一种基于CUDA的并行多目标进化算法

胡宾宾[1,2], 祁荣宾[1,2], 钱锋[1,2]

(1. 华东理工大学化工过程先进控制与优化技术教育部重点实验室，上海，200237;
2. 华东理工大学信息科学与工程学院，上海，200237)

摘要：传统的多目标进化算法多是基于Pareto最优概念的类随机搜索算法，求解速度较慢，特别是当问题维度变高，需要群体规模较大时，上述问题更加凸显。这一问题已经获得越来越多研究人员以及从业人员的关注。实验仿真中可以发现，构造非支配集和保持群体多样性这两部分工作占用了算法99%以上的执行时间。解决上述问题的一个有效方法就是对这一部分算法进行并行化改造。本文提出了一种基于CUDA平台的并行化解决方案，采用小生境技术实现共享适应度来维持候选解集的多样性，将多目标进化算法的实现全部置于GPU端，区别于以往研究中非支配排序的部分工作以及群体多样性保持的全部工作仍在CPU上执行。通过对ZDT系列函数的仿真结果，可以看出本文算法性能远远优于NSGA-II和NPGA。最后通过求解油品调和过程这一有约束多目标优化问题，可以看出在解决化工应用中的有约束多目标优化问题时，该算法依然表现出优异的加速效果。

关键词：多目标；进化算法；CUDA；GPU；并行计算

中图分类号：TE09; X24 **文献标识码**：A **文章编号**：1001-4160(2015)01-1-8

DOI: 10.11719/com.app.chem20150101

1 引言

现实生活中大多数优化和决策问题都是由多个目标组成的，如生产调度、人工智能、组合优化、工程设计、大规模数据处理等等，而这些目标之间往往是相互冲突的，这类复杂的问题就是所谓的多目标优化问题(MOP)。由此，用于求解多目标优化问题的多目标进化算法(multi-objective evolutionary algorithm, MOEA)获得越来越多研究人员以及从业人员的关注，在求解MOP过程中产生了很多经典的多目标进化算法。如2002年Deb[1]提出的带精英策略的非支配排序遗传算法NSGA-II，该算法引入了拥挤距离概念和精英保留机制，是迄今为止最优秀的多目标进化算法之一。从2003年至今，进化多目标优化前沿领域的研究呈现出新的特点[2-5]，为了更有效地求解高维多目标优化问题，一些区别于传统Pareto占优的新型占优机制相继涌现。

然而进化算法作为一种类随机搜索算法，其收敛速度较慢的本质不会发生改变。以NSGA-II算法为例，该算法时间复杂度为$O(mN^2)$(其中m为目标个数，N为群体规模)，当种群规模变大时这一问题更加凸显。这是因为种群个体的适应度评估函数计算必不可少，非支配排序以及多样性评估函数更加耗时(实验仿真中可以发现，两者占整个算法运行时间超过99%)。而增大种群规模是提高算法搜索精度，算法鲁棒性以及种群多样性的重要手段，因此对上述问题进行并行化求解是一种非常有效的提高算法效率方法。

近年来，GPU发展迅速，在数据并行处理能力和存储器带宽上优于CPU，其性能逐步向通用计算领域拓展。CUDA是一种将GPU作为数据并行计算设备的软硬件体系，目前已广泛应用于科学计算、生物计算、图像处理、动力学仿真、流体力学模拟、石油勘探领域，并在很多应用中取得了不同数量级的加速比。因此，将GPU以及CUDA技术应用于现代优化算法，尤其是应用于现代优化算法求解多目标问题的实现中，成为提高多目标算法求解速度和效率的一种关键技术。

本文提出的基于CUDA平台的多目标进化算法，在进化操作中引入算术交叉算子和多项式变异算子，并采用小生境技术实现共享适应度来维持候选解集的多样性。通过对ZDT系列函数的仿真结果，可以看出本文算法对于解决无约束的多目标优化问题有较快的求解速度。本文最后通过本算法解决油品调和过程这一有约束多目标优化问题。通过与NSGA-II，NPGA的比较可以看出，本文提出的算法对于解决化工应用中的有约束多目标优化问题，也表现出很好的加速效果。

2 概述

2.1 多目标优化问题[6]

多目标优化问题（MOP)：以最小化问题为例，传统的MOP问题表示如下：

$$\begin{aligned} &\min F(X)=\left[f_1(X),f_2(X),\cdots,f_m(X)\right] \\ &X=(x_1,x_2,\cdots,x_n), \\ &x_{j\min}\le x_j\le x_{j\max}, j=1,2,\cdots,n \end{aligned} \tag{1}$$

其中，$F(X)\in R^m$是m维的目标向量，$f_i(X)(i=1,2,\cdots,m)$是第i个目标函数；X为n维的可行

收稿日期：2014-09-28；**修回日期**：2014-12-10

基金项目：国家自然科学基金项目(U1162202, 61222303)；上海市自然科学基金(15ZR1408900)；
上海市“科技创新行动计划”研发平台建设项目(13DZ2295300)和上海市重点学科建设项目(B504)资助

作者简介：胡宾宾(1988—)，男，硕士，研究生，E-mail: hubinbinheda@126.com

联系人：祁荣宾，女，博士，副研究员. E-mail: qirongbin@ecust.edu.cn

决策向量，$x_j(j=1,2,\cdots n)$ 是 x 的第 j 个决策变量，$x_{j\min}, x_{j\max}$ 分别是其最小值与最大值。

设 X^1, X^2 是式(1)中 2 个可行解，则称 X^1 是较 X^2 Pareto 占优的，或称 X^1 支配 X^2，记为 $X^1 \succ X^2$，当且仅当：

$$\forall i \in \{1,2,\cdots,m\}: f_i(X^1) \le f_i(X^2) \wedge \exists j: f_j(X^1) < f_j(X^2) \quad (2)$$

其中 $j \in \{1,2,\cdots,m\}$。设 X 是式(1)中一个可行解，Ω 为其可行决策空间，称 X 为 Pareto 最优解或非支配解，当且仅当：

$$\nexists X^i \in \Omega, s.t. X^i \succ X \quad (3)$$

那么可以定义所有的 Pareto 最优解组成的集合称为 Pareto 最优解集。Pareto 最优解集中的解对应的目标向量组成的集合称为 Pareto 前沿。

2.2 多目标进化算法

多目标进化算法的基础是进化算法，它的处理对象则是多目标优化问题(multi-objective problem, MOP)。以基于 Pareto 的多目标进化算法为例，如图 1 所示。

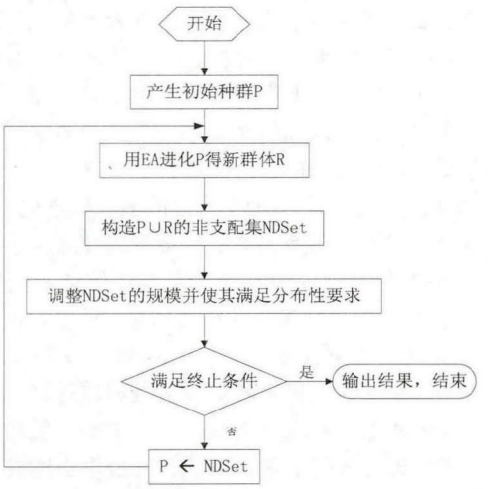

Fig.1 The basic framework of one class MOEA.
图 1 一类 MOEA 基本的框架

首先产生一个初始化种群 P，接着选择某个进化算法(如遗传算法)对 P 执行进化操作(如选择，交叉和变异)，得到新的进化群体 R。然后采用某种策略构造 $P \cup R$ 的非支配集 *NDSet*，一般情况下在设计算法时已设置了非支配集的大小(如 N)，若当前非支配集 *NDSet* 的大小大于或小于 N 时，需要按照某种策略对 *NDSet* 进行调整，一方面使 *NDSet* 满足大小要求，同时也必须使 *NDSet* 满足分布性要求。之后判断是否满足终止条件，若满足终止条件，则结束，否则将 *NDSet* 中的个体复制到 P 中并继续下一轮进化。

在 MOEA 中，保留上一代非支配集，并使之加入新一代的多目标进化操作时非常重要的，这类似于在进化算法中保留上一代的最优个体，从而使新一代的非支配集不比上一代差，这是算法收敛的必要条件。这样一代一代地进化下去，进化群体的非支配集不断地逼近真正的最优边界(true Pareto optimal front)，最终得到满意的解集(不一定是最优解)。

MOEA 种类很多，解决群体多样性的方法同样很多，且都取得了不错的效果。其中包括适应度共享(fitness sharing)如 NPGA，聚类(clustering)如 SPEA、SPEA2，拥挤距离(crowding distance)如 NSGA、NSGA-II，拥挤计数(crowding count)如 PESA、PESA-II、DMOEA 等等[7]。

2.3 图形处理器(GPU)和 CUDA 平台

与中央处理器（Central Processing Unit，CPU）传统的数据处理流水线相比，在图形处理器（Graphics Processing Unit，GPU）上执行通用计算还是一个新概念。基于 GPU 的通用计算指的是利用图形卡来实现一般意义上的计算[8]，而不单纯是图形的绘制。利用 GPU 来实现矢量、矩阵的基本代数运算，以及在这个基础上实现一些相对复杂的运算（如线性方程组的求解），从而实现复杂应用问题的求解。

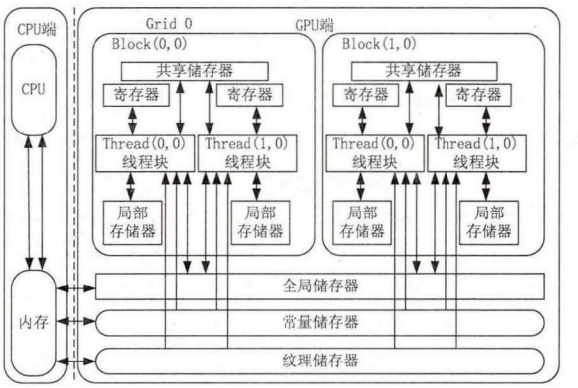

Fig.2 Multi-level storage model of GPU.
图 2 GPU 多级储存器模型

GPU 由许多晶体管组成，晶体管主要用于数据处理，而非数据缓存和流控制，这种结构专为计算密集型、高度并行化的计算而设计，因此在 CUDA 编程模型中 GPU 作为一个协处理器能产生大量的线程，并可以通过大量线程的并行计算来隐藏存储器访问延迟，而不必使用较大的数据缓存[9]。在 GPU 多层存储器架构下(见图 2)，显示芯片执行计算时的最小单位是线程(thread)，每个线程拥有一个私有的寄存器，多个线程可以组成一个

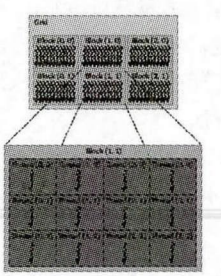

Fig.3 Grid of thread blocks.
图 3 网格内的线程块

线程块(block)。实际运行中，线程块会被分割为更小的线程束(warp)，block 内的所有 thread 可对块内的共享内存(shared memory)进行存取访问，多处理器内的所有 block 组成一个计算格 Grid(如图 3 所示)。GPU 中每个单线程 thread 执行的指令程序成为 kernel 函数，不同的 kernel 函数间串行执行，kernel 内部线程独立并行执行。

CUDA 采用了单指令多线程(Single Instruction Multiple Thread)执行模型，图 4 给出了基于 GPU-CUDA 的并行编程模型，该模型可归纳为 6 个步骤：[10]

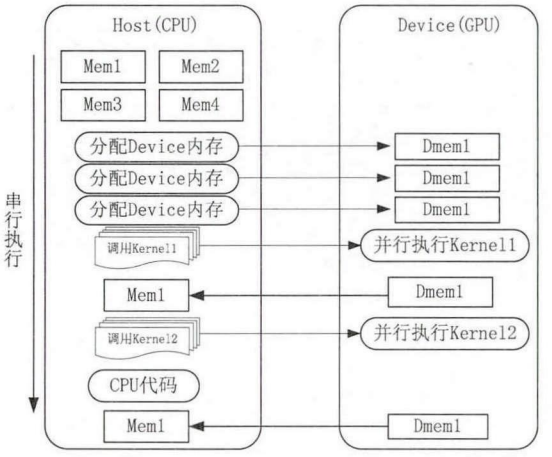

Fig.4 Parallel programming model of CUDA.
图 4 GPU-CUDA 并行编程模型

(1) 数据类型初始化：变量的定义与声明。例如 float *a_h(主机端变量)，float *a_d(设备端变量)；

(2) 存储空间分配：为 CPU 端和 GPU 端变量分别分配存储空间，用来存储不同类型的数据：

主机端：

a_h=(float*)malloc(N*sizeof(float));

设备端：

cudaMalloc((void* *)&a_d),N*sizeof(float));

(3) 数据传递：从主机端将数据传输到 GPU 端：

cudaMemcpy(a_d,a_h,Size,cudaMemcpyHostToDevice);

(4) 并行执行：调用 kernel 函数，并分配 GPU 端执行参数和函数参数，该函数将会被 GPU 内分配到的所有线程分别执行 1 次，从而实现单指令多数据的并行处理过程：

_global_void kernel<<<GridDim,BlockDim>>>(float*xx，float*yy，float*zz);

(5) 结果返回：将 GPU 端计算的结果传回 CPU 端：cudaMemcpy(b_h,b_d,Size,cudaMemcpyDeviceToHost);

(6) 释放显存：在 GPU 端的全局存储器中回收空间，通过调用函数 cudaFree()实现。

3 基于 CUDA 的多目标遗传算法实现

在多目标优化领域，为了提高多目标优化算法的求解性能，有不同的学者提出各种各样的解决方法，例如文献[11]中提出的并行多目标粒子群优化算法，文献[12]中提出的并行多目标局部搜索算法，文献[13]中提出的并行标签设置多目标优化算法，文献[14]中提出的并行多家族遗传算法，文献[15]中提出的基于多种群进化算法的多目标并行博弈设计，文献[16]中提出的伪并行 NSGA-II 算法，文献[17]中提出的基于 GPU 的多目标粒子群优化算法，文献[7]中提出的基于 CUDA 平台的多目标进化算法。这些并行优化算法均在不同程度上提高了多目标优化算法的求解性能，为本文研究提供了依据。纵观算法性能优化的历史，可以发现从单纯的算法改进(如文献[11]~[15])到算法改进与硬件发展相结合(如文献[7]，[16]，[17])是当下多目标优化算法性能优化的趋势。

本文提出一种基于 CUDA 平台的多目标遗传算法，类比于经典的多目标优化算法，提出了新的并行非支配集构造方法，以及使用适应度共享的并行实现来保证群体多样性。从而加速算法的收敛速度，大大提高算法的运行效率。下面几节将会介绍并行 MOEA 算法的具体实现方案。

3.1 数据结构

在以往的算法中，通常利用一个染色体使用一个线程进行遗传算法操作来实现 MOEA 算法的并行化。本文中的不同之处在于我们使用更多的线程来实现对染色体的操作，这样能更加有效的利用 GPU 资源。与此对应的就需要相匹配数据结构，本节就是描述将要使用的数据结构以及线程分配结构。

种群数据以二维矩阵的形态储存在 GPU 的全局内存上，其大小为 $N \times L$(N 为群体规模，L 为染色体长度)，其中列对应的是指染色体，行对应的是染色体中的基因，如图 5 所示。

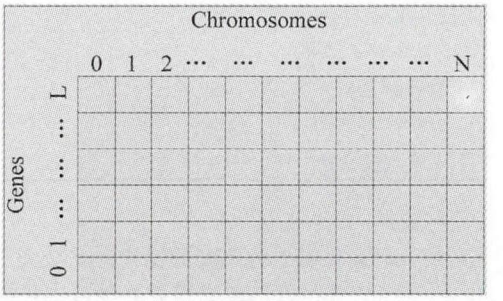

Fig.5 Population Matrix in memory.
图 5 内存中的种群数据矩阵

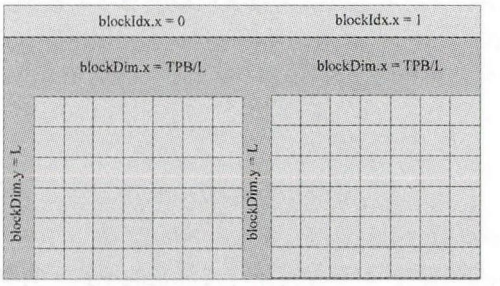

Fig.6 Thread layout.
图 6 线程布局

与此相对应的线程块内的线程也被设计为二维矩阵，如图 6 所示。其中 CUDA 核函数启动参数 blockDim.x 由 TPB(threads per block)控制。这样的设计使得线程与基

因的索引一一对应，染色体的所有基因可以被同时访问[18]。

3.2 适应度评价

适应度评价是指种群(包含父代和子代)中所有个体适应度评价函数计算。每个 CUDA 线程利用个体的决策变量通过求解多目标函数返回目标向量，这个评估过程通常耗时明显。因为不需要线程之间的交互，评估的过程是完全可以并行的。结合上文中数据结构一节可以知道，个体变量分解并存储在全局内存上。评价内核在对用内存区间上得到相应的变量对目标函数进行评估，每个线程生成包含 m 个评估值的目标向量，最后将目标向量输出并保存在大小为 $m \times N$ 的数组内。

3.3 父代选择

遗传算法的执行开始于父代的选择，在这一过程中被选中的个体将作为父代用来产生子代个体。常用的父代筛选方法有很多，比如轮盘赌选择、截断选择、随机竞争选择等等。相比之下随机竞争选择更加适用于我们的并行 MOEA 算法，因为在 GPU 上通过并行的方法来收集整个群体的统计信息是不切实际的，而随机竞争选择并不需要这些信息。因此随机竞争选择更加适合在 GPU 上使用。Pseudo-code 1 中描述了二元锦标赛选择的实现。

Pseudo-code 1 Tournament Selection

```
N ← Popsize
numThreads ← N/2
{For all threads in parallel }
i ← ThreadIdx.x
parent1 ← random(0,N-1)
parent2 ← random(0,N-1)
parent(i) ← choice (parent1, parent2)
```

3.4 交叉与变异

交叉算子是遗传操作中重要的操作，在交叉过程中优秀个体的基因模式得以迅速繁殖并在种群中扩散，使种群中其他个体向最优解的方向进行。适用于并行进化算法且和锦标赛选择紧密结合的交叉算子有模拟二进制交叉(SBX)算子、算术交叉算子等。本文采用算术交叉算子[19]实现种群的交叉操作。设 x_i^t 、 x_j^t 为第 t 代进化个体决策变量的实数编码，则交叉后个体相应的决策变量值为：

$$x_i^{t+1} = \alpha x_i^t + (1.0-\alpha) x_j^t \tag{4}$$

其中 α 为[-0.5,1.5]上均匀分布的随机数。将 α 不仅仅局限于[0,1]区间内，可以保证该交叉算子的搜索区间覆盖 x_i^t 和 x_j^t 的所有邻域，且两者之间的区域搜索几率较大。算术交叉算子具有较好的全局搜索能力，能更好的保持种群的多样性。

变异算子同样是遗传操作中重要的操作，因其自身具有的并行性，非常适合在 GPU 上运行。本文采用多项式变异算子[20]实现变异操作：

$$x^{(t+1)} = x^{(t)} + b(x_{\max} - x_{\min}) \tag{5}$$

其中 $x_{\max}$ 、 $x_{\min}$ 分别是个体决策变量的上下限，b为多项式变异系数：

$$b = \begin{cases} (2u)^{\frac{1}{\eta+1}} - 1 & \mu < 0.5 \\ 1 - [2(1-u)]^{\frac{1}{\eta+1}} & \mu \geq 0.5 \end{cases} \tag{6}$$

其中，η为变异分布指数。Pseudo-code 2 中描述了算术交叉算子与多项式变异算子的实现。

Pseudo-code 2 Arithmetic Crossover and Polynomial Mutation

```
N ← Popsize
numThreads ← N
{For all threads in parallel }
i ← ThreadIdx.x
if rand() < 0.9
  rand1 ← random(0, N/2-1)
  rand2 ← random(0, N/2-1)
  a ← rand(-0.5,1.5)
  Pop(i) ← a* parent(rand1)+(1.0-a)* parent(rand2)
else
  rand1 ← random(0, N/2-1)
  u ← rand(0,1)
  if u<0.5
    b ← pow(2*u,1/(η+1))-1
  else
    b ← 1-pow(2*(1-u),1/(η+1))
  Pop(i) ← parent(rand1)+b* (xmax-xmin)
```

3.5 非支配集的构造

构造非支配集是非支配遗传算法实现的基础，但非支配集的构造过程又是非常耗时的。从我们的实验中，99 %以上的执行时间被用于执行非支配排序和拥挤距离计算（以 NSGA2 算法为例）这两个程序。因此实现非支配遗传算法的并行化，非支配排序是首要考虑采用并行化处理的部分。Pseudo-code 3 中描述了构造非支配集的并行实现。

Pseudo-code 3 The Dominance-checking Algorithm

```
N ← Popsize
numThreads ← N*N
{For all threads in parallel }
i ← threadIdx.x + blockIdx.x * blockDim.x
j ← threadIdx.y + blockIdx.y * blockDim.y
offset ← i + j * blockDim.x * gridDim.x;
ido_res ← 0
int iequ_res ←0
if fitness[i] doman fitness[j]
  ido_res ← 1
do_res[offset] ← ido_res
```

3.6 解群体多样性

NSGA-II 中使用拥挤距离保持群体多样性的方式并不适合并行化计算，因此有的研究者将这一部分计算放

在 CPU 上执行[7]，本文选用适应度共享(fitness sharing)的方法来保持种群的多样性，而共享适应度的计算过程非常适合并行化处理。适应度共享是实现群体适应度的有效方法[6]。设个体 i 针对某个子目标的适应度为 $fitness(i)$，个体 i 的小生境计数(niche count)为 m_i，m_i 的计算方法如下：

$$m_i = \sum_{j \in Pop} sh[d(i,j)] \tag{7}$$

其中 Pop 为当前进化，$d(i,j)$ 为个体 i 和 j 之间的距离或称相似程度，$sh[d]$ 为共享函数，$sh[d]$ 的定义如下：

$$sh[d] = \begin{cases} 0 & d > \sigma_{share} \\ 1 - d / \sigma_{share} & d < \sigma_{share} \end{cases} \tag{8}$$

其中 σ_{share} 为小生境半径(niche radius)，通常由用户根据最优解集中个体之间的最小期望间距来确定。本文的 σ_{share} 取所有个体两两之间距离平均值的二分之一。

定义 $fitness(i)/m_i$ 为共享适应度(sharing fitness)，此处实质上 m_i 就是个体 i 的小生境聚集度，同一小生境内的个体相互降低对方的共享适应度。个体的聚集程度越高，其相对的共享适应度就被降低的越多。如图 7 所示，候选解 A 和 B 都是非支配的，但 A 的聚集密度比 B 大，因此 A 的共享适应度比 B 小，故而在 A 和 B 两个候选解中选择 B。使用共享适应度的目的在于进化群体分散到整个搜索空间的不同峰值区域上。

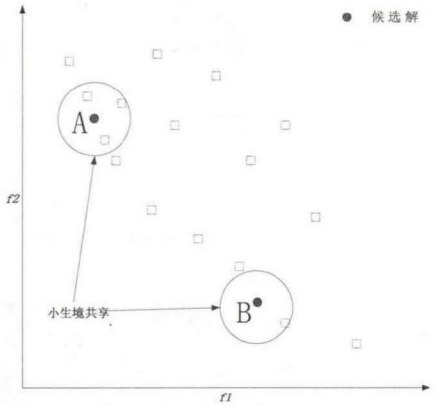

Fig.7 Niche sharing.
图 7 小生境共享

Pseudo-code 4 Fitness Distance

```
N ← Popsize
numThreads ← N*N
{For all threads in parallel }
i ← threadIdx.x + blockIdx.x * blockDim.x
j ← threadIdx.y + blockIdx.y * blockDim.y
offset ← i + j * blockDim.x * gridDim.x;
dis_temp=getDistance(&fitness[i * FitDim],& fitness[j * FitDim]);
dis[offset] ← dis_temp;
```

总之，整个 MOEA 程序都将在 GPU 上执行，这样，本文中提出的并行 MOEA 算法将在 SIMT(Single Instruction Multiple Threads) 架构的 GPU 上取得最大的加速增益。Pseudo-code 4、5 中描述了共享适应度的实现。首先 Pseudo-code 4 计算出个体两两之间的欧式距离，通过归约运算求得群体个体之间的平均距离，以实现小生境半径随着群体进化不断变化，然后再通过 Pseudo-code 5 解得个体共享适应度，再由归约运算进行求和操作。CUDA 中对数组进行求和操作都可以用归约操作实现，可以大大提高计算效率。

Pseudo-code 5 Sharing Fitness

```
N ← Popsize
numThreads ← N*N
{For all threads in parallel }
i ← threadIdx.x + blockIdx.x * blockDim.x
j ← threadIdx.y + blockIdx.y * blockDim.y
offset ← i + j * blockDim.x * gridDim.x;
sh_dis=0;
if(x!=y)
    dis_temp=dis[offset];
    if(dis_temp < sh_par)
        sh_dis=1-__fdividef(dis_temp,sh_par);
dis[offset] ← sh_dis;
```

4 算法实验仿真

本文比较了 CPU 和 GPU 2 个平台在实现 MOEA 解决多目标问题的平均耗时。文中数据为进行 20 次试验的平均性能。实验测试平台分别为 Intel Core i3 CPU M380 和 NVIDIA Geforce GT 540M。CPU 的频率是 2.53GHz，GPU 包含 96 个 CUDA 处理器核心，操作系统为 Windows 7 专业版，开发环境为 Microsoft Visual Studio 2010 和 NVIDIA CUDA 5.5。实验参数如下：

(1) 种群规模：1024,2048,4096
(2) 锦标赛系数：q = 2
(3) 最大代数：G = 100

4.1 标准测试函数模型

本文首先以无约束多目标优化问题测试函数集 ZDT 系列为例比较本文提出的基于 CUDA 平台的并行 MOEA 实现方案与 CPU 平台下 NSGA-II 以及同样采用共享适应度的 NPGA[21]效率之间差异。测试函数集 ZDT1，ZTD2，ZTD3，ZTD6[22]如表 1 所示，其中 ZDT1 的 Pareto 最优前沿是凸的；ZDT2 和 ZDT6 的 Pareto 最优前沿是凹的；ZDT3 的前沿面是非连续的。测试所得平均耗时以及加速比如表 2~表 5 所示，其中 NPGA 同样采用个体平均距离的一半作为小生境半径；时间单位均为秒；加速比为算法在 CPU 上所耗时间除以 GPU 上所耗时间得到。

通过表 2~表 5 可以发现，CUDA 平台相对于 CPU 平台，MOEA 算法的计算效率大大提高。随着群体规模的不断变大，与之对应的 MOEA 无论是在 CUDA 平台还是在 CPU 平台上算法的时间开销均会增加，但相对于 CPU 平台的时间消耗无论是 NSGA-II 还是 NPGA，并行 MOEA 都保持了较高的加速效果，从 17.4 到 31.2 不等。相对于文献[7]中针对 NSGA-II 算法的并行实现本文的解

决方案保有较大的优势，省去了数据在 CPU 内存与 GPU 内存之间的频繁复制，提高了算法的效率，这一点在群体规模增大时将更加明显。

表 1 无约束问题测试函数 ZDT 系列
Table 1 Unconstrained test problems ZDT.

Name	Objective functions	g(.)function/Optimal solutions	Variable bounds
ZDT1	$f_1(X)=x_1$ $f_2(X)=g(X)[1-\sqrt{\frac{x_1}{g(X)}}]$	$g(x)=1+\frac{9}{30-1}(\sum_{i=2}^{30}x_i^2)$ $x_1\in[0.01,1], x_i=0\forall i\neq 1$	$[0.01,1]\times[-1,1]^{29}$
ZDT2	$f_1(X)=x_1$ $f_2(X)=g(X)[1-(\frac{x_1}{g(X)})^2]$	$g(x)=1+\frac{9}{30-1}(\sum_{i=2}^{30}x_i^2)$ $x_1\in[0.01,1], x_i=0\forall i\neq 1$	$[0.01,1]\times[-1,1]^{29}$
ZDT3	$f_1(X)=x_1$ $f_2(X)=g(X)[1-\sqrt{\frac{x_1}{g(X)}}-\frac{x_1}{g(X)}\sin(10\pi x_1)]$	$g(x)=1+\frac{9}{30-1}\sum_{i=2}^{30}(x_i^2)$ $x_1\in[0.01,1], x_i=0\forall i\neq 1$	$[0.01,1]\times[-1,1]^{29}$
ZDT6	$f_1(X)=1-\exp(-4x_1)\sin^6(4\pi x_1)$ $f_2(X)=g(X)[1-(\frac{f_1(X)}{g(X)})^2]$	$g(x)=1+9(\frac{\sum_{i=2}^{10}(x_i^2)}{10-1})^{0.25}$ $x_1\in[0.01,1], x_i=0\forall i\neq 1$	$[0.01,1]\times[-1,1]^{9}$

表 2 ZDT1 实验数据
Table 2 Experimental data of ZDT1.

Population Size	CPU(NSGA-II)	CPU(NPGA)	GPU	Speed up (NSGA-II)	Speed up (NPGA)
1024	38.13	33.241	1.616	23.6	20.6
2048	150.238	132.107	5.487	27.4	24.1
4096	600.891	544.516	19.717	30.5	27.6

表 3 ZDT2 实验数据
Table 3 Experimental data of ZDT2.

Population Size	CPU(NSGA-II)	CPU(NPGA)	GPU	Speed up (NSGA-II)	Speed up (NPGA)
1024	37.354	31.232	1.799	20.7	17.4
2048	142.619	124.227	5.717	24.9	21.7
4096	570.674	510.278	20.906	27.3	24.4

表 4 ZDT3 实验数据
Table 4 Experimental data of ZDT3.

Population Size	CPU(NSGA-II)	CPU(NPGA)	GPU	Speed up (NSGA-II)	Speed up (NPGA)
1024	37.71	33.616	1.754	21.5	19.2
2048	146.939	133.197	5.564	26.4	23.9
4096	616.045	560.056	19.739	31.2	28.4

表 5 ZDT6 实验数据
Table 5 Experimental data of ZDT6.

Population Size	CPU(NSGA-II)	CPU(NPGA)	GPU	Speed up (NSGA-II)	Speed up (NPGA)
1024	34.234	30.022	1.676	20.4	17.9
2048	131.755	120.309	5.826	22.6	20.7
4096	519.933	480.713	21.392	24.3	22.5

4.2 油品调和过程多目标优化

汽油的调和过程因调和成本、成品油产值、质量等优化目标存在一定的矛盾关系，很难同时达到最优，是一个典型的多目标优化问题。又因为调和过程中需要考虑各种严格的限制条件，以方便得到符合规定的成品油，因此油品调和过程实际上是一个很典型的约束多目标优化问题。

4.2.1 调和工艺及其优化模型[23]

以 93#汽油调和工艺为例，目前比较常见的汽油调和工艺包含 7 种组分油包，它们分别是一催化汽油(x_1)、二催化汽油(x_2)、非芳烃汽油(x_3)、加氢/直硫油(x_4)、合成甲基叔丁基醚 MTBE(x_5)、C9(x_6)以及重整生成油(x_7)，另外，产品中加入了抗氧化剂、抗静电剂以及 MMT 等添加剂。本文采用调和成本(cost)和成品油辛烷指标值过剩量(ron)为 2 个优化目标函数，同时选择上述 7 种组分油百分比含量作为操作优化参数，则调和成本定义为：

$$\cos t = f_1(x) = \sum_{i=1}^{K} P_i x_i \tag{9}$$

其中，P_i表示第 i 中组分油的价格。

成品油辛烷指标值过剩量计算为：

$$ron = f_2(x) = (P_{RON} - IndRon) \tag{10}$$

其中 P_{RON} 为采用研究法辛烷值模型预测得到的辛烷值，$IndRon$ 为成品油的研究法辛烷值指标。

在实际工业过程中，油品调和过程必须满足物料守恒原理，同时产品质量属性指标也是严格的要求，故这里用物料守恒以及成品油质量性能指标作为调和过程的

约束条件，所以取约束条件为：

(1) 预测研究法辛烷值：

$$g_1(X)=P_{RON}\geq IndRon \tag{11}$$

(2) 预测马达法辛烷值：

$$g_2(X)=P_{MON}\geq IndMON \tag{12}$$

(3) 抗爆性能指数

$$g_3(X)=0.5*(g_1(X)+g_2(X))\geq 88.1 \tag{13}$$

(4) 雷德蒸汽压指标(kPa)：

$$g_4(X)=RVP\leq 74 \tag{14}$$

(5) 预测芳烃含量%($v\cdot v^{-1}$)：

$$g_5(X)=C1\leq 40 \tag{15}$$

(6) 预测苯含量%($v\cdot v^{-1}$)：

$$g_6(X)=C2\leq 2.3 \tag{16}$$

(7) 预测密度($Kg\cdot(m^3)^{-1}$)：

$$g_7(X)=\rho\leq 0.743 \tag{17}$$

(8) 预测烯烃含量%($v\cdot v^{-1}$)：

$$g_8(X)=C3\leq 34 \tag{18}$$

(9) 预测硫含量%($v\cdot v^{-1}$)：

$$g_9(X)=C5\leq 50 \tag{19}$$

(10) 物料守恒：

$$h_1(X)=\sum_{i=1}^{K}x_i-1=0 \tag{20}$$

4.2.2 实验仿真设置及结果分析

以某炼油厂调和车间为例调和 93#汽油，采用实际历史数据根据前面建立优化模型如下：

$$\begin{aligned}\min\ f_1&=\cos t=4200x_1+4300x_2+3900x_3\\&\quad+4200x_4+6000x_5+6500x_6+5000x_7\\\min\ f_2&=ron=92x_1+94x_2+78x_3+60x_4\\&\quad+108x_5+108x_6+95.7x_7-93.2\end{aligned} \tag{21}$$

其中约束条件如下：

$$\begin{aligned}g_1(X)&=92x_1+94x_2+78x_3+60x_4\\&\quad+108x_5+108x_6+95.7x_7\geq 93.2\\g_2(X)&=82x_1+83x_2+69x_3+54x_4\\&\quad+96x_5+96x_6+84.3x_7\geq 81\\g_3(X)&=(g_1(X)+g_2(X))/2\geq 88.1\\g_4(X)&=69x_1^{1.14}+72x_2^{1.14}+82.5x_3^{1.14}+110x_4^{1.14}\\&\quad+44x_5^{1.14}+7x_6^{1.14}+44x_7^{1.14}\leq 74\\g_5(X)&=20x_1+18x_2+1.5x_3+8.48x_4\\&\quad+98x_6+63.6x_7\leq 40\\g_6(X)&=0.6x_1+0.55x_2+0.2x_4\leq 2.3\\g_7(X)&=0.73x_1+0.75x_2+0.67x_3+0.73x_4\\&\quad+0.73x_5+0.86x_6+0.73x_7\leq 0.743\\g_8(X)&=40x_1+42x_2+5x_3+5x_4\\&\quad+5x_5+1.38x_7\leq 34\\g_9(X)&=(408.8x_1+270x_2+25*0.67x_3+16.75x_4\\&\quad+36.5x_5+4.3x_6+36.5x_7)/g_7(X)\leq 480\\h_1(X)&=x_1+x_2+x_3+x_4+x_5+x_6+x_7=1\\0.3\leq x_1\leq 0.6,&0.2\leq x_2\leq 0.5,0\leq x_7\leq 0.15,\\0\leq x_i\leq 0.1,&i=3,4,5,6\end{aligned} \tag{22}$$

通过分析可以发现，变量 x_7 与其他 6 个变量线性相关，故可用其他变量线性表示，同时将变量 x_7 的边界约束条件转换为 2 个不等式约束条件，所以有：

$$\begin{aligned}x_7&=1-(x_1+x_2+x_3+x_4+x_5+x_6)\\g_{10}(X)&=1-(x_1+x_2+x_3+x_4+x_5+x_6)\leq 0.15\\g_{11}(X)&=1-(x_1+x_2+x_3+x_4+x_5+x_6)\geq 0\end{aligned} \tag{23}$$

可以发现，虽然约束多目标优化问题式(21-23)的目标函数是线性函数，但是该问题包含的约束条件比较多，同时约束条件有线性函数也有非线性函数。为了得到更好的非支配前沿，增大群体规模是一种常用的方法，而当群体规模变大时，算法运行效率就变得重要起来。表 6 即为油品调和实例在实验中的性能表现。

表 6 汽油调和实验数据

Table 6 Experimental data of gasoline blending.

Population Size	CPU(NSGA-II)	CPU(NPGA)	GPU	Speed up(NSGA-II)	Speed up(NPGA)
1024	34.721	33.494	1.537	22.6	21.8
2048	132.737	135.713	5.373	24.7	25.3
4096	529.958	590.514	19.554	27.1	30.2

通过表 6 可以发现，本文提出的并行算法在有约束的多目标优化问题中依然保持着较好的性能。相对于 NSGA-II 以及 NPGA 在 CPU 平台上的时间消耗，并行 MOEA 都保持了较高的加速效果，从 21.8 到 30.2 不等，并随着种群规模的增大越来越优异。实验再次证明了本文算法具有优良的性能。

5 总结

本文基于 CUDA 平台实现了一种并行 MOEA，并在算法的进化操作中引入算术交叉算子和多项式变异算子。本文中的并行 MOEA 所有实现步骤均实现并行化编程并在 GPU 上执行，因此获得了满意的效率提升。

通过实验数据可以知道本文提出的并行 MOEA 解决方案相对与文献[7]中的方案更加高效，这是因为本文对 MOEA 所包含的各个部分均提出并行化解决方案，避免了主机端(CPU)与设备端(GPU)频繁的数据传输。

通过实验比较并行 MOEA 和传统 CPU 上实现的 MOEA，可以发现并行 MOEA 的加速效果十分明显，变化范围从 17.4 到 31.2。因此，可以知道并行 MOEA 在解决多目标优化问题，特别是问题所需种群规模巨大时将非常有用。

References:

1 Deb K, Pratap A, Agarwal S and Meyarivan T. "A fast and elitist multi-objective genetic algorithm: NSGA-II,". IEEE Trans Evol Comput, 2002, 6:182-197.

2 Laumanns M, Thiele L, Deb K, et al. Combining convergence and diversity in evolutionary multi-objective optimization. Evolutionary Computation, 2002, 10(3):263-282.
3 Brockoff D, Zitzler E. Are all objective necessary on dimensionality reduction in evolutionary multi-objective optimazation//Proceedings of 9th International Conference on Parallel Problem Solving from Nature. Berlin: Springer, 2006:533-542.
4 Hernandez-Diaz A G, Santana-Quintero L V, Coello Coello C A, el al. Pareto-adaptive ε-dominance. Evolutionary Computation, 2007, 15(4):533-542.
5 Deb K, Saxena D K. On finding Pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems, Technical Report 2005011. Kanpur: Indian Institute of Technology, 2005.
7 Wong M L. Parallel multi-objective evolutionary algorithms on graphics processing units//Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. ACM, 2009:2515-2522.
9 Chuang H H, Wu S J, Hong M Z, et al. Power integrity chip-package-PCB co-simulation for I/O interface of DDR3 high-speed memory//Advanced Packaging and Systems Symposium, 2008. EDAPS 2008. Electrical Design of. IEEE, 2008:31-34.
11 Soares J, Vale Z, Canizes B, et al. Multi-objective parallel particle swarm optimization for day-ahead Vehicle-to-Grid scheduling//Computational Intelligence Applications In Smart Grid (CIASG), 2013 IEEE Symposium on. IEEE, 2013:138-145.
12 Iturriaga S, Ruiz P, Nesmachnow S, et al. A Parallel Multi-objective Local Search for AEDB Protocol Tuning//Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, 2013:415-424.
13 Sanders P, Mandow L. Parallel Label-Setting Multi-objective Shortest Path Search//Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE, 2013: 215-224.
17 Zhou Y, Tan Y. GPU-based parallel particle swarm optimization//Evolutionary Computation, 2009. CEC'09. IEEE Congress on. IEEE, 2009:1493-1500.
18 Shah R, Narayanan P, Kothapalli K. GPU-accelerated genetic algorithms. Cvit Iiit Ac In, 2010.
20 Deb K, Goyal M. A combined genetic adaptive search (GeneAS) for engineering design. Computer Science and Informatics, 1996, 26:30-45.
21 Horn J, Nafpliotis N, Goldberg D E. A niched Pareto genetic algorithm for multiobjective optimization//Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. IEEE, 1994:82-87.
22 Zitzler E, Deb K and Thiele L. Comparison of multi-objective evolutionary algorithms: Empirical results. Evolutionary Computation, 2000, 8(2):173-195.

中文参考文献

6 金华. 多目标进化算法及其应用[M]. 北京:科学出版社, 2007.
8 王雷, 赵龙, 韩文报. 基于 GPU 平台的模乘算法实现[J]. 信息工程大学学报, 2010, 11(4):462-465.
10 张庆科, 杨波, 王琳, 等. 基于 GPU 的现代并行优化算法[J]. 计算机科学, 2012, 39(4):304-310.
14 卢海, 鄢烈祥, 史彬, 等. 并行多家族遗传算法解多目标优化问题[J]. 化工学报, 2012, 63(12):3985-3990.
15 谢能刚, 潘创业, 李锐, 等. 基于多种群进化算法的多目标并行博弈设计[J]. 数值计算与计算机应用, 2010, (2):81-91.
16 徐伟华, 凌晓波, 刘蓓, 等. 基于伪并行 NSGA-II 算法的火电站多目标负荷调度[J]. 上海交通大学学报, 2008, 42(3):421-425.
19 付立, 窦明罡, 朱建凯, 等. 非支配排序遗传算法的改进[J]. 计算机与数字工程, 2011, 39(2):11-15.
23 蚁仲杰, 祁荣宾, 徐斌, 等. 约束差分免疫克隆算法及其在汽油调合优化中的应用[J]. 华东理工大学学报: 自然科学版, 2013, 39(3):311-318.

A parallel multi-objective evolutionary algorithm based on CUDA

Hu Binbin[1,2], Qi Rongbin[1,2] and Qian Feng[1,2]

(1. Key Laboratory of Advanced Control and Optimization for Chemical Processes, Ministry of Education, Shanghai 200237, China)
(2. School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: Most of the basic multi-objective evolutionary algorithm (MOEA) is one kind of similar random search algorithm base on the concept of the Pareto Optimization, which with the slowly speed. Especially the dimension of the problem becomes larger which needs a larger population, the effectiveness of MOEA become more prominent. This problem has been gaining increasing attention among researchers and practitioners. From the experiments, the procedure to build the non-dominated set and to maintain diversity are time-consuming, more than 99 % of the execution time is used in performing the two procedures. A promising approach to overcome this limitation is to parallelize these algorithms. In this paper, we propose a parallel MOEA based on Compute Unified Device Architecture (CUDA). Fitness sharing which was based on niche technology is introduced to enhance population diversity. All steps of the parallel MOEA are performed on Graphics Processing Units (GPU). That's the difference between the previous studies which part of the procedure of building the non-dominated set and maintaining diversity is still executed on the CPU. Through the simulation of ZDT series test function and the analysis results with the NSGA-II and NPGA, it can be seen that the performance of the proposed algorithm is much better than the NSGA-II and NPGA with less run times. Finally, it is applied to constrained multi-objective optimization of gasoline blending, the algorithm shows better performance.

Keywords: Multi-objective; evolutionary algorithms; CUDA; GPU; parallel programming

(Received: 2014-09-28; Revised: 2014-12-10)