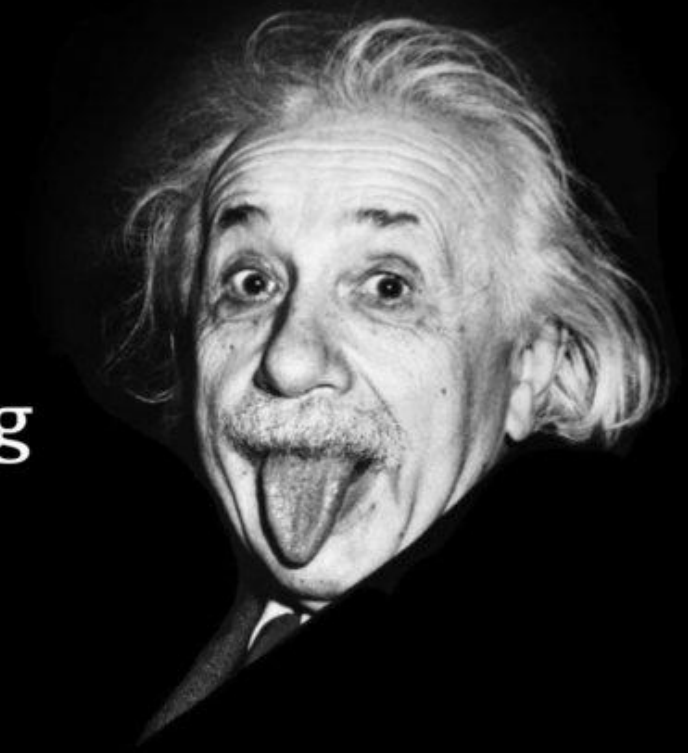
Generating Robust Parallel Programs via Model Driven Prediction of Compiler Optimizations for Non-determinism

Girish Mururu, Kaushik Ravichandran, Ada Gavrilovska, Santosh Pande

CREATING THE NEXT®

"Insanity is doing the same thing over and over again and expecting different results"

Albert Einstein



Computers can do the same thing over and over again emitting different results

Are computers **sane** or **Insane**?

Are Computers Sane or Insane ?

Regardless of using them to do sane or insane things, computers are **Non-deterministic**

Non-determinism

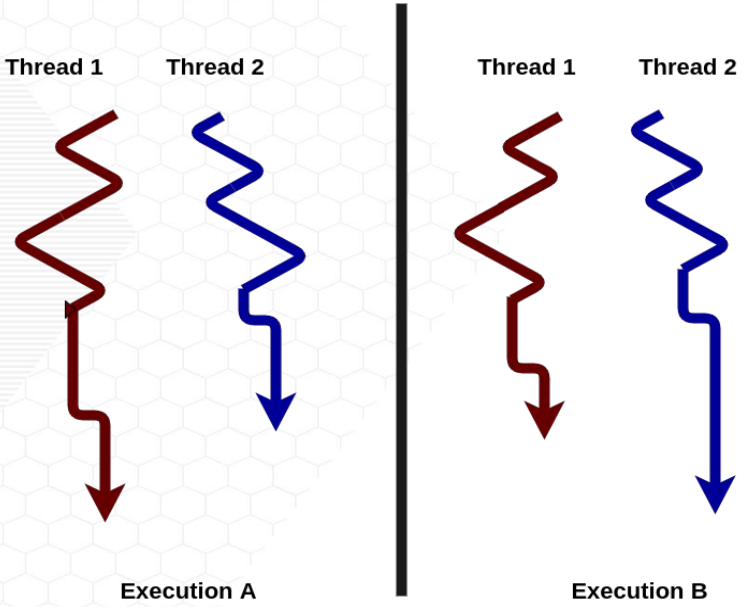
- ➔ Variant behavior exhibited during repeated execution with same input
- ➔ Different sources
  - ◆ Architecture, OS, runtime

Non-determinism in Sequential Programs

- ➔ Co-executing programs
- ➔ Context switches
- ➔ Architectural causes
  - ◆ Branch mispredictions , cache and TLB, misses etc.

Non-determinism in Parallel Programs

- ➔ In addition to sequential causes
- ➔ interference
- ➔ resource sharing
- ➔ scheduling decisions



Parallel Programs: Implications of Non-determinism

- ➔ Can suffer from extreme to debug concurrency behaviors
- ➔ Uncommon thread interleavings (Corner cases)
  - ◆ Can carry bug into production
  - ◆ Difficult for developers to reproduce
- ➔ Occurrence of the bug is related to Non-determinism
  - ◆ Thread interleavings and more (such as a pointer state)

Non-determinism and Bugs

- ➔ Non-determinism in parallel programs causes
  - ◆ Variability in thread execution interleavings
- ➔ Influencing Non-determinism affects rare interleavings
  - ◆ More Non-determinism, more interleavings possible
  - ◆ Less non-determinism, less interleavings possible
- ➔ Can Non-determinism be a control-knob for **Robustness?**

Non-determinism and Robust Programs

- ➔ Decreasing Non-determinism
  - ◆ Makes exhibition of a rare thread interleaving *less likely* in production
  - ◆ Testing covers most *likely* behaviors (interleavings)
- ➔ Increasing Non-determinism
  - ◆ Help developers reproduce rare bugs exercising corner cases of interleavings
- ➔ Completely Removing non-determinism slows down programs (prior works)

Compiler flags - debug vs optimize

- Developers select compiler optimization levels
- For speed or code-size
  - Release - e.g. O3
  - Debugging convention - e.g. O0
- Can compiler optimization effect Non-determinism?
  - No known non-determinism models

Goals

- ➔ We quantify and characterize architectural non-determinism
  - ◆ Develop non-determinism model
    - Use Hardware Performance Counters
  - ◆ Minimize OS effects
- ➔ Use model to predict the compiler optimization level that corresponds to the least and most non-determinism
  - ◆ We empirically show this correlation through regression testing
- ➔ We use the predicted flag to generate the corresponding debuggable or robust parallel program

Definitions

- Commit Order - order in which threads complete
  - E.g. 2,1,3,4 or 4,3,2,1 , different permutations
  - Commit orders possible = **n!**
- Non-determinism - Unique commit orders
  - All n! orders need not be exhibited
  - A new permutation exposes new synchronization behavior

Definitions

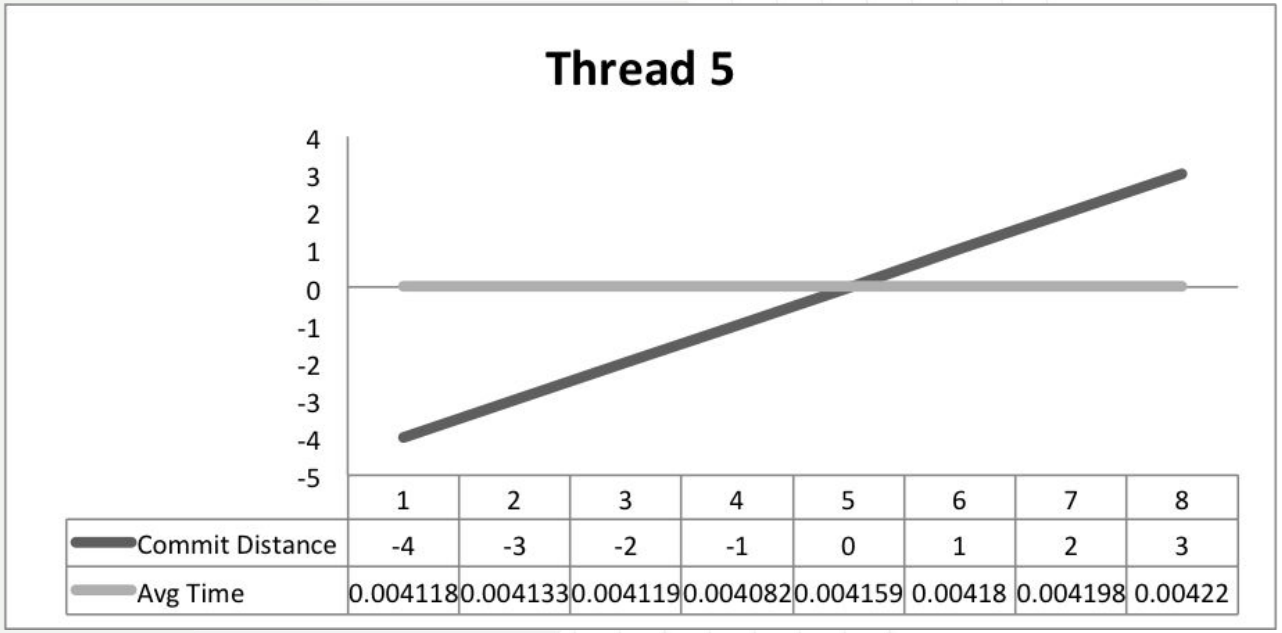
- Most Common Permutation (MCP)
  - The most frequent commit order
- Commit Distance of a thread (d_t)
  - Distance from the usual position in MCP
    - **If** MCP → 2,3,1,4 ; **then** 1,2,3,4 → d = 2 for thread 1

Pos → 0 1 2 3 Pos → 0 1 2 3

Definitions

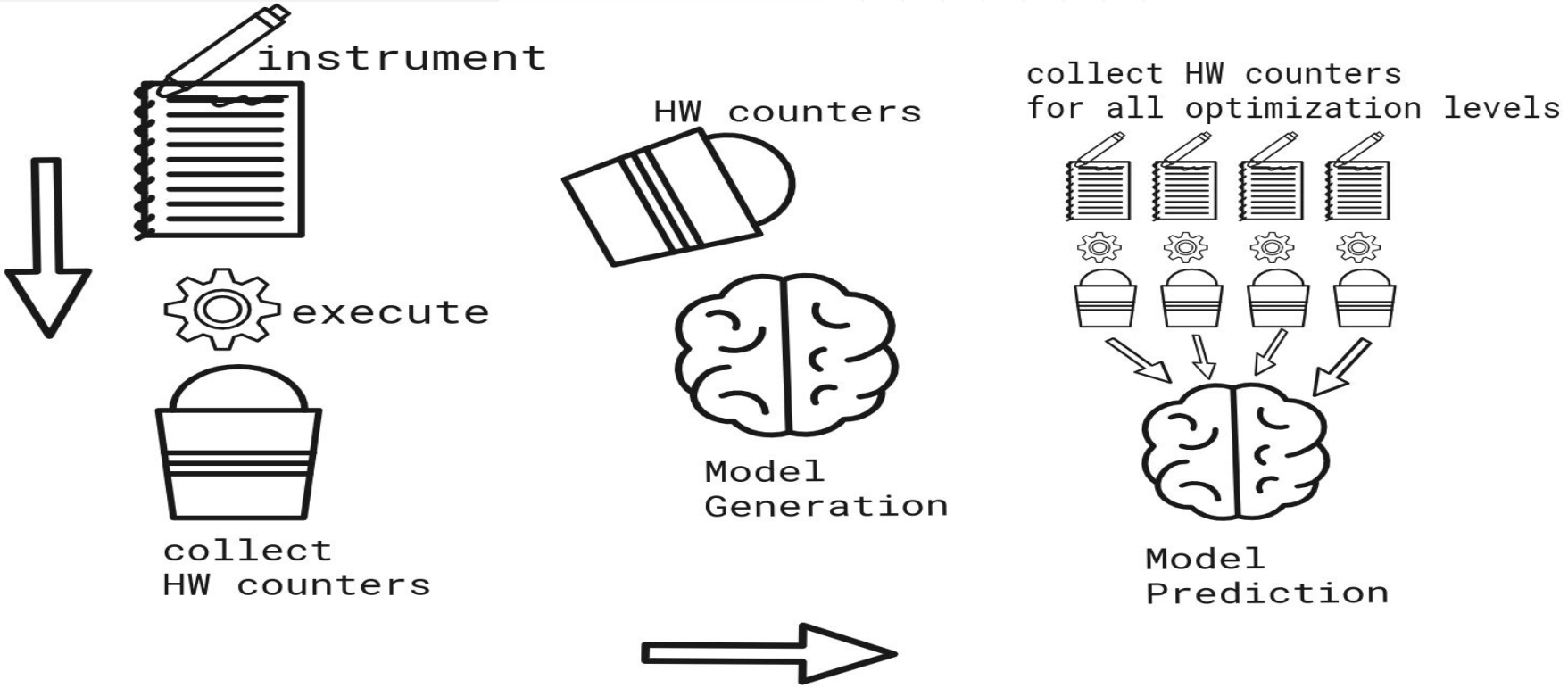
- Architectural Factors:
  - Architectural artifacts such as cache misses, branch mispredicts etc. which influence thread execution time
- We model commit distance in terms of change in artifacts
  - $$d_t = f(\Delta h1_{avg}, \Delta h2_{avg} \ldots \Delta hz_{avg})$$
  - h_1 , h_2, … h_z are different architectural artifacts

Model Generation



	1	2	3	4	5	6	7	8
Commit Distance	-4	-3	-2	-1	0	1	2	3
Avg Time	0.004118	0.004133	0.004119	0.004082	0.004159	0.00418	0.004198	0.00422

Overall Framework



Model Generation

- Profile hardware counters per thread
  - Average the counters for a thread in a commit order

$$h_{j_{avg}} = \frac{\sum_{k=0}^{m} h_{j_k}}{m}, \forall 1 <= j <= z$$

- Deviation from the hardware counter of the thread in MCP

$$\Delta h_{j_{avg}} = h_{j_{avg}} - hg_{j_{avg}}, \forall 1 <= j <= z.$$

Model Generation

- We learn non-deterministic behavior of each thread
  - Correlate commit distance to deviation in architectural artifacts

$$d_t = f(\sum_{j=1}^{z} \Delta h_{j_{avg}})$$

- We learn a linear regression because the relation between commit distance and execution time is linear

Model Generation Details

- ➔ We use R to find the coefficients in the linear model:

$$d_t = f(\sum_{j=1}^{z} \Delta h_{j_{avg}})$$

- ➔ Small and Medium inputs for training
- ➔ We normalize the hardware counter values of all commit orders over MCP

$$\Delta h_{j_{avg}} = \frac{h_{j_{avg}} - hg_{j_{avg}}}{hg_{j_{avg}}}, \forall 1 <= j <= z$$

Model Generation Details

- ➔ We generate a model for every step size (500 runs) using O0
- ➔ Check for convergence
  - ◆ Adjusted rsquared (**ar**): statistical measure of correctness
  - ◆ New thread model is adopted if **ar** is greater than previous
- ➔ Stopping condition: once model stabilizes for at least 4 iterations
- ➔ We use drop-term from MASS library to remove factors

Model-based Prediction

- Run O0, O1, O2, O3 at step size (500 runs)
- Plug average hardware counter values in the model
- All thread absolute values are added
- Must exercise the model adequate number of times
  - Too little will not capture correlation
  - Very large number of runs take too much time (infeasible)
- Stopping condition:
  - Based on Bollinger Bands

Model-based Prediction

- ➔ Bollinger Bands are used to measure stability (convergence)
- ➔ Each iteration generates **six** Bollinger band values
  - ◆ Last value(LV), middle(M), upper(U), lower(LB), bandwidth(BW), percent(P)
- ➔ Stopping condition:
  - ◆ Majority of values (4/6) are min or max
  - ◆ Over four successive iterations
  - ◆ Trend stabilization based on first four (LV, M, U, LB) or derivatives (BW, P)
- ➔ Model predicts the Least or Most non-deterministic opt level

Experiments

- ➔ Eight core Intel Xeon with Linux
- ➔ Threads pinned!
- ➔ GCC- 4.7
- ➔ PARSEC pthread benchmarks
- ➔ MCP observed in all benchmarks

Specifications	Values
CPU	Intel Xeon E5530
Core count	8
CPU Base frequency	2.40 GHz
Sockets	2
Core/socket	4
HyperThreading	disabled
LLC Cache	8 MB
Page Size	2MB
Main Memory	48 GB
OS	Linux

Model Accuracy & Non-determinism Reduced

➔ Verified by running each version 80,000 times

Benchmark	Most	Prediction Error
Blackscholes	O3	7.9%
Swaptions	O3	3.8%
Canneal	O2	0%
Bodytrack	O1	0%
Fluidanimate	O1	38%
Streamcluster	O2	13%
Raytrace	O0	0%

Least	Non-Determinism Reduced(%)
O2	25.94
O0	12.58
O0	32.35
O0	80.23
O2 (0.0%)	30.5
O3	15.34
O2	7.41

➔ Least Non-determinism prediction is ~100% accurate

➔ Non-determinism Reduced up to 80%

Model Characteristics

Benchmark	Model #runs	Gen Time (mins)	Prediction #runs	Pred Time(mins)
Blackscholes	9500	58	5500	183.6
Swaptions	2500	17.16	1500	65.35
Canneal	3500	17.501	18500	490.12
Bodytrack	7000	209.53	1500	174.25
Fluidanimate	1500	6.68	3500	61.21
Streamcluster	5000	3.30	3500	6.62
Raytrace	2000	6.19	1500	23.98

Bodytrack Model

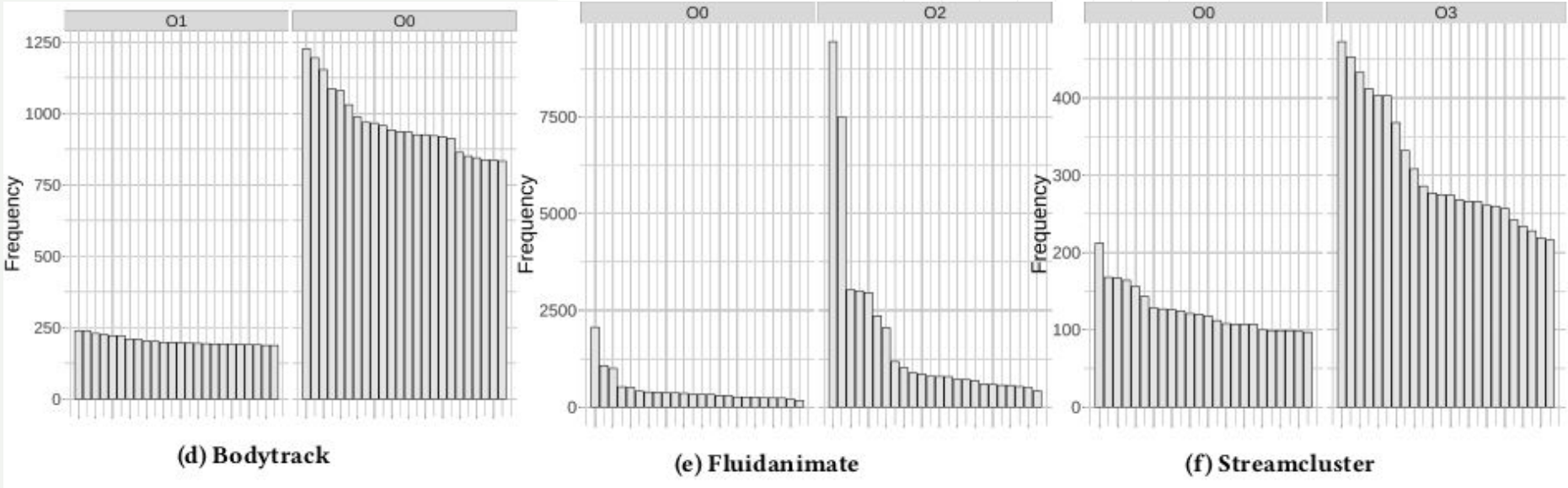
Thread	Linear Model
Thread0	0.047170Δh14 + 0.000902Δh5 + 0.003052Δh6 + 0.005743Δh1 + -0.000608Δh4 + 0.000129Δh7 + -0.036770Δh13 + -0.007761Δh12 + 0.000193Δh15
Thread1	-0.004823Δh14 + -0.000140Δh5 + 0.004911Δh6 + -0.002535Δh1 + 0.000493Δh4 + 0.000586Δh7 + 0.005430Δh13 + 0.215039Δh12 + 0.001700Δh15
Thread2	-0.052642Δh14 + 0.000185Δh5 + 0.004871Δh6 + 0.006364Δh1 + -0.001203Δh4 + 0.000815Δh7 + 0.095173Δh13 + 0.286653Δh12 + 0.001542Δh15
Thread3	-0.001099Δh14 + -0.000692Δh5 + 0.010845Δh6 + 0.001082Δh1 + -0.001843Δh4 + 0.000357Δh7 + 0.075876Δh13 + 0.073614Δh12 + 0.001148Δh15
Thread4	0.216100Δh14 + -0.000411Δh5 + 0.010000Δh6 + -0.002152Δh1 + -0.001766Δh4 + 0.000003Δh7 + 0.056330Δh13 + -0.508800Δh12 + 0.000469Δh15
Thread5	0.040220Δh14 + 0.001287Δh5 + 0.004525Δh6 + 0.000302Δh1 + -0.002025Δh4 + 0.000351Δh7 + -0.014940Δh13 + -0.251100Δh12 + 0.000449Δh15
Thread6	-0.034110Δh14 + 0.000163Δh5 + 0.001886Δh6 + 0.000022Δh1 + 0.000548Δh4 + 0.000757Δh7 + -0.074510Δh13 + 0.044290Δh12 + 0.000449Δh15
Thread7	-0.014660Δh14 + 0.000183Δh5 + 0.002347Δh6 + 0.003954Δh1 + 0.000489Δh4 + 0.000794Δh7 + -0.054610Δh13 + 0.008929Δh12 + 0.000298Δh15

Correlating Factors

Benchmark	Architectural Artifacts
Blackscholes	B, C, F, L, S, T
Swaptions	B, C, F, L, R, S, T
Canneal	B, C, F, L, R, S, T
Bodytrack	B, C, L, S, T
Fluidanimate	C
Streamcluster	C
Raytrace	B, C, F, L, R, S, T

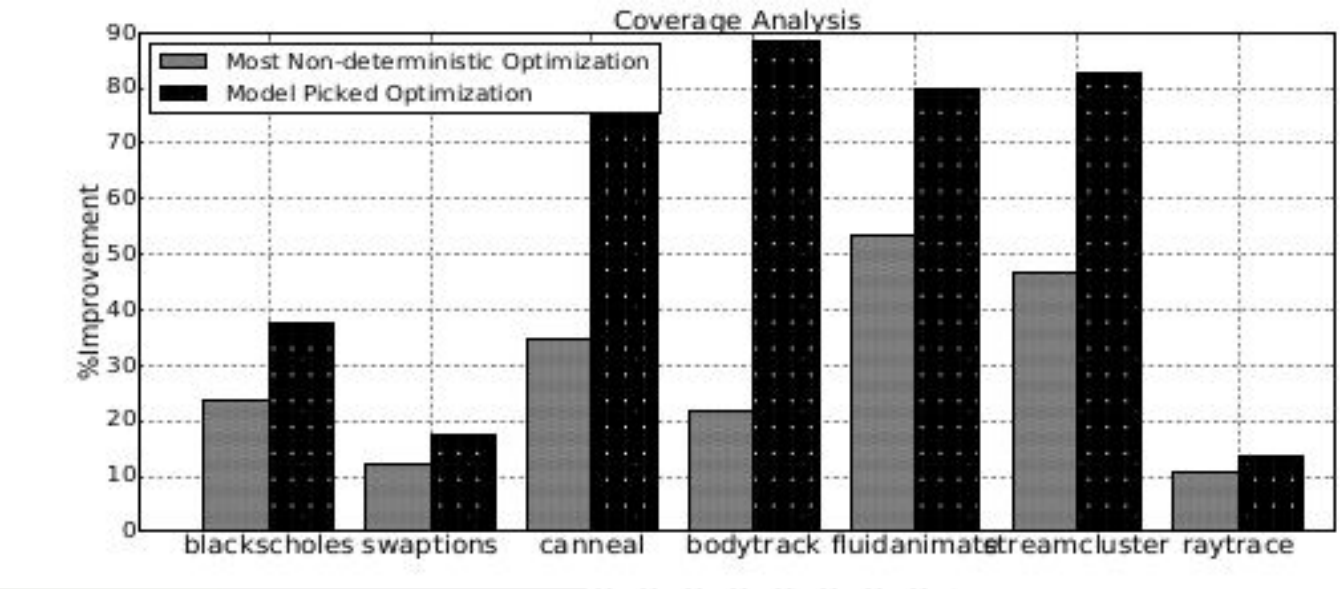
(B - branches) (C - cache) (F - floating point operations) (L - loads) (S - stores) (R - resource stall cycles) (T - TLB)

Most vs Least Non-determinism



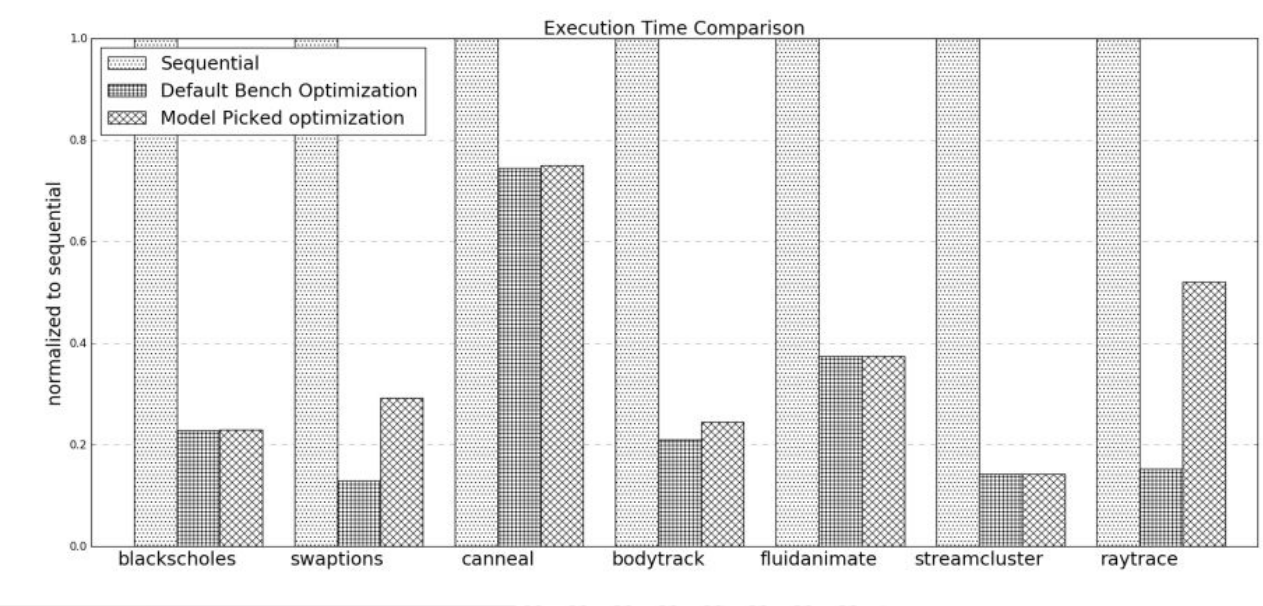
Robustness Improvement via Better Coverage

➔ Percentage of 80,000 runs covered by top 1000 permutations



Performance Comparison with Default Optimization

➔ Section specific compilation achieves close to default opt



Race Conditions Made Benign

- Known race conditions
  - Parsec 2.1 Streamcluster known race condition
    - Caused by a missing barrier
- Random race conditions (10 race conditions inserted)
  - Blackscholes - Debug 6 (11 failures), Release failed once
  - Swaptions - Debug 7 (one 7 times) , Release 5 (once)

Related Works

- ➔ DeSTM: harnessing determinism in STMs for application development
  - ◆ Kaushik Ravichandran, Ada Gavrilovska, Santosh Pande (Pact '14)
  - ◆ Improves debugging through deterministic replay of permutations
- ➔ Quantifying and Reducing Execution Variance in STM via Model Driven Commit Optimization
  - ◆ Girish Mururu, Ada Gavrilovska, Santosh Pande (CGO 19, PPoPP 18)'
  - ◆ Model Non-determinism to reduce timing variance in STM

Future work

- ➔ Different definitions of non-determinism
- ➔ Modelling other sources of non-determinism
  - ◆ OS, runtime
- ➔ Compiler optimizations geared towards non-determinism
- ➔ Debugging methodologies by actively controlling non-determinism

Conclusion

- ➔ Systematically models non-determinism
- ➔ Model predicts most and least non-deterministic versions
- ➔ Least non-deterministic version reduces non-determinism
  - ◆ Up to 80.23%, Coverage up to 66.48%
  - ◆ Make bugs benign
- ➔ Most non-deterministic version easy to reproduce bugs