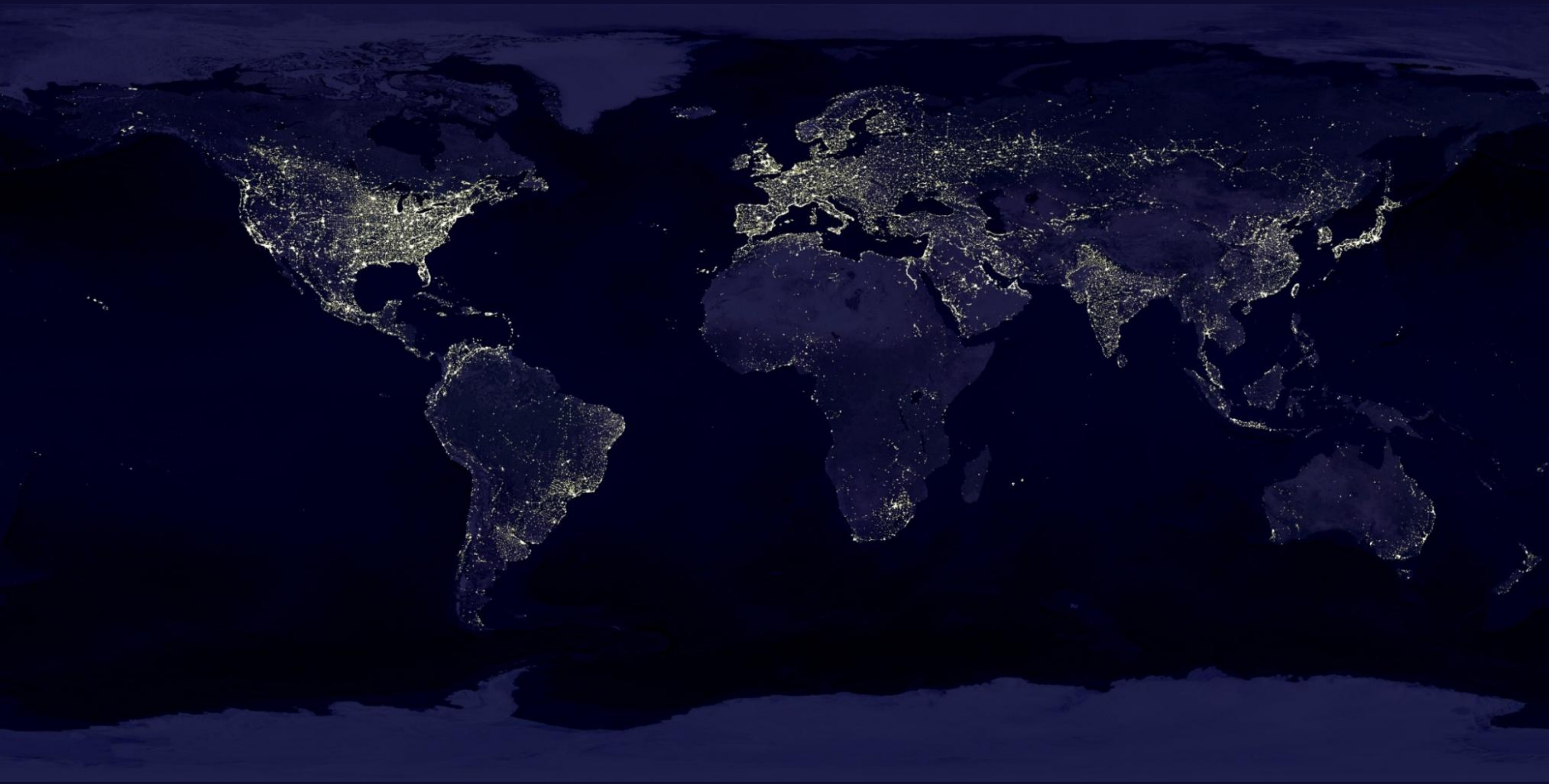
Panel on collaborative research methodology for large-scale computer systems



Grigori Fursin
INRIA, France

EXADAPT/ASPLOS
March 2012

Background

1993-1997	Semiconductor electronics, physics, neural networks First steps on auto-tuning and machine learning
1998-now	Auto-tuning Machine learning Data mining Run-time adaptation
1998-now	Common tools and repositories for collective tuning
2009-now	**cTuning.org** - public repository and infrastructure for collaborative application and architecture characterization and optimization
2012	**cTuning$_2$** – modular and extensible repository and infrastructure for collaborative R&D

Motivation

End-users demand:

- Increased computational resources
- Reduced costs

Resource providers need:

- Better products
- Faster time to market
- Increased Return on Investment (ROI)

Motivation

End-users demand:

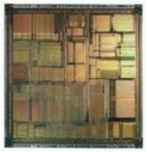
- Increased computational resources
- Reduced costs

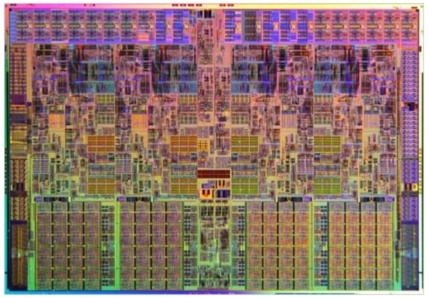
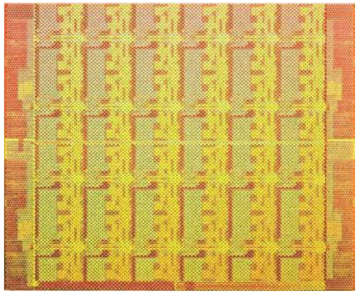
Resource providers need:

- Better products
- Faster time to market
- Increased Return on Investment (ROI)

Computer system designers produce:

Rapidly evolving HPC systems
that already reach petaflop and start targeting exaflop performance.



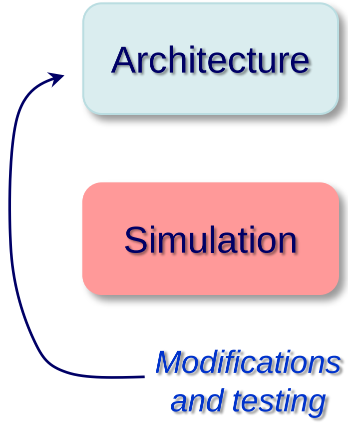



In the near future HPC systems may feature
millions of processors with hundreds of homo- and heterogeneous cores per processor.

Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

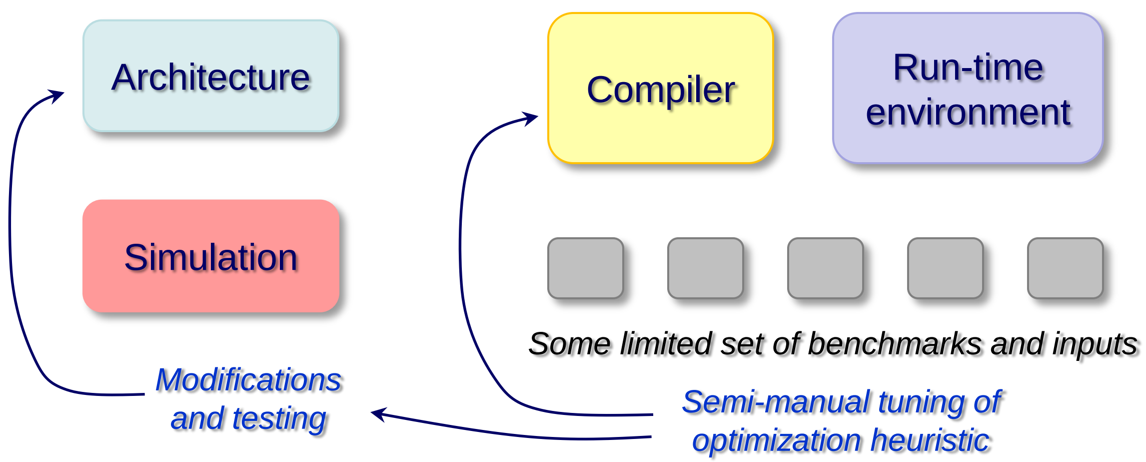


Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

2) Compiler is designed and tuned for a limited set of benchmarks / kernels.



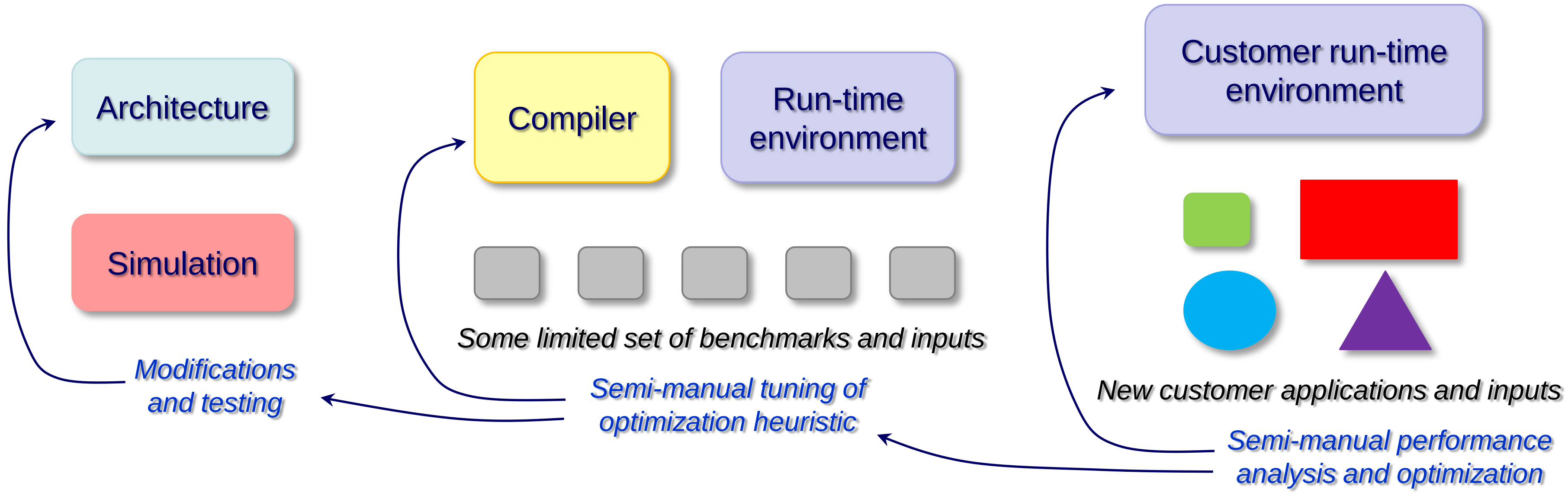
Motivation

While HPC systems (hardware and software) reach *unprecedented levels of complexity,* overall design and optimization methodology *hardly changed in decades:*

1) Architecture is designed, simulated and tested.

2) Compiler is designed and tuned for a limited set of benchmarks / kernels.

3) System is delivered to a customer. New applications are often underperforming and have to be manually analysed and optimized.



Motivation: auto-tuning

Potential solution during last 2 decades:
auto-tuning (iterative compilation)

Learn behavior of computer systems across executions while tuning various parameters

Optimization spaces:

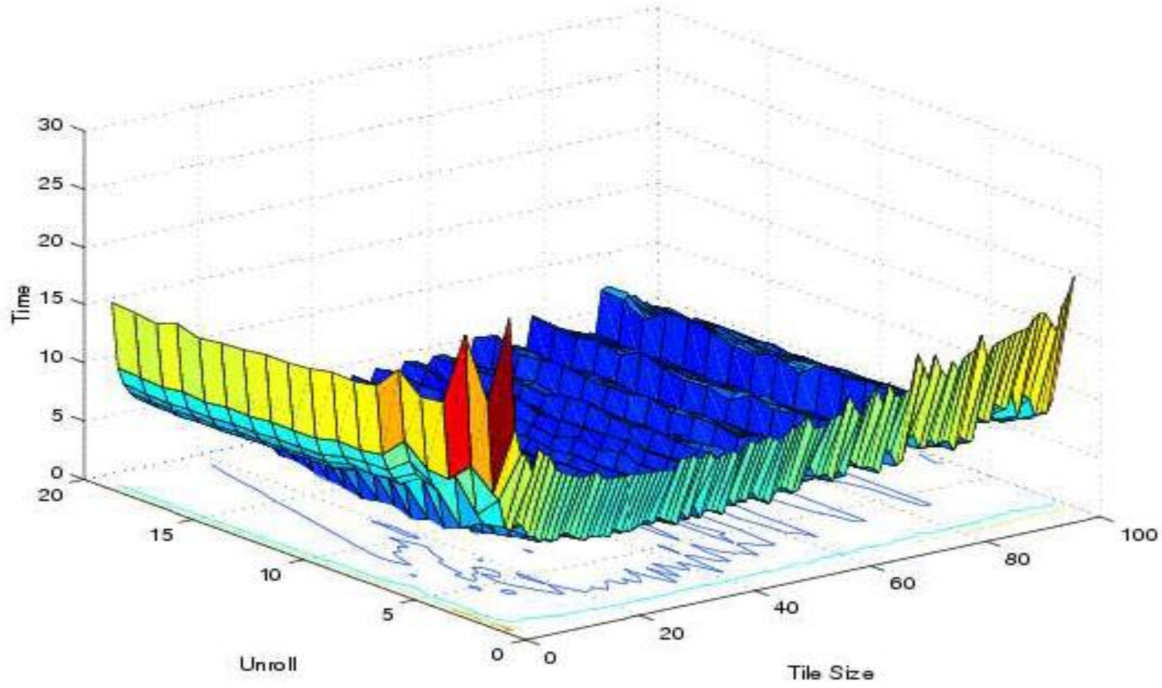
- combinations of compiler flags
- parametric transformations and their ordering
- cost-model tuning for individual transformations (meta optimization)
- parallelization (OpenMP vs MPI, number of threads)
- scheduling (heterogeneous systems, contention detection)
- architecture designs (cache size, frequency)

…

Motivation: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments. Why?

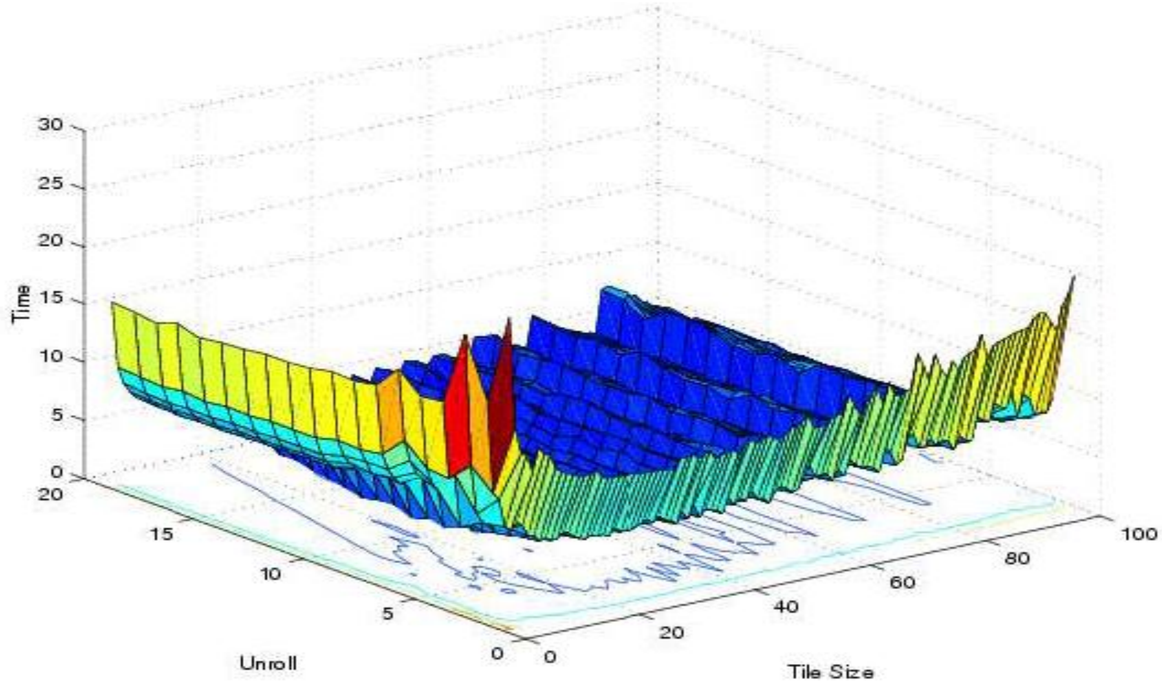
Matrix multiply kernel, 1 loop nest, 2 transformations, optimization space = 2000



Motivation: auto-tuning

Auto-tuning shows high potential for nearly 2 decades but still far from the mainstream in production environments. Why?

Matrix multiply kernel, 1 loop nest, 2 transformations, optimization space = 2000



Simple swim benchmark from SPEC2000, multiple loop nests, 3 transformations, optimization space = 10^{52}

Motivation: 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 part of the system is taken into account

(rarely reflect behavior of the whole system)

- Often the same (one) dataset is used
- Lack of run-time adaptation
- No optimization knowledge sharing and reuse

Motivation

Current state (acknowledged by most of the R&D roadmaps until 2020):

Developing, testing and optimizing computer systems is becoming:

- *non-systematic and highly non-trivial*
- *tedious, time consuming and error-prone*
- *inefficient and costly*

As a result:

- *slowing down innovation in science and technology*
- *enormous waste of expensive computing resources and energy*
- *considerable increase in time to market for new products*
- *low return on investment*

Motivation

Current state (acknowledged by most of the R&D roadmaps until 2020):

Developing, testing and optimizing computer systems is becoming:

- *non-systematic and highly non-trivial*
- *tedious, time consuming and error-prone*
- *inefficient and costly*

As a result:

- *slowing down innovation in science and technology*
- *enormous waste of expensive computing resources and energy*
- *considerable increase in time to market for new products*
- *low return on investment*

Current design and optimization methodology has to be dramatically revisited particularly if we want to achieve Exascale performance!

Fundamental challenges

Researchers and engineers tend to jump from one interesting technology to another and provide some quick ad-hoc solutions while fundamental problems are not solved in decades:

1) Rising complexity of computer systems:
too many tuning dimensions and choices

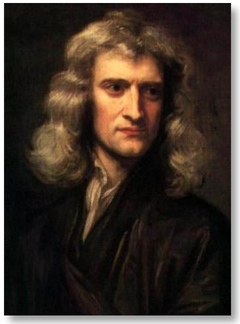
2) Performance is not anymore the only or main requirement for new computing systems: multiple objectives such as performance, power consumption, reliability, response time, etc. have to be carefully balanced :
user objectives vs choices
benefit vs optimization time

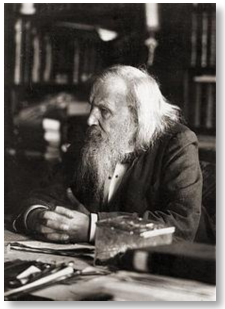
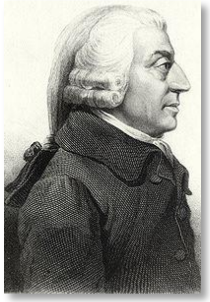
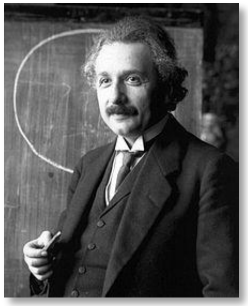
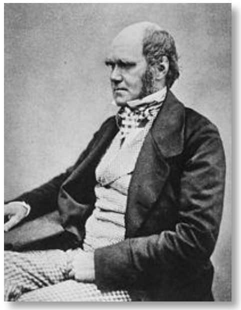
3) Complex relationship and interactions between ALL components at ALL levels.

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

Long-term interdisciplinary vision

Take the best of existing sciences that deal with complex systems:
physics, mathematics, chemistry, biology, computer science, etc

What can we learn?

Long-term vision

A physicist's view:

Develop interdisciplinary methodology and collaborative infrastructure to systematize, simplify and automate design, optimization and run-time adaptation of computer systems based on empirical, analytical and statistical techniques combined with learning, classification and predictive modeling

Why not to make collaborative, community-based framework and repository to start sharing data and modules just like in physics, biology, etc?

Software engineering in academic research

Why not to make collaborative, community-based framework and repository to start sharing data and modules just like in physics, biology, etc?

Academic research on program and architecture design and optimization rarely focuses on software engineering.

Often considered as a waste of time!

Main focus is often to publish as many papers as possible!

Reproducibility and statistical meaningfulness of results is often not even considered! In fact, it is often impossible!

cTuning: Collaborative tuning infrastructure and repository

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

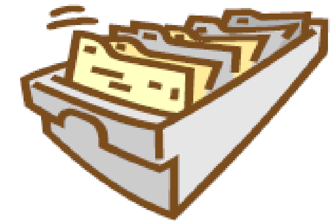
Continuous (transparent) monitoring of computing systems

Collective Benchmarks (from the community)

cBench

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)

Web services:

- collect static & dynamic optimization cases
- suggest good optimizations (based on program and architecture features, run-time behavior and optimization scenarios)

Released in 2009, used in MILEPOST project to enable machine learning self-tuning compiler

Collective Optimization Database

cTuning initiative (http://cTuning.org)

Public repository to share optimization cases:

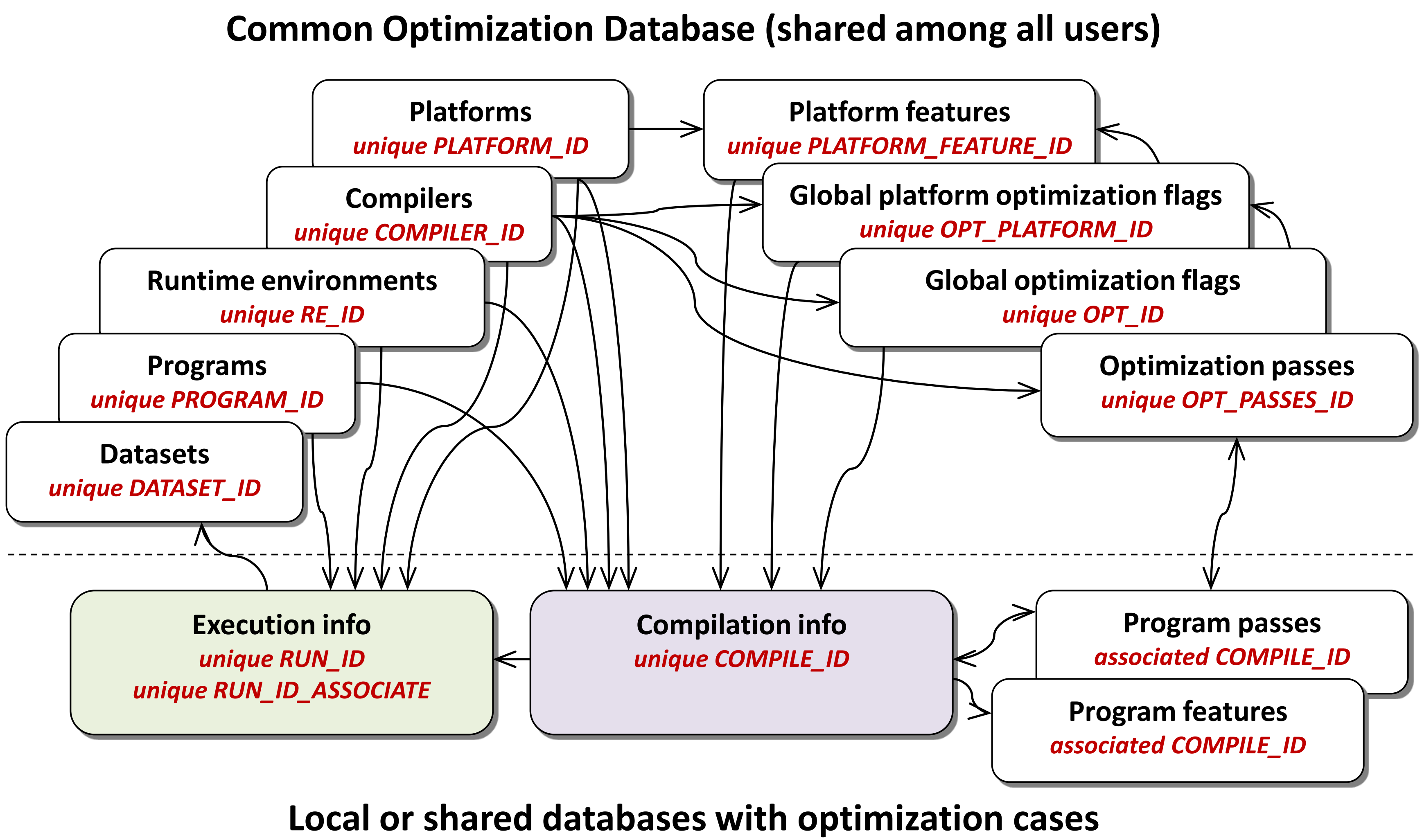
http://cTuning.org/cdatabase

- Cases include program optimizations and architecture configurations to improve execution time, code size, detect performance anomalies and bugs, etc.

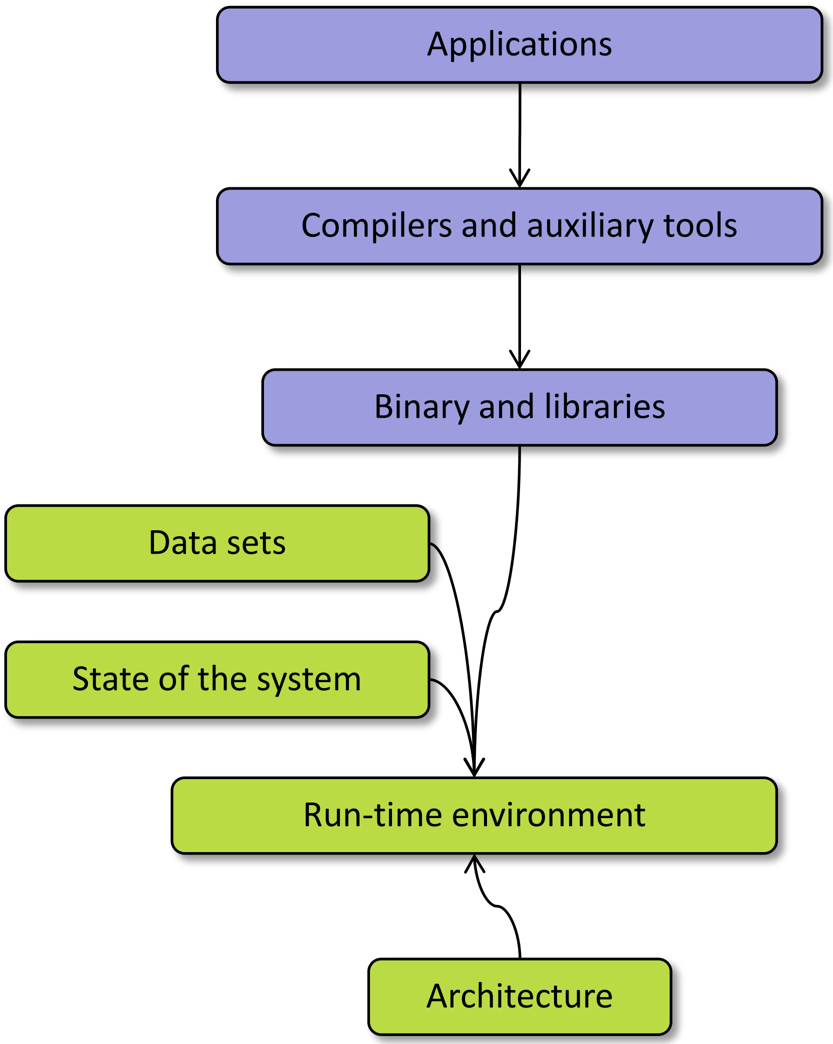
- All records have a unique UUID-based identifier to enable referencing of optimization cases and full decentralization of the infrastructure if needed.

- Optimization case consists of several compilations and executions with a baseline optimization (-O3) and some new selection of optimizations.

Collective Optimization Database



Recording information



- Provide wrappers (cTuning plugins) with standardized APIs around user tools and data to be able to record *information flow (particularly about compilation and execution)*

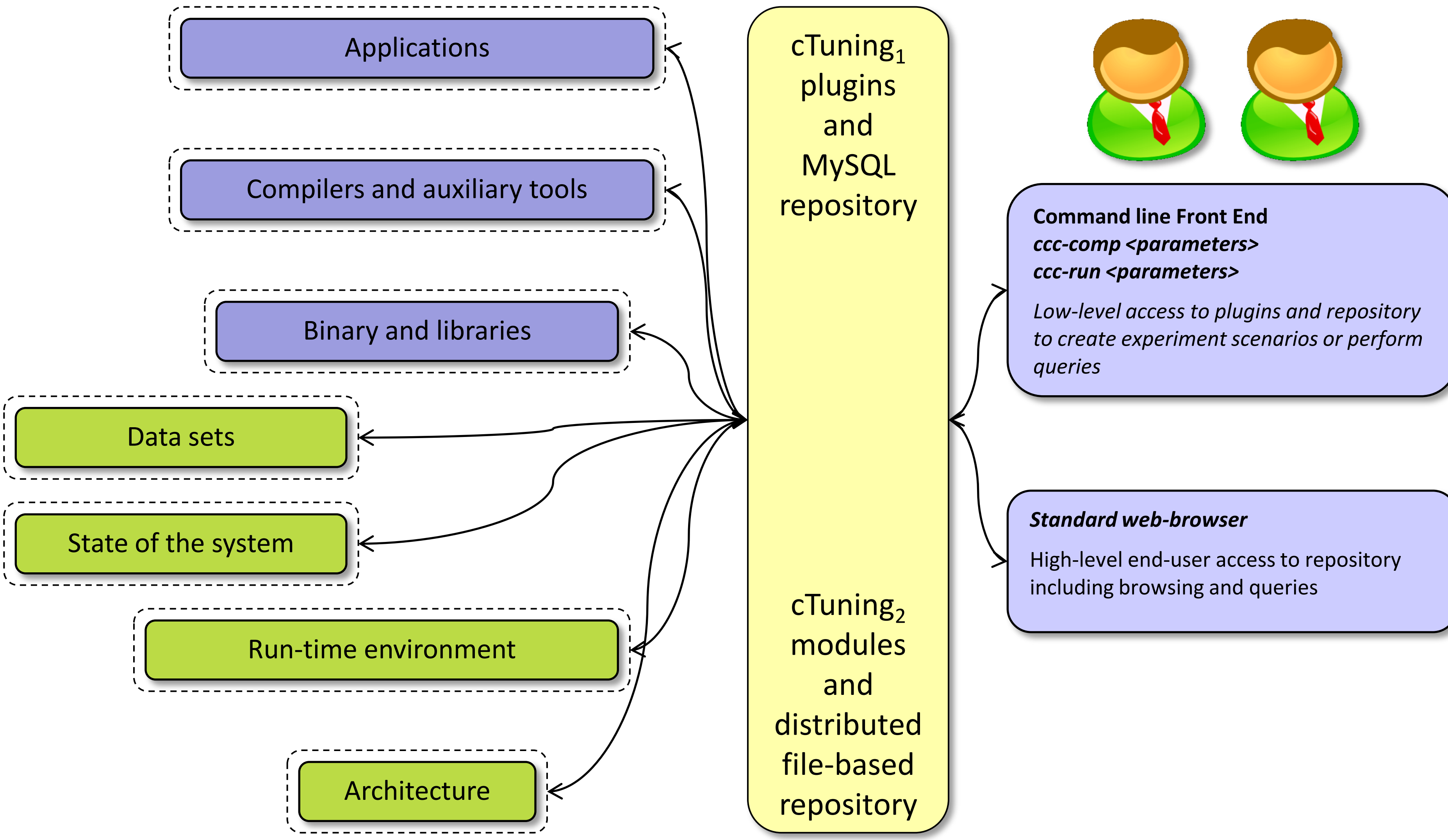
- Provide high-level plugins (php, java, python) and low-level plugins (C, C++, Fortran)

- Gradually expose tuning dimensions and characteristics instead of exposing everything at once to keep complexity under control!

- Add multiple collaborative benchmarks to the repository (kernels and real applications) and hundreds of datasets (cBench, MiDataSets)

Recording information

Connect all tools together through plugins with unified interfaces



Preparation for systematic exploration

Started collaborative exploration of optimization spaces (multiple dimensions):

- *Multiple datasets*
  - matrices of different sizes
- *Multiple compiler optimizations*
  - compiler flags
  - compiler pragmas
  - source to source transformations
- *Multiple run-time environment conditions*
  - sole execution
  - execution of multiple instances in parallel
- *Multiple architectures*
  - Intel, AMD, Longsoon, ARC, ARM with varied parameters:
  - frequency
  - cache size
- *Multiple objectives*
  - execution time, power consumption, CPI, code size, compilation time, etc

Empirical multi-objective auto-tuning

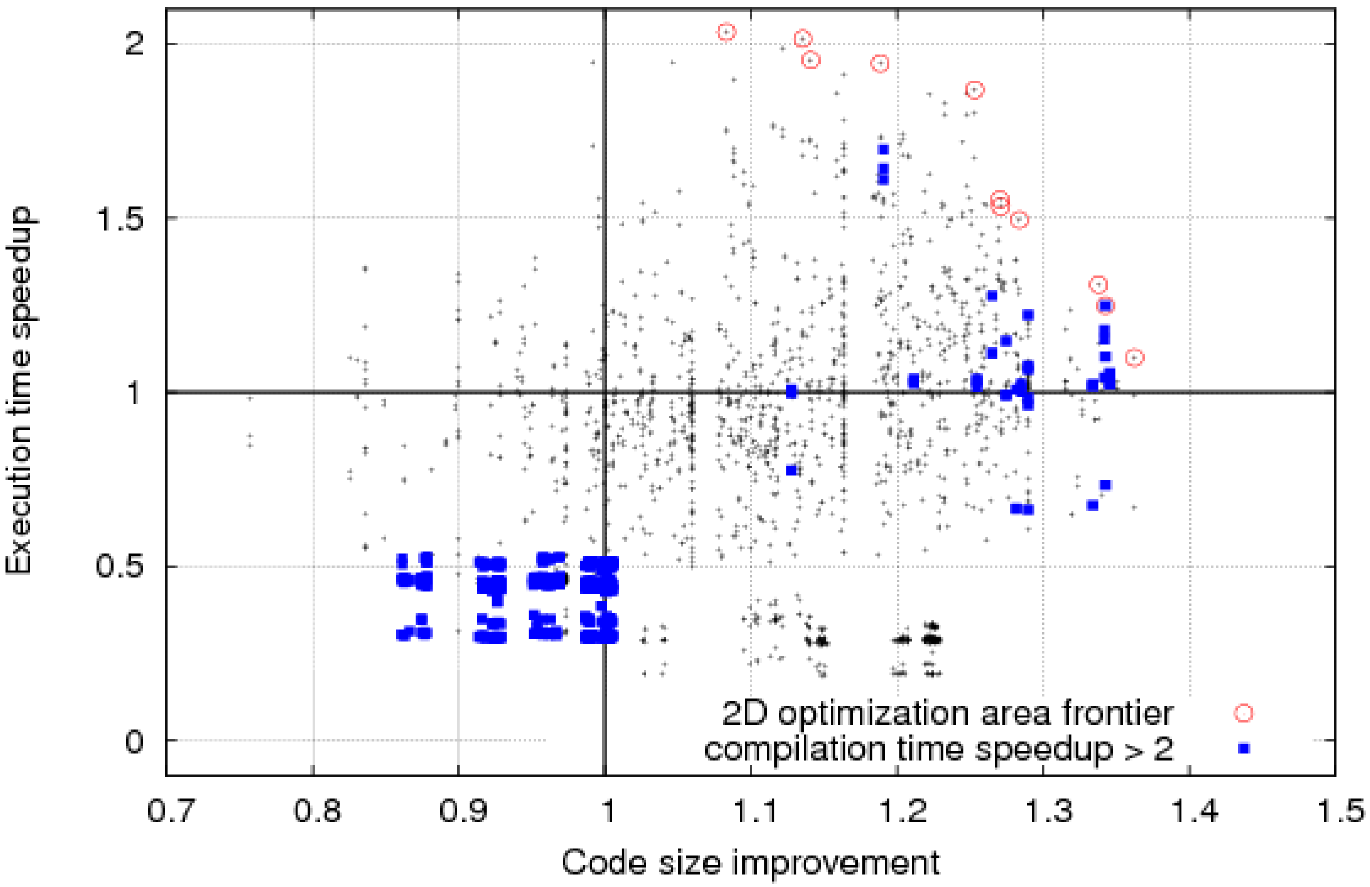
Multi-objective optimizations (depends on user scenarios):

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*

New additional requirement: *reduce power consumption*



susan corners kernel

Intel Core2

GCC 4.4.4
similar results on ICC 11.1

baseline opt=-O3
~100 optimizations

random combinations
(50% probability)

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

Machine learning and data mining

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

Code patterns:

```
for                    F
  for                  F
    for                F
               …
       load …          L
       mult …          A
       store …         S
               …
```

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!

Machine learning and data mining

Static/semantic features are often not enough to characterize dynamic behavior!

Use dynamic features (more characterizing dimensions)!

"Traditional" features:

performance counters (difficult to interpret, change from architecture to architecture though fine for learning per architecture).

Reactions to code changes:

perform changes and observe program reactions (change in execution time, power, etc).

Apply optimizations (compiler flags, pragmas, manual code/data partitioning, etc).

Change/break semantics (remove or add individual instructions(data accesses, arithmetic, etc) or threads, etc and observe reactions to such changes).

Sharing and reproducing experiments and modules

Share

Explore

Model

Discover

Reproduce

Extend

Have fun!

Grigori Fursin et al. **MILEPOST GCC: machine learning enabled self-tuning compiler**. International Journal of Parallel Programming (IJPP) , June 2011, Volume 39, Issue 3, pages 296-327

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

What have we learnt from cTuning$_1$

• 15 years ago - lots of disbelief

• Now we have a complete reference framework and repository to validate and extend research ideas on auto-tuning, run-time adaptation and machine learning (cTuning/MILEPOST GCC)

• Community can reproduce and share results

• Community can focus more on research using collective data sets

Technical issues:

• Global repository not scalable

• MySQL is slow and not extensible

• No easy way to share modules, benchmarks, data sets

• Programming modules in C/PHP was not so simple for end-users

What have we learnt from cTuning$_1$

It's fun working with the community!

My favorite comment 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…

What have we learnt from cTuning$_1$

It's fun working with the community!

My favorite comment about MILEPOST GCC from Slashdot.org:

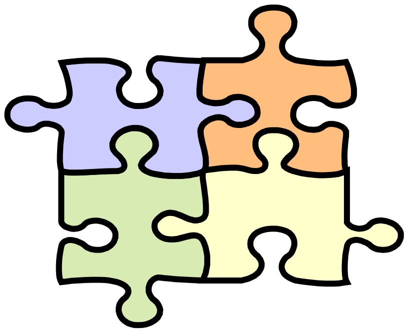
http://mobile.slashdot.org/story/08/07/02/1539252/using-ai-with-gcc-to-speed-up-mobile-design

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 can help you validate and speed up research!

cTuning$_2$ aka Collective Mind



Methodology for collaborative design and optimization of computer systems is ready!

- Build extensible infrastructure and distributed repository to record information flow inside computer systems and share data and modules from multiple users (applications, data sets, tools, optimization cases, algorithms, etc)
- Enable continuous observation of the behavior of the whole (!) system
- Enable continuous exploration of multiple design and optimization dimensions
- Explain, characterize and classify unusual/unexpected behavior (discover knowledge through data mining)
- Perform hierarchical analysis starting from very simple cases while gradually increasing complexity (decompose large applications into more understandable pieces and quickly perform first coarse-grain analysis/tuning while moving to finer-grain effects only when/if needed)

cTuning$_2$ aka Collective Mind

- Automatically and continuously classify and correlate program/architecture behaviour with “features”, optimizations and multiple objective functions using predictive modelling
- Build an expert system that queries repository and models to :
  - *quickly identify program and architecture behavior anomalies*
  - *suggest better optimizations for a given program*
  - *suggest better architecture designs*
  - *suggest run-time adaptation scenarios*

    (program optimizations and hardware reconfigurations as reaction to program and system behavior)

Join collaborative effort

- Release of the new framework as LGPL before summer 2012
- Collaborate with researchers and end-users to add various modules

to characterize and optimize existing computer systems:

  - compiler optimizations
  - parallelization (OpenMP/MPI)
  - run-time scheduling and adaptation (CPU/GPU, avoid contentions)

- Evaluate various machine learning techniques and data mining techniques

  for classification and predictive modeling

  - detect important characteristics of computer systems
  - evaluate various ML techniques (SVM, decision trees, hierarchical modeling)

- Continuously and rigorously rank solutions using statistical analysis

Join collaborative effort

cTuning$_1$: http://cTuning.org
http://groups.google.com/group/ctuning-discussions

cTuning$_2$: http://code.google.com/p/collective-mind
http://twitter.com/cresearch

Topic
"Collective characterization, optimization and design of computer systems"
has been as one of the thematic sessions of the upcoming
EU HiPEAC3 network of excellence

A few 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

- Grigori Fursin, Yuriy Kashnikov, Abdul Wahid Memon, Zbigniew Chamski, Olivier Temam, Mircea Namolaru, Elad Yom-Tov, Bilha Mendelson, Ayal Zaks, Eric Courtois, Francois Bodin, Phil Barnard, Elton Ashton, Edwin Bonilla, John Thomson, Chris Williams, Michael O'Boyle. **MILEPOST GCC: machine learning enabled self-tuning compiler.** International Journal of Parallel Programming (IJPP), June 2011, Volume 39, Issue 3, pages 296-327

- Victor Jimenez, Isaac Gelado, Lluis Vilanova, Marisa Gil, Grigori Fursin and Nacho Navarro. **Predictive runtime code scheduling for heterogeneous architectures.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2009), Paphos, Cyprus, January 2009

- Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long and Grigori Fursin. **Finding representative sets of optimizations for adaptive multiversioning applications.** 3rd International Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'09) co-located with HiPEAC'09, Paphos, Cyprus, January 2009

A few references

•Grigori Fursin, John Cavazos, Michael O'Boyle and Olivier Temam. **MiDataSets: Creating The Conditions For A More Realistic Evaluation of Iterative Optimization.** Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2007), Ghent, Belgium, January 2007

•F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M.F.P. O'Boyle, J. Thomson, M. Toussaint and C.K.I. Williams. **Using Machine Learning to Focus Iterative Optimization.** Proceedings of the 4th Annual International Symposium on Code Generation and Optimization (CGO), New York, NY, USA, March 2006

•Grigori Fursin, Albert Cohen, Michael O'Boyle and Oliver Temam. **A Practical Method For Quickly Evaluating Program Optimizations.** Proceedings of the 1st International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005), number 3793 in LNCS, pages 29-46, Barcelona, Spain, November 2005

•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

• Grigori Fursin. **Iterative Compilation and Performance Prediction for Numerical Applications.** Ph.D. thesis, University of Edinburgh, Edinburgh, UK, January 2004

PDFs available at http://fursin.net/dissemination

Questions?

Contact: grigori.fursin@inria.fr
grigori.fursin@exascalable.com

cTuning_1: http://cTuning.org
http://groups.google.com/group/ctuning-discussions

cTuning_2: http://code.google.com/p/collective-mind
http://twitter.com/cresearch