Deep reinforcement learning in loop fusion problem

Group 15: Siqi Shao, Yixin Shi, Ying Yang
04/03/2023

https://doi.org/10.1016/j.neucom.2022.01.032

Introduction

Problem Statement: Loop Fusion is a loop transformation in which several loops are combined to form a loop.

Goal:

- Reduce overhead
- Increase parallelism
- Enhance data reuse

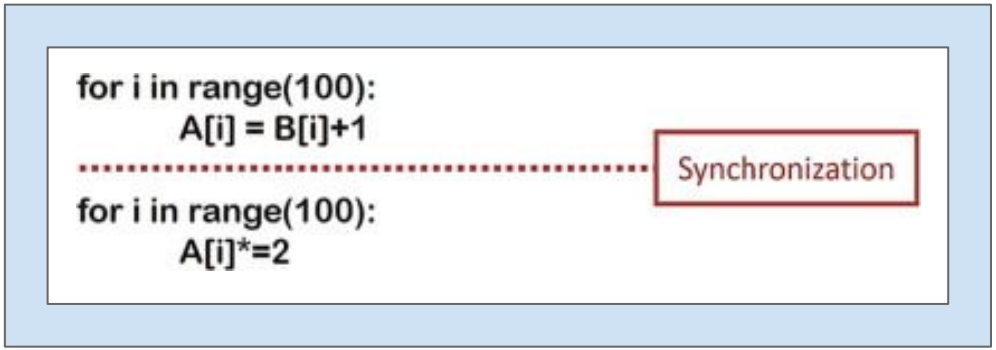


Fig1. A synchronization requisite between loops in a parallel system

Introduction

Problem Statement: Loop Fusion is a loop transformation in which several loops are combined to form a loop.

Goal:

- Reduce overhead
- Increase parallelism
- Enhance data reuse

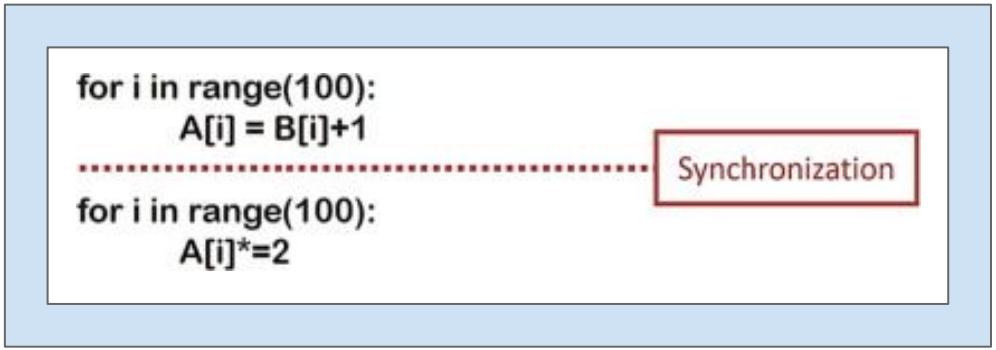


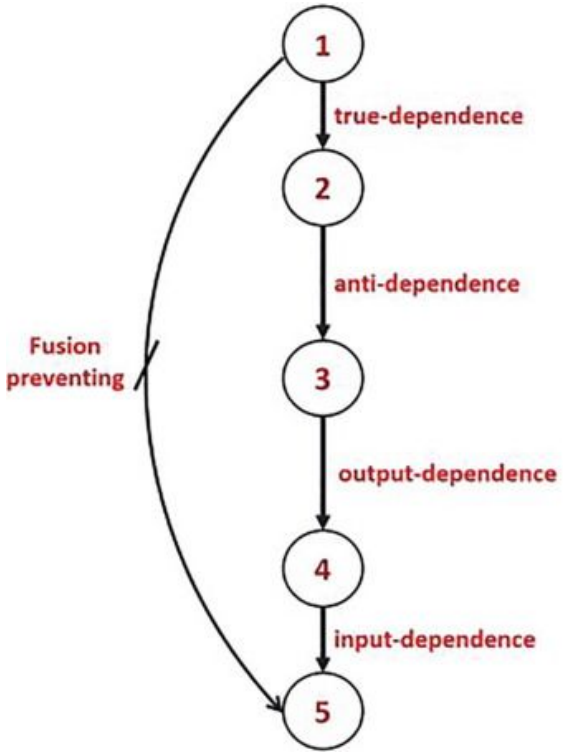
Fig1. A synchronization requisite between loops in a parallel system

Important technique for optimizing the performance of scientific or computational algorithms that involve repetitive operations.

Loop Fusion is a NP-hard problem

Factors to consider in loop fusion.

- Data dependencies
- Data reuse
- Loops' type
- Computer system's register size



Related code

```
for i in range(100):
  A[i] = i

for i in range(100):
  B[i]=A[i]+2
  F[i]=C[i]

for i in range(100):
  C[i]=4
  H[i]=i*3

for i in range(100):
  H[i]=G[i]

for i in range(100):
  K[i]=A[i-1]+G[i]
```

Loops
Dependence edges
Fusion preventing edges

Loop 1 and Loop 5: Loop carrier dependence based on array A.

Related Work

Year	Paper's goal	Number of types	Graph's node	Loop order	Modeling method	Improvement factors		
						parallelism	data reuse	Register size
1984	Improving parallelism	2	Loop and instruction	Topological order of loop independent dependence graph	hierarchical dependence graph (HDG)	✓	✓	×
1993	Provide two distinct methods for improving parallelism and data	2	loop	Breath-first-search with the parallel type's priority	Directed acyclic graph	✓	✓	×
2004	Improving data reuse considering register size	2	loop	Topological order with the priority of removing the edges with maximum weight	Directed weighted acyclic graph	×	✓	✓
2020	Improving data reuse with maintaining parallelism	2	permutation	Topological order considering a coloring heuristic	Fusion conflict graph (FCG)	✓	✓	×
2020	Improving parallelism and data reuse considering register size	>2	loop	The sequence generated by evolutionary algorithm	Directed acyclic graph	✓	✓	✓

Hard to achieve all three improvement factors; Long time to solve the problem.

Deep Reinforcement Learning Loop Fusion

- Find a proper order for loop fusion
- Enhance parallelism
- Consider the system's register size as a boundary for fusion
- First use deep reinforcement learning to solve loop fusion in a short span of time.

Proposed Algorithm

- Goal: determine the best order of loops for merging
  - the runtime of the resulting loops is optimal
- Main framework

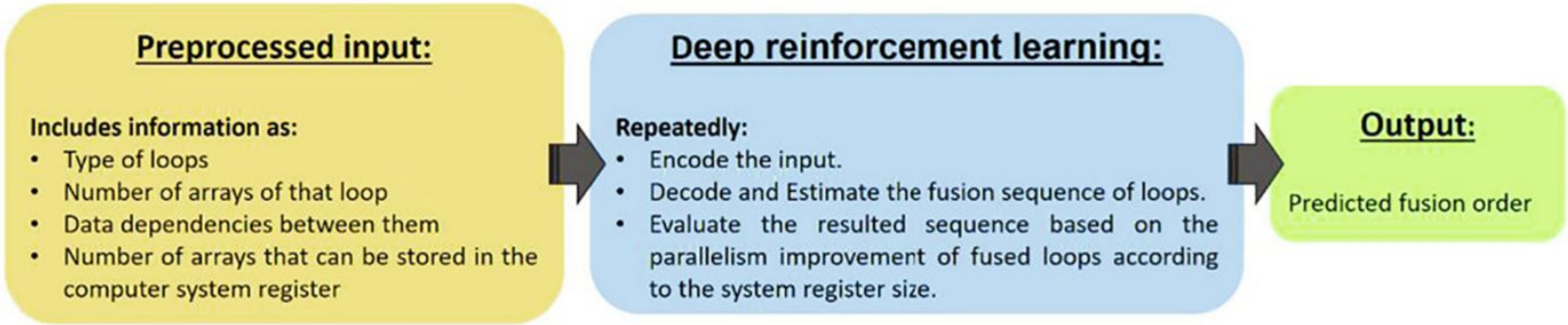


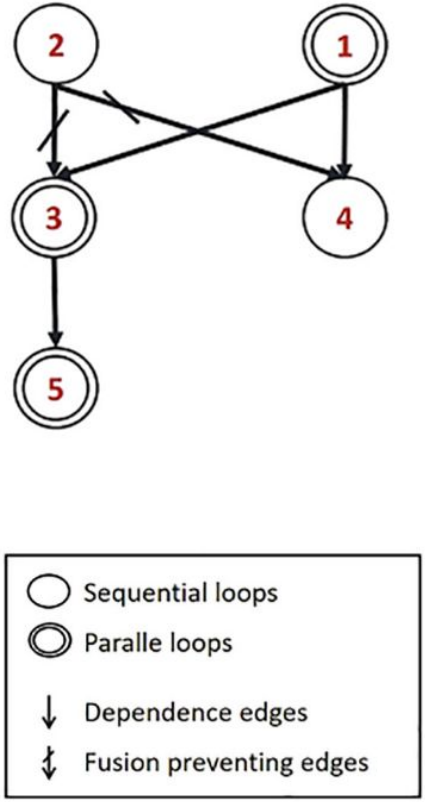
Fig. 3. The framework of the proposed model.

Input preprocessing

- **Draw a loop dependency graph**
  - Sequential loops vs Parallel loops
    - Whether it can execute multiple iterations simultaneously
  - Dependency edges
  - Fusion preventing edges

An Example Code

```
for(i=1; i<100;i++){
    a[i]=a[i]+1;
}

for(i=1; i<100;i++){
    b[i]=b[i-1]+c[i];
}

for(i=1; i<100;i++){
    d[i]=b[i-1]+a[i];
}

for(i=1; i<100;i++){
    e[i]=e[i-1]+b[i+1];
}

for(i=1; i<100;i++){
    g[i]=d[i]+f[i];
}
```

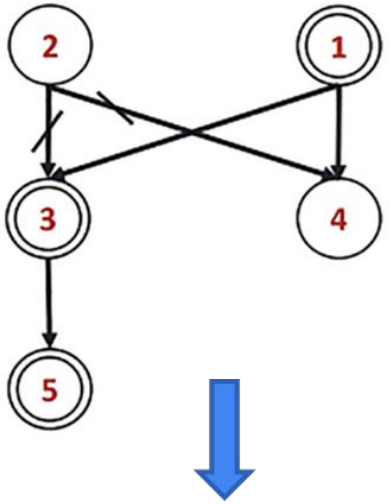


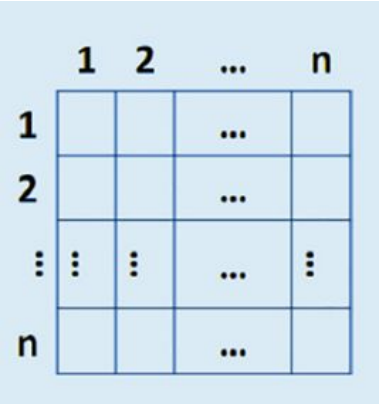
Input preprocessing

- **Create an adjacency matrix**
  - n × n matrix
    - (i, j) -> loop i to loop j
  - hold floating-point numbers
  - Include features
    - Type of loops
    - Data dependencies between loops
    - Number of arrays of loops
    - Number of arrays that can be stored in the computer system register

An Example Code

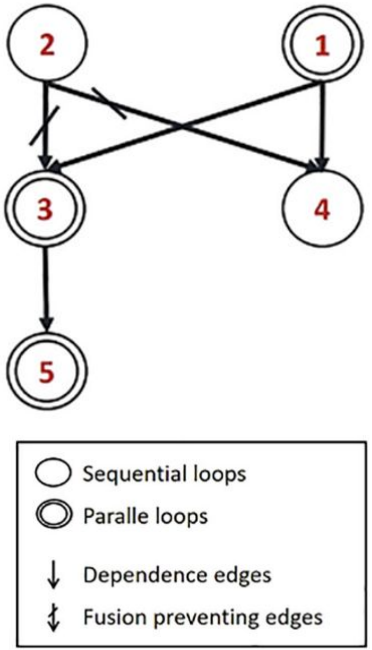
```
for(i=1; i<100;i++){
    a[i]=a[i]+1;
}

for(i=1; i<100;i++){
    b[i]=b[i-1]+c[i];
}

for(i=1; i<100;i++){
    d[i]=b[i-1]+a[i];
}

for(i=1; i<100;i++){
    e[i]=e[i-1]+b[i+1];
}

for(i=1; i<100;i++){
    g[i]=d[i]+f[i];
}
```





Input preprocessing

- **Create an adjacency matrix**
  - Main diagonal: information about the type of each loop
    - Integer part: the size of the system register

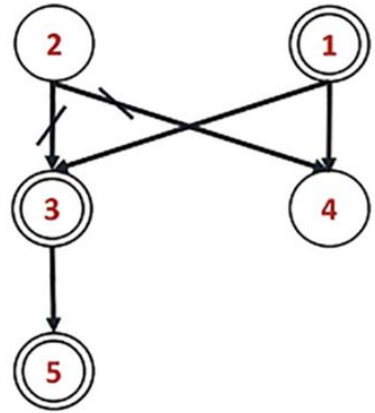


System register size

2.01	7.1	1.1	6.1	10.1
7.1	2.02	8.1	5.1	5.1
10.1	10.1	2.01	7.1	1.1
10.1	10.1	7.1	2.02	7.1
10.1	10.1	10.1	7.1	2.01

Input preprocessing

- **Create an adjacency matrix**
  - Main diagonal: information about the type of each loop
    - Fractional part: number to the loop's type

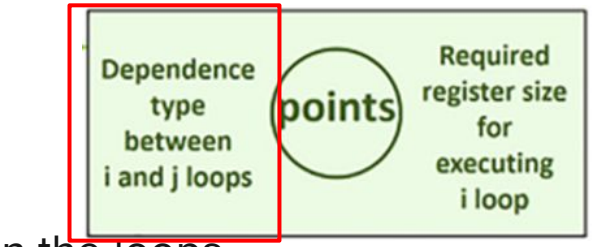


parallel (0.01) sequential (0.02)

2.01	7.1	1.1	6.1	10.1
7.1	2.02	8.1	5.1	5.1
10.1	10.1	2.01	7.1	1.1
10.1	10.1	7.1	2.02	7.1
10.1	10.1	10.1	7.1	2.01

Input preprocessing

- **Create an adjacency matrix**
  - The other matrix elements: dependencies between the loops
    - Integer part: type of dependence



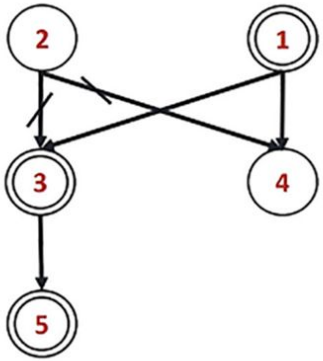


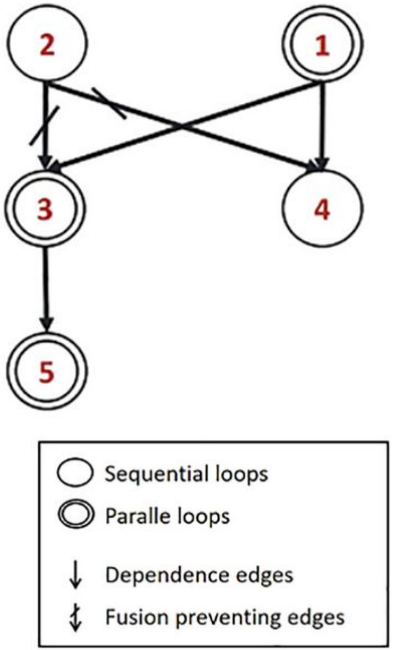
Table 2
The notion of the input matrix data values.

Dependence type	Assigned integer
Two dependent loops with the same type	1
Two not dependent loops with the same type	3
Two same type loops with a fusion preventing edge	5
Two dependent loops with different types	6
Two not dependent loops with different types	7
Two deferent types loop with a fusion preventing edge	8
Two loops that can not be topologically sequenced in fusion order	10

$$\begin{pmatrix} 2.01 & 7.1 & 1.1 & 6.1 & 10.1 \\ 7.1 & 2.02 & 8.1 & 5.1 & 5.1 \\ 10.1 & 10.1 & 2.01 & 7.1 & 1.1 \\ 10.1 & 10.1 & 7.1 & 2.02 & 7.1 \\ 10.1 & 10.1 & 10.1 & 7.1 & 2.01 \end{pmatrix}$$

Input preprocessing

- **Create an adjacency matrix**
  - The other matrix elements: dependencies between the loops
    - Fractional part : the number of register units required to execute the loop i



Dependence type between i and j loops — points — Required register size for executing i loop

Register size for each loop

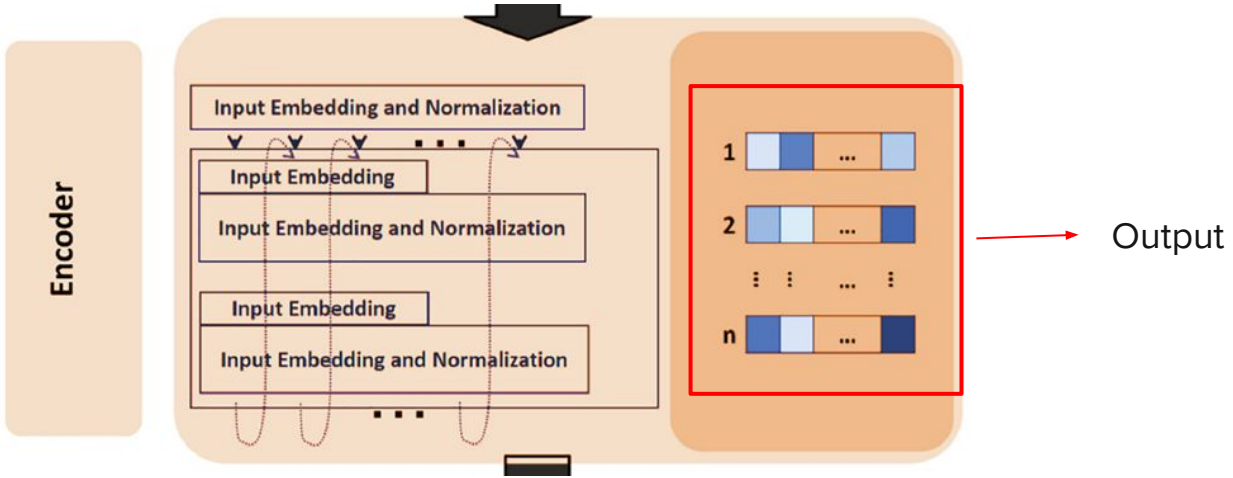
2.01	7.1	1.1	6.1	10.1
7.1	2.02	8.1	5.1	5.1
10.1	10.1	2.01	7.1	1.1
10.1	10.1	7.1	2.02	7.1
10.1	10.1	10.1	7.1	2.01

An Example Code

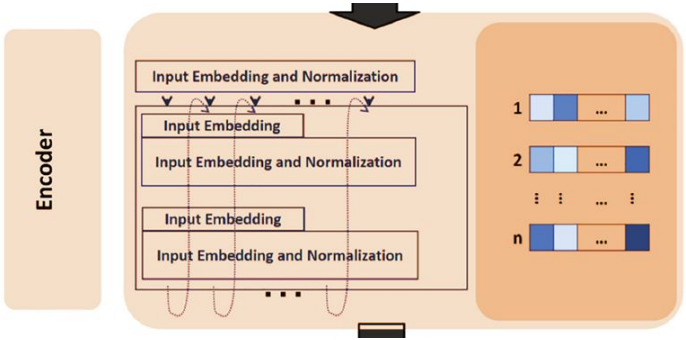
```
for(i=1; i<100;i++){
    a[i]=a[i]+1;
}

for(i=1; i<100;i++){
    b[i]=b[i-1]+c[i];
}

for(i=1; i<100;i++){
    d[i]=b[i-1]+a[i];
}

for(i=1; i<100;i++){
    e[i]=e[i-1]+b[i+1];
}

for(i=1; i<100;i++){
    g[i]=d[i]+f[i];
}
```

Encoder

- Input: loop specifications generated by input preprocessing
- Output: extract a set of vectors, each of which specifies a loop
  - Characteristics of each loop
  - Correlation between loops
- Produce the action space of the decoder



Encoder



- Sublayer: multi-head attention
  - Attention

$$Attention(Q;K;V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Q: queries

K, V: key-value pairs

Attention weight

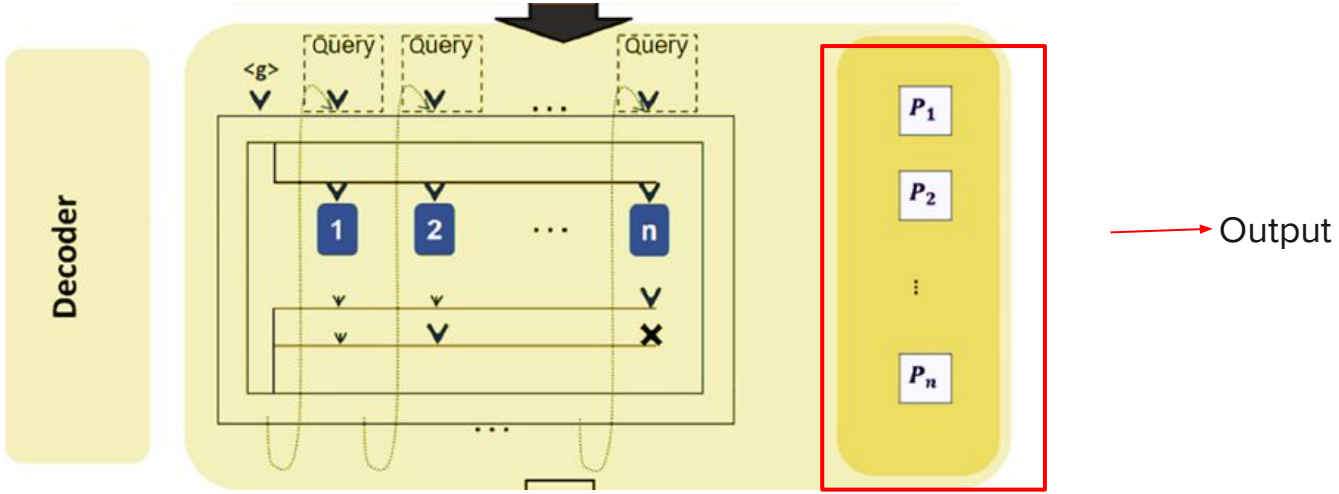
  - Multi-head
    - Attend to different parts of the input matrix "V" simultaneously
    - Use multiple sets of query, key, and value vectors.
    - Increase model capacity and performance

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

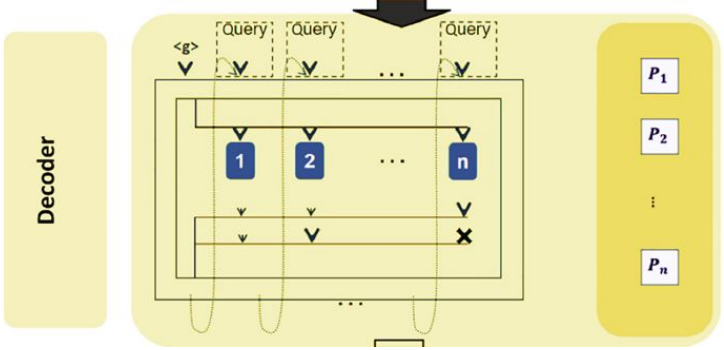
$$MultiHead(Q,K,V) = Concat(head_1, \ldots, head_h)$$

Decoder

- Input
  - Embedded loops vectors generated by encoder
- Output
  - The probability distribution of order for fusing loops



Decoder



- Calculate query vector
  - State representation in every single step
  - Consider the last three added loops (actions) at time t

$$q_t = ReLU\left(W_1 a_{\pi(t-1)} + W_2 a_{\pi(t-2)} + W_3 a_{\pi(t-3)}\right) \in \mathbb{R}^{d'}$$

- Calculate the probability distribution on the remaining loops
  - pointing mechanism

$$\forall i \leqslant n. u_i^t = \begin{cases} v^T \tanh\left(w_{ref} a_i + w_q q_t\right) & if\ i \notin \pi(0) \ldots \pi(t-1) \\ -\infty & otherwise. \end{cases}$$

$$p_\theta(\pi(t)|\pi(<t).s) = softmax(C \tanh(u^t/T))$$

ai: added loops (actions)
qt: query (state representation)
w: two attention matrix
v: attention vector
T: temperature hyper-parameter

Reward function

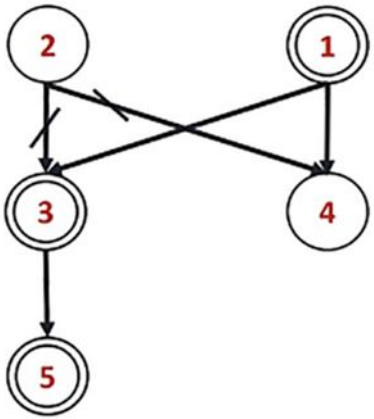
- Minimize

$$
\begin{aligned}
r(\pi|s) = & \textit{number of topological order violations in the considered sequence} \\
& + \textit{number of consecutive loops in the considered sequence that are not fusiable} \\
& + \textit{number of times that fusible loops' required register size passes the system's register size} \\
& + \textit{One tenth of the total weight of the edges between nodes in the sequence}
\end{aligned}
$$

- Encourage the agent to select loop orders that
  - maintain topological order
  - have more fusible paris
  - don’t exceed the available system register size
  - decrease the dependencies

Reward function

$$r(\pi|s) = \text{number of topological order violations in the considered sequence}$$
$$+ \text{number of consecutive loops in the considered sequence that are not fusiable}$$
$$+ \text{number of times that fusible loops' required register size passes the system's register size}$$
$$+ \text{One tenth of the total weight of the edges between nodes in the sequence}$$



2.01	7.1	1.1 ③	6.1	10.1
7.1	2.02	8.1	5.1 ①	5.1
10.1	10.1	2.01	7.1	1.1 ④
10.1 ②	10.1	7.1	2.02	7.1
10.1	10.1	10.1	7.1	2.01

An example fusion order: 2 4 1 3 5

$r(\pi|s) =$ (1)

not topological

not fusiable due to the fusion preventing edge between and wrong topological order

not fusiable due to their type variance and wrong topological order

$+(2)$

$+(1)$

LR (1+1) <= LS (2) fusiable

LR (1+1+1) > LS (2) not fusiable due to lack of register space

$+\left(\frac{①+②+③+④}{10}\right) = \frac{10+10+1+1}{10}$

$= 6.2$

Experiment

- Trained with a set of 1000 test cases, each with a maximum of 8 loops owing the size of the available benchmarks
- 8 * 8 matrices as input
- Compare with previous methods from two perspectives:

  1. Output in terms of fusion sequence and the resulting **number of loops**

  2. **Execution time** in sequential and parallel

Result

Table 4
Comparing the proposed method with comparative works in terms of fusion sequence and the resulted loops.

[Ref.]	Original loops sequence	Referred paper's result	DRLLFs result
[31]	|1|2|3|4|5|6|7|8|	**|1-2|3-4|5-6|7-8|**	**|1-2|3-4|5-6|7-8|**
	Number of loops: 8	**Number of loops: 4**	**Number of loops: 4**
[32]	|1|2|3|4|5|6|7|8|	**|1-3|2-4-6|5-8|7|**	**|1-3|2-4-6|5-8|7|**
	Number of loops: 8	**Number of loops: 4**	**Number of loops: 4**
[33]	|1|2|3|4|5|6|7|8|	|1-2-8|3-4-5-6|7|	**|1-2-3-8|7-4-5-6|**
	Number of loops: 8	Number of loops: 3	**Number of loops: 2**
[8]	|1|2|3|4|5|6|	**|2|1-3|4-5|6|**	**|2|1-3|4-5|6|**
	Number of loops: 6	**Number of loops: 4**	**Number of loops: 4**
[35]	|1|2|3|4|5|	|1|2|3-4-5|	**|1-2|3-4-5|**
	Number of loops: 5	Number of loops: 3	**Number of loops: 2**
[36]	|1|2|3|4|5|6|7|	**|1-2-3–4-6|7|5|**	**|1-2|7|3-4-5-6|**
	Number of loops: 7	**Number of loops: 3**	**Number of loops: 3**
[41]	|1|2|3|4|5|6|	**|1-2-4|3-5-6|**	**|1-2-4|3-5-6|**
	Number of loops: 6	**Number of loops: 2**	**Number of loops: 2**
Ben6[41]	|1|2|3|4|5|6|7|	**|1-3|4-7|2-6|5|**	**|1-3|4-7|2-6|5|**
	Number of loops: 7	**Number of loops: 4**	**Number of loops: 4**
LL8[41]	|1|2|3|4|5|6|	**|1-2-3–4-5-6|**	|1-2-3|4-5-6|
	Number of loops: 6	**Number of loops: 1**	Number of loops: 2
LL18[41]	|1|2|3|4|5|6|	**|1-2|4-3-5-6|**	**|1-2|4-3-5-6|**
	Number of loops: 6	**Number of loops: 2**	**Number of loops: 2**

Fewer loops

More loops

Result

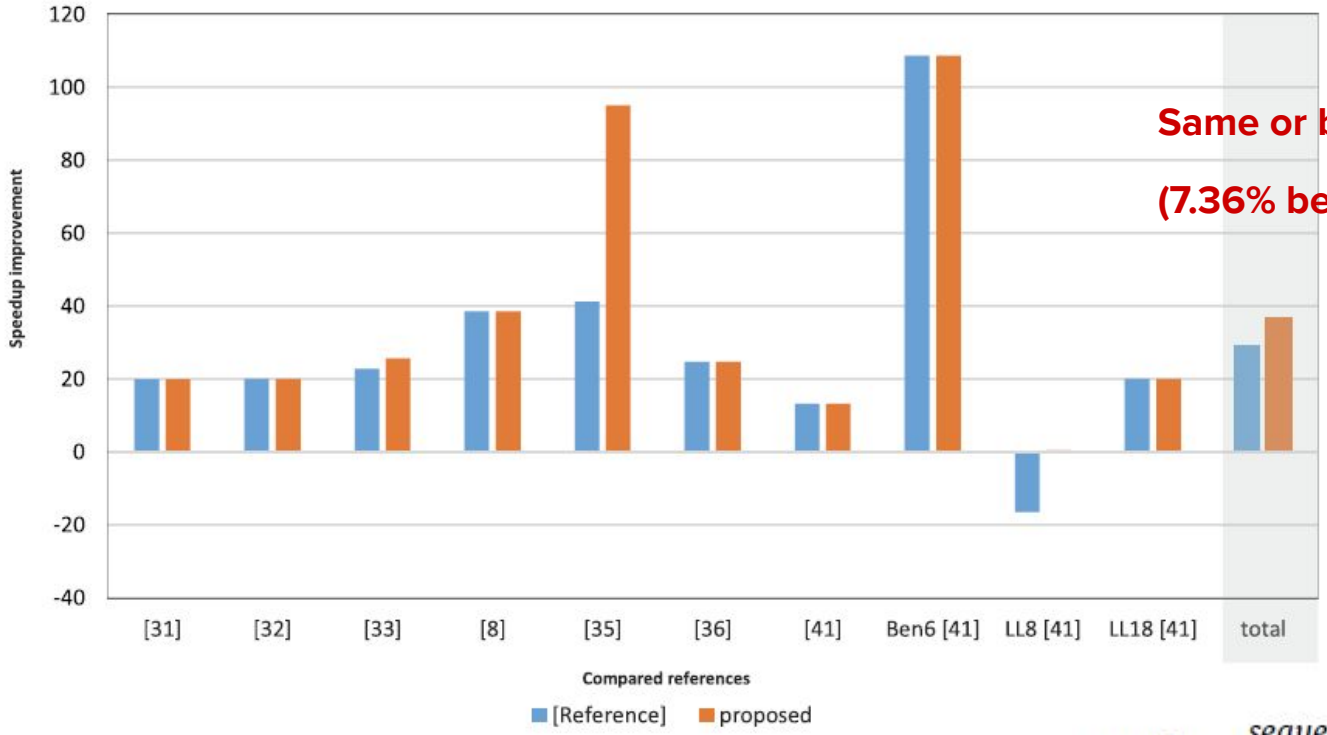


Fig. 8. The mean of speedup improvement rate.

Same or better!

(7.36% better in average)

$$speedup = \frac{sequential\ execution\ time}{parallel\ execution\ time}$$

Pros & Cons

- Pros
  - Effective deep reinforcement learning
  - Consider the system’s register size
  - Enhance parallelism
- Cons
  - Long training time (3 weeks for training!!)
  - May not generalize well to new datasets
  - Should consider more complex real-world dependencies

Future Work

- Explore the use of other deep learning techniques in loop fusion optimization, such as **CNN or RNN**.
- Investigate the application of the proposed method to other loop optimization problems, such as **loop tiling or unrolling**.
- Exploring the **use of unsupervised learning** methods to reduce the reliance on labeled training data and improve the scalability of the proposed approach.
- Investigate how this approach can be **integrated into existing parallelizing compilers** and evaluate its performance on a wider range of benchmarks.

Thanks

Q & A