第44卷第4期 中国科学技术大学学报 Vol. 44, No. 4
2014年4月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Apr. 2014

文章编号:0253-2778(2014)04-0325-08

一种基于顶点位置树的可重构软硬件迭代协同算法

李春生[1,2],周学海[1],曾芳玲[2],王 超[1]

(1.中国科学技术大学计算机科学与技术学院,安徽合肥 230027;
2.电子工程学院电磁制约重点实验室,安徽合肥 230037)

摘要:可重构计算分时复用有限的面积资源,实现更多的任务硬件加速运行,同时也给传统的软硬件协同设计带来了新的挑战.为此设计了一种基于顶点位置树的迭代协同ICS-VPT算法针对离线型、集中共享式可重构计算平台,综合软硬件划分、硬件布局和任务调度,提升系统性能:首次提出顶点位置树的数据结构,以较小的存储空间快速查找布局位置;迭代协同算法根据数据依赖图分组任务,结合通信代价获取软/硬件任务的优先级,进行合理划分和调度.实验结果表明,ICS-VPT算法在高效管理可重构资源和灵活处理通信代价的同时,保持了较低的系统运行时间.

关键词:可重构计算;顶点位置树;软硬件划分;迭代调度;硬件布局

中图分类号:TP302 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2014.04.010

引用格式:Li Chunsheng, Zhou Xuehai, Zeng Fangling, et al. A reconfigurable hardware-software iteration co-synthesis algorithm based on vertex position tree[J]. Journal of University of Science and Technology of China, 2014,44(4):325-332.

李春生,周学海,曾芳玲,等. 一种基于顶点位置树的可重构软硬件迭代协同算法[J]. 中国科学技术大学学报,2014,44(4):325-332.

A reconfigurable hardware-software iteration co-synthesis algorithm based on vertex position tree

LI Chunsheng[1,2], ZHOU Xuehai[1], ZENG Fangling[2], WANG Chao[1]

(1. *School of Computer Science, University of Science and Technology of China, Hefei* 230027, *China*;
2. *Key Laboratory of Electric Restriction, Electronic Engineering Institute, Hefei* 230037, *China*)

Abstract: In reconfigurable computing, more tasks can be executed at a higher speed in hardware by time multiplexing with the limited area resources. Meanwhile, it also brings new challenges to the traditional hardware-software codesign. For offline scheduling, centralized shared structure reconfigurable platform, the ICS-VPT (iteration co-synthesis based on vertex position tree) algorithm synthesized hardware-software partitioning, hardware placement and task scheduling to improve system performance: The VPT (vertex position tree) data structure was first proposed, which could find a placement position quickly with small storage space; The ICS (iteration co-synthesis) algorithm grouped tasks according to data dependence graph and obtained the hardware/software tasks' priorities by combining the communication cost, thus obtaining a reasonable partitioning and scheduling. Experimental results show that the ICS-

收稿日期:2013-04-15;**修回日期**:2013-09-28

基金项目:国家自然科学基金(61202053),江苏省自然科学基金(BK2012194)资助.

作者简介:李春生,男,1982年生,博士生. 研究方向:计算机系统结构. E-mail: cslee@mail.ustc.edu.cn

通讯作者:周学海,博士/教授. E-mail: xhzhou@ustc.edu.cn

VPT algorithm maintains a lower level of system running time by means of efficient reconfigurable resource management and flexible communication cost handling.

Key words: reconfigurable computing; vertex position tree; hardware-software partitioning; iteration scheduling; hardware placement

0 引言

可重构计算(reconfigurable computing,RC)将计算问题从微处理器(时间域)迁移到面向应用的硬件电路(空间域)上,弥补了通用计算和专用计算之间的空白,从而提高了计算效率和系统性能.可重构计算原型系统最早可追溯到20世纪60年代加州大学洛杉矶分校Estrin提出并设计实现的F+V(fixed plus variable)体系结构及其软件系统[1],但直到20世纪80年代中期,随着FPGA技术的逐渐成熟,可重构计算才成为研究的热点问题之一.近几年在"牧村浪潮(Makimoto's Wave)"的验证及预测推动下,可重构计算得到学术界和产业界越来越多的重视,成为嵌入式计算系统的发展趋势之一[2].

可重构计算通过改动若干物理控制点,实现某种形式的可编程硬件系统在不同时间内运行不同的应用,是一个难度大、涉及面广的课题.可重构计算根据任务类型可分为离线型和在线型两种.离线型多为依赖性任务,往往以数据依赖图(data dependence graph,DDG)的形式表示,其任务特征及依赖关系在系统运行前已知,通常可以获得最优解;在线型多为独立任务,在系统运行过程中随机到达,一般只能得到近似最优解.可重构计算底层硬件平台按通信及存储方式可抽象为集中共享式(如总线型)和分布存储式(如片上网络型)两类.前者主要适用于多个功能模块共享存储资源、系统吞吐量适中的应用场景;后者多面向并行计算、路由寻址等大规模数据流动.本文主要讨论基于DDG图离线型、集中共享式的可重构计算平台.

软硬件协同设计(hardware-software codesign)避免了单纯依靠软件(或硬件)设计无法满足开发周期、成本、风险等方面的不足,在优化系统性能、提升运行效率等方面起着全局性的影响.合理有效的软硬件划分与调度是软硬件协同的关键技术,普通被认为是NPC(non-deterministic polynomial complete)问题.可重构计算的引入在提升软硬件协同设计效率的同时也带来了新的挑战.一方面,灵活的可重构协同设计增大了算法的复杂性;另一方面,高效的资源管理策略是实现多任务分时硬件加速运行的关键之一.

近年来,一些研究对传统的软硬件协同算法加以改进,使之适用于可重构计算系统.如降低数据移动的RDMS(reduced data movement scheduling)调度算法[3]着重解决通信代价对可重构系统的影响;基于模拟退火和链式调度的SALSPA(simulated annealing and list scheduling based partitioning algorithm)算法[4]综合了可重构任务的划分、调度和配置预取等技术,平均减少25%左右的程序运行时间.但上述算法均未考虑资源管理和硬件布局对软硬件划分的影响.另外一些研究针对可重构计算的特点,着重解决资源管理及布局碎片问题.资源管理方面,如基于二维资源模型的分组-邻接边在线调度GC(group-contiguous)算法[5]引入硬件任务分组来提高调度成功率.硬件布局方面,比较有代表性的是Stuffing和Horizon两种预约调度算法[5-6].后续的研究还包括基于SUR(space utilization rate)分组的Classified Stuffing算法[7]、基于时间窗口的Window-based Stuffing算法[8]和基于位置调整的Intelligent Stuffing算法[9]等.碎片度量方面,典型的代价函数有占用区域连续度(degree of occupied area concatenation,DAOC)[10]和基于相对正交Q值的碎片度量标准[11].这些研究有的只针对在线布局问题,有的弱化了其他限制条件约束,均未能很好地与软硬件协同流程相结合.

结合上述研究的优缺点,本文提出一种基于顶点位置树的软硬件迭代协同(iteration co-synthesis based on vertex position tree,ICS-VPT)算法.ICS-VPT算法通过高效划分DDG图中的任务结点,迭代调度软硬件任务的运行顺序及灵活的资源管理策略,着重解决离线可重构软硬件协同过程中的通信代价和布局碎片等限制条件对系统性能的影响,以达到最大化硬件加速比、最小化系统运行时间(调度长度)的目的.

在资源管理方面,本文首次提出顶点位置树VPT的数据结构,实现资源的高效管理.利用VPT不同结点类型,快速准确查找适合的布局区域;在布

局碎片方面，借助碎片外形边（fragment shape edges，FSE）评价函数，减少系统碎片的产生，确保硬件布局位置的合理性；在通信代价方面，迭代分组 DDG 图中的任务结点，结合不同属性任务间通信成本赋予软/硬件运行优先级，合理划分任务属性，有效降低调度长度.

1　系统模型

1.1　抽象结构

图 1 是基于总线型、面向离线 DDG 图的可重构计算平台抽象结构. 该平台是一个包含了微控制器、数字信号处理器（digital signal processor，DSP）、专用集成电路（application specific integrated circuit，ASIC）、可重构硬件等运算、控制单元在内的异构多核平台. 每个单元模块都有各自对应的本地缓存，并通过总线共享全局存储. 微控制器对输入的 DDG 图进行合理的软硬件划分（划分器），调度软硬件任务高效有序运行（调度器），依据可重构资源管理信息为硬件任务选取合适的布局位置（布局器），并控制通信代价、布局碎片等额外开销，以获得较大硬件加速比和较小系统运行时间.

硬件任务通常以矩形形状放置于可重构二维面积资源上，分为一维重构和二维重构两种[12]，前者要求每个硬件任务独占完整的一列，后者则允许其他硬件任务占用同一列中未使用的面积. 无论采用何种重构方式，均不允许硬件任务在水平和垂直方向上有重叠. 显然，二维重构较一维重构更加灵活，本文主要讨论二维重构.

1.2　数据依赖图模型

数据依赖图 DDG 是一个有向无环图（directed acyclic graph，DAG），表示如下：

$$G=\{V\mid(w,h,Hw,Sw),E\mid Cm\}.$$

式中，V 是 DDG 图中所有结点的集合，每个结点代表一个粗粒度的任务. (w,h,Hw,Sw) 分别代表任务矩形的宽度、高度，任务的硬件运行时间和软件运行时间；E 是 DDG 图中边的集合，边的权值 Cm 代表数据从一个任务迁移到另一个任务所需的通信时间. 不同处理单元都对应有本地缓存（图 1），因此同一属性任务间的通信代价忽略不计.

2　可重构布局算法

2.1　VPT 定义

利用平面直角坐标系表示硬件任务在可重构区域内的位置信息，如图 2(a)所示. 最大矩形区域 (W,H) 反映该可重构面积资源总容量；任务 $V(w,h)$ 布局后，附加其右上角坐标值 (x,y)，（vertex position，VP）$VP=(x,y,w,h)$，即可唯一确定该硬件任务的位置信息.

时刻 t 在获取任务布局位置 $VP=(x,y,w,h)$ 的同时，附加其左上 $(x-w,y)$ 和右下 $(x,y-h)$ 两个位置坐标作为该任务结点的子结点，就得到一棵顶点位置树（vertex position tree，VPT），如图 2(b)所示.

顶点位置树 VPT 反映了时刻 t 硬件任务布局信息，共分为三类结点：占用结点、自由结点和终端结点. 在时刻 t，每个已布局任务 V_i 唯一对应 VTP 中某个占用结点（vertex occupied，VO）$VO_i(x_i,y_i,w_i,h_i)$，

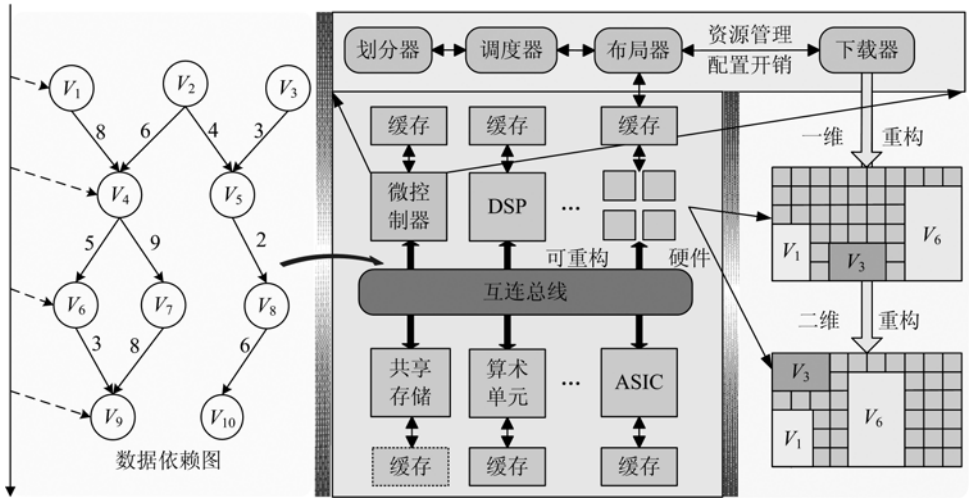
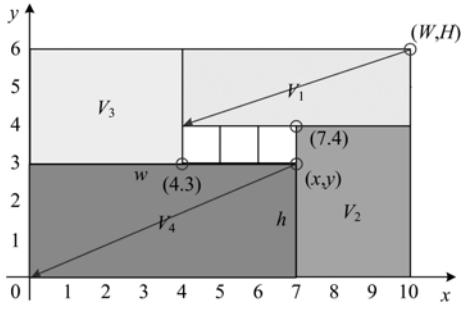


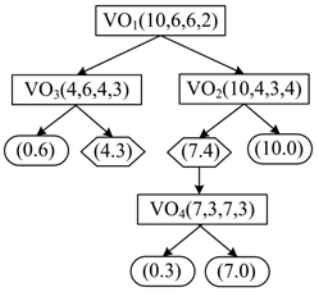
图 1　可重构平台抽象结构

Fig. 1　Abstract structure of reconfigurable platform

且每个占用结点有且仅有左子结点(x_i-w_i, y_i)和右子结点(x_i, y_i-h_i). 占用结点的左右子结点既可是其他任务对应的占用结点(如图 2(b)中任务 V_1 的两个子结点分别对应任务 V_2 和 V_3),又可以是自由结点和终端结点.



(a) 二维阵列



(b) 顶点位置树

图 2 可重构二维面积阵列及顶点位置树

Fig. 2 Reconfigurable 2-D area array & vertex position tree

终端结点是一类某个或全部坐标值为 0 的特殊结点,以(0. y)/(x. 0)表示,不拥有任何子结点(如图 2(b)中任务 V_3 的左子结点). 自由结点是占用结点的子结点中既非占用结点又非终端结点的一类结点,以(fx. fy)表示,其子结点为空结点(如图 2(b)中任务 V_3 的右子结点),或是满足($x_i \leqslant fx \vee y_i < fy$) $\wedge$ ($x_i < fx \vee y_i \leqslant fy$)的某个占用结点 $VO_i(x_i, y_i, w_i, h_i)$(如图 2(b)中任务 V_4 即为自由结点(7.4)的子结点),自由结点可以有多个占用子结点,以 x_i 升序排列.

2.2 VPT 操作

中国工匠在几千年的工艺中遵循着“金角银边草肚皮”的堆砌原则,即在货物堆放过程中,尽可能从最里端进行“占角”动作[13]. 文献[13]对 21 个算例的对比实验(填充总面积、面积利用率、运行时间及最优解)表明,改进的穴度算法无论在计算的优度还是计算的速度上都有明显的优势. VPT 的初始化、插入、删除等操作即来源于这一思想.

VPT 的初始化对应于空闲矩形右上角(W, H)放置第一个硬件任务,即根结点 V_r(W, H, w_r, h_r). VPT 的插入采用“占角”动作放置后续任务,角的位置对应于 VPT 中占用结点的自由子结点,后续硬件任务将某个自由结点改变为占用结点,并附加其两个子结点. 若放置于自由结点(fx. fy)的硬件任务与已布局任务发生(d_x, d_y)的“重叠”差值时,选取规避该差值的新坐标($fx'=fx-d_x$. $fy'=fy-d_y$)作为(fx. fy)的子结点,如图 2(b)中的 V_4. VPT 的删除对应于已完成硬件任务释放其所占面积资源,即任务所对应的占用结点重新改变为自由结点,在删除其自由子结点和终端子结点、保留其占用子结点的同时,将后继的子结点依次前移. 图 3 显示了任务 V_2 完成后,顶点位置树的更新情况.

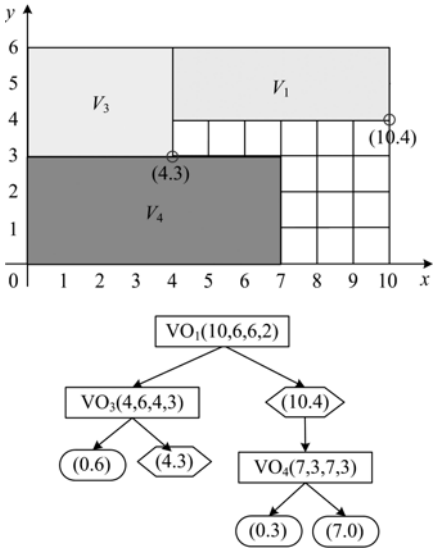


图 3 顶点位置树的删除操作

Fig. 3 Deletion of vertex position tree

2.3 AT 判定准则

以 VPT 中自由结点构成的角为“可能角位置”,如图 2 中的自由结点(7.4)仅适合面积在 3×1 以内的矩形任务;图 3 中的自由结点(4.3)则不适合任何矩形任务. AT(accept task)判定准则从可能角位置中选择适合矩形任务放置的“有效角位置”.

对于单个任务 V(x, y, w, h)或任意两个任务 $V_i(x_i, y_i, w_i, h_i)$ 和 $V_j(x_j, y_j, w_j, h_j)$,($i \neq j$),$x_l = x-w$, $x_r = x$, $y_l = y-h$ 及 $y_h = y$ 分别代表任务布局后水平 left, right 及垂直 low, high 的坐标值. AT 判定准则表示为

$$\mathrm{AT}\begin{cases} ①(x_l = x_r - w) \wedge (y_l = y_h - h) \wedge \\ \quad (1 \leqslant x_l < x_r \leqslant \mathrm{W}) \wedge (1 \leqslant y_l < y_h \leqslant \mathrm{W}). \\ ②\max[x_{li} - x_{rj}, x_{lj} - x_{ri}, y_{li} - y_{hj}, y_{lj} - y_{hi}] \geqslant 0 \end{cases}$$

2.4　FSE 判定准则

经 AT 准则判定后，有效角位置数目可能不止一个，进一步定义碎片外形边数(fragment shape edges，FSE)为

$$FSE = \{e \mid e \in \text{Border of } \Gamma,\ \Gamma = \bigcup F_T\}.$$

式中，Γ 是所有碎片所形成的碎片集，碎片集的外部边数总和即为 FSE 的值. 合适的布局位置选取在每个有效角位置预布局后具有最小 FSE 值的坐标. 如图 4 所示，左右两种硬件任务布局都占据相同数量的总面积，左边(FSE=18)的布局效率却明显优于右边(FSE=36).

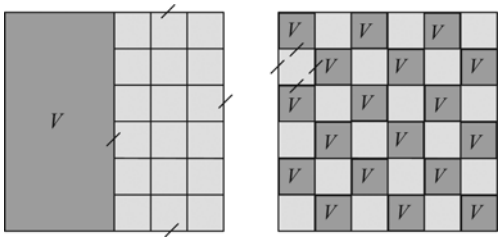


图 4　相同硬件面积下的 FSE 对比值

Fig. 4　Different FSE values with same hardware occupied areas

算法 2.1　计算任务的布局信息

输入：任务 V(w,h)

输出：任务的布局信息，返回 VO(x, y, w, h)或 false.

①从顶点位置树 VPT 的根结点出发，按广度优先搜索的顺序查找其所有自由结点，并插入到 FIFO 队列 Q 中. 初始化 Temp(x, y, FSE, Son)=(0,0,∞,0)；

②执行以下步骤，直到队列 Q 为空时，执行步骤⑧；

③令队列 Q 中自由结点(fx_i. fy_i)依次出队，判断($w \leqslant fx_i$) ∧ ($h \leqslant fy_i$)，满足则执行步骤④，不满足则删除该结点，返回步骤②；

④令 VP=(fx_i, fy_i, w, h)，判断是否满足 AT，满足则执行步骤⑦，若不满足，即与某个任务 V_j(x_j, y_j, w_j, h_j)发生重叠，则计算其差值(d_x, d_y)，执行步骤⑤；

⑤令 $fx_i' = fx_i - d_x$, $fy_i' = fy_i - d_y$，再次断定是否满足 AT，首次满足时则执行步骤⑥，若不满足，更新 d_x, d_y 重复进行 AT 判定，直到 $fx_i' = w$, $fy_i' = h$，若仍不满足，则删除该自由结点，返回步骤②；

⑥令 VP=(fx_i', fy_i', w, h)，计算其 FSE 值，小于 Temp 时，令 Temp=(fx_i', fy_i', FSE, 1)，跳转到⑧，大于 Temp 时删除该结点，返回步骤②；

⑦令 VP=(fx_i, fy_i, w, h)，计算其 FSE 值，小于 Temp 时，令 Temp=(fx_i, fy_i, FSE, 0)，大于 Temp 时删除该结点，返回步骤②；

⑧若 Temp(FSE)=∞，返回 false，否则跳转到⑨；

⑨若 Son=0，把自由结点(fx. fy)改为占用结点 VO($x=fx$, $y=fy$, w, h)，若 Son=1，把占用结点 VO($x=fx'$, $y=fy'$, w, h)作为自由结点(fx. fy)的子结点. 同时增加 VO 的两个子结点($x-w$, y)，(x, $y-h$)，返回 VO(x, y, w, h).

算法 2.2　计算任务的结束信息

输入：任务 V(w,h)和结束时间 t

输出：任务的结束信息，返回更新后的 VPT.

①等待任务 V(w,h)的结束时间 t；

②任务从占用结点 VO(x, y, w, h)改变为自由结点($fx=x$. $fy=y$)；

③若该自由结点(fx. fy)的子结点为占用结点 VO(x_i, y_i, w, h)，则保持不变；

④若该自由结点(fx. fy)的子结点为自由结点或终端结点(fx_i. fy_i)，则删除(fx_i. fy_i)，并将(fx_i. fy_i)的子结点上移为自由结点(fx. fy)的子结点；

⑤返回更新后的 VPT.

3　动态软硬件协同综合算法

3.1　任务间通信代价

DDG 图中任务最早开始运行时间取决于其所有父结点的结束时间及不同属性父结点间通信时间. 如图 5 所示，任务结点 V 有 m 个父结点 S_1，…，S_m 软件运行，n 个父结点 H_1，…，H_n 硬件运行，各个父结点任务完成时间分别为 f_{S_1}，…，f_{S_m} 和 f_{H_1}，…，f_{H_n}，到结点 V 的通信时间分别为 C_{SV1}，…，C_{SVm} 和 C_{HV1}，…，C_{HVn}.

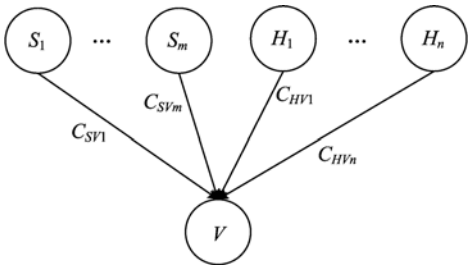


图 5　结点通信示意图

Fig. 5　Communication of nodes

定义任务 V 的最早开始软/硬件运行的时间分别为

$$VS_{Hw} = \max\{T_{Cm}, \max(f_{H_j})\},$$

$$i=1,\cdots,m,j=1,\cdots,n;$$

$$VS_{Sw}=\max\{\max(f_{S_j}),T_{Cn}\},$$

$$i=1,\cdots,m,j=1,\cdots,n.$$

设 k 个任务的完成时间分别是 $f_1,\cdots,f_k$，不妨设 $f_1\leqslant f_2\leqslant\cdots\leqslant f_k$，它们到共同子结点的通信时间分别为 $C_1,\cdots,C_k$；同时假定任务完成后立即将数据放置在总线上顺序传输. 则 T_{Ck}（上式中的 m 和 n 统一表示为 k）定义为

$$T_{Ck}=f_i+C_i+\cdots+C_k,$$

$$f_j+C_j+\cdots+C_{i-1}\leqslant f_i,1\leqslant j<i\leqslant k.$$

以图 6(a) 中三个任务为例，f_2 和 C_1 之间有重叠，故 C_2 紧接着 C_1 进行传输，同理 C_3 紧接着 C_2 进行传输，故总时间为 $\sum=f_1+C_1+C_2+C_3$；图 6(b) 中，f_2 和 C_1 之间不存在重叠，C_2 在 f_2 后进行传输，总时间为 $\sum=f_2+C_2+C_3$. 由此推广到一般.

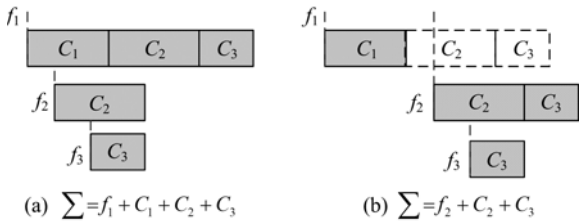


图 6 通信时间的证明和分析

Fig. 6 Proof and analysis of the communication time

综上，DDG 图中结点 V 的软/硬件完成时间表示为

$$f_{Hw}=VS_{Hw}+Hw;$$

$$f_{Sw}=VS_{Sw}+Sw.$$

3.2 任务调度优先级

当同时调度 m 个任务 $V_1,\cdots,V_m$ 时，定义任务调度的软/硬件优先级为

$$\text{Priority}(V_i)=\begin{cases}Hw\_\text{priority}(V_i)=f_{Sw}\\ Sw\_\text{priority}(V_i)=(f_{Hw}-f_{Sw})>0\end{cases}.$$

任务的硬件优先级选取最大软件完成时间；软件优先级选取硬件完成时间和软件完成时间的差值，当且仅当该任务的软件优先级为正值时，才考虑任务软件运行.

3.3 软硬件迭代划分

数据依赖图 DDG 中的任务间存在时序制约关系，采用迭代比较的软硬件划分方法，遵循以下准则：

（Ⅰ）初始化. 比较 DDG 图中所有的根结点，将完成软硬件划分的任务置于划分集，余下任务设为不定态；

（Ⅱ）动态更新. 删除划分集中任务，迭代放置仅依赖于该划分集的子结点任务，并重新和余下不定态任务进行比较；

（Ⅲ）循环迭代. 依次进行迭代比较，直到所有任务完成划分.

任务的软硬件划分、迭代调度和硬件布局相辅相成，同时考虑通信代价和布局碎片对系统性能的影响. 综合的软硬件协同迭代算法如下：

算法 3.1 软硬件协同迭代算法

输入：任务图 DDG

输出：甘特图及系统运行时间

①初始化，设 $i,m,n=0$；将 DDG 图中所有根结点置于比较组.

②计算比较组中每个任务的 f_{Hw} 和 f_{Sw}，$i++$；

③第 i 轮比较，按 $(f_{Hw}-f_{Sw})>0$ 逆序依次选取任务，划分为软件任务 $Sw-n$，$n++$，置于划分集. 在甘特图中标明软件任务运行次序、时间及与父结点硬件任务的通信时间；

④对本组余下的任务按 f_{Sw} 逆序选取，依次进行硬件布局判断（算法 2.1）：返回 false 时，设置为不定态；返回 VO(x,y,w,h) 时，划分为硬件任务 $Hw-m$，$m++$，置于划分集，并维护 VPT 树. 在甘特图中标明硬件任务运行次序、时间及与父结点软件任务的通信时间；

⑤标记划分集中硬件任务的完成时间，据此更新 VPT 树（算法 2.2）；

⑥删除划分集中任务，将仅依赖上述任务的子结点放置于比较组，跳转至步骤②；

⑦重复执行上述步骤，直至比较组中任务为空，返回甘特图和运行总时间.

4 示例验证

数据依赖图 DDG 见图 1 左，表 1 给出其 10 个任务的参数属性，同时定义可重构硬件的总面积为 10×6.

表 1 应用实例参数表

Tab. 1 Parameters table of illustrate example

参数	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
w	6	3	4	7	4	3	8	2	3	5
h	2	4	3	3	4	6	5	6	5	4
Sw	20	18	16	29	23	26	36	21	22	28
Hw	7	6	5	12	9	10	15	8	9	14

算法运行过程如表 2 所示. 经过 6 轮比较后，DDG 图中的 10 个任务被划分为 9 个硬件任务和 1 个软件任务，系统运行总时间为 59.

表 2　算法运行过程及结果

Tab. 2　The process and result of algorithm running

C	DDG	VS_{Hw}	Hw	f_{Hw}	VS_{Sw}	Sw	f_{Sw}	P_{Hw}	P_{Sw}	Result
	V_1	0	7	**7**	0	20	20	20	—	$Hw-1$
1	V_2	0	6	**6**	0	18	18	18	—	$Hw-2$
	V_3	0	5	**5**	0	16	16	16	—	$Hw-3$
2	V_4	7	12	**19**	20	29	49	49	—	$Hw-4$
	V_5	6	9	15	12	23	35	35	—	—
	V_5	19	9	28	12	23	35	35	—	—
3	V_6	19	10	29	24	26	50	50	—	—
	V_7	19	15	**34**	28	36	64	64	—	$Hw-5$
4	V_5	34	9	43	12	23	**35**	—	8	$Sw-1$
	V_6	34	10	**44**	24	26	50	50	—	$Hw-6$
5	V_8	37	8	**45**	35	21	56	56	—	$Hw-8$
	V_9	44	9	**53**	47	22	69	69	—	$Hw-7$
6	V_{10}	45	14	**59**	51	28	79	79	—	$Hw-9$

调度结果以甘特图(图 7)形式表示.

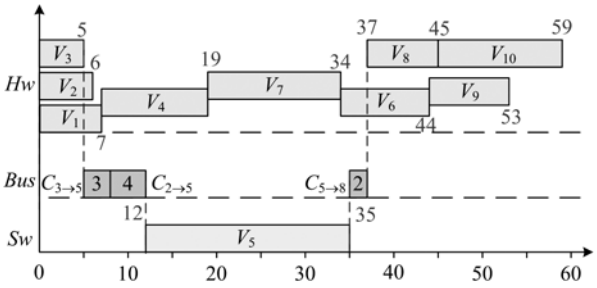


图 7　调度结果-甘特图

Fig. 7　Scheduling Result-Gantt chart

5　实验分析

为验证 ICS-VPT 算法的有效性，采用改进的 RGBOS (random graphs with branch-and-bound obtained optimal solutions)测试基准[14]. 改进后的 RGBOS 测试基准包括 6 个不同通信计算比(communication to computation ration，CCR)的子集，分别为 CCR=0.2，0.5，0.8，1.0，1.5 和 2.0，符合总线型结构通信量适中的情形. 任务的软硬件运行时间和任务间通信时间依据 CCR 约束随机产生，每类 CCR 取 30 次实验的平均值. 可重构硬件总面积分别定义为 10×6 和 10×10，硬件任务矩形的长度在[2，10]、高度在[2，6]间均匀分布.

对比算法采用本文提出的 ICS-VPT 算法、改进的 CPCS-VPT 算法[15]和基于分组-邻接边[5]的迭代协同 ICS-GC 算法. 改进的 CPCS-VPT 算法首先将通信代价折合到软/硬件运行时间 Hw' 和 Sw' 中去，再进一步计算软硬件划分的收益函数 b_V，其硬件任务在可重构面积区域上的布局仍采用 VPT 算法；ICS-GC 算法通过对硬件任务进行分组来减少布局碎片的产生，其软硬件协同的过程仍采用迭代协同算法. 三类算法的对比结果如图 8 所示.

总体上，系统调度长度(运行时间)随 CCR 的增长呈上升趋势，随可重构总面积的增加呈下降趋势. 三种对比算法中，ICS-VPT 算法充分利用 VPT 树状结构高效管理有限的可重构资源，通过优先级划分和迭代调度有效降低通信代价，因此保持较低的系统调度长度. CPCS-VPT 算法将通信代价折合到收益函数进行软硬件划分和调度，造成部分任务软硬件交替运行，从而增加了任务间的通信时间，系统调度长度相对较高. ICS-GC 算法在硬件任务布局中采用分组的方法，当可重构总面积较小时，会造成较多的布局碎片，从而增加了系统调度长度，因此其性能在图 8(a)中较接近于 CPCS-VPT 算法；随着可重构总面积的增加，硬件任务布局限制逐渐宽松，其

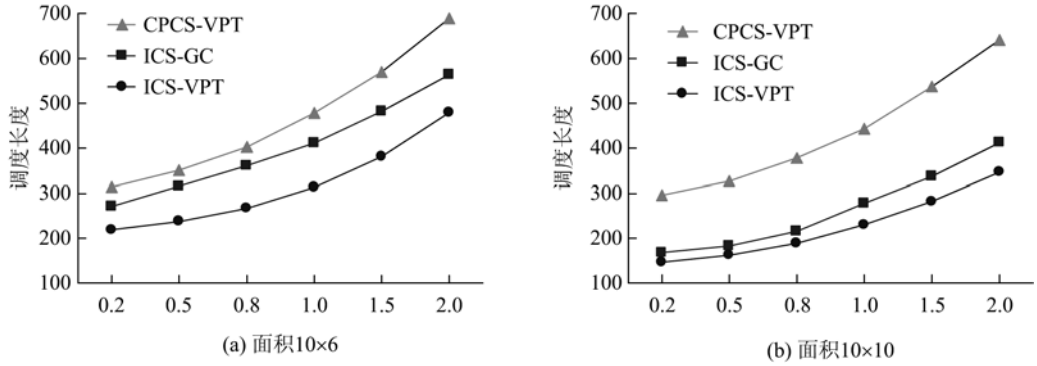


图 8　实验对比结果

Fig. 8　Experimental comparison results

性能则趋向于 ICS-VPT 算法(图 8(b)).

ICS-VPT 算法的时间复杂度分析如下:设 DDG 图中共有 N 个任务结点,经过 k 轮迭代比较后,划分为 N_{Sw} 个软件任务和 N_{Hw} 个硬件任务.每轮比较 n_k 个任务结点,其中有 n_{kSw} 个结点参加软件优先级比较,n_{kHw} 个结点参加硬件优先级比较.同时,每个硬件结点在选择布局位置时,VPT 树中已有 m_k 个已布局硬件顶点.则系统的时间复杂度计算为

$$\sum_{i=1}^{k}(n_{iSw}+n_{iHw}\times m_i)=(n_{1Sw}+\cdots+n_{kSw})+(n_{1Hw}\times m_1+\cdots+n_{kSw}\times m_k)\approx pN_{Sw}+qN_{Hw}^2\approx O(N^2).$$

ICS-VPT 算法的空间复杂度包括 VPT 树的存储空间和迭代比较时结点的存储空间.VPT 树有 m_k 个已布局硬件顶点和最多 $2m_k$ 个子结点,迭代比较组需要存储 n_k 个任务结点,故空间复杂度为 $O(n_k+3m_k)=O(N)$.

6 结论

可重构平台较高的系统灵活性和计算效率是通过分时复用有限的面积资源换取的,因此,有效降低系统运行时间是提升可重构计算平台系统性能的关键.ICS-VPT 算法对基于离线型、集中共享式可重构计算平台进行了有益的探索,以 $O(N^2)$ 的时间复杂度和 $O(N)$ 的空间复杂度,维持较低的系统运行时间开销.

参考文献(References)

[1] Estrin G, Bussell B, Turn R, et al. Parallel processing in a restructurable computer system [J]. IEEE Transactions on Electronic Computers, 1963, EC-12(6): 747-755.

[2] Salvadeo P A, Veca A C, Lopez R C. Historic behavior of the electronic technology: The wave of Makimoto and Moore's Law in the transistor's age [C]// 8th Conference on Programmable. Bento Gonçalves, Brazil: IEEE Press, 2012: 1-5.

[3] Huang M Q, Narayana V K, Simmler H, et al. Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing[J]. ACM Transactions on Reconfigurable Technology Systems, 2010, 3(4): 1-24.

[4] Shen Yingzhe, Zhou Xuehai. A hardware-software partitioning algorithm for reconfigurable computing systems [J]. Journal of University of Science and Technology of China, 2009, 39(2): 182-188.

沈英哲,周学海.一种用于可重构计算系统的软硬件划分算法[J].中国科学技术大学学报,2009,39(2):182-188.

[5] 刘彦, 李仁发, 许新达, 等. 一种异构可重构片上系统的实时任务调度算法[J]. 计算机研究与发展, 2010, 47(6): 1 116-1 124.

[6] Steiger C, Walder H, Platzner M, et al. Online scheduling and placement of real-time tasks to partially reconfigurable devices[C]// Proceedings of the 24th International Real-Time Systems Symposium. Cancun, Mexico: IEEE Press, 2003: 224-225.

[7] Chen Y H, Hsiung P A. Hardware task scheduling and placement in operating systems for dynamically reconfigurable SoC[C]// Embedded and Ubiquitous Computing, Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005: 489-498.

[8] Zhou X G, Wang Y, Huang X Z, et al. On-line scheduling of real-time tasks for reconfigurable computing system[C]// International Conference on Field Programmable Technology. Bangkok, Thailand: IEEE Press, 2006: 57-64.

[9] Marconi T, Lu Y, Bertelsm K, et al. Online hardware task scheduling and placement algorithm on partially reconfigurable devices[C]// Proceedings of the 4th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications. London, UK: IEEE Press, 2008: 306-311.

[10] 齐骥,李曦,胡楠,等.基于硬件任务顶点的可重构系统资源管理算法[J].电子学报,2006,34(11):2 094-2 098.

[11] Septién J, Mozos D, Mecha H, et al. Perimeter quadrature-based metric for estimating FPGA fragmentation in 2D HW multitasking[C]// 22nd IEEE International Symposium on Parallel and Distributed Processing. Miami, USA: IEEE Press, 2008:1-8.

[12] Cordone R, Redaelli F, Redaelli M A, et al. Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGAs [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2009, 28(5): 662-675.

[13] 何琨, 黄文奇, 金燕. 基于动作空间求解二维矩形 Packing 问题的高效算法[J]. 软件学报, 2012, 23(5): 1 037-1 044.

[14] Kwok Y K, Ahmad I. Benchmarking the task graph scheduling algorithms[C]// Proceedings of the First Merged International Patallel Processing Symposium. Orlando, USA: IEEE Press, 1998: 531-537.

[15] Wu J G, Srikanthan T, Jiao T. Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design [J]. Design Automation for Embedded Systems, 2008, 12(4): 345-375.