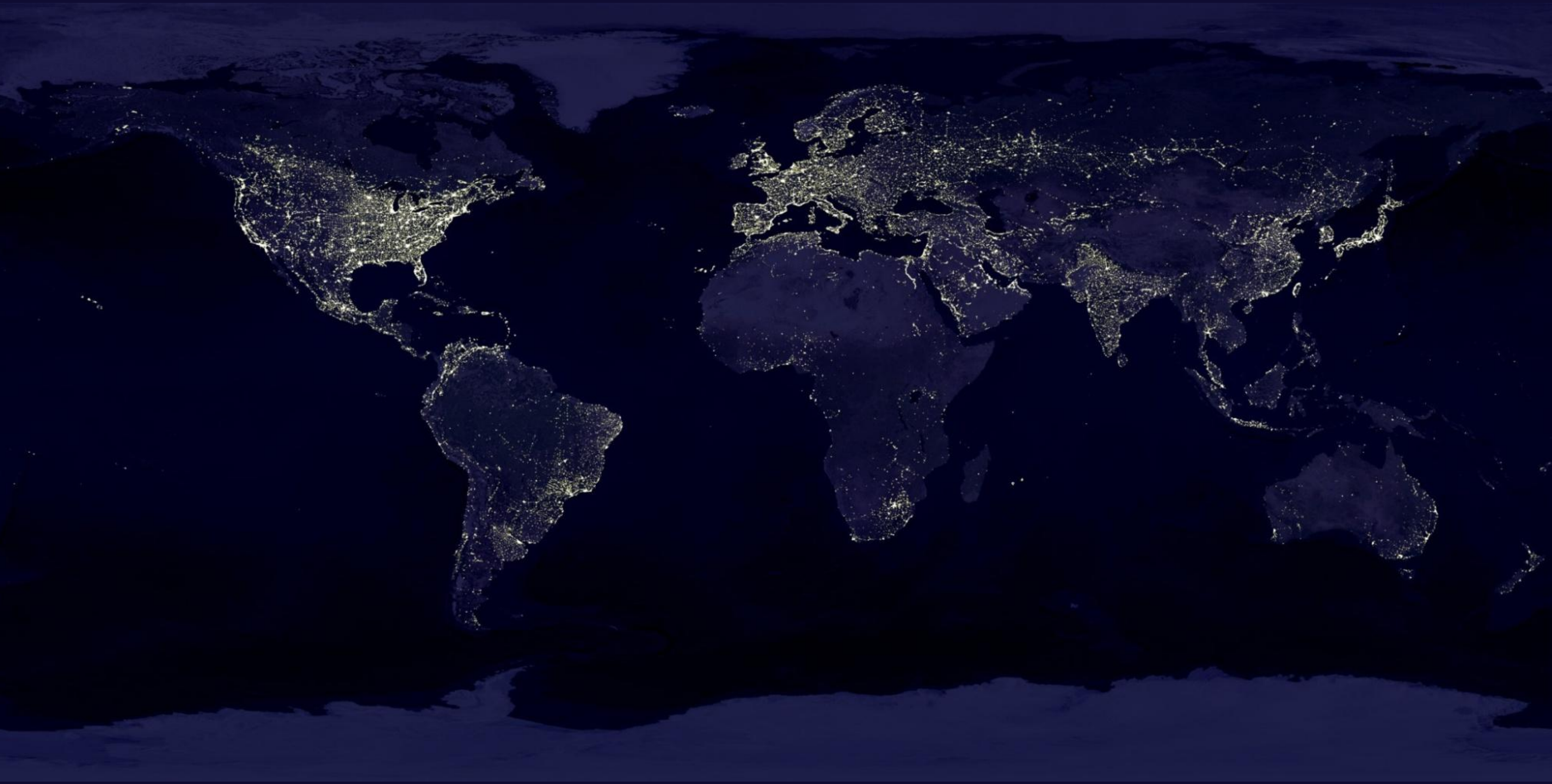
Collaborative application characterization and optimization



Grigori Fursin
INRIA, France

SEA, NCAR
February 2012

Abstract

Improving utilization of computational resources

through combination of auto-tuning, statistical analysis,

data mining, run-time adaptation

and collaborative participation of users

Covers around 13 years of R&D

Outline

- Background
- Motivation, challenges
- cTuning: Collective Tuning Initiative
    - Collective Optimization Repository
    - Empirical auto-tuning and predictive modeling
    - Interactive Compilation Interface
    - Machine learning and collaborative characterization
    - Run-time adaptation
- Summary and roadmap on cTuning2
- References

Background

1993-1997: BS in physics and electronics from MIPT (Russia)
Semiconductor neural networks combined with computer modeling

1998-1999: MS in computer engineering from MIPT (Russia)
Optimization of modeling software, HPC, GRID

1999-2004: PhD in computer science from the University of Edinburgh (UK)
Empirical auto-tuning, statistical analysis, machine learning

2005-2007: Postdoctoral researcher at INRIA Saclay (France)
Run-time adaptation, machine learning
Event-driven plugin framework for GCC/Open64 to “open up” compilers

2007-2010: Permanent research scientist at INRIA (France)
Part-time lecturer at Paris South University (France)
Collaborative (collective) optimization (cTuning.org)
Machine learning enabled self-tuning compiler (MILEPOST GCC)

2010-2011: Director of research and project manager at Intel Exascale Lab (France)
Application characterization and optimization for exascale systems

2012-cur.: Permanent research scientist at INRIA (France)
Open source infrastructure and repository for collaborative program and architecture tuning (performance, power) combined with data sharing and mining

Motivation

End-users demand:

- Increased computational resources
- Reduced costs

Resource providers need:

- Better products
- Faster time to market
- Increased Return on Investment (ROI)

Motivation

End-users demand:

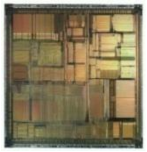
- Increased computational resources
- Reduced costs

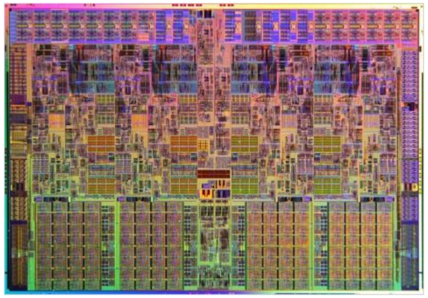
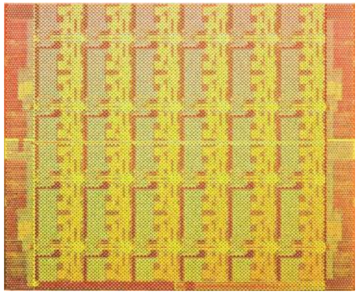
Resource providers need:

- Better products
- Faster time to market
- Increased Return on Investment (ROI)

Computer system designers produce:

Rapidly evolving HPC systems
that already reach petaflop and start targeting exaflop performance.



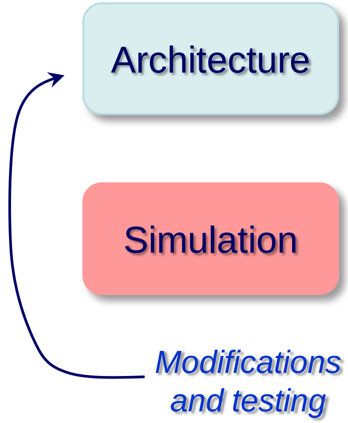



In the near future HPC systems may feature
millions of processors with hundreds of homo- and heterogeneous cores per processor.

Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

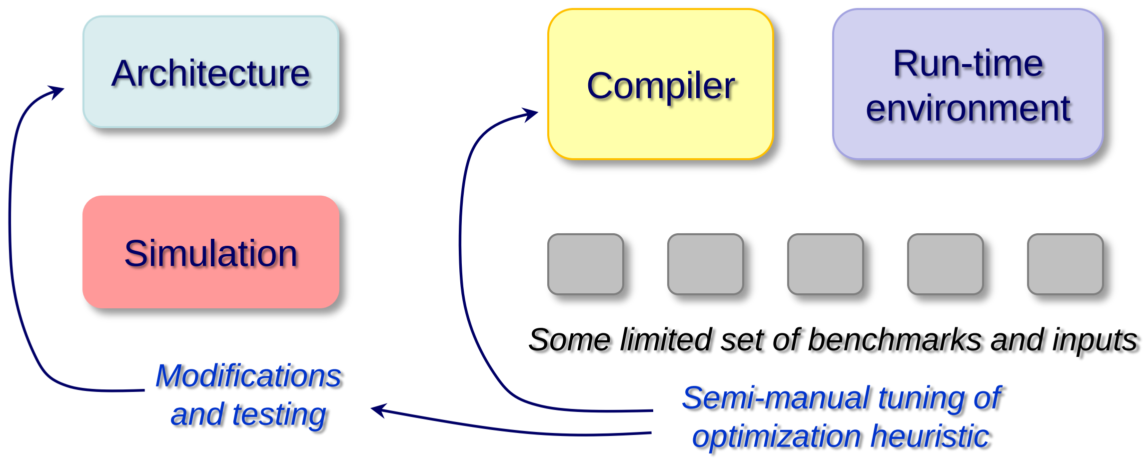


Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

2) Compiler is designed and tuned for a limited set of benchmarks / kernels.



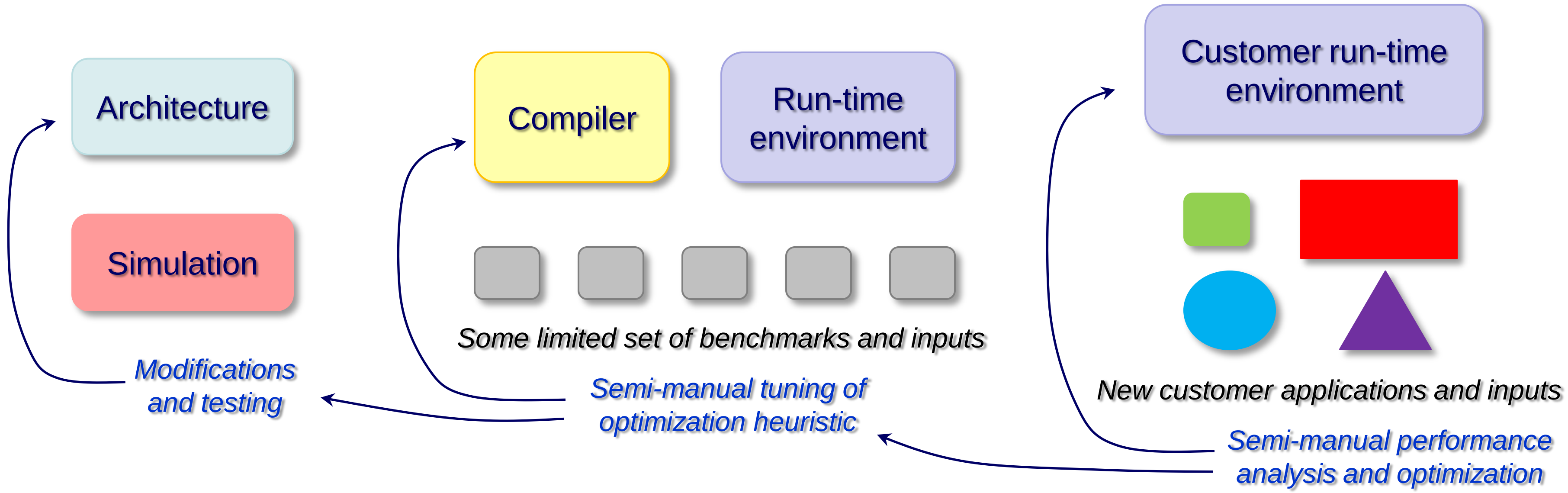
Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

2) Compiler is designed and tuned for a limited set of benchmarks / kernels.

3) System is delivered to a customer. New applications are often underperforming and have to be manually analysed and optimized.



Motivation: auto-tuning

Potential solution during last 2 decades:
auto-tuning (iterative compilation)

Learn behavior of computer systems across executions while tuning various parameters

Optimization spaces:

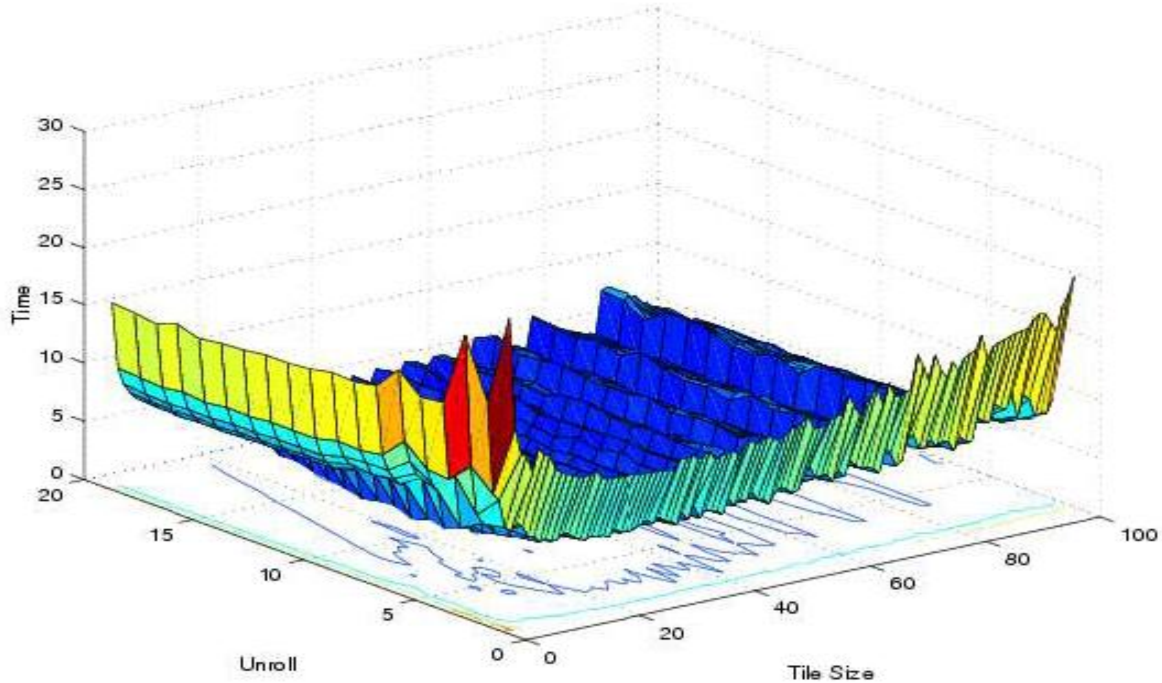
- combinations of compiler flags
- parametric transformations and their ordering
- cost-model tuning for individual transformations (meta optimization)
- parallelization (OpenMP vs MPI, number of threads)
- scheduling (heterogeneous systems, contention detection)
- architecture designs (cache size, frequency)

…

Motivation: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments. Why?

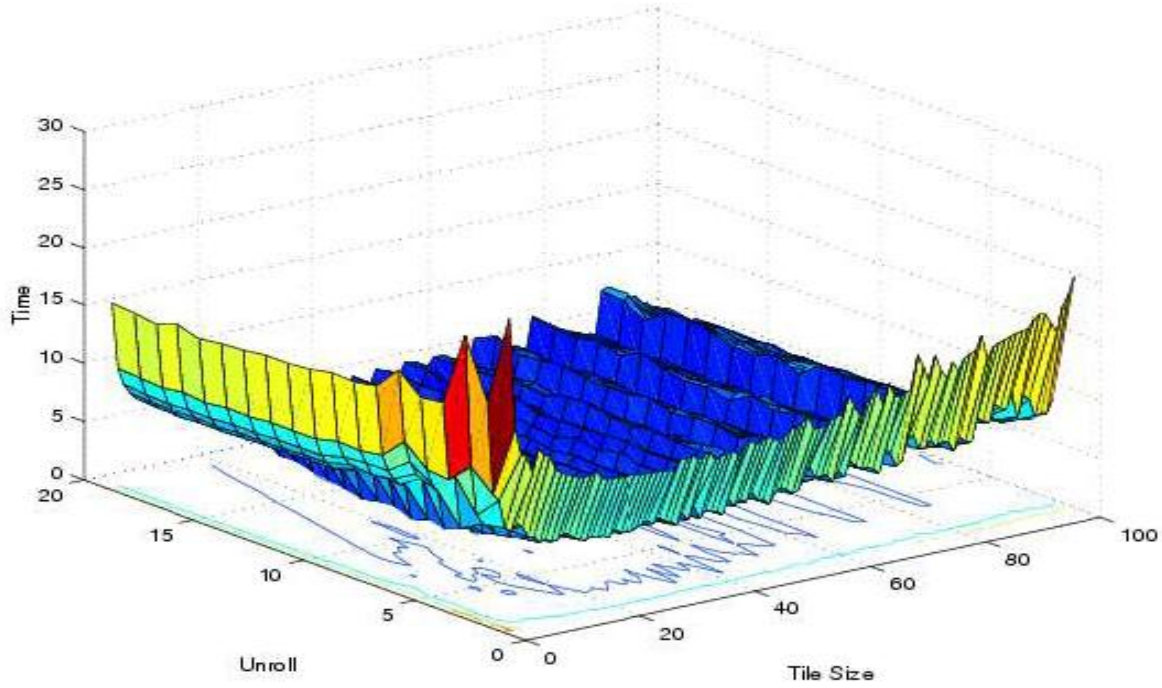
Matrix multiply kernel, 1 loop nest, 2 transformations, optimization space = 2000



Motivation: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments. Why?

Matrix multiply kernel, 1 loop nest, 2 transformations, optimization space = 2000



Simple swim benchmark from SPEC2000, multiple loop nests, 3 transformations, optimization space = 10^{52}

Motivation: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments. Why?

- Optimization spaces are large and non-linear with many local minima
- Exploration is slow and ad-hoc (random, genetic, some heuristics)
- Only part of the system is taken into account

(rarely reflect behavior of the whole system)

- Often the same (one) dataset is used
- Lack of run-time adaptation
- No optimization knowledge sharing and reuse

Motivation

Current state (acknowledged by most of the R&D roadmaps until 2020):

Developing, testing and optimizing computer systems is becoming:

- *non-systematic and highly non-trivial*
- *tedious, time consuming and error-prone*
- *inefficient and costly*

As a result:

- *slowing down innovation in science and technology*
- *enormous waste of expensive computing resources and energy*
- *considerable increase in time to market for new products*
- *low return on investment*

Motivation

Current state (acknowledged by most of the R&D roadmaps until 2020):

Developing, testing and optimizing computer systems is becoming:

- *non-systematic and highly non-trivial*
- *tedious, time consuming and error-prone*
- *inefficient and costly*

As a result:

- *slowing down innovation in science and technology*
- *enormous waste of expensive computing resources and energy*
- *considerable increase in time to market for new products*
- *low return on investment*

Current design and optimization methodology has to be dramatically revisited particularly if we want to achieve Exascale performance!

Fundamental challenges

Researchers and engineers tend to jump from one interesting technology to another and provide some quick ad-hoc solutions while fundamental problems are not solved in decades:

1) Rising complexity of computer systems:
   too many tuning dimensions and choices

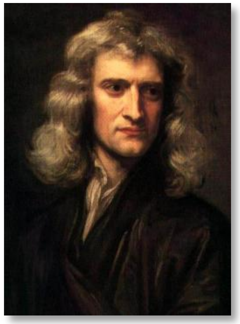
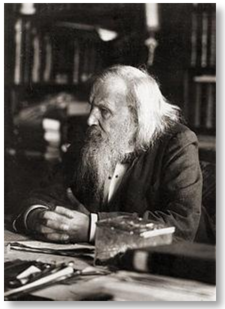
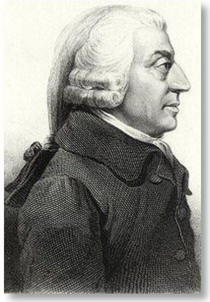
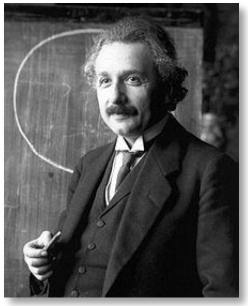
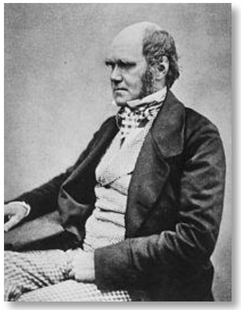
2) Performance is not anymore the only or main requirement for new computing systems: multiple objectives such as performance, power consumption, reliability, response time, etc. have to be carefully balanced :
   user objectives vs choices
   benefit vs optimization time

3) Complex relationship and interactions between ALL components at ALL levels.

4) Too many tools with non-unified interfaces changing from version to version:
   technological chaos

Long-term interdisciplinary vision

Take the best of existing sciences that deal with complex systems:
physics, mathematics, chemistry, biology, computer science, etc

What can we learn?

Long-term vision

Ambitious interdisciplinary approach:

Develop methodology and infrastructure
to systematize, simplify and automate
design, optimization and run-time adaptation of computer systems
based on empirical, analytical and statistical techniques
combined with
learning, classification and predictive modeling

Why not to make collaborative, community-based framework and repository to start sharing data and modules just like in physics, biology, etc?

Software engineering in academic research

Why not to make collaborative, community-based framework and repository to start sharing data and modules just like in physics, biology, etc?

Academic research on program and architecture design and optimization rarely focuses on software engineering.

Often considered as a waste of time!

Main focus is often to publish as many papers as possible!

Reproducibility and statistical meaningfulness of results is often not even considered! In fact, it is often impossible!

Collective Optimization Database

cTuning initiative (http://cTuning.org)

Public repository to share optimization cases:

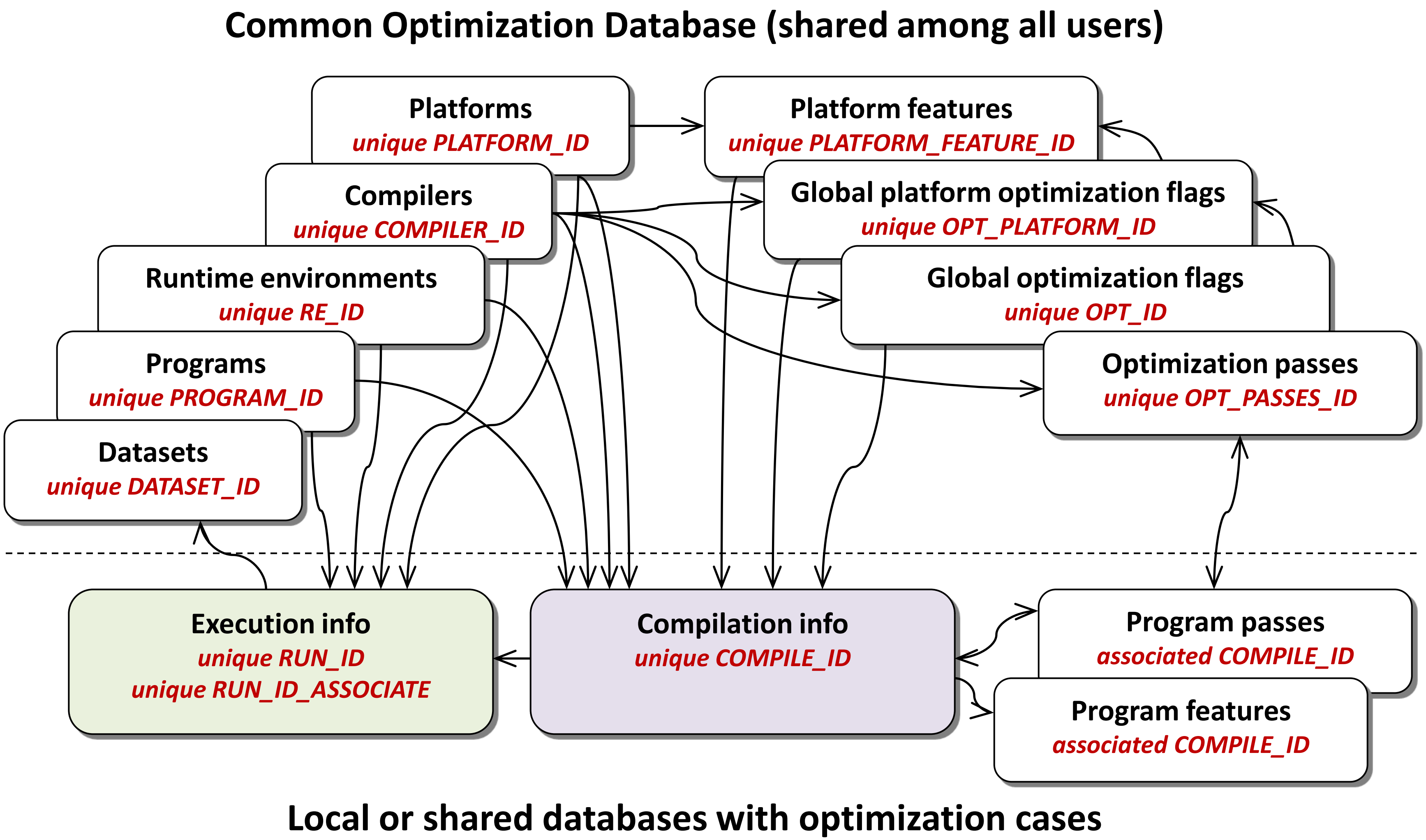
http://cTuning.org/cdatabase

•Cases include program optimizations and architecture configurations to improve execution time, code size, detect performance anomalies and bugs, etc.

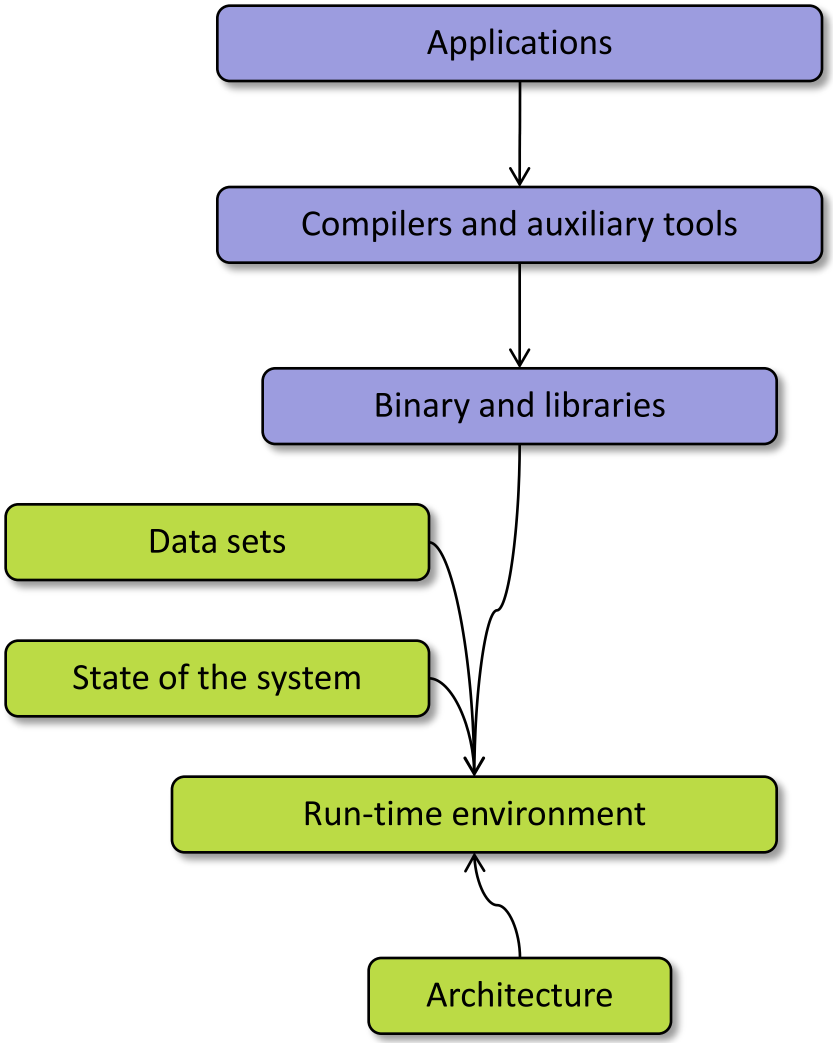
•All records have a unique UUID-based identifier to enable referencing of optimization cases and full decentralization of the infrastructure if needed.

•Optimization case consists of several compilations and executions with a baseline optimization (-O3) and some new selection of optimizations.

Collective Optimization Database



Recording information



• Provide wrappers (cTuning plugins) with standardized APIs around user tools and data to be able to record *information flow (particularly about compilation and execution)*

• Provide high-level plugins (php, java, python) and low-level plugins (C, C++, Fortran)

• Gradually expose tuning dimensions and characteristics instead of exposing everything at once to keep complexity under control!

• Add multiple collaborative benchmarks to the repository (kernels and real applications) and hundreds of datasets (cBench, MiDataSets)

Recording information

Connect all tools together through plugins with unified interfaces

Applications

Compilers and auxiliary tools

Binary and libraries

Data sets

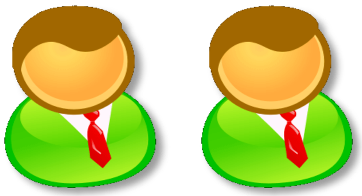
State of the system

Run-time environment

Architecture

cTuning$_1$ plugins and MySQL repository

cTuning$_2$ modules and distributed file-based repository



Command line Front End
ccc-comp <parameters>
ccc-run <parameters>

Low-level access to plugins and repository to create experiment scenarios or perform queries

Standard web-browser

High-level end-user access to repository including browsing and queries

Preparation for systematic exploration

Started collaborative exploration of optimization spaces (multiple dimensions):

- *Multiple datasets*
  - matrices of different sizes
- *Multiple compiler optimizations*
  - compiler flags
  - compiler pragmas
  - source to source transformations
- *Multiple run-time environment conditions*
  - sole execution
  - execution of multiple instances in parallel
- *Multiple architectures*
  - Intel, AMD, Longsoon, ARC, ARM with varied parameters:
  - frequency
  - cache size
- *Multiple objectives*
  - execution time, power consumption, CPI, code size, compilation time, etc

Systematic characterization and optimization methodology

Example

- Start from **ZERO** knowledge

- Select some point in the large multidimensional space for experiments

- Randomly select 1 program from the pool of available programs in some repository:

  LU decomposition from Numerical Recipes

- Randomly select 1 machine from a data center

  Microarchitecture: Intel Nehalem, 2 cores, 64-bit

  Frequency: 2.0GHz

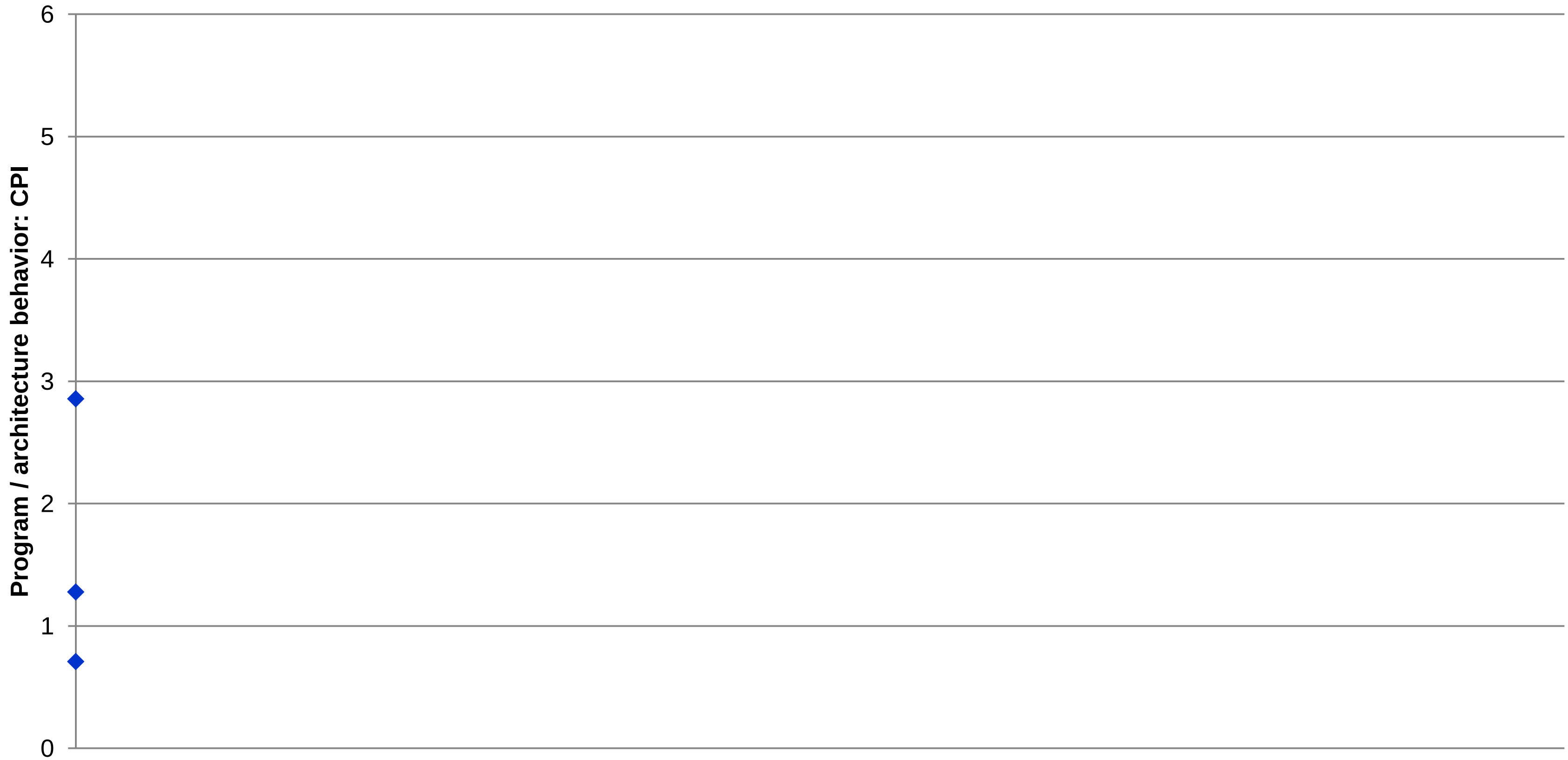
  Cache sizes: L1=64KiB, L2=512KiB, L3=4MiB

- Select compiler: **icc/ifort 12.0**
- Select optimization level: **O3**
- Select multiple datasets: **500 .. 4100**

- Observe/measure system: **execution time and CPI**

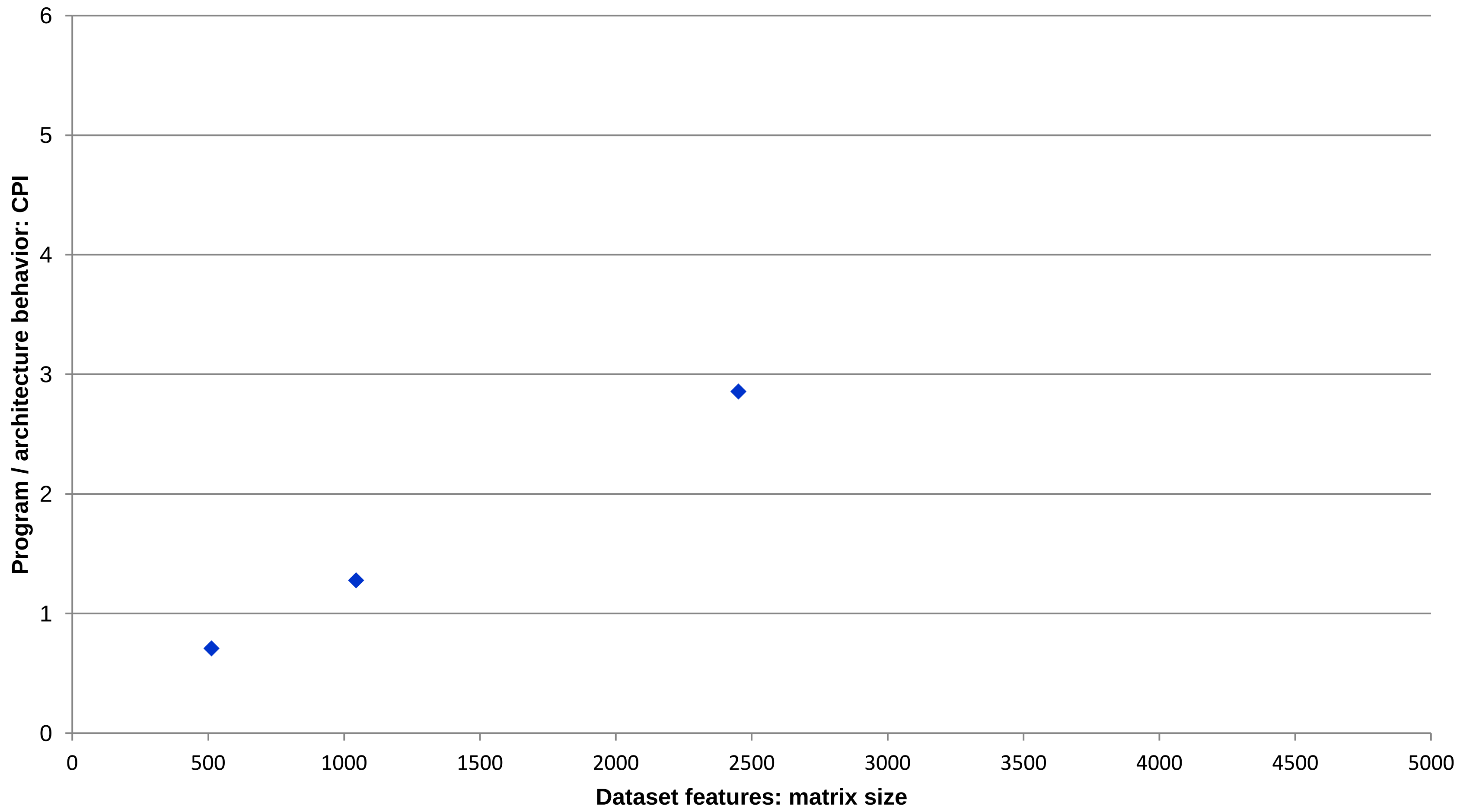
Predictive modeling

How we can explain the following results?



Predictive modeling

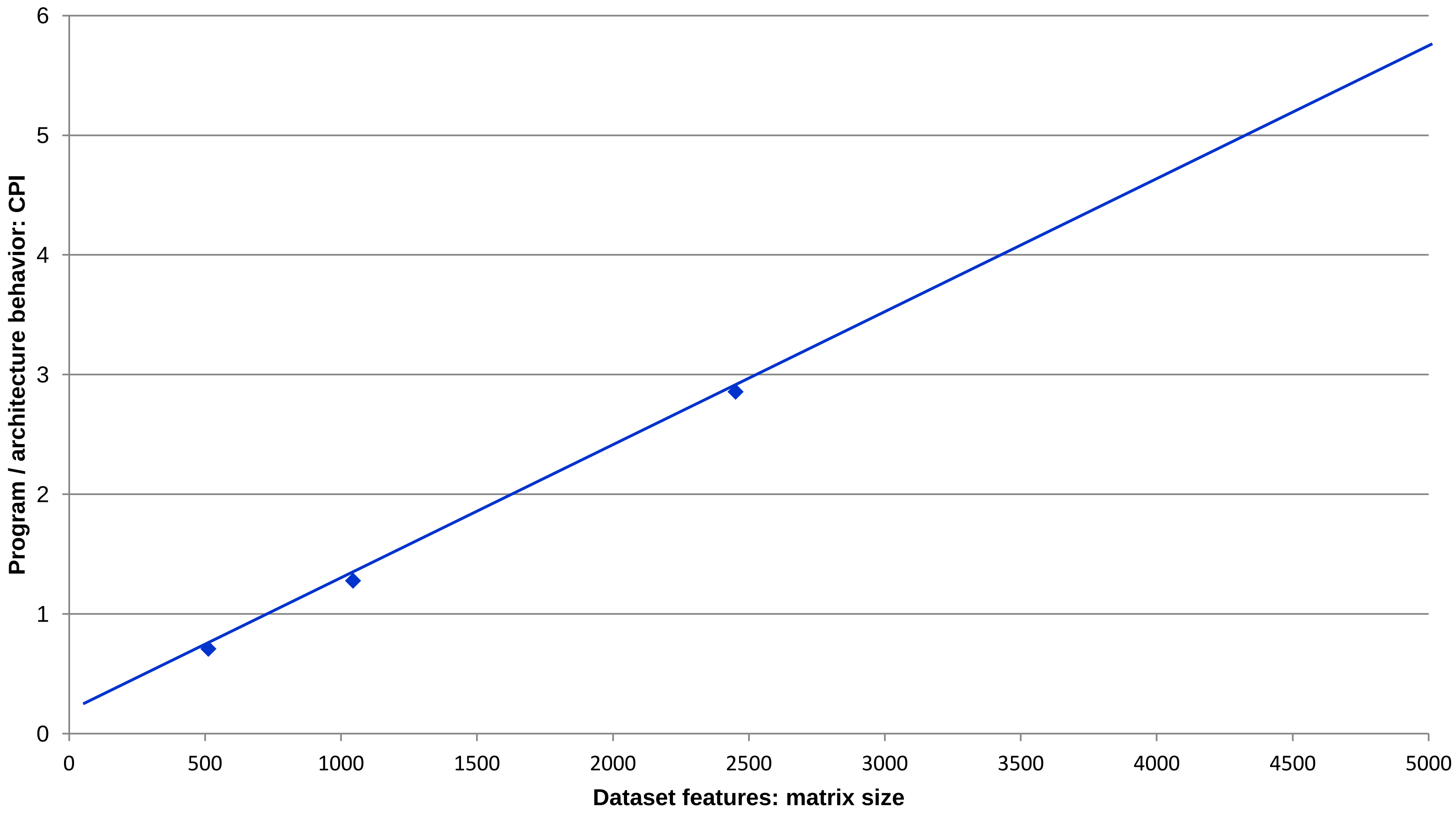
Add 1 characterization dimension: matrix size



Predictive modeling

Try to build a model to correlate objectives (CPI) and features (matrix size).

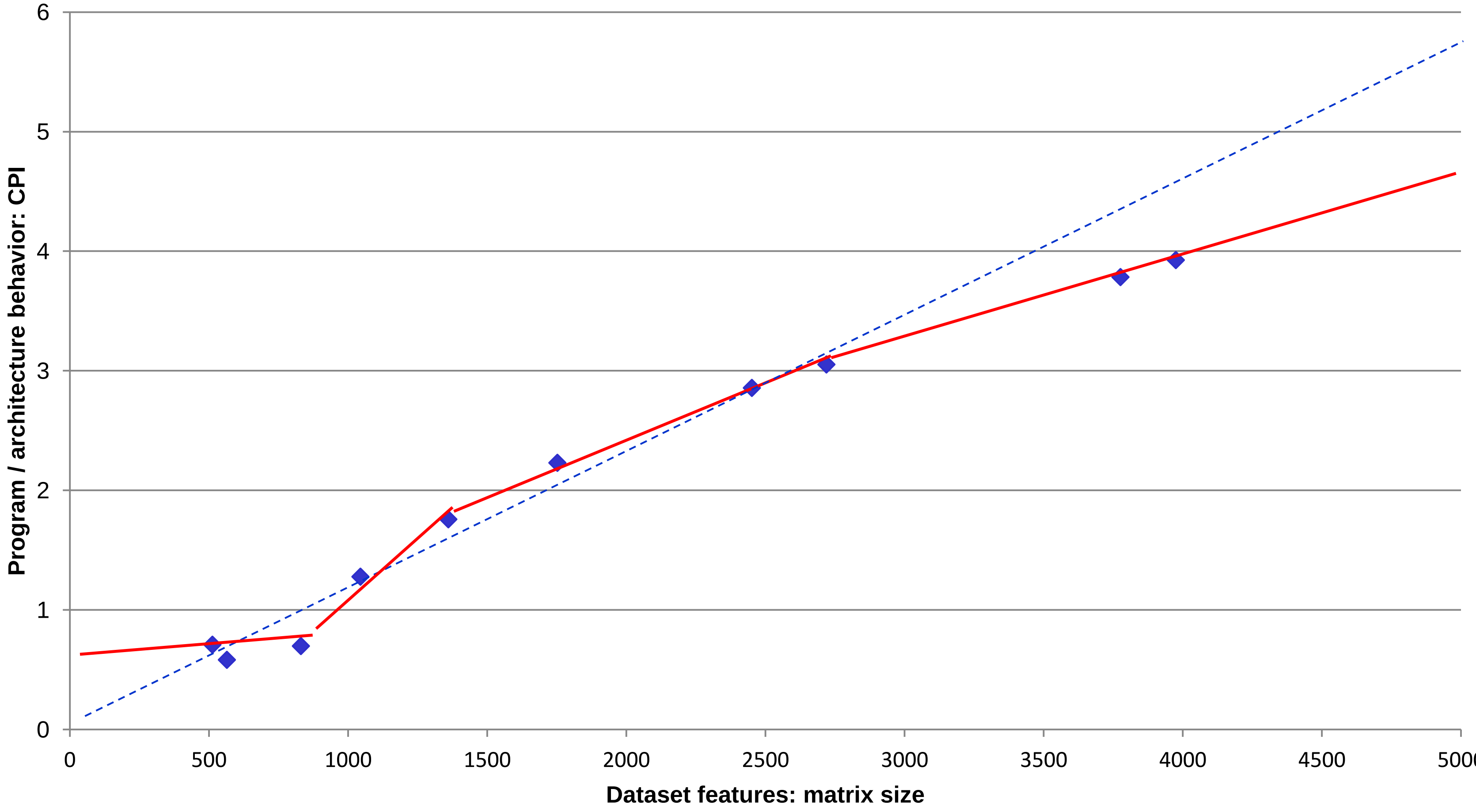
Start from simple models: linear regression (coarse grain effects)



Predictive modeling

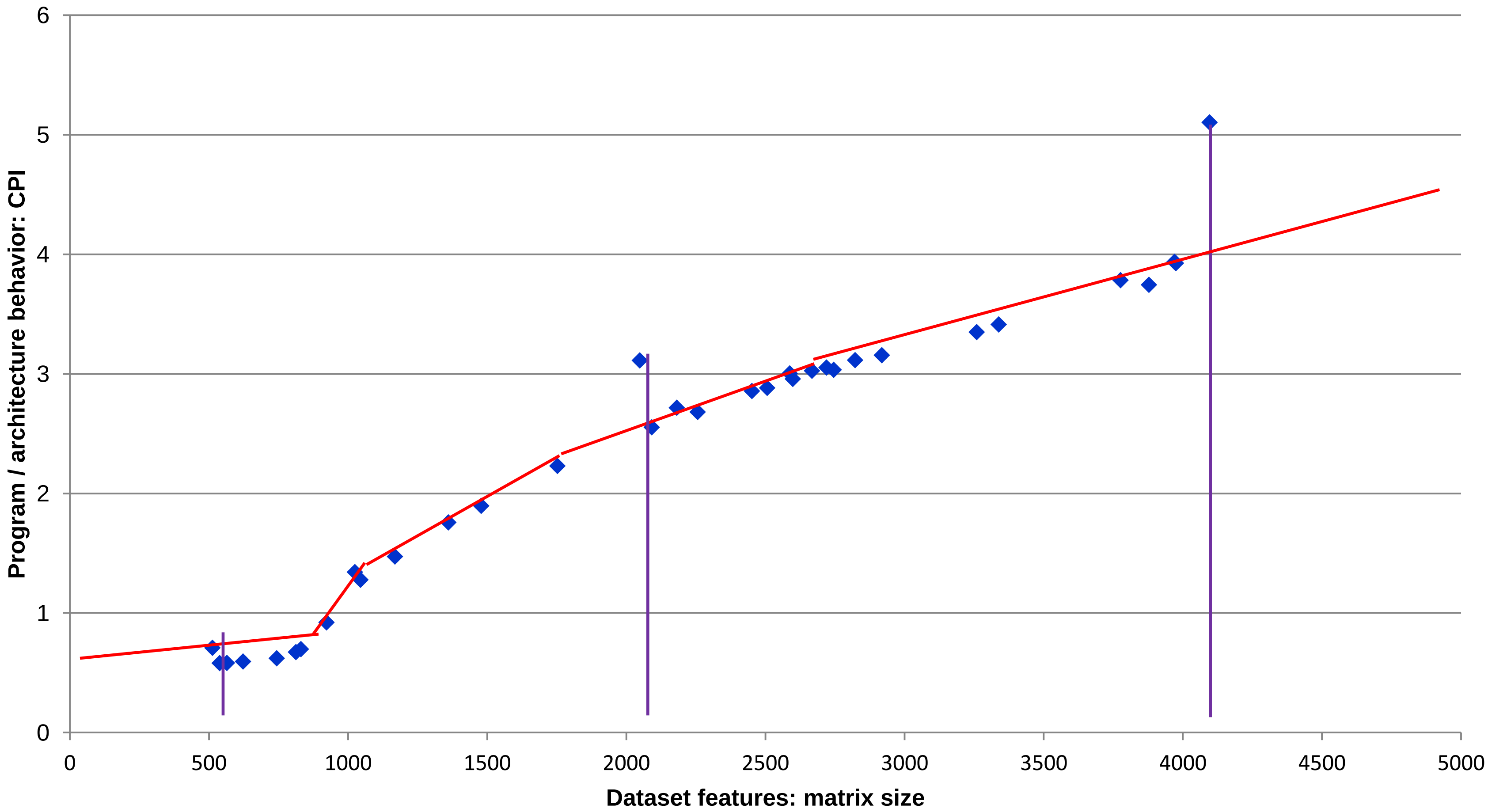
If more observations from multiple users in CTI, validate model and detect discrepancies!

Continuously retrain models to fit new data!



Predictive modeling

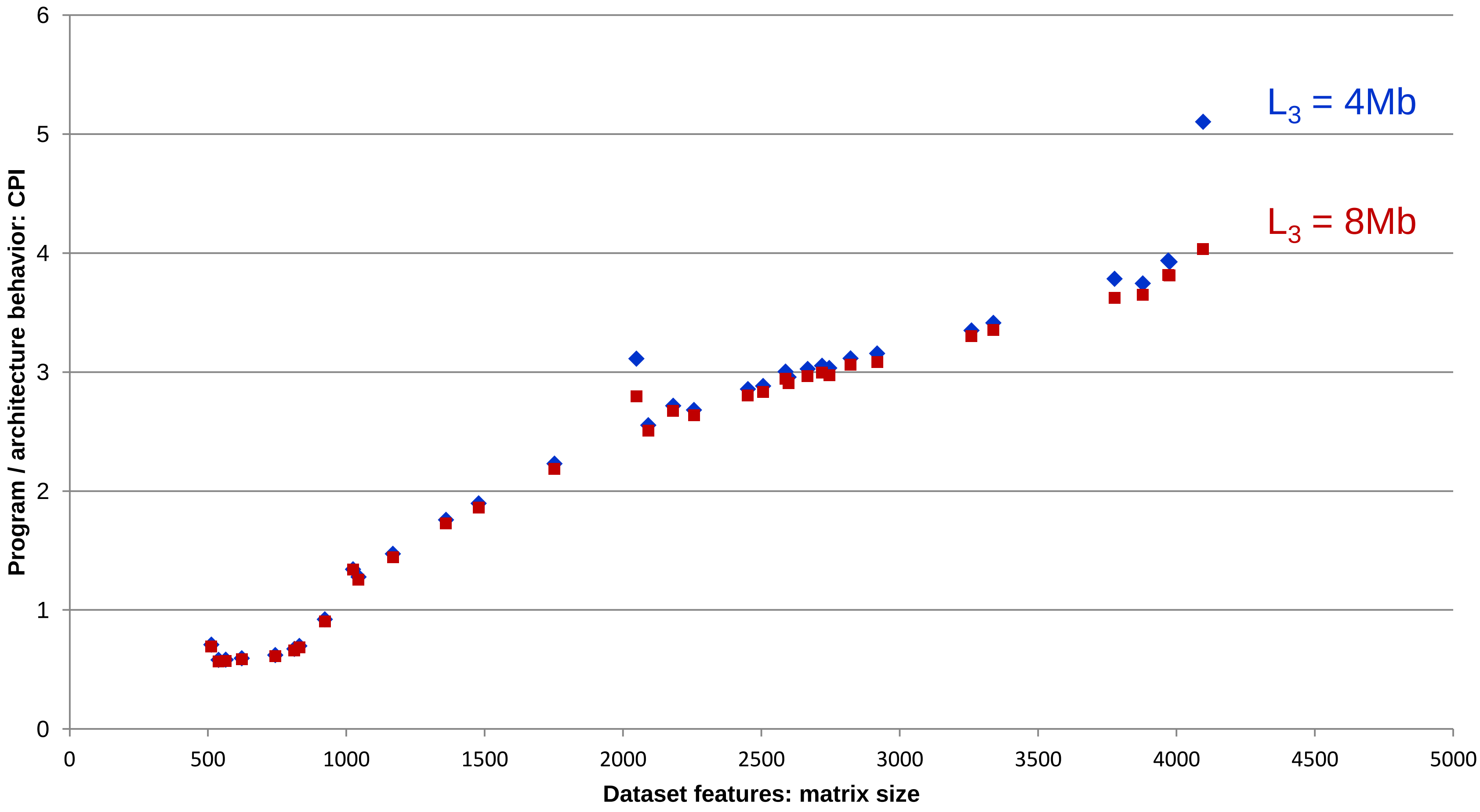
Add hierarchical modeling. For example, detect fine-grain effects (singularities) and characterize them.



Predictive modeling

Start adding more dimensions (one more architecture with twice bigger cache)!

Use automatic approach to correlate all objectives and features.

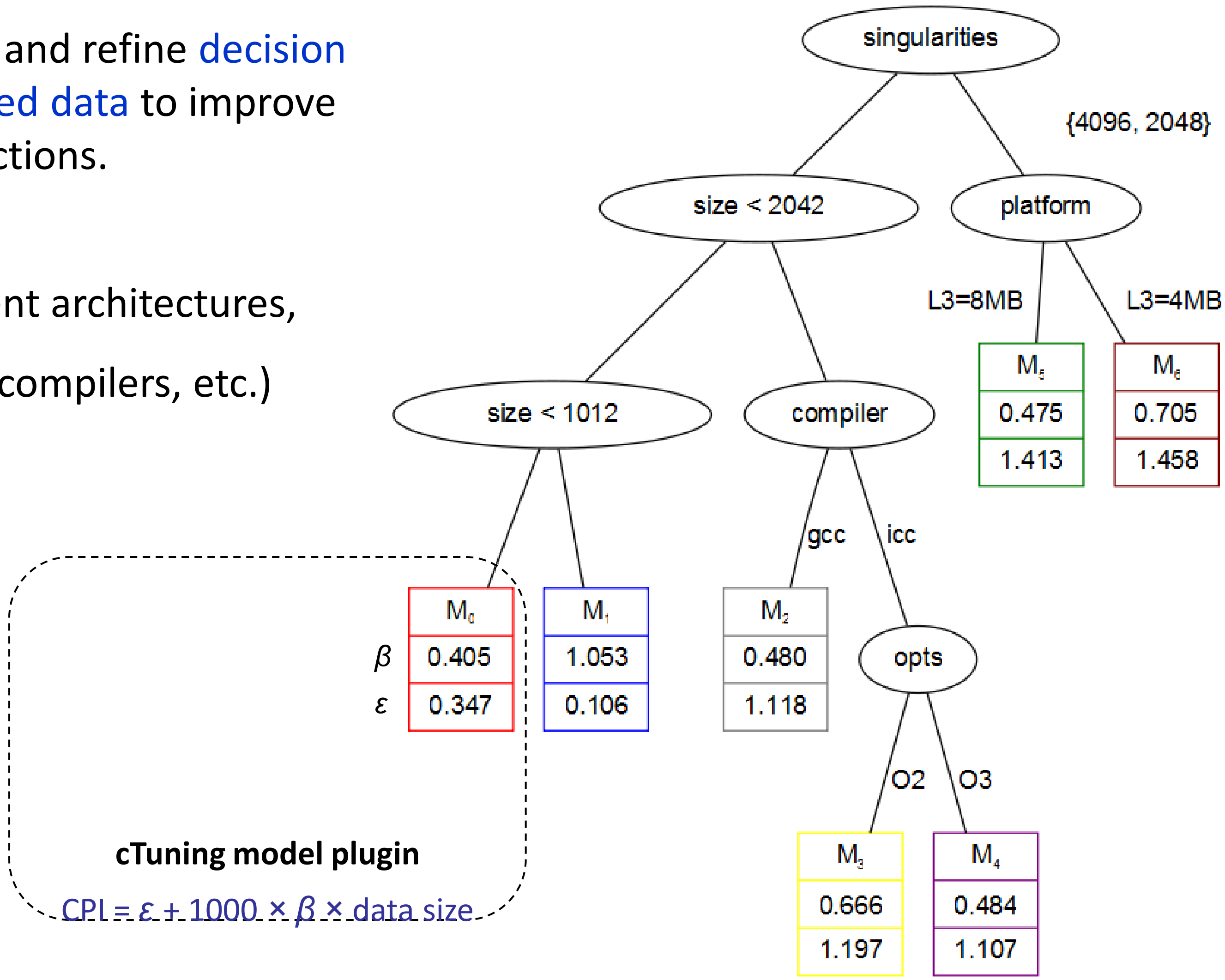


Predictive modeling

Continuously build and refine decision trees on all collected data to improve predictions.

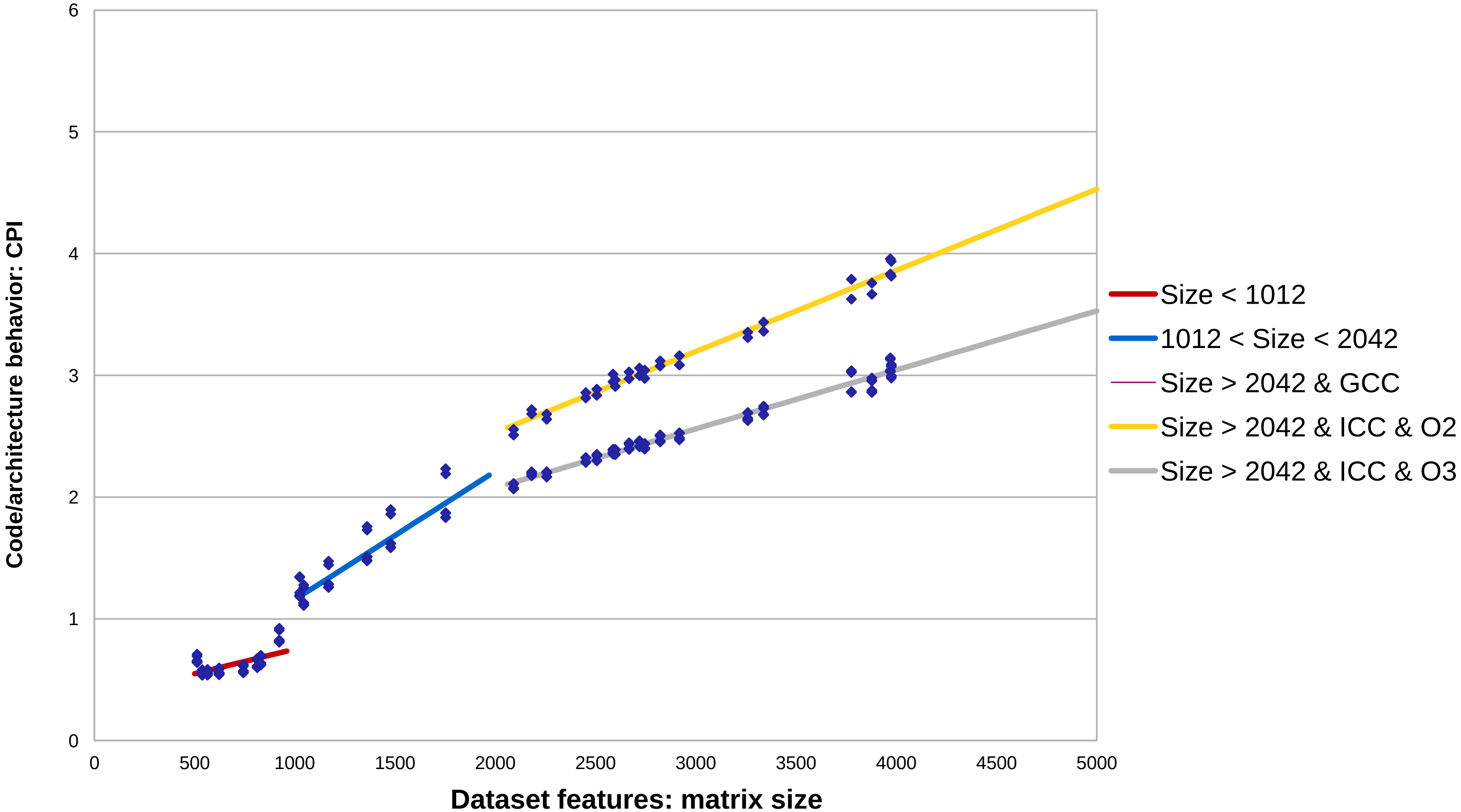
(evaluate different architectures,

optimizations, compilers, etc.)



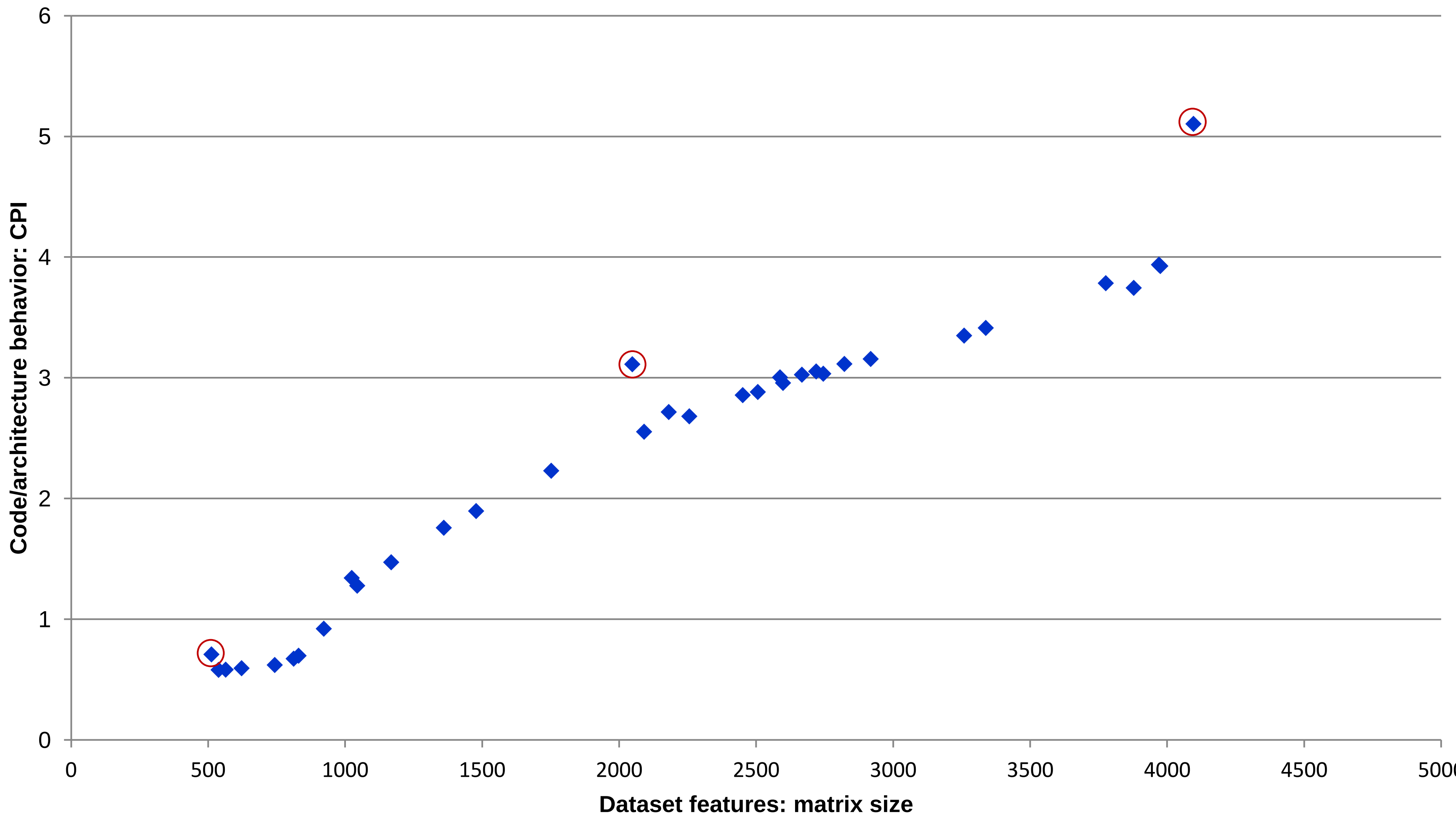
Predictive modeling

Optimize decision tree (many different algorithms)
Balance precision vs cost of modeling = ROI (coarse-grain vs fine-grain effects)
Compact data on-line before sharing with other users!



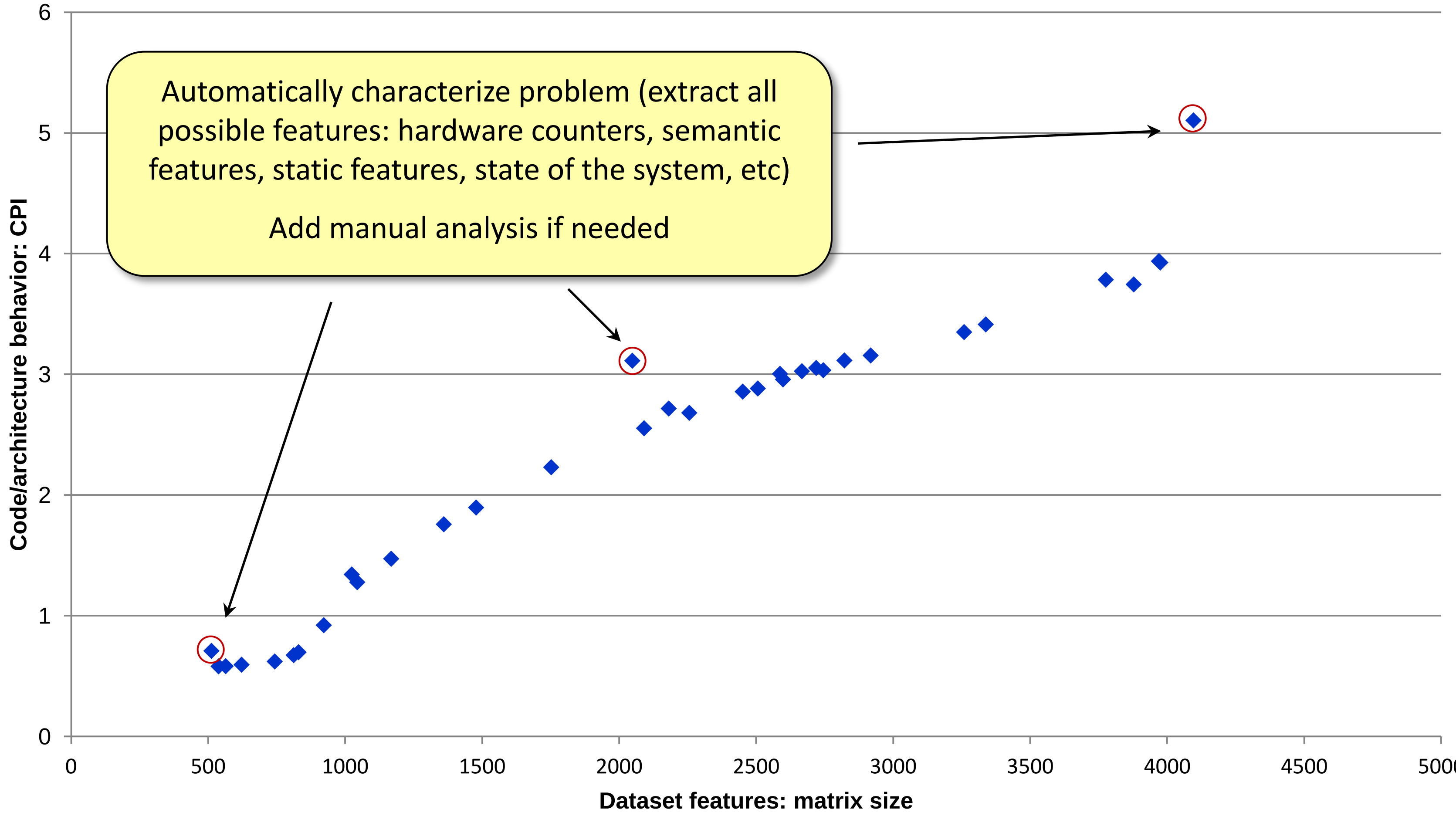
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



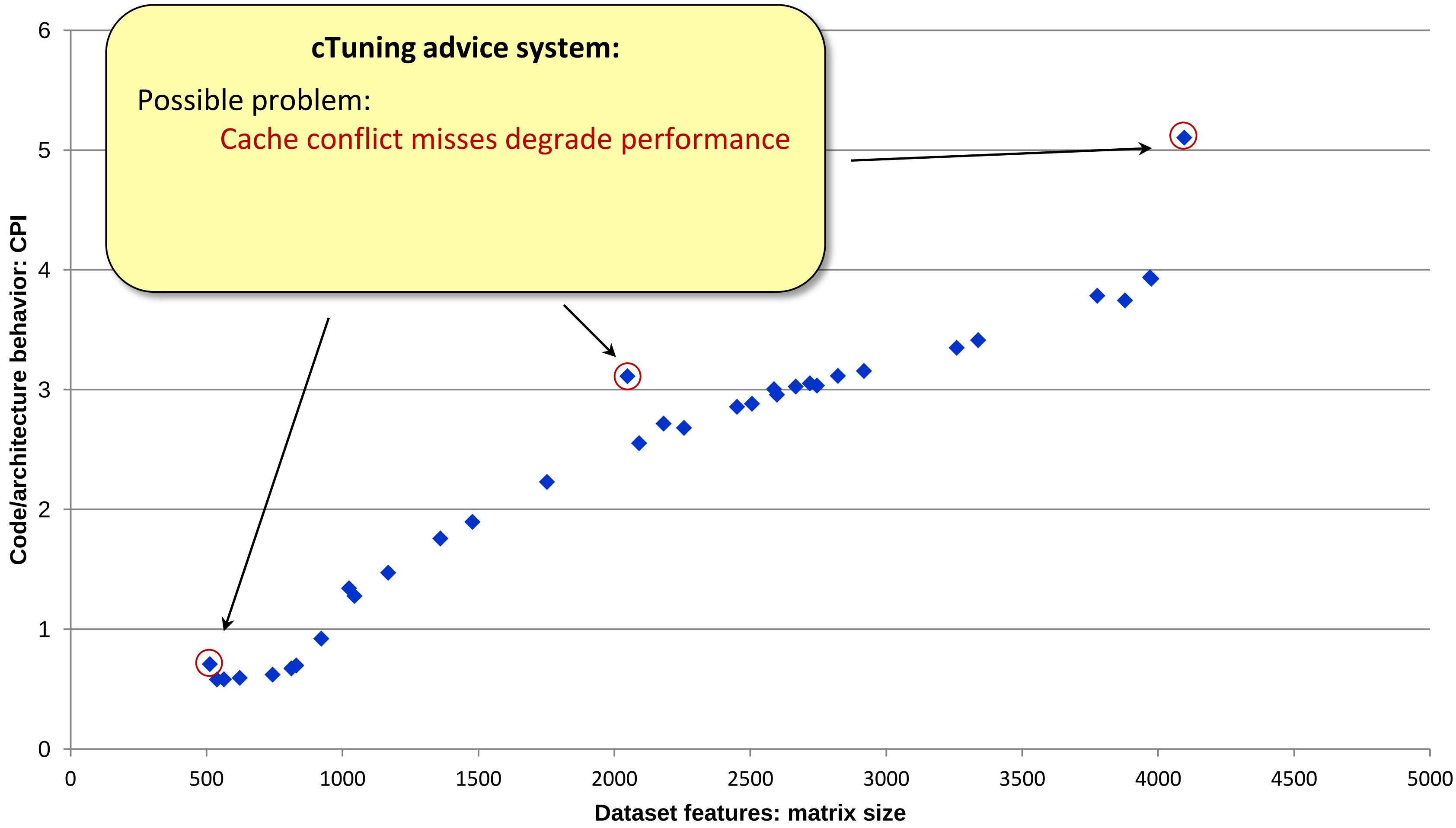
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



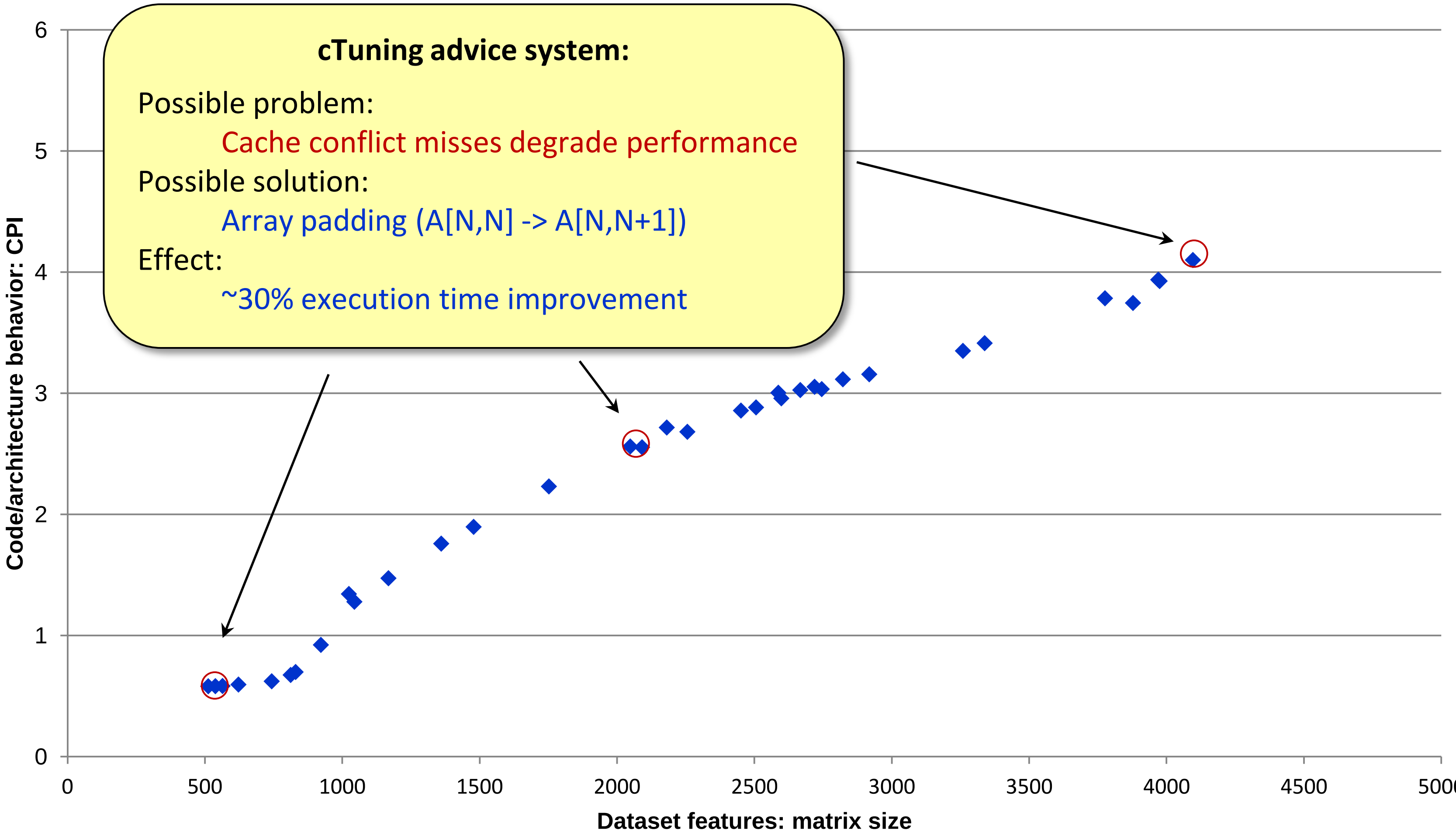
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.



Empirical multi-objective auto-tuning

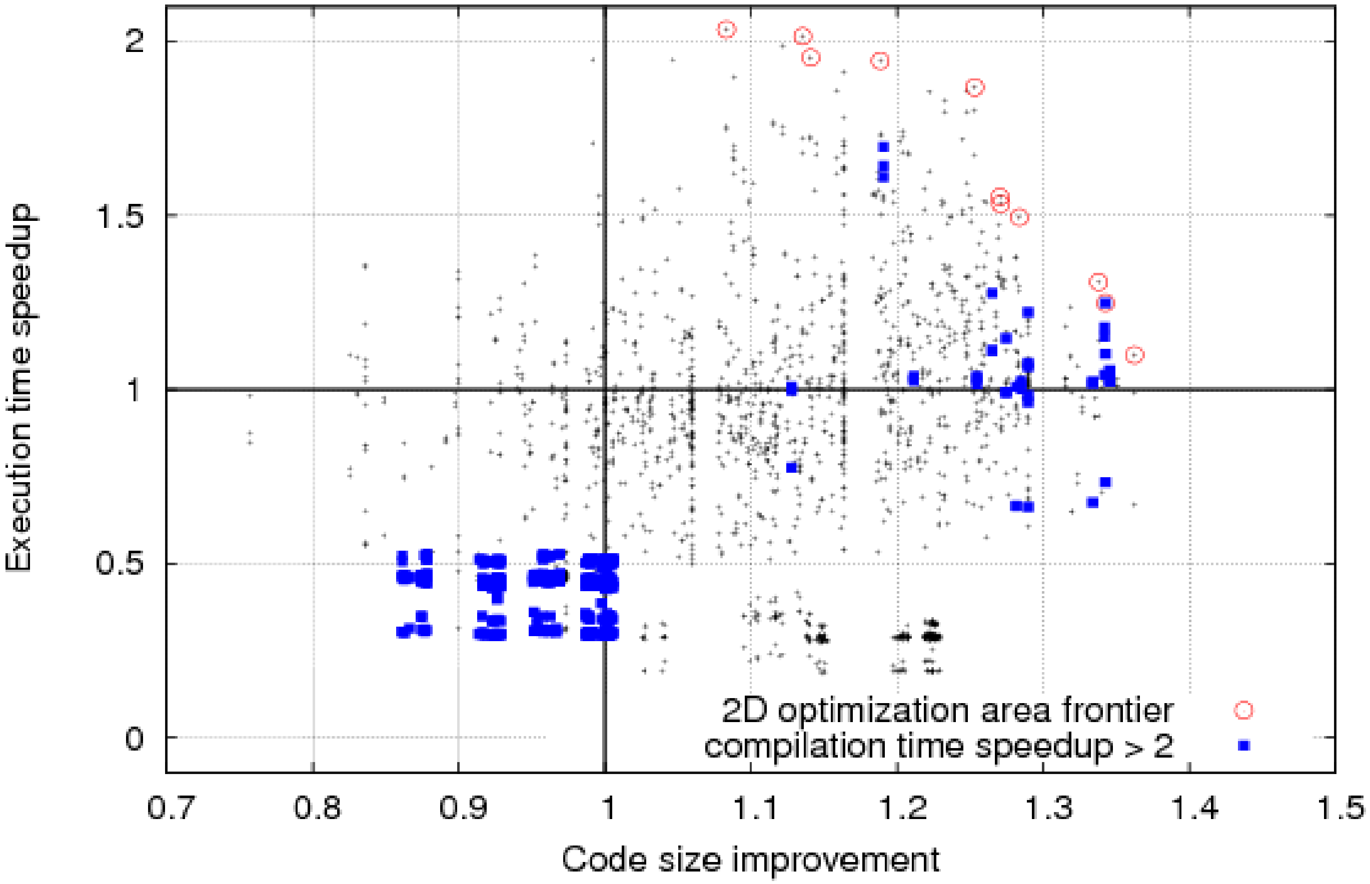
Multi-objective optimizations (depends on user scenarios):

HPC and desktops: *improving execution time*

Data centers and real-time systems: *improving execution and compilation time*

Embedded systems: *improving execution time and code size*

New additional requirement: *reduce power consumption*



susan corners kernel

Intel Core2

GCC 4.4.4
similar results on ICC 11.1

baseline opt=-O3
~100 optimizations

random combinations
(50% probability)

Nowadays used for auto-parallelization, reduction of contentions, reduction of communication costs, etc.

Share results

Share

Reproduce

Extend

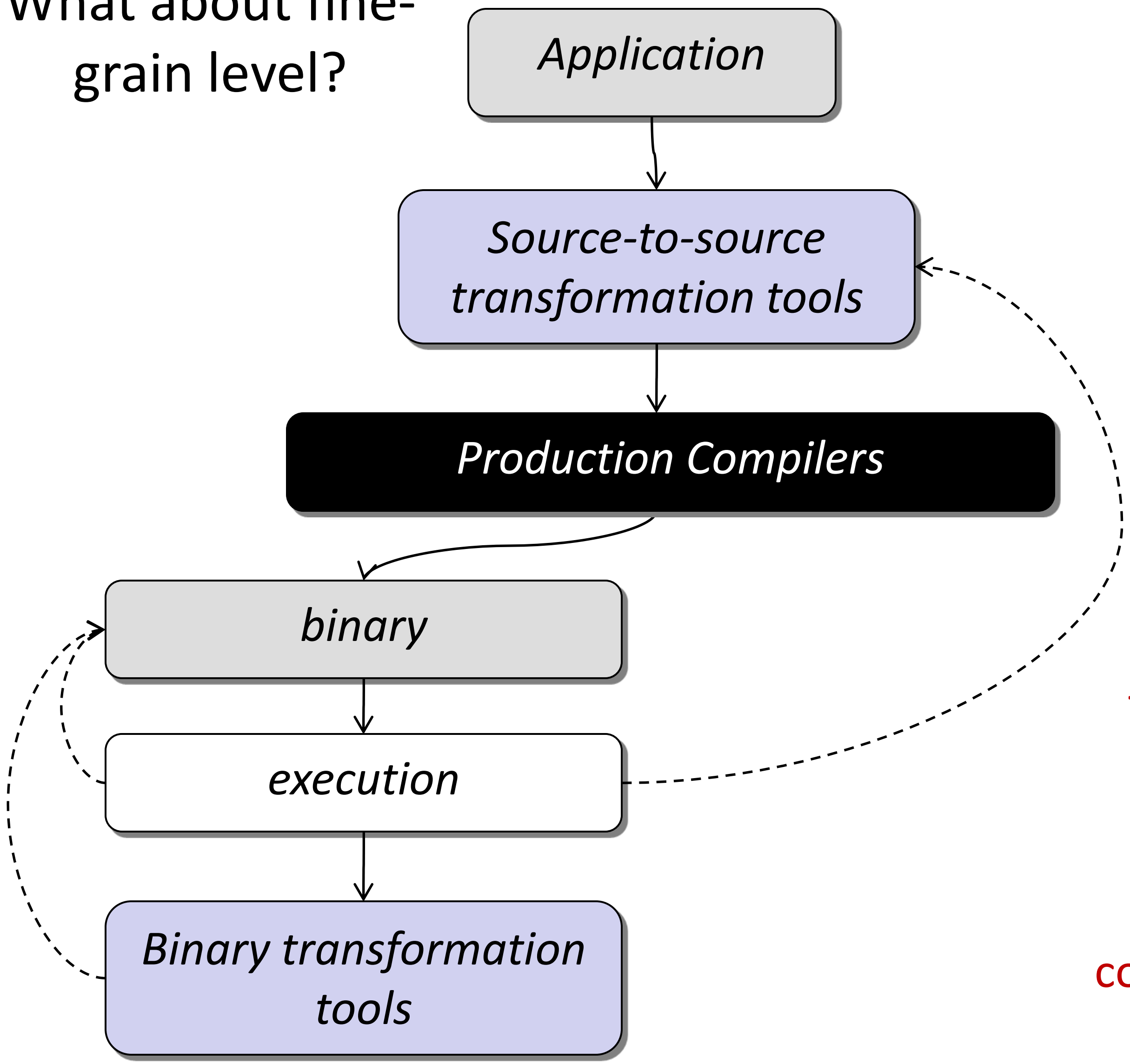
Have fun!

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

Substitute many tuning pragmas just with one that is converted into combination of optimizations:
#ctuning-opt-case 24857532370695782

Interactive compilers and tools

What about fine-grain level?

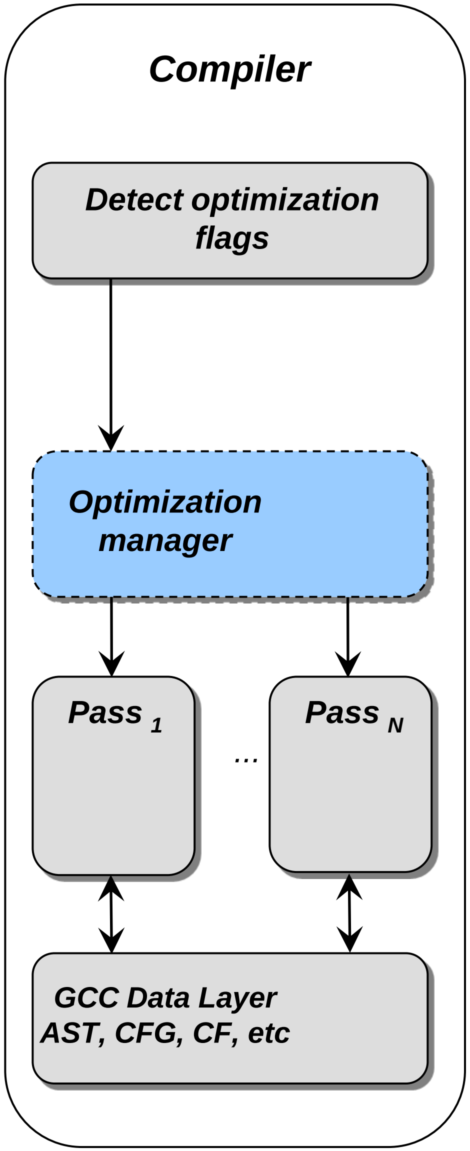


Traditional compilation, analysis and optimization

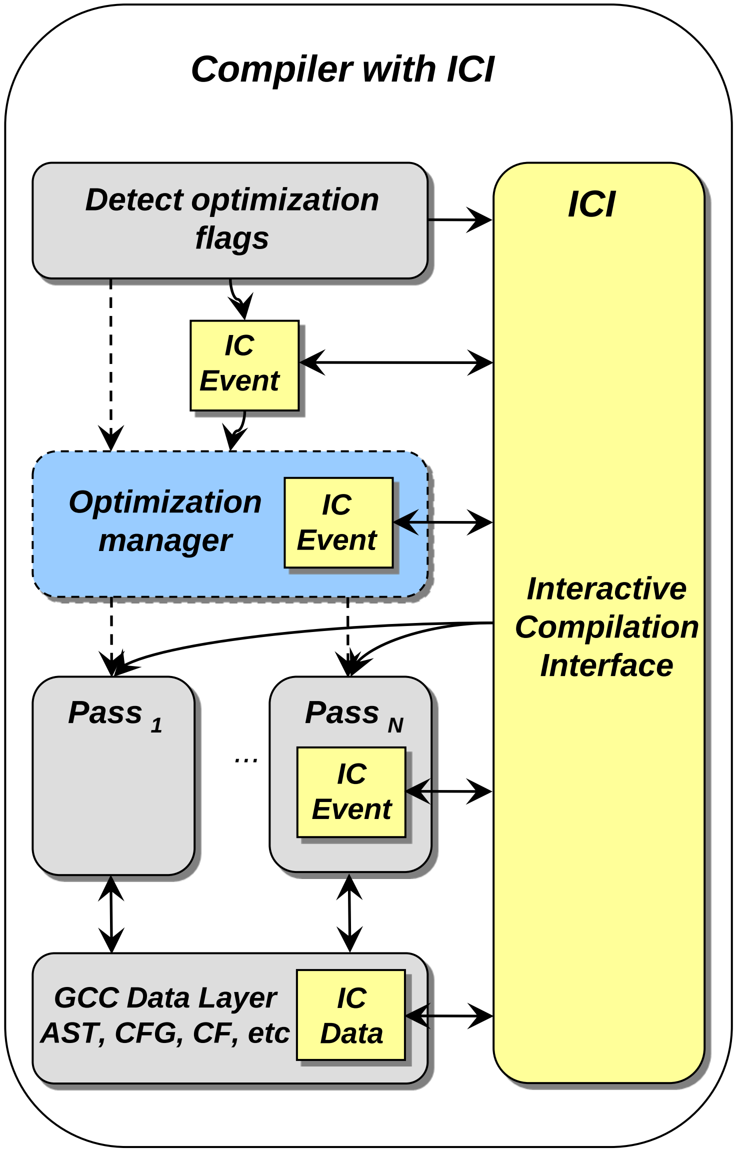
Often internal compiler decisions are not know or there is no precise control even through pragmas.

Interference with internal compiler optimizations complicates program analysis and characterization

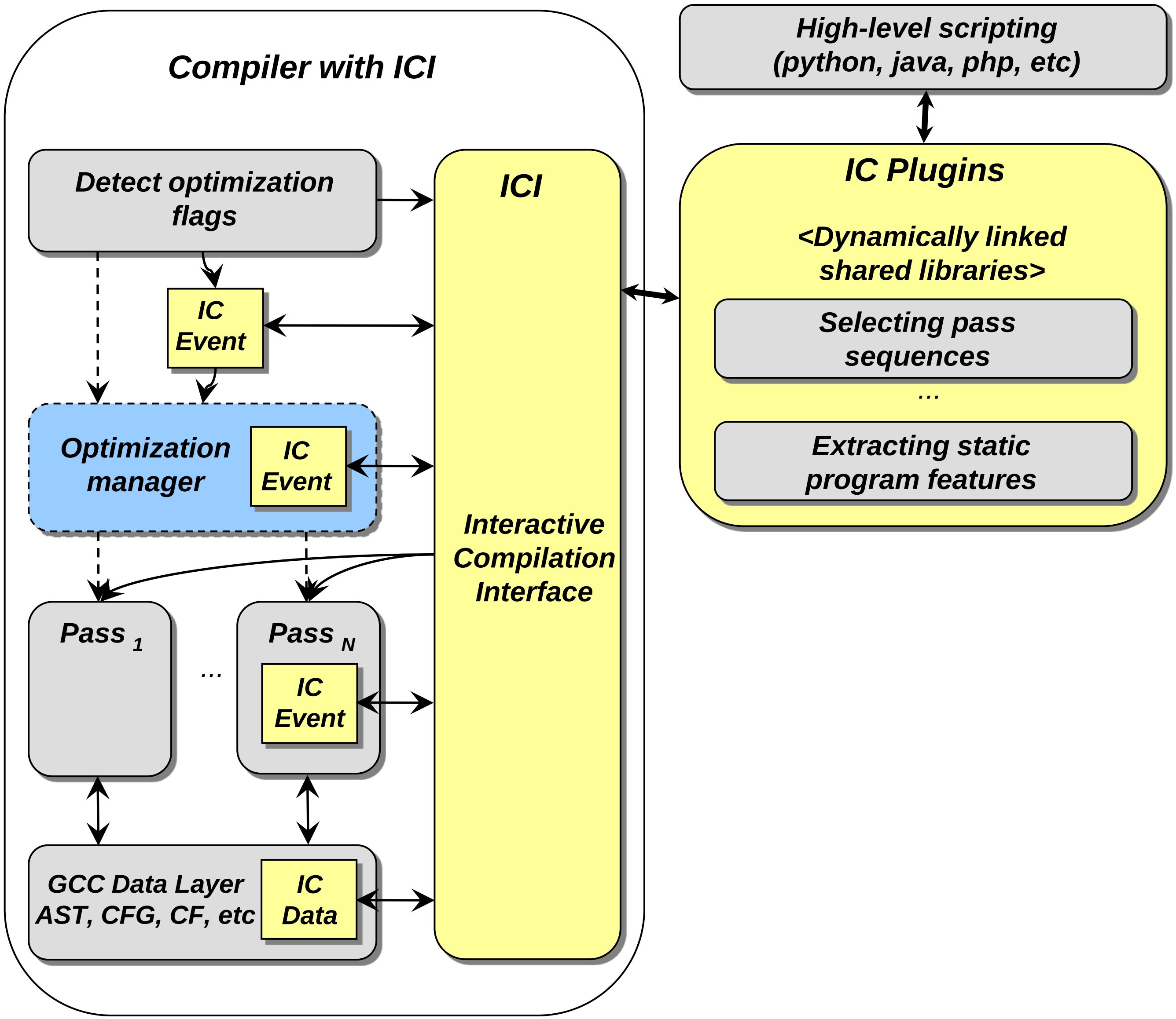
Interactive Compilation Interface (ICI)



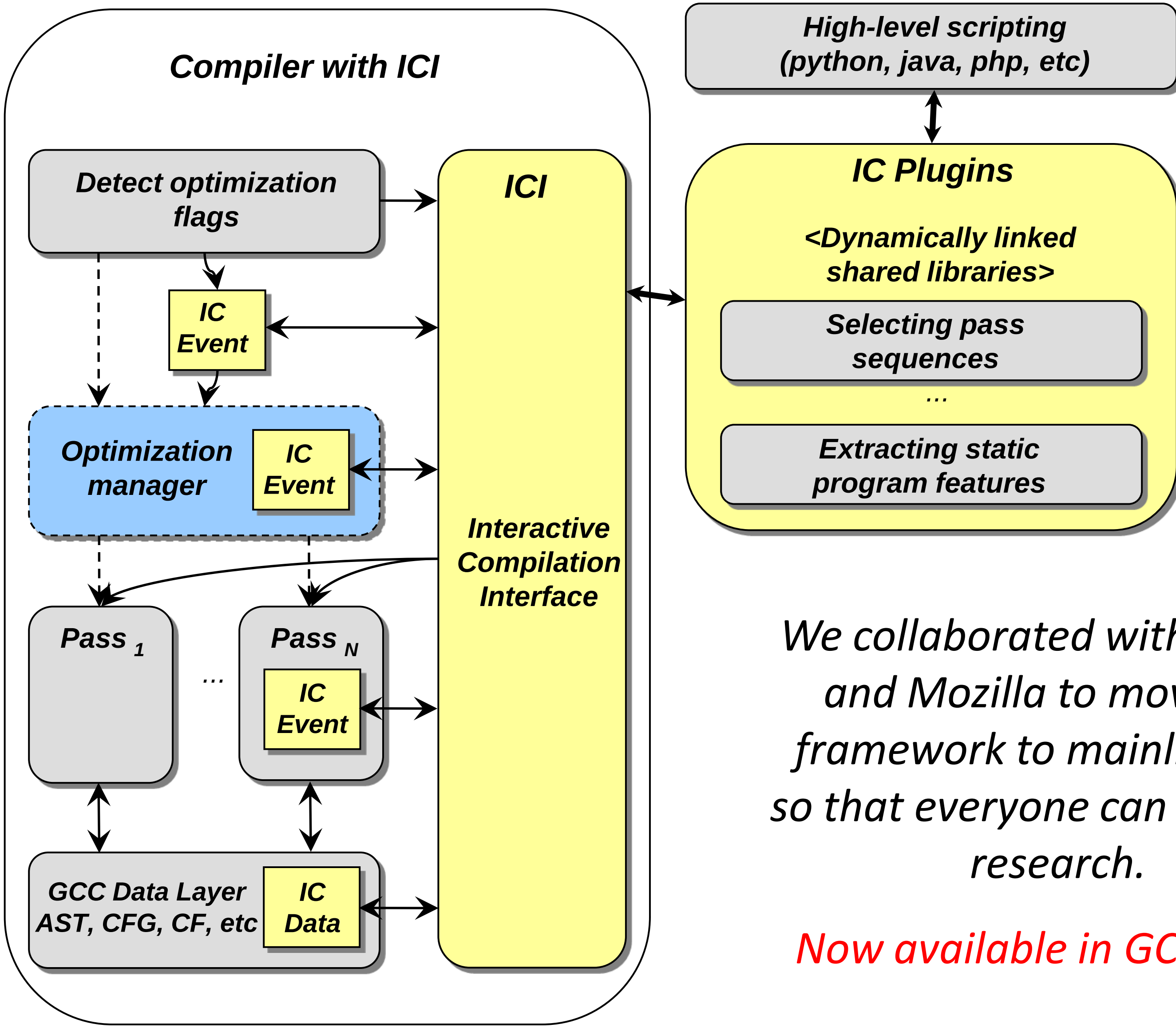
Interactive Compilation Interface (ICI)



Interactive Compilation Interface (ICI)



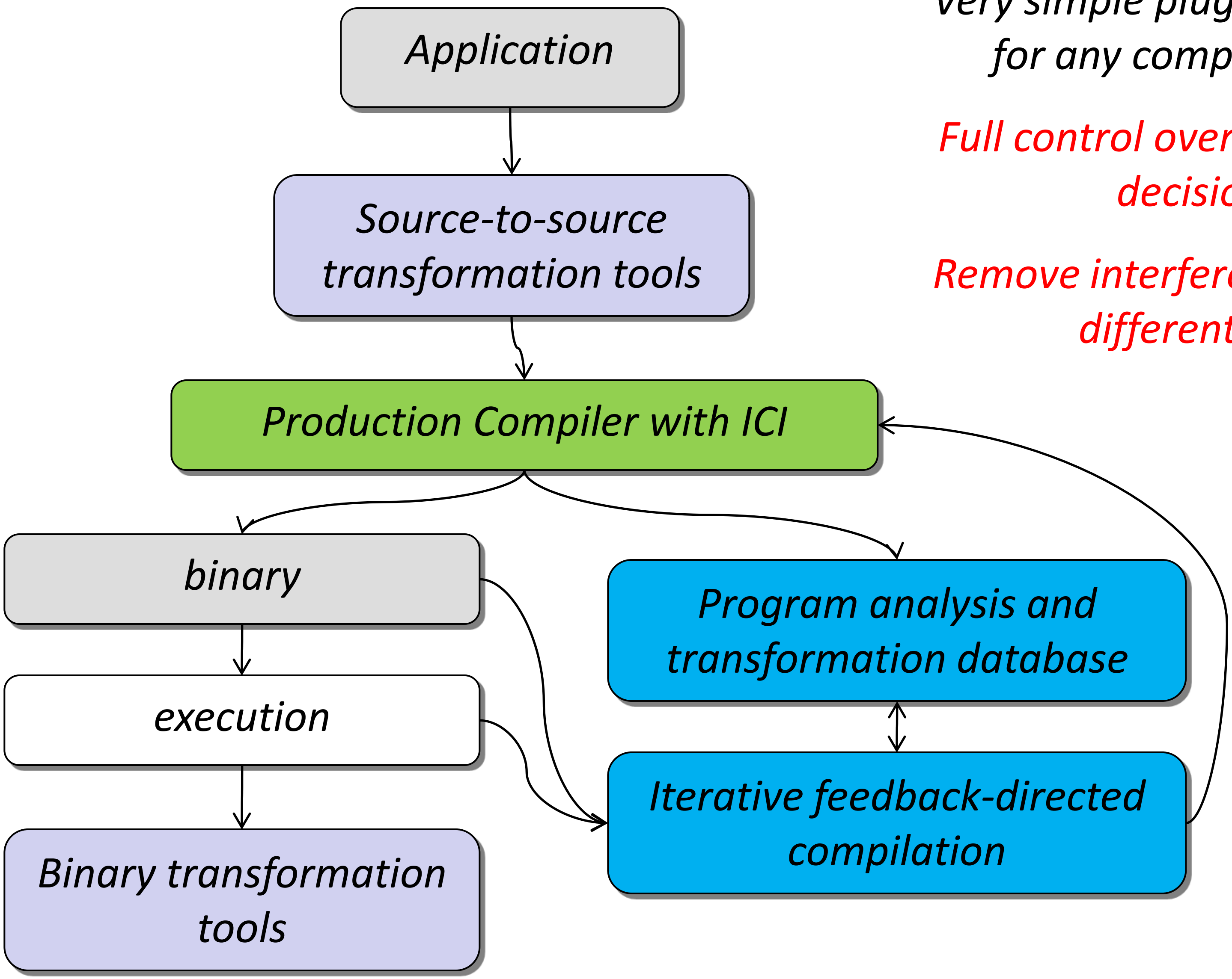
Interactive Compilation Interface (ICI)



We collaborated with Google and Mozilla to move this framework to mainline GCC so that everyone can use it for research.

Now available in GCC >=4.6

Interactive Compilation Interface (ICI)



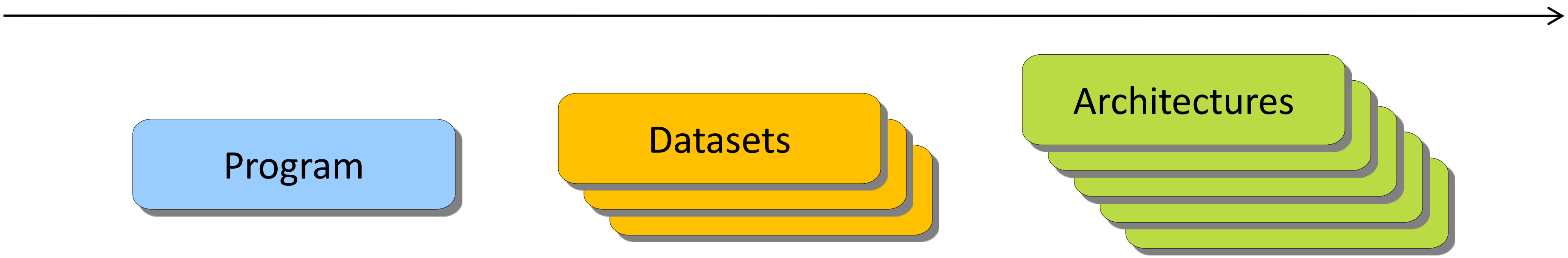
Very simple plugin framework for any compiler or tool

Full control over optimization decisions!

Remove interference between different tools

Optimization knowledge reuse across programs

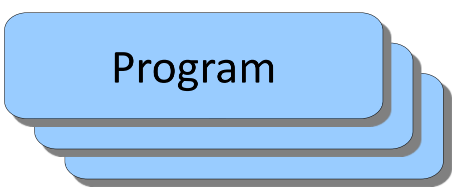
Started systematizing knowledge per program across datasets and architectures



Optimization knowledge reuse across programs

Started systematizing knowledge per program across datasets and architectures

How to reuse knowledge among programs?



Static/semantic features

Collecting data from multiple users in a unified way allows to apply various data mining (machine learning) techniques to detect relationship between the behaviour and features of all components of the computer systems

1) Add as many various features as possible (or use expert knowledge):

MILEPOST GCC with Interactive Compilation Interface:

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

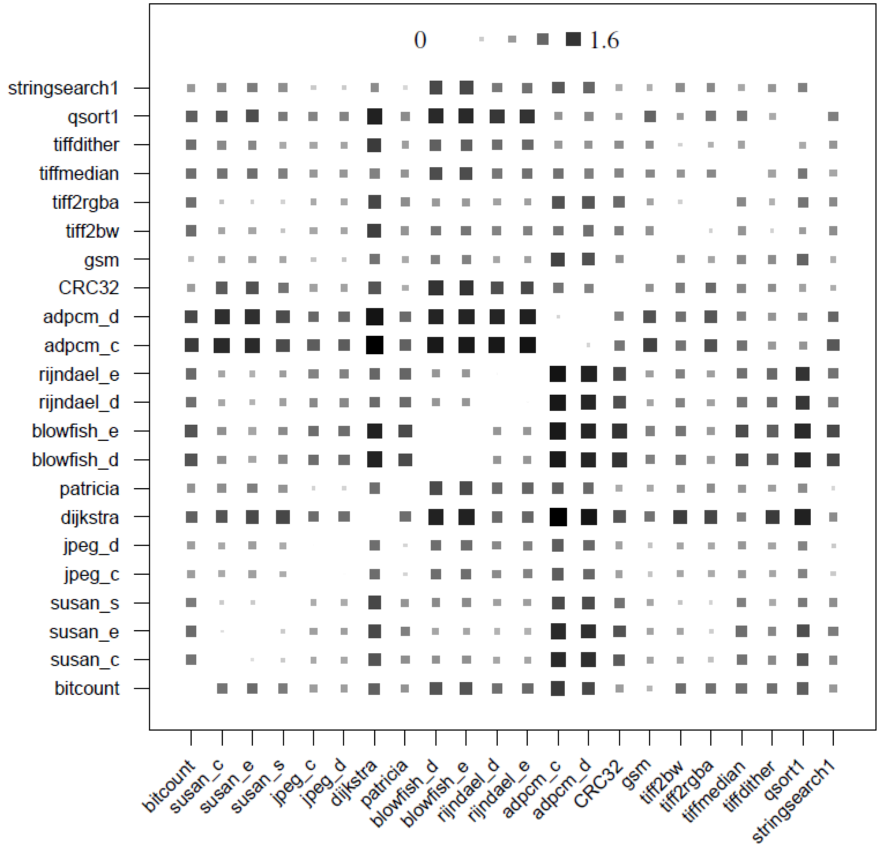
Code patterns:

```
for                     F
  for                   F
    for                 F
                  …
       load …           L
       mult …           A
       store …          S
                  …
```

2) Correlate features and objectives in cTuning using nearest neighbor classifiers, decision trees, SVM, fuzzy pattern matching, etc.

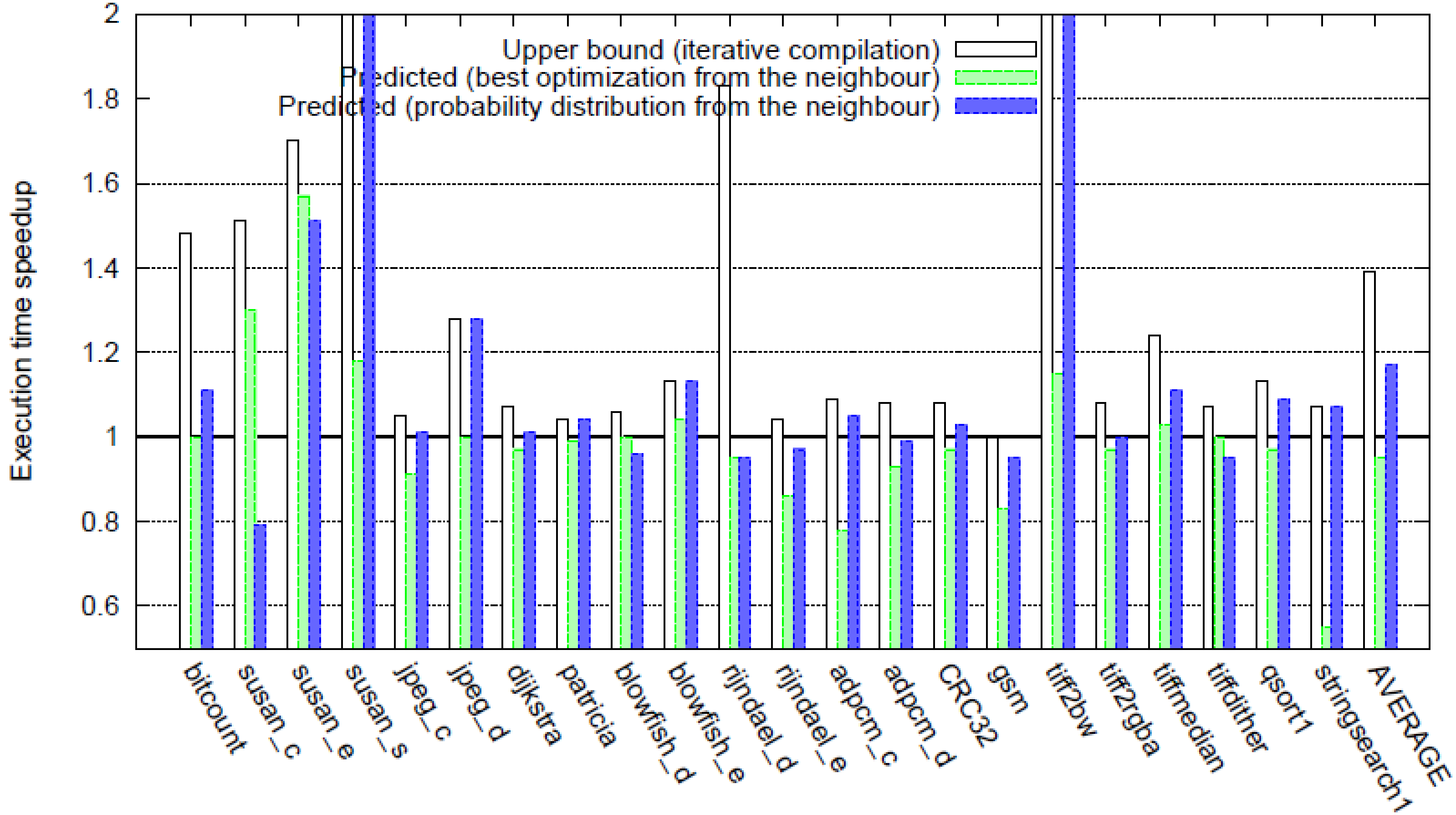
3) Given new program, dataset, architecture, predict behavior based on prior knowledge!

Nearest-neighbour classifier



Example: Euclidean distance based on static program features normalized by number of instructions

Optimization prediction (very preliminary)



Speedups achieved when using iterative compilation on Intel Xeon with random search strategy (1000 iterations; 50% probability to select each optimization), when selecting best optimization from the nearest program and when predicting optimization using probabilistic ML model based on program features.

Dynamic features

Static/semantic features are often not enough to characterize dynamic behavior!

Use dynamic features (more characterizing dimensions)!

“Traditional” features:

performance counters (difficult to interpret, change from architecture to architecture though fine for learning per architecture).

Reactions to code changes:

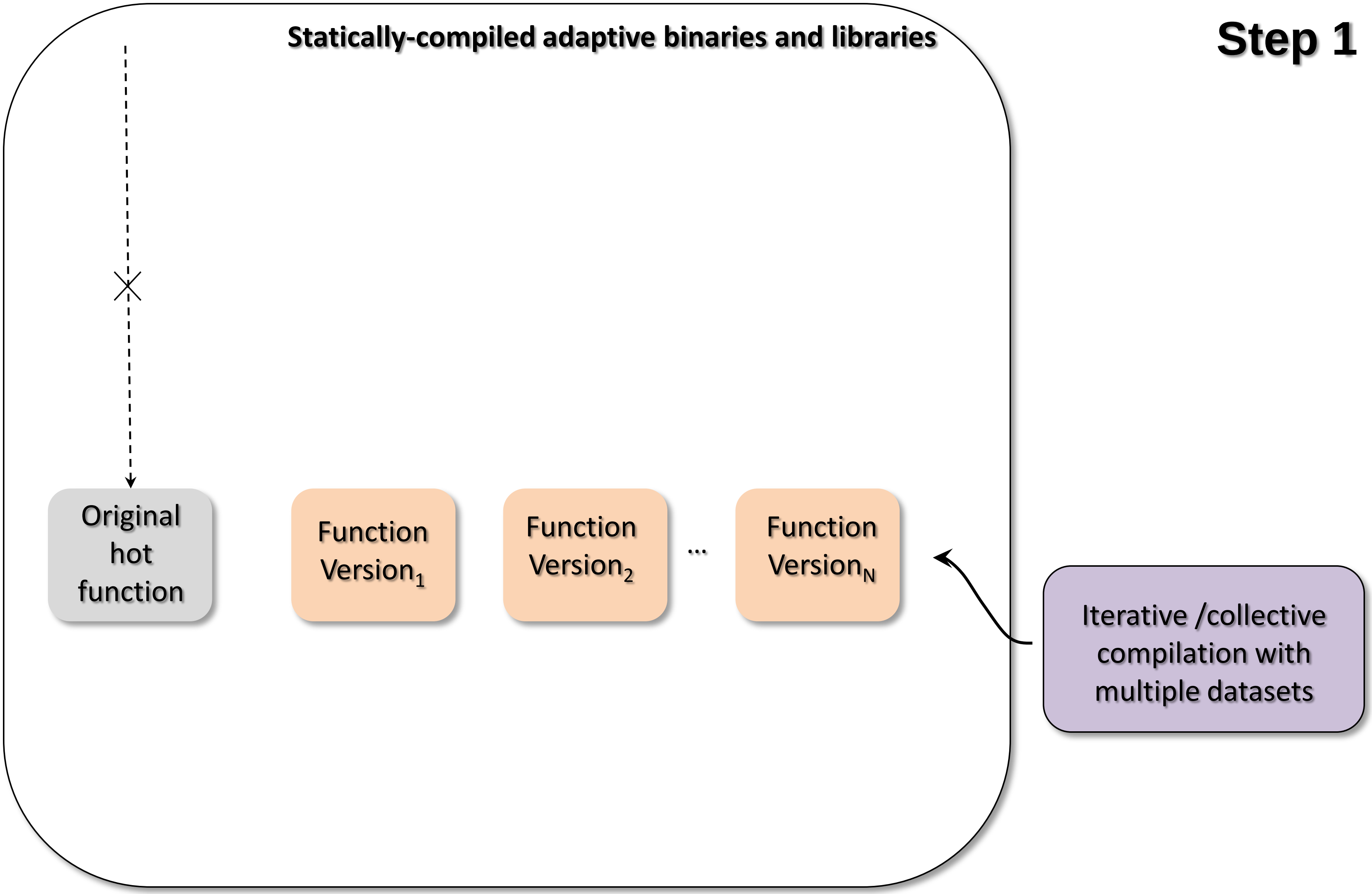
perform changes and observe program reactions (change in execution time, power, etc).

Apply optimizations (compiler flags, pragmas, manual code/data partitioning, etc).

Change/break semantics (remove or add individual instructions(data accesses, arithmetic, etc) or threads, etc and observe reactions to such changes).

Static multiversioning framework for dynamic optimizations

Step 1



Static multiversioning framework for dynamic optimizations

Step 2

Statically-compiled adaptive binaries and libraries

Original hot function

Function Version$_1$

Function Version$_2$

…

Function Version$_N$

Iterative /collective compilation with multiple datasets

Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

Static multiversioning framework for dynamic optimizations

Step 3

Statically-compiled adaptive binaries and libraries

Extract dataset features

Selection mechanism optimized for low run-time overhead

Original hot function

Function $Version_1$

Function $Version_2$

…

Function $Version_N$

Machine learning techniques to find mapping between different run-time contexts and representative versions

Iterative /collective compilation with multiple datasets

Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

Static multiversioning framework for dynamic optimizations

Dynamic

Statically-compiled adaptive binaries and libraries

Extract dataset features

Monitor run-time behavior or architectural changes (in virtual, reconfigurable or heterogeneous environments) using timers or performance counters

Selection mechanism optimized for low run-time overhead

Original hot function

Function Version_1

Function Version_2

…

Function Version_N

Machine learning techniques to find mapping between different run-time contexts and representative versions

Iterative /collective compilation with multiple datasets

Representative set of versions for the following optimization cases to minimize execution time, power consumption and code-size across all available datasets:

- optimizations for different datasets
- optimizations/compilation for different architectures (heterogeneous or reconfigurable processors with different ISA such as GPGPU, CELL, etc or the same ISA with extensions such as 3dnow, SSE, etc or virtual environments)
- optimizations for different program phases or different run-time environment behavior

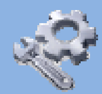
http://ctuning.org/unidapt

cTuning: Collaborative tuning infrastructure and repository

http://cTuning.org

community-based development and optimization of computing systems

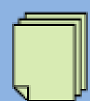
http://cTuning.org/ctools



Collaborative R&D Tools with common APIs:

GCC ICI, CCC, UNIDAPT

Continuous (transparent) monitoring of computing systems

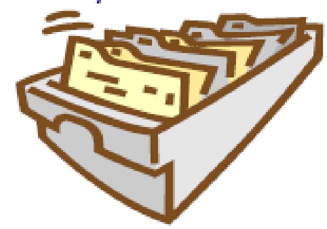


Collective Benchmarks (from the community)

cBench

http://cTuning.org/cdatabase

Statistical analysis and machine learning to systematize and reuse optimization knowledge and predict good optimizations



Collective Optimization Database with optimization cases from the community

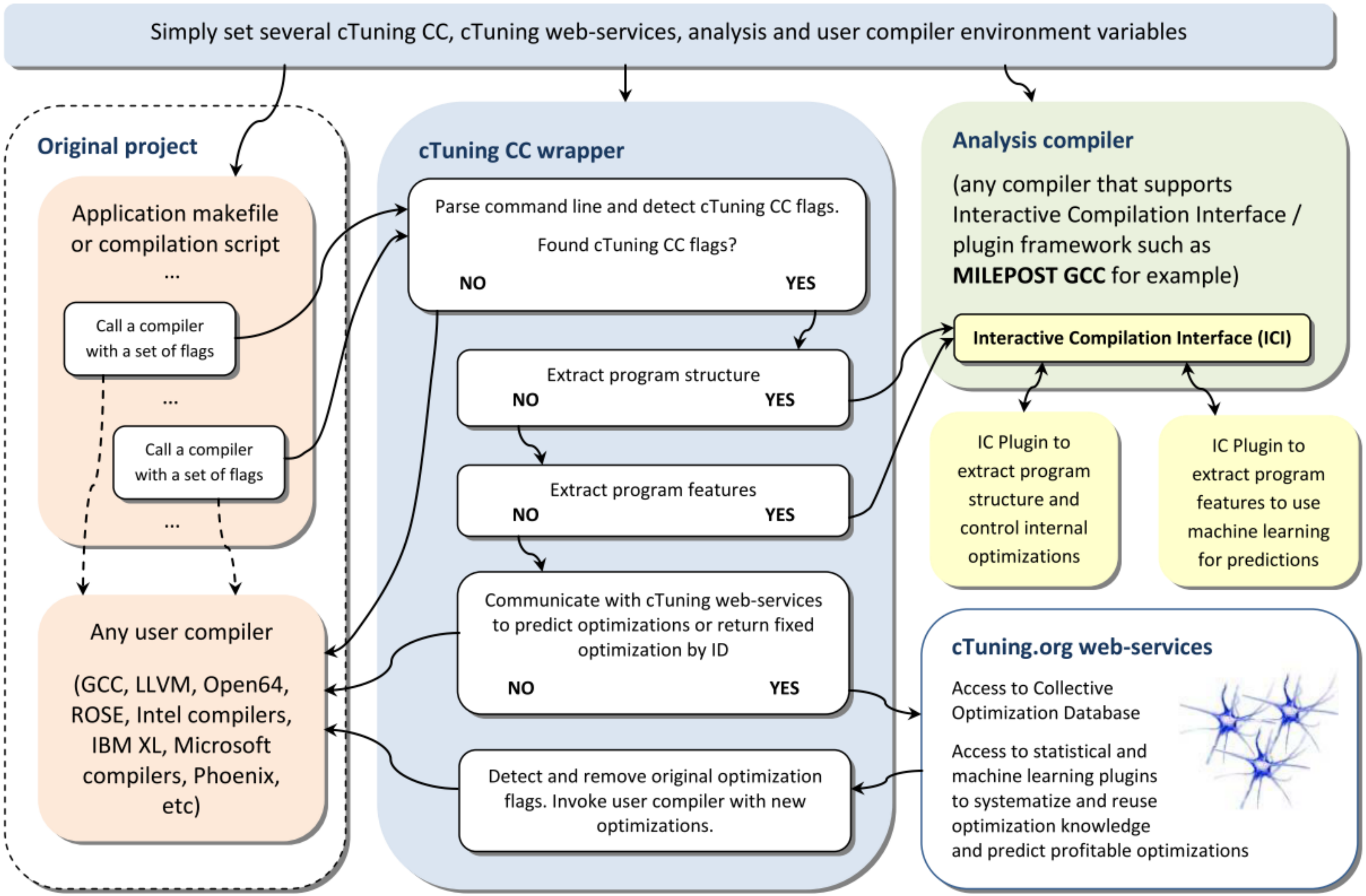
Optimize programs/architectures and enable adaptive self-tuning computing systems

(programs, run-time systems, compilers, architectures)

Web services:

- collect static & dynamic optimization cases
- suggest good optimizations (based on program and architecture features, run-time behavior and optimization scenarios)

Machine learning compiler (MILEPOST GCC / cTuning CC)



First proof-of-concept machine learning compiler connected with cTuning database through unified web-services has been released in 2009. Since then, it has been extended within collaborative projects and Google Summer of Code program. More info can be found at http://cTuning.org/ctuning-cc

What have we learnt from $cTuning_1$

- 15 years ago - lots of disbelief

- Now we have a complete reference framework and repository to validate and extend research ideas on auto-tuning, run-time adaptation and machine learning (cTuning/MILEPOST GCC)

- Community can reproduce and share results

- Community can focus more on research using collective data sets

Problems:

- Global repository not scalable

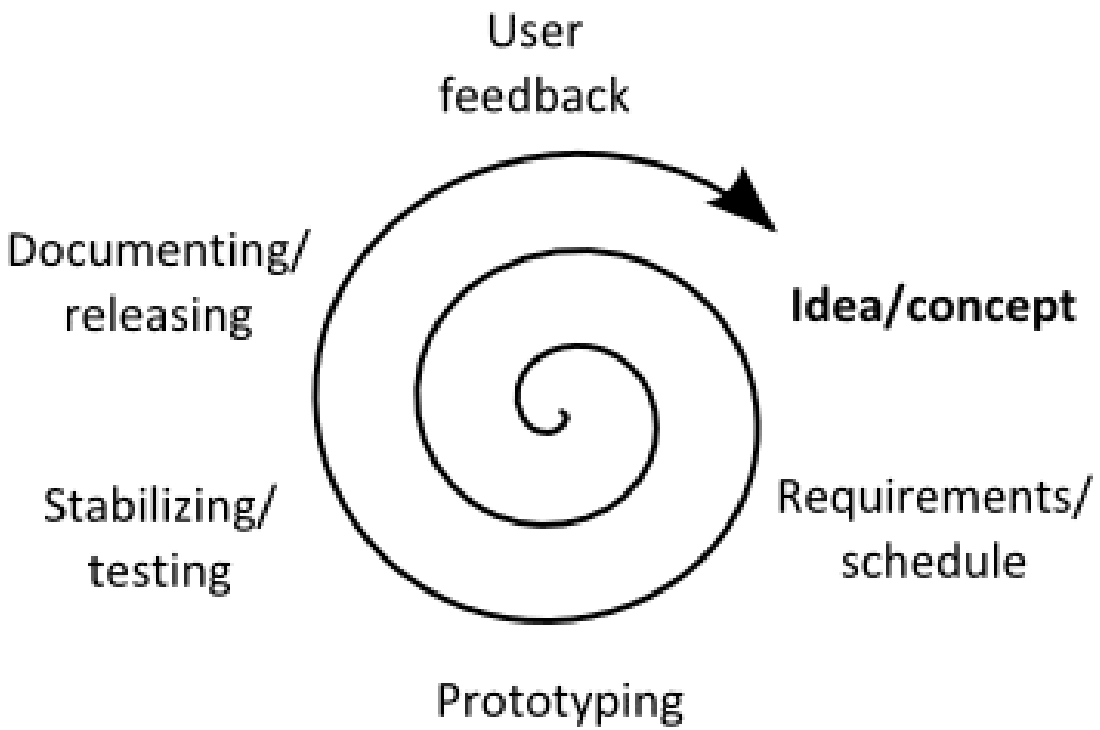
- MySQL is slow and not extensible

- No easy way to share modules, benchmarks, data sets

- Programming modules in C/PHP was not so simple for end-users

What have we learnt from cTuning[1]

Incremental agile development methodology is very useful!



Unlike traditional rigid development methodologies, we can adapt/modify plan if we encounter problems!

What have we learnt from cTuning$_1$

It's fun working with the community!

My favorite comment 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…

What have we learnt from cTuning$_1$

It's fun working with the community!

My favorite comment about MILEPOST GCC from Slashdot.org:

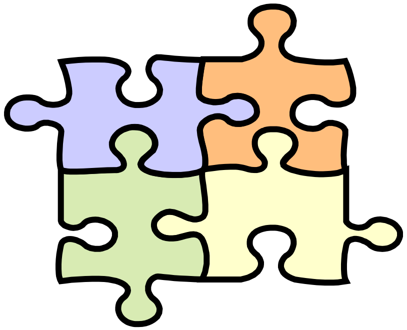
http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

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…

Not all feedback is positive - helps you learn, improve tools and motivate new research directions!

Community helps you to develop your tools and speed up your research!

cTuning$_2$ aka Collective Mind



Methodology for collaborative design and optimization of computer systems is ready (seems like we have all the pieces of the puzzle)!

- Build extensible infrastructure and distributed repository to record information flow inside computer systems and share data and modules from multiple users (applications, data sets, tools, optimization cases, algorithms, etc)
- Write core in python and use json to allow users quickly prototype their research ideas without long learning curve (research LEGO).
  Stable modules can be easily shared with the community
- Provide event mechanism for C, C++, Fortran, Java, PHP
- Gradually convert end-user applications into cM modules with unified interfaces
- End-users become researchers or "physicist" and think about how to make their code auto-tunable and cM-compatible to be able to apply auto-tuning and machine learning rather than hardwiring various optimizations

cTuning$_2$ aka Collective Mind

- Enable continuous observation of the behavior of the whole (!) system
- Enable continuous exploration of multiple design and optimization dimensions
- Explain, characterize and classify unusual/unexpected behavior
  (discover knowledge through data mining)
- Perform hierarchical analysis starting from very simple cases while gradually increasing complexity (decompose large applications into more understandable pieces and quickly perform first coarse-grain analysis/tuning while moving to finer-grain effects only when/if needed)

cTuning$_2$ aka Collective Mind

- Automatically and continuously classify and correlate program/architecture behaviour with “features”, optimizations and multiple objective functions using predictive modelling
- Build an expert system that queries repository and models to :
  - *quickly identify program and architecture behavior anomalies*
  - *suggest better optimizations for a given program*
  - *suggest better architecture designs*
  - *suggest run-time adaptation scenarios*

    (program optimizations and hardware reconfigurations as reaction to program and system behavior)

Future work

- Release of the new framework as LGPL before summer 2012
- Collaborate with researchers and end-users to add various modules

to characterize and optimize existing computer systems:

  - compiler optimizations
  - parallelization (OpenMP/MPI)
  - run-time scheduling and adaptation (CPU/GPU, avoid contentions)

- Evaluate various machine learning techniques for classification and predictive modeling
  - detect important characteristics of computer systems
  - evaluate various ML techniques (SVM, decision trees, hierarchical modeling)
- Rank solutions statistically and continuously
- Long-term: attract funding to support this open source development and research:
- Provide consulting on cTuning technology

A few references

• Grigori Fursin. **Collective Tuning Initiative: automating and accelerating development and optimization of computing systems.** Proceedings of the GCC Summit'09, Montreal, Canada, June 2009

• Grigori Fursin and Olivier Temam. **Collective Optimization: A Practical Collaborative Approach.** ACM Transactions on Architecture and Code Optimization (TACO), December 2010, Volume 7, Number 4, pages 20-49

• Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Chris Williams, Michael O'Boyle. **MILEPOST GCC: machine learning enabled self-tuning compiler.** International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

• Victor Jimenez, Isaac Gelado, Lluis Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2009), Paphos, Cyprus, January 2009

• Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. **Finding representative sets of optimizations for adaptive multiversioning applications.** 3rd International Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'09) co-located with HiPEAC'09, Paphos, Cyprus, January 2009

A few references

•Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. **MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2007), Ghent, Belgium, January 2007

•F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O'Boyle, J. Thomson, M. Toussaint and C.K.I. Williams. **Using Machine Learning to Focus Iterative Optimization.** Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO), New York, NY, USA, March 2006

•Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. **A Practical Method For Quickly Evaluating Program Optimizations.** Proceedings of the 1st International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005), number 3793 in LNCS, pages 29-46, Barcelona, Spain, November 2005

•Grigori Fursin, Mike O'Boyle, Olivier Temam, and Gregory Watts. **Fast and Accurate Method for Determining a Lower Bound on Execution Time.** Concurrency Practice and Experience, 16(2-3), pages 271-292, 2004

• Grigori Fursin. **Iterative Compilation and Performance Prediction for Numerical Applications.** Ph.D. thesis, University of Edinburgh, Edinburgh, UK, January 2004

PDFs available at http://fursin.net/dissemination

Workshops

EXADAPT

or extinct

http://exadapt.org

"It is not the strongest of the species that survives, or the most intelligent; it is the one most capable of change"
attributed to Charles Darwin

EXADAPT 2011 at FCRC/PLDI 2011

ACM International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, sponsored by Google and ACM

Keynote: "Autotuning in the Exascale Era!"
Prof. Katherine Yelick (LBNL and UC Berkeley, USA)

EXADAPT 2012 at ASPLOS 2012

Keynote: "Self-Tuning Bio-Inspired Massively-Parallel Computing"
Prof. Steve Furber (University of Manchester, UK)

Acknowledgements

- Davide Del Vento, NCAR, USA
- Vicki Holzhauer, NCAR, USA
- Abdul Wahid Memon, INRIA, France
- Yuriy Kashnikov, Intel Exascale Lab, France
- Colleagues from UIUC, USA and ICT, China
- cTuning community:

Collective TUNING

http://ctuning.org/wiki/index.php/Community:People

- EU FP6, FP7 program and HiPEAC network of excellence

http://www.hipeac.net

- IBM, Intel, Google, STMicroelectronics

Questions?

Contact: grigori.fursin@inria.fr
grigori.fursin@exascalable.com

cTuning$_1$: http://cTuning.org
http://groups.google.com/group/ctuning-discussions

cTuning$_2$: http://code.google.com/p/collective-mind
http://twitter.com/cresearch