DOI:10.19816/j.cnki.10-1594/TN.2021.02.074

高层次综合综述 *

徐瑞帆[1],肖有为[1],罗 进[1],梁 云[1,2]

(1. 北京大学 北京 100871; 2. 北京智源人工智能研究院 北京 100084)

摘 要:随着定制硬件设计的需求不断增加,需要一种方便、高层次的硬件设计方法。本文介绍了高层次综合的流程和发展历史,并且列举了相关的算法和优化技术,最后展望了未来的发展趋势。

关键词:高层次综合;调度;中间表示

中图分类号:TP391.7 **文献标识码**:A **国家标准学科分类代码**:520.6050

The overview of high-level synthesis

XU Ruifan[1], XIAO Youwei[1], LUO Jin[1], LIANG Yun[1,2]

(1. *Peking University*, *Beijing 100871*, *China*; 2. *Beijing Academy of Artificial Intelligence*, *Beijing 100084*, *China*)

Abstract: With the growing demand of customized hardware design, a convenient and high-level approach of hardware design is necessary. The concept and developing history of high-level synthesis is introduced, related algorithms and optimization techniques are listed as well. The prospect of future trend is declared at last.

Keywords: high-level synthesis; scheduling; intermediate representation

0 引 言

高层次综合(high-level synthesis, HLS)是一种将高层次语言描述的行为自动转换为具有相同功能硬件的综合技术,因此也被称为行为级综合。这项技术意图将硬件设计的难度大大降低,从而可以直接在更高的抽象层次定义电路,大大减少了硬件开发时间。高层次综合技术目的是让开发人员无需过多的关注于硬件实现,从而更多关注于算法层面的设计与实现。

近10年来,随着Xilinx、Intel、Mentor等厂商对于自家工具的推广,高层次综合技术逐步的推广开来。尤其是在FPGA领域,高层次综合已经成为了快速实现设计的常用方法。随着高层次综合工具的逐渐成熟[1-2],现在在许多情况下可以生成与人工设计近似的结果,为硬件设计实现提供了快捷且有效的方式。

1 高层次综合概况

早在20世纪80年代,高层次综合概念就已经被提出,并且在学术界已经发表了一些有影响力的工作。但是由于当时硬件规模还并未爆发式增长,并且随着Verilog、VHDL等硬件描述语言逐渐成熟并且被硬件工程师广泛使用,高层次综合并没有成为硬件设计的常用方式。随着硬件规模的不断提高,使用传统硬件描述语言实现RTL代码的难度大大增加。这种方法需要设计者在RTL层面设计电路,设

*基金项目:国家重点研发计划(2020AAA0105200)项目资助

徐瑞帆,博士研究生,主要研究方向为软硬件协同设计、EDA设计自动化。E-mail:xuruifan@pku.edu.cn

肖有为,本科生,主要研究方向为软硬件协同设计、EDA设计自动化。E-mail:shallwe@pku.edu.cn

罗进,本科生,主要研究方向为软硬件协同设计、EDA设计自动化。E-mail:luo-jin@pku.edu.cn

梁云(通信作者),副教授,主要研究方向为软硬件协同设计、EDA设计自动化。E-mail:ericlyun@pku.edu.cn

计具体的时间周期行为,这种设计方式过于底层,从而导致开发周期过长,无法快速迭代,因此在更高层次进行硬件设计的需求大大增加。与此同时,FPGA等可重构架构逐渐兴起大大增加了硬件实现需求,并且由于片上资源数量固定,相比于ASIC更易于进行高层次综合,高层次综合技术开始受到了广泛的关注。

现在的高层次综合工具通常选择C、C++、OpenCL等语言作为输入,这类高层次语言抽象程度较高,因此相比于低层次语言表达能力更强,开发效率更高。但是由于这类高层次语言不包含时序信息,因此无法被简单的转换为电路实现。传统的高层次综合主要可以分成3个过程:调度、分配和绑定。调度试图将每个运算调度到特定的时钟周期上,在保证数据依赖的同时希望尽可能地减少时间。通过赋予确切的时序信息,使用有限状态机操纵控制逻辑,使得高层次语言可以向硬件进行转化。分配过程根据功耗、资源等硬件限制,为硬件生成确定具体使用加法器、乘法器等的计算单元,然后绑定过程会将每个操作映射到对应的计算单元,同时需要为变量分配寄存器、多选器,并且尽可能的复用资源从而减少所需面积。

目前高层次综合工具主要应用于需要快速部署并且对于性能、资源没有过高要求的应用场景下。当前的高层次综合相比于底层硬件开发仍然存在许多不足,在性能、资源方面仍然面临较大挑战,尤其在不规则计算或者访存密集型任务自动分析仍然存在很大缺陷。因此,学术界仍然有许多研究专注于提高高层次综合的性能与资源,以追求在各种任务下都可以自动生成与人工近似甚至相媲美的结果。

2 高层次综合的发展历程

2.1 早期高层次综合的调度算法

在高层次综合的流程中,编译器前端将高层次语言转换成控制-数据流图(CDFG)形式,其中控制流由基本块的连边表示,基本块内有多个结点表示运算,数据流由运算之间的连边表示,HLS后端再对CDFG处理最终得到RTL层面的硬件描述语言。其中第1步是对CDFG进行调度,确定每个运算执行的开始时间。调度算法是高层次综合中最重要的步骤,HLS工具调度算法的效果对QoR有很大影响,因此早期高层次综合研究大多针对于调度算法。

调度问题可以进行如下形式化:

求出每个运算 op 的开始时间 t_{op},并满足任意运算开始时间不早于其依赖的运算的结束时间,并最小化总延迟(最晚结束的运算)。

该问题为无约束的调度问题,可以使用ASAP(as soon as possible)和ALAP(as late as possible)算法在多项式时间内得到最优解。HLS中的调度问题一般还有其它约束,例如资源约束:即任意时刻使用某类资源的运算数目不能够超过这类资源的数量;以及时间约束:即某些操作需要在某个时间之前执行。除此之外,若允许一个时钟周期内执行多个操作(operation chaining),需要满足同一时钟周期内两个操作的关键路径延迟不能超过时钟周期大小。

存在资源约束的调度问题是NP Hard的,可以通过整数线性规划[3]进行求解,时间复杂度是输入规模的指数级别。早期调度算法一般采取启发式的方法在多项式时间内进行求解。List Scheduling、Force-Directed Scheduling(FDS)[4]和SDC-Based Scheduling[5]是其中比较著名的调度算法。

List Scheduling算法采用了贪心的思想,算法维护前驱已经调度完成的运算集合,对于当前时刻,选出优先级最高的运算进行调度,如果当前没有空余的资源,则将这个运算的调度推后。List Scheduling算法的效果十分依赖于优先级函数的选取,可以选择运算ALAP与ASAP调度结果的差值作为优先级,也可以与FDS等其他调度算法相结合。

FDS解决时间约束下的调度问题,在满足调度结果的延迟不超过ASAP算法的情况下,最小化资源等开销。FDS首先通过ASAP算法和ALAP算法求得每个运算所允许的最早和最晚的开始时间,得到一个初始的调度可能,每次为一个运算确定开始时间,并为每类运算维护一个分布图,代表某个时间使用这个计算资源的期望运算数量,初始时认为每个运算等概率地被调度在这个区间中。FDS通过模仿计算弹簧弹力的胡克定律设计代价函数,通过该代价函数不断迭代,最终将每个运算的调度区间固定

为确切的周期,从而得到调度结果。每移动一个运算同时会影响到其前驱和后继的分布,类似于会对周围运算的弹力会产生变化,从而进一步对其前驱和后继产生影响,FDS算法将运算调度至总代价最小的时间上。FDS算法的代价函数目标是尽可能减少并行的同类运算从而减少需要的计算资源。

调度问题中的许多限制可以用 $t_i - t_j \leqslant c$ 形式表示,可以整个调度问题建模为一个差分约束系统,例如要求一个运算必须在其前驱结束之后才能开始。 SDC-Based Scheduling利用这一性质,将调度问题建模成差分约束系统,可以通过Bellman Ford算法判断解的存在性,并由于全幺模的性质,可以通过线性规划算法得到最优整数解。对于资源约束,SDC-Based Scheduling中使用启发式的方法将其转换为差分约束:对于同一个基本块中的运算,在不考虑资源约束的情况下确定一个可行的全序关系,对于不能共享资源的两个运算,要求他们的运算间隔必须大于运算的延迟。相比于List Scheduling和FDS算法,SDC-Based Scheduling对于控制逻辑复杂能够有较好的处理方式,而List Scheduling算法基本只适合用于数据流图。

2.2 高层次综合的优化手段

仅通过进行上述静态调度算法,HLS还无法生成能够与手动设计相匹配的RTL代码,主要问题在于将CDFG直接翻译成硬件并没有合理的探索并行度,导致硬件性能十分有限。因此,之后的许多研究试图在这些调度算法的基础上添加了更多的优化:例如循环展开、循环流水化、内存划分等。通过这些优化手段,HLS工具能够大幅提高并行度,生成的硬件性能得到了大大的提高,从而减小了与手动设计硬件的差距。

1)循环优化

在高层次综合中,常见的循环优化有循环展开与循环流水化。循环展开通过将循环展开为多个并行的运算,能够让HLS后端发现更多的并行度,代价则是会大大增加资源开销。通过恰当的循环展开,HLS工具可以在资源开销与硬件性能之间得到较好的平衡。

循环流水化则是将循环体综合成流水化的硬件,使下一次迭代能够在当前迭代结束之前开始执行,且能够共用硬件资源,其效率取决于流水线的起始间隔(initial interval, II)。HLS中的循环流水化算法借鉴于软件流水化中的Modulo Scheduling[6],先求出资源约束以及循环依赖下起始间隔的下界,再从小到达枚举起始间隔的大小并尝试进行调度,被调度到时间 t 的运算,会与调度时间为 $t + k \cdot II$ 的运算同时执行,算法维护一个大小为 II 的表记录资源占用情况,t 时刻的运算会被记录在表中 $t \bmod II$ 的位置。SDC-based Modulo Scheduling[7]中将循环依赖建模成SDC,通过线性规划求出最小化Register Pressure的解作为参考解,然后基于参考解用Modulo Scheduling求出满足资源约束的解。

2)内存划分

为了达到更高的并行度,往往需要支持对多个内存地址同时进行读写,由于RAM读写端口的限制,需要将一块内存地址划分到多个内存块中从而整体上增加了读写端口。划分后的内存需要保证同时在同一个内存块上的访存操作仍然不能超过读写端口的限制。文献[8]中提出在固定的 II 下,枚举可能的调度结果,并通过静态分析构建出内存访问的干扰图,求出干扰图中的最大团,其大小即为需要的端口数,如果端口数满足要求,则得到合法的解。文献[9]中利用多面体模型对内存的访问和划分进行建模,能够对可能的访存冲突与划分方案进行自动分析、并且支持更加复杂的划分策略。

3)高层次综合工具

高层次综合工具中有面向特定硬件进行综合的工具。ROCCC[10]是一款由UC Riverside开发,面向DSP的高层次综合工具,ROCCC的输入为C语言的子集(不支持指针等C语言特性),对程序进行划分并综合成数据流的形式。GAUT[11]是一款由University of South Brittany开发的面向流水化硬件的高层次综合工具,GAUT将输入的C程序综合成特定的流水线形式,此外用户必须指定流水线的吞吐量,如起始间隔与时钟周期等参数。

此外,更加常用的是面向通用场景的HLS工具。Vivado HLS和LegUp[12]是目前针对FPGA比较常用的高层次综合工具。Vivado HLS前身是AutoESL的

AutoPilot[13]，后被Xilinx收购。Vivado HLS以C，OpenCL程序作为输入，生成面向特定FPGA硬件平台的RTL文件，用户可以在输入程序中加入pragma来指定HLS后端进行特定的优化，例如内存划分与循环展开等，对于常规运算的优化成都较为显著。LegUp是一款由U Toronto开发的开源HLS工具，被Microchip收购后闭源。LegUp以C语言程序作为输入，综合得到FPGA-MIPS的混合结构，能够将输入程序中不适合综合成硬件的部分综合成MIPS处理器上执行的软件指令。LegUp整合了对两个标准并行方式Pthread和OpenMP的支持，对于多线程效果更好。这两个工具都以输入程序的LLVM IR作为综合的起点，在此基础上进行调度以及各种优化。

Catapult-C是Mentor Graphics于2004年发布的高层次工具，支持以C、C++和SystemC为输入，转化生成VHDL、Verilog和SystemC，同时提供了一些外部库和接口方便定义设计限制，相比于前两个工具还支持ASIC设计。EPFL于2018年提出的Dynamatic[14]，不同于之前的静态调度，采用在执行阶段动态调度从而达到以时钟周期最优解运行，牺牲了一部分资源的情况下大大提高了性能表现。

2.3 高层次综合进阶阶段

随着各种优化手段的引入，高层次综合工具可以在一些规则的计算中取得较好的结果。然而，在不规则计算、控制逻辑较为复杂时，上述的优化手段往往无法采用。例如，对于一个不完美的循环嵌套，即不同循环层次都需要进行一定计算，传统的流水线优化静态调度算法不能直接生成较好的结果。因此，许多HLS研究开始在不规则计算上面进行探索。

1）流水线

循环流水线是高层次综合中的关键性能优化。传统的调度算法存在资源限制、内存访问端口限制以及一些长时延循环携带的依赖时不能得到很好的循环流水设计。此外，流水线在编译时信息不足导致无法设计出最佳调度，典型的情况包括编译时不确定的内存访问、不可预测的控制流以及时延变化的运算。在具备这类特点的应用场景中，传统的高层次综合工具必须假设最坏的情况并做出保守设计来保证所有运行情况的正确性。

Josipovic等[14]借鉴现代超标量处理器的思想，提出新的动态调度综合方法，由高层次代码生成数据流电路。数据流电路由一些硬件单元构成，这些单元通过握手控制信号实现延迟无关性。数据在依赖满足时由单元向下一个单元传递，以一种分布式机制实现动态调度，避免由中心化的预设控制器激活运算。其后续工作包括，文献[15]讨论了该高层次综合方法中高吞吐量流水线的实现，文献[16-17]介绍了支持乱序内存访问的内存接口的构建，并拓展框架[18]支持预测执行。还有许多工作同样专注于对高层次综合中动态调度的探索。

另一种克服静态调度保守性并处理动态事件的思路是生成多种调度并在运行时动态选择。Tan等[19]提出的ElasticFlow方法在特定一类不规则的循环嵌套上应用循环流水线，多个有动态界限的内层循环实例被调度并行执行。Dai等[20]提出了通过执行静态调度获得多个流水线的初始间隔来解决可能的不同资源冲突，实现流水线冲洗；其后续工作[21]研究了特定应用的动态冒险检测电路并展示了严格约束下的预测能力。这些技术仍基于静态调度，适用于某些层次的动态行为，但只能在某些特殊的例子中获得表现的提升，相比之下动态调度有更好的通用性。

2）高层次综合验证

验证作为硬件设计的重要一环，随着高层次综合工具的逐渐流行，保证生成正确硬件愈发重要。高层次综合工具只能由编译器自己保证生成的硬件设计，然而当前没有工具能够保证生成的硬件与软件行为完全一致，甚至可能在处理合法的输入时出现无法预料的错误情况。只能通过添加大量的测试数据，通过仿真从而尽可能减少生成错误硬件的可能。

Herklotz等[22]采用类似软件测试的方法，在各种成熟的高层次综合商用工具中发现了一些漏洞，再次证明了无法完全信赖高层次综合工具。同时，其后续研究[23]试图形式化验证高层次综合结果，确保高层次综合生成结果与软件行为完全一致，从而避免生成错误硬件。

3 高层次综合未来发展趋势

1）调度算法&自动优化

目前高层次综合工具仍有许多不足,功能上仍然有许多缺陷。调度算法方面,虽然动态调度可以提高应对不规则计算的性能,但是由于无法进行资源复用导致资源开销过大。通过静态调度替代部分模块,可以一定程度上解决资源问题。当前,主流工具均采用单一调度算法,适用场景单一。DASS[24]能够结合两种调度的优点,可以在不规则计算上大大提供硬件性能。然而由于静态调度与动态调度在综合过程中的割裂,自动进行动静态调度结合的综合流程仍然需要探索。

除了调度算法以外,循环展开、内存划分、流水线优化等优化手段也经常需要在高层次综合工具中使用。当前成熟的商用工具都需要手动添加优化原语,而且只能在高层次综合结束后得到估计的周期数与资源开销。因此,这种方式仍然需要对于底层硬件有足够了解,否则无法生成较好的结果。COMBA[25],FlexCL[26-27]一些工作关注优化原语自动探索,进一步减少开发难度。

2）中间表示

高层次综合在使用中通常被当作一个黑盒,自动生成的RTL代码难以阅读、修改以及优化,并且生成的硬件难以预测。因此,对传统的高层次综合工具而言,优化生成硬件只能在源代码层面进行调整。为了可以在高层次综合的中间环节对生成的硬件进行调整和优化,HIR[28]等工作试图探索中间语言设计和不同层次的优化。利用多层中间表示形式,高层次综合流程可以变得足够透明,开发人员可以在需要的不同层次对于最终硬件生成进行一定的干预与修改。用户可以自行对设计进行修改,甚至可以根据对于计算的了解可以实现不同的优化方式,进一步提高硬件性能。同时,中间语言及其配套的优化框架能够应用于不同的前端语言或领域专用语言,生成不同平台上的硬件设计,可以为高层次综合工具的开发提供极大的便利。

参考文献

[1] LIANG Y, RUPNOW K, LI Y, et al. High-level synthesis: Productivity, performance, and software constraint[J]. Journal of Electrical and Computer Engineering, 2012, 2012.

[2] NANE R, SIMA V M, PILATO C, et al. A survey and evaluation of FPGA high-level synthesis tools[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 35(10): 1591-1604.

[3] LEE J . A new integer linear programming formulation for the scheduling problem in data-path synthesis[J]. Proc. of the International Conf. on Computer-Aided Design, 1989: 20-23.

[4] PAULIN P G, KNIGHT J P. Force-directed scheduling for the behavioral synthesis of ASICs[J]. IEEE Trans. on CAD, 1989, 8(6): 661-679.

[5] CONG J, ZHANG Z. An efficient and versatile scheduling algorithm based on SDC formulation[J]. 2006 43rd ACM/IEEE Design Automation Conference, 2006: 433-438.

[6] RAU B R. Iterative modulo scheduling: an algorithm for software pipelining loops[C]. International Symposium on Microarchitecture, IEEE, 1994: 63-74.

[7] ZHANG Z, LIU B. SDC-based modulo scheduling for pipeline synthesis[C]. 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, 2013: 211-218.

[8] CONG J, JIANG W, LIU B, et al. Automatic memory partitioning and scheduling for throughput and power optimization[J]. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2011, 16(2): 1-25.

[9] WANG Y, LI P, CONG J. Theory and algorithm for generalized memory partitioning in high-level synthesis[C]. Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '14), ACM, 2014: 199-208.

[10] VILLARREAL J , PARK A , NAJJAR W , et al. Designing modular hardware accelerators in C with ROCCC 2.0[C]. IEEE International Symposium on Field-Programmable Custom Computing Machines, IEEE, 2010: 127-134.

[11] COUSSY P, MORAWIEC A. GAUT: A high-level synthesis tool for DSP applications[J]. Springer Netherlands, 2008: 147-169.

[12] CANIS A, CHOI J, ALDHAM M, et al. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems[J]. ACM Transactions on Embedded Computing Systems (TECS), 2013, 13(2): 1-27.

[13] CONG J, LIU B, NEUENDORFFER S, et al. High-level synthesis for FPGAs: From prototyping to deployment[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2011, 30(4): 473-491.

[14] JOSIPOVIĆ L, GHOSAL R, IENNE P. Dynamically scheduled high-level synthesis[C]. Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2018: 127-136.

[15] JOSIPOVIĆ L, SHEIKHHA S, GUERRIERI A, et al. Buffer placement and sizing for high-performance dataflow circuits[C]. Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020: 186-196.

[16] JOSIPOVIC L, BRISK P, IENNE P. An out-of-order load-store queue for spatial computing[J]. ACM Transactions on Embedded Computing Systems (TECS), 2017, 16(5s): 1-19.

[17] JOSIPOVIĆ L, BHATTACHARYYA A, GUERRIERI A, et al. Shrink it or shed it! Minimize the use of lsqs in dataflow designs[C]. 2019 International Conference on Field-Programmable Technology (ICFPT), IEEE, 2019: 197-205.

[18] JOSIPOVIC L, GUERRIERI A, IENNE P. Speculative dataflow circuits[C]. Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019: 162-171.

[19] TAN M, LIU G, ZHAO R, et al. Elasticflow: A complexity-effective approach for pipelining irregular loop nests[C]. 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, 2015: 78-85.

[20] DAI S, TAN M, HAO K, et al. Flushing-enabled loop pipelining for high-level synthesis[C]. 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, 2014: 1-6.

[21] DAI S, ZHAO R, LIU G, et al. Dynamic hazard resolution for pipelining irregular loops in high-level synthesis [C]. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017: 189-194.

[22] HERKLOTZ Y, DU Z, RAMANATHAN N, et al. An empirical study of the reliability of high-level synthesis tools[C]. 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2021: 219-223.

[23] HERKLOTZ Y, POLLARD J D, RAMANATHAN N, et al. Formal verification of high-level synthesis[J]. Proceedings of the ACM on Programming Languages, 2021, 5: 1-30.

[24] CHENG J, JOSIPOVIC L, CONSTANTINIDES G A, et al. Combining dynamic & static scheduling in high-level synthesis[C]. Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020: 288-298.

[25] ZHAO J, FENG L, SINHA S, et al. COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications[C]. 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, 2017: 430-437.

[26] WANG S, LIANG Y, ZHANG W. FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs[C]. 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, 2017: 1-6.

[27] ZUO W, LIANG Y, LI P, et al. Improving high level synthesis optimization opportunity through polyhedral transformations[C]. Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, 2013: 9-18.

[28] MAJUMDER K, BONDHUGULA U. HIR: An MLIR-based intermediate representation for hardware accelerator description[J]. ArXiv Preprint, 2021, ArXiv: 2103.00194.