Collective Mind
a collaborative curation tool for program optimization

Towards collaborative, systematic and reproducible design and optimization of computer systems

Grigori Fursin
INRIA, France

SEA 2014, Boulder, USA
April 2014
Outline

• Interdisciplinary background

• Back to basics: major problems in computer engineering

• Collaborative and Reproducible Research and Development:
  ▪ cTuning.org framework and repository
  ▪ Exposed problems during practical usage
  ▪ Collective Mind framework and repository
  ▪ Collaborative research usage scenarios
  ▪ Collaborative curation for program optimization
  ▪ Changing R&D methodology and publication model

• Conclusions and future work
Back to 1993: physics, electronics, machine learning

Semiconductor neuron - base of neural accelerators and possible future neuro computers

Modeling and understanding brain functions

My problem with modeling:
- Slow
- Unreliable
- Costly
I needed help from computer scientists to find the best solutions for my tasks!

Back to basics: task pipeline (workflow)

Available solutions

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

Result

Service/application providers
(cloud computing, 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
(processors, memory, interconnects, languages, compilers, run-time systems)
Back to basics: ever rising complexity of computer systems

Deliver universal and optimal optimization heuristic is often impossible!

1) Too many design and optimization choices at all levels

2) Always multi-objective optimization: performance vs compilation time vs code size vs system size vs power consumption vs reliability vs return on investment

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

4) Users and developers often have to resort to empirical auto-tuning

Empirical auto-tuning is too time consuming, ad-hoc and tedious to be a mainstream!
Many end users and service providers run numerous applications on different architectures with different data sets, run-time systems, compilers and optimizations!

• Can we leverage their experience and computational resources?

• Can we build common research and experimentation infrastructure connected to a public repository of knowledge?

• Can we extrapolate collected knowledge using machine learning to predict optimal program optimizations, hardware designs and run-time adaptation scenarios?

*Proposed and developed in the MILEPOST/cTuning project (2006-2009) to automate and crowdsourcing training of a machine learning based self-tuning compiler for reconfigurable processors (SW/HW co-design).*
**cTuning.org: plugin-based auto-tuning and learning framework**

**cTuning fixed workflow**

- **cTuning auto-tuning plugin**: explore design and optimization space
  
  Program
  
  **cTuning wrapper**: extract program semantic (static) features using MILEPOST GCC with ICI
  
  **cTuning compile wrapper**
  
  **cTuning run wrapper**
  
  **cTuning third-party analysis plugins**: profile and collect hardware counters (dynamic features) using gprof, perf, PAPI, etc
  
  **Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc
  
  **Machine learning plugins**: KNN, SVM ...

**Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)**
cTuning.org: plugin-based auto-tuning and learning framework

**cTuning fixed workflow**

- **cTuning auto-tuning plugin**: explore design and optimization space
- **cTuning wrapper**: extract program semantic (static) features using MILEPOST GCC with ICI
- **cTuning compile wrapper**
- **cTuning run wrapper**
- **cTuning third-party analysis plugins**: profile and collect hardware counters (dynamic features) using gprof, perf, PAPI, etc
- **Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc
- **Machine learning plugins**: KNN, SVM ...

**Plugins with strategies**:
- random, hill climbing, probabilistic, genetic ...

**Shared benchmarks**, codelets, kernels from SVN (cTuning.org/cbench)

**Multiple shared compiler descriptions** for GCC, ICC, Open64, LLVM, PathScale, XLC... (txt)

**Shared datasets** from SVN (cTuning.org/cdatasets)

**OS and platform shared descriptions** (txt)

**Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc

**Common open-source cTuning CC framework** (http://cTuning.org/ctuning-cc)

**Shared material in various formats from multiple sources**
**cTuning.org:** plugin-based auto-tuning and learning framework

**cTuning fixed workflow**

- **cTuning auto-tuning plugin:** explore design and optimization space
- **Program**
- **cTuning wrapper:** extract program semantic (static) features using MILEPOST GCC with ICI
- **cTuning compile wrapper**
- **cTuning run wrapper**
- **Data set**
- **cTuning third-party analysis plugins:** profile and collect hardware counters (dynamic features) using gprof, perf, PAPI, etc
- **Pareto frontier plugin:** balance exec.time, code size, compilation time, power consumption, etc
- **Machine learning plugins:** KNN, SVM...

**Plugins with strategies:** random, hill climbing, probabilistic, genetic ...

**Shared benchmarks, codelets, kernels from SVN** (cTuning.org/cbench)

**Multiple shared compiler descriptions** for GCC, ICC, Open64, LLVM, PathScale, XLC... (txt)

**Shared datasets** from SVN (cTuning.org/cdatasets)

**OS and platform shared descriptions** (txt)

**Pareto frontier plugin:** balance exec.time, code size, compilation time, power consumption, etc

**Machine learning plugins:** KNN, SVM...

**MySQL cTuning database:** optimization statistics from multiple users

**cTuning unified php-based web services**

**Interdisciplinary crowd**

Collaboratively searching for best optimizations and learning techniques

Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)
In 2009, we managed to automatically tune customer benchmarks and compiler heuristics for a range of real platforms from IBM and ARC (Synopsis)

Machine learning plugins: KNN, SVM ...

Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)

Everything is solved?
cTuning practical usage exposed a few problems

Many thanks to international academic and industrial users for their feedback
http://cTuning.org/lab/people

Thanks to Davide Del Vento and his interns from UCAR
cTuning practical usage exposed a few problems

- Ever changing environment
  (SW and HW choices, features, tools and libraries versions, interfaces, data formats ...)
  - Difficult to reproduce experimental results from different users
  - Difficult to keep cTuning up-to-date - by the end of development cycle, new software and hardware is often available

- Complex deployment and maintenance
  - Complex, manual configuration of the framework, repository, plugins, hardware and all dependencies
  - Many used languages (C, C++, PHP, Java) and data exchange formats (txt, csv, xls, ...)
  - Different versions of benchmarks, data sets, plugins, models, compilers, third-party tools, etc are scattered around many directories and disks
cTuning practical usage exposed a few problems

- Difficult or impossible to reproduce and reuse techniques from existing publications
  - No culture of sharing code and data in computer engineering unlike other sciences
  - Impossible to publish negative results or just validation of existing works
  - Academia focuses more and more on quantity (number of publications and citations) rather than quality and reproducibility of results

- No simple and scalable mechanism to preserve, share and reuse all heterogeneous experimental material
  - Centralized, MySQL-based repository got quickly saturated when collecting and processing data from many users
  - MySQL repository is difficult to adapt to continuous changes in the system
  - Difficult to preserve different versions of benchmarks, data sets, plugins and tools, and make them co-exist using SVN repositories
cTuning practical usage exposed a few problems

- Difficult or impossible to reproduce and reuse techniques from existing publications
- No culture of sharing code and data in computer engineering unlike other sciences
- Impossible to publish negative results or just validation of existing works
- Academia focuses more and more on quantity (number of publications and citations) rather than quality and reproducibility of results
- No simple and scalable mechanism to preserve, share and reuse all heterogeneous experimental material
- Centralized, MySQL-based repository got quickly saturated when collecting and processing data from many users
- MySQL repository is difficult to adapt to continuous changes in the system
- Difficult to preserve different versions of benchmarks, data sets, plugins and tools, and make them co-exist using SVN repositories

We face typical “big data” problem!
Collective Mind: attempt to fix cTuning problems

**cTuning fixed workflow**

- **cTuning auto-tuning plugin**: explore design and optimization space
- **Program**
- **cTuning wrapper**: extract program semantic (static) features using MILEPOST GCC with ICI
- **cTuning compile wrapper**
- **cTuning run wrapper**
- **Data set**
- **cTuning third-party analysis plugins**: profile and collect hardware counters (dynamic features) using gprof, perf, PAPI, etc
- **Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc
- **Machine learning plugins**: KNN, SVM ...

**Plugins with strategies**:
- random, hill climbing, probabilistic, genetic ...

**Shared benchmarks**, codelets, kernels from SVN (cTuning.org/cbench)

**Multiple shared compiler descriptions** for GCC, ICC, Open64, LLVM, PathScale, XLC... (txt)

**Shared datasets** from SVN (cTuning.org/cdatasets)

**OS and platform shared descriptions** (txt)

**Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc

**Machine learning plugins**: KNN, SVM ...

**cTuning.org**

- **MySQL cTuning database**: optimization statistics from multiple users
- **Interdisciplinary crowd**
- **Collaboratively searching for best optimizations and learning techniques**

**Common open-source Collective Mind framework** (http://c-mind.org)
Collective Mind: attempt to fix cTuning problems

Collective Mind flexible pipeline

**cTuning auto-tuning plugin:** explore design and optimization space

**Program**

**cTuning wrapper:** extract program semantic (static) features using

**Unify plugins and tool wrappers:**

- cm <plugin_name> params -- orig. CMD
  
  **Use high-level, portable language**

**Pareto frontier plugin:** balance exec.time, code size, compilation time, power consumption, etc

**Machine learning plugins:** KNN, SVM ...

**Plugins with strategies:** random, hill climbing, probabilistic, genetic ...

**Shared benchmarks,**

- codelets, kernels from SVN (cTuning.org/cbench)

**Multiple shared compiler descriptions** for GCC, ICC, Open64, LLVM, PathScale, XLC... (txt)

**Shared datasets** from SVN (cTuning.org/cdatasets)

**OS and platform shared descriptions** (txt)

**Pareto frontier plugin:** balance exec.time, code size, compilation time, power consumption, etc

**Machine learning plugins:** KNN, SVM ...

**MySQL cTuning database:** optimization statistics from multiple users

**Interdisciplinary crowd**

**Collaboratively searching for best optimizations and learning techniques**

**Shared material in various formats from multiple sources**

**Common open-source Collective Mind framework** (http://c-mind.org)
Collective Mind: attempt to fix cTuning problems

Collective Mind flexible pipeline

- **cTuning auto-tuning plugin**: explore design and optimization space
- **cTuning wrapper**: extract program semantic (static) features using
- **Pareto frontier plugin**: balance exec.time, code size, compilation time, power consumption, etc
- **Machine learning plugins**: KNN, SVM...

Plugins with strategies:
- Random, hill climbing, probabilistic, genetic...

Shared benchmarks, codelets, kernels from SVN (cTuning.org/cbench)

New repository which can keep both collections of files and meta-description in some unified format (txt)

Shared material in various formats from multiple sources

Machine learning plugins: KNN, SVM ...

MySQL cTuning database: optimization statistics from multiple users

Interdisciplinary crowd

Collaborative curation of all research material

Common open-source Collective Mind framework (http://c-mind.org)
JSON: extensible and human-readable format

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

Python: high-level, portable, popular language with JIT, JSON support and many statistical and machine learning libraries

```python
import json
f=file('data.json')
array=json.loads(f.read())
f.close()
et=array['characteristics']['execution_times']
for tm in et:
    print 'Execution time:', tm
```

Can be directly indexed using web-based, distributed ElasticSearch (Apache2 license)
**Convert ad-hoc experiments to cM python modules**

**Offline tuning**

- User program, dataset
- Ad-hoc tuning scripts
- Compiler\(_i\)
  - Compiler\(_2\)
  - Compiler\(_N\)
- Run-time\(_i\)
- Ad-hoc analysis scripts
- Collection of CSV, XLS, TXT and other files

**Online tuning**

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

Meta description that should be exposed in the information flow for auto-tuning and machine learning
Convert ad-hoc experiments to cM python modules

Offline tuning

- User program, dataset
- Ad-hoc tuning scripts
- Compiler\(_1\)
- Compiler\(_2\)
- Compiler\(_N\)
- Run-time\(_1\)
- Run-time\(_2\)
- Ad-hoc analysis scripts
- Collection of CSV, XLS, TXT and other files

Online tuning

Process CMD

- cM module (wrapper) with unified and formalized input and output

Generated files

Tool B\(_V_i\)

Original unmodified ad-hoc input

Behavior

Choices

Features

State
Convert ad-hoc experiments to cM python modules

User program, dataset
Ad-hoc tuning scripts
Compiler$_1$
Compiler$_2$
Compiler$_N$
Run-time$_1$
Run-time$_2$
Ad-hoc analysis scripts
Collection of CSV, XLS, TXT and other files

Offline tuning
Online tuning

Unified JSON input (if exists)
Original unmodified ad-hoc input
Unified JSON input (meta-data)
Action
Behavior
Choices
Features
State
Action function
Parse and unify output
Unified JSON output (meta-data)
Tool B$_{Vi}$
Generated files

Formalized function (model) of a component behavior

$\vec{b} = B(\vec{c}, \vec{f}, \vec{s})$

Flattened JSON vectors (either string categories or integer/float values)
Convert ad-hoc experiments to cM python modules

Offline tuning

Unified JSON input (if exists)

Original unmodified ad-hoc input

Process CMD

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)

Tool \( B_{Vi} \)

Generated files

Formalized function (model) of a component behavior

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

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

Multiple tool versions can co-exist, while their interface is abstracted by cM module
Convert ad-hoc experiments to cM python modules

Offline tuning

Online tuning

User program, dataset
Ad-hoc tuning scripts
Compiler₁
Run-time₁
Ad-hoc analysis scripts
Compiler₂
Run-time₂

Collection of CSV, XLS, TXT and other files

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

Action function
Set environment for a given tool version

Unified JSON output (meta-data)

Tool Bᵥᵢ → Generated files

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)
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)!
Data abstraction in Collective Mind

- cM module
  - compiler

- JSON meta-description
  - Compiler flags
    - GCC 4.4.4
    - GCC 4.7.1
    - LLVM 3.1
    - LLVM 3.4
### Data abstraction in Collective Mind

<table>
<thead>
<tr>
<th>cM module</th>
<th>Files, directories</th>
<th>JSON meta-description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>package</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GloC 4.4.4</td>
<td>GCC 4.4.4</td>
</tr>
<tr>
<td></td>
<td>GloC 4.7.1</td>
<td>GCC 4.7.1</td>
</tr>
<tr>
<td></td>
<td>GloC 4.7.1 source</td>
<td>LLVM 3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LLVM 3.4</td>
</tr>
<tr>
<td></td>
<td>GloC 4.7.1</td>
<td>gmp 5.0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mpfr 3.1.0</td>
</tr>
<tr>
<td></td>
<td>GloC 4.7.1</td>
<td>lapack 2.3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>java apache commons codec 1.7</td>
</tr>
</tbody>
</table>

#### Compiler flags

- GCC 4.4.4
- GCC 4.7.1
- LLVM 3.1
- LLVM 3.4

#### Installation info

- ...
### Data Abstraction in Collective Mind

**cM Module Structure:**

- **Compiler**
- **Package**
- **Dataset**
- **Module**

**Files, Directories:**

- GCC 4.7.1 bin
- GCC 4.7.1 source
- LLVM 3.4
- gmp 5.0.5
- mpfr 3.1.0
- lapack 2.3.0
- Java Apache Commons Codec 1.7
- Image (jpeg-0001)
- bzip2-0006
- txt-0012
- Compiler
- Package
- Dataset

**JSON Meta-Description:**

- GCC 4.4.4
- GCC 4.7.1
- LLVM 3.1
- LLVM 3.4

**Compiler Flags**

**Installation Info**

**Features**

**Actions**

**Repository Directory Structure:**

- `.cmr`
- `/module UOA` / `data UOA (UID or alias)` / `.cm` / `data.json`
Data abstraction in Collective Mind

**cM module**
- compiler
- package
- dataset
- module

**Files, directories**
- GCC 4.7.1 bin
- GCC 4.4.4
- GCC 4.7.1
- LLVM 3.1
- LLVM 3.4

**JSON meta-description**
- Compiler flags
- Installation info
- Features
- Actions

---

Now can reference and find any data by CID (similar to DOI):

<module UOA : data UOA>

---

.cm / module UOA / data UOA (UID or alias) / .cm / data.json
## Data abstraction in Collective Mind

### cM module
- **compiler**
- **package**
- **dataset**
- **module**

### Files, directories
- **gcc 4.4.4**
- **gcc 4.7.1**
- **llvm 3.1**
- **llvm 3.4**

### JSON meta-description
- **gcc 4.7.1 bin**
- **gcc 4.7.1 source**
- **llvm 3.4**
- **gmp 5.0.5**
- **mpfr 3.1.0**
- **lapack 2.3.0**
- **java apache commons codec 1.7**

### Dependencies between data and modules

### Compiler flags

### Installation info

### Features

### Actions

### Repository directory structure:

- **cmr**
- **/module UOA**
- **/data UOA (UID or alias)**
- **/.cm**
- **/data.json**

---

Grigori Fursin  
“Collective Mind: a collaborative curation tool for program optimization”  
SEA’14, April 2014, Boulder, CO, USA
Gradually adding specification (agile development)

cTuning experiment module data.json

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

cTuning experiment module data.json

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

cM flattened JSON key

```text
###characteristics#execution_times@1

`flattened_json_key`:

- `type`: "text" | "integer" | "float" | "dict" | "list"
- `uid`
- `characteristic`: "yes" | "no"
- `feature`: "yes" | "no"
- `state`: "yes" | "no"
- `has_choice`: "yes" | "no"
- `choices`: [ list of strings if categorical choice],
- `explore_start`: "start number if numerical range",
- `explore_stop`: "stop number if numerical range",
- `explore_step`: "step if numerical range",
- `can_be_omitted`: "yes" | "no"
```
Gradually adding, extending and improving modules and data

<table>
<thead>
<tr>
<th>Category</th>
<th>cM module</th>
<th>Module actions</th>
<th>All data</th>
<th>Meta-description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Third-party tools, libraries</td>
<td>package</td>
<td>common*, install</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-level algorithms</td>
<td>algorithm</td>
<td>common*, transform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications, benchmarks, codelets, kernels</td>
<td>code.source</td>
<td>common*, build</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto-tuning compilers</td>
<td>ctuning.compiler</td>
<td>common*, compile_program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler machine learning plugins</td>
<td>math.model</td>
<td>common*, build, predict, fit, detect_representative_points</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binaries and libraries</td>
<td>code</td>
<td>common*, run</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data sets</td>
<td>dataset</td>
<td>common*, create</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating Systems</td>
<td>os</td>
<td>common*, detect_host_family</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processors</td>
<td>processor</td>
<td>common*, detect_host_processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statistical and data mining plugins</td>
<td>math.statistics.r</td>
<td>common*, analyze</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* add, list, view, copy, move, search

cM repository directory structure: .cmr / module UOA (UID or alias) / data UOA / .cm / data.json
Simple and minimalistic high-level cM interface - **one function (!)**

(python dictionary) \textit{output} = \texttt{cm\_kernel\_access} ( (python dictionary) \textit{input} )

Load data:

\begin{verbatim}
r = cm_kernel.access({‘cm_run_module_uoa’:’dataset’,
                     ‘cm_action’:’load’,
                     ‘cm_data_uoa’:’image-jpeg-0001’})
if r[‘cm_return’]>0: return r
d=r[‘cm_data_obj’][‘cfg’]
\end{verbatim}

Call R machine learning module:

\begin{verbatim}
r = cm_kernel.access({‘cm_run_module_uoa’:’cm math.model.r’,
                     ‘cm_action’:’build’,
                     ‘model_name’:’earth’, …})
if r[‘cm_return’]>0: return r
\end{verbatim}
Assembling, preserving, sharing and extending the whole pipeline as “LEGO”

Experiments

- Tool A_{V1}
- Tool A_{V2}
- Tool A_{VN}
- Tool B_{V1}
- Tool B_{V2}
- Tool B_{VM}

Ad-hoc tuning scripts
Ad-hoc analysis and learning scripts
Collection of CSV, XLS, TXT and other files

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

Formalized function (model) of a component behavior
\vec{b} = B(\vec{c}, \vec{f}, \vec{s})

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

Chaining cM 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 | time ... | compiler flags; pragmas ... |

**I now gradually convert to Collective Mind and validate past research techniques with the great help of volunteers (sadly such work is usually not funded, can’t be easily published, and is often considered as a waste of time by academic community)**

**Methodology similar to physics:**
**start from coarse-grain and gradually move to fine-grain level!**

<table>
<thead>
<tr>
<th>Run code</th>
<th>Run-time environment</th>
<th>time; CPI, power consumption ...</th>
<th>pinning/scheduling ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td></td>
<td>cost;</td>
<td>architecture; frequency; cache size...</td>
</tr>
<tr>
<td>Data set</td>
<td></td>
<td>size; values; description ...</td>
<td>precision ...</td>
</tr>
</tbody>
</table>

| Analyze profile | time; size ... | instrumentation; profiling ... |

Start coarse-grain decomposition of a system (detect coarse-grain effects first). Add universal learning modules.
Growing, plugin-based cM pipeline for auto-tuning and learning

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

http://c-mind.org/ctuning-pipeline
Our Collective Mind Buildbot and plugin-based auto-tuning pipeline 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
Implemented scenario: 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

**Use Pareto frontier filter;**

**Pack experimental data on the fly**

**Execution time (sec.)**
Collective Mind agile research and development

From agile development to agile research!

Community shares, validates and improves benchmarks, data sets, tools, design and optimization choices, features, characteristics, models, ...

Collective Mind experimental pipelines can now survive changes in the system! Modules and data can evolve with the evolution of the technology!
Reproducibility of experimental results

Reproducibility came as a side effect!

• Can preserve the whole experimental setup with all data and software dependencies
• Can perform statistical analysis (normality test) for characteristics
• Community can add missing features or improve machine learning models
Reproducibility of experimental results

Reproducibility came as a side effect!

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

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

Reproducibility came as a side effect!

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

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

![Graph showing distribution of execution time for Class A and Class B with CPU frequencies of 800MHz and 2400MHz.](image-url)
My dream: systematic, collaborative and reproducible computer engineering

- Prototype research idea
- Validate existing work
- Perform end-user task

- Quick, non-reproducible hack?
- Ad-hoc heuristic?
- Quick publication?
- No shared code and data?

Current state of computer engineering

- Share code, data with their meta-description and dependencies
- Systematize and classify collected optimization knowledge
- Develop and preserve the whole experimental pipeline
- Validate experimental results by the community
- Extrapolate collected knowledge to build faster, smaller, more power efficient and reliable computer systems

Collaborative Infrastructure and repository

c-mind.org/repo

Sharing of code and data

Classification, predictive modeling

Systematization and unification of collected knowledge (big data)

“crowd”

Grigori Fursin  “Collective Mind: a collaborative curation tool for program optimization”  SEA’14, April 2014, Boulder, CO, USA 42
## Current status and future work

- Pilot live repository for public curation of research material: [http://c-mind.org/repo](http://c-mind.org/repo)
- Infrastructure is available at SourceForge under standard BSD license: [http://c-mind.org](http://c-mind.org)
- Example of crowdsourcing compiler flag auto-tuning using mobile phones: “Collective Mind Node” in Google Play Store
- Several publications under submission
- Raising funding to make cM more user friendly and add more research scenarios

<table>
<thead>
<tr>
<th>Education</th>
<th>Academic research</th>
</tr>
</thead>
<tbody>
<tr>
<td>New publication model where research material and experimental results are shared, validated and reused by the community <a href="http://ctuning.org/reproducibility">http://ctuning.org/reproducibility</a></td>
<td>Systematizing, validating, sharing past research techniques on auto-tuning and machine learning through cM pipeline</td>
</tr>
<tr>
<td><strong>Panel at ADAPT 2014 @ HiPEAC 2014</strong> <a href="http://adapt-workshop.org">http://adapt-workshop.org</a> (this year we attempted to reproduce some submitted articles)</td>
<td>• Finding representative benchmarks and data sets</td>
</tr>
<tr>
<td></td>
<td>• Run-time adaptation and ML</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

- Colleagues from NCAR (USA): Davide Del Vento et al.
- Colleagues from STMicroelectronics (France):
  Christophe Guillone, Antoine Moynault, Christian Bertin
- Colleagues from ARM (UK): Anton Lokhmotov
- 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
- INRIA fellowship (2012-2016)

Thank you for your attention!

Questions and comments: grigori.fursin@inria.fr