* 简要介绍课题的问题背景、目标、解决思路和方法。

问题背景：

可重构处理器发展趋势：

迭代架构-处理器阵列

更高的主频更高的吞吐率

可重构处理器阵列通过轮函数流水展开，实现单周期完成一组数据加密，实现高吞吐率。但同时也是整个处理器的面积变得很大，其中可重构处理器阵列部分占据主要部分。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 架构 | 架构特点 | 工艺nm | 芯片面积m2 | 处理单元面积m2 |
| ProDFA | 迭代 | 130 | 0.72 |  |
| cryptoraptor | 处理单元阵列 | 45 | 6.32 | 4.54 |
| RCPA | 处理单元阵列 | 80 | 14.9 |  |
| 清华项目 | 处理单元阵列 | 40 |  |  |

导致可重构处理器阵列面积过大的原因分析：

每一个PE包含了所有算法的所有功能，这些功能单元的利用率很低，存在很大的冗余，下图列出了阵列架构中基本运算单元的利用率。从表中可以看出这些单元的利用率很低，存在很大的优化空间。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TH | AU | SH | PER | LOU | 4\*LUT256\*8 | GFM |
| AES | 0% | 0% | 0% | 8% | 25% |  |
| DES | 0% | 0% | 25% | 2% | 6% |  |
| SM4 | 0% | 25% | 0% | 4% | 3% |  |
| TWOFISH | 17% | 4% | 0% | 4% | 4% |  |
| RC5 | 13% | 13% | 0% | 2% | 0% |  |
| BLOWFISH | 19% | 0% | 0% | 3% | 3% |  |
|  |  |  |  |  |  |  |
| Cyptoraptor |  |  |  |  |  |  |
| AES | 0% | 0% | 0% | 50% | 7% |  |
| DES | 0% | 0% | 17% | 17% | 2% |  |
| SM4 | 0% | 25% | 0% | 25% | 1% |  |
| TWOFISH | 20% | 5% | 0% | 30% | 1% |  |
| RC5 | 13% | 13% | 0% | 13% | 0% |  |
| BLOWFISH | 25% | 0% | 0% | 25% | 1% |  |
|  |  |  |  |  |  |  |
| RCPA |  |  |  |  |  |  |
| AES | 0% | 0% | 0% | 25% | 25% | 50% |
| DES | 0% | 0% | 17% | 8% | 8% | 0% |
| SM4 | 0% | 25% | 0% | 13% | 3% | 0% |
| TWOFISH | 20% | 5% | 0% | 15% | 5% | 10% |
| RC5 | 13% | 13% | 0% | 6% | 0% | 0% |
| BLOWFISH | 25% | 0% | 0% | 13% | 4% | 0% |

只关注单个PE设计，没有考虑功能单元之间的拓扑关系

在当前的功能单元设计中，考虑的是算法中使用了哪些功能单元以及这些功能单元的简单组合，然后将这些功能单元堆砌在同一个PE中，形成一个同构的阵列架构。这样做的好处是在算法需要某个功能时，对应位置的PE就能提供所需的功能，其它位置的PE中这些功能单元就会被闲置。比如对于AES算法，假设映射时算法的一个轮函数被映射到3行中，如果一行中有4个PE，那么就有3x4=12个PE；AES算法中查表的操作会被映射到4个PE中，那么其它的8个PE中的查表单元就会被闲置，对于其他的功能单元也会有类似的闲置发生，而且闲置的单元数目会远远多于被使用的数目。在有些时候，为了兼顾多种算法，闲置是不可避免的，但是通过研究足够多的算法的功能单元拓扑结构，从中找出这些算法拓扑之间的共性，可以在某种程度上减少闲置的发生。

目标：

针对当前项目中和学术论文中的架构，新的PE设计方案在整体的性能面积比上有30%以上的提升。

解决思路：

本课题提出的PE设计方案，在优化PE内部功能单元设计的同时对各个功能单元在阵列中出现的拓扑关系进行了布局，减少在算法映射时冗余的发生，提高了功能单元的利用率，从而有效地提升了整个架构的面积效率。

方法：

1. 算法aoe加模、特征提取：建立了一个图模型来描述所有的分组加密算法，提取算法特征。主要涉及三列特征参数特征、组合特征、拓扑特征
2. PE设计方案：根据提取的算法特征，进行功能单元设计，组合设计，拓扑设计。
3. 算法aov建模、架构建模：建立了一个图模型来描述阵列架构和算法，这个算法模aov型和阵列架构的图模型会作为映射验证算法的输入。
4. 映射验证和进一步优化：将建模的算法图模型集合映射到阵列架构图模型上，分析映射结果，对架构中可能存在的闲置单元进行进一步优化。
5. 设计实现：RTL实现最终方案，仿真、综合。
6. 结果对比：与项目中、学术论文中的架构进行对比。



1. 算法aoe加模、特征提取

* 参数特征

参数特征主要指算法的一些基本特征和基本操作的功能特征，比如算法的分组大小、轮数、秘钥长度，移位的左右移位、循环移位等操作模式、S盒的不同输入输出位宽等。通过多算法集中算法进行统计，可以总结出这些特征。

* 组合特征

组合特征指算法中频繁出现的算子组合，很多算子会在算法中组合出现，比如S盒操作后面一般会接着进行抑或操作。对于这种组合操作，可以在上一行PE中先进行S盒操作，下一行PE中执行抑或操作。另一种更好的方式是在一个PE中将S盒和抑或串联组合设计，这样就可以在一个PE中完成两个操作，有效减少资源消耗。

对于算法中出现的每一个算子，发掘每一类算子的前驱、后继算子的种类和出现频率对于架构设计是十分有必要的，它为PE的组合功能设计提供依据。当架构的PE中有大量合理的组合功能存在，在算法实现时，很少的PE阵列规模就可以实现原来需要很大规模才能实现的算法。

* 拓扑特征

拓扑特征是指算子在算法轮函数中的位置特征；比如置换操作，它一般只出现在轮函数的开始或结束的位置进行初始置换和终结置换，因此在架构设计时只需要在映射轮函数的开始和结束的位置存在置换单元。拓扑特征主要针对大算子，这里的“大”是指这些算子对应的电路面积大，如表2-1所示。因此这些大算子对应的功能单元往往是阵列面积的主要部分，比如S盒操作，它一般会占据阵列总面积的一半以上，减少这类复杂单元的冗余数量对于降低阵列面积是十分有效的。

表3- 1文献[1]中的部分功能单元面积对比

|  |  |  |  |
| --- | --- | --- | --- |
| 功能单元 | 面积µm2 | 等效门数 | 相对值 |
| 移位单元 | 19286 | 1929 | 1.0 |
| 模加/减单元 | 12826 | 1283 | 0.7 |
| 逻辑运算 | 3426 | 343 | 0.2 |
| 有限域乘法 | 85473 | 8547 | 4.4 |
| 置换单元 | 49816 | 4982 | 2.6 |
| S盒 | 1160372 | 116037 | 60.2 |

算法的参数特征、组合特征、拓扑特征分别为架构设计中的功能单元设计、组合单元设计和复杂单元的拓扑设计提供依据。参数特征只是为了保证架构中的PE能够提供算法中算子所需要的功能，同时也避免在PE中添加不必要的功能。组合特征对应的组合功能是降低阵列规模的一个有效方法，很多通用可重构中，一个PE往往只会完成一个算子，这种设计虽然简单，但是在完成同等运算的情况下，PE的规模会很大，对应的存储、互连、配置等都会变得更加复杂。拓扑特征是从阵列的面积占比出发，针对复杂功能单元的优化，在密码算法中，如表2-1所示，这样的复杂功能单元主要有S盒、置换、有限域乘法。这些功能单元的电路面积很大，这些功能单元的冗余造成不必要的阵列面积开销。对于简单的功能单元虽然也有优化的空间，但是往往没有什么明显的效果，因此本文在拓扑优化上只针对S盒、置换、有限域乘法这三类复杂功能单元。

* 1. 算法特征提取
     1. 参数特征

参数特征决定了阵列的一些基本属性，比如阵列大小、PE的位宽、PE内部功能单元的种类等。这些参数特征在不同的分组密码算法中都表现出相同或相似性，密码算法的这些共性也是可重构密码处理器的基本理论依据。

文献[1]对DES、IDEA、AES候选算法等41种公开的分组加密的基本运算操作进行了统计，结果如表3-2所示。从表中可以发现，对于这些分组加密算法，它们的操作只有有限的几种，这意味着PE设计时，只需要对这几类基本运算进行支持就足以实现这些算法。

表3- 2 基本运算操作及其使用频率表

|  |  |  |  |
| --- | --- | --- | --- |
| 基本运算操作 | 使用频率 | 基本运算操作 | 使用频率 |
| 逻辑运算 | 97.56% | 模减运算 | 48.78% |
| S盒 | 73.17% | 置换运算 | 24.39% |
| 移位运算 | 85.36% | 有限域乘法运算 | 17.07% |
| 模加运算 | 48.78% | 模乘运算 | 17.07% |

虽然算法有着相同的基本运算操作，但是不同的算法在同一个运算上的要求是不一样的，这些基本运算有着不同的运算模式，在进行架构设计时需要支持具体的运算模式。

表3- 3 基本运算操作的参数特征

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 基本运算操作 | | 运算位宽 | 对应算法个数 | 运算模式 |
| 逻辑操作 | 抑或 | 32 | 36 | 位宽兼容 |
| 非 |  | 0 |
| 或 | 32 | 1 |
| 与 | 32 | 3 |
| 移位 | 移位 | 32 | 4 | 左右移位 |
| 循环移位 | 8 | 3 |
| 32 | 11 |
| 模运算 | 模加 | 16 | 2 | 模216 |
| 32 | 14 | 模232 |
| 模减 | 32 | 1 | 模232 |
| 模乘 | 16 | 1 | 模(216-1) |
| 32 | 1 | 模232 |
| S盒 | |  | 10 | 输入-输出  4-4 6-4 8-8  8-32 6-2 10-8 |
| 有限域乘法 | | 32 | 10 | GF(28) |
| 置换 | |  |  | 输入-输出  64-64 32-48 32-32 32-40  128-128 |

通过表3-3可以发现分组加密算法的基本运算具有如下特征：

1. 分组密码算法中大量使用逻辑操作，但是一般只会用到抑或操作，只有少量的算法会使用其它逻辑操作（与、或、非）。
2. 分组密码算法中大量使用了逻辑移位和循环移位，其移位模式既有固定移位模式（每次移位位数固定不变）又有可变移位模式（每次移位数依赖寄存器的值），移位的位数一般从1位到32位，数据位宽大多为8、32。
3. 分组密码算法涉及的算术运算（乘、加/减）和逻辑操作（抑或、与、或）位宽大多是字节或字节的整数倍，且算术运算大多带有取模操作，取摸操作都是2的幂次，多为28、216、232。
4. 表3-2列出了几种不同的S盒的输入输出模式，常见的有6-4 S盒、4-4 S盒、8-8S盒、8-32 S盒、6-2 S盒、10-8 S盒；其中出现最多的是8-8的S盒。而且根据变化程度和使用方式的不同，分组密码使用的S盒可分为两种方式：一种是每轮操作使用相同的S盒，如DES、AES、Blowfish等；另一种是每轮操作使用不同的S盒，如SERPENT。
5. 置换操作的位宽会比较大，因为更大的数据位宽能有效扩展输出对输入的依赖性。置换操作的位宽会出现64、48比特，甚至128比特的置换在分组密码设计中也较为常见，但是一般128比特的置换都是基于字节的置换，而基于比特置换的位宽最大的只有64比特。
6. 分组密码算法中出现的有限域乘法主要集中在有限域GF(28)上，而且一般都是以矩阵乘法的形式出现。
   * 1. 组合特征

参数特征关注的是单个功能单元的各个参数，组合特征则关注多个功能之间的关系特征，涉及到整个算法的运算流程，因此需要用一个统一的模型来表示算法流程。算法一般会用数据流图（Data Flow Graph，DFG）来表示，图中的顶点表示操作，边表示数据流向。DFG虽然能很清晰地表示出整个算法的操作流程，但是在进行组合特征提取时，会出现很多无法决断的仲裁，如图3-1所示，v2可以和v3组合成（v2，v3），也可以和v4组合成（v2，v4），但是在设计时只能选择一种进行实现。其实从DFG图中很容易发现（v2，v3）组合设计时一个有意义的组合，因为含有这个组合的路径更长，假设其它操作都不进行组合，那么（v2，v3）组合设计可以将整个DFG流程映射到5行PE阵列上。但如果（v2，v4）组合，没有改变长路径，映射依然需要6行PE。



图3- 1 组合选择冲突示例

从图3-1中的示例可以发现，有效的组合必须是关键路径上的组合，只有将关键路径上的某些操作组合才能在映射算法时有效降低映射所需的资源规模。因此需要找出算法流程中的关键路径，从关键路径上提取有效的操作组合。

图论中的关键路径算法是在边表示的活动网络（Activity On Edges network，AOE）上进行的，AOE网络指的是在一个不存在有向环的带权有向图中，用有向边来表示一个工程中的各项活动（Activity），用有向边上的权值表示活动的持续时间（Duration），用顶点表示事件（Event）。对应算法的AOE网络，则是用顶点表示算法的不同阶段，用边表示算法的的各项操作，边的权值表示各操作的电路延迟，边的方向表示数据流向。



图3- 2 算法AOE模型

* + 1. 拓扑特征

2. PE设计方案

3. 算法aov建模、架构建模

算法图建模

第三章中建立了算法的AOE模型，它是图模型的一种。这里的算法图建模则是基于顶点活动网（Activity On Vertex network，AOV），将算法的操作用顶点表示，算法的数据依赖用边来表示。可以很方便地将算法的数据流图转换成AOV网络，数据流图和AOV网络拥有完全一样的结构，只需要将数据流图的操作转换成AOV网络中的点，对这些点赋予对应的属性，然后将数据流图的数据流向转换成AOV网络中的有向边，这样就完成了数据流图到AOV网络的转换。



图5- 5

图xxx 算法AOV模型

**定义4.1（算法AOV图）：**一个架构图是一个二元组ALG=(V,E)，其中V是节点集，表示算法中的操作；E是边集，表示操作之间的数据依赖关系。

如图xxx所示，图中(a)是算法中的数据流程图的一部分，它标示了算法的各种操作以及数据方向，(b)则是与数据流图对应的算法图模型。表xxx列出了对应节点的属性，通过给(b)的节点赋予这些属性，(b)能够描述算法的完整信息。

表xxx 算法图结点属性

表5- 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 节点 | y | 操作属性 | 节点 | y | 操作属性 |
| v1 | 1 | xor | v5 | 3 | lut |
| v2 | 1 | sh | v7 | 1 | xor |
| v3 | 1 | lut | v8 | 1 | au |
| v4 | 2 | lut | v9 | 1 | xor |

架构图建模

上一小节中采用有向图对算法进行建模，利用图的节点描述算法中算子，图中的边描述数据依赖。完成算法的图建模后就可以利用图论算法来提取算法的某些关键特征为架构设计提供依据。在可重构架构中，同样可以对架构进行图建模，利用节点描述PE，边描述架构中的互连。这样算法和架构有了相同的基于图的描述方式，利用图论中的匹配算法可以完成算法到图的匹配映射[1][2]，第五章将会详细讨论算法在架构图上的映射验证问题。

**定义4.2（架构图）：**一个架构图是一个二元组ARG=(V,E)，其中V是节点集，表示架构中的PE；E是边集，表示架构中的数据互连。

和算法图相比，架构图中的节点具有更复杂的功能属性和位置属性，架构图中的边网络也更加庞大。

初始架构中定义了5类PE，因此架构图中的节点有五类不同的功能属性，在架构图中用字符串进行描述。如表4.xx所示，PE\_type1的功能呢属性的字符串描述了第一类PE的19中功能，在进行算法图节点和架构图节点进行匹配时，算法图节点的功能属性必须是架构图节点功能属性的子集，也就是架构图节点对应的PE能够提供算法节点所需的功能，在属性比较时，算法图节点属性字符串是架构图节点属性字符串的子串。架构图中的（x，y）位置属性标识了该节点对应的PE在架构中的位置。在完成算法映射匹配时根据这些位置信息从多个匹配中选择出占用最少行、列资源的结果。

表xxx 架构图结点属性

表5- 3

|  |  |  |  |
| --- | --- | --- | --- |
| PE（节点） | x | y | 功能属性 |
| PE\_type1 | 1 | 1,2 | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_bn\_xorbn\_bnxor\_xorbnxor\_ |
| PE\_type2 | 1 | 3,4 | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_ |
| PE\_type3 | 2 | 1,2,3,4 | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_lut\_xorlut\_lutxor\_xorlutxor\_ |
| PE\_type4 | 3 | 1,2 | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_bn\_xorbn\_bnxor\_xorbnxor\_gfm\_xorgfm\_gfmxor\_xorgfmxor\_ |
| PE\_type5 | 4 | 3,4 | au\_xorau\_auxor\_xorauxor\_xorxorau\_xorxorauxor\_sh\_xorsh\_shxor\_xorshxor\_xor\_xorxor\_xorxorxor\_or\_and\_gfm\_xorgfm\_gfmxor\_xorgfmxor\_ |

如图xxx所示为一个异构组的图模型，每一行的4个PE被抽象成四个点，每个点有与PE对应的功能属性，行与行之间的互连被抽象成从上一行到下一行的有向边，架构中的行间互连支持数据从上一行的PE到下一行的任意一个PE，因此在图模型中上一行的每一个点都有一条有向边到下一行的任意一个点。



图5- 6

图xxx 异构组的图模型

4.算法映射

基于子图同构的映射方案

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5- 2

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5- 3

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。

VF2子图同构算法

图的同构判定是图论学科的基本同题之一，文献[x-x]对这个问题进行了充分的研究。文献[x]提出了Ullmann算法，它是一个可前向剪枝的带有回溯的树搜索过程；文献[x]提出的VF2算法同样基于搜索方法，它利用一个快速计算的启发式规则进行剪枝，这使其性能得到显著提升。

表xxx Ullmann算法和VF2算法对比

表5- 1



VF2算法的核心思想是搜索加剪枝，重点就在于如何剪枝。状态s存储搜索过程中的部分匹配，以及算法需要的其它数据。代表中间状态，和表示当前状态s的部分匹配中图和图中的点，整个算法流程如图xxx所示。

|  |
| --- |
| 1) PROCEDURE Match(s)  2) 输入：中间状态，初始状态满足  3) 输出：两个图之间的所有匹配  4) IF M(s) 覆盖了的所有结点  5) 输出M(s)  6) ELSE  7) 计算可以被包含到M(s)中的候选点对集合P(s)  8) FOREACH P(s)中的点对p  9) IF 候选点对满足加入M(s)的所有约束规则  10) 将p加入M(s)，计算  11) CALL Match()  12) END IF  13) END FOREACH  14) 恢复数据状态  15) END IF  16) END PROCEDURE Match |

图5- 4

图xxx VF2子图同构算法流程

初始化时状态是，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代表的部分匹配包含了中的所有节点，则已经找到了在中同构的子图，搜索结束；否则，在当前的局部匹配基础上，再匹配一个点。找出所以可能进行匹配点对集合；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，是否还有扩展的可能性；如果加入匹配p可行，则将p加入s，递归调用Match()，继续搜索。如果若干次调用Match()后都没有找到同构的子图，则说明当前状态不可能扩展出可行的子图同构匹配；所以将生成新状态时加入的两点匹配p从s中删除，回溯到上一个状态。

在上述算法流程中，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。

先约定几个符号：和表示图和图中的点集。n和m分别表示图和图中的点。表示点n在图G中的前驱，表示点n在图G中的后继。和表示状态s在图和图中，指向当前已经匹配的点集的所有边的源点集合。和表示状态s在图和图中，从当前已经匹配的点集出发的所有边的终点集合。，即当前状态s在图中已经匹配的点集的所有一步邻居。，即图中，除了s中已经匹配的点，和这些点的一步邻居以外的点。

新加入的点p需要同时满足五个规则：-











和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在图中的所有前驱（或后继），必须能在图中m的前驱（或后继）里有相应的点与之对应。同样，对于m在图中的所有前驱（或后继），也必须能在图中n的前驱（或后继）里有相应的点与之对应。

、和都是剪枝策略。其中表示求集合中元素的个数。和表示n在（或）中的前驱（或后继）的数目，必须大于等于m在（或）中的前驱（或后继）的数目。如果不满足，则说明对于图中新匹配的点m，其邻居个数是大于图中n的邻居个数的，最终必然无法完全匹配图中所有的点。

跟和思想类似，只不过考虑的两步邻居。具体是，和中的考虑的邻居是和中的邻居。这些点既跟n相邻，又跟当前匹配中的其他点相邻。而考虑的邻居，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。

算法的搜索空间主要由候选点对集合P(s)决定，在没一个状态s下都对应不同的搜索空间。候选点对的规则为：

1. 如果和都不为空，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；
2. 如果和两个集合都为空，若和都不为空，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；
3. 如果上面四个集合都为空（对于非连通图会出现这种情况）。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。

细粒度的分类讨论，可以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。

Conte, Donatello, et al. "Thirty years of graph matching in pattern recognition."*International journal of pattern recognition and artificial intelligence* 18.03 (2004): 265-298.

Ullmann, Julian R. "An algorithm for subgraph isomorphism." *Journal of the ACM (JACM)* 23.1 (1976): 31-42.

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.

基于VF2算法的映射方案

算法适应性调整

子图同构是指在模板图中存在一个和查询图同构的子图，根据公式xxx可知同构是一个双射关系。在算法图和架构图匹配中，目标是从架构图中找出一个图，这个图包含算法图，但不要求同构，包含是一个从算法图到架构图的单射关系。因此在约束上，映射匹配的约束比子图同构弱。

图xxx描述了架构图和算法图的包含关系，映射{(v1+v2, a1), (v3, a5), (v4, a6), (v5, a7), (v6+v7+v8, a9)}中架构图的子图{a1, a5 ,a6, a7, a9}和对应的算法图不同构，子图中的多条边在算法图中没有对应的边存在，这只满足单射关系。

VF2算法的和规则对图同构的双射关系进行了约束，因此对于映射算法必须对这两个放宽这两个规则，新的规则如下：





在新的规则中，只要求图（架构图）中包含图（算法图）对应的前驱和后继，反向则不再约束。



图5- 7

图xxx 架构图和算法图的包含关系

单纯的VF2算法不考虑点的属性问题，规则和只保证了对应点的前驱和后继的对应关系，它不保证架构图中的点能提供算法图对应的点所包含的操作属性。在进行算法图和架构图建模时，用字符串来描述图中节点的功能属性，因此架构图节点功能属性字符串要包含算法图的操作属性字符串，增加一个属性约束规则如式xxx所示，其中attr(x)表示节点x的功能属性。



最优匹配约束规则

架构图对算法图满足包含关系的子图可能成千上万，这些子图都满足基本的映射要求，但并不都是我们所需要的。映射要考虑行资源、列资源、互联资源的成本，还要考虑映射聚集等因素。在本文提出的映射方案中主要考虑了5类约束，满足这5类约束的最终结果满足使用最少的行、列资源，最简单的数据互连，以及尽可能往架构的右上聚集。

对算法图中的节点加入拓扑信息a，v.a表示算法图中的点v出现在横向上的位置；对架构图中的节点加入拓扑信息（x, y），其中v.x表示架构图中点v代表的PE出现在纵向上的位置，v.y表示架构图中点v代表的PE出现在横向上的位置。S表示匹配算法找出的所有可能的匹配集合， 为S中的一个元素，表示一个有效的匹配；(,)表示一个有效匹配中的一个匹配点对。











5.结果对比

* 明确列出对比对象、对比目标的指标和代价、对比方案、对比条件以及对比结果。

对比对象的目标水平为工业界当前一流水平或学术界2年前一流水平。自行复现的对比对象需要验证正确性。

对比目标在指标和代价上必须量化，局部的间接指标必须能够反映到系统指标。

对比方案描述仿真验证或真实系统测试的基本思路和步骤。

对比条件描述仿真模型或真实系统的配置和参数设置，以及测试输入向量（仿真或真实数据）。

对比结果通过图表方式展示，有分析结论。

表2 不同架构实现算法所需功能单元对比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 本文 | AU | SH | PER | LOU | 4\*LUT256\*8 | GFM |
| AES | 12 | 12 | 4 | 12 | 4 | 4 |
| DES | 12 | 12 | 4 | 12 | 4 | 4 |
| SM4 | 12 | 12 | 4 | 12 | 4 | 4 |
| TWOFISH | 24 | 24 | 8 | 24 | 8 | 8 |
| RC5 | 16 | 16 | 5.3 | 16 | 5.3 | 5.3 |
| BLOWFISH | 12 | 12 | 4 | 12 | 4 | 4 |
|  |  |  |  |  |  |  |
| TH |  |  |  |  |  |  |
| AES | 8(67%) | 8(67%) | 4(100%) | 48(400%) | 16(400%) | 0(0%) |
| DES | 16(133%) | 16(133%) | 8(200%) | 96(800%) | 32(800%) | 0(0%) |
| SM4 | 16 | 16 | 8 | 96 | 32 | 0 |
| TWOFISH | 24 | 24 | 12 | 144 | 48 | 0 |
| RC5 | 16 | 16 | 8 | 96 | 32 | 0 |
| BLOWFISH | 16 | 16 | 8 | 96 | 32 | 0 |
|  |  |  |  |  |  |  |
| Cyptoraptor |  |  |  |  |  |  |
| AES | 8 | 8 | 8 | 8 | 56 | 0 |
| DES | 12 | 12 | 12 | 12 | 84 | 0 |
| SM4 | 16 | 16 | 16 | 16 | 112 | 0 |
| TWOFISH | 20 | 20 | 20 | 20 | 140 | 0 |
| RC5 | 16 | 16 | 16 | 16 | 112 | 0 |
| BLOWFISH | 12 | 12 | 12 | 12 | 84 | 0 |
|  |  |  |  |  |  |  |
| RCPA |  |  |  |  |  |  |
| AES | 8 | 8 | 8 | 16 | 16 | 8 |
| DES | 12 | 12 | 12 | 24 | 24 | 12 |
| SM4 | 16 | 16 | 16 | 32 | 32 | 16 |
| TWOFISH | 20 | 20 | 20 | 40 | 40 | 20 |
| RC5 | 16 | 16 | 16 | 32 | 32 | 16 |
| BLOWFISH | 12 | 12 | 12 | 24 | 24 | 12 |

表3 不同架构实现算法的性能面积比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 架构 | 工艺 | 主频 | 算法 | 映射面积(mm^2) | 面积减少 | 算法性能 | 性能面积比 | 性能面积比对比 |
| 本文 |  |  | AES | 0.1929094 |  |  |  |  |
| DES | 0.1929094 |  |  |  |
| SM4 | 0.1929094 |  |  |  |
| TWOFISH | 0.3858188 |  |  |  |
| RC5 | 0.2572125 |  |  |  |
| BLOWFISH | 0.1929094 |  |  |  |
| 清华项目中的架构 |  |  | AES | 0.407476 | 53% |  |  |  |
| DES | 0.814952 | 76% |  |  |  |
| SM4 | 0.814952 | 76% |  |  |  |
| TWOFISH | 1.222428 | 68% |  |  |  |
| RC5 | 0.814952 | 68% |  |  |  |
| BLOWFISH | 0.814952 | 76% |  |  |  |
| Cyptoraptor |  |  | AES | 0.873618 | 78% |  |  |  |
| DES | 1.310427 | 85% |  |  |  |
| SM4 | 1.747236 | 89% |  |  |  |
| TWOFISH | 2.184045 | 82% |  |  |  |
| RC5 | 1.747236 | 85% |  |  |  |
| BLOWFISH | 1.310427 | 85% |  |  |  |
| RCPA |  |  | AES | 0.3670776 | 47% |  |  |  |
| DES | 0.5506164 | 65% |  |  |  |
| SM4 | 0.7341552 | 74% |  |  |  |
| TWOFISH | 0.917694 | 58% |  |  |  |
| RC5 | 0.7341552 | 65% |  |  |  |
| BLOWFISH | 0.5506164 | 65% |  |  |  |