面向类仿射型数组下标应用的参数化并行存储结构模板

郭振华,吴艳霞,张国印,戴 葵
(哈尔滨工程大学计算机科学与技术学院,黑龙江哈尔滨 150001)

摘 要: 为了解决目前可重构编译技术在为类仿射型数组下标应用生成循环流水阵列时,生成的存储系统对数据并行与重用支持不完善的问题,本文提出了一种参数化并行存储结构模板. 此模板采用模块化设计思想,根据数据访存特征生成由多体交叉并行存储子模块、单体串行存储子模块、RAW Buffer 缓存子模块及 Smart Buffer 缓存子模块构成的存储结构. 为灵活生成存储结构及充分挖掘数据的并行性和重用性,本文采用访存数据依赖图方法计算存储模板的参数值. 和相关工作相比,根据本文提出的存储结构模板生成的硬件,可以在占用较少的硬件资源情况下,获得较高的硬件执行速度.

关键词: 类仿射数组下标; 可重构编译; 存储结构; 数据重用; 模板
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2016)08-1956-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.08.026

A Parameterized Parallelism Memory Template for Affine Array Subscript Application

GUO Zhen-hua, WU Yan-xia, ZHANG Guo-yin, DAI Kui
(*College of Computer Science and Technology, Harbin Engineering University, Harbin, Heilongjiang* 150001, *China*)

Abstract: In current reconfigurable compiling approach for solving affine subscript operations, the automatic generated feeding memory system is not optimal, especially to support an iteration pipeline structure. This paper presents a parameterized parallel memory template to mine parallelism and reusability of data, which is considered to address the lack of such aspect in reconfigurable compilers at hand. According to the analysis of characteristics of data access to affine subscript arrays in pipeline iteration, our template configures alternative sub-structures such as parallel multi-bank memory, sequential access memory, RAW Buffer and Smart Buffer. Furthermore, in phase of calculating parameter values to fill the template, the memory data dependence graph method is used, in which approach the flexibility of way to create memory structure is kept. The experimental result shows that compared with related works, the compiler can generate reconfigurable hardware performing a higher execution speed with less resources usage by employ the proposed memory template.

Key words: affine array subscript; compiler for reconfigurable computing; memory architecture; data reuse; template

1 引言

目前,一些国内外大学、科研机构及工业界针对循环流水中的并行存储结构技术已经进行了初步的相关研究,但是,这些编译器在为类仿射数组下标的应用生成硬件存储结构时,对数据重用及并行支持的还不够完善. 其中,GarpCC[1]、Nymble[2]、NAPA C[3]只为循环流水阵列生成了提供数据的单体串行存储结构;PACT[4]和 SPC[5]没有考虑循环流水阵列中数据重用问题;CHiMPS[6,7]主要采用多 Cache 结构提供并行访问数据,在提升程序性能的同时明显地增加了硬件面积开销,尤其当程序中存在循环迭代间流依赖时,生成的硬件存储结构会出现 Cache 一致性的问题;Vivado HLS[8,9]主要研究如何为类仿射数组下标应用自动生成并行存储体及防止数据并行访问冲

收稿日期:2014-12-01;修回日期:2015-03-10;责任编辑:马兰英
基金项目:国家自然科学基金(No. 61003036);计算机体系结构国家重点实验室开放课题(No. CARCH201301);博士后科研启动基金(No. LBH-Q12134);中央高校基本科研业务经费专项基金(No. HEUCF100606)

突的调度方法;ROCCC[10]、DEFACTO[11]针对滑动窗口类特殊应用提出的存储结构模板无法用于循环迭代间流依赖的应用;国防科技大学的窦勇教授项目组针对 ROCCC 中存在的问题提出了优化后的参数化三层存储体系结构模板[12],但此模板的并行性还存在可以改进的空间,并且无法为迭代间流依赖生成存储结构. 针对目前可重构编译器在为类仿射数组下标的应用生成硬件存储结构时的不足之处,本文主要探讨如何为只有层内数据重用的仿射类数组下标应用编译生成高效的模块化多层次存储结构. 并在 ASCRA(Application-Specific Compiler for Reconfigurable Architecture)编译器[13]中进行了相关技术实现与验证.

本文的主要创新点如下:

(1)提出了一种面向类仿射型数组下标应用的参数化并行存储结构模板. 采用多体交叉并行存储结构能够有效提高循环中数据的并行性,采用 RAW Buffer 和 Smart Buffer 结构能够提高循环中数据重用性,该模板可以在消耗较少硬件资源的情况下获得较高的硬件执行速度.

(2)提出了针对参数化并行存储结构模板的模板参数自动生成算法. 通过访存数据依赖图生成算法自动计算模板参数信息,提高了可重构编译器实现类仿射型数组下标应用到异构加速平台上的部署效率.

2　类仿射型数组下标应用定义

类仿射型数组下标应用是指参与循环运算的数组元素下标为类仿射型应用,如定义 1 所示.

定义 1　类仿射型数组下标应用.

在循环程序中,如果数组每维的下标都具有 $ai_n + c$ $(i_{n-1},\cdots,i_2,i_1)$ 的形式,其中 a 为整数,n 为循环嵌套层数,$i_1,i_2,\cdots,i_n$ 为循环索引变量,$c(i_{n-1},\cdots,i_2,i_1)$ 为由 $i_1,i_2,\cdots,i_{n-1}$ 所构成的函数(若 $n=1$,$c(i_{n-1},\cdots,i_2,i_1)$ 为常数),则称该循环程序为类仿射型数组下标应用.

3　参数化并行存储结构模板

本文针对只有层内数据重用的仿射类数组下标应用程序,提出了参数化并行存储模板结构,结构如图 1 所示.

根据数组数据依赖关系,生成基于 RAM 的多体交叉并行存储结构,为流水线提供并行的输入数据读取(Load)与输出数据写回(Store)操作. 同时,为复用数据生成 RAW Buffer、Smart Buffer 结构,专用于存储需要复用的数据,消除对 RAM 的访问冲突,保证流水线的效率. 根据需要访存的复用数据依赖类型生成不同的缓存结构:为输入依赖的复用数据生成 Smart Buffer 缓存结构,为循环迭代间的流依赖(写后读相关)的复用数据自动生成 RAW Buffer 缓存结构. 其中 Smart Buffer 直接服务于运算单元,RAW Buffer 和 RAM 为运算单元和 Smart Buffer 服务. 灵活匹配生成由多体交叉并行存储结构、单体串行存储结构(并行度为 1 的多体交叉并行存储结构)、RAW Buffer 缓存结构及 Smart Buffer 缓存结构组成的存储体系.

4　模板参数生成算法

本节主要论述自动生成并行存储硬件结构过程中的难点问题,即数组数据访存特征描述方法及各子模块参数值计算方法.

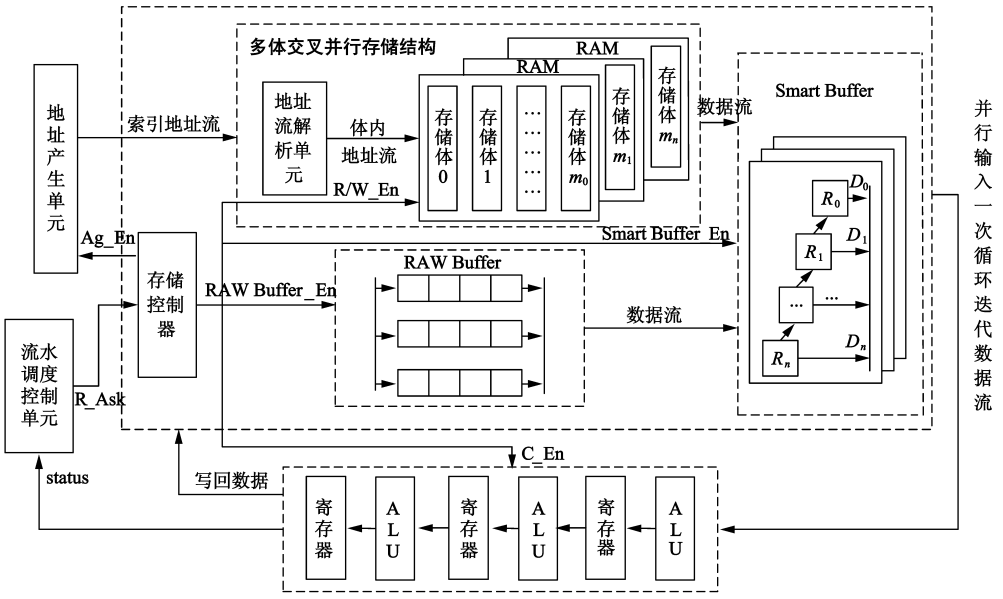


图1　参数化并行存储结构模板

4.1 访存数据依赖图生成算法

定义 2 访存数据依赖图.

访存数据依赖图 MDDG = (V,E,R),其中 V 为节点集合,E 为连接相邻节点的有向边,R 为数据重用度.

$XA_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]} \in V, X = \{L, S\}$,即循环迭代空间内数组元素 $A_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]}$ 进行了 Load 或 Store 操作,每个 $e = (XA_{[ai_n+c(i_{n-1},\cdots,i_2,i_1)]}, XA_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]}) \in E$ 存在一个数据重用度 $R < XA_{[ai_n+c(i_{n-1},\cdots,i_2,i_1)]}, XA_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]} > \in \mathbb{Z}^n$,表示节点 $XA_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]}$ 每隔 M($R < XA_{[ai_n+c(i_{n-1},\cdots,i_2,i_1)]}, XA_{[ai_n+d(i_{n-1},\cdots,i_2,i_1)]} > = M$)次循环迭代就会同节点 $A_{[ai_n+c(i_{n-1},\cdots,i_2,i_1)]}$ 发生对同一存储地址的读写.

本文提出的访存数据依赖图生成算法描述如算法 1 所示.

算法 1 访存数据依赖图生成算法

输入:循环展开后的 IR 指令集

输出:访存数据依赖图 $MDDG_{xamnv}$ 集合

步骤描述:

(1)遍历程序,对其中具有访存特征的数组进行分类. 首先,根据数组名将数据元素划分到不同的集合 G_x 中,其中,$x = \{1,2,3,\cdots,n\}$,n 表示程序中共存在 n 个不同名称的数组元素;其次,根据数组的访存操作属性进行数组元素名称修改;最后,根据修改名称后的数组元素对集合 G_x 进行更新,执行步骤(2).

(2)遍历集合$\{G_x \mid x = \{1,2,3,\cdots,n\}\}$,对集合 G_x 中数组元素按照数组元素下标斜率进行分类. 首先,如果进行 Store 操作的所有数组元素 $SA_{[ai+b]}$ 具有相同的下标斜率,或者不存在进行 Store 操作的数组元素,则将集合 G_x 中数组元素按不同下标斜率划分到不同集合 S_{xa} 中,其中,$a = 1\cdots N$,表示不同的数组下标斜率,并备份 N 份访存数据,执行步骤(3). 然后,如果数组元素 $SA_{[ai+b]}$ 存在不同的下标斜率 a 时,根据下标斜率的不同将集合 G_x 中数组元素划分到不同集合 S_{x0m} 中,其中,$m = \{a\}$,执行步骤(4).

(3)建立集合 S_{xa} 中数组元素下标截距模数据步长 k 同余等价关系 R. 其中,数据步长 k 表示每次循环迭代时数组元素增加的地址间隔,$k = a * Stepi$,$Stepi$ 表示循环步长. R 为集合 S_a 上的等价关系,对任何数组元素 $XA_{[ai+b]} \in S_{xk}$,集合$[XA_{[ai+b]}]R = \{x \mid x \in S_{xa}, ARx\}$,$[XA_{[ai+b]}]R$ 简化表示为 S_{xam},其中,$m = 1\cdots k-1$,表示模数据步长 k 的余数,则 $S_{xam} \subseteq S_{xa}$. 执行步骤(4).

(4)对集合 S_{xam} 或 S_{x0m} 中的数组元素进行写后读相关处理. 首先,对每个集合 S_{xam} 或 S_{x0m} 中的数组元素 $XA_{[ai+b]}$ 按类仿射数组下标的截距从小到大进行排序;其次,判断是否存在迭代间写后读相关,如果存在,则从集合 S_{xam} 或 S_{x0m} 中删除写后读相关中的 Load 操作的数组元素;然后,对集合 S_{xam} 或 S_{x0m} 中的数组元素根据类仿射数组下标的截距从小到大重新排序,如果存在进行 Load 操作的数组元素与进行 Store 操作的数组元素具有相同数组下标情况,将进行 Store 操作的数组元素排在进行 Load 操作的数组元素前面;最后,寻找集合 S_{xam} 或 S_{x0m} 中具有相同数组下标的进行 Load 操作的数组元素与进行 Store 操作的数组元素,判断两者之间是否存在迭代内写后读相关,如果存在,则从集合中删除进行 Load 操作的数组元素. 执行步骤(5).

(5)构建用于计算数据重用度的数组元素集合 S_{xamn} 或 S_{x0mn}. 遍历集合 S_{xam} 或 S_{x0m} 中的数组元素,每当遍历到进行 Store 操作的数组元素时,都将其及之前的所有数组元素构成一个新的集合 S_{xamn} 或 S_{x0mn}($n = 1\cdots N$,表示第 n 个新集合). 按此规则继续遍历剩余的数组元素,直到遍历完集合中的所有数组元素,如果不存在进行 Store 操作的数组元素,则剩余的数组元素构成一个新的集合. 执行步骤(6).

(6)计算数组元素之间的数据重用度. 如果集合 S_{xamn} 中只有一个数组元素,则不用计算数据重用度;否则,根据公式(1)计算集合 S_{xamn} 或 S_{x0mn} 中每两个相邻数组元素之间的数据重用度 $R < XA_{[ai+b]}, XA_{[ai+c]} >$,其中,$\Delta d < XA_{[ai+b]}, XA_{[ai+c]} >$ 表示一次循环迭代内数组元素之间的距离:$|c-b|$;如果公式(1)中不能整除,$R < XA_{[ai+b]}, XA_{[ai+c]} > = 0$;执行步骤(7).

$$R < XA_{[ai+b]}, XA_{[ai+c]} > = \Delta d < XA_{[ai+b]}, XA_{[ai+c]} > / k \quad (1)$$

(7)根据数据重用度继续对集合进行划分. 依次遍历数据重用度 R,如果存在 $R < XA_{[ai+b]}, XA_{[ai+c]} > = 0$,则将 $XA_{[ai+b]}$ 及之前的所有数组元素构成一个新的集合 S_{xamnv} 或 S_{x0mnv}($v = 1\cdots N$,表示第 v 个新集合);按此规则继续遍历剩余的数组元素,直到遍历完集合中的所有数组元素;如果不存在数据重用度 $R = 0$ 的情况,则集合 S_{xamn} 或 S_{x0mn} 中剩下的数组元素构成一个新的集合 S_{xamnv} 或 S_{x0mnv}. 执行步骤(8).

(8)生成访存数据依赖图 $MDDG_{xa\,mnv}$ 集合. 集合 $S_{xa\,mnv}$ 或 S_{x0mnv} 中的数组元素为访存数据依赖图 $MDDG_{xa\,mnv}$ 或 $MDDG_{x0\,mnv}$ 中的节点,其中,第一个数组元素为根节点,从根节点开始依次指向其后序节点,最后一个元素为叶子节点,每条边上的权值为数据重用度 $R < XA_{[ai+b]}, XA_{[ai+c]} >$,生成的访存数据依赖图的个数等于集合 $S_{xa\,mnv}$ 或 S_{x0mnv} 的个数.

4.2 计算模板参数

根据访存数据依赖图,为每个不同名的数组生成参数化并行存储结构所需参数值:(1)多体交叉存取度:Ram_num;(2)存储体深度:Ram_depth;(3)存储体位宽:Ram_width;(4)RAW Bufffer 个数:$RBuffer\_num$;(5)RAW Bufffer 深度:$RBuffer\_depth$;(6)Smart Buffer 个数:$SBuffer\_num$;(7)Smart Buffer 深度:$Register\_num$. 设输入的数组元素个数 $Arrary\_depth$;输入的数组元素位宽 I_width.

(1)多体交叉存储度 Ram_num

生成访存数据依赖图过程中,如果集合 S_{xam} 的个数为 n,或者集合 S_{x0m} 的个数为 m,计算 RAM 的多体交叉存储度 Ram_num 的公式如(2)所示. 当 Ram_num 等于 1 时,硬件结构为单体串行结构.

$$Ram\_num = \begin{cases} 1, & m > 0 \\ n, & m = 0 \end{cases} \quad (2)$$

(2)存储体深度 Ram_depth

计算方法如式(3).

$$Ram\_depth = Arrary\_depth \;/\; Ram\_num \quad (3)$$

(3)存储体位宽 Ram_width

计算方法如式(4).

$$Ram\_width = I\_width \quad (4)$$

(4)RAW Buffer 结构个数 $RBuffer\_num$

如果访存数据依赖图集合中存在流依赖的访存数据依赖图有 f 个,根据参数化并行存储模板规则,为重用数据生成 RAW Buffer 结构. 其 RAW Buffer 结构的个数等于有流依赖的访存数据依赖图中的叶子节点的个数 f,如式(5).

$$RBuffer\_num = f \tag{5}$$

(5)RAW Bufffer 深度 $RBuffer\_depth$

为具有流依赖的输入数据设计 RAW Buffer 结构,每个 RAW Buffer 结构的深度由进行 Store 操作的数组元素所在数据通路中的流水段号(Ln)同访存数据依赖图中叶子节点($SA_{[ai+d]}$)和其相连节点($LA_{[ai+b]}$)间的数据重用度 $R < LA_{[ai+b]}, SA_{[ai+d]} >$ 的关系来确定,如式(6).

$$RBuffer\_depth = \begin{cases} 0, & Ln \geq R \\ R < LA_{[ai+b]}, SA_{[ai+d]} >, & Ln < R \end{cases} \tag{6}$$

式(7)中 $RBuffer\_depth = 0$ 表示直连线,不生成寄存器.

(6)Smart Buffer 结构个数 $SBuffer\_num$

Smart Buffer 的个数等于访存数据依赖图中划分出的集合个数 S,如式(7).

$$SBuffer\_num = S \tag{7}$$

(7)Smart Buffer 结构深度 $Register\_num$

在含流依赖的访存数据依赖图中,其叶子节点生成的数据存储在 RAW Buffer 中,非叶子节点中所需的数据从 Smart Buffer 中读取,此时 Smart Buffer 的深度如式(8)所示.

$$Register\_num = \sum R^{xkmnv} - RBuffer\_depth + 1 \tag{8}$$

其中, $\sum R^{kmnv}$ 表示集合 $S_{xkmnv}(v = v_0, \cdots, v_n)$ 对应的访存数据依赖图中相邻节点数据重用度之和.

在含输入依赖(或只有单节点)的访存数据依赖图中,其叶子节点数据需要从并行存储结构中读取,并行读取的数据需要存储到相应 Smart Buffer 中,此时 Smart Buffer 的深度如式(9)所示.

$$Register\_num = MAX(\sum R^{xkmnv_0}, \sum R^{xkmnv_1}, \cdots, \sum R^{xkmnv_n}) + 1 \tag{9}$$

5　实验与性能比较

目前和本文研究内容接近的主要包括 CHIMPS 编译器生成的硬件存储结构[7]和窦勇教授提出的三层存储结构模板(DY)[12]. 本文采用 5-tap FIR 测试用例(其问题复杂度为 4096,数据输入\输出宽度为 32bits),将三种方法进行对比. 表 1 中 CHIMPS 的数据是根据文献[7]提到的方法计算生成,DY 中的数据根据文献[12]提出的模板公式计算生成.

通过表 1 中实验结果可以得知,针对类仿射型数组下标应用来说,与 CHIMPS 相比,随着程序循环并行度的增加,采用本文所提出的参数化并行存储结构模板自动生成的存储结构中,所消耗的 RAM 硬件资源具有明显的优势,并且在缓存方面也优于 CHIMPS;与 *DY* 相比,虽然在 RAM 资源整体消耗上,两者是相同的,但是,本文所提方法增加了并行的 RAM 个数,并且降低了 RAM 的深度,这种设计方式能够有效的提高存储结构的数据访问的并行度和复用度,降低 RAM 访存冲突,尤其是当程序可以同时读取多个 RAM 中数据时,本文具有明显的优势,此外,本文采用增加缓存高度的方式提高了层内循环数据重用性.

表 1　存储结构对比

测试用例	存储结构	循环展开度				
		1	2	4	8	16
CHIMPS	RAM 个数	2	4	8	16	32
	RAM 深度	4096	4096	4096	4096	4096
	RAM 字宽(bit)	32	32	32	32	32
	缓存高度	5	6	8	12	20
	缓存宽度	1	1	1	1	1
DY	RAM 个数	1	1	1	1	1
	RAM 深度	4096	4096	4096	4096	4096
	RAM 字宽(bit)	32	32	32	32	32
	缓存高度	1	1	1	1	1
	缓存宽度	5	6	8	12	20
ASCRA	RAM 个数	1	2	4	8	16
	RAM 深度	4096	2048	1024	512	256
	RAM 字宽(bit)	32	32	32	32	32
	缓存高度	1	2	4	8	16
	缓存宽度	5	3	2	2	2

表 2 为 5-tap FIR 测试用例采用三种不同方法生成硬件的执行时间随循环展开度的变化情况,从表 2 中可以看出,三种方法生成的硬件执行频率相当,但随着循环展开次数的不断增加,由于 DY 提出的方法没有为应用生成并行存储结构,因此,数据访存开销较大,增加了程序的执行时间,相比之下,本文生成的硬件执行时间和 CHIMPS 生成的硬件执行时间较短,当循环程序的并行度增加时,对程序循环并行性具有更好的适应性,能够提升程序的性能.

结合表1和表2结果可以得知,与CHIMPS相比,本文所提的模板在性能相当情况下具有明显的资源优势,而与DY所提方法相比,在消耗资源相当情况下,具有明显性能优势,结合了两者的优势,从而实现对类仿射型数组下标应用的存储结构进行了改进.

表2 硬件的执行时间随循环展开度的变化情况

测试用例	循环展开次数	频率(MHz)	周期数	执行时间(μs)
CHIMPS	1	151.550	4099	27.047
	2	151.550	2051	13.533
	4	151.550	1027	6.777
	8	151.550	515	3.398
	16	151.546	259	1.709
DY	1	151.550	4104	27.080
	2	151.293	4105	27.133
	4	149.981	4107	27.383
	8	151.438	4111	27.146
	16	152.117	4119	27.078
ASCRA	1	151.550	4104	27.080
	2	151.293	2054	13.576
	4	149.981	1029	6.861
	8	151.438	517	3.414
	16	152.117	261	1.716

6 结束语

本文针对类仿射数组下标应用,提出了一种参数化并行存储结构模板.该存储模板结构不仅为输入依赖的复用数据生成Smart Buffer缓存结构,还为循环迭代间的流依赖的复用数据生成RAW Buffer缓存结构.实验表明,本文提出的存储模板具有较高的灵活性,可以为不同应用生成不同结构的存储系统,较好的支持了数据的重用及并行访问,同时,在为循环展开程序生成存储结构时,和相关工作相比,在占用较少的资源情况下,可以获得较高的硬件执行速度,从整体上提高了程序性能.

参考文献

[1] Callahan T. Kernel formation in garpcc[A]. Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines[C]. Napa Valley, CA, USA, 2003. 308 – 309.

[2] Huthmann J, Liebig B, Oppermann J, et al. Hardware/software co-compilation with the Nymble system[A]. 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)[C]. Darmstadt, Germany, 2013. 1 – 8.

[3] Gokhale M B, Stone J M. NAPA C: Compiling for a hybrid RISC/FPGA architecture[A]. 1998 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines[C]. Napa Valley, CA, USA, 1998. 126 – 135.

[4] Alex J, Debabrata B, Sartajit P et al. PACT HDL: A C compiler targeting ASICs and fPGAs with power and performance optimizations[A]. Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 02[C]. Grenoble, France, 2002. 188 – 197.

[5] Metzgen P. Software-to-hardware compiler: U. S. Patent 8, 473, 926[P]. 2013-6-25.

[6] Seung J. Lee, David K. Raila, Voelodymyr V. Kindratenko. LLVM-CHiMPS: Compilation environment for FPGAs using LLVM compiler infrastructure and CHiMPS computational model[A]. Proceedings of 4th Annual Reconfigurable Systems Summer Institute[C]. Urbana, USA, 2008. 1 – 10.

[7] Putnam A, Bennett D, Dellinger E, et al. CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures[A]. 2008 IEEE International Conference on Field Programmable Logic and Applications[C]. Heidelberg, 2008. 173 – 178.

[8] Monson J, Wirthlin M, Hutchings B L. Implementing high-performance, low-power FPGA-based optical flow accelerators in C[A]. 2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)[C]. Washington, DC, 2013. 363 – 369.

[9] Navarro D, Lucia O, Barragan L A, et al. High-level synthesis for accelerating the FPGA implementation of computationally demanding control algorithms for power converters[J]. IEEE Trans. Industrial Informatics, 2013, 9(3): 1371 – 1379.

[10] Villarreal J, Park A, Najjar W, et al. Designing modular hardware accelerators in C with ROCCC 2.0[A]. 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)[C]. Charlotte, NC, 2010. 127 – 134.

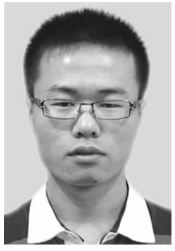
[11] Diniz P, Park J. Automatic synthesis of data storage and control structures for FPGA-based computing engines[A]. Proceedings of 8th IEEE Symposium on Field – Programmable Custom Computing Machines[C]. Napa Valley, CA, 2000. 91 – 100.

[12] 窦勇,董亚卓,徐进辉,等. 基于参数化存储结构的滑动

窗口 IP 核自动生成[J]. 软件学报,2009,20(2):246 -255.
Dou Y,Dong YZ,Xu JH et al. Automatic generation of IP core for sliding-window operations based on a parameterized memory architecutre[J]. Journal of Software,2009,20(2):246 -255. (in Chinese)

[13] 吴艳霞,顾国昌,孙延腾,等. 面向应用的可重构编译 ASCRA[J]. 计算机科学与探索,2011,5(3):267 -279.
Wu Yanxia, Gu Guochang, Sun Yanteng et al. Application-specific compiler for reconfigurable architecture ASCRA[J]. Journal of Fronotiers of Computer Science & Technology,2011,5(3):267 -279. (in Chinese)

作者简介



郭振华　男,1988 年生于河北省,2010 年获哈尔滨工程大学学士学位,现为哈尔滨工程大学计算机科学与技术学院博士研究生. 主要研究方向为计算机体系结构、可重构编译、信息安全.
E-mail:hrbeu. guozhenhua@ gmail. com

吴艳霞(通信作者)　女,1979 年生于哈尔滨,2008 年获得哈尔滨工程大学博士学位,哈尔滨工程大学副教授,硕士生导师,研究方向为计算机体系结构、可重构编译、信息安全.
E-mail:wuyanxia@ hrbeu. edu. cn