关键循环到粗粒度可重构体系结构的存储感知映射*

杨子煜,赵 鹏,王大伟,李思昆
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:针对已有工作面向粗粒度可重构结构(CGRA)研究循环映射的不足,提出一种新颖的存储感知的关键循环映射方法MALP。该方法定义RCP_CGRA体系结构模型并阐述关键循环到CGRA的映射问题,通过引入结合数组分簇的多面体数据域划分方法进行循环存储分析,根据分析结果,结合体系结构资源约束实现了循环的有效映射。实验结果表明,与已有的方法相比,MALP方法能够快速分析存储需求并有效降低循环映射的资源占用率,提高数据吞吐量,进一步提升了CGRA上循环映射的性能。

关键词:循环映射;存储感知;粗粒度可重构体系结构;数据密集型应用
中图分类号:TP391 **文献标志码**:A **文章编号**:1001-2486(2012)06-0046-08

Critical loop memory-aware mapping onto coarse-grained reconfigurable architecture

YANG Ziyu, *ZHAO Peng*, *WANG Dawei*, *LI Sikun*
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: Targeting at the mapping of key loops onto CGRA (Coarse-Grained Reconfigurable Architectures), this research proposes a novel approach called memory-aware kernel loop pipelining mapping (MALP). The RCP_CGRA model and the critical loop mapping formulation were shown first. Based on polyhedral model, then the array clustering and data domain partition were presented. An analysis of the critical loop storage requirement was described. Based on this analysis result, the MALP provided an efficient way for loop mapping under the resource constraints of CGRA. Experiment results show that MALP can improve the data throughput rate while costing less resource. MALP makes the loop mapping on CGRA more efficient.

Key words: loop mapping; memory-aware mapping; coarse-grained reconfigurable architecture; data-intensive application

音频、视频处理等数据密集型多媒体领域应用程序执行,85%左右的程序执行时间都集中在其中的嵌套循环部分[1]。粗粒度可重构体系结构(Coarse-Grained Reconfigurable Architecture, CGRA)加速这些数据密集型应用时兼具通用计算的灵活性和定制计算的高效性,并已有多种编译优化方法[2]提供支持。但大多数方法均通过复杂的流水转换技术来实现加速[3-4],并未针对这些关键循环带有的频繁数据存储访问进行优化。由于CGRA的存储资源紧缺且成本较高,如何在编译映射过程中分析并利用存储需求信息,进行存储性能优化,已成为影响CGRA加速性能的关键问题之一。

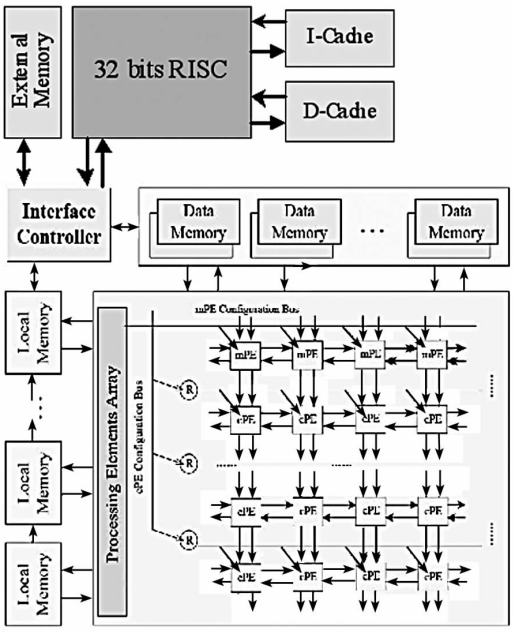
1 问题定义

1.1 粗粒度可重构体系结构模型

为实现面向粗粒度可重构的有效循环转换,本文提出了RCP_CGRA(CGRA资源/约束/性能)模型用以描述典型的粗粒度可重构体系结构信息并提供给后续的循环映射优化过程。图1所示的是一种典型粗粒度可重构体系结构LEAP[10],其CGRA阵列包含 $n \times m$ 个处理单元(PE)。PE是对数据进行处理的功能部件,为实现计算和存储功能分离的目的,PE被分成了两种:计算处理单元(cPE)和存储处理单元(mPE)。其中,m 个mPE与数据存储器(DM)直接连接,专门负责存储访问;cPE仅包含计算功能,$(n-1) \times m$ 个cPE构成了计算阵列。为了参数化描述体系结构的硬件特征,我们提出RCP_CGRA模型。RCP-CGRA从资源模型、性能模型和约束模型三个方面描述了以LEAP为典型代表的CGRA阵列的特征信息。其中,资源模型描述了CGRA中的资源配置,如:cPE和mPE数量、数据存储DM的大小和数量、cPE和mPE配置总线的数据

* 收稿日期:2012-07-11
基金项目:国家自然科学基金资助项目(61076020,61133007)
作者简介:杨子煜(1984—),女,湖南衡阳人,博士研究生, E-mail:zyyang@nudt.edu.cn;
李思昆(通信作者),男,教授,博士生导师,E-mail:lisikun@263.net.cn

带宽等。性能模型描述了 CGRA 在 FPGA 综合后的性能参数,如面积、时钟频率、峰值性能和能耗开销等。约束模型则描述了面向此结构进行循环转换应满足的计算、存储和通信等资源约束条件。




```
RCP_CGRAs{
General{
name        RCP_CGRA_LEAP;
path        "X:/XX/...";
Version     "XX.XX";
}
Resource_model{
    cmodule_number   CPE_NUM m*(n-1);
    mmodule_number   MPE_NUM m;
    lm_size          LM_SIZE lms;
    lm_number        LM_NUM lmn;
    dm_size          DM_SIZE dms;
    dm_number        DM_NUM dmn;
    bus_bandwidth    INPUT_CBW icbw;
                     OUTPUT_CBW ocbw;
                     INPUT_MBW imbw;
                     OUTPUT_MBW ombw;
}
Constraint_model{
    CPE(the number of used CPEs < m*(n-1));
    MPE(the number of used MPEs < m);
    LM(the size of LMij> the access size of MPE, MPE);
    BUS( the access data size of ith row of CPEs <obw);
}
Performance_model{
    AREA();
    CLOCK_RATE();
    PEAK_PERF();
    DYNAMIC_POWER();
 }
}
```

图 1　LEAP 及其 RCP_CGRA 模型

Fig. 1　LEAP and its RCP_CGRA model

1.2　CGRA 的应用计算核心映射问题

由于数据密集型应用对数据并行度和吞吐量有较高要求,通过对关键循环进行优化加速可以有效提高 CGRA 处理该类应用的性能。给定一个数据密集型应用中的关键循环块,该循环块到 CGRA 可重构 PE 阵列的映射过程涉及的主要技术难点在于:(1)如何在 PE 阵列上放置循环中的操作,以实现计算的有效映射;(2)如何合理利用已有 PEs 互连关系实现循环的数据流关系;(3)如何在局部存储与 mPE 互连约束的前提下放置循环的数组变量,实现数据的有效映射。以往针对 CGRA 映射的研究往往假设循环所需数据已存在于局部存储器上且处理单元对数据的访问不受限制。若考虑 CGRA 上的局部存储限制,分析循环的存储需求并最大限度地重用循环数组,则可以突破循环在 cPEs 加速时由于频繁的访存操作而导致的性能瓶颈。

在 RCP_CGRA 硬件模型中,同一行 PE 共享一条总线,相邻两个 mPE 共享一个 DM。利用数据驱动体系结构的优势,RCP_CGRA 运用资源流水技术把对循环的控制移至对存储体操作的控制上,并通过对存储体的操作来控制数据流的流动,从而控制着整个计算的正确运行。图 2 显示了一个典型的迭代间相关的嵌套循环,以及将其映射至 RCP_CGRA 结构上之后的数据流图(DFG)和循环流水时空图。如图 2 右半部分所示,其中

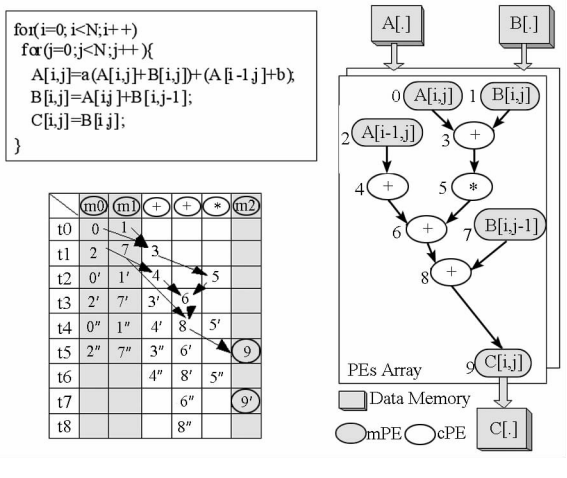


图 2　嵌套循环至 RCP_CGRA 的映射

Fig. 2　Nest loop mapping on RCP_CGRA

mPE 控制数据的流入与流出。图 2 左下的时空图则显示了包含三个 mPE 及两个 cPE 时,循环三次迭代的映射情况,其中箭头表示数据的流向,整数 i ($0 \leqslant i \leqslant 9$)表示操作标号,$i$ 代表第一次迭代,i' 和 i''分别代表第二、三次迭代。从图中可以看到吞吐率达到了每两拍一个结果。进一步地,若通过对循环的存储分析结果调整局部数据存储和 PE 调度方式,在 PE 资源充足且处理单元阵列都被数据流充满时,吞吐率可以达到每拍一个结果,从而极大地提高硬件资源利用率。因此,RCP_CGRA 硬件模型上的计算核心映射问题可转化为以最小资源占用率和最大数据吞吐量为目标的循环映射优化问题。

2 面向 CGRA 的关键循环存储分析

本文采用应用程序特征分析工具[9]分析和统计程序中所有循环的控制流和数据流信息，并通过 LooPo 工具[8]进行数据依赖/存储分析，得到循环的存储需求特征和数据依赖关系。若在应用程序中某些循环的执行时间百分比或存储需求百分比大于某一阈值，则称这些循环为关键循环。传统的 CGRA 编译通常忽略存储限制，不考虑循环数组在 DM 上的放置问题，从而导致对同一数组的载入操作被分配至不同 mPE、同一数据在多个 DM 重复存储的情况，浪费了存储资源，从而降低了硬件计算性能。本文目标即是通过分析关键循环存储需求，解决循环优化时的数据映射问题。

2.1 粗粒度可重构体系结构模型

在循环存储分析中我们首先对循环各数组及其引用情况进行检测，采用多面体模型[11]对循环迭代域构建相关多面体，通过对多面体仿射空间的操作，准确描述数据、数据操作与数据间关系，并对嵌套循环进行控制/数据流建模。图 3 描述了图 2 源代码中数组的多面体二维迭代域及依赖关系。

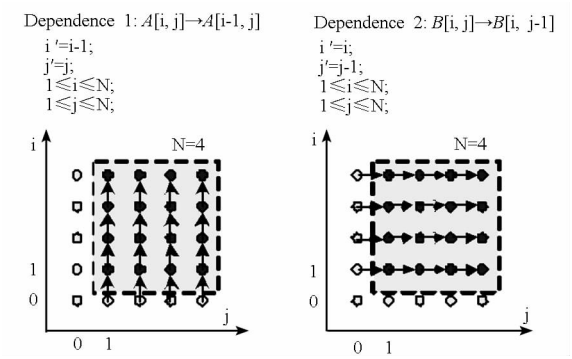


图 3 嵌套循环的多面体依赖关系

Fig. 3 Polyhedral dependents of nest loops

定义 1 解耦合多面体 令数组 A 的 r 次引用和第 r' 次引用的数据域多面体分别为 P_{DAr} 和 $P_{DAr'}$。当且仅当 $P_{DAr} \cap P_{DAr'} = \emptyset$ 时，这两个多面体之间不存在耦合，称为解耦合多面体(dP)。数组的各个引用以该数组引用的索引向量 $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}} \in Z^n$ 的形式表示。不失一般性，假设两个数据依赖的数组引用分别对应于索引向量 x_s 和 x_t，且 x_s 依赖于 x_t。当 $x_s \in \mathrm{dP}_s$ 且 $x_t \in \mathrm{dP}_t$，称 dP_s 依赖于 dP_t。

定义 2 dP 粒度的数据依赖图 dP 粒度的数据依赖图定义为 $G = \{N, E\}$。其中 N 是节点集合，每个节点 $n \in N$ 表示一个 dP。$E = \{e_{ij} \mid n_i > n_j, n_i, n_j \in N\}$ 是有向边的集合，$n_i > n_j$ 表示从节点 n_i 到节点 n_j 的有向边，反映了两节点间的数据依赖关系。

通过程序特征分析工具[9]可以得到循环中数组的特征信息。对于给定循环 L，可以得到其一次迭代中对任一数组 A 的存储访问次数，表示为 Na_L^A。令 Sz_L^A 为数组 A 的大小，若将 A 放置于某一数据存储 DM 上，根据体系结构 RCP_CGRA 模型可知 DM 大小为 dms，因此数据 A 的优先级 Pr_A 和数组 A 的分簇开销 $costa(AS, A)$ 如式(1)所示：

$$\begin{cases} Pr_A = Sz_L^A/dms + \sum_L Na_L^A/II_L \\ costa(AS, A) = Sz_L^A/LS_L^{AS} + \sum_L Na_L^A/Nas_L^{AS} \end{cases} \tag{1}$$

其中 II_L 为循环 L 在后续循环流水化的目标启动间距(Initiation Interval)，AS 为所有已在此 DM 上的数组分簇集合，LS_L^{AS} 为 AS 剩余可放置空间，其上界为 DM 的大小 dms，Nas_L^{AS} 为循环 L 中当前迭代的剩余访问次数，其上界为 II_L。$costa(AS, A)$ 反映了在数组分簇过程中，若当前 DM 上已放置的数组集合 AS 越大，即可放置剩余空间 LS_L^{AS} 越小，则将数组 A 放置于当前 DM 的开销越大。同理，若数组集合 AS 对当前 DM 的访问次数越多，则 Nas_L^{AS} 越小，因此数组 A 的放置开销也同样增大。通过这样的设置，可以使数组的分簇相对均衡，对各 DM 的访问相对平衡，避免了对某一 DM 过于频繁的访问造成的访存开销增大。

2.2 关键循环存储需求分析

利用多面体分析工具获得嵌套循环内每一数组元素的数据依赖关系后，本文提出了存储感知的循环存储需求计算算法。如图 4 算法 1 所示，该算法主要包含 5 步：(1) 从循环特征信息提取循环内所有数据流图，准备后续数组存储分析所需信息；(2) 根据数组的访存次数计算其优先级，并按优先级降序排列，以保证存储需求大的循环数组将被优先放置；(3) 计算将数组分配至所有 DM 簇上可能的开销，对循环内的所有数组进行分簇，然后选择开销最小的 DM 分簇放置当前数组；(4) 对簇中每一数组在 dP 粒度上生成数据依赖图；(5) 遍历簇中各 dP 依赖图，保证图中与每一个 dP 所相关的 dP 全部产生之后，该 dP 才能被产生。计算分簇中各数组的存储需求，统计所有已被标记至 DM 的数组存储量之和，进而求出循环的最小存储需求量。图 4 算法 2 遍历数组数据域并进行划分，随后利用算法 3 基于当前已分簇数组在 dP 粒度上的数据依赖图计算存储需求，最终得到的 *MindataSize* 即当

前分簇数组的最小存储需求量。该循环的最小存储需求量是该循环中数据的最小存储需求量与代码的存储需求量之和,即

$$MinSize_L = cs + \sum_{i=1}^{s} MinDataSize(AS_i) \quad (2)$$

其中,cs 为程序目标代码的大小,AS_i 为第 i 个数据存储器 DM,s 为循环使用的 DM 数目。循环的最大存储需求量为各个循环的最大存储需求量之和,即

$$MaxSize = cs + \sum_{i=1}^{l} \sum_{j=1}^{k} (A_j \times \mathrm{num}(v_j)) \quad (3)$$

其中,A_j 表示第 j 个数组变量所需的存储空间,$\mathrm{num}(v_j)$ 表示该循环内所有变量中第 j 个变量的数量;l 表示循环的个数;k 表示循环中的数组数。本文所采用的方法与文献[12]的不同之处在于:本文通过引入数组的存储开销判断对数组预先进行分簇,同时基于多面体理论对簇中数组的数据域进行划分。对数组进行分簇可以有效降低所生成 dP 依赖图的复杂度,计算出的存储需求更符合硬件资源约束,而多面体的引入则可以有效提高分析速度并保证分析过程的准确性。

ALGORITHM 1: ARRAY_CLUSTER_MINSIZE
Input: DFGs // a list of DFGs from the current loop L
ASet //a list of LM on which arrays can be mapped
Output: dPCost //a set of arrays mem cost on dP granularity

```
// array analysis
AL =initArrayList(); // a list of arrayInfo from DFGs
AL =arraySizeAnalysis(); //get size of each array
for(each DFGs) {     //get access number of each array
    * let dfg be an node of DFGs;
    AL = arrayAccessAnalysis(dfg);
 }
//array cluster generation
for(each arrayInfo in AL) {
    * let A be an element of AL
    calcPriority(A); //compute PrA of each array in AL
 }
sortArrayList(AL); // sort in descending order by priority
while(AL≠Ø) {    //get costa(AS,A)
  *let A be an array of AL
   for(#ASet)){     // a set of candidate array cluster
  *let AS be a ASet number
    ASetList+= calcCosta(AS,A);
    }
 }
 if (ASetList≠Ø) {
  asl = MinCostASet(ASetList);//select the object cluster
  assign(A, asl); //assign A to cluster asl
  }
//dPList generation
  if(!assign(A,asl)) {
  PList = getPList(A); // a list of an array's data domain
  *let dPL be an element of dPList
  dPList=par_all_data domain(PList);
  for(!dPL){
  dPG = DPAnalysis(dPL));
  }
  for(each node of dPG){
   * let d be an element of dPG;
   dPCost += CalcCostArray(d);
   }
}
```

ALGORITHM 2: Partition of an array's data domain
Input: PList // a list of an array's data domain
Output: dPList //a list of an array's data domain after partition

```
par_all_data_domain(PList){
//iterative partition of data domain
 for every i∈[0, n-1]
   for every j∈(i,m-1] .//m is the current length of PList
       PList+=par_two_data_domain(PList[i],PListt[j]);
 for every element in Plist from n to 0
   dPList+= Plist[i];
}

par_two_data_domain (p1, p2){
 if (p1∩ p2)
  the partiton is{p1, p2-p1};
  elseif(p2⊂ p1)
      the partiton is{p2, p1-p2}
       elseif(p1∩p2=?)
        the partiton is{p1, p2}
 else
   p=p1∩p2;
   the partiton is{p, p1-p, p2-p};
}
```

ALGORITHM 3: Array MinDataSize generation
Input: dPG // a dependence graph in dP granularity
Output: dPCost //a set of arrays mem cost on dP granularity

```
CalcCostArray(dPG){
  for(each unread node d of dPG&&done(predecessor of d)){
     *compute the cost function:
      peak=max(0,Min DataSize+MemInc-tmpMaxSize);
      cost=a*peak+ß*(MemInc-MemDec);
    //α and ß are design parameters for balancing memory cost
     *chose the node d1 with the min_cost;
      done(d1) =1;
    update(MinDataSize, temp_Max_Size);
}
if (all the dP in dPG computed)
    return MinDataSize;
}
```

图 4　关键循环的存储需求分析算法

Fig. 4　Storage requirement analysis algorithm for kernel loop

3　循环流水优化映射

3.1　存储感知的映射目标函数

我们构建了 RCP_CGRA 模型来描述映射关键循环的硬件信息,包括 mPE 和 cPE 资源数量、数据存储 DM 的大小与数量等。令 T_L 为循环消耗的总的时钟周期,则 $T_L = T_C + T_D$,其中 T_C 为循环计算时间,可以通过实验模拟得到,T_D 为循环访存时间,包括数据的载入与写回的存储时间,根据前述存储分析方法,可计算得出 T_D。

$$T_D=\begin{cases} MaxSize\times(\frac{1}{imbw}+\frac{1}{ombw})\times clock_{rate}, & \text{if ①}\\ MaxSize\times(\frac{1}{imbw}+\frac{1}{ombw})\times clock_{rate}+Tm\times R(\frac{MinSize}{dms}), & \text{if ②}\\ +\infty, & \text{if ③}\end{cases} \tag{4}$$

其中 $MemSize = dms \times dmn$，即所有局部数据存储器的大小之和，$MaxSize$ 和 $MinSize$ 分别为循环的最大和最小存储需求量，Tm 为预设的额外访存开销，$clock_{rate}$ 为硬件时钟周期。定义存储/计算时间比为 $DCP = T_D/T_C$，若 $DCP \geqslant 1$，则此循环为存储受限循环。式中其中三种情况分别为

① $MemSize \geqslant MaxSize$；

② $MinSize \leqslant MemSize \leqslant MaxSize$；

③ $MemSize \leqslant MinSize$。

根据前述分析可知映射某一循环 L 的存储感知的目标函数为

$$ObjFunction(L)=\begin{cases} minUoR, & \text{if } DCP\geqslant 1\\ maxThO, & \text{if } DCP<1\end{cases} \tag{5}$$

其中 UoR 为资源占用率，ThO 为循环流水化的吞吐率。若存储资源充足，优先考虑最大化 ThO，而当 $DCP \geqslant 1$ 时则认为存储资源不足，则优先考虑最小化 UoR。

在 RCP_CGRA 模型里，CPE 数目为 $m\times(n-1)$，MPE 数目为 m，DM 数目为 dmn。若循环 L 映射至此 PE 阵列上，占用了 i 个 cPE，j 个 mPE 以及 s 个 DM，包含 Num_L 个计算及存储操作，则 UoR，ThO 分别为

$$\begin{cases} UoR_L=\dfrac{i+j+s}{m\times n+dmn}\\ ThO_L=\dfrac{Num_L}{T_L}\end{cases} \tag{6}$$

因此在第 3 节所描述的循环映射此时可以细化为面向 CGRAs 的存储感知循环映射（Memory-Aware Loop Mapping，MALP）问题，即对于给定的某一数据密集型应用中的关键循环 L 在 RCP_CGRA 硬件结构上，寻找一个性能较优且满足其计算功能和存储约束的映射绑定，满足最小化的资源占用率 UoR_L 和最大化的数据吞吐量 ThO_L。在存储受限情况下优先满足目标 min（UoR），在存储资源充足时，优先满足目标 max（ThO）。

3.2 数据映射分析

考虑循环映射时，若循环迭代间对同一数组的访问是对同一 DM 上连续地址的存储操作，则可通过数据重用减少此存储操作数目。以图 2 代码为例，可知嵌套循环内层 j 迭代间数组 B[i,j] 和 B[i,j-1] 存在数据重用的可能。通过遍历已有 DFG，发现所有在同一数组上存在读后读（RAR）及写后读（RAW）相关的访存操作节点对，如图 2 的 DFG 中节点对（0，2）与（1，7）。消除这些节点对中的后续节点（2 和 7），并在这些节点对的前导节点（0 和 1）与后续节点的原有后继间添加重用边，即可生成数据重用图（Data Reuse Graph）。图 5 左半部分显示了对图 2 中 DFG 进行数据重用转换后的 DRG，右半部分则显示了经过转换后的循环流水时空图，其中虚线边代表了重用边。重用边的权值为其端点的重用距离（Reuse Distance，RD）之和。重用距离是指运算单元间重用数据的开销，等于从前一单元产生相关数据到后一单元能够利用相关数据进行计算所需的时钟周期数。对于具有直接操作数互联的运算单元，其 RD 视为 0；利用寄存器文件进行操作数传递的指令流水线，其功能单元之间的 RD 视为 1，因为需要增加一个写入寄存器的时钟周期。由此可知 RD 越大，数据重用的时间耗费越长。在实际应用中我们可以综合考虑目标体系结构的可用寄存器数、访存操作延迟及频率以及目标启动间距Ⅱ来设定重用距离 RD 的上限，从而保证数据重用有效。

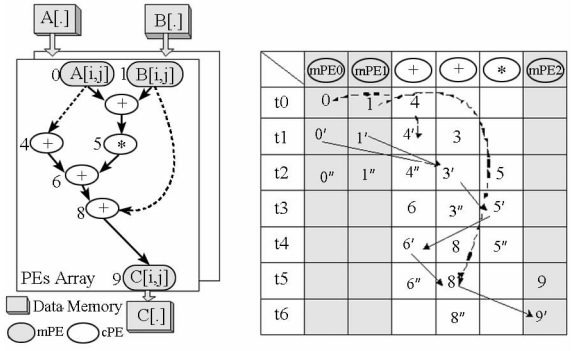


图 5 数据重用转换

Fig. 5 Data reuse conversion

4 实验与分析

本节实验所采用粗粒度可重构 SoC 平台包括一个嵌入式通用 RISC 处理器核 Estar Ⅲ以及一个可重构阵列 LEAP。其中，EstarⅢ用作主处理器核，LEAP 作为协处理器进行关键循环映射加速。LEAP 结构符合 RCP_CGRA 模型描述，采用循环自流水技术将循环控制表达式映射至 mPE，并将循环体映射至 cPE 阵列。LEAP 通过接口控制器与 EstarIII 互联，完成配置信息和数据的加载、结果的返回以及反馈循环运行状态。实验采用 LooPo 工具对循环内部数

组间的数据依赖关系进行分析,利用 PolyLib 对循环粒度的存储需求进行分析。采用的应用实例包括一些典型的数据密集型算法核心,在对其进行了存储需求分析后,面向 LEAP 采用文中方法进行映射,并测试其运行时间和映射性能,包括:1)基本算法核心:离散余弦变换 DCT ,矩阵向量转置 MVT 和矩阵 LU 分解。2)典型算法:我们选择了三个典型算法用于对存储感知循环流水映射方法(MALP)的性能进行评估,包括矩阵乘(MM)、运动检测(MD)和中值滤波(MF)。在对已有方法(SPKM、LKPM)和本文方法(MALP)进行对比时,比较的指标包括各算法所占用的计算/存储资源数目、存储需求、执行时间、吞吐量和资源占用率。

在典型算法的循环存储需求分析中,以 DCT、MVT 和 LU 中关键循环为例。这三个程序中计算复杂度最高的部分均为嵌套循环及其数组操作。采用第 3 节提出的数组分簇分析方法(记为 ACP),分别计算程序的最小存储需求量。表 1 给出了三种基本算法核心中的关键循环特征分析结果以及最小存储需求量。其中,数组操作一栏的括号内表示该数组的分簇优先级,加速提升是本文方法相对于已有 Balasa 方法分析时间的提升量。本文方法将数组分簇后,将数组数据域划分为多个耦合多面体,以分簇后的多面体为分析单位,相对于 Balasa 基于基本集的分析方法,速度有了较大提高。

表 1　典型循环特性分析结果

Tab. 1　Features analysis results of typical loops

目标程序	DCT(M = 2000)		MVT(N = 10000)		LU(N = 2000)
循环标号	main. loop1	main. loop2	main. loop1	main. loop2	main. loop1
循环类型	for	for	for	for	for
嵌套层数	3	3	2	2	3
数组操作	temp2d(0.43) block(1.02) cos1(1.02)	sum(0.36) cos1(0.90) temp2d(0.90) block(0.36)	x1(1.0) a(0.80) y_1(0.80)	x1(1.0) a(0.80) y_1(0.80)	a(1)
索引关系	temp2d:(x,y) = (i,j) block:(x,y) = (i,k) cos1:(x,y) = (j,k)	sum:(x,y) = (i,j) cos1:(x,y) = (j,k) temp2d:(x,y) = (k,j) block:(x,y) = (i,j)	x1:(x) = (i) a:(x,y) = (i,j) y_1:(x) = (j)	x2:(x) = (i) a:(x,y) = (i,j) y_2:(x) = (j)	a:(x,y) = (k,j) a:(x,y) = (k,k) a:(x,y) = (i,j) a:(x,y) = (i,k)
存储需求	536,328 bit		43,852bit		278,938bit
分析时间	81,723ms		3,147ms		12,433ms
加速提升	32%		13.3%		27.1%

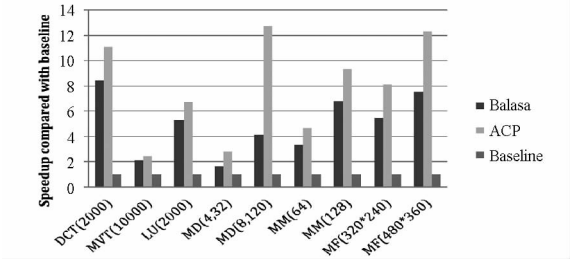


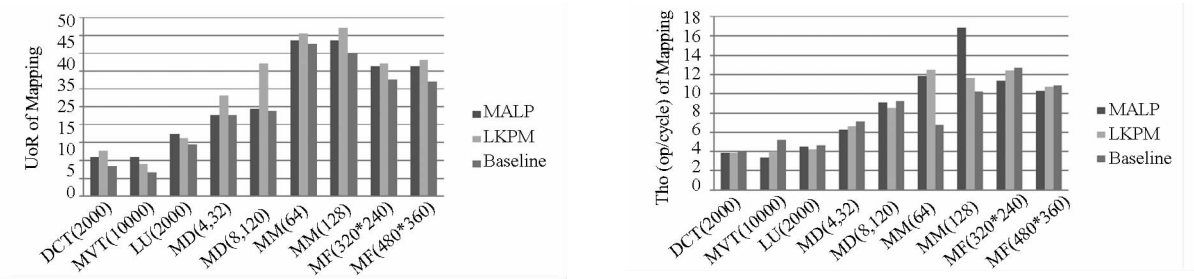
图 6　各算法存储分析时间的加速对比

Fig. 6　Speedup comparison between storage analysis time on algorithms

图 6 反映了各算法分别采用 Balasa 方法和本文方法分析存储需求后对应于基本标量方法(Baseline)的加速比。其中 MD(4,32)表示算法参数为 $m = n = 4$, $M = N = 32$, MM(64)表示矩阵规模为 64×64。由图可知在算法数组规模增大时标量方法和 Balasa 方法的分析速度由于问题规模扩大而降低,而在本文方法中,尽管多面体内部格点随数组规模而相应增多,但并不影响分析的准确度,分析加速比提升反而更大。

根据上述数据密集型算法的存储分析结果,随后采用本文提出的存储感知循环流水映射(MALP)方法对这些算法进行进一步实验分析。表 2 给出了在应用 MALP 方法后各算法在 RCP_CGRA 上的映射性能结果,包括占用 cPE/mPE/DM 数目、吞吐量 ThO、计算活跃度 AP、存储/访问时间比 DCR 以及执行节拍数 Cycle,我们在每项的括号内列出了采用 LKPM 方法优化后的结果作为参照。图 7 分别给出了 MALP 方法对比已有 LKPM 方法进行映射的资源利用率 UoR 和吞吐量 ThO,其中 Baseline 为对算法进行手工优化后的映射情况。可以发现我们的方法与手工映射方法在性能方面差距较小。同时由于对算法核心进行映射时充分考虑了存储影响,对矩阵乘、运动检测等具有较多数据依赖的算法核心进行了数据重用,节省了存储

资源,较之 LKPM 方法显著降低了资源占用率,且提高了数据吞吐量。



(a) UoR (b) ThO

图 7 典型算法映射性能比较

Fig. 7 Performance comparison between mapping results of algorithms

表 2 典型算法在 RCP_CGRA 映射性能结果

Tab. 2 Results of typical algorithms mapping on RCP_CGRA

算法	cPE/mPE/DM	ThO(OP/cycle)	AP(%)	DCR	Cycle
DCT(2000)	4(6)/4(4)/4	3.9%(3.9%)	40.3%(39.3%)	1.491	76207(78403)
MVT(10000)	6(4)/3(3)/3	3.1%(4%)	41.5%(43.6%)	1.409	17329(17482)
LU(2000)	10(10)/3(4)/4	4.5%(4.2%)	42.0%(40.3%)	1.381	48239(53759)
MD(4,32)	7(7)/10(16)/8	6.3%(6.6%)	40.5%(40.1%)	1.469	190862(210533)
MD(8,120)	7(7)/12(20)/8	9.2%(8.5%)	43.2%(42.1%)	1.315	391223(433748)
MM(64)	30(30)/8(10)/10	16.7%(12.5%)	41.2%(40.7%)	1.427	65293(79141)
MM(128)	30(30)/8(12)/10	11.8%(11.6%)	44.5%(42.6%)	1.247	304130(318901)
MF(320×240)	30(30)/6(7)/4	11.4%(12.4%)	39.1%(41.3%)	1.558	213932(220010)
MF(480×360)	30(30)/6(8)4	10.3%(10.7%)	39.3%(40.2%)	1.621	428467(479792)

5 相关工作

CGRA 的性能主要取决于其体系结构模板和数据密集型应用的映射策略。Yoon[3] 采用 SPKM 方法支持 CGRA 上应用的自动映射,并获得了较高资源利用率,但其并未考虑通过优化存储约束提升数据密集型应用的性能的方法。Cardoso[5] 讨论了循环自流水在可重构硬件上的实现,提高了循环吞吐率,但尚未考虑循环中的数组依赖关系。Wang[4] 针对专用的支持循环自流水 CGRA 结构采用 LKPM 方法实现循环的流水化自动执行,但同样未能考虑循环内的依赖关系以及存储资源对映射目标函数的影响。在体系结构模板方面,KiM[6] 引入粗粒度可重构体系结构模板并对可重构计算资源进行分类, 支持资源流水以进行专用领域的优化,但未考虑数据密集型应用。针对循环的存储需求方法中,典型的基于数组分析的方法有 Balasa[7],通过引入多面体和线性有界格对循环存储需求进行精确建模,获得其最小存储需求量且准确度较高,但此方法的分析速度随着迭代域的增大而下降,不适应面向多媒体数据密集型应用程序的分析。本文引入数组分簇及数据域划分方法,分析循环数组存储需求量,根据 CGRA 中的存储资源约束,采用存储感知的循环流水化映射方法,在满足硬件资源约束的前提下实现循环的数据映射和计算映射,有效提高了资源利用率和数据吞吐量。

6 结论与展望

粗粒度可重构体系结构 CGRAs 提供了高效和灵活的配置来加速复杂的数据密集型应用,但如何有效映射这些应用中的计算核心部分,尤其是关键循环,仍是一个挑战性任务。我们针对 CGRAs 的各种典型资源和计算特征,定义了 RCP_CGRA 模型并定义了循环到 CGRAs 映射的关键问题。其中针对在 CGRAs 映射关键循环时由于频繁的存储访问带来的延迟问题,我们将数组分簇,利用多面体理论对数组的数据域进行划分,随后获取循环的存储需求量。随后,结合存储分析结果给出了一种存储感知的循环流水映射方法 MALP。对几种典型应用算法的映射实验表明,本文提出的映射方法与已有方法相比,其资源占用率较小,吞吐量更高,接近于手工映射方法的性能。但 MALP 方法考虑数据映射时只采用了数据重用和冲突消除技术,未结合 CGRAs 的存储结构层次对循环进行更多优化,需要在未来工作中进一步考虑。

参考文献(References)

[1] Suresh D, Najjar W, Vahid F, et al. Profiling tools for hardware/software partitioning of embedded applications [C]// Proceedings of the ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems, San Diego, USA. 2003: 189 – 198.

[2] Cardoso P, Diniz C, Weinhardt M. Compiling for reconfigurable computing: a survey [J]. ACM Computing Surveys, 2010, 42(4): 1 – 65.

[3] Yoon J, Shrivastava A, Park S, et al. SPKM: A novel graph drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures [C]//Proceedings of the Asia South Pacific Design Automation Conference, Seoul, Korea: 2008:776 – 782.

[4] 王大伟,窦勇,李思昆. 核心循环到粗粒度可重构体系结构的流水化映射[J]. 计算机学报, 2009, 32(6): 1089 – 1099.
WANG Dawei, DOU Yong, LI Sikun. Loop kernel pipelining mapping onto coarse-grained reconfigurable architecture for data-intensive applications [J]. Journal of Computers, 2009, 32(6):1089 – 1099. (in Chinese)

[5] Joao M, Cardoso P. Dynamic Loop pipelining in data-driven architectures [C]//Proceedings of the 2nd Conference on Computing Frontiers, New York, NY, USA,2005:106 – 115.

[6] Kim Y, Mahapatra R. Hierarchical reconfigurable computing arrays for efficient CGRA-based embedded systems [C]// Proceedings of the 46th Annual Design Automation Conference, San Francisco, USA,2009:826 – 831.

[7] Balasa F, Zhu H, Luican I. Computation of storage requirements for multi-dimensional signal processing applications [J]. IEEE Transactions on Very Large Scale Integration Systems, 2007, 15(4): 447 – 460.

[8] LooPo - Loop parallelization in the polytope model [EB/OL]. University of Passau, 2012. [2012 – 5 – 2] http://www.fmi.uni-passau.de/loopo.

[9] ZHAO P, LI S K, WANG D W, et al. A new application feature analysis approach for system-on-chip hardware/software partitioning[C]//Proceedings of the International Congress on Image and Signal Processing, Sanya, China. 2008: 630 – 634.

[10] 窦勇,邬贵明,徐进辉,等. 支持循环自流水的粗粒度可重构阵列体系结构[J]. 中国科学, 2008. 38(4): 579 – 591
DOU Yong, WU Guiming, XU Jinhui, et al. A coarse-grained reconfigurable computing architecture with loop self-pipelining [J]. Science in China. 2008, 38(4): 579 – 591. (in Chinese)

[11] Bastoul C. Code generation in the polyhedral model is easier than you think [C]//Proceedings of the IEEE International Conference on Parallel Architecture and Compilation Techniques, Washington, DC, USA,2004: 7 – 16.

[12] 赵鹏, 王大伟, 李思昆. 面向 SoC 任务分配的应用程序存储需求量分析方法[J]. 电子学报, 2010. 38(3): 541 – 545.
ZHAO Peng, WANG Dawei, LI Sikun. Research on memory size estimation of application programs for system-on-chip task allocation[J]. Journal of Electronics, 2010, 38(3):541 – 545. (in Chinese)

(上接第 38 页)

5　结论

在互连网络中,全局气泡流控比局部流控有着潜在的优势。局部 BFC 需要接收缓冲区至少有两个报文的空间,而全局 BFC 只要有一个空闲报文空间即可避免死锁。CBS 机制首先提出了一种有意义的全局 BFC 实现方式,但是同时有阻塞的风险。在本文中,我们首先提出了伪报文协议来处理某些路径上没有报文流动的状况,然后结合该协议设计了移动气泡流控机制。通过使气泡不断移动,避免了阻塞,有效实现了全局气泡流控。实验结果表明,我们所提的机制在只有一个报文空间时网络是无阻塞的,而在使用两个报文缓冲空间的情况下,其吞吐率明显高于 BLOC 和 CBS 流控方式,除了 Transpose 传输模式外,MBS2 提高的幅度均大于 20% 。

参考文献(References)

[1] Alverson R, Roweth D, Kaplan L. The Gemini system interconnect [C]//Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects. Washington, DC: IEEE Computer Society, 2010:83 – 87.

[2] Chen D, Eisley N A, Heidelberger P, et al. The IBM blue gene/q interconnection fabric[J]. IEEE Micro, 2012, 32:32 – 43.

[3] Ajima Y, Sumimoto S, Shimizu T. Tofu: a 6d mesh/torus interconnect for exascale computers [J]. Computer, 2009, 42:36 – 40.

[4] Dally W, Towles B. Principles and practices of interconnection networks [M]. San Francisco: Morgan Kaufmann Publish, 2003.

[5] Carrión C, Beivide R, Gregorio J A, et al. A flow control mechanism to avoid message deadlock in k-ary n-cube networks [C]// Proceedings of the Fourth International Conference on High-Performance Computing. Washington, DC: IEEE Computer Society, 1997: 322 – 329.

[6] Puente V, Izu C, Beivide R, et al. The adaptive bubble router [J]. Journal of Parallel and Distributed Computing, 2001, 61: 1180 – 1208.

[7] Adiga N R, Blumrich M A, Chen D, et al. Blue gene/l torus interconnection network [J]. IBM Journal Res. Dev., 2005, 49: 265 – 276.

[8] Chen L, Wang R, Pinkston T M. Critical bubble scheme: an efficient implementation of globally aware network flow control [C]// Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Washington, DC: IEEE Computer Society, 2011:592 – 603.

[9] Glass C J, Ni L M. The turn model for adaptive routing [J]. Journal of ACM, 1994, 41(5):874 – 902.

[10] Navaridas J, Alonso J M, Pascual J A, et al. Simulating and evaluating interconnection networks with insee [J]. Simulation Modelling Practice and Theory, 2011, 19(1):494 – 515.