Collective Mind: making auto-tuning practical using crowdsourcing and predictive modeling

Grigori Fursin
INRIA, France

INRIA-Illinois-ANL 10th workshop
Urbana, IL, USA
November 2013
Summary

Challenges:

• How to abstract and unify whole system auto-tuning and modeling?
• How to predict optimizations while helping architecture or compiler designers?
• How to preserve all past tuning knowledge and extrapolate it to the new systems?

• General problems in computer engineering

• Cleaning up research and experimental mess
  ▪ Collective Mind Repository, infrastructure and methodology
  ▪ 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
Problems I have been facing since 1993

User’s task

Solutions

Algorithm

Application

Compilers

Binary and libraries

State of the system

Data set

Run-time environment

Architecture

Result

End-users care about performance, reliability, costs. Technology is secondary!
Problems I have been facing since 1993

Delivering optimal solutions is tough:

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

End-users care about performance, reliability, costs. Technology is secondary!
Summary of current problems

Auto-tuning, machine-learning, dynamic adaptation, 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)
- Very limited training sets (a few benchmarks, datasets, architectures)
- 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 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
Grigori Fursin        “Collective Mind: making auto-tuning practical using crowdsourcing and predictive modeling”

Compiler auto-tuning

Find empirically optimal optimizations in multi-dimensional space while balancing multiple characteristics:
• execution time
• code size
• compilation time

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, 1dataset, 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
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 connect disjoint analysis, tuning, learning tools together with public repository of knowledge?
How to implement?

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\textsubscript{V1}
- Tool A\textsubscript{V2}
- Tool A\textsubscript{VN}
- Tool B\textsubscript{V1}
- Tool B\textsubscript{V2}
- Tool B\textsubscript{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

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

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

**Unified JSON output (meta-data)**

**Generated files**
- Tool B_Vi

**Unified JSON input (if exists)**

**Original unmodified ad-hoc input**

Formalized function (model) of a component behavior

\[ b = B(\vec{c}, \vec{r}, \vec{s}) \]

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

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

**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)*!
**Experiments**

- **Tool wrapper with unified and formalized input and output**
  - **Unified JSON input (meta-data)**
    - Action
    - Behavior
    - Choices
    - Features
    - State
  - **Action function**
    - Parse and unify output
  - **Unified JSON output (meta-data)**

- **Tool B_v1**

- **Generated files**

- **Tool B_{V1}**

- **Tool B_{V2}**

- **Tool B_{VM}**

- **Tool A_v1**

- **Tool A_v2**

- **Tool A_{VN}**

**Formalized function (model) of a component behavior**

\[
\mathcal{B} = B(\mathcal{C}, \mathcal{F}, \mathcal{S})
\]

- **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**
Gradually expose some characteristics                  Gradually expose some choices and features

Compile Program                                        

<table>
<thead>
<tr>
<th>Compile Program</th>
<th>Gradually expose some characteristics</th>
<th>Gradually expose some choices and features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time ...</td>
<td>compiler flags; pragmas ...</td>
</tr>
<tr>
<td></td>
<td>time; CPI, power consumption ...</td>
<td>pinning/scheduling ...</td>
</tr>
<tr>
<td></td>
<td>cost;</td>
<td>architecture; frequency; cache size...</td>
</tr>
<tr>
<td></td>
<td>size; values; description ...</td>
<td>precision ...</td>
</tr>
<tr>
<td></td>
<td>time; size ...</td>
<td>instrumentation; profiling ...</td>
</tr>
</tbody>
</table>

**Combine expert knowledge with automatic feature learning!**

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

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

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)
Found solution

Universal complexity (dimension) reduction

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...

\[
\begin{align*}
\text{Auto-tuning experimental pipeline} & \quad = \\
\text{B(} & \text{c} & \text{)}
\end{align*}
\]

Grigori Fursin

“Collective Mind: making auto-tuning practical using crowdsourcing and predictive modeling”
Active learning to systematize and focus exploration

Start: 50% probability to select optimization (uniform distribution)

Avoiding collection of huge amount of data - filtering (compacting) and learning space on the fly
Active learning to systematize and focus exploration

Current random selection of optimizations increased execution time (bad):

*reduce probabilities of the selected optimizations*
Current random selection of optimizations improved execution time (good):

*reward probabilities of the selected optimizations*
Active learning to systematize and focus exploration

"good optimizations" across all programs:

A – Break up large expression trees
B – Value propagation
C – Hoisting of loop invariants
D – Loop normalization
E – Loop unrolling
F – Mark constant variables
G – Dismantle array instructions
H – Eliminating copies

Faster then traditional search (~50 iterations). Can stuck in local minima
Speedups 1.1-2x. Sometimes better to reduce Intel compiler optimization level!
Active learning to systematize and focus exploration

14 transformations, sequences of length 5, search space = \(396000\)

Universal complexity (dimension) reduction

Found solution


Pruned solution

-03 -fno-align-functions (15% of speedup)
-fdce
-fgcse
-fguess-branch-probability (70% of speedup)
-fmove-loop-invariants
-fomit-frame-pointer
-fmove-ter
-funswitch-loops
-fno ALL

Grigori Fursin
“Collective Mind: making auto-tuning practical using crowdsourcing and predictive modeling”
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

\[ \rightarrow \]

Some ad-hoc predictive model

Some ad-hoc features
Current machine learning usage

Training set: distinct combination of compiler optimizations (clusters)

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

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

MILEPOST GCC
features, hardware counters

Some ad-hoc predictive model
Some ad-hoc features

Optimization cluster

Unseen program

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

Prediction

Optimization cluster

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

Training set: distinct combination of compiler optimizations (clusters)

\[ \overrightarrow{c} (choices) \]
\[ \overrightarrow{f} (features) \]

Optimization cluster

MILEPOST GCC features, hardware counters

Some ad-hoc predictive model

Unseen program

\[ \overrightarrow{f} (features) \]

\[ \overrightarrow{c} (choices) \]

Prediction

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

Previous limited studies
Current machine learning usage

Training set: distinct combination of compiler optimizations (clusters)

$c$ (choices)
$f$ (features)

Optimization cluster

Unseen program

Some ad-hoc predictive model

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

MILEPOST GCC features, hardware counters
**Image B&W threshold filter**

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

<table>
<thead>
<tr>
<th>Class</th>
<th><code>-O3</code></th>
<th><code>-O3 -fno-if-conversion</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data set sample₁</td>
<td>reference execution time</td>
<td>no change</td>
</tr>
<tr>
<td><img src="image1.jpg" alt="Image 1" /></td>
<td><img src="image2.jpg" alt="Image 2" /></td>
<td><img src="image3.jpg" alt="Image 3" /></td>
</tr>
<tr>
<td>Shared data set sample₂</td>
<td>no change</td>
<td>+17.3% improvement</td>
</tr>
<tr>
<td><img src="image4.jpg" alt="Image 4" /></td>
<td><img src="image5.jpg" alt="Image 5" /></td>
<td><img src="image6.jpg" alt="Image 6" /></td>
</tr>
</tbody>
</table>
### Image B&W threshold filter

<table>
<thead>
<tr>
<th>Class</th>
<th>-O3</th>
<th>-O3 -fno-if-conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data set sample&lt;sub&gt;1&lt;/sub&gt;</td>
<td><em>reference execution time</em></td>
<td>no change</td>
</tr>
<tr>
<td>Monitored during <em>day</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared data set sample&lt;sub&gt;2&lt;/sub&gt;</td>
<td>no change</td>
<td>+17.3% improvement</td>
</tr>
<tr>
<td>Monitored during <em>night</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*matrix_ptr2++ = (temp1 > T) ? 255 : 0;*
### Learning feature by domain specialists

**Image B&W threshold filter**

```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><em>reference execution time</em></td>
<td>no change</td>
</tr>
<tr>
<td>Monitored during <em>day</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared data set sample₂</td>
<td>no change</td>
<td>+17.3% <em>improvement</em></td>
</tr>
<tr>
<td>Monitored during <em>night</em></td>
<td></td>
<td></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)**

```c
if get_feature(TIME_OF_THE_DAY)==NIGHT
    bw_filter_codelet_day(buffers);
else
    bw_filter_codelet_night(buffers);
```
Unexpected behavior - expose to the community including domain specialists, explain, find missing feature and add to the system
Normality test plugin

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

![Graph showing Normality test plugin results]

**Class A**

- **CPU Frequency**: 800MHz
- **Execution time (sec.)**

**Class B**

- **CPU Frequency**: 2400MHz
- **Execution time (sec.)**
How we can explain the following observations for some piece of code (“codelet object”)?

(LU-decomposition codelet, Intel Nehalem)
Using Collective Mind to explore and learn behavior of computer systems

Add 1 property: matrix size
Either fit existing or build a new model to correlate objectives (CPI) and features (matrix size) while minimizing RMSE.

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!
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.
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

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

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

Compact data on-line before sharing with other users!
Many new research and development opportunities

• Researchers can quickly replay, reproduce and validate existing results, and focus their effort on either feature learning and predictive models or on novel approaches combined with auto-tuning and machine learning

• 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
Gradually and collaboratively increase granularity and complexity

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

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

- Infrastructure is available under standard BSD license at http://cTuning.org/tools/cm
- Pilot repository is available at http://c-mind.org/repo

(hundreds of kernels, thousands of datasets, tools, models, etc)

- 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>Validation in industry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproducible and collaborative research; 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>REPRODUCE 2014 @ HPCA 2014 <a href="http://www.occamportal.org/reproduce">www.occamportal.org/reproduce</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://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
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