一种基于GPU的高精度体系结构级功耗模型

王卓薇 程良伦 肖 红
(广东工业大学计算机学院 广州 510006)

摘 要 随着硬件功能的不断丰富和软件开发环境的逐渐成熟,GPU开始被应用于通用计算领域,协助CPU加速程序运行。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得其性能远高于CPU的同时功耗也高于CPU,功耗问题已经成为制约GPU发展的重要问题之一。在深入研究Fermi GPU架构的基础上,提出一种高精度的体系结构级功耗模型,该模型首先计算不同native指令及每次访问存储器消耗的功耗;然后根据应用在硬件上的执行指令和采样工具获得采样结果,分析预测其功耗;最后通过13个基准测试应用对实际测试与功耗模型测试结果进行对比分析,该模型的预测精度可达90%左右。

关键词 GPU,Fermi,功耗模型,native指令,存储器功耗
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.11.006

High-precision Architecture-level Power Model Based on GPU

WANG Zhuo-wei CHENG Liang-lun XIAO Hong
(School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China)

Abstract As hardware functions are constantly developing, and software development environments gradually mature, the graphics processing unit (GPU) has been applied to general purpose computation to help the central processing unit (CPU) accelerate a program. To obtain high performance, a GPU generally contain hundreds of core arithmetic units. Owing to the existence of high-density computing resources, the performance of the GPU is much superior to that of the CPU, while its power consumption is larger than that of the CPU. Power consumption has become one of the important issues restricting the development of GPU. Based on the study of the Fermi GPU architecture, a high-precision architecture-level power model was proposed in this research. In this model, the power consumed by different native instructions, and each memory access, were first calculated, then the power consumption was analysed and predicted according to the execution instructions as applied to the hardware, and the sampling results were acquired using sampling instruments. Finally, the results obtained from practical testing and the power model were compared by using 13 benchmark applications. It is demonstrated that the prediction accuracy of the model can reach approximately 90%.

Keywords GPU, Fermi, Power model, Native instruction, Memory power consumption

1 引言

21世纪人类所面临的重要科技问题(如卫星成像数据的处理、基因工程、全球气候准确预报、核爆炸模拟等)数据规模已经达到TB甚至PB量级,没有万亿次以上的计算能力是无法解决的。与此同时,在日常应用(如游戏、高清视频播放)中面临的图形和数据计算也越来越复杂,对计算速度提出了严峻挑战。

图形处理器(Graphics Processing Units,GPU)在处理能力和存储器带宽上相较CPU有明显优势。同时,随着GPU编程环境的不断完善,使用GPU作为CPU的加速器构造异构并行计算系统已经成为高性能计算领域的热点方向[1]。例如由我国自主研发的超级计算机天河-1A系统就采用CPU-GPU异构系统结构,峰值计算性能超过4PFlops,位列2010年12月发布的Top500超级计算机排行榜之首。

由于采用了专用的加速部件,异构系统的性能/功耗比一般高于传统的同构系统,但是其功耗问题依然十分严峻。GPU芯片的计算单元密度高,发热量较大,其绝对功耗超出CPU。在构建大规模计算系统时,GPU功耗将提高计算成本,同时也会降低系统的可靠性,因此研究GPU低功耗优化问题具有重大意义。

低功耗优化首先需要对功耗进行精确建模。物理层功耗模型过于复杂,不便进行功耗优化,因此大部分功耗建模的工作都集中在体系结构级[2-4]。目前常见的GPU体系结构级功耗模型大多基于传统CPU功耗模型修改得到,然而GPU中许多体系结构特征是传统CPU中没有的,如成百上千个处理单元组成的计算阵列等。

针对以上问题,本文在深入研究GPU架构的基础上提

到稿日期:2015-11-04 返修日期:2016-01-22 本文受国家自然科学基金(61300029,61672168)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,E-mail:llcheng@gdut.cn;肖 红(1972-),女,副教授,主要研究方向为计算机网络,E-mail:wh_red@163.com。

出一种高精度的体系结构级功耗模型。该模型针对 Fermi GPU，根据实时功耗开销的来源分为计算单元和存储器两部分。再根据不同类型计算指令的情况以及访问存储器种类、次数，获得执行不同指令的能量消耗以及一次访问不同存储器的功耗开销。本文选取了 13 个基准测试应用进行功耗分析预测，实验结果表明本文提出的体系结构级 GPU 功耗模型对运行在 Fermi GPU 架构上的测试代码的能量开销预测精度达到了 90%左右。

本文第 2 节对功耗模型的相关研究现状及进展进行详细论述；第 3 节详细介绍体系结构级功耗模型的理论架构、实验装置搭建的过程以及实验环境干扰的消除，同时对功耗模型所需的全部参数进行提取，完成功耗模型的实现；第 4 节通过基准测试应用对功耗模型进行详细的预测与分析，验证该功耗模型的精度；最后总结全文。

2 相关工作

2.1 功耗模型分类

目前功耗建模的主要研究方式有 3 类：指令级功耗模型、基于采样的功耗模型以及体系结构级功耗模型。

(1)指令级功耗模型

指令级功耗模型是将一条指令的执行看成一次基本系统活动，在此基础上统计系统功耗。Tiwari 等人[5]提出一种指令能量模型，通过一个循环的反复执行来获得每条指令的能量代价，每条给定指令的基本能量代价差异主要来自不同的指令操作码，最终整个程序的基本能量代价为所有执行指令的能量代价总和。

(2)基于采样的功耗模型

基于采样的功耗模型[6,7]通常与系统的物理测试结合在一起，周期性地中断处理器的执行，记录执行过程记忆瞬时功耗，使用记录下的数据对各种处理情况下的能量使用状况创建能量 profile。

(3)体系结构级功耗模型

体系结构级功耗建模的基本思路是统计所有系统活动的功耗及其活动次数，累加得到系统的总功耗。研究工作的差异主要集中在如何抽象不同的系统活动、如何统计这些活动，以及如何获取每种活动的功耗上。目前最常见的是基于模拟的体系结构级功耗模型，其依靠对目标系统的模拟执行统计其部件执行情况，并累计功耗，其统计功耗的基本单位是功能块，如加法器、乘法器、控制器、寄存器文件、cache 等[8]。

2.2 典型体系结构级功耗模型

(1)统计型体系结构级功耗模型

普林斯顿大学的研究人员将功耗(Totalpower)分为两大部分：执行功耗(Power)和空载功耗(IdlePower)，如式(1)所示[9]。对于执行功耗，针对 CPU 芯片的 22 个主要组成部件分别进行测量和研究，将每个参数的功耗分为非门控时钟电路部件功耗与与门控时钟电路部件功耗两个部分。非门控时钟电路部件功耗大小为常量，与门控时钟电路部件功耗由 AccessRate，ArchitecturalScaling 和 MaxPower 参数决定，如式(2)所示。该功耗模型是针对 CPU 架构的统计性体系结构级功耗模型，对于 GPU 功耗模型的研究具有重要借鉴意义。

$$Totalpower=\sum_{i=1}^{n}Power(C_i)+IdlePower \tag{1}$$

$$Power(C_i)=AccessRate_i\times ArchitecturalScaling(C_i)\times MaxPower(C_i)+NonGateClockPower(C_i) \tag{2}$$

(2)基于图像显示的功耗模型

由休斯顿大学和莱斯大学研究人员建立的体系结构级功耗模型不是一个 GPGPU 应用的功耗模型，而是一个基于图像显示的功耗预测模型[10]。该模型的最大特点是基于 NVIDIA 公司的 PerfKit 工具，通过其中的渲染测试，获得参数记录结果的变化和对应时刻的功耗变化来建立功耗模型。

Perfkit[11]工具是 NVIDIA 公司为了分析和调试 OpenGL 和 Direct3D 应用开发的一套可以全面测试其性能的工具，其提供了 39 个动态变化的变量。而这个功耗模型只利用了其中 5 个变量信号：vertex shader busy(顶点着色器的执行时间占总执行时间的百分比)、pixel shader busy(像素着色器的执行时间占总执行时间的百分比)、texture busy(纹理单元的执行时间占总执行时间的百分比)、goem busy(几何着色器的执行时间占总执行时间的百分比)和 ROP busy(光栅操作活跃时间占总执行时间的百分比)。通过不同图像的测试分析，记录执行时上述 5 个参数的变化情况，同时测量其功耗变化情况。根据大量的不同图像的测试结果，得到功耗变化与这 5 个参数变化的内在联系，进而得出针对图像变化趋势的功耗模型。

基于图像显示的功耗模型反映出图像处理过程中 GPU 架构的功耗与这些参数之间的内在联系。但这个功耗模型的精度较低，同时其架构还是没有摆脱图形计算的限制，没有真实反映通用计算中 GPU 架构的功耗情况。

(3)基于 CUDA profiler 的功耗模型

日本东京大学研究人员利用 CUDA profiler version 2.3 采样一系列应用内核，获得一系列技术结果，测量应用的功耗，建立线性方程组，求解内核功耗和这些计数器结果的对应关系，获得功耗模型，如式(3)所示[12]。

$$p=\sum_{i=1}^{n}a_ic_i+\beta \tag{3}$$

其中，p 为应用的功耗值，α_i 为参数计数器数目，c_i 为计数器类型，β 为固定功耗值。该功耗模型将 CUDA profiler 与功耗模型联系起来，但是其研究结构过于简单，仅通过这些参数并不能准确反映计算单元的执行情况，当应用属于计算密集型，则其精度会大大降低。

乔治亚理工大学研究人员建立的功耗模型相较于日本东京大学做出了很大改进[3]。该功耗模型将架构单元分为：整型计算单元、浮点型计算单元、算术逻辑单元、全局存储器参数、局部存储器参数、纹理存储器参数、常数存储器参数、共享存储器参数以及寄存器参数。其总体功耗如式(4)所示，其中 Runtime_Power 由流处理器单元功耗(RP_SM)和存储器功耗(RP_Memory)两个部分组成，如式(5)所示。其中流处理器单元功耗为上述多种计算单元使用率与其对应的功耗参数的乘积，由于 CUDA profiler 记录的是一个流多处理器组中的统计结果，因此要乘以使用的流多处理器的数目，如式(6)所示。存储器功耗则由全局存储器功耗和局部存储器功耗之和组成，如式(7)所示。

$$GPU\_power=Runtime\_power+IdlePower \tag{4}$$

$$Runtime\_power=\sum_{i=1}^{n}RP\_Component(c_i)=RP\_SM_i+RP\_Memory \tag{5}$$

$$RP\_SM_s=Num\_SM_s\times\sum_{i=1}^{n}SM\_Component_i \tag{6}$$

$$RP\_Memory=RP\_GlobalMemory+RP\_LocalMemory \tag{7}$$

该功耗模型无论在测量的理论方法还是测量参数的提取方法上相较于之前的功耗模型都有较大的提高。但是该功耗模型仅将计算单元功耗归类于各种计算单元的使用率。实际上不同指令执行时使用的资源、执行周期、并发性能都不相同，所以功耗也不相同。同时该功耗模型对于存储结构复杂的功耗模型而言过于简单(Fermi 架构中存储器结构复杂)，因此需要一种更为精确的体系结构级功耗模型。

3 体系结构级 GPU 功耗模型

3.1 理论功耗建模

体系结构级 GPU 功耗模型的核心思想是分析一个在 GPU 架构上执行应用的指令和访存情况，建立一个系统级功耗分析模型，准确预测该应用消耗的能量或实施能量变化。

体系结构级 GPU 理论功耗模型建模的过程分为两个步骤：

(1)GPU 总功耗开销由静态功耗与动态功耗两个部分组成(见式(8))。静态功耗主要包括在应用内核执行时，时钟电路、片上总线、多线程动态调度控制等部分所消耗的能量(见式(9))。经过分析发现动态功耗主要由计算单元和存储器所消耗的能量构成。

(2)分别对计算单元和访存的功耗开销进行分析。应用在计算单元的能量消耗不仅与所使用的计算单元的种类有关，同时由于不同指令的执行周期和并发性能均不同，所产生的功耗也应有较大区别，因此需要统计执行内核中的不同指令数目，再测试出不同指令的能量开销，将其进行叠加。存储器的访存功耗则需要借助于 CUDA profiler，采样内核中 L1/L2/Dram 的访存次数，通过单次访存能量开销与访存次数的乘积来进行统计。因此动态功耗可由式(10)表示。

$$E_{total}=E_{idle}+E_{runtime} \tag{8}$$

$$E_{idle}=P_{idle}\times Time_{exe} \tag{9}$$

$$E_{runtime}=\sum_{i=1}^{n} memory_i\_enery\times count\_number_i+\sum_{i=1}^{n} Instruction_i\_enery\times instruction\_number_i \tag{10}$$

3.2 计算单元指令功耗参数提取

指令数目的记录是通过 native 指令在硬件上的执行结果来确定的。由于不同 native 指令在 GPU 上的执行周期和并发性能都不一样，因此对于不同类型的指令，需要分别进行测试。

Fermi 架构中一个流处理器组可以并发执行 32 条指令，C2050 中包含的流处理器组的数量为 14 个，因此 C2050 同时并发执行的计算单元指令数目最高可以达到 448 条。在 Fermi 架构中，由于每个流处理器组分别包含一个流处理器组控制器、L1 高级缓存和共享存储，流处理器组之间是完全独立的，因此理论上每条在整形指令处理单元和浮点型计算单元上执行的指令消耗的总能量与其使用流处理器组的数量成正比。

在 GPU 中，反映指令使用率的参数为每一时钟周期内所执行的指令(Instructions Per Cycle，IPC)。以 FADD 指令为例，IPC 最大理论值为 2，当线程块小到一定程度后(小于 448)，有些流处理器组没有工作，则可以获得 IPC 的不同值。改变线程块的数目，测试代码在不同线程的执行情况下的功耗变化，可以证明指令消耗的能量随着 IPC 的增加而线性增长，如图 1 所示。主要的 ALU/SFU 指令消耗的能量如表 1 所列。

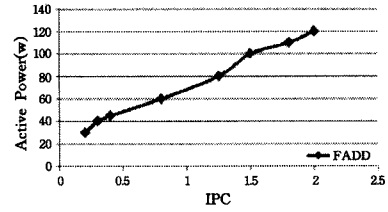

图 1 FADD 指令功耗与 IPC 关系

表 1 常用 native 指令功耗

Native instruction	Energy (nJ/warp)	Native instruction	Energy (nJ/warp)
FADD(2 REG)	5.09	IADD(2 REG)	2.24
FADD(1 REG)	3.39	IADD(1 REG)	2.12
FADD32I	3.58	ISCADD	1.74
FMUL32I	4.65	SEL	3.11
FMUL(2 REG)	5.54	IMUL(2 REG)	5.12
FMUL(1 REG)	4.86	IMUL(1 REG)	4.87
FFMA(3 REG)	5.91	IMAD(3 REG)	5.99
FFMA(2 REG)	5.47	IMAD(2 REG)	5.24
FMNMX	2.98	IMNMX	1.66
FSETP	4.02	ISETP	1.14
DADD	10.63	LOP. AND(OR,XOR)	2.01
DMUL	12.25	SHL	3.66
DFMA	13.38	BAR	4.16
DMNMX	7.75	MOV	1.52
DSETP	5.93	MOV32I	1.17

3.3 存储单元功耗参数提取

GPU 架构中的存储器主要包括寄存器、L1 Cache、L2 Cache、显存、共享存储器、常数存储器以及纹理存储器。由于寄存器位置与计算单元联系紧密，每次计算单元处理数据的过程中都会调用寄存器，因此没有必要将两者再进行区分。同时在测试中也发现，在不改变计算单元上执行指令的情况下，仅仅改变指令执行所需寄存器的数目并不会引起功耗的变化。常数存储器与寄存器的情况类似，访问周期较短，同样也不会引起功耗的显著变化。因此 GPU 存储部分消耗的能量主要来源只包括 L1 Cache、L2 Cache、显存、共享存储器和纹理存储器。

通过 CUDA profiler 可以获得各种存储器的读写次数，L1 Cache、L2 Cache 的命中次数，以及各种存储器执行过程中的带宽参数。以 L1 Cache 为例说明存储单元功耗参数的提取过程。由于 L1 Cache 位于流处理器组中，因此其功耗也是随着使用流处理器组数目的增加而增大。为了统一不同存储器的研究方法，同时在 GPU 架构中存储器的使用率在带宽上表现得最为明显，因此观察 14 个流处理器组在满载的情况下，L1 Cache 功耗和 L1 Cache 带宽的变化关系，如图 2 所示。在除去固定开销后，每次调用 L1 Cache 的能量开销与带宽的函数关系如式(11)所示(a,b 均为常量)。

$$L_1\_access\_energy=a*bandwidth^b \tag{11}$$

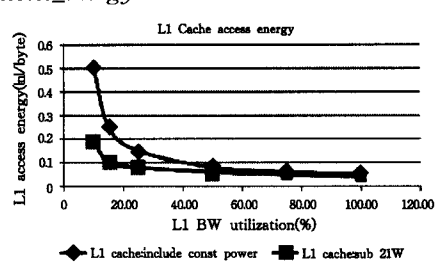

图 2 L1 访问能量开销与带宽关系

L2 Cache、共享存储器以及纹理存储器的能量开销和带宽的函数关系与 L1 Cache 类似，只是 a,b 的值不同。不同存储器的计算参数 a,b,以及存储器使用带宽接近满载时一次访问存储器的能耗如表 2 所列。

表 2　存储器能量开销

memory	Parameter a	Parameter b	Energy (nJ/byte)	Memory utilization
L1 Cache	0.0307	−0.578	0.021	99.7%
L2 Cache	0.145	−0.365	0.13	93.6%
Read shared memory	0.0292	−0.596	0.021	99.8%
Store shared memory	0.0297	−0.542	0.021	99.8%
Read Dram memory	–	–	0.54	90.3%
Write Dram memory	–	–	0.53	85.5%
Texture memory	0.145	−0.299	0.14	99.8%

4　实验结果与分析

4.1　实验平台

(1)实验装置

实验装置包括实验计算机(显卡为 C2050)，测试仪器(Extech Instruments 公司的电力分析监控记录仪 3808031)[13]和记录装置，如图 3 所示。

图 3　实验装置

实验计算机配备的是 Intel Core I7 CPU 和 2 个 GPU 显卡，其中 NVIDIA Tesla C2050 用作通用计算，NVIDIA Geforce 8800GT 用作显示，内存为 6GB，硬盘为 1TB。

电力分析监控记录仪的主要功能是在线监控和记录被测计算机的交流电源和负载设备的功耗情况。而记录装置则提供了记录结果的导出方式，通过 RS232 接口，将被测数据传输和保存至计算机并创建文档，提供强大的数据记录分析功能。

(2)实验环境干扰消除

为了能够准确测量 GPU 的实验数据，仅用上述的测量仪器和装置直接进行测量是存在误差的。这主要是因为在测试过程中存在众多干扰项，需要将这些干扰清除或降低干扰的影响，以提高测量的精度。

消除 CPU 频率变化与风扇转速的干扰：CPU 提供变频功能，这对测量实验计算机的空载功耗和满载功耗的值具有很大影响。同时，随着 CPU 使用频率的变化，CPU 的风扇转速也会变化。为了消除 CPU 频率与风扇转速变化的干扰，可以通过 BIOS 中的设置来固定 CPU 频率与风扇的转速。

消除 GPU 风扇转速的干扰：Tesla C2050 显卡本身也内置一个风扇，风扇转速会随着频率的变化及芯片温度的变化而迅速变化，这对功耗测量也会产生很大影响。为了消除 GPU 风扇转速的干扰，可以利用工具 NVIDIA Inspector 的设置来固定 GPU 风扇转速。

消除芯片温度变化的干扰：通过测试发现，GPU 芯片本身温度的增加会对功耗产生影响[14]。选择在 NVIDIA Tesla C2050 平台上对矩阵乘法运算(矩阵规模为 1024×1024)进行测试，芯片温度对功耗的影响如图 4 所示，从图中可以看到当程序开始运行后，能耗也随之增加，最低能耗与最高能耗之间的差值为 14watts。而该差值就是由于静态功耗的增加或风扇电源造成的能耗增加。将风扇转速由低调到最高，可以测试额外风扇电源能耗只有 4watts，因此余下 10watts 就是由于温度升高而导致的能耗增加。为了消除 GPU 芯片温度对功耗的影响，应减去温度引起的功耗变化的干扰。

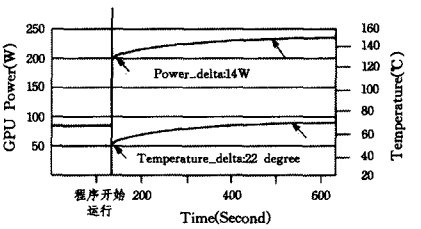

图 4　GPU 芯片温度对功耗的影响

4.2　结果与分析

针对 GPU 架构的基准测试代码种类众多，其中具有代表性的有 NVIDIA 开发的 GPU Computing SDK[15]、伊利诺斯微软体系结构项目团队开发的 parboil[16] 以及弗吉尼亚大学计算机科学学院研究小组开发的 Rodinia[17]。

(1)基准测试应用的选取

基准测试应用的选取应该能够全面真实地反映 GPU 架构的使用情况，并且同时能够充分测量出该功耗模型的准确性。因此基准测试应用的选取应该遵循以下原则：

1)测试代码的执行时间要尽量长，这样才能利用测量工具准确记录其功耗；

2)测试应用应该包含多种类型：计算密集型(混合使用多种计算单元)、访存密集型(访问多种存储器并包含 L1、L2 Cache 命中和显存访问)，专注于某种处理器或存储单元在应用中的使用。这样可以验证功耗模型在多种不同情况下的预测效果。

3)测试应用代码不应当全部是针对 Fermi 架构进行优化的。这是因为针对 Fermi 优化过的代码可以充分利用 Fermi 架构中的单元，并不能充分验证功耗模型是否能够真实反映硬件执行的功耗。

4)选取的测试应用应具有实际意义，能够解决现实工作中的一些问题，这样才能证明本课题研究的现实意义。

根据以上原则，本文选取了 12 个应用来进行测试：MonteCarlo(蒙特卡洛模型)[18]、SpMV(稀疏矩阵与向量乘法)[19-23]、Matrixmul(矩阵乘法)、Reduction(并行规约)[24]、Sgemm(单精度普通矩阵乘法)[25]、Black-scholer(布莱克-斯科尔斯期权定价模型)[26]、Transpose(矩阵转置)[27]、STREAM(数据流处理)、Histogram(直方图统计)[28]、MersenneTwister[29](马特赛特旋转演算法)、quasirandomGenerator[30](准随机数发生器)、sortingNetworks[31](排序网络)。

(2)结果分析

GPU Ocelot[32]是一个模块化为异构系统设计的动态编译框架，提供各种各样的后台目标程序和分析模块来处理 CUDA 的 PTX 虚拟指令集。利用 Ocelot 可以获得一个应用在执行过程中的全部 PTX 指令代码，进而根据 native 指令与

PTX 指令的对应关系可以获得该应用在硬件上执行不同种类 native 指令的数目。

本文选取的 13 个应用首先利用 CUDA profiler 进行采样，然后利用 Ocelot 工具对其进行指令分析，获得其指令执行次数。所有应用内核使用的指令类型及访存情况如表 3 所列。

表 3　内核指令类型列表

Kernel	Int	Float	Double	SFU	Shared mem	L1/L2/Dram
MC(float)		Y		Y		Y/Y/Y
MC(double)		Y	Y	Y		Y/Y/Y
SpMV	Y				Y	N/N/Y
matrixmul		Y			Y	Y/Y/Y
Reduction	Y					N/N/Y
Sgemm		Y			Y	N/Y/Y
Blackscholer		Y		Y		N/N/Y
Transpose	Y				Y	N/N/Y
STREAM	Y		Y			N/N/Y
histogram	Y				Y	N/N/Y
MersenneTwister	Y					Y/Y/Y
quasirandomGenerator	Y	Y				Y/Y/N
sortingNetworks	Y				Y	N/N/Y

在分析了所有内核应用并获得了指令执行情况和访存情况之后，利用这些应用对功耗模型进行验证。

如式(8)－式(10)所示，将参数提取中获得的各个指令的数据分别乘以利用 Ocelot 工具及 CUDA profiler 采样获得的指令执行数目和访存次数，叠加获得对应用的功耗分析和预测结果。预测结果与实测结果的对比如表 4 所列，对比效果如图 5 所示，可以得到本文提出的体系结构级 GPU 功耗模型对应用功耗的预测效果达到了 90%左右。其中测量精度相对较高的(精度达到 98%以上)应用有 MonteCarlo，Reduction，Transpose，STREAM；精度居中(精度在 92%～98%之间)的应用有 SpMV，Matixmul，Sgemm，MersenneTwister，sortingNetworks；精度较差(精度低于 92%)的应用有 Histogram，Black-scholer，quasirandomGenerator。

表 4　测试结果与功耗模型结果对比

	测试功耗(nj)	模型功耗(nj)	误差(%)
MC(float)	200	205	24.8
MC(double)	230	225	24
SpMV	180	200	26
matrixmul	180	170	24.2
Reduction	180	190	25
Segmm	200	210	25.5
blackscholer	200	220	27.5
transpose	185	190	25.2
stream	165	170	25
histogram	165	190	28
MersenneTwister	165	170	26
quasirandomGenerator	160	170	28
SortingNetworks	180	200	26

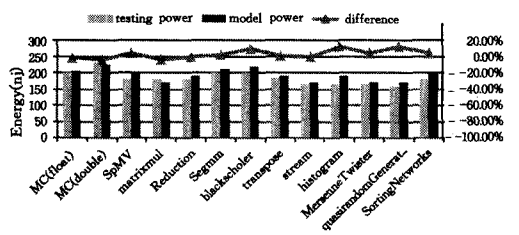

图 5　测试结果与功耗模型结果对比图

Monte Carlo(float)与 Monte Carlo(double)的误差分别为 1.42%与 1.67%，这两个内核中包含了大量的浮点操作与双精度指令操作，虽然有大量存储器操作，但是内核本身仍然是计算密集型，所以主要是计算单元的功耗影响其总功耗。

Reduction 应用主要是对全局存储器中的数据进行访问，整形指令在整个应用中所占比例较小，属于存储密集型应用，因此功耗主要取决于全局存储器访问的功耗。

Transpose 应用主要使用共享存储器，利用共享存储器来减少反复读写的访存开销。因此该应用也属于存储密集型应用，共享存储器功耗测量的准确性决定了这个应用功耗预测的准确性。

STREAM 应用本身是测试不同情况下存储器的访问，所以其预测结果的精度主要受存储器访问功耗精度的影响。由于其测试指令种类比较单一，因此其平均精度超过了 99%。

SpMV 与 transpose 相比，不仅包含大量的访存指令，同时还包含大量的乘加、乘法和加法操作。SpMV 应用相较于前几个应用复杂了很多，而且其中包含了一些 divergence 分支，因此功耗模型受其影响精度略有下降。

MersenneTwister 应用中包含大量的访存指令，并且访存的使用率并不高，因此受其影响，其精度相比于 MonteCarlo 更低。

SortingNetworks 中的指令类型与 Reduction 相类似，但是其中包含了很多同步指令，造成程序执行时间延长，影响了 warp 调度及执行效率，因此功耗模型预测的结果相较于实测结果偏大。

Histogram 应用计算单元较少，主要集中在存储器访问中。但 histogram 中包含了大量的 divergence 分支，存在大量的线程空执行，影响了 GPU 的执行效率，因此测量精度只在 90%左右。

Blackscholer 应用中包含大量计算指令，特别是超长指令，这影响了指令的执行效率。并且该应用还包含了大量的访问存储器操作。因此这些因素影响了功耗模型的预测精度，只有 92%左右。

quasirandomGenerator 应用本身代码量不大，执行时间较短，在测量时受其他干扰相对较大，影响了测量的精度。测量结果与功耗模型的预测结果有 9.1%左右的误差。

根据对每个应用的详细分析发现，精度较高的应用的共同特点是其中执行的指令类型或访问存储器种类比较单一，而且主要是存储密集型偏多。精度在 92%～98%之间的应用，这些内核中有的是指令类型较为复杂，影响了应用精度；有的是存在 divergence 影响或 warp 调度的影响，降低了功耗模型的预测精度。精度较低的应用一般分为两种情况：一种是测量应用中包含了大量超长指令和显存访问指令，又不能被其他指令掩盖其执行延迟，导致程序执行时间较长，执行效率偏低，使功耗模型不能准确获得其结果；另一种是被测试的应用因自身原因而干扰较大，不能反映出真实的功耗情况。

结束语　随着 GPU 越来越多地被应用到通用计算领域，人们也日益关注其性能以外的表现，如可靠性、功耗等等。本文在深入研究 GPU 架构的基础上，根据不同类型计算指令的情况以及访问存储器种类、次数，提出一种高精度体系结构级 GPU 功耗模型。通过对 13 个基准测试应用的评测表明，该功耗模型的预测精度达到 90%左右。

参 考 文 献

[1] Luebke D, Harris M, Govindaraju N, et al. GPGPU: General-purpose computation on graphics hareware[C]// Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06). Tampa, Florida, 2004: 87-91

[2] Lin Y S, Yang X J, Tang T, et al. A GPU low-power optimization based on parallelism analysis model [J]. Chinese Journal of Computers, 2011, 34(4): 705-716(in Chinese)
林一松,杨学军,唐滔,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

[3] Sumpo H, Hyesoon K. An integrated GPU power performance model [C]// Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10). New York, NY, USA, 2010: 280-289

[4] Wang G B. Model-driven multi-dimensional lower-power optimization method for GPU [J]. Chinese Journal of Computers, 2012, 35(5): 979-988(in Chinese)
王桂彬. 模型指导的多维 GPU 软件低功耗优化方法[J]. 计算机学报, 2012, 35(5): 979-989

[5] Tiwari V, Malik S, Wolfe A, et al. Instruction level power analysis and optimization of software [C]// Proceedings of Ninth International Conference on VLSI Design. Jan. 1996: 326-328

[6] Sinha A, Chandraksan A. Joule Track-a Web based tool for software energy profiling [C] // Design Automation Conference, 2001. 2001: 220-225

[7] Steinke S, Knauer M, Wehmeyer L, et al. An accurate and fine grain instruction-level energy model supporting software optimizations [C]// Proc. Int. Wkshp Power, Timing Modeling, Optimization and Simulation PATMOS. September 2001: 1-10

[8] Landman P. High-level power estimation [C]// Proceedings of the 1996 International Symposium on Low Power Electronics and Design. Piscataway, NJ, USA, 1996: 29-35

[9] Isci C, Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data[C]// Proceedings MICRO-36. 2003

[10] Ma X, Dong M, Zhong L, et al. Statistical Power Consumption Analysis and Modeling for GPU-based Computing[M]// Workshop on Power Aware Computing and Systems(Hot Power 09). 2009: 1-5

[11] NVIDIA Inc. NVIDIA PerfKit[OL]. http://developer. nvidia. com/nvidia-perfkit

[12] Nagasaka H, Maruyama N, Nukada A, et al. Statistical Power Modeling of GPU Kernels Using Performance Counters[C]// Proceedings of International Green Computing Conference. 2010: 115-122

[13] Extech Instruments Corporation. Power Analyzer Model 380801 [OL]. http://www. extech. com/instruments/resource/manuals/380801_803_UM. pdf

[14] Zhang Y, Parikh D, Sankaranarayanan K, et al. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects[R]. Technical Report, University of Virginia, 2003

[15] NVIDIA Inc. GPU Computing SDK [OL]. http://developer. nvidia. com/gpu-cpmputing-sdk

[16] Parboil Benchmark suite [OL]. http://impact. crhc. illinois. edu/parboil. php

[17] Che S, Boyer M, Meng J, et al. Rodinia: A benchmark suite for heterogeneous computing[C]// Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC'09). 2009: 44-54

[18] Lai Y, Spanier J. Applications of Monte Carlo/quasi-Monte Carlo methodes in Finance: Option pricing[C]// Monte Carlo/quasi-Monte Carlo Methods. Proceedings of a conference held at the Claremont Graduate Univ. , CA, USA, June 1998. Springer, Berlin, 1999: 284-295

[19] Williams S, Oliker L, Vuduc R, et al. Optimization of sparse matrixvector multiplication on emerging multicore platforms[C]// 2007 ACM/IEEE Conference on Supercomputing. 2007: 1-2

[20] Dazevedo E F, Fahey M R, Mills R T. Vectorized sparse matrix multiply for compressed row storage format[C]// International Conference on Computational Science (ICCS). Springer, 2005: 99-106

[21] Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors[C]// Proceedings of the 2009 ACM/IEEE conference on Supercomputing(SC '09). New York, NY, USA, 2009: 1-11

[22] Blelloch G E, Heroux M A, Zagha M. Segmented operations for sparse matrix computation on vector multiprocessors[R]. Technical Report CMU-CS-93-173, School of Computer Science, Carnegie Mellon University. 1993

[23] Williams S, Oliker L, Vuduc R, et al. Optimization of sparse matrix-vector multiplication on emerging multicore platforms[C]// Prceedings of the 2007 ACM/IEEE Conference on Supercomputing. ACM New York, NY, USA, 2007

[24] Harris M. Optimizng Parallel Reduction in CUDA [C]// Proc. of ACM SIGMOD. 2007: 104-110

[25] Li Y, Dongarra J, Tomov S. A note on auto-tuning GEMM for GPUs[R]. Tech. report, LAPACK Working Note 212, 2009

[26] Black F, Scholer M. Th Pricing of Options and Corporate Liabilities[J]. Journal of Political Economy, 1973, 81(3): 637-654

[27] Ruetsch G, Mcikevicius P. Optimizing Matrix Transpose in CUDA[R]. Nvidia Cuda Sdk Application Note, 2009

[28] Mathworld W. Histogram [OL]. http://mathworld. wolfram. com/Histogram. html

[29] Matsumoto M, Nishimura T, Twister M. A 623-dimensionallly euidistributed uniform pseudorandom number generator [J]. ACM Transactions on Modeling and Computer Simulation, 1998, 8(1): 3-30

[30] Matsumoto M, Nishimura T. Dynamic creation of pseudorandom number generators[C]// Monte-Carlo and Quasi-Monte Carlo Methods 1998: 56-69

[31] Kipfer P, Segal M, Westermann R, et al. Uberflow: A gpu-based particle engine[C]// In ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware 2004. Grenoble, France, 2004: 115-122

[32] GPUOcelot[OL]. http://code. google. com/p/gpuocelot