Collective Mind Methodology, Framework and Repository: systematizing and crowdsourcing multi-objective auto-tuning

Grigori Fursin
INRIA, France

INRIA/Paris South University, France
December 2013
**Message (nearly 20 years of R&D)**

**Question:** how to design most efficient computer systems (software and hardware) in terms of performance, power consumption, reliability, size, costs…

**Long term interdisciplinary vision:** from brain, crowdsourcing and big data to collaborative, systematic and reproducible computer engineering

Continuously validated in industrial projects with IBM, CAPS, ARC (Synopsys), ARM, Intel, STMicroelectronics
• My personal motivation since 1993 (still the same)
• General problems in computer engineering
• Cleaning up research and experimental mess
  ▪ Collective Mind Repository and Infrastructure
  ▪ Reproducible research and experimentation
  ▪ Crowdsourcing, predictive modelling
• Unifying compiler multi-objective auto-tuning
• Unifying performance modelling
• Conclusions and future work
Back to 1993

Semiconductor neural element - base of neural accelerators and computers

Modeling and understanding brain functions

My problem with modeling:
- Slow
- Unreliable
- Costly
How to solve it if background is more in physics, electronics and AI and not computer engineering?

User task

Available solution(s)
- Algorithm
- Application
- Compilers
- Binary and libraries
- State of the system
- Data set
- Run-time environment
- Architecture

Result

Service/application providers
(HPC, supercomputers, mobile systems)

User requirements:
- minimize all costs (characteristics)
  (execution time, power consumption, price, size, faults, etc)
- guarantee real-time constraints
  (bandwidth, QoS, etc)

Hardware and software designers
Delivering optimal solutions is tedious:

1) Rising complexity of computer systems: too many design and optimization choices at ALL levels

2) Performance is not anymore the only requirement: multiple user objectives vs choices benefit vs optimization time

3) Complex relationship and interactions between ALL software and hardware components

4) Too many tools with non-unified interfaces changing from version to version: technological chaos
Auto-tuning for compilation and architecture

Find empirically optimal optimizations in multi-dimensional space while balancing multiple characteristics:

- execution time
- code size
- compilation time
Too many dimensions, too long, too much data (big data)

Grigori Fursin  “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”

Major problems in my projects:

- **Long training times (both auto-tuning and ML)**
  
  *1999-2005 (PhD and EU MHAOTEU project)*
  
  4 kernels / SPEC2000, 1 datasets, 2 architectures, tiling/unrolling/padding, ~4 months of experiments, SHARED as CSV and thorough MySQL

  *2006-2009 (EU MILEPOST project)*
  
  16 benchmarks, 1 dataset, 3 architectures, GCC and ICC, 500 combinations of flags, ~6 months of experiments, SHARED through MySQL, plugin-based framework and web services

  *2009-2011 (Collective Tuning)*
  
  16 benchmarks, 20..1000 datasets, GRID5000 with 16 nodes, ~10 months of experiments, SHARED through MySQL, plugin based framework and web services

  *2011-cur (Collective Mind)*
  
  300 benchmarks, 20..1000 datasets
  
  GRID5000 with 100 nodes,

  **Some experiments are still in progress,**

  **SHARED ONLINE**

Find empirically optimal optimizations in multi-dimensional space while balancing multiple characteristics:

- execution time
- code size
- compilation time

**GCC optimization evolution**

![Graph showing GCC optimization evolution](image)
Machine Learning for compilation and architecture

Task

Classify behavior

Build predictive models

Result

Classification

Training set

programs, codelets, kernels, etc
Machine Learning for compilation and architecture

Task

- Classify behavior
- Build predictive models

Classification
- Monitor behavior
- Optimize
- Extract “properties” or “features”

Training set

Result

programs, codelets, kernels, etc
Machine Learning for compilation and architecture

- **Task**
  - Classify behavior
  - Build predictive models

- **Training set**
  - programs, codelets, kernels, etc

- **Classification**
  - Monitor behavior
  - Optimize
  - Extract “properties” or “features”

- **Result**
  - Find most close code
  - New “unseen” code

Grigori Fursin

“Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”
Machine Learning for compilation and architecture

**Task**
- Classify behavior
- Build predictive models

**Training set**
- Programs, codelets, kernels, etc

**Classification**
- Monitor behavior
- Optimize
- Extract “properties” or “features”

**Result**
- Find most close code
- Predict behavior or optimization
- New “unseen” code
Machine Learning for compilation and architecture

Task

Classify behavior

Build predictive models

Result

Training set

Classification

Monitor behavior

Optimize

Extract “properties” or “features”

Predict behavior or optimization

Find most close code

programs, codelets, kernels, etc

New “unseen” code

Regression

Classify behavior

Build predictive models

Result
Machine Learning for compilation and architecture

Task

Classify behavior

Build predictive models

Result

Classification

Monitor behavior
Optimize
Extract "properties" or "features"

Find most close code

Predict behavior or optimization

Regression

Performance vs. Data set size

Task: Classify behavior, Build predictive models. Result: Classification, Regression.
Auto-tuning, genetic programming, machine-learning, co-design shows high potential for more than 2 decades but still far from the mainstream in production environments due to:

- Optimization spaces are large and non-linear with many local minima
- Exploration is slow and ad-hoc (random, genetic, some heuristics)
- Only small part of the system is taken into account (rarely reflect behavior of the whole system)
- No common, large and diverse training sets (benchmarks and data sets)
- Black box model doesn’t help architecture or compiler designers
- Many statistical pitfalls and wrong usages of machine learning for compilation and architecture

By the end of experiments, new tool versions or architectures are often available; life span of experiments and ad-hoc frameworks - end of MS or PhD project; Researchers focus on publications rather than practical and reproducible solutions; Computer engineering is considered by students as hacking rather than science.
My step 1: formalize analysis and optimization (1999-cur.)

Combine interdisciplinary knowledge in physics, electronics, mathematics, neural networks and machine learning

Consider tasks and computational resources as a complex physical system

User task

Result

Exposure additional information

Gradually expose all available algorithm, design and optimization choices

Continuously observe behavior (characteristics); check for normality

Predict optimal choices / behavior if enough knowledge

If unexpected behavior, improve models, increase granularity, find more properties

Behavior / characteristics (\(\vec{b}\))

Requirements (\(\vec{r}\))

System/task state (\(\vec{s}\))

Properties (\(\vec{p}\))

Continuously learning (modeling) observed behavior

Combine interdisciplinary knowledge in physics, electronics, mathematics, neural networks and machine learning

My step 1: formalize analysis and optimization (1999-cur.)
Got stuck with a limited number of benchmarks, datasets, architectures and a large number of optimizations and generated data; could not validate data mining and machine learning techniques

Needed dramatically new approach!

Millions of users run realistic applications on different architectures with different datasets, run-time systems, compilers, optimizations!

Can we leverage their experience and computational resources?

Can we connect disjoint analysis, tuning, learning tools together with public repository of knowledge?
**Revolutionary approach:**

Let’s redesign the whole system and make it tunable and adaptable?

- Too complex and time consuming (decades)
- Community will not easily accept

*Hardwired experimental setups, very difficult to change, scale or share*
How to implement?

Experiments

- Ad-hoc tuning scripts
- Tool \( A_{v1} \)
- Tool \( A_{v2} \)
- Tool \( A_{VN} \)
- Tool \( B_{v1} \)
- Tool \( B_{v2} \)
- Tool \( B_{VM} \)
- Ad-hoc analysis and learning scripts
- Collection of CSV, XLS, TXT and other files

Revolutionary approach:
Let’s redesign the whole system and make it tunable and adaptable?

- Too complex and time consuming (decades)
- Community will not easily accept

Evolutionary agile methodology:
Gradually clean-up system and make it tunable and adaptable while involving community
How to implement?

Tool wrapper with unified and formalized input and output

Unified JSON input (meta-data)

Action function

Unified JSON output (meta-data)

Formalized function (model) of a component behavior

Flattened JSON vectors (either string categories or integer/float values)

cm [module name] [action] (param₁=value₁ param₂=value₂ ... -- unparsed command line)

Example commands:

cm compiler build -- icc -fast *.c

cm code.source build ct_compiler=icc13 ct_optimizations=-fast

cm code run os=android binary=./a.out dataset=image-crazy-scientist.pgm

Should be able to run on any OS (Windows, Linux, Android, MacOS, etc)!
Ad-hoc tuning scripts | Tool A_{V1} | Tool B_{V1} | Ad-hoc analysis and learning scripts

Tool wrapper with unified and formalized input and output

Unified JSON input (meta-data)
- Action
- Behavior
- Choices
- Features
- State

Action function
- Set environment for a given tool version
- Parse and unify output

Unified JSON output (meta-data)

Generated files
- Tool B_{Vi}

Collection of CSV, XLS, TXT and other files

Unified JSON input (if exists)

Original unmodified ad-hoc input

Formalized function (model) of a component behavior

\[ \mathcal{B} = \mathbf{B}(\mathbf{c}, \mathbf{f}, \mathbf{s}) \]

Flattened JSON vectors (either string categories or integer/float values)

Chaining components (wrappers) to an experimental pipeline for a given research and experimentation scenario

Choose exploration strategy

Generate choices (code sample, data set, compiler, flags, architecture …)

Compile source code

Run code

Test behavior normality

Pareto filter

Modeling and prediction

Complexity reduction

Public modular auto-tuning and machine learning repository and buildbot

Unified web services

Interdisciplinary crowd

Shared scenarios from past research

Grigori Fursin “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”
Grigori Fursin        “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”

Gradually expose some characteristics

Gradually expose some choices and features

Compile Program

time ...

compiler flags; pragmas ...

Start from coarse-grain and gradually move to fine-grain level!

Run code

Run-time environment

time; CPI, power consumption ...

pinning/scheduling ...

System

cost;

architecture; frequency; cache size ...

Data set

size; values; description ...

precision ...

Analyze profile

time; size ...

instrumentation; profiling ...

Start coarse-grain decomposition of a system (detect coarse-grain effects first). Add universal learning modules.
Experimental pipelines for auto-tuning and modeling

- **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)

- **Build 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**
Currently prepared experiments

Our Collective Mind Buildbot supports the following shared benchmarks and codelets:

- Polybench - numerical kernels with exposed parameters of all matrices in cM
  - CPU: 28 prepared benchmarks
  - CUDA: 15 prepared benchmarks
  - OpenCL: 15 prepared benchmarks
- cBench - 23 benchmarks with 20 and 1000 datasets per benchmark
- Codelets - 44 codelets from embedded domain (provided by CAPS Entreprise)
- SPEC 2000/2006
- Description of 32-bit and 64-bit OS: Windows, Linux, Android
- Description of major compilers: GCC 4.x, LLVM 3.x, Open64/Pathscale 5.x, ICC 12.x
- Support for collection of hardware counters: perf, Intel vTune
- Support for frequency modification
- Validated on laptops, mobiles, tables, GRID/cloud - can work even from the USB key
Multi-objective compiler auto-tuning using mobile phones

Use Pareto frontier filter;
Pack experimental data on the fly

Program: *image corner detection*
Compiler: *Sourcery GCC for ARM v4.7.3*
System: *Samsung Galaxy Y*

Processor: *ARM v6, 830MHz*
OS: *Android OS v2.3.5*
Data set: *MiDataSet #1, image, 600x450x8b PGM, 263KB*

500 combinations of random flags -O3 -f(no-)FLAG

Powered by Collective Mind Node (Android Apps on Google Play)
Universal complexity (dimension) reduction

Found solution


Not very useful for analysis
Universal complexity (dimension) reduction

- found solution

Chain complexity reduction filter
remove dimensions (or set to default)
iteratively, ANOVA, PCA, etc...

Auto-tuning experimental pipeline

\[ b \rightarrow B(c) \]

Grigori Fursin, “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”
Universal complexity (dimension) reduction

Found solution


Pruned solution


(25% of speedup)

Grigori Fursin  “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”
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
Current machine learning usage

Training set: distinct combination of compiler optimizations (clusters)

\[ \vec{c} \text{ (choices)} \]

\[ \vec{f} \text{ (features)} \]

MILEPOST GCC features, hardware counters

Optimization cluster

Some ad-hoc predictive model

Some ad-hoc features

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)

\[ \vec{c} \text{ (choices)} \]

\[ \vec{f} \text{ (features)} \]

\[ \text{MILEPOST GCC features, hardware counters} \]

Optimization cluster

\[ \text{Unseen program} \]

\[ \vec{f} \text{ (features)} \]

Some ad-hoc predictive model

Some ad-hoc features

Prediction

Optimization cluster

\[ \vec{c} \text{ (choices)} \]
Current machine learning usage

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%

Previous limited studies

Optimization cluster

Prediction

\( \vec{c} \) (choices)
Current machine learning usage

Training set: distinct combination of compiler optimizations (clusters)

$\vec{c}$ (choices)

$\vec{f}$ (features)

$\text{MILEPOST GCC}$
$\text{features, hardware counters}$

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
### Learning features by domain specialists

**Image B&W threshold filter**

\[
 matrix_{ptr2}++ = (temp1 > T) \ ? 255 : 0;
\]

<table>
<thead>
<tr>
<th>Class</th>
<th>-O3</th>
<th>-O3 -fno-if-conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data set sample(_1)</td>
<td><em>reference execution time</em></td>
<td>no change</td>
</tr>
<tr>
<td>Shared data set sample(_2)</td>
<td>no change</td>
<td>+17.3% improvement</td>
</tr>
</tbody>
</table>

---

*Matrix* `matrix_{ptr2}++` is updated based on a comparison between `temp1` and threshold `T`. The update rule is `(temp1 > T) \ ? 255 : 0;`. This operation applies a B&W threshold filter, where values above `T` are set to 255 (black), and others are set to 0 (white). The table illustrates the effect of optimizing the code with `-O3` and `-O3 -fno-if-conversion` flags on the `reference execution time` and execution speed, showing improvements. The first sample shows no change, while the second sample demonstrates a 17.3% improvement in execution speed.
### Learning features by domain specialists

**Image B&W threshold filter**


code

```c
*matrix_ptr2++ = (temp1 > T) ? 255 : 0;
```

<table>
<thead>
<tr>
<th>Class</th>
<th>-O3</th>
<th>-O3 -fno-if-conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data set sample₁</td>
<td>reference execution time</td>
<td>no change</td>
</tr>
<tr>
<td>Monitored during <em>day</em></td>
<td><img src="image1" alt="Day Image" /></td>
<td><img src="image2" alt="Day Image" /></td>
</tr>
<tr>
<td>Shared data set sample₂</td>
<td>no change</td>
<td>+17.3% improvement</td>
</tr>
<tr>
<td>Monitored during <em>night</em></td>
<td><img src="image3" alt="Night Image" /></td>
<td><img src="image4" alt="Night Image" /></td>
</tr>
</tbody>
</table>
Learning features by domain specialists

Image B&W threshold filter

\[ *\text{matrix}_\text{ptr}2++ = (\text{temp1} > T) \ ? \ 255 : 0; \]

<table>
<thead>
<tr>
<th>Class</th>
<th>-O3</th>
<th>-O3 -fno-if-conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data set sample(_1)</td>
<td>reference execution time</td>
<td>no change</td>
</tr>
<tr>
<td>Monitored during day</td>
<td><img src="image1.jpg" alt="Image" /></td>
<td><img src="image2.jpg" alt="Image" /></td>
</tr>
<tr>
<td>Shared data set sample(_2)</td>
<td>no change</td>
<td>+17.3% improvement</td>
</tr>
<tr>
<td>Monitored during night</td>
<td><img src="image3.jpg" alt="Image" /></td>
<td><img src="image4.jpg" alt="Image" /></td>
</tr>
</tbody>
</table>

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
  \[ \text{bw\_filter\_codelet\_day}(\text{buffers}); \]
else
  \[ \text{bw\_filter\_codelet\_night}(\text{buffers}); \]
Unexpected behavior - expose to the community including domain specialists, explain, find missing feature and add to the system
Unexpected behavior - expose to the community including domain specialists, explain, find missing feature and add to the system.

![Graph showing execution time distribution for Class A and Class B with CPU frequencies of 800MHz and 2400MHz.](image)
How we can explain the following observations for some piece of code ("codelet object")?

(LU-decomposition codelet, Intel Nehalem)
Add 1 property: matrix size
Try to build a model to correlate objectives (CPI) and features (matrix size).

Apply shared models, start from simple cases: linear regression (detect coarse grain effects)
If more observations, **validate model and detect discrepancies**!

**Continuously retrain models to fit new data!**

**Use model to “focus” exploration on “unusual” behavior!**

---

Using Collective Mind to learn behavior of computer systems

Dataset properties: matrix size

Program / architecture behavior: CPI
Gradually increase model complexity if needed (hierarchical modeling). For example, detect fine-grain effects (singularities) and characterize them.
Start adding **more properties** (one more architecture with **twice bigger cache**)!

Use automatic approach to correlate all objectives and features.

---

**Legend:**
- $L_3 = 4\text{Mb}$
- $L_3 = 8\text{Mb}$

**Plot:**
- X-axis: Dataset properties: matrix size
- Y-axis: Program / architecture behavior: CPI

---

Grigori Fursin  “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning” 44
Continuously build and refine classification (decision trees for example) and predictive models on all collected data to improve predictions.

Continue exploring design and optimization spaces (evaluate different architectures, optimizations, compilers, etc.)

Focus exploration on unexplored areas, areas with high variability or with high mispredict rate of models

**cM predictive model module**

\[ \text{CPI} = \epsilon + 1000 \times \beta \times \text{data size} \]
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!

Dataset features: matrix size

Code/architecture behavior: CPI

0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000

0 1 2 3 4 5 6

- Size < 1012
- 1012 < Size < 2042
- Size > 2042 & GCC
- Size > 2042 & ICC & O2
- Size > 2042 & ICC & O3

Complexity reduction
Collaboratively and continuously add expert advices or automatic optimizations.
Collaboratively and continuously add **expert advices** or **automatic optimizations**.

Automatically characterize problem (extract all possible features: hardware counters, semantic features, static features, state of the system, etc)

Add manual analysis if needed
Collaboratively and continuously add **expert advices** or **automatic optimizations**.

**cTuning advice system:**
Possible problem: 
**Cache conflict misses degrade performance**
Collaboratively and continuously add **expert advices** or **automatic optimizations**.

**cTuning advice system:**

Possible problem:  
**Cache conflict misses degrade performance**

Possible solution:  

Effect:  
**~30% execution time improvement**
Share
Explore
Model
Discover
Reproduce
Extend
Have fun!


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

Accepted as an EU HiPEAC theme (2012-2016)
It’s fun 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…
It’s fun 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...

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

Community was interested to validate and improve techniques!
New community-driven research, development and education

Quick, non-reproducible hack?
Ad-hoc heuristic?
Quick publication?
Waste of expensive resources?

User’s task

Current state of computer engineering

RADICALLY REVISIT CURRENT AD-HOC RESEARCH, DEVELOPMENT AND EDUCATIONAL METHODOLOGY

Public repository and infrastructure

Continuous systematization and unification of design and optimization of computer systems

Collaborative Infrastructure and repository for continuous online learning

Classification, predictive modeling

Optimal solutions

Systematization and unification of collective knowledge (big data)

“crowd”

Result

• Validate, share, enhance and systematize our past research knowledge and practical experience during auto-tuning
• Extrapolate collected knowledge to build faster and more power efficient computer systems to continue innovation in science and technology!

Grigori Fursin  “Collective Mind: systematizing and crowdsourcing multi-objective auto-tuning”
Current status

- Pilot repository is available at [http://c-mind.org/repo](http://c-mind.org/repo)
  
  *(hundreds of kernels, thousands of datasets, tools, models, etc)*

- Whole infrastructure is available at SourceForge under standard BSD license ([http://c-mind.org](http://c-mind.org))

- Collective Mind concept requires community effort at all levels (sharing benchmarks and data sets, providing wrappers, finding features, improving models) - currently building community around this concept and infrastructure with a focus on:

<table>
<thead>
<tr>
<th>Education</th>
<th>Academic research</th>
<th>Industrial validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproducible and collaborative research in computer engineering; new publication model where results are validated by the community.</td>
<td>• Systematizing, validating, sharing past research knowledge and practical experience during auto-tuning and ML</td>
<td>• Most of techniques have been validated in industry with IBM, ARM, Intel, ARC (Synopsys), CAPS, STMicroelectronics</td>
</tr>
<tr>
<td>• Panel at ADAPT 2014 @ HiPEAC 2014 <a href="http://adapt-workshop.org">http://adapt-workshop.org</a></td>
<td>• Optimal feature and model selection</td>
<td>• Continue extrapolating collected knowledge to build faster and more power efficient computer systems to continue innovation in science and technology!</td>
</tr>
<tr>
<td>• TRUST 2014 @ PLDI 2014 <a href="http://c-mind.org/events/trust2014">http://c-mind.org/events/trust2014</a></td>
<td>• Compacting and systematizing benchmarks and data sets</td>
<td></td>
</tr>
<tr>
<td>• Special journal issue on reproducible research in ACM TET</td>
<td>• Run-time adaptation and ML</td>
<td></td>
</tr>
</tbody>
</table>

Grigori Fursin, “Collective Mind: cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning”, INRIA Tech. report No 00850880, August 2013

[http://hal.inria.fr/hal-00850880](http://hal.inria.fr/hal-00850880)  [http://arxiv.org/abs/1308.2410](http://arxiv.org/abs/1308.2410)
Acknowledgements

• My 2 PhD students:
  Abdul Memon and Yuriy Kashnikov

• Colleagues from STMicroelectronics (France):
  Christophe Guillone, Antoine Moynault, Christian Bertin

• Colleagues from ARM (UK): Anton Lokhmotov

• Colleagues from NCAR (USA): Davide Del Vento and his interns

• Colleagues from CAPS Entreprise (France): Francois Bodin

• Colleagues from Intel (USA): David Kuck and David Wong

• cTuning community: http://cTuning.org/lab/people

• EU FP6, FP7 program and HiPEAC network of excellence http://www.hipeac.net
c-mind.org

Collective Mind Repository and Infrastructure

Systematic application and architecture analysis, characterization and optimization through collaborative knowledge discovery, systematization, sharing and reuse

Thank you for your attention!

Contact: Grigori.Fursin@cTuning.org

http://cTuning.org/lab/people/gfursin

Open repository to share optimization cases and programs

Gradual parameterization and unification of interfaces of computing systems

Modeling and advice system to predict optimizations, architecture designs, run-time adaptation, etc
Collective Mind summary: research LEGO

**Optimization and machine learning buildbot and repository**

- Algorithms
- Programs, codelets
- Compilers, tools
- Binaries, libraries
- Datasets
- Run-times
- Architectures
- Profilers

All shared artifacts

- Extensible meta-data: characteristics, features, choices, ...

Experimental pipelines for research scenarios

- Validate auto-tuning; Build training set

Best cases (optimizations, architecture designs, ...)

Strategies

Continuous training and validation

- Validate classification and predictive modeling

Shared models

- Optimal optimizations and features
- Best (hybrid) models

Realistic or generated programs, codelets, datasets, architecture designs

Update/share representative cases for behavior anomalies

Solve problem and find missing features (automatically or crowdsource)

Expose unusual or unexpected behavior and all related artifacts (for replay)

Interdisciplinary community