I/O Performance Evaluation of a Memory-Saving DNS Code on SX-Aurora TSUBASA

Yokokawa, Mitsuo ; Yamane, Yuki ; Yamaguchi, Kenta ; Soga, Takashi ; Matsumoto, Taiki ; Musa, Akihiro ; Komatsu, Kazuhiko ; Ishihara,…

(Citation)
2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW):692-696

(Issue Date)
2023-05

(Resource Type)
conference paper

(Version)
Accepted Manuscript

(URL)
https://hdl.handle.net/20.500.14094/0100489454

I/O Performance Evaluation of a Memory-Saving DNS Code on SX-Aurora TSUBASA

Mitsuo Yokokawa
Graduate School of System Informatics
Kobe University
Kobe, Japan
yokokawa@port.kobe-u.ac.jp

Yuki Yamane, Kenta Yamaguchi, Takashi Soga
NEC Solution Innovators, Ltd.
Koto-ku, Tokyo, Japan
{*yamane.yuki, yamaguchi-zx, soga_takashi*}*@nec.com*

Taiki Matsumoto
Graduate School of System Informatics
Kobe University
Kobe, Japan
t-matsumoto@stu.kobe-u.ac.jp

Akihiro Musa, Kazuhiko Komatsu
Cyberscience Center
Tohoku University
Sendai, Japan
{*musa, komatsu*}*@tohoku.ac.jp*

Takashi Ishihara
Graduate School of Environmental and Life Science
Okayama University
Okayama, Japan
takashi_ishihara@okayama-u.ac.jp

Hiroaki Kobayashi
Graduate School of Information Sciences
Tohoku University
Sendai, Japan
koba@tohoku.ac.jp

***Abstract*—Direct numerical simulation (DNS) by spectral methods is widely used to reveal turbulence properties because of its high accuracy. DNS with a larger number of grid points than simulations conducted to date, however, requires a large amount of computation time and memory capacity. Therefore, DNS with a large number of grid points is not realistic, even if state-of-the-art supercomputers are used. In this paper, in order to execute a DNS with a larger number of grid points, a memory-saving DNS code has been developed to solve the memory capacity constraint at the expense of increased CPU time. Variables in the code are subdivided and stored in multiple temporary files and are processed sequentially during the calculations. The DNS code has been carried out as an early performance evaluation on SX-Aurora TSUBASA equipped with synchronous and asynchronous input and output (I/O) functions. The results clarified that the synchronous I/O performance outperformed the asynchronous I/O performance. The shortest calculation time was achieved using the synchronous I/O to eight temporary files on solid state drives.**

***Keywords*-Direct numerical simulation; Turbulent flows; Memory-saving; SX-Aurora TSUBASA; Asynchronous I/O; Synchronous I/O**

I. Introduction

Recent rapid development in the capacities and capabilities of supercomputers has led to computational approaches becoming powerful tools in various science and engineering fields. In particular, direct numerical simulation (DNS) of turbulence by spectral methods can provide detailed turbulence data without experimental uncertainties under well-controlled conditions up to the maximum wavenumber at discretization. Since Orszag began to carry out DNS by spectral methods [1], many researchers have carried out DNSs for turbulent flows with higher Reynolds numbers [2], [3]. Although DNS with a larger number of grid points than previous DNSs computed to date is still expected, DNS that requires more memory than the system's memory capacity is extremely difficult to carry out.

Recent advances in secondary storage configurations equipped with hard disk drives (HDDs) or solid state drives (SSDs) make a secondary storage useful for conducting a DNS with a larger number of grid points by storing computed values to files in a DNS code [4]. Especially, because a file system configured with SSDs is directly connected to each compute node as a local file system and its access speed is faster than that of the second-tier storage system configured by a large-scale parallel file system like the Lustre file system, a local file system can be used as an extended memory space.

In this paper, a memory-saving DNS code has been developed to solve the memory capacity constraint at the expense of increased CPU time. In this code, variables that are distributed across compute nodes during parallel computations can be further partitioned into smaller portions in the node. Each variable subdivision is stored to a separate temporary file and used while reading and writing. To this end, this study has examined the input and output (I/O) performance on SX-Aurora TSUBASA using this code.

II. DNS Code Implementation

We consider homogeneous isotropic turbulence that obeys the incompressible Navier-Stokes equations and the continuum equation

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} = -\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{f}, \quad \text{and} \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{2}$$

in a periodic box of side length 2π. Here, $\boldsymbol{u} = (u_1, u_2, u_3)$, p, ν, and $\boldsymbol{f} = (f_1, f_2, f_3)$ denote the velocity, pressure, kinematic viscosity, and external force, respectively [5]. Because the boundary condition is periodic, the Fourier spectral method can be applied for discretization.

Then, equations (1) can be written as

$$\left(\frac{d}{dt} + \nu|\boldsymbol{k}|^2\right)\hat{u_i}(\boldsymbol{k}) = \left(\delta_{ij} - \frac{k_i k_j}{|\boldsymbol{k}|^2}\right)\hat{h_j}(\boldsymbol{k}) + \hat{f_i}(\boldsymbol{k}), \tag{3}$$

where Equation (2) is used to eliminate the pressure term. In equations (3), $\boldsymbol{k} = (k_1, k_2, k_3)$ is a wavenumber vector, $\hat{h_j}(\boldsymbol{k}) = ik_l\widehat{u_j u_l}$ is nonlinear term, and the hat symbol $\hat{}$ denotes Fourier coefficients. Ordinary differential equations (3) are integrated with respect to time by the fourth-order Runge-Kutta-Gill (RKG) method.

Fast Fourier transforms (FFT) can be efficiently used to calculate the nonlinear term $\hat{h_j}(\boldsymbol{k})$ that is expressed as convolution sums in the wave vector space. Aliasing errors generated by the spectral method are removed by the phase shift method and the cut-off beyond the maximum wavenumber $k_{\max} = \sqrt{2}N/3$, where N is the number of grid points in each of the Cartesian coordinates in the physical space.

The original parallel code has been developed by two-dimensional domain decomposition with MPI for the data distribution, and the three-dimensional FFT parallelized by pencil decomposition is used in the implementation. When targeting a DNS with $n^3 = 32,768^3$, our code requires approximately $23n^3$ double precision floating point variables in total, resulting in 5.6 petabytes. There is no supercomputer, however, that has this amount of memory capacity in Japan. Therefore, a memory-saving method is needed.

III. Memory-Saving Concept

Modern supercomputers provide large storage systems that can be used to save variables in the code. Especially, the first-tier storage, mostly consisting of SSDs, can be used as a temporary storage of variables in the code.

The idea to save the memory capacity declared in the code is that variables can be divided equally into smaller quantities, and each portion can be stored separately in a temporary file on the first-tier storage. The number of temporary files is the same as the number of portions. Figure 1 shows a schematic image of variable partitioning. First, for example, a wave vector of the x-direction velocity is partitioned by the pencil domain decomposition into the

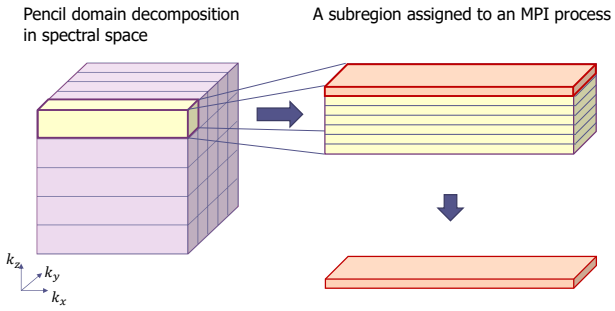

Figure 1. Variable partitioning in a compute node

number of parallel processes, as depicted on the left side of the figure. Each parallel process calculates its own assigned portion of the variables. In the memory-saving method, the portion assigned to each process is further divided equally into the number of temporary files, N_f, which are created on the first-tier storage. Then, the size of variables used in each process can decreases to $1/N_f$.

For computation, first, the program reads the first portion from the file, performs computations with the data in that portion, and then writes the updated values to the same file they are read from. After processing of the first portion, processing of the second portion is similarly performed. This procedure continues until all portions of the variables are processed. Figure 2 describes how to change the original code to a memory-saving code in a Fortran-like description. If the rank of the array variable is 3, the last rank is partitioned into the number of files at declaration, and a loop is added to repeatedly read each portion of the variables, perform calculations, and write updated values to the file.

```
real(kind=8) :: a(n,m,NN)
real(kind=8) :: b(n,m,NN)




do k = 1, NN
  do j = 1, m
    do i = 1, n
      a(i,j,k) = ... b(i,j,k)
    end do
  end d0
end do
```

```
real(kind=8) :: a(n,m,NN/nf)
real(kind=8) :: b(n,m,NN/nf)

do F = 1, nf

  read a(:,:,:), b(:,:,:) from the #F file.

  do k = 1, NN/nf
    do j = 1, m
      do i = 1, n
        a(i,j,k)=... b(i,j,k)
      end do
    end do
  end do

  write a(:,:,:) to the #F file.

end do
```

Figure 2. Pseudo kernel of the memory-saving code

Although the calculation time is increased by a greater amount of input and output (I/O) time, the memory capacity required to execute the code can be decreased significantly by this implementation.

IV. Asynchronous Execution of the Method on SX-Aurora TSUBASA

SX-Aurora TSUBASA is a vector-type supercomputer equipped with distinctive architectures and software [6]. The system consists primarily of a vector host (VH) and one or more vector engines (VEs). The VH is a standard x86 server with one or more processors running a standard Linux operating system. A VE is a vector accelerator implemented as a peripheral component interconnect express (PCIe) card, on which a vector processor is mounted with six high-bandwidth memory (HBM2) modules. The vector processor consists of eight vector cores, a 16 MB last-level-cache (LLC), and a VE direct memory access (DMA) engine. The LLC is connected to each core through a two-dimensional intra-network with a total cache bandwidth of 3.0 TB/s.

Operating system functions have been realized on the VH to the greatest extent possible so that the VEs are able to effectively use their computational power by concentrating on program executions with vector instructions. Because the GNU C library (*glibc*) is ported to the VEs, applications can call *glibc* functions such as normal I/O functions. Vectorization should also be applied to achieve high computation performance. The performance of SX-Aurora TSUBASA has been evaluated as excellent [7].

The SX-Aurora TSUBASA system provides three types of I/O functions namely synchronous normal I/O, synchronous accelerated I/O, and asynchronous I/O functions. The I/O performance using the synchronously normal and accelerated I/O functions has been studied in [8]. The synchronously accelerated I/O showed good performance for I/O in the program. In addition to these two synchronous I/O functions, an asynchronous I/O for VE (AIO) is also provided. The AIO enables VE programs to do their tasks during I/O system call operation, and does not need any VE resource while the I/O operation on VH is in progress [9]. When an asynchronous function is implemented in the code, the computation and I/O operation can be overlapped to reduce overall wall-clock time. A benchmark program has been developed to check the applicability of the AIO feature.

The pseudo code in a Fortran-like description is shown in Fig. 3. The 4th rank of the size of 3 is added to the original three-dimensional array, resulting in a four-dimensional array. The 4th rank is used for cyclic read, computation, and write operations. Although special treatment is necessary at the beginning and end of the loop, read, computation, and write operations can be pipelined by accessing different portions of the original array, each of which is stored in the different temporary file. The AIO feature overlaps the computing region (shaded pink in Fig. 3) and the I/O operation region (shaded blue).

In the benchmark, the case that the second rank of a two-dimensional array `u(n,n)` was divided by eight, the number of temporary files, is considered. An array `u(n,n/8,0:2)` was declared and it was used to test the AIO feature. Computation part is an loop in which each element of an array is incremented by one. The size `n` is taken as 30,720 and the loop length of the computation part was determined so that the computation time is greater than the I/O time.

Figure 4 shows the wall-clock time of three I/O functions when the computation time is greater than the I/O time. The computation time for the synchronously normal I/O of the leftmost bar in the figure is the longest computation time. The synchronously accelerated I/O can shorten the I/O time by approximately 1/4, as shown in the middle bar. For the AIO, the I/O time is completely hidden in the computation time. This experiment shows that the AIO feature efficiently shortens the wall-clock time. The next section describes how the synchronously accelerated I/O and AIO functions are applied to the DNS code.

V. Evaluation of I/O Functions in the Memory-Saving DNS Code

We applied the memory-saving method to the DNS code and carried out time measurement of the code execution on the SX-Aurora TSUBASA called "AOBA-A" installed at Tohoku University. Two secondary storage systems were used for the measurements: one is a ScaTeFS parallel file system composed of HDDs developed by NEC Corporation and the other is a local file system composed of SSDs directly connected to each VH.

The number of grids and the number of MPI processes are set as $n^3 = 512^3$ and 8×8 of the pencil decomposition, respectively, in the measurements. The size of an array variable on each MPI process after parallelization, e.g. a wave vector of x-directional velocity, is $512^3 \times$

```
real(kind=8) :: a(n,m,NN/nf,0:2)
real(kind=8) :: b(n,m,NN/nf,0:2)

do F = 1, nf
  nr = mod(F-1,3)
  nc = mod(F+1,3)
  nw = mod(F  ,3)

! Computation region for the data in the #F file.
  do k = 1, NN/nf
    do j = 1, m
      do i = 1, n
        a(i,j,k,nc) = ... b(i,j,k,nc)
      end do
    end do
  end do

! I/O operation region
  read  a(:,:,:,nr), b(:,:,:,nr) from the #(F+1) file.
  write a(:,:,:,nw) to the No. #(F-1) file.

end do
```

Figure 3. Pseudo code using the asynchronous I/O feature

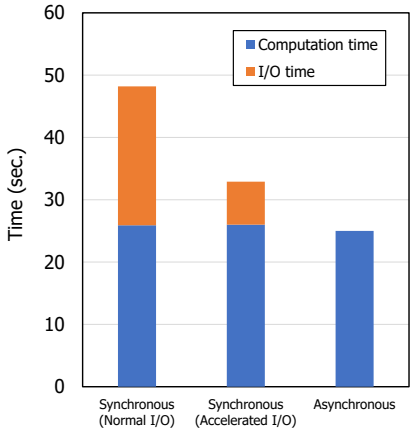

Figure 4. Computation time using three types of I/O functions

8 (bytes)/64 (processes) = 16 megabytes. 16 arrays in each MPI process are partitioned by the number of files that are taken as 2, 4, 8, 16, and 32. Therefore, the sizes of each file are 4096, 2048, 1024, 512, and 256 kilobytes. The code is executed on the 64 VEs of 8 VHs and with 64 MPI processes. In each process, 8 threads were used for multi-thread parallel computation. The computation result was confirmed to be the same as the result with the original code.

Figure 5 shows the computation time of one RKG time step for the measurement cases. The leftmost bar graph is the time without the memory-saving method, that is, the time taken by the original code. The four groups of the five bars shown on the right part of the graph represent the computation time using synchronous I/O to the second-tier file systems configured on HDDs, AIO to the second-tier files on HDDs, synchronous I/O to the local file system on SSDs, and AIO to the local file system on SSDs. The breakdown of the code indicates its measured time: blue is the nonlinear terms computation time, orange is computation time other than non-linear terms, grey is the I/O time in the non-linear terms calculation part, yellow is the I/O time of other than in the non-linear terms calculation part, and light blue is the other processing time.

Clearly, the original code that can be executed on memory without the memory-saving method takes the shortest time to execute. Unless the amount of memory used by the code exceeds the amount of memory capacity of the system, no memory-saving method is necessary. In the case that the DNS code exceeds the system memory capacity, it is necessary to evaluate the performance of the memory-saving method.

It is shown that the method with the AIO is slower than the methods with synchronous I/O. This is because the I/O time cannot be hidden due to the computation time being shorter than the I/O time. In addition, the overhead of using the AIO features on SX-Aurora TSUBASA is quite larger than expected. The behavior of the AIO feature of SX-Aurora TSUBASA needs to be investigated more precisely.

Because the parallel file system is shared by all users, it was found that the time needed to use the synchronous I/O feature with the parallel file system fluctuated during the measurement. We also confirmed that the I/O time for files on SSDs is shorter than that on HDDs. Eight subdivisions were the best case for the memory-saving method with the synchronous I/O from and to the local file system.

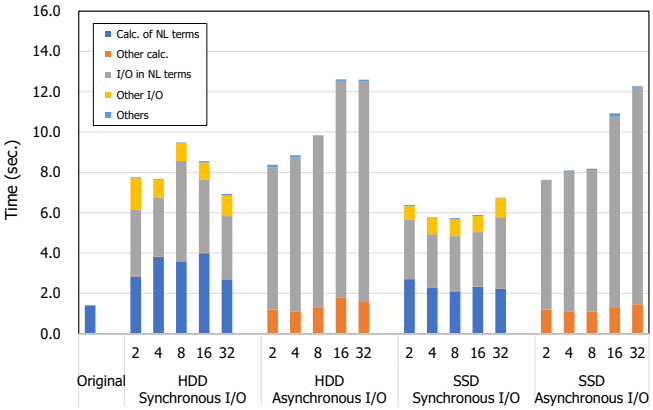

Figure 5. Computation time

VI. Conclusion

In this study, we have developed a memory-saving DNS code and evaluated the performance of the code on SX-Aurora TSUBASA. We found that, if the calculation time is longer than the I/O time, the file I/O time can be overlapped the calculation part of the code, and the I/O time can be hidden behind the calculation time, as described above. In experiments with the developed DNS code by a memory-saving method, the calculation time with the synchronously accelerated I/O feature is shorter than that with the AIO feature. The AIO feature should be tested in more depth in the future. Because the memory-saving code is portable, we plan to carry out it on other supercomputers like FUGAKU in the future.

Acknowledgments

This study was partly supported by JSPS KAKENHI (Grant Numbers JP18K11325 and 20H01948) and the Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (JHPCN) in Japan (Project IDs jh190068, jh200021, jh210034, and jh220027). This work was also supported partially by the Japan Ministry of Education, Culture, Sports, Science and Technology, Next Generation High-Performance Computing Infrastructure and Applications R&D Program, entitled “R&D of a Quantum-Annealing-Assisted Next Generation HPC Infrastructure and Its Applications.”

REFERENCES

[1] S. A. Orszag, "Numerical methods for the simulation of turbulence," *The Physics of Fluids*, vol. 12, no. 12, pp. II–250–II–257, 1969.

[2] T. Ishihara *et al.*, "Second-order velocity structure functions in direct numerical simulations of turbulence with R_λ up to 2250," *Phys. Rev. Fluids*, vol. 5, p. 104608, Oct 2020.

[3] K. Ravikumar *et al.*, "GPU acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism," in *SC'19: International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, U.S.A., 2019.

[4] G. Karniadakis and S. Orszag, "Nodes, modes, and flow codes," *Phys. Today*, vol. 46, no. 3, pp. 34–42, 2008.

[5] T. Ishihara *et al.*, "Small-scale statistics in high-resolution direct numerical simulation of turbulence: Reynolds number dependence of one-point velocity gradient statistics," *J. Fluid Mech.*, vol. 592, pp. 335–366, 2007.

[6] K. Komatsu *et al.*, "Performance and Power Analysis of a Vector Computing System," *Supercomputing Frontiers and Innovations*, vol. 8, no. 2, pp. 75–94, 2021.

[7] ——, "Performance Evaluation of a Vector Supercomputer SX-Aurora TSUBASA," in *SC'18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, U.S.A., 2018, pp. 685–696.

[8] M. Yokokawa *et al.*, "I/O performance of the SX-Aurora TSUBASA," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 27–35.

[9] NEC Corporation, "The Tutorial and API Reference of libsysve 2.11.0," 2018. [Online]. Available: https://sxauroratsubasa.sakura.ne.jp/documents/veos/en/libsysve/md_doc_VEAIO.html