计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131387
52(5):1160-1176, 2015

面向局部性和并行优化的循环分块技术

刘 松 伍卫国 赵 博 蒋 庆
(西安交通大学电子与信息工程学院 西安 710049)
(lsong28@stu.xjtu.edu.cn)

Loop Tiling for Optimization of Locality and Parallelism

Liu Song, Wu Weiguo, Zhao Bo, and Jiang Qing
(*School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an* 710049)

Abstract Loop tiling is a widely used loop transformation for exposing/exploiting parallelism and data locality in modern computer architecture. It is mainly divided into two categories: fixed and parameterized. These two types of tiling technologies are systematically summarized and their advantages and disadvantages are analyzed comprehensively. Since the tile size would significantly affect the performance of the tiled code, various methods of optimal tile size selection are described. Besides, various kinds of technologies applied to multi-level tiling, parallelism exploration and imperfectly nested loops are surveyed in this paper. Based on the detailed analysis of the current researches on loop tiling technologies, several conclusions are drawn as follows: 1) How to balance the trade-off between computation complexity and generation efficiency of tiled code has not been completely solved, and how to use loop boundaries to efficiently bound the iteration spaces for data locality enhancement also needs further study. 2) Optimal tile size selection is still a difficult and open question, and it would be significant to understand the influence of different level tile size in hierarchical memory system on performance. 3) From the perspective of application, how to automatically generate effective tiled code for arbitrarily nested loops needs further research. On the other hand, how to take full advantage of shared hierarchical memory and multi-core architectures to achieve high degree of parallelism for tiled code is another interesting direction.

Key words loop tiling; optimal tile size; program transformations; parallelism; performance optimization

摘 要 循环分块是一种广泛用于改善数据局部性和开发并行性的程序变换优化技术.主要分为2类:固定分块技术和参数化分块技术,系统地总结了这2类技术,并分析了其优缺点.由于分块大小的选择会严重影响分块代码的性能,因此介绍分析了选择最优分块大小的各种方法.此外,总结了循环分块在多级分块、并行性开发和不完美嵌套循环等方面应用的各项技术.通过对循环分块技术当前研究现状的分析,得出如下结论:1)循环分块技术中的计算复杂度和生成代码效率问题还未得到完全解决,如何利用循环边界有效地约束迭代空间并提高数据局部性还需要更深入的研究;2)最优分块大小的选择依然是一个开放式难题,研究清楚分级存储架构中每级分块对性能的影响具有重要的意义;3)从循环分块的应用角度,如何有效地构建面向任意嵌套循环集的自动分块代码生成系统,同时充分利用深度共享存储资源和多核架构实现分块代码的高并行度,也是一个需要深入研究的问题.

关键词 循环分块;最优分块大小;程序变换;并行性;性能优化

中图法分类号 TP314

收稿日期:2013-10-25;**修回日期**:2014-04-29

基金项目:国家自然科学基金项目(91330117);国家"八六三"高技术研究发展计划基金项目(2012AA01A306,2012AA010901)

多级 cache 系统利用数据局部性原理为程序的高速运行提供了有效的支持.随着主流多核架构的兴起,cache 的层次深度逐步提高,片上高层次 cache 的数据访问速度越来越接近处理器,而由于工艺的局限,高层次的 cache 容量却难以满足需求.如何让程序充分利用多级 cache 架构资源的优势受到越来越多的关注.由于单芯片多处理器(chip multiprocessors, CMP)技术的快速发展,处理器中核数不断增加,即将迎来众核时代,然而简单高效的并行程序开发仍然是一个开放式难题,如何使大量遗留串行程序自动并行化,挖掘现有多核处理器架构和加速器(如 GPUs[1-2])的并行性能依然是一个热门研究的课题.同时,通过研究发现:许多计算密集型应用程序,特别是科学和工程计算应用程序中的嵌套循环会消耗大部分的运行时间,嵌套循环已成为亟需解决的程序热点[3].为了有效利用多级 cache 架构资源,增强数据局部性,开发带嵌套循环程序的并行性,最大程度地发掘多核架构性能,循环分块(loop tiling)技术[4-12]已经得到广泛的研究和应用.一方面,利用循环分块技术可以把较大程序工作集划分成若干适应高层次 cache 容量的较小工作集,使数据得到充分重用,以减少 cache 失效[5-7],增加程序局部性;另一方面,利用循环分块技术可以把程序特征和硬件特征进行匹配[11],充分开发硬件架构的潜在性能,同时可以使串行程序生成具有较好粒度的并行分块代码,使其适用于具有广泛应用背景的线性代数库函数(如 BLAS 库[13]中的矩阵相乘、LU 分解、三角矩阵求解等)和 stencil 计算(如时域有限差分、雅各比迭代、高斯赛德尔方程等).

循环分块技术通过对带嵌套循环的程序进行变换,重新构建代码,对迭代空间进行分块重新排序访问,以实现改善数据局部性和分块并行性的目的.循环分块技术主要分为两大类:1)固定分块技术.针对固定分块大小的分块技术,利用基于 cache 行为分析的方法静态分析数组引用的依赖关系[6],运用扫描多面体方法[8]生成循环边界的约束不等式系统,而基于循环分解的方法[14]则为解决自动分块代码生成问题提供了全新的思路.2)参数化分块技术.利用边界框方法[11]、Outset 集方法[11-12]和符号傅里叶-莫茨消除法(symbolic Fourier-Motzkin elimination, SFME)[15],把分块大小作为符号参数进行分块,可以更加有效地生成分块代码,使程序的运行时反馈和动态自适应成为可能.另外,鉴于迭代空间是多面体集的特性,利用成熟的代码生成器[16-17]可以有效地生成参数化分块代码.

由于分块大小的选择[18]对分块后代码的执行性能有重要的影响,一些研究者提出了基于模型驱动的分析方法、经验搜索和二者混合的方法来确定最优分块的形状和大小.多级分块技术则可以更好地利用现代计算机层次存储系统资源,开发具有较好粒度的程序并行性.因为很多应用程序属于不完美嵌套循环的范畴,一些研究者提出了面向任意嵌套循环程序的分块方法[19-21].

基于目前研究现状,还有一些需要深入研究的问题,比如如何在保证分块代码质量的前提下,降低分块技术本身的算法复杂度;如何明确各级分块对性能的影响从而更好地选择分块大小;如何细化分块并行粒度;如何提出有效解决面向任意嵌套循环的一般化方法;如何充分利用新兴的多核共享 cache 存储架构实现多线程程序优化等.

1　固定分块技术

循环分块技术的应用对象是具有嵌套循环的程序,嵌套循环主要分为完美嵌套循环和不完美嵌套循环 2 类.完美嵌套循环是指循环体不包含任何出口分支,所有的循环体都必须在相同的循环层内.不完美嵌套循环的循环体可能出现在不同的循环层内.图 1 是一个 stencil 计算的完美嵌套循环示例.本文介绍的分块方法均以图 1 为参照示例.本文所述的嵌套循环上下界是线性的,循环的初始执行顺序是索引值的词典顺序.

```
for (k=1;k≤N_k;k++)
  for (i=k+1;i≤k+N_i;i++)
    S_1(k,i);
  endfor
endfor
```

Fig. 1　Perfect nested loop in stencil computations[11-12].

图 1　Stencil 计算的完美嵌套循环[11-12]

固定分块是指分块大小在编译时期是固定常量的分块方法,这类方法需要通过不断的重新编译代码来改变或调试分块大小.

1.1　基于 cache 行为分析的方法

cache 行为分析是对带循环的数值计算程序用数学方法进行分析,确定 cache 执行性能的技术.这类方法首先对循环体进行数据依赖分析[6],利用循环变换(如互换 interchange、偏斜 skewing 等)、幺模(unimodular)变换等技术[22]改变循环结构消除

依赖关系,采用 strip-mining[23] 使得原来循环深度增加,然后对变化后的迭代空间进行分块,根据迭代空间数组引用的足迹(footprint)[7] 调整分块大小,使得原本较大的工作集划分成若干满足分级存储容量的子集,充分利用数据重用性,减少 cache 的冲突干扰和容量失效,提高程序的局部性和分级存储体系的利用率.

文献[5]对迭代空间进行分块的合法性进行了介绍.文献[6]提出的数据重用性和局部性数学形式化方法在 SUIF[24] 编译器上得到成功应用,有效地提高了数值计算程序的执行性能.文献[7]表明问题规模和循环基址的轻微变化会导致预测的 cache 性能出现极大的差异.为了满足对 cache 数据访问准确估算的需求,文献[25]建立了一套精确的 cache 失效方程模型来分析 cache 行为.另外,基于 cache 行为分析的分块方法在多处理器上具有良好的并行性[25-27],启发式技术也得到应用[28].

由于 cache 行为分析技术要求非常详细的数据映射和引用模式,预测的性能不够稳定,因此很难精准估算 cache 的干扰行为.这种方法主要适用于完美循环的数值计算,不能有效解决诸多广泛应用的计算类型.

1.2 扫描多面体技术

当分块大小是固定常数时,分块后的迭代空间是一个多面体[8].由于这一理论得到证明,生成扫描多面体的循环技术得到广泛研究和应用[9,24,29-32].

一个完美嵌套循环的迭代空间是有限凸多面体 $\mathbb{Z}^N$,N 是嵌套循环的层数,也是迭代空间的维数.迭代空间的每一个元素是循环体的一次迭代,由迭代向量的值来确定,称为凸多面体的一个点.根据多面体模型可以精确描述点和点之间的依赖关系[29,33],利用线性代数和线性规划的机制进行相应的变换,改变多面体扫描顺序,从而生成能够充分开发 cache 性能的分块代码.多面体模型适用于数据访问函数和循环边界是循环内下标变量和参数的仿射组合的嵌套循环.

文献[9]提出一种经典的扫描单一多面体技术,对原始迭代空间的约束不等式系统采用傅里叶-莫茨消除法(Fourier-Motzkin elimination, FME),计算出仿射变换后新的循环边界.FME 是一种用来消除线性不等式组中变量的数学方法,消除线性不等式组中的变量集后,得到的新不等式组和原不等式组具有相同的解.对包含 n 个 r 元不等式$(x_1,\cdots,x_r)$的循环变量不等式系统,如果要消除循环变量 x_r,根据变量 x_r 的系数符号可以分为 3 种情况:1)系数为正时,$x_r \geqslant \boldsymbol{A}_i(x_1,\cdots,x_{r-1})(1\leqslant i\leqslant n_{\boldsymbol{A}})$,$\boldsymbol{A}$ 为系数矩阵,$n_{\boldsymbol{A}}$ 为该种不等式的数目;2)系数为负时,$x_r \leqslant \boldsymbol{B}_j(x_1,\cdots,x_{r-1})(1\leqslant j\leqslant n_{\boldsymbol{B}})$,$\boldsymbol{B}$ 为系数矩阵,$n_{\boldsymbol{B}}$ 为该种不等式的数目;3)系数为空时,没有对应的项,用 ϕ 表示.原始系统则可等价转换成式(1):

$$\max(\boldsymbol{A}_1(x_1,\cdots,x_{r-1}),\cdots,\boldsymbol{A}_{n_{\boldsymbol{A}}}(x_1,\cdots,x_{r-1})) \leqslant x_r \leqslant \min(\boldsymbol{B}_1(x_1,\cdots,x_{r-1}),\cdots,\boldsymbol{B}_{n_{\boldsymbol{B}}}(x_1,\cdots,x_{r-1})) \wedge \phi. \tag{1}$$

消除式(1)中的 x_r,不等式部分可以等价表示为 $n_{\boldsymbol{A}}\times n_{\boldsymbol{B}}$ 个不等式,如式(2)所示:

$$\boldsymbol{A}_i(x_1,\cdots,x_{r-1}) \leqslant \boldsymbol{B}_j(x_1,\cdots,x_{r-1}),\quad 1\leqslant i\leqslant n_{\boldsymbol{A}},1\leqslant j\leqslant n_{\boldsymbol{B}}. \tag{2}$$

原不等式系统通过消除循环变量 x_r 得到一个包含了$(n-n_{\boldsymbol{A}}-n_{\boldsymbol{B}}+n_{\boldsymbol{A}}n_{\boldsymbol{B}})$个不等式的等价不等式组.

采用这种固定分块方法生成的代码如图 2 所示.由于 FME 可以计算任意顺序的循环变量的边界条件,对绝大部分循环变换具有一般性,这种方法得到广泛的研究.文献[30]通过扩展文献[9]的方法,并运用一系列精致的优化技术扫描联合多面体,解决了为仿射循环生成通用代码的问题,利用该方法开发了应用广泛的 Omega 库[34],而 SUIF[24] 也包含了类似的算法.尽管如此,FME 的复杂度是循环维数的双指数函数,导致基于 FME 的扫描多面体方法效率较低.

```
for (kT=0;kT≤⌊(N_k/2)⌋;kT++)
  for (iT=max(1,kT);iT≤min(⌊(2×kT+N_i+1)/2⌋,
    ⌊N_k+N_i/2⌋);iT++)
    for (k=max(max(1,2×kT),2×iT−N_i);
      k≤min(min(2×kT+1,2×iT),N_k);k++)
      for (i=max(2×iT,k+1);
        i≤min(2×iT+1,k+N_i);i++)
        S_1(k,i);
      endfor
    endfor
  endfor
endfor
```

Fig. 2 Tiled loops generated for fixed tile size using the classic polyhedral scheme[11-12].

图 2 经典多面体方法生成的固定循环分块[11-12]

为了解决 FME 复杂度问题,文献[35]提出一种双表征算法,递归建立联合多面体的抽象语法树(abstract syntax tree, AST),用 AST 生成代码.该算法虽然能够保证避免冗余控制,但有着代码量爆炸、较高复杂度和控制开销大的局限性.文献[31,36]对该算法进行改进,减小了生成代码长度和代码生

成的处理时间.这些算法在开源工具 CLooG[31,37]中得到实现.CLooG 利用 PolyLib 库[38]进行核心的多面体计算,采用比 FME 更加高效的 Chernikova 算法[39].WRAP-IT 项目[32,40]则使用了由 CLooG 衍生的 WLoog.

由于扫描多面体技术的发展,大批基于多面体模型的代码生成器不断涌现.文献[31]比较了 3 种典型代表的代码生成器 CodeGen[41],LoopGen[35]和 ClooG[31,37],在 Intel Pentium Ⅲ架构上对 SPECfp2000 和 PerfectClub 基准程序进行测试,比较分析了 3 种代码生成器在代码生成时间和相对原始代码的生成代码扩展量上的性能,如图 3 所示.CLooG 和 CodeGen,LoopGen 相比,平均代码生成时间加速比分别为 4.05 和 2.57.同时 CLooG 生成的代码量相对原始代码增加最低,平均增加量为 6%.由于 LoopGen 以代码量为代价清除了更多的控制开销,在代码生成时间上较 CodeGen 有明显改善,但是生成代码量比 CodeGen 平均增加 38%.

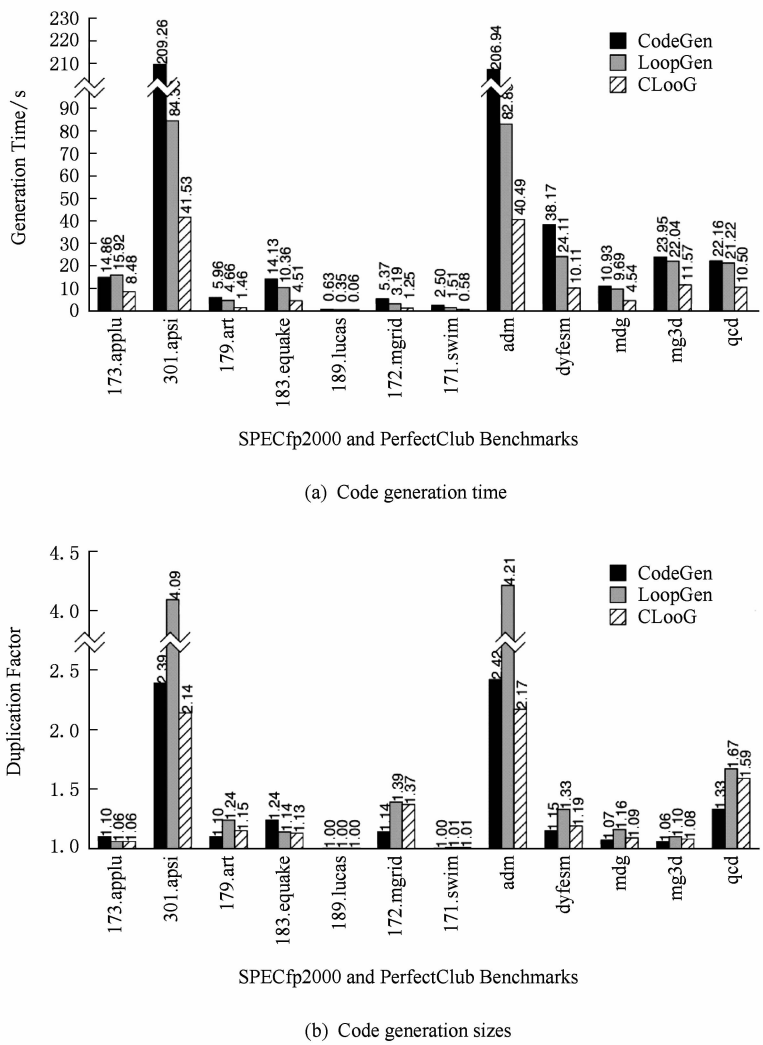


Fig. 3　Comparison of 3 tiled code generators[31].

图 3　3 种分块代码生成器的性能比较[31]

这些代码生成器也有一定的局限性,如只适用于计算密集型的带完美嵌套循环的程序,最坏情况下 CLooG 的复杂度仍然是双指数函数.此外,这些代码生成器都需要人为参与手动开发.鉴于 CLooG 的高效性,文献[19]设计实现了一个全自动多面体模型优化系统,该系统提出基于 statement-wise 的仿射变换提供给 CLooG 作为散射函数,使其能够扫描任意循环代码(包含不完美嵌套循环的代码)的联合多面体,并能完全自动生成分块代码,为带嵌套循环体的程序提供源到源(source-to-source)的优化.该系统的实现框架如图 4 所示.

多面体模型架构下,串行程序的并行化代码自动生成主要分为 3 个步骤:1)输入程序数据依赖关系的静态分析;2)对抽象多面体模型的循环变换;3)

变换后有效的循环代码生成. 文献[19]的实验表明多面体系统不仅能为不完美嵌套循环的 stencil 计算自动生成分块代码,而且在多核架构上也能获得较高的加速比. 但是该系统只适用固定分块的情况.

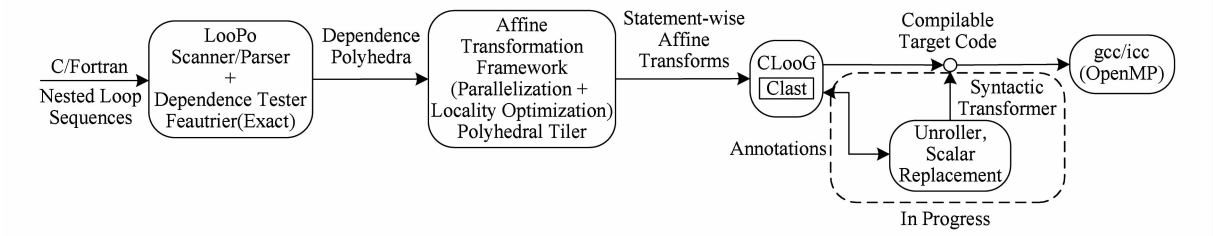


Fig. 4 The fully automatic polyhedral source-to-sourc program optimization system[19].

图 4 全自动源到源转换的多面体程序优化系统[19]

1.3 嵌套循环的分解方法

文献[14]提出一种全新的分块思路,其主要贡献在于把分块代码生成问题分解成 2 个子问题:1)枚举所有分块. 计算利用非幺模(nonunimodular)变换后迭代空间新的循环边界,精确地枚举所有分块的源点(origin),即分块内词典顺序访问的第 1 个点,它可以不属于迭代空间. 2)扫描每个分块内的点. 对第 2 个子问题,把最初的平行六面体分块转化成矩形分块,利用非幺模变换矩阵及其埃尔米特范式(Hermite normal form)生成有效的分块代码,扫描所有块内点.

这种分解的方法有效降低了应用 FME 时约束不等式的数量,使得分块代码生成的编译时间和执行时间大幅下降. 文献[14]把该方法与文献[9,34]提出的方法在时间和分块开销上进行性能比较,实验验证了该分解方法的优势. 尽管这种方法只适用于固定分块大小的情况,但是为后来的研究提供了颇具前景的方向.

1.4 固定分块技术小结

由第 1 节分析可知,固定分块技术可以避免 cache 失效行为,改善数据局部性,有效利用层次存储架构,实现程序中的循环体并行执行,但是这类方法仍具有明显的局限性:1)基于 cache 行为分析的方法难以精确模拟 cache 行为对循环程序的性能影响,同时存在分块效率和适用范围的问题;2)基于多面体扫描的方法虽然可以应用到任意循环程序,但用 FME 计算迭代空间边界的算法具有较高的算法复杂度,分块代码的生成和执行时间较长;3)虽然循环分解的思想有效地改善了之前方法的空间、时间复杂度及分块开销,但是这种方法具有所有固定分块方法共同的局限性,即分块因子是固定常数,导致分块代码无法动态调整,每次改变分块大小都需要重新编译代码,而相关研究表明找到最优分块因子较为困难[42],这就意味着固定分块方法需要一直不停地重新编译代码,使得该方法低效耗时从而很难在实际需求中得到高效的应用.

2 参数化分块技术

参数化分块是指应用分块变换时不固定分块大小,把分块大小当作符号参数提供给多面体系统,只在运行时才会被确定下来. 参数化分块不仅可以适用于迭代编译器、自动调优器(auto-tuners)[17]和一些代码生成器[37],而且支持运行时反馈和程序动态适应. 与固定分块方法需要不断重新编译代码不同,参数化分块方法对所有的分块大小只需一次编译生成分块代码. 目前,一些源到源转换的分块变换系统,如 TLoG[11,43], HiTLoG[44-45], PrimeTile[46] 和 PLuTo[19,47]已经发布使用.

2.1 边界框方法

几何学上的边界框(bounding box)方法可以作为参数化分块代码生成问题的一种简单解决方案[11]:1)生成分块循环(tile-loops)计算迭代空间的边界框内每个分块的源点;2)利用点循环(point-loops)检查每个分块是否包含有效的迭代空间内的点. 图 5 显示使用边界框方法生成的分块嵌套循环:

```
for (kT=1;kT≤N_k;kT+=S_k)
  for (iT=2;iT≤N_i+N_k;iT+=S_i)
    for (k=max(kT,1);k≤min(kT+S_k−1,N_k);k++)
      for (i=max(iT,k+1);i≤min(iT+S_i−1,k+N_i);i++)
        S_1(k,i);
      endfor
    endfor
  endfor
endfor
```

Fig. 5 Tiled loops generated using the bounding box scheme[11-12].

图 5 采用边界框方法生成分块的循环代码[11-12]

边界框方法把分块问题分解成2组循环生成的问题:1组循环边界把循环变量约束在分块边界框内的点;另外1组则把循环变量约束在迭代空间内的点.

图1示例嵌套循环的二维平行四边形迭代空间及其边界框如图6所示,其中 $N_i=N_k=6$,循环边界是方形分块和迭代空间的交集.通过观察可以发现分块有3种类型:1)完整分块(full tile),分块内的点全部属于迭代空间;2)局部分块(partial tile),分块内的部分点属于迭代空间;3)空分块(empty tile),分块和迭代空间的交集为空.边界框方法和文献[14]提供的分解思路类似,而且分块大小无需在循环生成时固定,可以作为符号参数使用,这些特性能实现参数化循环分块.尽管边界框方法适合很多应用,但是由于分块循环可能会计算很多空分块,导致该方法效率不高,特别是对非矩形的迭代空间,分块后的空分块计算开销可能高达总开销的50%[11].同时,根据迭代空间约束条件计算边界框可能会导致指数级复杂度.

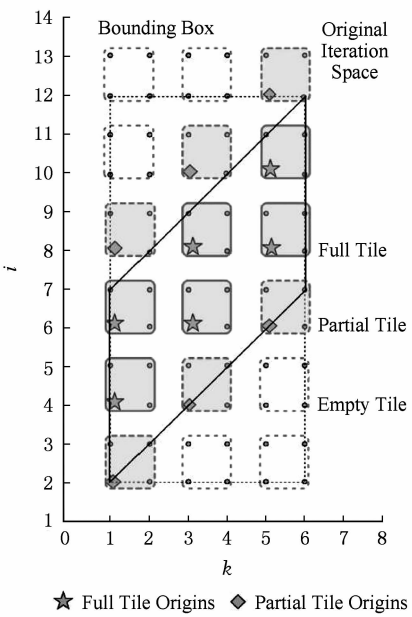


Fig. 6　The bounding box of the 2D stencil iteration space[11-12].

图6　二维stencil计算迭代空间的边界框[11-12]

2.2　基于SFME的方法

FME算法中重要的一步是判断循环变量系数的正负符号.当系数是固定常数时对线性约束条件的判断就简单明确,但是对于参数化分块,变量系数是符号参数时将难以判断正负,只能通过分多种情况来进行讨论,这显然不利于问题的求解.文献[15]提出的SFME算法利用约束不等式集的双线性性质,通过对异号的参数系数乘以比例因子作加减消元的操作消除变量.由该算法生成的分块循环代码如图7所示:

```
for (kT=0;kT≤⌊(N_k−1)/S_k⌋;kT++)
  for (iT=max(0,⌊(S_k×kT−S_i+1)/S_i⌋);
    iT≤min(⌊(N_i+N_k−2)/S_i⌋,⌊(S_k×kT+S_k+N_i−2)/
      S_i⌋);iT++)
    for (k=max((S_k×kT+1),(S_i×iT−N_i+2));
      k≤min(N_k,S_i×iT+S_i,S_k×kT+S_k);k++)
      for (i=max(k+1,S_i×iT+2);
        i≤min(N_i+k,S_i×iT+S_i+1);i++)
        S_1(k,i);
      endfor
    endfor
  endfor
endfor
```

Fig. 7　Tiled loops generated using the SFME scheme[11-12].

图7　采用SFME生成分块的循环代码[11-12]

SFME方法可以生成精准的分块循环,为分块代码生成面临的绝大部分问题提供了完整的解决方案.但是该算法具有非常高的计算复杂度,而且算法的每一步都需要进行符号运算,导致运行时的低效和代码生成时间过长[11],而且SFME算法不支持并行分块.文献[48]提出一种松弛的SFME算法,在利用SFME进行消除变量的过程中,当遇到变量的参数多项式系数无法确定正负号时,直接用变量的上下边界值 $x_r^{\min}$ 和 $x_r^{\max}$ 进行替换,然后利用FME处理得到松弛的边界不等式组.同时,该算法对分块计算实现了波阵面的并行性.

2.3　Outset集方法

构建Outset集是文献[11]方法的关键.Outset集是包含所有非空分块源点的集合.Outset集与文献[14]构建的分块源点空间(tile origin space)类似,但有2个明显的不同点:1)Outset集包含的参数中分块大小可以是符号参数也可以是固定常数;2)Outset集可以供应给任何能够扫描多面体的代码生成器.

Outset集由分块大小进行参数化,具有边界框方法的全部优点,同时只计算非常少的空分块.图8显示了二维stencil计算的Outset集,其中分块大小为2×2,可以看出Outset集只包含一个空分块,远小于边界框要计算的空分块数量.

从几何图形上来看,Outset集的构建可以看作是对迭代空间中循环下界进行超平面移位.以图8

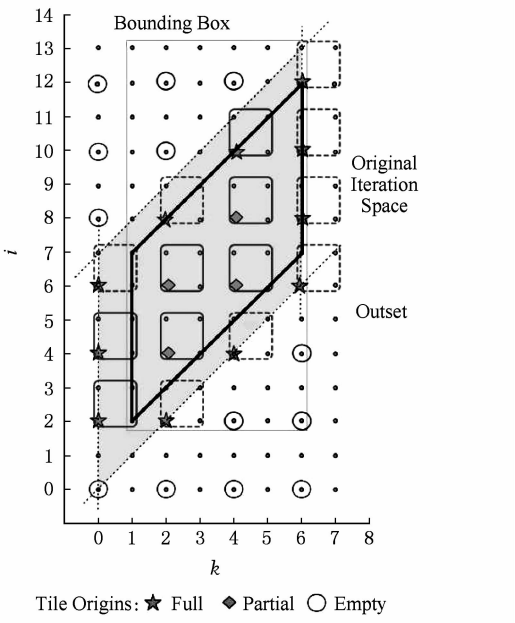


Fig. 8 The Outset and bounding box of the 2D stencil iteration space[11-12].

图 8 二维 stencil 计算迭代空间的 Outset 集和边界框[11-12]

的二维迭代空间为例,对左边垂直边界和上下 2 条 45°边界进行向外移位,这样得到的 Outset 集将会包含所有与迭代空间有非空交集的分块的源点. 其中左边垂直边界和顶部 45°边界组成了循环变量 k 的下界,底部 45°边界形成了循环变量 i 的下界. 对扫描 Outset 集的循环进行一定的处理得到分块循环,代码如图 9 所示:

```
/* shifte first iteration of the kT-loop to a tile origin */
kTLB=-S_k+2;kTLB=⌈kTLB/S_k⌉×S_k;
/* scans dimension k of Outset with stride S_k */
for (kT=kTLB;kT≤N_k;kT+=S_k)
  /* shift first iteration of the iT-loop to a tile origin */
  iTLB=kT-S_i+2;iTLB=⌈iTLB/S_i⌉×S_i;
  /* scans dimesion i of Outset with stride S_i */
  for (iT=iTLB;iT≤kT+N_i+S_k-1;iT+=S_i)
    for (k=max(kT,1);k≤min(kT+S_k-1,N_k);k++)
      for (i=max(iT,K+1);i≤min(iT+S_i-1,k+N_i);i++)
        S_1(k,i);
      endfor
    endfor
  endfor
endfor
```

Fig. 9 Parameterized tiled loops generated using the Outset[12].

图 9 采用 Outset 集生成参数化分块循环[12]

为了保证点循环最终扫描的点都是有效的,其边界由分块边界和迭代空间边界组成,但是对于原始迭代空间中的完整分块,迭代边界成为多余,这将造成过多的资源开销. 文献[9]建议对完整分块和局部分块分别处理来优化分块代码,文献[49]通过拆分索引集来划分迭代空间. Inset 集[20,44]则是由所有完整分块的源点构建而成,它也是一个多面体,可以应用于扫描多面体的代码生成器. 一旦构建好 Inset 集就可以通过判断分块源点是否属于 Inset 集,或从 Outset 集中划分 Inset 集,这 2 种方法有效地区分所有的完整分块和局部分块,消除完整分块的多余边界约束条件,从而减少了循环的计算开销.

Outset 集方法具有边界框算法和循环结构分解[14]的简单思路,生成的代码具有固定分块方法的代码质量,同时受益于参数化分块代码生成的优势,使其成为目前最为有效的参数化循环分块方法之一. 文献[11]把参数化 Outset 集方法和边界框方法以及 2 种经典的固定分块方法进行比较,结果如表 1 所示. 其中 fClassic, fDecom, pBbox, pOutset 分别表示固定多面体方法、固定循环分解方法、参数化边界框方法和参数化 Outset 集方法. LUD 是 LU 分解,SSYRK 是对称矩阵秩 k 校正,STRMM 是三角矩阵乘法,3D Stencil 是高斯赛德尔类型的三维 stencil 计算,前 3 个测试程序是 BLAS 库的线性代数程序,第 4 个是 stencil 计算程序. 实验结果表明 Outset 集方法和固定分解方法、边界框方法具有相似的代码生成效率和生成代码质量,固定多面体方法虽然在二维分块上具有明显的优势,但该方法具有可扩展性的问题,随着维数增加其代码生成效率会大大降低.

Table 1 Tiled Loop Generation Time of the Four Fixed and Parametric Methods on Four Benchmarks[11-12]

表 1 4 种方法在分块循环生成时间上的比较[11-12] ms

Method	LUD	SSYRK	STRMM	3D Stencil
fClassic	32.4	28.6	29.0	26.0
fDecom	55.2	51.0	50.4	45.0
pBbox	53.5	53.2	51.2	54.0
pOutset	52.0	53.8	52.1	54.1

另外,文献[12]在 Outset 集方法的基础上发展了一套完整的参数化循环分块的形式理论.

2.4 参数化分块技术小结

第 2 节介绍了 3 种参数化循环分块方法. 边界框方法提供了一种简单的参数化思想,但对非矩形迭代空间会产生较低的分块效率,使其适用范围受限. 与边界框方法不同,基于 SFME 的方法运用分块迭

代空间复杂的形式化特征,提供计算非空分块源点的精准算法,使得该方法计算复杂度过高,代码生成时间漫长.Outset集方法则在代码生成效率和生成代码质量上达到了与经典固定分块方法可比的性能.这归功于Outset集的重要特性:不仅可以由符号参数和固定常数分别构建,也适合于固定常数和符号参数混合的情况;Outset集在时间和空间复杂度上是循环边界数目的线性关系,使其构建较为高效;Outset集是一个多面体,可以利用现有的代码生成器实现参数化循环分块.另外,在Outset集的基础上本文介绍了一种利用Inset集消除多余边界条件的优化方法.

文献[11]介绍的Outset集方法仅适用于可以进行矩形分块的单一的完美嵌套循环,但文献[44,46]对Outset集方法进行扩展,使其可以应用于不完美循环嵌套和多级分块,详细内容将在第4节介绍.由于参数化分块技术的研究工作开展较晚,针对参数化循环分块方法的算法复杂度、代码生成效率、应用范围、多级存储架构映射、分块并行性等方面仍需要深入的研究.

3　最优分块大小选择方法

开发数据局部性是实现软硬件性能高效开发的关键.分块大小的选择对性能有着重要的影响.分块大小选择不仅是分块尺寸的选择,还是分块形状的选择.分块大小受到cache行大小、关联度、数据替换策略、硬件存储架构等多种因素的共同影响.分块过大时数据尚未充分利用cache或寄存器空间进行重用就有可能被置换出去导致不命中,而分块过小又会造成较大的成本开销从而淹没带来的性能优势.最优和最差的分块可能导致整体性能数10倍的差异.分块形状不仅和层次cache架构特征的映射相关,而且影响最优分块大小选择的难度,每一维循环采用不同大小的矩形分块比每一维采用相同大小的正方形分块具有更高的复杂度.随着处理器分级存储体系复杂度和深度的增长,有效选择最优分块大小成为一个更具有挑战性的问题.

3.1　基于分析的方法

早期研究者通过对程序循环嵌套和硬件存储特征进行静态分析,为编译器选择合适的分块大小[7,50-51].这类方法主要用来解决容量失效、自干扰失效、交叉干扰失效导致的cache不命中和局部性优化问题.

文献[6]首次对重用和局部性提出了精确的数学定义,并据此开发一系列程序变换,考虑cache组相联导致的数据冲突,提出一种LRW算法[7]来确定最大正方形分块大小,以提高数据局部性.文献[52]设计了一个估算不同高速缓存线(distinct cache lines,DL)模型.对一个包含在完美嵌套循环内的m维矩阵$\boldsymbol{A}$,下标变量为$i_1,\cdots,i_n$,$f_j(i_1,\cdots,i_n)$是一个仿射函数,可表示为$\boldsymbol{A}[f_m(i_1,\cdots,i_n)]\cdots[f_1(i_1,\cdots,i_n)]$.矩阵$\boldsymbol{A}$单维下标表达式$f(i_1,\cdots,i_n)$单一数组引用会访问到不同高速缓存线数目的上限如式(3)所示:

$$DL(f)\leqslant\min\left(\frac{(f^{hi}-f^{lo})}{g}+1,\left\lceil\frac{(f^{hi}-f^{lo})}{L}\right\rceil+1\right),\tag{3}$$

其中,g是下标表达式f系数的最大公约数,L是以数组元素大小为单位的高速缓存线尺寸,f^{hi}和f^{lo}分别是下标表达式f在整个循环过程中的最大值和最小值.由单维推广至多维数组引用$\boldsymbol{A}(f_1,\cdots,f_m)$,访问DL上限的估值如式(4)所示:

$$DL(f_1,\cdots,f_m)=DL(f_1)\times\prod_{j=2}^{m}\left(\frac{(f_j^{hi}-f_j^{lo})}{g_j}+1\right).\tag{4}$$

这样根据估算循环访问不同的高速缓存线的数目,建立更为精确的cache失效估算模型指导用户程序变换,实现较高的cache命中性能.另外一些研究者提出针对降低自干扰失效的分块大小选择方法[7,18].文献[25]提出cache失效方程,选择合适的分块大小来消除自干扰,同时计算适应cache容量的最大分块.文献[50]的分块方法则不仅可以把容量失效和交叉干扰失效减少到最小,而且能够避免自干扰失效的发生,提高cache利用率.另外,数组填充(padding)作为一种有效的数据布局优化技术,与上述一些算法合并使用,可以避免计算病态矩阵时循环分块技术的不稳定性[53].

虽然基于静态分析的分块大小选择方法得到广泛的研究,在减少干扰失效和提高数据局部性上有较好表现,但是这类方法在实际应用中存在一定的缺陷:1)理论分析不能完全反应实际存储的复杂过程,影响程序分块性能的潜在因素较多,导致实际最优分块的性能和分析方法选择分块的性能差距较大;2)分析建模过程复杂,成本较高,对硬件和程序布局严重依赖,不具有通用性.

3.2　基于经验搜索的方法

基于经验搜索的分块选择方法是把循环嵌套看作一个黑盒子,根据经验选择一系列不同分块大小

组合,在目标机器上对这些分块组合进行自动调优(auto-tuning),并从这些较好的分块大小组合中选择执行性能最优的分块大小[54-57].

自动调优线性代数软件(automatically tuned linear algebra software, ATLAS)系统[16,57]可以为不同规模的问题进行经验调优,找到具有最优性能的分块大小,调优只需在 BLAS 库[13]安装时进行一次就能完成.尽管 ATLAS 系统在性能优化上十分成功,但它只适用解决计算密集型的线性代数方程,对一般代码(如 stencil 计算)则不适用.经验调优方法面临的最大困难是对多层循环进行分块时,要遍历庞大的搜索空间,这是因为随着循环维数增多,每层循环分块因子不同的组合搜索空间将爆炸式的增长,导致这种方法时间成本过高.ATLAS 系统为了降低时间和空间复杂度,采用一个简单的分块选择模型,每一维的候选分块大小不超过 L1 cache 大小的平方根,并且只采用正方形分块因子.然而一些实验表明,线性代数和 stencil 计算的实际最优分块因子是非正方形分块[58-59],也就是说每一维循环的分块大小都不一定相同.这种非正方形分块相对于正方形分块因子在代码执行上可以获得更好的性能.

3.3 基于分析和经验混合的方法

由于经验调优在分块大小空间上进行全局搜索,这种低效的方法难以满足实际应用需求,目前结合分析模型的方法、特征化较好的分块大小、减少分块大小的搜索空间成为研究的热点[60-62].

文献[62]利用 2 个分析模型——DL 模型[52]和 ML(minimum working set lines)模型——来约束分块大小的搜索空间.ML 模型为分块内数据重用假设最优的 cache 块替换策略和 TLB 容量约束,它是一个为估算分块大小提供上界的积极模型.DL 模型则是提供下界的保守模型,它忽略分块内 cache 块替换策略.DL/ML 为单层分块的容量约束条件,如式(5)所示:

$$\begin{aligned} DL(T_1,T_2,\cdots,T_n) &\geqslant CS_1,\\ ML(T_2,T_3,\cdots,T_n) &\leqslant CS_1,\\ DL(T_1,T_2,\cdots,T_n) &\leqslant CS_k, \end{aligned} \tag{5}$$

第 1 个不等式表示 L1 cache 中块内数据重用的下界,第 2 个不等式表示 L1 cache 中块内数据重用的上界,第 3 个不等式表示 k 级 cache 中分块重用的上界.其中,CS_k 表示 k 级 cache 的高速缓存线数目或 TLB 目录的数目.基于 DL/ML 模型限制搜索空间的算法在编译器架构上已经得到实现,如图 10 所示.编译器工具提取输入循环的数组下标表达式,分析数组的依赖关系向量,只需这 2 种程序相关的数据计算 DL/ML 方程,最后加上机器相关信息,如不同 cache 层容量大小和 cache 行大小,就能得到一个候选分块大小的有界搜索空间,相比原始候选分块搜索空间得到了极大的缩小,因此降低了巨大的调试开销.

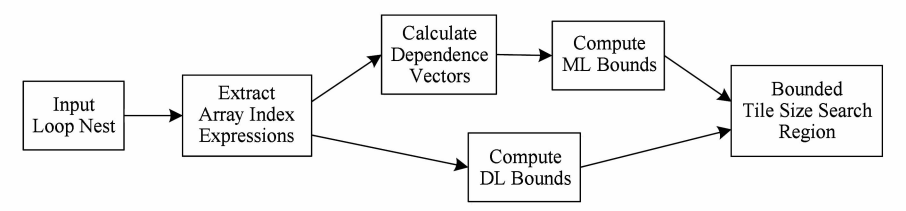


Fig. 10 Implementation of DL/ML bounding in compiler framework[62].

图 10 DL/ML 限界方法在编译器架构上的实现[62]

文献[62]在 3 种不同机器架构上,选取了一些经典的线性代数和 stencil 计算作为测试程序,采用 DL/ML 分析方法优化搜索空间并进行经验调优,并对 DL 和正方形分块算法进行了性能比较.表 2 显示了所有测试程序在不分块、DL 优化下最优分块大小、最优正方形分块大小、DL/ML 优化下最优分块大小等情况的执行时间和性能比较.表 2 中第 1 列测试程序名称后缀"-N","-P","-X"分别表示测试程序在 Nehalem,Power 7,Xeon 这 3 种机器架构上测试.最后 2 列表示测试程序采用 DL/ML 方法和采用 DL 方法,正方形分块方法性能比较的加速比.通过对比实验发现,DL/ML 技术使搜索空间缩小了 45～11 879 倍,优化后的搜索空间包含了 95%以上最佳性能的分块大小.从表 2 的实验数据还可以看出,非正方形分块较正方形分块具有更好的性能.

这种方法虽然可以适用于不完美的循环嵌套,但是目前只能处理单级分块,也不能处理循环迭代访问为非均匀步长的情况.

Active Harmony 项目[55]采用了其他的算法(如 Nelder-Mead 单纯型算法)来减小搜索空间.同时,有研究认为随着自动调优器能力的增强,将面临的一个重要挑战是如何在不同自动调优器中进行可扩

展的搜索[56].另外一些混合的方法采用了启发式思想来预测复杂体系结构对性能的影响[59,63-65].文献[63]为优化矩阵乘法建立一个学习分类器系统,该系统根据目标平台特点确定分块的级数和每级的分块大小.文献[64]为程序提取有效的特征值进行神经网络训练,在此基础上构建一种自动选择最优分块大小的架构.文献[65]则侧重于用机器学习的方法对分块大小的差异建立一个精准的性能分布模型,从而降低随机搜索分块空间时的性能变化.而文献[54]采用不同的技术,如遗传算法和模拟退火算法管理搜索空间的大小.这些方法有可能得到较好性能的分块,更适用于一些特定的程序或内核结构.

Table 2　1-Level Tiling Results with 4 Tile Size Selections[62]

表 2　4 种分块大小选择方法的一级分块结果[62]

Benchmark	United Time/s	DL		Best Square Tile		Best DL/ML		Speedup of DL/ML	
		Tile Size	Time/s	Tile Size	Time/s	Tile Size	Time/s	Vs DL	Vs Sq
MatMult-N	33.25	(40,40,30)	16.40	(80,80,80)	17.27	(150,30,80)	13.48	1.22	1.28
MatMult-P	25.46	(50,30,20)	13.90	(80,80,80)	12.28	(90,10,120)	10.60	1.31	1.15
MatMult-X	153.66	(40,40,30)	29.51	(50,50,50)	23.98	(100,20,120)	18.35	1.60	1.31
DSYRK-N	25.39	(30,40,40)	15.47	(80,80,80)	15.54	(30,30,90)	12.50	1.23	1.24
DSYRK-P	23.32	(40,30,30)	15.10	(300,300,300)	10.86	(60,10,1000)	9.16	1.64	1.19
DSYRK-X	84.89	(30,40,40)	26.08	(120,120,120)	25.44	(100,30,80)	18.19	1.43	1.40
DTRMM-N	142.42	(40,40,30)	19.20	(60,60,60)	18.87	(150,30,60)	18.20	1.05	1.04
DTRMM-P	62.74	(30,50,20)	14.60	(60,60,60)	13.06	(600,30,32)	11.96	1.22	1.09
DTRMM-X	114.70	(40,40,30)	28.98	(120,120,120)	29.13	(30,10,120)	23.49	1.23	1.24
2D-Jacobi-N	2.43	(10,40,10)	2.60	(50,50,50)	2.24	(10,8,150)	2.16	1.20	1.04
2D-Jacobi-P	2.10	(10,40,10)	2.09	(10,50,50)	1.31	(10,40,120)	1.19	1.76	1.10
2D-Jacobi-X	8.75	(10,40,10)	2.77	(10,8,8)	2.81	(50,40,20)	2.54	1.09	1.11
2D-FDTD-N	15.35	(10,60,8)	2.41	(50,8,8)	2.35	(50,50,8)	2.26	1.07	1.04
2D-FDTD-P	9.56	(10,40,1)	6.90	(50,70,70)	2.11	(40,70,40)	2.09	3.30	1.01
2D-FDTD-X	16.42	(10,60,8)	4.47	(100,40,40)	4.22	(50,100,8)	4.01	1.11	1.05

3.4　分块大小选择方法小结

第 3 节介绍了分块大小选择技术,利用以上各种方法可以使循环分块技术获得更好的性能,每种方法都有各自的特点和不足.基于分析的方法存在有效性的问题,基于经验调试的方法存在时间复杂度高和适用范围的问题,而混合方法存在如何平衡算法精确度和成本开销的问题.

分析目前的研究现状可以得出如下结论:1)3 种基本方法在成本开销、适用范围、性能上各有优势,但是尚无一种技术同时具有这些优势,如何在一种方法上同时实现这些特点将是一个研究难点;2)多级循环分块技术是目前和将来的研究主流,而现在大部分研究是针对单级分块大小选择进行的,如何打破当前技术局限,发展面向任意循环代码的多级分块大小选择方法,还需要进一步研究;3)相对于前 2 种方法,基于混合的方法在时间空间复杂度、性能上都有明显的提升,新的理论优化技术和有效的模型不断涌现,如何引用新的技术更好地优化分块搜索空间和选择各级分块大小仍需要深入地研究.

4　循环分块技术的扩展研究

现代计算机存储架构的层次不断加深,并行处理单元的数量也在快速增长.为了充分开发利用硬件高效的计算能力,循环分块技术不仅需要向多级发展,而且要求能够提供高并行度和可扩展性,同时保障多核架构下资源利用和成本开销.为了让更多的应用能够享受循环分块带来的性能优势,研究者们对面向不完美嵌套循环的应用和任意带循环体程序提出了一些解决方案.

4.1　多级分块和分块并行性研究

分块技术可以有效开发层次存储架构的机器资源.两级分块技术不仅成功应用于线性代数程序的 ATLAS[16-17] 和 PHiPAC[66],有效开发 cache 和寄存

器性能,实现指令级并行性[20],而且在 stencil 计算上也获得了高性能的实现[67]. 随着多核架构和存储深度的发展,两级 cache 分块或寄存器分块已经不能满足对更高层次粒度并行性的要求. 多级分块技术已经成为高性能应用的设计模板.

传统的多级分块技术[68]通过逐级分块的方法实现,其复杂性是多级分块代码中循环数目的双指数函数,导致这种方法在处理非矩形迭代空间时的代价过高. 文献[69]提出一种同时多级分块算法,该方法的计算复杂度仅依赖于原始循环的数目,因此可以用于高效的生成任意级数的多级分块代码,但是这种方法受到固定分块的局限.

随着参数化分块方法的开发,文献[44]利用 Outset 集和 Inset 集实现参数化多级分块方法. 基于 Outset 集的多级分块方法,是在对迭代空间进行第 k 级分块的基础上进行 $k+1$ 级分块. 由第 k 级的分块边界和原始迭代空间边界共同组成第 $k+1$ 级分块的原始迭代空间,几何图示如图 11 所示. 由于 Inset 集包含第 k 级的所有完整分块,对完整分块进行第 $k+1$ 级分块时,其迭代空间就是第 k 级完整分块的边界,无需原始迭代空间边界,消除多余的边界条件. 实验表明该多级分块方法在和单级分块方法具有相同开销的情况下,提高了分块代码的性能. 然而,为了简单起见,该方法只适用于分块是正方形的循环分块. 另外一些研究利用循环边界的 AST[70]和线性不等式系统变形[48]的方法把单级分块扩展成多级分块技术.

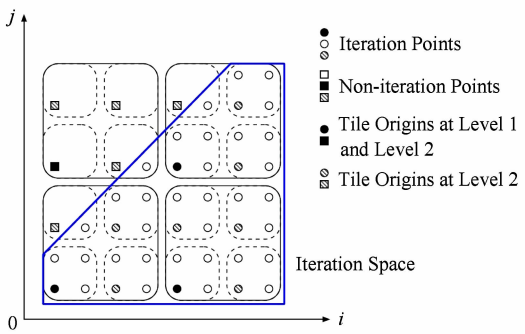


Fig. 11 Multi-level tiling with the Outset and Inset[44].

图 11 采用 Outset 集和 Inset 集的多级分块[44]

循环分块不仅可以增强数据局部性,而且能够实现分块代码的并行性. 文献[10]介绍了为分布式存储架构的并行计算机运用循环分块的方法,减少通信开销和改善并行性,并且为循环变换提供了一套数学理论基础. 文献[48]开发了一个为串行程序自动生成参数化并行分块代码的 PTile 系统,该系统基于松弛 SFME 算法实现了波阵面并行性,生成的并行代码和 PLuTo 系统[19,47]的生成代码相比,在多核架构上有明显的性能提升.

对于多线程编程语言级的分块粗粒度并行,波阵面分块并行[71]能更好地开发程序并行度. 部分线性代数程序的迭代空间本身无需变换就可以进行矩形分块,其并行执行比较简单和直观,本文不作赘述. 接下来以情况较为复杂的 stencil 计算为例进行介绍. stencil 计算中每一个矩阵元素根据该元素本身和其邻居元素的值进行更新,迭代空间进行分块之前必须对时间维度进行 skewing 操作. 分块后根据多面体模型可以分析矩阵数据流,分块边界之间的数据流向描述了分块间的通信,通常分块间的通信聚合起来在该组分块执行完成后和下一组分块执行前的时间内进行,因此这种分块间的数据流拓扑结构使得分块以流水线的形式并发执行. 分块的并发执行以波阵面的样式进展. 如图 12 所示,分块执行从左侧最低处的分块(记为 1)开始,接着同时执行该分块上侧和右侧的邻居分块(记为 2),然后继续同时执行刚刚完成的波阵面上每个分块的邻居分块(按照记号为 3,4,5,…的顺序执行下去),直至全部分块执行完毕. 为了简单理解,图 12 为一维 stencil 计算的分块波阵面示图. 因为流水线分块启动可能会导致负载不均衡的情况出现,文献[72]提供了一种分块并发启动执行的方法,能在高维数据

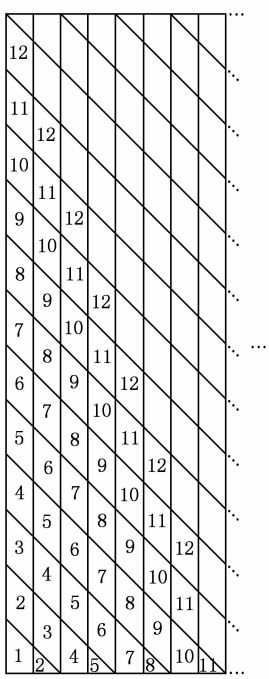


Fig. 12 Wavefront of pipelined tile execution[71].

图 12 分块流水线执行的波阵面[71]

集上缓解这种情况.同时,并行分块代码由分块算法结合多面体代码生成器生成,因此一些研究[70,73-74]基于多面体模型下循环变换的并行优化问题展开.

面对多核异构架构的兴起,GPU 架构上的分块技术也成为研究的热点[75-77].由于 GPU 计算的高效性,已有的技术利用增强数据重用来缓解一些低效的情况,如非均衡的并行性、冗余计算、增加的控制流开销等,高效的并行分块方法亟需得到开发.循环的时间维度重用对 GPU 架构上分块的高效执行十分关键.文献[2]利用 PPCG 多面体代码生成器开发一种为 GPU 生成拆分分块代码的方法.对于最外层循环为时间变量的循环体进行分块时,通常要先采用 skewing 等循环变换,这些操作会抑制分块间的并行度,该文提出拆分分块的方法避免作这类限制分块并行度的变换.根据依赖关系向量计算拆分下标索引集来执行分块,分块被细分成一系列不规则四边形分块的计算步骤,在避免线程同步操作的同时保持拆分分块的较高并行度,而且无需进行 skewing 操作和冗余计算.文献[78]则提出一种混合的六边形分块技术,对时间维度和一维空间采用六边形的分块,对剩余空间维度采用经典的分块技术.因为六边形分块能够使并行分块执行和时域上分块重用同时进行,因此该方法可以最大程度实现迭代 stencil 计算时域分块的有效性.该方法支持合并全局访存和共享内存/cache 上的数据重用,减少了线程发散度,为 GPU 上的计算提取了高度并行性.寄存器分块技术[20,49]则改善了程序的指令级并行性.

随着硬件架构的高速发展,分块技术也将面临应用数据规模爆炸扩展的挑战.文献[71]进行了循环分块方法可扩展性方面的研究.为了获得更高并行度,该文研究了各种循环分块策略对渐进的有效并行度的影响,并分析发现已有的绝大部分分块方法只能提供弱可扩展性.另一项正在进行的研究对数据集大小的可扩展性提供了一个解决方案[72].内存依赖关系是影响分块正确性和可扩展性的一个重要因素.内存依赖关系可分为真依赖和伪依赖关系,前者保证计算的正确执行顺序,后者由重复使用一个存储单元存入多个值引起,会降低循环变换的自由度.通过代码膨胀可以消除伪依赖关系,但同时带来的巨大内存开销和对寄存器级重用的不利影响,对性能而言都是灾难性的.为了避免伪依赖关系,文献[4]提出一种方法对迭代生命周期之间依赖冲突进行精准的特征描述,让编译器忽略迭代生命周期之间大量的伪依赖关系,发现更多的可分开循环,使得对内存的影响最小,而且避免了矩阵膨胀的开销.这种方法对三地址代码进行分块效果显著.

4.2　多核架构上共享 cache 的研究

由于多核架构下片上共享 cache 系统不同于单核 cache 结构,传统单核环境下的数据局部性优化技术对多核架构上多线程应用程序的开发已经失去性能优势[79].多核架构下多级 cache 系统的性能取决于多线程应用程序的特征和底层的 cache 结构.尽管片上多级 cache 系统有多种不同的形式(连接的核数、cache 拓扑、互连结构等),但各种片上多级 cache 系统都包含有 2 个或者多个核共享的 cache 层.图 13 显示了 2 种多核架构的片上 cache 系统.Intel 的 Harpertown 架构有 2 层 cache,4 个核共享 L2 cache.Dunnington 架构有 3 层 cache,2 个核共享 L2 cache,6 个核共享 L3 cache.无论哪种多核架构,最后一级 cache 系统的性能都至关重要,因为该级 cache 的不命中将导致片外访存,这对性能和功耗的损害都十分严重.

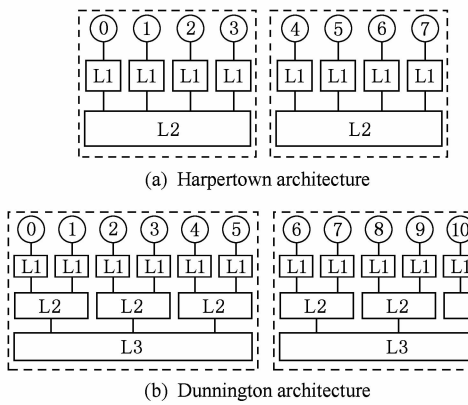


Fig. 13　Shared cache systems on 2 multicore architectures[79].

图 13　2 种多核架构的片上共享 cache 系统[79]

程序是否具有较好的局部性特征体现在数据重用时是否命中 cache 或寄存器,重用距离(reuse distance)用于描述这一重要特征.重用距离越短,cache 中相关数据重用的机会越大.以前提出的代码和数据优化技术的主要目标之一可以总结为减少重用距离.不同于单核环境,在多核架构中片上 cache 可以被不同的核共享,因此数据重用可以分为核内重用和核间重用 2 类.当连续对同一数据或数据块的 2 个访问来自同一个核时,就是核内重用,反之来自不同的核则是核间重用.如图 14 所示,核 $p0$ 上的指令 x,z 对数据 $\boldsymbol{A}[1]$ 的重用是核内重用,

数据 A[10]在核 $p0$ 上指令 y 和核 $p1$ 上指令 k 的重用是核间重用. 因为存在核间数据重用,当 2 个核共享的数据存在于相同的 cache 行,共享的 cache 空间具有积极的意义;如果 2 个核置换掉共享 cache 中对方所需要的数据,那么对性能必然造成不利的损害. 无论核间重用还是核内重用,减少重用距离可以减少干扰数据的访存,降低造成性能损害的数据替换的可能性,提高 cache 空间中数据重用的概率. 以往的优化策略只考虑到核内数据重用距离的优化,多核架构上共享 cache 层次存储系统具有进一步开发优化的潜能.

通过实验观察分析发现,绝大部分应用的核内数据重用距离一般较短,核间重用距离则很长. 若只考虑开发核间重用距离,应用的整体性能虽然能有所提高,但比单核环境下核内数据重用优化技术带来的性能要差. 因此,文献[79]提出一种多核架构上平衡核内重用和核间重用的优化策略. 该方法的关键思想是对应用存取的数组划分成相同大小的块,根据块之间的数据共享和依赖关系对块的调度表进行权值分配,从而决定在哪个时间槽给哪个核分配哪个数据块. 这种集成映射和调度的策略可以有效地开发目标架构上垂直(核内)和水平(核间)方向的数据重用,提高应用的整体性能.

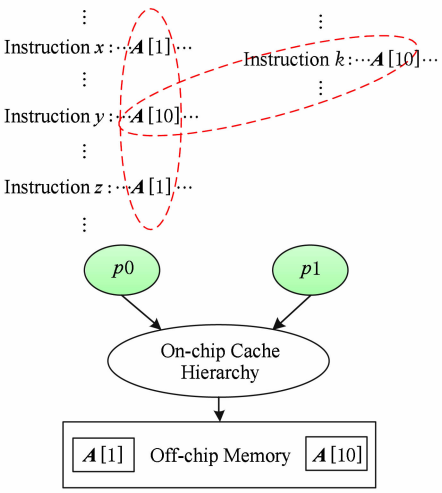


Fig. 14 An example of inter-core and intra-core data reuse[79].

图 14 数组元素的核内和核间数据重用示例[79]

文献[80]基于多核架构首先提出并发重用距离的概念,根据单线程的访存行为和多线程并发重用距离的关系建立一套概率模型,对非流水线多线程一类的应用具有良好的潜能. 文献[81]则实现了一种采样、并行的方法测量重用距离,为多线程程序提供一种快速廉价的局部性分析模型.

4.3 面向不完美嵌套循环的分块技术

许多带嵌套循环的应用不属于完美嵌套循环分块技术的适用范畴,因此研究者们对面向不完美嵌套循环和任意循环的分块技术展开了深入的研究. 文献[21,82]最早提出解决不完美嵌套循环分块的方法,对每个循环主体进行仿射嵌入,使不完美嵌套循环变换成单一的完美嵌套循环. 文献[83]开发了一个基于脚本构成的循环变换框架,可以用于不完美循环分块. 另一个用于对任意不完美嵌套循环集进行分块的 PLuTo 系统[19,47]则采用了基于多面体集的一般性方法,把多个循环迭代空间转换到统一的多面体迭代空间. 这些方法都要求固定分块大小,另外一些研究则是针对参数化分块而展开. 文献[46]描述了一种 PrimeTile 技术,沿着每一维把循环划分成 prolog,full 和 epilog 循环,分别计算完整分块和局部分块. PrimeTile 虽然能够对不完美嵌套循环集进行参数化多级分块,但会造成分块后代码长度激增. 文献[19]对不完美嵌套循环扫描多面体集的方法进行了参数化处理,而文献[48]则采用了对系统约束不等式组变换消去循环变量的非多面体方法. 同时,文献[20]则从不完美嵌套循环的 AST 上提取了紧凑的可展开内核,以此解决寄存器分块的问题.

4.4 循环分块相关扩展技术小结

通过上述对循环分块技术在多级分块、并行性、多核共享 cache 架构和任意应用范围的研究介绍可知,仍有不少问题需要深入研究:1)如何利用多级分块技术最大化的开发利用深度存储资源和多核架构的性能,同时有效控制多级分块带来的开销,还需要进一步研究,同时存储系统中每级分块对整体性能的影响尚未明确解决;2)面对主流的多核、异构架构(如 GPU),如何充分利用硬件的高效性、有效开发高并行度的分块程序同时保持良好的数据集可扩展性、负载均衡和最低的控制开销值得引起更多研究者的关注;3)传统的局部性评价方法不能满足多核共享 cache 架构访存特征的描述,需要建立系统的核间数据重用模型指导分块多线程的并发执行,新的架构资源需要得到进一步的开发;4)目前面向任意嵌套循环应用的分块方法仍受到算法复杂度过高、耗时较长和生成代码不够紧凑等问题的困扰,这些问题仍需要研究开发更为有效的优化方法来改善解决.

5　循环分块技术总结及展望

随着多核处理器架构和 cache 层次存储系统的发展,循环分块技术在数据局部性、代码并行化和编译性能方面存在着可探究的空间.针对 cache 失效的情况,研究者提出利用静态分析访存行为解决循环体中数据依赖关系的问题,扫描迭代空间的多面体集生成循环分块的边界约束条件,并对分块过程进行分解来降低计算复杂度.虽然这些方法可以建立精准的分块模型获取程序性能提升,但是实现过程复杂困难、时间开销较大、难以得到实际应用.于是,研究者又提出了基于参数化分块大小的循环分块方法,并将其扩展为多级分块和面向任意嵌套循环程序的分块技术,结合成熟的代码生成工具可以将程序循环体并行化,实现较好粒度的并行性.这类方法把分块大小作为符号参数处理,不仅可以灵活地编译生成分块代码,而且在硬件资源开发和程序性能上有所提升,但是在计算复杂度和细粒度并行性还有改善的空间.与此同时,因为分块大小对循环分块代码的性能有着重要影响,最优分块的形状和大小的选择方法得到了广泛深入的研究.在此基础上,本文综合分析了循环分块各项技术的研究现状,同时探讨了循环分块在多核存储架构发展趋势下面临的分块并行度、局部性评价、数据集可扩展性、负载均衡、计算成本等方面的挑战和问题.

根据以往的研究成果和目前正在进行的研究,我们可以得出如下结论:1)循环分块技术中的计算复杂度、代码生成时间和生成效率问题还未得到统一的解决,如何快速而准确地生成分块代码同时具有较低的算法复杂度,仍然存在着可探究的空间;2)如何更加有效地约束循环边界的迭代空间,进一步开发程序局部性,同时平衡分块开销,还需要进行全面深入地研究;3)最优分块大小的选择依然是一个开放式难题,研究清楚层次存储架构中每级分块对性能的影响,从而更好地指导确定分块大小,对性能的提升具有重要的意义;4)如何从目前线性代数库函数和 stencil 计算的应用范围扩展到一般的带嵌套循环的任意代码,并且有效地构建面向任意嵌套循环集的自动分块代码生成系统具有重要的现实意义;5)随着多核处理器的普及以及众核架构的来临,如何充分利用共享存储资源,开发具有高并行度、可扩展性、负载均衡的分块代码,与多核甚至众核架构资源进行性能匹配是一个需要深入研究的方面.针对上述的问题和挑战,展望循环分块技术在未来的研究发展趋势,应该是结合新的存储架构和应用数据规模特点,全面精准地描述多核共享 cache 架构上数据访存行为,提供更加有效、简洁、紧凑的线性空间多面体变换理论模型,在时空维度采用更为灵活的混合分块策略,利用粗细粒度结合的高并行度开发硬件资源高效的计算能力,针对优秀的代码生成器在代码质量、分块开销、可扩展性等方面进行相应的优化提升,同时构建分块代码性能的评估模型.

参　考　文　献

[1] Owens J D, Luebke D, Govindaraju N, et al. A survey of general-purpose computation on graphics hardware [J]. Computer Graphics Forum, 2007, 26(1): 80-113

[2] Grosser T, Cohen A, Kelly P, et al. Split tiling for GPUs: Automatic parallelization using trapezoidal tiles [C] //Proc of the 6th Workshop on General Purpose Processor Using Graphics Processing Units. New York: ACM, 2013: 24-31

[3] Kaspersky K. Code Optimization: Effective Memory Usage [M]. New Delhi, India: BPB Publications, 2004

[4] Baghdadi R, Cohen A, Verdoolaege S, et al. Improved loop tiling based on the removal of spurious false dependences [J]. ACM Trans on Architecture and Code Optimization(TACO): Special Issue on High-Performance Embedded Architectures and Compilers, 2013, 9(4): 1-26

[5] Pouchet L N, Bondhugula U, Bastoul C, et al. Loop transformations: Convexity, pruning and optimization [C] //Proc of the 38th ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages (POPL'11). New York: ACM, 2011: 549-562

[6] Lam M S, Wolf M E. A data locality optimizing algorithm [C] //Proc of the 12th ACM SIGPLAN Conf on Programming Language Design and Implementation (PLDI'91). New York: ACM, 1991: 30-44

[7] Lam M D, Rothberg E, Wolf M E. The cache performance and optimizations of blocked algorithms [C] //Proc of the 4th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 1991: 63-74

[8] Irigoin F, Triolet R. Supernode partitioning [C] //Proc of the 15th ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages (POPL'88). New York: ACM, 1988: 319-328

[9] Ancourt C, Irigoin F. Scanning polyhedra with DO loops [C] //Proc of the 3rd ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 1991: 39-50

[10] Xue Jingling. Loop Tiling for Parallelism [M]. Amsterdam, Netherlands: Kluwer Academic Publishers, 2000

[11] Renganarayana L, Kim D, Rajopadhye S, et al. Parameterized tiled loops for free [C] //Proc of 2007 ACM SIGPLAN Conf on Programming Language Design and Implementation(PLDI'07). New York: ACM, 2007: 405-414

[12] Renganarayanan L, Kim D, Strout M M, et al. Parameterized loop tiling [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 2012, 34(1): 1-41

[13] Betts S, Browne S, Dongarra J, et al. BLAS: Basic linear algebra subprograms [EB/OL]. (2005-07-25) [2014-03-06]. http://www.netlib.org/blas/

[14] Goumas G, Athanasaki M, Koziris N. An efficient code generation technique for tiled iteration spaces [J]. IEEE Trans on Parallel and Distributed Systems, 2003, 14(10): 1021-1034

[15] Renganarayana L. Scalable and efficient tools for multi-level tiling [D]. Fort Collins: Colorado State University, 2008

[16] Whaley R C, Dongarra J J. Automatically tuned linear algebra software [C] //Proc of 1998 ACM/IEEE Conf on Supercomputing (Supercomputing'98). Los Alamitos, CA: IEEE Computer Society, 1998: 1-27

[17] Whaley R C, Dongarra J J, Petitet A. ATLAS: Automatically tuned linear algebra software [EB/OL]. (2014-03-05) [2014-03-06]. http://www.netlib.org/atlas/

[18] Tavarageri S, Pouchet L N, Ramanujam J, et al. Dynamic selection of tile tizes [C] //Proc of the 18th IEEE Int Conf on High Performance Computing (HiPC'11). Piscataway, NJ: IEEE, 2011: 18-21

[19] Bondhugula U, Ramanujam J, Sadayappan P. A practical and fully automatic polyhedral program optimization system [C] //Proc of the 29th ACM SIGPLAN Conf on Programming Language Design and Implementation (PLDI'08). New York: ACM, 2008: 101-113

[20] Renganarayana L, Bondhugula U, Derisavi S, et al. Compact multi-dimensional kernel extraction for register tiling [C] //Proc of the 22nd Conf on High Performance Computing Networking, Storage and Analysis (SC'09). New York: ACM, 2009: 1-12

[21] Ahmed N, Mateev N, Pingali K. Tiling imperfectly-nested loop nests [C] //Proc of 2000 ACM/IEEE Conf on Supercomputing. Los Alamitos, CA: IEEE Computer Society, 2000: 4-10

[22] Wolfe M. More iteration space tiling [C] //Proc of 1989 ACM/IEEE Conf on Supercomputing. New York: ACM, 1989: 655-664

[23] Loveman D. Program improvement by source-to-source transformation [J]. Journal of the ACM (JACM), 1997, 24(1): 121-145

[24] Wilson R P, French R S, Wilson C S, et al. SUIF: An infrastructure for research on parallelizing and optimizing compilers [J]. ACM SIGPLAN Notices, 1994, 29(12): 31-37

[25] Liu Jun, Zhang Yuanrui, Ding Wei, et al. On-chip cache hierarchy-aware tile scheduling for multicore machines [C] //Proc of the 9th Annual IEEE/ACM Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2011: 161-170

[26] Stock K, Pouchet L N, Sadayappan P. Automatic transformations for effective parallel execution on Intel many integrated core [C] //Texas Advanced Computing Center Intel Highly Parallel Computing Symposium. Austin: Texas Advanced Computing Center, 2012

[27] Jia Yaocang, Wu Chenggang, Zhang Zhaoqing. Program's performance profiling optimization for guiding static cache partitioning [J]. Journal of Computer Research and Development, 2012, 49(1): 93-102 (in Chinese)
(贾耀仓, 武成岗, 张兆庆. 指导 cache 静态划分的程序性能 profiling 优化技术[J]. 计算机研究与发展, 2012, 49(1): 93-102)

[28] Stock K, Pouchet L N, Sadayappan P. Using machine learning to improve automatic vectorization [J]. ACM Trans on Architecture and Code Optimization (TACO): Special Issue on High-Performance and Embedded Architectures and Compilers, 2012, 8(4): 1-23

[29] Park E, Pouchet L N, Cavazos J, et al. Predictive modeling in a polyhedral optimization space [J]. International Journal of Parallel Programming, 2013, 41(5): 704-750

[30] Benabderrahmane M, Pouchet L N, Cohen A, et al. The polyhedral model is more widely applicable than you think [C] //Proc of the 19th Int Conf on Compiler Construction and European Joint Conf on Theory and Practice of Software (CC'10/ETAPS'10). Berlin: Springer, 2010: 283-303

[31] Bastoul C. Code generation in the polyhedral model is easier than you think [C] //Proc of the 13th Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2004: 7-16

[32] Cohen A, Sigler M, Girbal S, et al. Facilitating the search for compositions of program transformations [C] //Proc of the 19th Annual Int Conf on Supercomputing. New York: ACM, 2005: 151-160

[33] Vasilache N, Bastoul C, Cohen A, et al. Violated dependence analysis [C] //Proc of the 20th Annual Int Conf on Supercomputing. New York: ACM, 2006: 335-344

[34] Pugh W. Omega test: A practical algorithm for exact array dependency analysis [J]. Communications of the ACM, 1992, 35(8): 102-114

[35] Quilleré F, Rajopadhye S, Wilde D. Generation of efficient nested loops from polyhedral [J]. International Journal Parallel Programming, 2000, 28(5): 469-498

[36] Vasilache N, Bastoul C, Cohen A. Polyhedra code generation in the real world [G] //LNCS 3923: Proc of the 15th Int Conf on Compiler Construction. Berlin: Springer, 2006: 185-201

[37] Groβer T. CLooG: The chunky loop generator [EB/OL]. (2013-10-11) [2014-03-06]. http://www.cloog.org/

[38] Loechner V, Risset T. Polylib: A library of polyhedral functions [EB/OL]. [2014-03-06]. http://www.irisa.fr/polylib/

[39] Wilde D K. A library for doing polyhedral operations [J]. Parallel Algorithms and Applications, 2000, 15(3): 137-166

[40] Girbal S, Vasilache N, Bastoul C, et al. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies [J]. International Journal of Parallel Programming, 2006, 34(3): 261-317

[41] Kelly W, Pugh W, Rosser E. Code generation for multiple mappings [C] //Proc of the 5th Symp on the Frontiers of Massively Parallel Computation. Piscataway, NJ: IEEE, 1995: 332-341

[42] Kamil S, Datta K, Williams S, et al. Implicit and explicit optimizations for stencil computations [C] //Proc of the 2006 Workshop on Memory System Performance and Correctness. New York: ACM, 2006: 51-60

[43] Renganarayanan L, Kim D, Rajopadhye S, et al. TLoG: A parameterized tiled loop generator [EB/OL]. (2008-11-24) [2014-03-06]. http://www.cs.colostate.edu/MMAlpha/tiling/

[44] Kim D, Renganarayanan L, Rostron D, et al. Multilevel tiling: M for the price of one [C] //Proc of the ACM/IEEE Conf on Supercomputing (SC'07). New York: ACM, 2007: 1-12

[45] Kim D, Renganarayanan L, Rostron D, et al. HiTLoG: Hierarchical tiled loop generator [EB/OL]. (2008-11-24) [2014-03-06]. http://www.cs.colostate.edu/MMAlpha/tiling/

[46] Hartono A, Baskaran M M, Bastoul C, et al. PrimeTile: A parametric multi-level tiler for imperfect loop nests [C] //Proc of the 23rd Int Conf on Supercomputing. New York: ACM, 2009: 147-157

[47] Bondhugula U, Ramanujam J, Sadayappan P. PLuTo: A polyhedral automatic parallelizer and locality optimizer for multicores [EB/OL]. (2013-12-28) [2014-03-06]. http://pluto-compiler.sourceforge.net/

[48] Baskaran M, Hartono A, Tavarageri S, et al. Parameterized tiling revisited [C] //Proc of the 8th Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2010: 200-209

[49] Jiménez M, Llabería J M, Fernández A. Register tiling in nonrectangular iteration spaces [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 2002, 24(4): 409-453

[50] Chame J, Moon S. A tile selection algorithm for data locality and cache interference [C] //Proc of ACM Int Conf on Supercomputing. New York: ACM, 1999: 492-499

[51] Sarkar V, Megiddo N. An analytical model for loop tiling and its solutions [C] //Proc of IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2000: 146-153

[52] Ferrante J, Sarkar V, Thrash W. On estimating and enhancing cache effectiveness [C] //Proc of the 4th Int Workshop on Languages and Compilers for Parallel Computing. Berlin: Springer, 1991: 328-343

[53] Hsu C, Kremer U. A quantitative analysis of tile size selection algorithms [J]. The Journal of Supercomputing, 2004, 27(3): 279-294

[54] Kisuki T, Knijnenburg P, O'Boyle M. Combined selection of tile sizes and unroll factors using iterative compilation [J]. The Journal of Supercomputing, 2003, 24(1): 43-67

[55] Tapus C, Chung I-H, Hollingsworth K J. Active harmony: towards automated performance tuning [C] //Proc of 2002 ACM/IEEE Conf on Supercomputing. Los Alamitos, CA: IEEE Computer Society, 2002: 1-11

[56] Tiwari A, Chen C, Chame J, et al. A scalable auto-tuning framework for compiler optimization [C] //Proc of 2009 IEEE Int Symp on Parallel&Distributed Processing. Piscataway, NJ: IEEE, 2009: 1-12

[57] Whaley R, Petitet A, Dongarra J. Automated empirical optimization of software and the ATLAS project [J]. Parallel Computing, 2002, 27(1): 3-35

[58] Goto K, Geijn R. High-performance implementation of the level-3 BLAS [J]. ACM Trans on Mathematical Software, 2008, 35(1): 1-14

[59] Datta K. Auto-tuning stencil codes for cache-based multicore platforms [D]. Berkeley: University of California at Berkeley, 2009

[60] Chen C, Chame J, Hall M. Combining models and guided empirical search to optimize for multiple levels of the memory hierarchy [C] //Proc of the 3rd Int Symp on Code generation and optimization (CGO'05). Piscataway, NJ: IEEE, 2005: 111-122

[61] Yotov K, Pingali K, Stodghill P. Think globally, search locally [C] //Proc of the 19th Annual Int Conf on Supercomputing. New York: ACM, 2005: 141-150

[62] Shirako J, Sharma K, Fauzia N, et al. Analytical bounds for optimal tile size selection [C] //Proc of ETAPS Int Conf on Compiler Construction. Berlin: Springer, 2012: 101-121

[63] Ramanujam J, Sadayappan P. Tiling multidimensional iteration spaces for multicomputers [J]. Journal of Parallel and Distributed Computing, 1992, 16(2): 108-120

[64] Yuki T, Renganarayanan L, Rajopadhye S, et al. Automatic creation of tile size selection models [C] //Proc of the 8th Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2010: 190-199

[65] Rahman M, Pouchet L N, Sadayappan P. Neural network assisted tile size selection [C] //Proc of the 5th Int Workshop on Automatic Performance Tuning. Berlin: Springer, 2010

[66] Bilmes J, Asanovic K, Vuduc R, et al. PHiPAC: Portable high performance ANSI C [EB/OL]. [2014-03-06]. http://www1.icsi.berkeley.edu/~bilmes/phipac/

[67] Renganarayanan L, Matha M H, Dewri R, et al. Towards optimal multi-level tiling for stencil computations [C] //Proc of the 21st IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2007: 1-10

[68] Jiménez M. Multilevel tiling for non-rectangular iteration spaces [D]. Barcelona, Catalunya, Spain: University of Politècnica de Catalunya, 1999

[69] Jiménez M, Llabería J, Fernández A. A cost-effective implementation of multilevel tiling [J]. IEEE Trans on Parallel Distributed Systems, 2003, 14(10): 1006-1020

[70] Bondhugula U, Baskaran M, Krishnamoorthy S, et al. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model [C] //Proc of the 17th Int Conf on Compiler Construction and European Joint Conf on Theory and Practice of Software (CC'08/ETAPS'08). Berlin: Springer, 2008: 132-146

[71] Wonnacott D, Strout M. On the scalability of loop tiling techniques [C] //Proc of the 3rd Int Workshop on Polyhedral Compilation Techniques(IMPACT). Berlin: Springer, 2013: 3-11

[72] Bandishti V, Pananilath I, Bondhugula U. Tiling stencil computations to maximize parallelism [C] //Proc of 2012 Int Conf on High Performance Computing, Network, Storage and Analysis. Los Alamitos, CA: IEEE Computer Society, 2012: 1-11

[73] Hartono A, Baskaran M, Ramanujam J, et al. DynTile: Parametric tiled loop generation for parallel execution on multicore processors [C] //Proc of 2010 IEEE Int Symp on Parallel & Distributed Processing (IPDPS). Piscataway, NJ: IEEE, 2010: 1-12

[74] Baskaran M, Bondhugula U, Krishnamoorthy S, et al. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories [C] //Proc of the 13th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2008: 1-10

[75] Baskaran M, Bondhugula U, Krishnamoorthy S, et al. A compiler framework for optimization of affine loop nests for GPGPUs [C] //Proc of the 22nd Annual Int Conf on Supercomputing(ICS'08). New York: ACM, 2008: 225-234

[76] Di Peng, Wu Hui, Xue Jingling, et al. Parallelizing SOR for GPGPUs using alternate loop tiling [J]. Parallel Computing, 2012, 38(6): 310-328

[77] Holewinski J, Pouchet L N, Sadayappan P. High-performance code generation for stencil computations on GPU architectures [C] //Proc of the 26th ACM Int Conf on Supercomputing(ICS'12). New York: ACM, 2012: 311-320

[78] Grosser T, Verdoolaege S, Cohen A, et al. The promise of hybrid hexagonal/classical tiling for GPU, RR-8339 [R]. Paris: Institut National de Recherche en Informatique et en Automatique (INRIA), 2013

[79] Zhang Yuanrui, Kandemir M, Yemliha T. Studying inter-core data reuse in multicores [C] //Proc of the 13th ACM SIGMETRICS Joint Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 2011: 25-36

[80] Jiang Yulian, Zhang E, Tian Kai, et al. Is reuse distance applicable to data locality analysis on chip multiprocessors? [G] //LNCS 6011: Compiler Construction. Berlin: Springer, 2010: 264-282

[81] Schuff D, Kulkarni M, Pai V. Accelerating multicore reuse distance analysis with sampling and parallelization [C] //Proc of the 19th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2010: 53-64

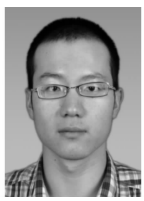
[82] Ahmed N, Mateev N, Pingali K. Synthesizing transformations for locality enhancement of imperfectly-nested loop nests [J]. International Journal of Parallel Programming, 2001, 29(5): 493-544

[83] Chen C, Chame J, Hall M. Chill: A framework for composing high-level loop transformations, 08-897 [R]. Los Angeles, CA: University of Southern California, 2008

Liu Song, born in 1987. PhD candidate. His research interests include code optimization and high performance computing.

Wu Weiguo, born in 1963. Professor and PhD supervisor. Member of China Computer Federation. His research interests include high performance computer architecture, cloud computing and embedded system(wgwu@mail.xjtu.edu.cn).

Zhao Bo, born in 1990. MSc. His research interests include multicore systems and source to source code transformation (zhaobo36@stu.xjtu.edu.cn).



Jiang Qing, born in 1991. MSc. His research interests include compiler optimization and machine learning (xiezhongxiu@gmail.com).