文章编号：2095-4980(2019)02-0299-07

多 Stream 并行 DAG 任务映射策略

王学成，马金全，李建军

(战略支援部队信息工程大学 信息系统工程学院，河南 郑州 450002)

摘　要： 伴随大数据量的应用任务在中央处理器(CPU)与图形处理器(GPU)组成的异构处理平台上的部署日益广泛，如何高效利用GPU硬件中的并行资源，成为亟待解决的问题。通过对单GPU任务映射策略进行研究，提出多Stream有向无环图(MS-DAG)任务映射策略。通过分析DAG图中的节点依赖关系，根据节点依赖关系的不同，划分合理的并行分支，利用多Stream流水线并行的方式，实现适合GPU硬件特点的任务映射策略。通过与HEFT在不同条件下的性能对比，可以看出：当HEFT算法中的各处理器性能不一致时，MS-DAG任务映射策略的任务映射效率相比HEFT算法有约10%的提升；当HEFT算法中的各处理器性能一致时，MS-DAG任务映射策略的任务映射效率相比HEFT算法有30%的提升。

关键词： CPU+GPU 异构平台；任务映射；有向无环图；并行计算

中图分类号： TN911.7　　**文献标志码：** A　　**doi：** 10.11805/TKYDA201902.0299

Multi-stream parallel DAG task mapping strategy

WANG Xuecheng, MA Jinquan, LI Jianjun

(School of Information System, PLA Information Engineering University, Zhengzhou Henan 450002, China)

Abstract: Utilizing Graphic Processing Unit(GPU) parallel resource efficiently as the application task of large data quantity deployed on the rapid development CPU+GPU heterogeneous platform became an urgent problem. A task mapping strategy of Multiple Stream-Direction Acyclic Graph(MS-DAG) is proposed, based on the research of task mapping strategy of single GPU. The task mapping strategy for GPU hardware is realized by analyzing the node dependencies in DAG graph, according to the difference of node dependency, dividing reasonable parallel branches and using multi stream pipelining. It shows that the task mapping efficiency of MS-DAG task mapping strategy is about 10% higher than that of Heterogeneous Earliest Finish Time(HEFT) algorithm when the performance of each processor is inconsistent in HEFT algorithm; and the task mapping efficiency of MS-DAG task mapping strategy is 30% higher than that of HEFT algorithm, when the performance of each processor in the HEFT algorithm is consistent.

Keywords: CPU+GPU heterogeneous platform; task mapping; Direction Acyclic Graph; parallel computation

近年来，伴随着中央处理器(CPU)与图形处理器(GPU)组成的异构处理平台的迅速发展，基于有向无环图(DAG)的任务映射策略在 CPU+GPU 异构平台上的应用日渐成为一个热点问题。关于异构平台的多任务映射策略以及多异构平台的任务映射策略的研究均已取得显著成就。常用的任务映射策略主要分为 2 大类：全局动态调度和全局静态调度[1]。全局动态调度算法是一种实时的任务调度算法，在进行任务调度时要消耗大量的处理器资源，算法复杂度较高，稳定性较差；相比于全局动态调度算法，全局静态调度算法具有较好的稳定性和较低的复杂度。

常用的静态调度算法主要有基于优先级的表映射算法[2-6]、聚类算法[7-11]、启发式搜索算法[12-15]、基于复制的任务映射算法[16-21]等。其中基于优先级的表映射算法的算法复杂度较低，在同等任务条件下，相比于其他 3 类算法花费时间少，效率高。而基于优先级的表映射算法的核心是任务优先级模型的构建，文献[2-6]根据任务映射的不同需求设计不同的优先级，提高任务映射的针对性；Topcuoglu 等提出的 HEFT 算法根据 DAG 图中的

收稿日期：2018-08-22；修回日期：2018-11-25

rank 值确定任务调度的优先级，任务映射效率突出，但当处理器性能一致时，映射策略的执行效果较差。虽然这些算法的提出在特殊的应用场景下确实有效提高了平台的执行效率，但是，针对 CPU+GPU 异构平台而言，由于 GPU 强大的并行处理能力，使得 CPU+GPU 异构平台的任务映射策略不仅要考虑跨平台的数据传输和不同平台计算能力等因素对异构平台的影响，还要考虑单 GPU 上计算能力相同、处理速度一致、节点之间的传输代价较小等特点对异构平台任务映射效率的影响。

为了解决单 GPU 上的 DAG 任务映射问题，结合单 GPU 的多任务并行处理能力，根据应用中数据流向建立有向无环图模型。通过分析有向无环图中节点与节点之间的连接关系，划分 DAG 层，依据数据的依赖关系划分不同的并行支路，利用统一计算架构(Compute Unified Device Architecture，CUDA)提供的多 Stream 并行方式，实现 DAG 在单 GPU 上的任务映射策略——MS-DAG。实验结果表明，在处理器性能相同和处理器性能不相同的条件下，MS-DAG 任务映射策略在不同节点数量下的任务映射效率均优于 HEFT 算法。

1 问题描述与数学基础

1.1 问题描述

伴随着并行计算技术的快速发展，CPU+GPU 异构平台在并行计算领域中占据越来越重要的地位，CUDA 的出现为 CPU+GPU 异构平台进行密集型数据处理提供了便捷的开发方式，同时其多粒度并行处理模式和多层次的内存结构使得数据并行、流水线并行以及任务并行这 3 种常用并行方式的实现变得更加轻松和高效[22]。CUDA 将平台分为 2 部分：host 端(CPU)和 device 端(GPU)，host 端主要进行初始化、设备的管理、数据的传输控制；device 端进行任务的执行，在 device 端上执行的函数称为 kernel 函数。在 device 端运行程序之前，需要先从 host 端完成数据的拷贝，数据完成后 host 端调用 kernel 函数。kernel 函数调用是一种异步的方式，即调用 kernel 函数后，host 控制权返回。CUDA 可以通过多 Stream 流水线并行的方式，实现多 kernel 函数的并行执行。根据 CUDA 提供的多种并行模式，可以在单 GPU 上实现多任务的并行，这是 CPU+GPU 异构平台与其他异构平台最大的不同。在单 GPU 上合理安排任务的执行顺序成为一个亟待解决的问题，而任务映射策略的目的就是通过一定的方式，合理安排任务的执行顺序，实现应用的高效执行。

任务映射分为独立任务映射和依赖任务映射。依赖任务映射通常也称为相关任务映射[23]。典型的相关任务映射模型都是建立在图的基础上，通常称它们为任务图，最常用的任务图是有向无环图(DAG)[24]，因此任务映射策略的研究常被转化为 DAG 优化问题。

1.2 数学基础

在数学和计算机科学中，DAG 是由集合的顶点和有向边组成，根据任务的特点需要，赋予节点和有向边不同的属性。任务映射的优化正是通过节点和有向边的属性设计映射策略实现高效的任务模型。任务映射 DAG 可以用四元组表示：$DAG=[V,E,C,T]$。其中 $V=\{v_0,v_1,\cdots,v_n\}$ 作为节点集合，表示子任务集合；$E=\{e_{ij}\}$ 表示有向边的连接关系，e_{ij} 表示数据流向是从 v_i 流向 v_j；$C=\{c_0,c_1,\cdots,c_n\}$ 中的元素表示对应节点的计算复杂度；$T=\{t_{ij}\}$ 表示数据由 v_i 流向 v_j 所需要的传输时间。

根据节点之间的连接关系可以确定数据之间的依赖关系。数据之间的依赖关系主要分为流依赖、反依赖、输出依赖以及输入依赖 4 种。假设有 2 个节点 v_i 和 v_j，$R(v_i)$ 表示 v_i 节点对数据 δ 的读取，$W(v_i)$ 表示 v_i 节点对数据 δ 的写入。则节点的依赖关系可以表示为：

1) 流依赖关系：$R(v_i)\cap W(v_j)\neq\varnothing$，且 $\exists e_{ji}\in E$，则称节点 v_i 流依赖于节点 v_j；

2) 反依赖关系：$W(v_i)\cap R(v_j)\neq\varnothing$，且 $\exists e_{ji}\in E$，则称节点 v_i 反依赖于节点 v_j；

3) 输出依赖：$W(v_i)\cap W(v_j)\neq\varnothing$，且 $\exists e_{ij},e_{ji}\in E$，则称节点 v_i 和 v_j 是输出依赖关系；

4) 输入依赖：$R(v_i)\cap R(v_j)\neq\varnothing$，且 $\exists e_{ij},e_{ji}\in E$，则称节点 v_i 和 v_j 是输入依赖关系。

通过分析数据依赖关系，可以确定 DAG 图中节点与节点之间的关系，为单 GPU 上的任务映射策略研究提供理论基础。

2 多 Stream DAG 任务映射策略建模

CPU+GPU 异构平台具有强大的并行计算能力，单 GPU 就可以实现多任务的并行。因此，为了更好地利用

单 GPU 的并行计算能力，需要合理规范任务在单 GPU 上的执行顺序。通过分析 CUDA 对 GPU 的开发方式，利用 CUDA 提供的多 Stream 流水线并行，实现 MS-DAG 任务映射策略。分析 DAG 中的节点连接关系，确定节点与节点之间的依赖关系，根据节点之间不同的依赖关系划分 Stream 支路，合理规划多任务并行执行顺序。

2.1 Stream 流水线并行开发

CUDA 的出现使 GPU 并行开发逐渐简单，它为用户提供了多层次的内存结构和多粒度的并行模式。而多 Stream 流水线并行开发方式，需要结合应用的不同要求，合理地规划内存结构和划分并行粒度。

通过分析 CUDA 提供的数据传输方式，选择合适的 CPU 到 GPU 数据传输方式。CUDA 为 CPU 端的数据存放提供了 2 种内存模式：Pageable Memory 和 Pinned Memory；以及 2 种数据传输的方式：同步数据传输和异步数据传输。同步传输和异步传输最大区别是控制权的返回时间，同步数据传输的控制权只有等数据传输完毕后才会返回；而异步数据传输，命令调用后控制权就返回。通过对 CPU 进行数据传输测试，可以看出 Pinned Memory 中的数据传输速度要快于 Pageable Memory 中的数据传输。

为了充分利用多 Stream 的并行能力，MS-DAG 任务映射策略研究异步数据传输的硬件资源和应用需求。从 CPU 端的 Memory 结构中发现，使用 Pinned Memory 进行异步数据传输的安全性要高于 Pageable Memory。由于 Pageable Memory 中的数据在进行异步传输时,可能会发生数据转移而造成数据传输错误,应用在 GPU 上执行时，需要先将应用所需的数据拷贝到 GPU 的显存中。数据的异步传输适合于可以将数据分批处理的应用,而对于某些必须要完成所有数据传输后才能进行各项数据处理的应用，异步传输和同步传输的效果差距较小。对于可以分批处理的应用，MS-DAG 任务映射策略利用多 Stream 的流水线并行，异步传输与异步调用相结合，实现数据传输和数据处理的折叠，提高效率，其传输和处理的流程如图 1 所示。

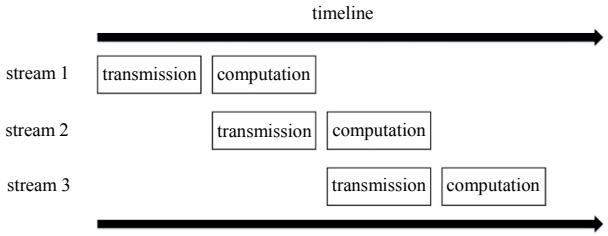

Fig.1 Diagram of multiple-stream pipeline
图 1 多 Stream 流水线并行示意图

2.2 依赖关系分析及支路划分

完成数据传输后，需要为 GPU 上的任务安排合理的执行顺序，即合理的任务映射策略。MS-DAG 任务映射策略通过分析应用 DAG 中的连接关系和节点之间的相互依赖关系，确定应用子任务的并行分支；利用 CUDA 提供的多 Stream 并行处理的技术，依据 MS-DAG 任务映射策略实现任务的有序高效执行。

任务间的数据交互是实现高效任务映射亟待解决的一个关键问题。单 GPU 上的任务交互过程就是一种跨 Stream 的任务交互过程，在单 GPU 上进行，数据交互代价较小。当在 DAG 中存在 2 个任务节点 v_i 和 v_j，且 $\exists e_{ji} \in E, R(v_i) \cap W(v_j) \neq \varnothing, I(v_i) > 1$ 时，则以节点 v_i 为新一层的开始进行分层，并根据每一层的输出节点数量确定任务执行的 Stream。当在 DAG 中存在 2 个任务节点 v_i 和 v_j，且 $\exists e_{ji} \in E$，v_j 的执行需要 v_i 的处理结果时，MS-DAG 任务映射策略采用事件驱动的 Stream 交互方式。当 Stream 中的任务队列中的任务节点需要调用其他层或其他 Stream 中的计算结果时，条件未满足之前，该 Stream 处于阻塞状态，直到条件满足继续执行，而其他输出节点的 Stream 执行队列不受影响。

在处理同一层任务节点的并行分支时，会涉及到读数据 $R(v)$和写数据 $W(v)$的操作。为了保证应用执行的正确性，需要保证节点对数据的操作符合应用执行顺序。而流依赖关系、反依赖关系以及输出依赖关系存在访存的先后顺序，MS-DAG 任务映射策略将具有流依赖关系、反依赖关系以及输出依赖关系的节点放入到同一任务队列中。而输入依赖关系，由于 2 个节点对同一部分数据进行读取操作，未对数据进行修改，可以将输入依赖关系的 2 个节点划分到不同的任务队列中。整体流程如图 2 所示。

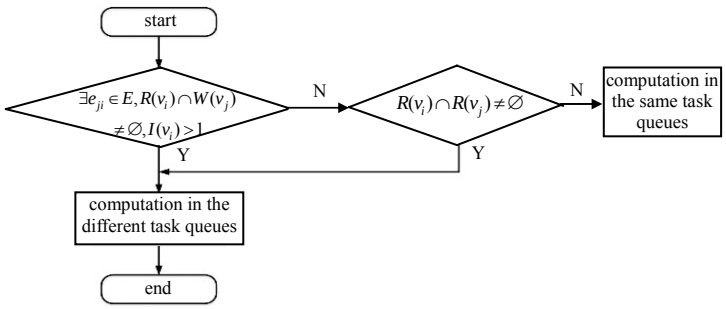

Fig.2 Flowchart of the MS-DAG.
图 2 MS-DAG 算法流程图

通过对算法进行分析，可以看出，DAG 中的节点 v_i 在执行之前需要经过 2 次判断，因此，MS-DAG 任务映射策略的算法复杂度近似为 $O(V)$，其中 V 表示 DAG 中节点的数量。

3 实验仿真及测试

为了评估 MS-DAG 任务映射策略的性能，将 MS-DAG 任务映射策略与 HEFT 算法进行对比；通过对典型 DAG 和随机 DAG 进行仿真，测试 2 个算法在应对不同节点数以及不同处理器数量和性能条件下的任务映射效率。并对 CUDA 中提供的 Pageable Memory 和 Pinned Memory 这 2 种内存进行数据传输测试，评估传输性能。

表 1 计算代价

Table1 Computation cost

task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

3.1 典型 DAG 图下的 HEFT 算法和 MS-DAG 任务映射策略对比

为了测试 MS-DAG 任务映射策略的有效性，利用 Topcuoglu H 等提出的典型 DAG 做模拟，比较 MS-DAG 任务映射策略与 HEFT 算法性能。HEFT 模拟 3 个不同处理器对典型 DAG 的任务映射，3 个处理器在处理不同节点的计算代价如表 1 所示。典型 DAG 如图 3 所示。

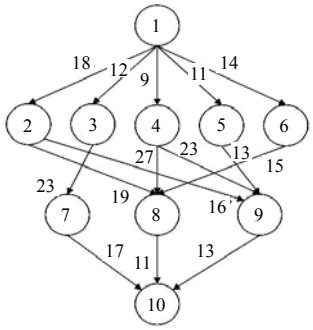

Fig.3 Classical DAG

图 3 典型 DAG

分别利用 HEFT 算法和 MS-DAG 任务映射策略对图 3 中的典型 DAG 进行任务部署。以表 1 中节点在各个处理器上的最高时间代价为 CPU+GPU 多 Stream 处理的时间代价。由于 MS-DAG 任务映射策略中任务的处理是在单 GPU 上完成，因此，任务与任务之间的数据传输时间可以忽略不计，其时间流水线分别如图 4 和图 5 所示。分析实验结果，可以看出 HEFT 算法完成图 3 所示的任务映射需要花费 80 个时间周期，MS-DAG 任务映射策略只需要花费 76 个时间周期。并且，仿真 MS-DAG 任务映射策略时假设多 Stream 中的节点计算效率要弱于 HEFT 中的各个处理器的处理速度，通过图 4 和图 5 的任务映射时间线可以看出 MS-DAG 任务映射策略在进行 DAG 任务映射时，效率优于 HEFT 算法。

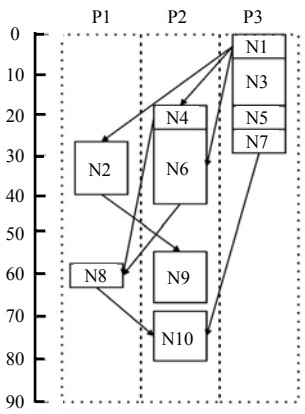

Fig.4 Timeline of the HEFT

图 4 HEFT 算法时间线

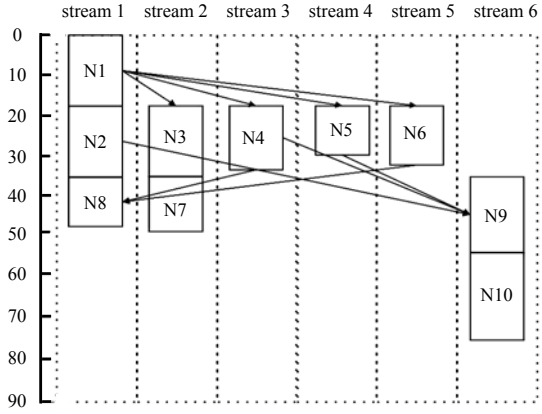

Fig.5 Timeline of the MS-DAG task scheduling strategy

图 5 MS-DAG 任务映射策略时间线

3.2 随机 DAG 下的 HEFT 算法和 MS-DAG 任务映射策略对比

为了测试 MS-DAG 任务映射策略的算法性能，探究随机 DAG 中节点数量以及处理器的数量对任务映射策略效率的影响，利用 HEFT 算法和 MS-DAG 任务映射策略进行测试。由于 MS-DAG 使用的是单 GPU 进行测试，因此，将多处理器对各个节点最大的计算代价作为 MS-DAG 任务映射策略的计算代价。在测试节点数量对算法性能影响时，固定处理器数量为 3；在测试处理器数量对算法性能的影响时，固定节点的数量为 100，进行仿真测试。

通过图 6 和图 7 的实验结果可以看出，MS-DAG 任务映射策略在不同节点数下的处理代价小于 HEFT 算法。在完成不同节点数的 DAG 任务映射时，MS-DAG 任务映射策略执行效率较 HEFT 算法提高了大约 10%。而对于随机的 DAG，传输代价为随机值，造成不同处理器下的 HEFT 算法性能波动较大，但由于 MS-DAG 任务映射策略是在单 GPU 上进行任务映射，任务之间的交互代价较小，使得 MS-DAG 任务映射策略相比于 HEFT 算法性能更优。

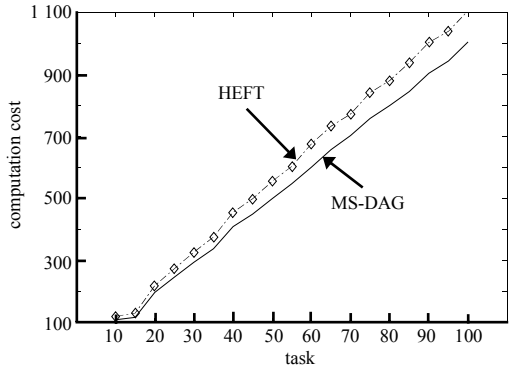

Fig.6 Comparing the algorithm of HEFT with the task scheduling strategy MS-DAG on different node numbers
图 6 不同节点数 HEFT 算法和 MS-DAG 任务映射策略对比

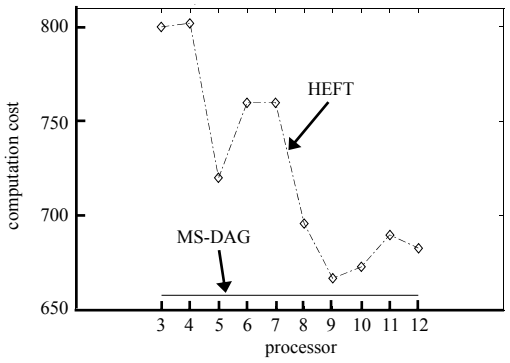

Fig.7 Comparing the algorithm of HEFT with the task scheduling strategy MS-DAG on different processor numbers on 100 nodes
图 7 节点数为 100 的条件下不同处理器数 HEFT 算法与 MS-DAG 任务映射策略对比

3.3 相同处理器条件下的 HEFT 算法和 MS-DAG 任务映射策略对比

为了探究 2 个算法在处理器性能一致的条件下的性能效果，分别测试了 HEFT 算法和 MS-DAG 任务映射策略在处理器数量固定为 3，各个处理器性能一致的条件下，随机 DAG 中节点数量对算法性能的影响。测试结果如图 8 所示。从图 8 的实验结果可以看出，HEFT 算法在处理器性能一致的情况下，各个节点在不同处理器上的代价一致，对节点优先级的定义效果较差，数据之间的交互代价较大，严重影响了算法的性能。MS-DAG 任务映射是针对单 GPU 的任务映射算法，根据节点之间的依赖关系确定任务执行顺序，且数据存储在单 GPU 上，数据之间的交互代价较小。通过仿真对比可以看出：在处理器性能相同的条件下，MS-DAG 任务映射策略与 HEFT 算法相比，性能有约 30%的提升。

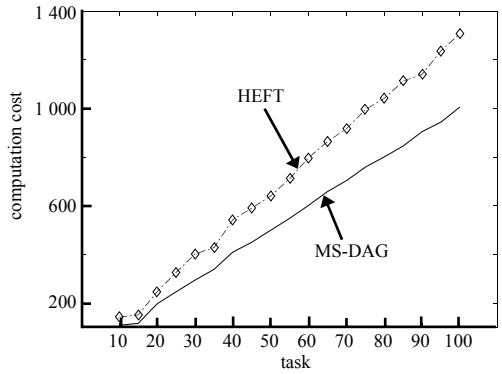

Fig.8 Comparing the algorithm HEFT with task scheduling strategy MS-DAG based on the same productivity processors
图 8 处理器性能相同的条件下 HEFT 算法与 MS-DAG 任务映射策略对比

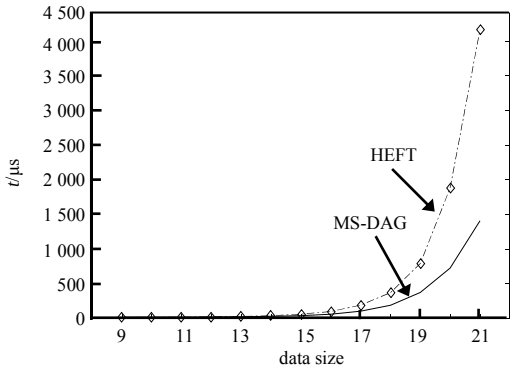

Fig.9 Comparing data transform ratio between the Pinned Memory and Pageable Memory
图 9 Pinned Memory 与 Pageable Memory 数据传输对比

3.4 数据传输测试

由于 CPU+GPU 异构平台中 GPU 在进行密集型计算之前需要将数据从 CPU 上传递到 GPU 中，为了测试不同的 CPU 端的内存存储方式对实验的影响，设计了 Pinned Memory 与 Pageable Memory 这 2 种内存传输效率的测试。将 2 种内存存放相同数量的 double 型数据，采用同样的传输方式，调用相同的 cudaMemcpy()函数进行数据传输，记录传输 10 000 次数据后的总时间，求其均值，得到近似的单次传输时间，结果如图 9 所示。

从图 9 可以看出，在小数据量的传输过程中，2 种内存的传输效率近乎相同，随着数据量的增加，可以很明显地看出：Pinned Memory 的传输效率要远远高于 Pageable Memory。因此，使用 Pinned Memory 来进行数据的存储可以有效提高 CPU+GPU 异构平台的数据传输效率，实现更高的实时性。

4 结论

针对 CPU+GPU 异构平台上的单 GPU 的任务映射策略算法，利用 CUDA 提供的多 Stream 流水线并行方式，分析节点之间的数据依赖关系，提出了 MS-DAG 任务映射策略，实现了 DAG 图与 GPU 硬件特点的耦合，优化

了 CPU+GPU 异构平台下任务执行效率。性能对比分析结果表明，MS-DAG 任务映射策略的任务映射效率相比处理器性能不一致条件下的 HEFT 算法有了约 10%的提升，相比处理器性能一致条件下的 HEFT 算法有了约 30%的提升，对 GPU 阵列条件下的任务映射策略研究具有重要的指导意义。

参考文献：

[1] 陆俊江,汤俊,张曦,等. 一种雷达信号处理多任务调度算法[J]. 太赫兹科学与电子信息学报, 2012,10(4):460-464. (LU Junjiang,TANG Jun,ZHANG Xi,et al. Multi-task scheduling algorithm for radar signal processing[J]. Journal of Terahertz Science and Electronic Information Technology, 2012,10(4):460-464.)

[2] MU P,NEZAN J F,RAULET M. A list scheduling heuristic with new node priorities and critical child technique for task scheduling with communication contention[J]. Lecture Notes in Electrical Engineering, 2013(73):217-236.

[3] ALEBRAHIM S,AHMAD I. Task scheduling for heterogeneous computing systems[J]. Journal of Supercomputing, 2017,73(6): 2313-2338.

[4] TOPCUOGLU H,HARIRI S,WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel & Distributed Systems, 2002,13(3):260-274.

[5] ILAVARASAN E,THAMBIDURAI P,MAHILMANNAN R. High performance task scheduling algorithm for heterogeneous computing system[C]// International Conference on Algorithms and Architectures for Parallel Processing. Melbourne. Australia:[s.n.], 2005:193-203.

[6] ILAVARASAN E,THAMBIDURAI P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments[J]. Journal of Computer Science, 2007,3(2):94-103.

[7] SHIN I,EASWARAN A,LEE I. Hierarchical scheduling framework for virtual clustering of multiprocessors[C]// Euromicro Conference on Real-Time Systems. Prague,Czech Republic:IEEE, 2008:181-190.

[8] TSAI Y L,LIU H C,HUANG K C. Adaptive dual-criteria task group allocation for clustering-based multi-workflow scheduling on parallel computing platform[J]. Journal of Supercomputing, 2015,71(10):3811-3831.

[9] GUO F,YU L,TIAN S,et al. A workflow task scheduling algorithm based on the resources' fuzzy clustering in cloud computing environment[J]. International Journal of Communication Systems, 2015,28(6):1053-1067.

[10] KANEMITSU H,HANADA M,NAKAZATO H . Clustering-based task scheduling in a large number of heterogeneous processors[J]. IEEE Transactions on Parallel & Distributed Systems, 2016,27(11):3144-3157.

[11] HUAN W,HONG-HUI L I,JUN-WEN Z. Cloud task scheduling algorithm based on modified K-means clustering[J]. Computer and Modernization, 2017(2):106-115.

[12] ZHANG L,CHEN Y,SUN R,et al. A task scheduling algorithm based on PSO for grid computing[J]. International Journal of Computational Intelligence Research, 2008,4(1):37-43.

[13] JIANG Y S,CHEN W M. Task scheduling for grid computing systems using a genetic algorithm[J]. Journal of Supercomputing, 2015,71(4):1-21.

[14] SHRIYA S,SHARMAh R S,SUMIT S,et al. Directed search-based PSO algorithm and its application to scheduling independent task in multiprocessor environment[C]// Proceedings of the 4th International Conference on Frontiers in Intelligent Computing:Theory and Applications. Bhubaneswar,India:[s.n.], 2016:206-217.

[15] XU A,YANG Y,MI Z,et al. Task scheduling algorithm based on PSO in cloud environment[C]// Ubiquitous Intelligence & Computing & IEEE Intl Conf on Autonomic & Trusted Computing & IEEE Intl Conf on Scalable Computing & Communications & Its Associated Workshops. Guangzhou,China:IEEE, 2016:108-119.

[16] ATEF A,HAGRAS T,MAHDY Y B,et al. Lower-bound complexity algorithm for task scheduling on heterogeneous grid[J]. Computing, 2017,99(11):1125-1145.

[17] ZHANG W,HU Y,HE H,et al. Linear and dynamic programming algorithms for real-time task scheduling with task duplication[J]. Journal of Supercomputing, 2017(1):1-16.

[18] RUAN M,LI Y,ZHANG Y. A duplication task scheduling algorithm in cloud environments[C]// International Conference on Intelligent Data Engineering and Automated Learning. Xiamen,China:[s.n.], 2016:285-292.

[19] BHATTACHARYA A,BANERJEE A,DE P. Scheduling with task duplication for application offloading[C]// Consumer Communications & NETWORKING Conference. Las Vegas,NV,USA:IEEE, 2017:678-683.

[20] GUPTA I,KUMAR M S,JANA P K. Task duplication-based workflow scheduling for heterogeneous cloud environment[C]// Ninth International Conference on Contemporary Computing. Noida,India:IEEE, 2016:1-7.

(下转第 314 页)