* + 1. 问题模型

应用可用数据流图（Data Flow Graph，DFG）表示，在DFG中每个节点表示应用中的某一个运算操作，有向边表示两个操作间的数据依赖关系和数据流向。应用到可重构阵列上的映射包括三个部分：DFG中的节点到可重构阵列上的PE的映射；DFG中的有向边到PE间互连的映射；数据到局部存储的映射。图xxx给出了一个应用到可重构阵列上的映射，图中的R表示该PE不进行运算，作为路由使用。



通常将应用映射到可重构阵列上需要考虑体系结构上的几点制约因素：

* 阵列

阵列的规模，即阵列上的资源数量，直接决定应用操作的并行度和吞吐量，较大的应用映射到资源数较少的阵列上时，需要根据资源数量对应用算法进行划分，在不同划分里的运算需要对可重构阵列进行配置切换才能继续执行。另一个关键因素是阵列的拓扑结构，包括阵列的横向执行单元的个数，这决定了应用可以在阵列上映射的数据宽度；阵列的纵向深度，这在应用进行流水展开时决定了可以映射的流水级数。

* 处理单元

每一个都有多个功能单元资源，功能资源包括单操作的功能单元例如移位器和支持多种操作的功能单元如一些功能单元组合操作。在异构的可重构阵列中，一些面积开销大的功能单元如查找表单元只存在阵列中的部分处理单元中，因而不是所有的操作都能映射到任意的处理单元上，部分操作只能由特定的处理单元完成。

* 局部存储器

通常上多个通过总线来共享一个局部存储器，总线的带宽有限，不能在一个周期来完成所有PE的数据获取，因此在映射时要考虑这种总线带宽限制。

* 互连网络

可重构阵列中PE间的互连资源是有限的，不是所有的PE间都可以直接互连；在某些时候还需要将某一个PE作为路由使用。互连网络结构是应用映射时通信路径选择的主要约束因素。

本文提出的可重构密码处理器上应用映射的特点

1. 可重构密码处理器阵列的互连是有导向的，数据从阵列的第一行输入，第一行的输出结果给第二行，数据经过整个阵列从最后一行输出。因此在映射时从阵列的第一行出发，数据流向和阵列的互连导向一致。
2. 可重构密码处理器阵列上的映射属于空间映射，密码算法会被循环展开，在架构上进行流水展开映射。
3. 行间的互连不是硬连线，而是数据选择。可以选择某条路径到达下一行的任何位置，但不可以同时到达所有的位置。也就是从某一个点出发的所有连线是互斥的，不可约同时存在，每一次只能互连到一个位置。
4. PE内部功能更复杂，不只是单个运算到单个PE的映；PE内部有很多组合功能设计，应用中的多个组合的功能会被同时映射到一个PE中。
5. 单个PE的输入不唯一、功能不唯一，也就是说在输入允许的情况下，横向上并行的多个功能可以被映射到同一个PE。
6. 可重构密码处理器阵列被设计成行间异构、列间异构。这样的不一致性结构在架构描述、映射算法上都有特异性。

决定方法的变化

* + 1. 研究现状

将循环映射到时，根据映射目标架构的不同，有两种常用的映射方法：时间映射（temporal mapping）和空间映射（spatial mapping）。

时间映射适用于处理单元只有一个或者一行的可重构处理器的架构。时间映射的方法将整个循环体映射到一个PE或一行PE上，每PE个或每行执行不同迭代次数的循环。PE要顺序执行多个操作，每执行完循环的一次操作就要进行一次重构，动态配置下一级的操作功能。时间映射方法的特点有：

* 由于循环的所有操作都在一个PE或一行内完成，不需要考虑PE间的数据互连；
* 可以直接利用传统的编程编译方法完成循环的映射；

这类CGRA映射问题[ V1-V3]的方案来自于VLIW架构的编译技术，它利用了VLIW中的时间流水模调度算法和VLIW中的存储共享特征。

空间映射的方法适用于可重构阵列架构，它将循环中的每个操作一次性映射到CGRA的PE阵列上，每个PE只绑定循环中的一个操作，只执行一种固定的操作，整个循环执行的过程中只用配置一次，不需要在经过重新的配置执行其他操作。空间映射的方法有以下特点：

* 能够充分利用CGRA计算资源和并行运算的能力，并行执行循环体的多次循环；
* PE只需要完成一个固定的操作，重构的开销小，整个循环的过程不需要进行重构；
* 操作节点的布局要充分考虑各个操作间的数据依赖关系和CGRA的互连资源，布局复杂，并且要使用较多的互连资源；

这类CGRA映射问题的研究更为广泛，最开始的方案灵感来自FPGA[v4-v5]综合里的布局布线技术，但是CGRA本身和FPGA有很大不同的，具体来说，CGRA中的互连线大部分是固定的，而FPGA中是可配置的；因此基于FPGA综合里的布局布线技术在固定的互连在寻找路由变得很困难。后来的一些映射方法主要来自于图论领域，[V6 ]利用子图同构算法获取DFG图到架构图之间的映射候选集；SPKM [V7] 引入分裂和外扩的方式 [V8] ，它将应用看成一个集合，每一次向外扩展一个点，余下的点作为新的扩展集合，一直到集合中的点全部外扩完毕。类似的图论方案还有如[V9]中的子图同胚[V10(134)]以及EPIMap[V10(56)]中的图满射技术。

本文中的可重构密码处理器是架构中包含PE阵列，因映射时将密码算法循环展开以流水的形式映射到阵列上，因此本文中的应用映射问题为空间映射。根据上小节总结的架构特征，本文选用适用性更好的子图同构算法作为映射的依据，子图同构算法能够在异构的不规整架构中找出所有的可能映射方案，然后再通过一定的映射规则找出资源使用最少的方案作为最终映射方案。

* + 1. 子图同构基本概念
* 基本定义

**定义5.1（图）：**一个图是一个四元组，其中，

1. 称为G的顶点集，其元素称为顶点或结点；
2. 称为G的边集，其元素称为边；
3. 为图G的顶点标记函数，说明顶点与其标记的对应关系；
4. 为图G的边标记函数，说明边与其标记的对应关系。

**定义5.2（子图）：**一个图 为图的一个子图，记为，如果有：

1. ；
2. )；
3. ；
4. 。

此时，我们也称图为图的一个超图。

**定义5.3（图同构）：**图和图是同构的，记为，如果存在一个双射函数，使得：

* 1. ；
  2. 。

这样的函数也称为图与的图同构。例如，图xxx中A和B两个图同构。



图 xxx 图同构的示例

**定义5.4（子图同构）：**给定图与的一个图同构以及另一个图，如果，则为图与的一个子图同构。图xxx是一个子图同构的示例，S是G的一个子图，它包含了G中顶点集的一个子集，而且S中顶点的连接方式与G中相同，所以称S是G的一个同构的子图。



图 xxx 子图同构的示例

* 复杂度分析

在理想情况下，问题都可以在多项式时间（也称为P时间）内解决。这意味着对于解决一个含有n项的问题，存在c，z，g几个常量，算法T所需时间受以下公式下公式[Vxx]的限制：

其中常量z是我们真正感兴趣的，它通常被叫做“增长因子”。它表示当n增加时算法预期增多的额外时间的数量。例如当z =1，n加倍时，算法所需的时间也加倍。

但是子图同构被证明是一个NP完全问题，即在最坏情况下，判定两个图子图同构所需时间与图中所包含的节点数量成指数增长关系。也就是说，最坏情况下解决子图同构问题的时间复杂度为，其中 n 为规模较大的图中的节点个数。

虽然子图同构问题具有先天的复杂性，但是由于可重构密码处理器映射本身的一些特性可以使问题的复杂度降低。

1. 算法图和架构图都是有向图，而且有明确的起点和终点，这样架构图中的算法图候选集非常有限。
2. 架构图中没一个结点（PE）的输入输出很有限，而且结点只有相邻才有互连，因此图的边集很小，这也简化了算法的复杂度。
3. 在本架构中一共只有5类PE，而且边没有属性，降低了图标记的难度。
   * 1. VF2子图同构算法

VF2算法是2004年由Cordella在文献[xxx]中提出的，随后被广泛采用来解决子图同构问题。

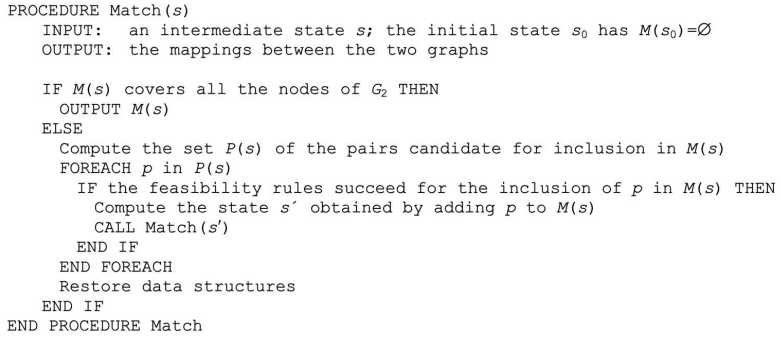
Cordella, Luigi P., et al. "A (sub) graph isomorphism algorithm for matching large graphs." Pattern Analysis and Machine Intelligence, IEEE Transactions on 26.10 (2004): 1367-1372.

核心思想其实很简单，就是搜索，加剪枝。重点就在于如何剪枝。

这里使用state s来存储搜索过程中的部分匹配（partial mapping），以及其他算法需要的数据。

下面定义几个符号，M(s)代表中间状态，s代表的部分匹配。M1(s)和M2(s)表示当前state s的部分匹配中，G1和G2中的点。

**算法伪代码如下：**



起初，状态是s0，M(s0)是空集，即还没有任何匹配。之后递归的进行搜索。

如果当前状态s代表的部分匹配M(s)包含了G2（query graph）中的所有节点，则已经找到了G2在G1中同构的子图，搜索结束。

否则，在当前的局部匹配基础上，再匹配一个点。首先，找出所以可能进行匹配点对集合P(s)。然后，对于每一个匹配对p，检查加入匹配p是否可行。即加入p后，两个图还是否同构。以及加入p之后，是否还有就扩展的可能性（即实行一些剪枝策略）。

如果加入匹配p可行，则将p加入s，递归调用Match(s)，继续搜索。

如果刚才若干次调用Match(s)后都没有找到同构的子图，则说明当前从状态不可能扩展出可行的子图同构匹配。所以，将生成改状态时加入的两点匹配p从s中删除，回溯到上一个状态。

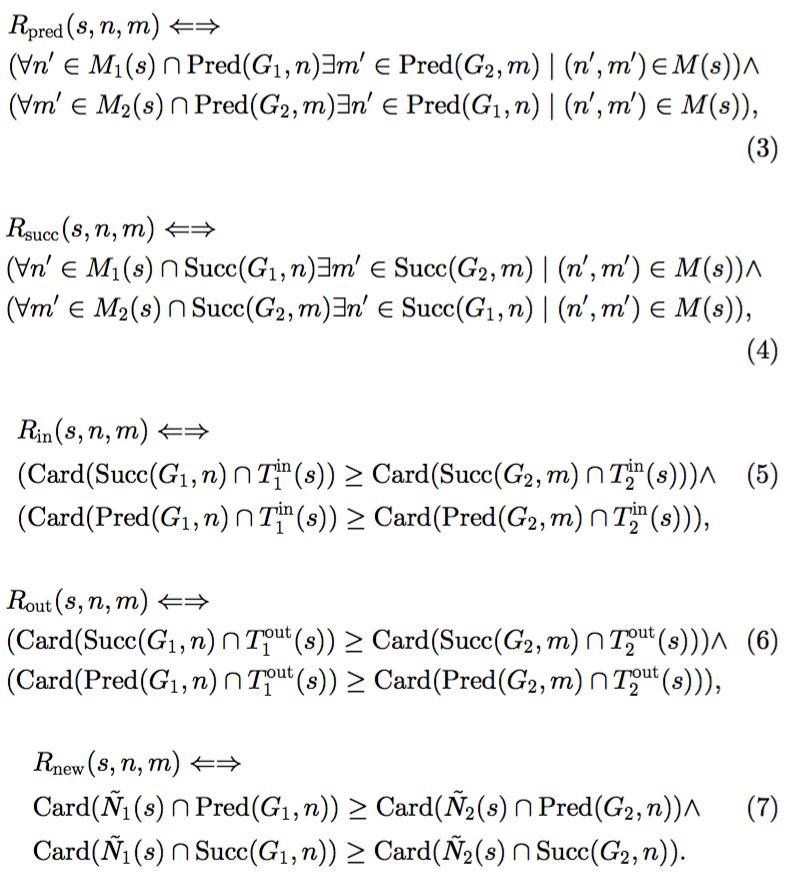
对于之前算法框架中，新加入的匹配p，我们要检验其加入的可行性，从而对搜索空间进行剪枝，来提高算法的效率。

**检查新匹配的可行性**

这里先定义几个符号：N1和N2表示图1和图2中的点集。n和m分别表示图1和图2中的点。Pred(G,n)表示点n在图G中的前驱，Succ(G,n)表示点n在图G中的后继。Tin1(s)和Tin2(s)表示状态s在图1和图2中，指向当前已经匹配的点集的所有边的source点集合（边的起点）。Tout1(s)和Tout2(s)表示状态s在图1和图2中，从当前已经匹配的点集出发的所有边的target点结合（边的终点）。

T1(s)=Tin1(s)∪Tout1(s)，即当前状态s在图1中已经匹配的点集的所有一步邻居。N˜=N1−M1(s)−T1(s)，即图1中，除了s中已经匹配的点，和这些点的一步邻居以外的点。

具体的判定规则如下：



前两条保证加入新的匹配对p后，两个子图仍然是同构的。设新加入的匹配对是(n,m)，则对于n在图1中的所有前驱（或后继），必须能在图2中m的前驱（或后继）里有相应的点与之对应。同样，对于m在图2中的所有前驱（或后继），也必须能在图1中n的前驱（或后继）里有相应的点与之对应。

之后的三条都是剪枝策略。其中Card表示求集合中元素的个数。

三四条表示，n在Tin1（或Tout1）中的前驱（或后继）的数目，必须大于等于m在Tin2（或Tout2）中的前驱（或后继）的数目。如果不满足，则说明对于query graph中新匹配的点m，其邻居个数是大于target graph中n的邻居个数的，所以说最终必然无法完全匹配query graph中所有的点。

第五条跟三四条思想类似，只不过考虑的两步邻居。具体来说，三四部中的考虑的邻居是Tin1和Tout1中的邻居。这些点即跟n相邻，又跟当前匹配中的其他点相邻。而第五条考虑的邻居，是只跟n相邻，跟当前匹配中其他店不相邻的邻居。这样细粒度的考虑的好处是，可以更细粒度的剪枝，从而提高剪枝效率。

**生成候选匹配对**

如果Tout1和Tout2都不为空，则取这两个集合中的所有点两两组合，生成候选匹配对集合；

否则如果上面两个集合都为空，若Tin1和Tin2都不为空，则取这两个集合中的所有点两两组合，生成候选匹配对集合；

最坏的情况，上面四个集合都为空（对于非连通图可能有这种情况）。则只能找两个图中，所有没有匹配的点两两组合，生成候选匹配对集合了。

之所以这么细粒度的分情况讨论，是想尽量减少单次生成的候选匹配对的数量。否则如果每次都按上面第三种方式生成，则每次递归都会生成很多之前生成过的匹配对，造成重复计算。

* + 1. 基于VF2算法的映射方案

1. 构建算法和架构图：

图的输入输出接口，从输入文本中构建图

结点属性：架构PE的属性（图点的属性），属性代表PE能提供的功能，在进行映射时算法中的点向架构图的映射时必须首先进行属性匹配

1. 打包：根据架构特点对算法进行功能组合
2. 点到点映射（改进的VF2算法）

要考虑属性匹配（在算法的五个规则的基础上加入属性匹配规则）

1. 映射过滤：从多个映射中选取最优匹配（最少行列映射）

需要向算法和架构图中的节点加入位置信息，根据映射结果的位置匹配进行选择