Optimizing Multigrid Poisson Solver of Cartesian CFD Code CUBE

Kazuto Ando
Operations and Computer Technologies Division, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan
kazuto.ando@riken.jp

Rahul Bale
Complex Phenomena Unified Simulation Research Team, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan

Keiji Onishi
Complex Phenomena Unified Simulation Research Team, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan

Kiyoshi Kumahata
Operations and Computer Technologies Division, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan

Kazuo Minami
Operations and Computer Technologies Division, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan

Makoto Tsubokura
Complex Phenomena Unified Simulation Research Team, RIKEN Center for Computational Science (R-CCS)
Kobe, Japan

ABSTRACT

We demonstrate an optimization of multigrid Poisson solver of Cartesian CFD code "CUBE (Complex Unified Building cubE method)". CUBE is a simulation framework for complex industrial flow problem, such as aerodynamics of vehicles, based on hierarchical Cartesian mesh. In incompressible CFD simulation, solving pressure Poisson equation is the most time-consuming part. In this study, we use a cavity flow simulation as a benchmark problem. With this problem, multigrid Poisson solver dominates 91% of execution time of the time-step loop. Specifically, we evaluate the performance of Gauss-Seidel loop as a computational kernel based on "Byte per Flop" approach. With optimization of the kernel, we achieved 9.8x speedup and peak floating point performance ratio increased from 0.4% to 4.0%. We also measured parallel performance up to 8,192 nodes (65,536 cores) on the K computer. With optimization of the parallel performance, we achieved 2.9x–3.9x sustainable speedup in the time-step loop.

KEYWORDS

CFD, Poisson equation, Multigrid, Cartesian mesh, Byte per flop, Roofline model

ACM Reference Format:
Kazuto Ando, Rahul Bale, Keiji Onishi, Kiyoshi Kumahata, Kazuo Minami, and Makoto Tsubokura. 2019. Optimizing Multigrid Poisson Solver of Cartesian CFD Code CUBE. In *Proceedings of Supercomputing Conference (SC19).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/1122445.1122456

SC19, November 17–22, 2019, Colorado, Denver

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In incompressible CFD simulation, pressure Poisson solver is the most major computational kernel. Besides, in the large scale simulation, the kernel is often solved with Mutigrid method in terms of convergence performance. "CUBE (Complex Unified Building cubE method)" is a simulation framework for complex industrial flow problem, such as aerodynamics of vehicles, based on hierarchical Cartesian mesh, developed by Complex Phenomena Unified Simulation Research Team in RIKEN R-CCS [2]. CUBE supports "BCM (Building Cube Method)" [1, 5, 6] and "IBM (Immersed Boundary Method)" [4]. With BCM, flow around a body is discretized with hierarchical mesh comprised with cubic units called "CUBE". Each CUBE is subdivided into the same number of "cell"s. Boundary cells in a CUBE have to be exchanged between neighbouring CUBEs. On the other hand, with IBM, a body is considered as a boundary immersed in flow field. In this study, we adopt "cavity flow" problem which involves single-level mesh and only flow field calculation for optimizing basic performance of the target code. The Poisson solver (V-cycle multigrid) dominates almost all of the total cost of a the time-step loop (91%). We optimize the multigrid Poisson solver of CUBE by investigating computational kernel loop and parallel performance. Computational kernel performance is evaluated based on "Byte per Flop" value (reciprocal of "operational intensity" in Roofline model [7]). Using this value, we can estimate theoretical upper limit of floating point peak performance and potential of optimization of a computational kernel that is bound by memory bandwidth [3]. Main strategy of our kernel optimization is minimizing Byte per Flop the kernel demands. Besides, we optimize parallel performance coping with thread level cost imbalance. Finally, we measured weak scaling up to 8,192 nodes (65,536 cores) on the K computer using the optimized code.

2 PERFORMANCE EVALUATION

The cost per time-step is dominated by the Poisson solver which accounts for 91%. Specifically, the Gauss-Seidel method based smoother accounts for 73% of the cost. The Gauss-Seidel loop included in this smoother is the highest cost component of the time-step loop (50%).

Routine for local halo exchange between inner cubes is called from many parts in program and its cost is 26% of the time-step loop.

We use the "Byte per Flop" performance model to evaluate theoretical peak performance of computational kernel loop. Byte per flop a program demands is $\frac{\text{Number of memory accesses} \times \text{Precision [Byte]}}{\text{Number of floating point operations}}$. Byte per flop a system provides is $\frac{\text{Effective memory bandwidth [Byte/sec]}}{\text{Peak floating point performance [FLOPS]}}$. "Effective memory bandwidth" is estimated with running STREAM Triad benchmark on the target system. Attainable peak performance ratio is $\frac{BF_{effective}}{BF_{demand}} \times 100$ [%].

We apply this model to the highest cost loop, Gauss-Seidel loop. Considering 32 iterations, number of floating point operations is (5 Add + 1 Subtract + 2 Multiply) × 32 Iterations = 256. Memory access is (15 × 2 + 3) cache lines × 128 Byte = 4224 Byte. Byte per flop which a program demands is $\frac{4224 \text{ Byte}}{256 \text{ Operations}} = 16.5$. Byte per flop the target system provides is $\frac{46 \text{ [GByte/sec]}}{128 \text{ [GFLOPS]}} = 0.36$. Therefore, the attainable peak performance ratio is $\frac{0.36}{16.5} \times 100 = 2.18\%$.

3 OPTIMIZATION OF COMPUTATIONAL KERNEL LOOP

We optimized Gauss-Seidel loop using Red-Black ordering as a computational kernel.

In the original code, physical values are contained in 5 dimensional array named "qcnt", which contains pressure value and is a deep nested struct member, and access of the array elements involved stride access. First, we used the pointer to directly refer to the array "qcnt" to decrease the integer operations for the address calculation. Furthermore, we applied swapping array dimensions and dividing array elements for "Black" cells and "Red" cells of Red-Black ordering. These care for stride access in first dimension of "qcnt" reduces wait time due to floating point load memory access. This result in a 9.8x speed up, and the peak ratio increases from 0.4% to 4.0% (Figure.1).

Finally, we applied Byte per flop model to the computational kernel of the optimized code. Considering 32 iterations, number of floating point operations is (5 Add + 1 Subtract + 2 Multiply) × 32 Iterations = 256. Memory access is (2 Load + 1 Store × 2) × 3 Cache lines × 128 Byte = 1536 Byte. Byte per flop which the program demands is $\frac{1536 \text{ Byte}}{256 \text{ Operations}} = 6.0$. Byte per flop for which the target system provides is 0.36. Attainable peak performance ratio is $\frac{0.36}{6.0} \times 100 = 6.00\%$.

4 OPTIMIZATION OF PARALLEL PERFORMANCE

We conducted thread parallel optimization of packing/unpacking for non-blocking MPI communication for halo exchange. Cost of packing operation is proportional to the number of CUBEs to pack. Packing operations divide each adjacent face (3–6 faces) with thread-level parallelization. Therefore, some cores are not utilized. Besides, if grid shape of CUBE in partition is not cubic, number of CUBEs to pack varies depending on the size of the adjacent face. In this study, each partition is composed of 4 × 4 × 2 CUBEs. Thus, size of adjacent face is 4 × 2 = 8 or 4 × 4 = 16 CUBEs. Therefore, it would also result in cost imbalance between threads.

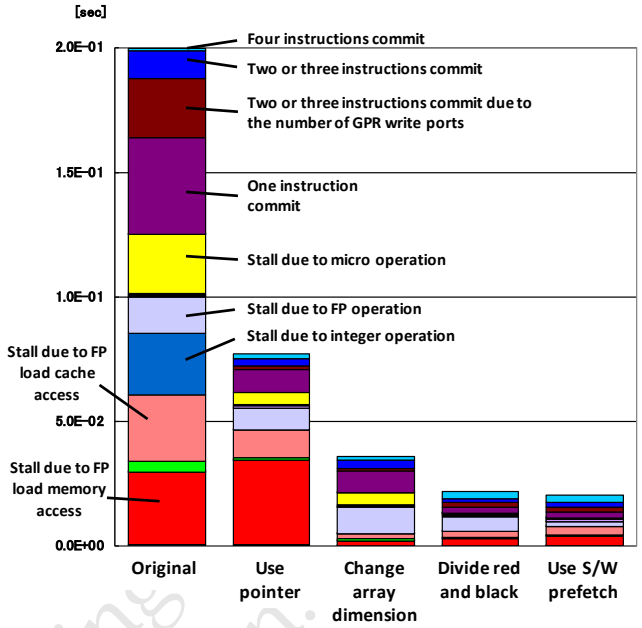

Figure 1: Execution time and its breakdown of Gauss-Seidel loop

To ameliorate this problem, we changed the thread parallelization loop for packing operation from loop of adjacent surface to loop of each cube. Besides, we conducted a thread level cost balancing in Conjugate Gradient method loop. Number of iterations of the thread parallelization loop is comparable with the number of threads, and the cost of each thread is statically analyzable. Therefore, we adopted "static" scheduling scheme after comparing with "dynamic" scheme.

We conducted a weak scaling test up to 8,192 nodes (65,536 cores) using original and several versions of the optimized code. With the most optimized code, we achieved 2.9x–3.9x speedup with 2–8,192 nodes in the time-step loop. (Figure.2)

5 CONCLUSION

We optimized multigrid Poisson solver which dominates 91% of execution time of the main time-stepping loop. With optimization of the computational kernel (Gauss-Seidel loop), we achieved a 9.8x speedup and peak ratio increased from 0.4% to 4.0%. With optimization of parallel performance, we achieved sustainable 2.9x–3.9x speedup up to 8,192 nodes in time-step loop. In case of using more number of nodes, scaling of CG method loop gets worse, therefore more MPI process level tuning is required.

ACKNOWLEDGMENTS

We greatly appreciate the advice and support of Dr. Jun Ikeda from the Kobe University and Dr. Akiyoshi Kuroda from the RIKEN Center for Comuptational Science (R-CCS). Part of the results is obtained by using the K computer at the RIKEN Center for Computational Science.

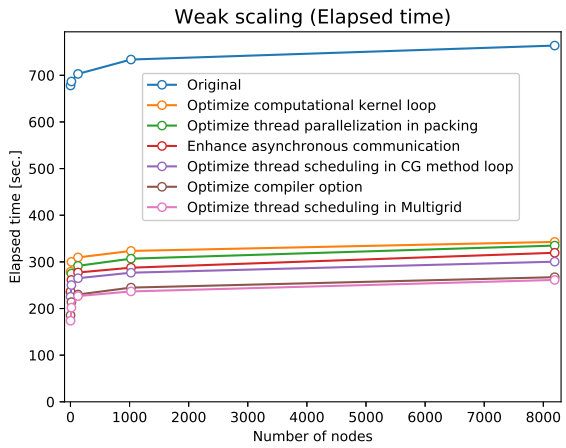

Figure 2: Weak scaling of time-step loop

REFERENCES

[1] Amneet Pal Singh Bhalla, Rahul Bale, Boyce E. Griffith, and Neelesh A. Patankar. 2013. A unified mathematical framework and an adaptive numerical method for fluid-structure interaction with rigid, deforming, and elastic bodies. *J. Comput. Phys.* 250, 1 (2013), 446–476. https://doi.org/10.1016/j.jcp.2013.04.033

[2] Niclas Jansson, Rahul Bale, Keiji Onishi, and Makoto Tsubokura. 2019. CUBE: A scalable framework for large-scale industrial simulations. *The International Journal of High Performance Computing Applications* 33, 4 (2019), 678–698. https://doi.org/10.1177/1094342018816377

[3] Kiyoshi Kumahata, Kazuo Minami, and Naoya Maruyama. 2016. High-performance conjugate gradient performance improvement on the K computer. *International Journal of High Performance Computing Applications* 30, 1 (2016), 55–70. https://doi.org/10.1177/1094342015607950

[4] Rajat Mittal and Gianluca Iaccarino. 2005. Immersed Boundary Methods. *Annual Review of Fluid Mechanics* 37, 1 (2005), 239–261. https://doi.org/10.1146/annurev.fluid.37.061903.175743

[5] Kazuhiro Nakahashi. 2003. Building-Cube Method for Flow Problems with Broadband Characteristic Length. *Computational Fluid Dynamics* 33, 4 (2003), 678–698. https://doi.org/10.1007/978-3-642-59334-5_7

[6] Keiji Onishi, Shigeru Obayashi, Kazuhiro Nakahashi, and Makoto Tsubokura. 2013. Use of the Immersed Boundary Method within the Building Cube Method and its Application to Real Vehicle CAD Data. In *21st AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, San Diego, CA. https://doi.org/10.2514/6.2013-2713

[7] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785