

# 一种基于可重构系统的动态软硬件划分算法

周立秋, 李仁发, 曾庆光

1 湖南大学计算机与通信学院, 湖南长沙 (410042)

E-mail: [fdws989@163.com](mailto:fdws989@163.com)

**摘要:** 面向可重构片上系统的软硬件划分从一开始就受到研究人员关注, 动态软硬件划分算法的难点在于其对实时性要求比较高。本文通过分析基于函数的动态软硬件划分问题, 提出了通过贪婪算法和禁忌搜索算法相结合的方式来实现对动态软硬件划分实时需求问题的解决。

**关键词:** 动态可重构; 软硬件划分; 贪婪算法; 禁忌搜索算法

**中图分类号:** TP301.6 TP302.7

## 1 引言

软硬件划分伴随着软硬件协同设计 (Hardware-Software Co-Design) 方法的产生而问世, 它是软硬件协同设计的关键步骤。随着可重构计算技术从理论走向实际应用, 软硬划分问题也从片上系统设计扩展到了实现可重构部件最大化利用的问题上。因此, 基于可重构系统的软硬件划分可以界定为软硬件划分研究的一个分支。面向可重构片上系统的软硬件划分从一开始就受到研究人员的关注, 其划分方法可以分为静态划分和动态划分。静态划分是基于利用可重构部件的重构特性, 扩充传统软硬件划分方法, 在可重构资源有限的前提下, 运用以时间换空间的方式, 通过合适的调度算法, 把更多的任务交替安排到可编程器件上执行, 以获得更好的性能<sup>[1][2]</sup>。虽然这种方法运用了重构资源动态可重构的特性, 但是, 由于软硬件划分在系统设计阶段就已经确定了, 动态的重构只是确定的划分的实现。可重构系统的动态划分是一种运行时的划分方法, 目前的方法一种是基于提取程序中高频度执行的循环结构, 将其实现在可重构部件中<sup>[3]</sup>, 另外一种是从高层着眼, 把配置到可编程器件的硬件电路看作一种硬件进(线)程, 通过构建支持硬件进(线)程的操作系统, 实现软硬件进(线)程的相互迁移和切换<sup>[4]</sup>。

本文将基于一种函数级的动态软硬件划分模型, 提出适应于动态调用的算法。

## 2. 基于函数的动态软硬件划分

### 2.1 目标体系结构

本文的目标体系结构采用单处理器, 单可重构器件, 两者各有一个访存端口能同时进行存储内容的读写操作, 并通过共享总线和系统内存的方式进行通信与同步。同时可重构器件只有一个配置端口, 各模块的配置需要串行的进行。体系结构图如下:

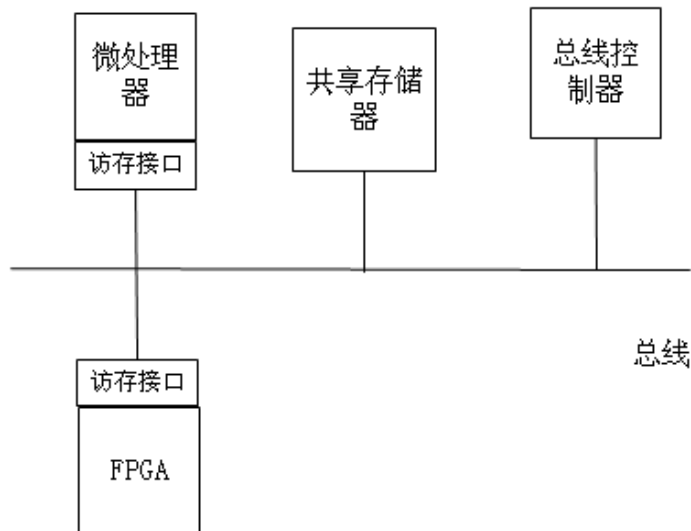


图 2.1 可重构片上系统体系结构

由于处理器与可重构器件具有天然的并行性，故而，本文假设，处理器和可重构器件是可以并行执行的。

## 2.2 软硬件函数库的基本特征

由于软硬件函数库中的函数既可以用软件来实现，又可以用硬件来实现，因此，对于其中任意的一个函数如果其被划分到软件执行，将调用其软件实现部分的代码，如果其被划分到硬件执行，将调用其封装好的硬件模块的软件执行代码。硬件模块包括两个部分，一部分是配置文件实现硬件模块的基本功能，一部分是软件代码，实现微处理器对硬件模块的调用。如果从软硬件执行的性能来分析，为了实现硬件加速的基本目标所有软硬件函数库中的函数都具有硬件执行速度（包括硬件执行时间，通信的时间代价）快于软件执行速度的基本特征。这是因为如果软硬函数库中的函数的硬件执行速度比软件执行速度慢，那么将其放入硬件执行将很难发挥可重构资源的特性。

总之，软硬件函数库中的函数既可以用软件来实现，又可以用硬件来实现，且在不考虑重构代价的情况下，硬件实现的速度都快于软件实现。

## 2.3 软硬件函数的整体封装

虽然程序人员只需要调用软硬件函数库中的软件或者硬件函数，但是事实上的调用过程应该是一个包含两者的整体调用。而实际实现才是根据划分过程来确定。

关于这个问题的原因实际上我们需要从程序的编译过程来分析。假设程序最初运行时，我们将所有的软硬件函数都划分给软件执行，如果程序中仅仅将软硬件函数软件实现部分的代码放入编译器中进行编译，那么在运行时我们希望调用硬件来实现该函数时将无从处理，因为编译器产生的中间代码中仅仅包涵了软件部分的实现。同样的反过来考虑也一样。因此编译阶段不能同时考虑软件和硬件实现，将给动态软硬件划分的实现造成巨大的困难。所以，函数需要一个整体的封装过程。软件人员调用软硬件函数时不需要知道其确切的是由软件还是硬件实现，但是我们规定他需要对所调用的软硬件函数给出一个确定状态的定义。而该状态参数由软硬件划分程序来控制。例如下面的小段 C 程序：

```
...  
int f=fun.value; /*fun.value 由软硬件划分程序来确定*/  
void fun(...,f); /*所调用的硬件函数*/  
...
```

这一小段程序中整型变量  $f$  由软硬件划分程序来确定，软件编程人员需要调用一个硬件函数时需要给其中的最后一个整型参数赋值，而其值需要由软硬件划分程序来确定。

那么被程序人员调用的  $fun$  函数则通过  $f$  来确定该函数实际运行中会由软件还是硬件来实现。例如下面的小段 C 程序：

```
...  
Void fun(...,f)  
{  
If(f==1)  
SFun(...); /* 软件实现*/  
If(f==0)  
HFun(...); /*硬件实现*/  
Else  
false;  
}
```

从上面的程序我们可以知道，如果编程人员调用了  $fun$  函数，那么就等同于同时将软件和硬件实现放入编译器中编译，其中间代码必然同时包涵了软件和硬件实现，而其实际如何实现则要根据参数  $f$  来确定，而  $f$  又由软硬件划分程序来确定，只要在程序运行的过程中定期地调用软硬件划分算法，那么  $f$  值就将动态地跟随划分结果而变化，从而我们就能实现动态的软硬件划分，而其过程软件人员仅仅需要做一个简单的赋值操作，这和底层硬件是毫无关系的。因此本文将假设该动态软硬件划分方法能够具体实现的前提下，分析划分算法的具体问题。

## 2.4 划分的基本问题

对于软硬件划分问题我们首先要明确划分的对象，本文中就是被调用的经过了整体封装的软硬件协同函数库中的抽象函数，应用程序中调用的其它函数可以不用考虑，因为只有经过了整体封装的软硬件协同函数库中的函数包含了软件和硬件实现的部分。而被调用的整体封装函数可以通过编译前端的词法和语法分析来获取，在词法和语法分析中只要遇到软硬件函数库中的函数就可确定这些函数对应的硬件模块配置文件，将其作为系统代码的一部分，与软件代码一起保存到系统非易失性存储器中。同时，构造一个与这些被引用函数对应的列表，记录每个被引用函数的软硬件划分决策结果，以便动态链接控制的代码读取并进行相应链接。系统初次启动时，可默认所有的划分决策为软件实现。

其次，我们要明确划分的目标。我们的目标是在可重构资源有限的情况下，实现系统运行时间的最优化。由于资源的有限性我们需要考虑各个待划分函数所对应的各方面的数据来确定我们的划分算法。我们将其分为确定性数据和系统运行时数据两种。确定性数据包括该函数的软件执行时间  $st_i$ ，硬件执行时间  $ht_i$ （我们假设包括了通信代价），重构时间  $rt_i$ ，所需的可重构资源  $a_i$ ，硬件总的可重构资源  $A$ ，运行时数据包括当前函数的配置状态  $c_i$ （其中  $c_i=0$  时表示划分到软件执行， $c_i=1$  时表示划分到硬件执行），当前剩余的可用资源  $AR$ ，

截止目前的各函数执行的次序次数。其中确定性数据的获取较为简单，由于都只针对同一个函数，软件执行时间通过软件模拟器对该函数软件实现的代码进行模拟实现来获得，硬件执行时间，重构时间，各函数所需资源通过 synopsys 的 design compil 进行硬件综合来获得，硬件总的资源可以通过可重构资源管理器来获取。而运行时数据除了当前配置的函数，和当前剩余的可用资源可以通过可重构资源管理器获取外，截止目前各函数的执行次序和次数都需要通过其它的渠道获得。这里我们通过特定的高速缓存来记录所执行过的各个函数，通过高速缓存来记录程序的运行状态在文献[5]中有提到。同时我们仅仅记录最近执行的若干次的软硬件函数，因此它是一个数据串的结构。如下图所示：

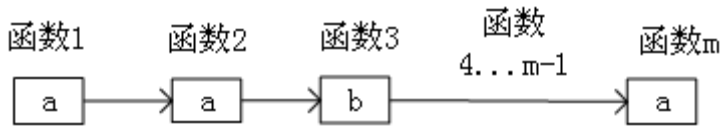


图 2.2 m 个最近执行函数数据结构

该串中存在多个的不同软硬件函数，并且这些函数很多都是重复出现的函数。虽然，整体的分析所有执行过的函数有其总体定性的区分的意义，但是考虑到我们的软硬件划分具有动态的性质，并非执行次数在整体突出的函数在所有的划分区间都比较突出。因此我们认为最近执行比较多的函数，将继续在后面的一个划分时间区间中被大量执行。这是符合大多程序都存在比较多的循环结构的性质的。下面我们根据所取得的数据串给出每个软硬件函数  $f_{uni}$

$$\text{下个周期的预计执行次数的计算方法: } g_i = 2 \sum_{j=1}^m ((1 - \frac{j-1}{m})(1 - state_{ij})) + 1 \quad \text{式 1}$$

其中  $m$  表示串长。  $state_{ij}$  表示函数  $f_{uni}$  是否出现在串的第  $j$  个位置，如果是则  $state_{ij} = 0$ ，如果不是  $state_{ij} = 1$ 。

下面我们为每个软硬件函数制定一个时间贡献度  $p_i$

$$p_i = (st_i - (ht_i + \frac{rt_i(1-c_i)}{g_i}))g_i \quad \text{式 2}$$

其中  $p_i$  越大表示该函数如果分配到硬件执行将获取比较大的时间收益。

$$\sigma_i = \frac{p_i}{a_i} \quad \text{式 3}$$

$\sigma_i$  表示函数单位面积时间贡献度，  $\sigma_i$  越大则该函数的单位面积时间贡献度就越高。

根据运行时间最优化的目标，我们的目标函数表达式如下：

$$\max(\sum_{i=1}^n p_i)$$

$$s.t. \quad \text{式 4}$$

$$A \leq \sum_{i=1}^n a_i$$

## 2.5 关于进一步缩小划分对象的分析

由于重构开销巨大，往往，  $st_i < ht_i + rt_i$  这和我们讨论的划分粒度关系密切。函数级的

粒度使得实现函数功能的 IP 核比较大，从而其重构时间的开销以及需要的可配置资源都会比较多。在实际的实验中重构的时间开销往往以秒级来计算，而每个 IP 核所占的可配置资源都是相当多的。因此，在不考虑这些限制的情况下对可重构资源进行盲目地重构是很难达到提高系统性能的目的。

从式 2 可以发现，我们在设定时间贡献度时，已配置资源将获得了一个重构时间的加成。同时考虑到 2.2 节的假设，对于  $c_i=1$  即函数已经被配置时，该函数的  $p_i$  值都会大于 0，而  $c_i=0$  时， $p_i$  的值如果没有经过  $g_i$  的强化，则基本会因为重构开销过大而使得  $p_i$  小于 0。但是如果  $g_i$  足够大  $p_i$  就会大于 0，那么该函数就有了同已配置函数和  $st_i - ht_i$  的值非常高的函数竞争资源的前提。其变化情况如下图：

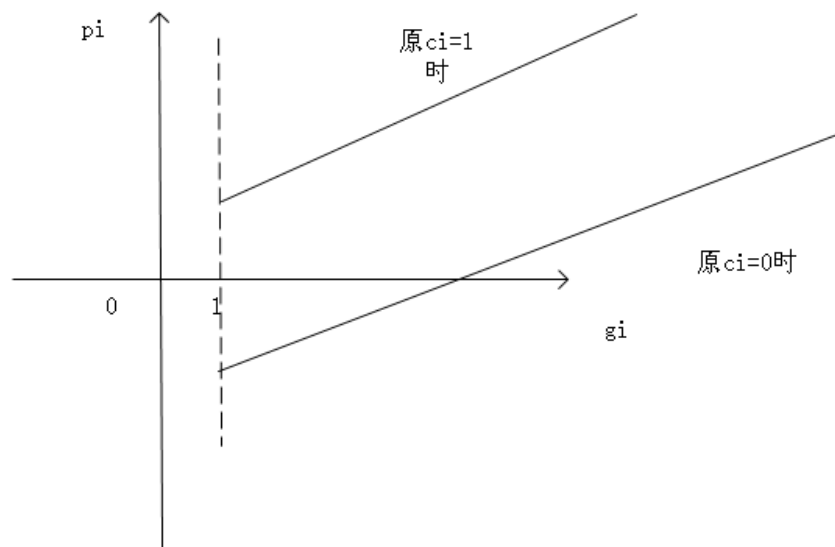


图 2.3  $g_i$  和  $p_i$  的关系

从图中我们可以知道， $g_i$  是决定  $p_i$  大小的关键因素。 $st_i - ht_i$  是斜率反映了其变化速度。从而我们可以得出这样的结论，在划分过程中只有在前段时间大量执行并且没有被分配到硬件资源的软硬件函数才会获得重构的机会。当然，是否重构还需要通过可用的可配置资源的限制条件来进行整体的分析。因此我们最终的解将由已配置函数和最近执行的函数中进行选取。这将大大减小我们划分问题的规模。

通过上面的分析我们将软硬件划分的对象缩小为，已配置函数和最近执行的函数。这有利于缩小软硬件划分算法规模，从而，使其更适应于动态调用。从下表中我们可以很明显地发现划分规模对算法时间的巨大影响：

表 2.1 各算法的运行时间

算法		T10	T20	T40	T100	T200	T400
SASM	最好次数	100	100	100	100	60	13
	最短时间	0.06	0.19	0.68	5.66	32.82	201.18
	最长时间	0.08	0.23	0.99	10.01	55	333.5
	平均时间	0.066	0.207	0.821	7.610	44.252	264.797
SAIM	最好次数	100	100	100	92	47	64
	最短时间	0.07	0.19	0.56	3.52	13.44	83.76
	最长时间	0.08	0.22	0.78	6.39	28.77	172.92
	平均时间	0.072	0.203	0.663	4.952	23.105	120.666
TS	最好次数	100	93	100	92	96	95
	最短时间	0.002	0.01	0.07	0.78	3.71	51.21
	最长时间	0.01	0.03	0.13	3.1	31.33	129.07
	平均时间	0.004	0.021	0.085	1.445	12.443	86.889
SSS	最好次数	100	100	100	91	82	95
	最短时间	0.02	0.05	0.23	1.36	5.31	20.31
	最长时间	0.03	0.08	0.28	1.56	5.93	23.23
	平均时间	0.027	0.075	0.254	1.469	5.634	21.752

该表引自文献[6]在上表中 SASM、SAIM 为模拟退火算法，TS 为禁忌搜索算法，SSS 为基于搜索空间平滑技术的算法。其中在平均算法执行时间方面 T100 与 T20 相比，不同的算法的执行时间基本保持在 20 倍左右。因此，缩小划分对象的方法无疑为适应动态划分的算法的提出提供了巨大支持。

### 3 划分算法

#### 3.1 划分算法的选择

为了使算法能够适应动态调用的特点，必须能够实现一定的实时性。所以，我们需要找到一种足够快速的划分方案。虽然 2.4 节我们将划分的对象充分地缩小为已配置函数和最近执行的函数，划分的对象可能在某些情况只有已配置函数，但是，在大多数情况下整个目标空间仍然可能比较大。如果假设在某次划分状态下被划分的函数规模基本保持在 20 个函数左右。那么对于精确算法例如：分支界限法，动态规划和整数线性规划等虽然能够保证找到最优解，但是对于一个  $2^{20}$  左右的完全搜索显然难以满足实时的要求。启发式算法国内外研究的比较多，例如：遗传算法，模拟退火算法，以及最近的粒子群优化算法，神经网络算法，蚁群算法等等，但是这些算法仍然是在  $2^{20}$  的解空间中通过迭代来运行获取最优解，其时间开销仍然比较大。贪婪算法也是启发式算法的一种，该算法只要获取一定的问题相关信息，通过计算  $\sigma_i$  的值，并尽可能将  $\sigma_i$  值最大的函数划分给硬件执行，就能获取所要求的解，因此该算法的执行非常简单。但是贪婪算法非常容易陷入局部最优。综合上面算法的特征，如果能够在可容忍范围内实现贪婪算法在大多数情况下的执行，并以一种比较高速的迭代搜索算法配合其对不可容忍状态进行优化，将能最小化地实现划分算法的执行时间。

对于贪婪算法的可容忍范围需要配合软硬件划分的实际问题来考虑。我们在前面讨论过

关于函数粒度的重构，每个函数对重构面积的实际需求是十分巨大的。往往同一时刻能在FPGA上执行的函数不会很多。而对于贪婪算法，其产生的解最后的剩余空间仅仅比当前划分函数中被划分到软件执行的函数的最小需求面积小。而该最小面积一般都比较大，所以最终解往往会存在一个比较大的剩余空间。同样的原因对于最优解，往往只是填补了部分的空间，很少能做到完全的填补。并且由于最优解并不是都采用 $\sigma_i$ 值最大的函数划分到硬件执行的方案，所以对已划分函数来讲，它们整体的单位时间贡献度肯定要比贪婪算法的划分结果低。同时执行其它迭代性的搜索算法也将抵消掉部分最优解带来的优化时间。所以，当贪婪算法执行的划分方案的剩余空间不是太大时，我们认为这次划分是接近于最优划分的。我们设定可容忍度为 $abide$ 。超过 $abide$ 是指空间利用率小于 $abide$ 的情况，否则就是在 $abide$ 的容忍范围内。

下面需要确定的是，对于在超过 $abide$ 的贪婪算法执行结果，需要调用什么算法来执行优化。由于本文的划分对象不是一个关系图的结构，因此，基于图的划分算法，例如：神经网络算法，蚁群算法都不适合于解决我们的问题。而基于多点并行搜索的算法虽然速度比较快但是容易陷入局部，例如：遗传算法，粒子群优化算法。因此，我们将模拟退火算法，基于空间平滑技术的划分算法，以及禁忌搜索算法作为考虑的重点。其中模拟退火算法是根据可控制接受度来接受较差解来跳出局部，基于空间平滑技术的划分算法通过将不同节点的参数向平均值的方向移动，将划分空间中的局部最优解减少的方式来跳出局部，而禁忌搜索算法则采用了禁忌策略尽量避免迂回搜索，它是一种确定性的局部极小突跳策略。这几个算法采用了不同的策略跳出局部从而实现全局搜索。文献[6]对这几个算法进行了实现，其执行效果效果如下图：

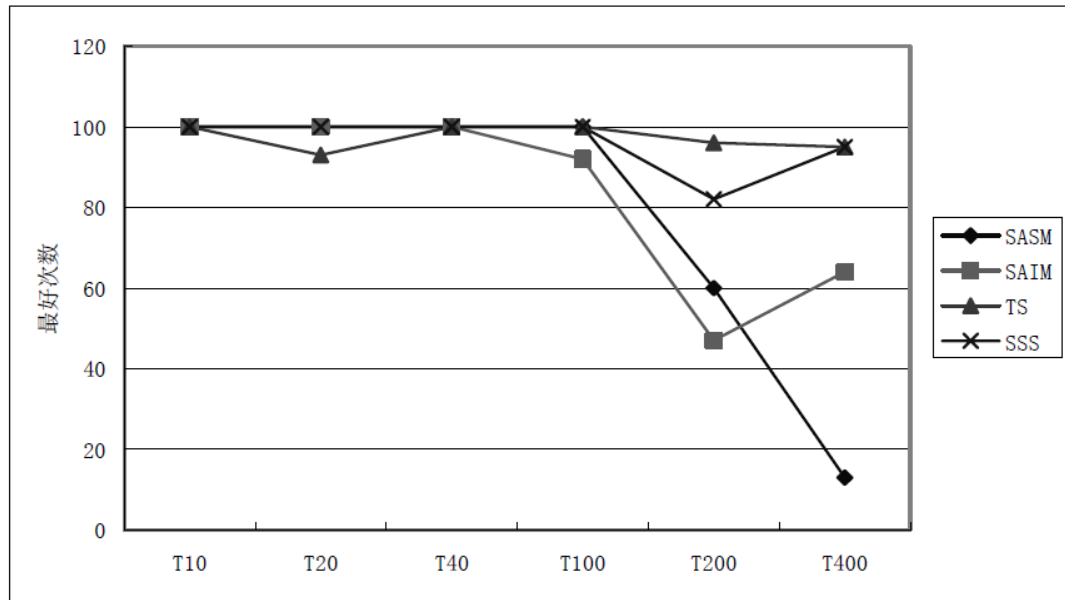


图 2.4 SASM、SATM、TS、SSS 算法求解质量对比

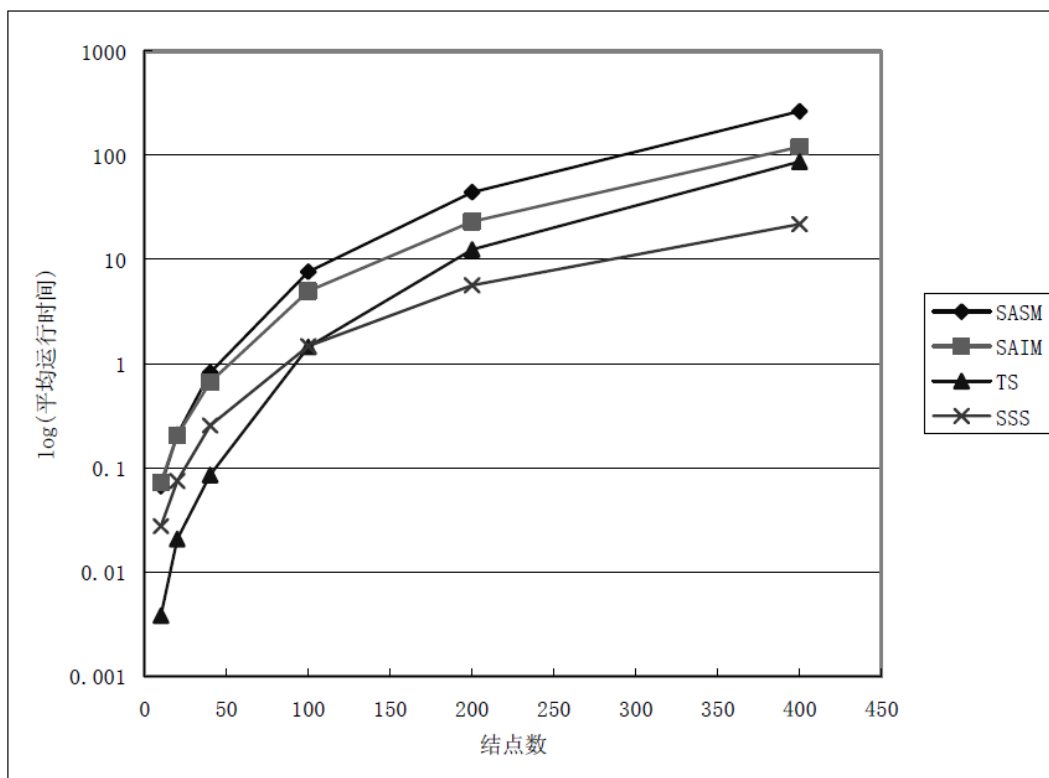


图 2.5 各算法找到最好解平均运行时间对数曲线

前面, 本文分析了所要解决的划分对象, 是一个比较小的群体。而从上面的表和图中我们可以知道, 禁忌搜索算法在节点数即划分对象比较少时有很突出的性能优势。同时算法在节点数为 20 时的平均执行时间仅为 0.021 秒, 并且该算法是被按几率调用执行的, 因此, 我们认为该算法同贪婪算法的结合执行是比较能适应动态划分的要求的。

## 3.2 算法的具体实现

### 3.2.1 贪婪算法的具体实现

对于包含  $n$  个待划分函数  $V = \langle V_1 \dots V_n \rangle$ , 计算出其所对应的所有单位面积时间贡献度  $\langle \sigma_1 \dots \sigma_n \rangle$  的值, 在  $\langle \sigma_1 \dots \sigma_n \rangle$  中找到最大的值, 如果其对应的面积需求满足面积约束则将其划分到硬件执行, 否则划分到软件执行, 并将其从  $\langle \sigma_1 \dots \sigma_n \rangle$  中删除。重复这样的操作直到  $\langle \sigma_1 \dots \sigma_n \rangle$  为空。

### 3.2.2 禁忌搜索算法

禁忌搜索算法是由美国科罗拉多州大学的 Fred Glover 教授在文献[7][8]中提出的一个用来跳出局部最优的搜寻方法。禁忌搜索是对局部邻域搜索的一种扩展, 是一种全局逐步寻求最优算法。禁忌搜索算法中充分体现了集中和扩散两个策略, 它的集中策略体现在局部搜索, 即从一点出发, 在这点的邻域内寻求更好的解, 以达到局部最优解而结束, 为了跳出局部最优解, 扩散策略通过禁忌表的功能来实现。禁忌表中记下已经到达的某些信息, 算法通过对禁忌表中点的禁忌, 而达到一些没有搜索的点, 从而实现更大区域的搜索。

对于该算法在本文中的应用我们首先需要确定如何进行领域的搜索产生新解。本文将采用一种基于定范围离散度模型的搜索方法对当前解的领域进行搜索。

### 3.3 定范围离散度模型的概念

#### 3.3.1 什么是 $f$ 变换

为了便于在多维 0-1 空间中对搜索范围和搜索精度进行描述和计算机实现, 本文提出了定范围离散度模型的概念。

设  $P = \{P_1, \dots, P_n\} = \{(P_{11}, \dots, P_{1m}), \dots, (P_{n1}, \dots, P_{nm})\}$  其中  $P$  是指一个群体,  $P_i, i \in [1, n]$ , 指一个个体,  $P_{ij}, j \in [1, m]$ , 指个体  $P_i$  中的一个节点,  $P_{ij} = 0$  或  $1$ 。若  $P_{ij} = (P_{ij} + 1) \% 2$ , 则表示  $P_{ij}$  进行了一次变异。

定义 1: 设  $P_k$  和  $P_l$  都为  $P$  中的一个个体。那么  $P_k$  和  $P_l$  的距离  $D = |P_{k1} - P_{l1}|, \dots, |P_{km} - P_{lm}|$ 。为了给出个体  $P_i$  的搜索能力我们提出以下变换:

$f(P_i, d, e)$ , 其中  $d$  和  $e$  为常数, 且  $d$  为整数,  $d \in [1, m]$ , 表示变异节点数,  $e \in [0, 1]$  表示变异概率,  $P_i$  表示中心个体,  $f$  表示一次变换, 其变换规则为:

以  $P_i$  为中心任意选取  $d$  个节点作为变异节点, 每个变异节点以概率为  $e$  进行变异。

#### 3.3.2 $f$ 变换规则的意义

定义 2: 设  $g_0, \dots, g_d$  分别为通过一次  $f$  变换产生与中心个体距离 (中心距) 分别为  $0, \dots, d$

的个体的概率, 若  $d$  为奇数时有  $\sum_{i=0}^{\frac{d+1}{2}} g_i > \sum_{j=\frac{d+1}{2}}^d g_j$ , 若  $d$  为偶数时有  $\sum_{i=0}^{\frac{d-1}{2}} g_i > \sum_{j=\frac{d}{2}+1}^d g_j$ , 则称

该变换产生的个体较集中。

定理 1: 当  $e < \frac{1}{2}$  时,  $f$  变换产生的个体较集中。

证明:

设  $e = \frac{1}{s}, a > b$  且  $a + b = d, a$  和  $b$  为正整数。那么通过  $f$  变换产生与中心个体距离为  $a$  和  $b$  的新个体的概率分别为

$$g_a = (1 - \frac{1}{s})^{d-a} (\frac{1}{s})^a C_d^a \text{ 和 } g_b = (1 - \frac{1}{s})^{d-b} (\frac{1}{s})^b C_d^b。$$

$$\because a + b = d \text{ 故 } g_a = (1 - \frac{1}{s})^{d-b} (\frac{1}{s})^b C_d^b (1 - \frac{1}{s})^{b-a} (\frac{1}{s})^{a-b} = g_b \left( \frac{1}{n-1} \right)^{a-b}$$

由于  $a > b$  故当  $n > 2$  时,  $g_a > g_b$ , 依此类推, 若  $d$  为奇数时有  $\sum_{i=0}^{\frac{d+1}{2}} g_i > \sum_{j=\frac{d+1}{2}}^d g_j$ , 若  $d$  为

偶数时有  $\sum_{i=0}^{\frac{d-1}{2}} g_i > \sum_{j=\frac{d}{2}+1}^d g_j$ , 根据定义 2 可得, 定理 1 成立。

虽然定理 1 能够给出  $f$  变换的一种定性的特征关系, 但是还是无法表示  $f$  变换集中程度的高低, 下面我们就来说明这一关系。

定义 3: 若有变换  $f_1(P_i, d, e_1)$  和  $f_2(P_i, d, e_2)$ , 如果通过  $f_1$  变换能够得到中心距较小的个体的概率比  $f_2$  大, 则称  $f_1$  变换比  $f_2$  变换集中, 或  $f_2$  变换比  $f_1$  变换离散。

定理 2: 若有变换  $f_1(P_i, d, e_1)$  和  $f_2(P_i, d, e_2)$ , 如果  $e_1 < e_2 < \frac{1}{2}$  则称  $f_1$  变换比  $f_2$  变换集中, 或  $f_2$  变换比  $f_1$  变换离散。

证明: 如果  $e_1 < e_2 < \frac{1}{2}$ , 那么对于任意的确定距离  $a > \left| \frac{d}{2} \right|$ , 通过  $f_1$  和  $f_2$  变换产生与中心个体距离为  $a$  的新个体的概率分别为:

$$g_{a1} = (1 - e_1)^{d-a} e_1^a C_d^a \text{ 和 } g_{a2} = (1 - e_2)^{d-a} e_2^a C_d^a$$

$$\frac{g_{a1}}{g_{a2}} = \frac{(1 - e_1)^{d-a} e_1^a C_d^a}{(1 - e_2)^{d-a} e_2^a C_d^a} = \frac{(1 - e_1)^{d-a} e_1^a}{(1 - e_2)^{d-a} e_2^a} < \left( \frac{1 - e_1}{1 - e_2} \right)^{\left| \frac{d}{2} \right|} \left( \frac{e_1}{e_2} \right)^{\left| \frac{d}{2} \right|}$$

如果能证明  $\frac{(1 - e_1)e_1}{(1 - e_2)e_2} < 1$  就可以证明  $g_{a1} < g_{a2}$

$$\text{由 } \frac{\frac{1}{4} - \left( e_1 - \frac{1}{2} \right)^2}{\frac{1}{4} - \left( e_2 - \frac{1}{2} \right)^2} < 1 \text{ 可知 } g_{a1} < g_{a2},$$

依此类推有当  $d$  为奇数时  $\sum_{j=\frac{d+1}{2}}^d g_{1j} < \sum_{j=\frac{d+1}{2}}^d g_{2j}$ , 当  $d$  为偶数时  $\sum_{j=\frac{d}{2}+1}^d g_{1j} < \sum_{j=\frac{d}{2}+1}^d g_{2j}$ 。说明通

过  $f_1$  变换能够得到中心距较高的个体的概率比  $f_2$  小, 那么必定有  $f_1$  变换能够得到中心距较小的个体的概率比  $f_2$  大, 那么定理 2 成立。

由定理 1 和定理 2 我们可以得出以下结论: 当  $e < \frac{1}{2}$  时  $f$  变换较集中, 且其集中度随  $e$  的减小而增加。

因此我们可以给  $f$  变换和其映射的个体定义以下关系:

定义 4 设群  $P = \{P_1, \dots, P_n\}$  为  $f(P_i, d, e)$  经过  $n$  次变换得到的所有个体的集合, 那么群  $P$  的分布将以  $P_i$  为中心, 以最大中心距为  $d$ , 有倾向的一种分布, 其倾向度为  $\frac{1}{e}$ , 倾向度越大, 群  $P$  越集中, 倾向度越低, 群  $P$  越离散。

总之,  $f$  变换是确定性概念的一次不确定的量化过程, 首先,  $d$  限定了变换的范围,  $e$  限定了变换的离散度,  $P_i$  确定了变换的中心; 其次, 经过变换产生的个体  $P_i$  是由  $(P_i, d, e)$  确定的一个不确定的个体, 因此, 我们称由  $f(P_i, d, e)$  经过若干次变换产生的个体集合  $P = \{P_1, \dots, P_n\}$  为一个定范围离散度模型。

### 3.3.3 定范围离散度模型的意义

定范围离散度模型反映了在多维 0-1 空间中一个确定个体同其通过  $f$  变换产生的个体之间的定性关系, 且  $f$  变换只能由计算机完成。通过该模型的形成过程, 我们可以实现自定义的搜索, 即以  $P_i$  为中心, 以  $d$  为范围, 以  $\frac{1}{e}$  为精度, 以  $f$  变换次数为搜索次数的搜索过程。并且  $d$  和  $e$  可以在搜索过程中根据实际情况进行动态变化, 以满足搜索对精度和求变的不同要求。由于该模型是定义在多维 0-1 空间中的, 因此, 可以预见它对于求解分支界限法无法承受其复杂度的 0-1 规划问题例如: 背包问题、人员安排、代码选取、可靠性等都可能发挥其作。

### 3.4 关于限定条件对搜索的影响

从 3.3 我们知道通过  $f$  变换是可以实现对某个已知个体的领域进行搜索的。但是, 在本文的问题中其搜索产生的新的个体的可行性受到了面积约束的限定。在满足  $A \leq \sum_{i=1}^n a_i$  的前提下划分才是可行的划分。因此, 无条件地运用  $f$  变换将可能生成非常多的无效解。虽然判断有效与否的方法很简单, 但是, 大量的累积这些无效计算很可能产生一个比较客观的运行时间消耗。因此我们需要采取一些措施以尽量避免这种无效解。

在已有的避免无效解产生的方法中, 文献[9]提出了在遗传算法产生初始群体时在其中大量加入可行个体的方法来避免大量的无效个体的产生。但是这种方案在本文中无法实现。鉴于很多的无效个体都是因为将划分到软件的个体划分到硬件执行, 而划分到硬件的个体在变换中被划分到软件执行的力度不够而造成的。例如: 对个体  $P_i$  如果节点  $P_{i1}, P_{i2}$  的当前解都是 1 表示划分到软件执行, 通过  $f$  变换其值变为了 0 表示被划分到硬件执行, 但是相应的将当前解是 0 而变换后变为 1 的节点却没有, 那么就很可能产生一个不可行的解。本文将对每个新个体的产生采用两次同样的  $f$  变换的方式进行, 即对当前划分中是 0 的节点和是 1 的节点各进行一次。相同的变换方式使得从 0 到 1 和从 1 到 0 的转换力度相同, 虽然不能杜绝无效解的产生。但是对减少无效解必定能发挥巨大的作用。

### 3.5 禁忌算法的其它问题

在确定新解产生的方案后, 下面本文将分析禁忌算法中的其它问题。

1. 候选解如何确定。候选解是下次局部搜索的中心节点产生的选择对象, 如果设通过  $f$  变换产生了  $t$  个个体, 那么可以设定以这些个体中  $\sum_{i=1}^n p_i$  最大的  $m$  个个体做为候选解。
2. 如何确定禁忌长度。禁忌长度太长则增加了搜索的时间开销, 太短则会使得搜索重复徘徊在几个局部解之间。本文中由于搜索领域规模比较大。例如:  $f$  中  $d = z$ , 那么由于  $f$  变换进行了 2 次假设 0 节点数为  $x$ , 1 节点数为  $y$ 。其搜索的领域将是 假设  $x + y = 20$ ;  $z = 2$ , 其最多的领域节点将有 2025 个。所以本

文设定如果某个个体  $P_i$  在禁忌表中那么与它距离为  $\frac{d}{2}$ （其值向上取整）的所有个体都属于禁忌个体。这样不需要设定太多的禁忌对象就能控制比较大的禁忌范围。同时，判断是否属于禁忌个体仅仅需要对禁忌表中所有的个体求一个距离，如果其值都大于  $\frac{d}{2}$ ，那么他就不属于禁忌范围。距离的计算可以根据定义 1 来进行。因此，本文中的禁忌长度可以设定为 10 个左右。

3. 特赦准则的设置。特赦准则是算法避免遗失优良状态，激励对优良状态的局部搜索，进而实现全局优化的关键步骤。本文中特赦的对象是候选解中出现了当前最优解。即使该解是在禁忌表中某个个体的  $d$  禁忌覆盖范围中，仍然可以按照 4.2 节中的算法第 4 步来处理。
4. 终止准则。如果算法搜索次数大于阈值  $\lambda_1$  且在过去的连续  $\lambda_2$  次中无法获取“best so far”状态更新则算法停止。其中比较大的  $\lambda_1$  保证了搜索的充分性。比较小的  $\lambda_2$  限制了无效搜索的持续进行。

### 3.6 禁忌算法的实现

实现步骤：

1. 将贪婪算法的划分结果作为“best so far”解 CP 和当前解 NOP，置禁忌表为空。J=0, K=0。
2. 对 NOP 中的 0, 1 状态分别使用相同的  $f$  变换，产生  $t$  个新解。并从中确定  $m$  个最好的可行解作为候选解。转 3。
3. 选取候选解中的最优解  $P_i$ ，判断  $P_i$  是否比“best so far”解要好。若是，则用  $P_i$  替代 NOP 成为新的当前解，即  $\text{NOP} = P_i$ ，如果禁忌表已满，用  $P_i$  替换最早进入禁忌表的禁忌对象，如果还没满直接将  $P_i$  放入禁忌表。并用  $P_i$  代替 CP 成为“best so far”解。J=J+1。K=0。然后转步骤 2；否则，转 4。
4. 计算所有候选解同禁忌表中的解的距离并判断对应的禁忌属性，选择处于非禁忌状态的候选解中的最优解  $P_i$  作为当前解 NOP。如果禁忌表已满，用  $P_i$  替换最早进入禁忌表的禁忌对象，否则直接将  $P_i$  放入禁忌表。J=J+1, K=K+1。
5. 如果  $J > \lambda_1$  且  $K = \lambda_2$ 。算法终止。否则转 2。

## 4 实验结果

为了验证本文提出的算法的有效性，本文将该算法同模拟退火算法进行了比较，其中它们都是以超过可容忍度  $abide$  的贪婪算法结果作为初始搜索个体。设定问题的规模是 20 个函数， $f$  变换中  $d=4$ ， $e = \frac{1}{3}$ ，产生的个体数为 100。高速缓存记录的函数次数  $m=50$ ，总的可重够面积  $A=100$ ，各函数的软件执行时间，硬件执行时间，硬件重构时间，所需的面积，下周期预计执行次数由随机产生。其中软件执行时间的取值为 50 到 100 的随机整数，硬件执行时间为 1 到 50 的随机整数，硬件重构时间为 100 到 200 的随机整数，所需的面积为 10 到 30 的随机整数。设定前  $x$  个函数为已配置到硬件执行的函数，其中这些函数的总面

积不超过 100。下周期预计执行次数每个函数初始化为 1，随机生成  $y$  个 2 到 10 的随机数，并将其随机赋值给各函数且所有函数对应的下周期执行次数之和小于  $72 - y$ 。 $\lambda_1 = 20$ ， $\lambda_2 = 5$ 。

表 4.1 执行 1000 次的运行结果

	平均执行时间	目标函数优化度
禁忌算法	0.012	9.2%
模拟退火算法	0.125	8.8%

从上表可以明显发现禁忌算法的执行时间比模拟退火算法快了 10 倍，并且其能更好地优化贪婪算法的结果。

## 5 结论

面向可重构片上系统的软硬件划分从一开始就受到研究人员关注，动态软硬件划分算法的难点在于其对实时性要求比较高。本文基于函数级的可重构动态软硬件划分的基本模型提出了一种基于贪婪算法和禁忌搜索算法综合决策的动态软硬件划分算法。最后实验表明，禁忌搜索算法能够比较好的配合贪婪算法实现在 *abide* 容忍范围之外的优化。软硬件划分问题通常需要综合考虑多方面的因素，包括能耗，面积，成本各因素，因此，在进一步的工作中，将综合考虑这些因素，来实现适应动态软硬件划分要求的划分算法。

## 6 致谢

非常感谢我的老师李仁发教授和曾庆光教授的细心指导，同时要感谢湖南大学嵌入式系统及网络实验室的同学在生活和学习过程中对我的帮助。

## 参考文献

- [1] H Singh, M-H Lee, G Lu, et al. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications[J]. IEEE Transactions on Computers, 2000, 49(5): 465-481.
- [2] Chatha, Vemuri. Hardware-software codesign for dynamically reconfigurable architectures[M]. Proc. of Intl. workshop on field programmable logic and applications, 1999.
- [3] Greg Stitt, Roman Lysecky, Frank Vahid. Dynamic Hardware/Software Partitioning: A First Approach[M]. Proceedings of DAC, 2003.
- [4] Hayden Kwok-Hay So, Artem Tkachenko, Robert Brodersen. A Unified Hardware/Software Runtime Environment for FPGA-Based Reconfigurable Computers using BORPH[M]. Proc. of CODES+ISSS, 2006.
- [5] Ann Gordon-Ross, Frank Vahid. Frequent Loop Detection Using Efficient Nonintrusive On-Chip Hardware[J]. IEEE Trans. on Computers, 2005,54(10): 1203-1215.
- [6] 吴强.面向系统芯片的软硬件协同设计方法研究 [D].北京:清华大学,2004.
- [7] F. Glover, Tabu search, Part I [M], ORSA Journal of Computing , 1989.
- [8] F. Glover, Tabu search, Part II [M], ORSA Journal of Computing ,1990 .
- [9] 赵敏媛,吕钊,顾君忠.嵌入式系统的软硬件划分[J].微计算机应用.2003,26(3):265-268.

## A dynamic hardware-software partitioning algorithm based on reconfigurable system

Zhou liqiu,Li renfa,Zeng qingguang

Hunan University

### Abstract

Hardware-software partitioning for reconfigure system on chip is a hotspot for researchers,and the

realtime demand about the dynamic hardware-software partitioning make it difficult to find a fit algorithm. This paper will analyse the question about a dynamic hardware-software partitioning based on function, and then an approach about greedy united tabu search algorithm will be made to resolve the realtime demand question.

**Key words:** Dynamic reconfigurable; Hardware-software partitioning; Greedy algorithm; Tabu search algorithm

作者简介：周立秋，湖南大学研究生，主要研究方向是可重构计算。

李仁发，教授，博士生导师，主要从事嵌入式系统和无线网络的研究。

曾庆光，教授，硕士，主要研究方向有智能计算、网络复杂行和服务计算。