基于指导语句的CUDA程序性能分析工具研究与实现

李建江，路 川，张 磊

(北京科技大学计算机科学与技术系 北京 海淀区 100083)

【摘要】近年来，GPU的快速发展与NVIDIA公司推出的CUDA技术，推动着GPU在高性能计算领域中的应用。研究并实现CUDA程序性能分析工具，对充分利用GPU的计算优势和提高CUDA架构下并行程序的执行性能具有重要的意义。该文分析了GPU硬件平台的特点和CUDA并行编程模型，结合CPU集群环境下并行程序的性能分析，设计并实现了一种基于指导语句的CUDA程序性能分析工具，并实验验证了其在不同GPU硬件平台上的有效性。

关 键 词 CUDA; 指导语句; 高性能计算; 性能分析; 程序优化

中图分类号 TP316.4 文献标识码 A doi:10.3969/j.issn.1001-0548.2012.02.021

Research and Implementation of Performance Analysis Tool for CUDA Programs with Directive

LI Jian-jiang, LU Chuan, and ZHANG Lei

(Department of computer science and technology, University of science and technology Beijing Haidian Beijing 100083)

Abstract In recent years, the rapid expansion of graphics processing unit (GPU) as well as the computer unified device architecture (CUDA) technology proposed by NVIDIA pushes forward the application of GPU in the field of high performance computing (HPC). In this paper, GPU's architecture and CUDA programming model are introduced first. According to the method of parallel program performance analysis in CPU cluster mode, a performance analysis tool for CUDA programs based on directive is designed and implemented. Experiment results validate the validity of this performance analysis tool on different GPU hardware platforms.

Key words CUDA; directive; HPC; performance analysis; program optimization

并行计算机的引入是为了解决科学研究、工程技术及大规模数据处理中的关键计算问题。进入21世纪以来，随着科技水平的不断发展，许多重要科技问题所处理的数据规模已达到TB甚至PB量级，这对计算机速度提出了严峻的挑战。

目前，并行计算机大多以集群方式搭建，各节点间通过网络互连进行通信，相互协作完成并行计算任务，以达到提高科学计算能力的目的。通常，通过提高单节点的计算能力或增加集群计算节点的数量来提高集群的整体计算能力，以达到提高集群环境下并行程序执行性能的目的。然而，该方法不仅耗费财力，而且对于一个给定的程序，简单增加计算节点的数量并非一定能提高其执行性能。针对上述问题，人们开始从单纯的CPU处理器转而借助计算能力更为强大的GPU处理器来加速计算，并采用异构集群的模式提高并行程序的执行性能。2007年NVIDIA推出的CUDA架构进一步推动了GPU在高性能计算领域中的应用。

本文主要研究CUDA架构的特点以及CUDA架构下并行程序的性能分析与优化技术。结合CPU集群环境下并行程序的性能分析与优化方法，提出并实现了基于指导语句的CUDA程序性能分析工具。

1 基于GPU硬件平台的CUDA并行编程模型

在CUDA编程模型[1]中，一个完整的CUDA程序由CPU与GPU协作完成。kernel函数中有且仅有一个grid，每个grid只能由一个Device调度执行，grid中的线程没有同步机制。grid由线程block组成，CUDA中的kernel函数实质上是以block为单位执行的，同一个block中的线程需要共享数据。因此，它们必须在同一个多线程流处理器中执行，而block中的每一

收稿日期：2010 – 07 – 26; 修回日期：2011 – 11 – 09
基金项目：教育部科学技术研究重点项目(108008); 北京市教委重点学科(XK100080537)
作者简介：李建江(1971 –)，男，博士，副教授，主要从事高性能计算、并行编译和并行软件工程方面的研究.

个线程则在一个SP上执行。一个block以流水线方式分配到一个多线程流处理器上执行，但是一个多线程流处理器中同一时刻可以有多个活动线程块(active block)在等待执行，即在一个多线程流处理器中可以存在多个block流水执行[2]。在CUDA程序的实际运行中，block会被分割成更小的线程束warp，由连续的32个线程组成。多线程流处理器中的指令是以warp为单位执行的，即SM每发出一条指令，SM中的8个SP会将该条指令执行4遍。如果warp中线程执行不同的指令，则warp的执行效率会大大降低。

2　CUDA架构下的性能分析策略

进行并行程序的性能分析的目的主要是为了寻找程序中的性能瓶颈，指导程序优化，提高程序的执行性能。

目前，在CPU集群环境下对并行程序进行性能分析，国内外已开展了大量的研究工作。现有的并行程序性能分析工具主要分为基于PVM、基于MPI与跨平台3种。其中，比较著名的开源性能分析工具有TAU[3]、Paradyn[4]、ParaGraph[5]以及VAMPIR[6]。另外，高性能计算厂商也开发了一些优秀的性能分析软件，如IBM的HPM[7](hardware performance monitor)、Intel的Vtune[8]等。

与MPI相比，OpenMP具有易编程性与支持增量并行化等优点。文献[9]设计并实现了一个面向分布式存储系统结构的OpenMP编译系统。文献[10]针对OpenMP程序，提出了加权剩余并行效率的概念以及源程序级同步段负载的监测方法与均衡策略。

针对CUDA架构的特点，本文借鉴OpenMP规范中的指导语句与TAU对OpenMP程序的性能分析方法，采用动态分析[11]技术设计基于指导语句的CUDA程序性能分析工具。在CUDA源程序中插入性能分析函数，动态获取程序并行执行的性能数据。应用程序员可选择性地获取CUDA程序中的性能数据，主要包括host程序中某一代码段的执行时间、CPU与GPU内存拷贝的时间、kernel中某一代码段的执行时间、kernel中的block内最优线程数等。

3　基于指导语句的CUDA程序性能分析工具设计与实现

3.1　性能分析工具基本框架及流程

NVIDIA公司开发的CUDA visual profiler[12]作为一种图形化性能分析工具，简单易用，且显示直观，但灵活性不足，只能对某些特定性能指标进行分析，无法为改进程序性能提供有效方案。

本文设计并实现的CUDA架构下的性能分析工具具有更好的可控性。通过插入指导语句灵活控制性能分析的位置，并可通过该工具分析出适合程序与当前硬件结构的block内最优线程数。该性能分析工具主要由分析模块、中间转换模块及性能数据采集显示模块3部分组成，如图1所示，各下层模块为上层模块提供接口以获取下层模块的分析结果，其解析流程如图2所示。

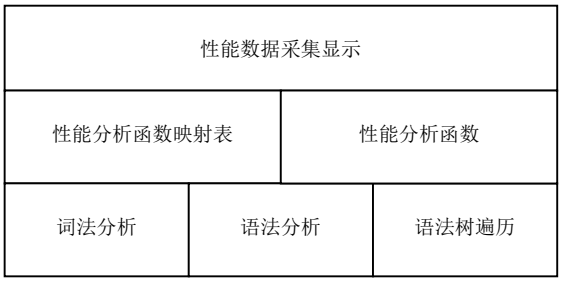

图1　CUDA架构下性能分析工具的基本框架

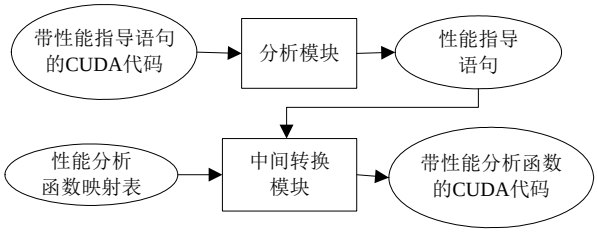

图2　CUDA程序性能分析工具的解析流程

图2中，分析模块由词法分析器、语法分析器与语法树遍历器组成。该层对CUDA程序源代码文件进行词法分析与语法分析，找出程序中的性能分析指导语句，同时产生保存性能指导语句的相关信息文件。中间转换模块根据分析模块所提供的性能分析指导语句相关信息文件和性能指导语句，利用性能分析函数映射表，实现CUDA程序中性能指导语句的替换，在源程序中自动插入性能分析函数。性能数据采集分析显示模块对程序执行过程中的数据进行收集与处理，并在终端进行显示。

3.2　性能分析指导语句的定义及性能分析函数的实现

在CUDA程序中插入性能指导语句，是根据应用程序员对CUDA程序的理解，在程序中插入一些关键词，并由性能分析工具处理，获取程序中不同部分性能分析参数的一种方式。根据要获取的性能信息定义不同的性能指导语句，一条性能指导语句对应一个性能分析函数，通过性能分析工具对其进

行自动替换。

在本文提出的CUDA架构下性能分析工具的设计中，所要获取的CUDA程序的性能数据主要包括有4种方式及性能指导语句，其定义如下：

1) 获取host程序中某一段代码执行的时间。为了保证获取的host程序性能数据的可比性，在性能分析函数中采用CUDA架构提供的时间测量API。测量host程序执行时间的性能分析指导语句为：#program cuda time inst begin与#program cuda time inst end。

2) CPU与GPU内存拷贝时间的获取。CPU与GPU之间通过PCI-E插槽实现数据交换。受带宽限制，CPU和GPU之间的数据交换需要耗费大量的时间，成为程序的性能瓶颈。所以，应将CPU与GPU内存之间的数据交换时间作为程序性能分析的关键点予以监测。测量CPU与GPU内存拷贝时间的性能分析指导语句为：#program cuda memcpyCPUtoGPU begin与#program cuda memcpyCPUtoGPU end。

3) 获取kernel中某一代码段的执行时间。由于kernel程序由grid中所有线程以局部并行、全局流水线的方式并发执行，且在grid中没有现成的同步机制。因此，如果要实现kernel中的线程同步，只能采用拆分kernel的方式。然而，该方式不仅实现复杂，而且很多情况下涉及线程对数据的重新划分，这对程序的性能有一定的影响。而如果采用测量每个线程的执行时间并求其平均值的方法，以获取kernel中某一代码段的执行时间，将耗费大量的时间，影响程序的性能，而且存储程序中的测试结果也需要占用GPU资源，所以该方式也不可取。在获取kernel中某一代码段执行时间的函数中，本文采用只针对某一线程进行测量，虽不能获得kernel中某一代码段的准确执行时间，但并不影响找出程序的性能瓶颈。获取kernel中某一代码段的性能分析指导语句为：#program cuda kernelTime inst begin与#program cuda kernelTime inst end。

4) 测试block中线程数对kernel程序的影响，获取block内最优线程数。SM处理器中可同时执行的block数与block所占用的SM中的共享存储器与寄存器资源的多少有关。GT200硬件核心每个SM中最多可同时执行8个线程block。在测试过程中，通过遍历执行kernel找出block内最优线程数。为了减少对程序的影响，在遍历过程中每次执行kernel程序时，需要对GPU的内存重新进行初始化。由于CUDA架构下每条指令的最小执行单位为warp，所以，在遍历执行时block内的线程数可最小16，依次递增32，最大为512。获取kernel中block内最优线程数的性能分析指导语句为：#program cuda kernel blocksize。

3.3 性能分析指导语句的识别与转换

在本文提出的CUDA架构下的性能分析工具中，分析模块实现程序中性能分析指导语句的识别，它对输入的源程序进行词法分析与语法分析，产生与源文件中性能分析指导语句相关的信息并提供给转换模块。转换模块根据性能分析指导语句与性能分析函数之间的映射关系，完成源程序中性能分析函数的自动替换，从而生成带有性能分析函数的CUDA程序代码。CUDA程序代码的解析转化过程如图3所示。

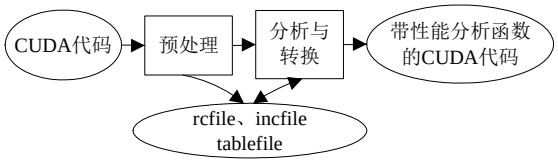

图3 CUDA程序代码解析转化实现流程

1) 预处理阶段。在该阶段中，根据输入文件的扩展名，判断是否为CUDA程序，如果扩展名为.cu，则该文件为CUDA程序；否则提示输入文件错误。性能分析工具会在本目录下的cuda.rc文件中记录相关的信息。如果输入的是已处理过的CUDA程序，则性能分析工具会以cuda.rc中的信息记录判断，提示该源码已经完成解析。另外，针对每一个进行性能分析的CUDA程序，性能分析工具会将所替换的性能分析语句的位置记录在以程序名加后缀.cuda.inc命名的incfile文件中。

2) 代码解析与转换。在对输入的CUDA程序进行解析之前，调用init_handler( )函数生成性能分析指导语句与性能分析函数之间的映射表，建立性能分析指导语句与性能分析函数间的对应关系。然后，由process_cuda( )函数实现对CUDA程序的代码解析。process_cuda( )函数逐行读入输入的CUDA程序，排除程序中的注释、常规程序语句，找到插入的性能分析指导语句，将其位置记录在incfile文件中。finalize_handler( )函数实现CUDA程序中性能分析指导语句的转换功能。根据incfile文件中包含的性能分析指导语句信息以及之前生成的性能分析指导语句与性能分析函数映射表，finalize_handler( )函数能够实现性能分析函数的自动插入转换，如图4所示。

```
……
// setup execution parameters
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(WC / threads.x, HC / threads.y);
#pragma cuda time inst begin

// execute the kernel
 matrixMul<<< grid, threads >>>
              (d_C, d_A, d_B, WA, WB);

#pragma cuda time inst end
// check if kernel execution generated and error
CUT_CHECK_ERROR("Kernel execution
failed");
……
```

```
……
// setup execution parameters
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(WC / threads.x, HC / threads.y);
// create and start timer
unsigned int timer = 0;
cutCreateTimer(&timer);
cutStartTimer(timer);
#line 127 "test.cu"

// execute the kernel
matrixMul<<< grid, threads >>>
              (d_C, d_A, d_B, WA, WB);

// stop and destroy timer
cutStopTimer(timer);
kernel_timer = cutGetTimerValue(timer);
cutDeleteTimer(timer);
#line 132 "test.cu"
// check if kernel execution generated and error
CUT_CHECK_ERROR("Kernel execution
failed");
……
```

图4　CUDA程序性能分析指导语句转换前后的程序片段

4　实验及结果分析

在不同GPU硬件环境下，分别以稀疏矩阵向量乘算法及图像中值滤波处理为例，使用本文研究并实现的CUDA架构下性能分析工具，找出kernel中的block内最优线程数，并对其进行实验验证。

实验以GPU Server(详细硬件配置如表1所示)和GPU PC(详细硬件配置如表2所示)为硬件资源环境。

表1　USTB GPU Server硬件环境 (GT200)

	Host		Device
处理器	2 × Intel Duo 2.8GHz/3072K Cache	GPU	2×NVIDIA GTX 295显卡
内存	8 GB	GPU硬件核心	4×GT200
硬盘	500 GB	显存	2×1G
网卡	2× Intel EtherExpress/1000	SP	4×240
PCI-E	16 ×PCI-E插槽2个	MP	4×30

在稀疏矩阵向量乘算法测试用例中，使用block kernel、warp kernel与subsection kernel共3种不同的数据划分与优化方式实现稀疏矩阵向量乘的kernel程序。其中，block kernel中稀疏矩阵的每一行由一个block执行；warp kernel中稀疏矩阵每一行由一个warp执行；subsection kernel则是基于warp kernel对GPU访存进行优化。

表2　USTB GPU PC硬件环境(G80)

	Host		Device
处理器	2×Intel Duo 2.20 GHz/2048K Cache	GPU	1×NVIDIA GTX 295显卡
内存	1 GB	GPU硬件核心	1×G80
硬盘	250 GB	显存	1×332M
网卡	1× Intel EtherExpress/1000	SP	1× 96
PCI-E	16 ×PCI-E 插槽1个	MP	1×12

在GT200硬件环境下，使用本文实现的性能分析工具测试出block kernel的最优线程数为96，warp kernel的最优线程数为64，subsection kernel的最优线程数为64。

将线程数设定为从32开始，依次递增16，最大为512，分别进行手动测试。在不同线程数情况下kernel的执行时间如图5所示。

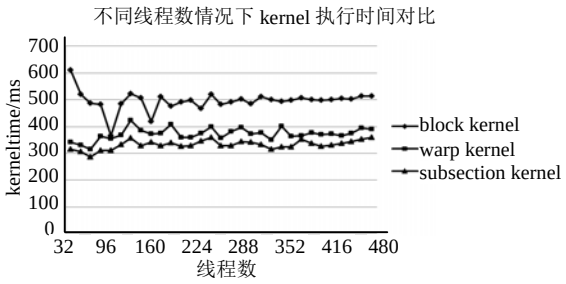

图5　在不同线程数情况下kernel的执行时间(GT200)

由图5可知，手动测试得到的block kernel的block内最优线程数与本文实现的性能分析工具的测试结果一致，而且block kernel的执行效率并不随着block内线程数的增加而提高。其原因是，在block kernel中如果block内线程数过大会出现线程空转，导致资源的浪费。与此相比，warp kernel与subsection kernel只是增加了对全局内存存取的合并执行，所用资源与block kernel基本一样。因此，随着block内线程数的变化，warp kernel与subsection kernel的性能变化的情况与block kernel大致相同。

当硬件环境发生改变，每个线程在多线程处理器中所获取的资源也随之改变，则kernel中的block内最优线程数也会发生变化。在G80硬件环境下，通过本文实现的性能分析工具测试出block kernel的最优线程数为288，warp kernel的最优线程数为128，

subsection kernel的最优线程数为480。在G80硬件环境下使用手动测试，在不同线程数情况下kernel的执行时间如图6所示。

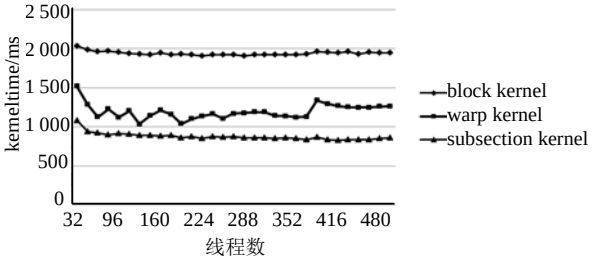

图6 在不同线程数情况下kernel的执行时间(G80)

由图6可知，使用本文实现的性能分析工具测试出的block内最优线程数与手动测试结果一致，与GT200硬件环境下的测试结果相比，相同kernel程序中的block内最优线程数发生了明显的变化。

在图像中值滤波处理中，利用本文实现的性能分析工具，以4×4的滤波窗口对多幅1024 ×1024的图片进行中值滤波测试。在GT200和G80硬件环境上测试出的block内最优线程数分别为128和224，与手动测试结果一致，如图7所示。

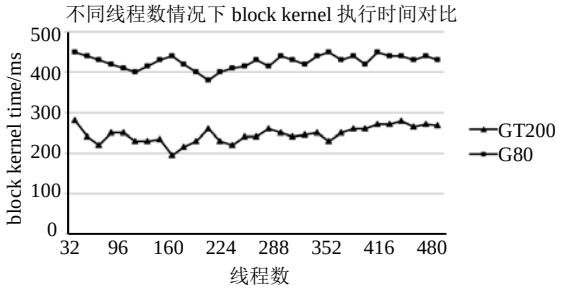

图7 不同线程数及硬件环境下block kernel的执行时间

因硬件平台变化带来的GPU资源的变化、编程执行模型中各执行模块与硬件执行单元之间相互制约的关系、程序编译时对线程所占资源的优化等因素决定了CUDA程序中最优线程数的不可预知。使用性能分析工具确定不同硬件环境下CUDA程序中的block内最优线程数，对提高CUDA程序的执行性能具有重要的意义。

5 结 论

为提高CUDA架构下并行程序的执行性能，本文针对CUDA架构的特点，设计并实现了基于指导语句的CUDA程序性能分析工具。该性能分析工具比CUDA visual profiler具有更好的可控性，能有效定位CUDA程序中的性能瓶颈并确定kernel函数中的block内最优线程数，从而为CUDA程序在不同GPU硬件环境下获得最高执行性能提供依据。

在下一步的研究工作中，将增加CUDA程序执行时GPU硬件信息的读取与分析，改进最优线程数的查找算法以缩短分析时间，借用图形化手段显示CUDA程序的性能分析数据等，进一步完善基于指导语句的CUDA程序性能分析工具的功能。

参 考 文 献

[1] NVIDIA. NVIDIA CUDA programming guide-version 2.1[EB/OL]. [2010-07-01]. http://www.nvidia.com/object/cuda_develop. html.

[2] NICKOLLS J, BUCK I, GARLAND M, et al. Scalable parallel programming with CUDA[J]. Queue, 2008, 6(2): 40-53.

[3] SHENDE S S, MALONY A D. The TAU parallel performance system[J]. International Journal of High Performance Computing Applications, 2006, 20(2): 287-331.

[4] MILLER B P, CALLAGHAN M D, CARGILLE J M, et al. The paradyn parallel performance measurement tool[J]. Computer, 1995, 28: 37-46.

[5] BLUEMKE I, FUGAS J. C code parallelization with paragraph[C]//Proceeding of the 2nd International Conference on Information Technology. Gdansk: Poland, 2010: 28-30.

[6] MOOER S, CORNK D, LONDON K, et al. Review of performance analysis tools for MPI parallel programs[J]. Lecture Notes In Computer Science, 2001, 2131: 241-248.

[7] DEROSE L. Hardware performance monitor (HPM) toolkit[Z]. [S.l.]: IBM Research Advanced Computing Technology Center, Version 2.5.1 edition, 2003.

[8] REINDEERS J. Vtune Performance Analyzer Essentials[M]. [S.l.]: Intel Press, 2005.

[9] 王珏, 胡长军, 张纪林,等. 面向分布式存储系统结构的OpenMP编译系统[J]. 中国科学F辑, 2010, 40(5): 678-691.
WANG Jue, HU Chang-jun, ZHANG Ji-lin, et al. A openMP compiler system architecture for the distributed storage system[J]. Science China F, 2010, 40(5): 678-691.

[10] 李建江, 舒继武, 陈永健, 等. OpenMP源程序级同步段负载监测方法与均衡策略[J]. 电子学报, 2005, 33(5): 852-856.
LI Jian-jiang, SHU Ji-wu, CHEN Yong-jian, et al. A load monitoring method and balancing strategy at source level of openMP programs for synchronization segments[J]. Acta Electronica Sinica, 2005, 33(5): 852-856.

[11] BOYER M, SKADRON K, WEIMER W. Automated dynamic analysis of CUDA programs. [EB/OL]. [2010-06-28]. http://www.cs.Virginia.edu/~skadron/perpers/stmcs08.pdf.

[12] NVIDIA. CUDA visual profiler[EB/OL]. [2010-07-01]. http://www.nvidia.cn/object/cuda_ develop_cn.html.

编 辑 张 俊