计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
44(8):1439~1447, 2007

一种面向动态可重构计算的调度算法

齐 骥[1] 李 曦[1] 于海晨[2] 胡 楠[1] 龚育昌[1] 王立刚[1]
[1](中国科学技术大学计算机科学技术系 合肥 230027)
[2](大连理工大学电子与信息工程学院 大连 116024)
(ycgong@ustc.edu.cn)

A Scheduling Algorithm for Dynamic Reconfigurable Computing

Qi Ji[1], Li Xi[1], Yu Haichen[2], Hu Nan[1], Gong Yuchang[1], and Wang Ligang[1]
[1](*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027)
[2](*School of Electronics and Information Engineering, Dalian University of Technology, Dalian* 116024)

Abstract Dynamic reconfigurable system can be partially configured at run-time, without disturbing the execution of original functions. Such system has become the focus of research in recent years. The scheduling of hardware tasks is one of the critical factors that is concerned closely with the performance of the dynamic reconfigurable computing system. In this paper, an MGS (minimum gap scheduling) algorithm is proposed, which employs a 2-dimentional time-space coordinate system for the hardware tasks to allocate the resources on chip and the time slices of execution. The concepts of task shadow and cost function are also presented to facilitate the process of scheduling. The algorithm can reduce the waste of reconfigurable resources and can effectively improve the parallelism of the tasks. The algorithm is quite easy to implement and is suitable to be applied for both situations of real-time and non-real-time. The simulation results show that the MGS algorithm can not only reduce the scheduling overhead, but also achieve higher chip utilization and lower task rejection ratio than using other existent scheduling algorithms at the same time.

Key words dynamic reconfiguration; scheduling algorithm; placement algorithm; time-space coordinate system; cost function

摘 要 硬件任务的调度是影响动态可重构系统性能的关键因素之一.提出一种任务间最小空隙调度算法MGS(minimum gap scheduling algorithm),该算法借助任务投影和调度代价函数,采用二维时空坐标系协调各硬件任务占用的芯片资源和执行时间,可有效减少系统资源浪费,提高并行度.MGS算法策略直观,调度开销小,且同时适用于实时和非实时场合.仿真实验表明,与已有算法相比,MGS算法不但降低了硬件任务的调度时间开销,而且具有更高的芯片利用率和更低的任务拒绝率.

关键词 动态可重构;调度算法;布局算法;时空坐标系;代价函数

中图法分类号 TP302

由CPU和FPGA(field programmable gate array)构成的可重构混成系统既可发挥专用集成电路的速度优势,又具有指令集处理器的灵活性.FPGA作为当前主流的可重构硬件,单片容量已达数百万门,且具有动态部分可重构功能.这意味着它可在运行时动态改变某一部分的配置,又允许其未更改部分仍按原有方式正常工作.可重构计算技术的发展对运行时系统环境提出了更高的要求.支持可重构混成体系结构的操作系统OSHRS(operating system for hybrid reconfigurable system)为可重构计算系统

收稿日期:2006-05-15;修回日期:2007-05-08
基金项目:高校博士点基金项目(20050358040);安徽省自然科学基金项目(070412030)

提供基本的运行环境和编程模型,可以对硬件任务模块进行布局和调度,从而更有效地利用可重构计算资源. 本文的讨论基于如图 1 所示的可重构混成系统结构,其中 OSHRS 的主要组件有:加载器、调度器、布局器、任务划分部件等.

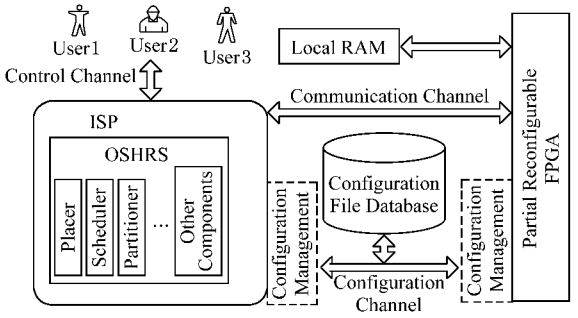

Fig. 1 The architecture of a reconfigurable hybrid system.

图 1 可重构混成系统结构

在 OSHRS 中,任务划分部件将应用逻辑分为若干面积适合的硬件任务,由调度器为这些硬件任务分配计算资源并决定其加载和执行时机. 当采用较复杂资源模型时,有专门的布局器负责管理和维护 FPGA 上的可重构资源. 本文的讨论针对动态调度,即在硬件任务执行序列未知的情况下进行调度.

文献[1]中提出的方法为缓冲队列中的硬件任务指定优先级,而后按优先级依次执行,但该方案的芯片利用率较低,且不支持实时调度. 文献[2]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[3]提出了两种多上下文 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. 文献[4]提出了一种调度和布局相结合的算法,降低了布局产生的开销,但在调度方面对任务接收率和芯片利用率的改善较小. 文献[5]提出了一种支持可重构混成体系结构的操作系统和一种基于抢占的双层任务调度模型,但该调度模型要求硬件任务设计者提供方法用于维护任务切换过程中需要保存的信息,通用性不强. 国内相关研究甚少,文献[6]综述了基于 FPGA 的可重构计算硬件结构的基本技术. 文献[7]提出了一种基于统一多任务模型可重构系统的实时操作系统,它不支持动态调度. 在已有调度算法中,由于文献[2]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. 但 Stuffing 算法单独管理系统的时间资源和空间资源,其资源模型不能有效反映动态可重构系统调度问题的实际需求,因此仍存在较大改进余地.

针对已有硬件任务调度算法的局限,本文提出一种同时支持实时、非实时硬件任务调度的算法 MGS(minimum gap scheduling). 该算法借助二维时空坐标系,采用一种新的编码策略进行实时和非实时硬件任务的调度,可显著提高系统芯片利用率,降低任务拒绝率,同时降低了调度开销.

1 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.

定义 1. 硬件任务指经过内部综合,具有固定长度和宽度、一定时钟频率界限以及实时调度相关参数的矩形逻辑功能模块. 它是 OSHRS 中最小的执行单位,可由 5 维向量 $T(h,w,a,e,d)$ 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,d 是截止期(若 T 为非实时任务则 $d=\infty$). 本文称由硬件任务构成的集合为硬件任务集,用符号 Γ 表示.

定义 2. 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.

定义 3. 芯片利用率 CU(chip utilization)指在一段时间内,芯片上被占用的计算资源面积与芯片总面积比例的平均值.

任务拒绝率和芯片利用率是衡量调度器调度质量的两个重要指标.

定义 4. 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 FPGA 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何位置,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.

图 2 的(a)(b)分别是一维、二维资源模型下硬

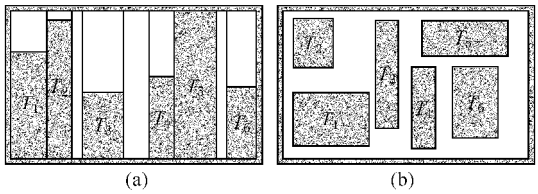

Fig. 2 1-D and 2-D resource model. (a) 1-D resource model and (b) 2-D resource model.

图 2 一维和二维资源模型. (a) 一维资源模型;(b) 二维资源模型

件任务的分布示意图. 二维模型可明显减少资源浪费,但其技术尚不成熟,很难在实际系统中应用;一维模型已有商用产品,例如 Xilinx 公司的 Virtex 系列芯片[8]. 因此,当前对可重构系统调度问题的研究大多基于一维模型,例如文献[2]中的 Stuffing 算法等. 本文的讨论同样基于一维模型.

定义 5. 碎片指在可重构芯片表面无法被利用的计算资源. 按碎片生成原因的不同,可分为两种:1)内部碎片. 离线设计工具进行内部综合时,很难使电路完全符合资源模型的限制,因此硬件任务内部可能存在未被利用的 RLU,本文称它们为内部碎片. 2)外部碎片. 由于 OSHRS 中布局、调度算法的局限,在硬件任务边界以外也会存在无法利用的 RLU,本文称它们为外部碎片.

内部碎片会降低芯片利用率,但其生成与调度算法无关. 因此,本文的讨论不考虑内部碎片,即假定硬件任务对应的功能模块在竖直方向上与可重构芯片的高度相同.

非抢占式调度是指硬件任务在运行过程中不可被其他任务打断. 抢占式调度与之相反,调度器在硬件任务被打断时保存其上下文信息,待再次执行时恢复. 由于保存、恢复硬件任务上下文的开销很大,故本文讨论非抢占式调度器.

若调度器用 σ 表示,则当硬件任务 $T_i(h_i,w_i,a_i,e_i,d_i)$ 到达时,σ 需做出接收/拒绝判断. 若接收,σ 返回该任务被布局的位置和启动时间:

$$\sigma(T_i)\begin{cases} AcceptWith(x_i,s_i), \\ Reject. \end{cases} \tag{1}$$

本文称 $\sigma(T_i)$ 为 T_i 的调度,若结果为 $Accept$,则称 $\sigma(T_i)$ 为 T_i 的可行调度. 其中,s_i 为任务启动时间,x_i 为布局位置. 在实时情况下,σ 保证 $s_i+e_i\leqslant d_i$. 图 3 是一个实时硬件任务调度的例子,任务集 $\Gamma=\{T_1(5,2,0,2,4),T_2(3,4,2,3,9)\}$,有

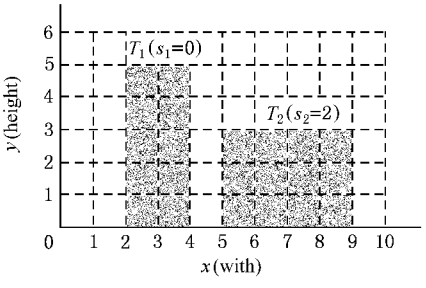

Fig. 3　An example of the hardware task scheduling.

图 3　硬件任务调度示例

$\sigma(T_1)=AcceptWith(2,0)$,$\sigma(T_2)=AcceptWith(5,2)$.

定义 6. 若硬件任务集 Γ 经 σ 调度后,有 $\forall T_i,T_j\in\Gamma\Rightarrow[(x_i+w_i)\leqslant x_j]\vee[x_i\geqslant(x_j+w_j)]\vee[(s_i+e_i)\leqslant s_j]\vee[s_i\geqslant(s_j+e_j)]$,则称 Γ 为 σ 的可调度任务集,σ 则为 Γ 的可行调度器,称由 σ 调度 Γ 的结果为可行调度序列,用 $\sigma(\Gamma)$ 表示($\sigma(\Gamma)$以各任务调度结果中 s_i 增加的顺序排列).

例如图 3 中由 σ 调度任务集 Γ 产生的可行调度序列为 $\sigma(\Gamma)=\{(T_1,2,0),(T_2,5,2)\}$.

定义 7. 对于任务 $T(h,w,a,e,d)$,称时间区间 $[a,d-e]$ 为 T 的松弛区间,则 $d-e-a$ 即为松弛区间的长度,本文称其为松缓时间. 只要 T 在松弛区间内开始执行,就肯定可以在截止期 d 之前完成.

图 4 显示了硬件任务从到达系统至执行结束经历的若干种状态. $current\_time$ 表示系统当前时间,被确保运行的任务在 $current\_time=s_i$ 时触发.

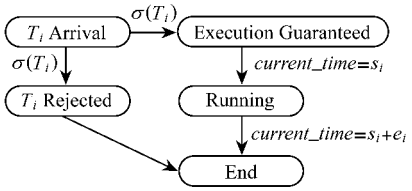

Fig. 4　State transforming of the hardware task.

图 4　硬件任务状态转换

2　调度算法

本文提出了一种应用于一维资源模型的非抢占、动态、实时/非实时硬件任务调度算法 MGS,该算法将任务的执行时间和所占资源结合考虑,以增加硬件任务并发度,提高芯片利用率,降低任务拒绝率.

2.1　MGS 实时调度算法

MGS 调度算法用二维时空坐标系描述系统的时间和空间资源,它分两步为新到达任务选择最佳布局位置和启动时机:1)为新到达的任务 T 找到所有满足截止期限制的可行调度;2)在所有可行调度中,选择最优的. 第 2.1.1 节和第 2.1.2 节通过将坐标系进行编码可解决问题 1),第 2.1.3 节通过定义代价函数可解决问题 2).

MGS 算法的目标是降低 TRR,同时尽可能地提高 CU.

2.1.1 数据结构

第1,对具有 W 列 RLU 的可重构芯片,建立二维时空坐标系 Ψ,横轴代表 $1\sim W$ 列可重构单元,纵轴代表运行时间,其精度定义为对调度器有意义的最小时间间隔. 例如图5(a)(b)即为宽度 $W=10$ 的可重构芯片对应的时空坐标系 Ψ.

定义8. 若任务 $T_i(h_i,w_i,a_i,e_i,d_i)$ 经 σ 调度有 $\sigma(T_i)=AcceptWith(x_i,s_i)$,则在 Ψ 中,直线 $x=x_i,x=x_i+w$ 与直线 $y=s_i,y=s_i+e_i$ 相交形成的矩形区域即为 T_i 的任务投影 TS(task shadow),用 TS_i 表示.

TS_i 即代表在时间区间 $[s_i,s_i+e]$ 内,资源 $[x_i,x_i+w]$ 被任务 T_i 占用. 例如图5(a)(b)中的 $TS_1\sim TS_6$ 即为表1中各任务经 σ 调度后在 Ψ 中形成的 TS.

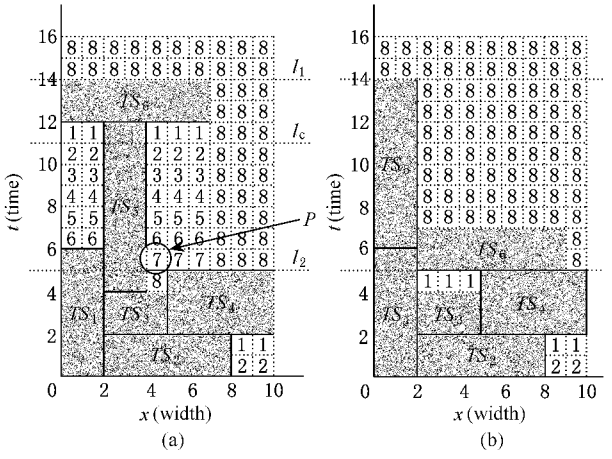

Fig. 5 The time-space coordinate system and feasible schedulings. (a) Stuffing algorithm scheduling and (b) MGS algorithm scheduling.

图5 时空坐标系 Ψ 与可行调度. (a) Stuffing 调度算法;(b) MGS 调度算法

坐标系 Ψ 中的每个方格都代表在对应单位时间内某列 RLU 的使用情况,有一个权值与其对应,该权值为正整数 k 或0. 正整数值 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;0代表此单位时间该位置对应的资源已被占用. 为方便起见,本文称坐标系 Ψ 中权值为正的区域为可用区域. 例如图5(a)中 P 所指示的方格,代表可重构芯片上的第5列资源在时间区间[5,6]内空闲,且在未来的6个单位时间(即时间区间[6,12])内均空闲. 通过时空坐标系 Ψ 和 Ψ 中的任务投影,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.

Table 1 An Example of Hardware Task Set Γ

表1 任务集 Γ 示例

Parameters	T_1	T_2	T_3	T_4	T_5	T_6
w_i	2	4	2	5	2	7
h_i	–	–	–	–	–	–
a_i	0	0	1	2	3	5
e_i	6	2	2	3	8	3
d_i	10	6	8	7	15	13

(The unit of w_i is RLU, the unit of a_i, e_i and d_i is the least meaningful time interval)

第2,建立链表 C 保存当前已被接收但尚未运行结束的任务所对应 TS 的顶点信息. 初始时 C 仅包含(0,0)和(W,0)两点. 若顶点 v 的坐标表示为(x,y),则 v 在链表中按照 y 值增加的顺序排列. C 用于为硬件任务选择调度的时机和布局位置. 例如,图6表示出图5(b)对应的顶点队列 C.

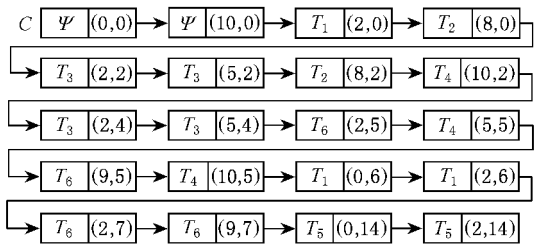

Fig. 6 An example of vertex array.

图6 顶点队列示例

2.1.2 算法描述

建立二维时空坐标系 Ψ 和 TS 顶点队列 C 后,调度问题实际上转化为将 TS 放置在 Ψ 中可用区域问题. MGS 调度算法尝试将新到达的硬件任务对应的 TS 放置在 Ψ 中已有 TS 的顶点处,利用 Ψ 中的编码可迅速判断某次放置的可行性(即是否完全处于可用区域内,如 Match-Algr 所示). 这样,新硬件任务对应的 TS 就沿着当前运行中硬件任务 TS 的顶点和边界发展,不但策略简单,并且减少了 TS 之间空隙造成的资源浪费. 放置 TS 时,若它的4个顶点之一与 C 中某点 $v(x,y)$ 重合,则称 TS 与 v 相匹配. 若某次匹配时该 TS 不与任何其他 TS 相重叠则匹配成功;否则匹配失败.

MGS 调度算法的伪代码如 MGS-Algr 所示. 当新任务 $T_i(h_i,w_i,a_i,e_i,d_i)$ 到达时,MGS 算法尝试调用 $TaskShadowMatch()$ 将 TS_i 与 C 中的各顶点相匹配(第③行),得到 C 中可成功匹配 TS_i 的所有顶点,进而找到代价函数 $MGS\text{-}Cost$ 最低的点(第⑤⑥⑦行,关于 $MGS\text{-}Cost$ 代价函数的详细说明见第

2.1.3节).最后,若找到可匹配成功的顶点,则更新 Ψ 中的相关权值(第⑫行),返回 accept;否则返回 reject(第⑬⑭行).

```
MGS-Algr
① MGS (Hardware_Task T, TimeSpace_Coodinate Ψ) {
② for ((v in C) && (v in Area_Slack)) {
③ succeed = TaskShadowMatch(T, v);
④ if (succeed) {
⑤ if (MGS-Cost(T, v) < MinCost) {
⑥ result = v;
⑦ MinCost = MGS-Cost(T, v);
⑧ }//end if
⑨ }//end if
⑩ }//end for
⑪ if (result <> Null) {
⑫ update (T, result, Ψ);
⑬ return accept;
⑭ }else return reject;
⑮ }//end MGS.
```

其中匹配的过程由函数 *TaskShadowMatch* 完成,算法如 Match-Algr 所示:

```
Match-Algr
① TaskShadowMatch (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 succeed;
⑥ }
```

图5(b)中显示的各 TS 可代表 MGS 算法调度表1中任务集 Γ 生成的调度序列.对比文献[1]中的 Stuffing 算法(产生的调度序列对应图5(a)中所示的各 TS),可见 MGS 算法使得 TS 在 Ψ 中分布得更紧密,有利于提高 CU 和降低 TRR.

定义9. 时间线指 Ψ 中与 x 轴平行的直线 l:$y = t_x$,它代表系统运行过程中的某一特定时刻.

在线调度器的工作流程如图7所示.当前时间线 l_c:$y = current\_time$ 代表当前时刻,每过一个时间单位,l_c 就沿着 y 轴向上平移一个单位.可见,l_c 穿过的 TS 即对应运行中的硬件任务,l_c 以上的 TS 即对应已确保执行的硬件任务,l_c 以下的 TS 即对应已执行结束的硬件任务.例如在图5(a)中,若当前时间线为 l_c,则运行中任务为 T_5,已执行结束任务为 $T_1 \sim T_4$,已确保执行的任务为 T_6.相比传统的基于规划的调度算法(如文献[2]中的 Stuffing 算法),MGS 算法不必频繁更新 Executing List,Reservation List,Empty List 等数据结构,效率更高也更直观.

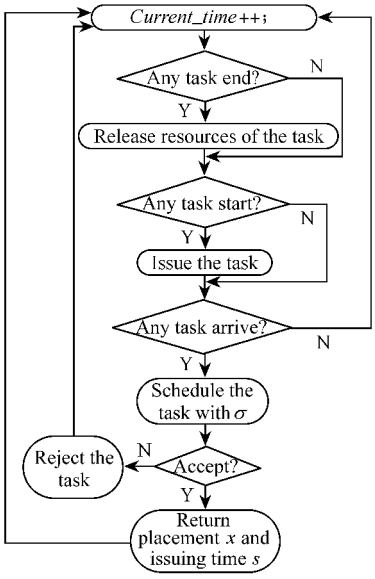

Fig. 7　The work flow of the scheduler.

图7　调度器的工作流程

需要指出,算法 Match-Algr 的真正实现可能较上述伪代码更为复杂.函数 *TaskShadowMatch* 只尝试将新任务投影 TS 的单个顶点与 v 相匹配,此种情况被称为 1v 情形;类似地,若布局时考虑将 TS 的任意两个顶点与 v 相匹配,则为 2v 情形;依此类推,还有 3v,4v 情形.由于 4v 的算法考虑最全面,所以运行时间更长,所生成调度序列的质量也优于其他算法.可见,MGS 实际上是一系列调度算法的集合,下文将用具体的名字指示此集合中的特定算法,例如称4顶点匹配的 MGS 算法为 MGS-4v.

2.1.3　代价函数

代价函数用于指导调度器在 Ψ 中放置 TS,即在所有可行的位置中选择最优的,是提高 CU 的关键之一.

定义10. 时空片段指 Ψ 中某两条时间线 l_{t_start} 和 l_{t_end} 之间的区域,用 $Interval[t\_start, t\_end]$ 表示.

假设从系统启动开始,共有 N 个任务 $T_1, T_2, \cdots, T_n$ 在芯片上运行,对应的任务投影分别为 $TS_1, TS_2, \cdots, TS_n$.$t_1, t_2, \cdots, t_n$ 分别为 $TS_1, TS_2, \cdots, TS_n$ 在 $Interval[t\_start, t\_end]$ 之间的高度(即在时间轴方向上被 l_{t_start} 和 l_{t_end} 截得的长度).芯片有 W_{chip} 列 RLU,总面积为 A_{chip}.在时刻 t 芯片在水平方向上被硬件任务占用了 $w_{in\_use}(t)$ 列,面积为

$A_{\text{in\_use}}(t)$,当不考虑内部碎片时,有:

$$CU = \int_{t\_start}^{t\_end} \frac{CU(t)}{t\_end - t\_start}\mathrm{d}t = \int_{t\_start}^{t\_end} \frac{A_{in\_use}(t)}{A_{\text{chip}} \times (t\_end - t\_start)}\mathrm{d}t = \int_{t\_start}^{t\_end} \frac{w_{in\_use}(t) \times H}{W_{\text{chip}} \times H \times (t\_end - t\_start)}\mathrm{d}t = \frac{\int_{t\_start}^{t\_end} w_{\text{used}}(t) \times \mathrm{d}t}{W_{\text{chip}} \times (t\_end - t\_start)} = \frac{t_1 \times w_1 + t_2 \times w_2 + \cdots + t_n \times w_n}{(t\_end - t\_start) \times W_{\text{chip}}}. \quad (2)$$

可见,系统某段时间内的 CU 即为 Ψ 中相应时空片段内各 TS 所覆盖的面积和与此时空片段总面积之比. 因此若调度器能将各 TS 在 Ψ 中安排得尽量紧凑,减少它们之间的空隙,则可提高 CU. 进而,由于 CU 升高,单位时间内芯片容纳的任务就更多,因此对 TRR 指标也会有所改善.

定义 11. 任务投影连续度 $DTSC$(degree of TS concatenation)是指特定时空片段内各 TS 的连续程度.

坐标系 Ψ 中每个方格有 4 条单位长度的边界,相邻两个方格共享一条边界. 例如图 8 中,当前时间线为 x 轴,截止期对应的时间线为 l_1,单元 X 有边界 a,b,c,d,单元 X 和 Y 共享边界 d. Θ 为某时空片段 $Interval[t_1,t_2]$ 中所有任务投影的并,即 $\Theta = \bigcup TS$,f 为某单位长度可重构单元的边界. 若 f 位于 Θ 内部,则记 $f \in \Theta$,若 f 是该时空片段边缘(如图 8 中虚线所示,上缘除外)的一部分(如图 8 中 f),则记 $f \in Border$,有:

$$DTSC = |\{f \mid f \in Border \vee f \in \Gamma\}|. \quad (3)$$

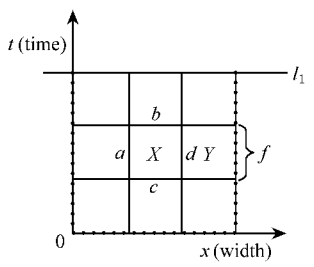

Fig. 8 An example of the adjacent units in a 2-D time-space coordinate system.

图 8 时空坐标系中边界相邻示意图

例如图 5(a)中,假设当前时间 $t_x = 5$,则 $DTSC(Interval[5,14]) = 52$;又如在图 5(b)中,$DTSC(Interval[5,14]) = 55$. 显然,$DTSC$ 越大,各 TS 在 Ψ 中分布得就越紧凑,对应调度序列的芯片利用率越高. 因此可将 $DTSC$ 作为调度器计算代价函数的依据.

定义 12. 对处于某时空片段 $Interval[t_1,t_2]$ 内的调度序列,定义代价函数为 $MGS\text{-}Cost(Interval[t_1,t_2]) = -DTSC(Interval[t_1,t_2])$.

一般,新到达任务 $T_i(h_i,w_i,a_i,e_i,d_i)$ 对应的 TS_i 产生的 $DTSC$ 增量为 $\Delta_{\text{DTSC}} = (2 \times e_i \times w_i - e_i - w_i) + X$. 其中 $(2 \times e_i \times w_i - e_i - w_i)$ 为 TS_i 内部产生的 $DTSC$ 增量,X 为 TS_i 边界产生的 $DTSC$ 增量,可通过遍历 TS_i 的边界得到. 例如对于图 5(a)中的时空片段 $Interval[5,14]$,当 T_6 尚未到达时,$DTSC = 29$;当 T_6 到达后,只需在原 $DTSC$ 上增加 T_6 对 $DTSC$ 的贡献即可,故 $DTSC = 52$.

MGS 调度算法不必计算 $DTSC$ 的绝对数值(MGS-Algr 第⑤行),仅需关注当新到达任务的 TS 置于时空片段中不同顶点时产生的 $DTSC$ 增量,返回增量最小顶点对应的布局位置和开始时间. 例如在图 5 中,当 $current\_time = 3$ 时 $T_5(2,h_5,3,8,15)$ 到达. 考虑时空片段 $Interval[currnet\_time, d_5]$. 若 T_5 到达前 $DTSC(Interval[currnet\_time, d]) = DTSC(Interval[3,15]) = DTSC_{\text{old}}$,则采用如图 5(a)所示的调度序列,有 $MGS\text{-}Cost(Interval[3,15]) = -DTSC(Interval[3,15]) = -(DTSC_{\text{old}} + \Delta_{\text{DTSC}}) = -(DTSC_{\text{old}} + 4)$ 采用如图 5(b)所示的调度序列,有 $MGS\text{-}Cost(Interval[3,15]) = -(DTSC_{\text{old}} + 10)$. 因此,按照 $MGS\text{-}Cost$ 代价函数,MGS 调度算法应将 T_5 对应的 TS 安排在图 5(b)所示的位置.

2.1.4 与 Stuffing 算法的对比

Stuffing 算法[2]对未来芯片的利用情况进行了一定的规划. 该算法在每个新任务 $T(h,w,a,e,d)$ 到达时,首先尝试将 T 在时刻 $t = a$ 触发,同时模拟所有已接收任务的执行. 如果 T 不与任何已接收的任务相冲突,则尝试成功,接收任务 T;否则,继续尝试将 T 在时刻 $t = a + t_{\text{unit}}$(t_{unit} 为对调度器有意义的最小时间间隔)触发,直到在某一时刻任务 T 被接收或者尝试的触发时间 t 已经超出任务 T 的松弛区间为止. 可见,Stuffing 算法总是将新到达的任务安排在尽量早的时间触发,该算法实现的调度目标是在保证新到达任务不与已接收任务冲突的前提下,尽早完成新到达的任务. 然而在实时调度环境下,任务只需在截止期之前完成即可,提前结束并不能提高芯片利用率或者降低任务拒绝率;同时,为了提前完成本不必提前完成的任务,很可能导致

任务拒绝率增高.

MGS 算法通过建立二维时空坐标系,以更直观的方式模拟未来芯片上硬件任务的触发和执行,通过代价函数尽量在一段时间内接收更多的任务. 与 Stuffing 算法[2]相比,MGS 算法在以下方面有所突破:1)MGS 算法借助时空坐标系描述和管理系统的时间和空间资源,从而能够统筹安排硬件任务的执行;2)MGS 算法通过定义代价函数将降低 TRR 和提升 CU 作为调度的直接目标,生成的调度序列更接近最优解.

2.2　MGS 非实时调度算法

在非实时场合,由于可重构系统中硬件任务的加载延迟相对较大,通常不用于实现交互密集型任务. 此时用户往往更关心系统吞吐率,而不是任务的平均响应时间. CU 和系统吞吐率之间具有密切关系,对于特定应用,总的来说 CU 越高系统的吞吐率就越大.

由式(2)可知,CU 即为 Ψ 中各任务投影面积与某时空片段总面积之比,通过将 Ψ 中的 TS 安排得尽量紧凑可提高 CU 和增大系统吞吐率. 由第 2.1 节中的相关分析可知,MGS 实时调度算法的总体目标符合上述非实时调度算法的需求,因此可将 MGS 调度算法扩展到非实时场合.

当采用 MGS-Algr 进行非实时调度时,硬件任务 $T(h,w,a,e,d)$ 的截止期 d 可视为无穷大(实际操作中可将 d 置为充分大的正值),松弛区间为当前时间线以上的开区域,即 $Interval\,[a,\infty)$. MGS 算法的其他部分不需改动.

3　仿真实验及结果分析

为测试 MGS 调度算法的性能和研究调度相关的重要参数,我们开发了 Recon-Emulator 调度仿真环境,进行了与文献[2]类同的仿真实验,对 MGS 和 Stuffing 调度算法进行了对比. 本节实验根据 FPGA 上典型应用中功能模块的相关参数设定硬件任务的面积、执行时间的取值范围,而后在此范围内随机生成硬件任务.

3.1　仿真实验平台与实验方案

Recon-Emulator 仿真环境的软、硬件配置如表 2 所示. Recon-Emulator 仿真环境可用于仿真计算单元矩阵式排列的可重构器件的布局和调度行为,它有效的最小时间单位为 10ms. 本文仿真实验模拟的可重构芯片规模相当于 Xilinx Virtex XCV1000 FPGA,大小为 64×96 个 RLU.

Table 2　Configuration of the Emulation Environment

表 2　仿真环境配置

Parameter	Value
CPU	Intel Celeron 2.8GHz
Memory	512MB
System Environment	Windows XP + Cygwin
Compiler	GCC 2.96

定义 13. 仿真硬件任务集 $EHTS$(emulation hardware task set)指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 9 维向量 $EHTS(c,\ w_{\min},\ w_{\max},\ L_{\min},\ L_{\max},\ E_{\min},\ E_{\max},\ D_{\min},\ D_{\max})$ 描述,其中:c 为该集合中硬件任务的数量. $w_{\min}$ 和 $w_{\max}$,$L_{\min}$ 和 $L_{\max}$,$E_{\min}$ 和 $E_{\max}$,$D_{\min}$ 和 $D_{\max}$ 分别是集合中各硬件任务宽度、松缓时间、执行时间、相继到达任务间隔时间的最大值和最小值. $\forall T(H,w,a,e,d)\in EHTS$,若硬件任务的松缓时间为 $Laxity$,相继到达任务的间隔时间为 $Delay$,则有 $w\sim R(w_{\min},w_{\max})$,$Laxity\sim R(L_{\min},\ L_{\max})$,$e\sim R(E_{\min},E_{\max})$,$Delay\sim R(D_{\min},\ D_{\max})$,其中 R 表示均匀分布,所有时间均以 10ms 为单位.

Recon-Emulator 仿真平台的工作方式如下:先指定仿真硬件任务集的 9 个参数,然后按照参数生成任务集中的所有任务. 系统根据所生成任务 $T(h,w,a,e,d)$ 的参数 a 在时间线上添加任务到达事件,若该任务被接收,则再根据参数 e 在时间线上添加任务结束事件. 仿真平台模拟时间线上所有事件的发生. 当处理任务到达事件时,调用调度器为任务分配时间、空间资源;当处理任务结束事件时,释放任务所占用的空间资源,更新调度器相关的数据结构.

由于 Stuffing 调度算法[2]具有一定代表性,在现有的实时调度算法中任务接收率最高,所以本文

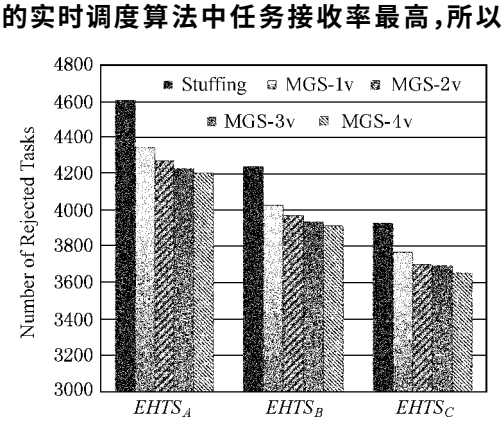

Fig. 9　Comparison of the number of rejected tasks.

图 9　任务拒绝数对比

的实验方案安排如下:1)对比 MGS 和 Stuffing 算法拒绝任务的数目(如图 9 所示)和芯片利用率 *CU*(如图 10 所示). 2)累计 MGS 和 Stuffing 调度算法的运算时间(如表 3 所示).

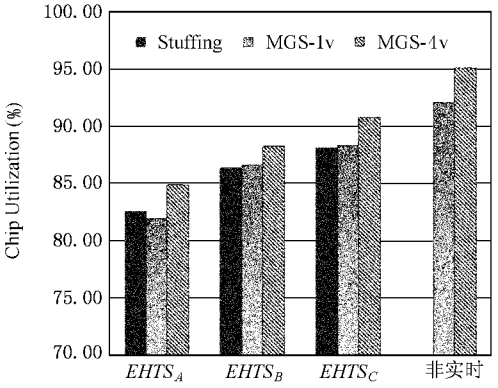

Fig. 10 Comparison of the chip utilization.

图 10 芯片利用率对比

3.2 实验结果及性能分析

图 9 显示了分别采用 Stuffing,MGS-1v,MGS-2v,MGS-3v,MGS-4v 五种算法针对不同松缓时间范围的 EHTS 调度时拒绝任务的数目,每种调度算法采用的仿真硬件任务集分别为 $EHTS_A$(10000,7,25,1,100,5,100,3,7),$EHTS_B$(10000,7,25,100,250,5,100,3,7),$EHTS_C$(10000,7,25,250,400,5,100,3,7). 由于这 5 种调度算法不仅考虑当前空闲的可重构资源,而且还尝试将任务布局在未来可能空闲的区域,所以 Laxity 越大,就有更多的机会为新到达的任务找到布局位置,拒绝的任务也就越少. 如图 9 所示,4 种 MGS 算法的调度质量均高于 Stuffing 算法,其中以 MGS-4v 算法为最好,可见本文定义的代价函数 *MGS-Cost* 是有效的.

图 10 显示了 Stuffing,MGS-1v,MGS-4v 三种算法 *CU* 的对比情况. 前 3 列数据分别对应 $EHTS_A$,$EHTS_B$,$EHTS_C$,可见,MGS 系列算法的 *CU* 明显好于 Stuffing 算法,对于 $EHTS_A$,$EHTS_B$ 和 $EHTS_C$,MGS-4v 算法比 Stuffing 算法分别提高了提高了 2.3%,1.9% 和 2.7%. 最后一列对应非实时仿真任务集,由于 Stuffing 算法不支持非实时应用,故仅对两种 MGS 算法进行比较.

表 3 显式了 3 种算法执行时间的对比情况,任务集为 *s*. 可见 MGS 系列算法的执行时间明显小于 Stuffing 算法,其中 MGS-1v 算法的执行时间约为 Stuffing 算法的 7.2%,MGS-4v 算法的执行时间约为 Stuffing 算法的 16.4%. 这是因为:1)MGS 算法不必维护 Stuffing 算法中需要频繁更新的 Executing List,Reservation List,Empty List 等数据结构;2)MGS 算法在时空坐标系中定义了对应于每个单元的权值,大大降低了调度、规划过程中尝试触发新到达硬件任务和判断是否存在冲突的开销. 在实际应用中,可根据需要选择一种 MGS 算法,使得系统既有较高的调度质量,又有合适的时间复杂度.

Table 3 Comparison of Executing Time

表 3 MGS 系列算法与 Stuffing 算法执行时间对比 ms

Data Serial	Stuffing	MGS-1v	MGS-4v
1	4.391×10^4	0.314×10^4	0.794×10^4
2	4.505×10^4	0.357×10^4	0.771×10^4
3	4.883×10^4	0.322×10^4	0.694×10^4
Average	4.593×10^4	0.331×10^4	0.753×10^4

4 结 论

已有面向可重构计算的硬件任务调度算法芯片利用率不高,且不支持非实时应用. 本文提出的 MGS 调度算法通过建立二维时空坐标系对时间和空间资源协调管理,呈现如下优点:1)可显著提高调度质量. 仿真实验表明,采用 MGS 调度算法可有效提高芯片利用率,并可在实时应用中降低任务拒绝率. 2)有效支持非实时应用. 3)减小了调度开销. 与 Stuffing 算法相比,MGS 调度算法采用的数据结构简单有效,不必创建多张表维护硬件任务的执行状态信息,故运行效率很高. 4)使用灵活. MGS 调度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.

本文的后续工作是在商用 FPGA 器件上实现硬件任务加载接口,并建立调度算法的实际测试平台,进一步实际验证 MGS 算法的有效性.

参 考 文 献

[1] M Handa, R Vemuri. An integrated online scheduling and placement methodology [C]. The 14th Int'l Conf on Field Programmable Logic and Application, Leuven, Belgium, 2004

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

[3] K Danne, M Platzer. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware [C]. The 15th Int'l Conf on Field Programmable Logic and Applications, Tampere, Finland, 2005

[4] A Ahmadinia, C Bobda, J Teich. A dynamic scheduling and placement algorithm for reconfigurable hardware [C]. Int'l Conf on Architecture of Computing Systems, Augsburg, Germany, 2004

[5] V Nollet, P Coene, D Verkest, *et al*. Designing an operating system for a heterogeneous reconfigurable SoC [C]. Int'l Parallel and Distributed Processing Symposium, Nice, France, 2003

[6] Li Renfa, Zhou Zude, *et al*. Hardware for reconfigurable computing [J]. Journal of Computer Research and Development, 2003, 40(3): 500-506 (in Chinese)
(李仁发, 周祖德, 等. 可重构计算的硬件结构[J]. 计算机研究与发展, 2003, 40(3): 500-506)

[7] Zhou Bo, Wang Shiji, Qiu Weidong, *et al*. SHUM-UCOS: A real-time operating system for reconfigurable systems using uniform multi-task model [J]. Chinese Journal of Computers, 2006, 29(2): 208-218 (in Chinese)
(周博, 王石记, 邱卫东, 等. SHUM-UCOS: 基于统一多任务模型可重构系统的实时操作系统[J]. 计算机学报, 2006, 29(2): 208-218)

[8] Xilinx Inc. Virtex-II platform FPGAs: Complete data sheet [OL]. http://www.xilinx.com, 2005-03

Qi Ji, born in 1978. Ph. D. of computer software and theory in the University of Science and Technology of China. His main research interests include reconfigurable computing, operating system and virtual machine.

齐　骥,1978 年生,博士,主要研究方向为可重构计算、操作系统.

Li Xi, born in 1963. Ph. D. and senior engineer. Also senior member of China Computer Federation. His main research interests include embedded system design.

李　曦,1963 年生,博士,高级工程师,中国计算机学会高级会员. 主要研究方向为嵌入式系统设计.

Yu Haichen, born in 1973. Ph. D. candidate and lecturer of Dalian University of Technology. His main research interests include networked control system and advanced process control technique.

于海晨,1973 年生,博士研究生,讲师,主要研究方向为网络化控制系统、先进控制技术.

Hu Nan, born in 1982. Ph. D. candidate of computer architecture since 2006. His main research interests include reconfigurable computing.

胡　楠,1982 年生,博士研究生,主要研究方向为可重构计算.

Gong Yuchang, born in 1943. Professor and Ph. D. supervisor. Her main research interests are system software and algorithm design.

龚育昌,1943 年生,教授,博士生导师,主要研究方向为系统软件、算法设计.

Wang Ligang, Born in 1979. Ph. D. of computer software and theory. His main research interests include operating system, real-time system and virtual machine.

王立刚,1979 年生,博士,主要研究方向为操作系统、实时系统和虚拟机技术.

Research Background

This work is supported by the National Natural Science Foundation of China (No. 60273042) and the Innovation Foundation of CAS. The projects aim at proposing an effective scheduling mechanism for hardware tasks, so as to improve the utilization of reconfigurable hardware and simplify the design flow of hybrid reconfigurable applications. The authors have focused their work on the area of RC (reconfigurable computing) for more than 5 years. Their research interests also include the dynamic placement/scheduling of hardware logic blocks and the programming model for RC. This paper discusses the fundamental abstraction model and proposes a series of scheduling algorithm—MGS, which have higher chip utilization and lower task rejection ratio than other existent scheduling algorithms. The authors are planning to implement the automatically loading interface on commercial FPGA devices and establish the actual experiment platform to verify the effectiveness of MGS algorithm.