计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150114
53(6):1173-1185,2016

基于2阶段同步的GPGPU线程块压缩调度方法

张 军[1,3] 何炎祥[1,2] 沈凡凡[1] 江 南[1,4] 李清安[1,2]
[1](武汉大学计算机学院 武汉 430072)
[2](软件工程国家重点实验室(武汉大学) 武汉 430072)
[3](东华理工大学软件学院 南昌 330013)
[4](湖北工业大学计算机学院 武汉 430068)
(zhangjun_whu@whu.edu.cn)

Two-Stage Synchronization Based Thread Block Compaction Scheduling Method of GPGPU

Zhang Jun[1,3], He Yanxiang[1,2], Shen Fanfan[1], Jiang Nan[1,4], and Li Qing'an[1,2]
[1](*Computer School*, *Wuhan University*, *Wuhan* 430072)
[2](*State Key Laboratory of Software Engineering* (*Wuhan University*), *Wuhan* 430072)
[3](*School of Software*, *East China University of Technology*, *Nanchang* 330013)
[4](*School of Computer Science*, *Hubei University of Technology*, *Wuhan* 430068)

Abstract The application of general purpose graphics processing unit (GPGPU) has become increasingly extensive in the general purpose computing fields facing high performance computing and high throughput. The powerful computing capability of GPGPU comes from single instruction multiple data (SIMD) execution model it takes. Currently, it has become the main stream for GPGPU to implement the efficient execution of the computing tasks via massive high parallel threads. However the parallel computing capability is affected during dealing with the branch divergent control flow as different branch path is processed sequentially. In this paper, we propose TSTBC (two-stage synchronization based thread block compaction scheduling) method based on analyzing the previously proposed thread block compaction scheduling methods in inefficient dealing with divergent branches. This method analyzes the effectiveness of thread block compaction and reconstruction via taking the use of the adequacy decision logic of thread block compaction and decreases the number of inefficient thread block compaction. The simulation experiment results show that the effectiveness of thread block compaction and reconstruction is improved to some extent relative to the other same type of methods, and the destruction on data locality inside the thread group and the on-chip level-one data cache miss rate can be reduced effectively. The performance of the whole system is increased by 19.27% over the baseline architecture.

Key words general purpose graphics processing unit (GPGPU); thread scheduling; thread block compaction and reconstruction; two-stage synchronization; branch divergence

---

收稿日期:2015-02-09;**修回日期**:2015-07-06
基金项目:国家自然科学基金重点项目(91118003);国家自然科学基金项目(61373039,61170022,61462004);江西省教育厅科技项目(GJJ150605)
This work was supported by the Key Program of the National Natural Science Foundation of China (91118003), the National Natural Science Foundation of China (61373039,61170022,61462004), and the Science and Technology Project of Jiangxi Province Education Department (GJJ150605).
通信作者:何炎祥(yxhe@whu.edu.cn)

摘 要 通用图形处理器(general purpose graphics processing unit, GPGPU)在面向高性能计算、高吞吐量的通用计算领域的应用日益广泛,它采用的 SIMD(single instruction multiple data)执行模式使其能获得强大的并行计算能力.目前主流的通用图形处理器均通过大量高度并行的线程完成计算任务的高效执行.但是在处理条件分支转移的控制流中,由于通用图形处理器采用串行的方式顺序处理不同的分支路径,使得其并行计算能力受到影响.在分析讨论前人针对分支转移处理低效的线程块压缩重组调度方法的基础上,提出了 2 阶段同步的线程块压缩重组调度方法 TSTBC(two-stage synchronization based thread block compaction scheduling),通过线程块压缩重组适合性判断逻辑部件,分 2 个阶段对线程块进行压缩重组有效性分析,进一步减少了无效的线程块压缩重组次数.模拟实验结果表明:该方法较好地提高了线程块的压缩重组有效性,相对于其他同类方法降低了对线程组内部数据局部性的破坏,并使得片上一级数据 cache 的访问失效率得到有效降低;相对于基准体系结构,系统性能提升了 19.27%.

关键词 通用图形处理器;线程调度;线程块压缩重组;2 阶段同步;分支转移

中图法分类号 TP302

近年来,通用图形处理器(general purpose graphics processing unit, GPGPU)以其强大的并行计算能力在高性能计算领域得到了广泛的应用,并随着其处理不规则应用程序能力的日益提升,GPGPU 在通用计算领域的应用也越来越广泛.GPGPU 通过采用单指令多数据(single instruction multiple data, SIMD)执行模式获得强大的并行处理能力和高计算吞吐量[1].SIMD 执行模式将单条程序指令映射到多个数据通道上同时执行.NVIDIA 将这种执行模式又称为单指令多线程(single instruction multiple thread, SIMT)执行模式[1-2],并将映射到不同数据通道的指令流看作逻辑线程.SIMT 执行模式对线程实行分组管理,每个分组包含多个线程,其大小(称为线程组宽度)一般为 32(NVIDIA 系列显卡)或 64(AMD 系列显卡),这些宽度的分组分别被称为 warp[3-4] 和 wavefront[5].多个相互协作的线程分组又构成了线程块(thread block, TB).

SIMT 执行模式将计算任务映射到不同的数据通道同时执行,以获得高的线程级并行度(thread level parallelism, TLP).但是当遇到不规则计算任务时,例如分支转移(branch divergence)指令,其执行任务的 TLP 会降低.主要是由于当出现分支转移指令时,同一个线程组中的不同线程会选择不同的分支路径执行,而不同分支路径以串行的方式执行.另外,由于线程组采用锁步的方式执行[3],执行完的线程需要等待其他分支路径上的线程完成执行.因此,系统性能会受到一定的影响.

为了有效降低由于分支转移引起的性能下降,当前有一类方法将线程块中执行相同分支路径的线程组合在一起执行,以提升分支转移情况下的线程级并行度,从而提升 SIMD 通道资源的利用率.Fung 等人[6]提出了线程块压缩重组调度策略 TBC(thread block compaction).然而,该方法在进行线程块压缩重组时,会出现属于不同 warp 的线程对应相同数据通道的情形,使得重组之后的 warp 数量并没有减少,总的通道利用率和线程级并行度并未得到提升,这种情况被视为无效的线程重组.针对这种情况.Rhu 等人[7]提出了基于压缩适合性预测的线程块压缩重组调度策略 CAPRI(compaction-adequacy predictor),实现了基于压缩重组历史的预测机制,有效减少了不合理的线程块压缩重组.但是该方法采用首次保守预测和 TB 范围内一次分支单次预测的静态预测方法,在一定程度上降低了预测的准确度.另外,CAPRI 方法在分析线程块重组适合性时,并未考虑线程块压缩重组产生的性能收益(performance gains, PG)和性能开销(performance overhead, PO)之间的关系,只有当 PG 大于 PO 时,线程块的压缩重组才是有益的.

基于对 TBC 和 CAPRI 两种策略的分析,本文提出了一种基于 2 阶段同步的线程块压缩重组调度方法 TSTBC(two-stage synchronization based thread block compaction scheduling).当遇到分支转移时,经过 2 个阶段的分析来判断当前线程块是否适合进行压缩重组.相对于 CAPRI 方法,压缩重组的有效性得到了较好的提升.

本文的主要贡献包括:分析了 CAPRI 方法对线程块压缩适合性判断方法的不足和局限性;提出了 2 阶段同步的线程块压缩重组调度方法和线程块局部压缩重组思想,进一步提高了线程块压缩重组

适合性判断的准确度,并减少了因同步线程块中的 warp 而产生的等待开销;在分析线程块压缩重组适合性时,考虑了线程块压缩重组带来的性能收益和开销之间的关系,进一步提升了线程块压缩重组的有效性;实验结果表明,相对于 CAPRI,TSTBC 在系统平均性能方面提升了 9.4%.

本文首先介绍了基准 GPGPU 体系结构和相关概念以及重汇聚栈的分支转移控制机制;其次分析了 CAPRI 方法的核心思想及其不足;然后重点分析阐述了 TSTBC 方法的算法设计和具体实现细节,并对实验方法和实验结果进行了分析;最后对相关工作进行了分析,并对全文进行了总结.

1　背　　景

1.1　基准 GPGPU 体系结构

本文讨论的基准 GPGPU 体系结构主要参考 NVIDIA Fermi 系列和文献[8],如图 1 所示:

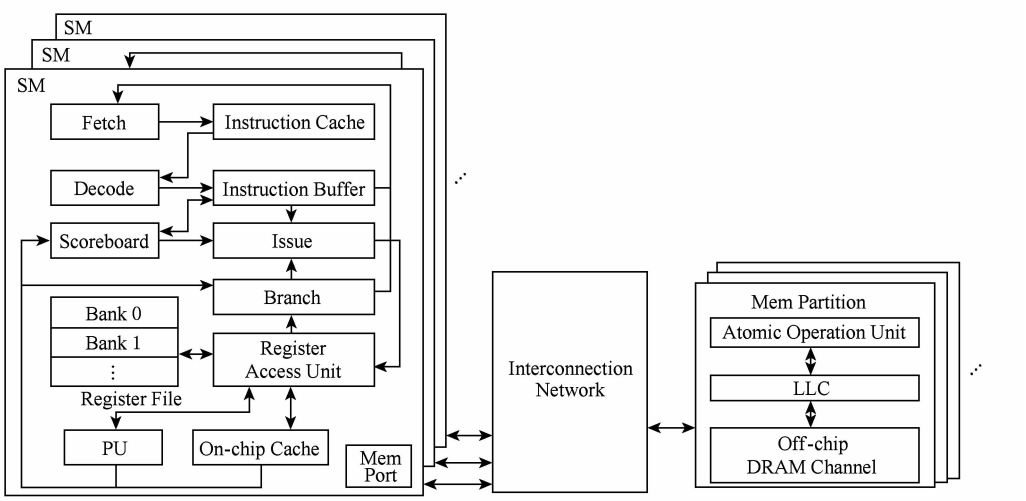


Fig. 1　Baseline architecture of GPGPU.

图 1　基准 GPGPU 体系结构

GPGPU 由多个流式多处理器(stream multiprocessor, SM)组成,每个 SM 称之为核.为了有效控制逻辑控制单元的数量,通常将多个 SM 组织成为一个图形处理核组(graphics processing cluster, GPC).每个 SM 包含了众多的处理单元(processing unit, PU),每个 PU 负责最终的指令执行.

SM 的整个处理流水线可以分为前端和后端.前端主要包括取指部件、译码部件、记分板、线程调度器及分支处理单元等部件,其余部件均属于后端.

取指部件负责根据每个线程组对应的 PC 值从片外存储将指令取到指令 cache 中.指令 cache 中的指令则依次送入到指令译码部件,经过译码后的指令被送入到指令 buffer 中.指令 buffer 为每个线程组缓冲译码后的指令,便于指令快速执行,并为每个线程组分配了一定数量的专用指令槽.记分板部件用来记录每个线程组当前执行指令目的寄存器的使用情况.指令发射器负责按一定策略选取准备就绪的指令发射.

经过指令发射器发射的指令到达 SIMD 流水线后端执行通道后,需要通过寄存器访存部件获取执行所需要的操作数.GPGPU 为了满足大量并发线程的同时执行,设置了数量众多的寄存器,并将这些寄存器组织为多个单端口的寄存器块,方便多个并发线程的同时访问.分支处理单元用来正确处理程序中的分支转移指令,该部件为每个线程组均分配了 1 个栈结构,又称为重汇聚栈.SM 通过访存端口和片上网络通信,以此实现对片外存储器的访问.

为了提高数据访问效率,GPGPU 采用多级高速缓存结构,目前常见的 GPGPU 采用 2 级高速缓存结构.通常情况下,寄存器和 1 级 cache 放置在 SM 片上,2 级 cache 则通过片上互联网络和所有的 SM 相连.

1.2　基于重汇聚栈的分支转移控制

分支转移指令在通用应用程序中非常普遍.如果线程组在执行过程中遇到分支转移指令,不同分支路径只能串行执行,必然降低执行任务的 TLP.而且,不同分支路径的线程在执行完相应的分支路径后,如果没有进行特殊的重汇聚处理,会影响后续指令的执行效率.基于重汇聚栈的重汇聚机制(post-

dominator, PDOM)[9-10]可以有效解决由于分支转移带来的性能降低. 该机制能在重汇聚处将同一线程组中执行不同分支路径的线程重组为分支前的线程组,以提高后续指令的 TLP.

PDOM 机制主要通过重汇聚栈实现对转移分支的控制执行. 重汇聚栈的表项由重汇聚地址(reconvergence program counter, RPC)、线程活跃掩码和每个程序基本块的首条指令 PC(program counter)3 部分组成,其中活跃掩码的每一位对应1 个线程的状态,为"1"表示对应线程活跃. 图 2 展示了 PDOM 重汇聚机制对分支控制流的处理过程示例,并假设 1 个 warp 由 8 个线程组成. 1)在遇到分支转移时,将栈顶初始化为重汇聚基本块的 PC,将其对应的活跃掩码位全部置为 1;2)将不同分支路径对应的表项依次压入栈中,并将栈顶指针 TOS 指向最终的栈顶,如图 2(b)所示;3)选择栈顶对应的分支路径执行,将该表项从栈中弹出,并修改栈顶指针 TOS 的指向,如图 2(c)(d)所示. 此时,栈顶只剩下重汇聚基本块对应的表项,warp 中的线程在此进行重汇聚,汇聚完成后继续按锁步的方式向前执行.

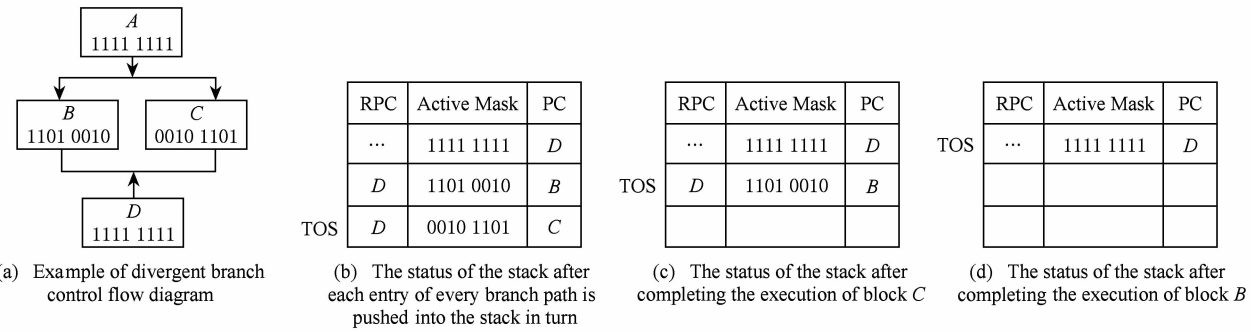


Fig. 2 Example of PDOM reconvergence mechanism based on reconvergence stack.

图 2 基于重汇聚栈的 PDOM 重汇聚机制示例

2 CAPRI

CAPRI 方法主要通过预测的方式来判断后续线程块是否适合压缩重组. 为了进行压缩适合性预测,CAPRI 方法设置了 1 张压缩预测表来记录各个转移分支的压缩重组历史. 在遇到分支转移时,依据压缩预测表中对应分支的压缩重组历史信息来判断是否对当前线程组进行压缩重组. 另外,不管是否进行压缩重组,在每次进行线程块压缩重组之后,CAPRI 都需要对当前分支的压缩适合性进行重新分析,并对相应的压缩重组历史信息进行修正. 图 3 展示了 CAPRI 方法进行线程块压缩重组预测的示例. 线程压缩重组单元沿用了 TBC 策略中的设计,它主要负责对多个线程组进行线程重组. 内部的优先级编码器会依次从不同的 SIMD 通道中选取第 1 个未选中的活跃线程进行组合,如图 3(b)(d)所示,其组合的结果分别如图 3(c)(e)所示.

线程块压缩重组完成后,会将对应的活跃掩码传送给线程块压缩适合性预测器进行分析. 该预测器主要统计每个通道的活跃线程数,并选择其中的最大值. 如果该最大值小于压缩重组前的活跃线程数,则表明此次压缩重组是适合的,并修正该分支的压缩重组历史位. 从图 3(f)可以看出,经过线程块

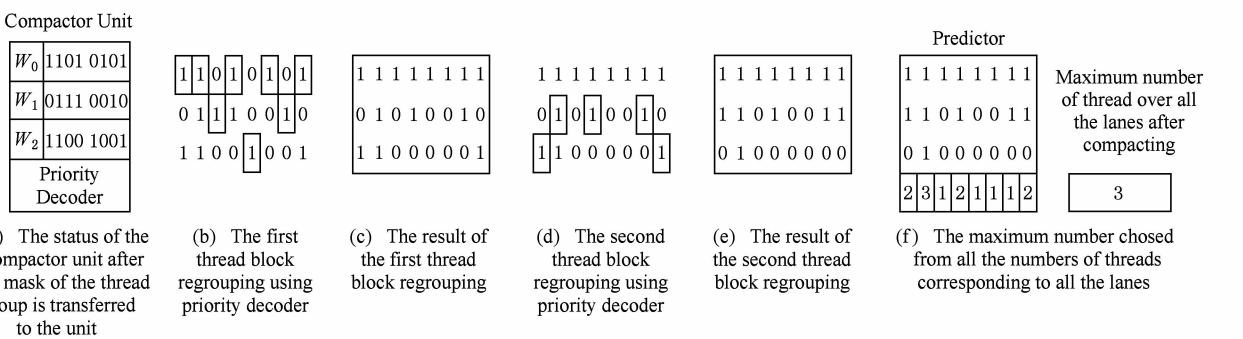


Fig. 3 Example of thread compaction, regrouping and prediction.

图 3 线程块压缩、重组、预测过程示例

压缩重组后的线程块数量为3,与压缩前的线程块数量相同,表明此次线程块压缩重组不适合.

然而,CAPRI方法仍然会产生一定的无效压缩重组,主要原因包括3个方面:1)由于CAPRI策略采用首次保守预测,某个分支第1次出现时均按适合压缩重组执行,这种预测错误的概率接近50%.对于某些类型的转移分支来说,这种预测错误可能会对性能产生较大的影响.例如访存密集型分支路径或是执行次数较少的分支,此时一次预测错误产生的开销可能占整个开销较大的比例.2)活跃线程对应的通道分布情况与分支指令计算参数相关,而不同TB的线程对应参数并不一定具有相似性和相关性,因此用当前TB的压缩适合性预测后续TB的压缩适合性存在不确定性.3)在进行压缩适合性预测时,CAPRI并没有考虑压缩重组产生的PG和PO之间的关系.即使CAPRI策略预测当前线程块适合压缩重组,若线程块压缩重组后PG小于PO,系统性能仍然会下降.

3　2阶段同步的线程块压缩重组调度(TSTBC)

CAPRI在预测线程块压缩适合性的准确度上存在一定的误差,仍然会出现一定比例的无效压缩重组.与CAPRI方法采用预测的方式不同,TSTBC采用主动分析的方式,分2个阶段对线程块压缩适合性进行分析,并在第2阶段采用线程块局部重组压缩,进一步提高了压缩重组的有效性.

为了实现TSTBC,相对于基准的GPGPU体系结构,对重汇聚栈进行了适当修改,增加了压缩适合性判断逻辑部件和线程块压缩重组部件.其中,线程块压缩重组部件的实现参考TBC方法中的实现.

3.1　线程块局部压缩重组

在对线程块进行压缩重组的过程中,对线程块压缩重组有利的warp分布与处理任务本身和其计算的数据有关,有利于压缩重组的warp将分布在各种可能的位置,甚至有可能集中分布在线程块的某个局部区域.例如对于线程块 T_{th},假设其包含8个warp,分别为 $w_1,w_2,\cdots,w_8$.如果只有 w_5,w_6,w_7,w_8 对 T_{th} 的压缩重组有贡献,则真正发生重组的warp集中在该线程块的后半部分,称这种情况为线程块的局部压缩重组,它是TSTBC方法的基础.

3.2　TSTBC的算法思想

TSTBC方法分2个阶段对线程块压缩适合性进行判断,在阶段1进行整体分析,在阶段2则对线程块进行局部压缩重组适合性分析,以进一步挖掘线程块中适合压缩重组的机会.

与TBC和CAPRI一样,TSTBC同样需要在分支转移处对属于同一个线程块的warp同步.然而,与它们不同的是,这种同步分为2个阶段.

阶段1. TSTBC仅对前 n 个到达的warp进行同步.n 在此取经验值,通过在实验中测试 n 取不同值时($n=1,2,\cdots,N$,其中 N 为TB包含的warp数)不同测试程序对应的系统性能,取对应平均性能最好的 n 值作为 n 的最终取值.在对前 n 个warp同步之后,需要对这 n 个warp进行压缩适合性分析.通过对这 n 个warp的活跃掩码分析,可以分析出压缩重组后的warp数量.如果分析的结果小于压缩重组前的warp数量,则表明前 n 个warp对整个压缩重组是有利的,从而可以认为该线程块有可能适合重组压缩.然后,再对当前TB余下的warp进行同步,并对所有warp对应的掩码进行进一步的压缩适合性分析.如果当前TB压缩后生成的warp数小于某个warp压缩阈值(warp compaction threshold,WCT) T_{WCT},表明当前TB压缩重组后产生的PG大于PO,则最终判断当前TB适合进行压缩重组,并对当前TB进行压缩重组.若分析结果表明前 n 个warp不适合压缩重组,则使前 n 个warp跳过线程块压缩重组部件继续向前执行,并进入到阶段2.压缩阈值参数 T_{WCT} 的取值与参数 n 取值的方法类似.

阶段2. 首先调度前 n 个已经到达的warp,让其跳过后续的线程压缩重组并继续向前执行;然后对余下的 $N-n$ 个warp进行同步;最后对这 $N-n$ 个warp进行压缩适合性分析.如果分析结果表明余下的warp适合压缩重组,则对余下的warp进行压缩重组,否则表明当前线程块不适合局部压缩重组.为了实现的简单,阶段2的压缩适合性判断仅比较压缩前后warp的数量.但是,阶段2也可以增加与新的压缩阈值参数的比较,以进一步提高重组压缩适合性判断的有效性.该压缩阈值参数与阶段1中压缩阈值参数选取的方法一致,此工作将在以后的研究中继续完善.

TSTBC具体的线程块压缩适合性判断算法如算法1所示.在算法1中,最外层的if语句对不同的分支路径进行处理,其if语句部分实现了对当前分支进行2阶段同步的线程块压缩重组;else部分用于处理与当前分支路径对应的其他分支.对其他分支的处理无需再对线程组进行同步,因为处理完第1条分支后,所有的warp均已达到.

算法 1. 2 阶段同步的线程块压缩适合性判断算法.

```
/* 函数 maximum 统计所有通道的最大线程数,函数 wcu 用于对线程块压缩,变量 t_max 表示所有通道的最大线程数,变量 wcnt 表示到达的 warp 数量 */
if (wcnt<N)
{ 同步前 n 个 warp;
  t_max=maximum(每个通道的线程数);
  if (t_max<n)
  { 同步所有的 warp;
    t_max=maximum(每个通道的线程数);
    if (t_max<T_WCT)
    /* 线程块压缩单元压缩当前线程块 t_TB */
      wcu(t_TB);
  }
  else
  { 调度前 n 个到达的 warp;
    同步剩余的 N-n 个 warp;
    t_max=maximum(剩余每个通道的线程数);
    if (t_max<N-n)
      wcu (剩余的 N-n 个 warp);
  }
}
else
{ t_max=maximum(每个通道的线程数);
  if (t_max< T_WCT)
  /* 线程块压缩单元压缩当前线程块 t_TB */
    wcu(t_TB);
}
```

如果某个 warp 中所有线程在分支转移处均选择相同的分支路径,称该 warp 没有发生转移. 对于这样的 warp 将不进行任何压缩适合性判断,因为这样的 warp 对于线程压缩重组不会有任何贡献. 因此,在进行 TB 的压缩适合性判断分析之前,应过滤掉所有没有发生转移的 warp,让它们继续向前执行. 因而,可能会存在一种特殊情形,即当所有 warp 达到后,发生转移的 warp 数量有可能少于 n,此时仅比较压缩前后的 warp 数量多少,主要原因是由于可供分析的 warp 数量偏少. 该细节在算法实现时进行了考虑,但在算法 1 中没有体现.

分 2 阶段对线程块进行同步,可以进一步减少因同步等待产生的开销. 在阶段 1 仅对前 n 个 warp 同步,并对它们进行了初步的压缩适合性判断. 如果分析结果为真,则表明它们对整个线程块的压缩重组是有利的,可以将它们纳入到整个线程块的压缩重组过程;否则,表明它们对整个线程块的压缩重组没有贡献,可以在后续的压缩重组处理过程中忽略它们,此时可以让这部分 warp 继续向前执行,减少了它们的同步等待延时,从而能减少整个线程块的同步等待开销.

当阶段 1 分析表明前 n 个 warp 对整个线程块的压缩重组没有贡献,根据线程块局部压缩重组的思想,并不意味着剩余的 warp 不适合进行压缩重组. 设置同步阶段 2,可以进一步挖掘适合压缩重组的局部线程块,提高压缩重组的机会和有效性. 因此,若通过分析判断剩余的 warp 适合进行压缩重组,则线程块局部压缩重组有利于系统的性能提升.

3.3 改进的重汇聚栈

改进的重汇聚栈中,每个表项的活跃掩码以线程块级为单位,且增加了字段 *wcnt*,该字段用来统计不同分支路径当前达到的线程块数. 虽然活跃掩码的宽度是线程块级,但是线程调度执行的基本单位仍然是 warp.

图 4 展示了改进的重汇聚栈结构示例. 这里假设一个 TB 包含 4 个 warp,每个 warp 包含 4 个线程. 图 4 中栈顶的 *wcnt* 值表明基本块 C 所在分支路径已经有 2 个 warp 到达.

	RPC	Active Mask	PC	*wcnt*
	…	1111 1111 1111 1111	*D*	0
	D	1101 1010	*B*	2
TOS	*D*	0010 0101	*C*	2

Fig. 4 Example of improved reconvergence stack structure.

图 4 改进的重汇聚栈结构示例

3.4 压缩适合性判断逻辑部件

压缩适合性判断逻辑部件在线程块压缩适合性分析的 2 个阶段都需要用到,它对每个 SIMD 执行通道对应的线程数进行统计,并从中选出最大值,该最大值就是线程块压缩重组后产生的 warp 数量. 图 5 展示了压缩适合性判断逻辑部件的结构示例. 此处仍然假设 1 个 TB 包含 4 个 warp,每个 warp 包含 4 个线程. 从图 5 可以看出,该 TB 对应活跃掩码经过压缩后将产生 3 个新的 warp. 在 CAPRI 方法中,下一个 TB 被判断为适合进行压缩重组;但是,TSTBC 方法还需要将分析得到的最大值与 T_{WCT} 进行比较方能进行判断.

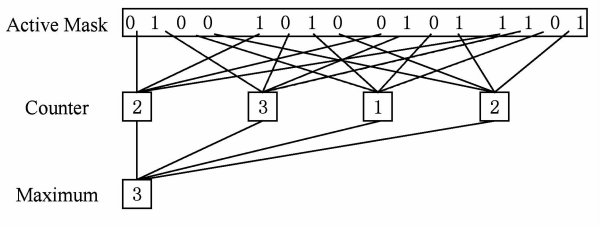
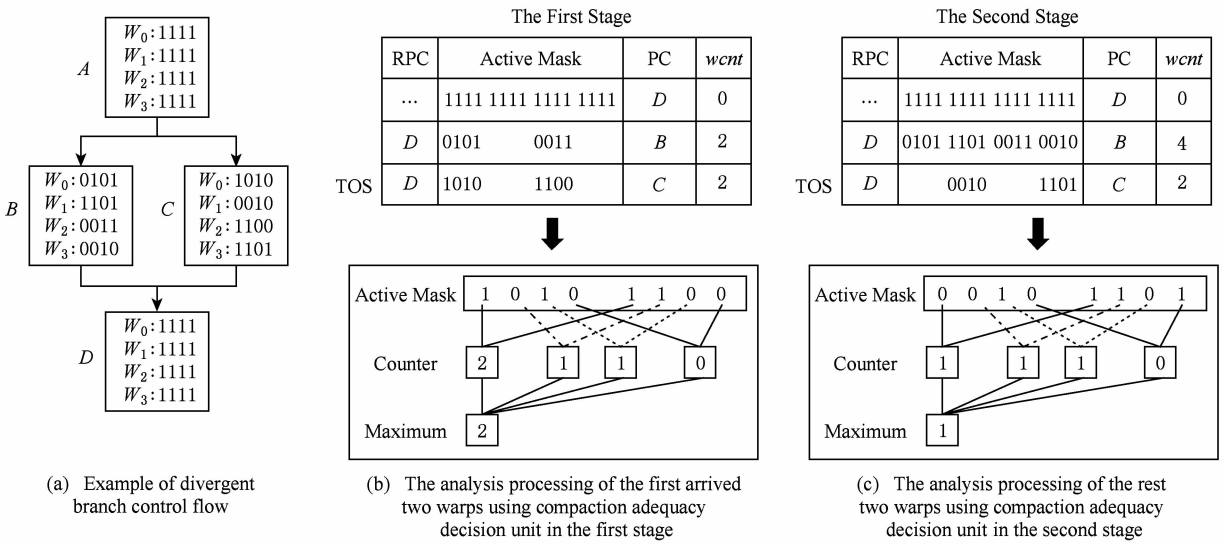


Fig. 5　Example of thread block compaction adequacy decision logic structure.

图 5　线程块压缩适合性判断逻辑部件结构示例

图 6 展示了出现分支转移时压缩适合性判断逻辑部件对 1 个线程块进行压缩适合性分析判断的示例. 该示例中同样假设 1 个 TB 包含 4 个 warp,每个 warp 包含 4 个线程,参数 n 和 T_{WCT} 均取值为 2. 图 6(b)表示线程块压缩适合性判断分析的阶段 1. 当遇到分支转移时,首先将重汇聚块 D 对应的表项压入栈中,此时同步等待前 2 个 warp 到达,并将不同分支路径对应的表项压入栈顶,通过逻辑分析部件对栈顶的掩码进行分析. 分析的结果为 2,表明如果对前 2 个 warp 进行压缩,压缩后将产生 2 个新的 warp,这和压缩之前的 warp 数量相同. 因此,先前到达的 2 个 warp 并不适合进行压缩重组. 图 6(c)表示线程块压缩适合性判断分析进入了阶段 2. 在阶段 2,首先对剩下的 warp 进行同步,再对它们进行压缩适合性分析. 分析的结果为 1,表示此部分线程块压缩重组后只产生 1 个新的 warp,小于压缩前的 warp 数量,表示余下的 warp 适合进行压缩重组.



(a) Example of divergent branch control flow

(b) The analysis processing of the first arrived two warps using compaction adequacy decision unit in the first stage

(c) The analysis processing of the rest two warps using compaction adequacy decision unit in the second stage

Fig. 6　Example of thread block compaction adequacy decision analysis using TSTBC as branch divergence occurs.

图 6　分支转移时 TSTBC 进行线程块压缩适合性判断分析示例

3.5　TSTBC 与 CAPRI 的比较

CAPRI 方法和本文提出的 TSTBC 方法都是从线程压缩重组适合性的角度对线程的压缩重组进行优化,都是为了提高线程压缩重组的有效性,尽量避免无效的线程块压缩重组,从而减少对线程组内甚至是线程组间的数据局部性的破坏. 然而,相对于 CAPRI 方法,TSTBC 方法在对线程块的压缩重组优化方面具有 2 方面的优势:

1) CAPRI 方法用预测的方式来对后续线程块压缩重组的适合性进行判断,而 TSTBC 方法对每个线程块均采用主动的分析判断其压缩重组的适合性,并分 2 个阶段进行判断,其准确度和有效性更高.

2) CAPRI 方法和 TSTBC 方法对于适合进行压缩重组的线程块均需对包含的 warp 同步,但是由于 TSTBC 方法分 2 个阶段来对线程块进行同步,当出现线程块适合局部压缩重组的情形时,TSTBC 方法导致的同步等待延时相对更短.

4　实验及结果分析

4.1　实验平台

为了对提出的方法进行验证,我们对时钟周期级性能模拟器 GPGPU-sim(3.2.2 版本)[11] 进行了相应的修改,并分别实现了对 TSTBC,SCC(swizzled cycle compaction)[12], CAPRI(1bL)(简称为 CAPRI)方法的模拟. 其中,选用 SCC 方法进行比对,是因为该方法是一种线程组内的线程压缩重组方法且具有代表性,这样可以从其他的角度进行分析比较,也使得实验更加充分合理. 实验过程中对 GPGPU-sim 的配置主要参照 NVIDIA Fermi 系列体系结构,具体

的部分参数配置如表 1 所示.

测试程序的选取来源于 GPGPU-sim 模拟器、NVIDIA CUDA SDK[13] 和 Rodinia[14] 等. 我们将测试程序分为 2 类:1)一类测试程序在出现分支时并未发生分支转移现象,对这类测试程序进行线程块压缩重组的意义不大,将这类程序划分为分支非转移类测试程序;2)另一类测试程序划分为转移类测试程序. 实验过程一共使用了 14 个测试程序,涵盖了图像处理、概率统计分析、生物信息、模式识别、线性代数等多个领域,其中分支转移类测试程序有 6 个,分支非转移类测试程序有 7 个. 所用的测试程序如表 2 所示.

Table 1 GPGPU-sim Parameter Configuration

表 1 GPGPU-sim 参数配置

Parameter	Configuration
Number of SM	15
Size of Warp	32
Maximum Number of Threads per SM	1536
Maximum Number of TB per SM	8
Pipeline Width of SIMD	16
Number of Registers per SM	32768
Size of Shared Memory per SM/KB	48
L1 Data Cache	16 KB, 128B line, 4-way association
L2 Data Cache	768 KB, 128B line, 8-way association
Number of Memory Access Controllers	6
Size of DRAM Scheduling Queue	16
Timing of GDDR5	$t_{CL}=12$, $t_{RP}=12$, $t_{RC}=40$, $t_{RAS}=28$, $t_{RCD}=12$, $t_{RRD}=6$
Thread Scheduling Strategy	GTO

Table 2 Benchmark for the Evaluation

表 2 实验所用测试程序

Application	Abbreviation	Number of Instruction	Type	Benchmark
MUMMER-GPU	MUM	8.3E+7	Divergent	GPGPU-sim
3D Laplace Solver	LPS	6.7E+7	Divergent	GPGPU-sim
fastWalshTransform	FWT	3.9E+10	Divergent	CUDA SDK
mergeSort	MS	1.33E+11	Divergent	CUDA SDK
HotSpot	HSP	1.03E+8	Divergent	Rodinia
Needleman-Wunsch	NW	1.96E+8	Divergent	Rodinia
MonteCarlo	MC	9.57E+8	Non-divergent	CUDA SDK
Ray Tracing	RAY	6.2E+7	Non-divergent	GPGPU-sim
Back Propagation	BP	1.82E+8	Non-divergent	Rodinia
LIBOR	LIB	5.75E+8	Non-divergent	GPGPU-sim
Coulomb Potential	CP	1.12E+8	Non-divergent	GPGPU-sim
StoreGPU	STO	9E+7	Non-divergent	GPGPU-sim
AES Cryptography	AES	2.7E+7	Non-divergent	GPGPU-sim

4.2 结果分析

本文从压缩有效性、1 级数据 cache(L1D cache)访问失效率、系统性能等方面对实验结果进行了分析,并分别对 PDOM, SCC, CAPRI, TSTBC 四种线程调度方法进行了比较.

4.2.1 压缩有效性

实验中主要用压缩适合性分析的准确度对压缩有效性进行度量. 图 7 展示了一组分支转移类测试程序的线程块压缩重组有效性. 图 7 中,Stall/Bypass 表示线程块不应该进行压缩重组,但实际进行了压缩

重组;Bypass/Stall 表示线程块应该进行压缩重组,但实际未进行压缩重组;Stall/Stall 表示对线程块进行了有效地压缩重组;Bypass/Bypass 表示有效地跳过了线程块的压缩重组. 4 种情形中,只有 Stall/Stall 和 Bypass/Bypass 表示对线程块压缩重组的适合性进行了正确分析. 实验中对每个测试程序均用 4 种不同线程调度方法进行测试. 从图 7 可以看出,对于该类测试程序,TSTBC 压缩重组的有效性分析(包括 Stall/Stall 和 Bypass/Bypass)均优于其他 3 种方法,其平均准确率达到了 92.24%,相对 PDOM 和 CAPRI 分别提高了 16.07%和 7.59%,尤其是对于需要跳过的线程块压缩重组,其准确率和 PDOM 非常接近;SCC 压缩重组的有效性最低,因为实际上只有平均 18.7%的分支真正发生了转移,而真正适合用 SCC 进行线程组内的压缩重组的分支比例为 15.68%. 另外,从图 7 可以看出,在 NW 中几乎所有的分支转移均不适合线程块压缩重组,这是由于在该测试程序中绝大部分的分支判断处线程块均未发生实际转移.

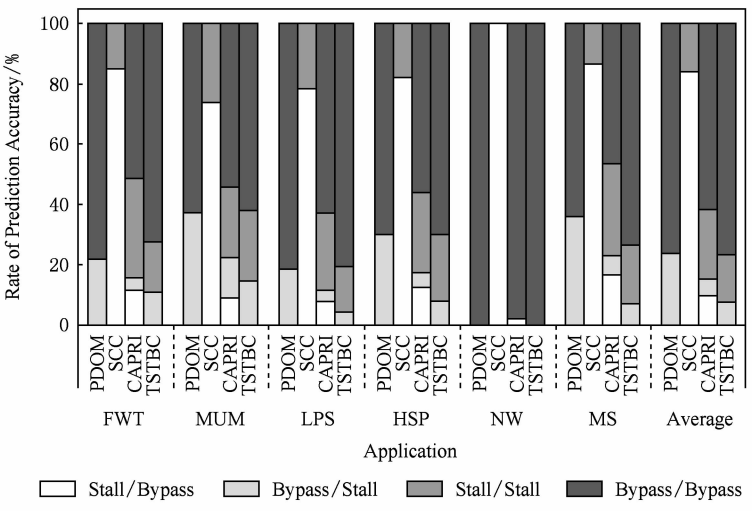


Fig. 7　Compaction validity of branch divergent benchmarks.

图 7　分支转移类测试程序压缩有效性

图 8 展示了一组分支非转移类测试程序的线程块压缩重组有效性. 由于此类程序执行过程中几乎没有发生分支转移,因此传统的 PDOM 方法执行此类程序时并不会对性能产生影响. TSTBC 对该类程序的线程块压缩重组有效性分析的准确率接近 100%,CAPRI 方法的准确率也达到了 99.3%,而 SCC 对此类程序进行的线程块压缩重组几乎都是无效的.

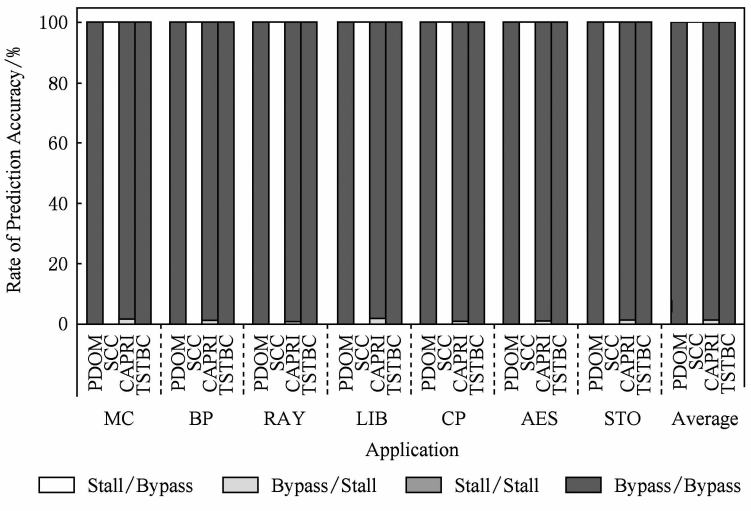


Fig. 8　Compaction validity of branch non-divergent benchmarks.

图 8　分支非转移类测试程序压缩有效性

4.2.2 L1D cache 访问失效率

由于将属于不同 warp 的线程重组在一起,线程压缩重组在一定程度上破坏了原来 warp 内部的数据局部性,会增加片外访存次数和 L1D cache 的访问失效率.图 9 和图 10 分别展示了分支转移类测试程序和分支非转移类测试程序对应于 PDOM,SCC,CAPRI,TSTBC 四种方法的 L1D cache 访问失效率对比情况.从图 9 可以看出,对于分支类测试程序,由于提高了线程块压缩重组的有效性,相对于 CAPRI,TSTBC 的 L1D cache 的平均访问失效率减少了 4.53%.对于 NW,由于 CAPRI 和 TSTBC 对线程压缩重组的判定基本和 PDOM 一致.因此,这 2 种方法产生的失效率也接近 PDOM.

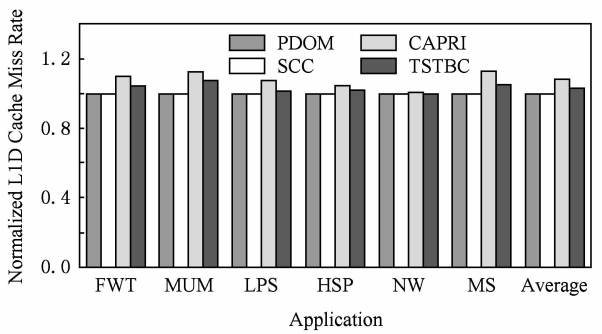


Fig. 9 Normalized L1D cache miss rate of branch divergent benchmarks.

图 9 分支转移类测试程序归一化 L1D cache 失效率

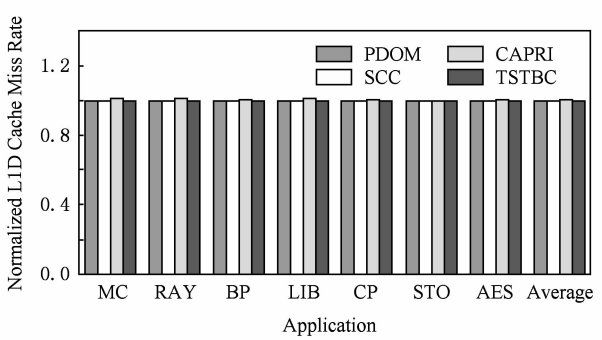


Fig. 10 Normalized L1D cache miss rate of branch non-divergent benchmarks.

图 10 分支非转移类测试程序归一化 L1D cache 失效率

对于分支非转移类测试程序,CAPRI 和 TSTBC 的压缩重组有效性基本上都接近 100%,因此对数据局部性的破坏很小,因此这 2 种方法的 L1D cache 访问失效率和 PDOM 的非常接近.由于 SCC 属于线程组内的线程压缩重组,对所有测试程序的 L1D cache 访问失效率均不会产生其他的影响,因此它和 PDOM 产生的失效率是一致的.

4.2.3 性能分析

本文主要以 IPC(instruction per cycle)为主要性能指标对测试程序的运行性能进行分析.图 11 和图 12 分别展示了对应于上述 4 种方法分支转移类测试程序和分支非转移类测试程序的性能结果比较.从图 11 可以看出,对于分支转移类测试程序,后 3 种线程块调度方法均有不同程度的性能提升,其中 TSTBC 的 IPC 提升幅度最大,平均 IPC 比 SCC 和 CAPRI 分别提高了 13.3%和 9.4%,相对于基准的 PDOM 方法提升了 19.27%.

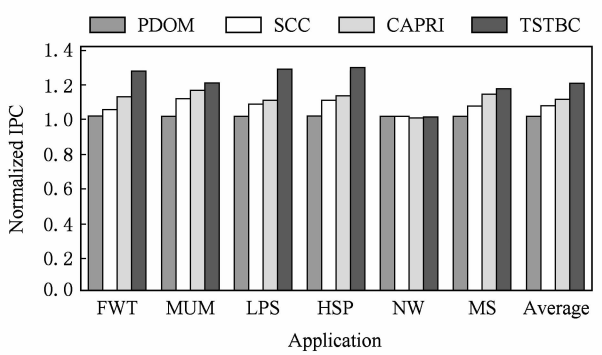


Fig. 11 Normalized IPC of branch divergent benchmarks.

图 11 分支转移类测试程序归一化 IPC

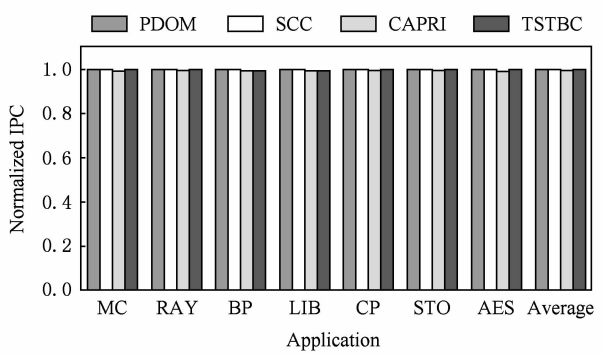


Fig. 12 Normalized IPC of branch non-divergent benchmarks.

图 12 分支非转移类测试程序归一化 IPC

对于分支非转移类测试程序,TSTBC 和 CAPRI 方法的性能均接近于 PDOM 方法,主要是因为它们都对非转移分支进行了过滤.因此,TSTBC 对这类应用程序的系统性能并不会产生太大的影响,CAPRI 方法由于对压缩重组适合性判断存在误差,会产生少许的同步开销.

4.2.4 参数取值分析

为了确定 TSTBC 算法中参数 n 和压缩阈值参数 T_{WCT} 的值,我们分别对 n 和 T_{WCT} 取 1,2,…,N 之间的所有值,然后测试对应每种 n 和 T_{WCT} 取值组合

下所有分支转移类测试程序运行的 IPC,从最终 N^2 个结果中选取 IPC 最好情况下对应的 n 和 T_{WCT} 的值作为这 2 个参数最终的值.由于篇幅的限制,在此仅将结果最好的 2 组数据展示出来.

图 13 展示了当 $T_{WCT}=2$、$n=1,2,\cdots,8$ 时整个分支转移类测试程序的平均 IPC.从图 13 可以看出,当 $T_{WCT}=2$,$n=3$ 时,该类测试程序的平均 IPC 最高.另外,当 T_{WCT} 取一定值且 $n=1,7,8$ 时的 IPC 相差无几,此时 TSTBC 基本上退化成阶段 1 同步的线程块压缩重组,因为此时阶段 2 分析中的某阶段的 warp 数量为 1 或 0,已经无需再进行分析处理.

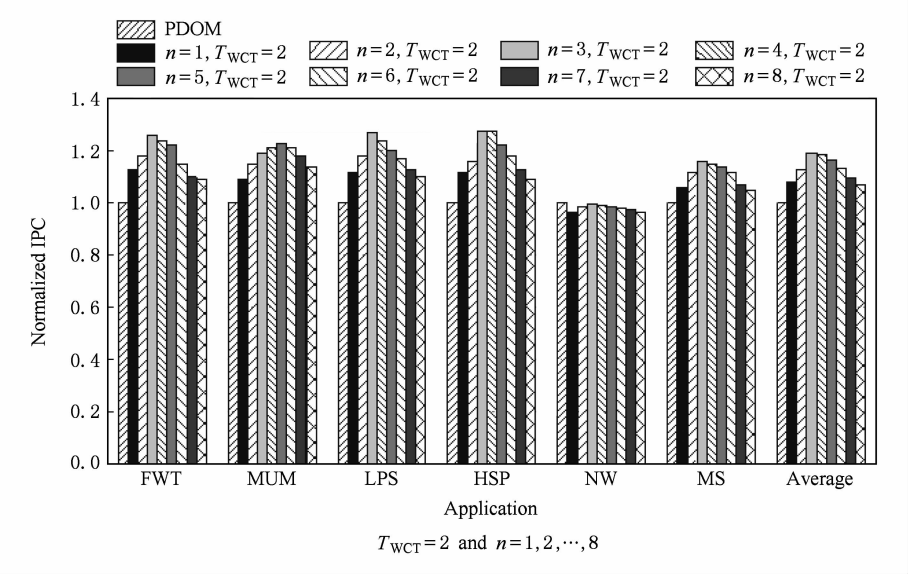


Fig. 13　Normalized IPC of branch divergent benchmarks.

图 13　分支转移类测试程序的归一化 IPC

图 14 展示了当 $n=3$ 且 $T_{WCT}=1,2,\cdots,8$ 时分支转移类测试程序的平均 IPC.图 14 同样可以发现当 $T_{WCT}=2$,$n=3$ 时,分支转移类测试程序的平均 IPC 最高.因此,最终将确定 $T_{WCT}=2$,$n=3$.在 4.2.1,4.2.2,4.2.3 节的实验结果均以此为基础.

另外,从图 14 可以看出,当 $T_{WCT}>5$ 时,各测试程序的性能基本上都是低于 PDOM 方法,原因是因为此时压缩程度超过 5 的线程块非常少,而且同步线程块内的 warp 会产生一定的开销,反而降低了系统性能.

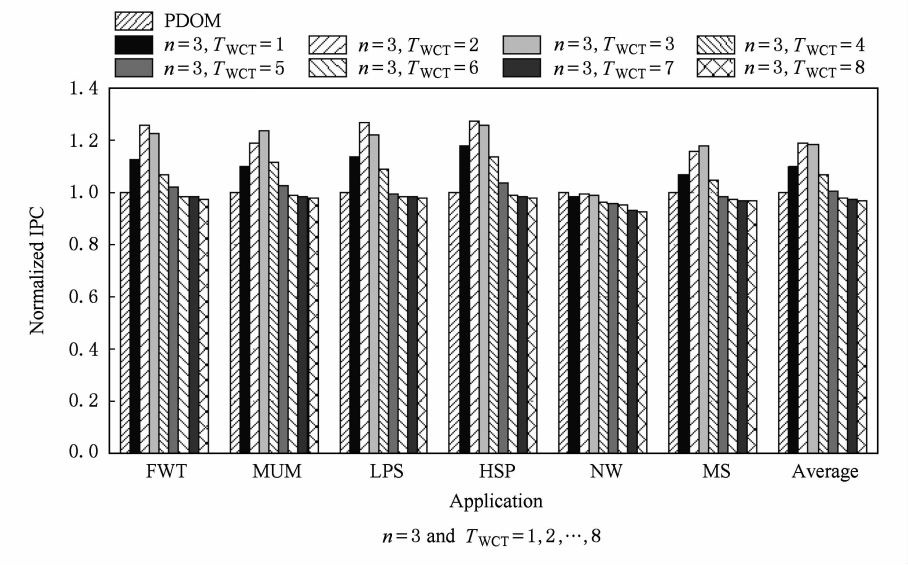


Fig. 14　Normalized IPC of branch divergent benchmarks.

图 14　分支转移类测试程序的归一化 IPC

4.2.5 硬件开销

本方法需在传统基准体系结构的基础上增加线程块压缩重组部件以及线程块压缩适合性判断逻辑2个部件,并对重汇聚栈的结构进行了一定的改动.其中,重汇聚栈的结构是在参照TBC方法中的基础上进行了一定的改动,每个表项增加了字段 $wcnt$,该字段用4 b表示.由于每个线程块设置一个栈,实验中每个栈的表项长度设为32,因此每个栈只需增加16 B.另外,每个核上允许执行的最大TB数为8,因此一个SM增加的硬件大小仅为128 B.线程块压缩重组部件的设计参考TBC的设计,其硬件开销参考文献[6]中的分析.线程块压缩适合性判断逻辑相对于CAPRI中的压缩适合性判断逻辑部件的设计,去掉了记录压缩重组适合性历史信息的表结构,因此其硬件开销要小于CAPRI的硬件开销.

5 相关工作

分支转移在很多通用应用程序中都存在,GPGPU在处理该类应用程序时的性能会由此受到影响.目前,对分支转移处理进行性能优化的研究主要集中在线程组间的线程重组、线程组内的线程重组和多分支并行执行3个方面.

Fung等人[10]提出了动态的warp重组策略DWF(dynamic warp formation)和线程块压缩重组策略TBC[6].Narasiman等人[15]提出了LWM(large warp microarchitecture)策略,通过提高线程组的宽度来增加每个分支路径上的活跃线程数量,并在大的线程组内进行线程重组.LWM中提到的large warp由多个warp组成,实质上还是在线程组间进行线程重组.Malits等人[16]提出的线程重组策略ODGS(oracle dynamic global scheduling)将线程组间的线程重组从SM内扩展至SM之间.上述线程重组压缩方法都属于线程组间的线程重组,但是这些方法都没有考虑线程压缩重组的有效性,产生的无效压缩重组会带来较大的开销.本文提出的TSTBC方法着重考虑了线程组间线程压缩重组的有效性,产生的开销更小.

Vaidya等人[12]提出了线程组内的线程重组机制BCC(basic cycle compaction)和SCC,利用实际的物理SIMD通道数比线程组宽度小的特点,将线程组内的线程重组为与实际物理SIMD通道宽度相等的线程组,压缩了原始线程组的执行周期数.Jin等人[17]提出的线程组内的线程重组机制HWS(hybrid warp size)也包含了这样的思想.此类方法避免了破坏线程组内的数据局部性,然而该方法受限于实际SIMD通道数的配置情况,尤其当实际SIMD物理通道数接近或等于线程组的宽度时,线程组内可供重组的机会大大减少,有效的线程压缩重组更少,对整个系统性能提升影响很小.因此,线程组内的线程重组方法仅适用于某些特定的硬件环境,适用性较差.而本文提出的TSTBC方法没有这方面的限制.

Brunie等人[18]同时提出的线程重组策略SBI(simultaneous branch interweaving)和SWI(simultaneous warp interweaving),通过对指令发射部件的修改实现了双分支路径上同时发射指令,同时也包含了线程压缩重组的思想.Wang等人[19]提出并实现了MSMD(multiple SIMD, multiple data)执行模型,设置了多个可灵活划分的、独立的SIMD数据通道,使得不同分支路径上的线程组可以同时执行.但是,该类方法的物理实现更加复杂,实用性较低.

6 总 结

本文对前人提出的线程块压缩重组调度算法进行了分析和讨论,并在此基础上提出了2阶段同步的线程块压缩重组调度方法TSTBC.该方法分2个阶段对线程块的压缩重组适合性进行分析,提出了线程块局部压缩重组的思想.相对于CAPRI方法,TSTBC方法能进一步提高线程块压缩重组的有效性,减少了由线程块压缩重组而产生的开销和对线程组内数据局部性的破坏,并降低L1D cache访问的失效率.实验结果表明相对于PDOM方法和CAPRI方法,TSTBC方法的平均IPC分别提高了19.27%和9.4%.

然而,本方法在实现上仍然存在一定的不足,主要是实验过程中对参数 n 和 T_{WCT} 选取静态值,对于有的应用程序并不是最佳取值.因此,后续工作中还需进一步考虑对不同的应用程序动态确定参数 n 和 T_{WCT} 的取值.

参 考 文 献

[1] Meng J, Tarjan D, Skadron K. Dynamic warp subdivision for integrated branch and memory divergence tolerance [C] // Proc of the 37th Int Symp on Computer Architecture. New York: ACM, 2010: 235-246

[2] Lindholm E, Nickolls J, Oberman S, et al. NVIDIA Tesla: A unified graphics and computing architecture [J]. IEEE Micro, 2008, 28(2): 39-55

[3] Keckler S W, Dally W J, Khailany B, et al. GPUs and the future of parallel computing [J]. IEEE Micro, 2011, 31(5): 7-17

[4] Nickolls J, Dally W J. The GPU computing era [J]. IEEE Micro, 2010, 30(2): 56-69

[5] Du P, Weber R, Luszczek P, et al. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming [J]. Parallel Computing, 2012, 38(8): 391-407

[6] Fung W W L, Aamodt T M. Thread block compaction for efficient SIMT control flow [C] //Proc of the 17th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2011: 25-36

[7] Rhu M, Erez M. CAPRI: Prediction of compaction-adequacy for handling control-divergence in GPGPU architectures [C] //Proc of the 39th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2012: 61-71

[8] Aamodt T M, Fung W W L. GPGPU-Sim3.x manual [EB/OL]. (2012-08-08) [2015-02-01]. http://gpgpu-sim.org/manual/index.php/GPGPU-Sim_3.x_Manual

[9] Levinthal A, Porter T. Chap—A SIMD graphics processor [C] //Proc of ACM SIGGRAPH'84. New York: ACM, 1984: 77-82

[10] Fung W W L, Sham I, Yuan G, et al. Dynamic warp formation and scheduling for efficient GPU control flow [C] //Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2007: 407-420

[11] Bakhoda A, Yuan G L, Fung W W L, et al. Analyzing CUDA workloads using a detailed GPU simulator [C] //Proc of the IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2009: 163-174

[12] Vaidya A S, Shayesteh A, Woo D H, et al. SIMD divergence optimization through intra-warp compaction [J]. ACM SIGARCH Computer Architecture News, 2013, 41(3): 368-379

[13] NVIDIA Corporation. CUDA C/C++ SDK 4.0 CODE Samples [CP/OL]. 2011 [2015-02-01]. https://developer.nvidia.com/cuda-toolkit-40

[14] Che S, Boyer M, Meng J, et al. Rodinia: A benchmark suite for heterogeneous computing [C] //Proc of the IEEE Int Symp on Workload Charact-erization. Piscataway, NJ: IEEE, 2009: 44-54

[15] Narasiman V, Shebanow M, Lee C J, et al. Improving GPU performance via large warps and two-level warp scheduling [C] //Proc of the 44th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2011: 308-317

[16] Malits R, Bolotin E, Kolodny A, et al. Exploring the limits of GPGPU scheduling in control flow bound applications [J]. ACM Trans on Architecture and Code Optimization, 2012, 8(4): 29

[17] Jin X, Daku B, Ko S B. Improved GPU SIMD control flow efficiency via hybrid warp size mechanism [J]. Microprocessors and Microsystems, 2014, 38(7): 717-729

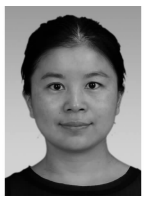
[18] Brunie N, Collange S, Diamos G. Simultaneous branch and warp interweaving for sustained GPU performance [C] //Proc of the 39th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2012: 49-60

[19] Wang Y, Chen S, Wan J, et al. A multiple SIMD, multiple data (MSMD) architecture: Parallel execution of dynamic and static SIMD fragments [C] //Proc of the 19th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2013: 603-614

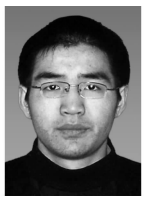
Zhang Jun, born in 1978. PhD candidate, associate professor. Member of China Computer Federation. His research interests include high performance computing and computer architecture.

He Yanxiang, born in 1952. PhD, professor and PhD supervisor. Member of China Computer Federation. His research interests include trusted software, distributed parallel processing and high performance computing.

Shen Fanfan, born in 1987. PhD candidate. His research interests include high performance computing, computer architecture and storage system (shenfanfan_168@qq.com).



Jiang Nan, born in 1976. PhD candidate, lecturer. Member of China Computer Federation. Her research interests include trusted software, trusted compilation and computer architecture (nanjiang@whu.edu.cn).



Li Qing'an, born in 1986. PhD, associate professor. Member of China Computer Federation. His research interests include compilation optimization, computer architecture and embedded system (qali@whu.edu.cn).