基于硬件任务顶点的可重构系统资源管理算法

齐 骥, 李 曦, 胡 楠, 周学海, 龚育昌, 王 峰
(中国科学技术大学计算机系, 安徽合肥 230027)

摘 要: 可重构资源的管理是影响动态可重构系统性能的关键因素之一. 本文提出了一种保持任务顶点信息的布局算法 KTVP, 并将其扩展为 KTVS 调度算法. KTVP/KTVS 算法根据已接收硬件任务的顶点信息安排新到达的硬件任务, 可通过对系统资源的编码迅速验证布局/调度的可行性, 能有效减少系统资源浪费, 提高系统内在的并行度. 仿真实验表明, 与已有算法相比, KTVP/KTVS 算法具有更低的任务拒绝率和更小的运行开销.

关键词: 动态可重构; 布局; 调度; 任务顶点
中图分类号: TP302 文献标识码: A 文章编号: 0372-2112 (2006) 11-2094-05

Algorithms of Resource Management for Reconfigurable Systems Based on Hardware Task Vertexes

QI Ji, LI Xi, HU Nan, ZHOU Xue-hai, GONG Yu-chang, WANG Feng
(*Deptament of Computer Science, University of Science and Technology of China, Hefei, Anhui 230027, China*)

Abstract: The management of reconfigurable resource is one of the most critical factors concerned deeply with the performance of dynamic reconfiguable systems. This paper presents two associated algorithms, KTVP (Keeping Tasks Vertexes for Placement) and KTVS (Keeping Tasks Vertexes for Scheduling), to maximize the system utilization. These two algorithms arrange the new-coming hardware tasks according to the vertexes' information of the already running tasks. By exploiting a novel encoding scheme, they can validate the feasibility of the scheduling/placement in a fast way, so as to effectively reduce the waste of resource and improve the parallelism of the whole system. The simulation results show that, compared with existent algorithms, KTVP/KTVS can achieve lower task rejection ratio and reduce the execution overhead significantly.

Key words: dynamic reconfiguration; placement; scheduling; task vertexes

1 引言

当前主流的可重构硬件是基于 SRAM 存储器的 FPGA (Field Programmable Gate Array), 其配置信息存储在 SRAM 中, 具有可任意次配置、容量大、配置速度快等优点. 由于可重构硬件在性能和灵活性方面的优势, 现已被广泛应用于嵌入式和实时系统. 部分可重构硬件既可在运行时动态改变芯片某一部分的配置, 又允许其未更改部分仍按原有方式正常工作, 与传统可重构硬件相比, 配置更灵活, 芯片利用率更高. 图 1 为部分可重构芯片和 CPU 构成的可重构混成平台. 布局/调度问题是指运行时可重构资源的动态分配和管理, 调度器的作用是为硬件任务分配计算资源和决定各硬件任务的触发时机, 布局器负责具体管理芯片上的空闲资源.

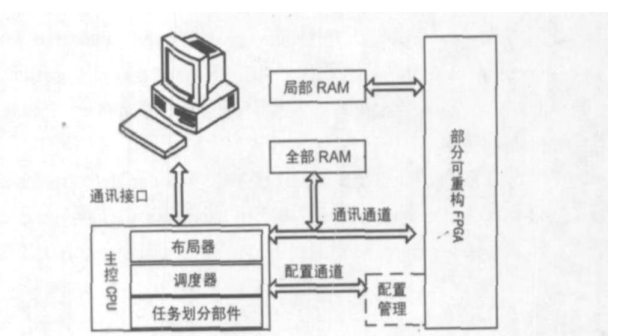


图 1 可重构计算平台结构

Barzargan 最早对上述布局问题进行了系统的研究[1], 他提出的 KMER 布局算法布局质量较高, 但开销很大, 时间/空间复杂度都至少为 $O(N^2)$. 文献[2]采用新的方法生成最大空闲矩形, 虽然平均性能较 KMER 算法有一定提升, 但时间复杂度却是 $O(N^2)+O(W\times H)$. 此外, KMER 算法和文献[2]中的算法都存在抖动现象, 即在某些情况下加入或删除特定的硬件任务会导致大量空闲矩形的分割合并和大范围的扫描操作, 以至布局器成为整个可重构计算系统的瓶颈. 在已有布局算法中[1~5], 文献[2]的算法执行速度最快, 布局质量最高, 下文将其称为 MH(Manish H.)算法.

收稿日期: 2005-10-21; 修回日期: 2006-05-30
基金项目: 国家自然科学基金(No. 60273042); 中国科学院创新基金; 安徽省自然科学基金(No. 03042203)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

针对调度问题,文献[6]提出的算法按照缓冲队列中硬件任务的优先级进行调度,但该方案芯片利用率较低. 文献[7]提出了两种基于保证的实时硬件任务调度算法,但不支持非实时调度. 文献[8]提出了多上下文 FPGA 中的周期任务调度算法,但算法要求的上下文数目过多,无法实用. 在已有调度算法中,由于文献[7]的 Stuffing 算法对未来资源的使用情况有所规划,故芯片利用率最高,调度效果最好.

布局/调度延时在动态可重构系统中是不可避免的,但在设计资源管理算法时应考虑尽量减少延时. 针对已有算法的局限性,本文提出保存任务顶点信息的布局算法 KTVP(Keeping Task Vertexes for Placement)和调度算法 KTVS(Keeping Task Vertexes for Scheduling). 这两种算法根据已接收硬件任务的顶点信息安排新到达的硬件任务,所有新到达任务的布局/调度都基于已接收任务的顶点而发展,可在保证布局/调度质量的基础上,大大降低动态资源管理产生的延时.

2　基本概念

可重构逻辑单元 RLU(Reconfigurable Logic Unit)是配置部分可重构芯片的最小单位. W、H 分别是可重构芯片单行、单列具备 RLU 的个数. N 代表当前可重构芯片上硬件任务的数目.

定义 1　硬件任务指经过内部综合,具有固定长度和宽度,一定时钟频率界限的矩形逻辑功能模块,可由 4 维向量 $\boldsymbol{T}(h, w, a, e)$ 描述. 其中,h、w 分别代表硬件任务的长和宽,a 是任务的到达时间,e 是执行时间.

二进制背包策略特指对候选布局位置进行选择的策略. 常见的策略有 Best Fit(BF)、First Fit(FF)等.

定义 2　碎片指可重构芯片表面无法被利用的计算资源.

有时硬件任务的面积小于芯片上空闲面积的总和却无法被接收,这就是受碎片影响的结果. 碎片是不可能被能完全消除的,优秀的布局器应尽量减少碎片.

定义 3　资源模型是指可重构芯片表面计算资源的抽象和运行时对硬件任务所占资源的约束. 目前,部分可重构芯片有两种资源模型:(1)一维模型. 硬件任务可被布局在水平方向上的任意位置,但在竖直方向上,每列不能同时存在一个以上的硬件任务;(2)二维模型. 硬件任务的布局在水平和竖直方向上都没有限制,只要其所占区域不相重叠. 本文的布局算法基于二维资源模型.

定义 4　芯片利用率 CU(Chip Utilization)指在一段时间内,芯片被占用的计算资源面积与芯片总面积比例的平均值.

定义 5　任务拒绝率 TRR(Task Rejection Ratio)指芯片拒绝的任务与到达芯片任务总数之比.

芯片利用率和任务拒绝率是衡量动态可重构系统资源管理算法布局/调度质量的两个重要参数.

3　KTVP 布局算法

本文提出一种保持任务顶点的布局算法 KTVP. 在线布局必须解决两个主要问题:(1)确定潜在的候选布局位置;(2)在候选位置中,选择最优的. 解决问题(2)需要定义代价函数,代价函数值最低即视为最优. KTVP 算法在 $O(N)$ 时间内解决问题(1)和(2).

3.1　数据结构

第一,对具有 H 行 W 列 RLU 的可重构芯片,建立二维权值矩阵 $\boldsymbol{M}[H][W]$. 矩阵中每个元素的权值 p 为正整数 k 或 0. $p = k$ 代表该单元没有被硬件任务占用,且此单元上方有 k 个连续的空闲单元;$p = 0$ 代表该单元已被占用. 例如,图 2 是一个权值矩阵,当前有两个正在运行的任务 T_1 和 T_2.

第二,建立顶点链表 C 保存当前运行中任务的顶点信息,该链表初始状态包含可重构芯片的四个顶点. 若顶点 v 的坐标为 (x, y),则 v 在链表中按照 $(x^2 + y^2)$ 值增加的顺序排列. C 中保存顶点的个数为 $O(N)$ 量级. 例如,图 3 显示了图 2 中权值矩阵对应的链表 C.

3.2　算法描述

定义 6　放置硬件任务 T 时,若它的四个顶点之一与 C 中保存的某点 $v(x, y)$ 相重合,则称 T 与 v 相匹配. 若此时 T 占用的资源范围不与当前运行的任何任务相重叠,则匹配成功;否则匹配失败.

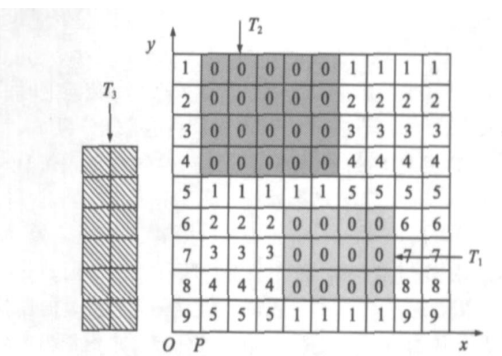


图 2　可重构芯片的编码

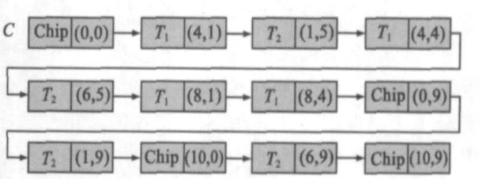


图 3　顶点队列

KTVP 布局算法的伪代码如 KTV Algorithm 所示. 当新任务 $\boldsymbol{T}(h, w, a, e)$ 到达可重构系统时,KTVP 算法尝试调用 TaskMatch 函数将 T 与 C 中的各顶点相匹配(第 2 行),若匹配成功则根据二进制背包策略(BF/FF)返回该顶点(第 5, 6 行),或者继续匹配 C 中的后续顶点以得到代价函数 KTVCost 最低的顶点(第 8, 9, 10 行);若匹配失败,继续将 T 与 C 中剩余顶点相匹配. 这样,新任务 T 就沿着当前运行任务的顶点和边界发展,不但策略简单,而且减少了任务之间空隙造成的资源浪费.

KTV Algorithm

```
LocateTask(T){
1. For each vertex v in C{
2.     succeed = TaskMatch(T, v);
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

```
3.      if ( succeed = = 1) {
4.        if( FF = 1){
5.          result_vertex = v;
6.          break;
7.        }else{
8.          if( ( cost= KTVCost( T, v) )< MINCOST);
9.          result_vertex = v;
10.         MINCOST= cost;}
11.       }
        }
12.     if ( result_vertex ! = NULL) return result_vertex;
13.     else return - 1;
14. }
```

其中匹配的过程由函数TaskMatch 完成, 算法如下:

Match Algr

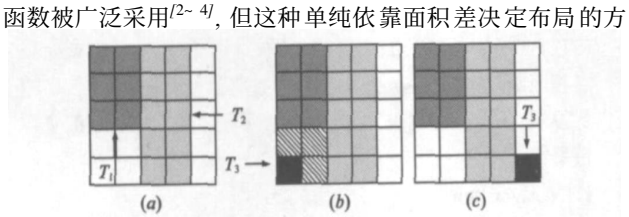
```
1. TaskMatch ( T, v) {
2.      if(weight( x, y) = = 0) return fail;
3.      else for( i = 0; i < w; i+ + )
4.          if( h > weight( x+ i, y) return fail;
5.      return 1;
6. }
```

Matchr Algr 说明了将任务 T 的左下顶点与 v 相匹配时的情形. 例如对于图 2 的情况, 若某时刻新到达的任务为 T_3, 首先尝试将 T_3 与点 O 匹配(第 3 行), 当 $i=0$ 时, weight(0, 0) = 9, 有 $h=6<$ weight(0, 0) (第 4 行), 算法继续. 当 $i=1$ 时, weight(0, 1) = 5, $h=6>$ weight(0, 1) (第 4 行), 匹配失败, TaskMatch 返回. 注意到, 任务划分部件将各复杂应用划分成大小基本固定的多个硬件任务, 随着可重构芯片容量的发展和任务划分技术的逐渐成熟, 单个硬件任务所占面积相对可重构芯片总面积而言甚小, 因此 h_{max}/w_{max} 可视为常量, 则 Matchr Algr 匹配算法可在常量时间内完成.

需要指出, Matchr Algr 只尝试将新任务 T 的单个顶点与 v 相匹配, 此种情况被称为 1v 情形, 类似的还有 2v、3v、4v 情形. 由于 4v 考虑最全面, 芯片利用率高于其他算法, 运行时间也更长. 可见, KTVP 布局算法实际上是一系列子算法的集合, 本文后续章节将用具体的名字指示此集合中的特定算法, 例如称采用 BF 策略, 且 4 顶点匹配的 KTVP 算法为 KTVP-BF-4v.

3.3 基于被占用区域连续度的代价函数

优秀的代价函数可有效控制碎片的产生. 文献[1]将空闲矩形的面积与新到达任务面积之差作为代价函数, 这种代价函数被广泛采用[2~4], 但这种单纯依靠面积差决定布局的方法可能产生潜在碎片(不规则区域). 例如, 对于图 4 的情况, 初始状态如图 4(a)所示, 加载 T_3 时, 若采用面积差代价函数则导致图 4(b) 所示的布局, 形成阴影所示的不规则区域.



(a) 可重构硬件初始状态有任务 T_1、T_2; (b) 按面积差代价函数布局 T_3; (c) 采用被占用区域连续度的代价函数布局 T_3

图 4 DOAC 计算示例

定义 **7** 芯片上已占用区域连续度 DOAC(Degree of Occurpied Area Concatenation) 是指当前被占用可重构资源连续程度的量化值.

芯片上每一个可重构计算单元有四条单位长度的边界, 相邻两个单元共享一条边界. 例如图 5 中, 单元 X 有边界 a, b, c, d, 单元 X 和 Y 共享边界 d. 定义 S_T 为单个任务 T 在芯片上覆盖的区域, Γ 是芯片上正在运行的所有任务覆盖的区域的并集, 即 $\Gamma=\bigcup S_T$. l 为某可重构单元单位长度的边界, 若 l 位于 Γ 内部, 则记 $l\in\Gamma$, 若 l 是整个可重构芯片边缘的一部分(如图 5 中的 l), 则记 $l\in$ Border, 有:

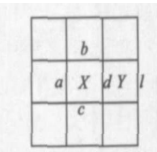


图 5 可重构计算单元边界相邻示例

$$\text{DOAC}=|\{l\mid l\in\text{Border}\vee l\in\Gamma\}| \quad (1)$$

例如图 4 中, DOAC(a) = 32; DOAC(b) = 34; DOAC(c) = 35. 直观上, DOAC 越大任务布局越紧密, 碎片越少. 因此, 可将 DOAC 作为计算布局代价函数的依据.

对于某一布局 G, 定义代价函数 KTVCost = Max-DOAC(G), 其中 Max 取充分大的正值. 例如图 4 中 DOAC(c) > DOAC(b), 故 KTVCost(c) < KTVCost(b), 布局器若采用 KTVCost 代价函数, 布局结果将如图 4(c) 所示. 由此例和分析可知, KTVCost 代价函数可在布局过程中减少潜在碎片, 从而增加后续任务布局成功的机会.

一般, 新任务 $T(h, w, a, e)$ 产生的 DOAC 增量为 $\triangle_{\text{DOAC}}=(2\times h\times w-h-w)+X$. 其中 $(2\times h\times w-h-w)$ 为新任务模块内部产生的 DOAC 增量, X 为新任务模块边界产生的 DOAC 增量, 可通过遍历任务 T 的边界在常量时间内得到.

4 KTVS 调度算法

可重构硬件可以随时间的变化, 改变芯片上给定面积的逻辑功能. 这种计算方式结合了 ASIC(Application Specific Integrated Circuit) 和 ISP(Instruction Set Processor) 技术的特点, 实现了在时间域和空间域上均可编程. 因此, 可重构计算资源的管理不同于传统计算资源的管理, 需要统筹分配硬件任务占用的时间和空间资源, 从而最大限度的发挥系统内在的并行性, 充分利用可重构计算资源. 为此, 本文扩展了第 3 节的 KTVP 算法, 将硬件任务运行的时间和所占空间看作类似的, 处于不同维度的计算资源, 进行统一的管理和分配. 为说明方便, 本节假设可重构芯片采用一维资源模型.

4.1 数据结构

第一, 对具有 W 列 RLU 的可重构芯片, 建立时空坐标系 Ψ, 横轴代表 1~W 列可重构单元, 纵轴代表运行时间. 在一维模型下, 硬件任务 $T(h, w, a, e)$ 在 Ψ 中表现为宽度为 w, 高度为 e 的矩形. 例如图 6 即为宽度 $W=10$ 的可重构芯片对应的时空坐标系 Ψ, $T_1(h:h1, w:4, a:0, e:5)$ 在 Ψ 中用 4×5 的矩形表示.

与 3.1 节的权值矩阵 M 类似, 坐标系 Ψ 中的每个方格

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

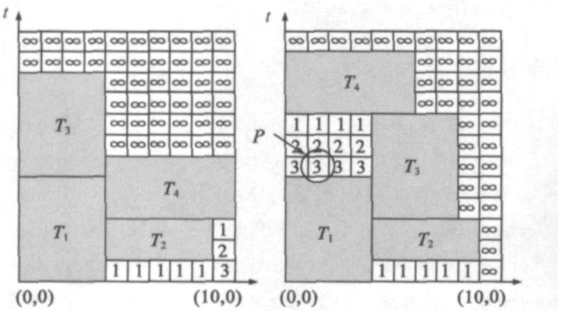


图 6　时空坐标系 Ψ

都代表在对应单位时间内某列 RLU 的使用情况, 并有一个权值 p 与其对应. $p=k$ 代表方格对应横坐标位置的可重构资源在此单位时间内未被占用, 且在未来 $k-1$ 个单位时间内仍未被占用; $p=0$ 代表此单位时间该位置对应的资源已被占用. 例如图 6(b)中 P 所指示的方格, 代表可重构芯片上的第 2 列资源在时间区间[5, 6]内空闲, 且在未来的 2 个单位时间(即时间区间[6, 8])内均空闲. 通过时空坐标系 Ψ 即可掌握当前芯片资源的利用情况, 同时也可规划未来芯片资源的分配.

第二, 建立链表 C 保存硬件任务在 Ψ 中对应的顶点信息. 初始时 C 仅包含(0, 0)和(W, 0)两点. 顶点 $v(x, t)$ 在链表中按纵坐标 t 值增加的顺序排列.

4.2　算法描述

建立时空坐标系 Ψ 和顶点队列 C 后, 调度问题实际上转化为硬件任务(以时间作为任务的一维)在 Ψ 中的布局问题. 因此, 调度和布局的过程可统一采用 KTV Algorithm 完成, 其中匹配仍由 Matchr Algr 完成, 代价函数仍采用 KTVCost. 与 KTVP 算法类似, KTVS 算法也包括相应的若干子算法, 此处不再赘述.

图 6(a)对应 $\boldsymbol{T}_1(h:h1, w:4, a:0, e:5)$、$\boldsymbol{T}_2(h:h2, w:5, a:1, e:2)$、$\boldsymbol{T}_3(h:h3, w:4, a:2, e:5)$和 $\boldsymbol{T}_4(h:h4, w:6, a:3, e:3)$经 KTVS 算法调度的结果, 各任务下缘在 Ψ 中对应的纵坐标即为任务的触发时间. 对比文献[7]中的 Stuffing 算法(调度 T_1~ T_4 结果如图 6(b)所示), KTVS 算法将硬件任务安排的更紧密, 有利于提高芯片利用率和降低任务拒绝率.

5　仿真实验及结果分析

5.1　仿真实验环境与实验方案

为测试 KTVP 和 KTVS 算法的性能, 我们开发了 ReEmu 仿真环境并进行了仿真实验. ReEmu 仿真环境的软硬件配置如下: CPU(Intel Pentium III 1.0GHz)、Memory(256M); Operating System(Fedora Core II, Linux Kernel 2.4.18); Compiler(GCC 2.96). 仿真环境最小时间单位为 10ms, 模拟可重构芯片的规模相当于 Xilinx Virtex XCV1000 FPGA, 大小为 64×96 计算单元.

定义 **8**　布局仿真硬件任务集 PEHTS(Placement Emulation Hardware Task Set)指布局仿真实验中一段时间内连续随机生成的硬件任务所构成的集合, 由 4 维向量 $\boldsymbol{PEHTS}(c, h_{max}, w_{max}, Delayfactor)$ 描述. 其中, c 为该任务集中硬件任务的数量; h_{max} 和 w_{max} 分别是任务集中各硬件任务的最大高度和最大宽度. $\forall \boldsymbol{T}(h, w, a, e) \in \boldsymbol{PEHTS}$, 有 $h \sim R(3, h_{max})$, $w \sim R(3, w_{max})$; $Delayfactor = D/E$, 是任务延迟因子, D 为任务集中相继触发任务的平均间隔时间, E 为任务的平均执行时间. 上述定义中, R 表示均匀分布.

定义 **9**　调度仿真硬件任务集 SEHTS(Scheduling Emulation Hardware Task Set)指调度仿真实验中一段时间内连续随机生成的硬件任务构成的集合, 在一维资源模型下, 由 7 维向量 $\boldsymbol{SEHTS}(c, w_{min}, w_{max}, E_{min}, E_{max}, D_{min}, D_{max})$ 描述, 其中: c 为该集合中硬件任务的数量; w_{min} 和 w_{max}、E_{min} 和 E_{max}、D_{min} 和 D_{max} 分别是集合中各硬件任务宽度、执行时间和相继到达任务间隔时间的最大值和最小值. $\forall \boldsymbol{T}(h, w, a, e) \in \boldsymbol{SEHTS}$, 有 $w \sim R(w_{min}, w_{max})$, $e \sim R(E_{min}, E_{max})$, $D \sim R(D_{min}, D_{max})$.

测试时 ReEmu 仿真环境的工作方式如下: 先指定硬件任务集的所有相关参数, 然后按参数生成任务集中的所有任务. 系统根据所生成任务 $\boldsymbol{T}(h, w, a, e)$ 的 a 参数在时间线上添加任务到达事件. 当测试布局算法性能时, 调用布局器立即为该任务在当前芯片上寻找最佳布局位置. 当测试调度算法性能时, 调用调度器, 为硬件任务在当前或未来的芯片上安排合适的触发时间和加载位置.

MH 布局算法[2]和 Stuffing[7]调度算法具有一定代表性, 在已有算法中开销较小, 布局/调度质量较高, 所以本文实验方案安排如下: (1)对比 KTVP 算法与 MH 算法的任务拒绝率和累计执行时间; (2)对比 KTVS 算法与 Stuffing 算法的任务拒绝率和累计执行时间.

5.2　实验结果及性能分析

图 7 显示了 KTVP 算法与 MH 算法 TRR 指标的对比. 任务集为 $\boldsymbol{PEHTS}(1000, h_{max}, w_{max}, 0.05)$, h_{max}/w_{max} 分别取 1~ 50 的整数, 图中的每个数据点, 都是运行 25 个硬件任务集的平均结果. 可见 KTVP-BF-4v 在多数情况下稍优于 MH 算法, 但两者差距最大处亦不到 0.5%. KTVP-BF-1v 与 MH 算法的差距也仅为 2%, 明显好于文献[3~ 5]的结果.

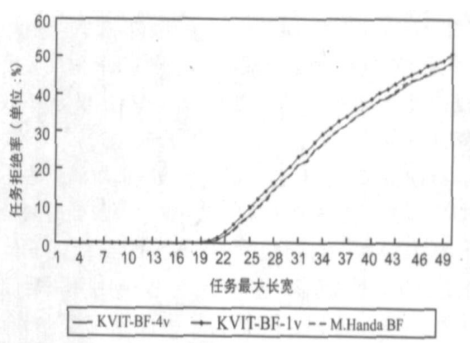


图 7　KTVP 系列算法与 MH 算法任务拒绝率对比

我们在 ReEmu 平台上随机触发 30000 个硬件任务进行布局, 分别采用 KTVP-BF-1v、KTVP-BF-4v 和 MH 算法, 结果如表 1 第 2~ 4 列所示. KTVP-BF-4v 的平均执行时间不到 MH 算法的 1/3. 应该看到, 由于 KTVP 系列算法的时间复杂度仅为 $O(N)$, 随着可重构芯片规模的增长, 它的优势将更加明显.

图 8 显示了分别采用 Stuffing、KTVS-BF-1v、KTVS-BF-4v 三

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

种算法针对 $SEHTS_A$(10000, 7, 15, 1, 100, 5, 100, 3, 7), $SEHTS_B$(10000, 15, 25, 100, 250, 5, 100, 3, 7), $SEHTS_C$(10000, 25, 35, 250, 400, 5, 100, 3, 7)任务集调度时的任务拒绝率. 可见, KTVS-BF-1v 和 KTVS-BF-4v 的 TRR 均低于 Stuffing 算法. 一般认为 TRR 超过 40% 的调度器无法实用, 而 $SEHTS_B$ 和 $SEHTS_C$ 对应三种算法的TRR 均超过 50%, 因此在实际应用中应尽量避免生成宽度过大的任务.

表 1 KTVP/KTVS 系列算法与 MH/Stuffing 算法执行时间对比

单位: 10^4ms	KVTP-BF-1v	KVTP-BF-4v	MH	KTVS-BF-1v	KTVS-BF-4v	Stuffing
1	0.7824	2.4071	7.0768	0.5743	1.3523	5.4646
2	0.7676	2.3901	7.9444	0.5963	1.3478	5.4738
3	0.7691	2.3686	7.2323	0.6027	1.3382	5.4779
平均	0.7730	2.3886	7.4178	0.5911	1.3461	5.4721

我们在 ReEmu 平台上随机触发 30000 个硬件任务进行调度, 结果如表第 5~7 列所示. KTVS-BF-4v 算法的执行时间约为 Stuffing 算法的 24.6%.

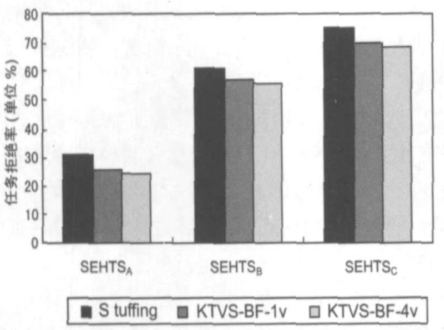


图 8 KTVS 系列算法与 Stuffing 算法任务拒绝率对比

6 结论与展望

纵观已有的部分可重构资源管理算法, 有的过分注重布局/调度质量, 导致算法开销过大; 有的牺牲质量以满足执行速度方面的要求, 资源浪费严重. 本文提出的 KTVP/KTVS 算法克服了已有算法的不足, 其优点在于: (1) 减小了调度开销. 由于采用的数据结构简单, 故算法的平均执行时间较已有算法有很大进步, 其中 KTVP 算法的时间复杂度仅为 $O(N)$, 这使得动态布局器在实用角度又迈进了一步; (2) 布局/调度质量高. KTVP 的 TRR 指标与当前效果最好的布局算法基本一致; KTVS 系列算法的 TRR 指标较当前效果最好的调度算法有所提高; (3) 布局算法响应时间的确定性强. KTVP 消除了文献[1, 2]算法的抖动现象, 适合实时应用. (4) 使用灵活. KTVP/KTVS 算法包括一系列子算法, 应用中可根据需要选择合适的子算法.

在本文工作的基础上, 今后可在商用 FPGA 器件上实现硬件任务加载接口, 并建立布局/调度算法的实际测试平台, 对 ReEmu 仿真环境所测得的结果进一步实际验证.

参考文献:

[1] 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.

[2] Handa M, Vemuri R. An efficient algorithm for finding empty space for online FPGA placement[A]. Proceedings of the 41th Design Automation Conference[C]. New York: ACM Press, 2004. 960-965.

[3] Walder H, Steiger C, Platzner M, Thiele L. Online scheduling and placement of real-time tasks to partially reconfigurable devices[A]. IEEE Real-Time Systems Symposium[C]. New York: IEEE Computer Society, 2003. 224-225.

[4] Walder H, Steiger C, Platzner M. Fast online task placement on FPGAs: Free space partitioning and 2-D hashing[A]. International Parallel (and Distributed) Processing Symposium[C]. New York: IEEE Computer Society, 2003. 178.

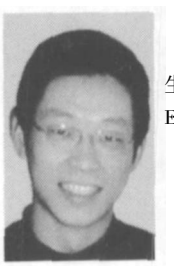
[5] Ahmadinia A, Bobda C, Jurgen T. A new approach for on-line placement on reconfigurable devices[A]. International Parallel (and Distributed) Processing Symposium[C]. New York: IEEE Computer Society 2004. 134.

[6] Handa M, Vemuri R. An integrated online scheduling and placement methodology[A]. Jürgen B. Field Programmable Logic and Application: 14th International Conference[C]. Berlin: Springer, 2004. 444-453.

[7] Steiger C, Walder H, Platzner M. Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks[J]. IEEE Transactions on Computers, 2004, 53(11): 1393-1407.

[8] Danne K, Platzer M. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware[A]. Rissa T. Proceedings of the 2005 International Conference on Field Programmable Logic and Applications[C]. New York: IEEE Computer Society, 2005. 568-573.

作者简介:



齐 骥 男, 1978 年生于黑龙江, 博士研究生, 研究方向: 可重构计算, 嵌入式操作系统.
E-mail: qiji@mail.ustc.edu.cn

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net