

基于 GPU 加速的细粒度并行人工蜂群算法

王文亮¹, 王智广¹, 刘伟峰^{2,3}

(1 中国石油大学(北京) 地球物理与信息工程学院, 北京 102249;

2 中国石油化工股份有限公司 石油勘探开发研究院, 北京 100083;

3 中国石油化工集团公司 多波地震技术重点实验室, 北京 100083)

摘 要: 为了改善人工蜂群算法对于大规模数据、高复杂度问题的执行效率, 采用开放计算语言(OpenCL)并行编程模型, 提出一种基于图形处理器(GPU)加速的并行人工蜂群算法。将每只采蜜蜂映射到 OpenCL 的一个工作组, 跟随蜂采用局部轮盘赌选择, 使得人工蜂群算法在 GPU 中加速执行。实验结果表明, 并行人工蜂群算法取得了较好的优化效果, 提高了算法的运算速度。

关键词: 人工蜂群算法; 并行处理; 图形处理器; 开放计算语言

中图分类号: TP301.6

文献标识码: A

文章编号: 1000-7180(2013)03-0110-05

A Fine-grained Parallel Artificial Bee Colony Algorithm Based on GPU-Acceleration

WANG Wen-liang¹, WANG Zhi-guang¹, LIU Wei-feng^{2,3}

(1 College of Geophysics and Information Engineering, China University of Petroleum, Beijing 102249, China;

2 Sinopec Exploration and Production Research Institute, Beijing 100083, China;

3 Sinopec Key Laboratory of Multi Components Seismic Technology, Beijing 100083, China)

Abstract: Artificial Bee Colony (ABC) algorithm still needs plenty of computing time when it processes large-scale data and high complexity problems. In order to solve those problems in practice, we proposed a parallel ABC algorithm based on GPU acceleration with OpenCL parallel programming model. The process of parallel ABC is each employee bee mapping to OpenCL work group and onlooker using local roulette wheel selection, which makes parallel ABC speed up. The simulation results show that parallel ABC algorithm achieves an effective optimization and speeds up its running.

Key words: Artificial Bee Colony; Parallel process; GPU; OpenCL

1 引言

土耳其学者 Karaboga 在 2005 年提出人工蜂群算法^[1] (Artificial Bee Colony, ABC) 是一种相对比较新的、模拟真实世界蜜蜂群体采蜜行为的群体智能算法。相对于其它的群体智能算法, 人工蜂群算法的突出优点在于控制参数少、鲁棒性强。该算法已经成功解决了一系列问题^[2-3], 包括函数优化问题、组合优化问题、神经网络训练、IIR 数字滤波器、图像

处理等。

尽管人工蜂群算法有上述的优点, 但对于大规模(如多维度和多种群)寻找最优解的问题, 仍然需要很长的时间。为了提高算法的计算效率, 已有一些工作对蜂群算法的并行化进行了研究, 如文献[4-6], 但这些方法都是采用共享存储器或消息传递接口并行体系结构来进行多个处理机之间的通信。

在过去的十多年时间里, 图形处理器已经从仅适用于图形图像处理转变为适用于广泛的通用计

收稿日期: 2012-06-08; 修回日期: 2012-07-15

基金项目: 国家自然科学基金项目(60803159); 国家科技重大专项(2011X05005-005-006)

算. CPU-GPU 的混合异构计算系统已经成为并行计算的主流趋势之一,但据我们所知,目前还不存在人工蜂群算法在 CPU-GPU 异构计算环境下的并行化研究. 本文实现了一种在 OpenCL 平台下基于 GPU 加速执行人工蜂群算法的新方法. 与目前基于并行机采用共享存储器或消息传递接口并行体系结构不同的是,本文中每只采蜜蜂对应为 OpenCL 一个工作组,跟随蜂采用局部轮盘赌选择,加速人工蜂群算法的执行. 该算法能扩大群体的规模 and 问题的维度,提高算法的收敛速度和执行效率.

2 OpenCL 编程模型

GPU 通用计算是近年来的主要发展方向之一. 早期出现了多个基于 C 语言扩展的 GPU 编程模型,其中 Brook、Brook+、CUDA 是这方面具有代表性的编程模型,都具有易于使用、学习曲线低的特点. 文献[7]对这些编程模型进行了分析和比较. 其中, CUDA 编程模型在 GPU 计算领域被广泛地采用,但通常必须应用于专用的平台、供应商或硬件. 这些限制使得应用程序开发人员很难将单一源代码跨硬件平台地应用于多种 CPU、GPU 和其它类型的处理单元. 经我们的实践, OpenCL 编程模型在通用性上占有比较明显的优势,所以我们采用 OpenCL 编程模型来实现本文的算法.

OpenCL^[8]是一种为异构系统制定的开放工业标准,由非盈利的 Khronos 负责管理. 它用于 CPU、GPU 以及其他分离的计算设备等所组成的异构系统中开发支持多平台和多供应商的可移植的并行应用程序. OpenCL 是一种并行编程框架,包含编程语言, API, 函数库和运行时系统来支持软件在整个平台上的开发,并提供了一个低级别硬件抽象层和一个编译框架支持编程,程序员无需直接针对硬件底层编程,函数库和运行时环境会自动完成计算到硬件的映射.

3 人工蜂群算法

自然界中的蜜蜂总能较快地找到优质蜜源,同时它们能适应环境的变化. 蜜蜂采蜜的群体智能通过不同角色之间的交流、转换及协作来实现.

在人工蜂群算法中,蜜蜂寻找蜜源的过程类似于优化问题寻找最优解的过程,即每个优化问题的解对应于搜索空间中蜜源的位置,解的优劣对应于蜜源收益度的大小. 首先,人工蜂群算法随机分布初始化种群 P 的蜜源位置(数量为 SN), SN 的大小为

采蜜蜂或者跟随蜂的数量. 每个解 $x_i (i = 1, 2, \dots, SN)$ 是一个 D 维度的向量. 初始化之后,开始寻找最优解,这个过程包括采蜜蜂、跟随蜂和侦查蜂的搜索过程,循环往复,直到满足循环最大次数 $C = 1, 2, \dots, MCN$.

采蜜蜂在当前蜜源位置附近搜索新的蜜源,是一个局部概率选择工程,表达式为:

$$v_{ij} = x_{ij} + \varphi_{ij}(x_{ij} - x_{kj}) \quad (1)$$

其中, $k \in \{1, 2, \dots, SN\}$ 和 $j \in \{1, 2, \dots, D\}$ 是随机的选取,并且 $k \neq j$. φ_{ij} 是介于 $[-1, 1]$ 之间的随机数.

在采蜜蜂和侦查蜂阶段,更新蜜源信息,即新蜜源的收益度高于先前的蜜源,需要进行贪婪选择过程,其选择策略描述如下:

$$P\{T_s(X_i, V_i) = V_i\} = \begin{cases} 1, & f(V_i) \geq f(X_i) \\ 0, & f(V_i) < f(X_i) \end{cases} \quad (2)$$

贪婪选择过程保证种群保留精英的特征,进化的方向不会后退.

跟随蜂根据蜜源的概率值 P_i , 进行全局概率选择,其表达式如下:

$$P_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (3)$$

其中 fit_i 代表蜜源 i 的花蜜收益度的大小, SN 是蜜源的数量,即等于采蜜蜂或侦查蜂的数量.

假设放弃蜜源 x_i , 并且 $j \in \{1, 2, \dots, D\}$, 之后侦查蜂在解空间里随机搜索一个新的蜜源来替代原来的蜜源 x_i , 其操作如下:

$$x_i^j = x_{\min}^j + \text{rand}[0, 1](x_{\max}^j - x_{\min}^j) \quad (4)$$

综上所述,具体的人工蜂群算法的伪代码如表 1 所示.

4 并行人工蜂群算法

通过对人工蜂群算法原理和基于 GPU 加速的其他群体智能算法的研究,利用 GPU 的高速浮点运算和并行计算来提高算法的执行效率. 基于 GPU 实现人工蜂群算法并行化需要解决以下问题: 数据存储, 随机数的 GPU 处理, 采蜜蜂搜索过程的 GPU 实现, 跟随蜂搜索过程的 GPU 实现和 ABC 算法的 GPU 转化.

4.1 GPU 中的数据存储

考虑到 GPU 的计算效率,蜜源和适应度值存储在 GPU 的全局存储器里. 设采蜜蜂和跟随蜂种群大小都为 N , D 为搜索的维度大小, 用 $N \times D$ 的二维数

表 1 人工蜂群算法伪代码

算法:人工蜂群算法伪代码
1:初始设定参数: $M CN$, $limit$
2:目标函数 $f(x)$, 其中 $x = [x_1, x_2, \dots, x_d]^T$
3:随机初始蜜源 x_i , 其中 $i = 1, 2, \dots, SN$
4:计算收益度值 $fit(x_i)$, 并进行评估
5:采蜜蜂阶段:
6:通过公式(1)产生新的蜜源 x_i
7:评估蜜源
8:通过公式(2)对采蜜蜂的蜜源信息进行贪婪选择
9:跟随蜂阶段:
10:通过公式(3)计算蜜源 x_i 的全局选择概率 P_i
11:在蜜源 x_i 通过全局选择概率 P_i 产生一个新的蜜源 x_i
12:评估蜜源
13:通过公式(2)对跟随蜂的蜜源信息进行贪婪选择
14:侦查蜂阶段:
15:如果超过 $limit$, 放弃该蜜源, 并通过公式(4)随机产生一个蜜源来替代原来的蜜源
16:记录到目前为止最佳的蜜源信息
17:如果满足停止条件, 则停止计算并输出最优适应度值及相应的参数, 否则转向第 5 步.

组存储 N 个蜜源的位置 ($Foods[N][D]$), 用 N 维度的列向量存储适应度值 ($fitness[N]$)、选择蜜源可能性大小 ($prob[N]$)、目标函数值 ($f[N]$) 和不能更新蜜源位置的次数 ($trial[N]$), 以及用 $ObjValSol$ 存储新蜜源的目标函数值, $FitnessSol$ 存储新蜜源的适应度值, $GlobalMin$ 存储全局最优解.

4.2 随机数的 GPU 处理

在人工蜂群算法采蜜蜂和跟随蜂的搜索过程都需要使用大量的随机数, 而 GPU 本身不提供随机数生成函数, 若利用 CPU 生成随机数, 在传给 GPU 会进行大量的通信而影响效率. 本文采用梅森旋转算法^[9] (Mersenne Twister, MT) 在 GPU 中生成大量随机数. 定义一张 1024×1024 大小的随机数数组 $rand$, 在 CPU 上初始化 32 个种子, 并且传递到 GPU, 在 GPU 中生成随机数数组, 之后在采蜜蜂和跟随蜂的搜索过程中进行使用. 整个过程只调用一次梅森旋转算法来产生所需的随机数.

4.3 采蜜蜂搜索过程的 GPU 实现

在采蜜蜂寻找蜜源的过程中, 每只采蜜蜂都是自己进行各自区域及其附近邻域的搜索, 采蜜蜂之间依赖程度低, 所以天然地存在并行性. 如图 1 所示, 对于每个蜜源只有一只采蜜蜂, 对应 OpenCL 中一个工作组 (work group), 独立地进行搜索工作. 等到搜索结束, 对应每个蜜源通过适应度值贪婪选择,

保留适应度值较高的蜜源信息, 并将全局存储器中对应蜜源的信息进行更新.

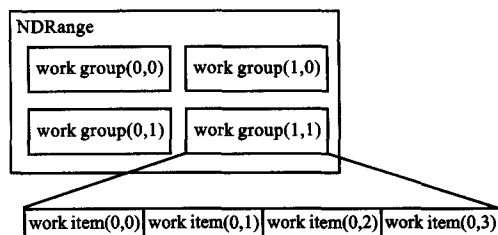


图 1 人工蜂群算法 GPU 并行化模型

4.4 跟随蜂搜索过程的 GPU 实现

在跟随蜂阶段, 由于其选择过程是全局概率选择, 是一种全局轮盘赌选择方式, 相互之间有依赖, 需要在整个种群搜集信息, 不适合细粒度并行算法. 本文采用局部轮盘赌选择方式进行选择操作. 将待处理的每只跟随蜂 i 对应搜索蜜源 i , 并选择其左右邻域分别为 ip 、 iq . 如图 2 所示, 跟随蜂 0 对应蜜源位置 0, 其左右邻域为 $N-1$ 、1. 对于这三个蜜源根据其适应度值进行轮盘赌选择, 选出符合条件的蜜源. 之后在选出的蜜源及其附近进行搜索, 等到搜索结束, 通过适应度值贪婪选择, 保留适应度值高的蜜源, 并将全局存储器中对应蜜源的信息进行更新. 由于跟随蜂的种群大小可能改变, 可以根据实际情况设置邻域大小.

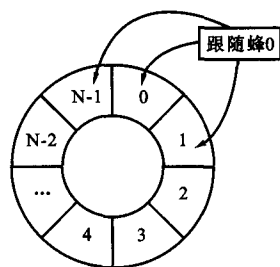


图 2 跟随蜂局部轮盘赌选择

4.5 人工蜂群算法的 GPU 实现

在解决上述问题的基础上, 根据 OpenCL 并行编程模型, 具体实现过程如下:

(1) 初始化参数. 在 CPU 中, 对 $Foods[N][D]$, $fitness[N]$, $prob[N]$, $trial[N]$ 以及梅森旋转算法种子等参数进行初始化.

(2) 将数据和参数传送到 GPU 中.

(3) 产生随机数. 采用 4.2 节提到的梅森旋转算法一次性产生 1024×1024 大小的随机数数组 $rand$.

(4) 采蜜蜂阶段. 根据 4.3 节采蜜蜂搜索过程并行化方法, 让每只采蜜蜂在原蜜源附近进行采蜜, 并计算适应度值, 按照适应度值大小更新蜜源信息.

(5) 跟随蜂阶段. 根据 4.4 节跟随蜂搜索过程并

行化方法,按照可能性概率进行局部轮盘赌选择,然后每只采蜜蜂在原蜜源附近进行采蜜,并计算适应度值,按照适应度值大小更新蜜源信息.

(6) 侦查蜂阶段. 若搜寻次数超过一定限制 *limit*, 仍没找到具有更高适应度值的蜜源, 则放弃该蜜源, 并且随机产生一个新的蜜源.

(7) 记录到目前的全局最优解, 并跳转至(4), 直到满足算法结束条件, 把最终的全局最优解等信息从 GPU 传回 CPU.

5 实验及其结果

5.1 测试的基准函数

为了评价算法性能, 我们利用本文的 GPU 人工蜂群算法执行 D. Karaboge 和 B. Basturk^[10] 中介绍的三个经典基准函数, 如表 2 所示.

表 2 基准函数

$f_1(\vec{x}) = \frac{1}{4000}(\sum_{i=1}^n (x_i - 100)^2) - (\prod_{i=1}^n \cos(\frac{x_i - 100}{\sqrt{i}})) + 1$
$f_2(\vec{x}) = \sum_{i=1}^n (x_i^2 - 10\cos(2\pi x_i) + 10)$
$f_3(\vec{x}) = \sum_{i=1}^{n-1} 100 (x_{i+1} - x_i^2)^2 + (x_i - 1)^2$

第一个函数是 Griewank 函数, 是可变维数的多峰函数, 其全局最小值为 0, 初始的定义域为 $[-600, 600]$. 第二个函数是 Rastrigin 函数, 是具有大量局部最优点的复杂多峰函数, 其全局最小值为 0, 初始的定义域为 $[-5.12, 5.12]$. 第三个函数是 Rosenbrock 函数, 全局最优解位于弯曲而又狭长山谷中, 是变量依赖性极强的多峰函数, 其全局最小值为 0, 初始的定义域为 $[-50, 50]$.

5.2 实验环境及参数设定

本实验 CPU 采用 Intel core i3 M380 处理器, 2GB 内存. GPU 为 ATI Mobility Radeon HD 5470 处理器. 操作系统为 Ubuntu 11. 04, SDK 为 AMDAPP v2. 5, gcc4. 5. 串行和并行人工蜂群算法都设置最大循环次数为 4000 次. 种群初始为 60, 采蜜蜂和跟随蜂各为 30, 侦查蜂为 1. *limit* 参数设置为 100.

5.3 实验结果及其分析

本实验为了说明问题, 分别选取 10, 30, 50 三个维度进行实验. 测试函数为表 2 给出的三个函数 Griewank, Rastrigin 和 Rosenbrock. 其结果如表 3、表 4、表 5 所示. 表中数据第一行为时间, 单位为秒; 第二行为 30 次平均的全局最小值.

表 3 Griewank 函数测试结果

维度	D=10	D=30	D=50
串行	7. 19 3. 176e-17	19. 75 3. 332e-16	32. 43 8. 343e-16
并行	4. 63 5. 069e-17	10. 91 4. 936e-16	16. 91 2. 589e-15

表 4 Rastrigin 函数测试结果

维度	D=10	D=30	D=50
串行	5. 04 3. 784e-17	12. 35 4. 660e-16	19. 73 3. 638e-14
并行	3. 568 8. 132e-17	7. 192 3. 030e-14	10. 818 2. 746e-07

表 5 Rosenbrock 函数测试结果

维度	D=10	D=30	D=50
串行	2. 11 1. 861e-01	3. 57 6. 344e-01	5. 11 4. 490e-01
并行	2. 19 3. 052e-01	3. 00 4. 643e-01	3. 86 1. 949e+00

通过测试结果可以看出, 在 10, 30, 50 维度时, Griewank 函数加速比分别为 1. 55x, 1. 81x, 1. 92x; Rastrigin 函数加速比分别为 1. 41x, 1. 72x, 1. 83x; Rosenbrock 函数加速比分别为 0. 96x, 1. 19x, 1. 33x. 说明并行算法是有效的, 加速比与维度的关系如图 3 所示. Griewank 函数和 Rastrigin 函数在维度大于 50 之后加速比已经趋于平缓, Rosenbrock 函数在维度大于 50 之后加速比继续增加.

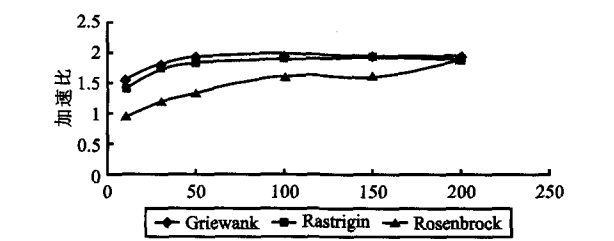


图 3 各函数加速比与维度的关系

在精度相同的情形下, 并行算法的收敛速度比串行算法慢, 如图 4、图 5 所示. 以 Rastrigin 函数和

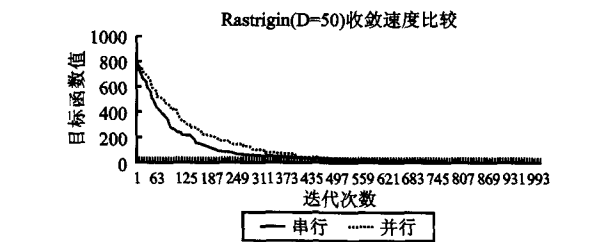


图 4 Rastrigin 函数性能

Rosenbrock 函数在维度为 50 的情况为例,经过分析,可能的原因是没有每次产生随机数都用不同的种子,导致并行算法容易陷入局部最优解,使得收敛速度比串行算法稍慢。

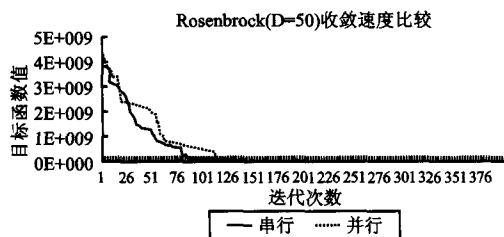


图5 Rosenbrock 函数性能

6 结束语

本文实现了在 OpenCL 平台下基于 GPU 加速的并行人工蜂群算法,其特点总结如下:(1)算法中蜂群的规模和执行时间是类似线性关系,当蜂群的规模增大时,本文的算法将显示更好的效果;(2)利用 GPU 浮点计算能力,提高了算法的运算速度。

对于本文并行人工蜂群算法的进一步研究,可以从以下几方面考虑。(1)分析并且解决并行版本收敛速度变慢的问题;(2)进一步优化算法,提高算法的执行效率;(3)将人工蜂群算法与其他群体智能算法相结合,寻找解决优化问题更好的方法。

参考文献:

- [1] Karaboga D, Akay B. A survey : algorithms simulating bee swarm intelligence[J]. Artificial Intelligence Review. 2009(31):61-85.
- [2] D. Karaboga. Artificial bee colony algorithm. scholarpedia [EB/OL]. [2012-04-20]. http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm.
- [3] Karaboga D. An idea based on honeybee swarm for numerical optimization[R]. Erciyes; Erciyes University, 2005.
- [4] Parpinelli R S, Benitez C, Lopes H. Parallel approaches for the artificial bee colony algorithm[J]. Handbook of Swarm Intelligence, Adaptation, Learning and Optimization, 2010(8):329-345.
- [5] Banharnsakun A, Achalakul T, Sirinaovakul B. Artificial bee colony algorithm on distributed environments [C]//2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC). Kitakyushu, Japan: IEEE, 2010:13-18.
- [6] Narasimhan H. Parallel artificial bee colony (PABC) algorithm[C]//2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). Coimbatore, India: IEEE, 2009:306-311.
- [7] 刘伟峰,王智广. 细粒度并行计算编程模型研究[J]. 微电子学与计算机, 2008;25(10):103-106.
- [8] Khronos OpenCL Working Group. The openCL specification[M]. Beaverton, USA: 2009.
- [9] Wikipedia. Mersenne Twister[EB/OL]. [2012-2-10]. http://en.wikipedia.org/wiki/Mersenne_twister.
- [10] Karaboga D, Basturk B. On the performance of artificial bee colony (ABC) algorithm[J]. Applied Soft Computing, 2008;8(1):687-697.

作者简介:



王文亮 男, (1987—), 硕士研究生, CCF 学生会员. 研究方向为并行计算。



王智广 男, (1964—), 博士, 教授, CCF 高级会员. 研究方向为分布式计算和并行计算。



刘伟峰 男, (1981—), 硕士, 工程师, CCF 会员. 研究方向为并行计算、计算机图形学和可视化。