The ANTAREX Tool Flow for Monitoring and Autotuning Energy Efficient HPC Systems

Cristina Silvano*, Giovanni Agosta*, Jorge Barbosa**, Andrea Bartolini†
Andrea R. Beccari‡, Luca Benini†, João Bispo**, João M.P. Cardoso**
Carlo Cavazzoni§, Stefano Cherubin*, Radim Cmar††, Davide Gadioli*
Candida Manelfi‡, Jan Martinovič‖, Ricardo Nobre**, Gianluca Palermo*
Martin Palkovič‖ Pedro Pinto**, Erven Rohou¶, Nico Sanna§, Kateřina Slaninová‖
*DEIB – Politecnico di Milano {cristina.silvano, giovanni.agosta, stefano.cherubin, davide.gadioli, gianluca.palermo}@polimi.it
†IIS – Eidgenössische Technische Hochschule Zürich {barandre, lbenini}@iis.ee.ethz.ch
‡Dompé Farmaceutici SpA {andrea.beccari, candida.manelfi}@dompe.it
§CINECA {c.cavazzoni, n.sanna}@cineca.it
¶Inria erven.rohou@inria.fr
‖IT4Innovations, VSB – Technical University of Ostrava {jan.martinovic, katerina.slaninova, martin.palkovic}@vsb.cz
**FEUP – Universidade do Porto {jbispo, jmpc, jbarbosa, p.pinto, rjfn}@fe.up.pt
††Sygic rcmar@sygic.com

***Abstract*—Designing and optimizing HPC applications are difficult and complex tasks, which require mastering specialized languages and tools for performance tuning. As this is incompatible with the current trend to open HPC infrastructures to a wider range of users, the availability of more sophisticated programming languages and tools to assist and automate the design stages is crucial to provide smoothly migration paths towards novel heterogeneous HPC platforms. The ANTAREX project intends to address these issues by providing a tool flow, a Domain Specific Launguage and APIs to provide application's adaptivity and to runtime manage and autotune applications for heterogeneous HPC systems. Our DSL provides a separation of concerns, where analysis, runtime adaptivity, performance tuning and energy strategies are specified separately from the application functionalities with the goal to increase productivity, significantly reduce time to solution, while making possible the deployment of substantially improved implementations. This paper presents the ANTAREX tool flow and shows the impact of optimization strategies in the context of one of the ANTAREX use cases related to personalized drug design. We show how simple strategies, not devised by typical compilers, can substantially speedup the execution and reduce energy consumption.**

***Keywords*—High-Performance Computing, Autotuning, Adaptivity, DSL, Compilers, Energy Efficiency**

I. Introduction

The ability to port applications designed for current platforms, based on GPGPUs like the NVIDIA Kepler or Tesla families, to heterogeneous systems such as those currently designed for embedded systems is critical to provide software support for future HPC systems.

Designing and implementing HPC applications is a difficult and complex task, which requires 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. To fulfill the 20MW target, energy-efficient heterogeneous supercomputers need to be coupled with a radically new software stack capable of exploiting the benefits offered by heterogeneity at different levels (supercomputer, job, node).

There has been a change in the focus in HPC from purely scientific challenges and rather restrict industrial domains to a point where it is now recognized as a powerful technology to increase the competitiveness of nations and their industrial sectors, including small scale but high-tech businesses – *to compete, you must compute* has become an ubiquitous slogan [1].

The current roadmap for HPC systems aims at reaching the Exascale level (10^{18} FLOPS) within the 2022–23 time frame, providing massive increases in computational capabilities, while still significantly limiting the energy envelope – the target power envelope for future Exascale system ranges between 20 and 30 MW. Nowadays, we see a trend of "Green" HPC systems, which are designed with a FLOPS/W metric in mind, rather than just FLOPS. Heterogeneous architectures are increasingly more frequent in such systems as they provide better performance than their homogeneous counterparts[1]. However, there is still a large gap to fill, as even with such heterogeneous architectures, the achieved efficiency is still two orders of magnitude lower than that needed for supporting Exascale systems at the target power envelope of 20 MW.

[1]www.green500.org, June 2015

978-1-5386-3437-0/17/$31.00 ©2017 IEEE

The main goal of the ANTAREX project [2, 3] is to express by a DSL the application self-adaptivity and to run-time manage and tune 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 the application functionality. This is 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, runtime adaptivity/energy/performance strategies.

This paper presents the toolflow of the ANTAREX project and briefly describes each module. Each module targets a specific layer of the HPC system and, together, they cover the entire stack, from the software to the hardware. In ANTAREX, we developed a DSL to specify adaptivity strategies that will enhance applications and systems to adapt, at runtime, to their execution context and meet the Exascale goals. More specifically, we can target key parts of the application and provide them with capabilities to read information from the system and act on it, use techniques such as autotuning for software knobs, split compilation and power and resource management.

The remainder of this paper is organized as follows. Section II briefly describes the ANTAREX project. Section III presents the ANTAREX toolflow and its main stages and components. Section IV presents results considering code transformation strategies and thread-level parallelization strategies via OpenMP in the context of the personalized drug design ANTAREX use case. Section V briefly introduces related work and related FET-HPC projects. Finally, Section VI concludes the paper.

II. About ANTAREX

In this section we introduce the project, namely the members of the consortium, the applications that drive our research scenarios and, finally, the target platforms used in the HPC centers of our partners.

A. Consortium

The ANTAREX Consortium comprises a wealth of expertise in all pertinent domains. Four top-ranked academic and research partners, Politecnico di Milano, ETHZ Zurich, University of Porto and INRIA, are complemented by the Italian Tier-0 Supercomputing Center, CINECA, the Tier-1 Czech National Supercomputing Center, IT4Innovations and two industrial application providers, Dompé, one of the leading biopharmaceutical companies in Europe, and Sygic, the top European navigation software company.

B. Application Scenarios

The ANTAREX project is driven by two industrial HPC applications chosen to address the self-adaptivity and scalability characteristics of two highly relevant scenarios towards the Exascale era. These are briefly explained below.

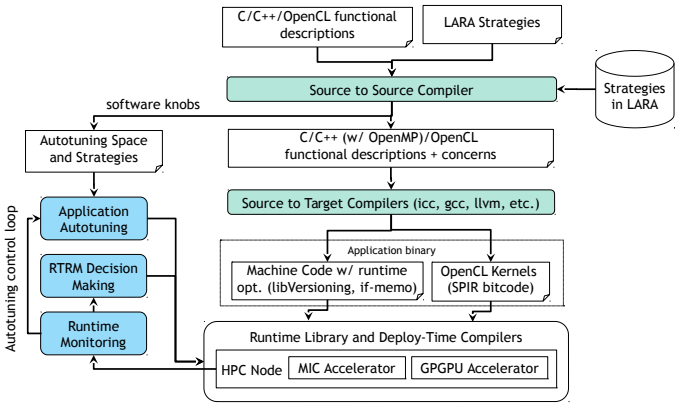

Fig. 1. The ANTAREX Tool Flow

a) Computer Accelerated Drug Discovery: Computational discovery of new drugs is a compute-intensive task: it is critical to explore the huge space of chemicals with potential applicability as pharmaceutical drugs. Typical problems include the prediction of properties of protein-ligand complexes (such as docking and affinity) and the verification of synthetic feasibility. These problems are massively parallel, but demonstrate unpredictable imbalances in the computational time, since the verification of each point in the solution space requires a widely varying time.

b) Self-Adaptive Navigation System: To solve the growing automotive traffic load, it is necessary to find the best utilization of an existing road network, under a variable workload. The basic idea is to provide contextual information from server-side to traditional mobile navigation users and vice versa. The approach will help to overcome the major shortcomings of the currently available navigation systems exploiting synergies between server-side and client-side computation capabilities.

C. Target Platforms

The target platforms are CINECA's Tier-1 IBM NeXtScale hybrid Linux cluster, based on Intel TrueScale interconnect as well as Xeon Haswell processors and MIC accelerators, and IT4Innovations Salomon supercomputer, which is a PetaFlop class system consisting 1008 computational nodes. Each node is equipped with 24 cores (two twelve-core Intel Haswell processors). These computing nodes are interconnected by InfiniBand FDR and Ethernet networks. Salomon also includes of 432 compute nodes with MIC accelerators.

III. Toolflow

The ANTAREX approach and related tool flow, shown in Figure 1, operates both at design-time and runtime. The application functionality is expressed through C/C++ code (possibly including legacy code), whereas the non-functional aspects of the application, including parallelization, mapping, and adaptivity strategies are expressed through the DSL developed in the project. In the definition of these strategies, the application developer or system integrator can leverage DSL strategy libraries that encapsulate specific mechanisms,

including how to generate code for OpenCL or OpenMP parallelization, and how to interact with the runtime resource manager. The DSL weaver and refactoring tool then enhance the C/C++ functional specification with the desired adaptivity strategies, generating a version of the code that includes the necessary libraries as well as the partitioning between the code for the general-purpose processors and the code for the accelerators (such as GPGPUs and MIC accelerators [4]). A mix of off-the-shelf and custom compilers will be used to generate code, balancing development effort and optimization level.

The ANTAREX compilation flow leverages a runtime phase with compilation steps, through the use of split-compilation techniques. Application autotuning is delayed to the runtime phase, where the software knobs (application parameters, code transformations and code variants) are configured according to runtime information coming from the execution environment. Finally, runtime resource and power managers are used to control the resource usage for the underlying computing infrastructure given the changing conditions. In the design phase, the application is augmented with control code that, at runtime, will provide monitoring and adaptivity strategies, as specified by the DSL extra-functional specification. Thus, the application is continuously monitored to guarantee the required Service Level Agreement (SLA), while communication with the runtime resource-manager takes place to control the amount of processing resources needed by the application. The application monitoring and autotuning are supported by a runtime layer implementing an application level collect-analyse-decide-act loop.

A. DSL

HPC applications might profit from adapting to operational and situational conditions, such as changes in contextual information (e.g., workloads), in requirements (e.g., deadlines, energy), and in resources availability (e.g., connectivity, number of processor nodes available). To accomplish this, the ANTAREX toolflow relies on LARA [5, 6], a DSL inspired by AOP concepts [7], which promotes a separation of concerns which would otherwise be difficult to achieve. We write self-adaptive strategies with LARA, which make use of the other developed tools, and perform an additional compilation step, called *weaving*, which merges these additional concerns with the main application.

The use of the DSL in ANTAREX will be crucial to decouple the functional specification of the application from the definition of the adaptivity strategies, e.g., the definition of software knobs, such as code variants or application parameters.

The LARA language and framework have been developed during this project and we support not only code insertions but also specific weaver actions that deal with code transformations, refactorings, software/hardware partitioning [8], compiler optimization sequences [9], among other concerns.

One of the most important developments of this framework is the implementation of libraries of strategies that enable us to make use of the other tools of the ANTAREX toolflow.

Figure 2 shows an example of a LARA aspect that uses the libVersioningCompiler LARA library to enhance an application with the capability to perform compilation and loading of specific functions during runtime. This LARA library exposes the high-level API provided by `libVersioningCompiler`[2].

The first step is to import the LARA library, which is performed in the first line of the aspect definition. This library is provided and distributed with the Clava weaver, the `C++` weaver developed during ANTAREX. The `input` block (lines 4 – 8) parameterizes this strategy with the name of the target function as well as the points in the code where key code will be inserted. The call to `vcSetup`, in line 10, performs several tasks. First, it inserts an initialization call at the start of the main function. Then, it adds the include directives needed to work with the versioning compiler. After this, and if needed, the setup decorates the definition of the target function in order to force `C` linkage. Finally, this setup declares a `typedef` of the function signature, an important piece of information needed for the following steps.

The call to `makeVersion`, in line 12, is responsible for generating the code that will instantiate a specific version (based on the provided options) and insert it at the correct location, which is provided by the `$versionTarget` join point. In this example, we are only providing an option option to compile with the `-O2` flag. Next, in line 14, the call to `compile` will generate the code that will force start the compilation and loading of the newly generated symbol. It will also insert it at the correct place, as pointed by the `$compileTarget` join point. This call returns the name of the function pointer that can be used inside the code to call the newly compiled version. From lines 16 to 20 we select all function calls to the original function and change them to use this function pointer instead.

This is a simple example to illustrate how the DSL and weaver provide libraries that are used to enhance applications with the tools from the ANTAREX toolflow, which are the building blocks of the strategies we develop for self-adaptivity. These concepts can be used for more sophisticated transformations, generating different versions based on location, inputs and previous profiling, resulting in a much more customized end result.

B. Split Compilation

The ANTAREX DSL approach aims at reaching a higher abstraction level, to separate and express data communication and computation parallelism, and to augment the capabilities of existing programming models by passing hints and metadata to the compilers for further optimization. The approach aims at improving performance portability with respect to current programming models, such as OpenCL, where fine-tuning of performance (which is very sensitive to even minimal variation in the architectural parameters [10, 11]) is left entirely to the programmer. This is done by exploiting the capabilities of the

[2]https://github.com/skeru/libVersioningCompiler

```
import antarex.tools.versioningcompiler;

aspectdef
  input
    funcName,
    $versionTarget,
    $compileTarget
  end

  call vc : vcSetup(funcName);

  vc.makeVersion([{'otp', '-O', '2'}], $versionTarget);

  ptr = vc.compile($compileTarget);

  select fCall end
  apply
    def name = ptr;
  end
  condition $fCall.name == funcName end
end
```

Fig. 2. Example of LARA aspect for the inclusion of `libVersioningCompiler` in a given application.

DSL to automatically explore the configuration space for the parallel code.

To this end, iterative compilation [12] techniques are attractive to identify the best compiler optimizations for a given program/code fragment by considering possible trade-offs. Given the diversity of heterogeneous multiprocessors and the potential for optimizations provided by runtime information, runtime optimization is also desirable. To combine the two approaches, *split compilation* will be used. The key idea is to split the compilation process in two steps - offline, and online - and to offload as much of the complexity as possible to the offline step, conveying the results to runtime optimizers [13]. We will express code generation strategies to drive a dynamic code generator in response to particular hardware features as well as dynamic information. This combination of iterative- and split-compilation will have a significant impact on the performance of applications, but also on the productivity of programmers by relieving programmers from the burden of repeatedly optimizing, tuning, compiling and testing.

One of the ways in which we provide split compilation is the use of the `C++` library for online compilation, `libVersioningCompiler`[3]. This library is able to dynamically compile a single function and load it as a function pointer, which allows the program to store multiple versions of the same target function. It is possible to choose among different compilers and define several options and flags which will lead to different versions. This library exposes both high- and low-level interfaces, suiting different users based on their current needs or experience.

Related to this topic, there is also the possibility of performing runtime specialization based on the inputs of function calls of hot functions. For instance, the `if-memo` library [14] is able to generate a memoization [15] table and store the results of the computation of pure functions for their frequently used inputs. This is able to save execution time as the program can skip entirely the computation of such calls (if the input is known) and instead return the saved result. Because this table is built online, it has great flexibility and is able to adapt to the input data. This library works in a non-intrusive way as it intercepts calls to dynamically-loaded libraries and changes them to call their version instead. Therefore, the user does not need to change the original code and may even use this library on a binary compiled without any special option. We are currently exploring profiling-based memoization, which instruments the code to gather input frequency data, builds the memoization table offline and inserts it in the application. These operations are controlled using strategies written with the LARA DSL.

C. Autotuning

The management of system adaptivity and autotuning is a key issue in HPC systems, where the system needs to react promptly to changing workloads and events, without impacting too much the extra-functional characteristics, such as energy and thermal features [16, 17]. The motivation can be easily explained by the requirement to meet the maximum performance/power ratio across all the possible deployments of the applications. This is especially important when considering the rapid growth of computing infrastructures that continue to evolve on one hand by increasing computing nodes, while on the other hand by increasing the performance exploiting heterogeneity in terms of accelerators/co-processors. Thus, there is a requirement on applications to become adaptive with respect to the computing resources [18]. In this direction, another interesting effect is that there is a growing need of guaranteeing SLA both at the server- and at the application-side. This need is related to the performance of the application, but also to the maximum power budget that can be allocated to a specific computation. In this context, efforts are mainly focused on two main paths: i) the development of an autotuning framework to configure and to adapt application-level parametersand ii) to apply the concept of precision autotuning to HPC applications.

Within ANTAREX the two approaches have been faced by proposing an application autotuning approach that starting from the idea of non-domain knowledge, it relies on code annotations done by the usage of the LARA DSL to select the area of interest, to expose and select the tunable knobs and to shrink the search space by focusing the autotuner on a certain subspace. One of the core features of our self-adaptive approach is the ability of changing the value of certain software knobs at runtime in order to control application parameters, code transformation parameters, algorithm choices and even framework parameters (e.g., OpenMP policies). To achieve this goal, we developed the mARGOt autotuning framework[4], which provides an adaptation layer to any given application. There are two main modules to this framework, the application monitors (e.g., time or throughput) and the application-specific runtime manager, which selects the most suitable configuration according to the current information, application knowledge

[3]https://github.com/skeru/libVersioningCompiler

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

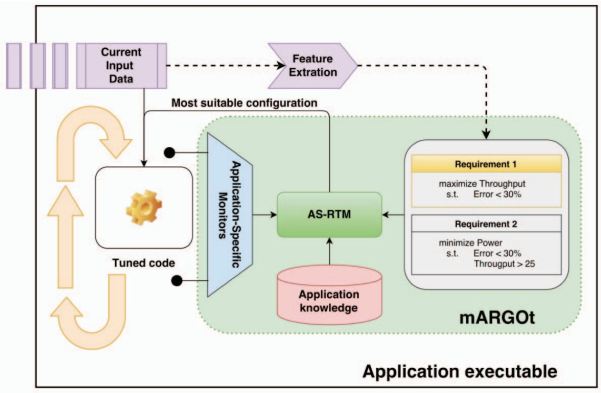

Fig. 3. mARGOt autotuning framework overview

(extracted at profile time) and the application requirements, see Figure 3.

The monitoring, together with application properties/features, represents the main support to the decision-making during the application autotuning phase since it is used to perform statistical analysis related to system performance and other SLA aspects. Continuous monitoring during the execution is adopted to update the knowledge from the data collected by the monitors, giving the possibility to autotune the system according to the most recent operating conditions.

D. Runtime Resource & Power Management

ANTAREX focuses on a holistic approach towards next-generation energy-efficient Exascale supercomputers. While traditional design of green supercomputers relies on the integration of best-in-class energy-efficient components [19], recent works [20, 21] show that as an effect of this design practice supercomputers are nowadays heterogeneous systems. Indeed, supercomputers are not only composed of heterogeneous computing architectures (GPGPUs and CPUs), but different instances of the same nominal component execute the same application with 15% of variation in the energy consumption. Different applications on the same resources show different performance-energy trade-offs, for which an optimal selection of operating points can save from 18% to 50% of node energy with respect to the default frequency selection of the Linux OS power governor. Moreover it has been recently shown that environmental conditions, such as ambient temperature, can significantly impact the overall cooling efficiency of a supercomputer, causing more than 10% *Power usage effectiveness* (PUE) loss when transitioning from winter to summer [22]. These sources of heterogeneity, coupled with current worst case design practices, lead to significant loss in energy-efficiency, and to some missed opportunities for runtime resource and power management (RTRM, RTPM). ANTAREX leverages RTRM and RTPM by combining: (1) novel introspection points, application progress and dynamic requirements; (2) autotuning capabilities enabled by the DSL in the applications; and (3) information coming from the processing elements of the Exascale machine and IT infrastructure and their related performance knobs (such as Dynamic Voltage and Frequency Scaling, cooling effort, room temperature). Information flows converge in a scalable multilayer resource management infrastructure. The information will be used to allocate to each application the set of resources and their operating points to maximize the overall supercomputer energy-efficiency, while respecting SLA and safe working conditions. The latter will be ensured by the resource management solution by optimal selection of the cooling effort and by a distributed optimal thermal management controller. The ANTAREX power management and resource allocation approach is based on: (1) Expanding the energy/performance control capabilities by introducing novel software control knobs (such as software reconfigurability and adaptability); (2) Designing scalable and hierarchical optimal control-loops capable of dynamically leveraging the control knobs together with classical performance/energy control knobs at runtime (task mapping and DVFS); (3) Monitoring the supercomputing evolution, the application status and requirements, bringing this information to the energy/performance-aware software stack. This approach will always enable the supercomputer and each application to operate at the most energy-efficient level.

One of the key aspects of power and resource management is the monitoring of the runtime status of a program. To this end, we developed a monitoring and profiling framework, Examon[5]. This is a portable and extensible framework to measure energy consumption as well as to access architecture-specific metrics (e.g., from hardware counters). With Examon, it is possible to collect information and distribute it across several remote information collectors. This framework uses the MQTT messaging protocol in order to transfer the data to the front-end, where it can be visualized, stored or exposed to an application manger using the provided API. Examon supports scalable analytics and knowledge extraction from the monitored data [23].

An actual power manager[6] has also been developed in the ANTAREX project. The power manager implements power capping strategies to select the best performance point for each core to maintain a given power constraint. The user can assign a priority to any given core and this will be taken into account by the power manager when assigning frequencies. The implementation mainly consists of an iterative optimization system, where at each iteration the manager takes into account the resources used by the previous iteration.

IV. Strategies and Performance Evaluation

In this section, we show a number of code transformations we have selected for performance improvement in terms of execution time and energy consumption for a code section of a preliminary version of the ANTAREX use case regarding personalized drug design. The application loads a number of ligands from a database and performs rotation-based geometric transformations on them in order to calculate the best overlap

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

[6]http://data-archive.ethz.ch/delivery/DeliveryManagerServlet?dps_pid=IE5768287

TABLE I
EXPERIMENTAL SETUP

	Setting	Value
Hardware	CPU	Intel Xeon E5-2630 v3
	#CPUs	2
	CPU Frenquency	2.40GHz
	RAM	128GB
	Energy Measure	CPU + RAM
System	OS	Ubuntu 16.04
	Compiler	GCC 5.4. -Ofast
OpenMP	PLACES	threads
	NUM_THREADS	32
	PROC_BIND	true
Application	#Ligands	500

TABLE II
LIST OF OPTIMIZATIONS USED FOR EACH VERSION.

Version	Optimizations
v1	1
v2	1, 2
v3	1, 3
v4	3
v5	3, 4
v6	1, 2, 3
v7	1, 2, 4
v8	3, 4, 5
v9	3, 4, 5, 6
v10	3, 4, 5, 6, 7
v11	3, 4, 5, 6, 7, 8

possible on the given target molecules. These computations represent the docking step needed for the simulation part of the personalized drug discovery.

The executions of the application and measurements have been carried out on a workstation that represents a single node of target HPC system available at CINECA supercomputing center. A description of the hardware of the node as well as system information and OpenMP parameters is shown in Table I.

A. Versions and Optimizations

We consider here as reference the sequential version of the code (in C++ and named as "original") and the execution run on a single node. The multiple versions of the application are compared against the original. Each version is composed of at least one code optimization. The list of versions and the associated code optimizations are presented in Table II.

The code transformations applied according to the versions are the following:

a) Avoiding implicit casts.: This transformation changes the type of two reduction variables declared inside the loops of the critical function of the application. These variables are declared as `float`, but being assigned values from functions returning `double`. By changing the types of the declared variables, we were able to reduce the number of casts performed during the execution.

b) Parallelization of `matchProbeShape` loops.: This transformation uses OpenMP to parallelize the loops in the function `matchProbeShape`, which is one of the functions in the critical execution path of the program. Some variables need be kept private for each thread, so they were copied as needed. Moreover, specific regions inside the loop are marked as critical for OpenMP.

c) AoS to SoA conversion.: This transformation converts one of the main containers of the application in order to achieve better vectorization performance. The transformation implies the insertion of code that copies the contents of a vector of atoms into three different vectors, each holding data for a different spacial coordinate, X, Y and Z. The functions that use this container have also been changed. The modifications include changing their signatures and functions calls, as well as adapting the code sites where the coordinates data is actually used.

d) Parallelization of calls to `matchProbeShape`.: This transformation moves the parallelization from inside the function `matchProbeShape`, in its loops, to the outside where this function is called. We found the loops where this function is called and annotated them with OpenMP pragmas. This enabled us to achieve higher levels of parallelism.

e) Change in the main data type of the program.: This transformation completely changes the data type of the main data structure used in the program from `double` to `float` in order to achieve better vectorization performance. The three vectors introduced in optimization 3 (AoS to SoA) have been changed to contain single precision floating point values. Finally, we updated the functions that use these arrays, changing the functions calls and the corresponding function signatures.

f) Replacement of the map structure.: This is a simple transformation that changes the map structure used to hold atom information for every molecule. This was possible because, in the critical region, the features of the map were not being used. Rather, the data structure was being used as an array. The changes are very simple and just target the declaration of the map and the declaration of the iterators used to traverse it.

g) Efficient argument passing.: This transformation changes the way arguments are passed to one of the functions that is most frequently called in the application. The function `Rotate` has, among other parameters, a constant vector of pointers to `double`. This vector was being copied every time the function was called, which we avoid by forcing this to be passed by reference. After updating the function signature, no other change was needed.

h) Memory alignment.: The vectors that were generated in optimization 3 (AoS to SoA) are now allocated with special calls that guarantee the memory is aligned to 32-byte boundaries. Furthermore, this optimization includes the addition of a `SIMD` pragma at the loop that uses these vectors inside the critical function of the program.

B. Time and Energy Measurements

In these experiments, we measured both execution time and energy consumption of each of the previous versions of the application and compare them to the original one. The original

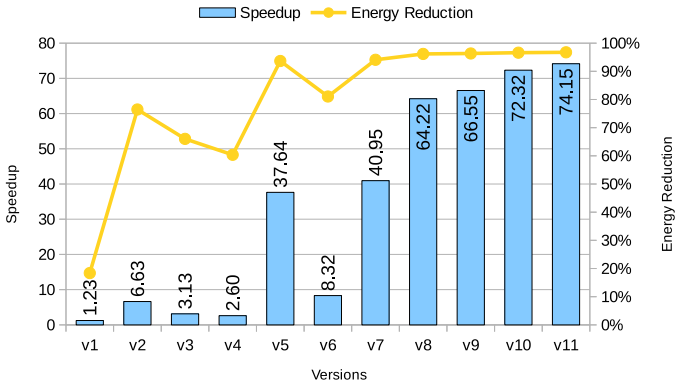

Fig. 4. Speedup and energy reduction by varying the versions of the original application.

version is a serial application and runs on a single CPU, while most of the generated versions are parallel and use all available threads (32). As mentioned in the experimental setup description, the energy consumption was measured considering both CPU and RAM.

Figure 4 presents the speedup and energy reductions (in percentage) achieved by each code version with respect to the original code.

As shown in Figure 4, the code optimizations and the thread-level based OpenMP parallelization enabled us to achieve significant speedup and energy reduction. The energy reduction follows the same trend of the speedup: as the speedup grows, so does the energy reduction (although the reduction in execution time is slightly greater than the reduction in energy).

Considering the code properties, the largest increases are related to the parallelization of the application (for instance optimization 4 in version 5). Another change which results in more parallelism (although not through OpenMP) is the modification introduced with version 8. In this version, we introduce optimization 5, which changes the data types across the sections of the program that use the main data structures. By moving from `double` to `float` we were able to theoretically double the number of floating point operations executed in the vector units in the Xeon processors, which led to an increase in performance which is almost twice than the previous more similar version (5).

The only parameter passed to the application is the number of ligands to be loaded from a database and used in the computation. The results presented here are scalable in terms of this parameter, as further increasing the number of ligands would not have a noticeable impact on speedup and energy consumption reductions.

Further work includes the exposition of a number of parameters (such as number of threads) to the ANTAREX autotuner, additional code transformations, and considering the execution of the application in multiple nodes (e.g., using an MPI version of the code).

V. Related Work

The are several approaches for achieving higher energy-efficiency in HPC.

We can identify previous work using DSLs to guide code transformation and optimization strategies through scripting recipes [24, 25, 26] and also through annotations [27, 28, 29], which are somewhat less complex but also less powerful and coarser-grained. One of the major differences to these annotation-based approaches is that LARA strategies are specified in different files and are separated from the original application code. LARA is also different from the recipe-based approaches since it enables whole-program transformations which leads to the development of strategies for runtime adaptivity. While the mentioned approaches generate a single, transformed version, LARA enables the generation of multiple versions as well as the means to switch them at runtime based on any given software knob.

Concerning autotuning, there are several offline approaches focusing the tuning of OpenMP parameters [30, 31, 32], which may also be knobs on the ANTAREX approach. There are also frameworks focused on more general, application-level autotuning [33, 34, 35]. However, these are either intrusive and require source code changes [33, 34], which our DSL avoids, or require that the application has pre-exposed knobs to tune, which we can also do as part of our strategy description with the LARA DSL.

We can also identify projects in the same domain as ANTAREX and even share common goals, although they propose to reach them following different paths. The READEX project [36] shares some goals with ANTAREX, especially the focus on energy efficiency in the HPC domain, and also intends to target the entire stack available at HPC centers, from software, system, to hardware. Their approach is based on two phases. First, in the design phase, the application is explored and analyzed in order to find runtime situations (that compose the dynamic nature of the application). In this phase, by using an extension to the Periscope Tuning Framework, optimal parameters for each situation are found and similar situations are grouped together in application scenarios, which, alongside the configurations are stored in a tuning model. Then, in the second phase, corresponding to actual production executions, this tuning model is used by their developed runtime library in order to predict the upcoming scenarios and switch to the corresponding optimal configurations. There is an addition calibration step in this phase in which the runtime component tries to refine the tuning model if it encounters a never-before-seen situation. One of the key differences between the READEX approach and the one presented by ANTAREX is the use of a DSL as a way of specifying the adaptivity strategies and bring together the control over all parts of the toolchain. This allows developers, system administrators, and domain experts to take full advantage of all the control provided by our approach, allowing them to target software, system and hardware knobs.

The AllScale project [37] proposes a toolchain that implements a parallel programming model based on nested recursive parallelism [38] as a way to specify parallelism in a target-independent way. The programming model is based on a C++ API that can be used in the code to be parallelized (e.g.,

`pfor` operator). At program execution, the AllScale runtime environment decides how and where the parallel sections are run. The AllScale approach separates the specification of parallelism from its implementation and legacy code must be manually modified to take advantage of the AllScale toolchain. In comparison, in the ANTAREX tool flow parallelization is applied using strategies specified with the LARA DSL.

Both ANTAREX, READEX, and AllScale have many complementary aspects which can be combined. With respect to the ANTAREX approach, we foresee opportunitis to use the LARA DSL, the monitoring and the autotuning approaches in the context of other approaches such as the ones presented by the READEX and AllScale projects.

VI. Conclusions

The goal of the ANTAREX project is to provide a holistic system-wide adaptive approach for next generation HPC systems. Our long-term vision is to explore an innovative application programming paradigm and description methodology to decouple functional and extra-functional aspects of the application. This paper introduced the ANTAREX tool flow and its main components, including the DSL, source to source compiler, approach to split compilation, and autotuning. We proved the importance of simple code transformations, not considered by typical compilers, to improve performance and energy consumption. These code transformations together with OpenMP based parallelization enabled us to achieve significant improvements on one of the ANTAREX use cases.

Ongoing work is focused on the source to source compiler and on the integration of all the ANTAREX tool flow components.

Acknowledgments

The ANTAREX project is supported by the EU H2020 program under grant no. 671623.

References

[1] J. Curley, "HPC and Big Data," *Innovation*, vol. 12, no. 3, Jul. 2014.

[2] C. Silvano, G. Agosta, A. Bartolini, A. Beccari, L. Benini, J. M. P. Cardoso, C. Cavazzoni, R. Cmar, J. Martinovic, G. Palermo, M. Palkovic, E. Rohou, N. Sanna, and K. Slaninova, "Antarex – autotuning and adaptivity approach for energy efficient exascale hpc systems," in *2015 IEEE 18th International Conference on Computational Science and Engineering*, Oct 2015, pp. 343–346.

[3] C. Silvano, G. Agosta, S. Cherubin, D. Gadioli, G. Palermo, A. Bartolini, L. Benini, J. Martinovič, M. Palkovič, K. Slaninová *et al.*, "The antarex approach to autotuning and adaptivity for energy efficient hpc systems," in *Proceedings of the ACM International Conference on Computing Frontiers*. ACM, 2016, pp. 288–293.

[4] G. Chrysos, "Intel® Xeon Phi™ Coprocessor-the Architecture," Intel Whitepaper, 2014.

[5] J. M. P. Cardoso, T. Carvalho, J. G. F. Coutinho, W. Luk, R. Nobre, P. Diniz, and Z. Petrov, "LARA: An Aspect-oriented Programming Language for Embedded Systems," in *Proc. 11th Annual Int'l Conf. on Aspect-oriented Software Development*. ACM, 2012, pp. 179–190.

[6] J. M. P. Cardoso, J. G. F. Coutinho, T. Carvalho, P. C. Diniz, Z. Petrov, W. Luk, and F. Gonćalves, "Performance-driven instrumentation and mapping strategies using the LARA aspect-oriented programming approach," *Software: Practice and Experience*, Dec. 2014.

[7] J. Irwin, G. Kickzales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, and J.-M. Loingtier, "Aspect-oriented Programming," in *ECOOP'97 – Object-Oriented Programming*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, vol. 1241, pp. 220–242.

[8] J. M. Cardoso, T. Carvalho, J. G. Coutinho, R. Nobre, R. Nane, P. C. Diniz, Z. Petrov, W. Luk, and K. Bertels, "Controlling a complete hardware synthesis toolchain with LARA aspects," *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 1073–1089, 2013.

[9] R. Nobre, L. G. Martins, and J. M. Cardoso, "Use of Previously Acquired Positioning of Optimizations for Phase Ordering Exploration," in *Proc. of Int'l Workshop on Software and Compilers for Embedded Systems*. ACM, 2015, pp. 58–67.

[10] G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale, "Towards Transparently Tackling Functionality and Performance Issues across Different OpenCL Platforms," in *Int'l Symp. on Comp. and Networking (CANDAR)*, Dec 2014, pp. 130–136.

[11] G. Agosta, A. Barenghi, A. Di Federico, and G. Pelosi, "OpenCL Performance Portability for General-purpose Computation on Graphics Processor Units: an Exploration on Cryptographic Primitives," *Concurrency and Computation: Practice and Experience*, 2014.

[12] F. Bodin, T. Kisuki, P. Knijnenburg, M. O'Boyle, and E. Rohou, "Iterative compilation in a non-linear optimisation space," 1998.

[13] A. Cohen and E. Rohou, "Processor virtualization and split compilation for heterogeneous multicore embedded systems," in *Proc. 47th Design Automation Conference*. ACM, 2010, pp. 102–107.

[14] A. Suresh, B. N. Swamy, E. Rohou, and A. Seznec, "Intercepting functions for memoization: A case study using transcendental functions," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 2, pp. 18:18:1–18:18:23, Jun. 2015.

[15] D. Michie, "Memo functions and machine learning," *Nature*, vol. 218, no. 5136, pp. 19–22, 1968.

[16] E. Paone, D. Gadioli, G. Palermo, V. Zaccaria, and C. Silvano, "Evaluating orthogonality between application auto-tuning and run-time resource management for adaptive opencl applications," in *IEEE 25th Int'l Conf. on Application-Specific Systems, Architectures and Processors, ASAP 2014, Zürich (CH), June 18-20, 2014*,

2014, pp. 161–168.

[17] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano, "Customization of opencl applications for efficient task mapping under heterogeneous platform constraints," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, 2015, pp. 736–741.

[18] D. Gadioli, G. Palermo, and C. Silvano, "Application autotuning to support runtime adaptivity in multicore architectures," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2015, pp. 173–180.

[19] B. Subramaniam, W. Saunders, T. Scogland, and W.-c. Feng, "Trends in Energy-Efficient Computing: A Perspective from the Green500," in *Int'l Green Computing Conf.*, June 2013.

[20] F. Fraternali, A. Bartolini, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Quantifying the Impact of Variability on the Energy Efficiency for a Next-generation Ultra-green Supercomputer," in *Proc. 2014 Int'l Symp. on Low Power Electronics and Design*. ACM, 2014, pp. 295–298.

[21] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde, "A Case Study of Energy Aware Scheduling on SuperMUC," in *Supercomputing*. Springer, 2014, vol. 8488, pp. 394–409.

[22] A. Borghesi, C. Conficoni, M. Lombardi, and A. Bartolini, "MS3: a Mediterranean-Stile Job Scheduler for Supercomputers - do less when it's too hot!" in *2015 Int'l Conf. on High Perf. Comp. & Simulation*. IEEE, 2015.

[23] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 1038–1043.

[24] G. Rudy, M. M. Khan, M. Hall, C. Chen, and J. Chame, "A programming language interface to describe transformations and code generation," in *International Workshop on Languages and Compilers for Parallel Computing*, 2010.

[25] Q. Yi, "POET: A scripting language for applying parameterized source-to-source program transformations," *Software—Practice & Experience*, vol. 42, no. 6, Jun. 2012.

[26] K. S. Namjoshi and N. Singhania, "Loopy: Programmable and formally verified loop transformations," in *International Static Analysis Symposium*. Springer, 2016, pp. 383–402.

[27] A. Qasem, G. Jin, and J. Mellor-crummey, "Improving performance with integrated program transformations," In manuscript, Tech. Rep., 2003.

[28] S. Donadio, J. Brodman, T. Roeder, K. Yotov, D. Barthou, A. Cohen, M. J. Garzarán, D. Padua, and K. Pingali, "A language for the compact representation of multiple program versions," in *Proceedings of the 18th International Conference on Languages and Compilers for Parallel Computing*, ser. LCPC'05. Springer-Verlag, 2006, pp. 136–151.

[29] A. Hartono, B. Norris, and P. Sadayappan, "Annotation-based empirical performance tuning using Orio," in *2009 IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–11.

[30] D. Mustafa and R. Eigenmann, "Portable Section-level Tuning of Compiler Parallelized Applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012.

[31] W. Wang, J. Cavazos, and A. Porterfield, "Energy Autotuning using the Polyhedral Approach," in *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*, S. Rajopadhye and S. Verdoolaege, Eds., 2014.

[32] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, *Auto-tuning for Energy Usage in Scientific Applications*. Springer, 2011, pp. 178–187.

[33] M. Tillmann, T. Karcher, C. Dachsbacher, and W. F. Tichy, "Application-independent autotuning for gpus." in *PARCO*, 2013, pp. 626–635.

[34] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, 2016.

[35] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *ACM SIGPLAN Notices*, vol. 46, no. 3, 2011.

[36] P. G. Kjeldsberg, A. Gocht, M. Gerndt, L. Riha, J. Schuchart, and U. S. Mian, "Readex: Linking two ends of the computing continuum to improve energy-efficiency in dynamic applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 109–114.

[37] A. Hendricks, T. Heller, H. Jordan, P. Thoman, T. Fahringer, and D. Fey, "The allscale runtime interface: Theoretical foundation and concept," in *Proceedings of the 9th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers*, ser. MTAGS '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 13–19. [Online]. Available: https://doi.org/10.1109/MTAGS.2016.4

[38] H. Jordan, P. Thoman, P. Zangerl, T. Heller, and T. Fahringer, "A context-aware primitive for nested recursive parallelism," in *IWMSE Workshop*, ser. Euro-Par 2016, Aug. 2016.