A Polyhedral-based SystemC Modeling and Generation Framework for Effective Low-power Design Space Exploration

January 29, 2018

1 Introduction

对于整个 Soc 的建模，业界已转向使用高层硬件描述，TLM(transaction level) 的 SystemC 广泛使用。在传统的设计流程中，the SoC design specification is often provided in a high level language such as C/C++ by software engineers as a golden reference model，然后手工进行软硬划分并编写 systemC 描述的硬件代码。这样的流程有以下弊端：1. 硬件部分的 system 实现需要 additional effort; 2. SystemC model 是手工实现的，难以有效和 extensively 探索不同的设计选择; 3. 这一阶段由于缺少低层次的实现细节，难以获得精确的能耗估计; 4. SystemC IP 一般考虑仿真速度、复用等，不一定可以作为 HLS 的输入去综合; 5. System C model 只能提供一个 design point 的仿真，整个空间太大了。

本文提出了自动化设计流程来解决上述问题，针对的是 the class of affine programs，例如图像处理、医学影像、统计等。这一类程序的一个特点是 having a control-flow that can be exactly described at compile-time [4] thereby allowing the design of accurate power/latency models, and very powerful optimization frameworks to expose parallelism with data reuse that have already been developed。输入是 C++ 应用，可以自动提取出 affine program 的区域，并自动化执行三个操作：自动进行 software transformation，体现并行度和 temporal data locality。自动生成两个版本的 System C，一个包括 power 和 latency 的高层模型用，一个用于 HLS 的可综合的 systemC。自动实现一个 DSE engine，考虑了不同的 loop tile sizes 和 parallism degree。

2 SystemC Generation Framework

2.1 Overview

相比较之前的工作，该 framework 的特点有以下几点：使用了 Polyopt compilation framework。生成 communication and computation blocks 并建立对应的 power、latency

模型,考虑了开关活性、加速器与系统的交互等因素。latency 和 power 的 characterization 是针对 1 个 tile，可以借助 polyopt 分析。相比较 functional level 的功耗分析，tile-based 可以更方便地使用 IP、blackbox 在设计中。快速仿真，使用行为级的 systemC。

整个流程如下图所示，首先将输入的 program 转化成可以做 loop tile 的形式并使用 Polyopt 来 tile loop。之后，将 tile 提取出来，分成不同的部分包括计算 block 和通信 block，后者在 tile 的开始或结束负责把 memory 和 local buffer 之间的数据移动。下一阶段，借助 gate-level 的仿真 characterize 这些 block 建立 power 和 latency，并建立功耗模型。最后借助 polyhedral analysis, 为 tiled loop kernel 生成 systemC 模型并将 power 和 latency 的模型加进去。

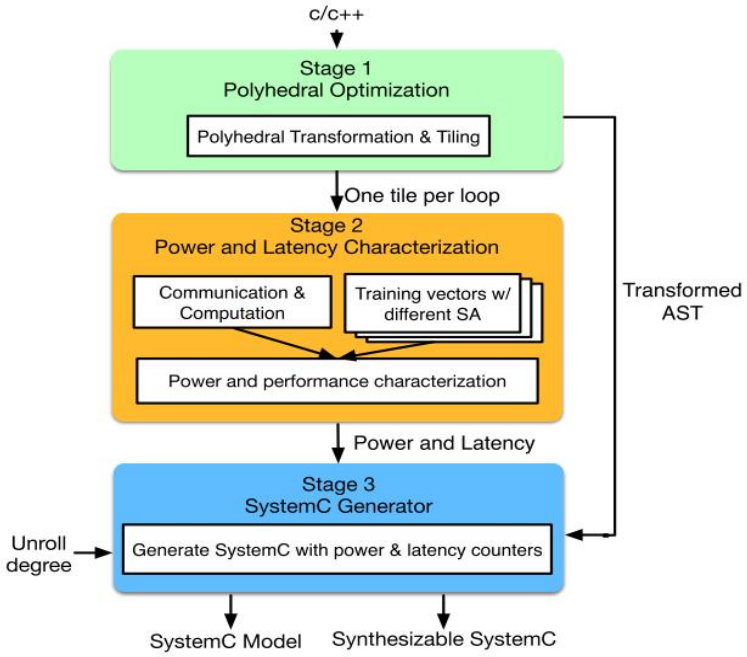

Figure 1: 系统流程

2.2 Architecture of Generated Accelerator

生成的加速器架构基本结构如下图所示，包括计算模块 (Acc_tile) 和片内存储 (Local_mem)。计算模块负责从 local_mem 中读取数据并计算和写回，而 communication blocks 负责 local_mem 和 main_mem 之间的数据交换。有两个可控制参数，一是 tile size，影响 loop tiling 的选择、buffer size 以及相应通信消耗; 二是 parallelism degree，决定哪一个层次的循环被展开; 这两个因素决定了 tile 的重复单元个数。此外，在每个 block 中实现了一个 input switching activity calculation function，用以分析使用活性。(利用率？)

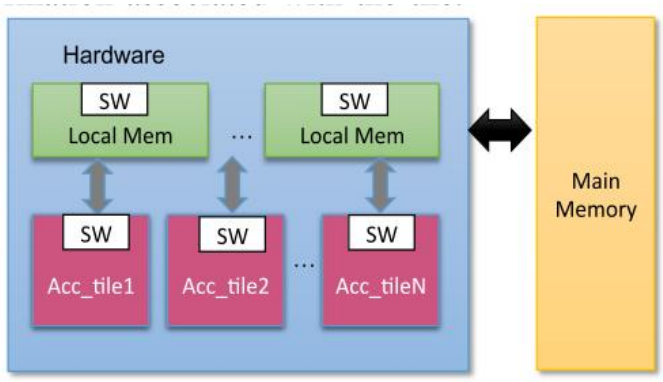

Figure 2: 硬件架构示意图

2.3 Software Transformation

软件转换有 3 个要求：(1) 要体现 temporal data locality 来减少数据移动; (2) 要体现并行性; (3)explicit atomic computation tiles and the associated tile data reuse buffers and data transfers operations(这一点还不太理解); 但所有这些都可以借助 Polyopt 实现。

2.4 Power and Latency Characterization

这一部分关注单独的 tile，把一个 tile 的 computation block 输入 HLS 和 Memory Compiler 分别生成 RTL code 和 local mem。这两部分结合起来做逻辑综合生成网表，并进行门级仿真，然后提取 power 和 latency 的信息。计算部分有 loop unroll 和 pipeline 的考虑，对于 communication block，生成一个 memory Ip 库 (with 不同 size) 并根据不同的开关活性提取 power 和 latency 信息组成查找表。此外，也会生成 testbench 和 input vectors。后者的开关活性可控，服从一定分布等。功耗有三部分：internal, switching and leakage power，前两个动态功耗，分别由 capacitive charging/discharging of output load and internal transistors of the logic gates 导致，最后一个是静态功耗。

2.5 SystemC Code Generation

high-level model: 每一个 tile 生成 systemC 中的 module，wait() 来模拟 function module 的 latency，power 信息根据不同的开关活性从 power model 中索引。switching activity calculation function 没看太明白。

synthesizable version: 可综合的版本有三个调整：1. 要解决访存冲突;2. 要解决同一存储地址的 read/write 冲突; 3. 移除 high-level 模型中的 latency 和 power 的函数。

2.6 Integration

生成 SystemC 并将一个 tile 的 power 和 latency 信息加入后，可以做快速的 systemC 仿真 (使用的就是 SystemC 仿真)。latency 通过在每一个 tile 中插入的 systemC wait() 函数来计算。功耗通过插入 monitor 函数，在每一个 tile module 开始和结束式调用

update_power( ) 来计算功耗，根据 wait( ) 计算执行时间提供信息。Power 和 latency 都是分别考虑 computation blocks 和 communication blocks，不同的 tile 构成系统时，由于是重复的调用且并行的 tile 是复制的，因此当系统启动和同步等一些操作的 overhead 可忽略时，系统的 power 和 latecy 是这些 tile 的求和，具体计算时会考虑动态功耗、静态功耗差异、开关活性、tile size、unroll factor 等信息。