Modeling Gather and Scatter with Hardware Performance Counters for Xeon Phi

James Lin*†, Akira Nukada†, Satoshi Matsuoka†
*Shanghai Jiao Tong University, China
†Tokyo Institute of Technology, Japan
james@sjtu.edu.cn, nukada@smg.is.titech.ac.jp, matsu@is.titech.ac.jp

***Abstract*—Intel Initial Many-Core Instructions (IMCI) for Xeon Phi introduces hardware-implemented Gather and Scatter (G/S) load/store contents of SIMD registers from/to non-contiguous memory locations. However, they can be one of key performance bottlenecks for Xeon Phi. Modeling G/S can provide insights to the performance on Xeon Phi, however, the existing solution needs a hand-written assembly implementation. Therefore, we modeled G/S with hardware performance counters which can be profiled by the tools like PAPI. We profiled Address Generation Interlock (AGI) events as the number of G/S, estimated the average latency of G/S with `VPU_DATA_READ`, and combined them to model the total latencies of G/S. We applied our model to the 3D 7-point stencil and the result showed G/S spent nearly 40% of total kernel time. We also validated the model by implementing a G/S-free version with intrinsics. The contribution of the work is a performance model for G/S built with hardware counters. We believe the model can be generally applicable to CPU as well.**

***Keywords*-Performance modeling; Gather and Scatter; Xeon Phi; Hardware performance counters**

I. Introduction

Gather and Scatter (G/S) for vector operations have been introduced by Intel IMCI and hardware-implemented on Intel Xeon Phi (code name: Knights Corner). They load/store the contents of SIMD registers from/to non-contiguous memory locations, however, they are one of key performance bottlenecks for Xeon Phi[1] and cannot be eliminated by the traditional optimizations [2]. Modeling G/S can provide insights to performance on Xeon Phi, however, existing solutions like ECM [3] needed a hand-written assembly implementation [4], which most of scientific applications do not have, for counting hardware counters.

Instead of implementing assembly codes, we model G/S with hardware counters, which can be profiled by the tools like PAPI[5]. We profile Address Generation Interlock (AGI) events as the number of G/S, estimate the average latency of G/S with `VPU_DATA_READ`, and combine them to model the total latencies of G/S. We apply our model to the 3D 7-point stencil and the result shows G/S spend nearly 40% of total kernel time. We also validate the model by implementing a G/S-free version with intrinsics.

Concisely, the contribution of our work is a performance model for G/S built with hardware counters.

The rest of this paper is structured as follows. Section II discusses related work. Section III introduces the background of Xeon Phi. Section IV explains how we build the model with hardware counters. Section V shows we apply the model to the 3D 7-point stencil and validate the result, leading to the ongoing work in Section VI.

II. Related Work

This section discusses related work on performance models for Xeon Phi, performance modeling with hardware counters, and G/S as performance bottlenecks on Xeon Phi.

A. Performance models for Xeon Phi

Treibig and Hager [6] presented an analytic performance model for bandwidth limited loop kernels on three modern x86-type quad-core architectures. The analytic approach used in the model can be also applied to Xeon Phi [4] to model kernel runtime but impossible to predicate runtime information needed to model G/S.

Ramos et al. [7] have proposed an intuitive performance model for cache-coherent architectures and demonstrated its use with Xeon Phi. Their model explored the communications among cores in a shared memory environment while ours focus on data transfers between cores and L1 cache.

Peraza et al. [8] studied the stream and stencil idioms and modeled the performance of these idioms on Xeon Phi. The model focused primarily on how memory behavior of applications related to their performance on Xeon Phi.

B. Performance modeling with hardware counters

Zhang and Owens [9] have developed a semi-empirical model to analyze and predict GPU execution time. Hong and Kim [10] presented detailed analytical models to predict the performance of GPU kernels. These models require low-level information specific to GPU, such as the amount of warps, hence make them difficult to apply to Xeon Phi directly.

C. G/S on Xeon Phi

Pennycook et al. [1] analyzed G/S performance bottlenecks in the molecular dynamics codes and the challenges that they pose for obtaining benefits from SIMD execution. They listed a breakdown of the number of static instructions in the inner-most loop of Sandia's miniMD benchmark and showed G/S instructions account for 64% of non-arithmetic instructions for Xeon Phi. To predicate performance more accurately, our model uses runtime hardware counters instead of static instructions.

Hofmann [3] counted the number of gathers with `rax` register for a hand-written assembly implementation of RabbitCT benchmarking framework. However, usually most scientific applications do not have assembly implementations. Our model use hardware counters which can be easily profiled by programmers with the tools like PAPI [5].

III. Xeon Phi

This section introduces the background of Xeon Phi related to our model including architecture, hardware counters, vector pipeline (U-Pipe), and G/S primitives.

A. Overview

Xeon Phi composes of up to 61 CPU cores connected on-die through Core Ring Interconnects (CRI). The cores are based on a modified version of the P54C design. Each core is a fully functional, in-order core and capable of switching between up to four hardware threads in a round-robin manner. Various programming modes are available on Xeon Phi including the native model and the offload model [11].

B. Hardware performance counters

Performance Monitoring Unit (PMU), a set of special-purpose registers built in Xeon Phi, can count occurrences of particular hardware events, including the number of cycles executed by the core `CPU_CLK_UNHALTED` and the number of loads and stores seen by a threads L1 data cache `DATA_READ_OR_WRITE`, to help better understand how pipelines, caches, and etc. are being utilized. These hardware counters can be profiled by the tools like Intel VTune Amplifier XE or PAPI.

C. Vector pipeline (U-Pipe)

Each core of Intel Xeon Phi features a scalar pipeline (V-pipe) and a vector pipeline (U-pipe). Connecting to the U-pipe, the Vector Processing Unit (VPU) implements IMCI vector extensions.

Once a vector instruction is decoded at the D2 stage of the main pipeline, it enters U-pipe, shown in Figure 1. At the E stage the VPU detects if there is any dependency stall. At the VC1/VC2 stage the VPU does loads and stores including G/S if necessary. At the V1-V4 stages it does the four-cycle multiply/add operations, followed by the WB stage.

D. G/S Primitives

To fetch all elements pointed at by register `%zmm30`, gather primitives need to be executed in a loop (Listing 1) implemented with the conditional jump instructions `jknzd` and `jkzd` until the vector mask register `%k3` contains all zero bits.

Furthermore, because one gather primitive `vgatherdpd` can only reads the elements on a single Cache Line (CL), to gather the element spreading across multiple CL, the single gather primitive is believed to be hardware implemented as a loop. Therefore, the latency for a single gather $\mathcal{T}_{gs,avg}$ will increase when the distance between the data to be gathered increases [3].

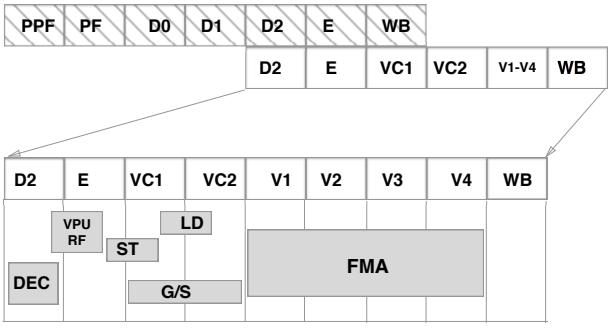

Figure 1: Vector pipeline in Xeon Phi

Listing 1: Gather in a loop

```
..L47:
vgatherdpd (%r12,%zmm30,8), %zmm25{%k3}
jkzd       ..L46, %k3     # Prob 50%
vgatherdpd (%r12,%zmm30,8), %zmm25{%k3}
jknzd      ..L47, %k3     # Prob 50%
```

IV. Modeling G/S with Hardware Counters

This section explains how we profile the hardware counter Address Generation Interlock (AGI) as the number of G/S, estimate the average latency of G/S with the hardware counter `VPU_DATA_READ`, and combine them to model overall latencies of G/S. Modeling the G/S-free kernel execution time is outside the scope of this paper. Therefore, we assume the execution time of the G/S-free kernel is already known by programmers, either with experimentation or existing performance models.

A. Profiling AGI as number of G/S, $N_{gs,total}$

Since memory banks are not known at the start of the instruction, G/S occur AGI, which has three clock cycles latencies and can be profiled with the hardware counter `PIPELINE_SG_AGI_STALLS`. E.g. one AGI needs to be inserted between two instructions to calculate the actual linear address from `rbx` before fetch happens (Listing 2). Therefore, the number of G/S $N_{gs,total}$ can be estimated by profiling `PIPELINE_SG_AGI_STALLS`.

Listing 2: AGI occurs between the two instructions

```
add rbx, 4;
mov rax [rbx];
```

B. Estimating average latency of G/S, $\mathcal{T}_{gs,avg}$

As explained in Section III-D, the latency of a single gather instruction $\mathcal{T}_{gs,avg}$ is determined by data distribution

Table I: Latency of gather primitive (*clock cycles*) [4]

R_{cl}	$\mathcal{T}_{gs,avg}$
4 per CL	3.7
2 per CL	2.9
1 per CL	2.3

per CL (Table I) R_{cl} which can be calculated by Equation (1). We can obtain $N_{vpu,read}$ by profiling the hardware counter `VPU_DATA_READ` and N_{gs} either by calculating analytically for loop-based codes or using $N_{gs,total}$ for non loop-based codes.

$$R_{cl} = \frac{N_{vpu,read}}{N_{gs,total}} \tag{1}$$

C. Modeling total latencies of G/S, $\mathcal{T}_{gs,total}$

We now show how to combine $N_{gs,total}$ and $\mathcal{T}_{gs,avg}$ to model total latencies of G/S. We assume each G/S causes a single AGI, hence the cost of each G/S should include two parts: 1) **three clock cycle** latencies for each AGI $\mathcal{T}_{agi}$; and 2) the average latency of G/S itself $\mathcal{T}_{gs,avg}$. Equation (2) shows $\mathcal{T}_{gs,total}$ the total latencies of G/S and the simplified model.

$$\mathcal{T}_{gs,total} = N_{agi} * (\mathcal{T}_{agi} + \mathcal{T}_{gs,avg}) \tag{2}$$

V. Evaluation and Validation

This section shows how we apply the model to a 3D 7-point stencil to predicate total latencies of G/S accurately, and validate the result by removing all G/S with SIMD intrinsics. We evaluate the code in the native mode with 60 threads, one thread for one core, on a Intel Xeon Phi 5100P from a single node of "CPU+GPU+Xeon Phi" supercomputer π in Shanghai Jiao Tong University, with Intel C/C++ Compiler 14.0 and PAPI 5.1.

A. Evaluation with a 3-D 7-point Stencil

The 3D 7-point Jacobi stencil (Figure 2) needs seven gather operations to fetch elements from f[j] to f[jb] in each 3-level nested loops (Listing 3).

Listing 3: Kernel loop for 3D 7-point Stencil

```
for(jz=0; jz < nz; jz++) {
    for(jy=0; jy < ny; jy++) {
        for(jx=0; jx < nx; jx++) {
        fn[j] = cc*f[j]
              + ce*f[je] + cw*f[jw]
              + cn*f[jn] + cs*f[js]
              + ct*f[jt] + cb*f[jb];
    } //end of jx
  }//end of jy
}//end of jz
```

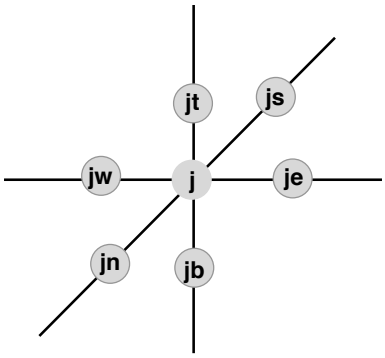

Figure 2: 3D 7-point stencil

As the 3D 7-point stencil is loop-based, we utilize semi-empirical performance modeling [12] to obtain $N_{gs,total}$ as N_{agi} by profiling `PIPELINE_SG_AGI_STALLS`, and $N_{vpu,read} = 10$ by profiling `VPU_DATA_READ` for each point. $N_{gs,total} = 7$ as seven gather operations are needed for each point (Listing 3). Therefore, we estimate $\mathcal{T}_{gs,avg} = 2.6$ because $R_{cl} = \frac{10}{7} \approx 1.5$ according to Equations (1) and R_{cl} for 1 per CL is 2.9 and 2 per CL is 2.3 (Table I). In summary, $\mathcal{T}_{gs,avg}$ of each point in the 3D 7-point stencil can be modeled with Equation (2) as:

$$\mathcal{T}_{gs,total} = N_{agi} * (3 + 2.6)$$

For the kernel execution time of each point, we measure **3.37**μs. For $\mathcal{T}_{gs,total}$, the simplified model predicts **1.34**μs. Therefore, our model predicts G/S spend nearly 40% of the total elapsed time. It shows how seriously G/S slow down the performance of the stencil on Xeon Phi.

B. Validation by removing G/S with intrinsics

We validate the predicated result by removing G/S with intrinsics. Firstly we need to determine the relationship between a kernel with G/S and the kernel without G/S. As G/S is only available in U-Pipe and cannot be paired in V-Pipe to overlap with Fused Multiply-Add (FMA) in U-Pipe [11], the elapsed time for a kernel with G/S $T_{ker,gs}$ can be modeled as Equation (3), where $T_{ker,gs-free}$ stands for the elapsed time for the kernel without G/S and f_{core} is the frequency of Xeon Phi, 1.05G for 5110P.

$$T_{ker,gs} = T_{ker,gs-free} + \frac{\mathcal{T}_{gs,total}}{f_{core}} \tag{3}$$

We use unaligned vector load intrinsics (Listing 4) to avoid gather primitives, where `_mm512_loadunpacklo_pd` loads the low 64-byte aligned portion of unaligned double word stream f[j] and `_mm512_loadunpackhi_pd` loads the high portion, then unpack mask-enabled elements that fall in that portion, and store those elements in a float64 vector. No G/S

primitives are found in the generated assembly codes after we apply these intrinsics.

Listing 4: Intrinsics to avoid gather primitives

```
_a = _mm512_loadunpacklo_pd(_a, f+j);
_a = _mm512_loadunpackhi_pd(_a, f+j+8);
```

For $T_{ker,gs-free}$, we measure 2.02μs. This makes $T_{ker,gs}$ predicated by the model 3.36μs very closely with the time we measured, 3.37μs. The validation shows our model can accurately predicate the latency of G/S for the stencil on Xeon Phi.

VI. Ongoing Work

Some limitations of the model are worth noting, such as the 3D 7-point stencil we evaluated does not need scatter operations, and it may hit CL for gather operations. Ongoing work therefore to improve the model include:

1) **More test cases**. We plan to evaluate with Molecular Dynamics (MD) to model scatter, and sparse matrix-vector multiplication (SpMV) for different types of gather operations.
2) **More ISAs**. Intel Advanced Vector Extensions (AVX)-2 adopted by Haswell CPU supports only gather which is similar to the IMCI gather but without a mask register to hold the completion mask. AVX-512 adopted by next generation Xeon Phi (code name: Knights Landing) provides both gather and scatter which can load/store all elements in one instructions, regardless how many CL the elements distribute. We plan to compare and model G/S among AVX2, AVX512, and IMCI.

Acknowledgement

We thank Prof. Takayuki Aoki and Takahiro Fujiyama of Tokyo Institute of Technoloy provided the original source code of the 3D 7-point Stencil. We also thank Shuo Li of Intel for his valuable suggestions. This research is supported by China National High-Tech R&D Plan (863 plan) 2014AA01A302. James also greatly acknowledges support from Japan JSPS RONPAKU (Thesis PhD) Program.

References

[1] 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.

[2] W. Xue, C. Yang, H. Fu, X. Wang, Y. Xu, L. Gan, Y. Lu, and X. Zhu, "Enabling and Scaling a Global Shallow-Water Atmospheric Model on Tianhe-2," in *IPDPS '14*. IEEE, May 2014.

[3] J. Hofmann, "Performance Evaluation of the Intel Many Integrated Core Architecture for 3D Image Reconstruction in Computed Tomography," Master's thesis, Friedrich-Alexander-University Erlangen-Nuremberg, 2013.

[4] 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, pp. 1–8.

[5] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *ACM/IEEE 2000 Conference Supercomputing (SC00)*, Nov 2000, pp. 42–42.

[6] J. Treibig and G. Hager, "Introducing a performance model for bandwidth-limited loop kernels," in *Parallel Processing and Applied Mathematics*, 2010.

[7] S. Ramos and T. Hoefler, "Modeling communication in cache-coherent SMP systems: a case-study with Xeon Phi," in *HPDC '13: Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, New York, USA, Jun. 2013, p. 97.

[8] J. Peraza, A. Tiwari, M. Laurenzano, L. Carrington, W. A. Ward, and R. Campbell, "Understanding the performance of stencil computations on Intel's Xeon Phi," in *2013 IEEE International Conference on Cluster Computing (CLUSTER13),*. IEEE, 2013, pp. 1–5.

[9] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *17th IEEE International Symposium on High Performance Computer Architecture (HPCA 2011)*, 2011, pp. 382–393.

[10] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*. ACM, Jun. 2009.

[11] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools*. Apress, 2013.

[12] T. Hoefler, W. Gropp, W. Kramer, and M. Snir, "Performance modeling for systematic performance tuning," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC11)*, 2011, pp. 1–12.