HOPES: Embedded Software Development Environment for MPSoC

Hae-woo Park, Jinwoo Kim, and Soonhoi Ha
{starlet, jwkim, sha}@iris.snu.ac.kr
CAPLAB – Seoul National University – Republic of Korea
http://peace.snu.ac.kr

Abstract

HOPES is a retargetable embedded software development environment for MPSoC. It is demonstrated with real-life examples on various targets including IBM Cell processor and a virtual prototyping system.

1. Introduction

As more processing elements are integrated in a single chip, embedded software design becomes more challenging: It becomes a parallel programming for non-trivial heterogeneous multi-processors with diverse communication architectures and design constraints such as hardware cost, power, and timeliness. In the current practice, the programmer should manually optimize the parallel code for each target architecture and design constraints.

To solve this problem, we are developing an embedded software development environment, called HOPES [1], for fast and correct design of MPSoC. This parallel programming framework is based on a novel programming model called Common Intermediate Code (CIC). The whole design-flow is described in Figure 1.

2. Common Intermediate Code (CIC)

In CIC, the potential functional and data parallelism of application tasks are specified independently of the target architecture and design constraints. CIC tasks are concurrent tasks communicating with each other through channels. CIC tasks can be automatically generated from other front-end program specifications or manually written by a programmer as illustrated in Figure 1.

A CIC task code is defined by the following three functions: \( \text{task name}_\text{init}() \), \( \text{task name}_\text{go}() \), and \( \text{task name}_\text{wrapup}() \). \( \text{task name}_\text{init}() \) is called once when the task is initialized. \( \text{task name}_\text{go}() \) is the main body of the task and is executed repeatedly by the scheduler. \( \text{task name}_\text{wrapup}() \) is called before the end of the task to reclaim the allocated resources.

To enable target-independent specification, generic APIs are used in the CIC task code. The APIs abstract the communication between tasks and architecture-dependent functions so that the CIC translator may translate them into target-specific codes.

Information on the target architecture and the design constraints is separately described in an xml-style file, called the architecture information file. Based on this information, the programmer maps tasks to the processing components, either manually or automatically, exploiting both functional and data parallelism simultaneously.

Then, the CIC translator automatically translates the task codes in the CIC model into the final parallel code, following the mapping decision. The CIC translation involves synthesizing the interface code between tasks and a run-time system that schedule the mapped tasks, extracting the necessary information from the architecture information file needed for each translation step. Based on the task-dependency information that tells how to connect the tasks, the translator determines the number of inter-task communication channels. Based on the period and deadline information of tasks, the run-time system is synthesized.

The CIC translator is the key ingredient that makes the CIC tasks truly retargetable with respect to architecture change and mapping decision. As a preliminary experiment, we have designed a CIC translator for the IBM® Cell® processor with an H.264 encoding algorithm as an example [2]. From the same CIC specification, we also generated a parallel program for an ARM® MPCore® processor that is a symmetric multi-processor, which confirms the retargetability of the CIC model. The HOPES design environment also provides an extensible heterogeneous multi-processor simulator to verify the correctness and performance of the synthesized MPSoC system.
3. Experiments

(1) H.263 Codec on a virtual prototyping system

The H.263 Codec application is specified as an SDF (synchronous data-flow) graph as shown in Figure 2. It consists of three application tasks (H.263 decoder, H.264 encoder, and Display). From the SDF specification, we generate the CIC code automatically. In the generated CIC code, a CIC task was detected as a data-parallel task. We map the CIC tasks to the target processors manually. And following the mapping decision, we generate the target code through a target-specific CIC translator. To show the CIC files are target-independent, we tested this example on two different targets: ARM® MPCore® development board and a virtual prototyping system that is built with an in-house multi-processor simulator.

For the ARM® MPCore® target, the CIC translator generates pthreads for the CIC tasks and a pthread scheduler for scheduling them in SMP manner. For the virtual prototyping system, the translator generates tasks and communication functions for each processor.

(2) H.264 Encoder on a Cell processor

Figure 2: H.263 codec in SDF model

Figure 3: The CIC tasks of H.264 encoder

In this experiment, we manually wrote the CIC tasks (as seen in Figure 3) for an H.264 encoder application where a CIC task, ME (motion estimation) task, is defined as a data parallel task. The ME task has wave-front parallelism, which is expressed by a “dependency vector” parameter associated with a data-parallel CIC task. We selected as the target platform the IBM® Cell® processor that is a heterogeneous multi-processor system, consisting of one PPE (PowerPC Processing Element) and eight SPEs (Synergistic Processing Elements), with non-trivial communication architecture. The CIC translator successfully generates PPE and SPE codes and we can take 2.8x speed-up compared with the PPE-alone reference code.

(3) Lego® Mindstorms® NXT robot

While the proposed CIC model can express the data-driven computational tasks naturally, it needs to express also the control behavior of the application in order to become a general embedded software development environment. So, as a preliminary experiment, we made a control application that changes the behavior of a Lego® Mindstorm® robot depending on the sensor inputs and their priorities. This experiment shows how to specify the control behavior in CIC.

4. Conclusion

In this project, we aim to develop a target-independent embedded software development environment for MPSoC. We proposed a novel parallel programming platform, called CIC, which separates initial algorithm specification and target code implementation. Various initial specification methods are supported in the proposed framework as long as they can be translated into a CIC specification. Target-specific codes are automatically generated by target-specific CIC translators.

We have tested the HOPES environment for various targets including the Cell® processor, MPCore® SMP system, and our multi-processor simulator with real-life examples such as H.263 codec and H.264 encoder.

References


Acknowledgement

This work was mainly supported by Acceleration Research sponsored by KOSEF research program (R17-2007-086-01001-0). The ICT and ISRC at Seoul National University and IDEC provided research facilities for this study. This work was also partly sponsored by BK21 project and ETRI SoC Industry Promotion Center, Human Resource Development Project for IT-SoC Architect.