

Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing

Abdul Wahid Memon, Grigori Fursin

▶ **To cite this version:**

Abdul Wahid Memon, Grigori Fursin. Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing. PARCO mini-symposium on "Application Autotuning for HPC (Architectures)", Sep 2013, Munich, Germany. hal-00944513

HAL Id: hal-00944513
https://inria.hal.science/hal-00944513

Submitted on 10 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Crowdtuning: systematizing auto-tuning using predictive modeling and crowdsourcing

Abdul MEMON [a], and Grigori FURSIN [b,1]
[a] *UVSQ, France*
[b] *INRIA Saclay, France*

Abstract.

Software and hardware co-design and optimization of HPC systems has become intolerably complex, ad-hoc, time consuming and error prone due to enormous number of available design and optimization choices, complex interactions between all software and hardware components, and multiple strict requirements placed on performance, power consumption, size, reliability and cost.

We present our novel long-term holistic and practical solution to this problem based on customizable, plugin-based, schema-free, heterogeneous, open-source Collective Mind repository and infrastructure with unified web interfaces and on-line advise system. This collaborative framework distributes analysis and multi-objective off-line and on-line auto-tuning of computer systems among many participants while utilizing any available smart phone, tablet, laptop, cluster or data center, and continuously observing, classifying and modeling their realistic behavior. Any unexpected behavior is analyzed using shared data mining and predictive modeling plugins or exposed to the community at cTuning.org for collaborative explanation, top-down complexity reduction, incremental problem decomposition and detection of correlating program, architecture or run-time properties (features). Gradually increasing optimization knowledge helps to continuously improve optimization heuristics of any compiler, predict optimizations for new programs or suggest efficient run-time (online) tuning and adaptation strategies depending on end-user requirements. We decided to share all our past research artifacts including hundreds of codelets, numerical applications, data sets, models, universal experimental analysis and auto-tuning pipelines, self-tuning machine learning based meta compiler, and unified statistical analysis and machine learning plugins in a public repository to initiate systematic, reproducible and collaborative R&D with a new publication model where experiments and techniques are validated, ranked and improved by the community.

Keywords. Collective Mind, crowdtuning, crowdsourcing auto-tuning and co-design, software and hardware co-design and co-optimization, compiler-agnostic tuning, on-line tuning and learning, systematic behavior modeling, predictive modeling, data mining, machine learning, on-line advice system, metadata, top-down optimization, incremental problem decomposition, decremental (differential) analysis, complexity reduction, tuning dimension reduction, customizable plugin-based infrastructure, public repository of knowledge, big data processing and compaction, agile research and development, cTuning.org, c-mind.org, systematic and reproducible research and experimentation, validation by community

[1] Corresponding author's e-mail:grigori.fursin@inria.fr or grigori.fursin@cTuning.org

Introduction, major challenges, and related work

Continuing innovation in science and technology is vital for our society and requires ever increasing computational resources. However, delivering such resources particularly with exascale performance for HPC or ultra low power for embedded systems is becoming intolerably complex, costly and error prone due to limitations of available technology, enormous number of available design and optimization choices, complex interactions between all software and hardware components, and growing number of incompatible tools and techniques with ad-hoc, intuition based heuristics. As a result, understanding and modeling of the overall relationship between end-user algorithms, applications, compiler optimizations, hardware designs, data sets and run-time behavior, essential for providing better solutions and computational resources, became simply infeasible as confirmed by numerous recent long-term international research visions about future computer systems [1,2,3,4,5,6]. On the other hand, the research and development methodology for computer systems has hardly changed in the past decades: computer architecture is first designed and later compiler is being tuned and adapted to the new architecture using some ad-hoc benchmarks and heuristics. As a result, peak performance of the new systems is often achieved only for a few previously optimized and not necessarily representative benchmarks such as SPEC for desktops and servers or LINPACK for TOP500 supercomputer ranking, while leaving most of the systems severely underperforming and wasting expensive resources and power.

Automatic off-line and on-line performance tuning techniques were introduced nearly two decades ago and later combined with machine learning in an attempt to solve some of the above problems. These approaches treat computer systems as a black box while exploring and learning their optimization parameter space empirically, i.e. compiling and executing a user program multiple times with varying optimizations or designs (compiler flags and passes, fine-grain transformations, frequency adaptation, cache reconfiguration, parallelization, etc) to empirically find better solutions that improve execution and compilation time, code size, power consumption and other characteristics [7,8,9,10,11,12,13,14,15,16,17]. Such techniques require little or no knowledge of the current platform and can adapt programs to any given architecture automatically. With time, auto-tuning has been accelerated with various adaptive exploration techniques including genetic algorithms, hill-climbing and probabilistic focused search. However, the main disadvantage of these techniques is an excessively long exploration time of large optimization spaces, with the lack of deep analysis about machine learning algorithms, their selection and scalability for ever growing training sets, optimization choices and available features which are often problem dependent, are the major research challenges in the field of machine learning for several decades, and far from being solved.

We believe that many of the above challenges and pitfalls are caused by the lack of a common experimental methodology, lack of interdisciplinary background, and lack of unified mechanisms for knowledge building and exchange apart from numerous similar publications where reproducibility and statistical meaningfulness of results as well as sharing of data and tools is often not even considered in contrast with other sciences including physics, biology and artificial intelligence. In fact, it is often impossible due to a lack of common and unified repositories, tools and data sets. At the same time, there is a vicious circle since initiatives to develop common tools and repositories to unify, systematize, share knowledge (data sets, tools, benchmarks, statistics, models) and make it widely available to the research and teaching community are practically not funded or rewarded academically where a number of publications often matter more than the repro-

ducibility and statistical quality of the research results. As a consequence, students, scientists and engineers are forced to resort to some intuitive, non-systematic, non-rigorous and error-prone techniques combined with unnecessary repetition of multiple experiments using ad-hoc tools, benchmarks and data sets. Furthermore, we witness slowed down innovation, dramatic increase in development costs and time-to-market for the new embedded and HPC systems, enormous waste of expensive computing resources and energy, and diminishing attractiveness of computer engineering often seen as "hacking" rather than systematic science.

1. Collective Mind approach

1.1. Back to basics

We would like to start with the formalization of the eventual needs of end-users and system developers or providers. End-users generally need to perform some tasks (playing games on a console, watching videos on mobile or tablet, surfing Web, modeling a new critical vaccine on a supercomputer or predicting a new crash of financial markets using cloud services) either as fast as possible or with some real-time constraints while minimizing or amortizing all associated costs including power consumption, soft and hard errors, and device or service price. Therefore, end-users or adaptive software require a function that can suggest most optimal design or optimization choices **c** based on properties of their tasks and data sets **p**, set of requirements **r**, as well as current state of a used computing system **s**:

$$\mathbf{c} = F(\mathbf{p}, \mathbf{r}, \mathbf{s})$$

This function is associated with another one representing behavior of a user task running on a given system depending on properties and choices:

$$\mathbf{b} = B(\mathbf{p}, \mathbf{c}, \mathbf{s})$$

This function is of particular importance for hardware and software designers that need to continuously provide and improve choices (solutions) for a broad range of user tasks, data sets and requirements while trying to improve own ROI and reduce time to market. In order to find optimal choices, it should be minimized in presence of possible end-user requirements (constraints). However, the fundamental problem is that nowadays this function is highly non-linear with such a multi-dimensional discrete and continuous parameter space which is not anymore possible to model analytically or evaluate empirically using exhaustive search [7,18]. For example, **b** is a behavior vector that can now include execution time, power consumption, compilation time, code size, device cost, and any other important characteristic; **p** is a vector of properties of a task and a system that can include semantic program features [19,20,21,22], dataset properties, hardware counters [23,24], system configuration, and run-time environment parameters among many others; **c** represents available design and optimization choices including algorithm selection, compiler and its optimizations, number of threads, scheduling, processor ISA, cache sizes, memory and interconnect bandwidth, frequency, etc; and finally **s** represents the state of the system during parallel execution of other programs, system or core frequency, cache contentions and so on.

1.2. Interdisciplinary collaborative methodology

Current multiple research projects mainly show that it is possible to use some off-the-shelf on-line or off-line adaptive exploration (sampling) algorithms combined with some existing models to approximate above function and predict behavior, design and optimization choices for 70-90% cases but in a very limited experimental setup. In contrast, our ambitious long-term goal is to understand how to continuously build, enhance, systematize and optimize hybrid models that can *explain and predict all possible behaviors and choices* while selecting minimal set of representative properties, benchmarks and data sets for predictive modeling [25]. We reuse our interdisciplinary knowledge in physics, quantum electronics and machine learning to build a new methodology that can effectively deal with rising complexity of computer systems through gradual and continuous top-down problem decomposition, analysis and learning. We also develop a modular infrastructure and repository that allows to easily interconnect various available tools and techniques to distribute adaptive probabilistic exploration, analysis and optimization of computer systems among many users [26,27] while exposing unexpected or unexplained behavior to the community with interdisciplinary backgrounds particularly in machine learning and data mining through unified web interfaces for collaborative solving and systematization.

1.3. Collective Mind infrastructure and repository

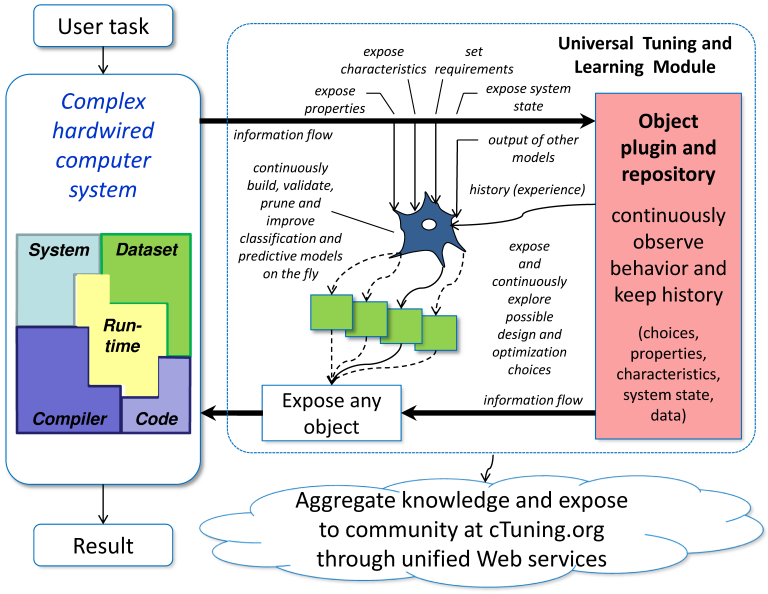


Figure 1. *Gradual decomposition, parameterization, observation, tuning and learning of complex hardwired computer systems.*

Collective Mind framework and repository (cM for short) enables continuous, collaborative and agile top-down decomposition of the whole complex hardwired computer systems into unified and connected subcomponents (modules) with gradually exposed

various characteristics, tuning choices (optimizations), properties and system state as conceptually shown in Figure 1. At a coarse-grain level, modules serve as wrappers around existing command line tools such as compilers, source-to-source transformers, code launchers, profilers, among many others. Such modules are written in python for productivity and portability reasons, and can be launched from command line in a unified way using Collective Mind front-end *cm* as following:

cm ⟨ *module name or UID* ⟩ ⟨ *command* ⟩ ⟨ *unified meta information* ⟩ – ⟨ *original cmd* ⟩

These modules enable transparent monitoring of information flow, exposure of various characteristics and properties in a unified way (meta information), and exploration or prediction of design and optimization choices, while helping researchers to abstract their experimental setups from constant changes in the system. Internally, modules can call each other using just one unified *cM access function* which uses a schema-free easily extensible nested dictionary that can be directly serialized to JSON as both input and output as following:

```
r=cm_kernel.access({'cm_run_module_uoa':<module name or UID>,
                    'cm_action':<command>,
                    parameters})
```

where command in each module is directly associated with some function. Since JSON can also be easily transmitted through Web using standard http post mechanisms, we implemented a simple cM web server that can be used for P2P communication or centralized repository during crowdsourcing and possibly multi-agent based on-line learning and tuning.

Each module has an associated storage that can preserve any collections of files (whole benchmark, data set, tool, trace, model, etc) and their meta-description in a JSON file. Thus each module can also be used for any data abstraction and includes various common commands standard to any repository such as *load, save, list, search, etc*. We use our own simple directory-based format as following:

```
.cmr/<Module name or UID>/<Data entry UID>
```

where .cmr is an acronym for Collective Mind Repository. In contrast with using SQL-based database in the first cTuning version that was fast but very complex for data sharing or extensions of structure and relations, a new open format allows users to be database and technology-independent with the possibility to modify, add, delete or share entries and whole repositories using standard OS functions and tools like SVN, GIT or Mercury, or easily convert them to any other format or database when necessary. Furthermore, cM can transparently use open source JSON-based indexing tools such as ElasticSearch [28] to enable fast and powerful queries over schema-free meta information. Now, any research artifact will not be lost and can now be referenced and directly found using the so called cID (Collective ID) of the format: ⟨ *module name or UID* ⟩*:*⟨ *data entry or UID* ⟩.

Such infrastructure allows researchers and engineers to connect existing or new modules into experimental pipelines like "research LEGO" with exposed characteristics, properties, constraints and states to quickly and collaboratively prototype and crowdsource their ideas or production scenarios such as traditional adaptive exploration of large experimental spaces, multi-objective program and architecture optimization or continuous on-line learning and run-time adaptation while easily utilizing all available benchmarks, data sets, tools and models provided by the community. Additionally, single and

unified access function enables transparent reproducibility and validation of any experiment by preserving input and output dictionaries for a given experimental pipeline module. Furthermore, we decided to keep all modules inside repository thus substituting various ad-hoc scripts and tools. With an additional cM possibility to install various packages and their dependencies automatically (compilers, libraries, profilers, etc) from the repository or keep all produced binaries in the repository, researchers now have an opportunity to preserve and share the whole experimental setup in a private or public repository possibly with a publication.

We started collaborative and gradual decomposition of large, coarse-grain components into simpler sub-modules including decomposition of programs into kernels or codelets [29] to keep complexity under control and possibly use multi-agent based or brain inspired modeling and adaptation of the behavior of the whole computer system locally or during P2P crowdsourcing. Such decomposition also allows community to first learn and optimize coarse-grain behavior, and later add more fine-grain effects depending on user requirements, time constraints and expected return on investment (ROI) similar to existing analysis methodologies in physics, electronics or finances.

1.4. Data and parameter description and classification

In traditional software engineering, all software components and their API are usually defined at the beginning of the project to avoid modifications later. However, in our case, due to ever evolving tools, APIs and data formats, we decided to use agile methodology together with type-free inputs and outputs for all functions focusing on quick and simple prototyping of research ideas. Only when modules and their inputs and outputs become mature or validated, then (meta)data and interface are defined, systematized and classified. However, they can still be extended and reclassified at any time later.

For example, any key in an input or output dictionary of a given function and a given module can be described as ”choice”, ”(statistical) characteristic”, ”property” and ”state”, besides a few internal types including ”module UID” or ”data UID” or ”class UID” to provide direct or semantic class-based connections between data and modules. Parameters can be discrete or continuous with a given range to enable automatic exploration. Thus, we can easily describe compiler optimizations; dataset properties such as image or matrix size, architecture properties such as cache size or frequency, represent execution time, power consumption, code size, hardware counters; categorize benchmarks and codelets in terms of reaction to optimizations or as CPU or memory bound, and so on.

1.5. OpenME interface for fine-grain analysis, tuning and adaptation

Most of current compilers, applications and run-time systems are not prepared for easy and straightforward fine-grain analysis and tuning due to associated software engineering complexity, sometimes proprietary internals, possible compile or run-time overheads, and still occasional disbeliefs in effective run-time adaptation. Some extremes included either fixing, hardwiring and hiding all optimization heuristics from end-users or oppositely exposing all possible optimizations, scheduling parameters, hardware counters, etc. Some other available mechanisms to control fine-grain compiler optimization through pragmas can also be very misleading since it is not always easy or possible to validate whether optimization was actually performed or not.

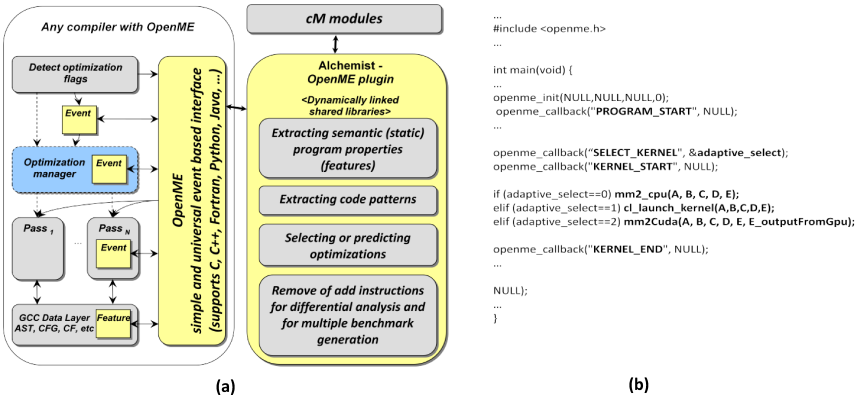


Figure 2. *Event and plugin-based OpenME interface to "open up" rigid tools (a) and applications (b) for external fine-grain analysis, tuning and adaptation, and connect them to cM*

Instead of developing yet another source-to-source tools or binary translators and analyzers, we developed a simple event-based plugin framework called Interactive Compilation Interface (ICI) to "open up" previously hardwired tools for external analysis and tuning. ICI was written in plain C originally for Open64 and later for GCC, requires minimal instrumentation of a compiler and helps to expose or modify only a subset of program properties or compiler optimization decisions through external dynamic plugins based on researcher needs and usage scenario. This interface can easily evolve with the compiler itself, has been successfully used in the MILEPOST project to build machine-learning self-tuning compiler [22], and is now available in mainline GCC. Based on this experience, we developed a new version of this interface (OpenME) [26] that is used to "open up" any available tool such as GCC, LLVM, Open64, architecture simulator, etc. in a unified way as shown in Figure 2(a), or any application for example to train predictive scheduler on heterogeneous many-core architectures [24] as shown in Figure 2(b).

2. Possible usage scenarios

2.1. Adaptive exploration (sampling) and dimensionality reduction

Figure 3 shows exploration of dataset parameters for LU-decomposition numerical kernel on GRID5000 machines with Intel Core2 and SandyBridge processors. Since all characteristics are usually dependent, we can apply cM plugin (module) to detect universal Paretto fronter on the fly in multi-dimensional space (currently not optimal) during on-line exploration and filter all other cases. A user can choose to explore any other available characteristic in a similar way such as power consumption, compilation time, etc depending on usage scenario and requirements. In order to speed up random exploration further, we use probabilistic focused search similar to ANOVA and PCA described in [30,17] that can suggest most important tuning/analysis dimensions with likely highest speedup or unusual behavior, and guide further finer-grain exploration in those areas. Collective exploration is critical to build and update a realistic training set for machine-learning based self-tuning meta-compiler cTuning-CC to automatically and continuously improve default optimization heuristic of GCC, LLVM, ICC, Open64 or any other compiler connected to cM [22,31].

2.2. On-line learning

Crowdtuning has a side effect - generation and processing of huge amount of data that is well-known in other fields as a ”big data” problem. However, in our past work on online tuning, we showed that it is possible not only to learn behavior and find correlations between characteristics, properties and choices to build models of behavior on the fly at each client or program, but also to effectively compact experimental data keeping only representative or unexpected points, and minimize communications between cM nodes thus making cM a giant, distributed learning network to some extent similar to brain [32,25,17].

Figure 3 demonstrates how on-line learning is performed in our framework using LU-decomposition benchmark as an example, CPI characteristic, and 2 Intel-based platforms (Intel Core2 Centrino T7500 Merom 2.2GHz L1=32KB 8-way set-associative, L2=4MB 16-way set associative - red dots vs. Intel Core i5 2540M 2.6GHz Sandy Bridge L1=32KB 8-way set associative, L2=256KB 8-way set associative, L3=3MB 12-way set associative - blue dots). At the beginning, our system does not have any knowledge about behavior of this (or any other) benchmark, so it simply observes and stores available characteristics while collecting as many properties of the whole system as possible (exposed by a researcher or user). At each step, system processes all historical observations using various available predictive models such as SVM or MARS in order to find correlations between properties and characteristics. In our example, after sufficient amount of observations, system can build a model that automatically correlated data set size N, cache size and CPI (in our case combination of linear models B-F that reflect memory hierarchy of a particular system) with nearly 100% prediction. However, system always continue observing behavior to continuously validate it against existing model in order to detect discrepancies (failed predictions). In our case, the system eventually detects outliers A that are due to cache alignment problems. Since off-the-shelf models rarely handle such cases, our framework allows to exclude such cases from modeling and expose them to the community through the unified Web services to reproduce and explain this behavior, find relevant features and improve or optimize existing models. In our case, we managed to fit a hybrid rule-based model that first validates cases where data set size is a power of 2, otherwise it uses linear models as functions of a data set and cache size.

Systematic characterization (modeling) of a program behavior across many systems and data sets allows researchers or end-users to focus further optimizations (tuning) only on representative areas with distinct behavior while collaboratively building an on-line advice system. In the above example, we evaluated and prepared the following advices for optimizations: points A can be optimized using array padding; area B can profit from parallelization and traditional compiler optimizations targeting ILP; areas C-E can benefit from loop tiling; area F and points A can benefit from reduced processor frequency to reduce power consumption using cM online adaptation plugin. Since auto-tuning is continuously performed, we will release final optimization details at cM live repository [27] during symposium.

Gradually increasing and systematized knowledge in the repository in form of models can now be used to detect and characterize an abnormal program or system behavior, suggest future architectural improvements, or predict most profitable program optimizations, run-time adaptation scenarios and architecture configurations depending on user requirements. For example, this knowledge can be effecitvely used for staged tuning (split compilation) to build static multi-versioning applications with cM plugins for phase-based adaptation [32] or predictive scheduling [24] in heterogeneous systems that

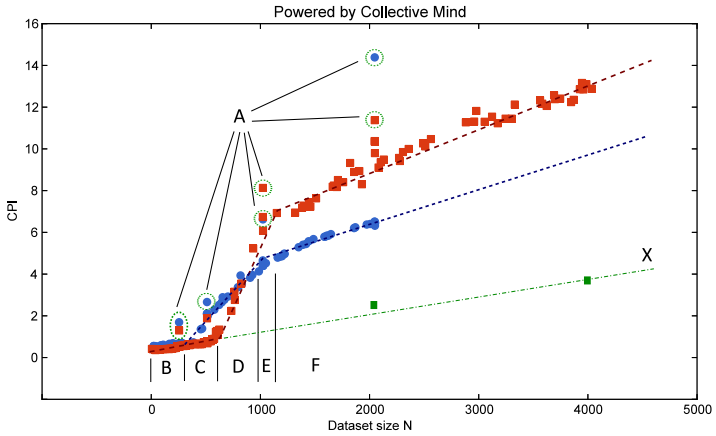


Figure 3. *On-line learning (predictive modeling) of a CPI behavior of ludcmp on 2 different platforms (Intel Core2 vs Intel i5) vs matrix size N and cache size*

can automatically adjust their behavior at run-time to varying data sets, environments, architectures and system state by selecting appropriate versions or changing frequency to maximize performance and minimize power consumption, while avoiding complex recompilation frameworks.

2.3. Benchmark automatic generation and decremental analysis

Projects on applying machine learning to auto-tuning suffer from yet another well-known problem: lack of benchmarks. Our experience with hundreds of codelets and thousands of data sets [33,34,22] shows that they are still not enough to cover all possible properties and behavior of computer systems. Generating numerous synthetic benchmarks and data sets is theoretically possible but will result in additional explosion in analysis and tuning dimensions. Instead, we use existing benchmarks, codelets and even data sets as templates, and utilize Alchemist plugin [26] for GCC to randomly or incrementally modify them by removing, modifying or adding various instructions, basic blocks, loops, and thus generating. Naturally, we ignore crashing variants of the code and continue evolving only the working ones. Importantly, we use this approach not only to extend realistic training sets, but also to gradually identify various behavior anomalies and detect code properties to explain these anomalies and improve predicting modeling without any need for slow and possibly imprecise system/architecture simulator or numerous and sometimes misleading hardware counters as originally presented in [16,18]. For example, we can scalarize memory accesses to characterize code and data set as CPU or memory bound [18] (line X in Figure 3 shows ideal codelet behavior when all floating point memory accesses are NOPed). Additionally, we use Alchemist plugin to extract code structure, patterns and other properties to improve our cTuning CC machine-learning based meta compiler connected to GCC, LLVM, Open64, Intel and Microsoft compilers, and to guide SW/HW co-design.

3. Conclusions and future work

We strongly believe that the only way forward now to address fundamental challenges in computer engineering is to start collaborative systematization and unification of available knowledge about design and optimization of computer systems. We use our interdisciplinary background and experience to develop the first to our knowledge integrated, extensible and collaborative infrastructure and repository (Collective Mind) that can represent, preserve and connect directly or semantically all research artifacts including data, executable code and interfaces in a unified way. We hope that our collaborative, evolutionary and agile methodology, and extensible plugin-based Lego-like framework can bring together interdisciplinary communities to continuously validate, systematize and improve collective knowledge about designing and optimizing whole computer systems, and extrapolate it to build faster, more power efficient, reliable and adaptive devices and software. We hope that community will continue developing more plugins (modules) to plug various third-party tools including TAU [35], Periscope [36], Scalasca [37], Intel vTune and many others to cM, or continue gradual decomposition of programs into codelets, and complex tools into simpler connected self-tuning modules while crowdsourcing learning, tuning and classifying of their behavior.

We started building a large live public repository of realistic behavior of multiple programs in realistic environments with realistic data sets ("big data") *c-mind.org/repo*. It is currently being populated with our past research artifacts including hundreds of codelets and benchmarks, thousands of data sets, universal compilation and execution pipeline with adaptive exploration (tuning), dimension reduction and statistical analysis modules, and classical off-the-shelf or hybrid predictive models. This repository and infrastructure should allow the community to quickly reproduce and validate existing results, and focus their effort on developing novel tuning techniques combined with data mining, classification and predictive modeling rather than wasting time on building individual experimental setups. It can also be used to address the challenge of collaboratively finding minimal representative set of benchmarks, codelets and datasets covering behavior of most of existing computer systems, detecting correlations in a collected data together with combinations of relevant properties (features), pruning irrelevant ones, systematizing and compacting existing experimental data, removing or exposing noisy or wrong experimental results. It can also be effectively used to validate, extend and compact existing models such as roofline [38], or evaluate and improve adaptation techniques including multi-agent based using cM P2P communication, classify programs by similarities in models, by reactions to optimizations [17] and to semantically non-equivalent changes [16], or collaboratively develop and optimize new complex hybrid predictive models that from our past practical experience can not yet be fully automated thus using data mining and machine learning as a powerful helper rather than black-box panacea at least at this stage.

We hope that our approach will help to shift current focus from publishing only good experimental results or speedups, to sharing all research artifacts, validating past techniques, and exposing unexplained behavior or encountered problems to the interdisciplinary community for reproducibility and collaborative solving and ranking. We also hope that Collective Mind framework will be of help to a broad range of researchers even outside of computer engineering not to drawn in their experimentation while processing, systematizing, and sharing their scientific data, code and models. Finally, we hope that Collective Mind methodology will help to restore the attractiveness of computer engineering making it a more systematic and rigorous discipline [39].

Acknowledgments

Grigori Fursin was funded by EU HiPEAC postdoctoral fellowship (2005-2006), EU MILEPOST project (2007-2010) and French Intel/CEA Exascale Lab (2010-2011). Abdul Memon was funded by INRIA and Higher Education Commission of Pakistan. We are thankful to Francois Bodin and CAPS Entreprise for sharing codelets from the MILEPOST project. We are very grateful to cTuning and HiPEAC communities for many interesting discussions and feedback during development of Collective Mind repository and infrastructure presented in this paper. Finally, we would like to thank GRID5000 project and community for providing an access to powerful computational resources.

References

[1] Krste Asanovic et.al. The landscape of parallel computing research: a view from Berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, December 2006.

[2] The HiPEAC vision on high-performance and embedded architecture and compilation (2012-2020). http://www.hipeac.net/roadmap, 2012.

[3] Jack Dongarra et.al. The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, February 2011.

[4] Tony Hey, Stewart Tansley, and Kristin M. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[5] Ubiquitous high performance computing (uhpc). Technical Report DARPA-BAA-10-37, USA, 2010.

[6] PRACE: partnership for advanced computing in europe. http://www.prace-project.eu.

[7] R. Whaley and J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the Conference on High Performance Networking and Computing*, 1998.

[8] F. Matteo and S. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1381–1384, Seattle, WA, May 1998.

[9] F. Bodin, T. Kisuki, P.M.W. Knijnenburg, M.F.P. O'Boyle, and E. Rohou. Iterative compilation in a non-linear optimisation space. In *Proceedings of the Workshop on Profile and Feedback Directed Compilation*, 1998.

[10] K.D. Cooper, P.J. Schielke, and D. Subramanian. Optimizing for reduced code space using genetic algorithms. In *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 1–9, 1999.

[11] T. Kisuki, P.M.W. Knijnenburg, and M.F.P. O'Boyle. Combined selection of tile sizes and unroll factors using iterative compilation. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 237–246, 2000.

[12] S. Triantafyllis, M. Vachharajani, N. Vachharajani, and D.I. August. Compiler optimization-space exploration. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 204–215, 2003.

[13] B. Singer and M. Veloso. Learning to predict performance from formula modeling and training data. In *Proceedings of the Conference on Machine Learning*, 2000.

[14] Z. Pan and R. Eigenmann. Rating compiler optimizations for automatic performance tuning. In *Proceedings of the International Conference on Supercomputing*, 2004.

[15] Michele Tartara and Stefano Crespi-Reghizzi. Continuous learning of compiler heuristics. *TACO*, 9(4):46, 2013.

[16] Grigori Fursin, Michael F. P. O'Boyle, Olivier Temam, and Gregory Watts. A fast and accurate method for determining a lower bound on execution time: Research articles. *Concurrency: Practice and Experience*, 16(2-3):271–292, January 2004.

[17] Grigori Fursin and Olivier Temam. Collective optimization: A practical collaborative approach. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(4):20:1–20:29, December 2010.

[18] Grigori Fursin. *Iterative Compilation and Performance Prediction for Numerical Applications*. PhD thesis, University of Edinburgh, United Kingdom, 2004.

[19] A. Monsifrot, F. Bodin, and R. Quiniou. A machine learning approach to automatic production of compiler heuristics. In *Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, Applications*, LNCS 2443, pages 41–50, 2002.

[20] Mark Stephenson, Saman Amarasinghe, Martin Martin, and Una-May O'Reilly. Meta optimization: Improving compiler heuristics with machine learning. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03)*, pages 77–90, June 2003.

[21] 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. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 2006.

[22] 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, Christopher Williams, and Michael F. P. OBoyle. Milepost gcc: Machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39:296–327, 2011. 10.1007/s10766-010-0161-2.

[23] John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael O'Boyle, and Olivier Temam. Rapidly selecting good compiler optimizations using performance counters. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, March 2007.

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

[25] Lianjie Luo, Yang Chen, Chengyong Wu, Shun Long, and Grigori Fursin. Finding representative sets of optimizations for adaptive multiversioning applications. In *3rd Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation (SMART'09), colocated with HiPEAC'09 conference*, January 2009.

[26] Collective Mind: open-source plugin-based infrastructure and repository for systematic and collaborative research, experimentation and management of large scientific data. http://cTuning.org/tools/cm.

[27] Collective Mind Live Repo: public repository of knowledge about design and optimization of computer systems. http://c-mind.org/repo.

[28] ElasticSearch: open source distributed real time search and analytics. http://www.elasticsearch.org.

[29] Stéphane Zuckerman, Joshua Suetterlein, Rob Knauerhase, and Guang R. Gao. Using a "codelet" program execution model for exascale machines: position paper. In *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, EXADAPT '11, pages 64–69, New York, NY, USA, 2011. ACM.

[30] B. Franke, M.F.P. O'Boyle, J. Thomson, and G. Fursin. Probabilistic source-level optimisation of embedded programs. In *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2005.

[31] MILEPOST project archive (MachIne Learning for Embedded PrOgramS opTimization). `http://cTuning.org/project-milepost`.

[32] Grigori Fursin, Albert Cohen, Michael O'Boyle, and Oliver Temam. A practical method for quickly evaluating program optimizations. In *Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005)*, pages 29–46, November 2005.

[33] Grigori Fursin, John Cavazos, Michael O'Boyle, and Olivier Temam. MiDataSets: Creating the conditions for a more realistic evaluation of iterative optimization. In *Proceedings of the International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2007)*, January 2007.

[34] Yang Chen, Lieven Eeckhout, Grigori Fursin, Liang Peng, Olivier Temam, and Chengyong Wu. Evaluating iterative optimization across 1000 data sets. In *In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI*, 2010.

[35] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, May 2006.

[36] Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. Periscope: An online-based distributed performance analysis tool. pages 1–16, 2010.

[37] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurr. Comput. : Pract. Exper.*, 22(6):702–719, April 2010.

[38] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

[39] HiPEAC thematic session at ACM FCRC'13: Making computer engineering a science. http://www.hipeac.net/thematic-session/making-computer-engineering-science, 2013.