文章编号:1671-8836(2012)02-0157-08

MrBayes (MC)^{3}GPU 算法的优化

暴　洁, 夏宏举, 周剑夫, 王　刚, 刘晓光
(南开大学 信息技术科学学院,天津 300071)

摘　要: 为了进一步提高 CPU-GPU 协同贝叶斯种系发生算法 n(MC)^3 的并发度,本文在 n(MC)^3 算法基础上提出了优化算法,修改算法并行策略,重组计算次序,削弱相邻计算节点之间的依赖关系,增强 GPU 空闲单元的利用,实现了更高的加速比,显著提高了算法性能.

关　键　词: 种系发生; 贝叶斯; CUDA; 流; 并发度
中图分类号: TP 391　　文献标识码:A

An Optimization of MrBayes (MC)^3 on GPU

BAO Jie, XIA Hongju, ZHOU Jianfu, WANG Gang, LIU Xiaoguang
(College of Information Techincal Science, Nankai University, Tianjin 300071, China)

Abstract: MrBayes is a phylogenetic analysis program, which now is extensively used in phylogenetics. Considering the high parallelism of GPU(graphic process units), Jianfu Zhou and others from Nankai University put forward a changed CPU-GPU collaborative algorithm of MrBayes, and have achieved wonderful result. In this thesis, we put forward an optimization algorithm of MrBayes on the basis of CPU-GPU collaborative algorithm above. By changing the algorithm and reorganizing the order of calculation, we improved speedup ratio and achieved better performance indeed.

Key words: phylogenetics; Bayes; CUDA; stream; parallization

0　引　言

种系发生学,也被称作系统发生,是研究物种之间进化关系的一门科学. 种系发生学研究的结果以系统发生树(也称为演化树)表示,用它描述物种之间的进化关系. 近年来,在探讨不同生物的核酸序列、蛋白质序列以及形态特征数据之间的进化关系上,种系发生分析发挥了重要作用.

种系发生研究的主要问题是,种系发生的进化过程无法直接通过观察和实验证实. 所以需要综合各个方面的证据,推断物种进化关系(演化树结构). 由各类证据侧重点的区别,产生了多个不同的演化树版本. 通常情况下我们使用不同的分析方法对同一序列数据进行分析,以对进化关系做出较为准确的判断,当前,种系发生分析通常借助计算机进行,不同分析方法就产生出不同的算法. 目前应用比较广泛的有邻近归并法(Neighbor Joining, NJ)[1]、最大简约法(Maximum Parsimony, MP)[2]和最大似然法(Maximum Likelihood, ML)[3].

MrBayes 是基于最大似然法(Maximum Likelihood, ML)和贝叶斯推理法(Bayesian Inference, BI)的系统发生分析程序[4]. 贝叶斯种系发生算法与其他种系发生理论相比有几个优点,包括容易解释结果、吸收先验信息的能力以及一些计算优势[5]. 到目前为止,实现贝叶斯种系发生算法中应用最广泛的是马尔可夫链蒙特卡罗算法(Markov Chain Monte Carlo, MCMC),以及它的变型 Metropolis-coupled MCMC 算法(MC)^3[6,7].

新一代的 DNA 序列生成技术,可以有效地提高 DNA 序列生产量,并减少时间花费,同时维持数据的高品质[8]. 目前,大量的并行 DNA 序列生成技术造成了 DNA 数据的大量增长. 随着实际应用中的 DNA 数据越来越庞大,对于生物学家来说,传统

收稿日期: 2011-05-26
基金项目: 国家自然科学基金(60903028,61070014)资助项目,天津市科技支撑计划重点项目(11ZCKFGX01100)
作者简介: 暴　洁,女,硕士生,现从事并行计算及其应用研究.　E-mail:bj_11@126.com

$(MC)^3$算法的运算速度已经不能满足需求.因此,一些学者提出各种并行算法来加速$(MC)^3$计算,例如$p(MC)^3$、PBPI 和$h(MC)^3$等[9~12].

近年来,用于通用计算(General-Purpose computing on Graphics Processing Units,GPGPU)的图形处理器(Graphics Processing Units,GPU)越来越受到研究者的关注.GPU 有着很强的计算能力和显存带宽,大量计算核心可以并发地运行上万个线程.因此,GPU 很适合作为 CPU 的加速装置来加速计算,处理海量数据集或进行大量运算操作[13].NVIDIA 推出的统一计算设备架构(Compute Unified Device Architecture,CUDA)显著缩短了 GPU 程序设计学习周期,是 GPGPU 成功的重要原因之一.在 GPU 上实现贝叶斯种系发生算法的加速是未来的发展趋势,目前代表性的算法有$g(MC)^3$等[14].周剑夫等提出的$n(MC)^3$算法更是实现了 40 倍以上的加速[15],本文是在$n(MC)^3$算法基础上的改进,通过计算次序重组,达到更好的加速效果.

1 贝叶斯种系发生算法与 CUDA 技术

1.1 贝叶斯种系发生算法综述

1.1.1 贝叶斯种系发生算法基本原理

在贝叶斯模型中,种系发生算法基于种系发生树的后验概率进行计算[7].在一次种系发生观察中,对于 DNA 序列(X),一棵种系发生树的后验概率,可以用贝叶斯定理计算:

$$f(\psi \mid X)=\frac{f(X \mid \psi)\times f(\psi)}{f(X)} \tag{1}$$

此公式利用种系发生树的先验概率$f(\psi)$和似然值$f(X \mid \psi)$,计算出树的后验概率$f(\psi \mid X)$.先验概率$f(\psi)$,是对参数分布的初始化估计;种系发生树的后验概率$f(\psi \mid X)$,可以解释成为这棵树是正确的概率.举例来说,拥有最高后验概率的树可能被选择为最好的种系发生模型.贝叶斯模型将模型参数视为随机变量,在不考虑实际输入数据序列的情况下,为参数假设先验概率,通常所有树的先验概率都被赋为相等的值.似然值$f(X \mid \psi)$将在后续的计算过程中得到.

1.1.2 $(MC)^3$ 算法

MCMC 的基本思想是构建一个马尔可夫链,元素空间模拟为它的状态空间,元素的概率分布作为静态分布,然后模拟链的实现,并将结果看作是一个在感兴趣的元素概率分布(这里指种系发生树的后验概率分布)中有效的、典型的抽样[11].对于种系发生问题,MCMC 使用 Metropolis-Hastings 策略构建马尔可夫链,通过随机干扰得到新树,计算新树的接受概率,来探索后验概率分布.能够证明,如果链运行足够长的时间,并且最终是不可约的和非周期的,那么任何树在链运行的过程中被访问的时间的比例,是对这棵树的后验概率的有效近似[16].

$(MC)^3$算法是 MCMC 算法的改进.$(MC)^3$增加了链的热量值属性,运行H(通常$H>1$)条马尔可夫链,其中$H-1$条是加热的.与冷链(或未加热的链)比较,加热的链有更高的接受概率.通过随机选择两条链交换状态,来克服局部最优的问题.

$(MC)^3$算法的过程描述如下[9]:

第 1 步 用ψ_i表示第i条马尔可夫链的当前树.如果这是第一次循环,随机地选择一个整数值赋给ψ_i.对于所有H条链都做同样的工作.

第 2 步 对于所有链,$i \in \{1,2,\cdots,H\}$

① 随机干扰ψ_i生成一棵新树(ψ_i').$q(\psi_i')$表示生成ψ_i'的概率,$q(\psi_i)$表示通过干扰ψ_i'提出ψ_i的概率(假设情况下的概率,实际上没有进行这一动作).

② 为ψ_i'计算接受概率(R_i'):

$$R_i'=\min\left[1,\left(\frac{f(X \mid \psi_i')}{f(X \mid \psi_i)}\times\frac{f(\psi_i')}{f(\psi_i)}\right)\beta_i\times\frac{q(\psi_i)}{q(\psi_i')}\right] \tag{2}$$

其中$\beta(0<\beta<1)$是链的热量值.对于冷链,$\beta_i=1$.

③ 在均匀分布的$(0,1)$区间抽取一个随机变量(U_i).如果$U_i<R_i'$,接受ψ_i'让$\psi_i=\psi_i'$.

第 3 步 在所有链完成了给定次数的循环后,随机选择两条链(j和k)交换状态.交换状态的接受概率(R)是

$$R=\min\left[1,\frac{f(\psi_k \mid X)^{\beta_j}f(\psi_j \mid X)^{\beta_k}}{f(\psi_j \mid X)^{\beta_j}f(\psi_k \mid X)^{\beta_k}}\right] \tag{3}$$

第 4 步 在均匀分布的$(0,1)$区间抽取一个随机变量(U),如果$U<R$,接受状态交换,即链j和k交换状态.

第 5 步 回到第 2 步.

1.1.3 $(MC)^3$算法的并行扩展

Altekar 和 Dwarkadas 等人提出了$p(MC)^3$算法,这是一个粗粒度链级并行算法,它将马尔可夫链分配给不同进程,然后并行地运行这些进程,每个进程独立计算若干条链[9].这个算法还采用了一种点对点的同步机制和热量交换机制来最小化进程间通信机制开销.但是,$p(MC)^3$算法的并发度被链的数目所限制,对于大多数实际应用来说难以达到较高的并发度.

另外两种主要的并行算法是贝叶斯并行算法 PBPI[10,11],和混合并行算法 h(MC)³[12]. 这两种算法都是同时在链级和 DNA 序列级两个层次上实现并行. 两个算法的不同之处是,在 PBPI 中,两个层次的并行机制都使用消息传递模型;而 h(MC)³ 使用消息传递模型实现链级并行,使用共享存储模型实现 DNA 序列级并行,而且还利用了超线程技术. 这两种算法都有良好的并行效果.

应用 GPU 加速的 g(MC)³ 算法,是一个 DNA 序列级的细粒度并行算法[14]. GPU 负责条件概率的计算,因为这是贝叶斯种系发生算法最耗时的部分,且由于不同条件概率的计算是完全独立的,所以易实现并行. 尽管如此,g(MC)³ 算法并未高效的实现,而且没有关注 CPU-GPU 间通信开销的优化.

南开大学周剑夫等人提出了一种 CPU-GPU 协同的 n(MC)³ 算法[15]. n(MC)³ 算法在(MC)³ 的基础上重新组织计算流程,将计算任务分配到大量 GPU 线程上并发执行,有效地降低了 CPU 和 GPU 之间的通信开销. 该算法在 CPU 端和 GPU 端都实现了并行:在 CPU 端使用消息传递模型实现链级并行,在 GPU 端使用 CUDA 编程模型实现 DNA 序列级并行. 此外,n(MC)³ 还设计了 CPU-GPU 混合流水线模型,允许 CPU-GPU 间的数据传输与 GPU 端的内核函数执行同时进行,显著地降低了 CPU-GPU 间的通信开销.

1.2　CUDA 计算平台

CUDA 是显卡厂商 NVDIA 公司于 2007 年推出的 GPU 通用并行计算架构,在产业界和学术界引发了巨大的反响.

1.2.1　CUDA 简介

传统的 GPGPU (General-purpose computing on graphics processing units,基于 GPU 的通用计算)计算,一般采用 CPU+GPU 异构模式,CPU 执行复杂逻辑处理和事务管理,GPU 负责大规模数据并行计算. 因为受到硬件可编程性和开发方式的制约,GPGPU 开发难度大,应用领域难以推广.

2007 年 6 月,NVIDA 公司推出了 CUDA,一种将 GPU 作为数据并行计算设备的软硬件体系. CUDA采用了比较容易掌握的类 C 语言开发,不需要借助于图形学 API,减少了开发难度. 同时,用来支持 CUDA 的 GPU 的架构也进行了相应改进. 到目前为止,CUDA 版本不断地进行功能完善,在各个领域都得到广泛应用.

1.2.2　CUDA 执行模型和存储模型

CUDA 执行模型将 CPU 作为主机(Host),GPU 作为设备(Device). 在 GPU 上运行的 CUDA 并行计算函数称为内核函数(kernel),是整个 CUDA程序中的一个可以被并行执行的步骤[13].

kernel 函数被 GPU 上大量线程并发执行,线程被组织成三层结构:最高层是线程网格(grid),中间层是线程块(block),每个块包含数目相等的线程,最底层是线程(thread). 调用 kernel 时需指定线程网格、线程块的维数和每一维的大小. 各 block 之间并行执行,无法通信,没有执行顺序. 线程之间通过共享存储器和同步机制进行通信.

CUDA 存储模型中,每个线程拥有自己的私有存储器寄存器(Register)和局部存储器(local memory). 寄存器速度最快容量最小,而局部存储器中的数据实际上存储在全局显存中,访问速度很慢.

一个 block 拥有一块共享存储器(shared memory),块内所有线程都可读写,因此可用于块内线程通信. 它是 GPU 片内的高速存储器,访问速度几乎和访问寄存器一样快,但容量较小.

Grid 中所有的线程都可以访问全局存储器(global memory),也就是所谓的"显存",它在CUDA存储层次中是最大的存储区域,但访问速度最慢. 同时,可以用来进行 GPU 线程间、CPU 和 GPU 间的通信. 此外,所有线程都可以访问两个只读存储器:常数存储器(constant memory)和纹理存储器(texture memory).

2　n(MC)³ 改进算法

本文的主要工作是对 n(MC)³ 算法的并行策略进行改进,因此我们首先了解 n(MC)³ 算法的思路.

2.1　n(MC)³ 算法介绍

n(MC)³ 算法是(MC)³ 算法的优化,是 MrBayes 的并行实现方案[15]. 按照上文提到的(MC)³ 算法思路,为了计算生成的新树(ψ_i')的接受概率,链 i 应首先计算似然值($f(X\mid\psi_i')$,或简称 L),这是整个(MC)³ 算法中最为耗时的部分. n(MC)³ 算法的主要改进也集中在这部分,调整计算流程,将计算任务分配给 GPU 线程并发执行,算法 1 如图 1 所示.

n(MC)³ 利用 GPU 与 CPU 协同进行(MC)³ 计算. 算法分为三个主要循环. Loop1 在 CPU 端运行,为生成的新树计算所有转移概率矩阵,并把矩阵从 CPU 传输到 GPU. Loop2 负责计算局部概率. 对每条链,首先调用 GPU 内核函数,自底向上地计算新树所有内部节点的条件概率;然后调用另一内核函数使用根节点的条件概率计算新树的局部概率;最

```
算法 1
1:Input:瞬时速率矩阵,种系发生树模型 ψ_i
2:Output:更优种系发生树模型ψ_i'
3:for all chains, i∈{1,2,…,H} do//Loop1
4:   随机干扰 ψ_i 生成新树(ψ_i')
5:   for (k=1;k≤2N-2;k++) do
6:      计算转移概率矩阵(Q_k)
7:   end for
8:   将所有矩阵从 CPU 传输到 GPU
9:end for
10:for all chains, i∈{1,2,…,H} do//Loop2
11:for (k=N+1;k≤2N-1;k++) do
12:      调用 GPU 内核函数并行计算树ψ_i'的节点 k 的条件
         概率
13:   end for
14:   调用 GPU 内核函数并行地计算ψ_i'的局部概率 L_u
15:   将局部概率 L_u 从 GPU 传输到 CPU
16:end for
17:for all chains, i∈{1,2,…,H} do//Loop3
18:   与相应的 GPU 流同步
19:   设置全局概率 L=1
20:   for (u=1;u≤M;u++) do
21:      L=L×L_u
22:   end for
23:   使用 L 为ψ_i'计算接受概率(R_i')
24:   接受或拒绝ψ_i'
25:end for
```

图 1 n(MC)^3 算法

后,将结果传回 CPU 端. Loop3 负责计算似然值,在 CPU 端执行. 每一次迭代,都要进行 CPU 和 GPU 的同步,以保证结果的正确性. 需要注意的是,虽然 Loop2 遍历树的所有内部节点,但实际程序中采用了一种局部优化策略:对于干扰后的树,只需计算与原树相比发生变化的局部——从某个叶节点至根节点的路径. 因此,需要计算的节点数目为 h(树高),而非 $N-1$(内部节点数,N 为 DNA 序列长度,即叶节点数).

2.2 n(MC)^{3}算法分析

n(MC)^{3}算法中,GPU 的计算工作主要集中在算法的 Loop2 中. 此循环是程序中最耗时的部分. 在程序中每条链的计算被分配在单独的流中运行,其并行执行状况需仔细分析.

Loop2 主要由一个二重循环构成,外循环逐条链处理通信和计算任务,内循环对每条链逐节点启动数据传输和内核函数执行. 在链与链之间(流之间),数据传输和计算、计算和计算是可以并发执行的,Fermi 架构最多允许四个内核函数并发执行,但 n(MC)^{3}算法的这种"链序"计算限制了算法的并发度. 在第 i 条链(流)的所有节点的数据传输指令和内核函数调用都启动之后,算法才启动第 $i+1$ 条链(流)的数据传输指令和内核函数调用. 即使一个流内的内核函数调用是异步的(仅仅启动内核函数,进入调度队列,并不等待内核函数执行完毕),仍然会花费一定时间. 这样,对于规模较小的数据集(DNA 序列较短),节点的计算时间相对于"准备时间"(包括数据传输和内核函数启动时间)较短,就可能造成这样的局面:当处理后续链时,前驱链的内核函数已执行完毕,导致多流无法真正并发执行.

更具体地,假设共有 H 条链,并且每条链的演化树树高均为 h,即对每条链,Loop2 内循环均处理 h 个节点. 节点的条件概率计算时间 T_{cl} 近似地认为相等,节点的准备时间 T_s 也近似地认为相等. 那么可实现两条链并行(两个流并行)的条件,即后继链开始时,前驱链尚未计算完毕的条件为:

$$T_s < T_{cl} \tag{4}$$

虽然在此条件下,算法执行过程中会出现两条链并发计算的状态,但不一定一直处于此种状态. 当 $T_s<T_{cl}\leqslant 2T_s$ 时,算法运行过程如图 2 所示.

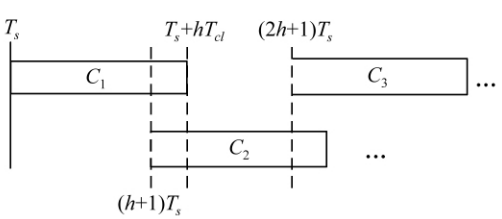

图 2 $T_s<T_{cl}\leqslant 2T_s$ 时算法运行时序

其中 C_i 表示第 i 条链的计算时间,假定节点准备时间小于计算时间,可被有效掩盖. 可以看出,由于链 1 计算完成时间 T_s+hT_{cl} 先于链 3 就绪时间 $(2h+1)T_s$,因此算法执行过程中有某些时段处于一条链单独计算的状态. 假定计算实例使用 8 条链,GPU 可充分支持 4 条链并发计算(可认为 GPU 有 4 个计算单元,分别能进行一个 kernel 函数的执行),可得算法运行时间如下:

$$T_c = (7h+1)T_s + hT_{cl} \tag{5}$$

类似的,可知当 $2T_s<T_{cl}\leqslant 3T_s$ 时,算法运行过程中会出现三条链并发计算的状态. 当 $3T_s<T_{cl}\leqslant 4T_s$ 时,算法运行过程中会出现四条链并发计算的状态. 这两种情况下,算法运行过程类似图 2,但分别可以利用到 3 个和 4 个计算单元. 算法运行时间仍如公式(5).

当 $T_{cl}>4T_s$ 时，后继链的就绪时间先于前驱链的完成时间，算法运行过程中一直处于 4 条链并发计算的状态. 可得算法运行时间如下：

$$T_c=(h+1)T_s+2hT_{cl} \tag{6}$$

2.3　“节点序”计算次序

通过上一小节的分析可知 n(MC)^3 算法的并发度受到限制，将“链序”计算次序改为“节点序”次序来改进这一问题：将外循环改为逐节点处理，内循环改为逐链处理. 即，首先启动所有链的第一个节点的条件概率计算，接着启动所有链的第二个节点的条件概率计算，依此类推. 修改后的算法如算法 2(图 3).

可以看出，算法 2 的 Loop1 和 Loop3 两个循环与算法 1 完全相同，没有进行任何修改. 而原来负责计算内部节点条件概率和树的局部概率的 Loop2，被拆成两个较小的循环. 算法 2 的第 10 到 14 行为第一个小循环，计算内部节点条件概率，仍是二重循环，但改为“节点序”计算次序. 第 15 到 18 行为第二个小循环，专门负责计算局部概率，并将结果传输回 CPU.

```
算法 2
1:Input:瞬时速率矩阵,种系发生树模型 ψ_i
2:Output:更优种系发生树模型ψ_i'
3:for all chains, i∈{1,2,…,H} do//Loop1
4:   随机干扰 ψ_i 生成新树(ψ_i')
5:   for (k=1;k≤2N−2;k++) do
6:      计算转移概率矩阵(Q_k)
7:   end for
8:   将所有矩阵 Q 从 CPU 传输到 GPU
9:end for
10:for all internal nodes,
   k∈{N+1,N+2,…,2N−1} do//Loop2
11:   for (i=1;i<=H;i++) do
12:      调用 GPU 内核函数并行计算节点 k 的条件概率
13:   end for
14:end for
15:for all chains, i∈{1,2,…,H} do
16:   调用 GPU 内核函数并行地计算ψ_i' 的局部概率 L_u
17:   将局部概率 L_u 从 GPU 传输到 CPU
18:end for
19:for all chains,i∈{1,2,…,H} do //Loop3
20:   与相应的 GPU 流同步
21:   设置全局概率 L=1
22:   for (u=1;u≤M;u++) do
23:      L=L×L_u
24:   end for
25:   使用 L 为ψ_i'计算接受概率(R)
26:   接受或拒绝ψ_i'
27:end for
```

图 3　GPU“节点序”算法

2.4　改进算法分析

为了比较两个算法的性能差异，我们延续上一节的假设条件. 当 $T_s<T_{cl}\leqslant 2T_s$ 时，改进算法的运行过程如图 4 所示.

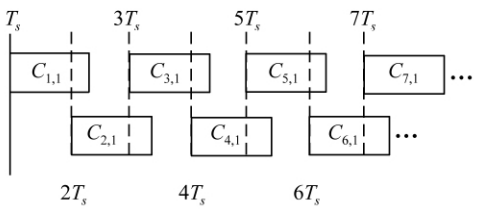

图 4　$T_s<T_{cl}\leqslant 2T_s$ 时改进算法运行时序

其中，$C_{i,j}$ 表示第 i 条链的第 j 个节点的计算时间. 易知，算法运行时间为：

$$T_n=8hT_s+T_{cl} \tag{7}$$

当 $2T_s<T_{cl}\leqslant 3T_s$ 及 $3T_s<T_{cl}\leqslant 4T_s$ 时，算法对 GPU 计算能力的利用效率不同，但运行时间形式上仍如公式(7). 当 $T_{cl}>4T_s$ 时，运行时间为：

$$T_n=4T_s+2hT_{cl} \tag{8}$$

对比公式(5)、(6)，易知，在所有情况下，改进算法的性能均优于 n(MC)^3 算法. 原因在于改进算法相继计算无依赖关系的节点(不同链中节点)，而非有依赖关系的同一链中的祖先后代节点，从而大大降低了 GPU 计算单元空闲等待的情况，图 2 和图 4 清楚地展示了这一点. 此外，由于 CUDA 硬件平台的特点，n(MC)^3 算法中“CPU→GPU 传输→kernel 调用→GPU→CPU 传输”的次序(见算法 1)无法保证 kernel 的并发执行，而改进算法的指令次序(见算法 2)不存在此问题. 因此，两个算法的实际性能差距要比公式(5)与(7)((6)与(8))所显示的更大.

3　实验及结果

3.1　实验平台及数据集

3.1.1　算法实验平台

运行程序的计算机配置如表 1 所示. 该计算机配备两块 NVIDIA GeForce GTX 480 卡，每块卡占据一条 16 速 PCI-E 总线.

算法采用的串行程序为 MrBayes Version 3.1.2，并行版本分别是 n(MC)^3 算法和本文的改进算法.

GPU 端代码的编译器是由 NVIDIA CUDA-Toolkit Version 4.0 提供的 NVCC，CPU 端代码的

表 1 系统配置

操作系统	Red Hat Enterprise Linux 5.3
CPU	1×AMD Phenom Ⅱ X4 945:核心类型,Deneb;核心数量,四核;主频,3.0 GHz
内存容量	2×2GB DDR3 1333
显卡驱动	2×NVIDIA GeForce GTX 480:计算核心,Fermi;CUDA 核心数,480;主频,1.4 GHz; 显存容量,1536MB GDDR5;并行运行核心数,4

编译器是 GCC Version 4.3.对于算法在 CPU 端的链级并行,我们采用 MPICH2 Version 1.1.

3.1.2 算法使用数据集

在实验 1 中我们使用了四个数据集,如表 2 所示.这四个数据集都来自在南开大学生命科学学院昆虫学系研究中的实际 DNA 数据[17,18].所有数据可以从美国国家健康协会的基因序列数据库中免费获取[19].这组数据用来测试算法的加速比.

在实验 2 中我们使用了十一个数据集,如表 3 所示.这十一个数据集的特征是物种集群数相同,而 DNA 序列长度呈递增状态,用于比较 DNA 序列长度改变对算法性能影响.

表 2 实验 1 中采用的 DNA 数据集

数据集	Taxa/个	DNA 长度/个
1	26	1 546
2	37	2 238
3	111	1 506
4	288	3 386

表 3 实验 2 中采用的 DNA 数据集

数据集	Taxa/个	DNA 长度/个
1	60	500
2	60	1 000
3	60	2 000
4	60	3 000
5	60	4 000
6	60	5 000
7	60	6 000
8	60	7 000
9	60	8 000
10	60	9 000
11	60	10 000

3.2 实验结果分析

我们的实验主要分为两部分,第一部分是用真实数据集来测试改进算法的加速比,第二部分评估 DNA 序列长度对算法性能的影响.

3.2.1 改进算法加速比实验结果

评估并行算法的性能,一个重要的指标就是加速比.一个并行算法的加速比被定义为单处理器系统计算问题花费的时间与由 p 个相同处理器单元构成的并行系统解决同一问题花费的时间之比.

我们在实验平台上比较了 n(MC)^3 算法和改进算法的性能.运行 8 条马尔可夫链,分析数据集 1 到 4,两个算法运行方式类似,都是将 8 条链均匀分布在 4 个进程上,每个进程分配一个单独的 CPU 处理核心,每两个进程共享使用一块 GTX 480 卡.

测试每个数据集 5 次,每次执行使用 4×4 核苷酸置换模型持续迭代计算 100 000 代,计算出 5 次执行的平均运行时间.

实验结果如图 5 和图 6.其中图 5 是 n(MC)^3 算法与改进算法运行时间的对比.

为了测试并行算法的加速比,我们选择 MrBayes Version 3.1.2 的串行版本作为已知最快的串行算法,根据它计算出两个算法的加速比.图 6 是改进算法与 n(MC)^3 算法加速比的对比.

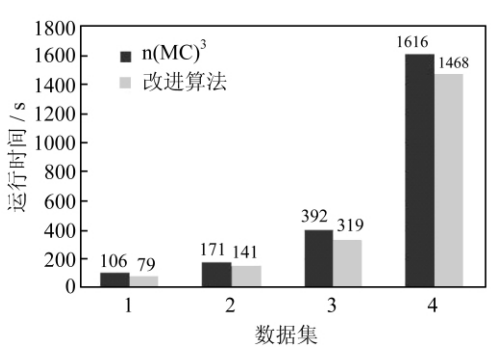

图 5 n(MC)^3 算法与改进算法运行时间对比

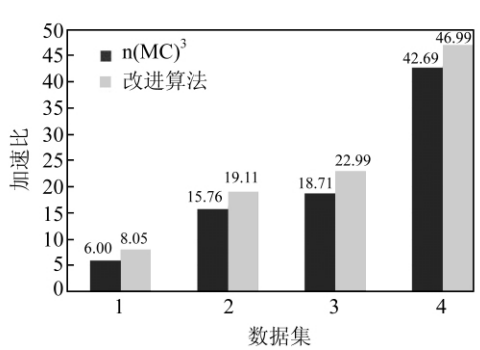

图 6 两个算法的加速比

可以看出，在 4 个数据集上，改进算法的运行时间都要少于原算法. 在数据集 3 上性能提高幅度较为明显，约为 22.9%.

分析数据集之间的差异，可以发现，性能提高幅度最大的数据集 3，其 DNA 序列长度是最短的. DNA 序列长度越短，节点计算时间相对于准备时间越短，n(MC)³ 算法就越难以实现充分并行，因而改进算法较之 n(MC)³ 算法的性能提升就越显著. 为了证明这一推断，我们进行了实验 2.

3.2.2　DNA 序列长度对算法性能的影响

实验 2 所用的数据集为表 3 所示数据集，这 11 个数据集的物种数目全部相等，DNA 序列长度递增. 即，在算法处理的矩阵中，行数不变，列数由小到大变化. 如 3.2.1 分析，随着 n(MC)³ 算法并发度的提高，改进算法的性能提升效果会越来越不明显.

实验环境保持不变，我们同样测试每个数据集 5 次，每一次执行使用 4×4 核苷酸置换模型迭代计算 100 000 代，计算 5 次执行的平均运行时间.

实验测试最终数据如表 4 所示.

表 4　两种算法平均运行时间比较

数据集	n(MC)³ /s	改进算法 /s	节约/%
1	201	150	25.37
2	263	209	20.53
3	362	315	12.98
4	468	419	10.47
5	562	518	7.83
6	673	623	8.92
7	738	704	4.61
8	860	818	4.88
9	944	921	2.44
10	1 063	1 031	3.01
11	1 152	1 113	3.39

可以看出，随着 DNA 序列长度增加，节约时间的百分比越来越小. 因此我们的推断是正确的，随着 DNA 序列长度的增加，改进算法的提升效果降低. 因为原始 n(MC)³ 算法的并发度受 DNA 序列长度影响较大，DNA 序列长度越短，演化树的树高越小，由表 4 分析可知，n(MC)³ 算法的并发度越低. 相应地，对 GPU 计算能力的利用就越差. DNA 序列长度增加时，n(MC)³ 算法的并发度提高，性能得到提升. 而改进算法受到 DNA 序列长度的影响更小，DNA 序列长度较短时，也能得到较好的并发度. 相应地，当 DNA 序列长度增大时，性能提升幅度不明显. 因此，当 DNA 序列长度增大时，两个算法的性能差异越来越小. 此外，当 DNA 序列长度较大时，GPU 的计算能力并不足以支撑 4 个节点完全并发计算，这也是造成两个算法性能差异变小的原因之一，同时也造成实验结果与第 2 节的分析结果并不完全吻合(第 2 节算法分析的前提是 GPU 可支撑 4 个节点完全并发计算).

4　结论

本文在 n(MC)³ 算法基础上，提出改进的计算次序，来进一步加速贝叶斯种系发生算法(MC)³. 就笔者所知，n(MC)³ 算法是目前为止贝叶斯种系发生计算领域最快的(MC)³ 算法. 而我们实验结果显示，与 n(MC)³ 算法相比，在 4 个实际应用的 DNA 数据集上，改进算法成功地达到了优化效果，获得了更高的加速比.

贝叶斯种系发生算法在生物领域的实际研究工作中应用十分广泛，而我们改进的基于 CUDA 加速的 n(MC)³ 算法还有很多优化的空间. 例如，对于处理数据的类型还有待进一步扩展，使得算法更全面地解决实际应用中的问题. 此外，算法的加速比仍有提升的可能，由于实际中的问题越来越趋向于海量数据规模，需要我们设计适合大规模 CPU-GPU 混合集群的并行算法来应对.

具体的技术手段，可以采用数据拆分方式，形成更多流的并行，以充分利用大量 CPU 和 GPU 计算资源，另一方面可研究 CPU 和 GPU 间的任务调度算法，实现异构计算单元间的负载均衡.

参考文献：

[1] 孙啸，陆祖宏，谢建明. 生物信息学基础[M]. 北京：清华大学出版社，2005：230-232.
Sun Xiao, Lu Zuhong, Xie Jianming. *Basics for Bioinformatics* [M]. Beijing: Tsinghua University Press, 2005:230-232(Ch).

[2] Swofford D L. *PAUP: Phylogenetic Analysis by Parsimony and Other Methods*[M]. Sunderland(MA): Sinauer, 1999.

[3] Schmidt H A, Strimmer K, Vingron M, *et al*. Tree-puzzle: Maximum likelihood phylogenetic analysis using quartets and parallel computing[J]. *Bioinformatics*, 2002, **18**(3): 1-3.

[4] Li S, Pearl D K, Doss H. Phylogenetic tree construction using Markov chain Monte Carlo[J]. *Journal of the American Statistical Association*, 2000, **95**(450): 493-508.

[5] Larget B, Simon D L. Markov chain Monte Carlo al-

gorithms for the Bayesian analysis of phylogenetic trees [J]. *Molecular Biology and Evolution*, 1999, **16**(6): 750-759.

[6] Gilks W R, Richardson S, Spiegelhalter D. *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*[M]. London: Chapman and Hall/CRC, 1996.

[7] Huelsenbeck J P, Ronquist F, Nielsen R, *et al*. Bayesian inference of phylogeny and its impact on evolutionary biology[J]. *Science*, 2001, **294**(5550): 2310-2314.

[8] Rogers Y H , Venter J C. Massively parallel sequencing[J]. *Nature*, 2005, **437**: 326-327.

[9] Altekar G, Dwarkadas S, Huelsenbeck J P, *et al*. Parallel metropolis coupled Markov Chain Monte Carlo for Bayesian phylogenetic inference[J]. *Bioinformatics*, 2004, **20**(3): 407-415.

[10] Feng X, Buell D A, Rose J R, *et al*. Parallel algorithms for Bayesian phylogenetic inference[J]. *Journal of Parallel and distributed Computing*, 2003, **63**(7-8): 707-718.

[11] Feng X, Cameron K W, Buell D A. PBPI: A high performance implementation of Bayesian phylogenetic inference[DB/OL]. [2011-03-04]. *http://sc06.supercomputing.org/schedule/pdf/pap292.pdf*.

[12] Zhou J F, Wang G, Liu X G. A new hybrid parallel algorithm for Mrbayes[J]. *ICA3PP*(1) (*LNCS*), 2010, **6081**: 102-112.

[13] **张舒, 褚艳利**. GPU **高性能运算之** CUDA[M]. **北京: 中国水利水电出版社**, 2009: 14-81.

Zhang Shu, Chu Yanli. *GPU High Performance Computation: CUDA*[M]. Beijing: China Water Power Press, 2009: 14-81(Ch).

[14] Pratas F, Trancoso P, Stamatakis A, *et al*. Fine-grain parallelism using multi-core, cell/be, and GPU systems: Accelerating the phylogenetic likelihood function [DB/OL]. [2011-03-04]. *http://www.pluylo.org/sub__sections/portal/portal-papers/ICPP2009__2.pdf*.

[15] Zhou J F, Wang G, Liu X G. MrBayes on a Graphics Processing Unit[J]. *Bioinformatics*, 2011, **27**(9): 1255-1261.

[16] Tierney L. Markov chains for exploring posterior distributions[J]. *Annals of Statistics*, 1994, **22**(4): 1701-1728.

[17] Xie Q, Bu W, Zheng L. The Bayesian phylogenetic analysis of the 18S rRNA sequences from the main lineages of Trichophora (Insecta: Heteroptera: pentatomomorpha)[J]. *Molecular Phylogenetics and Evolution*, 2005, **34**(2): 448-451.

[18] Xie Q, Tian Y, Zheng L, *et al*. 18S rRNA hyper-elongation and the phylogeny of Euhemiptera (Insecta: Hemiptera)[J]. *Molecular Phylogenetics and Evolution*, 2008, **47**(2): 463-471.

[19] National Center for Biotechnology Information, Genbank[DB/OL]. [2010-12-03]. *http://www.ncbi.nlm.nih.gov/genbank/*.

□