MDE4HPC: An Approach for Using Model-Driven Engineering in High-Performance Computing

Marc Palyart[1,2], David Lugato[1], Ileana Ober[2], and Jean-Michel Bruel[2]

[1] CEA* / CESTA,
33114 Le Barp - France
{marc.palyart,david.lugato}@cea.fr
[2] IRIT – Université de Toulouse,
118, route de Narbonne, 31062 Toulouse - France
{ober,bruel}@irit.fr

Abstract. With the increasing number of programming paradigms and hardware architectures, high performance computing is becoming more and more complex in exploiting efficiently and sustainably supercomputers resources. Our thesis is that Model Driven Engineering (MDE) can help us in dealing with this complexity, by abstracting some platform dependent details. In this paper we present our approach (MDE4HPC) based on Model Driven Engineering which – by describing the scientific knowledge independently of any specific platform – enables efficient code generation for multiple target architectures.

1 Introduction

One of the most visionary predictions in computer science forecasts that computer performance would increase by 40% per year. This prediction made in 1965 by Gordon Moore [1], is still remarkably accurate. While for over 30 years this performance increase preserved the sequential programming model, in more recent years multicore architectures became common among desktop computers, raising the need to rethink software evolution in terms of how to best exploit these new parallel architectures.

In a recent study [2], the author makes a critical analysis and examines how "Moore's dividend" was spent, since obviously software performance is far from observing the same performance gain laws. The analysis reveals that "Moore's dividend" was spent on increasing software size, software functionality, and programming complexity. In fact, a "law" on software performance evolution from Philipp Ross [3] states that "software is slowing faster than hardware is accelerating".

Some applications are by nature better suited for a parallel deployment [4], and high performance-computing (HPC) is undeniably one of the fields where the use of parallel architectures fits well.

* Commissariat à l'Énergie Atomique et aux énergies alternatives - French Atomic and Alternative Energy Commission.

I. Ober and I. Ober (Eds.): SDL 2011, LNCS 7083, pp. 247–261, 2011.

James Larus in [2] highlights the fact that according to the HPC community "each decimal order of magnitude increase in available processors required a major redesign and rewriting of parallel applications". Therefore, in order to ensure that the performance of HPC applications is increasing in step with the computer performance increase, major redesign and rewriting are needed.

In current practice, parallel programming models and in particular those addressing HPC are low level, machine specific and therefore complicate application porting. We believe this could change, and one way to do it is by applying Model Driven Engineering [5] specific techniques.

According to the MDE philosophy, system development should be built upon an abstract platform independent model. We are convinced that applying similar principles to HPC would be highly beneficial, although some effort is needed in order to set up the proper development environment. Our approach follows this direction, by proposing (1) a methodology based on successive model transformations that enrich progressively the model with platform information and (2) Archi-MDE an integrated development environment that supports the use of MDE for the development of HPC applications by implementing the proposed approach.

The rest of this paper is organised as follows: in Section 2 we present typical problems encountered when trying to program efficiently on forthcoming hybrid hardware architecture. In Section 3 we overview the state of the art of HPC development and we give the basic principles of MDE. In Section 4, we introduce MDE4HPC - our approach for applying MDE techniques in the development of HPC applications. In Section 5, we present Archi-MDE - an implementation of our approach that we use to asses the validity of MDE techniques in connection with HPC. Finally, in Section 6 we detail expected contributions of our research, that give directions for future work.

2 Problem Statement

In high performance computing, the development of efficient code for numerical simulations requires knowledge of a highly heterogeneous nature. Besides the fact that the modeling of a physical phenomenon is remarkably complex, developers are compelled to bear in mind the prerequisites for performance (tasks distribution, memory management, calculation precision). Moreover, the current trend is to depend on hardware specific library and instructions (NVIDIA Cuda [6], ATI Stream [7], IBM Cell [8]), hence code is strongly dependent on the hardware platform and often highly target dependent. Even though, as experience shows, good performances can be achieved, this approach has drawbacks: architecture dependency, mix-up of concerns and programming complexity. We discuss below these disadvantages in more detail.

Applications vs. supercomputers lifetime cycle. As shown in Figure 1 the life cycle of supercomputers is five to seven times shorter than scientific applications life cycle in our case. CEA experience shows that the simulation models

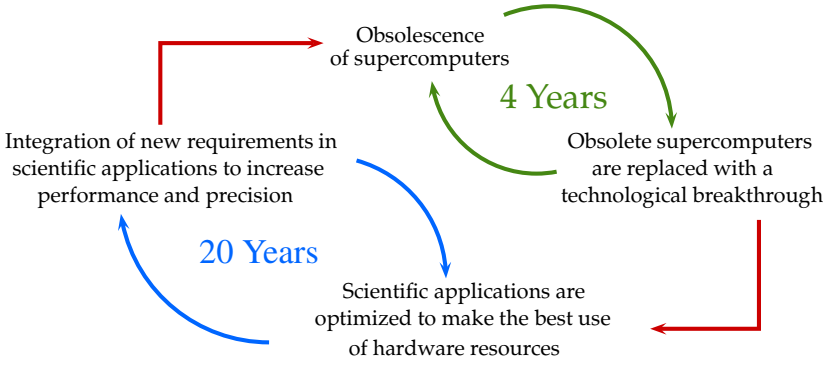

Fig. 1. CEA scientific applications life cycle versus supercomputers lifecycle

and numerical analysis methods associated with our professional problems have a lifetime in the order of 20 to 30 years and must therefore be maintained over that period, with all the additional problems that come with software maintenance over such a period of time (e.g. team turnover).

In parallel, through its TERA program [9], the CEA has decided that every four years its main supercomputer has to be replaced in order to increase its computation power by a factor superior to ten (Tera-1: 2002, Tera-10: 2006, Tera-100: 2010). With a pace faster than Moore´s law [1] hardware technological breakthroughs inevitably appear and software migration problems become an important issue.

Lack of separation of concerns. The problem to solve - the scientific knowledge of the physics - is entirely mixed with target dependent information, added to manage the parallelism. Once a complex system (multi-scale, multi-physics) has been built, it is hard to retrieve afterwards the physical models. As a result, maintenance and evolution become even more complicated.

Inaccessibility to domain experts. Programming complexity reduces the use of these workstations and supercomputers to a few scientists who are willing to spend a significant amount of time learning the specificities of a particular set of machines.

Furthermore, with the new generation of supercomputers made of hybrid machines (multicore CPUs mixed up with GPUs[1] or CELL processor) we will come close to the human limit to manage such systems at a low level. A classical study in human psychology [10], identified some limits on our capacity for processing information, especially in case it has "multidimensional" nature. These theoretical limits are seriously challenged by architectures with numerous processing elements, out of which some are built for very specific tasks.

This new generation of architecture would decrease even more the possibility of separation of concerns by adding more and more information to manage the

[1] Central Processing Unit / Graphics Processing Unit.

parallelism within the code. It is easily conceivable that a source code with MPI calls, OpenMP pragmas and CUDA code is not the best starting point to understand or retrieve the physics hidden inside the code. This is why we will look for an alternative solution.

The current tendency of replacing real-life experiments with numerical simulations in industrial systems conception in order to guarantee the reliability and safety of these systems is accelerating. As an example the CEA Simulation Program [11] has been built around three components: *physical modeling*, *numerical computation* and *experimental validation*. Thus, the use of computation results in order to guarantee the performance of a system is a replacement solution that allows to make up for the lack of directly exploitable experimental data. The credibility of such an approach is assessed through the restitution of past experiences and by domain expert certification. These additional constraints (domain validity assertion, uncertainty of forecasting computation, restitution of unexplored physics) due to the CEA's *Simulation Program* raise the need for a more abstract and formal approach. The progressive use of formal methods must be taken into account by the whole development process and associated supporting tools.

2.1 Current Development Process

The typical development process used is illustrated in Figure 2, using the SPEM [12] notation. Although the *design* phase is represented by only one activity in the diagram, three steps can be observed in reality but their precise limits are not formalized and may vary depending on the project. These three phases are presented below:

- *Physical model conception* – aims at describing the observable reality by finding which parts of the physics are required to solve the problem, and which are the best suited equations.
- *Numerical model conception* – aims at selecting a mathematical model to solve the physics equations selected in the previous phase.
- *Software conception* – aims at designing the software implementing the numerical model.

The design of these three models and of their relationships is critical. Thanks to MDE we believe that we can offer a more formalized way to define these models and the transformations which bind them together. As a collateral effect, we hope to enhance the validation phases.

3 State of the Art

In this section we review the state of the art with respect to the two main technologies we use in this paper: High Performance Computing and Model Driven Engineering. We start with an overview of technologies and research in High Performance Computing which are related to the problems exposed in the previous section. Then we present the Model Driven Engineering principles as well as its potential benefits for HPC.

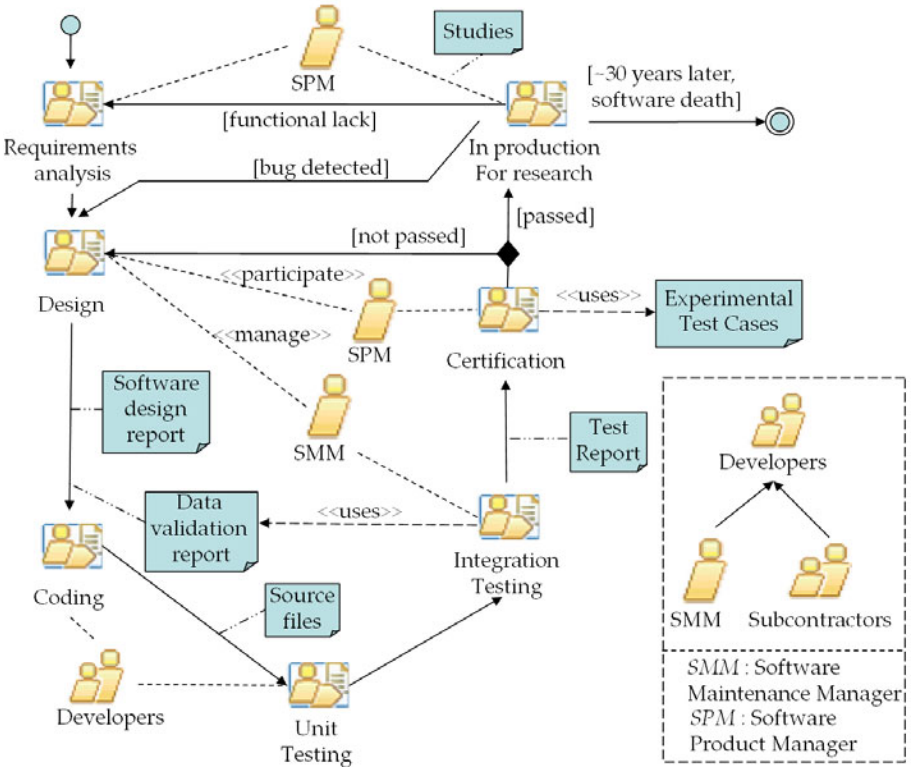

Fig. 2. Actual process development

3.1 The HPC World

Several solutions for the development of HPC applications are currently available. Most of them follow the tendency highlighted in Figure 3 [13]: the more abstract they are, the more specific to a science domain they tend to be.

For instance, two of the most used solutions on supercomputers are MPI (Message Passing Interface) [14] and Open MP (Open Multi Processing) [15]. MPI is a specification of routines which provides a low-level programming model which supports message passing programming. Thus it is suitable for distributed memory systems. OpenMP is an Application Program Interface which targets shared memory systems. It consists of a set of compiler directives (pragma), library routines, and environment variables which influence run-time behaviour. Both of these solutions can be used for nearly any scientific problem, but excluding the fact that they are suited for only a specific type of memory system they have a major problem: the abstraction level offered to the software developer is low. As expressed before, the consequences are multiple: low productivity of the developer, error prone, high complexity.

On the other side, several efforts of abstraction have been successfully attempted such as PETSc (Portable, Extensible Toolkit for Scientific Computation) [16] and

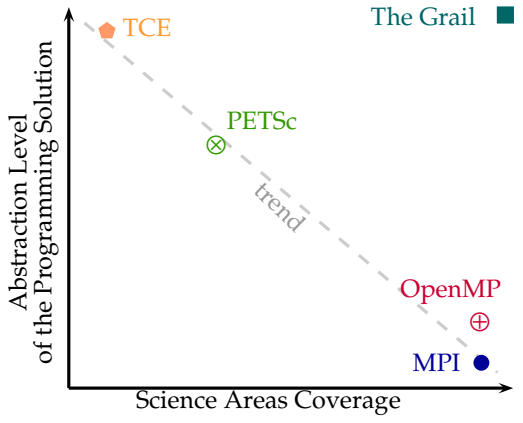

Fig. 3. Overall trend within HPC solutions

TCE (Tensor Contraction Engine) [17]. PETSc is a library built above MPI aimed at facilitating the solving of scientific applications modelled by partial differential equations and TCE is a compiler for a domain-specific language that enables chemists to develop their applications with only high-level descriptions. Unfortunately, these solutions tend to be to some extent specific to a particular scientific domain.

In the meantime, considerable research to tackle problems raised by the use of heterogeneous machines has been performed. Amongst the most noteworthy we can mention StarPU [18] and HMPP (Hybrid Multi-core Parallel Programming environment) [19]. StarPU is a unified runtime system that offers support for heterogeneous multicore architectures (CPU, GPGPUs, IBM Cell). The performances of such a system look promising but although part of the complexity is managed by the runtime, the creation of applications remains tedious for the developer. On the other hand, HMPP adopts an approach similar to OpenMP but is designed to handle HWAs(HardWare Accelerators) such as GPU. Thus it offers a respectable approach for improving legacy code. However, as its use is based on compiler directives which limit the separation of concerns, this solution can appear as less attractive for new developments.

3.2 Model Driven Engineering

In this section we try to review the basic principles of Model Driven Engineering and to analyse the benefits it can bring to the development of HPC applications.

MDE aims at raising the level of abstraction in software development, by using models that make it possible to abstract away from the technological and implementation dependent details. The models are organized in a way that encourages the separation of concerns: business and application logic is disconnected from the underlying platform technology. The goal of this abstraction is to enable

domain experts, who can be inexperienced in computer science, to concentrate on the problem to solve and on their expertise.

In MDE, the models are more or less abstract, depending on how much implementation dependent details they contain. High level models are refined by transformations into lower level models, the aim is to finally obtain a model that can be executed employing either code generation or direct model interpretation. The aim is also to generate a wide set of artefacts such as documentation or tests.

Models are commonly used in other engineering and science fields. MDE development led to their widespread use in software development, not only as passive entities or as "contemplative models" [20], but also as productive entities, intensively used in symbolic execution, model checking, model based testing, code generation, etc. Furthermore, MDE enables an increase of the automation of repetitive tasks through the use of models transformations.

Concerning the potential benefits a model based approach, the use of MDE philosophy and model based approaches in general have led to real success in industrial domains other than high performance computing. For instance, real-time and critical systems benefit from the UML (Unified Modeling Language) profiles UML-RT [21] (UML profile for Real Time) and MARTE [22] (Modeling and Analysis of Real-time and Embedded Systems). In fact, models enable the telecommunication [23], aeronautics and automotive industries to reduce cost and time in the development process [24,25]. Closer to our field, ongoing research based on MDE is being pursued on shared memory systems: standard multi-core computing systems [26] and multiprocessor System On Chip [27].

4 Overview of the Proposed Approach MDE4HPC

In this section we introduce the main concepts and technologies of our approach called MDE4HPC. First, we present the key ideas which define the core of our approach and finally we set out the additional services which aggregate around this core.

4.1 General MDA Principles Applied

The core of our methodology follows MDE principles and relies mainly on models and transformations. Figure 4 summarizes the key concepts of our approach.

In our approach, according to the MDE principles, the application developers (conjointly with scientists if they are not the same) start the development process by defining *Platform Independent Models* (PIMs) which are, by definition, independent from any specific hardware, library or even architecture. These models capture the solution to the problem that needs to be solved independently of any technological constraints. In consequence these models are sustainable, if the specifications do not change. Consequently, they are defined once, irrespective of the platform we aim to target.

A typical PIM contains a *data model* and a *tasks model*. The data model represents the data organization (for example the mesh structure) of the simulation

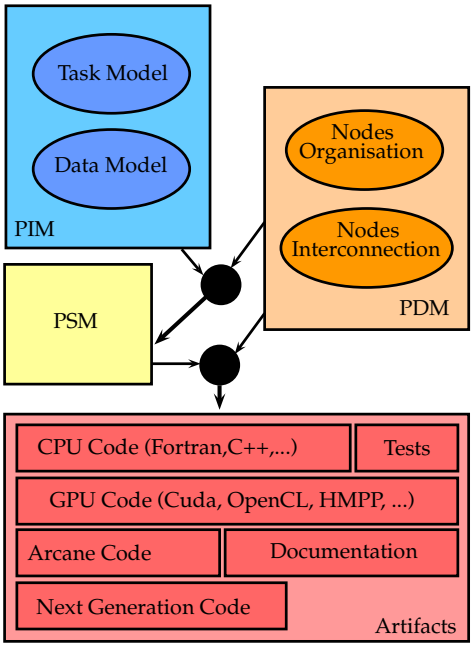

Fig. 4. Simplified description of the sequence of model transformations (represented as black circle) during the development process

and the task model is composed of elementary computational tasks that work on the data model, for example the mesh structure.

On the other side, the development makes use of a target platform model, under MDE vocabulary it is called *a Platform Description Model (PDM)*. In the case of our applications, the PDM covers:

- *Computational nodes organisation*: processing units (CPU, GPU, ...) architectures, memory description (size, type, bandwidth).
- *Computational nodes interconnection*: data link features (bandwidth).
- *File storage system* organisation and capacity.

The task of modelling the platform, that leads to the the PDM, can only be achieved by hardware architecture experts who fully understand the specificities of a particular computer (architecture). The same expert would also work on modelling transformations related to including platform aspects in the PIM.

As recommended by the MDA guidelines, the platform dependent model is combined with the PIM to form a *Platform Specific Model* (PSM). This process is often, a several step process, meaning that the transformation to a PSM is actually done in several stages, each integrating one or several aspects of the PSM. The principles of this multi-step transformation, are illustrated by Figure 5. Ideally, these transformations end up in a code generating transformation.

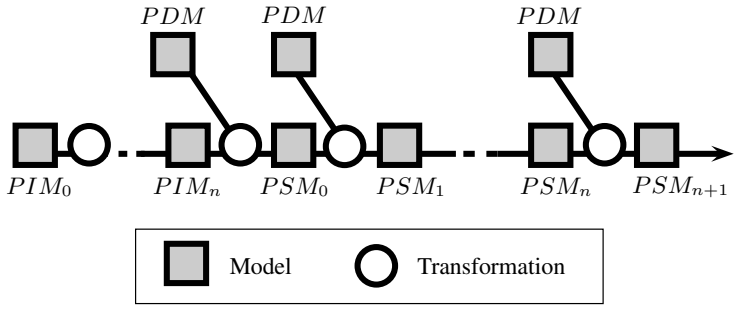

Fig. 5. Succession of transformations during the development process

The combination of different models helps us in dealing with the separation of concern as each model and each transformation are defined by a specialist of the domain modelled (physics, mathematics, software, hardware). However *the order* or *the combination* all of these transformations is not trivial because not all the transformation are commutative, associative or compatibles between them. For example, in the hypothesis of dealing with low level models a few steps before code generation, attempting to apply a set of optimizations, two optimizations such as loop unfolding and inlining could clash with each other if they were applied in the wrong order. As a result, the actual order in which the transformations should be applied is defined by domain experts, for instance by the optimisation expert who can decide the order of the unfolding and the inlining. Note that this problem is not new, the choice of the order of applying optimisations is always meaningful. What is new is the fact that this choice is more explicit.

Our approach does not target the automatic parallelization such as discussed in [28]. Instead, we introduce an approach where experts can focus on their domain of expertise.

The definition of our approach benefits from the exchanges between various domain experts. To achieve this, we have organised mixed team of physicists and numerical computer scientists to define the PIM. The computer system engineers were then in charge with the definition of the PDM. Finally, the transformations from PIM to PSM and from PSM to code were achieved conjointly by software engineers and computer system engineers, with the aid of model transformation tools.

In sum, none of these tasks were done completely automatically, still some of them were performed with automated assistance. The most important aspect here is that each of these tasks focuses on a particular expertise domain, thus was performed by a domain expert, that was able to focus on its own expertiese domain.

The models corresponding to the PIM, PSM and PDM, are described according to a Domain Specific Modeling Language (DSML) called *HPCML (High Performance Modeling Language)*. This DSML stands at level M_2 in the four-level architecture hierarchy [5] and it aims at formalize the description of models. The abstract syntax of a DSML is defined through a metamodel. The choice

of a DSML over a UML Profile stems from our previous work [29]. They are both situated at the M_2 level, UML profiles can be seen as an extension of the UML metamodel and DSML are defined with a new metamodel dedicated to the domain. If necessary, and as they both comply with the MOF (level M_3), transformations could be written to transform models compliant with our DSML HPCML into models compliant with its potential UML profile equivalent.

HPCML covers concepts related to task modeling (such as component, service) and data modeling (such as type, mesh). HPCML it is designed to suit constraints imposed by parallelism, and it is actually an evolution after feedbacks from a first version that was deployed and tested on mid-size projects in the context of the IDE Archi-MDE.

4.2 Openings Offered by the Models Use

The use of models in the way described in the previous section, opens the way to a collection of additional services that allow to increase developer productivity, improve code performance and quality. In the context of our approach, some of these services are already included in the Archi-MDE development environment, that will be presented in Section 5.

Model validation is one of the classical gains of using models in the development process, as it allows for early correctness checking and thus it makes it possible to improve quality by verifying the correctness of each component, before arriving to the actual code. The literature abundantly treats this topic and many techniques are available allowing for model based validations, the use of most of them would be beneficial in our approach.

In the context of our concrete setting, our goal, for the moment, is to put up a rapidly functional framework allowing to effectively use modelling in the context of HPC. Therefore, we do not intend, at least not for the moment, to develop specific validation techniques. Instead we will use already existing technologies that can be easily integrated into our setting. As Archi-MDE is developed with the Eclipse framework, for the moment we will use an existing Eclipse based OCL (Object Constraint Language) verification engine capable of inconsistency detection mechanism.

Symbolic execution and automatic test generation are also classical benefits of using modelling and the automatic test generation based on symbolic execution such as the approach presented on [30] can be a factor of both productivity and quality improvement. Although for the moment our approach does not cover this, we find it may bring substantial benefits in our context and we intend to enrich Archi-MDE with such capabilities, by adapting an already existing automatic execution and test generation engine.

Optimization. In the majority of cases, a model contains more and higher level information than the code resulting from it. The use of low level models, as introduced in the previous section, opens the way to start introducing optimizations in the context of these models [31].

In the context of Archi-MDE (that we will detail in Section 5) we generated sequential Fortran code to assess the interest of optimizations injected during model transformations. These optimisations concerned loop unrolling and inlining on Intel Xeon. We analysed the speed-up under the following two settings:

- the version optimized only by the compiler (in our case the Intel compiler);
- the version where the optimisation started at (low level) model stage, in the context of Archi-MDE and was then continued by the compiler.

Benchmarks comparing these optimisations gave us encouraging results, as we obtain a better speed-up when the optimizations started in Archi-MDE.

Software metrics proved their interest in the context of HPC applications [32]. The use of model metrics [33] should allow to perform live analysis of the models in order to determine ratings of the quality of the code produced and thus provide a useful feedback to the developer.

Performance forecasting is currently used in order to make system design decisions based on quantifiable demands of their key applications rather than using manual analysis which can be error prone and impractical for large systems [34]. These approaches could benefit from the existence of models that provide meaningful abstractions of the systems, in order to evaluate which parts of the application are compute-intensive and thus guide the developer as to where he needs to focus his attention to enhance performance.

Anti-patterns [35] provide an interesting mechanism of identifying possible errors before they actually manifest. The use of anti-patterns is more interesting in the context of models than of code. Therefore, our approach that consists in raising the place of models in the development cycle, opens the way to the definition and identification of anti-patterns for HPC models.

We overview here a list of functionalities that are enabled by a model based development of high-performance computing applications. This list is naturally not exhaustive, and was dressed up using feed-backs from HPC specialists on what would be useful and what would be feasible in this context.

The following section overviews Archi-MDE, the development framework that we have built for supporting the model based development of HPC applications. some of the functionalities we overview above are already part of this framework, such as the optimization, some other, such as the OCL consistency verification, can be re-used from a OCL based model analysis tool, while some other are for the moment on our future work list.

5 Archi-MDE

Archi-MDE is a user-supportive integrated development environment which aims to support the methodology associated with our approach.

There are marvellous IDEs and frameworks available. We chose to rely on the Eclipse platform and especially the open source Eclipse Modeling Project

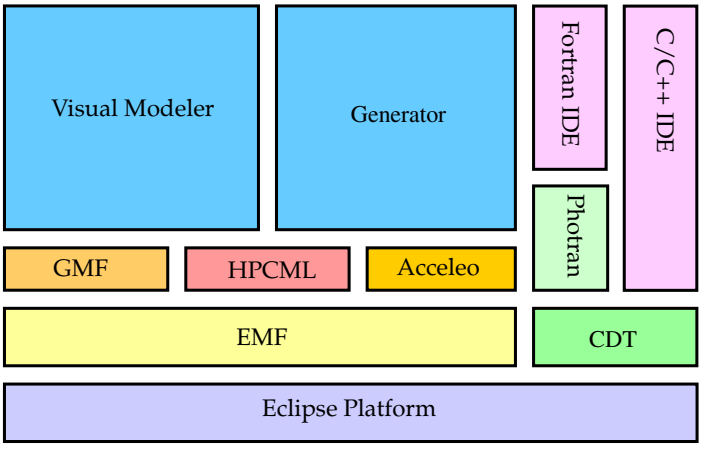

Fig. 6. Archi-MDE Architecture

[36] which focuses on the evolution and promotion of model-based development technologies. The principal reasons for this choice were the integration of code compilation/edition tools, the modularity, the wide and dynamic community behind the project, a robust infrastructure based on a system of plugins with extension points and a vast collection of existing plugins that can be reused. The open source aspect of the tool was also an important advantage.

Figure 6 gives an overview of the Archi-MDE tool architecture. The Eclipse Modeling Project provides a set of tooling, standards implementations and modeling frameworks such as EMF (Eclipse Modeling Framework) [37] and GMF (Graphical Modeling Framework) that we are using. The domain specific modeling language *HPCML*, introduced in in Section 4, is implemented based on EMF, while its graphical view is based on GMF.

Archi-MDE integrates two existing plugins that are used to edit, compile and debug the generated source files:

- ***CDT*:** Eclipse C/C++ Development Tooling [38] provides a fully functional C and C++ Integrated Development Environment (IDE)
- ***Photran*:** Photran [39] is an IDE and refactoring tool for Fortran based on top of the CDT plugiun.

The information on the target platform is injected during the model to text transformation, corresponding to the code generation that is performed using the Xpand engine [40]. Archi-MDE uses aspect-oriented programming techniques present in the Xpand templates. The ATL project [41] for M2M transformations seem to be well suited to our needs for the moment.

As the final goal of this process is code generation, Archi-MDE produces code for Arcane [42]. Arcane is a software development framework for 2D and 3D numerical simulation codes used by the CEA and the IFP (French Institute of Petroleum). It already offers a fair abstraction level of the underlying hardware

which is close to our task and data models (see PIM definition from Section 4) and provides utility services that we had no interest in developing again for the sake of the demonstration.

6 Conclusion and Future Work

Hardware architectures are evolving fast. Unfortunately present applications do not possess the required qualities to easily adapt to these frequent evolutions while maintaining optimal performances. However, there is an increasing demand advocating the need to add abstraction [43] in dealing with parallel architectures or to perform a more radical paradigm shift [44] in developing HPC software.

The main contribution of this paper is to propose a new way to deal with these problems by using MDE techniques for the development of HPC applications. We present the potential benefits of such approach, that include the more flexibility for maintenance and code porting towards new hardware architectures.

MDE4HPC, the methodology presented in this paper, is ready to offer better separation of concerns and knowledge capitalization. These points are a pillar of our approach by allowing people to focus and contribute fully on their area of specialization. We also presented Archi-MDE a user-supportive integrated development environment to sustain the methodology.

Our experiments on applications with few thousand lines of code are very encouraging and made us confident in the feasibility of this approach and in the added value for CEA simulation code development. Based on these positive feedbacks from the first version of Archi-MDE, we intend to deploy and assess our approach on bigger size projects within the CEA. Moreover, we are about to improve the additional services of Archi-MDE to draw maximum advantage from the use of models. Model verifications, symbolic execution and reverse engineering of legacy code are the next steps to complete the assessment of our approach.

Obviously, a lots of things have still to be done, such as applying MDE4HPC on real-size projects, identifying relevant benchmarks that would allow to quantify the gains of applying this technique, and identifying and implementing more functionalities so that Archi-MDE can exploit the use of models. The preliminary results make us very confident both in the fact that MDA can be applied for the development of HPC applications, and that it can bring serious benefits to it.

References

1. Moore, G.E.: Cramming more components onto integrated circuits. Electronics 38(8), 114–117 (1965)
2. Larus, J.R.: Spending moore's dividend. Commun. ACM 52(5), 62–69 (2009)
3. Ross, P.E.: Engineering: 5 commandments. IEEE Spectrum 40(12), 30–35 (2003)
4. Asanovic, K., Bodík, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D.A., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K.A.: A view of the parallel computing landscape. Communications of ACM 52(10), 56–67 (2009)

5. Miller, J., Mukerji, J.: Mda guide version 1.0.1. omg/2003-06-01. Technical report, OMG (2003)
6. Kirk, D.: Nvidia cuda software and gpu parallel computing architecture. In: ISMM, pp. 103–104 (2007)
7. Bayoumi, A.M., Chu, M., Hanafy, Y.Y., Harrell, P., Refai-Ahmed, G.: Scientific and engineering computing using ati stream technology. Computing in Science and Engineering 11(6), 92–97 (2009)
8. Johns, C.R., Brokenshire, D.A.: Introduction to the cell broadband engine architecture. IBM Journal of Research and Development 51(5), 503–520 (2007)
9. Gonnord, J., Leca, P., Robin, F.: Au delà de 50 mille milliards d´opérations par seconde? La Recherche (393) (January 2006)
10. Miller, G.A.: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review 63, 81–97 (1956)
11. Cea - The Simulation Program, `http://www.cea.fr/english_portal/defense/the_simulation_program_lmj_tera_airix`
12. Software Process Engineering Meta-Model, version 2.0. Technical report, Object Management Group (2008)
13. Bernholdt, D.E.: Raising the level of programming abstraction in scalable programming models. In: IEEE International Conference on High Performance Computer Architecture (HPCA), Workshop on Productivity and Performance in High-End Computing (P-PHEC), pp. 76–84. IEEE Computer Society, Los Alamitos (2004)
14. Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J.: MPI: The complete reference. MIT Press, Cambridge (1996)
15. Dagum, L., Menon, R.: Openmp: An industry-standard api for shared-memory programming. Computing in Science and Engineering 5, 46–55 (1998)
16. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (2004)
17. Baumgartner, G., Bernholdt, D.E., Cociorva, D., Harrison, R., Hirata, S., Lam, C.-C., Nooijen, M., Pitzer, R., Ramanujam, J., Sadayappan, P.: A high-level approach to synthesis of high-performance codes for quantum chemistry. In: Supercomputing 2002: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, pp. 1–10. IEEE Computer Society Press, Los Alamitos (2002)
18. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 863–874. Springer, Heidelberg (2009)
19. Bodin, F.: Keynote: Compilers in the manycore era. In: Seznec, A., Emer, J., O'Boyle, M., Martonosi, M., Ungerer, T. (eds.) HiPEAC 2009. LNCS, vol. 5409, pp. 2–3. Springer, Heidelberg (2009)
20. Bézivin, J., Gerbé, O.: Towards a precise definition of the OMG/MDA framework. In: ASE, pp. 273–280. IEEE Computer Press, Los Alamitos (2001)
21. Douglass, B.P.: Real Time UML: Advances in the UML for Real-Time Systems, 3rd edn. Addison Wesley Longman Publishing Co., Inc., Redwood City (2004)
22. Thomas, F., Gérard, S., Delatour, J., Terrier, F.: Software real-time resource modeling. In: FDL, pp. 231–236. ECSI (2007)
23. Reed, R.: Itu-t system design languages (sdl). Computer Networks 42(3), 283–284 (2003)
24. Lockheed Martin (MDA success story), `http://www.omg.org/mda/mda_files/lockheedmartin.pdf`

25. Objectsecurity: Helping to secure the friendly skies, http://www.omg.org/mda/mda_files/objectsecurity_final.pdf
26. Pllana, S., Benkner, S., Mehofer, E., Natvig, L., Xhafa, F.: Towards an intelligent environment for programming multi-core computing systems, pp. 141–151 (2009)
27. Taillard, J., Guyomarc'h, F., Dekeyser, J.-L.: A graphical framework for high performance computing using an mde approach. In: Euromicro Conference on Parallel, Distributed, and Network-Based Processing, pp. 165–173 (2008)
28. Banerjee, U., Eigenmann, R., Nicolau, A., Padua, D.A.: Automatic program parallelization. Proceedings of the IEEE 81(2), 211–243 (1993)
29. Lugato, D.: Model-driven engineering for high-performance computing applications. In: The 19th IASTED International Conference on Modelling and Simulation, Quebec City, Quebec, Canada (May 2008)
30. Lugato, D., Bigot, C., Valot, Y., Gallois, J.-P., Gérard, S., Terrier, F.: Validation and automatic test generation on UML models: the AGATHA approach. International Journal on Software Tools for Technology Transfer (STTT) 5(2), 124–139 (2004)
31. Metcalf, M.: FORTRAN Optimization. Academic Press, Inc., Orlando (1985)
32. Panas, T., Quinlan, D., Vuduc, R.: Tool support for inspecting the code quality of hpc applications. In: Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing Applications, SE-HPC 2007, p. 2. IEEE Computer Society, Washington, DC (2007)
33. Mohagheghi, P., Dehlen, V.: Existing model metrics and relations to model quality. In: ICSE Workshop on Software Quality, WOSQ 2009, pp. 39–45 (May 2009)
34. Grobelny, E., Bueno, D., Troxel, I., George, Vetter, J.S.: FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications. Simulation 83(10), 721–745 (2007)
35. Brown, W.J., Malveau, R.C., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley, Chichester (1998)
36. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, Reading (2009)
37. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, Reading (2009)
38. Eclipse C/C++ Development Tooling - CDT, http://www.eclipse.org/cdt/
39. Photran - an integrated development environment and refactoring tool for Fortran, http://www.eclipse.org/photran/
40. Xpand, http://wiki.eclipse.org/xpand
41. ATL : ATL Transformation Language, http://www.eclipse.org/atl/
42. Grospellier, G., Lelandais, B.: The arcane development framework. In: POOSC 2009: Proceedings of the 8th Workshop on Parallel/High-Performance Object-Oriented Scientific Computing, pp. 1–11. ACM, New York (2009)
43. Kulkarni, M., Pingali, K., Walter, B., Ramanarayanan, G., Bala, K., Chew, L.P.: Optimistic parallelism requires abstractions. Communications of ACM 52(9), 89–97 (2009)
44. Harrison, R.J.: The myriad costs of complexity will force a paradigm shift. In: Community Input on the Future of High-Performance Computing Workshop (December 2009), http://www.nics.tennessee.edu/sites/www.nics.tennessee.edu/files/NSF-HPC-Whitepaper-12-09.pdf