Performance Evaluation of a Next-Generation CFD on Various Supercomputing Systems

Kazuhiko Komatsu, Takashi Soga, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi

Abstract The Building-Cube Method (BCM) has been proposed as a new CFD method for an efficient three-dimensional flow simulation on large-scale supercomputing systems, and is based on equally-spaced Cartesian meshes. As a flow domain can be divided into equally-partitioned cells due to the equally-spaced meshes, the flow computations can be divided to partial computations of the same computational cost. To achieve a high sustained performance, architecture-aware implementations and optimizations considering characteristics of supercomputing systems are essential because there have been various types of supercomputing systems such as a scalar type, a vector type, and an accelerator type. This paper discusses the architecture-aware implementations and optimizations for various supercomputing systems such as an Intel Nehalem-EP cluster, an Intel Nehalem-EX cluster, Fujitsu FX-1, Hitachi SR16000 M1, NEC SX-9, and a GPU cluster, and analyses their sustained performance for BCM. The performance analysis shows that memory and network capabilities largely affect the performance of BCM rather than computational potentials.

K. Komatsu (✉) · R. Egawa · H. Kobayashi
Cyberscience Center, Tohoku University/JST CREST, 6–3 Aramaki-aza-aoba, Aoba, Sendai 980–8578, Japan
e-mail: komatsu@isc.tohoku.ac.jp; egawa@isc.tohoku.ac.jp; koba@isc.tohoku.ac.jp

T. Soga
NEC System Technologies, Ltd., Osaka 540–8551, Japan
e-mail: soga-txa@necst.nec.co.jp

H. Takizawa
Graduate School of Information Sciences, Tohoku University/JST CREST, 6-6-01 Aramaki-aza-aoba, Aoba, Sendai 980–8579, Japan
e-mail: tacky@isc.tohoku.ac.jp

M. Resch et al. (eds.), *Sustained Simulation Performance 2012*,
DOI 10.1007/978-3-642-32454-3_11,

1 Introduction

Since 1960s, the numerical calculation using computers has been utilized for simulations and analysis of fluid dynamics. In CFD, unstructured mesh and boundary-fitted mesh have generally been utilized to represent complicated geometries such as a three-dimensional whole airplane. These mesh methods have advantages of the mesh quality and the accuracy of the simulations. However, CFD algorithms with these mesh methods become complicated, and need a high computational cost. In addition, because the mesh is not regular, it is difficult to realize a spatial higher-order flow solver.

In order to solve these problems, the *Building-Cube Method (BCM)* has been proposed to efficiently simulate various fluids [4, 7, 8]. BCM uses equally-spaced Cartesian meshes. One of the advantages of BCM is that the algorithms of pre-processing, post-processing, and even the flow solver can be simplified [2] because of the equally-spaced Cartesian meshes. Another advantage is that it is well suited for highly parallel computation because equally-spaced meshes produce many parallel tasks with the same amount of computation.

Along with the development of CFD, the performance of supercomputing systems has also drastically been improved because of the rapid advancement of semiconductor technologies. To achieve a high sustained performance using supercomputing systems, architecture-aware implementation and optimization considering characteristics of supercomputing systems are essential due to various types of supercomputing systems such as a scalar type, a vector type, an accelerator type.

This paper describes architecture-aware implementations and optimizations of BCM on an Intel Nehalem-EP cluster, an Intel Nehalem-EX cluster, Fujitsu FX-1, Hitachi SR16000 M1, NEC SX-9, and a GPU cluster to examine the implication of their architectural features with BCM. From experimental results, this paper analyses the performances and scalabilities of BCM.

2 Overview of the Building Cube Method

BCM is designed for three-dimensional large-scale flow computations around practical geometries using high-density grids [4]. The basic idea of BCM is to decompose a whole flow domain into sub-domains called *cubes*, and further decompose each cube into high-density and equally-spaced Cartesian meshes called *cells* shown in Fig. 1. The size of each cube is determined by geometries and flow features at its location [2].

One of the advantages of BCM is that the algorithm is simple because it does not deal with complicated mesh structures. Thus, the simplicity of pre-processing, flow solvers, and post-processing lead to fast computation.

Another advantage of BCM is that the calculations of cubes can easily be decomposed into many data parallel tasks of the same size because the calculations

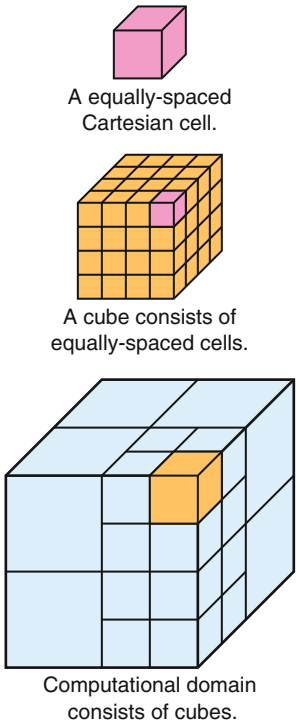

Fig. 1 Computational mesh in BCM

are independent each other. In addition, each cube has the same computational cost and data size for the calculations. Although there are data dependencies among adjacent cells in a cube, the computational cost per cell is also the same. More massive data parallelism in BCM might be obtained if the dependencies are eliminated.

The flowchart of the BCM incompressible flow solver is shown in Fig. 2 [7, 8]. The governing equations are incompressible Navier–Stokes equations. The fractional-step method [1, 3, 6] is used with the finite difference scheme on the staggered arrangement. In the fractional-step method, the solver can be classified into three major stages in one time step; a solver stage for calculating a temporal velocity field, a solver stage for calculating a pressure field, and a solver stage for calculating a real velocity field. In each stage, calculations of its field and data exchanges between cubes are included. The most dominant part in these stages is the calculation of the pressure field by solving the Poisson equation using the SOR method.

To calculate the pressure of one cell, a seven-point stencil calculation, which requires the pressure data of a cell and its six adjacent cells, is performed. As the stencil calculations for all cells in all cubes are repeated until the difference of the calculated field is convergent, the calculations of the pressure field dominates the most of time for the whole BCM calculations.

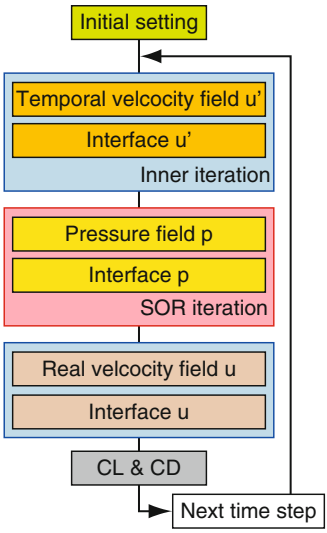

Fig. 2 Flowchart of the BCM flow solver

The pressure calculations for cubes can be performed in parallel because they are independent. In addition, the computational cost of each cube is completely the same. Therefore, parallel computing is adequate to accelerate the pressure calculations.

3 Implementation of BCM on Various Systems

To achieve significant acceleration by parallel processing on supercomputing systems, architecture-aware implementations and optimizations considering characteristics of supercomputers are essential.

This section describes the overview of the target supercomputing systems. The specifications of processors used in the systems are shown in Table 1. Then, the architecture-aware implementations and optimizations of BCM for these supercomputers are described as follows.

3.1 Implementation on Scalar Systems

The Nehalem-EP cluster, the Nehalem-EX cluster, FX-1, and SR16000 M1 are scalar parallel supercomputers that equip Nehalem-EP, Nehalem-EX, SPARC64VII, and Power7 processors, respectively. As shown in Table 1, these scalar processors also have large on-chip cache memories. On-chip L2 and/or L3 caches should be used for data with high locality to avoid redundant memory accesses. Moreover, uses of SIMD instructions are essential to efficiently process multiple data.

Table 1 Specifications of a processor in the supercomputing systems

System	GFlops/s	Mem.BW (GB/s)	# of Cores	On-chip memory	B/F
Nehalem EP	46.93	25.6	4	256 KB L2/core, 8 MB shared L3	0.55
Nehalem EX	74.48	34.1	8	256 KB L2/core, 24 MB shared L3	0.47
SPARC64VII	40.32	40.0	4	6 MB shared L2	1.0
Power 7	245.1	128	8	256 KB L2/core, 32 MB shared L3	0.52
SX-9	102.4	256	1	256 KB ADB	2.5
Tesla C1060	78	102	1	16 KB/SM	1.3

Although the Red–Black method can eliminate the data dependency, the stride memory accesses are required, resulting in performance degradation. Even though dividing an array into two arrays, non-unit-stride accesses are required. Thus, in the implementation on the scalar systems, the original SOR method is adopted to avoid degrading the performance by the stride memory accesses.

In the implementation of BCM on a scalar system, cubes are hierarchically assigned to nodes and then to processors in a node. As the computational cost of each cube is also the same, efficient parallel processing using a number of scalar processors can be carried out.

3.2 Implementation on a Vector System

SX-9 is a vector parallel supercomputer consisting of a large Symmetric Multi-Processing (SMP) nodes, each of which has 16 102.4Gflop/s-vector processors. In the implementation on SX-9, the Red–Black SOR method using mask tables is used. The Red–Black method can avoid indirect memory accesses and exploit the data parallelism among cells by removing the dependencies among cells. As parallelizing the SOR method generally shortens the length of loop, the mask tables can avoid accessing unnecessary data without shortening the length of a loop. Thus, the loop remains long enough to utilize all of the vector units of SX-9.

The effective use of an on-chip 256KB software-controllable cache named Assignable Data Buffer (ADB) in SX-9 is also a key to exploit the potential of SX-9. Once data specified by programmers are accessed, these data are stored in ADB and can be used in the next accesses. Thus, the ON_ADB directives are inserted to the source code to specify reusable data in the seven-point stencil calculations. As a result, the reusable data are kept in ADB at runtime. As the main memory and ADB can simultaneously provide data to vector pipelines, the vector processor can access those data at a high sustained bandwidth, and thereby achieve a high performance.

3.3 Implementation on a GPU System

A GPU cluster consists of multiple nodes, each of which has a CPU with main memory and one or more GPUs. Each GPU can be considered as a many-core processor consisting of hundreds of *stream processors (SPs)* in *CUDA (Compute Unified Device Architecture)* [5]. In CUDA, SPs are grouped into *stream multiprocessors (SMs)*, and several SPs in an SM work together.

The memory system of the CUDA platform is hierarchical. The *shared memory* is an on-chip memory space shared by threads. The capacity of the shared memory is small, but the memory access latency is very short. On the other hand, the *global memory* is the largest off-chip memory, but it needs a long access latency. Thus, it is necessary to use both shared memory and global memory appropriately. Key techniques for efficient data transfers are to make good use of the shared memory and to use coalesced global memory accesses as much as possible.

Massive parallelism of BCM is well suited for load balancing among nodes and also among many cores in each GPU. In the implementation of BCM on a GPU cluster system, after grouping cubes into subsets, each subset of cubes is assigned to one of GPU nodes as shown in Fig. 3. The cubes in a subset are further assigned to SMs of GPUs. The computations for the cells in cubes are assigned into threads, which are executed on SPs. As the computational cost is the same, efficient parallel processing using multiple nodes can be expected.

Besides, the effective use of the shared memory in a GPU is essential to reduce the number of global memory accesses requiring a long access latency. By storing data with high locality to a ring buffer on the shared memory as shown in Fig. 4, it can not only reduce the number of the off-chip memory accesses but also effectively utilize the limited capacity of the on-chip memory.

4 Performance Evaluation and Discussions

The flow simulations using BCM around 3D test models are performed on the supercomputing systems shown in Table 1. F1 is a large model of 200 million cells, and Sphere is a small model of 5 million cells which are shown in Fig. 5.

Figure 6 shows the sustained performance of BCM achieved for the F1 model. The results show that SX-9 achieves a higher sustained performance than the others. As BCM is a memory-intensive application, the sustained memory bandwidth has a great impact on the performance. In addition to the importance of a high memory bandwidth, the effective use of ADB further improves the sustained bytes/flop ratio, resulting in the high performance.

Even though the peak memory bandwidth of FX-1 outperforms those of Nehalem EP and EX, its sustained performance is lower. This is because the sustained memory bandwidth of Nehalem is higher than that of SPARC64VII. In the STREAM benchmark, FX1 achieves only 10.0 GB/s while Nehalem EP and EX achieve 17.0 and 17.6 GB/s, respectively.

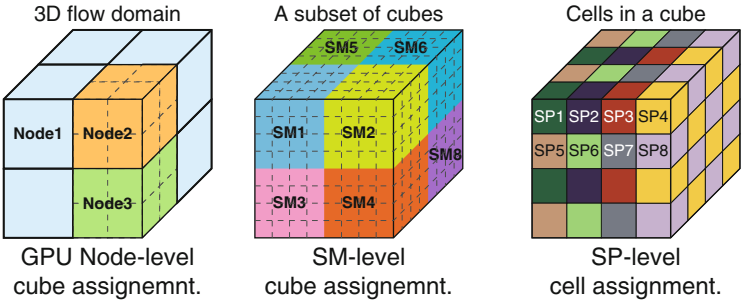

Fig. 3 Task assignments into a GPU cluster system

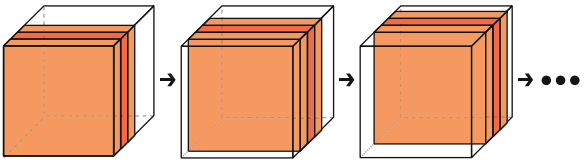

Fig. 4 Three planes for cyclical use of the shared memory

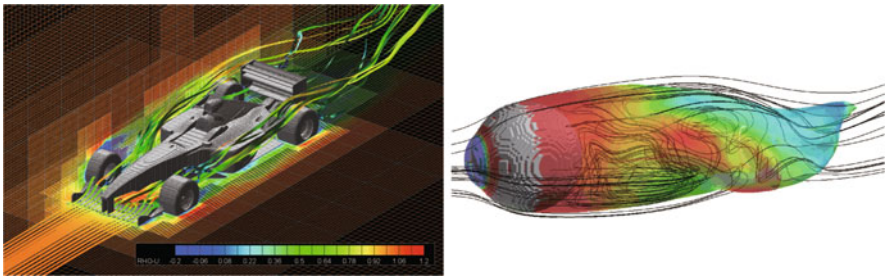

Fig. 5 3D test modes. (*Left*: F1, *Right*: Sphere)

Figure 7 shows the sustained performance of BCM on the Sphere model, which includes the results of the GPU system. This results shows that a GPU cluster system achieves comparable and/or better performance than the scalar cluster systems. The main reason is that GPUs can accelerate the data parallel calculations of BCM using a number of SPs and high memory bandwidth, even though it cannot execute a large problem such as the F1 model due to the limited global memory capacity. Effective use of SPs and shared memory contributes to the good sustained performance larger than the other scalar systems. Another reason is that the sustained performances of other supercomputers including SX-9 become lower for a small problem such as the Sphere model. However, even if the calculations using the GPUs are fast, data transfers between a GPU and a CPU in a node and between GPUs in different nodes

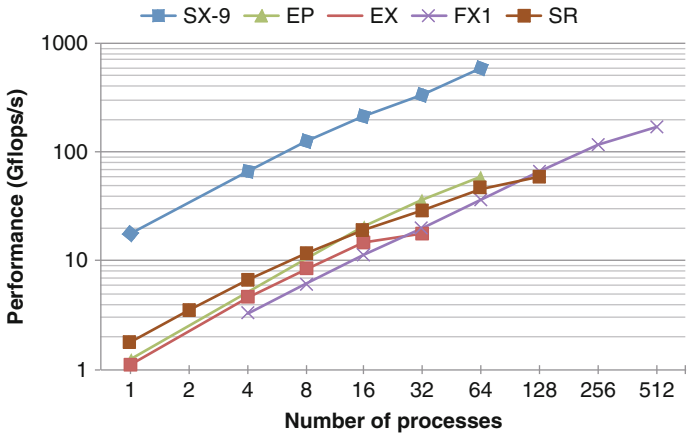

Fig. 6 Sustained performance of the F1 model

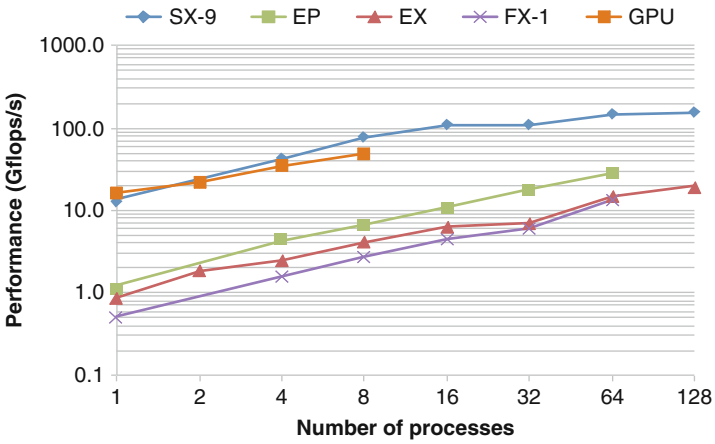

Fig. 7 Sustained performance of the sphere model

are slow and cannot be negligible. As a result, the data transfer dominates the most of time in the simulation. To further accelerate BCM using the GPU system, the time of data transfers should be shortened and be hidden by transferring data during the calculations.

The ratio of the sustained performance to the peak performance on an SX-9 vector processor is about 17 %, while those of the other systems are about 1.5–3.2 %. This is because the vector units in a vector processor are efficiently utilized for the calculations.

Taking a look at the scalability shown in Fig. 8, all of the systems achieve high scalability in the F1 model due to a large number of parallel tasks and sufficient

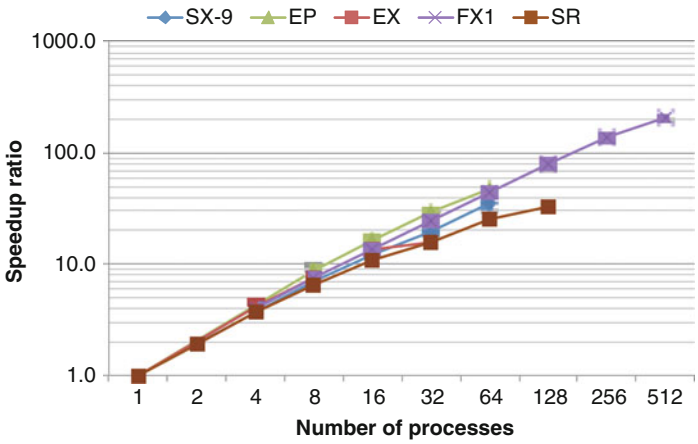

Fig. 8 Speedup ratio of the F1 model

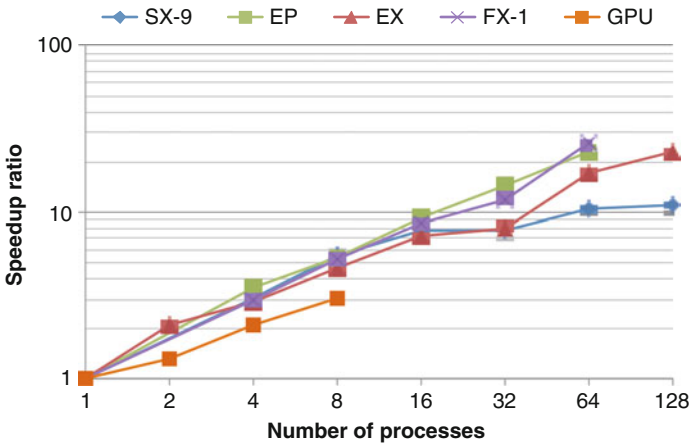

Fig. 9 Speedup ratio of the sphere model

network bandwidth. The scalabilities in the Sphere model shown in Fig. 9 are lower than those of the F1 model. The low scalability of the GPU system comes from the overhead of data transfers. The lower scalabilities of the other systems come from the lack of parallel tasks due to a small number of cubes in the Sphere model.

5 Concluding Remarks

This paper describes the implementations and optimizations of BCM for various types of supercomputing systems such as a scalar type, a vector type,

and an accelerator type. The implementation and optimizations considering the characteristics of both a supercomputing system and an application are necessary for high sustained performance. From the performance evaluations of BCM on SX-9, a Nehalem-EP cluster, a Nehalem-EX cluster, FX-1, Hitachi SR16000 M1, and a GPU cluster, it is clarified that the memory bandwidth and the network bandwidth greatly affect the sustained performance of BCM. Therefore, the supercomputing systems that can achieve a high-sustained memory bandwidth and network bandwidth, such as SX-9, are the most promising for further acceleration of BCM.

Acknowledgements The author would like to thank Dr. Kazuhiro Nakahashi of JAXA, Lecturer Daisuke Sasaki of Kanazawa Institute of Technology, Assistant Professor Shun Takahashi of Tokyo University Agriculture and Technology, and Dr. Akihiro Musa of NEC cooperation for valuable discussions on this research. This research was partially supported by Grant-in- Aid for Scientific Research (S) #21226018; Grant-in- Aid for Young Scientists (B) #23700028; Core Research of Evolutional Science and Technology of Japan Science and Technology Agency (JST CREST).

References

1. Dukowicz, J.K.: Approximate factorization as a high order splitting for the implicit incompressible flow equations. Journal of Computational Physics **102**, 336–347 (1992)
2. Ishida, T., Takahashi, S., Nakahashi, K.: Efficient and robust cartesian mesh generation for building-cube method. Journal of Computational Science and Technology **2**(4), 435–445 (2008)
3. Kim, J., Moin, P.: Application of a fractional-step method to incompressible navier-stokes equation. Journal of Computational Physics **59**, 308–323 (1985)
4. Nakahashi, K.: High-density mesh flow computations with pre-/post-data compressions. In: AIAA paper, pp. 2005–4876 (2005)
5. NVIDIA Corporation: NVIDIA CUDA Compute Unified Device Architecture. http://developer.nvidia.com/category/zone/cuda-zone
6. Perot, J.B.: An analysis of the fractional step method. Journal of Computational Physics **108**, 1–58 (1993)
7. Takahashi, S., Ishida, T., Nakahashi, K., Kobayashi, H., Okabe, K., Shimomura, Y., Soga, T., Musa, A.: Study of high resolution incompressible flow simulation based on cartesian mesh. In: AIAA paper: 47th AIAA Aerospace Sciences Meeting, pp. 2009–563 (2009)
8. Takashi, S.: Study of large scale simulation for unsteady flows. Ph.D. thesis, Tohoku University (2009)