Systematizing tuning of computer systems using crowdsourcing and statistics

Grigori Fursin
INRIA, France

HPSC 2013, Taiwan
March 2013
Messages

1st talk (Wednesday)
Systematizing tuning of computer systems using crowdsourcing and statistics

• Revisiting current design and optimization methodology
• Leveraging experience and computer resources of multiple users
• Using predictive modelling and data mining to improve computer systems

2nd talk (Friday)
Collective Mind: novel methodology, framework and repository to crowdsource auto-tuning

• New plugin-based extensible infrastructure and repository for collaborative analysis and tuning of computer systems - will be released in May 2013
• “Big data” enables cooperation between architecture, compiler, OS and application designers and mathematicians
• Examples of auto-tuning and predictive modeling for numerical kernels
Motivation: back to 1993

Semiconductor neural element - possible base of neural computers

Modeling and understanding brain functions

- Slow
- Unreliable
- Costly
Motivation: back to basics

End users

Task

Solution

User requirements:

- most common:
  - minimize all costs
    (time, power consumption,
    price, size, faults, etc)
  - guarantee real-time constraints
    (bandwidth, QOS, etc)

Result
Motivation: back to basics

User requirements:
most common:
minimize all costs
(time, power consumption, price, size, faults, etc)
guarantee real-time constraints (bandwidth, QOS, etc)

Decision
(depends on user requirements)

Available choices
(solutions)

Result
**Motivation: back to basics**

**End users**

**Task**

![Circuit Diagram]

**Solution**

**User requirements:**

*most common:*

- minimize all costs (time, power consumption, price, size, faults, etc)
- guarantee real-time constraints (bandwidth, QOS, etc)

**Decision** (depends on user requirements)

**Available choices (solutions)**

**Result**

*Should provide choices and help with decisions*

*Hardware and software designers*
Solution

Motivation: back to basics

End users

Task

User requirements:
most common:
minimize all costs (time, power consumption, price, size, faults, etc)
guarantee real-time constraints (bandwidth, QOS, etc)

Decision
(does not on user requirements)

Result

Available choices (solutions)

Service/application providers
(supercomputing, cloud computing, mobile systems)

Should provide choices and help with decisions

Hardware and software designers

End users

Task

User requirements:
most common:
minimize all costs (time, power consumption, price, size, faults, etc)
guarantee real-time constraints (bandwidth, QOS, etc)

Decision
(does not on user requirements)

Result

Available choices (solutions)
Available solutions: hardware

Companies compete hard to deliver many solutions with various characteristics: 

*performance, power consumption, size, bandwidth, response time, reliability, cost ...*
Software developers try to keep pace and produce various algorithms, programming models, languages, analysis tools, compilers, run-time systems, databases, etc.
Challenges

Task

Solutions

Result
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.

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

**Task**

**Solutions**

**Result:**

- 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**
Grigori Fursin
“Systematizing tuning of computer systems using crowdsourcing and statistics”
HPSC 2013, NTU, Taiwan
March, 2013

Challenges

- 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

Use auto-tuning:
Explore multiple choices empirically: learn behavior of computer systems across executions

Result

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 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

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 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

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

• Limited training sets

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

• No knowledge sharing
Attempts to solve these problems: co-design

Treat computer system as a black box

Task

Result

Co-design:
Explore choices and behavior of the whole system.

Showed high potential in the last years but ...
Attempts to solve these problems: co-design

Co-design is currently a buzz word and a hot research topic but still far from the mainstream.

Why?

- Even more choices to explore and analyze
- Often impossible to expose tuning choices or obtain characteristics at all levels
- Limited training sets
- Still no knowledge sharing
Got stuck with a limited number of benchmarks, datasets, architectures and a large number of optimizations and generated data - 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 transparently distribute optimization and learning across many users?
Challenges

How can we evaluate optimizations in a realistic environment without complex recompilation frameworks and without source code?
Challenges

First problem:
need reference run with the same dataset

User application

Hot function $O_{\text{ref}}$

Speed up

User application

Hot function $O_{\text{new}}$
Challenges

Second problem: variation in execution time due to different run-time states

User application

Hot function

$O_{ref}$

30 repetitions

Statistical characteristic: Program execution time (with module overhead) (sec.)(MIN)
Challenges

Second problem: variation in execution time due to different run-time states

User application

Hot function $O_{ref}$

![Graph](image)
Challenges

How can we evaluate some optimization in a realistic environment?

Second problem: variation in execution time due to different run-time states
(parallel processes, adaptive scheduling, pinning, cache state, bus state, frequency changes, etc)
Our approach: static multiversioning

Application

Select most time consuming code sections
Our approach: static multiversioning

Application

Create multiple versions of time consuming code sections
Our approach: static multiversioning

Add monitoring routines

Application

monitor_start

monitor_stop

monitor_start

monitor_stop
Our approach: static multiversioning

Application

Apply various transformations over clones of code sections
Our approach: static multiversioning

Select global or fine-grain internal compiler (or algorithm) optimizations

Application

Apply various transformations over clones of code sections

monitor_start

monitor_stop

monitor_start

monitor_stop
Our approach: static multiversioning

Application

monitor_start

monitor_stop

monitor_start

monitor_stop

Apply various transformations over clones of code sections
Our approach: static multiversioning

Different ISA; manual transformations, etc

Application

Apply various transformations over clones of code sections

monitor_start

monitor_stop

monitor_start

monitor_stop
Our approach: static multiversioning

Final instrumented program

Application

```
monitor_start

monitor_stop
```

```
monitor_start

monitor_stop
```
Observations: program execution phases

IPC for subroutine resid of benchmark mgrid across calls

- Define stability by 3 consecutive or periodic executions with the same IPC
- Predict further occurrences with the same IPC (using period and length of regions with stable performance)
• Define stability by 3 consecutive or periodic executions with the same IPC

• Predict further occurrences with the same IPC (using period and length of regions with stable performance)
Observations: program execution phases

Some programs exhibit stable behavior

1) Consider clone with new optimization is evaluated after 2 consecutive executions of the code section with the same performance
2) Ignore one next execution to avoid transitional effects
3) Check baseline performance (to verify stability prediction)
• Can transparently to end-user evaluate multiple optimizations

• Statically enable adaptive binaries (that can react to dataset or run-time state changes without any need for JIT or other complex frameworks)
• Can transparently to end-user evaluate multiple optimizations

• Statically enable adaptive binaries (that can react to dataset or run-time state changes without any need for JIT or other complex frameworks)

Observations: random behavior

Randomly select versions at run-time
Monitor speedup variation over time

jpeg decoder, GCC 4.5, Intel architecture
Grigori Fursin
“Systematizing tuning of computer systems using crowdsourcing and statistics”
HPSC 2013, NTU, Taiwan
March, 2013

**Collective Compiler**
- GCC Interface:
  - create code clones
  - Apply optimizations per clone
  - intercept \texttt{main()} and add auxiliary routines

**Binary**
Function clones with different optimizations
- Profiling Routines
- Collective Stats
- Unique IDs

**Execution**
- Start profiling and
- Randomly select version (original or clone)

**Dataset**
1. Dataset$_1$
2. Dataset$_N$

**Prolog of the time consuming code**
**Cloned code** (Optimizations$_1$)
**Original code** (Optimizations$_2$)

**Stop profiling**
**Epilog of the time consuming code**

**Intercept \texttt{exit()} and call Collective Stats Handler**
Transparencyly measuring the impact of optimizations

Collective Compiler

- **GCC Interface:**
  - create code clones
  - Apply optimizations per clone
  - intercept `main()` and add auxiliary routines

Binary

- Function clones with different optimizations
  - Profiling Routines
  - Collective Stats
  - Unique IDs

Execution

- **Prolog of the time consuming code**
- **Start profiling and**
- **Randomly select version (original or clone)**

- Original code (Optimizations₁)
- Cloned code (Optimizations₂)

- **Epilog of the time consuming code**

Collective Compiler

- **Initiate recompilation if better optimization setting**
  is suggested based on Collective Knowledge

Collective Optimization

- **Database**
  - COMPILATION table
  - EXECUTION table
  - AUXILIARY tables

MySQL

Collective Optimization

- **Web Services**
  - Register events
  - Query database
  - Get statistics
  ...

Web Server

Network

Dataset₁

Dataset₂

Userₓ

Programₐ

Archₐ

Userᵧ

Programᵧ

Archᵧ

cTuning.org

Grigori Fursin        “Systematizing tuning of computer systems using crowdsourcing and statistics”            HPSC 2013, NTU, Taiwan          March, 2013
Speeding up research (2005-cur.)

• Can observe behavior and evaluate optimizations in various GRID servers, cloud services, desktops, etc …
  
  • multiple benchmarks/datasets
  • multiple architectures
  • multiple compilers
  • multiple optimizations

Opened up many interesting research opportunities, particularly for data mining and predictive modeling!


  Concept is included into EU HiPEAC research vision 2012-2020

Collaborative exploration of large optimization spaces

Multi-objective optimizations (depend on user scenarios):

HPC and desktops: *improving execution time*

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

Embedded systems: *improving execution time and code size*

Now additional requirement: *reduce power consumption*

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

susan corners kernel
Intel Core2
GCC 4.4.4
similar results on ICC 11.1
baseline opt=-O3
~100 optimizations
random combinations
(50% probability)
Online focused exploration and learning

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

Avoiding collection of huge amount of data - filtering (compacting) and learning space on the fly
Online focused exploration and learning

Current random selection of optimizations reduced execution time:

\textit{reduce probabilities of the selected optimizations}
Online focused exploration and learning

Current random selection of optimizations improved execution time:

*reward probabilities of the selected optimizations*
Online focused exploration and learning

Faster than traditional search (~50 iterations). Can stuck in local minima
Speedups 1.1-2x. Sometimes better to reduce Intel compiler optimization level!

“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
14 transformations, sequences of length 5, search space = 396000

Online probabilistic exploration

AMD platform, GCC 4.5, image corner detection (susan_corners)
Reactions to optimizations across multiple datasets

MiBench, 20 datasets per benchmark, 200/1000 random combination of Open64 (GCC) compiler flags, 5 months of experiments

http://ctuning.org/cbench

dijkstra
(not sensitive)

jpeg_d
(clustering)
Unifying adaptation of statically compiled programs

Statically-compiled adaptive binaries and libraries

Original hot function

Function Version_1
Function Version_2
...
Function Version_N

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

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

Extract dataset features

Selection mechanism optimized for low run-time overhead

Original hot function

Function Version₁

Function Version₂

... Function Versionₙ

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

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

Iterative /collective compilation with multiple datasets

Step 3

Unifying adaptation of statically compiled programs
Unifying adaptation of statically compiled programs

Statically-compiled adaptive binaries and libraries

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

Selection mechanism optimized for low run-time overhead

- Original hot function
- Function Version$_1$
- Function Version$_2$
- ... Function Version$_N$

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

Dynamic

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

Iterative /collective compilation with multiple datasets
Online tuning: adaptive scheduling

- Why do we need dynamic predictive scheduling?
  - Best fitting PE depends on the code, the input and the system current workload
- Example (matrix multiplication on single-CPU/GPU):

![Graph showing matrix multiplication time vs size for CPU and GPU.](image)
Online tuning: adaptive scheduling

- Granularity
  - Function-level
- Function versioning
  - Orthogonal to the scheduling problem
- Explicit data transfer

1. Obtain both the CPU and the GPU versions from a current implementation

```c
void matmul(float* A, float* B, float* C, int N);
void matmul_cpu(float* A, float* B, float* C, int N);
void matmul_gpu(float* A, float* B, float* C, int N);
```

versioning (CUDA, MCUDA)

2. Change calls to that function by a request to the scheduler (+ wrapper)

```c
CallScheduler* cs = CallScheduler::getInstance();
MatmulFunc* f = new MatmulFunc;
MatmulArgs* a = new MatmulArgs(A,B,C,N);
```

```c
class MatmulFunc : public Func {
    void run(PE::Type peType, Args* args) {
        MatmulArgs* a = (MatmulArgs*) args;
        switch (peType) {
        case PE::CPU:
            matmul_cpu(a->A,a->B,a->C,a->N);
            break;
        case PE::GPU:
            matmul_gpu(a->A,a->B,a->C,a->N);
            break;
        }
    }
};
```
Online tuning: adaptive scheduling

- Estimate history (estimate-hist)
  1. Performance history table for every pair <function, PE>
  2. Predict the waiting time for a new task in each PE's queue
  3. Assign the task to the queue with the minimum waiting time

- Tries to reduce load unbalance

<table>
<thead>
<tr>
<th>CPU history</th>
<th>GPU history</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>func</td>
</tr>
<tr>
<td>matmul</td>
<td>matmul</td>
</tr>
<tr>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>time\textunderscore{matmul}</td>
<td>time\textunderscore{matmul}</td>
</tr>
<tr>
<td>fft</td>
<td>fft</td>
</tr>
<tr>
<td>time\textunderscore{fft}</td>
<td>time\textunderscore{fft}</td>
</tr>
</tbody>
</table>

CPU task queue: \[\rightarrow T_2 T_1\]
GPU task queue: \[\rightarrow T_3 T_2 T_1\]
Started systematizing knowledge per program across datasets and architectures
Optimization knowledge reuse across programs

Started systematizing knowledge per program across datasets and architectures

How to reuse knowledge among programs?

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

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

MILEPOST GCC with Interactive Compilation Interface:

- ft1 - Number of basic blocks in the method
- ... 
- ft19 - Number of direct calls in the method
- ft20 - Number of conditional branches in the method
- ft21 - Number of assignment instructions in the method
- ft22 - Number of binary integer operations in the method
- ft23 - Number of binary floating point operations in the method
- ft24 - Number of instructions in the method
- ...
- ft54 - Number of local variables that are pointers in the method
- ft55 - Number of static/extern variables that are pointers in the method

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

3) Given **new** program, dataset, architecture, **predict behavior** based on prior knowledge!
Nearest-neighbour classifier

Example: Euclidean distance based on static program features normalized by number of instructions
Grigori Fursin et al. MILEPOST GCC: machine learning enabled self-tuning compiler.
**Dynamic features**

**Principle Component Analysis:**

<table>
<thead>
<tr>
<th>Most informative Performance Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) L1_TCA</td>
</tr>
<tr>
<td>2) L1_DCH</td>
</tr>
<tr>
<td>3) TLB_DM</td>
</tr>
<tr>
<td>4) BR_INS</td>
</tr>
<tr>
<td>5) RES_STL</td>
</tr>
<tr>
<td>6) TOT_CYC</td>
</tr>
<tr>
<td>7) L2_ICH</td>
</tr>
<tr>
<td>8) VEC_INS</td>
</tr>
<tr>
<td>9) L2_DCH</td>
</tr>
<tr>
<td>10) L2_TCA</td>
</tr>
<tr>
<td>11) L1_DCA</td>
</tr>
<tr>
<td>12) HW_INT</td>
</tr>
<tr>
<td>13) L2_TCH</td>
</tr>
<tr>
<td>14) L1_TCH</td>
</tr>
<tr>
<td>15) BR_MS</td>
</tr>
</tbody>
</table>

**Analysis of the importance of the performance counters.**

The data contains one good optimization sequence per benchmark.

**Calculating mutual information between a subset of the performance counters and good optimization sequences**

• Analysis and detection of contentions in multi-core systems with shared cache

• Fast CPU/memory bound detection through breaking code semantics

• Software/hardware co-design (predicting better architecture designs)

• Performance/power balancing (through frequency variation)

• Decomposition of large applications into codelets for performance modeling
Grigori Fursin        “Systematizing tuning of computer systems using crowdsourcing and statistics”            HPSC 2013, NTU, Taiwan          March, 2013

• Used in MILEPOST project (2007-2009) by IBM, CAPS, University of Edinburgh, INRIA to build first public machine-learning based compiler
• Opened for public access in 2009 to continue collaborative R&D

Public Collective Tuning Portal (cTuning.org)

http://cTuning.org
community-based development and optimization of computing systems

http://cTuning.org/ctools
Collaborative R&D Tools with common APIs:
GCC ICI, CCC, UNIDAPT
Continuous (transparent) monitoring of computing systems

http://cTuning.org/cdatabase
Statistical analysis and machine learning to systematize and reuse optimization knowledge and predict good optimizations
Collective Optimization Database with optimization cases from the community

Optimize programs/architectures and enable adaptive self-tuning computing systems
(programs, run-time systems, compilers, architectures)

Collective Benchmarks (from the community)
cBench
Web services:
- collect static & dynamic optimization cases
- suggest good optimizations (based on program and architecture features, run-time behavior and optimization scenarios)

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!
Community is very interested in open “big data” for collaborative R&D

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

End-user task

Collaborative Infrastructure and repository for continuous online learning

Classification, predictive modeling

Optimal solutions

Systematization and unification of collective knowledge (big data)

“crowd”

Result

Extrapolate collective knowledge to build faster and more power efficient computer systems to continue innovation in science and technology!
Conclusions - much more to be done!

Now have public repository, tools, benchmarks, datasets and methodology that can help:

Academia (students and researchers):
• Instead of loosing time on developing tools for ever changing environments, focus on statistical, data mining and machine learning techniques to:
  • unify program optimization, design space exploration, run-time adaptation
  • detect important characteristics of computer systems
  • detect representative benchmarks and data sets
  • evaluate multiple machine learning algorithms to predict optimizations or hardware designs or dynamic multi-objective adaptation (SVM, decision trees, hierarchical modeling, etc)

Industry:
• restore confidence in academic research due to reproducibility of results
• use and share collaborative tools
• share statistics about behavior of computer systems and optimizations
• expose choices and characteristics to end-users through unified interfaces
Challenges for public repositories and collaborative tools:

- Data management
  - MySQL vs schema-free databases
  - central vs distributed repository
  - performance vs portability
  - extensibility
  - online learning and data compaction
  - easy sharing

- Portability of the framework across different architectures, OSes, tools

- Interfaces to “open up” tools, architectures, applications for external tuning
  - simplicity and portability

- Reproducibility of experiments

- New publication model
• Collective Mind: new plugin-based extensible infrastructure and schema-free repository for collaborative and holistic analysis and tuning of computer systems - will be released in May 2013 at HiPEAC computing week in Paris
• OpenME interface to “open up” compilers, run-time systems and applications for unified external tuning
• Hundreds of codelets, thousands of data sets, multiple packages prepared for various research scenarios on data mining
• Plugins for online auto-tuning and predictive modelling
• Portability across all major architectures and OS (Linux, Windows, Android)
• Collaboration with industry and academia

Google discussion groups
- ctuning-discussions
- collective-mind

Twitter
- c_tuning
- grigori_fursin
Acknowledgements

• PhD students and postdocs (my Intel Exascale team)

  *Abdul Wahid Memon, Pablo Oliveira, Yuriy Kashnikov*

• Colleague from NCAR, USA

  *Davide Del Vento and his colleagues/interns*

• Colleagues from IBM, CAPS, ARC (Synopsis), Intel, Google, ARM, ST

• Colleagues from Intel (USA)

  *David Kuck and David Wong*

• cTuning community:

  [Image of cTuning logo]

• EU FP6, FP7 program and HiPEAC network of excellence

  [Link to HiPEAC website: http://www.hipeac.net]
Main references

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

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


• Yang Chen, Shuangde Fang, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Olivier Temam and Chengyong Wu. **Deconstructing iterative optimization.** ACM Transactions on Architecture and Code Optimization (TACO), October 2012, Volume 9, Number 3

• Yang Chen, Yuanjie Huang, Lieven Eeckhout, Grigori Fursin, Liang Peng, Olivier Temam, Chengyong Wu. **Evaluating Iterative Optimization across 1000 Data Sets.** PLDI'10

• Victor Jimenez, Isaac Gelado, Lluis Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** HiPEAC’09
Main references

• Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. Finding representative sets of optimizations for adaptive multiversioning applications. SMART'09 co-located with HiPEAC'09

• Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization. HiPEAC’07


• Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. A Practical Method For Quickly Evaluating Program Optimizations. HiPEAC’05
