TOHOKU UNIVERSITY

Graduate School of Information Sciences, Tohoku University

Cyberscience Center

Quantum Compiler

Automatic Vectorization Assisted by Quantum Annealer

(URL) https://www.sc.cc.tohoku.ac.jp
(E-mail) yuta.sasaki.p4@dc.tohoku.ac.jp

Graduate School of Information Sciences & Cyberscience Center, Tohoku University

Yuta Sasaki, Michael Ryan Zielewski, Mulya Agung, Ryusuke Egawa, Hiroyuki Takizawa

Background

- Most modern processors support **vector processing** or similar techniques to improve performance.
- Compilers perform automatic **loop vectorization** after checking if the loop is safely vectorizable.
  - Programmers sometimes need to manually optimize a loop to help compiler.
  - When there is a cyclic data dependency in a loop, **variable renaming** is required to break it.
  - There are several kinds of variable renaming techniques for optimization, and the performance after vectorization depends on the techniques.

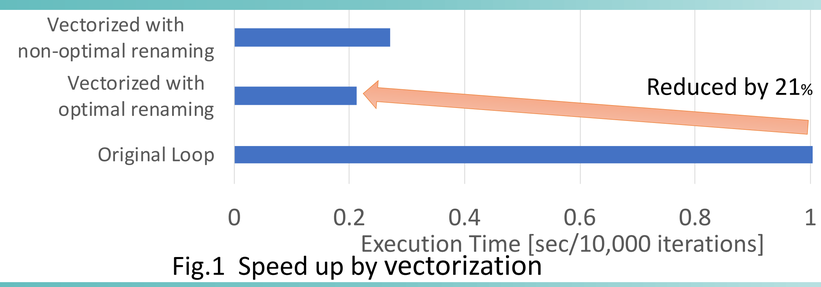
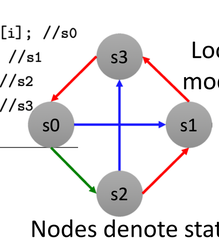


Fig.1 Speed up by vectorization

Problem – Combinatorial Optimization for Vectorization –

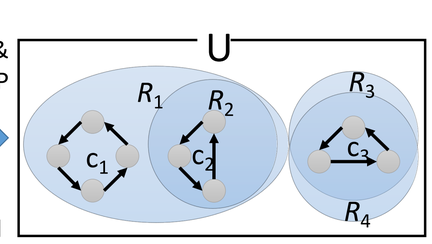
- A loop containing many cyclic data dependencies (c_j) needs several renaming techniques (R_i).
- Finding an optimal combination of R_is is a Set Cover Problem (SCP) [1].
  - To break all cyclic data dependencies (1) and to minimize runtime overhead (2).
  - It is NP-complete. Computational time increases rapidly with the number of cyclic data dependencies.

```
1 for(i=0; i<3; i++){
2         a[i]=c[i-1]+b[i]; //s0
3         b[i]=a[i-1];  //s1
4         a[i]=c[i];    //s2
5         c[i]=b[i];    //s3
6         }
```



Loop analysis & modeling as SCP

Nodes denote statements and edges denote dependencies



$c_j \in R_i$: R_i can break c_j

If an optimal combination instead of suboptimal ones, can be obtained within a practical time, compilers would perform better vectorization leading to higher performance.

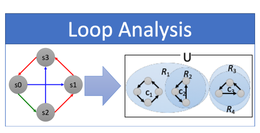
Proposal – Quantum Annealing for Compiler Support –

Find an optimal combination of R_is by using quantum annealing (QA), an emerging technology to solve combinatorial optimization problems.

- Model the SCP problem by PolySIMD [2], and solve the problem using QA.
  - A C source code as input, and an optimal combination of R_is as output.
  - Need to convert SCP and its constraint to a QUBO (Quadratic Unconstrained Binary Optimization) problem, which D-Wave can process.

Input:
C source code

```
1 for(i=0; i<3; i++){
2         a[i]=c[i-1]+b[i]; //s0
3         b[i]=a[i-1];  //s1
4         a[i]=c[i];    //s2
5         c[i]=b[i];    //s3
6         }
```



Formulation

$$\begin{cases} \forall c_j, \left(\sum_{\forall i \text{ such that } T_i \text{ can break } c_j} x_i\right) \geq 1 & (1) \\ \text{Minimize } H_A = \sum_{i=1}^{N}(w_i x_i) & (2) \end{cases}$$

$x_i = 1$ *when using* R_i, w_i : *overhead of* R_i

Generate QUBO matrix

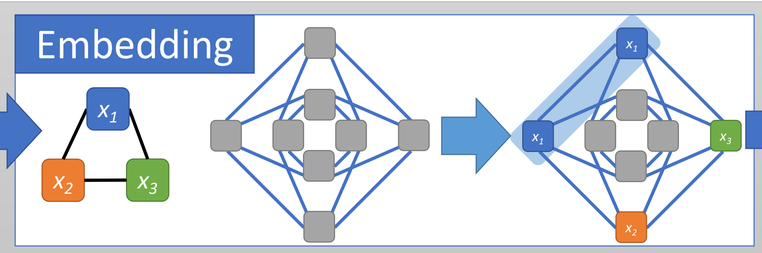
$$H = H_A + \sum_{j=1}^{n}\left(1 + \sum_{k=1}^{D-1} z_{j,k} - \sum_{\forall i \text{ such that } t_i \text{ can break } c_j} x_i\right)^2$$

$$= \boldsymbol{Y}^T Q \boldsymbol{Y} + 1 \ (\mathrm{Q}: \text{QUBO matrix})$$

$$(Y = [x_1, x_2, \ldots, x_N, z_{1,k}, z_{2,k}, \ldots, z_{M,k}]^T)$$

Loop analysis and formulation as weighted SCP by PolySIMD [2].
・Adopt two types of R_i having different runtime overheads.
Convert problem to QUBO using a penalty function.
・QUBO must not contain constraint expression (1).

Embedding



Qubits on D-Wave have a chimera structure (middle).
・To embed QUBO having 3-loop(left), copy same variable to several qubits, creating "chain"(right).
Finding embedding is harder for a larger QUBO matrix.

Solution

D-Wave 2000Q

https://www.dwavesys.com/

Output: the optimal combination of R_is

Evaluation

Setup and Tools

Processor	Intel Core i7-9700K
Memory	32GB
MIP solver (for comparison)	CBC[4]

QA solver	DW_2000Q_6
QUBO generation tool	PyQUBO[3]
Embedding tool	minorminer
Number of reads qubit	num_reads = 100

Input a loop containing eight dependency cycles (Fig. 2).
・The optimal solution is verified by matching the result with that of CBC, an exact solver.
  - Break all cycles (Fig. 3).
  - Execution time is was reduced by 21 % (Fig .1).

Experimental results demonstrate the feasibility of QA to assist automatic loop vectorization by compilers

```
1  for (i = 1; i < 16*1024; i++) {
2      a[i] = d[i]+2; //s0
3      b[i] = a[i]+3; //s1
4      e[i] = b[i]+4; //s2
5      b[i] = f[i]+5; //s3
6      c[i] = b[i-1]+6; //s4
7      a[i] = a[i+1]+7; //s5
8      a[i+1] = c[i]+8; //s6
9      h[i] = a[i+1]+9; //s7
10     }
```

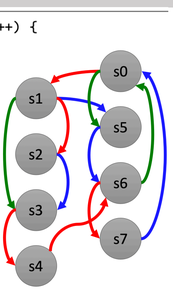


Fig.2 Original Loop

```
1  for (i = 1; i < 16*1024; i++) {
2      temp[i] = f[i] + 5;//s31
3      c[i] = (i >= 1)? temp[i-1]+6 : b[0]+6; //s4
4      k = a[i+1]; //s51
5      a[i + 1] = c[i] + 8;//s6
6      h[i] = a[i+1] + 9; //s7
7      a[i] = d[i] + 2; //s0
8      b[i] = a[i] + 3; //s1
9      e[i] = b[i] + 4; //s2
10     b[i] = temp[i]; //s3
11     a[i] = k + 7; //s5
12 }
```

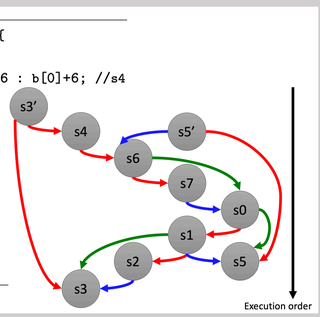
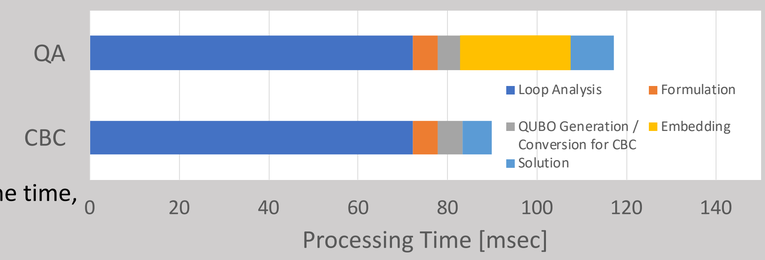


Fig.3 Renamed Loop

Processing time compared with CBC

- QA requires a longer time for this instance.
- Embedding is time-consuming.

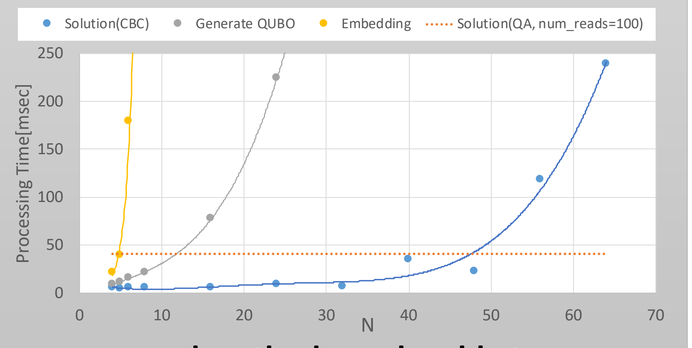
* Only QA needs to generate QUBO and embedding.
* QA solution time is measured from D-Wave machine time, not including communication time.



Processing time related to problem size

Solve SCP instances with R_i and c_j for i=1,...N.

- Solution time of CBC increases sharply in N > 50.
- QA preprocessing time (Generate QUBO and Embedding) also increases with N.
- Solution time of QA is not affected by N and constant for the number of reads.



If QA can process larger problems in the future, our proposal method may be able to solve SCP faster than the existing methods using only classical machines.

- **We need to reduce the overhead of preprocessing as problem size increases.**

Conclusion and Future Work

- Assuming offload to QA, we can implement compiler optimization without considering the number of cyclic dependencies.
- QA is promising not only to replace a combinatorial optimization part of HPC application execution but also to help automatic compiler optimization and hereby improve the performance even on conventional HPC applications.
- Future Work : We will discuss how to reduce the embedding time, which accounts for a large proportion of the total execution.
  Precompute and store embeddings to reduce embedding time to hashing and lookup.
  - Small QUBOs with 64 variables or less can be embedded as complete graphs, but this method is inefficient and not applicable to large problems.
  - Need to introduce parameters other than the number of variables.

Acknowledgement

The authors are grateful to Prasanth Chatarasi and Vivek Sarkar for willing to contribute the source code for PolySIMD, their tool to extend vectorization technologies. Their study motivates our proposed method and is very helpful in its implementation.
This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications," Grant-in-Aid for Scientific Research(B) #16H02822 and #17H01706.

References

[1]Pierre-Yves Calland, Alain Darte, Yves Robert, and Frédéric Vivien. On the removal of anti-and output-dependences. International Journal of Parallel Programming, 26(3):285–312, 1998.
[2]Prasanth Chatarasi, Jun Shirako, Albert Cohen, and Vivek Sarkar. A unified approach to variable renaming for enhanced vectorization. In International Workshop on Languages and Compilers for Parallel Computing, pages 1–20. Springer, 2018.
[3] Kotaro Tanahashi. Pyqubo. https://github.com/recruit-communications/pyqubo.
[4] Saltzman, Matthew J. "COIN-OR: an open-source library for optimization." Programming languages and systems in computational economics and finance. Springer, Boston, MA, 2002. 3-32.