Journal of System Simulation

Volume 36 | Issue 8 Article 16

8-15-2024

GPU Parallel Acceleration Framework for Heuristic Optimization Algorithm

Dongjie Wang
School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China

Sixin Wen
School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China

Wanzhi Meng
School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China

Di Wu
School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China

GPU Parallel Acceleration Framework for Heuristic Optimization Algorithm

Abstract

Abstract: Heuristic optimization algorithm are a type of algorithm that uses large-scale populations for iterative calculations and are widely used to solve all kinds of complex optimization problems. However, such algorithm have the disadvantages of large calculation and long time consumption. To solve this problem, heuristic optimization algorithms are parallelized using GPU and compute unified device architecture (CUDA) to substantially improve computational efficiency. A GPU parallel framework for heuristic optimization algorithm is proposed, which designs an information interaction framework and algorithm parallel optimization strategy with a parallel logical structure, and solves the problem of the dissimilarity of the logical structure of information interaction in series and parallel, this framework can parallelize various heuristic optimization algorithms with generality and efficiency. In order to verify the effectiveness of this framework, five common heuristic optimization algorithms are parallelized by using the parallel framework, and the comparison results of GPU parallel computation and CPU serial computation under different multiple test functions are given. in which DE, HHO, GWO, and WOA reach the acceleration ratio of 179.1, 178.6, 74.3 and 358.2 times respectively when the population dimension is 5000, while ensuring the accuracy of the results, which verifies the high effectiveness and practicability of the designed parallel framework.

Keywords

heuristic optimization algorithm, GPU parallelism, CUDA model, parallel framework, information exchange

Recommended Citation

Wang Dongjie, Wen Sixin, Meng Wanzhi, et al. GPU Parallel Acceleration Framework for Heuristic Optimization Algorithm[J]. Journal of System Simulation, 2024, 36(8): 1929-1943.

第 36 卷第 8 期 系统仿真学报© Vol. 36 No. 8
2024 年 8 月 Journal of System Simulation Aug. 2024

启发式优化算法的GPU并行加速框架

王东杰，温思歆*，孟万植，吴迪
(大连理工大学 控制科学与工程学院，辽宁 大连 116024)

摘要： 为解决启发式优化算法计算量大、耗时长的缺点，使用图形处理单元(GPU)以及统一计算架构(compute unified device architecture，CUDA)对启发式优化算法进行并行化。*提出了一种针对启发式优化算法的GPU并行框架，设计了具有并行逻辑结构的信息交互框架、算法并行优化策略，解决了信息交互的逻辑结构在串、并行中的相异性问题，该框架可并行化各类启发式优化算法，具有一般性与高效性。*为验证该框架的有效性，利用并行框架对5种常见启发式优化算法进行并行化，给出了多个测试函数下GPU并行计算与CPU串行计算的对比结果，其中差分进化算法、哈里斯鹰优化算法、灰狼优化算法、鲸鱼优化算法在种群维度为5 000时，分别加速高达179.1、178.6、74.3、358.2倍，同时保证了结果的准确性，表明所设计并行框架的高效性与实用性。

关键词： 启发式优化算法；GPU并行；CUDA模型；并行框架；信息交互
中图分类号：TP391.9 文献标志码：A 文章编号：1004-731X(2024)08-1929-15
DOI: 10.16182/j.issn1004731x.joss.23-0780
引用格式: 王东杰, 温思歆, 孟万植, 等. 启发式优化算法的GPU并行加速框架[J]. 系统仿真学报, 2024, 36(8): 1929-1943.
Reference format: Wang Dongjie, Wen Sixin, Meng Wanzhi, et al. GPU Parallel Acceleration Framework for Heuristic Optimization Algorithm[J]. Journal of System Simulation, 2024, 36(8): 1929-1943.

GPU Parallel Acceleration Framework for Heuristic Optimization Algorithm

Wang Dongjie, *Wen Sixin**, *Meng Wanzhi*, *Wu Di*
(School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China)

Abstract: Heuristic optimization algorithm are a type of algorithm that uses large-scale populations for iterative calculations and are widely used to solve all kinds of complex optimization problems. However, such algorithm have the disadvantages of large calculation and long time consumption. To solve this problem, heuristic optimization algorithms are parallelized using GPU and compute unified device architecture (CUDA) to substantially improve computational efficiency. *A GPU parallel framework for heuristic optimization algorithm is proposed, which designs an information interaction framework and algorithm parallel optimization strategy with a parallel logical structure, and solves the problem of the dissimilarity of the logical structure of information interaction in series and parallel, this framework can parallelize various heuristic optimization algorithms with generality and efficiency.* In order to verify the effectiveness of this framework, five common heuristic optimization algorithms are parallelized by using the parallel framework, and the comparison results of GPU parallel computation and CPU serial computation under different multiple test functions are given. in which DE, HHO, GWO, and WOA reach the acceleration ratio of 179.1,

收稿日期：2023-06-28 修回日期：2023-10-08
基金项目：国家自然科学基金(61890920、61890921)
第一作者：王东杰(1999-)，男，硕士生，研究方向为航空发动机并行性能寻优。
通讯作者：温思歆(1994-)，男，博士生，研究方向为航空发动机控制与仿真。

178.6, 74.3 and 358.2 times respectively when the population dimension is 5000, while ensuring the accuracy of the results, which verifies the high effectiveness and practicability of the designed parallel framework.

Keywords: heuristic optimization algorithm; GPU parallelism; CUDA model; parallel framework; information exchange

0 引言

现代科学与工程存在许多非凸的复杂优化问题，需要采用适当的优化求解算法进行迭代求解。传统的数值优化算法，如牛顿法、梯度下降法、内点法等，通过严格的迭代计算仅得到局部最优解，并且需要利用导数信息，在全局收敛性、问题复杂性等方面受到极大的限制。相比之下，启发式优化算法具有简单、灵活、搜索能力强、全局收敛性好等优点，能有效解决非凸的复杂优化问题，在各个领域得到广泛的应用，如在复杂机电系统[1]、流程工业[2]、城市污水处理[3]、航空航天[4]等领域。

启发式优化算法通常来源于自然生物行为的启发，如模拟鸟群觅食行为的粒子群优化算法[5]、模拟种群进化现象的差分进化算法[6]、模拟蚁群觅食行为的蚁群优化算法[7]等。近年来，许多高质量的启发式优化算法被相继提出，如文献[8]提出的灰狼优化算法模拟了灰狼的狩猎过程；文献[9]提出的飞蛾焰火算法(MFO)，灵感来源于飞蛾的横向定位导航机制；文献[10]提出的鲸鱼优化算法(WOA)模拟了座头鲸的社会行为；文献[11]提出的哈里斯鹰优化算法(HHO)，灵感来源于哈里斯鹰的合作行为以及追逐行为；文献[12]提出的粘液霉菌算法模拟了黏菌多头绒泡菌觅食过程中的行为和形态变化；文献[13]提出的火烈鸟搜索算法，灵感来源于火烈鸟的迁徙和觅食行为；文献[14]提出的人工蜂鸟算法模拟了自然界中蜂鸟的特殊飞行技能和智能觅食策略。

这些启发式优化算法虽然可有效求解复杂优化问题，但它们依赖于大规模种群的迭代计算，计算量较大，且随着问题规模的增加呈指数级增长[15]。针对这类密集型数据的计算，采用提高CPU主频的方式因功耗的限制而逐渐陷入瓶颈[16-17]。启发式优化算法具备天然的可并行性，因此，通过GPU并行计算是提高计算效率性能行之有效的方法[18]，受到广泛的关注，如文献[19]提出了并行遗传算法，使用了Quadro FX 580显卡，在种群规模为64的情况下达到了8.5倍的加速比，提高了公路线性优化效率；文献[20]使用并行粒子群优化算法提高稀疏重构概率，使用RTX 1650Ti显卡，在种群规模为4 096的情况下最快加速达44倍；文献[21]实现了并行加速差分进化算法，使用GTX 285显卡，在种群规模为100以及1 000情况下分别达到9.9~19.0倍、20.2~35.4倍的加速比；文献[22]实现并行化非确定花授粉算法，使用GTX A6000显卡，在1 024个路径规划解决方案下达到最高253.4倍的加速比，显著提高了无人机在复杂三维环境中最优轨迹的求解速率；文献[23]提出并行化的灰狼优化算法，使用了Quadro 4000 GPU显卡，在平均407个序列下，得到6.7倍的加速比，加快了生物分子序列分析的速度。此外，并行优化计算还在航天领域得到应用[24]。然而，各种启发式优化算法层出不穷，却没有统一的GPU并行加速框架，阻碍了它们的工程应用[25]。鉴于启发式优化算法均具备相似的计算机制与拓扑结构[26]，本文建立了统一的启发式优化算法并行框架。

1 基础知识

在GPU计算领域，CUDA(compute unified device architecture)编程模型具有高度灵活性与卓

DOI: 10.16182/j.issn1004731x.joss.23-0780

越的性能，是应用广泛的技术框架，如在卷积神经网络[27]与数据挖掘[28]等领域，因此，本文选用它进行启发式优化算法的并行框架设计。

1.1 CUDA编程模型

在CUDA编程模型中，GPU作为协处理器提供大量计算资源，实现并行线程，从而隐藏数据存储延迟。核函数是用于执行GPU计算的核心单元，它被CPU调用，使用限定符__global__进行修饰。仅由GPU进行调用计算的函数被称为设备函数，用__device__进行修饰。CUDA定义了一种抽象的层次概念以便进行内存分配与管理，该基本结构如图1所示，其中，由内核启动的线程统称为网格(Grid)，网格由多个线程块(Block)构成，而一个Block又由一组线程(Thread)构成，核函数运行时通过Block索引(BlockIdx)与Thread索引(ThreadIdx)辨别每一个计算任务。CUDA编程模型提供了多种可编程的内存类型[29]，包括寄存器、本地内存、全局内存、共享内存、常量内存、纹理内存，可根据需要进行分配、传输与释放。

1.2 传统串行的启发式优化算法

启发式优化算法本质上要解决一个带约束的组合优化问题，以找到一组达到最优性度量的可行解：

$$\begin{aligned}&\min y=f(\boldsymbol{x})\\&g_i(\boldsymbol{x})\leqslant 0\ ,\ i=1,\ 2\ ,\cdots,m\ ,\\&h_j(\boldsymbol{x})=0\ ,\ j=1,\ 2\ ,\cdots,k\ ,\end{aligned}\tag{1}$$

式中：$\boldsymbol{x}=[x_1,x_2,\cdots,x_D]$。

该带约束的优化问题通过拉格朗日乘子法、惩罚函数法等转换成无约束优化问题，作为启发式优化算法的适应度函数，然后迭代求解全局最优解。

传统串行的启发式优化算法流程如图2所示，在初始化完成后，迭代过程种群严格遵循适应度计算、信息交互、种群更新等步骤。

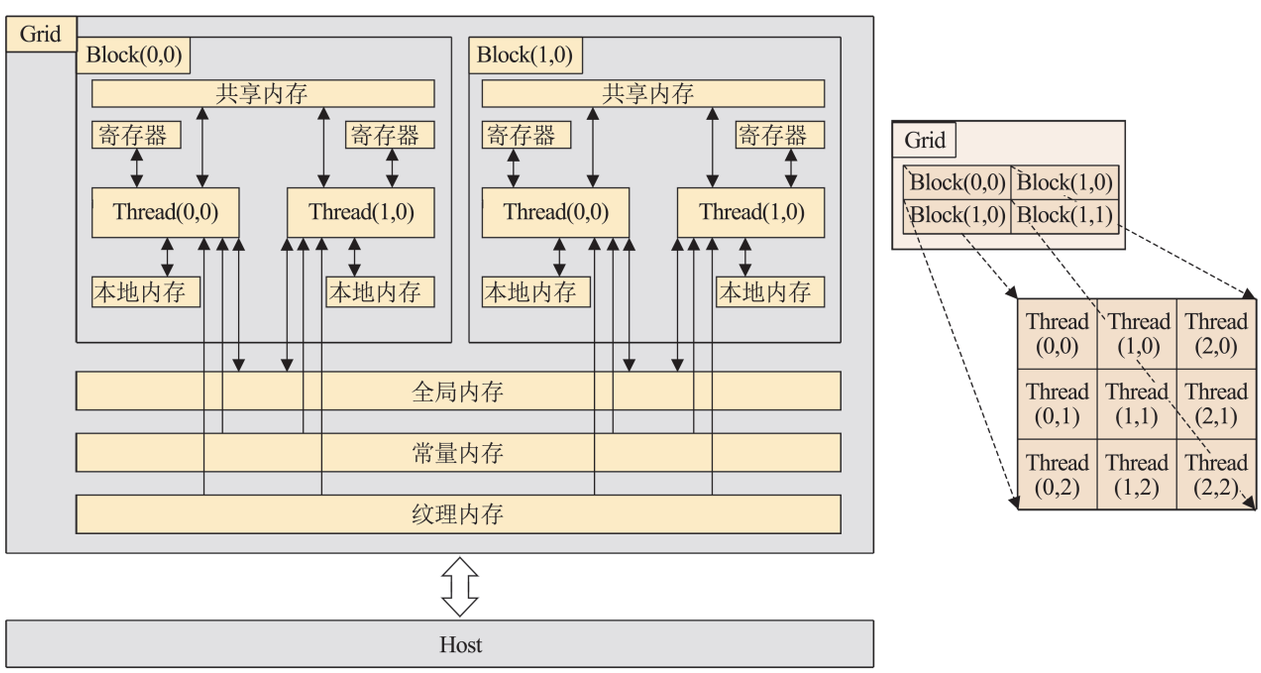

图1 CUDA编程模型
Fig. 1 CUDA programming model

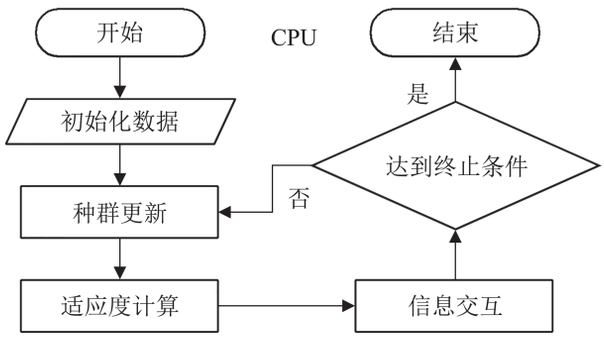

图2　串行启发式优化算法流程
Fig. 2　Flow of serial heuristic optimization algorithm

2　GPU 并行框架设计

启发式优化算法因各个种群任务的相似性，具备天然的可并行性。为最大程度地提高算法的执行效率，本文提出图3所示的GPU并行框架，其中迭代算法全权交由GPU并行执行。值得注意的是，细粒度的并行方式虽然提高了并行度，但伴随大量的数据交互与同步，将耗费一定的时间，因此，本文并行框架是在以粗粒度的并行方式为基础所设计的。

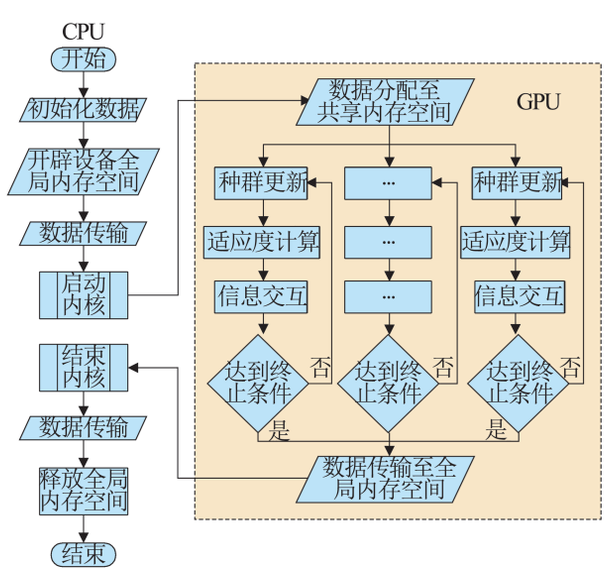

图3　启发式优化算法GPU编程模型
Fig. 3　Heuristic optimization algorithm GPU programming model

2.1　并行策略与框架

启发式优化算法在迭代过程中存在种群之间的数据关联性，尤其是具有强交互性的信息更新与遵循严格时序的迭代算法均要求保持其任务的前后一致性，如何处理信息交互行为成为算法高效并行的限制性因素。一种解决办法是采用分步并行，即将启发式优化算法按需要同步的节点拆分开，分别建立核函数，独立并行每个部分，在核函数之间施加同步指令 cudaDeviceSynchronize，以等待所有核函数完成任务。此时，由于CUDA的默认流是阻塞流，在未使用多流、不施加同步函数的情况下能起到隐式同步。但每一次迭代都需要重启一次核函数，如此频繁地启动内核将耗费大量的时间[25]。此外，核函数之间存在一定程度的信息重叠，需要频繁进行GPU与CPU之间的通讯，耗费大量时间，导致并行效益显著降低。因此，采用分步并行的解决方案并行化处理启发式优化算法，得到的时间效益还有待进一步提升。

为解决上述问题，本文提出了一种针对启发式优化算法的高效GPU并行框架，如图4所示。该框架主要针对种群间数据依赖程度较低的启发式优化算法，通过降低种群之间数据依赖度，在不影响迭代收敛情况下减少信息的交互行为，实现并行加速。根据信息交互对适应度收敛影响的强弱控制交互范围，对收敛效益影响较弱的行为从全局的更新迭代收缩至以Block为单位的小群体中。对收敛性影响密切的交互行为，选择全局内存作为通讯媒介建立共享空间，全局内存为每个Block开辟映射空间承担信息共享的载体，从而减少Block间存储的冲突访问，同时施加内存栅栏避免内存访问重排序。

信息的传递机制分成两部分：①将自身群体信息共享；②从共享空间获取信息。由于在特定时刻子种群执行相对独立的行为，这种信息交互策略的可靠性能够得到保证。在信息交互过程中，由于各Block在核函数中是异步进行的，子种群所处迭代步数不尽相同，但通过控制任务量可以使这种不同步限定在一定范围内，且扩大迭代的规

DOI: 10.16182/j.issn1004731x.joss.23-0780

模使这种偏差对收敛性影响较小。从时序上看，各Block在每次迭代都会在信息交互空间执行一次内存读写操作，使用内存栅栏对内存访问顺序保证，以及规模化迭代对数据传递保证，为这种信息传递机制提供了可靠性。

在本文提出的并行框架基础上，根据种群数量合理设置Block大小，可以有效提高并行效益。在弱信息交互下，增加Block数量，可增强信息通讯的容错率、提高GPU资源的利用率。在强信息交互下，Block数量过少会导致流式多处理器(stream multi-processor，SM)资源的浪费，反之则增加了复杂的信息通讯过程进而形成时间减益，因此，需要寻找一个权衡点。通常在Block数量比CUDA核心SP(streaming processor)数量大时，设置其值接近SM的整数倍，能充分发挥其并行加速效益。

针对数据间依赖程度较高的启发式优化算法，CUDA同步无法满足时序的一致性，该问题是维度扩张的限制性因素。本框架用图4的单个Block表示，__synThreads实现所有Thread的同步，并使用共享内存替代全局内存作为信息通讯的媒介。一个Block只能支持最大1 024个Thread，是此类算法维度的上限。此外，一个Block只能分配给一个SM执行，造成GPU硬件资源的浪费。

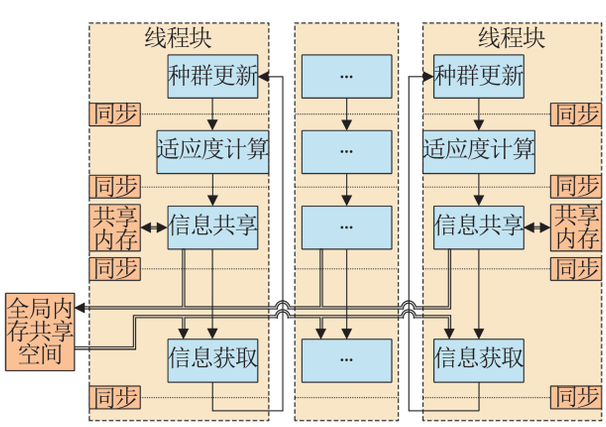

图4　启发式优化算法并行框架

Fig. 4　Parallel framework for heuristic optimization algorithm

2.2 算法信息交互

种群个体在更新过程中将进行信息交互，为保证算法时序的正确性，建立合理的信息交互逻辑是必要的。针对这一难点，提出了如下的信息交互并行框架。

极值：最优解是启发式优化算法中最常见的信息因素，求解过程需要获取每个个体信息的访问权限。并行思想为分群寻优并配合全局内存独立通讯，每个Block在各自内部寻优，将优于历史极值的可行解传递至信息共享空间。如此，共享空间中各个Block极值将被逐步更替，在需要获得全局最优解时，每个Block会遍历所有共享信息并获取优于自身的值，独立迭代的每个子群体能够在几次迭代内将全种群最优值散播到算法更新路径中。在Block的寻优中运用了规约算法[30]，Thread进行循环比较，并将较优值置入标号较低的内存单元内，活跃Thread在每次迭代都会减少一半。种群信息、极值信息会被存储在Block的共享内存内，由于规约过程会破坏原有信息，因此，需要创造一个种群信息共享内存副本。综上所述，并行启发式算法的极值计算伪代码如算法1所示。

算法1：并行启发式优化算法的极值信息交互框架伪代码

设置规约步长s为当前Block中Thread数量的一半

while $s > 0$ do

if Thread i的适应度值大于Thread $i+s$的适应度值 then

交换两个Thread的位置信息、适应度值

end if

将步长s右移一位

end while

if Thread 0的适应度值小于全局内存中对应该Block的适应度值 then

Thread 0的位置信息与适应度值传递

到全局内存中对应的Block中

end if

获取最大Block数量M_b

for k=1 to M_b do

if 当前Block的最优适应度值大于全局内存中第k个Block的最优适应度值 then

全局内存中第k个Block的位置信息与适应度值传递到当前Block

end if

end for

均值：启发式优化算法引入均值，用于在总体状态基础上加强局部搜索，均值对算法向全局最优靠拢的影响较小。本文给出均值的并行思想主要为分群求解，代替全局通讯。Block中均值的求解同样运用规约，先实现两两相加求和，然后将总和与每个Block的活跃Thread数进行比值。求解过程需要在共享内存中为每个种群建立数据副本。并行启发式算法的均值计算伪代码如算法2所示。

算法2：并行启发式优化算法均值的信息交互框架伪代码

设置规约步长s为当前Block中Thread数量的一半

while s > 0 do

将Thread i+s的位置信息相加至Thread i

步长s右移一位

end while

每个Block的均值为当前Block中Thread 0的数值除以当前活跃Thread数量

种群混合：启发式优化算法为提高搜索能力，会使随机种群个体之间进行数据的混合，这要求个体都具有互相访问权限。本文的策略是为种群信息开辟全局内存的副本以扩宽其作用域，当共享内存中数据更替时，副本中的信息也得到同步更新，保证种群混合能够实时、准确地执行。并行启发式算法的种群混合伪代码如算法3所示。

算法3：并行启发式优化算法种群混合的信息交互框架伪代码

算法的最大迭代次数被设置为T

for t =1 to T do

种群信息更新；

将种群信息从共享内存中传递至全局内存中;

种群迭代

获取全局内存中的种群信息

种群信息混合

end for

排序：排序是一种强交互性信息交互过程，线程同步对并行的排序过程起关键作用，排序结果会因个体迭代次序的错步而出现错误。全局的Threads同步需要以多个内核为分界点，频繁的内核启动以及内存传输将耗时过多，因此，本文的排序操作在单一Block中使用共享内存与__synThreads实现。常见的排序算法如选择、插入等由于对排序元素之间的操作较频繁，导致并行性较低。本文采用了一种简单、并行性高的排序算法，即奇偶交换排序。如图5所示，相邻内存的两组数据按序比较，并按优劣性交换次序，每轮迭代的奇偶顺序交替进行。排序打乱了原有的随机次序结构，同样需要为每个种群建立共享内存数据副本。并行启发式算法的排序伪代码如算法4所示。

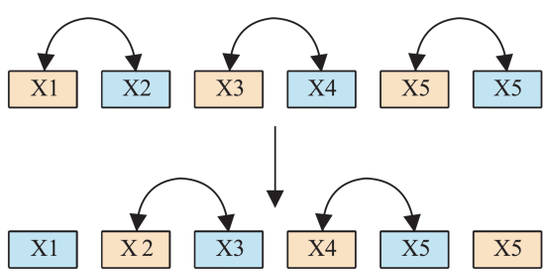

图5 奇偶交换排序结构

Fig. 5 Parity swap sort structure

算法4：并行启发式优化算法排序的信息交互框架伪代码

待排序的种群数量为M_n

https://dc-china-simulation.researchcommons.org/journal/vol36/iss8/16
DOI: 10.16182/j.issn1004731x.joss.23-0780

for j=1 to M_n do
if j 是偶数 then
if Thread 2i+1 的适应度值小于 Thread 2i 的适应度值 then
交换 Thread 2i+1 与 Thread 2i 的位置信息与适应度
end if
end if
if j 是奇数 then
if Thread 2i+2 的适应度值小于 Thread 2i+1 的适应度值 then
交换 Thread 2i+2 与 Thread 2i+1 的位置信息与适应度
end if
end if
end for

多极值：部分启发式优化算法存在多个极值参与迭代的情况，本文给出的并行思想是分群排序配合全局内存独立通讯，是对上文排序和极值的结合。通过群内排序在每个Block内提取多个极值，全局内存传输使得所有子群获取的极值互通，这些子群数据都会留存在全局内存中，并在每个群体中比较，得到最终极值，该信息通讯方式与求极值框架相同。

2.3 优化策略

CUDA并行过程中存在大量耗时因素[25]，本文给出了6种通过合理的规划带来大量性能优化的方式。

(1) 主机与设备间的数据传输以及内核启动耗时：初始化阶段将所有数据打包并传输至GPU端，尽可能只启动一次内核处理所有迭代任务，在算法并行前、后只需实行一次数据内存拷贝cudaMemcpy，最大程度上缩减了该过程的耗时。

(2) 内存读写的带宽以及延迟：CUDA常用内存按访问速度从快到慢的排序分别为寄存器、共享内存以及全局内存。寄存器内存空间与作用范围小，适合将算法中需要使用到的少量高频变量存储到寄存器，溢出的数据会存储到本地内存，在计算能力2.0之后本地内存数据被存储到片上的缓存内，相较于板载的全局内存有更低的延迟。共享内存相比于全局内存更接近于SM，有更低的延迟以及更高的带宽，启动内核后将部分种群数据从全局内存中传输至共享内存中迭代更新，能显著提升性能。对于作用域为全局的数据或是作为跨群通讯的媒介，则使用全局内存存储。此外，算法中提前给定的值通常会设置为常量内存。

(3) 内存的访问模式：全局内存的访问都会通过一级缓存或者二级缓存，分别由32个字节以及128个字节的内存事务实现。如果一个warp访问不连续的内存块，会出现非对齐的现象而导致带宽的浪费。共享内存被分为32个同样大小的内存模型，对应一个warp，共享内存的地址相应映射在不同存储体，如果warp访问同一个存储体时会发生存储体冲突，形成带宽浪费。因此，采用全局内存的对齐访问模式、共享内存非冲突访问模式，以有效提高通讯效率。

(4) 线程束分化：warp在执行语句过程面临分支时，未执行当前指令的Thread需要等待其他Thread完成指令。本文通过减少分支语句、利用循环展开，有效减少这种分行阻塞，提高并行效率。

(5) 延迟隐藏：指令延迟分为算术指令延迟以及内存指令的延迟，算术指令延迟是指空闲计算单元等待某些计算单元工作的过程，内存指令延迟是指计算单元对数据传输的等待。此时线程调度器能够调度其他warp执行工作，保证计算资源被充分利用。因此，本文为每个SM提供足量的Thread，为延迟隐藏提供必要条件，大幅提升线程束占有率。

(6) 线程活跃性：warp不会在不同Block之间分离，如果Block大小不是warp的整数倍，末位warp中会有Thread不活跃。因此，启动内核的Block大小通常设计为32的倍数，最大程度地提高资源利用率。

3 并行框架应用

本章简要介绍5种启发式优化算法的基本原理，并应用前文给出的并行框架对这5种算法进行并行化。启发式优化算法随机初始化n个种群，本文用向量$\boldsymbol{x}_i$表示初始解，每个个体由n_j个参数构成，即$\boldsymbol{x}_i=[x_{i,1}, x_{i,2}, \cdots, x_{i,n_j}]^{\mathrm{T}}$，$i=1,2,\cdots,n$。

3.1 差分进化算法

DE拥有全局收敛性好、收敛速度快、控制变量少、鲁棒性强的特点[6]，主要由变异、交叉、选择3个部分构成，算法种群之间数据依赖性较弱，因此，采用开辟多个Block的方式来加速并行。

(1) 变异

种群通过变异生成新的向量：

$$\boldsymbol{v}_i=\boldsymbol{x}_{r_1}+F\cdot(\boldsymbol{x}_{r_2}-\boldsymbol{x}_{r_3}) \tag{2}$$

式中：r_1、r_2、r_3为3个互不相同的种群索引；F为变异因子，它放大两个参与变异个体的差异，以避免搜索停滞，取值范围在[0.4, 1]之间。

变异过程需要对随机索引个体信息进行融合，在并行过程采用种群混合的信息交互并行框架，并创建共享内存中的种群信息的副本，为后续选择铺垫。

(2) 交叉

变异后的新种群与原种群进行交叉：

$$u_{i,j}=\begin{cases} v_{i,j}, & q<c_{\mathrm{r}} \\ x_{i,j}, & q\geqslant c_{\mathrm{r}} \end{cases} \tag{3}$$

式中：q为[0, 1]内的随机数；c_{r}为交叉率。q小于c_{r}则接受变异，大于等于c_{r}则保留原值，c_{r}取值在[0.2, 1]之间，交叉过程的个体分配给Thread独立并行。

(3) 选择

当前种群与原种群进行适应度比较，并择优选取。因此在完成了当轮迭代更新后，将共享内存信息与全局内存同步，保留最优值。

3.2 灰狼优化算法

GWO有较强的搜索能力与较好的全局收敛性[8]，算法定义3个极值点为3只头狼，分别用α、β、δ表示。捕猎过程主要有2个步骤，分别为追踪包围与攻击。GWO种群之间数据依赖性较弱，因此通过开辟多个Block来加速计算。

(1) 追踪包围

灰狼根据猎物位置调整自身并包围猎物，建立环绕行为模型：

$$\begin{cases} \boldsymbol{D}=\left|\boldsymbol{C}\cdot\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)\right| \\ \boldsymbol{x}_i(t+1)=\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{A}\cdot\boldsymbol{D} \end{cases} \tag{4}$$

式中：$\boldsymbol{x}_{\mathrm{b}}$为猎物的位置向量；$t$为当前迭代次数；$\boldsymbol{A}$与$\boldsymbol{C}$分别为$1\times n_j$维的系数向量。

$$\begin{cases} \boldsymbol{A}_j=2a\cdot\mathrm{rand}(0,\ 1)-a \\ \boldsymbol{C}_j=2\cdot\mathrm{rand}(0,\ 1) \end{cases} \tag{5}$$

式中：a的值随迭代次数的递增，从2线性递减至0。

头狼α、β、δ的位置被假定为最靠近猎物的位置，狼群通过环绕行为向头狼靠拢：

$$\begin{cases} \boldsymbol{D}_\alpha=\left|\boldsymbol{C}_\alpha\cdot\boldsymbol{x}_\alpha-\boldsymbol{x}_i(t)\right| \\ \boldsymbol{x}_{i,\alpha}(t+1)=\boldsymbol{x}_\alpha-\boldsymbol{A}_\alpha\cdot\boldsymbol{D}_\alpha \\ \boldsymbol{D}_\beta=\left|\boldsymbol{C}_\beta\cdot\boldsymbol{x}_\beta-\boldsymbol{x}_i(t)\right| \\ \boldsymbol{x}_{i,\beta}(t+1)=\boldsymbol{x}_\beta-\boldsymbol{A}_\beta\cdot\boldsymbol{D}_\beta \\ \boldsymbol{D}_\delta=\left|\boldsymbol{C}_\delta\cdot\boldsymbol{x}_\delta-\boldsymbol{x}_i(t)\right| \\ \boldsymbol{x}_{i,\delta}(t+1)=\boldsymbol{x}_\delta-\boldsymbol{A}_\delta\cdot\boldsymbol{D}_\delta \end{cases} \tag{6}$$

$$\boldsymbol{x}_i(t+1)=\frac{\boldsymbol{x}_{i,\alpha}(t+1)+\boldsymbol{x}_{i,\beta}(t+1)+\boldsymbol{x}_{i,\delta}(t+1)}{3} \tag{7}$$

包围猎物过程需利用3个极值点的信息，本文运用求多极值的信息交互并行框架，在子群体获取多极值信息时，种群个体并行执行位置更新算法。

(2) 攻击

通过递减$|\boldsymbol{A}|$来模拟狼群行为。$|\boldsymbol{A}|>1$时加大搜索范围，利于跳出局部最优，$|\boldsymbol{A}|$趋于0时表明猎物已被包围锁定，此时从α中获取全局最优解。

迭代步数控制$|\boldsymbol{A}|$的递减变化，并行过程个体发生攻击的时刻被分散在不同时刻，为每个个体设置相同的寄存器变量以提高算法并行度。

DOI: 10.16182/j.issn1004731x.joss.23-0780

3.3 飞蛾焰火算法

MFO具有一定跳出局部搜索并减小搜索停滞的能力[9]，飞蛾制导主要有3个流程，分别为选择火焰、绕焰飞行、更新火焰。MFO种群之间有较强的数据依赖，并行过程使用了单个Block。

(1) 选择火焰

飞蛾会选择顺序对应火焰，火焰数量为

$$m=\mathrm{round}\left(n-\frac{t}{T}(n-1)\right) \tag{8}$$

式中：T为最大迭代次数。超出火焰数量的飞蛾将会选择排序最末的火焰进行绕焰飞行。

火焰选择通过对应的线程序号配对，同时存储最末的火焰信息提供给多余的飞蛾。

(2) 绕焰飞行

飞蛾绕焰做螺旋式飞行，相应的位置更新公式为

$$\boldsymbol{x}_i(t+1)=\boldsymbol{d}\cdot \mathrm{e}^{b\cdot c}\cdot\cos(2\pi c)+\boldsymbol{x}_i(t) \tag{9}$$

式中：$\boldsymbol{d}$为飞蛾到火焰的距离；b为定义对数螺旋形状的常数；$c\in[-1, 1]$为均匀随机数。火焰与飞蛾信息已被获取，种群个体并行实现更新。

(3) 更新火焰

飞蛾绕焰飞行后会到达新的位置，移动后的飞蛾与m个火焰按适应度排序，适应度最优的m个位置被视作火焰位置。

更新火焰运用排序的信息交互并行框架实现。待排序的数据为当前飞蛾与火焰，因此，需开辟两倍种群数量的信息拷贝空间，排序中奇偶交换排序的循环次数与待排序数据量同步递减。

3.4 鲸鱼优化算法

WOA搜索能力强、全局收敛性好[10]，算法包含包围猎物、随机搜寻以及螺旋泡网捕食3个策略。WOA种群之间数据依赖性较弱，开辟多个Block进行并行加速。

(1) 包围猎物

座头鲸识别猎物并形成包围，最优的座头鲸位置被视为猎物位置，这种环绕行为与GWO在追踪包围时刻相同。

包围猎物与螺旋泡网捕食需最优值信息，因此，位置更新开始前首先运用求极值的信息交互并行框架。

(2) 随机搜寻

通过$|\boldsymbol{A}|$决定座头鲸进行包围猎物还是随机搜寻，当$|\boldsymbol{A}|\geqslant 1$时进行随机搜寻，当$|\boldsymbol{A}|<1$时进行包围猎物，随机搜寻的公式为

$$\begin{cases}\boldsymbol{D}=\left|\boldsymbol{C}\cdot\boldsymbol{x}_{\mathrm{r}}-\boldsymbol{x}_i(t)\right|\\ \boldsymbol{x}_i(t+1)=\boldsymbol{x}_{\mathrm{r}}-\boldsymbol{A}\cdot\boldsymbol{D}\end{cases} \tag{10}$$

式中：$\boldsymbol{x}_{\mathrm{r}}$为当前种群中随机座头鲸的位置向量。

随机搜寻过程需运用到种群混合的信息交互并行框架。

(3) 螺旋泡网捕食

座头鲸制造螺旋型气泡来驱赶猎物并调整自身位置。包围猎物、随机搜索与进行螺旋泡网捕食的行为概率相同，该行为的公式为

$$\boldsymbol{x}_i(t+1)=\left|\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)\right|\cdot e^{b\cdot c}\cdot\cos(2\pi c)+\boldsymbol{x}_{\mathrm{b}} \tag{11}$$

极值信息已传递、螺旋泡网捕食在个体间高效并行。个体信息在更新结束后，将种群信息同步至全局内存，以保证随机搜寻过程具有实时性。

3.5 哈里斯鹰优化算法

HHO收敛速度快、搜索能力强，有能快速跳出局部最优的潜力[11]。算法包含了搜索阶段、过渡阶段以及开发阶段。HHO种群之间数据依赖性较弱，因此，为其开辟多个Block进行并行加速。

(1) 搜索阶段

哈里斯鹰探查并发现猎物，分两种策略：

$$\boldsymbol{x}_i(t+1)=\begin{cases}\boldsymbol{x}_{\mathrm{r}}-\boldsymbol{q}_1\cdot\left|\boldsymbol{x}_{\mathrm{r}}-2\times\boldsymbol{q}_2\cdot\boldsymbol{x}_i(t)\right|, & q\geqslant 0.5\\ (\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_{\mathrm{m}}(t))-\boldsymbol{q}_1\cdot(\boldsymbol{x}_l+\boldsymbol{q}_2\cdot(\boldsymbol{x}_{\mathrm{u}}-\boldsymbol{x}_1)), & q<0.5\end{cases} \tag{12}$$

式中：$\boldsymbol{x}_{\mathrm{m}}$为老鹰位置的平均值；$\boldsymbol{q}_1$与$\boldsymbol{q}_2$为$1\times n_j$维的随机向量。第一条策略是结合随机个体形成位移，第二条策略为在猎物与追逐者平均位置的差异基础上添加随机缩放分量，生成新的搜索位置。

搜索阶段在并行过程分别运用均值、极值以及种群混合的信息交互并行框架，均值与极值求取均在更新开始前完成，种群混合则是在搜索阶段实现。搜索阶段的信息交互使得后续更新行为仅限于Thread自身，充分提高程序并行度。

(2) 过渡阶段

哈里斯鹰根据猎物的逃逸能量转换行为，猎物的能量模型定义为

$$E=2E_0(1-t/T) \tag{13}$$

式中：E_0为猎物的初始能量状态，每次迭代在[−1, 1]区间内随机变化；$|E|\geqslant 1$时进入搜索阶段，$|E|<1$时进入开发阶段。

种群个体使用寄存器存储能量状态，实现独立的模态转换过程。

(3) 开发阶段

根据猎物的逃跑行为，哈里斯鹰模拟了4种突袭策略。设定r为逃脱率，是[0, 1]内的随机数，$r<0.5$对应逃脱成功，反之则逃脱失败。

软围攻：$r\geqslant 0.5$并且$|E|\geqslant 0.5$时，哈里斯鹰包围猎物并实现猛扑，行为描述为

$$\boldsymbol{x}_i(t+1)=\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)-E\cdot\left|J\cdot\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)\right| \tag{14}$$

$$J=2\times(1-\mathrm{rand}\,(0,\ 1)) \tag{15}$$

式中：J为猎物的随机跳跃强度。

硬围攻：$r\geqslant 0.5$并且$|E|<0.5$时，哈里斯鹰在猎物力竭时展开突袭，行为描述为

$$\boldsymbol{x}_i(t+1)=\boldsymbol{x}_{\mathrm{b}}-E\cdot\left|\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)\right| \tag{16}$$

渐进快速俯冲软围攻：$r<0.5$且$|E|\geqslant 0.5$时，猎物能量充沛并给出欺骗性动作，哈里斯鹰进行几次快速俯冲以及时纠正位置，行为描述为

$$\boldsymbol{x}_i(t+1)=\begin{cases}\boldsymbol{Y}, & f(\boldsymbol{Y})<f(\boldsymbol{x}_i(t))\\ \boldsymbol{Z}, & f(\boldsymbol{Z})<f(\boldsymbol{x}_i(t))\end{cases} \tag{17}$$

$$\boldsymbol{Y}=\boldsymbol{x}_{\mathrm{b}}-E\cdot\left|J\cdot\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_i(t)\right| \tag{18}$$

若$\boldsymbol{Y}$未达成优化，将计算$\boldsymbol{Z}$：

$$\boldsymbol{Z}=\boldsymbol{Y}+\boldsymbol{S}\cdot\boldsymbol{L} \tag{19}$$

式中：$f(\cdot)$为种群的适应度计算函数；$\boldsymbol{S}$为在[0, 1]内$1\times n_j$维的随机向量，$\boldsymbol{L}$计算公式如下：

$$\boldsymbol{L}=0.01\times\frac{\sigma\cdot\boldsymbol{g}}{|\boldsymbol{h}|^{\frac{1}{\lambda}}} \tag{20}$$

式中：$\boldsymbol{g}$、$\boldsymbol{h}$为[0, 1]内的随机向量；λ为默认常数，通常取1.5；σ的计算公式如下：

$$\sigma=\left(\frac{\Gamma(1+\lambda)\cdot\sin\left(\frac{\pi\lambda}{2}\right)}{\Gamma\left(\frac{1+\lambda}{2}\right)\cdot\lambda\cdot 2^{\frac{\lambda-1}{2}}}\right)^{\frac{1}{\lambda}} \tag{21}$$

渐进快速俯冲硬围攻：$r<0.5$且$|E|<0.5$时，猎物没有足够能量逃跑，哈里斯鹰维持俯冲策略，并缩减种群位置与猎物的距离，行为描述为

$$\boldsymbol{x}_i(t+1)=\begin{cases}\boldsymbol{Y}, & f(\boldsymbol{Y})<f(\boldsymbol{x}_i(t))\\ \boldsymbol{Z}, & f(\boldsymbol{Z})<f(\boldsymbol{x}_i(t))\end{cases} \tag{22}$$

$$\boldsymbol{Y}=\boldsymbol{x}_{\mathrm{b}}-E\cdot\left|J\cdot\boldsymbol{x}_{\mathrm{b}}-\boldsymbol{x}_{\mathrm{m}}(t)\right| \tag{23}$$

$\boldsymbol{Z}$的计算与渐进快速俯冲软围攻相同。

开发阶段随机地执行一种突袭策略，涉及的极值与均值在搜索阶段前已传递至每个种群个体，因此，开发阶段可实现个体独立更新，更新结束后将种群信息同步至全局内存保证数据融合的进行。

4 并行实验与结果分析

4.1 实验设备

本节测试所用的GPU设备为Turing架构的NVIDIA GeForce MX450显卡，具有2 GB显存容量，最大时钟频率为1.58 GHz，具有14个SM，共896个CUDA核心，每个SM的共享内存限额为64 KB。在软件版本方面，安装的CUDA版本为11.4，计算能力为7.5。CPU是Intel i5-1135G7，时钟频率为2.42 GHz，具有16 GB 3200 MHz主存。本文实验的串行程序是在CPU上通过C代码实现，并行程序在GPU上通过C-CUDA代码实现。

4.2 测试函数

实验给出6个测试函数，分别用f_1~f_6表示，其中，$\boldsymbol{x}$为输入向量。测试函数f_1~f_5[21]在表1中展示，函数的输入数值限幅为[−100, 100]。

DOI: 10.16182/j.issn1004731x.joss.23-0780

表1 具有许多局部极小值的测试函数
Table 1 Test function with many local minima

Number	Functions	$f_{\min}$
f_1	$f_1(\boldsymbol{x})=-\sum_{j=1}^{n_j}x_j\sin\left(\sqrt{x_j}\right)$	0
f_2	$f_2(\boldsymbol{x})=\sum_{j=1}^{n_j}\{x_j^2-10\times\cos(2\pi x_j)+10\}$	0
f_3	$f_3(\boldsymbol{x})=1+f_a\cdot f_a/4\,000-\cos(f_a)+\sum_{j=1}^{n_j-1}(1+f_{b_j}\cdot f_{b_j}/4\,000-\cos(f_{b_j}))$ $f_a=100\times((x_{n_j}+1)^2-(x_0+1))^2+(1-(x_{n_j}+1))^2$ $f_{b_j}=100\times((x_j+1)^2-(x_{j+1}+1))^2+(1-(x_j+1))^2,\ j=1,2,\cdots,n_j-1$	0
f_4	$f_4(\boldsymbol{x})=\frac{\pi}{n}\left\{10\times\sin^2(\pi y_1)+\sum_{j=1}^{n_j-1}(y_j-1)^2\{1+10\times\sin^2(\pi y_{j+1})\}+(y_{n_j}-1)^2\right\}+\sum_{j=1}^{n_j}u(x_j,10,100,4)$ $y_j=1+(1/4)\cdot(x_j+1)$ $u(x,a,k,m)=\begin{cases}k\cdot(x-a)^m, & x>a\\ 0, & -a\leqslant x\leqslant a\\ k\cdot(-x-a)^m, & x<-a\end{cases}$	0
f_5	$f_5(\boldsymbol{x})=0.1\left\{\sin^2(3\pi x_1)+\sum_{j=1}^{n_j-1}(x_j-1)^2\{1+\sin^2(3\pi x_{j+1})\}+(x_{n_j}-1)^2\{1+\sin^2(2\pi x_{n_j})\}\right\}+\sum_{j=1}^{n_j}u(x_j,5,100,4)$	0

测试函数f_6实现对某二输入输出线性模型参数的辨识，输入输出数据量为1 500，输入限幅为[−1, 1]，X_1、X_2表示状态变量，U_1、U_2表示输入控制量，系统离散状态空间方程描述为

$$\begin{bmatrix}X_1(t+1)\\X_2(t+1)\end{bmatrix}=\begin{bmatrix}x_1 & x_2\\x_3 & x_4\end{bmatrix}\begin{bmatrix}X_1(t)\\X_2(t)\end{bmatrix}+\begin{bmatrix}x_5 & x_6\\x_7 & x_8\end{bmatrix}\begin{bmatrix}U_1(t)\\U_2(t)\end{bmatrix} \tag{24}$$

适应度函数为使用辨识模型得到的各时刻状态变量与实际状态变量的方差：

$$f_6(\boldsymbol{x})=\sum_{t=1}^{1\,500}((X_1(t)-Z_1(t))^2+(X_2(t)-Z_2(t))^2 \tag{25}$$

式中：Z_1与Z_2为实际已知的状态量值。

4.3 结果与分析

本文实验在Windows系统下进行，衡量并行效益的计时方式采用了Visual C++提供的高精度时间函数，并通过设备端接口函数实现启发式优化算法迭代过程中所需要的大量随机数[31]。实验中种群个体的向量维度被设置为8，运用多个Block并行的算法种群数量被设置为800、1 600、2 400以及3 200，运用单个Block并行的算法种群数量被设置为100、150、200以及250，迭代次数设置为5 000。算法搜索、更新均存在随机性，最终收敛结果与执行时间均按10次实验的结果求取平均。

收敛误差结果如表2所示，可看出并行算法的收敛性与串行算法相近，证实了并行算法的有效性，表明算法的并行加速不是以牺牲收敛精度为代价。对比收敛结果，并行算法的收敛误差在许多情况下甚至小于串行算法，可见各种启发式优化算法在运用本文设计的并行框架下创造的优势是有效的。

启发式优化算法在测试函数f_1~f_6下的串、并行运行时间结果如表3所示，DE、GWO、WOA以及HHO算法利用GPU取得了明显的加速效果提升。其中WOA的效果最显著，在f_6下最高能达到超过350倍的加速比，这归功于WOA算法种群间数据依赖性低的特性。

表2 启发式优化算法收敛误差

Table 2 Convergence error of heuristic optimization algorithm

函数	种群	DE		HHO		GWO		WOA		种群	MFO	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU		CPU	GPU
f_1	800	0	0	0	0	6.29×10^{-5}	7.43×10^{-5}	0	0	100	6.26×10^{-8}	5.07×10^{-8}
	1 600	0	0	0	0	5.86×10^{-5}	7.64×10^{-5}	0	0	150	2.98×10^{-8}	4.17×10^{-8}
	2 400	0	0	0	0	4.76×10^{-5}	9.02×10^{-5}	0	0	200	1.19×10^{-8}	2.98×10^{-8}
	3 200	0	0	0	0	4.09×10^{-5}	8.11×10^{-5}	0	0	250	0	0
f_2	800	0	0	0	0	0	0	0	0	100	0	0
	1 600	0	0	0	0	0	0	0	0	150	0	0
	2 400	0	0	0	0	0	0	0	0	200	0	0
	3 200	0	0	0	0	0	0	0	0	250	0	0
f_3	800	0.032	0.019	0	0	0	0	0	0	100	0.25	0.29
	1 600	0.027	0.010	0	0	0	0	0	0	150	0.22	0.28
	2 400	0.026	0.009 2	0	0	0	0	0	0	200	0.30	0.23
	3 200	0.018	0.005 4	0	0	0	0	0	0	250	0.26	0.24
f_4	800	5.69×10^{-17}	5.69×10^{-17}	7.68×10^{-16}	5.69×10^{-17}	2.52×10^{-6}	2.39×10^{-7}	5.69×10^{-17}	5.69×10^{-17}	100	7.54×10^{-14}	1.05×10^{-14}
	1 600	5.69×10^{-17}	5.69×10^{-17}	3.42×10^{-16}	5.69×10^{-17}	6.60×10^{-6}	1.64×10^{-6}	5.69×10^{-17}	5.69×10^{-17}	150	6.39×10^{-14}	2.75×10^{-14}
	2 400	5.69×10^{-17}	5.69×10^{-17}	1.99×10^{-16}	5.69×10^{-17}	3.85×10^{-6}	5.43×10^{-6}	5.69×10^{-17}	5.69×10^{-17}	200	7.38×10^{-14}	6.61×10^{-14}
	3 200	5.69×10^{-17}	5.69×10^{-17}	5.69×10^{-17}	5.69×10^{-17}	7.04×10^{-6}	9.01×10^{-7}	5.69×10^{-17}	5.69×10^{-17}	250	9.79×10^{-15}	8.98×10^{-15}
f_5	800	8.95×10^{-14}	8.95×10^{-14}	5.38×10^{-13}	9.70×10^{-14}	2.27×10^{-10}	4.85×10^{-10}	9.18×10^{-14}	8.95×10^{-14}	100	9.01×10^{-14}	9.57×10^{-14}
	1 600	8.95×10^{-14}	8.95×10^{-14}	1.52×10^{-13}	9.01×10^{-14}	1.24×10^{-10}	4.17×10^{-10}	9.29×10^{-14}	9.01×10^{-14}	150	8.95×10^{-14}	8.95×10^{-14}
	2 400	8.95×10^{-14}	8.95×10^{-14}	1.35×10^{-13}	8.95×10^{-14}	1.15×10^{-10}	4.38×10^{-10}	8.95×10^{-14}	8.95×10^{-14}	200	8.95×10^{-14}	1.23×10^{-13}
	3 200	8.95×10^{-14}	8.95×10^{-14}	1.08×10^{-13}	8.95×10^{-14}	1.30×10^{-10}	5.21×10^{-10}	9.01×10^{-14}	8.95×10^{-14}	250	8.95×10^{-14}	8.95×10^{-14}
f_6	800	1.20×10^{-6}	4.12×10^{-7}	1.37×10^{-5}	1.29×10^{-5}	5.49×10^{-6}	1.14×10^{-6}	6.74×10^{-7}	6.71×10^{-7}	100	6.41×10^{-5}	8.49×10^{-5}
	1 600	1.04×10^{-6}	4.12×10^{-7}	1.31×10^{-5}	1.11×10^{-5}	3.89×10^{-6}	4.68×10^{-6}	6.27×10^{-7}	6.37×10^{-7}	150	7.65×10^{-5}	2.52×10^{-5}
	2 400	1.03×10^{-6}	4.12×10^{-7}	3.78×10^{-6}	9.71×10^{-6}	5.04×10^{-6}	1.97×10^{-6}	8.01×10^{-7}	6.81×10^{-7}	200	6.87×10^{-5}	1.07×10^{-5}
	3 200	9.82×10^{-7}	4.12×10^{-7}	1.84×10^{-6}	5.79×10^{-6}	1.61×10^{-6}	1.73E-06	7.64×10^{-7}	6.75×10^{-7}	250	5.74×10^{-5}	1.79×10^{-5}

表3 测试函数f_1~f_6不同种群维度算法运行时间与加速比

Table 3 Running time (ms) and acceleration ratio of algorithms with different dimensions under the test function f_1~f_6

函数	种群	DE			HHO			GWO			WOA			种群	MFO		
		CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比		CPU/ms	GPU/ms	加速比
f_1	800	882.3	15.1	**58.3**	2 827.6	109.5	**25.8**	2 560.4	271.0	**9.4**	639.3	21.6	**29.5**	100	298.6	133.9	**2.2**
	1 600	1 758.3	26.3	**66.8**	5 655.6	133.3	**42.4**	5 139.7	443.4	**11.6**	1 279.5	24.2	**52.9**	150	498.8	215.9	**2.3**
	2 400	2 638.4	52.7	**50.1**	8 487.4	153.6	**55.2**	7 720.3	693.6	**11.1**	1 921.4	25.5	**75.3**	200	733.5	300.3	**2.4**
	3 200	3 526.5	97.0	**36.4**	11 352.5	184.5	**61.5**	10 976.2	1 036.9	**10.6**	2 560.4	29.8	**85.9**	250	985.7	449.7	**2.2**
f_2	800	1 040.8	16.9	**61.5**	4 365.4	142.5	**30.6**	2 710.0	271.1	**10.0**	1 359.5	25.2	**53.9**	100	487.9	204.4	**2.4**
	1 600	2 073.5	27.8	**74.7**	8 730.9	159.0	**54.9**	5 420.0	422.6	**12.8**	2 719.0	30.0	**90.5**	150	795.1	315.8	**2.5**
	2 400	3 117.3	51.8	**60.2**	13 109.3	175.0	**74.9**	8 116.5	618.3	**13.1**	4 091.1	33.3	**122.9**	200	1 140.4	447.7	**2.5**
	3 200	4 184.0	94.8	**44.1**	17 495.3	205.0	**85.3**	10 846.3	1 017.3	**10.7**	5 445.8	38.8	**140.4**	250	1 504.2	615.7	**2.4**

DOI: 10.16182/j.issn1004731x.joss.23-0780

续表

函数	种群	DE			HHO			GWO			WOA			种群	MFO		
		CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比	CPU/ms	GPU/ms	加速比		CPU/ms	GPU/ms	加速比
f_3	800	1 324.6	33.2	**39.9**	5 484.2	237.1	**23.1**	2 849.2	314.9	**9.0**	1 911.0	48.0	**39.8**	100	576.6	253.6	**2.3**
	1 600	2 580.9	48.1	**53.7**	11 039.5	294.7	**37.5**	5 707.4	477.9	**11.9**	3 836.6	56.6	**67.8**	150	937.3	375.7	**2.5**
	2 400	3 855.6	56.2	**68.6**	16 427.6	353.0	**46.5**	8 799.1	717.3	**12.3**	5 735.4	70.0	**81.9**	200	1 323.0	562.5	**2.4**
	3 200	5 180.6	79.9	**64.8**	22 006.0	435.5	**50.5**	11 579.8	1 066.6	**10.9**	7 659.8	89.8	**85.3**	250	1 756.2	812.6	**2.2**
f_4	800	1 896.6	33.5	**56.5**	9 778.3	228.0	**42.9**	4 123.1	314.6	**13.1**	4 298.9	46.8	**91.8**	100	830.8	181.5	**4.6**
	1 600	3 687.7	38.6	**95.5**	19 487.1	262.1	**74.4**	7 592.4	469.3	**16.2**	8 586.7	45.9	**187.3**	150	1 307.2	288.5	**4.5**
	2 400	5 769.2	46.7	**123.4**	29 233.0	305.6	**95.7**	11 511.3	724.5	**15.9**	12 721.5	51.1	**249.0**	200	1 773.7	403.6	**4.4**
	3 200	7 584.9	77.8	**97.5**	38 952.6	339.2	**114.8**	15 235.2	1 067.7	**14.3**	17 057.9	54.4	**313.8**	250	2 325.6	616.6	**3.8**
f_5	800	2 003.5	37.2	**53.9**	10 146.4	246.3	**41.2**	3 891.5	305.2	**12.8**	4 399.2	49.5	**88.8**	100	823.2	183.8	**4.5**
	1 600	3 996.5	42.2	**94.7**	20 151.7	285.5	**70.6**	7 805.9	472.1	**16.5**	8 727.3	50.3	**173.5**	150	1 309.3	298.0	**4.4**
	2 400	5 998.1	48.1	**124.8**	30 161.1	334.4	**90.2**	11 775.4	714.1	**16.5**	13 055.5	55.1	**236.8**	200	1 805.7	421.7	**4.3**
	3 200	8 029.1	72.0	**111.5**	40 053.3	369.9	**108.3**	16 529.8	1 064.2	**15.5**	17 396.9	61.3	**283.9**	250	2 377.5	629.4	**3.8**
f_6	800	24 800.0	454.2	**54.6**	87 272.0	1 235.8	**70.6**	28 577.0	762.7	**37.5**	52 439.2	463.1	**113.2**	100	3 566.8	672.8	**5.3**
	1 600	49 536.9	486.0	**101.9**	172 596.4	1 479.3	**116.7**	57 565.1	925.8	**62.2**	104 504.1	512.7	**203.8**	150	5 445.8	795.1	**6.8**
	2 400	74 420.2	502.5	**148.1**	262 271.5	1 717.4	**152.7**	84 221.4	1 138.4	**74.0**	148 271.4	486.0	**305.1**	200	7 330.2	956.1	**7.7**
	3 200	99 287.5	554.3	**179.1**	351 528.8	1 967.9	**178.6**	112 004.3	1 506.7	**74.3**	199 616.9	557.2	**358.2**	250	9 241.3	1 261.7	**7.3**

基于本文的信息交互并行框架，并行极值过程的时间复杂度从$O(n)$降低至$O(\mathrm{lb}\ n)$，并行排序过程的时间复杂度从$O(n^2)$降低至$O(n)$。而并行多极值将寻优转换为排序，并未实现时间复杂度的下降，加上线程同步耗时因素，并行信息交互过程的时间收益远低于没有数据交互的个体并行过程，因此并行效益与算法信息交互数量呈负相关。在GWO中，多极值求解在并行过程占据主体，因此它的加速效益在同类型算法中是最低的。DE求解过程调用了大量耗时的随机数生成函数，而且存在频繁的高延迟的全局内存通信过程，因此低频的信息交互并未给DE带来高效的加速效益。

使用单个Block的MFO算法加速效果并不理想，Block中低延迟的共享内存数据通讯、更便捷的同步方式没有给少量的Thread并行带来大幅的加速，可见启发式优化算法的并行效益还是要基于大规模Thread并行。然而，而单个Block的Thread限度以单一SM有限的Thread分配额度和稀缺的共享内存资源限制了维度的扩展，同时SM资源的不充分开发也降低了GPU并行能力。

不同的适应度函数也会影响到启发式优化算法的加速比，计算相对简单的适应度函数，更新计算耗时较低，从而反衬数据传输、线程同步等过程在整个并行中时间占比提高，导致加速效益降低。从结果可以发现选取函数f_6进行计算是最耗时的，却能获得最高的并行加速比，该现象是因为复杂的适应度函数可以提高算法迭代过程的时间占比，使得GPU计算能力得到充分开发。

从表3中发现，种群大小从初始开始逐步递增，算法并行加速比也呈递增趋势，此时GPU的计算能力并未饱和。数据规模增大过程中，并行迭代过程带来的收益超过数据传输以及线程同步的减益，更多Thread数量的分配使warp占用率同步提高，并行效益开始显著提升。但这种递增趋势在种群规模到达一定程度后开始缓减，甚至出现下降，各算法加速比的峰值时刻不尽相同，此时GPU的计算能力开

始趋近于饱和，因此在硬件资源维持不变情况下，过度增加Thread将导致并行效益的恶化。

5 结论

本文重点研究了基于CUDA的并行启发式优化算法，根据此类算法均具有规模化数据以及可并行化特点，建立了一种通用、高效的CUDA并行框架，并给出了相应信息交互框架以及算法CUDA并行优化策略，具有如下优势：

(1) 有效处理数据传输、内核启动、线程同步等环节耗时的问题，并提供了信息交互的并行逻辑策略，适用于各类启发式优化算法，充分提高了并行效率。

(2) 针对启发式优化算法常见的消息交互过程(包含极值、均值、排序、种群混合、多极值)在串、并行中的相异性，设计了并行信息交互框架。有效解决了在串行算法向并行算法转变过程中，由于逻辑差异而导致的读写不安全、时序错乱的问题。同时，挖掘了CUDA并行优化策略，最大程度地提高了并行效益。

(3) 利用该并行框架实现了五种启发式优化算法(DE、GWO、MFO、WOA、HHO)的GPU加速计算，在五个测试函数以及线性模型辨识中进行应用，通过与基于CPU的串行计算相比，验证所设计框架的合理性与高效性。

值得注意的是，本文所提的并行框架套用在数据依赖密切的启发式优化算法时，由于种群维度扩展性不高，并行加速的效益不显著，这个问题将在未来工作做进一步的研究。

参考文献：

[1] 范衡, 朱贵杰, 李文姬, 等. 进化计算在复杂机电系统设计自动化中的应用综述[J]. 自动化学报, 2021, 47(7): 1495-1515.
Fan Zhun, Zhu Guijie, Li Wenji, et al. Applications of Evolutionary Computation in the Design Automation of Complex Mechatronic System: A Survey[J]. Acta Automatica Sinica, 2021, 47(7): 1495-1515.

[2] 丁进良, 杨翠娥, 陈远东, 等. 复杂工业过程智能优化决策系统的现状与展望[J]. 自动化学报, 2018, 44(11): 1931-1943.
Ding Jinliang, Yang Cuie, Chen Yuandong, et al. Research Progress and Prospects of Intelligent Optimization Decision Making in Complex Industrial Process[J]. Acta Automatica Sinica, 2018, 44(11): 1931-1943.

[3] 韩红桂, 张璐, 卢薇, 等. 城市污水处理过程动态多目标智能优化控制研究[J]. 自动化学报, 2021, 47(3): 620-629.
Han Honggui, Zhang Lu, Lu Wei, et al. Research on Dynamic Multiobjective Intelligent Optimal Control for Municipal Wastewater Treatment Process[J]. Acta Automatica Sinica, 2021, 47(3): 620-629.

[4] 李晓苏, 晁涛, 王松艳. 基于鱼群算法的运载火箭上升段弹道优化设计[J]. 系统仿真学报, 2018, 30(12): 4747-4753, 4759.
Li Xiaosu, Chao Tao, Wang Songyan. Trajectory Optimization Design of Ascending Phase for Solid Launch Vehicle Based on Fish-swarm Algorithm[J]. Journal of System Simulation, 2018, 30(12): 4747-4753, 4759.

[5] Eberhart R, Kennedy J. A New Optimizer Using Particle Swarm Theory[C]//Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95). Piscataway: IEEE, 1995: 39-43.

[6] Rainer Storn, Price K. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces[J]. Journal of Global Optimization, 1997, 11(4): 341-359.

[7] Christian Blum. Ant Colony Optimization: Introduction and Recent Trends[J]. Physics of Life Reviews, 2005, 2 (4): 353-373.

[8] Mirjalili S, Seyed Mohammad Mirjalili, Lewis A. Grey Wolf Optimizer[J]. Advances in Engineering Software, 2014, 69: 46-61.

[9] Mirjalili S. Moth-flame Optimization Algorithm: A Novel Nature-inspired Heuristic Paradigm[J]. Knowledge-Based Systems, 2015, 89: 228-249.

[10] Mirjalili S, Lewis A. The Whale Optimization Algorithm[J]. Advances in Engineering Software, 2016, 95: 51-67.

[11] Ali Asghar Heidari, Mirjalili S, Hossam Faris, et al. Harris Hawks Optimization: Algorithm and Applications[J]. Future Generation Computer Systems, 2019, 97: 849-872.

[12] Li Shimin, Chen Huiling, Wang Mingjing, et al. Slime Mould Algorithm: A New Method for Stochastic Optimization[J]. Future Generation Computer Systems, 2020, 111: 300-323.

[13] Wang Zhiheng, Liu Jianhua. Flamingo Search Algorithm: A New Swarm Intelligence Optimization Algorithm[J].

DOI: 10.16182/j.issn1004731x.joss.23-0780

IEEE Access, 2021, 9: 88564-88582.

[14] Zhao Weiguo, Wang Liying, Mirjalili S. Artificial Hummingbird Algorithm: A New Bio-inspired Optimizer with Its Engineering Applications[J]. Computer Methods in Applied Mechanics and Engineering, 2022, 388: 114194.

[15] Xue Bing, Zhang Mengjie, Browne W N. Particle Swarm Optimization for Feature Selection in Classification: A Multi-objective Approach[J]. IEEE Transactions on Cybernetics, 2013, 43(6): 1656-1671.

[16] Sun Yifan, Agostini N B, Dong Shi, et al. Summarizing CPU and GPU Design Trends with Product Data [EB/OL]. (2020-07-13) [2023-01-10]. https://arxiv. org/abs/1911.11313.

[17] 龚春叶, 刘杰, 包为民, 等. 后摩尔时代国产高性能并行应用软件生态建设综述[J]. 系统仿真学报, 2022, 34(10): 2107-2118.
Gong Chunye, Liu Jie, Bao Weimin, et al. Review on Ecological Construction of Domestic High-performance Parallel Application Software in Post Moore Era[J]. Journal of System Simulation, 2022, 34(10): 2107-2118.

[18] Surendra Kumar Shukla, Bhaskar Pant. Characterization of SPEC2006 Benchmarks Under Multicore Platform to Identify Critical Architectural Aspects[C]//International Conference on IoT, Intelligent Computing and Security. Singapore: Springer Nature Singapore, 2023: 199-206.

[19] 陈国军, 刘岩. 基于并行遗传算法的公路线形优化[J]. 系统仿真学报, 2013, 25(10): 2332-2336.
Chen Guojun, Liu Yan. Optimization of Road Alignment Using Parallel Genetic Algorithms[J]. Journal of System Simulation, 2013, 25(10): 2332-2336.

[20] Han Wencheng, Li Hao, Gong Maoguo, et al. Multi-swarm Particle Swarm Optimization Based on CUDA for Sparse Reconstruction[J]. Swarm and Evolutionary Computation, 2022, 75: 101153.

[21] Lucas de P Veronese, Renato A Krohling. Differential Evolution Algorithm on the GPU with C-CUDA[C]// IEEE Congress on Evolutionary Computation. Piscataway: IEEE, 2010: 1-7.

[22] Roberge V, Tarbouchi M. Hybrid Deterministic Non-deterministic Data-parallel Algorithm for Real-time Unmanned Aerial Vehicle Trajectory Planning in CUDA[J]. e-Prime-Advances in Electrical Engineering, Electronics and Energy, 2022, 2: 100085.

[23] Jayapriya J, Michael Arock. A Parallel GWO Technique for Aligning Multiple Molecular Sequences[C]//2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Piscataway: IEEE, 2015: 210-215.

[24] Satoshi Ueda, Hideaki Ogawa. Multi-fidelity Approach for Global Trajectory Optimization Using GPU-based Highly Parallel Architecture[J]. Aerospace Science and Technology, 2021, 116: 106829.

[25] Tan Ying, Ding Ke. A Survey on GPU-based Implementation of Swarm Intelligence Algorithms[J]. IEEE Transactions on Cybernetics, 2016, 46(9): 2028-2041.

[26] 张国, 王锐, 雷洪涛, 等. 并行智能优化算法研究进展[J]. 控制理论与应用, 2023, 40(1): 1-11.
Zhang Guo, Wang Rui, Lei Hongtao, et al. Survey on Parallel Intelligent Optimization Algorithms[J]. Control Theory & Applications, 2023, 40(1): 1-11.

[27] 姜洋洋. 基于卷积神经网络与CUDA加速计算的手势识别算法应用研究[J]. 系统仿真技术, 2020, 16(1): 22-26.
Jiang Yangyang. Development of Gesture Recognition Algorithm Based on Convolutional Neural Network and CUDA Accelerated Calculation[J]. System Simulation Technology, 2020, 16(1): 22-26.

[28] Li Yan. Implementation of CUDA and Hadoop Based System for Intelligent Guiding Evaluation Algorithm Based on Data Center Data Mining[C]//2022 International Conference on Inventive Computation Technologies (ICICT). Piscataway: IEEE, 2022: 1119-1125.

[29] Cheng J, Grossman M, McKercher T. Professional CUDA C Programming[M]. Indianapolis: John Wiley & Sons, 2014.

[30] Zhuo Yanhong, Zhang Tao, Du Feng, et al. A Parallel Particle Swarm Optimization Algorithm Based on GPU/CUDA[J]. Applied Soft Computing, 2023, 144: 110499.

[31] Mohammed Elbes, Shadi Alzubi, Tarek Kanan, et al. A Survey on Particle Swarm Optimization with Emphasis on Engineering and Network Applications[J]. Evolutionary Intelligence, 2019, 12(2): 113-129.

