Methods and Tools for System on Chip Retargetable Parallel Programming

Alexey Syschikov, Boris Sedov
St-Petersburg University of Aerospace Instrumentation
Saint-Petersburg, Russia
alexey.syschikov@guap.ru, sedov.boris@gmail.com

Abstract

The large number of various systems on chip platforms requires developing many different programs for a single task to fulfill target platform language, architecture, memory and other specifications. Retargetable parallel programming method and tools allows to develop and debug program once and to make it ready for various target platform. The proposed approach is based on the formal model of computation and backend code generators. That will allow easily attaching target platforms to the retargetable parallel programming toolset.

Index Terms: parallel programming, retargeting, system on chip, model of computations.

I. Introduction

"In software engineering, retargeting is an attribute of software development tools that have been specifically designed to generate code for more than one computing platform" [1]. Since 1980-th, there was a variety of processor architectures, such as 68xx (Motorola), ARM, PDP and VAX (DEC), x86 (Intel), SPARC (Sun) etc. And there was a reasonable willing to have universal retargetable compilers, that will allow to generate code for all (or large subset) of architectures from a single source and some retargetable compilers were developed to solve this task.

Nowadays many new processors are developing. Most of them are built using the system on chip (SoC) approach with more or less cores inside. They have different architectures and configurations, some of them are run with operating systems and some of them have no ability and resources for it. All SoC have different programming languages, assemblers and tools. And all this variety of platforms are needed to be programmed and it's better to make possible to port developed programs over wide enough set of SoCs.

Existing retargetable compilers have the set of problems that do not allow using them for a large variety of new developed systems. Among them there are:

- Complexity of internal (intermediate) representation of a program: it needs to represent all aspects of frontend programming languages and it's hard to adopt it to the target platform needs;
- Insufficient formalism: For most compilers internal representation is not based on a formal model (model of computation) and is just a some kind of representation of parsed frontend language;

- Complexity of creating a backend: backend needs to take into account complex intermediate representation and can contain constructions that cannot be applied in target platform.

In this article we propose the new approach that will allow making a parallel programming easy for the multi-target SoC compilation.

II. Main part

A. Overall description

The model of computation (MoC) should be the basis of everything that is relative to programming, especially to parallel programming. The MoC should specify the rules of operations execution, provide the correctness of computations in different execution conditions, provide mechanisms for a formal verification, allows constructing formally proven transformations and optimizations.

The overall approach scheme (Fig. 1) looks similar to a common scheme for cross-compilers.

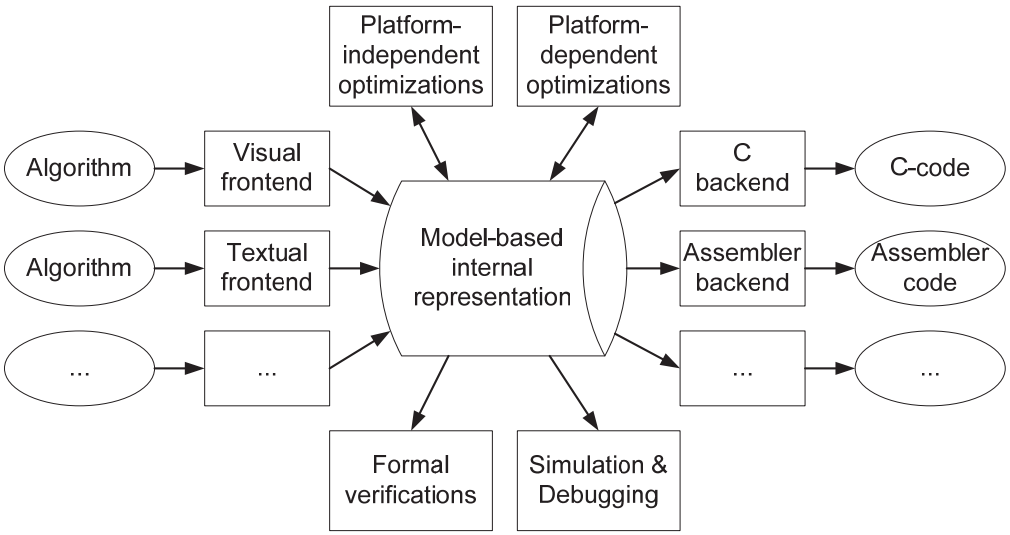

Fig. 1. Overall scheme of system on chip retargetable parallel programming

However it has set of ideas that make this approach rather different from traditional retargetable compilers approach.

B. Frontends

In [2] we propose the SoC programming concept that has the set of key thesis:

1. Medium-grained parallel computations with sequential processing inside medium-grained blocks;
2. Visual approach to parallel programming;
3. Dynamics of parallel computations.

We are still believe, that a decomposed algorithm with obviously specified split blocks and data and control dependencies between blocks corresponds to a network of objects or graph and that a graphical (visual) representation is a most natural

representation of a graph or network. Here (Fig. 2) is the example of the program scheme in the VPL (Visual Programming Language).

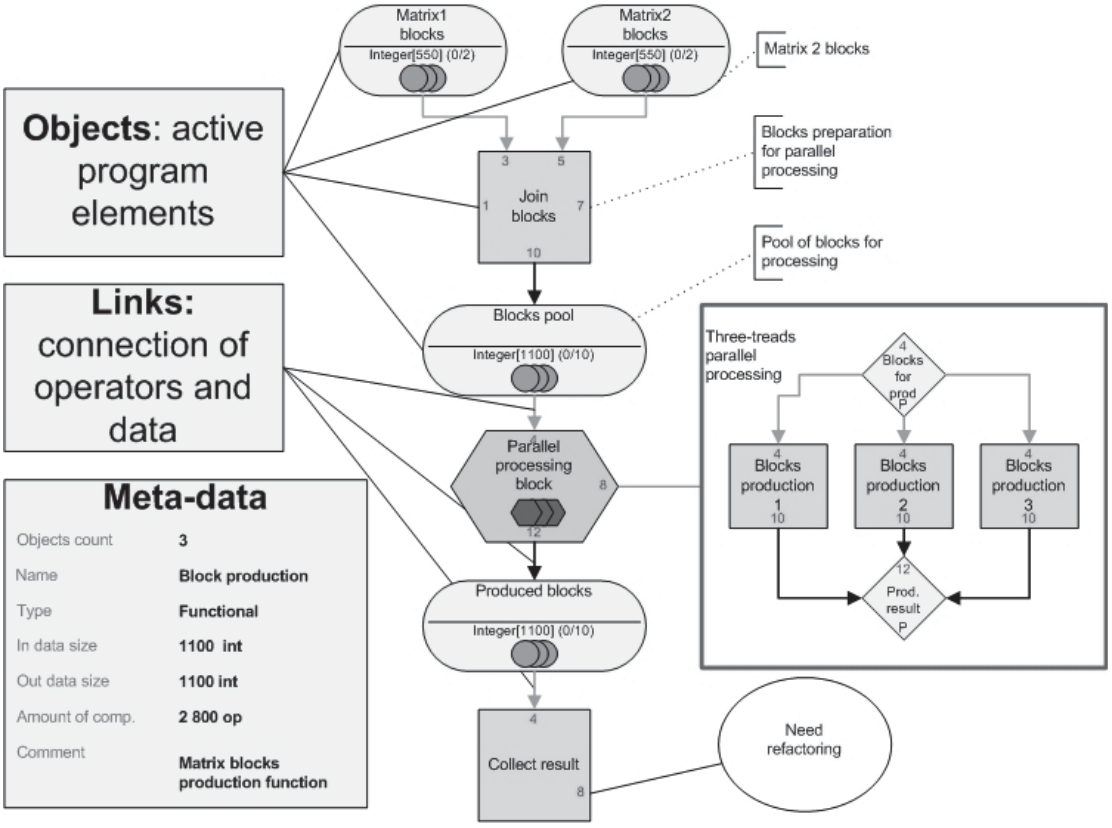

Fig. 2. Overall scheme of system on chip retargetable parallel programming

However it's not a crucial point of our approach. Any language can be attached to the retargetable environment, if the correspondent frontend can (and will) be developed. The only limitation is that this language via frontend should be translated to the required intermediate representation.

C. Formal model and model-based debugging

In our approach we use the AGP-model [3,4] (Asynchronous Growing Processes) as a formalism and basis for retargetable parallel programming methods. The AGP-model has three significant advantages:

- Is algorithmic complete: allows representing any type of computations;
- Is parallel, asynchronous and decentralized: allows to describe and implement truly parallel computations;
- Limited complexity: has only tenth of base computational components.

Also the AGP-model has two abilities, which are required to be represented in MoC used for internal representation of retargetable approach.

1. Verification of parallel program "by design".
   Computational model allows formulating and proofing theorems, corollaries and characteristics of program schemes. Presence of intermediate representation object formal descriptions allows applying proofed in the formal

model characteristics to any programs which language has the correspondent frontend. Such characteristics can be:

a. Formal correctness of program design;
b. Finding deadlocks/livelocks on translation stage;
c. Finding circularities on translation stage;
d. Ability to obtain conclusions about overall program execution basing only on some time of program execution/simulation.

2. Equivalent transformations.

Equivalent transformations are transformations from one program scheme to another program scheme that are formally equal to the source ones using rules that are defined in formal model. The mechanism of equivalent transformation is required for:

a. Optimization transformations of program scheme for specific tasks and platforms using aggregation, grouping etc.;
b. Functional validation of parallel program using debugging of equivalent sequential version with guaranty of equivalent execution of the source parallel implementation and other program characteristics.

We should note that all the above abilities of MoC can be applied to parallel programs on any language that has frontend to the AGP-based intermediate representation independently on what language was the program source and to what is the target platform.

The debugging of programs is another milestone of programs development. All target platforms have their own execution environment, some of them have debugging tools and some have not. Some of them have simulators and some of them have not. Unified model-based intermediate representation allows having the unified simulator that can be applied to any program after frontend passing and will allow making functional program debugging in terms of intermediate representation.

At the result of program processing in model-based intermediate representation we will have the syntactically, semantically and functionally correct program. All these results are propagated to any target platform if it has the correspondent backend.

D. Backend code generation

Backend code generation instead of backend compiling is one of the crucial ideas of the proposed approach. Traditional retargetable compilers are very hard to be extended with alternative back ends, because their intermediate representations are not formalized, are oriented on arbitrary textual input languages and looks more like abstract syntax trees then like MoC. Moreover, backends should generate target platform compiled code, i.e. to make a backend we need to make a real compiler.

Instead of this, we propose to construct not backend compilers, but backend code generators. The reason of this is that code generators are much easier then compilers and that there are many people knows how to program target platform and only few people who knows how to compile for target platform. Moreover, all target platforms in any way have ready compilers from some languages (usually platform-specific assembler, native C or platform-specific C), thus there is no real need to make another compiler for the same platform.

The proposed approach has other advantages against traditional retargetable compilers that are caused by the developed methodology:

- MoC used for intermediate representation is obvious and strictly specified (in implementation meaning). Thus it's obvious for backend developer what should be implemented and how the MoC mechanisms should work.
- MoC used for intermediate representation has three levels of complexity: data-flow (simple), shared data (medium) and parallel dynamics (advanced). Depending on target platform requirements, backend code generator can support any of these levels. For example, back end code generator that supports only data-flow part can be written in less than a week nearly for any target platform.
- It doesn't matter if the target platform is parallel or single-core, it has shared memory or message passing, and its compiler requires C, assembler or other language. As a test case, our team develops the set of backend code generators for different levels of abstraction and different platform architectures: VHDL code, assembler for MC-24 processor [5], sequential native C, parallel C++ using Microsoft agents library, parallel C++ using MPI, and it's not the end.
- The major advantage: backend code generator is the only tool that should be debugged for errors and correspondence with MoC rules. There is no need to debug the developed programs on the target platform if it was debugged on the intermediate representation level, because the correctness of its work will be guaranteed by the MoC.

The main trend of retargetable parallel programming is the target platform independence. However, the proposed methodology provides an ability to create platform-dependent optimizers. Such optimizer can either be a frontend part of the backend code generator or be developed as a separate tool which takes intermediate representation on input and produce optimized intermediate representation on output.

III. Conclusion

The proposed approach for retargetable parallel programming is based on the set of methods that are crucially different from traditional retargetable compilers. The MoC-based intermediate representation allows debugging and optimizing programs on the intermediate level, checking its semantics and making formal verifications independently from a frontend language. Backend code generators instead of backend compilers and various levels of MoC complexity allow developing backend rapidly and easily. The MoC-base of backend code generators allows producing verified target code without additional debugging.

References

[1] Retargeting. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Retargeting.
[2] Alexey Syschikov. Parallel Programming for Many-Core SoC. 7th Conference of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program: Proceedings printed by Saint-Petersburg State university of Aerospace Instrumentation (SUAI). 2010. P.138-146.
[3] V.I. Ivanov, Y.E. Sheynin, A.Y. Syschikov, "Programming model for coarse-grained distributed heterogeneous architecture", XI International Symposium on Problems of Redundancy in Information and Control Systems: Proceedings, SUAI, 2007, c.246-250.
[4] Y.E. Sheynin, "Formal model of dynamic parallel computations in parallel computing systems of experimental data processing", Scientific Instrumentation, 1999, vol. 9, #2. c.22–29.
[5] Signal processor 1892BM2Я (MC-24). http://multicore.ru/index.php?id=47s.