一种基于GPU的并行算法功耗评估方法

王卓薇[1,2] 程良伦[1] 赵武清[2]
(广东工业大学计算机学院 广州 510006)[1] (武汉大学计算机学院 武汉 430074)[2]

摘 要 随着软件和硬件的不断发展,图形处理器(GPUs)已经广泛用于通用计算领域,并作为加速器来协助CPU加速程序的运行。为了追求高性能,GPU往往包含成百上千个核心运算单元,高密度的计算资源使其在性能远高于CPU的同时功耗也高于CPU,因此功耗问题已经成为制约GPU发展的重要问题之一。分析了并行程序在GPU上运行时消耗的功耗,提出了并行算法在GPU上运行的功耗评估方法,接着通过并行前缀求和算法对该方法进行了详细的论述与分析。在实验部分通过稀疏矩阵向量乘算法的实际应用对该方法的正确性以及敏感性进行了证明与分析。结果表明,对于给定的程序,在满足性能要求的前提下,最优线程块数、存储访问方式以及任务分配顺序是影响系统功耗的关键因素。

关键词 GPU,并行算法,功耗,性能
中图法分类号 TP311 **文献标识码** A

GPU Parallel Algorithm Based on Power Evaluation Methods

WANG Zhuo-wei[1,2] CHENG Liang-lun[1] ZHAO Wu-qing[2]
(Faculty of Computer, Guangdong University of Technology, Guangzhou 510006, China)[1]
(School of Computer, Wuhan University, Wuhan 430074, China)[2]

Abstract With the continuous development of hardware and software, Graphics Processor Units(GPUs) have been used in the general-purpose computation field. They have emerged as a computational accelerator that dramatically reduces the application execution time with CPUs. To achieve high computing performance, a GPU typically includes hundreds of computing units. The high density of computing resource on a chip brings high power consumption. Therefore power consumption has become one of the most important problems for the development of GPUs. This paper analyzed the energy consumption of parallel algorithms executed in GPUs and provided a method to evaluate the energy scalability for parallel algorithms. Then the parallel prefix sum was analyzed to illustrate the method for the energy conservation, and the energy scalability was experimentally evaluated using Sparse Matrix-Vector Multiply(SpMV). The results show that the optimal number of blocks, memory choice and task scheduling are the important keys to balance the performance and the energy consumption of GPUs.

Keywords GPU, Parallel algorithms, Energy conservation, Performance

1 引言

随着GPU从传统的3D图像渲染到通用计算领域的扩展,GPU发挥出的强大计算能力已经震撼全世界。然而由于GPU中包含着高密度并行计算单元,使得GPU在性能高于CPU的同时功耗也高于CPU,研究人员开始关注其在功耗、可靠性以及稳定性方面的表现。GPU高于CPU的绝对功耗不仅导致了芯片的可靠性下降,而且随之带来的散热问题也影响了系统的稳定性。此外,移动或者嵌入式设备对GPU的广泛应用也对其高性能提出了严峻的挑战。因此GPU的高功耗已经成为制约GPU发展的重要问题。

在传统的低功耗研究中,大量的工作都专注于计算系统处理器的能量消耗,很少有人关注存储器的能量消耗。GPU除拥有成百上千个处理器单元,还拥有一个独立的DRAM存储器,而程序由于访存所消耗的能量不容小觑。目前高性能计算平台通常采用的是CPU+GPU并行架构,CPU与GPU协同工作,各司其职,在考虑GPU的能耗计算时,也应考虑系统的空闲功耗。因此基于GPU的并行算法能耗评估主要从3个方面进行考虑:计算能耗、存储能耗、空闲能耗。

本文在基于并行程序分析的基础上,综合考虑GPU功耗,建立了GPU环境下并行算法的功耗模型,提出了GPU并行算法的功耗评估方法,并以并行前缀加和算法为例对该功耗评估方法进行详细分析说明。实验结果表明,该功耗评估方法能够准确估算并行算法在GPU上的能量消耗,并且能够帮助编程人员设计和开发更加节能的高效的并行算法。

本文第2节介绍相关工作;第3节介绍PEM模型;第4

到稿日期:2013-01-21 返修日期:2013-04-14 本文受广州市科技项目(2012Y2-0031)资助。
王卓薇(1985—),女,博士,讲师,主要研究方向为高性能计算,E-mail:wangzhuowei0710@163.com;**程良伦**(1964—),男,博士,教授,主要研究方向为物联网、信息物理融合系统;**赵武清**(1986—),男,博士,主要研究方向为云计算。

节根据 PEM 模型提出 GPU 功耗模型；第 5 节给出了并行算法在 GPU 执行环境中能耗的评估方法；第 6 节主要通过并行前缀加和算法对能耗评估方法进行分析说明；实验的性能测评与分析工作由第 7 节给出；最后总结全文。

2 相关工作

GPU 功耗模型研究目前还处于起步阶段，许多研究基本都集中在能耗测评分析阶段。Collange 等人通过使用测量统计方法，针对不同应用程序在 GPU 平台运行，测试系统是如何消耗能量的[1]。Rofouei 等人[2]将应用程序在 CPU 平台上消耗的能耗与在 GPU 平台上消耗的能耗进行对比，发现当系统整体性能超过一个特定阈值后，系统就能够有效地达到节能的状态。但这一特定阈值需要经过实验进行统计测试分析才能得到。Ma X. 等人[3]通过统计分析的方法来建立 GPU 功耗模型，该模型的主要目标就是预测应用程序在 GPU 上运行时所消耗的能量，如果能量消耗过多，则需要提供有效策略来降低功耗。Takizawa 等人[4]提出了一个编程框架 SPART，其在系统运行的状态下可以动态选择能量消耗较低的处理器运行方式，并且考虑 CPU 与 GPU 之间消耗能量的差异，达到降低系统功耗的目的。林一松等人[5]根据 Hong 等人在 2009 年国际体系结构年会上提出的性能分析模型，提出基于并行度分析模型的 GPU 功耗优化方法，即根据 CWP(计算并行度)和 MWP(存储并行度)分别计算 GPU 处理器和存储器的频率调节因子，找出应用程序的性能瓶颈，在满足系统性能需求的前提下，使得系统的整体功耗降到最低。

综上所述，GPU 功耗研究领域缺少根据 GPU 存储模型建立精确的功耗模型，而这正是本文研究的出发点。本文基于 GPU 功耗模型提出了一种评估并行算法的能量消耗方法，通过对其参数的敏感性分析，为未来 GPU 体系架构的设计提供参考；选择适当的线程块数、存储访问方式以及任务分配，可以对程序行为进行准确分析，从而获得较好的优化效果。

3 存储模型

PEM(Parallel External Memory)模型是一个拥有 P 个处理器和两级存储模式的计算模型，两级存储包括外部存储器(Main Memory)和 P 个内部存储器(caches)。外部存储器可以被所有处理器共享，而单个处理器都拥有自己的 cache。每个 cache 的大小为 M，被分成大小为 B 的块。在 PEM 模型中，每个处理器不能访问属于其他处理器的 cache。如果要对数据执行任何操作，处理器必须首先在其 cache 中存储数据，数据则在外部存储器和大小为 B 的内部存储器之间进行传输。

本文根据 PEM 模型提出 GPU 环境下并行算法的存储模型，如图 1 所示。该模型中也有两级存储，一个是全局存储器，另外一个是共享存储器。一个网格中所有线程块都能访问全局存储器，而每个线程块同时拥有自己的共享存储器。在该存储模型中，每个线程块不能访问属于其他线程块的共享存储器。所有数据在全局存储器和共享存储器之间进行传输。

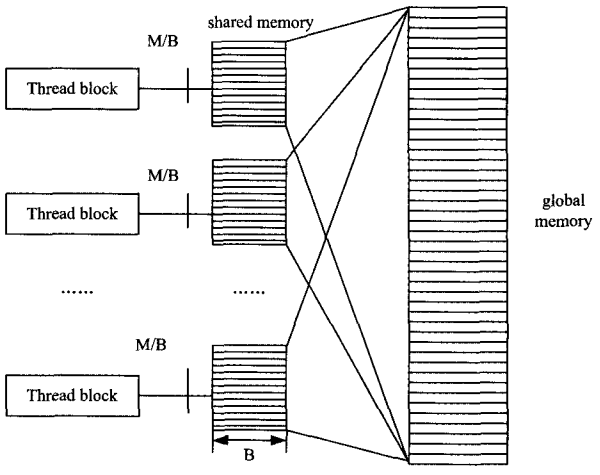

图 1 GPU 存储模型

在 PEM 模型中，不同处理器可以同时访问内部存储器/外部存储器中的不同块；而访问外部存储器中相同块则有 3 种访问方式：

(1)并发读，并发写(Concurrent Read, Concurrent Write, CRCW)：多个处理器可以同时对外部存储器中的同一块进行读和写。

(2)并发读，互斥写(Concurrent Read, Exclusive Write, CREW)：多个处理器只能同时对外部存储器中的同一块进行读，但是不能同时写。

(3)互斥读，互斥写(Exclusive Read, Exclusive Write, EREW)：多个处理器不能同时对外部存储器中的同一块进行读和写。

在 GPU 存储模型中，共享存储器为了能够在并行访问时获得高带宽，被划分成大小相等并且能同时被访问的存储器块(bank)。如果 half-warp 中 16 个线程访问的地址位于不同的 bank 中，由于共享存储器中不同的存储块可以同时工作并互不影响，因此对应于不同 bank 中不同的地址可以同时被访问，此时系统的有效带宽为一个 bank 访问时带宽的倍数。反之，如果 half-warp 中 16 个线程访问的多个地址位于同一 bank，此时就出现了访问冲突。GPU 系统硬件将这些访问请求作为顺序的访问次数被串行地完成，此时系统的有效带宽就会降低几倍，而降低的倍数就是顺序访问请求的次数。为了避免块冲突，一个 half-warp 中不同线程不能同时访问共享存储器中的同一 bank。因此在本文中，GPU 存储模型只考虑 EREW 存储访问模式。

4 GPU 功耗模型

为简化工作，本文将做以下假设：所有 active warps 都以相同的执行频率在 GPU 上运行，如果没有任务进行计算时，则切换到空闲状态。warp 参与任务的计算时间与访存时间都由 warp 的工作频率来度量。

GPU 上程序运行时间 T 与 GPU 执行时钟周期数成正比。假设 X 代表 GPU 的工作频率，A 为时钟周期，则有：

$$T=\frac{A}{X} \tag{1}$$

功耗优化通常通过 DVFS(动态电压/频率调节)技术实现，它通过在一定范围内降低电压以及轻负载处理器频率，使

得在损失部分性能甚至不影响性能的情况下减少能量消耗。电路核心电压 V 和 f 一般要同时调节才能保证电路正常工作，它们之间满足：

$$f \propto \frac{(V-V_t)^\gamma}{V} \tag{2}$$

式中，V_t 是阈值电压，γ 为一个工艺相关的参数。通常情况下 V_t 远小于 V，并且 $\gamma \in [1,2]$。本文假定 $\gamma=2$，此时频率 f 与电压 V 近似为线性关系。根据 CMOS 电路的能量消耗公式：

$$P=\alpha C V^2 f \tag{3}$$

α 是一个工艺相关的系数，功耗 P 可以看成和频率 f 的立方成正比，即：

$$P=kf^3 \tag{4}$$

因此 GPU 能耗可以表示为：

$$E=E_d \times X^3 \times T \tag{5}$$

其中，E_d 为硬件工艺相关的常数。

本文用到的所有参数和常量的相关定义如下：

①E_m：一次访问存储器消耗的能量(包括读和写)。

②F_g：GPU 处理器最大的工作频率。

③N：并行程序输入的数据规模。

④M：并行程序在 GPU 上分配的线程块数。

⑤M_c：GPU 以最大处理器频率访问共享存储器花费的时钟周期数。

⑥P_{cs}：CPU 的空闲功耗。

⑦P_{gs}：GPU 的空闲功耗。

5 GPU 中并行算法的功耗评估方法

接下来本文根据 GPU 的功耗模型提出了并行算法在 GPU 中运行时消耗能量的评估方法。该方法的具体执行步骤如下：

(1)找到并行算法的关键路径。注意这里的关键路径是指通过并行算法的任务依赖图找到的最长路径。关键路径的长度可以通过测量任务在 GPU 上执行的最长时间来确定。

(2)将关键路径分成 GPU 计算次数、存储器访存次数(包括读和写)以及栅栏同步通信次数。

(3)根据 GPU 计算次数、存储器访存次数以及栅栏同步通信次数可以得到所有线程块在时间 T 内完成运算时所需的处理器频率。

(4)根据计算次数，得到该并行算法在 GPU 上计算花费的时钟周期数。计算能耗表示为：

$$E_{comp}=E_d \times b \times (x')^2 \tag{6}$$

式中，b 表示计算时钟周期数，x' 表示步骤 3)中的处理器频率。

(5)根据访存次数，得到该并行算法在 GPU 上访存花费的时钟周期数。访存能耗表示为：

$$E_{mem}=E_m \times c \tag{7}$$

(6)计算系统空闲功耗。考虑两个任务 T_1 和 T_2，它们在单个 CPU 上运行的时间分别为 t_{1c} 和 t_{2c}，在单个 GPU 上运行的时间为 t_{1g} 和 t_{2g}。让 T_1 在 CPU 上运行，T_2 在 GPU 上运行。如果 T_2 在 GPU 上运行的时间小于 T_1 在 CPU 上运行的时间，T_2 将会在 T_1 之前运行结束，此时系统的空闲时间为 $t_{1c}-t_{2g}$；相反，如果 T_1 在 CPU 上运行时间小于 T_2 在 GPU 上运行的时间，T_1 将会在 T_2 之前运行结束，此时系统的空闲时间为 $t_{2g}-t_{1c}$。空闲功耗可以表示为：

$$t_{idle}=\begin{cases} t_{1c}-t_{2g}, & t_{2g}<t_{1c} \\ t_{2g}-t_{1c}, & t_{1c}<t_{2g} \end{cases} \tag{8}$$

$$E_{idle}=\begin{cases} t_{idle} \times P_{gs}, & t_{2g}<t_{1c}, \\ t_{idle} \times P_{cs}, & t_{1c}<t_{2g} \end{cases} \tag{9}$$

(7)系统总能耗可以表示为：

$$E=E_{comp}+E_{mem}+E_{idle} \tag{10}$$

6 并行前缀求和算法

并行前缀求和算法中所有前缀求和操作都需要对实体 I 中的数据进行二元操作符⊕运算。输入的数据是一个数组，输出的数据与输入的数据是具有同样规模的数组阵列，其中每个元素对应输入数据的总和。假设输入的数组为：

$$[a_0, a_1, \cdots, a_{n-1}] \tag{11}$$

则输出的数组为：

$$[a_0, (a_o \oplus a_1), \cdots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})] \tag{12}$$

数组中数据进行前缀求和的操作通常被称为扫描(Scan)，分为 Reduce phase 和 Down-sweep phase 两个过程。

在进行前缀求和扫描时，使用平衡树技术贯穿整个平行计算过程的始终。Reduce phase 实际上是一棵平衡二叉树，这里面非常巧妙的是，这棵平衡二叉树存储空间只有 N，在做完第一个 pass 后，会形成一棵逻辑向上的二叉树。而这棵二叉树只有非右节点有效，即任何一个节点作为父节点的右节点都是无效的。除了这点之外，这颗二叉树还具有另外一个性质，就是任何一个有效内部节点的值都是其所有叶子节点值之和，这个过程之所以被称为 Reduce phase，是因为在这个阶段结束之后，输出矩阵中最后一个节点数据保存了输入数组阵列中所有节点的总和。Reduce phase 的执行过程如图 2 所示。

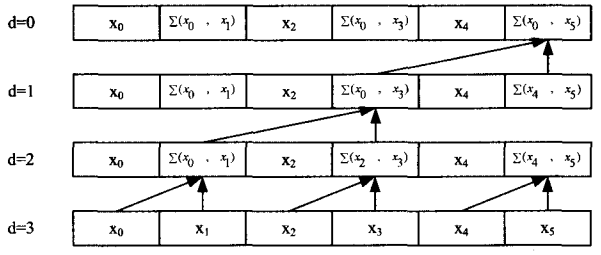

图 2 Reduce phase

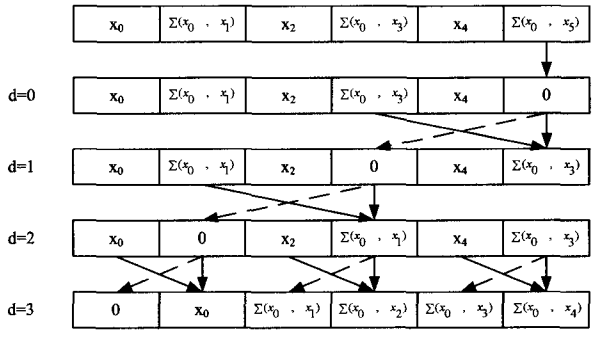

图 3 Down-sweep phase

Down-sweep 与 Reduce phase 的过程相似，也是一棵平衡二叉树。但不同的是，Down-sweep 的操作是把当前节点和左孩子的权值相加，赋予右孩子，然后再把父节点的权值直接给予左孩子。在 Down-sweep 过程中，需要特别指出的是，由于 Reduce phase 结果中最后一个元素不包含整个输入数组数据的总和，因此我们在进行 Down-sweep phase 时，首先将

数组的最后一个元素设置为零，然后由 0 开始向下扫描，在过程结束时，0 元素已经由数组的尾部扫描到了数组的头部。Down-sweep 的过程如图 3 所示。

在 Reduce phase 过程中，当 $d=3$ 时，一个线程中一半的线程从共享存储器读取相邻线程块中的数据，然后进行加和操作，将结果写入自己的线程块中。递归执行相同操作，直到最后只有一个线程进行了所有数据的求和操作。因此在每个递归步骤中，进行了一次读、一次写以及一次加操作，如图 4 所示。

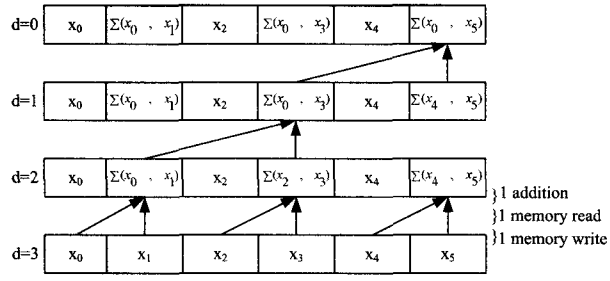

图 4　Reduce phase 读写操作

在 Down-sweep phase 过程中，首先以 0 元素来取代数组的最后一个元素。当 $d=0$ 时，将 0 和当前节点的左孩子的权值 $\Sigma(x_0 \cdots x_3)$ 进行相加，赋予其右孩子 $\Sigma(x_0 \cdots x_3)$，然后再把当前父节点的权值即 0 元素赋予其左孩子。递归执行相同的操作，直到最后只有一个线程进行了所有数据的求和操作，但是不包括最后一个元素。因此在每个递归步骤中，进行了两次读、一次写以及一次加操作，如图 5 所示。

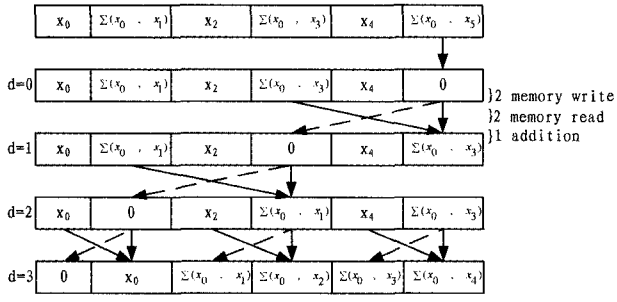

图 5　Down-sweep phase 读写操作

接下来本文根据第 5 节提出的功耗评估方法对并行前缀求和算法进行功耗评估。在上述算法中，关键路径很容易就可以找到：它的执行路径就是从 Reduce phase 到 Down-sweep phase，线程块完成所有的计算任务（第 1 步）。在计算过程中，所有数据执行栅栏同步，以确保整个数据都已经从全局存储器加载到共享存储器中。同样，栅栏同步也要确保所有数据写入前已经在计算过程中进行了更新。因此该算法的计算次数、内存访问次数以及栅栏同步次数如表 1 所列（第（2）步）。

表 1　并行前缀求和算法计算次数、内存访问次数以及栅栏同步次数

Synchronization	Memory reads	Computation steps
$2\log M+1$	$2\times\frac{N}{BM}+2\log M+1$	$2\times\frac{N}{M}+2\log M+1$

假设 α 为一次加法运算所需的时钟周期数，从表 1 得出 $Computation\ steps\times\alpha$ 表示整个计算所花费的时钟周期数；$(Memory\ access\ steps+Synchronization\ steps)\times M_c$ 表示内存访存和栅栏同步所花费的时钟周期数。因此实际计算周期所占比率为：

$$\frac{Computation\ steps\cdot\alpha}{T\cdot F_g-(Memory\ access\ steps+Synchronization\ steps)\cdot M_c} \tag{13}$$

当所有线程块都有任务执行时，其运行时间设为 T，则此时的计算频率（Reduced frequency）为（第（3）步）：

$$x'=F_g\cdot\frac{(\frac{2N}{M}+2\log M+1)\cdot\alpha}{T\cdot F_g-(\frac{2N}{BM}+4\log M+2)\cdot M_c} \tag{14}$$

并行前缀求和算法在 GPU 上运行时计算所花费的时钟周期为：

$$b=(\frac{2N}{M}+2\log M+1)\cdot\alpha \tag{15}$$

则计算能耗为（第（4）步）：

$$E_{comp}=E_d\cdot b\cdot(x')^2 \tag{16}$$

并行前缀求和算法在 GPU 上运行时存储器访存次数为：

$$c=\frac{2N}{BM}+2\log M+1 \tag{17}$$

则访存能耗为（第（5）步）

$$E_{mem}=E_m\cdot c \tag{18}$$

在实际运行中，一个优良的系统应保证 CPU 在等待 GPU 的计算结果时，CPU 可以运行其他任务。然而在并行前缀求和算法中，系统只运行一个任务，CPU 则作为 GPU 的协处理器，所做的工作仅仅是在运算开始前将数据传输到 GPU，以及在运算结束后等待从 GPU 传回的运算结果并输出该结果。因此在并行前缀求和算法中，当 GPU 进行运算时，CPU 是处于空闲状态的，所以系统空闲功耗为（第（6）步）：

$$E_{idle}=t_{GPU}\times P_{cs} \tag{19}$$

最后，系统功耗等于计算功耗、存储功耗以及空闲功耗的总和（第（7）步）。

7　性能测评与分析

扫描元（Scan primitives）是解决数据元并行的典型方法。稀疏矩阵矢量乘（SpMV）是应用扫描元求解该算法的典型示例。在实验部分，本文通过 SpMV 的实际应用来对 GPU 功耗模型进行分析验证。

本文的性能测评与分析阶段分为两个步骤进行，分别为：

（1）证明功耗模型的有效性

将实际测得功耗与本文功耗模型评估的功耗进行误差对比，来验证功耗模型的有效性。

（2）分析功耗模型的有用性

根据功耗评估方法对 SpMV 进行功耗分析。通过改变敏感参数分别对计算功耗、存储功耗以及空闲功耗进行分析，在满足性能要求的条件下，如何选择最优线程块数，使得系统的功耗降到最低。

7.1　实验平台

（1）硬件平台

硬件平台包括实验计算机（Experiment Computer）、分析计算机（Profiling Computer）和功率计（Power Meter）。实验计算机配备的是英特尔双核 CPU 和 2 个 GPU 的显卡。其中 NVIDIA Geforce 9600 用作显示，NVIDIA Tesla C1060 用作

计算。NVIDIA Tesla C1060 的 CUDA 核心频率为 1.15 GHz，存储频率为 1.5GHz。硬件平台架构如图 6 所示，功率计是电路中专门计量用电设备实际工作所损耗的电能计量装置，而分析计算机则用来定期从功率计中读取数据。

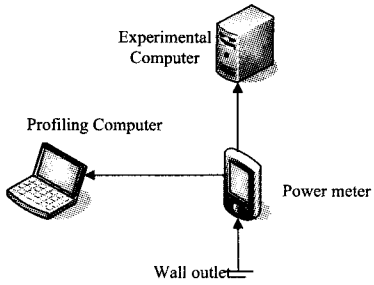

图 6　硬件平台

(2)软件平台

实验计算机配备的是 64 位 Windows Xp 操作系统，开发环境为 Microsoft Visual Studio 2005，CUDA 的版本为 2.1。软件平台的具体配置如图 7 所示。

GPU	NVIDIA GeForce 9600 GTX(Used for display) NVIDIA Tesla C1060(Used for compute)
CPU	Intel®Core™920
OS	Windows xp 64 bit
CUDA	CUDA 2.1 Beta
IDE	Microsoft visual studio 2005
DRAM	6GB

图 7　软件平台

由于稀疏矩阵非零元素个数稀少，佛罗里达 Tim Davis 教授在他个人网站上收集了各种各样来自于不同应用领域的稀疏矩阵[6]，本文选取了不同数据规模的有代表性的 3 个稀疏矩阵，如表 2 所列。

表 2　选取的 3 个稀疏矩阵

Matrix	Rows	Columns	Nozero
m_1	3200	3200	1880
m_2	93280	93280	652247
m_3	659033	659033	5959282

7.2　功耗模型的有效性

以下的评价指标用来验证功耗模型的有效性。

Time 表示 Kernel 的执行时间。为了避免特殊性，应将相同的程序执行多次，取其平均时间。

Energy 表示 Kernel 在 GPU 上执行时整个系统所消耗的能量。

Power 表示平均消耗的能量，表达式为：

$$Power=\frac{Energy}{Time} \tag{20}$$

首先，通过测试稀疏矩阵在实验计算机上执行的时间，可以得到 SpMV kernel 的执行时间，其实验结果如表 3 所列。从分析计算机上读取功率计中的数据，根据式(20)，计算得出 SpMV 在系统执行过程中平均消耗的能量。

表 3　SpMV kernel 的执行时间

Matrix	GPU-Kernel time (s)	GPU-HostToDevice time (s)	GPU-total time (s)
m_1	0.026145	0.052019	0.078764
m_2	0.140452	0.35549	0.495943
m_3	0.723514	1.376797	2.120311

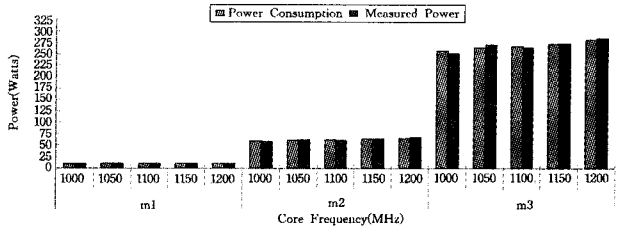

图 8　预测功耗与实际功耗

为了验证功耗模型的有效性，首先在稀疏矩阵数据输入中改变 GPU 的核心频率，然后比较通过功耗模型计算出的预测功耗和实际功耗。如图 8 所示，每根测试的蓝色柱子表示在不同核心频率下分别对 100 组数据进行测试，选取其平均值。可以发现在不同的工作负载中，实际平均测量的功耗和预测的功耗之间的误差为 2%左右，这是允许的误差范围。因此该功耗模型的正确性得到了有效证明。

从图 8 中还可以发现，稀疏矩阵的规模越大，其标准差就越小。这主要是因为较大规模的稀疏矩阵在运行过程中将会获得较小的实际工作频率(Reduce frequency)，因此可以承载更大的工作负载，相对而言，其功耗的变化就会越小。

7.3　功耗模型的有用性

在根据功耗模型得到的并行算法在 GPU 上执行时的功耗评估方法中，需要指出的是功耗表达式依赖于许多变量，这其中包括 N(输入数据的规模)、M(并行程序在 GPU 上运行时所分配的线程块数)、α(一次加法运算所需的时钟周期数)、M_c(共享存储器最大频率执行时访存所需时钟周期数)、E_m(一次访存所消耗的能量，包括读和写)、F_g(GPU 的最大核心计算频率)。

在实验中，本文假设 $\alpha=2$，这是由于在大多数的体系架构中，参与一次加法运算的时钟周期只有两个，一个是将数据加载到共享内存耗费的时钟周期，另外一个是做加法运算时所耗费的时钟周期。

(1)E_{comp} 和 E_{mem} 敏感性分析

在进行 SpMV 运算前，首先要为 SpMV 分配线程块，不同的线程块所需的执行时钟周期不同，这将会影响 GPU 的执行效率。图 9 显示了 SpMV 中线程块数量与功耗的关系。从图中可以看出，开始功耗随着线程块数量的增加而减少，随后又随着线程块数量的增加而增加，其主要原因在于开始功耗的减少是由于实际执行频率(Reduced frequency)的减少，随后的增加则是由于访存能耗的急骤增加。

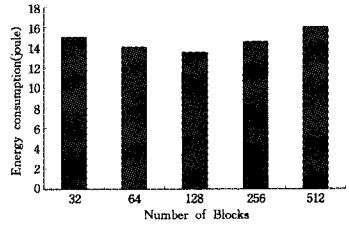

图 9　SpMV 分配的线程块数量与功耗的关系

前面的论述无非就是要证明线程块的数量是影响 GPU 功耗的关键因素，接下来本文将要考虑如何给并行程序分配最优线程块数，使得功耗达到最低。

根据式(14)—式(19)，可以知道最优线程块数(M)是一个理论值，并且系统的计算能耗 E_{comp} 和存储能耗 E_{mem} 都是关

于 M 的表达式：

$$E(M)=E_{comp}(M)+E_{mem}(M) \tag{21}$$

为了能够得到 $E_{comp}(M)+E_{mem}(M)$ 的最小值，可以对 M 进行求导，则有：

$$E(M)'=E_{comp}(M)'+E_{mem}(M)' \tag{22}$$

根据式(22)可以求得当系统功耗最低时，并行程序所需分配最优线程块数(M)。接下来本文需要对一些参数进行简化，并且不失一般性，对 E_{comp} 和 E_{mem} 进行敏感性分析。

E_m 表示一次访存所消耗的能量，而 E_c 则表示系统处于最大核心频率时执行一条指令所消耗的能量，根据式(6)则有：

$$E_c=E_d\times X^2 \tag{23}$$

假设参数 K 表示 E_m 和 E_c 之间的比率，E_m 和 E_c 之间的关系可以表示为：

$$E_m=k\times E_d\times X^2 \tag{24}$$

E_{idle} 表示系统处于最大核心频率时一个时钟周期所消耗的空闲能耗，假设 E_c 与 E_{idle} 之间的比率为 10，则有 $E_c=10E_{idle}$。

本文通过改变 K 和 M_c 的值对 $E_{comp}+E_{mem}$ 进行敏感性分析。图 10 显示了在稀疏矩阵规模 N 和 M_c 值不变的情况下($N=5959298$，$M_c=500$)，系统最低能耗所需的最优线程块数。从图中可以明显看出，最低能耗所需的最优线程块数随着 M_c 值的增加而减少。

接着图 11 显示了在稀疏矩阵规模 N 和 K 值不变的情况下($N=5959298$，$K=500$)，系统最低能耗所需的最优线程块数。从图中可以发现，最低能耗所需最优线程块数开始随着 M_c 值的增加而减少，随后随着 M_c 值的增加而增加。

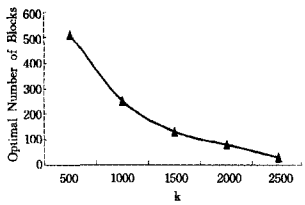

图 10 敏感性分析($N=5959298$，$M_c=500$)

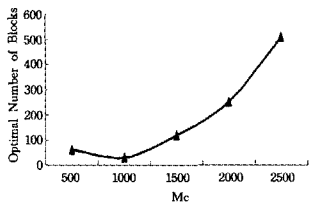

图 11 敏感性分析($N=5959298$，$K=500$)

(2) E_{idle} 敏感性分析

由于 $E_{idle}=t_{GPU}\times P_{cs}$，$P_{cs}$ 是一个常数，t_{GPU} 的值越小，则 E_{idle} 的值就越小。

当 GPU 进行 SpMV 任务计算时，CPU 是处于空闲状态的，因此此时 t_{GPU} 的值应该等于稀疏矩阵数据在 CPU 与 GPU 之间的传输时间与 SpMV Kernel 执行时间的总和。

在已发表的文献[7]中，我们设计了一种基于改进的行压缩表示下的稀疏矩阵向量乘并行算法，该算法从 CPU 移植到 GPU，并且根据 CUDA 程序优化原则，对该并行算法进行了优化，其具体的技术路线包括：动态分配线程、合并内存访问、行补齐以及使用 Write-Combined Memroy。

优化后的实验结果如表 4 所列，该算法与已有的 NVIDIA CUDPP library 和 NVIDIA SpMV library 中的 CSR-vector 算法比较，具有更高的存储器带宽和浮点运算吞吐量。

根据式(19)，依据系统空闲时间可以相应地计算出系统的空闲功耗，如图 12 所示。从图中可以看出，系统空闲功耗与空闲时间是成线性关系的。

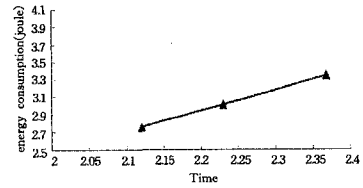

图 12 SpMV 的系统空闲功耗与空闲时间关系

表 4 SpMV 优化后的实验结果

Execution time	Optimization	SpMV library	CUDPP library
Kernel time(s)	0.723514	0.814531	0.789324
Transfer time(s)	1.376797	1.419073	1.578661
t_{GPU}(s)	2.120311	2.233604	2.367985

影响空闲功耗时间的因素有很多，其中包括：存储器访问方式、任务执行顺序、线程映射方式等等。根据图 12 中空闲能耗与空闲时间呈现的线性关系，我们可以知道，越小的空闲时间，就会有越小的空闲能耗。

结束语 本文首先根据 PEM 模型提出 GPU 存储模型，然后在该存储模型的基础上建立了 GPU 功耗模型，并提出了并行算法在 GPU 上运行时的功耗评估方法。接着通过并行前缀求和算法对该方法进行详细的论述和分析。实验部分通过稀疏矩阵向量乘算法的实际应用来对该模型的正确性进行证明，并通过改变影响功耗的参数对该模型的有用性进行分析。本研究能够帮助程序员对并行算法能耗进行预估分析，并且对 GPU 存储器选择、任务开销以及最优线程块数的确定提供了指导作用。

参 考 文 献

[1] Collange S, Defour D, Tisserand A. Power consumption of GPUs from a software perspective[C]// Allen G, Nabrzyki J, Seidel E, et al., eds. Proceedings of the 9the International Conference on Computational Science, Lecture Notes in Computer Science 5544. Berlin, Heidelberg: Springer Verlag, 2009: 914-923

[2] Rofouei M, Stathopoulos T, Ryffel S, et al. Energy-aware high performance computing with graphic processing units[C]// Proceedings of the 2008 conference on Power aware computing and systems. USENIX Association: San Diego, California, 2008: 11-11

[3] Ma X, Dong M, Zhong L, et al. Statistical Power Consumption Analysis and Modeling for GPU-based Computing[C]// Workshop on Power Aware Computing and Systems (HotPower'09). 2009

[4] Takizawa H, Sato K, Kobayashi H. SPRAT: Runtime processor selection for energy-aware computing[C]// Cluster Computing, 2008 IEEE International Conference. 2008

[5] 林一松，杨学军，唐涛，等. 一种基于并行度分析模型的 GPU 功耗优化技术[J]. 计算机学报，2011，34(4)：705-716

[6] Davis T[OL]. http://www.cise.urf.edu/research/sparse/matrices/

[7] Wang Zhuo-wei, Xu Xian-bin, Zhao Wu-qing, et al. Optimizing sparse matrix-vector multiplication on CUDA[C]// The 2nd International Conference on Education Technology and Computer. 2010(4): 109-113