The ANTAREX Approach to Autotuning and Adaptivity for Energy Efficient HPC Systems

Cristina Silvano, Giovanni Agosta, Stefano Cherubin, Davide Gadioli, Gianluca Palermo
DEIB – Politecnico di Milano
name.surname@polimi.it

Andrea Bartolini, Luca Benini
IIS – Eidgenössische Technische Hochschule Zürich
{barandre, lbenini}@iis.ee.ethz.ch

Jan Martinovič, Martin Palkovič, Kateřina Slaninová
IT4Innovations, VSB – Technical University of Ostrava
name.surname@vsb.cz

João Bispo, João M. P. Cardoso, Rui Abreu, Pedro Pinto
FEUP – Universidade do Porto
{jbispo, jmpc, rma, pmsp}@fe.up.pt

Carlo Cavazzoni, Nico Sanna
CINECA
n.surname@cineca.it

Andrea R. Beccari
Dompé Farmaceutici SpA
andrea.beccari@dompe.it

Radim Cmar
Sygic
rcmar@sygic.com

Erven Rohou
INRIA Rennes
erven.rohou@inria.fr

ABSTRACT

The ANTAREX [1] project aims at expressing the application self-adaptivity through a Domain Specific Language (DSL) and to runtime manage and autotune applications for green and heterogeneous High Performance Computing (HPC) systems up to Exascale. The DSL approach allows the definition of energy-efficiency, performance, and adaptivity strategies as well as their enforcement at runtime through application autotuning and resource and power management. We show through a mini-app extracted from one of the project application use cases some initial exploration of application precision tuning by means enabled by the DSL.

Keywords

High Performance Computing, Autotuning, Adaptivity, DSL, Compilers, Energy Efficiency

1. INTRODUCTION

The current roadmap for HPC systems aims at reaching the Exascale level (10^{18} FLOPS) within the 2023–24 timeframe – with a $\times 1000$ improvement over Petascale, reached in 2009, and a $\times 100$ improvement over current systems. Reaching Exascale poses the additional challenge of significantly limiting the energy envelope, while providing massive increases in computational capabilities – the target power envelope for future Exascale system ranges between 20 and 30 MW. To this end, European efforts have recently been focused on building supercomputers out of the less power-hungry ARM cores and GPGPUs [1]. 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. To fulfil the 20MW target, energy-efficient heterogeneous supercomputers need to be coupled with radically new software stacks to exploit the benefits offered by heterogeneity at all levels (supercomputer, job, node).

The ANTAREX [2] project intends to provide 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. The main goal of ANTAREX is to express by means of a DSL the application self-adaptivity and to runtime manage and autotune applications for green heterogeneous HPC systems up to the Exascale level. The use of a DSL allows the introduction of a separation of concerns, where self-adaptivity and energy efficient strategies are specified separately from the application functionalities. The new DSL, inspired by aspect-oriented programmin concepts, will express at compile time the adaptivity/energy/performance strategies and enforce at runtime application autotuning and resource 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, while reducing the Total Cost of Ownership (TCO) for companies and public organizations.

[1]ANTAREX is supported by the EU H2020 FET-HPC program under grant 671623

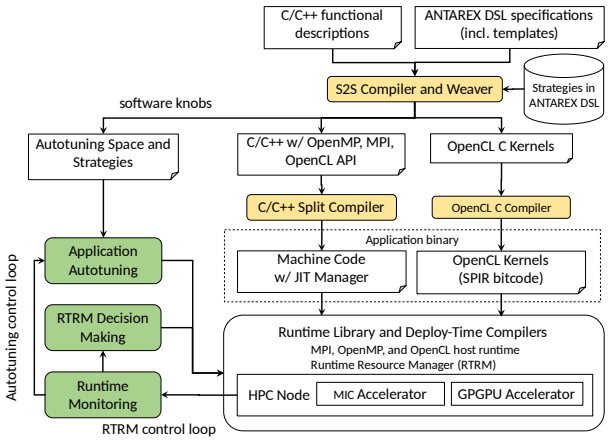

Figure 1: The ANTAREX Tool Flow

The ANTAREX project is driven by two use cases taken from highly relevant HPC application scenarios: (1) a biopharmaceutical application for drug discovery deployed on the 1.21 PetaFlops heterogeneous NeXtScale Intel-based IBM system at CINECA and (2) a self-adaptive navigation system for smart cities deployed on the server-side on the 1.46 PetaFlops heterogeneous Intel® Xeon Phi™ based system provided by IT4Innovations National Supercomputing Center. All the key ANTAREX innovations will be designed and engineered since the beginning to be scaled-up to the Exascale level. Performance metrics extracted from the two use cases will be modelled to extrapolate these results towards Exascale systems expected by the end of 2023

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 the Tier-1 Czech National Supercomputing Center (IT4Innovations) and two industrial application providers, one of the leading biopharmaceutical companies in Europe (Dompé) and the top European navigation software company (Sygic). The complementarity and deep expertise of the Consortium partners is expected to generate a breakthrough innovation from he ANTAREX project. Moreover, the presence of leading edge industrial partners will ensure a relevant impact and direct exploitation paths of ANTAREX results to industry and society. Politecnico di Milano, the largest Technical University in Italy, plays the role of Project Coordinator.

The ANTAREX approach and related tool flow, shown in Figure 1, operate 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 parallelisation, mapping, and adaptivity strategies are expressed through the DSL developed in the project. One of the benefits consists of facilitating the reuse of legacy code. In the definition of these strategies, the application developer or system integrator can leverage DSL templates that encapsulate specific mechanisms, including how to generate code for OpenCL or OpenMP parallelisation, and how to interact with the runtime resource manager. The DSL weaver and refactoring tool will 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 [3]). 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. The application autotuning is delayed to the runtime phase, where the software knobs (application parameters, code transformations and code variants) are configured according to the runtime information coming from the execution environment. Finally the runtime resource and power manager are used to control the resource usage for the underlying computing infrastructure given the changing conditions. At runtime, the application control code, thanks to the design-time phase, now contains also runtime monitoring and adaptivity strategies code derived from 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 will be supported by a runtime layer implementing an application level collect-analyse-decide-act loop.

Organization of the paper The rest of this paper is organized as follows. In Section 2 we present the main ANTAREX DSL concepts, with an example of DSL usage for precision tuning. In Section 3, we discuss the autotuning features, in particular for what concerns the tuning of computation precision and application parameters. In Section 4, we introduce the ANTAREX use cases and target platforms, while in Section 5 we provide an initial assessment of some parameters of the compilation flow, including precision, on a miniapp extracted from one of the ANTAREX use cases. Finally, in Section 6 we draw some conclusions.

2. THE ANTAREX DSL

HPC applications might profit by 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).

A simplistic approach to both adaptation specification and implementation (see, e.g., [4]) employs hard coding of, e.g., conditional expressions and parameterizations. In our approach, the specification of runtime adaptability strategies relies on a DSL implementing key concepts from Aspect-Oriented Programming (AOP) [5].

Our approach is based on the idea that certain application/system requirements (e.g., target-dependent optimizations, adaptivity behavior and concerns) should be specified separately from the source code that defines the functionality of the program. Those requirements are expressed as DSL aspects that embody strategies. An extra compilation step, performed by a *weaver*, merges the original source code and the aspects into the intended program [6]. Using aspects to separate concerns from the core objective of the program can result in cleaner programs and increased productivity (e.g., higher reusability of strategies).

As the development process of HPC applications typically involves two types of experts (application-domain experts and HPC system architects) that can split their responsibilities along the boundary of functional description and extra-functional aspects, our DSL-aided toolflow provides a suitable approach for dealing and helping to express their concerns.

2.1 LARA for Runtime Adaptivity and Compiler Optimization

LARA [7, 8] is an AOP approach to allow developers to capture non-functional requirements and concerns in the form of strategies,

which are decoupled from the functional description of the application. Compared to other approaches that usually focus on code injection (e.g., [9] and [10]), LARA provides access to other types of actions, e.g., code refactoring, compiler optimizations, and inclusion of additional information. All of which can guide compilers to generate more efficient implementations.

Additional types of actions may be defined in the language specification and associated weaver, such as software/hardware partitioning [11] or compiler optimization sequences [12].

We show herein illustrative examples of some of the strategies that can be specified using LARA in the context of a source-to-source compiler and currently used for one of the use cases. Figure 2 presents part of a LARA strategy that changes all declarations of a certain type to a target type (e.g., from double to float) for a given function and for all the functions accessed by calls from this function. The aspect recursively traverses the calls and considers the changes of types for functions with source code available. We note, however, that a practical and reusable aspect needs to deal with further issues, such as the cloning of functions whose types we want to change but are also called by other unrelated functions in the code, assignments of constants, casts, changing library functions to the ones related to the type used (e.g., `sqrtf` vs `sqrt` in Math.h), etc.

```
aspectdef ChangePrecision

  input funcName, oldType, newType end

  /* change type of variable declarations found
   *  inside the function */
  select func{funcName}.decl end
  apply
    def native_type = newType;
  end
  condition
    $decl.native_type == oldType
  end

  /* do the same with the function parameters ... */
  /* do the same with the function return type ... */

  /* recursively, do the same on functions that are
   * called inside this function */
  select func{funcName}.fCall end
  apply
    call ChangePrecision($fCall.name, oldType, newType);
  end
end
```

Figure 2: Example of LARA aspect to change the types of variables declared inside a given function.

A LARA aspect consists of three main steps. Firstly, one captures the points of interest in the code using a `select` statement, which in this example selects variable declarations. Then, using the `apply` statement, one acts over the selected program points. In this case, it will define the type of the variable, using the `native_type` attribute . We can then specify a `condition` statement to constrain the execution of the `apply` (i.e., only if the declared variable had a specific previous type). LARA promotes modularity and aspect reuse, and supports embedding JavaScript code, to specify more sophisticated strategies.

One of the strategies supported in the ANTAREX toolflow is the capability to generate versions of a function and to select the one that satisfies certain requirements at runtime. Figure 3 shows an aspect that clones a function *foo* into a function *foobar* if *bar* is specified as the `suffix`. Note that this strategy imports and uses the previously defined aspect, `ChangePrecision`.

```
import ChangePrecision;

aspectdef CreateFloatVersion

  input funcName, suffix end

  /* clone the target functions and the child calls */
  call cloned : CloneFunctions(funcName, '', suffix);

  /* change the precision of the cloned functions */
  for (var newFunc of cloned.newFunctions)
    call ChangePrecision(newFunc, 'double', 'float');

end

aspectdef CloneFunctions

  input funcName, callerFunction, suffix end
  output newFunctions = [] end

  var newName = funcName + suffix;

  /* clone the target function */
  select func{funcName} end
  apply
    exec Clone(newName);
    newFunctions.push(newName);
  end

  /* recursively, do the same on functions that are
   * called inside this function ... */

  /* change function calls to the cloned function */
  select func{callerFunction}.fCall{funcName} end
  apply
    def name = newName;
  end
end
```

Figure 3: Example of LARA aspect to clone an existing function and change its type.

The aspect `Main`, in Figure 4, adapts the source code of the application in order to have the possibility to call the original version of the function `SumOfInternalDistancies`, or the cloned version, according to the value of a parameter given by the autotuner at runtime. The main aspect calls the previously shown aspect, `CreateFloatVersion`, which clones the target function and every other function it uses, while also changing their variable types from `double` ot `float` (using the aspects presented in Figure 3 and Figure 2).

The aspect also includes the insertion of code for timing the execution of the function and calls to a possible autotuner. The code of the application is modified in order to measure the timing and to communicate the information needed for the autotunner. Then the autotunner communicates back the option defining which function is going to execute, `CreateFloatVersion` or `CreateFloatVersion_float`.

An excerpt of the resulting C code can bee seen in Figure 5.

One important feature of the LARA-aided source-to-source compiler proposed in ANTAREX is the capability to refactor the code of the application in order to expose adaptivity behavior and/or adaptivity design points that can be explored by the ANTAREX autotuning component.

2.2 ANTAREX DSL Concepts

The current LARA infrastructure represents a solid foundation to build a more sophisticated DSL that will enable us to specify runtime adaptability strategies.

The ANTAREX DSL approach aims at reaching a higher ab-

```
import CreateFloatVersion;

aspectdef Main

  input
    funcName = 'SumOfInternalDistancies',
    suffix = '_f'
  end

  /* create the new (float) version */
  call CreateFloatVersion(funcName, suffix);

  /* ... */
  /* create the monitors */
  call monitorOld : TimerMonitor(timerOld);
  call monitorNew : TimerMonitor(timerNew);

  /* ... */
  /* add the code to switch between versions */
  select file.fCall{funcName} end
  apply
    insert before '/*';
    insert after '*/';
    insert after %{
    /* the new code */
    switch(version) {
    case 0:
      [[monitorOld.start]]
      internal = [[funcName]](atoms, 1000);
      [[monitorOld.stop]]
      version = call_autotuner("[[funcName]]",
        [[monitorOld.get]]);
      break;
    case 1:
      [[monitorNew.start]]
      internal = [[newName]](atoms_f, 1000);
      [[monitorNew.stop]]
      version = call_autotuner("[[funcName]]",
        [[monitorNew.get]]);
      break;
    }
    }%;
  end
end
```

Figure 4: Example of LARA aspect which inserts code to allow the change between two versions of the same function.

```
/* the new code */
switch( version ) {
case 0:
    timer_start( tim_old );
    internal = SumOfInternalDistancies(atoms, 1000);
    timer_stop( tim_old );
    version = call_autotuner("SumOfInternalDistancies",
                             timer_get_time( tim_old ));
    break;
case 1:
    timer_start( tim_new );
    internal = SumOfInternalDistancies_f(atoms_f, 1000);
    timer_stop( tim_new );
    version = call_autotuner("SumOfInternalDistancies",
                             timer_get_time( tim_new ));
    break;
}
```

Figure 5: Excerpt of the resulting C code.

straction 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 [13, 14]) is left entirely to the programmer. This is done by exploiting the capabilities of the DSL to automatically explore the configuration space for the parallel code.

To this end, iterative compilation [15] 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 [16]. 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.

3. SELF-ADAPTIVITY & AUTOTUNING

The management of system adaptivity and autotuning is a key issue in HPC systems, 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 [17, 18]. 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 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. 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 parameters and ii) to apply the concept of precision autotuning to HPC applications.

Application Autotuning. Two types of approaches have been investigated so far to support application autotuning depending on the level of knowledge about the target domain: white-boxes and black-boxes. White-box techniques are those approaches based on autotuning libraries that deeply use the domain specific knowledge to fast surf the parameter space. On the other side, black-box techniques do not require any knowledge on the underlying application, but suffer of long convergence time and less custom possibilities. The proposed framework falls in the area of grey-box approaches. Starting from the idea of non-domain knowledge, it can rely on code annotations to shrink the search space by focusing the autotuner on a certain subspace. Moreover, the framework includes an application monitoring loop to trigger the application adaptation. 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 on-line learning techniques are adopted to update the knowledge from the data collected by the monitors, giving the possi-

bility to autotune the system according to the most recent operating conditions. Machine learning techniques are also adopted by the decision-making engine to support autotuning by predicting the most promising set of parameter settings.

Precision Autotuning. In recent years, customized precision has emerged as a promising approach to achieve power/performance trade-offs when an application can tolerate some loss of quality. In ANTAREX, the benefits of customized precision HPC applications will be investigated in tight collaboration with the domain experts of the two use cases. We also plan to apply fully automatic dynamic optimizations, based on profiling information, and data acquired at runtime, e.g. dynamic range of function parameters.

In both cases, the use of the ANTAREX DSL will be crucial to decouple the functional specification of the application from the definition of software knobs (such as code variants or application parameters) and from the precision tuning phase.

4. 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.

Use Case 1: Computer Accelerated Drug Discovery Computational discovery of new drugs is a compute-intensive task that 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. Moreover, different tasks might be more efficient on different type of processors, especially in a heterogeneous system. Dynamic load balancing and task placement are critical for the efficient solution of such problems [19, 20].

Use Case 2: 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. The efficient operation of such a system depends strongly on balancing data collection, big data analysis and extreme computational power [21, 22].

Prototypes of these two use cases will be developed, integrated and validated in relevant environments to practically assess the benefits of the ANTAREX self-adaptive holistic approach, as well as the scalability of the proposed approach towards Exascale systems.

Target Platforms The target platforms are the CINECA's Tier-1 IBM NeXtScale hybrid Linux cluster, based on Intel TrueScale interconnect as well as Xeon Haswell processors and MIC accelerators [3], 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.

5. EARLY EVALUATION

To provide an early analysis of the combined impact of parallelisation, precision tuning, compiler optimizations, we used a miniapp extracted from the Drug Discovery application. The miniapp implements one of the most time consuming computational kernel of the LiGen de-novo drug design software [20]. In particular the miniapp implements the computation of the internal inter-atomic distances of a drug molecule (*SumOfIntervalDistance*) and the computation of the overlap between the drug molecule and a set of protein active site probes (*MeasureOverlap*). Both computations are heavily used to compute a geometrical component of drug docking score. The miniapp does not introduce any change with respect to the LiGen code, preserving the same source and class hierarchy, whereas all the proprietary components and those not relevant for performances have been stripped out.

Table 1: Design Space characterization

Parameter	Values
OpenMP Threads	none, 2, 4, 8, 16
Precision	__float128, double, float, int
Optimization	-O0, -O1, -O2, -O3, -O

We employ test configurations with 4000 atoms per molecule and 400 ligands. We applied parallelisation via a DSL template which instantiate on the outermost loop of each kernel an OpenMP parallel for directive with reduction. We applied precision tuning, allowing the computation to be run in any of a range of floating point types, as well as using integers. Finally, we applied all high-level optimization flags (*-Ox*) exposed by GCC. The overall Design Space for the combined set of optimizations is reported in Table 1. We then performed a full search of the Design Space on three different target machines, a quad-core Intel i5, a 4x quad-core AMD NUMA, and a 2x octa-core Intel Xeon with hyperthreading. For each hardware platform and each kernel, we consider as the baseline the most effective optimization level using maximum precision and 16 OpenMP threads.

In Table 2, we report for each kernel and hardware platform the combination of parameters that provides the best speedup over the baseline, while limiting error to less than 5%. Timings are averaged over 30 computations with the same input data and configuration. For the *MeasureOverlap* kernel, the overhead of parallel execution exceeds its benefits on the Xeon server. While integer computation does not provide sufficient precision, single and double precision floating point arithmetics are generally a good match to the data (originally stored as double precision floating point) both for speed and precision. It is worth noting that the parallel reduction is inherently more precise than the sequential version, so __float128, which provides more precision for the sequential implementation of *SumOfIntervalDistance* is never actually needed in the parallel codes. Finally, as expected the more aggressive optimization levels often fail at matching the features of the target architecture, which points to the need for finer control of the compiler transformations.

6. CONCLUSIONS

Exascale HPC systems will need the definition of new software stacks to fully exploit heterogeneity, while meeting power efficiency requirements. The goal of ANTAREX 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. The impact and benefits of such technology are far reaching, beyond traditional HPC domains.

In this paper, we have shown how the ANTAREX DSL will enable the programmer to control the precision of a computation, as well as to enable parallelisation by means of OpenMP threads. An initial exploration of the resulting Design Space shows that these

Table 2: Exploration of the compiler parameters for the UC1 MiniApp. Speedups are computed over the fastest version with maximum parallelism (16) and precision (__float128). A constraint limiting the maximum error to 5% was imposed.

Kernel	Hardware	OpenMP Threads	Precision	Optimization	Speedup	Error
MeasureOverlap	4-core Intel i5-2500 3.30GHz	16	double	-Os	6.1	0.0%
SumOfIntervalDistance	4-core Intel i5-2500 3.30GHz	4	double	-Os	4.0	0.0%
MeasureOverlap	4x4-core AMD Opteron 8378 2.40GHz	16	float	-O3	2.7	0.0%
SumOfIntervalDistance	4x4-core AMD Opteron 8378 2.40GHz	16	double	-Os	2.7	0.0%
MeasureOverlap	2x8-core Intel Xeon E5-2630v3 2.40GHz	none	double	-O2	2.4	0.0%
SumOfIntervalDistance	2x8-core Intel Xeon E5-2630v3 2.40GHz	16	float	-Ofast	3.6	0.00065%

techniques can significantly improve the execution time (and therefore the cost) of the computation.

References

[1] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" in *Proc. Int'l Conf. on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, pp. 40:1–40:12.

[2] C. Silvano, G. Agosta, A. Bartolini, A. R. Beccari, L. Benini, J. Bispo, R. Cmar, J. M. P. Cardoso, C. Cavazzoni, J. Martinovič, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, and K. Slaninová, "AutoTuning and Adaptivity appRoach for Energy efficient eXascale HPC systems: the ANTAREX Approach," in *Design, Automation, and Test in Europe*, ser. Design, Automation, and Test in Europe, Dresden, Germany, Mar. 2016.

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

[4] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Softw.*, vol. 23, no. 2, pp. 62–70, Mar. 2006.

[5] 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.

[6] T. Elrad, R. E. Filman, and A. Bader, "Aspect-oriented Programming: Introduction," *Communications of the ACM*, vol. 44, no. 10, pp. 29–32, 2001.

[7] 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.

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

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," in *ECOOP 2001 – Object-Oriented Programming*, 2001, pp. 327–354.

[10] O. Spinczyk, A. Gal, and W. Schröder-Preikschat, "AspectC++: An Aspect-oriented Extension to the C++ Programming Language," in *Proc. 40th Int'l Conf on Tools Pacific: Objects for Internet, Mobile and Embedded Applications*, 2002, pp. 53–60.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] C. Beato, A. R. Beccari, C. Cavazzoni, S. Lorenzi, and G. Costantino, "Use of Experimental Design To Optimize Docking Performance: The Case of LiGenDock, the Docking Module of Ligen, a New De Novo Design Program," *J. Chem. Inf. Model.*, vol. 53, no. 6, pp. 1503–1517, 2013.

[20] A. R. Beccari, C. Cavazzoni, C. Beato, and G. Costantino, "LiGen: A High Performance Workflow for Chemistry Driven de Novo Design," *J. Chem. Inf. Model.*, vol. 53, no. 6, pp. 1518–1527, 2013.

[21] R. Tomis, J. Martinovič, K. Slaninová, L. Rapant, and I. Vondrák, "Time-dependent route planning for the highways in the czech republic," in *Proc. Int'l Conf. on Computer Information Systems and Industrial Management, CISIM 2015*, 2015.

[22] D. Fedorčák, T. Kocyan, M. Hájek, D. Szturcová, and J. Martinovič, "viaRODOS: Monitoring and Visualisation of Current Traffic Situation on Highways," in *Computer Information Systems and Industrial Management*. Springer, 2014, vol. 8838, pp. 290–300.