Intel Xeon Phi 平台上 HGGF 程序实现与优化

许志耿[1] 邱君仪[1] 韦跃明[1] 董一哲[1] 韦建文[1] 林新华[1,2]
[1]上海交通大学高性能计算中心，上海 200240
[2]NVIDIA Technology Center Asia Pacific
(james@sjtu.edu.cn)

Implementation and Optimization of HGGF Application on Intel Xeon Phi Platform

Zhigeng Xu[1] ,Junyi Qiu[1] ,Yueming Wei[1] ,Yizhe Dong[1],Jianwen Wei[1],James Lin[1,2]
[1](Center for High Performance Computing, Shanghai Jiao Tong University, shanghai 200240, China)
[2](NVIDIA Technology Center Asia Pacific)

Abstract We implemented the application of Halo-based Galaxy Group Finder on the Intel Xeon Phi platform and conducted paralleling optimizations to enable efficient processing on the massive observation data collected from Digital Sky Survey project. First, we accelerated the kernel part of the application using TLP (thread-level parallelism) and DLP (data-level parallelism) optimization methods on Intel Xeon Phi platform. These two classic means take fully advantages of the many cores and the vector process units of MIC cards. Next, in view of the irregular memory-access behavior of the HGGF application, we improved the memory access efficiency via various optimization methods including data pre-arrangement, memory copy reduction, date placement reconstruction on false-sharing L2 cache of MIC cards, etc. Finally we got performance boost of around 62 times on kernel. To the best of our knowledge, the followings are our original contributions: a. To port the HGGF application to the Intel Xeon Phi platform and explore parallelism, b. To take the first step of vectorization on the MIC card using intrinsics supported by KNC architecture, c. To conduct targeted memory-access optimization methods based on the memory hierarchy architecture of MIC card.

Key words Halo-based Galaxy Group Finder; Intel Xeon Phi; parallel computing; memory-access optimization

摘要 我们在 Intel Xeon Phi 平台上实现了 Halo-based Galaxy Group Finder 应用，并且通过并行优化使其能高效地处理数字化巡天项目中得到的海量观测数据。首先，我们在 MIC 平台上对算法的核心计算部分进行了线程级并行优化、数据级并行优化。这是两个经典的 MIC 上优化方法，分别利用了 MIC 卡的众多核心与向量处理单元。此外，我们根据应用的非规则访存特性，做了多种内存访问优化，包括数据预排序、减少内存复制、优化数据在 MIC 卡伪共享的二级缓存上的布局等，最终在程序热点部分获得了 62 倍左右的性能提升。本文的主要贡献有：（1）将 HGGF 应用移植到 Intel Xeon Phi 平台上，并进行了并行优化；（2）通过使用 KNC 架构的 intrinsic，初步在 MIC 卡上做向量化并行且做了向量化评估；（3）根据 MIC 卡的内存、缓存结构，实现了有针对性的访存优化。

关键词 天文星系群查找；Intel Xeon Phi 平台；并行计算；访存优化

---

*本文受国家高技术研究发展计划(863)“高性能计算环境应用服务优化关键技术研究”2014AA01A302 及日本学术振兴会 RONPAKU Fellowship 资助
作者简介：许志耿，男，硕士研究生，主要研究方向：体系结构和高性能计算。邱君仪，男，本科生，主要研究方向：体系结构和高性能计算。论文所属领域：高性能计算应用
通信地址：上海市闵行区东川路 800 号，上海交通大学网络信息中心 203，邮编：200240

1. 背景介绍

近年来，人类的天文观测能力与观测规模获得了极大提升，多个大型数字巡天项目已经在全球范围内展开。目前全球天文观测数据总量已经达到 10^{15}B 量级。一些规划中的项目如 SKA 等，其预期产生的数据量也达到了 EB 量级。一方面，这些产生的海量数据有助于对天文现象、天体动力、宇宙结构等领域的研究，是极其丰富的资料；另一方面，如何高效地筛选、加工、分析这大规模的数据也成为了当前研究的重要课题之一。

我们实现并优化的应用 Halo-based Galaxy Group Finder[1]就是用于处理在斯隆数字化巡天项目中得到的海量天文学数据[13]，通过计算分析对暗物质晕中的星系进行聚类。程序使用的算法是星系聚类算法的一个改进版本。在空间距离比较的基础上，通过进一步考察星系的红移、亮度、暗物质晕的大小、质量等属性[1][6]，比此前传统聚类算法更加准确高效。

在该算法中结合了确定潜在星系群的中心与估计其特征亮度的两个方法。通过迭代方法，程序根据每个星系群在前次迭代得到的平均质量-亮度比，对它们预设星群质量。该质量进一步用于估计包围着此星系群的暗物质晕属性，如大小差异、速度差异等。这些能够确定在红移空间中，星系群所包含的星系成员。最后，每个星系群都被赋予两个暗物质质量，一个由其特征亮度得出，另一个由其特征星体质量特出。

Intel Xeon Phi 协处理器基于集成众核(MIC)架构设计，含有约 60 个核心，每个核心有 4 个硬件线程。此外，MIC 卡支持宽度为 512 位的向量处理单元，即能够同时进行 16 个单精度浮点数或 8 个双精度浮点数的运算。它的效率接近 SSE 指令的 4 倍，AVX 指令的 2 倍。为了使程序能在 MIC 平台上得到更高的性能，我们使用了针对 MIC 卡的多种经典优化手段——线程级并行优化、数据级并行优化和内存访问优化，使得热点部分的性能提升了 62 倍。

本文剩余篇幅安排：在第二节，我们将介绍应用优化的相关工作；在第三节将简要阐述 HGGF 应用的算法并作性能分析；从第四节将详细讨论我们所使用的优化方法并作评估；第五节对优化结果进行分析；最后总结优化工作以及提出下一步工作展望。

2. 相关研究工作

Intel Xeon Phi 协处理器由于具有强大的计算能力，同时兼容 x86，支持 MPI 和 OpenMP 等编程模型而受到学术界和工业界的关注。目前有许多科研人员致力于探索 MIC 的众核计算能力和性能。例如，S. Saini 等对 Xeon Phi 进行了初期的性能测试和优化手段探究[5]。X. Liu 等针对 SpMV 在 ELLPACK 基础上，提出了 ESB 存储格式，提高了向量化效率,从而在 MIC 上高效率地实现了 SpMV[2]。Wu Q.等通过分级并行的方法实现了分子动力学在 CPU-MIC 上的高效并行方案[3]。

在MIC异构系统中，我们可以利用现有的的编程模型开发并行应用，但是简单地增加编译选项、仅采用传统的并行编程模型在MIC上并不能得到出色的优化效果。为了更充分挖掘MIC的计算能力，需要针对MIC的硬件体系架构采取相应的优化措施。MIC上的优化通常可以在以下几个方面进行考虑。

数据传输优化：在CPU和MIC协同运算的模式下，程序运行中处理器之间的数据传输占据了大部分额外负载。如果我们可以降低通信负载，并行程序的运行时间就会显著地缩短。由于MIC和CPU间的通信是通过较低传输速率的PCI-E实现，减少CPU和MIC从核间的通信可以有效地减少额外通信负载。

向量化：MIC 处理器支持 512bit 的 KNC 向量处理指令，可以同时进行 8 个双精度或者 16 个单精度浮点运算。但是并非所用程序都能适合用 SIMD 指令向量化。多媒体加工、数字信号处理、大规模矩阵运算等应用可以通过充分的向量化提升性能。在生物基因分析、分子动力模拟等领域的应用也可应从向量化领域中获益。而对于一些包含大量分支判断的算法则不容易实现向量化，比如一些与图搜索相关的算法。HGGF 应用属于后者，这类应用在数据级并行上有很大的挑战。

负载均衡：我们在两个层面考虑负载均衡：第一层是 CPU 和 MIC 间的负载均衡，另外一层是 MIC 内众核间的负载均衡。CPU 和 MIC 间的负载均衡可以通过创建不同的进程，根据 CPU 和 MIC 的计算能力分配线程数目。众核间的负载可以通过调整映射和调度方式来保证所有的计算任务尽可能均衡地分配在每个核上。

3. HGGF 应用算法与性能分析

3.1 HGGF 算法

HGGF 应用主要由四个部分组成：数据预处理与格式化、星系坐标转换与 L 空间初始化、星系成团筛选、搜索结果整理与输出。我们将对第二、三部分进行介绍以阐明该应用的算法原理与工作机制。

3.1.1 星系坐标转换与 L 空间初始化

HGGF 应用的算法数据结构主要依赖于数组，其

中大量的链表结构也通过数组实现。经过对输入数据的预处理之后，程序将原本以极坐标形式表示的星系转换为以三维空间坐标表示，同时将星系其余属性信息存储于主数组[8]之中。在此基础上，为了减少对远处的、从经验上分析不可能属于同一星系组的星系的计算，程序首先将星系所在的三维空间通过如公式（1）[10]所示的坐标转换与平移转换成一个以 L 为边长的正方体空间（L 空间）中。在后续的星系筛选过程中，只有与中心星系在 L 空间上临近的星系才有可能被纳入同一个星系组。这里 L 空间内的距离不是按照三维空间距离衡量的，而是通过公式（2）[10]的 *Func* 计算得到的数值进行比较。另外，由于在坐标转换过程中进行了取整操作，因此，每个星系与 L 空间中的点可能形成多对一的映射，也即不同的星系通过对其空间坐标进行坐标转换与平移后，都能够与 L 空间中的某一点对应，但不同的星系转换之后可能对应 L 空间中同一个点。

$$\begin{cases} x=\rho\sin\theta\cos\varphi,\ x'=x*\dfrac{L}{L'}+\dfrac{L}{2} \\ y=\rho\sin\theta\sin\varphi,\ y'=y*\dfrac{L}{L'}+\dfrac{L}{2} \\ z=\rho\cos\theta\quad\ ,\ z'=z*\dfrac{L}{L'}+\dfrac{L}{2} \end{cases} \tag{1}$$

$$Func(x'',y'',z'')=x''+L*y''+L^2*z'' \tag{2}$$

为了便于后续工作中对转换后的星系数据进行快速查找，映射到 L 空间中中同一个点的多个星系（设为集合 S）被存储到以数组方式构造的链接表（*Linklist*）中，称为索引数组[8]。S 中位于表头的星系是编号最小的星系，每个星系以星系编号作为数组下标，每个星系中存储的元素是其前一个星系的编号（也即下标），编号最小的星系在数组中存储的元素值为 0。同时，另一个数组 *Ihoc* 记录了每个星系群中编号最大的星系编号。因此，通过对 *Ihoc* 以及 *Linklist* 进行遍历，即可搜寻到所有的星系。

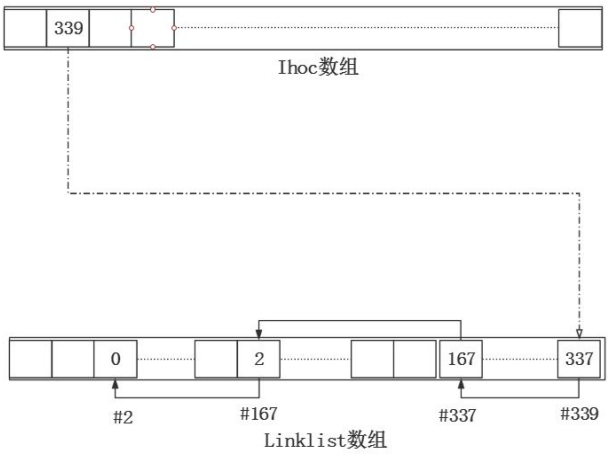

图 1 Ihoc 数组与 Linklist 数组访问模式

Fig1 Access pattern of Ihoc and Linklist

3.1.2 星系成团筛选与划分

在进行成团筛选与划分之前，每个星系先被初始化为一个独立的组（Group）。之后，对于每个组，先确定每个组的星系中心，并且以该星系为中心构造一个 L 空间中的正方体（连接体积 Linking Volume[7]），如图 2 中的大正方体。由于在 3.1.1 中，所有的星系都已经通过坐标转换和平移映射到了 L 空间中，因此，可以通过 *Ihoc* 和 *Linklist* 数组中记录的星系编号，访问存储星系属性的主数组，从而得到该连接体积中每个整数点对应的所有星系属性，最后根据星系的属性判断该星系是否可以划入当前的组当中。图 2 中的小正方体即 L 空间的整数点，其内部的点代表与该点对应的星系。判断星系是否属于当前星系组的标准包含空间距离和红移距离，当且仅当星系满足这两个标准时才会被纳入当前星系组。在筛选划分完毕之后，程序将更新当前星系组信息，然后开始筛选下一个星系组，直至所有的组都被遍历。在程序中，一共遍历了 2 遍所有的星系组。

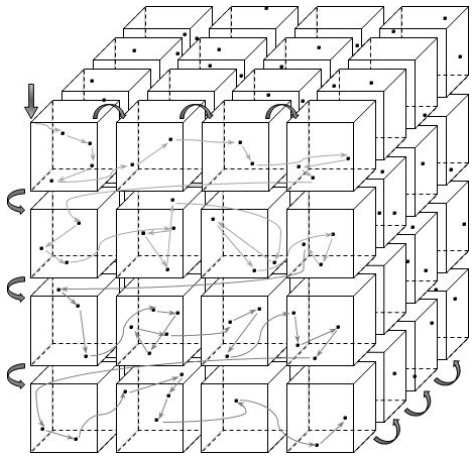

图 2 遍历筛选连接体积中星系示意图

Fig2 Traversing and filtering galaxies in Linking Volume

3.2 HGGF 性能瓶颈分析

通过使用 Intel® VTune™ Amplifier XE[9]性能分析工具对该应用进行热点分析，我们得到了如下结果：

表 1 HGGF 程序性能分析

Table1 Profiling of HGGF Application

	运行时间（s）
Search 函数	3336.231
程序整体	3960.613

从表 1 中我们可以看出耗时最长的函数是 *search*，对应星系成团筛选划分工作。因此我们将结合这部分程序的特性和 MIC 卡架构特点对 HGGF 算法进行优化。

由 3.1 节算法描述可知，星系成团筛选划分的过程中，需要根据 *Ihoc* 和 *Linklist* 数组对星系进行索引，再根据星系编号去访问主数组。由于 *Linklist* 将星系编号作为下标进行索引，而位于同一个链表中的不同星系间编号差别较大（星系个数达 639359，因此星系编号为 1~639359），而每次访存时载入的数据块大小仅为为一个缓存块，因此访问索引数组和主数组数据的不规则性以及较高频率的访存行为导致程序性能受限。另一方面，在筛选星系的过程中，L 空间中的星系数量巨大，因此遍历中心星系连接体积中的星系这一循环迭代也十分耗时，亟需并行化。

4 优化方法

4.1 线程级并行（TLP）优化

为了将星系成团筛选时遍历中心星系连接体积内星系的这一过程并行化，我们在 *search* 函数内部开启了 OpenMP 并行区域。每个 OpenMP 线程负责部分连接体积点上星系的筛选过程，如图 3 所示。

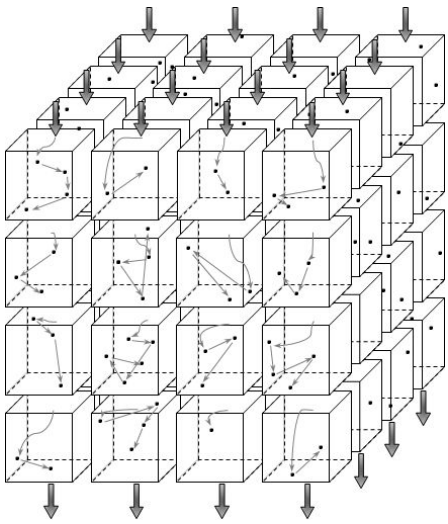

图 3 并行筛选连接体积中星系示意图

Fig3 Filtering galaxies in parallel in Linking Volume

由于函数内部需要对满足筛选条件的星系个数进行计数，因此累加计数这一过程需要保证线程之间访问的互斥性。为了保证程序的正确性，我们最初使用了 *critical* 子语对这一代码段进行保护，使程序能够利用 OpenMP 进行并行化，同时保证程序结果的正确性，得到了 OpenMP 初步并行优化结果。

由于 MIC 卡提供了大量的硬件线程，因此 *Critical* 互斥区域的存在导致各个线程之间存在一段串行区域，严重阻碍和降低了各个线程执行的效率，因此我们将为各个线程分配了内部计数器，各个线程的计数器存储在数组当中。通过这一优化，线程间的互斥区域即可消除。

4.2 数据级并行（DLP）优化

4.2.1 SIMD 优化

在使用 OpenMP 对应用做了线程并行优化的基础上，我们进一步使用 SIMD 指令作数据级并行。

在 HGGF 应用的核心部分需要通过计算两个星系之间的距离参数来筛选星系群组成员，这其中包括红移距离、三个维度上空间距离的计算。但是在计算部分中，计算量不大，并且在计算红移距离和空间距离后，分别都需要进行条件判断，确定后续的程序分支，从而简单的向量化不适用于 HGGF 应用。散布在计算中的分支判断给向量化带来了困难。对此，我们通过分支延迟，把第二次计算的主要部分提到第一次分支判断前，与第一次计算进行向量化并行，如图 4 所示。采用这种手段使计算部分与条件判断部分分离，易于对计算部分做数据并行优化。

我们通过在程序中使用由 MIC 卡的 KnightsCorner 架构所支持的 intrinsic 函数手动向量化。但是由于程序的计算量很小，在一次向量化计算的 16 个单浮点数中只有 4 个被实际使用，利用率只有理论性能的 1/4。这是由于 HGGF 算法逻辑限制导致在程序中无法充分利用向量化。

Before Branch Delay:

dist1 ← RedshiftDistance
Condition(*dist1*)
dist2 ← SpaceDistance
dist2 ← *dist2* - RedshiftDistance
Condition(*dist2*)

After Branch Delay:

[*dist1, dist2*]← [RedshiftDist, SpaceDist]
Condition(*dist1*)
Condition(*dist2* - *dist1*)

图 4 分支延迟示例

Fig4 Case of branch delay

4.2.2 数据地址对齐

在星系筛选过程中，所使用的星系属性仅包含主数组的前四列元素（都是单精度浮点型）[10]，用于计算红移距离和空间距离。这四个元素要载入同一个向量寄存器中。载入时，MIC 卡的向量处理单元将从缓存总获取连续多个单精度浮点数。如果这些数据没有被对齐，那么需要额外的指令来进行多次载入。鉴于 MIC 上的向量寄存器宽度为 512 位（64 字节），向量处理单元将从缓存中获取连续 16 个单精度浮点数。同时，缓存行大小也是 64B。为了简化 SIMD 的实现，我们将主数组的前四列元素单独取出构成一个数组，并将数据地址按照 64B 对齐，使每次从内存中载入数据时能够刚好一次性载入 4 组数据（一共 16 个数据，缓存行大小为 64B）。这样的处理能保证 SIMD 向量化的正确性与效率，同时能够对程序访存起到优化。

4.3 访存优化

4.3.1 数据预排序

由于在根据 *Ihoc* 和 *Linklist* 数组进行索引时，我们得到的一系列对应 L 空间中同一点的星系间编号相差较大，导致对主数组的访问非常不规则，由此产生的访存延迟严重影响了程序的性能。因此，为了提高主数组的访存效率，我们对主数组中的星系数据进行预排序，使主数组中星系属性的存储顺序更加贴近于程序访问的模式和习惯[10]。

首先，我们先将程序运行时各个星系对应的转换坐标进行记录，然后将转换坐标作为星系新的属性存储到主数组当中。在此基础上，我们根据星系的转换坐标大小，将主数组中的星系进行重排。经过预排序之后，每当星系成团筛选过程中需要根据索引数组访问主数组的星系属性时，由于映射到 L 空间同一点的星系的转换坐标相同，因此它们在主数组中的存储位置也是连续的，这就使得主数组与优化前相比，访存效率更高，缓存的利用率也更高。

4.3.2 减少内存数据复制

在 4.1 节的优化中，为了去除互斥区域带来的额外开销，我们将各个线程搜索到的星系个数记录到私有的计数器中，最后再进行累加。然而，为了将搜索到的星系属性信息存储下来，每个线程也必须额外使用一段数组将星系信息进行存储，最后再将这一系列数组拼凑成一维目标数组，用于后续计算。

考虑到星系的属性信息量巨大，因此这一过程引入了大量的数据复制：线程内部将星系属性信息从主数组复制到专门分配给线程的数组；再将星系属性信息复制至目标数组。为了减少这一内存复制开销，我们在线程内部仅将星系编号记录到数组中，最后根据编号将星系属性信息从主数组中复制至目标数组，并且对后一步骤进行并行化处理。经过优化之后，这一过程的内存复制量减少了约 40%。

4.3.3 数据预取

考虑到星系筛选过程中，每次对 L 空间中不同的点对应的星系进行筛选时，需要访问主数组的不同位置，因此数据预取是提高访存效率可以尝试的途径。星系筛选的第二层迭代中，我们将下一个 L 空间点所需的主数组数据从内存中提前载入缓存，与此同时当前 L 空间点的筛选工作同时进行。因此内存访问的延迟通过这一技术得到一定程度上的隐藏。

4.3.4 数据在 L2 伪共享缓存上的布局优化

为了存储每个线程的星系成团筛选划分结果，程序中引入了一个二维数组 *Ig*，以线程号作为下标，记录每个线程的搜索结果，在每个线程搜索完毕之后再对 Ig 中的数据进行拷贝、整合。但是这一数组的引入带来了类似于 NUMA 内存架构中的 First Touch[14] 的问题。

由于 MIC 卡上 L2 缓存是物理独立、逻辑共享的，当 *Ig* 数组被主线程初始化后，数组元素被载入了主线程所在处理器连接的 L2 物理缓存中，导致其余线程在访问 *Ig* 数组中各自所需的数据时，需要先将主线程的处理器所连接的 L2 缓存上的缓存行复制到自己的 L2 缓存中[12]，然而远程 L2 缓存访问的时间将明显长于访问本地缓存。实际上，远程 L2 缓存访问仅比内存访问快 17%，并且与双向环上两核间物理距离无关[12]。为了消除这一额外的访存开销，我们将 *Ig* 二维数组每行的分量设为每个线程内部的私有数据，并且在并行区域内部进行初始化。在记录完搜索结果后，同样进行数据拷贝和整合。与此同时，这一优化也将 *Ig* 数组中上万个元素的初始化进行了并行化，提高了程序的并行度。

5 优化结果与分析

我们的测试平台为搭载两块 Intel® Xeon® CPU 和两块 Intel® Xeon Phi™ 协处理器的单节点，具体参数如下：

表 2 测试平台硬件参数

Table2 Hardware configurations

	CPU	MIC
型号	E5-2683v3	Xeon Phi5110p
核数	14	60
一级缓存	448 KB (instruction) / 448 KB (data	32KB(instruction)/ 32KB(data)
二级缓存	3584KB	512KB*60
三级缓存	35840KB	无

我们使用的 profiling 程序性能测试工具为 Intel® VTune™ Amplifier XE 2016。

5.1 线程级并行（TLP）优化

我们测试了 HGGF 应用的两个 OpenMP 线程并行版本在 MIC 卡的 60 个核心上（不使用超线程）的强扩展性。第一个版本使用了 critical 导语为互斥变量建立临界区，另一个则通过互斥变量私有化使线程相互独立。我们把它们与原始的串行版本进行了比较。

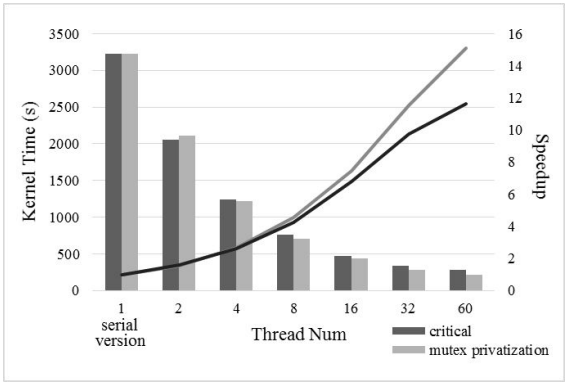

图 5 TLP 优化后 kernel 部分耗时与加速比

Fig5 Run time and speed-up of kernel with TLP optimization

通过程序热点用时随线程数的变化曲线（图 5），我们可以看到对于热点部分，随着运行时线程数的增加而成比例地减小。相较于建立临界区的版本，变量私有化版本的热点耗时更少，多线程的强扩展性更好。具体加速比如表 3 所示。

表 3 去除 critical 子语前后 kernel 加速比对比

Table3 Comparison of kernle speed-up with and w/out critical clause

Threads Num	critical region (speedup)	mutex privatization (speedup)
32	9.73	11.54
60	11.66	15.09

在线程数达到 30 时，热点部分加速约 10 倍，线程达 60 时，热点部分并没有再提升一倍，使用私有化的版本也只有约 15 倍的加速。

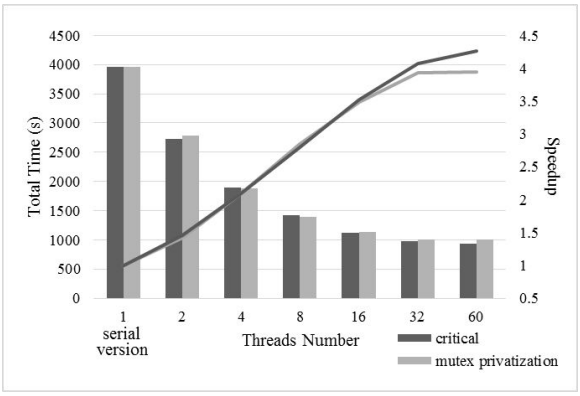

图 6 TLP 优化后总耗时与总加速比

Fig6 Total run time and total speed-up with TLP optimization

从应用整体用时看，使用临界区的版本耗时更少。导致该结果的主要原因在于：互斥变量在并行区中分解成私有变量，在并行区结束后要将它们汇总起来，这需要额外的开销。这一情况在并行线程数增加时更加显著。

根据 amdahl 定律，在 HGGF 中热点部分的串行耗时占总时长的 84.23%，所以在使用 60 个线程时的理论加速比为 5.82。实际上，临界区版本的加速比仅有 4.25，而私有化版本为 3.95。可见据理想加速比仍有差距。

5.2 数据级并行（DLP）优化

TLP 优化结果表明，有其他因素制约 HGGF 应用的多线程的扩展。根据之前的分析，最主要的是向量化程度低，和内存访问延时较长。这里将分析应用第 4 节中 DLP 优化方法的结果。

我们在互斥变量私有化的并行优化基础上，对热点部分的计算作向量化。这是因为使用私有化之后的程序更易于向量化。向量化的主要方法是用 Knights Corner 架构所支持的 intrinsic 函数中 SIMD 指令改写热点中的计算部分。为了保证 SIMD 指令运行的正确性和效率，我们还对所涉及的运算数做了数据地址按 64 字节对齐。

我们把使用和未使用 SIMD 指令的程序性能做比较。两者都使用 60 个线程运行。未使用 SIMD 指令时热点耗时 214(s)，使用后仅耗时 179(s)，加速比为 1.2。

通过 Intel® VTune™ Amplifier XE 可以得到程序运行时的向量化程度。从表 4 中可以看出使用 SIMD 指令极大地提升了向量处理单元的利用率。此外，通过结合使用数据地址对齐，L1 cache 的命中率也提高了，降低了程序受访存延时的影响。

在经过 TLP、DLP 两步优化之后，热点部分的总加速比为 18.64，总耗时加速比为 4.44。

表 4 SIMD 优化前后热点部分向量化程度与缓存命中率对比

Table 4 Comparison of vectorization intensity and cache hit ratio with and w/out SIMD in kernel part

	TLP	TLP+DLP
vectorization intensity (kernel)	1.000	15.897
L1 cache hit ratio (kernel)	76.5%	81.3%

5.3 访存优化

在结合了线程级和数据级并行优化后，热点部分的总加速比不到 20。这大约是所使用线程数的 1/3。之前已经指出，这是 HGGF 应用访存上的不规则行为所导致的。我们通过逐步使用数据预排序、减少内存复制、数据预取和优化 L2 Cache 数据布局来优化程序的访存性能。

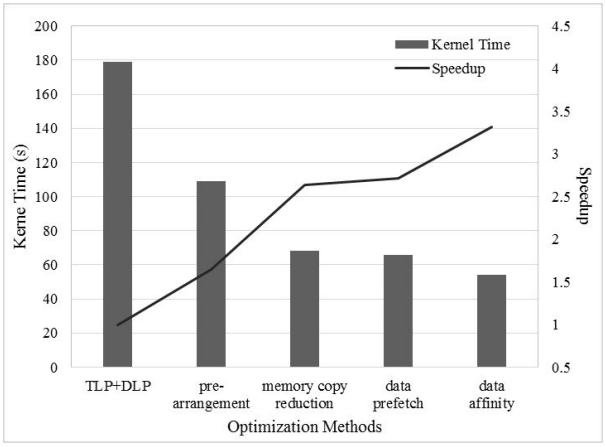

图 7 访存优化后 kernel 耗时与加速比

Fig7 Run time and speed-up of kernel with various memory access optimization methods

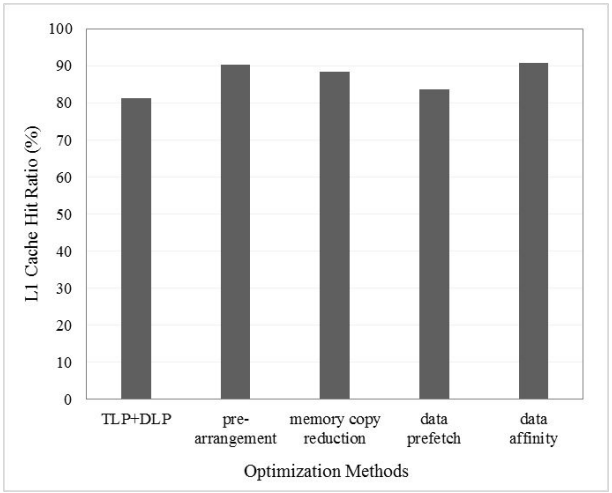

图 8 访存优化后 kernel 缓存命中率

Fig8 Cache hit ratio of kernel with various memory access optimization methods

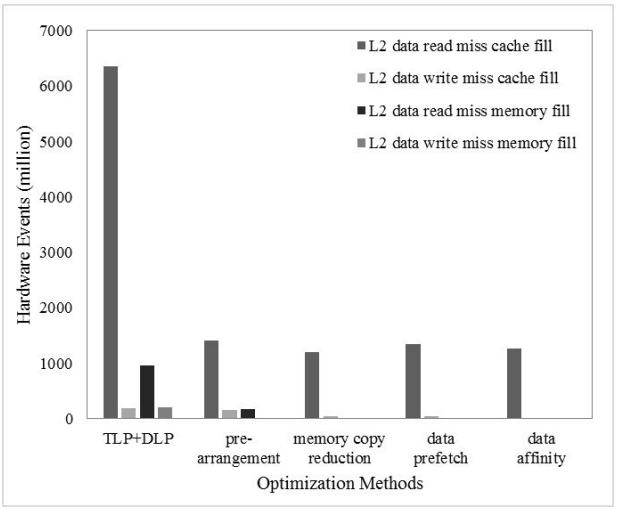

图 9 访存优化后 L2 缓存/内存硬件事件次数

Fig9 L2 Cache/Memory hardware events with various memory access optimization methods

以TLP与DLP优化为基准，我们可以看到几个访存优化方法都有对程序性能的提升。其中数据预排序、减少内存复制和优化L2 缓存上的数据布局都带来了一定的性能提升。通过Intel® VTune™ Amplifier XE统计热点部分L1 cache hit ratio的数值，可见数据预排序明显地增加了L1 cache的命中率（如图8所示），使数据访问局部性得到提升；在减少内存复制后，L2_DATA_READ/WRITE_MISS_MEM_FILL事件数量减少（如图9所示），程序的性能得到了进一步提升；在优化数据在L2缓存上的布局后，L2_DATA_READ/WRITE_MISS_CACHE_FILL事件——数据读写L2不命中而向邻核L2缓存读取数据的次数有所下降，从而降低了程序在I/O上等待的时间，提高了性能。

另一方面，软件数据预取却没有性能提升，甚至略微下降。经分析，可能由以下原因导致：（1）经过数据预排序的优化，数据局部性已经明显增强，由数组模拟的链表元素基本按序排列于数组中，此时额外的软件预取作用不大；（2）根据 L1 cache hit ratio 的下降，可能是软件预取和 MIC 卡上的基于 TLB 硬件预取发生冲突，产生 cache 的抖动；（3）经过数据级并行后，热点的计算耗时减少，无法与数据预取时间充分重叠。这些原因导致软件数据预取并未起到效果。

5.4 整体分析

经过线程级并行、数据级并行和内存访问优化，程序的热点部分与整体的性能变化如下：

表 5 各优化版本耗时与加速比总结

Table5 Summary of run time and speed-up of various optimization methods

版本	热点用时	热点加速	整体用时	整体加速
串行版本	3336.231	1	3960.613	1
线程级并行	213.998	15.59	1003.132	3.95
数据级并行	178.981	18.64	891.904	4.44
访存优化	54.002	61.78	743.421	5.33

最终，程序的热点耗时为串行版本的 1/62，这个结果好于线程级并行优化的理论加速比，说明数据级并行优化与访存优化的确起到了很好的效果。同时，访存优化也被应用于程序热点前后的串行部分，产生了额外的性能提升。应用整体加速比为 5.3，较为接近 amdahl 定律的理论加速比 5.8。

6 总结与下一步工作

经过对 HGGF 应用算法与访存行为的分析，以及结合 MIC 卡的架构特点，我们将 HGGF 程序在 MIC 卡上进行了线程级、数据级以及访存优化。与单线程性能基线相比，在仅使用 OpenMP 优化的情况下，MIC 卡的性能并未被充分挖掘，kernel 部分在开启 60 个线程情况下仅得到 15 倍左右的加速；通过改变应用部分代码结构使其适应 MIC 提供的超宽向量寄存器之后，SIMD 优化进一步提高了加速比。在此基础上，为了使访存行为极不规则的 HGGF 应用能够在 MIC 卡提供有限且较低的缓存的条件下得到性能的进一步提升，我们进行了数据预排序、减少内存复制、预取、改善数据在 L2 缓存上的布局等一系列优化措施，最终使 HGGF 应用的 kernel 部分在单 MIC 卡上获得了 62 倍左右的性能提升。

在下一步的工作中，我们将结合 MPI，使用多块 MIC 卡，对 HGGF 程序进行多节点扩展。然而由于 HGGF 并不是易并行(Embarassingly Parallel)的，其内在的数据紧耦合特点，以及部分星系组搜寻之间潜在的数据依赖关系，导致 HGGF 应用的问题分解算法将是一个复杂、困难、具有挑战性的问题。为此，我们将对算法进行改进，对数据结构进行重新设计，并且借鉴提交与回滚机制，依靠管理进程的仲裁，最大程度解除工作进程负责的不同星系组之间的依赖关系，以提高节点间并行度，从而将该问题扩展至多个计算节点，以充分利用异构计算集群的强大的计算资源，分析处理天文观测产生的大数据。

致谢 感谢上海交通大学高性能计算中心提供的试验环境，感谢上海交通大学物理与天文系杨小虎教授对程序算法的介绍与讲解，感谢上海交通大学高性能计算中心司雨濛同学在优化过程中提供的帮助与支持。

参 考 文 献

[1] Xiaohu Yang, H. J. Mo, Frank C. van den Bosch, Y. P. Jing. A halo-based Galaxy Group Finder. 2005.

[2] X. Liu , M. Smelyanskiy, E. Chow , et al. Efficient sparse matrix-vector multiplication on x86-based many-core processors[C]//Proceedings of the 27th international ACM conference on International conference on supercomputing. ACM, 2013: 273-282.

[3] Q. Wu, C. Yang, T. Tang, et al. MIC acceleration of short-range molecular dynamics simulations[C]//Proceedings of the First International Workshop on Code OptimiSation for MultI and many Cores. ACM, 2013: 2.

[4] S. Potluri, A. Venkatesh, D. Bureddy, et al. Efficient Intra-node Communication on Intel-MIC Clusters[C]//Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, 2013: 128-135.

[5] S. Saini, H Jin, D Jespersen , et al. An early performance evaluation of many integrated core architecture based SGI rackable computing system[C]//Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2013: 94.

[6] Xiaohu Yang, H. J. Mo, Frank C. van den Bosch, Anna Pasquali, Cheng Li,Marco Barden, 2007, ApJ, 671, 153.

[7] V.R. Eke et al. Galaxy groups in the 2dFGRS: the group-finding algorithm and the 2PIGG catalogue. 2004.

[8] Yuan Lin, David Padua. Analysis of Irregular Single-indexed Arrays and its Applications in Compiler Optimizations. 2001.

[9] VTune: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[10] He Hao, Yumeng Si, Optimizing Irregular Memory Access in Astrophysical Clustering Studies. 2015

[11] Jim Jeffers, James Reinders, Intel Xeon Phi High-Performance Programming. 2014

[12] James Lin, Shuo Li, Jiaming Zhao, Satoshi Matsuoka. Node-level Memory Access Optimization on Intel Knights Corner. 2014.

[13] Y.P. Jing and Yasushi Suto. Triaxial Modeling of Halo Density Profiles with High-Resolution N-Bbody Simulations. 2002.

[14] Lameter, Christoph. "Numa (non-uniform memory access): An overview." Queue 11.7 (2013): 40.