Intel Knights Corner 的结点级内存访问优化*

林新华 [1,2,+]，**李硕** [3]，**赵嘉明** [1]，**松岗聪** [2]

[1]（上海交通大学 高性能计算中心，上海 200240）

[2]（东京工业大学 学术国际情报中心，日本）

[3]（Intel 公司，软件与服务部门，美国）

Node-level Memory Access Optimization on Intel Knights Corner*

James LIN[1,2,+], Shuo LI[3], Jiaming ZHAO[1], Satoshi MATSUOKA[2]

[1](Center for High Performance Computing, Shanghai Jiao Tong University, Shanghai 200240, China)

[2](Global Scientific Information and Computing Center, Tokyo Institute of Technology, Japan)

[3](Intel Corporation, Software and Services Group, USA)

[+] Corresponding author: Tel: +86-21-34206060, E-mail: james@sjtu.edu.cn, http://hpc.sjtu.edu.cn

Abstract: Traditional Programming Optimization (TPO) has limited effects on Intel Knights Corner (KNC). Therefore, we propose Memory Access Optimization (MAO) for KNC. We applied MAO to TPO version of Diffusion 3D, and improve the performance 39.1%. We made two contributions in this paper: we believe 1) MAO is indispensable to KNC and TPO+MAO is the path to Ninja Performance, the best optimized performance; 2) Intrinsic-based MAO is more efficient to stencil code than compiler-based MAO. Our findings on MAO will inspire large-scale applications optimizations on KNC.

Key words: Traditional Programming Optimization (TPO); Intel Knights Corner (KNC); Memory Access Optimization (MAO); Ninja Performance;

摘　要：传统编程优化（Traditional Programming Optimization，简称 TPO）在 Intel Knights Corner （简称 KNC）上收效甚微。我们提出必须重视内存访问优化（Memory Access Optimization，简称 MAO）。我们将 MAO 应用到 Diffusion 3D 这个已经过 TPO 的程序上后发现性能仍然提高了 39.1%。本文主要有 2 个贡献：1) 提出 MAO，认为 TPO+MAO 有助于在 KNC 上获取最优化性能； 2)发现对于 stencil 代码，基于 intrinsic 的 MAO 比基于编译器的 MAO 要高效。这些发现对于在 KNC 上优化大规模应用有启发意义。

关键词：传统编程优化 Intel Knights Corner 内存访问优化 最优化性能

1　简介（Introduction）

传统编程优化（Traditional Programming Optimization，简称 TPO）[1]可分为 2 个层面：算法优化和最新编译器技术的应用。我们提出必须重视内存访问优化（Memory Access Optimization，简称 MAO）。我们将 MAO 应用到 Diffusion 3D 这个已经过 TPO 的程序上后发现性能仍然分别提高了 8.1%和 39.1%。本文主要有 2 个贡献：

1)　提出 MAO，认为 TPO+MAO 有助于在 KNC 上获取最优化性能；
2)　发现对于 stencil 代码，基于 intrinsic 的 MAO 比基于编译器的 MAO 要高效。

---

* 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” 及日本学术振兴会 RONPAKU Fellowship 资助。

作者简介：**林新华**，男，讲师，上海交通大学高性能计算中心副主任、东京工业大学学术国际情报中心客座准研究员，主要研究方向为性能建模与优化；李硕，男，Intel 公司资深工程师，主要研究方向为高性能计算在量化金融中的应用；赵嘉明，男，上海交通大学计算机本科生；松岗聪，男，东京工业大学教授、ACM Fellow，主要研究方向为高性能计算。

2 相关工作（Related Work）

Intel 公司 Pakesh Krishnaiyer 等人在 IPDPSW2013 上发表的论文[2]讨论了基于编译器的 Data Prefetching（数据预取）和 Streaming Non-temporal Store （简称 NT-store）。此文对于 KNC 体系架构的特点有比较深入的分析，并对数据预取和 NT-Store 的原理进行了详细介绍。但对于 MAO 仍局限于编译器技术的利用，而且没有指出有些测试用例对于基于编译器的优化不敏感是否与数据访问模式过于复杂有关。

3 内存访问优化（MAO）

3.1 数据对齐（Data Alignment）

- **前端对齐确保数组的起始地址对齐**

共有 3 种方式。第 1 种方法最基本，可以将普通的内存分配和释放改写为对齐的 intrinsic 原语，即将 malloc() 改为_mm_malloc(), free()改为_mm_free()。不过由于该方法采用了进程内单一全局锁，所以扩展性不佳。因此建议采用第 2 种方法，Intel Threading Building Blocks（简称 TTB）提供的函数保证了线程间相互独立，因此更适合 KNC。方法是将 malloc() 改为 scalable_aligned_malloc(), free()改为 scalable_aligned_free()。通常性能都会比第 1 种 intrinsic 原语要好，推荐使用。第 3 种方法是直接使用 SIMD 的 intrinsic 原语，例如使用_mm512_loadunpacklo_pd()和_mm512_loadunpackhi_pd()读入数值，并用_mm512_fmadd_pd()进行乘加融合操作（FMA）。这种方法性能最好，但编程相对困难。

- **后端对齐确保 2 维数组的每行第一个元素的地址对齐**

在每行末尾分配一些多余的空间，以此保证 2 维数据的左端对齐，这称为 padding。

3.2 Cache 预取（Cache Prefetching）

硬件预取只支持 L2 cache 的 stream 模式，其余的就需要利用编译器或是手动预取。

- 编译器指导语句

在迭代前加入#pragma prefetch var:hint:distance，其中 var 是要预取的变量，hint 为 0 表示预取到 L1 cache，为 1 则表示预取到 L2 cache。

- 使用 Intrinsic 手动预取

使用 void _mm_prefetch(char const*address, int hint)函数预取 cache，其中 hint 为 0 表示预取到 L1 cache，为 1 则表示预取到 L2 cache。在手动预取时，需要设置-opt-prefetch=0 或#pragmas noprefetch 以关闭编译器预取，避免发生不可知的冲突。

3.3 Streaming Non-temporal Store（简称 NT-Store）

通常更新某个数据时，如果该数据不在 cache 中，需要先把它（预）读取到 cache 中，在 cache 中更新后，然后再写回内存中，但这样既浪费内存带宽又浪费 cache 空间。更高效的做法是直接写到内存中，这称为 NT-Store。可以通过编译器选项-opt-streaming-stores 来设置，有 3 个可选值 auto、always 和 never。

4 Stencil 访问：Diffusion 3D

Diffusion 3D 用来模拟 3D 容器中溶质在溶液中的扩散过程，可以用公式 1 中的微分（ODE）方程来表示：

$$\frac{\partial f}{\partial t} = \kappa \nabla^2 f = \kappa(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2})$$

（公式 1）

离散化之后就得到 1 个 3D 7 点 Stencil 代码，如图 1 所示。

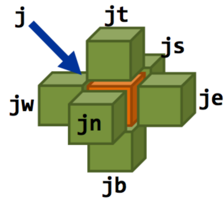

图 1. Stencil 示意图

```
fn[j] = cc*f[j]
        + ce*f[je] + cw*f[jw]
        + cn*f[jn] + cs*f[js]
        + ct*f[jt] + cb*f[jb];
```

表 1. Stencil 核心算法伪代码

表 1 的伪代码说明 Diffusion 3D 的核心算法包含了 7 个 stencil 读和 1 个 stream 写。MAO 的具体过程如表 2 所示。

优化版本	优化方法
A:TPO 基准	使用 KNC Native 模式，已经应用了 AOS 到 SOA 转换，cache blocking，并行化（使用 OpenMP）和向量化（使用#pragma simd）
B: A+TBB 数据对齐	设置对齐长度为 64，使用 scalable_aligned_malloc()和 scalable_aligned_free()函数，分配和释放 f 和 fn 这个 2 数组，并在核心循环前添加#pragma vector aligned
C: B+ cache 预取	针对 f[j]和 f[jb]这 7 个值，使用编译器指导语句#pragma prefetch 进行预取。L2 cache 的 distance 为 8，L1 cache 的 distance 为 1，同时添加编译器选项 -opt-prefetch-distance=8,1
D: C+ NT-Store	由于数组 fn 没有读入就直接写回，因此在核心循环前添加#pragma vector nontemporal (fn[j])，并添加编译器选项-opt-streaming-cache-evict=0
E: A+ intrinsic 数据对齐	使用 SIMD 的 intrinsic 原语_mm512_loadunpacklo_pd()和_mm512_loadunpackhi_pd()读入 f[j]和 f[jb]等，并使用_mm512_fmadd_pd()对它们进行 FMA 操作
F: E+ intrinsic cache 预取	针对 f[j]到 f[jb]这 7 个值，使用_mm_prefetch()原语进行预取。L2 cache 的 distance 仍为 8，L1 cache 的 distance 仍为 1
G: F+ intrinsic NT-Store	使用_mm512_storenrngo_pd()原语存储 fn 数组

表 2 Diffusion 3D 的 MAO 过程

5 实验环境（Experimental Setup）

我们的实验在上海交通大学超级计算机 π 上的 1 个 KNC 结点（配置见表 3）上进行。π 是目前国内高校排名第 1 的超级算机，使用 CPU+GPU+MIC 的异构架构，峰值计算能力达到 263TFlops，2014 年 6 月的 TOP500 排名为 251。

软硬件	配置
CPU	Intel SNB E5-2670 @2.6GHz
KNC	Intel KNC 5110p @1.05GHZ
MPSS	MPSS version 3.2.3
编译器	Intel Composers XE 2013.3.163

表 3. KNC 结点的硬件和系统软件配置

6 结果分析（Results Analysis）

运行时的环境参数为 OMP_NUM_THREADS=180 和 KMP_AFINNITY="scatter"。之所以使用 180 线

程而不是 240 线程是因为 Diffusion 3D 属于内存带宽受限（Memory Bandwidth Bound）的应用，单核上跑满 4 个线程反而会导致性能下降。我们使用了程序内置的计数器测量 wall clock，使用 Likwid 测量 VPU usage 和 L1 cache miss rate，使用程序内置的计数器测量 Memory Bandwidth。结果如表 4 中所示。

优化版本	Wall Clock(s) 越小约好	VPU usage 越大越好	L1 cache miss rate % 越小约好	Memory Bandwidth (GB/s) 越大越好
A:TPO 基准	3.91	5.8	5.7	67.7
B: A+TBB 数据对齐	3.65	6.1	6.3	71.4
C: B+ cache 预取	3.59	6.1	11.4	74.1
D: C+ NT-Store	3.02	6.1	4.3	89.2
E: A+ intrinsic 数据对齐	3.47	5.7	6.2	77.3
F: E+ intrinsic cache 预取	3.47	5.7	11.1	74.7
G: F+ intrinsic NT-Store	2.81	5.2	6.8	94.6

表 4. Diffusion 3D 的 MAO 性能结果

总体来说，进行 MAO 之后，虽然 VPU usage 和 L1 cache miss rate 变得更糟，但由于 Memory Bandwidth 利用率提高了 39.7%，所以版本 G 的性能与版本 A 相比还是提高了 39.1%。这说明 Memory Bandwidth 在 Diffusion 3D 的性能中起了决定性作用。Cache 预取的优化并不成功。基于编译器的版本 B 虽然性能有所提升，但极大增加了 L1 cache miss rate，而基于 SIMD 的版本 F 从 4 个指标来看几乎一无是处，因此后续还需要对 cache 预取的优化进行更进一步的研究。NT-Store 因为极大增加了 Memory Bandwidth，与 BSF 中的 MAO 一样，对性能提升的贡献比其他 2 种方法都要大。基于 SIMD intrinsic 的版本比对应基于编译器版本的性能要高出约 5%。综上所述，基于编译器的 MAO 对于单元跨步访问的应用有一定效果，为了追求最优化性能，可以使用基于 intrinsic 的 MAO，但编程会相对困难些。

7 总结与下一步工作（Conclusion and Future Work）

MAO 主要包括数据对齐、cache 预取和 NT-Store 这 3 种优化方法。我们将 MAO 应用到 Diffusion 3D 这个已经过 TPO 的程序上后发现性能仍然 39.1%。基于以上的工作，我们对 KNC 上的 MAO 有如下建议：

1） MAO 与 TPO 的优化效果可以叠加，因此 TPO+MAO 有助于在 KNC 上获取 Ninja Performance；
2） 对于 stencil 代码，为了追求最优化性能，可以使用 intrinsic，但要考虑所收获的性能提升与所付出的编程努力之间的平衡；
3） 尽量使用 TBB 的数据对齐；
4） 如有可能，一定要使用 NT-Store 优化；
5） cache 预取的编程比其他 2 种方法要相对难些，不过在数据连续访问的应用中通常能获得比较理想的效果。

References:

[1] N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey, “Can traditional programming bridge the Ninja performance gap for parallel computing applications?,” Computer Architecture (ISCA), 2012 39th Annual International Symposium on, pp. 440–451, 2012.

[2] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, and H. Saito, “Compiler-Based Data Prefetching and Streaming Non-temporal Store Generation for the Intel(R) Xeon Phi(TM) Coprocessor,” presented at the IPDPSW '13: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, 2013, pp. 1575–1586.