MILEPOST Project Experience: building an ML-based self-optimizing compiler

Invited talk, Google compiler+ML seminar, July 2020

Based on the following articles:

2011 "Milepost GCC: Machine Learning Enabled Self-tuning Compiler"
International Journal on Parallel Programming 39(3): 296-327
Collaboration with IBM, INRIA, U.Edinburgh, CAPS, Arc (Synopsys)

link.springer.com/article/10.1007/s10766-010-0161-2
github.com/ctuning/reproduce-milepost-project

2015 "Collective Mind, Part II: Towards Performance- and Cost-Aware Software Engineering as a Natural Science"

Collaboration with STMicroelectronics and Arm

arxiv.org/abs/1506.06256

2018 "A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques"

Collaboration with the Raspberry Pi foundation
cKnowledge.io/report/rpi3-crowd-tuning-2017-interactive

Grigori Fursin gfursin@cknowledge.io cKnowledge.org

Founder of the Collective Knowledge project
President of the cTuning foundation

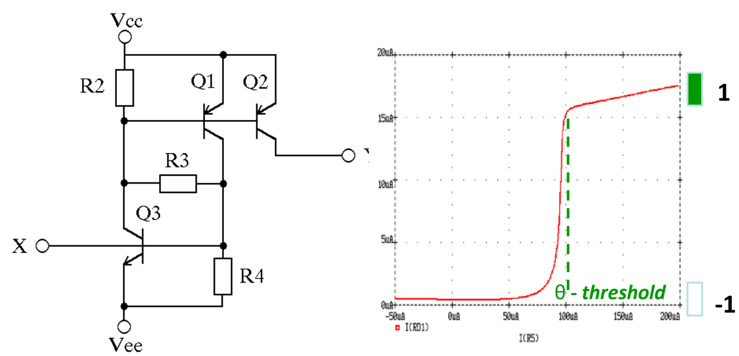
My background: physics, electronics, computer engineering, ML

Emulation of training/prediction on personal computers or supercomputers (Cray T3D)

- Hopfiled Neural Network
- ↓ Borland / Cray C compilers
- ↓ Binary with assembler inlines
- Images
- ↓ Run-time environment
- ↑ Intel/AMD/Alpha hardware
- ↑ PSpice simulator

I switched to computer engineering to learn how to make software and hardware more efficient

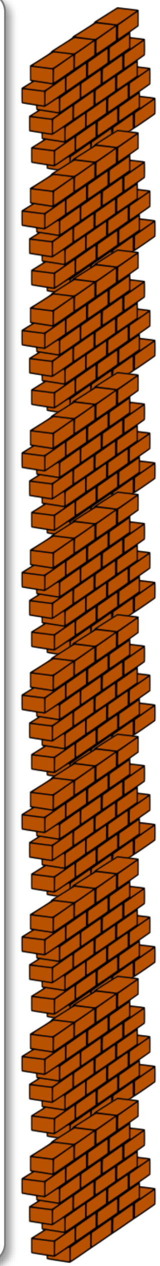
Designing analog semiconductor neural networks for brain-inspired computer systems (1995-1998)



10x smaller
100x faster / more energy efficient
10x more accurate

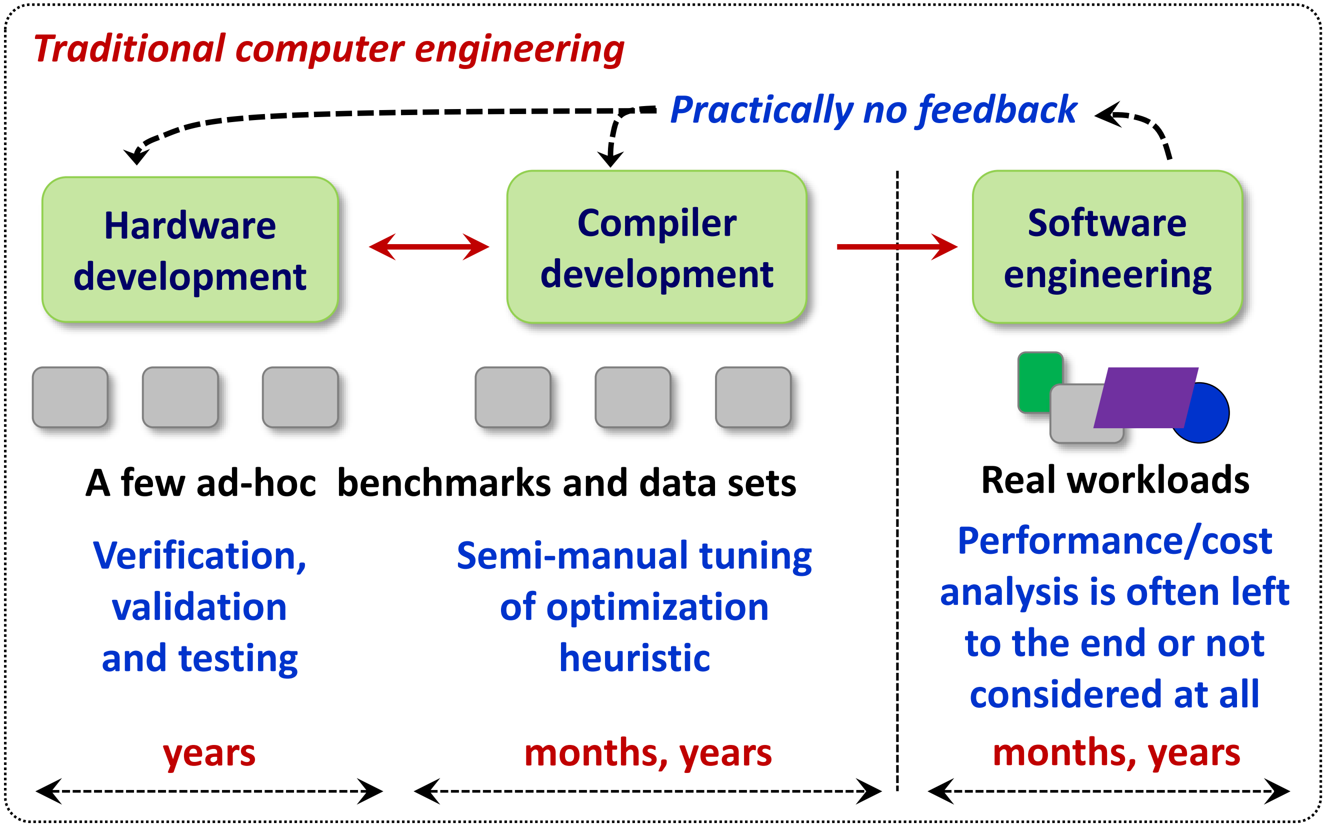
stalled because design and optimization was

- manual
- slow
- unreliable
- costly



AI / ML use cases

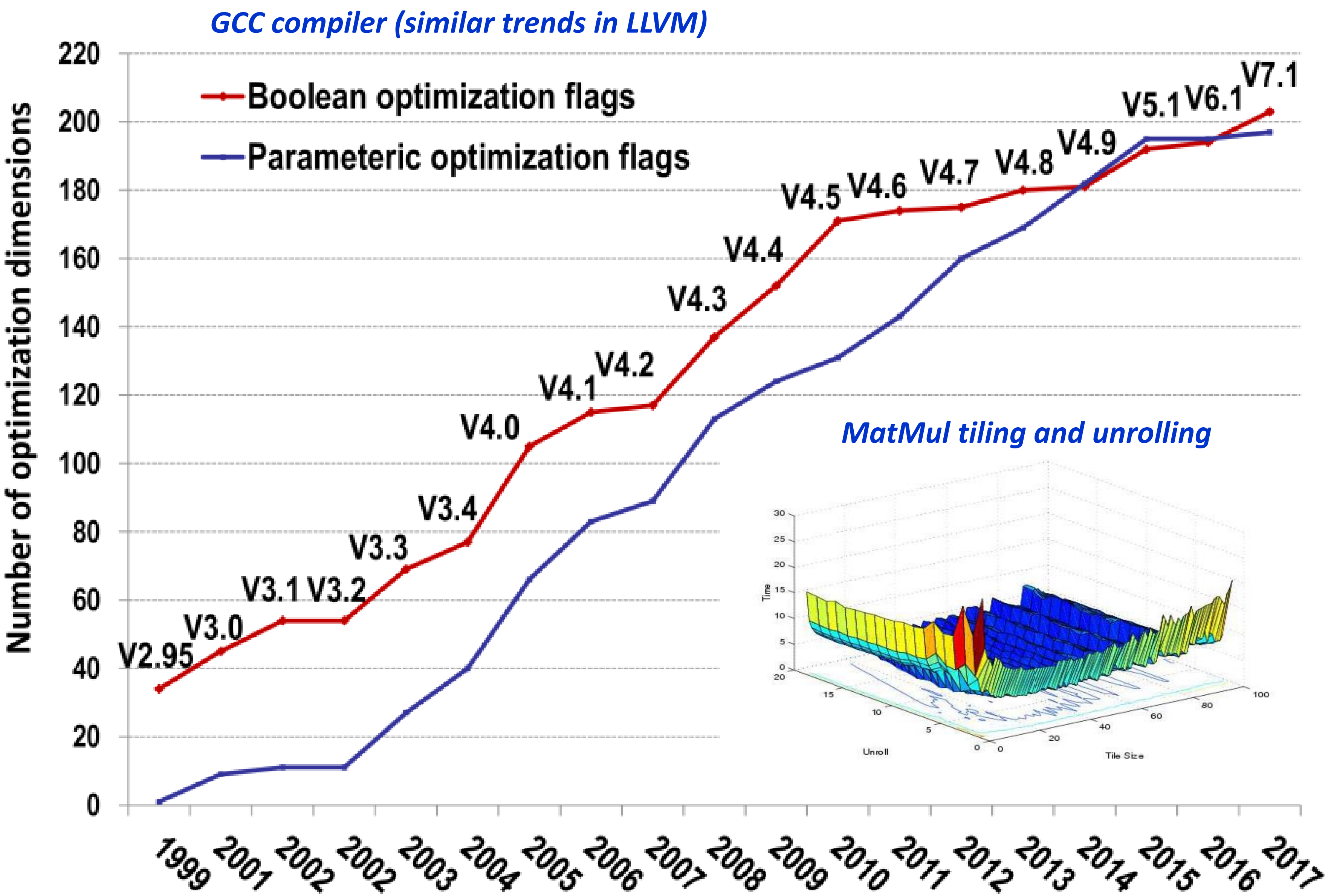
The methodology to design computer systems has hardly changed



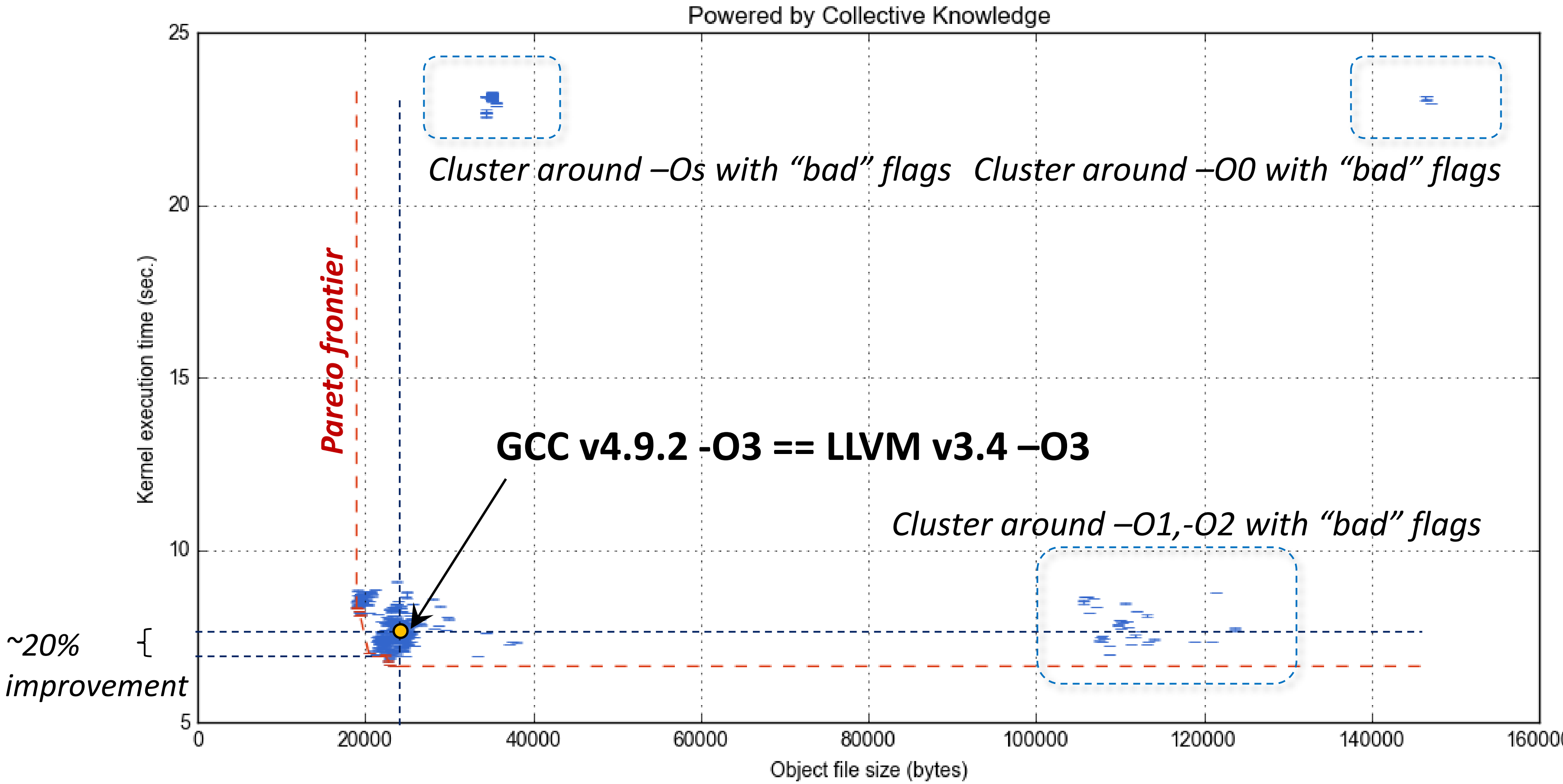
Challenges:

- Engineers and researchers waste their time on repetitive and ad-hoc tasks rather than innovation
- Simply no time to validate on many benchmarks, data sets, compiler/OS versions, hardware
- Increasing time to market for new products
- Waste of expensive resources
- Lowering ROI

Designing efficient computer systems is tedious, time consuming and costly due to large, multi-dimensional and non-linear optimization spaces



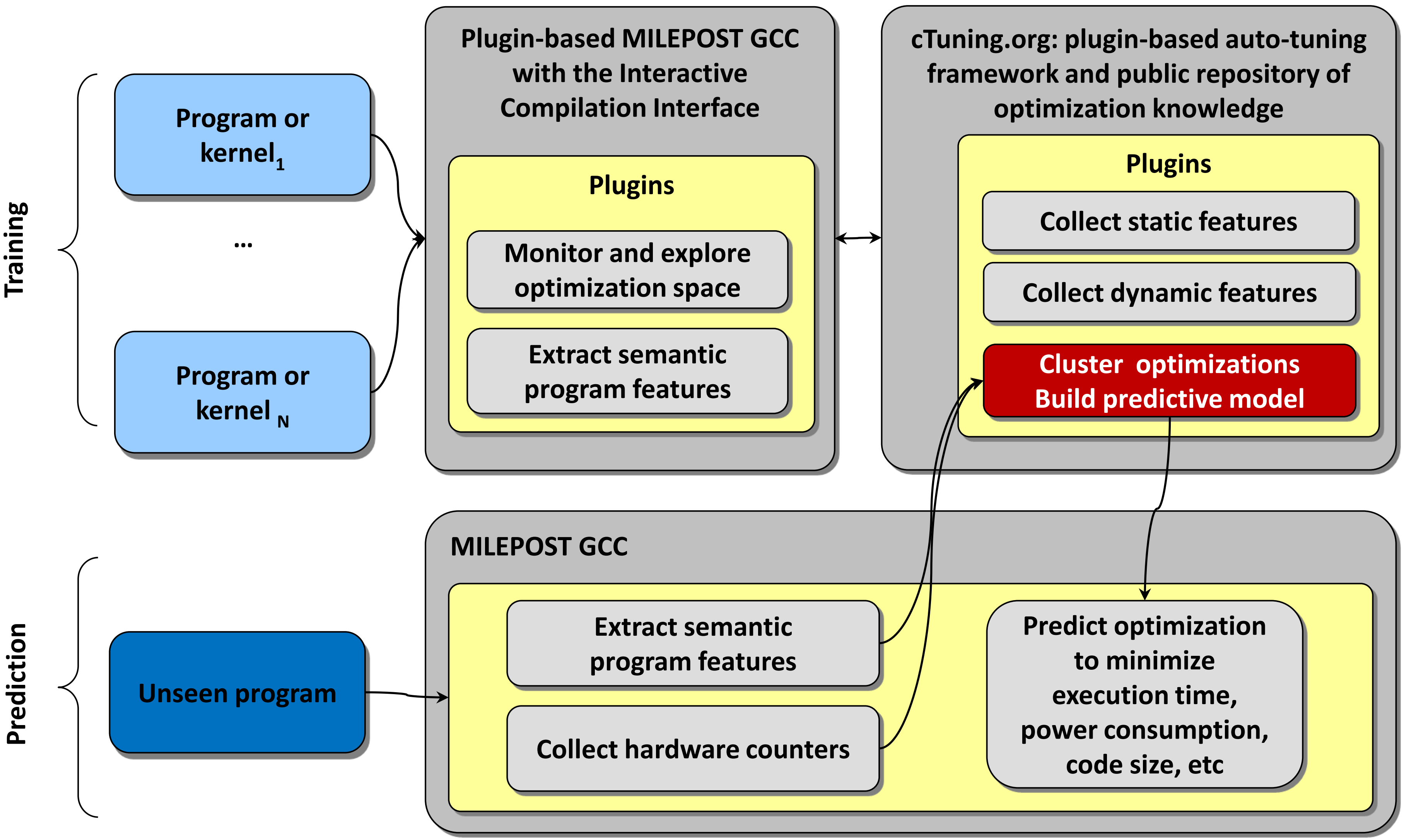
Compiler flag autotuning: too slow; lack of automation tools



Program:	*image corner detection*	Processor:	*ARM v7 (Cortex A15), 2.0GHz*
Compiler:	*GCC for ARM v4.9.2*	OS:	*Ubuntu 14.04.02 LTS*
System:	*ODROID-XU3*	Data set:	*MiDataSet #1, image, 600x450x8b PGM, 263KB*

*500 combinations of random flags -Ox -f**(no-)**FLAG*

MILEPOST project and cTuning.org (2006-2009): combining autotuning, machine learning and distributed training to predict optimization



Program characterization based on static features

MILEPOST GCC feature extractor (IBM Haifa & INRIA)

ft1 - Number of basic blocks in the method

…

ft19 - Number of direct calls in the method
ft20 - Number of conditional branches in the method
ft21 - Number of assignment instructions in the method
ft22 - Number of binary integer operations in the method
ft23 - Number of binary floating point operations in the method
ft24 - Number of instructions in the method
ft25 - Average of number of instructions in basic blocks

…

ft29 - Number of basic blocks with phi nodes in the interval [0, 3]

…

ft54 - Number of local variables that are pointers in the method
ft55 - Number of static/extern variables that are pointers in the method

cTuning.org/wiki/index.php/CTools:MilepostGCC:StaticFeatures:MILEPOST_V2.1

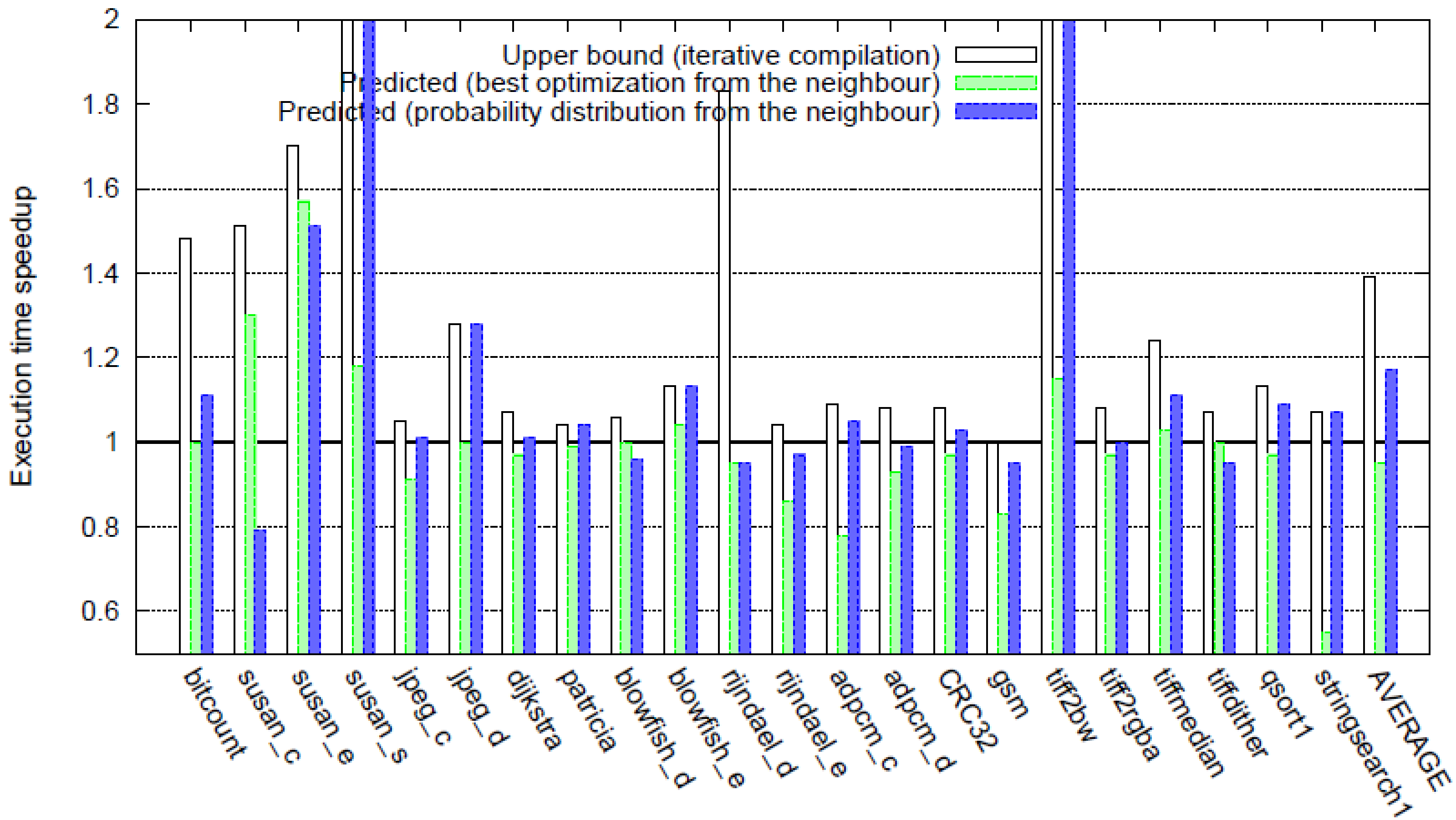
How constructed: human intuition

Critical to be able to make good predictions!

Can be auto-generated too:

"Practical aggregation of semantical program properties for machine learning based optimization"
Mircea Namolaru, Albert Cohen, Grigori Fursin, Ayal Zaks, Ari Freund CASES 2010

MILEPOST results: optimization prediction based on static features and the nearest-neighbors classification



Grigori Fursin et al. MILEPOST GCC: machine learning enabled self-tuning compiler.
International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

For dynamic features:
John Cavazos, Grigori Fursin, Felix V. Agakov, Edwin V. Bonilla, Michael F. P. O'Boyle, Olivier Temam.
Rapidly Selecting Good Compiler Optimizations using Performance Counters. CGO 2007 ***(CGO'17 test of time award)***

MILEPOST results: optimization prediction based on static features and the nearest-neighbors classification

Is everything solved now?

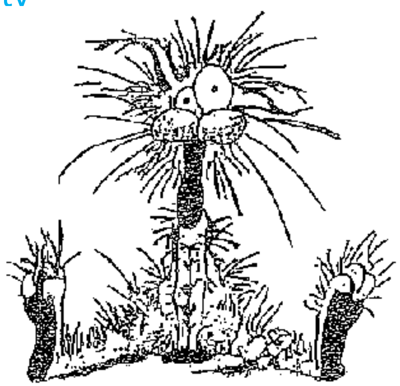
Grigori Fursin et al. MILEPOST GCC: machine learning enabled self-tuning compiler.
International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

For dynamic features:
John Cavazos, Grigori Fursin, Felix V. Agakov, Edwin V. Bonilla, Michael F. P. O'Boyle, Olivier Temam.
Rapidly Selecting Good Compiler Optimizations using Performance Counters. CGO 2007 ***(CGO'17 test of time award)***

Very challenging to make ML-based autotuning practical and use in production

Technological chaos

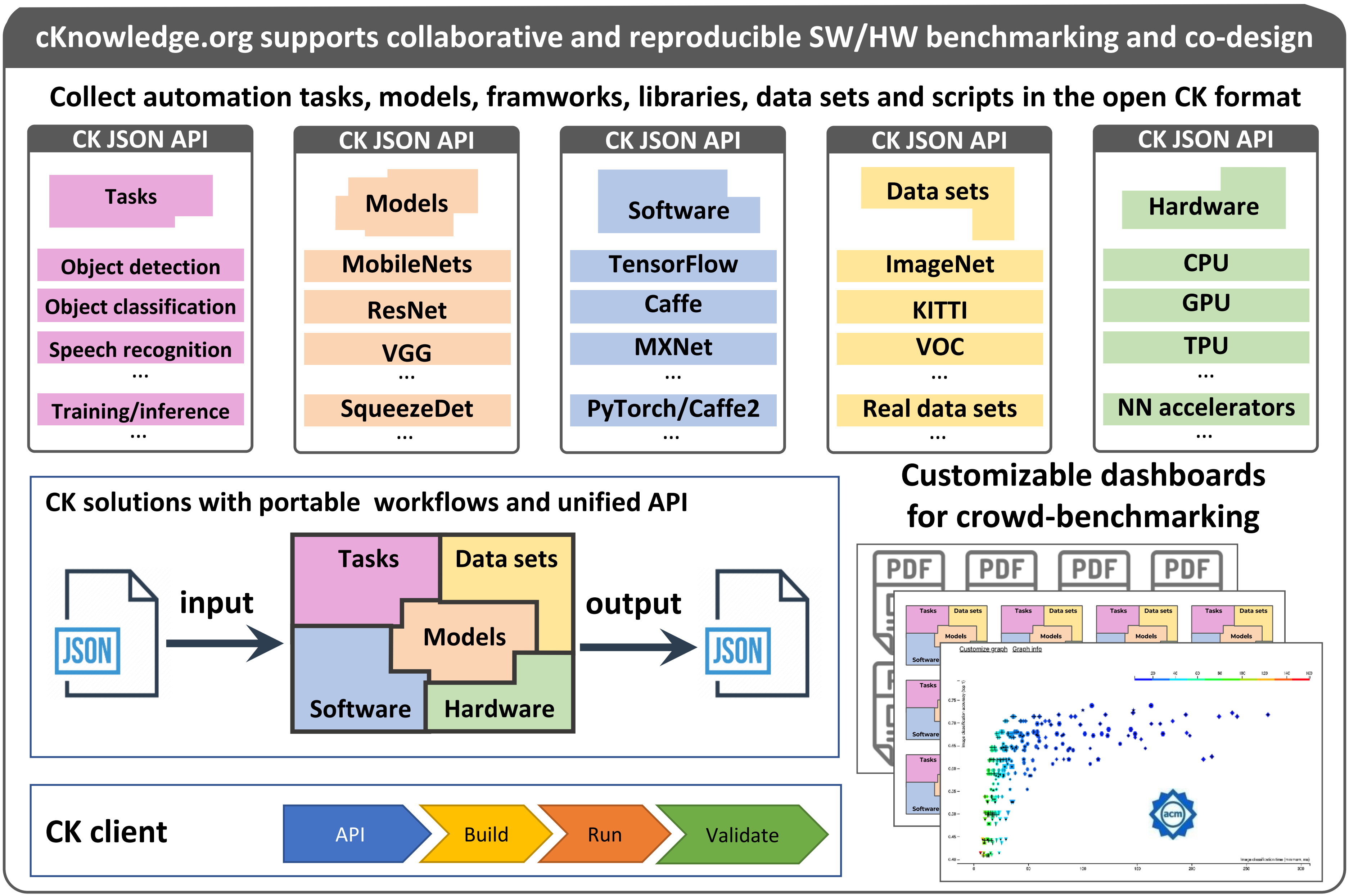
LLVM 3.4 *AVX*
GCC 4.8.x GCC 4.1.x ATLAS
frequency LLVM 2.8 hardware perf
genetic CUDA 5.x function counters
algorithms ICC 11.1 level pass
reordering
OpenMP *ARM v8* MPI OpenCL
prof gprof
reliability CUDA 4.x *Intel SandyBridge*
ARM v
se
ation



GCC 4.5 emory size
XLC
executi
KNN
Scalasca
C 11.0
MKL 4.6.x
LLVM 2.9 SimpleScalar Codelet PAPI
ICC 12.1 Jikes
process Testarossa Amplifier
Open64
oprofile HDD size
TLB loop-level ISA
GCC 4.7.x LLVM 2.7 SVM VTune
threads LTO ICC 10.1 threads
Phoenix program- algorithm-
TAU
cache size level level IPA TBB
algorithm precision

- Software/hardware/models and datasets are changing all the time
- Difficult to reproduce results collected from users (including variability of performance data and constant changes in the system)
- Difficult to expose choices, observe behavior and extract features (tools are not prepared for auto-tuning and machine learning)
- Difficult to share experimental setups (many SW/HW dependencies) including code, data and their features
- Difficult to collect huge, heterogeneous and continuously changing data in MySQL
- **Can't compare ML models and results from different papers – never enough info to reproduce results☹**

cKnowledge.org & cTuning.org/ae
enabling collaborative, reproducible and reusable ML&systems R&D



Universal complexity (dimension) reduction

Found solution

-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phiprop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vect-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web

Not very useful for analysis

- **2015: Collective Mind, Part II: Towards Performance- and Cost-Aware Software Engineering as a Natural Science**
  arxiv.org/abs/1506.06256
- **2018: A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques**
  cKnowledge.io/report/rpi3-crowd-tuning-2017-interactive (2018)

Universal complexity (dimension) reduction

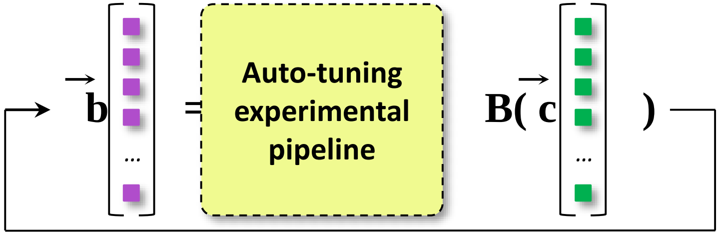
Found solution

-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phiprop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vect-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web

Chain complexity reduction filter

remove dimensions (or set to default)

iteratively, ANOVA, PCA, etc…



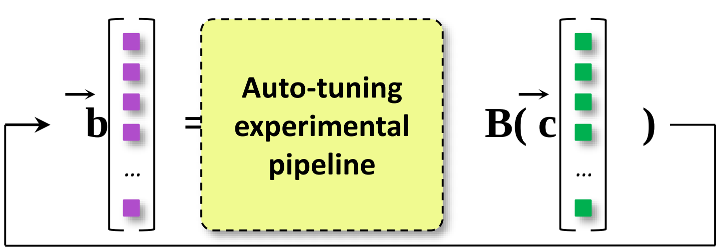
Universal complexity (dimension) reduction

Found solution

-O3 -fno-align-functions -fno-align-jumps -fno-align-labels -fno-align-loops -fno-asynchronous-unwind-tables -fno-branch-count-reg -fno-branch-target-load-optimize2 -fno-btr-bb-exclusive -fno-caller-saves -fno-combine-stack-adjustments -fno-common -fno-compare-elim -fno-conserve-stack -fno-cprop-registers -fno-crossjumping -fno-cse-follow-jumps -fno-cx-limited-range -fdce -fno-defer-pop -fno-delete-null-pointer-checks -fno-devirtualize -fno-dse -fno-early-inlining -fno-expensive-optimizations -fno-forward-propagate -fgcse -fno-gcse-after-reload -fno-gcse-las -fno-gcse-lm -fno-gcse-sm -fno-graphite-identity -fguess-branch-probability -fno-if-conversion -fno-if-conversion2 -fno-inline-functions -fno-inline-functions-called-once -fno-inline-small-functions -fno-ipa-cp -fno-ipa-cp-clone -fno-ipa-matrix-reorg -fno-ipa-profile -fno-ipa-pta -fno-ipa-pure-const -fno-ipa-reference -fno-ipa-sra -fno-ivopts -fno-jump-tables -fno-math-errno -fno-loop-block -fno-loop-flatten -fno-loop-interchange -fno-loop-parallelize-all -fno-loop-strip-mine -fno-merge-constants -fno-modulo-sched -fmove-loop-invariants -fomit-frame-pointer -fno-optimize-register-move -fno-optimize-sibling-calls -fno-peel-loops -fno-peephole -fno-peephole2 -fno-predictive-commoning -fno-prefetch-loop-arrays -fno-regmove -fno-rename-registers -fno-reorder-blocks -fno-reorder-blocks-and-partition -fno-reorder-functions -fno-rerun-cse-after-loop -fno-reschedule-modulo-scheduled-loops -fno-sched-critical-path-heuristic -fno-sched-dep-count-heuristic -fno-sched-group-heuristic -fno-sched-interblock -fno-sched-last-insn-heuristic -fno-sched-pressure -fno-sched-rank-heuristic -fno-sched-spec -fno-sched-spec-insn-heuristic -fno-sched-spec-load -fno-sched-spec-load-dangerous -fno-sched-stalled-insns -fno-sched-stalled-insns-dep -fno-sched2-use-superblocks -fno-schedule-insns -fno-schedule-insns2 -fno-short-enums -fno-signed-zeros -fno-sel-sched-pipelining -fno-sel-sched-pipelining-outer-loops -fno-sel-sched-reschedule-pipelined -fno-selective-scheduling -fno-selective-scheduling2 -fno-signaling-nans -fno-single-precision-constant -fno-split-ivs-in-unroller -fno-split-wide-types -fno-strict-aliasing -fno-thread-jumps -fno-trapping-math -fno-tree-bit-ccp -fno-tree-builtin-call-dce -fno-tree-ccp -fno-tree-ch -fno-tree-copy-prop -fno-tree-copyrename -fno-tree-cselim -fno-tree-dce -fno-tree-dominator-opts -fno-tree-dse -ftree-forwprop -fno-tree-fre -fno-tree-loop-distribute-patterns -fno-tree-loop-distribution -fno-tree-loop-if-convert -fno-tree-loop-if-convert-stores -fno-tree-loop-im -fno-tree-loop-ivcanon -fno-tree-loop-optimize -fno-tree-lrs -fno-tree-phiprop -fno-tree-pre -fno-tree-pta -fno-tree-reassoc -fno-tree-scev-cprop -fno-tree-sink -fno-tree-slp-vectorize -fno-tree-sra -fno-tree-switch-conversion -ftree-ter -fno-tree-vect-loop-version -fno-tree-vectorize -fno-tree-vrp -fno-unroll-all-loops -fno-unsafe-loop-optimizations -fno-unsafe-math-optimizations -funswitch-loops -fno-variable-expansion-in-unroller -fno-vect-cost-model -fno-web

Chain complexity reduction filter

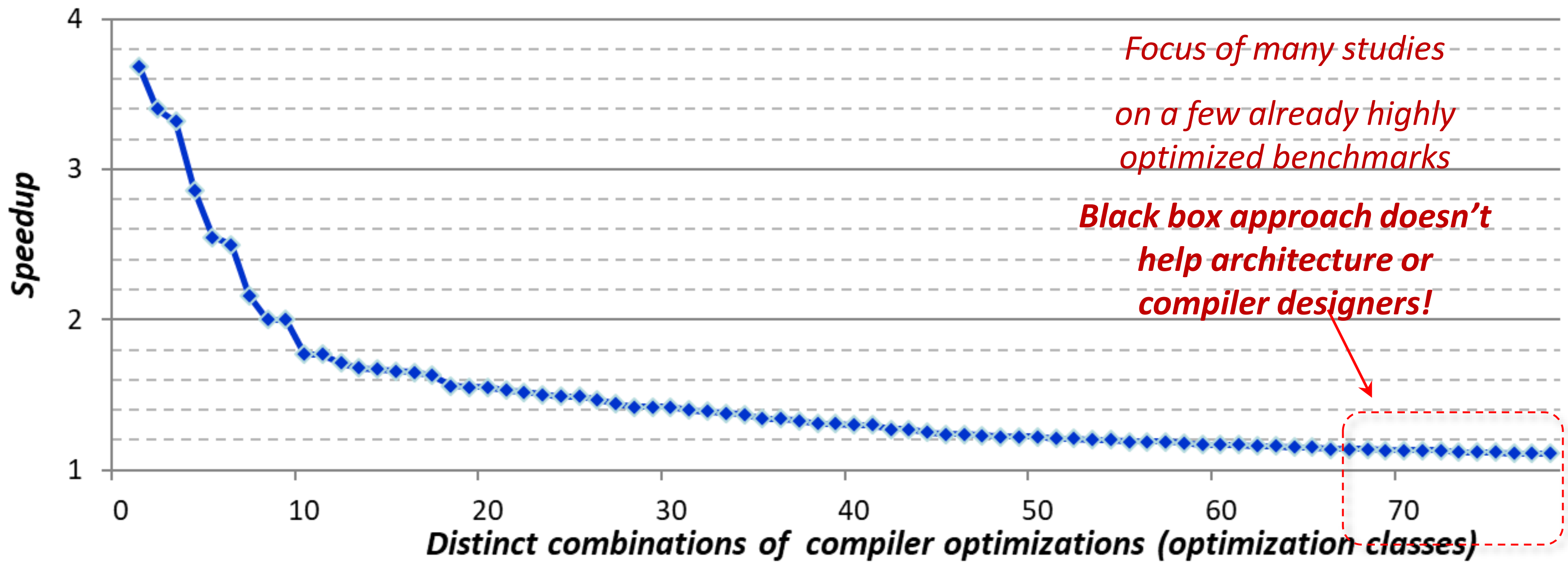
remove dimensions (or set to default)
iteratively, ANOVA, PCA, etc…



Pruned solution

-O3
-fno-align-functions ***(15% of speedup)***
-fdce
-fgcse
-fguess-branch-probability ***(70% of speedup)***
-fmove-loop-invariants
-fomit-frame-pointer
-ftree-ter
-funswitch-loops
-fno-ALL

Open repository of optimization knowledge



Continuously crowdtuning 285 shared code and dataset combinations from 8 benchmarks including NAS, MiBench, SPEC2000, SPEC2006, Powerstone, UTDSP and SNU-RT

using GRID 5000; Intel E5520, 2.6MHz;

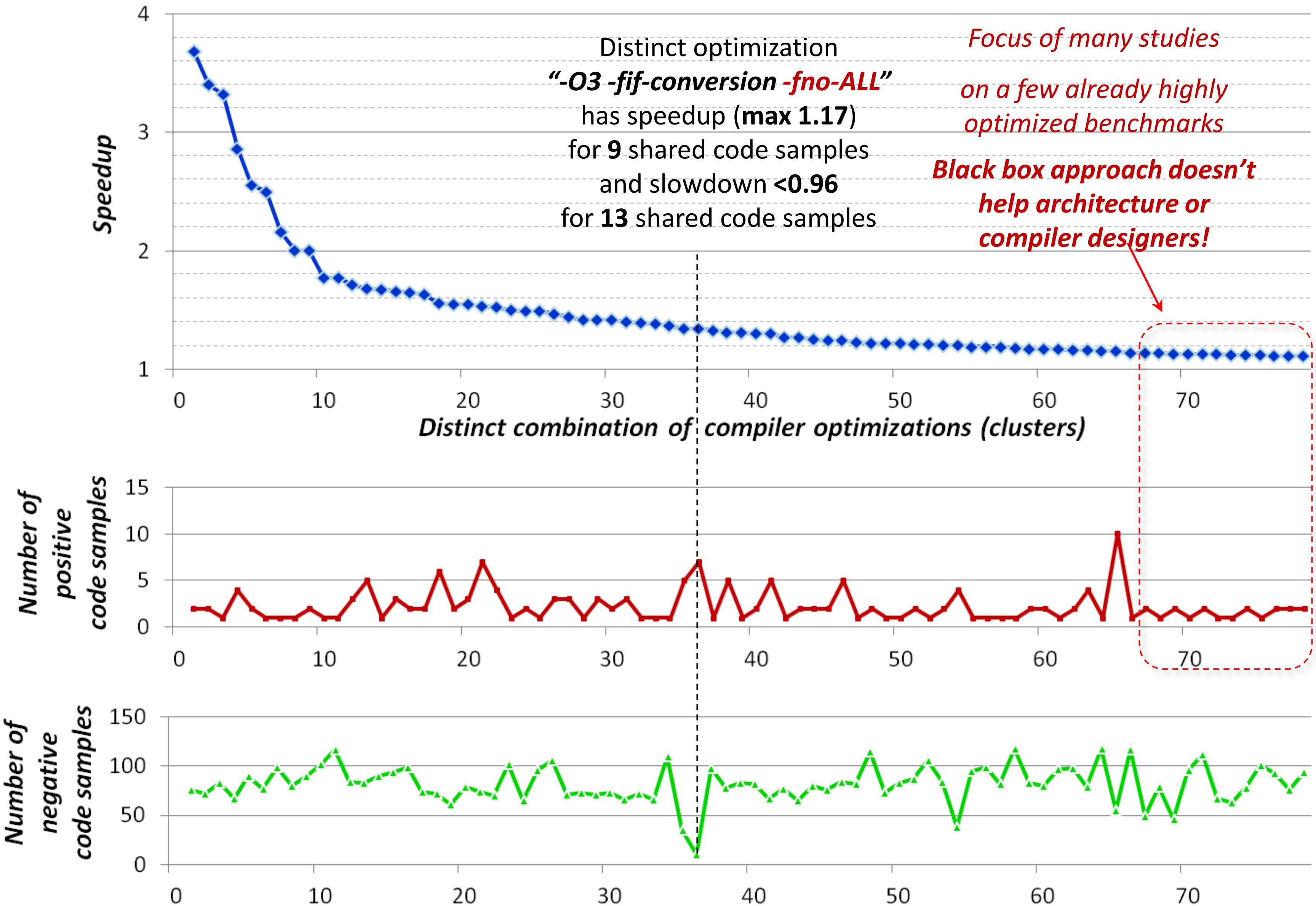
GCC 4.6.3; at least 5000 random combinations of flags

cKnowledge.org/gcc-crowd-benchmarking-results

cKnowledge.org/llvm-crowd-benchmarking-results

github.com/ctuning/reproduce-milepost-project

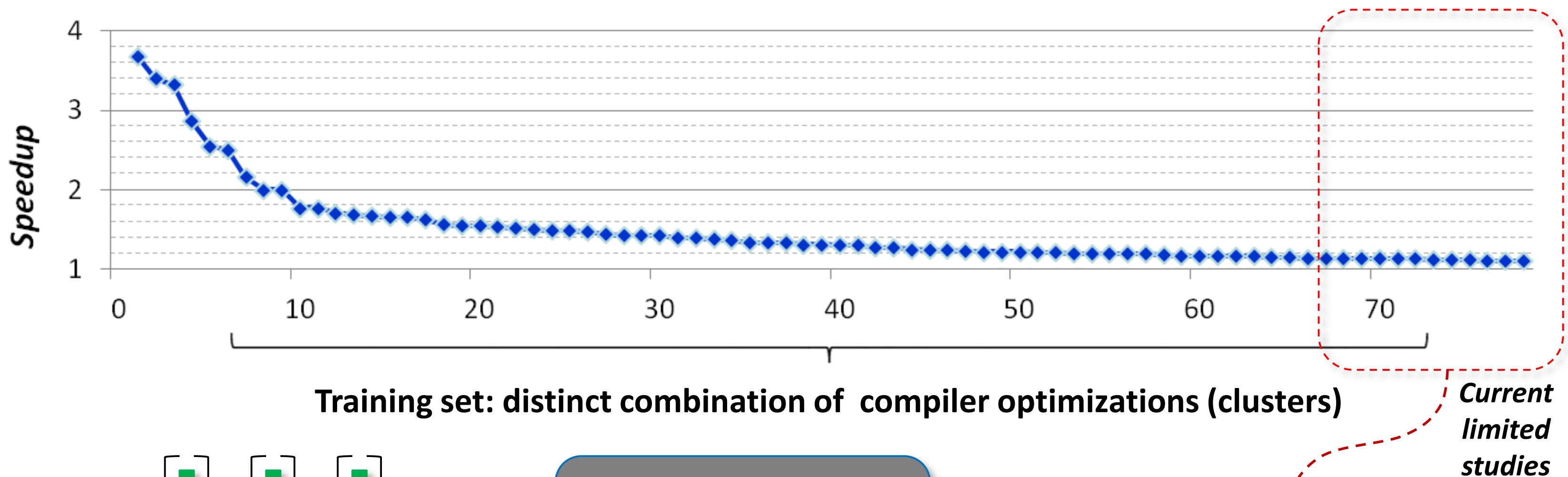
Universal and practical crowdtuning



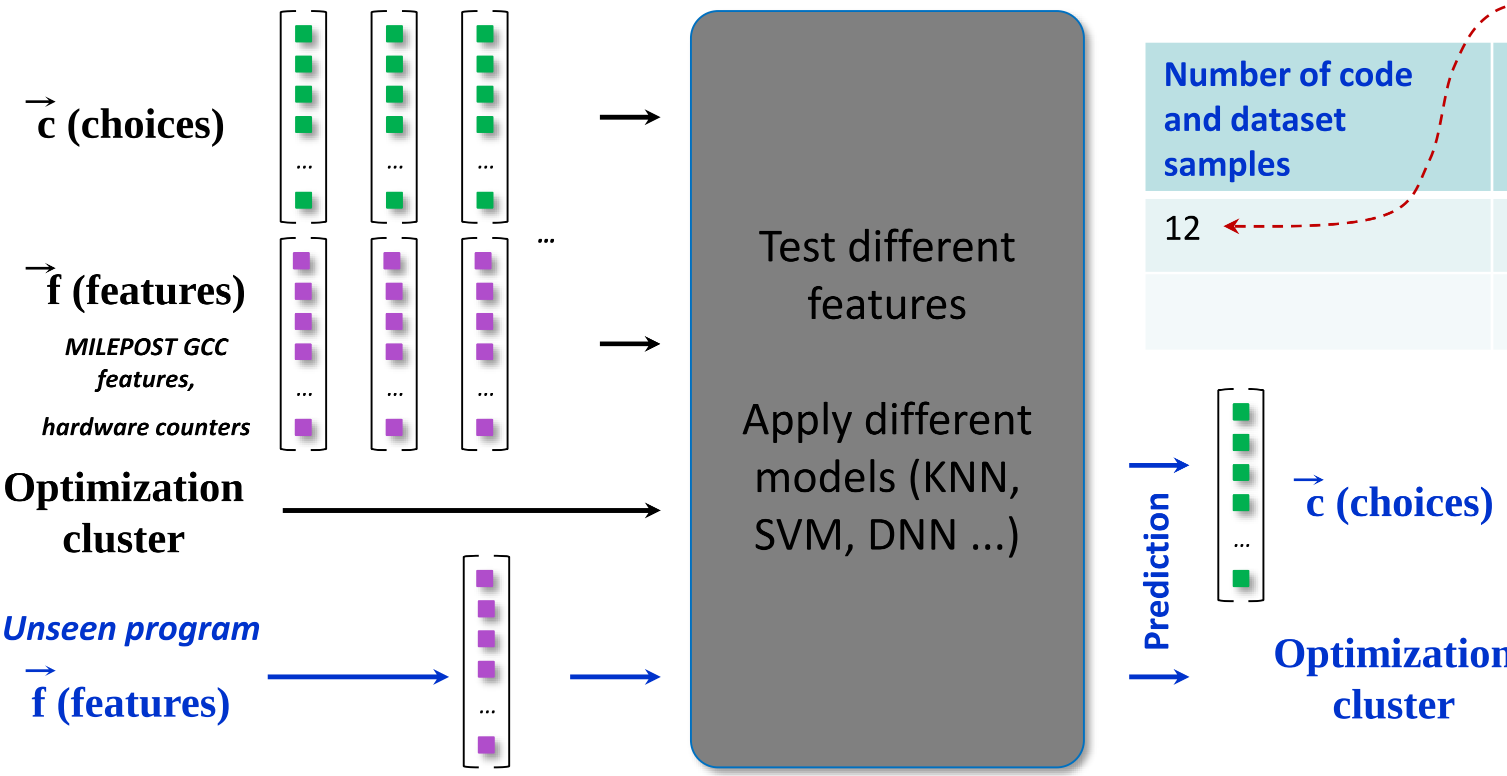
285 shared code and dataset samples; GRID 5000; Intel E5520, 2.6MHz;

GCC 4.6.3; at least 5000 random combinations of flags

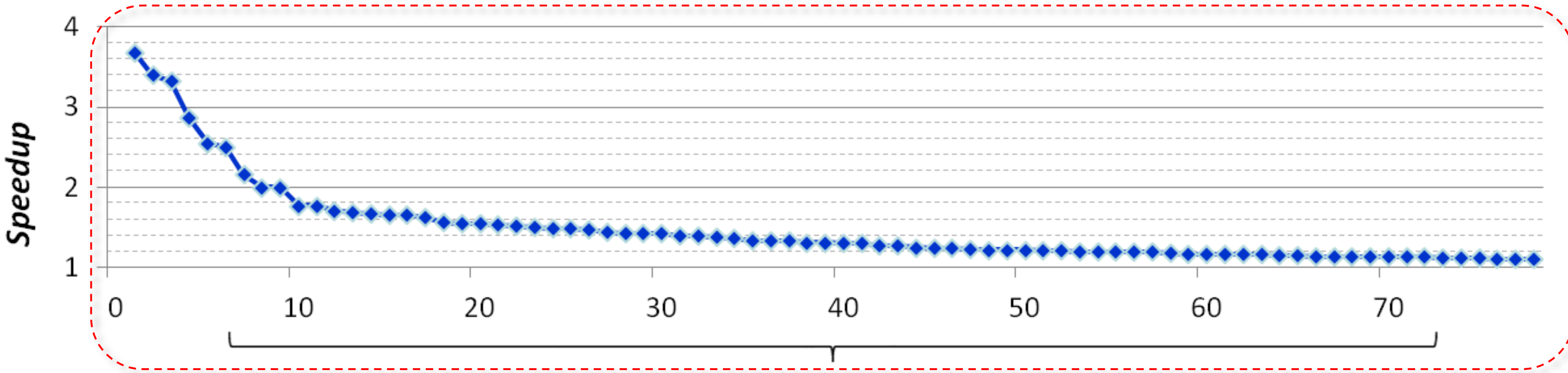
Automated training and learning



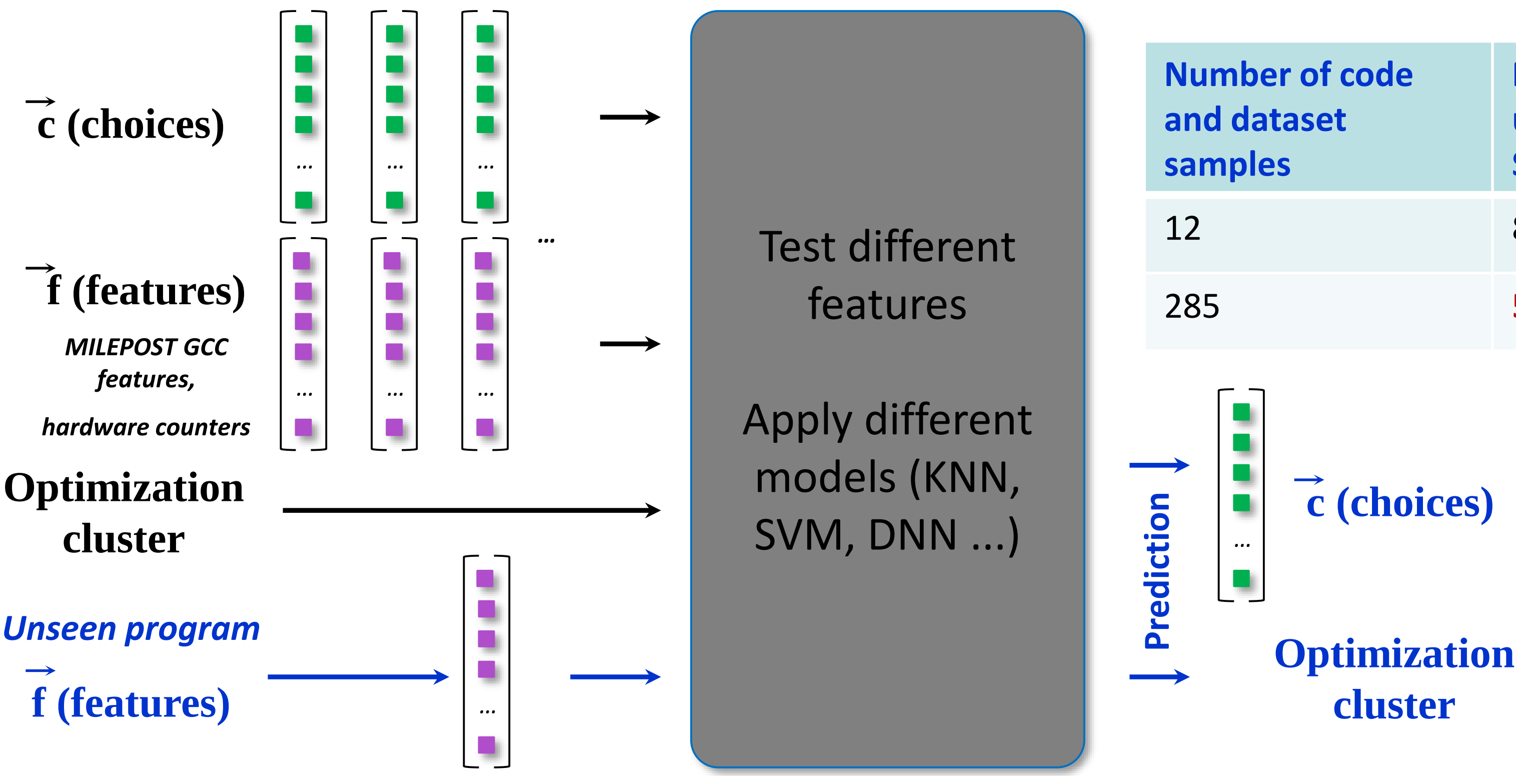
Number of code and dataset samples	Prediction accuracy using optimized SVM
12	87%



Automated training and learning



Training set: distinct combination of compiler optimizations (clusters)

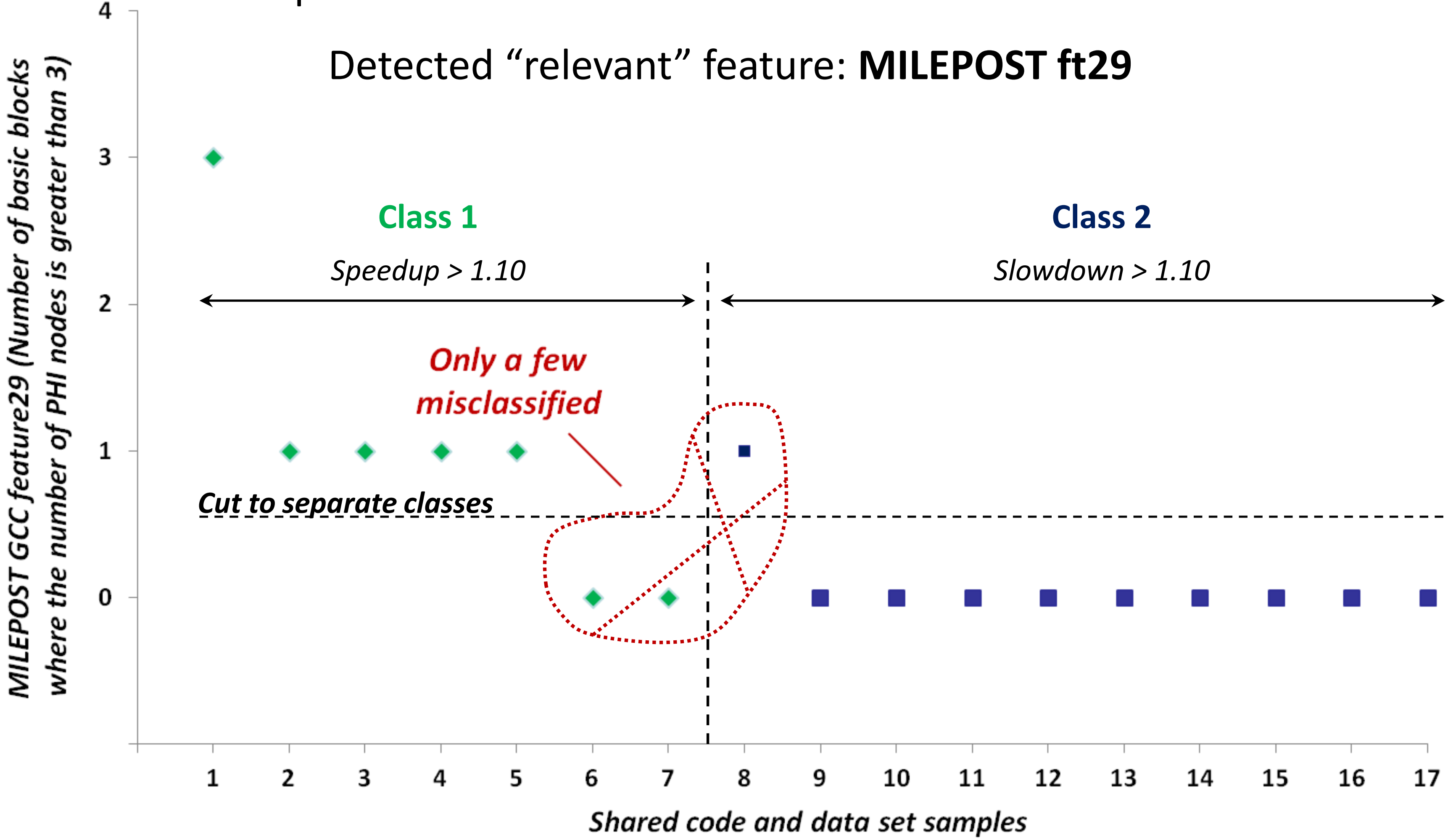


Number of code and dataset samples	Prediction accuracy using optimized SVM
12	87%
285	56%

Validating generated models and features

Applied complexity reduction to MILEPOST GCC features and hardware counters for optimization cluster **"-O3 -fno-if-conversion -fno-ALL"**

Detected "relevant" feature: **MILEPOST ft29**

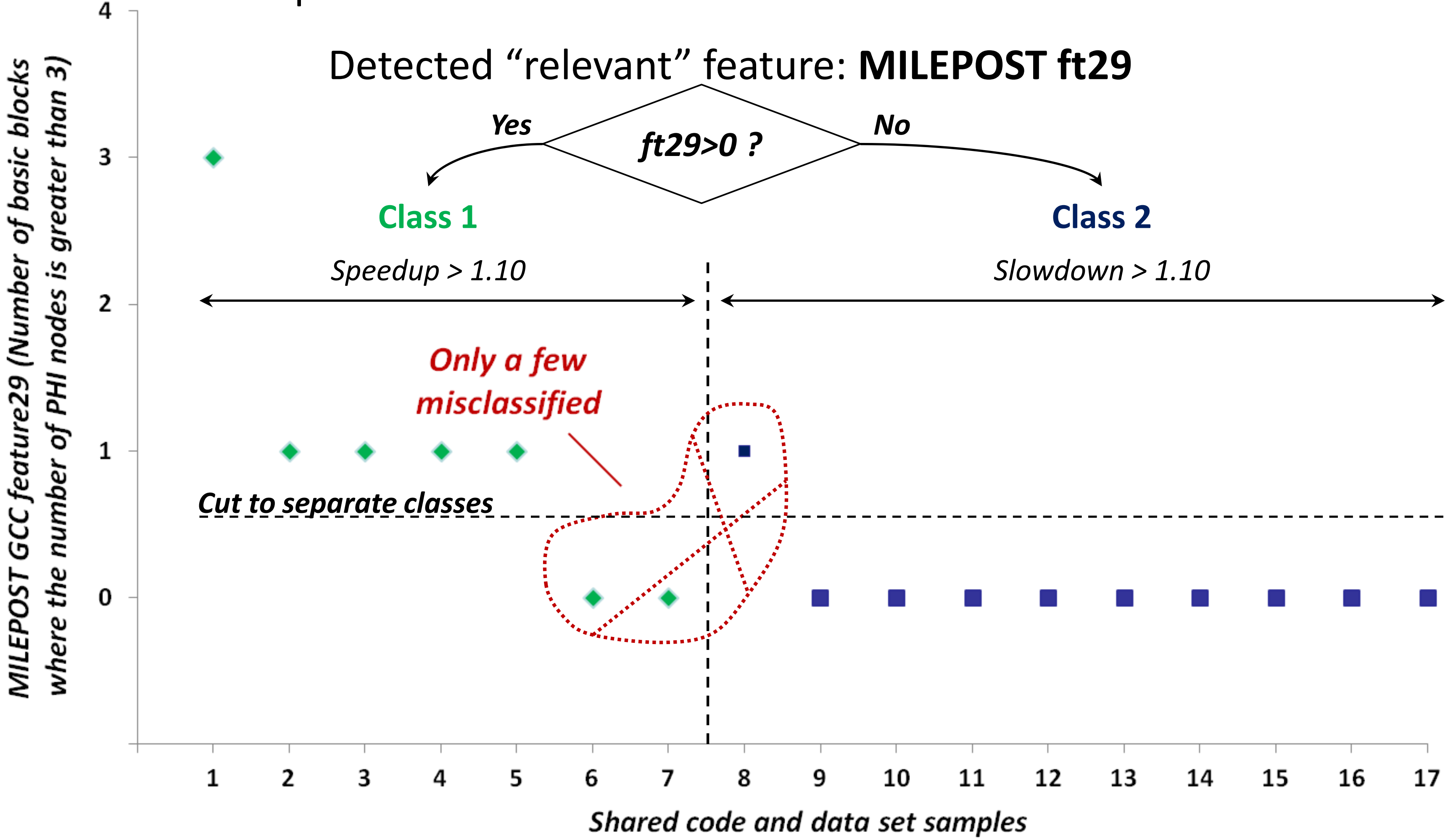


Validating generated models and features

Applied complexity reduction to MILEPOST GCC features and hardware counters

for optimization cluster **"-O3 -fno-if-conversion -fno-ALL"**

Detected "relevant" feature: **MILEPOST ft29**



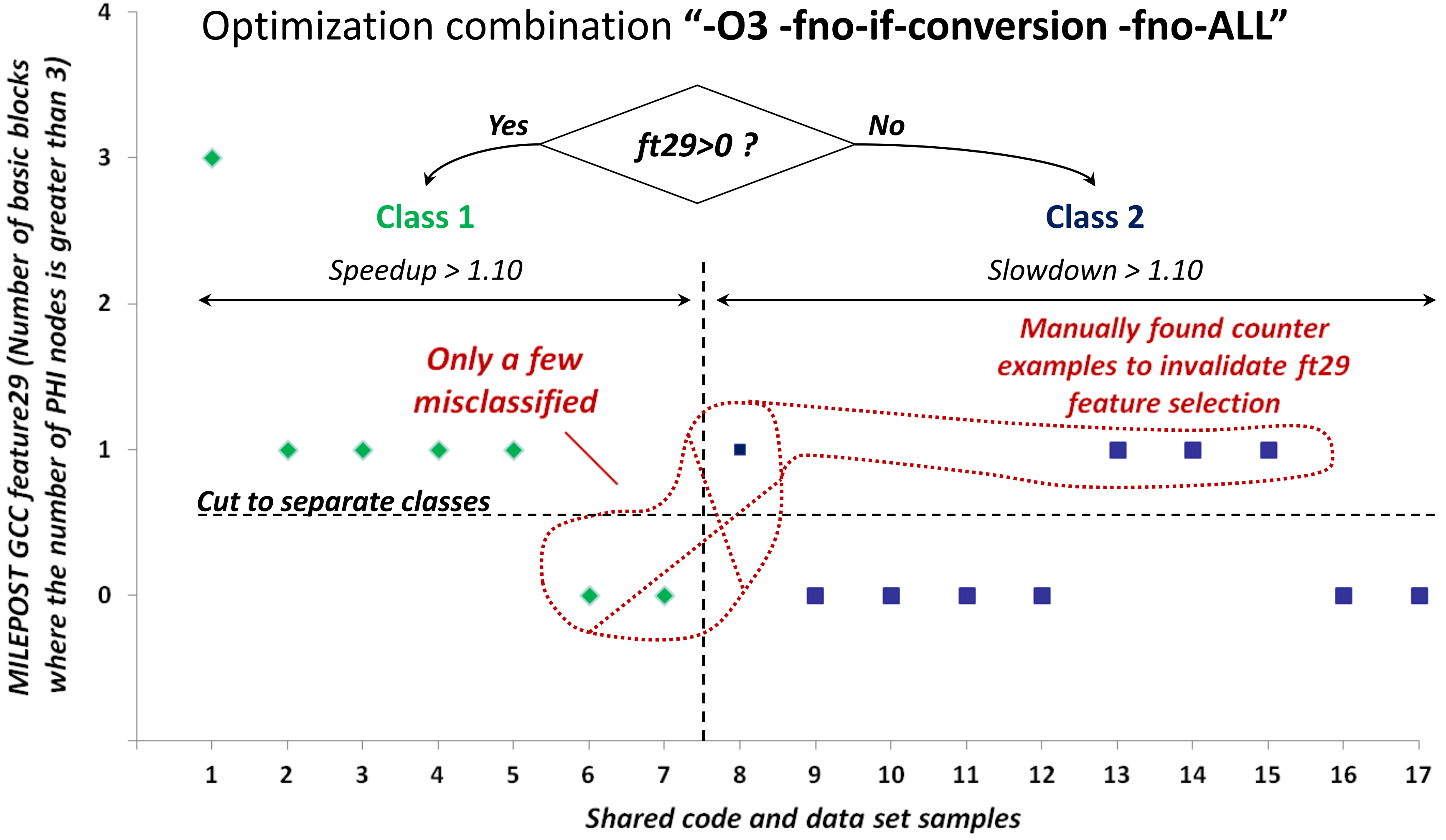
Most of the current paper finish at this stage saying that good model is found!

But does it make sense? We can expose it to domain specialists for validation

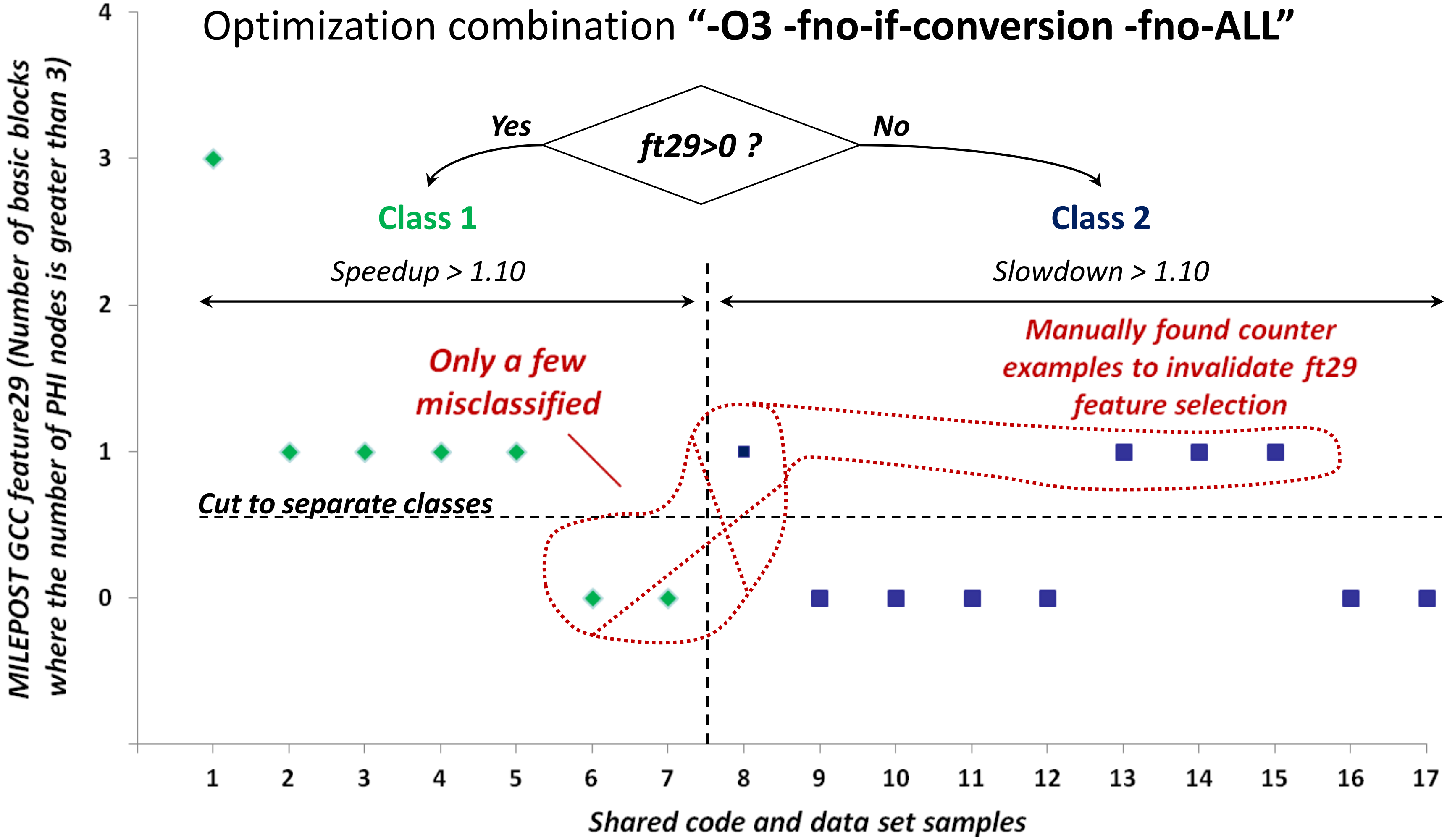
blocksort function for x264 video codec

```
...
volatile int sum, value = 3;
int sumA = 0;
int sumB = 0;
int sumC = 0;

for (j = ftab[ss << 8] & (~((1 << 21))) ; j < copyStart[ss] ; j++)
{
    k = ptr[j] - 1;
    sumA += value;
    sumB += value;
    sumC += value;

    if (k < 0)
        k += nblock;

    sum = sumA + sumB + sumC;
    c1 = block[k];
    if ( !bigDone[c1])
        ptr[copyStart[c1]++] = k;
}
```

Validating generated models and features



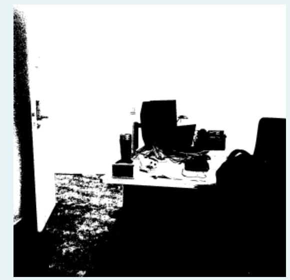
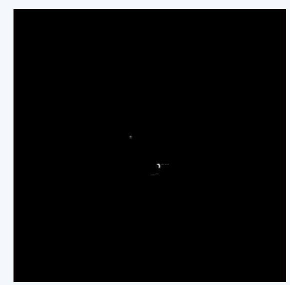
Validating generated models and features



Black box statistical machine learning can be very misleading particularly with small training sets - need community approach to share many benchmarks, codelets, data sets, models …

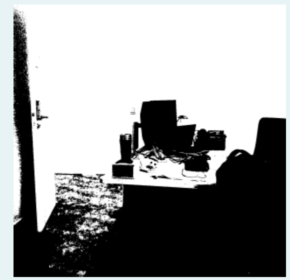
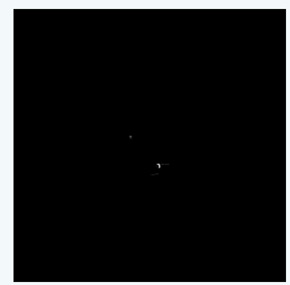
Learning features by domain specialists; taking data sets into account

Image B&W threshold filter **matrix_ptr2++ = (temp1 > T) ? 255 : 0;*

Class	-O3	-O3 -fno-if-conversion
Shared data set $sample_1$	***reference execution time*** 	no change 
Shared data set $sample_2$	no change 	***+17.3% improvement*** 

Learning features by domain specialists; taking data sets into account

Image B&W threshold filter **matrix_ptr2++ = (temp1 > T) ? 255 : 0;*

Class	-O3	-O3 -fno-if-conversion
Shared data set $sample_1$ *Monitored during **day***	***reference execution time*** 	no change 
Shared data set $sample_2$ *Monitored during **night***	no change 	***+17.3% improvement*** 

Feature "TIME_OF_THE_DAY" related to algorithm, data set and run-time
Can't be found by ML - simply does not exist in the system!

Need split-compilation (cloning and run-time adaptation)

```
if get_feature(TIME_OF_THE_DAY)==NIGHT      bw_filter_codelet_day(buffers);
else                                        bw_filter_codelet_night(buffers);
```

Conclusions

- Test your autotuning and ML techniques in real systems and real life conditions
- Perform full system optimization and co-design
- Rather than guessing new features and models based on tiny or synthetic training sets, try to build a very large and diverse training set and then look for better models and features
- Try to understand programs, optimizations, features and models rather than using black box approach
- Use open Kaggle-like competitions to find better optimizations, features and models

My dream: help researchers share portable workflows and reusable artifacts along with their research papers to let the community validate results, participate in crowd-tuning while trying different software, hardware, models and data sets, and aggregate all optimization knowledge in an open repository!

cKnowledge.io platform and Artifact Evaluation (cTuning.org/ae): we are only the beginning of the long journey and there is still a lot to be done!

The latest MILEPOST results in the live paper: cKnowledge.org/rpi-crowd-tuning
My vision paper about the Collective Knowledge project: arxiv.org/abs/2006.07161

cKnowledge.org   cKnowledge.io/reproduced-results   cKnowledge.io/reproduced-papers