Acceleration of Boltzmann Equation for Core-Collapse Supernova Simulations on PEZY-SC Processors

Hideo Matsufuru[1(✉)] and Kohsuke Sumiyoshi[2]

[1] High Energy Accelerator Research Organization (KEK),
1-1 Oho, Tsukuba, Ibaraki 305-0801, Japan
hideo.matsufuru@kek.jp
[2] National Institute of Technology, Numazu College,
3600 Ooka, Numazu, Shizuoka 410-8501, Japan
sumi@numazu-ct.ac.jp

Abstract. Performing large scale numerical simulations is essential to understand the explosion mechanism of core-collapse supernovae. It is mandatory to solve a multi-physics system described by coupled equations of hydrodynamics and neutrino-radiation transfer in multi-dimensions. Since the neutrino transfer is in principle governed by the Boltzmann equation in six-dimensional coordinates, numerical simulations require large computational resources. In this work, we focus on the acceleration of the Boltzmann equation by exploiting the PEZY-SC processors. The PEZY-SC processor possesses many-core MIMD cores on each chip, and works as an accelerator device similarly to GPUs. We examine two simulation codes for the neutrino transport in the supernova simulations. One is a radiation-hydrodynamics code under the spherical symmetry. The second is a 6D Boltzmann equation solver applied to two-dimensional space. We examine the bottlenecks of these codes and offload them to the PEZY-SC devices. The performance is measured on the Suiren Blue and Suiren2 systems at KEK.

Keywords: Many-core accelerator · PEZY-SC · Boltzmann equation · Supernova explosion · Radiation-hydrodynamics

1 Introduction

Core-collapse supernova explosions are phenomena that occurs at the end of stellar life whose mass is larger than 10 times the solar mass. The gravitational compression of Fe core and the subsequent core bounce launches the shock wave that leads to the explosion. Despite the long history of investigations based on this general idea, the detailed mechanism of the core-collapse supernova explosion is still elusive, because only prohibitively large-scale numerical simulations can explore the multi-dimensional mechanism with multi-physics in multi-scale [1,2]. Precise understanding of the supernova explosion is one of the essential

O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12253, pp. 177–192, 2020.
https://doi.org/10.1007/978-3-030-58814-4_13

issues in astrophysics in order to reveal the origin of heavy elements, the formation of compact objects (neutron stars or black holes), and observation of neutrino bursts and gravitational waves.

Among the ingredients of this multi-physics system, the Boltzmann equation for the neutrino radiation regulates the scale of numerical simulations, since its time evolution is described by a six-dimensional (space and neutrino momentum) distribution function. Increasing the resolution of space and neutrino momentum, the numerical cost rapidly increases. Fully six-dimensional computation of the Boltzmann equation has become feasible only recently. To provide quantitative theoretical prediction at the level demanded for comparison with observational data, however, it is essential to systematically study variety of massive stars and the effect of many components of microphysics, such as the equation of state for the dense matter and the neutrino reaction processes.

Recent supercomputers are categorized into two types. One is massively parallel clusters whose nodes are composed of multi-cores and shared memory, such as the K-computer and the forthcoming Fugaku supercomputer in Japan. The other type makes use of the arithmetic accelerators, such as GPUs, and are becoming popular for its high cost performance. Most of supernova simulations have been performed on the former, presumably because the latter requires involved implementation of code to offload the tasks to the accelerator devices. In our previous work, we offloaded the bottlenecks of the Boltzmann equation in supernova simulations to GPUs and achieved substantial acceleration [3,4].

In this work we investigate another accelerator architecture, the PEZY-SC processors. The PEZY-SC processors are novel many-core architecture that share several advantages with GPUs while also have distinct features, such as the threads working as MIMD processing elements. To exploit their potentially high arithmetic performance, understanding of their characteristic structure is important. It is worth establishing the knowledge; which kinds of applications are suitable to this architecture and how to optimize the codes.

We examine two types of the Boltzmann equation solvers. One is the radiation-hydrodynamics code under the spherically symmetry, *i.e.*, on one-dimensional spatial coordinate [5,6]. This code was also used in our previous studies on GPUs [3,4]. The second code is a multi-dimensional Boltzmann equation solver [7], which is a basis of recent large-scale simulations.

From a numerical point of view, three types of problems are examined: (1) A linear equation solver for a block tridiagonal matrix whose block matrices are dense, (2) Determination of the inverse of dense block matrices, and (3) Computation of neutrino reaction processes which are represented as integration of reaction rates over the energy and angles of neutrinos. We examine whether these bottlenecks can be accelerated by offloading to the PEZY-SC processors. As shown below, although effect of offloading is restricted by the bandwidth of device memory, one needs to employ optimization techniques to properly exploit the potential performance of hardware under this limitation. Establishing such recipes is also useful to prepare for future architecture including the next generation of the PEZY-SC processor.

This paper is organized as follows. Next section summarizes the formulation to investigate the supernova explosion focusing on the structure as computational problems. In Sect. 3 we summarize the features of PEZY-SC processors and the programming environment. The implementation of codes and the results of performance measurement are described in Sects. 4 and 5 for one- and two-dimensional simulations, respectively. Section 6 is devoted to our conclusion.

2 Formulation

2.1 Boltzmann Equation

For numerical simulations of supernovae, one needs to solve the evolution of the stellar matter at high temperature and density and the radiation transfer of neutrinos generated by the weak interaction simultaneously. The former is described by hydrodynamics under the gravitational effect governed by the general relativity. The latter is described by the Boltzmann equation, since the neutrinos are not always in statistical equilibrium and thus their distribution is not always isotropic and equilibrium values, therefore, a function of spatial coordinates and their momentum. Leaving the details of these formulation to literature [5–8], here we briefly summarize the costly parts in numerical simulations.

The general relativistic Boltzmann equation for the neutrino distribution function is written as

$$\frac{dx^\mu}{d\tau}\frac{\partial f_\nu}{\partial x^\mu} + \frac{dp^i}{d\tau}\frac{\partial f_\nu}{dp^i} = \left(\frac{\delta f_\nu}{\delta \tau}\right)_{\text{coll}} \tag{1}$$

where x^μ are the space-time coordinates and p^i the momentum coordinates, τ the affine parameter (specific time). This equation describes the conservation of the neutrino distribution function, and thus rewritten as an evolution equation. The left hand side describes the time derivative of f_ν and advection terms. The collision term, the right hand side, represents the change of the neutrino number caused by the weak interaction with matter. This term is composed of several reaction processes [9], and computationally demanding.

To solve the Boltzmann equation, the discrete ordinate (S_N) method is adopted through finite differencing the spatial and momentum coordinates. We adopt a fully implicit differencing for time advance since the equation is stiff with largely different time scales due to energy dependence of the interaction. In contrast to the explicit scheme where the time step is constrained by the Courant number, it is advantageous to increase the step size to follow the time evolution in a long time scale. However, it is numerically expensive to solve a linear equation with a large sparse matrix at each time step for time advancing. This linear equation solver tends to be the primary bottleneck of our simulations.

2.2 Spherically Symmetric (1D) Simulation

For one-dimensional simulation, *i.e.* under the spherical symmetry, coupled equations of hydrodynamics and neutrino transfer under general relativistic geometry

can be solved [5,6]. In this case the evolution equation contains both the degrees of freedom of the neutrino radiation and hydrodynamics. Under the spherical symmetry, the neutrino distribution function is a function of the radial coordinate r, the neutrino energy E_ν, and the angle of neutrino momentum against the radial direction θ_ν. Four neutrino species are treated ($N_\nu = 4$): electron neutrino ν_e and μ-neutrino ν_μ, and their anti-neutrinos $\bar{\nu}_e$ and $\bar{\nu}_\mu$. In addition to the distribution function $f_\nu(r, E_\nu, \theta_\nu)$ for each neutrino species, eleven hydrodynamical variables, such as the temperature and entropy, are evolved simultaneously in the time evolution equation. We discretize radial coordinate, neutrino angle, and energy to N_r, $N_{\rm ang}$, and N_{E_ν} grids, respectively. Thus at each radial point there are $N_{E_\nu} \cdot N_{\rm ang} \cdot N_\nu$ neutrino degrees of freedom and $N_{\rm hyd} = 11$ hydrodynamical variables. Typically $N_r = O(100)$, $N_{\rm ang} = O(10)$, $N_{E_\nu} = O(10)$ are adopted currently, while ten times higher resolutions are demanded for more quantitative investigation.

In the implicit scheme, the most costly part of numerical simulation is a linear equation solver for the coefficient matrix of the time evolution equation. From the above equation, the coefficient matrix results in a tri-diagonal form,

$$M = \begin{pmatrix} B_1 & C_1 & 0 & & \cdots & \\ A_2 & B_2 & C_2 & 0 & & \\ 0 & A_3 & B_3 & C_3 & & \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ & & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & \cdots & & 0 & A_n & B_n \end{pmatrix}, \tag{2}$$

where $n = N_r$ is the number of radial grids. At each radial point, i, the block matrices A_i, B_i, C_i are dense matrices of rank $N_{\rm max} = N_{E_\nu} N_{\rm ang} N_\nu + N_{\rm hyd}$.

In the case of one-dimensional simulations, direct methods for the linear equation are also applicable [10]. We instead adopt an iterative solver algorithm based on the Krylov subspace method as a preparation to the multi-dimensional simulations. We employ the BiCGStab algorithm with the weighted Jacobi-type preconditioner [11]. The preconditioning is represented as

$$x_{k+1} = \omega[-M_D^{-1}(M - M_D)x_k + M_D^{-1}b] + (1 - \omega)x_k \tag{3}$$

where ω is the weight parameter, M_D the block-diagonal part of M in Eq. (2). Since M_D^{-1} is also in a block diagonal form, it can be determined preceding to the solver algorithm. The procedure to determine the weight parameter ω is described in Ref. [11]. Here we just note that it suffices to determine several small eigenvalues approximately by applying the Arnoldi algorithms to a Krylov subspace of degree around 20. The preconditioning is performed by repeatedly applying the operation (3), $N_{\rm Jacobi}$ times, which is practically determined. $N_{\rm Jacobi} = 25$ sufficiently works in the present case.

2.3 Multi-dimensional Simulation

In our multidimensional simulations, the hydrodynamics of stellar matter and the Boltzmann equation of neutrinos are solved not in parallel but sequentially

at each substep of the time evolution. Thus we focus on the Boltzmann equation for the neutrino radiation transfer [7], as a benchmark code of the coupled system with the hydrodynamics under the gravity [12]. The neutrino distribution is a function of spatial coordinate $x = (r, \theta, \phi)$ and neutrino energy and two angles, $(E_\nu,\ \theta_\nu,\ \phi_\nu)$. Applying the discrete ordinate method and employ the implicit differencing scheme for the time advancing, the structure of numerical problem is similar to that of the spherically symmetric case. The coefficient matrix of the time evolution equation takes a similar form to Eq. (2), while the subdiagonal blocks exist in not only r-direction but also θ- and ϕ-directions. These subdiagonal block matrices, $A_x^{(r)}$, $A_x^{(\theta)}$, $A_x^{(\phi)}$ and $C_x^{(r)}$, $C_x^{(\theta)}$, $C_x^{(\phi)}$ represent the advection of neutrinos. Contrary to the 1D case, these subdiagonal block matrices are not dense but just diagonal in the neutrino energy and angles. As a linear equation solver for this coefficient matrix, we currently adopt an iterative algorithm without preconditioning, while its application is straightforward.

2.4 Related Works

Use of GPUs in the simulations of core-collapse supernovae has been progressed mainly for application to the hydrodynamics. As for the simulation code including the neutrino transport, the VERTEX code was ported to GPUs by employing CUDA of NVIDIA [13]. Its most time consuming part was one reaction term in the collision term of the Boltzmann equation and offloaded to GPUs. On the Kepler architecture, the target kernel was accelerated by factor of 54 which resulted in 1.8 times acceleration of the whole simulation time compared to the execution on the host processors.

In our previous work [3,4], we offloaded the bottlenecks of the supernova simulations under the spherical symmetry to GPUs and demonstrated their efficiency. In Ref. [3] the iterative linear equation solver was offloaded by implementing with OpenACC and exploiting cuBLAS library provided by NVIDIA. In successive work [4] the other time consuming parts, the computation of the collision term and the inversion of the block matrices, were also offloaded by using CUDA framework. The present work is based on these results.

3 Pezy-SC Processor

3.1 Architecture

The PEZY-SC processors are novel many-core architecture provided by Pezy Computing K.K. The processor is hierarchically composed of PEs (Processing Elements) each launches eight MIMD threads. Being equipped in a liquid immersion cooling system "ZettaScaler" provided by ExaScaler Inc., it achieves high power efficiency as awarded in the Green500 list several times [14]. Currently two generations of the PEZY-SC processors are available: PEZY-SC and PEZY-SC2. Hereafter we call the former as PEZY-SC1 for distinction. We develop our simulation codes for PEZY-SCs on the Suiren Blue and Suiren2 systems at KEK whose features are summarized in Table 1.

Table 1. Specification of the Suiren Blue and Suiren2 systems at KEK.

System	Suiren Blue (64 nodes)	Suiren2 (48 nodes)
Accelerator	PEZY-SC1	PEZY-SC2 (700 MHz)
Peak DP performance/device	1.5 TFlops	2.8 TFlops
Number of PE/device	1024	1984
Number of threads per PE	8	8
Number of PE per City	16	16
Size of local memory/PE	16 KB	20 KB
L1 cache	2 KB/2PE	2 KB/PE
L2 cache	64 KB/City	64 KB/City
L3/Last Level cache	2 MB/Prefecture	40 MB/device
Device memory size	16 GB	64 GB
Device memory bandwidth	102 GB/s	76.8 GB/s
Host-device network/device	PCIe Gen3 ×8	PCIe Gen3 ×8

Each node of the Suiren Blue system is composed of an Intel Xeon E5-2618Lv3 host processor and four PEZY-SC1 processors connected by the PCIe Gen3 with 8 lanes. The PEZY-SC1 device possesses 1024 PEs that are hierarchically installed in units of Village (four PEs), City (four Villages), Prefecture (16 Cities) so that four Prefectures compose one device. Each PE launches 8 threads that share the local memory of PE. The peak performance of one PEZY-SC processor is 1.5 and 3 TFlops for double and single precision arithmetics, respectively. The device memory bandwidth is 102 GB/s. The L1 cache is shared by 2PEs, L2 by 16 PEs in a City, and L3 by 16 Cities in a Prefecture.

Each node of the Suiren2 system is composed of an Intel Xeon D1571 host processor and eight PEZY-SC2 processors connected by the PCIe Gen3 with 8 lanes. While the number of cores is 2048 in the architecture design, current products disable 64 PEs to increase the yield of the processor chip. Furthermore the clock cycle is reduced from 1 GHz in the design to 700 MHz for stable operation. The structure of PEZY-SC2 device is similar to that of PEZY-SC. The last level cache assigns 32 MB for access from PEs and 8 MB for PCIe.

Although the arithmetic performance of the PEZY-SC2 processor is higher than that of PEZY-SC1, the latter has better memory bandwidth. The primary bottleneck of our application is regulated by the device memory bandwidth. Thus we mainly use the PEZY-SC1 processor for developing the code, and at the last stage measure the performance on the PEZY-SC2 processors for comparison.

3.2 Programming Environment

As the programming environment, Pezy Computing provides PZCL SDK which is a subset of the OpenCL framework while includes extensions to exploit the PEZY-SC architecture. We use the PZCL SDK 4.1 on Suiren Blue and 5.0 on Suiren2. For PEZY-SC1 there are two addressing modes in PZCL, 32-bit and 64-bit. In this work we use the latter considering our target size of simulations.

There are several functions added to the OpenCL standard to make use of the hierarchical structure of the PEZY-SC architecture. It is essential to understand such structure and corresponding functions to achieve desired performance. Here we summarize the several such features and techniques.

Hierarchical Synchronization. PZCL provides functions to synchronize the threads at several levels. Synchronization of threads in the Village, City, and the whole device are performed by calling built-in functions. Synchronization at other levels, such as in PE or Prefecture, is provided as assembly commands. Since the threads of PEZY-SC work as MIMD processes, memory access timing may differ thread by thread. Explicit synchronization of threads before memory access may increase the memory access performance by inducing coalesced access.

Hierarchical Cache. In the PEZY-SC architecture, the cache coherence is assured on user's responsibility. Thus flushing the cached data must explicitly be done by function calls. Similarly to the synchronization, there is a built-in function to flush the data at each level of the cache.

Changing Threads. Eight threads within PE execute the commands in turn in two bunches each containing four threads. These execution bunches are explicitly exchanged by a built-in function of PZCL, `chgthread()`. Putting this function after the device memory access may partially hide the memory access latency.

Local Memory. Each PE has a local memory shared by the threads in that PE. Usually this local memory is used as stack areas of the threads. It is possible to use this memory area as a shared memory of the threads by changing the size of stack area before launching the kernel using an API function in PZCL. The freed memory area can be accessed by directly referring the memory address.

3.3 Related Works

There have been several published works on utilizing the PEZY-SC processors including the porting and performance evaluation of libraries and simulation codes [15–18] as well as application to scientific studies [19,20]. Among these works, the problem of lattice QCD simulations [15] has similarity to this work and gives insights on the optimization of our code. In a viewpoint of efficiency in porting a codes to new architecture, Ref. [21] designs the OpenACC on PEZY-SC as an alternative programming framework to the currently available PZCL/OpenCL. In addition to these works, several projects to make use of the PEZY-SC processors are in progress.

4 Results of 1D Simulation

4.1 Implementation and Numerical Setup

Our base code is described in Fortran and parallelized with MPI. In our previous works on GPUs [3,4], we employed OpenACC or CUDA after porting the

bottleneck parts into the C language. For PEZY-SC processors we can make use of these C codes as the starting point to develop the code in PZCL. Since there is almost direct correspondence between the CUDA and PZCL kernels, the first step of porting to the latter is rather straightforward, while thereafter optimization specific to the PEZY-SC architecture is required.

The scale of 1D simulation is determined by the parameters N_r, N_{E_ν}, and $N_{\rm ang}$. While currently typical values of these parameters are (N_r, N_{E_ν}, $N_{\rm ang}$) = (256, 14, 6), we aim at increasing them up to (1024, 16, 8) or (1024, 24, 12) within acceptable cost. In Table 2, we summarize the elapsed time budget for ingredients of one Newton-Raphson iteration, that is repeated a few times within each time step, executed at $N_r = 256$ and with four MPI processes. The first part of Table 2 shows the result of execution on the host processor of Suiren Blue. As already mentioned in Sect. 1, there are three bottlenecks in our numerical simulation: (1) Iterative linear equation solver for the coefficient matrix of the time evolution equation, (2) Inversion of the block dense matrices of the coefficient matrix used in the weighted Jacobi-type preconditioning for the iterative solver, and (3) Calculation of the collision term of the Boltzmann equation. In the same way as the GPU case [3,4], we offload these parts to the Pezy-SC processors and optimize the corresponding kernel codes. Result for the PEZY-SC processor is also summarized in Table 2. In the following subsections, we explain the details of implementation and discuss these results.

Table 2. Time budget of one Newton-Raphson iteration on Suiren Blue system with four MPI processes.

(N_{E_ν}, $N_{\rm ang}$)	Elapsed time [sec]			
	(14, 6)	(16, 8)	(24, 12)	(32, 16)
Host (Suiren Blue)				
Hydrodynamics	0.0024	0.0037	0.0086	0.013
Advection term	0.022	0.029	0.069	0.12
Collision term	1.93	4.32	27.0	104
Matrix setup (inverse)	5.63	19.3	219	1217
Matrix setup (Arnoldi)	0.51	0.93	4.59	15.6
Iterative solver	9.37	14.9	71.9	213
PEZY-SC (Suiren Blue)				
Collision term	2.49	7.00	36.28	148
Matrix setup (inverse)	0.28	0.89	8.87	47.3
Matrix setup (Arnoldi)	0.16	0.24	0.89	2.57
Iterative solver	2.00	2.57	11.52	32.5
Matrix data conversion	0.44	0.95	6.72	21.2

4.2 Linear Equation Solver

Matrix-vector multiplication. The primary bottleneck of our numerical simulation is the linear equation solver for the coefficient matrix of the evolution equation. As noted in Subsect. 2.2, M in Eq. (2), $M - M_D$, and M_D^{-1} are the matrices in the preconditioned iterative solver. Since the determination of M_D^{-1} is the subject of Subsect. 4.3, here we regard M_D^{-1} as already determined explicitly. In the following we summarize the implementation of the matrix-vector multiplication, $y = Mx$, and optimization techniques applied.

Task Assignment. Since the MPI parallelization is done in the radial coordinate, each device processes them in units of the block in Eq. (2). We assign one block (one values of r) of vector y to one City, *i.e.* the 16 PEs sharing the L2 cache. For our present setup, $N_r = 256$ and four MPI processes, this is a practical choice, since each device processes the number of vector blocks equal to the number of Cities in the case of PEZY-SC1.

Boundary Communication. The boundary data of the vector x must be exchanged before the matrix is multiplied. Although this communication can be overlapped with the bulk part of the matrix-vector operation, we currently adopt the blocking communication since this overhead is relatively small as comparing the performance of M_D^{-1} and $M - M_D$.

Padding. The rank of the block matrices is $N_{\max} = N_{\rm ang} N_{E_\nu} N_\nu + N_{\rm hyd}$, and $N_{\rm hyd} = 11$ is always fractional with respect to the number of threads. Such an alignment may affect the memory access performance. We thus extend the arrays to $\tilde{N}_{\rm hyd} = 16$ by padding the matrices with unity and the vectors with zero.

Coalesced Access. The layout of the components of the block matrices is important for memory access performance. Representing the index of matrix component A_{ij} as an inline function `index(i,j)`, `index=i+Nmax*j` is a standard layout. Assigning computation of the component y_i to each thread, this layout corresponds to the coalesced access. However, this implies the access in `j`-loop is not continuous. Better performance is achieved with modified indexing, expressing in the C language,

$$\texttt{index} = (\texttt{i}\%\texttt{nth}) + \texttt{nth} * (\texttt{j} + \texttt{Nmax} * (\texttt{i}/\texttt{nth})), \tag{4}$$

where `nth` $= 8$ is the number of threads in PE.

Unrolling. The loaded value of x_j is reused by unrolling the `i`-loop. Considering the size of $N_{\max}$, $N_{\rm unroll} = 3$, 5, and 10 seem appropriate for (N_{E_ν}, $N_{\rm ang}$) = (14, 6), (16, 8), and (24, 12), respectively. We implemented the code with these three values of $N_{\rm unroll}$, and found that $N_{\rm unroll} = 3$ always shows the best performance. Since the number of rows processed in each thread may be different, the MIMD feature of the PEZY-SC thread is efficient in this implementation.

The performance of the matrix-vector multiplication for these matrices is displayed in Fig. 1. As already noted, no sizable effect of communication is observed

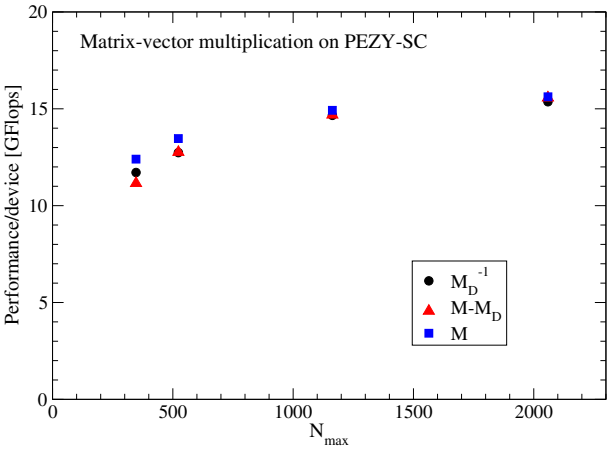

Fig. 1. The performance of the matrix-vector multiplication on PEZY-SC processor.

in the performance of M_D^{-1} and other two matrices. There is a tendency that M acquires better performance than M_D^{-1} in spite of the communication overhead. This can be explained as the effect of cached data of the right hand side vector x. Assuming complete reuse of cached data, the byte-per-flop of the bulk part of these matrices is about 4. Considering the device memory bandwidth, about 25 GFlops is the performance limit of the matrix multiplication. Ref. [15] estimated the sustained bandwidth of the device memory for the 'axpy' operation (`y = a*x+y`) which results in about 50 GB/s, half the peak bandwidth. We therefore realize that our code is sufficiently optimized. Compared to the result for NVIDIA Tesla P100 GPU (4.7 TFlops FP64 peak performance and 720 GB/s device memory B/W) in our previous work [3], the performance of the matrix-vector multiplication on PEZY-SC is about 10% of that on P100. This is explained by the less memory bandwidth of PEZY-SC than the latter.

The elapsed time of the whole iterative solver is shown in Table 2. For two small matrix sizes, acceleration is substantial even considering the overhead of matrix data conversion that includes the change of matrix layout and data transfer from the host to device. The latter is performed once before the iterative solver and takes 10% of the quoted time. This is reasonable considering about 8 GB/s of PCIe bandwidth. For larger sizes, this overhead becomes sizable, even considering that number of iteration until the convergence strongly depends on the parameters and the stage of time evolution. One needs to appropriately choose the number of MPI processes and devices so that this overhead becomes negligibly small compared to the iteration time.

Arnoldi Method. As a by-product of acceleration of matrix-vector multiplication, the Arnoldi method used to determine the weight parameter ω in Eq. (3) is also accelerated, since the consumed time is dominated by construction of the Krylov subspace rather than the Arnoldi algorithm itself. The former part is offloaded to

the accelerator devices and the resultant Hessenberg matrix is transferred to the host processor. The Arnoldi algorithms is performed at the host by calling the DGEEV routine of the LAPACK library with negligible numerical cost. The time budget of this part is denoted as 'matrix setup (Arnoldi)' in Table 2. Comparing to the results on the host processor, sufficient acceleration is achieved.

Result for PEZY-SC2 Processor. Finally we briefly summarize the performance result on the Suiren2 system. We do not quote the measured values of performance explicitly, since our code is not optimized for the PEZY-SC2 processor. We rather observe the qualitative difference of the performance on these two generations without changing the code. Among the parameter sets in Table 2, the largest size does not run presumably due to the required memory size exceeding that of the system. For the matrix-vector multiplication, we assigned one block on each r to one City. With this setup and present parameter values, not all the PEs of PEZY-SC2 are used. The performance of the matrix-vector multiplication on PEZY-SC2 becomes about 60% of those on PEZY-SC1. This is explained with the difference of the device memory bandwidth displayed in Table 1 and that the PEs are not saturated by the launched threads. Accordingly the values of elapsed time for the linear solver, the Arnoldi method, and the matrix data conversion increase several tens of percent from those on PEZY-SC1.

4.3 Inversion of Dense Block Matrices

The block diagonal matrix M_D^{-1} is repeatedly applied in the preconditioning and thus it needs to be efficient to determine preceding to the iterative solver. To determine $M_D^{-1} = \{B_i^{-1} | i = 1, \ldots, n\}$ from $M_D = \{B_i | i = 1, \ldots, n\}$, the original code adopts the LU decomposition by calling the DGETRF and DGETRS routines in the LAPACK library. As shown in Table 2, this computation rapidly becomes costly as the matrix size increases, and thus worth to offload. It is however nontrivial to parallelize the LU decomposition efficiently on many-core processors. We instead employ an alternative algorithm, the blocked Gauss-Jordan elimination method [22]. The algorithm is represented as follows.

$$(B||A) = \left(\begin{array}{c||c|c} B_{00} & A_{01} & A_{02} \\ \hline\hline B_{10} & A_{11} & A_{12} \\ \hline\hline B_{20} & A_{21} & A_{22} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} B_{00} & 0 & A_{02} \\ \hline\hline 0 & 0 & 0 \\ \hline\hline B_{20} & 0 & A_{22} \end{array}\right) + \left(\begin{array}{c} -A_{01}A_{11}^{-1} \\ \hline\hline A_{11}^{-1} \\ \hline\hline -A_{21}A_{11}^{-1} \end{array}\right) (B_{10}|I||A_{12}), \quad (5)$$

where B_{ij} and A_{ij} are block matrices. Setting the size of A_{11} to be $b \times b$, each application of Eq. (5) extends the size of B part by b columns and rows. Starting from the original matrix A, *i.e.* $B = 0$, repeated application of Eq. (5) finally arrives at $B = A^{-1}$. A_{11}^{-1} can be determined by unblocked Gauss-Jordan method.

While the Gauss-Jordan elimination is nothing but a rearrangement of the LU decomposition, its blocked form can be parallelized to threads in units of the block size b. Only A_{11}^{-1} must be solved on a single thread. Thus the value of b is a tunable parameter. In this work, we set $b = 8$ considering the number of threads per PE. On each PE, A_{11} is loaded from the device memory to the local memory, and one thread solve A_{11}^{-1} with the unblocked Gauss-Jordan algorithm. After obtaining A_{11}^{-1}, remaining computation in each step of Eq. (5) can

be performed in parallel. We assign them so that each City (16 PEs) inverts one block matrix B_i. As shown in Table 2, adopting the Gauss-Jordan algorithm and offloading to the PEZY-SC processor achieve sufficient acceleration of computational time. However, as the matrix size increases, elapsed time of this process rapidly increases. For $(N_{E_\nu}, N_{\rm ang})$ larger than (24, 12), rearrangement of the number of devices and the number of PEs to solve each block matrix is desirable. Comparing these results to those for NVIDIA Tesla P100 GPU [4], the elapsed time of the matrix inversion is just about 40% larger than the latter. This implies the optimization on PEZY-SC works better than on P100, and is worth to feed back to the latter.

Result for PEZY-SC2 Processor. In the same way as for the matrix-vector multiplication, in our setup the launched threads do not saturate all the PEs of PEZY-SC2. Presumably this explains that the measured values of elapsed time for the blocked Gauss-Jordan algorithm on PEZY-SC2 are almost the same as those on PEZY-SC1.

4.4 Collision Term

Calculation of the collision term of the Boltzmann equation (1) is composed of several physical processes [9]. Table 3 summarizes the neutrino reaction processes included in this work. The first column is abbreviation of the physical process described in the second column. The third column shows the elapsed time measured on the host processors of Suiren Blue with four MPI processes. The last four processes, *esc*, *pap*, *plp*, *nbr*, are dominant, and thus the target of offloading. Indeed on GPUs sufficient acceleration has been achieved [4].

At the radial point r, the reaction rate of neutrino species ν is represented as $R^\nu_{\rm int}(E'_\nu, \theta'_\nu, E_\nu, \theta_\nu, r)$, where $(\theta_\nu,\ E_\nu)$ and $(\theta'_\nu,\ E'_\nu)$ are neutrino momentum angle and energy before and after the interaction, respectively. The contributions to the coefficient matrix and the source vector of the linear equation are obtained by multiplying $R_{\rm int}$ with the kinematic factors, neutrino distribution function under the matter environment, and integrating it over θ'_ν and E'_ν. Thus the calculation is performed in two steps: (a) Determination of the reaction rate $R^\nu_{\rm int}$, and (b) Calculation of the contribution to the collision term. In each step, each thread undertakes a contribution to each $(r,\ E_\nu,\ \theta_\nu)$ point of $R^\nu_{\rm int}$ or the collision term. These tasks are flatly assigned to the threads of device using all the resources, in contrary to the matrix-vector operations. This is mainly for simplicity of porting, since the calculation of the reaction rate is much more involved than the latter.

Table 3 shows the elapsed time on PEZY-SC1 and PEZY-SC2 for the parameter set $(N_r,\ N_{E_\nu},\ N_{\rm ang}) = (256, 14, 6)$. The data calculated on the devices are transferred to the host and changed to the data layout on the host, at the cost of the time quoted as 'data conversion' in Table 3. The result of Table 3 indicates that in spite of substantial acceleration in the *esc* process, there is almost no advantage in total by offloading these processes to PEZY-SC1. This is the case also for the larger matrix sizes as listed in Table 2.

Table 3. Elapsed time budget in the calculation of collision term at $(N_r, N_{E_\nu}, N_{\rm ang}) = (256, 14, 6)$, with four MPI processes on the host processor of Suiren Blue, PEZY-SC1, and PEZY-SC2.

Abbrev.	Physics process	Elapsed time [sec]		
		Host	PEZY-SC1	PEZY-SC2
ecp	Absorption/emission of ν_e on nucleons	0.06		
aecp	Absorption/emission of $\bar{\nu}_e$ on nucleons	0.06		
nsc	Neutrino scattering on nucleons	0.10		
csc	Coherent scattering of neutrino on nuclei	0.02		
ecpa	Absorption/emission of ν_e on nuclei	0.06		
esc	Scattering with electrons	0.94	0.37	0.04
pap	Neutrino pair creation/annihilation	0.25	0.42	0.04
plp	Plasmon process	0.28	0.42	0.04
nbr	Bremsstrahrung	0.22	0.42	0.04
	Data conversion	–	0.21	0.25

On the other hand, for the PEZY-SC2 processors, computation is much more accelerated than PEZY-SC1. On PEZY-SC2, the memory access performance is improved for non-coalesced access compared to PEZY-SC1. This explains the results displayed in Table 3 and implies the potential improvement on PEZY-SC1 by rearranging the data layout and task assignment. Thus considering the data conversion overhead, there is a substantial gain in offloading to the PEZY-SC2 processors. For larger two sizes, we observe the acceleration slightly less than a factor of two.

5 Results of 2D Simulation

5.1 Implementation and Numerical Setup

Our base code of the multi-dimensional Boltzmann solver is described in Fortran and parallelized with MPI. Applying the same strategy as the 1D code, we first convert the bottleneck parts of the code into the C language and then modify them to the kernels working on the PEZY-SC processors. The codes to setup the device environment, transfer the data, and launch the kernels in the 1D simulation are almost directly reused. Although the code is applicable to all the one, two, and three dimensional spaces, here we examine the performance of the code in two-dimensional space as a prototype of practical application.

As a benchmark setup, we adopt the numbers of grids in spatial coordinates, $N_r = 256$ and $N_\theta = 128$ (and $N_\phi = 1$). For the numbers of neutrino's momentum grids, $(N_{\theta_\nu}, N_{\phi_\nu}, N_{E_\nu})$, we consider two sets (6, 12, 14) and (8, 16, 16) as values practically feasible. We measure the performance of the code with four MPI processes each handles one PEZY-SC device. As noted in Subsect. 2.3, we employ an iterative solver without preconditioning. Thus the inversion of the dense block matrices is not the subject of present consideration. For the collision term, we offloaded the processes *ecp*, *nsc*, *csc*, *ecpa*, and *pap* in Table 3. However, the

observed elapsed time for these processes on the Suiren Blue system exhibits no advantage of offloading. The situation is similar to the 1D case, and we do not discuss further the offloading of the collision term in multi-dimensional code, considering the effort of implementation not deserving the gain. Thus we focus on the matrix-vector multiplication in the iterative linear equation solver.

5.2 Matrix-Vector Multiplication

The structure of the matrix M of the form Eq. (2) implies that the multiplication of block diagonal part, $M_D = \{B_i | i = 1, \ldots, n\}$ where $n = N_r \times N_\theta \times N_\phi$, to a vector is the dominant arithmetic operations, since B_i are dense matrices while the block matrices A_i and C_i are diagonal in the present case. Thus the byte-per-flop of $y_i = B_i x_i$, about 4 for the double precision arithmetics, is the representative of the matrix M multiplied to a vector x. Considering the device memory bandwidth of the PEZY-SC processor, about 25 GFlops is a guideline of the performance limit.

We apply the same prescription described in Subsect. 4.2 except for padding. $N_{\mathrm{unroll}} = 8$ provides the best performance for the present case. The sustained performance of the matrix-vector multiplication on the PEZY-SC1 processors is 14.2 GFlops and 13.4 GFlops for $(N_{\theta_\nu}, N_{\phi_\nu}, N_{E_\nu}) = (6, 12, 14)$ and $(8, 16, 16)$, respectively. These values of performance are slightly less than the corresponding results of one-dimensional setup displayed in Fig. 1, presumably due to the structure of present matrix. The offloading substantially improve the elapsed time of the linear equation solver. For example, for $(N_{\theta_\nu}, N_{\phi_\nu}, N_{E_\nu}) = (6, 12, 14)$, the time for 22 BiCGStab iterations is improved from 25.2 s to 7.0 s (iteration) + 1.5 s (data conversion). The more the number of iteration is required, the more acceleration is acquired.

For the PEZY-SC2 processors, we execute the same code as on the PEZY-SC1 changing only the total number of PEs from 1024 to 1984, since the number of spatial grids is enough larger than the number of Cities. We obtain the sustained performance 14.4 and 15.4 GFlops for $(N_{\theta_\nu}, N_{\phi_\nu}, N_{E_\nu}) = (6, 12, 14)$ and $(8, 16, 16)$, respectively. The better values of performance than those on PEZY-SC1 may imply that the memory access pattern of the present matrix is more suitable to the PEZY-SC2 architecture. We realize that our optimization prescriptions effectively apply also to the linear equation solver in the multi-dimensional Boltzmann equation code.

6 Conclusion

In this work, we examined viability of the PEZY-SC processors applied to the Boltzmann equation solver in the core-collapse supernova simulations. For one-dimensional code, we offloaded three bottlenecks to the PEZY-SC devices. For the iterative linear equation solver and inversion of dense block matrices, we achieved sufficient acceleration by offloading, while for the former the sustained performance is restricted by the device memory bandwidth. For the computation of the collision term, however, we judged it is not practical to extend

application because the achieved acceleration seems not to deserve large effort of porting and optimizing the code of this part. The good performance exhibited by the PEZY-SC2 processor seems to indicate the architecture is developing in promising direction. For the multi-dimensional Boltzmann equation solver, we demonstrated that the linear equation solver is offloaded efficiently by applying the prescriptions learnt in developing the one-dimensional code.

In conclusion, the PEZY-SC processors are attractive many-core architecture which potentially accelerates some kinds of applications. There are commonly applicable prescriptions to improve the performance, as examined in Sect. 4. For extensive application, however, two issues are to be settled. One is the device memory bandwidth worse than recent GPUs. This would be much improved in the next generation of the PEZY-SC processor. The other is the porting efficiency, which is crucial for application to wide area of researches. Efficient libraries and frameworks such as the OpenACC [21] are strongly desired.

Acknowledgment. The authors are grateful to A. Imakura, H. Okawa, T. Takiwaki and S. Yamada for HPC researches on core-collapse supernovae, to T. Aoyama, K-I. Ishikawa, T. Ishikawa, N. Kurosawa, R. Sakamoto, Y. Yamaura for development of code on PEZY-SC processors and for maintaining the Suiren Blue and Suiren2 systems. This work was supported 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, Grant-in-Aid for the Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan (15K05093, 17H06357, 17H06365, 19K03837, 20H01905), HPCI Strategic Program of Japanese MEXT, research programs at K-computer of the RIKEN AICS and Post-K project (Project ID: hp180111, hp180179, hp180239, hp190100, hp190160, hp200102, hp200124), the Particle, Nuclear and Astro Physics Simulation Program (Nos. 2019-002, 2020-004) of Institute of Particle and Nuclear Studies, KEK.

References

1. Kotake, K., et al.: Core-collapse supernovae as supercomputing science: a status report toward 6D simulations with exact Boltzmann neutrino transport in full general relativity. Prog. Theor. Exp. Phys. **2012**, 01A301 (2012). https://doi.org/10.1093/ptep/pts009
2. Janka, H.-T., Melson, T., Summa, A.: Physics of core-collapse supernovae in three dimensions: a sneak preview. Ann. Rev. Nucl. Part. Sci. **66**, 341–375 (2016). https://doi.org/10.1146/annurev-nucl-102115-044747
3. Matsufuru, H., Sumiyoshi, K.: Simulation of supernova explosion accelerated on GPU: spherically symmetric neutrino-radiation hydrodynamics. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10962, pp. 440–455. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95168-3_30
4. Matsufuru, H., Sumiyoshi, K.: Accelerating numerical simulations of supernovae with GPUs. In: 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW) (2018). https://doi.org/10.1109/CANDARW.2018.00056
5. Yamada, S.: An Implicit Lagrangian code for spherically symmetric general relativistic hydrodynamics with an approximate Riemann solver. Astrophys. J. **475**, 720–739 (1997). https://doi.org/10.1086/303548

6. Yamada, S., Janka, H.-T., Suzuki, H.: Neutrino transport in type II supernovae: Boltzmann solver vs. Monte Carlo method. Astron. Astrophys. **344**, 1468–1470 (1999). arXiv:astro-ph/9809009
7. Sumiyoshi, K., Yamada, S.: Neutrino transfer in three dimensions for core-collapse supernovae I. static configurations. Astrophys. J. Suppl. **199**, 17 (2012). https://doi.org/10.1088/0067-0049/199/1/17
8. Shibata, M., Nagakura, H., Sekiguchi, Y., Yamada, S.: Conservative form of Boltzmann's equation in general relativity. Phys. Rev. D **89**, 084073 (2014). https://doi.org/10.1103/PhysRevD.89.084073
9. Bruenn, S.W.: Stellar core collapse: numerical model and infall epoch. Astrophys. J. Suppl. **58**, 771–841 (1985). https://doi.org/10.1086/191056
10. Sumiyoshi, K., Ebisuzaki, T.: Performance of parallel solution of a block-tridiagonal linear system on Fujitsu VPP500. Parallel Comput. **24**, 287–304 (1998). https://doi.org/10.1016/S0167-8191(98)00007-6
11. Imakura, A., Sakurai, T., Sumiyoshi, K., Matsufuru, H.: A parameter optimization technique for a weighted Jacobi-type preconditioner. JSIAM Lett. **4**, 41–44 (2012). https://doi.org/10.14495/jsiaml.4.41
12. Nagakura, H., et al.: Simulations of core-collapse supernovae in spatial axisymmetry with full Boltzmann neutrino transport. Astrophys. J. **854**, 136 (2018). https://doi.org/10.3847/1538-4357/aaac29
13. Dannert, T., Marek, A., Rampp, M.: Porting large HPC applications to GPU clusters: the codes GENE and VERTEX. Adv. Parallel Comput. **25**, 305–314 (2014). https://doi.org/10.3233/978-1-61499-381-0-305
14. The Green500 site. https://www.top500.org/green500/
15. Aoyama, T., et al.: First application of lattice QCD to pezy-SC processor. Procedia Comput. Sci. **80**, 1418–1427 (2016). https://doi.org/10.1016/j.procs.2016.05.457
16. Yoshifuji, N., Sakamoto, R., Nitadori, K., Makino, J.: Implementation and evaluation of data-compression algorithms for irregular-grid iterative methods on the PEZY-SC processor. In: 2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3), pp. 58–61 (2016)
17. Haribara, Y., Ishikawa, H., Utsunomiya, S., Aihara, K., Yamamoto, Y.: Performance evaluation of coherent Ising machines against classical neural networks. Quantum Sci. Technol. **2**, 044002 (2017). https://doi.org/10.1088/2058-9565/aa8190
18. Hishinuma, T., Nakata, M.: pzqd: PEZY-SC2 acceleration of double-double precision arithmetic library for high-precision BLAS. In: Okada, H., Atluri, S.N. (eds.) ICCES 2019. MMS, vol. 75, pp. 717–736. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-27053-7_61
19. Yamazaki, T., Igarashi, J., Makino, J., Ebisuzaki, T.: Real-time simulation of a cat-scale artificial cerebellum on PEZY-SC processors. Int. J. High Perform. Comput. Appl. **33**, 155–168 (2017). https://doi.org/10.1177/1094342017710705
20. Sasaki, T., Hosono, N.: Particle number dependence of the n-body simulations of moon formation. Astrophys. J. **856**, 175(14pp) (2018). https://doi.org/10.3847/1538-4357/aab369
21. Tabuchi, A., et al.: Design and preliminary evaluation of omni OpenACC compiler for massive MIMD processor PEZY-SC. In: Maruyama, N., de Supinski, B.R., Wahib, M. (eds.) IWOMP 2016. LNCS, vol. 9903, pp. 293–305. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45550-1_21
22. Quintana, E.S., Quintana, G., Sun, X., van de Geijn, R.: A note on parallel matrix inversion. SIAM J. Sci. Comput. **22**, 1762–1771 (2001). https://doi.org/10.1137/S1064827598345679