中国科学: 信息科学 2018 年 第 48 卷 第 8 期: 978–999
SCIENTIA SINICA Informationis
论文

基于 GPU 的自适应邻域压缩禁忌搜索的软硬件划分算法

侯能[1,2], 何发智[1,2*], 周毅[3], 陈壹林[1,2]

1. 武汉大学软件工程国家重点实验室, 武汉 430072
2. 武汉大学计算机学院, 武汉 430072
3. 武汉科技大学信息科学与工程学院, 武汉 430081
* 通信作者. E-mail: fzhe@whu.edu.cn

收稿日期: 2017–02–07; 接受日期: 2017–09–18; 网络出版日期: 2018–02–07

国家自然科学基金 (批准号: 61472289) 和湖北省自然科学基金 (批准号: 2015CBF254) 资助项目

摘要 软硬件划分是软硬件协同设计中的关键步骤, 决定了哪些功能由硬件执行, 哪些功能由软件执行. 软硬件划分属于 NP 难问题. 现代嵌入式系统的复杂性提高, 造成软硬件划分问题规模变大, 需要采用启发式方法求解. 禁忌搜索是求解软硬件划分的有效方法. 然而, 算法的求解过程非常耗时. 已有的禁忌搜索求解软硬件划分是串行实现, 要折中考虑解的质量和算法的运行时间. 这种考虑牺牲了解的质量. 本文提出基于 GPU 的自适应邻域压缩 (compacting neighborhood) 禁忌搜索的软硬件划分算法. 首先, 提出自适应策略. 自适应策略能够增强算法的搜索集中性, 提高解的质量. GPU 的大规模并行特性可以降低算法的运行时间. 其次, 为了使算法在 GPU 上高效地执行, 提出基于 GPU 的任务图表达、线程 – 候选解映射、数据布局和访存等一系列优化策略. 最后, 实验采用统一设备架构 (CUDA) 编程, 并根据相关基准任务图, 通过不同的计算 – 通信比和实时约束条件, 对提出的方法进行验证. 结果表明, 本文方法的解质量要优于已有的方法. 对比将自适应邻域压缩禁忌搜索自然移植到 GPU 后的运行时间, 提出的 GPU 上的执行优化策略明显地降低了求解时间. 另外, 在更大规模的软硬件划分上验证了基于 GPU 的方法在时间上的优势.

关键词 软硬件协同设计, 启发式方法, 图形处理单元, 禁忌搜索, 自适应算法

1 引言

嵌入式系统通常由硬件执行单元和软件执行单元组成. 硬件单元主要是指面向特定应用的加速处理器, 如 FPGA 或 ASIC. 硬件单元的处理速度快、功耗低, 但是维护成本高; 而软件单元执行速度慢、功耗高, 但是成本低, 易于维护. 在软硬件协同设计过程中, 嵌入式应用程序被抽象为任务图表达. 任

引用格式: 侯能, 何发智, 周毅, 等. 基于 GPU 的自适应邻域压缩禁忌搜索的软硬件划分算法. 中国科学: 信息科学, 2018, 48: 978–999, doi: 10.1360/N112017-00024
Hou N, He F Z, Zhou Y, et al. GPU-based adaptive compacting neighborhood tabu search for hardware/software partitioning (in Chinese). Sci Sin Inform, 2018, 48: 978–999, doi: 10.1360/N112017-00024

务图中存在资源需求不一的节点集合. 在满足特定约束下, 软硬件划分将任务图中的各节点合理地映射到软件或硬件上, 保证系统的执行方案是性能最优的. 因此, 软硬件划分是软硬件协同设计中的关键步骤 [1].

现代嵌入式系统的架构越来越复杂. 目标架构可能由多个软件执行单元和单个硬件执行单元、单个软件执行单元和多个硬件执行单元或者混合的可重构系统片上系统构成 [2~4]. 另外, 嵌入式系统涉及的性能指标繁多, 往往根据具体应用选择影响系统性能的单个或几个指标作为优化目标. 因此, 在划分目标上, 软硬件划分又分成单目标划分和多目标划分 [5~8]; 也可以将多目标线性化加权为单目标来求解 [9]. 在大多数软硬件划分中, 软硬件划分的模型建立都是取决于具体的应用背景. 在最新的研究工作中, Wang [10] 将非确定理论应用到软硬件划分的建模中. 基于该理论的建模方法更具普遍意义, 能够适应不同应用背景下的嵌入式系统在架构和优化目标上的多样性.

在表达待划分的嵌入式应用时, 多数工作将待划分的任务图表示成控制 – 数据流图 (control–data flow graph). 而在文献 [11, 12] 中, Arato 在硬件层面上没有针对特定的硬件架构, 而在抽象层面上将待划分任务表示为无向通信图, 进一步通过理论描述指出软硬划分问题可分为两类, 一类为可以通过多项式求解的问题, 另一类则为 NP 难的问题.

在算法方面, 求解软硬件划分问题的常用算法主要分为两类, 即精确求解算法和近似求解算法. 精确求解算法主要用来处理小规模的软硬件划分问题. 这类方法包括动态规划 [13, 14]、线性规划 [15, 16]、分支限界方法 [17, 18] 等. 当问题规模增大时, 解的搜索空间呈指数增长, 精确求解变得不可行, 转而通过启发式方法求近似解.

在早期软硬件协同设计阶段, 主要采用传统启发式方法. 典型的传统启发式方法分为面向性能的硬件约束优先方法或面向时间的软件约束优先方法 [19, 20]. 而通用启发式方法主要指通过各类智能搜索算法, 如遗传算法 [11, 12, 21]、蚁群算法 [4, 22]、粒子群优化方 [23]、模拟退火方法 [24, 25]、人工免疫算法 [26]、人工蜂群算法 [27]、禁忌搜索算法 [28] 等, 以及这些方法的结合 [3, 29~32].

软硬件划分中用到的通用启发式算法大多属于种群类或基于轨迹的元启发式算法. 从并行计算的角度看, 这些算法本身具有内在的并行性. 因此, 很自然地可以对这些算法并行化以加速问题的求解. 关于并行算法求解软硬件划分问题的工作, 已有的工作包括基于集群计算平台的并行遗传算法 [33] 和并行混合粒子群算法 [34]. 其中, 前者为 4 个 PC 组成的小规模集群, 每个 PC 的 CPU 为奔腾 4 处理器; 后者为包含 174 个计算节点的至强 3000 高性能集群系统, 理论计算峰值为 2.15 万亿次/秒. 另外, 文献 [35] 提出并行的和声搜索增强的粒子群优化求解软硬件划分. 在求解过程中用到了多核 CPU 并行来加速适应度的计算, 但是该方法仅强调多核并行部分与未并行部分的比较, 未能说明整个算法下在多核加速下的受益.

近年来, 图形处理单元 (graphical processing unit, GPU) 作为并行计算平台在多个领域得到了广泛应用 [36~41]. GPU 具有随处可得、低功耗、高性价比等优势. 在同一价位上, GPU 的整体性能要高出 CPU 一个数量级. 将 GPU 用于加速软硬件划分的求解是一个值得探索的问题. 尽管目前存在各种软硬件划分方法, 由于软硬件协同设计环境和解决的问题各不相同, 使得算法间的比较缺乏统一的基准 [28, 42, 43]. 在各种方法中, 遗传算法、模拟退火算法和禁忌搜索算法是求解软硬件划分问题的常用算法. 与遗传算法和模拟退火相比, 禁忌搜索能得到更高质量的解. 然而, 禁忌搜索的求解过程非常耗时 [9, 28, 42~44]. 已有的禁忌搜索求解软硬件划分主要描述串行实现. 在算法的实现过程中, 即使折中考虑了解的质量和算法的运行时间, 但是相对于遗传和模拟退火等方法, 禁忌搜索的求解时间依然很慢. 重要的是, 折中考虑导致算法为了运行时间的减少而牺牲了解的质量.

因此, 本文提出基于 GPU 的自适应邻域压缩禁忌搜索的软硬件划分算法. 本文的主要贡献有: 首

先, 本文提出自适应策略的禁忌搜索求解软硬件划分的算法. 自适应策略能够增强算法的搜索集中性, 提高解的质量. 同时, GPU 的大规模并行特性能够使自适应策略在可接受的时间内完成. 其次, 为了使算法在 GPU 上高效地执行, 提出基于 GPU 的任务图表达、数据布局、线程 – 候选解映射和访存等一系列优化策略. 最后, 实验部分验证了方法的有效性.

本文的组织如下: 第 2 部分介绍所需的背景知识; 第 3 部分首先介绍提出的禁忌搜索算法优化初始划分的预实现及细节; 然后详细描述方法在 GPU 上所做的适应性改变以及优化策略; 第 4 部分通过实验说明本文工作的有效性; 最后给出本文工作的结论.

2 相关背景知识

2.1 软硬件划分问题的通用描述

如文献 [12] 所示, 将待划分的任务图描述成无向通信图, 图中的节点或者由软件执行, 或者由硬件执行. 图中的边表示节点之间有可能存在通信关系. 通过形式化, 令任务图为 $G(V,E)$. 其中, $V=\{v_1,v_2,\ldots,v_n\}$, 表示待划分的任务节点集合. 每个节点包含两个耗费, 即节点划分为硬件执行时的耗费 $h(v_i)$ 和软件执行时的耗费 $s(v_i)$. E 代表节点之间的边集合, 边上的权值表示当两个相邻节点属于不同的划分时, 节点之间会产生相应的通信耗费 $c(v_i,v_j)$. 将 $P=\{V_H,V_S\}$ 称为一个软硬件划分. 其中, V_H 表示划分为硬件执行的节点集合; V_S 表示划分为软件执行的节点集合. 并且, V_H 和 V_S 要满足 $V_H\bigcap V_S=\varnothing$ 以及 $V_H\bigcup V_S=V$. 相应地, 划分 P 的边集合定义为 $E_P=\{(v_i,v_j):v_i\in V_H,v_j\in V_S$ 或者 $v_i\in V_S,v_j\in V_H\}$.

划分 P 主要由 3 个耗费进行度量, 即硬件耗费 H_P、软件耗费 S_P, 以及通信耗费 C_P. 这 3 个耗费形式化如下:

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

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

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

通过式 (1)∼(3), 划分问题 P 可定义如下.

定义1 给定图 G, 耗费函数 s,h,c 和 $R\geqslant 0$. 找到一个软硬件划分 P, 使其在满足约束 $S_P+C_P\leqslant R$ 的条件下, 该划分的硬件耗费 H_P 最小.

划分 P 的解可表示成 n 维向量 $\boldsymbol{x}=\{x_1,x_2,\ldots,x_n\}$. 分量 $x_i=1$ 时, 表示节点由软件执行; $x_i=0$ 时, 表示节点由硬件执行. 因此, 问题 P 可以归纳为如下最小化问题:

$$P\begin{cases}\min\sum\limits_{i=1}^{n}h_i(1-x_i),\\ \sum\limits_{i=1}^{n}s_ix_i+C(x_i)\leqslant R,\end{cases}$$

其中, h_i 表示节点 i 的硬件耗费, s_i 表示节点 i 的软件耗费, $C(\boldsymbol{x})$ 表示整个系统的通信耗费. 进一步地, 问题 P 的硬件耗费、软件耗费, 以及通信耗费的如式 (4)∼(6) 所示:

$$H(\boldsymbol{x}) = \sum_{i=1}^{n} h_i(1 - x_i), \tag{4}$$

$$S(\boldsymbol{x}) = \sum_{i=1}^{n} s_i(x_i), \tag{5}$$

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

问题 P 中, 用来评价一个解是否满足约束的计算公式如下:

$$\sum_{i=1}^{n} s_i(x_i) + \sum_{i=1}^{n-1} \sum_{j=i-1}^{n} c_{ij}|x_i - x_j| \leqslant R. \tag{7}$$

2.2 通用描述的求解方法现状

通过将问题 P 看成标准 0-1 背包的变种问题, Wu 等 [45] 提出一维搜索方法, 相对于文献 [12] 提出的二维搜索方法, 降低了算法的复杂度, 使得算法在质量和速度上都得到了提升. 然而, 该方法在理论上有所欠缺. 因此, Quan 等 [46] 完善了方法的理论描述; Wang 等 [44] 提出了 HEUR 算法. 该方法首先忽略通信耗费, 将问题 P 当成标准 0-1 背包问题得到基础解, 在此基础上再根据通信耗费调整基础解得到满足约束的初始解. 提出的算法在求解质量上较一维搜索方法得到了提升. 另外, 作者还使用禁忌搜索 TABU 对算法的初始解进一步进行优化. 从实验部分来看, 当问题规模较大时, TABU 算法变得非常耗时; 最后, Chen 等 [47] 基于 Page Rank 算法的主要思想, 对 HEUR 算法作迭代改进, 提出 NodeRank 算法. 使得解的质量在部分情况下优于 HEUR 和 TABU 组合得到的解质量.

2.3 基本禁忌搜索和当前的 GPU 禁忌搜索

禁忌搜索元启发式方法由 Glover 等 [48] 提出. 目前已广泛地用来求解各种优化问题. 禁忌搜索的过程可以描述为: 以当前解为起点, 随机生成一定规模的候选解, 通过局部搜索找到最佳候选解作为下次迭代时的起点, 此过程一直重复, 直到满足停止准则. 在搜索的过程中, 禁忌搜索通过禁忌表的记忆机制禁止重复搜索已找过的最优解, 转去寻找未禁忌的次优解, 以此避免陷入局部最优. 另外, 当某个被禁忌的候选解的目标函数值优于已知的全局最优解时, 该禁忌解通过赦免准则解禁. 禁忌搜索的过程既体现了围绕当前解的集中搜索策略, 又体现了跳出局部最优的多样性搜索策略. 最后, 为防止算法无限迭代, 需要通过停止准则使算法终止. 算法停止的依据包括解的精度达到了可接受的程度或算法运行至指定周期数.

目前, 尽管已有求解其他问题的 GPU 禁忌搜索方法, 但是这些问题大多属于求解旅行商 [37]、二次分派 [38,39]、流水车间调度 [40] 等经典问题. 并且, 这些问题在求解过程中不用考虑约束的影响. 在文献 [41] 中, 作者就资源约束的工程调度问题提出一套禁忌搜索解决方案. 该方法能够根据先验移除不满足约束的候选者, 并且是多个禁忌搜索之间的协作并行. 这与本文的软硬件划分的求解方案在本质上是不同的. 另外, 该问题也没有考虑不存在可行候选者的情况.

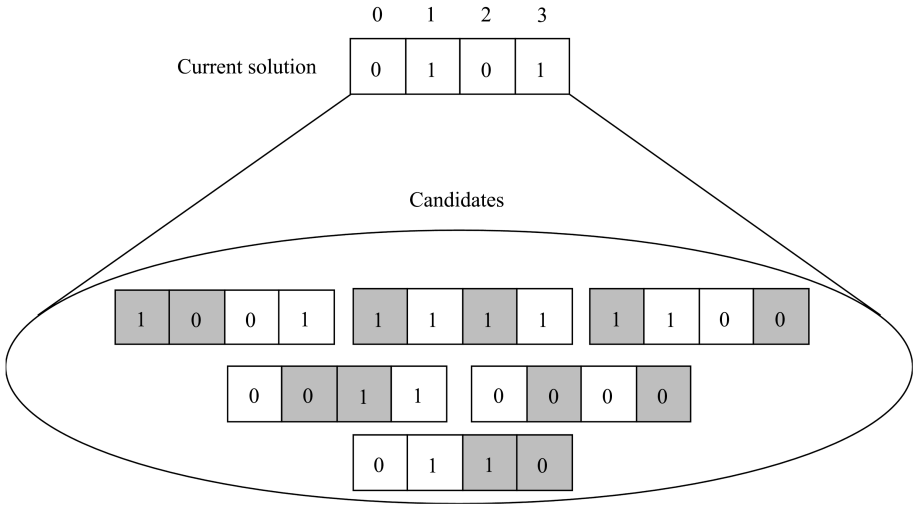

图 1　邻域生成

Figure 1　Neighborhood generation

3　文中方法

本文的禁忌搜索与文献 [44] 中的 TABU 的不同之处在于本文方法充分利用了 GPU 的体系结构特性, 是一种更适合 GPU 加速的软硬件划分算法.

假定软硬件划分的初始解由文献 [44] 中的 HEUR 算法得到, 接下来提出本文的基于 GPU 的自适应邻域禁忌搜索的软硬件划分算法.

3.1　自适应策略与邻域压缩

3.1.1　基于任务节点规模的自适应邻域生成

在文献 [44] 中, 邻域中的候选解通过随机地对当前解的两个位置进行翻转得到. 对于任意节点规模的任务, 邻域的规模都是固定值. 在本文中, 邻域规模与任务节点个数相关. 假定带划分任务的节点个数为 n, 候选解通过顺序地翻转当前解的两个位置来生成. 因此, 候选解的个数一共为 $\frac{n\times(n-1)}{2}$. 图 1 为邻域生成的示意图.

考虑邻域中所有的 2 翻转候选解的必要性体现在两个方面. 一方面, 在每次迭代时, 仅需将当前解传输给 GPU 即可, 可以节省将候选邻域的翻转位置传输给 GPU 引起的传输开销. 另一方面, 在 CPU 上执行禁忌搜索时会权衡算法的执行时间和求解质量的改进程度, 当问题规模很大时, 常用做法是随机地挑选部分邻域解进行评价; 而在 GPU 上, 计算单元的数量远多于 CPU. 再加上禁忌搜索的计算邻域候选解的过程是独立的, 具有内在的并行性. 为充分发挥 GPU 的计算能力, 并旨在寻找更优的候选解, 本文的邻域将所有 2 翻转的候选解都考虑进来, 并评价是否满足约束.

3.1.2　基于增量策略的候选解耗费计算

在邻域中, 每个候选解都要计算软件耗费、硬件耗费, 以及通信耗费. 若当前解为 $\boldsymbol{x}_{\text{current}}$, 则当前解对应的硬件耗费为 $H(\boldsymbol{x}_{\text{current}})$, 软件耗费为 $S(\boldsymbol{x}_{\text{current}})$, 通信耗费为 $C(\boldsymbol{x}_{\text{current}})$. 在计算候选解的耗费时, 假定被翻转的位置为 i, 则对应的解分量为 x_i. 根据 x_i 的状态变化, 此时将候选解的硬件耗费

更新为

$$H(\boldsymbol{x}_{\text{new}}) = \begin{cases} H(\boldsymbol{x}_{\text{current}}) - h_i, & x_i = 1, \\ H(\boldsymbol{x}_{\text{current}}) + h_i, & x_i = 0, \end{cases}$$

其中, h_i 表示节点 i 由硬件执行时的耗费. 同样地, 候选解的软件耗费更新为

$$S(\boldsymbol{x}_{\text{new}}) = \begin{cases} S(\boldsymbol{x}_{\text{current}}) + s_i, & x_i = 1, \\ S(\boldsymbol{x}_{\text{current}}) - s_i, & x_i = 0, \end{cases}$$

其中, s_i 表示节点 i 由软件执行时的耗费.

当翻转某个位置的状态时, 如果翻转后与相邻节点处于同一划分状态, 两节点间就不产生通信耗费; 否则, 两节点间会产生新的通信耗费. 因此, 邻域的通信耗费的更新如算法 1.

算法 1 Update of communication cost

Require: Communication cost of current solution $C(\boldsymbol{x}_{\text{current}})$;
Ensure: Communication cost of new solution $C(\boldsymbol{x}_{\text{new}})$;
  $C(\boldsymbol{x}_{\text{new}}) \Leftarrow C(\boldsymbol{x}_{\text{current}})$;
  while adjacent nodes x_{adjacent} of x_i exist **do**
    if $x_i = x_{\text{adjacent}}$ **then**
      $C(\boldsymbol{x}_{\text{new}}) \Leftarrow C(\boldsymbol{x}_{\text{new}}) - c(x_i, x_{\text{adjcent}})$;
    else
      $C(\boldsymbol{x}_{\text{new}}) \Leftarrow C(\boldsymbol{x}_{\text{new}}) + c(x_i, x_{\text{adjcent}})$;
    end if
  end while

需要注意的是, 以上过程只表示对当前解的位置进行翻转一次的计算方法. 由于邻域候选解是翻转两个位置生成的, 所有上述过程需要计算两次.

3.1.3 基于邻域压缩的可行解鉴别与统计方法

软硬件划分问题是带约束的优化问题, 因此每个候选解在得到对应的软件耗费、硬件耗费, 以及通信耗费以后, 需要根据不等式 (7) 判断候选解是否满足约束条件 R. 如果满足, 则标示该候选解是可行解; 否则为不可行解. 基于此判断, 本文通过邻域压缩来鉴别和保留满足约束的可行解, 并对其数量进行统计.

值得注意地是, 极端情况下会出现当前解的所有 2 翻转邻域都不满足约束. 此时, 本文提出重新初始化当前解的策略. 即首先将当前解初始化为全硬件实现. 在此基础上, 随机地翻转两个位置生成新的当前解, 再由新的当前解生成新的 2 翻转邻域. 如果仍然不能出现满足约束的候选解, 则再次重新初始化, 直到有满足约束条件的候选解存在为止. 重新初始化的策略保证了迭代无法进行下去的时候通过转移搜索空间来达到搜索过程的多样性的目的.

3.1.4 基于邻域索引的禁忌状态表与禁忌表

在邻域压缩阶段后, 对于所有满足约束的候选解, 需要进一步对其禁忌状态进行评价. 该过程主要包括两个方面, 一方面从所有的非禁忌候选解中选择硬件耗费最小的解; 另一方面, 如果被禁忌的候选解的硬件耗费小于已知的全局最优解的硬件耗费, 则通过藐视准则解禁该候选解.

在禁忌搜索中, 禁忌表的构造对禁忌评价的效率至关重要. 禁忌表既要保证搜索的高效, 又要避免构造开销过大. 本文方法中, 禁忌表通过循环队列实现. 同样地, 队列长度与任务中的节点相关. 本文将禁忌表的长度设置为 $\sqrt{n}$. 其中, n 表示任务图的节点数量. 禁忌表中的禁忌对象为邻域标示号, 而邻域标示号可以间接地表示形成该邻域所需的两个翻转位置. 由于队列的先进先出特性, 被禁忌对象的禁忌周期即为队列的长度.

由于每个满足约束的候选解都需要通过遍历禁忌表的方式来查找该候选解是否禁忌. 所以反复地对禁忌表进行遍历使得查询效率较低. 为避免禁忌评价可行候选解时反复地遍历禁忌表, 本文使用禁忌状态数组标示所有候选邻域的禁忌状态. 如果该候选解为禁忌的, 则标示为 1; 否则为 0. 禁忌状态数组的大小与邻域规模有关, 即整个数组大小为 $\frac{n\times(n-1)}{2}$. 值得注意的是, 在每次对禁忌表进行更新时, 也要同时对缓存数组进行更新. 算法 2 为禁忌评价过程的描述.

算法 2 Tabu evaluation

Require: Feasible candidates, tabu list, tabu status table;
Ensure: Optimal candidate;
  Initialize the optimal candidate as maximal value;
  for all feasible candidates **do**
    if tabu status is true **then**
      if $H(\boldsymbol{x}) \leqslant H(\boldsymbol{x}_{\text{globaloptimal}})$ **then**
        Update the optimal candidate by aspiration criteria;
      end if
    else
      if $H(\boldsymbol{x}) \leqslant H(\boldsymbol{x}_{\text{optimal}})$ **then**
        $H(\boldsymbol{x}_{\text{optimal}}) \Leftarrow H(\boldsymbol{x})$;
      end if
    end if
  end for

当所有可行解都被禁忌且没有触发赦免准则时, 本文会随机解禁一个可行邻域标示符. 相应地, 在更新禁忌表时, 首先检查最优候选邻域的标示符是否在禁忌表中. 在检查候选邻域标示符是否禁忌时, 只需访问禁忌状态表中的对应位置即可. 如果在禁忌表中, 表明该解是通过随机解禁或者赦免准则得到的, 此时要先将表尾至该标示符之前的所有禁忌标示符往表头方向前进一位, 然后将该最优候选标示符放至表尾, 完成禁忌表的更新. 如果不在禁忌表中, 表明该最优候选标示符是新的待禁忌对象. 此时, 算法会进一步判断禁忌表是否已满, 如果禁忌表已满, 则解禁表头的禁忌标示符, 并将该标示符在禁忌状态表中的值设置为 0. 如果禁忌表未满, 直接表尾入队, 并设置标示符在禁忌状态表中的值设置为 1.

在得到最优候选邻域标示后, 即可更新当前解以及对应的 3 个耗费, 作为下次迭代的起点. 另外, 如果新的当前解优于全局最优解, 更新全局最优解.

3.2 基于 GPU 的自适应邻域压缩禁忌搜索

3.2.1 算法框架

结合 3.1 小节的描述, 本文基于 GPU 的自适应邻域压缩禁忌搜索算法的框架如图 2 表示. 在 GPU 部分的计算主要包括 3 个部分: 第 1 部分为计算邻域中的候选解的软硬件耗费、通信耗费, 以及是否满足约束条件. 在该阶段, 可以将所有的候选解均匀分配给 GPU 的标量处理器中, 即逻辑上一

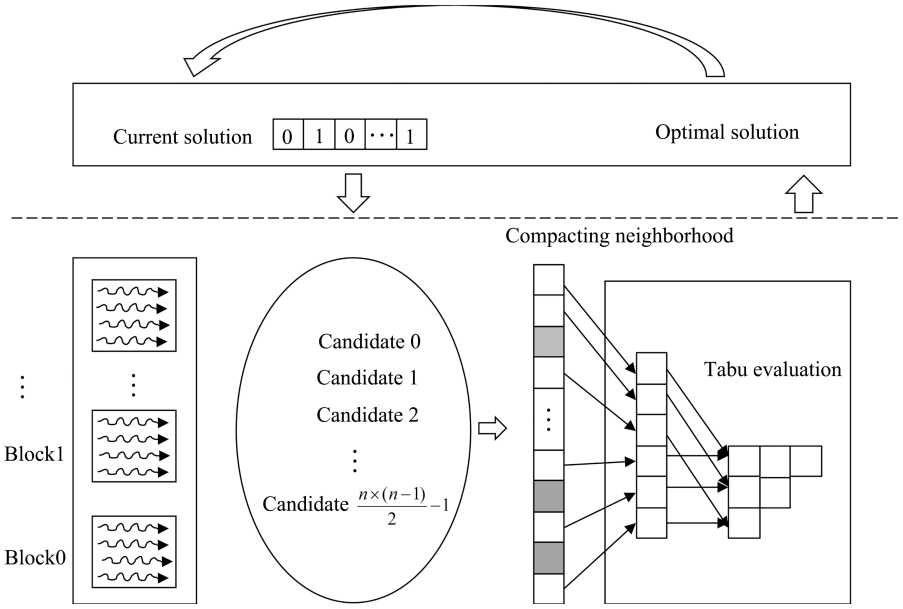

图 2 基于 GPU 的算法框架

Figure 2 GPU-based framework of proposed algorithm

个 GPU 线程对应一个候选解. 第 2 部分为基于 GPU 上的邻域压缩来保留邻域中满足约束 R 的可行解; 第 3 部分为可行解的禁忌评价, 该部分选出满足禁忌条件的最优候选解. 此后, 该候选解被传回主机端. 在主机端完成更新禁忌表、禁忌状态表, 以及当前解的工作; 另外, 如果该候选解优于全局最优解, 也要更新全局最优解. 随后, 算法进入下一次迭代.

在结合 GPU 的体系结构特点进行实现时, 本文方法还需要从以下几个方面考虑.

3.2.2 面向 GPU 的任务图表达

如 2.1 小节所述, 给定的软硬件划分问题的任务图为无向图. 在任务图中, 边上的权值表示相邻节点被划分为不同执行时产生的通信耗费. 由通信耗费的表达式可知, 懒惰执行 (naive implementation) 通信耗费时的计算复杂度为 $O(n^2)$. 因此, 如何保证候选解高效地在 GPU 中执行通信耗费的更新是至关重要的.

高效的通信耗费更新不仅取决于高效的更新算法, 同时也依赖高效的任务图表达. 如果将任务图直接以邻接矩阵的表达方式存放在 GPU 中, 那么即使是本文的增量策略, 在计算通信耗费时仍然存在大量的冗余计算. 这意味着如何在 GPU 中存储任务图也很重要. 在软硬件划分的任务图中, 边的数量通常很稀疏. 因此, 本文采用适合 GPU 存储的 compressed sparse row (CSR) 格式来表示任务图, 图 3 为 CSR 格式表达软硬件划分任务图的一个简单示例.

图中, data 数组保存每条边的通信耗费. row_ptr 数组的大小等于任务图中的节点数目, 数组中的元素表示对应节点的邻接节点的通信耗费在 data 数组中的起始索引. col_index 数组中的元素表示 data 数组中每个通信耗费所对应的节点索引. 因此, 保存整个任务图的存储开销为 $O(V+E)$.

3.2.3 鲁棒的 2 翻转邻域识别

在邻域内, 每个候选解计算各自的软硬件耗费和通信耗费时是相互独立的. 由于本文的自适应策

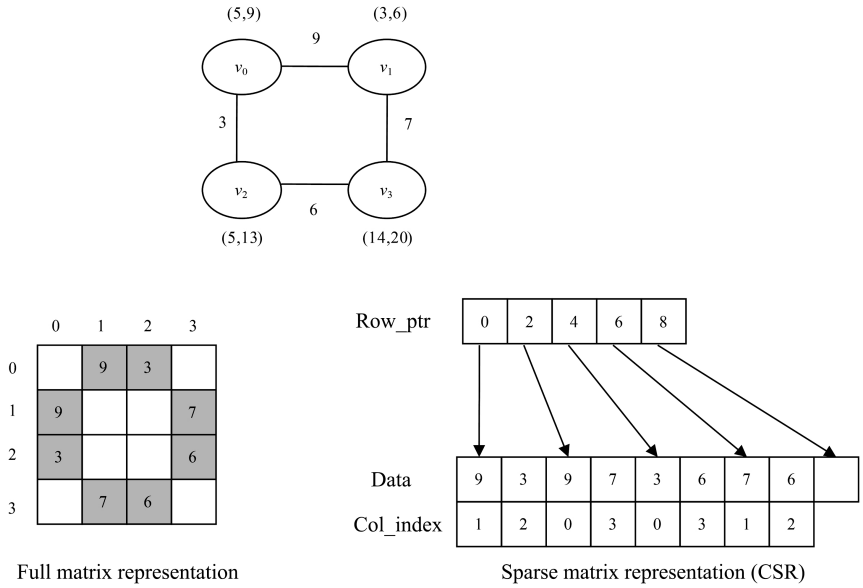

图 3　任务图的 CSR 表达

Figure 3　CSR representation of task graph

略, 使得邻域的规模与问题规模成正比. 这也意味着将邻域内的候选解进行并行化可以显著地降低耗费计算的求解时间.

本文在生成邻域时, 采用的是考虑当前解的所有 2 翻转候选解. 该翻转过程的串行实现语义如图 4(a) 部分的伪代码所示. 然而, 如果自然地对该串行语义进行并行化, 会造成线程间处理的候选解的个数不均衡的情况, 效果如图 4(a) 部分的图例. 在 GPU 中, 虽然线程是 GPU 的基本运行单位, 但是在 GPU 调度时以线程块内连续的 32 个线程组成一个 warp 来调度的. 在执行指令时, 同一个 warp 的线程是同步执行的. 即只有最慢的那个线程执行完毕, 整个 warp 的线程才算工作完成. 其结果是自然的并行化导致线程间处理的候选解个数之差可高达 32 个.

为了使得线程间处理的候选解个数尽可能地均衡化. 本文的并行方式将线程和候选解进行映射, 使得一个线程在逻辑上对应一个候选解. 如果线程数量无限, 则线程的逻辑索引可代表候选解的索引. 在候选解索引的基础上, 进一步通过数学表达式算出生成该候选解时被翻转的两个位置. 该映射方式使线程间处理的候选解个数的均衡化. 具体方式和效果如图 4(b) 所示.

在 GPU 中, 线程以线程网格和线程块两个层次体现. 在具体计算时, GPU 的线程块编号 B_ID, 线程块大小 B_DIM 以及线程块内的线程编号 T_ID 都可以通过内置函数获取, 则邻域候选索引的计算如下式:

$$\text{neibIdx} = \text{B\_ID} \times \text{B\_DIM} + \text{T\_ID}. \tag{8}$$

接下来, 需要通过候选解索引获取该候选解代表的两个翻转位置. 在文献 [42] 中, 作者提出通过邻域编号计算两个翻转位置的方法. 理论上, 该方法能够在 GPU 端求出任意节点规模下对应邻域中的候选 2 翻转位置. 然而实际测试发现, 该方法是否有效取决于两个方面: 一是计算中的数据类型; 二是任务图中的节点规模. 在单精度下, 划分的任务节点规模超过 4608 时, 计算容易产生越界的错误. 而这种错误在本问题中是不允许的. 造成该错误的原因是节点规模增大后, 计算过程中的开根号运算和向下取整运算对浮点数的不当处理造成的. 总的来说, 文献 [42] 的方法在双精度下可以准确计算.

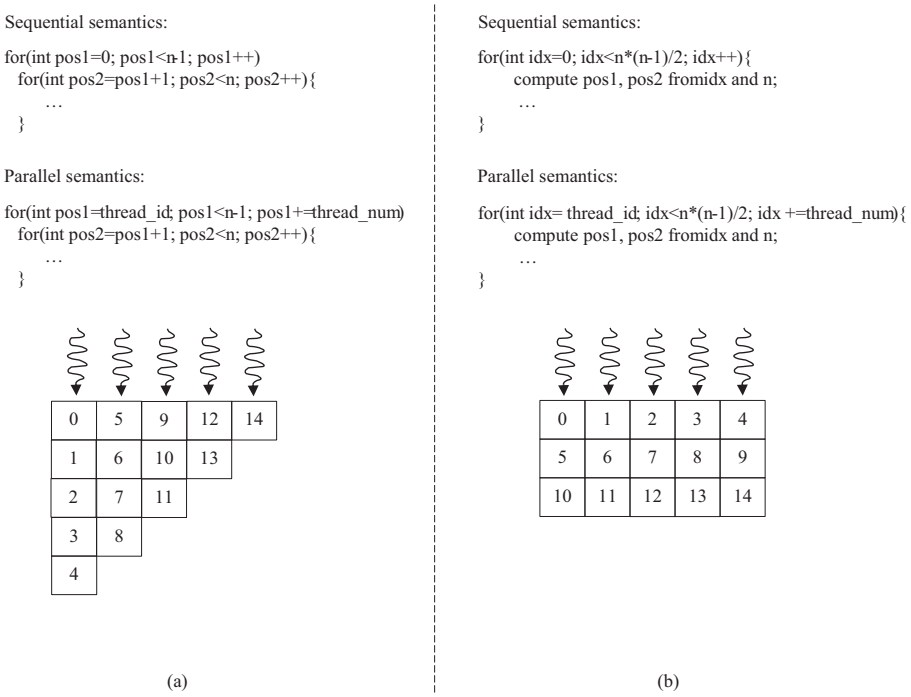

图 4 线程 – 候选解映射

Figure 4 Thread-candidate mapping. (a) Sequential and parallel semantic of naive loop; (b) sequential and parallel semantic of proposed loop

但是, 在 GPU 中尤其是消费级 GPU 中, 双精度运算单元的数量要少于单精度运算单元的数量. 因此, 就 2 翻转计算而言, 文献 [42] 的方法无疑是耗时的.

基于同样的原理, 本文提出一种更鲁棒的 2 翻转计算方法. 在节点规模非常大时, 本文计算方法的优势为在单精度下仍然能准确地计算出候选解的 2 翻转位置. 本文提出的计算方法如下:

$$\text{flip1} = (\text{neibIdx}\%n + \frac{\text{neibIdx}}{n} + 1)\%n, \tag{9}$$

$$\text{flip2} = (\text{flip1} - \frac{\text{neibIdx}}{n} + n - 2)\%n, \tag{10}$$

其中, flip1 和 flip2 分别表示翻转位置 1 和 2. n 表示任务图的节点规模.

最后, 结合 3.2.2 小节的任务图表达和 3.1.2 小节的增量耗费计算方法, 将得到的所有邻域候选解耗费存入全局内存中.

3.2.4 邻域压缩

软硬件划分问题是带约束条件的优化问题. 在计算完邻域解的耗费后, 本文采用邻域压缩方法过滤掉那些不满足约束的邻域解, 保留满足约束的邻域解. 与邻域耗费计算的并行过程不同, GPU 上的邻域压缩是一种隐式的实现.

具体来说, 首先需要在 GPU 端的全局内存分配候选邻域规模大小的空间, 用来保存每个邻域是否满足约束的标识. 如果满足约束, 该邻域设置为 1; 否则设置为 0. 在此基础上, GPU 端的邻域压缩的主要过程分为两步, 即前缀和 (scan) 操作以及散布 (scatter) 操作 [49]. 前缀和操作得到每个可行解

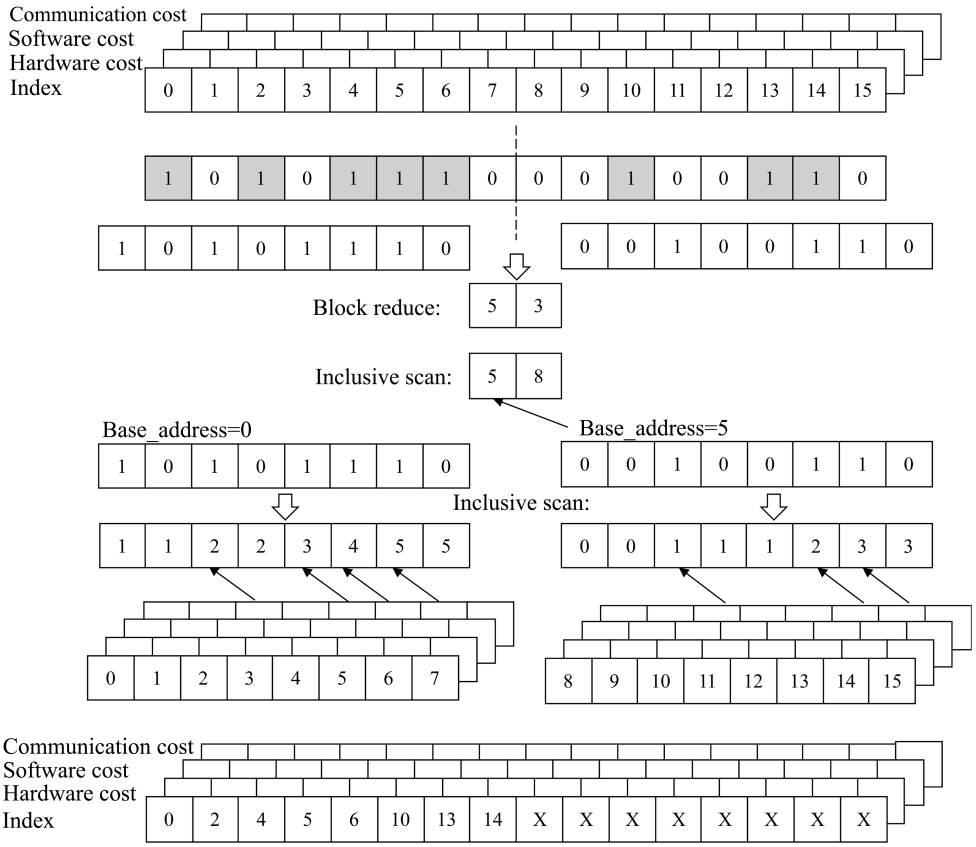

图 5　reduce-scan-scan 模式的 GPU 邻域压缩

Figure 5　GPU compacting neighborhood based on reduce-scan-scan pattern

在目标数组中的位置以及满足约束的候选解的数量. 散布操作将所有可行解压缩输出至对应的位置中. 当所有候选邻域都为可行解时, 邻域压缩退化为前缀和. 图 5 表示整个邻域压缩的流程图.

在已有的工作中, 文献 [49] 提出 SIMD Compaction 并发布了 chag 并行函数库. 文献 [50] 给出了 Compaction 方法另一种的 $O(3n)$ 实现. 这里, n 表示数据规模. $3n$ 则表示算法的执行过程中访问 GPU 全局内存的最少次数. 尽管两个方法本质上都属于 reduce-scan-scan 的实现模式, 通过在标准数据集上的测试, 发现文献 [50] 的实现优于 SIMD Compaction.

为了求解本问题, 本文改进了现有 Compaction 方法, 主要体现在: (1) 保证了算法在计算完邻域耗费时的写全局内存合并; (2) 执行邻域压缩时的读全局内存时的合并访问.

3.2.5　基于 GPU 端的禁忌评价

当挑选出所有满足约束的可行解后, 接下来进行禁忌评价选择当前迭代的最优候选解. 在 GPU 端执行时, 可以将算法 2 看成带禁忌状态数组的最小值比较过程.

因此, 为了在 GPU 中完成禁忌评价, 本文通过融合基于文献 [51] 中的并行规约和禁忌状态表来达到禁忌评价的目的. 并且, 在禁忌评价完成后, 将当前迭代的最优候选解传回 CPU 端.

3.2.6 面向 GPU 的访存模式

GPU 的存储系统的层次结构是影响并行程序执行效率的瓶颈. 算法移植到 GPU 时, 还需要考虑执行过程中的访存特点.

首先, 访问全局内存需要遵循合并的原则. 本问题中, 邻域中的候选解主要包含邻域索引、硬件耗费、软件耗费, 以及通信耗费. 与串行程序采用结构数组 (array of structure, AOS) 不同的是, 在 GPU 中倾向使用数组结构 (structure of array, SOA). 如 3.2.4 小节的图例, 这么做既能保证邻域计算完耗费之后的写全局内存合并, 又能保证在后续邻域压缩阶段的读全局内存合并.

其次, 每个邻域在计算通信耗费时, 对任务图的访问具有不规则性. 而且, 任务图是只读数据. 因此, 任务图的节点关系及对应的通信耗费适合使用纹理内存保存. 此外, 任务图的节点信息也同样使用纹理内存存储.

在邻域耗费计算内核中, 线程的寄存器主要是保存一些临时变量如邻域编号、耗费计算的中间结果等; 另外, 当前解的硬件耗费、软硬耗费和通信耗费会被每个邻域候选解访问. 因此, 可以使用常量内存保存. 同样地, 约束 R 也存放在常量内存中.

关于共享内存的使用, 主要体现在邻域压缩阶段的前缀和操作以及禁忌评价阶段候选解之间的两两比较.

最后, 算法 3 给出基于 GPU 的自适应邻域压缩禁忌搜索的软硬件划分算法的主要过程.

4 实验

4.1 实验基准和运行环境

为验证基于 GPU 的自适应邻域压缩禁忌搜索的软硬件划分算法的有效性. 基准任务和规模按照文献 [11, 12] 提到的方法生成. 表 1 给出所每个任务图的节点数 n、边数 m, 以及相应的问题规模 $\text{size} = 2 \times n + 3 \times m$ 的大小.

任务图中, 每个节点的软件耗费为 $[1, 100]$ 之间的均匀分布的随机数; 硬件耗费为对应节点软件耗费的服从正态分布的随机数; 节点之间的通信耗费为区间 $[0, 2 \cdot \rho \cdot s_{\max}]$ 之间的服从均匀分布的随机数. 其中, $s_{\max}$ 为任务图中单个节点的软件耗费中的最大值; ρ 为通信 – 计算比 (CCR), 当 ρ 取值为 0.1, 1, 10 时, 分别对应计算密集任务、中间型任务, 以及通信密集型任务. 约束 R 的取值分为两种情况: (1) 严格实时约束, 取值为 $[0, \frac{1}{2} \times \sum s_i]$ 之间的均匀分布的随机数; (2) 弱实时约束, 取值为 $[\frac{1}{2} \times \sum s_i, \sum s_i]$ 之间均匀分布的随机数. 因此, 在考虑通信 – 计算比和实时约束 R 的情况下, 每个任务图有 6 种情况.

开发工具为 Visual Studio 2012, 使用 C++ 实现, 操作系统为 Windows 10. 算法的并行部分用 CUDA 8.0 实现. 并行平台为 Nvidia GTX 780 GPU. 该 GPU 的架构为 KEPLER GK110 架构. GPU 上有 12 个流多处理器 (stream multi-processpr, SM), 每个 SM 有 192 个标量处理器 (scalar processor, SP), GPU 的时钟频率最大为 1.059 GHz, 全局内存大小为 3 GB. 算法的串行部分用 C++ 实现, 串行部分的运行平台为 Intel i7-4770 CPU. 该 CPU 的时钟频率为 3.4 GHZ, 内存大小为 16 GB.

4.2 比较方法

本文首先通过与已有方法的在解质量上的比较来说明本文方法的优势. 然后, 在运行时间上, 本文通过加速比来说明本文算法的串行实现和 GPU 实现的差异. 为了进一步说明执行一系列优化策略

算法 3 GPU-based adaptive compacting neighborhood tabu search for hardware/software partitioning

Require: Initial solution, task graph, node information;
Ensure: Optimal solution;
  Initialize the current solution and global optimal;
  Initialize the tabu list at host side;
  Initialize the tabu status at host side;
  Allocate the node cost of task graph space on GPU memory;
  Allocate the neighborhood space on GPU memory;
  Bind the task graph on GPU texture memory;
  Transfer the node cost of task graph to GPU memory;
  while stop criterion isn't satisfied **do**
    Copy the current solution to GPU memory;
    Compute the costs of candidates in the neighborhood at GPU side;
    Compating neighborhood at GPU side;
    if the number of feasible solutions is 0 **then**
      Reinitialize the current solution;
      goto Compute the costs of candidates in the neighborhood at GPU side;
    end if
    Launch tabu evaluations kernel;
    Transfer back the optimal candidate to the host side;
    if all feasible solutions are tabu **then**
      Randomly choose a feasible solution;
      Update the optimal candidate;
    end if
    Update the tabu list at the host side;
    Update the tabu status table at the host side;
    Replace the current solution with the optimal candidate solution at the host side;
    if $H(\boldsymbol{x}_{\text{current}}) \leqslant H(\boldsymbol{x}_{\text{global}})$ **then**
      Update the global solution;
      Clear $\text{count}_{\text{noimprovement}}$;
    else
      Increase $\text{count}_{\text{noimprovement}}$ by 1;
    end if
  end while

的效果, 本文还给出了将提出算法的串行实现自然地移植到 GPU 后的运行时间. 通过两者之间的比较体现出在本问题中, 将算法移植到 GPU 所做的一系列优化执行的必要性. 最后, 将任务的节点规模进一步扩大, 即在节点规模为 10000 的情况下, 通过运行时间的比较体现 GPU 用在更大规模软硬件划分中的优势.

4.3 算法的解质量的实验结果

图 6(a)∼(f) 分别比较了文献 [44] 中的 HEUR 和 TABU 方法, 文献 [47] 中的 NODERANK 方法, 以及本文的自适应邻域压缩禁忌搜索算法的串行实现和 GPU 实现 (简写为 ACNTS 和 GACNTS) 的解质量, 在 6 种通信 – 计算比和约束情况下, 相对文献 [45] 的 alg-new3 算法的解质量的改进程度. 本文方法的初始解由 HEUR 算法得到, 算法的停止条件为达到最大迭代周期 M 或者全局解连续 N 个周期都没有改进. 其中, M 和 N 的取值分别设定为 2000 和 200. NODERANK 的迭代次数同文献 [47],

表 1 测试基准

Table 1 Benchmark

Name	n	m	Size
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

设定为 4 次. alg-new3 中的步长 d_x 同文献 [45], 设定为 0.05. 关于算法 B 的硬件耗费相对算法 A 的硬件耗费的改进程度, 计算公式为

$$\text{Improvement} = \frac{H(\boldsymbol{x}_A) - H(\boldsymbol{x}_B)}{H(\boldsymbol{x}_A)} \times 100. \tag{11}$$

尽管在计算 – 通信比为 10, 严格实时约束下, NODERANK 方法在部分任务图中的改进程度比本文方法更好. 但从整体上看, ACNTS 和 GACNTS 算法对 alg-new3 方法的解质量改进程度, 明显优于文献中的 HEUR、TABU 以及 NODERANK 算法. 另外, 在所有 6 种情况下, 本文方法在前 3 个任务图中都能得到明显的质量改进. 这主要说明了在前 3 个任务图中, 由于节点规模最大仅有 26 个, 此时邻域中的候选解仅有数百个. 在邻域压缩阶段, 如果候选解都不满足约束, 会通过随机初始化方法从其他当前解开始搜索. 而解质量的明显改进也说明了重新初始化策略是有效的.

表 2 为在 6 种情况下, 各个算法相对文献 [45] 的 alg–new3 算法的硬件耗费的平均改进程度在所有任务图中的整体比较. 通过对结果进行对比来看, 同样是 HEUR 得到的初始解, 本文提出方法的串行实现和 GPU 实现对初始解的优化相对于文献 [44] 的 TABU 方法改进程度更明显, 也优于文献 [47] 中 NODERANK 方法对 alg–new3 方法的改进程度. 这也说明了就本问题, 本文方法在普遍情况下, 能得到更优的解质量.

最后, 图 7 具体给出了在不同情况下的任务图中, 本文方法的 GPU 执行对 HEUR 方法得到的初始解的改进程度以体现本文方法对整个解质量的贡献. 可以看出, 在计算 – 通信比为 0.1, 弱实时约束下. 本文方法体现出了明显优势. 在其他情况下, 本文方法的改进程度相对于文献 [44] 给出的 TABU 方法的优化程度也更好.

4.4 算法的 GPU 实现的实验结果

为了使提出方法能高效地在 GPU 上运行, 本文提出了基于 GPU 的任务图表达、线程 – 候选解

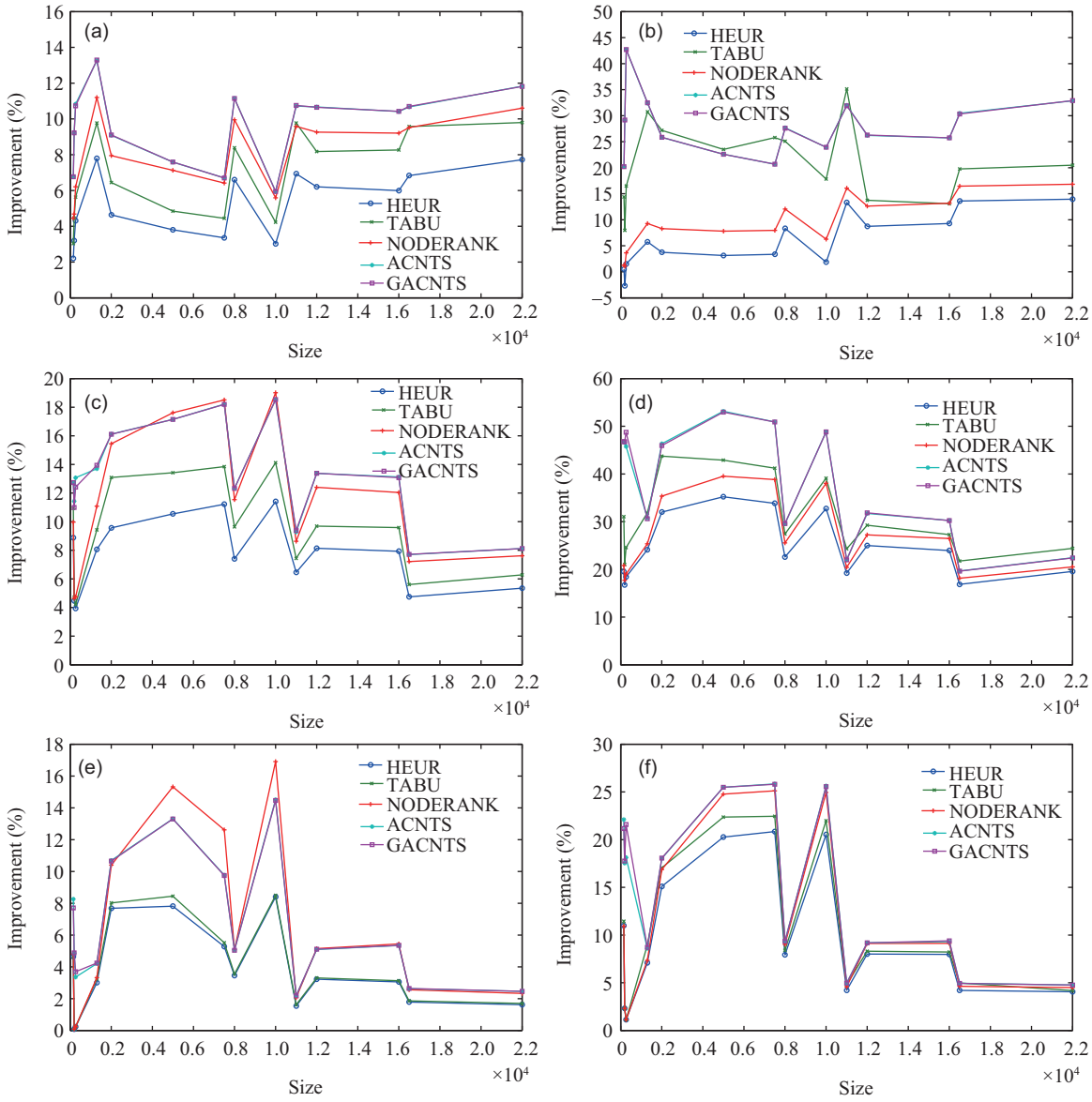

图 6 (网络版彩图) 算法在不同的通信 − 计算比以及约束条件下对 30 个实例取平均值的硬件耗费

Figure 6 (Color online) Average hardware cost over 30 random instances by different algorithms under different CCR and R. (a) CCR = 0.1, R = low; (b) CCR = 0.1, R = high; (c) CCR = 1, R = low; (d) CCR = 1, R = high; (e) CCR = 10, R = low; (f) CCR = 10, R = high

映射、数据布局和访存等一系列优化策略. 为体现这些优化策略的意义, 本文给出了将 ACNTS 自然地移植到 GPU 后的运行时间. 在自然移植时, GPU 端使用邻接矩阵存储任务图, 而 ACNTS 实现的任务图表达为邻接表表达; 候选解的并行化采用图 4(a) 的方式; 计算耗费时保留增量策略, 但候选解的软硬件耗费和通信耗费的存储不考虑全局内存读写的合并; 邻域压缩阶段采用 $4n$ 全局内存访问量的实现; 禁忌评价阶段的并行规约部分不做访存优化. 图 8 包括本文方法的串行实现 (ACNTS)、自然移植到 GPU 上的实现 (naive–GPU)、执行优化策略的 GPU 实现 (GACNTS) 和文献 [44] 的 TABU 方法在运行时间上的对比. 运行时间为每种情况下, 每个任务图运行 30 次后取平均值. 横坐标表示任

表 2 算法对 alg-new3 改进程度的比较

Table 2 Comparison among improvements of algorithm over alg-new3

CCR	R	TABU	NODERANK	ACNTS	GACNTS
0.1	low	5.2	7.9	**9.6**	**9.6**
0.1	high	20.8	11.3	**27.8**	**27.8**
1	low	9.4	11.5	13.2	**13.7**
1	high	30.7	26.6	**37.5**	37.4
10	low	3.8	6.2	**6.5**	**6.5**
10	high	10.5	11.1	14.4	**14.8**

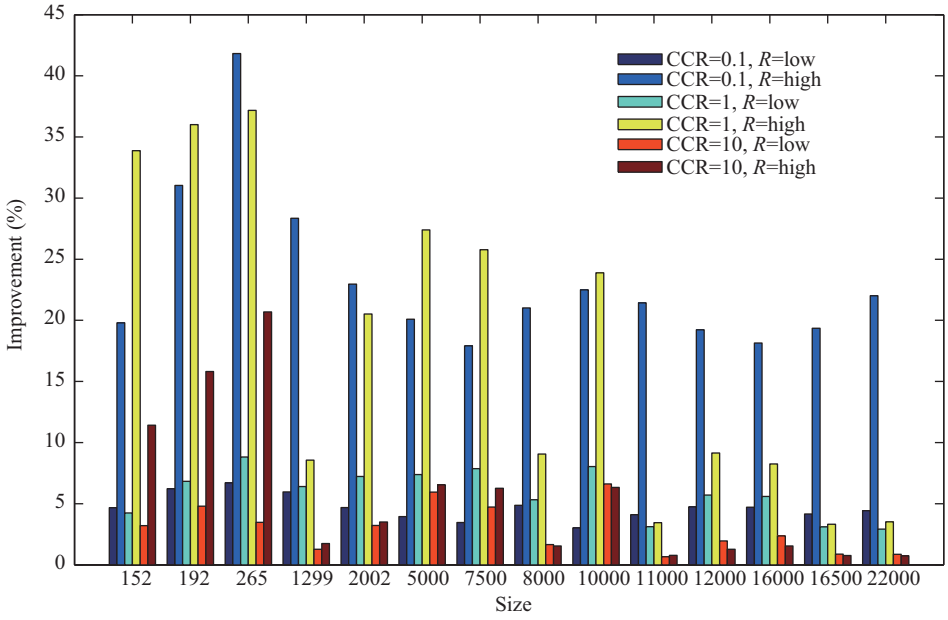

图 7 (网络版彩图) GACNTS 对 HEUR 方法在每种情况下的的平均改进

Figure 7 (Color online) Average improvement of GACNTS over HEUR in each case

务图的规模, 纵坐标为对数方式表达的运行时间 (单位: ms). 对于 GPU 上邻域候选解的增量计算的线程配置统一为 128 个线程块以及每个线程块的线程为 512 个. 在邻域压缩阶段, 维持文献 [50] 中的线程配置参数不变. 而对于可行解的禁忌评价部分, 本文仅用一个线程块实现, 线程数量为 1024.

目前, 常用术语 “加速比”, 即整个算法的串行执行时间与 GPU 加速后的执行时间的比值来说明经过 GPU 加速后的优势. 本文继续沿用加速比的概念对串行执行和 GPU 执行的运行时间进行分析. 算法的运行时间主要考虑每次迭代时涉及到的运算. 对于必要的显存初始化以及仅有一次数据传输过程的时间开销不做计时考虑. 另外, 按照节点生成的邻域规模大小, 进一步将 14 个任务图分为 3 种规模. 即 crc32, patricia, dijkstra 为小规模任务图; clustering 和 rc6 为中等规模任务图; random1 ∼ random9 为大规模任务图.

关于 ACNTS 和 GACNTS 之间的比较, 在所有的 6 种情况下, ACNTS 方法执行前 3 个小规模任务图的优化更有优势. 此时的加速比范围在 0.55 ∼ 1.06 之间. 在通信 – 计算比为 10, 严格实时约束时, 算法的运行时间增加明显. 因为在该情况下, 当前解形成的所有 2 翻转邻域候选解都不满足约束.

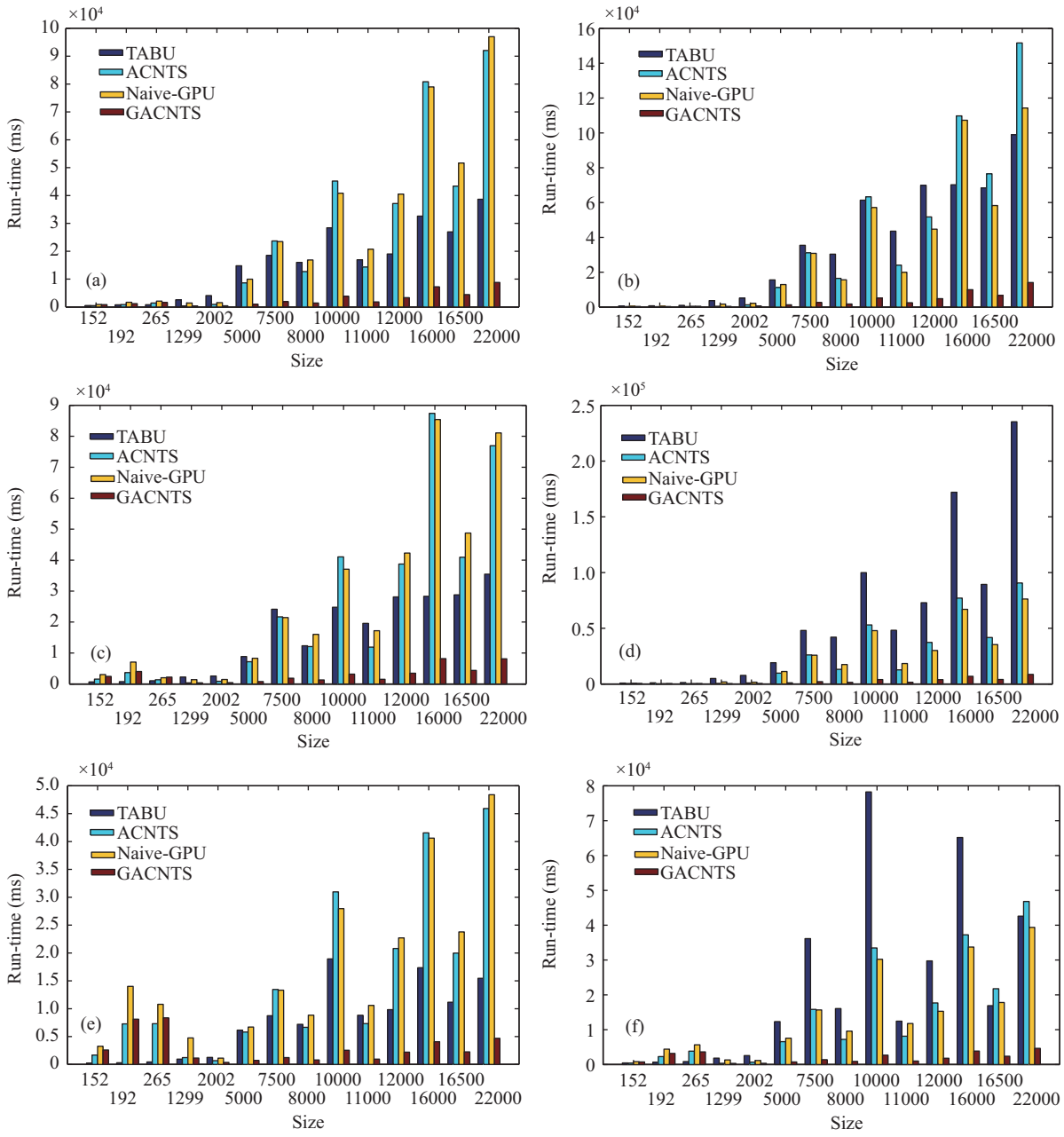

图 8 (网络版彩图) 提出算法的串行执行与 GPU 执行的运行时间比较

Figure 8 (Color online) Comparison of excution time between sequential implemtation and GPU implementation. (a) CCR = 0.1, R = low; (b) CCR = 0.1, R = high; (c) CCR = 1, R = low; (d) CCR = 1, R = high; (e) CCR = 10, R = low; (f) CCR = 10, R = high

因此, 需要反复地对当前解实施重新初始化策略并判断新的当前解的邻域可行性. 尽管重新初始化当前解带有随机性, 但是当节点规模较小时, 重新初始化后的当前解的邻域仍然不能保证有至少一个满足约束的可行解, 因此造成了该过程的重复执行次数过多, 使得总的执行时间增加. 对于中等规模的任务图, 加速比的范围在 1.06 ~ 2.3 之间. 当规模为 1299 时, 加速比在 1.52 以内. 当节点规模为 2002

表 3 节点数量为 10000 时, 本文方法的串行执行与 GPU 执行的运行时间 (分钟)

Table 3 Running time of serial execution and GPU-based execution when the number of nodes is 10000 (min)

n	m	ACNTS	GACNTS
10000	10000	102	9
10000	20000	148	12
10000	30000	215	16

时, 加速比的范围在 1.85 ∼ 2.3 之间. 此时基于 GPU 的方法开始体现出了速度上的优势. 对于剩下的大规模任务图, 加速比的范围在 7.71 ∼ 13.48 之间. 毫无疑问, GACNTS 在速度上的优势体现了出来.

从图 8 中可以看出, 将 ACNTS 自然移植到 GPU (naive–GPU) 的表现, 仅相当于 ACNTS 的运行时间. 这说明仅依靠 GPU 本身的并行特性无法对本文的方法提供更好的性能支持. 最主要的原因在于自然的 GPU 端任务图表达和候选解的并行化方法造成了通信耗费的大量冗余计算和线程间的候选解个数不均衡. 这也体现了本文对算法移植到 GPU 上所做的一系列优化策略的必要性. 最重要的是, 本文方法在实行 GPU 优化策略后, 使得 GACNTS 的性能不仅得到提升, 在绝大多数情况下要快于文献 [44] 的 TABU 方法. 就本问题而言, GACNTS 在求解质量和运行时间上都超过了已有的禁忌搜索求解软硬件划分方法.

4.5 更大规模下的软硬件划分问题

本部分将节点规模扩大至 10000 用于验证基于 GPU 的软硬件划分方法的优势. 该节点数量下, 生成的 2 翻转邻域规模要远远高于已有基准节点数量下的邻域规模. 表 3 为算法的运行时间. 可以看出, 在停止准则不变的情况下, 本文方法的串行执行时间最长超过 3 个小时, 而通过 GPU 加速后的执行时间不到 20 分钟. 这也进一步体现了将 GPU 用在更大规模的软硬件划分中的意义.

5 结论和未来工作

本文提出一种基于 GPU 的自适应邻域压缩 (compacting neighborhood) 禁忌搜索的软硬件划分算法. 在相关基准任务图上, 通过不同的计算 – 通信比和实时约束, 对提出的方法进行验证. 结果表明, 本文方法的求解质量要优于已有的启发式方法和禁忌搜索方法. 在大规模任务图中, 本文方法的 GPU 实现明显具有速度上的优势. 通过将自适应策略的禁忌搜索方法自然移植到 GPU, 并和该方法的串行实现以及执行优化策略的 GPU 实现进行运行时间的对比, 体现了本文工作中一系列基于 GPU 的优化策略的必要性. 最后, 在节点规模为 10000 时验证了基于 GPU 的软硬件划分方法在时间上的明显优势.

致谢 在此, 我们对给予本文的工作提出宝贵修改意见和建议的匿名审稿人表示感谢.

参考文献

1 Teich J. Hardware/software codesign: the past, the present, and predicting the future. Proc IEEE, 2012, 100: 1411–1430

2 Sha E, Wang L, Zhuge Q F, et al. Power efficiency for hardware/software partitioning with time and area constraints on MPSoC. Int J Parallel Prog, 2015, 43: 381–402

3 Lin G, Zhu W X, Ali M M. A tabu search-based memetic algorithm for hardware/software partitioning. Math Probl Eng, 2014: 1–16

4 Zhang Y D, Wu L N, Wei G, et al. Hardware/software partition using adaptive ant colony algorithm. Control Decis, 2009, 24: 1385–1389 [张煜东, 吴乐南, 韦耿, 等. 基于自适应蚁群算法的软硬件划分. 控制与决策, 2009, 24: 1385–1389]

5 Erbas C, Cerav-Erbas S, Pimentel A D. Multi-objective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Trans Evolut Comput, 2006, 10: 358–374

6 Luo S Q, Ma X X, Lu Y. An advanced non-dominated sorting genetic algorithm based SOC hardware/software partitioning. Acta Electron Sin, 2009, 37: 2595–2599 [罗胜钦, 马萧萧, 陆忆. 基于改进的 NSGA 遗传算法的 SOC 软硬件划分方法. 电子学报, 2009, 37: 2595–2599]

7 Luo L, Xia J, He H J, et al. Effective multi-objective genetic algorithm for hardware-software partitioning. Comput Sci, 2010, 37: 275–279 [罗莉, 夏军, 何鸿君, 等. 一种有效的面向多目标软硬件划分的遗传算法. 计算机科学, 2010, 37: 275–279]

8 Nath P K, Datta D. Multi-objective hardware/software partitioning of embedded systems: a case study of JPEG encoder. Appl Soft Comput, 2014, 15: 30–41

9 Shi W J, Wu J G, Lam S, et al. Algorithms for bi-objective multiple-choice hardware/software partitioning. Comput Electron Eng, 2016, 50: 127–142

10 Wang R, Hung W, Yang G W, et al. Uncertainty model for configurable hardware/software and resource partitioning. IEEE Trans Comput, 2016, 66: 3217–3223

11 Arato P, Juhasz S, Mann Z A, et al. Hardware-software partitioning in embedded system design. In: Proceedings of IEEE International Symposium on Intelligent Signal Processing, Budapest, 2003. 197–202

12 Arato P, Mann Z A, Orban A. Algorithmic aspects of hardware/software partitioning. ACM Trans Design Autom Electron Syst, 2005, 10: 136–156

13 Zhu J F, Wu J G, Shi W J, et al. Computign model and dynamic programming algorithm for multi-choice hardware/software partitioning on MPSoC. Comput Eng Sci, 2015, 37: 41–46 [朱峰军, 武继刚, 史雯隽, 等. 多选择软硬件划分问题的计算模型与动态规划算法. 计算机工程与科学, 2015, 37: 41–46]

14 Jiang G Y, Wu J G, Lam S K, et al. Algorithmic aspects of graph reduction for hardware/software partitioning. J Supercomput, 2015, 71: 2251–2274

15 Ma Y C, Zhang C, Luk W. Hybrid two-stage HW/SW partitioning algorithm for dynamic partial reconfigurable FPGAs. J Tsinghua Univ: Nat Sci Ed, 2016, 56: 246–252 [马昱春, 张超, Luk W. 基于混合式两阶段的动态部分重构 FPGA 软硬件划分算法. 清华大学学报 (自然科学版), 2016, 56: 246–252]

16 Ma Y C, Liu J L, Zhang C, et al. HW/SW partitioning for region-based dynamic partial reconfigurable FPGAs. In: Proceedings of the 32nd International Conference on Computer Design, Seoul, 2014. 470–476

17 Chatha K S, Vemuri R. Hardware-software partitioning and pipelined scheduling of transformative applications. IEEE Trans VLSI Syst, 2002, 10: 193–208

18 Wu J G, Chang B F, Srikanthan T. A hybrid branch-and-bound strategy for hardware/software partitioning. In: Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science, Shanghai, 2009. 641–644

19 Gupta R K, De Micheli G. Hardware-software co-synthesis for digital systems. IEEE Des Test Comput, 1993, 10: 29–41

20 Ernst R, Henkel J, Benner T. Hardware-software co-synthesis for microcontrollers. IEEE Des Test Comput, 1993, 10: 64–75

21 Mishra A, Vakharia D, Hati A J, et al. Hardware software partitioning of task graph using genetic algorithm. In: Proceedings of Recent Advances and Innovations in Engineering, Jaipur, 2014. 1–5

22 Ferrandi F, Lanzi P L, Pilato C, et al. Ant colony optimization for mapping, scheduling and placing in reconfigurable systems. In: Proceedings of IEEE NASA/ESA Conference on Adaptive Hardware and Systems, Torino, 2013. 47–54

23 Abdelhalim M B, Habib S E D. An integrated high-level hardware/software partitioning methodology. Des Autom Embed Syst, 2011, 15: 19–50

24 Henkel J, Ernst R. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. IEEE Trans VLSI Syst, 2001, 9: 273–289

25 Zhu W B, Jin T B, Yin J Y. Hardware/software partitioning of RISP based on improved simulated annealing algorithm. Comput Measur Contrl, 2014, 22: 2991–2993 [朱闻博, 金同标, 殷进勇. 基于改进模拟退火的 RISP 软硬件划分. 计算机测量与控制, 2014, 22: 2991–2993]

26 Zhang Y, Luo W, Zhang Z, et al. A hardware/software partitioning algorithm based on artificial immune principles. Appl Soft Comput, 2008, 8: 383–391

27 Koudil M, Benatchba K, Tarabet A, et al. Using artificial bees to solve partitioning and scheduling problems in co-design. Aool Math Comput, 2007, 186: 1710–1722

28 Wiangtong T, Cheung P Y K, Luk W. Comparing three heuristic search methods for functional partitioning in hardware/software co-design. Des Autom Embed Syst, 2002, 6: 425–449

29 Xiong Z H, Li S K, Chen J H. Hardware/software partitioning based on dynamic combination of genetic algorithm and ant algorithm. J Soft, 2005, 16: 503–512 [熊志辉, 李思昆, 陈吉华. 遗传算法与蚂蚁算法动态融合的软硬件划分. 软件学报, 2005, 16: 503–512]

30 Liu A, Feng J F, Liang X L, et al. Algorithm of hardware/software partitioning based on genetic particle swarm optimization. J Comput-Aided Design Comput Graph, 2010, 22: 927–933 [刘安, 冯金富, 梁晓龙, 等. 基于遗传粒子群优化的嵌入式系统软硬件划分算法. 计算机辅助设计与图形学学报, 2010, 22: 927–933]

31 Jiang Y, Zhang H, Jiao X, et al. Uncertain model and algorithm for hardware/software partitioning. In: Proceesings of IEEE Computer Society Annual Symposium on VLSI, Amherst, 2012. 243–248

32 Li G S, Feng J F, Wang C, et al. Hardware/software partitioning algorithm based on the combination of genetic algorithm and tabu search. Eng Rev, 2014, 34: 151–160

33 Farahani A F, Kamal M, Salmani-Jelodar M. Parallel genetic algorithm based HW/SW partitioning. In: Proceedings of International Symposium on Parallel Computing in Electrical Engineering, Bialystok, 2006. 337–342

34 Wu Y, Zhang H, Yang H B. Research on parallel HW/SW partitioning based on hybrid PSO algorithm. In: Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing, Taipei, 2009. 449–459

35 Yan X H, He F Z, Chen Y L. A parallel hardware/software partitioning method based on conformity particle-swarm optimization with harmony search. Sci Sin Inform, 2016, 46: 1321–1338 [鄢小虎, 何发智, 陈壹林. 一种基于从众和声粒子群算法的并行软硬件划分方法. 中国科学: 信息科学, 2016, 46: 1321–1338]

36 Owens J D, Houston M, Luebke D, et al. GPU computing. Proc IEEE, 2008, 96: 879–899

37 Van Luong T, Melab N, Talbi E G. GPU computing for parallel local search meta-heuristic algorithms. IEEE Trans Comput, 2013, 62: 173–185

38 Zhu W, Curry J, Marquez A. SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration. Int J Prod Res, 2010, 48: 1035–1047

39 Czapiński M. An effective parallel multi-start tabu search for quadratic assignment problem on CUDA platform. J Parallel Distr Comput, 2013, 73: 1461–1468

40 Czapiński M, Barnes S. Tabu search with two approaches to parallel flow-shop evaluation on CUDA platform. J Parallel Distr Comput, 2011, 71: 802–811

41 Bukata L, Sucha P, Hanzálek Z. Solving the resource constrained project scheduling problem using the parallel tabu search designed for the CUDA platform. J Parallel Distr Comput, 2015, 77: 58–68

42 Zhu F J, Wu J G, Song Q Z. Heuristic algorithm comparisons for multi-choice hardware/software partitioning. Comput Applic Soft, 2015, 32: 215–219 [朱峰军, 武继刚, 宋庆增. 多选择软硬件划分问题的启发式算法比较. 计算机应用与软件, 2015, 32: 215–219]

43 Wu J G, Thambipillai S, Tao J. Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design. Des Autom Embed Syst, 2008, 12: 345–375

44 Wu J G, Wang P, Lam S K, et al. Efficient heuristic and tabu search for hardware/software partitioning. J Supercomput, 2013, 66: 118–134

45 Wu J, Srikanthan T, Chen G. Algorithmic aspects of hardware/software partitioning: 1D search algorithms. IEEE Trans Comput, 2010, 59: 532–544

46 Quan H J, Zhang T, Liu Q, et al. Comments on "algorithmic aspects of hardware/software partitioning: 1D search algorithms". IEEE Trans Comput, 2014, 4: 1055–1056

47 Chen Z, Wu J G, Song G Z, et al. NodeRank: an efficient algorithm for hardware/software partitioning. Chin J Comput, 2013, 36: 2033–2040 [陈志, 武继刚, 宋国治, 等. NodeRank: 一种高效软硬件划分算法. 计算机学报, 2013,

36: 2033–2040]
48 Glover F, Taillard E. A user's guide to tabu search. Ann Oper Res, 1993, 41: 1–28
49 Billeter M, Olsson O, Assarsson U. Efficient stream compaction on wide SIMD many-core architectures. In: Proceedings of the Conference on High Performance Graphics, New Orleans, 2009. 159–166
50 Wilt N. The CUDA Handbook: a Comprehensive Guide to GPU Programming. Upper Saddle River: Pearson Education, 2013. 385–419
51 Harris M. Optimizing parallel reduction in CUDA. NVIDIA Developer Technology. 2007. http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

GPU-based adaptive compacting neighborhood tabu search for hardware/software partitioning

Neng HOU[1,2], Fazhi HE[1,2*], Yi ZHOU[3] & Yilin CHEN[1,2]

1. *State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072*, China*;
2. *School of Computer Science, Wuhan University, Wuhan* 430072*, China*;
3. *School of Information Science and Engineering, Wuhan University of Science and Technology, Wuhan* 430081*, China*
* Corresponding author. E-mail: fzhe@whu.edu.cn

Abstract Hardware/software (HW/SW) partitioning is an essential step in hardware/software co-design as it determines the functions to be implemented in the hardware and software. HW/SW partitioning problems are NP hard. The increasing complexities of modern embedded systems worsen the problem, thus motivating the search for solutions by heuristics. The tabu search algorithm is an effective method for HW/SW partitioning. However, the process is time consuming. The existing tabu search methods for HW/SW partitioning focus on sequential implementations that involve a trade-off between solution time and solution quality. This trade-off sacrifices the solution quality. This paper presents a GPU-based adaptive compacting neighborhood tabu search for HW/SW partitioning. First, we present an adaptive strategy that enhances the search intensification to improve the solution quality. The massive parallelism of GPU architecture can reduce the solution time of the proposed strategy. Next, to ensure that the algorithm execute efficiently on the GPU, we further propose several implementation strategies such as the representation of task graph on the GPU, the mapping between the GPU thread and candidate, and data layout and memory access optimization. Finally, we realize the proposed method in a computing unified device architecture, namely CUDA, and verify the effectiveness according to the related benchmark with different communication-to-computation ratios and real-time constraint requirements. The results show that the proposed method can yield a better solution quality compared to the existing methods. By comparing with the naive GPU implementation of adaptive compacting neighborhood tabu search, the proposed implementation strategies on the GPU significantly reduced the solution time. In addition, we verified the advantage of the GPU-based method for very large HW/SW partitioning.

Keywords hardware-software codesign, heuristic methods, graphics processing unit, tabu search, adaptive algorithms

Neng HOU was born in 1983. He received his B.S. degree from Huazhong University of science and technology in 2008, and his M.S. degree from Guangxi University in 2012. Currently, he is a Ph.D. candidate in the School of Computer Science, Wuhan University. His research interests include hardware/software co-design, artificial intelligence, and GPU computing.

Yi ZHOU was born in 1983. He received his Ph.D. degree in Wuhan University in 2016. Currently, he is a lecturer in the School of Information Science and Engineering, the Wuhan University of Science and Technology. His research interests include artificial intelligence and parallel computing.

Fazhi HE was born in 1968. He received his B.S., M.S., and Ph.D. degrees from the Wuhan University of Technology, China. He was a post-doctor in Zhejiang University, a visiting researcher in the Korea Institute of Science and Technology, and a visiting faculty member at the University of North Carolina at Chapel Hill. Currently, he is a professor in the School of Computer Science and Technology, Wuhan University. His research interests are hardware/software partitioning, computer-aided design, computer graphics, and collaborative computation.

Yilin CHEN was born in 1989. He is currently a Ph.D. candidate in the School of Computer Science, Wuhan University. His research interests include computer graphics, machine learning, and GPU computing.