ISING REGISTER ALLOCATION
～アニーラによるレジスタアロケーション～

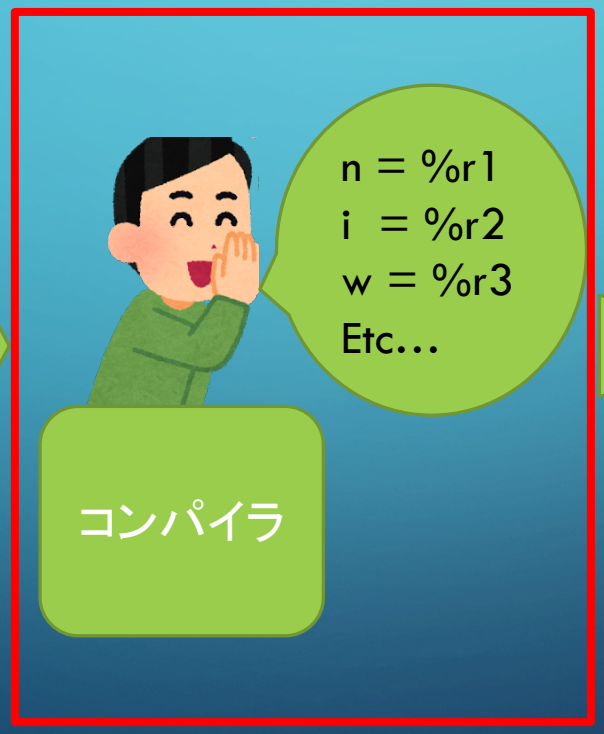
高屋敷 光（東北大学）, 熊谷 政仁（東北大学）

レジスタアロケーションとは

- コンパイル中にプログラム内の変数をCPUの物理レジスタに割り当てる処理のこと

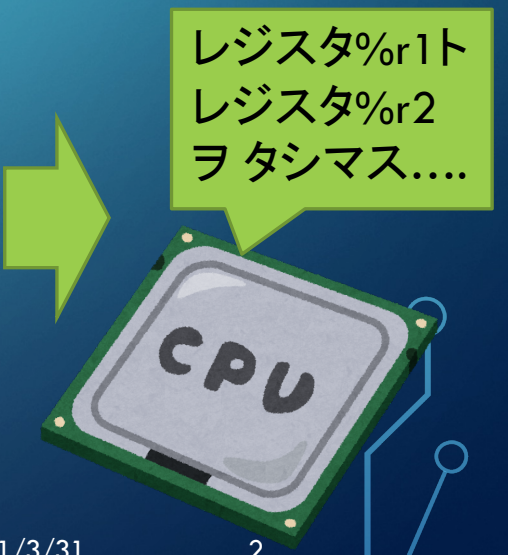
```
void solve(int * data){
    int n,i,w;
    for( n=MAX; n>1; n-- )
    {
        for( i=0; i<n-1; i++ )
        {
            if ( data[i]>data[i+1] )
            {
                w=data[i];
                data[i]=data[i+1];
                data[i+1]=w;
            }
        }
    }
}
```

高級言語




```
movl 4(%rdi,%rdx,4), %eax
cmpl %eax, %esi
jle .LBB0_4
movl %eax, (%rdi,%rdx,4)
movl %esi, 4(%rdi,%rdx,4)
jmp .LBB0_4
.p2align    4, 0x90
.LBB0_7:
movq %r10, %r8
andq $-2, %r8
xorl %eax, %eax
jmp .LBB0_8
.p2align    4, 0x90
            .
            .
            .
```

アセンブリ言語



レジスタアロケーションの難しさ

- コードには多数の変数が存在
- CPUが実際に持つレジスタの数は少ない

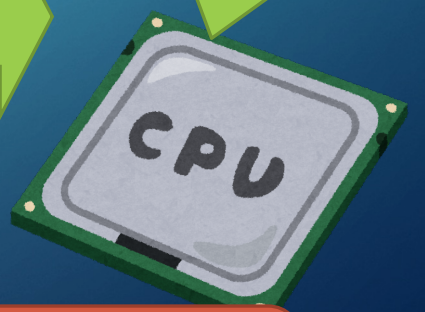
```
void solve(int * data){
    int n,i,w;
    for( n=MAX; n>1; n-- )
    {
        for( i=0; i<n-1; i++ )
        {
            if ( data[i]>data[i+1] )
            {
                w=data[i];
                data[i]=da
                data[i+1]
            }
        }
    }
}
```

変数100個くらい必要だわ

コンパイラ

えっ？このCPUレジスタ10個しかないの？

エッ？



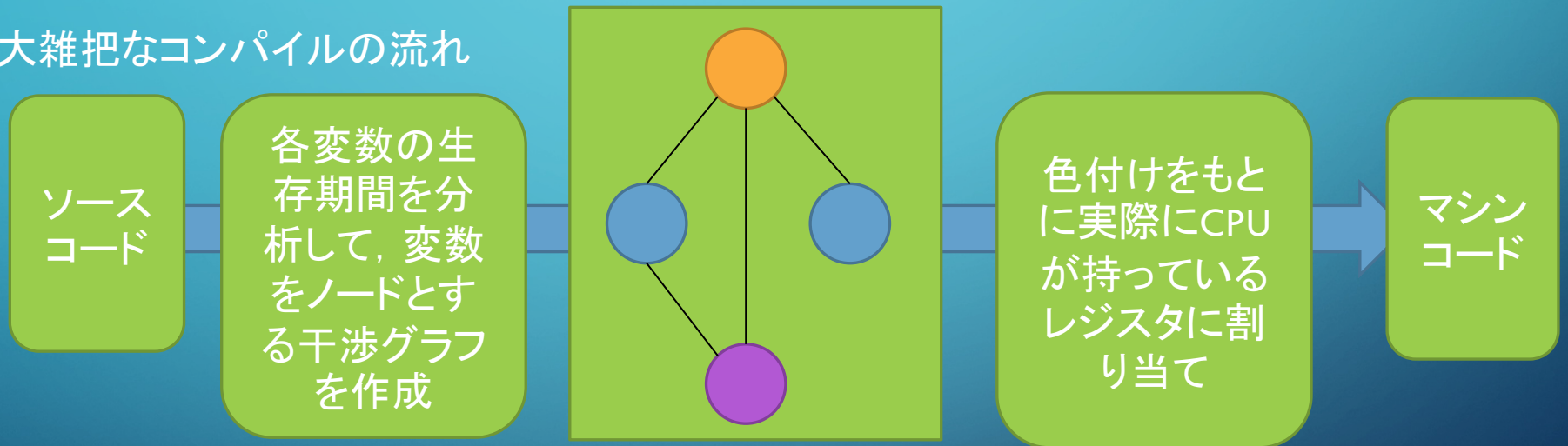
高級言語

- 複数の変数が同時に同じレジスタを使用しないように割り当て
- うまくアロケーションしないとレジスタのどれかを一旦メモリに退避(スピル)が必要
  →オーバーヘッドが発生

レジスタアロケーションと彩色問題

- 古典的にレジスタアロケーションは彩色問題を応用して解くことが出来る

大雑把なコンパイルの流れ



ノード間にエッジが存在する時は変数が同時に生存しているので，違うレジスタ（色）を与える必要がある

ただし，グラフを色づけることが出来る最小色数を求めるのはNP-hard問題！
そこで最適化問題としてQAで解くことを考える

QUBO定式化

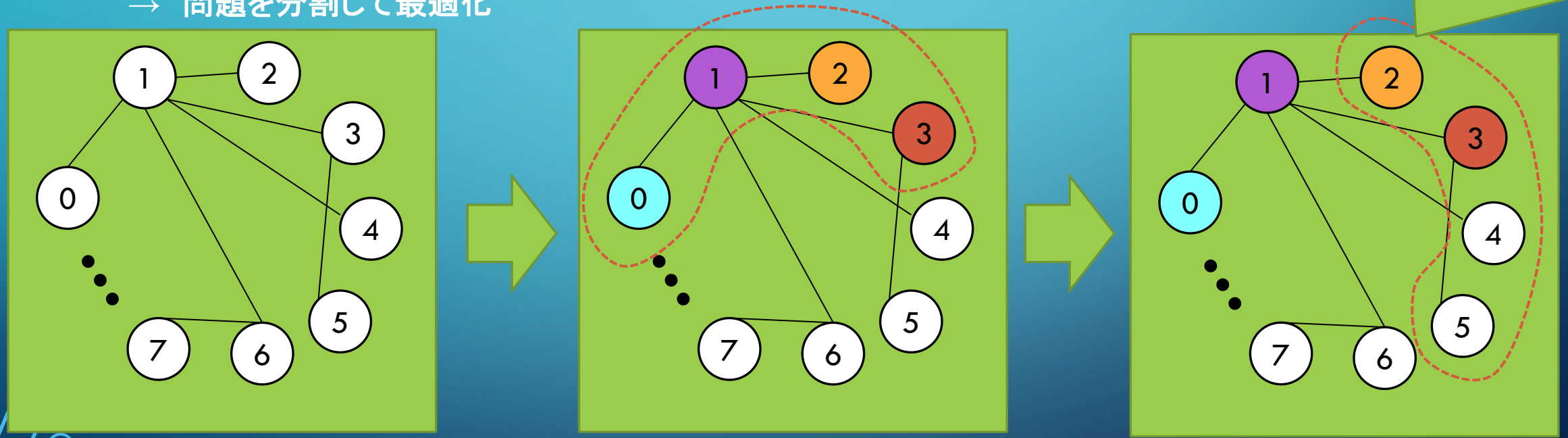
- $H = \sum_{i=0}^{N-1}\left(1 - \sum_{r=0}^{R-1} q_{i,r}\right)^2 + \sum_{r=0}^{R-1}\sum_{j \in E[i]} q_{i,r}q_{j,r} + \sum_{i=0}^{N-1}\sum_{r \notin A[i]} q_{i,r}$
  - 変数の数：N，レジスタの数：R
  - 変数iをレジスタrに割り当てる時：$q_{i,r} = 1$，割り当てない時：$q_{i,r} = 0$
  - $E[i]$：変数 i と生存期間が重なる変数の集合
  - $A[i]$：変数 i が割り当て可能なレジスタの集合
- 制約条件
  - 各変数をただ1つのレジスタに割り当て（ノードをただ1色で塗る）
  - 生存期間の重なる変数同士を別のレジスタに割り当て（エッジのあるノード間は別色）
    - できるだけスピルするレジスタを減らすため
  - 各変数を割り当て可能なレジスタに割り当て（ノードの色は色付け可能な色から選ぶ）
    - 変数ごとに型やCPUの構成によって割り当て可能なレジスタが異なるため

大規模な彩色問題の分割について

- 状況に応じて問題サイズを分割する必要
  - 変数が多くて量子ビット数を超える場合
  - グラフ構造に対してレジスタの数が明らかに足りない場合

→ **問題を分割して最適化**

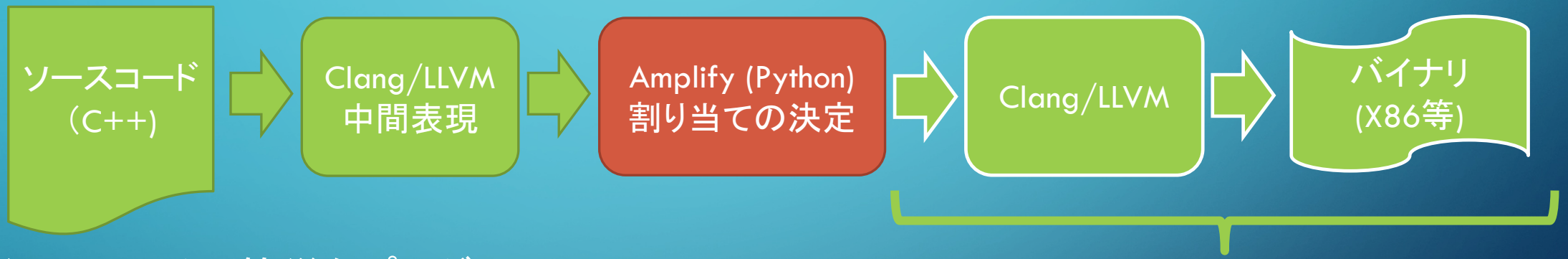
前の分割時に得られた解を含めて次の解を求める．割り当て可能なレジスタを絞る制約に解のみを与えることにより実現可能



特定の依存関係だけを取り出して，変数のレジスタ割り当てを決定

評価環境

- Clang/LLVMを利用して中間表現からレジスタ情報を抽出
  - コードの分析や最適化はLLVMのエコシステムを利用



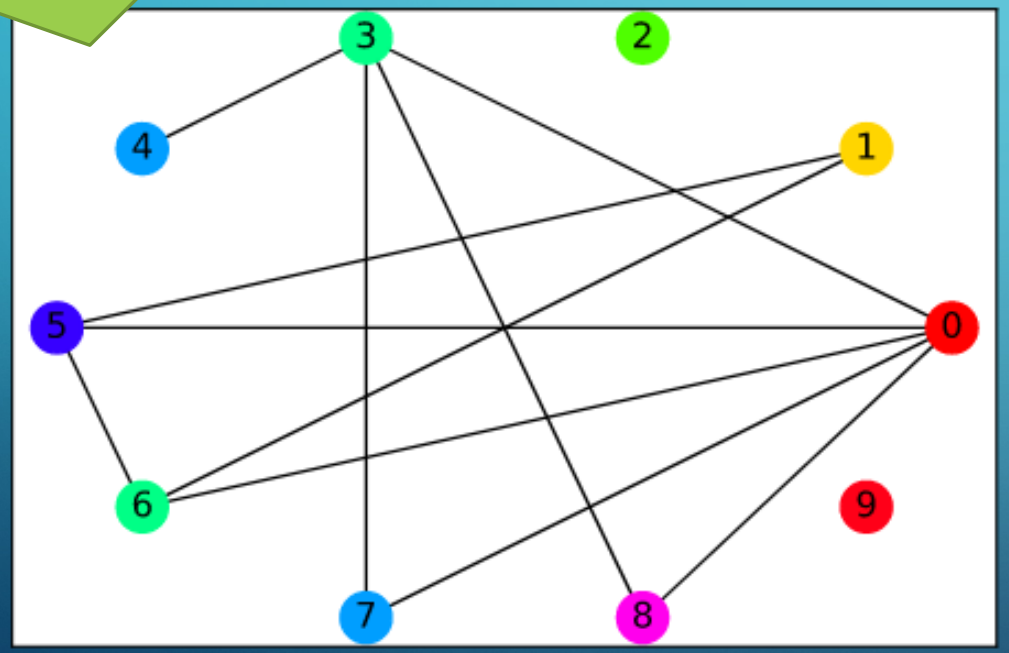
- ベンチマーク（簡単なプログラム）
  - モンテカルロ法，バブルソート
- レジスタはX86_64を想定
- 問題は4変数ずつに分割して求解

評価（モンテカルロ法）

```
for( j=0,Ni=0; j<jmax; j++ ){
   x=(double)rand()/max;
   y=(double)rand()/max;
   if ( x*x+y*y<=1.0 ) Ni++;
}

pi=(double)Ni/(double)jmax*4;
```

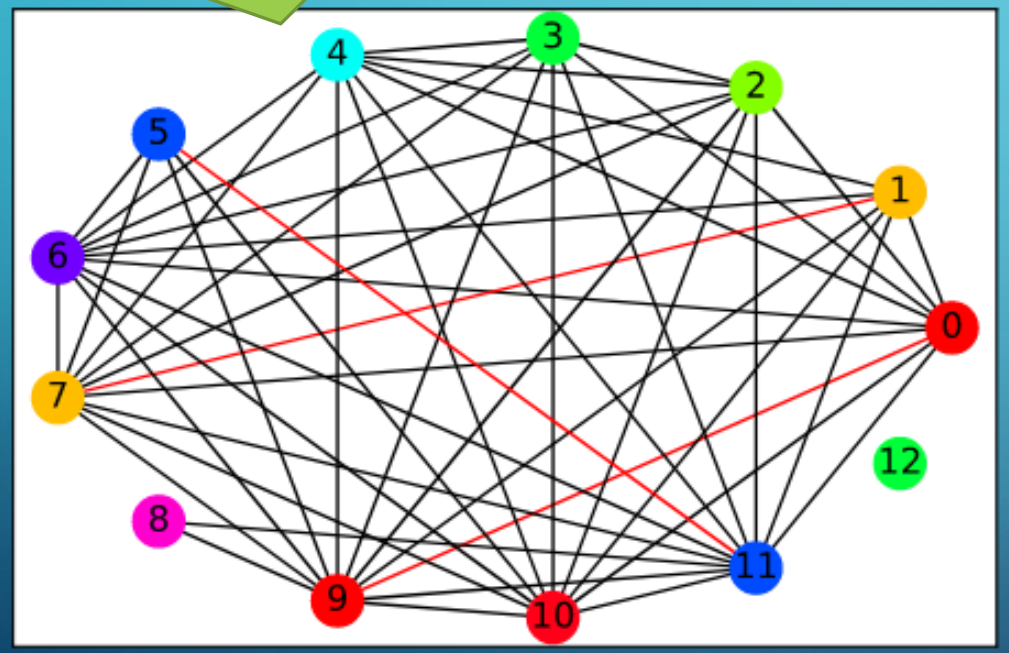
エッジは各変数(ノード)が重複する
生存期間を持つことを示す



- 全ての制約を満たした割り当てに成功
- 同色のノードは同じレジスタに割り当てられている
  - エッジがない（生存期間が重ならない）ため，制約を満たせている

評価（バブルソート）

エッジが黒：制約が満たされる
エッジが赤：制約が満たされない




```
void solve(int * data){
    int n,i,w;
    for( n=MAX; n>1; n-- )
    {
        for( i=0; i<n-1; i++ )
        {
            if ( data[i]>data[i+1] )
            {
                w=data[i];
                data[i]=data[i+1];
                data[i+1]=w;
            }
        }
    }
}
```

- 概ね制約を満たした解を求めることに成功
- 同じレジスタに割り当てられるが生存期間の重なる変数が存在
  - この場合スピルが発生する
  - 問題の分割方法に改善の余地あり

まとめ

- 背景
  - レジスタアロケーションについて
  - 彩色問題によるレジスタアロケーション
- QUBOを用いた彩色問題の定式化
  - 変数にレジスタが1つだけ割り当てられる制約
  - 生存期間が被る変数に別のレジスタを割り当てる制約
  - 各変数に割り当て可能なレジスタに応じて割り当てる制約
  - 問題が大きい場合の分割について
- 評価
  - LLVMから得られた変数情報・レジスタ情報をもとにした色付け評価
  - モンテカルロ法では正しく割り当てが出来ていた
  - バブルソートでは制約が破られる場合があった

今後の展望

- アルゴリズムに関して
  - スピルコストを考慮した彩色と，スピル無しには解けない問題でスピルさせるレジスタの選択アルゴリズムの作成
  - 解品質を維持したまま問題を分割する手法の考案
- 実用性に関して
  - QAを用いてレジスタアロケーションを行うLLVM Passの作成
  - FPGAの高位合成時のリソース割り当てなど他の手法へ応用など
- 評価に関して
  - ほかのレジスタアロケーション手法との実行時間・解品質の比較
- その他
  - いつか論文にしたい(LLVMやコンパイラ最適化など，この辺のドメインに詳しい人がもし居たらご連絡ください)

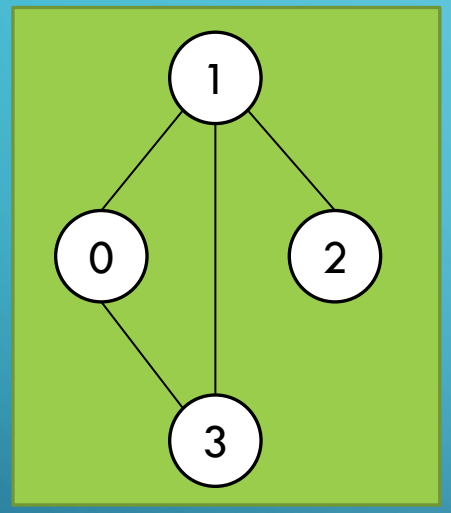
提出物の工夫ポイント！

- 新規性
  - 量子コンピュータを用いたレジスタアロケーション手法の例はまだ無い
  - 彩色問題を色付け制約付きで定式化した例も無い
- 進歩性
  - 彩色問題によるレジスタアロケーションは品質は良いが柔軟性及び求解コストの側面から不利とされていた
    - 本手法は様々な制約を当てはめたうえで（たぶん）高速に解を求められる！
- 技術力
  - 色付け制約付き彩色問題のQUBO定式化
  - LLVMのバックエンドから中間表現を抽出しベンチマークを作成
  - Amplifyを用いて手法のProof of workを作成
- 実用性
  - 色付け制約により割り付けレジスタの指定，量子ビット数が不足する時の問題分割，色不足の際の問題分割を考慮
  - 制約違反を起こさず彩色が行えているため，あとはLLVM側に結果を戻せば実際に使用可能

以下，予備スライド

量子ビットによるレジスタ割り当ての表現

4つの変数を4本のレジスタに割り当て

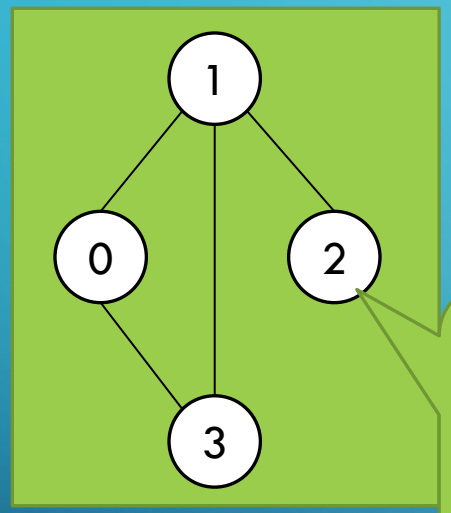


4×4個の量子ビットで表現

		レジスタ r			
		0	1	2	3
変数 i	0	$q[0][0]$	$q[0][1]$	$q[0][2]$	$q[0][3]$
	1	$q[1][0]$	$q[1][1]$	$q[1][2]$	$q[1][3]$
	2	$q[2][0]$	$q[2][1]$	$q[2][2]$	$q[2][3]$
	3	$q[3][0]$	$q[3][1]$	$q[3][2]$	$q[3][3]$

変数iをレジスタrに割り当てる時，　$q[i][r] = 1$
割り当てない時，$q[i][r] = 0$

各変数をただ一つのレジスタに割り当てる制約



レジスタ4本のうち，1つにしか割り当てられない

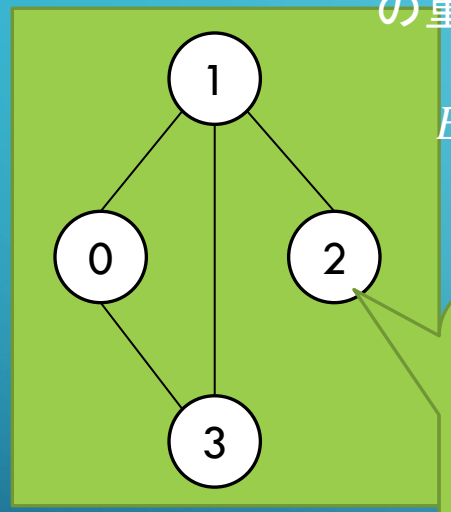
		レジスタ			
		0	1	2	3
変数	0				
	1				
	2	1	0	0	0
	3				

$$\sum_{r=0}^{R-1} q_{2,r} = q_{2,0} + q_{2,1} + q_{2,2} + q_{2,3} = 1$$

制約条件： $\sum_{r=0}^{R-1} q_{i,r} = 1, \quad i \in \{0, 1, \cdots, N-1\}$

QUBO： $H = \sum_{i=0}^{N-1} \left(1 - \sum_{r=0}^{R-1} q_{i,r}\right)^2$

生存期間の重なる変数同士を別のレジスタに割り当てる制約



各変数が生存期間の重なる変数

$E[0] = [1, 3]$
$E[1] = [0, 2, 3]$
$E[2] = [1]$
$E[3] = [0, 1]$

変数1番とは同じレジスタにならない

		レジスタ			
		0	1	2	3
変数	0				
	1	1	0	0	0
	2	0	0	1	0
	3				

同じレジスタでは，変数1番と2番の両方が1になることはない

$$\sum_{j \in E[2]} q_{2,0} q_{j,0} = q_{2,0} q_{1,0} = 0$$

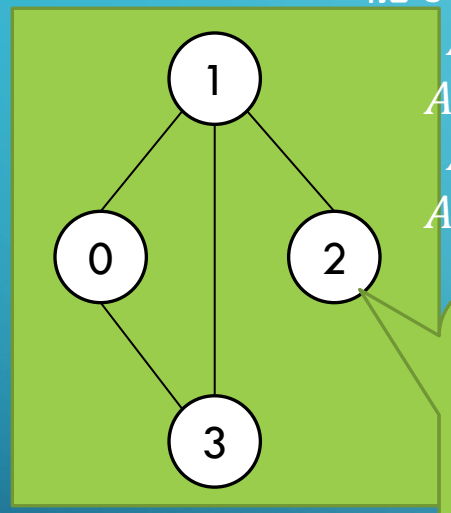
制約条件： $\sum_{j \in E[i]} q_{i,r} q_{j,r} = 0, \quad r \in \{0, 1, \cdots, R-1\}$

QUBO： $H = \sum_{r=0}^{R-1} \sum_{j \in E[i]} q_{i,r} q_{j,r}$

各変数が割り当て可能なレジスタに関する制約

各変数が割り当て可能なレジスタ番号

$A[0] = [0, 1]$
$A[1] = [0, 2, 3]$
$A[2] = [0, 3]$
$A[3] = [1, 2, 3]$



レジスタ4本あるけど，0番と3番にしか割り当てできない

		レジスタ			
		0	1	2	3
変数	0			0	0
	1		0		
	2		0	0	
	3	0			

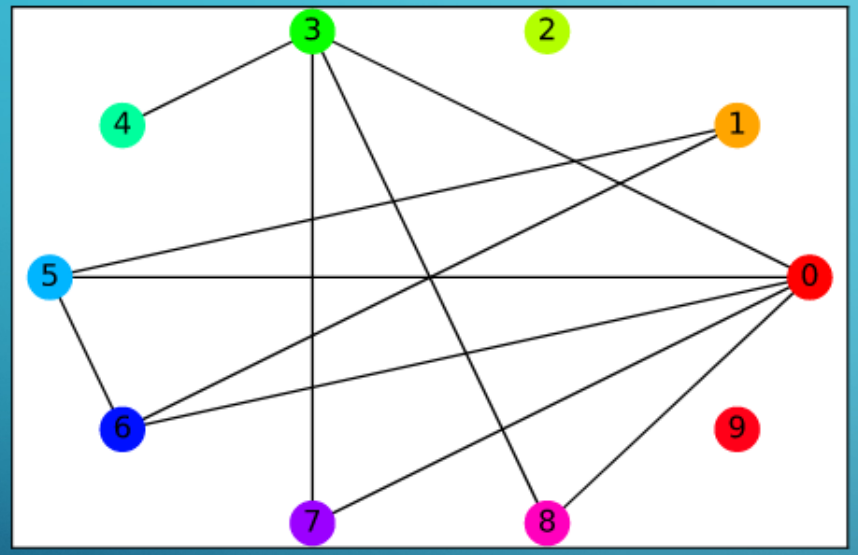
$A[2]$ に入ってない1番と2番は0になるようにする

$$\sum_{r \notin A[2]} q_{i,r} = q_{2,1} + q_{2,2} = 0$$

制約条件： $\sum_{r \notin A[i]} q_{i,r} = 0, \quad i \in \{0, 1, \cdots, N-1\}$

QUBO： $H = \sum_{i=0}^{N-1} \sum_{r \notin A[i]} q_{i,r}$

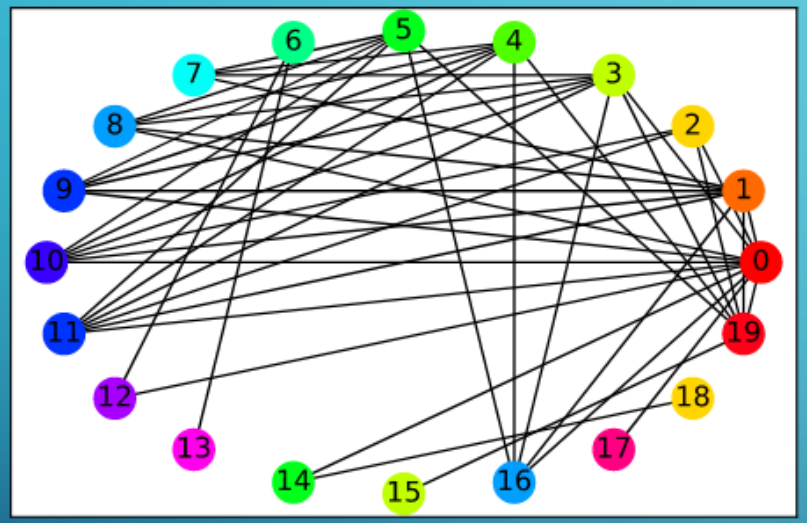
評価（モンテカルロ）



LLVMから抽出した，各変数が利用可能なレジスタ番号
0: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
2: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
3: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
4: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
5: [1, 2, 9, 10, 11, 12],
6: [31, 32, 33, 34, 35, 36],
7: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
8: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
9: [37, 38, 31, 32, 33, 34, 39, 40, 41, 35, 36, 42, 43, 44, 45]

- 全ての変数が，それぞれの利用可能レジスタに正しく割り当て
- 割り当てられたレジスタが全て異なるため，オーバーヘッドが発生しない

評価（シンプソン）



- 変数が多くても制約を守ってレジスタを割り当てられている

LLVMから抽出した，各変数が利用可能なレジスタ番号

0: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
1: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
2: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
3: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
4: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
5: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
6: [31, 32, 33, 34, 35, 36],
7: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
8: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
9: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
10: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
11: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
12: [2, 3, 4, 5, 9, 10],
13: [31, 32, 33, 34, 35, 36],
14: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
15: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
16: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
17: [2, 3, 4, 5, 9, 10],
18: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
19: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]