# 基于GPU的现代并行优化算法

张庆科[1] 杨 波[1] 王 琳[1] 朱福祥[2]
(济南大学山东省网络环境与智能计算技术重点实验室 济南 250022)[1]
(北京理工大学网络与分布式计算研究所 北京 100081)[2]

**摘 要** 针对现代优化算法在处理相对复杂问题中所面临的求解时间复杂度较高的问题,引入基于GPU的并行处理解决方法。首先从宏观角度阐释了基于计算统一设备架构CUDA的并行编程模型,然后在GPU环境下给出了基于CUDA架构的5种典型现代优化算法(模拟退火算法、禁忌搜索算法、遗传算法、粒子群算法以及人工神经网络)的并行实现过程。通过对比分析在不同环境下测试的实验案例统计结果,指出基于GPU的单指令多线程并行优化策略的优势及其未来发展趋势。

**关键词** 现代优化算法,图形处理器(GPU),计算统一设备架构(CUDA),组合优化,并行计算
**中图法分类号** TP399 **文献标识码** A

## Research on Parallel Modern Optimization Algorithms Using GPU

ZHANG Qing-ke[1] YANG Bo[1] WANG Lin[1] ZHU Fu-xiang[2]
(Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China)[1]
(Computer Networks and Distributed Systems Laboratory, Beijing Institute Technology, Beijing 100081, China)[2]

**Abstract** In order to deal with the relatively high time-complexity of practical issue, parallel modern optimization based on GPU was presented in this paper. Firstly, CUDA parallel programming architecture and programming model were summarized at a macroscopic level. Then the parallel processes of five typical modern optimization algorithms(Simulated Annealing, Tabu Search, Genetic Algorithms, Particle Swarm Optimization and Artificial Neural Network) using CUDA programming model were provided. Experimental statistics measured in different environment indicate that the parallel method can obtain better performance on average than CPU. Finally the parallel optimization strategy was discussed and the outlook of future direction of parallel optimization algorithm was also pointed out.

**Keywords** Modern optimization algorithms, GPU, CUDA, Combinatorial optimization, Parallel computing

## 1 引言

现代优化算法是20世纪80年代初兴起的启发式算法,它是相对经典优化算法而言的,经典优化问题针对的是连续性问题而且问题规模相对较小,主要采用的优化方法有线性与非线性规划、动态规划、多目标规划、整数规划、排队论和决策论等[1]。随着实际中问题的多元化和离散化,大量的组合优化问题尤其是NP(Non-deterministic Polynomial,非确定多项式复杂度)完全问题[2],其求解时间随规模有呈指数级增长趋势。经典优化算法由于求解代价较高,因此已不能有效地应对组合优化问题求解。现代优化算法则是建立在客观存在的自然现象基础上,通过仿效生物进化或生物群体智能的方式来实现对实际复杂问题的优化求解。

现代优化算法主要包括模拟退火(Simulated Annealing)、禁忌搜索(Tabu Search)、遗传算法(Genetic Algorithms)、群体智能(Swarm Intelligence)和人工神经网络(Neural Networks)。这些算法在理论和实际应用中得到了较快的发展,并用于解决大量的实际问题,但由于NP类的复杂性和数据处理的高密性,使得求解的速度和效率相对较低。因此,实现现代优化算法的并行化对解决当前的实际优化问题颇为重要。

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。

## 2 并行技术

### 2.1 并行概述

算法的并行实现方式有多种,从硬件角度分析,当前并行计算的类别归纳起来可分为两种:一种是独立式并行,可划分

到稿日期:2011-07-30 返修日期:2011-09-17 本文受国家自然科学基金(60873089, 60573065, 61070130, 60903176),山东省自然科学杰出青年基金(JQ200820),新世纪优秀人才支持计划(NCET-10-0863)资助。

张庆科(1985—),男,硕士生,CCF学生会员,主要研究方向为高性能计算与水泥建模,E-mail:miczqk@hotmail.com;杨 波(1965—),男,教授,博士生导师,主要研究方向为网络环境与计算智能;王 琳(1982—),男,博士,讲师,主要研究方向为水泥建模与高性能计算;朱福祥(1985—),男,硕士生,主要研究方向为网络分布式计算与高性能计算。

为多核计算并行和采用异构模式(GPU 和 CPU 结合)下的协同并行;另一种是基于集群的并行,当前较为典型的是"对称式多重集群系统"并行模式,它将多个处理器的节点通过高速网络连接设备连接成为多级体系并行计算结构,在节点间实行粗粒度并行,而在节点内实现细粒度并行的两级并行模拟;从软件编程角度分析,典型的并行编程模型又可以分为以下几种:基于消息传递的 MPI(Message Passing Interface)模式、集群系统节点内的 OpenMP 多线程模式、基于 CUDA 的异构编程模式和混合异构并行模式。多级混合编程模式又可包含:MPI 和 OpenMP 的二级混合编程模型,MPI+OpenMP+CUDA 的三级混合编程模型[3],而当前基于 GPU 的编程模式发展最为迅速。

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 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[4]。在 GPU 多层存储器架构下(见图 1),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。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。

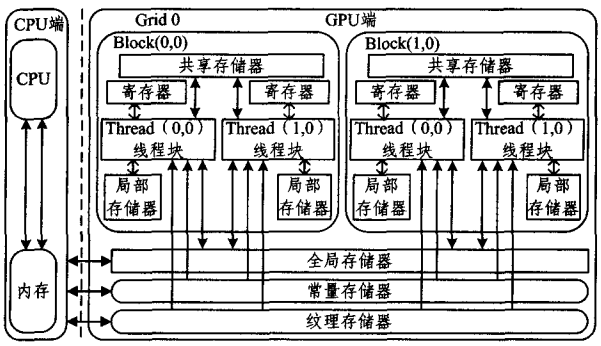


图 1 GPU 多级存储器模型

### 2.2 基于 CUDA 的并行编程

CUDA 采用了单指令多线程(Single Instruction Multiple Thread)执行模型,图 2 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:

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。

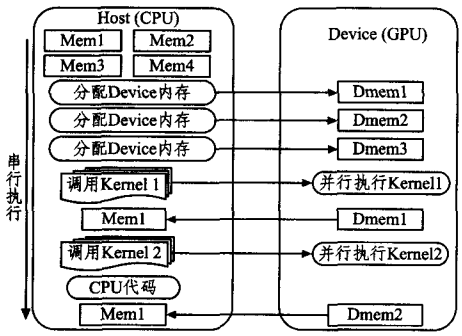


图 2 GPU-CUDA 并行编程模型

## 3 现代并行优化算法

所谓优化问题就是在满足一定的约束条件下找到一组参数值,以使某些最优性度量得到满足[5]。优化算法实际上是一种搜索过程或搜索规则,它基于某种思想或机制,通过一定途径寻找到问题的解。优化问题可分为函数优化和组合优化,函数优化的对象是一定区间内的连续变量,组合优化的对象则是解空间中的离散状态。在组合优化中,首先需要建立问题的目标函数,然后通过优化算法求解出该目标函数的最优解,该过程的数学模型可以简单地描述为:$\min\delta=f(x)$;$S.t.\ g_i(X)\geqslant 0(i\geqslant 1, X\in D)$;其中 $\delta=f(x)$ 为目标函数,$g(x)$ 为目标约束函数,其数目可以有多个。$D$ 为约束域,令 $F(X)$ 表示可行解,$F(X)=\{X\mid X\in D, g(X)\geqslant 0\}$,其中满足 $f(X^*)=\{\min f(X)\mid X\in F\}$ 的解 $X^*$ 即为该组合优化问题的最优解。

### 3.1 并行模拟退火算法

模拟退火算 SA(Simulated Annealing algorithm)由 Metropolis 在 1953 年提出,Kirkpatrick 在 1983 年首次将其应用在组合优化问题上。SA 来源于固体退火原理,是局部搜索算法的拓展,区别于局部搜索之处就是以一定的概率选择邻域中代价值大的状态,是基于 Monte Carlo 迭代求解策略的一种随机全局寻优算法。通过马尔可夫(Markov chain)过程理论推导,SA 以概率 1 收敛于全局最优解[6]。该算法中主要包括:新状态产生函数、新状态接受函数、退温函数、抽样稳定准则和退火结束准则。算法开始时设计一个初始温度值,初始温度和上面的函数和准则将是直接影响算法优化结果的主要环节,但 SA 与初始值无关,即算法求得的解与初始解的状态或算法迭代的起点无关。算法运行时是从某一较高初温开始,结合具有概率突跳特性的 Metropolis 抽样策略在解空间中随机寻找目标函数的全局最优解,伴随温度参数的不断下降重复抽样过程,最终得到问题的全局最优解。该算法主要是解决 NP 复杂性问题,克服优化过程陷入局部极小值的情况和算法对初值的依赖性问题[7]。

由于算法要求较高的初始温度、较慢的降温速率、较低的终止温度,以及各温度下足够多的抽样频率,而冷却进度表并不能从根本上提高算法的效率,因而算法实现效率较低。由

于 SA 算法初始阶段和结束阶段与算法进程具有一定的独立性，抽样过程和退火过程也具有一定的独立性的特点，因此算法适合实现并行化。SA 的并行研究主要集中在以下 3 个方面：操作并行性、进程并行性、空间并行性。操作并行性即将算法的各个执行环节在不同的处理机上分别完成。进程并行即由处理机独立模拟退火搜索过程，以空间资源弥补串行搜索的不足。空间并行即将搜索空间划分成多个子区域，每个区域由不同的处理机执行 SA 的搜索过程，最后综合得到原问题的最优解。文献[8]集中利用多线程技术实现并行化，并对算法本身提出 4 种并行方案：操作并行、试验并行、区域分裂并行和混乱松弛并行。文献[9]提出了一种在大型并行机上运行混合 SPMD 模拟退火算法，以机群系统作为算法实现的主要并行计算平台，在克服经典模拟退火算法内在串行性的同时，进一步和下山法结合起来，并综合多种优化方法，取得大规模可扩展的并行效果，提高了算法的收敛速度，克服了算法性能对初始值和参数选择的过分依赖。在上述方法中进程间通信的消耗限制了线程规模，多线程技术是在 CPU 上用串行模拟并行，不能真正提高性能，此外并行机使用也比较复杂。目前基于 GPU 的高速浮点运算能力、并行计算和可编程功能优势明显，拥有空间和时间双重并行机制，为并行模拟退火算法提供了新的思路。基于 GPU 的并行 SA 算法的一般流程如图 3 所示，具体的步骤可概括为：

(1)将搜索空间分解成若干子区域，各子区域首先产生初始温度 $T\_k$，并随机产生 $N$ 个不同的初始解 $X_i(i=0,\cdots,N)$；

(2)在未达到温度 $T\_k$ 的平衡状态时，每个初始解在 GPU 线程内部并行执行下面①－③的操作；

①根据解 $x$ 的范围，生成新的可行解；

②并行计算 $\Delta f=|f(X')-f(x)|$ 的值，其中 $f(X')$ 为新解的评价函数，$f(x)$ 为旧解的评价函数；

③根据概率 $\min\{1,\exp(-\Delta f/T\_k)\}>random[0,1]$ 接受新解，其中 $random[0,1]$ 代表[0,1]区间内的随机数；

(3)令 $T\_(k+1)=\alpha,k\leftarrow k+1$，其中 $\alpha\in(0,1)$。若满足收敛条件，则结束退火过程；否则，转(2)重复执行；

(4)综合各子区域优化结果，计算比较后求出最终的最优解。

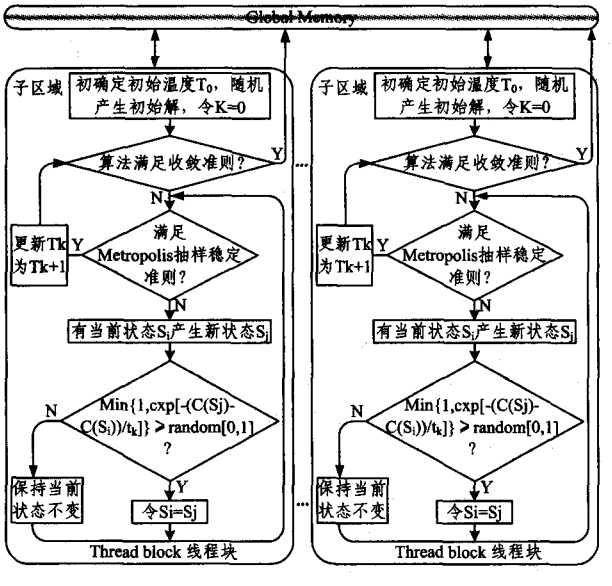


图 3　基于 GPU 的并行模拟退火过程

### 3.2　并行禁忌搜索算法

禁忌搜索算法 TS(Tabu Search)是由美国的 Glover 教授于 1986 年提出的，该算法是对局部领域搜索法的推广，是一种全局逐步寻优算法。该算法在搜索策略上采用了集中搜索和扩散搜索机制。集中搜索就是从某一点出发，在这点的领域内集中寻找最优解，直到找到局部最优解，然后将其存放在禁忌表中。禁忌表中存放搜索过的 $K$($K$ 称为禁忌表长度)个邻域移动，这些移动称作“禁忌移动”(Tabu Move)。禁忌表是一个循环表，搜索过程中被循环地修改，使禁忌表中始终保存着 $K$ 个移动，$K$ 次以后释放该移动，但释放前是被禁忌访问的，以避免回到原先的解。扩散策略就是根据禁忌表，不再选取或有选择地搜索禁忌表内的某些局部最优点，从而避免了算法陷入早熟现象，实现全局范围的搜索。

禁忌搜索算法的缺点是对初始解高度依赖，好的初始解可使禁忌搜索在解空间中搜索到更优的解，差的初始解则会降低禁忌搜索的收敛速度，搜索到的解也相对较差。当陷入局部极小值时则无法保证全局最优性。当前对禁忌搜索算法的优化改进主要有串行混合改进法以及并行改进法。串行方面，文献[10]提出了一种将 GEP 和 TS 结合的方法，由于 TS 是串行结构而且依赖于初始解，而基因表达式编程是并行结构，具有较强的全局搜索能力，因此将 TS 和 GEP 嵌套使用后不仅弥补了 TS 对初始解的依赖问题，同时 TS 也增强了 GEP 的局部搜索能力。并行方面，当前的并行优化改进策略可归总为两大类：基于空间分解的并行优化策略以及基于多搜索任务的并行策略。基于问题空间分解的并行策略思想是通过禁忌搜索空间分解将原问题划分为若干子问题，各子问题采用不同的禁忌搜索算法求解，进而实现并行化，但这种空间分解并行机制对同步性要求较高。基于多任务并行策略即将多个禁忌搜索算法同时执行，每个算法内部采用相同的或不同的搜索参数。文献[11]提出了禁忌搜索的 GPU 并行实现，主要思想是将模拟退火算法中的退火过程和最优解的评估和计算过程由 GPU 上的 thread 线程并行执行，多线程并行计算不仅提高了退火的速度，而且节省了算法执行的时间。基于 GPU 的禁忌搜索算法的一般流程如图 4 所示，具体的执行步骤如下：

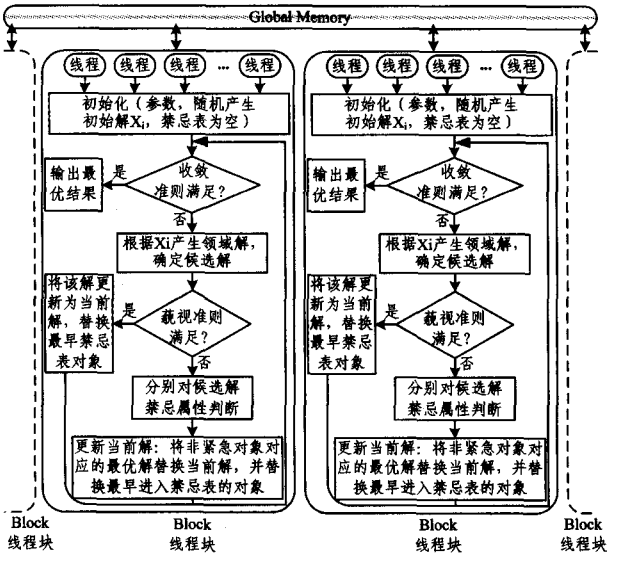


图 4　基于 GPU 的并行禁忌搜索算法

(1)首先随机初始可行解 $x$，初始可行解 $x$ 可以从一个启

发式算法获得或者在可行解集合 $X$ 中任意选择，确定完初始可行解后，定义可行解 $x$ 的邻域移动集 $s(x)$。

(2)从邻域移动中挑选一个能改进当前解 $x$ 的移动 $s \in s(x)$，并将求得的解存放在禁忌表中。

(3)在主机端通过调用 cudaMalloc()函数为 GPU 端变量分配存储空间，用来存储变量、可行解以及邻域适应结构等内容。

(4)在主机 CPU 端通过调用 cudaMemcpy()函数将相应的变量、可行解等内容拷贝到 GPU 的显存中。

(5)从邻域 $s(x)$ 中产生新解 $x'$ 并重复以下操作，直到达到最大的迭代次数或寻找到最优解便停止操作。

1. 对候选解进行增量式评估；

2. 将评估的结果插入到邻域适应结构中；

3. 所有的候选解评估结束后，通过调 cudaMemcpy()函数将邻域相应结构中的内容拷贝到主机端的内存中；

4. 选择其中最优且可被接受的解作为当前的最优解；

5. 更新禁忌表，即将当前系统选择的最优解的相关信息存储到禁忌表中。

### 3.3 并行遗传算法

遗传算法(Genetic Algorithm)是由 Holland 等人受达尔文生物进化论的启发于 1973 年提出的。它是一种通过模拟生物界自然选择和遗传机制的随机搜索算法，是一种比较通用的优化算法。算法主要包括个体适应度评价、选择、交叉、变异等遗传操作。其中选择算子体现了适者生存的原则；交叉算子是组合父代群体中的优良信息，产生新的后代，具有遗传功能；变异算子的作用是保持群体中基因的多样性。遗传算法是一个多点并行迭代的过程，在每次迭代中都重复进行如下操作：将一组以一定的基因形式描述的候选解进行交叉、变异操作和适应环境能力的评价；选取参与产生后代的候选解。重复该过程，直到满足某种收敛准则。

目前 GA 在小规模的应用问题上得到了较为广泛的应用，而且优化效果良好，而对于大规模的多变量求解任务，GA 算法个体适应度函数值的计算与评估需要耗费大量的计算时间，通过对适应度函数值实现并行计算则可以大大提高算法的执行效率。当前遗传算法的并行模型可以分为 3 类[12]，即全局主从式并行模型、独立粗粒度并行模型、分散细粒度并行模型。全局主从式模型并行遗传算法系统由一个主处理器和若干从处理器组成，主处理器监控整个染色体种群，并基于全局统计执行的选择操作，各从处理器接受来自主处理器的个体并进行重组交叉和变异，产生新一代个体，然后计算适应度再把结果传给从处理器。由于基于全局处理，因此系统开销大，并行效率不高，系统的稳定性也不理想。独立粗粒度模型将群体分成若干个子群，并分配给各自对应的处理器。每个处理器不仅独立计算适应度，而且独立进行选择、交叉、变异的操作，还要定期地互相传送适应度最好的个体，从而加快满足终止条件。独立型分散细粒度模型：为群体中的每个个体分配一个处理器，每个处理器进行适应度计算，而选择、交叉、变异的操作仅在与之相邻的一个处理器之间互传个体进行遗传操作。目前基于 GPU 的细粒度并行模型是一种新的优化方法，文献[13]提出了基于 GPU 的进化规划算法，得到了很好的优化效果，但由于 PCIE 总传输带宽限制，影响了 CPU 与 GPU 数据交换的速度，加速效果一般。文献[14]提出使用实数编码进行相关交叉和变异操作的 GPU 并行遗传算法，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。文献[15]中提出了一种改进的基于 GPU 加速细粒度并行遗传算法的实现方法，其通过 GPU 约减技术减少了 CPU 与 GPU 的数据交换，提高了算法性能，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。基于 GPU 的 GA 算法流程如图 5 所示，具体的并行过程如下：

(1)首先初始化算法参数及其各个变量，并分配存储空间。

(2)随机初始化一组种群个体。

(3)对所有的种群个体进行适应值计算与评估，计算和评估过程由 GPU 上的每一个线程 thread 并行实现。

(4)将适应值计算结果通过调用 cudaMemcpy()函数，传回到 CPU 端。

(5)判断算法收敛准则是否成立，如果成立，则结束搜索，输出最优解，否则，执行下面的步骤。

(6)根据比较适应值，选择一定数量的种群个体，并将其复制。CPU 端主机复制后，通过调用 cudaMemcpy() 函数将新一代个体拷贝到 GPU 的显存中。

(7)对新一代种群个体按照交叉概率 $Pc$ 从当前种群中选择一定数量的个体执行交叉操作，操作结束后产生下一代新的个体。

(8)对当前最新种群个体按照变异概率 $Pm$ 从当前种群中选择一定数量的个体执行变异操作，操作结束后产生下一代新的个体。

(9)返回步骤(3)，重新对当前新的个体进行适应值评估与遗传操作。

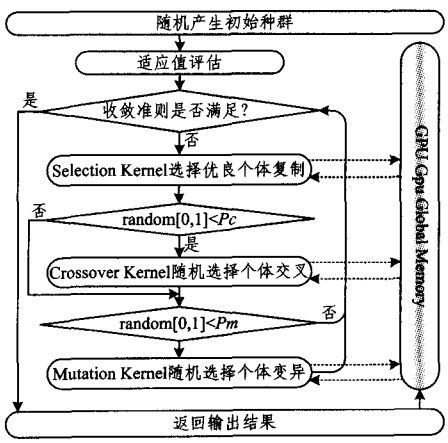


图 5 基于 GPU 的并行遗传算法

### 3.4 并行粒子群优化算法

粒子群优化算法 PSO(particle swarm optimization) 是 Eberhart 和 Kennedy 在研究鸟类的群体行为建模和仿真研究中得到启发而提出的算法。它是一种新兴的基于群体智能的启发式全局搜索算法。粒子群优化(PSO)最初是处理连续优化问题的，目前其应用已扩展到组合优化问题[16]。PSO 初始化为一群随机粒子，算法搜索空间中的每一个粒子(Particle)都视为一个候选解，空间中所有粒子都有一个由被优化的函数决定的适应值，根据适应值来控制空间粒子迭代产生最优

解。在每一次迭代中，粒子通过跟踪两个极值来更新自己的位置和速度。其中第一个是粒子本身所找到的最优解，即个体极值 *pBest*，另一个是种群目前全局最优解，即全局极值 *gBest*，*gBest* 产生于种群中个体极值的历史最优解。搜索过程中每个粒子根据式(1)、式(2)更新当前速度和位置。

$$\begin{cases} v_{ij}(t+1)=v_{ij}(t)+c_1r_1(P_{ij}(t)- \\ \quad X_{ij}(t))+c_2r_2(P_{gi}(t)-X_{ij}(t)) \end{cases} \tag{1}$$

$$X_{ij}(t+1)=X_{ij}(t)+V_{ij}(t+1) \tag{2}$$

每个粒子都有一个速度决定它们飞翔的方向和距离，粒子个体就跟随当前的最优粒子在解空间中进行搜索。其中 $v_{ij}$ 表示空间中第 $i$ 个粒子在 $d$ 维空间内的飞行速度矢量，$x_{ij}$ 表示空间第 $i$ 个粒子在 $d$ 维空间中的位置矢量。下标 $j$ 表示粒子的第 $j$ 维，$i$ 表示第 $i$ 个粒子，$c_1$ 和 $c_2$ 称为学习因子。每个粒子到目前为止的最好位置为 $p_i$(称为粒子个体极值)，现在所处的位置 $x_i$ 以及到目前为止所有粒子的最好位置 $p_g$(称为粒子全局极值)，速度 $v_{ij}$ 通常限定于一定的范围内。粒子不断调整自己的位置以找到问题的最优解。

粒子群算法的瓶颈在路径搜索过程，随着问题规模扩大，蚂蚁数目也将会增加，计算复杂度和时间也会相应的增加。对粒子群优化算法的改进主要体现在串行优化和并行优化。串行优化的方法有增加惯性权重和收敛因子、依据一定的标准为整个群体或某些微粒的状态量重新赋值、使用新的位置和速度更新等式及新的群体组织结构，归纳起来可分为：粒子群初始化优化、邻域拓扑优化、参数选择优化和混合策略优化[17]。并行优化中，将路径搜索过程在 GPU 上实现并行化，即分别让多只蚂蚁在 GPU 核处理器上进行并行搜索，依据 CUDA 的单指令多线程架构，每一只蚂蚁对应一个线程进行并行搜索，以提高搜索效率。基于 GPU 的标准并行粒子群优化流程如图 6 所示，具体步骤如下：

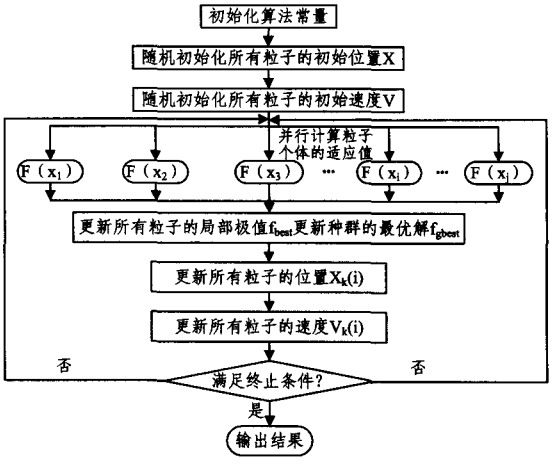


图 6　基于 GPU 的并行粒子群算法

(1)首先设定算法的执行参数和变量的定义($c_1$，$c_2$，粒子的位置、速度等信息)。

(2)通过在 CPU 和 GPU 端分别调用 malloc()函数和 cudaMalloc()函数，为参数、变量分配存储空间。

(3)初始化粒子的初始位置和初始速度。

(4)在主机端通过调用 cudaMemcpy()函数将粒子的数据信息传递到 GPU 显存上。

(5)在 GPU 设备端对粒子进行适应值评估，重复执行下列操作。

(6)for $i$=1 to Max do(1－4)

1. 计算每个粒子的适应值；

2. 更新每个粒子的局部最优值 *pBest*；

3. 更新粒子的全局最优值 *gBest*；

4. 更新每个粒子的位置和速度。

(7)在 GPU 端通过 cudaMemcpy()函数将搜索到的最优粒子信息拷贝到主机端的内存中，算法搜索结束。

### 3.5　并行神经网络算法

人工神经网络(Artificial Neural Network)最早是由 Warren McCulloch 和 Walter Pitts 于 1943 年基于生物学中神经网络的基本原理而建立的一种模仿人脑工作的计算模型[18]，目前神经网络模型已有近 40 种，典型的神经网络有反传网络、感知器、自组织映射、Hopfield 网络、波耳兹曼机、适应谐振理论等。根据连接的拓扑结构，神经网络模型可以分为：前向传播神经网络和后向反馈神经网络。前向神经网络没有反馈，可以用一个有向无环路图表示。反馈神经网络即网络内神经元间有反馈，可以用一个无向的完备图表示。这种神经网络的信息处理是状态的变换，可以用动力学系统理论处理。Hopfield 网络、波耳兹曼机均属于反馈神经网络的类型。人工神经元是对生物神经元功能和特性的数学抽象。基本的神经网络分为输入层、隐含层和输出层三部分，隐含层数目可以设置为多个，具体如图 7 所示。神经网络通过学习获取知识，并将知识分布存储在连接权重因子中，然后以训练好的神经网络对未知的样本进行分析预测。当神经网络中神经元有多个输入 $x_i$ 和单个输出 $y_j$ 时，输入和输出的数学表达式为：

$$\begin{cases} s_j=\sum_{i=1}^{m}w_{ij}x_i-\theta_j & (3) \\ y_j=f(s_j) & (4) \end{cases}$$

式(3)和式(4)中，$\theta_j$ 为阈值，$w_{ij}$ 为从神经元 $i$ 到神经元 $j$ 的连接重因子，$f()$为称激励函数或传递函数，该函数可为线性函数也可以是非线性函数。

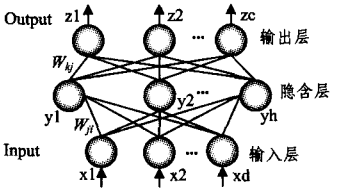


图 7　三层神经网络基本结构

20 世纪 80 年代，Hopfield[19,20]首次将人工神经网络应用于组合优化问题，提出了反馈神经网络，并引入了能量函数，使网络的平衡态与能量函数极小值解相对应。此外也证明了在高强度连接下的神经网络依靠集体协同作用能自发产生计算行为。由于人工神经网络是模拟人脑大量连接神经元的运作机理，因此具有大量连接的分布式并行计算结构[21]。ANN 在结构上的并行性使神经网络的信息存储必然采用分布式方式，即信息不是存储在网络的某个局部，而是分布在网络所有的连接权中。一个神经网络可存储多种信息，其中每个神经元的连接权中存储的是多种信息的一部分。神经网络内在的并行性与分布性表现在其信息的存储与处理都是空间上分布、时间上并行的。当前神经网络的并行方法主要有分布式并行、异构模式并行、系统集群并行等。文献[22]提出了一种

新颖的神经网络的并行计算体系结构和计算网络权函数的训练算法，算法训练的是神经网络的权函数而不是传统意义上的连接权值。文献[23]提出了基于分布计算环境下采用主从式结构的集中式粗粒度并行进化神经网络模型。文献[24]采用 GPU 实现了针对任意的卷积神经网络的训练和分类阶段并行化，并获得了较好的加速比，实现了对神经网络中耗费时间较多的训练阶段的并行化。文献[25]中采用 GPU 实现了对自组织神经网 SOM(self-organizing-maps)和多层神经感知器 MLP(multi-layer-perception)的并行计算。图 8 给出了基本 BP 神经网络的并行流程图，具体步骤如下：

(1) 首先初始化算法中神经网络的权值和阈值，以及算法中其它相应变量。

(2) 在 CPU 和 GPU 端通过调用 malloc()函数为不同架构下的各个参数和变量分配存储空间。

(3) 给定连续输入样本 $x_i$(其中 $i=0,1,2,\cdots,n-1$)和目标输出 $C(t)$。

(4) 将已知样本数据通过 cudaMemcpy()函数将训练数据拷贝到 GPU 端显存中。

(5) GPU 中的线程并行执行 Kernel 函数，计算实际输出 $y(t)=f(\sum_{i=0}^{n-1}(\omega_i(t)x_i(t)-\theta)$，其中 $f(*)$为神经元激励函数。

(6)调整函数的权值，$\omega_i(t+1)=\omega_i(t)+\varphi[c(t)-y(t)]+x_i(t)$。

(7)将最优神经网络权值和阈值输出，完成训练。

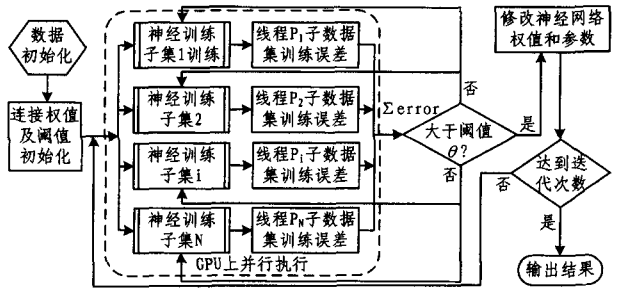


图 8 BP 神经网络模式并行算法

## 4 实验统计与分析

通过几种典型的基于 GPU 架构的实现算例，依次对比分析算法在不同型号的主机 CPU 和 GPU 的环境下，算法执行的实验结果，通过分析实验结果的差异挖掘出 GPU 性能优化提升的关键点，归纳出当前并行算法在基于 GPU 并行架构上的优化思路和基本策略。

表 1 模拟退火实验结果统计

| 数据集 | Cluster Size=1 | Cluster Size=4 | Cluster Size=16 |
|---|---|---|---|
| | Seedup | Seedup | Seedup |
| b17_1 | 10.16 | 21.97 | 12.29 |
| b18_1 | 10.69 | 23.60 | 12.76 |
| b18 | 10.03 | 22.47 | 13.31 |
| B19_1 | 9.94 | 25.31 | 14.97 |
| Leon2 | 6.60 | 14.89 | 9.92 |
| Leon3mp | 7.03 | 15.28 | 9.04 |
| netcard | 7.40 | 16.82 | 9.69 |

(1)并行模拟退火算法实验：当前针对模拟退火的并行研究主要集中在 3 个方面，即数据级并行、任务级并行以及两者的混合并行。基于 Alexander Choong 等人[26]的基本并行策略，在 64bit 的 Linux 系统环境下分别对比测试了 CPU 和 GPU 对模拟退火算法的执行状况。其中 CPU 采用的是 Intel Core2 Quad 处理器，该处理器主频为 2.66GHz，拥有 2GB 内存存储空间。GPU 采用的是 NVIDIA GTX280，主频为 1.35GHz，拥有 1GB 显存。以 IWLS Benchmarks 作为实验测试对象，分别在不同的数据集上进行测试，得到了如表 1 所列测试结果。

(2) 禁忌搜索算法实验：基于 Michah Czapinski[27] 等人的并行策略，在 Tesla C1060 架构上将并行禁忌搜索算法应用于车间流水线队列调度中。串行程序在 Intel Xeon 处理器上进行，主频为 3.0Hz，内存为 2GB 存储空间。并行程序在 NVIDIA Tesla C1060 上进行测试，主频为2.8GHz，4GB 专用内存，相应的主机为 Intel Core2 Duo 处理器，主频为2.5GHz，4GB 的内存空间。Tesla C1060 包含 30 个流多处理器，每个处理器上含有 8 个标量处理器，计算能力为 1.3，单精度操作峰值速度达 933GFlops，双精度操作峰值速度为 78GFlops，实验由 Taillard 提供的并被广泛采用的流水线调度数据测试基准进行测试，此外通过 Taillard 方法又随机生产了部分数据集来进行测试。算法测试中选取 task(任务)数目为 $n$，其中的取值集共设定为 10 个值，$n\in\{10,20,40,50,60,100,120,200,300,500\}$，选取 machine(机器)数目为 $m$，$m\in\{5,10,15,20,25,30\}$，实验测试属性分别为不同任务数和不同机器数下的串行和并行各自执行的时间，单位为秒。实验对比结果如表 2 所列。

表 2 禁忌搜索算法实验结果统计

| Tasks/个 | machine=5 | | machine=15 | | machine=30 | |
|---|---|---|---|---|---|---|
| | GPU | CPU | GPU | CPU | GPU | CPU |
| 10 | 0.10 | 10.10 | 0.18 | 0.18 | 0.31 | 0.29 |
| 20 | 0.46 | 0.44 | 1.11 | 1.02 | 2.12 | 1.64 |
| 40 | 2.87 | 2.68 | 7.35 | 6.02 | 10.20 | 9.17 |
| 50 | 5.04 | 4.81 | 10.85 | 10.57 | 15.11 | 15.74 |
| 60 | 8.16 | 7.76 | 15.72 | 16.74 | 20.40 | 24.24 |
| 100 | 17.33 | 23.89 | 26.46 | 33.94 | 29.51 | 37.67 |
| 120 | 23.22 | 36.12 | 31.01 | 51.40 | 32.65 | 54.05 |
| 300 | 34.84 | 60.86 | 37.48 | 67.63 | 27.46 | 53.83 |
| 500 | 36.48 | 65.92 | 39.17 | 73.10 | 6.37 | 35.79 |

(3)遗传算法实验：基于 Rajvi Shah 等人[28]的并行策略，采用 Tesla S1070 架构对遗传算法中的适应值评估阶段和遗传操作阶段进行 GPU 多线程并行处理。该架构含有 30 个流多处理器，每个处理器含有 8 个流标量处理核心，算法执行结果同 GAlib 比对，其中该串行执行过程在 Intel Core2 Duo E7500，主频为 2.93GHz 的处理器上执行，经过 $N$ 代实验测试得出的结果如表 3 所列。

表 3 遗传操作时间测试结果

| N/代 | CPU/秒 | GPU/秒 | 加速比 |
|---|---|---|---|
| 100 | 0.141 | 0.046 | 3.01 |
| 1000 | 1.629 | 0.053 | 30.38 |
| 10000 | 21.609 | 0.209 | 103.19 |
| 100000 | 497.927 | 1.724 | 286.04 |
| 1000000 | 7233.716 | 4.727 | 1530.14 |

(4)粒子群算法实验：基于 You Zhou 等人[29]提出的对传统并行 PSO 算法的思路在 GPU 上并行执行，CPU 为 Intel Core2 Duo 处理器，主频为 2.20GHz，内存为 2GB 存储空间；GPU 为 NVIDIA Geforce 8600GT，实验中选择 4 个基准测试函数作为测试对象，测试结果具体如图 9 所示。

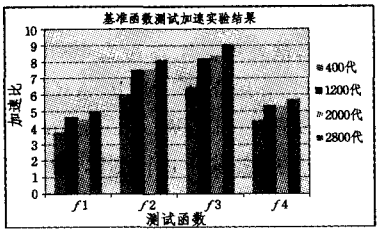


图 9 基准函数测试加速实验结果

(5)BP 神经网络算法实验:Noel Lopes 等人[30]将传统的 BP(Back Propagation)反向传播神经网络改造成为多重的反向传播网络 MBP,并分别将 BP 以及改造后的 MBP 算法在 GPU 上进行了并行实现,算法减少了神经网络学习在训练阶段的耗时,实验中选取了两个测试基准函数,分别如下:"$f(x)=\sin(x)/x$"(Benchmark1)和"two-spirals"(Benchmark2),实验测试平台采用的处理器分别为 CPU Core2 6600,主频为 2.4GHz,GPU 有两个,为 NVIDIA GeForce 8600GT(GPU1)和 NVIDIA GTX280(GPU2),MBP 的串并行测试结果如表 4 所列。

表 4 MBP 算法测试结果

| 函数 | $N_{hl}$/个 | CPU/秒 | GPU1/秒 | GPU2/秒 |
|---|---|---|---|---|
| MBP (Benchmark1) | 5 | 239.598 | 55.007 | 39.448 |
| | 7 | 261.764 | 45.746 | 32.789 |
| | 9 | 340.272 | 50.277 | 33.254 |
| MBP (Benchmark2) | 15 | 629.321 | 63.637 | 19.159 |
| | 20 | 662.434 | 67.880 | 17.373 |
| | 25 | 864.878 | 89.598 | 20.140 |

通过上述 5 种典型的基于 GPU 的现代并行优化算法实验结果的比较可以发现,当算法处理的任务平均数目较少或迭代的次数较小时,GPU 并没有带来较为明显的性能提升,而随着处理数据的规模增大,问题迭代规模增大时,GPU 的加速效果明显,最大可以达到 1～3 个数量级的加速比。

综上分析,造成性能差异的根本原因可以概括为两个方面:其一是当数据量处理的数目较少时,该模型的时间主成分由数据通信时间所主导,例如 GPU 中 Global memory 的数据访问延迟约为数百个时钟周期,较少的线程计算不能有效地隐藏相对较大的时间延迟。其二,当数据量较大时,该模型的时间主成分由计算时间所主导,而大量多线程间的并行执行可以很好地隐藏数据通信时间延迟,并且数据处理的吞吐量越大,时间延迟效果越好。基本的优化策略为:第一,从软件角度分析,要尽可能将算法中耗用时间较长的部分进行并行化,减少算法中条件分支语句,避免递归出现。第二,从硬件角度分析,在使用 GPU 时要尽可能让多的线程参与数据计算,通过共享存储器来实现 block 块内线程间的高效低延迟通讯,避免高延迟线程全局同步操作等,同时减少线程访存和数据传输的次数。合理使用 GPU 内不同层次的存储器。第三,协同优化,采用多 GPU 的协同并行计算,设计合理的数据组织结构,使多 GPU 处理峰值贴近理论最佳性能。

**结束语** 结合现代优化并行优化算法,重点研究了基于 CUDA 的并行改进与实现。通过不同环境下每种算法串并行实验测试数据的比较分析发现,基于 CUDA 的优化算法在执行效率上都获得了不同程度的提升。上述优化算法在具体实现过程中虽然存在差异,但各算法核心主线相同,都是从随机的可行初始解出发,采用迭代改进的策略,去逼近问题的最优解;同样,在 GPU 并行实现原理上,均是选择程序中高密度独立耗时计算的模块经由 GPU 调用大量的线程 thread 执行单元实现并行处理效果。GPU 带来性能提升的同时,应当注意到当 GPU 端处理的数据量相对较小时,其性能将不会得到大幅提升,执行效率有时会有所降低。当前 GPU 和 CPU 间异构模式下的数据通信仍然是制约算法性能的瓶颈。

目前现代优化算法逐渐朝向高度多元、并行化和集群智能优化方向发展,未来随着 CPU 多核处理器的拓展以及 GPU 芯片集成技术的优化改进,算法执行架构将会更加趋向于 CPU/GPU 集群协作模式。同时,基于 GPU 的应用也将逐渐融入高密度、海量数据实时信息处理中。在解决当前较为复杂 NP 组合优化问题时,基于 CPU/GPU 异构集群模式的并行部署方式相对传统的 CPU 集群部署方式优势将更加突出。

## 参 考 文 献


[1] 谢金星,邢文训. 现代优化算法(第 2 版)[M]. 北京:清华大学出版社,2005

[2] Garey M, Johnson D S. Computers and Intractability: A Guide to the Theory of Np-Completeness[M]. W. H. Freeman & Co., New York, USA, 1979

[3] 滕人达,刘青昆. CUDA、MPI 和 OpenMP 三级混合并行模型的研究[J]. 微计算机应用,2010,31(9):63-69

[4] 左颢睿,张启衡,等. 基于 GPU 的并行优化技术[J]. 计算机应用研究,2009,26(11):4115-4118

[5] 王凌. 智能优化算法及其应用[M]. 北京:清华大学出版社,2004

[6] Kang L S, Xie Y, You S Y, et al. Nonnumerical Parallel Algorithm-Simulated Annealing Algorithm [M]. Beijing: Science Press, 1994

[7] 吴浩扬,常炳国,朱长纯. 基于模拟退火机制的多种群并行遗传算法[J]. 软件学报,2000,11(3):416-442

[8] 谢云. 模拟退火算法综述[J]. 微计算机信息,1998,15(5):63-65

[9] 都志辉,李三立,吴梦月. 混合 SPMD 模拟退火算法及其应用[J]. 计算机学报,2001,24(1):91-98

[10] Chen G L, Wu J M, Zhang F. Concurrent computer architecture [M]. Beijing: Higher Education Press, 2002

[11] Luong V, Melab N, Talbi E-G. Multi-start local search algorithms on GPU [C]// International Conference on Metaheuristics and Nature Inspired Computing (META). Djerba: Tunisia, 2010

[12] 张雪东,饶元,元昌安. 应用禁忌基因表达式编程提高模型精度[J]. 计算机工程与应用,2009,45(28):35-28

[13] 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

[14] Qi Z Y, Chong C C, Zhi G P. Parallel genetic algorithms on programmable graphics hardware[J]. Advances in Natural Computation of LNCS, Springer, 2005, 3162: 1051-1059

[15] 李建明,迟忠先,万单领. 一种基于 GPU 加速细粒度并行遗传算法的实现方法[J]. 控制与决策,2008,23(6):697-704

[16] 纪震,廖惠连,吴青华. 粒子群算法及应用[M]. 北京:科学出版社,2009

[17] 朱丽莉,杨志鹏,袁华. 粒子群优化算法分析及研究进展[J]. 计算机工程与应用,2007,43(5):24-27

[18] McCulloch W, Pitts W. A logical calculus of the ideas imminent

in nervous activity[J]. Bulletin of Mathematical Biophysics, 1943,5:115-133

[19] Wen J,Zhao J L,Luo S W. The improvements of BP neural network learning algorithm[C]// Signal Processing Proceedings, 2000 WCCC-ICSP 2000. 5th International Conference. 2000(3): 1647-1649

[20] Hopfield J J. Neural networks and physical systems with emergent collective computational abilities[C]//Proc Natl Acad Sci. USA,1984:2554-2558

[21] 郭文生,李国和. 人工神经网络在并行计算机集群上的设计研究[J]. 计算机应用与软件,2010,27(5):12-14

[22] 张代远. 基于分布式并行计算的神经网络算法[J]. 系统工程与电子技术,2010,32(2):386-391

[23] 于漫,朱岩. 集中式粗粒度分布并行模型和并行进化神经网络[J]. 系统工程理论与实践,2003,23(6):74-79

[24] Strigl D, Kofler K, Podlipnig. Performance and scalability of GPU-based convolutional neural networks[C]// Proc in 18th Euromicro Conference on Parallel, Distributed, and Network-Based Processing. 2010

[25] Luo Z W,Liu H Z,Wu X C. Artificial Neural Network Computation on Graphic Process Unit[C]//Proceedings of International Joint Conference on Neural Networks. Montreal,Canada,2005

[26] Choong A, Beidas R, Zhu J. Parallelizing Simulated Annealing-Based Placement Using GPGPU[C]//Proc of the international conference on Field Programmable Logic and Applications. IEEE,2010

[27] Czapinski M, Barnes S. Tabu Search with two approaches to parallel flowshop evaluation on CUDA platform [J]. Parallel Distribute Computer,2011(71):802-811

[28] Shah R,Narayanan P J,Kothapalli K. GPU-Accelerated Genetic Algorithms [C/OL]. http://cvit. iiit. ac. in/papers/Rajvi10-GPU. pdf,2011-07-20

[29] Zhou Y,Tan Y. Gpu-based parallel particle swarm optimization [C]//Proceedings of the IEEE Congress on Evolutionary Computation,CEC 2009. Trondheim,Norway,2009:1493-1500

[30] Lopes N,Ribeiro B. GPU implementation of the multiple back-propagation algorithm[C]//Proceedings of Intelligent Data Engineering and Automated Learning, LNCS. Springer-Verlag, 2009:449-456

(上接第 295 页)

虑,得到了 DDR3-800 Mbps 写操作时建立、保持时间的时序裕量。最后计算得到的建立、保持时间总的时序裕量为 164ps,这说明即使在工艺条件 SS 情况下以及对影响时序的因素都做保守的估计,DDR3-800Mbps 的时序设计仍然能够留有裕量,因此对时序的预算视为通过。

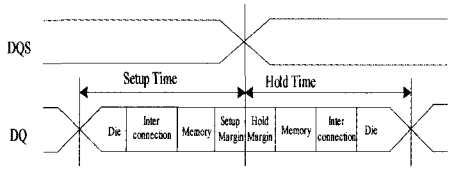


图 8 DDR3 写操作的数据窗口

表 2 Worst Case 下 DDR3-800 写操作建立、保持时序裕量预算

| TimingComponent | Description | Uncertainty | Units | Method |
|---|---|---|---|---|
| DDR3 Controller | ASIC PLL Jitter | 20 | ps | Estimate |
| | DQS Duty Cycle | 75 | ps | Simulation |
| | ASIC DLL Jitter | 20 | ps | Estimate |
| | DQ Duty Cycle | 75 * 2 | ps | Simulation |
| | ASIC SSO | 150 | ps | Simulation |
| 芯片输出驱动器总时序偏斜 | | 415 | ps | Calculate |
| Interconnect Timing | Package Interconnect Routing Skew | 20 | ps | Estimate |
| | PCB Interconnect Routing Skew | 20 | ps | Estimate |
| | PCB Interconnect SSN/Crosstalk/ISI | 218 | ps | Simulation |
| 互连产生的总的时序偏斜 | | 258 | ps | Calculate |
| SDRAM Timing Requirements | SDRAM Setup time | 125 | ps | SPEC |
| | Slew rate derating | 88 | | |
| | SDRAM Hold time | 150 | ps | SPEC |
| | Slew rate derating | 50 | | |
| 总的 SDRAM 建立保持时间需求 | | 413 | ps | Calculate |
| Total Uncertainty | | 1086 | ps | Calculate |
| Bit time at 800Mbps | | 1250 | ps | SPEC |
| Worst Case Timing Margin (Setup+Hold time Margin) | | 164 | ps | Calculate |

**结束语** 本文针对某自研处理器及服务器主板的 DDR3 存储系统设计,简要分析了 DDR3 源同步信号传输的基本原理,使用时域信号仿真工具量化分析了 DDR3 系统通道中影响时序的主要因素,对 DDR3 的写操作时序进行了分析,并给出了时序裕量计算表。分析表明影响 DDR3 时序的主要因素为芯片 IO 信号的时序偏斜和板级信号串扰。仿真结果表明,DQ 信号和 DQS 信号的占空比失真随着信号 ODT 值的改变和同时开关的 I/O 数目增加将会加剧 3%～5%左右,而串扰引入的时序偏斜最大可达 218ps。只有通过严格的仿真设计,才能实现 DDR3 系统的时序分析与裕量计算。

## 参 考 文 献

[1] JEDEC DDR3 SDRAM Specification[S]. JESD79-3E. 2010. 07

[2] Brennan C,Tudor C,Schroeter E,et al. Signal Integrity and PCB layout considerations for DDR2-800 Mbps and DDR3 Memories [C]//CDNLIVE Silicon Valley:Cadence. 2007

[3] Raj Mahajan,MemCore Inc. Memory Design Consideration when Migrating to DDR3 interface from DDR2[C]//Proc of Designcon Santa Clara:DesignCon. 2007

[4] Chuang H-H,Wu Shu-jung,Hong Ming-zhang,et al. Power Integrity Chip-Package-PCB Co-Simulation for I/O Interface of DDR3 High-Speed Memory[C]//IEEE Electrical Design of Advanced Packaging and Systems Symposium. 2008

[5] Ren Ji-hong,Oh D,Chang S,et al. Statistical Link Analysis of High-Speed Memory I O Interfaces during Simultaneous Switching Events[C]//IEEE Electrical Design of Advanced Packaging and Systems Symposium. 2009

[6] Xu Tao. IBIS EBD Modeling, Usage and Enhancement-An Example of Memory Channel Multi-board Simulation[C]//Proc of Asian IBIS. Shanghai,IBIS,2008

[7] Shen Huang-hui,Wang Zhen-song,Zheng Wei-min. PCB Level SI Simulation Based on IBIS Model for High speed FPGA System[C]// The Ninth International Conference on Electronic Measurement & Instruments. 2009