

# Managing Performance vs Accuracy Trade-offs With Loop Perforation

Presented By  
Harsh Agarwal  
IIT Hyderabad





# Authors

Stelios Sidiroglou , Sasa Misailovic , Henry Hoffmann , Martin Rinard

From Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology

Link - <https://people.csail.mit.edu/stelios/papers/fse11.pdf>



# Outline

1. Introduction
2. Algorithm
3. Experiment Results
4. Perforation Evaluation



# Introduction



# What is loop perforation ?

- **Loop perforation** provides a general technique to **trade accuracy for performance** by transforming loops to execute a **subset of their iterations**
- The goal is to **reduce** the **amount of computational work** (and therefore the amount of time and/or other resources such as power) that the computation requires to produce its result.
- A perforation space exploration algorithm perforates **combinations of tunable loops** to find **Pareto-optimal perforation policies**.
- Can produce significant performance gains (up to a factor of seven reduction in overall execution time) in return for small (less than 10%) accuracy losses.



# Scope of Loop perforation

- Applications that interact well with loop perforation have **some flexibility** to **change the result** that they produce (subject, of course, to accuracy requirements).
- They also contain some **underlying source of redundancy** that enables them to produce an acceptable result even after discarding parts of their computation.
- Ex : **video, audio, and image processing, Monte Carlo simulations, information retrieval** and machine learning computations, scientific and economics computations (output within an acceptable precision range).
- Not a universally applicable technique  
Ex: compilers, databases, and operating systems



# Why Loop Perforation Works ?

- Analysis gives a probabilistic guarantee that perforated computation is highly likely to produce a result that is close to the original result
- Some **local computational patterns** that interact well with loop perforation
  - Sum pattern (Sum)
  - Argmin pattern (Argmin)
  - Ratio of 2 sum (Ratio)
  - Mean of elements (Mean)
- Some **global computational patterns** that interact well with loop perforation
  - Search Space Enumeration (SSE)
  - Search Metric(SM) (Selection (S/E), Filtering(F), Termination(T))
  - Monte-Carlo Simulation (MC)
  - Iterative Improvement (II)
  - Data Structure Update (DSU)
- Exploit computations partially redundant at both the local (loop) and global (application) level  
Ex : searching a set (of similar terms) to find most desirable item



# Benefits of Loop Perforation

- Performance Enhancement
- Energy Savings
- New Platforms or Contexts
- Dynamic Adaptation
- Developer Insight





# Algorithm



# Overview of the Transformation

- The perforator works with any loop that the existing **LLVM loop canonicalization pass, loop-simplify**, can convert into the following form:  

```
for (i = 0; i < b; i++) { ... }
```
- loop perforation rate  $r$  - the expected percentage of loop iterations to skip.
- **Interleaving perforation** transforms the loop to perform every  $n$ -th iteration (here the perforation rate is  $r = 1 - 1/n$ ).  

```
for (i = 0; i < b; i += n) { ... }
```
- Focuses on interleaving, can be extended to
  - **Truncation** : skip a contiguous sequence of iterations at either the beginning or the end of the loop
  - **Random perforation** : randomly skips loop iterations



# ACCURACY METRIC

- Is the difference between an output from the original program and a corresponding output from the perforated program run on the same input.
- Decompose into
  - **output abstraction (oa)** - maps an output to a number or set of numbers,
  - **accuracy calculation** - measures the weighted mean scaled difference between the output abstractions from the original and perforated executions.


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

- $o_1, \dots, o_m$  - output abstraction components from original program
- $\hat{o}_1, \dots, \hat{o}_m$  from the perforated program
- $w_i$  - the relative importance of the  $i$ th component of the output abstraction
- Closer the accuracy metric  $acc$  is to zero, the more accurate the perforated program



# PERFORATION EXPLORATION

- In the loop perforation space exploration algorithm
  - Input - application, accuracy metric, training inputs, accuracy bound  $b$  (a maximum acceptable accuracy metric for the application) & a set of perforation rates
  - Output - set  $S$  of loops to perforate at specified perforation rates.
- Divided into 2 parts
  - **Criticality Testing** - find the set of set of tunable loops  $P = \{ \langle l_0, r_0 \rangle, \dots, \langle l_n, r_n \rangle \}$ , where  $\langle l_i, r_i \rangle$  specifies the perforation of loop  $l_i$  at rate  $r_i$ .
  - **Perforation Space Exploration Algorithms** - find the pareto-optimal combination of set of tunable loops obtained from criticality testing
    - Exhaustive Exploration
    - Greedy Exploration



# Criticality Testing

$L$  consists of loops that account for at least 1% of the executed instructions (identified by profiling)

perforating a candidate loop may

- cause the program to crash
- generate unacceptable output
- produce an infinite loop
- decrease its performance

---

**Algorithm 1** (Criticality Testing) Find the set of tunable loops  $P$  in  $A$  given training inputs  $T$  and accuracy bound  $b$

---

**Inputs:**

$A$  - an application

$T$  - a set of representative inputs

$b$  - an accuracy bound

$L$  - a set of candidate loops for perforation

$R$  - a set of perforation rates

**Outputs:**  $P$  - a set of tunable loops and perforation rates for  $A$  given  $b$  and  $T$

$P = \emptyset$

**for**  $\langle l, r \rangle \in L \times R$  **do**

Let  $A_{\langle l, r \rangle}$  be  $A$  with  $l$  perforated at rate  $r$

**for**  $t \in T$  **do**

Run  $A_{\langle l, r \rangle}$  on  $t$ , record speedup  $sp_t$  and accuracy  $acc_t$

$\overline{sp} = (\sum_{t \in T} sp_t) / \|T\|$ ;  $\overline{acc} = (\sum_{t \in T} acc_t) / \|T\|$

**if**  $\overline{acc} < b \wedge \overline{sp} > 1$  **then**

**for**  $t \in T$  **do**

Run  $A_{\langle l, r \rangle}$  using Valgrind to find  $Err_t$  (memory errors)

**if**  $\bigcup_{t \in T} Err_t = \emptyset$  **then**

$P = P \cup \{\langle l, r \rangle\}$

**return**  $P$

---



# Perforation Space Exploration Algorithms

## Exhaustive Exploration Algorithm

- Exhaustively explore all combinations of tunable loops  $l$  at their specified perforation rates  $r$ .
- Execute all combinations on all training inputs and record the resulting speedup and accuracy. It also runs each combination under Valgrind (detect memory leak),
- Use the results to compute the set of Pareto-optimal perforations in the induced performance vs. accuracy tradeoff space.
- Feasible for applications that spend most of their time in relatively few loops.



## Greedy Algorithm

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

- where  $\overline{sp}_{\langle l, r \rangle}$  and  $\overline{acc}_{\langle l, r \rangle}$  are the mean speedup and accuracy metric (respectively) for the  $\langle l, r \rangle$  perforation over all training inputs
- $b$  is the accuracy bound.
- heuristic - harmonic mean based metric require a much higher performance increase to select the loop that causes a small increase in accuracy loss.
- In comparison to other heuristic functions harmonic better than arithmetic mean or geometric mean.

---

**Algorithm 2** (Greedy Exploration) Find a set  $S$  of loops to perforate in  $A$  given training inputs  $T$  and accuracy bound  $b$

---

**Inputs:**

$A$  - an application

$T$  - a set of representative inputs

$b$  - an accuracy bound

$P$  - a set of tunable loops generated by Algorithm 1

**Outputs:**  $S$  - a set of loops to perforate in  $A$  given  $b$

---

$P' = \{ \langle l, r \rangle \mid \langle l, r \rangle \in P \wedge \forall p, score_{\langle l, r \rangle} \leq score_{\langle l, p \rangle} \}$

$S = \emptyset$

**for**  $\langle l, r \rangle \in P'$  in sorted order according to  $score_{\langle l, p \rangle}$  **do**

$C = S \cup \{ \langle l, r \rangle \}$

Let  $A_C$  be  $A$  with all loops in  $C$  perforated

**for**  $t \in T$  **do**

Run  $A_C$  on  $t$ , record speedup  $sp_t$  and accuracy  $acc_t$

$\overline{sp} = (\sum_{t \in T} sp_t) / \|T\|$ ;  $\overline{acc} = (\sum_{t \in T} acc_t) / \|T\|$

**if**  $\overline{acc} < b \wedge \overline{sp} > 1$  **then**

**for**  $t \in T$  **do**

Run  $A_C$  using Valgrind to find  $Err_t$  (memory errors)

**if**  $\bigcup_{t \in T} Err_t = \emptyset$  **then**

$S = C$

**return**  $S$

---



# Experiment Results





# Data Points

Benchmark	Training Inputs	Production Inputs	Source
x264	4 HD videos of 200+ frames	12 HD videos of 200+ frames	PARSEC & xiph.org [1]
bodytrack	sequence of 100 frames	sequence of 261 frames	PARSEC & additional input provided by benchmark authors
swaptions	64 swaptions	512 swaptions	PARSEC & randomly generated swaptions
ferret	256 image queries	3500 image queries	PARSEC
canneal	4 netlists of 2M+ elements	16 netlists of 2M+ elements	PARSEC & additional inputs provided by benchmark authors
blackscholes	64K options	10M options	PARSEC
streamcluster	4 streams of 19K-100K data points	10 streams of 100K data points	UCI Machine Learning Repository [2]

## 1: Summary of Training and Production Inputs

- **X264** - H.264 encoding on video data (output abstraction(oa)  $\rightarrow$  peak-signal-to-noise ratio, weight  $\rightarrow$  1)
- **Bodytrack** - track movement of body (computer vision) (oa  $\rightarrow$  vector representing body position, weight  $\rightarrow$  vector magnitude)
- **Swaptions** - use monte carlo simulation solve PDE that prices swaptions (oa  $\rightarrow$  swaption prices, weight  $\rightarrow$  1)
- **Ferret** - Content-based similarity search on images (acc  $\rightarrow$  1 - intersection of sets of images/size of original set of images ))
- **Canneal** - Simulated annealing to minimize routing cost of microchip design (oa  $\rightarrow$  total routing cost, weight  $\rightarrow$  1)
- **Blackscholes** - Solve PDE to find price of portfolio of European options (oa  $\rightarrow$  price, weight  $\rightarrow$  1)
- **Streamcluster** - online clustering (acc  $\rightarrow$  quality of clustering BCubed metric)



# Criticality Testing Results

Application	Algorithm 1		Total Time
	Accuracy	Valgrind	
x264	500 (108m)	110 (840m)	949m
bodytrack	100 (35m)	47 (1316m)	1351m
swaptions	100 (7m)	16 (108m)	115m
ferret	100 (17m)	40 (53m)	71m
canneal	256 (405m)	60 (540m)	945m
blackscholes	24 (0.5m)	12 (5m)	5.5m
streamcluster	500 (3083m)	17 (782m)	3865m

## 2: Criticality Testing Statistics for Benchmark Applications

- X(Y) indicates that algorithm considered X different combinations of perforated loops and that the executions took a total of Y minutes to complete.

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
<b>Remaining</b>	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
<b>Remaining</b>	7	8	9	13

# **Criticality Testing Results for Individual Loops**

(acc bound - 10%)

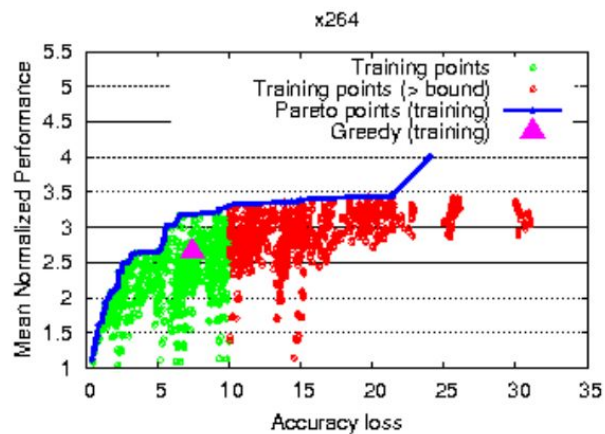
ferret				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	8	12	12	6
Accuracy	13	11	11	17
Speed	0	0	0	0
Valgrind	0	0	0	0
<b>Remaining</b>	4	2	2	2

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
<b>Remaining</b>	1	1	1	1

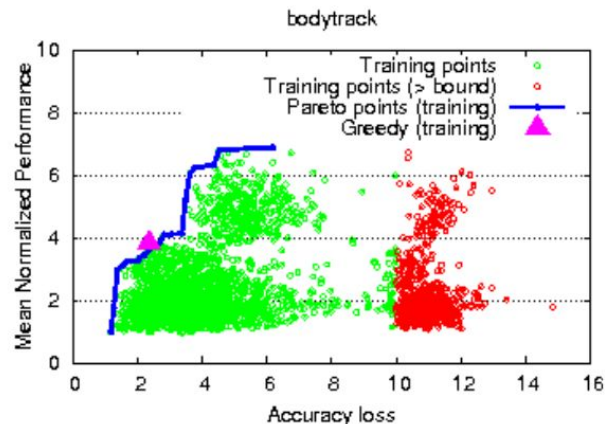
blackscholes				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	6	6	6	6
Crash	4	4	4	4
Accuracy	1	1	1	1
Speed	0	0	0	0
Valgrind	0	0	0	0
<b>Remaining</b>	1	1	1	1

streamcluster				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	15	15	15	15
Crash	1	1	2	4
Accuracy	0	0	0	0
Speed	13	11	8	7
Valgrind	0	0	0	0
<b>Remaining</b>	1	3	5	4

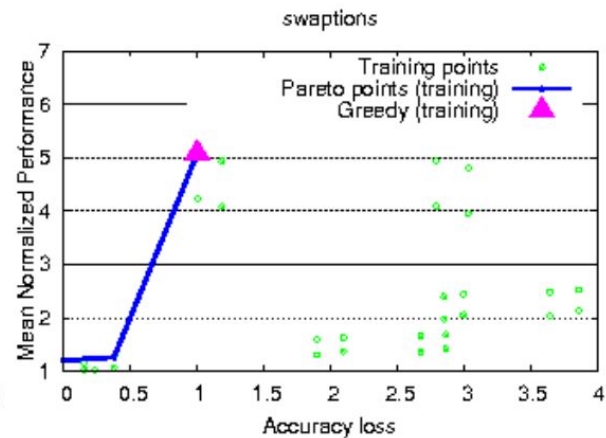
swaptions				
Filter	0.25%	0.50%	0.75%	1 iter
Candidate	25	25	25	25
Crash	3	6	8	0
Accuracy	13	12	13	16
Speed	5	4	2	2
Valgrind	2	2	1	5
<b>Remaining</b>	2	1	1	2



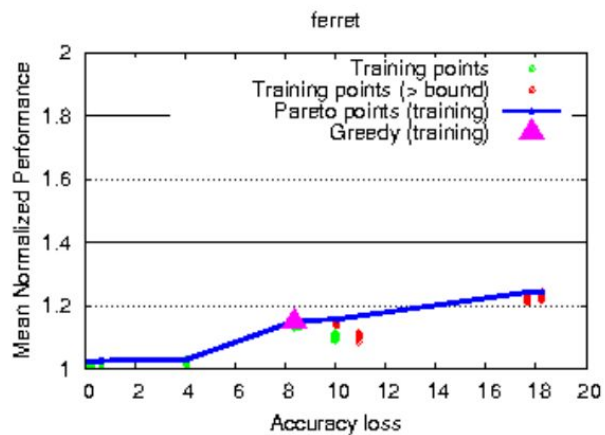
1: x264 (exhaustive)



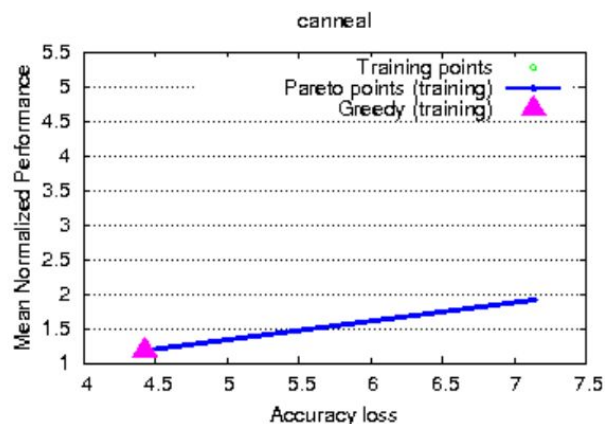
2: Bodytrack (exhaustive)



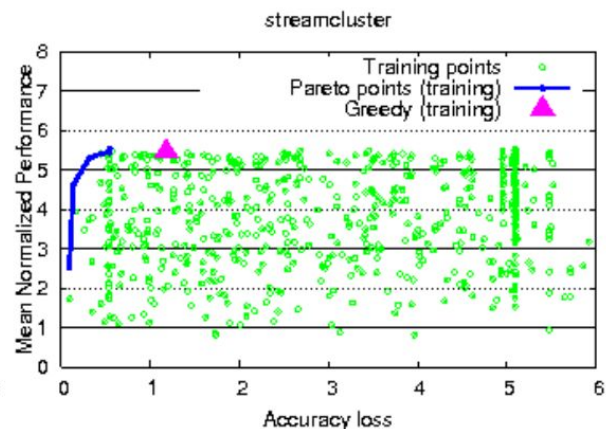
3: Swaptions (exhaustive)




4: Ferret (exhaustive)



5: Canneal (exhaustive)



6: Streamcluster (exhaustive)



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

- X(Y) indicates that algorithm considered X different perforated versions of the application and that the executions took a total of Y minutes to complete.



# Perforation Evaluation



# Training and Production Results

Application	Training				Production			
	2.5%	5%	7.5%	10%	2.5%	5%	7.5%	10%
x264	2.38 (2.5%)	2.66 (5%)	3.17 (6.53%)	3.25 (9.31%)	2.34 (5.15%)	2.53 (6.08%)	3.12 (8.72%)	3.19 (10.3%)
bodytrack	3.44 (2.23%)	6.32 (4.36%)	6.89 (6.19%)	6.89 (6.19%)	2.70 (4.00%)	4.93 (6.12%)	4.811 (6.58%)	4.811 (6.58%)
swaptions	5.08 (1%)	5.08 (1%)	5.08 (1%)	5.08 (1%)	5.05 (0.2%)	5.05 (0.2%)	5.05 (0.2%)	5.05 (0.2%)
ferret	1.02 (0.2%)	1.03 (4%)	1.03 (4%)	1.16 (10%)	1.002 (0.15%)	1.02 (0.23%)	1.02 (0.23%)	1.07 (7.90%)
canneal	1.14 (4.38%)	1.18 (4.43%)	1.913 (7.14%)	1.913 (7.14%)	1.14 (4.38%)	1.14 (4.38%)	1.46 (7.88%)	1.46 (7.88%)
blackscholes	33 (0.0%)	33 (0.0%)	33 (0.0%)	33 (0.0%)	28.9 (0.0%)	28.9 (0.0%)	28.9 (0.0%)	28.9 (0.0%)
streamcluster	5.51 (0.54%)	5.51 (0.54%)	5.51 (0.54%)	5.51 (0.54%)	4.87 (1.71%)	4.87 (1.71%)	4.87 (1.71%)	4.87 (1.71%)

5: Training and Production Results for Pareto-optimal Perforations for Varying Accuracy Bounds

Mean Speedup (Mean Accuracy)

x264		
Function	Time	Type
x264_mb_analyse_inter_p16x16	64.20%	SSE / Argmin
x264_pixel_sad_16x16, outer	55.80%	SMS+T / Sum
x264_pixel_sad_16x16, inner	54.60%	SMS+T / Sum
x264_me_search_ref	25.00%	SSE / Argmin
pixel_satd_wxh, outer	18.50%	SMS+T / Sum
pixel_satd_wxh, inner	18.30%	SMS+T / Sum

bodytrack		
Function	Time	Type
Update	77.00%	II
ImageErrorInside, inner	37.00%	SME / Ratio
ImageErrorEdge, inner	29.10%	SME / Ratio
InsideError, outer	28.90%	SME / Sum
IntersectingCylinders	1.16%	SMF+SSE

swaptions		
Function	Time	Type
HJM_Swaption_Blocking, outer	100.00%	MC / Mean
HJM_SimPath_Forward_Blocking, outer	45.80%	DSU
HJM_SimPath_Forward_Blocking, inner	31.00%	DSU
Discount_Factors_Blocking	1.97%	DSU

## Local / Global Patterns in Pareto-optimal Perforations

ferret		
Function	Time	Type
emd	37.60%	SMS+II
LSH_query_bootstrap, outer	27.10%	SSE
LSH_query_bootstrap, middle	26.70%	SSE
LSH_query_bootstrap, inner	2.70%	SSE

cannal		
Function	Time	Type
reload	2.38%	DSU

blackscholes		
Function	Time	Type
bs_thread	98.70%	–

streamcluster		
Function	Time	Type
pFL, inner	98.50%	II
pgain	84.00%	SME+T+DSU
dist	69.30%	SME+T / Sum
pgain	5.01%	SME+T





# Related Work

- One can trade accuracy for performance robustness, energy consumption, fault tolerance , and efficient parallel execution.
- Loop Perforation and Task Skipping
  - loop perforation can be seen as a special case of task skipping
  - Task Splitting : 1st publication used linear regression to obtain empirical statistical models of the time and accuracy effects of skipping tasks
  - Loop Perforation Timeline
    - 2009 - empirical justification of loop perforation with no formal reasoning
    - 2010 - first statistical justification using Monte Carlo simulation
    - 2011 - first probabilistic justification for loop perforation using static analysis of local computational patterns
    - 2011 - assume that inputs form a Gaussian random walk and the loop body is a robust function, derives a probabilistic bound to provide a justification for applying loop perforation.
    - Present paper - analyze loop perforation and global computational pattern



# Conclusion

- loop perforation can effectively augment a range of applications with the ability to operate at various attractive points in the tradeoff space
- perforated applications often deliver significant performance improvements (typically around a factor of two reduction in running time) at the cost of a small (typically 5% or less) decrease in the accuracy.
- Within target class of applications, it can dramatically increase the ability of applications to trade off accuracy in return for other benefits such as increased performance and decreased energy consumption.