基于图形处理器的并行遗传过程挖掘

李龚亮[1,2]，敬思远[3]，郭 兵[1*]，沈 艳[4]

(1. 四川大学计算机学院 成都 610065；2. 中国工程物理研究院计算机应用研究所 四川 绵阳 621000；
3. 乐山师范学院计算机科学学院 四川 乐山 614000；4. 成都信息工程大学控制工程学院 成都 610225)

【摘要】为提高遗传过程挖掘算法对大规模事件日志处理的性能，该文提出一种基于GPU的并行遗传过程挖掘算法。由于传统基于二进制的染色体编码不能表示因果矩阵中的AND-Split/AND-Join和OR-Split/OR-Join结构，提出一种新的染色体编码方案。该方案通过内容、标识、位置3个数组，有效地解决了GPU上因果矩阵的遗传表示问题。同时，设计并实现了高效的遗传交叉/变异算子和适应度并行计算方法。仿真实验表明，与当前CPU上的遗传过程挖掘算法相比，本文算法在求解精度和收敛速度方面都具有明显优势，并且在两个数据集上分别取得36.4倍和47.2倍的执行时间加速比。

关 键 词 遗传算法； 图形处理器； Petri网； 过程挖掘； 工作流
中图分类号 TP311.1 **文献标志码** A doi:10.3969/j.issn.1001-0548.2019.06.017

Parallel Genetic Process Mining Based on Graphics Processing Unit

LI Gong-liang[1,2], JING Si-yuan[3], GUO Bin[1*], and SHEN Yan[4]

(1. School of Computer, Sichuan University Chengdu 610065;
2. Institute of Computing Applications, China Academy of Engineering Physics Mianyang Sichuan 621000;
3. School of Computer Science，Leshan Normal University Leshan Sichuan 614000;
4. School of Control Engineering, Chengdu University of Information Technology Chengdu 610225)

Abstract To improve the performance of genetic process mining algorithm for handling large scale event log, a GPU-based parallel genetic process mining algorithm is proposed. Since traditional binary chromosome coding method can not represent the AND-Split/AND-Join and the OR-Split/OR-Join structures in causal matrix, a new coding method of chromosome is proposed. The proposed method can effectively solve the problem of genetic representation of causal matrix on graphics processing units (GPU) by three arrays, which are content, labels and position. Meanwhile, the efficient genetic crossover/mutation operators and a parallel method of fitness value computation are designed and implemented. Simulation experiments show that the proposed algorithm, compared with the CPU-based genetic process mining algorithm, has obvious advantages in precision and convergence rate, and moreover it obtains speedup of 36.4 and 47.2 on two data sets respectively.

Key words genetic algorithm; graphics processing unit; Petri net; process mining; workflow

过程感知信息系统能够自动记录业务执行的实际过程，并生成事件日志。过程挖掘(也称为工作流挖掘)的任务，即通过分析事件日志，重现业务过程的真实执行过程[1]，并利用该知识实现过程分析和优化[2]、过程再造[3]和决策支撑[4]等。

根据文献[5]的划分，当前的过程挖掘算法可以分为3类，分别是以α算法[6]、α++算法[7]为代表的直接算法、以ILP Miner[8]为代表的启发式算法以及以遗传算法[9-10]为代表的计算智能方法。直接算法不能处理事件日志中的噪声，所以在实际应用中采用较少。启发式算法的优点是具有抗噪能力，并且搜索结果快，但是搜索结果往往会陷入局部最优。文献[9]首次将遗传算法应用到过程挖掘中，通过定义良好的适应度函数，以及交叉、变异等过程遗传操作算子，最终得到与事件日志非常一致的过程模型。而且，遗传过程挖掘用一种统一方式解决了非自由选择结构、不可见任务、以及同名任务的挖掘。文献[10]对上述方法进行了改进，提出了一种混合自适

收稿日期：2019－01－07；修回日期：2019－04－19
基金项目：国家自然科学基金重点项目(61332001)；国家自然科学基金(61772352，61472050)；四川省科技计划(2018ZDZX0010，2017GZDZX0003，2018JY0182)
作者简介：李龚亮(1982－)，男，博士生，高级工程师，主要从事过程挖掘、信息系统、企业大数据方面的研究.
通信作者：郭兵，E-mail: guobin@scu.edu.cn

应的遗传过程挖掘算法。总的来说，遗传过程挖掘算法有较好的抗噪能力，能够同时处理多种复杂的过程挖掘任务，具有非常好的应用价值，但缺点是运行时间过长。

目前过程挖掘领域面临的主要问题之一是对过程大数据的挖掘[11]。以医院信息管理系统为例，它包含了门诊信息管理系统、药品信息管理系统、住院信息管理系统等数十个系统，每天产生数百兆的事件日志，其中包含上千个任务。当前的过程挖掘算法很难对这样的过程大数据进行处理。文献[12]提出了一种分布式的遗传过程挖掘算法，但计算性能并不能令人满意。文献[13]提出了一种分而治之的方法来处理过程大数据。该方法的核心思想是通过分解任务集和事件日志，快速挖掘出局部的过程模型，最后将其合并得到完整的过程模型。遗憾的是，该方法并不适合遗传过程挖掘，因为分而治之的方法可能会极大地削弱遗传算法的全局搜索能力；此外，分解后的任务子集和事件日志规模难以确定，因此很难对遗传过程挖掘算法的参数进行预先设置。

GPU是一种性价比非常高、且便于使用的高性能计算平台，目前已经在图形图像处理、人工智能、科学计算等领域得到广泛应用。本文提出了一种基于GPU的并行遗传过程挖掘算法，以解决遗传过程挖掘算法运行时间长，且不能处理过程大数据的问题。本文的创新主要包括3个方面：1) 提出了一种因果矩阵在GPU上的染色体编码方法，解决了过程模型在GPU上的表示问题；2) 利用GPU高并行的特点，采用大量线程对过程模型进行搜索，提高了遗传过程挖掘算法的收敛速度和搜索精度；3) 充分考虑了GPU的计算原理，对遗传过程挖掘算法中计算量最大的3个操作：适应度计算、遗传交叉和遗传变异，进行了细致地并行化算法设计。

1 背景知识

1.1 过程挖掘的相关概念

本节简要回顾与本文相关的一些过程挖掘概念，更详细的内容可以参考文献[1,6,14]。在过程挖掘问题中，过程模型往往被建模为工作流网。工作流网属于一种特殊的Petri网，称为库所(Place)/变迁(Transition)网(P/T网)，其定义如下：

定义 1 (P/T网)[6]：P/T网表示为一个三元组 $N=(P,T,F)$，其中：

1) P 是库所的有穷集合；

2) T 是变迁的有穷集合，满足 $P\bigcap T=\varnothing$ 且 $P\bigcup T\neq\varnothing$；

3) $F=(P\times T)\bigcup(T\times P)$ 是有向弧的集合，称为流关系。

定义 2 (工作流网)[6]：令 $N=(P,T,F)$ 为P/T网，t' 为不属于 $P\bigcup T$ 的新结点，N 是工作流网当且仅当：

1) N 有两个特殊的库所 i 和 o，其中 i 称为开始库所，o 称为结束库所，即 $\bullet i=\varnothing$，$o\bullet=\varnothing$；

2) $N'=(P,\ T\bigcup\{t'\},\ F\bigcup\{(t',\ i),(o,\ t')\})$ 是强连通的。

定义2中的符号 $\bullet i$ 和 $o\bullet$ 分别表示结点 i 的前驱结点和结点 o 的后继结点。图1给出了一个采用工作流网表示的过程模型，其中圆圈表示库所，方框表示变迁(变迁对应了过程挖掘问题中的任务)，方框内的字母表示任务的编号。

过程挖掘的目标是从事件日志中发现过程模型，下面给出事件轨迹和事件日志的定义。

定义 3 (事件轨迹、事件日志)[6]：令 T 为任务集合，T^* 为由 T 中任务构成的有限序列的集合，$\sigma\in T^*$ 是一个事件轨迹，$W\subseteq T^*$ 称为一个事件日志。

需要说明的是，一个事件日志中包含了若干个事件轨迹，每一个事件轨迹对应业务过程中的一个实际的执行案例。以到医院门诊看病为例，网上预约挂号 $\rightarrow$ 现场取号 $\rightarrow$ 医生看病 $\rightarrow$ 化验 $\rightarrow$ 医生确诊开药 $\rightarrow$ 取药，就是一个案例，这个案例生成的所有实际业务执行的过程记录就是事件轨迹。图1的过程模型中包含了4个事件轨迹：{*ABH*，*ACH*，*ADEFGH*，*ADFEGH*}。

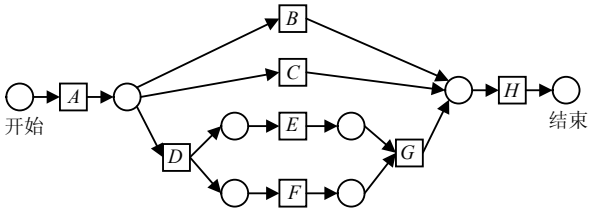

图1　过程模型示例

1.2 因果矩阵

遗传过程挖掘算法将过程模型表示为因果矩阵，通过遗传搜索，最终发现适应度最高的因果矩阵作为算法输出。因果矩阵可以按照一定的规则转换为P/T网。因果矩阵的定义如下。

定义 4 (因果矩阵)[9]：因果矩阵表示为一个四元组 $\mathrm{CM}=(T,C,I,O)$，其中：

1) T 表示所有任务的有穷集合；

2) $C\subseteq T\times T$ 是因果关系的有限集；

3) $I:T\rightarrow I(T)$ 为输入映射函数，$\forall t\in T$，

$I(t)=\{t'|(t',t)\in C\}$；

4) $O:T\to O(T)$ 为输出映射函数，$\forall t\in T$，$O(t)=\{t'|(t,t')\in C\}$。

图1中的过程模型，其对应的因果矩阵的集合表示如表1所示。其中 $I(A)=\{\}$ 表示任务 A 的输入为空集；$O(A)=\{\{B,C,D\}\}$ 表示任务 A 的输出为任务 B,C,D，并且 B,C,D 之间为选择关系。$O(D)=\{\{E\},\{F\}\}$ 表示 D 的后继为 E,F，但是 E,F 之间为并行关系。换句话说，子集内部的元素之间为选择关系，子集与子集之间为并行关系。

表1 因果矩阵

任务	$I(T)$	$O(T)$
A	{}	{{B, C, D}}
B	{{A}}	{{H}}
C	{{A}}	{{H}}
D	{{A}}	{{E}, {F}}
E	{{D}}	{{G}}
F	{{D}}	{{G}}
G	{{E}, {F}}	{{H}}
H	{{B, C, G}}	{}

2 基于CUDA的并行程序设计

本节先简要回顾CUDA编程模型，然后再介绍CUDA程序设计优化的几个关键点。

在CUDA编程模型中，程序被分为两个部分，分别是主机端(Host)和设备端(Device)[15]。Host端是指在CPU上执行的程序，Device端是指在GPU上执行的并行程序。Device端程序又被称为内核函数(kernel)。Nvidia GPU采用了一种层次化的方式对线程进行组织管理。首先，线程是指令执行的最小单位；32个线程组成一个线程束(warp)。warp是线程调度的最小单位。同一warp中的线程，采用单指令多线程(single instruction multiple threads, SIMT)的模式执行。其次，若干个warp被组织成一个线程块(block)。同一block中的线程会在相同的流多处理器上执行，因此同一block中的线程能够通过共享存储器(shared memory)进行通信和同步。最后，若干个block被组织成一个网格(grid)。CUDA程序每次启动kernel时都需要创建一个grid。

在进行CUDA程序设计优化时，有以下4点需要注意[16]：1) CUDA程序需要启动足够多的线程以隐藏指令延迟，以提升GPU的利用率；2) 共享内存的带宽远远高于全局内存，因此充分利用共享内存能够大幅提升CUDA程序的性能；3) 同一warp中线程的指令执行采用的是SIMT模式，因此要尽量避免分支；4) 由于GPU和主机内存之间进行数据传输依靠的是PCI-E总线，数据传输的效率非常低，因此在进行CUDA程序设计时要尽量减少数据传输。

3 基于GPU的并行遗传过程挖掘

3.1 算法框架

本文提出的并行遗传过程挖掘框架如下所示。

Host端算法流程如下：

1) 在CPU上初始化种群，并按照3.2节中介绍的方法对因果矩阵进行染色体编码。因果矩阵的随机初始化方法仍然采用文献[9]中的方法。

2) 将事件日志、初始化的种群、相关参数拷贝到GPU的全局内存；

3) 启动kernel进行并行遗传挖掘；

4) 等待Device端程序执行完毕。将当前最优染色体的适应度值拷贝回Host端，判断是否满足算法结束条件。如果满足，则跳转到步骤5)；否则跳转到步骤3)；

5) Host端算法结束。

Device端算法(kernel)流程如下：

1) 各线程从当前种群中随机选择一个个体；

2) 采用3.3节方法进行并行遗传交叉；

3) 采用3.4节方法进行并行遗传变异；

4) 采用3.5节方法对所有个体的适应度进行计算和排序，并保留其中的精英；

5) Device端算法结束。

3.2 因果矩阵在GPU中的染色体编码

文献[10]中采用二进制方式进行染色体编码，因果矩阵按行组合形成一个染色体。这种编码方案的缺陷是无法反映过程模型中的AND-Split/AND-Join和OR-Split/OR-Join关系。举例来说，假设因果矩阵某一行的值为[0 1 1 0 1]，5个位置分别对应任务A～E。此时算法将无法判定B，C，E之间是并行关系还是选择关系(如{{B, C, E}}，{{B, C}, {E}}，{{B}, {C}, {E}}等等)。文献[9]采用集合方式进行编码，能够有效表示过程模型，但目前集合编码还难以在GPU上实现。

针对以上问题，本文提出一种新的实数编码方案。该方案中，因果矩阵由集合内容(chromosome)、集合标识(set label)和集合位置(set position)3个数组共同表示。为了便于说明，图2给出了表1中 $I(T)$ 的编码。数组chromosome中存储的是从 $I(A)$ 到 $I(H)$ 中包含的所有任务ID(内容)，如

果集合为空，则对应位置的值为–1。数组set position中存储的是 $I(t)$ 在chromosome中的起始地址，该数组的长度与任务集合 T 的大小一致。例如，图2中set position的第6个元素的起始位置是6，结束位置是下一个元素的起始位置减1(即为7)，所以 $I(G)$ 中有两个任务ID，分别是 E 和 F 。最后，数组set label中存储的是 $I(t)$ 中元素所属子集合的ID。例如，set label中E对应的值为0，F对应的值为1，它们对应的 $I(G)=\{\{E\},\{F\}\}$；同理，$I(H)=\{B,C,G\}$。

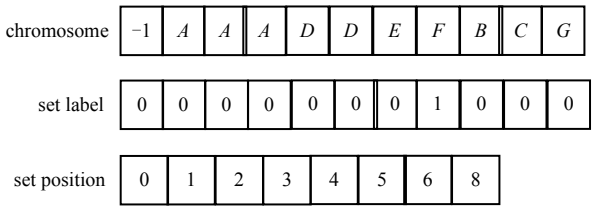

图2　因果矩阵在GPU上的染色体编码

3.3　并行遗传交叉

本文提出的并行交叉遗传算子计算步骤如下：

1) 各个线程随机计算一个个体ID；

2) 每个warp中的前16个线程各计算一个活动ID，并将计算结果保存在共享内存中；

3) 每个warp中的第i号线程(i<16)和第i+16号线程去查找第i个活动ID，并结合自身的个体ID，从全局内存中读取对应活动的染色体片段，并保存在共享内存中；

4) 各线程随机计算一个交叉点，取值范围不超过染色体片段中子集的个数，将结果保存在共享内存中；

5) 第i号线程和第i+16号线程进行遗传交叉运算，得到新的个体；

在第2节中提到过，同一个warp中的线程基于SIMT模式执行指令。而在指令层，GPU是按半个warp进行线程调度。因此为了避免出现执行分支，上述算法中让warp中前16个线程计算活动ID。然后让第i号线程与第i+16号线程共享同一个活动ID，并进行遗传交叉得到新的个体(下文中称为后代)。图3给出了一个遗传交叉的例子。此外，上述算法中，活动ID、染色体片段、交叉点都保存在共享内存中，这样算法的性能可以得到最大提升。

本文算法的另一个特点是没有采用交叉概率，而是通过启动大量的线程进行随机交叉遗传，以此增强算法的搜索范围来加快算法收敛。由于在CPU上无法进行大规模并行操作，因此文献[9]中遗传交叉算子采用的方法是先计算一个随机数，若该随机数小于交叉概率才对随机选择的两个个体进行交叉遗传，以此提高算法的随机性。

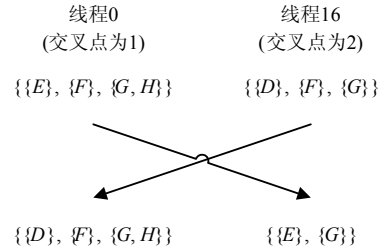

图3　一个遗传交叉的例子

3.4　并行遗传变异

本文提出的并行遗传变异算子计算步骤如下：

1) 每个warp的第0号线程计算一个随机数，并保存在共享内存中；

2) 各线程计算自己的warp ID，然后去读取步骤1)中生成的随机数；

3) 将得到的随机数与变异概率进行比较。若随机数小于变异概率，则继续执行；否则算法结束；

4) 按照文献[9]中的方法对后代个体进行遗传变异操作。

从上述算法可以看出，本文采用了一种粗粒度的方式进行并行遗传变异。这样做的目的仍然是因为GPU的SIMT机制。若采用细粒度的方式，即各个线程各自计算随机数以决定是否进行遗传变异，则会导致大量的执行分支，降低GPU的利用率。遗传变异结束后，本文算法采用了动态并行的方式，对后代中每个任务对应的染色体片段进行修复。

3.5　并行适应度计算

本文采用文献[9]中的适应度函数，如式(1)所示：

$$F(L,\boldsymbol{C},\boldsymbol{C}[])=\mathrm{PF}_{\mathrm{complete}}(L,\boldsymbol{C})-\kappa*\mathrm{PF}_{\mathrm{precise}}(L,\boldsymbol{C},\boldsymbol{C}[]) \tag{1}$$

式中，L 表示事件日志；$\boldsymbol{C}$ 表示因果矩阵(即遗传个体)；$\boldsymbol{C}[]$ 表示种群中所有的遗传个体；$\mathrm{PF}_{\mathrm{complete}}$ 表示的是模型(即遗传个体代表的因果矩阵)的完整性；$\mathrm{PF}_{\mathrm{precise}}$ 表示的是模型的精度；κ 是惩罚系数。更详细的解释请参考文献[9]，在此不再累述。

上述适应度函数的基本思想是利用事件日志来检验模型的完整性和精度。具体来讲，就是将事件日志中的事件轨迹依次输入到模型中进行检验。本文算法在计算适应度时，同一block中的线程处理相同的事件轨迹和不同的遗传个体。这样设计的目的仍然是为了遵循SIMT机制。同一warp中的线程读取相同的事件轨迹，这样同一warp中的线程之间

就能保持执行同步。每个线程计算完毕后，再通过CUDA提供的atomicAdd()原子操作，将计算结果累加到全局内存的数据中，从而实现适应度的计算。

4 仿真实验

4.1 实验设计

本节通过仿真实验对本文算法进行验证。实验的硬件平台是Dell PowerEdge T630服务器，装配有一个至强E5-2609处理器和32 GB内存；GPU是Nvidia GTX 1080Ti。仿真实验分为两个部分，首先验证本文算法中优化策略的有效性，其次是通过比较本文算法(GPM-GPU)与已有算法来验证算法性能。在第二部分实验中，实验基线采用的是原始的遗传过程挖掘算法(genetic process miner, GPM)[9]，另一个用于比较的算法采用的是有4个计算节点的分布式遗传过程挖掘算法(distributed genetic process miner, DGPM)[16]。本文提出算法采用C/C++实现，软件环境是windows server 2012 + CUDA 8.0,；实验比较所采用算法已经作为插件集成在ProM中，实验中采用的是版本是ProM 6.8(http:// www.promtools.org/)。

与文献[9，16]中的实验类似，本文采用了两个不同规模的事件日志作为实验数据，均是通过CPN工具(http://cpntools.org/)生成。第一个事件日志(event log 48)中包含了48个任务和300个随机生成的事件轨迹，第二个事件日志(event log 72)中包含了72个任务和500个随机生成的事件轨迹。此外，生成事件日志的模型中除了顺序、选择、并行结构外，还随机加入了少量循环结构和非自由选择结构。

遗传过程挖掘算法的参数设置如表2所示。在GPM-GPU算法中，block大小设置为256，grid大小分别设置为6和8。

表2 遗传过程挖掘算法参数设置

参数	event log 48		event log 72	
	GPM / DGPM	GPM-GPU	GPM / DGPM	GPM-GPU
种群数	150	150	200	200
进化代数	200	200	400	400
交叉概率	0.8	—	0.8	—
变异概率	0.2	0.2	0.2	0.2
κ	0.025	0.025	0.025	0.025

4.2 本文算法的优化策略验证

本节首先通过3组实验来验证本文算法中优化策略的有效性。第一组实验比较warp内的遗传交叉(warp crossover, WC)和相邻线程遗传交叉的性能差异(neighbor crossover, NC)。二者实现上的区别在于，前者是用warp内的前16个线程计算任务ID，然后由第i号线程和i+16号线程进行遗传交叉；后者用warp内的偶数号线程计算任务ID，然后由第i号线程和i+1号线程进行遗传交叉。第二组实验比较粗粒度遗传变异和细粒度遗传变异的性能差异。粗粒度遗传变异即本文算法中采用的策略(coarse-grained mutation, CGM)；细粒度遗传变异则是由线程各自计算一个随机数与变异概率进行比较，从而决定是否发生遗传变异(fine-grained mutation, FGM)。第三组实验比较的是两种不同的适应度计算方法，第一种方法使用的是本文提出的策略(same trace in block, STB)，第二种方法则是同一线程块中的线程处理不同的事件轨迹和相同的遗传个体(same individual in block, SIB)。

实验结果如表3所示。在第一组实验中，基于不同交叉策略的GPM-GPU算法在两个数据集上的搜索质量几乎没有区别，但是在遗传交叉算子的平均执行时间开销上，WC策略比NC策略在两个数据集上分别快约71%和75%。在第二组实验中，两种不同的遗传变异策略同样对算法的搜索质量没有影响，但是在遗传变异的平均计算时间开销上，粗粒度遗传变异策略CGM比细粒度遗传变异策略FGM在两个数据集上分别快约121%和123%。在第三组实验中，不同的适应度计算顺序对计算结果无影响，但是本文提出的策略STB要比第二种策略SIB在两个数据集上分别快约450%和438%。以上实验结果表明，本文提出的优化策略是有效的。

表3 优化策略的适应度值与执行时间开销比较

实验序号	遗传算子	event log 48		event log 72	
		适应度	执行时间/ms	适应度	执行时间/ms
实验一	WC	0.929	23.6	0.875	27.3
	NC	0.923	40.3	0.873	47.8
实验二	CGM	0.929	65.2	0.875	71.6
	FGM	0.928	143.8	0.878	159.5
实验三	STB	-	123.7	-	142.2
	SIB	-	680.4	-	764.8

4.3 本文算法的性能验证

本节通过比较本文算法GPM-GPU和CPU上的遗传过程挖掘算法以及4个计算节点的分布式遗传过程挖掘算法DGPM来验证提出算法的性能。实验结果如图4、图5所示。

从实验结果看，GPM-GPU的收敛速度和搜索

结果都要明显优于GPM和DGPM。在event log 48数据集上，GPM-GPU在第10代左右已经趋于收敛，而GPM和DGPM在遗传进化至200代时，其最优遗传个体的适应度值依然低于GPM-GPU搜索到的最优个体。而在event log 72数据集上，GPM-GPU在第50代左右已经趋于收敛，而GPM和DGPM则均是遗传进化至200代时才逐渐趋于收敛，并且GPM-GPU搜索到的最优个体的适应度值也要略优于GPM和DGPM搜索到的最优个体。

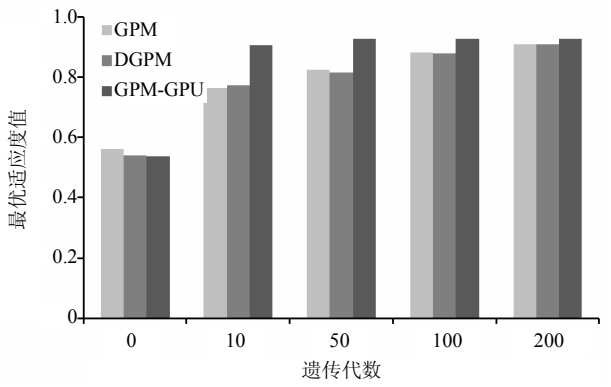

图4　event log 48数据集下的性能比较

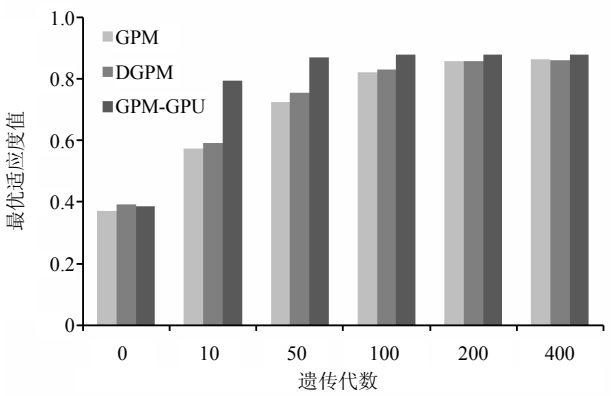

图5　event log 72数据集下的性能比较

本实验还比较了3个算法在两个数据集上进化200代/400代的执行时间开销，并且计算了GPM-GPU和DGPM相对于基线算法GPM的执行时间加速比，结果如表4所示。从表中可以看出，4个计算节点的DGPM算法相对于GPM算法能取得3.2倍和3.4倍的执行时间加速比。这与文献[16]中的实验结果基本一致。根据文献[16]中所述，DGPM算法在计算节点继续增加时并不能进一步降低执行时间开销，这是因为计算节点增多会导致节点之间的数据传输次数增多，反而降低加速性能。而GPM-GPU与DGPM相比则能够获得更好的执行时间加速比。从表4可以看出，GPM-GPU算法在两个数据集上分别取得了36.4倍和47.2倍的加速比。而且可以看出，事件日志中的任务数和事件轨迹数越多，GPM-GPU算法得到的执行时间加速比越大。

表4　执行时间开销及加速比

事件日志	GPM	DGPM		GPM-GPU	
	执行时间/s	执行时间/s	加速比	执行时间/s	加速比
event log 48	1 632.9	482.8	3.4	44.8	36.4
event log 72	5 022.3	1572.3	3.2	106.5	47.2

5　结 束 语

本文提出了一种基于GPU的并行遗传过程挖掘算法GPM-GPU。该算法首先解决了遗传过程挖掘算法中的因果矩阵在GPU上的表示问题；其次，充分利用GPU高并行的特点，利用海量线程对过程模型进行搜索；第三，对并行遗传过程挖掘算法中的遗传交叉和遗传变异算子、以及适应度计算都进行了较好的并行设计和实现。本文通过仿真实验对提出的算法进行了验证。实验结果表明：1) 本文算法中采用的优化策略是有效的，并且对所提出算法的整体性能提升能起到较大帮助；2) 本文算法能够充分利用GPU计算性能，其收敛速度和搜索质量都要明显优于现有遗传挖掘算法GPM和DGPM。在两个实验数据集上，本文算法与GPM算法相比达到了36.4倍和47.2倍的执行时间加速比。以上实验结果证实了本文算法的有效性及性能。

参 考 文 献

[1] VAN DER AALST W. Process mining: Data science in action[M]. 2nd edition. Berlin: Springer, 2016.

[2] 赵卫东, 刘海涛. 流程挖掘在流程优化中的应用[J]. 计算机集成制造系统, 2014, 20(10): 2633-2642.
ZHAO Wei-dong, LIU Hai-tao. Application of process mining in process optimization[J]. Computer Integrated Manufacturing Systems, 2014, 20(10): 2633-2642.

[3] 余建波, 郑小云, 李传峰, 等. 基于过程挖掘的临床路径Petri网建模[J]. 同济大学学报, 2018, 46(4): 524-534.
YU JIAN-bo, ZHENG Xiao-yun, LI Chuan-feng, et al. Clinical pathway modeling by petri net based on process mining[J]. Journal of Tongji University, 2018, 46(4): 524-534.

[4] 魏志杰, 金涛, 王建民. 基于临床数据挖掘的医疗过程异常发现方法及应用[J]. 计算机集成制造系统, 2018, 24(7): 1631-1641.
WEI Zhi-jie, JIN Tao, WANG Jian-ming. Outlier detection method in healthcare process based on clinical data mining[J]. Computer Integrated Manufacturing Systems, 2018, 24(7): 1631-1641.

[5] 杨丽琴, 康国盛, 蔡伟刚, 等. 业务流程挖掘算法研究[J]. 计算机应用与软件, 2016, 33(4): 44-50.
YANG Li-qin, KANG Guo-sheng, CAI Wei-gang, et al. On business process mining algorithms[J]. Computer Applications and Software, 2016, 33(4): 44-50.
[6] VAN DER AALST W, WEIJTERS T, MARUSTER L. Workflow mining: Discovering process model from event logs[J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(9): 1128-1142.
[7] WEN L J, VAN DER AALST W, WANG J M. Mining process model with non-free-choice constructs[J]. Data Mining and Knowledge Discovering, 2007, 15(2): 145-180.
[8] VAN ZELST S, VAN DONGEN B, VAN DER AALST W, et al. Discovering workflow nets using integer linear programming[J]. Computing, 2018, 100(5): 529-556.
[9] ALVES DE MEDEIROS A, WEIJTERS A, VAN DER AALST W. Genetic process mining: An experimental evaluation[J]. Data Mining and Knowledge Discovering, 2007, 14(2): 245-304.
[10] 顾春琴, 陶乾, 吴家培, 等. 基于混合自适应遗传算法的工作流挖掘优化[J]. 计算机科学, 2010, 37(3): 234-238.
GU Chun-Qin, TAO Qian, WU Jian-pei, et al. Workflow mining optimization based on hybrid adaptive genetic algorithm[J]. Computer Science, 2010, 37(3): 234-238.
[11] VAN DER AALST W, DAMIANI E. Processes meet big data: Connecting data science with process science[J]. IEEE Transactions on Services Computing, 2015, 8(2): 810-819.
[12] BRATOSIN C, SIDOROVA N, VAN DER AALST W. Distributed genetic process mining[C]//Proceedings of the IEEE World Congress on Computational Intelligence. Los Alamitos: IEEE, 2010: 1951-1958.
[13] VERBEEK H M W, VAN DER AALST W, MUNOZ-GAMA J. Divide and conquer: A tool framework for supporting decomposed discovery in process mining[J]. The Computer Journal, 2017, 60(11): 1649-1674.
[14] WANG J M, WONG R K, DING J, et al. Efficient selection of process mining algorithms[J]. IEEE Transactions on Service Computing, 2013, 6(4): 484-496.
[15] DAVID K, HWU W M. Programming massively parallel processors: A hand-on approach[M]. San Francisco: Morgan Kaufmann Publishers, 2010.
[16] RYOO S, RODRIGUES C I, BAGHSORKHI S S, et al. Optimization principles and application performance evaluation of a multi-threaded GPU using CUDA[C]// Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM, 2008: 73-82.

编 辑 叶 芳

(上接第909页)

[8] SANGUANSAT P, ASDORNWISED W, JITAPUNKUL S, et al. Two-dimensional linear discriminant analysis of principle component vectors for face recognition[C]//IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. [S.l.]: IEEE, 2006:II-II.
[9] ZHANG D Q, ZHOU Z H. (2D)2PCA: Two-directional two-dimensional PCA for efficient face representation and recognition[J]. Neurocomputing, 2005, 69(1): 224-231.
[10] NOUSHATH S, KUMAR G H, SHIVAKUMARA P. (2D)2LDA: An efficient approach for face recognition[J]. Pattern Recognition, 2006, 39(7): 1396-1400.
[11] QI Y F, ZHANG J S. (2D)2PCALDA: An efficient approach for face recognition[J]. Applied Mathematics and Computation, 2009, 213(1): 1-7.
[12] 赵冬娟, 梁久祯. 融合2DPCA和模糊2DLDA的人脸识别[J]. 计算机应用, 2011, 31(2) : 420-422.
ZHAO Dong-juan, LIANG Jiu-zhen. Face recognition algorithm fusing 2DPCA and fuzzy 2DLDA[J]. Journal of Computer Applications, 2011, 31(2): 420-422.
[13] 齐永锋, 火元莲. 一种结合2DLPP与2DPCA的人脸识别方法[J]. 西南交通大学学报, 2011, 46(6): 910-916.
QI Yong-feng, HUO Yuan-lian. Face recognition method combining 2DLPP with 2DPCA[J]. Journal of Southwest Jiaotong University, 2011, 46(6): 910-916.
[14] 李球球, 杨恢先, 奉俊鹏, 等. 2DPCA+ 2DLDA和改进的LPP相结合的人脸识别算法[J]. 计算机工程与应用, 2015, 51(21): 199-204.
LI Qiu-qiu, YANG Hui-xian, FENG Jun-Peng, et al. Face recognition algorithm based on 2DPCA+ 2DLDA and improved LPP[J]. Computer Engineering and Applications, 2015, 51(21): 199-204.

编 辑 刘飞阳