Automatic Generation of High-Order Finite-Difference Code with Temporal Blocking For Extreme-Scale Many-Core Systems

Hideyuki Tanaka*, Youhei Ishihara†, Ryo Sakamoto§, Takashi Nakamura§, Yasuyuki Kimura*, Keigo Nitadori‡, Miyuki Tsubouchi‡, Jun Makino¶
* ExaScaler Inc.
2-1 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Email: tanaka.hideyuki@gmail.com
†Yukawa Institute for Theoretical Physics, Kyoto University
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto, 606-8502, Japan
Email: youhei.ishihara@yukawa.kyoto-u.ac.jp
‡RIKEN R-CCS
7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan
§ PEZY Computing K. K.
1-11 Ogawa-machi, Chiyoda-ku, Tokyo, 101-0052, Japan
Email: yasuyuki@pezy.co.jp
¶ Kobe University
1-1, Rokkodai-cho, Nada-ku, Kobe, 657-8501, Japan
Email: jmakino@riken.jp

Abstract—**In this paper we describe the basic idea, implementation and achieved performance of our DSL for stencil computation, Formura, on systems based on PEZY-SC2 many-core processor. Formura generates, from high-level description of the differential equation and simple description of finite-difference stencil, the entire simulation code with MPI parallelization with overlapped communication and calculation, advanced temporal blocking and parallelization for many-core processors. Achieved performance is 4.78 PF, or 21.5% of the theoretical peak performance for an explicit scheme for compressive CFD, with the accuracy of fourth-order in space and third-order in time. For a slightly modified implementation of the same scheme, efficiency was slightly lower (17.5%) but actual calculation time per one timestep was faster by 25%. Temporal blocking improved the performance by up to 70%. Even though the B/F number of PEZY-SC2 is low, around 0.02, we have achieved the efficiency comparable to those of highly optimized CFD codes on machines with much higher memory bandwidth such as K computer. We have demonstrated that automatic generation of the code with temporal blocking is a quite effective way to make use of very large-scale machines with low memory bandwidth for large-scale CFD calculations.**

I. Introduction

Explicit stencil computation is an important class of numerical methods used in a variety of HPC applications, for example computational fluid dynamics, FDTD method applied to many applications for wave propagation in electromagnetic, acoustics, seismic and other calculations.

In several important areas of computational fluid dynamics, implicit methods have been used for incompressive fluid, because they can overcome the CFL limit for the timestep due to the speed of information propagation. By using the combination of approximations such as incompressive approximation and implicit methods, one can use the timestep not limited by the CFL condition.

For example, Yang *et al.*[1], the Gordon-Bell Prize winner for 2016, used implicit method for non-hydrostatic atmospheric dynamics. They have achieved the efficiency of 7% on Sunway TaihuLight. Number of floating point operations per timestep per grid point is 9.3×10^4, which is 20-40 times larger than those of explicit methods (see table 1). The reason why the operation count is very high is that implicit methods require iterations to solve the Poisson equation for pressure at each timestep. They have developed highly sophisticated hybrid DD-MG method. Even so, the number of iterations necessary almost doubles when the number of grid points is increased by a factor of eight.

Recently, the reduced speed of sound method (RSS method) [2], [3] made it possible to increase the size of the timestep for explicit methods over the physical CFL limit. The basic idea of RSS is to apply the incompressive approximation to the opposite direction. We know that we can approximate the subsonic flow with different Mach numbers, for example $M = 0.01$, $M = 0.1$ and even $M = 0.5$ with the same incompressive flow. Therefore, we can actually approximate the flow with $M = 0.01$ with that with $M = 0.5$, thereby relaxing the CFL condition by a factor of 50.

The number of floating point operations per timestep per grid point is orders of magnitudes smaller for explicit schemes compared to implicit schemes. In addition, the use of the

RSS method is highly advantageous on modern HPC systems, because of the following three reasons:

- The requirements for communication bandwidth and latency are much less severe for explicit schemes than for implicit schemes. Explicit schemes require communication only with neighboring nodes. In the case of implicit methods, when the multigrid method is used, a large number of communications with faraway nodes are required per timestep, resulting in very large communication overhead.
- The requirement for memory bandwidth is also much less severe for explicit schemes than for implicit schemes. With proper cache blocking, an explicit method need to read and write physical variables only once per timestep, while in the case of an implicit method the main memory is read and written at each iteration within one timestep.
- When the RSS method is used, there is no essential difference between the timestep limits for implicit and explicit methods.

Note that the idea similar to that of the RSS method can be applied to many other problems where the implicit method have been used, such as mantle convection [4] and radiative transfer [5].

Therefore, we believe that it is quite important to develop an efficient way to implement explicit schemes, ideally combined with the temporal blocking, in order to use modern large HPC systems effectively.

The basic idea of temporal blocking is to apply multiple timesteps to local small block of grid that fit to the cache, so that we can reduce the access of the main memory. Theoretically, the upper bound of the performance $F_{\max}$ of an explicit scheme without temporal blocking is given by

$$F_{\max} = \min\left(F, \frac{C_e G}{2H_e}\right), \tag{1}$$

where C_e and H_e are the amounts of the computation and the data transactions per grid point, F and G are the computation throughput (flops) and the main memory bandwidth, in terms of operations per seconds and words per seconds.

With the temporal blocking method, this limitation is relaxed to [6]

$$F_{\max,\mathrm{TB}} = \min\left[F, \frac{C_e G}{2H_e}\left(\frac{1}{N_F} + \frac{2dN_s}{N_T}\right)^{-1}\right], \tag{2}$$

where N_F, N_T and N_s are the number of timesteps combined in temporal blocking, the size of grid in one dimension which fit into the last-level cache, and the width of the stencil, respectively, and d is the number of spatial dimensions.

We can see that an important parameter which determines the efficiency of the explicit method with the temporal blocking is $2dN_s/N_T$. The effect of the temporal blocking is larger for smaller N_s and larger N_T. The physical size of LLC determines N_T. In order to take advantage of temporal blocking, therefore, we should make N_s as small as possible. In the case of simple three-point stencil in one dimension, $N_s = 1$. If we use 4-point, staggered grid used in many FDTD applications, $N_s = 3$. We also have to take into account the effect of the time integration method. With the Runge-Kutta methods which are widely used in CFD applications, the effective size of the stencil becomes $N_s s$, where s is the number of stages of the Runge-Kutta method. Thus, if we use a $N_s = 3$ stencil with $s = 4$ Runge-Kutta method, we effectively have the stencil width of 12, which means the advantage of the temporal blocking would be completely lost for the cache of the size of several MBs.

Therefore, in order to reduce the memory access by temporal blocking, it is very important to use a scheme with small N_s and s. However, not much research has been done in this direction, since the need for such numerical schemes has not been recognized. Thus, even though the temporal blocking shows clear advantage for demonstrations with 7-point stencil for diffusion equation, its advantage in real applications is yet to be demonstrated.

Table I summarizes the state of the art for implicit and explicit solvers for large-scale CFD simulations of subsonic flows in earth science and astrophysics.

Hotta *et al.* used a four-point, staggered-grid spatial difference with 4-stage Runge-Kutta scheme. Therefore, the potential gain by using the temporal blocking is small. In the next section, we briefly describe the new finite difference scheme we have developed.

In the case of two calculations on K computer [7], [8], the number of floating-point operations per timestep is around 2000, and the number of basic physical variables is five. From equation (1), we can express the upper bound of the efficiency, defined as the computational speed determined by the memory bandwidth normalized by the theoretical peak speed, as

$$\eta_{\max} = \frac{B}{8F}\frac{C_e}{2H_e}, \tag{3}$$

where B is memory bandwidth in terms of bytes per seconds and we assume that one words is expressed in eight bytes. In other words, we assume $G = B/8$. In this notation, B/F in equation (3) is B/F in the widely used form. One processor of K computer has the theoretical peak of 128 Gflops and effective memory bandwidth of around 50GB/s. Thus, we have $B/F \sim 0.4$, $C_e \sim 2000$ and $H_e = 5$. Thus, we have $\eta_{\max} \sim 10$. In other words, efficiency of these calculations are actually not limited by the main memory bandwidth but by something else. The efficiency should be much closer to 100%.

This observation may be a bit surprizing, since the common wisdom is that the efficiency of explicit methods on modern cache-based architectures is limited by the main memory bandwidth.

There are several reasons why the achieved efficiency numbers of the best codes available on K computer are much lower than the theoretical limit. One is simply that these codes are not written in the way which satisfies the assumption in the derivation of equation (1). We assumed that each variable is read and written only once per one timestep, and all intermediate results are stored only in the cache and never

application	Yashiro et. al.	Yang et. al.	Hotta et.al.	This work
reference	[7]	[1]	[8]	–
method	explicit	implicit	explicit	explicit
# grids	5.5×10^{12}	1.29×10^{11}	2.61×10^{11}	3.97×10^{12}
WCT per Timestep (seconds)	12.4	9	0.254	3.37
# grids processed per second	4.6×10^{11}	8.6×10^{10}	1.03×10^{12}	1.18×10^{12}
flop per mesh per timestep	1888	93023	2505	4246
Computer	K computer	TaihuLight	K computer	Gyoukou
Efficiency(%)	10.2	7	24.3	21.5

TABLE I
COMPARISON OF STATE-OF-THE-ARTS HIGH-RESOLUTION MESH SCHEMES

spill over to the main memory. If we use 4-stage Runge-Kutta method, a usual implementation would require one read and write in each stage, thus we need four read and four write operations per variable. In theory, we should be able to avoid these additional read and write by cache blocking. However, that would effectively require the implementation of temporal blocking.

Another difficulty is that the processor architecture of K computer requires that the loop count for the innermost loop to be relatively large (such as 256 or more) in order to achieve reasonable efficiency. On the other hand, efficient implementation of temporal blocking requires the size of one dimension in the block to be read in to LLC to be very small (such as 16 or 32). Thus, to achieve the efficiency defined in equation (1) is not easy on recent machines.

In this paper, we propose a new approach to achieve very high efficiency for compressive CFD calculations, by means of temporal blocking, new CFD scheme designed to achieve high efficiency with temporal blocking, and automatic generation of the actual program from high-level description of the original equation of fluid and spacial difference scheme. As we have stated, it is almost impossible to take advantage of the temporal blocking with widely used combination of 4-point staggered grid and 4-stage Runge-Kutta time integration. In order to achieve a meaningful speedup with temporal blocking, we need a high-order scheme with a smaller number of stages in time and narrower stencil size in space. We have constructed a new scheme, SL4TH3, based on the collocation grid and Hermite time integration. SL4TH3 achieves fourth order accuracy in space and third-order accuracy in time, with only one space derivative evaluation per timestep. Thus, $N_s = 2$ for SL4TH3, compared to $N_s = 12$ for widely used schemes.

One practical problem of the SL4TH3 scheme is that it is based on the formal differentiation of the original equation and thus is very complicated with a large number of terms. This does not necessarily means that the calculation cost is high, since it requires one derivative calculations per timestep, while 4-stage Runge-Kutta methods require four derivative calculations. However, the program becomes very complicated and long, and it is impractical to write and debug the code. Therefore, we rely on automatic generation of actual stencil by formula manipulation programs and automatic generation of actual code with temporal blocking and parallelization from stencil by our Formura DSL [9], [10].

We have measured the performance of our generated code on two systems with PEZY-SC2 processors: Gyoukou and Shoubu System B. There are two main reasons why we chose these PEZY-SC2 based systems. One is that they currently offer the highest performance per watt, and thus can be regarded as most efficient HPC systems. The other reason is that they have probably the lowest relative memory bandwidth (B/F ratio) among existing HPC systems listed in, for example, Top500 or Green500. Thus, if we can demonstrate that our approach works well on PEZY-SC2 based systems, we can regard our approach is "future-proof", since most likely HPC systems in the future will have lower B/F numbers than that of mainstream HPC systems at present.

The measured efficiency was 21.5%, which is comparable to the efficiency of highly optimized CFD calculation on K computer. K computer has the nominal B/F value of 0.5, which is around 20 times higher that that of PEZY-SC2. We have successfully demonstrated that the simulation program with temporal blocking, automatically generated from well-conceived finite-difference scheme, can achieve very high efficiency on actual hardware with low memory bandwidth.

We describe our new approach in section 2 and measured performance in section 3, and we summarize our work in section 4.

II. NEW APPROACHES

A. The SL4TH3 scheme

As we discussed earlier, in order to achieve high efficiency using temporal blocking, it is necessary to use high-order scheme with stencils of small size and small number of stages. High order in space can be achieved just by using high-order spatial difference operator. However, we need to achieve high order in time, without making the stencil size large. In other words, we need a high-order time integration scheme which requires the minimum number of derivative calculations per timestep.

The obvious way to achieve high order in time with one derivative calculation per timestep is the use of the predictor-corrector method. In the predictor-corrector method, we first predict the solution at the new timestep from the polynomial fit of solution (usually the calculated derivatives) of several previous timesteps, and then calculate the derivatives at the new time, and usually apply the correction to the new solution using the calculated derivatives (so-called PEC mode). Sometimes the calculation of the derivatives and the correction are performed multiple times, to improve the stability. (P(EC)^n mode).

Unfortunately, an n-step predictor-corrector scheme need to store the time derivatives calculated in $n-1$ previous steps, resulting in the increase of the memory usage and decrease of the effect of the temporal blocking.

Yet another way to achieve high-order time integration with single derivative calculation per timestep is the direct calculation of high-order time derivatives from high-order space derivatives. We can calculate the high-order time derivatives by taking the time derivative of the original partial differential equations and replace the terms of time derivatives in the right-hand side of the new equation by space derivatives using the original equation. This approach have been used in the IDO scheme[11]. However, the calculation cost of high-order time derivatives is prohibitingly large, in particular for three-dimensional calculations. Therefore, we adopted a hybrid of predictor-corrector method and Taylor expansion method: the Hermite scheme [12].

In the Hermite scheme, we do calculate the first time derivatives of the left-hand side of the original equations from spatial derivatives, and use the predictor-corrector scheme to achieve higher order in time. In the case of usual predictor-corrector scheme, the order of a p-step scheme is p. The order of p-step Hermite predictor-corrector method is $2p$. Thus, this scheme has been widely used for accurate time integration of particle systems.

In order to achieve the fourth-order accuracy, the Hermite scheme requires two steps, however, the order of the predictor polynomial can be lower. In this case, the fourth-order Hermite scheme becomes effectively a single-step fourth-order scheme[13]. Thus, it is ideal for the use with temporal blocking.

Let us illustrate the derivation of high-order time derivatives in the case of the continuity equation,

$$\rho_t = -(u_x + v_y + w_z)\rho - u\rho_x - v\rho_y - w\rho_z, \qquad (4)$$

where ρ, u, v, w are the density and x-, y-, and z-components of the velocity. The subscript α indicates the partial derivative on α, $\partial/\partial\alpha$. By differentiating the both hand sides by t we obtain:

$$\begin{aligned}\rho_{tt} = &-(u_x + v_y + w_z)\rho_t - (u_{tx} + v_{ty} + w_{tz})\rho \\ &- u\rho_{tx} - v\rho_{ty} - w\rho_{tz} - \rho_x u_t - \rho_y v_t - \rho_z w_t. \end{aligned} \qquad (5)$$

This gives the second time derivative of the density. We still have terms like ρ_{tx}, which can then be rewritten as

$$\begin{aligned}\rho_{tx} = &-(u_x + v_y + w_z)\rho_x - (u_{xx} + v_{xy} + w_{xz})\rho \\ &- u\rho_{xx} - v\rho_{xy} - w\rho_{xz} - \rho_x u_x - \rho_y v_x - \rho_z w_x. \end{aligned} \qquad (6)$$

Now we have expressed ρ_{tt} in terms of high-order spatial derivatives.

As we can see, though this derivation is simple and straightforward, the resulted equation contains very large number of terms. Therefore we implemented a simple program to derive second-order time derivatives from the original equations, and let that program generate the inner kernel for the stencil calculation.

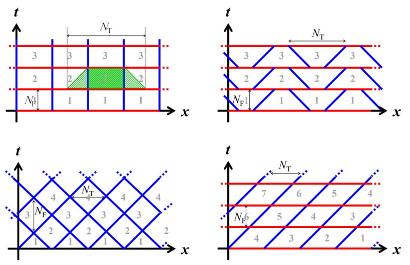

Fig. 1. Various methods of temporal blocking. The gray numbers indicate the order of regions that can be processed in parallel.

For the spatial derivatives, we use an 125-point ($5\times5\times5$ grids) finite difference form. For derivatives in one dimension, we derive the coefficients from fourth-order polynomial fitting. For cross derivatives we simply applied the finite difference operator in three directions sequentially to obtain the final form. We have tested the resulted scheme with a number of test problems and obtained very good result. In this paper, we call this scheme SL4TH3, as abbreviation for spatial Lagrange interpolation with fourth order and temporal Hermite interpolation with third order.

B. PEZY-SC2 and Gyoukou

In this section, we describe the features of Gyoukou, which consists of 13312 PEZY-SC2 processors.

One PEZY-SC2 processor chip consists of physically 2048 processing elements (PEs). Currently, 64 of them are disabled and the available number of PEs is 1984. Each PE can perform 1 or 2 multiply-and-add operation for FP64 and FP32 data. In this paper we use FP64 calculations only, to retain accuracy. With the clock speed of 700MHz, the theoretical peak speed is 2.8TF. Each SC2 processor have 4 channels of DDR4 memory, for the peak throughput of 76.8GB/s. Thus B/F is 0.027. The total size of LLC is 40MB. In addition to the cache memories, each processor has 20KB of local memory, in a separate address space.

A PEZY-SC2 processor has three levels of shared, non-coherent cache memories. Each PE has its own L1D, and 16 PEs share L2D, and all PEs LLC. Each PE runs either four or eight threads simultaneously. Thus, it is relatively easy to hide the latency of the arithmetic units and L1D.

In the Gyoukou system, eight SC2 processor chips are connected to single Xeon-D 1571 processor through a PLX PCIe switch chip.

The nominal theoretical peak speed of Gyoukou system is 37Pflops. At the time when we measured the performance, maximum configuration we can use was 8000 chips, for the peak performance of 22.2 PF.

C. Implementation of Temporal Blocking

In our previous implementation[9], [10], we have used theoretically efficient, but complicated temporal blocking based

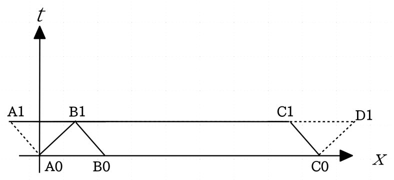

Fig. 2. Calculation on one processor. For simplicity, we show one-dimensional case. One processor initially has region B0-C0. In order to start calculation, it generates a dummy "wall" C0-D1, and calculation proceeds from right to left. During the calculation, the data of region A0B0 is transferred from the neighboring processor. At the end of one blocked calculation, the valid region is B1C1.

on trapezoids. One problem with the trapezoidal blocking is that the loop size changes during the interaction, and thus it is difficult to use many PEs of SC2 efficiently. Therefore, we decided to use a much simpler blocking scheme based on parallelogram (The right-bottom one in figure 1). In our current implementation, all PEs process one region, in order to make the size of the region, which is limited by the available size of LLC, as large as possible. For one timestep, one processor chip takes care of multiple regions sequentially. Each region have the size of (n_x, n_y, n_z), and one chip processes $N_x \times N_y \times N_z$ regions. One problem of the parallelogram blocking is that the calculation can only proceed from right to left (in the case of figure 1), and thus we cannot start calculation. We have "solved" this problem by using rectangular blocking (top left panel of figure 1) for the total regions of one processor. Figure 2 describes our current implementation. One processor initially has the region B0C0, and at the end of one blocked calculation, it should have B1C1. Thus, data of A0B0 should be transferred from the left-hand-side neighboring process. In order to start calculation, a dummy "wall" of C0D1 is generated, and calculation starts from there, and proceeds to left. During the calculation, the communication is done. Thus, we can see that the communication and the computation are naturally and maximally overlapped. In this scheme, the calculation of the triangle C0C1D1 will not be used, and A0B0B1 is redundant. Thus, the amount of unnecessarily calculation is not zero, but practically small enough since it is proportional to $N_T/(N_x n_x)$.

D. Spatial difference

We adopted five-point collocated central differences for all spatial derivatives. They are given by

$$\begin{aligned}
q &= q^0 \\
q_x &= \frac{1}{12h}(q^{-2} - 8q^{-1} + 8q^{+1} - q^{+2}) + O(h^4) \\
q_{xx} &= \frac{1}{12h^2}(-q^{-2} + 16q^{-1} - 30q^0 \\
&+ 16q^{+1} - q^{+2}) + O(h^4) \\
q_{xxx} &= \frac{1}{2h^3}(-q^{-2} + 2q^{-1} - 2q^{+1} + q^{+2}) + O(h^2) \\
q_{xxxx} &= \frac{1}{h^4}(q^{-2} - 4q^{-1} + 6q^0 - 4q^{+1} \\
&+ q^{+2}) + O(h^2) \qquad (7)
\end{aligned}$$

where q^i denotes$q(x + i\Delta x)$ and $q(x)$ is the value of some physical quantity at position x.

In the case of multi-dimensional space, we need finite difference approximation of cross derivatives. They are obtained by applying the finite difference formulae of equation (7) in each dimension. For example, the finite difference approximation of $\partial^2/\partial x \partial y$ is given by

$$\begin{aligned}
q_{xy} = \; & \big[q^{-2,-2} - 8q^{-2,-1} + 8q^{-2,+1} - q^{-2,+2} \\
& -8(q^{-1,-2} - 8q^{-1,-1} + 8q^{-1,+1} - q^{-1,+2}) \\
& +8(q^{+1,-2} - 8q^{+1,-1} + 8q^{+1,+1} - q^{+1,+2}) \\
& -q^{+2,-2} - 8q^{+2,-1} + 8q^{+2,+1} - q^{+2,+2}\big] \\
& \tfrac{1}{144h^2}. \qquad (8)
\end{aligned}$$

Note that the resulted formulae for cross derivatives are independent of the order in which the finite difference operators are applied.

There are several different ways to calculate necessary high-order finite difference terms. There are 3, 6, 11, and 15 first, second, third, and fourth-order spacial derivatives in three dimensions, respectively. A naive approach is to calculate all these terms directly from the data of 125 grid points (in a $5 \times 5 \times 5$ cube). The advantage of this approach is that calculations for different grid points are independent and thus can be assigned to different processors or different threads easily. In addition, we do not need any additional storage for intermediate variables except for that needed for the calculation of one difference term. Thus, we can expect that many-core processors such as PEZY-SC2 achieve fairly high efficiency.

On the other hand, this naive approach has several drawbacks. The first one is that actual calculation code derived by this approach contains many redundant operations. for example, q_x is calculated once for each grid points, but calculation of q_{xy}, as we can see from equation (8), contains the calculations of q_x for four different grid points.

Another problem is that the L1D hit rate would be low in the case of PEZY-SC2, since the amount of data read for one timestep of one grid point is much larger than the size of L1D. Since each grid point has 5 variables, the calculation of spatial difference terms requires the access to $5 \times 125 = 675$ variables or 5.4KB of data, while each processor of PEZY-SC2 has only 2KB of L1D cache, which is shared by either

```
r[t,x,y,z]_t = -u[t,x,y,z]*r[t,x,y,z]_x - v[t,x,y,z]*r[t,x,y,z]_y
          - w[t,x,y,z]*r[t,x,y,z]_z
          - r[t,x,y,z]*(u[t,x,y,z]_x + v[t,x,y,z]_y + w[t,x,y,z]_z)
u[t,x,y,z]_t = -u[t,x,y,z]*u[t,x,y,z]_x - v[t,x,y,z]*u[t,x,y,z]_y
           - w[t,x,y,z]*u[t,x,y,z]_z
           - p[t,x,y,z]_x/r[t,x,y,z] + c*vis1[t,x,y,z]/r[t,x,y,z]
v[t,x,y,z]_t = -u[t,x,y,z]*v[t,x,y,z]_x - v[t,x,y,z]*v[t,x,y,z]_y
          - w[t,x,y,z]*v[t,x,y,z]_z
          - p[t,x,y,z]_y/r[t,x,y,z] + c*vis2[t,x,y,z]/r[t,x,y,z]
w[t,x,y,z]_t = -u[t,x,y,z]*w[t,x,y,z]_x - v[t,x,y,z]*w[t,x,y,z]_y
          - w[t,x,y,z]*w[t,x,y,z]_z
          - p[t,x,y,z]_z/r[t,x,y,z] + c*vis3[t,x,y,z]/r[t,x,y,z]
p[t,x,y,z]_t = -u[t,x,y,z]*p[t,x,y,z]_x - v[t,x,y,z]*p[t,x,y,z]_y
                - w[t,x,y,z]*p[t,x,y,z]_z
           - gm*p[t,x,y,z]*(u[t,x,y,z]_x + v[t,x,y,z]_y + w[t,x,y,z]_z)
           - c2*(u[t,x,y,z]*vis1[t,x,y,z] + v[t,x,y,z]*vis2[t,x,y,z]
           + w[t,x,y,z]*vis3[t,x,y,z])
```

Fig. 3. The input differential equation.

four or eight threads. Thus, L1D is too small to keep the data for one grid point. If one processor can keep the data of 625 grid points, it can reuse the data of 500 grid points in the next iteration, if it handles the neighboring grid point. In order to achieve this goal, L1D must be large enough to keep the grid data, calculated spatial derivatives and other intermediate terms for the calculation of high-order time derivatives. Total size would be around 10KB. Thus, if the L1D size is more than 10KB/thread, L1D hit rate would be much better.

A unique feature of PEZY-SCx processors is that each processor has its local memory in separate address space. In the case of SC2 its size is 20KB. So each thread can have up to 5KB of local memory. Unfortunately, even this local memory is not large enough to keep the grid data.

We can eliminate redundant calculations by the following way. We first calculate all finite difference terms in x direction for all grid points and store them in memory. Then we calculate, again for all grid points, all terms which contain difference in y coordinates, using previously calculated x terms. Finally, we calculate all terms which contain difference in z coordinates, using all previously calculated xy terms.

This approach is optimal in terms of the operation count, but unpractical since it requires huge amount of memory to store high-order derivatives and thus not cache-friendly. We then can apply some cache blocking to this algorithm, to obtain the most efficient algorithm.

Currently, we have implemented two approaches. In the first approach, all difference terms are calculated independently. In the second scheme, the redundant calculation is removed only in the calculation of terms containing z coordinate. In the second scheme, for each grid point, we calculate terms which do not contain the finite difference in z direction, such as the x, y, xx and xy terms. Calculation proceeds in the direction of z coordinate. Thus after we calculated non-z terms to grid point k (here k denotes the grid number along z axis), we can calculate z terms for grid point $k-2$. This means we need to store the finite differences for only five grid points in z direction. These differences are stored not in the cache memories but in the local memory. Thus, we should be able to reduce the access to grid data and yet to improve the L1D hit rate.

In the rest of this paper, we call the first approach as naive difference and the second one modified difference. The operation count for the naive difference (only for spatial difference operations) is 2849 and for modified difference 1415. Note that these two approaches give the result same to each other except for the roundoff error.

E. Code generation

In our current implementation, only the stencil kernel for one block is problem-specific, and all other codes can be used for any stencil calculation. The kernel code is generated by FORMURA system[9], [10] from the description of the finite difference scheme (see section II-A), and the finite difference scheme itself is generated from the original fluid equation (figure 3) by formal differentiation and templates for space derivatives. The generated kernel for the update of actual grid point is around 400 lines of code, with some lines containing around 100 terms in the case of naive approach for spatial differences.

III. Measured performance

To measure the performance, we measured the time for one blocked step. The fluctuation of the wall clock time turned out to be very small. The execution time is measured by the wallclock timer, and operation count is from the counted number of interactions calculated. We excluded the operations for redundant and unused regions we described in section II-C.

We first present the parallel performance and scalability of the code based on the naive approach, measured on the Gyoukou system, and then present the single-node performance of the improved approach. The reason why we do not present the performance numbers of the second approach on the Gyoukou system is that Gyoukou was turned off on March 31, 2018 and currently unavailable.

A. Efficiency and Scalability on Gyoukou

The computation of spatial derivative with the naive approach requires 2849 floating-point operations, while the equation of motion and other small calculations require 1415 floating-point operations. This adds up to 4264 floating-point operations per grid point.

We used this operation count to calculate the total number of floating-point operations. Because of the limited availability of the Gyoukou system, we could not finish the strong scaling test. So in this paper we report the result of weak scaling test only. For the weak-scaling measurement, we have performed runs with 792^3 grid points per MPI process, and varied the number of processes from 512 to 8000. For all measurements of the naive approach on Gyoukou, we used $N_T = 4$. We will discuss the effect of N_T in more detail for the case of the improved scheme in the next subsection.

Figure 4 shows the time per one timestep. We can see that the total time per timestep does not show any variation, while the communication time shows rather large variations and tends to be larger for larger number of MPI processes. As we described in section II-C, the communication is perfectly

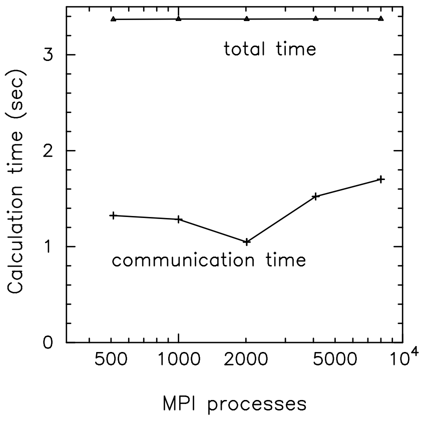

Fig. 4. Time per timestep in second for the weak-scaling test. The horizontal axis is the number of MPI processes. Triangles and crosses show the total time per timestep and communication time per timestep, respectively.

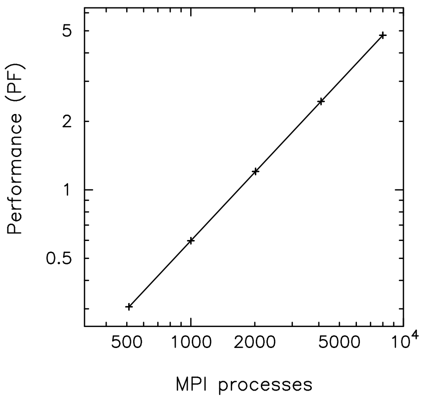

Fig. 5. Performance in petaflops for weak-scaling test. The calculation speed in Petaflops is plotted as a function of the number of MPI processes.

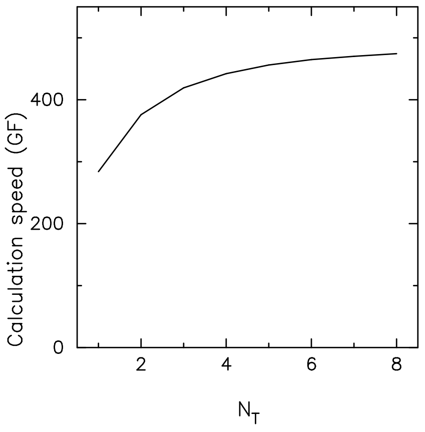

Fig. 6. Calculation speed plotted as a function of the size of temporal blocking N_T. The improved scheme is used.

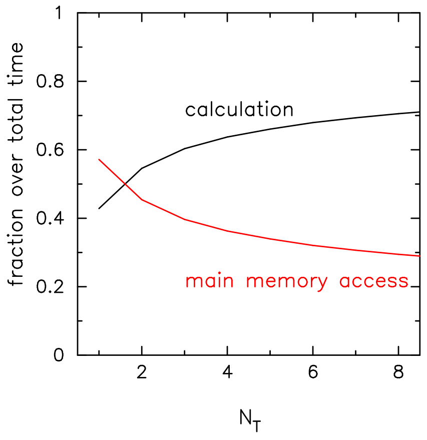

Fig. 7. Ratios of times spent in calculation and main memory access plotted against the size of temporal blocking N_T. The improved scheme is used.

overlapped with communication. Therefore, as long as the communication time does not exceed the calculation time, the total time remains constant.

Figure 5 shows the calculation speed in terms of Petaflops. The calculation speed shows perfectly linear speedup as we increase the number of processes.

To summarize, the performance of run on 8000 MPI processes is 4.78 PF, or 21.5% of the theoretical peak performance.

B. Improved approach

In this section, we present the performance of the "improved" approach. As we have discussed in section II-D, the performance of the naive approach for the spacial difference is rather low on machines with very small L1D cache. In fact, the cache miss rate of L1D is nearly 40% for the naive approach. At the same time, the operation count is rather large.

In the improved scheme, both the number of unique words and the total number of words accessed for one grid point is reduced by a large factor (factor of five for unique words, from 625 to 125. and by a similar factor for total number of words). Thus, we hoped to achieve much better efficiency. Currently, however, the efficiency is slightly lower. The actual calculation speed is improved by 25%, because the number of redundant calculations for space differences is reduced.

Figure 6 shows the effect of the temporal blocking. As we increase N_T, we can see quite significant improvement of the total calculation time.

The fact that we have achieved quite significant speedup by means of the temporal blocking implies that the calculation speed of our implementation for large N_T is no longer limited by the main memory bandwidth, and we have achieved the efficiency close to 20%, which is comparable to those achieved on machines with much higher memory bandwidth. Figure 7 shows how the calculation time and main memory access time varies when we change N_T. For $N_T = 1$ (no temporal blocking), main memory access consumes 60% of the total time, but for $N_T = 8$, it consumes less than 30%. This clearly is the reason for the speedup we observed for the total time.

The fact that 70% of the total time is spent for calculation in the case of $N_T = 8$ seems to imply we should be able to achieve the overall efficiency somewhat higher than 20%, which we actually measured. We are still investigating the reason for this discrepancy.

The number of grid points processed on Gyoukou in one second was 1.18×10^{12}. If Gyoukou were still available, the improved scheme would provide 1.5×10^{12}, assuming the scaling similar to that of the original scheme.

IV. Discussion and Summary

In this paper, we described the implementation and performance of a highly efficient explicit scheme SL4TH3, combined with temporal blocking, for the simulation of compressive fluid on Gyoukou supercomputer.

The measured performance is 4.78 PF, or 21.5% of the theoretical peak, for simulation of the subsonic turbulence with the grid of 15840^3 on 8000 PEZY-SC2 chips. This speed corresponds to the update of 1.18×10^{12} grid point per second. This measurement was done with our "naive" scheme, and "improved" scheme should have given 1.5×10^{12} updates. Also, for the "improved" scheme, the effect of the temporal blocking is around a factor of 1.7 for the case of $N_T = 8$. So we have achieved quite significant speedup by means of temporal blocking.

The achieved efficiency is comparable to what is achieved on K computer, even though the relative memory bandwidth of PEZY-SC2 is around 1/20 of that of K computer. Also, we demonstrated that the proper use of temporal blocking can actually decrease the total execution time.

Our result implies that it is possible to use HPC systems with relatively low memory bandwidth (and also network bandwidth) quite efficiently for calculations of explicit schemes on regular grids. It is also possible to apply the idea of temporal blocking, to further improve the performance.

In our approach, as we stated in section II-E, the implementation of parallelization and temporal blocking framework and that of the finite difference scheme for the differential equation are completely separated, and the optimization such as the overlapping of the communication and calculation is taken care in the side of the parallelization framework. The implementation of the finite difference scheme should provide the code to perform the update of one block in the right-bottom panel of figure 1. All of the physics and mathematics of the problem to be solved is in this part, and not in the parallelization framework. We can apply our FORMURA framework to any explicit scheme on the regular grid, though the effect of the temporal blocking might not be as good as that for the SL4TH3 scheme.

Explicit methods, in particular when combined with adequate timescale transformations such as the RSS method, can outperform traditional implicit methods quite easily, since the maximum timesteps allowed are not much different, while there are huge differences in the requirements for communication latency and main memory bandwidth.

We believe it is important to shift the research and development from clever and complex methods for implicit iterative solvers to explicit methods and clever ways to scale physical quantities to relax the limits such as the CFL limit. It is also important to develop and improve new high-order explicit schemes which do not need high memory bandwidth and yet computationally efficient.

Acknowledgment

We express our deepest appreciation to late Dr. Takayuki Muranushi, whose contribution to this work was essential. This work was supported in part by The Large Scale Computational Sciences with Heterogeneous Many-Core Computers in grant-in-aid for High Performance Computing with General Purpose Computers in MEXT (Ministry of Education, Culture, Sports, Science and Technology-Japan), and also in part by JAMSTEC, RIKEN, and PEZY Computing, K.K./ExaScaler Inc. In this research computational resources of the Gyoukou supercomputer, developed under the "Large-scale energy-efficient supercomputer with inductive coupling DRAM interface" project of NexTEP program of Japan Science and Technology Agency, has been used.

References

[1] C. Yang, W. Xue, H. Fu, H. You, X. Wang, Y. Ao, F. Liu, L. Gan, P. Xu, L. Wang *et al.*, "10m-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 6.

[2] M. Rempel, "Solar Differential Rotation and Meridional Flow: The Role of a Subadiabatic Tachocline for the Taylor-Proudman Balance," *ApJ*, vol. 622, pp. 1320–1332, Apr. 2005.

[3] ——, "Flux-Transport Dynamos with Lorentz Force Feedback on Differential Rotation and Meridional Flow: Saturation Mechanism and Torsional Oscillations," *ApJ*, vol. 647, pp. 662–675, Aug. 2006.

[4] K. Takeyama, T. R. Saitoh, and J. Makino, "Variable inertia method: A novel numerical method for mantle convection simulation," *New Astronomy*, vol. 50, pp. 82–103, Jan. 2017.

[5] N. Y. Gnedin and T. Abel, "Multi-dimensional cosmological radiative transfer with a Variable Eddington Tensor formalism," *New Astronomy*, vol. 6, pp. 437–455, Oct. 2001.

[6] T. Muranushi and J. Makino, "Optimal temporal blocking for stencil computation," *Procedia Computer Science*, vol. 51, pp. 1303–1312, 2015.

[7] H. Yashiro, M. Terai, R. Yoshida, S.-i. Iga, K. Minami, and H. Tomita, "Performance analysis and optimization of nonhydrostatic icosahedral atmospheric model (nicam) on the k computer and tsubame2.5," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, ser. PASC '16. New York, NY, USA: ACM, 2016, pp. 3:1–3:8. [Online]. Available: http://doi.acm.org/10.1145/2929908.2929911

[8] H. Hotta, M. Rempel, and T. Yokoyama, "High-resolution calculations of the solar global convection with the reduced speed of sound technique. i. the structure of the convection and the magnetic field without the rotation," *The Astrophysical Journal*, vol. 786, no. 1, p. 24, 2014.

[9] T. Muranushi, H. Hotta, J. Makino, S. Nishizawa, H. Tomita, K. Nitadori, M. Iwasawa, N. Hosono, Y. Maruyama, H. Inoue *et al.*, "Simulations of below-ground dynamics of fungi: 1.184 pflops attained by automated generation and autotuning of temporal blocking codes," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 23–33.

[10] T. Muranushi, S. Nishizawa, H. Tomita, K. Nitadori, M. Iwasawa, Y. Maruyama, H. Yashiro, Y. Nakamura, H. Hotta, J. Makino *et al.*, "Automatic generation of efficient codes from mathematical descriptions of stencil computation," in *Proceedings of the 5th International Workshop on Functional High-Performance Computing*. ACM, 2016, pp. 17–22.

[11] T. Aoki, "Interpolated differential operator (ido) scheme for solving partial differential equations," *Computer Physics Communications*, vol. 102, no. 1, pp. 132 – 146, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465597000209

[12] J. Makino, "Optimal order and time-step criterion for aarseth-type n-body integrators," *ApJ*, vol. 369, pp. 200–212, 1991.

[13] J. Makino and S. J. Aarseth, "On a hermite integrator with ahmad-cohen scheme for gravitational many-body problems," *PASJ*, vol. 44, pp. 141–151, 1992.