文章编号: 1001-0920(2008)06-0697-04

# 一种基于 GPU 加速细粒度并行遗传算法的实现方法

李建明, 迟忠先, 万单领
(大连理工大学 电子与信息工程学院, 辽宁 大连 116024)

**摘 要**: 为改善遗传算法对大规模多变量求解的性能,提出一种基于图形处理器(GPU)加速细粒度并行遗传算法的实现方法. 将并行遗传算法求解过程转化为 GPU 纹理渲染过程,使得遗传算法在 GPU 中加速执行. 实验结果表明,该算法抑制了早熟现象,增大了并行遗传算法的种群规模,提高了算法的运算速度,并为普通用户研究并行遗传算法提供了一种可行的方法.

**关键词**: 遗传算法; 并行处理; 图形处理器; 细粒度
**中图分类号**: TP301.6 **文献标识码**: A

# Parallel genetic algorithm based on fine-grained model with GPU-accelerated

*LI Jian-ming, CHI Zhong-xian, WAN Dan-ling*
(School of Electronic and Information Engineering, Dalian University of Technology, Dalian 116024, China. Correspondent: LI Jian-ming, E-mail: lijm@dlut.edu.cn)

**Abstract**: An algorithm based on GPU (graphics processing unit) acceleration fine-grained parallel genetic algorithm (PGA) is proposed to improve the performance of genetic algorithm for application to large-scale problems and multivariable solutions. The process of parallel genetic algorithms is converted into that of texture-rendering based on GPU, which maks PGA greatly accelerated in it. The experimental results show that the algorithm inhibits the phenomenon of premature efficiently, increases the particle population in the PGA, speeds up its running and provides ordinary user with a feasible PGA solution.

**Key words**: Genetic algorithm; Parallel process; Graphics processing unit (GPU); Fine-grained

## 1 引 言

遗传算法(GA)是一种随机搜索技术,用来求解优化问题的近优解[1]. 对于中小规模的应用问题,GA 得到了广泛的应用,取得了很好的优化效果. 而对于大规模或超大规模的多变量求解任务,GA 算法往往需要大量的计算时间而显得力不从心. 并行 GA 算法能较大幅度地缩减问题求解的时间,因而成为一个研究热点[2-4].

细粒度并行遗传算法(FGPGA)是并行遗传算法中的一个重要模型[5-7],具有维持群体多样性、抑制早熟和保持最大并行性等优势. 同时,在处理高维空间的优化问题时,FGPGA 能得到更好的优化效果[5]. 当前关于 FGPGA 的研究主要在并行机上运行或用多线程技术模拟,在取得较好优化效果的同时存在下述不足:1) FGPGA 中每个个体运行单独的进程,而复杂问题几百甚至更多的个体规模导致了进程间很大的通信损耗,大多并行机难以承受;2) 多线程技术是在 CPU 上用串行来模拟并行,并不能真正提高速度;3) 大多研究人员很难接触到上述并行机,同时并行机的管理和使用相对复杂.

近年来,图形处理器(GPU)高速发展,提高了计算机图形处理的速度. 同时 GPU 的高速度、并行计算和可编程功能为通用计算提供了良好的并行计算平台[8]. 通过 GPU 进行并行优化算法的加速,是解决上述 FGPGA 所面对问题的一种可行的方法.

Wong 提出了基于 GPU 的进化规划算法[9],得到了很好的优化效果. 但该算法通过 CPU 进行竞争操作和最优值搜索,需要较多的 CPU 与 GPU 数据交换,对算法速度影响较大. Wong 同时指出,采用该方法来加速遗传算法,由于较多的数据交换和多遍 GPU 渲染,得不到好的加速效果. Yu 提出基于 GPU 的遗传算法[10],使用实数编码进行相关交

**收稿日期**: 2007-02-07; **修回日期**: 2007-06-14.
**作者简介**: 李建明(1975—),男,辽宁葫芦岛人,博士生,从事进化计算、GPU 通用计算、图像处理等研究; 迟忠先(1939—),男,辽宁大连人,教授,博士生导师,从事数据仓库与智能控制的研究.

叉和变异操作,得到了很好的加速效果. 但 Yu 指出,由于 GPU 中没有相应的二进制操作函数,遗传算法中广泛应用的二进制编码很难在 GPU 中实现.

在上述方法的基础上,本文提出一种改进的基于 GPU 加速并行遗传算法的实现方法(GFGPGA). 通过 GPU 约减技术减少 CPU 与 GPU 的数据交换,提高算法性能,同时在 GPU 中实现了遗传算法的二进制编码方式. 该方法增大了算法的个体规模,提高了算法运行速度,并为普通用户研究并行遗传算法提供了一种可行的途径.

## 2 基于 GPU 的细粒度并行遗传算法

通过对细粒度并行遗传算法的研究,将其求解过程转化为 GPU 纹理渲染过程,利用 GPU 的高速浮点运算和并行计算提高了算法速度,增大了个体规模. 在 GPU 中实现该算法需解决以下问题:数据存储,最佳适应值的 GPU 搜索,随机数的 GPU 处理,GPU 的二进制操作,遗传操作的 GPU 实现和 FGPGA 算法的 GPU 转化.

### 2.1 GPU 中的数据存储

GPU 中并行处理的对象为纹理图像,因此首先要解决遗传算法中相关数据的纹理存储问题.

对于多维空间的 GA 求解问题,设空间维数为 $d$,群体规模为 $n$,每条染色体包含 $d$ 个基因组,即有 $d$ 维分量. 使用像素的一个颜色通道保存染色体的一个基因数据,即使用 $\lceil d/4 \rceil$ 张像素个数为 $n$ 的纹理组保存相应染色体数据信息. 对于 1 维数据信息的适应值,使用像素的 $r$ 通道存储数组的一个元素,一张像素个数为 $n$ 的纹理即可保存相应数组.

下面以群体规模为 $n$,每条染色体包含 16 个基因组的情况为例,说明相应的纹理转换方法. 图 1 为详细存储过程,其中各纹理像素个数为 $n$,所有染色体 1-4 基因组信息用纹理 1 的相应像素保存,第 $n$ 个像素的 $(r, g, b, a)$ 通道保存第 $n$ 个染色体 1-4 基因组 $(x_1, x_2, x_3, x_4)$,依此类推.

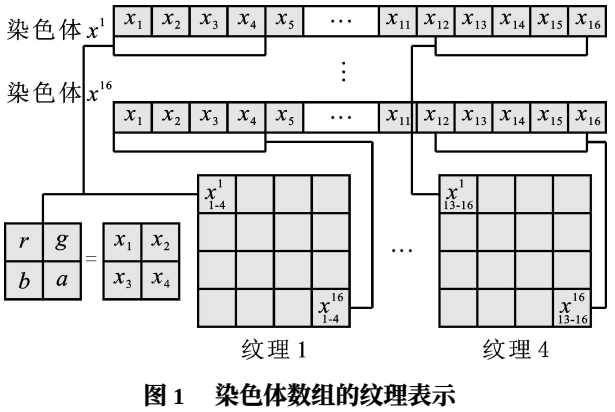


**图 1 染色体数组的纹理表示**

### 2.2 最佳适应值的 GPU 并行搜索

GPU 中的像素渲染程序是并行处理数据流输入的同时产生数据流输出的过程,采用传统的 GPU 程序无法实现最佳适应值的搜索. 而如果在 CPU 中进行搜索,则需将适应值纹理数据从 GPU 回传到 CPU. 但由于硬件限制,该数据回传的速度很慢,将极大地影响算法速度. 为此,可采用 GPU 约减方法解决这一问题[11],实现 GPU 中适应值的快速选取,提高程序的效率.

### 2.3 随机数的 GPU 处理

算法遗传操作时需要大量的随机数,而 GPU 不提供随机数生成函数,若利用 CPU 生成随机数再传给 GPU 的方法,将由于大量数据传递而影响效率. 作者采用随机值纹理的方法来解决这一问题. 定义一张 1 024 ×1 024 大小的随机值纹理,在 CPU 中使用随机值对其初始化,在每次进行遗传操作前,由 CPU 随机生成 1 个 2 维随机偏移量 $(i, j)$ 和 1 个 2 维干扰值 $(p, q)$ 传入 GPU,使用如下方法计算随机值:

$$(x_1, y_1) = in + (i, j) + (p, q)\ \mathrm{TR}(in)\ rg,$$

$$r_1 = \mathrm{TR}(x_1, y_1)\ r,$$

其中 $in$ 为待处理像素的纹理坐标. 本文算法需要 2 个随机纹理,一个是随机概率纹理,用于随机得到一个概率,其每个像素各通道取值范围为 0 ～ 1 的浮点数;另一个是随机交叉位置纹理,用于在交叉操作中随机产生一个交叉位置,其每个像素各通道取值范围为 0 ～ $(L - 1)$ 的整数,$L$ 为染色体长度.

### 2.4 GPU 的二进制操作

1) 染色体编码

本文采用二进制编码,所用显卡支持纹理像素的最大整数精度为 $2^{24}$,对应的二进制串长度为 24. 该编码长度可以满足大多数优化问题的精度要求,当需要更高的精度和更长编码时,可使用多个像素通道来保存染色体信息.

2) GPU 中的二进制操作

GPU 中没有位操作的指令,可通过整数 $R$ 与 $2^i$ 差值的正负来判断 $R$ 的第 $i$ 位是否为 1,从而模拟位运算.

### 2.5 遗传操作的 GPU 实现

1) 选择

由于全局轮盘赌法不适合细粒度并行算法,本文使用星形局部选择法进行选择操作. 对于待处理个体,在其 3 ×3 邻域内进行局部选择,根据待处理个体自身及邻域共 9 个个体的适应度进行轮盘赌选择,由选出的个体替换待处理个体. 由于邻域的大小将影响信息传播的速度和种群的多样化,可根据实际情况设置邻域的大小.

2) 交叉

针对 GPU 的并行特点,设计奇偶邻接配对法进行个体对的选取,并使用单点交叉进行交叉操作. 设染色体纹理宽和高分别为 $w$ 和 $h$,二进制编码长度为 $L$,交叉操作的具体步骤如下:

Step1: 定义随机交叉位置纹理 TCP, 宽为 $w/2$,高为 $h$. 并行渲染该纹理,对每个像素取随机值 $r$,若交叉率 $p_c > r$,则像素$(r, g, b, a)$ 各通道分别赋值 0 ～ $(L - 1)$ 间的随机值,否则各通道赋值 - 1.

Step2: 渲染染色体纹理. 对于第 $k$ 行像素,将第 $2i$ 和 $2i + 1$ 个像素 $X_{2i}$ 和 $X_{2i+1}$ 选为一组进行交叉操作,生成像素 $C_{2i}$ 和 $C_{2i+1}$. 该操作依次处理像素的$(r, g, b, a)$ 通道,对于 $j$ 通道,读取随机交叉位置纹理 TCP 第 $k$ 行,第 $i$ 个像素的 $j$ 通道值 $P_j$ 作为交叉点,若 $P_j = -1$,则直接将 $X_{2i}$ 和 $X_{2i+1}$ 复制到 $C_{2i}$ 和 $C_{2i+1}$,否则使用 2.4 节交叉算法进行交叉操作. 其中:像素 $C_{2i}$ 的 $j$ 通道由 $X_{2i}$ 像素 $j$ 通道的 0 ～ $(P_j - 1)$ 位和 $X_{2i+1}$ 像素 $j$ 通道的 $P_j$ ～ $(L - 1)$ 位组成; $C_{2i+1}$ 的 $j$ 通道由 $X_{2i+1}$ 的 $j$ 通道的 0 ～ $(P_j - 1)$ 位和 $X_{2i}$ 的 $j$ 通道 $P_j$ ～ $(L - 1)$ 位组成.

Step3: 由 Step2 交叉操作生成新纹理,并用该纹理替换原染色体纹理,产生交叉后新的种群.

因为交叉操作处理的染色体纹理通过局部随机选择产生,因此奇偶邻接配对选择的个体,也保持了一定的随机性.

3) 变异

使用 2.4 节的二进制操作方法进行变异操作,对染色体所有二进制位进行循环操作. 从随机概率纹理上取随机值 $R$,当 $R$ 大于变异率 $p_m$ 时,对当前位进行变异操作.

### 2.6 FGPGA 算法的 GPU 转化

在解决上述问题的基础上,将 FGPGA 算法转化为 GPU 中纹理渲染过程,建立模型如下:

1) 变量定义

设优化问题目标函数为 $f(x)$,定义域为$[-r, r]$,维度为 $d$,基因数为 $n$. 定义纹理如下:

染色体纹理组 TX$[m]$ 存储染色体的编码,其中 $m = \lceil d/4 \rceil$,各纹理像素个数为 $n$;适应值纹理 TF 存储当前的适应值,像素个数为 $n$;随机概率纹理 TR 存储 0～1 的随机值;随机交叉位置纹理 TCP 存储随机交叉位置信息. 对于纹理 $T$, $T(j)$ 代表纹理的第 $j$ 个像素, $T(j)c$ 代表该像素的 $c$ 颜色通道.

2) 算法描述

Step1: 读入交叉率 $p_c$,变异率 $p_m$ 和染色体编码长度 $L$. 初始化历史最佳适应值,初始化染色体纹理组 TX$[m]$, 随机概率纹理 TR 和随机位置纹理 TCP,生成空白适应值纹理 TF.

Step2: 加载染色体纹理组到 GPU. 根据适应值函数 $f(x)$ 对纹理像素并行计算,计算过程如下:

$$\mathrm{TF}(i) = f(X_i).$$

用输出纹理替换当前适应值纹理 TF.

Step3: 用 2.2 节的方法搜索纹理 TF,得到当前群体最佳适应值.

Step4: 比较当前群体最佳适应值和历史最佳适应值,更新历史最佳适应值.

Step5: 如果停止条件满足,则输出结果,程序终止;否则转 Step6.

Step6: 加载随机概率纹理 TR 和适应值纹理 TF,按 2.5 节选择操作方法渲染得到新染色体纹理组.

Step7: 加载随机概率纹理 TR 和随机位置纹理 TCP,进行交叉操作渲染得到新的染色体纹理组.

Step8: 加载随机概率纹理 TR,进行变异操作渲染得到新的染色体纹理组.

Step9: 循环执行 Step3 ～ Step8,直到求得最终结果.

## 3 实验与分析

实验环境为 Pentium IV 2.66 GHz CPU, 256M RAM,NVIDIA GeForce 6800 LE 显卡. 本文选择 3 个典型的多峰函数来测试 GFGPGA 算法的性能,测试函数如下:

$f_1$(Schwefel 函数)

$$f(x) = -\sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right),$$
$$-500 \leq x_i \leq 500;$$

$f_2$(Shaffer 函数)

$$f(x,y) = 0.5 - \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{(1 + 0.001(x^2+y^2))^2},$$
$$-100 < x, y < 100;$$

$f_3$(Camel 函数)

$$f(x,y) = (4 - 2.1x^2 + x^4/3)x^2 + xy + (-4 + 4y^2)y^2,$$
$$-100 < x, y < 100.$$

上述函数存在多个局部极值,遗传算法很容易停滞在局部极值点,出现早熟现象. 这里分别采用标准遗传算法(SGA) 和本文算法,对每个函数的不同个体数和不同繁衍次数进行 100 次实验. 实验数据表明,本文算法具有以下特点:

1) 抑制早熟现象

本文算法较好地保持了 FGPGA 抑制早熟的特性. 表 1 的数据显示,对于不同函数,算法收敛性能不完全相同,但本文算法求出全局最优解的成功率

略高于标准的遗传算法,这一点在函数 $f_2$ 的优化中表现得较为明显,表明本文算法能有效地抑制早熟现象,更容易找到全局最优点.

表 1 平均收敛次数比较

| 函数 | 算法 | 发生早熟现象次数 | 求出全局最优点次数 | 成功率/ % |
|---|---|---|---|---|
| $f_1$ | SGA | 39 | 61 | 61 |
| | GFGPGA | 22 | 78 | 78 |
| $f_2$ | SGA | 36 | 64 | 64 |
| | GFGPGA | 3 | 97 | 97 |
| $f_3$ | SGA | 21 | 79 | 79 |
| | GFGPGA | 16 | 84 | 84 |

2) 较好的加速比

表 2 数据显示了不同个体规模进行 1 000 次迭代时,本文算法对标准 GA 算法的加速比,个体数范围为 200 到 10 000 时,加速范围为 1.4～73.6 倍;同时表明,个体规模越大,加速效果越好. 与 Wong 算法相比,本文算法避免了 CPU 与 GPU 中的数据交换,加速比高于 Wong 算法. 以种群规模 3 200 为例,Wong 算法加速比在 3 到 4 倍之间,而本文算法在 9 倍以上.

表 2 GPU 算法相对 CPU 算法的加速比

| 种群规模 | $f_1$ | | |
|---|---|---|---|
| | SGA/ s | GFGPGA/ s | 加速比 |
| 200 | 9.3 | 5.8 | 1.6 |
| 400 | 19.4 | 6.1 | 3.2 |
| 800 | 41.6 | 8.1 | 5.1 |
| 3 200 | 229.1 | 23.9 | 9.6 |
| 10 000 | 1 303.1 | 62.1 | 21 |
| 种群规模 | $f_2$ | | |
| | SGA/ s | GFGPGA/ s | 加速比 |
| 200 | 5.1 | 3.3 | 1.59 |
| 400 | 10.6 | 3.5 | 3.1 |
| 800 | 24.5 | 4.5 | 5.4 |
| 3 200 | 160.8 | 12.5 | 12.9 |
| 10 000 | 1 086.7 | 21.4 | 50.8 |
| 种群规模 | $f_3$ | | |
| | SGA/ s | GFGPGA/ s | 加速比 |
| 200 | 3.0 | 2.2 | 1.4 |
| 400 | 6.8 | 2.5 | 2.7 |
| 800 | 16.2 | 3.3 | 4.9 |
| 3 200 | 128.8 | 7.6 | 16.9 |
| 10 000 | 978.5 | 13.3 | 73.6 |

3) 增大了并行 GA 个体规模

对于函数 $f_2$,本文算法运行时间与迭代次数的关系如图 2 所示. 数据显示,在选取上述规模个体的情况下,本文算法得到了很好的运行速度,个体规模为 10 000,迭代 1 000 次时,运行时间在 22 s 左右. 如此大的个体规模,在当前的并行机上是很难实现的. 理论上,本文算法的个体规模只受限于 GPU 可加载纹理的像素个数,本文使用的显卡支持 4 096 × 4 096 的纹理,即个体规模可以达到 16 777 216 个,对于大多数优化问题,该个体规模已能满足需要.

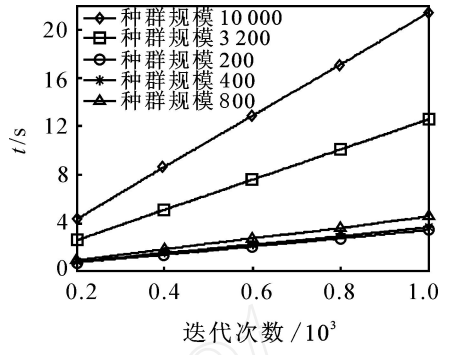


图 2 不同粒子规模在不同迭代次数下的运行时间

本文算法的运行时间随群体规模的增大呈次线性增长. 图 3 数据显示了进行 1 000 次迭代时,群体规模对运行时间的影响. 设个体数为 $n$. 数据显示,CPU 算法中运行时间与个体规模近似为一种线性关系,即时间复杂度为 $o(n)$;而 GPU 算法中,运行时间与个体数近似为一种次线性的关系,运行时间随个体规模的增长速度远远小于 CPU 算法,随着个体规模增大,本文算法保持了较快的运行速度.

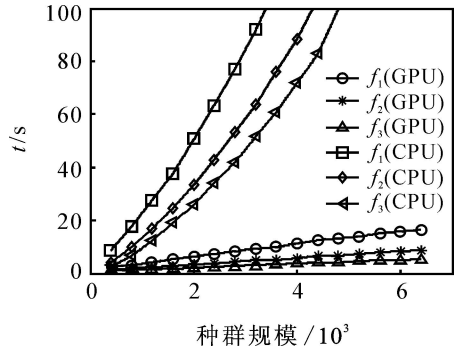


图 3 算法运行时间与粒子群规模的关系

## 4 结 论

本文提出了一种基于 GPU 加速细粒度并行遗传算法的实现方法,将并行 GA 转化为 GPU 纹理渲染过程. 算法具有如下特点:1) 较好地维持群体多样性,保持了 FGPGA 抑制早熟的优良特性;2) 利用 GPU 的高速浮点计算和并行特性,提高了算法的运算速度,取得了较好的加速比;3) 增大了细粒度并行 GA 的个体规模,同时算法执行时间和个体规模为次线性关系,当需要足够大的个体规模来解决复杂问题时,本文算法是一个较好的解决方案;4) 由于当前普通 PC 机的显卡中,大都配置 GPU 芯片,更多的研究人员可使用本文的并行 GA 算法来解决实际问题,避免了并行机等硬件环境对算法应用的限制.

进一步的研究可考虑以下两点:1) 将 GPU 应用于其他改进的 GA 算法中;2) 使用 GPU 对其他并行优化算法进行处理.

(下转第 704 页)

的奇异值分解方法,维数降低到 SOON 和 ARC 的 1/9,所以速度明显要快几倍. 本文方法所得到的图库归类如图 3 所示.

## 5 结 论

本文首先改进奇异值分解,用于高维空间的降维;然后构造低维空间的密度函数,通过爬山策略实现图像数据库的聚类和归类. 一方面,利用图库聚类不需要固定 $k$ 值以及对噪音不敏感等优点,可以加快基于内容的图像检索的速度. 另一方面,在聚类的基础上提取每一个聚类中最具有代表性的图例供用户使用,改善了基于内容的图像检索的人机交互性能和界面.

## 参考文献( References)


[1] Datta Ritendrata, Li Jia, Wang James Z. Content-based image retrieval-approaches and trends of the new age [C]. Proc of the 7th Int Workshop on Multimedia Information Retrieval. Singapore: Conjunction with ACM Int Conf on Multimedia, 2005: 253-262.

[2] 郑欣, 林学闾. 图像数据库的保局聚类[J]. 计算机研究与发展, 2006, 43(3): 463-469.
(Zheng X, Lin X Y. Locality preserving clustering for image database [J]. J of Computer Research and Development, 2006, 43(3): 463-469.)

[3] Petros Drineas, Alan M Frieze, Ravi Kannan, et al. Clustering large graphs via the singular value decomposition[J]. Machine Learning, 2004, 56(3): 9-33.

[4] Andrew Y Ng, Michael I Jordan, Yair Weiss. On spectral clustering: Analysis and an algorithm [M]. Cambridge: MIT Press, 2001.

[5] Francis R Bach, Michael I Jordan. Learning spectral clustering[M]. Cambridge: MIT Press, 2004.

[6] Bertrand Le Saux, Nozha Boujemaa. Unsupervised robust clustering for image database categorization[C]. IEEE-IAPR Int Conf on Pattern Recognition (ICPR'2002). Quebec, 2002, (1): 259-262.

[7] Frigui Hichem, Nozha Boujemaa, Soon-Ann Lim. Unsupervised clustering and feature discrimination with application to image database categorization[C]. Proc of the IFSA World Congress and 20th NAFIPS Int Conf. Vancouver, 2001: 401-406.

[8] Alexander Hinneburg, Daniel A Keim. A general approach to clustering in large databases with noise[J]. Knowledge and Information Systems, 2003, 5(4): 387-415.

[9] Nene S A, Nayar S K, Murase H. Columbia object image library (coil-100) [R]. New York: Columbia University, 1996.


(上接第 700 页)

## 参考文献( References)


[1] Goldberg D E. Genetic algorithms in search, optimization and machine learning, reading[M]. MA: Addison-Wesley, 1989.

[2] Emily G, Jessie B. Parallel genetic algorithms: An exploration of weather prediction through clustered computing[J]. J of Computing Sciences in Colleges, 2003, 18(5): 272-273.

[3] Enrique A, Francisco L, Antonio J N. Parallel heterogeneous genetic algorithms for continuous optimization[J]. Parallel Computing, 2004, 30(5): 699-719.

[4] 刘立芳, 霍红卫, 王宝树. PHGA-COFFEE:多序列比对问题的并行混合遗传算法求解[J]. 计算机学报, 2006, 29(5): 727-733.
(Liu L F, Huo H W, Wang B S. PHGA-COFFEE: Aligning multiple sequences by parallel hybrid genetic algorithm[J]. J of Computers, 2006, 29(5): 727-733.)

[5] Kohlmorgen U, Schmeck H, Haase K. Experiences with fine-grained parallel genetic algorithms[J]. Annals of Operations Research, 1999, 90: 203-219.

[6] Martin P, Prasanna P, Arun R. Fine-grained parallel genetic algorithms in Charm + + [J]. ACM Crossroads Magazine: Parallel Computing, 2002, 8(3).

[7] Fernando G L, Claudio F, Hugo M. Massive parallelization of the compact genetic algorithm [C]. Proc of the Int Conf on Adaptive and Natural Computing Algorithms. Coimbra, 2005: 530-533.

[8] Jowens J D, Luebke D, Govindaraju N. A survey of general purpose computation on graphics hardware[C]. Euro-Graphics 2005. Dublin, 2005: 21-51.

[9] Fok K L, Wong T T, Wong M L. Evolutionary computing on consumer-level graphics hardware [J]. IEEE Intelligent Systems, 2005, 22(2): 69-78.

[10] Qizhi Yu, Chongcheng Chen, Zhigeng Pan. Parallel genetic algorithms on programmable graphics hardware [J]. Lecture Notes in Computer Science, 2005, 36(12): 1051-1059.

[11] 吴恩华. 图形处理器用于通用计算的技术现状及其挑战[J]. 软件学报, 2004, 15(10): 1493-1504.
(Wu E H. State of the art and future challenge on general purpose computation by graphics processing unit[J]. J of Software, 2004, 15(10): 1493-1504.)