Out-of-Step Pipeline for Gather/Scatter Instructions

Yi Ge[*], Katsuhiro Yoda[*], Makiko Ito[*], Toshiyuki Ichiba[*], Takahide Yoshikawa[*], Ryota Shioya[†] and Masahiro Goshima[‡]

[*] Fujitsu Ltd.
Tokyo, Japan
Email: katsu-t@fujitsu.com

[†] The University of Tokyo
Tokyo, Japan
Email: shioya@ci.i.u-tokyo.ac.jp

[‡] National Institute of Informatics
Tokyo, Japan
Email: goshima@nii.ac.jp

***Abstract*—Wider SIMD units suffer from low scalability of *gather/scatter* instructions that appear in sparse matrix calculations. We address this problem with an *out-of-step* pipeline which tolerates bank conflicts of a multibank L1D by allowing element operations of SIMD instructions to proceed out of step with each other. We evaluated it with a sparse matrix-vector product kernel for matrices from *HPCG* and *SuiteSparse Matrix Collection*. The results show that, for the SIMD width of 1024 bit, it achieves 1.91 times improvement over a model of a conventional pipeline.**

I. Introduction

Demands for sparse matrix calculation have been increasing. Examples are given as follows: NVIDIA implemented the Sparse Tensor Core for sparse matrix in AI workloads. Graph processing also requires large sparse matrices for large graphs. The work that won the 2021 ACM Gordon Bell Special Prize for HPC-Based COVID-19 Research simulates aerosolized droplet with a linear system in a large sparse matrix. The *HPCG* benchmark was developed to reflect the characteristics of these programs [1]. HPCG solves a linear system in a large sparse matrix by the CG method. HPCG is now one of the three categories of TOP500 along with HPL (HP Linpack).

A wider SIMD unit contributes greatly to dense but little to sparse matrix calculations, which is typically seen in the TOP500 results of the Supercomputer Fugaku. Though it won the first ranks in HPL/HPCG in 2021, the efficiencies were as high as 82.3% for HPL but as low as 3.0% for HPCG.

We found that the cause of this low efficiency is a scalability problem of SIMD *gather* instructions that inevitably appear in sparse matrix calculation. We address this problem with a *many-bank level-1 data cache* (*MBL1D*), and a newly proposed *out-of-step* pipeline which does not reduce the bank conflicts on MBL1D but tolerates them by allowing element operations of SIMD instructions to proceed out of step with each other.

II. Out-of-Step Backend Pipeline

Fig. 1 shows conceptual block diagrams of conventional *in-step* (*InS*) and our *out-of-step* (*OoS*) backend pipelines. Each of the pipelines has two SIMD pipes of two lanes, and the left pipe executes gather/scatter instructions to the MBL1D.

a) In-Step Pipeline: The InS pipeline can be considered as a single pipeline. Between adjacent stages is a single-entry pipeline register that spans all the lanes. Instructions issued to the lanes flow through the stages in step with each other.

At the cycle $t = t_0$, in the InS pipeline are SIMD instructions A, B, and C, each of which has two element operations, such as A_0 and A_1. The instruction C depends on the gather instruction A, and receives the gathered results through the bypass. A_0 and A_1 access the same bank, resulting in a bank conflict. In this situation, no SIMD instructions can proceed while keeping the InS principle, for the following reasons:

- $\boldsymbol{A_1}$ cannot, because it loses in the bank conflict.
- $\boldsymbol{B_1}$ cannot, or it will overwrite A_1.
- $\boldsymbol{C_1}$ cannot, or it will miss the result of A_1 through the bypass.

This situation is equivalent to an L1D miss in the sense that a load target cannot be retrieved at the timing that the scheduler has assumed. Out-of-order cores generally reschedule instructions on an L1D miss, because pipeline stall is difficult to implement due to heavy load of the write enable signals to deassert [2]. However, rescheduling on a bank conflict diminishes the gain of MBL1D, because the bank conflict rate is extremely higher than the L1D miss rate.

b) Out-of-Step Pipeline: In the OoS pipeline on the right side of Fig. 1, the single-entry pipeline register immediately before the execution stages is replaced with small FIFO buffers for the lanes. In addition, the downstream pipeline registers are divided for the lanes. As a result, the downstream part of the lanes from the buffers can independently proceed.

In the OoS pipeline in the figure, the positions of the element operations are at the cycle $t = t_0 + 1$. These element operations can proceed out of step with each other, as follows:

- ***A*** A_1 lost in the bank conflict has remained in the buffer, and in turn accesses the bank.
- ***B*** B_1 has been stored in the second entry of the buffer without overwriting A_1. B_0 accesses another bank in parallel with A_1.

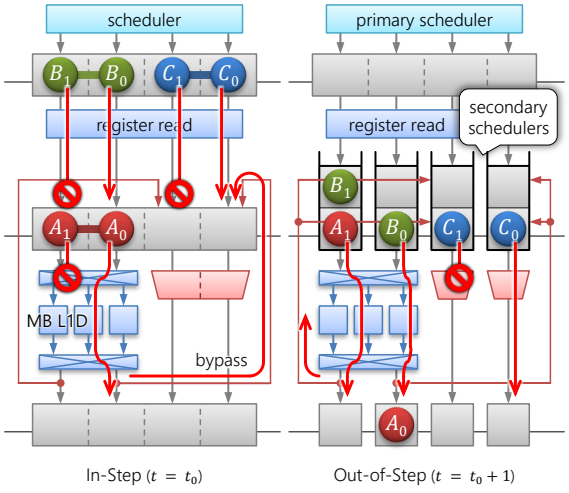

Fig. 1. In-/Out-of-Step backend pipelines.

978-3-9819263-7-8/DATE23/© 2023 EDAA

C C_0 proceeds with the result of A_0 which has been passed through the bypass. C_1 waits for the result of A_1 in the buffer for another cycle.

In this way, the FIFO buffers work as the *secondary schedulers* for element operations, and the OoS pipeline can proceed on a bank conflict without pipeline stall or instruction rescheduling.

The area and energy consumption of the secondary schedulers are negligibly small compared with the out-of-order primary scheduler. Matrix-based control [3] can also be applied to the secondary schedulers.

III. Evaluation

We evaluated the OoS pipeline with a Sparse Matrix-Vector product (*SpMV*) kernel in HPCG for the *SELL-C-σ* format [4] on a fully cycle-accurate simulator Onikiri2 [5]. HPCG is designed to have computations and data access patterns commonly found in scientific applications [1]. In HPCG, most of the execution time is occupied by SpMV and SpMV excluding the diagonal elements. Therefore, using the SpMV kernel in HPCG as the benchmark highlights the effect of the OoS control for the scientific applications. However, the matrix in HPCG is regular and easy for MBL1D. Thus, we also evaluated large sparse matrices from the *SuiteSparse Matrix Collection* [6] selected by Kreutzer et al. to evaluate their SELL-*C*-*σ* format [4].

Tab. I shows the parameters of the baseline model. We evaluated the SIMD widths of v = 8, 16, 32, and found that the results for the different widths are quite similar. Thus, we disclose only the result for the width of v = 16 (1024 b).

We evaluated the following four models with different L1D and pipeline configurations:

NMB Non-MB L1D. A gather instruction occupies the non-multibank L1D with 2 ports for $v/2$ cycles.

InS InS pipeline with MBL1D. Instructions that cannot proceed on a bank conflict, such as *A*, *B*, and *C* in Fig. 1 (left), are rescheduled as on an MBL1D miss.

OoS(s) OoS pipeline with MBL1D and the s-entry secondary schedulers. Instructions are rescheduled not on a bank conflict but on an MBL1D miss.

FP Full-Port L1D. It is ideal and free from bank conflicts, but causes L1D misses as with the other models.

Fig. 2 shows a scatter plot of the efficiency versus bank conflict rate. The efficiency is for the peak performance given by 2 FMA instructions in 3 cycles required to issue 4 SIMD load and 2 gather instructions. The bank conflict rate is measured at the MBL1D bank arbiters of the OoS(∞) model.

TABLE I
Parameters for evaluated models.

ISA	RISC-V RV64IMFD w/ "V" ext. v0.10				
Frontend	fetch to dispatch: 6 insns				
Primary Scheduler	int, fp, mem, SIMD, SIMD mem: 32 µOPs, 2 issue each				
Stages	fetch to dispatch: 7, sched: 1, issue: 2, reg read, WB: 2				
Exec Lat	int: 1, others: 6				
Phy Regs	GR, FR, VR: 256 each				

Branch Pred		gshare			
SIMD Width		8, 16, 32 double words			
MB-L1D	Banks	32, 64, 128 (SIMD width ×4)			
	Ports	1-read/write per bank			
	Size, etc	128 KB	8-way	512 B line	5τ
L2C	Size, etc	1 MB	8-way		12τ
LLC	Size, etc	8 MB	16-way		40τ
Main Memory		∞			200τ
Prefetcher		stream-based (L1, L2, LLC)			

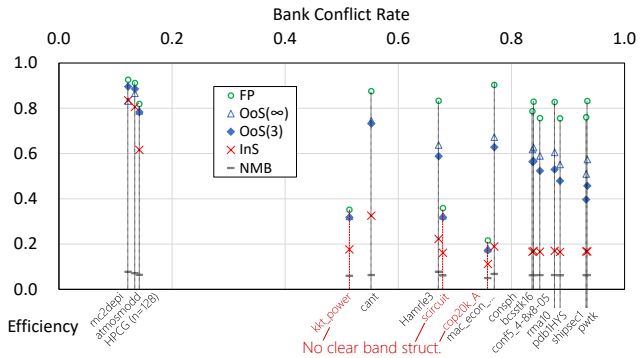

Fig. 2. Efficiency versus bank conflict rate for each matrix.

The efficiency of the NMB model is limited to approximately 0.08 bound by the throughput of gather instructions for the 2 ports of its non-MB L1D. InS and OoS(3) show clear inverse correlation; while InS is sensitive, OoS(3) tolerates higher bank conflict rates. Three matrices show lower efficiency for their rates even for FP, because these matrices do not have clear band structure and the prefetchers do not efficiently work.

We evaluated the efficiency of OoS(s) for the secondary scheduler sizes $s = 2, 3, \ldots, 7, \infty$. The efficiency of OoS(3) is close to that of OoS(∞) for all the matrices. A buffer full state is harmful only if frequent full states in a lane make other lanes idle via stall of instruction issuing. The efficiency of OoS(∞) free from this situation is determined simply by the throughput of MBL1D for the bank conflict patterns. This is almost true for OoS(3) whose efficiency is close to that of OoS(∞). We can conclude that the OoS control with the 3-entry secondary schedulers makes full use of MBL1D for these sparse matrices.

IV. Conclusion

We described the out-of-step pipeline which tolerates bank conflicts in the many-bank L1D by allowing element operations of SIMD instructions to proceed out of step with each other. We evaluated it with a SpMV kernel in HPCG [1] for the SELL-*C*-*σ* format [4] on a fully cycle-accurate simulator Onikiri2 [5]. A wide variety of matrices are selected from the SuiteSparse Matrix Collection [6] and HPCG. The results show that, for the SIMD width of 1024 b/2048 b, it achieves 76.8%/85.1% efficiency for an ideal model with full-port L1D free from bank conflicts, and 1.91/1.82 times improvement over a model of a conventional in-step pipeline.

References

[1] J. Dongarra, M. A. Heroux, and P. Luszczek, "HPCG benchmark: a new metric for ranking high performance computing systems," University of Tennessee, Tech. Rep. ut-eecs-15-736, Jan. 2015.

[2] A. Perais, A. Seznec, P. Michaud, A. Sembrant, and E. Hagersten, "Cost-effective speculative scheduling in high performance processors," in *Proc. ISCA*, 2015, pp. 247–259.

[3] M. Goshima *et al.*, "A high-speed dynamic instruction scheduling scheme for superscalar processors," in *Proc. MICRO*, 2001, pp. 225 – 236.

[4] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop, "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units," *SIAM J. Scientific Computing*, vol. 36, no. 5, pp. C401–C423, 2014.

[5] R. Shioya. Onikiri2. https://github.com/onikiri/onikiri2

[6] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1–25, Dec. 2011.