文章编号：0253-374X(2016)12-1949-07 DOI:10.11908/j.issn.0253-374x.2016.12.021

基于异构平台的并行最大最小蚁群算法

黄震华[1]，赵振岐[1]，林培裕[1]，梅建华[2]
(1.同济大学 电子与信息工程学院，上海 200092；2.上海昕煜实业发展有限公司，上海 200062)

摘要：最大最小蚂蚁系统(Max-min Ant System，MMAS)是一种性能优良的启发式算法，常用于解决组合优化问题.当解决的目标问题规模较大、迭代轮次较多时，最大最小蚁群算法存在运行时间长的缺点.试验以开源串行包ACOTSP为基准，利用GPU多线程并发的优势，采用并行蚂蚁策略将MMAS在CPU-GPU协同异构计算平台上并发实现.算法在GPU上运行时的影响因素，如数据传输、内存层次、库函数调用等，也得到有效分析，并作出针对性优化.试验最终取得了高达13倍的加速，表明并行MMAS策略具有高效性和实用性.

关键词：并行计算；异构平台；最大最小蚁群系统；加速比
中图分类号：TP301.6 **文献标志码**：A

Parallel Max-min Ant System Based on Heterogeneous Platform

HUANG Zhenhua[1], *ZHAO Zhenqi*[1], *LIN Peiyu*[1], *MEI Jianhua*[2]
(1. College of Electronics and Information Engineering, Tongji University, Shanghai 200092, China; 2. Shanghai Xinyu Industrial Development Co., Ltd., Shanghai 200062, China)

Abstract: Max-min Ant System is a kind of heuristic algorithm with excellent performance, which is commonly used to solve combinatorial optimization problems. But it costs a long time when scale of the target problem is large as well as iterations are a lot. The experiment took the open source packet ACOTSP as a reference, used the advantage of multi-threaded GPU, and implemented ACO algorithm on CPU-GPU platform by parallel ants strategy. While the parallel algorithm is running on GPU, we also analyzed the impact factors carefully, such as data transmission, memory hierarchy, library calls et al, and made useful optimization. Eventually, the experiment made 13 times speedup, proving the parallel strategy is highly efficient and applicable.

Key words: parallel computing; heterogeneous platform; Max-min Ant System; speedup ratio

蚁群优化算法(ACO)[1]来源于真实蚂蚁的觅食行为，常用于解决组合优化问题.经过改进而产生的最大最小蚂蚁系统(MMAS)[2]由于其性能优良在车辆调度[3]、路径规划[4]、服务组合[5]等方面都得到成功应用.然而，传统蚁群算法在解决规模较大、迭代次数较多的目标问题时，都存在算法运行时间长、效率不高的缺点.并行ACO算法将组合优化问题的求解过程分解为多个可并行执行的逻辑单元，每个逻辑单元包含一定的计算量[6].将并发逻辑单元的计算量称为并发粒度.通常，细粒度并行蚁群优化算法使用并行蚂蚁策略，粗粒度算法使用并行蚁群策略.

并行蚂蚁策略：细粒度并行蚂蚁策略仅使用一个蚁群，以蚂蚁为并发逻辑单元，最初被Bullnheimer等[7]采用.基于消息传递及分布式内存架构，对蚂蚁系统提出了2种并行策略：第1种策略粒度较细，以主从(master-slave)模式将蚂蚁分配给不同的处理单元来加速运算，在每轮迭代中，master将信息素值广播给slave，并行计算路径完成之后又将该值反馈给master，这些全局通信和同步带来了明显的时间消耗；第2种策略提升算法粒度，经过一定迭代次数后再进行master和slave间的信息反馈，以此减少通信时间，从而提高运行速度.对于解的质量方面，Fu[8]针对并行蚂蚁方案提出了新的随机选择算法——AIR(All-In-Roulette)，旨在通过提升随机数的质量来影响和提高解的质量，防止系统的早熟现象.

收稿日期：2015-12-23
基金项目：国家自然科学基金(61272268)；上海市青年科技启明星计划(15QA1403900)；霍英东教育基金会高等院校青年教师基金(142002)；教育部新世纪优秀人才支持计划(NCET-12-0413)；同济大学中央高校基本科研业务费专项资金
第一作者：黄震华(1981—)，男，副教授，工学博士，主要研究方向为信息推荐、分布式计算、数据挖掘.
E-mail：huangzhenhua@tongji.edu.cn
通讯作者：赵振岐(1992—)，男，硕士生，主要研究方向为高性能计算、数据挖掘.E-mail：zhaozhenqi@tongji.edu.cn

并行蚁群策略:粗粒度并行蚁群算法以蚁群作为并发逻辑单元.该方案同样基于消息传递及分布式内存的计算模式,旨在将所有蚁群执行在不同的处理单元,被 Stützle[9] 首先引入. Middendorf 等[10] 扩展了该方法,引入了4种蚁群间的信息交换方式:交换全局最优解、循环交换局部最优解、交换迁移、交换局部最优解和迁移,结果表明并行蚁群策略可以有效降低多个蚁群副本间的通信量,减小同步开销.

对于执行单元分类,又可以分为 CPU-GPU 异构平台和 CPU 集群2种.

王诏远等[11] 基于内存计算框架 Spark 实现了 ACO 算法,把蚂蚁封装为弹性分布式数据集,将信息素矩阵等全局信息利用 Broadcast 广播共享,并通过调用 Spark 内置的应用接口实现解构造的并行化,最终取得了10倍以上的加速.

Delévacq 等[12] 以开源串行包 ACOTSP 为基准,采用了并行蚂蚁和并行蚁群2种方式,实现了基于 Fermi 架构 GPU 的最大最小蚂蚁系统,并详细阐述了试验过程.在采用并行蚂蚁(ant-thread)方案时,最高取得5.84倍的加速比.

虽然王诏远等[11,13] 将蚁群算法在 Hadoop 和 Spark 集群上实现,并取得了10倍以上的加速,但是硬件代价很大,所能解决的旅行商问题的城市规模也很有限.利用 CPU-GPU 异构平台实施蚁群算法,不仅具有代价优势,而且在城市规模增加时能够提升性能.而以往蚁群算法在 GPU 上实现时虽然将计算量最大、消耗时间最长的路径建立阶段并发[12,14-15],但并未对信息素矩阵的更新过程进行加速,Delévacq 等[12] 所采用的 ant-thread 方案的加速比也比较有限.

在对前文文献进行理解和总结的基础上,本文以经典组合优化问题——旅行商问题(Traveling Salesman Problem,TSP)[16] 为目标,基于 CPU-GPU 异构平台运行并行版本的 MMAS 算法求解该问题,相较于传统方式并发 MMAS 的路径建立阶段,为避免数据传输延迟而将信息素更新阶段也并发实现,并对 MMAS 在 GPU 上运行时的优化策略进行深入探究,最终算法取得了显著的性能提升.以 Stützle[9] 所做的原始工作为基准,借助 GPU 多线程并发的优势,以并行蚂蚁方式对 ACO 算法中的 MMAS 进行加速.按照 Delévacq 等[12] 的建议,将不利于并发操作的 local search[17] 去除,运行基础版本的蚁群算法.借鉴 Dawson 等[18] 采用蚁群算法完成并行边缘检测的案例,通过控制信息素和距离的权重、迭代次数和蚂蚁数目来维持解的质量,提高运行速度和试验效果.

1 问题描述

1.1 旅行商问题

旅行商问题可建模为图论中的 Hamilton 最小圈问题,被证明具有 NP(Non-Deterministic Polynomial)计算复杂度[19].TSP 问题可抽象表示为 $G=(V,E)$,其中 V 为顶点集,包含旅行商需要访问的所有城市(城市数量为 n);E 为边集,包含任意2个城市之间的旅行代价(通常是距离).TSP 问题的可行解是遍历每个城市的一条 Hamilton 回路,$p=\{V_0,V_1,\cdots,V_{n-2},V_{n-1}\}$,其中 p 为路径,$\{V_0,V_1,\cdots,V_{n-2},V_{n-1}\}$ 是集合 V 中所有城市的一个置换;解的代价函数为 $c(p)=\sum_{i=0}^{n-2}e_{v_i,v_{i+1}}+e_{v_{n-1},v_0}$,$e_{v_i,v_{i+1}}$ 为连接顶点 v_i 和 v_{i+1} 之间的边.TSP 问题的优化目标是求代价最小的可行解,即得出 $\arg_p\min(c(p))$,p 为可行解.本文着重解决以无向图表示的对称 TSP 问题.

1.2 蚁群算法

算法主要分为初始化、路径建立和信息素更新3个阶段.利用 MMAS 算法解决 tsp 问题的主要步骤伪代码如下:

```
Initialize_CPU_memory_and_constants();
While( ! termination_condition() ){
    //ants construct tours for tsp
    TourConstruction();
    PheromoneUpdate();
}
```

1.2.1 初始化

初始化阶段主要完成路径建立之前的数据准备工作.包括读取 tsp 文件,计算距离矩阵(distance),初始化信息素矩阵(pheromone),申请蚂蚁数组(ant_array)等操作.

1.2.2 路径建立

该步骤对应伪代码中的 TourConstruction().

每只蚂蚁具有如下属性:走过的路径、路径的长度、走过城市的数量、当前所在城市编号、未访问城市的集合.蚂蚁搜索路径流程如下:①随机选择一个城市作为蚂蚁的出发点;②蚂蚁依据状态转换规则(式(1))从未访问城市集合中用“转轮盘”方式[8] 选

择下个待访问城市；③循环执行步骤②直到蚂蚁周游所有城市 1 轮；④将蚂蚁路径中的城市按照距离依次求和；⑤判断是否全部蚂蚁都建立了路径，如果为真，执行步骤⑥，否则转步骤①；⑥通过比较所有蚂蚁的路径长度获得本轮最优路径.

在状态转换规则中，一只在城市 i 的蚂蚁 k 通过式(1)计算访问下个城市 j 的概率为

$$p_{ij,k}=\frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{j\in N}\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}} \tag{1}$$

式中：τ 为信息素；η 为城市间距离的倒数；i,j 为城市编号；N 代表尚未访问的城市集合；α,β 分别为 τ 和 η 的权值. τ 和 η 的值分别从信息素矩阵(pheromone)和距离矩阵(distance)中获得，而 α,β 的值会影响解的收敛速度.

一次迭代的结果是生成本轮最优路径，这只是寻找最优路径过程中的一个步骤. 在实际算法中需要设置蚂蚁搜索的迭代次数或其他限制条件，即伪代码中的 termination_condition()，从而体现出信息素对蚂蚁形成最优解的引导作用，以便控制解的收敛速度，同时也防止程序无限执行下去.

1.2.3　信息素更新

在得出本轮最优路径后，下一步要执行信息素矩阵的更新操作，对应伪代码中的 PheromoneUpdate(). 该阶段根据路径建立阶段的解对信息素矩阵进行更新，又分为信息素的蒸发与增强 2 个步骤.

(1)蒸发. 信息素矩阵的每个值都乘以参数 γ，即

$$\tau_{ij}(t+1)\leftarrow\tau_{ij}(t)\cdot(1-\gamma) \tag{2}$$

式中：$\tau(t)$ 为第 t 轮迭代的信息素值，$\gamma\in(0,1)$，该步骤使得质量较差的解随着时间推移而被忘记.

(2)增强. 包括经典蚁群(Ant System)、MMAS、精英蚂蚁系统(Elitist Strategy for Ant System, EAS)及基于排序的蚂蚁系统(Rank Based version AS, ASrank)[14]等方法.

本文采用的是 MMAS，特征是设置了信息素的最小值 $\tau_{\min}$ 和最大值 $\tau_{\max}$，且只有全局最优或每轮最佳的路径沉积信息素[16]. 操作如下：

$$\tau_{ij}(t+1)\leftarrow\tau_{ij}(t+1)+\Delta\tau_{ij,\text{best}}(t) \tag{3}$$

式中：$\Delta\tau_{ij,\text{best}}(t)=1/\text{len}_{\text{best}}(t)$，$\text{len}_{\text{best}}(t)$ 是第 t 轮迭代产生的最优路径的长度.

因此蚂蚁建立的路径越短，该条路径沉积的信息素就越多. 一般情况下，被多数蚂蚁经过且属于最短路径的边能够沉积更多的信息素，因此更有可能在未来的迭代中被其他蚂蚁选择. 从该种意义上，信息素 τ_{ij} 代表了一只当前在城市 i 的蚂蚁在下个城市选择 j 的可取性.

2　并行蚁群优化算法

在国际高性能计算组织 HPC TOP 500 公布的 2015 年榜单中，由中国国防科技大学研制的超级计算机“天河二号”依靠在 Linpack 基准测试最新版本中每秒 33.86 千万亿次的表现再度荣登榜首，美国的超级计算机“泰坦”以每秒 17.59 千万亿次的浮点运算能力居于亚军. 综观整个榜单，除了使用数以万计的 Intel CPU 以外，大多数超级计算机还采用了 Nvidia GPU 作为协处理器来提高性能，展现出很强的运算能力.

如今 GPU 在异构计算平台加速上的应用已经越来越广泛. 相比于 CPU，GPU 将更多的晶体管用于数据处理，而非数据缓存和流控制，因此计算能力突出. 其多流处理器、多线程的特点注定其适合并发操作. 虽然主频略低于 CPU，但其并发优势却令 CPU 望尘莫及. 影响 GPU 加速的瓶颈往往在于数据传输的时间，因此 GPU 更加适合处理数据传输次数较少而并行度高、计算密集的任务.

通常将 CPU 所在机器称为主机，将 GPU 称为设备[20]. 在主机上实现 ACO 算法之后，采用 CPU-GPU 计算平台对 ACO 算法进行提速，并测试不同的 GPU 内存结构以及不同参数对算法运行速度的影响，并对 2 种平台下的试验结果进行比较，主要包括运行速度、解的质量等方面.

2.1　并行蚂蚁策略

由于每只蚂蚁独立地搜索路径，ACO 算法的路径建立阶段本身具有良好的可并行性. 但由于当前 GPU 架构的限制，还无法将 ACO 算法完全运行在 GPU 上. 依然要采取 CPU 作为主机、GPU 作为设备的协同计算模式，将耗时最多的可并行部分在设备上运行，主机负责核函数调度和必要的串行处理.

采取并行蚂蚁策略，以 GPU 作为协处理器，为每只蚂蚁分配一个 CUDA 线程，并设置一定的迭代次数. 算法伪代码如下：

```
Initialize_CPU_memory_and_constants();
copy_Distance_to_GPU();
Pheromone_initialize_on_GPU();
cuRand_generate_randoms_on_GPU();
While( ! termination_condition() ){
  //each ant corresponds to a CUDA thread
```

```
  antSearch_Kernel<<<ceil( ants/N1 ),
    N1>>>( Distance, Pheromone );
  //pheromone update in two steps
  pheromoneEvaporate_Kernel<<<
    ceil( cities * cities/N2 ), N2>>>
( Pheromone );
  pheromoneEnhance_Kernel<<<1,
    1>>>( Pheromone ); //single thread on GPU
  copy_tour_from_GPU();
}
```

2.1.1 初始化

与串行版本一致,首先要进行 tsp 数据初始化.不同之处在于,信息素矩阵在设备端进行初始化(τ_0),而距离矩阵由主机端计算完毕后传送到设备端.通过数组对每只蚂蚁进行编号,为了在 GPU 上寻址的方便,需要申请一块地址连续的空间存储蚂蚁相关数据.其次,该步骤在 GPU 端生成了蚂蚁搜索所需要的全部随机数,并存储在一个显存数组中,供程序需要时读取.在初始化 tsp 数据之后,执行并行 ACO 算法的路径搜索过程,所有步骤均在设备端(GPU)实现.流程如图 1.

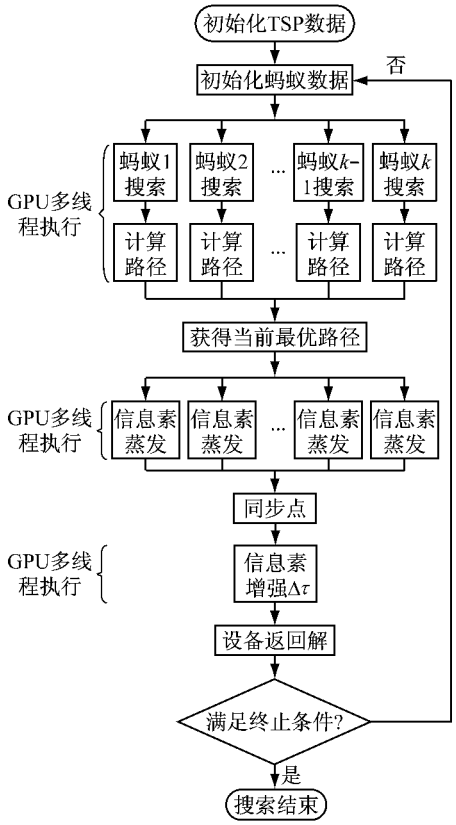

图 1 并行 ACO 算法流程

Fig.1 Procedures of parallel ACO algorithm

2.1.2 路径建立与信息素更新

并行蚁群算法最大的特点是为所有蚂蚁分配线程,然后同时进行路径建立过程.这能够大大缩短路径建立过程消耗的时间,同时掩盖数据传输带来的延迟.所有步骤均在 GPU 上进行,能最大限度地减少主机与设备间的数据传输.路径建立及信息素更新采用多线程并发形式执行,充分发挥 GPU 运算能力,而计算最优路径则以单线程方式在 GPU 上运行.

(1)初始化蚂蚁数据.利用 GPU 生成的随机数同时为每只蚂蚁随机选择城市作为出发点.

(2)蚂蚁进行一次搜索.即蚂蚁移动到下个待访问城市,所有蚂蚁同时执行该步骤.原理同串行算法.

(3)循环执行步骤(2)直到蚂蚁周游全部城市 1 遍;为了保证并行蚂蚁都建立自己的路径以及数据安全,在该步骤结束时执行_syncthreads()(线程同步)操作.

(4)计算蚂蚁路径长度.

(5)搜索当前蚂蚁建立的最优路径.

(6)进行信息素矩阵更新.分为 2 步,首先对所有路径同时执行信息素蒸发操作,然后按照 MMAS 策略选择当前最优路径沉积信息素.该步骤对于之后蚂蚁建立路径具有引导作用.

(7)判断是否满足终止条件.

步骤(1)至(5)对应图 1 中核函数 antSearch_Kernel,步骤(6)对应 pheromoneEvaporate_Kernel,pheromoneEnhance_Kernel.核函数参数如 ants/N1,N1,它们分别表示线程块数和每个线程块中的线程数,其中 ants 代表蚂蚁数量.核函数的线程组织实际是对串行算法中 for 循环的分解,在尽可能增大数据并行度的同时,避免因数据相关性引起“写冲突”,从而导致解的质量下降.其中,pheromoneEnhance 的并发性是由 MMAS 的特点决定的.由于每轮迭代结束后只有建立当前最优路径的蚂蚁执行信息素增强的操作,不存在 2 只蚂蚁同时修改信息素矩阵的情况,因此避免了“写冲突”.其并发点在于可以对当前最优路径的不同路段同时进行信息素的增强操作.

如果给定城市规模为 n,则解空间的大小为 $n!$.由于采用启发式解法,MMAS 求解 TSP 问题的时间复杂度为 $O(mn^2)$,其中 m 是蚂蚁数量.由分析可知,路径建立阶段的时间复杂度为 $O(n^2)$,且需要重

复 m 次. 而算法其他部分所需计算量均少于建立阶段，如路径评估的时间复杂度为 $O(mn)$，信息素挥发的时间复杂度为 $O(n^2)$. 假设有 q 个可用的并行处理单元，理论上算法时间复杂度可以降为 $O(mn^2q^{-1})$. 在并行 MMAS 策略中，可用的并行处理单元的数目即为核函数的 2 个参数(线程、线程块数)的乘积.

2.2　优化技术

经过对并行程序的分析，将影响加速比的因素列举如下，并给出相应的优化策略.

(1)数据传输. 在主机和设备之间传送数据造成的延迟是限制程序性能提升的重要因素. 以往 ACO 程序的任务划分是将具有并行特性、且消耗时间最多的路径建立阶段在 GPU 上并发，而对最优路径的求解和信息素的更新在 CPU 上进行. 由于距离矩阵和信息素矩阵的规模均为 n^2，随着城市规模的增加，二者的大小以平方形式增长. 考虑到数据传输带来的延迟以及充分发挥 GPU 的运算能力，将信息素的更新步骤在 GPU 上进行多线程并发，而最优路径求解以单线程方式在 GPU 上执行. 每轮迭代结束时只传送当前最优解，如此一来距离矩阵和信息素矩阵就可以常驻设备内存，最大限度避免设备和主机之间的数据传输.

(2)幂次计算. 每只蚂蚁在计算状态转换规则时都要用到 α 和 β，由于是幂次运算，所以存在相当大的浮点计算量. 以往蚁群算法在计算状态转换规则(式(1))时，对于幂次的计算均通过 C 库函数(头文件“math. h”)中的 pow 函数来完成. 经研究发现，调用 pow 函数并传参(双精度)的开销较大，会消耗许多寄存器，并且会破坏 CPU 的分支预测和缓存优化. 而在 GPU 端调用该库函数由于需要启动总线，开销更是成倍增加. 但是 α 和 β 的值往往是较小的整数，对于概率 p 的计算可以简化为手动相乘，这样一来大大减少了计算量，非常有利于程序在 GPU 上速度的提升. 而 pow 函数最好在求非整数幂时采用.

(3)随机数. 蚂蚁在选择出发城市和下个城市时需要用到随机数. 随机数由主机生成后传送给设备的开销较大，由 GPU 调用 CUDA 随机数库 cuRand 生成随机数可以节省时间.

(4)城市数目与蚂蚁只数. CUDA 文档[21]强烈建议使用每块的线程是线程束(warp，由连续的 32 个线程组成)的倍数，因此选择与城市数目相接近的 2^n 作为蚂蚁数目，以便最大限度提升计算效率. 同时适当增加蚂蚁数目也能够加快解的收敛速度.

(5)并行度. 采用单个蚂蚁对应单一线程的方式，如果在一个线程块(block)中设置的线程数过多，会由于线程间的资源竞争(register)而导致程序运行速度减慢. 同一 block 中的所有线程都会被分配到同一个处理器核上运行，共享有限的存储资源. 应根据蚂蚁数目设置相应 block 中的 thread 数量.

(6)内存层次. 在 GPU 端，相比于全局内存(global memory)，纹理内存(texture memory)是有缓存(cache)的. 如果同一个 warp 内 thread 的访问地址很相近，那么访问速度更快，性能会更高. 因此使用纹理内存的效果要好于全局内存. 本文对 2 个最大的矩阵——距离矩阵和信息素矩阵，采用纹理内存进行存储.

3　试验设计与结果分析

3.1　试验设计

利用 Linux 性能分析工具 Gprof[22]对 ACO 算法进行分析，能够找到消耗最多运行时间的函数. 基于不同的 tsp，分析结果显示 TourConstruction()阶段占总运行时间的 95%以上，并且随着城市数目增加该比例呈上升趋势. 由 Amdahl 加速比定律[23]可知，如果将 TourConstruction()阶段并发，理论上最大加速比可达 10 倍以上.

由于本文尽力使并行 MMAS 各计算阶段的前驱后继关系与原始串行算法相照应，宏观上二者具有相同的执行逻辑，且 GPU 的运算精度与 CPU 相当[21]，因此在提高加速比的同时能够保持解的质量. 根据并行蚂蚁思想，以单个蚂蚁对应单一线程的方案实现了基于 CPU-GPU 异构平台的并行 MMAS 算法.

试验中相关参数的值设置如下：$\alpha=2.0$，$\beta=3.0$，$\gamma=0.1$，迭代次数为 1 000. 进行测试的 tsp 文件均来自通用测试集 TSPLIB[24]，并选择与城市数目相接近的 2^n 作为蚂蚁数目. 由于每次试验过程中所产生的随机数不尽相同，导致结果会有细微差异，因此取 5 次测试结果的平均值作为最优路径长度.

试验环境如下：①硬件环境. CPU：Intel Xeon E5-2620 处理器，六核十二线程，主频 2. 00GHz，最大睿频 2. 60GHz；QPI 总线. GPU：NVIDIA Tesla K20C，参数详见表 1. ②软件环境. CentOS Linux release 7. 1 64 位操作系统，NVCC 编译器，CUDA 7. 5版本.

3.2　数据采集与分析

(1)第 1 次试验. 并行 MMAS 采用 pow 函数计

算状态转换规则(式(1)),部分结果见表2.

表1 Tesla K20C计算卡参数

Tab.1 Specifications of Tesla K20C

架构	Kepler
CUDA核心数	2 496
CUDA处理器频率	706 MHz
显存组态	4 800 MB
显存位宽	384.0 GB·s^{-1}
单精度浮点运算能力	3.52Tflops·s^{-1}
双精度浮点运算能力	1.17Tflops·s^{-1}

表2 采用pow函数计算MMAS状态转换规则的部分结果

Tab.2 Partial results of MMAS computing transition rules with pow function

数据集	蚂蚁数目	ACOTSP	异构平台1	加速比
eil51	64	427.80	427.76	0.20
kroA100	128	21 336.90	21 327.82	0.88
ch150	128	6 548.30	6 548.25	1.40
pr226	256	80 869.80	80 827.19	3.24
a280	256	2 609.20	2 613.23	4.15
lin318	512	42 346.60	42 348.63	8.34
pcb442	512	50 978.20	50 972.72	8.15
d493	512	35 372.50	35 376.20	8.16

(2)第2次试验.并行MMAS采用直接相乘方式计算状态转换规则(式(1)),部分结果见表3.

表3 采用直接相乘计算MMAS状态转换规则的部分结果

Tab.3 Partial results of MMAS computing transition rules by direct multiplying

数据集	蚂蚁数目	ACOTSP	异构平台2	加速比
eil51	64	427.80	427.76	0.83
kroA100	128	21 336.90	21 328.30	1.64
ch150	128	6 548.30	6 548.00	2.97
pr226	256	80 869.80	80 827.19	5.30
a280	256	2 609.20	2 613.23	5.47
lin318	512	42 346.60	42 347.75	13.53
pcb442	512	50 978.20	50 972.72	12.60
d493	512	35 372.50	35 376.20	12.82

依据2次试验结果作图如图2.结果表明,将MMAS算法耗时最多的路径建立过程及信息素矩阵更新过程在CPU-GPU异构平台上进行并发执行,在城市规模达到一定数量时,并行MMAS(pow)对串行算法体现出明显的性能提升,最大加速比可达8.16,且解的质量也与串行算法基本保持一致.对于设备端pow函数的优化使得并行MMAS(pow)在加速基础上获得了更为显著的性能提升.并行MMAS算法在不损失结果精度的前提下,获得了最高可达13.53倍的加速比,并且随着城市规模的增大加速比呈上升趋势.

由图3可知,随着城市规模的增大,tsp文件的读取时间并未明显增加,而数据传输与运算时间显著提升.以pcb442为例[24],图4所示的是tsp文件读取后数据在CPU和GPU之间传递和运算的时间分布,该分布表明并行MMAS将计算量最大的运算任务如路径建立阶段及信息素更新阶段在GPU上执行,充分发挥其多核多线程的计算优势,因而能给算法带来显著的性能提升.

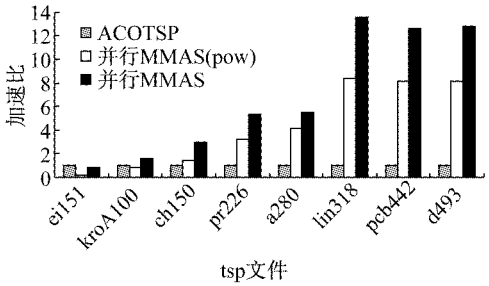

图2 并行MMAS算法加速趋势

Fig.2 Speedup trend of parallel MMAS

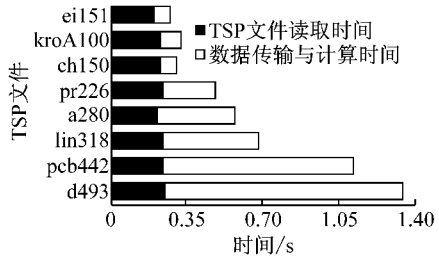

图3 单次运算数据读取时间与运算时间

Fig.3 Data access time and computing time in single run

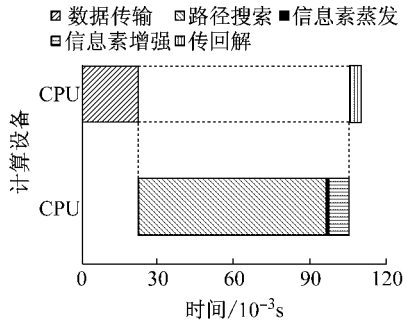

图4 CPU-GPU平台运算时间分布

Fig.4 Computing time distribution of CPU-GPU platform

4 结语

提出并行MMAS算法采用CPU作为主机、GPU作为设备的协同计算模式,CPU负责控制,GPU执行绝大部分的运算任务.在逻辑层面,算法使用并行蚁群策略,在运算量最大的路径建立阶段为每只蚂蚁分配CUDA线程.

详细分析了 MMAS 算法每个阶段的计算量和并发潜力，分别采用了 2 个对提高算法运行速度至关重要的策略：

(1)距离矩阵和信息素矩阵常驻设备内存，有效避免了每轮迭代结束后 2 个矩阵在主机内存和设备内存之间的传输，极大减少了数据延迟.

(2)用直接相乘方式代替 pow 函数计算状态转换规则，这一改变大大减少了计算量，非常有利于程序在 GPU 上运行速度的提升.

借鉴 CUDA 编程模型的数据流并行特点，对数据传输、内存层次、库函数调用实施针对性的优化，并行 MMAS 算法比 Delévacq 等采用的并行蚂蚁方案取得显著的性能提升. 该方式使得并行 MMAS 更加适合 GPU 的多线程架构，提供了细粒度条件下并行算法行之有效的优化策略，充分挖掘了单 GPU 异构计算模式的计算潜能. 同时建立了在 CPU-GPU 异构平台上并行 ACO 算法解决组合优化问题的模型，对其他蚁群算法在异构平台的性能提升也有启发意义. 由于不同蚁群算法的特点主要表现于信息素矩阵的更新过程，因此对于路径建立阶段存在并发潜力的蚂蚁系统如 AS(Ant System)，GPU 的加速效果是可复制的，仅需要针对不同 ACO 算法的信息素更新操作来制定相应的并发策略. 而 ACS(Ant Colony System)存在信息素矩阵局部更新操作，使不同蚂蚁的路径建立过程存在依赖性，所以很难对此过程进行并发.

参考文献：

[1] Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 1996, 26(1): 29.

[2] Stützle T, Hoos H H. Max-min ant system [J]. Future Generation Computer Systems, 2000, 16(8): 889.

[3] 王建玲，齐紫茜，何璐. 基于蚁群算法的车辆调度问题[J]. 交通科技与经济，2014，16(6):37.
WANG Jianling, QI Ziqian, HE Lu. Research on vehicle scheduling problem based on ant colony algorithm [J]. Technology & Economy in Areas of Communications, 2014, 16(6):37.

[4] 周明秀，程科，汪正霞. 动态路径规划中的改进蚁群算法[J]. 计算机科学，2013，40(1)：314.
ZHOU Mingxiu, CHENG Ke, WANG Zhengxia. Improved ant colony algorithm with planning of dynamic path[J]. Computer Science, 2013, 40(1): 314.

[5] 夏亚梅，程渤，陈俊亮，等. 基于改进蚁群算法的服务组合优化[J]. 计算机学报，2012，35(2):270.
XIA Yamei, CHENG Bo, CHEN Junliang, *et al*. Optimizing services composition based on improved ant colony algorithm [J]. Chinese Journal of Computers, 2012, 35(2):270.

[6] Khatri K, Gupta V K. A survey paper on solving TSP using ant colony optimization on GPU[J]. Compusoft, 2014, 3(12): 1354.

[7] Bullnheimer B, Kotsis G, Strauβ C. Parallelization strategies for the ant system[M]// High Performance Algorithms and Software in Nonlinear Optimization. [S. l.]: Springer US, 1998:1-8.

[8] Fu J. Parallel ant colony optimization algorithm with GPU-acceleration based on All-In-Roulette selection[J]. Computer & Digital Engineering, 2011, 84(10):260.

[9] Stützle T. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem[EB/OL]. [2015-12-01]. http://www.aco-metaheuristic.org/aco-code.

[10] Middendorf M, Reischle F, Schmeck H. Multi colony ant algorithms[J]. Journal of Heuristics, 2002, 8(3):305.

[11] 王诏远，王宏杰，邢焕来，等. 基于 Spark 的蚁群优化算法[J]. 计算机应用，2015，35(10)：2777.
WANG Zhaoyuan, WANG Hongjie, XING Huanlai, *et al*. Ant colony optimization algorithm based on Spark[J]. Computer Application, 2015, 35(10): 2777.

[12] Delévacq A, Delisle P, Gravel M, *et al*. Parallel ant colony optimization on graphics processing units [J]. Journal of Parallel & Distributed Computing, 2013, 73(1): 52.

[13] Wang Z Y, Tian-Rui L I, Xiu-Wen Y I. Approach for development of ant colony optimization based on mapReduce [J]. Computer Science, 2014, 41(7):261.

[14] Khatri K, Gupta V K. Research on solving travelling salesman problem using rank based ant system on GPU[J]. Compusoft, 2015, 4(5): 1778.

[15] Cecilia J M, García J M, Nisbet A, *et al*. Enhancing data parallelism for ant colony optimization on gpus[J]. Journal of Parallel & Distributed Computing, 2013, 73(1):42.

[16] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem[J]. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 53.

[17] Stutzle T, Hoos H. Max-min ant system and local search for the traveling salesman problem [C] // Evolutionary Computation. Indianapolis: IEEE, 1997: 309-314.

[18] Dawson L, Stewart I A. Accelerating ant colony optimization-based edge detection on the GPU using CUDA [C] // Evolutionary Computation. Beijing: IEEE, 2014:1736-1743.

[19] Garey M R, Johnson D S, Stockmeyer L. Some simplified NP-complete graph problems[J]. Theoretical Computer Science, 1976, 1(3): 237.

[20] Sanders J, Kandrot E. CUDA by example: An introduction to general-purpose GPU programming [M]. [S. l.]: Addison-Wesley Professional, 2011.

[21] Nvidia C. Nvidia Cuda C programming guide [J]. Nvdia Corporation, 2011, 120(18): 8.

[22] Graham S L, Kessler P B, Mckusick M K. Gprof: A call graph execution profiler[J]. ACM Sigplan Notices, 2004,39(4): 49.

[23] Amdahl G M. Validity of the single-processor approach to achieving large scale computing capabilities[C]// Spring Joint Computer Conference. Atlantic City: ACM, 1967: 483-485.

[24] Reinelt G. TSPLIB—A traveling salesman problem library[J]. ORSA Journal on Computing, 1991, 3(4): 376.