl_addr, length = 0x1000)
```

② データ転送

```
In [22]: ## J & h
         for y in range(MAX_N):
             for x in range(MAX_N):
                 idx = y * MAX_N + x
                 if x != y:
                     bram0.write(4*idx, int(J[y][x]))
                 else:
                     bram0.write(4*idx, int(h[y]))

         ## spin states
         for i in range(pack_num):
             bram1.write(4*(i+8), int(state_compressed[i]))

         ## parameters
         bram1.write(4*0, int(beta_init))
         bram1.write(4*1, int(beta_delta))
         bram1.write(4*2, int(o_loop))
         bram1.write(4*3, int(i_loop))
         bram1.write(4*4, int(rand_value_init))
         bram1.write(4*5, int(N - 1))

         ## one-hot table
         for i in range(32):
             bram1.write(4*(i+16), int(tbl[i][0]))
```

③ 実行

```
In [24]: import time
         ctrl.write(4, 0)
         ctrl.write(0, 1)

         t0 = time.time()
         ctrl.write(4, 0xFFFFFFFF)
         while ctrl.read(0) != 0:
             pass
         t1 = time.time()
         elapsed_time_prop = t1 - t0

         print('Elapsed time (Prop) [sec]:', elapsed_time_prop)
         print('Elapsed time (Conv) [sec]:', elapsed_time_conv)
         print('Rate:', elapsed_time_prop / elapsed_time_conv)

         Elapsed time (Prop) [sec]: 0.06171679496765137
         Elapsed time (Conv) [sec]: 0.027403354644775391
         Rate: 2.252162035184865
```

④ 結果の取得

```
In [25]: read_state = np.ones((MAX_N,), dtype=np.int32)
         for i in range(pack_num):
             val = bram1.read(4*(i+48))
             val_str_list = list(format(val, '032b'))
             for j in range(32):
                 if val_str_list[31-j] == '0':
                     read_state[i*32+j] = 0
                 else:
                     read_state[i*32+j] = 1
         read_energy = np.dot(np.dot(read_state, J), read_state) // 2 + np.dot(h, read_state) + C
         print('-- read energy --')
         print(read_energy)

         -- read energy --
         0
```

☹

- 現状、FPGAマシンは富士通デジタルアニーラやCPUに性能面で劣る

☺

- One-hot制約に特化した提案アルゴリズムは従来よりも安定して低エネルギー状態に到達
- 提案アルゴリズムはFPGA・CPU間の性能差が僅か
  → FPGAは追加動作の実装オーバーヘッドが小さい
- 小型なFPGAであっても高性能CPUに匹敵しうる性能