Collective Knowledge Technology

From ad hoc computer engineering
to collaborative and reproducible data science

github.com/ctuning/ck

Grigori Fursin
CSO, non-profit cTuning foundation, France
CTO, dividiti, UK

The University of Manchester
November 2015

Computer systems can be very inefficient, power hungry and unreliable

Require tedious, ad-hoc, semi-automatic tuning and run-time adaptation

Face recognition
using mobile phones

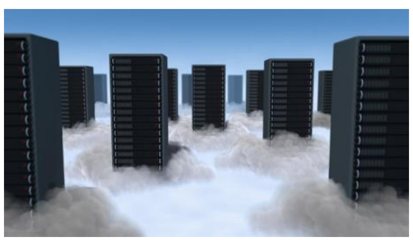
→

→

OpenCL-based algorithm

7x speedup, 5x energy savings,
but poor accuracy

2x speedup without
sacrificing accuracy –
enough to enable RT processing



Weather prediction in
supercomputer centers

→

→

MPI-based program

5% speed up
with the same accuracy

dramatic savings
in energy bill per year

What do we do wrong?

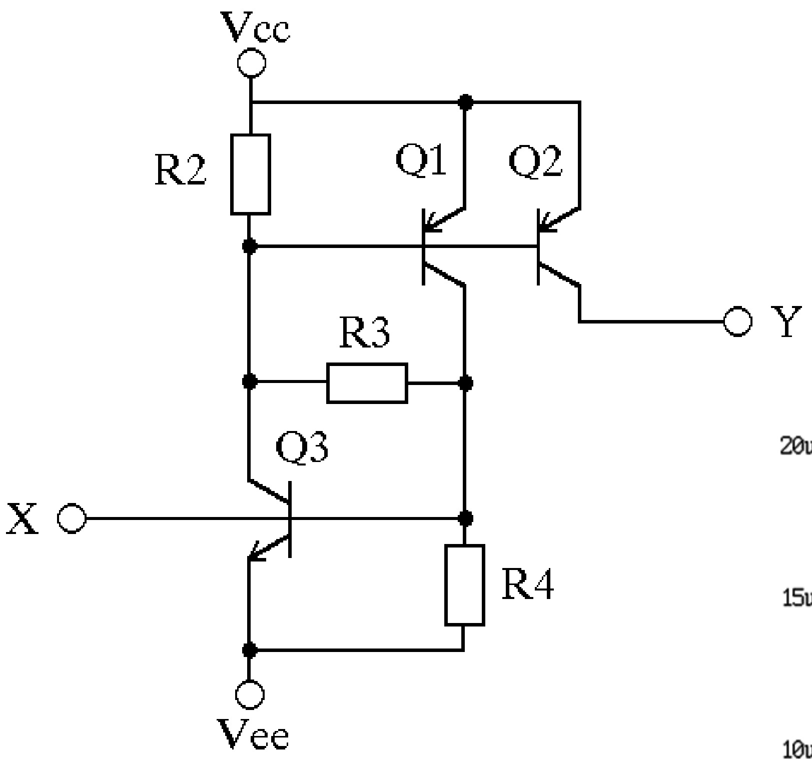
How can we reproduce such results and build upon them?

We can take advantage of powerful data science methods?

Talk outline

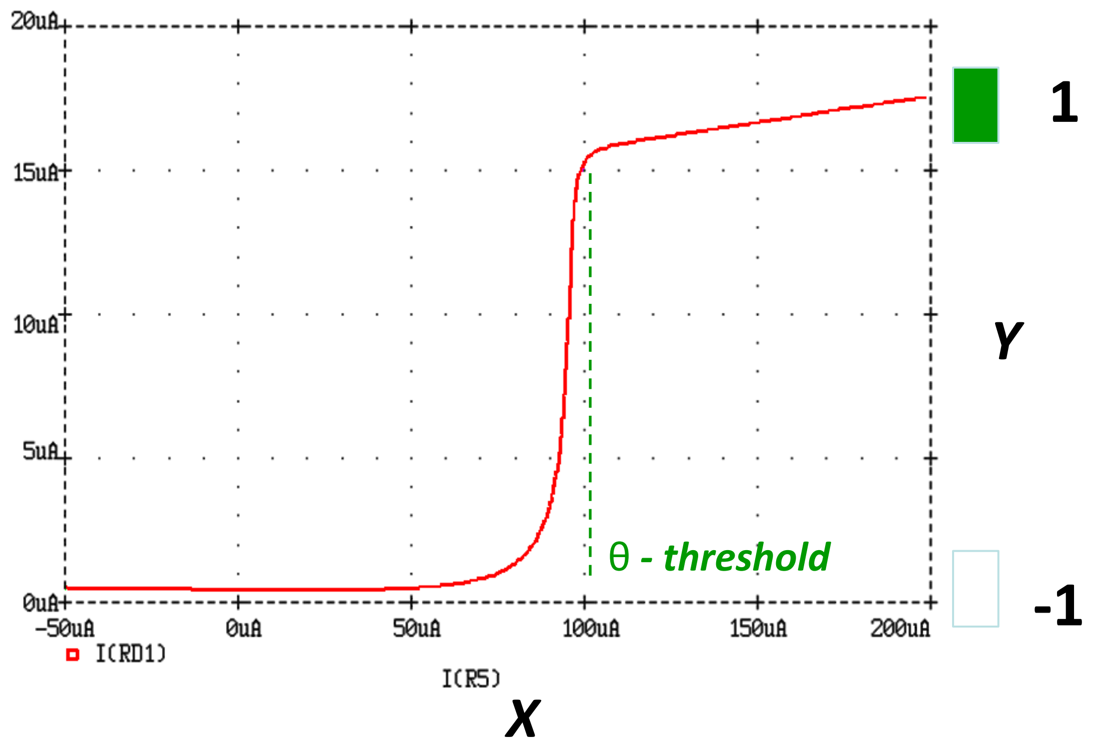
- Major problems in computer engineering
- Our community-driven solution: Collective Knowledge Framework and Repository
- Solving old problems with our approach (crowdsourcing autotuning and learning)
    - *Practical compiler heuristic tuning via machine learning*
    - *Avoiding common pitfalls in machine learning based tuning*
    - *Feature selection and model improvement by domain specialists*
    - *ML-based run-time adaptation and predictive scheduling*
- Our open research initiatives for major conferences (CGO/PPoPP)
- Conclusions, future work and possible collaboration

All techniques were validated in industrial projects
with IBM, ARC, Intel, STMicroelectronics and ARM



Semiconductor neuron

My first R&D project (1993-1996) developing neural accelerators for brain-inspired computers



Failed because modeling was

Too slow

Unreliable

Costly

and we didn't have GPGPUs

Spent last 15 years searching for practical solutions

1999-2004: PhD in computer science, University of Edinburgh, UK

Prepared foundation for machine-learning based performance autotuning

2007-2010: Tenured research scientist at INRIA, France
Adjunct professor at Paris South University, France
Developed self-tuning compiler GCC combined with machine learning via cTuning.org –public optimization knowledge repository

2010-2011: Head of application optimization group at Intel Exascale Lab, France
Software/Hardware co-design and adaptation using machine learning

2012-2014: Senior tenured research scientist, INRIA, France
Collective Mind Project – platform to share artifacts and crowdsrouce experiments in computer engineering

Developed methodology for performance and cost-aware computer engineering

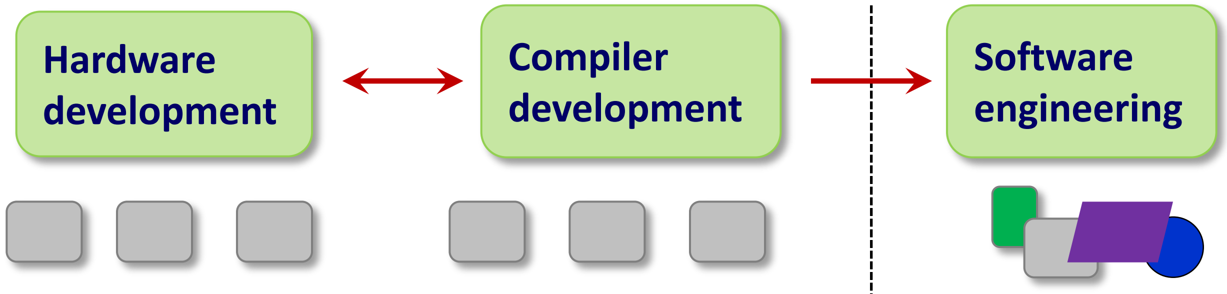
2015-now: CTO, dividiti, UK
Collective Knowledge Project – python-based framework and repository for collaborative and reproducible experimentation in computer engineering combined with predictive analytics – bringing all the missing pieces of the puzzle together

Close collaboration with ARM, IBM, Intel, ARC, STMicroelectronics

Presented work and opinions are my own!

Motivation and challenges

Traditional computer engineering



A few ad-hoc benchmarks and data sets

Verification, validation and testing

Semi-manual tuning of optimization heuristic

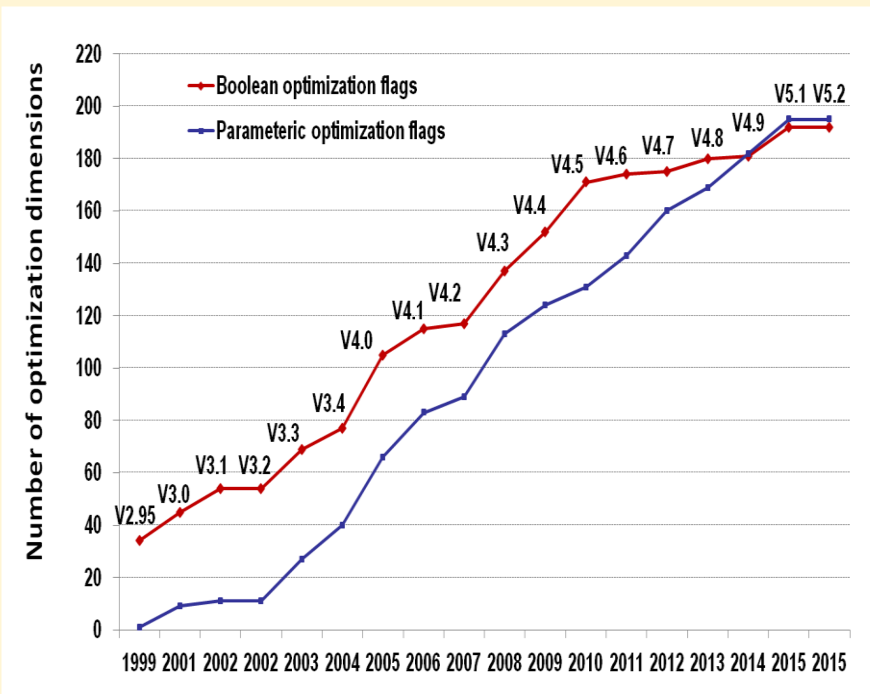
Real software

Performance/cost analysis is often left to the end or not considered at all

Motivation and challenges

Traditional computer engineering

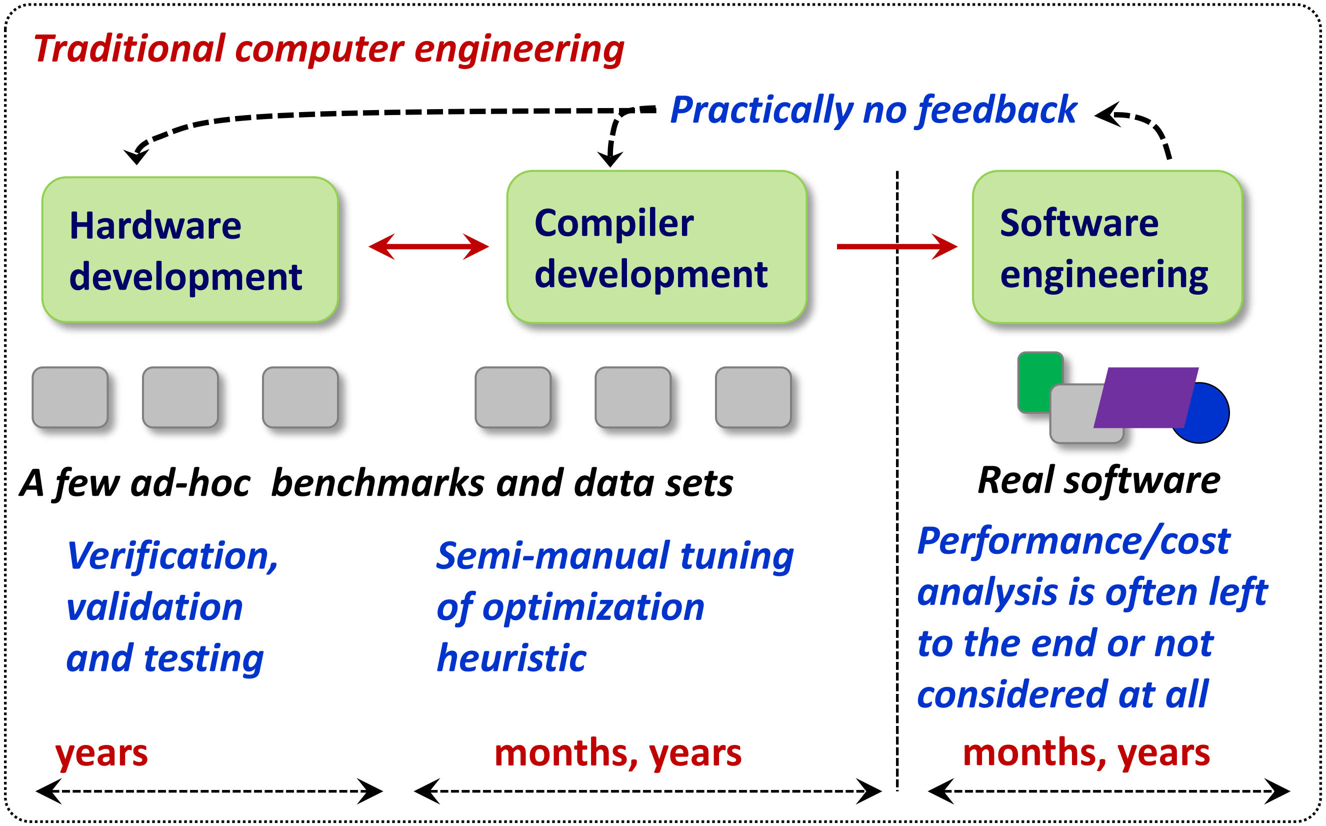
Hardware development ↔ Compiler development → Software engineering



Well-known fundamental problems:

1) Too many design and optimization choices at all levels
2) Multi-objective optimization: performance vs compilation time vs code size vs system size vs power consumption vs reliability vs ROI
3) Complex relationship and interactions between SW/HW components

Motivation and challenges



Machine-learning based autotuning, dynamic adaptation, co-design: high potential for more than 2 decades but still far from production use!

- Lack of representative benchmarks and data sets for training
- Tuning and training is still very long – no optimization knowledge reuse
- Black box model doesn’t help architecture or compiler designers
- No common experimental methodology - many statistical pitfalls and wrong usages of machine learning

MILEPOST project (2006-2009): crowdsourcing iterative compilation (cTuning.org)?

Traditional computer engineering

Practically no feedback

Hardware development ↔ **Compiler development** → **Software engineering**

A few ad-hoc benchmarks and data sets

Real software

Verification, validation and testing

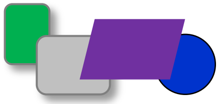
Semi-manual tuning of optimization heuristic

Performance/cost analysis is often left to the end or not considered at all

years

months, years

months, years



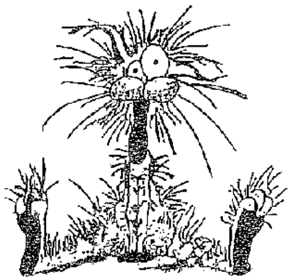
continuous feedback how to improve hardware and any software including compilers

- cTuning.org repository of optimization knowledge with shared benchmarks and data sets
- Distributed performance and cost tracking and tuning
- Machine learning to predict optimizations
- Interdisciplinary community to improve models

Faced more problems: technological chaos and irreproducible results

Practically no feedback

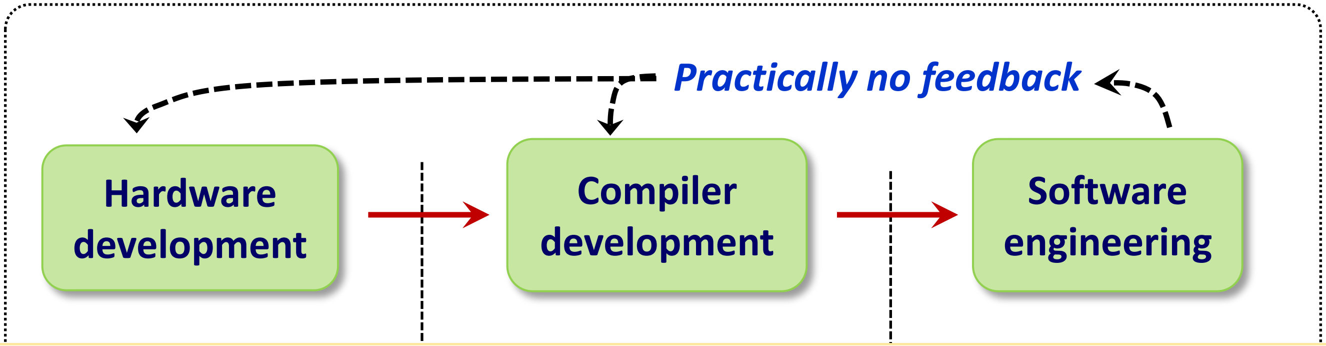
Hardware development → **Compiler development** → **Software engineering**

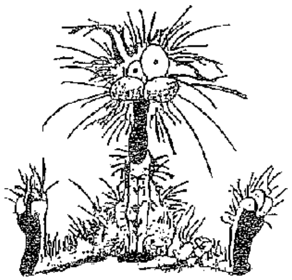


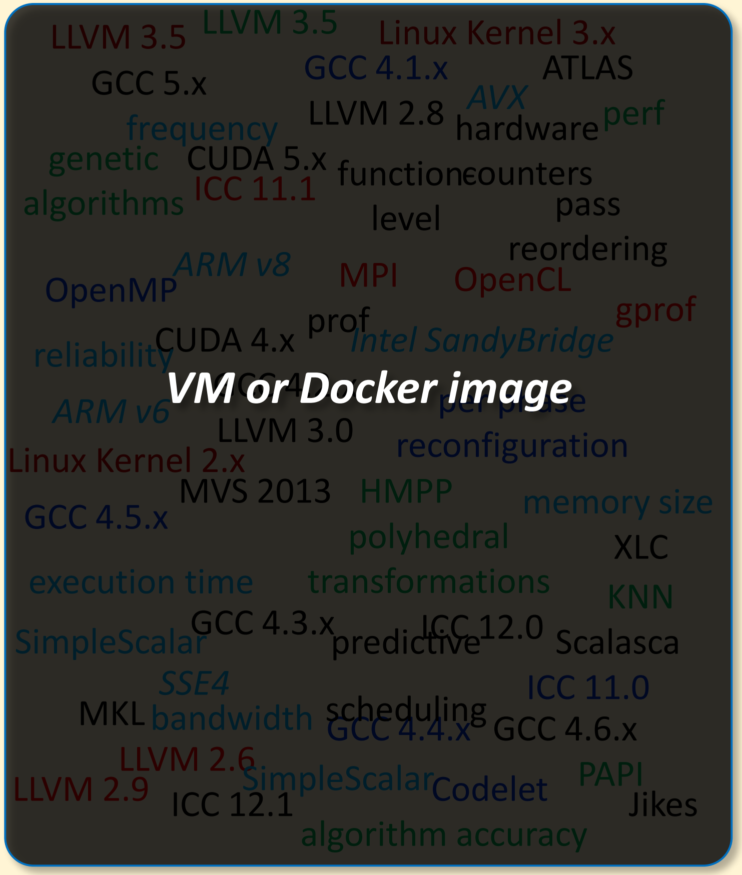
LLVM 3.5 LLVM 3.5 Linux Kernel 3.x
GCC 5.x GCC 4.1.x ATLAS
frequency LLVM 2.8 *AVX* hardware perf
genetic CUDA 5.x function counters
algorithms ICC 11.1 level pass
reordering
OpenMP *ARM v8* MPI OpenCL
gprof
reliability CUDA 4.x prof *Intel SandyBridge*
GCC 4.2.x
ARM v6 per phase
LLVM 3.0 reconfiguration
Linux Kernel 2.x
GCC 4.5.x MVS 2013 HMPP memory size
polyhedral XLC
execution time transformations KNN
SimpleScalar GCC 4.3.x predictive ICC 12.0 Scalasca
SSE4 ICC 11.0
MKL bandwidth scheduling GCC 4.4.x GCC 4.6.x
LLVM 2.6
LLVM 2.9 ICC 12.1 SimpleScalar Codelet PAPI Jikes
algorithm accuracy

- Difficulty to reproduce results (speedups vs optimizations) collected from the community
- Moving research target: continuously evolving software and hardware; stochastic behavior
- Big data problem
- Difficult to expose design and optimization choices
- Difficult to capture all all SW/HW dependencies and run-time state
- Benchmarks and data sets do not have meta-info
- Hardwired workflows with ad-hoc scripts - difficult to customize
- Possibly proprietary benchmarks and compilers

Docker and VM: useful tool to automatically capture all SW deps







VM or Docker do not address many other issues vital for computer systems' research, i.e. how to

1) work with a native user SW/HW environment
2) customize and reuse components (meta-info)
3) capture run-time state
4) deal with hardware dependencies
5) deal with proprietary benchmarks and tools
6) automate validation of experiments

Can be very large in size!

Existing workflow automation tools do not yet address all above problems

Evolvable Collective Knowledge Framework: (2015-cur.)

I would like to

- to organize, describe, interlink, search and reuse my own local research artifacts and workflows while handling evolving SW/HW;
- quickly prototype research ideas from shared components;
- crowdsource and reproduce experiments;
- open my results to powerful predictive analytics;
- enable interactive graphs and articles to share knowledge;
- easily reproduce others' experiments and build upon them

Acknowledgments

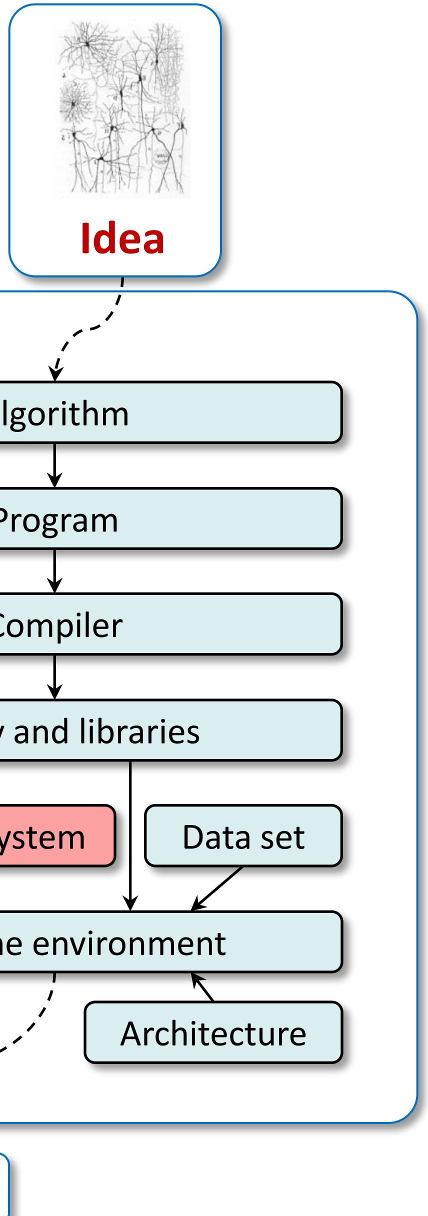
tetracom

SEVENTH FRAMEWORK PROGRAMME

ARM®

cTuning Foundation

Typical experimental workflow in computer engineering



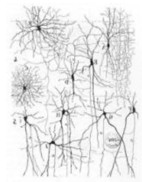
- **get result as fast as possible**
- **minimize all costs**

power consumption, data/memory footprint, inaccuracies, price, size, faults …

- **guarantee some constraints**

power budget, real-time processing, bandwidth, QoS …

Noticed in all past research: similar project structure



Idea

Algorithm

Program

Compiler

Binary and libraries

State of the system

Data set

Run-time environment

Architecture

Result

Create project directory

Ad-hoc scripts to compile and run a program…

- **image corner detection**
- **matmul OpenCL**
- **compression**
- **DNN CUDA/OpenCL**

Have some common meta: *which datasets can use, how to compile, CMD, …*

Ad-hoc scripts to set up environment for a given and possibly proprietary compiler

- **GCC V5.2**
- **LLVM 3.6**
- **Intel Compilers 2015**

Have some common meta: *compilation, linking and optimization flags*

Ad-hoc dirs for data sets with some ad-hoc scripts to find them, extract features, etc

- **image-jpeg-0001**
- **bzip2-0006**
- **txt-0012**
- **video-raw-1280x1024**

Have some (common) meta: *filename, size, width, height, colors, …*

Ad-hoc dirs and scripts to record and analyze experiments

- **cvs speedups**
- **txt hardware counters**
- **xls table with graph**

Have some common meta: *features, characteristics, optimizations*

Convert ad-hoc scripts into Python-wrappers; abstract data; add JSON meta

module UID and alias

data UID and alias

Python module
"program"
with functions:
compile and run

- image corner detection
- matmul OpenCL
- compression
- neural network CUDA

meta.json

Python module
"soft"
with function:
setup

- GCC V5.2
- LLVM 3.6
- Intel Compilers 2015

meta.json

Python module
"dataset"
with function:
extract_features

- image-jpeg-0001
- bzip2-0006
- txt-0012
- video-raw-1280x1024

meta.json

Python module
"experiment"
with function:
add, get, analyze

- cvs speedups
- txt hardware counters
- xls table with graph

meta.json

Provide unified command line front-end (ck)

CK: small python module (~200Kb); no extra dependencies; Linux; Win; MacOS

data UID and alias

ck <function> <module UID>:<data UID> @input.json

JSON input → Python module **"program"** with functions: **compile and run** → *JSON output*

- image corner detection
- matmul OpenCL
- compression
- neural network CUDA

meta.json

JSON input → Python module **"soft"** with function: **setup** → *JSON output*

- GCC V5.2
- LLVM 3.6
- Intel Compilers 2015

meta.json

JSON input → Python module **"dataset"** with function: **extract_features** → *JSON output*

- image-jpeg-0001
- bzip2-0006
- txt-0012
- video-raw-1280x1024

meta.json

JSON input → Python module **"experiment"** with function: **add, get, analyze** → *JSON output*

- cvs speedups
- txt hardware counters
- xls table with graph

meta.json

Helps to implement workflows from CMD as simple as LEGO™

CK: small python module (~200Kb); no extra dependencies; Linux; Win; MacOS

Connect into workflows;

JSON input →

Python module **"program"** with functions: **compile and run**

Python module **"soft"** with function: **setup**

Python module **"dataset"** with function: **extract_features**

Python module **"experiment"** with function: **add, get, analyze**

→ *JSON output*

data UID and alias

- **image corner detection**
- **matmul OpenCL**
- **compression**
- **neural network CUDA**

meta.json

- **GCC V5.2**
- **LLVM 3.6**
- **Intel Compilers 2015**

meta.json

- **image-jpeg-0001**
- **bzip2-0006**
- **txt-0012**
- **video-raw-1280x1024**

meta.json

- **cvs speedups**
- **txt hardware counters**
- **xls table with graph**

meta.json

Pack into directory (CK repository) and share via GitHub/Bitbucket

CK local project repo

/ module UID and alias	/ data UID or alias	/ .cm / meta.json
program	image corner detection	...
	matmul OpenCL	...
	compression	...
	neural network CUDA	...
soft	gcc 5.2	...
	llvm 3.6	...
	icc 2015	...
dataset	image-jpeg-0001	...
	bzip2-0006	...
	video-raw-1280x1024	...
module	program	...
	soft	...
	dataset	...
	experiment	...

Local - can be shared via GIT/SVN/etc

Can be easily connected to HADOOP based repositories

Can be easily connected to powerful predictive analytics

Both code (with API) and data (with meta) inside repository

Can be referenced and cross-linked via CID (similar to DOI but distributed):

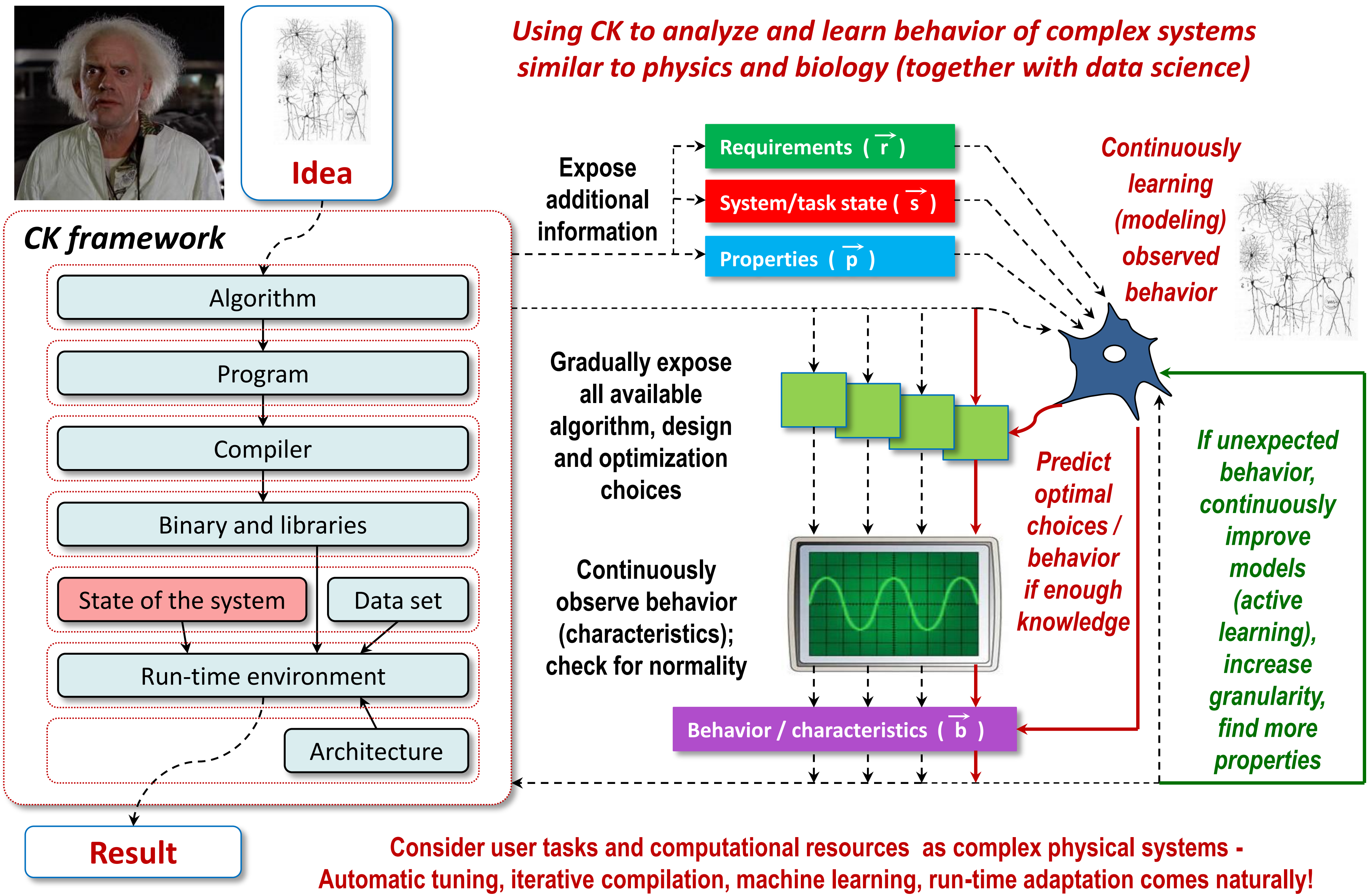
module UOA : data UOA

Making it simple – let researchers quickly prototype ideas!

Create repository:	**ck add repo:my_new_project**
Add new module:	**ck add my_new_project:module:my_module**
Add new data for this module:	**ck add my_new_project:my_module:my_data @@dict**
{"tags":"cool","data"}	
Add dummy function to module:	**ck add_action my_module –func=my_func**
Test dummy function:	**ck my_func my_module**
List my_module data:	**ck list my_module**
Find data by tags:	**ck search my_module –tags=cool**
Pull existing repo from GitHub:	**ck pull repo:ck-autotuning**
List modules from this repo:	**ck list ck-autotuning:module:***
Compile program (using GCC):	**ck compile program: cbench-automotive-susan --speed**
Run program:	**ck run program: cbench-automotive-susan**
Start server for crowdsourcing:	**ck start web**
View interactive articles:	**firefox http://localhost:3344**

Creating new workflows takes from a few minutes to a few hours rather than days and months of hard work!

Can now implement experimental methodology from physics and biology!



Gradually add JSON specification (depends on research scenario)

Autotuning and machine learning specification:

CK flattened JSON key

##characteristics#execution_times@1

```
{
  "characteristics":{
    "execution times": ["10.3","10.1","13.3"],
    "code size": "131938", ...},
  "choices":{
    "os":"linux", "os version":"2.6.32-5-amd64",
    "compiler":"gcc", "compiler version":"4.6.3",
    "compiler_flags":"-O3 -fno-if-conversion",
    "platform":{"processor":"intel xeon e5520",
           "l2":"8192", ...}, ...},
  "features":{
    "semantic features": {"number_of_bb": "24", ...},
    "hardware counters": {"cpi": "1.4" ...}, ... }
  "state":{
    "frequency":"2.27", ...}
}
```

```
"flattened_json_key":{
        "type": "text"|"integer" | "float" | "dict" | "list" | "uid",
        "characteristic": "yes" | "no",
        "feature": "yes" | "no",
        "state": "yes" | "no",
        "has_choice": "yes" | "no",
        "choices": [ list of strings if categorical choice],
        "explore_start": "start number if numerical range",
        "explore_stop": "stop number if numerical range",
        "explore_step": "step if numerical range",
        "can_be_omitted" : "yes" | "no"
        ...
}
```

Quickly prototype experimental workflows from shared components

We can easily assemble, extend and customize research, design and experimentation pipelines for company needs!

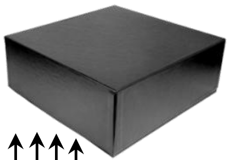
We gradually unify and clean up ad-hoc setups!

- **Init pipeline**
- Detected system information
- Initialize parameters
- Prepare dataset
- **Clean program**
- **Prepare compiler flags**
- Use compiler profiling
- Use cTuning CC/MILEPOST GCC for fine-grain program analysis and tuning
- Use universal Alchemist plugin (with any OpenME-compatible compiler or tool)
- Use Alchemist plugin (currently for GCC)
- **Compile program**
- Get objdump and md5sum (if supported)
- Use OpenME for fine-grain program analysis and online tuning (build & run)
- Use 'Intel VTune Amplifier' to collect hardware counters
- Use 'perf' to collect hardware counters
- Set frequency (in Unix, if supported)
- Get system state before execution
- **Run program**
- Check output for correctness (use dataset UID to save different outputs)
- Finish OpenME
- Misc info
- **Observed characteristics**
- Observed statistical characteristics
- **Finalize pipeline**

http://cknowledge.org/repo

- *Hundreds of benchmarks/kernels/codelets (CPU, OpenMP, OpenCL, CUDA)*
- *Thousands of data sets*
- *Description of major compilers: GCC 4.x, GCC 5.x, LLVM 3.x, ICC 12.x*

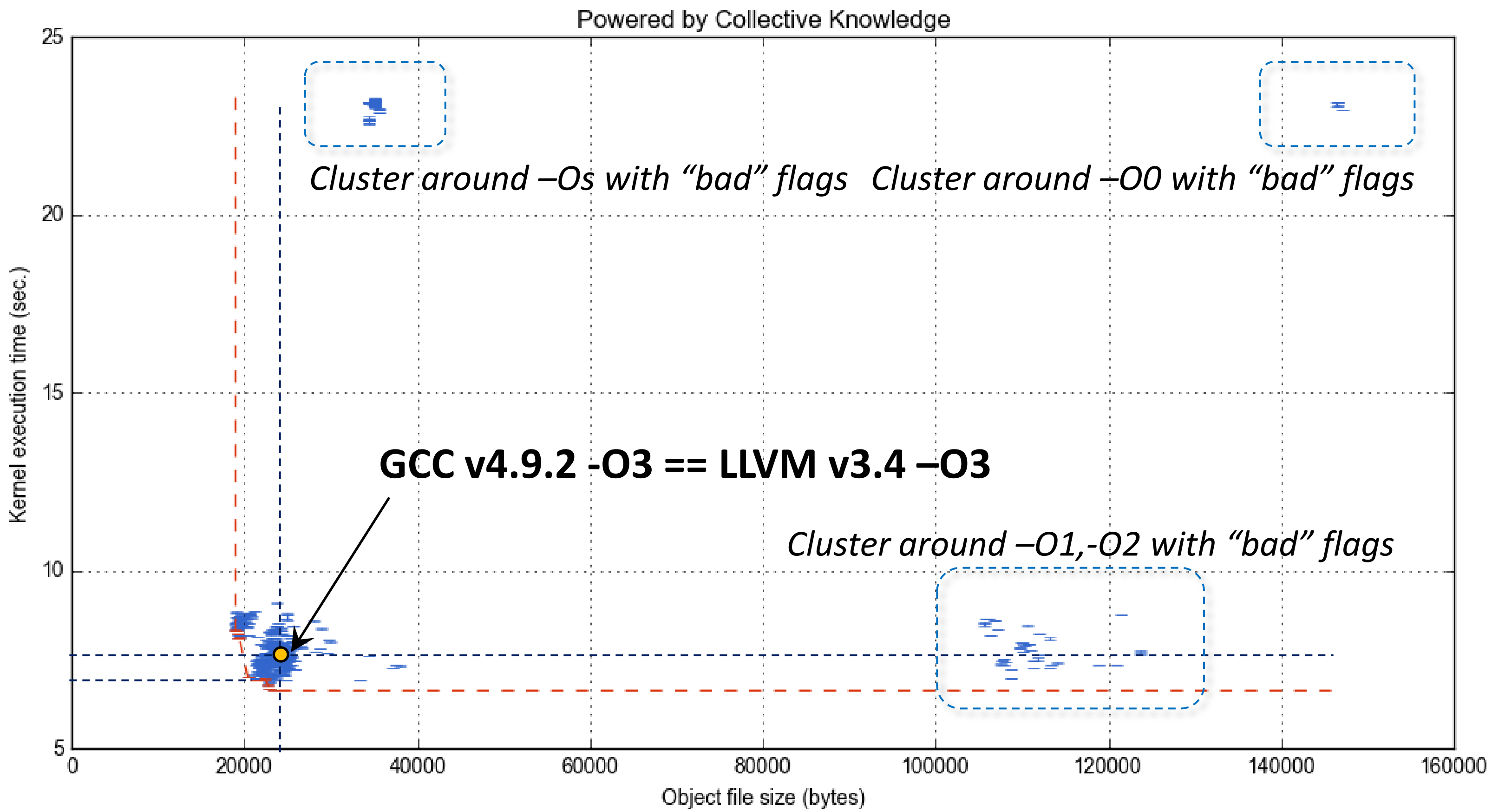
Apply top-down experimental methodology similar to physics



				Gradually expose some characteristics	Gradually expose some choices
	Algorithm selection			(time) productivity, variable-accuracy, complexity …	Language, MPI, OpenMP, TBB, MapReduce …
Compile Program				time …	compiler flags; pragmas …
	Code analysis & Transformations			time; memory usage; code size …	transformation ordering; polyhedral transformations; transformation parameters; instruction ordering …
		Process			
		Thread			
		Function			
		Codelet			
		Loop			
			Instruction		
Run code	Run-time environment			time; power consumption …	pinning/scheduling …
	System			cost; size …	CPU/GPU; frequency; memory hierarchy …
	Data set			size; values; description …	precision …
	Run-time analysis			time; precision …	hardware counters; power meters …
	Run-time state			processor state; cache state …	helper threads; hardware counters …
Analyze profile				time; size …	instrumentation; profiling …

Coarse-grain vs. fine-grain effects: depends on user requirements and expected ROI

Crowdsourcing iterative compilation using mobile devices



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 -O3 -f**(no-)**FLAG*

Collective Mind Node (Android App on Google Play):
https://play.google.com/store/apps/details?id=com.collective_mind.node

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

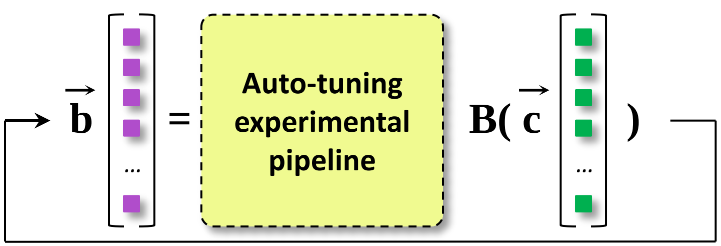
Not very useful for analysis; SHOULD NOT BE USED for machine learning

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

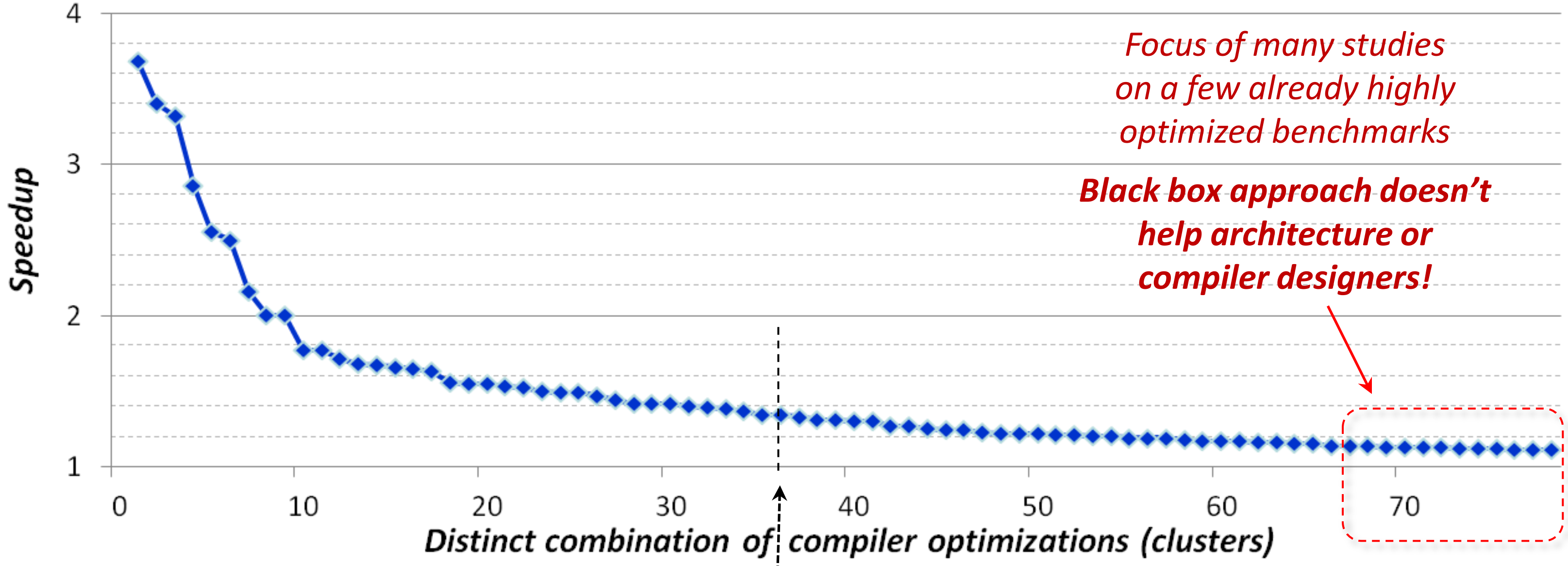
remove dimensions (or set to default)
iteratively, ANOVA, PCA, etc…



Pruned solution

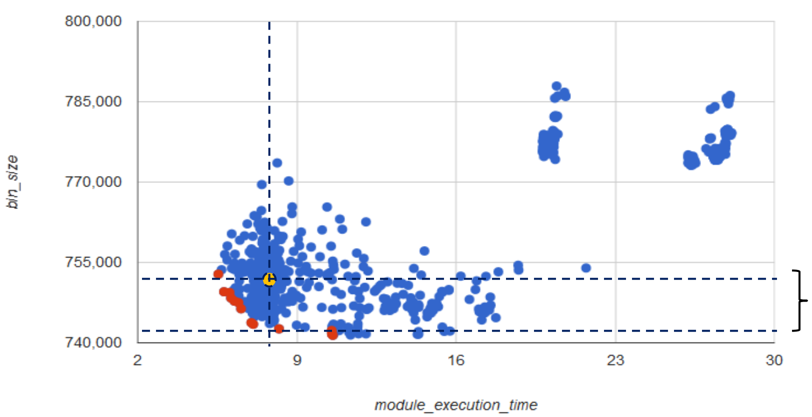
-O3
-fno-align-functions ***(25% of speedup)***
-fdce
-fgcse
-fguess-branch-probability ***(60% of speedup)***
-fmove-loop-invariants
-fomit-frame-pointer
-ftree-ter
-funswitch-loops
-fno-ALL

Crowdsourcing and clustering compiler optimizations



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

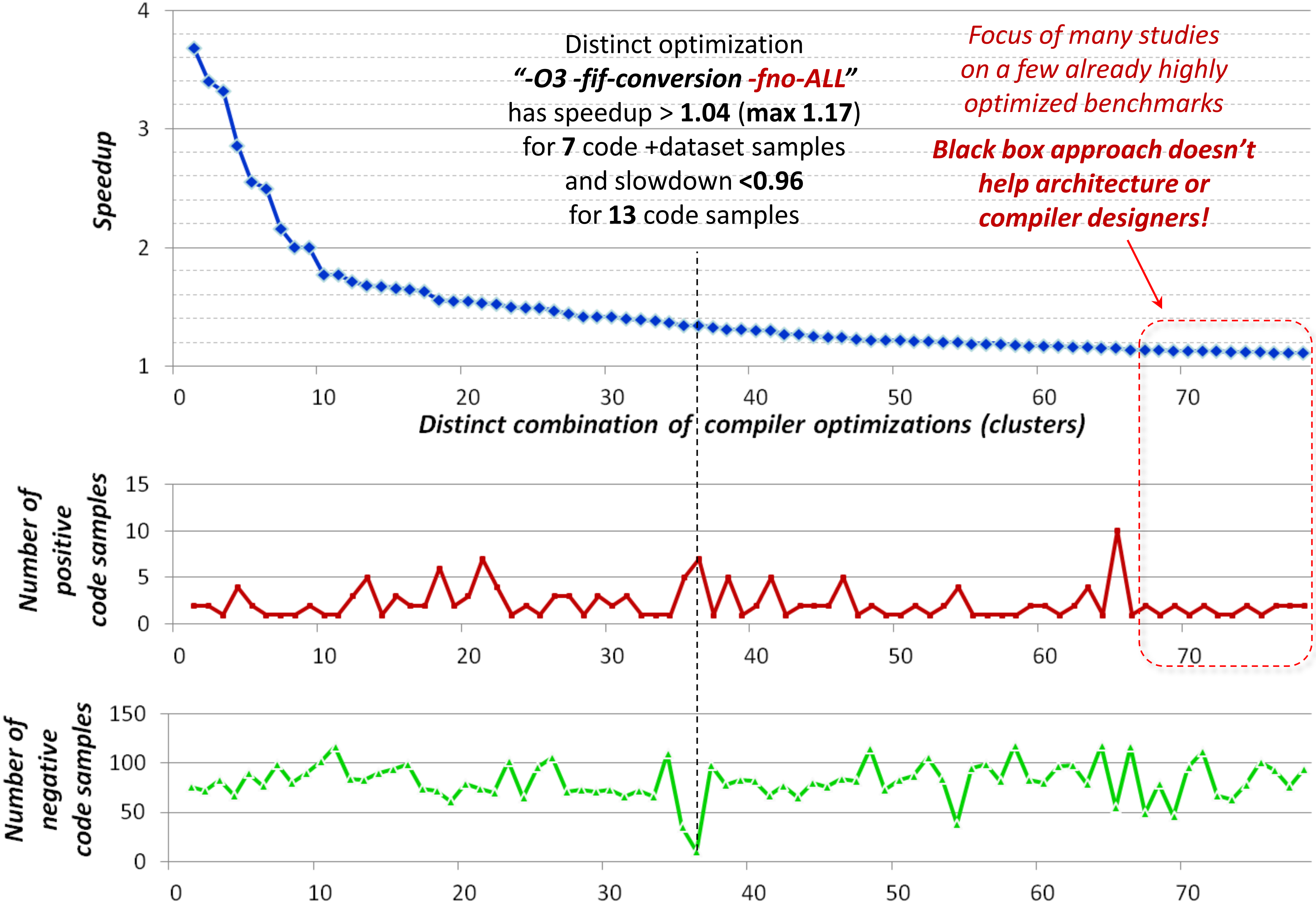
using GRID 5000; Intel E5520, 2.6MHz;
GCC 4.6.3; at least 5000 random combinations of flags



Continuously tuning (crowd-tuning) shared benchmarks and datasets using GRID5000, mobile phones, tablets, laptops, and other spare resources:

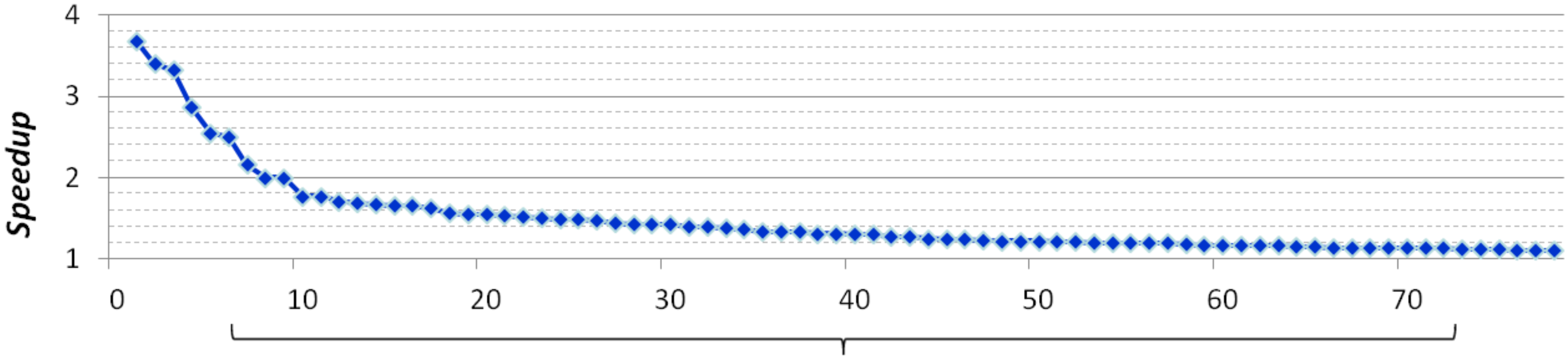
Collective Mind Node (Android Apps on Google Play):
https://play.google.com/store/apps/details?id=com.collective_mind.node

Crowdsourcing and clustering compiler optimizations

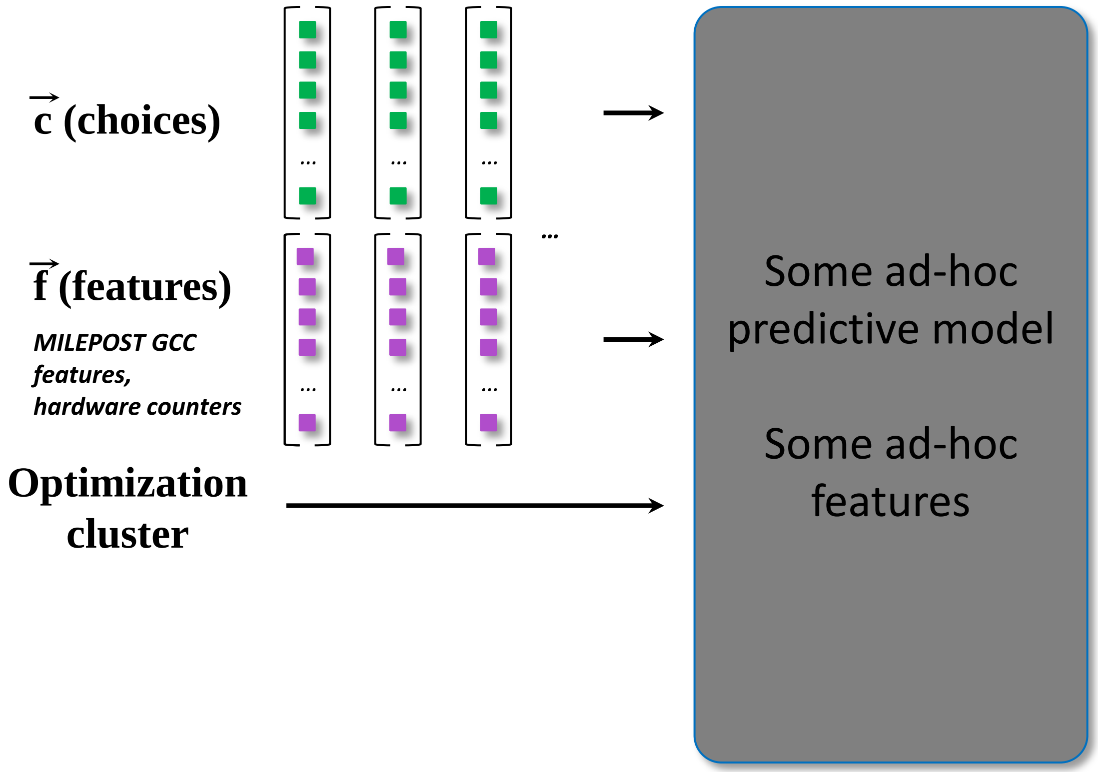


Grigori Fursin, Anton Lokhmotov, et.al. “Collective Mind, Part II: Towards Performance and Cost-Aware Software Engineering as a Natural Science”, CPC’15, London, UK

Current machine learning usage



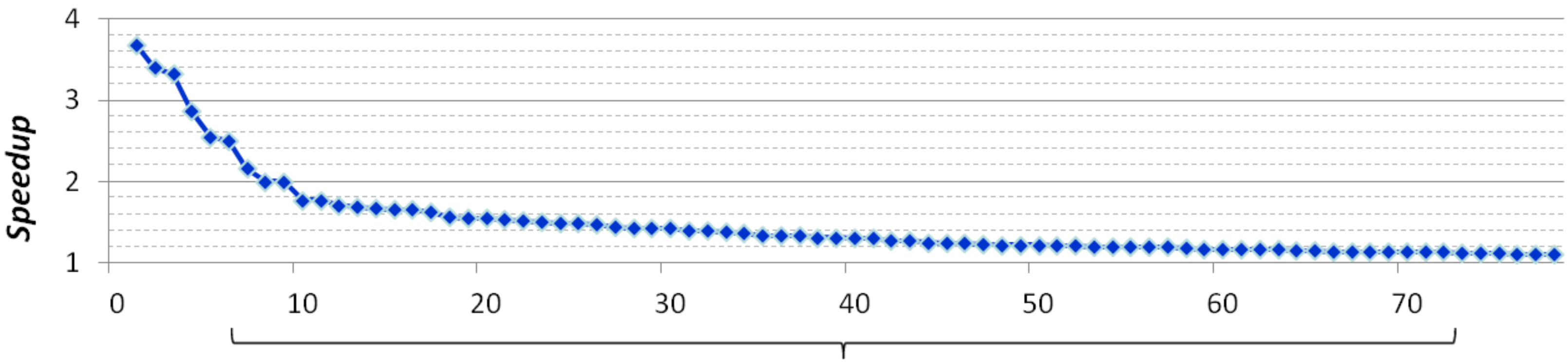
Training set: distinct combination of compiler optimizations (clusters)



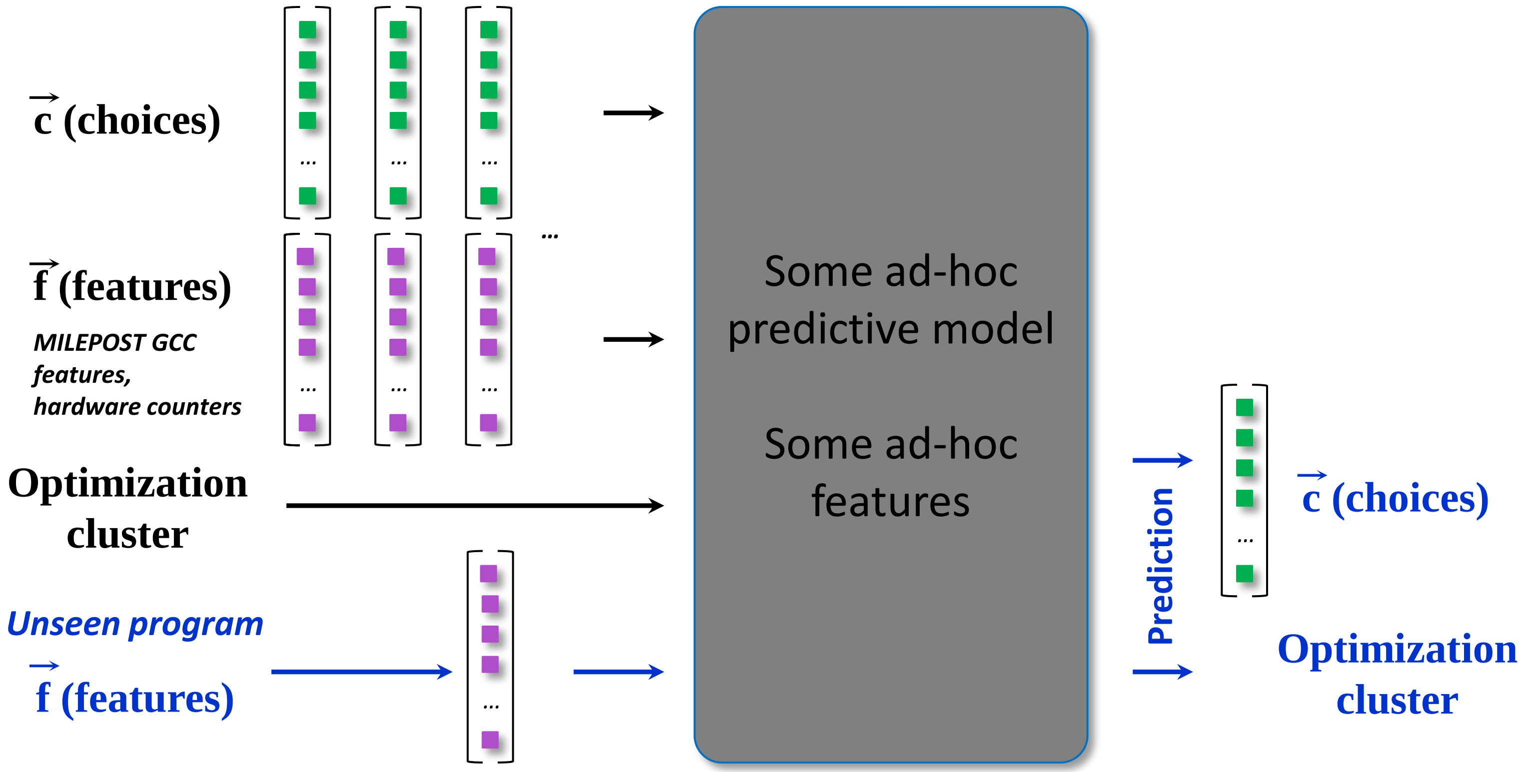
MILEPOST GCC features:

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

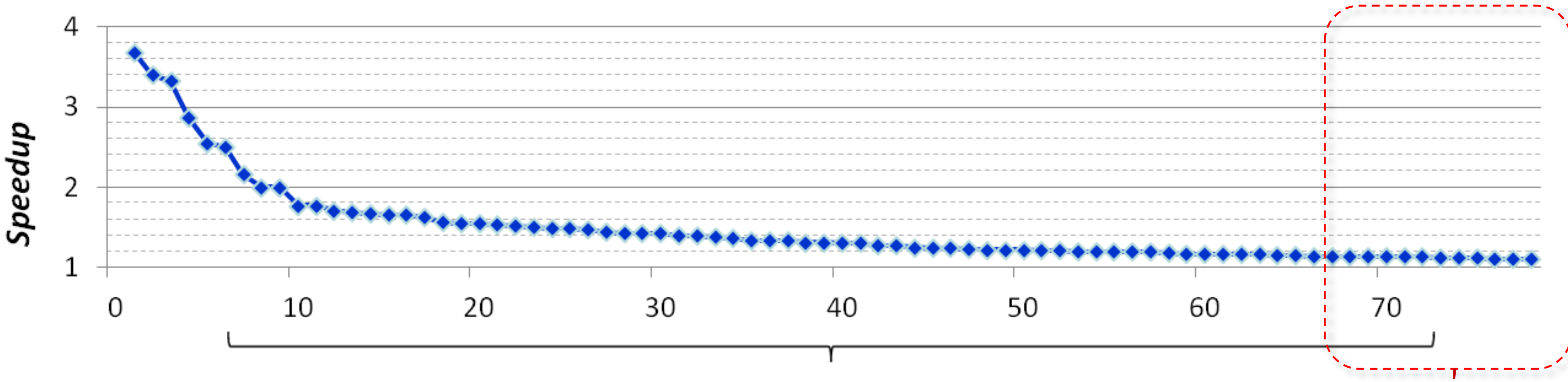
Current machine learning usage



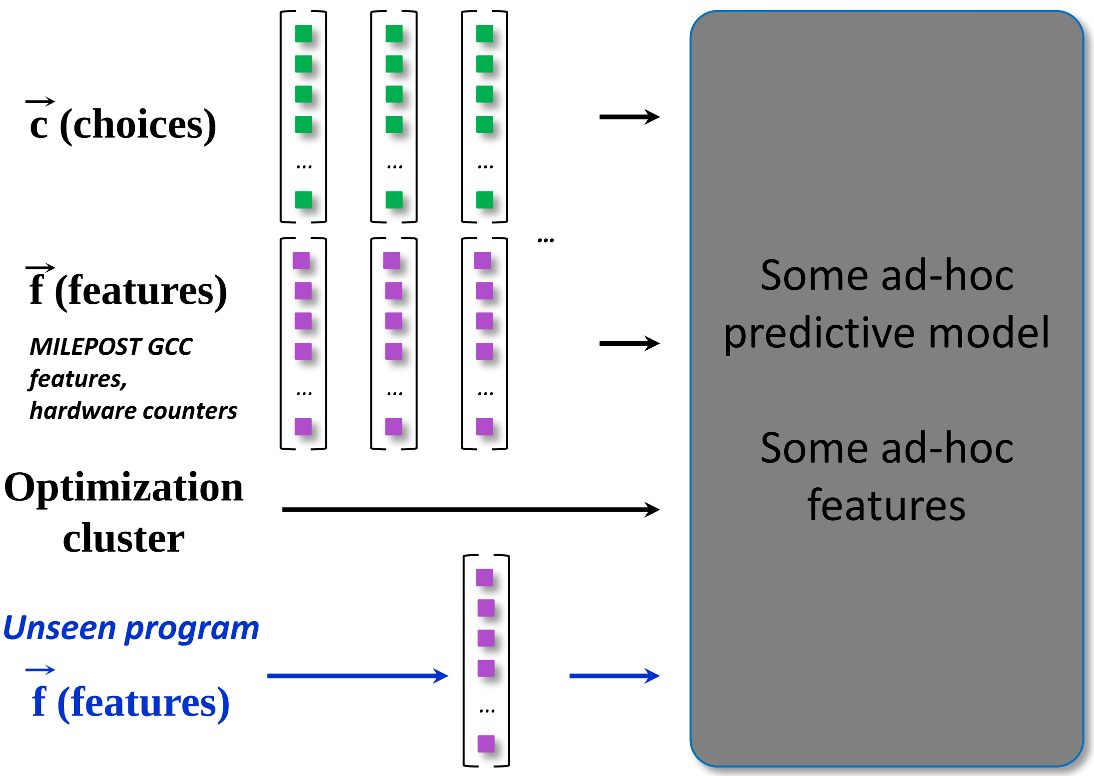
Training set: distinct combination of compiler optimizations (clusters)

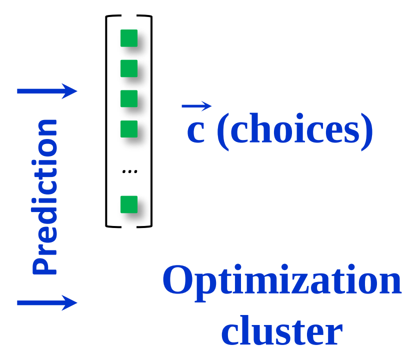


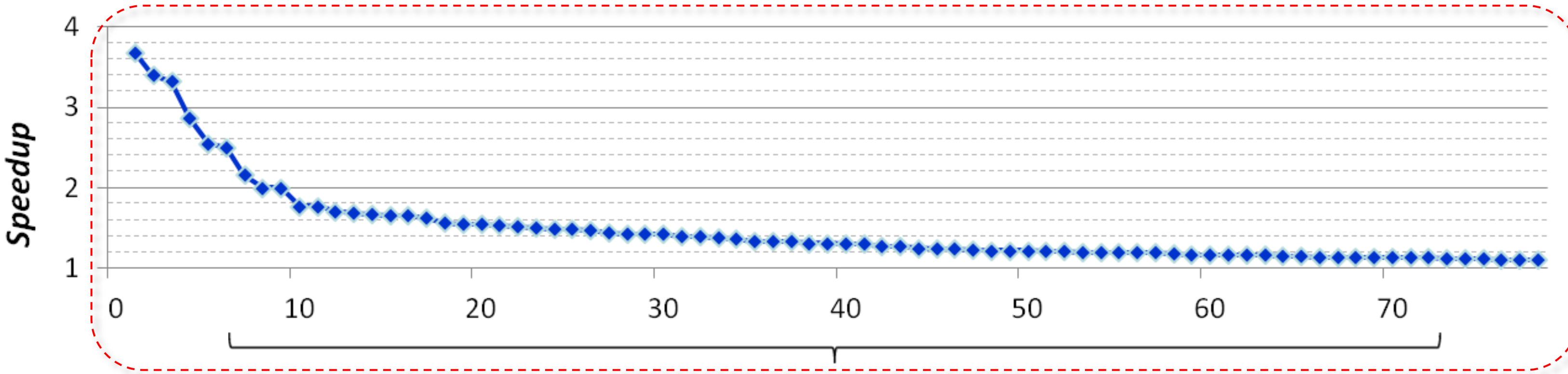
Current machine learning usage



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







Training set: distinct combination of compiler optimizations (clusters)

$\vec{c}$ (choices)

$\vec{f}$ (features)

MILEPOST GCC features, hardware counters

Optimization cluster

Unseen program

$\vec{f}$ (features)

Some ad-hoc predictive model

Some ad-hoc features

Number of code and dataset samples	Prediction accuracy using optimized SVM, KNN
12	87%
285	56% (no prediction)

Prediction

$\vec{c}$ (choices)

Optimization cluster

Why?

Common pitfall – missing features

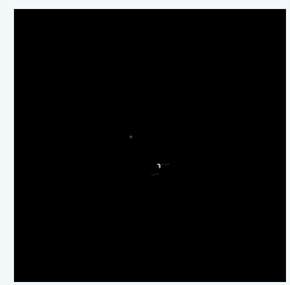
Learning features by domain specialists

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***	**-11.9% (degradation)**
Shared data set $sample_2$	no change	***+17.3% (improvement)***

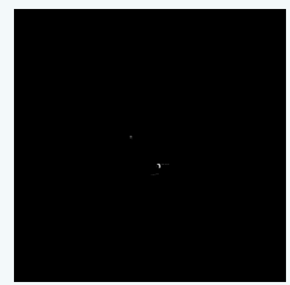
Learning features by domain specialists

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*** 	**-11.9% (degradation)** 
Shared data set $sample_2$	no change 	***+17.3% improvement*** 

Learning features by domain specialists

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*** 	**-11.9% (degradation)** 
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!
Feature generators would not help either!

Need split-compilation (multi-versioning and run-time adaptation)

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

Adaptive workload scheduling combined with active learning

Our user had an real-time and machine-learning based image processing applications run on mobile device with GPUs – should it be always offloaded to GPU?

ck build model.sklearn *ck validate module.sklearn*

(operates with 'features' and 'characteristics' keys in JSON)

Application:

OpenCL based real time video stream processing for mobile devices

Experiments:

276 builds/runs with random features

Characteristics:

CPU execution time

GPU ONLY execution time

GPU + MEM COPY execution time

Devices:

Chromebook 1: 4x Mali-T60x / 2x A15

Chromebook 2: 4x Mali-T62x / 4x A15

Objective (divide execution time):

CPU/GPU COPY > 1.07 (true/false)?

(useful for adaptive scheduling)

Original features (properties) :

V1=GWS0

V2=GWS1

V3=GWS2

V4=cpu_freq

V5=gpu_freq

V6=block size

V7=image cols

V8=image rows

Designed features:

V9=image size

V10=size_div_by_cpu_freq

V11=size_div_by_gpu_freq

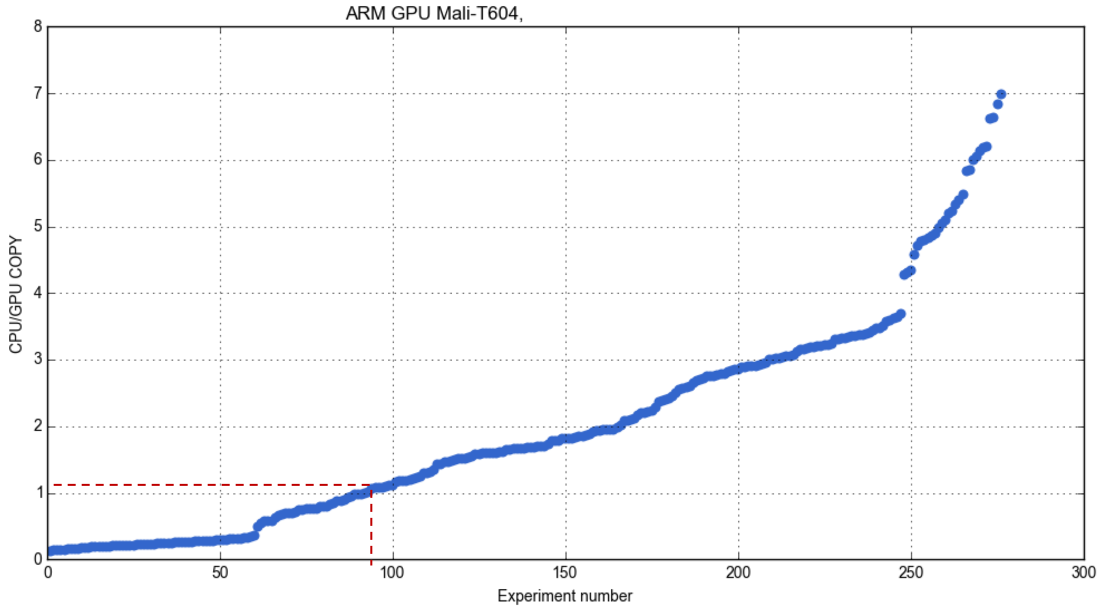
V12=cpu_freq_div_by_gpu

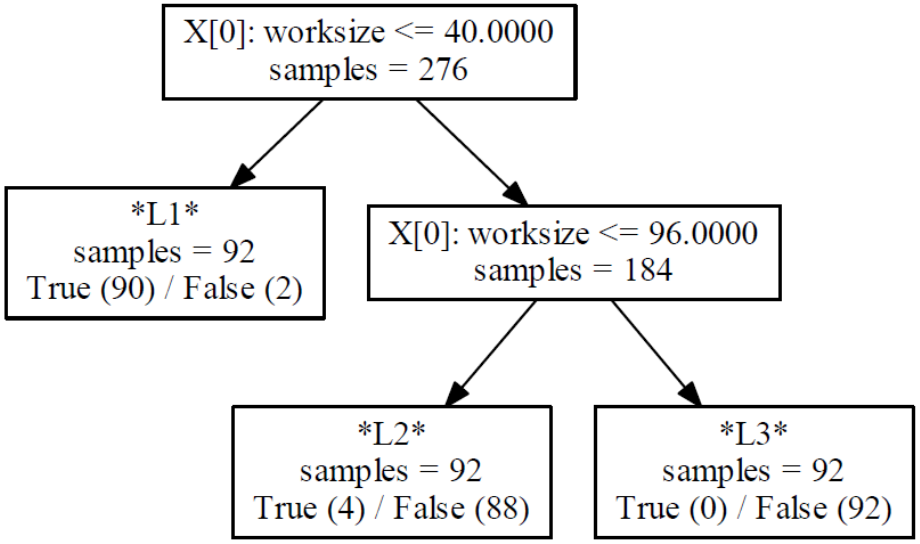
V13=size_div_by_cpu_div_by_gpu_freq

V14=image_size_div_by_cpu_freq

EU FP7 TETRACOM project: cTuning and ARM

Adaptive workload scheduling combined with active learning





Samsung Chromebook$_1$

Automatically built decision tree with scikit-learn when more data is available.

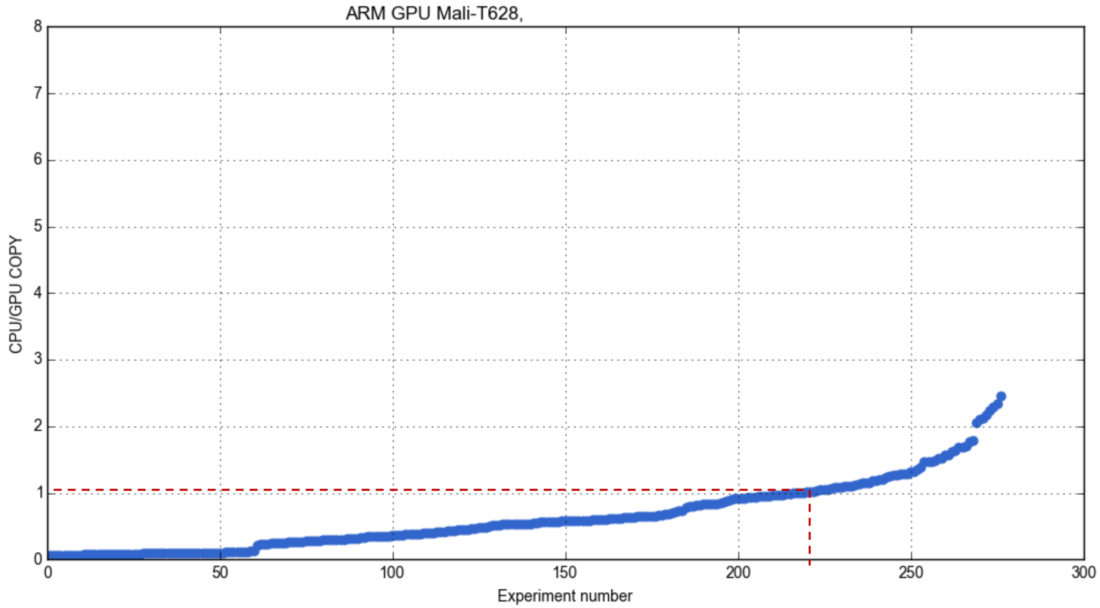
Not a black box - gives hints to engineers where to focus their attention.

Can drive further exploration on areas with “unusual” behavior.

96% prediction rate

EU FP7 TETRACOM project: cTuning and ARM

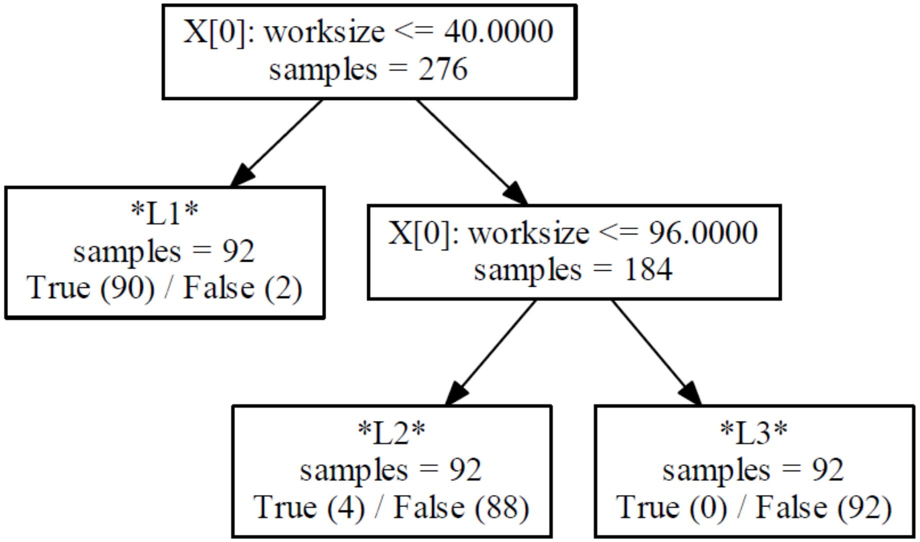
Adaptive workload scheduling combined with active learning



Samsung Chromebook$_2$

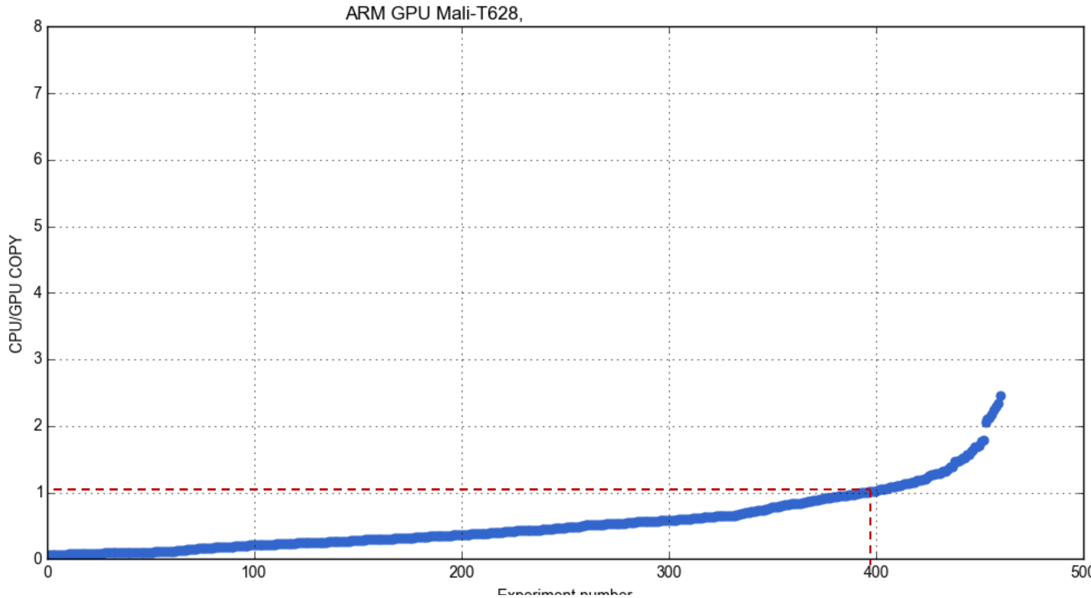
Using old model

74% prediction rate



EU FP7 TETRACOM project: cTuning and ARM

Adaptive workload scheduling combined with active learning

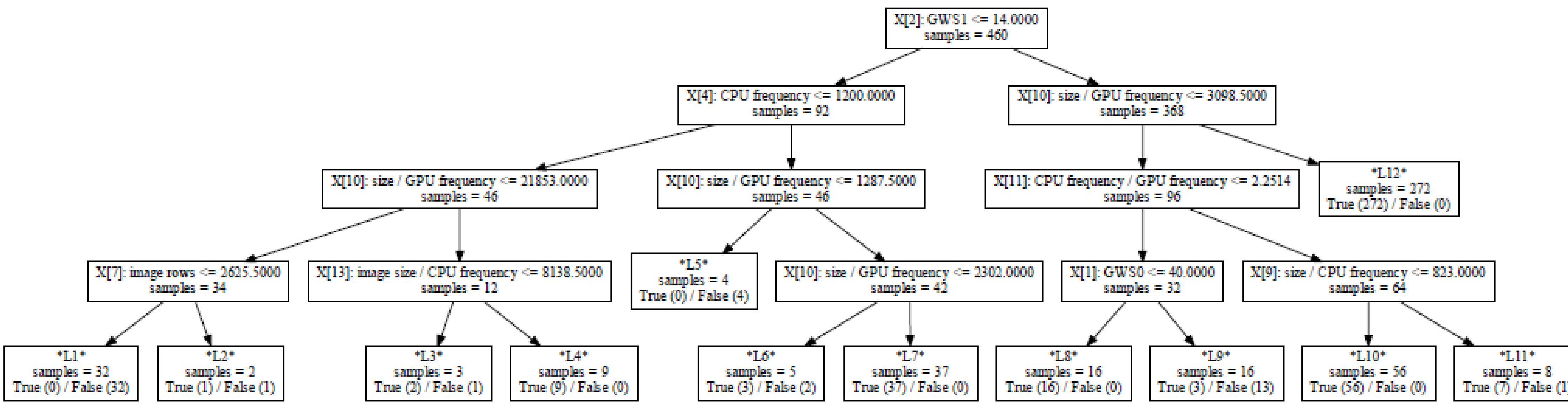


Samsung Chromebook$_2$

More data, more features, better model

96% prediction rate

ADAPTIVE SCHEDULING gives ~32% performance improvement in comparison with always using GPU



Results shared with the community for reproducibility:

cknowledge.org/repo/web.php?wcid=bc0409fb61f0aa82:fd54cd4b3b73b72b
cknowledge.org/repo/web.php?wcid=bc0409fb61f0aa82:3bfd697a48fbba16

Converted 2 projects to CK: http://github.com/ctuning/reproduce-*

SLAMBench from PAMELA project (OpenCL, CUDA, CPU)

Real, live, 3D scene processing application

HOG from CARP project (OpenCL, CPU, TBB)

Real, live, 2D image processing application

We converted it to CK to balance FPS, accuracy and energy across numerous platforms and environments (Linux, Windows, Android, MacOS)

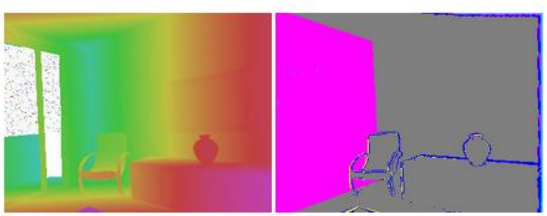
Run-time state (via OpenME):

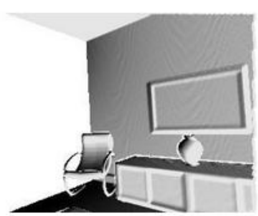
Average FPS: 1.40

FPS per thread: 1.40
Threads: 1

Elapsed time: 416.97
Frames: 585

Image: 640x480





http://cknowledge.org/interactive-reports

Reproducibility of experimental results as a side effect

Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

Execution time:

10 sec.

Reproducibility of experimental results as a side effect

Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

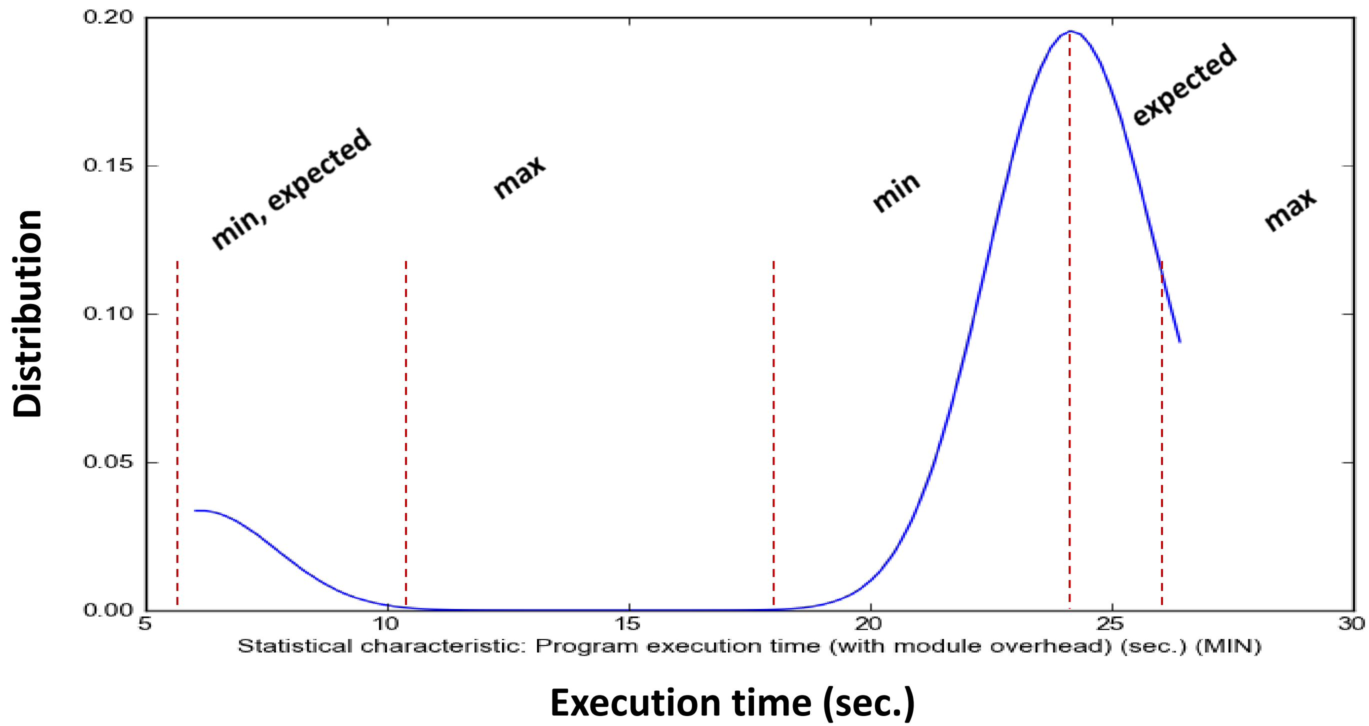
Variation of experimental results:

10 ± 5 secs.

Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

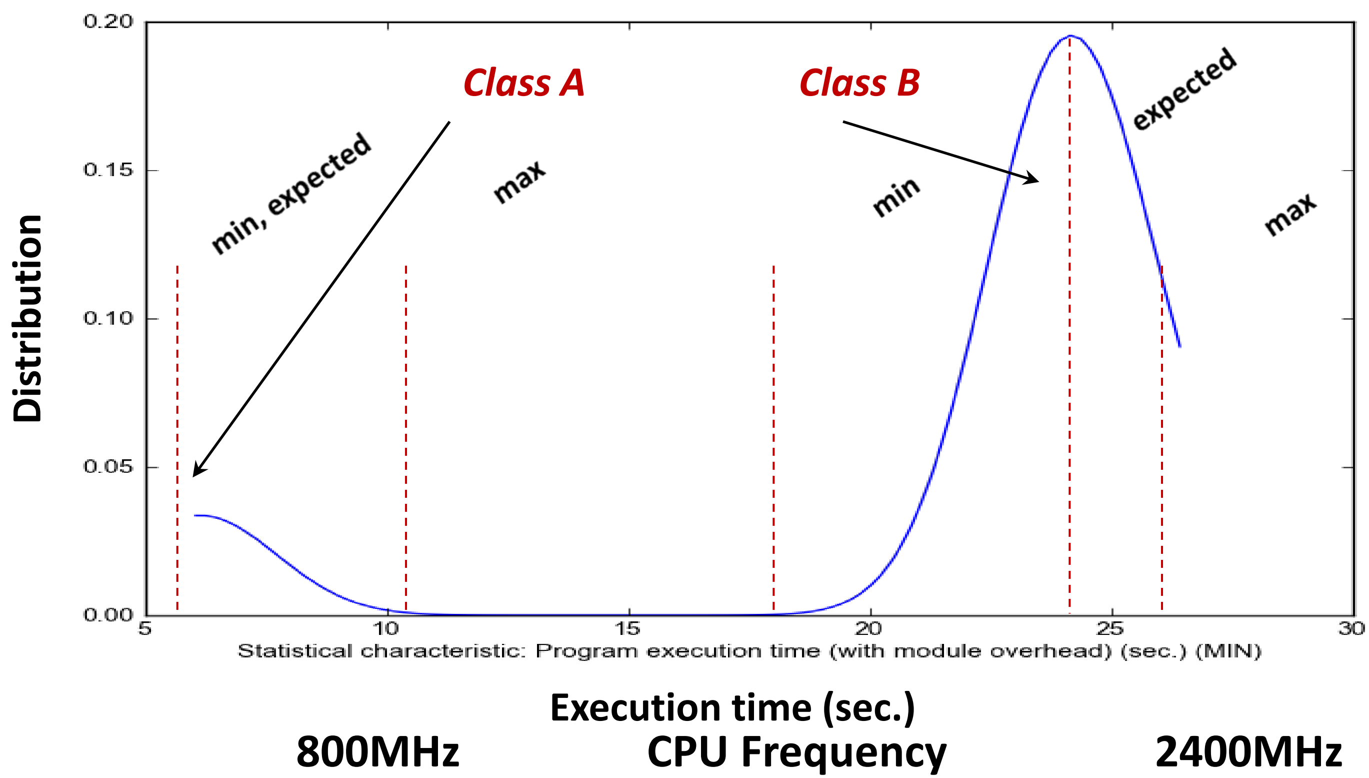
Unexpected behavior - expose to the community including experts to explain, find missing feature and add to the system



Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

Unexpected behavior - expose to the community including experts to explain, find missing feature and add to the system



Enabling open computer systems' research

Enabling collaborative and reproducible research and experimentation in computer engineering similar to natural sciences (physics, biology)

- Submit papers to open access archives (arXiv, HAL, etc)
- Make all related research material either at the personal website or at public sharing services
- Initiate discussion at social networking sites with ranking (Reddit, SlashDot, StackExchange) or without (Google+, Facebook)
- Arrange first small program committee that monitors discussions to filter obviously wrong, unreproducible or possibly plagiarized
- Select a set of "interesting" papers and send it to a interdisiplinary program committee based on paper topics and public discussions
- Select final papers based on public discussions and professional reviews
- Create an open access reproducible online journal with all related materials from the most interesting, advanced and highest ranked publications
- Send considerably updated papers to traditional journals (not to break current system but make open access and traditional publication models co-exist)

*Grigori Fursin and Christophe Dubach, **"Community-driven reviewing and validation of publications",** Proceedings of the 1st ACM SIGPLAN TRUST Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering, 2014*

Can it work? Our experience with cTuning/MILEPOST

Since 2006 I share all my code, data and experimental results – it's fun and motivating working with the community!

Some comments about MILEPOST GCC from Slashdot.org:

http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

GCC goes online on the 2nd of July, 2008.
Human decisions are removed from compilation.
GCC begins to learn at a geometric rate.
It becomes self-aware 2:14 AM, Eastern time, August 29th.
In a panic, they try to pull the plug. GCC strikes back…

Can it work? Our experience with cTuning/MILEPOST

Since 2006 I share all my code, data and experimental results – it's fun and motivating working with the community!

Some comments about MILEPOST GCC from Slashdot.org:

http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

GCC goes online on the 2nd of July, 2008.
Human decisions are removed from compilation.
GCC begins to learn at a geometric rate.
It becomes self-aware 2:14 AM, Eastern time, August 29th.
In a panic, they try to pull the plug. GCC strikes back…

Community was interested to validate and improve techniques!
Community can identify missing related citations and projects!
Open discussions can provide new directions for research!
You can fight wrong or biased reviews!

Can it work? Our experience with cTuning/MILEPOST

Since 2006 I share all my code, data and experimental results – it's fun and motivating working with the community!

Some comments about MILEPOST GCC from Slashdot.org:

http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

GCC goes online on the 2nd of July, 2008.
Human decisions are removed from compilation.
GCC begins to learn at a geometric rate.
It becomes self-aware 2:14 AM, Eastern time, August 29th.
In a panic, they try to pull the plug. GCC strikes back…

Community was interested to validate and improve techniques!
Community can identify missing related citations and projects!
Open discussions can provide new directions for research!
You can fight wrong or biased reviews!

Successfully validated at ADAPT'16 (adapt-workshop.org)
workshop on adaptive, self-tuning computing systems
Reddit discussion: https://www.reddit.com/r/adaptworkshop
Artifacts: 2 shared in CK format (OpenCL crowd-tuning + bug detection)

cTuning.org/ae

- Artifact Evaluation for CGO'15/PPoPP'15 (18 artifacts submitted)
- Artifact Evaluation for CGO'16/PPoPP'16 (23 artifacts submitted)
- Dagstuhl Perspective Workshop on Artifact Evaluation in November
(Bruce Childers, Grigori Fursin, Shriram Krishnamurthi, Andreas Zeller)
- Discussions with ACM on unification of AE

Conclusions: Collective Knowledge approach to computer engineering

- Changing the mentality of computer systems' researchers:
    - sharing artifacts and workflows
    - crowdsourcing experiments and sharing negative/unexpected results
    - collaboratively improving reproducibility
    - collaboratively improving prediction models and finding missing features
    - formulating and solving important real-world problems
- Defining representative workloads for the future
- Bringing closer together industry and academia
  (common research methodology, reproducible research, real data access)
- Enabling disruptive innovation:
    - Fujitsu made a press-release in 2014 about their $100-million Exascale project combined with autotuning and machine learning, referencing our technology as inspiration

http://github.com/ctuning/ck **http://cknowledge.org/repo**

A few references

- “Collective Tuning Initiative: automating and accelerating development and optimization of computing systems”, GCC Summit 2009
  https://hal.inria.fr/inria-00436029
- “Collective optimization: A practical collaborative approach”, v7, #4, ACM TACO 2010
  https://hal.inria.fr/inria-00436029
- “Milepost GCC: Machine Learning Enabled Self-tuning Compiler”, IJPP 2011
  https://hal.inria.fr/inria-00436029
- “Community-driven reviewing and validation of publications”, TRUST’14@PLDI’14
  http://arxiv.org/abs/1406.4020
- "Collective Mind: Towards practical and collaborative autotuning“,
  Journal of Scientific Programming 22 (4), 2014
  http://hal.inria.fr/hal-01054763
- “Collective Mind, Part II: Towards Performance- and Cost-Aware
  Software Engineering as a Natural Science”, CPC 2015, London, UK,
  http://arxiv.org/abs/1506.06256
- “Collective Mind Node: crowdsourcing iterative compilation across mobile phones”,
  http://cTuning.org/crowdtuning-node
- “Collective Knowledge: towards R&D sustainability”, DATE 2016, Dresden, Germany
  TO APPEAR

It’s only the beginning of the new and exciting journey!

Establishing industrial and academic consortiums and laboratories

Preparing interactive lectures with shared artifacts and reproducible experiments

$$\frac{d\vec{v}}{dt}$$

Join us!

cTuning Foundation

Grigori.Fursin@cTuning.org / grigori@dividiti.com

http:/github.com/ctuning/ck