动态部分可重构系统空闲资源全集管理研究

柴亚辉[1,2] 张胜辉[2] 黄卫春[2] 刘觉夫[2] 徐炜民[1]
(上海大学计算机工程与科学学院 上海 200072)[1] (华东交通大学信息工程学院 南昌 330013)[2]

摘 要 可重构系统兼具了传统处理器的灵活性和接近于ASIC的计算速度,FPGA的动态部分重构能够实现计算和重构操作的同时进行,使系统能够动态地改变任务的运行。在动态部分可重构系统中,高效的空闲资源管理策略对系统整体性起着非常重要的作用。提出了一种基于单向栈的算法来寻找最大空闲矩形(MFR)。利用可重构计算单元的不同M值进出单向栈来找到所有最大空闲矩形。通过实验表明,算法通过使用单向栈与算法优化,有效地提高了查找空闲资源全集的性能。

关键词 动态部分可重构,可配置的逻辑门阵列,最大空闲矩形,单向栈
中图法分类号 TP302,TP316 **文献标识码** A

Completed Free Resource Management Research on Dynamic Partial Reconfigurable System

CHAI Ya-hui[1,2] ZHANG Sheng-hui[2] HUANG Wei-chun[2] LIU Jue-fu[2] XU Wei-min[1]
(College of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)[1]
(College of Information Engineering, East China Jiaotong University, Nanchang 330013, China)[2]

Abstract Reconfigurable computing system has the flexibility of traditional processor and the speed of ASIC approximately. Dynamic partial reconfigurable system realizes the computing and reconfiguration at the same time, in which an efficient free resource management scheme is very important to achieve high performance. This paper introduced an efficient algorithm to find a series of maximal free rectangles (MFR) based on one-way stack. The algorithm uses different M value in and out of one-way stack to find all maximal free rectangles. We used simulation experiments to simulate the algorithm, and the results show that the this algorithm improves the performance of searching complete free resources.

Keywords Dynamic partial reconfiguration, Field programmable gate array, Maximal free rectangles, One-way stack

近年来,FPGA(Field Programmable Gate Array)作为可重构硬件技术的代表,既具有通用处理器的灵活性,又具备硬件的速度优势,一直备受业界关注[1,3]。其重构方式有以下2种分类:一种分类为完全可重构和部分可重构;另一种分类为静态可重构与动态可重构。由于动态部分可重构硬件在运行时改变芯片配置不影响芯片上原有任务的执行,因此与传统可重构硬件相比,其配置更灵活,芯片利用率更高[2]。合理、高效地寻找空闲资源全集策略对系统整体性能起着非常重要的作用[4]。

1 相关研究

自20世纪80年代以来,许多人已经提出了一些有效的空间资源的管理方法。Bazargan[5]首先提出了一种互不交迭的空闲矩形可重构资源管理方法。文献[6]采用二维Hash函数对Bazargan所提方法进行了改进,但由于这种方式大多数情况下不能标识出可重构资源中的最大空闲区域,因此任务放置的成功率受到较大影响。通过顶点链表来管理FPGA上的空闲资源是由Tabero[7]提出来的,但是任务的动态添加和删除使得顶点链表的管理和查找比较复杂。Handa[8]使用寻找最大空闲矩形的完全集来实现空闲资源的管理,但其使用的楼梯算法比较复杂。SLA(Scanning Line Algorithm)[9]利用叫做"Valley Point"的算法来搜索所有的最大空闲矩形,其搜索所有的最大空闲矩形MFR(Maximal Free Rectangle)的算法的复杂度比较高。在文献[10]中通过管理被任务占用的区域来间接地管理空闲区域,这种方法由于不能很好地评价任务在FPGA上的布局好坏,因此不利于资源的高效使用。

2 SCA

使用最大空闲矩形(MFR)来管理可重构资源是非常有效的,其通过对任务的添加和删除,动态地更新MFR能够提高任务分配空闲资源的成功率。提出的基于扫描列算法

到稿日期:2012-04-18 返修日期:2012-09-21 本文受国家高技术研究发展计划(863计划)重点项目(2009AA012201),江西省自然科学基金项目(20114BAB201028),华东交通大学校立科研课题(11XX04),上海市重点学科建设项目(J50103)和江西省自然科学基金(2010GZS0031)资助。

柴亚辉(1977—),男,博士生,讲师,主要研究方向为高性能计算、可重构计算,E-mail:shiba@shu.edu.cn;张胜辉(1986—),男,硕士,主要研究方向为可重构计算、信息安全;黄卫春(1968—),男,硕士,副教授,主要研究方向为数据挖掘、信息处理;刘觉夫(1963—),男,硕士,教授,硕士生导师,主要研究方向为可重构计算、云计算;徐炜民(1951—),男,硕士,教授,博士生导师,主要研究方向为高性能计算、可重构计算。

(SCA)就是根据 FPGA 上扫描列与扫描起始点来管理所有 MFR,以管理空闲的可重构资源。

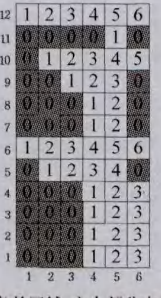
2.1 基本的定义

FPGA 是由可重构计算单元 RCU(Reconfigurable Computing Uint)组成,假设一个可重构器件由 $N=W\times H$ 个 RCU(器件中的每个 RCU 都相同)组成,就形成了一个 W 列 H 行的二维矩形阵列。每个 RCU 具有一个坐标位置(i,j),其中 $1\leqslant i\leqslant W$,$1\leqslant j\leqslant H$ 分别代表 RCU 所在的列和行。$S(w,h)$表示器件中的一个 w 列 h 行的矩形区域,其中 $1\leqslant w\leqslant W$,$1\leqslant h\leqslant H$。S 在器件中的位置由其左下角 RCU 的坐标和 w、h 表示:(x,y,w,h)。左下角位置(x,y)称为 S 的基点[11]。

我们用二维矩阵 $M[W+1][H]$标识 FPGA 上的空间区域,RCU 的 M 值定义如下:

$$M[i][k]=\begin{cases}M[i-1][k]+1, & \text{未被占的 RCU 的 } M \text{ 值}\\ 0, & \text{被占的或 } i=0 \text{ 或 } i=W+1 \text{ 时 RCU 的 } M \text{ 值}\end{cases}$$

图 1 显示了 RCU 中的 M 值。若 RCU 未被放置任务,则 M 的值代表其左侧空 RCU 的数量;若 RCU 已经被放置任务,则 M 值为 0。



阴影部分表示放置任务的区域,空白部分表示未放置任务的区域

图 1 FPGA 标识图

定义 1(最大空闲矩形 MFR) 不能被其他任何一个空闲矩形所完全覆盖的空闲矩形为最大空闲矩形(MFR)。对 MFR 的表示有多种方式,如:矩形的 4 个顶点坐标、矩形放置的左下角基点与矩形的长宽等。本文使用(x,y,w,h)来表示一个 MFR,(x,y)是其左下角 RCU 的坐标,(w,h)是其长与宽。

确定一个 MFR 时要确定其 4 条边不能向外"扩展",如图 1 中的 MFR(4,1,3,4)的顶边 TE(Top Edge)被 RCU(6,5)挡住,所以 MFR(4,1,3,4)的 TE 就不能向上"扩展"。

设 i 为扫描列 SC 所在的列,则:

MFR 底边 BE(Bottom Edge):$M[i][j]>M[i][j-1]$,j 为底边 BE 所在的行;

MFR 顶边 TE(Top Edge):$M[i][t]>M[i][t+1]$,t 为顶边 TE 所在的行;

MFR 右边 RE(Right Edge):至少存在一个 w,$j\leqslant w\leqslant t$,$M[i+1][w]=0$,t 为顶边 TE 所在的行;

MFR 左边 LE(Left Edge):至少存在一个 k,$j\leqslant k\leqslant t$,$M[i][k]=\min(M[i][j],M[i][t])$,$t$ 为顶边 TE 所在的行;

以上 4 条边就可以确定一个 MFR$(i,j,M[i][k],t-j)$,其中(i,j)为 MFR 的左下角坐标,$h-j$ 为 MFR 的高,$M[i][k]$为 MFR 的宽。

通过以上对 4 条边的描述可以看出,确定一个 MFR 就转化为了确定其 4 条边的过程。

图 1 中,根据定义,共有 8 个 MFR。如:MFR(4,1,3,4),在列 $i=6$ 上,底边 BE,其 $M[1][6]=3>M[0][6]=0$;其顶边 TE 的 $M[4][6]=3>M[5][6]=0$;左边 LE,其 $\min(M[1][6],M[4][6])=3$;其右边 RE,存在一个 1,$M[1][6+1]=0$。其余的 7 个 MFR 分别为:MFR (2,5,4,2),MFR (4,1,2,10),MFR(3,9,3,2),MFR(5,1,1,12),MFR (1,6,6,1),MFR (2,10,5,1),MFR (1,12,6,1)。

定义 2(KVP,Key Value Point) 如果 FPGA 上的某 RCU(x,y)所在的行与列分别为(i,j),其 $M[i][j]$满足以下条件:

(1)$1<i\leqslant W$

(2)$M[i][j]\neq 0$ 并且 $M[i+1][j]=0$

则该 RCU(x,y)为 KVP。如图 1 所示,RCU(6,1)、RCU(5,5)、RCU(11,5)为 KVP。

定义 3(SC,Scan Column) KVP 所在的列为 SC,如果该 KVP 所在的列为 i,则扫描列 SC 就为列 i。如图 1 所示,列 6 为 SC。

Key Value Point 就是右邻 RCU 被占用而本 RCU 未被占用的空 RCU 或者是此 RCU 的右侧为 FPGA 区域的右边界。由于一条扫描列包含一个或多个 KVP,因此,此扫描列的右侧有一个或多个被占用的 RCU。我们只需要在扫描列 SC 的左侧查找 MFR,也就是说 MFR 的右边界落在 SC 上。

定义 4(SSP,Scan Starting Point) FPGA 上满足以下条件的 RCU (i,j):

(1)列 i 为 SC

(2)$M[i][j]\neq 0$ 且 $M[i][j-1]=0$

如图 1 所示,RCU(6,1)为 SSP。

总之,SSP(Scan Starting Point)就是位于 SC 上的 RCU,其下面的 RCU 被占用,而 SSP 未被占用。

2.2 算法

根据前面的定义,通过 KVP(6,1)可以确定 MFR(4,1,3,4)的右边 RE 为列 6,而通过 SSP 可以确定 MFR(4,1,3,4)的底边 BE 为行 1,通过向上扫描 SC 的各行去寻找最小但不为 0 的 RCU,则可以确定其宽度为 3,继续向上扫描遇到 M 值为 0 的 RCU 时可以确定其高度为 4。最后可以确定一个 MFR(4,1,3,4)。

提出了一些算法并证明了其正确性和完整性。首先以基本算法开始(见 2.2.1 节),在此基础上提出了入栈优化算法(见 2.2.2 节)和 M 标示优化算法(见 2.2.3 节)。

2.2.1 基本算法

算法 1 基于栈和 SSP 寻找 MFR 的基本算法。

从一个 SSP(i,j)出发,用一个栈 Stack 来存储 SC 上 RCU 的相关数据。

Stack 里的一个结点保存以下信息:

(1)$M[i][k]$:RCU 的 M 值;

(2)flag 标识位:0 表示该 RCU 为 KVP,1 表示为非 KVP,2 表示为非 KVP 但与刚出栈的 RCU 具有相同的 M 值;

(3)K:RCU 所在的行。

对栈的操作主要包括:入栈与出栈。设要入栈的 SC 为 i $(1\leqslant i\leqslant W)$。

首先把 SSP 所在 RCU 的 M 值入栈,若此 SSP 所在的

RCU为KVP，则SSP的flag标为0，否则标为1。继续向上扫描，指向上一行的RCU。

入栈操作：

入栈操作的条件：RCU的M值大于或等于栈顶的M值。

如果RCU的M值大于栈顶结点的M值，则该RCU的M值入栈，如果该RCU为KVP，则标识其flag为0；如果非KVP，则标识flag为1；

如果RCU的M值等于栈顶结点的M值，首先进行出栈操作，该RCU的M值入栈，如果该RCU为KVP，则标识其flag为0；如果非KVP，并且栈顶的RCU的flag值为0或2，则标识flag为2。

继续向上扫描，指向$K+1$行的RCU。

出栈操作：

出栈操作的条件：RCU的M值小于或等于栈顶的M值。

MFR的产生是在出栈操作的过程中，产生MFR的条件为：RCU的M值小于栈顶的M值。

如果RCU的M值等于栈顶结点的M值，将栈顶的结点出栈；

如果RCU的M值小于栈顶结点的M值，Stack顶部结点出栈，并产生MFR，然后比较新的栈顶。

MFR的生成过程为：

首先确定MFR的左下角坐标：x的值为$i-M+1$，y的值为新栈顶的K值加1。

确定宽度与高度：W的值为出栈结点的M值，高度H为RCU的K-新栈顶的K值减1。

```
Input:FPGA area marix M and SSP at (i,j).
Output:Set of MFRs generated by the SSP at (i,j).
Init Stack(h)
DO
  Get RCU(p)
  If M[i][p]==M[i][h] /* M[i][h]为要被放入栈的RCU的M值,M[i][h]为栈顶结点M值*/
    Push();
    Pop();
  If M[i][p]>M[i][h]
    Push();
  F:If M[i][p]<M[i][h]
    Pop()
    GenMFR();
    Goto F
  While M[i][p]! =0
```

例如，图2显示了从SSP(5,0)寻找MFR的过程，只是简单地说明寻找过程而省略了中间出入栈的过程。

图2 基本算法下从SSP点(5,0)寻找MFR

第1步 $w=4$ $n=2$ record MFR(2,5,4,2)

第2步 $w=3$ $n=2$ record MFR(3,9,3,2)

第3步 $w=2$ $n=10$ record MFR(4,5,10,10)

第4步 $w=1$ $n=12$ record MFR(5,1,4,2)

定理1 任意一个MFR的右边界必定位于某条SC上，并且任意一条SC上至少存在一个右侧边界落在其上的MFR。

证明：如果存在一个MFR，其右边界落在列i上(但i不是SC)，根据SC的定义可知，在列i上就不存在KVP，则列i就不会是FPGA的最右侧列且列$i+1$就不会有被任务所占用的RCU。此时MFR可以向右扩展到列$i+1$，这就表明MFR不是最大的，这与MFR的定义相矛盾。所以，假设不成立，故定理1正确。

定理2 通过扫描SC_i上的所有RCU的M值可以找到所有的右侧边界落在列i上的MFR。

证明：如果存在一个右侧边界落在列i上的MFR(M_1)，它不能通过扫描SC_i上的所有M值得到。假设M_1的宽度为w，高度为h，它的右边界的两个顶点分别为(i,j_1)、(i,j_2)(见图3(a))。此时一定会存在$M[i][j_1]>M[i][j_1-1]$且$M[i][j_2]>M[i][j_2+1]$，因为如果$M[i][j_2]\leqslant M[i][j_2+1]$，必然$M_1$还可以向上扩展(见图3(b))，则原来的$M_1$就与MFR的定义相矛盾，而$M[i][j_1]>M[i][j_1-1]$且$M[i][j_2]>M[i][j_2+1]$同时成立是出栈时形成MFR的条件。所以，假设不成立，而定理2正确。

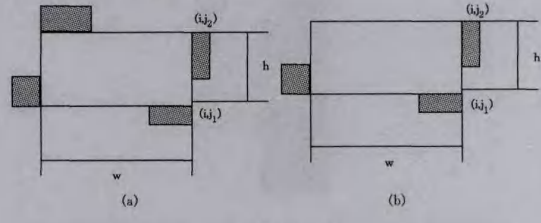


图3

2.2.2 入栈优化算法

算法2 基于栈和SSP(i,j)来寻找MFR的入栈优化算法。

入栈优化算法是对基本算法的改进，标识SC上RCU的M值时，基本算法只用“M值”来标识RCU，而入栈优化算法用“M值+连续相同M值计数器C”来标识RCU。如图4所示，RCU(6,1)的M值为3，自此RCU向上与其有连续相同M值的RCU个数有4个(即$C(6,1)$为4)。

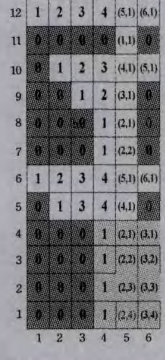


图4 改进算法下从SSP点(6,1)寻找MFR

入栈时，如果RCU的M值大于栈顶结点的M值，那么该RCU的M值入栈，如果该RCU为KVP，则标识其flag为0；如果非KVP，则标识flag为1；继续向上扫描，指向第$K+$

C行的RCU。

```
Input:FPGA area marix M and SSP at (i,j).
Output:Set of MFRs generated by the SSP at (i,j).
Init Stack(h)
DO
  Get RCU(p);
  If M[i][p]>M[i][h] /*(M[i][p],C(i,p))为要被放入栈的RCU
    的M值,(M[i][h],C(i,h))为栈顶结点M值*/
    Push();
    p=p+C(i,p);
While M[i][p]! =0
```

如图4所示,SSP为RCU(6,1)时,如果使用第一改进算法来寻找MFR,RCU(6,1)的值(3,4)入栈,由于$C(6,1)$为4,下一个要入栈RCU的K值为$K(6,1)+C(6,1)=1+4=5$,继续向上扫描RCU(6,5),其M值为0,RCU(6,1)结点出栈,形成MFR(4,1,3,4)。

改进算法减少了重复的出入栈的操作,提高了算法的执行效率。

2.2.3 M标示优化算法

算法3 基于栈和扫描起始点(i,j)来找MFR的M标示优化算法。

M标示优化算法只标识FPGA区域的部分RCU。已放置任务的FPGA区域的标识方法:放置任务的4周RCU标0,任务中间RCU不标号;未放置任务的FPGA区域的标识方法:仅标扫描列RCU的值,其余未放置任务的FPGA区域不标号。这样标号的好处是:

(1)省去了标号处理时间

对任务$T(x,y,w,h)$标号的时间节省率为:

$$1-\frac{2(w+h)-4}{w\times h}$$

(2)在任务查找时只是查询已标号的RCU而不必查询所有的RCU。

本算法在频繁调度大的任务时,能大大提高性能。

如图5所示,D、F是两个正在运行的任务,R是新调度进来的任务。

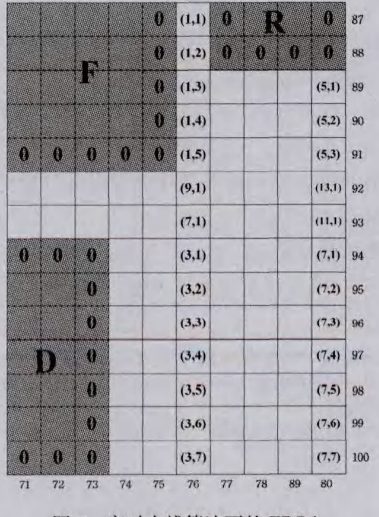


图5 实时在线算法下的FPGA

在FPGA上实时地增加或删除任务时,只有部分区域受影响,此时只是扫描和更新受影响区域的MFR。

3 性能仿真

使用实验来比较SCA调度算法和SLA调度算法的性能,实验只是关注在实时调度中的空闲资源全集管理,而不考虑实时调度算法中的任务队列、调度的优先级等问题。实验目的是在部分可重构的FPGA中找到所有的MFR,假定使用FIFO(先进先出)的任务调度算法,如果一个将要被调度的任务T不能被放入到FPGA中,队列中的其他任务必须等待,直到任务T能够被放入到FPGA中为止(随着其它的一些任务的运行完成使FPGA中有MFR来放置任务T);使用First-Fit的策略来选择一个能满足放置任务要求的MFR,在任务放置时,把任务放在所选MFR的左下角;假定FPGA的大小是100×80,在每次进行实验时有10000个任务,每个任务的宽度和高度分别在2到8个RCU之间,并且每个任务的执行时间在2到10个时间单位之间。在任务放入FPGA或从其删除的时候都要重新计算MFR全集;由于要确保在每次进行模拟时,有20000次调用MFR更新算法(10000个任务的放入和10000个任务的删除),因此同一时刻有多个任务同时放入FPGA或从其删除的时候都要更新一次所有的MFR。本实验运行在一个拥有3.0GHz CPU和512M内存的Linnux工作站上。每个任务的到达时间是一个在$(0,U)$区间上的随机值。使用不同的U来控制FPGA区域的利用率和工作负荷。U的值越小,代表任务的到达更频繁和FPGA区域的利用率更高。

图6的实验结果表示算法运行时间的概率分布图。例如,图6中点(35,0.14)的物理解释是:共有2800(0.14×20000=2800)次调用了改进的SLA算法,t表示算法的运行时间。从图6中能够看出,优化算法下的SCA负载越重,其性能越好。

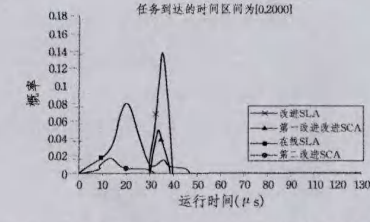


图6 不同算法更新MFR所需要的运行时间

假设标识1个RCU所用时间为单位1。表1和图7分别示出在不同的FPGA区域上分别使用SLA算法和SCA算法放置相同的任务时SCA算法的性能提高率和总性能提高率。

表1 性能提高表

FPGA的M与N	主要任务的W与H	算法优化部分性能
100×80	3×2—24×32	51%
500×400	6×3—80×40	80%
1000×800	18×9—100×80	83%
2000×1600	25×8—300×220	85%

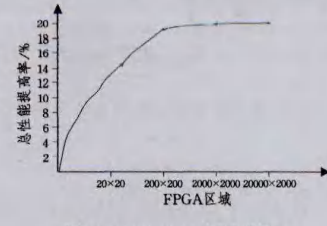


图7 SCA总性能提高率

(下转第34页)

图 7 中标实线的时间开销曲线是路网权值变化较少导致的动态路由表的重置间隔较长的测试结果。该组测试中，路由表的重置时刻 t_0 为第 11 次测试的起始时刻，协作更新的起始时刻 t_1 为第 9 次测试的起始时刻，即阈值 $\Delta t = t_0 - t_1$ 为 2 次测试的时间间隔。

图 7 中标虚线的时间开销曲线是路网权值变化较频繁而导致动态路由表的充值间隔较短的测试结果。该组测试中，路由表的重置时刻 t_0 分别为第 4、7、10、13、16、19 次测试的起始时刻，协作更新的起始时刻为每次重置时刻的前一次测试起始时刻，即阈值 $\Delta t = 1$ 次测试的时间间隔。

结束语 基于路由机制的 k 近邻算法在实际应用时需要一定的前提条件，即在动态路网道路权值的变化不太频繁和剧烈的情况下，基于路由机制的 k 近邻算法既能通过实时计算保证结果的精确性，又能避免大量重复计算，显著降低计算开销。在路网权值变化较少，使得动态路由表的重置间隔较长时，随着动态路由表 T_D 内历史数据的逐渐丰富，计算时的时间开销显著降低；而当路网权值频繁变化而导致动态路由表的频繁重置时，因动态路由表始终无法存储足够多的历史数据而导致在候选集的筛选等步骤上的时间复杂度增加，并且无法大量地减少重复计算从而抵消或者降低路由机制中更新策略、导航策略等带来的额外的时间开销，导致性能降低。

参 考 文 献

[1] 孟小峰，丁治明. 移动数据管理概念与技术[M]. 北京：清华大学出版社，2009

[2] Goodsell G. On finding p-th nearest neighbors of scattered points in two dimensions for small p[J]. Computer Aided Geometric Design，2000，17：387-392

[3] Jensen C S，Pedersen K T B，et al. Nearest neighbor queries in road networks[C]// Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems. 2003：1-8

[4] Roussopoulos N，Kelley S，Vincent F. Nearest neighbor queries [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. 1995：71-79

[5] Bespamyatnikh S，Snoeyink J. Queries with segments in Voronoi diagrams[C]// SODA '99 Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. 1999：122-129

[6] 郝忠孝. 时空数据库查询与推理[M]. 北京：科学出版社，2010

[7] 赵亮，陈荦，景宁，等. 道路网中的移动对象连续 K 近邻查询[J]. 计算机学报，2010，33(8)：1396-1404

[8] 陈子军，任彩平，刘文远. 路网中查询点速度不确定的连续 k 近邻查询方法[J]. 小型微型计算机系统，2011，32(3)：430-434

[9] 卢秉亮，刘娜. 路网中移动对象快照 K 近邻查询处理[J]. 计算机应用，2011，31(11)：3078-3083

[10] Demiryurek U，Banaei-Kashani F，Shahabi C. Efficient K-Nearest Neighbor Search in Time-Dependent Spatial Networks[C]// DEXA 2010，Part I，LNCS 6261. 2010：432-449

[11] 唐俊，张栋良. 基于路由机制的变权网络路径快速生成算法[J]. 计算机科学，2011，38(12)：110-124

(上接第 23 页)

结束语 为了在 FPGA 硬件空间上管理空闲资源全集 MFR，本文将栈技术引入到对 MFR 的查找与生成中，提出了一种利用单向栈的算法。本算法是 FPGA 上硬件任务实时在线放置与调度的基础，只有有效地管理好空闲资源全集，才能有效地应用任务调度算法。下一步我们将该基于单向栈的 MFR 全集管理算法应用于 CPU/FPGA 体系结构的集群系统上任务调度算法的研究。

参 考 文 献

[1] 余国良，伍卫国，杨志华，等. 一种采用边界表进行可重构资源管理及硬件任务调度的算法[J]. 计算机研究与发展，2011，48(4)：699-708

[2] 龚育昌，齐骥，胡楠，等. 部分可重构系统布局的一种新算法[J]. 中国科学技术大学学报，2007，37(9)：1047-1053

[3] 黄勋章，周学功，彭澄廉. 可重构系统中高效的二维任务放置策略[J]. 计算机工程与设计，2008，29(7)：1745-1749

[4] 焦铭，李仁发，李浪，等. 可重构系统中基于空间邻接度的实时任务放置算法[J]. 计算机应用研究，2011，28(4)：1290-1295

[5] Bazargan K，Kastner R，Sarrafzadeh M. Fast Template Placement For Reconfigurable Computing Systems[J]. IEEE Design and Test of Computers，2000，17(1)：68-83

[6] Walder H，Steiger C，Platzner M. Fast Online Task Placement on FPGAs：Free Space Partitioning and 2D hashing[C]// Proceedings of the 17th International Parallel and Distributed Processing Symposium. Washington D C，USA：IEEE Computer Society，2003：178

[7] Tabero J，Steptien J，Mecha H，et al. A Vertex-list Approach to 2D HW Multitasking Management in RTR FPGAs[C]// Proc. of DCIS. Cludad Real，Spain：[s. n.]，2003：545-550

[8] Handa M，Vemuri R. An Efficient Algorithm for Finding Empty Space for Online FPGA Placement[C]// Proceedings of the 41st Design Automation Conference. NY，USA：[s. n.]，2004

[9] Cui Jin，Deng Qing-xu，He Xiu-qiang，et al. An efficiental agorithm for online management of 2D area of partially reconfigurable FPGAs[C]// 2007 Design，Automati on and Test in Europe Con ference and Exposition. Nice：DAA EDAC IEEE Computer Society TTTC，2007：129-134

[10] Ahmadinia A，Bobda C，Urgen T J. A New Approach for On-line Placement on Reconfigurable Devices[C]// Proceedings of the International Parallel and Distributed Processing Symposium，Reconfigurable Architectures Workshop. Santa FNM，USA：IEEE-CS Press，2004-04

[11] 周学功，梁梁，黄勋章，等. 可重构系统中的实时任务在线调度与放置算法[J]. 计算机学报，2011，48(4)：1901-1909