巡回セールスマン問題のFPGAによる実装と高速化

佐藤 龍吾

はじめに　～実験概要とテーマ選択～

・EEIC後期実験課題12「FPGAを用いたアルゴリズム実装」
・アルゴリズムを選択し、Verilogで記述・FPGA上で実行
・ハード/ソフトの違いを理解・活用

はじめに　～実験概要とテーマ選択～

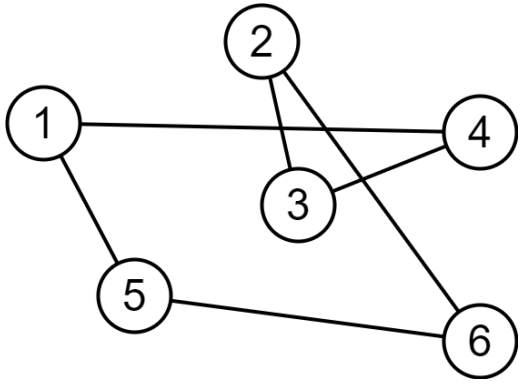
- ハードウエア実装最大のメリット: 並列計算が可能
  - ◯◯を計算するのと同時に△△も計算する
  - 複数の◯◯を同時並行して計算する

- アルゴリズムの並列計算可能性
  - 同じ(似た)計算を各部分で行う(行列計算、暗号計算など)
  - それぞれの部分が独立に計算可能(画像処理・音声処理など)

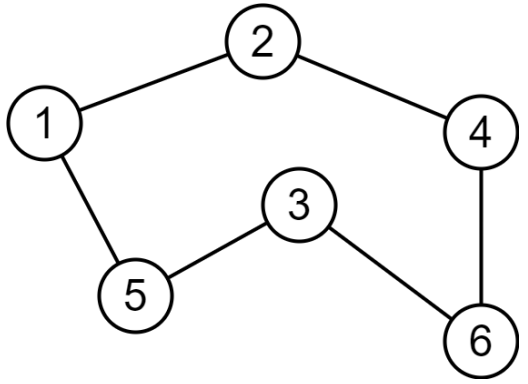
  - 各部分に依存関係がある探索問題・最適化問題は厳しいがち？
    - 「Aを変えたらBも変わる」とき、AとBを同時に変えるのは難しい

巡回セールスマン問題(TSP)

- N都市すべてを一度ずつ巡って戻ってくる閉路(ハミルトン閉路)
- 合計コストが最小のものを求めよ
- NP困難: 多項式時間で厳密解を出すのが困難
  - Held–Karp algorithm (いわゆるBit DP) $O(n^2 2^n)$ ← $n = 20$ 程度まで
  - n-近似解(最適解のn倍以下であることが保証されている解)を多項式時間で求める
  - なるべく最適解に近い解を求めるヒューリスティックアルゴリズム



← イマイチ



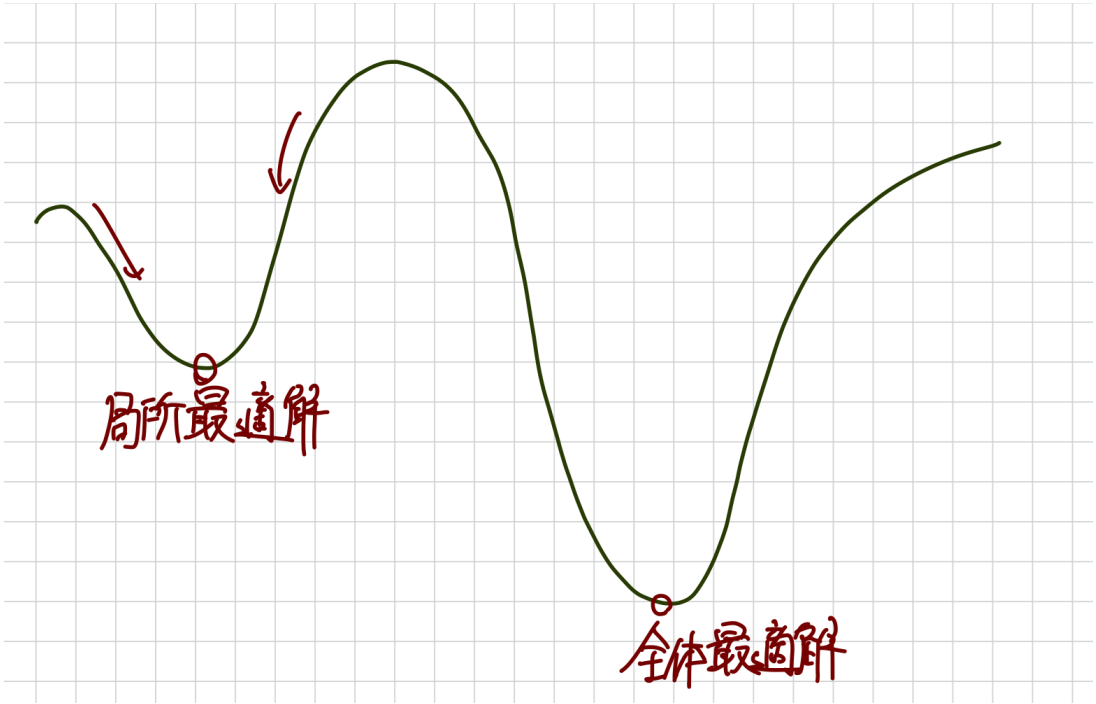
← イイカンジ

巡回セールスマン問題(TSP)

- ヒューリスティックアルゴリズム
- 試行錯誤的

- 状態をちょっと変えて(近傍)、改善(局所探索法)

- 局所最適解: 近傍の範囲で最適
  - 井の中の蛙
- 全体最適解: 全域で最適
  - 最終的なゴール



TSPのヒューリスティック解法例

局所探索法 (ちょっと変えて良くなってるかチェック)

山登り法

貪欲的に変更を採用

焼き鈍し法

悪くなる変更も確率的に採用

反復局所探索法

局所最適に達したら一部を崩す

進化計算
(遺伝的アルゴリズム)

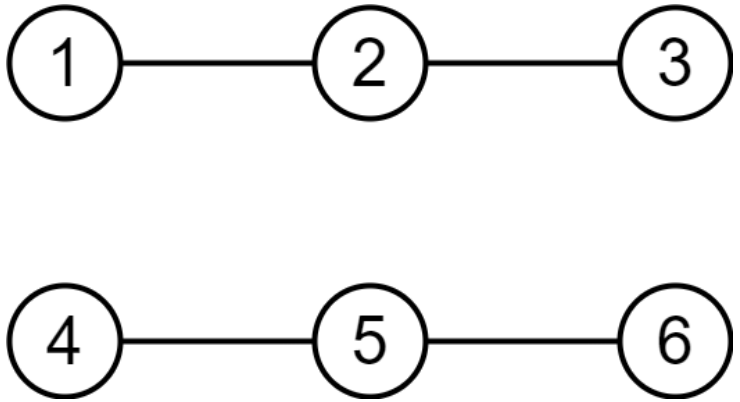
試行回数が重要！
→ 高速化需要が高い

実験テーマ

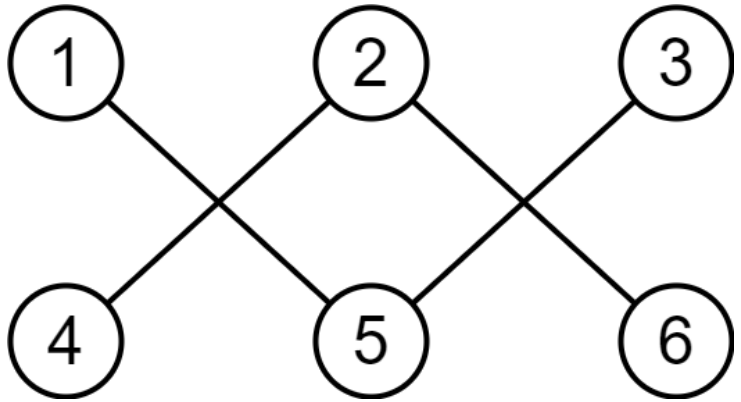
- FPGA上で、巡回セールスマン問題を解くプログラムを実装する
- 山登り法を採用。なるべく早く局所最適解に達するようにする
  - 試行回数が必要で、高速化需要が最も高い部分
  - この部分さえ早ければ、焼き鈍し法・反復局所探索法にアレンジすればより良い解を狙える
- 64頂点、256×256の2次元グリッド上のグラフを扱う

高速化のアイデア

- アルゴリズムのコア「経路の一部を変更し (近傍)、評価する」
  - 結果が良くなっていれば採用
- 変えた部分だけで採用すべきか判定できれば良い。
- 2点を入れ替えるべきか？ → 前後含めた6点で計算可能



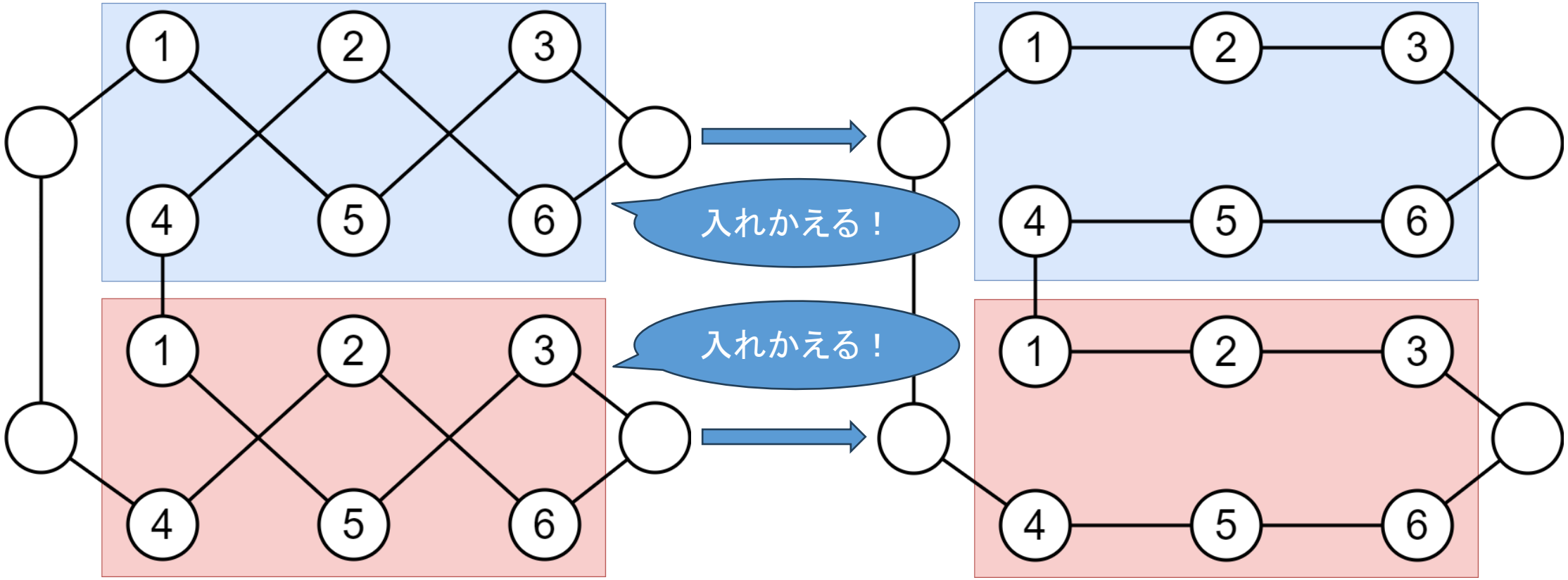
1 → 2 → 3 / 4 → 5 → 6



1 → **5** → 3 / 4 → **2** → 6

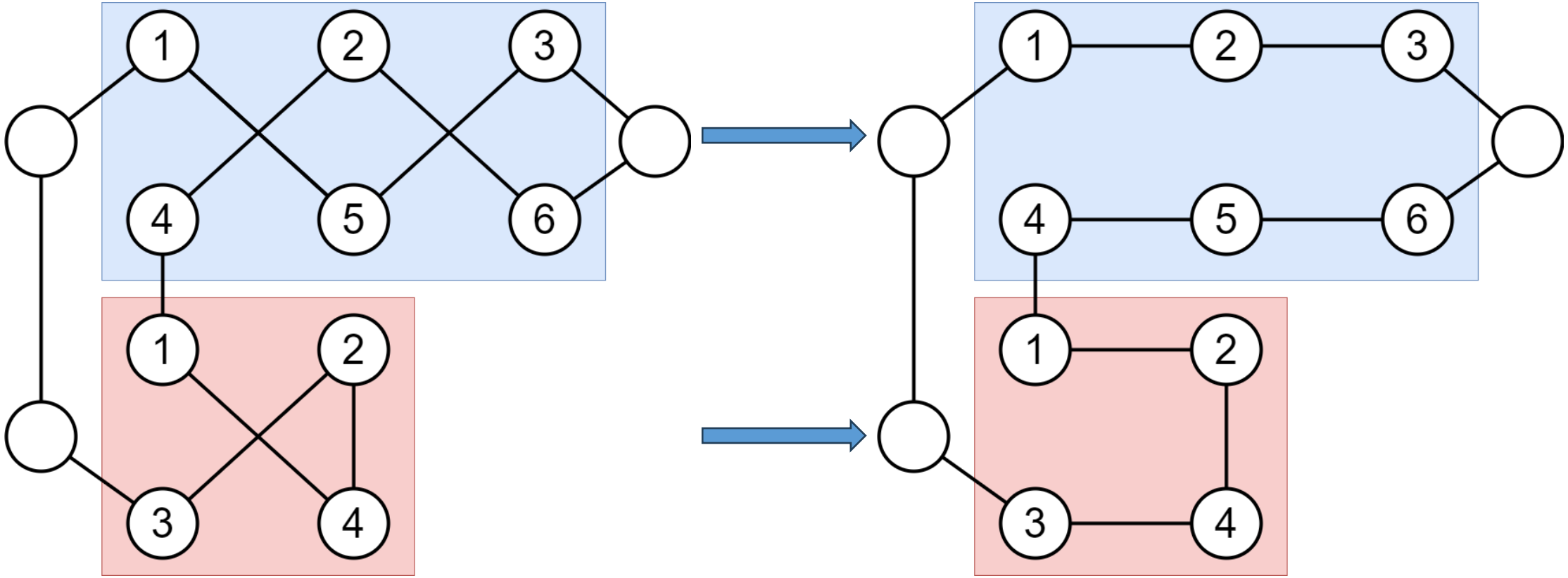
高速化のアイデア

・6点に重複がないように選択すれば、それぞれで判定・更新可能

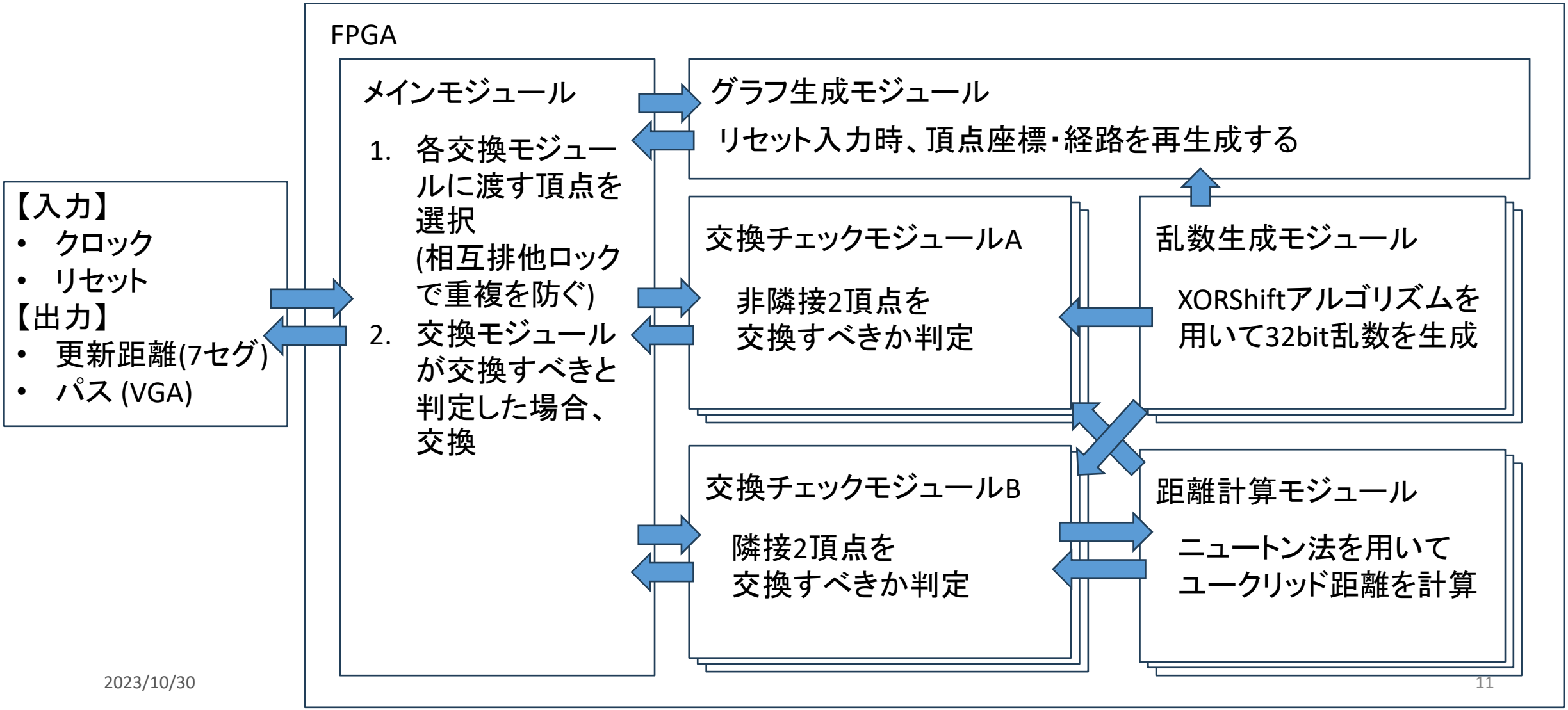


高速化のアイデア

・複数の入れ替えロジックを並列で走らせることも可能



ロジック設計


FPGA
メインモジュール
1. 各交換モジュールに渡す頂点を選択
(相互排他ロックで重複を防ぐ)
2. 交換モジュールが交換すべきと判定した場合、交換
グラフ生成モジュール
リセット入力時、頂点座標・経路を再生成する
交換チェックモジュールA
非隣接2頂点を
交換すべきか判定
乱数生成モジュール
XORShiftアルゴリズムを
用いて32bit乱数を生成
交換チェックモジュールB
隣接2頂点を
交換すべきか判定
距離計算モジュール
ニュートン法を用いて
ユークリッド距離を計算
【入力】
• クロック
• リセット
【出力】
• 更新距離(7セグ)
• パス (VGA)

アルゴリズム紹介　～乱数生成～

- XOR Shift: 疑似乱数生成アルゴリズム
- 周期が$2^{32}-1$ (32bitの場合)になる、とてもイイカンジの乱数
- JavaScriptの標準ライブラリの乱数はこれ
- 排他的論理和とビットシフトだけで計算可能
  x<=y;y<=z;z<=w;w<=(w^(w>>19))^((x^(x<<11))^((x^(x<<11))>>8));

アルゴリズム紹介　～距離計算～

- ユークリッド距離 $d = \sqrt{(\Delta x)^2 + (\Delta y)^2}$
- 整数aに対して、平方根$\sqrt{a}$を高速に求める必要がある

- ニュートン法: $a_{n+1} = \frac{a+a_n^2}{2a_n}$ は$\sqrt{a}$に収束する
- マンハッタン距離を初期値a_0として与えると収束が早い
- 256×256の範囲では、a_3の時点で整数範囲でほぼ一致する。
  - → 3回で打ち切る
  - 誤差が1以上(2未満)になってしまうのは、3パターン/65536

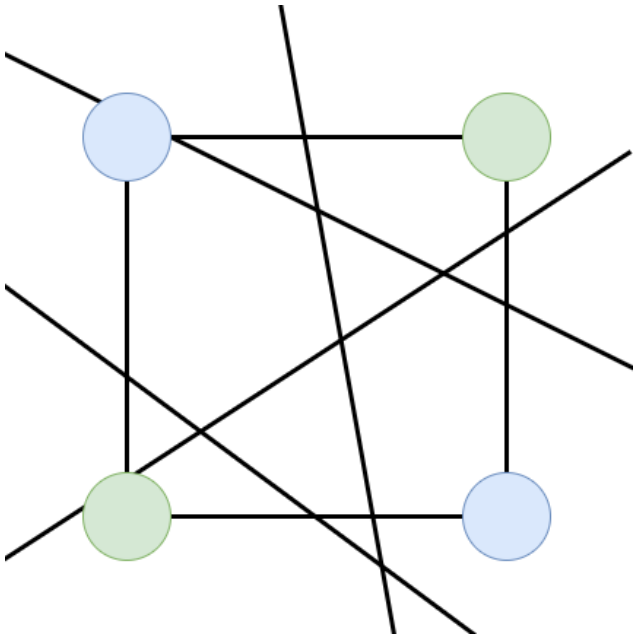
アルゴリズム紹介　～相互排他ロック～

・複数の交換モジュールが同一の頂点にアクセスするとバグる
・各交換モジュールは、頂点のアクセスにロックをかける必要がある

1. 交換する頂点を決める
   ・現時点で前後の頂点含めロックされていない必要がある
2. ロック配列にモジュール番号を書き込む
3. ロックに成功していれば処理を開始する
   ・使うすべての頂点がロックできていない場合、ロックできている部分を解除して1に戻る
4. 処理が完了したら、アンロック(ロック配列を-1に)する。1に戻る

アルゴリズム紹介　～直線を引く(未完)～

- $(x_1, y_1), (x_2, y_2)$の2点を結ぶ線分をVGAで表示
- 64本の線分のうち、(X, Y)を通る直線はあるか？
  - (課題3を流用するならば)1クロックで計算せよ
  - (よく考えたら、64クロックかければよかったな)
- 対角の2点が直線の反対側にあれば良い
  - $(aX + bY + c)\{a(X + 1) + b(Y + 1) + c\} < 0$
  - $\{a(X + 1) + bY + c\}\{aX + b(Y + 1) + c\} < 0$
  - のいずれかが成立すれば良い
- 9.5日目に実装しようとしたが、タイムオーバー



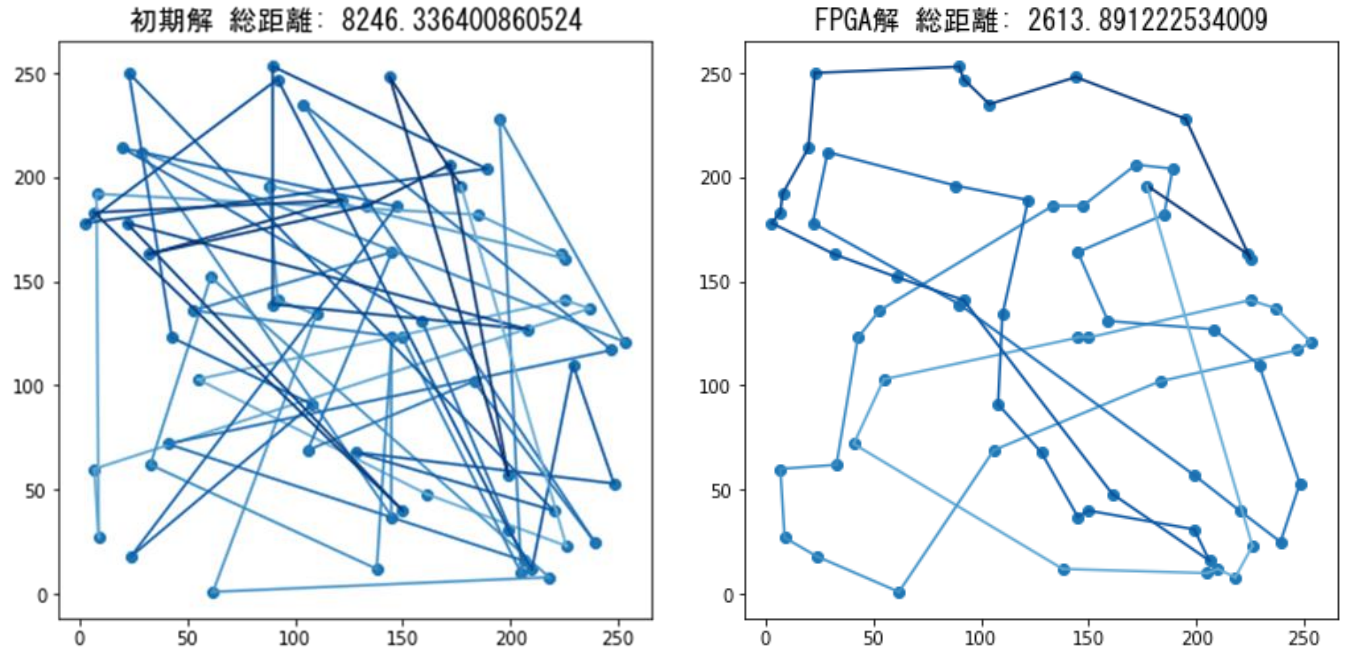
実行結果例 (テストベンチによる出力)

- 左図: ランダムな経路
- 右図: テストベンチ実行結果

- 並列度1: 19.4 kクロック
- 並列度2: 11.3 kクロック
- 並列度3: 6.5 kクロック
- 10MHzで動作させれば6.5ms

- 【参考スコア】
  - 貪欲解2265
  - (この方法を元にした)SA解2024

局所最適解の一つ
= (今回の)どの近傍でも
この状態から改善しない



結果分析 ～実装量～

ファイル名	機能	実装量
TSPTop.v / TSPTop_wrap.sv	トップモジュール	40行 + 57行
Tsp.sv	グラフ情報の管理 各交換モジュールの呼び出し 頂点スワップの実行	249行
Swap.sv	非隣接点交換モジュール	95行
Swap_adjacent.sv	隣接点交換モジュール	82行
Graph.sv	グラフ生成モジュール	43行
Distance.v	距離計算モジュール	37行
Xorshift.v	乱数生成モジュール	25行
Seg7.sv	7セグ16進出力モジュール	32行
		約660行

結果分析　～合成結果～

並列度 (非隣接点+隣接点)	最大動作周波数　[MHz]	使用ロジック数	使用演算器数
1 + 0	12.76	7470	8
0 + 1	12.69	6743	8
1 + 1	12.83	14821	16
2 + 1 (最終的に採用)	12.73	23152	24
3 + 1	×	50546	32

- 使用ロジック数はpow(2, 並列度)に比例 = 回路規模が**指数的に増大**
  - 各交換モジュールは4000程度(うち距離計算が2000)程度で、各モジュールの並列化に膨大な回路
- 使用演算器は並列度に比例(きれいに8個ずつ使用)

結果分析　～距離計算とクロック周波数～

頂点間距離の計算方法によって、クロック周波数は大きく変化した。
(以下、並列度1で実行した結果)

- マンハッタン距離 $d = |\Delta x| + |\Delta y|$: 47.8 MHz
- ユークリッド距離 $d = \sqrt{(\Delta x)^2 + (\Delta y)^2}$
  - ① function文 + assign で1クロック計算した場合(除算3回): 4.3 MHz
  - ② reg + clkで5クロックに分割した場合(1クロックあたり除算1回): 12.7 MHz
    - パイプライン化により5並列で計算できる
  - ③(実験終了後に検証) ②で、除算を最低限のビット長(18bit)で実行: 21.4 MHz

- 距離計算(特に除算)が全体のボトルネックになっている
  - Q. 全点対距離を前計算すれば？ → A. おっしゃるとおりです

結果分析　～CPUとの比較～

- Pythonで同様の処理を実装した場合、局所最適到達に平均45ms
  - 並列化なし・10MHzでも19.5ms。Pythonに勝利！
  - 並列度3・10MHzだと6.5ms。圧勝！
- 全力でチューニング・並列化をすれば、もう少し差をつけて勝てるはず
  - バグ取りに数日を要し、チューニングの時間がとれなかったのが悲しい
- ~~リソースさえあれば、並列度はガンガン上げられる！~~
  - **並列度に比例して回路規模が指数的に増大してしまう**
    - 頂点情報へのランダムアクセス回路がネックでは？→レジスタの分離・整理が必要
  - バースデーパラドックス: 乱択の衝突確率が増加しないか？

まとめ① 成果

実験時間の制約

～できたこと～

- 巡回セールスマン問題の解を改善するプログラムをハードウエア実装できた！
- 同等のCPU プログラムより高速に計算できた！
- リソース次第で並列化可能なアーキテクチャを作れた！

～できなかったこと～

- 局所解→最適解
  - 近傍採用の確率化(焼き鈍し)
  - 別ロジックによる解の部分リセット(Kick→反復局所探索法)
- さらなる高速化アイデア
  - 頂点選択ロジックの分離・FIFO化
  - 距離計算のアルゴリズム変更
  - 演算・レジスタサイズを最低限に
- VGA出力
  - ディスプレイ出力がされない謎

まとめ② 感想

たのしかったこと

- ハード特有の実装方法
  - 同時代入で一時変数を減らす
  - パイプライン化でO(NM)→O(N+M)
  - 必要な精度を意識したロジック
- みんなで残業するお祭り感
  - TAさんありがとうございました

つらかったこと

- デバッグ
  - 4MHzしか出ないのに、CLOCK50をそのまま与えていることに気づかないまま2日…
  - 「ロジックは合ってるはず」
  - 「テストベンチは完璧」
- 除算が遅すぎる
  - Just idea: 二分法なら除算いらないんじゃね？
- VGA出力