

结合片上数据布局和数据传输效率： 一种针对 multi-bank CGRA 的优化算法

刘延涛, 赵仲元, 绳伟光, 何卫锋

(上海交通大学 电子信息与电气工程学院, 上海 200240)

摘 要: 多 bank 的粗粒度可重构处理器结合了高性能和高灵活性的优点, 但是现有的编译技术难以充分利用其多 bank 存储资源, 本文在同时考虑数据放置和数据传输效率的基础上, 提出了一种可以从优化整体运算性能的一种算法, 并建立模型检验优化效果, 结果显示, 优化后的可重构处理器效率是现有优化算法的 2.74 倍。

关键词: 可重构处理器; 数据放置; 数据传输; 优化算法

中图分类号: TN492

文献标识码: A

文章编号: 1000-7180(2018)07-0013-06

Joint Data Placement and Transformation: An Optimizing Algorithm for Multi-bank CGRA System

LIU Yan-tao, ZHAO Zhong-yuan, SHENG Wei-guang, HE Wei-feng

(School of Microelectronics, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: Multi-bank Coarse-grained reconfigured processor combines the advantages of high performance and high flexibility, but the existing compiler technology is difficult to make full use of the Multi-bank memory resources, this paper proposes an efficient optimizing algorithm on optimizing the data placement and efficiency of data transformation simultaneously, then we build a model to evaluate the performance, the result shows our approach is capable of optimizing the performance to 2.74x of the state-of-the-art method.

Key words: reconfigured processor; data placement; data transformation; optimizing algorithm

1 引言

粗粒度可重构处理器(Coarse-Grained Reconfigured Architecture, CGRA)是一种新的处理器结构^[1-2], 其结合了通用处理器的灵活性和专用集成电路的高性能的优点, 通常 CGRA 运算系统由 CGRA 加速器和主处理器组成, 程序中控制密集的部分会映射到 CGRA 上, 然后主处理器处理相关部分, 为了高效地将循环映射到 CGRA 上, 文献[3-5]中的 CGRA 编译器使用流水技术, 这种技术基于循环和指令级并行, 然而, 存储器带宽的限制了并行存储的效率, 随着技术发展, multi-bank CGRA 的出现提供了解决并行化存储访问的硬件基础, 同时这种硬

件也存在一些限制, 首先是片上 bank 的数目, 由于芯片功耗的限制, bank 的数目不会过多, 过多的 bank 会造成芯片功耗的大量消耗. 除此之外, bank 的数目也会受到芯片尺寸的限制. 因此在使用 bank 存取数据时, 就会发生存取冲突, 从而造成额外的延时. 在这种情况下, 一种可以充分且高效利用其多 bank 资源的编译算法必不可少.

已经出现的一些数据放置技术试图将数据分别放到不同的 bank 中来最大程度地弱化程序流水技术带来的访存冲突^[6-7], 却忽略了传输代价的提高. 因此在实际检测过程中, 整体的性能在大部分情况下不会有明显提升甚至会出现明显下降的情况. 基于这种情况, 本文提出了一种关注整体性能优化的

启发式算法. 这种算法与之前算法的不同之处在于本文算法结合考虑了数据传输效率和片上数据计算效率. 同时为了量化 multi-bank CGRA 的性能和功耗, 本文建立了一个运行模型, 这个模型可以反映出整体 CGRA 运算系统的性能, 既模拟了主存到多 bank 存储系统的数据传输, 也提供了 CGRA 加速器的计算性能.

2 multi-bank CGRA 及其现有优化算法

2.1 multi-bank CGRA 运算系统简介

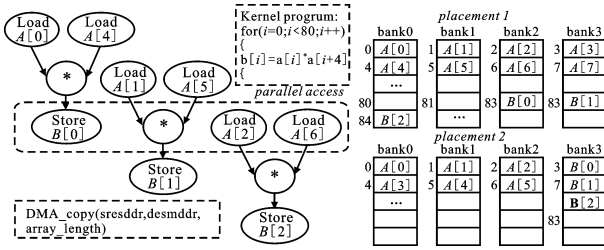
在 multi-bank CGRA 运算系统中, 有两种计算单元(主处理器和 CGRA 加速器)以及三种存储器(主存储器、CGRA 加速器的 multi-bank 存储器、配置存储器). CGRA 由相互连接的 4×4 可重构处理单元组成, 每一个处理单元(PE)可以从它的本地寄存器、multi-bank 存储器以及其他 PE 的输出寄存器中获取数据, 并且在每一个 Control Step 之后, 有一个同步机制来同步所有 PE 的行为, 如果 PE 之间产生冲突或者延时等情况发生, 一个 Control Step 就会延长.

CGRA 计算系统的执行过程如下, 首先主处理器初始化所有的 CGRA 配置信息和主存储器里面的数据. 在 CGRA 加速一个循环之前, 循环的输入数据应当从主存传入到 bank 中, 且其配置信息也应当传入到 CGRA 的配置存储器中, 当 CGRA 运算结束之后, 数据又从 bank 中传输回到主存中. 这些过程的时间消耗有: 从主存和 bank 之间的数据传输时间, 和 CGRA 的配置、计算时间, 据调研, 现有的 bank 优化算法仅仅考虑了 CGRA 计算性能的优化, 完全忽略了数据传输的额外开销, 经过前期测试, CGRA 整体性能提升不大甚至有所降低.

2.2 基于 multi-bank CGRA 的优化算法

基于 multi-bank 的优化算法都能够充分利用 CGRA 的存储器资源, 文献[8-9]中的无冲突映射算法使用 force-directed 方法, 这种调度方法在循环的流水执行过程中优化了并行执行时产生的冲突, 其数据放置算法主要思想是将同时会使用的数据分散到不同的 bank 中, 针对 CGRA 的 Conflict-free 映射算法有效的减少了 CGRA 计算时的访存冲突, 并且每个 Control Step 里面的 cycle 数也有了明显的减少, 其结果是计算时间明显减少.

图 1 显示了数据放置算法的一个实例, 循环以一种嵌套流水的方式被执行, 其中冲突的来源主要有两个, 一个是单个数据向量 A 内部另外是来自于



(a)程序流水执行图例 (b)两种数据放置策略下的 multi-bank 上的数据组织形式

Placement method	Data transform	Multi-bank conflict
Placement 1	DMA_copy(A,0,total_length), DMA_copy(B,80, total_length)	1. for array A, A[i] and A[i+4] conflict in every control step 2. for array A and B, B[i], A[i+2] and A[i+6] conflict in every control step
Placement 2	DMA_copy(A,0,3), DMA_copy(A+3,4,3),... DMA_copy(B+3,1), DMA_copy(B+1,7,1),...	No conflict inside array A, and between array A and B during any control step

(c)两种数据放置策略下的数据传输和流水执行

图 1 两种数据放置策略的效率对比

两个不同的数据向量 A 和 B 之间. 如果我们使用数据放置算法 1 来放置 A 和 B, 冲突在每一个 control step 都会产生, 在每次循环下标为 i 的情况, A[i] 和 A[i+4] 被放置在同一个 bank 中, 所以就会分两次获取这两个元素, 而且, 由于基于调度算法的软件并行化, B[i] 被访问同时, A[i+2] 和 A[i+6] 也被访问, 因为这几个元素在相同的 bank 中, Control Step 就会增加延时, 然而, 可以看到这两个数据向量在 bank 存储系统中是连续存放的, 因而数据在主存和 bank 中的传输效率却是最高的, 这是因为 DMA 只会被初始化两次, 而 conflict-free 算法会产生另外一种数据放置方式, 这里称为方式 2, 在每个 Control Step 中的冲突都会消除, 但是带来的影响就是数据传输效率的下降, 这也极大的影响了整体的性能, 因此本文的目的就是找出一种数据放置算法可以同时考虑数据传输效率以及数据运算性能, 实现整体性能的优化

3 构建性能模型

本文基于图 2 中的硬件结构构建性能模型.

在构建性能模型之前, 首先将 multi-bank 存储系统上的数据的位置分布用数学化的方式进行表示以方便后续的建模, 具体表示方式如下:

考虑一个有限的 n 维数据向量 A, 在 A 中任何一个元素的地址可以表示为 $A(\vec{x}) = (x_0^A, x_1^A, \dots, x_n^A)$ 如果使用线性传输向量 $\vec{\alpha} = (\alpha_0, \alpha_1, \dots, \alpha_{n-1})$ 来转换多维向量到一维向量, 则 $A(\vec{x})$ 在 bank 存储系

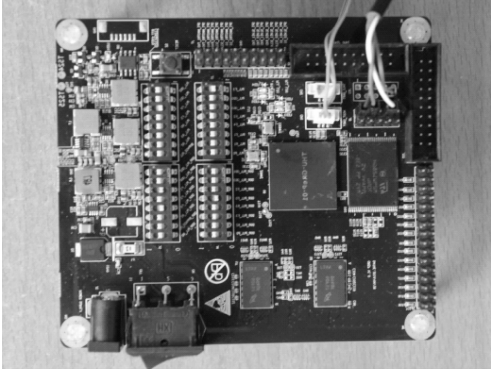


图2 CGRA 运算系统的测试芯片,包括一个主处理器,16个 bank 的片上存储器以及一个主存储器

统中的地址可以表示为:

$$\text{Adr}(A(\vec{x})) = I_A + \vec{\alpha} \cdot \vec{x} \quad (1)$$

I_A 是数据向量 A 在 multi-bank 中的存储系统的偏移地址,等式表明:如果在一个数据向量内部,元素地址之间的差值不会受到 A 的偏移地址的影响,然而处在不同 bank 向量中的数据 $A(\vec{x}_1)$ 和 $B(\vec{x}_2)$,就会受到 A 和 B 的偏移地址的影响。

经过仔细考虑我们将总体的运行时间分为三部分,如式(2)所示:

$$T_{\text{total}} = T_{\text{cgra}} + T_{\text{invoke}} + T_{\text{trans}} \quad (2)$$

式中, T_{cgra} 是 CGRA 加速时间,取决于调度和映射算法, T_{invoke} 是 CGRA 的调用和重新配置的时间,调用和重新配置主要包括准备 CGRA 的相关参数,初始化 CGRA 的配置信息,虽然 CGRA 在程序执行期间动态重构,CGRA 的准备开销仍然不可以被忽略, T_{trans} 则取决于 DMA 控制器的效率,过多的 DMA 准备时间将会极大的降低数据传输的效率, T_{cgra} 和 T_{trans} 彼此是负相关的,从多个向量中获取数据或者说从单个向量中获取多个元素时就可能会产生冲突,因此数据放置算法会把这些数据放在不同的 bank 中,然而这样会产生数据传输之间的开销,而如果连续将这些元素放置,就可能会产生过多的运算冲突,因此 T_{cgra} 被优化的同时, T_{trans} 就会恶化。所以本文的主要思路就是平衡 T_{cgra} 和 T_{trans} ,使总体达到最优。具体的模型构建主要分为三部分:

(1) CGRA 的计算时间模型

CGRA 计算时间是由三个因素决定的:

①具体的程序代码及其数据流图(Data Flow Graph, DFG)的具体形式,根据数据流图和具体的程序,就可以计算出这个程序全部执行所需要的 Control Step 数量。

②启动间隔 Π 的大小 根据 DFG 我们可以确定一个循环中的算子数目 k ,根据每次计算的最大算子数 32,所以可以确定 $\Pi = 32/k$,之后向上取整即可确定。

③数据在多 bank 存储器上的放置分布,理想情况下,一个 Control Step 包含一个 cycle,但是数据在存储器上的分布会影响数据在存取时候发生的冲突次数,从而使得一个 Control Step 中的 cycle 数增加,每次循环具体发生的冲突次数决定于数据流图的具体形式和数据的在 bank 上的放置分布。

如果给出多 bank 存储器下的数据放置策略 D 以及调度和映射的策略 S ,CGRA 的运算时间就可以计算出来,首先在策略 S 下,软件流水之间的启动间隔 Π 以及总体的 Control Step 数量都可以计算出来,我们用 $T_{\text{cs}}(S)$ 来表示在现有策略 S 下的 Control Step 总数,如式(3)所示:

$$T_{\text{cs}}(S) = L + \Pi \times (N - 1) \quad (3)$$

式中, L 指的是 DFG 中 Control Step 的延时; N 指的是循环的总次数。在数据放置方法策略 D 下和调度映射的策略 S 下的 CGRA 计算时间由式(4)给出:

$$T_{\text{cgra}}(D, S) = \sum_{i=1}^{T_{\text{cs}}(S)} C_i(D, S) \quad (4)$$

式中, $C_i(D, S)$ 是第 i 个 Control Step 所包含的 cycle 数。

(2) 数据传输时间模型

给定数据在多 bank 存储系统的放置分布,我们假设一共有 m 个待传数据向量,这些数据向量一共使用 N_{max} 个 bank,对于其中任何一个数据向量 A_i ,假设使用 $N(i)$ 个 bank,在本文中使用 $T_i(D)$ 表示在数据放置分布策略 D 下面数据向量 A_i 的传输时间。

$$T_i(D) = \begin{cases} 1, & N_i(D) = N_{\text{max}} \\ \frac{L_{A_i}}{N_i(D)}, & \text{else} \end{cases} \quad (5)$$

式中, L_{A_i} 是数据向量 A_i 的长度; $N_i(D)$ 是在数据放置布局 D 下分配给 A_i 的 bank 数目。给出多 bank 存储器上所有数据向量的数据放置方式,总体的数据传输时间就可以计算出来,如式(6):

$$T_{\text{trans}}(D) = H \times \sum_{i=1}^m T_i(D) + \sum_{i=1}^m T_{\text{trans}}(L_{A_i}) + \delta \quad (6)$$

式中, H 是 DMA 调用开销; δ 则是考虑到 cache 失误影响而带来的额外时间,在我们的运行模型里面,我们假定 DMA 调用开销 H 是一个常值,对每一次 cache 失误,只要访问主存的方式相同,不同的数据

放置策略下, δ 不变。

(3) 差异判断

基于式(1)到式(5), 在数据放置策略 D 和调度和映射策略 S 给定的情况下, 可以得出整体运行时间的式(7):

$$T_{\text{total}}(D, S) = \sum_{i=1}^{T_{\text{cs}}(S)} C_i(D, S) + T_{\text{invoke}}(t) + H \times \sum_{j=1}^m T_i(D) + \sum_{j=1}^m T_{\text{trans}}(L_{A_i}) + \delta \quad (7)$$

因此, 我们可以消除相同的不变因素, 只留下可变部分, 从而只需要计算不同数据放置策略下的性能差异来比较不同放置策略下的性能优劣:

$$\text{Diff}(D_p, D_q, S) = \sum_{i=1}^{T_{\text{cs}}(S)} (C_i(D_p, S) - C_i(D_q, S)) + \sum_{j=1}^m H \times (T_j(D_p) - T_j(D_q)) \quad (8)$$

如果 $\text{Diff}(D_p, D_q, S) > 0$, 则表示数据放置策略 D_q 要好于 D_p , 否则 D_p 好于 D_q 。

建立这个模型的目的是为了分析数据放置策略是如何影响 CGRA 运算系统性能, 而且在比较性能之间的差值时, 也不需要在每一种策略下的精确的性能数据, 第一, 模型模拟了 cache 上的动作以及 CGRA 的调用, 这保证了模型的准确性, 第二, 不同的数据放置策略仅仅改变了在 multi-bank memory 上数据的放置布局, 主存中的数据仍然不变, 第三, 数据放置策略并不影响 CGRA 的调用时间。

4 优化算法

优化算法以初始化数据放置策略 D_{ini} 为基础, 在此之上再去寻找其他的数据放置策略 D_{new} , 之后循环往复, 在 D_{ini} 和 D_{new} 进行比较过程中逐步找到最优解。

当 multi-bank 存储器的地址空间很大的情况下, 算法的搜索空间也会很大, 为了减小搜索域, 本文给出了一种启发式的方法来从不同的候选策略中来找出最优的数据放置策略, 使不同的数据向量在 multi-bank memory 上连续放置, 因此我们可以通过改变不同数据向量的偏移, 并且根据性能差异判断方法的优劣, 我们要做的就是从数据传输效率最好的几个候选策略中筛选出一个最好的数据放置策略, 可选的数据放置策略一共有 N^{m-1} 种, 其中 N 是 bank 的总数, m 指的是 kernel 中要用到的数据向量

数, 因为, 对任何 p 和 q , bank 差异 BD 可以计算为:

$$\begin{aligned} \text{BD} &= |(I_B + pN + \vec{\alpha}_B \cdot \vec{x}_2) - (I_A + qN + \vec{\alpha}_A \cdot \vec{x}_1)| \% N \\ &= |(I_B + \vec{\alpha}_B \cdot \vec{x}_2) - (I_A + \vec{\alpha}_A \cdot \vec{x}_1) + (p - q)N| \% N \\ &= |(I_B + \vec{\alpha}_B \cdot \vec{x}_2) - (I_A + \vec{\alpha}_A \cdot \vec{x}_1)| \% N \end{aligned} \quad (9)$$

式(9)表明搜索空间没有必要存储器的全部空间, 只需要对空间按照 bank 进行划分, 并对不同 bank 的情况按照式(9)进行优劣判断. 对 N 取余数, 是用来判断两个元素是否处在相同的 bank 中, 按照之前对搜索空间的划分, 如果两个元素处在相同的 bank 中, 就会产生访问冲突。

找到新的数据放置策略之后, 我们使用 conflict-free 的映射策略与之作比较, 根据性能差异选择更优的数据放置策略, 最终的算法如下所示。

```
function Data Placement( $(A_1, A_2, \dots, A_m)$ )
 $D_{\text{ini}} \leftarrow \text{Place}(A_1, 0), (A_2, 0), \dots, (A_m, 0);$ 
 $D_{\text{old}} \leftarrow D_{\text{ini}}$ 
for  $0 \leq i_1 < N$  do
...
for  $0 \leq i_m < N$  do
 $D_{\text{new}} \leftarrow \text{Place}(A_1, i_1), (A_2, i_2), \dots, (A_m, i_m);$ 
if  $\text{Diff}(D_{\text{old}}, D_{\text{new}}) > 0$  then
 $D_{\text{old}} \leftarrow D_{\text{new}}$ 
else
continue;
end if
end for
...
end for
 $D_{\text{new}} \leftarrow \text{Place Conflict Free Map}()$ 
if  $\text{Diff}(D_{\text{old}}, D_{\text{new}}) > 0$  then
return  $D_{\text{new}};$ 
end if
end function
```

将具有最优数据传输效率的策略作为基准的原因是数据传输所消耗的时间通常占据整体时间的大部分, 所以我们首先考虑数据传输效率, 之后考虑 CGRA 的计算时间。

本文选择了 9 个样本 kernel, 表 1 显示的是这 9 个样本 kernel 的 CGRA 的计算时间和数据传输时间, 我们选择有利于优化数据传输的数据放置策略, 从总体来看数据传输时间占据总时间的 50% 以上, 另外, 如果数据在 bank 中合理放置, CGRA 的计算时间就会得到明显的优化, 因为 CGRA 对数据的访问冲突数量会大幅度削减甚至完全消除, 最后我们也比较了不同策略的性能, 这些策略着重于提高 CGRA 的计算性能, 比如 aes, fil 和 auctor 这三个 kernel, 和传输相比, 其计算时间消耗要多得多, 因

为极大地优化了 CGRA 的计算时间, 总体时间得到了很大的优化.

表 1 CGRA 计算时间和传输时间(cycle)

kernel	execution time	trans time
conven	1 268	1 946
lcs	16 929	10 932
dotp	1 408	5 711
spmv	3 080	8 700
mcf	144	1 111
auctor	88 854	15 056
aes	445 030	16 998
fil	338 758	8 763
karatsuba	121 385	9 987

5 实验结果分析

实验中所采用的样本 kernel 部分来自于 Mi-bench^[10], 我们在样本 kernel 修改后使用图 2 中的硬件来检测算法性能.

5.1 CGRA 计算时间

通过比较上述提到的三种方法下的 CGRA 计算时间, 我们发现 conflict-free 映射算法相对于传输最优算法, 计算效率提高了 2.92 倍, 对于 aes, fil 和 karatsuba 这 3 个 kernel, CGRA 的计算时间要远长于他们的数据传输时间(如表 1 所示, 这三个 kernel 的计算时间几乎相当于他们传输时间的 10 倍), 基于表 2 和表 3, 如果我们比较数据传输和 CGRA 的计算, 可以发现尽管数据放置算法可以优化计算时间, 但是这些优化却会被新增的数据传输代价抵消, 以上数据证明不能优化不能只局部与 CGRA 的计算优化, 同时考虑数据计算优化和数据传输优化的方法是有意义的.

表 2 CGRA 数据传输时间对比(cycle)

kernel	ours	conflict-free	best-trans
conven	1 946	11 931	1 946
lcs	10 932	29 985	10 932
dotp	5 711	19 672	5 711
spmv	8 700	28 864	8 700
mcf	1 111	7 600	1 111
auctor	15 056	104 430	15 056
aes	118 854	118 854	16 998
fil	42 786	42 786	8 763
karatsuba	73 627	73 627	9 987
average	2.8	5.4	1

表 3 CGRA 计算时间对比(cycle)

kernel	ours	conflict-free	best-trans
conven	1 268	1 032	1 268
lcs	16 929	18 992	16 929
dotp	1 408	459	1 408
spmv	3 080	568	3 080
mcf	144	96	144
auctor	88 854	32 432	88 854
aes	16 998	16 998	118 854
fil	21 130	211 320	338 758
karatsuba	43 256	43 256	121 385
average	1.98	1	2.92

5.2 数据传输效率

通过将本文算法, 将无冲突映射算法即计算效率最优的算法和传输效率最优的算法作比较, conflict-free 映射算法将传输代价平均提高了 5.4 倍(表 2), 而我们的算法提高传输代价平均 2.8 倍.

表 4 显示三种策略之下的 DMA 初始化次数比较. 从实验结果来看, 对 dotp, auctor 和 mcf 等 kernel 来说, 无冲突映射算法是低效率的, 因为这种映射算法会造成过多的 DMA 初始化, 极大的提高了数据传输的代价, 而对于计算量的优化不足以抵消这些增加的开销.

表 4 DMA 调用时间对比(cycle)

kernel	ours	conflict-free	best-trans
conven	3	92	3
lcs	257	455	257
dotp	4	179	4
spmv	58	369	58
mcf	4	64	4
auctor	32	705	32
aes	96	96	17
fil	296	296	176
karatsuba	36	36	21
average	1.7	14.5	1

5.3 整体影响

表 5 显示三种方法下的整体性能的比较, 对于测试 kernel, 依据本文的方法对样本 kernel 进行优化后均得到了三种方法中最好的效果, 综合来看, 本文提出的优化算法性能相当于现有优化算法的 2.74 倍.

综合以上各表数据可以看出, 优化数据传输相对整体性能来说效果更为明显, 这是因为这些 kernel 的数据传输耗费明显高于计算耗费, 然而在某些计算量非常大的情况下, 如本文中的 3 个 kernel, 无冲突映射算法也是很有意义的.

表 5 整体运行时间对比 (cycle)

kernel	ours	conflict-free	best-trans
conven	3 214	12 963	3 214
lcs	27 861	68 950	27 861
dotp	7 119	20 131	7 119
spmv	11 780	59 432	11 780
mcf	1 255	7 696	1 255
auctor	103 910	136 862	103 910
aes	295 664	295 664	462 028
fil	254 106	254 106	347 521
karatsuba	131 372	116 883	131 372
average	1	2. 74	1. 1

6 结束语

对于当今的 CGRA 计算系统来说,同时考虑数据传输效率和 CGRA 加速计算是很有意义的,本文的研究工作重点的同时考虑数据传输和 CGRA 加速计算的优化.

数据放置算法很难实现完全自动化,因为优化主存和 bank 缓冲之间的数据传输是极其困难的,且由于样本 kernel 数量的有限,可能存在一定的数据偏差,但最后结论的得出并不影响. 且我们的工作可以给使用者做出一种反馈,并帮助开发人员使用一种更加高效的方式来管理数据.

参考文献:

[1] Karunaratne M, Mohite A K , Mitra T, et al. Hy-CUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect[C]// 2017 54th ACM ED-AC/ IEEE Design Automation Conference (DAC). Austin, TX, USA, 2017:1-6.

[2] Cong J, Huang H, Ma C, et al. A fully pipelined and dynamically composable architecture of CGRA[C]// 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines. Boston, 2014:9-16.

[3] Ramakrishna Rau B. Iterative modulo scheduling: An algorithm for software pipelining loops[C]// Proceedings of the 27th annual international symposium on Microarchitecture. ACM, 1994,63-74.

[4] Bingfeng Mei, Serge Vernalde, Diederik Verkest, et al. Exploiting loop-level parallelism on coarse-grained

reconfigurable architectures using modulo scheduling [C]//In Computers and Digital Techniques, IEEE Proceedings. 2003,150:IET, 255-261.

[5] Ramakrishna Rau V B. Iterative modulo scheduling: An algorithm for software pipelining loops [C]// Proceedings of the 27th annual international symposium on Microarchitecture. ACM, 1994 :63-74.

[6] Shouyi Yin, Zhicong Xie, Chenyue Meng, et al. Multi-bank memory optimization for parallel data access in multiple dataArrays [C]//2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). Austin, TX, 2016:1-8.

[7] Song L, Feng M, Ravi N, et al. COMP: compiler optimizations for many-core processors[C] //2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge. 2014:659-671.

[8] S Yin, Xianqing Yao, Tianyi Lu, et al. Joint loop mapping and data placement for coarse-grained reconfigurable architecture with multi-bank memory [C]// 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). Austin, TX, 2016: 1-8.

[9] Alexander Fell, Zoltan Endre Rakossy, Anupam Chattopadhyay. Force-directed scheduling for data flow graph mapping on coarse-grained reconfigurable architectures [C]// ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on. IEEE, 2014:1-8.

[10] Krste Asanovic, Ras Bodik, Bryan Christopher Cantanzaro, et al. The landscape of parallel computing research: A view from Berkeley[R]. Berkeley, EECS Department, University of California, 2006: U-CB/EECS-2006-183.

作者简介:

刘延涛 男,(1991-),硕士研究生.研究方向为粗粒度可重构处理器编译技术. E-mail: 329282644@sjtu.edu.cn.

赵仲元 男,(1990-),博士研究生.研究方向为可重构计算的编译架构.

绳伟光 男,(1977-),博士,助理研究员.研究方向为高可靠集成电路设计.

何卫锋 男,(1976-),博士,副研究员.研究方向为 SoC 设计与系统集成.