A Coordination-based Model-Driven Method for Parallel Application Development

S. Gudenkauf[1]

OFFIS Institute for Information Technology, R&D-Division Energy, Escherweg 2, 26121 Oldenburg, Germany
email: `stefan.gudenkauf@offis.de`

Abstract. A continuous trend in computing is the demand for increasing computing performance. With the advent of multicore processors in the consumer market, parallel systems moved out of the scientific niche and became commodity. This raises the need to exploit concurrency in software of all kinds and domains. Unfortunately, the majority of software developers today are short on parallel programming experience, and at least in the near future tools and techniques will not be able to fully exploit concurrency in application development automatically.
In this position paper we propose to regard software architectures of parallel systems as the main development artifact, focusing on the behavioral view and top-down application development. To address the need for higher abstractions and to facilitate reuse, we propose a model-driven software development approach based on a visual domain-specific language that hierarchically separates coordination from computation.

1 Introduction

To continue to improve processor performance, companies such as Intel and AMD turned to hyperthreading and multicore architectures since physical limitations impede further performance gains that base on increasing clock speed and optimizing execution flow [1]. These new performance drivers require to explicitly consider concurrency to exploit them in total. Unfortunately, after years of sequential programming practice the majority of software developers today are short on parallel programming experience, and at least in the near future there will be no tools and techniques to fully exploit concurrency automatically. Also, although being the predominant model for general-purpose parallel programming, the threading model makes parallel programming harder than it has to be because it is enormously nondeterministic and requires software developers to cut away unwanted nondeterminism [2].

Concurrency has now to be exploited in applications of all kinds and domains, especially regarding complex large-scale systems with a variety of associated roles and stakeholders. The challenge is not solely software performance and speedup, but also to provide a convenient way to participate in the new performance drivers in general, and to facilitate reuse and portability. Especially the latter aspects may turn out to be of major importance in parallel programming in the large because of the high development costs of (re-)developing failure-safe parallel software. In the following, we present related work in Sec. 2, motivating the necessity of high level abstractions for parallel program development. In Sec. 3 we present our model-driven solution that is primarily targeted on abstraction from the threading model, and the expected contributions. Finally, we describe our plans for evaluation in Sec. 4 and conclude the paper in Sec. 5.

2 Related work

Lee argues that most of the difficulties in parallel programming are a consequence of our concurrency abstractions [2]. He shows that the threading model, although being a minor syntactical extension to existing languages, implies severe consequences to programming since it is enormously nondeterministic and demands to extensively cut away unwanted nondeterminism. He argues that coordination languages provide a solution to challenge concurrency, since they are orthogonal to established programming languages and facilitate to regard nondeterminism explicitly in an otherwise deterministic problem solution. Pankratius et al. present a case study on parallelizing the open source compression program BZip2 for multicore systems. [3]. At least in the context of this study, it is shown that considerable speedup can be gained by exploiting concurrency on higher abstraction levels, and that parallel patterns turned out to be more significant to speedup than fine-grained loop parallelization. It is also noted that industry approaches often propagate the feasibility of inserting parallelization constructs in existing sequential code, thus limiting the amount of exploitable concurrency.

There are few approaches in model-driven parallel program development. IBM alphaWorks provides a tool that generates parallel code from UML models and supports concurrent patterns for multicore environments [4]. Using the tool involves different activities such as the creation of concurrency patterns by pattern developers and serial computing kernels by C++ developers. Although promising, there is very few information available and the current status of the project is to the best of our knowledge unclear. Pllana et al. [5] propose an intelligent programming environment that targets multi-core systems and proactively supports a user in program composition, design space exploration, and resource usage optimization. This environment is envisioned to combine model-driven development with software agents and high-level parallel building blocks to automatize, for example, time-consuming tasks such as performance tuning. Although proposing to use UML extension for graphical program composition, the work falls short on describing the actual method and DSL to be used.

3 Solution approach

In addition to [3] and [2], we argue that for the broad mass of software developers, large-scale parallel systems development requires appropriate high-level abstractions and methods to cope with complexity and scale.

Solution. We propose a model-driven method that regards the architectures of parallel programs as the main development artifacts, based upon the following aspects: (1) Top-down problem decomposition is facilitated. (2) Nondeterminism is introduced when needed instead of being cut away when not needed. (3) Mapping high-level units of execution to low-level processing entities is left for model-driven development. (4) The development of large-scale parallel programs of all kinds and domains is facilitated. (5) Ordinary software developers are supported in developing parallel applications with considerable effort.

Contributions. (1) *DSL.* A domain-specific language that abstracts from the underlying low-level technologies, particularly, from the threading model. (2) *Method.* An appropriate method to employ the DSL to model the behavioral view of parallel software architectures, resulting in formally specified models as a basis for model driven development. (3) *Tooling.* Tool support that integrates with existing technologies and complementary architectural views.

Hypotheses. (1) Regarding parallel software architectures as the main development artifact facilitates reuse. (2) Hierarchically separating coordination from computation on all levels of software development facilitates the development of large-scale parallel software systems. (3) Focusing parallel software architectures encourages using parallel programming patterns at the highest level of abstraction. (4) The ordinary software developer is short on parallel programming expertise. (5) There will be the need to develop large numbers of parallel software systems of all kinds and domains. (6) For many software development projects, the goal will not be ultimate speedup but speedup and scalability as properties among others (e.g., maintainability, portability, understandability).

Domain analysis. We define our target domain as parallel systems software architecture engineering. The archetypical stakeholder roles are system/domain architects, application domain experts, customers, and software developers. Regarding the software development stages construction, debugging, and performance tuning (cp. [6]), we focus on program construction, see Fig. 1. We further focus on the behavioral view, regarding the behavior of a software system as concurrent processes that act with or upon data, where each process is itself deterministic. Stakeholders are thus encouraged to decompose the problem to be solved in terms of processes that consists of sub-processes and tasks to be performed sequentially or in parallel, and control flows between them. Although emphasizing control flows, data flows and objects are also parts of the view. Fig. 2 present an appropriate feature diagram.

DSL. Developing parallel applications using traditional programming languages can be very tedious and error-prone due to the linearity of textual source code. Visual DSLs are multi-dimensional, thus able to present multiple concurrent control flows naturally, while fine-grained concurrency control may be encapsulated in appropriate language feature semantics [7], [8]. We regard the threading model as the primary technology underlying our (therefore horizontal) DSL. Orthogonality to existing programming languages is also considered to provide an understandable large-scale overview of program structure (cp. [7]). We require the DSL to be visual (RQ-1) and graph-based (RQ-2). Language con-

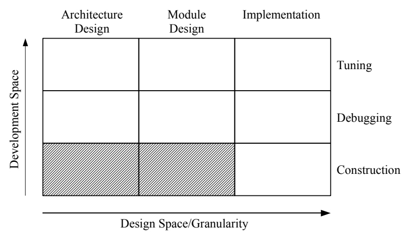

Fig. 1. Target domain context.

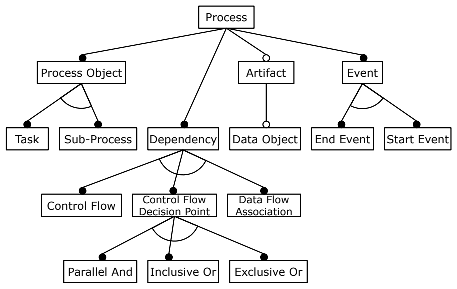

Fig. 2. Feature diagram.

structs must conform to domain concepts, ideally providing distinct constructs for each distinct concept (RQ-3). Also, the vocabulary of the language shall be as small as possible (RQ-4) and the constructs shall facilitate model quality (RQ-5). The language must be scalable (RQ-6) and hierarchically composable (RQ-7, cp. Fig. 3). Concurrency must be expressed explicitly as the coordination of tasks and (sub-) processes to separate coordination from computation (RQ-8). Thereby, tasks represent computation and sub-processes represent further compositions of coordinated tasks and sub-processes. Both should be instantiable to represent concurrent execution of the same computation (RQ-9). Also, the language should be control-driven [9] (RQ-10).

Method. System architects use the proposed DSL to construct a model for the behavioral view of the architecture of parallel systems. From this model, there may be subsequent model-to-model transformations, before code is generated that represents the executional framework of the parallel system to be developed. These transformations are created by transformation developers (special domain architects), while functional implementation is left for complementing modeling stages or manual implementation. This method scenario is presented in Fig. 4. The benefits of the method are: (1) *Knowledge capture.* Architecture models provide a basis for communication between domain experts, system architects, and software developers. (2) *Reuse and portability.* Reference models and transformations can be reused, providing a basis for software product lining; different target platform transformation sets can be applied to the same architecture model. (3) *Quality.* Model bugs, as well as the respective responsibilities, are separated from implementation bugs – the former having to be corrected only once in the transformation descriptions instead of multiple times in the source code. (4) *Information Hiding.* Transformations encapsulate platform-specific implementation.(5) *Development time reduction.* Reusing models and transformations saves development time.

4 Plans for Evaluation and Future Work

Our plans on evaluation are as follows: ***Domain model***. To provide a refined domain model, relevant concepts, their (shared and differentiating) features,

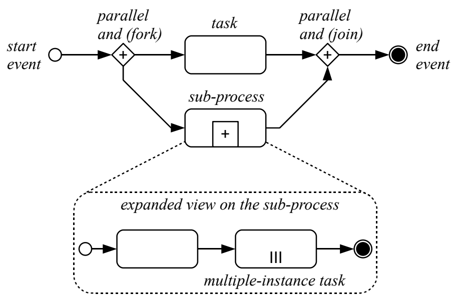

Fig. 3. DSL example.

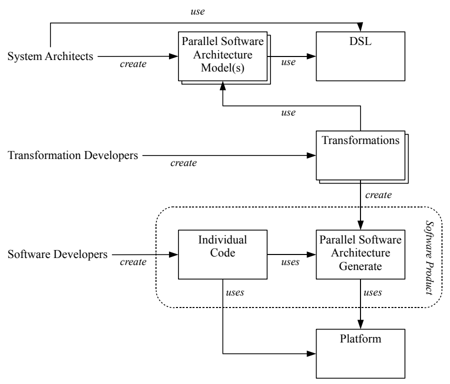

Fig. 4. Method scenario.

and additional requirements have to be further identified. This can be done by identifying and analyzing reference applications for repetitive patterns [8] ***Case studies***. We intend to perform student case studies on developing an exemplary parallel application using the proposed method, focussing on speedup. Possible study scenarios are (a) a scenario in which the example application is developed using the target platform technologies directly, and (b), a scenario where the proposed method is used for architecture modeling, and subsequent implementation of domain-specific functional aspects. Thereby, the case studies have to be carefully designed regarding, for example, knowledge level, learning effects, and favouritism. Also, the exemplary application has to be selected carefully considering, for example, source code and documentation availability, implementation language, application size, algorithm complexity, and estimated concurrency (cp. [3]). A possible application candidate currently regarded is the Desmo-J[1] discrete-event simulation framework. A possible target platform is Java since it is widely used in industry, supposed to be the first exposure to parallel programming for many programmers, and provides JVM-supported low-level thread management.

Our future work includes: (1) ***Language research***. There are a number of languages that can be regarded as coordination languages, and target to reduce complexity by representing complex control-flow behavior of parallel programs graphically, for example, discussed in[7], [9], and [10]. We will examine these languages and their features for applicability to the envisioned DSL. We also suppose the Business Process Modeling Notation (BPMN) to meet many of the presented DSL requirements. Since it is well-known in the business workflow domain it may provide a basis for the visual DSL representation with prospect of broad dissemination, provided that the underlying semantics of the elements remain fundamentally intact. (2) ***DSL Extension***. We will examine the possibility to extend the DSL to also abstract from message passing, thus covering

[1] http://desmoj.sourceforge.net/

the two most important parallel programming technologies. This may require to extend the DSL by constructs for logical process locality. (3) ***Tool development***. We intend to support the method by appropriate tooling. This includes a modeling environment that may base upon the Eclipse Modeling Tools[2] and the openArchitectureWare MDA/MDD generator framework[3].

5 Conclusion

In this position paper we discussed the need for higher abstractions in parallel software development. This need is motivated by the inappropriateness of the threading model since it requires to tame nondeterminism, the lack of parallel programming experience, and the supposed impact of higher-level abstractions on application performance. To satisfy this need, we proposed a model-driven method that regards the architectures of parallel programs as the main development artifact, and an adequate visual domain-specific language. Thereby, we focused on the behavioral view, on top-down problem decomposition, and on the controlled use of nondeterminism.

References

1. Sutter, H.: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal **30** (2005)
2. Lee, E.A.: The Problem with Threads. IEEE Computer **39** (2006) 33–42
3. Pankratius, V., Jannesari, A., Tichy, W.F.: Parallelizing BZip2. A Case Study in Multicore Software Engineering. Technical report (2008)
4. IBM alphaWorks: Model-Driven Development Tool for Parallel Applications. (http://www.alphaworks.ibm.com/tech/ngspattern) [Date posted: November 1, 2007; last visited: January 16, 2009].
5. Pllana, S., Benkner, S., Mehofer, E., Natvig, L., Xhafa, F.: Towards an Intelligent Environment for Programming Multi-core Computing Systems. In Eduardo, C., Alexander, M., Streit, A., Träff, J.L., Cérin, C., Knüpfer, A., Kranzlmüller, D., Shantenu, J., eds.: Euro-Par 2008 Workshops - Parallel Processing. Volume 5415., Berlin / Heidelberg, Springer (2009) 141–151
6. Zhang, K., Ma, W.: Graphical Assistance in Parallel Program Development. In: Proc. of the 10th IEEE Intl Symp. on VisualVisual Languages, 1994. Proceedings., IEEE Symposium on. (1994) 168–170
7. Browne, J.C., Dongarra, J., Hyder, S.I., Moore, K., Newton, P.: Visual Programming and Parallel Computing. Technical report, Knoxville, TN, USA (1994)
8. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional (2008)
9. Papadopoulos, G.A., Arbab, F.: Coordination Models and Languages. In: Advances in Computers, Academic Press (1998) 329–400
10. Lee, P.A., Webber, J.: Taxonomy for Visual Parallel Programming Languages. Technical report, School of Computing Science, University of Newcastle upon Tyne (2003) Technical Report CS-TR-793.

[2] http://www.eclipse.org
[3] http://www.openarchitectureware.org/