一种面向部分可重构FPGA的混合实时调度算法

殷进勇 顾国昌 吴艳霞

(哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001)

摘 要 在CPU/FPGA平台上运行的实时任务通常由软/硬件子任务组成并存在优先约束关系。提出了一种软/硬件混合实时任务调度算法。在截止期限错失时刻,通过分析系统的运行情况,推导出实时任务可调度的充分条件。每个实时任务的硬件子任务分成多组,每组硬件子任务重叠配置到FPGA上。通过手工布局硬件子任务端口和总线端口,使得硬件子任务可动态的连接到系统总线上。实验表明,该算法能够满足任务的实时性,充分利用FPGA资源。

关键词 实时任务调度,软硬件混合任务,优先约束,CPU/FPGA

中图法分类号 TP316 **文献标识码** A

Hybrid Real-time Scheduling Algorithm Based on Partial Reconfigurable FPGA

YIN Jin-yong GU Guo-chang WU Yan-xia

(College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China)

Abstract Real-time task running on CPU/FPGA is usually composed of software/hardware subtasks with precedence constraints. A scheduling algorithm was proposed for software/hardware hybrid real-time tasks. The schedulability sufficient condition for real-time tasks was derived from analyzing what happens when the first deadline is missed. Hardware subtasks of each task were partitioned into several groups and subtasks within the same group were configured on PFGA overlapped. Hardware subtasks can be connected to the system bus dynamically by placing and routing ports of hardware subtasks and system bus manually. The experimental results demonstrate that the scheduling algorithm can meet real-time tasks' deadlines and make full use of FPGA.

Keywords Real-time tasks scheduling, Software/hardware hybrid tasks, Precedence constraints, CPU/FPGA

随着实时任务复杂性的增加,在实时系统设计中通常把一个实时任务分解成一组子任务,并用一个有向无环图(Directed Acyclic Graph,DAG)来表示。基于DAG的任务调度是一个NP难问题[1],针对此问题已提出了多种算法。近年来,一些研究者引进了人工智能中的优化方法以解决基于多处理机的任务调度问题。文献[2]基于遗传算法对多处理机任务调度问题进行优化,以减少所需的处理机个数和任务的平均响应时间。文献[3]通过引进竞争学习规则,改进了Hopfield网络,以解决实时任务调度问题。为了解决优先约束和资源约束问题,文献[4]提出了一种改进的蚁群算法。

与基于多处理机的软件任务调度相比,基于部分可重构PFGA的硬件任务调度是一个更加复杂的问题,它不仅要考虑任务的执行顺序,还要考虑任务的放置位置。根据可重构资源模型,调度算法可分为一维调度算法和二维调度算法。文献[5]提出了两种一维调度算法:Horizon和Stuffing,并扩展到二维模型。MGS[6]算法是对Stuffing算法的改进,MGS采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。Horizon和Stuffing算法以及MGS算法均需要维护系统的空闲区域,调度开销较大。为了降低调度开销,周学功[7]等从已调度的任务入手,研究了任务的放置问题,通过计算任务的覆盖域和可重构逻辑单元的最早有效时间,提出了CR(Compact Reservation)算法。

以上算法是针对软件任务或硬件任务单一任务类型提出的,而基于CPU/FPGA结构的任务通常由CPU执行的指令集(软件子任务)和FPGA执行逻辑电路(硬件子任务)两部分组成。文献[8]的混合任务调度算法采用预配置策略,降低了配置时间对任务执行的影响,但没有考虑任务的实时性。本文提出了一种软/硬件混合实时任务调度算法(Software/Hardware Real-time Tasks Scheduling,SHRTS),算法首先判定任务的可调度性,在系统运行时动态地调度软件子任务在多个CPU上执行,动态地配置硬件子任务,满足了任务的实时性。

1 任务和系统模型

1.1 任务模型

系统调度的任务是一组具有优先约束关系的实时周期任务$T=\{T_i \mid i=1,2,\cdots,n\}$。每个任务$T_i \in T$可用一个五元组的DAG来表示:

$$T_i=(ST_i,E_i,D_i,P_i,Prio_i)$$

式中,$ST_i=\{st_{ij} \mid j=1,\cdots,q\}$表示任务$T_i$的子任务集合;$E_i=$

到稿日期:2009-11-04 返修日期:2010-01-25 本文受Intel异构多核可重构计算体系结构研究基金项目资助。

殷进勇(1978—),男,博士生;顾国昌(1946—),男,教授,博士生导师。

$\{e_{ijk} \mid e_{ijk}$ 为 st_{ij} 到 st_{jk} 的有向边$\}$表示子任务之间的约束关系；D_i，P_i 和 $Prio_i$ 分别表示任务 T_i 的相对截止期限、周期和优先级。

子任务 st_{ij} 可用一个五元组来表示：

$$st_{ij}=\{C_{ij},w_{ij},h_{ij},prednum_{ij},type_{ij}\}$$

式中，C_{ij} 表示子任务的执行时间；w_{ij} 表示子任务的宽度，即在FPGA上占用的可重构单元列数；h_{ij} 表示子任务高度，即在FPGA上占用的可重构单元行数；$prednum_{ij}$ 表示子任务 st_{ij} 的直接前驱个数；$type_{ij}=\{Hard,Soft\}$，$Hard$ 表示 st_{ij} 是硬件子任务，$Soft$ 表示 st_{ij} 是软件子任务。当 st_{ij} 为软件子任务时，$w_{ij}=h_{ij}=0$。

$IPred(st_{ik})=\{st_{ij} \mid (st_{ij},st_{ik})\in E_i\}$，表示子任务 st_{ik} 的直接前驱子任务集合；$ISucc(st_{ij})=\{st_{ik} \mid (st_{ij},st_{ik})\in E_i\}$，表示子任务 st_{ij} 的直接后继子任务集合；如果 $|IPred(st_{ik})|=0$，那么称 st_{ik} 为开始子任务，如果 $|ISucc(st_{ij})|=0$，那么称 st_{ij} 为结束子任务。不失一般性，假设每个任务 T_i 只有一个开始子任务 st_{ientry} 和一个结束子任务 st_{iexit}。图1为用DAG描述的一个包含8个子任务的任务实例，方框表示硬件子任务，圆圈表示软件子任务。

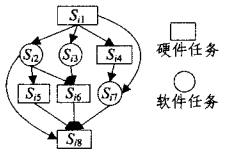

图1 实时任务结构

C_i^{CP} 表示任务 T_i 关键路径上的子任务执行时间总和；C_i^{soft} 表示任务 T_i 所有软件子任务执行时间的总和。

任务 T_i 在每个周期内执行一次，称为一个作业，它由 q 个子任务作业组成。分别用 T_{il} 和 st_{ijl} 表示任务 T_i 和子任务 st_{ij} 在第 l 个周期内的作业。T_{il} 的到达时间 $A_{il}=(l-1)\times P_i$，绝对截止期限 $d_{il}=A_{il}+D_i$。

1.2 系统模型

如图2所示，硬件系统为一个嵌入 m 个CPU的部分可重构FPGA。m 个CPU通过总线连在一起，形成对称式结构，共享主存。图2中的模块对应于2.1节中的硬件子任务，受CPU控制，可动态地连接到总线上。在时间上顺序执行的模块可重叠配置，以节约硬件资源。

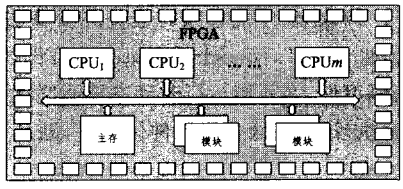

图2 硬件目标平台结构

2 硬件任务分组与放置

2.1 任务分组

按照子任务的约束关系，把可串行执行的硬件子任务分成一组，重叠配置到FPGA中的同一个区域，以提高资源利用率。分组算法描述如下：

```
Partition(Ti)
{
 while(Ti 存在硬件子任务)
 {
  MaxTask=面积最大的子任务;
  CurTask= MaxTask;
  while(CurTask 存在硬件子任务前驱)
  {
   Pred=CurTask 的面积最大前驱;
   Group=Group∪Path(Pred,CurTask);
   CurTask=PredTask;
  }
  CurTask= MaxTask;
  while(CurTask 存在硬件子任务后继)
  {
   Succ= CurTask 的面积最大后继;
   Group=Group∪Path(CurTask,Succ);
   CurTask= SuccTask;
  }
  从 Ti 中删除 Group 中的子任务;
  Group=Φ;
 }
}
```

其中，Path(SubTask1，SubTask2)表示从SubTask1到SubTask2的所有路径中面积最大的路径上的硬件子任务集合。算法每循环一次，分出一组硬件子任务，直到没有硬件子任务为止。

2.2 任务放置

由于同一组中的硬件子任务被配置到FPGA中的同一个区域，共享一个系统总线接口，因此每个硬件子任务必须动态地配置并连接到总线接口上。任务放置的目的就是确定硬件子任务的放置位置并把硬件子任务动态地连接到总线接口上。

CPU和系统总线等系统部分综合成一个配置文件，在系统运行前首先配置到FPGA上，每个硬件子任务综合成一个配置文件，在系统运行时动态配置。现在的EDA工具可以在用户约束文件中添加用户约束以指导综合器的综合过程，同时提供了手工布局布线工具，如Xilinx ISE中的FPGA Editor。在综合CPU和系统总线时，通过位置约束，固定总线接口的端口位置，通过区域约束为硬件子任务预留硬件资源。在综合硬件子任务时，通过区域约束，使得每组硬件子任务综合到同一个预留的区域内。在硬件子任务布局布线完成后，还需要使用手工布线工具来删除相应的FPGA输入输出管脚，同时把输入输出端口连到总线接口的输出输入端口上，从而在配置子任务时能够动态地连在总线上。

3 任务可调度性分析

如果任务 T_k 的每条路径上的子任务均可调度，那么任务 T_k 可调度。假设任务 T_k 的第 j 个作业 T_{kj} 错失了截止期限，那么在区间$[A_{kj},d_{kj}]$内 T_k 至少有一条路径 $fault\text{-}path$ 上的子任务没有完成。图3是 $m=3$ 个处理机时的执行情况，阴影部分表示执行 T_k 的 $fault\text{-}path$ 路径上的子任务，而白色部分表示3个处理机在执行 T_k 其他路径上的子任务和其他任务。

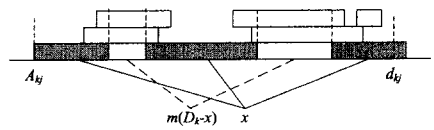

图3 如果没有执行fault-path条路径上的子任务，所有处理机都必定在忙碌状态

定理 1 一组周期任务 $T=\{T_i \mid i=1,2,\cdots,n\}$，并按任务优先级不降顺序排列。如果每个任务 $T_k \in T$ 满足不等式(1)，则 T 可在 m 个处理机上被优先级驱动的算法调度。

$$\sum_{i=1}^{k} \lceil D_k/P_i \rceil C_i^{soft} \leqslant mD_k-(m-1)C_k^{CP} \quad (1)$$

证明：设 *fault-path* 上的子任务执行时间为 C_k^{fp}，根据第 2 节的硬件子任务放置方法，硬件子任务可在其前驱结束时立即执行，所以系统执行 T_k 的其他路径和其他任务的软件子任务的时间总和必定大于等于 $m(D_k-C_k^{fp})$，否则作业 T_{kj} 不会由于路径 fault-path 上的子任务没有完成而错失截止期限。如图 3 所示，用 x 表示 fault-path 路径上的子任务在 $[A_{kj},d_{kj}]$ 内的执行时间总和，那么系统执行 T_k 的其他路径和其他任务的软件子任务的时间总和等于 $m(D_k-x)+x$，否则路径 fault-path 上的子任务在 $[A_{kj},d_{kj}]$ 执行时间总和大于 x。任务 $T_i(1\leqslant i\leqslant k)$ 在区间 $[A_{kj},d_{kj}]$ 内的执行时间至多为 $\lceil D_k/P_i \rceil C_i^{soft}$，所以 $\lceil D_k/P_i \rceil C_i^{soft} \geqslant m(D_k-x)+x = mD_k-(m-1)x$。由于 T_{kj} 错失了截止期限，因此 $x<C_k^{fp}$，所以 $\lceil D_k/P_i \rceil C_i^{soft} > mD_k-(m-1)C_k^{fp} \geqslant mD_k-(m-1)C_k^{CP}$，这与不等式(1)矛盾。所以如果每个任务 $T_k \in T$ 满足不等式(1)，那么 T 可在 m 个处理机上被优先级驱动的算法调度，证毕。

4 SHRTS 算法

在 SHRTS 算法中，子任务是最小的调度单位。就绪的子任务、正在执行的子任务和即将到达的周期任务分别保存在 ReadyList，RunList 和 ArrList 链表中。ReadyList 和 RunList 中的子任务按任务的优先级升序排列，ArrList 中的任务按照到达时间升序排列。Head，Tail 和 Length 分别表示链表的头部元素、尾部元素和元素个数。函数 Del(element)和 Insert(element)分别表示从链表中删除元素 element 和按照链表的原有顺序把元素 element 插入到链表中。

周期任务由定时器触发，并由 Timer_Handler 处理任务到达事件，子任务执行完毕时，调度 JobFinish 函数，处理任务完成事件。

算法包括离线和在线两部分，离线部分包括对硬件子任务分组、放置和判定任务可调度性，在线部分描述如下：

```
Timer_Handler(time)
{
  while(∃ Tij∈ArrList∧Tij. Aij<time)
  {
    if(stientry. type=Hard)
    {
      Load(stientry);
      Run(stientry);
    }
    else {ReadyList. Insert(stientry);}
    ArrList. Del(Tij);
    ArrList. Insert(Ti(j+1));
  }
  SetTimer(ArrList. Head. A);
  Schedule();
}
JobFinish(job)
{
  While(∀ succ∈ISucc(job))
  {
    succ. prednum=succ. prednum-1;
    if(succ. prednum =0)
    {
      if(succ. type=Hard)
      {
        if(succ 未配置)
        {Load(succ);}
        Run(succ);
      }
      else {ReadyList. Insert(succ);}
    }
  }
  if(job. type=Hard)
  {Load(同一组中的后继硬件子任务);}
  else { RunList. Del(job);}
  Schedule();
}
```

其中，SetTimer(time)表示设置时钟的触发时间为 time，Load(hardtask)和 Run(hardtask)分别表示把硬件任务 hardtask 配置到 FPGA 上和运行 hardtask。

```
Schedule()
{
  While(RunList. Length<m)
  {
    ReadyList. Del(ReadyList. Head);
    RunList. Insert(ReadyList. Head);
  }
  while(ReadyList. Head. Prio<RunList. Tail. Prio)
  {
    ReadyList. Del(ReadyList. Head);
    RunList. Insert(ReadyList. Head);
    RunList. Del(RunList. Tail);
    ReadyList. Insert(RunList. Tail);
  }
}
```

用符号 *SubNum* 表示系统中所有子任务的个数，那么链表 ReadyList 和 RunList 的长度最多为 *SubNum* 且是有序的，所以插入元素的时间复杂性为 O(log*SubNum*)。Schedule 内有两个循环，循环次数至多为 m，所以 Schedule 的时间复杂性为 O(m log*SubNum*)。Time_Handler 和 JobFinish 的时间复杂性分别为 $O((n+m)\log SubNum)$ 和 $O((MaxSucc+m)\log SubNum)$，其中 Max*Succ* 为直接后继子任务个数的最大值。所以 SHRTS 算法的在线部分的时间复杂性为 O((Max*Succ*+n+3m) log*SubNum*)。

5 实验结果

周期任务 T_i 的周期 P_i 在[100,500]内均匀分布，相对截止期限 $D_i=P_i$，子任务个数 q 在[10,40]内均匀分布，子任务的执行时间 C_{ij} 在(0,0.025P_i]内均匀分布。任意两个子任务 st_{ij} 和 st_{ik}，如果 st_{ik} 不是 st_{ij} 的前驱，那么从 st_{ij} 到 st_{ik} 存在一条有向边的概率为 $P(0\leqslant P\leqslant 1)$。$P$ 值的大小反映了子任务间的并行度，如果 $P=0$，则所有子任务可并行执行，如果 $P=1$，则所有子任务必须串行执行。

定义配置重叠度＝分组前硬件子任务需要的面积/分组

后硬件子任务需要的面积，硬件子任务的配置重叠度越高，FPGA 的利用率就越高。以配置重叠度为测试标准，在 $P=0.5$，软/硬件子任务个数比为 1∶1 的条件下对分组算法进行仿真。结果如图 4 所示，随着子任务个数增加，配置重叠度呈线性增长。

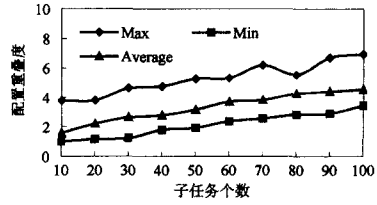

图 4　硬件子任务配置重叠度

假设硬件子任务可配置，对软件子任务的调度做了仿真实验。每次实验随机产生 10 组任务，求出处理机利用率的最小值、最大值和平均值。分别在 $P=0.5$，软/硬件子任务个数比等于 1∶1 和 $P=0.5$，任务个数为 50 的情况下测试了任务个数和硬件子任务所占的比例对算法的影响，结果如图 5 和图 6 所示。从实验结果中可看出随着任务个数的增加和硬件子任务所占的比例增加，处理机利用率略有下降。

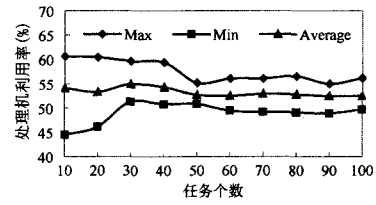

图 5　任务数量对算法的影响

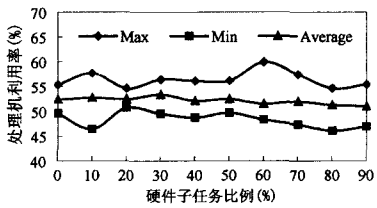

图 6　硬件子任务比例对算法影响

实时多目标遗传算法[2] (Real Time Multiple goals Genetic Algorithm, RT-MGA)和动态延时蚁群系统[4] (Dynamic and Delay Ant Colony System, DDACS)是两种静态智能搜索型算法。实验随机产生一组周期任务，SHRTS 算法计算出所需要的处理机个数，并求出处理机平均利用率；由于一个计划周期内的作业个数太多，RT-MGA 和 DDACS 均无法实现，因此随机选取一组连续到达的作业进行分配，当经过 100 次迭代而没有产生更优的调度结果时，算法结束。实验结果如图 7 所示，随着作业个数增多，RT-MGA 和 DDACS 对处理机的利用率都有明显的下降。

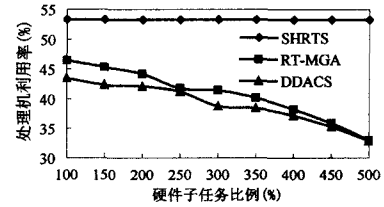

图 7　SHRTS, RT-MGA 和 DDACS 处理机利用率

在以上实验中，所有实时任务均在截止期限内完成，验证了算法的实时性。

结束语　提出了一种能够调度软/硬件混合实时任务的调度算法。算法离线判定实时任务的可调度性，降低了系统的调度开销。通过对硬件子任务分组，每组可重叠配置，有效地利用了 FPGA 资源。在执行前驱软件子任务的同时配置硬件子任务，降低了配置时间对实时任务执行的影响。实验结果表明该算法是有效的。

参 考 文 献

[1] 李仁发，刘彦，徐成. 多处理器片上系统任务调度研究进展评述[J]. 计算机研究与发展，2008，45(9)：1620-1629

[2] OH J，WU C. Genetic-algorithm-based Real-time Task Scheduling with Multiple Goals [J]. Journal of Systems and Software，2004，71(3)：245-258

[3] Chen R M，Lo S T，Huang Y M. Combining Competitive Scheme with Slack Neurons to Solve Real-time Job Scheduling Problem [J]. Expert Systems with Applications，2007，33(1)：75-85

[4] Lo S T，Chem R M，Huang Y M，et al. Multiprocessor System Scheduling with Precedence and Resource Constraints Using an Enhanced ant Colony System [J]. Expert Systems with Applications，2008，34(3)：2071-2081

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

[6] 齐骥，李曦，于海晨，等. 一种面向动态可重构计算的调度算法[J]. 计算机研究与发展，2007，44(8)：1439-1447

[7] 周学功，梁棵，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，2007，30(11)：1901-1909

[8] 梁棵，周学功，王颖，等. 采用预配置策略的可重构混合任务调度算法[J]. 计算机辅助设计与图形学学报，2007，19(5)：635-641

(上接第 238 页)

[80] 刘彬. 基于遗传算法的 NURBS 曲线降阶[J]. 计算机工程，2008：194-196

[81] Lou W P，Chen F L，Chen X Q，et al. Optimal degree reduction of interval polynomials and interval Bézier curves under L^1 norm [C]// The 6th International conference on CAD/CG. Shanghai，China，Vol. 3，Dec. 1999：1010-1015

[82] Chen F L，et al. Degree reduction of interval Bézier curves[J]. Computer-aided design，2000：571-582

[83] Chen F L，Yang X F，Yang W. Degree reduction of interval B-spline curves[J]. 软件学报，2002，14(04)：490-499

[84] 孙红兵，等. 区间 Bézier 曲面的降阶[J]. 中国科学技术大学，2002，32(2)：147-154

[85] Chen F L，et al. Degree reduction of disk Bézier curves[J]. Computer aided geometric design，2004(21)：263-280

[86] 孙红兵，等. 区间有理曲线的降阶[J]. 中国科学技术大学学报，2006，36(9)：968-976

[87] 林新辉. C_B_Bézier 曲线降阶逼近[D]. 杭州：浙江大学，2007

[88] Huang Y，Wang G Z. Constructing a quasi-Legendre basis based on the C-Bézier basis[J]. Progress in Nature Science，2005，15(6)：559-563

[89] Huang Y，Wang G Z. An orthogonal basis for the hyperbolic hybrid polynomial space[J]. Sci China Ser F-Inf Sci，2007，150(1)：21-28