文章编号：2095-2163(2020)02-0341-05 中图分类号：TP391.4 文献标志码：A

面向 CUDA 程序的线程放置优化策略研究

谢根栓，张伟哲
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：GPU 具备强大的数据并行处理与浮点计算能力，因而被越来越广泛地应用于数值模拟和科学计算等领域。提高 GPU 上程序开发效率以及程序的性能就显得尤为重要，线程放置策略是其中重要的一环。本文在程序信息基础上，使用机器学习算法建立了 CUDA 程序线程放置优化模型。本文首先在运行时信息采集环节做了改进，只保留了其中与程序性能相关的核心信息。接着，提出并实现了基于 LLVM 的静态信息替代运行时信息的方法，进一步降低了信息采集环节的时间。此外，提出了全新的设置标签算法。综合上述改进，最终确定了采用支持向量机算法、借助网格搜索方法择优的模型。本文在公开程序集上选择与已有模型在同等条件下展开测试，结果表明本文的模型比现有模型在时间上更具优势，并获得了更高的训练准确率。

关键词：CUDA；线程；机器学习；LLVM

Research of thread placement optimization strategy for CUDA programs

XIE Genshuan, ZHANG Weizhe
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】 GPU has powerful data parallel processing and floating-point computing capabilities, so it is more and more widely used in numerical simulation and scientific computing. It is particularly important to improve the efficiency of program development and program performance on GPU. Thread placement strategy is an important part of it. On the basis of program information, this paper establishes an optimization model of thread placement in CUDA program by using machine learning algorithm. Firstly, this paper improves the information acquisition link at runtime, and only retains the core information related to program performance. Then, a method of replacing runtime information with static information based on LLVM is proposed and implemented, which further reduces the time of information acquisition. In addition, the paper proposes a new label setting algorithm. By synthesizing the above improvements, the model of selecting the best by using the support vector machine algorithm and the grid search method is finally determined. The open assembly on the same condition is tested between the existing model and the propesed model. The results show that the proposed model has more advantages in time and higher training accuracy than the existing model.

【Key words】 CUDA; thread; machine learning; LLVM

0 引 言

近年来，基于强大的数据并行处理与浮点计算能力，GPU 已广泛应用于工程应用和科学计算领域，并且取得了长足的进步和可观成绩。在 GPU 程序中，程序员需要设置线程块大小(block size)以明确程序的线程数量。不同的线程块大小会带来不同的线程的并发度。只有通过合理设置线程块大小才能使程序运行的性能达到最好，因此线程块大小是影响程序性能的关键因素。

本文中，使用机器学习完成性能最佳线程块大小的自动设置，可以避免全局遍历方法中多次反复运行程序的高耗时，相较于传统启发式搜索等方法，也避免了搜索结果陷入局部最优的问题。

1 相关工作

目前，CUDA 线程块放置优化方法主要有 2 类。一类是不运行程序、简单地依靠经验完成线程块优化；另一类是基于性能模型驱动的优化。

用第一类方法进行线程块优化的典型代表有 Sorensen 等人[1]提出的在 Fermi 架构 GPU 上对一系列 BLAS 程序设置线程块大小等参数的方法。

基于性能模型驱动的优化研究成果相对多一些。Tran 等人[2]开发了一个评估线程块大小优化效果的调参模型，该模型会提出多个备选的线程块大小，从而避免了大范围穷举线程块大小。但该模型评估标准只是 GPU 利用率，并且忽略了决定线程块大小更重要的 kernel 函数的信息。Gupta 等人[3]设计了一款名为 STAtuner 的工具。该工具使用 LLVM 编译 CUDA 核函数以从中收集 kernel 函数的多个静态信息。在此基础上，还使用支持向量机预测性能最佳的线程块修正方向。Connors[4]提出了

作者简介：谢根栓(1990 -)，男，硕士研究生，主要研究方向：高性能计算；张伟哲(1976 -)，男，博士，教授，博士生导师，主要研究方向：高性能计算、云计算、网络信息安全等。

收稿日期：2019-06-13

类似的思路,都是使用机器学习来获得最佳线程规格调整方向,不同之处在于使用动态程序信息而不是静态程序信息。

国内关于 CUDA 程序线程放置优化方面,郑祯等人[5]设计实现了一套针对 GPU 程序性能的分析工具。该工具的编程接口采用 CUPTI 底层接口,在测试环节对部分 Benchmark 程序做性能瓶颈分析,并分析总结了常见性能瓶颈的原因,同时给出了一些开发高效 CUDA 应用的建议。

2 模型研究

2.1 整体设计

本文的模型研究基于机器学习的常规设计,主要包括训练数据获取、训练模型、模型评估三个步骤。模型整体设计见图 1。

2.2 程序信息采集与处理

2.2.1 采集运行时信息

nvprof 是 NVIDIA 官方提供的一款命令行形式的 CUDA 程序的性能分析工具,可以收集 GPU 的硬件计数器信息生成程序的运行时信息。本文参考郑祯等人[5]对 CUDA 性能信息的提炼汇总,只采集这些信息中最具代表性的一部分。本文获取的 metric 信息见表 1。此外,本文训练数据的标签数据也通过 nvprof 采集 CUDA 程序执行时间获得。

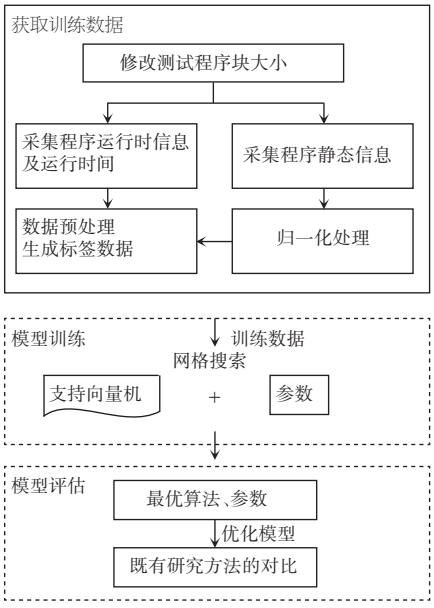

图 1 基于 LLVM 对 CUDA 程序线程分配优化模型的改进

Fig. 1 Improvement of thread allocation optimization model for CUDA program based on LLVM

表 1 通过 nvprof 获得的 CUDA 函数运行时信息

Tab. 1 Runtime information of CUDA function obtained by nvprof

Metric 名称	描述
branch_efficiency	分支指令占全部指令比率
dram_read_throughput	设备内存加载吞吐量
dram_write_throughput	设备内存存储吞吐量
gld_throughput	全局内存加载吞吐量
gst_throughput	全局内存存储吞吐量
achieved_occupancy	GPU 利用率
gld_efficiency	全局内存加载效率
gst_efficiency	全局内存存储效率
shared_efficiency	共享内存效率
shared_load_throughput	共享内存加载吞吐量
shared_store_throughput	共享内存存储吞吐量
warp_execution_efficiency	各个 warp 中实际运行的线程数量占最大理论线程数量的比率的平均值
warp_nonpred_execution_efficiency	各个 warp 中执行非分支预测指令的线程数量占最大理论线程数量的比率的平均值
flop_dp_efficiency、flop_sp_efficiency	浮点计算单元利用率

2.2.2 静态信息替代运行时信息的方法

由于 nvprof 采集运行时信息会存在耗时过长的问题,研究提出了静态信息替代运行时信息的方法。CUDA 程序静态信息对运行时信息的替代详见表 2。研究可得阐释分述如下。

(1)浮点计算单元利用率。研究可以用静态信息浮点指令数量 N_{instr_float} 计算得到浮点计算单元利用率。设核函数的执行时间为 T_{kernel}, GPU 浮点计算理论峰值为 $P_{\max}$, 则 GPU 浮点计算单元利用率 U_{float_static} 可以表示为:

$$U_{float\_static} = \frac{N_{instr\_float}}{T_{kernel} * P_{\max}}, \quad (1)$$

(2)线程分歧率。在 CUDA 程序的指令集中，影响线程分歧率的指令包括分支指令、同步指令，因此研究中用静态信息分支指令数量 N_{branch}、同步指令数量 N_{sync}、指令总数 N_{instr} 等替代 nvprof 工具的信息 *branch_efficiency*，计算公式为：

$$branch\_efficiency = 1 - \frac{N_{branch} * N_{loop} + N_{sync}}{N_{instr}}. \quad (2)$$

表 2　CUDA 程序静态信息对运行时信息的替代

Tab. 2　Replacement of runtime information by static information of CUDA program

运行时信息	用以替换的静态信息
branch_efficiency	N_{loop}、N_{branch}、N_{sync}、N_{instr}
gst_throughput	N_{instr_store}
gld_throughput	N_{instr_load}
flop_dp_efficiency flop_sp_efficiency	N_{instr_float}

2.2.3　基于 LLVM 的 CUDA 程序信息提取

LLVM 项目是一个获得广泛使用的编译工具框架，Pass 是 LLVM 系统的重要组成部分，可以通过分析程序的模块、函数、基本块等信息以提供程序的高级的信息。本文中，使用 LLVM 完成 CUDA 程序静态信息的采集。对此可做重点剖析如下。

在进入 IR 的 Module 后，分析 pass 会遍历 Module 内的全部函数，进入 kernel 函数内部后，pass 会遍历函数的全部基本块与指令，依据与 CUDA 程序性能指标关联度，pass 主要完成对 int 型数据操作指令、float 型数据操作指令、存储交互操作指令、程序分支操作指令、线程同步指令、函数内循环次数的统计。

分析 pass 的指令信息统计的实现流程如图 2 所示。

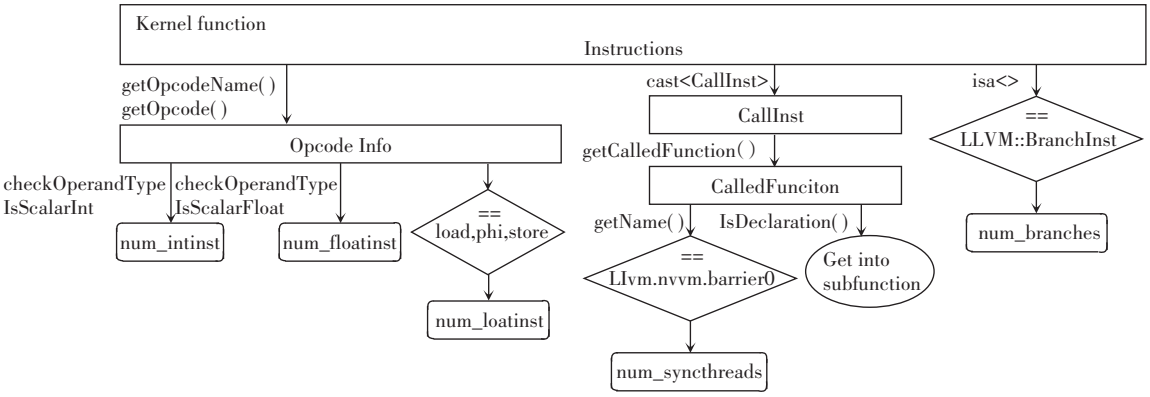

图 2　分析 pass 的指令信息统计的实现

Fig. 2　Implementation of instruction information statistics of analysis pass

2.2.4　设置标签

本文提出的添加标签算法是针对每个核函数，按照执行时间排序，取执行时间较短的前 20%设置标签为 1，剩余的信息设置标签为 0。添加标签算法的过程设计具体如下。

算法 1　设置标签的算法

输入：D 为原始训练数据；F 为测试集中全体核函数集合

输出：D_L 为添加标签后的训练数据

过程：

for f_i in F do

T_{fi} ← 提取 D 中的核函数 f_i 的全部执行时间信息

D_{fi} ← 提取 D 中的核函数 f_i 的全部信息

依据 T_{fi} 对 D_{fi} 快速排序：$D_{fi} \leftarrow QuickSort(D_{fi})$

排序后的 D_{fi} 的前 20% 标签设置为 1：前 20%D_{fi} 的 $D_L \leftarrow 1$

剩余 D_{fi} 的标签设置为 0：剩余 D_{fi} 的 $D_L \leftarrow 0$

end for

2.3　模型训练

scikit-learn[6] 工具为机器学习提供了良好的支持，本文通过其来建立优化模型。文中对此将给出如下研究论述。

2.3.1　参数组合

在小批量数据集下，支持向量机算法的可调参数包括惩罚系数和核函数参数。其中，惩罚系数表

示了模型对误差的宽容度;核函数参数 σ 决定了数据映射到新的特征空间后的分布。研究中分别对应 scikit-learn 中的 C、$gamma$。本次实验中 SVC 算法遍历的参数见表 3。

表 3 SVC 网格搜索的参数

Tab. 3 Parameters of SVC grid search

gamma	C			
	1	10	100	500
10	(10,1)	(10,10)	(10,100)	(10,500)
5	(5,1)	(5,10)	(5,100)	(5,500)
2.5	(2.5,1)	(2.5,10)	(2.5,100)	(2.5,500)
1.25	(1.25,1)	(1.25,10)	(1.25,100)	(1.25,500)
0.625	(0.625,1)	(0.625,10)	(0.625,100)	(0.625,500)
0.312 5	(0.312 5,1)	(0.312 5,10)	(0.312 5,100)	(0.312 5,500)

2.3.2 交叉验证

交叉验证(Cross Validation)也称作循环估计,指的是对原始数据分组,取大部分样本做训练集,留取剩余少部分数据做验证集用来评价训练出的模型效果,重复上述过程,直到全部数据都做过验证集。最后,选取这些验证评价的平均值作为最终的评价。

研究使用 scikit-learn 的网格搜索函数 *GridSearchCV* 遍历 2.3.1 节的参数组合,通过交叉验证确定最佳效果参数。*GridSearchCV* 的参数 *cv* 是交叉验证参数,该参数指定了 fold 数量,本文设置为 10。接着,研究通过调用 *fit* 函数利用建立好的网格搜索对象训练数据,得到最优的参数组合。

3 实验

3.1 实验条件

本次研究中的运行环境配置见表 4。

表 4 模型运行环境

Tab. 4 Model running environment

硬件/软件	版本/大小
CPU	Intel E5-2697 v4 * 2
内存	256 G
操作系统	Ubuntu 16.04
GPU	NVIDIA K40 m
CUDA	9.1

全部代码运行在 python3.6.5 环境。模型训练选用的 Benchmark 为 Rodinia[7]。

3.2 测试分析

在测试分析部分,本文与已有的 Connors[4] 模型进行了相同训练数据下的训练时间、训练精度对比分析。2 个模型运行时间的对比结果如图 3 所示,2 个模型在充足训练集下的训练准确率的对比结果则如图 4 所示。

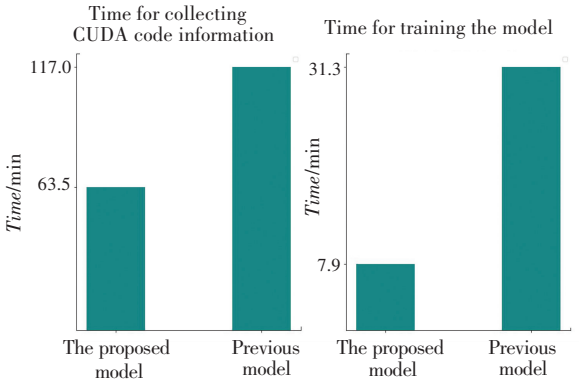

图 3 与已有模型的时间对比

Fig. 3 Time comparison with existing model

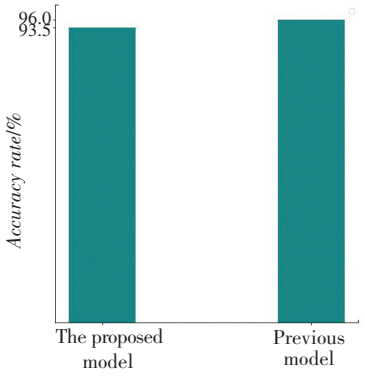

图 4 与已有模型的训练准确率对比

Fig. 4 Comparison of training accuracy with existing model

通过对比 2 个模型的训练数据收集时间和模型训练时间可以看到,本文的模型相较 Connors 等人提出的模型只花费了近一半的时间。究其原因则在于:首先,本文模型采集的 CUDA 程序信息较少,而 Connors 的模型未对 nvprof 采集的程序信息做筛选;其次,本文的模型部分信息转为静态信息提取,降低了采集运行时信息的种类。综上分析可知,本文模型中 nvprof 潜在反复执行 CUDA 程序的次数减少,这使模型训练时间也随之减少。

由图 4 可以看到,在相同机器学习算法下,本文的模型基于上述改进,训练准确率获得了提升。究其原因则在于:首先,研究采集筛选后程序信息使模型避免了冗余信息可能产生的噪点和过拟合问题;其次,研究的模型提出了全新的标签设置算法,可以更真实地反映了不同线程配置下的程序运行性能间的差异。

4 结束语

本文提出了一个基于 CUDA 程序运行时信息

与静态信息结合的线程配置优化模型。研究的创新点包括：

(1)运行时信息采集环节中,对采集信息做筛选,只保留了其中与程序性能相关的核心信息。

(2)提出并实现了基于 LLVM 的静态信息替代运行时信息的方法。

(3)提出了全新的标签设置算法,使模型的标签数据更真实地反映程序性能情况。

基于上述创新,将本文模型与已有研究中的模型在时间、准确率上进行对比分析,结果表明新模型在时间上获得了更大优势,准确率也有更高保证。

参考文献

[1] SORENSEN T, GOPALAKRISHNAN G, GROVER V. Towards shared memory consistency models for GPUS[C]//Proceedings of the 27th International ACM Conference on Supercomputing. Eugene, Oregon, USA:ACM, 2013: 489.

[2] TRAN N P, LEE M. Parameter tuning model for optimizing application performance on GPU [C]//IEEE International Workshops on Foundations and Applications of Self * Systems. Augsburg, Germany: IEEE, 2016:78.

[3] Gupta R, Laguna I, Ahn D H, et al. STATuner: Efficient tuning of CUDA kernels parameters [EB/OL]. [2015]. http://sc15.supercomputing.org/sites/all/themes/SC15images/tech_poster/poster_files/post276s2-file2.pdf.

[4] Connors T A. Automatically selecting profitable thread block sizes using machine learning [D]. San Marcos, Texas: Texas State University,2017.

[5] 郑祯, 翟季冬, 李焱,等. 基于 CUPTI 接口的典型 GPU 程序负载特征分析[J]. 计算机研究与发展, 2016, 53(6):1249.

[6] PEDREGOSA F, VAROQUAUX G, GRAMFORT A, et al. Scikit-learn: Machine learning in Python [J]. Journal of Machine Learning Research, 2011, 12(10):2825-2830.

[7] LEE S H, SKADRON K, SHEAFFER J W, et al. Rodinia: A benchmark suite for heterogeneous computing [C]// IEEE International Symposium on Workload Characterization (IISWC). Austin, TX, USA:IEEE,2009:44.

(上接第 340 页)

4　结束语

医患文化作为文化在医患之间的缩影,时刻影响着医患关系。医患关系紧张有着深刻复杂的根源,真正缓解医患冲突,必须建设新的医患文化,让文化的缝隙逐渐缩小,减少冲突,实现共赢。医者作为医患主体,主导性较强,积极引导,主动承担建设者和沟通者的角色,做好文化榜样,营造良好的医患文化氛围。患者应该提高自身的医学素养,积极配合治疗。医患既对立又统一,做到互相理解,包容彼此,共同构建和谐的医患文化。

参考文献

[1] 何军,夏保京,李晓庆. 医患文化冲突分析—对医患关系紧张的新审视[J]. 医学与哲学,2008,29(19):36.

[2] TONSAKER T, BARTLETT G TRPKOV C. Health information on the Internet: gold mine or minefield [J]. Candadian Family Physician Medecin de Famille Canadien,2014,60(5): 407.

[3] 谢广宽. 互联网技术对医患关系的影响[J]. 中国心理卫生杂志,2015,29(10):755.

[4] 杨一苗,李亚红,廖君,等. 医药分开不能只走一条路[J]. 决策探索,2013(13):26.