计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190734
57(12):2621-2634,2020

一种基于空间密铺的星型 Stencil 并行算法

曹 杭[1,2] 袁 良[1] 黄 珊[1,2] 张云泉[1] 徐勇军[1] 陆鹏起[1,2] 张广婷[1]
[1](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)
[2](中国科学院大学 北京 100049)
(caohang@ict.ac.cn)

A Parallel Star Stencil Algorithm Based on Tessellating

Cao Hang[1,2], Yuan Liang[1], Huang Shan[1,2], Zhang Yunquan[1], Xu Yongjun[1], Lu Pengqi[1,2], and Zhang Guangting[1]
[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract The Stencil computation, i.e. the structured grid computing, is a very common kind of loop nesting algorithm in scientific and engineering applications. The exhaustively studied tiling method is of great effectiveness as one of the transformation techniques to exploit the data locality and parallelism of Stencil computations. However, the state-of-the-art work of tiling often uniformly handles different Stencil shapes. We first present a concept called natural block to identify the difference between the star and box Stencils. Then we propose a new two-level tessellation scheme for star Stencils, where the natural block, as well as its successive blocks can tessellate the spatial space and their extensions along the time dimension are able to form a tessellation of the iteration space. Furthermore, a novel implementation technique called double updating is developed for star Stencils specifically, which updates each element twice continuously and improves the in-core data reuse pattern. In addition, we adopt coarsening and block reuse to enhance the parallelization performance. Theoretical analysis shows that our scheme achieves a better cache complexity than existing methods such as Girih and Pluto. The experiments on performance and bandwidth are conducted on a multicore system. The results demonstrate the effectiveness of our approach.

Key words Stencil computation; tessellation; star Stencil; box Stencil; natural block

摘 要 Stencil 计算(模板计算)是科学工程应用中一类常见的嵌套循环算法.分块方法是提高数据局部性和并行性的高效优化技术之一,目前已有大量针对分块方法的探索,但现有工作往往对不同 Stencil 形状都采用同一处理方法.首先在空间层面引出“自然块”的概念来区分星型 Stencil 和盒型 Stencil 的

收稿日期:2019-10-12;**修回日期**:2020-04-07
基金项目:国家重点研发计划项目(2016YFB0200803);中国科学院战略性先导科技专项(C 类)(XDC01040100);国家自然科学基金项目(61972376,62072431,61432018);北京市自然科学基金项目(L182053)
This work was supported by the National Key Research and Development Program of China (2016YFB0200803), the Strategic Priority Research Program of Chinese Academy of Sciences (C) (XDC01040100), the National Natural Science Foundation of China (61972376, 62072431, 61432018), and the Beijing Natural Science Foundation (L182053).
通信作者:袁良(yuanliang@ict.ac.cn)

特征,然后提出一个新的针对星型 Stencil 的 2 层密铺方案,此方案中自然块和它的后继块可以密铺数据空间区域,这些分块沿着时间维度扩展,能够密铺整个迭代空间.此外,针对星型 Stencil 设计了一个新颖的"2 次更新"优化技术,改善了核内数据重用模式.理论分析表明:此方案相比现有方法有更低的缓存复杂度,实验结果证实了此方案的有效性.

关键词 Stencil 计算;密铺;星型 Stencil;盒型 Stencil;自然块

中图法分类号 TP301

Stencil 计算(模板计算)是科学工程应用中一类常见的嵌套循环算法,也被称为"结构化网格计算",是 13 个伯克利核心计算模式之一.在动态规划、图像处理等领域的很多算法中也包含相似的依赖模式.Stencil 定义了一种更新结点所需邻居结点的模式.Stencil 计算在时间维度上迭代更新规则的 d 维网格(也称为数据空间),数据空间沿着时间维度更新,产生的 $d+1$ 维空间被称为迭代空间.

与其他的数值算法不同,例如稠密或稀疏矩阵计算类型的多样性主要来源于输入数据如稠密或稀疏矩阵的数据量,Stencil 的计算类型本质上取决于问题的类型.Stencil 可以从不同的角度进行分类,如格点维度(1 维,2 维,…),邻居数目,即阶数(3 点,5 点,…),形状(盒型,星型,…),依赖类型(高斯-赛德尔,雅可比,…)和边界条件(常量的,周期的,…)等.

Stencil 计算具有计算密集度低的特征,d 维 Stencil 的简单实现由 $d+1$ 个循环组成,其中最外层循环遍历时间维度,内层循环更新 d 维空间的所有格点.简单实现的数据重用度较差且典型的带宽受限.

分块是提高多重嵌套循环的数据局部性和并行度最有效的优化技术之一,目前有大量针对 Stencil 计算分块方案的研究.空间分块算法促进了 2 维至更高维度 Stencil 在一个时间步内的数据重用.为了进一步增强数据重用度,往往同时考虑时间维度和空间维度,后文将详细介绍这些技术.另一类工作采用简单的超矩形分块形状[1-4]并引入冗余计算来解决块间数据依赖.这些研究包括自动调优框架[5-12]、编程模型[13-14]、CPU[2,15-18]、GPU[19-23]和 Phi[24-25]上的手动调优实现.

不同的 Stencil 类型也衍生出不同的解决方案.Pluto 系统[26]延伸到周期性边界条件 Stencil 的处理[27].菱形分块最初是针对 1 维 Stencil 的分块[28],然后扩展到更高维度的 Stencil[29].高速缓存参数无关(cache oblivious)方案从最初的算法[30-31],CORALS(cache oblivious parallelograms)方案[32],发展到能处理任意 Stencil 并达到最大并发度的 Pochoir 系统[33].

据我们所知,目前没有针对不同 Stencil 形状的解决方案,特别是盒型和星型 Stencil.星型 Stencil 只依赖于每个坐标轴上的点,盒型 Stencil 在此之外还有对角线上的数据依赖.不区分盒型和星型 Stencil 的根本原因可能是盒型 Stencil 包含星型 Stencil,任何适用于盒型 Stencil 的算法也能应用到星型 Stencil.

本文首先在数据空间上提出"自然块"的概念来确定星型和盒型 Stencil 之间的区别.然后针对星型 Stencil 提出了一个新的 2 层密铺方案,在此方案中,自然块和它的后继块可以密铺整个空间,而他们沿着时间维度扩展后形成对迭代空间的密铺.密铺框架类似于文献[34]中提出的方法.我们将统一阐述基于自然块概念的 2 种方案并将已有的方案称为盒型密铺而将新提出的方案称为星型密铺.

在密铺方案中区别盒型和星型 Stencil 有 2 个主要的优点,分别利用不同层次上的数据局部性.首先,当给定缓存大小时,相比盒型密铺,用星型 Stencil 的自然块密铺能得到更大的块,这样可以更有效地减少内存系统的压力;其次,我们针对第二后继块中的元素提出了一个新颖的"2 次更新"技术,可对 1 个元素连续进行 2 次更新,进一步提升了核心数据的重用.

1 相关工作

1.1 分块技术

分块技术[35-39]是探究多层嵌套循环数据局部性和并行度最有效的转换技术之一.Wonnacott 和 Strout[40]对比了一些现有分块方案的可扩展性,下面我们将总结一些和 Stencil 计算有关的技术.

1) 重叠分块(overlapped tiling).高性能领域中经常在手动调优的实现中采用超矩形分块[1-4].为了执行多个时间步,往往采用冗余计算[41]来解决分块之间的依赖,这就是重叠分块[28,42].Philips 和

Fatica[22]提出了 GPU 上手动调优的手写 3.5 维分块实现方案,在 2.5 维分块方案[4]上增加了时间维度上的划分.Demmel 等人[43-44]减少了 2 倍的冗余计算.形状规则的超矩形可以获得更高的并发度和更多细粒度优化.

2) 时间偏移(time skewing).时间偏移分块[45-47](分块形状在 2 维上是平行四边形,在 3 维上是平行六面体,在更高维上是超平行体)消除了冗余计算,但是大多数方法只用单一形状的块填充空间,导致流水线启动和有限的并发度[48-49].

3) 菱形分块(diamond tiling).Bondhugula 等人[26]对 1 维 Stencil 采用菱形分块.Bandishti 等人[29,50]将此方法扩展到更高空间维度的 Stencil.Grosser 等人[51]对 2 维的菱形尖端粗粒度化形成六边形,在 3 维上演化成截掉顶端的八面体.

4) 缓存无关分块(cache oblivious tiling).缓存无关技术旨在内存层次结构参数未知时充分利用数据局部性.Frigo 和 Strumpen 提出了第 1 个串行[30]和并行[31]缓存无关 Stencil 算法.缓存无关平行四边形方法[32]同时分离时间空间维度,但是会导致波阵面并行.Pochior[33]实现了多维空间划分,能够同时划分所有空间维度.

5) 分裂分块(split tiling).分裂分块方法发掘每个块中的独立子块,然后将这些子块发送给他们的后继块,使得剩下的区域同样可以并发执行[28,52-54].Grosser 等人[55]提出了另一个类似于具有缓存无关范式的多维空间划分的分裂分块方法.嵌套分裂分块(the nested split-tiling)[56]能在所有空间维度递归分裂.

6) 混合分块(hybrid tiling).Strzodka 等人[57]提出的 CATS(cache accurate time skewing)算法结合了菱形分块、平行四边形分块和流水线处理.Grosser 等人[58]采用混合六边形和平行四边形的分块算法.这些算法取某一维空间和时间维度组成平面,用六边形分块和菱形分块进行划分,在剩下的空间维度中采用时间偏移分块,从而分解迭代空间.混合分裂分块方法[56]结合了嵌套分裂分块和时间偏移分块.

7) 密铺分块(tessellating).密铺方案[34]适用于包括星型和盒型等不同 Stencil 类型.空间被一系列分块密铺,分块之间可以无冗余并行执行.相应地,这些分块在时间维度上扩展后得到的扩展块可以在迭代空间中形成密铺.简单明了的数学特性使得这种方法可以与细粒度优化方法相结合.

1.2　优化方法

为了使核心程序达到更高的性能,核心内的优化技术如计算重用[59-60]、计算重组[61]、向量化[15,62-64]和数据布局转换[65-66]等十分关键.但是这些方法大多没有对不同的 Stencil 形状进行区分.本文设计实现了一个新颖的技术来提升星型 Stencil 核心程序级别上的数据重用.这种技术也可以应用到其他的分块方法上.

2　算法描述

本节首先介绍一个新的概念,即自然块,来区分不同的 Stencil 形状特征.然后围绕自然块和后继块这 2 个方面,详细描述新提出的星型 Stencil 密铺方案以及已有的盒型密铺方案[34].最后将密铺方案应用于高阶 Stencil 和不同的边界条件.

Gauss-Seidel Stencil 需要执行逐点 45°流水线启动,使用分块技术也不能使其完全并发启动[28].因此我们仅考虑 Jacobi Stencil.此外,星型 Stencil 只能应用到 2 维数据空间中,我们只讨论 2 维 Stencil.在第 5 节的实验中,对于 3 维星型 Stencil 将保留 1 维数据空间不进行切分.

密铺 3 维迭代空间对应于计算 2 维 Stencil,我们首先将其划分成多个相同的时间片.在一个时间片开始时所有格点处于同一时间维度,结束时所有格点都更新了 t 个时间步(在下面的例子中 $t=4$).不失一般性,我们只讨论第 1 个时间片的密铺,即密铺前所有格点的时间维度均为 0,密铺后所有格点的时间维度为 t.

2.1　自然块 $\boldsymbol{B}_0(\mathbb{B}_0)$

当所有格点在时间维度上的值相同时,更新某一格点 t 个时间步所需的最小邻居点集被称为时间步为 t 的自然块,简称自然块.

图 1(a)和图 1(b)分别展示了 2D9P 盒型 Stencil 和 2D5P 星型 Stencil 时间步长为 4 的自然块.

```
0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 0
0 1 2 2 2 2 2 1 0
0 1 2 3 3 3 2 1 0
0 1 2 3 4 3 2 1 0
0 1 2 3 3 3 2 1 0
0 1 2 2 2 2 2 1 0
0 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0
```

(a) 2D9P box Stencil

```
      0
    0 1 0
  0 1 2 1 0
0 1 2 3 2 1 0
  0 1 2 3 2 1 0
    0 1 2 1 0
      0 1 0
        0
```

(b) 2D5P star Stencil

Fig. 1　Natural block $B_0(\mathbb{B}_0)$

图 1　自然块 $B_0(\mathbb{B}_0)$

对于1维Stencil,盒型和星型形状是相同的,时间步长为4的自然块是0,1,2,3,4,3,2,1,0.自然块中的数字代表相应位置上格点的时间更新步数.因为开始时所有格点处于同一时间维度,所以它们在第1个时间步同时开始更新.代表数据空间中物理格点的集合时,自然块用 B_0 表示(之后将只会用到自然块的形状,即盒型Stencil的自然块是正方形的,星型Stencil的自然块是菱形的).自然块 B_0 为Stencil定义的扩展,将一个2D9P Stencil的网格点 (x,y),更新 t 个时间步使之从 $(x,y,0)$ 迭代到 (x,y,t),需要 $(2t+1)^2$ 个网格点,具体范围为:$(x+a,y+b,0)$,其中 $a\in[-t,t]$,$b\in[-t,t]$;而对于2D5P Stencil,范围为 $2t^2+2t+1$ 个网格点,其中 $a\in[-t,t]$,$b\in[-t,t]$,且 $a+b\leqslant t$.而当自然块定义中 $t=1$ 时,自然块与Stencil定义等价.在 B_0 基础上赋予每个格点1个新的时间维度来表示其时间更新步数,此时自然块用 $\mathbb{B}_0$ 表示,用于解释迭代空间的密铺.

最大更新方法[34]与自然块概念相近.具体定义为:在数据空间中给定 B,沿着时间维度最大更新每个格点,直到不满足Stencil定义的依赖关系为止.形式上而言,对所有格点 $b\in B$,满足以下2个条件中任一个时,停止更新:

1) $time[b]=t$,t 是给定的最大时间更新步数;

2) 存在至少一个 b 的邻居 b',b' 的时间维度严格小于 $b(time[b']<time[b])$ 或 $b'\notin B$.

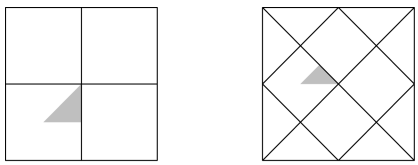
自然块和最大更新在某种程度上可以看作是对偶的概念.给定同样的格点集 B_0,且所有格点更新前都处于同一时间维度,则这2种机制实际上会产生同样的 $\mathbb{B}_0$.此外,因为分块之间没有依赖关系,用这2种方法产生的所有分块都能并发执行.

但是这2种概念并不完全等价,实际上它们互为补充.一方面,最大更新没有给出密铺数据空间块的具体形状.盒型密铺[34]采用盒型Stencil的自然块(超立方体)来密铺盒型和星型Stencil,盒型Stencil包含星型Stencil,因此可以保证结果的正确性.在本工作中,我们将采用不同类型Stencil各自的自然块.

另一方面,单一的自然块 $\mathbb{B}_0$ 不能密铺迭代空间.我们需要其他2个后继块 B_1 和 B_2.但是它们包含的格点在更新前初始时间维度并不完全相同,这使我们不能采用自然块的概念.最大更新方法可以无视格点的时间维度,因此迭代空间中后继块的扩展块 $\mathbb{B}_1$ 或 $\mathbb{B}_2$ 可以由最大更新方法产生.

2.2 用 $B_0(\mathbb{B}_0)$,$B_1(\mathbb{B}_1)$ 和 $B_2(\mathbb{B}_2)$ 密铺

盒型Stencil自然块 B_0 的形状是正方形,星型Stencil自然块的形状是菱形,这2种形状都能密铺2维数据空间.图2(a)和图2(b)(忽略灰色区域)分别表示盒型Stencil的4个自然块和星型Stencil的12个自然块.注意这些自然块可以并行执行.



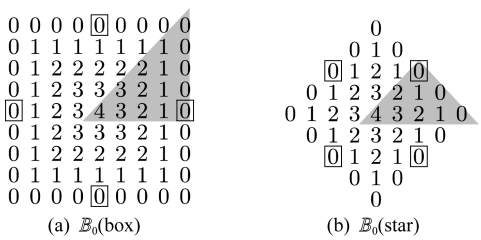
(a) $4B_0$(box) (b) $12B_0$(star)

Fig. 2 Tessellating data space with B_0

图2 用 B_0 密铺数据空间

下面给出数学公式将 B_i 扩展为 $\mathbb{B}_i$($i=1,2,3$),即赋予每个点 (x,y) 在 $\mathbb{B}_i$ 中的时间更新步数,时间更新步数在盒型Stencil中用 $T_{B_i(x,y)}$ 表示,在星型Stencil中用 $T_{S_i(x,y)}$ 表示.密铺方案的特性使得我们只需推导一个自然块 B_0 中的公式.我们将这个自然块的中心点(更新 $t=4$ 个时间步的格点)在笛卡儿坐标系中的坐标设为(0,0),2个坐标轴的正方向分别是向右和向上.此外,由 B_i 密铺到 B_{i+1} 密铺的转变允许我们只需关注满足条件 $x\geqslant y\geqslant 0$ 的点 (x,y),因此我们只对每个 B_i 密铺图形中的灰色三角区域进行说明,并给出 $\mathbb{B}_i$ 图形中灰色三角形区域的公式.需要注意的是,图2~8中的灰色三角形区域表示同一格点集合,这个集合是总能属于一个 B_i 块的最大格点集合.如图3所示,根据自然块的定义,1维盒型和星型Stencil中 $\mathbb{B}_0$ 的计算为

$$\begin{aligned} T_{B_0}(x,y)&=t-x,\\ T_{S_0}(x,y)&=t-(x+y). \end{aligned} \tag{1}$$



(a) $\mathbb{B}_0$(box) (b) $\mathbb{B}_0$(star)

Fig. 3 Determining the center points of B_1

图3 由 $\mathbb{B}_0$ 确定 B_1 块的中心点

由图3可知,这些自然块中只有中心点被更新了 t 次,单独由 $\mathbb{B}_0$ 并不能密铺更新步数为 t 的时间片,更不能密铺整个3维迭代空间.我们需要引入另外2个后继块 B_1 和 B_2 来密铺数据空间,并且利用

其对应的$\mathcal{B}_1$和$\mathcal{B}_2$完整地密铺迭代空间,即更新所有格点 t 个时间步.

后继块 B_1,B_2 等是对自然块 B_0 进行分裂而后重组格点所产生的.分裂和重组的关键在于确定这些 B_1 的中心点,之后每个格点可以简单地划分给离它最近的中心点.这种方法保证除边界上的点外,每个格点只属于某一个 B_1,并且 B_1 可以密铺数据空间.

自然块的一个性质是中心点在时间维度更新步数最多,因此最显然的处理方式是将更新次数为 0 的格点凸集的中心点作为后继块的中心点.需要注意的是,后继块的中心点总在前驱块的边界上,只用考虑边界上标为 0 的格点凸集.

由图 3 可知,盒型和星型 Stencil 中所有标为 0 的格点分别在正方形和菱形的边界上.根据凸集的要求,4 条边被视为 4 个凸集,因此方框中的 4 个格点 0 即为对应 4 个 B_1 的中心点.

图 4(a)和图 4(b)中展示了由 B_0 到 B_1 密铺数据空间的转变过程,其中虚线代表对 B_0 的划分,可以看到每个 B_0 被划分成 4 个子块.图 4(c)和图 4(d)展示了用 B_1 密铺数据空间的方法,其中共享同一个中心点的 B_0 子块合并组成 1 个 B_1.经过此转变过程,盒型 Stencil 由 4 个 B_0 转换为 12 个 B_1,星型 Stencil 由 12 个 B_0 转换为 16 个 B_1.

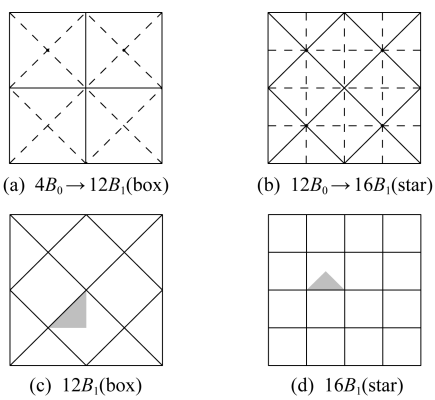


Fig. 4　Tessellating data space with B_1

图 4　用 B_1 块密铺数据空间

通过最大更新策略,相应的$\mathcal{B}_1$如图 5 所示.1 阶盒型和星型 Stencil 的$\mathcal{B}_1$可以计算得出:

$$T_{B_1}(x,y)=x-y,$$
$$T_{S_1}(x,y)=2y. \tag{2}$$

结合$\mathcal{B}_0$和$\mathcal{B}_1$密铺迭代空间相当于将对应格点的值相加,即 $T_{B_0}(x,y)+T_{B_1}(x,y)=t-y$ 或 $T_{S_0}(x,y)+T_{S_1}(x,y)=t-x+y$.图 6 展示了累加

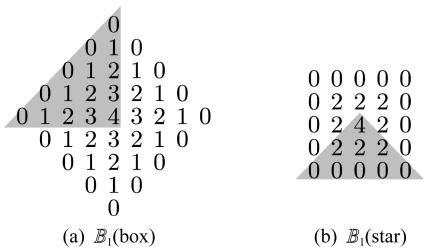


Fig. 5　$\mathcal{B}_1$

图 5　$\mathcal{B}_1$

后的值.我们只用检查灰色三角形区域的结果,将$\mathcal{B}_0$和$\mathcal{B}_1$中相应灰色三角形区域中的值相加可以验证其结果正确性.

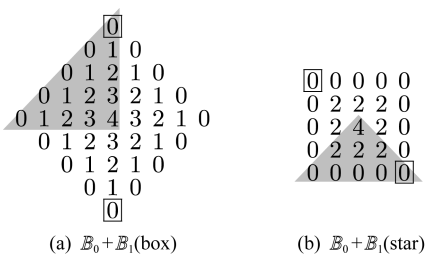


Fig. 6　Determining the center points of B_2

图 6　由$\mathcal{B}_0$和$\mathcal{B}_1$确定 B_2 的中心点

通过相似的步骤确定图 6(a)和图 6(b)中所示的 2 个 0 点是 B_2 的中心点.从 B_1 到 B_2 的转换过程如图 7(a)和图 7(b)中虚线所示,由 B_2 密铺数据空间的方式如图 7(c)和图 7(d)所示.盒型 Stencil 的密铺方式由 12 个 B_1 转换为 9 个 B_2,星型 Stencil 由 16 个 B_1 转换为 13 个 B_2.

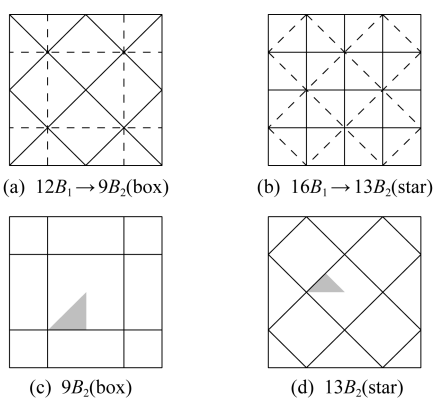


Fig. 7　Tessellating data space with B_2

图 7　用 B_2 块密铺数据空间

相似地,采用最大更新策略,相应的$\mathcal{B}_2$如图 8 所示.1 阶盒型和星型 Stencil 中$\mathcal{B}_2$可以计算得出:

$$T_{B_2}(x,y)=y,$$
$$T_{S_2}(x,y)=x-y. \tag{3}$$

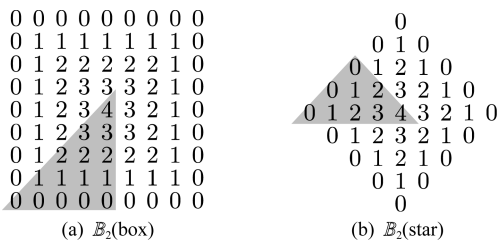


Fig. 8 Natural block $B_2(\mathbb{B}_2)$

图 8 自然块 $B_2(\mathbb{B}_2)$

结合$\mathbb{B}_0$,$\mathbb{B}_1$,$\mathbb{B}_2$块密铺迭代空间相当于将对应格点值相加,由式(1)~(3)相加可得:

$$\sum_{i=0}^{2} T_{B_i}(x,y)=\sum_{i=0}^{2} T_{S_i}(x,y)=t.$$

每个格点都更新 t 步,因此$\mathbb{B}_i(i=1,2,3)$可以密铺盒型和星型 Stencil 迭代空间.

需要注意的是存在 2 种不同类型的$\mathbb{B}_1$.以图 4(a)(c)和图 7(a)中的盒型 Stencil 为例,尽管所有的 B_1 都是菱形的,但它们从 2 个不同方向产生,即△+▽→◇和◁+▷→◇,并且将被沿着与产生方向相垂直的方向划分,即◇→◁+▷和◇→△+▽.这 2 种类型分别对应于满足 $x\geqslant y\geqslant 0$ 或 $y\geqslant x\geqslant 0$ 的三角形块.第 2 种$\mathbb{B}_1$的生成公式可以由交换 x 和 y 得到.

2.3 高阶 Stencil 和边界条件

2.1 和 2.2 节中针对 1 阶 Stencil 进行了讨论,2 阶甚至更高阶 Stencil 计算在实际应用中也很常见.假设 Stencil 在 1 个维度的阶数为 m,可以将沿着这一维度的 m 个格点组合成 1 个超格点.对于所有高阶依赖关系,我们都可以采用这种组合方式将 m 阶依赖减少为超格点间的 1 阶依赖.采用 B_i 密铺的方法可以应用到由超格点组成的 1 阶 Stencil 中,在每个$\mathbb{B}_i$中,同一超格点中的所有格点更新相同的时间步数.

解决周期性边界条件问题时,我们可能需要在 1 维 Stencil 中对 1 个块进行拉伸,或者在高维 Stencil 中拉伸 1 组块.采用拉伸过的六边形块来保证在目标维度 2 边的 2 个子块可以组成 1 个自然块.

3 实现方法

基于第 2 节描述的数学框架可以手工实现代码.本节将介绍手工实现代码的具体细节.自动代码生成工具的实现将作为未来的工作.

3.1 粗粒度化

粗粒度化的实现有 2 个动机.首先,密铺方案需要在$\mathbb{B}_i$的尖端处理 B_i 中较小的数据区域,特别是 B_0 的开始和 B_2 的结束.较差的硬件预取利用率或其他因素可能会限制硬件能力的发挥.其次,相邻的块可能会造成共享缓存上的伪共享.

星型密铺的粗粒度化和盒型密铺类似,但是菱形的大小只取决于 1 个参数即菱形边长,因此星型密铺中我们只需另外 1 个参数来将$\mathbb{B}_0$和$\mathbb{B}_2$的顶点扩展成 1 个小的菱形.图 9 展示了 2 维 Stencil 的粗粒度化方法.$\mathbb{B}_i$中第 1 个(最后 1 个)时间步更新的格点区域被称为开始(结束)区域.为了提高空间局部性,$\mathbb{B}_0$的结束区域(图 9(a)中深灰色区域)和开始区域(图 9(c)中黑色区域)被粗化为 1 个同样大小的 2 维块.粗粒度化使得每个阶段的每 1 个时间步都能处理 1 个 3 维块.例如,$\mathbb{B}_1$块的结束位置是一个区域而不是 1 条对角线.

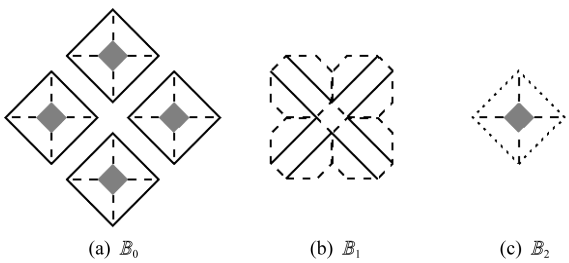


Fig. 9 Coarsening star tessellation

图 9 粗粒度化星型密铺

3.2 $\mathbb{B}_0$和$\mathbb{B}_2$块间重用

如图 9 所示,B_0 和 B_2 块在数据空间中具有同样的形状.因此可以将相邻时间片的 B_0 和 B_2 块合并,减少一次同步并增加数据重用.

此外,从图 9 中可以看出,2 个 B_0 在某一维度的距离等于 B_0 结束区域的大小,这使得 B_0 的开始区域和 B_2 的结束区域完全相同.

3.3 $\mathbb{B}_1$中 2 次更新技术

星型密铺一个重要的优势是可以在$\mathbb{B}_1$中连续 2 次更新元素.从图 5(b)中可以观察到,每个元素都将更新偶数次($T_{S_1}(x,y)=2y$).基于这一特征,我们对每个元素连续更新 2 次,这种实现技术称为 2 次更新技术,下面将从$\mathbb{B}_1$的更新过程中引出这个技术.$\mathbb{B}_1$常规更新方法为

$$\mathbb{B}_1=\begin{bmatrix}2&2&2\\2&4&2\\2&2&2\end{bmatrix}=\begin{bmatrix}1&&\\&1&\\&&1\end{bmatrix}+\begin{bmatrix}1&1&\\1&1&1\\&1&1\end{bmatrix}+\begin{bmatrix}&1&1\\1&1&1\\1&1&\end{bmatrix}+\begin{bmatrix}&&1\\&1&\\1&&\end{bmatrix}.$$

等号右边的块中依次包含了$\mathcal{B}_1$在连续 4 个时间步中每个时间步内更新的格点,即右边每个块中的格点在同一个时间步内更新.由于同一块中格点的更新次序是任意的,我们可以进一步将中间 2 个$\mathcal{B}_1$的块分解成为

$$\begin{bmatrix}1&1&\\1&1&1\\&1&1\end{bmatrix}\Rightarrow\begin{bmatrix}1&&\\&1&\\&&1\end{bmatrix}+\begin{bmatrix}&1&\\1&&1\\&1&\end{bmatrix},$$

$$\begin{bmatrix}&1&1\\1&1&1\\1&1&\end{bmatrix}\Rightarrow\begin{bmatrix}&1&\\1&&1\\&1&\end{bmatrix}+\begin{bmatrix}&&1\\&1&\\1&&\end{bmatrix}.$$

最后,将表示中间 2 个时间步更新格点的$\mathcal{B}_1$替换为分解后的形式,并将最左块(表示第 1 个时间步更新格点)和分解后的第 1 个块结合,将最右块(表示第 4 个时间步更新格点)和分解后的最后 1 个块结合,得到的执行过程如下所示,其中每个格点连续更新 2 次,有效提升核内执行效率.

$$\mathcal{B}_1=\begin{bmatrix}2&2&2\\2&4&2\\2&2&2\end{bmatrix}=\begin{bmatrix}2&&\\&2&\\&&2\end{bmatrix}+\begin{bmatrix}&2&\\2&&2\\&2&\end{bmatrix}+\begin{bmatrix}&&2\\&2&\\2&&\end{bmatrix}.$$

3.4　并行化和向量化

在同一阶段的所有块可以并发执行,因此很容易实现并行化密铺方案.因此,我们只在共享内存机器上简单使用 OpenMP 编译指示 parallel for.对于分布式内存计算机,清晰的密铺方案使得我们可以实现一个简单的数据/计算分布以及高效的数据通信,留做下一步工作.

星型密铺方案只适用于 2 维 Stencil,而且 B_1 的计算很难向量化,因此我们只对此方案在 3 维星型 Stencil 上的计算性能进行评估.因为 3 维 Stencil 中每个维度上的大小通常小于 1 000,我们选择不划分单位跨度维度来利用硬件预取的能力,采用 pragma simd 对最内层循环进行向量化.

3.5　代码实现

代码实现主体包含 5 层循环,如算法 1 所示.最外层循环控制时间维度上时间片的切换,与分块相关的时间片大小为 t_B,t_0 即为当前时间片的开始时间.内部包含 2 个 4 层循环,第 1 个循环用来进行 B_0 块和 B_2 块的更新,第 2 个循环更新 B_1 块.

在第 1 个循环中,第 1 层循环根据时间片层数的奇偶遍历所有 B_0 块或 B_2 块.$\mathcal{B}_2$结束时间维度即为下一个时间片$\mathcal{B}_0$开始时间维度,且共用相同的数据空间,可以将$\mathcal{B}_0$和$\mathcal{B}_2$合并为$\mathcal{B}_{02}$一起更新,除了迭代空间边界上的块,合并后的更新时间步长一般为 $2\times t_B$.第 2 层循环遍历$\mathcal{B}_{02}$所在的时间维度,t 代表时间维度值,跨度为 $2\times t_B$,但 t 的值会被时间维度的下界 0 和上界 T 所限定.求出$\mathcal{B}_{02}$块在特定时间上的数据空间区域位置即菱形边界 $x_{\min}$,$x_{\max}$,$y_{\min}$,$y_{\max}$后,在第 3 层和第 4 层循环中遍历更新$\mathcal{B}_{02}$对应数据空间区域中的格点.

第 2 个循环的第 1 层循环遍历所有 B_1 块,在星型密铺中,B_1 块是沿 2 个不同的对角线方向合并产生的,可以按照对角线方向的不同将 B_1 块进一步细分为 B_{11} 块和 B_{12} 块,对应的 d_y 值分别为 1 和 −1,同时根据 B_1 块的位置可以确定对应数据空间区域边界 $x_{\min}$,$y_{\max}$.第 2 层循环遍历 B_1 所在的时间维度,跨度为 t_B,并根据 t 计算特定时间维度上对角线区域边界 $x_{\max}$,$y_{\min}$.第 3 层循环遍历对角线区域中每条对角线,确定每条对角线的起始坐标 x,y 和结束横坐标 x_r.第 4 层循环遍历某一条对角线上的所有格点,每个格点被连续更新 2 次.

算法 1. 2D 星型 Stencil 算法.

输入:时间步长 t_B;迭代空间上下界 0,T;数据空间菱形边界 $x_{\min}$,$x_{\max}$,$y_{\min}$,$y_{\max}$;数据空间(t,x,y);

输出:更新后的数据空间(t,x,y).

① for $t_0=-t_B$ to T step t_B do
② 　#pragma omp parallel for private($x_{\min}$, $x_{\max}$,$y_{\min}$,$y_{\max}$,t,x,y)
③ 　for $n=0$ to B_0 or B_2 do
④ 　　for $t=\max(t_0,0)$ to $\min(t_0+2\times t_B$, T) do
⑤ 　　$set(x_{\min},x_{\max},y_{\min},y_{\max})$;
⑥ 　　$update(t,x,y)$;
　　/*在边界内更新 B_0 和 B_2*/
⑦ 　　end for
⑧ 　end for
⑨ 　#pragma omp parallel for private($x_{\min}$, $x_{\max}$,$y_{\min}$,$y_{\max}$,t,x,y,x_r,d_y,i)
⑩ 　for $n=0$ to B_{11} or B_{12} do
⑪ 　　$set(x_{\min},x_{\max},y_{\min},y_{\max},x,y,x_r,d_y)$;
⑫ 　　while $x<x_r$ do
⑬ 　　$update(t,x,y)$ twice;
⑭ 　　/*2 次更新 B_1 块*/
⑮ 　　end while
⑯ 　end for
⑰ end for

4 实验结果

4.1 实验环境

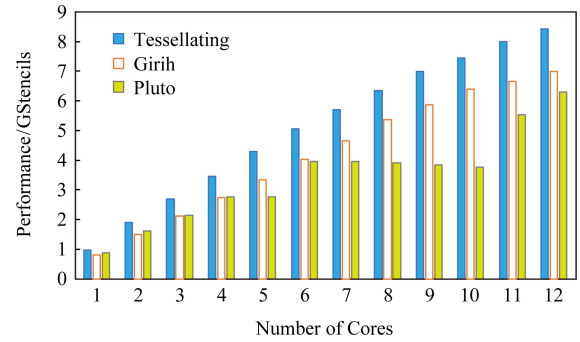
我们的实验运行在 Intel Xeon E5-2670 处理器上,处理器时钟频率为 2.70 GHz,实验规模由单核扩展到所有 12 核.每个核独享 32 KB L1 cache 和 256 KB L2 cache,12 个核共享 1 个统一的 30 MB L3 cache.使用版本号为 16.0.1 的 ICC 编译器编译程序并使用优化标志'-O3 -openmp'.

将我们的方案与 1.1 节提及的 2 种高并发方案 Pluto 和 Girih 进行比较.盒型密铺、Pochoir 和 SDSL (Stencil domain specific language)[56]因其较差的性能结果(特别是 3 维 Stencil)不作考虑.测试用例包括 3 种 3 维星型 Stencil:1 阶(3D7P)、2 阶(3D13P)和 4 阶(3D25P)的具有非周期性边界条件的情况.测试的问题规模为 $256^3 \times 500$, $384^3 \times 800$, $512^3 \times 1\,000$.

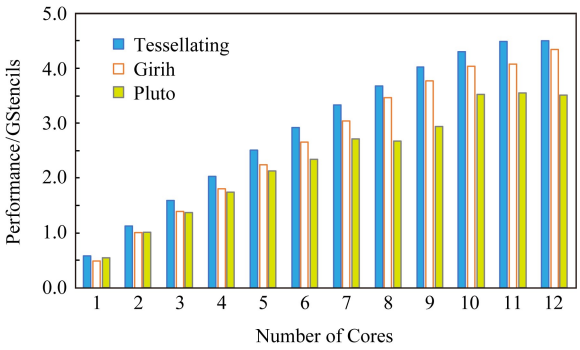
Girih 提供一种自动调优组件来选择不同线程数时的最优分块大小.对于 Pluto 和星型密铺方案,我们将测试所有可能的分块大小,然后选择 12 核时性能最优的分块大小并应用于所有 1～12 核测试中.

4.2 结果分析

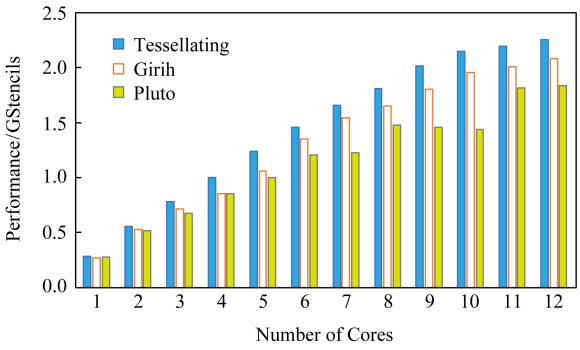
图 10～12 分别显示了星型密铺(深色柱)、Girih(白色柱)和 Pluto(浅色柱)在 3D7P, 3D13P, 3D25P 星型 Stencil 上大小为 256^3, 384^3, 512^3 时的性能.



(a) 3D7P



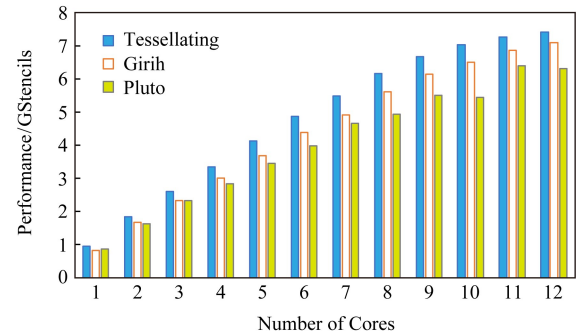
(b) 3D13P



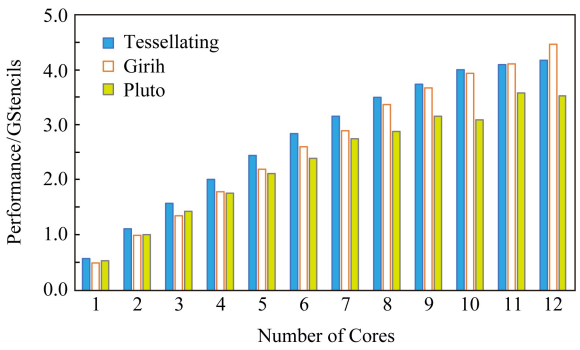
(c) 3D25P

Fig. 10 Performance of size 256^3

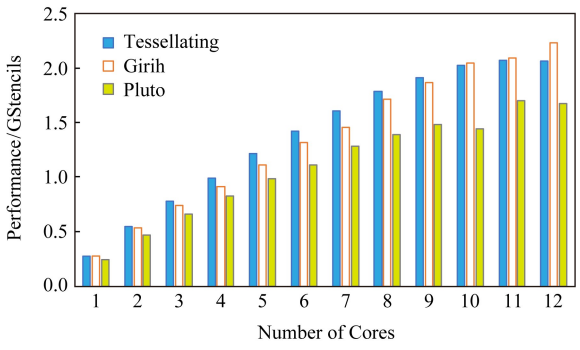
图 10 问题规模为 256^3 的性能



(a) 3D7P



(b) 3D13P

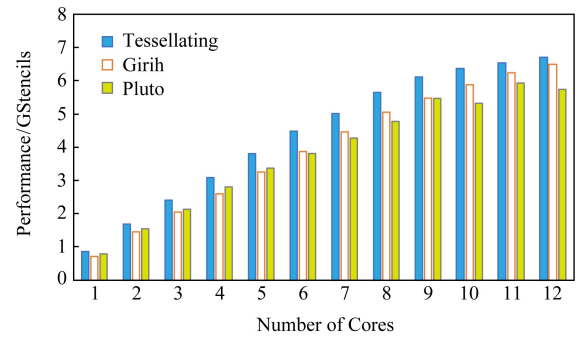


(c) 3D25P

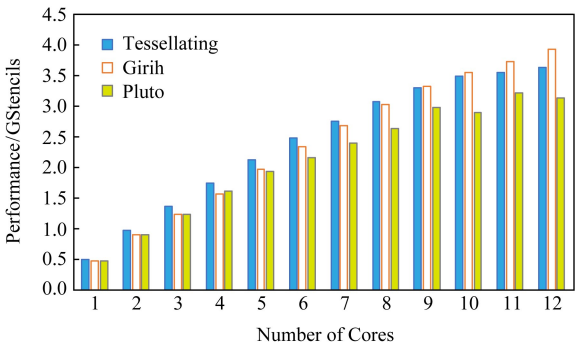
Fig. 11 Performance of size 384^3

图 11 问题规模为 384^3 的性能

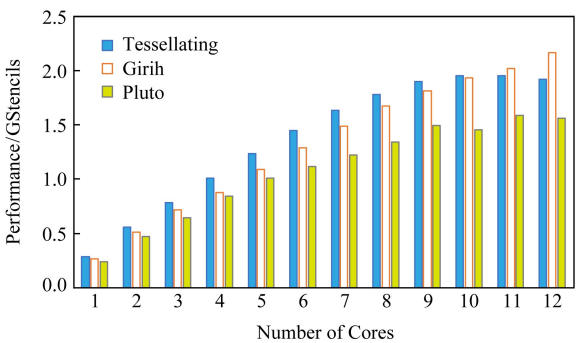
其中性能 GStencil 的定义为:以最内层循环计算量看为一个 Stencil,单位时间完成给定的 Stencil 迭代更新计算的次数为 GStencil.由图 10～12 中可以明显看出,Pluto 对应的性能条柱性能都比我们的代码和 Girih 低,且其结果不是一条直线,特别是在 6 核或更多核时.这和文献[29]中的结果一致.因此,下面主要分析星型密铺方案和 Girih.



(a) 3D7P



(b) 3D13P



(c) 3D25P

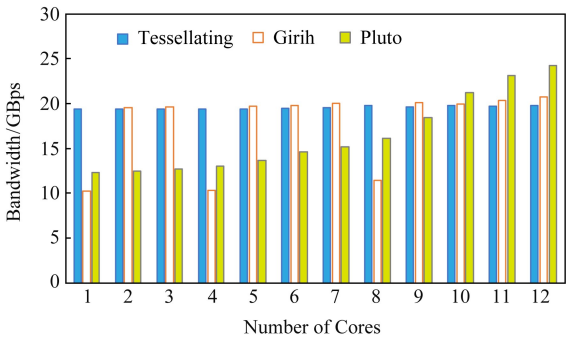
Fig. 12　Performance of size 512^3

图 12　问题规模为 512^3 的性能

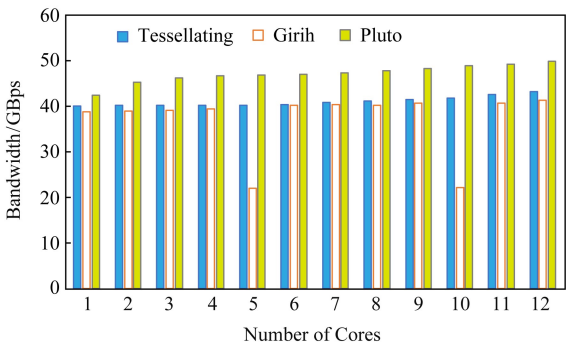
在问题规模大小为 256^3 的 3D7P 星型 Stencil 中,星型密铺方案性能比 Girih 高,相比于 Girih,星型密铺方案在 2 核时最高加速 1.24 倍,平均加速 1.19 倍.对于图 10 中更高阶的 Stencil 类型,我们的方案仍保持较高的性能,但随着阶数增加,这 3 种方法对应的性能柱变得更加接近,因为对于给定的自然块大小,更高阶的 Stencil 会产生更小的时间维度分块,造成更大的内存数据传输,后面将对此进行定量分析.

类似地,问题规模变大时 3 条性能曲线也会相互接近,也是因为 Pluto 和我们的密铺方案不对单位跨度的维度进行分块,导致时间维度分块大小受限.我们的方案在核数较少时仍取得了性能提升,在 8～12 核时虽然性能提升速度比 Girih 慢,但仍达到了可相比的性能.例如,在大小为 384^3 的 3D13P 星型 Stencil 的实验结果中,星型密铺和 Girih 在 12 核上的性能分别为 4.2,4.4 GStencil.

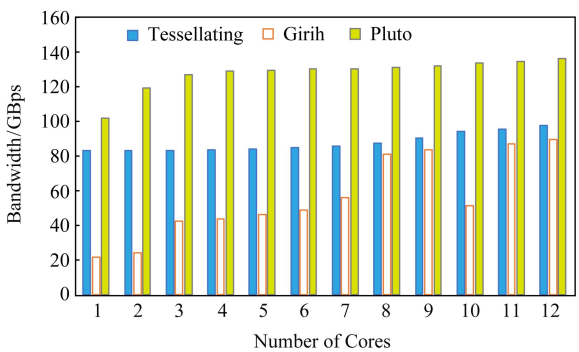
图 13～15 中显示了对应的最后一级 cache 和内存之间数据传输量.Girih 利用整个 L3 cache 来最大限度开发数据局部性,因此具有最小的数据传输量.



(a) 3D7P



(b) 3D13P

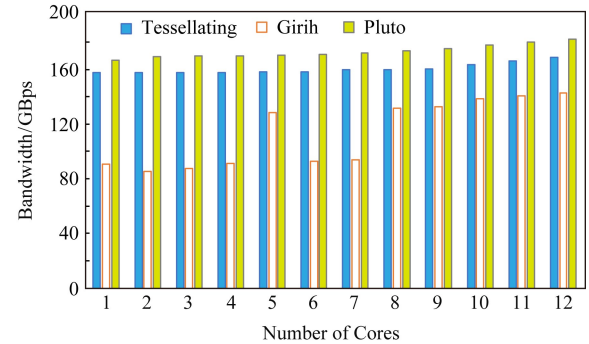


(c) 3D25P

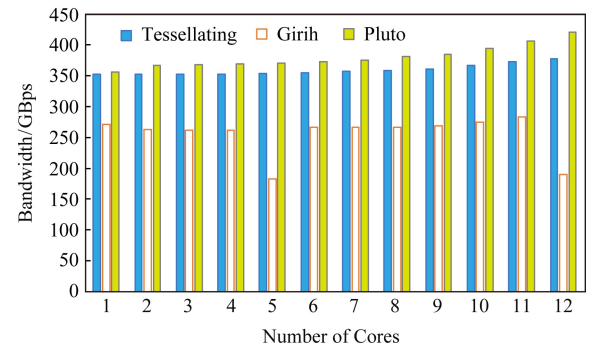
Fig. 13　Memory transfers of size 256^3

图 13　问题规模为 256^3 的数据传输量

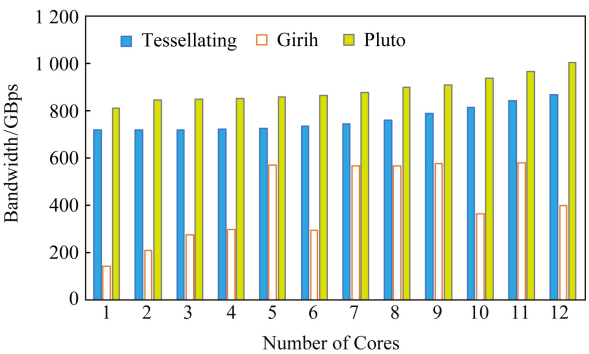
Girih 在 9 个测试中的数据传输量都不规则变化,与密铺或 Pluto 的变化曲线不同,因为自动调优方案在不同核数时确定的分块大小不一样,例如,大小为 256^3 的 3D7P Stencil 的 Girih 测量值在 4 核上是 10.2 GBps,6 核上是 19.8 GBps,相应的菱形分块宽度最优值分别为 32 和 16.Girih 事实上是菱形分块和波前并行的混合方法,并采用细粒度并行,所有线程同时处理一个块,因此我们不对它进行理论分析.



(a) 3D7P



(b) 3D13P

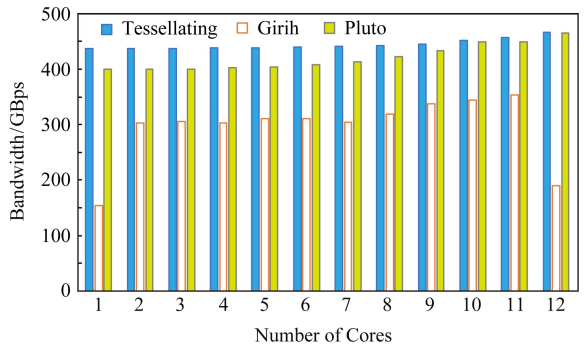


(c) 3D25P

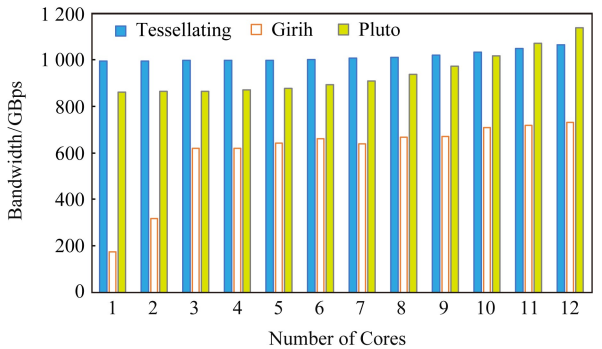
Fig. 14 Memory transfers of size 384^3

图 14 问题规模为 384^3 的数据传输量

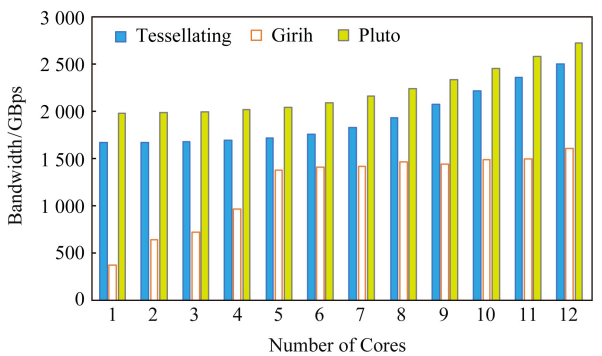
对于 3D25P Stencil,星型密铺方案在 3 个问题规模上的最优分块大小分别是 36×4,28×3,28×3,对应自然块大小分别为 3.9 MB,4 MB,5.3 MB.因此 30 MB 的 L3 cache 将分别在 8 核,8 核和 6 核时被完全使用,对应图 13(c)~15(c)中星型密铺方案



(a) 3D7P



(b) 3D13P



(c) 3D25P

Fig. 15 Memory transfers of size 512^3

图 15 问题规模为 512^3 的数据传输量

的数据传输量开始增长的位置.尽管核数更大时超出了 L3 cache 大小,但由于高阶依赖,数据块将快速收缩.此外,用时间维度块大小为 2 的分块密铺方案等价于原始算法,因此它们产生同样的缓存-内存传输量即 $T\times N^2$.Pluto 在 3D25P Stencil 上也有类似地变化趋势和原因.

对于另外 2 个低阶 Stencil,L3 cache 能保留所有 12 个核的自然块.例如大小为 256^3 的 3D7P Stencil 中自然块占用内存为 2.5 MB,12 核将占用所有 30 MB L3 cache.因此,星型密铺的带宽变化接近于水平.但是性能在核数变多时增长变缓,这可能是由于有限的 L3 带宽.

最后比较盒型密铺的性能.总体上看盒型密铺和 Girih 性能相近.例如在大小为 256^3 的 3D7P Stencil 中,盒型密铺的最高性能是 6.9 GStencils 而 Girih 是 7.0 GStencil.盒型密铺对应的块大小是 23×11,约为星型密铺最优块大小 33×16 的 $\sqrt{2}$ 倍.分块大小使得这 2 种方案都在 12 核时完全占用 L3 cache.星型和盒型密铺数据传输量的测量值分别为 19.3 GBps 和 26.4 GBps,同样有 $19.3\times\sqrt{2}\approx26.4$ 的关系.

4 总　　结

本文提出了一个新的并行框架,能够高并发执行星型 Stencil,并以"自然块"的新概念来标识不同形状 Stencil 的特征.自然块及其后继块能够密铺数据空间,并且它们沿着时间维度扩展后能形成迭代空间的密铺.我们的方案能够导出简洁的并行框架,揭示一个格点的坐标和它更新时间步之间的关系.此外,专为星型 Stencil 研发了一个新颖的实现方式即"2 次更新",能够在第 1 个后继块的执行过程中对每个元素更新 2 次,从而改进核内数据重用模式.实验结果证明了此方法的有效性.

未来我们将着重于自动调优方法来高效寻找最优块大小,基于提出的框架设计一个工具来自动生成 Stencil 代码.

参　考　文　献

[1] Ding C, He Yun. A ghost cell expansion method for reducing communications in solving pde problems [C] //Proc of the ACM/IEEE SC'01. New York: ACM, 2001: 55-55

[2] Nguyen A, Satish N, Chhugani J, et al. 3.5-d blocking optimization for Stencil computations on modern CPUs and GPUs [C] //Proc of the ACM/IEEE SC'10. New York: ACM, 2010: 1-13

[3] Rastello F, Dauxois T. Efficient tiling for an ode discrete integration program: Redundant tasks instead of trapezoidal shaped-tiles [C] //Proc of the IEEE IPDPS'02. Piscataway, NJ: IEEE, 2002: 1-8

[4] Rivera G, Tseng C W. Tiling optimizations for 3d scientific computations [C] //Proc of the 2000 ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 2000: 32-32

[5] Christen M, Schenk O, Burkhart H. Patus: A code generation and autotuning framework for parallel iterative Stencil computations on modern microarchitectures [C] //Proc of the 25th IEEE Int Symp on Parallel and Distributed Processing. Piscataway, NJ: IEEE, 2011: 676-687

[6] Gysi T, Grosser T, Hoefler T. Modesto: Data-centric analytic optimization of complex Stencil programs on heterogeneous architectures [C] //Proc of the 29th ACM Int Conf on Supercomputing. New York: ACM, 2015: 177-186

[7] Kamil S, Chan C, Oliker L, et al. An auto-tuning framework for parallel multicore Stencil computations [C] //Proc of 2010 IEEE Int Symp on Parallel & Distributed Processing. Piscataway, NJ: IEEE, 2010: 1-12

[8] Luo Yulong, Tan Guangming, Mo Zeyao, et al. Fast: A fast Stencil autotuning framework based on an optimal-solution space model [C] //Proc of the 29th ACM on Int Conf on Supercomputing. New York: ACM, 2015: 187-196

[9] Lutz T, Fensch C, Cole M. Partans: An autotuning framework for Stencil computation on multi-GPU systems [J]. ACM Transactions on Architecture and Code Optimization, 2013, 9(4): No.59

[10] Mametjanov A, Lowell D, Ma C C, et al. Autotuning Stencil-based computations on GPUs [C] //Proc of 2012 IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2012: 266-274

[11] Rocha R C O, Pereira A D, Ramos L, et al. Toast: Automatic tiling for iterative Stencil computations on GPUs [J]. Concurrency and Computation: Practice and Experience, 2017, 29(8): No.e4053

[12] Zhang Yongpeng, Mueller F. Auto-generation and auto-tuning of 3d Stencil codes on GPU clusters [C] //Proc of the 10th Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2012: 155-164

[13] Maruyama N, Nomura T, Sato K, et al. Physis: An implicitly parallel programming model for Stencil computations on large-scale GPU-accelerated supercomputers [C] //Proc of the 2011 Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2011: No.11

[14] Unat D, Cai Xing, Baden S B. Mint: Realizing cuda performance in 3d Stencil methods with annotated c [C] //Proc of the 25th Int Conf on Supercomputing. New York: ACM, 2011: 214-224

[15] Datta K, Murphy M, Volkov V, et al. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures [C] //Proc of the 2008 ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 2008: No.4

[16] Prieto M, Llorente I M, Tirado F. Data locality exploitation in the decomposition of regular domain problems [J]. IEEE Transactions on Parallel and Distributed Systems, 2000, 11(11): 1141-1150

[17] Venkatasubramanian S, Vuduc R W. Tuned and wildly asynchronous Stencil kernels for hybrid CPU/GPU systems [C] //Proc of the 23rd Int Conf on Supercomputing. New York: ACM, 2009: 244-255

[18] Williams S, Oliker L, Carter J, et al. Extracting ultra-scale lattice boltzmann performance via hierarchical and distributed auto-tuning [C] //Proc of the ACM/IEEE SC'11. New York: ACM, 2011: No.55

[19] Christen M, Schenk O, Neufeld E, et al. Parallel data-locality aware Stencil computations on modern micro-architectures [C] //Proc of the IEEE IPDPS'09. Piscataway, NJ: IEEE, 2009

[20] Krotkiewski M, Dabrowski M. Efficient 3d Stencil computations using CUDA [J]. Parallel Computing, 2013, 39(10): 533-548

[21] Olschanowsky C, Strout M M, Guzik S, et al. A study on balancing parallelism, data locality, and recomputation in existing pde solvers [C] //Proc of the 2014 Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2014: 793-804

[22] Phillips E H, Fatica M. Implementing the himeno benchmark with CUDA on GPU clusters [C] //Proc of the IEEE IPDPS'10. Piscataway, NJ: IEEE, 2010

[23] Rawat P S, Hong Changwan, Ravishankar M, et al. Effective resource management for enhancing performance of 2d and 3d Stencils on GPUs [C] //Proc of the 9th Annual Workshop on General Purpose Processing Using Graphics Processing Unit. New York: ACM, 2016: 92-102

[24] Heybrock S, Joo B, Kalamkar D D, et al. Lattice qcd with domain decomposition on Intel xeon phi co-processors [C] //Proc of the ACM/IEEE SC'14. New York: ACM, 2014: 69-80

[25] Yount C, Duran A. Effective use of large high-bandwidth memory caches in HPC Stencil computation via temporal wave-front tiling [C] //Proc of the 7th Int Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems. Piscataway, NJ: IEEE, 2016: 65-75

[26] Bondhugula U, Hartono A, Ramanujam J, et al. A practical automatic polyhedral parallelizer and locality optimizer [C] //Proc of the 29th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2008: 101-113

[27] Bondhugula U, Bandishti V, Cohen A, et al. Tiling and optimizing time-iterated computations on periodic domains [C] //Proc of the 23rd Int Conf on Parallel Architecture and Compilation Techniques. New York: ACM, 2014: 39-50

[28] Krishnamoorthy S, Baskaran M, Bondhugula U, et al. Effective automatic parallelization of Stencil computations [C] //Proc of the ACM SIGPLAN 2007 Conf on Programming Language Design and Implementation. New York: ACM, 2007: 235-244

[29] Bandishti V, Pananilath I, Bondhugula U. Tiling Stencil computations to maximize parallelism [C] //Proc of the Int Conf on High Performance Computing Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2012: No.40

[30] Frigo M, Strumpen V. Cache oblivious Stencil computations [C] //Proc of the 19th Annual Int Conf on Supercomputing. New York: ACM, 2005: 361-366

[31] Frigo M, Strumpen V. The cache complexity of multithreaded cache oblivious algorithms [C] //Proc of the 18th Annual ACM Symp on Parallelism in Algorithms and Architectures. New York: ACM, 2006: 271-280

[32] Strzodka R, Shaheen M, Pajak D, et al. Cache oblivious parallelograms in iterative Stencil computations [C] //Proc of the 24th ACM Int Conf on Supercomputing. New York: ACM, 2010: 49-59

[33] Tang Yuan, Chowdhury R A, Kuszmaul B C, et al. The pochoir Stencil compiler [C] //Proc of the 23rd Annual ACM Symp on Parallelism in Algorithms and Architectures. New York: ACM, 2011: 117-128

[34] Yuan Liang, Zhang Yunquan, Guo Peng, et al. Tessellating Stencils [C] //Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2017: No.49

[35] Irigoin F, Triolet R. Supernode partitioning [C] //Proc of the 15th Annual ACM Symp on Principles of Programming Languages. New York: ACM, 1988: 319-329

[36] Lam M D, Rothberg E 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

[37] McKellar A C, Coffman E G Jr. Organizing matrices and matrix operations for paged memory systems [J]. Communications of the ACM, 1969, 12(3): 153-165

[38] Wolf M E, Lam M S. A data locality optimizing algorithm [C] //Proc of the ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 1991: 30-44

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

[40] Wonnacott D G, Strout M M. On the scalability of loop tiling techniques [C] //Proc of the 3rd Int Workshop on Polyhedral Compilation Techniques. New York: ACM, 2013: 3-11

[41] Meng J, Skadron K. Performance modeling and automatic ghost zone optimization for iterative Stencil loops on GPUs [C] //Proc of the 23rd Int Conf on Supercomputing. New York: ACM, 2009: 256-265

[42] 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. New York: ACM, 2012: 311-320

[43] Demmel J, Hoemmen M F, Mohiyuddin M, et al. Avoiding communication in computing krylov subspaces [OL]. [2020-02-10]. http://citeseerx.ist.psu.edu/

[44] Mohiyuddin M, Hoemmen M, Demmel J, et al. Minimizing communication in sparse matrix solvers [C] //Proc of the Conf on High Performance Computing Networking, Storage and Analysis. New York: ACM, 2009: No.36

[45] Jin Guohua, Mellor-Crummey J, Fowler R. Increasing temporal locality with skewing and recursive blocking [C] //Proc of the 2001 ACM/IEEE Conf on Supercomputing. New York: ACM, 2001: No.43

[46] Song Yonghong, Li Zhiyuan. New tiling techniques to improve cache temporal locality [C] //Proc of the 1999 ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 1999: 215-228

[47] Wonnacott D. Achieving scalable locality with time skewing [J]. International Journal of Parallel Programming, 2002, 30 (3): 181-221

[48] Malas T, Hager G, Ltaief H, et al. Multicore-optimized wavefront diamond blocking for optimizing Stencil updates [J]. SIAM Journal on Scientific Computing, 2015, 37(4): C439-C464

[49] Wellein G, Hager G, Zeiser T, et al. Efficient temporal blocking for Stencil computations by multicore-aware wavefront parallelization [C] //Proc of the 33rd Annual IEEE Int Computer Software and Applications Conf. Piscataway, NJ: IEEE, 2009: 579-586

[50] Pananilath I, Acharya A, Vasista V, et al. An optimizing code generator for a class of lattice-boltzmann computations [J]. ACM Transactions on Architecture & Code Optimization, 2015, 12(2): No.14

[51] Grosser T, Verdoolaege S, Cohen A, et al. The relation between diamond tiling and hexagonal tiling [J]. Parallel Processing Letters, 2014, 24(3): No.1441002

[52] Cui Huimin, Wang Lei, Fan Dongrui, et al. Landing Stencil code on godson-t [J]. Journal of Computer Science and Technology, 2010, 25(4): 886-894

[53] Wonnacott D. Using time skewing to eliminate idle time due to memory bandwidth and network limitations [C] //Proc of the 14th Int Parallel and Distributed Processing Symposium. Piscataway, NJ: IEEE, 2000: 171-180

[54] Zhou Xing, Giacalone J P, Garzarán M J, et al. Hierarchical overlapped tiling [C] //Proc of the 10th Int Symp on Code Generation and Optimization. New York: ACM, 2012: 207-218

[55] Grosser T, Cohen A, Kelly P H J, 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

[56] Henretty T, Veras R, Franchetti F, et al. A Stencil compiler for short-vector simd architectures [C] //Proc of the 27th Int ACM Conf on Int Conf on Supercomputing. New York: ACM, 2013: 13-24

[57] Strzodka R, Shaheen M, Pajak D, et al. Cache accurate time skewing in iterative Stencil computations [C] //Proc of 2011 Int Conf on Parallel Processing. Piscataway, NJ: IEEE, 2011: 571-581

[58] Grosser T, Cohen A, Holewinski J, et al. Hybrid hexagonal/classical tiling for GPUs [C] //Proc of Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2014: No.66

[59] Basu P, Hall M, Williams S, et al. Compiler-directed transformation for higher-order Stencils [C] //Proc of 2015 IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2015: 313-323

[60] Deitz S J, Chamberlain B L, Snyder L. Eliminating redundancies in sum-of-product array computations [C] //Proc of the 15th Int Conf on Supercomputing. New York: ACM, 2001: 65-77

[61] de la Cruz R, Araya-Polo M. Algorithm 942: Semi-Stencil [J]. ACM Transactions on Mathematical Software, 2014, 40 (3): 23:1-23:39

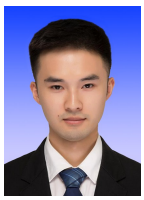
[62] Jin Mengyao, Fu Haohuan, Lü Zihong, et al. Libra: An automated code generation and tuning framework for register-limited Stencils on GPUs [C] //Proc of the ACM Int Conf on Computing Frontiers. New York: ACM, 2016: 92-99

[63] Sedaghati N, Thomas R, Pouchet L N, et al. Stvec: A vector instruction extension for high performance Stencil computation [C] //Proc of 2011 Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE. 2011: 276-287

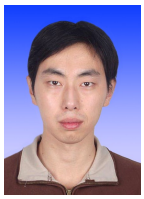
[64] Zumbusch G. Vectorized higher order finite difference kernels [C] //Proc of 2012 Int Workshop on Applied Parallel Computing. Berlin: Springer, 2012: 343-357

[65] Hadade I, Mare L. Exploiting simd and thread-level parallelism in multiblock CFD [C] //Proc of the 29th Int Supercomputing Conf. Berlin: Springer, 2014: 410-419

[66] Henretty T, Stock K, Pouchet L N, et al. Data layout transformation for Stencil computations on short-vector simd architectures [C] //Proc of 2011 Int Conf on Compiler Construction. Berlin: Springer, 2011: 225-245



Cao Hang, born in 1992. PhD from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His main research interests include large-scale parallel computing and heterogeneous parallel computing performance optimization.

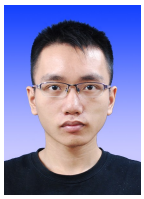


Yuan Liang, born in 1984. PhD, assistant professor with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Member of CCF. His main research interests include large-scale parallel computing and heterogeneous computing.

Huang Shan, born in 1994. MSc degree in the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Her main research interests include parallel computing model and parallel computing algorithm performance optimization.

Zhang Yunquan, born in 1973. PhD, professor of computer science with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Senior member of CCF. His main research interests include high performance parallel computing, big data processing.

Xu Yongjun, born in 1979. PhD, PhD supervisor, professor level senior engineer with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His main research interests include big data processing, artificial intelligence, IoT.



Lu Pengqi, born in 1995. Master candidate in the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His main research interests is high performance computing.

Zhang Guangting, born in 1987. Master, assistant professor with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Her main research interest is high performance computing.

2018 年《计算机研究与发展》高被引论文 TOP10

排名	论文信息
1	李然，林政，林海伦，王伟平，孟丹. **文本情绪分析综述**[J]. 计算机研究与发展，2018，55(1)：30-52 Li Ran, Lin Zheng, Lin Hailun, Wang Weiping, Meng Dan. Text Emotion Analysis: A Survey [J]. Journal of Computer Research and Development, 2018, 55(1): 30-52
2	吕华章，陈丹，范斌，王友祥，乌云霄. **边缘计算标准化进展与案例分析**[J]. 计算机研究与发展，2018，55(3)：487-511 Lü Huazhang, Chen Dan, Fan Bin, Wang Youxiang, Wu Yunxiao. Standardization Progress and Case Analysis of Edge Computing [J]. Journal of Computer Research and Development, 2018, 55(3): 487-511
3	赵梓铭，刘芳，蔡志平，肖侬. **边缘计算：平台、应用与挑战**[J]. 计算机研究与发展，2018，55(2)：327-337 Zhao Ziming, Liu Fang, Cai Zhiping, Xiao Nong. Edge Computing: Platforms, Applications and Challenges [J]. Journal of Computer Research and Development, 2018, 55(2): 327-337/
4	陈珂，梁斌，柯文德，许波，曾国超. **基于多通道卷积神经网络的中文微博情感分析**[J]. 计算机研究与发展，2018，55(5)：945-957 Chen Ke, Liang Bin, Ke Wende, Xu Bo, Zeng Guochao. Chinese Micro-Blog Sentiment Analysis Based on Multi-Channels Convolutional Neural Networks [J]. Journal of Computer Research and Development, 2018, 55(5): 945-957
5	余永红，高阳，王皓，孙栓柱. **融合用户社会地位和矩阵分解的推荐算法**[J]. 计算机研究与发展，2018，55(1)：113-124 Yu Yonghong, Gao Yang, Wang Hao, Sun Shuanzhu. Integrating User Social Status and Matrix Factorization for Item Recommendation [J]. Journal of Computer Research and Development, 2018, 55(1): 113-124
6	贺海武，延安，陈泽华. **基于区块链的智能合约技术与应用综述**[J]. 计算机研究与发展，2018，55(11)：2452-2466 He Haiwu, Yan An, Chen Zehua. Survey of Smart Contract Technology and Application Based on Blockchain [J]. Journal of Computer Research and Development, 2018, 55(11): 2452-2466
7	齐彦丽，周一青，刘玲，田霖，石晶林. **融合移动边缘计算的未来 5G 移动通信网络**[J]. 计算机研究与发展，2018，55(3)：478-486 Qi Yanli, Zhou Yiqing, Liu Ling, Tian Lin, Shi Jinglin. MEC Coordinated Future 5G Mobile Wireless Networks [J]. Journal of Computer Research and Development, 2018, 55(3): 478-486
8	谢振平，金晨，刘渊. **基于建构主义学习理论的个性化知识推荐模型**[J]. 计算机研究与发展，2018，55(1)：125-138 Xie Zhenping, Jin Chen, Liu Yuan. Personalized Knowledge Recommendation Model Based on Constructivist Learning Theory [J]. Journal of Computer Research and Development, 2018, 55(1): 125-138
9	陈伟利，郑子彬. **区块链数据分析：现状、趋势与挑战**[J]. 计算机研究与发展，2018，55(9)：1853-1870 Chen Weili, Zheng Zibin. Blockchain Data Analysis: A Review of Status, Trends and Challenges [J]. Journal of Computer Research and Development, 2018, 55(9): 1853-1870
10	邓晓衡，关培源，万志文，刘恩陆，罗杰，赵智慧，刘亚军，张洪刚. **基于综合信任的边缘计算资源协同研究**[J]. 计算机研究与发展，2018，55(3)：449-477 Deng Xiaoheng, Guan Peiyuan, Wan Zhiwen, Liu Enlu, Luo Jie, Zhao Zhihui, Liu Yajun, Zhang Honggang. Integrated Trust Based Resource Cooperation in Edge Computing [J]. Journal of Computer Research and Development, 2018, 55(3): 449-477

数据来源：CSCD，中国知网；统计日期：2019 年 12 月 23 日