OPENMP CODE GENERATION BASED ON A MODEL DRIVEN ENGINEERING APPROACH*

Julien Taillard
LIFL / INRIA
Email: Julien.Taillard@lifl.fr

Frédéric Guyomarc'h
University of Rennes / INRIA
Email: Frederic.Guyomarch@inria.fr

Jean-Luc Dekeyser
LIFL / INRIA
Email: Jean-Luc.Dekeyser@lifl.fr

KEYWORDS

MDE, HPC, Code generation, OpenMP

ABSTRACT

In this paper, we present a methodology which allows OpenMP code generation and makes the design of parallel applications easier. The methodology is based on the Model Driven Engineering (MDE) approach. Starting from UML models at a high abstraction level, OpenMP code is generated through several metamodels which have been defined. Results show that the produced code is competitive with optimized code.

INTRODUCTION

A physical barrier has been attained by processor suppliers which implies that frequency can not be increased like in the past. Previously, a basic processor replacement allowed to gain performance thanks to the higher frequency. Nowadays, the only way to reach performance is to carry out parallel computing and the multi-core processors development is making it even more important.

Unfortunately parallel computing is not easy for non-specialists. It requires knowledge of parallel algorithms and parallel coding for different kinds of architectures. From shared memory to distributed memory, the range of parallel machines is wide and program optimization is machine dependent. The complexity is even increasing now with the mix of all these concepts into multi-core machines joined into a grid.

Higher levels of parallel languages (like Fortress (2)) are made to simplify writing, but they are still for specialists. Methods to help the non-specialists to write parallel code must be defined.

The trend in software engineering is to model software at a high abstraction level in order to be more productive and to make the software durable. A high abstraction level enables to be independent from any language and the designers do not have to handle all implementation details. In the Model Driven Engineering (MDE) conception flow, input models (expressed at a very high level) are transformed into lower abstraction levels to finally generate code.

Our contribution is twofold. First, we propose a metamodel to model OpenMP programs that could be used in any model approach. This metamodel is based on a metamodel of procedural language wherein OpenMP concepts have been added. Secondly, we present a way to generate OpenMP programs starting from a high abstraction model using the OpenMP metamodel.

The paper is organized as follows. The next section presents some works about the automatic generation of parallel code. Then, the main concepts of the MDE are introduced. Later, an approach based on the MDE to produce OpenMP code is presented. Afterwards, some results about the produced code is analyzed. Finally, conclusions are given and some on-going works are presented.

RELATED WORK

The automatic parallelization is a widely studied domain. Lots of tools are available to handle it. Two approaches can be distinguished: the generation of parallel code and the parallelization during the compilation. Since our goal is to generate OpenMP code, we describe here only a few works closely related, which also deal with parallelization using OpenMP.

The classical approach is to generate OpenMP code starting from a sequential code. Tools such as CAPO (11; 10) and the POST project (1) are based on this approach. Starting from a sequential code, and through a data dependencies analysis, the tools will automatically identify the main characteristics of the code including the different types of loops. Then, there are two approaches: either the code is automatically generated with the insertion of the appropriate OpenMP directives or the tool indicates the user which loops can be parallel and helps him to decide.

Another approach is to generate OpenMP code from another parallel language. Thus, Krawezik et al. (13) propose to generate OpenMP code starting from MPI code. The resulting program is a program written in the same style as the MPI code: the *Single Program Multiple Data* (SPMD) style. Such approach is used because OpenMP is often better on shared memory architecture than MPI.

Our approach promotes the use of visual programming (the models), thus users don't need to handle code directly. As the models are made at a very high abstraction level, they can be reused for any other parallel languages. Code generation is made automatically without any user's intervention.

*This work has been partially supported by the CNRT FuturElec and the CNRS PEPS program

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

MODEL DRIVEN ENGINEERING (MDE)

Models are being used since a long time. Painting and sculpture can be considered as models of what they represent (8). They are used in computer science since time now. The first use of models in computer science is to make a system understandable for different developers. It is used as documentation and specification for the system developers. These kind of models is named *contemplatives*. In order to raise productivity, the idea is to make models *productive* and *executable*, and the MDE (18) has been introduced. It consists of using models at each level of conception, and to make models transformations between these levels. So that, once the system is modeled at a high abstraction level, only refinements are needed instead of rebuilding all the system at the new level. Such an approach allows to be independent of implementation details since they are managed by the transformations.

Model

A model is an abstraction of the reality. It is composed by concepts and relations in order to represent the system. When a designer models a system, he thinks about what he is interested in the system. Models are made with the designer's point of view. The same system could be modeled in different ways depending on the designer.

Models are usually made using the Unified Modeling Language (UML) (14) which is the standard from the Object Management Group (OMG) for visual programming.

In the HPC field, the designers are interested in the parallelism. Models of application and architecture need to express all the available parallelism to take advantage of the high number of processors available.

Metamodel

A metamodel defines the available concepts and relations that can be used to create a model. It could be compared to the grammar of a programming language. Designers are guided by the metamodel in order to produce same kind of models. A model made using a metamodel is said *conform* to this metamodel.

In our framework the metamodels must propose a mechanism to express all the parallelism in the models and also the mapping of software onto hardware architectures. The distribution of tasks on hardware is an important point to raise performances as different mappings of the same application can highly impact these performances.

Transformation

As the MDE promotes the use of models in each level of conception (abstraction level), an automatic generation of the different intermediate models is mandatory. A *model transformation* is performed between a source model and a target model, respectively conformed to the source and target metamodels. It could be seen as a compilation process. A transformation relies on a set of rules: a rule expresses how to transform a source concept (resp. relation) into an equivalent target concept (resp. relation). It facilitates the writing, extensions and maintainability since each rule is independent of others and can then be modified independently. The transformation of models are typically used to reach a low level model (the code) from a high abstraction level model. Each transformation is designed in order to add details to the models. It is a refinement used to be able to obtain the code with the appropriate implementation details. Several approaches of model transformations have been proposed. The Query/View/Transformation (QVT) (15) approach is an OMG standard for transformation. It proposes a declarative and an imperative language to write transformations. Unfortunately transformations tools are not yet mature and no complete implementation of QVT is available. SmartQVT (20) is a partial implementation of QVT based on the declarative language but it was not entirely developed when we started development. So, we developed our transformation tool called MoMoTE (Model to Model Transformation Engine). It is build as an Eclipse plugin using the EMF tools (7).

For the HPC area, the goal is to produce parallel code starting from a high abstraction level. Thus, a language independent model to express parallelism will be transformed into a language dependent model to express the parallelism. The high level expression of parallelism will be refined in order to produce optimized code in the targeted language, here OpenMP (C and Fortran).

METAMODEL OF PROCEDURAL LANGUAGE USING OPENMP

As the targeted languages were Fortran and C, the most efficient strategy was to define a metamodel which allows to generate both languages. Since both are procedural languages, a metamodel of a general procedural language wherein the OpenMP concepts are added has been specified. The metamodel has been inspired by the C-ANSI Yacc grammar (12). An overview of the metamodel of procedural language is first presented and then the introduction of OpenMP statements in the procedural language metamodel is exhibited. Other procedural languages than C and Fortran have not been studied in detail to create the metamodel, but should be compatible with this metamodel with only minor changes.

Procedural language metamodel description

This metamodel aims to be of general-interest. It is not specific for our goal and could be used for any other purposes. It permits to declare programs, libraries, routines, functions, variables, expressions and all the available constructions. Pointers are not yet supported in the metamodel because we do not need it at the language level. In fact, in the higher level, all data are organized as multidimensional arrays which can be natively translated in languages we target.

The figure 1 presents an excerpt of the statements

available in the procedural language. The most common expressions of languages can be modeled: the conditional statements (*if* and *ifelse*), the loop statements (*for* and *while*), the *call* of subroutines/functions and the construction of *expressions*.

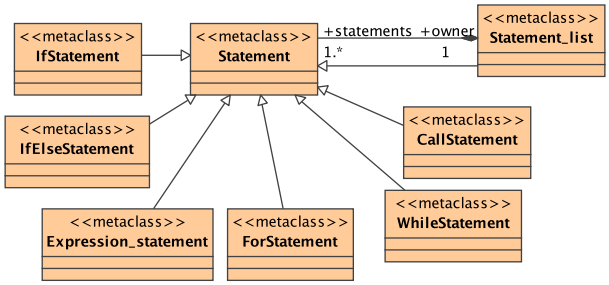

Figure 1: Statements in the procedural language metamodel

Adding OpenMP statement to procedural language metamodel

As the goal is to generate OpenMP code, OpenMP statements have to be available to generate OpenMP directives and function calls. OpenMP statements are added to the metamodel while OpenMP functions, like *omp_get_num_thread()*, are modeled in a library which is referenced when OpenMP procedural language model makes a function call.

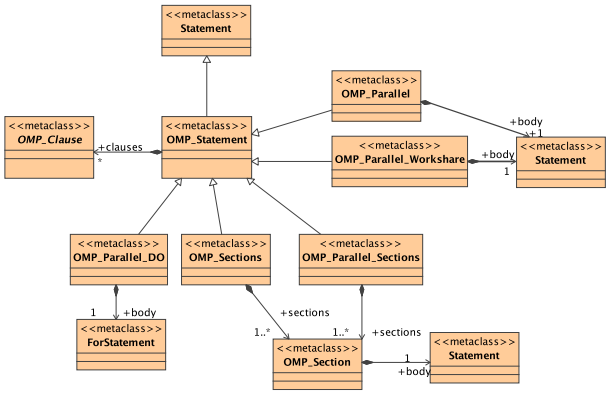

Figure 2: OpenMP statement added

The figure 2 illustrates the OpenMP statements added in the classical statement. An OpenMP statement can have *OMP_Clause* (such as *private* or *shared)*.

The figure 3 illustrates a model made with the metamodel (as a graphical representation is not generated with the metamodel). In this example, we illustrate the tree organization of a model: an OMP Parallel statement contains an ordered list of statements (S1,S2,S3) and two synchronization barriers (B). Each statement is also hierarchic: S2 contains an *if* statement.

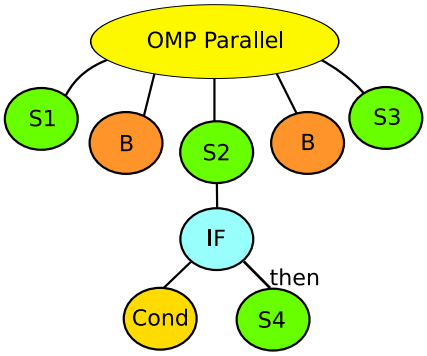

Figure 3: Tree representation of a model

OPENMP CODE GENERATION

The OpenMP code generation is implemented in a framework called Gaspard2 (21). It provides an Integrated Development Environment (IDE) for Multiprocessor System-On-Chip (MpSOC) co-modeling and for High Performance Computing application design (figure 4). Gaspard2 is able to manage both targets because the modeled systems are the same: MpSOC are massively parallel architectures and so are computers used for HPC, targeted applications are parallel in both cases and mapping of application on hardware architecture also needs to be expressed. Thus Gaspard2 has different targets (*Synchronous, SystemC, VHDL, OpenMP*) starting from the same high level model. The *Synchronous* target allows to do verification and validation of an application with the help of synchronous language. The *SystemC* target is able to make System On Chip co-simulation with SystemC. The *VHDL* one allows the generation of an hardware accelerator on a FPGA for a part of the application. The OpenMP target generates shared memory code executable on supercomputers. In the following, we focus on this later.

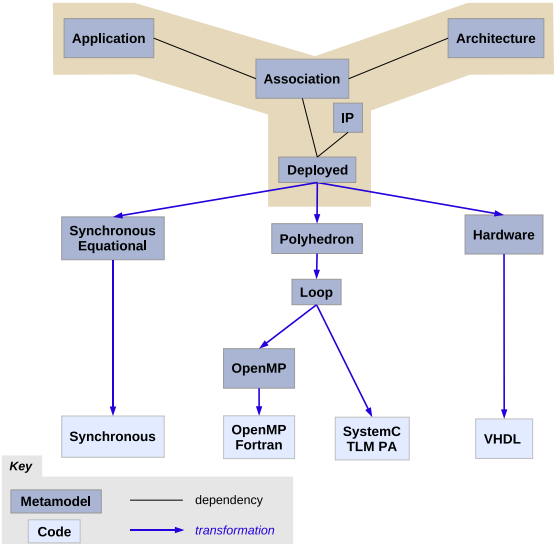

Figure 4: The Gaspard Y Chart

Gaspard2 is based on the component assembly in order to make components reusable. As a component, we define an elementary or composed system with fixed size input and output ports (the data). There is no state in a component, output values (on output ports) depend only on input values. The basic units of models are *Elementary Components* which are deployed to a function (usually from a library), they can be compared to a *black box* associated to a piece of code.

The global methodology, as seen in figure 4, is as follows: the first step is to design application and hardware architecture independently of each other. This separation of concepts allows to reuse these models and to keep them human readable. Then, with the help of an allocation mechanism, the computation tasks are distributed over processors and the data are mapped onto memories[1]. After that, a transformation chain (a succession of models transformations) leads to the generation of the desired code.

This MDE approach permits a great flexibility: targets can be added, reusing a part of the actual chains. Moreover, it encourages reuse of models and components. Once a component has been designed, it could be reused in several models. An application could also be reused to target several languages and hardware architectures. Thus, to target a new hardware architecture, users have to do the association between application model and hardware architecture. Code will be automatically generated for this architecture with the specified mapping. Similarly, several distributions of an application on an architecture could be tested easily. User just has to change the distribution specification, the new code , with the new distribution, will be automatically generated.

The integration of a new standard of OpenMP, such as the future version 3.0 (17), could also be easily done since user just needs to modify the last transformation generating the code itself, and also maybe between the loop and the OpenMP model (and add the new OpenMP features in the corresponding metamodel). In fact the transformations before these are mainly used to express the parallel loops. Concept will always be present in data-parallel languages.

OpenMP code is generated in the SPMD style. SPMD style is one of the most efficient styles compared to the classical loop level one (13) because there are less OpenMP directives since all the code is in a `PARALLEL` directive. As the mapping of application tasks on threads is explicit (using the number of threads), data locality can be assured whereas it cannot be with OpenMP directives which depends on the OpenMP compiler efficiency. In the code generation, a thread is considered to be associated to a processor in order to use the cache efficiently.

[1]for OpenMP, we do not place data onto different memories but for other targets of code like *SystemC*, we could simulate the effect of different placements

The high level model

The high level models are made with a profile (4) which is a subset of the UML Modeling and Analysis of Real-Time and Embedded systems (MARTE) profile (16; 19). A profile allows to add semantics to UML elements. This profile allows the modeling of software, hardware architecture and the association between them. It is a component based approach using UML 2 (14) components. Using a concept called *repetitive structure modeling*, the profile allows to express the repetitions of the same component instance (details could be found in (6)). This concept could be used in software as well as in hardware. It is based on the Array-Oriented Language (Array-OL) expression (5) which is a specification language to express all the parallelism available. Data-parallelism and task-parallelism could be modeled in the application model and available parallelism in the hardware architecture model is also expressed.

An application model can be seen as a hierarchical Directed Acyclic Graph (DAG) of tasks where nodes are tasks and edges are data dependencies between tasks. An application model is illustrated in figure 5. This is a simple application which consists of the initialization of two matrices, making the multiplication between both and checking the result obtained.

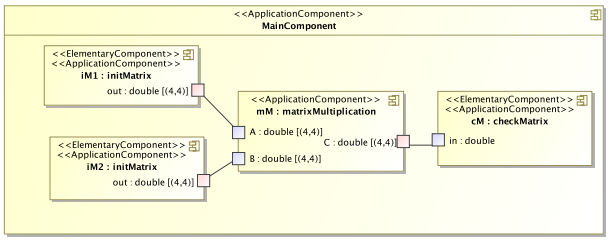

Figure 5: A simple application model

The task parallelism is expressed by several tasks without data dependencies between them. Then, if all the inputs data of the tasks have already been produced, all the tasks can be executed in parallel. Thus, in the figure 5, tasks *iM1* and *iM2* can be executed in parallel: there is no data dependency between them. A model which expresses data-parallelism is shown in figure 6. It represents a matrix multiplication, used in the application model of figure 5, using dot product algorithm. The *dP* instance is repeated (4,4) times to compute the (4,4) output matrix. There is no dependency between each repetition of the *dP* instance. This means the 16 computations can be done in parallel. The connectors stereotyped as *Tiler* express which part of the input/output arrays each repetition uses. Detailed information about *Tiler* can be found in (5).

Tasks distribution over processors is expressed using Array-OL expression. It expresses the distribution of the repetition of tasks over the repetition of processors (see details in (6)). Classical High Performance Fortran distributions such as `BLOCK` or `CYCLIC` distribution can easily be expressed.

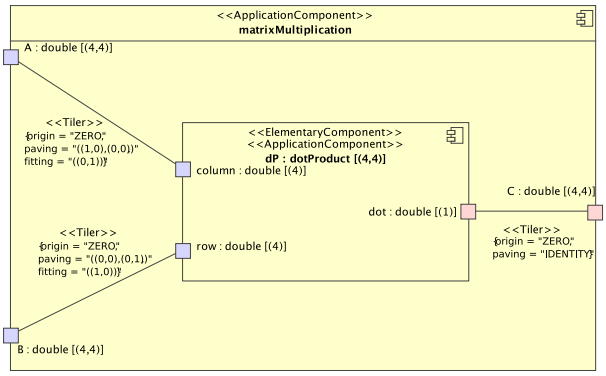

Figure 6: Matrix multiplication expressed with data-parallelism

Only "complete" models can be used for code generation. A "complete" model is an application model associated with an hardware architecture. To target OpenMP, each elementary component needs to be deployed on a function and each instance of elementary component has to be mapped onto a processor (directly or one component instance it belongs to using the hierarchy tree).

Brief explanation from the high level to loop model
The transformation starts from a "complete" high level model. Transformations are decomposed into small ones which are easier to develop, debug and can be reused for several targets. Both transformations briefly explained here are used for the OpenMP code generation as well as for the SystemC code generation.

The first step (high level model to polyhedron model) is to merge high level models into one model: the polyhedron one. During this transformation the expression of the tasks distribution over processors is transformed into the polyhedral model. This is a classical model in the data parallelism area. Considering that the repetition of a task can be seen as a multidimensional polyhedron, the mapping expression defines how to scan this polyhedron depending on the processor number (or the thread number in OpenMP). A polyhedron is generated for each mapping information and is given to the concerned tasks which are linked with the processor (or repetition of processors).

Once polyhedron model is obtained, the second step is to get closer to the implementation. Polyhedrons are transformed into loops using a tool called CLooG (3; Chunky Loop Generator) which generates loops scanning the polyhedrons. In the application part of the model, each polyhedron will be replaced by a nested loop scanning the polyhedron. The resulting application model is still a DAG where the mapping of tasks is expressed in loop expressions.

Generation of OpenMP procedural language model
This is the last transformation of the chain. The goal is to obtain a model in OpenMP procedural language metamodel starting from a loop model. Only the application is used until now, but hardware could be used to optimized code depending on the architecture.

Different tasks have to be done from this model to generate OpenMP code, they are:

- task scheduling
- generate OpenMP directives
- allocate variable
- determine shared variables
- put synchronization barrier

As we generate SPMD code style, all the generated code, which depends on thread number, is included in an OpenMP parallel section. The allocation and affectation of the thread number (p0) is also done automatically.Variables are private by default and shared variables have to be declared.

The transformation is made as follow: transformation rules first analyze the components at the highest hierarchical level, afterward each of these components is analyzed again to determine the sub-levels of component hierarchy.

The mapping between application tasks and processors expresses where each task or repetition of tasks will be executed but it does not express an execution order. Therefore a scheduling of the tasks of the graph is needed in order to produce a valid application. A scheduling is determined for each level of the hierarchy. The basic rule to determine the scheduling is: a task can be scheduled only if all its inputs have already been produced. Once a component can be scheduled, the transformation analyzes the sub-level of the component hierarchy.

The model in the OpenMP procedural language metamodel contains variables which have to be allocated. As we deal with shared memory architecture, data placement are not taken into account but data needs to be allocated and declared shared or private between threads. The basic way is to declare each port as variable, but this causes lot of variable allocation and worse, many useless memory copies penalizing performance. Ports which connect a sub-task to its containing task can be discarded: we just need to know where to read the data into the memory (they are sub-values of the superior port actually), instead of copying the data in a private variable. A variable is identified as shared variable and put on the `SHARED` clause declaration when it is used on tasks mapped on several threads. Variables names depend on the port name and a generated number is used to assure the unicity of variables.

When the transformation deals with an *elementary component*, it generates a call to the function on which the component is deployed. Tilers are computed or optimized in order to reduce the number of intermediary variables.

Insertion of synchronization barrier is needed in order to respect the data dependencies between parallel tasks.

The determination of synchronization barrier is done on the same principle. A synchronization is needed when tasks with data dependencies are not in the same thread (this could be determined with analysis of the loops). Once a hierarchical level has been mapped on a thread (as this is a sequential part), no synchronization are needed and variables used in this part should be private.

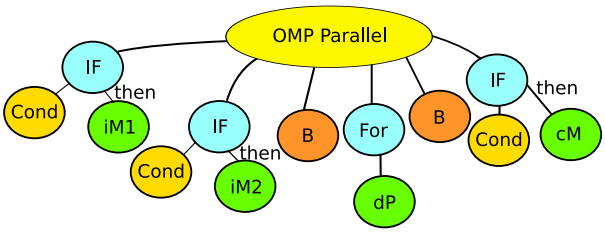

Figure 7: Tree representation of the OMP model

Based on a tree representation of the model, the generated model for the example given on figure 5 will look like in figure 7. It illustrates the OpenMP procedural language model generated for the application mapped on four processors. The mapping expressed that the *iM1* instance is placed on processor 0, the *iM2* instance is placed on processor 1 and the *cM* instance is placed on the processor 0. The distribution of the (4,4) repetitions of the *dP* instance expresses that each processor has to compute a column of computation.

We can observe that two synchronization barriers have been generated: one before the matrix multiplication (threads are waiting at the end of the initialization) and one after the matrix multiplication (waiting for all threads to finish the tasks they have to compute). The *if* statements are generated by the mapping of a single task on a single processor whereas the *for* statement is generated to distribute repetition of the (4,4) *dP* instance on the processor.

Code generation from OpenMP procedural language model

Since the model of lowest level is actually very close to the code itself (at least in structure), the code generation from the OpenMP procedural language model is nothing but a "pretty printer"; it translates the model representing the code into the code itself using templates.

The figure 8 illustrates the code generated from the model presented in figure 7. *Elementary components* are deployed on a routine corresponding to the component name (initMatrix routine for the initMatrix component).

RESULTS

In order to illustrate the use and the efficiency of such an approach, we have compared the execution time of different implementations of a classical program: the matrix multiplication (with (2000,2000) matrices). We have compared automatically produced code with the hand-written library GotoBLAS (9). Code was executed on a

```
program matrixMultiplication
  double precision, dimension(4,4) :: out1
  double precision, dimension(4,4) :: out2
  double precision, dimension(4,4) :: C
  integer :: p0 ! processor number
  integer :: x
  integer :: y

  !$omp parallel default(private)
  !$shared(out1,out2,C)
    p0 = omp_get_num_thread()
    !init matrices
    if (p0==0) then
      call initMatrix(out1)
    end if
    if (p0==1) then
      call initMatrix(out2)
    end if
    !$omp barrier
    do y=0,3
      x=p0
      call dotProduct(..)
    end do
    !$omp barrier

    if (p0==0) then
      call checkMatrix(C)
    end if
  !$omp end parallel

end program
```

Figure 8: Generated code example

3Ghz bi-Xeon dual core processor, running Linux with a total of 2Gb of shared memory. Parallel programs run over four threads.

Algorithm	Best execution time	Average
Row-column algorithm	0:21.11	0:21.60
Block multiplication	0:12.59	0:13.17
Block multiplication with GotoBLAS task	0:01.25	0:01.26
Parallel GotoBLAS	0:01.03	0:01.05
Sequential GotoBLAS	0:03.34	0:03.39

Figure 9: Square matrices - Execution times on four threads - Average is made on 100 executions

Three codes have been generated through Gaspard2: row-column multiplication, block multiplication and the block multiplication using GotoBLAS tasks on sequential parts. The two first generated algorithms go to the scalar operation (addition and multiplication) where as the last one use GotoBLAS function as soon as we are in a sequential part. Results are given in figure 9. We can observe that both codes which go to scalar operation are not competitive compared to the sequential and parallel hand written code. This is due to the fact that we are not able to optimize sequential code. The block multiplication using GotoBLAS task for sequential part is competitive with the parallel hand-written function. Results show that the best way to use Gaspard2 is to let Gaspard2 manage the parallelism using optimized sequential tasks.

CONCLUSION AND FURTHER WORK

In summary, we have presented the Gaspard2 framework that generates OpenMP code based on an MDE approach. All the transformations have been implemented as an Eclipse plugin. The results show that letting Gaspard2 manage the parallelism using optimized tasks on sequential part is the right way to use the framework. This method permits to generate code with performances closed to the optimized library. Programming at a high abstraction level allows to be language independent and to reuse models for different targeted language. This approach has also been tested on a conjugate gradient solving Maxwell equation and the acceleration is promising.

Several further works can be extracted from this approach. The first one is to optimize the generated OpenMP code to provide a better data locality. The use of private arrays instead of shared arrays should improve the use of cache and raise performance. As the multi-core processors are widely widespread, the second work which should be carried out is to optimized code for multi-core architecture which is not a classical SMP architecture since multi-core architectures share more resources as cache memory level. The last task is to target new hardware architectures such as the Graphical Processing Units which are a special kind of shared memory architecture, and the distributed memory architecture using MPI to manage communications.

REFERENCES

[1] Adhianto, L., Chapman, B., Lancaster, D., and Wolton, I. (2000). Tools for OpenMP Application Development: The POST Project. *Concurrency: Practice and Experience*, 12:1177–1191.

[2] Allen, E., Chase, D., Hallett, J., Luchangco, V., Maessn, J.-W., Ryu, S., Jr., G. L. S., and Tobin-Hochstadt, S. (2007). The Fortress Language Specification Version 1.0 Beta. Technical report, Sun Microsystems, Inc.

[3] Bastoul, C. (2004). Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16, Juan-les-Pins.

[4] Ben Atitallah, R., Boulet, P., Cuccuru, A., Dekeyser, J.-L., Honoré, A., Labbani, O., Le Beux, S., Marquet, P., Piel, E., Taillard, J., and Yu, H. (2007). Gaspard2 uml profile documentation. Technical Report 0342, INRIA.

[5] Boulet, P. (2007). Array-OL revisited, multidimensional intensive signal processing specification. Research Report RR-6113, INRIA.

[6] Boulet, P., Marquet, P., Piel, E., and Taillard, J. (2007). Repetitive Allocation Modeling with MARTE. In *Forum on specification and design languages (FDL'07)*, Barcelona, Spain. Invited Paper.

[Chunky Loop Generator] Chunky Loop Generator. *CLooG home page*. `http://www.cloog.org`.

[7] eclipse.org (2005). Eclipse. `http://www.eclipse.org`.

[8] Favre, J.-M. (2005). Foundations of model (driven) (reverse) engineering : Models – episode i: Stories of the fidus papyrus and of the solarus. In Bezivin, J. and Heckel, R., editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[9] Goto, K. and van de Geijn, R. (2002). On Reducing TLB Misses in Matrix Multiplication. Technical Report TR-2002-55, The University of Texas at Austin, Departement of Computer Sciences. FLAME Working Note #9.

[10] Ierotheou, C. S., Jin, H., Matthews, G., Johnson, S. P., and Hood, R. (2005). Generating OpenMP code using an interactive parallelization environment. *Parallel Computing*, 31(10-12):999–1012.

[11] Jin, H., Frumkin, M., and Yan, J. (2000). Automatic Generation of OpenMP Directives and Its Application to Computational Fluid Dynamics Codes. pages 440–456.

[12] Jutta Degener (1995). Ansi c yacc grammar. URL: `http://www.lysator.liu.se/c/ANSI-C-grammar-y.html`.

[13] Krawezik, G. and Capello, F. (2003). Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 118–127, New York, NY, USA. ACM.

[14] Object Management Group, Inc., editor (2004). *UML 2 Superstructure (Available Specification)*. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`.

[15] Object Management Group, Inc. (2005a). MOF Query / Views / Transformations. `http://www.omg.org/docs/ptc/05-11-01.pdf`. OMG paper.

[16] Object Management Group, Inc., editor (2005b). *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP*. `http://www.omg.org/cgi-bin/doc?realtime/2005-02-06`.

[17] OpenMP Architecture Review Board (2007). OpenMP Application Program Interface Draft 3.0.

[18] Planet MDE (2007). *Model Driven Engineering*. `http://planetmde.org/`.

[19] ProMarte partners (2007). UML Profile for MARTE, Beta 1. `http://www.omg.org/cgi-bin/doc?ptc/2007-08-04`.

[20] SmartQVT (2007). A QVT implementation.

[21] WEST Team LIFL, Lille, France (2005). Graphical Array Specification for Parallel and Distributed Computing (GASPARD-2). `http://www.lifl.fr/west/gaspard/`.

AUTHOR BIOGRAPHIES

JULIEN TAILLARD is a PhD student on computer science. His research interest are models and high performance computing.

FRÉDÉRIC GUYOMARC'H is assistant professor in University of Rennes and currently spends one year as researcher at the INRIA Lille - Nord Europe. He is interested in Linear Algebra and parallel algorithms.

JEAN-LUC DEKEYSER is professor in the Computer Science department at the University of Lille. He is also the scientific leader of the DaRT project at INRIA Lille - Nord Europe and the WEST team at LIFL (Laboratoire d'Informatique Fondamentale de Lille).