云南大学学报（自然科学版），2012，34（1）：15～19　　CN 53－1045/N　ISSN 0258－7971
Journal of Yunnan University　　http://www.yndxxb.ynu.edu.cn

并发遗传退火算法求解复杂非线性方程组*[1]

付振岳[1]，王顺芳[1]，丁海燕[1]，黄光能[2]
（1.云南大学 信息学院，云南 昆明　650091；2.云南大学 软件学院，云南 昆明　650091）

摘要：问题求解空间的扩大和种群规模的增加，导致传统的遗传退火算法在求解复杂非线性方程组时显得迟缓和性能不足.在多核处理器的环境下，把并发机制和最大堆引入遗传退火算法，并应用于复杂非线性方程组的求解中，给出一种具体设计思路.仿真实验结果表明，该机制有效地提高了遗传退火算法的性能，加快了求解速度.

关键词：复杂非线性方程组；并发；遗传退火算法；最大堆
中图分类号：TP 319　　**文献标识码**：A　　**文章编号**：0258－7971(2012)01－0015－05

在计算机工程领域经常面临求解复杂非线性方程组问题，此类方程组中部分方程含高次代数、超越函数等，根的数目和位置均未知，对它的求解是很困难的.当前，对复杂非线性方程组解的研究是一个国际热点.

遗传退火算法是由遗传算法和模拟退火算法各自取长补短组合而成的一种进化类型的算法.遗传算法是对自然界生物进化过程进行模拟，具有强大的全局搜索能力和较快的收敛速度，但容易陷入局部最优而达不到全局最优.模拟退火算法是由Metropolis等基于热力学的退火机制提出来的一种对退火过程进行模拟，具有迅速跳出局部最优的特性.

遗传退火算法综合了遗传算法以及模拟退火算法各自的优势，使得在求解非线性方程组中有着很高的实用性[1－3].但是在一些复杂的含有三角函数或是对数的非线性方程组中，由于此类方程组的特性，扩大了问题求解空间的范围，导致算法需要初始化更多种群个体进行搜索，这无疑对算法的性能提出了更高的要求.算法的性能可以从2个方面进行分析：①算法对硬件的利用率；②算法关键步骤的时间复杂度.

基于以上分析，本文在利用遗传退火算法(GAA)[2]求解此类非线性方程组时，引入了并发机制和最大堆来提高硬件利用率和降低某些关键步骤的时间复杂度.仿真实验结果证明，遗传退火算法经过并发设计和使用最大堆之后，很大程度上提高了求解此类非线性方程组的速度.经过改进的遗传退火算法，是一种有效求解该类方程组的方法.

1　遗传退火算法并发设计求解复杂非线性方程组可行性分析

1.1　并发是一种特殊的并行机制　基于CPU主频提高的技术瓶颈，多核处理器成为提高计算机计算速度的主要手段.多核处理器的出现和流行，对一些需要进行大量的CPU计算操作的程序使用并发来提高计算效率提供了有利的条件[4].

并发机制的核心是创建多线程.相对于串行程序，并发程序的优势在于：串行程序在执行的过程

* **收稿日期**：2011－04－10
基金项目：国家自然科学基金资助项目(10901135，11171293，10626048)；云南省社发计划应用基础研究面上资助项目(2008CD081，2010CC003)；昆明市第九批中青年学术和技术带头人后备人选资助项目；云南大学中青年骨干教师培养计划资助项目.
作者简介：付振岳(1983－)，男，湖南人，硕士生，主要研究方向：计算方法，计算机系统结构.
通讯作者：王顺芳(1974－)，女，云南人，教授，博士，主要从事计算方法、统计机器学习和数理统计方面的研究.E－maill：shunfangw@yahoo.com.cn.

中,由于程序的设计策略和单一的进程,即便是在多核处理器的机器中,也仅有1个处理器进行运算,其他处理器均被闲置,这样极大地浪费了处理器资源,然而并发程序却能充分利用多线程机制,创建多个线程同时使用多个处理器,提高多核处理器利用率. 就一定程度而言,并发是一种特殊的并行机制,它的实质是在处理器级别上的并行.

串行算法并发设计之后对性能的改善程度主要从以下2个方面考虑:首先,不同算法对不用硬件的利用率有差别. 有些问题的求解算法由于其对内存或IO等其他硬件利用率较大,对CPU利用率较小,并发设计之后反而适得其反. 其次,任何算法都是有一部分是不能被并行化的[5],一个算法被并行化之后,速度提高率主要是由其可并行化的部分和必须串行化的部分所决定,并且遵循Amdahl定律[5]:

$$\text{Speedup} \leqslant \frac{1}{F + \dfrac{(1-F)}{N_{\text{cpu}}}},$$

其中,Speedup是速度能被提高的比率,F是算法中必须串行化的部分($0 \leqslant F \leqslant 1$),$N_{\text{cpu}}$是处理器的个数. 例如,10%必须串行化的算法在一台有10个处理器的计算机上能够加速5.3倍,即便是在一台拥有100个处理器的计算机上也只能加速到9.2倍. 由此可知,在对程序的并发设计过程中,应使可并行化的部分尽可能的多,从而提高资源利用率.

1.2 将并发机制引入遗传退火算法的可行性分析 将并发机制引入遗传退火算法基于以下3点考虑:

(1) 遗传退火算法具有先天的并发性. 遗传退火算法本质是对自然界种群的模拟从而寻得最佳个体(即最优解),而自然界个体的行为从来都是并发的. 算法被设计得并发性越强,则越能准确的体现种群个体的行为和相互之间的影响.

(2) 遗传退火算法个体具有一定的独立性和群体相关性. 算法中种群个体各自具有相关的属性,自身属性一方面由本身决定(如变异行为),但更多的方面是由种群中的其他个体决定的. 串行算法在一定程度上不能准确地模拟自然界个体并发的行为,亦难以体现个体的独立性和群体相关性.

(3) 遗传退火算法本身在求解复杂非线性方程组时涉及到大量的浮点数运算,计算量相当大. 一般而言,基于存储数据的操作对存储介质以及总线传输速度的要求较高,而基于浮点数运算的遗传退火算法则要求有较强运算能力的处理器,如多核处理器. 并发设计的遗传退火算法则能更加充分利用多核处理器的计算能力.

这些都给将算法由串行转化为并发提供了一定的理论依据.

1.3 将最大堆引入遗传退火算法的可行性分析 将最大堆的引入对遗传退火算法基于以下3点考虑[6-9]:

(1) 传统的遗传退火算法每经过每1次寻优、交叉以及变异之后都会将按照个体的自身状态进行一次排序,由此可见排序算法的好坏对整个遗传退火算法的效率有着极为重要的影响. 目前基于比较的具有稳定性的内部排序算法的时间复杂度为$\Omega(N\log_2 N)$(此时间复杂度不包括基于非比较排序算法如桶排序等,并且此类排序算法还有些其他的条件作为限制). 由此可知,即便是最佳的内部排序算法在种群规模很大时计算量也很大,而将最大堆(以下统称为堆)引入之后,则可以利用堆的特性,将排序过程转化为堆的初始化过程,对个体的排名的处理将转化为对个体所处层在堆中的位置的处理,其中堆进行初始化的时间复杂度的仅为$O(N)$.

(2) 在对遗传退火算法步骤的分析当中,可以看出算法对最佳个体的操作颇为频繁. 在堆中,最佳个体总是位于堆的堆顶,很容易找到.

(3) 利用堆的特性在遗传退火算法中地变异操作时有效的避开了繁重的计算量.

2 遗传退火算法并发设计

2.1 并发设计遗传退火算法的基本原理 本文的设计思路是按照生成的线程数,将整个种群平均分割成若干个组群,组群之间可以交换信息,每个线程对应1个组群,每个组群对应1个堆. 个体存储在堆中,堆顶元素为该组群中最大元素,堆使用数组顺序存储方式. 在对组群进行遗传退火的相关操作以及组群之间的信息交互的过程中,逐步找到符合要求的1个或者是1群个体.

2.2 算法相关符号说明 方程组描述,在实际应用中,非线性方程组的一组解通常以向量$X = (x_1, x_2, \cdots, x_m)$表示. 实函数非线性方程组的一般

形式为:

$$\begin{cases} f_1(x_1,x_2,\cdots,x_m) = 0, \\ \quad\vdots \\ f_m(x_1,x_2,\cdots,x_m) = 0, \end{cases}$$

其中 $f_i(i=1,\cdots,m)$ 为 $X=(x_1,x_2,\cdots,x_m)$ 的非线性函数.

个体状态:本算法中,为了形象表示求解非线性方程组的动态过程,用 $X^i=(x_1^i,x_2^i,\cdots,x_m^i)$ 来表示个体 i 及其状态,其中 $x_p^i(p=1,2,\cdots,m)$ 表示欲寻优的分量.

组群:整个种群根据线程数来分割成相应的组群. 种群规模用 N 表示,组群规模用 n 表示.

评估函数:在模拟遗传或是退火的过程中,X^i 的状态会不断发生变化,然每变化 1 次,便可根据 X^i 中的值,通过评估函数进行计算,从而得出个体状态对准确解的接近程度. 评估函数记为 $F(X^i)$,一般而言,$0<F(X^i)\leqslant 1$,当 $F(X^i)=1$ 时,表示个体此时的状态为准确解. 本文采用经典的评估函数:

$$F(X^i) = 1/\left(\sqrt{\sum_{k=1}^{m} f_k^2(X^i)} + 1\right).$$

堆排名:X^i 所处层在堆中的位置,记做 $P(X^i)$. 在用数组顺序结构存储元素的堆中,位于位置 i 的个体 X^i 在组群中的堆排名为 $\log_2 i+1$. 如堆顶元素堆排名为 1,位置 4,5,6,7 的个体堆排名为 3.

2.3　算法基本行为

2.3.1　寻优行为　个体向组群最优个体靠近,即自身每个属性在最优个体的方位上进行微调,最优个体保持不变. 其寻优按照以下公式进行:

$$x_p^i = x_p^i + (x_p^1 - x_p^i)\mathrm{Random}(0,1).$$

其中 x^1 表示最优个体,$\mathrm{Random}(0,1)$ 表示在 0 到 1 的范围内的由均匀分布产生的任一随机数.

2.3.2　交叉行为　交叉行为是为了将优良个体的基因尽可能地遗传给子代,从而使子代能够迅速收敛,达到全局最优.

其操作过程为将组群内的每个个体与最优个体进行交叉操作,产生新个体. 同时根据淘汰过程中所淘汰个体的数量,按照一定比率让靠近堆顶的个体与最优个体进行多次交叉操作,产生多个新个体,弥补淘汰过程中个体减少的数量,使组群规模保持不变. 交叉行为按照以下公式进行:

$$x_p^i = (x_p^i + x_p^1)/2 + (x_p^1 - x_p^i)\mathrm{Random}(0,1).$$

2.3.3　变异行为　变异行为是为了防止本组群过早地收敛,而将模拟退火个体扰动的思想引入个体变异的过程中,并按照个体堆排名和退火过程中温度的状况进行不同幅度的扰动. 个体的变异函数是一个关于当前个体状态和当前温度的联合函数:

$$x_p^i = x_p^i + \mathrm{Random}(0,1)g(a)g_1(P(X^i)) + \mathrm{Random}(0,1)h(b)g_2(t).$$

$g_1(P(X^i))$ 是个体堆排名扰动力度函数:

$$g_1(P(X^i)) = P(X^i)/(\log_2 n + 1).$$

堆排名靠前的个体扰动较少,堆排名靠后的元素扰动较大,位于同一层的元素扰动力度相同. 这样便可以将某一个体的 $g_1(P(X^i))$ 值存储起来,并应用同层其他元素,即便是当组群规模 n 非常大时,由于堆的层数不超过 $\log_2 n+1$,所以这个值仅需要计算 $\log_2 n+1$ 次,有效地减少了计算的次数.

$g_2(t)$ 是个体当前温度扰动力度函数:

$$g_2(t) = (t_{\max} - t)/t_{\max},$$

$t_{\max}$ 为初始温度,t 为当前温度.

其中系数函数:

$$g(x) = \begin{cases} -1, x < 0.5 \\ 1, x \geqslant 0.5 \end{cases},\text{随机数 } x \in [0,1).$$

2.3.4　淘汰行为　进化过程中,遗传退火算法按照一定的比例,选取堆排名靠后的个体进行淘汰. 目的在于避免状态较差的个体所耗费的计算,同时使更多优良个体集中在最优区间附近.

2.4　遗传退火算法并发设计中需要注意的问题

2.4.1　线程数目[10-11]　线程数目是遗传退火算法并发设计中最为重要的部分. 一方面,线程数目越多,越能提高 CPU 的利用率;但另一方面,线程的创建、销毁以及上下文切换时需要耗费一定的资源. 作为一种折衷的方案,所创建的线程数目可以遵循以下公式[1]:

$$N_{\text{threads}} = N_{\text{cpu}} * U_{\text{cpu}} * \left(1 + \frac{W}{C}\right).$$

其中,N_{threads} 表示所需要创建的线程数,U_{cpu} 表示 CPU 的利用率,$\frac{W}{C}$ 表示计算等待时间的比率[5]. 因为计算 U_{cpu},$\frac{W}{C}$ 颇为复杂,这里我们采用另一种效果相近但计算简便的创建线程数目的方法[5],即 $N_{\text{threads}} = N_{\text{cpu}} + 1$.

2.4.2 并发设计及同步化控制 在并行程序的设计中,需要对使用全局变量的部分进行同步化控制,其目的是为了避免竞态条件保证线程安全. 如果同步控制不当,则会导致资源浪费或者是死锁等情况. 在并发设计中,将本组群内所有操作由1个线程完成,其产生的中间变量用ThreadLocal策略[5,10]进行控制. 每轮遗传退火操作结束后进行组群之间信息交互. 将遗传退火算法的并行部分最大化,缩小同步块范围来提高资源利用率.

2.5 并发设计的遗传退火算法解非线性方程组的步骤

步骤1:输入种群规模、算法终止条件(准确解接近度)、退火初始温度、退火终止温度、温度下降因子,淘汰率以及需要产生的线程数目.

步骤2:根据线程数将种群分割成为相应的组群.

步骤3:每个线程对组群先后进行堆初始化→寻优→堆初始化→交叉→堆初始化→变异→堆初始化→淘汰等操作,若此过程中出现满足条件的个体,则退出程序.

步骤4:因为每个线程执行的速度不一致,故设一个全局变量记录全体线程在执行遗传退火算法中的产生的最优个体,每个线程在完成1轮遗传退火行为后,将本组群的最优值与全局最优值进行比较,如果本组群最优值优于全局最优值,则用本组群最优值替换全局最优值,反之,则用全局最优值替换本组群最优值.

步骤5:返回步骤3.

3 数值仿真实验

本文采用Java语言实现GAA的串行设计和并发设计,均在一台配置为CPU双核1.73 GHz,内存1.5 GB的计算机上进行,操作系统为Windows XP sp2,jdk版本为1.6. 为了比较遗传退火算法串行设计以及并行设计性能上的差别,对算法其他部分的参数统一选取为:

$N = 3\,000, L = 0.99, r = 0.1, T\_start = 100, T\_end = 0.000\,000\,1, k = 0.96.$

其中N为种群规模,L为算法终止条件,r淘汰率,T_start为退火初始温度,T_end为退火终止温度,k为温度下降因子. 用遗传退火算法的串行方式和并发方式对以下2组有代表性的方程组各测试10次,所得结果如下:

$$\begin{cases} 3\sin x_1 + x_2^2 - 3 = 0, \\ x_1 + \cos x_2 - 2 = 0; \end{cases} \tag{1}$$

$$\begin{cases} \ln x_1 + \ln x_2 = 6, \\ 2\ln x_1 + \ln 3x_2 = 10. \end{cases} \tag{2}$$

由以上结果可知,将并发机制和最大堆引入多遗传退火算法之后,算法求解时间有了显著的缩短. 算法中需要同步控制的串行部分仅仅是各组群最优值与全局最优值的信息交互的环节,相对于并行部分而言几乎可以忽略. 按照Amdahl定律,在具有2个处理器的机器上,可以加速到接近2倍. 但是由于线程的创建、销毁、上下文转换等开销以及其他硬件方面的限制,最终结果如表1和表2所示,分别提高了1.67倍和1.44倍.

表1 方程组(1)测试结果

Fig. 1 The test results of equations(1)

模式	x_1^i	x_2^i	$F(X^i)$	时间/ms
并发设计	1.136	-0.528	1.0	93
(两组解)	1.136	0.528	1.0	
串行设计	1.136	-0.528	1.0	156
(两组解)	1.136	0.528	1.0	

表2 方程组(2)测试结果

Fig. 2 The test results of equations(2)

模式	x_1^i	x_2^i	$F(X^i)$	时间/ms
并发设计	18.199	22.167	1.0	1 636
串行设计	18.199	22.167	1.0	2 357

4 结束语

遗传退火算法的并发设计,极大地提高了CPU的利用率,获得了良好的求解速度. 遗传退火算法由于自身的特点,为其并发设计带来了诸多的便利. 同时在遗传退火算法的设计过程中用堆来存储元素,符合遗传退火算法求解非线性方程组的特点,并从理论上来说,组群规模很大的情况下,在排序和变异这2个关键步骤中,降低了时间复杂度,提高了算法的性能. 遗传退火算法的并发设计,需要考虑安全和效率2个因素. 在步骤4中,对全局

变量的访问时需要同步的,否则会发生脏读取而不安全,然而对同步控制的范围也应当进行合理的设计,否则会导致效率的低下. 在并发设计中,不能只追求安全或是只追求效率,应当根据不同的实际情况在安全与效率中做出适当选择,折衷考虑往往会达到更好的效果.

参考文献:

[1] 袁泉,何志庆,冷慧男. 用于一类函数全局优化问题的混合遗传算法[J]. 计算机工程,2008,34(12):181-183.

[2] 邵平凡,万程鹏. 求解全局优化问题的遗传退火算法[J]. 计算机工程与应用,2007,43(12):62-65.

[3] 蓝海,王雄,王凌. 复杂函数全局最优化的改进遗传退火算法[J]. 清华大学学报:自然科学版,2002,42(9):1 237-1 240.

[4] HERLIHY Maurice,SHAVIT Nir. Multiprocessor synchronization and concurrent data structures[M]. Morgan - Kaufman,2006.

[5] GOETZ Brian,PEIERLS Tim,BLOLCH Joshua. Java concurrency in practice[M]. Addison Wesley Professional,2006.

[6] WEISS M A. Data structures and algorithm analysis in C[M]. 2nd ed. 北京:人民邮电出版社,2005.

[7] WILLIAMS J W J. Algorithm 232(heapsort)[J]. Communications of the ACM,1964,7(6):347-348.

[8] P van Emde Boas. Preserving order in a forest in less than logarithmic time[C]//Proceedings of the 16th Annual Symposium on Foundations of Computer Science, IEEE Computer Society,1975:75-84.

[9] FLOYD R W. Algorithm 245 (treesort)[J]. Communications of the ACM,1964,7:701.

[10] BLOCH J. Effective Java programming language guide[M]. AddisonWesley,2001.

[11] HARRIS T,MARLOW S,PEYTON - JONES S,et al. Composable memory transactions[C]//PPoPP 05:Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,ACM Press,2005:4860.

Concurrent genetic - annealing algorithm for solving complex nonlinear equations

FU Zhen-yue[1], WANG Shun-fang[1], DING Hai-yan[1], HUANG Guang-neng[2]

(1. School of Information Science and Engineering,Yunnan University,Kunming 650091,China;
2. School of Software,Yunnan University,Kunming 650091,China)

Abstract:The expanding of problem - solving space and the increasing of population bring insufficient to genetic - annealing algorithm (GAA) which is based on classical design. In the condition of multi - processor,this paper not only takes concurrent mechanism and max heap into GAA,which is applied to solve the complex nonlinear equations,but also gives a specific designing idea. Simulation results demonstrate that the proposed methods improve the performance of GAA and accelerate the speed for solving such equations.

Key words:complex nonlinear equations;concurrecy;genetic - annealing algorithm;max heap