H2020-FETHPC-1-2014 ANTAREX-671623

AutoTuning and Adaptivity approach for Energy efficient eXascale HPC systems

http://www.antarex-project.eu/

Deliverable D1.1: Requirements

Horizon 2020
European Union funding
for Research & Innovation

Deliverable Title:	Requirements		
Lead beneficiary:	CINECA (Italy)		
Keywords:	Functional requirements, extra-functional requirements, use cases		
Author(s):	Giovanni Agosta (POLIMI), Andrea Bartolini (ETHZ) Andrea Beccari (DOMPE), Luca Benini (ETHZ), João Bispo (UPORTO), João Cardoso (UPORTO), Radim Cmar (SYGIC), Carlo Cavazzoni (CINECA), Jan Martinovic (IT4I), Gianluca Palermo (POLIMI), Pedro Pinto (UPORTO), Erven Rohou (INRIA), Nico Sanna (CINECA), Katerina Slaninova (IT4I), Cristina Silvano (POLIMI);		
Reviewer(s):	João Cardoso (UPORTO), Cristina Silvano (POLIMI).		
WP:	WP1	**Task:**	T1.1
Nature:	Report	**Dissemination level:**	Public
Identifier:	D1.1	**Version:**	V1.7
Delivery due date:	November 30th, 2015	**Actual submission date:**	December 18th, 2015

Executive Summary:

This deliverable (issued at M03) presents the results of the activities carried out by the project partners from M01 to M03 in **Task 1.1 "Requirements Analysis"** under the leadership of CINECA. This document collects the requirements of the ANTAREX design flow and of the two use cases according to the concept of separation of concerns between functional requirements (related to the functional behaviour) and extra-functional requirements (such as self-adaptivity, parallelization, energy/thermal features). This deliverable is organized as follows:

- **Section 1** describes the requirements related to the ANTAREX holistic approach, language, APIs and tools;
- **Section 2** describes the requirements related to the Use Case 1: Computer Accelerated Drug Discovery System
- **Section 3** describes the requirements related to the Use Case 2: Self-Adaptive navigation System.

Based on the requirements collected in this deliverable, the **Task 1.2** will define the initial specification of ANTAREX holistic approach, language, APIs and tools (**D1.3**), while the **Task 1.3** and **Task 1.4** will define the initial specifications and integration plans for the Use Case 1 (**D1.4**) and for the Use Case 2 (**D1.5**) respectively.

Approved and issued by the Project Coordinator: 	**Date:** Dec. 18th, 2015

Project Coordinator: Prof. Dr. Cristina SILVANO – Politecnico di Milano
e-mail: silvano@elet.polimi.it - **Phone:** +39-02-2399-3692- **Fax:** +39-02-2399-3411

Table of Contents

1 Holistic Approach, Language, APIs and Tools

1.1 Summary

The main goal of the ANTAREX project is to provide a breakthrough approach to map, runtime manage and autotune applications for green and heterogeneous High Performance Computing (HPC) systems up to the Exascale level expected to be reached by 2023. The approach is based on the tool flow shown in **Figure 1.1.** 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 Domain Specific Language (DSL) inspired by aspect-oriented programming concepts for heterogeneous systems. The new DSL is introduced for expressing adaptivity/energy/performance strategies and to enforce at runtime application autotuning and resource and power management. The goal is to support the parallelism, scalability and adaptability 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.

1.2 ANTAREX Holistic Approach

In the ANTAREX project, we will *develop ground-breaking techniques to express and manage runtime adaptivity strategies in the context of HPC Systems.* The primary goals are to express self-adaptivity, to guide compilers in mapping computations to the target systems according to the application and system requirements, as well as to guide the runtime system and orchestrate the flow of information between application and system in a scalable and distributed way. The ANTAREX approach focuses on the concept of *separation of concerns between extra-functional (self-adaptivity, parallelisation, energy/thermal management) strategies and application functionality.* This will be enabled by the design of a Domain Specific Language (DSL). To make the implementation feasible, the DSL will be based on extensions to the LARA language focused on the expression of runtime self-adaptivity and extra-functional properties. The ANTAREX DSL will enable developers to code adaptive behaviours and to specify strategies for adaptation, helping tools to map those strategies to the target system. The proposed techniques will provide a new path for expressing runtime adaptivity strategies and will make *an important scientific breakthrough over current state-of-the-art approaches for HPC applications,* which are based on a mix of libraries, pragmas, and custom annotations embedded in comments to the application functional code. The ANTAREX DSL aims at fully decoupling the functional code from the extra-functional specifications, as well as providing a homogeneous way to express all the necessary extra-functional aspects – through an inherently extensible DSL.

The ability to react promptly to changes in the workload, unpredictable before runtime, is critical to efficiently manage HPC systems, especially at extreme scales where the cost of an incorrect mapping can be huge. Variability in performance may be caused by several factors, including thermal issues, network congestion, and evolution of the data set during the computation or as a result of real-time HPC workloads. These workload variations can be effectively managed by taking advantage of the heterogeneous hardware resources and properties of the systems (e.g., cooling system), through appropriate runtime choices.

The main breakthrough consists of the replacement of the multiple languages, libraries, and annotations currently used to enrich the application code, through the definition and assessment of a novel programming model to fully decoupling the application knowledge from the system-awareness needed to automatically support self-adaptivity at runtime. Current methodologies require the system integrator to understand the algorithms, at least to the point where they can be instrumented and mapped to the system. Through ANTAREX, we will provide for the first time an automated adaptivity

support and runtime self-adaptivity at the application level. To this end, the ANTAREX DSL will enable the expression of decision policies while exploiting the system features (such as the heterogeneity) to provide effective adaptation mechanisms. Such mechanisms will include, scaling in a hierarchical way from a single node to the whole system, adaptive resource allocation and mapping as well as the ability to control voltage and frequency scaling to achieve energy and thermal efficiency.

Different operating modes and usage contexts imply different mapping decisions. In HPC, it is all but impossible to explore the huge resulting design space – such a feat itself would require HPC computation capabilities. Thus, ANTAREX relies on runtime analysis of the behaviour of the application (or parts of it) to build a model of the possible operating modes, and on split-compilation, continuous optimisation, multiple code versions and runtime self-adaptability strategies to tune the system for maximum energy efficiency (FLOPS/w). Self-adaptation strategies should be exposed to application developers as a high-level DSL, to promote: (1) the portability of the strategies; (2) the exploration of different strategies and software knobs (control mechanisms/features) at design-time; (3) a simpler process for optimisation and mapping of the behaviour responsible for the runtime adaptation; (4) the introduction of additional behaviours (code variants) at a later time.

The ANTAREX approach and related tool flow operates at design-time and runtime. The ANTAREX tool flow is shown in **Figure 1.1.**

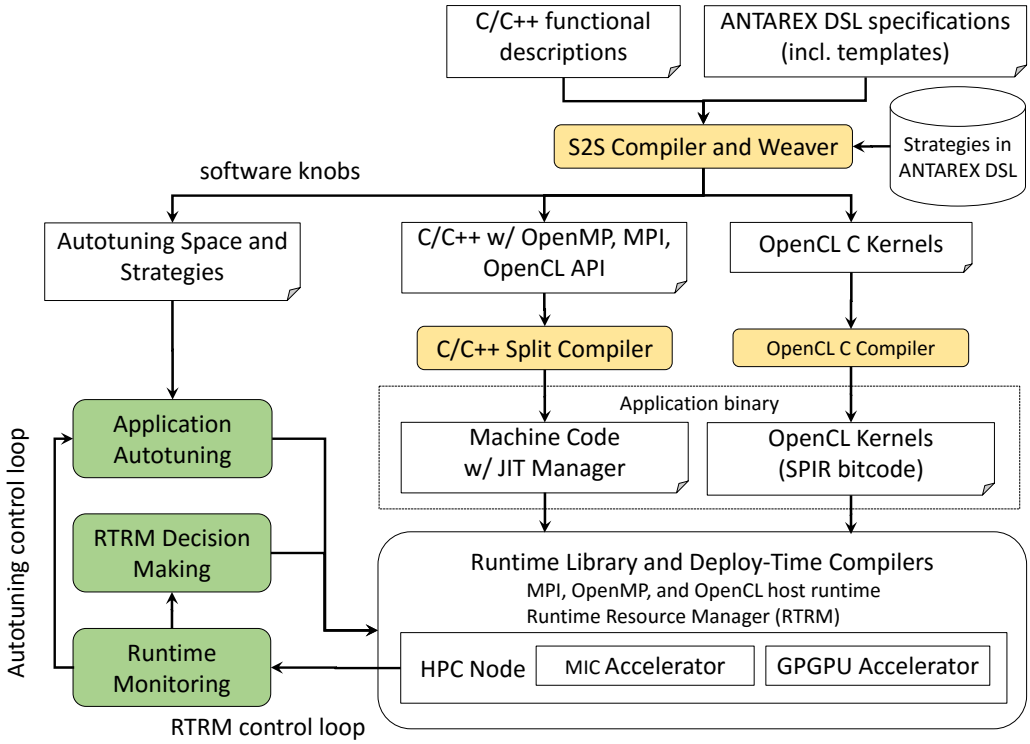

Figure 1.1. The ANTAREX tool flow

The application functionality is expressed through C/C++ code (possibly including legacy code), whereas the non-functional aspects of the application, including adaptivity, parallelisation, and mapping strategies, are expressed through the LARA-based DSL features developed in the project. One of the benefits is to facilitate 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 GPPs and the code for the accelerators (such as GPGPUs). When targeting GPGPU-based accelerators, the code for the accelerators is then translated to OpenCL C, and then deployed using commercial, off-the-shelf compilers. Although the C/C++ code for the GPPs can be compiled using off-the-shelf C compilers, the ANTAREX compilation flow favours the use of a C/C++ split compiler enhanced by specific analysis steps and fully controlled by DSL strategies.

Thus, the ANTAREX compilation flow leverages a runtime phase with compilation steps, by means of split-compilation techniques. The application autotuning is post-poned to runtime, 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 contains now runtime **self-monitoring** and **self-adaptivity** strategies derived from the DSL extra-functional specification and added in the design-time phase. Thus, the application is continuously monitored to guarantee the required Service Level Agreement (SLA), while communication with the runtime resource-manager takes place to regulate the amount of processing resources needed by the application. Application monitoring and autotuning will be supported by the development of a runtime layer implementing a **collect-analyse-decide-act loop** at the application level (as in **Figure 1.2**).

The *collect* phase implemented by the monitoring framework offers the possibility to the application to inspect extra-functional properties (such as energy efficiency) instead of only raw architecture-specific metrics (such as low-level values coming from HW counters). The *analyse* phase will use the data coming from the monitoring phase to build a model considering the application design space (composed of application parameters, code transformations, and code variants) and the high-level metrics (such as performance and energy consumption). The results of the *analysis* phase are used during the *decide* phase for selecting the most suitable combination of application parameters and code transformations (given the allocated platform resources), thus making the application behaviour self-aware. Finally, the *act* phase implements in the application the decisions taken according to the constraints assigned by the resource and power manager.

This complex control would require an in-depth knowledge of the application behaviour, of the target platform features, and of the system-level workload, which is hard or even impossible to achieve by the application developer or system integrator. The ANTAREX approach enables an easier cooperation of these different actors, significantly reducing the cost of the runtime control.

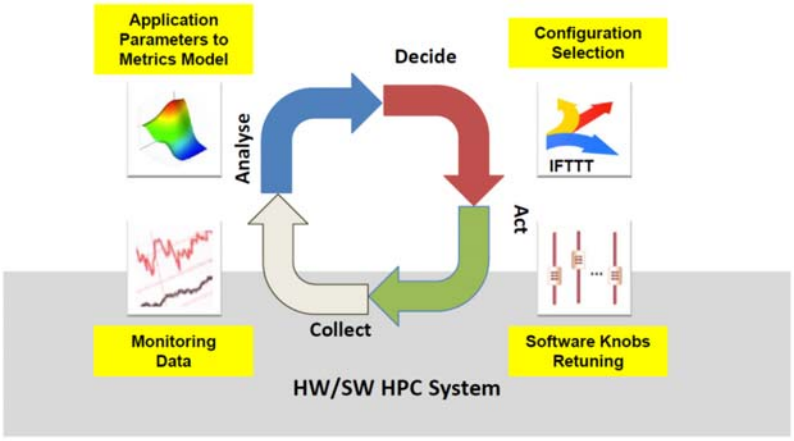

Figure 1.2. The ANTAREX runtime loop

1.3 ANTAREX Tool Flow: Main Components

The ANTAREX tool flow presented in **Figure 1.1** consists of the following five main components described hereafter: the DSL, the Source-to-Source (S2S) Compiler and Weaver, the C/C++ Split-Compiler, the OpenCL compiler and the Runtime Autotuning/Adaptivity Framework.

1.3.1 The Domain Specific Language

The ANTAREX DSL will enable developers to code adaptive behaviours and to specify strategies for adaptation, helping tools to map those strategies to the target system. The proposed techniques will provide a new path for expressing runtime adaptivity strategies and will make an important scientific breakthrough over current state-of-the-art approaches for HPC applications, which are based on a mix of libraries, pragmas, and custom annotations embedded in comments to the application functional code. The ANTAREX DSL aims at a full decoupling of the functional code from the extra-functional specifications, as well as providing a homogeneous way to express all the necessary extra-functional aspects – through an inherently extensible DSL.

This component includes the DSL compiler and a weaver to merge/integrate the strategies specified in the strategies in the target application. The goal here is to propose a powerful DSL to support adaptivity and specialization for HPC applications. Specifically, the intention is to extend the LARA DSL and respective weaver with a powerful mechanism to program strategies for both code refactoring and transformations, and the runtime adaptivity needed by the target applications. The DSL strategies will allow developers to evaluate different multicore/multimode task mapping strategies and to consider application- and architecture-specific strategies.

1.3.2 The S2S (Source to Source) Compiler and Weaver

This component is responsible to the code transformations/refactoring/weaving and the generation of the resultant C/C++ code for the following stages of the tool flow. This stage will be also responsible for generating OpenCL code from a subset of C/C++. The S2S component will be fully controlled by the DSL strategies in order to provide the required adaptivity strategies, code optimizations, code generation (including OpenMP primitives as a result of the DSL parallelisation strategies), and partitioning considering the existence of accelerators. Based on the DSL strategies, the S2S Compiler may also generate multiple versions of the same code (with different optimizations and/or specializations) to be managed at runtime by the autotuning component.

The proposed approach based on the ANTAREX DSL will enable the generation of different versions of the same application code parallelised using different underlying programming models (MPI,

OpenMP, and OpenCL) as parallelisation mechanisms. Then, the DSL will enable the encoding of parallelisation strategies, which will select at runtime the specific target platform, and execute the corresponding code. Further deployment-time optimisation will be enabled through split-compilation.

1.3.3 The C/C++ Split Compiler

At this stage, the intention is to provide optimization levels beyond the usual heuristics tuned for average use cases. By relaxing compilation time, compilers can explore more options for each particular fragment of code, in particular to exploit resources and parallelism (DLP, ILP, thread-level) as much as possible by fine-tuning the generated code for specific data-path properties of specific architectures. In this context, we will use iterative compilation techniques for exploring phase ordering of compiler optimization passes in order to provide the agility in finding the optimal compiler configuration for a given performance metric. This exploration will consider two distinct stages: one at compile time and the other at runtime. The split compiler will provide multiple versions of the same function by considering the possible multiple versions generated by the S2S compiler and the additional code generation version provided by different compiler optimizations (sequences of compiler optimizations). These multiple versions will be used by the autotuning component to achieve better performance/energy trade-offs. We also envision the possibility of exploring phase ordering at runtime (possibly by executing at runtime the compilers responsible for generating target machine code).

Leveraging the multi-stage compilation (such as source-to-source and source-to-native, possibly using code templates, followed by native-to-native), this component splits the compilation process in two steps – offline, and online –to offload as much of the complexity as possible to the offline step, conveying the result to runtime optimizers:

- **Offline:** This step occurs on the developer's workstation. Resources are virtually unlimited. Ideally, all the work should have been done offline. However, target and execution environment are unknown. Target-dependent optimizations are impossible – or risky at the best. Dynamic events are also unknown (program inputs, competing processes on a server), causing missed opportunities.
- **Online:** This step occurs on the user's device, at runtime. In case of an embedded system, resources are much more limited than in the offline step. However, the actual target system (hardware and computing environment) is known. Many classical optimizations can be revisited in this context, pushing as much of the work as possible to the offline step. Several other scenarios are also possible, beyond mere splitting of optimizations. When relevant, optimizations can also be partially or speculatively applied offline, at compile-time, as has been shown in the case of vectorization.

In ANTAREX, we will combine iterative- and split-compilation and extend their applicability. In the context of HPC applications and changing computing environments, iterative compilation is suitable to identify interesting trade-offs. However, it will be practically impossible to generate all potential relevant versions of code due to combinatorial explosion and code bloat. Instead, we will combine iterative-compilation with split-compilation: instead of generating code, we will encode code generation strategies (“recipes”) to drive a dynamic code generator in response to particular hardware features (such as number of registers or degree off available instruction level parallelism) as well as dynamic events (cache misses, branch misprediction). Contributions will consist in identifying the relevant characteristics and the proper dynamic events, how to express them in the ANTAREX DSL in a way manageable by the programmers, and finally in defining the appropriate response to these events. Finally, split-compilation will have a significant impact on the performance of applications, but also on the productivity of programmers. Split compilation will relieve programmers from the burden of repeatedly optimizing, tuning, compiling and testing: much of the effort can be automated, including in cases where diverse and heterogeneous targets are envisioned.

1.3.4 The OpenCL Compiler

The generation of OpenCL will provide the ability to manage effectively heterogeneous accelerators and to express simple but powerful parallelism, while mitigating its inefficiencies as a high-productivity language by employing it as an intermediate language to deploy kernels to different platforms. The development of the application hot spots will be performed in C (with limitations similar to those of OpenCL C) within the host C/C++ code, and the parallelisation strategy will be encoded in a DSL strategy, which will encode suitable optimisation strategies for different target platforms. Boilerplate code will then be automatically instantiated from DSL templates. The DSL modelling environment will thus offer device independence and transparency to the developer. This approach, not addressed in the past to the best of our knowledge (each OpenCL application is written with respect to a specific implementation architecture), will further enhance OpenCL efficiency. The OpenCL compiler stage envisioned here is responsible for optimizing the OpenCL code according to the target architecture in order to contribute to performance portability.

1.3.5 The Runtime Autotuning/Adaptivity Framework

The ANTAREX ambitions in the field of runtime methodologies for HPC systems are related to the management of system adaptivity. This is a key issue in HPC systems where the system needs to react promptly to changing workloads and events, without affecting too much its extra-functional characteristics, such as energy and thermal features.

ANTAREX will solve these challenging problems by following a disruptive holistic approach spanning all the decision layers composing the supercomputer software stack. The ANTAREX approach is based on:

1. Expanding the energy/performance control capabilities by introducing novel software control knobs (i.e. 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 (job dispatching, resource management and Dynamic Voltage and Frequency Scaling) at different time scale (compile time, deployment time and runtime);
3. Monitoring the HW supercomputing evolution as well as the application status and requirements and bringing this information to the energy/performance-aware software stack.

This approach will allow to always operate the supercomputer and each application at the best energy-efficient and thermally-safe point.

Application Autotuning framework

This ANTAREX Autotuning/Adaptivity infrastructure is responsible for tuning the application at runtime considering target power/energy/performance/QoS/SLA envelops. It will consider the tuning of software knobs such as application parameters, multiple version of the same function, code transformations, and customized precision options. We envision that this component starts with the user-defined autotuning and dynamic adaptivity strategies and is capable to adjust/extend strategies at runtime based on the workloads, system properties, and power/energy measurements.

Our goal is to provide a scalable framework to support the autotuning for HPC applications. The proposed framework will fall in the area of grey-box approaches since it starts from non-domain knowledge; it can be fed by code annotations to shrink the search space by focusing the autotuner on a certain subspace. The framework will include a monitoring loop that will be used to monitor the application and to trigger the application adaptation. The monitoring, together with application properties/features, will represent the main support to the decision-making during the application autotuning phase, since it will be used to conduct statistical analysis related to system performance and other SLA aspects. Continuous online learning techniques will be adopted to update the

knowledge from the data collected by the monitors, enabling the possibility to autotune the system always according to the more recent operating conditions. Machine learning techniques, such as neural networks and decision trees, will be adopted in ANTAREX for the decision-making engine to support the autotuning by predicting the most promising set of parameter settings.

The ANTAREX-DSL will be used to express code variants (i.e., different implementations of the same computation), application parameters, in terms of discrete set of values to govern code generation and execution of a single code variant, and architecture-agnostic programmer hint to describe the program behaviour. Thus, the ANTAREX approach will export the application knobs out of the box for being managed by the application autotuning framework.

Runtime Resource Management Framework

Today's commercial job dispatchers do not exploit application specific features for power management as they are lacking the support for directly inserting in the job-to-resource mapping energy and performance metrics as well as physical system information and constraints. ANTAREX will overcome this limitation by bringing in today job-dispatcher the information flow of the energy/thermal-efficiency and resource dependability concept. ANTAREX will make the job-dispatcher aware of the underling hardware.

To pursue this goal, a hierarchical solution is proposed where (i) the job-dispatcher will coordinate multiple job-level RTRMs to keep under control the distributed resource allocation, (ii) a job-level RTRM will cope with goals requiring control at different time granularity, (iii) a node level thermal controller will select the operating conditions exploiting predictive models.

In addition, model learning strategies will be developed based on the developed runtime monitoring infrastructure and optimisation model for the job scheduler. The result will bias the allocation of the jobs toward the available more efficient nodes, with an interaction between the hardware monitoring, the job-dispatcher, the job-level RTRMs and the applications to optimize the power and performance. The learning strategies will take full advantages of such tightly coupled interaction, generating reliable predictive models of the application requirements and optimal execution frequencies

In conclusion, in the ANTAREX project, we will deploy a scalable hierarchical energy/thermal management infrastructure capable of finding at the same time the optimal application and supercomputer operating point while ensuring safe working temperature.

Monitoring Framework

Runtime adaptation and decision strategies cannot be easily applied without a proper monitoring infrastructure to handle the huge amount of data coming from a fine grain monitoring of the supercomputer performance counters. The scalability of the monitoring infrastructure is guaranteed by data structures and data flow patterns tailored for big-data. Such data collection will enable accurate model learning, as well as context-aware actions to be taken by the different controllers. The efficient communication big-data knowledge distribution network will enable the higher level to take action on more aggregated data, while the low level to have faster control loop on more local fine grain information as well as to promptly react to application events.

1.4 Key Benefits

In conclusion, the ANTAREX approach, by coupling a system-independent application code specification with advanced runtime self-adaptivity capabilities, offers several **key benefits:**

1. Thanks to a well-structured separation of concerns (strategies vs. application), the runtime management can be fine grained at the application function level, opening new perspectives to cope with extra-functional requirements, such as energy efficiency and heterogeneity, posed by next generation Exascale HPC systems.

2. Functional and extra-functional requirements (e.g., power and thermal) will be no longer intertwined to lead to substantial savings in application development and deployment by **30%** compared to state-of-the-art in 2014.
3. The proposed runtime self-adaptive approach will enhance the reaction time of the system to a change of application workload and system resources. This opens the analysis of a wide spectrum of energy/performance trade-offs. Moreover, we believe this will lead to a significant reduction **(25% up to 40%)** of the energy requirement, while preserving SLA.

In the ANTAREX project, these benefits will be assessed through the application of the proposed approach to two industrial use cases deployed on the state-of-the-art supercomputing infrastructures available at CINECA and IT4I supercomputing centres. Deployment of the two industrial applications to these infrastructures will leverage the autotuning and runtime power management capability of the ANTAREX approach as well as the benefits of the DSL to express extra-functional (self-adaptivity, parallelisation, energy/thermal management) strategies on the side of application functionalities.

The introduction of the ANTAREX DSL responds to an explicit requirement expressed by the CINECA and IT4I supercomputing centres and the end users (DOMPE and SYGIC) to replace pre-existing solutions to express extra-functional features and strategies based on ad hoc hard-to-maintain annotations to the source language. The proposed ANTAREX DSL will be completely transparent to the end user and provide a single entry point to the supercomputing expert.

These use cases show two critical kinds of HPC applications: applications with unpredictable workloads due to data-dependent computational effort, and real-time HPC applications. For each use case, the Consortium includes a partner with specific application domain expertise (DOMPE for the drug discovery application and SYGIC for the navigation application) and a partner managing the HPC infrastructure on which the use case will run (CINECA, a Tier-0 supercomputing centre, and IT4I, a Tier-1 centre)**.**

The following sections present the requirements elicited and includes a description of the use cases.

1.5 Requirements of holistic approach, language, APIs, tools

In this Section, we present the elicited requirements of the ANTAREX project regarding the approach, domain-specific language, APIs and tools. The specific requirements related to the two use cases are presented in **Section 2** and **Section 3.** The requirements presented in this Section are grouped in the following tables:

- Table 1.1 Generic Requirements
- Table 1.2 DSL Requirements
- Table 1.3 DSL Compiler Requirements
- Table 1.4 DSL Weaver Requirements
- Table 1.5 Source to Source Compiler (S2S) Requirements
- Table 1.6 Compiler Middle/Back End Requirements
- Table 1.7 Split Compilation Requirements
- Table 1.8 Power Management Requirements
- Table 1.9 API Requirements
- Table 1.10 Autotuning Requirements
- Table 1.11 Experimental Platform Requirements

Table 1.1: Generic Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
GEN	REQ1	ANTAREX shall use open standards and tools whenever they exist and are suitable.	Must have	Language/Compiler/Tool/API/ Methodology	Khronos OpenCL, LARA, LLVM, C, C++.
GEN	REQ2	The ANTAREX approach shall support logging and tracing of runtime adaptability decisions and operating modes.	Must have	Language/API	D1.3/MS2
GEN	REQ3	The ANTAREX approach shall allow runtime estimation of resource usage/consumption.	Must have	Methodology	D1.3/MS2
GEN	REQ4	The ANTAREX framework should support the performance analysis to evaluate the scalability	Should have	Methodology	D1.3/MS2
GEN	REQ5	The ANTAREX framework should support the projection analysis towards Exascale	Should have	Methodology	D1.3/MS2

Table 1.2: DSL Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
DSL	REQ1	The ANTAREX DSL shall support an aspect-oriented modelling approach.	Must have	Language	D2.1/MS3
DSL	REQ2	The ANTAREX DSL shall be based on a separation of concerns approach.	Must have	Language	D2.1/MS3
DSL	REQ3	The ANTAREX DSL shall support the modelling of functional requirements (e.g. constraints).	Could have	Language	This is not the main focus of ANTAREX D2.1/MS3
DSL	REQ4	The ANTAREX DSL shall support the modelling of extra-functional requirements.	Must have	Language	D2.1/MS3 D2.5/MS5
DSL	REQ5	The ANTAREX DSL shall support the definition of autotuning strategies.	Must have	Language.	D2.1/MS3 D2.4/MS5
DSL	REQ6	The ANTAREX DSL shall be based on an extended version of the LARA language.	Must have	Language.	D2.1/MS3
DSL	REQ7	The DSL shall support the specification of strategies using alternative algorithms with different performance/power/energy/QoS characteristics.	Must have	Language (related to S2S Compiler)	D2.1/MS3 D2.5/MS5
DSL	REQ8	The DSL shall support the expression of different types of parallel paradigms	Should have	Language (related to S2S Compiler)	Parallel programming paradigms such as such as OpenMP, MPI, OpenCL. D2.2/MS4 D2.3/MS4

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
DSL	REQ9	The DSL shall express what is the level of customized accuracy	Must have	Language (related to S2S Compiler)	D2.5/MS5
DSL	REQ10	The DSL shall express the adaptivity conditions to apply runtime changes	Must have	Language/S2S Compiler	Conditions might include thermal conditions, performance of the memory hierarchy, IPC...) D2.5/MS5
DSL	REQ11	The DSL shall allow to specify code locations where an operating mode switch is possible	Must have	Language/S2S Compiler	D2.4/MS5
DSL	REQ12	The DSL shall allow to describe what data must be saved before the operating mode switch	Must have	Language/S2S Compiler	D2.4/MS5
DSL	REQ13	The DSL shall express compilation strategies (e.g., phase ordering) to be applied by the Compiler Back-End	Must have	Language/Compiler back-end	D2.3/MS4
DSL	REQ14	The DSL shall allow the programming of DSE (Design-Space Exploration) strategies to be applied to the ANTAREX toolflow	Must have	Language/Tool/Methodology	D2.3/MS4
DSL	REQ15	The DSL shall allow to express code instrumentation strategies	Must have	Language/S2S Compiler	D2.3/MS4
DSL	REQ16	The DSL shall allow to control code transformations/refactoring	Must have	Language/S2S Compiler	D2.3/MS4
DSL	REQ17	The DSL shall allow to specify static and dynamic “apply” actions.	Must have	Language	D2.1/MS3 D2.4/MS5
DSL	REQ18	The DSL shall be generic to deal with other target languages	Could have	Language	Other target languages besides C/C++ (e.g., Java, MATLAB). D2.1/MS3
DSL	REQ19	The DSL shall be configurable to deal with other target languages.	Could have	Language	Other target languages besides C/C++ (e.g., Java, MATLAB). D2.1/MS3

Table 1.3: DSL Compiler Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
CMP	REQ1	The DSL compiler shall process the DSL code and generate an intermediate representation for the weavers.	Must have	Compiler	D2.2/MS4
CMP	REQ2	The DSL compiler shall generate C/C++ code from dynamic actions expressed in DSL aspects.	Must have	Compiler	The generated C/C++ code will be added to the application code. D2.2/MS4 D2.4/MS5
CMP	REQ3	The DSL compiler shall weave the C/C++ code specified in static “insert” actions expressed in DSL aspects.	Must have	Compiler	The C/C++ code will be added to the application code. D2.2/MS4 D2.4/MS5
CMP	REQ4	The DSL compiler shall be flexible to deal with other target languages.	Could have	Compiler	Other target languages besides C/C++ (e.g., Java, MATLAB). D2.2/MS4 D2.3/MS4

Table 1.4: DSL Weaver Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
WVR	REQ1	The DSL weaver shall process the DSL intermediate representation.	Must have	Compiler/S2S	This requires an interaction to the compiler integrating the weaver. D2.2/MS4
WVR	REQ2	The DSL weaver shall interact with the source-to-source compiler.	Must have	Compiler/S2S	This requires an interaction to the compiler integrating the weaver. D2.2/MS4

Table 1.5: Source-to-Source (S2S) Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
S2S	REQ1	The S2S compiler shall be able to process C/C++ input code and to generate C/C++ code.	Must have	Compiler/S2S	D2.2/MS4
S2S	REQ2	The S2S compiler shall be able to refactor/transform the input C/C++ code.	Must have	Compiler/ S2S	D2.2/MS4
S2S	REQ3	The S2S compiler shall be able to partition the input source code and to include synchronization/communication primitives.	Must have	Compiler/ S2S	D2.2/MS4
S2S	REQ4	The S2S compiler shall be able to instrument the code based on DSL strategies.	Must have	Compiler/ S2S	D2.2/MS4
S2S	REQ5	The S2S compiler shall be able to insert monitoring and adaptability points.	Must have	Compiler/ S2S	D2.2/MS4 D2.4/MS5
S2S	REQ6	The S2S compiler shall be able to perform loop transformations (e.g. loop tiling, loop unrolling, loop interchange, unroll and jam, loop splitting, loop fusion).	Must have	Compiler/ S2S	D2.2/MS4
S2S	REQ7	The S2S compiler shall be able to perform data partitioning.	Should have	Compiler/ S2S	Partitioning data structures such as arrays. D2.2/MS4
S2S	REQ8	The S2S compiler shall be able to generate and integrate multiple versions of the same function.	Must have	Compiler/ S2S	The multiple versions are then orchestrated by strategies or by autotuning schemes. D2.3/MS4
S2S	REQ9	The code refactoring/transformations/partitioning performed by the S2S compiler shall be controlled by the DSL.	Must have	Compiler/ S2S	DSL strategies will control and guide the source to source compiler. D2.2/MS4 D2.3/MS4
S2S	REQ10	The S2S compiler shall be able to generate C/OpenCL code from a target subset of C/C++ code.	Must have	Compiler/ S2S	A translation from C/C++ to OpenCL (guided by the DSL). D2.4/MS4, MS5
S2S	REQ11	The C/OpenCL code generation shall be controlled by the DSL.	Must have	Compiler/ S2S	D2.4/ MS4, MS5

Table 1.6: Compiler Middle/Back End Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
MBE	REQ1	The compiler shall be able to apply various optimizations that adjust the workload to the current hardware, and available resources.	Must have	Language/Compiler Back-End	Set of optimizations will be specified in D1.3 Resources include bandwidth, co-running processes. Transformations include loop tiling, vectorization for various SSE/AVX families.
MBE	REQ2	The compiler shall be able to apply additional optimizations specialized for the OpenCL C language and OpenCL API (e.g., work item coalescing).	Should have	Language/Compiler Back-End	Set of optimizations will be specified in D1.3
MBE	REQ3	The compiler shall be able to apply additional optimizations specialized for individual platforms.	Could have	Language/Compiler Back-End	Set of optimizations will be specified in D1.3. Possible specific optimization include memory transfer optimization
MBE	REQ4	The compiler shall be able to expose the optimizations provided as result of REQ1, to enable external tools to control the ordering and parametrization of the transformations.	Must have	Language/Compiler Back-End	Fine-grained compiler control API/option set will be specified in D1.3.
MBE	REQ5	The compiler should be able to compute as soon as possible in the compilation toolchain useful information for selecting the best target platform for each application kernel.	Must have	Language/Compiler Back-End	Set of computed information will be specified in D1.3. Useful information includes kernel stack size, number of work items per work group, control flow divergence.
MBE	REQ6	The compiler should be able to expose to external tools the information related to REQ5	Must have	Language/Compiler Back-End	API/data formats to access the information will be specified in D1.3.
MBE	REQ7	The compiler should be able to expose to external tools additional information from its internal analysis passes	Should have	Language/Compiler Back-End	API/data formats to access the information will be specified in D1.3. Internal analysis passes include register pressure, ability to exploit specific instruction set extensions, etc.

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
MBE	REQ8	The compiler shall be able to support computations in customized precision (analysis and code generation).	Should have	Language/Compiler Back-End	Set of analyses/ optimizations specified in D1.3.
MBE	REQ9	The compiler shall support the code instrumentation strategies defined by the DSL.	Must have	Compiler Back-End	D2.3/MS4
MBE	REQ10	The compilation flow shall allow the application of automatic DSE strategies for selection of compiler optimizations and phase ordering.	Must have	Compiler	D2.3/MS4

Table 1.7: Split Compilation Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
SLT	REQ1	The compilation flow shall provide hooks to support split-compilation. Hooks include encode and embed a subset of the DSL in the generated executable, enabling dynamic code generation (possibly code templates), and recommended strategies.	Should have	Compiler Back-End/toolchain	D2.3/MS4
SLT	REQ2	The compiler shall support function specialization.	Should have	Compiler Back-End	D2.3/MS4

Table 1.8: Power Management Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
MNG	REQ1	The power manager shall have access to the allocation of the task on the resources and the processing elements operating conditions	Must have	Tool/ Methodology	Needs to interact (have hooks) with the programming model run-time, Cgroups, power governors and OS scheduler. D1.3/MS2 D3.1, MS7
MNG	REQ2	The power manager and monitoring framework need to be compliant with the supercomputer “in production environment”.	Must have	Tool/ Methodology	D1.3/MS2 D3.3, MS9.

Table 1.9: API Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
API	REQ1	The ANTAREX Monitoring framework shall allow access to the HW performance counters with fine granularity and low overhead.	Must have	Tool, Methodology	Time sampling @ seconds and sub-milliseconds on demand Below 1% of overhead. D1.3/MS2 D3.1, MS7
API	REQ2	ANTAREX power manager needs to have predictive model and allow interaction with optimization libraries.	Should have	Tool, Methodology	Predictive control and optimal allocation outperform reactive policies. D1.3/MS2 D3.1, MS7
API	REQ3	ANTAREX Monitoring framework shall allow energy and performance measurement/estimation of the different processing components.	Must have	Tool, Methodology	Energy measures not power (can be obtained with integration of finer granularity power measurement). D1.3/MS2 D3.1, MS7
API	REQ4	The ANTAREX Monitoring framework shall allow integration / correlation of the different node time traces and scalability among the large set of nodes	Must have	Tool, Methodology	Accurate time stamps (sub ms), low overhead and hierarchy. D1.3/MS2 D3.1, MS7
API	REQ5	The ANTAREX Monitoring framework shall allow to interface with the DSL.	Must have	Tool, Methodology	D1.3/MS2 D3.1, MS7
API	REQ6	ANTAREX power manager needs to control the task allocation on the resources and operating conditions of the processing elements.	Must have	Tool, Methodology	Needs to interact (have hooks) with the programming model run-time, Cgroups, power governors and OS scheduler. D1.3/MS2 D3.1, MS7

Table 1.10: Autotuning Requirements

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
ATU	REQ1	The autotuning framework should adapt depending on data coming from the monitoring framework	Must have	Tool/Methodology	The application adaptation should be triggered by changes in the execution scenario identified by the monitors D1.3/MS2 D3.2, MS6
ATU	REQ2	The autotuning framework should manage the software knobs exposed by the application.	Must have	Tool/Methodology	D1.3/MS2 D3.2, MS6
ATU	REQ3	The autotuning framework should have access to the application knobs	Must have	Tool/Methodology/Compiler	The access should be provided by means of the DSL or by the application itself D1.3/MS2 D2.1, MS3
ATU	REQ4	The autotuning framework should autonomously select the optimal application configuration (software knobs)	Must have	Tool/Methodology	The adaptation should be triggered automatically and not by hand D1.3/MS2 D3.2, MS6
ATU	REQ5	The autotuning framework should change the software knobs only considering stateless application conditions or considering conditions where an explicit checkpoint function is coded by means of the DSL	Must have	Tool/Methodology	D1.3/MS2 D3.2, MS6
ATU	REQ6	The monitoring framework should expose information capable to drive the application adaptivity	Must have	Tool/Methodology	Information about the execution needed for triggering the adaptivity D1.3/MS2 D3.1, MS7
ATU	REQ7	The DSL shall provide operating points and mechanisms for the runtime selection of the most appropriate operating point.	Must have	Language/ Compiler/Tool	D1.3/MS2 D2.1, MS3

Table 1.11: Experimental Platform Requirements [1]

		Requirement	Priority	Applicability	Comments
REQ ID	REQ No	Short description of the requirement	Must have/ Should have/ Could have	Language/ Compiler/ Tool/API/ Methodology	Short comments & links to DEL/MS
EXP	REQ1	The experimental platform must support an Intel Xeon Phi board	Must have	Experimental hardware setup	Minimum 1KW Power Supply Unit (PSU) D1.3/MS2
EXP	REQ2	The experimental platform must be endowed with sufficient memory to support double buffering in memory transfer to/from the Xeon Phi accelerator	Should have	Experimental hardware setup	Minimum 64 GB RAM D1.3/MS2
EXP	REQ3	The experimental platform should support a GPGPU accelerator (min 1.5 KW PSU, compatible motherboard)	Should have	Experimental hardware setup	Minimum 1.5 KW PSU D1.3/MS2
EXP	REQ4	The experimental platform should support a measurement system for blade-level energy consumption	Could have	Experimental hardware setup	Hot-swappable PSU D1.3/MS2
EXP	REQ4	The experimental platform must be compatible with both rack-mounted and server/tower form factors	Should have	Experimental hardware setup	Motherboard form factor must be taken into account D1.3/MS2
EXP	REQ5	The experimental platform must be representative of a typical HPC system node	Must have	Experimental hardware setup	At least a dual-socket Xeon must be included D1.3/MS2

[1] In order to support an early experimentation phase, a heterogeneous experimental platform based on Xeon Phi architecture will be adopted by the academic partners before deployment on the HPC systems.

2 USE CASE 1: Computer Accelerated Drug Discovery System

2.1 Summary

A new de novo design program Ligand Generator (LiGen) [1] was developed by CINECA and DOMPE. The program is able to employ a user-defined set of true chemical reactions as connection rules and to use tangible commercial reagents as building blocks. The LiGen suite has been designed to be, at the same time, a modular de novo design application and a computational toolbox.

More in detail, LiGen is a suite of programs to define flexible de novo design workflows by using both pharmacophores and molecular docking engines, linking the fragments in situ, and directly scoring the drug likeness of the generated molecules. Therefore, LiGen consists of a set of tools, which can be combined in a user-defined manner to generate project centric workflows. In a standard application, the modules work sequentially, from the generation of the input constraints (either structure-based, through active site identification, or ligand-based, through pharmacophore definition) to docking and de novo generation. Alternatively, the modules can be used as a standalone service according to the user's need. Figure illustrates the general LiGen workflow. LiGen consists of a set of modules: LiGenPass for binding site recognition, LiGenPocket for binding site analysis and structure-based pharmacophore definition, LiGenDock for docking and virtual screening, and LiGenBuilder for de novo design.

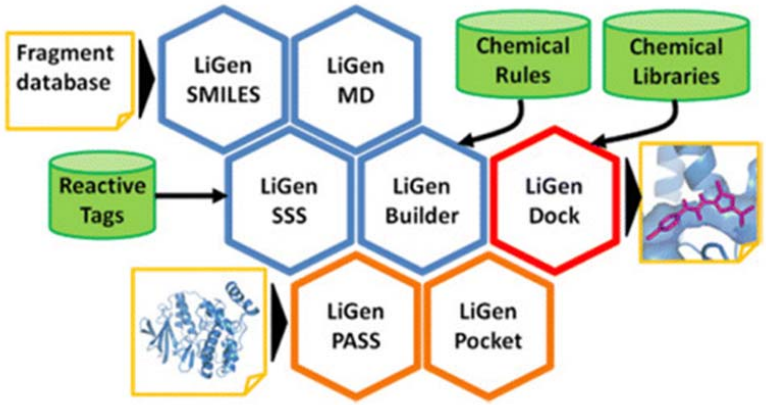

Figure 2.1. The LiGen Workflow

Specific features of LiGen are the use of a pharmacophore-based docking procedure, which allows flexible docking without conformer enumeration and accurate and flexible reactant mapping coupled with reactant tagging through substructure searching. The new tool is suitable for HPC applications and portable to many hardware architectures.

2.2 Motivations

Nowadays several molecular de novo design programs, such as LUDI [2], LEGEND [3], LeapFrog [4], LigBuilder [5, 6] SPROUT [7,8], HOOK [9], PROLIGANDS [10−12], and DOGS [13] have been developed and their advantages and drawbacks [14-17] have been highlighted.

Despite several successful applications have been reported in literature, many problems eventually prevented molecular de novo design approaches to become established tools in drug design, and, in fact, methods like virtual screening and molecular docking received higher attention, either in terms of successful applications or in terms of widespread use. The most common problems associated with these de novo design methods are the following:

- producing chemically invalid structures or structures that do not have drug-like properties;
- poor synthetic accessibility of the suggested ligand
- low structural diversity; [18]
- low potential for parallel synthesis applications (a part when combinatorial chemistry is directly addressed [19]);
- generally low throughput if compared to docking programs.

De facto, none of these de novo design programs are designed to run on HPC architecture.

The problem of synthetic feasibility in particular, is the most relevant one, and its impact on the outcome of the de novo approach is highly dependent on the stage of the project and on the size of the library, which has to be assembled. When the main goal is a primary screening setup, many docking as well as de novo design programs are able to handle combinatorial explosions. Under these conditions, the definition of a set of fragments (reagents) with simple anchor points, and the availability of a predefined set of reaction steps can lead to the generation of targeted libraries of arbitrary size, which can be in principle synthesized in parallel. In this context, the ability of controlling chemistry, in terms of reaction steps and reagent's availability, is crucial to move the designed libraries from virtual to real.

Another common application of de novo design is lead optimization. In this case, the binding mode of the core structure of the lead has usually already been validated and the scope of the de novo approach is to optimize decoration toward increased affinity and/or improved ADME properties. The value of the class under study is therefore high, and the number of compounds to generate is reasonably low. Thus, the selection of the fragments is driven by the optimization process and not by the accessibility of the pre-synthesized reagents.

The LiGen suite has been designed to be, at the same time, a modular de novo design application and a computational tool box. The LiGen modules can be used independently from each other and combined with other tools typically used in drug design research. This is made possible by the fact that modules communicate each other by using a few standard data formats, with a few minor extensions for inter modules communication not breaking the original standard format. Moreover the modular structure of LiGen allows the implementation of different workflows using the basic building block tools, depending on the specific computation or analysis required.

LiGen is written in C++, with the use of STL (Standard Template Library) at a high level, and its own algorithms (functions) and data structures (classes) at a low level. In this way, the code is flexible, easily extensible, and fast on many architectures. Performance portability to different hardware has been a main priority in LiGen design. Using STL at all levels (e.g., basic vector for interatomic distances) may end-up with a code not equally efficient on all hardware or possibly not at all portable (think to GPU computing where CUDA or OpenCL kernels are required, and STL may not be fully supported). All LiGen modules share the same basic C++ classes representing the main biochemical objects-atoms, residues, molecules - or mathematical objects - vector, grid, ring, etc. LiGen modules can be compiled in standalone executables or built together in a single executable implementing a given workflow. The basic modules of LiGen are the following: LiGenPass, LiGenPocket, LiGenSmiles, LiGenMD, LiGenMinimizer, LiGenSSS, LiGen-Dock, LiGenScore, and LiGenBuilder.

2.3 State-of-the-art

Molecular de novo design is aimed at generating novel chemo types endowed with a particular set of desired properties, in most cases pharmacological properties. The de novo design process is ideally driven by knowledge, and this knowledge is used to define constraints, which the novel molecular

structures being generated are supposed to obey. The search for novel chemo types is a local and a multi-objective optimization process, which can lead to several solutions according to the input constraints and to the generation rules chosen by the chemist. On the other hand, finding a globally optimal solution is unworkable given the enormous size of the available chemical space. Therefore, computer programs have been actively sought during the past two decades to help (medicinal) chemists to find the best local solutions in terms of pharmacological properties, chemical feasibility, innovation, and patentability.

When the main goal is a primary screening setup, many docking as well as de novo design programs are able to handle combinatorial explosions. Under these conditions, the definition of a set of fragments (reagents) with simple anchor points, and the availability of a predefined set of reaction steps can lead to the generation of targeted libraries of arbitrary size, which can be in principle synthesized in parallel. In this context, the ability of controlling chemistry, in terms of reaction steps and reagent's availability, is crucial to move the designed libraries from virtual to real. In the field of molecular modelling, docking is a method to predict the preferred orientation of one molecule to a second when bound to each other to form a stable complex. Knowledge of the preferred orientation in turn may be used to predict the strength of association or binding affinity between two molecules using for example, scoring functions.

Despite their theoretical very high relevance, de novo design methods did not become established tools in drug design because of the presence of several drawbacks that limited their application to real world cases.

In the development of a new docking algorithm, there are two key features of docking simulations to keep in mind: speed and accuracy. The main objective is to obtain a fast method that is able to screen molecular libraries to discover novel lead compounds (in virtual screening) and also to reproduce experimental ligand conformation (i.e., crystallographic ligand pose). The docking process begins with the application of the docking algorithm to find a pose of the ligand molecule in the active site. Sampling the degrees of freedom is not a trivial task because even relatively small organic molecules can contain many conformational degrees of freedom, and the process must be performed with a certain accuracy to identify the conformation that best matches the receptor active site. Then the different proposed ligand−receptor complexes are evaluated and ranked by the scoring function. The first molecular docking programs were used to treat both the protein and ligand molecule as rigid bodies, fixing all the internal degrees of freedom, except for the three translations and three rotations. Within this concept, the only way to address the conformational flexibility of the ligands is to pre-generate a library of ligand conformations that are formally treated as separated molecular entities. Examples are FRED and DOCK4.0. These approaches were quickly replaced by algorithms able to explicitly take into account the conformational degrees of freedom during docking. Several ligand flexibility algorithms have been proposed and can be divided into three main families according to the type of search: systematic, stochastic, and deterministic or simulation methods. To the first family belong those algorithms that try to explore all the degrees of freedom in a molecule, as QXP, that carries out full conformational searches for flexible cyclic and acyclic molecules with extremely good results. The main issue of this kind of approach is the combinatorial explosion of ligand conformations number. Therefore, ligands are generally first divided into fragments and then incrementally grown within the binding pocket; examples are FlexX, Surflex, and eHITS. Stochastic-based algorithms make random changes, usually changing one degree of freedom at a time; examples are Monte Carlo (MC) methods and evolutionary algorithms applied, for example, by ICM and AutoDock. Molecular dynamic is the most used deterministic approach. However, the main concern regarding this kind of method is that often molecular dynamic is not able to cross high-energy barriers within short simulation time, and therefore, the system gets trapped in local minima. To avoid this problem, several attempts have been proposed, such as starting the simulation from different ligand positions or simulate the system at different temperatures; however, these efforts are quite expensive

in terms of calculation time, limiting the application of molecular dynamic to docking only one or few compounds.

Regardless of the way in which the docking process is handled, all the available docking software share at least the feature that the docking process is controlled by the values of several user-adjustable parameters, and an appropriate choice of these parameters is a prerequisite to obtain meaningful results. However, in many applications, users tend to adopt default settings, assuming that they will yield reasonably good results, regardless of the specific problem in which they are involved. Thus, providing the best ensemble of "default settings" is crucial for optimizing the program's output under standard conditions. Furthermore, having an optimized set of default parameters enables benchmarking of the performance of the program at the best of its possibility.

The LiGen suite of programs, consisting of a set a modules including docking modules LiGenPass for binding site recognition, LiGenPocket for binding site analysis and structure-based pharmacophore definition, LiGenDock for docking and virtual screening, in addition to LiGenBuilder for de novo design.

The LiGen source code has been written in such a way that a given workflow can be composed either by a pipe of different executables or can be compiled in a single embedded binary executable. In conclusion, LiGen represents a new method for de novo design based on pharmacophore-driven docking and accurate chemical reaction mapping and implemented to be suitable for HPC applications and portable to many hardware architectures.

2.5 Description of the infrastructure (System Architecture)

The GALILEO supercomputer has been introduced in January 2015 and it is available to the Italian public and industrial researchers. It is the national **Tier-1** system for scientific research. The system is also available to European researchers as a **Tier-1** system of the **PRACE** **(www.prace-project.eu)** infrastructure. GALILEO is ranked #105 in the TOP500 list of June 2015, with R_{max} of 0.7 PFLOPS and R_{peak} of 1.1 PFLOPS (Xeon Phi). GALILEO supercomputer is used to optimize and develop applications targeted at hybrid architectures, leveraging software applications in the fields of computational fluid dynamics, material and life science, and geophysics.

2.5.1 Hardware Overview

The Galileo cluster consists of 516 computational nodes of which 132 are regular compute nodes and 384 accelerated nodes. Each node is a powerful x86-64 computer, equipped with 16 cores (two 8-cores Intel Xeon processors) and 128GB RAM. The nodes are interlinked by high speed InfiniBand and Ethernet networks. All nodes share 3.0+ PB /home, /work and /scratch GPFS disk storage to store the user files. Users may also use node's storage with a global capacity of 0.5 PB which is available as local scratch data. The user access to the Galileo cluster is provided by 8 login plus 8 visualization nodes.

Table 2.2: Configuration parameters of GALILEO supercomputer

General	
Primary purpose	High Performance Computing
Architecture of compute nodes	x86-64
Operating system	CentOS 7.0 Linux
Compute nodes	
Totally	516
Processor	2x Intel Xeon E5-2630v3, 2.4GHz, 8cores
RAM	128GB, 8.0GB per core, DDR4@2133 MHz
Local disk drive	Yes, 1 TB
Compute network / Topology	InfiniBand QDR40 / Fat-tree
W/o accelerator	132
MIC/GPU accelerated	384
Total	
Total theoretical peak performance (Rpeak)	1213 TFLOPS
Total amount of RAM	80 TB

Table 2.3: Technical details of the compute node of GALILEO supercomputer

Node	Count	Processor	Cores	Memory	Accelerator
w/o accelerator	132	IBM NX365 M5, 2x Intel Xeon E5-2630v3, 2.4GHz	16	128GB	-
MIC accelerated	344	IBM NX365 M5, 2x Intel Xeon E5-2680v3, 2.5GHz	16	128GB	2x Intel Xeon Phi 7120P, 61cores, 16GB RAM
GPU accelerated	40	IBM NX365 M5, 2x Intel Xeon E5-2680v3, 2.5GHz	16	128GB	2x NVIDIA K80, 4992cores, 16GB RAM

Table 2.4 Login and visualization nodes of GALILEO supercomputer

Node	Count	Processor	Cores	Memory	GPU Accelerator
Login/Viz	8	IBM NX365 M5, 2x Intel Xeon E5-2680v3, 2.5GHz	16	128GB	2x NVIDIA K40, 2880cores, 12GB RAM

Storage

The overall storage architecture of HPC and HPDA CINECA systems is shown in **Figure 2.4.**

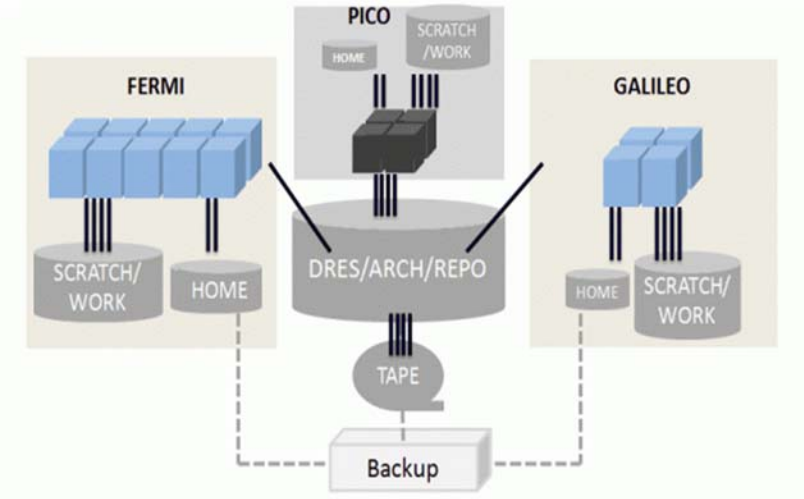

Figure 2.4. Overall storage model of I/O at CINECA.

There are three main shared file systems on Galileo cluster, the HOME, WORK and SCRATCH. All login and compute nodes may access same data on shared filesystems. Compute nodes are also equipped with local (non-shared) scratch and tmp filesystems. All shared filesystem are based on IBM GPFS served by 12 fileserver nodes to all computing and login/viz nodes.

The storage organisation conforms to the CINECA infrastructure. In addition to the home directory ($HOME), for each user is defined a scratch area $CINECA_SCRATCH, a large disk for storing run time data and files. $WORK is defined for each active project on the system, reserved for all the collaborators of the project. This is a safe storage area to keep run time data for the whole life of the project.

$DRES points to the shared repository where Data RESources are maintained. This is a data archive area available only on-request, shared with all CINECA HPC systems and among different projects. $DRES is not mounted on the compute nodes. This means that you can't access it within a batch job: all data needed during the batch execution has to be moved on $WORK or $CINECA_SCRATCH before the run starts.

Since all the filesystems are based on gpfs (General Parallel FIle System), the usual unix command "quota" is not working. Use the local command "cindata" to query for disk usage and quota ("cindata -h" for help).

Table 2.5: Summary of storage technical details

Mountpoint	Usage	Protocol	Net Capacity	Throughput	Limitations	Access	Services
/home	home directory	GPFS	5.5 TB	120 GB/s(*)	Quota 50GB	Compute and login nodes	backed up
/work	large project files	GPFS	2.8 PB	120 GB/s(*)	Quota *on-project*	Compute and login nodes	Backed up *on-project*
/scratch	job temporary data	GPFS	285 TB	120 GB/s(*)	Quota 52TB	Compute and login nodes	files older 30 days removed
local /scratch	job temporary data, node local	local	0.5 PB	1.2 GB/s(**)	none	Compute nodes	purged after job ends

(*) 12x nodes servicing I/O at 10 GB/s each. 120 GB/s shared among /home, /work and /scratch.
(**) 1.2 GB/s per computing node; 620 GB/s aggregated I/O throughput.

Network
All compute and login nodes of Galileo are interconnected by a fat-tree Intel Infiniband network with OFED v1.5.3, capable of a maximum bandwidth of 40Gbit/s between each pair of nodes and by Gigabit Ethernet network. Only Infiniband network may be used to transfer user data.

Infiniband Network
All compute and login nodes of Galileo are interconnected by 35 Intel/Q-Logic Infiniband QDR network switches (36 ports @40Gbps each). The network topology is a Fat-tree, 2-1 blocking.

The compute nodes may be accessed via the Infiniband network using ib0 network interface. MPI can be used to establish native Infiniband connection among the nodes.

2.5.2 Software Overview

Batch
Batch jobs are managed by the PBS batch scheduler, described in the Section "Batch Scheduler PBS".

On GALILEO, it is possible to submit jobs of different types, using only one "routing" queue: just declare how many resources you need and your job will be directed into the right queue with a right priority.

The maximum number of nodes that you can request is 128 with a maximum walltime of 24 hours. If you do not specify the walltime, a default value of 30 minutes will be assumed. The maximum amount of memory for each node is 120 GB (to be specified as the value of the "mem" resource in the #PBS -l select directive).

For moving your data the "archive" queue is available with one cpu and a maximum walltime of 4 hours. In order to use this queue you have to specify the PBS flag "-q": #PBS -q archive

Graphic session
For remote visualization we use the RCM tool and the EnginFrame environment.

Programming environment
The programming environment of the GALILEO machine consists of a choice of compilers for the main scientific languages (Fortran, C and C++), debuggers to help users in finding bugs and errors in

the codes, profilers to help in code optimisation. In general you must "load" the correct environment (module) also for using programming tools like compilers, since "native" compilers are not available.

The general available compilers are:

- INTEL (ifort, icc, icpc) : ► module load intel
- PGI - Portland Group (pgf77,pgf90, pgf95, pghpf, pgcc, pgCC): ► module load pgi
- GNU (gcc, g77, g95): ► module load gnu

Hybrid programming (Intel PHI)
Some of the nodes of the system are equipped with 2 accelerators per node. They can be addressed within C or Fortran programs by means of the Intel compilers suite.

Furthermore, the end-user has access to a full implementation of the Intel MKL library. The Intel Math Kernel Library (Intel MKL) enables improving performance of scientific, engineering, and financial software that solves large computational problems. Intel MKL provides a set of linear algebra routines, fast Fourier transforms, as well as vectorised math and random number generation functions, all optimized for the latest Intel processors, including processors with multiple cores. Intel MKL is thread-safe and extensively threaded using the OpenMP technology.

Hybrid programming (NVIDIA K80)
All tools and libraries required in the GPU programming environment are contained in the CUDA toolkit. The CUDA toolkit is made available through the “cuda” module which made available to the end user the most updated CUDA toolkit and driver (at present version V7).

CUDA, in addition to the C compiler, provides optimized GPU-enabled scientific libraries for linear algebra, FFT, random number generators, and basic algorithms (such as sorting, reductions, signal processing and image processing,) through the following libraries:

- CUBLAS: GPU-accelerated BLAS library
- CUFFT: GPU-accelerated FFT library
- CUSPARSE: GPU-accelerated Sparse Matrix library
- CURAND: GPU-accelerated RNG library
- CUSOLVER: provides a collection of dense and sparse direct solvers which deliver significant acceleration for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications.
- CUDA NPP: nVidia Performance Primitives
- THRUST: a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL).

Debuggers and profilers
Along with the standard debuggers (GNU gdb, PGI pgdbg, Intel idb) and profilers (GNU gprof, PGI pgprof, Intel iprof) coming with the compilers described so far, the Galileo development system offers to the end-users the following tools:

- SCALASCA
- TOTALVIEW
- TAO
- VALGRIND

Parallel programming
The parallel programming on Galileo is based on IntelMPI and OpenMPI versions of MPI. The libraries and special wrappers to compile and link the personal programs are contained in several modules, one for each supported suite of compilers.

2.8 References

[1] Beccari, A. R. et al; J. Chem. Inf. Model. 2013, 53, 1518-1527

[2] Bohm, H. J.; Jcomput. Aid. Mol. Des. 1992, 6, 61-78

[3] Honma, T.; et al.; J. Med. Chem. 2001, 44, 4615−27

[4] Tan, J. J. et al.; J. Med. Chem. 2011, 7, 309−16

[5] Nicolaou, C. A. et al.; J. Chem. Inf. Model. 2009, 49, 295−307.

[6] Yuan, Y. et al.; J. Chem. Inf. Model. 2011, 51, 1083−1091.

[7] Gillet, V. J. et al.; J. Chem. Inf. Comput. Sci. 1994, 34, 207−17.

[8] Sova, M. et al.; Bioorg. Med. Chem. Lett. 2009, 19, 1376−9.

[9] Eisen, M. B. et al.; Proteins 1994, 19, 199−221.

[10] Westhead, D. R. et al.; J. Comput. Aided Mol. Des. 1995, 9, 139−48.

[11] Clark, D. E. et al.; J. Comput. Aided Mol. Des. 1995, 9, 13−32.

[12] Waszkowycz, B. et al.; J. Med. Chem. 1994, 37, 3994−4002.

[13] Hartenfeller, M. et al.; PLoS Comput. Biol. 2012, 8, e1002380.

[14] Schneider, G. et al.; Nat. Rev. Drug Discovery 2005, 4, 649−663.

[15] Roe, D. C. Computer-Aided Molecular Design: De Novo Design. In Handbook of Chemoinformatics Algorithms, first ed.; Faulon,J.-L., Bender, A., Eds.; Chapman and Hall/CRC Taylor & Francis Group: Boca Raton, FL, 2010; pp 295−315.

[16] Ji, H. Fragment-Based Drug Design: Considerations for Good ADME Properties. In ADMET for Medicinal Chemists: A Practical Guide, first ed.; Tsaioun, K., Kates, S. A., Eds.; John Wiley & Sons, Inc.: Hoboken, NJ, 2011; pp 417−485.

[17] Proschak, E. et al.; In Chemoinformatics Approaches to Virtual Screening, first ed.; Varnek, A., Tropsha, A., Eds.; The Royal Society of Chemistry: Cambridge, UK, 2008; Vol. 0, pp 217−239.

[18] Kutchukian, P. S. et al.; Exp. Opin. Drug Disc. 2010, 5, 789−812.

[19] Jia, Y. et al.; Eur. J. Med. Chem. 2010, 45, 1304−13.

[20] E.g. see, extension to the GCC language family at http://www.univ-orleans.fr/SCIENCES/INFO/RESSOURCES/webada/doc/gnat/gcc_6.html

[21] https://en.wikipedia.org/wiki/Libfixmath

[22] http://www.codeproject.com/Articles/37636/Fixed-Point-Class

[23] E.g. see, http://www.mpfr.org/ for a review.

3 USE CASE 2: Self-adaptive Navigation System

3.1 Summary

The basic idea of the self-adaptive navigation system use case is to combine server-side and client-side data knowledge and their routing capabilities to provide the most efficient navigation system in the context of smart cities. In such a use case, we assume a significantly large portion of drivers participating in the system. The efficiency is essential given that the routing system needs to serve many requests requiring potentially huge computation power. Then still, even without limits to be reached, it is desirable to optimize the execution of such a system with respect to the energy efficiency.

We are identifying hot spots of the system at the level of algorithms and hardware runtime configurability, which are suitable candidates for an application of auto-tuning optimization techniques. The hot spots are not only the places, where it seems necessary to boost the performance, e.g. with exploiting parallelism, but also the places, where it is interesting to apply strategies to provide graceful degradation of a quality of service, e.g. to support scalability or achieve energy savings. Finally, the novel optimization techniques, developed in the scope of this project, can be applied at those places.

In the specification of the software architecture for our system, we identified the places for an application of optimization techniques in the following modules: server-side dynamic online routing calculation, server-side global traffic view calculation, communication interface between the server-side and the client-side, and the navigation top-level controller.

In a deeper analysis of those modules, several requirements for optimization techniques have been collected.

3.2 Motivations

Current navigation systems are divided into two categories: (1) systems that can provide navigation in an *offline mode* (maps are stored on a device, route calculation is done locally on a device), and (2) systems that extensively and intensively use a server side, called *online* navigation (both for map loading and for route calculation). The main disadvantage of the offline navigation is that it might suffer from not having the maps up-to-date, not knowing the online traffic situation, or that the routing algorithm is limited in terms of performance capacity, thus leading to suboptimal routes. The main disadvantage of the online navigation is that the internet connectivity is not always stable, especially with 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 better of the two worlds can be applied at appropriate moments. Currently, there are no such navigation systems on the market, due to their higher complexity and the lack of relevant input data.

Compared to other navigation systems, we can do a proper load balancing between client side (SYGIC) and server side (IT4I) navigation. Moreover, thanks to the collecting of on-line information about the traffic from different sources and because of the computing power available on the server side, we are able to take more accurate routing decisions.

The main opportunity for applying a sophisticated self-adaptive navigation system with dynamic and combined routing lies in the context of Smart Cities. The motivation is to solve growing traffic load in cities by finding the best utilization of the road network ideally involving all the drivers in the city into the central navigation system run on a HPC centre as shown in **Figure 3.1.**

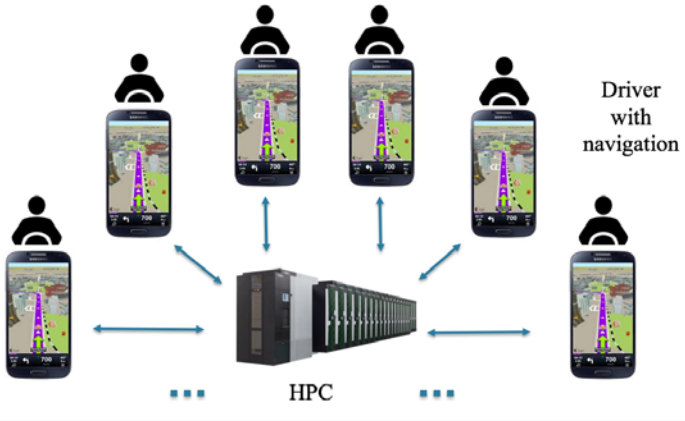

Figure 3.1: Main system actors.

Another aspect is the inclusion of in-car data into the routing and rerouting algorithms. In-car data from the client-side navigation systems can bring valuable information to route calculations to provide increased routing efficiency. Examples are detection of excessive acceleration and braking, detection of traffic flow, building up the knowledge of driver driving preferences.

While it is too early to forecast market figures for HPC-based navigation systems, it is worth noting that the ETP4HPC [ETP4HPC13b] vision paper identifies complex navigation systems as a target for HPC: "Complex transportation systems (air or ground) can also be optimized through the use of HPC. The availability of sensors to track vehicles and of communication to influence their behaviour enables a control system that can optimize time to destination and fuel consumption. However, such a control system will need huge computing power and data bandwidth. "

The outcome of the ANTAREX project could enable SYGIC, as European SME, to access at an affordable cost, the supercomputing resources needed to offer an enhanced navigation system to the customers, keeping their leading position in the market worldwide. Especially in cities, which are now challenged by traffic jams, and are becoming denser each day (predicted 60% population of the world lives in cities by 2025), the need for a sophisticated and high performant central navigation system processing many vehicles simultaneously to a global maximum efficiency will be more emphasized. SYGIC believes that it can play the key role in such systems through its adaptable client-side navigation when first such systems emerge.

The impact of the project on the future development of navigation systems could be significant for the society. With the technology developed in this project, the navigation experience can be tailored to specific needs of individual drivers, with an increased security, as well as with a potential to provide a global navigation optimum. This could lead to stimulating the development of smart city navigation systems as it is in interest of cities to optimize the traffic from the global point of view.

3.3 State-of-the-art

The integration of real-time or near real-time data into the routing algorithms is subject of dynamic routing. Dynamic routing is based on various types of input data sources including traffic data (community data from navigation provider, floating car data (FCD), data from traffic detectors, smart camera devices, etc.) [Liu13], meteorological data, and event data (integrated emergency system reports, reports from navigation providers, etc.).

FCD provide average travel times and speeds along road links or paths [Khan12, Zheng12, McLeod12, Zeng12, Jones03]. It is very useful type of data, because it better represents dynamism of the traffic than stationary sensors. It is possible to deploy FCD in order to predict short-term travel

conditions and automatically detect incident or critical situations [Ghosh09, Vlahogianni09, Wu11], or determine Origin-Destination traffic flow patterns [Shafer02, Ma12]. Akira Kinoshita et al. [Adachi14] use probabilistic topic model, while Ruimin Li et al. [Ben-Akiva15] use mixture models to compute traffic incident duration. Yangbeibei Ji et al. [Ji09] present slightly different approach to traffic incident length predictions in form of cell transmission model.

Reliability of all types of estimates based on FCD highly depends on the percentage of floating cars participating in the traffic flow. Several FCD systems were presented, integrating short-term traffic forecasting based on current and historical FCD. However, these systems exploit data mostly from car flotillas to deliver real time traffic speed information throughout large cities, signalized urban arterial roads or particular parts of traffic network, for example Italian motorway [Fabritiis08], Berlin [Kuhns11], Beijing [Li09], Vienna [Graser12], and many others.

Departure time highly affects route planning in the real road network. The fastest path is actually a time-dependent shortest path [Orda90] in a graph with variable travel times used as edge weights. Travel times on the edges are usually computed as historic averages [Rice04], even though this is not sometimes sufficient [Fan05, Hofleitner12].

The main disadvantage of time-dependent route planning is that it does not consider accidental events. There can be irregular but recurrent traffic congestions at some roads which navigation services should consider in the computation of the path and offer alternative routes [Miller-Hooks00]. Even though the probability of the traffic congestion can be very low, the delay in the case the congestion happens can be very long. Therefore, authors of [Nikolova06] and [Yang14] consider uncertain traffic events and their probabilities in the computation of the path. Algorithms described in [Hua10, Chen14a, Chen14b, Sun14] try to find a solution for time-dependent shortest path problem in uncertain networks.

Recently, new algorithms for routing in the road networks using dynamic data [Bast07], [Yuan11] have been developed. Unfortunately, dynamic data pre-processing requires graph reweighting in the routing algorithms in many cases [Ding08], which leads to higher computational complexity. The dynamic navigation problem becomes more complex even if we take into account the current network traffic situation with the focus on other navigation clients relevant to the same location. Such implementation of the dynamic routing regarding the large amount of vehicles within a city requires high computational power on the server side, specialized data structures, optimized algorithms for HPC, and open communication interfaces.

Projects published in [Schrank12] and [Otaegui13] are focused on traffic data processing. The projects address different aspects: detection of current traffic, modelling of traffic flow, traffic data publication, traffic control, etc. Processed data are offered for a variety of users. Different types of users need different styles of presentation of results. In order to satisfy users' needs several styles of visualization have been introduced in [ITO14], [SCL14] and [Calabrese11].

Other navigations are usually either client based or server based. Compared to state-of-the-art navigation systems, we can do a proper load balancing between client side (SYGIC) and server side (IT4I) navigation. Moreover, thanks to the collecting on-line information about the traffic from different sources and because of the computing power we have on the server side, we are able to do more accurate decisions based on the global view on the traffic situation and other factors that can influence it.

3.5 Description of the infrastructure (System Architecture)

3.5.1 Specification of the infrastructure – IT4Innovations

IT4Innovations supercomputing infrastructure consists of two supercomputers: **ANSELM** supercomputer with a theoretical computing performance of 94 TFLOPS, and **SALOMON** supercomputer with a theoretical computing performance of 2 PFLOPS.

Hardware Overview – ANSELM

ANSELM supercomputer consists of 209 computational nodes named cn[1-209] of which 180 are regular compute nodes, 23 GPU Kepler K20 accelerated nodes, 4 MIC Xeon Phi 5110 accelerated nodes and 2 fat nodes. Each node is a powerful x86-64 computer, equipped with 16 cores (two eight-core Intel Sandy Bridge processors), at least 64GB RAM, and local hard drive. The user access to the ANSELM cluster is provided by two login nodes login[1-2]. The nodes are interlinked by high speed InfiniBand and Ethernet networks. All nodes share 320TB /home disk storage to store the user files. The 146TB shared /scratch storage is available for the scratch data.

The fat nodes are equipped with large amount (512GB) of memory. Virtualization infrastructure provides resources to run long term servers and services in virtual mode. Fat nodes and virtual servers may access 45TB of dedicated block storage.

There are four types of compute nodes:

- 180 compute nodes without the accelerator
- 23 compute nodes with GPU accelerator - equipped with NVIDIA Tesla Kepler K20
- 4 compute nodes with MIC accelerator - equipped with Intel Xeon Phi 5110P
- 2 fat nodes - equipped with 512GB RAM and two 100GB SSD drives

Table 3.1: Configuration parameters of ANSELM supercomputer

In general	
Primary purpose	High Performance Computing
Architecture of compute nodes	x86-64
Operating system	Linux
Compute nodes	
Totally	209
Processor cores	16 (2x8 cores)
RAM	min. 64 GB, min. 4 GB per core
Local disk drive	yes - usually 500 GB
Compute network	InfiniBand QDR, fully non-blocking, fat-tree
w/o accelerator	180, cn[1-180]
GPU accelerated	23, cn[181-203]
MIC accelerated	4, cn[204-207]
Fat compute nodes	2, cn[208-209]
In total	
Total theoretical peak performance (Rpeak)	94 Tflop/s
Total max. LINPACK performance (Rmax)	73 Tflop/s
Total amount of RAM	15.136 TB

Table 3.2: Compute nodes of ANSELM supercomputer

Node	Processor	Memory	Accelerator
w/o accelerator	2x Intel Sandy Bridge E5-2665, 2.4GHz	64GB	-
GPU accelerated	2x Intel Sandy Bridge E5-2470, 2.3GHz	96GB	NVIDIA Kepler K20
MIC accelerated	2x Intel Sandy Bridge E5-2470, 2.3GHz	96GB	Intel Xeon Phi P5110
Fat compute node	2x Intel Sandy Bridge E5-2665, 2.4GHz	512GB	-

Storage

There are two main shared file systems on Anselm cluster, the HOME and SCRATCH. All login and compute nodes may access same data on shared filesystems. Compute nodes are also equipped with local (non-shared) scratch, ramdisk and tmp filesystems.

Configuration of the storages:

HOME Lustre object storage

- One disk array NetApp E5400
- 22 OSTs
- 227 2TB NL-SAS 7.2krpm disks
- 22 groups of 10 disks in RAID6 (8+2)
- 7 hot-spare disks

SCRATCH Lustre object storage

- Two disk arrays NetApp E5400
- 10 OSTs
- 106 2TB NL-SAS 7.2krpm disks
- 10 groups of 10 disks in RAID6 (8+2)
- 6 hot-spare disks

Lustre metadata storage

- One disk array NetApp E2600
- 12 300GB SAS 15krpm disks
- 2 groups of 5 disks in RAID5
- 2 hot-spare disks

Network

All compute and login nodes of Anselm are interconnected by Infiniband QDR network and by Gigabit Ethernet network. Both networks may be used to transfer user data.

Infiniband Network

All compute and login nodes of Anselm are interconnected by a high-bandwidth, low-latency Infiniband QDR network (IB 4x QDR, 40 Gbps). The network topology is a fully non-blocking fat-tree.

The compute nodes may be accessed via the Infiniband network using ib0 network interface. The MPI may be used to establish native Infiniband connection among the nodes.

The Fat tree topology ensures that peak transfer rates are achieved between any two nodes, independent of network traffic exchanged among other nodes concurrently.

Ethernet Network

The compute nodes might be accessed via the regular Gigabit Ethernet network interface eth0.

The network provides 114MB/s transfer rates via the TCP connection.

Hardware Overview – SALOMON

The Salomon cluster consists of 1008 computational nodes of which 576 are regular compute nodes and 432 accelerated nodes. Each node is a powerful x86-64 computer, equipped with 24 cores (two twelve-core Intel Xeon processors) and 128GB RAM. The nodes are interlinked by high speed InfiniBand and Ethernet networks. All nodes share 0.5PB/home NFS disk storage to store the user files. Users may use a DDN Lustre shared storage with capacity of 1.69 PB which is available for the scratch project data. The user access to the Salomon cluster is provided by four login nodes.

The technical parameters are summarized in the following **Table 3.3, 3.4, 3.5** and **3.6.**

Table 3.3: Configuration parameters of SALOMON supercomputer

In general	
Primary purpose	High Performance Computing
Architecture of compute nodes	x86-64
Operating system	CentOS 6.6 Linux
Compute nodes	
Totally	1008
Processor	2x Intel Xeon E5-2680v3, 2.5GHz, 12cores
RAM	128GB, 5.3GB per core, DDR4@2133 MHz
Local disk drive	no
Compute network / Topology	InfiniBand FDR56 / 7D Enhanced hypercube
w/o accelerator	576
MIC accelerated	432
In total	
Total theoretical peak performance (Rpeak)	2011 Tflop/s
Total amount of RAM	129.024 TB

Table 3.4: Compute nodes of SALOMON supercomputer

Node	Count	Processor	Cores	Memory	Accelerator
w/o accelerator	576	2x Intel Xeon E5-2680v3, 2.5GHz	24	128GB	-
MIC accelerated	432	2x Intel Xeon E5-2680v3, 2.5GHz	24	128GB	2x Intel Xeon Phi 7120P, 61cores, 16GB RAM

For remote visualization two nodes with NICE DCV software are available each configured as in **Table 3.5**.

Table 3.5: Remote visualization nodes of SALOMON supercomputer

Node	Count	Processor	Cores	Memory	GPU Accelerator
visualization	2	2x Intel Xeon E5-2695v3, 2.3GHz	28	512GB	NVIDIA QUADRO K5000, 4GB RAM

SGI UV 2000: For large memory computations a special SMP/NUMA SGI UV 2000 server is available as in **Table 3.6.**

Table 3.6: SMP/NUMA SGI UV 2000 server

Node	Count	Processor	Cores	Memory	Extra HW
UV2000	1	14x Intel Xeon E5-4627v2, 3.3GHz, 8cores	112	3328GB DDR3 @1866MHz	2x 400GB local SSD 1x NVIDIA GM200 (GeForce GTX TITAN X), 12GB RAM

Storage

There are two main shared file systems on Salomon cluster, the HOME and SCRATCH. All login and compute nodes may access same data on shared filesystems. Compute nodes are also equipped with local (non-shared) scratch, ramdisk and tmp filesystems.

HOME filesystem

The HOME filesystem is realized as a Tiered filesystem, exported via NFS. The first tier has capacity 100TB, second tier has capacity 400TB. The filesystem is available on all login and computational nodes. The Home filesystem hosts the HOME workspace.

SCRATCH filesystem

The architecture of Lustre on Salomon is composed of two metadata servers (MDS) and six data/object storage servers (OSS). Accessible capacity is 1.69 PB, shared among all users. The SCRATCH filesystem hosts the WORK and TEMP workspaces.

Configuration of the SCRATCH Lustre storage:

SCRATCH Lustre object storage

- Disk array SFA12KX
- 540 4TB SAS 7.2krpm disks
- 54 OSTs of 10 disks in RAID6 (8+2)
- 15 hot-spare disks
- 4x 400GB SSD cache

SCRATCH Lustre metadata storage

- Disk array EF3015
- 12 600GB SAS 15krpm disks

Table 3.7: Summary of storage technical details

Mountpoint	Usage	Protocol	Net Capacity	Through put	Limitations	Access	Services
/home	home directory	NFS, 2-Tier	0.5 PB	6 GB/s	Quota 250GB	Compute and login nodes	backed up
/scratch/work	large project files	Lustre	1.69 PB	30 GB/s	Quota 1TB	Compute and login nodes	none
/scratch/temp	job temporary data	Lustre	1.69 PB	30 GB/s	Quota 100TB	Compute and login nodes	files older 90 days removed
/ramdisk	job temporary data, node local	local	120GB	90 GB/s	none	Compute nodes	purged after job ends

Network
All compute and login nodes of Salomon are interconnected by 7D Enhanced hypercube Infiniband network and by Gigabit Ethernet network. Only Infiniband network may be used to transfer user data.

Infiniband Network
All compute and login nodes of Salomon are interconnected by 7D Enhanced hypercube Infiniband network (56 Gbps). The network topology is a 7D Enhanced hypercube.

The compute nodes may be accessed via the Infiniband network using ib0 network interface. The MPI may be used to establish native Infiniband connection among the nodes.

Monitoring
All software and tools are updated.

Table 3.8: Summary of software on ANSELM and SALOMON

Software	Anselm	Salomon
top	Yes	Yes
perf	Yes	Yes
valgrind	Yes	Yes
intel vtune	Yes	Not yet
Time	Yes	Yes
Likwid	Yes	Yes
intel pcm	Yes	Not yet
papi	Yes	Yes

Others
User can turn on or turn off turbo on nodes. It is impossible change number of nodes if the job is running.

3.5.2 Specification of the infrastructure – SYGIC

The client side navigation component (see block diagram in **Figure 3.3)** is the key component of the whole system as it provides the navigation service to an end client. In order to provide quality of service in terms of optimal routes, the SYGIC engine cooperates with a server-side. From a different perspective, the client side navigation component can be seen as an additional resource for route calculation for the server-side system.

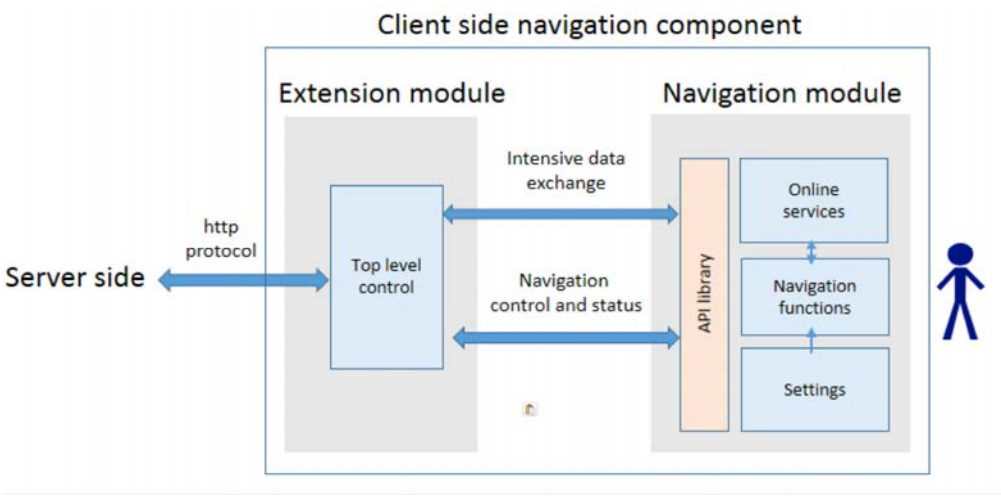

Figure 3.3: Client side navigation component.

Client side navigation component consists of the standard SYGIC navigation module equipped with API (containing ca. 50 functions), through which it is possible to access and control navigation internals, and the extension module, which will secure the cooperation with the server-side. We envision the extension module to communicate with the server-side using a dedicated http protocol and on the other side to communicate with the navigation component through the available API functions. The extension module will implement a top level control of the client side exploiting some auto-tuning and optimization techniques. In addition to it, it will implement a server connectivity protocol to meet our motivations for defining the communication interface (possibly a good candidate for standardization), which forms the system framework offering plug&play for different versions of client-side navigation components.

3.8 References

[Adachi14] J. Adachi, A. Kinoshita, and A. Takasu. Traffic incident detection using probabilistic topic model. Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference, 2014.

[Bast07] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. Science, Vol. 316, No. 5824, p. 566, 2007. DOI: 10.1126/science.1137521

[Ben-Akiva15] M. E. Ben-Akiva, R. Li, and F. C. Pereira. Competing risks mixture model for traffic incident duration prediction. Accident Analysis & Prevention Volume 75, 2015.

[Calabrese11] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti. Real-time urban monitoring using cell phones: A case study in Rome. Intelligent Transportation Systems, IEEE Transactions on 12(1), 141–151, 2011.

[Chen14] B. Y. Chen, W. Lam, A. Sumalee, Q. Li, H. Shao, and Z. Fang. Finding reliable shortest paths in road networks under uncertainty. Networks and Spatial Economics, 13(2):123–148, 2013.

[Chen14b] B. Y. Chen, W. H. K. Lam, A. Sumalee, Q. Li, and M. L. Tam. Reliable shortest path problems in stochastic time-dependent networks. Journal of Intelligent Transportation Systems, 18(2):177–189, 2014.

[Ding08] B. Ding, J. X. Yu and L. Qin. Finding Time-dependent Shortest Paths over Large Graphs. In Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '08, pp. 205-216, New York, NY, USA, ACM, 2008. DOI:10.1145/1353343.1353371

[ITO14] ITO map (2014), http://www.itoworld.com/static/map.html

[Fabritiis08] C. de Fabritiis, R. Ragona, and G. Valenti. Traffic estimation and prediction based on real time floating car data. In proceedings of 11th International IEEE Conference on Intelligent Transportation Systems, 2008. ITSC 2008., pages 197–203, 2008.

[Fan05] Y. Y. Fan, R. E. Kalaba, and J. E. Moore II. Arriving on time. Journal of Optimization Theory and Applications, 127(3):497–513, 2005.

[Ghosh09] B. Ghosh, B. Basu, and M. O'Mahony. Multivariate short-term traffic flow fore- casting using time-series analysis. Intelligent Transportation Systems, IEEE Trans- actions on, 10(2):246–254, 2009.

[Graser12] A. Graser, M. Dragaschnig, W. Ponweiser, H. Koller, M. S. Marcinek, and P. Widhalm. Fcd in the real world – system capabilities and applications. In proceedings of 19th ITS World Congress, Vienna, Austria, page 7, 2012.

[Hofleitner12] A. Hofleitner, R. Herring, P. Abbeel, and A. Bayen. Learning the dynamics of arterial traffic from probe data using a dynamic bayesian network. Intelligent Transportation Systems, IEEE Transactions on, 13(4):1679–1693, Dec 2012.

[Hua10] M. Hua, and J. Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In Proceedings of the 13th International Conference on Extending Database Technology, pages 347–358. ACM, 2010.

[Jones13] M. Jones, Y. Geng, D. Nikovski, and T. Hirata. Predicting link travel times from floating car data. In Proceedings of International IEEE Conference on Intelligent Transport Systems (ITSC), 2013.

[Khan12] R. A. I. Khan, B. Landfeldt, and A. Dhamdher. Predicting travel times in dense and highly varying road traffic networks using starima models. Technical report, School of Information Technologies, The University of Sydney and National ICT Australia, 2012.

[Kuhns11] G. Kuhns, R. Ebendt, P. Wagner, A. Sohr, and E. Brockfeld. Self evaluation of floating car data based on travel times from actual vehicle trajectories. In IEEE Forum on Integrated and Sustainable Transportation Systems, 2011.

[Li09] M. Li, Y. Zhang, and W. Wang. Analysis of congestion points based on probe car data. In Proceedings of International IEEE Conference on Intelligent Transporta- tion Systems, ITSC '09, pages 1–5, 2009.

[Liu13] S. Liu, Y. Yue, and R. Krishnan. Adaptive Collective Routing Using Gaussian Process Dynamic Congestion Models. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, pp. 704-712, New York, NY, USA, ACM, 2013. DOI:10.1145/2487575.2487598

[Ma12] Y. Ma, H. J. van Zuylen, and J. van Dalen. Freight origin-destination matrix estimation based on multiple data sources: Methodological study. In TRB 2012 Annual Meeting, 2012.

[McLeod12] D. S. McLeod, L. Elefteriadou, and L. Jin. Travel time reliability as a performance measure: Applying florida's predictive model on the state's freeway system. In TRB 2012 Annual Meeting, 2012.

[Miller-Hooks00] E. Miller-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. Networks, pages 35–52, 2000.

[Nikolova06] E. Nikolova, M. Brand, and D. R. Karger. Optimal route planning under uncertainty. In ICAPS, volume 6, pages 131–141, 2006.

[Orda90] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J. ACM, 37(3):607–625, July 1990.

[Otaegui13] O. Otaegui, O. Desenfans, L. Plault, and A. Lago. TAXISAT: A driverless GNSS based taxi application capable of operating cost effectively. In 9th ITS European Congress, pp. 1634-1641, 2013.

[Rice04] J. Rice and E. Van Zwet. A simple and effective method for predicting travel times on freeways. Intelligent Transportation Systems, 5(3):200–207, 2004.

[Schafer02] R. P. Schafer, K.-U. Thiessenhusen, and P. Wagner. A traffic information system by means of real-time floating-car data. In Proceedings of ITS World Congress 2002, Chicago, USA, 2002.

[Schrank12] D. Schrank, B. Eisele, and T. Lomax. TTI's 2012 urban mobility report. Texas A&M Transportation Institute. The Texas A&M University System, 2012.

[SCL14] SENSEable City Laboratory MIT: REAL TIME ROME (2011), (2014), (2006), http://senseable.mit.edu/realtimerome/

[Sun14] S. Sun, Z. Duan, S. Sun, and D. Yang. How to find the optimal paths in stochastic time-dependent transportation networks? In Intelligent Transportation Systems, 2014 IEEE 17th International Conference on, pages 2348–2353. IEEE, 2014.

[Vlahogianni09] E. I. Vlahogianni. Enhancing predictions in signalized arterials with information on short-term traffic flow dynamics. Journal of Intelligent Transportation Systems, 13(2):73–84, 2009.

[Wu11] Y. J. Wu, F. Chen, C. T. Lu, and B. Smith. Traffic flow prediction for urban network using spatio-temporal random effects model. In 91st Annual Meeting of the Transportation Research Board, 2011.

[Ji09] Y. Ji, W., Daamen, X. Zhang, and L. Sun. Traffic incident recovery time prediction model based on cell transmission model. Proceedings of the 12th International IEEE Conference on Intelligent Transportation Systems, 2009.

[Yuan11] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with Knowledge from the Physical World. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, pp. 316-324, New York, NY, USA, ACM, 2011. DOI: 10.1145/2020408.2020462

[Yang14] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Multi-cost optimal route planning under time-varying uncertainty. In Proceedings of the 30th International Conference on Data Engineering (ICDE), Chicago, IL, USA, 2014.

[Zeng12] W. Zeng, Z. He, R. Lu, Zhuang, Lijiang, and X. Xia. Freeway segment speed estimation model based on distribution features of floating-car data. In TRB 2012 Annual Meeting, 2012.

[Zheng12] J. Zheng. Road travel time estimation with GPS floating car data. In Stander Symposium Posters. Book 191, 2012.