高效软硬件划分算法及其提升技术

王 璞 武继刚
(天津工业大学计算机科学与软件学院 天津 300387)
(中国科学院软件所计算机科学国家重点实验室 北京 100190)

摘 要 软硬件划分是软硬件协同设计的关键环节,它决定系统中哪些组件由软件实现,哪些由硬件实现。软硬件划分问题已被证明是NP完全问题。将一类软硬件划分问题看作变异的0-1背包问题,在求解背包问题的算法基础上构造出软硬件划分问题的优质启发解。此外,采用禁忌搜索(Tabu Search)算法对求得的启发解进行改进,在软件开销和通信开销满足一定约束的条件下,使得硬件开销尽可能小。实验结果证明,所提算法对当前最新算法的改进最大可达到28%。

关键词 软硬件划分,启发式算法,0-1背包问题,禁忌搜索
中图法分类号 TP3-0 **文献标识码** A

Efficient Heuristic and Tabu Search for Hardware/Software Partitioning

WANG Pu WU Ji-gang
(School of Computer Science and Software, Tianjin Polytechnic University, Tianjin 300387, China)
(State Key Laboratory of Computer Science, The Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract Hardware/software(HW/SW) partitioning is one of the crucial steps in HW/SW co-design. It determines which componented of the system are implemented on hardware and which ones on software. It has been proved that the HW/SW partitioning problem is NP-hard. This paper presented an heuristic algorithm for the HW/SW partitioning problem, which has been treated as an extended 0-1 knapsack problem. Tabu search was used to further the solution obtained through the proposed heuristic algorithm, in order to minimize the hardware cost with the constraints of the software cost and the communication cost. Experimental results show that the algorithms proposed in the paper can produce better solution than the latest work, and the improvement is up to 28%.

Keywords Hardware/Software partitioning, Heuristic algorithm, 0-1 knapsack problem, Tabu search

1 引言

嵌入式系统被广泛应用于工作、学习、生活等各个领域,标志着人们对高科技产品的集成度与计算能力的要求越来越高。软硬件协同设计已成为嵌入式系统开发的主流技术,大大缩短了系统的研发周期。软硬件划分是软硬件协同设计的关键环节,对提高系统性能起着至关重要的作用。众所周知,硬件执行速度快但成本较高,而软件执行速度慢但生产与维护成本较低,因此,考虑将关键性能部件由硬件实现,其他的由软件来实现,这就是软硬件划分技术的目的所在[1]。

近年来,国内外学者在软硬件划分问题方面已经取得了显著成果,已有的软硬件划分算法包括精确算法和启发式算法两类。精确算法有动态规划算法[2,3]、整数线性规划算法[4,5]和分支限界法[6]等,这些算法一般用于较小规模的划分问题。当问题规模较大时,精确算法变得不可行,因而启发式算法成为可行的选择。

传统的启发式算法包括面向硬件和面向软件两类。面向硬件的方法是从完全由硬件实现开始,在满足性能约束的条件下,反复将组件向软件移动[7,8];面向软件的方法则是从完全由软件实现开始,并逐渐向硬件移动,直到满足时间的限制为止[9,10]。软硬件划分中已有的启发式算法包括遗传算法[11]、模拟退火算法[12]、禁忌搜索算法(tabu search)[13]和贪心算法[14]等。

已有的高效软硬件划分算法将软硬件划分问题看作一类变异的0-1背包问题,提出了一个一维搜索算法[15]。本文在文献[15]的系统模型上,借鉴求解0-1背包问题的策略,提出了一个求解软硬件划分问题的启发式算法,并利用禁忌搜索技术对该启发式算法得到的近似解进行改进。实验结果表明,本文所提算法对文献[15]中的算法的改进度高达28%。

本文第2节介绍计算模型及问题描述;第3节讨论启发式算法;第4节使用禁忌搜索算法对启发式算法得到的可行解进行优化;第5节展示实验结果并对其进行分析;最后给出结论。

到稿日期:2011-05-13 返修日期:2011-07-01 本文受国家自然科学基金(60970016)资助。
王 璞(1986-),男,研究生,主要研究方向为高性能体系结构,E-mail:wangpu_ah@163.com;武继刚(1963-),男,教授,主要研究方向为理论计算机科学并行分布计算领域。

2 计算模型与问题描述

本文采用文献[15]中提出的计算模型，系统由一个无向图给出，如图1所示，记作 $G=(V,E)$。其中，V 是结点的集合，$V=\{V_1,V_2,\cdots,V_n\}$，代表将被映射到软件或硬件的系统组件；E 是边的集合，代表组件之间的关系。

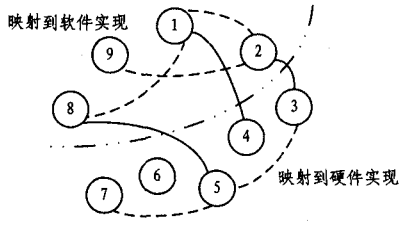


图1 任务图实例

如果两个组件同时被映射到硬件(软件)实现，那么这两个组件间的通信费用可忽略不计(图1中的虚线表示)；如果一个组件被映射到硬件(软件)实现，另一个被映射到软件(硬件)实现，那么它们之间的通信费用需要考虑(图1中的实线表示)，记为 $c(v_i,v_j)$ 或 c_{ij}。

软硬件划分实质是对结点集合做划分，即将原结点集合 V 划分为两个子集，亦即 $P=(V_H,V_S)$。其中，$V_H\cup V_S=V$ 且 $V_H\cap V_S=\varnothing$。划分 P 的割边集被定义为 $E_P=\{(v_i,v_j)\mid i\neq j,v_i\in V_S,v_j\in V_H\}$。

划分后系统总的硬件代价、软件代价以及通信代价的形式化定义分别由式(1)－式(3)给出[15]。

$$H_P=\sum_{v_i\in V_H}h_i \tag{1}$$

$$S_P=\sum_{v_i\in V_S}s_i \tag{2}$$

$$C_P=\sum_{(v_i,v_j)\in E_P}c_{(v_i,v_j)} \tag{3}$$

式中，h_i,s_i 分别表示第 i 个组件交由硬件实现和软件实现的系统开销。

问题 P：给定一个任务图 G 与代价函数 s,h,c(分别表示软件代价、硬件代价和通信代价)以及约束 $R\geqslant 0$，求解一个软硬件划分，在满足 $S_P+C_P\leqslant R$ 的条件下，使得 H_P 在这些划分中最小。

用 $\boldsymbol{x}=(x_1,x_2,x_3,\cdots,x_n)$ 表示一个划分，$x_i=1$ 表示该组件由软件实现；$x_i=0$ 表示由硬件实现，则系统的硬件代价、软件代价以及通信代价可以分别表示为式(4)－式(6)。

$$S(\boldsymbol{x})=\sum_{i=1}^{n}s_ix_i \tag{4}$$

$$H(\boldsymbol{x})=\sum_{i=1}^{n}h_i(1-x_i) \tag{5}$$

$$C(\boldsymbol{x})=\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}\mid x_i-x_j\mid \tag{6}$$

则 P 问题的形式化描述如式(7)所示。

$$P\begin{cases}\text{minimize } H(\boldsymbol{x})\\ \text{subject to } S(\boldsymbol{x})+C(\boldsymbol{x})\leqslant R\\ \qquad\qquad x_i\in\{0,1\},i=1,2,\cdots,n\end{cases} \tag{7}$$

3 基于0-1背包问题的启发式算法

背包问题是如何在一堆不同重量和价值的物品中选择一部分放进背包，在背包容量的约束下，使得背包中物品的价值总量最大。这个在空间限制下求最大收益的问题可以被形式化如下：

给定背包容量 K 和一组物品 $S=\{1,2,\cdots,n\}$，其中每个物品拥有一个重量 w_i 和一个收益 b_i，背包问题就是要找到一个子集 $S'\subset S$，在满足约束条件 $\sum w_i\leqslant K,i\in S'$(即物品的总容量要小于等于背包的总容量 K)下，使总收益 $\sum b_i,i\in S'$ 最大。

0-1背包问题是一般背包问题的一种特例，即每个物品只能完全被选择或者不被选择，而不能仅被部分地选择，形式化描述如式(8)所示。

$$0\text{-}1KP\begin{cases}\text{maximize } \sum_{i=1}^{n}b_ix_i\\ \text{subject to } \sum_{i=1}^{n}w_ix_i\leqslant K\\ \qquad\qquad x_i\in\{0,1\},i=1,2,\cdots,n\end{cases} \tag{8}$$

式中，x_i 是一个二元变量，如果物品 i 放进背包，则 x_i 为1，否则为0。

通过简单地等价变换，P 问题可以变换为如下形式的问题 Q，如式(9)所示。

$$Q\begin{cases}\text{maximize } \sum_{i=1}^{n}h_ix_i\\ \text{subject to } \sum_{i=1}^{n}s_ix_i+C(x)\leqslant R\\ \qquad\qquad x_i\in\{0,1\},i=1,2,\cdots,n\end{cases} \tag{9}$$

令问题 Q 中的 h_i,s_i,R 分别对应0-1背包中的 b_i,w_i,K，则问题 Q 可以看作是一个约束条件中带有额外通讯代价的扩展的0-1背包问题，于是一些解决0-1背包问题的算法也可以用来解决问题 Q。因此，本文提出了一个基于求解背包问题的软硬件划分算法。

对于0-1背包问题，一个简单但很有效的启发式算法[16]是：先按照物品的价值与重量比的值进行排序，如式(10)所示；再依此选择物品放入背包，直到没有满足条件的物品装入背包为止。

$$\frac{b_1}{w_1}\geqslant\frac{b_2}{w_2}\geqslant\cdots\geqslant\frac{b_n}{w_n} \tag{10}$$

文献[15]中的算法利用这样的贪心策略，通过搜索一个一维搜索空间来获得近似解。这一算法在本文中记作 Alg-old。

本文提出的算法最初不考虑通讯费用，按组件的硬件代价与软件代价的比值排序，然后依次选择组件给予软件实现，这样就得到了一个贪心解。因为当两个组件用不同的方式实现时会产生通讯费用，这就导致由此产生的解可能不是一个可行解。所以，必须对这个解进行判断与调整。

调整分两步进行。首先，对于每个由软件实现的组件，分别计算将其硬件实现时造成的通讯费用与软件代价的变化，即对每个已经装入背包的物品，计算将其移出背包所释放的空间。为了使得物品移出背包时所带来的价值损失最小，且释放的空间最大，这里按照价值损失与释放空间的比值由小到大的顺序来选择物品拿出背包。相应地，对于每个硬件实现的组件，分别计算出将其交由软件实现，所造成的通讯费用和软件代价的变化，即对每个未装入背包的物品，计算将其放入背包所带来的背包空间的占有量。为了使物品放入背包时所带来的价值增量最大，并占用的空间最小，这里按价值增加与占有空间的比值由大到小的顺序来选择物品装入背包。启发式算法的形式化描述如下。

Input: communication graph G and the constraint R.

Output: a partitioning solution $\boldsymbol{x}=(x_1,x_2,x_3,\cdots,x_n)$ and the total hardware cost.

Algorithm Alg-adj(gG,R,$\boldsymbol{x}$)

begin

1. $\boldsymbol{x}=(0,0,\cdots,0)$; max_sum_hixi=0; /* initialzing */

2. Sort all nodes according to

$$\frac{h_1}{s_1}\geqslant\frac{h_2}{s_2}\geqslant\cdots\geqslant\frac{h_n}{s_n};$$

3. $i:=1$; $rec:=R$; /* rec means *residual_capacity* */

4. repeat

/* the communication costs are not considered */

4.1 if $s_i\leqslant rec$ then

/* task i fits in the unused capacity */

Pack task i into the knapsack, and $x_i:=1$;

$rec:=rec-s_i$;

4.2 $i:=i+1$

until ($rec\leqslant 0$) or ($i>n$);

/* A greedy solution and the hardware cost is obtained */

5. if ($S(\boldsymbol{x})+C(\boldsymbol{x})\leqslant R$) then max_sum_hixi:=$H(\boldsymbol{x})$

else

5.1 max_sum_hixi:=0;

5.2 repeat

5.2.1 for i:=1 to n do

if ($x_i==1$) then /* task i is in knapsack */

Evaluate communication change c_i and $h_i/(s_i+c_i)$ for moving task i out of the knapsack;

end for

5.2.2 Move the task k with the minimum value of $h_k/(s_k+c_k)$ out of the knapsack and set x_k to 0;

5.2.3 update $S(\boldsymbol{x})$, $C(\boldsymbol{x})$, $H(\boldsymbol{x})$;

until ($S(\boldsymbol{x})+C(\boldsymbol{x})\leqslant R$)

end else

6. $rec':=R-(S(\boldsymbol{x})+C(\boldsymbol{x}))$

7. repeat

7.1 for i:=1 to n do

if ($x_i==0$) then /* task i is not in knapsack */

Evaluate communication change c_i and $h_i/(s_i+c_i)$ for moving task i into the knapsack;

end for

7.2 if $s_k+c_k\leqslant rec'$ then

Move the task k with the maximum value of $h_k/(s_k+c_k)$ into the knapsack and $x_k:=1$;

$rec':=rec'-(s_k+c_k)$;

7.3 update $S(\boldsymbol{x})$, $C(\boldsymbol{x})$, $H(\boldsymbol{x})$;

until ($rec'\leqslant 0$) or (no more task to fit for the remaining capacity)

8. max_sum_hixi:=$H(x)$;

end

用 n 和 m 分别表示图中的结点数和边数，任务的编号和图中结点的编号是一致的。算法 Alg-adj 的时间复杂度主要由第 5 步和第 7 步决定。第 4 步可以在 O(n)时间内完成，因为第 2 步排序完成后，已经有了一个有序序列。要完成排序，需要的时间是 O($n\log n$)。对于第 5.2.1 步，虽然每次将一个任务移出背包后都需要重新计算其他任务的通讯费用的变化值(算法描述中用 c_i 表示)，但实际上，如果在上一轮循环中任务 k 已被移出了背包，则图中受影响的只是和结点 k 直接相连的那些结点，所以只需要重新对这些结点进行评估。用 m_i 表示和结点 v_i 关联的边数，则通过 $\sum_{i=1}^{n}m_i=2m$ 可以推断出这一步的执行时间是 O(m)。第 5.2.2 步是在一个无序序列中查找最小值，显然可以在 O(n)时间内完成。第 5.2.3 步关键是要计算式(4)－式(6)的右边，如果将任务 k 移出背包，则计算 $S(\boldsymbol{x})$，$H(\boldsymbol{x})$只需要重新考虑将任务 k 移出带来的变化值，而 $C(\boldsymbol{x})$也只需要考虑和结点 k 关联的那些边。因此这一步可以在 O($n+m$)内完成。

综上所述，算法 Alg-adj 的时间复杂度为 O($n\log n+m$)。

4 禁忌搜索算法

禁忌搜索 (Tabu Search) 最早由 Glover (1986)提出，它是对局部邻域搜索的一种扩展，并且是一种全局逐步寻优算法。所谓禁忌，就是禁止重复前面达到的局部最优状态。禁忌搜索通过引入一个灵活的存储结构，并配备一个相应的禁忌准则来避免重复搜索，陷入局部最优。禁忌搜索还通过附加一个特赦准则来赦免一些质量优良但处于禁忌状态的解，通过这些手段来保证搜索的多样化，最终实现全局优化[17]。

本文就是利用禁忌搜索对上面的基于背包问题提出的启发解进行优化。禁忌搜索的相关参数设置如下。

邻域结构：用 $\boldsymbol{x}=(x_1,x_2,x_3,\cdots,x_n)$表示一个划分，$x_i=1$ 表示该组件系统由软件实现，$x_i=0$ 则表示该组件由硬件实现。因此，启发式算法的解是一个 0-1 序列。在此 0-1 序列中随机选取两位，同时进行翻转来产生邻域解。所谓翻转，即原来 $x_i=1$，翻转后 $x_i=0$；原来 $x_i=0$，翻转后 $x_i=1$。

禁忌对象：解的分量的变化，即 x_i 的变化。

评价函数：是选取元素构成候选集合的一个评价标准，这里采用基于目标函数的评价函数，并且能接受所记忆的频率信息的反馈来修正评价值。

候选集：本文候选集是由从邻域解的集合中选取评价函数最好的若干可行解构成。

禁忌长度：禁忌对象不允许选取的迭代次数。它的选取十分重要，太小容易使搜索陷入局部最优解，太大则不利于全局寻优。本文采取动态的禁忌长度，即禁忌长度在整个迭代过程中并不是保持不变的，它会在频率记忆的指导下随时做相应改变。

特赦准则：当候选集中全部对象都处于禁忌状态或某个禁忌对象能使目标值有较大的减少时，采用特赦准则，使这些禁忌对象重新可选。

频率控制：在若干轮迭代后，现有的参数可能无法得到更好的解，因此需记忆被禁对象的出现频率，从而动态调整参数，如禁忌长度、评价函数等。

终止准则：当算法执行到了事先设置的最大迭代次数或在一个给定的迭代次数内，目标值没有明显变化，算法终止。

本文采用的禁忌搜索算法在实现过程中的具体参数设置如下：最大迭代次数设为 2000；迭代产生的解如果在 200 代内没有提升，则算法终止；邻域解集合的大小设为 2000；候选集大小设为 800；禁忌长度则在$(1/2\cdot\sqrt{n},\sqrt{n})$中随机产生。其中，$n$ 为任务图中的结点数。

使用以上规则构造的禁忌搜索算法在本文中记作 Alg-tabu。受篇幅所限，这里就不再给出该算法的形式化描述。

5 实验结果及分析

本文在文献[15]中提供的实验基准(benchmark)上，使用

了文献[15]的算法源代码进行算法性能的比较。实验中依然采用了文献[15]的参数设置，简略介绍如下。

软件代价 s_i 在[1,100]中随机产生并服从均匀分布，硬件代价根据数学期望为 $k \cdot s_i$ 的正态分布随机产生，其中 s_i 是对应 h_i 模块的软件代价，而 k 与 s_i，h_i 的单位选择有关，因而是无关紧要的。

通讯代价在$[0,2\rho S_{max}]$中随机产生，并服从均匀分布，其中 S_{max} 是最大软件代价，ρ 为通信计算比 CCR（communication to computation ratio），分别取 0.1，1.0 与 10.0。相应的任务图分别被称为计算密集型、中间型和通讯密集型。

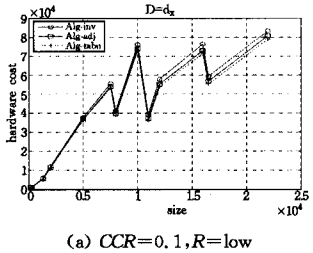
问题 P 中定义的约束 R 服从均匀分布，根据约束条件的不同，分别在$[0,1/2 \cdot \sum s]$或$[1/2 \cdot \sum s, \sum s]$中随机产生，前者生成的 R 值较小，系统的软件代价和通讯代价受到的约束较强，称为强实时约束，在本文里对应于 R=low；后者生成的 R 值较大，对应的约束较弱，称为弱实时约束，在本文里对应于 R=high。

表 1 给出文献[15]中使用的全部基准，其中 n 和 m 分别代表通讯图中的结点数和边数，而问题大小被定义为 $2n+3m$，这是考虑到任务图中，每个结点被赋予两个值：软件代价和硬件代价；而每条边都被赋予 3 个值：连结的两个结点的结点号以及通讯代价。

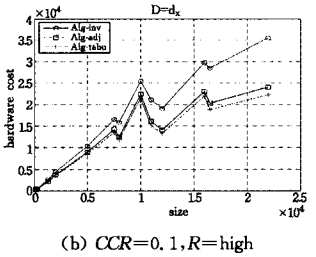
表 1　文献[15]中涉及的 Benchmark

Benchmark	n	m	Question size (2n+3m)
Crc32	25	34	152
Patricia	21	50	192
Dijkstra	26	71	265
Clustering	150	333	1299
RC6	329	448	2002
Random1	1000	1000	5000
Random2	1000	2000	8000
Random3	1000	3000	11000
Random4	1500	1500	7500
Random5	1500	3000	12000
Random6	1500	4500	16500
Random7	2000	2000	10000
Random8	2000	4000	16000
Random9	2000	6000	22000

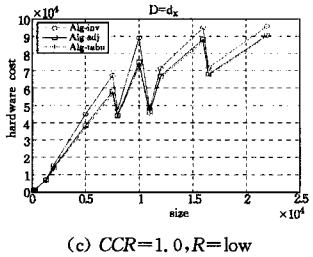
图 2(a)至图 2(f)是对不同算法求得的解的质量进行比较，横坐标是问题大小，纵坐标为硬件代价。图中硬件代价越小，表示解的质量越好。



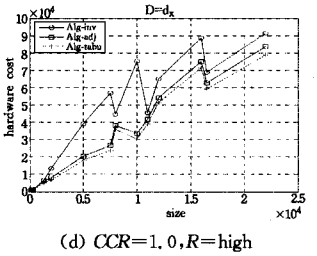
(a) CCR=0.1，R=low



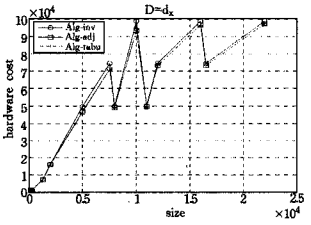
(b) CCR=0.1，R=high



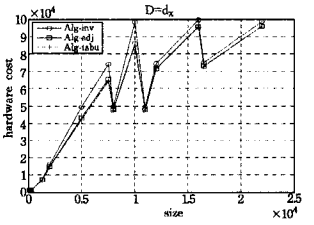
(c) CCR=1.0，R=low



(d) CCR=1.0，R=high



(e) CCR=10.0，R=low



(f) CCR=10.0，R=high

图 2

实验结果显示，在多数情况下，本文提出的启发式算法得到的近似解明显优于文献[15]中算法得到的结果。通过禁忌搜索算法对启发式算法得到的近似解进一步优化后，结果改进明显。在“CCR=0.1，R=low”时，大约改进 5.2%；而在“CCR=0.1，R=high”的情况下，改进为 21.7%。当“CCR=1，R=low”时，改进为 9.4%；而在“CCR=1.0，R=high”的情况下，改进率达到 28.3%。在“CCR=10.0，R=low”时，改进率为 2.3%，而在“CCR=10.0，R=high”的情况下，改进率为 6.0%。由此可以看出，不论通信计算比多大，随着约束条件 R 的增大，改进率也增加。这是由于 R 相当于背包容量 K，背包容量越大，选择的余地越大，也即解的搜索空间就越大，就更有可能找到质量更好的解。

结束语　基于解决 0-1 背包问题的思路，本文提出了一个高效软硬件划分算法，并使用禁忌搜索算法对启发式算法得到的近似解进行优化。使用的基准及设定的参数均与文献[15]中的相同。实验结果表明，本文提出的算法明显优于文献[15]中的算法，解的改进幅度最高达到 28.3%。

参 考 文 献

[1] Wu Ji-gang, Srikanthan T, Jiao Tao. Efficient Heuristics for Functional Partitioning and Scheduling in Hardware/Software Co-design[J]. Design Automation for Embedded Systems, 2008, 12(4): 345-375

[2] Onils M, Jantsch A, Hemani A, et al. Interactive hardware-software partitioning and memory allocation based on data transfer profiling[C]// International Conference on Recent Advances in Mechatronics. Istanbul, Turkey, 1995: 447-452

[3] Wu Ji-gang, Srikanthan T. Low-Complex Dynamic Programming Algorithm for Hardware/Software Partitioning[J]. Information Processing Letters, 2006, 98(2): 41-46

[4] Niemann R, Marwedel P. An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming[J]. Design Automation for Embedded Systems. special Issue: Partitioning Methods for Embedded Systems, 1997, 2(2): 165-193

[5] Weinhardt M. Integer Programming for Partitioning in Software Oriented Codesign[J]. Lecture Notes of Computer Science, 1999, 975: 227-234

[6] Karam S C, Ranga V. Hardware-software Partitioning and Pipelined Scheduling of Transformative Applications[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2002, 10(3): 193-208

[7] Wu Ji-gang, Srikanthan T, Yan Cheng-bin. Algorithmic Aspects for Power-Efficient Hardware/Software Partitioning[J]. Mathematics and Computers in Simulation, 2008, 79(4): 1204-1215

[8] Niemann R, Marwedel P. Hardware/software Partitioning Using Integer Programming[C]// IEEE/ACM European Design Automation Conference(EDAC). Paris, France, 1996: 473-479

[9] Vahid F, Gajski D D. Clustering for Improved Systemlevel Func-

tional Partitioning[C]// IEEE/ACM Int. Symp. System Synthesic. New York, USA, 1995: 28-33

[10] Vahid F, Gajski D D, Gong J. A Binary-constraint Search Algorithm for Minimizing Hardware During Hardware/software Partitioning[C]// IEEE/ACM European Design Automation Conference (EDAC). CA, USA, 1994: 214-219

[11] Quan G, Hu X, Greenwood G W. Preferencedriven Hierarchical Hierarchical Hardware/software Partitioning[C]// IEEE Int. conf. on Computer Design. Austin, Texas, USA, 1999: 652-657

[12] Ernst R, Henkel J, Benner T. Hardware-Software Cosynthesis for Micro-controllers[J]. IEEE Design and Test of Computer, 1993, 10(4): 64-75

[13] Wiangtong T, Cheung P Y K, Luk W. Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-software Codesign[J]. Design Automation for Embedded Systems, 2002, 6: 425-449

[14] Chatha K S, Vemuri R. MAGELLAN: Multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs [C] // CODES. New York, NY, USA, 2001: 42-47

[15] Wu Ji-gang, Srikanthan T. Algorithmic Aspects of Hardware/Software Partitioning: 1D Search Algorithms[J]. IEEE Transactions on computers, 2010, 59(4): 532-544

[16] Toth M S P. Knapsack Problems: Algorithms and Computer Implementations[M]. New York, NY, USA: John Wiley & Sons, 1990: 52-200

[17] Glover F, Laguna M. Tabu Search[M]. Boston: Kluwer Academic Publishers, 1997: 10-150

(上接第 286 页)

在得到隶属度矩阵的基础上，重新在 GPU 中申请一个线程块，块中线程数为聚类数目。按式(6)计算各个新聚类中心，核心函数声明如下：

ComCenter<<<1, *number*>>>(*data*, *data2*, *data3*)

number 为聚类数目。迭代计算隶属度矩阵和各新聚类中心，直到满足所要求的精度。最后，在 GPU 中重新分配线程块及线程数，编写新的内核函数。线程块执行该函数，将每列像素按照隶属度归入相应的类别，声明如下：

Classify<<<*width*, 64>>>(*data*, *data2*)

基于 GPU 加速的 FCM 算法流程如图 2 所示。

5 实验与分析

实验的硬件平台：CPU 为 Pentium4 2.4G；2G 内存；显卡为 Geforce 8800GT。软件平台：Visual Studio 2003。实验均使用红外图像，如图 3 所示。其中，海洋人和马图像大小为 256 * 256，飞机爬升和飞机尾后图像大小为 128 * 128。处理时间比较如表 2 所列。分割结果如图 4 所示。对于 FCM 算法所需要的初始聚类数目，根据图像特点并考虑灰度因素，海洋人图像分为天空、人与摩托艇、深色海水和浪花 4 类；马图像分为道路、草地、树与栅栏、马匹 4 类；飞机图像分为机身、尾焰和天空 3 类。从表 2 中可看出，基于 GPU 的 FCM 分割算法效率改进明显。对于 256 * 256 图像，平均加速比为 10.489。当图像大小为 128 * 128 时，平均加速比达到 4.084。随着数据量的增大，基于 GPU 的算法获得了更大的速度提升。从图 4 可看出，马、飞机爬升和飞机尾后得到了期望的分割结果。而对于海洋人图像，将人物、摩托艇与天空划分为一类，未达到期望效果，说明将 FCM 算法用于图像分割还需要进行改进。

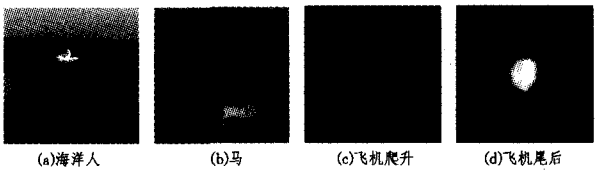


图 3 红外原图像

表 2 基于 GPU 与 CPU 的 FCM 算法执行时间比较

时间(s)	海洋人 4 类	马 4 类	飞机爬升 3 类	飞机尾后 3 类
串行算法	6.4	3.4	0.258	0.125
并行算法	0.605	0.327	0.052	0.039
加速比	10.579	10.398	4.962	3.205

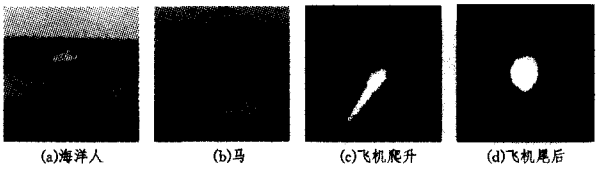


图 4 FCM 分割效果图

结束语 本文实现了基于 GPU 的 FCM 图像分割算法，并对实验结果进行了分析。结果表明，通过分析 FCM 算法各阶段不同数据相同处理的特性，利用 GPU 硬件结构先天的并行计算特点，将 FCM 算法改造成适合 GPU 运行的形式。与 CPU 串行算法相比，本文方法获得了明显效率提升。在图像数据量增大时，这种提升更加明显。在本实验平台上，GPU 算法获得了平均 7 倍的提升。鉴于大多数图像处理算法均具有大量数据进行相同运算的特性，利用 GPU 进行加速具有普适性。

参 考 文 献

[1] Bezdek J. Pattern Recognition with Fuzzy Objective Function Algorithms[M]. New York: Plenum Press, 1981

[2] Guthe M, Balázs á, Klein R. GPU-based Trimming and Tessellation of NURBS and T-spline Surfaces [J]. ACM Transaction on Graphics, 2005, 24(3): 1016-1023

[3] Thompson C J, Hahn S, Oskin M. Using modern graphics architectures for general-purpose computing: A framework and analysis[A]// Proceedings of International Symposium on Microarchitecture[C]. Istanbul, 2002: 306-317

[4] 周世哲，满家巨. 基于多重网格法的实时流体模拟[J]. 计算机辅助设计与图形学学报，2007，19(7)：935-940

[5] 韩峰. 基于 GPU 的有限角度投影数据的 CT 三维重建[D]. 上海：上海交通大学，2008

[6] Govindaraju N K, Henson M, Lin M C, et al. Interactive Visibility Ordering and Transparency Computations Among Geometric Primitives in Complex Environments [C]// Proc. of ACM Symposium on Interactive 3D Graphics and Games. New York: ACM Press, 2005

[7] 杨珂. 基于图形处理器的数据管理技术研究[D]. 浙江：浙江大学，2008

[8] NVIDIA Corporation. CUDA Programming Guide 1.0 [EB/OL]. http:// www. nvidia. com, 2007