软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(8):2168−2184 [doi: 10.13328/j.cnki.jos.005104]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

异构架构下基于放松重用距离的多平台数据布局优化*

刘 颖, 黄 磊, 吕 方, 崔慧敏, 王 蕾, 冯晓兵

(计算机体系结构国家重点实验室(中国科学院 计算技术研究所),北京 100190)

通讯作者: 刘颖, E-mail: liuying2007@ict.ac.cn, http://www.ict.ac.cn

摘 要: 异构架构迅速发展,依靠编译器来挖掘应用程序的数据局部性、充分发挥加速设备片上 cache 的硬件优势,是十分重要的.然而,传统的重用距离在异构背景下面临平台差异性挑战,缺乏统一的计算框架.为了更好地刻画和优化异构程序的局部性,建立了一个多平台统一的重用距离计算机制和数据布局优化框架.该框架根据应用在异构架构下的并行执行方式,从统计平均的角度提出了放松重用距离,并以 OpenCL 程序为例给出了它的计算方法,为多平台数据布局优化决策提供统一的依据.为了验证该方法的有效性,在 Intel Xeon Phi,AMD Opteron CPU,Tilera TileGX-36 这 3 个平台上进行了实验,结果表明,该方法在多平台上可获得至少平均 1.14x 的加速比.

关键词: OpenCL;数据布局;重用距离

中图法分类号: TP316

中文引用格式: 刘颖,黄磊,吕方,崔慧敏,王蕾,冯晓兵.异构架构下基于放松重用距离的多平台数据布局优化.软件学报,2016, 27(8):2168−2184. http://www.jos.org.cn/1000-9825/5104.htm

英文引用格式: Liu Y, Huang L, Lü F, Cui HM, Wang L, Feng XB. Cross-Platform data layout optimization based on relaxed reuse distance on heterogeneous architectures. Ruan Jian Xue Bao/Journal of Software, 2016,27(8):2168−2184 (in Chinese). http:// www.jos.org.cn/1000-9825/5104.htm

Cross-Platform Data Layout Optimization Based on Relaxed Reuse Distance on Heterogeneous Architectures

LIU Ying, HUANG Lei, LÜ Fang, CUI Hui-Min, WANG Lei, FENG Xiao-Bing

(State Key Laboratory of Computer Architecture (Institute of Computing Technology, The Chinese Academy of Sciences), Beijing 100190, China)

Abstract: With the rapid development of heterogeneous system, it's important to enhance data locality and fully utilize on-chip cache via compiler. However, classic reuse distance criteria exhibites platform-sensitive attribute in heterogeneous systems, therefore a unified reused distance calculation framework is needed for compiler to describe and optimize data locality. This paper proposes relaxed reuse distance with a unified calculation method in OpenCL programs as criteria for data layout optimization. Relaxed reuse distance is calculated with heterogeneous execution models and statistical approximation. Experiments are conducted on Intel Xeon Phi, AMD Opteron CPU, and Tilera Tile-GX36, and results show that this optimization can achieve at least 1.23x speedup on average.

Key words: OpenCL; data layout; reuse distance

近年来,主流计算系统越来越多地采用异构架构:通用处理器搭载众核加速设备,共同完成计算任务.一方面,执行在异构系统上的应用,从最初的流式应用逐步扩大到广泛的通用计算领域[1−3],使得应用程序的数据重用显著增加;另一方面,加速设备的微结构,也从最初的不使用片上 cache 向使用 cache 降低访存延迟的方向发

* 基金项目: 国家自然科学基金(61202055, 61402445); 国家高技术研究发展计划(863)(2015AA011505)

Foundation item: National Natural Science Foundation of China (61202055, 61402445); National High-Tech R&D Program of China (863) (2015AA011505)

收稿时间: 2015-08-10; 修改时间: 2016-03-18; 采用时间: 2016-05-23; jos 在线出版时间: 2016-07-30

CNKI 网络优先出版: 2016-08-01 09:38:57, http://www.cnki.net/kcms/detail/11.2560.TP.20160801.0938.006.html

展[4,5].因此,依靠编译器来挖掘应用程序的数据局部性,充分发挥加速设备片上 cache 的硬件优势是十分重要的.

在编译领域,重用距离的大小刻画了程序局部性的的好坏[6],它的计算依赖于程序内存访问序列.在单核单线程应用中,编译器通过静态程序分析技术,确定内存访问序列并计算重用距离,进而以缩短重用距离为目标实施优化,以改善程序局部性[7,8].在多核多线程应用中,多个线程并发执行导致内存访问序列杂乱无序,重用距离难以计算.研究人员尝试在运行时动态采集访存序列[9,10],或者在编译时对访存序列进行预测和近似[11,12],但前者开销巨大、后者误差较大,因此,编译器难以对多核多线程应用实施局部性优化.在异构并行应用中,尽管也同样存在多个线程并发执行导致内存访问序列杂乱无序的问题,但是与多核多线程的并发性具有很大的灵活度和不确定性不同,异构架构下并发执行的线程是按照一定约束进行组织的,编译器可以利用这种约束、同时引入合理假设,以较小的误差和开销计算出应用的重用距离.然而,异构系统中的重用距离面临着平台差异性的挑战:并发线程的组织在不同平台上依照不同的规则和约束,导致同一程序在不同平台上的内存访问序列不同,重用距离不再是一个平台无关的程序特征,而是呈现出平台差异性.因此在异构的背景下,编译器存在以较小的误差和开销计算重用距离的机遇,也同时面临重用距离具有平台差异性的挑战.为了在异构背景下更好地挖掘程序局部性,编译器亟需建立一个多平台统一的重用距离构建框架和数据布局优化机制.

为了解决这个问题,本文针对在多种异构平台上(加速设备包括 GPU/Xeon Phi/CPU 等)得到广泛应用的 OpenCL 程序,引入合理的假设,提出了一种多平台通用的重用距离构建方法,并以此为基础实施了多平台的数据布局优化.该方法为编译器在异构架构下挖掘应用数据局部性、充分发挥加速设备片上 cache 硬件优势提供了重要途径,主要贡献在于:

(1) 在 OpenCL work-group 内部,利用平台特定的执行模型确定内存访问序列,精确计算平台相关的重用距离(称为平台重用距离),弥补了重用距离的平台差异性;

(2) 在 OpenCL work-group 之间,以平台重用距离为基础,辅以对程序并发执行状态的合理假设,近似计算出应用的重用距离(称为放松重用距离),作为实施数据布局优化的依据.

在此基础上,本文建立了多平台统一的数据布局优化方法,并在在多个主流异构平台上进行了实验验证.

本文将在第 1 节对异构架构下的数据布局问题进行深入分析.第 2 节对所提出的基于放松重用距离的数据布局优化方法进行详细描述.第 3 节通过多组实验对该方法的有效性进行验证.第 4 节给出结论.

1 异构架构下的数据布局问题

当前,异构系统搭载的加速设备有相当一部分带有片上高速缓存(cache),典型的例如 Intel Xeon Phi.除此之外,GPU 也呈现出使用 cache 的趋势:AMD GCN 架构 GPU 已开始使用 L1 cache,NVIDIA Fermi/Kepler 架构的 GPU 也可以选择将片上存储配置为 L1 cache.应用在带有 cache 的加速设备上执行,如何进行数据布局以提高 cache 利用率、进而提升性能,是一个关键问题.本节将对这个问题进行深入分析.

1.1 单程序多数据(SPMD)编程模型中计算与数据的关系

在异构系统中运行的程序,绝大多数采用单程序多数据(single program multiple data,简称 SPMD)编程模型,典型的例如 CUDA[13]/OpenCL[14].分析 SPMD 程序中计算任务与数据之间的对应关系,对实施数据布局优化十分关键.下面以 OpenCL 程序为例进行分析.

一个 OpenCL 程序包括运行在通用处理器上的 host 代码和运行在加速设备上的 kernel 代码,其中:host 代码通过 OpenCL API 实现对 kernel 代码的控制,完成数据传输、kernel 编译等辅助工作;Kernel 是执行在加速设备上的主体,由众多轻量级线程构成,这些线程使用相同的代码(kernel 代码)操作不同的数据,一个 OpenCL 线程被称为 work-item.OpenCL 工作空间(NDRange)由一个 kernel 的所有 work-item 构成,通常被进一步划分成数个 work-group:每个 work-group 包含数目相同的相邻 work-item,并保证它们在加速设备上的同一个计算单元(compute unit)中同时执行.图 1 给出了一个二维 OpenCL 工作空间示意图.

在 OpenCL 中,work-item 操作的数据称为内存对象,根据一个 work-item 操作内存对象的元素数目,我们可以将 work-item 与内存对象之间的关系分为 3 类.

① 一对多型:每个 work-item 访问内存对象的多个元素.OpenCL 程序通常处理较大的数据量,即,内存对象的元素数目远远大于工作空间中 work-item 的总数,使得一个 work-item 需要处理多个元素.这是因为:一方面,从应用的角度,某些计算的基本单位需要由多个元素联合构成,例如一个 work-item 处理矩阵的一行、一列、一块等;另一方面,从加速设备的角度,过多的 work-item 将带来巨大的调度开销.因此当数据量很大时,一个 work-item 处理多个元素,是降低运行时开销的首选方案.

② 一对一型:每个 work-item 访问内存对象的一个元素.在数据量不大时,是最直观的计算方法.

③ 多对一或多对多型:多个 work-item 访问内存对象的同一个或同几个元素.OpenCL 通过这种方式进行 work-item 之间的数据共享.

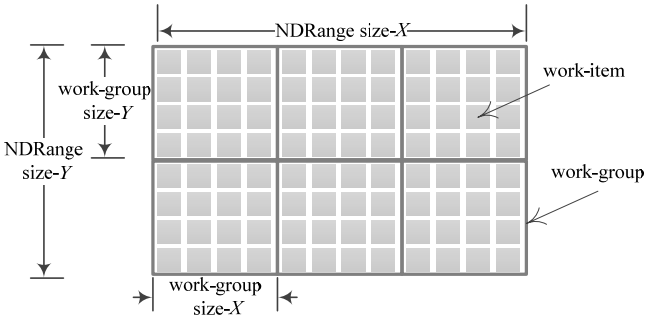


Fig.1 OpenCL work space

图 1 OpenCL 工作空间

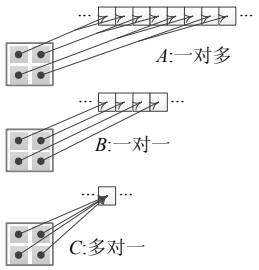
在 OpenCL 应用中,work-item 与内存对象之间的关系以一对多型和一对一型为主,图 2 给出了这 3 种类型的一个示例:假设一个 OpenCL 程序的 kernel 代码如图 2(a)所示,每个 work-item 访问 *A*,*B*,*C* 这 3 个内存对象,那么 work-item 与 *A*,*B*,*C* 的关系即分别为一对多型、一对一型和多对一型,如图 2(b)所示.

```
Kernel:

global=get_global_id();
global=get_group_id();

A[2*global]=B[global];
A[2*global+1]=C[group];
```

(a) OpenCL kernel 代码



(b) Work-Item 与内存对象关系

Fig.2 OpenCL work space and memory object

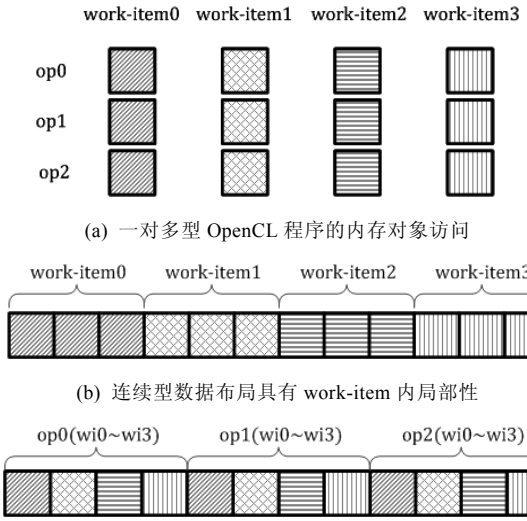
图 2 OpenCL 工作空间与内存对象

1.2 数据布局与局部性

在 OpenCL 数据布局方面,一对多型的情况普遍存在,是数据布局优化的重要应用场景.如前所述,在一对多型 OpenCL 程序中,每个 work-item 访问内存对象的多个元素,如图 3(a)所示.这些被访问的元素在内存中以何种顺序排布,才能构造出有利于提高访存效率的数据局部性,是一对多型 OpenCL 程序数据布局优化的核心问题.

直观的,在一对多型 OpenCL 程序中,数据可以按照 work-item 方向和 op 方向(op 指编译器生成的伪操作(operation),包含一个操作符(operator)和多个操作数(operands))两种方式进行排布,如图 3(b)和图 3(c)所示.在图 3(b)中,一个 work-item 处理的多个元素在内存中连续存放,即程序执行时每个 work-item 依次访问连续的内存

空间,数据局部性存在于 work-item 内部.我们将这种数据布局方式定义为连续型.在图 3(c)中,相邻 work-item 同时处理的元素在内存中连续存放,即相邻 work-item 同时访问连续的内存空间,数据局部性存在于 work-item 之间.我们将这种数据布局方式定义为合并型.



(a) 一对多型 OpenCL 程序的内存对象访问

(b) 连续型数据布局具有 work-item 内局部性

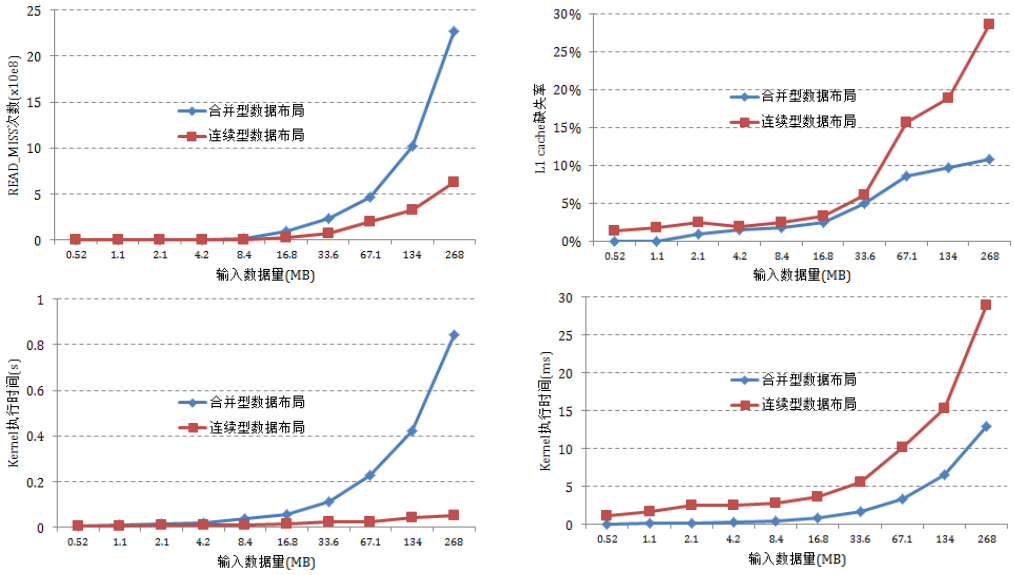
(c) 合并型数据布局具有 work-item 间局部性

Fig.3 OpenCL data layout and locality

图 3 OpenCL 数据布局与局部性

1.3 局部性对cache行为及性能的影响

OpenCL 程序的两种数据布局方式具有不同的数据局部性,将对 cache 行为产生不同的影响,进而影响程序的执行效率.图 4 给出了一个归约求和 OpenCL 程序的两种布局方式对 cache 行为和执行效率的影响.



(a) Tilera TileGX-36 平台

(b) Intel Xeon Phi 平台

Fig.4 Impact on cache behavior and performance of data layout of an OpenCL reduction program

图 4 归约求和 OpenCL 程序数据布局对 cache 行为及性能的影响

在图 4(a)中,该程序执行在 Tilera TileGX-36 平台上,连续型数据布局具有较低的 READ_MISS 和较短的执行时间;在图 4(b)中,该程序执行在 Intel Xeon Phi 平台上,合并型数据布局具有较低的 L1 cache 缺失率和较短的执行时间.

从图 4 中,我们可以看出:

(1) 数据布局方式对性能具有显著影响:在 TileGX-36 和 Xeon Phi 两个平台上,连续型数据布局方式与合并型数据布局方式均表现出几倍到几十倍的性能差异.

(2) 数据布局方式对性能的影响具有平台差异性:在 TileGX-36 平台上,合并型数据布局具有更低的 cache miss 和更短的执行时间;而在 Xeon Phi 平台上,连续型数据布局则具有更低的 cache miss 和更短的执行时间.

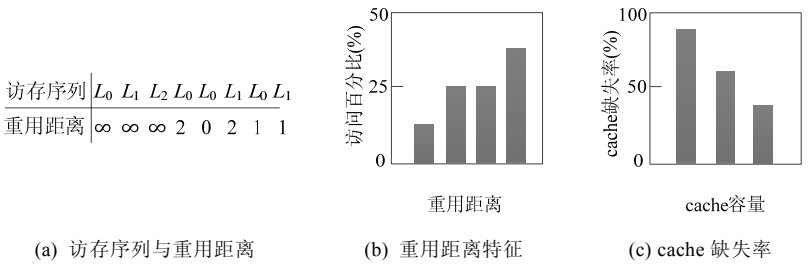
上述现象说明:不同的数据布局方式,在不同平台上具有显著不同的 cache 行为;采用一种固定的数据布局方式(连续型或合并型),无法在多个平台上都获得令人满意的执行效率.因此,在带有 cache 的加速设备上执行一对多型 OpenCL 程序时,编译器需要进行多平台数据布局优化决策,在连续型和合并型二者之间,选取在该平台上 cache 行为更好的数据布局方式,以获得更高的执行效率.

2 基于放松重用距离的多平台数据布局优化

第 1 节对数据布局问题的分析表明:在 OpenCL 程序中,不同的数据布局方式将构建出不同的数据局部性,从而对 cache 行为和程序性能产生不同影响;同时,局部性对性能的影响具有显著的平台差异性.实施 OpenCL 程序的数据布局优化,关键在于量化比较不同局部性在不同平台上的优劣,并据此选择合适的数据布局方式.在编译领域,重用距离是衡量程序局部性的重要标准,本文在传统重用距离的基础上对其进行合理扩展和放松,使之可以作为量化 OpenCL 程序不同局部性在不同平台上优劣的标准.下面将分别介绍编译计算放松重用距离和实施数据布局优化决策的方法.

2.1 放松重用距离

在编译领域,重用距离是 cache 行为分析的重要依据,它被定义为某数据被再次访问时,距其上次被访问之间,程序访问的存储位置不同的数据总量.根据重用距离,编译器可以预测 cache 缺失:重用距离超过 cache 容量的数据访问,将导致 cache 缺失,如图 5 所示.



(a) 访存序列与重用距离 (b) 重用距离特征 (c) cache 缺失率

Fig.5 Reuse distance and cache miss (calculated in cache line)

图 5 重用距离与 cache 缺失(以 cache line 为单位)

重用距离的构建基础是程序访存序列(如图 5(a)所示).在单线程应用中,编译器可以通过静态程序分析获取访存序列;在多线程应用中,程序的并发执行使得访存序列难以静态获取,通常需要在程序执行时动态收集并记录访存序列,具有极大的运行时开销;在异构并行应用中,尽管程序高度并发执行,例如 OpenCL 程序通常有成百上千个 work-item 同时执行,但是与多线程应用的高度灵活并发执行相比,这些线程(work-item)以固定的规则

(work-space)进行组织,在执行时符合相关约束.因此,我们在这些规则和约束的基础上,辅以合理的假设,即可在静态时近似的计算出应用的重用距离.同时,OpenCL 线程组织服从的规则和约束,具有平台差异性,在这种背景下,重用距离不再仅仅是一个程序自身的属性,而是体现出平台相关性.本文提出了一种放松重用距离的静态计算方法,为多种异构平台提供了统一的局部性刻画与优化框架,其关键点包括:

① Work-Item 交叉(work-item interleave):根据 OpenCL 执行模型,同一 work-group 内部的所有 work-item 必须映射到加速设备上的同一个执行部件上执行.实际的执行平台将依据加速设备的硬件结构特征,保证这些 work-item 按照确定的次序完成.因此在 work-group 内部,work-item 以确定但平台相关的次序交叉执行.相应地,重用距离的计算具有平台差异性,但是可以精确获取.本文提出了一个多平台统一的 work-group 内部重用距离计算框架,并称其为平台重用距离.

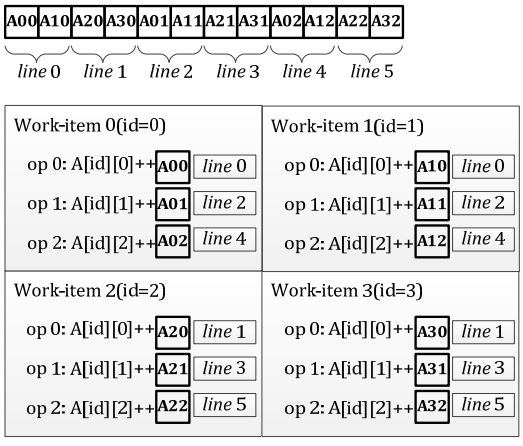
② Work-Group 交叉(work-group interleave):根据 OpenCL 执行模型,不同 work-group 的执行是相互独立的,既没有固定的次序,也不保证与加速设备的执行部件具有一定的映射关系.实际的执行平台,也多以 work-group 为基本单位实施静态或动态的调度.因此在不同 work-group 之间,work-item 以不固定且不可静态预测的次序交叉执行.在这样的背景下,我们在平台重用距离的基础上引入合理假设,从统计平均的角度对重用距离的计算方法进行适当放松,即放松重用距离,用于辅助数据布局优化决策.

下面将分别对 work-item 交叉和 work-group 交叉这两个关键点进行详细描述.

2.2 Work-item交叉

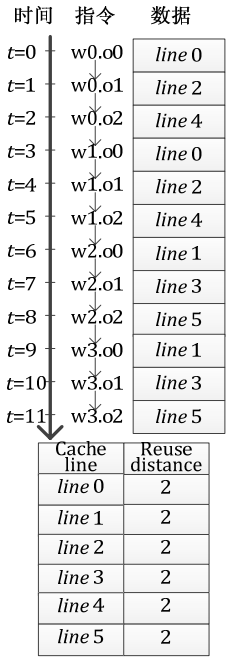
在 OpenCL 中,每个 work-item 都是顺序执行的普通单线程程序,因此,work-group 内部数据的交叉访问顺序由 work-item 的并行执行方式决定.如前所述,work-item 的并行执行方式是平台相关的,是影响平台重用距离的关键因素.目前,各平台上的典型执行方式为向量型或迭代型两种.

图 6 说明了重用距离的平台相关性,即 work-item 的向量型和迭代型执行方式对重用距离产生的影响.



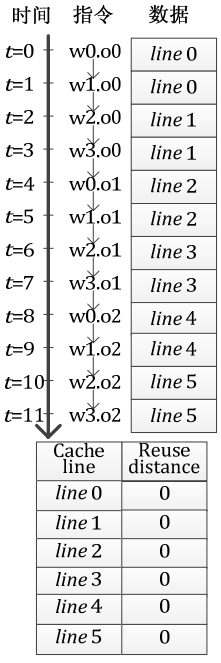
OpenCL 程序的一个 work-group 包含 4 个 work-item
每个 work-item 分别访问内存对象 *A* 的 3 个元素
A 的每两个元素占据一个 cache line

(a) OpenCL kernel 源程序及数据访问



Work-Item 依次执行
访问 cache line 0-2-4-0…
每 line 重用 1 次,距离 2

(b) 迭代型执行



work-item 轮流执行
访问 cache line 0-0-1-1…
每 line 重用 1 次,距离 0

(c) 向量型执行

Fig.6 Work-Item interleave

图 6 Work-Item 交叉

从图中可以看出,同一个 OpenCL 程序(图 6(a)),在 work-item 并行执行方式不同的平台上(图 6(b)和图 6(c))具有不同的重用距离.

重用距离的平台相关性,来源于 work-group 内部 work-item 的执行顺序存在平台差异.

- 典型的向量型执行模型,例如 NVIDIA GPU 中,work-group 中相邻的 32 个 work-item 组成一个 warp,它们以硬件保证的 SIMD 方式执行;例如 Intel Xeon Phi 上,work-group 中相邻的 16 个 work-item 被自动向量化,使用向量部件计算.
- 典型的迭代型执行模型,例如 AMD CPU 上,work-group 中所有 work-item 被打包为循环的不同迭代,依次顺序执行.

总之,不同异构平台具有各自特定的 work-item 交叉执行次序,我们用函数(即 work-item interleave)来描述,如公式(1)所示.

$$order=WII^{p}(tid,op) \tag{1}$$

公式(1)表示,在平台 p 上,线程号为 tid 的 work-item 中的第 op 条伪操作,在经过 work-item 交叉后,在 work-group 中是第 $order$ 条执行的伪操作.其中 tid=0,1,2,…$work\_group\_size$,op=0,1,2,…num_of_op,$order$=0,1,2,….

对于平台 p,WII^{p} 在编译时刻是固定的,其形式由系统开发人员确定.例如,图 6(b)的平台上,$WII^{6b}(tid,op)=num\_of\_op*tid+op$;图 6(c)的平台上,$WII^{6c}(tid,op)=tid+work\_group\_size*op$;在 NVIDIA GPU 平台上,$WII^{nvidia}(tid,op)=\left[\frac{tid}{32}\right]+op$.进一步地,在平台 p 上,work-group 中第 $order$ 条执行的伪操作,其开始执行时刻为

$$t^{p}=\sum_{i=0}^{i<order} lat_{i}^{p} \tag{2}$$

其中,lat_{i}^{p} 代表平台上第 i 条伪操作的静态执行时间(以 cycle 计),在编译时刻是固定的.

以公式(1)、公式(2)给出的 work-item 交叉公式为依据,我们可以计算任意平台 p 的平台重用距离:

1) 计算 work-item 私有访存序列(work-item private trace).

对 kernel 代码进行静态分析,获取每条伪操作访问数据的存储位置,形成 $TRACE_{wi}(op,data)$序列(见表 1),其中,$line\_op_{k}^{tid}$ 表示线程编号为 tid 的 work-item 中第 k 个 op 访问的数据所占据的 cache line.

Table 1 Work-Item private trace $TRACE_{wi}(op,data)$

表 1 Work-Item 私有访存序列 $TRACE_{wi}(op,data)$

op	0	1	…	k	…	op_num
$data$	$line\_op_{0}^{tid}$	$line\_op_{1}^{tid}$	…	$line\_op_{k}^{tid}$	…	$line\_op_{op\_num}^{tid}$

2) 进行 work-item 交叉(work-item interleave).

根据平台 p 的 work-item 交叉公式 $order=WII^{p}(tid,op)$和 $t^{p}=\sum_{i=0}^{i<order} lat_{i}^{p}$,将 work-item 私有访存序列 $TRACE_{wi}(op,data)$变换为 work-group 私有访存序列(work-group private trace)$TRACE_{wg}(time,data)$.即对于 $TRACE_{wi}(op,data)$中的每一项(k,$line\_op_{k}^{tid}$),若它在 work-group 中的开始执行时刻为第 n 个 cycle ($WII^{p}(tid,k)=m$ 且 $\sum_{k=0}^{k<m} lat_{k}^{p}=n$),则 $TRACE_{wg}(time,data)$中对应的项为(n,$line\_t_{n}$),其中,$line\_t_{n}=line\_op_{k}^{tid}$, 见表 2.

Table 2 Work-Group private trace $TRACE_{wg}(time,data)$

表 2 Work-Group 私有访存序列 $TRACE_{wg}(time,data)$

Time	0	1	…	n	…
Data	$line\_t_{0}$	$line\_t_{1}$	…	$line\_t_{n}$	…

3) 提取平台重用特征(platform reuse signature).

平台重用特征包括平台重用距离和平台重用时间,其中,平台重用时间是指该 work-group 在两次数据使用之间所执行的 cycle 数目.它们的作用范围是 work-group 内部,记为集合 SIG_{wg}(*rd_platform*,*rt_platform*),表示对于 work-group 内发生的每一个数据重用,其平台重用距离为 *rd_platform*,对应的平台重用时间为 *rt_platform*. SIG_{wg}(*rd_private*,*rt_private*)的构建方法是:按顺序(即,*time* 由小到大)对 work-group 私有访存序列 $TRACE_{wg}$(*time*, *data*)进行扫描,若第 *i* 个 cycle 访问的数据在第 *j* 个 cycle(*j*>*i*)发生重用,即 $line\_t_i=line\_t_j$,则平台重用距离 *rd_platform* 等于集合 $\{line_{t_i},line_{t_{i+1}},...,line_{t_j}\}$ 中不重合(即存储位置不同)的 line 的数目,平台重用时间 *rt_platform* 等于(*j*−*i*).

2.3 Work-group交叉

在第 2.2 节中,我们介绍了 work-group 内重用距离的计算方法,它具有平台相关性,称为平台重用距离.本节,我们将在平台重用距离的基础上讨论其他 work-group 对其产生的影响.

OpenCL 程序在加速设备上执行时,若加速设备上的某些计算部件共享 cache,那么执行在这些计算部件上的 work-group 将对彼此的平台重用距离产生影响,如图 7 所示.假设在某平台上,一个 work-group 的数据访问为 cache *line* 0-*line* 0-*line* 1-*line* 1-*line* 2-*line* 2…,那么 cache *line* 0,*line* 1,*line* 2 的平台重用距离均为 0.但是,在来自该 work-group 的两次数据访问之间,并行执行的其他 work-group 也会发生数据访问,如图 7 中斜线部分所示,导致数据的重用距离拉长.

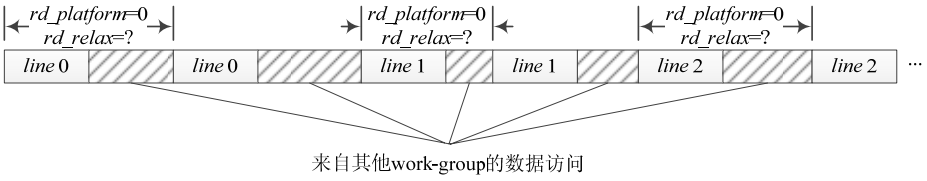


Fig.7 Work-Group interleave

图 7 Work-Group 交叉

进行 work-group 交叉的关键是,获取两次数据重用之间其他 work-group 访问的存储位置不同的数据总量,即访存脚印(footprint).尽管 work-group 的并行执行使访存脚印具有随机性而无法精确计算,但是 OpenCL 程序中所有 work-group 代码相同、work-group 执行时占用的计算单元也相同的属性,促使我们可以对 work-group 的交叉情形进行一定放松:将其近似为均匀交叉,进而从统计平均的角度提出放松重用距离.下面将对均匀交叉的理论依据加以说明.

对于一个 work-group,我们用函数 *FP*(即 footprint)来描述其访问的存储地址不同的数据总量随时间的变化趋势,如公式(3)所示.

$$data\_volume=\begin{cases}FP(t), & 0\leqslant t<T\\ 0, & t<0 \text{ or } t\geqslant T\end{cases} \tag{3}$$

其中,*t* 表示以该 work-group 开始执行时刻为参照的执行时刻(以 cycle 计),*T* 为 work-group 的静态执行时间.特别的,我们将以程序开始执行为参照的执行时刻定义为绝对时刻,将以 work-group 开始执行时刻为参照的时刻定义为相对时刻.显然,相对时刻的取值范围是[0,*T*).*data_volume* 表示在相对时刻 *t*,该 work-group 正在执行的指令所访问的存储地址不同的数据总量(以 cache line 计).显然,在 OpenCL 程序中,所有 work-group 的函数 *FP* 相同.

根据公式(3),若 *work_group_id*=*k* 的 work-group 在相对时刻 t_i^k 和 t_j^k (对应的绝对时刻为 t_i 和 t_j)访问了同一 cache line,那么在时间区间[t_i,t_j]内,其他所有 work-group 形成的访存脚印为

$$footprint_{all}=\sum_{m=0}^{m\neq k,m<wg\_num}\sum_{n=i}^{n<j}FP(t_n^m) \tag{4}$$

其中, $t_n^m(n=i,i+1,i-j-1)$ 表示 work-group m(m=0,1,2,…,wg_num 且 $m\neq k$)落在时间区间[t_i,t_j]内的相对时刻.若 work-group$m$ 在时刻 t_i 前已执行完毕,则 $t_i^m=t_j^m=-1$;或在时刻 t_j 后才开始执行,则 $t_i^m=t_j^m=T$,如图 8 所示.

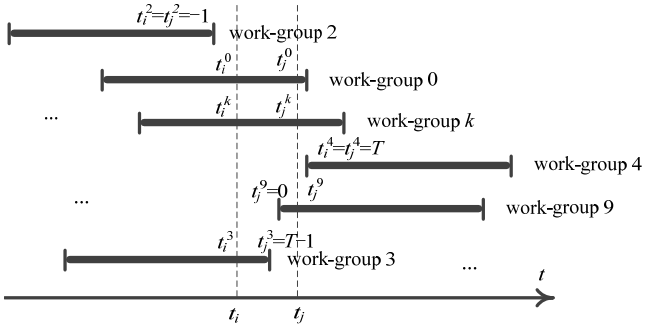


Fig.8 Illustration of Active work-groups in time region [t_i,t_j]

图 8 时间区间[t_i,t_j]内活跃 work-group 示意图

在公式(4)中,那些FP值不为零的元素来自在时间区间[t_i,t_j]内活跃(一直处于或曾经处于执行状态)的 work-group.考虑到加速设备中所有计算单元(compute unit,简称 CU)均在程序正常执行时被占用,任意时刻并发执行的 work-group 数目均等于 CU 总数,我们提出活跃 work-group 放松条件.

放松 1. 某时间区间内活跃的 work-group 等效于 ***CU_num*** 个 work-group 在此期间一直处于执行状态.

其中,CU_num 表示加速设备中与 $work\_group\_id=k$ 所在 CU 共享 cache 的 CU 总数.基于放松 1,公式(4)在经过加法交换后,可以进一步变化为

$$footprint_{all}=\sum_{n=i}^{n<j}\left(\sum_{m=0}^{m\neq k,m<wg\_num}FP(t_n^m)\right)\cong\sum_{n=i}^{n<j}\left(\sum_{\tilde{m}}FP(t_n^{\tilde{m}})\right) \tag{5}$$

其中, $\tilde{m}$ 为等效活跃 work-group 的 id,共计 CU_num-1 个(扣除作为参照的 $work\_group\_id=k$ 的 1 个 work-group). $t_n^{\tilde{m}}$ 表示绝对时刻 t_n 在 work-group $\tilde{m}$ 中的相对时刻,所有等效 work-group 的 $t_n^{\tilde{m}}$ 构成了绝对时刻 t_n 的等效相对时刻集合.

在公式(5)中, $\left(\sum_{\tilde{m}}FP(t_n^{\tilde{m}})\right)$ 表示 FP 在绝对时刻 t_n 的等效相对时刻集合上的函数值之和,它可以改写为

$$\sum_{\tilde{m}}FP(t_n^{\tilde{m}})=(CU\_num-1)*\overline{FP_{CU}(t_n)} \tag{6}$$

其中, $\overline{FP_{CU}(t_n)}$ 表示 FP 在绝对时刻 t_n 的等效相对时刻集合上的函数均值.将公式(6)代入公式(5),则 work-group 的访存脚印为

$$footprint_{all}\cong(CU\_num-1)*\sum_{n=i}^{n<j}\overline{FP_{CU}(t_n)} \tag{7}$$

显然, $\overline{FP_{CU}(t_n)}$ 与时刻有关.但是从统计平均的角度讲,在程序正常执行过程中的任意时刻,所有等效 work-group 的执行进度都是相互独立的,可以认为等效相对时刻集合趋于均匀的分布在[0,T)内.基于此,我们对 $\overline{FP_{CU}(t_n)}$ 的计算条件进行放松如下.

放松 2. FP 在任意绝对时刻 t_n 的等效相对时刻集合上的函数均值 $\overline{\boldsymbol{FP}_{CU}(\boldsymbol{t}_n)}$ 均相等,记为 $\overline{\boldsymbol{FP}_{CU}(\boldsymbol{t})}$.

放松 3. FP 在等效相对时刻集合上的函数均值 $\overline{\boldsymbol{FP}_{CU}(\boldsymbol{t})}$,等于在时间区间[0,***T***)内的函数均值 $\overline{\boldsymbol{FP}_{cycle}(\boldsymbol{t})}$.

基于放松 2,公式(7)可以进一步变化为

$$footprint_{all} \cong (CU\_num-1)*\sum_{n=i}^{n<j}\overline{FP_{CU}(t_n)} = (CU\_num-1)*(j-i)*\overline{FP_{CU}(t)} \tag{8}$$

基于放松 3,我们可以得到:

$$\overline{FP_{CU}(t_n)} \cong \overline{FP_{cycle}(t)} = \frac{\sum_{t=0}^{t<T} FP(t)}{T} = \frac{data\_volum\_total}{T} \tag{9}$$

其中,*data_volum_total* 为一个 work-group 访问的存储位置不同的数据总量.将公式(9)带入公式(8),则有:

$$footprint_{all} \cong (j-i)*\frac{(CU\_num-1)*data\_volum\_total}{T} \tag{10}$$

公式(10)中,(*j*–*i*)为第 2.2 节中得到的 work-group*k* 中访问同一 cache line 的平台重用时间 *rt_platform*; $\frac{(CU\_num-1)*data\_volum\_total}{T}$ 则是一个固定值,其含义是(*CU_num*–1)个 work-group 平均每 cycle 产生的存储位置不同的数据访问总量,即:work-group 产生访存脚印的速度是均匀的,work-group 之间均匀交叉.

在上述 3 个放松条件下,我们即可以计算程序的放松重用距离集合 *SIG*(*rd_relax*):对第 2.2 节 work-item 交叉得到的平台重用特征序列 SIG_{wg}(*rd_platform*,*rt_platform*)中的每一项(*rd_platform*,*rt_platform*),对应的放松重用距离 *rd_relax* 可以用公式(11)进行计算:

$$rd\_relax = rd\_platform + rt\_platform*\frac{(CU\_num-1)*data\_volum\_total}{T} \tag{11}$$

2.4 数据布局优化决策

编译器在进行数据布局优化决策时,以放松重用距离集合 *SIG*(*rd_relax*)为依据.若需要优化的内存对象对应的放松重用距离为集合 *SIG*(*rd_relax*)的子集 SIG_{opt}(*rd_relax*),那么这些内存对象的平均放松重用距离为

$$\overline{rd\_relax} = \frac{1}{n}\sum_{i=0}^{n-1} rd\_relax_i \tag{12}$$

其中,*n* 为集合 SIG_{opt}(*rd_relax*)包含的元素总数;rd_relax_i 表示该集合的第 *i* 个元素,*i*=0,1,2,…,*n*–1.

若应用在连续型数据布局方式和合并型数据布局方式下,需要优化的内存对象的平均放松重用距离分别为 $\overline{rd\_relax_{continuous}}$ 和 $\overline{rd\_relax_{coalescing}}$,那么决策公式可以表示为

$$数据布局 = \begin{cases} 连续型, \text{ if } \overline{rd\_relax_{continuous}} \leqslant \overline{rd\_relax_{coalescing}} \\ 合并型, \text{ if } \overline{rd\_relax_{continuous}} > \overline{rd\_relax_{coalescing}} \end{cases} \tag{13}$$

综上所述,图 9 给出了基于放松重用距离的多平台数据布局编译优化框图,虚线框中是本文提出的编译器.该编译器首先依据第 2.2 节、第 2.3 节介绍的 work-item 交叉和 work-group 交叉方法计算 OpenCL 程序在不同平台上的放松重用距离;再依据公式(12)、公式(13)介绍的方法为每个平台选择合并型或连续型数据布局方式,并完成相关的代码变换.优化后的代码,经过各异构平台上本地编译器的代码生成模块生成可执行代码.

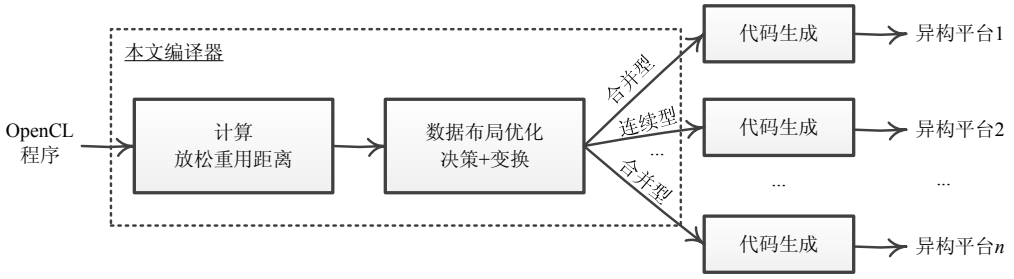


Fig.9 Compilation framework of cross-platform data layout optimization based on relaxed reuse distance

图 9 基于放松重用距离的多平台数据布局编译优化框图

3 实验与分析

为了验证本文提出的基于放松重用距离的多平台数据布局优化方法的有效性,我们在 Clang 上实现了本文的编译方法,同时使用来自多个测试集的 OpenCL 用例在 Intel Xeon Phi 众核协处理器、AMD Opteron 多核处理器和 Tilera TileGX-36 众核处理器这 3 个平台上进行了实验.

3.1 实验平台与测试用例

本文使用的实验平台包括:

① Intel Xeon Phi 众核协处理器,57 个物理计算核心,每核 4 个硬件线程,私有 L1 cache 32KB,共享 L2 cache 512KB,cache line 大小均为 64B.该处理器通过 PCIE 搭载在 Intel Xeon E5-2670 处理器上,Xeon CPU 作为 OpenCL host,Xeon Phi 执行 OpenCL kernel.用作代码生成的本地编译器使用 Intel OpenCL SDK,work-item 交叉函数为 $order = WII^{Phi}(tid, op) = \left\lfloor \frac{tid}{16} \right\rfloor + op$; 性能分析和统计工具为 Intel Vtune,监控 L1 事件.

② AMD Opteron 多核处理器,8 个计算核心,私有 L1 cache 32KB,共享 L2 cache 512KB,cache line 大小均为 64B.该处理器的一个计算核心作为 OpenCL host,所有计算核心执行 OpenCL kernel.用作代码生成的本地编译器使用 AMD APP SDK,work-item 交叉函数为 $order=WII^{Opteron}(tid,op)=num\_of\_op\times tid+op$;性能分析和统计工具使用 Oprofile,监控 L2 事件.

③ Tilera TileGX-36 众核处理器,36 个计算核心,私有 L1 cache 8KB,共享 L2 cache 64KB,cache line 大小分别为 16B 和 64B.该处理器的一个计算核心作为 OpenCL host,所有计算核心执行 OpenCL kernel.用作代码生成的本地编译器使用 SNU-SAMSUNG OpenCL Framework,work-item 交叉函数为 $order=WII^{TileGX}(tid,op)=num\_of\_op\times tid+op$;性能分析和统计工具使用 Perf,监控 L2 事件.

需要说明的是:尽管 GPU 也属于带有 cache 的加速设备,但 GPU 的片外访存合并策略(memory coalescing)对 OpenCL 程序性能的影响远大于 cache 行为对性能的影响,因此放松重用距离不是数据布局优化决策的决定性因素.

本文使用的 OpenCL 测试用例和输入集,见表 3.

Table 3 OpenCL testing cases and input sets used in experiments

表 3 实验使用的 OpenCL 测试用例及输入集

测试用例名称	所属测试集	功能描述	输入集
kmeans	Rodinia[15]	数据挖掘	819200 数据点×34 特征
lbm	Parboil[16]	流体动力学	120×120×150 立方体
md	SHOC[17]	分子动力学	12288 元素×128 邻居
RecursiveGaussian	AMD-APP-SDK[18]	高斯滤波器	512×512 彩色图像
oclSimpleMultiGPU	NVIDIA-GPU-Computing-SDK[13]	归约求和	67 108 864 元素
oclBlackScholes	NVIDIA-GPU-Computing-SDK	期权定价	4 000 000 选项
oclDotProduct	NVIDIA-GPU-Computing-SDK	向量点积	1277944×1277944 元素
nw	Rodinia	动态规划	2048×2048×4 数据点
reduction	SHOC	归约求和	8192×8192 元素
MatrixTranspose	AMD-APP-SDK	矩阵转置	1024×1024 元素

上述 OpenCL 测试用例中,kernel 使用的内存对象见表 4,实验将对与 work-item 存在一对多关系的内存对象进行数据布局优化.

Table 4 OpenCL memory objects used in experiments

表 4 实验使用的 OpenCL 内存对象

work-item 与内存对象对应关系		一对一	一对多		多对一/多对多
kmeans	内存对象名称	membership	feature		clusters
	内存对象大小	3.125MB	106.25MB		0.664KB
RecursiveGaussian	内存对象名称	–	input	output	–

	内存对象大小		4MB	4MB	
md	内存对象名称 内存对象大小	force 192KB	neighList 768MB		position 192KB
oclSimpleMultiGPU	内存对象名称 内存对象大小	d_Result 64KB	d_Input 256MB		–

Table 4 OpenCL memory objects used in experiments (Contiuned)

表 4 实验使用的 OpenCL 内存对象(续)

work-item 与内存对象对应关系		一对一	一对多					多对一/多对多
oclBlackScholes	内存对象名称	–	d_Call	d_Put	d_S	d_X	d_T	–
	内存对象大小		15.3MB	15.3MB	15.3MB	15.3MB	15.3MB	
lbm	内存对象名称	–	dstGrid					srcGrid
	内存对象大小		164.8MB					164.8MB
oclDotProduct	内存对象名称	c	a		b			–
	内存对象大小	4.9MB	19.5MB		19.5MB			
nw	内存对象名称	–	reference_d		input_itemsets_d			–
	内存对象大小		16MB		16MB			
reduction	内存对象名称	–	g_idata					g_odata
	内存对象大小		256MB					256B
MatrixTranspose	内存对象名称	–	input		output			–
	内存对象大小		4MB		4MB			

3.2 实验结果与分析

在实验中,我们对每个 OpenCL 测试用例,分别计算表 4 标明的一对多内存对象,在每个平台上的平均放松重用距离 $\overline{rd\_relax}$,表 5 给出了计算结果.

Table 5 Results of relaxed reuse distance computation of three platforms

表 5 3 个平台放松重用距离计算结果

测试用例	Intel Xeon Phi		AMD Opteron CPU		Tilera TileGX-36	
	合并型 $\overline{rd\_relax}$	连续型 $\overline{rd\_relax}$	合并型 $\overline{rd\_relax}$	连续型 $\overline{rd\_relax}$	合并型 $\overline{rd\_relax}$	连续型 $\overline{rd\_relax}$
kmeans	117.6	22.9	47.5	1.8	95.5	4.4
RecursiveGaussian	128.3	374.1	1 408	52.6	4 992	189.4
md	0.05	48.5	704	5.5	2 496	19.5
oclSimpleMultiGPU	0.06	15.2	5 887	0.4	13 055	2.2
oclBlackScholes	0.01	79.9	466.2	4.2	1 034.9	14.9
lbm	0.02	15.2	55.9	2.9	125.5	9.2
oclDotProduct	0.08	7.6	11.5	1.4	25.5	2.8
nw	0.006	15.1	46	0.01	102	0.07
reduction	0.002	15	368	0.006	816	0.03
MatrixTranspose	0.03	0.3	10.5	1.2	24.5	4.9

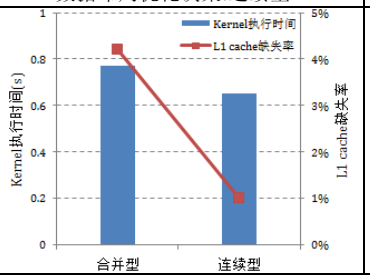
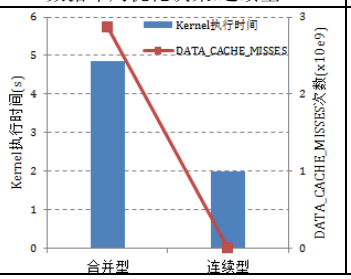
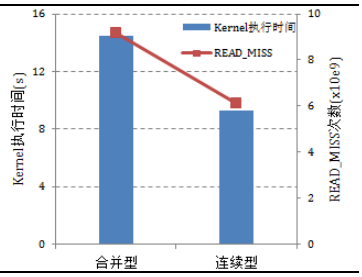
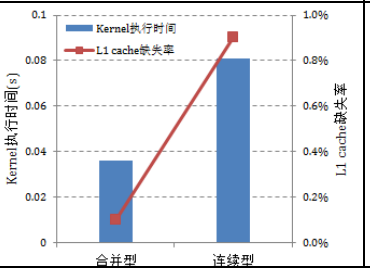
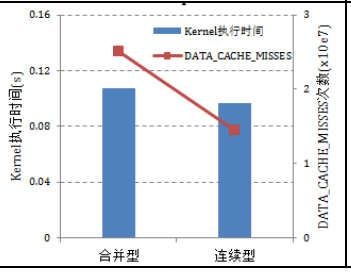
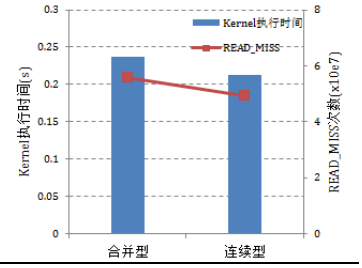
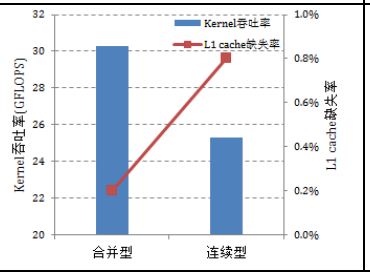
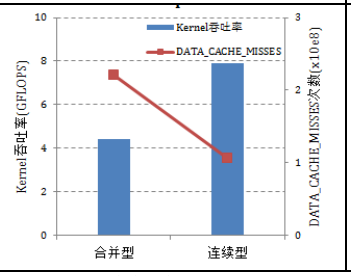
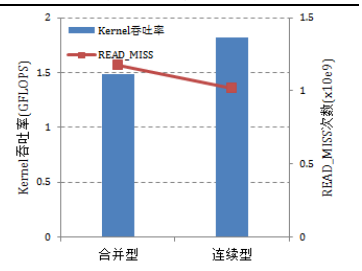
从表 5 可以看出,放松重用距离体现出显著的平台相关性:同一应用在不同平台上的重用距离大小不同、趋势也不同

在获得了不同平台上的放松重用距离之后,编译器依据公式(13)进行数据布局优化决策,表 6 给出了各应用在 3 个平台上的决策结果.为了验证上述优化的效果,表 6 同时给出了合并型和连续型数据布局在不同平台上的 cache 行为(图中方块所示)和性能表现(图中柱状表示).

从中可以看出,本文提出的基于放松重用距离的数据布局优化决策方法与应用的 cache 行为和性能表现吻合.即:本文所提出的放松重用距离可以很好地刻画程序的局部性,作为编译器的优化依据.同时,实验表明:Intel Xeon Phi 作为典型的向量型执行平台,多数程序采用合并型数据布局方式将获得更好的性能(也有特例,例如 kmeans);而 AMD Opteron CPU 和 Tilera TileGX-36 作为典型的迭代型执行平台,多数程序采用连续型数据布局方式的性能更好.

Table 6 Results of compiler optimization decision, cache behavior and performance on 3 platforms

表 6 3 个平台放松优化决策与性能、cache 行为实验结果

<table>
<tr><td colspan="3">kmeans</td></tr>
<tr><td>Intel Xeon Phi</td><td>AMD Opteron CPU</td><td>Tilera TileGX-36</td></tr>
<tr><td>数据布局优化决策:连续型</td><td>数据布局优化决策:连续型</td><td>数据布局优化决策:连续型</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">RecursiveGaussian</td></tr>
<tr><td>Intel Xeon Phi</td><td>AMD Opteron CPU</td><td>Tilera TileGX-36</td></tr>
<tr><td>数据布局优化决策:合并型</td><td>数据布局优化决策:连续型</td><td>数据布局优化决策:连续型</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">md</td></tr>
<tr><td>Intel Xeon Phi</td><td>AMD Opteron CPU</td><td>Tilera TileGX-36</td></tr>
<tr><td>数据布局优化决策:合并型</td><td>数据布局优化决策:连续型</td><td>数据布局优化决策:连续型</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">oclSimpleGPU</td></tr>
<tr><td>Intel Xeon Phi</td><td>Intel Xeon Phi</td><td>Intel Xeon Phi</td></tr>
<tr><td>数据布局优化决策:合并型</td><td>数据布局优化决策:连续型</td><td>数据布局优化决策:连续型</td></tr>
</table>

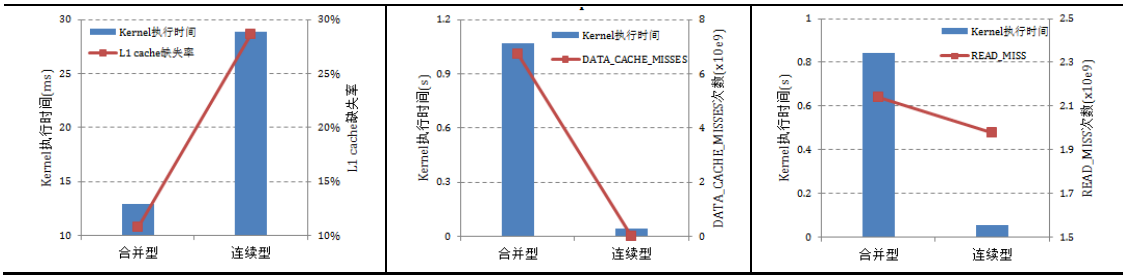
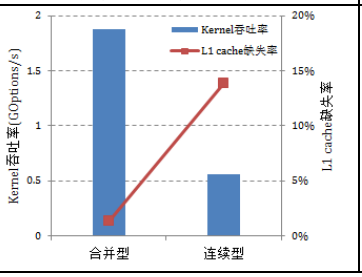
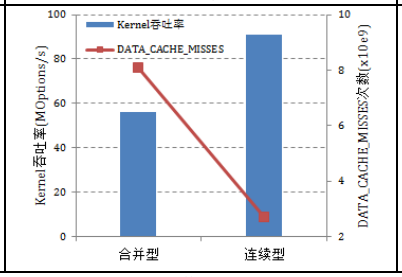
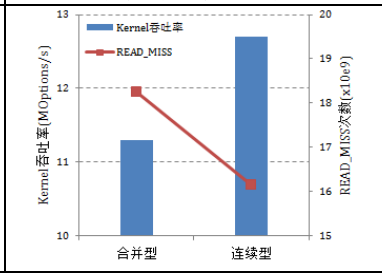
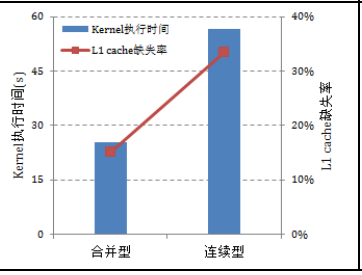
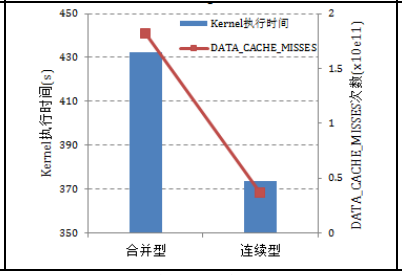
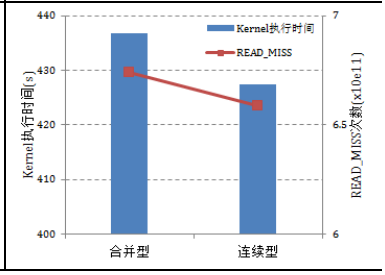
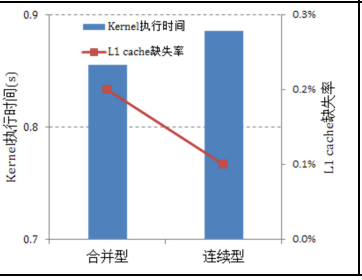
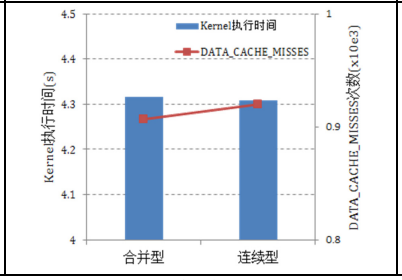
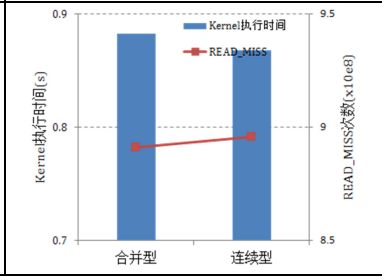


Table 6 Results of compiler optimization decision, cache behavior and performance on 3 platforms (Contiuned)

表 6 3 个平台放松优化决策与性能、cache 行为实验结果(续)

oclBlackScholes		
Intel Xeon Phi	AMD Opteron CPU	Tilera TileGX-36
数据布局优化决策:合并型	数据布局优化决策:连续型	数据布局优化决策:连续型
		
lbm		
Intel Xeon Phi	AMD Opteron CPU	Tilera TileGX-36
数据布局优化决策:合并型	数据布局优化决策:连续型	数据布局优化决策:连续型
		
oclDotProduct		
Intel Xeon Phi	AMD Opteron CPU	Tilera TileGX-36
数据布局优化决策:合并型	数据布局优化决策:连续型	数据布局优化决策:连续型
		
reduction		
Intel Xeon Phi	AMD Opteron CPU	Tilera TileGX-36

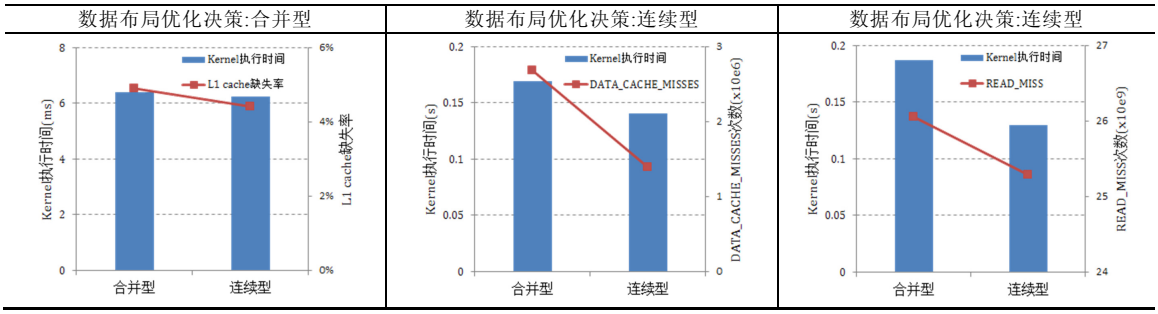
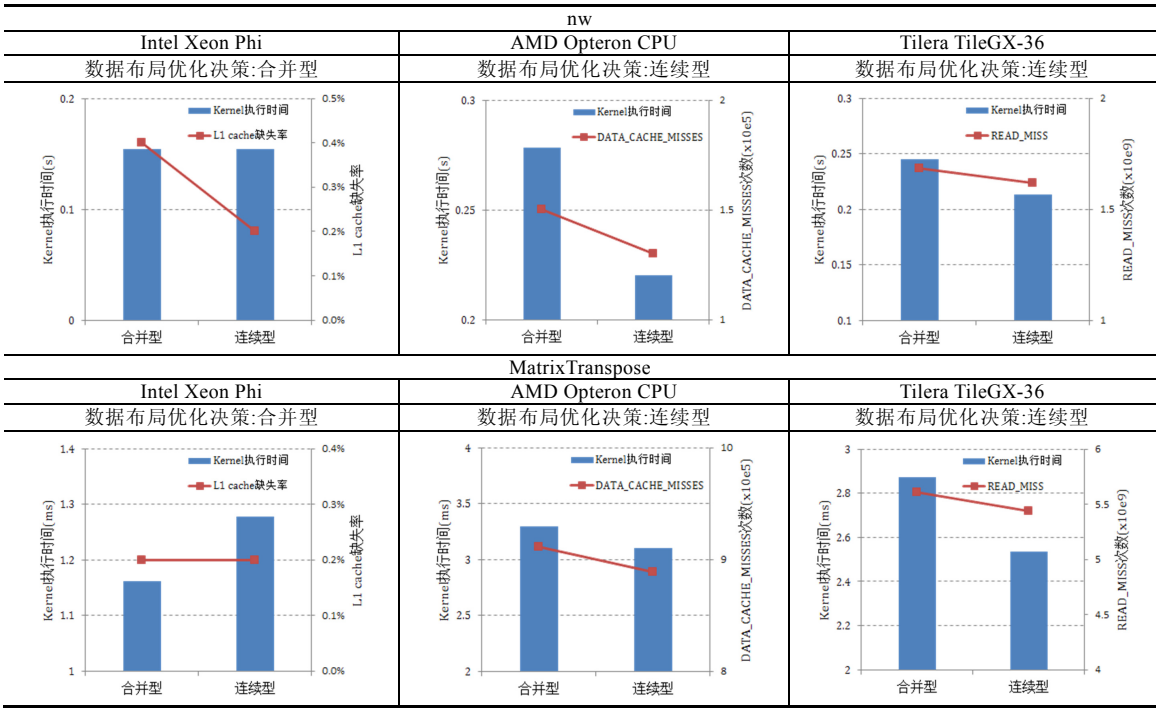


Table 6 Results of compiler optimization decision, cache behavior and performance on 3 platforms (Contiuned)

表 6 3 个平台放松优化决策与性能、cache 行为实验结果(续)



为进一步说明本文提出的优化方法的有效性,图 10 给出了不使用本文方法和使用本文方法的性能对比.不使用本文方法是指:对于每个测试用例,在所有平台上均采用某种固定的(合并型或连续型)数据布局方式.使用本文方法是指:对于每个测试用例,分别在每个平台上依据放松重用距离,选取合理的数据布局方式.从图 10 中可以看出:与在所有平台上都采用合并型数据布局方式相比,本文方法可获得平均 20.3%的性能提升;与在所有平台上都采用连续型数据布局方式相比,本文方法可获得平均 14.1%的性能提升.

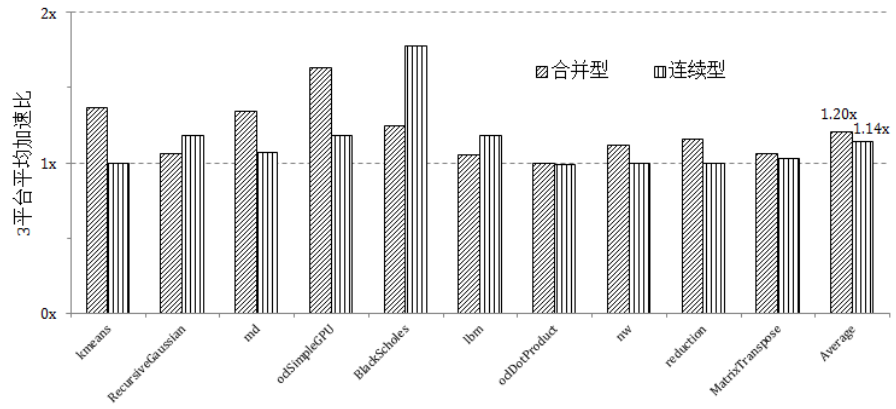


Fig.10 Average speedup on 3 platforms

图 10 多平台平均加速比

4 结束语

本文提出了一种异构架构下基于放松重用距离的多平台数据布局优化方法,其贡献在于克服了重用距离在异构系统中的平台差异性,为多平台提供了统一的局部性刻画与优化框架.该框架可用于解决在带有 cache 的加速设备上执行一对多型 OpenCL 程序时的数据布局问题:合并型和连续型两种数据布局方式,分别在 work-item 之间和 work-item 之内构建了数据局部性,成为影响 cache 行为和程序性能的重要因素.本文方法通过计算不同数据布局方式的放松重用距离,辅助局部性优化决策,其核心技术在于:

(1) work-item 交叉:将平台相关的 OpenCL 执行模型抽象为 work-item 交叉函数,以确定 work-group 内部所有 work-item 的所有 op 的交叉执行顺序,进而确定 work-group 内部的数据访问顺序,计算重用距离;这个重用距离是与平台密切相关的,因此称为平台重用距离.

(2) work-group 交叉:为解决来自不同 work-group 的 work-item 之间执行顺序无法确定、从而无法精确估计本 work-group 内部平台重用距离将遭受来自其他 work-group 何种影响的问题,提出根据 OpenCL 程序 work-group 之间对等的的特性,从统计平均的角度对重用距离的计算方法进行放松,称为放松重用距离.

实验验证了上述方法的有效性,在 Intel Xeon Phi 众核协处理器、AMD Opteron CPU 多核处理器、Tilera Tile-GX36 众核处理器这 3 个不同平台上,与在 3 个平台均使用合并型数据布局或连续型数据布局相比,本文方法分别获得了平均 20.3%和平均 14.1%的性能提升.

References:

[1] Datta D, Mehta S, Shalivahan, Srivastava R. CUDA based particle swarm optimization for geophysical inversion. In: Proc. of the 1st Int'l Conf. on Recent Advances in Information Technology (RAIT). 2012. 416−420. [doi: 10.1109/RAIT.2012.6194456]

[2] Fasih A, Hartley T. GPU-Accelerated synthetic aperture radar back projection in CUDA. In: Proc. of the 2010 IEEE Radar Conf. 2011. 1408−1413. [doi: 10.1109/RADAR.2010.5494395]

[3] Pavlović D, Vaser R, Korpar M, Šikić M. Protein database search optimization based on CUDA and MPI. In: Proc. of the 36th Int'l Convention on Information and Communication Technology Electronics and Microelectronics (MIPRO). 2013. 1278−1280.

[4] Kepler. http://www.nvidia.com/object/nvidia-kepler.html

[5] AMD GCN microarchitecture—Whitepaper. http://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf

[6] Beyls K, D'Hollander E. Reuse distance as a metric for cache behavior. In: Proc. of the Conf. on Parallel and Distributed Computing and Systems. 2001. 617−662.

[7] Ding C, Zhong Y. Predicting whole-program locality through reuse distance analysis. In: Proc. of the 24th Programming Language Design and Implementation (PLDI). 2003. 245−257. [doi: 10.1145/781131.781159]

[8] Xue J, Vera X. Efficient and accurate analytical modeling of whole-program data cache behavior. IEEE Trans. on Computers, 2004, 53(5):547−566. [doi: 10.1109/TC.2004.1275296]

[9] Ding C, Chilimbi T. A composable model for analyzing locality of multi-threaded programs. Technical Report, MSR-TR-2009-107, Microsoft Research, 2009.

[10] Schuff DL, Kulkarni M, Pai VS. Accelerating multicore reuse distance analysis with sampling and parallelization. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2010. [doi: 10.1145/1854273.1854286]

[11] Chandra D, Guo F, Kim S, Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. In: Proc. of the Int'l Symp. on High-Performance Computer Architecture. 2005. 340−351. [doi: 10.1109/HPCA.2005.27]

[12] Suh GE, Devadas S, Rudolph L. Analytical cache models with applications to cache partitioning. In: Proc. of the Int'l Conf. on Supercomputing (ICS). 2001. 1−12. [doi: 10.1145/377792.377797]

[13] NVIDIA CUDA toolkit. 2013. https://developer.nvidia.com/cuda-downloads

[14] KHRONOS Group. The open standard for parallel programming of heterogeneous systems. 2013. http://www.khronos.org/opencl

[15] Rodinia: Accelerating compute-intensive applications with accelerators. 2009. http://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators

[16] Parboil benchmark suite. http://impact.crhc.illinois.edu/parboil/parboil.aspx

[17] The scalable HeterOgeneous computing (SHOC) benchmark suite. 2012. https://github.com/vetter/shoc/wiki

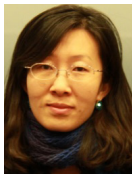
[18] AMD APP SDK. 2013. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/download-archive/

刘颖(1982－),女,辽宁大连人,助理研究员,主要研究领域为异构并行编程,编译系统及相关工具.

崔慧敏(1979－),女,博士,副研究员,CCF专业会员,主要研究领域为并行编译,并行编程.

黄磊(1980－),女,博士,工程师,主要研究领域为编译优化.

王蕾(1976－),女,博士,助理研究员,主要研究领域为并行计算,编译系统和相关工具.



吕方(1975－),女,博士,助理研究员,主要研究领域为性能分析,编译系统及相关工具.

冯晓兵(1969－),男,博士,研究员,博士生导师,CCF杰出会员,主要研究领域为先进编译技术及相关工具环境.