Cristina Silvano, Gianluca Palermo, Giovanni Agosta, Amir H. Ashouri, Davide Gadioli, Stefano Cherubin, Emanuele Vitali, Luca Benini, Andrea Bartolini, Daniele Cesarini, João Cardoso, João Bispo, Pedro Pinto, Riccardo Nobre, Erven Rohou, Loïc Besnard, Imane Lasri, Nico Sanna, Carlo Cavazzoni, Radim Cmar, Jan Martinovič, Kateřina Slaninová, Martin Golasowski, Andrea R. Beccari, Candida Manelfi

Autotuning and Adaptivity in Energy Efficient HPC Systems:
The ANTAREX Toolbox (Invited Paper)

CF '18: Computing Frontiers Conference
May 8–10, 2018, Ischia, Italy.

The published version is available online at: https://doi.org/**10.1145/3203217.3205338**

Autotuning and Adaptivity in Energy Efficient HPC Systems: The ANTAREX Toolbox (Invited Paper)

Cristina Silvano[1], Gianluca Palermo[1], Giovanni Agosta[1], Amir H. Ashouri[1], Davide Gadioli[1], Stefano Cherubin[1], Emanuele Vitali[1], Luca Benini[2], Andrea Bartolini[2], Daniele Cesarini[2], João Cardoso[3], João Bispo[3], Pedro Pinto[3], Riccardo Nobre[3], Erven Rohou[4], Loïc Besnard[5], Imane Lasri[4], Nico Sanna[6], Carlo Cavazzoni[6], Radim Cmar[7], Jan Martinovič[8], Kateřina Slaninová[8], Martin Golasowski[8], Andrea R. Beccari[9], Candida Manelfi[9]

[1]Politecnico di Milano – DEIB, Italy [2]Eidgenoessische Technische Hochschule Zuerich – IIS, Switzerland [3]Universidade do Porto – FEUP, Portugal [4]INRIA, France [5]CNRS, France [6]CINECA – SCAI, Italy [7]SYGIC, Slovakia [8]IT4Innovations, VŠB - Technical University of Ostrava, Czech Republic [9]Dompé Farmaceutici SpA, L'Aquila, Via Campo di Pile, 67100, Italy.

ABSTRACT

Designing and optimizing applications for energy-efficient High Performance Computing systems up to the Exascale era is an extremely challenging problem. This paper presents the toolbox developed in the ANTAREX European project for autotuning and adaptivity in energy efficient HPC systems. In particular, the modules of the ANTAREX toolbox are described as well as some preliminary results of the application to two target use cases. [1]

KEYWORDS

High-Performance & Power-aware Computing, Autotuning, DSL

ACM Reference Format:

Cristina Silvano[1], Gianluca Palermo[1], Giovanni Agosta[1], Amir H. Ashouri[1], Davide Gadioli[1], Stefano Cherubin[1], Emanuele Vitali[1], Luca Benini[2], Andrea Bartolini[2], Daniele Cesarini[2], João Cardoso[3], João Bispo[3], Pedro Pinto[3], Riccardo Nobre[3], Erven Rohou[4], Loïc Besnard[5], Imane Lasri[4], Nico Sanna[6], Carlo Cavazzoni[6], Radim Cmar[7], Jan Martinovič[8], Kateřina Slaninová[8], Martin Golasowski[8], Andrea R. Beccari[9], Candida Manelfi[9]. 2018. Autotuning and Adaptivity in Energy Efficient HPC Systems: The ANTAREX Toolbox (Invited Paper). In *CF '18: CF '18: Computing Frontiers Conference, May 8–10, 2018, Ischia, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3203217.3205338

1 INTRODUCTION

Designing and implementing HPC applications are difficult and complex tasks, which require mastering several specialized languages and tools for performance tuning. This is incompatible with the current trend to open HPC infrastructures to a wider range of users. The current model where the HPC center staff directly supports the development of applications will become unsustainable in the long term. Thus, the availability of effective standard programming languages and APIs is crucial to provide migration paths towards novel heterogeneous HPC platforms as well as to guarantee the ability of developers to work effectively on these platforms. Moreover, to fulfil the target of energy-efficient supercomputers there is the need to combine the current trend of HPC systems with radically new software stacks to exploit the benefits offered by heterogeneity at several levels (supercomputer, job, node).

ANTAREX project addresses these challenging problems through a holistic approach spanning all the decision layers composing the supercomputer software stack and exploiting effectively the full system capabilities (including heterogeneity and energy management) [13, 14]. The main goal of the ANTAREX project is to express by a DSL the application self-adaptivity and to runtime manage and autotune applications for green heterogeneous HPC systems up to Exascale. One key innovation of the proposed approach consists of introducing a separation of concerns (where self-adaptivity and energy efficient strategies are specified aside to application functionalities) promoted by the definition of a DSL inspired by aspect-oriented programming concepts for heterogeneous systems. The new DSL will be introduced for expressing at compile time the adaptivity/energy/performance strategies and to enforce at runtime application autotuning and power management. The goal is to support the parallelism, scalability and adaptivity of a dynamic workload by exploiting the full system capabilities (including energy management) for emerging large-scale and extreme-scale systems. Moreover, ANTAREX enables the performance/energy control capabilities by introducing software knobs (including application parameters, code transformations and code variants), and designs scalable and hierarchical optimal control-loops capable of dynamically leveraging them together with performance/energy control knobs at different time scale to always operate the supercomputer and each application at the most efficient operating point. The key ANTAREX software innovations are designed and engineered since the beginning to be scaled-up to the Exascale level.

The ANTAREX project is driven by two use cases coming from highly social relevant HPC application scenarios: (i) a biopharmaceutical HPC application for drug discovery and (ii) a self-adaptive navigation system for smart cities.

This paper presents an overview of the toolbox developed in the context of the ANTAREX project. In particular, Section 2 presents

[1]This work has been funded by the EU grant 671623 FET-HPC-ANTAREX.

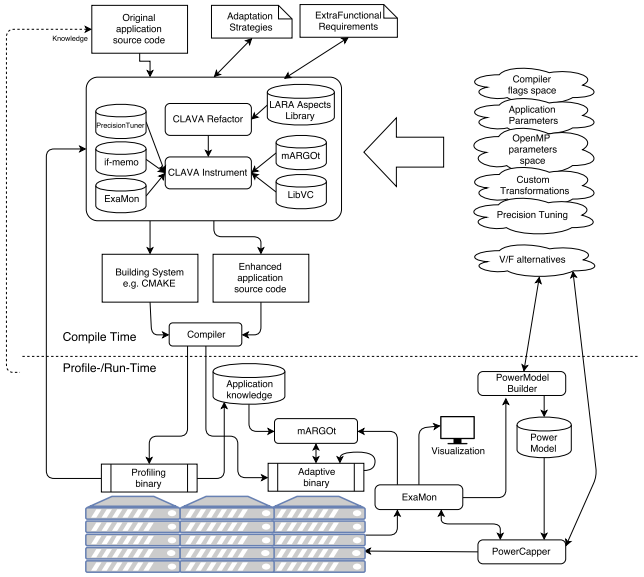

Figure 1: Overview of the ANTAREX flow

an overview of the ANTAREX framework, while Sections 3 and 4 describe respectively the modules to be used at compile-time and at run-time to optimize the target application. Finally, Section 5 discuss some application of the developed toolbox on the target application use cases.

2 ANTAREX TOOLBOX OVERVIEW

The ANTAREX approach and related tool flow (shown in Figure 1) operates both at compile-time and runtime. The application functionality is expressed through C/C++ code (possibly including legacy code), whereas the extra-functional aspects of the application, including requirements and adaptivity strategies are expressed by the DSL developed in the project (namely LARA). This guarantees a separation of concerns between functional and extra-functional aspects, thus facilitating the code reuse without weighing down the application developer with optimization details. The target optimization space of the ANTAREX toolbox is represented on the right side of Figure 1 with cloud symbols: compiler flags and custom transformations, application software knobs including variable approximation, OpenMP parameters, custom precision alternatives, and Voltge/Frequency levels. To enhance the code thus enabling the possibility of those target optimizations, the Clava source-to-source compiler (see Section 3.1) driven by LARA aspects has been used. This guarantees to the user the possibility to adopt several tools with their own interface by using a single front-end language. The outcome of this phase is not only an enhanced version of the code including tunable knobs and observable points, but also a complete building system for putting all together the several components. The application autotuning is postponed at the runtime phase, where software knobs (application parameters, code transformations and code variants) are configured according to the runtime information coming from the execution environment. The dynamic autotuning phase is based on profile-time information collected to build the application knowledge. Finally, an infrastructure-level monitoring framework is used at run-time to collect information about resource usage and application behavior. The same monitoring framework is also used to build automatically an infrastructure-level power-model used by the power manager to control and optimize the power usage.

3 COMPILE TIME FRAMEWORK

This section provides an overview of the main components developed in the ANTAREX compile-time framework (see Figure 1).

3.1 Lara DSL and Clava Compiler

One of the main challenges of ANTAREX is to develop a DSL able to integrate several technologies developed during the project [15]. Given that the use cases and the technologies are implemented in C/C++, we took a two-pronged approach regarding the DSL: 1) create a DSL that could transform and adapt C/C++ source-code, and 2) develop DSL API libraries, one for each technology.

Leveraging some previous work done around the LARA framework, and using the Clang compiler as a parser, we developed the tool Clava during ANTAREX. Clava is a C/C++ source-to-source compiler based on the LARA framework [9] that executes scripts written in the LARA language, a JavaScript-compatible language created to describe source-code analyses and transformations. The LARA language adds a few keywords to JavaScript such as `select`, to select points in the given source code (`function`, `loop`), or `apply`, to perform analyses and transformations over these points. However, the LARA language (and related framework) are language-independent, and they are decoupled from the target language that will be analyzed and transformed. They depend on a specific implementation for each target language, and Clava provides that implementation for C/C++.

For each module in Figure 1 (e.g., mARGOt, LibVC) we developed an API in LARA[2] that can be used to integrate that specific module in a target source-code. The API for each module has two purposes: 1) to provide a coherent interface for using each ANTAREX technology and 2) to avoid the users from learning the details of each technology for using it. Currently, it is possible, from a single LARA script, to transform automatically a "clean" C/C++ application to use any of the technologies developed by ANTAREX project. To better leverage the efforts spent in Clava and improve its usability, it has been developed a dedicated IDE including also a documentation generator, CMake integration, and a testing framework.

3.2 Dynamic Compilation

Dynamic compilation, or the ability to (re-)compile at runtime part of the application code, is potentially useful in HPC applications when profiling the application for code specialization opportunities would take too long, or when data characteristics may change over time, making a static optimization less effective. Dynamic compilation is a well-known technique in the general purpose computing world, but it is difficult to apply in a controlled way as required in HPC. To allow application developers to control dynamic compilation of multiple versions of a given code fragment, we introduce libVersioningCompiler[3] (abbreviated libVC), an open-source C++ library. libVC can dynamically compile any C/C++

[2] http://specs.fe.up.pt/tools/clava/doc/
[3] https://github.com/skeru/libVersioningCompiler

code without the burden of a full dynamic compiler, and can be used to support split compilation, continuous optimization and code specialization based on the input data and on workload changes [12]. libVC has been integrated in the ANTAREX tool flow through several LARA aspects covering the most common usage of the dynamic compilation framework.

3.3 Memoization

Memoization is a well known technique to improve program execution time by exploiting the redundancy in program execution [23] [22]. The main idea is to save in a table the results of execution of a section of program so that future execution of the same section with same input set can benefit from the saved results. In memoization schemes, two conditions are important: (1) memoized code should not cause any side-effect; (2) memoization should always produce the same result for the same inputs. At function-level, by applying these two conditions, memoizable functions are deterministic functions that do not modify the global state of the program. In ANTAREX, we implement the works done in [22] with some extensions. Considering a memoizable C function `foo`, the memoization consists of 1) the insertion of a wrapper function `foo_wrapper` and an associated table, 2) the substitution of the references to `foo` by `foo_wrapper` in the application. `foo_wrapper` calls the `foo` orignal function when the result of a call is not in the table.

In the ANTAREX project, this technique has been extented for C++ memoizable methods that require to take into account the mangling technique and the references to the objects. Moreover, several parameters have been added to improve several memoization approaches that can be addressed by using runtime autotuning or statically: management of the memoization table (updating policies, size of the table, approximation on the input arguments, pre-load table), dynamical "stop/run" of the memoization. Following the philosophy of the project, the memoization strategies are provided to the user by means of a set LARA aspects. An automatic detection of the potential functions/methods to be memoized is also provided to facilitate the integration of the proposed technique.

3.4 Compiler Autotuning

A major challenge in automatically choosing the right set of compiler optimizations is the fact that these code optimizations are programming language, application, and architecture dependent. Additionally, the word optimization is a not appropriate given that there is no guarantee that the transformed code will perform better than the original one. In fact, aggressive optimizations can also cause degradation of the performance of the code. Understanding the behavior of the optimizations, the perceived effects on the source-code, and the interaction of the optimizations with each other are complex modeling problems. This understanding is particularly difficult because compiler developers must consider hundreds of different optimizations that can be applied during the various compilation phases.

In ANTAREX, we faced the problem by developing a compiler autotuning framework based on machine learning techniques [2, 4]. In particular, we addressed the problem for GCC [3, 16], limiting the problem to compiler flag selection, and for LLVM [1, 21], where ordering and replication of optimizations enter into the game. In particular, the developed framework is based on an application characterization in terms of features and by off-line learned models to predict the optimal configurations for the specific case.

4 RUNTIME FRAMEWORK

This section provides an overview of the main ANTAREX components of the runtime framework (see Figure 1). Even if these tools work at runtime, we can find them even at compile time in 1 because they are integrated in the code by using the DSL and source-to-source compiler.

4.1 Exascale Monitoring Framework

Today processing elements can monitor performance efficiency by inspecting the activity of microarchitectural components as well as physical parameters (such as power consumption and temperature). This is possible throughout HW performance counters which in x86 systems can be read by privileged users, thus creating practical problems for user-space libraries to access them. Besides sensors which can be read directly from the software running on the core, supercomputing machines embed sensors external to computing elements, but relevant to the overall energy-efficiency. These sensors are used to monitor the node and rack cooling components as well as environmental parameters such as room and ambient temperature.

In ANTAREX, we developed EXAMON[4] [8] (standing for EXAscale MONitoring) for accessing the performance and power monitoring sensors distributely. EXAMON decouples the sensors' reading from the sensor value usage. Indeed, EXAMON uses a scalable approach where each sensor has associated a sensing agent which periodically collects the metrics and send the measured values with a synchronized time-stamp to an external data broker. The data broker organizes the incoming data in communication channels with an associated topic, while maintaining a list of subscribers to these topics to which broadcast each new incoming message on a specific topic. The subscriber registers a callback function to a given topic which gets called each time a new message is received. EXAMON uses a state-of-the-art big-data technologies to let the other layers of the ANTAREX toolchain to take advantage of the collected data. To combine EXAMON monitoring services with application knowledge, we developed a fine-grain profiling library capable of collecting in real-time architectural information during MPI phases for each MPI process independently, thus enabling application-aware power management solutions.

4.2 Dynamic Autotuning: mARGOt

Main goal of mARGOt is to enhance an application with an adaptive layer to tune software knobs to satisfy application requirements at runtime. In ANTAREX, we consider an application as a parametric function that elaborates input data to produce an output (i.e. $o = f(i, k_1, \ldots, k_n)$), with associated extra-functional requirements. In this context, the parameters of the function $(k_1, \ldots, k_n)$ are software knobs that alter the behavior of the application (such as the parallelism level or the number of trials in a MonteCarlo simulation). Even if the functional behavior of the application is utterly under the control of the developer, extra-functional properties depend on the execution context. In particular, they usually depend

[4] https://github.com/fbeneventi/examon

on the underlying architecture, the actual input, the system workload and configuration (e.g. Dynamic Voltage Frequency Scaling) and the software knob configuration. Moreover, the application requirements might change at runtime and they typically address properties that are in contrast with each other, thus leading to trade-offs. For these reasons, selecting the most suitable configuration is a complex task.

To achieve this goal, the mARGOt [5] dynamic autotuning framework is rooted in the MAPE-K feedback loop [20]. In particular, it relies on the application knowledge, derived either at deploy time or at runtime, that states the expected behavior of extra-functional properties of interest. On one side, mARGOt to adapt uses runtime observations as feedback information for reacting to the evolution of the execution context. On the other side, it considers the features of the actual input to adapt in a more proactive fashion. Moreover, the mARGOt framework is designed to be flexible, defining the application requirements as a multi-objective constrained optimization problem that might change at runtime.

4.3 Power Management of HPC Resources

Currently, in HPC systems, computing elements and nodes are power constrained. For this reason, state-of-the-art processing elements embed the capability of fine-tuning their performance to control dynamically their power consumption. This includes dynamic scaling of voltage and frequency, power gating of the main architectural blocks of the processing elements, but also some feedback control logic to keep the total power consumption of the processing element within a safe power budget. This logic in x86 systems is named RAPL. However, demanding the power control of processing elements entirely to RAPL might not be the best choice. Indeed, it has been recently discovered that RAPL is application-agnostic and thus it tends to waste power in application phases which exhibit IOs or memory slacks. Under these circumstances, there are operating points more energy efficient than the one selected by RAPL while respecting the same power budget[10]. However, these operating points are only viable if the power capping logic is aware of the application requirements. Therefore, we developed a new power capping runtime based on a set of user space APIs to define the priority for tasks currently in execution on a given core[5, 11]. In parallel applications running on supercomputing systems, MPI primitives are used for exchanging data and for synchronizing processes. During synchronization primitives, significant power is waste without doing useful computation. In ANTAREX, we extended the power manager to be aware of these synchronization primitives without impacting performance.

5 USE CASE EVALUATION

This section highlights two application scenarios and the benefits introduced by applying the ANTAREX toolbox. Reported results will be further improved during the last six months of final validation.

5.1 Molecular Docking for Drug Discovery

The drug discovery process involves several tasks to be performed *in vivo*, *in vitro* and *in silico*. Molecular docking is a task typically performed *in silico*. It aims at finding the three-dimensional pose of a given molecule when it interacts with the target protein binding site. This task is often used for virtual screening a huge set of molecules to find the most promising ones, which will be forwarded to the later stages of the drug discovery process. Given the huge complexity of the problem, molecular docking cannot be solved by exploring the entire space of the ligand poses. This opens a lot of possible directions to address the molecular docking problem.

In the last ten years, Dompe' has undertaken an action in partnership with CINECA, to develop a commercial product, named LiGen [7], to perform *in silico* drug design. In Ligen, the most computational intensive module is LiGenDock [6], that implements the molecular docking phase. The main ANTAREX challenges for this use case are: (1) to develop an energy and resource efficient docking algorithm that adapts its configuration to the execution environment and (2) to scale-out with the ligand database size.

The ANTAREX contributions to this use case are related to several directions:

(i) We developed a drug design *mini-app* based on LiGenDock, named GeoDock-MA. It is licensed by DOMPE' free-of-charge for no profit purposes to serve as a demo and benchmarking tool for the drug design domain. The mini-app attempts to capture key computation kernels of the molecular docking application for drug discovery implemented in LiGenDock. By developing GeoDock-MA in parallel with the new version of LiGenDock, application developers can collaborate with system architects and domain experts to evaluate alternative algorithms that can either better satisfy end-user constraints, or better exploit architectural features.

(ii) We applied application autotuning approaches to the de-novo computer accelerated drug design based on LiGenDock. The goal was to demonstrate the possibility to optimize both energy-to-solution and time-to-solution by means of an improved task processing strategy and exploiting energy-awareness. We exploited the low-intrusiveness and low-overhead features of EXAMON to keep track continuously of the application behavior and its effect on the target architectures. This has been used to support manual and automatic tuning and to deliver application components to available resources that best fit the computational complexity of each task. Figure 2 shows the power and frequency profile of the use case application running on 4 nodes (8 sockets) with and without fine-grain power management optimizations. From the results, we can see how the power managemement library is capable of reducing the frequency and thus the power cost during load unbalanced phases.

(iii) We enhanced LiGenDock and the related mini-app with a set of software knobs, derived from approximate computing approaches, for trading off quality of the result and performance of the computation. The trade off has been used to control time-to-solution in the virtual screening task. Figure 3 shows the accuracy-performance trade-off extracted by changing approximation knobs at the application-level. The *overlap* is the cost function used to evaluate each molecule, the original version is the one in the bottom-left corner showing no overlap degradation. In particular, we can observe that the introduced parameterization, even when considering limited degradation, enables a large speedup in terms of application throughput. The new version of the docking application can be configured in different ways according to the goal of the virtual screening task: a faster and less precise way for large screening tasks or a slower and more precise way when targeting a more focused search.

[5] https://gitlab.com/margot_project/core

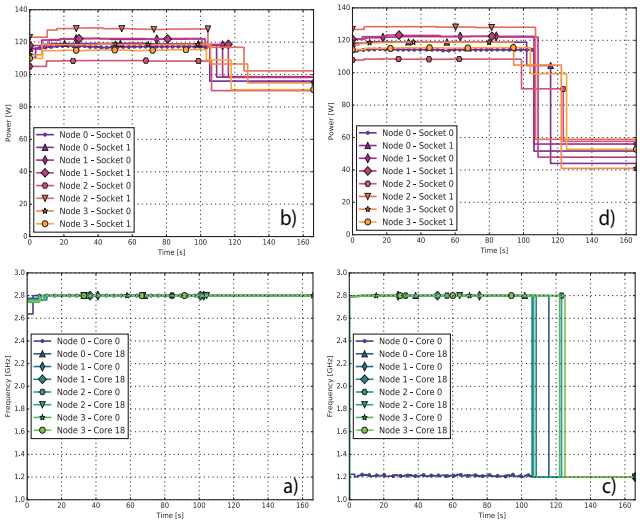

Figure 2: a) Frequency for core0 and 18 for each socket of the 4 nodes of MARCONI supercomputer running the mini-app; b) Power consumption of each socket; c) and d) report the same results when fine-grain power management is used.

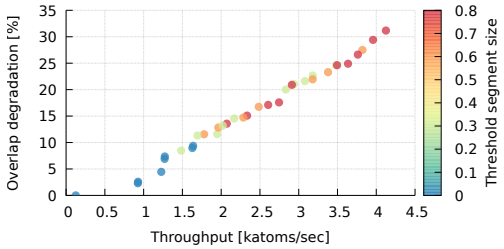

Figure 3: LiGenDock accuracy-performance trade-off. The color of each point represents the value of one of the several knobs introduced in the application.

(iv) We implemented an application model to estimate dynamically the task complexity and to inform the underlying system (and the user) about expected values in terms of time-to-solution or energy-to-solution metrics. In particular, this model provides the capability to tune the application accuracy, and thus the target knobs, according to a specified budget.

(v) Finally, the deep understanding of the application code offered us the possibility to build, aside of LiGenDock, a new module for molecular docking that is based only on geometric evaluations. This reduces a lot the cost of the docking process by enabling the user to increase the size of the ligand database to be processed during the virtual-screening task.

5.2 HPC-based Car Navigation System

Current navigation systems can be classified in two main categories: (1) those which can provide navigation in an offline mode (maps are stored on a device, route calculation is done locally on a device), and (2) those, which extensively and intensively use a server side, called online navigation (both for map loading and route calculation). The main disadvantage of the offline navigation is that it might suffer from not having the maps up-to-date, as well as from not knowing the online traffic situation. The other disadvantage is that the routing algorithm is limited in terms of performance capacity, thus leading to sub-optimal routes, thus heuristics with limited search space must be used. The main disadvantage of online navigation is that the Internet connectivity is not always stable, especially for moving objects, which often leads to slower responsiveness, and might even lead to a total failure of navigation. The best result can obviously be obtained when both worlds are combined in such a way that the best of the two worlds can be applied at the most appropriate moment. This was the situation when the collaboration between Sygic and IT4I started and the use case was designed. In particular, main goal of the use case was to build a navigation infrastructure for making use of (near)real-time data coming from client devices (navigator appliance or smart-phones) and other data sources in order to create a so-called global view on the current traffic situation. Moreover, the challenge was to maintain the efficient operation of the system in response to the dynamic nature of the traffic and the amount of navigation requests.

The improvements driven by ANTAREX tools in the self-adaptive navigation system have been the following:

(i) We designed and developed an infrastructure to collect traffic data by heterogeneous data sources. The system has been designed to handle simultaneously the processing of large volume of data from various sources. The system has been deployed on an HPC infrastructure to handle a large number of incoming routing requests, while maintaining the Service Level Agreement (SLA). Moreover, the high computational power provided by the infrastructure has been used to elaborate more accurately the traffic data, thus providing the highest quality response to the navigation requests.

(ii) We designed the HPC-side of the navigation system to optimize traffic over a particular area. The scenario of a smart city where all cars are self-driving has been considered. This impacted not only the scalability analysis of the deployed service, but also the algorithm implementing the optimal routing that are thought to be not only for the single user, but more towards the entire set of cars.

(iii) We used the mARGOt autotuner to maintain the computation efficiency according to the dynamic nature of the traffic. A first tangible result as the fine-grain tuning of the application was obtained on the Probabilistic Time-Dependent Routing (PTDR) algorithm based on the Monte Carlo method [18]. Given that this algorithm is the most computationally intensive part of our routing pipeline, the mARGOt autotuner was used to determine the minimal amount of samples needed to obtain distribution of the travel time with sufficient precision. In the PTDR case, the number of samples used was reduced by more than 50%. An example of the adaptivity in the number of sample selection can be seen in Figure 4 where we plot the number of samples needed to satisfy the extra-functional requirement when the same requests have been made every 15 minutes during the entire week. The daily behavior of rush hours where more complicated is the arrival time prediction is identified by requests with higher samples.

(iv) We adopted a similar approach – considering a broader perspective – to optimize the amount of compute resources occupied by the system on an HPC cluster according to extra-functional properties constraints (e.g. cost, quality of results, response time).

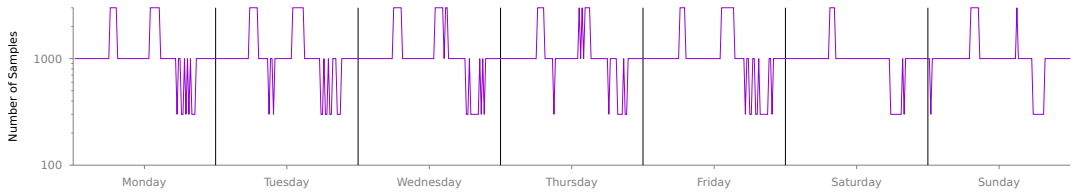

Figure 4: Adaptive number of samples for the PTDR kernel of the navigation system. The same navigation requests is repeated over the week every 15 minutes. The original static version was designed for the worst case, 3K samples.

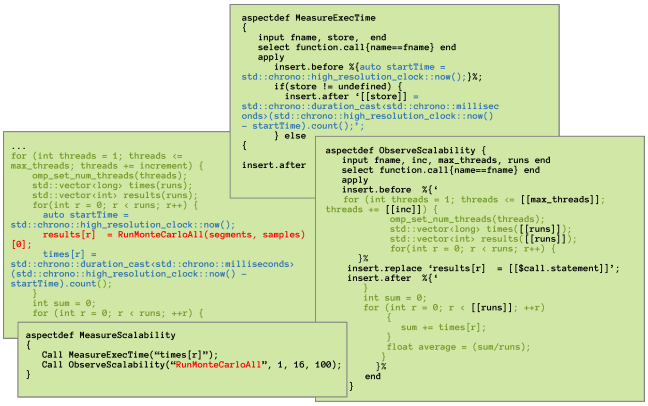

Figure 5: Example of LARA-DSL adoption

The implementation of such approach is currently in progress: We expect it to dynamically respond to variations in the routing requests load by adjusting the amount of occupied compute nodes by each part of the system, which will lead to more efficient operation of the system.

(v) We used the toolbox to integrate efficiently the new code features to existing codebase. This was successfully achieved by the integration of the LARA DSL in the built process to modify the source code in a transparent way, thus enabling a seamless integration of the necessary instrumentation needed by the tools. The DSL has been used to integrate the mARGOt autotuner, evaluate the code scalability analysis [19] (see Figure 5) and to generate parts of the data access API which handles the road network graph stored in a HDF5 format [17].

6 CONCLUSIONS

Exascale HPC systems need the definition of new software stacks to fully exploit heterogeneity and meeting power efficiency requirements. ANTAREX provides an holistic programming framework capable to decouple functional and extra-functional aspects of the application for energy efficient supercomputing systems. In this paper, we have presented the ANTAREX toolbox and some preliminary results carried out on two different case studies.

REFERENCES

[1] A. H. Ashouri, A. Bignoli, G. Palermo, C. Silvano, S. Kulkarni, and J. Cavazos. 2017. MiCOMP: Mitigating the Compiler Phase-Ordering Problem Using Optimization Sub-Sequences and Machine Learning. *ACM TACO* 14, 3 (2017), 29:1–29:28.

[2] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano. 2018. A Survey on Compiler Autotuning using Machine Learning. *Comput. Surveys* (2018).

[3] A. H. Ashouri, G. Mariani, G. Palermo, E. Park, J. Cavazos, and C. Silvano. 2016. COBAYN: Compiler Autotuning Framework Using Bayesian Networks. *ACM TACO* 13, 2, Article 21 (June 2016), 25 pages.

[4] A. H. Ashouri, G. Palermo, J. Cavazos, and C. Silvano. 2018. *Automatic Tuning of Compilers Using Machine Learning.* Springer.

[5] A. Bartolini, R. Diversi, D. Cesarini, and F. Beneventi. 2017. Self-Aware Thermal Management for High Performance Computing Processors. *IEEE Design & Test* (2017).

[6] C. Beato, A. Beccari, C. Cavazzoni, S. Lorenzi, and G. Costantino. 2013. Use of experimental design to optimize docking performance: The case of ligendock, the docking module of ligen, a new de novo design program. *Journal of Chemical Information and Modeling* 53, 6 (2013), 1503–1517.

[7] A. Beccari, C. Cavazzoni, C. Beato, and Gabriele G. Costantino. 2013. LiGen: a High Performance workflow for chemistry driven de novo design. *Journal of Chemical Information and Modeling* 53, 6 (2013), 1518–1527.

[8] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini. 2017. Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools. In *Proc. of DATE*. 1038–1043.

[9] J. Cardoso, J. Coutinho, T. Carvalho, P. Diniz, Z. Petrov, W. Luk, and F. Gonçalves. 2016. Performance-driven instrumentation and mapping strategies using the LARA aspect-oriented programming approach. *Software: Practice and Experience* 46, 2 (2016), 251–287.

[10] D. Cesarini, A. Bartolini, and L. Benini. 2017. Benefits in Relaxing the Power Capping Constraint. In *Proceedings of ANDARE Workshop.*

[11] D. Cesarini, A. Bartolini, and L. Benini. 2017. Prediction horizon vs. efficiency of optimal dynamic thermal control policies in HPC nodes. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC).*

[12] S. Cherubin and G. Agosta. 2018. libVersioningCompiler: An easy-to-use library for dynamic generation and invocation of multiple code versions. *SoftwareX* 7 (2018), 95 – 100.

[13] C. Silvano et al. 2016. The ANTAREX approach to autotuning and adaptivity for energy efficient HPC systems. In *Proc. of the ACM International Conference on Computing Frontiers, CF'16.* 288–293.

[14] C. Silvano et al. 2016. AutoTuning and Adaptivity appRoach for Energy efficient eXascale HPC systems: the ANTAREX Approach. In *Proc. of DATE.* 1518–1527.

[15] C. Silvano et al. 2018. ANTAREX: A DSL-based Approach to Adaptively Optimizing and Enforcing Extra-functional Properties in High Performance Computing. In *Euromicro Conference on Digital System Design - DSD.*

[16] D. Gadioli, R. Nobre, P. Pinto, E. Vitali, A.H. Ashouri, G. Palermo, C. Silvano, and J. Cardoso. 2018. SOCRATES - A Seamless Online Compiler and System Runtime AutoTuning Framework for Energy-Aware Applications. In *Proc. of DATE.* 1149–1152.

[17] M. Golasowski, J. Bispo, J. Martinovič, K. Slaninová, and J. Cardoso. 2017. Expressing and Applying C++ Code Transformations for the HDF5 API Through a DSL. In *IFIP International Conference on Computer Information Systems and Industrial Management.* 303–314.

[18] M. Golasowski, R. Tomis, J. Martinovič, K. Slaninová, and L. Rapant. 2016. Performance evaluation of probabilistic time-dependent travel time computation. In *IFIP International Conference on Computer Information Systems and Industrial Management.* 377–388.

[19] M. Golasowski R. Cmar J. Cardoso J. Bispo G. Palermo D. Gadioli J. Martinovic, K. SlaninovÃą and C. Silvano. 2016. DSL and Autotuning Tools for Code Optimization on HPC Inspired by Navigation Use Case. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis.*

[20] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[21] R. Nobre, L. Martins, and J. Cardoso. 2016. A Graph-Based Iterative Compiler Pass Selection and Phase Ordering Approach. (2016), 21–30.

[22] A. Suresh, E. Rohou, and A. Seznec. 2017. Compile-Time Function Memoization. In *26th International Conference on Compiler Construction.* Austin, United States.

[23] A. Suresh, B. Narasimha Swamy, E. Rohou, and A. Seznec. 2015. Intercepting Functions for Memoization: A Case Study Using Transcendental Functions. *ACM TACO* 12, 2 (2015), 23.