Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing

Abdul Memon (UVSQ, France)
Grigori Fursin (INRIA, France)

MS Application Autotuning for HPC
PARCO 2013, September 2013
Munich, Germany
Back to 1993

Semiconductor neural element - possible base of neural computers and accelerators

Modeling and understanding brain functions

My problem with modeling:
- Slow
- Unreliable
- Costly
Revisiting current computer design and optimization methodology

Leveraging experience and computer resources of multiple users

Systematizing auto-tuning, predictive modelling and data mining to improve computer systems using plugin-based knowledge management system (Collective Mind)

Starting international initiative to build collaborative R&D infrastructure and public repository of knowledge (EU HiPEAC, USA OCCAM, various universities and companies)

Tools, benchmarks, datasets, models and repository are gradually released to public since 2006!
Problems in computer engineering

Task

Solutions

Result
Problems in computer engineering

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

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 (co-design).

4) Too many tools with non-unified interfaces changing from version to version: technological chaos
Challenges for end-users and companies:

- finding the right solution for end-user is extremely challenging
- everyone is lost in choices
- dramatic increase in development time
- low ROI
- underperforming systems
- waste of energy
- ad-hoc, repetitive and error-prone manual tuning
- **slowing innovation in science and technology**
Challenges for end-users and companies:

- finding the right solution for end-user is extremely challenging
- everyone is lost in choices
- dramatic increase in development time
- low ROI
- underperforming systems
- waste of energy
- ad-hoc, repetitive and error-prone manual tuning
- **slowing innovation in science and technology**

Understanding and modeling of the overall relationship between end-user algorithms, applications, compiler optimizations, hardware designs, data sets and run-time behavior became simply infeasible!
Attempts to solve these problems: auto-tuning

Treat computer system as a black box

Task

Algorithm

Application

Compilers and auxiliary tools

Binary and libraries

Data set

State of the system

Run-time environment

Architecture

Use auto-tuning:

Explore multiple choices empirically: learn behavior of computer systems across executions

Covered all components in the last 2 decades and showed high potential but ...
Attempts to solve these problems: 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 adaptive heuristics)
- Only a few benchmarks are considered
- Often the same (one) dataset is used
- Only part of the system is taken into account (rarely reflect behavior of the whole system)
- No knowledge sharing
Attempts to solve these problems: machine learning and online tuning

Task

Classify behavior

Build predictive models

Result

Training set

Classification

programs, codelets, kernels, etc
Attempts to solve these problems: machine learning and online tuning

- Classify behavior
- Build predictive models

Training set

- **Classification**
  - Red: Monitor behavior
  - Blue: Optimize
  - Green: Extract “properties” or “features”

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

- **Result**
  - Programs, codelets, kernels, etc
Attempts to solve these problems: machine learning and online tuning

**Task**

- Classify behavior
- Build predictive models

**Classification**

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

**Training set**

- Find most close code

**Result**

- Programs, codelets, kernels, etc
- New “unseen” code
Attempts to solve these problems: machine learning and online tuning

Classify behavior
Build predictive models

Task

Training set

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

Predict behavior or optimization
Find most close code

New “unseen” code

programs, codelets, kernels, etc

Result
Attempts to solve these problems: machine learning and online tuning

Task

Classify behavior

Build predictive models

Result

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Training set

Predict behavior or optimization

Find most close code

programs, codelets, kernels, etc

New “unseen” code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression

Classification

Monitor behavior
Optimize
Extract “properties” or “features”

Find most close code

Regression
Attempts to solve these problems: machine learning and online tuning

Classify behavior
Build predictive models

Task

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

Training set
- Predict behavior or optimization
- Find most close code

programs, codelets, kernels, etc

New “unseen” code

Regression

Attempts to solve these problems: machine learning and online tuning

Classify behavior
Build predictive models

Task

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

Training set
- Predict behavior or optimization
- Find most close code

programs, codelets, kernels, etc

New “unseen” code

Regression
Attempts to solve these problems: machine learning and online tuning

Classify behavior

Build predictive models

Use machine learning to speed up exploration

- Apply predictive modeling to suggest profitable solutions based on properties of a task and a system

Covered all components in the last decade and showed high potential but ...
Attempts to solve these problems: machine learning and online tuning

Machine learning (classification, predictive modeling) shows high potential during past decade but still far from the mainstream.

Why?

• Selection of machine learning models and right properties is non-trivial: ad-hoc in most of the cases - well known problem in machine learning for decades - non-specialists can be easily trapped in wrong interpretation of results

• Only part of the system is taken into account (rarely reflect behavior of the whole system)

• Limited training sets
Can we crowdsource auto-tuning? My main focus since 2004

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 build public repository of knowledge?

Abdul Memon and Grigori Fursin “Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
EU MILEPOST project (2006-2010) and Collective Mind (2012-cur):
We have proposed and started developing collective methodology and infrastructure to crowdsource auto-tuning (cTuning):

• public repository is dynamically evolving and contains all encountered benchmarks, data sets, tools, codelets, optimized binaries and libraries, choices, properties, characteristics, predictive models, decision trees

• repository and plugin-based infrastructure are distributed among many users and can automatically exchange information about
  ▪ unexplored choices
  ▪ optimization areas with high variability
  ▪ optimal predictive models
  ▪ abnormal behavior to focus further exploration and validate or improve classification and models
EU MILEPOST project (2006-2010) and Collective Mind (2012-cur):
We have proposed and started developing collective methodology and infrastructure to crowdsource auto-tuning (cTuning):

- Public repository is dynamically evolving and contains all encountered benchmarks, data sets, tools, codelets, optimized binaries and libraries, choices, properties, characteristics, predictive models, decision trees.
- Repository and plugin-based infrastructure are distributed among many users and:
  - Unexplored choices
  - Optimization areas with high variability
  - Optimal predictive models
  - Abnormal behavior to focus further exploration and validate or improve classification and models.

Main challenge:
How to make it simple, extensible and implement with very limited funding and 1..2 researchers instead of redesigning the whole software/hardware stack (like in IBM's Liquid Metal project)?
cTuning plugin-based framework basics

```
cd [application_directory]
make CC=icc CC_OPTS=-fast
  or
icc -fast *.c
time ./a.out < [my_dataset] > [output]
  record “-fast”, execution time
```
cTuning plugin-based framework basics

```
cd [application_directory]

make CC=icc CC_OPTS=-fast

or

icc -fast *.c

time ./a.out < [my_dataset] > [output]

record “-fast”, execution time
```

\[ E n d-u s e r s o r cM d e v e l o p e r s C M D \]

\[ U n i v e r s a l cM F E \]

\[ cM p l u g i n s (m o d u l e s) \]

- `code.source build`
- `compiler build`
- `code run`

\[ p y t h o n \]

or any other language

Abdul Memon and Grigori Fursin “Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
cd [application_directory]

**make CC=icc CC_OPTS=-fast**

or

**icc -fast *.c**

time ./a.out < [my_dataset] > [output]

record “-fast”, execution time

cm **[module name]** [action] (param\_1=value\_1 param\_2=value\_2 ... -- unparsed command line)

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

cm **compiler build** -- icc -fast *.c

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)!*
Gradual decomposition, parameterization, tuning and learning of computer systems

User task

Complex hardwired computer system

System

Dataset

Compiler

Code

Run-time

Result

Abdul Memon and Grigori Fursin

“Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Gradual decomposition, parameterization, tuning and learning of computer systems

User task

Complex hardwired computer system

Universal Tuning and Learning Plugin

Object plugin

Expose any object

information flow

System

Dataset

Run-time

Compiler

Code

Result

Abdul Memon and Grigori Fursin · “Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Gradual decomposition, parameterization, tuning and learning of computer systems

User task

Complex hardwired computer system

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

System Dataset Runtime

Extractor

Result

Expose any object

information flow

Run-time

expose properties

set requirements

expose system state

expose characteristics

Universal Tuning and Learning Plugin

Dataset

Code

Compiler

Exposing any object

information flow

System

Dataset

Code

Compiler

Exposing any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(history (experience))

Exposed any object

information flow
Gradual decomposition, parameterization, tuning and learning of computer systems

User task

Complex hardwired computer system

System Dataset Runtime Compiler Code

Exposure any object

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(choices, properties, characteristics, system state, data)

Exposure, parameterization, tuning and learning of computer systems

Exposure requirements

Properties exposure

Characteristics exposure

System state exposure

Examples of information flow
continuously build, validate, prune and improve classification and predictive models on the fly
continuously explore possible design and optimization choices
continuously observe behavior and keep history
Exposure history (experience)

Output of other models

Universal Tuning and Learning Plugin

Tune system continuously

Exposure and keep history

Exposure experimental choices, properties, characteristics, system state, data

Universal Tuning and Learning Plugin

Exposure

Exposure System Dataset Runtime Compiler Code

System Dataset Runtime Compiler Code

Abdul Memon and Grigori Fursin
“Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Gradual decomposition, parameterization, tuning and learning of computer systems

User task

Complex hardwired computer system

System
Dataset
Compiler
Code
Run-time

Result

User task

Universal Tuning and Learning Plugin

Object plugin and repository
continuously observe behavior and keep history
(choices, properties, characteristics, system state, data)

Expose any object

Expose any object
information flow
continuously build, validate, prune and improve classification and predictive models on the fly
expose properties
expose characteristics
expose system state
history (experience)
output of other models

Aggregate knowledge and expose to community at cTuning.org through unified Web services

Abdul Memon and Grigori Fursin
“Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Top-down decomposition of computer system to keep complexity under control

Treat computer system as a black box

Task

Result

Algorithm
Application
Compilers and auxiliary tools
Binary and libraries
Data set
State of the system
Run-time environment
Architecture

Compilers and auxiliary tools
Binary and libraries
Data set
State of the system
Run-time environment
Architecture
Top-down decomposition of computer system to keep complexity under control

*Task*

Treat computer system as a black box

*Result*

**cTuning\textsubscript{3} framework aka Collective Mind**

- Algorithm
- Application
- Compilers and auxiliary tools
- Binary and libraries
- Data set
- State of the system
- Run-time environment
- Architecture

Each component is associated with a `Plugin/repo/model`.
Top-down decomposition of computer system to keep complexity under control

- **Task**
  - Treat computer system as a black box

- **Result**

---

**cTuning³ framework aka Collective Mind**

- **Algorithm**
  - Plugin/repo/model

- **Application**
  - Plugin/repo/model

- **Compilers and auxiliary tools**
  - Plugin/repo/model

- **Binary and libraries**
  - Plugin/repo/model

- **Data set**
  - Plugin/repo/model

- **State of the system**
  - Plugin/repo/model

- **Run-time environment**
  - Plugin/repo/model

- **Architecture**
  - Plugin/repo/model

---

Light-weight interface to connect modules, data and models

---

Abdul Memon and Grigori Fursin

“Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Top-down decomposition of computer system to keep complexity under control

<table>
<thead>
<tr>
<th>Gradually expose some characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradually expose some properties/choices</td>
</tr>
</tbody>
</table>

Compile Program

Gradually expose some characteristics

Compile Program

Gradually expose some properties/choices

<table>
<thead>
<tr>
<th>Gradually expose some characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradually expose some properties/choices</td>
</tr>
</tbody>
</table>

Gradually expose some characteristics

Gradually expose some properties/choices

Gradually expose some characteristics

Gradually expose some properties/choices

Gradually expose some characteristics

Gradually expose some properties/choices

Combine expert knowledge with automatic detection!

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

---

<table>
<thead>
<tr>
<th>Start coarse-grain decomposition of a system (detect coarse-grain effects first). Add universal learning modules.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Compile Program</th>
<th>time ...</th>
<th>compiler flags; pragmas ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run code</td>
<td>Run-time environment</td>
<td>time; CPI, power consumption ...</td>
</tr>
<tr>
<td>Run code</td>
<td>Run-time environment</td>
<td>time; CPI, power consumption ...</td>
</tr>
<tr>
<td>Run code</td>
<td>Run-time environment</td>
<td>time; CPI, power consumption ...</td>
</tr>
<tr>
<td>System</td>
<td>cost;</td>
<td>architecture; frequency; cache size...</td>
</tr>
<tr>
<td>Data set</td>
<td>size; values; description ...</td>
<td>precision ...</td>
</tr>
<tr>
<td>Analyze profile</td>
<td>time; size ...</td>
<td>instrumentation; profiling ...</td>
</tr>
</tbody>
</table>
How we can explain the following observations for some piece of code ("codelet object")?
(LU-decomposition codelet, Intel Nehalem)
Example of characterizing/explaining behavior of computer systems

Add 1 property: matrix size

![Graph showing the relationship between dataset property (matrix size) and program/architecture behavior (CPI). The graph has a scatter plot with data points indicating a correlation between the two variables.]
Try to build a model to correlate objectives (CPI) and features (matrix size).

Start from simple models: 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!
Example of characterizing/explaining behavior of computer systems

Gradually increase model complexity if needed (hierarchical modeling). For example, detect fine-grain effects (singularities) and characterize them.

![Graph showing the relationship between program/architecture behavior: CPI and dataset properties: matrix size.](image)
Start adding **more properties** (one more architecture with **twice bigger cache**)!

Use automatic approach to correlate all objectives and features.

**Example of characterizing/explaining behavior of computer systems**

![Graph showing dataset properties vs. program/architecture behavior: CPI](image-url)
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**

$$\text{CPI} = \varepsilon + 1000 \times \beta \times \text{data size}$$
Model optimization and data compaction

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
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.

![Graph showing the relationship between dataset features and code/architecture behavior: CPI](image)
Extensible and collaborative advice system

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

![Graph](image-url)
Extensible and collaborative advice system

Collaboratively and continuously add expert advices or automatic optimizations.

**cM advice system:**
Possible problem:
Cache conflict misses degrade performance

Dataset features: matrix size

Code/architecture behavior: CPI
Extensible and collaborative expert system

Collaboratively and continuously add expert advices or automatic optimizations.

**cM advice system:**

Possible problem:
Cache conflict misses degrade performance

Possible solution:
Array padding (A[N,N] -> A[N,N+1])

Effect:
~30% execution time improvement

Dataset features: matrix size

Code/architecture behavior: CPI
Add dynamic memory characterization through semantically non-equivalent modifications.

*For example, convert all array accesses to scalars to detect balance between CPU/memory accesses.*

Intentionally change/break semantics to observe reaction in terms of performance/power etc!

Add dynamic memory characterization through semantically non-equivalent modifications. For example, convert all array accesses to scalars to detect balance between CPU/memory accesses. Intentionally change/break semantics to observe reaction in terms of performance/power etc!

System reaction to code changes: physicist’s view

Expert or automatic advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

Advice:
**Small gap (arithmetic dominates):**
- Focus on ILP optimizations
- Run on complex out-of-order core
- Increase processor frequency to speed up application

System reaction to code changes: physicist’s view

Expert or automatic advices based on additional information in the repository!

Focus optimizations to speed up search: which/where?

---

**Advice:**

**Small gap (arithmetic dominates):**
- Focus on ILP optimizations
- Run on complex out-of-order core
- Increase processor frequency to speed up application

**Big gap (data accesses dominate):**
- Focus on memory optimizations
- Run on simple core
- Decrease processor frequency to save power

---

**Graph:**
- X-axis: Dataset features: matrix size
- Y-axis: Execution time, sec
- Data points representing different matrix sizes and their corresponding execution times.

---

Expert or automatic advices based on additional information in the repository!

**Focus optimizations to speed up search**: which/where?

**Advice:**

**Big gap (data accesses dominate):**
- Focus on memory optimizations
- Run on simple core
- Decrease processor frequency to save power

**Small gap (arithmetic dominates):**
- Focus on ILP optimizations
- Run on complex out-of-order core
- Increase processor frequency to speed up application

**Can add/remove instructions, Can add/remove threads, etc…**

**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
Multi-objective crowd-tuning using mobile phones

Program: \textit{cBench: susan corners} 
Compiler: \textit{Sourcery GCC for ARM v4.6.1} 
System: \textit{Samsung Galaxy Y} 
Processor: \textit{ARM v6, 830MHz} 
OS: \textit{Android OS v2.3.5} 
Data set: \textit{MiDataSet #1, image, 600x450x8b PGM, 263KB}
Online learning, tuning and split compilation

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

Dynamic

- Machine learning techniques to find mapping between different run-time contexts and representative versions
- Crowd-tuning 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
- Run-time frequency switching
Gradually increasing complexity

<table>
<thead>
<tr>
<th>Gradually expose some characteristics</th>
<th>Gradually expose some choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm selection</td>
<td>(time) productivity, variable-accuracy, complexity ...</td>
</tr>
<tr>
<td>Compile Program</td>
<td>time ...</td>
</tr>
<tr>
<td>Code analysis &amp; Transformations</td>
<td>time;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compile Program</td>
<td></td>
</tr>
<tr>
<td>Code analysis</td>
<td></td>
</tr>
<tr>
<td>Transformations</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Run code</td>
<td></td>
</tr>
<tr>
<td>Run-time environment</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data set</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Run-time analysis</td>
<td></td>
</tr>
<tr>
<td>Run-time state</td>
<td></td>
</tr>
<tr>
<td>Analyze profile</td>
<td></td>
</tr>
</tbody>
</table>

OpenME - simple event-based plugin framework to “open up” applications for external tuning

Connect cM to other third-party tools such as PERISCOPE, SCALSCA, etc ...

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

New idea?

Current non-systematic, tedious R&D: easy to lose time and motivation.
New idea?

Current non-systematic, tedious R&D: easy to lose time and motivation.

Collective Mind Framework
systematize, unify, share, validate experiments and knowledge

Academia / Industry

- download experiment and all cM deps
- replay / validate
- improve
- rank
- use for teaching

Sharing using unified cM web services

Community

Collective Mind Concept

• download experiment and all cM deps
• replay / validate
• improve
• rank
• use for teaching

Industry

Applications
Benchmarks
Codelets

Datasets

Tools

Experimental pipeline

Experimental exploration

Results

Statistical analysis

Models

Abdul Memon and Grigori Fursin
“Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing”
Collective Mind: benefits of common infrastructure

• Researchers can quickly replay, reproduce and validate existing results, and focus their effort on novel approaches combined with data mining, classification and predictive modeling

• Developers can produce tools immediately compatible with collective methodology and infrastructure

• Any person can join collaborative effort to build or extend global expert system that uses Collective Knowledge to:

  • quickly identify program and architecture behavior anomalies
  • suggest better multi-objective program optimizations and hardware configuration for a given user scenario (requirements)
  • suggest run-time adaptation scenarios (co-design and co-optimization)
  • eventually enable self-tuning computer systems
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) Gradually add/expose various program features. Automate process 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 ...
  mult ...
  store ...
```

2) Collect run-time, architecture and OS properties (currently hardware counters and architecture descriptions)

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

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

Validate and improve existing predictive modeling techniques
Collective Mind: current status

- Collective Mind: new plugin-based extensible knowledge management system for collaborative and holistic analysis and tuning of computer systems is currently in validation stage with industrial collaborators and HiPEAC community.
- Simple tool, data and model integration into experimental setups
- Web-services to collect, analyze and visualize data, run and reply experiments, etc
- OpenME interface to “open up” compilers, run-time systems and applications for unified fine-grain analysis and tuning (based on our ICI interface from mainline GCC)
- Hundreds of codelets, thousands of data sets, multiple packages prepared for various research scenarios on data mining
- Automation of experiments, simple record/replay modes
- Plugins for online auto-tuning and predictive modelling
- Portability across all major architectures and OS (Linux, Windows, Android)

Web: cTuning.org
Google groups: ctuning-discussions collective-mind
Twitter: c_tuning grigori_fursin
Events: ADAPT 2014 HiPEAC thematic sessions

Get in touch for more info!
Acknowledgements

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

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