软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(2):298−313 [doi: 10.13328/j.cnki.jos.004527]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

GPU 上两阶段负载调度问题的建模与近似算法*

孙景昊, 邓庆绪, 孟亚坤

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

通讯作者: 邓庆绪, E-mail: dengqx@mail.neu.edu.cn

摘 要: 随着硬件功能的不断丰富和软件开发环境的逐渐成熟,GPU(graphics processing unit)越来越多地被应用到通用计算领域,并对诸多计算系统(尤其是嵌入式系统)性能的显著提升起到了至关重要的作用.在基于 GPU 的计算系统中,大规模并行负载同时进行数据传输和加载的情况时常发生,数据传输延时在系统性能全局最优化中变得不容忽视.综合考虑负载的传输时间和执行时间,以总负载 makespan 最小化作为系统性能的全局优化目标,研究了 GPU 上负载"传输-执行"联合调度问题.首先,将负载的时间信息和并行任务数与矩形域的二维空间联系起来,建立了负载的 2D 双层矩形域模型;然后,将 GPU 上负载调度问题归结为一类 Strip-Packing 问题;最后,基于贪婪策略给出了近似度为 3 的多项式时间近似算法,算法复杂度为 $O(n\log n)$.该近似算法的核心是对数据传输阶段进行负载排序调度.这从理论层面上证明了 GPU 系统采取"传输-执行"两阶段调度的有效性,即,在数据传输阶段采取负载排序调度,在负载执行阶段采取先来先服务(first-come-first-serve,简称 FCFS)调度,能够使 GPU 性能达到全局最优或近似最优.

关键词: GPU(graphics processing unit);数据传输;负载排序;strip-packing;近似算法

中图法分类号: TP316 **文献标识码**: A

中文引用格式: 孙景昊,邓庆绪,孟亚坤.GPU 上两阶段负载调度问题的建模与近似算法.软件学报,2014,25(2):298−313. http://www.jos.org.cn/1000-9825/4527.htm

英文引用格式: Sun JH, Deng QX, Meng YK. Two-Stage workload scheduling problem on GPU architectures: Formulation and approximation algorithm. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):298−313 (in Chinese). http://www.jos.org.cn/1000-9825/4527.htm

Two-Stage Workload Scheduling Problem on GPU Architectures: Formulation and Approximation Algorithm

SUN Jing-Hao, DENG Qing-Xu, MENG Ya-Kun

(School of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

Corresponding author: DENG Qing-Xu, E-mail: dengqx@mail.neu.edu.cn

Abstract: With the prevalence of general purpose computation, GPUs (graphics processing units) are becoming extremely important to significantly improve system performances for many computing systems, including embedded systems. Running massively parallel kernels on GPUs is challenging for system's overall performance especially when large amount of workloads (kernels) are running together. This paper investigates how to schedule large amount of workloads that have to be executed on GPUs to minimize the makespan of all workloads to improve the system overall performance. By considering the transfer time and execution time together, the study makes an abstraction for each workload and formulate the scheduling problem on GPUs into a 2D rectangular strip-packing model. A polynomial 3-approxiamation algorithm is proposed to solve the strip-packing problem. The approximation results exhibit an effective approach for workload sequencing during the data offloading on GPUs. It also implies that the scheduling jointed by workload sequencing

* 基金项目: 国家自然科学基金(61300194); 国家教育部博士点基金(20110042110021); 国家科技支撑计划(2012BAK24B01); 河北省自然科学基金(F2013501048)

收稿时间: 2013-05-06; 定稿时间: 2003-09-29

for GPUs data offloading and first-come-first-serve (FCFS) scheduling inside GPUs with workload conserving can improve the system performance optimally or near-optimally.

Key words: GPU (graphics processing unit); data transfer; workload sequencing; strip-packing; approximation algorithm

随着半导体工艺的日臻成熟,处理器芯片上集成的晶体管越来越多,目前已突破 10 亿量级,图形处理器(graphics processing unit,简称 GPU)的性能也因此得到飞速提升,并远远超过通用 CPU[1].2012 年,GPU 集群已成为世界 TOP 500 超级计算机中主要的计算资源.GPU 优良的计算性能,一方面得益于核心运算单元独享内存容量和内核数量的增长,另一方面,更依赖于并行计算平台中负载调度策略的程序实现.

CUDA 和 OpenCL 是 GPU 系统中应用最广泛的并行计算平台.在这两类经典程序设计范式中,负载被定义为 GPU 上运行的一个或多个并行程序段(内核),并关联有程序运行时所需的数据资源.在 GPU 集群中,负载一旦被分配到某个 GPU 上,该负载就只能访问这个 GPU 独享的内存,而且,负载必须在其关联数据加载到独享内存后才能开始处理和执行.在过去的几十年间,主存和 GPU 独享内存之间数据传输带宽的提升速度一直落后于 GPU 内核数量的增长速度,数据传输延时问题变得不容忽视.特别是在实时系统中[2,3],大规模并行负载同时加载到 GPU 上时,主存和独享内存间的数据传输时间成为影响系统全局性能的重要参数.目前,考虑数据传输的 GPU 内核负载调度还处于较低的自动化水平,这成为制约 GPU 系统全局性能提升的重要问题之一.

针对以上问题,本文综合考虑数据传输时间和负载均衡,对 GPU 上的大规模并行负载调度问题展开研究,将总负载 makespan 最小化作为 GPU 全局性能的优化目标,给出一系列能够同时提升负载执行并行度和缩短数据传输时延的调度策略.迄今为止,GPU 负载调度的研究工作主要集中在处理器如何提升负载并行度上,其中代表性的工作有:Gregg 等人[4]提出了一种 GPU 上细粒度的动态控制策略,实现对大规模负载的并行调度;Li 等人[5]提出了一类虚拟化方法,使多处理器上的不同负载共享 GPU 资源,提升负载执行的并行度;Kato 等人[6−8]针对优先级、抢占和隔离等实时需求,提出了一系列实时计算系统上的 GPU 资源管理策略.以上研究尚未考虑 GPU 独享内存与主存之间的数据传输延时问题.对于 GPU 系统全局性能而言,尤其是在实时系统中,当大规模并行负载同时加载到存储器上时,数据传输和负载均衡同样重要.下面通过 Nvidia Geforce GT 630M 平台中的例子,说明数据传输在调度中的重要性.

图 1 给出了由 Nvidia SDK 开发的两段程序在 GPU 上调度的情况[9].其中,τ_1 为异步 API 程序关联的负载,大小为 32 768 区块(每个区块的规模为 1024KB×1KB);τ_2 为向量加载程序关联的负载,大小为 35 157 区块(每个区块的规模为 256KB×1KB);负载 τ_1 和 τ_2 均在 GPU 上由两个内核并行执行.通过 API 函数 *cudaEventRecord* 获取 GPU 流处理器时钟周期的方法对负载时间信息进行估算得到:τ_1(以及 τ_2)的数据传输时间和负载执行时间分别为 23.392ms 和 29.215ms(以及 28.626ms 和 6.398ms).两负载数据传输顺序改变,将导致不同的调度结果:

- 首先,假设两并行负载任务 τ_1 和 τ_2 同时到达,若不考虑负载的传输时间,随机对负载进行调度,不妨设 τ_2 先于 τ_1 进行传输;然后,负载的第 2 阶段按照先来先服务(first-come-first-serve,简称 FCFS)策略进行调度执行,如图 1(a)所示.这种调度方案对应的 makespan 值为 81.233.
- 若综合考虑负载的传输时间和执行时间,先传输负载 τ_1,如图 1(b)所示.在这种方案下,调度系统的 makespan 值降低至 58.416.

由此对比可知,τ_1 在 τ_2 之前传输能够有效提升 GPU 利用率.一种优良的调度算法应该能够很好地统筹负载的数据传输时间和执行时间,综合考虑 GPU 中的通信和计算资源,使二者相互协调,以达到更好的优化效果.

本文的主要贡献如下:针对 GPU 上负载“传输-执行”联合调度问题,从理论上给出一系列多项式时间的近似结果.首先,将负载的时间信息和并行任务数与矩形域的二维空间联系起来,建立负载的 2D 双层矩形域模型;然后,将 GPU 上负载调度问题归结为一类新的 Strip-Packing 问题;最后,基于贪婪策略给出近似度为 3 的多项式时间近似算法,算法复杂度为 $O(n\log n)$.从文中近似算法能够很容易地得出 GPU 最优或近似最优的调度策略:在数据传输阶段采取负载排序调度,在负载执行阶段采取 FCFS 调度.这也从理论层面上证明了 GPU 系统采取“传输-执行”两阶段调度的正确性和有效性.

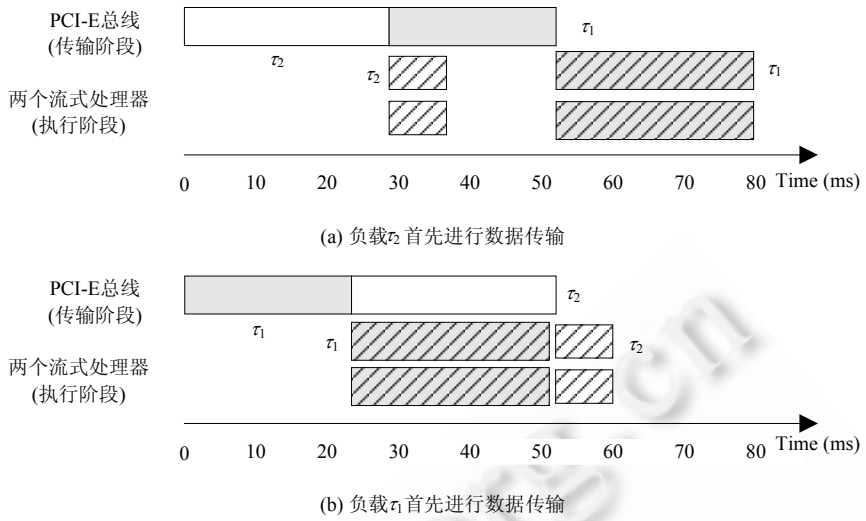


Fig.1 Different results for scheduling two workloads τ_1 and τ_2 by considering the transfer times or not[9]

图 1 考虑传输时间与否对负载τ_1和τ_2调度结果的影响[9]

本文第 1 节介绍 GPU 计算系统模型和问题定义.第 2 节提出调度问题的 2D 双层矩形 Strip-Packing 模型.第 3 节给出问题的近似算法以及相关的理论分析和证明.第 4 节是实验结果.第 5 节介绍多处理器调度领域在近似算法方面的相关研究进展.最后是结束语.

1 系统模型和问题定义

1.1 GPU计算系统模型

本文研究具有 m 个流式多处理器的单 GPU 计算系统,m 为常数($m\geq 2$).在 GPU 系统中,负载调度架构基于 Linux 3.0.0.48 内核设计,并遵循 CUDA 5.0 规范,如图 2 所示.GPU 上的负载调度分为两个阶段:(1) 负载关联的数据传输称为调度的第 1 阶段;(2) 负载在 GPU 上并行执行称为调度的第 2 阶段.

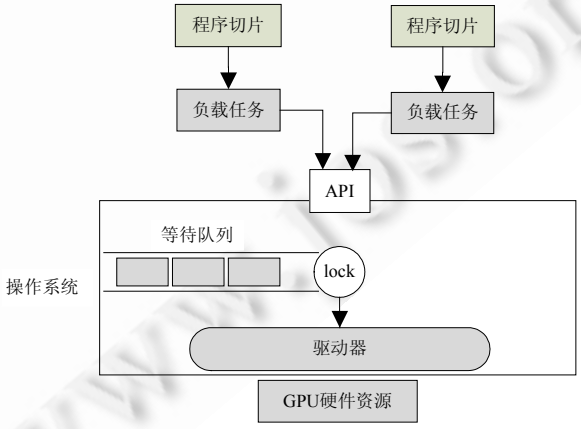


Fig.2 Architecture for scheduling workload on GPU system[9]

图 2 GPU 系统中的负载调度架构[9]

1.1.1 数据传输阶段

在本文的计算系统中,GPU 独享内存和主存之间的数据传输采取异步模式,这具体由 Linux 内核中的 wrapper 模块应用二元信号量实现.也就是说,任何负载若要占用 GPU 的硬件资源(PCI-E 总线),在主存和独享内存之间进行数据传输,必须先获取二元信号量,并实现对信号量的加锁操作.相应地,一旦某负载完成了数据传输操作,它必须释放二元信号量,实现对信号量的解锁操作.本文将数据传输占用 PCI-E 总线看作一种独立的计算资源,比如,在第 1.3 节的问题定义中,PCI-E 总线被抽象为调度问题中第 1 阶段的单机资源.

注意到,在大规模并行负载同时进行数据传输时,如何控制二元信号量的锁操作、确定数据传输顺序,对计算系统全局性能有重要影响(详见图 1 中实例).数据传输阶段的负载排序是本文关注的主要问题.

1.1.2 负载执行阶段

在负载执行阶段,Linux 内核采用默认的 FCFS 等待队列实现并行负载的分配操作.即,先完成数据传输的负载先被加载到 GPU 上的多处理器上并行执行.多处理器对应第 1.3 节调度问题定义中的 m 台并行机器.

FCFS 策略实现简单,不会占用额外的计算资源,但是调度效果会受到阶段 1 中数据传输顺序的影响.数据传输阶段不同的负载排序将导致不同的调度结果.FCFS 调度策略是否能保证 GPU 系统的全局最优性,还尚未可知.本文研究 GPU 上“传输-执行”联合调度问题,第 3 节给出的近似算法结果从理论上证明:FCFS 策略与数据传输阶段负载排序策略相结合的方法能够保证 GPU 系统全局最优或近似最优.

1.2 并行负载模型

本文考虑 GPU 计算系统中的一组并行负载(又称任务)集合 $\Gamma=\{\tau_1,\ldots,\tau_n\}$,负载 τ_i 用三元组 $(tran_i,size_i,exec_i)$ 表示,其中,$tran_i$ 是负载 τ_i 关联的数据从主存传输到 GPU 独享内存上所耗费的时间;$size_i$ 是负载 τ_i 运行所需的处理器个数,一般情况下,$size_i$ 需少于 GPU 中的处理器总数 m;$exec_i$ 是负载 τ_i 在 GPU 上的执行时间.本文假设负载符合伙伴调度(gang scheduling)方式,即负载 τ_i 的所有并行程序块同时分配到一组处理器上执行,并且每个程序块的执行时间相同,且同时开始执行.

当系统加载 τ_i 时,以上三元组变量将被存储在 Linux 内核的 task_struct 结构中.下面介绍负载特征量在 GPU 计算系统中的抽象过程.

1.2.1 数据传输时间

为了获取负载 τ_i 的数据传输时间 $tran_i$,首先需要估算出由主存传输到 GPU 内存的数据量 α_i,以及 PCI-E 总线的有效带宽 β;然后,数据传输时间 $tran_i$ 即可由公式 $\lceil \alpha_i/\beta \rceil$ 计算得出.τ_i 在经过数据传输后,即可加载到 GPU 上开始执行.

1.2.2 并行程序块

对于大多数程序而言,负载通常由大量的程序块构成,每个程序块又被划分为若干线程.GPU 将每个程序块分配到一个流式处理器中执行,每个程序块中并行线程的数量由程序员根据处理器参数决定.

本文模型不具体考虑 GPU 上处理器内核运行时的动态参数,而是采取一种粗粒度的静态方法[8]估算出每个程序块中的并行线程数量.该方法基于如下考虑:GPU 中一个处理器的计算资源是有限的,因此,一个负载加载到该处理器上的最大并行线程数量也是有限的.不妨设负载 τ_i 中的线程总数为 μ_i,τ_i 在一个处理器上能够并行运行的最大线程数为 γ_i.在满足处理器资源约束的条件下,负载 τ_i 并行执行所需的机器数 $size_i$ 等于 $\lceil \mu_i/\tau_i \rceil$.在第 1.3 节中,负载的并行程序块对应于任务在第 2 阶段的并行操作.

1.2.3 负载执行时间

同一负载的所有并行程序块在 GPU 上的执行时间均相等,这是本文的重要假设之一.该假设在基于 Nvidia SDK 开发的绝大多数程序中普遍成立.

由上一节中负载并行程序块模型易知,在负载 τ_i 的 $size_i$ 个并行程序块中,至少有 $size_i-1$ 个程序块包含相同的线程数 γ_i(剩余的一个程序块所包含的线程数一定小于 γ_i,其线程数计算公式为 $\mu_i \bmod \gamma_i$).对于具有相同线程数的程序块,假设所有线程的执行时间均相同,则可应用文献[10]中的方法估算负载执行时间 $exec_i$.另外,对于线程数小于 γ_i 的程序块,令其执行时间等于 $exec_i$.这种近似使得调度算法的实际效果一定不会低于理论值.

1.3 问题定义

GPU 上“传输-执行”联合调度问题可归结为二阶段 flow-shop 调度问题,其形式化定义如下:

给定一般的任务集$\Gamma=\{\tau_1,\ldots,\tau_n\}$以及一个二阶段 flow-shop 调度环境,其中,阶段 1 只有 1 台机器,阶段 2 中有 m 台同构并行机.每个任务 τ_i 包含 $size_i$+1 个操作,其中,第 1 个操作 o_i 只能在阶段 1 的单机上执行,其执行时长为 $tran_i$.在 o_i 完成之后,另外 $size_i$ 个操作方可开工,且必须由第 2 阶段的 $size_i$ 台机器同时并行执行($size_i\leqslant m$),这 $size_i$ 个操作的执行时长均为 $exec_i$.在每个时刻,1 台机器只能执行 1 个操作,且 1 个操作只能由 1 台机器执行.另外,在操作执行过程中不允许抢占,即操作一旦开始执行,在它完成之前不能有中断发生.

问题的优化目标是:所有任务完成加工的时间 makespan 最短,其中,最优的 makespan 定义为 $C_{\max}$.根据调度问题的三参数表示法[11],本文问题可记为 $F2(1,P_m)|size_i|C_{\max}$.$F2(1,P_m)$表示二阶段 flow-shop 调度环境,其中,阶段 1 和阶段 2 的机器数分别为 1 和 m.$size_i$ 代表并行任务集合,即任意任务 τ_i 包含 $size_i$ 个操作,其必须在第 2 阶段的机器上同时并行执行.

问题 $F2(1,P_m)|size_i|C_{\max}$ 属于强 NP 困难问题,甚至在最简单的 $F2(1,2)$情况下该结论也成立,见推论 1.为了解决该强 NP 困难问题,本文首先给出原问题的二维双层矩形 Strip-Packing 模型(见第 2 节),然后基于 Strip-Packing 模型给出 $F2(1,P_m)|size_i|C_{\max}$ 的近似算法和理论结果.

推论 1. 问题 $F2(1,P_m)|size_i|C_{\max}$ 是强 NP 困难问题,甚至当 $size_i$=1 且 m=2 时,该结论也成立.

证明:令每个任务在第 2 阶段仅包含 1 个子操作,$F2(1,P_m)|size_i|C_{\max}$ 即等价为文献[12]中的 $F2(P)|\cdot|C_{\max}$ 问题,由文献[13]即可证明阶段 1 的机器数为 1、阶段 2 的机器数为 2 的 $F2(P)|\cdot|C_{\max}$ 为强 NP 困难问题. □

2 二维双层矩形 Strip-Packing 模型

由第 1.3 节的定义可知,本文的问题属于并行任务调度(multiprocessor task secheduling,简称 MTS)问题在两阶段流水调度环境中的扩展.传统的单阶段 MTS 问题已有大量的算法理论研究[14−23],其中大多数算法结果主要基于一种二维 Strip-Packing 模型[23].Strip-Packing 模型应用矩形域的二维空间刻画任务的执行时间和并行子操作数这两类特征参数,是 MTS 问题天然的理论模型之一.然而,现有的 Strip-Packing 模型尚未扩展到二阶段调度领域,不能用于两阶段 MST 问题的近似算法研究.为此,本节首先在第 2.1 节建立两阶段并行任务的双层矩形域模型,其中,矩形的两层子域具有不同的属性,并且满足不同的约束.这些属性和约束用来刻画并行任务两阶段子操作的时序关系和执行时需要满足的计算资源约束.第 2.2 节给出了双层矩形域的属性和约束的形式化定义,并最终将 $F2(1,P_m)|size_i|C_{\max}$ 归结为 2D 双层矩形 Strip-Packing 问题求解.

2.1 任务 τ_i 与2D双层矩形 (w_i,h_i^1,h_i^2) 的对应关系

任务 τ_i 对应的矩形域 r_i 如图 3 所示.

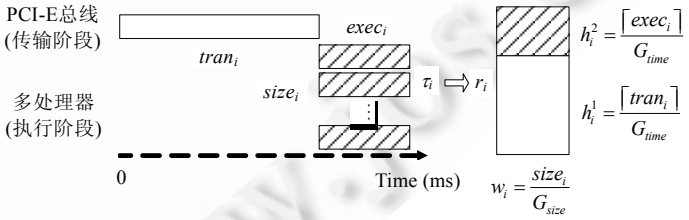


Fig.3 Rectangle model for workload τ_i

图 3 任务 τ_i 对应的矩形模型

τ_i 两阶段的子操作分别对应矩形 r_i 的双层区域,其中,第 1 阶段的子操作 o_i 对应底层的空白矩形域,执行时间 $tran_i$ 对应空白域的高度 h_i^1;第 2 阶段的 $size_i$ 个子操作对应上层的条纹矩形域,执行时间 $exec_i$ 对应条纹域的高度 h_i^2.另外,任务在第 2 阶段执行所需的并行机器数 $size_i$ 对应矩形 r_i 的宽度 w_i.由第 1.3 节的定义可知,$size_i$

小于等于第 2 阶段总机器数 m,因此,并行机数 m 也对应一个宽度值 W,使得$\forall \tau_i \in \Gamma, \tau_i$ 对应矩形 r_i 的宽度 $w_i \leqslant W$.为了方便问题求解,令 W=1,对 $size_i$ 进行归一化,得到落在[0,1]区间的矩形宽度值 w_i.相应地,通过为负载任务的执行时间选择合适的阈值 G_{time},对矩形域的高度 h_i^1 和 h_i^2 做与矩形宽度类似的规范化处理.具体的规范化过程见算法 1.

算法 1. 负载任务特征参数的规范化算法.

输入:任务集合Γ及并行机器数 m.

输出:规范化的双层矩形域集合.

BEGIN

1. SET $G_{size}:=m; G_{time}:=\max_{1\leqslant i\leqslant n}\lceil exec_i \rceil$;

2. REPEAT $1\leqslant i\leqslant n$:

3. SET $w_i:=size_i/G_{\text{size}}$; $h_i^1 := \lceil tran_i \rceil / G_{time}$; $h_i^2 := \lceil exec_i \rceil / G_{time}$.

END.

由算法 1 易知:经过规范化后,任务负载对应的矩形上层条纹域落在[0,1]×[0,1]区域内,底层空白域落在[0,1]×[0,h_g]区域内,h_g 为 $\max_{1\leqslant i\leqslant n}\{tran_i\}/G_{time}$,可能大于 1.这种规范化矩形域更有利于构造近似算法.而且,算法 1 中第 1 步和第 3 步的取整操作能够保证第 3 节中算法的实际效果优于理论值.

2.2 $F2(1,P_m)|size_i|C_{\max}$归结为2D双层矩形Strip-Packing问题

在经过负载任务参数规范化后,问题 $F2(1,P_m)|size_i|C_{\max}$ 可转化为 2D 双层矩形 Strip-Packing 问题,其形式化的定义为:给定 2D 双层矩形域列表 $L=(r_1,\ldots,r_n)$,其中,每个矩形 r_i 均关联三元组 (w_i,h_i^1,h_i^2), 且满足 $0\leqslant w_i, h_i^2 \leqslant 1 \wedge 0\leqslant h_i^1 \leqslant h_g$,如图 3 所示.定义列表 L 的面积(高度)为 L 中所有矩形的面积(高度)之和.列表 L 上 Strip-Packing 问题的目标是:在一个有限二维区域[0,1]×[0,+∞]中,为 L 中每个矩形域 r_i 的空白域和条纹域分别指定一个位置坐标$\langle x_i,y_i\rangle$和 $\langle x_i,y_i'\rangle$ (也可简记为三元组 $\langle x_i,(y_i,y_i')\rangle$), 使得:

(1) 每个矩形 r_i 的条纹域均在其空白域之上,且彼此不相交,即,$\forall r_i \in L$,有 $y_i' \geqslant y_i + h_i^1$;

(2) 所有矩形的上层条纹域彼此不相交,即,$\forall r_i, r_j \in L$,有:

$$[x_i, x_i + w_i] \times [y_i', y_i' + h_i^2] \cap [x_j, x_j + w_j] \times [y_j', y_j' + h_j^2] = \varnothing;$$

(3) 区域[0,1]×[0,+∞]中任一水平线最多与 1 个矩形空白域相交,即,$\forall r_i, r_j \in L$,有:

$$[y_i, y_i + h_i^1] \cap [y_j, y_j + h_j^1] = \varnothing.$$

在区域[0,1]×[0,+∞]中,若矩形 r_i 的空白域和条纹域分别定位在$[x_i,x_i+w_i]\times[y_i,y_i+h_i^1]$和$[x_i,x_i+w_i]\times[y_i',y_i'+h_i^2]$两区域,则 y_i 为空白域下确界,$y_i+h_i^1$ 为空白域上确界(同时也是条纹域的下界), y_i' 是条纹域的下确界, $y_i'+h_i^2$ 是条纹域的上确界,同时也是整个矩形域 r_i 的上确界.列表 L 的 Strip-Packing 高度即为 L 中所有矩形上确界的最大值.问题最优解(L 的 Strip-Packing 高度最小值)定义为 $Opt(L)=\inf\{f$ 的高度$|f$ 是 L 的任意 Strip-Packing 解$\}$.

例 1:图 1 中两负载任务 τ_1 和 τ_2 对应的矩形域分别为 $r_1=(w_1,h_1^1,h_1^2)$ 和 $r_2=(w_2,h_2^1,h_2^2)$,如图 4 所示.

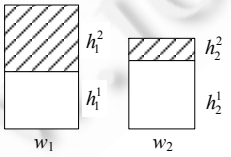
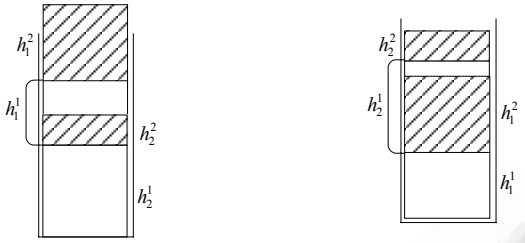


Fig.4 Rectangle models for τ_1 and τ_2

图 4 τ_1 和 τ_2 对应的矩形模型

由算法 1 可得,$w_1=w_2=1$, $h_1^1=0.8, h_1^2=1, h_2^1=0.97, h_2^2=0.23$.矩形列表 $L=(r_1,r_2)$在二维区域[0,1]×[0,+∞]中两组可行的 Strip-Packing 解如图 5(a)和图 5(b)所示,分别对应图 1(a)和图 1(b)中的两组负载分配方案.其中,第 1 组解

的高度为 2.77,第 2 组解的高度为 2.03.显然,第 2 组解更优,对应在 $F2(1,P_m)|size_i|C_{\max}$ 问题中的解为:在第 1 阶段,子操作 o_1 在 o_2 之后执行;在第 2 阶段,τ_1 和 τ_2 按照 FCFS 策略进行调度即可.



(a) 矩形 r_2 先于 r_1 放入二维区域中　　(b) 矩形 r_1 先于 r_2 放入二维区域中

Fig.5 Two strip-packing solutions with different heights for Instace 1

图 5 例 1 中两组不同高度的 Strip-Packing 解

例 1 直观地给出了 Strip-Packing 问题和调度问题之间的对应关系.在 2D 双层矩形 Strip-Packing 问题中,条件(1)利用同一矩形双层域之间的互斥性,刻画任务两阶段子操作顺序执行的特性;条件(2)利用不同矩形条纹域之间的互斥性,刻画在每一时刻,1 台机器只能执行 1 个操作,以及 1 个操作只能由 1 台机器执行的约束条件;条件(3)利用不同矩形空白域之间的互斥性,刻画调度问题第 1 阶段仅有 1 台机器的约束条件.另外,调度问题中的不可抢占条件由矩形空白域和条纹域的完整性刻画.Strip-Packing 问题的最优解 $Opt(L)$对应于 $F2(1,P_m)|size_i|C_{\max}$ 问题的最早完工时间 $C_{\max}$.综合本节和第 1.3 节的问题定义易知,Strip-Packing 问题中的最优解即为调度问题参数规范化后的最优解.本文将在第 3 节给出 Strip-Packing 解转化为 $F2(1,P_m)|size_i|C_{\max}$ 解的算法.

3 多项式时间近似算法

3.1 矩形条纹域可塑性假设

本节讨论一种满足条纹域可塑假设的 Strip-Packing 解,又称为可塑解.相应地,本节的假设条件又称为可塑条件,具体指以下 3 个方面:

(1) 每个矩形 r_i 的条纹域可在总面积 $w_i\times h_i^2$ 保持不变的前提下,改变宽度和高度.设 r_i 条纹域的宽度和高度增量分别为Δw 和Δh,则有 $0<w_i+\Delta w\leqslant 1,0< h_i^2+\Delta h,\Delta w\times\Delta h<0$ 成立,并且$(w_i+\Delta w)\times(h_i^2+\Delta h)\equiv w_i\times h_i^2$,如图 6(a)所示.

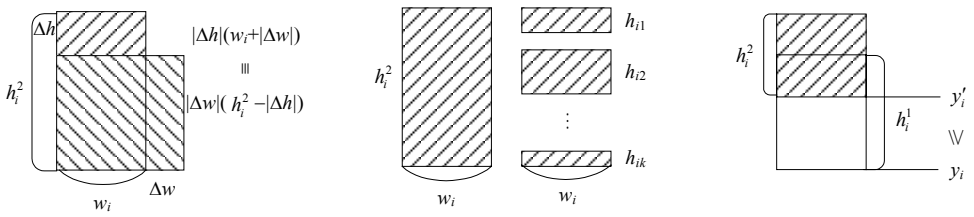
(2) 每个矩形 r_i 的条纹域$[0,w_i]\times[0,h_i^2]$可被水平线切割分为多个子条纹域$([0,w_i]\times[0,h_{i1}],[0,w_i]\times[0,h_{i2}],\ldots,[0,w_i]\times[0,h_{ik}])$,且 $\sum_{l=1}^{k}h_{il}=h_i^2$,如图 6(b)所示.另外,这 k 个子域也可在不改变面积的前提下改变宽度和高度.即,矩形域对条件(1)和条件(2)可以递归性地相互嵌套.

(3) 每个矩形 r_i 的条纹域与空白域允许相交,但条纹域下界 y_i' 不能低于相应空白域的下界 y_i,如图 6(c)所示.该条件是第 2.2 节 Strip-Packing 问题中约束条件(1)的松弛.

前两个假设条件分别对应 flow-shop 调度领域中的两个经典假设,其中,条件(1)对应任务可变性(malleable),即,任务可以在执行过程中任意改变并行度[2];条件(2)对应可抢占性[24],即,任务在执行过程中允许中断.易知,满足任务可变性和可抢占条件的调度解通常优于任务不可变且不可抢占的解.

另外,假设条件(3)在 $F2(1,P_m)|size_i|C_{\max}$ 中即对应允许负载执行操作和数据传输操作同时并行执行.提出该假设是为了方便计算问题解的下界.

综上所述,满足以上 3 个假设条件的可塑解是 Strip-Packing 问题的下界.接下来,第 3.2 节将 Strip-Packing 问题的任一可行解转化为可塑解,同时获得问题的下界.



(a) 假设条件(1)的示意图 (b) 假设条件(2)的示意图 (c) 假设条件(3)的示意图

Fig.6 Moldable assumptions

图 6 可塑性假设

3.2 Strip-Packing可行解到可塑解的缩界过程

给定 Strip-Packing 问题任一可行解 $f=(\langle x_{[1]},(y_{[1]},y'_{[1]})\rangle,...,\langle x_{[n]},(y_{[n]},y'_{[n]})\rangle)$,其中,$\langle x_{[i]},(y_{[i]},y'_{[i]})\rangle$ 为第[i]个矩形在二维区域[0,1]×[0,+∞]中的坐标,不妨设f中矩形按空白域下界递增序排列,即 $y_{[1]}<y_{[2]}<\ldots<y_{[n]}$.算法 2 给出了由可行解$f$转化为 Strip-Packing 可塑解 f_{m}的步骤.

算法 2. Strip-Packing 可行解到可塑解的转化算法.

输入:Strip-Packing 可行解f.

输出:Strip-Packing 可塑解f_m.

BEGIN

1. REPEAT $1\leqslant i\leqslant n$:
2. 令 $S_{[i]}$为区域$[0,1]\times[y_{[i]},y_{[i]}+h^1_{[i]}]$中刨除条纹域后的剩余面积;
3. IF $S_{[i]}\geqslant w_{[i]}\times h^2_{[i]}$ THEN
4. 等面积调整矩形 $r_{[i]}$条纹域的形状,得到新条纹域 $B_{[i]}$;
5. 将 $B_{[i]}$置入$[0,1]\times[y_{[i]},y_{[i]}+h^1_{[i]}]$中;
6. ELSE
7. 截取矩形 $r_{[i]}$条纹域上半段区域$[x_{[i]},x_{[i]}+w_{[i]}]\times\left[y'_{[i]}+S_{[i]}/w_{[i]},y'_{[i]}+h^2_{[i]}\right]$;
8. 调整截取区域的形状,得到新条纹域 $B_{[i]}$,跳转到步骤 5;

END

注意到,第 3.1 节的假设条件(3)允许矩形条纹域与空白域相互重合,因此,算法 2 的目标是尽量让每个矩形的条纹域回填到其相应的空白域中,以提高二维区域中条纹域的密度.算法从水平位置最低的矩形空白域开始扫描,首先获取空白域所在的二维域中不含条纹域的空白区(算法第 2 行),然后将相应条纹域(或部分)填入到该空白区内(算法第 3 行~第 8 行).根据第 3.1 节的假设条件(1)和条件(2),条纹域可任意调整宽度和高度,并可以任意横向切割,因此在理论上,条纹块回填到空白区的面积为 $\min\{S_{[i]},w_{[i]}\times h^2_{[i]}\}$.由算法 2 不难得出定理 1.

定理 1. 满足可塑假设(1)~(3)的最优解值是 Strip-Packing 问题下界.

证明:对于 Strip-Packing 问题任意的可行解 f,都可应用算法 2 将其转化为满足假设条件(1)~(3)的可塑解f_m.可行解f中任一矩形 $r_{[i]}$的条纹域$[x_{[i]},x_{[i]}+w_{[i]}]\times[y'_{[i]},y'_{[i]}+h^2_{[i]}]$均可全部或部分回填到其关联的空白区 $S_{[i]}$中,如图 7 所示.如果存在$1\leqslant i\leqslant n$,使得 $S_{[i]}>0$,则回填过程分为两种情况:(1) 若条纹域面积小于等于 $S_{[i]}$,则整个条纹域可回填到 $S_{[i]}$中(如算法 2 第 4 行、第 5 行);(2) 否则,条纹域的上半部分回填到 $S_{[i]}$中(如算法 2 第 7 行、第 8 行).以上两种情况均降低了矩形 $r_{[i]}$条纹域的上确界f 中的其他矩形 $r_{[j]}(i<j\leqslant n)$即可进行相应的位置调整,填补 $r_{[i]}$条纹域减少的部分.最终,算法 2 求得的 f_m,其高度必定小于 f 的高度.另外,若所有 $S_{[i]}$均等于 0,则经过算法 2 求得的f_m与原f相同,f_m的高度亦与f相同.综上所述,任意 Strip-Packing 可行解f的高度均大于等于其相应可塑

解 f_m 的高度.因此,最优可塑解值是 Strip-Packing 问题的下界. □

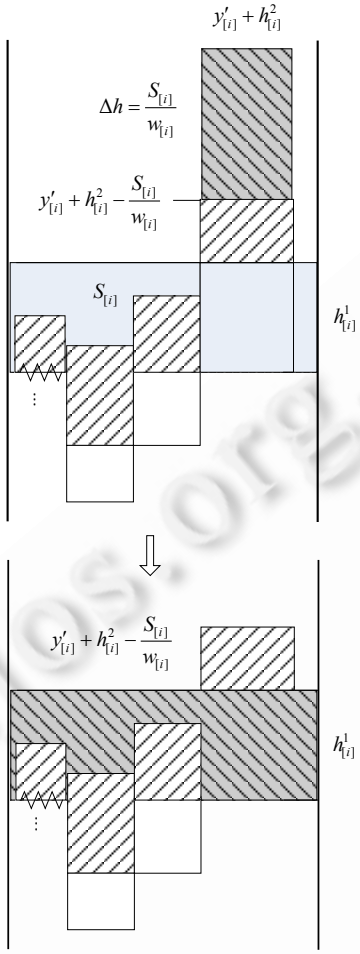


Fig.7 Heights of the striped regions decrease after backfilling

图 7 条纹域回填后高度变小

在矩形列表 L 的 Strip-Packing 问题中,设满足可塑假设的最优解为 f_m^*, 其高度定义为 $Opt(L_m)$.由定理 1 易知,$Opt(L_m)\leqslant Opt(L)$.接下来,在第 3.3 节将 f_m^* 扩张为原 Strip-Packing 问题的可行解,并给出 Strip-Packing 问题的上界.

3.3 Strip-Packing可塑解到可行解的扩界过程

本节的目的是获得 Strip-packing 问题最优解的上界 $Opt_u(L)$,如定理 2 所述.

定理 2. 矩形列表 L 上 Strip-packing 问题的上界 $Opt_u(L)=Opt(L_m)+2\sum_{i=1}^{n}w_i$.

证明:为了证明定理 2,Strip-Packing 最优可塑解 f_m^* 按算法 3 给出的步骤扩张为可行解 f_u.

算法 3. Strip-Packing 可塑解到可行解的转化算法.

输入:Strip-Packing 可塑解 f_m^*.

输出:Strip-Packing 可行解 f_u.

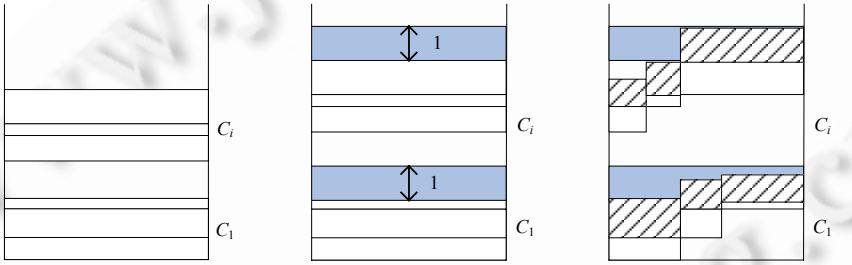
BEGIN

1. SET j:=1;C_j:=∅;
2. REPEAT $1\leqslant i\leqslant n$:
3. IF $\sum_{r_l\in C_j} w_l + w_{[i]} \leqslant 1$ THEN
4. SET $r_{[i]}$:→C_j;
5. ELSE
6. SET h_{low}:=$y_{[i-1]}$+ $h^1_{[i-1]}$;h_{up}:=$y_{[i]}$;
7. 在高度 h_{low} 和 h_{up} 间向二维域插入空白[0,1]×[h_{low},1];
8. 令第 1 个加入 C_j 矩形的空白域下界为 $y(j)$;
9. 在空白区[0,1]×[$y(j)$,h_{low}+1]内放置 C_j 中的矩形(如图 8 所示);
10. SET j:=j+1;C_j:=∅;

END

该算法分为 3 个阶段:(1) 按照可塑解 f_m^* 中矩形空白域下界的递增序,将矩形集合 L 划分为若干互不相交的子集{C_1,…,C_k},如图 8(a)所示(算法的第 3 行、第 4 行);(2) 将每个子集 C_j 对应的空白域扩界,即,在 C_j 最顶端空白域的上界基础上再增加 1 个单位空白区[0,1]×[0,1],如图 8(b)所示(算法的第 6 行、第 7 行);(3) 将 C_j 安置到扩张后的空白域内,如图 8(c)所示(算法的第 8 行、第 9 行).

在阶段(1)中,算法的第 3 行保证了每个子集 C_j 中矩形的宽度之和小于等于 1.这意味着 C_j 中的矩形一定能顺次排布在[0,1]区间内,不会超过有限二维区域的横向边界.由于 L 中每个矩形 r_i 条纹域的高度 $h_i^2\leqslant 1$,阶段(2)恰好为每个 C_j 扩界 1 个单位,所以 C_j 中每个矩形的条纹域均可安置在其空白域之上,且高度不超过 h_{low}+1,如图 8 所示.



(a) L 中矩形按宽度划分子集 (b) 对每个子集进行扩界 (c) 安置每个子集的矩形域

Fig.8 Transform from strip-packing moldable solution to the feasible one

图 8 Strip-Packing 可塑解转换为可行解

注意到,相邻的两个子集 C_j 和 C_{j+1} 的宽度之和一定大于 1,否则,依据算法 3 的第 3 行、第 4 行,C_j 和 C_{j+1} 应合并为一个矩形子集.因此,算法 3 中的扩界次数不超过 $2\sum_{i=1}^{n} w_i$,即算法求得的 Strip-Packing 问题可行解的上界 $Opt_u(L)$为 $Opt(L_m)$+$2\sum_{i=1}^{n} w_i$. 从而定理 2 得证. □

3.4 近似算法及理论分析

近似算法分为两个阶段:首先,基于贪婪策略求出 Strip-Packing 问题最优的可塑解 f_m^*;然后,再应用算法 3 对 f_m^* 进行扩界,得到可行解 f_u.两个阶段的全局算法描述如下:

算法 4. Strip-Packing 问题的近似算法.

输入:矩形列表 L.

输出:Strip-Packing 可行解 f_u.

BEGIN

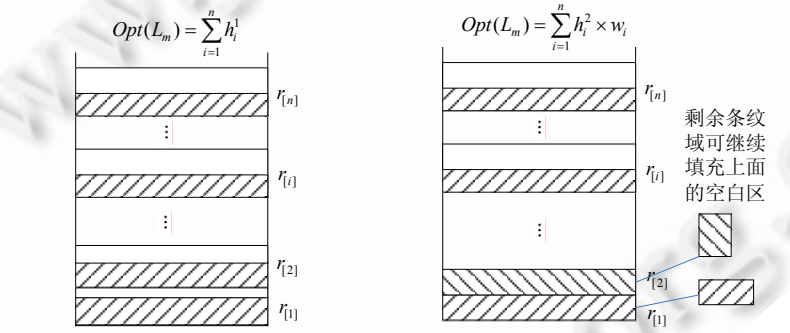
1. 将 L 中的矩形按 $w_i\times h_i^2/h_i^1$ 非递增序排列:$(r_{[1]},\ldots,r_{[n]})$;
2. REPEAT $1\leqslant i\leqslant n$:
3. 将空白区 $S_i=[0,1]\times[0,h_{[i]}^1]$放置在 S_{i-1} 之上(令 $S_0=[0,1]\times[0,0]$);
4. IF $w_{[i]}\times h_{[i]}^2\leqslant S_i$ THEN
5. 调整矩形 $r_{[i]}$条纹域,得到等面积条纹域 $B_{[i]}=[0,1]\times[w_i\times h_i^2]$;
6. 将 $B_{[i]}$放置在 S_i 的底部;
7. ELSE
8. 截取矩形 $r_{[i]}$条纹域上半段区域$[0,w_{[i]}]\times\left[h_{[i]}^2-S_{[i]}/w_{[i]},h_{[i]}^2\right]$;
9. 调整截取区域的形状,得到新条纹域 $B_{[i]}$,跳转到步骤 6;
10. 令 $r_{[i]}$条纹域剩余部分为 $R_{[i]}$;
11. 将所有 $R_{[i]}$自底向上添加到空白区剩余的面积中,得到可塑解 f_m;
12. 对 f_m 应用算法 3,将其调整为 Strip-Packing 可行解 f_u;

END

定理 3. 算法 4 能够求得 Strip-Packing 问题的最优可塑解.

证明:首先给出算法 4 求得可塑解的取值.根据算法 4 的第 2 行、第 3 行,将 L 中所有矩形对应的空白区 $S_1,\ldots,S_n$ 放置在有限域$[0,1]\times[0,+\infty]$中.所有空白区的高度之和为 $\sum_{i=1}^n h_i^1$.算法 4 的第 4 行~第 11 行是将矩形条纹域回填的过程,面临两种情况:

(1) 每个矩形 $r_{[i]}$的条形域面积均小于等于其对应的空白区面积 S_i(算法的第 4 行),在这种情况下,所有条纹域均回填到各自的空白区内,回填操作不影响可塑解的高度,如图 9(a)所示.



(a) 矩形条纹域面积小于等于空白区面积 (b) 某些矩形条纹域面积大于空白区面积

Fig.9 Backfilling with striped blocks

图 9 条纹域的回填

(2) 存在部分矩形的条纹域面积大于其对应的空白区面积(算法的第 7 行).根据算法的第 1 行易知,这类矩形的空白域均安置在有限域$[0,1]\times[0,+\infty]$的底层部分,如图 9(a)所示.不妨设这部分矩形为$(r_{[1]},\ldots,r_{[k]})$,其关联的剩余条纹域(见算法的第 10 行)为$(R_{[1]},\ldots,R_{[k]})$.根据算法的第 11 行以及第 3.1 节的假设条件(1)和条件(2),可将每个 $R_{[i]}$任意调整形状和水平切割,将其填入空白区 $S_{k+1},\ldots,S_n$ 的剩余面积中.若剩余条纹域未填满所有空白区,则此二次回填操作并未改变可塑解高度,其值依然为 $\sum_{i=1}^n h_i^1$;若剩余条纹域填满所有空白区,则继续填入$[0,1]\times[\sum_{i=1}^n h_i^1,+\infty]$区域,则该操作改变可塑解的高度,其值为 $\sum_{i=1}^n w_{[i]}\times h_{[i]}^2$.

易知:针对情况(1),可塑解的最低高度即为 $\sum_{i=1}^{n} h_i^1$; 对于情况(2),可塑解的最低高度为 $\sum_{i=1}^{n} w_{[i]} \times h_{[i]}^2$. 因此,算法 4 求得的可塑解值为 $\max\left\{\sum_{i=1}^{n} h_i^1, \sum_{i=1}^{n} w_i \times h_i^2\right\}$.

接下来给出问题最优可塑解的下确界.

注意到,在有限域[0,1]×[0,+∞]中,一条水平线不能与两个矩形的空白域同时相交,因此,所有空白区的高度之和 $\sum_{i=1}^{n} h_i^1$ 为问题可塑解的下界.又因为在理想情况下,所有矩形条纹块经过可塑性条件(1)和条件(2)递归式调整,无空隙地放置在有限域[0,1]×[0,+∞]内.此时,可塑解的另一下界为 $\sum_{i=1}^{n} w_i \times h_i^2$. 故可塑解的下界解大于等于 $\max\left\{\sum_{i=1}^{n} h_i^1, \sum_{i=1}^{n} w_i \times h_i^2\right\}$. 已证明算法 4 求得的可塑解值等于 $\max\left\{\sum_{i=1}^{n} h_i^1, \sum_{i=1}^{n} w_i \times h_i^2\right\}$. 综上所述,算法 4 能够求得 Strip-Packing 问题的最优可塑解. □

根据定理 1 和定理 3,可得如下推论:

推论 2. Strip-Packing 问题的下界 $Opt_{low}(L)=\max\left\{\sum_{i=1}^{n} h_i^1, \sum_{i=1}^{n} w_i \times h_i^2\right\}$.

根据推论 2 以及定理 2,可得 Strip-Packing 问题最优解的上界如下:

定理 4. Strip-Packing 问题的上界为 $3\sum_{i=1}^{n} w_i$, 若 $Opt_{low}(L)=\sum_{i=1}^{n} w_i \times h_i^2$.

证明:由于 $Opt_{low}(L)=\sum_{i=1}^{n} w_i \times h_i^2$, 根据推论 2 可得 $\sum_{i=1}^{n} h_i^1 \leqslant \sum_{i=1}^{n} w_i \times h_i^2$;

又因为 $0 \leqslant h_i^2 \leqslant 1$,易知 $\sum_{i=1}^{n} w_i \geqslant \sum_{i=1}^{n} w_i \times h_i^2$. 因此, $\sum_{i=1}^{n} w_i \geqslant Opt_{low}(L)$.

根据定理 2,问题最优解 $Opt(L)$满足不等式:$Opt(L) \leqslant Opt_{low}(L)+2\sum_{i=1}^{n} w_i$, 即,$Opt(L) \leqslant 3\sum_{i=1}^{n} w_i$. □

由定理 4 可知,当 $Opt_{low}(L)=\sum_{i=1}^{n} w_i \times h_i^2$ 时,Strip-Packing 问题存在伪多项式的上界.对于 $Opt_{low}(L)=\sum_{i=1}^{n} h_i^1$ 的某些情况,定理 5 给出了算法 4 具有常近似度的理论分析.

定理 5. 算法 4 的近似度为 3,若 $\min_{1 \leqslant i \leqslant n}\lceil tran_i \rceil \geqslant \max_{1 \leqslant i \leqslant n}\lceil exec_i \rceil$.

证明:由 $\min_{1 \leqslant i \leqslant n}\lceil tran_i \rceil \geqslant \max_{1 \leqslant i \leqslant n}\lceil exec_i \rceil$可知:$\forall r_i \in L$, $h_i^1 \geqslant 1 \geqslant h_i^2$. 易知, $\sum_{i=1}^{n} h_i^1 \geqslant \sum_{i=1}^{n} w_i \times h_i^1 \geqslant \sum_{i=1}^{n} w_i$, 且 $Opt_{low}(L)=\sum_{i=1}^{n} h_i^1$. 即,$Opt_{low}(L) \geqslant \sum_{i=1}^{n} w_i$.

根据定理 2 可得,$Opt(L) \leqslant Opt_{low}(L)+2\sum_{i=1}^{n} w_i$. 因此,$Opt(L) \leqslant 3Opt_{low}(L)$. □

定理 5 的充分条件是:数据传输时间大于单个处理器上的负载执行时间.本文研究的即是大规模并行负载同时加载到 GPU 上的问题,数据传输时间往往是不可忽视的问题参数.定理 5 刻画了数据传输时间大于负载执行时间的情况,在这种情况下,算法 4 具有常近似度.若存在负载执行时间大于数据传输时间的负载实例,算法 4 求得的解则具有伪多项式上界,如定理 4 所述.

注意到,算法 4 通过调用算法 3 将每个矩形的条纹域直接放置在其空白域上,这对应负载在数据传输完成后立即执行的情况,符合 GPU 内部的 FCFS 调度策略.基于此,可给出在 $F2(1,P_m)|size_i|C_{\max}$ 问题的两阶段调度策略,如算法 5 所示:首先,在第 1 阶段,根据算法 4 对负载排序,按照顺序进行数据传输(算法 5 的第 1 行~第 3 行);然后,在第 2 阶段,应用 FCFS 策略调度负载在 GPU 多处理器上执行(算法 5 的第 4 行、第 5 行).

算法 5. $F2(1,P_m)|size_i|C_{\max}$ 问题的两阶段调度算法.

输入:负载任务集合Γ及并行处理器个数 m.

输出:负载的数据传输及执行次序.

BEGIN

1. 将Γ中负载按$(size_i \times exec_i)/(m \times tran_i)$非递增序排列$(\tau_{[1]},\ldots,\tau_{[n]})$;
2. REPEAT $1 \leqslant i \leqslant n$:
3. 将$\tau_{[i]}$关联的数据在 PCI-E 总线上进行传输;
4. REPEAT $1 \leqslant i \leqslant n$:
5. 按照 FCFS 策略,对完成传输的负载进行调度.

END

4 实验结果

为了验证本文算法的有效性,本节针对由 4 类负载组成的测试集进行实验.这 4 类负载的名称分别为 *vectorAdd*,*incrementKernel*,*BlackScholesGPU* 和 *pad-DataClampToBorder*.对于每类负载,又考虑两组不同的参数,见表 1.因此,共生成了 16 组负载集合用于测试.每组负载集合详细的参数预估算法详见文献[9].

Table 1 Characteristics of workloads[9]

表 1 负载的特征参数值[9]

工作负载	传输数据量(MB)	程序块大小	程序块个数	工作负载	传输数据大小(MB)	程序块大小	程序块个数
vectorAdd[1]	152.588	1024×1	19 542	*vectorAdd*[2]	133.514	896×1	19 542
incrementKernel[1]	91.544	256×1	61 440	*incrementKernel*[2]	91.544	512×1	61 440
BlackScholesGPU[1]	46.73	16×8	100 352	*BlackScholesGPU*[2]	46.73	16×4	200 704
padDataClampToBorder[1]	96	512×1	49 152	*padDataClampToBorder*[2]	16	1024×1	4 096

图 10 给出了本文算法、随机排序算法以及基于穷举解法(最优解)之间的比对结果.

16 组实验数据表明:本文算法明显优于随机排序算法,并十分接近最优解,Gap 值平均为 1.34%.另外,本文算法有 6 组结果等同于最优解.

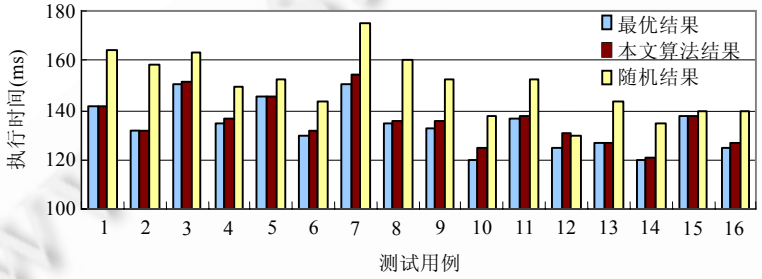


Fig.10 Makespans obtained by our algorithm, ramdom and emuneration strategies

图 10 本文算法、随机算法及枚举策略的求解结果

5 相关工作

本节介绍相关问题的近似算法结果.注意到,本文研究的GPU上“传输-执行”二阶段调度问题在运筹学领域属于多阶段多处理器流水车间调度(国际上通常称为 flexible flow shop 或 hybrid flow shop,简称 HFS)和并行任务调度交叉的范畴.在过去的几十年间,多阶段和并行任务这两类特性很少放在一个问题中研究,HFS 和 MTS 这两个问题分别在各自的研究范畴取得了众多经典的近似算法理论结果.

HFS 问题的近似结果始于 20 世纪 70 年代,Sahni[25]为机器数固定的 HFS 问题给出了具有伪多项式上界的完全多项式近似策略(fully polynomial time approximation strategy,简称 FPTAS).Hochbaum 和 Shmoys[26]研究了机器数作为变量的问题实例,并为该问题提出了目前最好的 PTAS 算法.Hall[27]研究了阶段数和机器数均为常量的问题,提出了一种优美的 PTAS 算法,能够给出问题的多项式上界,改进了 Sahni[25]的结果.Williamson[28]研究了阶段数为变量的问题,证明了该问题不存在近似度小于 5/4 的 PTAS 算法.Jansen 和 Sviridenko[29]研究了机器数为变量的 HFS 问题,应用动态规划技术给出了近似度为 3/2 的 PTAS 算法.对于二阶段的 HFS 问题,众多学者给了很多近似算法[30,31].这些算法的近似度都大于等于 3/2.Schuurman 和 Woeginger[32]基于线性规划技术证明了二阶段 HFS 问题具有近似度为$(1+\varepsilon)$的 PTAS 算法,这是 HFS 近似算法研究中里程碑式的成果.近年来,具有特殊约束条件的 HFS 问题特例得到研究.Xie 等人[24]为了满足可获性(availability)约束的二阶段 HFS,给出了近似度

为 3 的近似算法.Sevastyanov[33]研究了多阶段共用一个机器集合的情况,并进一步改进了 Hall[27]的近似结果.Choi 等人[34]研究了第 1 阶段单机、第 2 阶段有 m 台机器,且每个阶段任务的执行时间均相同的情况,并给出了 PTAS 算法.该算法在 m=2 时,具有常数近似度为 5/4;在 $m\geq 2$ 时,近似度变为$((1+m^2)^{1/2}+m+1)/2m$.HFS 最新的研究结果见文献[35−37].

MTS 问题的近似算法研究,最初受到资源约束调度问题近似结果的启发.1975 年,Garey 等人[14]为资源约束调度问题给出了近似度为 2 的多项式时间算法.Ludwig 等人[15]发现,MTS 中的处理器可看做一种调度资源.于是,借鉴 Garey 和 Granham 的思路,证明了 MTS 问题多项式时间近似算法的存在性.Jansen 等人[16]以及 Amoura 等人[17]分别对机器数固定的问题实例进行了研究,提出了若干 PTAS 算法.对于机器数为变量的问题,Scharbrodt 等人[18]提出了近似度为 3 和$(2+\varepsilon)$的 PTAS 算法,并证明了对于固定机器数的问题不存在近似度低于 3/2 的 PTAS 算法.Diedrich 等人[19]针对机器数固定和不固定两种情况,提出了近似度为$(3/2+\varepsilon)$的 PTAS 算法,但是该算法应用了大量的枚举策略.Jansen 等人[20]改进了文献[19]的工作,将近似度降低到 3/2,并且避免了复杂的枚举操作.另外,Jansen 研究团队还研究了异构机器 MTS 问题的近似算法[21],并给出了可变性并行任务的$(3/2+\varepsilon)$近似度算法[22].关于 MTS 问题最新的研究见文献[23].

二阶段并行任务调度问题的研究一直集中在智能算法和启发式算法[38].目前,该问题仅有 1 项关于近似算法的研究.2011 年,UIUC 大学的 Moseley 和 Yahoo 研究院的 Dasgupta 等人[39]将 Map-Reduce 问题归结为一类二阶段并行任务调度问题,并给出了近似度为 12 的算法.而本文将 GPU 上负载调度问题归结为第 1 阶段单机、第 2 阶段有 m 台机器的二阶段并行任务调度问题,并给出了近似度为 3 的多项式时间算法.

6 结束语

本文给出了 GPU 上大规模并行负载调度问题的近似算法.文中综合考虑数据传输和负载执行时间,将 GPU 上负载调度问题归结为一类二阶段并行任务调度问题 $F2(1,P_m)|size_i|C_{\max}$.然后,将负载任务的时间信息和并行机器数与矩形域的二维空间联系起来,为此类问题建立了 2D 双层矩形 Strip-Packing 模型,进而将 $F2(1,P_m)|size_i|C_{\max}$ 中 makespan 最小化问题转化为列表 L 中矩形在有限域的最佳安置问题.基于可塑性假设,本文提出了一种多项式时间的近似算法,计算复杂度仅有 $O(n\log n)$.该算法采用二阶段策略,即,先在数据传输阶段进行负载排序,再在 GPU 内核中基于 FCFS 策略调度,因此易于在操作系统中实现.另外,算法相关的理论分析和实验结果也证明了这种两阶段调度策略能够保障 GPU 性能全局最优或近似最优.

致谢 感谢在 NASAC 2013 会议上,南京大学李宣东教授、东北大学于戈教授等出席了本文的报告,并提出了宝贵意见.感谢匿名审稿专家提出的建设性修改建议.

References:

[1] Lin YS, Yang XJ, Tang T, Wang GB, Xu XH. A GPU low-power optimization based on parallelism analysis model. Chinese Journal of Computers, 2011,34(4):706−716 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.00705]

[2] Lin YH, Kong FX, Xu HT, Jin X, Deng QX. Minimizing engergy consumption for linear speedup parallel real-time tasks. Chinese Journal of Computers, 2013,26(2):384−392 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.00384]

[3] Tan GZ, Sun JH, Wang BC, Yao WH. Solving Chinese postman problem on time varying network with timed automata. Ruan Jian Xue Bao/Journal of Software, 2011,22(6):1267−1280 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4033.htm [doi: 10.3724/SP.J.1001.2011.04033]

[4] Gregg C, Dorn J, Hazelwood K, Skadron K. Finegrained resource sharing for concurrent GPGPU kernels. In: Proc. of the 4th USENIX Conf. on Hot Topics in Parallelism. USENIX Association Berkeley, 2012. 10−16.

[5] Li T, Narayana VK, Araby EE, Ghazawi TE. Gpu resource sharing and virtualization on high performance computing systems. In: Proc. of the 2011 Int'l Conf. on IEEE Parallel Processing (ICPP). IEEE, 2011. 733−742. [doi: 10.1109/ICPP.2011.88]

[6] Kato S, Lakshmanan K, Kumar A, Kelkar M, Ishikawa Y, Rajkumar R. RGEM: A responsive GPGPU execution model for runtime engines. In: Proc. of the IEEE 32nd Real-Time Systems Symp. (RTSS). IEEE, 2011. 57−66. [doi: 10.1109/RTSS.2011.13]

[7] Kato S, Lakshmanan K, Rajkumar R, Ishikawa Y. Timegraph: GPU scheduling for real-time multi-tasking environments. In: Proc. of the 2011 USENIX Annual Technical Conf. (USENIX ATC11). USENIX Association Berkeley, 2011. 17−31.

[8] Kato S, Throw M, Maltzahn C, Brandt S. Gdev: First-class gpu resource management in the operating system. In: Proc. of the 2012 USENIX Conf. on Annual Technical Conf. (USENIX ATC 2012). USENIX Association Berkeley, 2012. 37−49.

[9] Liu W, Chen JJ, Kuo TW, Deng QX, Liu X. Optimize overall system performance through workload seqencing for gpus data offloading. In: Proc. of the 5th USENIX Workshop on Hot Topics in Parallelism (HotPar 2013). San Jose, USENIX Association Berkeley, 2013. http://hgpu.org/?p=9800

[10] Hong S, Kim H. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. ACM SIGARCH Computer Architecture News, 2009,37(3):152−163.

[11] Oğuz C, Ercan MF, Cheng TCE, Fung YF. Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. European Journal of Operational Research, 2003,149(2):390−403. [doi: 10.1145/1555815.1555775]

[12] Schuurman P, Woeginger GJ. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. Theoretical Computer Science, 2000,237(1):105−122. [doi: 10.1016/S0377-2217(02)00766-X]

[13] Hoogeveen JA, Lenstra JK, Veltman B. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. European Journal of Operational Research, 1996,89(1):172−175.

[14] Garey MR, Graham RL. Bounds for multiprocessor scheduling with resource constraints. SIAM Journal on Computing, 1975,4(2): 187−200. [doi: 10.4018/jgc.2012070106]

[15] Ludwig W, Tiwari P. Scheduling malleable and nonmalleable parallel tasks. In: Proc. of the 5th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 1994. 167−176. [doi: 10.1137/0204015]

[16] Jansen K, Porkolab L. Linear-Time approximation schemes for scheduling malleable parallel tasks. In: Proc. of the 10th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 1999. 490−498.

[17] Amoura AK, Bampis E, Kenyon C, Manoussakis Y. Scheduling independent multiprocessor tasks. Algorithmica, 2002,32(2): 247−261.

[18] Scharbrodt M, Steger A, Weisser H. Approximability of scheduling with fixed jobs. In: Proc. of the 10th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 1999. 961−962. [doi: 10.1007/s00453-001-0076-9]

[19] Diedrich F, Jansen K. Improved approximation algorithms for scheduling with fixed jobs. In: Proc. of the 10th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2009. 675−684.

[20] Jansen K, Prädel L, Schwarz UM, Svensson O. Faster approximation algorithms for scheduling with fixed jobs. In: Proc. of the Conf. of Computing: The Australasian Theory Symp. (CATS). Darlinghurst: Australian Computer Society, Inc., 2011. 3−9.

[21] Bougeret M, Dutot PF, Jansen K, Robenek C, Trystram D. Scheduling jobs on heterogeneous platforms. In: Proc. of the Computing and Combinatorics. Berlin, Heidelberg: Springer-Verlag, 2011. 271−283.

[22] Jansen K. A (3/2+ε) approximation algorithm for scheduling moldable and non-moldable parallel tasks. In: Proc. of the 24th ACM Symp. on Parallelism in Algorithms and Architectures. New York: ACM Press, 2012. 224−235. [doi: 10.1145/2312005.2312048]

[23] Jansen K. Approximation algorithms for scheduling and packing problems. In: Proc. of the Approximation and Online Algorithms. Berlin, Heidelberg: Springer-Verlag, 2012. 1−8. [doi: 10.1007/978-3-642-29116-6_1]

[24] Xie J, Wang X. Complexity and algorithms for two-stage flexible flowshop scheduling with availability constraints. Computers & Mathematics with Applications, 2005,50(10):1629−1638. [doi: 10.1016/j.camwa.2005.07.008]

[25] Sahni SK. Algorithms for scheduling independent tasks. Journal of the ACM, 1976,23(1):116−127. [doi: 10.1145/321921.3 21934]

[26] Hochbaum DS, Shmoys DB. Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM, 1987,34(1):144−162. [doi: 10.1145/321921.321934]

[27] Hall LA. Approximability of flow shop scheduling. Mathematical Programming, 1998,82(1-2):175−190. [doi: 10.1145/7531.7535]

[28] Williamson DP, Hall LA, Hoogeveen JA, Hurkens AJ, Lenstra JK, Sevast'janov SV, Shmoys DB. Short shop schedules. Operations Research, 1997,45(2):288−294.

[29] Jansen K, Sviridenko MI. Polynomial time approximation schemes for the multiprocessor open and flow shop scheduling problem. In: Proc. of the STACS 2000. Berlin, Heidelberg: Springer-Verlag, 2000. 455−465. [doi: 10.1287/opre.45.2.288]

[30] Gupta JND. Two-Stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society, 1988,39(4):359−364.

[31] Chen B. Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at one stage. Journal of the Operational Research Society, 1995,46(2):234−244. [doi: 10.1057/jors.1995.28]

[32] Schuurman P, Woeginger GJ. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. Theoretical Computer Science, 2000,237(1):105−122.

[33] Sevastyanov SV. An improved approximation scheme for the Johnson problem with parallel machines. Journal of Applied and Industrial Mathematics, 2008,2(3):406−420.

[34] Choi BC, Lee K. Two-Stage proportionate flexible flow shop to minimize the makespan. Journal of Combinatorial Optimization, 2013,25(1):123−134. [doi: 10.1134/S1990478908030113]

[35] Ruiz R, Vázquez-Rodríguez JA. The hybrid flow shop scheduling problem. European Journal of Operational Research, 2010,205(1): 1−18. [doi: 10.1007/s10878-011-9423-1]

[36] Emmons H, Vairaktarakis G. The Hybrid Flow Shop. In: Flow Shop Scheduling: Int'l Series in Operations Research & Management Science Vol. New York: Springer-Verlag, 2013. 161−187. [doi: 10.1016/j.ejor.2009.09.024]

[37] Saravanan M, Sridhar S. An overview of hybrid flow shop scheduling: sustainability perspective. Int'l Journal of Green Computing, 2012,3(2):78−91. [doi: 10.4018/jgc.2012070106]

[38] Oğuz C, Ercan MF, Cheng TCE, Fung YF. Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. European Journal of Operational Research, 2003,149(2):390−403. [doi: 10.1016/S0377-2217(02)00766-X]

[39] Moseley B, Dasgupta A, Kumar R, Sarlós T. On scheduling in map-reduce and flow-shops. In: Proc. of the SPAA. New York: ACM, 2011. [doi: 10.1145/1989493.1989540]

附中文参考文献:

[1] 林一松,杨学军,唐滔,王桂彬,徐新海.一种基于并行度分析模型的 GPU 功耗优化技术.计算机学报,2011,34(4):706−716. [doi: 10.3724/SP.J.1016.2011.00705]

[2] 林宇晗,孔繁鑫,徐惠婷,金曦,邓庆绪.线性加速比并行实时任务的节能研究.计算机学报,2013,26(2):384−392. [doi: 10.3724/SP.J.1016.2013.00384]

[3] 谭国真,孙景昊,王宝财,姚卫红.时变网络中国邮路问题的时间自动机模型.软件学报,2011,22(6):1267−1280. http://www.jos.org.cn/1000-9825/4033.htm [doi: 10.3724/SP.J.1001.2011.04033]

孙景昊(1985－),男,河北沧州人,博士,讲师,主要研究领域为实时系统调度算法,最优化理论,时间自动机.
E-mail: jhsun@mail.dlut.edu.cn

孟亚坤(1987－),女,博士生,主要研究领域为实时系统,多核系统调度.
E-mail: mengyakun@gmail.com

邓庆绪(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为实时嵌入式系统,可重构计算,物联网.
E-mail: dengqx@mail.neu.edu.cn