Chapter 2
Compilation Tool Chains and Intermediate Representations

Julien Mottin, François Pacull, Ronan Keryell, and Pascal Schleuniger

2.1 Introduction

In SMECY, we believe that an efficient tool chain could only be defined when the type of parallelism required by an application domain and the hardware architecture is fixed. Furthermore, we believe that once a set of tools is available, it is possible with reasonable effort to change hardware architectures or change the type of parallelism exploited.

2.1.1 Application Domains

In the SMECY consortium, the application providers have selected more than 15 applications. These have been clustered into the following three sets:

Radar Signal Processing

- Passive coherent location, PCL, radar
- Space-time adaptive processing, STAP

J. Mottin (✉) • F. Pacull
CEA, LETI, DACLE/LIALP, F-38054, Grenoble, France
e-mail: julien.mottin@cea.fr; francois.pacull@cea.fr

R. Keryell
SYLKAN Wild Systems 4962 El Camino Real 201 Los Altos, CA 94022, USA
e-mail: Ronan.Keryell@silkan.com

P. Schleuniger
DTU Compute, Technical University of Denmark, Matematiktorvet, 2800 Lyngby, Denmark
e-mail: pass@dtu.dk

M. Torquati et al. (eds.), *Smart Multicore Embedded Systems*,
DOI 10.1007/978-1-4614-8800-2_2,

- Active electronically scanned array, AESA, radar
- Measurement and analysis of time-varying targets

Radar applications reconstruct a view of the world from echoes of radio signals. In the SMECY project, both active and passive radar approaches have been studied. The structure of the radar applications varies but generally consists of an input filtering stage followed by a stage which correlates echoes to real-world features. The correlation phase tends to be very computational intensive but also requires high memory performance.

Multimedia, Mobile, and Wireless Transmission

- Orthogonal frequency division multiplexing, OFDM, modem
- Audio decoding
- Software spectrum sniffer for cognitive radio systems in the ISM Band.
- Mobile network protocol analyzer
- Video processing on mobile nodes

The applications in this set are more diverse than the applications in the radar set. Three applications address radio communication. These three range from data coding applications to a spread spectrum radio sniffer. The mobile aspect is represented with an application which performs distributed video processing on mobile devices. The final application in the set is an analyzer for network protocols with high demands on throughput.

Stream Processing (Video Surveillance)

- Video surveillance
- Video encoding
- Object detection
- Wavelet image processing
- High dynamic range image processing
- 3D graphics particle rendering
- Time-space features processing

The final set of applications focus on high performance image and video processing. All the applications have strict requirements on latency and are commonly used in a data-streaming fashion. While the algorithms used in the applications are diverse, they are all computational intense due to high image resolutions and strict latency requirements.

2.1.2 Target Platforms

For each application set, we have chosen, as target, one of the two platforms provided by the project partners: P2012, also known as STHORM, provided

by STMicroelectronics and EdkDSP, and its last version ASVP, provided by the academic partner UTIA, Institute of Information Theory and Automation in Czech Republic. These two targets are very different and so require different tool chains. The main difference between the platforms is that P2012 is based on replicated patterns interconnected by a NoC, network on chip, while EdkDSP utilizes heterogeneous hardware accelerators.

The two first sets of applications used P2012 and the third EdkDSP.

2.2 The Tool Chains

One of the major outcomes of the SMECY project is the interaction of the different tools provided by the partners as assets at the beginning of the project and the ones developed on purpose during the project. The former have been adapted to use the defined intermediate representation.

Figure 2.1 illustrates the big picture with the three application sets: green for radar signal processing on P2012, red for multimedia, mobile, and wireless transmission also on P2102, and finally blue for stream processing on EdkDSP. We call the application sets, *clusters*.

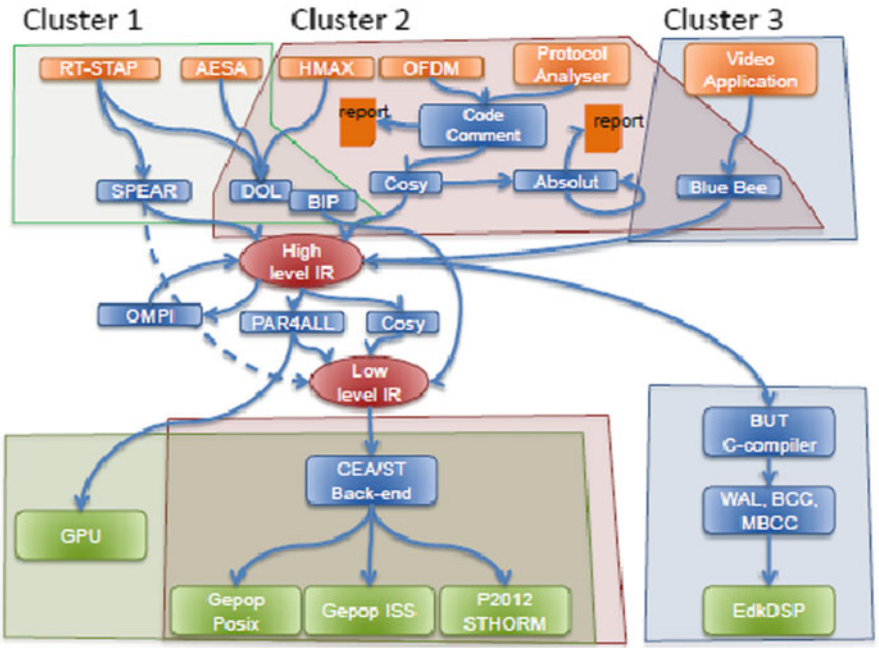

Fig. 2.1 The three tool chains

On top, in orange, we can find the applications from the three application sets previously are introduced. On the bottom, in light green, we have the two target platforms: EdkDSP and P2012, plus some other platform used as reference and some simulators for the P2012 platforms. In the middle, in blue, we have all the tools we considered in the project:

- Front-end tools that guide the programmer in his/her work of mapping the application on a multicore system. This part is independent of a given physical architecture and considers only an abstraction of the actual hardware.
- Back-end tools, that are dedicated to a given hardware in order to obtain the best performances out of a given multicore platform.

In between the front-end and back-end tools, in red, is a key result of the project: the two level intermediate representations presented into more details hereafter.

2.3 Intermediate Representations

The SMECY intermediate representations have been developed to be sufficiently generic to allow expressiveness and be compatible with various programming models or model of computation. We decided on two intermediate representations, both capitalizing on existing standards:

- A high-level intermediate representation, called *SME-C*, based on standard C with pragmas similar to OpenMP.
- A low-level intermediate representation, called *IR2*, meant to be used by back-end tools for target platform code generation. The representation is based on standard C with a set of APIs based on a standard proposed by the Multicore Association.

We decided on this multitiered approach because a single intermediate representation is not a good match for the different levels of abstraction used by the tools in the tool chains. Furthermore, we have developed tools that can transform between the two intermediate representations. Examples include Par4All, by SILKAN-project, and CoSy, by ACE, which can transform SME-C source files to the low-level intermediate representation.

We will now describe the two intermediate representations.

2.3.1 High Level Intermediate Representation: SME-C

The SME-C representation is a #pragma-based language extending C and C++ to allow heterogeneous computing with several accelerators and memory spaces. SME-C is based on OpenMP to express parallelism on a main host, extended with some specific #pragma to invoke accelerated functions on a given accelerator or

to pipeline loops. The programming model is based on C processes, with a virtual shared memory and threads *à la* OpenMP. SME-C includes mapping information stating on which hardware part a function is to be placed and run.

An important part of SME-C is the support for describing memory dependencies at the function call level. Memory dependencies are approximated with rectangles, more generally hyperparallelepipede in any dimension, in multidimensional arrays. The hyperparallelepipede is tagged as read, written, or both. With this information, the tools can infer the memory communication. This approach is similar to XMP pragmas and high performance Fortran, HPF.

SME-C leverages the TR 18037 Embedded C standard to express detailed hardware properties such as hardware register names and precision of fixed-point arithmetic computations.

In several of the SMECY applications, data is processed in a pipelined fashion however with strong data dependencies that require data to be processed sequentially at specific points in the pipeline. This is opposed to a data-parallel model, where such dependencies do not exist. A streaming model is a suitable parallel computation model for such programming patterns.

Streaming models can exploit both a coarse-grained level of parallelism and parallelism at a fine-grained level. At the fine-grained level, the overhead of passing data between processing nodes in the pipeline must of course be minimized. For fine-grained parallelism, this may require hardware support. To achieve good load balancing, it is important that the processing nodes have a comparable grain size. If one node required much more processing than the others, it becomes the bottleneck and no parallelism can be exploited.

Streaming has the advantage of data locality. Data is passed around in the distributed point-to-point network. Such networks can be implemented far more efficiently than, for example, a shared memory connection between the processing nodes.

SME-C supports streaming through annotations. The communication is derived from the program source and generated by the compiler. This is very important for the parallel performance tuning of the application because it allows for experimentation with different load partitioning without having to reprogram process communication.

2.3.2 Low-Level Intermediate Representation: IR2

The low-level intermediate representation, called *IR2*, is based on an API developed by the Multi-core Association [1,2]. The Multi-core Association has developed APIs for communication, resource management, and task management. In the SMECY project, we have used the communication API called *MCAPI*. This improves interoperability and allows us to leverage the effort the Multi-core Association has put into developing the API. The low-level intermediate representation consists of C source code with MCAPI calls. MCAPI is based on three main concepts:

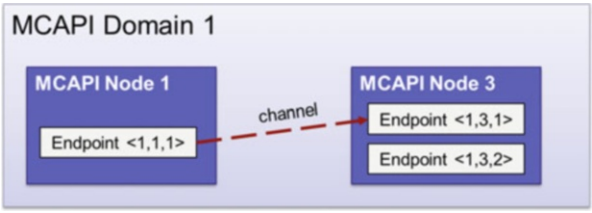

Fig. 2.2 MCAPI main concepts

- A *node* is an independent thread of control that can communicate with other nodes. The exact nature of a node is defined by the implementation of the MCAPI. It could, for example, be a process, a core, a thread, or an HW IP.
- An *endpoint* is a communication termination point and is therefore connected to a node. One node can have multiple endpoints, but one endpoint belongs to a single node.
- *Domain* is a set of MCAPI nodes that are grouped together for identification or routing purpose. The semantics attached to a domain is given by the implementation. Each node can only belong to a single domain.

Compared to MPI, MCAPI offers inter-core communication with low latency with minimal footprint. MCAPI communication nodes are all statically predefined by the implementation.

MCAPI offers three fundamental communication mechanisms:

- *Messages* are streams sent from one endpoint to another. No connection is established between the two endpoints to send a message. This is the easiest way of communicating between two nodes.
- A *packet channel* is a FIFO unidirectional stream of data packets of variable size, sent from one endpoint to another.
- A *scalar channel* is similar to a packet channel, except that only fixed-length word of data can be sent through the channel. A word may be 8, 16, 32, or 64 bits of data.

Figure 2.2 presents an overview of MCAPI. The communication channels can then be set up between two different endpoints.

Within the project, we have developed an implementation of MCAPI for the STHORM platforms. The main objective of this implementation was to offer a uniform level of abstraction and a homogeneous programming interface for the whole platform.

2.3.3 Source-to-Source Compilers

In the project, we have developed two source-to-source compilers whose role is to translate from the high-level intermediate representation to the low-level intermediate representation. The two compilers are complementary and translate

different parts of SME-C. The first one is dedicated to the streaming annotation outlined above. The second source-to-source compiler developed translates the rest of SME-C.

In addition, a more general OpenMP source-to-source compiler has been developed.

Source-to-Source for Streaming Annotation (ACE)

ACE has implemented a source-to-source compiler that accepts C programs with the streaming annotation and produces a partitioned program with separate processes, nodes, for each stage of the streaming pipeline. In addition, it implements the communication links between the nodes.

The generated code includes library calls to implement the low-level tasks of process creation and synchronization. This small library of about five calls is currently implemented on top of, shared memory, POSIX pthreads. It is not hard to retarget this library to different underlying runtime systems.

To be used, the streaming model requires a part of the target application to be rewritten into a particular form, using a while loop and the SME-C stream annotations. Only this part needs to be passed through the source-to-source compiler. Hence, large parts of the application remain unmodified and do not need to be processed by the stream compiler.

Stream termination is currently not handled well. Stream termination has to be mapped from a sequential to a distributed decision process and the design and implementation of that is still to be done.

It is possible to stream a `while` loop in several pipeline stages that execute in parallel and pass information between stages.

The two pragmas used here are:

`#pragma smecy stream_loop`: this indicated the following `while` loop must be turned into a stream of processes.

`#pragma smecy stage`: this acts as a separator between groups of statements and defines the boundary of pipeline stages. Only data passing over these separators is turned into communication.

Smecc Source-to-Source (SILKAN)

Smecc is a source-to-source translator from SME-C pragma-oriented C and C++ to OpenMP and IR2. It is based on the ROSE compiler. The translator is intended to be used as a front-end to the Par4All compiler.

Par4All is an automatic parallelizing and optimizing compiler developed notably by SILKAN (HPC Project). Par4All uses a set of macros and API functions collectively called *Par4All Accel runtime* to ease parallelized code generation by masking implementation-dependent or hardware-dependent details. Par4All can parallelize to several CPUs, using OpenMP, or GPUs, using Cuda and OpenCL.

OMPi Source-to-Source (UOI)

OMPi is a lightweight source-to-source OpenMP compiler and runtime system for C, conforming to version 3.0 of the specifications. The OMPi compiler takes C source code annotated with OpenMP #pragmas and produces transformed multithreaded C code, ready to be compiled by the native compiler of the system. It provides a multitude of runtime libraries for supporting efficient execution. OMPi supports shared memory systems, threads, or processes, with loop or task-based parallelism. Task-based parallelism can be combined with message passing for a hybrid model. OMPi yields a performance improvement through OpenMP parallelization and sophisticated dynamic runtime scheduling.

2.4 The Tools

We conclude with brief descriptions of each tool in the tool chains.

2.4.1 Front-End Tools

BIP (VERIMAG)

BIP (Behavior, Interaction, Priority) is a formal component-based framework, which allows building complex systems by coordinating the behavior of a set of atomic components [3]. Atomic components are described as Petri-nets extended with data and functions described in C. The BIP toolbox includes translators from various programming models into BIP, source-to-source transformers for BIP, as well as a configurable compiler for generating code executable by a dedicated middle-ware engine. The BIP framework and toolbox aims to support the design flow for embedded applications. The tool accepts input in BIP language and outputs debugging functionalities, deadlock analysis, performance analysis based on simulation, and C/C++ implementation for general purpose platforms, according to different options (real-time, single/multithreaded, monolithic, distributed, ...).

ABSOLUT (VTT)

ABSOLUT is a tool to perform system-level performance exploration. It considers standard multithreaded C code as input programming language. ABSOLUT creates an abstract workload model and simulates the workload model on performance

capacity model of the target platform. The approach enables early performance evaluation, exhibits light modeling effort, allows fast exploration iteration, and reuses application and platform models. It also provides performance results that are accurate enough for system-level exploration.

BlueBee (TUDelft)

BlueBee is a tool chain that allows (embedded systems) developers to port their applications to heterogeneous multicore platforms. It consists of a portioning and mapping toolbox that determines on the basis of performance driven profiling information what parts of the application should be mapped on what particular computing element. Once the kernels identified, the necessary code transformations are performed to insert the necessary instructions to start and stop the different kernels and to transfer the parameters to the different computing elements. The appropriate back-end tools compile the identified kernels for the different HW units to which they are mapped. The HW units can be FPGAs, DSPs, and GPPs. The partitioning toolbox and the back-end tools can be used independently. BlueBee aims to improve performance through (semi)automatic partitioning and mapping on a heterogeneous multicore platform. BlueBee supports ANSI C as input language and outputs ANSI C with pragmas or binary.

Code Comment (DTU)

DTU has explored the use of tools which provide code comments to the programmer. The tools help the programmers make better use of compilers optimization features. For many parallel applications, performance relies not on instruction-level parallelism, but on loop-level parallelism. Unfortunately, many modern applications are written in ways that obstruct automatic loop parallelization. The aforementioned generated comments guide the programmer in iteratively modifying application source code to exploit the compiler's ability to generate loop-parallel code.

SpearDE (Thales)

SpearDE is a code generation environment for custom parallel embedded architectures. The graphical model-based environment provides the user with both domain-specific application interfaces and a heterogeneous architecture description interface, which help the implementation of data-streaming applications for parallel architectures. Using SpearDE, the programmer should be able to fill the gap between functional and implementation levels by using a seamless design flow that includes modeling of both the application and the target architecture. If needed,

the programmer can try several mapping strategies by using the design space exploration facility provided by Spear. SpearDE provides support to integrate code generators for specific targets (e.g., Thales SIMD architecture on FPGA for image processing), or it can be used to properly interface external tools from a more domain-relevant point of view.

Ptolemy II and HAMMER (HH)

Ptolemy II is a software system for modeling and simulation of concurrent real-time embedded systems. It is developed at University of California, Berkley. The tool has a thematic focus on system design through assembly of concurrent components, which is relying on the use of well-defined models of computation to define the interaction between the components. One of the main research areas on the Ptolemy system concerns the use of heterogeneous mixture of models of computation. However, Ptolemy II provides no direct support for multi- and many-core modeling, scheduling, and code deployment. Ptolemy II's code generator infrastructure got modified and extended during the SMECY project by HH. The system is able to generate input to the back-end tools in the form of the common SMECY IR (intermediate representation).

HAMMER is a multi- and many core analysis and mapping tool. HAMMER is built on top of the Ptolemy II system. It uses the Ptolemy II infrastructure for programming and modeling concurrent functional data flow behavior of applications. HAMMER adds the functionality needed for mapping applications on multi- and many-core systems and for analyzing nonfunctional behavior such as timing analysis.

2.4.2 Back-End Tools

UTIA ASVP SDK (UTIA)

UTIA ASVP SDK consists of a C-compiler, assembler, and linker and two application interfaces to the ASVP accelerators. The first part of the SDK contains a C-compiler and assembler (binutils) for hardware accelerator's microcontroller in the used ASVP platform. The tools compile input C codes to microcontroller firmware binaries. Two APIs represented by header files and libraries are the next part of the SDK. The first API is called WAL (worker abstraction layer) and it defines an interface between the host CPU and the accelerators. It offers functions for data transfer and execution control between the host CPU and local accelerator's data memories. The second API defines an interface between the microcontroller and data flow unit and between the accelerator and the host CPU from the side of the accelerator. It offers functions for communication with the host CPU and functions to parameterize and control basic operations in hardware. The UTIA ASVP SDK

provides target-independent interface between ASVP platform and the partner's tools, namely (1) C-compiler and linker—compilation of a control C code to binary firmware for the accelerator microcontroller and (2) APIs—provide basic interface functions.

BUT C-Compiler (BUT)

The Vecta is a C language compiler that generates code for architectures with accelerator and was developed primarily for the EdkDSP platform. It is a fully automatic compilation chain. This compilation chain can be used as back-end compiler for code prepared by BlueBee and Par4All. The approach to compilation is based on a view that the workers may have a predefined set of possible vector operations that they can perform. This set of operations is based on the WAL API provided by UTIA, where a complex operation may consist of a sequence of provided basic operations defined by this API. Vecta allows using multiple BCEs automatically and also legacy code without much modification can be quickly ported to the EdkDSP. Description of BCEs for the BCE acceleration pass is parameterizable and new operations can be simply added or removed. Also, the operation tree obtained from the for-loop body can be analyzed in order to generate application-specific dataflow units for BCEs. The overall design is prepared to be modified for other architectures with external accelerators. The Vecta C-compiler provides automatic off-loading of expensive for-loops to basic computing elements.

deGoal (CEA)

deGoal is a tool designed to build specialized code generators (also known as compilettes) customized for each computing kernel we want to accelerate in an application. Such compilettes are designed with the aim to perform data- and architecture-dependent code optimizations and code generation at runtime. Furthermore, compilettes provide very fast code generation and low memory footprint. This approach is fundamentally different from the standard approach for dynamic compilation as used, for example, in Java Virtual Machines or LLVM JIT. In order to target computing architectures that include domain-specific accelerators and to raise the level of abstraction of the source code of compilettes, deGoal uses a dedicated programming language, which is later transformed to C language by an automatic source-to-source translation in order to ease integration with standard compilation tool chains.

CoSy Compiler Development System (ACE)

The CoSy compiler development system is a modular toolbox for compiler construction. CoSy is a product that is used by many companies worldwide.

CoSy supports C, C++, and extensions such as Embedded C. The CoSy front-end also accepts OpenMP and can be programmed to accept arbitrary pragma extensions to implement domain-specific programming models. For shared memory multiprocessing, CoSy supports OpenMP's relaxed memory consistency model. CoSy is typically used to generate compilers that produce assembly code from C. CoSy can also generate compilers that manipulate or transform source code, and produce source code again. CoSy aims to create best-in-class compilers in the embedded application domain. For this reason, it supports a wide variety of extensions that are not found in other compilers or compiler systems.

References

1. The Multicore Association, "Industry standards to solve multicore challenges," http://www.multicore-association.org/, 2011.
2. The Multicore Association, "Multicore Communication APIs," http://www.multicore-association.org/workgroup/mcapi.php, 2011.
3. A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in bip," in *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, ser. SEFM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12. [Online]. Available: http://dx.doi.org/10.1109/SEFM.2006.27