Managing performance vs. accuracy trade-offs with loop perforation

Authors: Stelios Sidiroglou Sasa Misailovic Henry Hoffmann

Group 7: Max Froehlich, Murali Mohan, Austin Ye

Motivation

- Processing large quantities of data is often done in loops
- Improvements to these loops can have a huge impact for these applications
- Depending on the context, large loops such as these also don’t have strict accuracy requirements
- This is the perfect scenario for some accuracy/performance tradeoff optimizations

Applications

- Image/video processing and lossy encoding
- Optimization/simulated annealing
- Numerical methods

Loop perforation basics

Loop perforation transforms loops to execute a subset of their iterations

Goal: reduce the amount of computational work (and therefore the amount of time and/or other resources such as power)

Perforation rate, `r`: The percentage of iterations that we want to skip

In essence, we perform every nth iteration:

```
for (i = 0; i < b; i++) { ... }
```

→

```
for (i = 0; i < b; i += n) { ... }
```

Accuracy metric of loop perforation

$$acc = \frac{1}{m}\sum_{i=1}^{m} w_i \left|\frac{o_i - \hat{o}_i}{o_i}\right|$$

acc= weighted mean scaled difference between the output abstraction components from the original program and the perforated program

$o_1 \ldots o_m$ = output abstraction components from the original program

$\hat{o}_1 \ldots \hat{o}_m$ = output abstraction components from the perforated program

$w_1 \ldots w_m$ = weight that captures the relative importance of each component

acc → 0 means our perforation is better

Algorithm overview

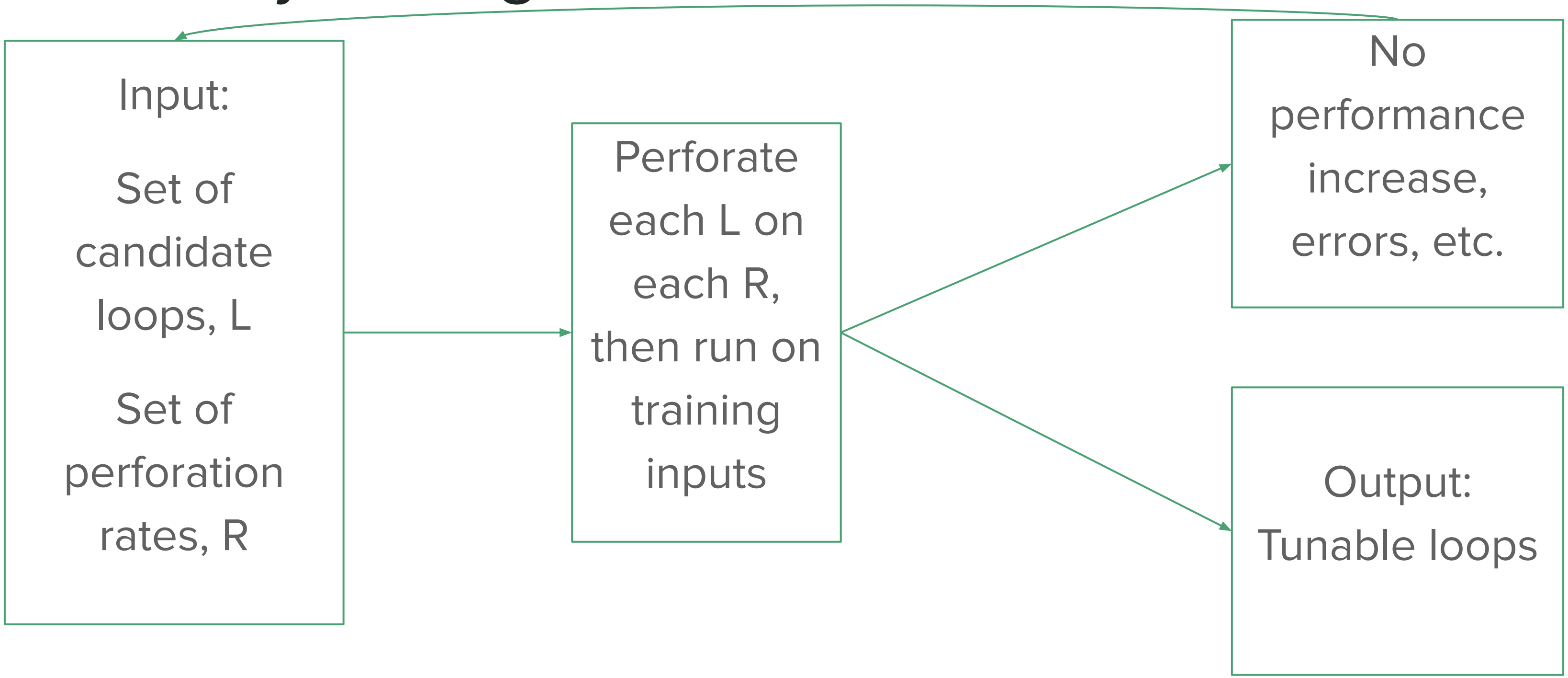
Criticality Testing

Filter out critical loops whose perforation causes the computation to produce unacceptable results, crash, etc.

Perforation Space Exploration

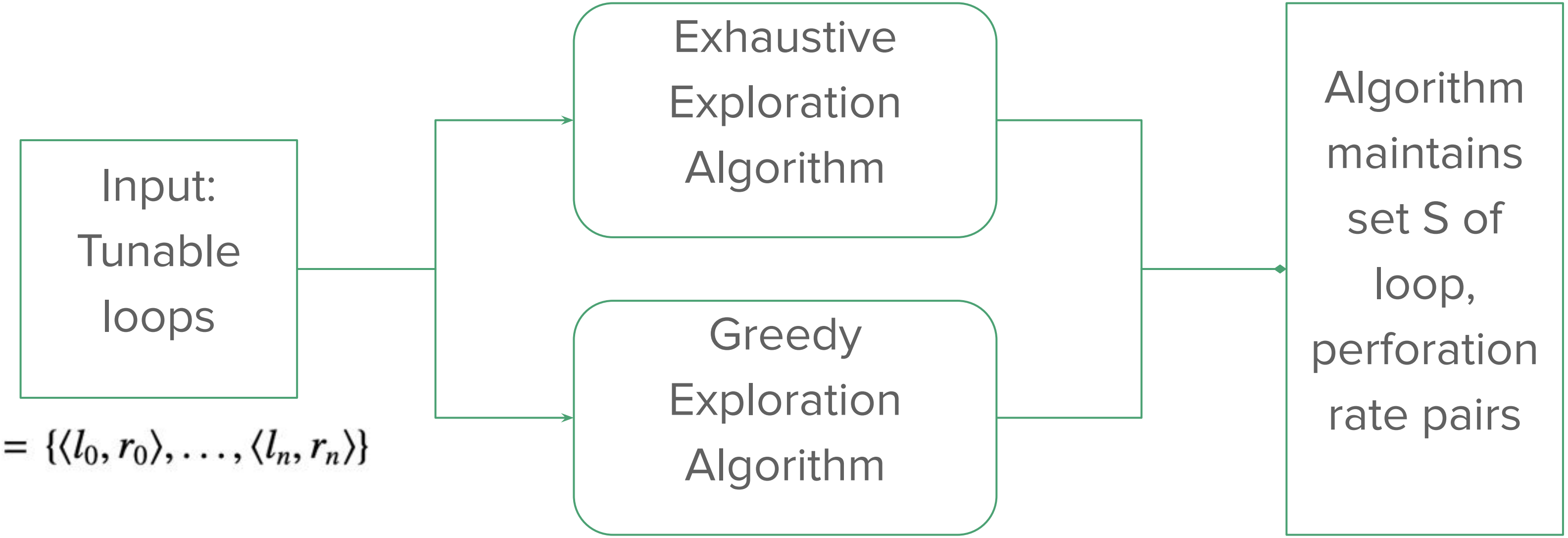
Exploration of the space of variants generated

Criticality testing



$$P = \{\langle l_0, r_0 \rangle, \ldots, \langle l_n, r_n \rangle\}$$

Perforation space exploration



$$P = \{\langle l_0, r_0 \rangle, \ldots, \langle l_n, r_n \rangle\}$$

$$score_{\langle l,r \rangle} = \frac{2}{\frac{1}{\overline{sp}_{\langle l,r \rangle} - 1} + \frac{1}{1 - \frac{\overline{acc}_{\langle l,r \rangle}}{b}}}$$

When to use loop perforation?

Applications that have some flexibility to change the result that they produce

Performance enhancement

Dynamic adaptation

Energy savings

Developer insight

New platforms or contexts

Reasons to use loop perforation

Results

- Generally, most of the loops in a program are *not* perforable
  - Fail one or more of the criticality tests (memory issues, unacceptable accuracy, etc.)

x264				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	1	1	1	1
Accuracy	6	7	7	6
Speed	16	12	10	11
Valgrind	0	0	1	1
Remaining	2	6	6	6
bodytrack				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	2	5	7	1
Accuracy	1	1	2	2
Speed	12	10	1	1
Valgrind	3	1	6	8
Remaining	7	8	9	13
swaptions				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	3	6	8	0
Accuracy	13	12	13	16
Speed	5	4	2	2
Valgrind	2	2	1	5
Remaining	2	1	1	2

ferret				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	8	12	12	6
Accuracy	13	11	11	17
Speed	0	0	0	0
Valgrind	0	0	0	0
Remaining	4	2	2	2
canneal				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	16	16	16	16
Crash	7	10	10	6
Accuracy	1	1	1	4
Speed	7	4	4	5
Valgrind	0	0	0	0
Remaining	1	1	1	1
blackscholes				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	6	6	6	6
Accuracy	4	4	4	4
Speed	1	1	1	1
Valgrind	0	0	0	0
Remaining	1	1	1	1

Results

- Performance gains are **highly** dependent on use case - some use cases didn't get much improvements while others got a 5x improvement in performance.
- Accuracy bounds didn't actually affect performance for a lot of use cases, but a few had great speedups by relaxing accuracy requirements
- Speedups generalize well across inputs not seen before

Application	Production			
	2.5%	5%	7.5%	10%
x264	2.34 (5.15%)	2.53 (6.08%)	3.12 (8.72%)	3.19 (10.3%)
bodytrack	2.70 (4.00%)	4.93 (6.12%)	4.811 (6.58%)	4.811 (6.58%)
swaptions	5.05 (0.2%)	5.05 (0.2%)	5.05 (0.2%)	5.05 (0.2%)
ferret	1.002 (0.15%)	1.02 (0.23%)	1.02 (0.23%)	1.07 (7.90%)
canneal	1.14 (4.38%)	1.14 (4.38%)	1.46 (7.88%)	1.46 (7.88%)
blackscholes	28.9 (0.0%)	28.9 (0.0%)	28.9 (0.0%)	28.9 (0.0%)
streamcluster	4.87 (1.71%)	4.87 (1.71%)	4.87 (1.71%)	4.87 (1.71%)

Commentary

- Very good introduction to loop perforation
- Training time and other practical limitations are not in the focus of the paper
- Accuracy in general case isn't thoroughly explored, especially as downsides relate to different contexts
- This especially leaves room for us to explore how loop perforation would extend to image operations

Application	Exhaustive	Greedy
x264	3071 (665m)	6 (9m)
bodytrack	5624 (1968m)	14 (33m)
swaptions	32 (9m)	4 (7m)
ferret	255 (43m)	6 (2m)
canneal	2 (12m)	1 (11m)
blackscholes	19 (1m)	3 (0.5m)
streamcluster	639 (3941m)	5 (31m)

4: Exhaustive and Greedy Search Times

Questions?