面向 CUDA 程序的性能预测框架

曲海成,于思淼,刘万军,王鑫源
(辽宁工程技术大学软件学院,辽宁葫芦岛 125105)

摘 要: 为对 CUDA 并行程序内核性能进行分析和预测,从而指导并行程序设计及性能优化,提出一种性能预测框架.1)从 GPU 编程模型和设备架构细节入手,以线程束为研究单位,通过整合与 GPU 程序用时密切相关的软硬件基本特征,定义了并行空间闲置度、流处理器线程束负载、并行效应因子等高层次性能相关特征.2)基于上述特征,框架针对线程负载均衡型 GPU 程序,评估内核函数在不同问题规模以及执行配置下的执行时间.3)依据性能评估原理提出了内核函数执行配置参数的优化策略.验证实验结果表明,该框架在两种典型情境下对现有程序性能的平均预测准确率分别达到89%和94%,客观归纳了高层次特征与程序性能间的相关关系,且能定性分析并行算法性能水平.

关键词: 性能预测; 线程束; 设备并行空间; 并行效应; 性能特征; 执行配置参数优化
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2020)04-0654-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.04.006

Performance Prediction Framework for CUDA Programs

QU Hai-cheng, YU Si-miao, LIU Wan-jun, WANG Xin-yuan
(*College of Software, Liaoning Technical University, Huludao, Liaoning* 125105, *China*)

Abstract: In order to analyze and predict the performance of CUDA program kernel and guide parallel program design and performance optimization, a performance prediction framework is proposed. This paper starts with the GPU programming model and hardware architecture details, with warp as the research unit. By integrating hardware and software factors closely related to GPU program time, high-level performance-related features such as device parallel space idle degree (DPSID), number of streaming multiprocessor warp (NSMW) are defined. Based on the above features, a framework for evaluating the execution time of kernel functions under different problem sizes and execution configurations is built for thread load balancing GPU programs. The principle of optimizing configuration parameters of kernel function execution is put forward to guide optimizing program performance. The experimental results show that the average prediction accuracy of the framework is 89% and 94% in the two scenarios, respectively.

Key words: performance prediction; thread warp; device active warps; parallel effect; performance features; execution configuration optimization

1 引言

在 NVIDIA 等显卡厂商的推动下,图形处理单元(GPU)逐渐演变为可支持多线程编程、并行计算、具备强大浮点计算能力以及高内存带宽的通用图形处理器(GPGPU),尤其擅长处理可以被转换为数据并行操作的高计算强度问题.同时,为使 GPU 应用能够自主地提高指令并行度以适应不断增加的处理器核心数目,NVIDIA 于 2006 年推出了可扩展并行编程模型[1] CUDA[2].在众多科学领域[3~5]内,基于此平台使用 GPU 取得显著加速效果的应用程序几乎无处不在.然而,并行必定会带来包括通信、线程管理、同步在内的多项额外时间开销.研究如何预估 CUDA 程序时间性能对缩短开发周期、控制开发风险、定位程序性能瓶颈、指导程序性能优化、平衡与其它设备的负载等方面具有重要作用[6].围绕此问题,相关研究提出了一系列与性能有关的高层次特征及度量,在特定问题或场景下建立了性能分析或预测模型.

Amdahl 定律与 Gustafson 定律[7]描述了串行程序并行化后所能取得的性能提升上界;加速比与并行效率用于衡量并行程序性能优化效果.文献[8,9]总结了多

收稿日期:2018-11-26;修回日期:2019-11-12;责任编辑:李勇锋
基金项目:国家自然科学基金青年基金(No.41701479);辽宁省自然科学基金(No.20180550529);辽宁省教育厅科学研究基础研究(No.LJ2019JL010)

项有关访问效率、执行效率等的性能指标,用于对 GPU 程序性能行为进行分析与评估. 文献[10]引入了一个能够准确预测带有控制流分支的 GPU 程序性能的组合性分析度量参数,通过预测不同改进线程重组算法的性能优劣,优化线程控制流分支提升程序性能. 此外,文献[11]在汇编层次提出了一个性能分析框架,通过分析汇编指令,生成 DAG 图来模拟指令行为、分割区块等来实现精确预测. 受机器学习思想的启发,文献[12~14]分别应用 NNGE/SVM Classifier、Stepwise Regression、Random Forest Regression 监督学习方法和机器学习技术建立性能预测模型,学习程序属性、硬件特征与程序性能的相关关系. 他们在各自的测试集上取得的准确率在 70.7% ~90% 之间. 另两个常被用来预测程序性能的方法是数据法和模拟法[15],这两种方法难以在预测精度与计算代价间取得平衡. 在特定场景下,文献[16]针对卷积神经网络提出 DeLTA 模型,基于内存"流量"预测在 GPU 计算资源或内存资源增加的条件下,神经网络所能获得的性能增益. 文献[17]使用后缀树结构和近似匹配算法来提取 GPU trace 中的重复模式,并定义了一些度量量化程序性能. 文献[18]综述了关于 CPU、GPU、FPGA 性能预测模型的最新进展,并从准确率、效率、优缺点等方面对各类模型进行了详尽的对比. 经典研究[5,10]等在问题规模(Size of Problem,SOP)层次上对内核性能实现了粗粒度的预测,并量化了性能与 SOP 间的单调关系,或是建立映射内核功能与性能的模型,采用了加速比[12,19]、GFLOPS[11,15]、并行代价或收益[20]等性能度量指标评估程序性能.

本文则在细粒度的 warp 层次上,针对 warp 负载均衡型 CUDA 程序,以预测现有程序内核函数或并行算法在特定 GPU 上的运行时间(Kernel Duration,KDT)为目标,定义了若干项高层次性能特征,并建立性能预测框架,最后提出内核函数执行配置参数优化策略.

2　CUDA 编程模型与 GPU 硬件架构

2.1　CUDA 编程模型

CUDA 既是并行编程模型,又是通用并行计算平台,同样也是开发者直接面对的软件开发环境,主要涉及了内核,线程层次,存储器层次,异构编程模型四个方面的内容. 一个由 CUDA-C 语言定义,带有__global__修饰符,在 GPU 端(设备端)执行的函数称为内核(kernel),声明及调用形式如图 1 所示.

图 1 执行配置 <<< Dg,Db >>> 参数指定了内核的线程网格(grid)及线程块(block)配置,各有 x,y,z 三个成员属性,大小为属性乘积. Db 尺寸最大被限制为 1024,Dg 指定网格样式,即 block 的组织方式及数目. kernel 被分配的线程总数(threads)等于 Dg 与 Db 的乘积.

```
//  内核声明
__global__ void kernel(float*parameter);
//  内核调用
kernel<<< Dg, Db>>>(parameters);
```

图1　内核声明及调用形式

2.2　GPU 硬件架构

NVIDIA GPU 内置一组多线程流多处理器(Streaming Multiprocessors,SM),SM 内包含多个 CUDA Core/MP. SM 采用 SIMT(Single Instruction,Multiple Threads)执行模型,当 CUDA 程序在 CPU 主机端调用一个内核时,网格中的线程块被分配到 SM 当中等待执行. 线程块以及其中的线程均可以在同一 SM 上并行执行,但同一线程块内的线程不能被分配到不同 SM 当中执行. SM 以 32 个 CUDA 线程为一组,创建、管理、调度以及执行线程,每组线程被称为线程束(warp). warp 调度器(warp schedule)负责调度 warp 在设备端执行,warp 间可相互隐藏访存开销. 线程块内 warp 数目(Number of Block Warp,NBW)的计算公式为:

$$\mathrm{NBW} = \mathrm{ceil}\left(\frac{b}{w},1\right) \tag{1}$$

ceil(m,n)把 m 向上取整到 n 的最近倍数. 本文涉及属性或特征符号及含义见表 1.

表 1　主要符号及其含义

符号	含义	取值
b	线程块大小	$\{b \in N^+ \mid 1 \leqslant b \leqslant 1024\}$
w	线程束大小	32
g	网格大小	$x \in N^+$
n	SM 数量	
c	SM CUDA Core 个数	
D	DPS *	
r	线程使用的寄存器数量	$y \in N$
s	线程块使用的共享内存(bytes)	
A	APB *	
v	NBW	$\{v \in N^+ \mid 1 \leqslant v \leqslant 32\}$
a	SM 激活 warp 数	$\{a \in N^+ \mid 0 \leqslant a \leqslant 64\}$
k	KDT(ns)	$(0,+\infty)$
t	warp 单位用时(ns)	
K	KDTV *	
W	NSMW *	$W \in N^+$
I	DPSID *	$[1,32]$
S	SPEF *	$(0,1]$

注:标 * 符号为自定义特征,x,y 取值上界与设备型号有关.

3 程序性能特征

GPU warp schedule 以 warp 的形式管理和调度 CUDA 线程,因此,本节以 warp 为基本研究单位,提出 5 项高层次特征作为分析程序性能的基础理论.

(1)设备并行空间

为描述 GPU 理论上能够并行的 warp 数量以及 GPU 可用计算资源的相对大小,以设备占有率(occupancy)指标为依据,定义设备并行空间(Device Parallelism Space,DPS)特征,计算方法为:

$$\mathrm{DPS}=n\times\mathrm{calculator}(b,r,s) \tag{2}$$

上式中,calculator 为 CUDA 提供的设备占有率计算工具[21],返回结果为 SM 激活 warp 数目.

(2)可绝对并行线程块数目

依据 CUDA Core 数量,量化 warp 执行时间可完全重合的线程块数目. 假设每一个 CUDA Core 独立执行 1 个 CUDA 线程,可绝对并行线程块数目(Absolute Parallel Block,APB)定义如下:

$$\mathrm{APB}=\mathrm{floor}\left(\frac{\min(c,a\times w)}{w\times v}\right)\times n \tag{3}$$

其中,floor(x)将 x 向下取整. 此时 $k\approx t$.

(3)SM warp 负载

warp schedule 发射 warp 到不同 SM 上,分别记分配到单个 SM 上的 warp,block 数目为(Number of SM warp,NSMW),(Number of SM Block,NSMB). 在同一 block 内的 warp 不能被分配到不同 SM 上的前提下,假设不存在非满载 warp(有空闲线程),且它们在 SM 间被平均分配,从而定义 NSMW 计算方法为:

$$\mathrm{NSMW}=\frac{\mathrm{ceil}(g,n)\times v}{n} \tag{4}$$

同理,NSMB 可由 NSMW 与 v 相除进行计算. 无论 mod(g,n)是否等于 0,NSMW 都代表了担有最大负载 SM 被分配到的 warp 数目. SM 间通常可相互无依赖的并行工作,因此,NSMW 又可表示设备的 warp 负载,与内核耗时高度正相关.

(4)设备并行空间闲置度

为表示 DPS 的实际利用率以及 warp 竞争 DPS 的激烈程度,提出设备并行空间闲置度(DPS Idle Degree,DPSID)特征辅助评估 kernel 性能,计算方法为:

$$\mathrm{DPSID}=\frac{D}{\mathrm{ceil}(g,n)\times v} \tag{5}$$

DPSID 有效取值范围为[1,32],在该范围内,特征能够对 NSMW 相同的不同 kernel 性能进行“纵向”比较. DPSID 越小,设备理论上能够并行更多 warp 的能力越弱,表现为内核耗时相对更长.

(5)并行效应因子

为便于构建上述特征与目标变量 KDT 的函数关系,将 KDT 与 warp 单位执行时间相除,定义 KDT“体积”(Kernel Duration Volume,KDTV)特征,从而将分析目标转换为 warp 用时的倍数. 基于此,又定义并行效应因子(Parallelism Effect Factor,PEF):

$$\mathrm{PEF}=\frac{k}{t\times W} \tag{6}$$

实验数据表明,随着 W 增大,PEF 数值由大变小,由波动趋于平稳,将数值稳定后的 PEF 称为 PEF 常数(Stable PEF,SPEF),SPEF 的大小间接反映了程序内核优化程度的好坏,数值越小,内核性能越好.

上述特征在 warp 层面上量化了设备及内核属性,描述了 GPU 对 warp 的并行能力,或是映射了基本特征与 KDT 的相关关系,并在形式上达成统一,同时适用于分析 SM 与 GPU 设备性能.

4 性能预测框架

框架以预测现有 CUDA 程序内核以及并行算法 KDT 为目标,适用的分析对象为 warp 负载均衡且恒定型程序. 负载均衡是指全部 warp 对不同计算对象采取相同的指令操作,符合 1 对 1 型并行通信模式[22];恒定指负载与 SOP 以及 block 尺寸无关,否则,可通过固定限制实现恒定. 执行配置不同的内核称为程序用例,由三元组(Dg,Db,NSMW)表示.

4.1 框架结构

依据 NSMW 特征与 KDTV 的相关性关系,框架将在 3 种“情境”下对程序用例性能进行分析及预测,情境划分方法及 KDTV 评估策略如下:

(1)情境 1

当 $g\leqslant A$,即 $W\in[1,\frac{A\times v}{n}]$ 时,假设 warp 间能够完全并行,此时 KDT 近似等于 warp 单位执行时间,于是令 KDTV 估算方法为 $K=1$.

(2)情境 2

当 $g>A$ 且 $I\leqslant 1$ 时,并行空间尚未达到最大占有率,此时,KDTV 与 NSMW、DPSID、NBW 特征等有关. 设该情境下 NSMW 的取值范围为[W_1,W_2],其中 W_1,W_2 分别为满足情境判定条件的边界值. 分别在 NSMW 左右边界处确定 KDTV 下界 $K_1=1$,上界 $K_2=W_2\times S$,则该情境下 KDTV 预测值估算方法为:

$$K=1+\frac{K_2-K_1}{W_2-W_1}\times(W-W_1) \tag{7}$$

预测值估算了内核耗时的平均水平,在此基础上,当 NSMW 相同时,KDTV 可由其与 DPSID、NBW 特征相关性进行修正. 相关性表现为 KDTV 与$\frac{v}{I}$成正比,验证

方法见实验 5.3.

(3)情境 3

当 $W > a$,即 $I < 1$ 时,并行空间被充分利用,PEF 在 SPEF 上下轻微浮动,并行效应达到最佳,由式(6)可推导出该情境下 KDTV 估算方法为:

$$K = W \times S \tag{8}$$

KDTV 此时与 NSMW 呈线性相关关系,增长速率为 SPEF. 实际上,SPEF 与 $I < 1$ 对应的 NSMW 范围存在重叠,对于重叠部分,将其划分为第 2 种情境.

在具体情境下,依据相关策略估计出 KDTV 后,再经公式 $k = K \times t$ 将 KDTV 还原为 KDT.

4.2　特征值评估

对于 CUDA 程序和并行算法两类对象,4.1 节情境判别规则和性能评估策略通用,warp 单位用时及 SPEF 评估方法如下.

(1)warp 单位用时

取 CUDA 程序属于第 1 种情境的程序用例 KDT 平均值作为评估值,估值误差会与 SPEF 计算误差相抵消. 对于并行算法,可以从算法表达的信息中统计对应 CUDA 程序 warp 包含的访存或计算指令类型与规模,再依据式(9)进行估算.

$$t = \sum_{i=1}^{N} N_i \times L_i \tag{9}$$

其中,N 代表指令类型数目,N_i 表示第 i 种类型指令的数目,L_i 代表 i 类型指令的延迟.

(2)SPEF

取 CUDA 程序属于第 3 种情境程序用例的 PEF 平均值作为 SPEF 评估值. 对于并行算法,从性能优化的角度来评估 SPEF. 合理使用全局内存合并访问、避免共享内存存储体访问冲突、减少控制流分支等优化策略的内核 SPEF 更低. 此外,GPU 体系架构、CUDA Core 数量、共享内存、寄存器等硬件资源大小和指令执行机制等因素都将在一定程度上影响 SPEF. 评估并行算法的 SPEF 是一个十分复杂的问题,降低 SPEF 可以作为优化程序性能的综合性目标.

4.3　执行配置参数优化策略

执行配置对内核性能影响较大,但配置选项众多难以择优. 本节依据 NSMW,DPSID 等高层次特征与 KDTV 相关性规律将该问题的优化目标和约束条件表示为式(10). 目标函数意义为对$(W, \frac{v}{I})$二元组按字符串排序规则进行升序排序,约束条件中 P 代表 SOP,优化策略即为比较出使目标函数最小的执行配置.

$$\min \ \mathrm{sort}(W, \frac{v}{I})$$

$$\text{s.t.} \begin{cases} \mathrm{mod}(b, w) = 0 \\ g = \mathrm{ceil}(P, b)/b \\ I = \begin{cases} I, I \geq 1 \\ 1, I < 1 \end{cases} \end{cases} \tag{10}$$

配置参数 Db 取值见表 1,Dg 由约束条件 2 计算. 若 $P = g \times b$,NSMW、NBW、DPSID 特征值分别由式(4),式(1)与式(5)进行计算;若 $P < g \times b$,将存在一个有空载线程的线程块,则需要分两种情况进行讨论:若 $\mathrm{mod}(g, n) \neq 1$,特征值计算与 $P = g \times b$ 情况相同;若 $\mathrm{mod}(g, n) = 1$,NSMW 首先由式(4)计算,再由式(11)修正. 通常来说,当 Dg 为 NSM 的整数倍时,会使目标函数取得较小值,有利于提高程序性能.

$$W = W - v + \mathrm{ceil}(\frac{\mathrm{mod}(P, b)}{w}, 1) \tag{11}$$

5　验证实验与结果分析

拟进行 4 部分实验,首先就 NSMW,SPEF 特征有效性进行验证;其次对不同情境下的程序用例执行时间进行预测,验证框架的通用性与预测准确性;之后与基于机器学习方法的性能预测模型进行对比;最后验证执行配置参数优化策略的有效性. 用于验证实验效果的 CUDA 程序内核详情见表 2.

表 2　CUDA 程序内核列表

内核	功能描述	线程关联对象	类型
VecAddX3	向量加 ×3	1 个向量元素	1
ImgFilter	各向异性扩散滤波	1 个/列图像元素	1/2
Transpose	矩阵转置	1 个矩阵元素	3
MatMul	分块矩阵乘	1 行 +1 列向量	4

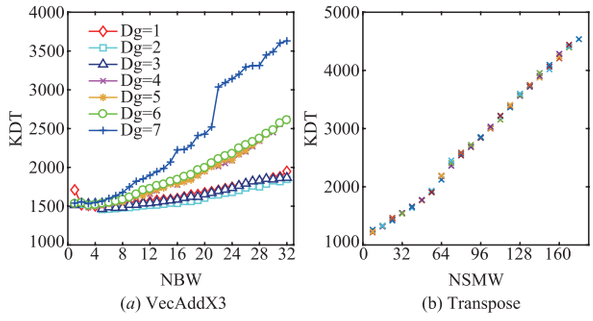
表 2 内核应用了共享内存、合并访问等优化技术,确保实验具有普适性. ImgFilter 对图像进行迭代,迭代次数 i 控制滤波程度,实验通过调整 i 改变 warp 计算负载. 类型 1 和 4 分别表示 warp 负载与 block 大小以及 SOP 均无关或有关;2 和 3 分别表示只与前者或后者有关. 若有关,则固定 block 大小或 SOP 使 warp 负载恒定. 对每个程序用例重复运行 10 次,借助 NVVP[23] 工具收集 KDT 数据并取其均值. 实验平台为 Windows 10 操作系统,设备型号为 GTX 940MX,有 3 个 SM,384 个 CUDA Cores,显存为 4GB,CUDA 版本为 8.0.

5.1　性能特征有效性验证

(1)NSMW 验证

VecAddX3 与 Transpose 内核程序用例相关特征与实际 KDT 关系如图 2 所示,每个点对应 1 个用例.

图 2 子图(a)中 NSMW 相同的程序用例重叠从而使 Dg = {1,2,3} 或 Dg = {4,5,6} 对应的 KDT 曲线基本重合. 由于当 $(p-1) \times n < g \leqslant p \times n, p \in N^+$ 且 Db 相等时



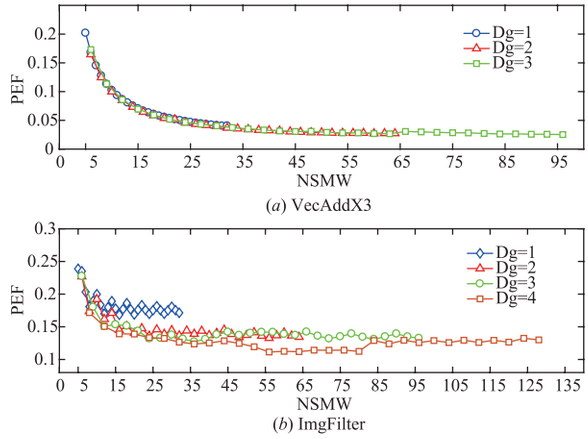
(*a*) VecAddX3 (b) Transpose

图2 NSMW有效性验证

KDT 近似相等,实验对 ceil(g,n)与 Db 相同的程序用例 KDT 取平均,图 3 ~5 标注的 Dg 是经$\frac{\text{ceil}(g,n)}{n}$运算后的结果. 子图($b$)绘制了 64 个程序用例,当 NSMW 相等时,KDT 基本相同,此外,KDT 随 NSMW 增大呈线性增长趋势. 由此得出结论,KDT 主要取决于 NSMW 大小,而与执行配置或线程总数无绝对关系.

(2)PEF 验证

不同内核程序用例 PEF 随 NSMW 增大的变化趋势如图 3 所示. 图 3 验证了当 NSMW 大于某个阈值后,PEF 数值趋于稳定,且该阈值一般小于 W_2,因此,第 3 情境下的 PFF 可视作恒等于 SPEF.



(*a*) VecAddX3

(*b*) ImgFilter

图3 PEF有效性验证

5.2 性能预测效果验证

实验中不同内核程序用例 warp 单位执行时间估值,SPEF 数值以及执行配置信息见表 3.

表 3 程序用例基本信息表

kernel	warp(ns)	SPEF	Dg	Db	samples
VecAddX3	1500	0.0275	[1:1:10]	[32:32:1024]	320
ImgFilter i = 10	70500	0.119	[1:1:15]	[32:32:1024]	480
ImgFilter i = 20	142600	0.115	[1:1:150]	if Dg≤15 Db = [32:32:1024] elseif Dg > 15 Db = {32,64}	770
MatMul	13200	0.04	Dg. x = [1:1:20] Dg. y = [1:1:40]	(16,16)	800
Transpose	1200	0.025	Dg. x = [1:1:64] Dg. y = Dg. x	(16,16)	64
Transpose	1300	0.042	Dg. x = [1:1:32] Dg. y = Dg. x	(32,32)	32

(1)情境 1 验证

VecAddX3 与 ImgFilter i = 10 内核各有 7 个和 8 个程序用例属于情境 1,KDTV 预测误差均在 W = 1 时最大,分别为 0.1,0.06. 因此在评估 warp 执行时间时,不考虑 W = 1 的程序用例.

(2)情境 2 验证

不同内核程序用例的 NSMW,KDTV 真实值与预测基线关系如图 4 所示. 图中横纵坐标分别为 NSMW 与 KDTV,其中 $W_1 \geqslant 5$, $W_2 \leqslant 64$, $K < 5$, NSMW 增长步长与线程块大小有关.

$$\text{acc} = \left(1 - \frac{1}{N} \times \sum_{i=1}^{N} \frac{|r - p|}{r}\right) \times 100\% \quad (12)$$

KDTV 平均预测准确率由式(12)计算结果为 89.32%. 式中 N 为程序用例个数,r,p 分别为 KDTV 真实值与预测值. 图 4 显示当 NSMW 相同时,KDTV 具有离散性,与$\frac{v}{I}$特征成正比. 验证方法为统计 NSMW 相同的程序用例 KDTV 由小到大排名与其 DPSID 倒数,NBW 排名相等的个数与用例总数的比例,统计结果见表 4.

表 4 数据集特征组

kernel	DPSID	NBW
VecAddX3	0.65	0.73
ImgFilter i = 10	0.70	0.91
ImgFilter i = 20	0.76	0.94
AVG	0.70	0.86

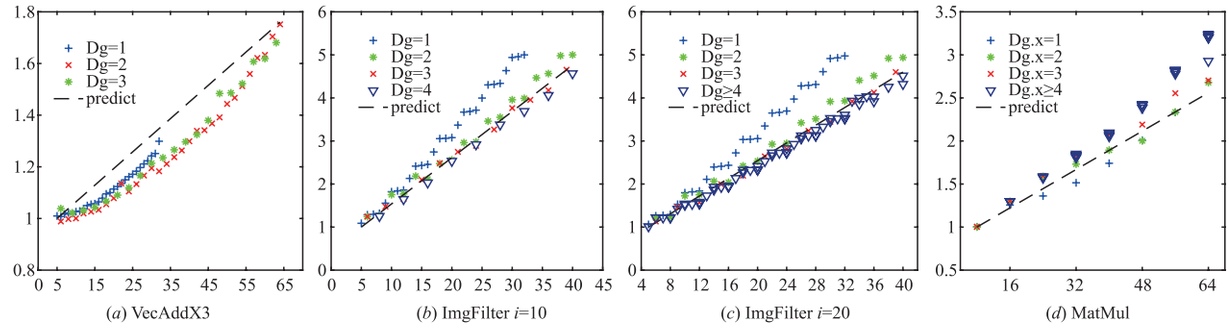


图4　情境2程序用例KDTV预测值与真实值对比图

表 4 数值越大,表示正相关性越强. 对于 MatMul 内核,Db 被固定为(16,16),因此任意程序用例 DPSID 与 NBW 均相同. 实验结果表示,此时内核性能差异受 Dg. x 与 Dg. y 影响,Dg. x 越大,内核性能相对越差.

(3)情境 3 验证

图 5 子图(*a*),(*b*),(*c*)分别对应了表 3 第 2 ~ 4 个内核的实验结果,子图(*d*)合并了 Transpose 内核的实验结果,展示了 KDTV 预测基线与实际值. 横纵坐标同为 NSMW 与 KDTV,两者呈线性相关关系,相关系数为 SPEF. KDTV 平均预测准确率为 94. 73%.

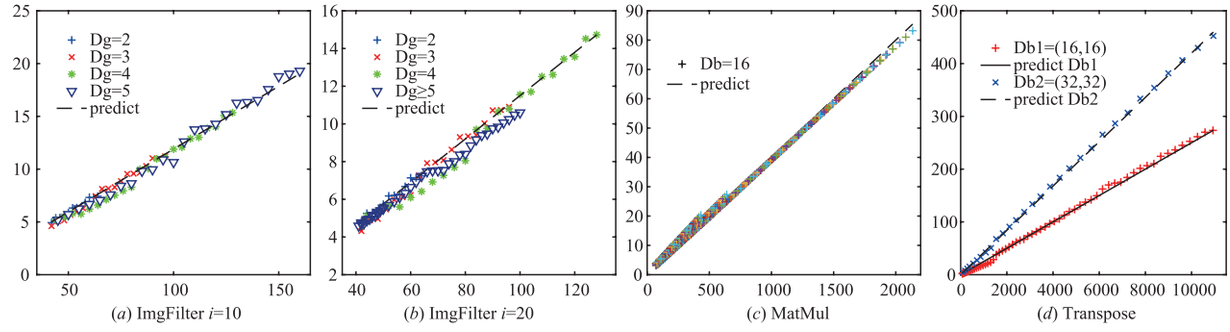


图5　情境3程序用例KDTV预测值与真实值对比图

5.3　对比实验

为使本文框架与基于机器学习方法的性能评估模型具备可比性,采用表 3 中的程序用例集合作为训练集,共计 2466 个数据点. 测试集等同于训练集,按 kernel 分为 4 类. 训练前随机删除 50% 训练集,目的使训练集数据,测试集中训练集以外的数据都尽量多. 数据集采用两组特征进行表示,特征描述见表 4. 性能预测目标为连续型变量 KDTV,因此,采用回归类模型作为性能预测模型,首先对特征值进行归一化处理. 其次,在多项式回归、K 近邻回归、MLP 回归中选择表现最优的多项式回归作为性能预测模型. 对比实验结果如图 6 所示.

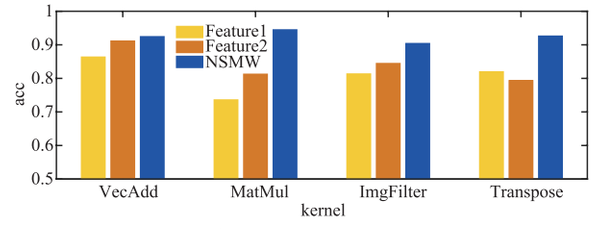


图6　模型间性能预测结果对比

从图中可以得出结论:(1)本文模型优于机器学习模型;(2)高层次特征有效提升了机器学习模型性能;(3)KDTV 离散程度越大,机器学习模型表现越差.

5.4　执行配置参数优化

固定 VecAdd 内核 ***A***,***B*** 向量长度为 1920,操作线程对向量元素分别做 50 次加法、减法及乘法运算. 为内核设置不同的执行配置,配置参数及实验结果见表 5.

表 5　数据集特征组

Feature	Description	ID
Dg,Db,n,c,a,w GPU Max Clock rate, Memory Clock rate	程序固有属性,设备参数等基本特征	1
Throughput of Native Arithmetic Instructions[2]. Capability 3. x Metrics[23]	NVVP metrics,Throughput 等低层次特征	
NSMW,DPSID,NBW,APB,SPEF	与内核性能高度相关的高层次特征	2

表行位置按 sort(W,v/I)结果排序,rank 列为 KDT 名次. 其中 Db = {256,512,1024} 为常用配置参数,但在该例中会导致部分线程空载. 为便于与非空载情况进

行区分和讨论,将它们列于表尾. sort,rank 分别是理论上和实际上的排序结果. 在该问题下,理论上的最优配置为 <<<15,128 >>>,实际耗时排名第 2,且优于 Db = {256,512,1024} 配置,因此常用配置不一定是最优方案. 此外,实验 5.2 从不同的角度在 MatMul,Transpose,ImgFilter 内核上验证了特征与性能的关系,佐证了优化策略的正确性.

表 6 执行配置参数表

$<g,b>$	K	W	I	v	v/I	sort	rank
<15,128>	32022	20	3.2	4	1.25	1	2
<12,160>	32001	20	3	5	1.67	2	1
<6,320>	32107.44	20	3	10	3.33	3	4
<3,640>	32022.11	20	3	20	6.67	4	3
<60,32>	32185.67	20	0.8	1	1	5	6
<30,64>	32221.44	20	1.6	2	1.25	6	7
<20,96>	32164.22	21	2.29	3	1.31	7	5
<10,192>	32438.22	24	2.5	6	2.4	8	9
<5,384>	32377.78	24	2.5	12	4.8	9	8
<4,480>	36246.33	30	2	15	7.5	10	10
<2,960>	36335.22	30	2	30	15	11	11
<8,256>	32090.1	24	-				
<4,512>	32909.4	28					
<2,1024>	36567.4	32					

6 结论

程序性能预测是程序设计过程中的重要环节,它在简化开发流程,优化性能等方面起着重要作用. 本文针对这一问题,定义了高层次特征,搭建了 KDT 性能预测框架. 实验结果验证该框架对程序用例 KDTV 在第 2 种以及第 3 种情境下的平均预测准确率分别达到 89.32% 和 94.73%. 除此之外,执行配置优化策略能有效指导优化内核性能. 特征与框架融合了硬件特征,理论上有能力兼容计算能力不同的硬件设备. 在未来工作,量化 warp 负载,提高 warp 执行时间、SPEF 估值精度,建立 warp 负载不均衡、非恒定内核耗时与 NSMW 的复杂度关系,将是重要的研究方向.

参考文献

[1] 巨涛,朱正东,董小社. 异构众核系统及其编程模型与性能优化技术研究综述[J]. 电子学报,2015,43(1):111-119.
JU Tao, ZHU Zheng-dong, DONG Xiao-she. The feature, programming model and performance optimization strategy of heterogeneous many-core system: a review[J]. Acta Electronica Sinica, 2015, 43(1): 111-119. (in Chinese)

[2] NVIDIA Corporation. CUDA C programming guide[OL]. https://docs.nvidia.com/cuda/archive/8.0/, 2018.

[3] 张珩,张立波,武延军. 基于 Multi-GPU 平台的大规模图数据处理[J]. 计算机研究与发展,2018,55(2):273-288.
ZHANG Heng, ZHANG Li-bo, WU Yan-jun. Large-scale graph processing on multi-GPU platforms[J]. Journal of Computer Research and Development, 2018, 55(2): 273-288. (in Chinese)

[4] ZHU Z, XU S, TANG J, et al. GraphVite: A high-performanceCPU-GPU hybrid system for node embedding[A]. Proceedings of the The World Wide Web Conference[C]. New York: ACM, 2019. 2494-2504.

[5] GONG L, ZHANG C, DUAN L, et al. Nonrigid image registration using spatially region-weighted correlation ratio and GPU-acceleration[J]. IEEE Journal of Biomedical and Health Informatics, 2018, 23(2): 766-778.

[6] CHAPUIS G, EIDENBENZ S, SANTHI N. GPU performance prediction through parallel discrete event simulation and common sense[A]. Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools[C]. Brussels: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. 204-211.

[7] 冯晓,戴紫彬,蔡路亭,等. 基于 Amdahl 定律扩展的多核处理器性能模型研究[J]. 电子学报,2017,45(6):1424-1430.
FENG Xiao, DAI Zi-bin, CAI Lu-ting, et al. Performance model for multicore processor based on extended Amdahl's law[J]. Acta Electronica Sinica, 2017, 45(6): 1424-1430. (in Chinese)

[8] 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):1249-1262.
ZHENG Zhen, ZHAI Ji-dong, LI yan, et al. Workload analysis for typical GPU programs using CUPTI interface[J]. Journal of Computer Research and Development, 2016, 53(6): 1249-1262. (in Chinese)

[9] PARK I K, SINGHAL N, LEE M H, et al. Design and performance evaluation of image processing algorithms on GPUs[J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 22(1): 91-104.

[10] CUI Z, LIANG Y, RUPNOW K, et al. An accurate GPU performance model for effective control flow divergence optimization[A]. Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium[C]. Piscataway: IEEE, 2012. 83-94.

[11] ZHOU K, TAN G, ZHANG X, et al. A performance anal-

ysis framework for exploiting GPU microarchitectural capability[A]. Proceedings of the International Conference on Supercomputing[C]. New York:ACM,2017. 15.

[12] BALDINI I,FINK S J,ALTMAN E. Predicting GPU performance from CPU runs using machine learning[A]. Proceedings of the 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing[C]. Piscataway:IEEE,2014. 254 – 261.

[13] ARDALANI N,LESTOURGEON C,SANKARALINGAM K,et al. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance[A]. Proceedings of the 48th International Symposium on Microarchitecture[C]. New York:ACM,2015. 725 – 737.

[14] O'NEAL K,BRISK P,ABOUSAMRA A,et al. GPU performance estimation using software rasterization and machine learning[J]. ACM Transactions on Embedded Computing Systems (TECS),2017,16(5s):148.

[15] LOUBOUTIN M,LANGE M,HERRMANN F J,et al. Performance prediction of finite-difference solvers for different computer architectures[J]. Computers & Geosciences,2017,105:148 – 157.

[16] LYM S,LEE D,O'CONNOR M,et al. DeLTA:GPU performance model for deep learning applications with in-depth memory system traffic analysis[A]. Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)[C]. Piscataway:IEEE,2019. 293 – 303.

[17] GU J,LIU H,ZHOU Y,et al. DeepProf:performance analysis for deep learning applications via mining GPU execution patterns[J]. arXiv preprint,2017,arXiv:1707. 03750.

[18] O'NEAL K,BRISK P. Predictive modeling for CPU,GPU,and FPGA performance and power consumption:a survey[A]. Proceedings of the 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)[C]. Piscataway:IEEE,2018. 763 – 768.

[19] KONSTANTINIDIS E,COTRONIS Y. A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling[J]. Journal of Parallel and Distributed Computing,2017,107:37 – 56.

[20] 黄品丰,赵荣彩,姚远,等. 面向异构多核处理器的并行代价模型[J]. 计算机应用,2013,33(6):1544 – 1547.
HUANG Pin-feng,ZHAO Rong-cai,YAO Yuan,et al. Parallel cost model for heterogeneous multi-core processors[J]. Journal of Computer Applications,2013,33(6):1544 – 1547. (in Chinese)

[21] NVIDIA Corporation. CUDA occupancy calculato[OL]. http://developer. download. nvidia. com/compute/cuda/CUDA_Occupancy_calculator. xls,2018.

[22] LUEBKE D. CUDA 并行编程入门[OL]. https://cn. udacity. com/course/intro-to-parallel-programming-cs344,2018.

[23] NVIDIA Corporation. NVIDIA visual profiler[OL]. https://developer. nvidia. com/nvidia-visual-profiler,2018.

作者简介

曲海成　男,1981 年 9 月出生,山东烟台人. 副教授、硕士生导师、CCF 会员、IEEE 会员. 2005 年、2008 年和 2016 年分别在青岛理工大学、辽宁工程技术大学和哈尔滨工业大学获学士学位、硕士学位和博士学位. 现为辽宁工程技术大学软件学院软件工程系主任,主要研究方向为遥感影像高性能计算,数字图像处理等.
E-mail:quhaicheng@ lntu. edu. cn

于思淼(通信作者)　男,1994 年 4 月出生,辽宁阜新人. 现为辽宁工程技术大学软件学院硕士研究生. 主要研究方向为 GPU 通用高性能计算,分布式计算,性能优化理论.
E-mail:2607149221@ qq. com

刘万军　男,1959 年 10 月出生,辽宁北镇人. 教授、博士生导师、CCF 高级会员. 1991 年在辽宁工程技术大学获硕士学位,主要从事数字图像处理,运动目标检测与跟踪等研究工作.
Email:liuwanjun@ lntu. edu. cn