HLS Final Project

Simulated Quantum Annealing

Team 9
R09922190 王祥任
R09922187 鄭又愷
R09922150 洪崗竣

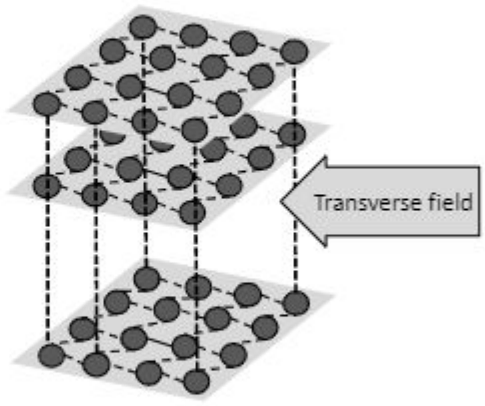
[Github]

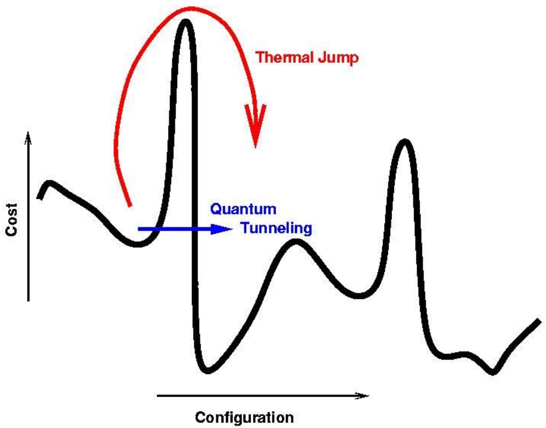
Outline

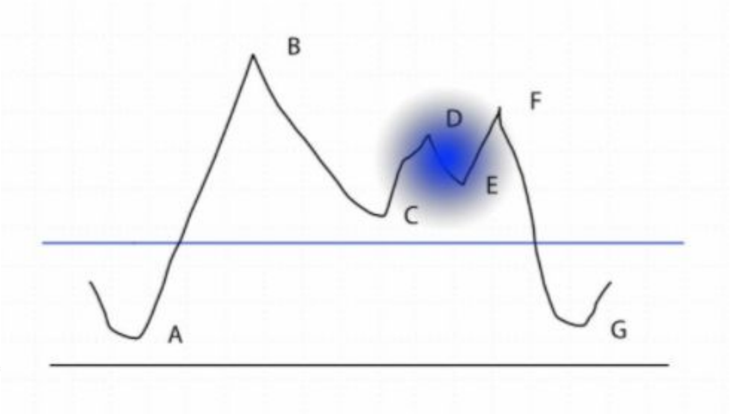
- **Introduction**
- Analysis
- Implementation
- Evaluation
- Guideline of mapping real world problem - take sudoku as example
- Conclusion

Introduction - Simulated Quantum Annealing

- Solving combinatorial optimization problem
- Good for the problem with high and thin wall
    - Tunneling
- As the number of spins increased, the elapsed time grows dramatically.
    - Time Complexity : O(TSS)







Analogy of quantum annealing

Introduction - Target Platform and Our Goal

- Target Platform
  - PYNQ-Z2
- Reproduce this paper with Xilnx HLS. We will also modify the design according to our environment.

Highly-Parallel FPGA Accelerator for Simulated Quantum Annealing

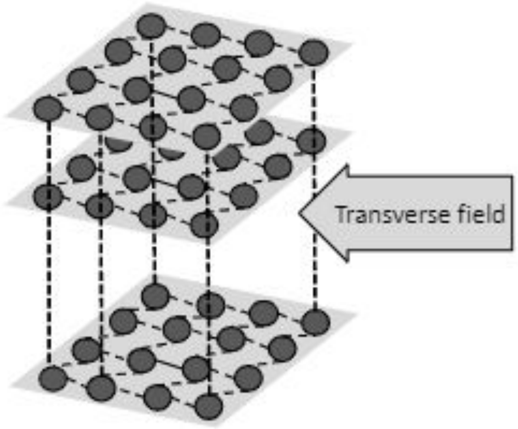
Hasitha Muthumala Waidyasooriya, *Member, IEEE,* and Masanori Hariyama, *Member, IEEE*

Outline

- Introduction
- **Analysis**
- Implementation
- Evaluation
- Guideline of mapping real world problem - take sudoku as example
- Conclusion

Analysis - Pseudocode of Naive Implementation

```
For t in 1 ~ #T: # for each trotter slice (solution)
    For i in 1 ~ #S: # calculate ΔH for each spin
        ΔH = h[i]
        For j in 1 ~ #S:
            ΔH = ΔH + Jcoup[i, j] × trotters[t, j]
        ΔH = ΔH - Jperp × #T × (trotters[t-1, i] + trotters[t+1, i])
        ΔH = ΔH × 2 × trotters[t, i]

        # flip with probability
        if e^(-Beta × ΔH) > randomNumber:
            trotters[t, i] = -trotters[t, i]
```



The energy cost of each trotter slice (our final solution) will gradually decrease during the annealing process.

Time Complexity: O(T×S×S)

Analysis - Memory Size of J Coupling

The space complexcity of the **J coupling** is **O(N×N)**

- $1024 \times 1024 \times 4$ Bytes = **4 MB**

The on-chip block ram that PYNQ-Z2 has is only about 0.6152 MB

- $280 \times 18 \times 1024$ Bits ≈ **0.6152 MB**

...

```
For i in 1 ~ #S: # calculate ΔH for each spin
    ΔH = h[i]
    For j in 1 ~ #S:
        ΔH = ΔH + Jcoup[i, j] × trotters[t, j]
```

...

Analysis - Access Pattern of J Coupling

Naive implementation reads the full J coupling **(#T × #S)** times.

For ***i-th Spin*** in different trotters, they use the same J coupling array.

- Reuse of ***Jcoup[i]*** between different trotters is possible.

```
For t in 1 ~ #T: # for each trotter slice (solution)
    For i in 1 ~ #S: # calculate ΔH for each spin
        ΔH = h[i]
        For j in 1 ~ #S:
            ΔH = ΔH + Jcoup[i, j] × trotters[t, j]
```

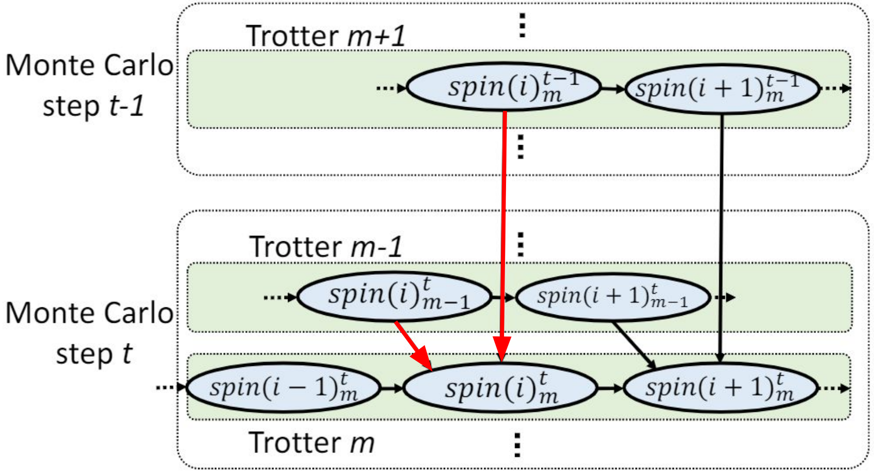
The order of the trotter won't affect the selection of J coupling.

Analysis - Data Dependency (Finding Parallelism)

The **update of Spin(m, i) at Iteration(t)** relys on

- `Intra-Trotter Spins (Summation)`
- `Spin(m  , 1   ~ i-1) at Iteration(t)`
- `Spin(m  , i+1 ~ N  ) at Iteration(t-1)`

- `Inter-Trotter Spins (Tunneling)`
- `Spin(m-1, i ) at Iteration(t)`
- `Spin(m+1, i ) at Iteration(t-1)`



Also, there exists the **sequential order**

- Which trotter update first
- Which spin update first

Ref: Highly-Parallel FPGA Accelerator for Simulated Quantum Annealing

Analysis - Random Number

- HLS doesn't support the standard C++ library of the random number generator.
- Need lots of it (#Iter × #T × #S)
- Two possible solutions :
  - Implement pseudo random number generator in HLS
    - Which PRNG is most suitable ?
  - Generate the random number in the host then send to the kernel.
    - I/O Latency is a problem.

Analysis - Simple Summary

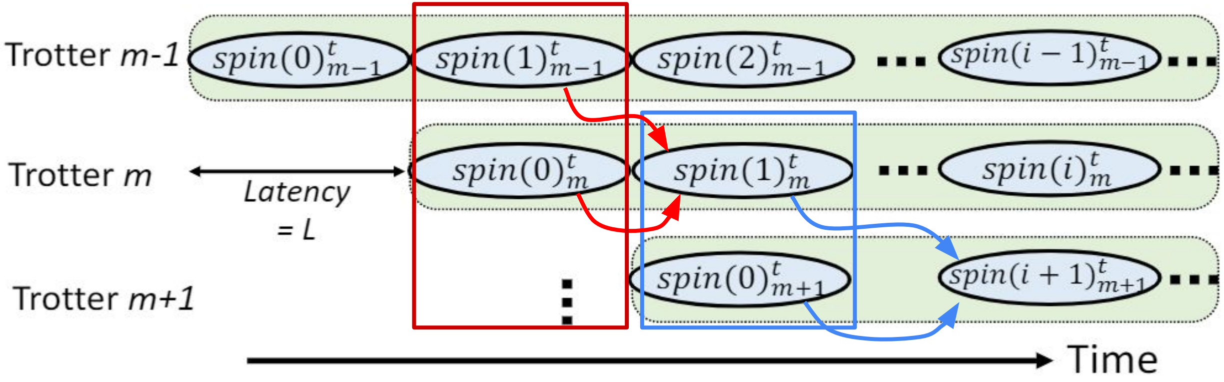
- Reduce the utilization of the on-chip memory for J coupling
- Design interface for J coupling to exploit its access pattern
- Exploit inter-trotter parallelism
- Exploit intra-trotter parallelism
- Generate random number

Outline

- Introduction
- Analysis
- **Implementation**
- Evaluation
- Guideline of mapping real world problem - take sudoku as example
- Conclusion

Implementation - Proposed Architecture (Trotter Unit)

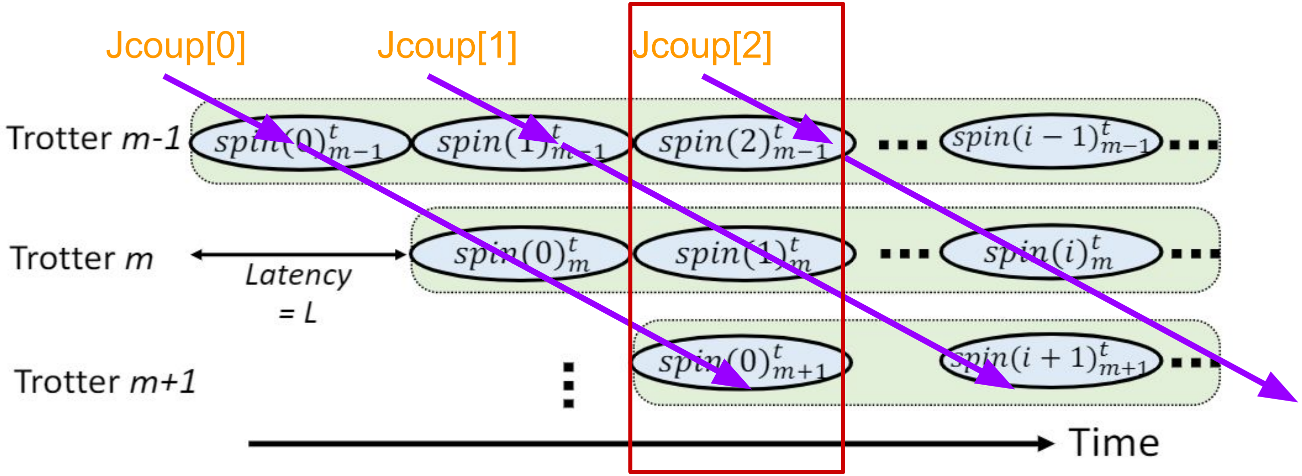
- Each Trotter Unit proccesses the spins in the one trotter in sequential order.
  - **Hold the intra-trotter dependencies.**
- Each Trotter Unit starts with different initial latency to avoid the data dependency between trotters.
  - **Hold the inter-trotter dependencies and exploit the inter-trotter parallelism.**



Ref: Highly-Parallel FPGA Accelerator for Simulated Quantum Annealing

Implementation - Interface and J coupling

- Reuse of ***Jcoup[i]*** between different trotters is possible.
- Using **hls::stream** to read the **Jcoup** in sequential order and cache it.
- Moving the cache between trotters.



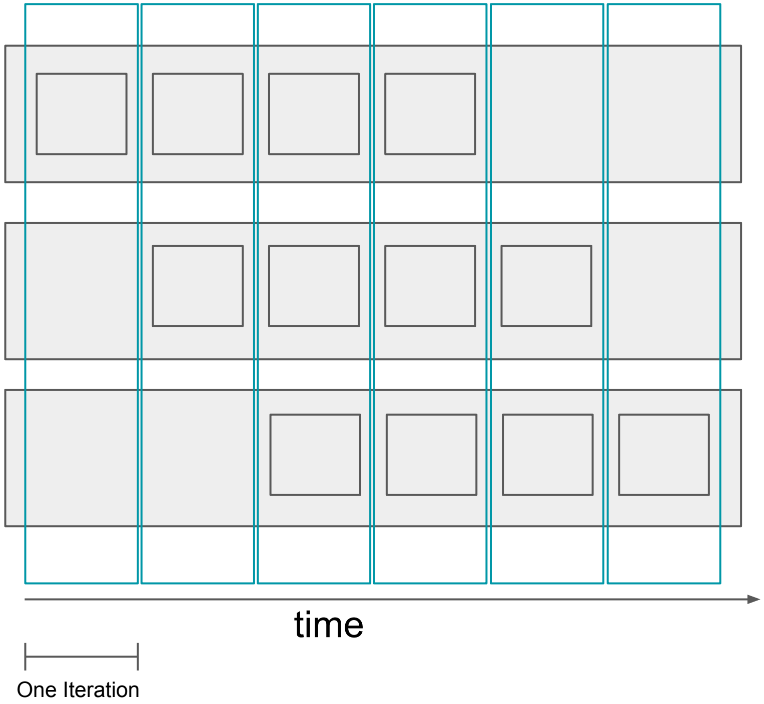
Implementation - Inter-Trotter Parallelism

- **All the units can perform in parallel, but hard to express in sequential code.**
- Explicitly control the steps and make the inner-most loop of trotters to unroll.

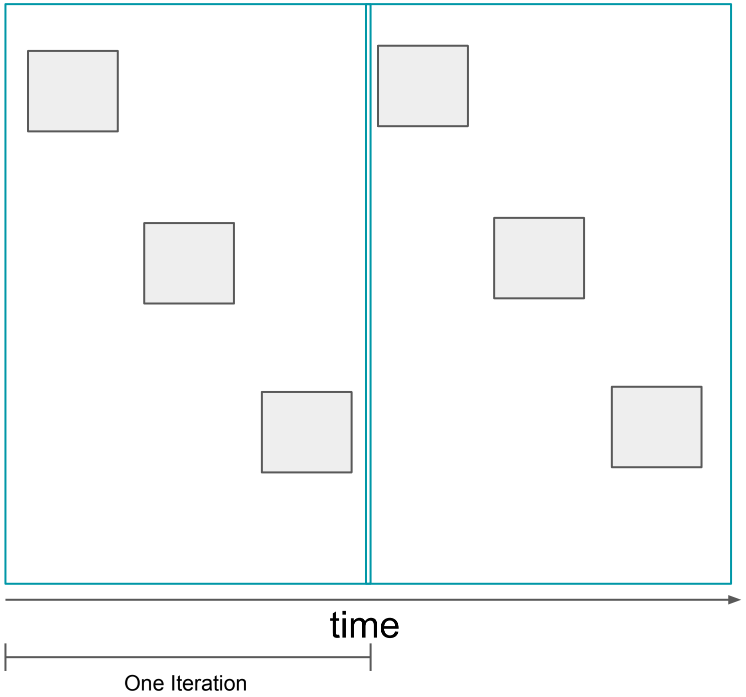
```
For i in 1 ~ (#S + #T - 1) :  // Total number of stages
    UpdateInputStates(inpuStates[1~t]);
    For j in 1 ~ (#S) :       // Each stage needs #S steps
        For t in 1 ~ (#T) :  // Unroll this loop
            if ( i >= t ) :  // Initial latency
                TrotterUnit(t, inpuStates[t], ... );
```

Implementation - Inter-Trotter Parallelism (Cont.)

- What we expect (Overlap)



- What we get (Sequential)

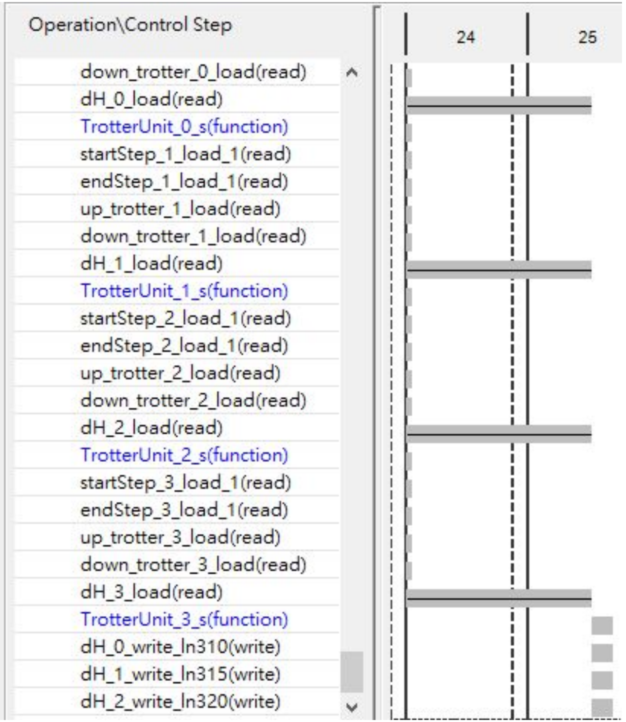


Implementation - Inter-Trotter Parallelism (Cont.)

- Use pragma to eliminate most of inter-trotter dependency.
  - #pragma HLS DEPDENCE
  - #pragma HLS ARRAY_PARTITION
- Use explicit unrolling by meta-programming.
  - #pragma HLS UNROLL doesn’t work.
- Make sure all the parameters to the trotter units are independant.
  - How to pass **Spin(m-1, i)** and **Spin(m+1, i) ?**
  - **Indpendent Input State.**

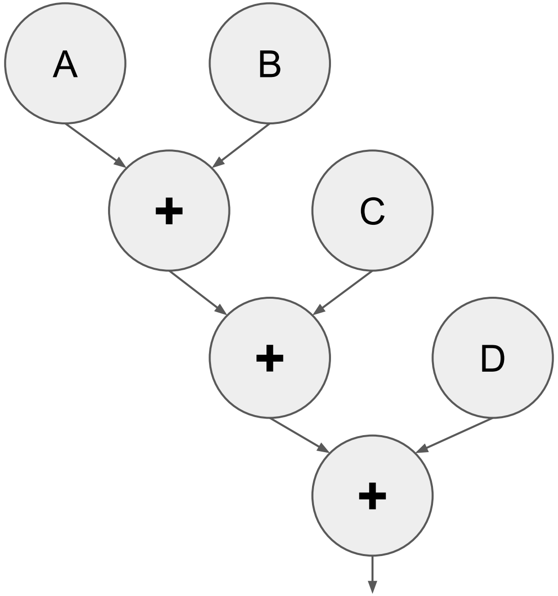
Implementation - Inter-Trotter Parallelism (Cont.)

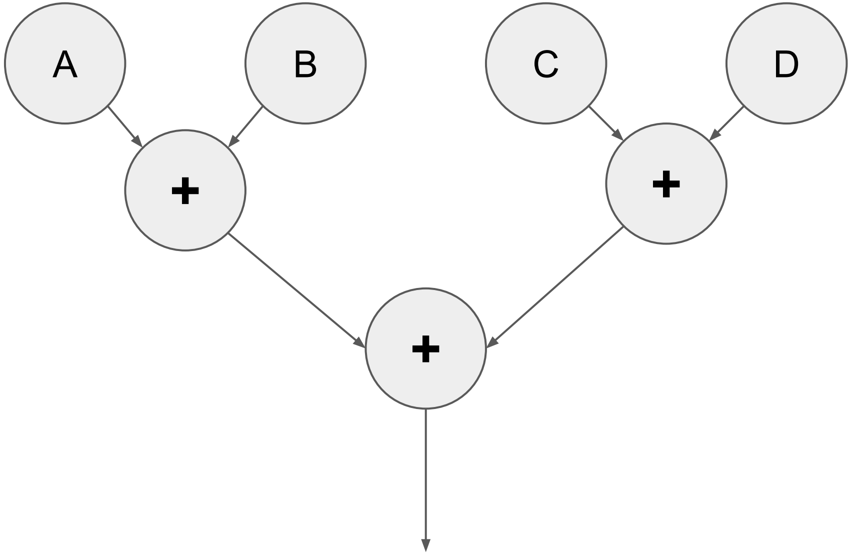
```
fp_t hNext_0 = (iPre[0] != nSpin - 1) ? h[iPre[0] + 1] : 0.0f;
TrotterUnit<0>(nTrot, nSpin, ctlStep, iPre[0], j, startStep[0],
               endStep[0], trotters[0], up_trotter[0], down_trotter[0],
               dH[0], hNext_0, Beta, dHTunnel, logRandNumber[0],
               JcoupLocal[0]);

fp_t hNext_1 = (iPre[1] != nSpin - 1) ? h[iPre[1] + 1] : 0.0f;
TrotterUnit<1>(nTrot, nSpin, ctlStep, iPre[1], j, startStep[1],
               endStep[1], trotters[1], up_trotter[1], down_trotter[1],
               dH[1], hNext_1, Beta, dHTunnel, logRandNumber[1],
               JcoupLocal[1]);

fp_t hNext_2 = (iPre[2] != nSpin - 1) ? h[iPre[2] + 1] : 0.0f;
TrotterUnit<2>(nTrot, nSpin, ctlStep, iPre[2], j, startStep[2],
               endStep[2], trotters[2], up_trotter[2], down_trotter[2],
               dH[2], hNext_2, Beta, dHTunnel, logRandNumber[2],
               JcoupLocal[2]);
```



Implementation - Intra-Trotter Parallelism

- Gathering multiple steps in one stage.
- Reduction with binary-tree form
- Consume resources a lot.





Implementation - PRNG

- After on-board testing, we find out that the transmission of random numbers from the host to the device dominates the execution time.
- To reduce the overhead, we implement a simple PRNG in HLS.
    - Reference : UNIFORM - A Uniform Random Generator
- The natural log opertion harms the performance and the resource usage a lot but it is still worth to implemnt a PRNG inside.
    - Especially the DSP48E usage.

Outline

- Introduction
- Analysis
- Implementation
- **Evaluation**
- Guideline of mapping real world problem - take sudoku as example
- Conclusion

Comparison - Interface and Memory Usage

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|      -|       -|      -|    -|
|FIFO             |        -|      -|       -|      -|    -|
|Instance         |     2070|      7|    1896|   3387|    -|
|Memory           |        -|      -|       -|      -|    -|
|Multiplexer      |        -|      -|       -|     15|    -|
|Register         |        -|      -|     131|      -|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |     2070|      7|    2027|   3402|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |      739|      3|       1|      6|    0|
+-----------------+---------+-------+--------+-------+-----+
```

Using s_axilite for J coupling
(Basic)

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|     27|       0|   1983|    -|
|FIFO             |        -|      -|       -|      -|    -|
|Instance         |       40|     31|    5815|  10731|    -|
|Memory           |        8|      -|       0|      0|    0|
|Multiplexer      |        -|      -|       -|    959|    -|
|Register         |        -|      -|    1942|      -|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |       48|     58|    7757|  13673|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |       17|     26|       7|     25|    0|
+-----------------+---------+-------+--------+-------+-----+
```

Using axis for J coupling
(Optimization 1)

Comparison - Latency - Inter-Trotter Parallelism

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
40	58884123	0.400 us	0.589 sec	40	58884123	none

Basic

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
20	40118282	0.200 us	0.401 sec	20	40118282	none

Optimization 1

Comparison - Latency - Intra-Trotter Parallelism

- 16 Operations Once
- Get 5.73x Boost Up

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
20	40118282	0.200 us	0.401 sec	20	40118282	none

Optimization 1

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
39	6994314	0.390 us	69.943 ms	39	6994314	none

Optimization 2

5.73x

Comparison - Latency - Trotter Size

- **O(#T × #S × #S) vs O((#S + #T) × #S)**
- As the #T grows, the time needed for naive implementation will grows a lot.

#T = 8

```
+ Latency:
    * Summary:
    +---------+----------+----------+----------+---------+----------+---------+
    |  Latency (cycles)  |  Latency (absolute) |      Interval      | Pipeline|
    |   min   |    max   |    min   |    max   |   min   |    max   |   Type  |
    +---------+----------+----------+----------+---------+----------+---------+
    |       40|  58884123| 0.400 us | 0.589 sec|       40|  58884123|   none  |
    +---------+----------+----------+----------+---------+----------+---------+
```

#T = 16

```
+ Latency:
    * Summary:
    +---------+-----------+----------+----------+---------+-----------+---------+
    |   Latency (cycles)  |  Latency (absolute) |       Interval      | Pipeline|
    |   min   |    max    |    min   |    max   |   min   |    max    |   Type  |
    +---------+-----------+----------+----------+---------+-----------+---------+
    |       40|  117768235| 0.400 us | 1.178 sec|       40|  117768235|   none  |
    +---------+-----------+----------+----------+---------+-----------+---------+
```

Naive

2x

Comparison - Latency - Trotter Size (Cont.)

- The main overhead comes from the memory movement of J coupling between trotter units. (There exists some problem for circular buffer)

#T = 8

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
39	6994314	0.390 us	69.943 ms	39	6994314	none

#T = 16

+ Latency:
  * Summary:

Latency (cycles) min	Latency (cycles) max	Latency (absolute) min	Latency (absolute) max	Interval min	Interval max	Pipeline Type
47	8112522	0.470 us	81.125 ms	47	8112522	none

Optimization 2

1.16x

Comparison - Area - Intra-Trotter Parallelism

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	27	0	1983	-
FIFO	-	-	-	-	-
Instance	40	31	5815	10731	-
Memory	8	-	0	0	0
Multiplexer	-	-	-	959	-
Register	-	-	1942	-	-
Total	48	58	7757	13673	0
Available	280	220	106400	53200	0
Utilization (%)	17	26	7	25	0

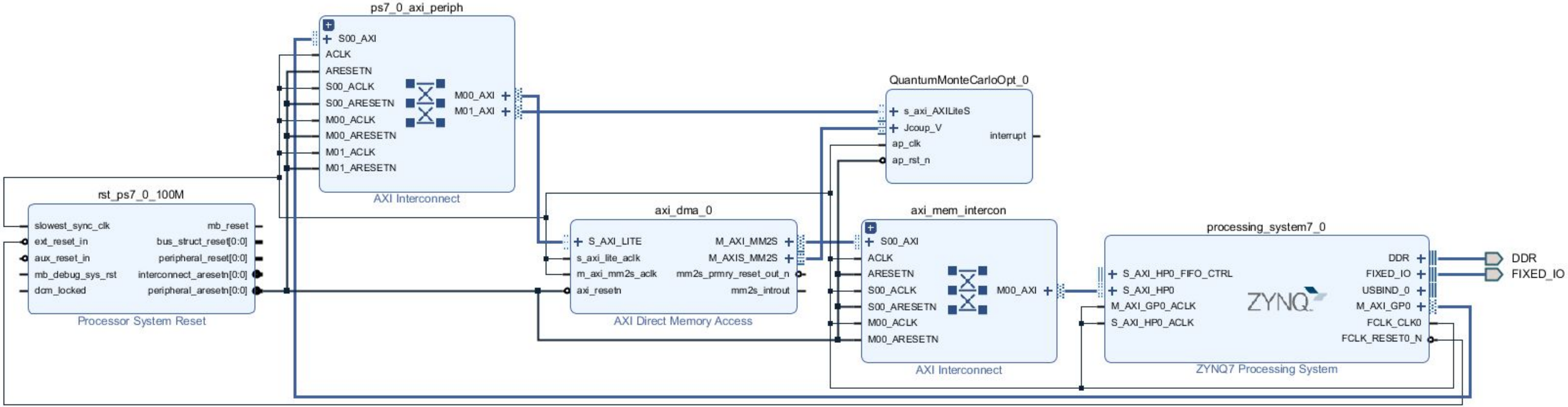
Optimization 1

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	27	0	2538	-
FIFO	-	-	-	-	-
Instance	40	59	14831	31315	-
Memory	16	-	0	0	0
Multiplexer	-	-	-	2901	-
Register	-	-	5953	-	-
Total	56	86	20784	36754	0
Available	280	220	106400	53200	0
Utilization (%)	20	39	19	69	0

Optimization 2

On-Board - Block Diagram



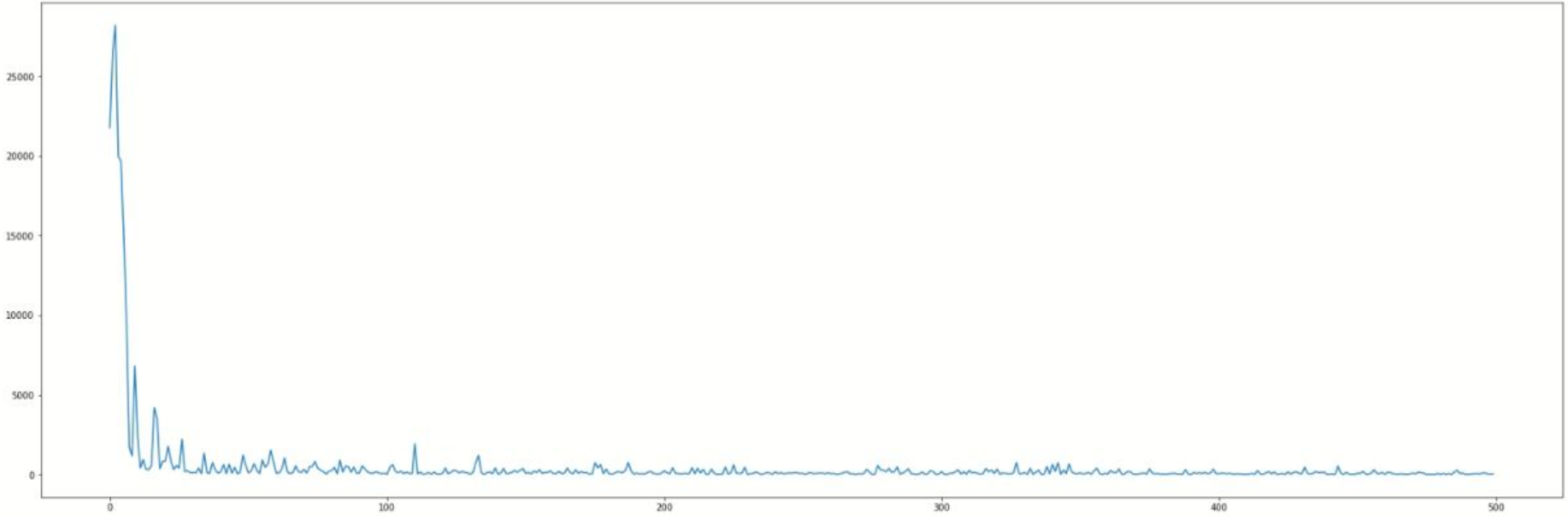
On-Board - Validation

- Number Partition Problem
- The difference between the sum of the two sets is the minimal.

$$H = -\left(\sum_{i=1}^{N} n_i \sigma_i\right)^2 = -\sum_{i,j=1}^{N} n_i n_j \sigma_i \sigma_j$$

On-Board - Validation

```
In [66]: plt.figure(figsize=(30,10))
         plt.plot(sumEnergy)

Out[66]: [<matplotlib.lines.Line2D at 0xaf2e0af0>]
```




```
In [67]: best

Out[67]: (345, 1, 511.4990234375, 511.5009765625, 3.814697265625e-06)
```

On-Board - Random Number from Host

- Most of time is spending on the input of the random numbers
    - Including generation and casting (float to bytes)
- Only **7%** is the execution time of the kernel.

```
Random Number Generate time: 4.06361222267 s
Passing Random Number time : 322.989094257 s
Kernel execution time      : 28.2271444798 s
Getting Trotters time      : 42.1241939068 s
```

```python
# Write Random Nubmers
k = 0
for addr in range(0x4000, 0xC000, 0x04):
    ipSQA.write(addr, float2bytes(rn[k]))
    k += 1
```

```
print("Kernel execution time: " + str(np.sum(timeList)) + " s")

100%|██████████| 500/500 [06:38<00:00,  1.26it/s]

Kernel execution time: 28.2303230762 s
```

On-Board - Random Number from PRNG

- The execution time of the kernel grow, but the total execution time decreased.
- The tested version contains more overhead than the previous one
  - Boundary Check (a.k.a. Branch Divergence)
  - Pseudo Random Number Generator
  - Natural Log
- Same problem size, **0.68 speedup** of kernel but **4.68 speedup** of the overall process.
- Harm the kernel but good for the whole system.

```
print("Kernel execution time: " + str(np.sum(timeList)) + " s")
```

```
100%|██████████| 500/500 [01:25<00:00,  5.83it/s]
Kernel execution time: 41.455773592 s
```

Outline

- Introduction
- Analysis
- Implementation
- Evaluation
- **Guideline of mapping real world problem - take sudoku as example**
- Conclusion

Guideline of mapping real world problem

Parameters: beta, gamma

Real world problem → Combinational problem → (Jcoup, h; trotters) →

```
for step in monte carlo {
    Kernel
}
```

→ answer

Easy to implement and verify

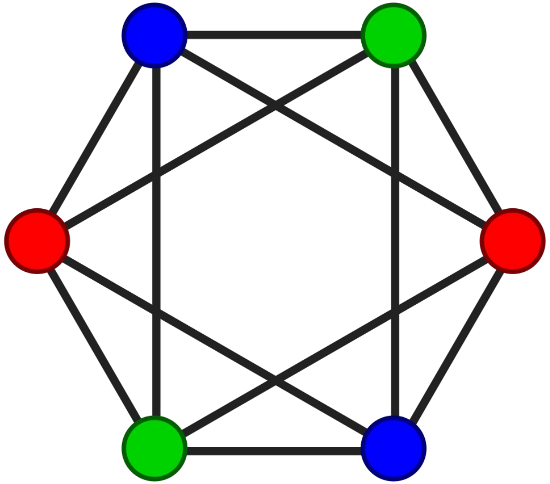
Sudoku → Graph Coloring Problem → SQA

Job scheduling

Register allocation

Graph coloring problem

- Given an undirected graph G = (V, E), and a set of n colors, is it possible to color each vertex in the graph with a specific color, such that no edge connects two vertices of the same color?
- NP-complete problem



Mathematical formulation of graph coloring problem

Xv,i = 1: **vertex v is color i**
Xv,i = 0: **vertex v is not color i**

$$H = A\sum_{v}\left(1-\sum_{i=1}^{n}x_{v,i}\right)^{2} + A\sum_{(uv)\in E}\sum_{i=1}^{n}x_{u,i}x_{v,i}.$$

Ref: Ising formulations of many NP problems - Andrew Lucas - 2014

Mathematical formulation of graph coloring problem

$X_{v,i} = 1$: **vertex v is color i**
$X_{v,i} = 0$: **vertex v is not color i**

for all color

$$H = A \underbrace{\sum_{v} \left(1 - \sum_{i=1}^{n} x_{v,i}\right)^2}_{\text{Each vertex has only one color}} + A \sum_{(uv)\in E} \sum_{i=1}^{n} x_{u,i} x_{v,i}.$$

Ref: Ising formulations of many NP problems - Andrew Lucas - 2014

Mathematical formulation of graph coloring problem

$X_{v,i} = 1$: **vertex v is color i**
$X_{v,i} = 0$: **vertex v is not color i**

for all color

$$H = A \underbrace{\sum_v \left(1 - \sum_{i=1}^{n} x_{v,i}\right)^2}_{\text{Each vertex has only one color}} + A \underbrace{\sum_{(uv)\in E} \sum_{i=1}^{n} x_{u,i} x_{v,i}}_{\text{Energy penalty: each time an edge connects two vertices of the same color}} .$$

Minimize H to 0

Ref: Ising formulations of many NP problems - Andrew Lucas - 2014

Mathematical formulation of graph coloring problem

$$H = A\sum_{v}\left(1-\sum_{i=1}^{n} x_{v,i}\right)^2 + A\sum_{(uv)\in E}\sum_{i=1}^{n} x_{u,i}x_{v,i}.$$

Transform to Ising Hamiltonian

Problem definition

$$H(s_1,\ldots,s_n) = -\sum_{i=1}^{n} \boxed{h_i} s_i - \sum_{i,j=1}^{n} \boxed{J_{ij}} s_i s_j$$

Find best combination of s (trotter) to minimize H

Trotter slice

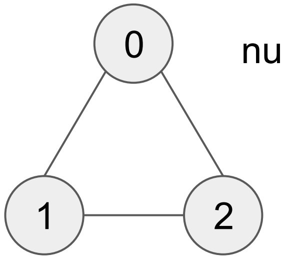
color

	0	1	2
0	?	?	?
1	?	?	?
2	?	?	?
3	?	?	?

vertex

Example

$$H = \boxed{A\sum_{v}\left(1-\sum_{i=1}^{n}x_{v,i}\right)^{2}} + \boxed{A\sum_{(uv)\in E}\sum_{i=1}^{n}x_{u,i}x_{v,i}.}$$



num_color = 3

Trotter slice

color →

vertex ↓ \ color	0	1	2
0	S0	S1	S2
1	S3	S4	S5
2	S6	S7	S8

Jcoup (9x9) (relation between spin)

	0	1	2	3	4	5	6	7	8
0	0	-0.25	-0.25	-0.125	0	0	-0.125	-0	-0
1	-0.25	0	-0.25	0	-0.125	0	0	-0.125	0

….

H (1x9) (constant energy for each spin)

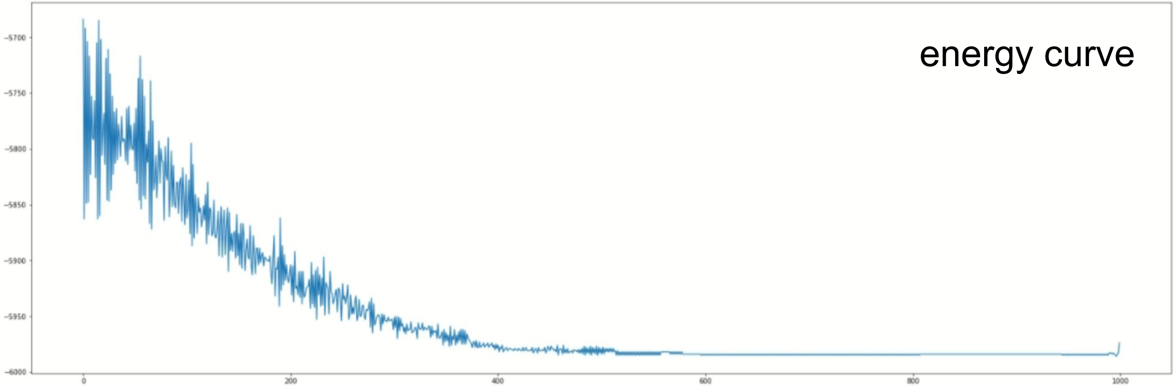
0	-1	-1	-1	-1	-1	-1	-1	-1	-1

From sudoku to graph coloring

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- **81 vertices**
- **9 colors**
- Two distinct vertices will be adjacent if and only if the corresponding cells in the grid are either in
  - **same row**
  - **same column**
  - **same sub-grid**
- Parametes of SQA
  - 1 trotter slice = 81 (vertices) x 9 (color)
    - 729 spins
  - Jcoup (729 x 729)
  - h (1 x 729)
  - beta
  - gamma

Result



We solve the 9x9 sudoku problem from 5 empties to 25 empites

1	6(V)	8	3	5	2	7	9	4
7	4	9	1	6	8	3	2	5
3	5	2	7	4(V)	9(V)	1	8	6
9	7	6	8	1	5	2	4	3
8	1	5	2	3	4	9	6	7
2	3(V)	4	9(V)	7	6	8	5	1
6	9	1	5	8	3	4	7	2
5	8	3	4	2	7	6	1	9
4	2	7	6	9	1	5	3	8

```
100%|██████████| 1000/1000 [02:15<00:00,  7.32it/s]
Kernel execution time: 52.1875731945 s
```

5 empites example using 1000 iteration
# of trotter: 2
# of spins: 729
maxBeta = 1 / 0.15
minBeta = 1 / 0.4
Gamma = 0.0001

Outline

- Introduction
- Analysis
- Implementation
- Evaluation
- Guideline of mapping real world problem - take sudoku as example
- **Conclusion**

Conclusion

- We reproduce the paper [1] in a more general form with HLS.
  - Design a suitable interface in HLS
  - Exploit Parallelism of Inter-Trotters
  - Exploit Parallelism of Intra-Trotters
  - About 6x speedup than naive implementation

- According to the on-board testing, we modify our deisgn, add PRNG into our own HLS implementation to reduce the I/O latency and overall processing time of the whole system.

- Provide a simple guideline about mapping the real world problem to the quantum annealer, and take sudoku as a example.

Furture Work

- Improve the I/O latency
  - Reading trotters from the device after every iteartion is time-consuming.
- Integrate mutiple iteration.
  - Eliminate the restart gap.
- Develop an efficient way to move the cache of J couping.
  - Be careful about how the compiler handle the data dependencies.

Reference

- Highly-Parallel FPGA Accelerator for Simulated Quantum Annealing
  - Hasitha Muthumala Waidyasooriya, and Masanori Hariyama, both are Members in IEEE(Dec. 2019)
- OpenCL-based design of an FPGA accelerator for quantum annealing simulation
  - Hasitha Muthumala Waidyasooriya, Masanori Hariyama, Masamichi J. Miyama, Masayuki Ohzeki (Feb. 2019)
- An Accelerator Architecture for Combinatorial Optimization Problems
  - Sanroku Tsukamoto, Motomu Takatsu, Satoshi Matsubara, Hirotaka Tamura (Fujistu)
  - r08943099/MSOCFall2020
- Quantum annealing of the Traveling Salesman Problem
  - Roman Martonak, Giuseppe E. Santoro, and Erio Tosatti (Feb. 2008)

Reference

- UNIFORM - A Uniform Random Generator
- Zynq-7000 SoC Family Product Seletion Guide

Github

- https://github.com/allen880117/Simulated-Quantum-Annealing

Q & A