使用 Stencil 评估 Intel AVX2 vgather 指令*

林新华[1,2,4+]，秦强[1]，李硕[3]，文敏华[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）

[4]（NVIDIA Technology Center，新加坡）

Evaluating Intel AVX2 Vgather Instructions with Stencils*

James LIN[1,2,4+], Qiang QIN[1], Shuo LI[3], Minhua WEN[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)

[4](NVIDIA Technology Center, Singapore)

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

Abstract: Intel provided AVX2 vgather instruction on Haswell CPU to better support reading discontinued data in vectorization. We find the compiler generates vgather instructions, which slow down the performance of Stencil on Haswell, because the branches exist in defining boundary condition of Stencils. We propose to utilize peel optimization or intrinsic load to avoid these vgather instructions. We have applied these optimization to three Stencil benchmarks, a long-range Stencil 3DFD, and a hybrid Stencil application, and archived the speedup from 1.22X to 3.88X on Haswell. By analyzing the implementation of the instruction, we find the vgather instructions are decoded into multiple micro-operations (μops), and the instructions generate one μops for each element to be gathered. Due to the high overhead of decoder, the vgather instructions become the performance bottleneck of Stencils on Haswell. We believe the understanding of the implementation of AVX2 vgather instructions and adopting the optimizations to avoid the vgather instructions are quite helpful for performance tuning the applications with good spatial locality on Haswell.

Key words: AVX2 vgather; Stencil; performance evaluation;

摘　要：为了更好地在向量化时读取离散的数据，Intel 在 Haswell CPU 提供了 AVX2 vgather 指令。我们发现由于 Stencil 在设置边界条件时使用了条件判断，因此编译器生成了 vgather 指令，并降低了 Stencil 在 Haswell 上的性能。我们提出使用 peel 优化或 intrinsic load 的方法来避免 vgather 指令的生成，并把该方法应用到 3 个 Stencil 基准算例、长程 Stencil 程序 3DFD、以及混合 Stencil 应用 3DEW 上。发现这些 Stencil 在 Haswell 上性能都获得了 1.22X 至 3.88X 不等的提升。通过研究指令的实现，我们发现 vgather 指令会被解码成多个微操作（μops），并为每个要读入的元素生成一个 μops。由于 vgather 指令解码时会产生较高的开销，导致 vgather 指令成为 Stencil 在 Haswell 上的性能瓶颈。我们认为了解 AVX2 vgather 指令的实现以及掌握避免生成 vgather 指令的优化方法，对在 Haswell 上调优具有良好空间局部性应用的性能有一定的参

* 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：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，主要研究方向为体系结构与代码优化；秦强，男，上海交通大学计算机硕士研究生；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；文敏华，男，上海交通大学计算机高性能计算中心工程计算专员；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。

考价值。

关键词：AVX2 vgather 指令 Stencil 性能评估

1 简介

为了更好地在向量化时读取离散的数据，Intel 陆续在不同平台上提供了硬件支持的 vgather 指令：2013 年上半年发布的 Knight Corner（缩写为 KNC）上的 IMCI（Initial Many Core Instructions）vgather 指令。2013 年 6 月发布的 Haswell（缩写为 HSW）CPU 上的 AVX （Advanced Vector Extension）2 vgather 指令，而且 2014 年发布的 Broadwell CPU 改进了该指令的实现。2016 年将要发布的 Knight Landing（KNL）以及 Skylake Xeon CPU 都会支持 AVX-512 vgather 指令。

有一些文献表明 KNC 上的 IMCI vgather 指令会成为数据离散的应用[1]和 Stencil[2]的性能瓶颈。因此本文拟回答以下 2 个问题：

1) AVX2 vgather 指令是否也会降低 Stencil 在 HSW 上的性能？编译器在何种情况下会生成 vgather 指令？应该如何避免？
2) 为什么 AVX2 vgather 指令会降低 Stencil 的性能？

为了回答清楚这 2 个问题，首先，我们分析了 Stencil 的源代码，发现在 Stencil 中设置边界条件时，需要对元素地址进行条件判断。编译器因为在编译时无法确定这些元素的地址，就生成了 vgather 指令。我们提出使用 peel 优化或 intrinsic load 的方法来避免生成 vgather 指令。我们对 1D3P、2D5P、3D7P 这三个 Stencil 基准算例、长程 Stencil 程序 3DFD、以及混合 Stencil 应用 3DEW 进行了性能验证，发现消除了 vgather 指令之后，所有 Stencil 都获得了 1.22X 至 3.88X 不等的性能提升。其次，我们使用 Intel Architecture Code Analysis（IACA）分析 vgather 指令的实现，发现 vgather 指令会为每个要读取的元素生成一个 μop。据此我们初步分析了 vgather 指令对单核和多核性能的影响。

概括来说，本文有 3 个创新点：

1) 首先发现 AVX2 vgather 指令会降低 Stencil 在 HSW 上的性能。
2) 发现由于设定元素地址时进行了条件判断，编译器会生成 AVX2 vgather 指令，因此首先提出使用 peel 优化或 intrinsic load 的方法来避免生成 vgather 指令。
3) 发现 vgather 指令会被解码成多个 μops，并为每个要读入的元素生成一个 μops，并首先用 IACA 予以证明。

余下篇幅安排：第 2 节介绍相关工作；第 3 节介绍 vgather 指令；第 4 节说明实验配置；第 5 节分析了编译器生成 vgather 指令的原因，提出了相应的优化方法，并评估了不同 Stencil 优化前后的性能；第 6 节深入分析了 AVX2 vgather 指令的实现，并评估了 vgather 指令对单核和多核性能的影响；最后是总结与下一步工作展望。

2 相关工作

有一些文献讨论了 KNC 上 IMCI vgather 指令的性能瓶颈问题。George Hager 等人[1]将不规则数据访问的医学图像算法 FDK 进行性能优化，并用性能建模的方法分析了 vgather 成为 FDK 在 KNC 上性能瓶颈的原因。本文作者[2] 使用 semi-empirical 的性能建模方法，分析 vgather 指令在 KNC 上对 Stencil 性能的影响。实验结果和模型推测一致，vgather 引起的流水线停滞会占到 Stencil 全部运行时间的 20%左右。本文的研究对象与这些文献不同。本文是针对 AVX2 vgather 指令进行的分析。

3 Vgather 指令

Intel 从 AVX2 开始提供了专门的 vgather 指令。之前的 SSE 和 AVX 没有专门的 gather 指令，就需要使用标量 load 逐个读入每个元素，然后再用 shuffles (比如 pinsrd,、extractps, vinsertf128 等指令) 将这些元素插入到向量寄存器中。

- 指令格式与寻址方式

列表 1 中 vgatherdpq 的 v 表示是 AVX 操作，d 是指元素索引的大小，这里是 doubleword（4 Byte）。p 是指 packed，表示这是向量化的操作。P 出现在第 2 个位置时（即 vpgatherdp），表示操作数为整数；出现在倒数第 2 个位置时（即 vgatherdpq），表示操作数为浮点数。q 是指要元素的大小，这里是 quadword（8 Byte）。元素大小和索引大小有多种组合，对应的 vgather 指令如表 1 所示。

vgatherdpq ymm0, [esi+xmm1*4], ymm1

列表 1　vgather 的指令格式

		元素大小	
		4 Byte	8 Byte
索引大小	4 Byte	vgatherdps	vgatehrdpd
	8 Byte	vgatherqps	vgatherqpd

表　1　vgather 指令的不同形式

列表 1 中的指令使用了 Intel 汇编格式，因此 ymm0 是目的地寄存器，[esi+xmm1*4+8]是源地址，ymm1 是 mask 寄存器。ymm 是 256bit 的向量寄存器，由于 HSW 没有专门的 mask 寄存器，因此就用 ymm 寄存器来代替。但这可能会造成 mask 寄存器空间的浪费（通用向量寄存器有 256bit，而用作 mask 则最多只需要 8bit）以及通用向量寄存器的紧张。如列表 1 中 vgatherdpd 读入 4 个双精度元素，因此 mask 寄存器只需要 4bit 就可以了。ymm1 寄存器将 256bit 分成 4 个 lane，每个 lane 长度为 64bit，正好对应 1 个双精度元素。若某 lane 中的数值为 0，则对应的元素不必读入或者已经读入。反之，则读入对应的元素，并将该 lane 中的数值置 0。

- 指令操作

以列表 1 中的指令为例，vgather 读入元素分成三步。第一步是生成地址。针对每一个要读入的元素，将指令中的相对地址发送到 AGU 中，按照公式（1）生成绝对地址。如图 1 中所示，xmm1 中要读取的第 2 个元素的 index 是 3，即通用寄存器 esi 所指向的数组的第 3 个元素。第二步，检测 mask 寄存器 ymm1 中每个元素对应的数值，如果为 0，则不读入元素。反之，则进入第三步。比如由于 ymm1 中第 3 个元素对应的 mask 数值为 0，就不需要读入元素，因此 vgather 只需读入 3 个元素。注意到地址生成是在检测 mask 寄存器之前，这就意味着即使 mask 寄存器对应的数值为 0，vgather 指令还是会为那个元素生成绝对地址。第三步是读入该元素到目的地寄存器的相应位置中。注意这 3 个元素的读入是乱序的，不一定是从高位到低位或是从低位到高位。而且 Haswell 有 2 个读入端口（端口 2 和 3），因此能同时读入 2 个元素。

图　1　vgatherdpq ymm0, [esi+xmm1*4], ymm1 的操作示意图

- 指令解码

x86 指令集属于 CISC，其每条指令长度都不同。为了简化执行单元（execution unit）的设计，解码器中将 x86 指令动态翻译成类似 RISC 的指令，称为微操作（micro-operations，即 μops），但 μops 并不等价于 RISC 指令，而是更接近于解码后的 RISC 指令。x86 的解码过程需要好几个时间周期（即 cycle），因此会成为处理器 Front End 性能瓶颈的来源之一。

如图 2 所示，解码可分为 2 个阶段：预解码阶段和解码阶段。预解码阶段中，指令长度解码器（Instruction

length Decoder，ILD）将原始的二进制流划分成有效的x86指令流，然后发送给指令队列（Instructions Queue，IQ）。解码阶段则将从IQ传来的x86指令流动态翻译成对应的μops。与Nehalem和Sandy Bridge架构相同，Haswell有3个简单解码器和1个复杂解码器。简单解码器负责解码那些可以翻译成1条μops的指令，而复杂解码器则负责动态解码那些最多可以翻译为4个μops的指令。需要翻译成4个以上μops的指令，如vgather，则首先会被发送至复杂指令解码器，然后停止正常的解码流水线，发送给MSROM（micro-sequencer）单元。MSROM单元有一个sequencer回路和一个ROM数组，它会输出一个预先翻译好的μops程序。

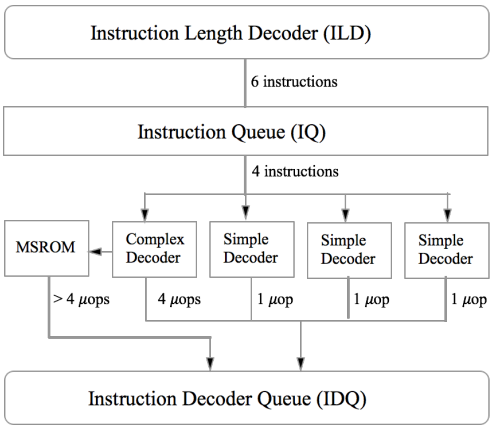

图 2 Haswell Decoder

4 实验配置

4.1 硬件

我们使用了Intel Haswell E5-2693，14核，主频为2.3GHz。L1和L2是每个核私有的，而L3是所有核共享的。L1D和L1I的大小均为32KB，8路，latency为4 cycle。L2 cache的大小为256KB，也是8路，latency为11 cycle。L3 cache的大小为35MB。内存峰值带宽为60GB/s，Stream实测带宽为53GB/s。

4.2 软件

我们主要使用了以下2个软件：Intel C/C++ Compiler 15.2.164编译器（ICC），本文中所有的测试程序都是用它编译的。Intel Architecture Code Analysis（IACA）2.2 [6]是在Intel平台上对代码进行静态分析的工具，考察其数据相关性、throughput和latency。它会提供某条指令绑定到相应执行单元端口的信息，并提供统计后的throughput和latency。我们用它来分析vgather指令在HSW上执行单元端口的绑定情况以及throughput。

4.3 Stencil

Stencil利用迭代的有限差分方法来求解偏微分方程（Partial Differential Equations，简称PDE），对于一个给定的正规网格，它通常是在时间与空间上根据周边格点数值加权求和的过程。Stencil需要读入离散的数据，但又有具有一定的空间局部性，且其性能往往受限于内存带宽。Stencil被广泛应用于计算流体力学、数值天气模拟、石油勘探等多个与国民经济息息相关的领域。

Stencil有2种不同的元素更新方法：Jacobi和Gauss-Seidel。Jacobi方法是读和写在不同的网格上，而Gauss-Seidel方法是读写在同一网格上。本文中3个Stencil基准算例、1个长程Stencil程序和1个混合Stencil应用都采用了Jacobi方法。

- Stencil基准算例

Stencil代码按维度可以分成1D3P、2D5P和3D7P这三种形式，如图3所示。1D3P更新中间的点，需要自己以及左右2个点的数据。类似的，2D5P需要自己以及上下左右4个点的数据，3D7P需要自己以及上下前后左右6个点的数据。从图3中还可以看出，1D3P在内存中具有良好的空间局部性。2D3P的X方向与1D3P相同，而Y方向，则有了NX的间距。类似的，3D7P中Z方向，则有了NX*NY的间距。

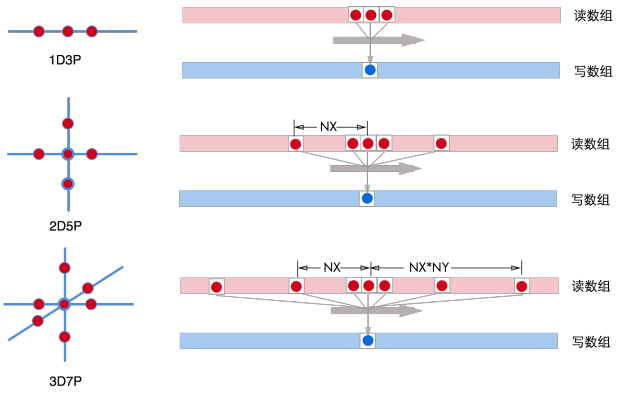

图 3 Stencil 基准算例示意图

- 长程 Stencil 程序：3DFD

3DFD[7]的全称为各向同性介质中三维有限差分程序（3-Dimensional Finite Difference Code with an Isotropic）。该程序利用有限差分方法，求解各向同性介质中噪音波动方程。更新 3DFD 的一个点，需要自己以及上下前后左右 24 个点，一共 25 个点，即 3D25P。

- 混合 Stencil 应用：3DEW

3DEW 全称为三维纵横波分离的弹性波方程模拟（3D pure P and S wave elastic wave equation modeling）。该应用由中国石油东方地球物理公司自主研发[8]，采用波场延拓技术来模拟弹性波在各向同性的弹性介质中传播。3DEW 分别模拟纵波（P 波）和横波（S 波），以便更好地研究这两种波在弹性介质中的传递，同时利用高阶有限差分方法分析弹性波的传递情况。3DEW 是一个混合 Stencil，波被分解为 u、v、w 3 个方向，空间被分解为 x、y、z 3 个方向。更新一个点，首先需要计算 1 维上波的传播，比如 u 在 x 方向上的传播，这是一个 1D10P Stencil。而这样的 Stencil 根据组合一共有 9 个。其次需要计算 2 维平面上波的传播，这是一个 2D100P Stencil，而这样的 Stencil 根据排列组合一共有 6 个。因此 3DEW 每更新一个点，就需要进行 9 个 1D10P 加上 6 个 2D100P 的 Stencil 计算。

5 Stencil 代码在 Haswell 上的性能评估

5.1 Stencil 代码优化方法

我们对 3 个 Stencil 基准算例以及 3DFD、3DEW 分别进行了各种优化，如表 2 所示。版本 A 是最基本的串行实现。版本 B 是在 Stencil 最外层循环前添加#pragma openmp for 进行多线程并行。版本 C 是在最内层循环前添加#pragma simd 进行向量化。版本 D 和版本 E 都是对 cache 空间局部性进行的优化。tiling 是将数据分块以增加 cache 中的空间局部性。Tiling 是比较常用的优化方式，但其缺点是分块的大小需要根据应用以及所运行的硬件进行调整才能达到最佳性能。版本 F 和 G 将在 5.2 中介绍。实现了的✔用表示，有些因为技术原因没来得及实现的✘用表示。

优化版本	1D3P	2D5P	3D7P	3DFD	3DEW
A: baseline	✔	✔	✔	✔	✔
B: A + OpenMP	✔	✔	✔	✔	✔
C: B + Vector	✔	✔	✔	✔	✔
D: C + Tiling	✘	✔	✔	✔	✔
E: C + Cache Oblivious	✔	✔	✔	✘	✘
F: D + Peel	✔	✔	✔	✘	✘
G: D + Intrinsic	✔	✔	✔	✔	✔

表 2 不同 Stencil 的各种优化版本

5.2 编译器生成 vgather 指令的原因及对策

检查不同 Stencil 的版本 C、D、E 生成的汇编代码，我们发现编译器都生成了 vgather 指令。我们认为这是由于编译器在进行向量化时，遇到设置边界条件所使用的条件判断语句引起的。以 3D7P 为例，我们需要为 6 个方向都设置边界条件。比如 w 是当前元素西边的一个元素，通常情况是当前元素的索引减 1，即 c-1。但当前元素是最左边的那个元素时，那么 w 就应该还是这个元素自己，即 c。类似的，我们用这种条件判断语句来设定其他 5 个方向的边界条件。对于编译器而言，w 的取值到底是 c 还是 c-1 在编译时并不确定，要等到运行时知道 x 的数值时才能确定，这种访问就属于间接读入，因此也就会导致 ICC 在 HSW 上生成 vgather 指令。

有 2 种方法可以避免编译器生成 vgather。核心思想相同，即避免在定义数组元素时使用条件判断语句。分别是 Peel 优化和 Intrinsic load。限于篇幅，这 2 种优化的完整示例代码可以从 Github[11]上下载。

- Peel 优化

比较大的 Stencil 遇到边界而需要更换读入元素以避免越界的情况其实是非常少的，因此我们可以把边界条件的判断从内层循环剥离出去，因为绝大多数的内层循环是不需要边界条件的。以 3D7P 为例，x 方向上只有第一个元素 0 和最后一个元素 nx-1 需要进行边界条件判断，因此我们将这 2 个元素作为头尾剥离出来单独处理，把不需要进行边界判断的元素 1 到 nx-2 仍留在内层循环中。然后由于内层循环中的元素不会遇到边界，所以可以直接使用 e=c-1 和 w=c+1 消除对东边元素和西边元素的索引计算。这种方式既能避免生成 vgather 指令，又能提高向量化的效率。我们建议在 HSW 上对 Stencil 尽量采用这种优化，即表 2 中的版本 F。

- Intrinsic load

使用 intrinsic 编程和使用其他 C/C++函数库类似，都需要包含正确的头文件，然后调用相应的 intrinsic 函数。编译器通常会将 Intrinsic 函数逐一翻译成汇编指令，所不同在于使用汇编语言需要程序员自己管理寄存器，而使用 intrinsic 则还是由编译器来管理寄存器。

表 2 中版本 G 就是采用这种优化。以 3D7P 为例，首先，我们定义一组__m256d 的数据，256 表示总长为 256bit，d 表示每个元素为双精度浮点数（64bit），因此这个数据包含了 4 个元素。其次，使用_mm256_load_pd()读入 4 个连续地址的双精度元素到__m256d 中，且这 4 个元素需要 32 byte 对齐。使用_mm256_permute4x64_pd()设定边界条件。使用_mm256_set1_pd 将某一个常数广播到其余 3 个元素上。使用_mm256_fmadd_pd()执行(_a*_b)+_f2_t 的 FMA 操作。使用_mm256_store_pd()存储更新后的元素。最后，因为_mm256_load_pd()一次读入 4 个元素，因此内层循环的步长应该要加 4，而不再是加 1。

5.3 优化结果

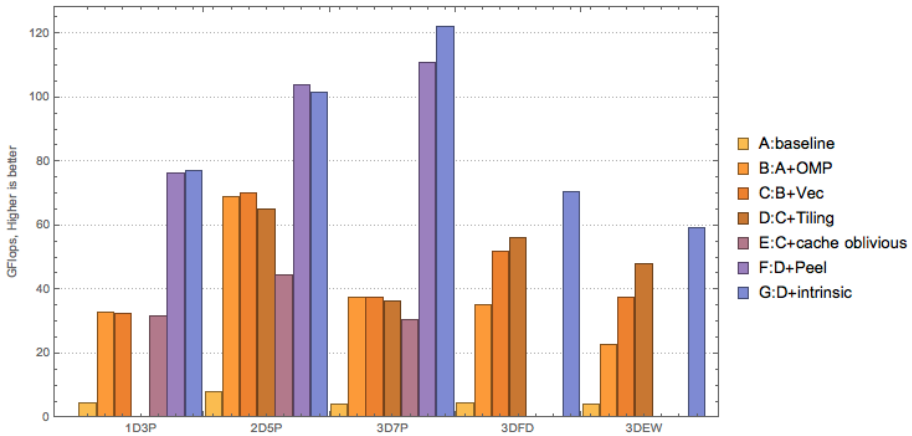

图 4 不同 Stencil 的优化版本在 Haswell 上的性能

我们通过检查汇编代码，发现版本 C、D、E 中都存在 vgather 指令，而版本 A、B、F 和 G 则都没有。这说明在内存循环前添加#pragma simd 进行向量化，由于边界条件设定中存在条件判断语句，编译器会生

成 vgather 指令。且 Tiling 和 cache oblivious 这些改进空间局部性的优化方法无法避免编译器生成 vgather 指令，只有使用 Peel 优化和 Intrinsic 优化才可以。

5 个 Stencil 不同优化版本的性能如图 4 所示。由此我们发现 AVX2 vgather 指令会降低 Stencil 在 HSW 上的性能。在消除 vgather 指令之后，各 Stencil 的版本 F 和 G 上均有 1.22X 至 3.88X 不等的性能提升。

6 vgather 指令实现分析

为了进一步探究 vgather 指令影响 Stencil 性能的原因，我们以 3D7P 为例，从 μops 层面分析 vgather 指令的实现。

6.1 使用 IACA 分析 vgather 的 μops

我们使用 IACA 检测 intrinsic load、以及 intrinsic vgather 的双精度和单精度版本。我们首先简单介绍如何解释 IACA 的结果。如图 5 所示，vgatherqpd 版本进行一个元素更新的 throughput 为 70.00 cycle。IACA 明确提示了 throughput 的瓶颈在 Gather 操作。vgatheqpd 指令在端口 2 和端口 3 上都执行了 2.0 cycle 的 AGU 和 2.0 cycle 的读入 μops。类似的，vfmadd132pd 指令被解码为 2 个 μops。在端口 0 和端口 1 上分别有 0.1 和 0.9 cycle，这是因为 HSW 在这 2 个端口上分别有一个 FMA 单元，而 IACA 显示的是统计结果，即假设有 100 条 FMA 指令，其中 10 条在端口 0 上执行，90 条在端口 1 上执行，那么统计结果就是端口 0 上有 0.1 个 cycle，端口 1 上有 0.9 个 cycle。另外，由于这条 FMA 指令还要访存，所以和 vmovupd 指令相似，它在端口 2 上执行了 1.0 cycle 的读入 μops 以及 1.0 cycle 的 AGU。最后的 vmovpd 指令是 store 操作，使用端口 2 和端口 3 的 AGU 生成地址的同时，在端口 4 执行 1.0 cycle 的写回 μop。我们注意到虽然端口 7 是专门为 store 准备的 AGU，但因为这是一个精简版本的 AGU，不能处理地址中带 index 的 store 操作。vgatherdps 版本进行一个元素更新的 throughput 更是高达了 105.00cycle，而且 IACA 也明确提示了 throughput 的瓶颈在 Gather 操作。而 load 版本进行一个元素更新的 throughput 仅为 16.25 cycle。限于篇幅，此处不能一一列出完整的 IACA 结果。完整的结果可以从 Github 上 [9]下载。

```
Throughput Analysis Report
--------------------------
Block Throughput: 70.00 Cycles       Throughput Bottleneck: GatherTpt

Port Binding In Cycles Per Iteration:
-------------------------------------------------------------------------------------------
|  Port  |  0   -  DV  |  1   |  2   -  D   |  3   -  D   |  4   |  5   |  6   |  7   |
-------------------------------------------------------------------------------------------
| Cycles | 15.9   0.0  | 8.6  | 25.5   24.5 | 25.5   24.5 | 2.0  | 20.3 | 20.2 | 0.0  |
-------------------------------------------------------------------------------------------

| Num Of |                    Ports pressure in cycles                     |    |
|  Uops  |  0  - DV  |  1  |  2  -  D  |  3  -  D  |  4  |  5  |  6  |  7  |    |
---------------------------------------------------------------------------------
...
|   1    |           |     |           |           |     | 1.0 |     |     |    | lea rcx, ptr [r12+rax*8]
|   1    |           | 0.1 |           |           |     | 0.9 |     |     |    | vpcmpeqd ymm0, ymm0, ymm0
|   1    |           |     | 0.5   0.5 | 0.5   0.5 |     |     |     |     | CP | vmovups ymm1, ymmword ptr [rip+0x527b]
|   5    | 1.0       |     | 2.0   2.0 | 2.0   2.0 |     | 1.0 | 1.0 |     | CP | vgatherqpd ymm3, qword ptr [rcx+ymm1*8], ymm0
|   2^   |           | 1.0 | 0.5   0.5 | 0.5   0.5 |     |     |     |     |    | vfmadd132pd ymm3, ymm2, ymmword ptr [rsp+0x120]
...
|   2    |           |     | 0.6       | 0.4       | 1.0 |     |     |     | CP | vmovupd ymmword ptr [rcx+rax*8], ymm10
...

Total Num Of Uops: 104
```

图 5 intrinsic vgather 双精度版本的 IACA 结果摘要

因为 vgatherqpd 指令需要读取 4 个双精度元素，而 IACA 显示 vgatheqpd 指令在端口 2 和端口 3 上都执行了 2.0 cycle 的 AGU 和 2.0 cycle 的读入 μops；而 vgatherdps 指令需要读入 8 个单精度元素，而 IACA 显示 vgatherdps 指令在端口 2 和 3 上都执行了 4.0 cycle 的 AGU 和 4.0 cycle 的读入 μops。因此**我们推测 vgather 指令会被解码成多个读入 μops，每一个 μops 对应读入一个元素**。IACA 提示 vgather 指令被解码成了 5 个 μops，但实际上 vgatherdps 至少应该被解码成 8 个读入 μops，因此我们推测 IACA 将所有解码成 4 个以上 μops 都标记为 5 个 μops。

注意到 load 版本进行一个元素更新的 throughput 仅为 16.25 cycle，vgatherqpd 版本进行一个元素更新

的throughput则增加到70.00 cycle，而vgatherdps版本进行一个元素更新的throughput更是高达了105.00cycle。那么 vgather 版本那么高的开销是从何而来的呢？分析图 5 中 IACA 的结果不难看出，vgather 版本在执行单元（即 Back-end）上花费的时间最高也就是 25.5cycle。这多出来的 70-25.5=44.5cycle 很可能是 Front-end 而带来的。由于我们已经知道每条 vgather 指令至少会被解码成 4 个 µops 以上，因此**我们推测这是由于使用 MSROM 进行解码所带来的高开销**。

6.2 Vgather 对 Stencil 性能的影响

- 单核性能

为了清楚了解 vgather 指令对 stencil 性能的影响，我们对比了 load 版本和 vgather 版本在 HSW 单核上的性能。如图 6 所示，纵轴是性能 GFlops，横轴是 3D7P 立方体一个方向上点的数量，因此其所占的存储空间可由公式(1)获得：

$$Data\_size=Cubic\_size^3*2*8 \quad (1)$$

其中 2 是因为使用了 Jacobi 方法更新元素，读和写各有一个数组。8 是因为双精度浮点大小为 8 byte。HSW 的 L1D 为 32KB，L2 为 256KB，L3 为 35MB，因此当 cubic size 小于 12 时，数据在 L1D 中；在 13-25 之间时，数据在 L2 中；26-129 之间时，数据在 L3 中；130 以上时，数据在内存中。当数据在 L1D 或 L2 中时，由于数据太小，不足以填满流水线，所以 load 版本和 vgather 版本都不能获得最高性能。当数据在 L3 时，load 版本的最高性能接近 vgather 版本性能的 3 倍。当数据在内存中时，load 版本的性能依然为 vgather 性能的 1.5 倍左右。

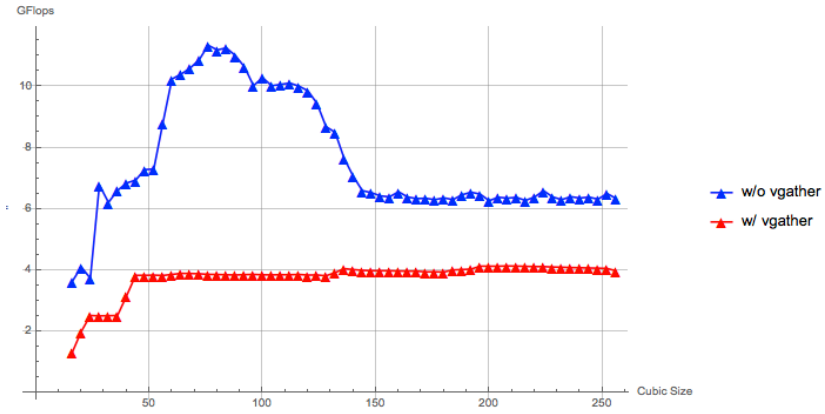

图 6 vgather 指令对单核性能的影响

与向量 load 指令相比，vgather 指令对 Stencil 性能的影响可能来源于 3 部分。第一，在读入 n 个元素时，vgather 指令需要执行 n 个 µops，而向量 load 指令只需执行 1 个 µops。第二，由于 vgather 指令使用了 VSIB 的寻址方式以及 mask 寄存器，为了执行 vgather 指令还需要配合一系列准备指令。如图 5 所示，首先花 1.0 cycle 使用 lea 指令载入 base 地址，然后花 1.0 cycle 的准备 mask 寄存器，再花 1.0 cycle 准备 vgather 所需的 index。这一系列准备指令需要 3.0 cycle，大于 vgatherqpd 本身的 2.0 cycle。而向量 load 指令则不需要这些准备指令。第三，如 6.1 中分析的，vgather 指令由于需要被解码为多个 µops，因此就由解码器中的 MSROM 进行解码，从而产生了较高的 Front-end 开销。

- 多核性能

我们还对比了 load 版本和 vgather 版本在 HSW 多核上的性能。如图 7 所示，由于 Stencil 代码受限于内存带宽，因此 load 版本在 9 核时达到了带宽饱和点(saturation point)。而 vgather 版本的性能由于没有达到饱和点，所以在 14 个核上的性能达到了线性加速，但其 14 核的性能还不如 load 版本 4 核的性能。

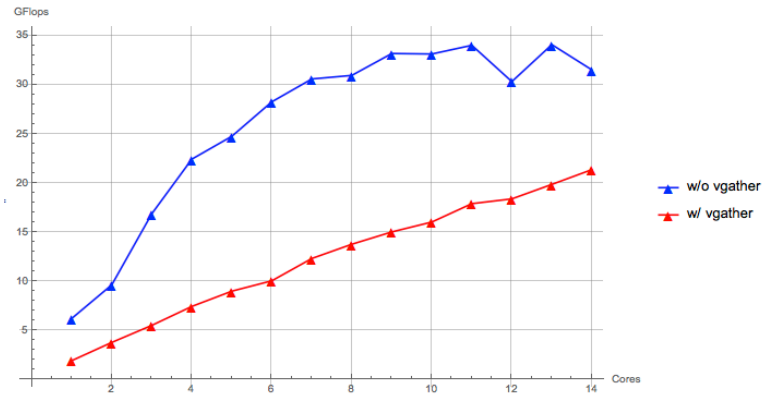

图 7　vgather 指令对多核性能的影响

7　总结与下一步工作展望

本文通过评估 AVX2 vgather 指令在 Intel Haswell 上的性能，得到了以下 3 个结论：

1） AVX2 vgather 指令会降低 Stencil 的性能。这个结论对其他具有良好空间局部性的应用也适用。

2） vgather 指令会为每个要读入的元素生成一个 μops。

3） 由于 vgather 指令需要由 MSROM 进行解码，因此会产生较高的 Front-end 开销。

据此，对 Stencil 这类具有良好空间局部性的应用在 Haswell 上的编程建议是要尽量避免让编译器生成 vgather 指令。有 2 种优化方式：1）不要在指定元素地址时进行条件判断。推荐对 Stencil 使用 Peel 优化；2）使用 intrinsic load 读入元素。

本文的工作有一些局限性，比如没有建立一个性能模型来细致分析 vgather 指令对 Stencil 性能的影响，这是下一步工作的主要方向之一。此外，Haswell 的下一代 Broadwell 对 vgather 指令的实现进行了一些改进，使用 Gather Index Table （GIT）后据说能减少最多 60%的 μop，因此我们也计划对 GIT 进行评估。

致谢　感谢上海交通大学高性能计算中心 π 集群提供测试环境。

References:

[1] J. Hofmann, J. Treibig, G. Hager, and G. Wellein, “Performance Engineering for a Medical Imaging Application on the Intel Xeon Phi Accelerator,” in 27th International Conference on Architecture of Computing Systems (ARCS2014). VDE, 2014.

[2] J. Lin, A. Nukada, S. Matsuoka, Modeling Gather and Scatter with Hardware Performance Counters for Xeon Phi, ACM/IEEE CCGrid15, Shenzhen, China

[3] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, “Exploring SIMD for Molecular Dynamics, Using Intel Xeon Processors and Intel Xeon Phi Coprocessors,” in IPDPS ’13. IEEE, May 2013, pp. 1085–1097.

[4] J. Hofmann, J. Treibig, G. Hager, and G. Wellein, “Comparing the performance of different x86 SIMD instruction sets for a medical imaging application on modern multi- and manycore chips ” WPMVP '14: Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing, New York, 2014

[5] D. Kusswurm, Modern X86 Assembly Language Programming 32-bit, 64-bit, SSE, and AVX, Apress, 2014

[6] IACA: https://software.intel.com/en-us/articles/intel-architecture-code-analyzer

[7] 3DFD: https://software.intel.com/en-us/articles/eight-optimizations-for-3-dimensional-finite-difference-3dfd-code-with-an-isotropic-iso

[8] C. W. J. Zhang, Z. Tian, “P- and s-wave separated elastic wave equation numerical modeling using 2d staggered-grid,” in SEG/San Antonio 2007 Annual Meeting, 2007.

[9] AVX2-vgather 的部分源代码以及 IACA 结果: https://github.com/jameslinsjtu/AVX2-vgather